Universitat Autònoma de Barcelona

Deep Learning and Bayesian Techniques applied to Big Data in Industry and Neutrino Oscillations

Sebastian Pina Otey

Deep Learning and Bayesian Techniques applied to Big Data in Industry and Neutrino Oscillations

PhD in Physics dissertation by

Sebastian Pina Otey

Institut de Física d'Altes Energies (IFAE)

Aplicaciones en Informática Avanzada, S.L. (Grupo AIA)

Supervisors:
Thorsten Lux
and
Vicens Gaitan Alcalde

Tutor:
María Pilar Casado Lechuga

Departament de Física

Universitat Autònoma de Barcelona

Barcelona, Spain; November, 2020

Generalitat de Catalunya
Departament d'Empresa i Coneixement
Secretaria d'Universitats i Recerca

Con el apoyo de la Secretaría de Universidades e Investigación del Departamento de Empresa y Conocimiento de la Generalidad de Cataluña.

GRUPO **AIA**

Universitat Autònoma de Barcelona

Institut de Física d'Altes Energies

ABSTRACT

Neutrino oscillations are a complex phenomenon of theoretical and experimental interest in fundamental physics, studied through diverse experiments, such as the T2K Collaboration situated in Japan. T2K is composed of two facilities, which produce and measure neutrino interactions to get a better understanding of their oscillations through data analysis in the form of parameter inference, model simulation and detector response. Through this work, state-of-the-art deep learning techniques in the form of neural density estimators and graph neural networks will be applied and thoroughly verified in T2K use cases, assessing their benefits and shortcomings compared to traditional methods. Additionally an industrial usage of these methodologies for the Spanish electrical network will be discussed.

AGRADECIMIENTOS [ACKNOWLEDGEMENTS]

Los tres años del doctorado han sido duros por diferentes motivos, con muchos altibajos llegando a desesperar tanto en la parte de investigación como en la versante burocrática. Por ello, quería agradecer a las personas que formaron directa e indirectamente parte de esta vivencia en estas líneas.

Los primeros que me han apoyado desde antes de que tuviera memoria son mi madre y mi padre. ¡Gracias por darlo todo para que pueda llegar a ser quien soy, y para que pueda seguir creciendo como quiero, no podría tener más suerte con mis progenitores! Tampoco puede faltar mi hermano aquí, que desde pequeño me intentó impregnar con su pasión por la historia, literatura y filosofía, que algo se ha pegado después de mucho insistir.

En cuanto a mis directores, Federico, extraoficialmente en los dos últimos años, se embarcó en la aventura de aplicar técnicas de aprendizaje profundo a diferentes cuestiones planteadas desde la oscilación de neutrinos en el experimento T2K. Aprecio especialmente las discusiones sobre la rigurosidad en los diferentes trabajos que hemos realizado y las implicaciones físicas de los mismos. Por otro lado, Vicens me ha enseñado a extraer mucho más de los datos de lo que a primera vista parece obvio, buscando siempre metodologías nuevas para responder a preguntas antiguas, teniendo presente la naturaleza de los datos mismos. Ojalá domine algún día la algoritmia de ciencia de datos como haces tú. Thorsten aceptó mi dirección tras la marcha de Federico, buscando nuevos proyectos como la colaboración con el equipo del CERN, que he disfrutado mucho, además de estar (casi) siempre disponible para mis constantes preguntas burocráticas y dotarme de los recursos que necesitaba en cada momento. ¡Gracias a los tres!

Mi día a día no hubiese sido lo mismo estos cuatro últimos años sin Violeta, que ha compartido mis sufrimientos y mis victorias, sacando el más brillante lado de las cosas. Un doctorado a mi parecer es muy duro, y sin su afecto y presencia no sé si lo hubiera sobrellevado hasta su finalización. ¡Esta culminación es claramente de los dos! Me alegro de que haya podido dotar la cubierta de la tesis con sus encantadores ilustraciones para darle el toque cálido y afectuoso a este frío y meticuloso trabajo (además del enorme esfuerzo de leerse la tesis y darme un feedback muy necesario). Gracias también a Carmen y a Fernando, que han estado ahí para cualquier cosa.

Por estar haciendo un doctorado industrial, he estado semi-conviviendo en varios sitios, por lo que mi relación con los compañeros no ha sido amplia en número, pero puedo decir que me he sentido muy a gusto en todos lados. Del IFAE, quiero dar gracias por estos tres años a mis compañeros del grupo de neutrinos, César y Danaisis, que han aguantado mis discusiones de temas algo alejados de sus propias investigaciones (y que César ha proporcionados buenas simulaciones a la carta). En AIA me he sentido personalmente muy cómodo, y agradezco el trato recibido por todo el personal. En especial quiero agradecer a Carlos, por embarcarse en este proyecto de doctorado industrial confiando en mí como primer candidato y dotándome de todos los recursos que me fueran necesarios; a Rosa, la super gestora de todos los papeleos habidos y por haber; a Yovana por estar siempre en recepción cogiendo mis paquetes; a Dani, Silvia, José Antonio y Marcos, por las incontables comidas que hemos pasado juntos hablando de nuestras vidas; a José Vicente y Josep Maria, por ser el mejor soporte informático que podía haber recibido. Como antiguo miembro de AIA, quiero agradecer a Toni como mi compañero de mesa con el que más conecté, discutiendo siempre de temas laborales y no laborales, mostrándole cómo iba avanzando mi investigación. El doctorado empezó como nexo también al SEA bajo la dirección de Anna Espinal, que tristemente nos dejó en 2019, y a la que agradezco las discusiones que tuvimos. Por otro lado quiero dar gracias a la compañía de todo el equipo del SEA, Llorenç, Anabel, Oliver, Esther y Ana, que me dieron un despacho y me acogieron en su día a día. De los alumnos que pasaron por el SEA formé especialmente amistades con Joan y Sandra, que hicieron muy amenas las comidas y que tuve la suerte de conocer.

Regarding my international experience, I would like to thank Asher for inviting me to stay at his group at Royal Holloway at the beginning of the PhD, showing me the potential applications of what I was going to do research on. At CERN, I formed a group together with Saúl, Dana, Davide and Leigh (also with César and Thorsten from IFAE) to do some nice research together which I thoroughly enjoyed. Special thanks to Saúl, who showed me around the beautiful area, embarked on climbing Crêt de la Neige with me among other things and who I count as a personal friend. Also in Geneva, I would like to thank Tobias for the interest shown in my work to invite me to give a presentation in his group.

Antes del doctorado he tenido muchos mentores, y a los siguientes quiero agradecer en especial: a Alfredo, por mostrarme el divertido mundo de las matemáticas y la lógica desde pequeño; a Jordi, por empujarme hacia el mundo de la investigación y darme tantas oportunidades, como ir a una escuela en Okinawa; a Josep María y a Lluís, por estar allí para mis dudas y discusiones numéricas; y a Carlos, al que admiro como investigador y como persona, que me ha enseñado el camino de la academia.

Por último, pero no menos importante, quiero agradecer a mis amigas y amigos que no han estado directamente involucrados en el doctorado, pero cuya amistad tengo siempre presente: a Oscar, por ser el guy in the chair; a Carles, por su sincera transparencia; a Patri, por nuestra excursiones; to Matt, for... you know why; a Pasqu, por ser el profe más enrollado; a Carlos, por su sabiduría de perros; a Bellido, por estar ahí; a Dani T., no preguntes porque; a Roger T., por las largas charlas; a Neus, por ser una Okaa-san a distancia; a Dani S., por ser un frikazo en las comidas; a Roger G., por las partidas del Smash; a Joan, por su energía extrema; a Juanca, por perdonarme el hígado; a Eloi, por el volley que nunca fue; a Ronna, por un futuro reencuentro; a Sergio, por la gran convivencia en Madrid; a Alina, por escaparnos de Torrent; a Pablo, por una amistad de toda la vida; a Lluis, por las partidas del pin-pon.

Y gracias a todas las demás personas que se han cruzado en mi camino y que, con mi mala memoria, me haya dejado por nombrar. ¡No os lo toméis mal!

Un gran abrazo,

Sebastian

ABBREVIATIONS

This section provides a concise reference describing the relevant abbreviations used recursively throughout this thesis.

AI	Artificial intelligence
AIA	Aplicaciones en Informática Avanzada
BN	Batch normalization
CCQE	Charged current quasielastic
CNN	Convolutional neural network
CP	Charge Parity
ENIS	Exhaustive neural importance sampling
FGD	Fine-Grained Detector
GNN	Graph neural network
GPU	Graphic Process Units
HEP	High Energy Physics
HPD	Highest posterior density
IFAE	Institut de Física d'Altes Energies
KL-divergence	Kullback-Leibler divergence
MAE	Mean absolute error
MAPE	Mean absolute percentage error
MC	Monte Carlo
MCMC	Markov chain Monte Carlo
ML	Machine learning
NIS	Neural importance sampling
NN	Neural network
NSF	Neural spline flow
PDF	Probability density function
PMNS	Pontecorvo-Maki-Nakagawa-Sakata

PMU	Phasor measurement unit
REE	Red Eléctrica Española
ReLU	Rectified Linear Unit
RMSE	Root mean squared error
SGD	Stochastic gradient descent
SK	Super-Kamiokande
SM	Standard Model
T2K	Tokai-to-Kamioka

CONTENTS

1. OVERVIEW

In this introductory chapter to the thesis, I will lay out the motivation behind the work done over the past years through my research, explaining why an industrial PhD between two entities, *IFAE (Institut de Física d'Altes Energies)*[1] and *Grupo AIA: Aplicaciones en Informática Avanzada*[2], was formed. The structure of the contents of the thesis will be detailed in Sec. 1.2, where I make emphasis on the foundations my work is built on as well as pointing out the original contributions made by me and my collaborators through different publications.

1.1 Motivation

Current neutrino experiments such as the T2K experiment [1] and future ones such as Hyper-K [2] and the Deep Underground Neutrino Experiment (DUNE) [3], and other experiments in high energy physics, e.g., the Large Hadron Collider (LHC) [4] and its upgrade, the high luminosity LHC [5], have as an objective to expand and verify the understanding of fundamental physics. For this purpose, an enormous amount of data is generated and processed both in real time and offline, to extract from the measurement of the detectors the physics contained in it.

The proper analysis of this data is vital to improve the precision of the Standard Model [6] and discover new physics beyond the Standard Model while answering open questions in the field: the neutrino mass [7], the strong Charge-Parity problem [8], the matter-antimatter asymmetry [9], the nature of dark matter [10] and dark energy [11], etc. In order to not limit the reach

[1] `http://www.ifae.es/`
[2] `https://aia.es/`

of the experiments' physics by the algorithms and computational resources, machine learning has been proposed and shown to produce promising results compared to classical analysis tools [12].

The neutrino group at IFAE, currently lead by Thorsten Lux (previously also co-led by Federico Sánchez until 2018), has been actively involved in the T2K experiment, contributing both from a theoretical point of view of model building [13, 14] and an experimental point of view of designing new detector elements [15], generally speaking. When dealing with the analysis of the data, the T2K experiment has so far relied on classical methods, ranging from Markov chain Monte Carlo for parameter inference, to standard energy cuts for event reconstruction. With the fast evolution that data science is taking through the form of machine learning, in particular through deep learning, new opportunities open to revolutionize the way data analysis is performed at T2K.

The other entity involved in the industrial PhD, Grupo AIA, is a consultancy firm founded in 1988 under the motto "Algorithms for a better world". The company focuses on transferring concepts and methodologies from basic science like maths and physics to the business world, enabling the application of new technologies to solve problems in an innovative way. Over the more than 30 years of existence, Grupo AIA has developed strategies involving advanced analytics and artificial intelligence, approaching challenges from different enterprises: energy, retail, textile, telecommunication, media, banking and insurance sectors, among others. Vicens Gaitan has been part of the data science department since graduating as a PhD in physics at IFAE in 1993, leading the group currently as chief data scientist. During his thesis, he already began exploring the application of neural networks to the world of high energy physics [16].

Thanks to the good relationship between IFAE and Grupo AIA through the years, as many graduates are and have been part of the company after leaving academia, a joint effort of forming and performing research through an industrial PhD was conceived. In particular, Federico Sánchez offered the T2K neutrino oscillation parameter inference analysis as an interesting starting point to begin exploring synergies with the knowledge that Grupo AIA, and in particular Vicens Gaitan, could bring to the table, specifically through machine learning.

Neutrino physics experiments such as T2K, as all particle physics experiments, encompass mainly three big branches when dealing with data analysis:

- Detector response: The data measured by the different detectors has to be processed to reconstruct the physics which is observed through the raw signals, transforming it into meaningful and interpretable physical magnitudes.
- Model simulation: Given theoretical models describing the physics at hand, both from a fundamental level and from a detector response level, it is essential to be able to generate data which follows these models in order to perform a proper analysis.
- Parameter inference: Arguably the last part of the analysis which strongly depends on the previous two branches, parameter inference is essential to constrain different magnitudes given a model and some observed quantities, finding the relation that leads to determining their values.

All of these processes are vital to explore and assess new physics.

In recent years, machine learning algorithms have overtaken and outperformed classical statistical approaches for the above tasks in many fields of application. Particle physics, however, requires an understanding and precision of the obtained results which might be unmatched in other areas. The main objective behind this thesis is to prototype and make a proof of concept of machine learning applications to these processes, comparing them to traditional methods while explaining the benefits and disadvantages of the new methods. Fortunately, as will be laid out in Sec. 1.2, I have been able to touch on each of the three main branches through three different works in my thesis.

Additionally, as part of Grupo AIA and the industrial PhD, some of the tools developed through the research phase also have found business applications.

1.2 Structure of the thesis

In order to make this thesis as self-contained as possible, as well as to account for the majority of readers with a more physics heavy background rather than a machine learning one, the thesis has been structured into two major blocks. The first block contains a series of chapters dedicated to introduce the physics and machine learning knowledge needed in order to properly follow and be able to trace back the following chapters. The second block consists in chapters presenting the (pre-)publications realized during my PhD. In what follows, I will describe shortly each of the chapters of the two blocks.

As already mentioned, the first block will serve as the background needed to have a good understanding of the different models and algorithms utilized through the PhD. The chapters forming part of this block are the following:

Chapter 2. Neutrino physics and the T2K experiment. A brief historical introduction to neutrino physics is presented, leading to the main property to be studied in this work: the neutrino oscillations. With the definition of the oscillation model formulated, the T2K experiment in Japan is laid out, describing its main two components: the near detector, ND280, and the far detector, the Super Kamiokande.

Chapter 3. Introduction to neural networks. Basic concepts of machine learning are introduced in order to familiarize the reader with the thought process when applying this kind of algorithm. Afterwards, the attention is focused on constructing step-by-step a basic neural network, explaining the fitting process utilizing backpropagation, as well as many essential parts such as optimizers, activation functions and regularization techniques.

Chapter 4. Normalizing flows. The majority of models follow a probability density function, which can be modeled through a normalizing flow. Normalizing flows are neural networks which construct approximations of density functions by defining a transformation from an unknown target density to an easy-to-evaluate base density. This allows to sample and evaluate the target density once the model is fitted. Neural spline flows are a particular, powerful implementation of the normalizing flows in the form of rational quadratic splines. To efficiently evaluate these flows, the notion of masked autoregressive networks is presented, parallelizing heavy computations.

Chapter 5. Graph neural networks. Data can be structured in many ways, being a mathematical graph one of the most flexible and rich one by explicitly linking connected samples without the Euclidean restriction. Graph neural networks have been growing in the past years, which have the property of aggregating and embedding the information of a node and its neighbors in an intelligent way. Through this work in particular, we are interested in GraphSAGE, one of the first graph neural networks which enabled the usage of the trained the trained model on new graphs with different number of nodes and adjacency matrices.

The second block focuses on the contributions made during the PhD. This includes three articles and an industrial project. For the (pre-)publications, the chapters are almost entirely the content of said (pre-)publications, while removing certain sections which are well expanded in the first block and are instead changed to references to these chapters. The original contributions of the thesis are:

Chapter 6. Likelihood-free inference of experimental neutrino oscillations using neural spline flows [17], by Sebastian Pina-Otey, Federico Sánchez, Vicens Gaitan and Thorsten Lux, published in Physical Review D 101, 113001, on the 2$^{\text{nd}}$ of June of 2020. The application of neural spline flows to perform likelihood-free inference on a simplified two-flavour neutrino oscillation parameter model for the T2K experiment is presented. In particular, the usage is exemplified through the case of the disappearance muon neutrino, assessing the advantages gained by utilizing this novel approach, comparing it to the traditional histogram-based inference. **My original contributions in this work are the modification and fusion of neural density estimators in the form of neural spline flows with the theoretical two-flavour oscillation model to perform likelihood-free inference, and the verification of its integrity by comparing it to traditional methods.**

Chapter 7. Exhaustive neural importance sampling applied to Monte Carlo event generation [18], by Sebastian Pina-Otey, Federico Sánchez, Thorsten Lux and Vicens Gaitan, published in Physical Review D 102, 013003, on the 16$^{\text{th}}$ of July of 2020. Exhaustive neural importance sampling, a method based on normalizing flows to find a suitable proposal density for rejection sampling automatically and efficiently, is proposed to solve the issue of slow and impractical Monte Carlo generators. In particular, the approach is tested on generating accurately a neutrino-nucleus cross section model, needed for neutrino oscillation experiments. **My original contributions are the modification of neural importance sampling with a background density, defining a new algorithm, and the comparison of utilizing the new obtained proposal function with generic uniform densities.**

Chapter 8. Graph neural network for 3D classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors [19], by Saúl Alonso-Monsalve, Dana Douqa, César Jesús-Valls, Thorsten Lux, Sebastian Pina-Otey, Federico Sánchez, Davide Sgalaberna and Leigh H. Whitehead, submitted to Physical Review D on the 1$^{\text{st}}$ of September 2020 (preprint arXiv:2009.00688). The reconstruction of neutrino interactions in future 3D-granular plastic-scintillator detectors will be an essential task in the near detector upgrade of the T2K experiment. When reconstructing the three-dimensional particle tracks produced in the detector, ambiguities due to high multiplicity signatures in the detector or leakage of signal between neighboring arise. In this work, we present a graph neural network, inspired by GraphSAGE, to classify the reconstructed 3D-signatures, boosting both efficiencies and purities on the simulated events, while verifying the robustness against systematic effects which may appear. **My original contributions in this work are the development and adaptation of a GraphSAGE-like algorithm to this particular dataset, and the verification against systematic uncertainties.**

Chapter 9. Predictive analysis of the damping rate in inter-area oscillations. As part of an industrial project within Grupo AIA, the damping ratio of the power grid is analyzed and modelled with respect to different magnitudes of the Spanish electric network in a qualitative way. In particular, the damping ratio signal properties with respect to time and space correlations are studied, a predictive model is constructed, the graph structure of the electric network is exploited through graph neural networks and an accurate uncertainty assessment is performed through quantile regressions and normalizing flows. **My original contributions are the study of the signal's autocorrelations, the graph neural network experiments and the assessment of uncertainties done through the two models.**

Additionally, the Appendix includes additional information and graphics present in the (pre-)publications to complement the work shown.

2. NEUTRINO PHYSICS AND THE T2K EXPERIMENT

Neutrinos and their oscillation property are the fundamental objects of study through this thesis. This chapter is dedicated to give a general overview of the neutrinos within the particle physics field while also presenting the T2K experiment. We start by speaking shortly about the historical context of neutrinos within particle physics and how they fit inside the Standard Model of particles, Sec. 2.1. Afterwards, in Sec. 2.2, we focus on the neutrino oscillation phenomenon from a theoretical point of view, introducing the oscillation parameters, relevant for the analysis. We finalize the chapter in Sec. 2.3, by describing the T2K experiment and its set up for the near and far detector facilities, explaining how the components contribute to measure and detect neutrino oscillations.

2.1 Brief historical background and basic characteristics

The notion of neutrino particles, ν, was first postulated in 1930 by Pauli, at the time named "the neutron", a neutral particle to explain the missing energy distribution observed experimentally in the β-decay by Chadwick [20, 21]. In 1934, Fermi [22] further incorporated the neutrino into his own β-decay theory as an invisible four-momentum carrier in the $n \to p + e^- + \bar{\nu}_e$ process, which now is known as electron anti-neutrino, $\bar{\nu}_e$.

No experimental evidences of a neutrino particle were found for the next 26 years, until Reines and Cowan designed a set up in which inverse β-decays from the Savannah River nuclear reactor in South Carolina produced a measurable number of anti-neutrinos in 1956 [23, 24].

Up to that point in time, neutrinos appeared to be all the same type of particle. In 1959, Pontecorvo suggested to verify if indeed the neutrinos are produced in association with muons in a pion decay are the same as ones emitted with electrons in the β-decay [25]. After a few years, in 1962, an experiment at the Alternating Gradient Synchrotron at Brookhaven National Laboratory properly identified a distinct type of neutrino, different from the previously found one by generating μ leptons but no electrons in the process: the muon neutrino, ν_μ [26].

The last predicted neutrino, the tau neutrino, ν_τ, was elusive and only directly observed in 2000 by the Direct Observation of Nu Tau (DONuT) experiment at Fermilab [27]. With this, all three neutrino flavours were established experimentally.

The three neutrinos ν (plus their antiparticles $\bar{\nu}$, corresponding to particles with their same mass but opposite physical charges, such as the electric charge, lepton number, etc.) form part of the Standard Model of elementary particles, or simply Standard Model (SM), depicted in Fig. 2.1. The SM is a relativistic quantum field theory which describes the strong, weak and electromagnetic interactions between the elemental particles. Most of the experimental observations can be explained under this theory, including the previously mentioned neutrino experiments.

According to the SM, there exist three generations of fermions, each of them consisting of an up-type quark, a down-type quark, a charged lepton and a neutrino. As we advance through the generations, the structure is repeated, increasing the mass of the corresponding particles. All the fermions interact between them through the weak interaction (through either Z bosons or $W^\pm$ bosons). Additionally quarks and charged leptons interact through the electromagnetic interaction (with exchanging photons γ), and the quarks themselves can interact through the strong interaction (with exchanging gluons g).

The neutrinos, within the SM, only feel weak interactions and are assumed to be massless. (In Chapter 7 we will go through one particular example of this interaction, the charged current quasielastic (CCQE) interaction.) By being massless and travelling in vacuum, the neutrino generation (also referred to as flavour) should remain the same. This assumption has been proven wrong experimentally by the observations of neutrino oscillation, implying that neutrinos have indeed non-zero mass.

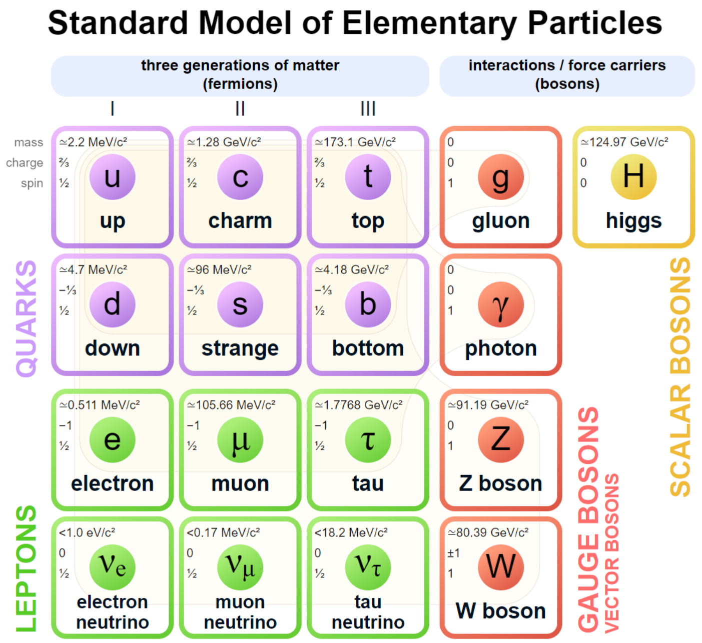

Standard Model of Elementary Particles
three generations of matter (fermions)
interactions / force carriers (bosons)
I
II
III
mass
charge
spin
≃2.2 MeV/c² ⅔ ½ u up
≃1.28 GeV/c² ⅔ ½ c charm
≃173.1 GeV/c² ⅔ ½ t top
0 0 1 g gluon
≃124.97 GeV/c² 0 0 H higgs
≃4.7 MeV/c² −⅓ ½ d down
≃96 MeV/c² −⅓ ½ s strange
≃4.18 GeV/c² −⅓ ½ b bottom
0 0 1 γ photon
≃0.511 MeV/c² −1 ½ e electron
≃105.66 MeV/c² −1 ½ μ muon
≃1.7768 GeV/c² −1 ½ τ tau
≃91.19 GeV/c² 0 1 Z Z boson
<1.0 eV/c² 0 ½ νe electron neutrino
<0.17 MeV/c² 0 ½ νμ muon neutrino
<18.2 MeV/c² 0 ½ ντ tau neutrino
≃80.39 GeV/c² ±1 1 W W boson
QUARKS
LEPTONS
GAUGE BOSONS
VECTOR BOSONS
SCALAR BOSONS

Fig. 2.1: The Standard Model of elementary particles. Source: [28].

2.2 Neutrino oscillations

The phenomenon of neutrino oscillations is a direct consequence of them having mass, and consists of neutrinos changing their flavour as they travel in space.

From a historical point of view, the concept of oscillations was first proposed by Pontecorvo in 1957 [29], before the discovery of the muon neutrino, where he suggested the possibility of neutrino-antineutrino oscillations. When the second generation of neutrinos were discovered, Maki, Nagakawa and Sakata considered the possibility of oscillation between the two types of neutrinos [30]. Pontecorvo, in 1967, predicted that Sun-emitted electron neutrinos would transition to muon neutrinos [31], even before the first measurement by the

Homestake experiment in 1968 showed deficit in the solar electron neutrino flux [32], known as the solar neutrino problem. A similar issue appeared in the second half of the 1980s, known as the atmospheric neutrino anomaly, where measurements of the atmospheric neutrino flux showed a deficit. These two experimental observations remained controversial for years, and were only accepted when Super-Kamiokande [33] and Sudbury Neutrino Observatory (SNO) [34, 35] verified the oscillation hypothesis for atmospheric neutrinos and solar neutrino, respectively.

From a formal point of view, as we will see in what follows, the neutrino oscillations are defined by a flavour-mass eigenstate mixing matrix described by 4 angles (including a charge-parity (CP) violating phase δ_{CP}) and the two mass differences Δm^2 between the three mass states.

In the standard theory of neutrino oscillations, oscillations can be constructed via a unitary mixing matrix ($U^\dagger U = 1$), known as the Pontecorvo-Maki-Nakagawa-Sakata (PMNS) matrix, analogous to the one used in the quark sector, which conserves the lepton number of the particle [36–38]. The neutrino flavours, or flavour eigenstates, ν_α, $\alpha = e, \mu, \tau$, are interpreted as a linear superposition of massive Dirac particles, or mass eigenstates, ν_k, $k = 1, 2, 3$, with corresponding masses m_k:

$$|\nu_\alpha\rangle = \sum_k U^*_{\alpha k} |\nu_k\rangle ,$$

where U is the unitary mixing matrix. In vacuum, the massive neutrinos with momentum $\vec{p}$ are eigenvectors of the Hamiltonian H:

$$H |\nu_k\rangle = E_k |\nu_k\rangle = \sqrt{|\vec{p}|^2 + m_k^2}\, |\nu_k\rangle ,$$

with eigenvalues $E_k = \sqrt{|\vec{p}|^2 + m_k^2}$, where we are assuming natural units [39], i.e., the speed of light $c = 1$, as will be done through the rest of the work.

Consider a neutrino flavour state $|\nu_\alpha(t)\rangle$, describing a neutrino created at time $t = 0$, with $|\nu_\alpha(t = 0)\rangle = |\nu_\alpha\rangle$, and time evolution in vacuum given by

$$|\nu_\alpha(t)\rangle = \sum_k U^*_{\alpha k} e^{-iE_k t} |\nu_k\rangle . \tag{2.1}$$

Because of the unitarity of the mixing matrix, the massive neutrino states can be written as

$$|\nu_k\rangle = \sum_\alpha U_{\alpha k} \, |\nu_\alpha\rangle \, .$$

The time evolution for a neutrino flavour state, Eq. (2.1), then becomes

$$|\nu_\alpha(t)\rangle = \sum_\beta \left(\sum_k U^*_{\alpha k} e^{-iE_k t} U_{\beta k} \right) |\nu_\beta\rangle \, .$$

Therefore, the initial pure neutrino flavour at time $t = 0$ becomes a superposition of different flavour states for $t > 0$, producing neutrino oscillations.

The time-dependent transition amplitude for a flavour conversion $\nu_\alpha \to \nu_\beta$ is given by

$$A(\nu_\alpha \to \nu_\beta) = \langle \nu_\beta | \, \nu_\alpha(t) \rangle = \sum_k U^*_{\alpha k} e^{-iE_k t} U_{\beta k}.$$

Hence, the transition probability is

$$P(\nu_\alpha \to \nu_\beta) = |A(\nu_\alpha \to \nu_\beta)|^2 = \sum_{k,j} U^*_{\alpha k} U_{\beta k} U_{\alpha j} U^*_{\beta j} e^{-i(E_k - E_j)t}. \quad (2.2)$$

When considering ultra-relativistic neutrinos, such as the one produced in the T2K experiment, where the momentum is orders of magnitude larger than the mass, $|\vec{p}| \gg m_k$, the energy eigenvalues can be approximated by

$$E_k = \sqrt{|\vec{p}|^2 + m_k^2} = |\vec{p}| \sqrt{1 + \frac{m_k^2}{|\vec{p}|^2}} \simeq |\vec{p}| + \frac{m_k^2}{2|\vec{p}|} = E + \frac{m_k^2}{2E},$$

where the neutrino energy is defined as $E = |\vec{p}|$, neglecting the mass contribution. In this regime, the difference $E_k - E_j$ becomes

$$E_k - E_j = \frac{m_k^2 - m_j^2}{2E} = \frac{\Delta m_{kj}^2}{2E}.$$

Therefore, the transition probability of Eq. (2.2) takes the form of

$$P(\nu_\alpha \to \nu_\beta) = \sum_{k,j} U^*_{\alpha k} U_{\beta k} U_{\alpha j} U^*_{\beta j} e^{-i\frac{\Delta m^2_{kj} t}{2E}} .$$

If ultra-relativistic neutrinos travel a fixed distance, L, as is the case for the T2K experiment, the above expression can be written as

$$P(\nu_\alpha \to \nu_\beta) = \sum_{k,j} U^*_{\alpha k} U_{\beta k} U_{\alpha j} U^*_{\beta j} e^{-i\frac{\Delta m^2_{kj} L}{2E}} , \tag{2.3}$$

showing that the phase responsible for the neutrino oscillation depends on the source-detector distance L, the neutrino energy E and the difference of squared-mass. Notice how only two difference of squared-mass are independent: given Δm^2_{21} and Δm^2_{31}, Δm^2_{32} is just a the difference: $\Delta m^2_{32} = \Delta m^2_{31} - \Delta m^2_{21}$.

The PMNS matrix for the three neutrino families can be decomposed as a product of three matrices:

$$U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & s_{13}e^{-i\delta_{\mathrm{CP}}} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta_{\mathrm{CP}}} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

where $c_{ij} = \cos\theta_{ij}$ and $s_{ij} = \sin\theta_{ij}$. The first matrix describes the oscillation of atmospheric neutrinos, while the third does so for the oscillation of solar neutrinos.

With this, the six free parameters of the mixing matrix have been introduced. To measure them, the different experiments compare the rate of neutrinos with certain flavour and energy E at the production point and after travelling a fixed distance L, obtaining different mixing combinations for different energies of $P(\nu_\alpha \to \nu_\beta)$ (see Fig. 2.2). These combinations are broken down into two channels: disappearance and appearance.

For the disappearance channel, the survival probability of a certain flavour is measured, e.g., for the ν_μ disappearance, the rate between the initially produced ν_μ neutrinos at the creation point and the measured one after travelling

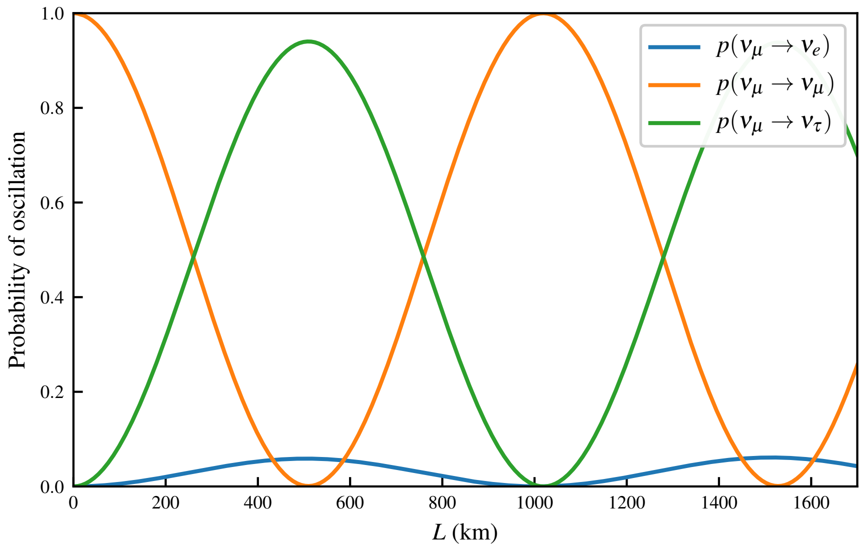


Fig. 2.2: Neutrino oscillation probabilities for a pure ν_μ produced with an energy of 1 GeV for different distances L.

a distance L, $P(\nu_\mu \to \nu_\mu)$. Contrary, for the appearance channel, the oscillation from one flavour to another is measured, e.g., for the appearance of electron neutrinos from muon neutrinos, the rate of ν_μ is measured at the creation point and is compared to the rate of ν_e measured after travelling a certain distance L, $P(\nu_\mu \to \nu_e)$.

The different neutrino experiments are designed by selecting a specific source for producing neutrinos with certain flavour and the energy distribution while fixing an appropriate distance L so that they are sensitive to a specific mixing channel, and therefore, to specific parameters of the PMNS matrix. The experiments can be broadly classified into one of four categories:

- Solar neutrino experiments: thermonuclear reactions in the Sun produce neutrinos which are measurable on Earth. E.g.: the Homestake solar neutrino observatory [40] and the GALLium EXperiment (GALLEX)

[41].

- Atmospheric neutrino experiments: Pions and kaons coming from primary cosmic rays interacting in the upper layer of the atmosphere decay into neutrinos. While decaying, the muons produced also further decay into additional neutrino sources. E.g.: the Irvine–Michigan–Brookhav experiment [42] and Super Kamiokande [33].

- Reactor neutrino experiments: Electron antineutrinos fluxes are produced in nuclear reactors via β-decay of certain isotopes. E.g.: the CHOOZ experiment [43] and KamLAND [44].

- Accelerator neutrino experiments: Neutrinos are produced via the decay of pions, kaons and muons generated by an accelerated proton hitting a target. E.g.: the NOvA experiment [45] and the T2K experiment (see next section).

2.3 The T2K experiment

The T2K (Tokai-to-Kamioka) experiment [1] is one of the leading accelerator neutrino experiments. In these experiments, neutrinos are produced in particle accelerators, where protons are accelerated to desired levels of energies in order to collide onto a nuclear target. When this happens, mesons are generated (primarily pions π and secondly kaons K and muons μ) which decay after travelling a certain distance into neutrinos and their associated leptons.

T2K is a long-baseline neutrino oscillation experiment, located in Japan, designed to analyze the mixing channels of muon neutrinos. It started its operation in March 2010, being the first long-baseline experiment to find electron neutrinos appearance from muon neutrinos $\nu_\mu \to \nu_e$ [46], measuring the last unknown mixing angle at the time, θ_{13}, among the other PMNS matrix parameters.

The setup of the experiment is depicted in Fig. 2.3. An accelerator at the J-PARC produces neutrinos, which are detected at the far detector Super-Kamiokande (SK) [47], 295 km away from the production target. The near detector facility has two detectors: the on-axis detector INGRID and the off-axis detector (2.5°) ND280, both situated 280 m from the production target.

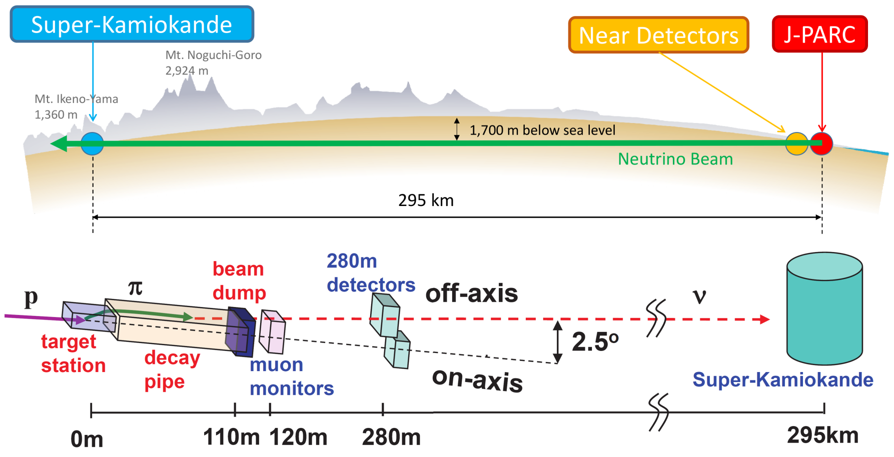


Fig. 2.3: Schematic representation of the T2K experiment. Top shows a geographical representation while bottom details the different components of the source, near detectors and the far detector.

The beam used for the experiment is the off-axis one, which produces a narrow-band neutrino beam in energy, with peak energy at 0.6 GeV. The angle can be reduced up to 2°, allowing one to modify the energy peak if necessary. Below, the different parts are briefly detailed.

The J-PARC facility [48] produces the (anti)neutrino beam. First, protons are accelerated via a linear accelerator (LINAC), a rapid cycling synchrotron (RCS) and a main ring (MR) synchrotron. The beamline consists of two parts (see Fig. 2.4): i) the primary beamline, which takes fast-extracted protons from the MR, bending them to point towards SK and colliding them onto a graphite target ii) the secondary beamline, which directs the mesons, mainly pions, from the proton-target interaction through a decay volume, finishing with a beam dump. In this secondary volume, the pions are collected by a first magnetic horn, while two consecutive horns focus them further in the proper direction. The polarity of the horns allow them to focus positive or negative hadrons to produce either a very pure muon neutrino beam or a very pure

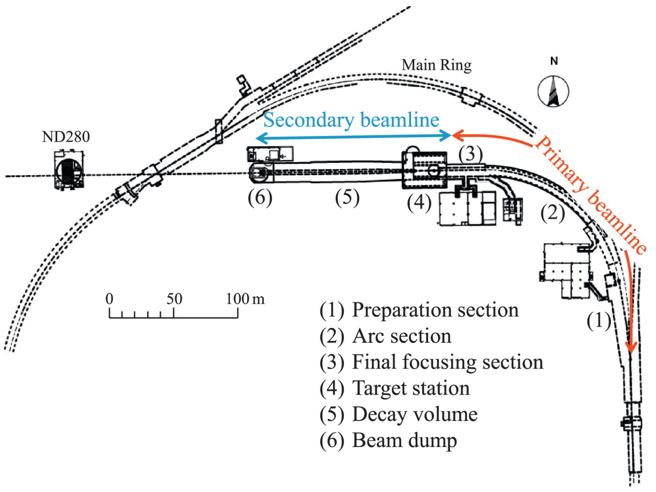
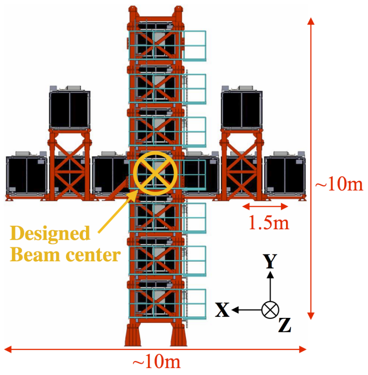


Fig. 2.4: Overview of the T2K neutrino beamline at the J-PARC facility.

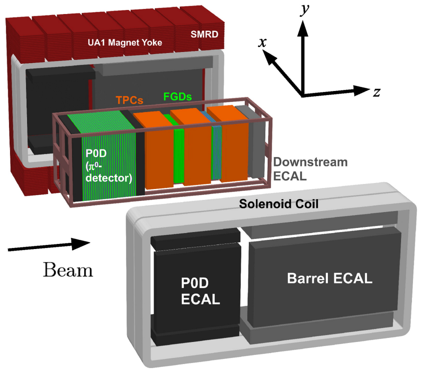
muon antineutrino beam by reversing the magnetic field direction. The beam of hadrons enters then a decay volume, to decay into neutrinos and leptons, mainly muons. This decay volume is optimized to enhance the ν_μ contribution while reducing the ν_e and $\bar{\nu}_\mu$ contamination. These decayed particles finally hit a graphite and iron beam dump, where only very high energetic muons (above 5 GeV) and neutrinos can pass.

The first near detector, the on-axis detector INGRID [49] (see Fig. 2.5 (a)), is composed of an array of iron/scintillator sandwiches, and measures the neutrino beam direction and profile. Determining the beam direction is crucial to set up the experiment at the desired off-axis of 2.5°, and INGRID achieves this by comparing the different number of events in each of its modules to a precision of 0.4 mrad. Additionally, INGRID helps to characterize the proper intensity of the muon neutrino beam, as high statistics are obtained daily on it.

The second near detector, the off-axis detector ND280 (see Fig. 2.5 (b)), has as a purpose to measure the flux, energy spectrum and composition of neutrinos in the direction of the far detector, while also to obtain the rates



(a)



(b)

Fig. 2.5: Schematic representation of the two near detectors: (a) the on-axis detector INGRID and (b) the off-axis detector ND280 with its different sub-detectors composition.

for exclusive neutrino reactions. It is composed of two main sections: the P0D and the TPC/FGD sandwich (trackers). The P0D (π^0 detector) [50], is designed to observe neutral current interactions on water, where π^0s are emitted. This process is one of the main backgrounds in the ν_e selection at Super-Kamiokande. The TPC/FGD is composed by a Time Projection Chamber (TPC) [51] and a Fine-Grained Detector (FGD) [52]. The FGD is composed of 15+7 modules, and in each of them two layers of 192 scintillator bars are oriented in the x and y directions alternatively, perpendicular to the beam direction. In the later 7, the modules are interleaved with thin layers of water (2.5 mm thick). This fine segmentation allows for precise vertex and charged particle reconstruction, while providing a target mass for the neutrino interactions. An upgrade for the ND280 is being developed at the moment of writing this thesis [53], including an upgrade for the FGD, called the Super FGD, which will be detailed in Chapter 8. In the TPC, charged particles that pass through it ionize a gas, producing a readout for precise tracking, as well as helping with the particle identification (PID) thanks to the characteristic amount of ionization.

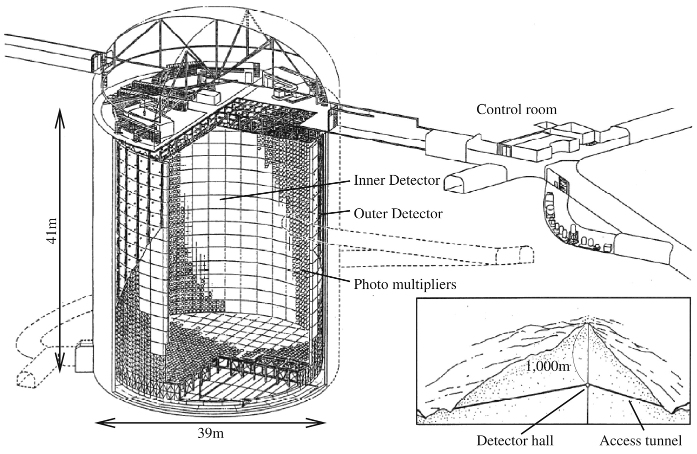


Fig. 2.6: Diagram of the Super-Kamiokande Detector.

Super-Kamiokande, Fig. 2.6, the far detector of the T2K experiment, is the world's largest land-based water Cherenkov detector, located 295 km west from the beam source, built 1 km within the center of Mt. Ikenoyama. Its construction finished in 1996, and consists of a cylindrical cavern filled with 50 kton of pure water together with approximately 13,000 photomultiplier tubes (PMTs). It measures the flux of the incoming beam, searching for both the disappearance $\nu_\mu \to \nu_\mu$ and the appearance $\nu_\mu \to \nu_e$ channels via the rates of ν_μ and ν_e CCQE interactions' produced leptons. These charged leptons produce a cone of Cherenkov light which is detected by the PMTs. The pattern of the light allows to extract the position of interaction and the momentum of the outgoing particle by classifying the Cherenkov rings observed as either muon-like or electron-like via comparison to an analytically calculated expected pattern in the case of muons and an Monte Carlo-generated expected pattern in the case of electrons [1], helping to reconstruct the neutrino energy with the outgoing lepton kinematics (assuming four-body interaction).

Since its first data-taking in January of 2010, the T2K experiment has accomplished important results in the field of neutrino oscillations, especially standing out:

- The discovery of the appearance oscillation channel $\nu_\mu \rightarrow \nu_e$ [46].
- One of the most precise measurements of the θ_{23} and Δm^2_{23} oscillation parameters [54].
- A 3-σ confidence interval for the constraint on the matter–antimatter symmetry-violating phase δ_{CP} in neutrino oscillations [55].

3. INTRODUCTION TO NEURAL NETWORKS

Data analysis has advanced at a rapid pace over the last 70 years, since the appearance of digital computations through computers [56]. One specific branch of data analysis, machine learning, was conceived thanks to this additional computational power, which exceeds any previous form of calculations while growing exponentially rapidly in its own capacity [57]. This has only been accentuated thanks to the fast evolution brought by the versatility of a specific machine learning algorithm, the neural networks [58], and the boost in processing power through graphic cards or graphic process units (GPUs), allowing for highly parallelizable operations to take place [59]. Another factor influencing the push through machine learning methodologies has been the abundant growth in data production [60] both from a business point of view, with super markets, delivery systems, banks, etc. collecting large information on their clients and transactions, as well as from a scientific and engineering point of view, with devices digitizing information and models generating new data through simulations.

This chapter serves as an introduction to machine learning in general as well as focusing specifically on neural networks. At the end of the chapter, we will have built a feed-forward fully-connected neural network, the simplest architecture which serves as a building block to construct more advanced ones, and is therefore fundamental to have a complete understanding of.

In Sec. 3.1 we present the concepts shared by most machine learning algorithms, as well as defining some important jargon of the field which will be used extensively through the thesis. In Sec. 3.2, the construction until reaching the feed-forward neural network will be performed from a historical point of view, showing the progress of the methodologies which originated the modern concept of neural networks. To finish the chapter, Sec. 3.3 introduces additional components which are part of a neural network, such as the functions which propagate through it and the initialization of the parameters, among others.

3.1 Concepts of Machine learning

Machine learning (ML) is the science of algorithms allowing the computer to learn automatically from experiences in the form of data. The roots of ML lie in the existing statistical methods before having the modern notion of it, and current theoretical research in ML is considered a subfield of modern statistics and artificial intelligence (AI).

The history of ML started in 1949, when Donald O. Hebb proposed a model for explaining the plasticity and adaptation of the brain [61], which led in 1951 to Marvin Minsky and Dean Edmonds to build the first neural network machine, the SNARC (Stochastic Neural Analog Reinforcement Calculator) [62]. A year later, Arthur Samuel, together with IBM, began working on the first programs that play checkers [63]. From there on, different ML algorithms were developed in the following decades, with the perceptron in 1957 [64], Nearest Neighbours Pattern recognition in 1967 [65], Random decision forests [66] and Support-vector machines [67] in 1995, among many others. From all the different types of ML algorithms, neural networks will be the focus of this thesis, and their history will be expanded in depth in Sec. 3.2.

The data is normally structured in an aggregation of instances $\mathbf{x}_i$, forming a dataset $\{\mathbf{x}_i\}_{i=1}^N$. These instances share a defined description of the reality they are representing, and can take many different mathematical shapes, the most common being tabular data. Tabular data is defined as a vector of values, which can be correlated or not, and where each component of the vector can have its own type of variable, such as float, integer, categorical, string, etc. For example, the variables involved in a cross section probability, such as the momentum p and angle θ of the outgoing particle, can form a tabular data, $\mathbf{x}_i = \{p_i, \theta_i\}$. Other types of typical data instances are images, where each instance is a $h \times w \times 3$ tensors, where h (w) are the pixel height (width) of the image, and each of these pixels have a 3-dimensional RGB color encoding. Time-series of different length can also form a dataset, such as the one produced by a seismograph or an electrocardiogram. Mathematical graphs composed by nodes and edges also form datasets, as will be shown in Chapter 5, where each node would correspond to an entry in a tabular dataset, and they would be connected via an adjacency matrix, giving additional information of connections between the entries of the tabular dataset. As illustrated with these examples, datasets can have many different shapes and compositions, but share this same data structure between the different instances of the dataset.

Once the dataset is defined, the next step is to describe the task to fulfill using ML techniques. For this purpose, ML approaches can be broadly classified into three categories (sometimes an algorithm combines two of them or directly does not fit into any):

- Supervised learning: the data $\{\mathbf{x}_i\}_{i=1}^N$ fed into the algorithm, known as the input or features, is paired with the expected solution or outcome $\{\mathbf{y}_i\}_{i=1}^N$, known as the output, target or label. The algorithm's task is to find a relation between the input and output in the form of a function f_θ, $f_\theta(\mathbf{x}) = \hat{\mathbf{y}}$, where θ are the parameters of the algorithm's function. $\hat{\mathbf{y}}$ is the prediction of the algorithm given an input $\mathbf{x}$, which for the training data should be as close as possible to the real value $\mathbf{y}$ while generalizing properly to equally distributed unseen data, tuned by the parameters θ (the optimization of the parameters will be discussed later). The target is usually a scalar value, and can be either discrete (e.g. the particle ID of a trace) or continuous (e.g., the energy of an event). When the target is discrete, the algorithm performs a classification task, while when the target is continuous, it does a regression.

- Unsupervised learning: Contrary to the previous set of algorithms, the data $\{\mathbf{x}_i\}_{i=1}^N$ is unlabeled. The objective of unsupervised algorithms is to understand and extract information of the structure of the data, which encompasses a large variety of tasks. A very common goal is to cluster the data into different groups sharing some structure beneath the data, which can then be further analyzed (e.g., k-means [68] is a clustering algorithm by partitioning the n vectors of data into k clusters). Sometimes the data needs to be transformed in order to process it more efficiently, visualize it in a lower dimension or simply reduce its dimension. Principal Component Analysis (PCA) [69] projects the data from a higher dimensional space into a lower dimensional space by maximizing the variance of each component via linear combinations of the original space, getting rid of linear correlations and defining a new orthogonal basis known as principal components. T-distributed Stochastic Neighbor Embedding (t-SNE) [70], compared to PCA, offers a non-linear embedding to visualize high dimensional data into a two or three dimensional space. Neural density estimators also fall into unsupervised learning, where the probability density function (PDF) is learned through the data. Normalizing flows, a subfamily of neural density estimators, will be discussed in depth in Chapter 4.

- Reinforcement learning: reinforcement algorithms work very differently than the previous ones: instead of having data from which one wishes to extract direct information, the learning system, called an agent, observes an environment, selects and performs actions and obtains a reward or penalty in return. The decisions this agent takes come from a strategy that it learns through the algorithms, known as policy, with the aim to maximize the reward obtained at each step until reaching a goal. A typical application of reinforcement learning is creating an AI to play a game, such as chess or Go: in these games, the observation would be the state of the game, the actions would be to move/place the game pieces and the maximum reward would be to win the game while the maximum penalty would be to loss it. The AlphaGo documentary[1] illustrates the complexity of designing a good reinforcement algorithm while also showing the potential it has.

For this work, the focus lies in supervised and unsupervised algorithms, hence we will not go any further in reinforcement learning. In particular, in what follows this section, we will be speaking strictly about supervised learning, since the unsupervised learning of normalizing flows in Chapter 4 will be approached in an analogous way.

As anticipated above, algorithms take the form of a parametrized function $f_\theta(\mathbf{x})$, with parameters θ and input the data $\mathbf{x}$. For supervised learning, the output of the function gives the estimate $f_\theta(\mathbf{x}) = \hat{\mathbf{y}}$. To find the parameters θ that best fit the data, a loss function over the parameters' space is defined, which quantifies the quality of the model, usually through a measure of distance between the true value of the output and the estimated output of the algorithm. The loss function is also known as objective or cost function.

[1] `https://www.youtube.com/watch?v=WXuK6gekU1Y`

For regression problems, typical loss functions to compare true versus predicted outputs are:

- The mean absolute error (MAE):

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} |\mathbf{y}_i - f_\theta(\mathbf{x}_i)|.$$

- The root mean squared error (RMSE):

$$L(\theta) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} [\mathbf{y}_i - f_\theta(\mathbf{x}_i)]^2}. \tag{3.1}$$

- The mean absolute percentage error (MAPE):

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \left| \frac{\mathbf{y}_i - f_\theta(\mathbf{x}_i)}{f_\theta(\mathbf{x}_i)} \right|. \tag{3.2}$$

For classification problems of k different classes, the output of the model returns a vector of dimension k, $f_\theta(\mathbf{x}) = \mathbf{p}$, where each component p_j is the probability of the sample $\mathbf{x}$ being of class j according to the model. For this application, the most common loss function is the log-loss or cross-entropy function:

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \left[\sum_{j=1}^{k} \delta_{jy_i} \log p_j \right],$$

where y_i is the label of the i-th data and δ_{jy_i} is the Kronecker delta.

The above loss functions are defined over the multivariate parameter space and satisfy to be always non-negative and only equal to zero if the model fits perfectly the data. The latter rarely happens as the model could be inadequate or insufficient to fit the data, or the data itself could contain stochastic fluctuations which make a perfect fit impossible (consider the errors appearing in a perfect linear model). Instead, minima of the loss functions are obtained for different models, which, when using the same metric, help to compare the different models. Additionally, for ML algorithms, and more specifically for neural networks, the parameter space can range from hundreds to millions of parameters. This makes it infeasible to find an analytical solution, urging the need of numerical methods to minimize the loss function, as will be explained more concretely for neural networks in Secs. 3.2 and 3.3.

Choosing different forms of the loss function for the same goal, such as MAE or RMSE for a regression problem, has two implications, which are correlated: i) The gradient of the numerical optimization varies, having larger smoothness, being convex, etc. ii) Since the perfect model does not usually exist, the minimum non-zero of the function will depend largely on the form of the loss function and how it quantifies the mismodeled data samples, e.g., taking the loss in the outliers more into account, etc.

When fitting the parameters of a model through a loss function, the model can get too specific to the training data, losing its generalization value to new data. This phenomenon is called overfitting, and holds especially true for the two most used ML algorithms at the time of this writing, neural networks and boosting decision trees [71]: the first ones usually have large number of parameters, sometimes even greater than the size of the training samples; the latter ones can construct an arbitrary size of chained models. On the other hand, if the model is too simple or the parameters are not adjusted well enough to the training data, i.e., the model is underfitted, it will also mispredict on new samples. In practice, a model should be simple and predictive at the same time, which is known as the general principle of machine learning. The trade-off between both is called the bias-variance trade-off, illustrated in Fig. 3.1. The goal is to obtain a model that fits and predicts properly the data in (a). In (b), the model fits pretty decently the data, but is too complex to accurately predict new data. In (c), the model is simpler, but the prediction is not accurate, giving a large error through the loss function. In (d), both simplicity and predictability are achieved, minimizing the loss function while generalizing properly to new data samples.

For the neural network models focused in this work, underfitting is not a concern, while overfitting to the training data is, due to the large flexibility these networks have of adapting to data. To avoid overfitting a model with a large number of parameters, there are two kinds of techniques: model independent ones and model dependent ones. The latter one for neural networks will

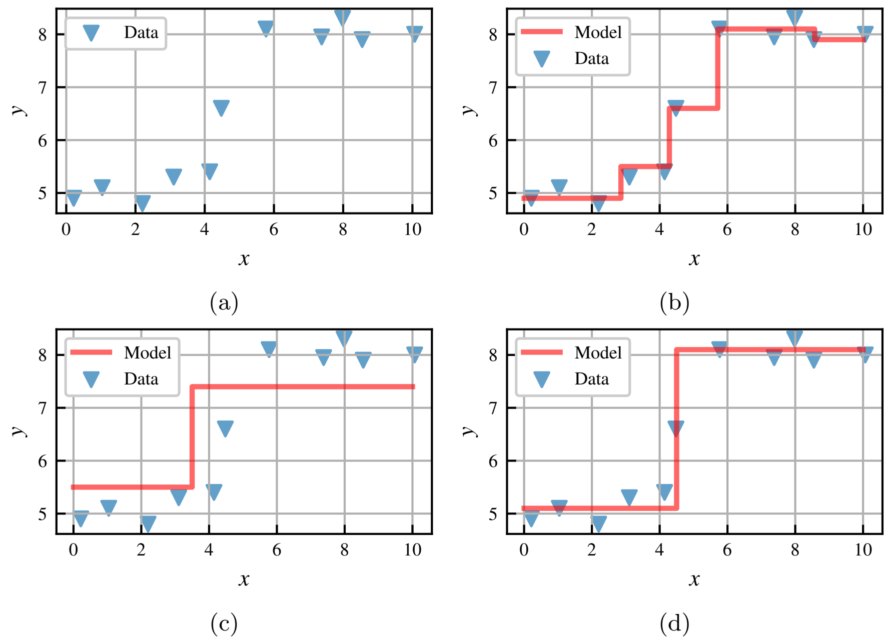


Fig. 3.1: Example of the general principle of machine learning: a model must be simple and predictive. In this case, we want to find a model fitting the data of (a). In (b) the model is too specific, because of its large complexity. In (c) the model does not fit the training data close enough, hence the new predictions will be poor. In (d) both the complexity and the predictability of the model are balanced, having a good bias-variance trade-off.

be discussed in Sec. 3.3.4. Regarding model independent techniques, holdout and cross-validation are model validation techniques to assess that the model generalizes well enough when trained thoroughly in data coming from the same distribution as the one used for training.

The simpler method of the two, the holdout method, consists in dividing the training dataset into three disjoint subsets: the training set, to optimize the model's parameters; the validation set, to optimize hyperparameters and avoid overfitting; the test set, to verify the integrity of the model for new data.

- Training set: The data the ML algorithm sees during the optimization process and fits its parameters to.
- Validation set: The data outside of the ML optimization process used to stop the training once the improvement has halted for this set, or, equivalently, to save the model that best adjusts to the validation set during the training process according to a chosen metric. Additionally, the validation set is utilized to fine tune the hyperparameters of a model, to maximize the generalization of the algorithm by comparing how the model behaves with different hyperparameters for the validation set. Hyperparameters are parameters of the model which cannot be learned and optimized from data, but have to be chosen by hand due to experience or by trial and error. There are some algorithms to optimize these hyperparameters based on the metric received from the validation set, such as Bayesian optimization [72], but they are out of the scope of this work.
- Test set: The data used after optimizing the parameters with the training set and choosing a model with the validation set, to verify that the model really generalizes to data not used at all during the model construction phase, as the validation set is used indirectly when constructing the final model.

The general principle allows us to define the ideal model in terms of balance underfitting and overfitting, while the model selection can be performed utilizing the validation loss. This is shown in Fig. 3.2, where the final model should be the one minimizing the loss of the validation set, which contains data not seen directly by the model fitting stage. In Sec. 3.3.4, early stopping, a method of regularizing and selecting the proper model, will be discussed, which precisely utilizes the validation loss curve.

The split of the dataset into the three subsets is usually performed at random in order to avoid model dependence on the data chosen, and to cover the whole distributions of the input and target spaces. In certain cases, however, the division has to be done in a more sophisticated way, taking into account that there might occur a phenomenon such as data leaking (discrepancy between test set performance and out of data performance due to training and

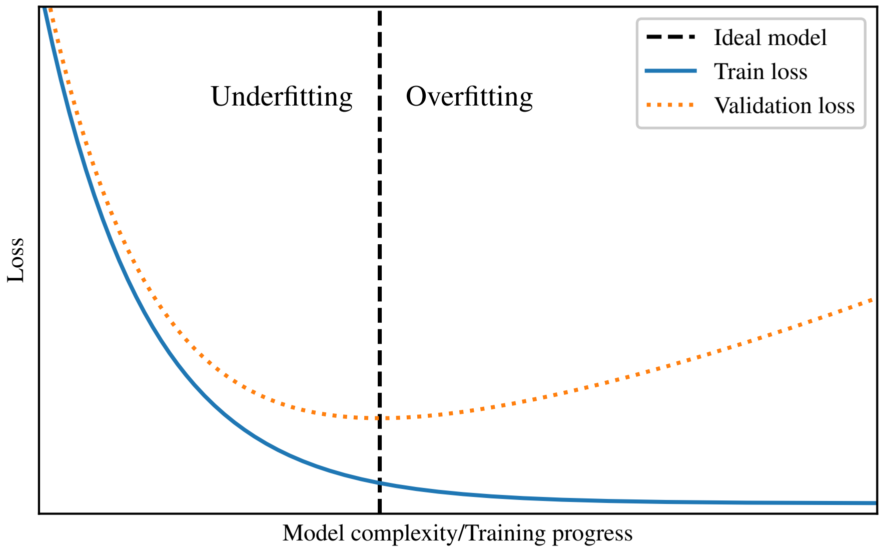


Fig. 3.2: Model selection/training progress showing the discrepancy between training and validation loss, depicting the ideal model where the algorithm minimizes the loss for data not used to fit parameters. If the model is too complex or one trains to overfit on training data, the model might generalize poorly on out-of-train data, such as the validation set, which allows to measure the model's performance on precisely new data.

testing on a specific distribution, overfitting to it and not generalize well). The percentage dedicated to each of the subsets may vary depending on the amount of data available, but usual ratios for train-validation-test are 0.6-0.2-0.2 and 0.8-0.1-0.1.

As an example of a typical cross-validation implementation, the k-fold cross validation [73] consists in utilizing k-folds on the training+validation sets, where k models are build using different folds for training and validation.

3.2 Building a feed forward neural network

In computational terms, the brain is a very versatile and powerful tool of learning and recognising patterns, which is precisely the goal of a supervised learning algorithm. It can filter out noise and properly process inconsistent data, while producing quite accurate answers from very high dimensional data (such as images or different sounds) in a short amount of time.

At the most basic level, the processing units of the brain are nerve cells called neurons. On average, the number of neurons in a brain is of the order of 10^{11}, interconnected as a complex network. Their main task, however, is quite simple: through the fluids of the brain, transmitter chemicals raise or lower the electrical potential inside the neurons. If the membrane potential surpasses some threshold, the neuron fires a pulse of fixed strength and duration down the axon (which is an elongation of the neuron to precisely transmit this pulse). The axon then divides itself into different connections, known as synapses, which transmit the fired pulse to many other neurons. Hence, each neuron can be seen as a separate processor, deciding whether to fire an impulse or not, making the brain a highly parallel computer.

Even with such a simple behaviour as the one explained above, the brain adapts quickly to many kinds of tasks. Additionally, there is a strong belief that neurons are the same all over the brain, but they adapt to their specific function. This is known as plasticity: the brain can modify the strength of the synaptic connections between neurons, and create new connections depending on their purpose.

In 1949, Donald O. Hebb proposed a model for explaining the plasticity and adaptation of the brain in his book *The Organization of Behavior* [61], known as Hebb's rule:

> Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. [...] When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

This means that the change in strength of synaptic connections is proportional to the correlation in the firing of the two connecting neurons. Therefore, if two consecutive neurons fire consistently, their connections will change in strength, making it stronger. Contrary, if two consecutive neurons do never fire simultaneously, their connection dies away.

Having this picture in mind, neural networks (NNs) are ML algorithms which emerged with the goal of simulating the brain's behaviour of learning and adapting. Even if they arise originally in analogy to a biological brain, they have followed their own evolution, retaining only its name. The first scratch of the framework appeared even before the publication of Hebb's principle, in 1943, with a first model coming from McCulloch and Pitts [74], as shown in Sec. 3.2.1. With this basic mathematical model of how neurons work, the perceptron is the first ML algorithm to be explored in Sec. 3.2.2, and acts as the elemental building block of NNs. With the surge of powerful parallelizable computing via GPUs, NNs are nowadays one of the most important ML algorithms because of the complexity they can capture by stacking perceptron-like pieces in imaginative ways, being the feed forward neural network architecture in Sec. 3.2.3 the basic composed structure. To train a composed neural network as the feed forward one, the backpropagation algorithm will also be introduced.

3.2.1 McCulloch and Pitts neurons

In 1943, Warren S. McCulloch and Walter Pitts published their paper *A logical calculus of the ideas immanent in nervous activity* [74], which presents a very simple model of how the firing of a neuron is related to their membrane potential and the firing of other neurons. The model they presented is also known as the "all or nothing", since a neuron either fires or does not fire, represented by a 1 and 0, respectively.

The model, shown in Fig. 3.3, has three parts:

1. A set of weighted inputs w_i corresponding to the connections.
2. An adder that sums the input signals (the electrical potential that is collected in the membrane).
3. An activation function σ (which corresponds initially to the threshold function) deciding whether the neuron fires or not with the current inputs.

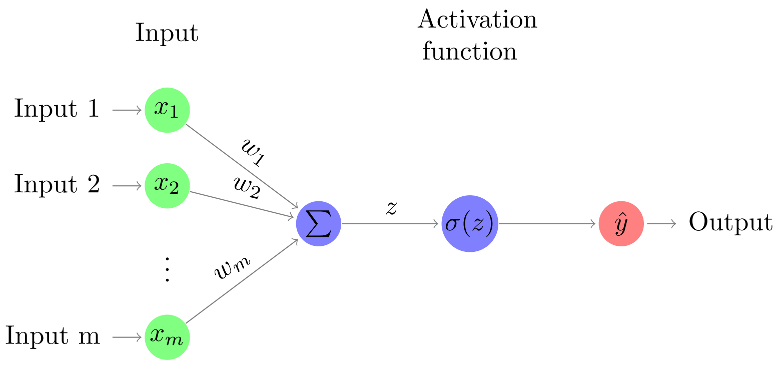


Fig. 3.3: Schematic drawing of how the McCulloch and Pitts neuronal model works to describe the firing of a neuron. m inputs are fed into the model, each having a weight w_i corresponding to the strength of the synaptic connections of each input. They are summed up into a variable $z = \sum x_i w_i$, which is then passed to an activation function σ. In biological terms, the activation function decides whether there is enough membrane potential for the neuron to fire or not via a threshold function (as in Eq. (3.3)), but in NNs they can be any function, introducing non-linearity to the algorithm when multiple layers are added.

In real life, the inputs come from other neurons which are firing some signal, hence they are either 0 or 1. For NNs however, the inputs can have any value, although they may not have any biological meaning. Each of those neurons that fires (or not) a signal, passes it through the synapses, which have different strengths, called *weights* w_i. The strength of the synapse influences the strength of the signal the neuron gets, hence the need to multiply the input times the weight of the synapse. Assuming all signals gather at the neuron without disappearing, they are added up to see if there is enough membrane potential to fire:

$$z = \sum_{i=1}^{m} x_i w_i .$$

After computing the total potential, it passes through the *activation function* for the neuron to fire. In the biological case, it would be a threshold function with parameter θ,

$$\hat{y} = \sigma(z) = \begin{cases} 1, & \text{if } z > \theta, \\ 0, & \text{if } z \leq \theta, \end{cases} \tag{3.3}$$

to decide whether the neuron fires or not. In general for NNs, however, the activation function can be very different, such as the sigmoid or the hyperbolic tangent functions, and helps to introduce a non-linearity to the algorithm when dealing with more than one layer.

Although the McCulloch and Pitts neuronal model is far from being realistic, it serves for the ML algorithm to have a solid and simple basis to construct the perceptron.

3.2.2 The perceptron

In the previous section the McCulloch and Pitts model for a single neuron was introduced. This, however, is not interesting from the ML point of view, in the sense that it is not able to learn. Instead, for a proper algorithm, a set of neurons should be put together, forming a neural network. Once these neurons are put together, one should be able to understand how the NN can learn the appropriate weights from the training dataset. The basic NN for supervised learning is the *perceptron*.

The perceptron algorithm was introduced by Frank Rosenblatt in 1957 at the Cornell Aeronautical Laboratory [64]. It is one of the most simple NNs one can construct, consisting only of grouping many McCulloch and Pitts neurons together with the same input. Figure 3.4 displays an example of a case with 4 inputs and 5 outputs, where each of the outputs share the same activation function σ. This can change in some specific cases, where one is interested in using a custom activation function $\sigma_1, \ldots, \sigma_5$ for each of them.

Notice that neurons are independent of each other: they use their own weights to compute the sum z_i, which decides whether to fire or not, regardless of other neurons performing that action. The only thing they share are the inputs. The goal is for the perceptron to learn to reproduce a particular target vector for a given input via the training data. The parameters that are learned for a concrete perceptron are the weights W_{ji}, where i is the index of the input vector and j is the index of the output. In the next section we will see that

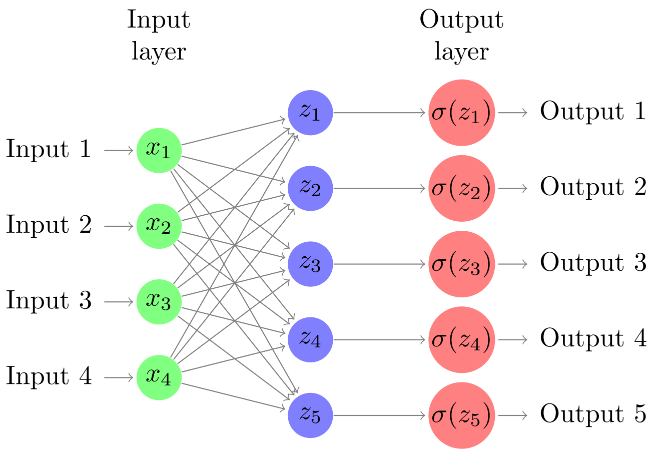


Fig. 3.4: Scheme of perceptron example. It is an ensemble of McCulloch and Pitts neurons sharing the same inputs. In general, they also share the same activation function σ, although this can be customized in practice to have different functions $\sigma_1, \dots, \sigma_5$. The perceptron is a method to find the weights of w_{ji}, where i is the index of the input and j is the index of the output neuron. It does so by learning from training data. In this case there are 4 inputs and 5 outputs.

the indices are put in this order to provide a clean matrix notation for bigger networks. We will now introduce a heuristic form of learning, which is not used in practice. Instead, a method known as backpropagation, which will be shown during this introduction, is used in the learning process.

With the scheme in Fig. 3.4 one can compute the output for a given input. Having a training set, one wants to adjust the weights in a way so that the loss function over all the dataset is as small as possible. Imagine that only one of the output neurons, the component k of the target $\mathbf{y}$, is wrong (the output does not match the target) for binary outputs. Consider that there are m input neurons, hence having m weights W_{ki}, $i \in \{1, \dots, m\}$ connected to it. To learn whether each weight is too big or too small, in order to correct the output, one starts by computing $\hat{y}_k - y_k$, the difference between our output $\hat{y}_k$

and the target value y_k, which is a very simple loss function. The change in weight is then

$$\Delta W_{ki} = -(\hat{y}_k - y_k) \cdot x_i.$$

This makes sense, since one needs to take into account the sign of the input and the difference between the prediction and the target to make corrections.

One way is to simply add this change in weight to the original one. This, however, would make the weights change too abrupt for each data point introduced in the training. Instead, one defines a *learning rate* η to regulate the learning. Typical values for η are between 0.1 and 0.4 (Sec. 3.3 of [75]). With this, the rule for updating the weight W_{ji} is

$$W_{ji} \leftarrow W_{ji} + \eta \cdot \Delta W_{ki},$$

enabling to teach the perceptron to move towards the appropriate weights.

To finish with the perceptron method, a *bias input* b_j is introduced as an additional input with label 0, x_0, which always has the value 1. Hence an extra constant is added when adding over the inputs,

$$z_j = \sum_{i=0}^{m} W_{ji} x_i = W_{j0} + \sum_{i=1}^{m} W_{ji} x_i = b_j + \sum_{i=1}^{m} W_{ji} x_i,$$

where an off-set has been added for the summation of the neurons. In the biological case, it can change thresholds of different activation functions σ. This input $x_0 = 1$ is called the *bias node* and exists for every neuron. This basically means that there is a constant b_j which has to be adjusted for all neuron activations.

3.2.3 Feed forward neural network and backpropagation

The perceptron method introduced in the last section is easy to understand and to use, but it is only a linear model, being able to identify straight lines, planes or hyperplanes. When confronting a sufficiently interesting problem, this usually is not enough. To make the network more complex, neurons between the input and output nodes can be added.

Commonly, instead of adding single neurons, one adds layers consisting of multiple neurons. For example, in Fig. 3.5, a 4 layer NN is shown (the input

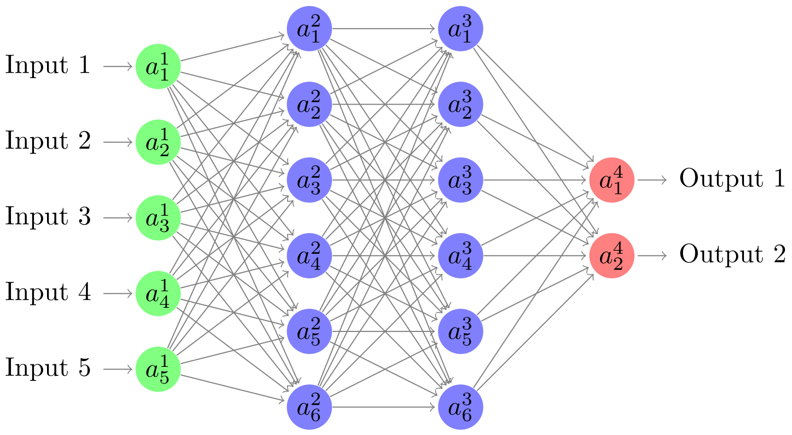


Fig. 3.5: Example of a deep learning neural network. It consists of 4 layers (the input, the output and 2 hidden layers), where the output of each layer serves as input for the next one. The input layer has 5 neurons, the 2 hidden layers have 6 neurons each, and the output layer has 2 neurons.

and output layers also count). Each output of a layer is used as input for the next one. In this particular case, there are 5 input neurons, 2 intermediate or *hidden* layers of 6 neurons each, and an output layer of 2 neurons.

Let us denote the value of the neuron after activation at each layer by a_i^l, where l is the layer going from 1 to L, with L the maximum number of layers (in Fig. 3.5 $L = 4$), and i indicating the neuron in hand (the range of i depends on the layer one is considering). The *output vector* at each layer l will be denoted simply by a^l. The *bias vector* at each layer will be labelled b^l, being b_i^l the bias at each neuron of the l-th layer, and the *weight matrix* connecting the layer $l-1$ and the layer l will be W^l. In the perceptron notation, the relation between neurons is

$$a_j^l = \sigma(z_j^l) = \sigma\left(\sum_i W_{ji}^l a_i^{l-1} + b_j^l\right).$$

Hence, one can write the computation of whole layer using matrix notation,

$$a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l),$$

where the activation function σ is applied to each component as a single-valued function.

Now, given a NN such as the one in Fig. 3.5, one is able to compute the output by applying the previous equation in succession for all the layers. This process is known as *forward propagation*, and this kind of standard NN is known as (deep) feed forward neural networks. Additionally, it is a fully connected neural network, since all neurons from one layer are connected to all neurons of the next layer.

For feed forward neural networks to learn, the method used in the last section for the perceptron is not very useful, since one only gets a total error for the network in the output layer. This makes it hard to know how each of the internal weights and biases of the hidden layers affected the output (hence the name, one does not know directly how they influence the outcome) and one does not know how to modify them. A different approach was developed by Rumelhart *et al.* in 1986 [76], where the final error was passed backwards through the network. This method is known as *backpropagation*, and will be explained in depth, since it is the standard procedure to fit parameters of all neural networks. The goal, as usual for supervised learning, is to minimize the loss function.

Consider a loss function $C(a_j^L, y_j)$, which measures how bad the prediction a_j^L is compared to the true value y_j. It can be any differentiable function such as RMSE or cross-entropy. There will be one assumption on this loss function: it has to be linear on each of the training samples, i.e., $C = \sum_x C_x$, where C_x is the error for the x-th training sample. This is reasonable because the error should help to fix individual mistakes in the predictions when computing the cost without canceling each other out as an ensemble, and all loss functions introduced in Sec. 3.1 take this form. The objective is to minimize this loss function for the training space, i.e., to obtain the best parameters θ in the parameter space of weights and biases, where θ denotes both W and b. Since this space is too large to find them analytically, the parameters θ are initialized randomly, and then the gradient of C with respect to the parameters θ is used to minimize it numerically. This can be done in many ways, such as the

Stochastic Gradient method [77] and Adam optimizers [78], as will be discussed later. All of them depend on the gradient of the loss function C with respect to the parameters, $\nabla_\theta C$, hence we need to have a way to compute it. This is the task of the backpropagation.

The backpropagation algorithm to compute $\nabla_\theta C$ is explained in great detail in Chapter 2 of Michael Nielsen's online book *Neural Networks and Deep Learning* [79], with several examples. To compute $\partial C/\partial W^l_{ji}$ and $\partial C/\partial b^l_j$, a quantity

$$\delta^l_j := \frac{\partial C}{\partial z^l_j},$$

is defined, which is the "error" term of neuron j in layer l. Since $z^l = W^l a^{l-1} + b^l$, by knowing this error, one can compute the partial derivatives with respect to the weights and biases of that layer using the chain rule:

$$\begin{aligned}\frac{\partial C}{\partial W^l_{jk}} &= \frac{\partial C}{\partial z^l_j}\frac{\partial z^l_j}{\partial W^l_{jk}} = \frac{\partial C}{\partial z^l_j} a^{l-1}_k =: \delta^l_j a^{l-1}_k,\\ \frac{\partial C}{\partial b^l_j} &= \frac{\partial C}{\partial z^l_j}\frac{\partial z^l_j}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} = \delta^l_j.\end{aligned}$$

Hence, it is crucial to compute the δ^l for all the layers to obtain the gradient of C.

For the output layer one has that it is simply

$$\delta^L_j = \frac{\partial C}{\partial z^l_j} = \frac{\partial C}{\partial a^l_j}\frac{\partial a^l_j}{\partial z^l_j} = \frac{\partial C}{\partial a^l_j}\sigma'(z^l_j),$$

or in matrix notation,

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L),$$

where $\odot$ is the entrywise or Hadamard product. This means that the error δ^L of the output layer can be computed directly after forward propagation. Now, by finding a relation of δ^l as a function of δ^{l+1}, one can compute the error terms of all the layers, going backwards from δ^L. The exact equation, using again the chain rule, can be obtained:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} W_{kj}^{l+1} \sigma'(z_j^l),$$

where

$$z_k^{l+1} = \sum_i \sigma(z_i^l) W_{ki}^{l+1} + b_k^{l+1}.$$

With this, the relation between δ^l as a function of δ^{l+1} is

$$\delta^l = \left(\left(W^{l+1} \right)^T \delta^{l+1} \right) \odot \sigma'(z^l),$$

and all the error terms for all the layers can be computed.

To sum up, the backpropagation is a method to compute the partial derivatives of the loss function with respect to the weights and biases for one training sample. It consists of the following four equations:

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L), \tag{3.4a}$$

$$\delta^l = \left(\left(w^{l+1} \right)^T \delta^{l+1} \right) \odot \sigma'(z^l), \tag{3.4b}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l, \tag{3.4c}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}. \tag{3.4d}$$

The algorithm consists in computing so-called "error" terms, δ^l, for each layer. To do so, first one has to do a forward propagation, i.e., evaluate the network for the training sample, and save all the a^l and z^l. Then, using Eq. (3.4a), one computes the first error term, δ^L. Once this term is computed, using Eq. (3.4b), one calculates in a recurrent way all the other error terms δ^l. With all the error terms obtained, one obtains the partial derivatives of the loss function by using the Eq. (3.4c) and (3.4d).

Why is this clever algorithm used to find the partial derivatives? Instead, one could approximate the derivatives of the loss function with respect to a parameter θ_i simply by their approximate partial derivatives,

$$\frac{\partial C}{\partial \theta_i} \simeq \frac{C(\theta_i + \varepsilon) - C(\theta_i)}{\varepsilon}.$$

Working with the order of thousands or millions of parameters θ_i, one would have to evaluate the network the same order of times to compute all derivatives. The backpropagation algorithm, however, takes of the same order in time as doing a single forward propagation, obtaining all the derivatives at the same time in a very efficient way.

With the partial derivatives of the loss function, one might apply many different algorithms to update and make the NN learn, i.e., minimize the loss function C, as stated earlier, which will be described in Sec. 3.3.1.

As will be seen in the next section and in Chapters 4 and 5, deep learning based on neural networks has started to drift away from their biological origin while maintaining the concept of highly connected structures with differentiable components to compute their gradients. This has induced a shift in denoting deep learning as representations of biological neural networks to a branch within differentiable programming [80–82]. For the rest of the thesis, however, we will still consider the frameworks as part of neural networks, since this change of paradigm is one of the discussion issues in the community as of today.

3.3 Components of a neural network

In the previous section, it has been described how a feed forward neural network is built and how the gradient of its parameters are computed with respect to the loss function. With the understanding of the formal part of the network laid out, this section covers some additional components part of neural network algorithms. In particular, Sec. 3.3.1 will describe the most popular optimizers used to apply the gradients to the parameters to fit the network. Afterwards, in Sec. 3.3.2, activation functions for the hidden and final layers will be depicted, some inspired by the threshold function of McCulloch and Pitts while others deviating strongly from the original function to introduce improvements. In Sec. 3.3.3, the random initialization of the parameters will be explained, arguing why certain distributions for the randomization are preferred to others. Finally, in Sec. 3.3.4, regularization techniques to avoid overfitting neural networks are shown.

3.3.1 Neural network optimizers

Backpropagation enables the computation of partial derivatives of the loss function with respect to the parameters of a NN. If the gradients would be applied directly on the weights to update them, the learning process would become unstable, as it is usual to overshoot in a parameter update step when utilizing the raw gradient. *Optimizers* are methodologies that apply changes to the parameters of the NN to reduce the loss function adequately. Two of these algorithms will be shown in this section: the stochastic gradient descent and the Adam optimizer.

The *stochastic gradient descent (SGD)* [77] algorithm is based on the Robbins–Monro algorithm [83], which modifies the standard gradient descent to use stochastic subsets of the training data instead of the whole ensemble. Consider having N samples in the training data. SGD selects randomly M samples to form a so called *batch* of data, and computes the derivatives of the weights and biases of the NN for each of them using Eq. (3.4). The weights and biases (jointly denoted by weights w from now on) of each layer are updated as follows:

$$w \leftarrow w - \eta \frac{\partial C}{\partial w}, \quad \text{with } \frac{\partial C}{\partial w} = \frac{1}{M} \sum_{m=1}^{M} \frac{\partial C^m}{\partial w},$$

where η is the learning rate to tune the amount of change in the parameters and $\partial C^m / \partial w$ is the partial derivative for the m-th data. Notice that this is simply subtracting the batch's average partial derivative of the loss function with respect to the parameter, hence moving in the opposite direction of the gradient in the parameter space and minimizing the loss function.

The purpose of utilizing a random subset of the training set to compute the derivative and not the whole set is to accelerate the computation, since one needs to perform M backpropagations instead of N while maintaining approximately the same direction in the parameter space. The statistical noise from selecting constantly only a subset also helps to avoid getting stuck in a local minimum.

SGD was used as a base to construct all subsequent optimization algorithms, i.e., the notion of using smaller random batches to approximate the derivative of the loss function. Momentum based optimization [84], aside from updating the weights, computes a momentum of the gradient which is also applied and updated. Adaptive gradient (AdaGrad) [85] constructs a per-parameter learning rate during the optimization. Root Mean Square Propagation[2] (RMSProp) also computes an adapted learning rate for each of the parameters by dividing it for a weight by a running average of the magnitudes of previous gradients.

Adaptive Moment Estimation (Adam) [78] is another algorithm based on RMSProp, which itself is based on SGD. As its predecessor, it utilizes individual learning rates for each parameter, but in this case running averages of both gradients and second momentum of the gradients are used. Consider the training iteration t in order to compute the weights at $t+1$. Adam's parameters' update is the following:

$$\begin{aligned}
m_w^{(t+1)} &\leftarrow \beta_1 \cdot m_w^{(t)} + (1-\beta_1) \cdot \partial C/\partial w, \\
v_w^{(t+1)} &\leftarrow \beta_2 \cdot v_w^{(t)} + (1-\beta_2) \cdot (\partial C/\partial w)^2, \\
\hat{m}_w &= \frac{m_w^{(t+1)}}{1-\beta_1^{t+1}}, \\
\hat{v}_w &= \frac{v_w^{(t+1)}}{1-\beta_2^{t+1}}, \\
w^{(t+1)} &\leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \varepsilon},
\end{aligned}$$

where β_1 and β_2 are the exponential decay rates for the first and second momentum of the gradient, η the learning rate and ε a small factor (10^{-8}) to avoid division by 0.

Adam is straightforward to implement, not memory nor computational intensive and adapts each component of the parameter space with its own lower momenta. The majority of practical cases benefit highly from Adam, since it

[2] RMSProp was first introduced by Geoff Hinton on a Coursera online course. Slides can be found here: `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`

speeds up the convergence by building adequate momenta. The hyperparameters β_1, β_2, η and ε are very intuitive to understand and to tune. All of this makes Adam the go-to optimizer for NNs. It is important to note that the first and second moments might hinder a proper learning in very concrete cases. For this work, Adam was successfully used in all applications.

3.3.2 Activation functions

Biologically, the McCulloch and Pitts neurons interact with certain weights, forming a membrane potential z which leads to activate the connected neuron according to a threshold function $\hat{y} = \sigma(z)$. For NNs, however, this activation function σ can be any function, as there is no biological meaning behind it. The main purpose of activation functions are to introduce non-linearity in an otherwise linear combination of variables. This allows the NNs to have a far richer expressiveness (i.e., the networks are able to approximate a wider family of functions) than otherwise linear. In fact, there is a large branch of studies that show through different activation functions and network architectures that feed-forward neural networks are universal approximators of continuous functions [86–91]. The other utility activation functions have is to restrict the output on a certain domain, e.g., in the range [0,1], in $\mathbb{R}^+$, etc. All univariate activation functions described are depicted in Fig. 3.6.

The first activation function used was the sigmoid function, as it can be interpreted as a smooth threshold function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}.$$

This activation function gives outputs between 0 and 1, adds a non-linearity for values which are not close to zero and has non-zero gradient, which is fundamental to apply backpropagation properly, but at the same times is similar enough to the threshold function to justify its usage by early believes of neural networks having to be similar to biological ones. Sigmoid functions can be utilized as a last output activation to ensure obtaining values between 0 and 1, such as when trying to assign a binary probability to the data. However, chaining many hidden layers of sigmoid functions can lead to the vanishing gradient phenomenon, as the gradient of sigmoid function quickly saturates outside its

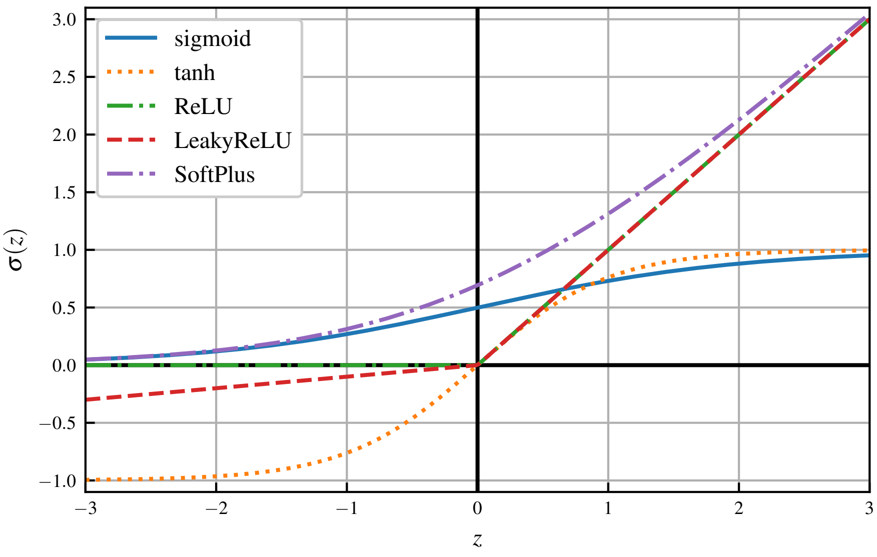


Fig. 3.6: Typical activation functions for neural networks to induce non-linearity in the algorithm.

center. When multiplying these small gradients in the chain rule, one obtains gradients close to zero, making it hard for the NN to update properly.

A similar activation function used is the hyperbolic tangent,

$$\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

which provides an output centered around zero with values in the range [-1,1]. Even though the output is now centered, tanh suffers from the same issue as the sigmoid regarding the vanishing gradient.

Both sigmoid and hyperbolic tangent come from the concept of imitating the McCulloch and Pitts neuron model. In 2000, Hahnloser et al. [92, 93] introduced a non-saturating, very simple, but highly effective new activation function: the Rectified Linear Unit (ReLU),

$$\sigma(z) = \mathrm{ReLU}(z) = \max(0, z).$$

This function is continuous and differentiable in every point except 0. For $z > 0$, the gradient is always equal to 1, getting rid of the vanishing gradient problem. Even though the non-linearity is very simple, because the function and its gradient are computationally very cheap to compute, one can stack a large amount of simple ReLUs to induce stronger non-linearities.

Since the appearance of ReLU, many small tweaks on non-saturating activation functions keep appearing, such as the Exponential Linear Unit (ELU) [94], Scaled Exponential Linear Unit (SELU) [95] and Leaky ReLU [96]. The latter one will be used through this work, and consists in modifying ReLU for negative values:

$$\sigma(z) = \text{Leaky ReLU}_\alpha(z) = \max(\alpha \cdot z, z).$$

Here, α is a hyperparameter of the model, with usual values between 0.01 and 0.1, and quantifies how much the activation "leaks". Adding this condition for negative values solves one of the main issues of the original ReLU: saturating in negative values, obtaining only zero gradients. The simplicity of ReLU and its non-linearity is still kept, making it an excellent evolution of the original activation function.

The last univariate activation function visited will be the SoftPlus function [97, 98]:

$$\sigma(z) = S_+(z) = \log(1 + e^z). \tag{3.5}$$

This function allows to map $\mathbb{R} \to \mathbb{R}^+$, which will be useful when wanting to force a non-negative output.

There are also multivariate activation functions, i.e., where all the neurons of a layer are taken as input together and produce a certain output. The softmax activation takes as an input a real vector and outputs a normalized probability vector according to the values of such a vector. Each component i of the probabilities is given by

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{3.6}$$

This allows to give probabilities of different outputs, such as when performing a multiclass classification.

3.3.3 Initialization of the parameters

Neural networks follow an iterative process when optimizing their parameters, as discussed before while presenting the different methodologies to do so. Contrary to other numerical methods, the convergence to a minimum of this iterative process depends heavily on the initialization of the parameters, determining whether it converges at all or encounters numerical instabilities and fails to do so. When converging, the initial point can determine how fast learning converges, and if the minimum is close to the global one or not. Additionally, it can affect how the converged model generalizes to out-of-training data.

Initialization of a neural network is not well understood, forcing simple and heuristic strategies, many of which are focused on the network satisfying nice properties when initialized, such as having symmetry break between the initial values of the neurons and avoiding exploding or vanishing gradients. There is no guarantee, however, that these properties persist after training.

The only thing known for sure is the desire to break an initial symmetry so that the network does learn something different with each neuron. Two units connected to the same inputs and with the same activation function have to be initialized differently, otherwise they will be updated exactly the same. This motivates a random initialization, typically setting the biases at a constant heuristic value while the weights get assigned a random number from a distribution, usually a Gaussian or uniform distribution. Even though the shape of the distribution does not seem to matter at all, its scale does have a large impact on the network's ability to learn.

Large initial values yield a stronger symmetry-break effect and avoid loss of signal during forward and backward propagation. Too large values, however, can lead to exploding values, which produce exploding gradients, making the training highly unstable. Many proposals on how to balance the two extremes and find a compromise have been made, being the Xavier or Glorot initialization [99] the most popular one. For a fully connected layer of m inputs and n outputs, Glorot and Bengio proposed to use a uniform initialization for the components of the weight matrices W:

$$W_{jk} \sim \text{Unif}\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right),$$

while initializing the biases terms b to zero. This heuristic is designed to have for all layers both the same activation variance and same gradient variance, by making the assumption that there are only chains of matrix multiplications with no non-linearities. Through this work, Glorot initialization will be performed unless otherwise stated.

3.3.4 Regularization of neural networks

The expressiveness of neural networks allows them to adapt to complex relations between the input and output data. Because of their flexibility, this leads in many cases to overfit to the training data, as having a parameter space of thousands or millions of parameters may be large enough to memorize the training data.

Additionally, neural networks suffer from exploding/vanishing gradients during backpropagation, which can make some architectures unstable.

To prevent both these things from happening, different methodologies arise, which vary on how they approach the phenomenon of overfitting and exploding/vanishing gradients. Four techniques used in this work will be presented: batch normalization, dropout, gradient clipping and early stopping.

Batch normalization

As seen with the initialization and with the activation functions, the values of the neurons should always be close to zero in order to perform properly. Having a mixture of positive and negative values allows to prevent exploding and vanishing gradients. However, after beginning training, the initial distribution may change, starting to spread and displace itself. In 2015, Ioffe and Szegedy proposed a technique called batch normalization (BN) [100] to solve this issue. It adds an operation before applying the activation function of standardizing the inputs (i.e., making them have a mean of zero and a standard deviation of one), then shifting and scaling them as needed by the algorithm. This operation simply scales and shifts the data adequately so that the algorithm can profit from them as much as possible given the next operation in the NN. The equations to apply BN are:

$$\begin{aligned}
\mu_B &= \frac{1}{m_B}\sum_{i=1}^{m_B} x^{(i)},\\
\sigma_B^2 &= \frac{1}{m_B}\sum_{i=1}^{m_B}(x^{(i)}-\mu_B)^2,\\
\hat{x}^{(i)} &= \frac{x^{(i)}-\mu_B}{\sqrt{\sigma_B^2+\varepsilon}},\\
z^{(i)} &= \gamma\hat{x}^{(i)}+\beta,
\end{aligned}$$

where:

- $x^{(i)}$ is the i-th input of the BN.
- μ_B is the empirical mean over a batch of data B.
- σ_B^2 is the empirical variance over a batch of data B.
- m_B is the size of the batch.
- $\hat{x}^{(i)}$ is the zero-centered, normalized input.
- γ is the scaling parameter for the layer.
- β is the shifting parameter for the layer.
- ε is a small factor (10^{-8}) to avoid division by 0.
- $z^{(i)}$ is the i-th output of the BN.

Notice that the above equation is computed during training, as we are using a batch of data to obtain μ_B and σ_B^2. The parameters γ and β are optimized as usual via backpropagation. When evaluating the BN layer, it cannot rely on having a batch to compute the mean and variance. Instead, the layer saves these quantities as running averages of the means and variances obtained during training, and uses them when evaluating.

Ioffe and Szegedy showed in their work [100] that BN improved considerably all the neural networks they experimented with. Vanishing gradients disappeared, allowing one to use saturating activation functions such as the sigmoid or tanh. The networks were also less sensitive to weight initialization, higher learning rates could be applied and, therefore, faster convergence was found.

Dropout

Introduced in 2012 by Hinton *et al.* and further detailed in 2014 by Srivastava *et al.*, dropout [101,102] is one of the most popular and successful regularization techniques for neural networks, boosting even state-of-the-art neural network a 1-2%, which are already at 95% accuracy. The algorithm is quite simple: during every training step, each neuron (including input neurons but excluding output neurons) has a probability p of dropping out temporally, i.e., it will be ignored during the training step, coming back on the next iteration. The hyperparameter p is the dropout rate, typically in the range of [0.1,0.5]. After training, no neuron gets dropped out anymore, but the weights have to be multiplied by p, as all neurons will be present and the proper ratio to each connection has to be preserved.

The concept of dropout revolves around the idea that a neuron cannot depend on a single connection it has with another one, but has to learn to extract information from a group of neurons it is connected to instead. Additionally, a neuron cannot co-adapt with others of the same layer, as some of them might drop randomly, making more emphasis in the information contained in each neuron. This gives the network higher robustness to slight changes in the input and makes it generalize better.

Gradient clipping

Even with all the precaution of choosing small learning rates and applying batch normalization, gradients can explode and make the network unstable while training. The shape of the loss function over this large multivariate space might change drastically in different points, a phenomenon which occurs with certain network architectures more than with others. Gradient clipping regularization helps to prevent having abrupt changes in the network due to the gradient exploding by restricting its norm.

The simplest way of applying gradient clipping is by gathering all the gradients computed in a backpropagation step of the model into a one-dimensional vector $\Delta\mathbf{W}$ before applying any optimization algorithm. Then, the norm of the vector $\|\Delta\mathbf{W}\|$ is computed, usually the euclidean one, in order to define its normalized version $\Delta\mathbf{w} = \Delta\mathbf{W}/\|\Delta\mathbf{W}\|$. Given the hyperparameter of maximum norm, $m > 0$, the new gradient utilized for optimization $\Delta\mathbf{W}'$ is given by

$$\Delta\mathbf{W}' = \Delta\mathbf{w} \cdot \min(m, \|\Delta\mathbf{W}\|).$$

By redefining the gradient in each training step this way, a maximum norm m is ensured while maintaining the proper direction of the original gradient.

Early stopping

When expressive models such as neural networks have the capacity to overfit, one observes during training that the loss on the training set keeps decreasing steadily while for the validation set the error begins to rise again. By selecting the parameters of the model for which the error in the validation set is minimal, the model will in theory yield better results for new data, as it is generalizing best at that point in time. Therefore, each time the error on the validation set improves, the model's parameters are stored. When the algorithm terminates, the last saved parameters are restored rather than the latest ones, allowing us to select the ideal model in Fig. 3.2. The algorithm finishes once the validation error has not improved over a pre-specified number of iterations, which is a hyperparameter known as patience. This process, known as early stopping, is the most common regularization method in deep learning, due to its effectiveness and simplicity.

When training a model, sometimes the validation error stops improving over a short amount of iterations until it continues to do so. However, it might be the case that the model does not improve anymore and one is in the initial phase of the training. This makes the larger part of the learning procedure unnecessary, as the model is simply overfitting. By applying early stopping with an appropriate patience, both situations can be avoided, helping to remove one of the more delicate parameters to tune, the total number of training iterations.

4. NORMALIZING FLOWS

In modern science and engineering disciplines, well-specified probabilistic models are able to describe and reproduce the data processing of observed quantities and link it to an underlying theory. One fundamental task of current ML research is to develop new utilities which allow richer probabilistic descriptions to be made, enabling better-specified models.

In this chapter, normalizing flows will be presented as one of the tools available to build probabilistic distributions. Section 4.1 will introduce the general concept of normalizing flows while describing their abstract building blocks. Later on, Sec. 4.2 will present a specific implementation of these building blocks, the neural spline flows, which allow for very flexible and complex probabilistic distributions to be constructed in an efficient way. Finally, Sec. 4.3 will show a practical way of computing the building blocks through a single neural network in the form of an autoregressive network.

4.1 Normalizing flows

Normalizing flows construct probability distributions by pushing an initial density through a series of transformations to obtain a richer distribution, similar to a fluid going through a chain of tubes of different sizes and shapes. As will be shown, repeated applications of simple transformations over a unimodal initial density can lead to complex, multi-modal models.

In high energy physics, normalizing flows have been primarily used for sampling, e.g., lattice gauge theory and Drell-Yan type processes at the LHC sampling [103, 104], and integration, e.g., matrix elements for top-quark and gluon productions [105], but their versatility can also be applied for anomaly detection [106] and for parameter inference of simulator models [17], as will be seen in Chapter 6.

4.1.1 Basic concepts

From a historical point of view, the predecessor of modern normalizing flows were the whitening transformations [107, 108], shaping data into white noise with uncorrelated components, each with variance one, to enforce statistical independence of the variables to work in simpler environments. The first to use whitening as a density estimation technique instead of data preprocessing were Chen and Gopinath (2000) [109]. Tabak and Turner (2013) [110] defined the modern concept of normalizing flows, by being the first to introduce the term *normalizing flows* and describing them as a composition of K simple maps, as will be shown later. Constructing these transformations as parameterized flows with deep neural networks was shown by Rippel and Adams (2013) [111], recognizing the potential in defining expressive and general distribution classes. Dinh *et al.* (2015) [112] developed a scalable and computational efficient architecture for image modeling and inference. An in-depth review over the current state of normalizing flows was recently written by Papamakarios *et al.* [113], from which we will only concern on the parts applied through the thesis.

Normalizing flows enable to construct probability distributions over continuous random variables. Consider $\mathbf{x}$ to be a D-dimensional vector in $\mathbb{R}^D$ over which one would like to define a joint distribution. The idea of a flow-based modeling is to express $\mathbf{x}$ as a transformation T of a real vector $\mathbf{u}$ sampled from $p_{\mathbf{u}}(\mathbf{u})$:

$$\mathbf{x} = T(\mathbf{u}), \text{ with } \mathbf{u} \sim p_{\mathbf{u}}(\mathbf{u}) .$$

$p_{\mathbf{u}}(\mathbf{u})$ is referred to as the base distribution of the flow-based model. The transformation T usually has parameters ϕ which induce a family of density functions over $\mathbf{x}$ parameterized by ϕ.

The defining property of the transformation T in a flow is that it has to be invertible, and both T and T^{-1} have to be differentiable, i.e., T defines a diffeomorphism over $\mathbb{R}^D$, requiring $\mathbf{u}$ to be in $\mathbb{R}^D$ as well [114]. Under these conditions, the density of $\mathbf{x}$ is well-defined and can be computed via a change of variables for density functions:

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{u}}(\mathbf{u}) \, |\det J_T(\mathbf{u})|^{-1}, \text{ where } \mathbf{u} = T^{-1}(\mathbf{x}), \tag{4.1}$$

with $J_T(\mathbf{u})$ the Jacobian of the transformation, given by a $D \times D$ matrix of the partial derivatives of the transformation T:

$$J_T(\mathbf{u}) = \begin{bmatrix} \frac{\partial T_1}{\partial u_1} & \cdots & \frac{\partial T_1}{\partial u_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial T_D}{\partial u_1} & \cdots & \frac{\partial T_D}{\partial u_D} \end{bmatrix}.$$

The density can also be described in terms of the inverse transformation T^{-1}:

$$p_{\mathbf{x}}(\mathbf{x}) = p_{\mathbf{u}}\left(T^{-1}(\mathbf{x})\right) |\det J_{T^{-1}}(\mathbf{x})|. \tag{4.2}$$

In practice, the implementation of T (or T^{-1}) is given partially by a neural network, and the base density $p_{\mathbf{u}}(\mathbf{u})$ is chosen to be a multivariate normal, as it is a well understood distribution, both easy to sample and evaluate, with a domain covering the whole space $\mathbb{R}^D$ while being concentrated in a small and known area.

Intuitively, the transformation T can be thought of as expanding and compressing the space $\mathbb{R}^D$ to shape the base density $p_{\mathbf{u}}(\mathbf{u})$ into $p_{\mathbf{x}}(\mathbf{x})$. The relative change of volume in a small neighborhood around $\mathbf{u}$ through T is measured by the absolute Jacobian determinant $|\det J_T(\mathbf{u})|$.

If the transformation is flexible enough, the flow could be used to evaluate any continuous density in $\mathbb{R}^D$. In practice, however, the property that the composition of diffeomorphisms is a diffeomorphism is used, allowing one to construct a complex transformation via composition of simpler transformations (analogous to how rich feed forward neural networks are a sequence of simple layer functions). This does not compromise any of the properties needed to define a transformation for a change of variables, such as invertibility and differentiability. Consider the transformation T as a composition of simpler T_k transformations,

$$T = T_K \circ \cdots \circ T_1. \tag{4.3}$$

Assuming $\mathbf{z}_0 = \mathbf{u}$ and $\mathbf{z}_K = \mathbf{x}$, the forward evaluation and Jacobian are

$$\begin{aligned} \mathbf{z}_k &= T_k(\mathbf{z}_{k-1}),\ k = 1 : K, \\ |J_T(\mathbf{x})| &= \left|\prod_{k=1}^{K} J_{T_k}(\mathbf{z}_{k-1})\right|. \end{aligned} \tag{4.4}$$

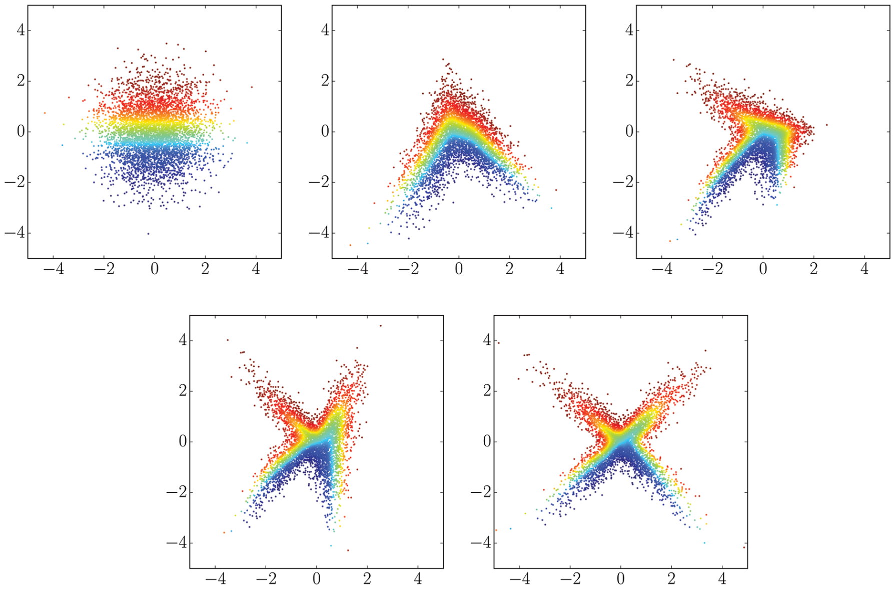


Fig. 4.1: An example of a normalizing flow, transforming the base density, a two-dimensional normal distribution (up left), to the target density, a star-shape distribution (bottom right). The complete transformation of the flow is a composition of four simpler ones (from top left to bottom right). The colors are added to follow the transformation of the points through the flow. Source: [113].

These two computations (plus their inverse) are the building blocks of a normalizing flow [115]. Hence, to make a transformation efficient (i.e., to have a small computational cost), both operations have to be efficient. Figure 4.1 shows a normalizing flow from a two-dimensional normal distribution to a star-shape distribution as a composition of 4 transformations. From now on, we will focus on a simple transformation $\mathbf{u} = T(\mathbf{x})$, since constructing a flow from it is simply applying compositions.

The expressiveness of a flow is the flexibility it has to adapt to arbitrary distributions. Papamakarios *et al.* [113] show that under reasonable conditions on $p_{\mathbf{x}}(\mathbf{x})$, a flow will be able to represent said distribution, i.e., flows are highly expressive. For this, the density has to satisfy that $p_{\mathbf{x}}(\mathbf{x}) > 0$ for all $\mathbf{x} \in \mathbb{R}^D$ and that all the conditional probabilities $\Pr(x'_i \leq x_i | \mathbf{x}_{<i})$ are differentiable with respect to $(x_i, \mathbf{x}_{<i})$, with $\mathbf{x}_{<i} = \{x_1, x_2, \ldots, x_{i-1}\}$.

4.1.2 Objective function

In order to fit a flow-based model, denoted from here on forth by $q_\phi(\mathbf{x})$, to a target distribution $p(\mathbf{x})$, some divergence or discrepancy between both distributions is minimized over the parameters ϕ of the transformation T of the flow. There are a number of divergences that could be used, such as f-divergences [116], that use density ratios for comparison of distributions, or integral probability metrics [117], which use differences to compare.

The most popular choice is the Kullback-Leibler divergence (KL-divergence) [118], a particular implementation of the family of f-divergences, defined as

$$D_{\mathrm{KL}}(p(\mathbf{x}) \| q_\phi(\mathbf{x})) = \int p(\mathbf{x}) \log\left(\frac{p(\mathbf{x})}{q_\phi(\mathbf{x})}\right) d\mathbf{x}. \tag{4.5}$$

As a divergence, it is always non-negative and only zero if and only if both densities are equal. Rewriting the above equation as a loss function with respect to the parameters ϕ, it becomes:

$$\begin{aligned} L(\phi) &= D_{\mathrm{KL}}(p(\mathbf{x}) \| q_\phi(\mathbf{x})) \\ &= \int p(\mathbf{x}) \log\left(\frac{p(\mathbf{x})}{q_\phi(\mathbf{x})}\right) d\mathbf{x} \\ &= \int p(\mathbf{x}) \log p(\mathbf{x}) \; d\mathbf{x} - \int p(\mathbf{x}) \log q_\phi(\mathbf{x}) \; d\mathbf{x} \\ &= -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log q_\phi(\mathbf{x})\right] + \text{ const.} \\ &= -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log p_{\mathbf{u}}\left(T^{-1}(\mathbf{x};\phi)\right) + \log|\det J_T^{-1}(\mathbf{x};\phi)|\right] + \text{ const.} \end{aligned} \tag{4.6}$$

Notice that the loss function can be computed for target densities $p(\mathbf{x})$ from which one can sample from but not necessarily evaluate explicitly its density for a given point $\mathbf{x}$. Consider having a set of samples $\{\mathbf{x}_n\}_{n=1}^N$ from $p(\mathbf{x})$. The

Monte Carlo estimate over the loss function expectation becomes:

$$L(\phi) \approx -\frac{1}{N}\sum_{n=1}^{N}\left[\log p_{\mathbf{u}}\left(T^{-1}(\mathbf{x}_n;\phi)\right) + \log|\det J_T^{-1}(\mathbf{x}_n;\phi)|\right] + \text{const.} \tag{4.7}$$

Minimizing the Monte Carlo approximation of the KL-divergence is equivalent to maximizing the log-likelihood of the $\{\mathbf{x}_n\}_{n=1}^{N}$ samples with the flow-model.

When performing optimization on the transformation T in the form of a neural network using methods such as SGD or Adam described in Sec. 3.3.1, the unbiased estimate of the gradient of the KL-divergence with respect to the parameters is:

$$\nabla_\phi L(\phi) \approx -\frac{1}{N}\sum_{n=1}^{N}\left[\nabla_\phi \log p_{\mathbf{u}}\left(T^{-1}(\mathbf{x}_n;\phi)\right) + \nabla_\phi \log|\det J_T^{-1}(\mathbf{x}_n;\phi)|\right].$$

To fit a flow-based model via maximum likelihood, one needs to be able to compute and to differentiate the inverse transformation T^{-1}, its Jacobian determinant and the base density $p_{\mathbf{u}}(\mathbf{u})$. The trained model allows one to evaluate the fitted density even if one is unable to compute T or sample from $p_{\mathbf{u}}(\mathbf{u})$, although both are required to be able to sample from the fitted model.

The KL-divergence is not symmetric, and the order of the factors was chosen purposefully as in Eq. (4.5). This form of the KL-divergence is known as *forward* KL-divergence, although the name is only a rhetorical convenience. The forward KL-divergence allows one to fit a model where, as stated, one is able to sample from the target but is not able to evaluate explicitly the target density. This situation is typical in many science models, as one is able to produce samples as a chain of stochastic decisions, whose density is either computationally prohibited due to complex integrals or directly unavailable.

When altering the order of the KL-divergence factors, one gets the so called *reverse* KL-divergence:

$$\begin{aligned} L(\phi) &= D_{\mathrm{KL}}\left(q_\phi(\mathbf{x})\,\|p(\mathbf{x})\right) \\ &= \mathbb{E}_{\mathbf{x}\sim q_\phi(\mathbf{x})}\left[\log q_\phi(\mathbf{x}) - \log p(\mathbf{x})\right] \\ &= \mathbb{E}_{\mathbf{u}\sim p_{\mathbf{u}}(\mathbf{u})}\left[\log p_{\mathbf{u}}(\mathbf{u}) - \log|\det J_T(\mathbf{u};\phi)| - \log p\left(T(\mathbf{u};\phi)\right)\right], \end{aligned}$$

where in the last equality a change of variable is performed to express the expectation with respect to $\mathbf{u}$. The reverse KL-divergence is suitable to fit the flow-model where one is able to evaluate the target density $p(\mathbf{x})$ but cannot necessarily sample from it. This even holds when the target density is only available to evaluate up to a constant ($\tilde{p}(\mathbf{x})$, with $p(\mathbf{x}) = \tilde{p}(\mathbf{x})/C$), as the normalization constant is a constant factor for the loss function which is ignored when computing the gradient during optimization. In particular, if $\{\mathbf{u}_n\}_{n=1}^N$ samples are drawn from the base density $p_{\mathbf{u}}(\mathbf{u})$, the Monte Carlo estimation of the gradient is:

$$\nabla_\phi L(\phi) \approx -\frac{1}{N}\sum_{n=1}^{N}\left[\nabla_\phi \log|\det J_T(\mathbf{u}_n;\phi)| + \nabla_\phi \log \tilde{p}\left(T(\mathbf{u}_n;\phi)\right)\right].$$

To minimize the reverse KL-divergence one needs to be able to sample from the base density as well as compute and differentiate the transformation T and its Jacobian determinant. Evaluating the base density or computing the inverse transformation T^{-1} is not necessary to be able to fit the model under the reverse KL-divergence, but they enable the evaluation of the trained model. This makes the reverse KL-divergence, in theory, a great candidate for obtaining a model from a target density which one can evaluate from but is hard to sample. This is not necessarily the case, however, as the term $\nabla_\phi \log \tilde{p}\left(T(\mathbf{u}_n;\phi)\right)$ might explode when the density $q_\phi(\mathbf{x})$ is far from the target $\tilde{p}(\mathbf{x})$, and a tweaked forward KL-divergence might fit better the task, which will be discussed in Chapter 7.

4.1.3 Constructing finite composition flows

To construct a flow, the transformation that defines it can be considered a finite composition as shown in Eq. (4.3) or a continuous time flow [119], where the transformation takes the form of an ordinary differential equation running from time $t = t_0$ to time $t = t_1$, with $\mathbf{z}_{t_0} = \mathbf{u}$ and $\mathbf{z}_{t_1} = \mathbf{x}$. In this work, we will focus on the former one, describing the transformation of the flow following Eq. (4.4) as a composition of simpler discrete transformations. The number K of composed sub-flows results only in an $\mathcal{O}(K)$ growth in computational complexity, which is a pleasant trade-off for gaining expressiveness.

Depending on the application at hand, it is more convenient to implement the transformation blocks T_k if one desires to generate samples, as one wants to compute $T(\mathbf{u}) = \mathbf{x}$ after generating $\mathbf{u} \sim p_{\mathbf{u}}(\mathbf{u})$, or it can be more useful to implement T_k^{-1} when interested in evaluating the density of a sample $\mathbf{x}$ to apply Eq. (4.2). Either transformations, denoted generalized as f_{ϕ_k}, will be constructed using a neural network with parameters ϕ_k. The transformation has to be invertible and has to have a tractable Jacobian determinant. Even a network which makes the transformation guaranteed to be invertible theoretically might, in practice, be too expensive or intractable to compute exactly. Additionally, if both T and T^{-1} are needed, one should take into account that the design of f_{ϕ_k} would allow to compute the two operations plus their Jacobian determinants efficiently.

When speaking about tractable Jacobian determinant, it means that the operation should be at most $\mathcal{O}(D)$, since all the Jacobian determinants are computable at time $\mathcal{O}(D^3)$. Constructing the transformations and the architecture of the neural networks in certain ways can ensure this computational cost for the Jacobian determinant. Two choices typically used in the community are either autoregressive networks [120] and coupling layers [112, 121]. In this work we will focus on the first ones, as, even though they are less efficient compared to coupling layers, they offer more expressiveness.

To simplify the notation, the dependency on k is dropped for the transformation, denoting it simply by f_ϕ. Additionally the input of the transformation is $\mathbf{z}$, while its output is $\mathbf{z}'$, regardless of implementing T or T^{-1}.

Autoregressive flows are a direct implementation of decomposing the density $p_{\mathbf{x}}(\mathbf{x})$ into conditional densities $p(x_i|\mathbf{x}_{<i})$. This specifies a componentwise transformation of the following form [122, 123]:

$$z_i' = \tau(z_i; \mathbf{h}_i) \text{ with } \mathbf{h}_i = c_i(\mathbf{z}_{<i}; \phi), \tag{4.8}$$

where z_i' is the i-th component of $\mathbf{z}'$ and z_i the i-th of $\mathbf{z}$. τ is the transformer, which is a one-dimensional diffeomorphism with respect to z_i with parameters $\mathbf{h}_i$. c_i is the i-th conditioner which takes as input $\mathbf{z}_{<i} = (z_1, z_2, \ldots, z_{i-1})$, i.e., the previous components of $\mathbf{z}$, and is a a neural network with parameters ϕ. The conditioner provides the parameters $\mathbf{h}_i$ of the i-th transformer of z_i depending on the previous components $\mathbf{z}_{<i}$, defining implicitly a conditional density over z_i with respect to $\mathbf{z}_{<i}$.

Notice that all parameters $\mathbf{h}_i$ given by the conditioners c_i can be computed simultaneously in parallel since they are independent. Additionally, it is easy to see that the Jacobian of the transformation $f_\phi = (c_1, c_2, \ldots, c_D)$ is triangular,

$$J_{f_\phi}(\mathbf{z}) = \begin{bmatrix} \frac{\partial\tau}{\partial z_1}(z_1; \mathbf{h}_1) & & \mathbf{0} \\ & \ddots & \\ \mathbf{L}(\mathbf{z}) & & \frac{\partial\tau}{\partial z_D}(z_D; \mathbf{h}_D) \end{bmatrix}, \tag{4.9}$$

making the Jacobian determinant tractable. The computation of the log-absolute-determinant of $J_{f_\phi}(\mathbf{z})$ can be computed in an $\mathcal{O}(D)$ time:

$$\log|\det J_{f_\phi}(\mathbf{z})| = \log\left|\prod_{i=1}^{D} \frac{\partial\tau}{\partial z_i}(z_i; \mathbf{h}_i)\right| = \sum_{i=1}^{D} \log\left|\frac{\partial\tau}{\partial z_i}(z_i; \mathbf{h}_i)\right|.$$

The lower-triangular part of the Jacobian, $\mathbf{L}(\mathbf{z})$, is irrelevant to compute the determinant, which for triangular matrices like in Eq. (4.9) depends only on the diagonal elements. The derivative of the transformer can have either an analytical form or can be computed by automatic differentiation [80], the technique used to perform backpropagation.

In theory, an autoregressive flow is an universal approximator [122, 123], provided the transformer and the conditioner are flexible enough to approximate any function arbitrarily well.

To ensure the property of being a diffeomorphism, transformers are usually taken as monotonic differentiable functions. Affine autoregressive flows present a simple choice for the transformers and have been extensively used [112, 121, 124–126]. The transformer is an affine function:

$$\tau(z_i; \mathbf{h}_i) = \alpha_i z_i + \beta_i \text{ where } \mathbf{h}_i = \{\alpha_i, \beta_i\}.$$

Notice that for $\alpha_i \neq 0$ it is invertible, which can be guaranteed, e.g., by taking $\alpha_i = \exp\tilde{\alpha}_i$, where $\tilde{\alpha}_i$ is unconstrained (making $\mathbf{h}_i = \{\tilde{\alpha}_i, \beta_i\}$). Since the derivative of the transformer with respect to z_i is α_i, the log absolute Jacobian determinant is:

$$\log|\det J_{f_\phi}(\mathbf{z})| = \sum_{i=1}^{D} \log|\alpha_i| = \sum_{i=1}^{D} \tilde{\alpha}_i.$$

Affine transformers present simple and analytically tractable transformations, but have limited expressiveness. Other more complex transformers have been suggested, such as monotonic neural networks [122, 127, 128], sum-of-squares polynomials [123] or monotonic splines [129, 130].

In Sec. 4.2 a concrete implementation of monotonic splines will be presented, the neural spline flows [131], which will be used through the thesis. Afterwards, in Sec. 4.3, a typical implementation of how one can compute the parameters $\mathbf{h}_i$ of the conditioners c_i in parallel in a single forward pass of the neural network using masked autoregressive networks is laid out.

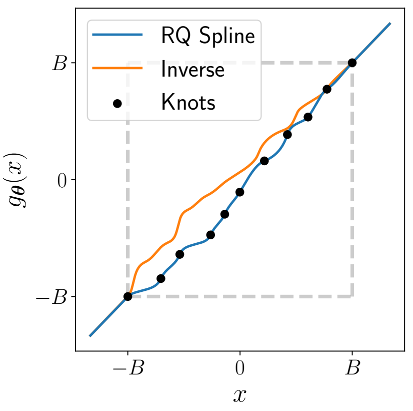
4.2 Neural spline flows

Non-affine transformers such as monotonic neural networks or sum-of-squares polynomials are arbitrarily flexible, but do not possess an analytic inverse. This forces the usage of numerical techniques, e.g., bijection search, to calculate the inverse, which in this specific case, to find the inverse with an accuracy ε, takes $\mathcal{O}(\log \frac{1}{\varepsilon})$ iterations, carrying a trade-off between accuracy and computational cost.

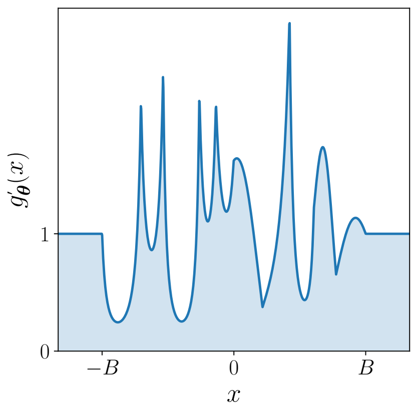
To overcome this problem, transformers in the form of monotonic splines were proposed [129]. A spline [132] consists of a piecewise function of K segments, where each of these segments is an easy to invert function.

In their work on neural spline flows (NSF) [131], Durkan *et al.* advocate for utilizing monotonic rational-quadratic splines as transformers τ, which are easily differentiable, more flexible than previous attempts of using polynomials for these spline segments (since the Taylor-series expansion of a rational quadratic polynomial is infinite) and are analytically invertible.

Each monotonic rational-quadratic function in the splines is defined by a quotient of two quadratic polynomials. In particular, the splines map the interval $[-B, B]$ to $[-B, B]$, and outside of it the identity function is considered. The splines are parameterized following Gregory and Delbourgo [133], where K different rational-quadratic monotonic and differentiable functions are used, with boundaries set by the pair of coordinates $\{(x^{(k)}, y^{(k)}\}_{k=0}^{K}$, known as knots of the spline and are the points where it passes through. Note that $(x^{(0)}, y^{(0)}) = (-B, -B)$ and $(x^{(K)}, y^{(K)}) = (B, B)$. Additionally, $K - 1$ intermediate positive derivative values $\{\delta^{(k)}\}_{k=1}^{K-1}$ need to be defined, since the boundary points derivatives are set to 1 to match the identity function ($\delta^{(0)} = \delta^{(K)} = 1$). Figure 4.2 shows, in (a), an example of a rational-quadratic



(a) Rational-quadratic spline function.



(b) Rational-quadratic spline derivative.

Fig. 4.2: Rational-quadratic splines transformer example. In (a), the rational-quadratic function defined by the knots and derivatives is shown, which is monotonic. Additionally its inverse is shown. In (b), the derivative is shown, displaying the expressiveness and flexibility this kind of transformers have. Source: [131].

spline passing through the defined knots, as well as its inverse; in (b), the derivative of the function is displayed, demonstrating the flexibility and expressiveness that these transformers can have for a monotonic function.

With the description of the splines at hand, the conditioner $c_i(\mathbf{z}_{<i}; \phi)$ in the form of a neural network of parameters ϕ has as an output a vector $\mathbf{h}_i = [\mathbf{x}, \mathbf{y}, \delta]$, computed as:

- $\mathbf{x}$: a vector of length K given the x coordinates of the knots for the spline between $-B$ and B. For this, an analogous vector $\mathbf{x}'$ of length K is computed by the network with unbounded values in $\mathbb{R}$, which is processed through a softmax operation (Eq. (3.6)) and multiplied by $2 \cdot B$ to define the bin width of the points:

$$\mathbf{x} = -B + \textsc{cumsum}\left[2 \cdot B \cdot \textsc{softmax}(\mathbf{x}')\right]. \tag{4.10}$$

- $\mathbf{y}$: analogous to $\mathbf{x}$, the y coordinates of the K knots are found, after computing an unbounded vector $\mathbf{y}'$ of length K with the network, following the same operation as Eq. (4.10).
- δ: a positive vector of length $K-1$ computed after applying SoftPlus (Eq. (3.5)) to an unbounded vector given by the network in order to impose its positivity.

The monotonic, continuously-differentiable, rational-quadratic spline in the k-th bin, defining $s^{(k)} = (y^{(k+1)} - y^{(k)})/(x^{(k+1)} - x^{(k)})$ and $\xi(x) = (x - x^{(k)})/(x^{(k+1)} - x^{(k)})$, can be written in terms of ξ as:

$$\frac{\alpha^{(k)}(\xi)}{\beta^{(k)}(\xi)} = y^{(k)} + \frac{\left(y^{(k+1)} - y^{(k)}\right)\left[s^{(k)}\xi^2 + \delta^{(k)}\xi(1-\xi)\right]}{s^{(k)} + \left[\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}\right]\xi(1-\xi)}, \tag{4.11}$$

where $\alpha^{(k)}(\xi)$ and $\beta^{(k)}(\xi)$ are quadratic polynomials of ξ and, hence, of x. Since the splines are defined componentwise with respect to z_i, the logarithm of the absolute Jacobian determinant can be computed as sum of logarithms of the derivative of Eq. (4.11) with respect to x:

$$\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{\alpha^{(k)}(\xi)}{\beta^{(k)}(\xi)}\right] = \frac{\left(s^{(k)}\right)^2\left[\delta^{(k+1)}\xi^2 + 2s^{(k)}\xi(1-\xi) + \delta^{(k)}(1-\xi)^2\right]}{\left[s^{(k)} + \left[\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}\right]\xi(1-\xi)\right]^2}.$$

Additionally, the inverse of a rational-quadratic can be computed analytically by inverting Eq. (4.11). The solution, as presented in [131], is given by $\xi(x) = 2c/(-b - \sqrt{b^2 - 4ac})$, where

$$\begin{aligned}
a &= \left(y^{(k+1)} - y^{(k)}\right)\left[s^{(k)} - \delta^{(k)}\right] + \left(y - y^{(k)}\right)\left[\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}\right], \\
b &= \left(y^{(k+1)} - y^{(k)}\right)\delta^{(k)} - \left(y - y^{(k)}\right)\left[\delta^{(k+1)} + \delta^{(k)} - 2s^{(k)}\right], \\
c &= -s^{(k)}\left(y - y^{(k)}\right),
\end{aligned}$$

and can be used to determine x given y. Because the transformation is monotonic, one can always determine which quadratic root is correct. The bin is determined via binary search.

4.3 Masked autoregressive networks

The D conditioners c_i of the transformers in Eq. (4.8) (which in the case of the NSF in Sec. 4.2 give the knot positions and the derivative in the intermediate knots as these are the parameters of the transformer) can be any function of $\mathbf{z}_{<i}$. Implementing them as separated neural networks of input $\mathbf{z}_{<i}$ and output $\mathbf{h}_i$ is quite naive, since it would scale poorly in the dimension D, requiring D forward propagation of vectors with average size $D/2$, as well as training and storing the parameters of D neural networks. This can become quickly computationally prohibitive, as the dimension of the data can be of the order of hundreds, thousands or even millions when considering videos or high-resolution images.

There are several ways to get around this, such as using recurrent neural networks [134,135] over the components, where the internal hidden state is used as an input variable to a standard neural network [136–138], or, as mentioned already earlier, coupling layers [112, 121]. In this work, the implementation chosen was the masked autoregressive network [120], which led to masked autoregressive flows.

The concept of a masked autoregressive network is to take as input the whole vector $\mathbf{z}$ and to give as an output all the parameters of the D conditioners $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_D)$ at once. To do so, one takes an arbitrary feed forward network, such as the one presented in Sec. 3.2.3 and removes the connection between the neurons until there is no path from the input z_i to the outputs $(\mathbf{h}_1, \ldots, \mathbf{h}_i)$. A direct way to perform this connection cut is by multiplying, component to component, the weight matrices which connect the neurons of the neural network by a binary matrix of the same size. This binary matrix "masks out" the connections by making them effectively zero, while maintaining all other connections intact.

As proposed in Germain *et al.* [120], to form an appropriate matrix, one assigns to each input, hidden and output neuron of the neural network a degree between 1 and D. For the input $\mathbf{z}$, each component has simply a degree according to its index. For the output, each neuron corresponding to the vector $\mathbf{h}_i$ has a degree of i. The hidden layer neurons can have randomly assigned degrees, but to ensure a more distributed layout and that all degrees have a meaningful number of connections, one can assign the degree in a sequential way to the hidden units in a layer. The binary mask matrix for the layer l

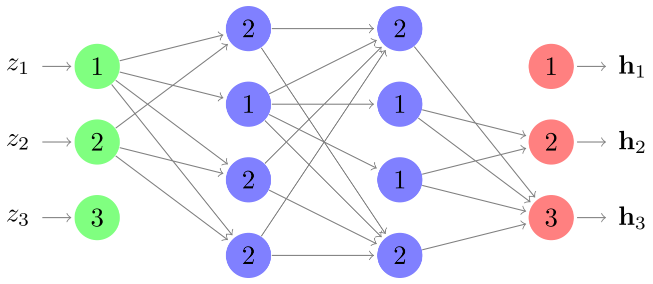


Fig. 4.3: Example of a masked autoregressive network. The numbers indicate the degrees assigned to each neuron, and they are connected following Eqs. (4.12) and (4.13), with the i-th output depending only on the inputs $1, \ldots, i-1$.

($l = 0$ is the input layer), corresponding to the weight matrix W^l, is then constructed as follows for all layers except the output layer:

$$M^{W^l}_{k',k} = 1_{m^l(k') \geq m^{l-1}(k)} = \begin{cases} 1 & \text{if } m^l(k') \geq m^{l-1}(k) \\ 0 & \text{otherwise.} \end{cases}, \tag{4.12}$$

where $m^l(k)$ is the degree of the k-th neuron in the l-th layer. For the output layer L, to ensure that the output with degree i has only connections with lower degrees, the last binary matrix is given by:

$$M^{W^L}_{k',k} = 1_{m^L(k') > m^{L-1}(k)} = \begin{cases} 1 & \text{if } m^L(k') > m^{L-1}(k) \\ 0 & \text{otherwise.} \end{cases}. \tag{4.13}$$

When multiplying these matrices M^{W^l} with their corresponding weight matrices W^l, one creates an autoregressive network where the outputs with degree i only depend on the inputs with degree $< i$. Figure 4.3 shows an example of a masked autoregressive network to compute all conditional parameters $\mathbf{h}_i$ in one forward pass by assigning degrees to each neuron and masking out connections according to Eqs. (4.12) and (4.13).

Compared to other flows, masked autoregressive flows come with two main advantages. On one hand, they are efficient to evaluate, as all transformer parameters $(\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_D)$ are evaluated in one neural network pass. On the other hand, given a flexible enough conditioner and form of the transformer, they are universal approximators, i.e., they can represent any autoregressive transformation under the conditions mentioned in Sec. 4.1. The main disadvantage is, when inverting the flow, one has to evaluate the network D times, since the parameters for inverting the i-th component are only available when having the parameters for all the components of index $< i$. Despite this computational difficulty, the expressiveness of masked autoregressive flows makes them highly utilized, forming part of many popular flows [122, 124, 125, 127].

Notice how by constructing a sequence of masked conditioners where one uses as input $\mathbf{z}_{<i}$ for the i-th conditioner, one would always construct the a sequential relation for all the $i = 1, \ldots, D$ components, making the later one richer and the first relations poorer, specially for $i = 1$, which has no condition and would be a fixed density. To ensure that all the input variables interact with each other, it is usual to randomly permute the dimensions in between transformations of the normalizing flow in Eq. (4.3). Permutations are invertible linear transformations, with absolute determinant equal to 1, adding a trivial term to the log absolute Jacobian determinant of the transformation. The inverse is also easy to compute and can be stored, as these transformations do not change along the training of the flow. Permutations were generalized by Oliva *et al.* [139] to a more general class of linear transformations, with a matrix W parameterized in terms of its LU-decomposition $W = PLU$, with P a fixed permutation matrix, L a lower triangular with ones on the diagonal, and U an upper triangular. If one restricts the diagonal elements of U to be positive, W is guaranteed to be invertible. These linear transformations lead to a rich interaction between the components of $\mathbf{z}$ in each flow step.

5. GRAPH NEURAL NETWORKS

So far through this work, feed forward neural networks were considered, which work on tabular input and tabular output. Constructing the networks with this structure was due to building an analogy with respect to the brain. Since the appearance of neural networks, however, new ways of building the shapes and architectures of the interconnections of the neurons within neural networks have been developed to make full use of the structure of the data. For instance, to analyze and extract information from sequential data such as time-series predictions [140, 141], speech synthesis (i.e., artificial production of human speech) [142], machine translation [137], recurrent neural networks (RNNs) [134, 135] were built, which exhibit a sequential dynamic behavior capturing the underlying trend of the data. Another popular type of data are visual images, which gave rise to convolutional neural networks (CNNs) [143], characterized by having shared-weights architecture and translation invariance characteristics due to laying images as a regular grid in the Euclidean space. Popular applications include image recognition [144, 145], video analysis [146] and natural language processing (i.e., the analysis and interpretation of human language) [147]. Both sequential and imagery data could have been flatten into a one dimensional vector to apply a feed forward neural network. However, by developing new kinds of network architectures more befitting to the natural structure of the data, the results improved vastly.

In this chapter we will focus on graph neural networks, neural networks that revolve around data structured as a mathematical graph. A general introduction will be shown in Sec. 5.1, while Sec. 5.2 will develop the algorithm of GraphSAGE, a particular implementation of graph neural networks.

5.1 A general introduction

There is an increasing amount of cases where data is represented by a non-Euclidean domain, in the form of graphs with complex relationships and interdependencies between objects. This spawns a large range of applications: modeling the molecules with their bioactivity in chemistry to discover drugs, relationship between e-customers to recommend products, or the labeling of hits read out by a physics detector (as will be discussed in Chapter 8), among many others. Graphs carry additional complexity when compared to other structured data such as images, as they may be irregular, can have a variety of unordered number of nodes, and the nodes of the graph may have an arbitrary number of neighbors, making it difficult to design operations such as convolutions over them.

From the formal point of view, a graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of vertices or nodes, and $\mathcal{E}$ is the set of corresponding edges. Let $v_i \in V$ be a node of the graph and let $e_{ij} = (v_i, v_j) \in \mathcal{E}$ be an edge pointing from v_i to v_j. The neighborhood of a node v is defined as $N(v) = \{u \in \mathcal{V} | (v, u) \in \mathcal{E}\}$. The adjacency matrix A is an $n \times n$ matrix, where n is the number of nodes, with $A_{i,j} = 1$ if $e_{ij} \in E$, while $A_{i,j} = 0$ if $e_{ij} \notin E$. An attributed graph is a graph where a node v has d attributes or features in the form of a vector $\mathbf{x}_v \in \mathbb{R}^d$, forming a matrix $X \in \mathbb{R}^{n \times d}$. They may also have c edge attributes $X^e \in \mathbb{R}^{m \times c}$, where m is the total number of edges, and each row is an edge feature vector $\mathbf{x}^e_{v,u} \in \mathbb{R}^c$ of the edge $e = (v, u)$. If all edges of a graph are directed from one node to another, the graph is a directed graph. An undirected graph is a particular case of a directed graph, where there is a pair of edges with inverse direction if two nodes are connected, implying a symmetric adjacency matrix. If the node attributes change dynamically over time, the graph is considered a spatial-temporal graph $\mathcal{G}^{(t)} = (\mathcal{V}, \mathcal{E}, X^{(t)})$.

Figure 5.1 shows an example relating the feature matrix plus adjacency matrix of an undirected graph to its actual representation. Another example of a graph would be a social network, with the nodes being the users, the edges being connections within that network, and the node features containing the user's information, such as age, location, etc.

To deal with graph structured data, graph neural networks (GNNs) were developed, first outlined by Gori *et al.* (2005) [148] and further developed by Scarselli *et al.* (2009) [149], and Gallicchio *et al.* (2010) [150]. These first

$$X = \begin{pmatrix} x_1^1 & x_1^2 & x_1^3 \\ x_2^1 & x_2^2 & x_2^3 \\ x_3^1 & x_3^2 & x_3^3 \\ x_4^1 & x_4^2 & x_4^3 \end{pmatrix}$$

$$+$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

$$\Longleftrightarrow$$

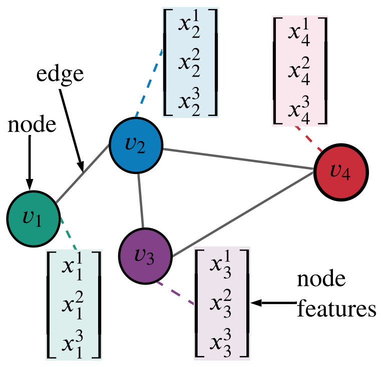


Fig. 5.1: Example of an undirected attributed graph of four nodes, each with three attributes. On the left, the node feature matrix X with the features $\mathbf{x}_i$ corresponding to the nodes v_i plus the adjacency matrix A are defined. On the right, the visual representation of the graph structure and information is shown.

models fall under the category of recurrent graph neural networks (RecGNNs), pioneers of future GNNs, with the aim to learn node representation via a recurrent neural architecture. They revolve around assuming that nodes exchange information/messages with their neighbors until a stable equilibrium is found. Inspired by the concept of message passing and CNNs, convolutional graph neural networks (ConvGNNs) operate by generalizing the convolution from grid data to graph data. They can be broadly categorized into two branches, with the first one being spectral-based approaches initiated by Bruna *et al.* (2013) [151], which operate over the spectral eigendecomposition of the graph Laplacian matrix. The other category would be the spatial-based ConvGNNs, first introduced by Micheli (2009) [152], where the convolutional operation is based on a node's spatial relations. GraphSAGE (2017) [153], which will be seen in Sec. 5.2, is also a spatial-based ConvGNN. Graph autoencoders (GAEs) are unsupervised learning algorithms to encode nodes or graphs into a latent vector space, which is later used to decode and reconstruct the graph. The focus lies mainly on two applications: network embedding, e.g., Cao *et* *al.* (2016) [154], where GAEs learn a latent node representation to reconstruct graph information, such as the adjacency matrix; and graph generation, as shown by Li *et al.* (2018) [155], focused on generating nodes and edges of a graph step by step, or the whole graph at once. Lastly, spatial-temporal graph neural networks (STGNNs) enable to learn hidden patterns from spatial-temporal graphs by considering both spatial and temporal dependencies at the same time. STGNNs have seen applications for example in traffic speed forecasting [156] or human action recognition [157].

Given a graph structure and node information, GNNs' outputs can be operated and used for different tasks:

- Node-level: outputs are related to performing node regression or classification. To do so, RecGNNs and ConvGNNs extract high-level node representation which is vectorized and passed to a feed forward network to perform regression or classification.

- Edge-level: outputs are related to edge classification and link prediction. Given two node representations by GNNs, a feed forward neural network is able to perform label prediction or measurement of the connection strength of an edge.

- Graph-level: outputs related to graph regression and classification after obtaining a compact representation on graph level by a GNN. Information related to the graph as a whole, not to individual nodes or connections, is extracted.

Graph neural networks have become steadily popular due to the variety of data that can be represented by graphs and the potential and generalization of operations one can perform on them. Zhou *et al.* [158] and Wu *et al.* [159] perform great reviews on the topic, giving a clean overview on the current state of GNNs.

In high energy physics, graph neural networks have become a well-received choice for many implementations due to the abstraction of relational data [160]. Applications include, but are not limited to, jet [161–163] and event [164,165] classification, calorimeter [166], particle flow [167] and secondary vertex [168] reconstructions.

5.2 The GraphSAGE algorithm

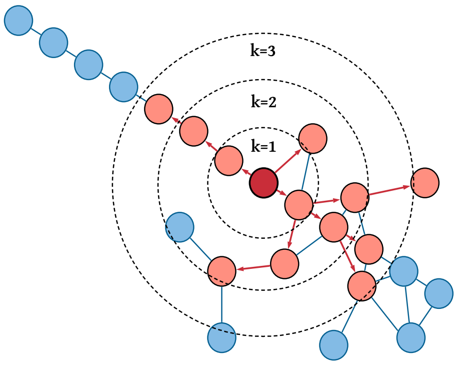
Initial RecGNNs and ConvGNNs to generate node embedding were mainly transductive, i.e., they directly optimize embeddings for each node using matrix-factorization-based objectives, which does not generalize to unseen data easily since they make predictions for nodes in a single, fixed graph. These approaches are useful to label data or quantify magnitudes on fixed graphs, where the structure does not change and there is no interest in making inference on different graphs, as they can specialize exactly to the graph at hand. When dealing with dynamic graphs, where nodes and edges can appear or disappear, or when considering utilizing the model for new graphs, inductive models are needed. In an inductive model, when producing node embedding, the algorithm has to generalize newly observed subgraphs to the node embeddings. It has to recognize the structural properties of a node's neighborhood, revealing the node's local role in the graph as well as its global position.

Hamilton *et al.* (2017) presented one of the first variations of inductive node embedding via a spatial based ConvGNN, GraphSAGE (SAmple and aggreGatE) [153], leveraging node features to learn generalizable embedding functions applicable to unseen nodes. The algorithm incorporates node features and simultaneously learns the topological structure of each node's neighborhood while taking into account the distribution of node features in the neighborhood.

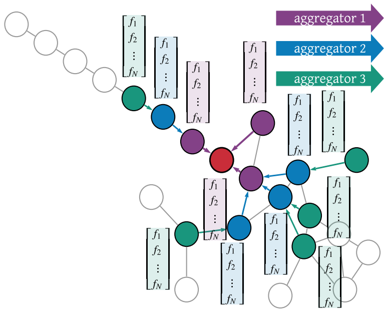
GNNs up to that point trained distinct embedding vectors for each node. GraphSAGE changed the paradigm by learning a set of shared aggregator functions from a node's local neighborhood, depicted in Fig. 5.2. Each aggregator function learns to join information at different numbers of hops[1]/search depths away from a given node. When performing inference, the new local neighborhood is constructed and the information is assembled using the learned aggregator functions.

The details of the GraphSAGE algorithm to generate the embeddings of an undirected graph using neighborhood information are described in Algorithm 1, also known as the forward propagation of the algorithm. To

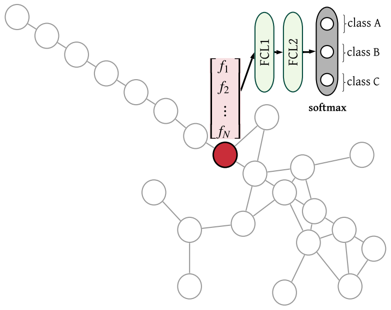
[1] The number of hops indicates the number of nodes a path would cross from the starting node, without counting it. E.g., a number of hops of one would include all the direct neighbors of the starting node, while a number of hops of two would include the nodes which are neighbors of neighbors of that starting node.



(a) Neighborhood sampling of depth $k = 3$ for the red node given the graph drawn. Orange nodes belong to the local neighborhood while blue ones do not.



(b) Aggregator function representation at different neighbor depths.



(c) Example of using the aggregated function (red vector) to a fully-connected network to predict the class of the node.

Fig. 5.2: Visual illustration of the GraphSAGE sample and aggregate approach with a depth of three. Source: [19].

compute the embeddings, K aggregator functions (denoted by AGGREGATE$_k$, $k = 1, \ldots, K$) as well as a set of weight matrices $\mathbf{W}^k$, $k = 1, \ldots, K$ are needed, which propagate the information between the different layers or neighbor depths of the model. A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with feature $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$ in the nodes and a neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$ mapping the nodes to its neighbors are provided. In each step k, the node aggregates information from their local neighbors, storing it in an embedding $\mathbf{h}^k$ and, due to iteration, it gains information from a neighbor at distance k. For this, a node $v \in \mathcal{V}$ aggregates its direct neighbor's information at the previous step $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$ in a single vector $\mathbf{h}_{\mathcal{N}(v)}^{k-1}$. For $k = 0$, the initial node features are taken as their information. The aggregated information is then concatenated to the current local information $\mathbf{h}_v^{k-1}$ and fed to a fully connected layer with a nonlinear activation function σ, transforming them into the node information (the embedding) for the next iteration $\mathbf{h}_v^k$. The final node embedding is denoted by $\mathbf{z}_v \equiv \mathbf{h}_v^K$.

In practice, a random subset of the graph is taken to perform mini-batch computation during the training of the aggregators AGGREGATE$_k$ and weight matrices $\mathbf{W}^k$. Additionally, Hamilton *et al.* proposed to use a maximum number of neighbors instead of the full set of neighbors available as shown in Algorithm 1, where the members of this set are sampled uniformly from the neighbors of the node if the neighbor number is higher than the maximum. This measurement is taken in order to consider possible large variations on the number of neighbors, which can lead to excessive computational operations. In the application concerned in this work (Chapter 8), the number of neighbors is low, having a maximum of 26 neighbors, hence all neighbors are considered to obtain maximum neighbor information available.

Regarding the aggregator architectures, compared to operations on N-dimensional lattices (e.g., sentences, images, 3-D volumes), a node's neighbors have no natural ordering. Additionally, the number of neighbors can be arbitrary. Thus, aggregators have to be functions over an unordered set of arbitrary length of vectors, i.e., the function has to have an arbitrary number of inputs and has to be symmetric with respect to them (i.e., invariant to permutations of its inputs). In their work, Hamilton *et al.* examine three aggregators:

- **Mean aggregator**: One of the most direct operations satisfying the

Algorithm 1: GraphSAGE [153] embedding generation (i.e., forward propagation) algorithm

```
Input  : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$;
         weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity $\sigma$;
         differentiable aggregator functions
         $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function
         $\mathcal{N} : v \to 2^{\mathcal{V}}$
Output: Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$
1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6     end
7     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 end
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

above properties is computing simply the elementwise mean of the vectors in $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$. This operation is nearly equivalent to the convolution propagation rule in transductive ConvGNNs [169]. As in [153], lines 4 and 5 of Algorithm 1 can be replaced by:

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}),$$

transforming the mean-based aggregator to a linear approximation of a localized spectral convolution, leading to significant gains in performance.

- **LSTM aggregator**: recurrent neural networks, and specifically LSTM architectures [134], have a large expressive capacity to extract underlying information from a set of ordered vectors through their hidden state. However, since the vectors have to be processed in a sequential manner, the operation is not symmetric as would be desired for an aggregator.

One way of mitigating this flaw is by introducing the neighbor nodes each time in a random order so that the LSTM cannot learn any sequence, only a joint underlying information.

- **Pooling aggregator**: In a pooling approach, neighbor embeddings are fed independently into a fully-connected neural network, after which they follow a transformation: an elementwise max-polling operation:

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\}),$$

  where max denotes the elementwise max operator and σ is a nonlinear activation function. The pooling weight matrix $\mathbf{W}_{\text{pool}}$ is learned during training.

Notice how the mean aggregator has no learnable parameters, while the LSTM aggregator is not strictly symmetric. The pooling aggregator is the only one satisfying the desired properties while having the flexibility of a neural network.

The final embeddings obtained by GraphSAGE are encodings of the aggregated information of the nodes with their neighbors at distance k, and can be utilized for a variety of tasks, such as clustering, information compression, etc. One particular task to be performed with the embeddings is to classify the nodes, using the embeddings as an input for a secondary feed-forward neural network, as depicted in Fig. 5.2c. The results of GraphSAGE for such a task will be thoroughly discussed in Chapter 8.

6. LIKELIHOOD-FREE INFERENCE OF EXPERIMENTAL NEUTRINO OSCILLATIONS USING NEURAL SPLINE FLOWS

Adapted from the original publication in Physical Review D 101, 113001 on the 2nd of June 2020 [17].

In the current state of the determination of neutrino oscillation parameter (introduced in Chapter 2), all mixing angles have been measured [170], as well as the two mass differences between the three mass neutrino eigenstates. The remaining mixing matrix parameters to be measured are the imaginary phase responsible for the CP violation and the sign of one of the two neutrino mass splittings, which determines the so-called hierarchy. Both measurements are at reach for current and near future oscillation experiments.

Statistical methods used in the neutrino oscillation analysis so far comprise both frequentist and Bayesian approaches [45, 171]. There are, however, some limitations to these methods. Both approaches are based as of today on a binned likelihood algorithm which might limit the sensitivity of the experiment and some of them impose a Gaussian dependency in some of the nuisance parameters affecting the precision of the results and correctness of the evaluated uncertainties. Additionally, they require very intensive central processing unit (CPU) time, which is a limiting factor that reduces the flexibility of the statistical analysis and checks, and introduces strong constraints on the delivery of the results. These limitations are derived from the intrinsic difficulties of the statistical data analysis that are depicted in Sec. 6.1, using the T2K experiment as a reference example.

In this chapter we propose an alternative statistical method to overcome some of the limitations of the current methods in use. The proposed procedure is based on an unbinned likelihood inference using neural density estimators.

This method has the potential of being accurate, fast and to reduce the possible bias due to the intrinsic binning in other approaches. Neural spline flows (see Sec. 4.2), the implementation of neural density estimators we chose, have also some advantages since the Gaussian generator intrinsic to the method will facilitate the introduction of experimental errors in the distributions. We will discuss in this chapter the basic concepts of the method and show the potential with a simplified example.

6.1 Problem definition and physical simulator

Neutrino oscillation experiments search for the modification of the flavour content of a neutrino beam travelling in vacuum or matter for a certain distance. Beams are normally characterized at a near site, where the neutrino energy spectrum and flavour composition are not yet altered by oscillations. The same beam is sampled after a certain flight distance L. The change on the flavour composition can be determined in two different ways which are as follows:

(i) The neutrino flavour disappearance ($P(\nu_\alpha \to \nu_\alpha)$) experiments search for the disappearance of a certain neutrino flavour as a function of the neutrino energy. The disappearance produces both a reduction in the flux of neutrinos of a given flavour and the distortion of the neutrino energy spectra that is observed in the distribution of the measured quantities. For example, T2K uses the muon momentum (p_μ) and the angle with respect to the neutrino direction (θ_μ).

(ii) The neutrino flavour appearance ($P(\nu_\alpha \to \nu_\beta), \alpha \neq \beta$), experiments search for the appearance of a neutrino flavour that is normally suppressed in the original neutrino flux. In T2K, this new flavour is the electron neutrino. The dependency of the oscillation with the neutrino energy is inferred from the momentum and the angle with respect to the neutrino direction of the electron ejected in the interaction of neutrinos with matter.

The neutrino flavour is determined by the flavour of the charged lepton (muon, electron or tauon) produced in charged current interactions of neutrinos with the nuclei. For the current analysis, we will concentrate on the disappearance phenomenon (i).

In a synthetic way, the experimental number of observed neutrinos with observed properties ($\vec{\theta}_\nu^{\text{reco}}$) can be described by:

$$N_{\text{evts}}^{\text{near}}(\vec{\theta}_\nu^{\text{reco}}) = \int \sigma(E_\nu)\phi^{\text{near}}(E_\nu)P_{\text{near}}(\vec{\theta}_\nu^{\text{reco}}|E_\nu)dE_\nu + \text{Back}_{\text{near}}(\vec{\theta}^{\text{reco}})$$

for the near detector and

$$N_{\text{evts}}^{\text{far}}(\vec{\theta}_\nu^{\text{reco}}) = \int \sigma(E_\nu)\phi^{\text{far}}(E_\nu)P_{\text{far}}(\vec{\theta}_\nu^{\text{reco}}|E_\nu)P_{\text{osc}}(E_\nu)dE_\nu + \text{Back}_{\text{far}}(\vec{\theta}^{\text{reco}})$$

for the far detector. The number of observed neutrinos depends on the cross section ($\sigma(E_\nu)$), the neutrino flux ($\phi^{\text{far,near}}(E_\nu)$), the probability of observing the experimentally accessible quantities ($\vec{\theta}_\nu^{\text{reco}}$) given a neutrino energy ($P_{\text{far,near}}(\vec{\theta}_\nu^{\text{reco}}|E_\nu)$), the oscillation probability ($P_{\text{osc}}(E_\nu)$) and the backgrounds observed in the detectors ($\text{Back}_{\text{near,far}}(\vec{\theta}^{\text{reco}})$).

The experimental challenge comes from inferring the neutrino energy, E_ν, given the experimental observable ($\vec{\theta}_\nu^{\text{reco}}$). The term $P_{\text{far,near}}(\vec{\theta}_\nu^{\text{reco}}|E_\nu)$ encapsulates not only the detector resolution, but also the neutrino-nucleus cross section model predictions and uncertainties. Other difficulties raise from the limited knowledge ($\approx$9% in the latest T2K results [172]) of the neutrino flux ($\phi^{\text{far,near}}(E_\nu)$) and of neutrino cross sections as function of the energy ($\sigma(E_\nu)$). The background terms ($\text{Back}_{\text{far,near}}(\vec{\theta}^{\text{reco}})$) are normally relevant ($\approx$20% in T2K) [173] and they subsequently depend on the neutrino-nucleus cross sections in a nontrivial manner.

Both the frequentist and the Bayesian statistical approaches [45,171] utilize the near detector data to predict the probability density function at the far detector in the absence of oscillations,

$$f(\vec{\theta}_\nu^{\text{reco}}|E_\nu) = \sigma(E_\nu)\phi^{\text{far}}(E_\nu)P_{\text{far}}(\vec{\theta}_\nu^{\text{reco}}|E_\nu).$$

Once determined, the conditional probability density function $f(\vec{\theta}_\nu^{\text{reco}}|E_\nu)$ can be used to determine the oscillation parameters ($P_{\text{osc}}(E_\nu)$) by comparing it to the far detector events. Most of the experimental effort is actually devoted to the determination of this conditional probability which depends also on a large number of hidden and correlated parameters describing uncertainties in detector performances, cross section models and the neutrino flux. Hidden parameters are marginalized or profiled in the analysis, providing the experimental result for oscillation parameters as (the posterior in the case of Bayesian approaches) probability maps.

In the particular case of the T2K experiment, the experimentally accessible observables ($\vec{\theta}_\nu^{\text{reco}}$) are the momentum and direction of the μ lepton ($p_\mu^{\text{reco}}, \theta\mu^{\text{reco}}$). Near and far detectors are able to provide also the kinematic of pions (charged and neutral) and protons, or the total released energy in the interaction, but we will ignore these capabilities to simplify the discussion. The μ lepton is produced at the interaction of the neutrino with the target nucleus. In this case the probability density function ($f(\vec{\theta}_\nu^{\text{reco}}|E_\nu)$) can be simplified to $p(p_\mu^{\text{reco}}, \theta_\mu^{\text{reco}}|E_\nu)$. The near detector of the experiment measures the neutrino flux and tunes the model of neutrino-nucleus interaction providing the estimation of the probability density function $p(p_\mu^{\text{reco}}, \theta_\mu^{\text{reco}}, E_\nu) = p(p_\mu^{\text{reco}}, \theta_\mu^{\text{reco}}|E_\nu)p(E_\nu)$ together with the expected number of interactions in the far detector in the absence of oscillations. The near detector also provides a dependency with free parameters in the model and a full error covariance matrix relating all of them. To simplify the exercise, we ignore the error covariance matrix in this study and assume that the probability $p(p_\mu^{\text{reco}}, \theta_\mu^{\text{reco}}, E_\nu)$ is implicitly known to the experiment through simulations.

The neutrino oscillation disappearance probability, introduced in Eq. (2.3) in Sec. 2.2, can be approached by the simplified two-flavour [170] oscillation. The disappearance probability as a function of the initial energy of the neutrino E_ν is

$$\begin{aligned} p_{\text{osc}}(E_\nu, \theta_{\text{mix}}, \Delta m^2) = \\ \sin^2(2\cdot\theta_{\text{mix}})\sin^2\left(1.27\cdot\frac{\Delta m^2\cdot 295}{E_\nu}\right), \end{aligned} \tag{6.1}$$

where 295 is the distance in kilometers between the near and far sites in the T2K experiment and 1.27 is a scaling parameter to adjust the oscillation phase to distance in kilometers. E_ν is the neutrino energy in GeV, θ_{mix} the mixing angle of the two flavours and Δm^2 the difference in mass of the two mass eigenstates in eV^2. θ_{mix} and Δm^2 are the parameters governing the oscillations.

Although the problem was simplified in the following analysis to make a proof of concept, in Sec. 6.4 we outline how the methodology could be applied to the full complex analysis.

6.1.1 Physical simulator

We have simplified the problem to demonstrate the viability of the proposed method to determine the oscillation parameters using neural spline flows. Event samples are generated using the NEUT [174] Monte Carlo (MC) event generator model that describes the interactions of neutrinos with nuclei. We also use a realistic neutrino flux energy spectrum provided by the T2K Collaboration [175]. With both inputs, we generate CCQE events. CCQE is the most probable reaction at T2K energies, and the one dominating the statistical sensitivity of the experiment, where the neutrino transforms into a muon exchanging a neutron into a proton ($\nu + n \to \mu + p$). To simplify, we ignore other reaction channels and potential backgrounds, and also detector effects. The generator provides n-tuples of events weighted according to their probability as a function of neutrino energy and angle and momentum of the muon.

6.2 Methodology

In this section we will explain how the likelihood for Bayesian inference is constructed. In the context of machine learning, likelihood-free inference, as stated in the abstract, refers to the task of performing such analysis when the densities are data-driven, but no explicit likelihood function can be constructed. This is the case for the near detector, where we have data for the different magnitudes but no analytical density available. Therefore, a density estimation for the near detector is performed through a particular implementation of normalizing flows (introduced in Chapter 4), the neural spline flows (see Sec. 4.2), which allow to learn an explicit density from data. This near detector density is then combined with the analytical formula for neutrino oscillation for the far detector in order to obtain the likelihood for the experiment. By doing so, we are combining the potential of NSF with expertise of the particular problem.

We start by estimating the explicit density of the expected energy spectrum E_ν of the neutrinos, together with the momentum p_μ and angle θ_μ of the measured muon without oscillations, obtaining $p(p_\mu, \theta_\mu, E_\nu)$, as measured by the near detector. This is done by learning the density using a NSF from the Monte Carlo data, generated as presented in Sec. 6.1.1. The base density $p_{\mathbf{u}}(\mathbf{u})$ used for Eq. (4.1) is a three-dimensional standard normal distribution.

Having estimated the joint probability $p(p_\mu, \theta_\mu, E_\nu)$ of the initial distribution at the near detector, we need to construct the conditional density $p\left(p_\mu, \theta_\mu | \theta_{\text{mix}}, \Delta m^2\right)$ of the observed magnitudes given the oscillation parameters at the far detector in order to perform Bayesian inference. For this, we simply integrate the probability of not oscillating, $1 - p_{\text{osc}}$, using Eq. (6.1), over the energy spectrum of the joint distribution:

$$p\left(p_\mu, \theta_\mu | \theta_{\text{mix}}, \Delta m^2\right) = C\left(\theta_{\text{mix}}, \Delta m^2\right) \cdot \\ \int p(p_\mu, \theta_\mu, E_\nu)\big(1 - p_{\text{osc}}(E_\nu, \theta_{\text{mix}}, \Delta m^2)\big) dE_\nu,$$

where $C\left(\theta_{\text{mix}}, \Delta m^2\right)$ is a constant of normalization computed after performing the integral. With this we have the probability of observing a single muon with momentum p_μ and angle θ_μ after oscillating given the parameters θ_{mix} and Δm^2.

In order to take into account the number of observed samples, the extended likelihood [176,177] is used, modifying the likelihood with a Poisson count term to consider the expected number of events for a given set of parameters and the actual observed number:

$$L(\boldsymbol{\theta}) = \frac{[\mu(\boldsymbol{\theta})]^n}{n!} e^{-\mu(\boldsymbol{\theta})} \prod_{i=1}^{n} p\left(\mathbf{x}_i | \boldsymbol{\theta}\right).$$

In our case, the Poisson parameter $\mu(\boldsymbol{\theta})$ is obtained by integrating the possible oscillations over all the energy spectrum, scaled to the initial number of particles N_{ini}:

$$\mu\left(\theta_{\text{mix}}, \Delta m^2\right) = N_{\text{ini}} \times \\ \iiint p(p_\mu, \theta_\mu, E_\nu)\Big(1 - p_{\text{osc}}\left(E_\nu, \theta_{\text{mix}}, \Delta m^2\right)\Big) dE_\nu dp_\mu d\theta_\mu.$$

Hence, the extended likelihood we apply for the analysis is

$$L\left(\theta_{\text{mix}}, \Delta m^2\right) = \frac{\left[\mu\left(\theta_{\text{mix}}, \Delta m^2\right)\right]^n}{n!} e^{-\mu\left(\theta_{\text{mix}}, \Delta m^2\right)} \times \prod_{i=1}^{n} p\left(p_\mu^{(i)}, \theta_\mu^{(i)} | \theta_{\text{mix}}, \Delta m^2\right), \tag{6.2}$$

and the posterior for the parameters takes the form of

$$p\left(\theta_{\text{mix}}, \Delta m^2 | \left\{p_\mu^{(i)}, \theta_\mu^{(i)}\right\}_{i=1}^{n}\right) \propto L\left(\theta_{\text{mix}}, \Delta m^2\right) p\left(\theta_{\text{mix}}, \Delta m^2\right), \tag{6.3}$$

with $p\left(\theta_{\text{mix}}, \Delta m^2\right)$ the prior information before observing the events and $\left\{p_\mu^{(i)}, \theta_\mu^{(i)}\right\}_{i=1}^{n}$ the set of observed events.

6.2.1 Reference analysis using an approximate unbinned likelihood

The results of the experiments are validated using an approximate unbinned likelihood. The likelihood is computed following Eq. (6.2). To do so, event histograms ($M(p_\mu^i, \theta_\mu^j | E^k)$) binned in muon momentum and angle given a neutrino energy are generated from the simulated data described in Sec. 6.1.1. The oscillated probability is computed by reweighting the histogram content, using Eq. (6.1), as

$$M^{\text{osc}}(p_\mu^i, \theta_\mu^j | \theta_{\text{mix}}, \Delta m^2) = \sum_k M(p_\mu^i, \theta_\mu^j | E^k) p_{\text{osc}}(E^k, \theta_{\text{mix}}, \Delta m^2),$$

and is interpolated linearly to reduce the effects due to the coarse binning:

$$M^{\text{osc}}(p_\mu^i, \theta_\mu^j | \theta_{\text{mix}}, \Delta m^2) \to M^{\text{osc}}(p_\mu, \theta_\mu | \theta_{\text{mix}}, \Delta m^2).$$

The number of expected events ($\mu(\theta, \Delta m^2)$) is computed by summing the binned probability:

$$\mu(\theta_{\text{mix}}, \Delta m^2) = \sum_{i,j} M^{\text{osc}}(p_\mu^i, \theta_\mu^j | \theta_{\text{mix}}, \Delta m^2).$$

The probability obtained by normalizing the oscillated maps takes the form

$$p\left(p_\mu, \theta_\mu | \theta, \Delta m^2\right) = \frac{M^{\text{osc}}(p_\mu, \theta_\mu | \theta_{\text{mix}}, \Delta m^2)}{\mu(\theta_{\text{mix}}, \Delta m^2)}.$$

The likelihood probability $L(\theta_{\text{mix}}, \Delta m^2)$ is finally computed for discrete values of θ_{mix} and Δm^2. Those values are distributed in a grid identical to the one used for the NSF approach for proper comparison between both methods. We have tested the results with different numbers of initial bins in energy, momentum and angle to find a good compromise between stability of the result and speed. We call this cross-check method in what follows the *Hist* method and it will be used as a reference for the NSF calculations.

6.3 Experiments

The methodology is tested on experiments of simulated neutrino oscillations according to the T2K experiment. For this, ten different sets of observed events are constructed (see Appendix A), with additional MC data used to fit both the NSF and construct the unbinned likelihood. The training of the NSF to fit the density of $(p_\mu, \theta_\mu, E_\nu)$ is shown and verified in Sec. 6.3.1. Afterward, in Sec. 6.3.2, the inference on the ten experiments are performed utilizing both NSF and the unbinned likelihood, discussing the findings of both methodologies and the possible bias introduced by them.

All events were generated as defined in Sec. 6.1.1, using the neutrino flux energy spectrum from the T2K Collaboration [175] to produce the energy of the incoming neutrino E_ν, and the NEUT event generator [174] to compute the momentum p_μ and angle θ_μ of the resulting muon, forming a triplet of $(p_\mu, \theta_\mu, E_\nu)$, describing an implicit density of these three magnitudes.

6.3.1 Training and validation of the NSF

In order to apply the methodology described in Sec. 6.2, we start by estimating the density $p(p_\mu, \theta_\mu, E_\nu)$ using a neural spline flow[1] on $\approx$ 15M MC events as

[1] We would like to thank C. Durkan, A. Bekasov, I. Murray and G. Papamakarios for their implementation of NSF, on which this work's code is based on: `https://github.com/bayesiains/nsf`

the training set to fit the parameters of the flow. As is a standard procedure in machine learning, an additional set of $\approx$ 4M events is used for validation of the model outside of the training set, i.e., to check the integrity of the models for data not seen to fit the parameters.

Because of the similarities regarding dimension of the data and number of samples, but with a simpler structure, we have chosen the hyperparameters almost as the POWER dataset in Appendix B from [131], i.e., ADAM optimizer (see Sec. 3.3) with learning rate 0.0005, batch size 512, training steps 400k, flow transformations 5, hidden layers in the conditioner 5, hidden features in the conditioner 128 and bins 8. Additionally, the learning rate was decreased during the training using a cosine scheduler [178] to ensure stabilization at the end of the training procedure. As shown in Fig. 6.1, the validation set stabilizes at the end of the training and appears to converge for the selected architecture of the network. This asymptotic behaviour towards a value of the log probability shows that the estimated density has converged properly, independently of the log probability obtained, which is hard to quantify and only serves as a means to compare different density estimators (the higher, the better).

To ensure that the transformation T^{-1} of Eq. (4.1) was found properly by the NSF, consider the transformed data $\mathbf{u}$ for a new set of $\approx$ 1M MC events, never seen before by the algorithm. If T^{-1} is correctly approximated, $\mathbf{u}$ should follow a three-dimensional standard normal distribution, as is shown qualitatively in Fig. 6.2.

For this, a χ^2-test was performed over a binning of $50 \times 50 \times 50$ in the domain $[-3, 3]^3$ of the transformed data $\mathbf{u}$ to test the goodness-of-fit to a three-dimensional standard normal distribution, obtaining a p-value of 0.3062. With this, we can assume that the transformed data $\mathbf{u}$ correspond to samples from such base distribution, justifying that the transformation T^{-1} was properly found by the NSF, hence allowing to evaluate $p(p_\mu, \theta_\mu, E_\nu)$ accurately through it.

6.3.2 Inference results

To test the performance of obtaining the posterior according to Sec. 6.2, ten different observation sets were constructed, as explained in Appendix A, with five different mixing angles θ_{mix} and difference in mass squared Δm^2.

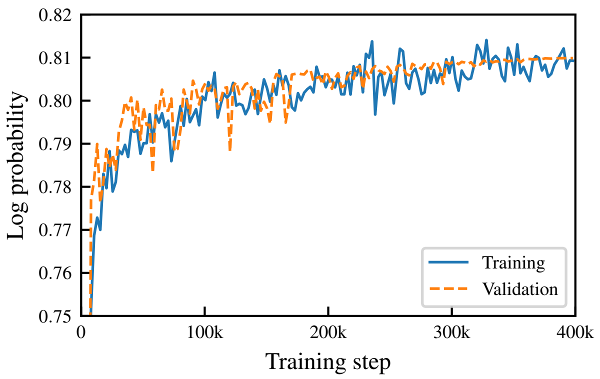


Fig. 6.1: Neural spline flow training log probability for estimating $p(p_\mu, \theta_\mu, E_\nu)$ for training (solid) and validation (dashed) sets, as shown by Eq. (4.7). For the validation, the log probability stabilizes during the training to converge to a certain value which depends on the architecture of the network.

For each of the five combinations of parameters, low and high statistics (number of observed events) experiments were performed. Low statistics are of the order of the real observed samples at T2K and used to assure its performance when a small number of events is dealt with. High statistics (2 orders of magnitude larger number of observed events compared to the usual expected number in the low statistics case) allow us to check the agreement between traditional binning methodology with a large number of very fine bins and the unbinned NSF posterior.

Table 6.1 shows the ten experiments, with the number of observed samples, the true parameters and the results using the NSF (Sec. 6.2). Additionally, a result using an approximate unbinned likelihood, denoted by *Hist* (Sec. 6.2.1), is also displayed, which would correspond to the limit case when histograms can be performed with a large number of bins to behave like an unbinned estimation. Since Bayesian inference is used, the inference on the parameters describes a density function according to Eq. (6.3). The central value shown for each parameter is the one that maximizes the joint posterior density. The

Tab. 6.1: Posterior inference of ten different experiments, alternating between low and high number of observed events. For the inferred parameters using NSF and the unbinned histogram approximation, *Hist*, the 95% confidence level was computed using the one-dimensional marginalized densities of each parameter. θ_{mix} is given in rad and Δm^2 in $\times 10^{-3}\text{eV}^2$.

Exp. #	N_{obs}	Parameter	True values	NSF 95 % C.L.	*Hist* 95 % C.L.
1	506	θ_{mix}	0.7594	$0.785^{+0.055}_{-0.056}$	$0.785^{+0.055}_{-0.056}$
		Δm^2	2.463	$2.440^{+0.091}_{-0.085}$	$2.446^{+0.092}_{-0.087}$
2	49672	θ_{mix}	0.7594	$0.751^{+0.074}_{-0.006}$	$0.754^{+0.069}_{-0.007}$
		Δm^2	2.463	$2.464^{+0.008}_{-0.012}$	$2.467^{+0.007}_{-0.012}$
3	532	θ_{mix}	0.7353	$0.71^{+0.18}_{-0.03}$	$0.71^{+0.18}_{-0.03}$
		Δm^2	2.463	$2.46^{+0.10}_{-0.11}$	$2.46^{+0.10}_{-0.11}$
4	49646	θ_{mix}	0.7353	$0.738^{+0.099}_{-0.006}$	$0.739^{+0.097}_{-0.006}$
		Δm^2	2.463	$2.458^{+0.009}_{-0.011}$	$2.461^{+0.009}_{-0.011}$
5	493	θ_{mix}	0.6847	$0.65^{+0.28}_{-0.02}$	$0.65^{+0.28}_{-0.02}$
		Δm^2	2.463	$2.554^{+0.120}_{-0.127}$	$2.563^{+0.122}_{-0.128}$
6	49665	θ_{mix}	0.6847	$0.682^{+0.210}_{-0.003}$	$0.683^{+0.208}_{-0.004}$
		Δm^2	2.463	$2.479^{+0.008}_{-0.014}$	$2.482^{+0.009}_{-0.014}$
7	506	θ_{mix}	0.7353	$0.73^{+0.14}_{-0.04}$	$0.73^{+0.14}_{-0.04}$
		Δm^2	2.363	$2.347^{+0.097}_{-0.095}$	$2.353^{+0.096}_{-0.099}$
8	50145	θ_{mix}	0.7353	$0.734^{+0.108}_{-0.006}$	$0.735^{+0.106}_{-0.006}$
		Δm^2	2.363	$2.365^{+0.010}_{-0.011}$	$2.368^{+0.009}_{-0.012}$
9	481	θ_{mix}	0.7353	$0.785^{+0.060}_{-0.061}$	$0.785^{+0.060}_{-0.061}$
		Δm^2	2.663	$2.620^{+0.086}_{-0.087}$	$2.626^{+0.087}_{-0.089}$
10	49710	θ_{mix}	0.7353	$0.741^{+0.094}_{-0.005}$	$0.743^{+0.089}_{-0.005}$
		Δm^2	2.663	$2.659^{+0.007}_{-0.011}$	$2.662^{+0.008}_{-0.010}$

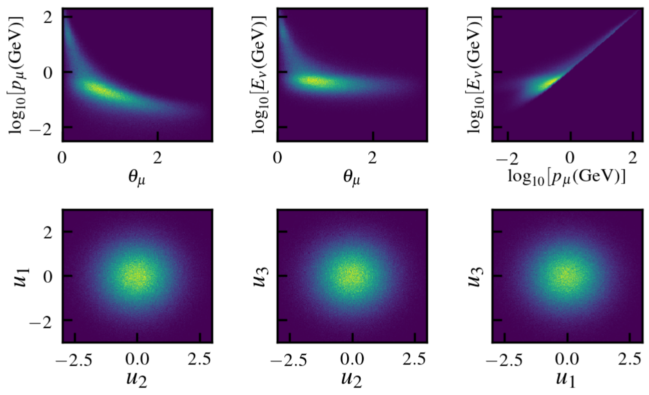


Fig. 6.2: Two-dimensional histograms of samples from the initial distribution of $(p_\mu, \theta_\mu, E_\nu)$ (top) and the transformed data $\mathbf{u} = (u_1, u_2, u_3)$ (bottom) under T^{-1} according to Eq. (4.1). If transformation T^{-1} is approximated properly, $\mathbf{u}$ should follow a three-dimensional standard normal distribution, as is depicted qualitatively in this figure.

uncertainty is then computed by marginalizing in the two-dimensional density one of the parameters to obtain the one-dimensional one of the other, and finding the interval such that for a $1-\alpha$ confidence level (CL), $\alpha/2$ of the density is found on each side. This is done for both NSF posterior and *Hist* posterior.

In general, the results of both methodologies agree, with slight fluctuations in the confidence levels. The difference could come from the NSF not learning perfectly the density of the points, from the interpolation done by the *Hist* method introducing wrong approximations or from intrinsic biases, which will be discussed at the end of this subsection.

Additionally, in order to visualize the agreement in two-dimensions in Figs. 6.3 and 6.4, the highest posterior density (HPD) curves [179] of 68% and 95%, together with the best fit (highest posterior value) were computed for experiments 1 (Fig. 6.3 top), 2 (Fig. 6.3 bottom), 7 (Fig. 6.4 top) and 8 (Fig. 6.4 bottom). In both HPD regions a clear overlap for low statistics, and slight fluctuations on larger statistics can be observed. When comparing the difference in area size, the relative difference of the *Hist* method with respect to NSF through the ten experiments is $3.1 \pm 1.9\%$, showing that the areas agree within 2-σ on average.

In Figs. 6.3 and 6.4, one observes that the areas, even being similar in size, are slightly shifted one from another. Empirical experiments of computing the posterior using actual discrete binning show that, by using a denser binning, its posterior was shifting toward the NSF result. To measure the bias $(\hat{\theta} - \theta)$ of the estimated parameters $(\hat{\theta}_{\text{mix}}, \hat{\Delta m}^2)$ by both methods, an observation of $N_{\text{obs}} \approx 500\text{k}$ events was used for the two sets of parameters used in Figs. 6.3 and 6.4. Results are summarized in Tab. 6.2 where $(\hat{\theta}_{\text{mix}}, \hat{\Delta m}^2)$ are taken as the maximum value of the posterior probability obtained in each set of parameters. Tab. 6.2 shows that NSF has a significantly smaller bias compared to the *Hist* method. This explains the discrepancy in the plots, aside from justifying a better performance by the NSF method. The *Hist* bias comes from binning and interpolation approximations, while for NSF the bias may come from the density estimation for the near detector not being perfect. In real experiments, this bias can be estimated using a representative MC dataset, called the *Asimov dataset*[2].

Both quantitative, Tab. 6.1, and qualitative, Figs. 6.3 and 6.4, show that NSF indeed provide a tool to perform likelihood-free inference on physical simulators such as the one of the T2K experiment, in agreement with unbinned likelihood approaches as we have compared it to, but with less bias as shown in Tab. 6.2.

6.4 Modelling systematic uncertainties

A comprehensive description of the problem is beyond the scope of this study, but we will sketch possible implementation alternatives. In experiments, near detector neutrino interactions data are used to constrain uncertainties in cross section models $(\hat{\phi}_{\text{xsect}})$, neutrino flux $(\hat{\phi}_{\text{flux}})$ and detector smearing and efficiency $(\hat{\phi}_{\text{det}})$. Comparing the experimental data from the near detector to the

[2] The concept of the Asimov data is being a representative date set, inspired by the short story Franchise by Isaac Asimov, in which the entire electorate is replaced by selecting the most representative voter.

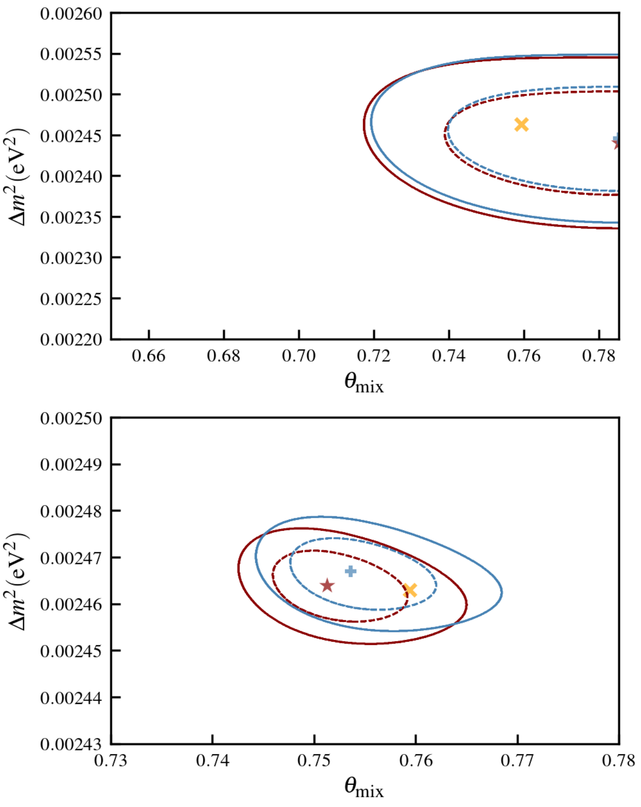


Fig. 6.3: Highest posterior density curves for both NSF and *Hist* posteriors of experiments 1 (top) and 2 (bottom), together with best fit of each posterior. Red (blue) lines indicate 68 % (dashed) and 95% (continuous) HPD curves for the NSF (*Hist*) method. Orange x-crosses indicate the true parameter used to generate the observed events. Red stars (blue +-crosses) indicate the best fit for the NSF (*Hist*) method. A clear overlap can be found in experiments of low statistics (top) and a slight fluctuation on large statistics (bottom). Notice a change of scale in the high statistics plots.

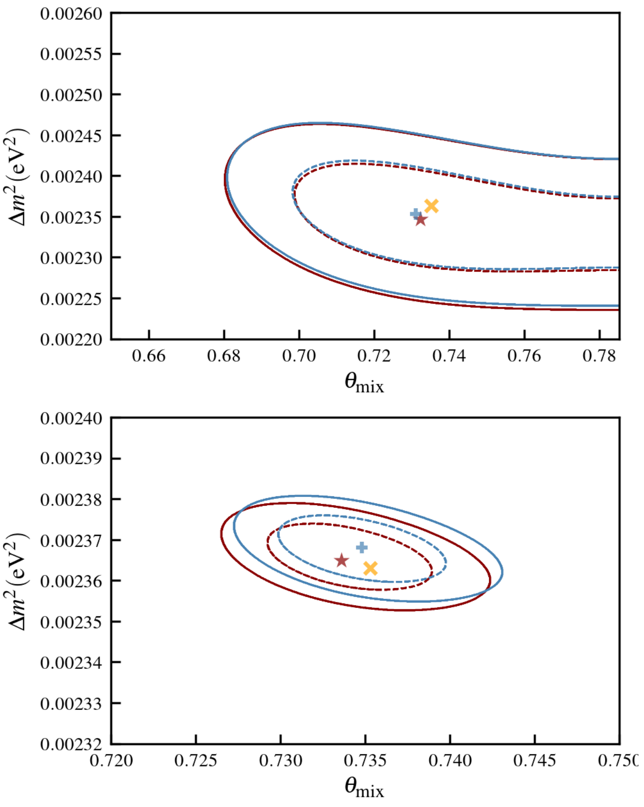


Fig. 6.4: Highest posterior density curves for both NSF and *Hist* posteriors of experiments 7 (top) and 8 (bottom), together with best fit of each posterior. Red (Blue) lines indicate 68 % (dashed) and 95% (continuous) HPD curves for the NSF (*Hist*) method. Orange x-crosses indicate the true parameter used to generate the observed events. Red stars (blue +-crosses) indicate the best fit for the NSF (*Hist*) method. A clear overlap can be found in experiments of low statistics (top) and a slight fluctuation on large statistics (bottom). Notice a change of scale in the high statistics plots.

Tab. 6.2: Bias computation for the posterior densities of the parameters in Figs. 6.3 and 6.4. The estimators $(\hat{\theta}_{\text{mix}}, \hat{\Delta m}^2)$ are taken as the maximum value of the posterior obtained in each set of parameters for a high-statistical experiment ($N_{\text{obs}} \approx 500\text{k}$), and the bias is defined as $\hat{\theta} - \theta$. NSF shows a significant reduction of bias compared to the *Hist* method. θ_{mix} is given in rad and Δm^2 in $\times 10^{-3}\text{eV}^2$.

Parameter	True values	NSF bias	*Hist* bias
θ_{mix}	0.7594	0.0020	0.0087
Δm^2	2.463	-0.0012	0.0031
θ_{mix}	0.7353	-0.00050	0.00410
Δm^2	2.363	-0.00063	0.00675

Monte Carlo model, experiments obtain the distribution for the parameters, $p(\hat{\phi}|\text{ND})$. The uncertainty parameters $(\hat{\phi}_i)$ are applied to the far detector to predict the data distributions in the absence of oscillations. The transport of constrained uncertainties are done either by traditional covariance matrices or by a more sophisticated Markov Chain Monte Carlo that easily accounts for non-Gaussian probability distributions. The uncertainties are then marginalized or profiled to propagate the uncertainties to the oscillation analysis.

The proposed method can be used in different ways in this analysis framework. The simplest approach is to obtain the $p(\hat{\phi}|\text{ND})$ that includes all possible parameter correlations. In this case, the model provides at the same time a simple way to generate Monte Carlo to sample the distributions. This method, using Gaussian *base densities*, will easily learn the nuisance parameters probability density function which is expected to be close to Gaussian. The next level of complexity is to learn, as we have done in this example, the probability density function but adding nuisance parameters, $p(p_\mu, \theta_\mu, E_\nu, \hat{\phi})$. This implementation will allow us to perform unbinned likelihoods as described in this work. The more complex and inclusive approach is to avoid the intermediate nuisance parameter density function $(p(\hat{\phi}|\text{ND}))$ description and model both near and far detectors with a set of common uncertainties. The advantage of this final description is that all the analysis is carried out in a single fit avoiding the description of hundreds of uncertainties $(\hat{\phi})$.

6.5 Conclusions

In this chapter, we have presented the viability of a likelihood-free inference methodology through neural spline flows on a simplified neutrino oscillation problem at the T2K experiment. We developed a framework to use this estimation of the density from data taken at the near detector in order to perform inference of the oscillation parameters at the far detector for a simplified two-flavour neutrino problem, allowing to perform exact inference if the density is properly estimated. This method provides potential advantages over traditional binned histogram methods, especially when the statistics is low as is the case in the T2K experiment. An unbinned alternative formulated by interpolating the histogram method was constructed to check the results. Additionally, the integrity of the learned density was thoroughly verified through different statistical and empirical tests. The results obtained using the neural spline flow methodology and the unbinned likelihood methodology show results which are in agreement with each other in the estimation of the statistical errors. The alternative method is not refined enough and it shows larger bias in the estimated parameters. The results presented in this work open new possibilities to use similar likelihood-free neural network inference for more complex statistical analyses.

7. EXHAUSTIVE NEURAL IMPORTANCE SAMPLING APPLIED TO MONTE CARLO EVENT GENERATION

Adapted from the original publication in Physical Review D 102, 013003 on the 16th of July 2020 [18].

In modern science and engineering disciplines, the generation of random samples from a probability density function to obtain datasets or compute expectation values has become an essential tool. These theoretical models can be described by a target probability density function $p(\mathbf{x})$. Ideally, to generate samples following $p(\mathbf{x})$, the inverse transformation method is used. To perform the inverse transformation, the cumulative probability has to be calculated and the inverse to this function has to be found. Numerical methods have to be applied to obtain the MC samples when this is not feasible computationally. This is especially true for high-dimensional spaces, where the integrals required to find such inverse transformation become analytically challenging.

A standard numerical method to obtain such datasets is to perform a Markov Chain Monte Carlo (MCMC) algorithm [180], which provides good results for expected value calculations. Compared to other methods, it has the advantage that, in general, it requires very little calibration, and high dimensions can be broken down into conditional smaller dimension densities [181]. However, the MCMC method produces samples that form a correlated sequence. Also, the convergence of the samples' chain to the target density cannot be guaranteed for all possible models.

Another standard algorithm to produce MC samples is the acceptance-rejection or simply rejection sampling [182–185], which produces i.i.d. (independent and identically distributed) samples from the target density via an auxiliary proposal function. The proposal has to satisfy being a density which can both be sampled from and evaluated efficiently, as well as being as close to the target density as possible. The main disadvantages of the method are the following [186]:

1. Designing the proposal function close to a particular target density can be very costly in human time.

2. If a generic proposal function is taken, such as a uniform distribution over the domain, the algorithm is usually very inefficient.

3. The sampling efficiency decreases rapidly with the number of dimensions.

Ideally, to avoid these inconveniences, one would like to have a method to find a proposal function that adapts to a given target density automatically. This would solve simultaneously the human time cost as well as the inefficiency of generic proposal densities.

An approach of the usage of normalizing flows (see Chapter 4) to find a suitable proposal for a given target density has been suggested previously as neural importance sampling (NIS) [129], focused on the integration of functions via importance sampling [187]. The concept of integrating via importance sampling with normalizing flows for HEP has been explored in other works to obtain top-quark pair production and gluon-induced multi-jet production [105] or to simulate collider experimental observables for the Large Hadron Collider [104].

In this work we further explore the possibility of utilizing normalizing flows to find a proposal function for a given target density to perform rejection sampling for MC samples, and analyze its viability through the following points:

- We discuss the importance of adding an additional density (background) to the target one to assure the coverage of the whole phase space when performing rejection sampling.

- We define a two-phase training scheme for the normalizing flow to boost initial inefficiency in the optimization when adjusting the initialized random density towards the target one.

- We measure the performance of the method and argue for relaxing the rejection sampling constant factor k to improve largely the efficiency of acceptance while quantifying the error committed in doing this approximation via the concept of coverage.

Considering the proposed algorithm covers the whole domain of interest by modifying NIS with the background density, we denote this method by exhaustive neural importance sampling (ENIS).

We apply the above algorithm to a HEP problem, in the form of the CCQE cross section for anti-neutrinos interactions with nuclei, performing in-depth analysis and discussion of the efficiency of the method. Neutrino-nucleus cross section modeling is one of the main sources of systematic uncertainties in neutrino oscillations measurements [172,188,189]. Cross section models are either analytically simple but describe the experimental data poorly or involving complex numerical computations, normally related to the description of the nucleus, that imposes limitations in their MC implementation. New tendencies in the field also call for a fully exclusive description of the interaction adding complexity to the calculations. The analytical model utilized in this paper is simple, but it is a realistic one and a good reference to demonstrate the capabilities of the proposed method to generate neutrino-nucleus cross sections efficiently. We will show that ENIS opens the possibility to incorporate efficiently complex theoretical models in the existing MC models enhancing the physics reach of running and future neutrino oscillation experiments.

ENIS algorithm may be used beyond the scope of neutrino physics. Further applications to be evaluated in detail in the future are particle/nuclear physics experiments, detector responses for medical physics, engineering studies or theoretical modelling. In general, it could be applied to any Monte Carlo simulation that is limited by the algorithm's speed, such as for importance sampling to provide fast Monte Carlo with sufficient accuracy (i.e. fast detector simulation, design studies, minimum bias background simulations, etc.). Additionally, the technique may help model developers extract expected values from their theoretical predictions in realistic conditions by including simple detector effects in models, such as effects of detector acceptance cuts, impact of model degrees of freedom on the predictions or uncertainty propagation.

7.1 Framework

In this section we will describe the background and framework needed for the rest of the chapter. Sec. 7.1.1 explains the physical model of charged current quasielastic neutrino interaction we will apply ENIS to in Sec. 7.3. As a summary and to introduce our notation, Sec. 7.1.2 overviews the rejection sampling algorithm.

7.1.1 Model of charged current quasielastic anti-neutrinos interactions with nuclei

The CCQE is a basic model of neutrino interactions that might be expressed in simple formulae. The CCQE model has many advantages during this exploratory work, as it can be implemented in a simple software function, while at the same time it is also a realistic environment to understand the implications of modeling cross sections with the proposed methodology. The selected model to describe CCQE is the well established Smith-Monith [190]. The nucleon momentum distribution follows a relativistic Fermi gas (non-interacting nucleons in a nuclear potential well) with a 0.225 GeV/c Fermi level. The model includes the Pauli blocking effect, preventing the creation of final state nucleons below the nucleus Fermi level. The model can be applied both to neutrino and antineutrino interactions. Antineutrinos are selected for this study due to the vector axial current cancellation imposing more stringent conditions at the edges of the kinematic phase space. The model includes the following degrees of freedom generated by the MC model: the neutrino energy, the $\mu^{\pm}$ momentum and angle, and the target nucleon Fermi momentum. Contrary to other MC implementations, the neutrino energy is not a fixed input value but it is generated by the algorithm to add complexity to the calculations and to check the capabilities of the calculations to reproduce the cross section as a function of the neutrino energy. The implementation of this model for fixed energy value is also possible. The basic kinematic distributions obtained with this model will be discussed in Sec. 7.3.

7.1.2 Rejection sampling

Rejection sampling is a well known technique [182–186] to obtain MC samples from a target density $p(\mathbf{x})$ which can be evaluated (up to a constant), but cannot be sampled from through the inverse transform. It relies on an auxiliary proposal function $q(\mathbf{x})$, from which one should be able to sample from and evaluate efficiently. A constant $k > 0$ is introduced which has to satisfy that

$$k \cdot q(\mathbf{x}) \geq p(\mathbf{x}) \quad \forall\, \mathbf{x} : p(\mathbf{x}) > 0. \tag{7.1}$$

The resulting function $k \cdot q(\mathbf{x})$ is called the comparison function.

The procedure to sample from the target density is then the following:

1. A sample $\mathbf{x}$ is generated following $q(\mathbf{x})$, $\mathbf{x} \sim q(\mathbf{x})$.
2. A random number u is generated uniformly in the range $[0, k \cdot q(\mathbf{x})]$, $u \sim \text{Unif}(0, k \cdot q(\mathbf{x}))$.
3. If u fulfills the condition $u \leq p(\mathbf{x})$, the sample is accepted; otherwise, it is rejected.

Additionally, if $p(\mathbf{x})$ is normalized, the probability that a sample is accepted is proportional to $p_{\text{acc}} \propto 1/k$, i.e., k gives an intuition of the number of tries until we obtain an accepted sample.

7.2 Methodology

With the framework introduced in Sec. 7.1 and the potential of normalizing flows seen in Chapter 4, we are now in a position to define the ENIS method and the different metrics we will use to measure its performance.

We start in Sec. 7.2.1 by showing how to modify the objective function to be minimized by the normalizing flow (Sec. 4.1.2) to adjust its proposal function $q_\phi(\mathbf{x})$ to the target density $p(\mathbf{x})$ in the case of performing neural importance sampling. Then, in Sec. 7.2.2, we discuss the importance of adding background noise to both ensure coverage of the whole phase space of $p(\mathbf{x})$ and to boost the initial phase of the training of ENIS. The exact training scheme is then shown in Sec. 7.2.3, differentiating the warm-up phase from the iterative phase. Finally, in Sec. 7.2.4, the performance metrics are introduced, explaining the concept of coverage and effective sample size when considering a more relaxed condition on the rejection constant k.

7.2.1 Modified optimization for NIS

Consider a target probability density function $p(\mathbf{x})$ which can be evaluated for all $\mathbf{x}$ but from which we are unable to generate samples directly through an analytical inverse transform. If we could approximate this target density by our neural density estimator $q_\phi(\mathbf{x})$, then we could exactly sample from the target density using rejection sampling, since we can both sample and evaluate from $q_\phi(\mathbf{x})$.

To obtain the parameters ϕ of $q_\phi(\mathbf{x})$ given a density $p(\mathbf{x})$ which can be evaluated, we want to minimize the KL-divergence, Eq. (4.5), between both distributions. This can be done by computing the gradient of Eq. (4.6), which could be approximated numerically if we could sample $\mathbf{x} \sim p(\mathbf{x})$, since it corresponds to approximating an expected value of a function we can evaluate, $\log q_\phi(\mathbf{x})$. This, however, is not the case.

Müller *et al.* [129] propose a solution for computing the gradient with respect to ϕ for this maximization problem. They suggest using importance sampling [187] for this particular expected value:

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\log q_\phi(\mathbf{x})\right] &= \int p(\mathbf{x})\, \nabla_\phi \log q_\phi(\mathbf{x})\; d\mathbf{x} \\
&= \int q_\phi(\mathbf{x})\, \frac{p(\mathbf{x})}{q_\phi(\mathbf{x})} \nabla_\phi \log q_\phi(\mathbf{x})\; d\mathbf{x} \\
&= \mathbb{E}_{\mathbf{x}\sim q_\phi(\mathbf{x})}\left[w(\mathbf{x}) \nabla_\phi \log q_\phi(\mathbf{x})\right] \\
&\approx \frac{1}{N} \sum_{i=1}^{N} w(\mathbf{x}_i) \nabla_\phi \log q_\phi(\mathbf{x}_i)\,, \qquad (7.2)
\end{aligned}$$

with $\mathbf{x}_i \sim q_\phi(\mathbf{x})$ and the weights defined as $w(\mathbf{x}) = p(\mathbf{x})/q_\phi(\mathbf{x})$. Notice how we only need to be able to evaluate $p(\mathbf{x})$ to compute this quantity. With this gradient, we are able to minimize the KL-divergence in Eq. (4.6) if the support of $q_\phi(\mathbf{x})$ (i.e., the domain where the function is non-zero) contains the support of $p(\mathbf{x})$ to perform the importance sampling of Eq. (7.2) correctly. Notice that, in order to properly optimize the parameters ϕ, $p(\mathbf{x})$ does not need to be normalized, since this simply changes the magnitude of the gradient, but not its direction. The lack of proper normalization can be properly handled by standard neural network optimizers such as Adam (see Sec. 3.3).

The method described by Eq. (7.2) implies an iterative way of optimizing $q_\phi(\mathbf{x})$ with the following steps:

1. A batch of $\mathbf{x}$ is generated according to the current state of the neural network, $q_\phi(\mathbf{x})$.

2. Using this batch, the parameters ϕ of the neural network are optimized via the gradient of Eq. (7.2).

3. This updated neural network then generates the next batch.

7.2.2 Relevance of background noise

As briefly discussed in Sec. 7.1, in order to optimize the neural network following Eq. (7.2), the gradient is only correctly computed if the support of $q_\phi(\mathbf{x})$ contains the one of $p(\mathbf{x})$. Moreover, if we want to use $q_\phi(\mathbf{x})$ as our proposal function to sample from $p(\mathbf{x})$ via rejection sampling, this also has to hold.

To ensure the proper $p(\mathbf{x})$ support, we introduce the concept of a background density function, $p_{\text{bg}}(\mathbf{x})$. In HEP, as in many other scientific areas, the density is restricted to a certain domain of $\mathbf{x} \in \mathbb{R}^D$, e.g., the cosine has to be in $[-1, 1]$, the magnitude of the momentum in an experiment has to be positive and has a maximum value of p_{max}, there are constraints in the conservation of energy and momentum, etc. Hence $p_{\text{bg}}(\mathbf{x})$ should be a density that has a support beyond these phase-space boundaries. In what follows, a uniform distribution will be considered, with limits in each dimension according to the phase space of that coordinate. The selection of the functional form of the $p_{\text{bg}}(\mathbf{x})$ is arbitrary and it can be selected to adapt to the requirements of each project.

The background density $p_{\text{bg}}(\mathbf{x})$ will be used for two tasks:

(i) Improve initial training: At the beginning of the training, we cannot assure that the support of $q_\phi(\mathbf{x})$ contains the one of $p(\mathbf{x})$. Hence, instead of using $q_\phi(\mathbf{x})$ for the importance sampling of Eq. (7.2), $p_{\text{bg}}(\mathbf{x})$ will be used during the warm-up phase of the training. The distribution of the weight function $w(\mathbf{x}) = p(\mathbf{x})/p_{\text{bg}}(\mathbf{x})$ might span several orders of magnitude, but this way we ensure the full support of $p(\mathbf{x})$. This strategy gives a better approximation than the one obtained by the randomly initialized neural network $q_\phi(\mathbf{x})$ at the start of the training.

(ii) Ensure exhaustive coverage of the phase space: The target density $p_{\text{target}}(\mathbf{x})$ that the neural network will learn will be constructed as a linear combination of the true target density $p(\mathbf{x})$ and the background $p_{\text{bg}}(\mathbf{x})$:

$$p_{\text{target}}(\mathbf{x}) = (1-\alpha) \cdot p(\mathbf{x}) + \alpha \cdot p_{\text{bg}}(\mathbf{x}), \tag{7.3}$$

with $\alpha \in (0,1)$. This implementation adds a certain percentage α of background noise to the target density, spreading it over all the domain of the background density, allowing to properly apply the methods rejection and importance sampling with $q_\phi(\mathbf{x})$ as the proposal function, covering exhaustively the phase space. Experimentally we have found good compromise with $\alpha = 0.05$.

Optimizing $q_\phi(\mathbf{x})$ to match $p_{\text{target}}(\mathbf{x})$ of Eq. (7.3) instead of $p(\mathbf{x})$ will make the proposal $q_\phi(\mathbf{x})$ slightly worse for rejection/importance sampling. By performing the optimization to $p(\mathbf{x})$ directly in an iterative way, as explained at the end of the last section, some regions of the phase space might disappear for future samplings. These regions are located normally close to the boundaries of sampled volume. Having a constant background noise prevents these losses from appearing, as the neural network has to also learn to generate this noise, covering properly the required phase-space volume. We will discuss the impact of the background term on the method performance in Sec. 7.3.

7.2.3 ENIS training scheme of the proposal function

The training procedure to obtain $q_\phi(\mathbf{x})$ from $p(\mathbf{x})$ following ENIS consists of two phases:

1. Warm-up phase:

   (i) Sample $\mathbf{x}_p \sim p_{\text{bg}}(\mathbf{x})$ and compute their weights $w_p(\mathbf{x}_p) = p(\mathbf{x}_p)/p_{\text{bg}}(\mathbf{x}_p)$.

   (ii) Sample background $\mathbf{x}_{\text{bg}} \sim p_{\text{bg}}(\mathbf{x})$ with associated weights $w_{\text{bg}}(\mathbf{x}_{\text{bg}}) = C_{w_{\text{bg}}} \cdot p_{\text{bg}}(\mathbf{x}_{\text{bg}})$, where $C_{w_{\text{bg}}} = \frac{\alpha}{1-\alpha}\frac{\langle w_p(\mathbf{x}_p)\rangle}{\langle p_{\text{bg}}(\mathbf{x}_{\text{bg}})\rangle}$.

   (iii) Optimize the parameters of $q_\phi(\mathbf{x})$ via Eq. (7.2) using $\mathbf{x} = \{\mathbf{x}_p, \mathbf{x}_{\text{bg}}\}$ with weights $w(\mathbf{x}) = \{w_p(\mathbf{x}_p), w_{\text{bg}}(\mathbf{x}_{\text{bg}})\}$.

2. Iterative phase:

   (i) Sample $\mathbf{x}_q \sim q_\phi(\mathbf{x})$ and compute their weights $w_q(\mathbf{x}_q) = p(\mathbf{x}_q)/q_\phi(\mathbf{x}_q)$.

   (ii) Sample background $\mathbf{x}_{\text{bg}} \sim p_{\text{bg}}(\mathbf{x})$ with associated weights $w_{\text{bg}}(\mathbf{x}_{\text{bg}}) = C'_{w_{\text{bg}}} p_{\text{bg}}(\mathbf{x}_{\text{bg}})$, where $C'_{w_{\text{bg}}} = \frac{\alpha}{1-\alpha}\frac{\langle w_q(\mathbf{x}_q)\rangle}{\langle p_{\text{bg}}(\mathbf{x}_{\text{bg}})\rangle}$.

(iii) Optimize the parameters of $q_\phi(\mathbf{x})$ via Eq. (7.2) using $\mathbf{x} = \{\mathbf{x}_q, \mathbf{x}_{\text{bg}}\}$ with weights $w(\mathbf{x}) = \{w_q(\mathbf{x}_q), w_{\text{bg}}(\mathbf{x}_{\text{bg}})\}$.

Figures 7.1 and 7.2 depict a flow diagram of the training method for ENIS, showing in Fig. 7.1 the warm-up phase, while in Fig. 7.2 the iterative phase is depicted. The phase transition from warm-up to iterative phase is chosen heuristically. In our particular implementation of Sec. 7.3.1 we have chosen the warm-up phase to comprise 20 % of the total training iterations.

Steps 1. (ii) and 2. (ii) allow the method to add background following Eq. (7.3) to construct $p_{\text{target}}(\mathbf{x})$ even if $p(\mathbf{x})$ is not normalized.

7.2.4 Measuring the performance of the proposal function

The proposed method is not to use $q_\phi(\mathbf{x})$ as a direct approximation of $p(\mathbf{x})$, but as proposal function to perform either rejection (Sec. 7.1.2) or importance sampling [187]. This allows for the methods to correct any deviation in the neural network modeling of the exact density while utilizing its proximity to such density.

We use the learned probability density function $q_\phi(\mathbf{x})$ to generate samples via rejection sampling (see Sec. 7.1.2), which, in HEP, is of high interest and costly via standard procedures. The parameter k of the rejection algorithm has to be estimated empirically. Consider n samples $\{\mathbf{x}_i\}_{i=1}^n$ generated with the proposal function $\mathbf{x} \sim q(\mathbf{x})$, with weights $w(\mathbf{x}_i) = p(\mathbf{x})/q(\mathbf{x})$, satisfying:

- The average of the weights is

$$\langle w \rangle = \frac{1}{N}\sum_{i=1}^{N} w(\mathbf{x}_i) \approx \int q(\mathbf{x})\, w(\mathbf{x}) d\mathbf{x} = C,$$

  where C is the normalization of the density $p(\mathbf{x})$, i.e., its volume.

- $k_{\max}$, the smallest constant $k > 0$ such that the inequality of Eq. (7.1) holds, is equal to $(\max w(\mathbf{x}_i))^{-1}$.

In real conditions, the parameter k can be relaxed. Instead of choosing the maximum value among the empirically computed weight distribution, it can be taken as the inverse of the Q-quantile of these weights, w_{Q}, denoted by k_{Q}:

$$k_{\text{Q}} = (Q\text{-quantile}(w))^{-1} = w_{\text{Q}}^{-1}. \tag{7.4}$$

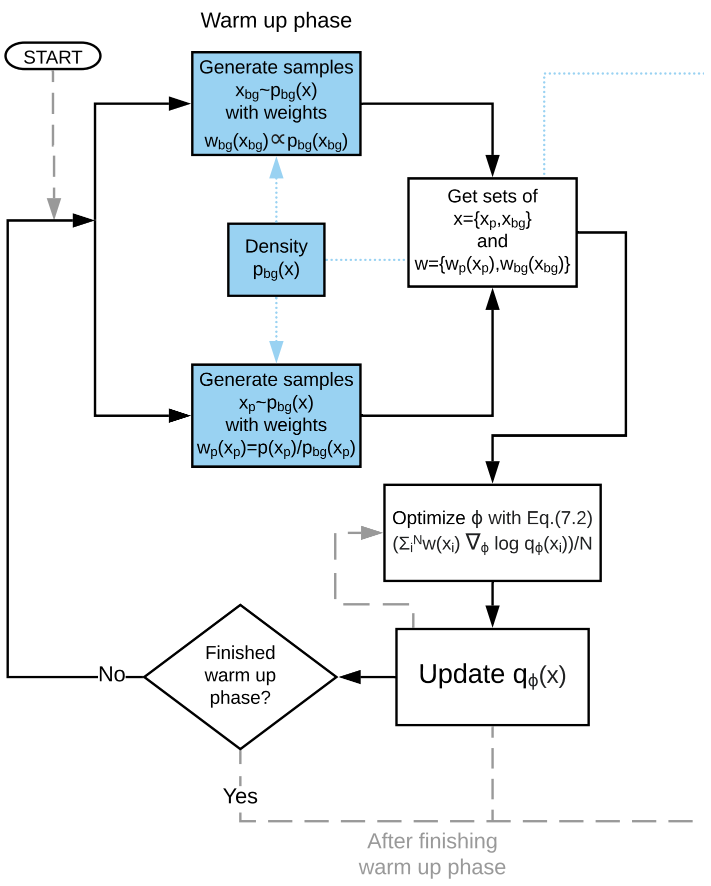


Fig. 7.1: Exhaustive neural importance sampling flow diagram: warm-up phase.

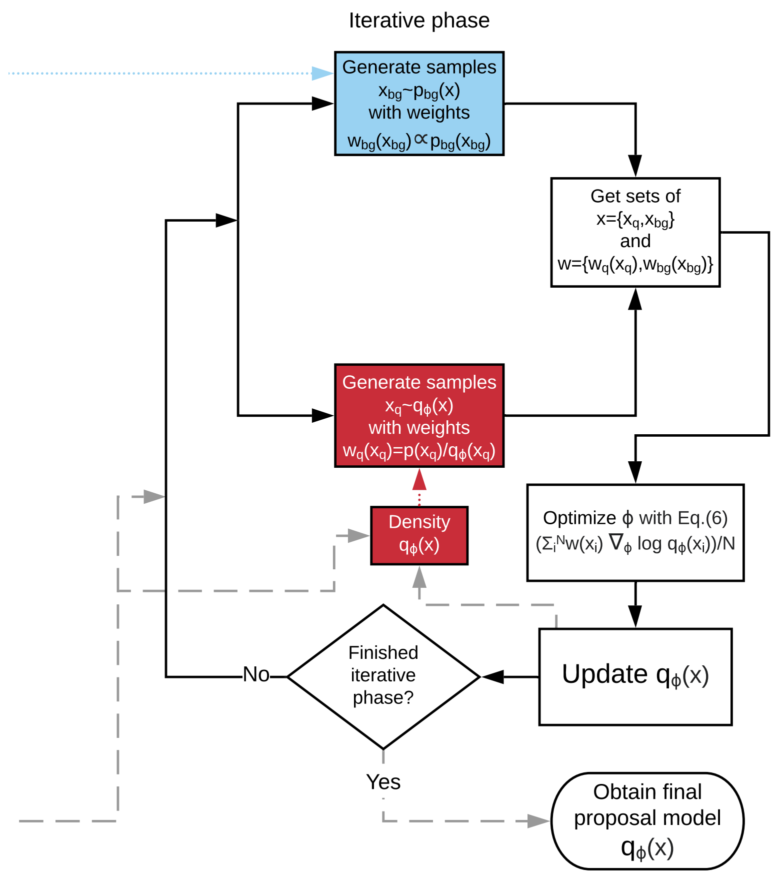

Iterative phase
Generate samples x_{bg}~$p_{bg}(x)$ with weights $w_{bg}(x_{bg}) \propto p_{bg}(x_{bg})$
Get sets of $x=\{x_q,x_{bg}\}$ and $w=\{w_q(x_q),w_{bg}(x_{bg})\}$
Generate samples x_q~$q_\phi(x)$ with weights $w_q(x_q)=p(x_q)/q_\phi(x_q)$
Density $q_\phi(x)$
Optimize ϕ with Eq.(6) $(\Sigma_i^N w(x_i)\ \nabla_\phi \log q_\phi(x_i))/N$
Update $q_\phi(x)$
Finished iterative phase?
No
Yes
Obtain final proposal model $q_\phi(x)$

Fig. 7.2: Exhaustive neural importance sampling flow diagram: iterative phase.

This is equivalent to clipping the weights' maximum value to the Q-quantile of w, capping the desired density function $p(\mathbf{x})$ we are generating using these weights for the rejection. The new weights $w'(\mathbf{x})$ are simply:

$$w'(\mathbf{x}_i) = \begin{cases} w(\mathbf{x}_i) & \text{if } w(\mathbf{x}_i) \leq w_\mathrm{Q} \\ w_\mathrm{Q} & \text{if } w(\mathbf{x}_i) > w_\mathrm{Q} \end{cases} .$$

The ratio of volume with respect to the original density $p(\mathbf{x})$ we are maintaining by clipping the weights this way defines the coverage we have of the rejection sampling, and is equal to

$$\text{Coverage} = \frac{\sum_{i=1}^{N} w'(\mathbf{x}_i)}{\sum_{i=1}^{N} w(\mathbf{x}_i)}. \tag{7.5}$$

This allows us to quantify the error we are committing when choosing a quantile over the maximum of weights when defining a constant k for rejection sampling.

The idea behind relaxing this constant k is that we will wrongly approximate only a small region of $p(\mathbf{x})$ with $q(\mathbf{x})$. In that small region, the ratio $p(\mathbf{x})/q(\mathbf{x})$ is large compared to the rest of the domain but still it is occupying a small volume of the density $p(\mathbf{x})$. This region can be ignored by relaxing k, making the overall ratio of $p(\mathbf{x})/(k \cdot q(\mathbf{x}))$ closer to 1 and improving drastically the rejection sampling at the cost of this small discrepancy which we are committing, quantified in Eq. (7.5).

As an additional qualitative measurement of the goodness of different proposals under different constants k, the effective sample size (ESS) will be used [191], which corresponds approximately to the number of independent samples drawn. The ESS for n samples of weights $\{w(\mathbf{x}_i)\}_{i=1}^{n}$ is defined as:

$$N_\mathrm{ESS} = \left(\sum_{i=1}^{N} w(\mathbf{x}_i) \right)^2 / \sum_{i=1}^{N} w(\mathbf{x}_i)^2. \tag{7.6}$$

This is a rule of thumb to obtain the number of independent samples. The closer N_ESS is to the number of samples n, the more uncorrelated the weighted samples are. If large weights appear, then the independence of the samples will be diminished, as the same sample gets represented many times. We define N_ESS/N as a rough estimate for the ratio of independence of the samples.

7.3 Monte Carlo generation of the CCQE antineutrino cross section

We will now proceed to apply ENIS to the CCQE antineutrino cross section density. In Sec. 7.3.1, we discuss how the training for the NSF was performed, describing the background we added to cover the phase space. We show qualitatively the obtained densities and compare them to the target one. After obtaining a suitable proposal, we discuss in depth the performance of the obtained result in Sec. 7.3.2, comparing the ENIS proposal to a generic uniform one, demonstrating its potential while justifying the relaxation on the constant k for the rejection sampling.

7.3.1 Training

To find the proposal function $q_\phi(\mathbf{x})$ via NSF[1] for the CCQE antineutrino interaction cross section density, described in Sec. 7.1.1, we followed the training scheme from Sec. 7.2.3.

The background chosen is a uniform distribution, covering a range of $[0, 10]$ for the incoming neutrino energy E_ν (in GeV), $[0, 10]$ for the outgoing muon momentum p_μ (in GeV/c), $[0, \pi]$ for the angle of the outgoing muon θ_μ (in rad), and $[0, 0.225]$ for the target nucleon Fermi momentum p_{nucleon} (in GeV/c). These bounds were expanded by covering a slightly more extended domain, of an additional 2 % at the beginning and end of each dimension, to assure that the physical boundaries are completely covered. This expanded background was added with a $\alpha = 0.05$ contribution to the cross section density in Eq. (7.3), as well as used during training for the warm-up phase.

As for the hyperparameters of the NSF, we have chosen the Adam optimizer [78] with learning rate 0.0005, batch size 5 000, training steps 400 000, 5 flow steps, 2 transform blocks, 32 hidden features and 8 bins. This gives a total dimension of 37 220 for the parameters ϕ of $q_\phi(\mathbf{x})$. This configuration for the NSF was chosen experimentally to have a relatively low number of parameters (one can have easily six million parameters instead of the $\approx 37\,000$ we have) since a lower number speeds up the generation and evaluation of samples

[1] Again, we would like to thank C. Durkan, A. Bekasov, I. Murray and G. Papamakarios for their implementation of NSF, on which this work's code is based on: `https://github.com/bayesiains/nsf`

$\mathbf{x} \sim q_\phi(\mathbf{x})$. Additionally, the learning rate was decreased during the training using a cosine scheduler [178] to ensure stabilization at the end of the training procedure.

The training consists in maximizing the log-likelihood of Eq. (4.7) by computing its gradient via Eq. (7.2), and is shown over the 400 000 iterations in Fig. 7.3. In the grey area, the training is performed with samples of the background distribution $p_{\mathrm{bg}}(\mathbf{x})$, while in the white area the samples of the training samples are generated by the current proposal distribution $q_\phi(\mathbf{x})$. Notice that since the samples are generated in real-time during the training, there is no need to worry about possible overfitting of the parameters of the neural network, which is a common issue in many machine-learning applications. The values of Fig. 7.3 are computed every one thousand steps, for a batch of 200 000 samples $\mathbf{x} \sim q_\phi(\mathbf{x})$. The log probability can be seen to converge at the end of the training, which is mainly due to the cosine scheduler, but also due to the saturation over the family of parametrized densities $q_\phi(\mathbf{x})$.

To have a visual representation, Fig. 7.4 shows the marginalized 1-dimensional densities of the four cross section variables of the target density $p(\mathbf{x})$ (blue) vs the NSF proposal $q_\phi(\mathbf{x})$ (orange). The plots show a small discrepancy in each variable, but an overall agreement between the two densities. Aside from a mismodeling on the side of $q_\phi(\mathbf{x})$ in certain regions, the differences can also come from the fact that the NSF is learning a modified target density (Eq. (7.3)).

To assess qualitatively that the correlation between the variables are also captured by $q_\phi(\mathbf{x})$, Figs. 7.5 and 7.6 show 2D-histograms for both the real density $p(\mathbf{x})$ and the the proposal density $q_\phi(\mathbf{x})$, respectively. Visually, an overall agreement can be seen. There is a slight discrepancy for high energy p_{nucleon} values, where the attenuation indicates that for the NSF proposal function it is more spread due to the background noise $p_{\mathrm{bg}}(\mathbf{x})$ it is also learning (Eq. (7.3)).

In what follows the performance of the NSF proposal will be discussed in more quantitative ways, and compared it to a uniform proposal.

7.3.2 Performance and discussion

In this section we will focus on analyzing the performance of the proposal density obtained by the NSF while also comparing it to a uniform proposal

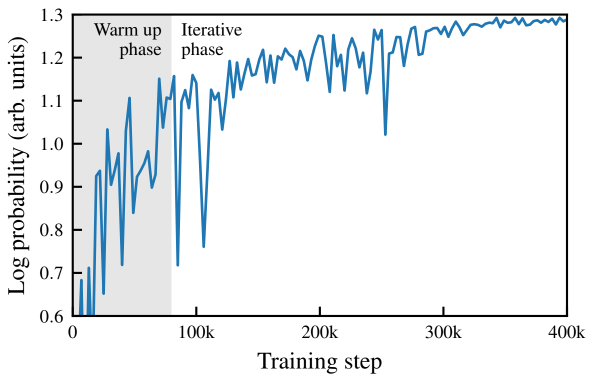


Fig. 7.3: Neural spline flow training log probability for estimating the modified CCQE cross section with background noise, following Eq. (4.7). In grey, the warm-up phase is performed, using $p_{\mathrm{bg}}(\mathbf{x})$ to generate the weighted samples, while in the white area the current state of the NSF $q_\phi(\mathbf{x})$ is used. The log probability stabilizes during the training to converge to a certain value which depends on the expressiveness of the network and the normalization of the target density.

density, $p_{\mathrm{Unif}}(\mathbf{x})$, which in our case will be the same as $p_{\mathrm{bg}}(\mathbf{x})$, defined in Sec. 7.3.1.

We start by generating ten million samples from each proposal density and compute their associated weights. The proportion of samples with weight equal to zero is 5.56 % for the NSF proposal, compared to the 98.03 % for the uniform one. To understand the distribution of such weights, Fig. 7.7 shows the logarithmic scale of them (for the weights > 0), assuming the average of the weights is equal to 1. For the NSF $q_\phi(\mathbf{x})$ (left), all weights are concentrated around $\log_{10} w = 0$ with a small dispersion around it. Notice that there are only three weights in ten million slightly over one hundred. This shape justifies using not the maximum value of w to perform rejection sampling, but some quantile of it, as we will discuss below. Conversely, for the uniform distribution $p_{\mathrm{Unif}}(\mathbf{x})$ (right) we can see that the spectrum of weights goes over nine orders

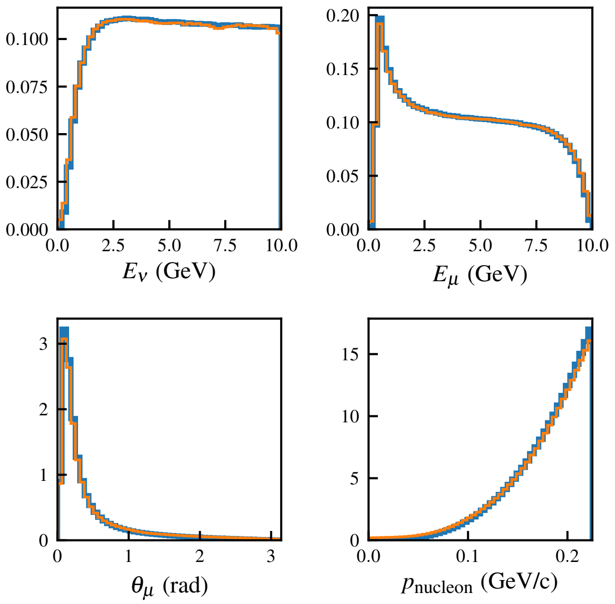


Fig. 7.4: $p\left(\mathbf{x}\right)$ (blue) vs $q_\phi\left(\mathbf{x}\right)$ (orange) 1-dimensional normalized histograms of the marginalized CCQE cross section density for each of the variables. The plots show light discrepancy in each variable, but an overall agreement between the NSF proposal $q_\phi\left(\mathbf{x}\right)$ and the CCQE cross section density $p\left(\mathbf{x}\right)$. Notice how the distribution of $q_\phi\left(\mathbf{x}\right)$ are taken before performing rejection sampling on it.

of magnitude. The mean for $\log_{10} w_{q_\phi}$ is 0.023 ± 0.040, while for $\log_{10} w_{p_{\text{Unif}}}$ we obtain an average of 0.85 ± 0.88, indicating a huge fluctuation in the magnitude of the weights.

The results of the performance test for rejection sampling are summarized in Tab. 7.1, where we compare various quantities for the NSF $q_\phi\left(\mathbf{x}\right)$ and uniform $p_{\text{Unif}}\left(\mathbf{x}\right)$ proposal functions. For this, different quantiles for the constant k for the rejection method are used, following Eq. (7.4), relaxing its restriction

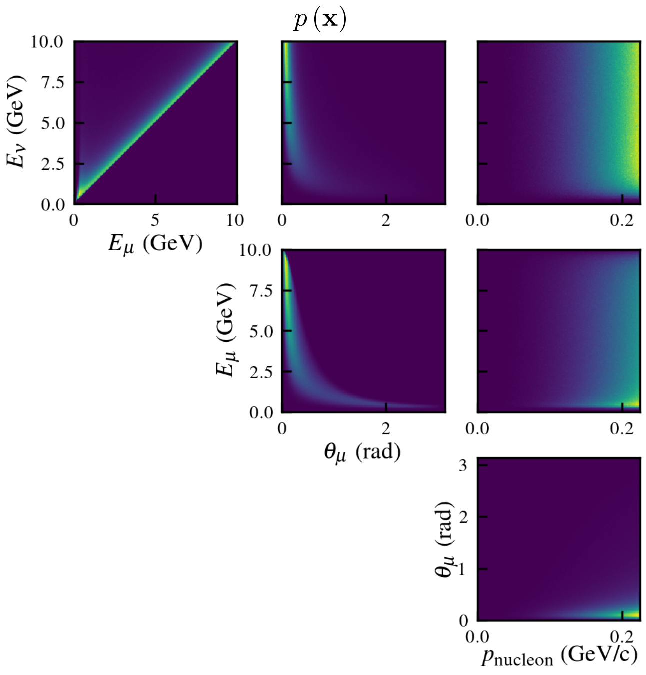


Fig. 7.5: 2D histograms of the cross section density for the real cross section $p(\mathbf{x})$.

as discussed in Sec. 7.2.4. The quantiles for k were chosen using the ten million weights computed for the previous discussion of the weight magnitudes, as well as the probability of acceptance, the coverage, and the effective sample size. We considered a case of sampling one million accepted samples via rejection sampling, where samples from the proposal were generated and checked for acceptance/rejection in parallel, in batches of 300 000 samples. The purpose of the parallelization is to exploit the computational capacities of a GPU. We denoted each of these batches of generating and checking a cycle of the rejection sampling. The values in Tab. 7.1, for each quantile value and a proposal

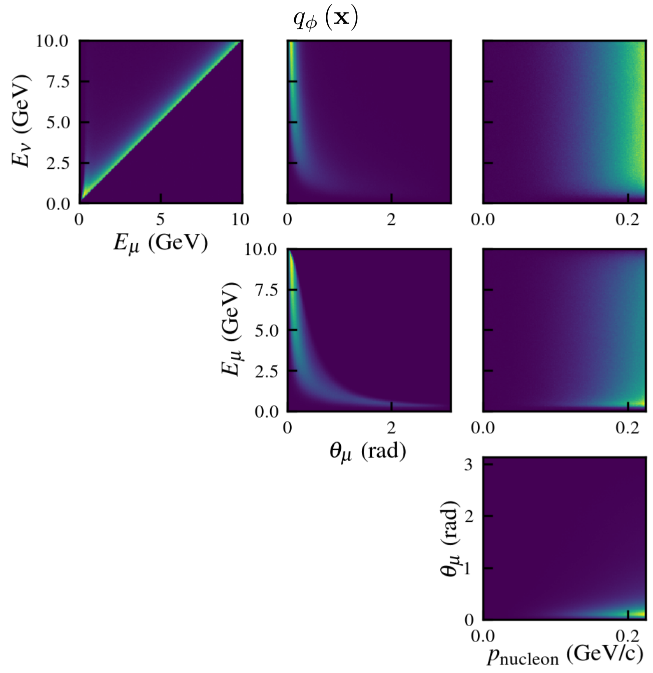


Fig. 7.6: 2D histograms of the cross section density for the proposal density $q_\phi(\mathbf{x})$. Visually, an overall agreement can be seen with Fig. 7.5.

function, are the following:

- p_{accept}: probability of accepting a single event, given by the average of $p(\mathbf{x})/(k \cdot q_\phi(\mathbf{x}))$. If $p(\mathbf{x})/(k \cdot q_\phi(\mathbf{x})) > 1$, it is taken as 1 for the computation.

- Cycles: number of rejection sampling cycles of size 300 000 samples

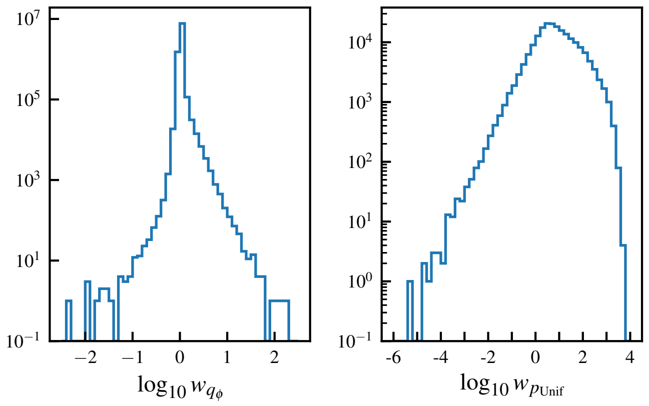


Fig. 7.7: Logarithmic weight distribution for ten million samples from the NSF proposal $q_\phi(\mathbf{x})$, $w_{q_\phi}(\mathbf{x}) = p(\mathbf{x})/q_\phi(\mathbf{x})$, (left) vs the same number of samples from the uniform proposal $p_{\text{Unif}}(\mathbf{x})$, $w_{p_{\text{Unif}}}(\mathbf{x}) = p(\mathbf{x})/p_{\text{Unif}}(\mathbf{x})$, (right). Notice that we are only computing the logarithm for weights > 0. For the NSF, all weights are concentrated around $\log_{10} w = 0$ with a small dispersion around it, while for the uniform distribution the spectrum of weights goes over 9 orders of magnitude.

needed to obtain one million accepted samples:

$$\text{Cycles} = \left\lceil \frac{10^6}{p_{\text{accept}} \cdot 3 \times 10^5} \right\rceil \tag{7.7}$$

- Time: seconds it takes to compute these cycles and obtain one million accepted samples: $t_{\text{cycle}} \cdot \text{Cycles}$.
- Coverage: volume of the original density covering when taking k with a certain quantile (Eq. (7.4)), following Eq. (7.5).
- N_{ESS}/N: the ratio of effective sample size over the total number of samples, quantifying an estimate of the ratio of independence of the events. This was computed for a sample size of 10 million.

Tab. 7.1: Performance values for different quantile choices of k for rejection sampling, as discussed in Sec. 7.2.4, comparing both NSF $q_\phi(\mathbf{x})$ and uniform $p_{\mathrm{Unif}}(\mathbf{x})$ proposal functions. For this exercise, one million samples were generated, performing rejection sampling in batches of 300 000 tries. The quantities are the probability of accepting a single sample p_{accept}, the number of rejection cycles (batches of 300 000) used to obtain one million accepted samples, the time it took in seconds to generate these accepted samples, the coverage of the target density for that particular quantile (Eq. (7.5)) and the ratio of effective sample size N_{ESS} (Eq. (7.6)) over the total number of samples N.

Quantile	Prop.	p_{accept}	Cycles	Time (s)	Coverage	N_{ESS}/N
1.00000	NSF	0.0051	649	201.822	1.0000	0.9140
	Unif.	0.0002	16199	47.886	1.0000	0.0016
0.99999	NSF	0.0633	53	16.482	0.9999	0.9242
	Unif.	0.0004	8056	23.814	0.9947	0.0017
0.99990	NSF	0.1623	21	6.530	0.9996	0.9284
	Unif.	0.0008	4240	12.534	0.9480	0.0020
0.99900	NSF	0.3590	10	3.110	0.9984	0.9338
	Unif.	0.0027	1217	3.598	0.6185	0.0045
0.99000	NSF	0.7187	5	1.555	0.9939	0.9400
	Unif.	0.0137	244	0.721	0.0818	0.0154
0.98500	NSF	0.7730	5	1.555	0.9927	0.9405
	Unif.	0.0171	195	0.576	0.0325	0.0179
0.98100	NSF	0.7968	5	1.555	0.9920	0.9408
	Unif.	0.0193	173	0.511	0.0039	0.0195

The probability of acceptance, p_{accept}, for the NSF is at least one order of magnitude higher than the one obtained from uniform sampling. Additionally, NSF grows rapidly towards ~ 70 % acceptance while also covering > 99 % of the original density volume, as shown in the Coverage column. This is not the case for the uniform distribution, which, while being only one order of magnitude behind NSF with regards to acceptance, is missing a large volume of coverage of the original density.

The number of rejection sampling cycles needed to achieve the desired number of accepted samples is inversely proportional to p_{accept}, as shown in

Eq. (7.7). In a cycle, the algorithm has to sample from the proposal and evaluate both the proposal and $p(\mathbf{x})$. For the NSF, the cycles get stalled when reaching a high percentage of acceptance, since the number of cycles has to be a whole number, which is equivalent to the whole number $+1$. Notice how the number of cycles for the NSF is two orders of magnitude smaller compared to the uniform one, however, the coverage of the uniform drops drastically when decreasing the quantile, and hence the quality of the samples. We will discuss more in-depth in Appendix B.

When looking at the time it takes to obtain one million accepted samples, it is directly proportional to the cycles for each proposal. The main difference is that a cycle for the uniform proposal takes a fraction of the time of a cycle for the NSF. This is because sampling and evaluating for the NSF is heavy computationally compared to doing this task for a uniform distribution. As mentioned before, see Appendix B for a more in-depth discussion.

The coverage is the main quantity of measurement of the quality of the produced samples since it measures the volume conserved of the original distribution when performing rejection sampling with certain quantiles. For all the chosen quantiles, the NSF drops a volume < 1 %, while for the uniform distribution the loss is of > 5 % for quantile 0.9999, > 38 % for 0.999, and > 91 % for 0.99, which is unacceptable when trying to produce samples from the original distribution. For the NSF this level of performance when taking the above quantiles is expected, as in Fig. 7.7 we have seen that the upper tail of weights with large magnitudes is a small percentage of the whole distribution. However, for the uniform distribution, the loss of coverage is caused by two facts: (i) 98.03 % of the weights are zero, hence placing the whole distribution on a 1.97 % of the weights. (ii) These weights, as seen in Fig. 7.7, span over many orders of magnitude, making a cut on the quantile of their distribution more noticeable, as will be discussed below.

To visualize the coverage and the regions missing by choosing a quantile $k_{\mathrm{Q}0.999}$ and different proposal functions, Figs. 7.8 and 7.9 show 2-dimensional histogram representation of the marginalized coverage bin-to-bin, taking the variables in pairs, where each bin quantifies the coverage of that bin (i.e., the sum of weights in that bin after choosing a certain quantile over the sum of weights of those weights without clipping). On the left, the coverage for the NSF is presented and shows that only a few regions of the phase space have values smaller than 1, and even in those regions the coverage has no noticeable

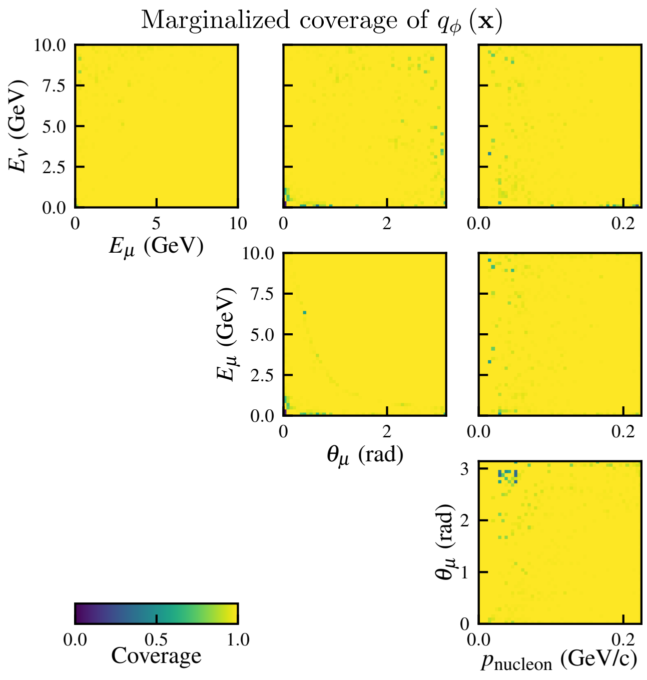


Fig. 7.8: 2D histogram representation of the marginalized coverage bin-to-bin for NSF proposal for k-quantile= 0.999.

discrepancies. On the right, the coverage of the uniform proposal is shown for the same quantile 0.999, marking clear regions where the coverage drops drastically to values close to zero.

When comparing both coverage regions in Figs. 7.8 and 7.9, a clear pattern is seen for the uniform one, while it looks quite random for the NSF. This is because the coverage is related to the ratio $p(\mathbf{x})/q(\mathbf{x})$, with $q(\mathbf{x})$ the corresponding proposal density. For the NSF, $q_\phi(\mathbf{x})$ has a shape very closely related to $p(\mathbf{x})$, as shown in Fig. 7.4 and Figs. 7.5 and 7.6, so the coverage would correspond to regions where the discrepancy is large, which has a noisy behavior.

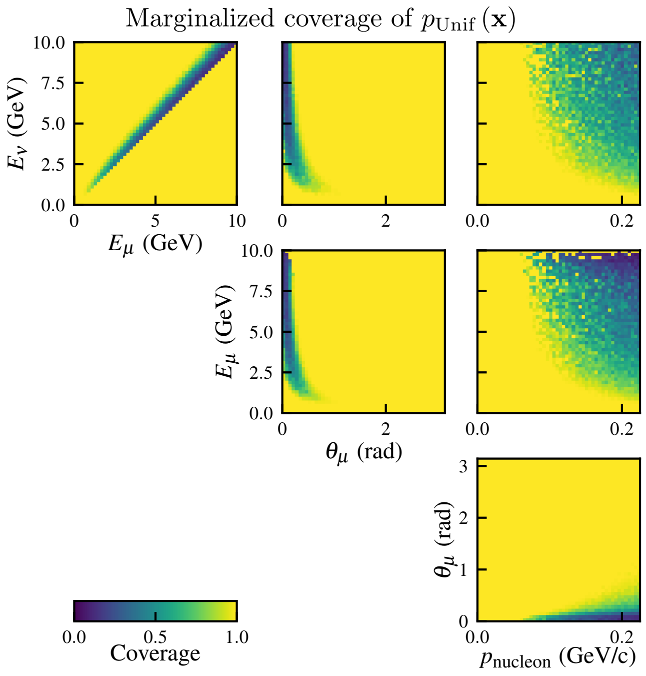


Fig. 7.9: 2D histogram representation of the marginalized coverage bin-to-bin for the uniform proposal, for k-quantile$= 0.999$. When comparing to the NSF proposal in Fig. 7.8, the coverage of the NSF presents a negligible discrepancies in small areas, justifying the use of a quantile for k to improve acceptance and time, as shown in Tab. 7.1. For the uniform proposal, the coverage presents an important size of the total area with significant low coverage, which is unacceptable when trying to perform rejection sampling from it.

Conversely, for the uniform proposal, $p_{\mathrm{Unif}}(\mathbf{x})$, this ratio is proportional to $p(\mathbf{x})$, hence, by clipping, we are doing so according to that particular shape, making the coverage less chaotic and more structured. This translates into making highly probable areas equally likely than others with less probability, affecting this exact group of regions as we will now analyze.

Figs. 7.8 and 7.9 give us an overall picture of where the densities are wrongly estimated by choosing certain quantile, but it does not quantify or indicate the amount of error, that is, it is not telling us whether the coverage is poor in areas of small or high density. To answer this question, a multidimensional histogram over all four dimensions was performed, with 20 bins in each dimension. Then, for each bin, we compute the percentage of weight for a proposal q,

$$\%\ w_q \text{ of bin } = \sum_{\mathbf{x}\in\mathrm{bin}} w_q(\mathbf{x}) / \sum_{\mathbf{x}} w_q(\mathbf{x}),$$

which is equivalent to the percentage of density $p(\mathbf{x})$ in that bin, and the coverage for the quantile $k_{\mathrm{Q}0.999}$. Fig. 7.10 shows a histogram of the number of bins according to their $\%\ w_q$ of bin vs their coverage. Notice how $\%\ w_q$ is presented on a logarithmic scale. For the NSF (Fig. 7.10 left), the regions of coverage visibly smaller than one are two to four orders of magnitude smaller in $\%\ w_q$ than the denser high $\%\ w_q$ region on the top right. This means that the areas being misrepresented by taking the quantile $k_{\mathrm{Q}0.999}$ are relatively small. Also, most of the area with coverage < 1 are close to full coverage. Contrary, the uniform proposal (Fig. 7.10 right) shows the coverage dropping for high values of $\%\ w_q$, indicating that important regions of the original density are being trimmed down by choosing $k_{\mathrm{Q}0.999}$. This observation is in agreement with the total coverage we are seeing in Tab. 7.1.

Concerning the ratio of ESS, we can see that for the NSF it is larger than 90 %, giving highly uncorrelated events. For the uniform proposal however, the ESS drops to the range of $0.16 - 1.95$ %, even for lower quantiles. The differences can be understood from Eq. (7.6) and by looking at the weight distribution for each proposal in Fig. 7.7. A large percentage of NSF weights have the same order of magnitude while for the uniform we go over a spectrum of 8 orders of magnitudes. Additionally this ratio of ESS has to be considered for the area of the original distribution given by the coverage, where the uniform distribution poorly reproduces important regions of phase space by choosing smaller quantiles.

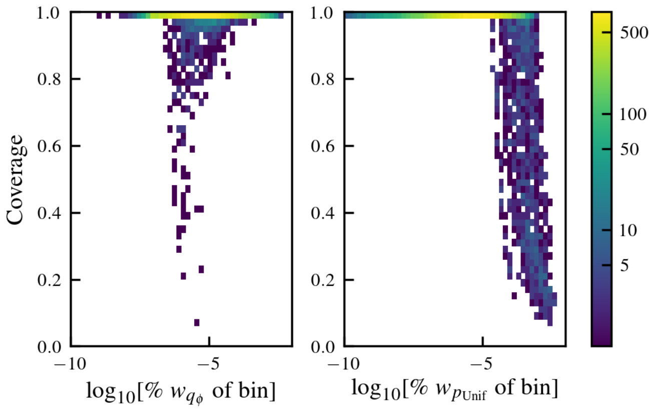


Fig. 7.10: Representation of the number of bins according to their $\% w_q$ of bin vs their coverage for four-dimensional bins in $p\,(\mathbf{x})$, taking $k_{\mathrm{Q}0.999}$. For the NSF (left), the coverage is close to one in most of the bins, and when it drops it is for low $\% w_q$. Contrary, for the uniform proposal (right), the coverage drops for high $\% w_q$, making the overall coverage way smaller than the one for NSF, as shown in Tab. 7.1.

To summarize the analysis performed on the results of Tab. 7.1, in the case of the NSF, by lowering the quantile, the probability of acceptance grows until reaching almost 80 %, reducing the time to a 0.7 % of the original one while maintaining coverage of over 0.99 %. These scores allow us to justify using a smaller quantile for rejection sampling to improve significantly the performance in time, while also quantifying the misrepresentation we are doing by lowering the constant k. On the contrary, for the uniform proposal, the analysis showed weakness when trying to utilize smaller quantiles, lowering the coverage by over 38 % when using a quantile of only 0.999. Additionally, looking at the N_{ESS} and Fig. 7.7, we can see that most of the distribution is concentrated in a relatively small number of samples.

7.4 Conclusions

In this chapter we have presented exhaustive neural importance sampling (ENIS), a framework to find accurate proposal density functions in the form of normalizing flows. This proposal density is subsequently used to perform rejection sampling on a target density to obtain MC datasets or compute expected values. We argue that ENIS solves the main issues associated with rejection sampling algorithms as described in the introduction: (i) The training to find a good proposal is done automatically from the target density, with little configuration needed from the human point of view. (ii) Compared to generic proposal functions such as the uniform one, the normalizing flow adapts its density over the target one, getting rid of the inefficiencies which are usually on the downside of the method. (iii) The proposal function is generated based on a set of trivial normally distributed random numbers transformed through the flow, without any rejection method applied.

The performance of the method has been demonstrated and analyzed in a real case scenario, the CCQE cross section of the antineutrino interaction with a nucleus, where the density is four-dimensional. We have shown that, for the normalizing flow proposal, we can relax the condition in the constant k, used to construct the comparison function, boosting greatly the efficiency (up to ≈ 80 % of acceptance rate) while committing a very small error on the target density (less than a 1 %), bringing orders of magnitude of speed up in computing time compared to the same error committed by a uniform proposal. Additionally, we investigated the coverage of the generation method as a function of the constant k. We showed that the areas of the phase space where the error is committed are less relevant to the final result compared to the error in the uniform case.

The method can be used to generate fast MC samples in cases where the precision is less relevant versus the algorithm speed. High Energy Physics presents some of these examples such as extensive statistical studies based on "Asimov datasets", fast detector simulations, or simply in fast studies for detector developments and designs. In those cases, the learned proposal function might be sufficiently precise and easy to generate with a set of simple normal random generators transformed through the flow.

Regarding its usage, ENIS brings the possibility of applying the same normalizing flow for rejection sampling of similar densities, e.g., densities coming from a model where the parameters are changed slightly, altering the overall density smoothly. The weight distribution of the ratio between target and proposal will be altered, but no additional training of the neural network would be needed regarding the theoretical model remains similar to the original one. This is a significant advantage compared to methods like MCMC, where one would have to rerun the complete algorithm to obtain samples from each of the different densities.

As a last remark, we have demonstrated the potential of ENIS for the four-dimensional CCQE interaction density. We believe this will only be more noticeable when applying it to higher dimensions, as the original paper of NSF [131] shows a remarkable performance on spaces of dimensions up to sixty-three. The advantages will be also more obvious as the underlying cross section model becomes more complex and computationally involved.

8. GRAPH NEURAL NETWORK FOR 3D CLASSIFICATION OF AMBIGUITIES AND OPTICAL CROSSTALK IN SCINTILLATOR-BASED NEUTRINO DETECTORS

Submitted to Physical Review D on the 1st of September 2020 (preprint arXiv:2009.00688 [19]).

Long-baseline neutrino oscillation experiments [172,192,193], introduced in Chapter 2, use two detectors to characterize a beam of (anti-)neutrinos: a near detector, located a few hundred meters away from the target that measures the original beam composition, and a far detector, located several hundred kilometres away, that allows for the determination of the beam composition after neutrino flavor oscillations.

The energy of these beam neutrinos ranges from a few hundred MeV up to several GeV. Charged particles can be produced in neutrino interactions, and the energy that they deposit as they traverse the detector can be used to reconstruct the events. In general, the larger the energy transferred from the neutrino to the nucleus, the larger the number of particles and particle types produced in the final state. Modeling nuclear interactions in the target nuclei is highly complex, particularly for high energy transfers where the hadronic component of the interaction is more important. As a result, current long-baseline neutrino oscillation experiments mostly analyze interactions with low particle multiplicity. This situation, however, is expected to change in the coming years. On one hand, the statistical and systematic uncertainties of current experiments have decreased significantly over recent years such that neutrino-nucleus modeling is becoming a dominant source of uncertainty [172, 194]. On the other hand, future experiments like DUNE [3] will use a broad-band energy neutrino beam, expecting a significant fraction of the neutrino interactions to have a high energy transfer to the nucleus.

As a result, in recent years, the neutrino physics community has turned its attention to measuring neutrino-nucleus interaction cross sections for different ranges of energies and target materials [170] as a way to constrain the oscillation uncertainties while providing new measurements to further develop the interaction models. In parallel, a new generation of neutrino detectors are under development that aim to resolve and reliably identify short particle tracks even in very complex interactions. To achieve this, two main detector technologies stand out: one is based on Liquid Argon Time-Projection-Chambers (LArTPCs) [195] and the other is based on finely segmented plastic scintillators with three readout views [196] that will form part of the near detectors for T2K [197] and, possibly, DUNE [198].

For the latter, the detector response to a charged particle is read out into three orthogonal 2D projections. When reconstructing the 3D neutrino event, different types of hits are rebuilt, introducing non-physical entities that can hinder the reconstruction process. Due to the spatial disposition of such hits, an approach of utilizing graph neural networks (see Chapter 5) is proposed to perform the classification of 3D hits to provide clean tracks for event reconstruction.

The chapter proceeds in the following way: Sec. 8.1 describes properly the motivation behind the methodology given the details of the detector technology. Section 8.2 provides the background of deep learning techniques in neutrino detectors as well as justifying the use of GNNs for the concrete problem at hand. The simulated data samples and GNN training are discussed in Sec. 8.3. Results and a study of systematic uncertainties are given in Secs. 8.4 and 8.5, respectively, followed by concluding remarks in Sec. 8.6.

8.1 Motivation

A finely segmented scintillator detector consists of a 3D matrix of plastic scintillator cubes. The scintillation light produced by charged particles traversing the cubes is read out by three orthogonal wavelength-shifting (WLS) fibers that transport the scintillation light out of the detector where silicon photomultipliers (SiPMs) convert it into a certain number of photoelectrons (p.e.), as illustrated in Figs. 8.1 and 8.2.

Here, we consider the Super Fine-Grained Detector (SuperFGD) [197], a future upgrade to the current FGD subdetector at ND280 in T2K (see Sec. 2.3),

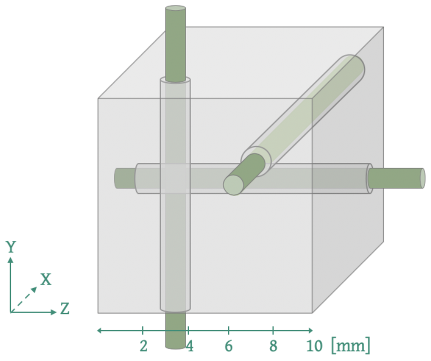


Fig. 8.1: Geometry of a single SuperFGD element.

as a specific case-study. The detector contains 192×56×184 plastic scintillator cubes, each 1×1×1 cm^3 in size, and provides three orthogonal 2D projections of particle tracks produced by a neutrino interaction, as depicted in Fig. 8.4a.

To reconstruct neutrino interactions in three dimensions, the light yield measurements in the three 2D views are matched together, as shown in Fig. 8.4b. The 3D objects, corresponding to the cubes where the energy deposition is reconstructed, are referred to as *voxels*. In addition to the cubes where a particle has passed and deposited energy, light-leakage between neighboring cubes can create additional *crosstalk* signals [15, 199], as depicted in Fig. 8.2. Moreover, ambiguities in the matching process can give rise to *ghost* voxels, shown in Fig. 8.3.

To accurately reconstruct neutrino interactions in these detectors, it is crucial to be able to classify each voxel as one of the three types:

- **Track**: a real energy deposit from a charged particle, henceforth referred to as *track* signals.
- **Crosstalk**: a real energy deposit from light-leakage between neighboring cubes.
- **Ghost**: fake signals coming from the ambiguity when matching the three 2D views into 3D.

Figure 8.4c shows the three types of voxels using truth information after 3D matching has been performed for an example neutrino interaction. Once these voxels are classified, the ghost voxels can be removed before the full event reconstruction proceeds, while simultaneously cleaning the particle tracks of crosstalk.

In this chapter, we represent the voxels as nodes in a graph and classify the signals using a deep learning technique based on a GNN. The abstract data representation provided by graphs makes this method very versatile and applicable to any experiment where the output data from the detector elements can be represented as a list of features with arbitrary dimensionality.

8.2 Background

Deep learning techniques are now commonly applied within the field of neutrino physics. In particular, CNN [143] algorithms that operate on two-dimensional images of the neutrino interactions have been very successful in a number of tasks, such as event classification [3, 200–202] and pixel-level identification of track-like (linear) and shower-like (locally dense) energy deposits [203, 204]. However, images of neutrino interactions are typically very sparse as only those readout channels with a detected signal give rise to pixels with non-zero values, and in the case of the detector presented in Sec. 8.1 the average occupancy of the detector for a neutrino interaction is less than 0.02%. Thus, much of the computation time is spent unnecessarily applying convolutions to a large number of pixels with zero values.

The goal of this work is to classify 3D voxels as one of three categories (track, crosstalk or ghost), which is a natively three dimensional problem. To apply a 3D CNN-based algorithm to this detector would require two million voxels to avoid any downsampling or cropping of the input data, which is computationally prohibitive. We here investigate a sparse data representation, where voxels are represented as nodes in a graph (see Sec. 5.1). Figure 8.5 shows a comparison of the 3D CNN and graph data structures, as well as the radial search method used for defining edges between nodes.

As mentioned above, each detector cube is represented as a node in a graph, and each node consists of a list of input variables called features that describe the physical properties of the detected signal (see Section 8.3 and Ap-

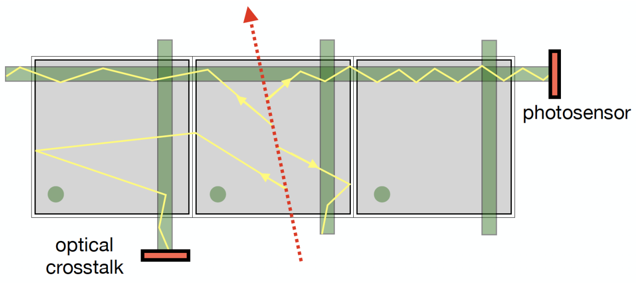


Fig. 8.2: Sketch of the signal generation, fiber transport and signal detection processes highlighting the production of optical crosstalk signals.

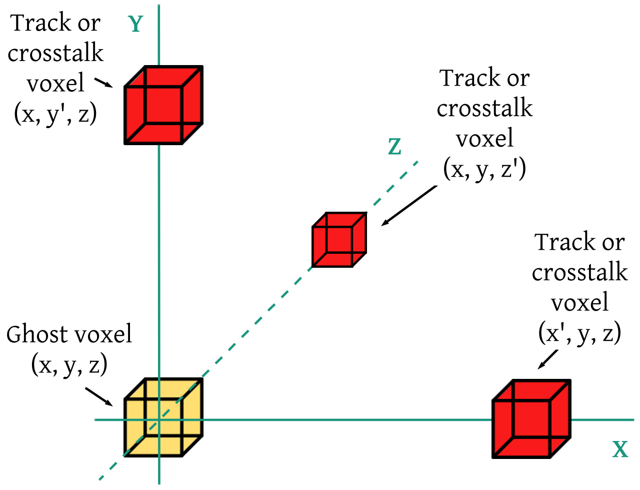
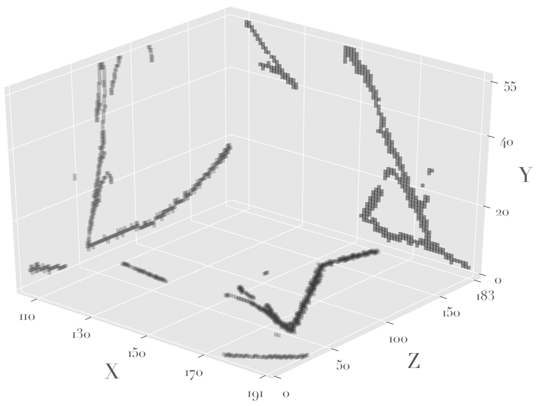
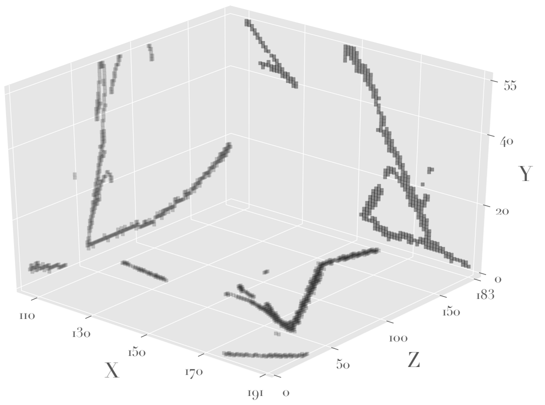

Track or crosstalk voxel (x, y', z)
Y
Z
Track or crosstalk voxel (x, y, z')
Track or crosstalk voxel (x', y, z)
Ghost voxel (x, y, z)
X

Fig. 8.3: Example of a ghost voxel appearance. When matching the coordinates of the fibers that recorded energy deposition, voxels may appear where no true signal exists, becoming non-physical ghost voxels.

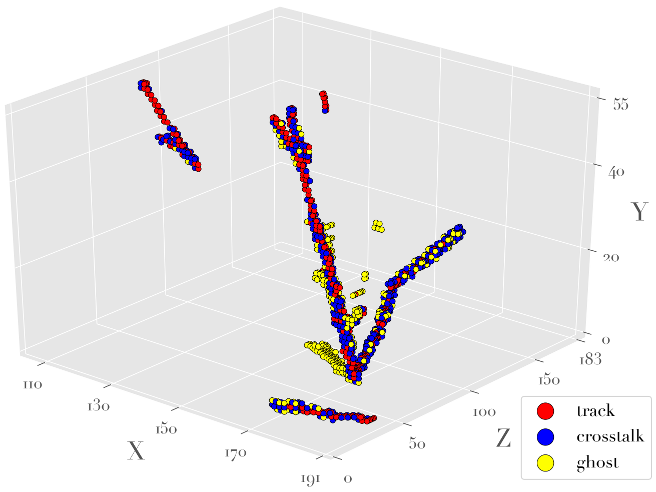


(a) Projections of the observed neutrino interaction onto the three 2D detector views (XY, XZ, and YZ).



(b) 3D view of the neutrino interaction after the 3D matching of the three 2D views. The 3D voxels are shown as dark points. Projections of the observed neutrino interaction onto the three 2D detector views (XY, XZ, and YZ) are shown as shadow.

Fig. 8.4: Visualization of a neutrino interaction in a finely segmented 3D scintillator detector, demonstrating the relationship between the observed 2D projections onto the three orthogonal 2D views, the reconstructed 3D voxels and the true classification of the voxels.



(c) 3D view of the neutrino interaction after the 3D matching of the three 2D views. The 3D voxels are labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation. Projections of the observed neutrino interaction onto the three 2D detector views (XY, XZ, and YZ) are shown as shadows.

Fig. 8.4: Visualization of a neutrino interaction in a finely segmented 3D scintillator detector (cont.).

pendix C.1). The deep learning algorithm that operates on graphs is the Graph Neural Network (GNN), and in this study, a GNN inspired by the GraphSAGE algorithm (Sec. 5.2) is used to classify individual voxels in SuperFGD events. The application of GNNs to data from neutrino experiments has been recently demonstrated by the IceCube experiment in order to identify entire events as atmospheric neutrino interactions, outperforming a 3D CNN [164]. Other GNN-based studies have been performed for particle reconstruction in high energy physics detectors [205–207]. To the best of our knowledge, the approach we present in this work is the first attempt to use GNNs for node classification in neutrino experiments.

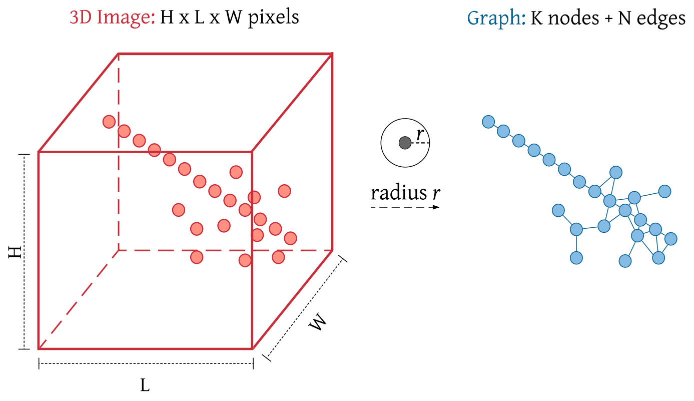


Fig. 8.5: Data and computation size comparison between a 3D image and a graph. The size of the 3D image on the left is fixed ($H \times L \times W$) regardless of the number of hits as CNNs require fixed image sizes. The connected graph shown on the right is a much more efficient representation of the data. Each hit is represented as a graph node and connections, called edges, are made between neighboring hits within a sphere of radius r.

8.3 Methodology

This Section contains the different processes needed to properly simulate the detector data, define a GNN model and train it, before proceeding to the results discussion in Sec. 8.4. For this, in Sec. 8.3.1 the data sampling of the detector is described, forming two datasets containing different physics. Section 8.3.2 formalizes the precise GNN model utilized to classify the voxels, while Sec. 8.3.3 shows the training procedure of the model for the two datasets.

8.3.1 Data sample generation

In order to generate datasets of neutrino interactions with true labels that allow training and benchmarking the classification algorithm, the steps below are followed. For each neutrino interaction:

1. Initial particle types and initial kinematics are specified for all final-state

particles produced in the interaction.

2. Initial particles are propagated through the detector geometry producing further particles and leaving signals in the form of energy deposits.

3. Using particle energy deposits, the detector response is simulated.

4. The information is stored as a list of voxels with a known true label.

Initial particle types and kinematics
The initial particle types and their associated kinematics were simulated following two approaches. Firstly, GENIE datasets were created using `GENIE-G18.10b` neutrino interaction software [208]. For a given neutrino flux[1] and target geometry specification, it generates a list of realistic neutrino event interactions both in the number and type of outgoing particles, often referred to as event topologies, and in their individual initial kinematics. Secondly, particle-gun (Pgun) (1-track) and particle-bomb (P-Bomb) (multi-track) datasets were constructed to be as complementary as possible to the GENIE datasets by minimizing the modeling dependency. The number of initial particles and their types were fixed in each of these datasets and their input kinematics were chosen to be randomly and uniformly distributed in the range 10-1000 MeV/c. A summary regarding the number of events and voxels in the two datasets, as well as of the class distribution is presented in Tab. 8.1.

Particle propagation simulation in the detector
The SuperFGD detector geometry was simulated as described in Ref. [197]. The particle propagation and physics simulation is done by means of `GEANT v4-10.6.1` [210]. GEANT is a Monte Carlo based toolkit that provides realistic propagation of particles through matter. It outputs a list of energy deposits.

All energy deposits[2] occurring in the same detector cube, including the effect of Birks' quenching [211], are summed to form the list of *track voxels*. To simulate imperfect cube light-tightness, the 3D voxelized energy is then randomly shared ($\mu = 2.7\%$) with the neighboring cubes, creating a new set

[1] We used the T2K flux, which peaks at 600 MeV/c, see Ref. [209].

[2] Only signals in the first 100 ns are considered. Further delayed signals, such as decays, can be treated as independent graphs.

GENIE dataset		**Training**	**Validation**	**Testing**
	# Events	6k	2k	11.5k
	# Voxels	1.83M	606.7k	3.58M
		Track	**Crosstalk**	**Ghost**
	Fraction	43%	37%	20%
P-Bomb dataset		**Training**	**Validation**	**Testing**
	# Events	6k	2k	39.5k
	# Voxels	1.84M	618k	12.3M
		Track	**Crosstalk**	**Ghost**
	Fraction	49%	38%	13%

Tab. 8.1: Descriptions of both GENIE and P-Bomb datasets, displaying the number of events and number of voxels used for training, validating and testing the models. Additionally the fractions of the different classes of voxels are shown.

of voxels that originally had no energy deposits, the *crosstalk voxels* (see Figure 8.2). Then, the 3D voxelized energy of both track and crosstalk voxels is projected onto its three orthogonal planes where the detector 2D signals are simulated, converting the continuous energy deposit into discretized photons, weighted by distance-dependent attenuation factors, which are detected with 35% probability. To mimic a minimum threshold detection sensitivity, only 2D hits with three or more detected photons are kept. Then, the 2D hits are matched into 3D reconstructed voxels only if the same XYZ coordinate combination can be made using two different combinations of 2D planes. In this process, due to ambiguities some extra voxels are created, the *ghost voxels* (see Fig. 8.3). Finally, those track and crosstalk voxels not reconstructed after the 3D matching are discarded from the original lists. An example of the 2D to 3D reconstruction is shown in Figures 8.4a and 8.4b.

Simulation output

The resulting output from the simulation is a list of voxels and their associated energy deposits in the three planes, each with one of the following three labels that we want to classify, as described in Sec. 8.1: track, crosstalk or ghost voxel.

8.3.2 Network architecture

Each graph in GraphSAGE is constructed using the proximity of two voxels in that graph. If both voxels are spatially located within a radius of $1.75\,\mathrm{cm}^3$, then we consider them to be connected in the graph by an edge; we repeat the same procedure for each pair of voxels[4]. Additionally, we consider a neighborhood depth of three, i.e., to produce the embedding of a voxel, we use the voxel features together with its first neighbors' features, the features of the neighbors of its neighbors, i.e, second neighbors' features, and the features of the neighbors of the neighbors of its neighbors, i.e., third neighbors' features. The aggregator used to combine the features of the neighbors is the mean aggregator, which produces the average of the neighbors' values. This final embedding is then passed to an MLP consisting of two fully connected layers - each followed by a LeakyReLU activation function, and a final output layer followed by a softmax activation function. Figure 5.2 illustrates the GraphSAGE-based approach used, while Tab. 8.2 shows the architectural parameters chosen. Categorical cross-entropy is chosen as the loss function to minimize during training, as it is considered the standard one for multi-class classification problems:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{c} y_j^{(i)} \log \hat{y}_j^{(i)},$$

where:

- $\boldsymbol{y}^{(k)}$: true values corresponding to the k^{th} training example.
- $\hat{\boldsymbol{y}}^{(k)}$: predicted values corresponding to the k^{th} training example.
- m: number of training examples.

[3] To link only those voxels within the 3×3×3 cube of voxels centred on the target voxel (the maximum diagonal distance from the center of this cube is $\sqrt{1^2+1^2+1^2} \approx 1.75$).

[4] If a voxel has no neighbors, it is discarded from the graph and cannot be classified; this happens for less than 0.6% of the total number of voxels.

- c: number of classes/neurons corresponding to the output. In this case, the three classes are: track, crosstalk, and ghost.

Parameter	value
Encoding size	128
Depth	3
Aggregator	mean
Fully Connected Layer 1	128 neurons
Fully Connected Layer 2	128 neurons
Fully Connected Layer 3 (output)	3 neurons

Tab. 8.2: Architectural parameters; for more information about the meaning of the parameters, see Sec. 5.2.

The output layer of the model consists of three neurons, one for each of the three classes, with values v_i for $i = 1, 2, 3$. The sum of neuron values is given by $\sum_{i=1}^{3} v_i = 1$ such that each neuron value gives a fractional score that can be used to classify voxels. In other words, the model returns scores for each voxel to be one of the three desired outputs, which can be interpreted as the probability: track-like, crosstalk-like, or ghost-like.

8.3.3 Training

The network[5] was trained for 50 epochs[6] using Python 3.6.9 and PyTorch 1.3.0 [212] as the deep learning framework, on an NVIDIA RTX 2080 Ti GPU. Adam [78] is used as the optimizer, with a mini-batch size of 32, and an initial learning rate of 0.001 (divided by 10 when the error plateaus, as suggested in [213]). The model has a total of 105,347 parameters. Figure 8.6 shows the validation results during the training process, measured by the F_1-score

[5] We would like to thank T. Jiang, T. Zhao and D. Wang for their implementation of GraphSAGE, on which this work's code is based on: `https://github.com/twjiang/graphSAGE-pytorch`

[6] Epoch: one forward pass and one backward pass of all the training examples. In other words, an epoch is one pass over the entire dataset.

metric:

$$F_1 = 2\,\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

The precision and recall are defined as:

$$\text{precision} = \frac{\text{true}_{\text{positives}}}{\text{true}_{\text{positives}} + \text{false}_{\text{positives}}},$$
$$\text{recall} = \frac{\text{true}_{\text{positives}}}{\text{true}_{\text{positives}} + \text{true}_{\text{negatives}}},$$

where the labels are compared as one class vs. all the others. The model used later for inference on new data is the one that maximizes the F_1-score for the validation set, as it has the best generalization for unseen data.

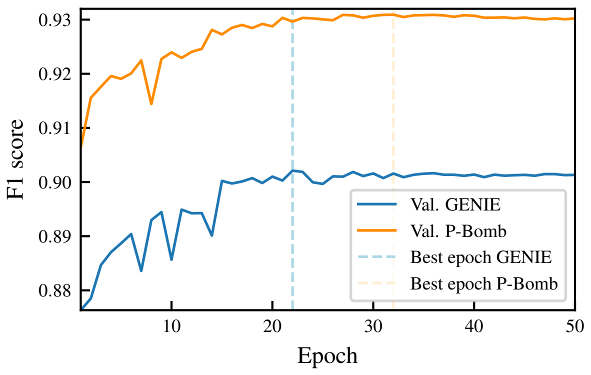


Fig. 8.6: Validation F1 results on GENIE and P-Bomb samples.

8.4 Results

The GNN voxel-type predictions are compared against the true labels to evaluate the network performance and identify possible areas of improvement. Here, we choose the output class with the highest score as the predicted class of each voxel although, depending on the type of analysis, different selection criteria could be applied in the future.

The efficiencies and purities of these predictions are calculated by two methods: per voxel and per event. The latter method evaluates the correctness of predictions on an event-by-event basis, while the former does an overall calculation of the efficiencies and purities for all voxels in all events of the sample. The results of both methods for four sets of training/testing samples are shown in Tab. 8.3, giving nearly identical performance that is independent of the dataset used to train and test the GNN.

As an example, Fig 8.7 shows the voxel prediction results from the GNN when applied to the event shown in Fig. 8.4, a GENIE event that features a track almost completely composed of ghost voxels. Figure 8.7a shows the class predicted for each voxel, while Fig. 8.7b displays which voxels were correctly/incorrectly classified.

A more in-depth analysis of the GNN performance can be carried out by studying the effects of different event properties on the efficiencies and purities of the predictions. For these studies, the results of the GNN trained and tested on the GENIE dataset are used.

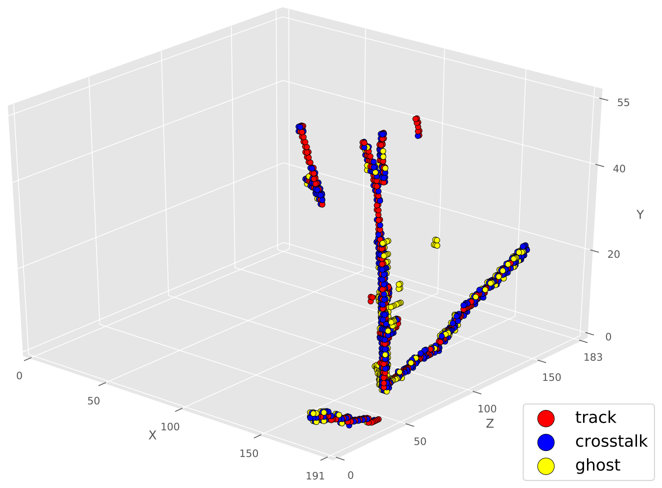
One of the factors expected to affect these predictions is the number of voxels in the event. Figure 8.8 shows the relationship between the mean efficiency and purity per event for each type of voxel as a function of the total number of voxels in the event. The figure also shows the mean number of events in each bin (in light blue). It is clear that both the efficiencies and purities of the three types of voxels decrease as the number of voxels in the event increases. This decrease is coupled with an increase of the fraction of ghost voxels as the total number of voxels increases, which are the hardest for the GNN to classify.

The number of tracks in the event is an estimate of the complexity of its topology. According to Fig. 8.9, the classification efficiencies and purities drop as the number of tracks increases. This behaviour is also correlated with the increasing fraction of ghost voxels in the events.

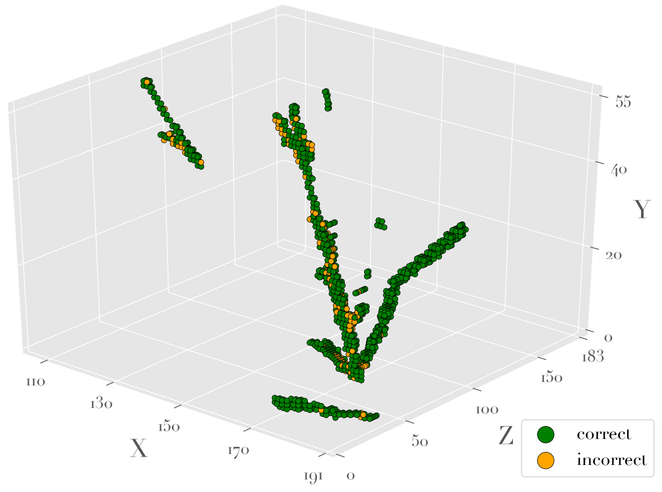
The region around the interaction vertex is of particular interest in the event. It is expected that a high spatial density of voxels within a certain volume of the detector may pose a challenge for the GNN to correctly identify the voxel type. This can be observed by studying the efficiencies and purities as a function of the distance to the interaction vertex, as shown in Fig. 8.10.

		GENIE Training			
GENIE Testing	**Per Voxel**		Track	Crosstalk	Ghost
		Efficiency	93%	90%	84%
		Purity	93%	87%	91%
	Per Event		Track	Crosstalk	Ghost
		Efficiency	94%	94%	88%
		Purity	96%	91%	92%
P-Bomb Testing	**Per Voxel**				
		Efficiency	94%	93%	87%
		Purity	95%	90%	92%
	Per Event		Track	Crosstalk	Ghost
		Efficiency	94%	94%	87%
		Purity	96%	90%	92%
		P-Bomb Training			
GENIE Testing	**Per Voxel**		Track	Crosstalk	Ghost
		Efficiency	93%	89%	80%
		Purity	91%	86%	89%
	Per Event		Track	Crosstalk	Ghost
		Efficiency	94%	93%	88%
		Purity	95%	91%	91%
P-Bomb Testing	**Per Voxel**		Track	Crosstalk	Ghost
		Efficiency	95%	93%	88%
		Purity	95%	91%	92%
	Per Event		Track	Crosstalk	Ghost
		Efficiency	95%	93%	88%
		Purity	96%	91%	92%

Tab. 8.3: Mean efficiencies and purities of voxel classification, calculated for the whole sample (per voxel) and as a mean of the event-by-event efficiencies and purities (per event).

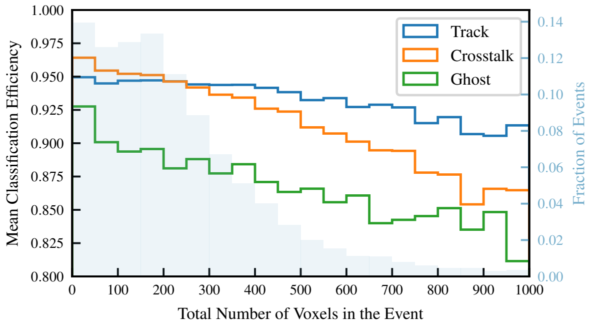


(a) Prediction: voxels are colored based on the GNN predictions.

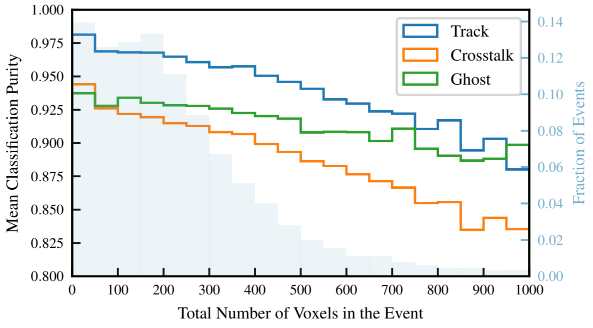


(b) Accuracy: voxels correctly classified by the GNN are shown in green.

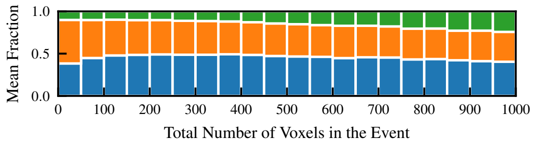
Fig. 8.7: Example GNN prediction results for the interaction shown in Fig. 8.4.



(a) Efficiency.

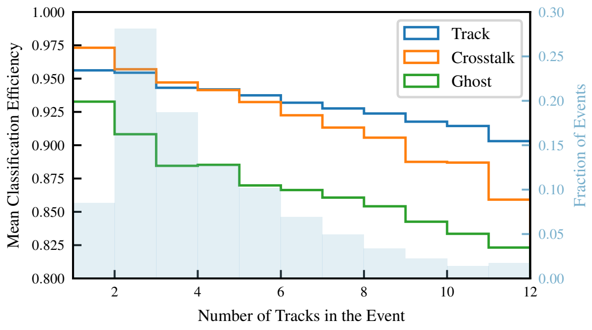


(b) Purity.

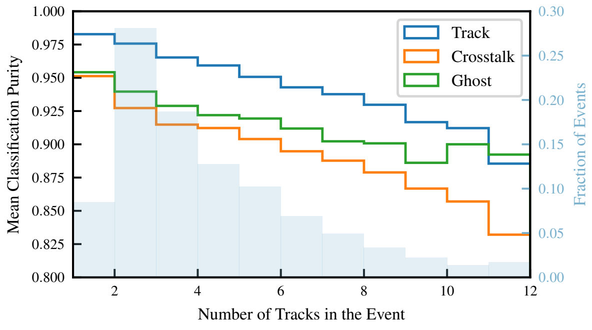


(c) Mean fraction of each type of voxel as a function of the number of voxels in the event (blue = track, orange = crosstalk, green = ghost).

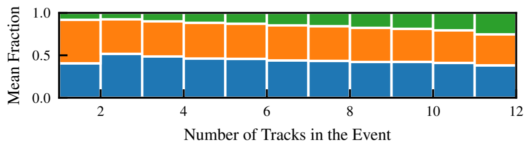
Fig. 8.8: Efficiency and purity as a function of the number of voxels in the event for a sample trained and tested on GENIE simulated data.



(a) Efficiency.



(b) Purity.



(c) Mean fraction of each type of voxel as a function of the number of tracks in the event (blue = track, orange = crosstalk, green = ghost).

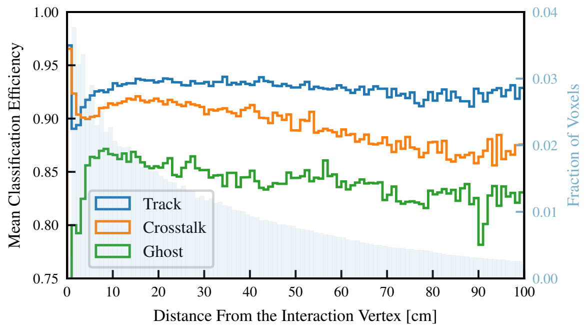
Fig. 8.9: Efficiency and purity as a function of the number of tracks in the event for a sample trained and tested on GENIE simulated data.

At the interaction vertex itself, it is clear that there are only track voxels and the GNN can identify them with over 96% efficiency and 100% purity. The following 2 cm exhibit only a small fraction of ghost voxels, mainly due to the high spatial density of voxels with real signals in that volume, which is mainly occupied by track and crosstalk voxels. As we go further from the vertex, the efficiencies and purities increase up to 10 cm, after which we expect the density of voxels to decrease allowing for more ghost voxels. Therefore, at large distances we observe that the efficiencies and purities tend to decrease, most notably the efficiencies of crosstalk and ghost voxels and the purity of crosstalk voxels.

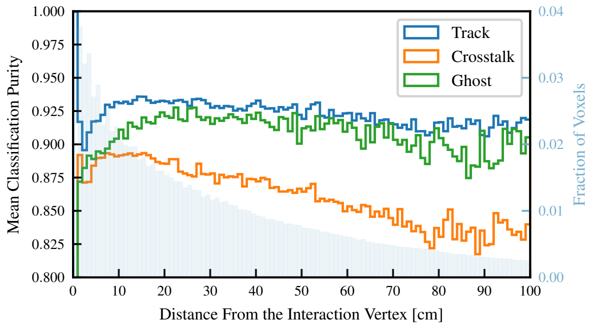
As the main goal of this GNN is to identify ghost voxels in order to eliminate them from the events, it is important to make sure that true track and crosstalk voxels are not lost in the process. According to the tests performed, only 1.1% of all true track voxels and 3.3% of crosstalk voxels are incorrectly classified as ghost voxels by the GNN. In addition, it is important not to miss ghost voxels: the GNN correctly identified 84.5% of all ghost voxels, where 72.1% of those classified incorrectly were predicted as crosstalk. Therefore, although not ideal, this issue is not critical as crosstalk voxels have a smaller influence on future studies than track voxels.

Lastly, we compare the results of the GNN against a conventional method of voxel classification which relies on a charge cut. As described in Appendix C.1, each voxel has three charges that correspond to the signals from the three fibers passing through it. Since other voxels along the same fiber may have signals causing a larger amplitude to be recorded, we consider the smallest of these three charges to be the most accurate estimation of the true voxel charge. Hence, this minimum charge is used for the purposes of this charge cut. Since, by definition, we expect higher energy deposition in track voxels compared to crosstalk and ghost voxels, we set a lower limit for the minimum charge in a voxel such that any voxels with a higher minimum charge than the threshold are classified as track voxels. Fig. 8.11 shows the distribution of the minimum voxel charge for the three types of voxels. From this figure, it is clear that it is not possible to separate ghost from crosstalk voxels. Thus, this classification is only binary such that we have two categories: track or other. We decide to place this cut at 12 p.e., where the track and non-track voxel PDFs intersect.

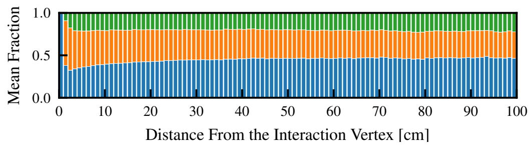
To compare the results of this cut with those of our GNN, we combine the predictions of the crosstalk and ghost categories. Table 8.4 shows the efficiency



(a) Efficiency.



(b) Purity.



(c) Mean fraction of each type of voxel as a function of the distance to the vertex (blue = track, orange = crosstalk, green = ghost).

Fig. 8.10: Efficiency and purity as a function of the distance to the neutrino interaction vertex for a sample trained and tested on GENIE data.

and purity of the classifications for the two methods. It is evident that using only a charge cut can still yield a comparable track voxel classification efficiency to the GNN. However, it struggles to correctly classify non-track voxels which, in turn, reduces the purity of the predicted track voxels.

Another improvement given by GNN over the charge cut is the capability of rejecting "fake" tracks, i.e. a cluster of ghost voxels that closely resembles the structure of a real particle track. Since fake tracks are usually produced by the shadowing of real tracks, the corresponding number of p.e. measured in the three readout views is higher than 12 p.e., hence the charge cut cannot reject them easily. The ability of the GNN to reject ghost tracks is shown in Appendix C.3 for a number of neutrino interactions and compared to the charge cut method.

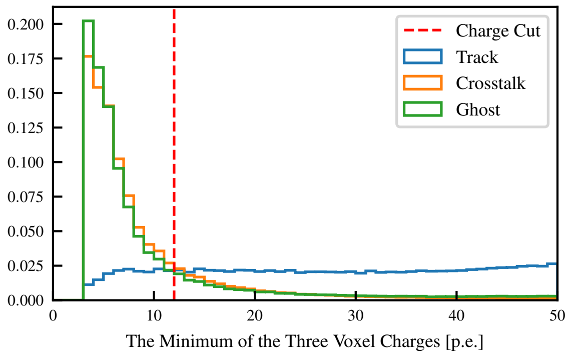


Fig. 8.11: The distribution of the minimum charge among the three voxel charges for the GENIE sample.

Figure 8.12 shows the advantage of the three-fold classification of the GNN over the binary classification of the charge cut when comparing the fraction of true total deposited energy obtained using each method. In the case of the GNN, the total deposited energy in an event is the sum of the true energy deposited in all non-ghost voxels. For the charge cut, only the energy deposited in track voxels is used. This causes an average energy loss of 5% per event

GNN			**Charge Cut**		
	Track	Other		Track	Other
Efficiency	94%	96%	Efficiency	93%	80%
Purity	96%	95%	Purity	80%	91%

Tab. 8.4: Mean efficiencies and purities of voxel classification for the GNN and a simple charge cut.

when using a method that also excludes the crosstalk voxels, compared to less than 1% when using the GNN that can isolate ghost voxels.

8.5 *Systematic uncertainty considerations*

The results presented in Sec. 8.4 show that the GNN is a very powerful technique for removing ghost voxels and identifying optical crosstalk in 3D-reconstructed neutrino interactions. It is important to demonstrate that this technique is robust and does not introduce new systematic uncertainties, potentially given by a sub-optimal choice of the training sample.

One of the main limitations in the measurement of the neutrino oscillation parameters in long-baseline experiments comes from uncertainties in the modeling of neutrino interactions, not yet fully constrained by data and partially incomplete for describing all the details of the interaction final state. For example, the modeling of hadron multiplicity and kinematics may considerably change the image of the neutrino interaction, particularly near the neutrino vertex, or the total energy deposited by all the particles produced by the neutrino interaction. Hence, it is hard to obtain a data-driven control sample to train a neural network without making any prior assumptions. Since the GNN is trained only on a subset of the parameter space, the results could be biased if the detected neutrino interactions belong to a region of the parameter space not well covered by the MC generator. Thus, there must be careful studies of the systematic uncertainties in order to account for a potentially incomplete sampling of the parameter space.

One advantage of GNNs over algorithms like CNNs is that one can control the input variables used for the parameterization.

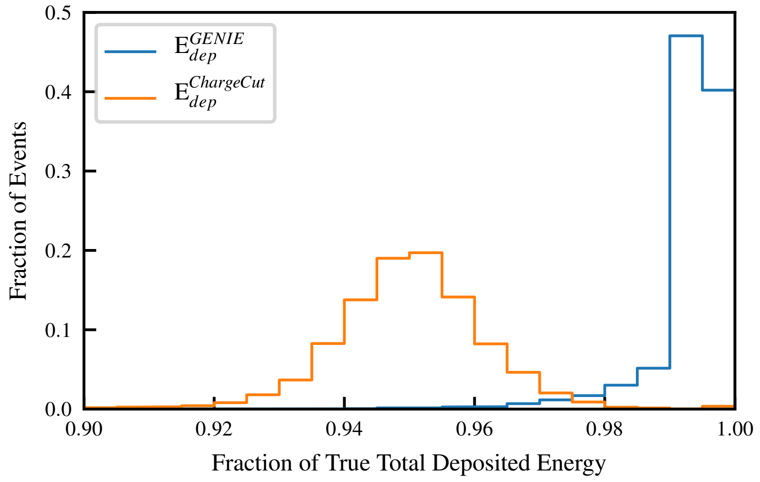


Fig. 8.12: The fraction of the true total deposited energy obtained when using the GNN (trained on GENIE) or the charge cut as a classification method.

In this section, we investigate potential sources of systematic uncertainty.

As described in Sec. 8.4, different training samples, (GENIE or P-Bomb) were generated and the results were summarised in Tab. 8.3, demonstrating that the performance is still very good even when the samples used for training and testing were different.

A few comments about the training sample generation are necessary. Whilst the GENIE sample belongs to a particular choice of the neutrino interaction model, the P-Bomb sample aims, in principle, to be as model-independent as possible. However, generalizing the training sample enough to contain all possible final states of the neutrino interaction is computationally prohibitive. The training sample will always have some limitations from the subjective decisions about which neutrino interaction topologies are sufficiently improbable to be omitted. Thus, it is necessary to adopt a figure of merit to quantitatively evaluate how two training samples differ in terms of sampling of the parameter space. Following the prescription described in Ref. [214], we computed the distance between the correlation matrices of the GNN input variables of the different training samples (GENIE and P-Bomb), defined as

$$d(C_1, C_2) = 1 - \frac{tr\,\{C_1 C_2\}}{\|C_1\| \|C_2\|} \tag{8.1}$$

where C_1 and C_2 are the correlation matrices obtained from two different training samples. In the text we will refer to $d(C_1, C_2)$ simply as to the distance between training samples. Although this method is an approximation to a multivariate Gaussian distribution of the probability density function and does not take into account the scales and means of the variables, it still provides a quick way to understand how similar the training samples are. Nevertheless, other heavily computational methods, such as those involving the difference of probability density function integrals, could be used, but are beyond the scope of this study. The correlation matrices of the features for the GENIE and P-Bomb datasets are presented in Fig. C.1 in Appendix C.2. An "alternative" P-Bomb sample, inclusive of non-physical event topologies (e.g. events not predicted by neutrino event generators) with respect to the other one, was generated for the systematic uncertainty evaluation. However, since the distance between the original and the alternative P-Bomb samples is nearly zero, the latter was not used. In Tab. 8.5, the distances of Eq. (8.1) between the correlation matrices from three generated training samples are shown.

	GENIE	**P-Bomb**	**Alternative**
GENIE	-	0.020075	0.020803
P-Bomb	0.020075	-	0.000136
Alternative	0.020803	0.000136	-

Tab. 8.5: Distance as defined in Eq. 8.1 between the correlation matrices obtained from the different training samples. Details of the generation of the GENIE and P-Bomb samples is described in Sec. 8.3.1. The Alternative sample was built with a particle bomb similar to the P-Bomb sample but with additional event topologies.

The robustness of the GNN against model dependencies can be verified by training different neural networks on different event samples and applying them to the same set of neutrino interactions. A difference in the observables used in the physics measurement, such as particle momenta, energy deposit, etc., obtained by the different training can be assigned as a systematic uncertainty introduced by the method.

A study was performed to evaluate the impact of the method on the total true energy deposited in the detector. The difference between the total energy deposit computed after rejecting the voxels classified as ghosts for both network trainings was computed. Fig. 8.13 shows the distribution of the total true deposited energy before and after discarding the voxels classified as ghosts. Both GENIE- and P-Bomb- trained GNNs give very similar results over the full range of total deposited energy. The total true deposited energy computed with and without ghost rejection differ on average by less than 1 MeV. Hence, it is expected to be improved by increasing the statistics of the training samples. The total difference between GENIE- and P-Bomb- trained GNNs is found to be less than 1 MeV with a standard deviation of approximately 5.5 MeV, mainly due to a few outlier entries, and 68% of the events with a difference better than 0.192 MeV, as shown in Fig. 8.14.

This corresponds to less than 2% of the mean total deposited energy per event. In Fig. 8.15 the impact of the different training sample is shown as a function of the total deposited energy. The fractional standard deviation, defined as the standard deviation of the difference between deposited energy computed from different GNN trainings and divided by the true deposited energy, shown in the bottom panel, is less than 2% and almost constant as a function of the deposited energy. This means that the performance of the method is about the same irrespective of the total deposited energy. This study confirms that GNN can be used for classifying 3D voxels potentially with limited systematic uncertainties in the deposited energy, while drastically improving the tracking capability.

Another potential issue could be given by a mismodeling of the amount of crosstalk. In addition to the nominal optical crosstalk (2.7%), two further datasets were simulated using 2% and 5% crosstalk and the voxel classification was performed using the GNN trained with nominal crosstalk. As shown in Tab. 8.6, the efficiency and the purity is relatively stable even in the case where the crosstalk model is wrong, in particular for identifying track voxels. However, crosstalk can be precisely measured with even small prototypes, thus

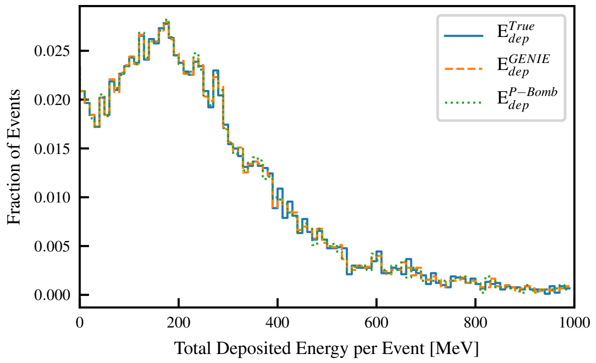


Fig. 8.13: Distribution of the total true deposited energy after rejecting the ghost voxels classified either with GENIE- (dashed orange) or P-Bomb- (dotted green) trained GNNs and without any ghost rejection (solid blue). The mean total deposited energy per event is about 288 MeV.

it is not considered to be a source of additional systematic uncertainty as it can be accurately simulated.

8.6 Conclusions

A graph neural network inspired by GraphSAGE was developed and tested on simulated neutrino interactions in a 3D voxelized fine-granularity plastic-scintillator detector with three 2D readout views. The advantage of this neural network is that, the graph data structures provide a natural representation of the neutrino interactions.

The neural network was able to identify ambiguities and scintillation light leakage between neighboring active scintillator detector volumes as well as real signatures left by particles with efficiencies and purities in the range of 94-96% per event, with a clear improvement with respect to less sophisticated methods. In particular, it can reject fake tracks produced by the shadowing of real tracks observed in the 2D readout views. The performance was tested for

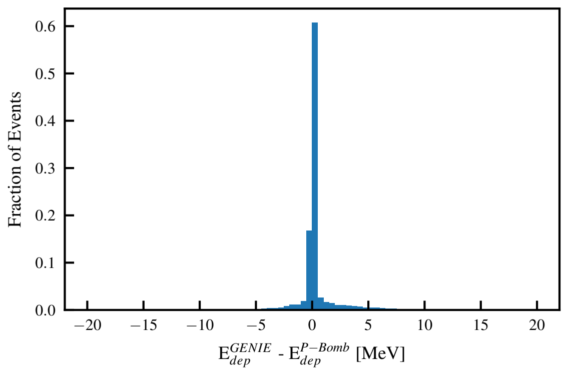


Fig. 8.14: Difference between the total true deposited energy computed after rejecting the ghost voxels classified with GENIE- and P-Bomb- trained GNNs. The mean is 0.78 MeV while the standard deviation is 5.5 MeV. About 40% of events show no difference between P-Bomb and GENIE, 68% have a difference within ± 0.192 MeV, while only 5% of the events have a difference outside the range ± 6.35 MeV.

neutrino events with different number of voxels, number of tracks and voxels at different distances from the vertex, variables that could hint to interaction model dependencies of the method. Efficiencies and purities were found to be relatively stable and the trends were consistent with the expectation. The robustness of the neural network against possible systematic uncertainties introduced by the method was tested. The results were obtained using neural networks trained on different samples, produced either with the GENIE event generator or by randomizing the number of final state particles and relative momentum to obtain a more generic sample that does not belong to any particular theoretical model. It was found that the bias introduced on the total deposited energy of the event by arbitrarily choosing a different training sample is, on average, less than 1 MeV. The impact of potential mismodeling of the light leakage between neighboring scintillator volumes was tested. Results show that the performance of the neural network is robust to expected changes

Crosstalk 2.7% (nominal)			
	Track	Crosstalk	Ghost
Efficiency	93%	90%	84%
Purity	92%	87%	91%
Crosstalk 2%			
	Track	Crosstalk	Ghost
Efficiency	92%	89%	81%
Purity	94%	83%	89%
Crosstalk 5%			
	Track	Crosstalk	Ghost
Efficiency	94%	89%	88%
Purity	86%	91%	93%

Tab. 8.6: Mean efficiencies and purities of voxel classification, per voxel, for different crosstalk values, i.e. 2.7% (nominal) 2% and 5%. The GNN was trained with GENIE training samples with nominal crosstalk and tested on the same GENIE sample with different crosstalk values to study its robustness.

in the crosstalk modeling.

To conclude, we showed that a graph neural network has great potential in assisting a 3D particle-flow reconstruction of neutrino interactions. Similar results may be expected for other types of detectors that aim to a 3D reconstruction of the neutrino event from 2D projections and that share analogous features like ambiguities and leakage of signal between detector voxels.

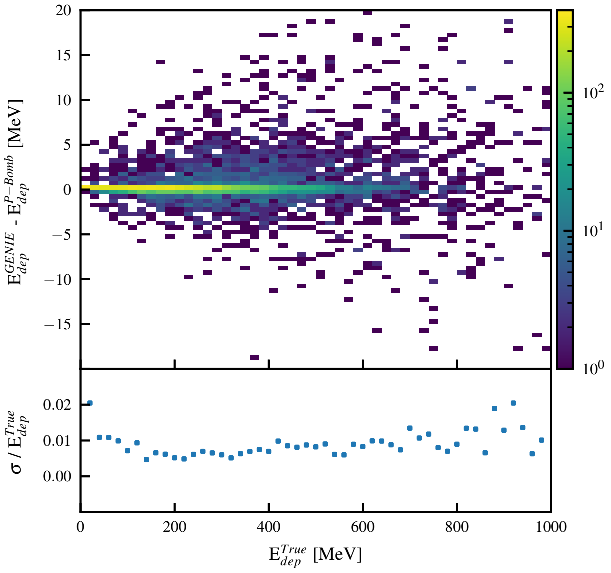


Fig. 8.15: Top: difference between the total true deposited energy computed after rejecting the ghost voxels classified with GENIE- and P-Bomb- trained GNNs as a function of the total true deposited energy. Bottom: fractional standard deviation of the difference of the total true deposited energy computed after rejecting the ghost voxels classified with GENIE- and P-Bomb-trained GNNs as a function of the total true deposited energy.

9. PREDICTIVE ANALYSIS OF THE DAMPING RATE IN INTER-AREA OSCILLATIONS

This chapter contains a descriptive report of an industrial application within the context of the collaboration between Grupo AIA and IFAE during the PhD. Due to business secrets and privacy of the data, the following report does not contain numbers or names quantifying or precisely describing the nature of the given datasets.

The damping ratio of the electric grid will be analyzed as a model of features of the Spanish network, as it has strong implications on potential blackouts (see Sec. 9.1). A first study on the proper nature of the data and a baseline model are proposed in Sections 9.2 and 9.3. Later, in Sec 9.4, a restructuring of the data to apply a graph neural network model is discussed. Finally, the uncertainty of the models is presented in Sec. 9.5 through quantile regression and normalizing flows.

9.1 Problem description

The electric grid in Europe, and in particular in Spain, operates by producing and distributing alternating current (AC) with an oscillating frequency of 50 Hz. The stability of maintaining this precise frequency is crucial for the network to work accordingly and to bring proper electrical power to the end-user. When having such a system with interconnected synchronous generators, electro-mechanical oscillations appear around the main frequency. These low-frequency modes caused by groups of generators in different geographical areas are inter-area modes [215], producing inter-area oscillations with corresponding damping ratios for each of the modes. When the system is out of equilibrium, implying oscillations of the main frequency around 50 Hz, the damping ratios of the inter-area oscillations diminish those oscillations, bringing them to low amplitude [216]. There have been instances, however, where the damping ratios were not strong enough to reduce the oscillations, producing blackouts in large areas of population.

The goal of this analysis is to construct a model which relates the state of the electric grid in Spain to the damping ratio for a particular inter-area oscillation observed in the network, to both predict the damping ratio at future states of the grid and to asses which adjustments one needs to take in order to increase the damping ratio above a threshold level. To do so, Grupo AIA is working together with Red Eléctrica Española (REE)[1] through Elewit[2] to understand at a fundamental level the behaviour of the damping ratio and how to prevent it from reaching a critical level.

For a fixed inter-area oscillation mode, the damping ratio [217] is defined as the ratio between the actual damping of the oscillation and the intrinsic critical damping of the mode,

$$\zeta = \frac{\text{actual damping}}{\text{critical damping}}, \tag{9.1}$$

and is a measurable quantity in the electric grid by phasor measurement units (PMUs) [218] in the different substations. We consider a threshold of 5% for the damping ratio as potentially dangerous, below of which the factor is smaller than desired according to REE, and unstable conditions in the network can arise.

For the purpose of constructing a model, REE has provided a dataset of damping ratios together with features of the electric grid of Spain taken every 5 minutes, going from January to August of 2019. The features include local magnitudes, such as generator powers at different points of the grid; regional magnitudes, such as the solar power obtained in the different autonomous communities; and global magnitudes, such as the exchange with France and Morocco. The complete list of features has a length of over 1100 different magnitudes, over which we will build a model to predict the damping ratio.

Through the rest of the chapter, we will study the following properties regarding the damping ratio:

[1] https://www.ree.es/en
[2] https://www.elewit.ventures/

- A baseline model utilizing gradient boosting trees is constructed to establish the minimal relation between the power grid features and the damping ratio.
- Given the mentioned dataset, we determine the time-correlation of the damping ratios and features of the electric network, analyzing how this impacts potential model buildings.
- The signal's fast variations are analyzed through a new dataset with higher frequency sampling (every 20 seconds) to understand whether these variations are noise or have real physical meaning behind them, comparing the damping ratio measured at different stations of the grid.
- A first model of graph neural networks to exploit the graph structure of generation and tension of the network is proposed to analyze if local dependencies can capture additional features compared to treating the features in a tabular way.
- A probabilistic model is formalized through normalizing flows, showing an improvement over the standard approach of quantile regression, determining properly the uncertainties of the prediction.

9.2 Fingerprinting and baseline model

The baseline model of the analysis will be a gradient boosting tree regressor (an ensemble of intelligent-built decision trees) through LightGBM [219]. Although gradient boosting trees were not detailed in the introduction to machine learning in Chapter 3, they offer powerful models which are complementary to neural networks and, contrary to them, are very fast to train. As for the loss functions, both RMSE in Eq. (3.1) and MAPE in Eq. (3.2) will be considered.

When dealing with dividing the data into train, validation and test, we noticed that there might be a phenomenon of "fingerprinting". We denote fingerprinting as a model overfitting to the training data, learning by memorizing directly pairs of features with targets due to strong correlations between train, validation and test sets, since learning by memorizing maximizes its performance across the data. The model learns without forming a concept relating a richer structure between both, hence performing well for data very close to the training one but generalizing poorly. When a sample has features very close to the ones memorized in training, the prediction will be accurate because the algorithm has created a relation analogous to a database, but when the features start to be different from the ones memorized during training, the algorithm gets lost due to not having the proper target in said database. In this case, this means that data points close in time to each other have very similar characteristics, studied through a autocorrelation factor, which showed strong correlations of points with lag up to 3 (i.e., 15 minutes apart), after which the autocorrelation stabilizes slightly below 0.6. Hence, by randomized train-validation-test splitting, the performance of the model is accurate (with an absolute error smaller than a 9%) because it memorizes the samples instead of finding an underlying relation between the input features and the damping ratio. To prevent this from happening, we have separated the data in groups which have a certain lag in time t_{lag}. By grouping the data in sets of fixed duration t_{lag}, for example in groups of one hour, we generate separated groups for training and testing, while discarding every two groups in order to have the data without continuation and, hence, without autocorrelation, eliminating the phenomenon of fingerprinting. Additionally, when dividing the training into training and validation, the subgroups of fixed duration were kept together, so that there was always at least a fixed time t_{lag} between the samples in training and validation, removing potential autocorrelations when verifying the model through the validation set.

To verify that there is indeed fingerprinting, we have trained different baseline models, where the only difference is the time separation t_{lag} between the groups, these durations being: 0 minutes (no distance), 5 minutes, 10 minutes, 30 minutes, 1 hour, 4 hours, 12 hours and 1 day. The results on the train and test sets were computed for both RMSE and MAPE. It appears that the lag in time between train and test t_{lag} has to be large in order for the errors to stabilize, although it seems to start converging for the largest separation. This is expected due to the stabilization we observed of the autocorrelation of the damping ratio, hence removing the largest correlation factors, which are of the order of 0.84. For the final baseline model we take as separation 1 hour and 5 minutes (i.e., a lag of 13) to ensure eliminating sufficient autocorrelation, considering it achieves a large enough decoupling of the fingerprinting. The factor of 13 in lag was chosen in order to make groups which spread all over the day, since 13 is not a divisor of 60 (the minutes in a day) nor 24 (the hours in a day), which is achieved by having groups of 1 hour and 5 minutes.

The parameters for the LightGBM chosen for the final baseline model were found utilizing Bayesian hyperparameter optimization [72], an algorithm to fine-tune hyperparameters in an automated way. The model's performance for the values below threshold of the damping ratio, i.e., the true values $< 5\%$, is quite inaccurate. This might be due to the low number of training samples in that regime and that they are truly anomalies of the system. The RMSE score (similar to the standard deviation) for the test samples' damping ratio is 2.481, while the MAPE (the absolute percentage error) is 0.180.

With this, we have established that the time-correlations of the data samples have to be taken into account when building a model around them, suggesting a methodology to properly split train, validation and test sets. Additionally, the baseline model shows the existence of an underlying relation between the electric grid features and the damping ratio, quantifying a first approach.

9.3 Damping ratio signal analysis

The damping ratio signal provided in the previous dataset comes from a PMU in a substation of the electric grid. At first glance, the signal appears to have some sort of noise, since the points oscillate around a more stable trend with some noisy variance around it.

To verify this, REE provided three new datasets of purely the damping ratio signal from three different substations, located at well-distanced geographical cities in Spain, with an average distance of over 900 between them. These datasets have the signal taken every 20 seconds, compared to the previous one which was every 5 minutes. The upside of these datasets is that for a particular PMU, 8 months of data is available; the downside is that only 10 global features of the electric grid are measured[3] with this frequency.

When plotting the damping ratio of the three substations, the apparent noisy behaviour is actually the true signal, without noise, as all three of them show the exact same response through time of growing and decreasing, although with a slight displacement from each other. This synchronized behavior

[3] These features are not measured by PMUs, but are aggregated from other sources through timestamps.

indicates that the noisy signal is not noisy at all, but correct, so no smoothing on it can be applied. Nevertheless, working on scales of 5 minutes or even 20 seconds is a low sampling frequency when considering the true damping ratio time scale variation, making it hard to capture the exact relation between the network state and the damping ratio at that scale.

Additionally, albeit there being a small displacement between the signals in the different substations, the discrepancy is small, especially when having a small damping ratio close to the critical one, where the three signals come very close together (their difference is smaller than 1% when below threshold). This, together with the synchronized trend of the different signals, indicate that the damping ratio is a global property of the electric grid, up to a small scaling factor.

9.4 Graph neural network study

Contrary to the GNN used in Chapter 8, the GNN implementation has to change due to two reasons:

1. The objectives of the models are different: For the usage in Super FGD, the interest was to perform individual node classification, while here we want to extract a global feature of the graph, i.e., we want to model a graph regression.

2. The algorithms, although being GNNs, are completely different, as will be stated in the description of the network below, due to wanting to extract a different kind of information (a global compared to a local) than before.

Among the variables in the 5 minutes dataset provided, there is a large amount of columns corresponding to electric generation and tension in specific nodes of the Spanish electric grid. By matching the string names of those variables to a second database which outlines the electric grid structure, we are able to assign for each node a generation and a tension (assumed to be zero if the data is not given in the dataset).

The remaining variables, which do not correspond to tensions or generations, are stored in a vector and are considered global features of the graph.

With node features distributed through the graph and global features available, we perform a first study to analyze the potential of GNN to aggregate and extract richer information from the electric grid structure.

The GNN, constructed using Pytorch [212] and Pytorch Geometric [220], has then the following ordered structure of operations:

1. Node features are processed to obtain richer encodings via aggregation functions of GNNs, such as the Chebyshev spectral graph convolutional [221], the graph attentional convolutional [222], or the standard graph convolutional operators [169].

2. The encodings of all the nodes are concatenated into a flat, long vector, which is also concatenated with the global features of the graph, forming a processed feature vector.

3. The processed feature vector is then passed to a feed-forward neural network to perform a non-negative regression to predict the damping ratio of Eq. (9.1).

Testing out different combinations of encoding size, neighbour depth and aggregation functions, the model did not improve the RMSE/MAPE of the baseline obtained. Additionally, by stacking[4] [223] the GNN model together with the LightGBM one, we barely got an improvement on the overall performance, pointing out that GNN was learning closely related structures of the data when comparing it to LightGBM (i.e., their information was not complementary). This also holds when constructing a straight feed-forward model from the tabular data, yielding the same results as GNN.

We have currently three hypotheses of why this could be happening:

- The phenomenon of damping ratio we are modelling does not depend on the structure of the electric grid and the information exchange happening between the nodes explicitly, hence having a richer encoding in the nodes does not provide any additional benefit. This holds especially true when

[4] Stacking consists in building a meta-model utilizing prediction of other models as input. In this particular case, a new LightGBM model was constructed using as inputs the GNN and LightGBM predictions. This allows to account for shortcomings of one model via the other, as both models might have learned to extract different information from the data, even though their performance is similar.

considering that the most important information is already gathered in the global features, as has been verified by building a model discarding generation and tension information.

- The data between the readouts is not properly synchronized, as it is presented every 5 minutes. Some features might be aggregated as the average, or sampled at slightly different times.
- A lack of data to properly extract a more complex and rich information through the aggregation functions. The number of data samples is of the order of tens of thousands, which is low when considering training a deep learning model of the complexity we are expecting.

Further studies in the future are required to fine-tune if GNNs can really help to extract a more accurate model for the damping ratio or not, as it is natural to think that an electric grid in the form of a graph could utilize the potential of GNNs for this particular task. This concept has been applied for other tasks of electric grids, such as emulating the physics within it [224], predicting the power flow [225] or predicting power outages in a different context [226].

9.5 Uncertainty measurement through quantile regression and normalizing flows

Assessing the uncertainty of the model is crucial to understand if the RMSE obtained in Sec. 9.2 comes from a mismodelling of the damping ratio or if the relation between the features and the target has an intrinsic uncertainty. For this purpose, two methodologies are applied to quantify the uncertainty: quantile regression [227, 228] and normalizing flows (see Chapter 4).

Quantile regressions are standard ML regressions with the pinball loss function [227] as objective function. Let τ be the target quantile, y be the real value and z_τ be the quantile prediction, the pinball loss function takes the form:

$$L_\tau(y,z) = \begin{cases} (y - z_\tau)\,\tau, & \text{if } y \geq z_\tau, \\ (z_\tau - y)(1-\tau), & \text{if } y < z_\tau. \end{cases}$$

This loss function can be applied to any ML algorithm, allowing it to predict a certain quantile τ instead of the target value. In particular, we have used to construct a quantile regressor via LightGBM for the quantiles $\{0.025, 0.16, 0.50, 0.84, .975\}$, to have the uncertainty at one and two standard deviations (if the damping ratio follows a Gaussian-like distribution).

Another method utilized to obtain the quantiles is by performing a density estimation via normalizing flows of the conditional density

$$P(\text{damping ratio}|\text{electric grid features}).$$

Given the electric grid features, the density of the damping ratio can be estimated, allowing to compute the desired quantiles, among other quantities.

The quantile models were constructed over the 20 second dataset, as the number of data available is larger. Additionally, when constructing the baseline model for the 5 minutes dataset but using only the 10 global features observed in the 20 second dataset, the performance dropped only slightly (from a RMSE of 2.481 to 2.563), justifying that most of the relevant information is contained in these 10 variables at this point in time. To verify the integrity of the quantiles obtained by both methods, one can simply choose to compute the ratio of test values below the quantile values, which should be as close to the quantile chosen as possible, i.e., a τ fraction of the data should be below z_τ. For the quantile regressions of LightGBM, the mean absolute quantile error is of 2.27%, while the normalizing flow obtains a mean absolute quantile error of 0.60%, showing the expressiveness of capturing properly the uncertainty of the problem. With this, we can define accurately enough the quantiles of the uncertainty of the model, which have an average of $\sigma \equiv (z_{0.84} - z_{0.16})/2$ of 1.79.

Additionally, thanks to predicting the distribution of the damping ratio instead of a point value, normalizing flows allow to answer the question of how probable it is for the damping ratio to be below 5% given an electric network configuration, enabling to detect crucial moments in the power grid.

With all this, normalizing flows show an accurate way of quantifying the uncertainty contained in modeling the damping ratio as a magnitude of the rest of the electric grid features.

9.6 Conclusions

The damping ratio is a complex phenomenon which can make a strong impact on the electric grid, producing blackouts. Modeling this quantity through features of the system presents a challenging problem due to the stochastic variations the damping ratio has intrinsically with these features.

A baseline model was constructed, showing the potential that relates the features of the grid to the damping ratio. This model gives a quantitative estimation of how well machine learning can perform when dealing with the prediction of the damping ratio. Additionally, to build a robust model able to accurately capture the mechanics which relate the features to the damping ratio, we have shown that the data has to be split properly to remove temporal connections between train, validation and test sets which otherwise lead to fingerprinting of the learned model.

When studying the damping ratio at different stations of the grid for a higher frequency dataset, it shows that smoothing the signal is incorrect, since we might be eliminating true physics contained in the data and will be learning an inaccurate model, emphasizing that the fast variations observed are true signals.

We have shown that the data can be structured into a graph following the layout of the electric grid in order to extract potentially new information using rich models, such as graph neural networks. So far, no additional information has been obtained using GNNs, possibly explained by one of the three reasons described previously.

Finally, the stochastic nature of the damping ratio has been quantified properly both through quantile regressions and specially through normalizing flows, allowing for the measurement of the probability of being in certain ranges and below the threshold value.

Future approaches to study more in-depth the impact of the potential benefits a graph neural network could add to the model will be performed with more data. Furthermore, mechanisms of altering the features of the electric grid to minimize the risk of the damping ratio to be below threshold will be proposed. The established baseline, GNN study and uncertainty assessment open up a future collaboration between Grupo AIA and REE.

APPENDIX

A. APPENDIX: LIKELIHOOD-FREE INFERENCE OF EXPERIMENTAL NEUTRINO OSCILLATIONS USING NEURAL SPLINE FLOWS

In this Appendix, we will describe how observed events $(p_\mu, \theta_\mu, E_\nu)$ are generated for the different experiments. Because of statistical fluctuations due to an event oscillation with certain probability (following Eq. (6.1)), the exact number of observed events cannot be determined beforehand, but can approximately be chosen. In order to generate the observations of a fixed set of parameters $(\theta_{\text{mix}}, \Delta m^2)$, the procedure is the following:

1. Choose an approximated number of observed events N_{approx}.
2. Generate a MC event $(p_\mu, \theta_\mu, E_\nu)$, make the event oscillate according to its energy E_ν and accept it with probability given by Eq. (6.1), until accepting a total of N_{approx} events. This takes a stochastic number of N_{init} tries, which corresponds to the number of initial events before oscillating.
3. Generate new N_{init} number of MC events, and accept them with probability according to Eq. (6.1). The accepted samples form the observation set, with a number of N_{obs} samples, which is similar to N_{approx}.

To summarize, given a fixed set of parameters $(\theta_{\text{mix}}, \Delta m^2)$, we determine an approximate initial number N_{init} of events before oscillating needed to generate N_{approx} oscillated observations. However, because this is a stochastic procedure and we stopped exactly when N_{approx} were generated, the procedure has to be repeated with a fixed number of N_{init} tries, giving us $N_{\text{obs}} \sim N_{\text{approx}}$ observed events.

Two kind of experiments were performed for five different set of parameters: one of low statistics (around 500 observations), with a number of the order of the real number of observed events in T2K, and one of high statistics (around 50k observations), to see how the algorithm behaves and compares to traditional methods when having a larger amount of data available.

The parameters of the initial experiments 1 and 2 in Sec. 6.3.2 are chosen following the best fit value of $\sin^2\theta_{23}$ and Δm^2 from [229]. The parameter $\theta_{\rm mix}$ is computed as

$$\theta_{\rm mix} = \frac{\pi}{2} - \arcsin\sqrt{\sin^2\theta_{23}},$$

where we have used the fact that Eq. 6.1 is symmetric over $\theta_{\rm mix} = \pi/4$ and fixed it in the interval $[0, \pi/4]$. For the best fit value, maximal mixing is almost achieved ($\sin 2\theta_{\rm mix} = 0.9986$). Experiments 3 and 4 (5 and 6) follow the same procedure, but choosing $\sin^2\theta_{23}$ within 1 (2) σ of the best fit. Experiments 7-10 the mixing angle has the value of experiments 3 and 4, but Δm^2 is changed in order to explore the behaviour on the parameter space.

B. APPENDIX: EXHAUSTIVE NEURAL IMPORTANCE SAMPLING APPLIED TO MONTE CARLO EVENT GENERATION

We can see in Tab. 7.1 that although the number of rejection sampling cycles is remarkably lower for the NSF proposal, the time for obtaining one million samples is fairly similar. This is because the time for a cycle is the sum of the time in generating and evaluating the proposal plus the time it takes to evaluate $p(\mathbf{x})$, in this case the CCQE cross section. On average, for one rejection cycle of this experiment, generating+sampling for the NSF a batch of 300 000 samples takes 0.31 seconds, for a uniform proposal it only takes 0.00079 seconds, and evaluating 300 000 samples for the CCQE cross section model takes 0.0021 seconds. When summing up the time for the selected proposal plus the time of the model, we can see that the NSF takes a large fraction of the computational time (more than hundred times more than evaluating the cross section model), so overall a cycle for the NSF proposal takes over a hundred cycles of uniform proposal. This is because the CCQE cross section model we have chosen for this study (Sec. 7.1.1) is relatively simple, especially compared to other applications in HEP. Additionally we are still relatively low in the number of dimensions. As a thought experiment, we considered different cases for a similar set up: sampling ten million samples using batches of 300 000 in each cycle, where the time of evaluating the model $p(\mathbf{x})$ is increased by a factor of t_{increase}.

Tab. B.1 shows the results for obtaining one million samples under these circumstances for the quantiles $k_{\text{Q}1.0}$, $k_{\text{Q}0.9999}$ and $k_{\text{Q}0.9990}$ with the same acceptance probability as in Tab. 7.1, consider different hypothetical models with factors $t_{\text{increase}} \in \{1, 10, 100, 1000\}$ compared to the CCQE cross section. We show the time (in seconds) it takes to generate these ten million samples for

Tab. B.1: Model complexity time comparison table for both NSF and uniform proposals to generate ten million samples via rejection sampling. Choosing k_Q as one of the below quantiles, we increase the time to evaluate the model $p(\mathbf{x})$ for each rejection cycle by a factor t_{increase}. t_{NSF} (t_{Unif}) is the time it takes to obtain ten million accepted samples, in seconds, for the NSF (uniform) proposal. $t_{\text{Unif}}/t_{\text{NSF}}$ is the ratio of the time between both proposals, showing that for more complex models than the CCQE cross section of this paper, NSF outperforms a uniform proposal by orders of magnitude.

Quantile	t_{increase}	t_{NSF}	t_{Unif}	$t_{\text{Unif}}/t_{\text{NSF}}$
1.0000	1	2017.26	471.64	0.23
	10	2141.10	3567.01	1.67
	100	3379.54	34520.76	10.21
	1000	15763.94	344058.17	21.83
0.9999	1	63.95	123.19	1.93
	10	67.87	928.73	13.68
	100	107.01	8984.12	83.96
	1000	498.39	89538.01	179.65
0.9990	1	28.87	35.36	1.22
	10	30.64	266.54	8.70
	100	48.31	2578.40	53.37
	1000	225.00	25696.93	114.21

the NSF, t_{NSF}, and for the uniform proposal, t_{Unif}, while also computing their ratio $t_{\text{Unif}}/t_{\text{NSF}}$. For models of $p(\mathbf{x})$ with larger computationally complexity, we can see that even for the maximum quantile $k_{\text{Q1.0}}$ NSF is faster to obtain ten million accepted samples than using a uniform proposal, gaining a whole order of magnitude in time when the model is at least hundred times more complex. Choosing quantiles smaller than 1.0, for models which take a thousand times longer to compute compared to the CCQE cross section one, the NSF proposal generates these ten million samples over a hundred times faster than the uniform distribution. Even for simpler models with only hundred more computation complexity, the gain of using the NSF proposal is noticeable.

Regarding the number of parameters used by the NSF with increasing number of dimensions, when maintaining the same architecture for the NSF, the number scales linearly with the dimension of the data. However, when considering higher dimensions, to enrich further the flexibility of the family of densities produced by the NSF to account for more complex correlations and dependencies between each dimension, the architecture might have to change, adding an increase larger than linear for the number of parameters.

C. APPENDIX: GRAPH NEURAL NETWORK FOR 3D CLASSIFICATION OF AMBIGUITIES AND OPTICAL CROSSTALK IN SCINTILLATOR-BASED NEUTRINO DETECTORS

C.1 Input variables

The list of variables used as features for the graph nodes is given below. Each node is placed at XYZ coordinates matching the center of a cube, however, these center coordinates are not node variables by themselves since the detector response is isotropic. The numbers in front of each variable match those in Fig C.1.

- `0-2: peXY, peXZ, peYZ`
  Number of photons detected in the XY, XZ or YZ-fiber intersecting the cube under consideration corrected by the expected attenuation.

- `3-5: mXY, mXZ, mYZ`
  Number of active voxels intersected by the fiber associated to `peXY, peXZ` or `peYZ`

- `6: pewav`
  Average number of detected photons `peXY, peXZ, peYZ`, each weighted by the fiber multiplicity `mXY, mXZ, mYZ`.

$$\texttt{pewav} = \frac{\frac{\texttt{peXY}}{\texttt{mXY}} + \frac{\texttt{peXZ}}{\texttt{mXZ}} + \frac{\texttt{peYZ}}{\texttt{mYZ}}}{3}$$

- `7-9: pullX, pullY, pullZ`

Relative difference between the light measured in two different 2D planes.

$$\texttt{pullX} = \frac{\texttt{peXY} - \texttt{peXZ}}{\texttt{peXY} + \texttt{peXZ}}$$

$$\texttt{pullY} = \frac{\texttt{peXY} - \texttt{peYZ}}{\texttt{peXY} + \texttt{peYZ}}$$

$$\texttt{pullZ} = \frac{\texttt{peXZ} - \texttt{peYZ}}{\texttt{peXZ} + \texttt{peYZ}}$$

- `10: residual`
  Similarity of the light yield measured in the three 2D planes, measured as the squared distance from each `peXY, peXZ, peYZ` to the average, weighted by the squared average.

$$\mu = \frac{\texttt{peXY} + \texttt{peXZ} + \texttt{peYZ}}{3}$$

$$\texttt{residual} = \frac{(\texttt{peXY} - \mu)^2 + (\texttt{peXZ} - \mu)^2 + (\texttt{peYZ} - \mu)^2}{\mu^2}$$

- `11: pullXYZ`
  Similarity of the light yield measured in the three 2D planes, measured as a combination of 2D pulls (a_1,a_2,a_3) weighted by `pewav`.

$$a_1 = \frac{\frac{\texttt{peXY}}{\texttt{mXY}} - \frac{\texttt{peXZ}}{\texttt{mXZ}}}{\frac{\texttt{peXY}}{\texttt{mXY}} + \frac{\texttt{peXZ}}{\texttt{mXZ}}}$$

$$a_2 = \frac{\frac{\texttt{peXY}}{\texttt{mXY}} - \frac{\texttt{peYZ}}{\texttt{mYZ}}}{\frac{\texttt{peXY}}{\texttt{mXY}} + \frac{\texttt{peYZ}}{\texttt{mYZ}}}$$

$$a_3 = \frac{\frac{\texttt{peXZ}}{\texttt{mXZ}} - \frac{\texttt{peYZ}}{\texttt{mYZ}}}{\frac{\texttt{peXZ}}{\texttt{mXZ}} + \frac{\texttt{peYZ}}{\texttt{mYZ}}}$$

$$\texttt{pullXYZ} = \frac{a_1 a_2 + a_1 a_3 + a_2 a_3}{\texttt{pewav}}$$

- `12: ratioMQ`
  Ratio between the average voxel multiplicity in the three fibers and `pewav`.

$$\texttt{ratioMQ} = \frac{\frac{\texttt{mXY}+\texttt{mXZ}+\texttt{mYZ}}{3}}{\texttt{pewav}}$$

- `13-14: R1, R3`
  Number of active neighbor voxels in a sphere of certain radius.
  $\hookrightarrow$ `R1`, r=1 cm.
  $\hookrightarrow$ `R2`, r=2 cm.
  $\hookrightarrow$ `R3`, r=5 cm. `R2` was not used as a variable due to the high correlation with `R1`, but is used to compute `RR`.

- `15-20: x+, x-, y+, y-, z+, z-`
  Boolean variables representing the existence of immediate neighbors in each of the 6 surrounding cubes

- `21: orthogonal_neighbor`
  It is 1 if any of `x+, x-, y+, y-, z+, z-` is 1.

- `22: RR`
  Ratio between the number of close and far voxels. The $\epsilon = 10^{-7}$ prevents numerical problems when `R3`=0.

$$\texttt{RR} = \frac{\texttt{R2}}{\texttt{R3} + \epsilon}$$

- `23: ratioDQ`
  Ratio between the average voxel distance `aveDist` around the voxel and the weighted average light yield `pewav`.

$$\texttt{ratioDQ} = \frac{\texttt{aveDist}}{\texttt{pewav}}$$

- `24: aveDist`
  Average distance from the voxel center C to all fired voxel centers (C_i)

within a sphere of radius 2.5 cm.

$$\texttt{aveDist} = \frac{1}{N}\sum_{i}^{N} \text{EuclidianDist}(C, C_i)$$

A number of these variables are calculated from the same underlying properties of the energy deposits. In theory, an infinitely deep GNN trained on an infinite amount of training data would be able to extract all of the information required for classification from the few underlying properties. In practice, we use a larger number of derived variables to guide the GNN to allow it to more easily extract information from the data and to converge quickly in the training process. Global position was intentionally not used as a variable to avoid the GNN to learn neutrino modelling specific behaviours.

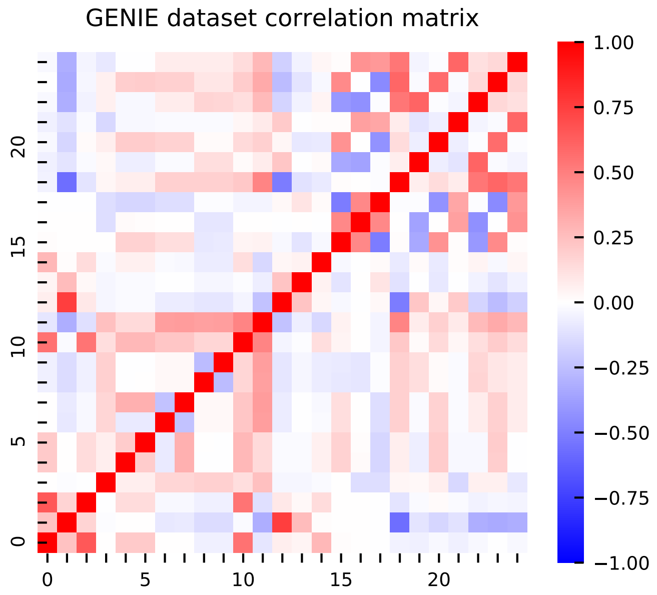
C.2 Comparison of GENIE and P-Bomb simulated data samples

Figure C.1 shows the correlations of the input variables defined in Appendix C.1 for the GENIE and P-Bomb data samples. Differences between the two matrices arise from the different topologies of interactions produced by the two generator methods.

C.3 Event Gallery

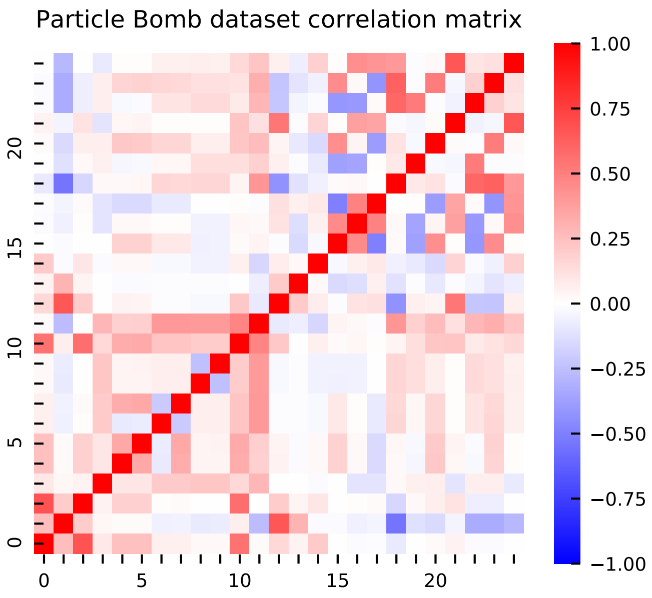
This section contains a number of visualizations to show the classification performance of the GNN for a number of neutrino interactions with different complexity and topology. Displays are shown for different events in Figs C.2 - C.7: all voxels with their true classification, only the true track voxels, the classified track voxels using the charge cut method, and the classified track voxels using the GNN. The interactions shown here are examples of interactions containing many ghost voxels in order to showcase the GNN performance.

The track voxel classification ability of the charge cut and GNN methods can be seen by comparing subfigures (c) and (d) with (b), respectively, for each interaction. The GNN is able to reject ghost voxels very well, as shown in Figs C.3, C.5, C.6 and C.7 where ghost tracks remain using the charge cut method. In general, the performance improvement from the GNN increases with the complexity of the interactions. For simple interactions with only a single muon in the final state both methods perform similarly.



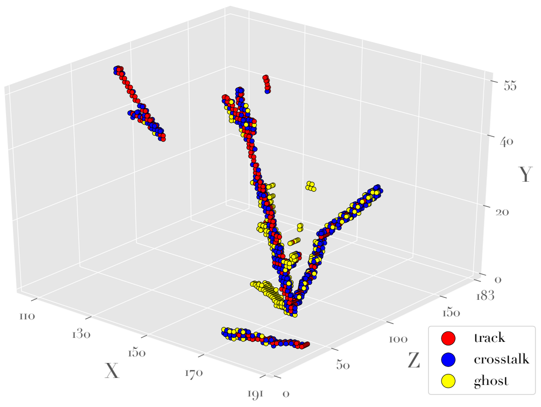
(a) GENIE dataset correlation matrix.

Fig. C.1: Correlation matrices for the input variables of the GENIE and P-Bomb datasets used. Appendix C.1 gives the mapping between the numbers on the axes and the variable names.

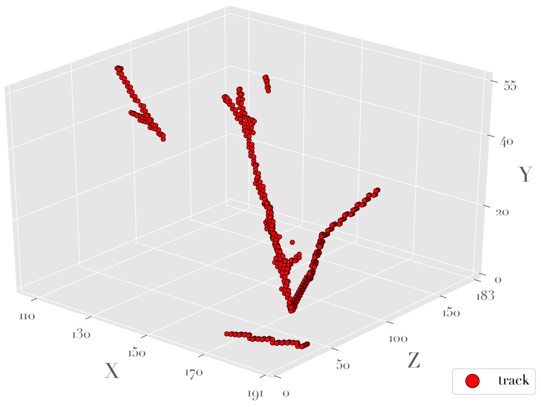


(b) P-Bomb dataset correlation matrix.

Fig. C.1: Correlation matrices for the input variables of the GENIE and P-Bomb datasets used (cont.).

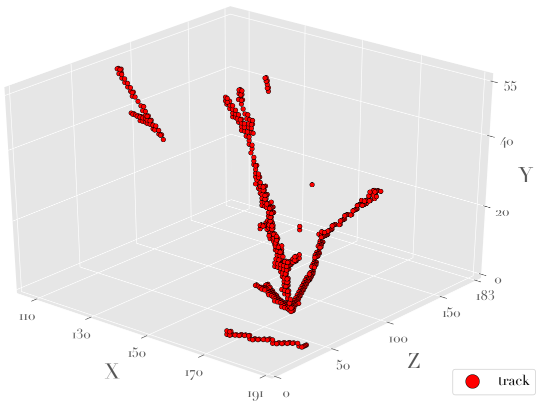


(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

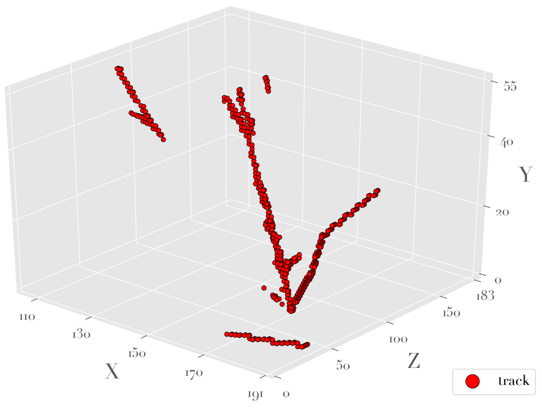


(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

Fig. C.2: 3D visualization of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. The GNN cut is able to almost entirely reject the fake track traveling on the XZ plane and stopping near to the vertex at X∼160 mm and Z∼70 mm, while the charge cut cannot.

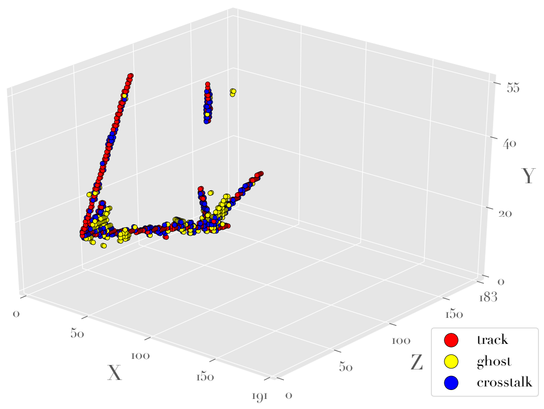


(c) The 3D voxels labelled as track according to the charge cut classification are shown.

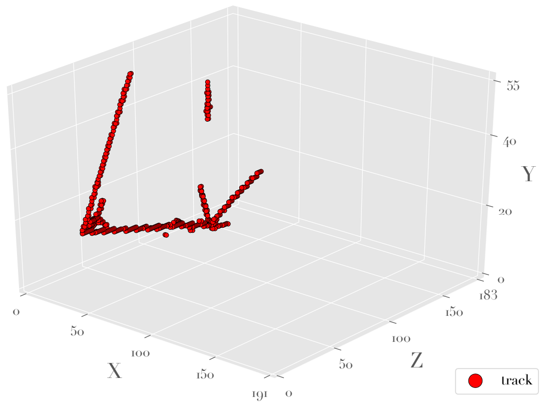


(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.2: Continuation.

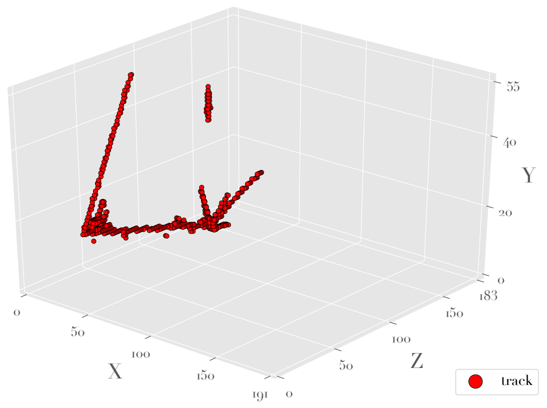


(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

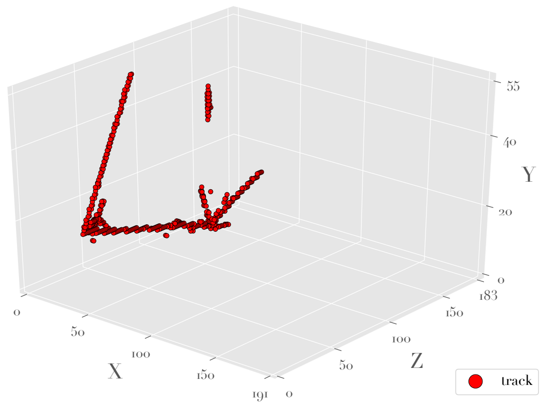


(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

Fig. C.3: 3D visualization of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. The charge cut is not able to reject two fake tracks, one coming from a vertex a X¡50 mm Z¡50 mm traveling on the XZ plane and stopping near to the vertex at X$\sim$160 and Z$\sim$70. Moreover, the charge cut leaves a bump of ghost voxels around the vertex that could mimic the interaction of a few low-energy protons, an effect that could bias the reconstruction of the neutrino energy.

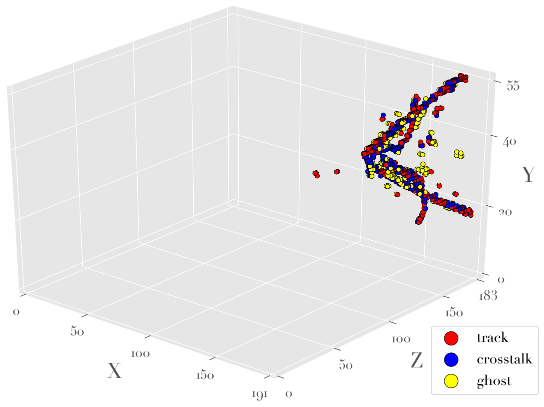


(c) The 3D voxels labelled as track according to the charge cut classification are shown.

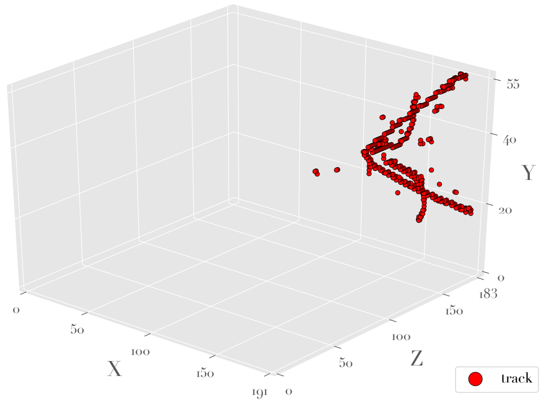


(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.3: Continuation.

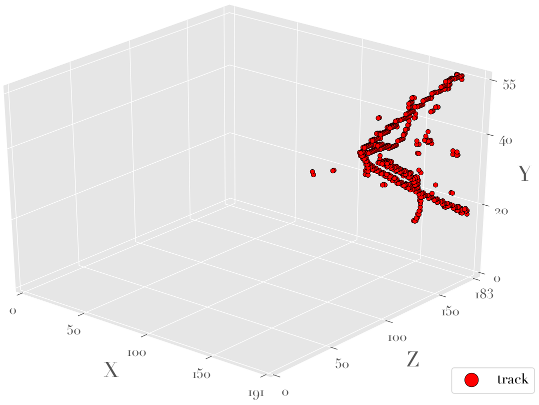


(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

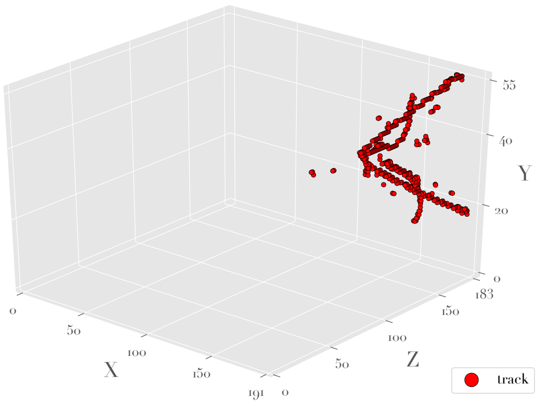


(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

Fig. C.4: 3D visualization of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. In this event the performance of GNN and the charge cut is quite similar because the ghost voxels are mainly given by the overlap of crosstalk hits in the 2D readout views.

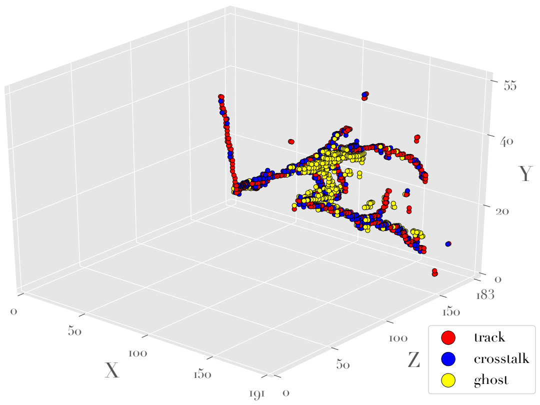


(c) The 3D voxels labelled as track according to the charge cut classification are shown.

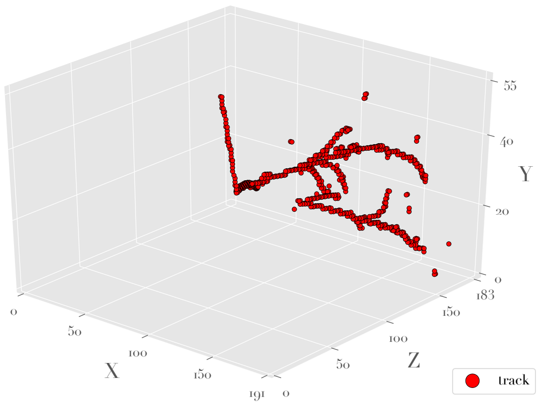


(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.4: Continuation.

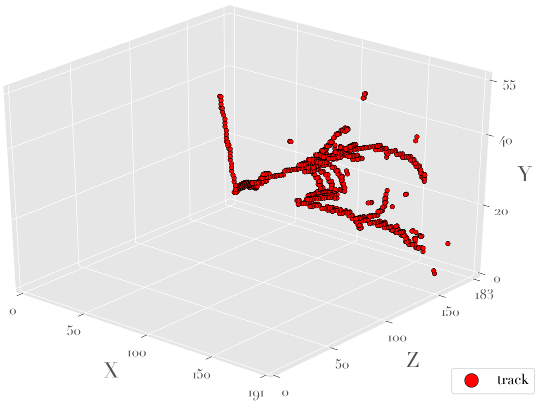


(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

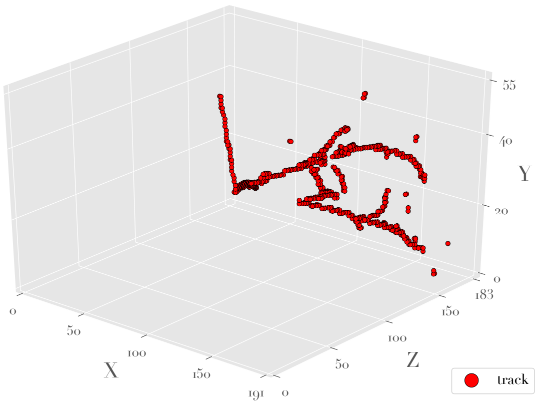


(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

Fig. C.5: 3D visualization of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. This neutrino event has a quite high multiplicity and tracks are quite close to each other. This produces relatively big clusters of ghost voxels that produce at least two fake tracks even after the charge cut. Instead GNN allows us to classify ghosts more precisely and correctly visualize the correct number of tracks. Moreover, the charge cut makes true tracks more fat making their separation harder and, potentially, less precise the particle momentum reconstruction.

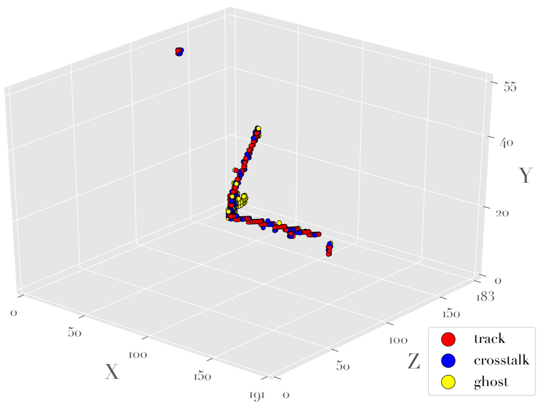


(c) The 3D voxels labelled as track according to the charge cut classification are shown.

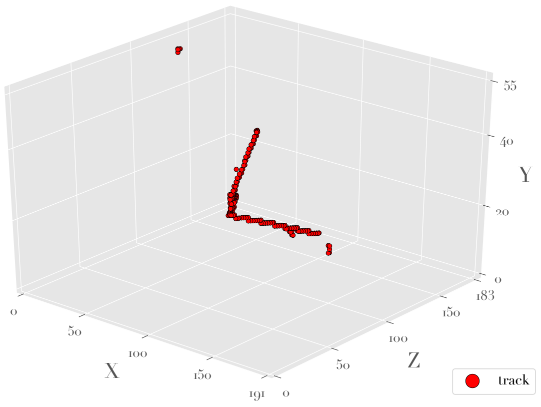


(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.5: Continuation.

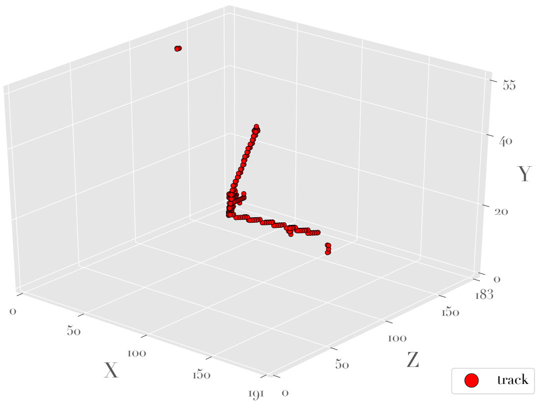


(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

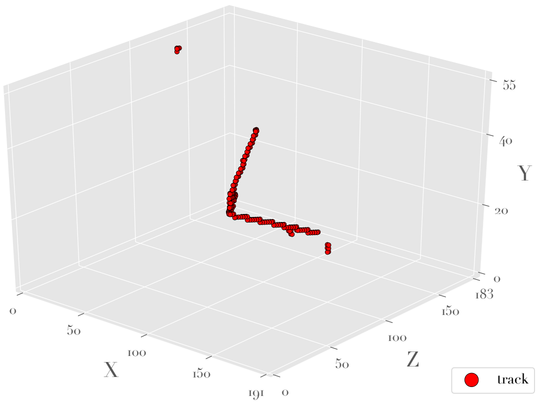


(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

Fig. C.6: 3D visualization of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. Although this is a relatively simple neutrino event, the charge cut is not able to reject a fake track stopping near the neutrino interaction vertex while GNN can provide a much cleaner reconstruction.

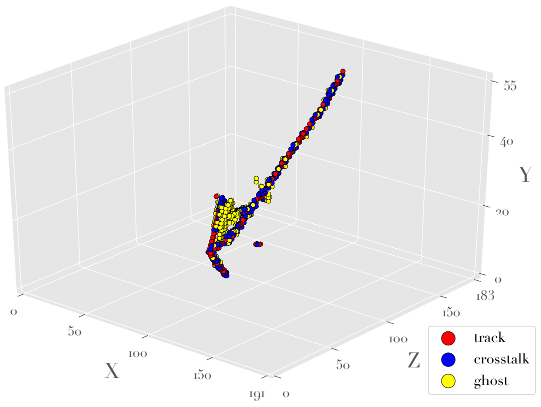


(c) The 3D voxels labelled as track according to the charge cut classification are shown.

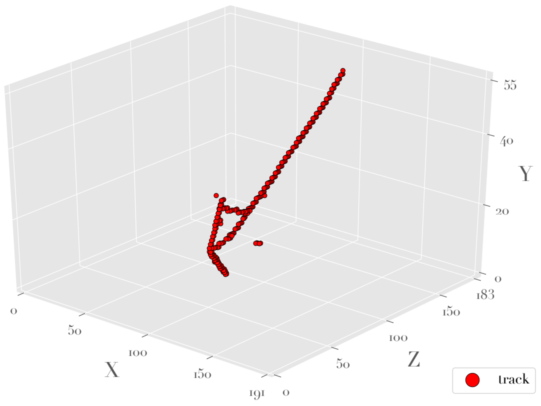


(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.6: Continuation.

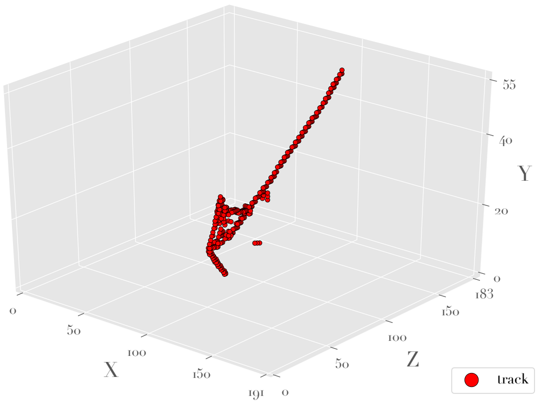


(a) The 3D voxels labelled as track (red), crosstalk (blue) and ghost (yellow) according to the truth information from the simulation are shown.

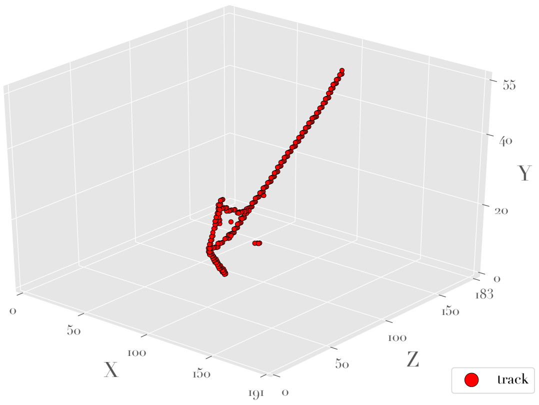


(b) Only the 3D voxels labelled as track according to the truth information from the simulation are shown.

Fig. C.7: 3D visualization of a neutrino interaction in a finely segmented 3D scintillator detector after the 3D matching of the three 2D views. In the neutrino event GNN can easily reject the relatively big cluster of ghost voxels that would make difficult a proper reconstruction of the number of tracks and corresponding energy, in particular near to the interaction vertex.



(c) The 3D voxels labelled as track according to the charge cut classification are shown.



(d) The 3D voxels labelled as track according to the GNN classification are shown.

Fig. C.7: Continuation.

BIBLIOGRAPHY

[1] K. Abe *et al.*, "The t2k experiment," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 659, no. 1, pp. 106 – 135, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900211011910

[2] H.-K. Proto-Collaboration, K. Abe *et al.*, "Physics potential of a long-baseline neutrino oscillation experiment using a J-PARC neutrino beam and Hyper-Kamiokande," *Progress of Theoretical and Experimental Physics*, vol. 2015, no. 5, 05 2015, 053C02. [Online]. Available: https://doi.org/10.1093/ptep/ptv061

[3] B. Abi *et al.*, "Deep Underground Neutrino Experiment (DUNE), Far Detector Technical Design Report, Volume II DUNE Physics," 2020.

[4] L. Evans and P. Bryant, "LHC machine," *Journal of Instrumentation*, vol. 3, no. 08, pp. S08 001–S08 001, aug 2008. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F3%2F08%2Fs08001

[5] L. Rossi and O. Brüning, "Introduction to the HL-LHC project," in *The High Luminosity Large Hadron Collider.* WORLD SCIENTIFIC, Sep. 2015, pp. 1–17. [Online]. Available: https://doi.org/10.1142/9789814675475_0001

[6] R. Oerter, *The theory of almost everything : the Standard Model, the unsung triumph of modern physics.* New York: Pi Press, 2006.

[7] Y. V. Kozlov *et al.*, "Neutrino mass problem: the state of the art," *Physics-Uspekhi*, vol. 40, no. 8, pp. 807–842, aug 1997. [Online]. Available: https://doi.org/10.1070%2Fpu1997v040n08abeh000273

[8] T. Mannel, “Theory and phenomenology of cp violation,” *Nuclear Physics B - Proceedings Supplements*, vol. 167, pp. 170 – 174, 2007, proceedings of the 7th International Conference on Hyperons, Charm and Beauty Hadrons. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920563206010711

[9] G. R. Farrar and M. E. Shaposhnikov, “Baryon asymmetry of the universe in the minimal standard model,” *Phys. Rev. Lett.*, vol. 70, pp. 2833–2836, May 1993. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.70.2833

[10] V. Trimble, “Existence and nature of dark matter in the universe,” *Annual Review of Astronomy and Astrophysics*, vol. 25, no. 1, pp. 425–472, 1987. [Online]. Available: https://doi.org/10.1146/annurev.aa.25.090187.002233

[11] P. J. E. Peebles and B. Ratra, “The cosmological constant and dark energy,” *Rev. Mod. Phys.*, vol. 75, pp. 559–606, Apr 2003. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.75.559

[12] K. Albertsson *et al.*, “Machine learning in high energy physics community white paper,” *Journal of Physics: Conference Series*, vol. 1085, p. 022008, sep 2018. [Online]. Available: https://doi.org/10.1088%2F1742-6596%2F1085%2F2%2F022008

[13] J. Nieves, F. Sánchez, I. R. Simo, and M. J. V. Vacas, “Neutrino energy reconstruction and the shape of the charged current quasielastic-like total cross section,” *Phys. Rev. D*, vol. 85, p. 113008, Jun 2012. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.85.113008

[14] R. Gran, J. Nieves, F. Sanchez, and M. J. V. Vacas, “Neutrino-nucleus quasi-elastic and 2p2h interactions up to 10 gev,” *Phys. Rev. D*, vol. 88, p. 113007, Dec 2013. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.88.113007

[15] A. Blondel *et al.*, “The SuperFGD Prototype Charged Particle Beam Tests,” 2020.

[16] V. Gaitan, *Neural Networks in High Energy Physics: From Pattern Recognition to Exploratory Data Analysis*, ser. Tesis doctorals de la Universitat Autònoma de Barcelona. Publicacions de la Universitat Autònoma de Barcelona, 1994. [Online]. Available: https://books.google.es/books?id=SWs5xgEACAAJ

[17] S. Pina-Otey, F. Sánchez, V. Gaitan, and T. Lux, "Likelihood-free inference of experimental neutrino oscillations using neural spline flows," *Phys. Rev. D*, vol. 101, p. 113001, Jun 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.101.113001

[18] S. Pina-Otey, F. Sánchez, T. Lux, and V. Gaitan, "Exhaustive neural importance sampling applied to monte carlo event generation," *Phys. Rev. D*, vol. 102, p. 013003, Jul 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.102.013003

[19] S. Alonso-Monsalve, D. Douqa, C. Jesús-Valls, T. Lux, S. Pina-Otey, F. Sánchez, D. Sgalaberna, and L. H. Whitehead, "Graph neural network for 3d classification of ambiguities and optical crosstalk in scintillator-based neutrino detectors," 2020.

[20] J. Chadwick, "The Existence of a Neutron," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering .*, vol. 136, pp. 692–708, 1932.

[21] ——, "The Neutron," *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering .*, vol. 142, pp. 1–25, 1933.

[22] E. Fermi, "Versuch einer theorie der β-strahlen. i," *Zeitschrift für Physik*, vol. 88, no. 3-4, pp. 161–177, 1934.

[23] F. Reines and C. L. Cowan, "Detection of the free neutrino," *Phys. Rev.*, vol. 92, pp. 830–831, Nov 1953. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRev.92.830

[24] C. L. Cowan, F. Reines, F. B. Harrison, H. W. Kruse, and A. D. McGuire, "Detection of the free neutrino: a confirmation," *Science*, vol. 124, no. 3212, pp. 103–104, 1956. [Online]. Available: https://science.sciencemag.org/content/124/3212/103

[25] B. Pontecorvo, “Electron and Muon Neutrinos,” *Sov. Phys. JETP*, vol. 10, pp. 1236–1240, 1960.

[26] G. Danby, J.-M. Gaillard, K. Goulianos, L. M. Lederman, N. Mistry, M. Schwartz, and J. Steinberger, “Observation of high-energy neutrino reactions and the existence of two kinds of neutrinos,” *Phys. Rev. Lett.*, vol. 9, pp. 36–44, Jul 1962. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.9.36

[27] K. Kodama *et al.*, “Observation of tau neutrino interactions,” *Phys. Lett. B*, vol. 504, pp. 218–224, 2001.

[28] Wikipedia contributors, “Standard model,” 2020, [Online; accessed 12-June-2020]. [Online]. Available: https://en.wikipedia.org/wiki/Standard_Model

[29] B. Pontecorvo, “Mesonium and anti-mesonium,” *Sov. Phys. JETP*, vol. 6, p. 429, 1957.

[30] Z. Maki, M. Nakagawa, and S. Sakata, “Remarks on the unified model of elementary particles,” *Prog. Theor. Phys.*, vol. 28, pp. 870–880, 1962, [,34(1962)].

[31] B. Pontecorvo, “Neutrino Experiments and the Problem of Conservation of Leptonic Charge,” *Sov. Phys. JETP*, vol. 26, pp. 984–988, 1968.

[32] R. Davis, D. S. Harmer, and K. C. Hoffman, “Search for neutrinos from the sun,” *Phys. Rev. Lett.*, vol. 20, pp. 1205–1209, May 1968. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.20.1205

[33] Y. Fukuda *et al.*, “Evidence for oscillation of atmospheric neutrinos,” *Phys. Rev. Lett.*, vol. 81, pp. 1562–1567, Aug 1998. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.81.1562

[34] Q. R. Ahmad *et al.*, “Measurement of the rate of $\nu_e + d \to p + p + e^-$ interactions produced by ^8b solar neutrinos at the sudbury neutrino observatory,” *Phys. Rev. Lett.*, vol. 87, p. 071301, Jul 2001. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.87.071301

[35] ——, "Direct evidence for neutrino flavor transformation from neutral-current interactions in the sudbury neutrino observatory," *Phys. Rev. Lett.*, vol. 89, p. 011301, Jun 2002. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.89.011301

[36] S. M. Bilenky and B. Pontecorvo, "The Quark-Lepton Analogy and the Muonic Charge," *Yad. Fiz.*, vol. 24, pp. 603–608, 1976.

[37] ——, "Again on Neutrino Oscillations," *Lett. Nuovo Cim.*, vol. 17, p. 569, 1976.

[38] S. Bilenky and B. Pontecorvo, "Lepton mixing and neutrino oscillations," *Physics Reports*, vol. 41, no. 4, pp. 225 – 261, 1978. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0370157378900959

[39] *The International System of Units.* Bureau International des Poids et Mesures, 2006.

[40] R. Davis, "A review of the homestake solar neutrino experiment," *Progress in Particle and Nuclear Physics*, vol. 32, pp. 13 – 32, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0146641094900043

[41] W. Hampel *et al.*, "Gallex solar neutrino observations: results for gallex iv," *Physics Letters B*, vol. 447, no. 1, pp. 127 – 133, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0370269398015792

[42] T. J. Haines *et al.*, "Calculation of atmospheric neutrino-induced backgrounds in a nucleon-decay search," *Phys. Rev. Lett.*, vol. 57, pp. 1986–1989, Oct 1986. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.57.1986

[43] M. Apollonio *et al.*, "Search for neutrino oscillations on a long base-line at the CHOOZ nuclear power station," *The European Physical Journal C*, vol. 27, no. 3, pp. 331–374, Apr. 2003. [Online]. Available: https://doi.org/10.1140/epjc/s2002-01127-9

[44] K. Eguchi *et al.*, "First results from kamland: Evidence for reactor antineutrino disappearance," *Phys. Rev. Lett.*, vol. 90, p. 021802, Jan 2003. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.90.021802

[45] M. A. Acero, P. Adamson, L. Aliaga, T. Alion, V. Allakhverdian, S. Altakarli, N. Anfimov, A. Antoshkin, A. Aurisano, A. Back, C. Backhouse, M. Baird, N. Balashov, P. Baldi, B. A. Bambah *et al.*, "First measurement of neutrino oscillation parameters using neutrinos and antineutrinos by nova," *Phys. Rev. Lett.*, vol. 123, p. 151803, Oct 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.123.151803

[46] K. Abe *et al.*, "Observation of electron neutrino appearance in a muon neutrino beam," *Phys. Rev. Lett.*, vol. 112, p. 061802, Feb 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.112.061802

[47] S. Fukuda *et al.*, "The super-kamiokande detector," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 501, no. 2, pp. 418 – 462, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016890020300425X

[48] Y. Yamazaki *et al.*, "Accelerator technical design report for J-PARC," 3 2003.

[49] K. Abe *et al.*, "Measurements of the t2k neutrino beam properties using the ingrid on-axis near detector," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 694, pp. 211 – 223, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900212002987

[50] S. Assylbekov *et al.*, "The t2k nd280 off-axis pi–zero detector," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 686, pp. 48 – 63, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900212005153

[51] N. Abgrall *et al.*, "Time projection chambers for the t2k near detectors," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 637, no. 1, pp. 25 – 46, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900211003421

[52] P.-A. Amaudruz *et al.*, "The t2k fine-grained detectors," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 696, pp. 1 – 31, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168900212008789

[53] K. Abe *et al.*, "T2k nd280 upgrade - technical design report," 2019.

[54] ——, "Measurement of neutrino and antineutrino oscillations by the t2k experiment including a new additional sample of ν_e interactions at the far detector," *Phys. Rev. D*, vol. 96, p. 092006, Nov 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.96.092006

[55] ——, "Constraint on the matter–antimatter symmetry-violating phase in neutrino oscillations," *Nature*, vol. 580, no. 7803, pp. 339–344, Apr. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2177-0

[56] D. R. Raban and A. Gordon, "The evolution of data science and big data research: A bibliometric analysis," *Scientometrics*, vol. 122, no. 3, pp. 1563–1581, Jan. 2020. [Online]. Available: https://doi.org/10.1007/s11192-020-03371-2

[57] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "Machine learning: A historical and methodological analysis," vol. 4, p. 69, Sep. 1983. [Online]. Available: https://www.aaai.org/ojs/index.php/aimagazine/article/view/406

[58] H. Wang and B. Raj, "On the origin of deep learning," 2017.

[59] M. Madiajagan and S. S. Raj, "Chapter 1 - parallel computing, graphics processing unit (gpu) and new hardware for deep learning in computational intelligence research," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, A. K. Sangaiah,

Ed. Academic Press, 2019, pp. 1 – 15. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128167182000087

[60] M. Hilbert and P. Lopez, “The world’s technological capacity to store, communicate, and compute information,” *Science*, vol. 332, no. 6025, pp. 60–65, Feb. 2011. [Online]. Available: https://doi.org/10.1126/science.1200970

[61] D. O. Hebb, *The organization of behavior : a neuropsychological theory.* Mahwah, N.J: L. Erlbaum Associates, 2002.

[62] D. Crevier, *AI : the tumultuous history of the search for artificial intelligence.* New York, NY: Basic Books, 1993.

[63] J. McCarthy and E. A. Feigenbaum, “In memoriam: Arthur samuel: Pioneer in machine learning,” *AI Magazine*, vol. 11, no. 3, p. 10, Sep. 1990. [Online]. Available: https://www.aaai.org/ojs/index.php/aimagazine/article/view/840

[64] F. Rosenblatt, “The perceptron–a perceiving and recognizing automaton,” *Cornell Aeronautical Laboratory*, 1957, report 85-460-1.

[65] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[66] Tin Kam Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[67] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: https://doi.org/10.1007/bf00994018

[68] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics.* Berkeley, Calif.: University of California Press, 1967, pp. 281–297. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992

[69] K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space," Nov. 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720

[70] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[71] T. Chen and C. Guestrin, "Xgboost," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016. [Online]. Available: http://dx.doi.org/10.1145/2939672.2939785

[72] J. Mockus, "On bayesian methods for seeking the extremum," in *Proceedings of the IFIP Technical Conference.* Berlin, Heidelberg: Springer-Verlag, 1974, p. 400–404.

[73] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer New York, 2009. [Online]. Available: https://doi.org/10.1007/978-0-387-84858-7

[74] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[75] S. Marshland, *Machine Learning : an algorithmic perspective.* Boca Raton, FL: CRC Press, 2015.

[76] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations.* Cambridge, MA: MIT Press, 1986, pp. 318–362.

[77] D. Saad, *On-line learning in neural networks.* Cambridge England New York: Cambridge University Press, 1998.

[78] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[79] M. Nielsen, "Neural networks and deep learning," 2017, online Book: http://neuralnetworksanddeeplearning.com, Accessed: 2017-05-27.

[80] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018. [Online]. Available: http://jmlr.org/papers/v18/17-468.html

[81] F. Wang, J. Decker, X. Wu, G. Essertel, and T. Rompf, "Backpropagation with callbacks: Foundations for efficient and expressive differentiable programming," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 10 180–10 191.

[82] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt, "A differentiable programming system to bridge machine learning and scientific computing," 2019.

[83] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, 09 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[85] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. 61, pp. 2121–2159, 2011. [Online]. Available: http://jmlr.org/papers/v12/duchi11a.html

[86] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251 – 257, 1991. [Online]. Available: http://www.sciencedirect.com/science/article/pii/089360809190009T

[87] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861 – 867, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005801315

[88] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, vol. 8, p. 143–195, 1999.

[89] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6231–6239. [Online]. Available: http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width.pdf

[90] B. Hanin and M. Sellke, "Approximating continuous functions by relu nets of minimal width," 2018.

[91] P. Kidger and T. Lyons, "Universal Approximation with Deep Narrow Networks," ser. Proceedings of Machine Learning Research, J. Abernethy and S. Agarwal, Eds., vol. 125. PMLR, 09–12 Jul 2020, pp. 2306–2327. [Online]. Available: http://proceedings.mlr.press/v125/kidger20a.html

[92] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000. [Online]. Available: https://doi.org/10.1038/35016072

[93] R. H. R. Hahnloser and H. S. Seung, "Permitted and forbidden sets in symmetric threshold-linear networks," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 217–223.

[94] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2016.

[95] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-normalizing neural networks,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 971–980. [Online]. Available: http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf

[96] A. L. Maas, “Rectifier nonlinearities improve neural network acoustic models,” 2013.

[97] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, “Incorporating second-order functional knowledge for better option pricing,” in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, ser. NIPS’00. Cambridge, MA, USA: MIT Press, 2000, p. 451–457.

[98] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[99] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[100] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *ArXiv*, vol. abs/1502.03167, 2015.

[101] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *ArXiv*, vol. abs/1207.0580, 2012.

[102] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[103] G. Kanwar, M. S. Albergo, D. Boyda, K. Cranmer, D. C. Hackett, S. Racanière, D. J. Rezende, and P. E. Shanahan, "Equivariant flow-based sampling for lattice gauge theory," *Phys. Rev. Lett.*, vol. 125, p. 121601, Sep 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.125.121601

[104] C. Gao, S. Höche, J. Isaacson, C. Krause, and H. Schulz, "Event generation with normalizing flows," *Phys. Rev. D*, vol. 101, p. 076002, Apr 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.101.076002

[105] E. Bothmann, T. Janßen, M. Knobbe, T. Schmale, and S. Schumann, "Exploring phase space with Neural Importance Sampling," *SciPost Phys.*, vol. 8, no. 4, p. 069, 2020.

[106] B. Nachman and D. Shih, "Anomaly detection with density estimation," *Phys. Rev. D*, vol. 101, p. 075042, Apr 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.101.075042

[107] R. Johnson, "The minimal transformation to orthonormality," *Psychometrika*, vol. 31, pp. 61–66, 1966.

[108] J. H. Friedman, "Exploratory projection pursuit," *Journal of the American Statistical Association*, vol. 82, no. 397, pp. 249–266, 1987. [Online]. Available: http://www.jstor.org/stable/2289161

[109] S. S. Chen and R. A. Gopinath, "Gaussianization," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 423–429. [Online]. Available: http://papers.nips.cc/paper/1856-gaussianization.pdf

[110] E. Tabak and C. Turner, "A family of nonparametric density estimation algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, pp. 145–164, 2013.

[111] O. Rippel and R. P. Adams, "High-dimensional probability estimation with deep density models," 2013.

[112] L. Dinh, D. Krueger, and Y. Bengio, "NICE: non-linear independent components estimation," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1410.8516

[113] G. Papamakarios, E. T. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *ArXiv*, vol. abs/1912.02762, 2019.

[114] J. Milnor, *Topology from the differentiable viewpoint.* Princeton, N.J: Princeton University Press, 1997.

[115] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *ArXiv*, vol. abs/1505.05770, 2015.

[116] A. Rényi, "On measures of entropy and information," in *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics.* Berkeley, Calif.: University of California Press, 1961, pp. 547–561. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512181

[117] A. Müller, "Integral probability metrics and their generating classes of functions," *Advances in Applied Probability*, vol. 29, no. 2, pp. 429–443, 1997. [Online]. Available: http://www.jstor.org/stable/1428011

[118] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 03 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[119] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 6571–6583. [Online]. Available: http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf

[120] M. Germain, K. Gregor, I. Murray, and H. Larochelle, "Made: Masked autoencoder for distribution estimation," ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 881–889. [Online]. Available: http://proceedings.mlr.press/v37/germain15.html

[121] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=HkpbnH9lx

[122] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville, "Neural autoregressive flows," ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 2078–2087. [Online]. Available: http://proceedings.mlr.press/v80/huang18d.html

[123] P. Jaini, K. A. Selby, and Y. Yu, "Sum-of-squares polynomial flow," in *ICML*, 2019.

[124] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4743–4751. [Online]. Available: http://papers.nips.cc/paper/6581-improved-variational-inference-with-inverse-autoregressive-flow.pdf

[125] G. Papamakarios, T. Pavlakou, and I. Murray, "Masked autoregressive flow for density estimation," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 2338–2347. [Online]. Available: http://papers.nips.cc/paper/6828-masked-autoregressive-flow-for-density-estimation.pdf

[126] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 10 215–10 224. [Online]. Available: http://papers.nips.cc/paper/8224-glow-generative-flow-with-invertible-1x1-convolutions.pdf

[127] N. D. Cao, I. Titov, and W. Aziz, “Block neural autoregressive flow,” in *UAI*, 2019.

[128] A. Wehenkel and G. Louppe, “Unconstrained monotonic neural networks,” in *BNAIC/BENELEARN*, 2019.

[129] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák, “Neural importance sampling,” *ACM Trans. Graph.*, vol. 38, pp. 145:1–145:19, 2018.

[130] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, “Cubic-spline flows,” *ArXiv*, vol. abs/1906.02145, 2019.

[131] ——, “Neural spline flows,” in *NeurIPS*, 2019.

[132] J. H. Ahlberg, *The Theory of Splines and Their Applications : Mathematics in Science and Engineering: A Series of Monographs and Textbooks, Vol. 38.* Saint Louis: Elsevier Science, 2016.

[133] J. A. Gregory and R. Delbourgo, “Piecewise Rational Quadratic Interpolation to Monotonic Data,” *IMA Journal of Numerical Analysis*, vol. 2, no. 2, pp. 123–130, 04 1982. [Online]. Available: https://doi.org/10.1093/imanum/2.2.123

[134] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[135] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in*

Natural Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: https://www.aclweb.org/anthology/D14-1179

[136] H. Larochelle and I. Murray, "The neural autoregressive distribution estimator," ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, 11–13 Apr 2011, pp. 29–37. [Online]. Available: http://proceedings.mlr.press/v15/larochelle11a.html

[137] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3104–3112. [Online]. Available: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[138] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video pixel networks," ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. International Convention Centre, Sydney, Australia: PMLR, 06–11 Aug 2017, pp. 1771–1779. [Online]. Available: http://proceedings.mlr.press/v70/kalchbrenner17a.html

[139] J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider, "Transformation autoregressive networks," ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 3898–3907. [Online]. Available: http://proceedings.mlr.press/v80/oliva18a.html

[140] J. Schmidhuber, D. Wierstra, and F. Gomez, "Evolino: Hybrid neuroevolution / optimal linear search for sequence prediction," in *In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJ-CAI.* Morgan, 2005, pp. 853–858.

[141] G. Petneházi, "Recurrent neural networks for time series forecasting," 2019.

[142] G. K. Anumanchipalli, J. Chartier, and E. F. Chang, "Speech synthesis from neural decoding of spoken sentences," *Nature*, vol. 568, no. 7753, pp. 493–498, Apr. 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1119-1

[143] Y. LeCun and Y. Bengio, *Convolutional Networks for Images, Speech, and Time Series.* Cambridge, MA, USA: MIT Press, 1998, p. 255–258.

[144] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[145] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[146] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[147] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "Learning semantic representations using convolutional neural networks for web search." WWW 2014, April 2014.

[148] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 2005, pp. 729–734 vol. 2.

[149] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[150] C. Gallicchio and A. Micheli, "Graph echo state networks," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010, pp. 1–8.

[151] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *2nd International Conference*

on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, Y. Bengio and Y. LeCun, Eds., 2014. [Online]. Available: http://arxiv.org/abs/1312.6203

[152] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[153] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1024–1034. [Online]. Available: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[154] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 1145–1152.

[155] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," 2018.

[156] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=SJiHXGWAZ

[157] Y. Li, Z. He, X. Ye, Z. He, and K. Han, "Spatial temporal graph convolutional networks for skeleton-based dynamic hand gesture recognition," *EURASIP Journal on Image and Video Processing*, vol. 2019, no. 1, Sep. 2019. [Online]. Available: https://doi.org/10.1186/s13640-019-0476-x

[158] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," 2018.

[159] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020.

[160] J. Shlomi, P. Battaglia, and J.-R. Vlimant, "Graph neural networks in particle physics," 2020.

[161] H. Qu and L. Gouskos, "Jet tagging via particle clouds," *Phys. Rev. D*, vol. 101, p. 056019, Mar 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.101.056019

[162] E. Bernreuther, T. Finke, F. Kahlhoefer, M. Krämer, and A. Mück, "Casting a graph net to catch dark showers," 2020.

[163] E. A. Moreno, O. Cerri, J. M. Duarte, H. B. Newman, T. Q. Nguyen, A. Periwal, M. Pierini, A. Serikova, M. Spiropulu, and J.-R. Vlimant, "JEDI-net: a jet identification algorithm based on interaction networks," *The European Physical Journal C*, vol. 80, no. 1, Jan. 2020. [Online]. Available: https://doi.org/10.1140/epjc/s10052-020-7608-4

[164] N. Choma, F. Monti, L. Gerhardt, T. Palczewski, Z. Ronaghi, M. Prabhat, W. Bhimji, M. Bronstein, S. Klein, and J. Bruna, "Graph neural networks for icecube signal classification," 12 2018, pp. 386–391.

[165] M. Abdughani, J. Ren, L. Wu, and J. M. Yang, "Probing stop pair production at the LHC with graph neural networks," *Journal of High Energy Physics*, vol. 2019, no. 8, Aug. 2019. [Online]. Available: https://doi.org/10.1007/jhep08(2019)055

[166] S. R. Qasim, J. Kieseler, Y. Iiyama, and M. Pierini, "Learning representations of irregular particle-detector geometry with distance-weighted graph networks," *The European Physical Journal C*, vol. 79, no. 7, Jul. 2019. [Online]. Available: https://doi.org/10.1140/epjc/s10052-019-7113-9

[167] J. Kieseler, "Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph, and image data," *The European Physical Journal C*, vol. 80, no. 9, Sep. 2020. [Online]. Available: https://doi.org/10.1140/epjc/s10052-020-08461-2

[168] H. Serviansky, N. Segol, J. Shlomi, K. Cranmer, E. Gross, H. Maron, and Y. Lipman, "Set2graph: Learning graphs from sets," 2020.

[169] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[170] M. Tanabashi *et al.*, "Review of Particle Physics," *Phys. Rev. D*, vol. 98, no. 3, p. 030001, 2018.

[171] K. Abe *et al.*, "Measurement of neutrino and antineutrino oscillations by the T2K experiment including a new additional sample of ν_e interactions at the far detector," *Phys. Rev.*, vol. D96, no. 9, p. 092006, 2017, [Erratum: Phys. Rev.D98,no.1,019902(2018)].

[172] ——, "Constraint on the matter–antimatter symmetry-violating phase in neutrino oscillations," *Nature*, vol. 580, no. 7803, pp. 339–344, 2020.

[173] ——, "Measurement of double-differential muon neutrino charged-current interactions on C_8H_8 without pions in the final state using the T2K off-axis beam," *Phys. Rev.*, vol. D93, no. 11, p. 112012, 2016.

[174] Y. Hayato, "A neutrino interaction simulation program library NEUT," *Acta Phys. Polon.*, vol. B40, pp. 2477–2489, 2009.

[175] K. Abe, N. Abgrall, H. Aihara, T. Akiri, J. B. Albert, C. Andreopoulos, S. Aoki, A. Ariga, T. Ariga, S. Assylbekov, D. Autiero, M. Barbi, G. J. Barker, G. Barr *et al.*, "T2k neutrino flux prediction," *Phys. Rev. D*, vol. 87, p. 012001, Jan 2013. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.87.012001

[176] M. Tanabashi, K. Hagiwara, K. Hikasa, K. Nakamura *et al.*, "Review of particle physics," *Phys. Rev. D*, vol. 98, p. 030001, Aug 2018.

[177] R. Barlow, "Extended maximum likelihood," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 297, no. 3, pp. 496–506, Dec. 1990.

[178] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=Skq89Scxx

[179] M.-H. Chen and Q.-M. Shao, "Monte carlo estimation of bayesian credible and hpd intervals," *Journal of Computational and Graphical Statistics*, vol. 8, no. 1, pp. 69–92, 1999. [Online]. Available: http://www.jstor.org/stable/1390921

[180] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 04 1970. [Online]. Available: https://doi.org/10.1093/biomet/57.1.97

[181] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984.

[182] G. Casella, C. P. Robert, and M. T. Wells, *Generalized Accept-Reject sampling schemes*, ser. Lecture Notes–Monograph Series. Beachwood, Ohio, USA: Institute of Mathematical Statistics, 2004, vol. Volume 45, pp. 342–347. [Online]. Available: https://doi.org/10.1214/lnms/1196285403

[183] L. Martino and J. Míguez, "Generalized rejection sampling schemes and applications in signal processing," *Signal Processing*, vol. 90, no. 11, pp. 2981 – 2995, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168410001866

[184] C. M. Bishop and N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electronic Imaging*, vol. 16, p. 049901, 2007.

[185] J. von Neumann, "Various techniques used in connection with random digits," in *Monte Carlo Method*, ser. National Bureau of Standards Applied Mathematics Series, A. S. Householder, G. E. Forsythe, and H. H. Germond, Eds. Washington, DC: US Government Printing Office, 1951, vol. 12, ch. 13, pp. 36–38.

[186] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*. Springer New York, 2004. [Online]. Available: https://doi.org/10.1007/978-1-4757-4145-2

[187] H. Kahn and A. W. Marshall, "Methods of reducing sample size in monte carlo computations," *Journal of the Operations Research Society of America*, vol. 1, no. 5, pp. 263–278, 1953. [Online]. Available: http://www.jstor.org/stable/166789

[188] L. Alvarez-Ruso *et al.*, "NuSTEC White Paper: Status and challenges of neutrino–nucleus scattering," *Prog. Part. Nucl. Phys.*, vol. 100, pp. 1–68, 2018.

[189] M. Acero *et al.*, "First Measurement of Neutrino Oscillation Parameters using Neutrinos and Antineutrinos by NOvA," *Phys. Rev. Lett.*, vol. 123, no. 15, p. 151803, 2019.

[190] R. Smith and E. Moniz, "NEUTRINO REACTIONS ON NUCLEAR TARGETS," *Nucl. Phys. B*, vol. 43, p. 605, 1972, [Erratum: Nucl.Phys.B 101, 547 (1975)].

[191] J. S. Liu, "Metropolized independent sampling with comparisons to rejection sampling and importance sampling," *Statistics and Computing*, vol. 6, no. 2, pp. 113–119, Jun. 1996. [Online]. Available: https://doi.org/10.1007/bf00162521

[192] M. H. Ahn *et al.*, "Measurement of Neutrino Oscillation by the K2K Experiment," *Phys. Rev. D*, vol. 74, p. 072003, 2006.

[193] P. Adamson *et al.*, "Combined analysis of ν_μ disappearance and $\nu_\mu \to \nu_e$ appearance in MINOS using accelerator and atmospheric neutrinos," *Phys. Rev. Lett.*, vol. 112, p. 191801, 2014.

[194] M. A. Acero *et al.*, "New constraints on oscillation parameters from ν_e appearance and ν_μ disappearance in the NOvA experiment," *Phys. Rev.*, vol. D98, p. 032012, 2018.

[195] C. Rubbia, "The Liquid Argon Time Projection Chamber: A New Concept for Neutrino Detectors," 5 1977.

[196] A. Blondel, F. Cadoux, S. Fedotov, M. Khabibullin, A. Khotjantsev, A. Korzenev, A. Kostin, Y. Kudenko, A. Longhin, A. Mefodiev, P. Mermod, O. Mineev, E. Noah, D. Sgalaberna, A. Smirnov, and N. Yershov, "A fully-active fine-grained detector with three readout views," *Journal of Instrumentation*, vol. 13, no. 02, pp. P02 006–P02 006, feb 2018. [Online]. Available: https://doi.org/10.1088%2F1748-0221%2F13%2F02%2Fp02006

[197] K. Abe *et al.*, "T2K ND280 Upgrade - Technical Design Report," 2019.

[198] G. Yang, "3D Projection Scintillator Tracker in the DUNE Near Detector," *PoS*, vol. ICHEP2018, p. 868, 2019.

[199] O. Mineev *et al.*, "Beam test results of 3D fine-grained scintillator detector prototype for a T2K ND280 neutrino active target," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 923, pp. 134–138, 2019.

[200] A. Aurisano *et al.*, "A convolutional neural network neutrino event classifier," *Journal of Instrumentation*, vol. 11, no. 09, p. P09001, 2016. [Online]. Available: http://stacks.iop.org/1748-0221/11/i=09/a=P09001

[201] R. Acciarri *et al.*, "Convolutional Neural Networks Applied to Neutrino Events in a Liquid Argon Time Projection Chamber," *JINST*, vol. 12, no. 03, p. P03011, 2017.

[202] B. Abi *et al.*, "Neutrino interaction classification with a convolutional neural network in the DUNE far detector," 2020. [Online]. Available: http://arxiv.org/abs/2006.15052

[203] ——, "The Single-Phase ProtoDUNE Technical Design Report," 6 2017.

[204] C. Adams *et al.*, "Deep neural network for pixel-level electromagnetic particle identification in the microboone liquid argon time projection chamber," *Phys. Rev. D*, vol. 99, p. 092001, May 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.99.092001

[205] S. Farrell, P. Calafiura, M. Mudigonda, Prabhat, D. Anderson, J.-R. Vlimant, S. Zheng, J. Bendavid, M. Spiropulu, G. Cerati, L. Gray, J. Kowalkowski, P. Spentzouris, and A. Tsaris, "Novel deep learning methods for track reconstruction," 2018.

[206] S. R. Qasim, J. Kieseler, Y. Iiyama, and M. Pierini, "Learning representations of irregular particle-detector geometry with distance-weighted graph networks," *The European Physical Journal C*, vol. 79, no. 7, Jul 2019. [Online]. Available: http://dx.doi.org/10.1140/epjc/s10052-019-7113-9

[207] X. Ju, S. Farrell, P. Calafiura, D. Murnane, Prabhat, L. Gray, T. Klijnsma, K. Pedro, G. Cerati, J. Kowalkowski, G. Perdue, P. Spentzouris, N. Tran, J.-R. Vlimant, A. Zlokapa, J. Pata, M. Spiropulu, S. An, A. Aurisano, J. Hewes, A. Tsaris, K. Terao, and T. Usher, "Graph Neural Networks for Particle Reconstruction in High Energy Physics detectors," 2020.

[208] C. Andreopoulos *et al.*, "The GENIE Neutrino Monte Carlo Generator," *Nucl. Instrum. Meth. A*, vol. 614, pp. 87–104, 2010.

[209] K. Abe *et al.*, "T2K neutrino flux prediction," *Phys. Rev.*, vol. D87, no. 1, p. 012001, 2013, [Addendum: Phys. Rev.D87,no.1,019902(2013)].

[210] S. Agostinelli *et al.*, "GEANT4: A Simulation toolkit," *Nucl. Instrum. Meth.*, vol. A506, pp. 250–303, 2003.

[211] J. B. Birks, "Scintillations from Organic Crystals: Specific Fluorescence and Relative Response to Different Radiations," *Proceedings of the Physical Society. Section A*, vol. 64, no. 10, pp. 874–877, oct 1951. [Online]. Available: https://doi.org/10.1088%2F0370-1298%2F64%2F10%2F303

[212] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8026–8037.

[213] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[214] M. Herdin, N. Czink, H. Ozcelik, and E. Bonek, "Correlation matrix distance, a meaningful measure for evaluation of non-stationary MIMO channels," in *2005 IEEE 61st Vehicular Technology Conference*, vol. 1, 2005, pp. 136–140 Vol. 1.

[215] M. Klein, G. J. Rogers, and P. Kundur, "A fundamental study of inter-area oscillations in power systems," *IEEE Transactions on Power Systems*, vol. 6, no. 3, pp. 914–921, 1991.

[216] N. Janssens and A. Kamagate, "Interarea oscillations in power systems," *IFAC Proceedings Volumes*, vol. 33, no. 5, pp. 217 – 226, 2000, iFAC Symposium on Power Plants and Power Systems Control 2000, Brussels, Belgium, 26-29 April 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017409633

[217] R. Steidel, *An introduction to mechanical vibrations.* New York: Wiley, 1971, p. 39.

[218] A. Phadke, "Synchronized phasor measurements-a historical overview," in *IEEE/PES Transmission and Distribution Conference and Exhibition.* IEEE, 2002. [Online]. Available: https://doi.org/10.1109/tdc.2002.1178427

[219] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154. [Online]. Available: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[220] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[221] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 3844–3852.

[222] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[223] D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241 – 259, 1992. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005800231

[224] B. Donon, B. Donnot, I. Guyon, and A. Marot, “Graph neural solver for power systems,” in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[225] B. Donon, R. Clément, B. Donnot, A. Marot, I. Guyon, and M. Schoenauer, “Neural networks for power flow: Graph neural solver,” *Electric Power Systems Research*, vol. 189, p. 106547, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0378779620303515

[226] D. Owerko, F. Gama, and A. Ribeiro, “Predicting power outages using graph neural networks,” in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2018, pp. 743–747.

[227] R. Koenker and G. Bassett, “Regression quantiles,” *Econometrica*, vol. 46, no. 1, pp. 33–50, 1978. [Online]. Available: http://www.jstor.org/stable/1913643

[228] B. S. Cade and B. R. Noon, “A gentle introduction to quantile regression for ecologists,” *Frontiers in Ecology and the Environment*, vol. 1, no. 8, pp. 412–420, Oct. 2003. [Online]. Available: https://doi.org/10.1890/1540-9295(2003)001[0412:agitqr]2.0.co;2

[229] K. Abe, R. Akutsu, A. Ali, J. Amey, C. Andreopoulos, L. Anthony, M. Antonova, S. Aoki, A. Ariga, Y. Ashida, Y. Azuma, S. Ban, M. Barbi, G. J. Barker *et al.*, "Search for *cp* violation in neutrino and antineutrino oscillations by the t2k experiment with 2.2×10^{21} protons on target," *Phys. Rev. Lett.*, vol. 121, p. 171802, Oct 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.121.171802