Supersymmetry Cross Section Predictions with Bayesian Neural Networks

Per-Dimitri B. Sønderland

Thesis submitted for the degree of
Master of Science in Nuclear and Particle Physics

Department of Physics
University of Oslo

04.06.2021

Contents

Acknowledgments

Først og fremst vil jeg takke min kjære Vala. Hadde jeg ikke møtt (diltet etter) henne på mitt først år som fysikk-student, så er jeg temmelig sikker på at jeg ikke hadde sittet her i dag og skrevet ferdig en master i fysikk. Takk for dine støtte og varme! Jeg vil også takke Are Raklev for utmerket veiledning, og generelt sett bare være en kul type, jeg kunne ikke fått en veilder som passet meg bedre. En takk må også rettes til Prof. Raklevs elite tropp av teoretikere som ukentlig ga meg råd og engasjerte seg i diskujoner om hva som går og ikke går, så takk til Anders Kvellestad og Jeriek Vda. Videre vil jeg takke familien min, mamma for å være en støtte og stor ressurs gjennom hele mitt liv, og takk til Bjørn Magne for å alltid stille opp rett og slett være en fantastisk person å ha i livet sitt. Jeg vil takke pappa for at jeg i det hele tatt fant på å ta fysikk. De uendelig mange turene på teknisk museum som ung hadde en effekt. Takk til min søster og familie for at dere er så utrolig festlige og flotte folk. The CPU and GPU intensive computations were performed on resources provided by UNINETT Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway, allocation nn9284k.

Abstract

This thesis examines the use of Bayesian neural networks for the purpose of evaluating supersymmetric cross sections at next-to-leading order with reliable uncertainty estimates. Calculating cross sections is a central part of the search for new physics beyond the Standard Model, however, it is a time consuming endeavour. Machine learning methods can speed up the production process by learning the relationships between physical parameters and cross sections, however, the uncertainty on their predictions is for most methods unavailable. A variational Bayesian neural network was trained on cross section data produced by `Prospino 2.1` for electroweakino pair production in proton-proton collisions. To improve its performance deterministic pretraining was implemented and various annealing schemes were tested. We found that with the additional implementations the Bayesian neural network consistently produced reliable uncertainties and favourable predictions with relatively short run-times.

Introduction

The Standard Model (SM) of particle physics is one of man's greatest achievements to date, culminating a century of fundamental research by physicists. The SM classifies and describes the interactions of the basic building blocks of nature, namely the fundamental particles. Through a wide variety of experiments the SM has stood the test of time like few other theoretical frameworks. However, it is not a complete theory of nature and its deficiencies are most pronounced in the description of energies above the eletroweak scale.

Specifically, in order to reconcile the measured Higgs boson mass [1] with the corresponding SM prediction, a huge amount of fine-tuning of its parameters is required. This is called the *hierarchy problem*, and it is deemed a problem in the same manner as we in hindsight may look upon superseded theories of the past. One example is the ancient Ptolemaic geocentric model of the solar system, where through enormous fine tuning one can through the use of *epicycles* predict the retrograde motions of the planets, but as we know today, its fundamental assumption was wrong.

The appeal of a theory that does not require excessive fine-tuning comes natural, and such a theory is conveniently said to exhibit *naturalness*. A proposed extension to the SM that deals with the hierarchy problem, among other shortcomings, is called *supersymmetry*. Its defining attribute is simple: It suggest a symmetry between the two particle families of nature, namely the fermions and bosons. In supersymmetry, for every SM fermion there exists a bosonic supersymmetric partner, and similarly for every SM boson there exists a fermionic supersymmetric partner. These supersymmetric particles are called *sparticles*. The only difference between an SM particle and its supersymmetric partner is that their spin differs by $\frac{1}{2}$.

At the time of writing no sparticles have been discovered, but the energy regimes of 13 TeV center of mass energy, at which sparticles by some estimates are expected to reside [2], are currently being probed at the Large Hadron Collider (LHC) at CERN. Collider experiments investigate the subatomic realm where length scales are small, therefore searching for signs of supersymmetry in LHC data is essentially an act of methodically looking for a needle in a haystack. The most sophisticated are performed by a method called *global fits*, where by producing predictions according to theory and comparing to data one may narrow down the realm of possibilities of where supersymmetry can and cannot exist.

The observables searched for in LHC data depend on theoretical predictions of supersymmetric cross sections. This is done by perturbative quantum field theory, particularly, the supersymmetry model of interest in this thesis is the *Minimal Supersymmetric Standard Model*. The state-of-the-art computer programs that are suited for this purpose are `Prospino 2.1` [3] and `NLL-fast 2.1` [4], of which the latter is, as its name

implies, a fast method for cross section calculations, but at the expense of a restricted parameter space. The former method does not have such limitation and is the program of choice in this thesis, however, it is very time expensive. To calculate these cross section involves hefty numerical integration, and producing just a single cross section at the femtobarn scale (fb = 10^{-43}m^{-2}) takes in the order of minutes on a standard computer. In the quest for new physics in global fits this is problematic as it implies the need for billions of points in parameter space, but we can possibly do better.

The run times of cross section predictions can be improved by attempting to "learn" the complex relationship between the parameter inputs and the cross section outputs, so that we can reproduce the relationship to a certain degree of accuracy in an efficient manner for new parameter choices. This has been done with success using various machine learning methods [5], however, these methods are not full proof as they are always at risk of making erroneous predictions. They are fully dependent of having "seen" relationships that are sufficiently similar to what they are meant to predict, but it may be hard to assess this sufficiency. This issue can be resolved if a computational method provides an reliable uncertainty together with every prediction, hence disclosing the quality of a prediction. In this thesis we seek to do just that. We do it by applying Bayesian inference to machine learning regression, specifically employing the computing structure *Bayesian neural networks*. Our goal is for the structure to reliably provide high uncertainties when predictions deviate from the true `Prospino 2.1` cross section values they are meant to predict.

This thesis is structured as follows: In Chapter 1 we first investigate the shortcomings of the Standard Model, such as the hierarchy problem, then we give an introduction to supersymmetry and the specific model used in this thesis. In Chapter 2 we enter the realm of machine learning and give a basic overview of supervised learning in the context of regression. In Chapter 3 we outline the deterministic neural network computing structure, and in Chapter 4 we introduce the essentials of Bayesian statistics and take a probabilistic look at the problem of regression. In Chapter 5 we investigate the use of entropy and the relative entropy as a means for probabilistic optimization. We then tie the previously covered deterministic neural network together with probabilistic concepts and define the variational Bayesian neural network. Next we discuss various manner in which to improve performance, such as annealing schemes and deterministic pretraining. In Chapter 6 we elaborate on the data preprocessing and scaling that has been used and the metrics used for performance evaluation. In Chapter 7 we present the results from the prediction of the electroweakino production process $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$ and $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$, evaluate the quality of the uncertainties, before we finally present our conclusions and suggestions for further work and improvements.

Part I

Physics Background

Chapter 1

Supersymmetry

In this chapter we study the underpinnings of supersymmetry, a theoretical framework which serves as an extension to the Standard Model of particle physics. We shall first cover briefly the Standard Model and explore some of the reasons as to why physicists have devised an expansion package to this remarkable model, which to date has been one of man's most successful theoretical undertakings. We only give the Standard Model a short visit, as doing so we assume on the behalf the reader some familiarity with the topic, along with its mathematical basis in quantum field theory when we subsequently introduce and outline supersymmetry. To keep some sections concise we will invariably postpone some formulae from group theory and supersymmetry in Appendix A.

1.1 The Standard Model of Particle Physics

In the first part of the 20th century, when the first stirrings of knowledge about the subatomic realm began, new particles were discovered at an increasing pace. The introduction of particle accelerators, allowing for high energy collisions, at the start of the latter half of the 20th century lead to an explosion in new discoveries. The large variety of detected particles were deemed a subnuclear *zoo* [6], because at that point in time an overarching framework for the seemingly unrelated species of particles was yet to be devised. The eventual framework for this zoo was created in multiple stages and finalized in the 1970s, and is called the *Standard Model* (SM) of particle physics. Its main feat was to categorize the elementary particles into a clearly defined structure of classes, that describes how the particles of this zoo is expected to behave out in the wild. Firstly, it poses that all particles come in two types: fermions and bosons. The classifying feature for this categorizaiton is their *spin* (their intrinsic angular momentum), where half-integer spin particles ($\frac{1}{2}$) are the fermions and integer spin particles $(0, 1)$ are bosons. Fermions obey the *Pauli exclusion principle*, such that only one may occupy a certain quantum state at a time. Elementary fermions are divided into two types, *quarks* and *leptons*, where both are further subdivided into six flavours of particles each defined by their flavour quantum numbers. Fermions interact with one another through the four forces of nature to a varying degree, three of these forces are described by the Standard Model, leaving out gravity. The mediation of force is where the bosons role comes into play, as most of the bosons are force carrying particles. The Standard Model specifies five types of bosons, four of which are the carriers of force, namely the *vector bosons* or *gauge bosons*, depending on the context, with "vector" alluding to their spin of 1. The four gauge bosons are the *photon* γ, which is the carrier of the electromagnetic

interaction, the eight *gluons* g, which are the carriers of the strong interaction, and lastly the neutral weak boson Z and the two charged weak bosons $W^{\pm}$, which mediate the weak interaction. The fifth boson type of the Standard Model is the *Higgs boson*, which has a spin of 0. While it is not traditionally grouped in as one of the forces of nature it makes itself forcefully present, as the Higgs mediated interaction is the manner by which elementary particles acquire their mass, emerging as a consequence of their interactions with the Higgs field. All the labels given above come with mathematical attributes in quantum field theories allowing us to model their nature and how they interact, specifically we may devise equations of motion derived from the Standard Model *Lagrangian*, which can be seen as the mathematical culmination of what constitutes the Standard Model.

Shortcomings of the Standard Model Physicist have through the recent decades put the Standard Model to the test through a large number of experiments, and for many purposes it has stood the test of time. One of its greatest triumph is in the prediction of the anomalous magnetic dipole moment of the electron $a_e \equiv (g-2)/2$, where g is called the g-factor. The discrepancy between theoretical calculations and experiment for a_e is at

$$\Delta a_e \equiv a_e^{\text{exp}} - a_e^{\text{SM}} = -(8.8 \pm 3.6) \times 10^{-13}, \tag{1.1}$$

where the theoretical results is 2.4 standard deviations *below* the measured experimental value. Considering the order of the discrepancy this is astoundingly accurate [7].

However, recent experiments on muon $g-2$, where the muon is the electron's heavier cousin, has put the Standard Model on possibly shaky ground and might point to new physics. Specifically, as it stands a_μ^{exp} exceeds the Standard Model predictions with a combined experimental significance of 4.2 standard deviations [8]. For both the electron and the muon the g-factor is equal to 2 if solely calculated by *relativistic quantum physics*, but for higher order contributions from quantum field theory the value increases to slightly above 2. These contributions account for the notion that the muon and the electron interacts with a sea of virtual particles, hence, if experimental measurements of $g-2$ are different from the theoretical calculations ($\Delta a_\mu \neq 0$) it might indicate that it is interacting with particles that are not accounted for.

Since we have mentioned that the Standard Model describes three out of the four forces of nature, we note, once more, that gravity is not described by the Standard Model, but by the theory of *general relativity*. The Standard Model and general relativity can together be deemed the current paradigm of modern (relativistic) physics, but it is also their stringent disagreement at high energy that have kept theoretical physicists hard at work for decades on end. Hence a theory which describes all the forces of nature under a single framework is needed. Additionally, while the Standard Model does unify two of the forces as the electroweak interaction, it does not provide a mechanism for unification with the strong interaction, a mechanism called *gauge-coupling unification*.

Furthermore, evidence suggests that an unknown particle amounts to about 27% of the energy content of the observable universe [9]. A particle consistent with observation is thought to be a weakly interacting massive particle (WIMP), because as it stands the Standard Model does not provide an obvious candidate that fit those criteria.

These shortcomings motivate an extension to the Standard Model, in which different models of supersymmetry serve as viable candidates. Of greater concern to our subsequent discussion is a specific shortcoming not yet mentioned, namely the *Hierarchy*

problem, which is related to the elusive Higgs boson. We will outline this problem in Section 1.3, but first we give a short primer on what supersymmetry entails.

1.2 Supersymmetry – A Very Short Primer

Supersymmetry (SUSY), as the name suggest, introduces a symmetry, namely between fermions and bosons. Colloquially speaking we may say that fermions and bosons are two sides of the same coin. The symmetry entails that every particle has a superpartner, that differs only by a spin of $\frac{1}{2}$, so for an unbroken super symmetry the mass and quantum numbers, except the spin, remain the same. We can summarize this idea with an anticommuting operator Q that transforms the spin of a particle by $\frac{1}{2}$ such that we have

Supersymmetry Operator – The Basic Idea

$$Q|\text{Boson}\rangle = |\text{Fermion}\rangle, \quad Q|\text{Fermion}\rangle = |\text{Boson}\rangle. \tag{1.2}$$

For the fermions these supersymmetric partners are called *sfermions*, and for the bosons they are called *bosinos*, where both fall under the label *sparticles*. The naming rule of adding the prefix "s" for fermionic superpartners and the suffix "-ino" for bosonic superpartners applies to the subclasses of particles as well. For example the quarks have partners called squarks, and the gluons have partners called gluinos.

1.3 The Hierarchy Problem

Where the Standard Model falls short is where the motivation for an extension lies. Here we outline one of these shortcomings in greater detail and how it motivates the extension of supersymmetry.

The Higgs Mechanism In the Standard Model Lagrangian, the Higgs field presents itself as a complex scalar field $\phi(x)$ with potential

$$V(\phi) = \mu^2\phi^\dagger\phi + \lambda\left(\phi^\dagger\phi\right)^2. \tag{1.3}$$

For the Standard Model to be consistent with experiment the gauge symmetry $SU(2)_L \times U(1)_Y$ of the electroweak interaction must be broken. This symmetry is *spontaneously broken*, meaning that the Lagrangian of the theory still obides by the symmetry, but the minimum-energy vacuum solutions do not. This is ensured when a non-vanishing *vacuum expectiation value* (VEV) of the Higgs field is attained at the minimum of the potential in Equation (1.3), meaning that the minimum must satisfy $\langle\phi\rangle \neq 0$. For this to be the case we need the parameters of Equation (1.3) to satisfy $\lambda > 0$ and $\mu^2 < 0$, which yields a minimum of

$$\langle\phi\rangle = \sqrt{\frac{-\mu^2}{2\lambda}} = \frac{v}{\sqrt{2}}, \tag{1.4}$$

where the latter equality gives the vacuum expectation value v. From the electroweak interaction model (Glashow–Salam–Weinberg model) we have that

$$m_\text{W} = \frac{1}{2}g_\text{W}v, \tag{1.5}$$

where m_{W} is the mass of the $W^{\pm}$ boson and g_{W} is the coupling constant of the weak interaction, both measureable quantities. Using the measured values for these quantities we obtain a vacuum expectation of $v = 246$ GeV, which is the scale at which the electroweak interaction is unified (or broken). We can expand the potential in Equation (1.3) in terms of $\phi(x) = v + h(x)$ to second order in $h(x)^2$, where the scalar field $h(x)$ describes the physical Higgs boson. This gives

$$V = V_0 + \lambda v^2 h^2, \tag{1.6}$$

which gives us the Higgs boson mass

$$m_h^2 = 2\lambda v^2 \quad \Rightarrow \quad m_h = \sqrt{2}|\mu|, \tag{1.7}$$

where we inserted for v, and have that the Higgs mass is proportional to the Higgs mass parameter of the potential.

The Big Hierarchy Problem As it stands, Equation (1.7) completely ignores the effect particle interactions have on the Higgs mass. To account for interactions we write

$$m_h^2 = 2|\mu|^2 + \Delta m_h^2, \tag{1.8}$$

where the first term from the Lagrangian is called the *bare mass* $m_{0,h}^2 = 2|\mu|^2$, and the second term Δm_h^2 is the loop-contributions stemming from particle interactions with the Higgs field. Figure 1.1 depicts Feynman diagrams of such contributions, the largest of which is the top quark contribution at a coupling of about $\lambda_t \sim 1$. These loop-contributions diverge such that infinities arise. Divergences of this type are called *ultraviolet divergences* (UV) and arise either as energy becomes unbounded (in momentum space) or as distances of a propagator become zero (in position space). UV divergences can be handled by applying a regularization scheme to divergent terms in order to obtain a finite value. Specifically, the procedure can involve applying a cut-off, Λ_{UV}, in the limits of the relevant integral. This cut-off is usually set at high energies, such as the planck scale: $\Lambda_{UV} = M_P = 2.4 \times 10^{18}$GeV, beyond which we can be pretty certain that the Standard Model is invalid, as it requires an inclusion of quantum gravity. The loop corrections to the Higgs mass from the diagrams in Figure 1.1 are

$$\Delta m_h^2 = -\frac{|\lambda_f|^2}{8\pi^2}\Lambda_{UV}^2 + \frac{\lambda_s}{16\pi^2}\Lambda_{UV}^2 + \dots, \tag{1.9}$$

where λ_f and λ_s are the coupling constants of fermions and scalar particles to the Higgs field. For the calculation of the Higgs mass to conform to the experimentally measured value of 125 GeV we would need a massive cancellation of terms in Equation (1.9). One could try to fine tune the parameters of each particle contribution in Δm_h^2, making sure that a given term has the magnitude and sign to cancel another term with equivalent magnitude and opposite sign, such that the condition $m_h = 125$ GeV is always met. However, this is considered a violation of *naturalness* in that it requires a stupendous degree of fine tuning. Naturalness is a hotly debated topic, as it is not a physical law of nature (as far as we know), but rather a rational expectation.

Supersymmetry seeks to resolve the hierarchy problem by introducing a symmetry between fermions and bosons. It postulates that there are precisely twice as many bosons as there are fermions, and to equate the couplings as follows

$$|\lambda_f|^2 = \lambda_s. \tag{1.10}$$

This solves the hierarchy problem, in that it causes an exact cancellation of the quadratic divergences in Equation (1.9) effecitvely leaving us with the only the bare mass $m_{0,h}$ in Equation (1.8).

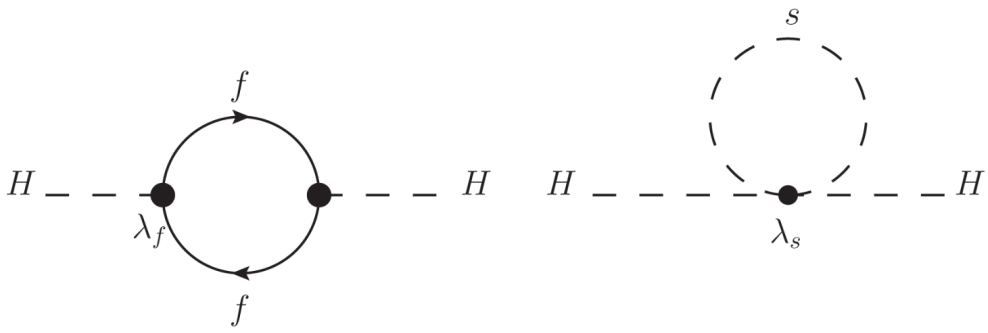


Figure 1.1: One-loop diagrams contributing to the Higgs mass m_h, left is a fermion loop, and right is a scalar loop. Figure taken from [10].

The Little Hierarchy Problem Ufortunately, this alone does not solve the problem in the universe we reside, as if it was the case we should have already observed sparticles, hence the symmetry must be broken. The manner in which to solve this problem is to introduce *soft supersymmetry breaking*, where we do retain some contributions from Δm_h^2 to the Higgs mass, but with an upper limit of at most

$$\Delta m_h^2 = -\frac{\lambda_s}{16\pi^2} m_s^2 \ln \frac{\Lambda_{UV}^2}{m_s^2} + \dots, \tag{1.11}$$

at leading order in Λ_{UV}. The term m_s is the mass scale of the soft breaking terms, and in order for Higgs mass corrections to be sufficiently small we require that m_s is relatively small as well, specifically at the scale of about $m_s \sim \mathcal{O}(1\text{TeV})$. However, new particles have not been observed at this energy scale, which is often deemed the *little hierarchy problem* [10].

1.4 Super-Poincaré Algebra

For any relativistic field theory the Poincaré group is the full symmetry group (in flat space) of the theory. Consequentially, all particles of the Standard Model can be described as irreducible representations of the Poincaré group. For a supersymmetry theory we will need to extend these symmetries. On the face of it, extending the Poincaré algebra to include internal gauge symmetries seemed undoable in a non-trivial way, as posited by Coleman and Mandula [11] in what is called the "no-go theorem". However, Haag, Lopuszanski and Sohnius gave the no-go theorem a revision in 1974 by extending the concept of Lie algebras used for the symmetry group [12]. The revised "positive" version of the no-go theorem is here shortly summarized by Sohnius himself [13]

> "All generators of supersymmetries must be fermionic, i.e., they must change the spin by a half-odd amount and change the statistics of the state."

The extension to the Poincaré algebra that allows for fermionic generators is a *graded Lie algebra*, which is summarized in Appendix A. By the use of this algebra we can formalize supersymmetry by extending the Poincaré algebra to what is called the *super-Poincaré algebra*. The simplest manner in which we can perform this extension is by adding to the Poincaré algebra a Majorana spinor with the components Q_a, where $a = 1, \dots 4$, where these operators are called *spinor charges*. A more convenient way to formulate this extension is to define it in terms of two-component Weyl spinors Q_A and $\bar{Q}_{\dot{A}}$, where

the latter is the hermitian conjugate of the former. The relationship between Majorana spinors and Weyl spinors is $Q_a = \binom{Q_A}{\bar{Q}^{\dot{A}}}$. The resulting algebra of commutators and anti-commutators, in terms of Weyl spinors, is then

Super-Poincaré Algebra – Weyl Spinors

$$\{Q_A, Q_B\} = \{\bar{Q}_{\dot{A}}, \bar{Q}_{\dot{B}}\} = 0, \tag{1.12}$$
$$\{Q_A, \bar{Q}_{\dot{B}}\} = 2\sigma^{\mu}_{A\dot{B}} P_\mu, \tag{1.13}$$
$$[Q_A, P_\mu] = [\bar{Q}_{\dot{A}}, P_\mu] = 0 \tag{1.14}$$
$$[Q_A, M^{\mu\nu}] = \sigma_A^{\mu\nu B} Q_B, \tag{1.15}$$

where we have that $\sigma^{\mu\nu} = \frac{i}{4}(\sigma^\mu \bar{\sigma}^\nu - \sigma^\nu \bar{\sigma}^\mu)$, which is related to the standard pauli matrices by $\sigma^\mu = (\mathbf{I}_{2\times 2}, \sigma^i)$, where $\mathbf{I}$ is the identity matrix.[1] The dotted and undotted indicies run over values $A = 1, 2$ and $\dot{A} = \dot{1}, \dot{2}$, respectively. The dot is a reminder that the corresponding operator transforms according to the conjugate representation; left-handed Weyl spinor representation $(\frac{1}{2}, 0)$, with a total spin of $1\frac{1}{2}$ and representation dimensions of two. The undotted is then naturally the left-handed Weyl spinor representation $(0, \frac{1}{2})$. Both are representation of the special linear group $SL(2, \mathbb{C})$. Lastly P_μ are the translations- and M_μ the boost and rotations of the Poincaré algebra.

As it stands, Equations (1.12) to (1.15) do not include any internal symmetry groups. We can assume sets of additional generators, that is, spinor charges $Q_A^\alpha = 1, \ldots, N$, that transforms in accordance to a representation of a compact gauge Lie Group. The algebra corresponding to $N = 1$ is called the *supersymmetry algebra*, while algebras $N > 1$ are called *extended supersymmetry algebras*. Usually only a single supersymmetry $N = 1$ is assumed to exist in nature, as additional supersymmetries $N > 1$ would introduce an extensive number of extra particles. We can end this section by the statement: Under some reasonable assumptions, one can prove that the super-Poincaré algebras are the largest possible extensions that can be made to special relativity [10].

1.5 Superspace

The Q operators we defined in the previous section, that is, the elements of the super-Poincaré algebra, act upon the quantity called superfields, which are analogous to the fields of the Standard Model and quantum field theory. But whereas the fields of the Standard Model live in 4-dimensional Minkowski space $x^\mu = (t, x, y, z)$, the superfields live in a 8-dimensional manifold called *superspace*,

$$z^\pi = \left(x^\mu, \theta^A, \bar{\theta}^{\dot{A}}\right), \tag{1.16}$$

such that any superfield is a function of these coordinates in superspace. The first four coordinates are the coordinates of Minkowski space x^μ, which are commuting dimensions, or bosonic degrees of freedom. The four remaining coordinates are the fermionic degrees of freedom. These are Grassman numbers θ that anti-commute with one another but not with ordinary real numbers. The indicies A and $\dot{A}$ have the same meaning as before (see Section 1.4), and thus these quantities are Grassmann numbers contained in the two-component Weyl spinors[2]. The anticommutation relations of these elements

[1] The Pauli matrices are listed in Appendix A.

[2] We note that the introduction of indexed Grassmann numbers allows us to formulate the super-Poincaré Algebra only in terms of commutators. This algebra is listed in Appendix A.1

Component field	Type	D.o.f.
$f(x), m(x), n(x)$	Complex (pseudo) scalar	2
$\psi_A(x), \phi_A(x)$	Left-handed Weyl spinor	4
$\bar{\chi}^{\dot{A}}(x), \bar{\lambda}^{\dot{A}}(x)$	Right-handed Weyl spinor	4
$V_\mu(x)$	Lorentz four-vector	8
$d(x)$	Complex scalar	2

Table 1.1: The components of the general superfield

are

$$\{\theta_A, \theta_B\} = 0, \quad \left\{\bar{\theta}_{\dot{A}}, \bar{\theta}_{\dot{B}}\right\} = 0, \quad \left\{\theta_A, \bar{\theta}_{\dot{B}}\right\} = 0. \tag{1.17}$$

The nature of Grassmann numbers is considerably different from that of ordinary numbers, and of relevance to us in the next section is the powers series expansion around these quantities. Following directly from the anti-commutation relations in Equation (1.17) we have that any powers series of a function $f(\theta_A)$ expanded around the Grassmann number is simply.

$$f(\theta_A) = a + b\theta_A. \tag{1.18}$$

We will not cover the calculus of Grassmann numbers here, but note that it follows from the anticommutation relations in Equation (1.17)

1.5.1 Superfields

In the previous section we noted that a superfield Φ is a function of the superspace coordinates, so that we have $\Phi = \Phi\left(x^\mu, \theta^A, \bar{\theta}_{\dot{A}}\right)$. We may expand such a superfield around the anti-commutating variables by using Equation (1.17) and Equation (1.18), so that generally we have

Superfield

$$\begin{aligned}\Phi(x, \theta, \bar{\theta}) =& f(x) + \theta^A \phi_A(x) + \bar{\theta}_{\dot{A}} \bar{\chi}^{\dot{A}}(x) + \theta\theta m(x) + \bar{\theta}\bar{\theta} n(x) \\ &+ \theta\sigma^\mu\bar{\theta} V_\mu(x) + \theta\theta\bar{\theta}_{\dot{A}}\bar{\lambda}^{\dot{A}} + \bar{\theta}\bar{\theta}\theta^A\psi_A(x) + \theta\theta\bar{\theta}\bar{\theta} d(x).\end{aligned} \tag{1.19}$$

where the component field content is summarized in Table 1.1 As it stands, the general superfield of Equation (1.19) is too general and has too many degrees of freedom for it to serve as a mathematical object that can represent physical particles. To inch closer to a suited superfield we will have to restrict the degrees of freedom of the superfield, and define the *scalar superfield* and *vector superfield*, both of which are irreducible representations of the super-Poincaré algebra.

1.5.2 Scalar Superfields

To restrict a superfield, and consequently construct the scalar superfield, we employ the concept of *covariant derivatives*, familiar from gauge theory as the quantity introduced

to make Lagrangians invariant under gauge transformations. We essentially follow along the same line as in ordinary gauge theory as applied in quantum field theory, but now we require that the Lagrangian should be invariant under supersymmetric transformation. The covariant derivatives that commute with supersymmetry transformation are

$$D_A \equiv \partial_A + i\left(\sigma^\mu \bar{\theta}\right)_A \partial_\mu, \qquad \bar{D}_{\dot{A}} \equiv -\partial_{\dot{A}} - i\left(\theta\sigma^\mu\right)_{\dot{A}} \partial_\mu. \tag{1.20}$$

We can obtain the scalar superfields by imposing the constraints

$$\bar{D}_{\dot{A}}\Phi(x,\theta,\bar{\theta}) = 0, \tag{1.21}$$

$$D_A\Phi^\dagger(x,\theta,\bar{\theta}) = 0, \tag{1.22}$$

where Equation (1.21) and Equation (1.22) correspond to the left-handed scalar superfield constraint and the right-handed scalar superfield constraint, respectively. Let us see how we can derive the left-handed scalar superfield by the use of Equation (1.21). We employ a change of variable $y^\mu = x^\mu + i\theta\sigma^\mu\bar{\theta}$ to the second definition in Equation (1.20) the second term cancels, and together with the constraint in Equation (1.21) we obtain

$$\bar{D}_{\dot{A}}\Phi(x,\theta,\bar{\theta}) = -\partial_{\dot{A}}\Phi(x,\theta,\bar{\theta}) = -\frac{\partial}{\partial\bar{\theta}_{\dot{A}}}\Phi(x,\theta,\bar{\theta}) = 0, \tag{1.23}$$

implying that the left-handed scalar superfield should be independent of $\bar{\theta}_{\dot{A}}$ in the new coordinates, in accordance with this independence we write the scalar field as

$$\Phi(y,\theta) = A(y) + \sqrt{2}\theta\psi(y) + \theta\theta F(y), \tag{1.24}$$

where $A(y)$ and $F(y)$ are complex scalars and $\psi(y)$ are left-handed Weyl spinor. For the right-handed scalar superfield we apply a similar coordinate change, $y^\mu \equiv x^\mu - i\theta\sigma^\mu\bar{\theta}$, together with condition Equation (1.22), to obtain the coordinate-transformed *right*-handed scalar field. Now, by undoing both the transformed left-handed and right-handed scalar fields we obtain the general form for the scalar fields, which we have listed in Appendix A.

Physical Interpretation In terms of physical particles, a scalar superfield cannot represent SM particles on its own, because a single Weyl spinor is only one of the two parts needed to characterize a Dirac fermion. Therefore we need *two* scalar superfields, of opposite handedness, in order to produce a fermion and its corresponding anti-particle. In addition to the four fermionic degrees of freedom of the particle-antiparticle pair, we also have four bosonic degrees of freedom in the form of two scalar particles and their anti-particles. These are the supersymmetric partners of the fermion, namely *sfermions*. Note, that the above discussion is subsequent to applying the equations of motion.

1.5.3 Vector Superfields

A *vector superfield* is defined by the constraint

$$\Phi^\dagger(x,\theta,\bar{\theta}) = \Phi(x,\theta,\bar{\theta}). \tag{1.25}$$

By comparing the expression for Φ and its hermitian conjugate $\Phi^\dagger$ Equation (1.19), we obtain by using the constraint in Equation (1.25) the general vector superfield

Vector Superfield

$$\begin{aligned}\Phi(x,\theta,\bar{\theta}) =& f(x) + \theta\varphi(x) + \bar{\theta}\bar{\varphi}(x) + \theta\theta m(x) + \bar{\theta}\bar{\theta}m^*(x) \\ &+ \theta\sigma^\mu\bar{\theta}V_\mu(x) + \theta\theta\bar{\theta}\bar{\lambda}(x) + \bar{\theta}\bar{\theta}\theta\lambda(x) + \theta\theta\bar{\theta}\bar{\theta}d(x).\end{aligned} \tag{1.26}$$

We have that $f(x)$ and $d(x)$ are real scalar fields, $\varphi(x)$ and $\lambda(x)$ are Weyl spinors, $m(x)$ is a complex scalar field and finally $V_\mu(x)$ is a real Lorentz 4-vector. The condition Equation (1.25) also applies for various combinations of superfields, so that the following are also vector superfields $V = \Phi^\dagger\Phi$, $\Phi+\Phi^\dagger$ and $i\left(\Phi - \Phi^\dagger\right)$, as they satisfy the condition.

Wess-Zumino Gauge The vector field in Equation (1.26) has a reducible number of degrees of freedom in terms of what may be expected from the representations of the super-Poincaré algebra. In the Standard Model, gauge conditions resulting from the requirement of invariance under a gauge transformation, reduces the number of degrees of freedom. Similarly we introduce a supersymmetric generalization of the gauge transformation, called a *supergauge transformation*. Particularly we implement the *Wess-Zumino gauge*, where the vector superfield is subjected to a gauge transformation by scalar superfields, the resulting vector superfield can be written as

$$\begin{aligned}V_{WZ}(x,\theta,\bar{\theta}) = &\left(\theta\sigma^\mu\bar{\theta}\right)\left[V_\mu(x) + i\partial_\mu\left(A(x) - A^*(x)\right)\right] \\ &+ \theta\theta\bar{\theta}\bar{\lambda}(x) + \bar{\theta}\bar{\theta}\theta\lambda(x) + \theta\theta\bar{\theta}\bar{\theta}d(x),\end{aligned} \tag{1.27}$$

where $A(x) + A^*(x) = -f(x)$ is a constraint, such that $[V_\mu(x) + i\partial_\mu\left(A(x) - A^*(x)\right)]$ provides in total three bosonic degrees of freedom. Additionally, the Weyl spinors $\bar{\lambda}(x)$ and $\lambda(x)$ provides four fermionic degrees of freedom, and lastly the real scalar field $d(x)$ provides one.

Physical Interpretation The gauge bosons of the SM are spin-1 vector fields, and the vector superfields of supersymmetry that we have just outlined, are the apropriate objects to use in order to reproduce the SM gauge bosons. After applying the equations motion, the vector superfield of the previous paragraph will have two less bosonic degree of freedom, and as such describe a mass-less SM vector boson with two bosonic degrees of freedom. Additionally, the field content will have two Weyl spinors, each of opposite handedness, with two fermionic degrees of freedom. In terms of particles, these constitute the two fermionic superpartners of the SM gauge boson. These types of particles are fittingly called *gauginos* and reside in the bosino family of sparticles.

1.6 Supersymmetric Lagrangian

To formulate a supersymmetric model suitable for describing the subatomic realm, we need to do as one often does in physics; Construct a Lagrangian $\mathcal{L}$. The Standard Model Lagrangian is invariant under Poincaré transformations, and in a like manner the supersymmetric Lagrangian we seek to construct should be invariant under supersymmetry transformations. Specifically, *the action*

$$S \equiv \int_R d^4x\mathcal{L}, \tag{1.28}$$

is invariant under supersymmetry transformations if it transforms as $\mathcal{L}$ as $\mathcal{L} \to \mathcal{L}' = \mathcal{L} + \partial^\mu f(x)$, where $f(x) \to 0$ on the surface of the integration. It can be shown that

the highest order components in θ and $\bar{\theta}$ of a superfield expansion will always transform by a total derivative as above, applying to both scalar and vector superfields and that integration over Grassmann numbers projects out the components. As a consequence we can redefine the integration to include the four Grassmann numbers, such that we get

$$S = \int d^4x \int d^4\theta \mathcal{L} = \int d^4x \int d^2\theta d^2\bar{\theta} \mathcal{L}, \tag{1.29}$$

where the lagrangian is a function of N number of superfields $\mathcal{L} = f(\Phi_i, \Phi_j, \Phi_i^\dagger, \Phi_j^\dagger)$[3], and is guaranteed to be supersymmetry invariant. Next, we write down a generic form of the Lagrangian of scalar superfields,

$$\mathcal{L} = \mathcal{L}_{\theta\theta\bar{\theta}\bar{\theta}} + \theta\theta\mathcal{L}_{\bar{\theta}\bar{\theta}} + \bar{\theta}\bar{\theta}\mathcal{L}_{\theta\theta}, \tag{1.30}$$

where the Grassmann indicies indicate the highest power of the Lagrangian components. By requiring that the Lagrangian should be renormalizable we obtain an additional restriction on the fields in the Lagrangian, specifically, this forbids powers of scalar fields higher than the third power. The precursor to what will be our general Lagrangian can then be written as

$$\mathcal{L} = \Phi_i^\dagger \Phi_i + \bar{\theta}\bar{\theta} W[\Phi] + \theta\theta W[\Phi^\dagger], \tag{1.31}$$

where the first term $\Phi_i^\dagger \Phi_i$ is denoted the kinetic terms, and W the *superpotential* which is written as

$$W[\Phi] = g_i \Phi_i + m_{ij} \Phi_i \Phi_j + \lambda_{ijk} \Phi_i \Phi_j \Phi_k \tag{1.32}$$

1.6.1 Supergauge Invariance

Just as the Standard Model Lagrangian is invariant under gauge transformations, the supersymmetry Lagrangian should be invariant under a supergauge transformation. We consider the transformation of a left-handed scalar superfield under a group G,

$$\Phi \to \Phi' = e^{-iq\Lambda^a t_a} \Phi. \tag{1.33}$$

We have that the group generators t_a satisfies $[t_a, t_b] = i f_{ab}^c t_c$ for the non-Abelian case, where f_{ab}^c is the *structure constant*, and for the Abelian case the commutator vanishes such that the generator of a transformation is only a multiplicative factor. Furthermore, q is the charge of the superfield Φ and Λ is the parameter of the gauge transformation, which can be shown to be a left-handed scalar superfield itself in order for the transformation of Φ to transform to a left-handed scalar field. For a right-handed scalar superfield the above discussion is equivalent, but considering $\Phi_i^\dagger$ instead.

For the constituent parts of the Lagrangian in Equation (1.31) to be invariant under the aforementioned transformation certain conditions must be met, and for the potential $W[\Phi]$ in Equation (1.32) this greatly restricts its form. We will not delve into the specifics of the effects of these conditions, but refer to [10]. For the kinetic term we must introduce a *gauge compensating vector superfield* V to appease the invariance requirement. The kinetic term must then be invariant under the gauge transformation such that

$$\Phi^\dagger e^{qV^a t_a} \Phi \to \Phi'^\dagger e^{qV'^a t_a} \Phi' = \Phi^\dagger e^{iq\Lambda^{a\dagger} t_a} e^{qV'^a t_a} e^{-iq\Lambda^a t_a} \Phi, \tag{1.34}$$

which is satisfied if, $e^{qV'^a t_a} = e^{-iq\Lambda^{a\dagger} t_a} e^{qV^a t_a} e^{iq\Lambda^a t_a}$.

[3] Summation is implicit

1.6.2 Field Strength

The supersymmetric Lagrangian we have been building as our model is almost ready to be put to use. The finishing piece is to introduce terms expressing the field strength, such as the field strength, expressed by the tensor $F^a_{\mu\nu}$, of quantum electrodynamics. We define the supersymmetric field strength by the spinor matrix scalar superfields,

$$\begin{aligned} \mathcal{W}_A &\equiv -\frac{1}{4}\bar{D}\bar{D}e^{-V^a t_a} D_A e^{V^a t_a} \\ \bar{\mathcal{W}}_{\dot{A}} &\equiv -\frac{1}{4} D D e^{-V^a t_a} \bar{D}_{\dot{A}} e^{V^a t_a}. \end{aligned} \tag{1.35}$$

It can be shown that $\mathcal{W}_A$ and $\bar{\mathcal{W}}_{\dot{A}}$) are left and right-handed superfield, respectively, and that $\mathrm{Tr}\left[\mathcal{W}^A\mathcal{W}_A\right]$ and $\mathrm{Tr}\left[\bar{\mathcal{W}}_{\dot{A}}\bar{\mathcal{W}}^{\dot{A}}\right]$ are the corresponding supergauge invariant terms, appearing as potential terms in the supersymmetry Lagrangian. Expanding W_A one can show that it contains the ordinary field strength tensor familiar from the Standard Model, and that the corresponding trace that we just saw includes terms with the inner product $F^a_{\mu\nu}F^{\mu\nu a}$.[4]

1.6.3 The General Supersymmetric Lagrangian

Finally, taking the Lagrangian of Equation (1.31) together with non-Abelian gauge groups of the previous paragraph we have our general Lagrangian

The Supersymmetric Lagrangian

$$\mathcal{L} = \Phi^\dagger e^{qV^a t_a}\Phi + \bar{\theta}\bar{\theta}W[\Phi] + \theta\theta W\left[\Phi^\dagger\right] + \frac{1}{2T(R)}\bar{\theta}\bar{\theta}\,\mathrm{Tr}\left[\mathcal{W}^A\mathcal{W}_A\right], \tag{1.36}$$

where $T(R)$ is the *Dynkin index*, a normalizing constant corresponding to the selected representation of the gauge group. For a Lagrangian suited for phenomenological analysis we necessitate soft breaking terms, as introduced in context of the Hierarchy problem Section 1.3. We will discuss these in the context of the MSSM model in section Section 1.7.3.

1.7 The Minimal Supersymmetry Standard Model

The Minimal Supersymmetric Standard Model(MSSM) corresponds to the $N = 1$, *supersymmetry algebra*, as described in Section 1.4. It is called the "Minimal" as it is the supersymmetric model with the least field content consistent with the Standard Model. We can construct the MSSM using the concepts regarding superfields of the previous sections.

[4] In the Standard Model it appears in the Lagrangian of quantum electrodynamics:

$$\mathcal{L}_{QED} = \bar{\psi}\left(i\gamma^\mu D_\mu - m\right)\psi - \frac{1}{4}F_{\mu\nu}F^{\mu\nu}$$

1.7.1 Field Content of the MSSM

In the following paragraphs we go through the field content of the MSSM. It is also summarized, along with the MSSM-4 model (soon to be introduced) in Table 1.2.

- **Leptons and Sleptons**
  In Section 1.5.2 we touched upon how to construct a Standard Model Dirac fermion particle-antiparticle pair; combine two Weyl spinors with opposite handedness from two different superfields. Included with this construction addition is also a four scalar particle-antiparticle pairs, which constitute the supersymmetric *slepton* partners. For the lepton field content we introduce the superfields

$$L_i = (\nu_i, l_i) \quad \text{and} \quad \bar{E}_i, \tag{1.37}$$

  where l_i and $\bar{E}_i$ are the superfields for the charged leptons and sleptons and ν_i for the left-handed neutrinos and sneutrinos. The indicies $i = 1, 2, 3$ denotes the particle generation. Superfields l_i and ν_i are placed in a $SU(2)_L$ doublet, and $\bar{E}_i$ are $SU(2)_L$ singlets. Right-handed neutrinos $\bar{N}_i$ are here assumed not to exist.

- **Quarks and Squarks**
  With quarks and squarks the case is similar to the (s)leptons.

$$Q_i = (u_i, d_i)\,, \quad \bar{U}_i \quad \text{and} \quad \bar{D}_i, \tag{1.38}$$

  where color indicies are omitted for brevity. Left-handed superfields are placed in the $SU(2)_L$ doublet Q, where the remaining transform as a singlet.

- **Gauge Bosons and Gauginos**
  In Section 1.5.3 we desribed how a vector superfield consists of a massless vector boson together with two opposite-handed Weyl spinors, so to no surpise we here employ vector superfields to reproduce the gauge bosons of the Standard Model. In the supersymmetric Lagrangian in Equation (1.36) we have the term $qV^a t_a$ in the exponent, which means we need one vector superfield V^a for every generator t_a of the three SM symmetry groups $SU(3)_C, SU(2)_L$ and $U(1)_Y$. So we fittingly denote superfields pertaining to the three symmetry groups as

$$C^a, \quad W^a \quad \text{and} \quad B^0, \tag{1.39}$$

  respectively. C^a give the 8 gluons, while the remaining W^a and B^0 give the $3+1$ weak interaction veector bosons before electroweak symmetry breaking. The fermionic superpartners of the SM gauge bosons, the gauginos, are denoted $\tilde{g}, \tilde{W}^a$ and $\tilde{B}^0$, and are called the gluino, wino, and the bino, respectively.

- **Higgs and Higgsinos**
  Finally we need Higgs superfields. The Higgs $SU(2)_L$ doublet of the SM cannot be used in the superpotential of the MSSM Lagrangian as it would mix left- and righ-handed superfields. In order for Higgs superfield to give mass to both up-type and down-type quarks we will need to define *two* Higgs doublets

$$H_u = \left(H_u^+, H_u^0\right), \quad H_d = \left(H_d^0, H_d^-\right), \tag{1.40}$$

  where the charge is denoted by the signs. In equation (1.40) we have what contitutes four left-handed scalar superfields, which yields four Weyl spinors, that

combine into the *Higgsinos*, and eight bosonic degree of freedom; of which three are Goldstone bosons that are "eaten" by Z and $W\pm$ through the Higgs mechanics and give these bosons masses, while the surviving five degrees of freedom embodies the mass eigenstates h^0, H^0, A^0 and $H^\pm$. The lightest Higgs is possibly the $m_h \sim 125.7$ discovered at the LHC.

1.7.2 The MSSM Lagrangian

By applying the concepts pertaining to superfields of the previous sections we can reproduce the Standard Model field content bu formulating The MSSM Langrangian. Its form is as follows:

The MSSM Lagrangian

$$\mathcal{L}_{\text{MSSM}} = \mathcal{L}_{\text{kin}} + \mathcal{L}_W + \mathcal{L}_V + \mathcal{L}_{\text{soft}}, \tag{1.41}$$

with kinetic terms, superpotential terms, supersymmetric field strength terms, and lastly soft-breaking-terms which we will describe in the next section. The consituent lagrangians of the kinetic, superpotential, and supersymmetric field strength contributions are listed in Appendix A.

1.7.3 MSSM Soft Supersymmetry Breaking

In Section 1.3 on the hierarchy problem we argued for the incluson of soft breaking terms in the supersymmetry Lagrangian, and here we study the case of soft breaking terms in the MSSM model. The soft breaking should be such that at high energies the symmetry is kept intact, while at low energies the symmetry is broken by the soft terms. Importantly, the included terms should preserve the cancellations in the loop contributions to scalar masses, such as in the Higgs mass. These requirements puts restrictions on the allowed soft terms to ones with couplings of mass dimension greater or equal to one. The soft-breaking terms of the MSSM are the following

Soft Supersymmetry Breaking Lagrangian in the MSSM

$$\begin{aligned}\mathcal{L}_{\text{soft}}^{\text{MSSM}} = &-\frac{1}{2}\left(M_3\tilde{g}g + M_2\widetilde{W}\widetilde{W} + M_1\widetilde{B}\widetilde{B} + \text{ c.c. }\right)\\ &-\left(\widetilde{\bar{u}}\mathbf{a_u}\widetilde{Q}H_u - \widetilde{\bar{d}}\mathbf{a_d}\widetilde{Q}H_d - \widetilde{\bar{e}}\mathbf{a_e}\widetilde{L}H_d + \text{c.c.}\right)\\ &-\widetilde{Q}^\dagger\mathbf{m}_{\mathbf{Q}}^2\widetilde{Q} - \widetilde{L}^\dagger\mathbf{m}_{\mathbf{L}}^2\widetilde{L} - \widetilde{\bar{u}}\mathbf{m}_{\bar{\mathbf{u}}}^2\widetilde{\bar{u}}^\dagger - \widetilde{\bar{d}}\mathbf{m}_{\bar{\mathbf{d}}}^2\widetilde{\bar{d}}^\dagger - \widetilde{\bar{e}}\mathbf{m}_{\bar{\mathbf{e}}}^2\widetilde{\bar{e}}^\dagger\\ &-m_{H_u}^2H_u^*H_u - m_{H_d}^2H_d^*H_d - (bH_uH_d + \text{ c.c. }),\end{aligned} \tag{1.42}$$

where $M_3, M_2,\ M_1$ are the gluino, wino, and bino mass terms. The second line in Equation (1.42) consists of the *trilinear couplings* to the scalar fields, where each $\mathbf{a}$ is a complex 3×3 matrix, in which the elements are in correspondence with the Yukawa couplings of the superpotential. The third line of Equation (1.42) contain squark and slepton mass terms, where each $\mathbf{m}^2$ is a hermitian 3×3 matrix (To avoid a notation-wise mess we keep the tildes on the subscripts of $\mathbf{m}^2$ terms implicit). Lastly, the fourth line consist of scalar mass terms, higgs mass terms $m_{H_u}^2$, $m_{H_d}^2$ and b, which are the ones that contribute to the Higgs potential. Note that the terms of any soft breaking

Lagrangian are not supersymmetric, as they correspond to terms with factors higher than third power $\theta\theta\bar{\theta}\bar{\theta}$ (See Section 1.6). Now that we have formulated the soft breaking terms we may note that the MSSM has 105 free parameters in terms of masses, phases and mixing angles, where 104 of them come from the soft terms and the remaining one from the superpotential [14].

1.7.4 R-Parity

In the supersymmetry Lagrangian of the MSSM the *bayron number* B and the *lepton number* L are no longer conserved quantities, and terms such as LH_u and $\bar{U}\bar{D}\bar{D}$ in the MSSM superpotential (Listed in Appendix A.3) violate baryon number and lepton number conservation, respectively. This is important, as these two conservation laws have been probed experimentially, whereby no processes violating L or B have been observed. Perceivably the violations may exist, but be very small. Take for instance the measured lower limit on the lifetime of protons: $\tau_{p\to e^+\pi^0} \leq 5.0 \cdot 10^{40}$ s, to account for the infrequency(or non-existence) of proton decay we would have to set the coupling λ of the violation terms to be extremely tiny. Alternatively, we can use a conservation law that let us avoid such couplings, namely R-parity

$$R = (-1)^{2s+3B+L}, \tag{1.43}$$

where s is the particle spin. All the SM particles will have $R = 1$, and all superpartners will have $R = -1$. If we define MSSM as conserving R-parity, terms $LH_u, LL\bar{E}, LQ\bar{D}$ and $\bar{U}\bar{D}\bar{D}$ are excluded from the superpotential.

Imposing R-parity conservation has some important implications: Sparticles are always produced in even numbers (usually in pairs). Sparticles must ultimately decay, but cannot decay beyond *the lighest supersymmetric particle* (LSP), which is completely stable and the final destination of any sparticle decay chain. This final resultant state of a decay chain consists of an odd number of sparticles (usually just one). The LSP is an important conceptual particle, because if it exists, is electrically neutral and thus interacts weakly with ordinary matter then it is a WIMP (As introduced in Section 1.1) and makes a suitable candidate for *dark matter*.

1.7.5 Radiative Electroweak Symmetry Breaking

In Section 1.3 we gave a rundown of the Higgs mechanism, whereby fermions and vector bosons are given mass by electroweak symmetry breaking (EWSB). Similarly in supersymmetry an EWSB mechanism is also neccessary, but here the equivalent mechanism is called *radiative*-EWSB (REWSB), where the name shall soon be clear. The MSSM potential for the scalar Higgs component fields is

$$\begin{aligned} V\left(H_u, H_d\right) =& |\mu|^2 \left(\left|H_u^0\right|^2 + \left|H_u^+\right|^2 + \left|H_d^0\right|^2 + \left|H_d^-\right|^2\right) \\ &+ \frac{1}{8}\left(g^2 + g'^2\right)\left(\left|H_u^0\right|^2 + \left|H_u^+\right|^2 - \left|H_d^0\right|^2 - \left|H_d^-\right|^2\right)^2 \\ &+ \frac{1}{2}g^2\left|H_u^+ H_d^{0*} + H_u^0 H_d^{-*}\right|^2 \\ &+ m_{H_u}^2\left(\left|H_u^0\right|^2 + \left|H_u^+\right|^2\right) + m_{H_d}^2\left(\left|H_d^0\right|^2 + \left|H_d^-\right|^2\right) \\ &+ \left[b\left(H_u^+ H_d^- - H_u^0 H_d^0\right) + c.c\right], \end{aligned} \tag{1.44}$$

where μ is the supersymmetric variant of the Higgs boson mass parameter encountered in the Standard Model. The terms g^2 and g'^2 are the gauge coupling terms, pertaining to

the $SU(2)_L$ and $U(1)_Y$ symmetry groups, respectively. The remaining three coefficients m_{H_u}, m_{H_d} and b are the parameters corresponding to terms from the soft breaking Lagrangian. From the four constituent complex scalar fields, H_u^+, H_u^0, H_d^0 and H_d^-, we have that the Higgs potential has eight degrees of freedom. In order for SM particles to acquire mass we will have to break the symmetry $SU(2)_L \times U(1)_Y \to U(1)_{\text{em}}$. For this to materialize the following conditions on the Higgs potential must be satisfied: 1. The minimum must retain the $U(1)$ symmetry; 2. It must have non-zero values at the minimum; 3. It must be bounded from below. For the SM case these conditions were satisfied by the inequalities $\lambda > 0$ and $\mu^2 < 0$. For the MSSM case, we begin by noting that the freedom to produce $SU(2)_L$ gauge transformations permits us to rotate away a VEV (See Section 1.3) for one weak isospin component, such that at the minimum of the potential, $V(H_u^0, H_d^0)$, we can set $H_u^+ = 0$. We have that at the minimum $\partial V/\partial H_u^+ = 0$ must be satisfied, which one can show that is the case if $H_d^- = 0$. This means that the charged components of the Higgs scalar are both zero $H_u^+ = H_d^- = 0$, thus they cannot attain VEVs and $U(1)_{\text{em}}$ is unbroken, fulfilling the first condition. Setting the aformentioned terms to zero in Equation (1.44) yields the potential

$$\begin{aligned} V\left(H_u^0, H_d^0\right) = &\left(|\mu|^2 + m_{H_u}^2\right)\left|H_u^0\right|^2 + \left(|\mu|^2 + m_{H_d}^2\right)\left|H_d^0\right|^2 \\ &+ \frac{1}{8}\left(g^2 + g'^2\right)\left(\left|H_u^0\right|^2 - \left|H_d^0\right|^2\right)^2 - \left(bH_u^0 H_d^0 + c.c.\right) \end{aligned} \tag{1.45}$$

It can be shown that symmetry breaking is ensured if the following inequalities on the potential are satisfied,

$$2b < 2|\mu|^2 + m_{H_u}^2 + m_{H_d}^2 \tag{1.46}$$

$$b^2 > \left(|\mu|^2 + m_{H_u}^2\right)\left(|\mu|^2 + m_{H_d}^2\right). \tag{1.47}$$

The first condition on the potential (non-zero values at minimum) is satisfied by Equation (1.46), while the second condition (bounded from below) is satisfied by Equation (1.47). If we assume $m_{H_u} = m_{H_d}$ at some high energy scale, such as the *grand unification scale* scale $\Lambda_{\text{GUT}} \sim 10^{16}$, then Equation (1.46) and Equation (1.47) cannot together be satisfied *at that scale*, but we need not expect them to follow this behaviour as they run down to lower energy scales. Of interest to us, is that the running of the masses m_{H_u} and m_{H_d} behave differently, in that they are dependent on the running of their *Yukawa couplings*, where m_{H_u} will run down faster than m_{H_d} as its dominated by the top quark Yukawa coupling y_t, while the latter by the bottom quark Yukawa coupling y_b, that is, $y_t \gg y_b$. So while Equation (1.46) and Equation (1.47) are not valid at some high energy scale; they emerge gradually towards the electroweak scale as a direct consequence of their different coupling evolution. It is this property that is referred to by the addition of *radiative* to EWSB, giving REWSB. The symmetry breaking resulting from REWSB should be able to reproduce the EWSB results and is therefore subject to the constraint

$$\left\langle H_u^0 \right\rangle + \left\langle H_d^0 \right\rangle = v_u^2 + v_d^2 \equiv v^2 = \frac{2m_Z^2}{g^2 + g'^2} \approx (246\text{GeV})^2, \tag{1.48}$$

where the rigth hand side is essentially the same equation as in Equation (1.5). The constraint on the VEVs allows us to formulate the expression in terms of one parameter

$$\tan\beta \equiv \frac{v_u}{v_d}, \quad 0 < \beta < \pi/2. \tag{1.49}$$

Furthermore, the condition that the minimum should satisfy

$$\frac{\partial V}{\partial H_u^0} = \frac{\partial V}{\partial H_d^0} = 0, \tag{1.50}$$

allows us to eliminate the parameters b and $|\mu|$ in Equation (1.45), but not the sign of μ, alternatively we may eliminate $m^2_{H_u}$ and $m^2_{H_d}$.

1.7.6 Electroweakinos

Of most interest to us in this thesis are the composite particles *neutralinos and charginos* (electroweakinos) pertaining to the electroweak (EW) interaction, hence we will not review matters relating to squarks and gluinos.

At the energy scales where the electroweak symmetry is broken we will have particle states that are a mix of multiple fermion fields. Since the electroweak symmetry is broken we do not consider $SU(2)_L \times U(1)_Y$ charges, only the $U(1)_{\text{em}}$ charges are of importance. The supersymmetric fermion fields, the bosinos, that are candidates for mixing are the Higgsinos $\tilde{H}_u^+, \tilde{H}_u^0, \tilde{H}_d^-, \tilde{H}_d^0$ and the electroweak gauginos: the bino $\tilde{B}^0$ and the three winos $\tilde{W}^0, \tilde{W}^\pm$.

In order for the fields to mix the only condition we impose is that the fields must have the same charge. We therefore have an eigenstate basis consisting of the neutral fields

$$\tilde{\psi}^0 = \left(\tilde{B}, \tilde{W}^0, \tilde{H}_d^0, \tilde{H}_u^0\right), \tag{1.51}$$

and one for the charged fields

$$\tilde{\psi}^\pm = \left(\tilde{W}^+, \tilde{H}_u^+, \tilde{W}^-, \tilde{H}_d^-\right). \tag{1.52}$$

In these bases we write down the electroweakino Lagrangian as

Electroweakino Lagrangian

$$\mathcal{L}_{\text{EWino}} = -\frac{1}{2}\left(\tilde{\psi}^0\right)^T M_N \tilde{\psi}^0 - \frac{1}{2}\left(\tilde{\psi}^\pm\right)^T M_{\tilde{C}} \tilde{\psi}^\pm + \text{c.c.}\ , \tag{1.53}$$

where the matrices $M_{\tilde{\chi}}$ and $M_{\tilde{C}}$ are the mass matrices. These are found from the general MSSM Langrangian; let us see how by first studying the neutral fields.

Photino and Zino The *photino* $\tilde{\gamma}$, and the *zino* $\tilde{Z}$, are the supersymmetric partners of the photon and the Z-boson, respectively, and emerge as mixings of just the neutral gauginos

$$\begin{aligned} \tilde{\gamma} &= N'_{11}\tilde{B}^0 + N'_{12}\tilde{W}^0 \\ \tilde{Z} &= N'_{21}\tilde{B}^0 + N'_{22}\tilde{W}^0, \end{aligned} \tag{1.54}$$

where N' is the 2×2 mixing matrix of the Standard Model gauge boson mixing. These particles do not include the neutral Higgsinos, so a more pertinent case is how all four neutral components mix, which brings us to the next category of composite particles.

Neutralinos When mixing all four neutral EW bosinos of Equation (1.52) we obtain a *neutralino*, comprised of the following mass eigenstates:

Neutralinos – Mass Eigenstates

$$\tilde{\chi}_i^0 = N_{i1}\tilde{B}^0 + N_{i2}\tilde{W}^0 + N_{i3}\tilde{H}_d^0 + N_{i4}\tilde{H}_u^0, \tag{1.55}$$

There are four neutralinos where $i = 1, 2, 3, 4$ denote which of these four neutralinos we are working with. The neutralinos are labeled in ascending order according to mass $m_{\tilde{\chi}_1^0} < m_{\tilde{\chi}_2^0} < m_{\tilde{\chi}_3^0} < m_{\tilde{\chi}_4^0}$, where the lightest neutralino $\tilde{\chi}_1^0$ is usually considered to be the LSP described in Section 1.7.4. The matrix N is a 4×4 unitary mixing matrix where the element j indiciate the size of the corresponding component field in the ith neutralino. We shall have more to say about N shortly.

The mass matrix for the neutralinos encountered in Equation (1.53) can be written, at tree level, as

$$M_{\tilde{\chi}^0} = \begin{bmatrix} M_1 & 0 & -c_\beta s_W m_Z & s_\beta s_W m_Z \\ 0 & M_2 & c_\beta c_W m_Z & -s_\beta c_W m_Z \\ -c_\beta s_W m_Z & c_\beta c_W m_Z & 0 & -\mu \\ s_\beta s_W m_Z & -s_\beta c_W m_Z & -\mu & 0 \end{bmatrix} \tag{1.56}$$

The entries M_1 and M_2 are the bino $\tilde{B}^0$ and neutral wino $\tilde{W}^0$ mass terms from the MSSM soft Lagrangian (See Section 1.7.3), and the entries $-\mu$ are from the higgsino mass terms $\mu H_u H_d$ in the superpotential, finally we have the abbreviations $s_\beta = \sin\beta, c_\beta = \cos\beta, s_W = \sin\theta_W$ and $c_W = \cos\theta_W$. These terms are from the Higgs-higgsino-gaugino couplings of the kinetic part of the Langrangian, for example the term $H_u^\dagger e^{\frac{1}{2}g\sigma W + g' B} H_u$. We have rewritten these four expressions in terms of the weak mixing angle and Z-mass as follows

$$\frac{1}{\sqrt{2}} g' v_{d/u} = c_\beta / s_\beta s_W m_Z, \quad \frac{1}{\sqrt{2}} g v_{d/u} = c_\beta / s_\beta c_W m_Z$$

To find the $\tilde{\chi}_i^0$ masses we diagonalize the mass matrix. The unitary mixing matrix N that we recently encountered is the matrix that diagonalizes the mass matrix M, that is

$$N M_{\tilde{\chi}} N^{-1} = D, \quad \text{where} \quad D = \left(m_{\tilde{\chi}_1^0}, m_{\tilde{\chi}_2^0}, m_{\tilde{\chi}_3^0}, m_{\tilde{\chi}_4^0} \right) \tag{1.57}$$

Charginos Similarly if we mix the charged fields we obtain the *charginos*, where the mass eigenstates relates to the gauge eigenstates of Equation (1.52) according to

Charginos – Mass Eigenstates

$$\tilde{\chi}_i^+ = V_{i1} \widetilde{W}^+ + V_{i2} \widetilde{H}_u^+, \quad \tilde{\chi}_i^- = U_{i1} \widetilde{W}^- + U_{i2} \widetilde{H}_d^- \tag{1.58}$$

where U and V are 2×2 unitary mixing matrices. The mass matrix of the charged part of the Lagrangian in Equation (1.53) is

$$M_{\tilde{\chi}_i^\pm} = \begin{bmatrix} 0 & X^T \\ X & 0 \end{bmatrix}, \quad X = \begin{bmatrix} M_2 & \sqrt{2} s_\beta m_W \\ \sqrt{2} c_\beta m_W & \mu \end{bmatrix}, \tag{1.59}$$

As before the terms come from the MSSM Langrangian, where the M_2 entries come from the soft terms of the charged wino $W^\pm$, the entries μ from the superpotential as before,

and the rest is from the kinetic terms, where we have used $gv_{u/d} = \sqrt{2}s_\beta/c_\beta m_W$.. Similar to the neutralinos, the mixing matrices U and V are chosen such that they diagonalize the mass matrix

$$UXV^{-1} = D, \tag{1.60}$$

and equivalently for the upper quadrant X^T.

1.8 Phenomenology

Name	Spin	P_R	Gauge eigenstates	Mass eigenstates
Higgs bosons	0	+1	$H_u^0\ H_d^0\ H_u^+\ H_d^-$	$h^0\ H^0\ A^0 H^\pm$
squarks	0	−1	$\tilde{u}_L\ \tilde{u}_R\ \tilde{d}_L\ \tilde{d}_R$	(same)
			$\tilde{t}_L\ \tilde{s}_R\ \tilde{c}_L\ \tilde{c}_R$	(same)
			$\tilde{t}_L\ \tilde{t}_R\ \tilde{b}_L\ \tilde{b}_R$	$\tilde{t}_1\ \tilde{t}_2\ \tilde{b}_1\ \tilde{b}_2$
sleptons	0	−1	$\tilde{e}_L\ \tilde{e}_R\ \tilde{\nu}_e$	
			$\tilde{\mu}_L\ \tilde{\mu}_R\ \tilde{\nu}_\mu$	(same)
			$\tilde{\tau}_L\ \tilde{\tau}_R\ \tilde{\nu}_\tau$	$\tilde{\tau}_1\ \tilde{\tau}_2\ \tilde{\nu}_\tau$
neutralinos	1/2	−1	$\tilde{B}^0\ \tilde{W}^0\ \tilde{H}_u^0\ \tilde{H}_d^0$	$\tilde{\chi}_1^0\ \tilde{\chi}_2^0\ \tilde{\chi}_3^0\ \tilde{\chi}_4^0$
charginos	1/2	−1	$\tilde{W}^\pm\ \tilde{H}_u^+\ \tilde{H}_d^-$	$\tilde{\chi}_1^\pm\ \tilde{\chi}_2^\pm$
gluinos	1/2	−1	$\tilde{g}$	(same)

Table 1.2: The MSSM particle content, where MSSM-4 is boxed in red. Standard Model particles are not included in the table.

We noted in Section 1.7.3 that the MSSM requires 105 parameters to be configured. Multiple models exist that reduce these number of parameters through experimental bounds and qualified guesses. In this thesis we shall use such a simplified MSSM model that shaves of a large number of free parameters. The model of choice is one that serves the purpose of studying the electroweakino sector of the MSSM, and is called the MSSM-4 as it has four free parameters at a given scale Q. This model is sufficient to our main objective of training a machine learning model to make cross-section predictions with uncertainty estimates, as metioned in the introduction to this thesis. We note that the specific sampling ranges of the free parameters in the MSSM-4, and a discussion of the data is covered in Chapter 6

1.8.1 MSSM-4

The electroweakinos of section Section 1.7.6 can all be described using four parameters

MSSM-4 – Free Parameters

$$M_1, M_2, \mu, \tan\beta, \tag{1.61}$$

which is apparent from the mass matrices of the neutralinos in Equation (1.56) and charginos in Equation (1.59). In this section we seek to construct an *electroweakino effective field theory* (MSSM-4) [15], valid for low energies, which is only dependent on the four electroweak parameters in Equation (1.61). In short terms we do this by fixing all other MSSM parameters such that we decouple all sparticles except the electroweakinos. By decoupling we mean that masses and parameters are set to values such that the corresponding interactions are impossible or negligble at the energies studied.

The model can be created by including the SM fermions, the gauge bosons and a SM-like Higgs boson, and fixing all SM couplings to experimental values, that is g, g' and the Yukawa SM couplings. In the following the specific values we use are not of importance, but rather that we force the model into the decoupling regime. We stated previously that we can parametrize the Higgs sector by just $\tan\beta$ and m_A. Setting m_A to a value well beyond the "decoupling limit" $m_A \gg m_Z$ will make the Higgs particles H^0, A^0, $H^\pm$ much heavier and almost degenerate and effectively decouples them from experiments. This leaves us with just $\tan\beta$ as a free parameter, specifically we set $m_A = 5$ TeV. Furthemore, for the following soft terms (See soft breaking Lagrangian Equation (1.42)) we fix the gluino mass parameter also to $M_3 = 5$ TeV, and all dimensionsful parameters to the SUSY scale $Q = M_{\text{SUSY}} = 3$ TeV. Finally we set all trilinear couplings to zero

$$\mathbf{a_u} = \mathbf{a_d} = \mathbf{a_e} = \mathbf{0}, \tag{1.62}$$

and all diagonal entries of the squared squark and slepton mass matrices to M^2_{SUSY} and the off-diagonal elements to zero

$$\mathbf{m^2_Q} = \mathbf{m^2_L} = \mathbf{m^2_{\bar{u}}} = \mathbf{m^2_{\bar{d}}} = \mathbf{m^2_{\bar{e}}} = \text{diag}\left(M^2_{\text{SUSY}}, M^2_{\text{SUSY}}, M^2_{\text{SUSY}}\right) \tag{1.63}$$

We assume that the lightest neutralino $\tilde{\chi}^0_1$ is the LSP, meaning that any parameters introducing lighter particles (gravitinos) are not part of the model. Additional this means we assume R-parity (See Section 1.7.4) to be conserved or broken weakly enough to make the lightest neutralino sufficiently stable.

1.8.2 Electroweakino Pair Production Cross Section

As we have already given away, the goal of this thesis is to use machine learning tools in order to aid in the production of electroweakino cross sections, for this to be possible we will need cross section data produced by perturbative methods. The cross sections are evaluated with QCD (the strong interaction) and SUSY-QCD, meaning that we account for interactions with quarks and gluons and supersymmetric squarks and gluinos.

The only electroweakino $\tilde{\chi}_{i,j}$ pair production process at leading order (LO) in pertubation theory that can be produced at hadron colliders is through the anniliation of quarks q and anti-quarks $\bar{q}'$

$$q\bar{q}' \to \tilde{\chi}_i\tilde{\chi}_j, \tag{1.64}$$

because gluons do not couple to electroweak bosons nor their supersymmetric partners. The production can occur through the s-channel by the propagation of a photon γ, neutral Z or charged $W^\pm$ weak gauge boson, or through the t- and u-channel with the propagation of squark $\tilde{q}$. These tree level diagrams are shown in Figure 1.2.

Let us look specifically at the process

$$q\bar{q}' \to \tilde{\chi}^+_i\tilde{\chi}^0_j \tag{1.65}$$

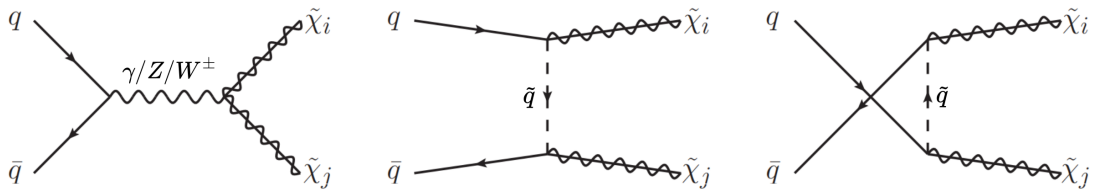


Figure 1.2: Tree-level Feynman diagrams for gaugino pair production, where the s-channel propagator can be $\gamma/Z/W^{\pm}$, and u/t-channel $\tilde{q}$. Figure derived from [16].

where the annihilation of a quark anti-quark leads to the pair production of a positive chargino, where $\tilde{\chi}_i^+$ is the sum over positive mass eigenstates in Equation (1.58), and a neutralino, where $\tilde{\chi}_j^0$ the sum over mass eigenstates in Equation (1.55). To formulate the transition matrix element $\mathcal{M}$, a *Fierz transformation* is first required for the u/t-channel squark-exchange amplitudes $\mathcal{M}_{u/t}$, afterwards the matrix element can be expressed in terms of bilinear charges $Q_{\alpha\beta}$. These charges are simply a neat way of writing the coefficients of the fermion (quark and gaugino) currents originating from the appropriate vertex rule and propagator. α and β denote left- and/or right-handed chiralities, giving four types of charges $Q_{LL}, Q_{RR}, Q_{RL}, Q_{LR}$. For the specific example in Equation (1.65) we have the bilinear charges

$$\begin{aligned} Q_{LL} &= \frac{1}{\sqrt{2}s_W^2}\left[\frac{N_{j2}^* V_{i1} - N_{j4}^* V_{i2}/\sqrt{2}}{s - M_W^2} + \frac{V_{i1}}{c_W}\frac{N_{j1}^*\left(e_{\tilde{q}} - I_{3\tilde{q}}\right)s_W + N_{j2}^* I_{3\tilde{q}} c_W}{u - m_{\tilde{q}}^2}\right] \\ Q_{LR} &= \frac{1}{\sqrt{2}s_W^2}\left[\frac{N_{j2} U_{i1}^* + N_{j3} U_{i2}^*/\sqrt{2}}{s - M_W^2} - \frac{U_{i1}^*}{c_W}\frac{N_{j1}\left(e_{\tilde{q}'} - I_{3\tilde{q}'}\right)s_W + N_{j2} I_{3\tilde{q}'} c_W}{t - m_{\tilde{q}'}^2}\right] \\ Q_{RL} &= Q_{RR} = 0, \end{aligned} \tag{1.66}$$

where the Mandelstam variables are $s = \left(p_q + p_{\bar{q}'}\right)^2, t = \left(p_q - p_{\tilde{\chi}_i}\right)^2$ and $u = \left(p_q - p_{\tilde{\chi}_j}\right)^2$., and electric charges and third isospin components of the squark propagator is expressed by $e_{\tilde{q}}$ and $I_{3\tilde{q}}$, respectively. The short forms of the weak mixing angle expressions and the mixing matrices are the same as in Section 1.7.6. Note, that quark and squark generational mixing have been neglected in Equation (1.66). For the specific example we consider here, the LO differential cross section *after* spin and colour averaging have been accounted for is

$$\begin{aligned} \frac{d\hat{\sigma}}{dt}\left[q\bar{q}' \to \tilde{\chi}_i^+ \tilde{\chi}_j^0\right] = \frac{\pi\alpha^2}{3s^2}\Big[&\left|Q_{LL}\right|^2 u_i u_j + \left|Q_{LR}\right|^2 t_i t_j \\ &+ 2\,\mathrm{Re}\left(Q_{LL}^* Q_{LR}\right) m_{\tilde{\chi}_i^+} m_{\tilde{\chi}_j^0} s\Big], \end{aligned} \tag{1.67}$$

where the abbreviations $t_{i,j} = t - m_{\tilde{\chi}_{i,j}}^2$ and $u_{i,j} = u - m_{\tilde{\chi}_{i,j}^2}^2$ are used. Integration can be performed on Equation (1.67) in order to calculate the leading order cross section $\sigma_{LO}\left(q\bar{q}' \to \tilde{\chi}_i^+ \tilde{\chi}_j^0\right)$. We refer to [16] for the generic form of the differential cross sections with all charges listed.

The Necessity of Next-to-Leading Order Cross Sections According to [17] the unphysical scale dependence of the LO cross sections of the gaugino processes of the previous section amounts to up to about 40%, meaning that the LO cross section is heavily dependent on the renormalization scale Q. Additional perturbative corrections such as *next-to-leading order* (NLO) cross sections are less dependent on the renormalization scale, as scale dependence is a decreasing function of the number of perturbative

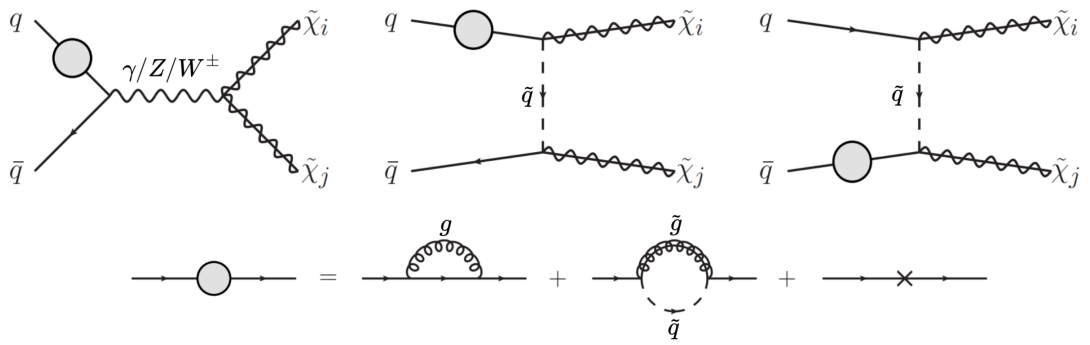


Figure 1.3: Examples of electroweakino production diagrams resulting from proton-proton collisions. Figure derived from [16].

corrections. Decreasing the degree of scale dependence is advantageous as it reduces the uncertainty of the cross section predictions. The reason for this is that the renormalization scale Q is unknown, and is set *a priori*, reducing the dependence on Q thus leads to more certain predictions. Figure Figure 1.5 shows an example of the difference in Q dependence between LO and NLO cross sections. Reducing the uncertainty and thus increasing the precision of predictions is important in that it tightens the exclusion limits of where supersymmetry can and cannot exist. [18] demonstrates a tightening of about 10% of the exclusion limits for gauginos with NLO corrections, relative to just the LO estimate. Having NLO corrections are thus preferrable over LO, that is, not taking computational expenses into account. At NLO of SUSY-QCD the particles of the gaugino pair produciton process that couples to gluons (q,$\bar{q}$,$\tilde{q}$) will recieve contributions from virtual one-loop diagrams, such as those shown in Figure 1.3. The depicted diagrams are just some of the self-energy diagrams, and in addition there are also vertex corrections diagrams, box diagrams, gluon and quark emission diagrams. We shall not go into the tricky details of higher order cross section calculations, but refer to [16].

To perform the NLO calculations we employ the numerical tool `Prospino 2.1` [3], which uses the Monte Carlo integration algorithm VEGAS [19] in order to approximate the cross sections of a given process. On that note, a central point to make is that NLO cross sections involve computationally costly integrals where the evaluation time per parameter configuration, amounting to a single final state, can take in the range of minutes on a modern CPU. This is far too slow for parameter scans where the necessary number of configuration are in the order of 10^4 and above. Before we consider how we may boost this production process by machine learning methods we should account for the fact that at hadron colliders they do not collide quark anti-quark in isolation,[5] but rather as part of proton-proton collisions.

1.8.3 Parton Distribution Function

Protons, the victims of choice for particle physicists at the LHC, are what we call *hadrons*. The straight-forward definition of hadrons is that they are composite particles of two or more quarks that are held together by the gluons, the mediators of the strong interaction. The quarks of this definition are called *valence quarks* and are the static bound state quarks that provide the quantum numbers used to classify a given hadron, and for the proton these are the three quarks *uud*. However, this picture alone underestimates the lively environment within a hadron like the proton. A proton is a dynamic

[5] Which would be a futile endevour as a consquence of *color confinement*

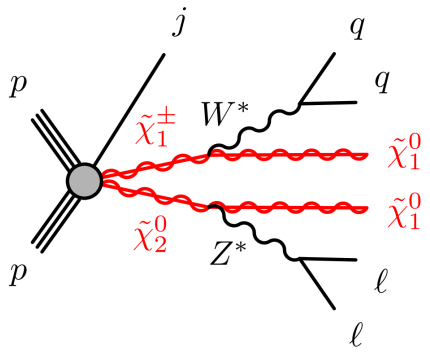


Figure 1.4: Example of a SUSY-scenario involving the production of electroweakinos $\tilde{\chi}_2^0\tilde{\chi}_1^{\pm}$ through proton-proton collision, with further decay to SM-particles and LSPs. W^* star denotes off-shell particles and j initial state radiation jet. Figure taken from [2]

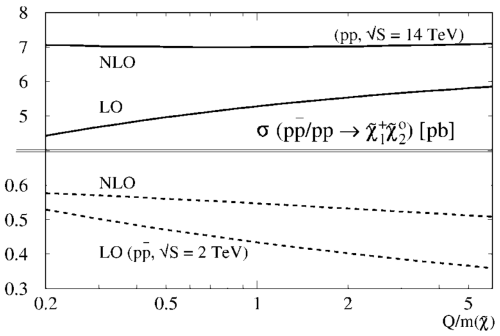


Figure 1.5: Figure showing the scale dependence Q of LO and NLO cross sections (Vertical axis: σ), for $\sqrt{s} = 14$ TeV and $\sqrt{s} = 2$ TeV. Note how the NLO plots are flatter than the LO plots. The supersymmetric model here is mSUGRA, not discussed in this thesis. Figure taken from [17]

object, so in addition to the valence quarks we have a sea of *virtual* quarks, anti-quarks and gluons which come about constantly as a consequence of the quarks interacting with one another through the exchange of gluons, which themself carry color charge and may further emit a gluon or split into a quark–anti-quark pair.

In the time before quarks and gluons were generally accepted as phenomena of nature, Feynman proposed that a proton is made up of point-like constituents, coined *partons*. This term has since stuck, and a parton refers to the constituent "parts" of the hadron, which we have established is more than just the valence quarks. The manner in which one describes these parts is in terms of the fraction of the total momentum. Specifically, one formulates what is called a *parton distribution function* (PDF), which is a distribution over parton momenta. For example, we could formulate an up-quark PDF for the proton $u^{\mathrm{P}}(x)$, such that $u^{\mathrm{P}}(x)\mathrm{d}x$ represents the number of up-quarks we have within the proton of momentum fraction between x and $x + dx$. PDFs as used in practice are extracted from measurements in scattering experiments, as QCD has a large coupling constant $\alpha_S \sim O(1)$ when the momentum transfer is small, which disallows perturbative methods for this purpose.[6] The integral over a PDF, $f(x, Q^2)$, can be interpreted as the total fraction of the momentum carried by the particle of interest. Such as for a proton this is has been measured to about $\int f_u dx \approx 0.36$ and $\int f_d dx \approx 0.18$ for the up and down quarks, respectively, at $2\mathrm{GeV}^2 < Q^2 < 30\mathrm{GeV}^2$ [21]. This fits well with the *uud*-structure, but interestingly enough this leaves more than 50% of the momentum to the gluon. It should be clear by now that the vibrant inner nature of the proton should be taken into account when calculating cross sections of processes involving protons.

The cross sections described in section Equation (1.67) are examples of what are called partonic cross sections. Now, we take the PDFs into account to gauge the partonic cross sections influence on the total cross section in consistency with experiment. For initial states q_i and q_j we evaluate the partonic cross section as a function of the momentum fractions of the initial states and the total squared center-of-mass energy

$$\hat{\sigma}(q_i\bar{q}_j \to \tilde{\chi}\tilde{\chi} \mid Q^2, s = x_1x_2S), \tag{1.68}$$

[6] However *lattice QCD* may provide an alternative route some day [20].

where Q is the renormalization scale and s the partonic center-of-mass energy. We now weigh the partonic cross section by the corresponding PDFs, and take the weighted average by integrating with respect to the momentum fractions x_1 and x_2, this then yields the *total hadronic cross section*:

Total Hadronic Cross Section

$$\sigma\left(S, Q^2\right) = \int dx_1 \int dx_2 \sum_{i,j} f_i\left(x_1, Q^2\right) f_j\left(x_2, Q^2\right) \hat{\sigma}\left(q_i \bar{q}_j \to \tilde{\chi}\tilde{\chi} \mid Q^2, s = x_1 x_2 S\right), \tag{1.69}$$

where the sum runs over all species of quarks and anti-quarks, u, d, $\bar{u}$, $\bar{d}, \ldots$. When generating the cross section data by using `Prospino 2.1` we will need an appropriate PDF, fortunately for us we do not need to seep through LHC data, as plenty of PDFs have already been constructed for a variety of purposes. In this thesis we shall employ the general-purpose `NLO PDF CTEQ6.6M` [22]. An example of a possible SUSY-scenario resulting from a proton-proton is shown in figure Figure 1.4. The cross section of the electroweakino production $\widetilde{\chi}_2^0 \widetilde{\chi}_1^{\pm}$ depicted is essentially what we would seek to produce with the total hadronic cross section. Now that we have covered the basics of supersymmetry and the model used to generate our cross section data, we will in the following chapters deep dive into the field of machine learning.

Part II

Machine Learning

Chapter 2

Basic Concepts in Machine Learning

Machine Learning is the study and application of automated methods where the principal aim is to uncover meaningful patterns in complex data-structures. Particularly, the objective is to tune the parameters of an adaptive model to recognize patterns suitable for inductive inference, allowing predictions and statistical propositions that extends beyond the seen data. We say that a model has the ability to *generalize* if we accomplish the objective of correctly predicting new examples of data.

2.1 Approaches

There are three approaches to Machine Learning: *Supervised learning*, *unsupervised learning*, and finally what is arguably a mixture of the previous two, *reinforcement Learning*. Through the remaining chapters of this thesis we shall reside within the realm of supervised learning. All three approaches have at the outset a set of input data in one form or another $\mathbf{X} = [\mathbf{x}_1^\intercal, \ldots, \mathbf{x}_n^\intercal]^\intercal$ called the *design matrix*, where $\mathbf{x}_i^\intercal = (x_{i1}, \ldots, x_{ip})$ is a p-dimensional *feature* vector, so the design matrix contains n rows of data examples and p columns of features. The features are the explanatory variables of the input data. In these expression we have denoted $i = 1, \ldots, n$ for data points, and $j = 1, \ldots, p$ for features. One may choose to discard or keep certain features depending on the relevance they have to the problem, this is called *feature selection*. The features that are kept and used to make predictions are the *predictor variables* of the model.

Supervised Learning In supervised learning the goal is to train a model in order to make a mapping from input $\mathbf{x}_i$ to output y_i, where y_i is called a *response variable*, *target* or *output*. We shall use these terms interchangeably. The output is a n-dimensional column vector $\mathbf{y}^\intercal = (y_i, \ldots, y_n)$. The complete data set consist of input–output pairs, $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{\mathbf{x}_i^\intercal, y_i\}_{i=1}^n$, so one data point of a data set corresponds to a pair of one $p \times 1$ vector $\mathbf{x}_i$ and a scalar value y_i. The data $\mathcal{D}$ is used to train the parameters of a model, and test its performance.

Remark on notation: We shall use $\mathbf{x}$ when we refer to what may be a single variable or a vector of p variables, while $\mathbf{x}_i$ is similar but with an emphasis on it being a single data point. We use the design matrix $\mathbf{X}$ when we refer to a matrix of data examples, so we

emphasise that we are working with a data set, meaning we have n samples of a column vector of $\mathbf{x}_i$. In this context, "variable" and "feature" are used interchangeably. We will use y to denote a target variable and similarly y_i as a single data point realization, and unlike in the case of the input variables, we use boldface $\mathbf{y}$ for a set of target data points, i.e. n samples of y. If the target contains samples of several target variables, i.e. multiple columns, we would write $\mathbf{Y}$, however we will not find use for the latter.

Unsupervised Learning In this case we are only given the inputs $\mathcal{D} = \mathbf{X}$, and there are no known targets $\mathbf{y}$. While supervised learning is a well defined problem, this category of learning is much less so. In unsupervised learning we have no consistent way of assessing, by use of an error metric (See Section 2.3), how far off a prediction is from some target value, as the latter is not defined. Examples of this type of learning is clustering algorithms, where one seeks to divide data into distinct groups.

Reinforcement Learning This type of learning is arguably somewhere between the previous two. In reinforcement learning the algorithm has no pre-set targets $\mathbf{y}$ to rely on, but must discover the optimal targets through a process of trial and error, where the learning algorithm is punished or rewarded accordingly. Its main goal is thus to maximize reward. A typical example is learning to play a computer game, for example like tetris. We could have used supervised learning, where we would let the algorithm learn how to play recorded games of the best human players. In this approach we would effectively be providing a pre-set target $\mathbf{y}$, but such an approach wouldn't allow the algorithm to become *much better* than the best human players. If we allow the algorithm to discover the targets by itself, within the rules of the game, we open for the possibility of it discovering game strategies never used by humans. The reward it would seek to maximize would be the game score. The search for targets implies that reinforcement learning is usually very computationally expensive. A famous example of reinforcement learning is Google DeepMind's AlphaGo [23]. In this case they used *self-play*, where the board game in Go is randomly initialized, then two competing algorithms play as black and white pieces, respectively. The algorithm then rewards the winning model, and updates the parameters accordingly.

Supervised Learning We continue our discussion on supervised learning, the most widely used approach in machine learning, and the one applied to solve the problems presented in this thesis. Supervised learning problems are mainly divided into two categories, *classification* and *regression*.

Classification Classification problems involves discrete categories (integers or labels) as targets $\mathbf{y}$, where for given input data $\mathbf{X}$ a categorical variable $\hat{\mathbf{y}}$ is predicted. This could be as trivial of an example as having an algorithm classify images of cats and dogs, to more involved problems such as classifying jets of particles in LHC data.

Regression Regression problems involve continuous target variables. So instead of trying to classify whether an image is of a cat or a dog, we might want to predict the weight of the animals based on the images. And instead of trying to classify jets of particles, we might want to predict cross sections, also a continuous measure. As predicting supersymmetric cross sections *is* indeed the purpose of this thesis we will stick to methods and concepts relevant for regression problems.

Whether we are dealing with classification or regression, supervised learning is essentially about using a method to learn a mapping function $\hat{f}(\mathbf{x})$ from input variable(s) $\mathbf{x}$ to output(s) y that approximates the true function $y = f(\mathbf{x})$.

2.2 Regression

Finding a suitable approximation $y = \hat{f}(\mathbf{x}) \approx f(\mathbf{x})$ for a true mapping could be as simple as finding the optimal coefficients w_0, w_1, in a 1-dimensional regression problem $y_i = w_0 + w_1 x_{i,1} = \mathbf{x}_i^\intercal \mathbf{w}$. In that regard, let us outline the basics of linear regression. Establishing a good intuition of linear regression is a good starting point for the methods in the coming chapters. The following discussion applies for higher dimensions as well, which is called multiple regression, i.e. $y_i = w_0 + w_1 x_{i,1} + \ldots + w_p x_{i,p} = \mathbf{x}_i^\intercal \mathbf{w}$.

Linear Regression We assume that the output y is a linear function of the input $\mathbf{x}$. We also assume the values of the true function do not fall perfectly on the line $y = \mathbf{x}^\intercal \mathbf{w}$, but that they are disturbed by an unknown error ϵ. This gives for some observation i,

$$y_i = \mathbf{x}_i^T \mathbf{w} + \epsilon_i. \tag{2.1}$$

The error ϵ is referred to as the *irreducible error* or *the noise*, and captures the natural variations in a real data set. Depending on the purpose of the analysis, when working with synthetic data it is often modelled by sampling from a normal distribution with zero mean and a choice of variance σ^2.

Let us say we are given a sample of datapoints originating from the true function, $(\mathbf{x}_0, y_0), \ldots, (\mathbf{x}_n, y_n)$. The goal of regression is to try to find a good estimate of the true parameters, that is, to find $\hat{\mathbf{w}} \approx \mathbf{w}$, by fitting the model to the observed points. The degree to which the predicted values $\hat{y}_i = \mathbf{x}_i^\intercal \hat{\mathbf{w}}$ differ from the observed values y_i is called the residuals $\hat{\epsilon}_i = y_i - \hat{y}_i$. So to obtain a good estimate we have to minimize the sum of the residuals. We do not wish to discern between predictions overestimating $\hat{y}_i > y_i$, or underestimating $\hat{y}_i < y_i$ the true values, so we minimize the square residual instead $\hat{\epsilon}_i^2 = (y_i - \hat{y}_i)^2$. The minimization objective is therefore the residual sum of squares(RSS),

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \left\{ (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}})^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) \right\}. \tag{2.2}$$

The normalized version of this expression, is the mean squared error(MSE)(See Equation (2.7)). In terms of minimization the RSS and MSE are equivalent. Note that the word *error* in MSE does not refer to the noise term ϵ, but the *observed* residuals $\hat{\epsilon}$.

Inserting our model for $\hat{y}$, the quadratic minimization problem is

$$\begin{aligned} \frac{\partial \text{RSS}}{\partial \hat{\mathbf{w}}} &= 0 \\ \mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) &= 0 \end{aligned} \tag{2.3}$$

Solving for $\hat{\mathbf{w}}$, and given that the moment matrix $\mathbf{X}^T\mathbf{X}$ can be effectively inverted(well-conditioned), we derive the *ordinary least squares* estimate

$$\hat{\mathbf{w}}_{\text{OLS}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}. \tag{2.4}$$

We now have an estimate $\hat{\mathbf{y}} = \hat{\mathbf{w}}\mathbf{X}$ of the true function y. If the underlying function was indeed perfectly linear, and we added datapoints ad infinitum, we would expect

the residuals to approach the irreducible error $\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n}(\hat{\epsilon}_i - \epsilon_i) = 0$. This is the same statistical discussion whereby the sample mean $\bar{x}$ approaches the population mean μ. If we were working with a normal distributed noise with $\mu = 0$, then $\mathbb{E}(\hat{\epsilon}) = 0$.

Higher Order Regression In the ordinary least squares discussion we assumed linear regression, but the result in Equation (2.4) also applies to various forms of non-linear regression. For example polynomial regression, where we initialize polynomial *basis-functions*. Take for instance a second degree polynomial with two *basis features* would take the form $\hat{\mathbf{y}} = w_0 + w_1\mathbf{x}_{i,1} + w_2\mathbf{x}_{i,2} + w_3\mathbf{x}_{i,1}^2 + w_4\mathbf{x}_{i,2}^2 + w_5\mathbf{x}_{i,1}\mathbf{x}_{i,2} = \mathbf{x}^\intercal\mathbf{w}$, giving in total six *derived features*. In $\mathbf{w}$ from the second degree polynomial example we see that we rapidly increase the number of parameters and derived features with increasing complexity of the model. If we from the get go have many basis features and prefer a highly complex model, then we run the risk of having more parameters than we have data points $n < p$, which will make $\mathbf{X}^T\mathbf{X}$ ill-conditioned. This problem can be mitigated by introducing a regularization parameter λ, which we will have more to say about in Section 2.5.

Regression in supervised learning It should be noted, that the method producing the estimate Equation (2.4) is not a supervised learning method. The ordinary least squares method mainly involves calculation of one set of derivatives used to analytically find the minimum, and thus *one* update of the estimated parameters $\hat{\mathbf{w}}$. For many problems, simple analytical solutions like the one in Equation (2.4) are either not available, intractable, or do not provide the predictive power we would hope to attain. So, if to obtain an optimal estimate $\hat{\mathbf{w}}$ we instead employ an optimization algorithm consisting of a sequence of updates which iteratively adapt the parameters

$$\begin{aligned} \mathbf{w}_1 &= \mathbf{w}_0 + \Delta\mathbf{w}_0 \\ &\vdots \\ \mathbf{w}_n^* &= \mathbf{w}_{n-1} + \Delta\mathbf{w}_{n-1}, \end{aligned} \tag{2.5}$$

to fit the data $\mathcal{D}$, we say that we are using an algorithmic model. Because of the adaptive nature where the parameters change and the model gradually attains more information about the optimal mapping, we say that the model undergoes a process of *learning*, and as its directed to fit known targets we deem it *supervised learning*. With large datasets and many algorithmic steps such algorithms could only be feasibly executed by the use of computers, hence *Machine Learning*. An Artifical Neural Network is an example of such an algorithmic structure and is presented in the next chapter.

2.3 Error Metrics

In order to assess a regression model's performance we need metrics that can gauge the distance between a model prediction $\hat{y}$ and the true value y, that is, the prediction accuracy. We will mainly use two metrics for this purpose, the first one of the two have already been mentioned:

Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\,. \tag{2.6}$$

The MSE is used to compare different models using the same data. The model with the lowest MSE is generally the favourable one. The MSE is a reasonable metric to monitor as it has a clear-cut distance related intuition. Often the minimization object of the optimization procedure *is* the MSE itself or contains the MSE, so the metric ties directly to the task the model has been given. That said, the MSE is *scale-dependent*, meaning it makes no sense to compare MSE values from data sets of different scales. Data with lengths in the order of light years will presumably give larger MSE values than data in the order of millimeters, but that does not mean the model studying light years is worse than the one studying millimeters. Another point to make about the MSE is that we often will not know whether a certain MSE constitutes a high absolute prediction accuracy or not, meaning that we can only compare the MSE value to other models on the same data set and assess the relative prediction accuracy. That said, this is to an extent, different if we know the noise ϵ, the term we encountered in Section 2.2. In that case the noise, or the irreducible error, serves as a lower bound on the error, $\epsilon^2 < \text{MSE}$.

In the results of this thesis we will mostly employ the root mean squared error (RMSE)

Root Mean Squared Error

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{2.7}$$

The discussion for the MSE naturally applies here as well. The main reason we want to use the RMSE is simply to get the metric in the same scale as the data, given small residuals, so that it is easier to interpret.

The coefficient of determination, or the R^2 score, is a metric that compares the residual sum of squares (RSS) in Equation (2.3) by the total sum of squares (TSS) as the fraction of the two:

R^2−Score

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}. \tag{2.8}$$

We could also interpret it as being the MSE divided by the variance of the targets

$$R^2 = 1 - \frac{\text{MSE}}{\text{Var}(y)} = \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}, \tag{2.9}$$

as $\frac{1}{n}$ cancels. In the the 1-dimensional case the variance, $\text{Var}(y)$, can be interpreted as the mean squared distances to a horizontal line $y = \bar{y}$ representing the average. The MSE on the other hand is the mean squared distances to your model line $\hat{y}$. So we can understand this metric as measuring to what degree the squared distance from the datapoints to the average is smaller/larger than the squared distances to your model:

- R^2 scores close to zero is an indication of poor model performance. This can be interpreted as the accuracy of the horizontal average line being similar to the accuracy of your model line, $\text{MSE} \approx \text{Var}(y)$ and $\text{R}^2 = 1 - \frac{\text{MSE}}{\text{Var}(y)} \approx 0$.

- R^2 scores close to one is an indication of good model performance, so that the regression model is a good fit to the observed data, MSE $\ll \text{Var}(y)$ then $R^2 \approx 1$.

The R^2 score has some clear limitations, and should not be used without context. For example, say that the MSE is only slightly smaller than the variance, e.g. MSE $= 0.99 \cdot \text{Var}(y)$, which would give $R^2 = 0.01$. Can we be certain that this is a bad model? Not really, the model might still be perfectly correct. Even though there is seemingly a small difference in performance between the horizontal average line and the model line, "small" is completely relative to the problem in question, and the model might still be excellent depending on the problem. The same type of limitation applies for a high R^2 score, it does not necessarily mean you have a good model, so we use it with caution and always rely on multiple metrics.

2.4 Train, test and Validation Sets

If we used all the input–output pairs $\mathcal{D} = (\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ at our disposal to train a model then would not have any data left for model testing. Of course, we could test how well a model predicts on data it has already "seen", which has its purpose, but that would not tell us how well the model predicts "unseen" data. In statistics jargon these types of prediction are called *in-sample* forecast for predictions on "seen", and *out-of-sample* forecast for predictions on "unseen" -data.

2.4.1 Train and Test sets

To assess a model's ability to generalize and predict new and "unseen" data, one can extract out a part of the data by random sampling. This part is called the *test data*, and is left "untouched" during training of the model. The remaining part of the data which is used to train the model is called the *training data*.

Let us illustrate this idea of splitting the data with an example of predicting rainfall: Say, you want to use a data set $\mathbf{X}$ of n rows of data, where each row of data, called an *example*, contains a set of p features. These features could be measurements of windspeed, atmospheric pressure, etc.

Every example in $\mathbf{X}$ has a corresponding numerical outcome in $\mathbf{y}$, which is precipitation (rain, snow, hail etc) on the next day after the day the example was sampled. Together the measurements and outcomes make up the data $\mathcal{D} = (\mathbf{X}, \mathbf{y})$. Now, you want to use this data to predict tomorrow's weather, y_{unseen}. Naturally, you wouldn't have access to tomorrow's weather, so if you wanted to evaluate how well your model predicts "unseen" weather *before* tomorrow, you could treat some examples of your data *as if* it was unseen. You would then split $\mathcal{D}$ along the rows, here with the design matrix X,

$$
\begin{array}{c}
1 \\ 2 \\ \\ n
\end{array}
\begin{bmatrix}
x_{11} & x_{12} & \cdots & x_{1p} \\
x_{21} & x_{22} & \cdots & x_{2p} \\
\vdots & \vdots & \ddots & \vdots \\
x_{n1} & x_{n2} & \cdots & x_{np}
\end{bmatrix}
\begin{array}{l}
\left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} X_{\text{train}} \\
\left.\vphantom{\begin{matrix}a\\a\end{matrix}}\right\} X_{\text{test}}
\end{array}
$$

into $\mathcal{D}_{\text{train}} = (\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$ and $\mathcal{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$. Usually, one commits most of the data examples for the training set $\mathcal{D}_{\text{train}}$. A common choice is 80% training data, and 20% test data. Apart from the training portion being the largest, these percentages are arbitrarily chosen, thus other ratios might be more suited depending on the problem.

Now we can fit a model to the data $\mathcal{D}_{\text{train}}$, yielding as set of parameters $\hat{\mathbf{w}}$. For illustration we could use the ordinary linear squares model and its estimate Equation (2.4), the same ideas apply apply for supervised learning models. The OLS estimate is then $\hat{\mathbf{w}} = (\mathbf{X}_{\text{train}}^{\intercal}\mathbf{X}_{\text{train}})^{-1}\mathbf{X}_{\text{train}}^{\intercal}\mathbf{y}_{\text{train}}$, and the model has been fitted to the training data.

We may now use these model parameters to predict the out-of-sample targets y_{test}, and the in-sample targets y_{train},

$$
\begin{aligned}
\hat{y} &= \mathbf{w}^{*}\mathbf{X}_{\text{test}} \\
\tilde{y} &= \mathbf{w}^{*}\mathbf{X}_{\text{train}}
\end{aligned}
\tag{2.10}
$$

These predictions allow us to calculate the *test error*, also called the *generalization error*:

Generalization Error

$$
\text{MSE}_{\text{test}} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i^{\text{test}} - \hat{y}_i\right)^2 \tag{2.11}
$$

and the *train error*, also called the *empirical error*:

Empirical Error

$$
\text{MSE}_{\text{train}} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i^{\text{train}} - \tilde{y}_i\right)^2 \tag{2.12}
$$

Comparing the test error and the train error is central to Machine Learning analysis.

2.4.2 Hyperparameters

Hyperparameters are quantities that have to be set manually before the training session. The prefix hyper- is added to differentiate hyperparameters from the parameters optimized during training, which up until now have been $\mathbf{w}$. There can be a long range of various hyperparameters in a model, but the most common ones in deep learning are, some form of regularization parameter λ, and the learning rate η, the former we

shall cover in chapter Chapter 4 and the latter in chapter Chapter 3. Optimal hyperparameters usually have to be searched for by a brute-force method. This involves training several models using different hyperparameters, e.g. $[\hat{\mathbf{w}}(\lambda_0), \hat{\mathbf{w}}(\lambda_1) \ldots, \hat{\mathbf{w}}(\lambda_n)]$, and checking which model gives the lowest generalization error.

2.4.3 Validation set and Data Leakage

If model selection is done solely with the train-test setup we run the risk of introducing a bias to the model. Say we obtain the model $\hat{\mathbf{w}}(\lambda_0)$ using $\mathbf{X}_{\text{test}}$, and then calculate $\text{MSE}(\hat{\mathbf{y}}, \mathbf{y}_{\text{test}})$ to check model performance. Now that we have peeked at the test values *we can no longer change our model*. If we choose to discard $\hat{\mathbf{w}}(\lambda_0)$ and try $\hat{\mathbf{w}}(\lambda_1)$ instead then we have essentially cheated. Looking again at the previous weather example: If we peek at the data that is emulating tomorrow's unseen weather multiple times, then we can not in earnest pretend that the test data emulates *unseen weather*. Committing such a "crime" in Machine Learning is called *data leakage*, which is the procedural error of using information during the training process that one could not expect to possess when predicting actual out-of-sample outcome, such as information about the parameter λ only disclosed by the test set. The consequence is that the error metrics may overestimate a model's ability to generalize and thus overestimate its usefulness. By overestimating we mean that the model gives on average a lower error than what we could expect of the out-of-sample prediction error.

The solution is to introduce a *validation set*, which we do not pretend to be "unseen" data, but nonetheless is data that the model parameters $\mathbf{w}$ have not been directly fitted on. The resulting validation error $\text{MSE}(\hat{\mathbf{y}}, \mathbf{y}_{val})$ might overestimate after hyperparameter tuning. However, as the test set should be kept untouched and only tested after all model adjustments have been done, the final and appropriate performance evaluation of the model is expressed by $\text{MSE}(\hat{\mathbf{y}}, \mathbf{y}_{\text{test}})$ which thus represents the model's ability to predict "unseen" targets. Practically, we begin a machine learning task that involves model selection by splitting the data into *three* parts. Usually the validation and test set are equally partitioned, so for example, train set: 70%, validation set: 15% and test set: 15%. Again, these partitioning schemes are somewhat arbitrary, but the rule of thumb is usually in the range $70 - 80\%$ for the training data-set[24].

Model selection consists of more than just hyperparameter tuning, thus there are additional reasons to keep a validation set. For example feature selection procedures, such as sequentially adding or removing features from the dataset while monitoring the validation error. Another reason to include a validation set relates to the topic of *early stopping*, this is covered in Section 2.5.

2.5 Overfitting, Underfitting and Early Stopping

2.5.1 Overfitting and Underfitting

Imagine that we have trained a model on a set of data points $\mathcal{D}_{\text{train}} = (\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$, and that we sought to predict the out-of-sample data points $\mathcal{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$. In the process of training and testing the model, we would possibly encounter two types of problems: Either the model was too well fit to $\mathbf{y}_{\text{train}}$ to predict $\mathbf{y}_{\text{test}}$ (overfitting), or the model was too simple and lacked the complexity to capture the variation of $\mathbf{y}_{\text{train}}$ (underfitting). This is best illustrated with a figure: In Figure 2.1 polynomial

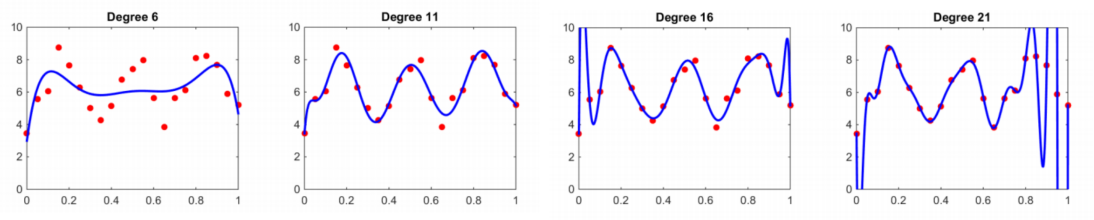


Figure 2.1: Example illustrating underfitting and overfitting with polynomial regression with increasing complexity. Figure taken from [25].

regression has been performed with varying degrees of complexity. If we were to use this regression to predict new data points we would want to strike a balance between the extremes of overfitting and underfitting. Of the two extremes it is overfitting that is usually a problem in machine learning. However, in Section 7.1.1 we shall see that underfitting is a central issue in the Bayesian model that will be presented. One way to mitigate overfitting is to introduce *regularization*, which we have already touched upon in Section 2.2. This is done by using the method of Lagrange multipliers and adding a constraint to our minimization objective, so with the Equation (2.2) we would simply add $\frac{\lambda}{2}\|\mathbf{w}\|^2$ to the expression. The OLS estimate will then take the form of the *Ridge* estimator

$$\hat{\mathbf{w}}_{\text{Ridge}} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}, \tag{2.13}$$

and this type of regression is naturally called *Ridge regression*. We shall see in Equation (4.24) that this method can be tied to a probabilistic formulation.

2.5.2 Early Stopping

In Equation (2.5) we showed a general example of what supervised learning entails. Such model training is adaptive and usually fits better to the training data the more iterations it gets to run. But as we have just elaborated on, if a model is too well fit to the data, overfitting may occur. Hence we would not want to run a model over too many iterations, that is, we would want to execute *early stopping* to halt the algorithm when a certain condition has been met. The manner in which to implement this technique is to include a validation data set for this purpose alone, which in the context of *early stopping* we shall denote $\mathcal{D}_{\text{es}}$. The technique then involves calculating a *validation error*, such as by the MSE, at regular intervals, and as the model is not being trained on $\mathcal{D}_{\text{es}}$ the error obtained would be an estimate of the generalization error. How often one chooses to calculate the validation error depends on the problem and on computational costs. Using the MSE_{es}, a typical case in regression would be that for the first number of iterations the MSE_{es} would decrease as the model heads out of underfitting territory. However, at some point the model begins to overfit on the training data and its ability to generalize steadily decreases, and from this point on the MSE_{es} would start increasing. Hence the condition of early stopping is simply: Stop the training when the MSE_{es} starts increasing. It is common to set a *patience* parameter, as we have done in our experiments, to avoid stopping the training immediately, in case the increasing MSE_{es} was just a random fluctuations. So we could wait, say 50 iterations, and if the MSE_{es} still has not achieved new lows we stop the algorithm.

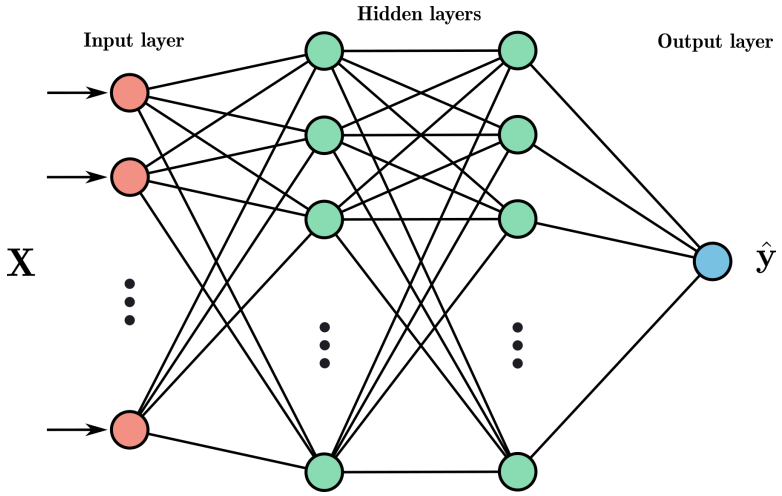


Figure 3.1: Fully connected (dense) neural network with two hidden layers and input layer with undefined number of nodes. One node in the output layer.

Chapter 3

Dense Neural Network

Before we outline the details of a Bayesian neural network (BNN), the main focus of this thesis, we should cover the architecture of its ancestor, the *multilayered perceptron* (MLP). The inner workings and mechanisms of a MLP is the backbone of a Bayesian neural network, and we shall refer back to this section at several points later on. Let us clear up some jargon first, the MLP is a special case of a *feed-forward neural network* (FFNN) where every node is *fully connected* to the next layer of nodes[1]. Exactly what we mean mathematically by a node, or a neuron, will be made clear shortly, but for now we refer to the illustration in Figure 3.1. The category FFNN, covers network designs where information flows in the forward direction from input to output. This is in contrast to other types of neural network architectures, such as *recurrent neural networks* where information may cycle back to previous nodes. We shall use the word "deterministic" when we refer to a non-Bayesian neural network structure.

[1]The names node, neuron and perceptron all refer to the same thing in the context of neural networks.

3.1 Feed-Forward propagation

We begin by again looking at the linear regression model, but unlike the ordinary linear regression from Section 2.2, we here formulate a more general expression

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1\phi_1(x) + \ldots + w_{p-1}\phi_{p-1}(x) \tag{3.1}$$

which can be written more compactly as

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{p-1} w_j\phi_j(\mathbf{x}) = \mathbf{w}^\intercal\boldsymbol{\phi}(\mathbf{x}). \tag{3.2}$$

As before the w_j are the parameters we wish to adjust in order for the regression to fit some data. In a neural network we call these parameters *weights*. The first parameter w_0 has its own name, namely the *bias* (not to be confused with *statistical bias)*, often denoted b in the context of a neural network. For a simple $y = w_0 + w_1x$ model, as in Section 2.2, the bias parameter is familiar as it is simply the intercept of the y-axis. We also have the structure $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{p-1})^\intercal$, which is known as the set of *basis functions*. In the previous chapter the basis functions were of first order, for example with *multiple inputs* we would have $\boldsymbol{\phi} = (1, x_1, \ldots, x_{p-1})^\intercal$, and thus linear with respect to the inputs. We also spoke briefly about choosing non-linear basis functions such as polynomial basis functions, for example with *one input* to an arbitrary polynomial degree we would have $\boldsymbol{\phi} = \left(1, x^2, x^3, \ldots, x^{p-1}\right)^\intercal$. So in the polynomial case the resulting linear regression model would be a linear combination of fixed non-linear basis functions. However, it is still a *linear* model, as its linearity refers to it being linear with respect to the parameters $\mathbf{w}$. Note, that when the basis functions are initialized for a data set of inputs $\mathbf{X} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{p-1}\}$ we had the *design matrix*, with $n \times p$ number of rows and columns. However, to simplify, in the following discussion we shall assume we only have one data point with p features (input nodes), $\mathbf{x} = \{x_0, \ldots, x_{p-1}\}$.

Continuing, we may wrap a function around our linear combination,

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^{p-1} w_j\phi_j(\mathbf{x})\right). \tag{3.3}$$

This function is called an *activation function*, which transform the input and then determines the output. For linear regression the activation function is just the identity, $f(x) = x$, and in fact, a neural network regression with no hidden layers and with the identity activation function *is* linear regression. We shall soon see, that using different activation functions allow us to formulate a non-linear model. Note, that the expression Equation (3.3) is the mathematical description of a single neuron.

A neural network parts way with the linear regression model in that the basis functions $\phi_j(\mathbf{x})$ are *no longer fixed*, but are themselves a non-linear function that depends on a linear combination of inputs that can be tuned during the training process. The basis functions in a given layer $\phi_j(\mathbf{x})$ is thus the output of the preceding layer in the network. The result is a series of transformations called a feed-forward propagation. Using non-linear activation functions in Equation (3.3) is central to this structure, as if we only use the identity the repeated linear transformations could be re-written into a single transformation [26].

Feed-forward with two hidden layers We will now show an example with two hidden layers. At the outset we have the *input layer*, which we may call $l = 0$, where the data features x_j are fed into to the network. Next, we step into the network by creating a linear combination of the network inputs to a given node i, denoted by z_i:

$$z_i^{(1)} = \sum_{j=1}^{n^{(0)}} w_{ij}^{(1)} x_j + b_i^{(1)}. \tag{3.4}$$

The superscripts signifies that this is the input to layer $l = 1$. The subscript i tells us which node in the current layer ($l = 1$) the linear combination z_i corresponds to, and the subscript j tells us which feature-node of the previous layer ($l = 0$) the data x_j is received from. The subscripts in the weight w_{ij} follow the same logic and inform us of the starting-point j and end-point i of the connection the weight applies to. We will denote the number of terms in a sum by $n^{(l)}$ with the superscript l denoting the relevant layer. For a better overview we summarize the notation in Table 3.1.

$n^{(l)}$	Number of nodes in layer.	$w_{ij}^{(l)}$	Weight from node j in $l-1$ to node i in l.
x_j	Input feature j.	$b_i^{(l)}$	Bias to node i in layer l.
$z_i^{(l)}$	Linear combination in node i in layer l.	$a_i^{(l)}$	Activation in node i in layer l.

Table 3.1: Notation and index explanation for a dense neural network

Equation (3.4) is a weighted sum over all the possible inputs $n^{(0)}$ for a given node i, meaning that we open for the possibility of giving different strengths, i.e. w_{ij}, to the various input data x_j. Just as with Equation (3.3), the linear combination is transformed by a nonlinear *activation function* $f(.)$, which we denote $a_i^{(l)}$ and call the *activation*. The output of the first hidden layer is,

Output of the first hidden layer

$$a_i^{(1)} = f(z_i^{(1)}) = f\left(\sum_{j=1}^{n^{(0)}} w_{ij}^{(1)} x_j + b_i^{(1)}\right). \tag{3.5}$$

Figure 3.2 shows a schematic illustration of the construction so far, specifically the activation for node $i = 1$ in the first hidden layer ($l = 1$), as an example.

The output of a node i in the next hidden layer, $l = 2$, will then use the activations, Equation (3.5), of the nodes of the previous layer as follows,

Output of the second hidden layer

$$\begin{aligned} a_i^{(2)} &= f^{(2)}\left(\sum_{j=1}^{n^{(1)}} w_{ij}^{(2)} a_j^{(1)} + b_i^2\right) \\ &= f^{(2)}\left[\sum_{j=1}^{n^{(1)}} w_{ij}^{(2)} f^{(1)}\left(\sum_{k=1}^{n_x} w_{jk}^{(1)} x_k + b_j^{(1)}\right) + b_i^{(2)}\right] \end{aligned} \tag{3.6}$$

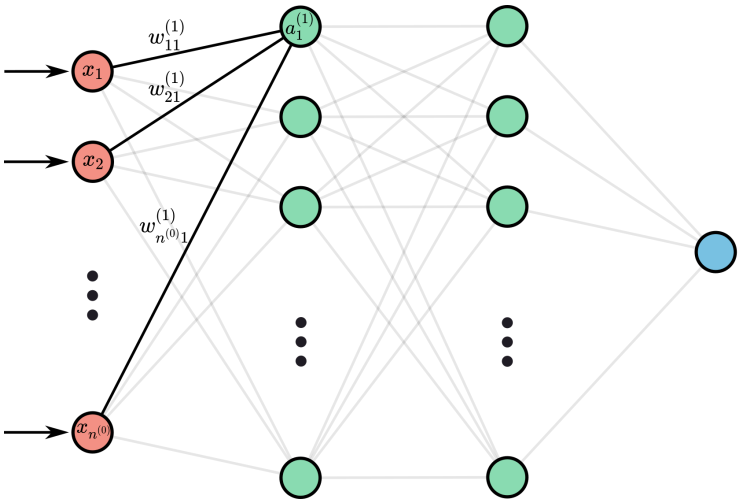


Figure 3.2: Figure showing the activation $a_1^{(1)}$ in a dense neural network

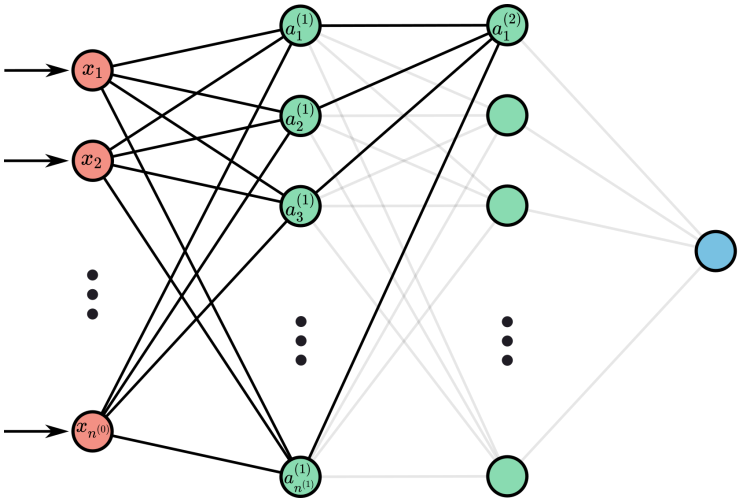


Figure 3.3: Figure showing the activation $a_1^{(2)}$ in a dense neural network

Figure 3.3 shows an illustration of the activation for node $j = 1$ in the second hidden layer ($l = 2$).

Continuing with our example, we are now at the end-point of our network, namely the *output layer*($l = 3$):

Output of the last layer

$$y_i^{(3)} = f^{(3)}\left(\sum_{j=1}^{n^{(2)}} w_{ij}^{(3)} a_j^{(2)} + b_i^{(3)}\right), \qquad i = 1. \tag{3.7}$$

By inserting the activation expression Equation (3.6) for $a_i^{(2)}$ into expression $a_j^{(2)}$ in Equation (3.7) we obtain the final output, but also a colossal equation so we will leave its form up to the imagination of the reader. We note that we have only one output node so $i = 1$ is indeed the final and only output. In the case of *regression* the output node(s) is usually the identity $f^{(3)}(z_i) = z_i$, while for *classification* it is usually the sigmoid or its multi-class generalization, the softmax (See Section 7.1.2). General formula for feed-forward propagation is thus

Feed-Forward formula

$$\begin{aligned} a_i^{(l)} &= f\left(\sum_{j=1}^{n^{(l-1)}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}\right), \\ i &\in \left\{1, 2, \ldots, n^{(l)}\right\}, \quad l \in \{1, 2, \ldots, L\}, \end{aligned} \tag{3.8}$$

where L is total number of layers, excluding the input layer [27].

3.2 Activation Functions

A range of activation functions are applied and tested in this thesis in order to find the best suited for the problem. We can say that the application of various activation functions serve two purposes: Transforming the final output of the neural network into a range sensible for classification purposes, or transforming the output of the hidden layers to impose non-linearity, consequently enhancing the predictive power of our network. This thesis focuses on a regression problem, thus the final output will *always use the identity as activation function*. If our problem was one of classification we would have to use the sigmoid, or the softmax, in order to conform the outputs to the normalization of probability, i.e. transform the range to $0 \leq y_j \leq 1$ for output from node j.

What type of activation function is appropriate for the hidden layer of a neural network may vary depending on the type of network, thus we shall use a variety of different activation functions and evaluate their performance. We list the functions and their derivatives in Table 3.2 for reference. In the context of choosing the right activation function there are two recurrent concepts that come up in the literature: *Exploding gradients*, and the *vanishing gradient problem*. Shortly put, the former essentially means divergent learning where weights in each step of an iterative procedure (See Equation (2.5)) tends to infinity. The latter is essentially the opposite problem. If we look at the sigmoid in

Table 3.2 it can be seen that as $\lim_{x\to\pm\infty}$ the curve becomes approximately flat and the derivative in this region will thus be approximately zero. When we have covered back-propagation in Equation (3.14) it should become clear that gradients of zero means the network will not learn. Choosing the right activation function for the hidden layers is therefore crucial to avoid the stated problems. For the ensuing definitions and examples in this chapter we shall use the sigmoid activation function as the example function,

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{3.9}$$

3.3 Cost Functions

The *cost function* $\mathcal{C}$ is the optimization objective that we seek to minimize. We have in fact encountered a cost function already in Section 2.2, namely the MSE,

$$\mathcal{C}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2\,. \tag{3.10}$$

which is the most commonly used cost function in regression problems. The term *loss function* $\mathcal{L}$ is often used interchangeably with cost function, but strictly speaking the loss function is defined for a *single data point*, while the cost function $\mathcal{C}$ is the average of these losses:

$$\mathcal{C}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_i. \tag{3.11}$$

We will use these terms interchangeably as well, but the context should be clear. The reason for this is because of concepts, that we have yet to encounter, such as *loss-curves* (See Figure 7.2), are almost exclusively referred to as loss or cost, irregardless of the number of data points.

Let us recap some of the discussion from Section 2.2, but add some points to notice. The MSE has the simple interpretation of being the average of the squared distances between the prediction $\hat{y}_i$, and the true value, or target y_i (we shall use target and true value interchangeably). As covered in section 2.2, we square the difference $y_i - \hat{y}_i$ so we do not differentiate between the prediction overestimating or underestimating the target value. This has the added effect of the cost having quadratic growth with respect to the difference. Depending on the problem at hand, this may be an advantage if we have data points situated far away from the main bulk of the data, called *outliers*, that we want our model to accommodate. It is a disadvantage in the cases where we have outliers that we want to ignore. In the cases where we want outliers to incur a smaller loss upon our average cost we can use the *mean absolute error* (MAE) instead,

$$\mathcal{C} = \frac{1}{n}\sum_{i=1}^{n}|y_i-\hat{y}_i|\,. \tag{3.12}$$

Even though the MAE is more robust to outliers its added expense is that its gradient $\nabla_{\mathbf{w}}\text{MAE} = \textit{constant}$ is constant, giving a large gradient irrespective of whether the loss is large or small. It is therefore best applied together with an adaptive scheme for gradients, iteratively reducing the size of the gradients. Either way, MSE or MAE, an adaptive scheme is generally superior for neural networks, and thus we will cover it briefly in Equation (3.30).

We spoke briefly about gradients in the discussion above, so in that regard, let us next introduce the method of *Gradient Descent*.

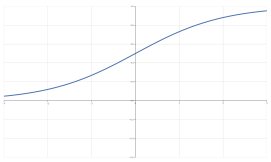
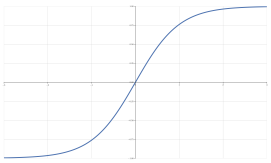
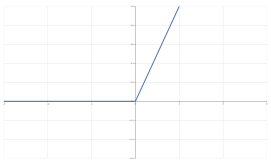
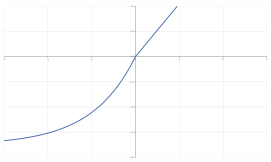
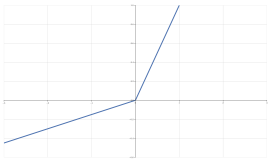
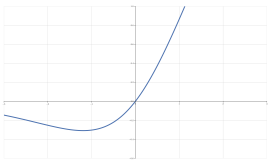
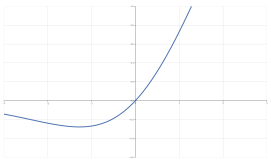
Name	$f(x) =$	$f'(x) =$	*f(x), 1D graph*
Sigmoid $\sigma(x)$	$\frac{1}{1+e^{-x}}$	$\frac{e^{-x}}{(e^{-x}+1)^2}$	
TanH	$\frac{2}{1+e^{-2x}} - 1$	$1 - f(x)^2$	
ReLU	$\begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$\begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	
S(ELU)	$\begin{cases} \lambda x & \text{if } x > 0 \\ \lambda\alpha\left(e^x - 1\right) & \text{if } x \leq 0 \end{cases}$	$\begin{cases} \lambda & \text{if } x > 0 \\ \lambda\alpha e^x & \text{if } x \leq 0 \end{cases}$	
Leaky ReLU	$\begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$	$\begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases}$	
Mish	$x \cdot \tanh(\text{sp}(x))$	$\text{sech}^2(\text{sp}(x))x\sigma(x) + \frac{f(x)}{x}$	
Swish	$x \cdot \sigma(\beta x) = \frac{x}{1+e^{-\beta x}}$	$\beta f(x) + \sigma(\beta x)(1-\beta f(x))$	

Table 3.2: Activation functions. The ELU [28] activation function is a special case of what we have written as SELU, but with $\lambda = 1$. The SELU activation function is specifically defined by its authors as having $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$ [29] (The figure plotted). For the ELU and LeakyReLU [30] standard choices of the coefficient is $\alpha = 1$ and $\alpha = 0.15$ respectively. For the Swish [31] activation function we will use $\beta = 1$. The argument of Mish [32] is commonly referred to as the *softplus* and is $\text{sp}(x) = \ln(1+e^x)$, which we shall encounter again in Chapter 5. Note, for comparison, the SELU and TanH vertical axis goes to -2 and -1, respectively, while the other plots to -0.6, all else equal.

3.4 Gradient Descent Optimization

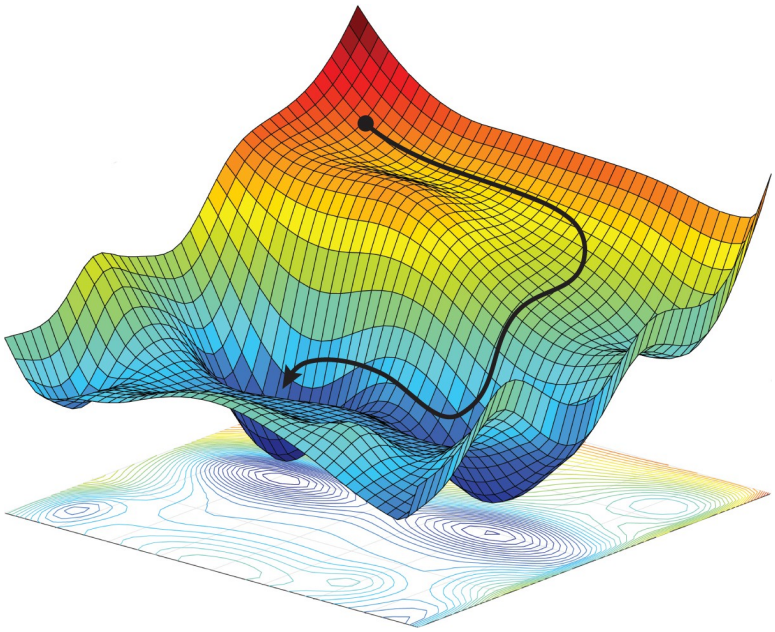


Figure 3.4: Gradient descent in a cost landscape, $\mathcal{C}(w_0, w_1)$. Image from: [33].

The feed-forward algorithm of Section 3.1 outputs initial predictions $\hat{y}_i$ for every data point x, and with these predictions we may evaluate the cost function $\mathcal{C}(\mathbf{w})$ from Section 3.3. Next we want to adapt our network weights $\mathbf{w}$ with the goal of reducing the *cost*, the output value of the cost function. There are several kinds of methods used to attain this goal, the most widely used class of methods for neural networks is *gradient descent* (GD). Let us recap briefly from calculus: The *gradient* of a function f is the vector field ∇f, and for parameters $\mathbf{w} = (w_0, w_1, \dots w_n)$, its components are equal to the partial derivatives

$$\nabla f(\mathbf{w}) = \begin{bmatrix} \frac{\partial f}{\partial w_0} \\ \vdots \\ \frac{\partial f}{\partial w_n} \end{bmatrix}. \tag{3.13}$$

The gradient vector will point in the direction of fastest ascent. If we flip the vector by changing sign $\nabla f \Rightarrow -\nabla f$ the vector will point in the direction of fastest descent. We may use this feature to update our weights by a small step in the direction of the negative gradient of the cost function [26]:

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla \mathcal{C}\left(\mathbf{w}_t\right). \tag{3.14}$$

Here η is called *the learning rate*. The learning rate modulates the magnitude of the gradient vector, i.e. the length of the steps in parameter space. The learning rate is a *hyperparameter*, which are parameters that are set manually before model-training. Optimal hyperparameters usually have to be searched for by brute-force method. The prefix is added to differentiate hyperparameters from the parameters optimized during training (weights and biases). Using the update rule in Equation (3.14) we would

continue to update our weights until we reached a stopping criteria. Unfortunately, we are not guaranteed that we will reach the globally lowest point of the terrain that is our parameter space. Gradient descent is ensured to reach a local minimum, but not necessarily a global minimum [24]. Only if the cost function $\mathcal{C}(\mathbf{w})$ is convex on its whole domain can we rely on it reaching the global minimum, but in a non-linear, vast parameter space, as is the case with a deep neural network, we are far from being able to count on convexity.

3.4.1 Batch and Mini-batch gradient descent

Every update of the weights in the gradient descent algorithm involves calculating $\nabla\mathcal{C}$. We saw in Equation (3.11) that the total cost $\mathcal{C}$ is formulated as the mean value of losses. This means we are assessing the whole set of losses $\mathcal{C} = \mathcal{L}_0 + \dots \mathcal{L}_n$ in one go. Computing the gradient is very expensive, as we consider the entire set of data examples and make a model prediction for each, in order to compute just one iteration of the gradient descent algorithm. If we have a huge dataset it might take a long time for the algorithm to converge. A different approach, typical in machine learning, is to estimate the cost function using only a *subset of randomly sampled sets* of pairs $(y, \hat{y})$, called a *batch*:

$$\nabla\mathcal{C} = \frac{1}{S}\sum_{s=1}^{S}\nabla\mathcal{L}_s, \tag{3.15}$$

where S is the total number of samples in one *batch*, and is called *batch-size*. The sampling is executed *without* replacement, as this gives faster convergence than with replacement [34]. If we choose the batch-size to be equal to the number of data examples, $S = n$, we have standard gradient descent, often called *batch gradient descent*(BGD), while if $S = 1$ the method is called *stochastic gradient descent*(SGD). In the following discussion we refer to *iterations* as iterating over the batches, e.g. in SGD we would iterate n times to cover the whole data set. We refer to *epochs* as iterating over the whole data set multiple times, so that the total number of iterations is equal to iterations $\times$ epochs.

With SGD we may speed up the learning process considerably for non-convex manifolds compared to BGD, because the updating of the weights is processed after each data example. However for the same number of epochs, the SGD is more computationally intensive than BGD. Going through the entire dataset involves sampling, calculating $\nabla\mathcal{C}$, and updating weights, each n times. While BGD requires just one update for n examples, but as already stated SGD may still be faster as it does not necessarily need as many epochs, or even iterations, as BGD. The SGD has unstable convergence, meaning that we have noisy gradients. The consequence of this is that the cost fluctuates markedly on its path towards the global minimum. The fluctuations also includes the expense of the gradient not being able to settle completely at the global minimum. Though the noise may be disadvantageous, it also produces the favourable trait that the SGD is more robust to local minimia and saddles.

It is common with neural networks to try to obtain the best of both worlds and pick a batch-size somewhere between the two extremes of BGD and SGD, when this is the case the method is called *mini-batch gradient descent*(MBGD). The choice of S is another hyperparamter that affects model perfomance, thus batch-size is a parameter that requires tuning with respect to the data and model-choice.

3.4.2 Momentum based gradient descent

Stochastic variants of gradient descent as described in the previous section remain popular and may be well suited to many problems. That said, they can be slow, and they still bear the risk of floundering about in a local minimum. In order to accelerate learning, we introduce a method loosely inspired by physics, namely the method of *momentum*. Imagine a ball rolling down a frictionless mountain. It picks up momentum, and as it reaches a small valley or plateu, its accumulated momentum is enough to propel it further overshooting the valley. It further continues down the mountain slope until it reaches sea level.

We can consider the negative gradient vector $\nabla C_{\mathbf{w}}$ as analogous to an external potential $F = -\nabla f$ acting on a particle through parameter space. Just as the ball in the example above, we want the particle to overshoot local minima on its way to the global minimum. Put another way, we want momentum to smooth out the gradients over time. Say our particle has mass m and time-dependent position $x(t)$ and velocity $v(t) = \dot{x}(t)$. We simply start with Newton's second law:

$$-\nabla f(x(t)) = m\ddot{x}(t). \tag{3.16}$$

Adding to this equation of motion we would also like a dampening effect on our particle so that it comes to rest at the global minimum. Let us look at some resistive forces as options: Turbulent drag $f_D \propto -v^2$ will be too weak to prevent motion when the velocity is small, while dry friction $f_\mu = -$constant will be to strong and may halt motion before the minimum has been reached. The linear velocity, laminar drag $f_D \propto -v$, is the best suited physical analogy for our problem. We implement laminar drag parameterised with a constant ϵ into our equation of motion,

$$-\nabla f(x(t)) = m\left[\ddot{x}(t) + \epsilon\dot{x}(t)\right]. \tag{3.17}$$

Expressed as a system of ordinary first order differential equations suitable for numerical solutions,

$$\dot{x}(t) = v(t) \quad \text{and} \quad \dot{v}(t) = -\frac{\nabla f(x(t))}{m} - \frac{\epsilon}{m}v(t). \tag{3.18}$$

An approximate solution for v and x can be obtained by applying the *semi-implicit Euler method*:

$$v_{t+1} = v_t - \frac{\epsilon}{m}v_t - \frac{\Delta t}{m}\nabla f(x(t)) \tag{3.19}$$

$$x_{t+1} = x_t + v_{t+1}\Delta t, \tag{3.20}$$

where t is the time steps of site Δt. We redefine Equation (3.19) writing $p_{t+1} = mv_{t+1}\Delta t$ and $p_t = mv_t$, essentially moving the time step of the Equation (3.20) to Equation (3.19). We call p momentum, however, p_{t+1} is not in units of momentum. The resulting equation is then

$$\begin{aligned} p_{t+1} &= \Delta t(m-\epsilon)p_t - (\Delta t^2)\nabla f(x(t)) \\ x_{t+1} &= x_t + p_{t+1}. \end{aligned} \tag{3.21}$$

We have illustrated the physical connection of this scheme, and we are now ready to move this into the realm of machine learning. First we assume unit mass $m = 1$ in Equation (3.21), so that $p = v$. Next we may redefine the coefficients in the update

rule to be the *hyperparameters* $\gamma = \Delta t(1-\epsilon)$ and $\eta = (\Delta t)^2$. We can freely set the parameters without considering their mutual dependence on Δt, because whatever value we choose for η we can always obtain any real value for γ by picking the right $1-\epsilon$. Lastly we will change $\nabla f(x) \Rightarrow \nabla \mathcal{C}(\mathbf{w})$, and the position x to a multi-dimensional vector of weights $\mathbf{w}$. Putting all of the above together gives

Momentum based gradient descent

$$\begin{aligned} \mathbf{v}_{t+1} &= \gamma \mathbf{v}_t - \eta \nabla \mathcal{C}(\mathbf{w}_t), \quad \gamma \in [0,1) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \mathbf{v}_{t+1}, \end{aligned} \tag{3.22}$$

which is the update rule for momentum based gradient descent. We identify η as the learning rate, and γ as the new momentum parameter. Both are hyperparameters and will have to be tuned according to the model and data at hand. We see that if $\gamma = 0$ in Equation (3.22) the update rule reduces to the update rule of ordinary gradient descent, Equation (3.14). If we write out a series of steps of v_{t+1} we can also see why we restricted $\gamma \in [0,1)$. We use initial condition $\mathbf{v}_0 = 0$ and assume for the sake of an example that the gradient is unchanged $\nabla \mathcal{C}(\mathbf{w}_t) = \mathbf{g}$ for all t,

$$\begin{aligned} \mathbf{v}_1 &= -\eta \mathbf{g} \\ \mathbf{v}_2 &= \gamma \mathbf{v}_1 - \eta \mathbf{g} = -\eta \mathbf{g}(\gamma + 1) \\ \mathbf{v}_3 &= \gamma \mathbf{v}_2 - \eta \mathbf{g} = -\eta \mathbf{g}\left(\gamma^2 + \gamma + 1\right) \\ &\dots \\ \text{If } |\gamma| < 1 &\text{ , this converges to} \\ \lim_{t \to \infty} \mathbf{v}_t &= -\frac{\eta \mathbf{g}}{1-\gamma}. \end{aligned} \tag{3.23}$$

Negative values $\gamma < 0$ will just reduce the effect of the gradients compared to ordinary gradient descent, defeating the purpose of momentum, while $\gamma > 1$ will change the direction of motion. On the contrary, if we choose $\gamma = 0.99$ we get a 100-fold terminal velocity increase compared to ordinary gradient descent, $v_\infty = -100\eta \mathbf{g}$. If the gradients are instead variable with respect to t we can see that the γ parameter decides the degree of smoothing between the gradients, like a moving average. For example the series in Equation (3.23) for $t = 3$ would be $\mathbf{v}_3 = -\eta\left(\mathbf{g}_1\gamma^2 + \mathbf{g}_2\gamma + \mathbf{g}_3\right)$, so there is an exponential decay of previous gradients.

3.4.3 Adaptive Gradient Algorithm(AdaGrad)

Another way to improve the rate of convergence of gradient descent is to change the learning rate η adaptively during training. A simple scheme would be to begin with a high learning rate, and then experiment with the most appropriate way to iteratively reduce it as the gradient approaches the global minimum, so it does not overshoot. The problem with such an approach is that in a multilayered neural network, the learning rates that is most appropriate can vary substantially for different weights $w^{(i)}$, as the magnitude of the gradients can be very different. It is thus the case that since all the weights are updated simultaneously by the same adjustment we might overshoot the target-minimum in some of the weight dimensions. Writing out the update rule

Equation (3.14),

$$\begin{bmatrix} w_{t+1}^{(1)} \\ w_{t+1}^{(2)} \\ \vdots \\ w_{t+1}^{(M)} \end{bmatrix} = \begin{bmatrix} w_{t}^{(1)} \\ w_{t}^{(2)} \\ \vdots \\ w_{t}^{(M)} \end{bmatrix} - \begin{bmatrix} \eta g_{t}^{(1)} \\ \eta g_{t}^{(2)} \\ \vdots \\ \eta g_{t}^{(M)} \end{bmatrix}, \tag{3.24}$$

where the superscripts now denote vector components. We see how the gradient elements $g_t^{(i)}$ share the same η. Take for examples two of the gradient components, $g_t^{(1)} = \frac{\partial \mathcal{C}}{\partial w^{(1)}}$ and $g_t^{(2)} = \frac{\partial \mathcal{C}}{\partial w^{(2)}}$. Say we have $g_t^{(1)} \gg g_t^{(2)}$, with $g_t^{(1)}$ being so large that it overshoots the minima and diverges in the $w^{(1)}$ dimension, while $g_t^{(2)}$ is very small and makes almost no progress at all in the $w^{(2)}$ dimension. A common adapted η cannot fix both problems at once. We could instead add a per-parameter scaling factor to the initial learning rate that adjusts the gradient components independently. In *AdaGrad* we scale the learning rate by previous gradients as follows

AdaGrad

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{((G_t) + \varepsilon I)_{ii}}} \cdot \mathbf{g}_t \tag{3.25}$$

where we take the diagonal of the terms in the denominator. G_t is a diagonal matrix where every element along the diagonal is the sum of the squares of the previous gradients up to the current time step t, $G_t = \sum_{\tau=1}^{t} \mathbf{g}_\tau \mathbf{g}_\tau^{\mathrm{T}}$. We also add ε, which is a small number, to avoid dividing by zero.

For a given weight $w = w^i$, $g = g^i$ the AdaGrad algorithm is

$$w_{t+i} = w_t - \frac{\eta}{\sqrt{g_1^2 + g_2^2 \dots g_t^2 + \varepsilon}} g_t, \tag{3.26}$$

where $g_j = \left(\frac{\partial \mathcal{C}}{\partial w^i}\right)_{\tau=1}$ is the partial derivative calculated for the jth iteration, corresponding to the weight w^i.

AdaGrad's advantage: It is pretty insensitive to the initial learning rate. It performs larger updates on weights with a history of meager gradients, speeding up the learning, and smaller updates on weights with a history of sizeable gradients. AdaGrad's disadvantage: The learning rate decreases monotonically as the sum in the denominator keeps accumulating positive terms. At some point the learning rate will become so small that the neural network in effect stops learning. This may, depending on the problem and the model, be too radical of a reduction to obtain the best results. The next method we cover seeks to ameliorate this issue.

3.4.4 Root mean squared propagation (RMSprop)

This method is an unpublished method that was first presented by Geoffrey Hinton, considered one of the "godfathers" of deep learning, as part of his open online course on neural networks [35]. The method seeks to resolve AdaGrad's problem of excessively reducing the learning rates. The *RMSprop* update rule is:

RMSprop

$$\begin{aligned} \mathbf{s}_{t+1} &= \rho \mathbf{s}_t + (\rho - 1)\mathbf{g}_t \odot \mathbf{g}_t \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{\eta}{\sqrt{\mathbf{s}_{t+1} + \epsilon}} \odot \mathbf{g}_t, \end{aligned} \tag{3.27}$$

where we have, by a slight abuse of notation, changed the notation from Equation (3.25) to using $\mathbf{g}_t \odot \mathbf{g}_t$ instead, which is the component-wise multiplication(Hadamard product) common in machine learning literature:

$$\begin{bmatrix} g_t^{(1)} \\ g_t^{(2)} \\ \vdots \end{bmatrix} \odot \begin{bmatrix} g_t^{(1)} \\ g_t^{(2)} \\ \vdots \end{bmatrix} = \begin{bmatrix} g_t^{(1)} * g_t^{(1)} \\ g_t^{(2)} * g_t^{(2)} \\ \vdots \end{bmatrix}. \tag{3.28}$$

For a given weight $w = w^i$ and $g = g^i$ we write out the expectation s_t with $s_0 = 0$,

$$\begin{aligned} s_1 &= \rho s_0 + (\rho - 1)g_1^2 \\ s_2 &= \rho s_1 + (\rho - 1)g_2^2 \\ &= \rho(\rho - 1)g_1^2 + (\rho - 1)g_2^2 \\ &\dots \\ s_t &= \rho^t(\rho - 1)g_1^2 + \rho^{t-1}(\rho - 1)g_2^2 + \ldots + (\rho - 1)g_t^2. \\ s_t &= (1 - \rho)(\rho^t g_1^2 + \rho^{t-1} g_2^2 + \ldots + g_t^2). \end{aligned} \tag{3.29}$$

Just as with the momentum series in Equation (3.23), we have $1 + \rho + \rho^2 + \ldots = \frac{1}{1-\rho}$. Setting the gradients $g_1 = g_2 \ldots = g_t$ and assuming $t \Rightarrow \infty$, we that s_t in Equation (3.29) is normalized to 1.

Furthermore, given a step t the previous gradients in $g_{t-1}, g_{t-1}, \ldots$ in Equation (3.29) exponentially fades from memory in the learning rate's adjustment factor, and s_t is therefore called an *exponentially weighted moving average* of previous gradients. Given, of course, that we pick $\rho \in [0, 1)$. A popular choice in software is $\gamma = 0.9$.

3.4.5 Adaptive Moment Optimization(ADAM)

All the of the gradient methods covered by now have really just been a warm up for the main gradient method used in this thesis, namely ADAM [36]. The reason for this build up is that Adam combines the momentum method Section 3.4.2, for a smoother update of the gradients, and RMSprop Section 3.4.4 for the normalizing tendency it has on the learning rates. It is also well suited for and commonly used with mini-batches Section 3.4.1. ADAM is at the time of writing one of the most popular optimization methods within the field of deep learning. The ADAM update rule is as follows:

ADAM

$$
\begin{aligned}
&\mathcal{C}\left(\mathbf{w}_t\right) \\
&\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + \left(1-\beta_1\right)\mathbf{g}_t \\
&\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + \left(1-\beta_2\right)\mathbf{g}_t \odot \mathbf{g}_t \\
&\hat{\mathbf{m}}_{t+1} = \frac{\mathbf{m}_{t+1}}{1-\beta_1^{t+1}} \\
&\hat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1-\beta_2^{t+1}} \\
&\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{\hat{\mathbf{v}}_{t+1}+\epsilon}} \odot \hat{\mathbf{m}}_{t+1}
\end{aligned}
\tag{3.30}
$$

The first line of Equation (3.30) is the momentum method of Section 3.4.2, but with weighted moving average $\beta \in [0,1)$. The second line is the RMSprop of Section 3.4.4. ADAM also introduces a statistical bias correction of the first and second moments in line four and five.

3.5 Backpropagation

Now that we have established how we can update and optimize the weights and biases of a classical neural network, what remains is to formulate a method for calculating the gradient $\mathbf{g} = \nabla_{\mathbf{w}}\mathcal{C}$. The most widely used method is called *backpropagation*.

During training, we calculate a new instance of the gradient, in accordance with a gradient descent procedure, for every iteration $t_0, t_1, \ldots t_{max}$. One such iteration involves first calculating a forward pass using the feed-forward procedure Equation (3.8), which will yield an output $\hat{y}$. Next the backward pass is executed: The backward pass involves calculating the partial derivatives of the cost function $\mathcal{C} = C(y, \hat{y}(\vec{w}, \vec{b}))$ with respect to all weights and biases in all layers, so that one step of the gradient descent constitutes

$$
\begin{aligned}
w_{jk}^{(l)} &\leftarrow w_{jk}^{(l)} - \eta \frac{\partial \mathcal{C}}{\partial w_{jk}^{(l)}} \\
b_{k}^{(l)} &\leftarrow b_{k}^{(l)} - \eta \frac{\partial \mathcal{C}}{\partial b_{k}^{(l)}},
\end{aligned}
\tag{3.31}
$$

for all layers and nodes

$$
\begin{aligned}
j &= 1, \ldots, n^{(l-1)} \\
k &= 1, \ldots, n^{(l)} \\
l &= 1, \ldots, L
\end{aligned}
\tag{3.32}
$$

Here we have used the simplest of the gradient descent techniques covered in Equation (3.14). As the goal here is to access all partial derivatives of the weights in all layers, the backpropagation is in essence just a glorified chain rule formula. We shall use the MSE cost function (See Section 3.3) for the subsequent calculations, but for the sake of an example we can work with a simpler expression. First, we assume that we only have a single output $n^{(L)} = 1$. Secondly, we will just calculate the partial

derivatives corresponding to a single data example, so $S = 1$, that is the loss $\mathcal{L}$. Thus the gradient objective is

$$\mathcal{L} = (y - \hat{y})^2. \tag{3.33}$$

For the hidden layers we use $\sigma(x)$ for activation function and its derivative $\sigma'(x)$, see Table 3.2 for the full expression of the derivative. Furthermore, as this is regression, we do not transform the outputs, that is, we use the identity activation for the output node. Hence the output is equal to the linear combinations

$$\hat{y} = a_k^{(l)} = f(z_k^{(l)}) = z_k^{(l)}. \tag{3.34}$$

Recall from Section 3.1 that we have

$$z_k^{(l)} = \sum_{j=1}^{n^{(l-1)}} w_{jk}^{(l)} a_j^{(l-1)} + b_k^{(l)}. \tag{3.35}$$

Since we only have one output node in this network, that is, $\mathcal{L} = (y - a_k^{(l)})^2$, we will not have a sum over k for the output layer. Note though that this would be different if we had two outputs, such as for a Bayesian neural network with a standard deviation and a mean, which we will encounter later.

Let us follow the same example-network that we looked at in Section 3.1, where we predefined the number of hidden layers to two. Using Equation (3.35) the partials with respect to the weights of the last hidden layer to the output node are

Partials to the 3rd (output) layer

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(3)}} = \frac{\partial \mathcal{L}}{\partial z_k^{(3)}} \frac{\partial z_k^{(3)}}{\partial w_{jk}^{(3)}} = 2(y - \hat{y}) a_j^{(2)}, \tag{3.36}$$

where $a_j^{(2)}$ is the feed-forward activation values from the nodes of the preceding layer,

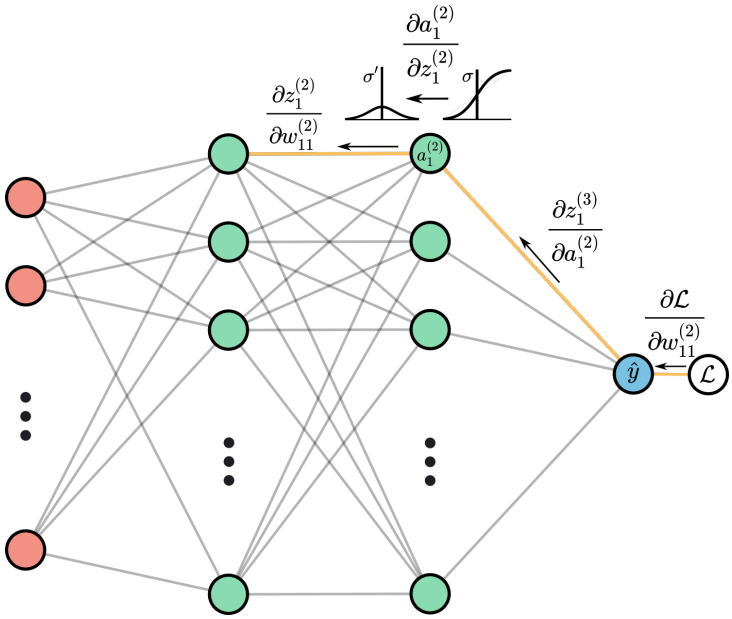
$$a_j^{(2)} = \sigma(z_j^{(2)}) = \sigma(\sum_{i=1}^{n^{(2)}} w_{ij}^{(2)} a_i^{(1)} + b_j^{(2)}). \tag{3.37}$$

We can propagate deeper into the network through the activation $a_j^{(2)}$, which is illustrated in Figure 3.5a, and formulated as follows

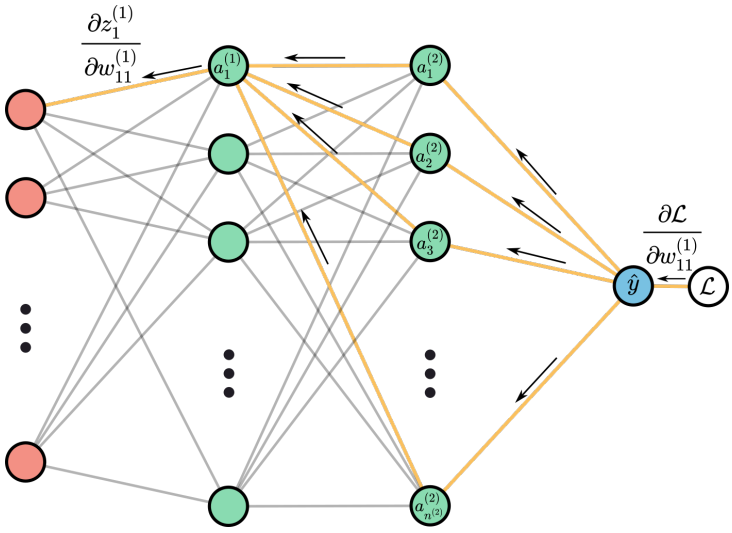
Partials to the 2nd hidden layer

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_{ij}^{(2)}} &== \frac{\partial \mathcal{L}}{\partial z_k^{(3)}} \frac{\partial z^{(3)_k}}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial w_{ij}^{(2)}} \\ &= 2(y - \hat{y}) w_{jk}^{(3)} \sigma'(z_k^{(2)}) a_i^{(1)}. \end{aligned} \tag{3.38}$$

For the edges between $l = 0$ to $l = 1$ the weights may influence the output through multiple different paths which can be seen in Figure 3.5b. The sum over these paths is kept as an implicit sum over j,



(a) Partial $\frac{\partial \mathcal{L}}{\partial w_{11}^{(2)}}$ to the 2nd hidden layer.



(b) Partial $\frac{\partial \mathcal{L}}{\partial w_{11}^{(1)}}$ to the 1st hidden layer. Note that there are many paths to the partial derivative.

Figure 3.5: Partial derivative examples from the backpropagation procedure

Partials to the 1st hidden layer

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ki}^{(1)}} &= \frac{\partial \mathcal{L}}{\partial z_k^{(3)}} \frac{\partial z_k^{(3)}}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial a_i^{(1)}} \frac{\partial a_i^{(1)}}{\partial z_i^{(1)}} \frac{\partial z_i^{(1)}}{\partial w_{ki}^{(1)}} \\ &= 2(y-\hat{y}) w_{jk}^{(3)} \sigma'(z_j^{(2)}) w_{ij}^{(2)} \sigma'(z_i^{(1)}) x_k.\end{aligned} \tag{3.39}$$

Calculation of partials w.r.t. to the biases follows the same logic. It might be clear by now that there is a recursion rule at play. The partial derivatives of deeper layers depend on the partial derivatives of shallower layers. In general we have,

Backpropagation

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} &= \frac{\partial \mathcal{L}}{\partial z_k^{(l)}} a_j^{(l-1)}, \quad a_j^{(0)} = x_j, \quad l = 1, \ldots, L, \\ \frac{\partial \mathcal{L}}{\partial b_k^{(l)}} &= \frac{\partial \mathcal{L}}{\partial z_k^{(l)}}, \quad l = 1, \ldots, L \\ \frac{\partial \mathcal{L}}{\partial z_k^{(l)}} &= f'\left(z_k^{(l)}\right) \sum_{j=1}^{n^{(l+1)}} \frac{\partial \mathcal{L}}{\partial z_j^{(l+1)}} w_{kj}^{(l+1)}, \\ l &= 1, \ldots, L-1 \\ \frac{\partial \mathcal{L}}{\partial z_k^{(l)}} &= 2(y-\hat{y}).\end{aligned} \tag{3.40}$$

We should note that the last equation is specific for the MSE cost function. The gradient vector of the loss, $\nabla \mathcal{L}$, will have length equal to the number of weights and biases. The components of the gradient vector are all the partials $\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial b_k^{(l)}}$. Note, that for computation it is convenient to vectorize the backpropagation algorithm, however, we will omit the description and vectorize only when necessary.

3.6 Predictions

When the network has been trained we will have a set of parameters $(\hat{\mathbf{w}}, \hat{\mathbf{b}})$ fitted to the specific dataset $\mathcal{D} = (\mathbf{X}_{train}, \mathbf{y}_{train})$. In a similar fashion to what we did with linear regression in Equation (2.10) we can now produce the out-of-sample and in-sample predictions,

$$\begin{aligned}\hat{\mathbf{y}} &= \hat{f}(\mathbf{X}_{\text{test}}) \\ \tilde{\mathbf{y}} &= \hat{f}(\mathbf{X}_{\text{train}}),\end{aligned} \tag{3.41}$$

and calculate the corresponding test and train errors, $\text{MSE}(\hat{\mathbf{y}}, \mathbf{y})$ and $\text{MSE}(\tilde{\mathbf{y}}, \mathbf{y})$ respectively. In equation Equation (3.41) $\hat{f}$ is the fitted network. Passing in the test examples involves one feed-forward pass, that is, using $\mathbf{X}_{\text{test}}$ as input into the nested function which we provided an example of in Equation (3.7). But unlike the multiple forward-passes executed during the training procedure, we now use the fitted parameters $(\hat{\mathbf{w}}, \hat{\mathbf{b}})$ in the feed-forward formula.

Chapter 4

A Probabilistic Perspective

4.1 Interpretations of Probability

As an intellectual warm-up before we delve into mathematical technicalities, let us ponder for a second on a question of a philosophical nature: What is probability? According to Murphy[37] there are at least two different interpretations of probability. The two interpretations commonly contrasted with one another are the *frequentist* and *Bayesian* interpretations.

In the frequentist interpretation probabilities represent frequencies of events ad infinitum. So if we say a coin is fair with a 50% probability of giving heads, what we actually mean is that after a hypothetical infinite number of perfectly balanced tosses the number of outcomes giving heads should be half of the total number of outcomes. The Bayesian interpretation on the other hand considers probability not as a long run frequency, but as a quantification of uncertainity. It is at its core related to a judgement based on the information at hand rather than postulation about the outcome of repeated trials [37]. In the case of the fair coin we would assert a 50% degree of belief that the coin toss will give heads based on previous trials, or possibly trials of similar coins. As experiments are carried out we update our degree of belief in correspondance with the outcomes. In this format probabilities do not represent a feature of the external world, but rather a feature of your subjective belief.

The convenience of a Bayesian interpretation is at its most pertinent when considering non-repeatable events. For example we may ask, what is the probability $p(H_A)$ of candidate A winning over candidate B in an election a given year, where H_A is the hypothesis that A will win the election. The outcome can only occur 0 or 1 number of times, so it's not repeatable. A frequentist may lend us useful mathematical tools, but his interpretation of probability as it relates to the real world, will arguably lead to absurd thought experiments of repeating the election a large number of times with identical circumstances. With the Bayesian interpretation, you may instead rationalize the probability $p(H_A)$ as a degree of belief, or plausible reasoning. Of course, as humans are notoriously bad at assessing probabilities, just any degree of belief might be a hard sell. Multiple arguments and frameworks exist that seek to put logical conditions onto degrees of beliefs, so as to argue whether they are rational or not. Such arguments are essentially based upon the notion that degrees of beliefs are not devised in a vacuum, meaning that a degree of belief is ranked relative to other degrees of belief.

Cox's postulates An example of an axiomatic approach to Bayesian probability is outlined in Cox's theorem [38], which is a set of postulates that serves as a justification for the Bayesian perspective and formalizes the Bayesian interpretation of probability. With Cox's postulates, different *agents* with the same prior information assessing the same set of outcomes will not necessarily have the same values for probabilities corresponding to different hypotheses, but they should agree on the order $p(H_0) < p(H_1) < \ldots < p(H_n)$ of probabilities.

The postulates involve, for example, that agents should be consistent, quoted from Jaynes' book, where agents are robots [38],

> "The robot always takes into account all of the evidence it has relevant to a question. It does not arbitrarily ignore some of the information, basing its conclusions only on what remains. In other words, the robot is completely non-ideological"

Jaynes shies away from calling the elements of this system for "axioms", and rather calls them "desirable goals", and uses "robot" instead of "agent" possibly to indicate that the rationality is not a behaviour fully expected by humans[1]. We will not delve much further into the details of the formalization of Bayesian probability, but we will bring along with us the main takeaway: The postulates can be formulated in terms of boolean logic, encoding "common sense". It can then be shown that *the laws of probability theory can be derived from Cox's postulates*. The postulates serves as a vindication for the Bayesian approach as they provide an alternative to the canonical Kolmogorov axioms which formalizes probability from a frequentist point of view. We should remark that we have entered contested territory and that there is an on-going debate on the validity of Cox's theorem [38].

Consequentially, underneath the hood of both Bayesian and frequentist statistics lies the rules of probability theory. So, before we flesh out Bayesian statistics let us state the rules, namely *the sum rule* and *the product rule*.

Probability Rules

$$P(X) = \sum_{Y} P(X, Y) \tag{4.1}$$

$$P(X, Y) = P(Y|X)P(X) \tag{4.2}$$

A remark on notation: Capital $P(.)$ denotes a discrete point probability, while $p(.)$, which will appear later, is a discrete or contiuous probability distribution.

The Sum Rule The sum rule of probability seen in Equation (4.1) is often referred to as *marginalization*, because we are marginalizing out or summing out a variable. The rule essentially says that if the probability of X is sliced up according to which Y it appears with, then to obtain the marginal probability of X you sum up the probabilities of the slices for all possible outcomes of Y. These "slices" where X and Y occur jointly are what we call *joint probabilities*. The sum rule often appears in various sources

[1] We may note that Bayesians are divided on whether a Bayesian system of reasoning purely corresponds to a personal belief (subjective) or if it is an extension of logic (objective). The Cox postulates is an objectivist approach.

under the name *law of total probability* with the joint probability referred to as *the intersection*($\cap$) of X and Y: $P(X) = \sum_i P(X \cap Y_i)$ [26].

The Product Rule The product rule seen in Equation (4.2) states that the *joint probability* of X and Y is equal to the *the conditional probability* of Y given X, times the stand alone probability of X.

4.2 Bayes' Rule

Next, we can now use Equation (4.1) and Equation (4.2) to derive the theorem at the basis of this study. The joint probability $P(X, Y)$ has the convenient property of symmetry: If X and Y were two hypotheses of events we could easily see that "X and Y are both true" is the same as "Y and X are both true", hence we can state that $P(X, Y) = P(Y, X)$. We exploit this property by applying the product rule Equation (4.2) on both sides of the equation:

$$P(X|Y)P(Y) = P(Y|X)P(X). \tag{4.3}$$

Rearranging Equation (4.3) then gives *Bayes' rule*:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}. \tag{4.4}$$

The left hand side is the conditional probability $P(X|Y)$ of an event X given event Y. The right hand side has the reversed conditional probability $P(Y|X)$ times the fraction of marginal probabilities $\frac{P(X)}{P(Y)}$. We shall say more about the specific terms later, but for now note that in this probability scheme we emphasize the prior probability $P(X)$, which is called prior probability because it takes into account the probability available to us before we consider the outcome $P(Y|X)$.

Bayes' rule is at the very center of Bayesian statistics and encapsulates almost everything pertaining to this field, but just using the rule alone does not necessarily make the analysis Bayesian. The rule is readily applied in a frequentist framework. Frequentists of course do not reject the conditional probability upon which Bayes' theorem is based. The mathematics may be the same for the two frameworks, but the interpretation of the resulting probability being different. However, as the Bayesian interpretation evaluates the probability of an outcome as a degree of belief, the calculation should rely on every quantifiable uncertainty you have about the quantities involved, as it affects your degree of belief. For the sake of example, say we are modelling the probablity that some patient has contracted an infectious disease. A prior probability in such cases would typically be the prevalence of the disease in the population. A frequentist might provide this prevalence as exactly, say 0.01. However, this might be an unreasonable assumption for many real life purposes. It would be hard to discern if it was 0.0101 or 0.01 prevalence, and in the case of disease there are indeed a lot of cases that go unnoticed. In fact, for most experiments, exact prior probabilities about hypotheses are unknown [39], such that in a Bayesian perspective we should impose a prior probability *distribution* onto our problem that express our level of initial uncertainty.

4.2.1 Bayes' Rule in Statistical Inference

Let us now turn our attention to evaluating a model like the regression models from the previous chapters, $\hat{y} = \hat{f}(\mathbf{w}, \mathbf{x})$, with a set of parameters $\mathbf{w}$. Say we would like

to quantify the uncertainty about the quantities of the model. A process of drawing conclusions about quantities that are not directly observable is called *statistical inference* [40]. Typically these quantities are the characteristics of an underlying probability distribution. Within the frequentist notion we would assume a distribution, and thus an uncertainty, on the true target variable y by making an assumption on the theoretical long-run frequency of the data, so the distribution of possible datasets. For example, we could make the assumption that it is normally distributed, $y_i = f(\mathbf{x}_i, \mathbf{w}) + \epsilon$, with $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$. This allows us to write down the conditional probability of the data $p(y_i \mid \mathbf{x}_i, \mathbf{w}, \sigma^2) = \mathcal{N}\left(y_i \mid f(\mathbf{x}_i, \mathbf{w}), \sigma^2\right)$, this is justified in in Section 4.4. We could then formulate confidence intervals and evaluate the quality of the predictions $\hat{y}_i$ in regards to the assumed underlying distribution of y. We may also calculate variance and confidence intervals of the estimated parameters, $Var = (\hat{\mathbf{w}})$, but these quantities are derived from the assumption we made on the data y. This long-run assumption on the data alone implies that the frequentist considers the underlying true parameters $\mathbf{w}$ that we seek to estimate as *fixed* quantities, and only the target variables y are random and expressed by probability distributions.

The Bayesian perspective on the other hand allows *any* quantity, that has an element of uncertainty to it, to be expressed by a probability distribution. We may then instead consider the parameters to be random variables instead of fixed as in the frequentist prospective. In fact, we will often assume both the data *and* the parameters to be random variables in this thesis. We can formulate such an uncertainty on the parameters by the use of the prior probability distribution $p(\mathbf{w})$ from Bayes' rule. In this regard Bayes' rule takes on a new significance in that it allows us to express the strength and weaknesses of our assumptions about the parameters $\mathbf{w}$ *before* we have observed the data. We can subsequently update our knowledge by considering the observed data $\mathcal{D} = (y_0, \ldots, y_n)$ and its effect on the model parameters via the *likelihood* $p(\mathcal{D}|\mathbf{w})$.

Taking all of this into account, Bayes' rule takes the form

Bayes' rule

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}. \tag{4.5}$$

This now allows us to assess the uncertainty about $\mathbf{w}$ *after* observing the data $\mathcal{D}$. Every term in Bayes' rule expressed in this manner has its own designated name and plays a different role in analysis.

- $p(\mathbf{w})$: The prior probability distribution is a distribution over $\mathbf{w}$ and expresses our uncertainty about parameters $\mathbf{w}$ *before* we consider the data $\mathcal{D}$.

- $p(\mathcal{D}|\mathbf{w})$: The likelihood function is the probability of the observed data given different configrations of the parameters $\mathbf{w}$. This is not a probability distribution over $\mathbf{w}$ and as such does not necessarily integrate to unity.

- $p(\mathcal{D})$: The evidence is a normalizing constant that makes sure the posterior integrates to unity, $\int p(\mathbf{w}|\mathcal{D})\mathrm{d}\mathbf{w} = 1$. It is also called the *marginal likelihood*, because we can use equation Equation (4.1) in continuous form, where we marginalize out $\mathbf{w}$ from what may be viewed as a likelihood function (that includes the prior) over

the space of models [26]:

$$p(\mathcal{D}) = \int p(\mathcal{D}, \mathbf{w}) \mathrm{d}\mathbf{w} = \int p(\mathcal{D}|\mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}. \tag{4.6}$$

Here we use the product rule Equation (4.2) in the last equality.

- $p(\mathbf{w}|\mathcal{D})$: The posterior probability distribution is a distribution over $\mathbf{w}$ and expresses our uncertainty about parameters $\mathbf{w}$ *after* we consider the data $\mathcal{D}$.

It should be mentioned that in *Bayesian parameter estimation* Bayes' rule is often written $p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$, that is, without the evidence term Equation (4.6). This is because the evidence does not depend on the parameters, as it is a normalizing constant that can be derived later on if needed. On the other hand when doing *Bayesian model selection* the evidence term will vary with the model, as such it is deemed a crucial part of the procedure. For this reason it is given the name *evidence* [41]. In our study and in the subsequent Bayesian treatment of neural networks we shall see that we are indeed dependent on the evidence, but we shall find clever ways to avoid that dependence as the evidence may be tricky to calculate. Let us wrap up the frequentist and Bayesian comparison in the context of model fitting by stating: In the frequentist perspective, to derive the quantities of inference it is mainly the conditional probability of the data in the likelihood function that is of interest, while in the Bayesian perspective it is the posterior probability distribution, thus the whole of Bayes' rule.

The posterior probability distribution is the centerpiece of Bayesian inference and allows us to further derive appropriate statements of inference, such as point estimates and interval estimates. Additionally we may also integrate out from the posterior components of $\mathbf{w}$, by use of the sum rule in Equation (4.1), in order to focus our interest on certain parameters. Integrating out $\mathbf{w}$ completetly let us focus solely on data predictions as we see shall see in Section 4.7 with the posterior predictive distribution.

4.3 Neural Network as a Probabilistic Model

In chapter Chapter 3 we saw that training a neural network is at its core a procedure of minimising a cost function $\mathcal{C}(\hat{\mathbf{y}}, \mathbf{y})$ with respect to some model parameters $(\mathbf{w}, \mathbf{b}, \lambda)$ given the data $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$. The model parameters include network weights $\mathbf{w}$, biases $\mathbf{b}$ and, depending on the model, various hyperparameters λ. In what follows we will for the sake of brevity incorporate biases $\mathbf{b}$ when we refer to the weights $\mathbf{w}$. The cost function $\mathcal{C}(\hat{\mathbf{y}}, \mathbf{y})$ is some measure of proximity between target values $\mathbf{y}$ and predicted values $\hat{\mathbf{y}}$ produced by the network, or put in another way, the cost function tells us how likely it is that our network produces the true value $\mathbf{y}$ given $\mathbf{w}$ and $\mathbf{x}$. A cost function can thus be seen as tantamount to a likelihood function, or more precisely, a negative-log likelihood:

$$\mathcal{C}(\hat{\mathbf{y}}, \mathbf{y}) \simeq -\ln p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \lambda). \tag{4.7}$$

The perspective here has essentially been turned to interpreting a neural network as a probabilistic model $p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \lambda)$: Where for a given input $\mathbf{x} \in \mathbb{R}^p$ the network assigns a probability $p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \lambda)$ to each possible output y [42].

4.4 Maximum Likelihood Estimation

In the following we define argmax and argmin as the operators that return the argument of the maximum and the minimum of the function, respectively. Minimising the cost

function in Equation (4.7) then is equivalent to minimising the negative-log likelihood, which is a maximum likelihood estimate

$$\underset{\mathbf{w}}{\operatorname{argmax}}\, p\left(\mathbf{y}|\mathbf{x}, \mathbf{w}, \lambda\right) = \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\ln p\left(\mathbf{y}|\mathbf{x}, \mathbf{w}, \lambda\right)\right], \tag{4.8}$$

implying that

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{C}\left(\hat{\mathbf{y}}, \mathbf{y}\right) = \underset{\mathbf{w}}{\operatorname{argmin}} \left[-\ln p\left(\mathbf{y}|\mathbf{x}, \mathbf{w}, \lambda\right)\right] = \mathbf{w}_{\text{MLE}}. \tag{4.9}$$

The takeaway from this is that training a neural network can be considered *maximum likelihood estimation* (MLE) of the weights.

We shall now return to linear regression, which we covered in Section 2.2, and show that the probabilistic interpretation does reproduce the least squares estimator $\mathbf{w}_{\text{OLS}}$ as we would expect as a consequence of Equation (4.9). The purpose of the following discussion is to establish an analogy with neural networks. We noted in Equation (3.8) that linear regression is equivalent to a neural network with no hidden layers and with the identity activation function, so the point of the following discussion is to study simple linear regression examples and transfer the intuition to neural networks.

Consider once again a data set of n points abbreviated as $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ and the relationship of each single target y_i and feature example $\mathbf{x}_i$. We assume as before a linear relationship between input and outputs

$$y_i = \mathbf{x}_i^\intercal \mathbf{w} + \epsilon, \tag{4.10}$$

where ϵ is Gaussian noise $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$. The latter is the crux of the argument, as it implies that our target y_i is a Gaussian random variable. We can show that the expected value and variance are, respectively

$$\begin{aligned} \mathbb{E}\left[y_i\right] &= \mathbb{E}\left[\mathbf{x}_i^\intercal \mathbf{w} + \epsilon\right] = \mathbf{x}_i^\intercal \mathbf{w} \\ \operatorname{Var}\left[y_i\right] &= \operatorname{Var}\left[\mathbf{x}_i^\intercal \mathbf{w} + \epsilon\right] = \sigma^2, \end{aligned} \tag{4.11}$$

and thus it follows that the conditional probability of g_i is

Conditional Probability

$$p(y_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) = \mathcal{N}\left(y_i|\mathbf{x}_i^\intercal \mathbf{w}, \sigma^2\right). \tag{4.12}$$

Explicit formula for $\mathcal{N}(.)$ in Chapter B. We now consider all the training data $\mathcal{D}$ and assume that the datapoints of $\mathbf{x}$ are drawn independently. This then yields an expression for the combined *likelihood function*:

Likelihood function

$$p(\mathcal{D}|\mathbf{w}, \sigma^2) = \prod_{n=1}^{n} \mathcal{N}\left(y_i|\mathbf{x}_i^\intercal \mathbf{w}, \sigma^2\right). \tag{4.13}$$

We want to obtain the MLE, thus we do as in Equation (4.8) and take the negative

log-likelihood

$$\begin{aligned} -\ln p(\mathcal{D}|\mathbf{w},\sigma^2) &= -\sum_{i=1}^{n} \ln \mathcal{N}\left(y_i|\mathbf{x}_i^\intercal \mathbf{w},\sigma^2\right) \\ &= \frac{n}{2}\ln\sigma^2 + \frac{n}{2}\ln(2\pi) + \frac{1}{2}\sum_{i=1}^{n}\left(\frac{y_i - \mathbf{x}_i^\intercal \mathbf{w}}{\sigma}\right)^2. \end{aligned} \tag{4.14}$$

We convert the last expression into matrix notation using that, $\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i^\intercal \mathbf{w}\right)^2 = (\mathbf{y}-\mathbf{X}\mathbf{w})^\intercal(\mathbf{y}-\mathbf{X}\mathbf{w})$ and then we take the gradient of this expression with respect to $\mathbf{w}$ and set it equal to zero

$$\begin{aligned} \nabla_{\mathbf{w}}\,(\mathbf{y}-\mathbf{X}\mathbf{w})^\intercal(\mathbf{y}-\mathbf{X}\mathbf{w}) &= 0 \\ \Rightarrow \mathbf{X}^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\mathbf{w}) &= 0. \end{aligned} \tag{4.15}$$

Solving this expression for $\mathbf{w}$ gives

$$\mathbf{w}_{\mathrm{MLE}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \tag{4.16}$$

This is the exact same expression as the one we obtain in Equation (2.4) by minimizing the mean squared error (MSE) cost function $\mathcal{C}_{\mathrm{MSE}} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \mathbf{x}_i^\intercal \mathbf{w}\right)^2$. We thus conclude that with normally distributed likelihood $\mathbf{w}_{\mathrm{MLE}} = \mathbf{w}_{\mathrm{OLS}}$.

4.5 Central Tendencies

Before we continue on the thread of maximum estimation we should elaborate on the three most commonly used central tendencies in statistical inference, namely the mode, the mean and the median. The *mode* is for the discrete case the most frequent element in a list of samples, so in $[1,2,2,4,6]$ that would simply be 2. In terms of a probability distribution it would be the value m where $p(X = m)$ is at its maximum. This is simply the MLE estimate we encountered in Section 4.4. Any function can have multiple maxima, thus we may have multiple modes. The *median* for the discrete case is the value with equal number of elements on both sides, so for our previous example that would be median = 2. For a probability distribution it is the value that satisfies

$$p(X \leq \text{median}) \geq \frac{1}{2} \quad \text{and} \quad p(X \geq \text{median}) \geq \frac{1}{2}. \tag{4.17}$$

Lastly the *arithmetic mean*, or just the mean, is in the discrete case probably already too familiar to be stated, but for a continuous probability distribution it is the expected value

$$\mathbb{E}(x) = \int_{\infty}^{\infty} x f(x) dx. \tag{4.18}$$

We will in fact cover additional central tendencies, based of the median and the mean in Chapter 6

4.6 Regularized Least Squares and Maximum A Priori Estimation

If we include a *regularization* on the weights as introduced in Section 2.5 we can interpret the optimization procedure as a *maximum a priori estimation* (MAP), which is the same as MLE but with a prior. In this case the regularization parameter λ can be viewed as proportional to the logarithm of a prior weight distribution. As with with

the previous case of MLE, we shall now show that a probabilistic approach reproduces the regularized estimator, namely the *Ridge* estimator. In Equation (4.13) we presented a likelihood distribution for the data with a Gaussian error, we shall now introduce a prior probability distribution, $p(\mathbf{w})$, over the weights $\mathbf{w}$. We pick a basic n-dimensional *Gaussian prior* that simplifies the treatment but that hits the point home

Normal Prior with Zero Mean

$$p(\mathbf{w}|\tau^2) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \tau^2\mathbf{I}\right). \tag{4.19}$$

The prior has a mean of zero and a diagonal covariance matrix with constant variance τ^2 along the diagonal. Note that we here have a multivariate normal distribution over *all* parameters. The normal distribution is a convenient prior as its a *conjugate prior* with respect to the likelihood, meaning that the product of the prior and the likelihood, $p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$, will produce a distribution of the same type as the prior, specifically the posterior probability distribution will be a Gaussian distribution over the weights,

Normal Posterior Distribution

$$p(\mathbf{w}|\mathcal{D}) = \mathcal{N}\left(\mathbf{w}|\mathbf{m}, \mathbf{S}\right), \tag{4.20}$$

where

$$\begin{aligned} \mathbf{m} &= \frac{\mathbf{S}\mathbf{X}^\intercal\mathbf{y}}{\sigma^2}, \\ \mathbf{S}^{-1} &= \frac{\mathbf{I}}{\tau^2} + \frac{\mathbf{X}^T\mathbf{X}}{\sigma^2}. \end{aligned} \tag{4.21}$$

We refer to Murphy's book [26] for the details of this derivation. Involving the use of the relations given in Chapter B and gathering up the terms to formulate a new Gaussian. Taking the logarithm of the posterior in Equation (4.20) gives

$$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{1}{2}\frac{(\mathbf{y}-\mathbf{x}\mathbf{w})^\intercal(\mathbf{y}-\mathbf{X}\mathbf{w})}{\sigma^2} - \frac{\mathbf{w}^{\mathrm{T}}\mathbf{w}}{2\tau^2} + \text{ const.} \tag{4.22}$$

By maximizing the log-posterior we obtain

$$\begin{aligned} \nabla_{\mathbf{w}}\left[\frac{(\mathbf{y}-\mathbf{x}\mathbf{w})^\intercal(\mathbf{y}-\mathbf{X}\mathbf{w})}{\sigma^2} + \frac{\mathbf{w}^{\mathrm{T}}\mathbf{w}}{\tau^2}\right] &= 0 \\ \Rightarrow -\frac{\mathbf{X}^{\mathrm{T}}(\mathbf{y}-\mathbf{X}\mathbf{w})}{\sigma^2} + \frac{\mathbf{w}}{\tau^2} &= 0. \end{aligned} \tag{4.23}$$

Solving for $\mathbf{w}$ in Equation (4.23) then gives

$$\mathbf{w}_{\mathrm{MAP}} = \left(\mathbf{X}^\intercal\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^\intercal\mathbf{y}\ ,\ \lambda = \frac{\sigma^2}{\tau^2}, \tag{4.24}$$

which is the Ridge regression estimator we encountered in Equation (2.13). Note that in this case the procedure is equivalent to maximizing the log-likelihood and log-prior individually, and then summed [26]:

$$\mathbf{w}_{\mathrm{MAP}} = \arg\max_{\mathbf{w}}[\log p(\mathcal{D}|\mathbf{w})] + \arg\max_{\mathbf{w}}[\log p(\mathbf{w})]. \tag{4.25}$$

The cost function for ridge regression is

$$\mathcal{C}_{\text{ridge}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^{\intercal}\mathbf{w})^2 + \lambda \mathbf{w}^{\mathrm{T}}\mathbf{w}, \tag{4.26}$$

minimizing it, $\nabla_{\mathbf{w}}\, \mathcal{C}_{\text{ridge}} = 0$, gives exactly as in Equation (4.24). Thus we conclude that $\mathbf{w}_{\text{MAP}} = \mathbf{w}_{\text{ridge}}$.

Comparing the MAP procedure result in Equation (4.24), with the MLE in Equation (4.16), we see that by including a prior $p(\mathbf{w})$ we effectively include a regularization on the optimal weights of the type $\lambda = \frac{\sigma^2}{\tau^2}$. To be specific, as the posterior in this case is a normal distribution the mode, median and the mean are the same. To be clear, the MAP estimate *is* the mode of the distribution, and thus we have that $\mathbf{w}_{\text{MAP}} = \mathbf{m}$.

In Section 2.5 we introduced the concepts overfitting and underfitting. We shall now see how we can interpret the regularization parameter in a probablistic context. Say we have $\mathcal{D}_{train} = (\mathbf{X}_{train}, \mathbf{y}_{\text{train}})$, and we seek to predict the out-of-sample data points $\mathcal{D}_{test} = (\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.

Overfitting If we had some prior knowledge regarding the weights best suited to predict $\mathbf{y}_{\text{test}}$ we could introduce this knowledge through the prior. If we were very uncertain about our prior belief regarding the value of the weights relative to the new knowledge from the likelihood, i.e. a large variance $\tau^2 \gg \sigma^2$, our prior knowledge would essentially not guide us in our choice of weights and effectively provide no regularization effect on the weights $\lambda \approx 0$. The mean of the posterior would just be equal to the maximum likelihood estimation, that is, $\mathbf{m} = \mathbf{w}_{\text{OLS/MLE}}$. So if our model was overfitting, our very broad prior distribution would not introduce a sufficiently large statistical bias to reduce our prediction error on $\mathbf{y}_{\text{test}}$. We could say that the data overwhelms the prior [37].

Underfitting On the other hand if we were very certain (correctly or not) about our prior knowledge on the optimal weights, $\tau^2 \ll \sigma^2$, this would dominate our new more uncertain knowledge arising from the likelihood function. Attempts at training the model on the uncertain (or noisy) data would be futile and we would fail to capture the regularities of $\mathbf{y}_{\text{train}}$. In other words, by introducing a very large statistical bias through regularization we would force our model to underfit on the training data.

There is a bias-variance trade-off at play here, meaning that there is often an optimal trade-off to be found in-between the two scenarios described above. In a Bayesian model, choosing a reasonable prior *can* lower the generalization error through the regularization effect that λ has on the weights. This is especially true as the complexity of a model increases, and this is a general concept that is important in the context of neural networks. In that regard we have shown that Bayesian models naturally regularize through the effect of the prior.

4.7 Posterior Predictive Distribution

In order to make predictions $\hat{y}_i$ for new values $\mathbf{x}_i$, where $\mathbf{x}_i \in \mathbf{X}_{test}$, we formulate the *posterior predictive distribution*

Posterior Predictive Distribution

$$\begin{aligned} p(\hat{y}_i|\mathbf{x}_i) &= \int p(\hat{y}_i|\mathbf{x}_i, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) \mathrm{d}\mathbf{w} \\ &= \mathbb{E}_{p(\mathbf{w}|\mathcal{D})}[p(\hat{y}_i|\mathbf{x}_i, \mathbf{w})]. \end{aligned} \tag{4.27}$$

The first factor in the integrand is the conditional distribution for the target variable in terms of the features and the model weights. We encountered this factor Equation (4.12) when constructing the likelihood. The second factor of the intergrand is the posterior distribution of the model weights given the data.

MAP-learning Now, we could just plug in $\mathbf{w}_{\text{MAP}}$ into Equation (4.27) which is reminiscent of the prediction approach of linear regression in Equation (2.10). This is a MAP-learning approach called *plug-in approximation* which is often used because of its simplicity [37]. It forces the distribution to become a delta function that is centered on the MAP estimate, $p(\mathbf{w}_{\text{MAP}} \mid \text{D}) = \delta(\mathbf{w} - \mathbf{w}_{\text{MAP}})$. The rational behind this approach is as follows: As we sample more data examples the posterior distribution should get narrower, approaching the delta function described above. Thus if we have a large number of data examples the approximation is appropriate. With the plug-in approximation our posterior predictive distribution would simply become

$$\begin{aligned} p(\hat{y}_i|\mathbf{x}_i) &= \int p(\hat{y}_i|\mathbf{x}_i, \mathbf{w}) \delta(\mathbf{w} - \mathbf{w}_{\text{MAP}}) \mathrm{d}\mathbf{w} \\ &= p(\hat{y}_i|\mathbf{x}_i, \mathbf{w}_{\text{MAP}}). \end{aligned} \tag{4.28}$$

The procedure is computationally appealing as we avoid calculating the evidence term of Bayes' rule, which is notoriously tricky as we shall see in the next chapter. That said, the procedure is not what is considered a fully Bayesian approach. This is due to the fact that it does not provide any measure of uncertainty on the prediction from the model parameters $\mathbf{w}$. The uncertainty comes from the data through the conditional distribution, and is thus more akin to a frequentist approach. The procedure runs the risk of giving overconfident predictions, in other words, it may more easily overfit. Also, in the case of an asymmetrical distribution the mode, median, and mean are not equal, and the mode, and thus the MAP, is for many such asymmetrical cases a bad estimate of the central tendency.

Bayesian Approach For a fully Bayesian approach to the posterior predictive distribution in Equation (4.27) we must take the posterior-weighted average over the conditional distribution of the data predictions. Taking the expected value weighted by the posterior is notationally reflected in the second equation in Equation (4.27). So instead of using a point estimate we will need the full expression for the posterior probability distribution.

Conveniently we defined a posterior in our previous example of Bayesian linear regression in Equation (4.20). Using this example, the predictive distribution results from a convolution of the conditional probability and the posterior

$$\begin{aligned} p(\hat{y}_i|\mathbf{x}_i) &= \int p(\hat{y}_i|\mathbf{x}_i, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) \mathrm{d}\mathbf{w} = \int \mathcal{N}\left(\hat{y}_i|\mathbf{x}_i^\intercal \mathbf{w}, \sigma^2\right) \mathcal{N}\left(\mathbf{w}|\mathbf{m}, \mathbf{S}\right) \mathrm{d}\mathbf{w} \\ &= \mathcal{N}\left(\hat{y}_i \mid \mathbf{m}^\intercal \mathbf{x}_i, \alpha^2\right). \end{aligned} \tag{4.29}$$

Here the variance is given by (See derivation in [26])

$$\alpha^2(\mathbf{x}_i) = \sigma^2 + \mathbf{x}_i^\intercal \mathbf{S} \mathbf{x}_i. \tag{4.30}$$

We see from Equation (4.29) that we once again end up with a normal distribution. Thus using conjugate forms of the distributions involved, simplifies calculations.

We may add here that we have assumed that variances σ^2 and τ^2 were known quantities, but often it is unreasonable to assume that we are certain about these as well. In that case we could impose a prior probability distribution on the variances, called a hyperprior. If we were indeed uncertain about the variances we would have to integrate over all possible values of the variances for the procedure to be a fully Bayesian procedure, strictly speaking. This type of approach is called *hierarchical modeling*. In this thesis we shall rather treat variances related to the prior as a neural network hyperparameter that we tune by trial and error, with the caveat of possible underestimation of the uncertainty.

The Evidence In the discussion above we used that $p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$, and avoided calculating the evidence term $p(\mathcal{D})$. We had a *fixed model* (Gaussian distribution, number of parameters) where the normalization constant ensures that the posterior distribution is a valid probability density that is known. The normalization condition gave us Equation (4.6), which for our Bayesian linear regression example is a simple calculation after we have found the posterior,

$$\int p(\mathcal{D} \mid \mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w} = \int \mathcal{N}\left(\mathbf{w}|\mathbf{m}, \mathbf{S}\right) \mathrm{d}\mathbf{w} = 1. \tag{4.31}$$

This comes from the unsurprising result that the normal distribution is normalized. So in such cases the evidence is harmless. For most problems calculating the evidence is an ardous undertaking, as we shall cover more thoroughly in the next chapter. In some cases we can avoid calculating it, such as when finding central tendencies in the posterior, commonly the mode (MLE/MAP), the mean and the median. However, when predicting new data based on the posterior predictive distribution, and also seeking to quantify the uncertainty on those predictions, we have to make sure we are dealing with a valid probability density. So in summary there is potentially two troublesome, often multivariate, integrals that need to be solved, and without any simplifying assumption the problem which will confront us in the next chapter is essensially

$$\begin{aligned} p(\hat{y}_i|\mathbf{x}_i) &= \int p(\hat{y}_i|\mathbf{x}_i, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) \mathrm{d}\mathbf{w} \\ &= \int p(\hat{y}_i|\mathbf{x}_i, \mathbf{w}) \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{\int p(\mathcal{D}|\mathbf{w}')p(\mathbf{w}')\mathrm{d}\mathbf{w}'} \mathrm{d}\mathbf{w}. \end{aligned} \tag{4.32}$$

4.8 Bayesian Model Selection

In practice we may want to evaluate several models defined by different distributions, number of parameters, hyperparameters etc. We can consider a predictive distribution as one model instance, $p\left(\hat{y} \mid \mathbf{x}, \mathcal{M}_j\right)$, in a set of models $\{\mathcal{M}_j\}_{j=1}^L$. We can then average over them in a Bayesian fashion by

$$p(\hat{y} \mid \mathbf{x}, \mathcal{D}) = \sum_{j=1}^{L} p\left(\hat{y} \mid \mathbf{x}, \mathcal{M}_j, \mathcal{D}\right) p\left(\mathcal{M}_j \mid \mathcal{D}\right). \tag{4.33}$$

This is called Bayesian model averaging (BMA) and is a posterior predictive *mixture* distribution of every model's posterior predictive distribution weighted by the posterior distributions of the models. The BMA would introduce an uncertainty on the selection process itself and also give most weight to the most probable model.

A simpler method is to pick the model with the largest model evidence. We can write the posterior distribution over models as $p(\mathcal{M}_i \mid \mathcal{D}) \propto p(\mathcal{M}_i)\, p(\mathcal{D} \mid \mathcal{M}_i)$, and *Bayes factor* between two models $\mathcal{M}_1$ and $\mathcal{M}_2$ is

$$K = \frac{p(\mathcal{D}|\mathcal{M}_1)}{p(\mathcal{D}|\mathcal{M}_2)} \tag{4.34}$$

where we choose $\mathcal{M}_1$ if $K > 1$ and vice versa. Bayesian model selection methods are very attractive, as you do not need to forgo any data for validation and testing.

Frequentist Validation In this thesis we shall not apply Bayesian model selection techniques to choose the hyperparameters of a Bayesian neural network. Instead we shall use validation methods as presented in Chapter 2, where we reserve parts of the data for model selection. The reason is that the evidence yields the probability of the data *given* the model, $p(\mathcal{D} \mid \mathcal{M}_i)$, meaning that if the model is mis-specified the reliability of the evidence is in jeopardy. The method of variational inference that we shall use, does not necessarily give the correct functional form of the true posterior distribution (This will become clear in Chapter 5). Model validation on an out-of-sample data set gives estimates of the probability of the data regardless of the validity of the model assumptions. Its main necessity is that our data set is large enough and that the validation data is sampled from the same population as the training data. These are necessities that we can fulfill, so the choice falls on frequentist validation.

4.9 Prediction Interval

A credible interval is the Bayesian quantity analogous to confidence intervals in frequentist statistics. However, the interpretation is different between the two and follows the reasoning of Section 4.2.1 where the confidence interval is dependent on the assumption of the long run frequency, where the credible interval gives a degree of confidence given the prior and the observed data. "Credible interval" is usually used to refer to the interval about a parameter, that is, an interval of the posterior probability distribution. We will focus more on the posterior predictive distribution, that is, the uncertainty about a prediction $\hat{y}$. This type of credible interval is often called a *prediction interval*, and is what we shall refer to it as. There are several ways of defining an interval about a distribution. In this thesis we shall use the *equal tailed interval* (ETI) for analysis. This is, as its name suggests, an interval that has the same probability mass on either tail, $\mathrm{I} = [q_{\alpha/2}, q_{1-\alpha/2}]$, where α is the confidence level, and q is the quantiles. A popular choice is the 95% ETI, so $\mathrm{I}_{0.05} = [q_{0.025}, q_{0.975}]$. These can be found by formulating the *quantile function*, however we will just work with samples of probability distribution and use `Python` programming libraries to find the percentiles of interest, mainly the 2.5th and 97.5th percentile for 95% ETI.

Chapter 5

Bayesian Neural Network

In Chapter 3 we expanded upon the concept of regression from Chapter 2 and defined the dense neural network, which was also a deterministic neural network (DNN[1].). Now we seek to combine the Bayesian approach of Chapter 4 with the DNN in order to create a *Bayesian neural network* (BNN). The Bayesian linear regression we introduced in Chapter 4 is equivalent to a BNN with no hidden layers and with identity activation. Our goal is to create a deep BNN with an arbitrary number of hidden layers with respective non-linear activation functions.

For the purpose of pattern recognition a DNN is a powerful tool suited for a large variety of different problems. Despite its appeal, it only provides point estimates of targets and does not yield a reliable way to gauge the uncertainty of a given model. Knowing the uncertainty of a prediction is of key importance in many problems. Apart from being essential for comparison to experiments in physics, one may imagine a practical example of a bioscientific nature whereby a neural network has been trained to recognize malignant tumors in images from cancer patients. A DNN[2] may do an excellent job of this task, however, if a new test image for some reason looks different from the images of the training data, the network will still make a prediction even though it may not "know" how to handle the new type of data. This may lead to overconfident predictions which can have detrimental effects, such as classifying a patient with cancer as not having cancer, for example because the patient moved during imaging[3]. A researcher may try to reproduce such real world noise through the technique of *adversarial attacks*, where attemps are made to deceive the model by carefully crafting noise, and tune it thereafter to account for and defend against such noise. Alternatively, one may train a network to be "aware" of its uncertainty such that one may discard highly uncertain predictions. This creates the motivation for a Bayesian approach to neural networks.

By introducing stochasticity through prior distributions on the weights (and usually also the likelihood), we may promote the weights of a network from single valued quantities to the components of a posterior probability distribution. As a consequence of this, Bayesian deep learning, of the type we shall study here, is about learning distributions over parameters rather than point estimates.

[1]DNN is often used to abbreviate deep neural network, which also includes multilayered Bayesian neural networks. By DNN we will mean, as pointed out in Chapter 3, a *deterministic* neural network

[2]For this type of task, a convolutional neural network (CNN) is especially suited.

[3]For further reading on the sensitivity of neural networks in imaging we refer to [43].

5.1 Bayesian Feed-Forward Propagation

We shall now illustrate a simple example of how we can naively setup a feed-forward algorithm in a Bayesian framework and study the challenges that this entails. For now we will keep the variances of our distributions constant, but eventually we will also consider these as trainable weights.

Feed-forward propagation example Let us once again define a regression problem with additive Gaussian noise ϵ,

$$\hat{y}_i = y(\mathbf{x}_i, \mathbf{w}) + \epsilon. \tag{5.1}$$

This time we will for the purpose of illustration include one hidden layer with a *non-linear* activation. So in accordance with the Feed-Forward formula the output layer is

$$\begin{aligned} y(\mathbf{x}_i, \mathbf{w}) &= \sum_{j=1}^{N^{(1)}} w_{ij}^{(2)} f^{(1)} \left(\sum_{k=1}^{N^{(2)}} w_{jk}^{(1)} x_k + b_j^{(1)} \right) + b_i^{(2)} \\ &= \mathbf{W}^{(2)T} f(\mathbf{W}^{(1)T} \mathbf{x}_i + b^{(1)}) + b^{(2)}, \end{aligned} \tag{5.2}$$

which is similar to the feed-forward Equation (3.6). The only difference is that the outer activation has been turned into the identity (output) activation, as we only have one hidden layer. Notation-wise we use $\mathbf{W}^{(l)}$ to signify the matrix of weights between two layers, while we use the vector $\mathbf{w}$ to signify *all* the weights of the model. As a consequence of the Gaussian noise we obtain the same form for the likelihood as in Section 4.4

$$p(\hat{y}_i | y(\mathbf{x}_i, \mathbf{w}), \sigma^2) = \mathcal{N}\left(\hat{y}_i | y(\mathbf{x}_i, \mathbf{w}), \sigma^2\right). \tag{5.3}$$

We continue with the adoption of previous expressions and use the same basic prior on the weights as in 4.6, with zero mean and a diagonal covariance matrix

$$p(\mathbf{w}|\tau^2) = \mathcal{N}\left(\mathbf{w}|\mathbf{0}, \tau^2\mathbf{I}\right). \tag{5.4}$$

For n observations of training examples $\mathcal{D} = \{\mathbf{x}_i, y_i\}_i^n$ we have the likelihood

$$p(\mathcal{D}|\mathbf{w}, \sigma^2) = \prod_{i=1}^{n} \mathcal{N}\left(\hat{y}_i | y(\mathbf{x}_i, \mathbf{w}), \sigma^2\right). \tag{5.5}$$

With these expressions we have the necessary components for a forward pass (see Section 3.1 on forward propagation) of a BNN, with the un-normalized posterior being

$$p(\mathbf{w}|\mathcal{D}, \sigma^2, \tau^2) \propto p(\mathcal{D}|\mathbf{w}, \sigma^2) p(\mathbf{w}|\tau^2). \tag{5.6}$$

As promised, the posterior is a probability distribution over the weights of the network.

Unfortunately, unlike the Bayesian linear regression example of the previous chapter, the posterior in Equation (5.6) is non-Gaussian. If we investigate the argument, that is, the logarithm of the posterior, this should become apparent,

$$\begin{aligned} \log p(\mathbf{w}|\mathcal{D}, \sigma^2, \tau^2) &\propto \log\{\prod_{i=1}^{n} \mathcal{N}\left(y_i | \hat{y}(\mathbf{x}_i, \mathbf{w}), \sigma^2\right)\} + \log\{\mathcal{N}\left(\mathbf{w}|\mathbf{0}, \tau^2\mathbf{I}\right)\} \\ &= -\frac{1}{2\sigma^2} \sum_{i=1}^{n} \left\{\hat{y}\left(\mathbf{x}_i, \mathbf{w}\right) - y_i\right\}^2 - \frac{1}{2\tau^2} \mathbf{w}^\intercal \mathbf{w} + \text{const.} \end{aligned} \tag{5.7}$$

For linear regression the model had the form $\hat{y}(\mathbf{x}_i, \mathbf{w}) = \mathbf{x}_i^\intercal \mathbf{w}$. Now we have that the model is the output of a neural network with one hidden layer, $\hat{y}(\mathbf{x}_i, \mathbf{w}) = \mathbb{NN}(\mathbf{x}_i, \mathbf{w})_{L_h=1}$. We insert the neural network output Equation (5.2) into the unnormalized posterior argument (see Equation (5.7)), and relabel the right hand side of Equation (5.7) as a cost function $\mathcal{C}(\mathbf{w})$, so that $\mathcal{C}(\mathbf{w}) \propto -\log p(\mathbf{w}|\mathcal{D}, \sigma^2, \tau^2)$. We thus obtain for the cost function

$$\mathcal{C}(\mathbf{w}) = \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left\{ \mathbf{W}^{(2)_T} f\left(\mathbf{W}^{(1)_T} \mathbf{x} + b^{(1)} \right) + b^{(2)} - y_i \right\}^2 + \frac{1}{2\tau^2} \mathbf{w}^\intercal \mathbf{w} + \text{const.} \tag{5.8}$$

Including the evidence of Bayes' rule, we then have the negative log posterior as

$$-\log p(\mathbf{w}|\mathcal{D}, \sigma^2, \tau^2) = \mathcal{C}(\mathbf{w}) + \log \left(\int \mathcal{C}(\mathbf{w}) \mathrm{d}w \right) + \text{const.} \tag{5.9}$$

Defining $Z = \exp(\text{const}) \int \mathcal{C}(\mathbf{w}) \mathrm{d}w$, we can write Equation (5.9) more succinctly as

$$\hat{p}(\mathbf{w}|\mathcal{D}, \sigma^2, \tau^2) = \frac{1}{Z} \exp(-\mathcal{C}(\mathbf{w})). \tag{5.10}$$

Consider again Equation (5.8). The feed-forward procedure, without activation functions, produces non-linearity naturally, in that the product of successive layers creates polynomial expressions. Additionally, recall that the activation function $f(.)$ is a non-linear function, such as those in Table 3.2. There are thus two sources of non-linearity, and we therefor conclude that we are far from having a squared dependence on $\mathbf{w}$ in (see Equation (5.8)), so it should be clear, that unlike the Bayesian linear regression case we can not rearrange these expressions to produce a Gaussian argument. Calculating the posterior, and especially the model evidence part of Z, easily becomes an ardous task. For a deep BNN model $\mathbb{BNN}(\mathbf{x}_i, \mathbf{w})_{L_h=N_L}$, where $N_L > 1$, the degree of non-linearity increases as a function of layers and the posterior becomes even more estranged to a conjugate relationship to the prior as we saw in the previous chapter.

Intractability We have established that for a multilayered BNN the posterior has a highly non-linear dependence on the weights. This is problematic, as to do standard Bayesian inference we wish to marginalize out the weights of the posterior distribution to obtain the model evidence and the predictive posterior distribution, as in Equation (4.32). Without a closed form solution, the marginalizing procedure will genereally be an intractable problem, meaning that there is no efficient algorithm in which to solve the problem within reasonable time frames, that is, without restorting to an approximation [42].

Let us study the marginalization procedure to obtain the model evidence $p(\mathcal{D}|\mathbf{w}) = \int p(\mathcal{D}|\mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}$. The integrand is no longer a normalized distribution as in Chapter 4, so there is no shortcut to solve the integral for an arbitrary network of non-linear activations. The weight term $\mathbf{w}$ is a vector containing all the network's parameters, such that the integral is over all possible combinations of the parameters. This does not bode well for a tractable solution. We illustrate the intractability issue with a simple example in 1.

Example 1: Intractable Evidence

Let us imagine that every weight is a binary quantity that can only take on one of two values, say $w_i \in \{0, 1\}$. The discreteness of the weights implies our evidence-integral is now a sum $\sum_w p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$. We construct a BNN architecture that has one input, two hidden layers with five nodes each, and one output,

$$[n_x^{(0)}, n_h^{(1)}, n_h^{(2)}, n_y^{(3)}] = [1, 5, 5, 1].$$

The number of trainable weights in a BNN, ignoring biases, is

$$\begin{aligned} N_w &= (n_x \cdot n_h + n_h \cdot n_h + n_y \cdot n_y) \cdot 2 \\ &= (1 \cdot 5 + 5 \cdot 5 + 5 \cdot 1) \cdot 2 = 70 \end{aligned}$$

Ignoring the biases, the number of binary permutations is

$$2^{N_w} = 2^{70} \approx 10^{21}.$$

Which is the number of possible configurations of $\mathbf{w}$ the integral in the evidence term, or the summation in this case, would have to sum over. In a BNN the weights can in theory take on *any* real number, not just two.

With example 1 considered, we can firmly state that the integral in the evidence is in general indeed intractable, and thus the posterior as a whole. The totality of the posterior predictive distribution must be approximated in a different manner.

5.2 Bayesian Neural Network Methods

The most precise approach to obtaining a posterior approximation is to use a clever sampling technique, for example one in the family of *Markov Chain Monte Carlo*(MCMC) methods, specifically Hamiltonian MCMC is one of the most powerful choices of sampling algorithms for high dimensional posteriors. But even with posterior sampling methods the computational cost usually scales out of control as a function of the number of parameters in the BNN [44].

Several methods have been developed to produce approximate posterior weight distributions or direct approximations to the predictive distribution. These methods are often more favourable in terms of scalability, but vary in terms of prediction accuracy and reliability of uncertainty estimates. We list here several of the approaches to constructing a Bayesian (or Bayesian-like) neural network.

- **Bayes with Dropout**
  Dropout is a popular regularization technique employed in DNNs to reduce overfitting. The method involves doing what its name suggest, dropping out weights of the network at random during training. Gal and Ghahramani proposed the extension of its use for the evaluation of predictive uncertainty. The dropout technique produces a stochastic variation in the weight combinations for every forward pass, but at test time. This method is simple to implement, but has garnered criticism for inducing an improper posterior distribution (cannot be normalized) among other issues, and hence it is not seen as a truly Bayesian technique [45].

- **Deep Ensembles**
  Deep Ensembles is the method of aggregating the estimates of multiple individually trained neural networks, each with a variation in its initial weight settings [46]. The distributions of the initial weight settings is deemed the prior distribution. This technique has been challenged on whether it is truly a Bayesian method, as the members of the ensemble may all collapse on the same hypothesis and fail to produce the diversity required. However, deep ensembles remains an active subject of research, and recently developed methods have, according to the authors, a Bayesian nature and reliable predictive uncertainties [47].

- **Laplace Approximation**
  This method involves finding a MAP estimate of the posterior through numerical optimization, and then build a approximate local Gaussian posterior about the estimate. The covariance matrix is estimated by evaluating the Hessian of the sum of squared error functions, and it is therefore a second order optimization method. Laplace approximation is an "old" approach to BNNs and is found in several textbooks. It has recently got some traction when combined with Kronecker-factored Approximate Curvature [48].

- **Stochastic Gradient MCMC**
  Hamiltonian Monte Carlo (HMC) involves treating the sampling of a posterior as a physics problem, where kinetic and potential energies are defined and a Hamiltonian dynamical system is simulated. The posterior distribution is treated as a potential well such that new samples of the posterior tend towards an area of highest density/low potential. This involves calculating the gradient of a potential function, which can be very costly for large parameter spaces. Recent developments of these sampling techniques seek to improve scalability, such as by combining HMC with various stochastic gradient descent methods [49].

- **SWAG**
  Stochastic Weight Averaging of Gaussian samples (SWAG) is a method that utilizes the stochastic gradient descent method (see Section 3.4) as the provider of uncertainty. It uses the fact that a gradient trajectory will traverse the weight space of a posterior on its way to the optimal weights. One can capture the local geometry of the posterior by collecting SGD weight samples and averaging at some frequency. With the SWAG method one may produce a approximate Bayesian model [50].

- **BNN with Gaussian Processes**
  This method connects Bayesian feed-forward neural networks with Gaussian Processes (GP). It replaces the procedure of backpropagation with the calculation of a GP posterior. The motivation for using this technique is that there exists efficient quantum matrix inversion algorithms that can be efficiently applied to a GP. Hence this is a BNN method apt for quantum computing [51].

- **Variational Inference**
  Finally, we have variational inference which will be the main focus of our study and the topic for the subsequent sections.

5.3 Variational Inference

Variational inference is a group of approximation techniques applying ideas coming from the mathematical technique of *calculus of variations* to Bayesian inference, typically to

find an approximation of an intractable posterior distribution. Specifically, we seek to obtain a distribution $q(\mathbf{w}|\boldsymbol{\theta})$ in which $q(\mathbf{w}|\boldsymbol{\theta}) \approx p(\mathbf{w}|\mathcal{D})$, for the set of weights dependent upon some training data $\mathcal{D}$. Additionally we have the set of adjustable variational parameters $\boldsymbol{\theta}$.

Variational principles are commonly encountered within physics as the working mechanism behind Lagrangian and Hamiltonian mechanics. Interestingly, we shall later on use variational principles to conduct statistical inference about data created by the use of variational principles. However, this curiosity is probably no more suprising than encountering addition and subtraction at different levels of a problem, as nature itself obeys variational principles [52]. We already assumed in Chapter 1 that the reader is familiar with quantum field theory, so variational principles should be familiar. In the subsequent sections will make the case for a specific *functional*, such as the action in physics, but for the purpose of finding a surrogate posterior $q(\mathbf{w}|\boldsymbol{\theta})$. Next, we will explore the concept of *entropy* as applied in information theory.

5.3.1 Entropy

Let us now take a brief tour through the realm of information theory in order to build an idea of what functional is appropriate in the context of minimizing the dissimilarity between probability distributions. Specifically, we shall look at the concept of entropy as first applied to information theory by *Claude Shannon* in his seminal paper *A Mathematical Theory of Communication* [53]. Entropy is a familiar concept from physics and has its origins in the field of thermodynamics, where 18th and 19th century physicists sought to formalize the phenomenon of energy being lost to dissipation in physical processes (2nd law of thermodynamics). A probabilistic formulation of entropy was devised by *Ludwig Boltzmann*, which laid the foundation for the field of statistical mechanics. Of most relevance to information theory is the more generalized version of entropy by *Josiah Willard Gibbs* [54].

The Multiplicity of a System In statistical mechanics, the probability that we find a system in a given state depends on the *multiplicity* of a system. Meaning that the probabily that we measure a certain outcome, the macrostate, is dependent on all the possible ways the constituent parts, the microstates, can arrange themselves to produce the outcome.

Example 2: Dices

For example if you are told that the outcome of two six-eyed dices is 4, then you would reason that the possible ways to produce 4 is $(1,3)$, $(3,1)$ and $(2,2)$. So the multiplicity is 3. This is an unnormalized probability. If we sum up all the multiplicities for the 11 possible dice outcomes we find that there are 36 microstates in total. So $P(4) = 3/36$ is the normalized probability.

For a physical system of N non-interacting identical particles, the multiplicity is denoted by W, the counting formula for the number of permutations of picking n_i into state i (and picking all, $\sum n_i = N$), and thus the multiplicity is

$$W = \frac{N!}{\prod_i n_i!}. \tag{5.11}$$

.

The *Boltzmann entropy* is defined as the log of the multiplicity:

$$S_B = k_\mathrm{B} \ln W, \tag{5.12}$$

where the expression is scaled by the physical Boltzmann constant $\boldsymbol{k}_\mathrm{B}$.

Example 3: Boltzmann Entropy

Say we have two bins, and 6 identical particles.

I Let's imagine that all the particles are in one of the two bins. The particles can be arranged in $6! = 720$ ways, but as all of these states are indistinguishable we only have $W_1 = \frac{6!}{6!} = 1$ microstates. So the entropy is

$$S_1 = \boldsymbol{k}_\mathrm{B} \ln(1) = 0.$$

II If we imagine that the particles are equally distributed in the two bins then have that $W_2 = \frac{6!}{3!3!} = 20$. We say then that we have 20 possible *microstates* for the given macrostate. The entropy is thus larger than in the previous case,

$$S_2 = \boldsymbol{k}_\mathrm{B} \ln(20) \approx 3k_\mathrm{B}.$$

The most probable macrostate is the one that can be produced by the most number of distinguishable permutations, or in terms of S, the one with the *maximum entropy*. There are of course many more ways we could distribute the particles into the two boxes in example 3, but state I and II are the two states with lowest and highest entropy, respectively.

Derivation of Gibbs' entropy Let us now see how we can devise a more general formula for entropy, for which the entropy in Equation (5.12) is just a special case.

Say, we have a large number of particles $N \to \infty$, we might ask what would be the most probable macrostate W_{max} and its corresponding entropy S_{max}?

We first rewrite the Boltzmann entropy (see Equation (5.12)) as

$$\ln W = \ln N! - \sum_i \ln n_i!, \tag{5.13}$$

where we've omitted k_B for now. Next we use *Stirling's approximation* [26]

$$\ln N! \simeq N \ln N - N. \tag{5.14}$$

We assume that both $N \to \infty$ and $n_i \to \infty$, and thus apply the approximation for both $N!$ and $n_i!$, but hold the fraction $\frac{n_i}{N}$ fixed. Inserting the approximation in Equation (5.13) gives

$$\ln W = N \ln N - \sum_i n_i \ln n_i. \tag{5.15}$$

Next we define a probability in a frequentist fashion. The probability of picking a given bin is the number of particles in the given bin n_i divided by the total number of particles N. Hence we have $p_i = \lim_{N\to\infty}(n_i/N)$. We rearrange $n_i = p_i N$ and insert

into Equation (5.15),

$$\begin{aligned}\ln W &= N\ln N - \sum_i p_i N\left(\ln N + \ln p_i\right)\\ &= N\ln N - N\ln N\sum_i p_i - N\sum_i p_i \ln p_i\\ &= -N\sum_i p_i \ln p_i,\end{aligned} \tag{5.16}$$

where we have used that $\sum_i p_i = 1$ which follows from $\sum_i n_i = N$. If what we seek is to maximize entropy S for a fixed total number of particles N, the prefactor N isn't of any concern. By reintroducing the Boltzmann constant we thus finalize the Gibbs entropy as

Gibbs Entropy

$$S = -k_{\mathrm{B}}\sum_i p\left(x_i\right)\ln p\left(x_i\right). \tag{5.17}$$

Here we have been a bit more statistically formal than previously, and viewed the bins as states x_i of a discrete random variable X, such that $p\left(X = x_i\right) = p_i$.

The Boltzmann entropy is now a special case of Equation (5.17), where the different arrangements of the system are equiprobable, with the probability being $p_i = \frac{1}{W}$. Such a probability distribution function is called the *uniform distribution*. We can note from Equation (5.17) that distributions $p\left(x_i\right)$ that spread particles out more evenly will have a higher entropy than distributions that are more peaked around certain values.

The uniform probability distribution used in the Boltzmann entropy can also be obtained by using Lagrangian undetermined multipliers and maximizing the Gibbs entropy expression

$$\widetilde{S} = -\sum_i p\left(x_i\right)\ln p\left(x_i\right) + \lambda_1\left(\sum_i p\left(x_i\right) - 1\right), \tag{5.18}$$

which in the final term includes a normalizing condition ensuring $\sum_i p_i = 1$. If we used the continuous form of Equation (5.18), replacing our sums with integrals, the equation is two functional integrals and we could solve it by functional optimization.

In fact, we can derive a whole range of probability distributions by using different constraints on the entropy. We could for example use the Gibbs entropy in continuous form and subject it to the constraints using Lagrangian multipliers (see [26] for derivation),

$$\lambda_2\left(\int_{-\infty}^{\infty} xp(x)\mathrm{d}x - \mu\right) + \lambda_3\left(\int_{-\infty}^{\infty}(x-\mu)^2 p(x)\mathrm{d}x - \sigma^2\right). \tag{5.19}$$

These constraints are the first and second moment, namely the mean and the variance. By maximizing the entropy the resulting probability distribution will be the normal distribution.

In a statistical physics context we could obtain what is called the canonical ensemble if we subjected the entropy to some physical constraints instead: The total number of particles $-\lambda_1(\sum_j n_j - N)$ and the total energy of the system $-\lambda_2\sum_j(E_j n_j - E_{total})$. This would allow us to, among other things, for example identify $\lambda_2 = \beta$ as the inverse

temperature $\beta = \frac{1}{k_B T}$. It is clear that entropy is a powerful concept and tool, which we shall bring with us for later use.

Shannon Entropy If we again consider the simple example of particles in bins in Example 3, there is a point to be made regarding the concept of information. We may ask, which of the two cases require the most amount of information to describe? It makes sense that it must be the one with the highest number of unique arrangements to describe, that is, case II. Macrostates resulting from fewer number of configurations would naturally require less information to describe. In fact, we can take this intuition seriously, and use it to define the average amount of information needed to describe a certain configuration. Specifically, we want to define the information entropy as the expected amount of information obtained when drawing a sample from a distribution $p(x_i)$. So on average, observing Case II in our previous example would give us more information about the underlying distribution (the uniform distribution) than Case I as is evident from its higher entropy.

In an information context the Boltzmann constant is redundant, and thus the information entropy is a dimensionless quantity. However, the output of an entropy operation is still described as being in units of information. In that regard we speak of the unit *nat(s)* when we use logarithm of base e, where 1 nat $= \ln(e^1)$. In fact we may use any base to quantify information, but the logarithmic of base 2 is often the most convenient in the context of computation as it corresponds to using the units of *bit(s)*, so that 1 bit $= \log_2(2^1)$. Meaning that one binary digit of information is the logarithm of two possible values, 0 and 1. Setting $k_\mathrm{B} = 1$ in Equation (5.17) and allowing for any logarithmic base gives the *information entropy*, or the *Shannon Entropy*

Shannon Entropy

$$H_b = -\sum_i p\left(x_i\right) \log_b p\left(x_i\right). \tag{5.20}$$

The logarithms to $b = e$ and $b = 2$ bases are related by a numerical factor, so if we want to convert the results we would do that as follows $H_e = \frac{H_2}{\log_2 e}$, or $H_2 = \frac{H_e}{\ln 2}$. We can thus easily convert from nat(s) to bit(s) and vice versa. We shall not carry the subscript b with us further, so we will just denote the entropy as H, or $H(p)$. For the next couple of examples we will use the logarithm base 2 for illustration, but for our numerical experiments we will default to the logarithm base e.

In Shannon's theory of information the interpretation of Equation (5.20) in the context of bits is as follows: Transmitting 1 bit of useful information is equivalent to reducing the receivers uncertainty by a factor of 2. This is best illustrated by an example:

Example 4: Entropy: Weather Forecast

Say, that the weather in some area has a 50% of being sunny or rainy for the following day. If the weather forecast tells you it will be rainy then they have reduced your uncertainty by a factor of 2, you had two possible states and now you have one. This reduction in uncertainty corresponds to 1 bit of useful information.

We can use the same example but now with 8 equiprobable possible states for tomorrow's weather, for example cloudy, partly cloudy, rainy etc. When you now receive the weather forecast (one state being chosen), your uncertainity has been reduced by a factor of 8, that is, 2^3. The number of bits are then $\log_2(2^3) = 3$ bits.

What if we have different probabilities for the possible states? For instance, say we had $P(\text{Sunny}) = 60\%$, $P(\text{Cloudy}) = 30\%$ and $P(\text{Rainy}) = 10\%$. We take the weighted average by using Equation (5.20) obtaining

$$H(p) = -0.6\log_2(0.6) - 0.3\log_2(0.3) - 0.1\log_2(0.1) \approx 1.3 \text{ bits} \tag{5.21}$$

What we've just found is the entropy of the problem. If we imagine now that you live in the rainforest and at the time of the forecast it is during the wet season. To hit the point home we assume $P(\text{Rainy}) \approx 100\%$. The weather is highly predictable and as a consequence the average entropy will be low, $H(p) \approx -1.00\log_2(1.00) = 0$. From these two example we see that the entropy provides a receiver with knowledge about how unpredictable a probability distribution is.

5.3.2 Cross-Entropy

In the cases where the true probability distribution $p(x_i)$ is unknown, the encoding of the underlying outcome x may be predicted by some other probability distribution $q(x_i)$ that approximates $p(x_i)$. When this is the case, the measure for the average number of bits needed to recognize a draw from $p(x_i)$ is called the *Cross-Entropy*:

Cross-Entropy

$$H_p(q) = -\sum_i p(x_i)\log_2 q(x_i) \tag{5.22}$$

The intuition behind the Cross-Entropy is pretty straight-forward and can be illustrated by continuing our weather example

Example 5: Cross Entropy and Weather Forecast

Suppose that the weather station that provided you with the forecast from the previous example, encodes each of the three possible options in packages of 3 bit message length. Using Equation (5.22) the cross entropy would simply be equal to

$$H_p(q) = -0.6\log_2(\frac{1}{2^3}) - 0.3\log_2(\frac{1}{2^3}) - 0.1\log_2(\frac{1}{2^3}) = 3 \text{ bits}, \tag{5.23}$$

where we assume that the encoding in $q(x_i)$ assumes a uniform probability $q(x_i) = \frac{1}{2^{\text{bits}}}$. We can see from Equation (5.23) that the cross entropy corresponds to the average message length. Comparing this result with the entropy in example 4, which was $H \approx 1.3$ bits, we see that the weather station is sending more than twice as many bits as there are useful bits.

To reduce the redundancy we could try out a different configuration of message lengths, assuming that it could convey the information we use the message lengths. For example the bit 1 for the $P(\text{Sunny}) = 60\%$ case, bits 01 for $P(\text{Cloudy}) = 30\%$ and finally three bits 001 for $P(\text{Rainy}) = 10\%$. In this case the cross entropy would be reduced to

$$H_p(q) = -0.6\log_2(\frac{1}{2^1}) - 0.3\log_1(\frac{1}{2^2}) - 0.1\log_2(\frac{1}{2^3}) = 1.5 \text{ bits}. \tag{5.24}$$

The lower bound of the cross entropy is the entropy itself, which follows from Gibbs' inequality which we shall cover in a moment. So there are $H(p) - H_p(q) \approx 1.5 - 1.43 = 0.07$ bits remaining for the probability distribution $q(x)$ to perfectly correspond to the true distribution $p(x)$. Consider now the true probability for rain, 10%. By using 3 bits for this outcome we are implicitly making the prediction, using the $q(x)$ distribution we have defined, that there's a $q(\text{Rain}) = \frac{100}{2^3} = 12.5\%$ probability for rain. With no other assumptions this is the best we can obtain using discrete units of bits.

5.3.3 Relative Entropy (KL-divergence)

In our explanation of the cross entropy, we saw that the natural point of comparison for the cross-entropy $H_p(q)$ was the entropy $H(p)$. The entropy gave the minimal amount of bits needed to convey some random outcome, while the cross-entropy measured how many bits was needed from some distribution $q(x)$ to convey a message about a sample from $p(x)$. A perfect model, with distribution $q(x)$, would have to be equal to the true distribution $p(x)$, and for this case the cross entropy and the entropy would be equal, $H_p(q) = H(p)$.

Now the crux of the whole entropy discussion remains: How much does the cross-entropy exceed the entropy? We have already encountered this difference between $H_p(q)$ and $H(p)$ in example 5, but we shall now formalize it. This quantity has the suitable name *relative entropy*, albeit the enigmatic name *Kullback-Leibler divergence* is more commonly used, thus we shall use it as well. The KL-divergence's relationship with the cross-entropy and the entropy is

$$\begin{aligned} &\textit{Relative Entropy} = \textit{Cross Entropy} - \textit{Entropy} \\ \Rightarrow \quad &\mathbb{KL}[p\|q] = H_p(q) - H(p). \end{aligned} \tag{5.25}$$

Inserting the expressions for the entropy (see Equation (5.17)) and the cross entropy (see Equation (5.22)) into the final equation in (see Equation (5.25)) we get

$$\begin{aligned} \mathbb{KL}[p\|q] &= \left[-\sum_i p\left(x_i\right) \log q\left(x_i\right)\right] - \left[-\sum_i p\left(x_i\right) \log p\left(x_i\right)\right] \\ &= \sum_i p\left(x_i\right)\left(\log p\left(x_i\right) - \log q\left(x_i\right)\right), \end{aligned} \tag{5.26}$$

and finally the KL-divergence in a discrete form is

KL-divergence

$$\mathbb{KL}[p\|q] = \sum_i p\left(x_i\right) \log \frac{p\left(x_i\right)}{q\left(x_i\right)} \tag{5.27}$$

In terms of information the KL-divergence gives us an expression for the average number of additional bits required to encode the outcome x when we wrongly assume $q\left(x\right)$ as the distribution instead of the true distribution $p\left(x\right)$. Another way of looking at it, is that it is the error, or the information loss, we attain by using $q\left(x\right)$ to approximate $p\left(x\right)$.

Properties of th KL-divergence Next, we list some important properties regarding the KL-divergence.

- The KL-divergence satisfies
$$\mathbb{KL}[p\|q] \geqslant 0, \tag{5.28}$$
- The KL-divergence is not a symmetrical quantity
$$\mathbb{KL}[p\|q] \not\equiv \mathbb{KL}[q\|p] \tag{5.29}$$

Equation (5.28) is called the *Gibbs' inequality*. The equality is valid if, and only if, $p(x) = q(x)$. The proof of Gibb's inequality follows directly from *Jensen's inequality*. Jensen's inequality is fairly intuitive in the simple case of two points on a curve: If a function $f(x)$ is convex, a line drawn between any two points on a curve will always lie on or above the curve. Moreover this proposition can be generalized, so that for any set of points $\{x_i\}$ a convex function satisfies $f\left(\sum_{i=1}^M \lambda_i x_i\right) \leqslant \sum_{i=1}^M \lambda_i f\left(x_i\right)$, where the weight obey $\lambda_i \geqslant 0$ and $\sum_i \lambda_i = 1$, and this is Jensen's inequality. Furthermore, if the points are random variables and λ_i a distribution over said variables we can extend this to probability theory. For the convex function f, and the function $Y = Y(X)$ where X is a random variable, Jensen's inequality states that

$$\mathbb{E}[f(Y)] \geq f(\mathbb{E}[Y]). \tag{5.30}$$

This statement is probably quite familiar in the specific case of $f(X) = X^2$, which says that the variance is positive or equal to zero $\mathbb{E}[X^2] - \mathbb{E}[X]^2 \geq 0$.

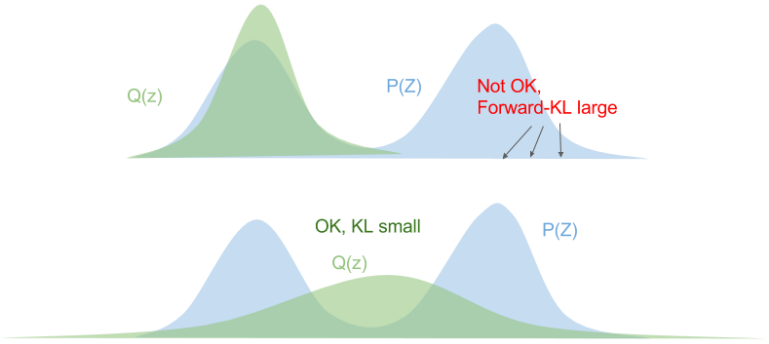


Figure 5.1: Figure showing how the forward KL-divergence, $\mathbb{KL}(p\|q)$, is zero-avoiding. Figure taken from [55].

We use Jensen's inequality and set $f(y_i) = -\log(y_i)$, which is a strictly (contains one minimum) convex function, and $y_i = \frac{q(x_i)}{p(x_i)}$. Using Jensen's inequality with the KL-divergence, we have

$$\mathbb{KL}[p\|q] = -\sum_i p(x_i) \log \frac{q(x_i)}{p(x_i)} \geqslant -\log \sum_i p(x_i) \frac{q(x_i)}{p(x_i)} = 0. \tag{5.31}$$

Where we flipped the numerator and the denominator adding a sign as compared to Equation (5.27), and in the last equality we used the normalization condition $\sum_i q(x_i) = 1$. The inequality in Equation (5.28) is thus proven.

The negation in expression (see Equation (5.29)) shows that the two ways in which we can formulate the KL-divergence are not equal, that is, the KL-divergence is an *asymmetric quantity*. Since both the expressions obey Equation (5.28) we have that $\mathbb{KL}[p\|q] + \mathbb{KL}[q\|p] \geq 0$. Their sum would have to be zero for the KL-divergence to be symmetric, which can only occur when $p(x) = q(x)$. The two directions of the KL-divergence serve different purposes, and they even have their own respective names. $\mathbb{KL}[p\|q]$ is called the *forward KL*, while $\mathbb{KL}[q\|p]$ is called the *reverse KL*. For our problem of approximating a posterior, we need to know which one of the two is the most appropriate. To gauge the difference we follow the explanation delivered in [37], along with the further discussion in [55] about said explanation.

The Forward KL-Divergence We can consider the expression for the KL-divergence in Equation (5.27) as the expectation of a penalty $\log \frac{p(x_i)}{q(x_i)}$, weighed by the function $p(x_i)$. The penalty adds to the KL-divergence whenever $p(x_i) > 0$. As long as $p(x_i) > 0$ we have that $\lim_{q(z)\to 0} \log \frac{p(z)}{q(z)} \to \infty$. We must thus make sure that the optimal $\hat{q}(x_i) > 0$ whenever $p(x_i) > 0$, the forwards KL-divergence is therefore said to be *zero avoiding* for q. The implication of this zero avoidance can be seen in figure 5.1, where the optimal approximation $\hat{q}(x_i)$ for a multimodal distribution will be situated in a region of low density between the two peaks in order to avoid containing zeros on its tails. As we see in the figure, it will typically over-estimate p.

The Reverse KL-Divergence Exchanging the distributions in Equation (5.27) yields the reverse KL,

$$\mathbb{KL}[q\|p] = \sum_i q(x_i) \log \frac{q(x_i)}{p(x_i)}. \tag{5.32}$$

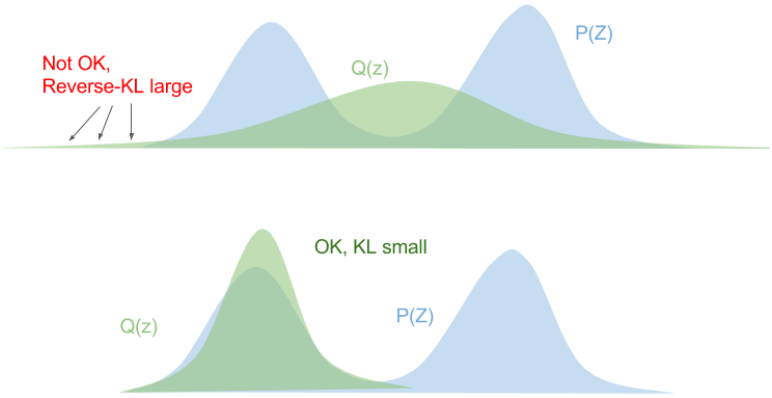


Figure 5.2: Figure showing how the reverse KL-divergence, $\mathbb{KL}[q\|p]$, is zero-enforcing. Figure taken from [55].

Here we have that the penality goes to infinity when the true distribution goes to zero, $\lim_{p(z)\to 0} \log \frac{q(z)}{p(z)} \to \infty$. So when $p\left(x_i\right) = 0$ we need to ensure that $q\left(x_i\right) = 0$. In figure 5.2 we see an illustration of this. The reverse KL-divergence will blow up on the tails if we tried to place q in the middle of the two peaks. The reverse KL-divergence is said to be *zero-enforcing*.

A simple summary of the two is that the forward KL-divergence *stretches* the distribution $q\left(x\right)$ to make sure it covers the whole $p\left(x\right)$ domain, while reverse KL-divergence *squeezes* it into a domain devoid of $p\left(x_i\right) = 0$. We need to choose which types of the KL-divergence is suited. In our context we seek a distribution as a surrogate posterior over neural network weights, the KL-divergence will help us fit that distribution. We want to avoid an approximate distribution that predicts a high density region where the true distribution has a low density, *thus our metric of choice is the reverse KL*. We should note that the cost of choosing the reverse KL, is that if the true distribution $p\left(x\right)$ has a larger multitude of modes than the distribution we are fitting $q\left(x\right)$, we run up the risk of predicting that there is no probability mass density in a region where there actually is a large mass density.

Continuous Multivariate Form All the entropy relations of the previous sections, that is, Equations (see Equation (5.17)), (see Equation (5.22)), (see Equation (5.27)) can be written in continuous form, written with integrals over distributions $p(x)$ and dependent on a continuous variable x. We will not show the transition from discrete to continuous here, but refer to [26] for the curious reader. We may also write the relations over multiple continuous variables, summarized by the bold vector notation $\mathbf{x}$. This means that the differential version of the reverse KL, which is the form we shall use later, is written

Reverse KL-divergence

$$\mathbb{KL}(q\|p) = \int q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \mathrm{d}\mathbf{x} \tag{5.33}$$

5.4 Variational Inference for Neural Networks

After having established a feasible measure we can use we are now ready to study the specific problem of creating a tractable approximation to the posterior of a BNN.

5.4.1 Variational Free Energy

Let us begin by changing the terms of the KL-Divergence in Equation (5.33) such that they contain the relevant expression (see Equation (4.5) and Equation (5.6)):

$$\begin{aligned}\mathbb{KL}[q(\mathbf{w}|\boldsymbol{\theta})\|p(\mathbf{w}|\mathcal{D}]) &= \int q(\mathbf{w}|\boldsymbol{\theta}) \log \frac{q(\mathbf{w}|\boldsymbol{\theta})}{p(\mathbf{w}|\mathcal{D})} d\mathbf{w} \\ &= \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log \frac{q(\mathbf{w}|\boldsymbol{\theta})}{p(\mathbf{w}|\mathcal{D})}\right],\end{aligned} \tag{5.34}$$

where $p(\mathbf{w}|\mathcal{D})$ is the true posterior, and $q(\mathbf{w}|\boldsymbol{\theta})$ is the approximation to the posterior, which will have a known functional form dependent on the parameters $\boldsymbol{\theta}$. What we mean by a known functional form is that we will have to specify which type of probability distribution function we will attempt to fit to the true posterior. As mentioned in Section 5.3.3 this integral (or sum) is just the expected value over the penalty (the log term), weighted by q, so we can choose the more succint notation $\mathbb{E}$ in Equation (5.34). To get an idea of our true posterior we seek an approximate distribution that minimizes the KL-divergence. However, the whole premise for variational inference was that the posterior of the BNN was intractable. So at first glance it seems a bit worrisome that the KL-divergence is dependent on the true posterior distribution $p(\mathbf{w}|\mathcal{D})$, but we can do better.

In fact, as shown by Graves [56] and Blundell *et al.* [42] we do not necessarily need to minimize the KL-divergence explicitly. Following their procedure, let us insert Bayes' rule from Equation (4.5) into the KL-divergence Equation (5.34):

$$\begin{aligned}&\mathbb{KL}[q(\mathbf{w}|\boldsymbol{\theta})\|p(\mathbf{w}|\mathcal{D})] \\ &= \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log \frac{q(\mathbf{w}|\boldsymbol{\theta})p(\mathcal{D})}{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}\right] \\ &= \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log \frac{q(\mathbf{w}|\boldsymbol{\theta})}{p(\mathbf{w})} - \log p(\mathcal{D}|\mathbf{w}) + \log p(\mathcal{D})\right] \\ &= \mathbb{KL}[q(\mathbf{w}|\boldsymbol{\theta})\|p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log p(\mathcal{D}|\mathbf{w})\right] + \log p(\mathcal{D}).\end{aligned} \tag{5.35}$$

Here we have used that $\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log p(\mathcal{D}) = \log p(\mathcal{D})$, because the model evidence is independent of the parameters, and that $q(\mathbf{w}|\boldsymbol{\theta})$ is a normalized distribution and thus $\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} = \int_{\mathbf{w}} q(\mathbf{w}|\boldsymbol{\theta}) = 1$. The first two terms of the last expression in Equation (5.35) is called the *variational free energy* in the literature and is denoted by $\mathcal{F}(\mathcal{D}, \boldsymbol{\theta})$, we shall invariably refer to this quantity as "free energy" as well.

We rewrite Equation (5.35)) in terms of the free energy,

$$\mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta})\|p(\mathbf{w} \mid \mathcal{D})] = \mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) + \log p(\mathcal{D}) \tag{5.36}$$

As just stated, the evidence term in Equation (5.36) does not have a $\boldsymbol{\theta}$ dependence, this also implies that when we eventually optimize the equation we will have that $\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}) = 0$. Thus our optimization objective, that is, our cost function, is simply the variational free energy

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \left[\mathcal{F}(\mathcal{D}, \boldsymbol{\theta})\right],$$

meaning that the $\hat{\boldsymbol{\theta}}$ is the argument that minimizes the free energy. We also see from Equation (5.36) that with $p(\mathcal{D})$ being constant, we minimize the KL-divergence $\mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w} \mid \mathcal{D})]$ by minimizing the free energy. In light of the intractability discussion of Section 5.1, this is a pretty decent result, as it is not tied down by the intractable evidence term $\log p(\mathcal{D}) = \int p(\mathcal{D}|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}$. In Section 5.3 we were seeking a functional, an optimization objective, that we could work with. The variational free energy is that objective

Variational Free Energy

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}[\log P(\mathcal{D} \mid \mathbf{w})]. \tag{5.37}$$

The negative variational free energy is often called the *evidence lower bound* (ELBO). Let us see why by again considering Equation (5.36), but now together with Gibbs' inequality (see Equation (5.28)) so that we have

$$\begin{aligned} \mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) + \log p(\mathcal{D}) &\geq 0 \\ p(\mathcal{D}) &\geq -\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \mathcal{L}(\mathcal{D}, \boldsymbol{\theta}). \end{aligned} \tag{5.38}$$

Where $\mathcal{L}(\mathcal{D}, \boldsymbol{\theta})$ is the ELBO, which is frequently used in variational inference literature. Equation (5.38) illustrates what the name implies, that the ELBO serves as a lower bound on the log model evidence. When we minimize $\mathcal{F}(\mathcal{D}, \boldsymbol{\theta})$ we minimize the KL-divergence $\mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w} \mid \mathcal{D})]$, which means that we shrink the gap between the ELBO and the model evidence.

Parameterization Until now we have not elaborated on what the $\boldsymbol{\theta}$ parameters constitute in the surrogate posterior. In the numerical experiments performed in this thesis we shall rely heavily on the Gaussian distribution. Gaussians are a natural choice for a variety of reasons, but particularly for this study is that they yield simple closed form expressions for the KL-divergence between the surrogate posterior and the prior, that is, in $\mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w})]$. That said, a closed form is not a necessity in this scheme, but a computational convenience. From the use of Gaussian distributions, $\boldsymbol{\theta}$ consist of the means and standard deviations of these distributions

$$\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma}), \tag{5.39}$$

and the Gaussian distributions are over the various weights $\mathbf{w}$ of a neural network. We do not parametrize our BNNs directly with the weights and biases $\mathbf{w}$ but with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, and then sample $\mathbf{w}$ from the variational posterior distribution. This implies that we will have twice as many parameters as in a DNN, as we saw in example 1.

Layer-wise Posterior We should mention that the variational posterior is initialized at each layer $q(\mathbf{w}^l \mid \boldsymbol{\theta}^l)$, yet we will invariably refer to $q(\mathbf{w} \mid \boldsymbol{\theta})$ as *the* variational posterior refering to the totality of the BNN's weight posterior distribution. For example a complete forward pass is a sample from the variational posterior $q(\mathbf{w} \mid \boldsymbol{\theta})$, but in actuality this entails a sample from the variational posteriors at every layer $q(\mathbf{w}^l \mid \boldsymbol{\theta}^l)$.

5.4.2 The Cost Function

The variational free energy is used as a cost function and the cost function we will use to optimize our BNN. Before we outline how we minimize it, let us interpret what it

tells us by writing

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \underbrace{\mathbb{KL}(q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w}))}_{\text{complexity cost}} - \underbrace{\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}[\log p(\mathcal{D} \mid \mathbf{w})]}_{\text{likelihood cost}}. \tag{5.40}$$

Following the nomenclature of [42], the free energy cost function consists of two terms, a prior-dependent part which is referred to as the *complexity cost*, and a data-dependent part referred to as the *likelihood cost*. The conceptual division into these two parts comes naturally from the fact that one term depends on the data while the other does not. Minimizing the cost function with respect to $\boldsymbol{\theta}$ realizes a trade-off betweeen, on the one hand, the model being able to account for the training data (the likelihood cost), and on the other hand minimizing the distance[4] from the prior $p(\mathbf{w})$ to $q(\mathbf{w}|\boldsymbol{\theta})$. We shall soon see a specific example and further interpretation of this cost function after we have chosen the appropriate form of the prior and the approximate posterior. To ease reference we will at times refer to the losses (single data point), so

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta})_i = L^C(\mathbf{w}, \boldsymbol{\theta}) + L^L(\mathbf{w}, \mathcal{D}_i) \tag{5.41}$$

Monte Carlo Approximation Using a Monte Carlo (MC) procedure we can approximate our cost function by sampling weights $\mathbf{w}$ from the variational distribution $q(\mathbf{w}|\boldsymbol{\theta})$. Let us rewrite the free energy in Equation (5.40) in terms of the expectation value

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}[\log q(\mathbf{w} \mid \boldsymbol{\theta}) - \log p(\mathbf{w}, \mathcal{D})] \tag{5.42}$$

and then by using MC integration we approximate the cost $\mathcal{F}(\mathcal{D}, \boldsymbol{\theta})$ as

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) \approx \hat{\mathcal{F}}(\mathcal{D}, \boldsymbol{\theta}) = \frac{1}{N_{mc}} \sum_{i=1}^{N_{mc}} [\log q(\mathbf{w}_i \mid \boldsymbol{\theta}) - \log p(\mathbf{w}_i, \mathcal{D})]. \tag{5.43}$$

where we have that $\mathbf{w}_i$ is the ith MC sample of N_{mc} from the approximate variational posterior $q(\mathbf{w}_i \mid \boldsymbol{\theta})$. This is the cost we will use in our experiments and report when presenting our results. This MC approximated cost is amenable to any continouos choice of $q(\mathbf{w}_i \mid \boldsymbol{\theta})$ and $p(\mathbf{w})$. But as mentioned previously, when using Gaussians the KL-divergence part, the complexity cost, has a closed form expression so that in most of our calculations the MC approximation will only be done on the likelihood cost. However we will invariably refer to the MC approximation as this is can be used generally.

A brief clarification on notation that will be used in the remainder of the chapter: We use n to signify number of datapoints, N for number of parameters or distributions, N_{mc} for MC samples, M for number of mini-batches, S for samples in a mini-batch, and L for number of layers.

5.5 Network Training

Before we lay out an algorithmic receipt for training a variational BNN, we first study some details of the procedure.

5.5.1 Backpropagation

There are two popular ways of minimizing the free energy, the *Black Box Variational Inference* (BBVI) and *Bayes by Backprop* (BbB), where the latter is the one we shall

[4] KL-divergence is strictly speaking not a distance measure, its asymmetric nature disqualifies it.

study in this thesis. To be able to perform backpropagation as described in Section 3.5, we need to be able to formulate a gradient $\nabla_{\boldsymbol{\theta}}\mathcal{F}(\boldsymbol{\theta})$ of the cost function. However, for deep BNNs the analytical derivatives needed to form $\nabla_{\boldsymbol{\theta}}\mathcal{F}(\boldsymbol{\theta})$ are usually not practically available. This is of course apparent by the fact that we have already opted for a MC integration procedure for the cost function in Equation (5.43). We seek to obtain the gradient of the sampled cost function, $\nabla_{\boldsymbol{\theta}}\hat{\mathcal{F}}(\boldsymbol{\theta})$, but unfortunately $\hat{\mathcal{F}}(\mathcal{D}, \boldsymbol{\theta})$ is not a smooth function of $\boldsymbol{\theta}$, and thus the function is not differentiable as it stands.

Let us study the origin of this problem closer before we investigate the solution. Say we want to find the gradient of the expectation value of some differentiable function $f(\mathbf{w}|\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, and where the probability density of the expectation is $q(\mathbf{w})$. Taking the gradient of this expectation gives

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{q(\mathbf{w})}\left[f(\mathbf{w}|\boldsymbol{\theta})\right] &= \nabla_{\boldsymbol{\theta}}\left[\int q(\mathbf{w}) f(\mathbf{w}|\boldsymbol{\theta}) d\mathbf{w}\right] \\ &= \int q(\mathbf{w})\left[\nabla_{\boldsymbol{\theta}} f(\mathbf{w}|\boldsymbol{\theta})\right] d\mathbf{w} \\ &= \mathbb{E}_{q(\mathbf{w})}\left[\nabla_{\boldsymbol{\theta}} f(\mathbf{w}|\boldsymbol{\theta})\right].\end{aligned} \tag{5.44}$$

In this case, taking the gradient of the expectation is equal to taking the expectation of the gradient, which is equivalent to stating that *Leibniz's rule* applies. This allows us to use Monte Carlo integration.

However, if we say that the probability distribution, $q(\mathbf{w}, \boldsymbol{\theta})$, serving as the expectation weight itself, is also parametrized by $\boldsymbol{\theta}$, which *is* the case in the free energy cost function (see Equation (5.42)), we will end up with the differentiation [57],

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[f(\mathbf{w}|\boldsymbol{\theta})\right] &= \int \nabla_{\boldsymbol{\theta}}\left[q(\mathbf{w}|\boldsymbol{\theta}) f(\mathbf{w}|\boldsymbol{\theta})\right] d\mathbf{w} \\ &= \int f(\mathbf{w}|\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} q(\mathbf{w}|\boldsymbol{\theta}) d\mathbf{w} + \int q(\mathbf{w}|\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} f(\mathbf{w}|\boldsymbol{\theta}) d\mathbf{w} \\ &= \underbrace{\int f(\mathbf{w}|\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} q(\mathbf{w}|\boldsymbol{\theta}) d\mathbf{w}}_{\text{Problematic term}} + \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}} f(\mathbf{w}|\boldsymbol{\theta})\right].\end{aligned} \tag{5.45}$$

If $\nabla_{\boldsymbol{\theta}} q(\mathbf{w}|\boldsymbol{\theta})$ had an analytical expression we would not have any problem, but this is generally not the case. Monte-Carlo integration ensures convergence if we can sample from the probability distribution, but this does not translate to the sampling of the gradient of a probability distribution [26]. Simply put, taking gradients of a stochastic quantity is both conceptually and practically problematic. Therefore, as it stands, we can not use Monte Carlo integration as a means for estimation.

5.5.2 Reparametrization trick

The solution to the problem of the MC estimation of the gradients, as solved in Bayes By Backprop [42], is to employ a *change of variables* technique called the *reparameterization trick*:

- Begin by introducing the random variable $\boldsymbol{\epsilon}$ drawn from a suitable *parameter free* probability density.
- Assume a parameterization of the weights with a deterministic function $t(\boldsymbol{\theta}, \boldsymbol{\epsilon})$, that is $\mathbf{w} = t\left(\boldsymbol{\theta}, \boldsymbol{\epsilon}\right)$.

- Through change of variables we have that the marginal probability density obeys $q(\boldsymbol{\epsilon})d\boldsymbol{\epsilon} = q(\mathbf{w} \mid \boldsymbol{\theta})d\mathbf{w}$.

This allows us to write down the relation,

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\theta}}\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}[f(\mathbf{w},\boldsymbol{\theta})] &= \frac{\partial}{\partial \boldsymbol{\theta}}\int f(\mathbf{w},\boldsymbol{\theta})q(\mathbf{w} \mid \boldsymbol{\theta})\mathrm{d}\mathbf{w} \\ &= \frac{\partial}{\partial \boldsymbol{\theta}}\int f(\mathbf{w},\boldsymbol{\theta})q(\boldsymbol{\epsilon})\mathrm{d}\boldsymbol{\epsilon}.\end{aligned} \tag{5.46}$$

We now have an appropriate functional derivative, and we do a total differentiation of the integrand, giving

$$\frac{\partial}{\partial \boldsymbol{\theta}}\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}[f(\mathbf{w},\boldsymbol{\theta})] = \mathbb{E}_{q(\boldsymbol{\epsilon})}\left[\frac{\partial f(\mathbf{w},\boldsymbol{\theta})}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{\partial \boldsymbol{\theta}} + \frac{\partial f(\mathbf{w},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]. \tag{5.47}$$

Let us now apply this technique and give free energy minimization a new attempt, first by changing variables in Equation (5.43),

$$\hat{\mathcal{F}}(\boldsymbol{\theta}) = \frac{1}{N_{mc}}\sum_{i=1}^{N_{mc}}\left[\log q\left(\mathbf{w}\left(\boldsymbol{\theta},\boldsymbol{\epsilon}_i\right),\boldsymbol{\theta}\right) - \log p\left(\mathbf{w}\left(\boldsymbol{\theta},\boldsymbol{\epsilon}_i\right),\mathcal{D}\right)\right]. \tag{5.48}$$

Note that we are now summing over $\boldsymbol{\epsilon}_i$ instead of $\mathbf{w}_i$. In order to find the gradients of $\hat{\mathcal{F}}(\boldsymbol{\theta})$ we use (see Equation (5.47)), giving

Free Energy Gradient

$$\nabla_{\boldsymbol{\theta}}\hat{\mathcal{F}}(\boldsymbol{\theta}) = \frac{1}{N_{mc}}\sum_{i=1}^{N_{mc}}\left[\frac{\partial \hat{\mathcal{F}}_i}{\partial \mathbf{w}}\frac{\partial \mathbf{w}}{\partial \boldsymbol{\theta}} + \frac{\partial \hat{\mathcal{F}}_i}{\partial \boldsymbol{\theta}}\right], \tag{5.49}$$

where

$$\hat{\mathcal{F}}_i = \log q\left(\mathbf{w}\left(\boldsymbol{\theta},\boldsymbol{\epsilon}_i\right),\boldsymbol{\theta}\right) - \log p\left(\mathbf{w}\left(\boldsymbol{\theta},\boldsymbol{\epsilon}_i\right),\mathcal{D}\right). \tag{5.50}$$

As we now have our gradient we can employ the backpropagation algorithm and all the gradient methods of Chapter 3. We shall see a specific example on how the reparametrization trick is implemented later in the next chapter

5.6 Training Algorithm and Implementation

We will in this section cover some important details regarding variational inference and also look at a specific example illustrating what the previously outlined concepts entails in practice.

5.6.1 Gaussian Variational Posterior

At every hidden layer $1 \leq l \leq L$ of the BNN we implement a variational Gaussian posterior $q(\mathbf{w}^{(l)} \mid \boldsymbol{\theta}^{(l)})$. We can either choose a mean-field Gaussian approximation (MFVI), or a full covariance Gaussian approximation (FCVI). Specifically MFVI implements a fully factorised Gaussian distribution as our approximate posterior for a given layer, so that

$$q(\mathbf{w}^{(l)} \mid \boldsymbol{\theta}^{(l)}) = \mathcal{N}(\mathbf{w}^{(l)}|\boldsymbol{\mu}^{(l)},\boldsymbol{\sigma}^{2,(l)}) = \prod_{i=1}^{N}\mathcal{N}_i(\mu_i^{(l)},{\sigma_i}^{2,(l)}). \tag{5.51}$$

The mean and the standard deviation $\boldsymbol{\mu}^{(l)}$, $\boldsymbol{\sigma}^{(l)}$ have the same dimensions in the case of MFVI and their length N corresponds to all the weight connections from layer $l-1$ to l. For the purpose of feed-forward and backpropagation algorithms these vectors are transformed into a weight matrix $\mathbf{W}^{(l)}$ as in section (see Section 5.1). For a one layered network MFVI will then have $2N$ number of independent parameters.

For FCVI on the other hand we have the full covariance matrix such that the off-diagonal elements are included and represent the covariances between weight distributions in the same layer. For a symmetric covariance matrix we will have $N(N+1)/2$ independent parameters, together with N number of μ_i, so we have a total of $N(N+3)/2$ independent parameters. Thus it grows quadratically with respect to the number of weights. The variance of a Gaussian is also found in the denominator of the normalization factor (see Appendix B). Variational BNNs thus involves the computation of the inverse of a covariance matrix, which is an expensive procedure in the case of FCVI. In fact, for BNNs the time complexity for MFVI with respect to the number of independent parameters is $\mathcal{O}(n)$ according to [58], while for FCVI it is $\mathcal{O}(n^6)$. While implementations that are more complex than MFVI, but as complex as FCVI have been devised [59], it still remains considerably more expensive than MFVI.

The MFVI is thus attractive judging from computational efficiency alone, but it comes with the added expense of being less flexible. It is reasonable to assume that there are often strong correlations between parameters in the true posterior, thus the MFVI may fall short of being a good approximation. However, in [58] they demonstrate, by comparison with Hamiltonian Monte Carlo sampling methods, that the MFVI *can* provide the required complexity *if* the BNN architecture is wide and deep, that is, many layers and nodes. The theoretical argument is summarized as follows: Assuming each weight matrix is an MFVI initialization, meaning that the elements correspond to the means and variances of a diagonal Gaussian. For 2×2 weight matrices $\mathbf{W}^{(l)}$ in a two layered BNN, with only the identity activation function, we have for feed-forward propagation

$$\begin{aligned} y &= \prod^{L=2} \mathbf{W}^{(l)} \mathbf{x} \\ &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \mathbf{x} \\ &= \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} \mathbf{x}, \end{aligned} \tag{5.52}$$

where A and B terms are individual weights of the neural network. The take away here is that there are shared A_{ij} and B_{ij} weights in the different elements in the final product matrix. The implication is that while the individual weight matrices do not have correlations, the final product matrix does. In this thesis we shall only use a MFVI approach.

5.6.2 Algorithmic Example

Consider a Gaussian as our variational posterior

$$q(\mathbf{w} \mid \boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right), \tag{5.53}$$

with a mean $\boldsymbol{\mu}$ and a standard deviation parameterized by a softplus function

$$\boldsymbol{\sigma} = \log(1 + e^{\boldsymbol{\rho}}). \tag{5.54}$$

The softplus function provides numerical stability for gradient calculations, in that its derivative $\lim_{\boldsymbol{\rho}\to+\infty}(\frac{1}{1+e^{-\boldsymbol{\rho}}} = 1$ or 0 for $-\infty$. Additionally is ensures that the standard deviation is always non-negative. The parameters of the variational posterior is thus

$$\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\rho}). \tag{5.55}$$

We assume that for the first optimization step some initialization scheme $\boldsymbol{\theta}_{t=0} = (\boldsymbol{\mu}_0, \boldsymbol{\rho}_0)$ has been implemented. Now, as we have already covered in Section 5.5.2, we will have issues calculating the gradient by taking samples as

$$\mathbf{w} \sim q(\mathbf{w} \mid \boldsymbol{\theta}). \tag{5.56}$$

Instead we can exploit the fact that the normal distribution for the MFVI case in Equation (5.53) can be written as

$$\mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\right) = \boldsymbol{\mu} + \boldsymbol{\sigma} \circ \mathcal{N}\left(0, \mathbf{I}_{N\times N}\right). \tag{5.57}$$

The symbol $\circ$ in Equation (5.57) means elementwise multiplication, and the standard normal distribution is the distribution $q(\epsilon)$ we referred to in Section 5.5.2. Note that since this distribution is set as the standard normal it is fixed and thus parameter free during training. So the fitting procedure and thus the gradients can be calculated with respect to the non-stochastic $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$.

Let us now work through the full algorithm for one training iteration, consisting of one forward pass and one backward pass. The walkthrough is inspired by Blundell [42]. Three points to make first: We only look at one single ϵ sample, and one sample of the prior. We have already covered backpropagation in Chapter 3 so it is implicit in the following that weight derivatives involve a recursive chain rule. The complexity loss L^C is calculated for each layer and summed up successively $L^C = \sum_l^L L_l^C$. The data dependent part, that is, the likelihood loss L^L is calculated in standard neural network fashion from the output of the network (see Section 3.3). The complete algorithm can then be found in Algorithm 1.

Forward Pass

Input data $\mathcal{D}$

Set initial $\boldsymbol{\theta}_0$

for variational layer l in $1 \leq l \leq L$ **do**

Sample $\epsilon \sim \mathcal{N}(0, \mathbf{I}_{N \times N})$

Obtain posterior sample

$$\mathbf{w}_q^{(l)} = \boldsymbol{\mu}^{(l)} + \log(1 + \exp(\boldsymbol{\rho}^{(l)})) \circ \epsilon \tag{5.58}$$

Let $\boldsymbol{\theta}^{(l)} = (\boldsymbol{\mu}^{(l)}, \boldsymbol{\rho}^{(l)})$

Sample the prior $\mathbf{w}_p^{(l)} \sim p(\mathbf{w})$

Add samples to the complexity loss

$$\begin{aligned} L_l^C &= \log q(\mathbf{w}^{(l)} | \boldsymbol{\theta}^{(l)}) - \log p(\mathbf{w}^{(l)}) \\ &= \log(\mathbf{w}_q^{(l)}) - \log(\mathbf{w}_p^{(l)}) \end{aligned} \tag{5.59}$$

Convert to weight matrix and bias vector, $\mathbf{w}^{(l)} \Rightarrow (\mathbf{W}^{(l)}, \mathbf{B}^{(l)})$.[a]

Execute forward pass $\mathbf{A}^{(l)} = f\left(\mathbf{W}^{(l)} \mathbf{A}^{(l-1)} + \mathbf{B}^{(l)}\right)$

end for

Retrieve outputs $\boldsymbol{\theta}$ from final layer

Update free energy loss be $L(\mathbf{w}, \boldsymbol{\theta}) = \sum L_l^C + L^L$

Backward Pass

for iterations and epochs[b] **do**

Calculate gradient with respect to the mean

$$\nabla_\mu L(\mathbf{w}, \boldsymbol{\theta}) = \frac{\partial L(\mathbf{w}, \boldsymbol{\theta})}{\partial \mathbf{w}} + \frac{\partial L(\mathbf{w}, \boldsymbol{\theta})}{\partial \boldsymbol{\mu}}. \tag{5.60}$$

Calculate gradient with respect to the standard deviation

$$\nabla_{\boldsymbol{\rho}} L(\mathbf{w}, \boldsymbol{\theta}) = \frac{\partial L(\mathbf{w}, \boldsymbol{\theta})}{\partial \mathbf{w}} \frac{\epsilon}{1 + \exp(-\boldsymbol{\rho})} + \frac{\partial L(\mathbf{w}, \boldsymbol{\theta})}{\partial \boldsymbol{\rho}}. \tag{5.61}$$

Update variational parameters

$$\begin{aligned} \boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} - \eta \nabla_{\boldsymbol{\mu}} L(\mathbf{w}, \boldsymbol{\theta}) \\ \boldsymbol{\rho} &\leftarrow \boldsymbol{\rho} - \eta \nabla_{\boldsymbol{\rho}} L(\mathbf{w}, \boldsymbol{\theta}). \end{aligned} \tag{5.62}$$

end for

Algorithm 1: Variational Bayesian neural network algorithm.

[a] With dimensions $(n^{(l)} \times n^{(l-1)}, n^{(l)} \times m)$, where m is the number of feature columns.

[b] Iterations is over mini-batches, and epochs is number of times over all mini-batches.

To expand on the points made above, the weight derivatives are $\frac{\partial L(\mathbf{w}, \boldsymbol{\theta})}{\partial \mathbf{w}}$, and these are shared by both Equation (5.60) and (see Equation (5.61), so we only need to calculate

these once per iteration. It is only these derivatives that are calculated by standard backpropagation as in Chapter 3. The other two derivative terms, $\frac{\partial L(\mathbf{w},\boldsymbol{\theta})}{\partial \boldsymbol{\rho}}$ and $\frac{\partial L(\mathbf{w},\boldsymbol{\theta})}{\partial \boldsymbol{\mu}}$, are calculated directly on the surrogate posterior $\log q(\mathbf{w} \mid \boldsymbol{\theta})$ by means of gradient descent only. So essentially the algorithm is such that to find the means and standard deviations, we calculate the standard backpropagation gradients and then we scale and shift them. The symbol η is simply the learning rate, but as mentioned previously, we will use more involved gradient descent schemes as covered in Equation (3.14).

5.6.3 KL weighting

The complexity cost, $\mathbb{KL}(q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w}))$, of the cost function is usually weighted by a KL-weight parameter π [42]. This follows from the use of mini-batch gradient descent as outlined in Section 3.4.1. In mini-batch gradient descent the data $\mathcal{D}$ is randomly partitioned into equal batches, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_M$. For each iteration the gradient is averaged over the data points in a batch. The use of mini-batch gradient descent means that the complexity cost, without weighting, will vary depending on the number of mini-batches used. The complexity cost is of course independent of the data, but the number of mini-batches M consitutes how many forward and backward passes (iterations) is performed for every run through the whole data (epochs). We should therefore scale every contribution to the complexity cost according to the number of batches, so that $\pi = \frac{1}{M}$, and the free energy cost function is thus

$$\mathcal{F}_i(\mathcal{D}_i, \boldsymbol{\theta}) = \frac{1}{M}\mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w})] - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}[\log p(\mathcal{D}_i \mid \mathbf{w})] \tag{5.63}$$

for every mini-batch $i = 1, 2, \ldots, M$.

The scaling of the complexity cost can be taken further and considered a hyperparameter suitable for optimization. As minimization of the complexity cost involves minimizing the dissimilarity between the variational posterior $q(\mathbf{w} \mid \boldsymbol{\theta})$ and the prior distribution $p(\mathbf{w})$, the KL-weight necessarily controls the relevance of the prior during training.

- As an extreme, if we set $\pi = 0$ the cost function will be comparable to a negative log-likelihood cost function as described in Section 4.3. The KL-weight should therefore not bee too small as this will produces variances that underestimate the true variance, and hence underestimates the uncertainty.
- If π is set very large then the movement of the gradient in parameter space will favour steps that reduces the complexity cost. This may thus overshadow the likelihood cost, and the results may be a bad fit to the data.

The hyperparameter optimization problem is thus to find a π that fits well to the data and produces reliable predictive uncertainties.

Annealing Scheme Alternatively, [42] proposes an annealing scheme where for every iteration i the KL-weight is reduced according to

$$\pi_i = \frac{2^{M-i}}{2^M - 1}, \tag{5.64}$$

where M is the number of mini-batches as before. This is then restarted and repeated for every epoch. The reasoning behind this scheme is as follows: As the first mini-batches are fed into the network the cost function is heavily weighted by the complexity

cost. This weighting then rapidly drops off such that the likelihood cost then becomes the dominant contribution to the cost function, and the influence of the prior becomes small. Other authors, have found quiet different schemes to be ideal for their problem [60], varying the KL-weight over epochs as well(or alternatively), such as starting out with a *small* KL-weight for a warm-up period of a few number of epochs and then increase it linearly afterwards. This is done such that the gradient can first locate a region in parameter space that allows for stable convergence before the focus is put on the complexity cost. This involves the introduction of four hyperparameters, the kl-weight on start of training π_a, the kl-weight at the end of the annealing period π_b, and finally two "epochs" parameters that decides when to start annealing τ_0, and how many epochs it should last τ_Δ (until the maximum kl weight π_b has been reached). The linear coefficient was then calculated as $g = \frac{\pi_b - \pi_a}{\tau_\Delta}$, and the linear equation defining the kl weight during annealing is then

$$\pi = g(\tau - \tau_0) + \pi_a, \tag{5.65}$$

where τ is the current epoch.

In this thesis we shall try both the exponential mini-batch annealing of Equation (5.64) and the linear epoch annealing of Equation (5.65). For the former we introduce a hyperparameter allowing us to control the speed at which π_i is reduced. We rewrite Equation (5.64) as follows

$$\begin{aligned} \frac{2^{M-i}}{2^M - 1} &= \frac{e^{(M-i)\ln 2}}{e^{M\ln 2} - 1} \\ &= \frac{e^{-i\ln 2}}{1 - e^{-M\ln 2}}. \end{aligned} \tag{5.66}$$

We observe that the term in the denominator is

$$1 - e^{-(M)\ln 2} \approx 1.00 \text{ (2 d.p.)}, \qquad M \geq 20. \tag{5.67}$$

The programming library `TensorFlow` [61] uses a default batch-size of $S = 32$, and common user choices are usually between $S = 1$ and $S = 512$. A data set of $N = 10.000$ data points would for $S = 32$ give $M = 313$ mini-batches, and for $S = 512$ give $M = 20$ mini-batches. We can thus safely omit the denominator after the last equality in Equation (5.66). Lastly we replace $\ln 2$ in the exponential argument in the numerator in Equation (5.66) by the hyperparamter κ, so our annealing scheme is simply

Annealing Scheme

$$\pi_i = e^{-i\kappa}, \qquad i = 1, 2, \ldots, M. \tag{5.68}$$

Figure 5.3 depicts the reduction of π_i with respect to mini-batch i, for different values of κ. Note that $\kappa = \ln 2 \approx 0.693$, which is depicted in figure 5.3, is approximately the same scheme used by Blundell *et. al.* [42]

5.6.4 Cost Implementation

In the following discussion we derive the free energy cost function for a diagonal Gaussian posterior, described in Section 5.6.1,

$$q(\mathbf{w} \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}_i(\mu_{q,i}, \sigma_i^2). \tag{5.69}$$

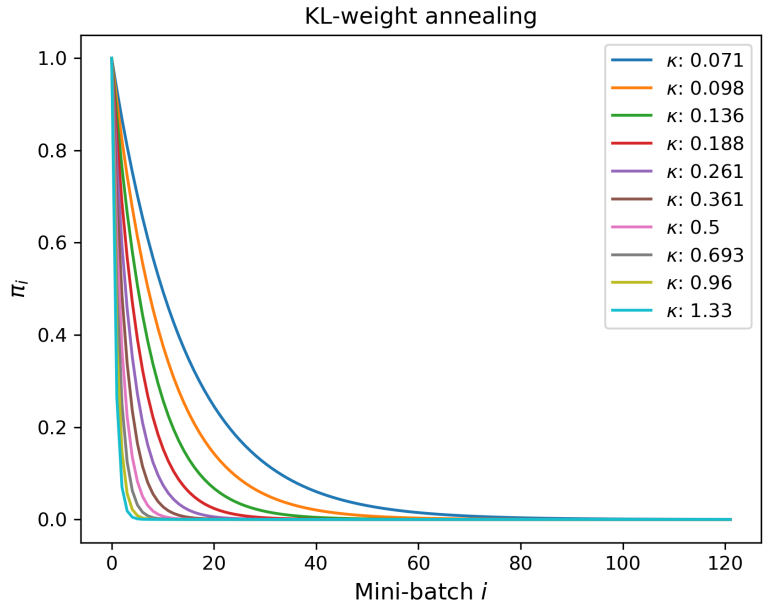


Figure 5.3: KL-weight annealing using Equation (5.68) for different values of κ.

We choose the same form for the prior,

$$p(\mathbf{w}) = \prod_{i=1}^{N} \mathcal{N}_i(\mu_{p,i}, \tau_i^2). \tag{5.70}$$

Note that the prior distribution is non-trainable, and would be unaffected by the training process. We can calculate the KL-divergence exactly. The formula for the KL-divergence between two multivariate Gaussians is listed in Equation (B.2). We apply for the case stated, and obtain

$$\begin{aligned} L^C &= \mathbb{KL}[q(\mathbf{w} \mid \boldsymbol{\theta}) \| p(\mathbf{w})] \\ &= \sum_i^N \frac{1}{2} \left[\frac{\sigma_i}{\tau_i} + \left(\frac{\mu_{p,i} - \mu_{q,i}}{\tau_i} \right)^2 - N + \log \frac{\tau_i^2}{\sigma_i^2} \right] \\ &= \sum_i^N \left[\frac{\sigma_i + (\mu_{p,i} - \mu_{q,i})^2}{2\tau_i^2} - N + \log \frac{\tau_i}{\sigma_i} \right]. \end{aligned} \tag{5.71}$$

Next we assume we are working with a full data set $\mathcal{D}$. For the likelihood cost we use a Gaussian likelihood function, similar to what we have worked with before in Equation (5.5) and take the negative logarithm such that we obtain

$$\sum_j^n L_j^L = \frac{1}{2\alpha^2} \sum_i^{N_{mc}} \sum_j^n \left\{ \hat{y}\left(\mathbf{x}_j, \mathbf{w}_i\right) - y_j \right\}^2, \tag{5.72}$$

where we have used α^2 to denote the likelihood variance. We can write the likelihood in terms of the MSE function so as to relate the discussion to the cost function of a DNN in Section 3.3. Equation (5.72) thus becomes

$$\sum_j^n L_j^L = \frac{n}{2\alpha^2 N_{mc}} \sum_i^{N_{mc}} \text{MSE}(\hat{\mathbf{y}}_i, \mathbf{y}). \tag{5.73}$$

We should mention that we could have used a different probability distribution to define the likelihood, or the posterior or prior for that matter. Had we used the Laplace distribution for the likelihood(see Chapter B), the argument of the exponential would have been an absolute value (L1 norm) and we would have MAE$(\hat{\mathbf{y}}_i, \mathbf{y})$ in Equation (5.73) instead, as described in Section 3.3.

Assuming that we use one batch for the data the free energy cost function is

$$\begin{aligned}\hat{\mathcal{F}} &= L^C + \sum_j^n L_j^L \\ &= \sum_i^N \left[\frac{\sigma_i + (\mu_{p,i} - \mu_{q,i})^2}{2\tau_i^2} - N + \log\frac{\tau_i}{\sigma_i}\right] - \frac{n}{2\alpha^2 N_{mc}} \sum_i^{N_{mc}} \text{MSE}(\hat{\mathbf{y}}_i, \mathbf{y}).\end{aligned} \tag{5.74}$$

The coefficient $\frac{n}{N_{mc}}$ could be omitted, as they will only affect the magnitude of the gradient during optimization. This magnitude is again tuned by the learning rate η, so the coefficients only shift the space in which to search for the optimal learning rate given a fixed number of data points and MC samples. However, if we want to keep the learning rate independent they should be included. The same argument applies for batch-size S using mini-batches.

Sample Size While a point has been made about taking multiple MC samples, in practice it is usually sufficient to take just a single sample at each layer, so $N_{mc} = 1$ [62]. This is also the default in `TensorFlow`. A possible explanation for this efficiency is the use of mini-batch gradient descent. Specifically, assuming $\Delta\mathbf{w}$ in $\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta\mathbf{w}_i$ is sufficiently small at each iteration, partitioning a dataset into M mini-batches serves approximately the same purpose as increasing the number of MC Samples. In effect, partitioning the data set into M mini-batches leads to M forward passes and thus $M \cdot N_{mc}$ samples within the same region of parameter space, at each layer for every epoch.

Variable Variance The current architecture of the BNN contains one output node yielding a target prediction $\hat{\mathbf{y}}$ for multiple datapoints and one forward pass. One may set up multiple output nodes for multiple target labels, but this is not of concern for the experiments in this thesis. However, there is a reason why we would want at least two output nodes. Until now we have assumed that the variance of the likelihood, α^2, is constant. But we can add another output node to the network dedicated to feeding the variance of the likelihood, with an output value from the network. In that case we have $\alpha_j^2 = \alpha^2(\mathbf{x}_j, \mathbf{w})$, for a given datapoint. Since the variance is either positive or zero we need to transform the network output, which is akin to adding an activation function on the output node that feeds values to the variance. The other node that is feeding the mean uses the indentity activation as in DNN regression. The choice of activation function for the variance node is the same as before (see Equation (5.54), namely the softplus function

$$\alpha_j = \log(1 + \exp(\hat{y}_{j,\alpha})). \tag{5.75}$$

Where we have that $\hat{y}_{j,\alpha}$ denotes the untransformed variance-output for one data point j, before it is transformed by the activation function. The motivation behind this addition relates to uncertainty quantification, which we shall cover in Section 5.7.2.

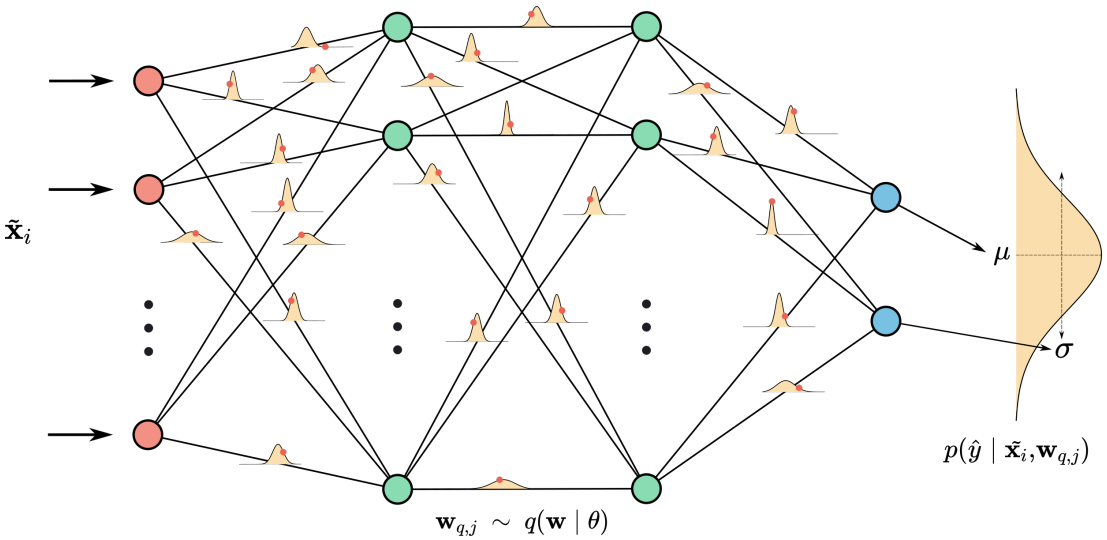


Figure 5.4: Showing a feed-forward sample of the BNN. One such sample is equivalent to a single neural network, and a single prediction of the mean and the variance of a Gaussian likelihood. To produce a posterior predictive mixture distribution multiple samples, such as this one, is required.

5.7 Prediction and Quantification of Uncertainty

After the BNN has been trained on a data set $\mathcal{D}_{train}$ the next step of the process is to make predictions and assess our degree of confidence in those predictions.

Posterior Predictive Distribution For a single test data point, $\tilde{\mathbf{x}}_i$, we wish to obtain the posterior predictive distribution for a corresponding unknown target $\hat{y}_i$ as in Equation (4.27) and as shown in Figure 5.4. By obtaining the predictive distribution we will have what we need for both predictions and uncertainty quantification. Unlike in Equation (4.27) we now integrate over a variational posterior $q(\mathbf{w} \mid \boldsymbol{\theta}, \mathcal{D}_{train})$, that is

$$p(\hat{y}_i|\tilde{\mathbf{x}}_i) = \int p\left(\hat{y}_i \mid \tilde{\mathbf{x}}_i, \mathbf{w}\right) q\left(\mathbf{w} \mid \boldsymbol{\theta}\right) \mathrm{d}\mathbf{w}. \tag{5.76}$$

To obtain an approximation of this integral we sample from the *joint distribution*

$$p(\hat{y}_i|\tilde{\mathbf{x}}_i) = \int p\left(\hat{y}_i, \tilde{\mathbf{x}}_i \mid \mathbf{w}_q\right) \mathrm{d}\mathbf{w}_q, \tag{5.77}$$

So in the discrete sampling case every set of $\{\hat{y}_{i,1} \dots \hat{y}_{i,N_l}\}$, where N_l is the number of samples from the likelihood function, has a corresponding set of $\mathbf{w}_j$ sampled from the variational posterior. Note that when making predictions dependent on a trained model, each new test data point is evaluated independently of other test points, so the likelihood function that we refer to here is equivalent to a normalized conditional probability.

Sampling the posterior predictive distribution involves a two stage sampling, shown in Algorithm 2.

Posterior Predictive Distribution
For data point $(\hat{y}, \tilde{\mathbf{x}}) = (\hat{y}_i, \tilde{\mathbf{x}}_i)$
1: **for** j in number of forward-passes N_{mc} **do**
2: Sample weights $\mathbf{w}_{q,j} \sim q(\mathbf{w} \mid \boldsymbol{\theta})$
3: **for** k in number of likelihood samples N_l **do**
4: $\hat{y}_{k,j} \sim p(\hat{y} \mid \tilde{\mathbf{x}}_i, \mathbf{w}_{q,j})$
5: **end for**
6: **end for**
7: Posterior predictive distribution for $\hat{y}_i$ is then

$$p(\hat{y}_i|\tilde{\mathbf{x}}_i) \approx \{\hat{y}_1, \dots, \hat{y}_{N_l}\}_1 \cup, \dots, \cup \{\hat{y}_1, \dots, \hat{y}_{N_l}\}_{N_{mc}} \tag{5.78}$$

Algorithm 2: BNN: Posterior Predictive sampling

In practice, parts of Algorithm 2 is somewhat a default in the BNN architecture. A forward pass of the BNN is already equivalent to a weight sample from the variational posterior, and at the end of the forward pass the weight-dependent output of the network is fed into the likelihood function. So to construct the predictive distribution for a given datum we simply do multiple forward passes N_{mc} and sample N_l from the likelihood for each pass.

Note that we could just set $N_l = 1$ and rather increased N_{mc}, so as to cover a sufficient number of parameter variations. In that case we would have

$$p(\hat{y}_i|\tilde{\mathbf{x}}_i) \approx \{\hat{y}_1, \dots, \hat{y}_{N_{mc}}\}. \tag{5.79}$$

It may, however, be wise to sample multiple values from the likelihood function as in algorithm 2 instead of increasing N_{mc}, as forward pass through the network is more computationally costly. In the further discussion we will denote the sets that we have discussed as $\{\hat{y}_{i,j}\}_j^{N_{mc}}$ and $\{\hat{y}_{i,j}\}_j^{N_{mc},N_l}$, but mainly the former for brevity.

The likelihood function we use is, as always, a Gaussian

$$p\left(\hat{y}_i, \alpha^2 \mid \tilde{\mathbf{x}}_i, \mathbf{w}\right) = \mathcal{N}(\hat{y}_i|y_\mu(\mathbf{x}_i, \mathbf{w}), \alpha^2|y_\alpha(\mathbf{x}_i, \mathbf{w})), \tag{5.80}$$

where we have assumed a two-headed output architecture of the network that yields both an estimate for the mean and the -variance, as described in 5.6.4.

5.7.1 Prediction

In the previous section we outlined how we can obtain a range of predictions $\{\hat{y}_{i,j}\}_j^{N_{mc}}$ for a given data point $\mathbf{x}_i$. This is essentially an esemble of predictions from multiple neural networks. We can obtain a final prediction $\bar{y}_i$ for the true target y_i by choosing a central tendency (see Section 4.9) and use it to evaluate the posterior predictive distribution. For a symmetric distribution all the central tendencies are the same. So if the posterior predictive distribution tends towards a symmetric Gaussian mixture for a large number of samples, then we may simply opt for the mean of the samples that we obtained by Algorithm 2,

$$\bar{y}_i = \frac{1}{N_{mc}} \sum_j^{N_{mc}} \hat{y}_{i,j}, \qquad \hat{y}_{i,j} \in \{\hat{y}_{i,j}\}_j^{N_{mc}}. \tag{5.81}$$

This would be our final prediction for y_i. We shall see in Chapter 6 that we will also be dealing with non-symmetric distributions which require some more consideration.

Equation (5.81) is a more general approach, and has some redundancy in that we know that the distributions for each forward-pass are normalized Gaussians. So the mean of all the samples from the multiple Gaussians will converge to the mean of the means as a function of number of samples. We can consider each forward-pass, either with multiple Gaussian samples $N_l > 1$ or not, as an approximation to that specific Gaussian. So a set of N_{mc} is a set of many approximate Gaussians which together constitute an approximate *Gaussian mixture*. The mean of a Gaussian mixture is the mean of all the component means[26], so instead of sampling as we did in algorithm 2, we can just fetch the output from the network directly, that is $\hat{y}_\mu(\mathbf{x}_i, \mathbf{w}_{q,j})$, for each forward-pass N_{mc} number of times. This would then give Algorithm 3.

Mean of means
For data point $(\hat{y}, \tilde{\mathbf{x}}) = (\hat{y}_i, \tilde{\mathbf{x}}_i)$
1: **for** j in number of forward-passes N_{mc} **do**
2: Sample weights $\mathbf{w}_{q,j} \sim q(\mathbf{w} \mid \boldsymbol{\theta})$
3: Fetch network output $\hat{y}_\mu(\mathbf{x}_i, \mathbf{w}_{q,j})$
4: **end for**.

Algorithm 3: Sampling of means from the multiple forward-passes of the BNN

The final prediction is then

$$\bar{y}_i = \frac{1}{N_{mc}} \sum_j^{N_{mc}} \hat{y}_{i,j}^\mu \qquad , \hat{y}_{i,j}^\mu \in \{\hat{y}_\mu(\mathbf{x}_i, \mathbf{w}_j)\}_j^{N_{mc}}. \tag{5.82}$$

Note, that by only using algorithm 3 the full uncertainty is lost, as it does not capture the variance of each Gaussian distribution, only the variance in the set of $\hat{y}^\mu$ predictions.

5.7.2 Uncertainty Quantification

To gauge the uncertainty of a final prediction $\bar{y}_i$ we can simply find the prediction intervals as described in Section 4.9, where we described the equal-tailed interval (ETI) about the median. The probability density that we calculate the prediction intervals from is the posterior predictive distribution produced by algorithm 2. This is the most straight forward approach typical in Bayesian inference.

Another way we can quantify the uncertainty is to exploit the fact that our network outputs the means $\hat{y}_\mu(\mathbf{x}_i, \mathbf{w}_{q,j})$ and the standard deviations $\alpha_i = \ln(1 + \exp(\hat{y}_{i,\alpha}))$ separately. This also allows us to evaluate two types of uncertainty. As a consequence of the Gaussian likelihood used in Equation (5.80) the set of samples constitute a Gaussian mixture. We assume a given data point i in the ensuing notation. If for every neural network we sample, that is $\mathbf{w}_{q,j} \sim q(\mathbf{w} \mid \boldsymbol{\theta})$, we store both the means $\hat{y}_{\mu,j}$ and variances α_j^2 outputted, then we can calculate the variance of the Gaussian mixture over N_{mc} and its components.

Aleoteric Uncertainty The variance component of the mixture variance is simply the mean of variances

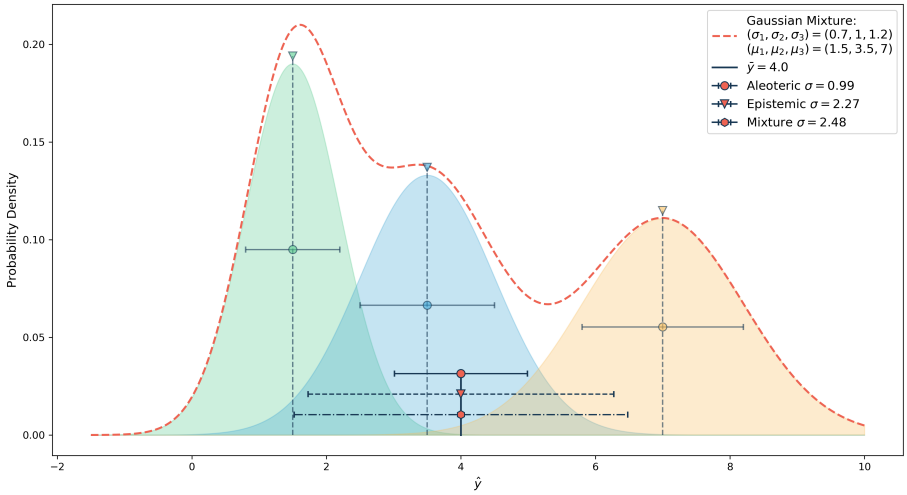


Figure 5.5: Example of a univariate Gaussian mixture distribution with three univariate Gaussian components, all weighted equally. Aleoteric, Epistemic and Mixture standard deviations are depicted with their respective symbols.

Aleoteric Uncertainty Estimate

$$\sigma_a = \sqrt{\frac{1}{N_{mc}} \sum_j \alpha_j^2}. \tag{5.83}$$

This component explains irreducible noise contribution to the uncertainty and is a measure of *Aleoteric uncertainty*. The name aleoteric comes from the Latin word *alea*, meaning dice [63], and fittingly for its name it is the uncertainty that captures the random noise of the data. Specifically it is the uncertainty in predictions that is incurred by the irreducible noise ϵ, which we described in Chapter 2. This is an uncertainty we cannot hope to reduce even if the model has been trained on an infinite number of data points. Assessing aleoteric uncertainty in Bayesian deep learning is not an easy task, and it is an ongoing topic of research, Equation (5.83) is thus one estimate of aleoteric uncertainty.

Epistemic Uncertainty The mean component of the mixture variance is simply the variance of the means

Epistemic Uncertainty Estimate

$$\sigma_e = \sqrt{\frac{1}{N_{mc}} \sum_j^{N_{mc}} (\hat{y}_{\mu,j}^2 - \bar{y}_\mu^2)}. \tag{5.84}$$

This component is the model dependent uncertainty. The name epistemic comes from the Greek word *episteme*, meaning knowledge. This is thus the uncertainty that captures the confident(or lack thereof) we have in our predictions based on our degree of knowledge about the underlying relationship we are attempting to model. The training

data is essentially the quantification of this knowledge and the uncertainty estimate is a measure of its sufficiency. If our prediction on a test data point $\hat{y}_i = f(\tilde{\mathbf{x}}_i)$, where $f(.)$ is the model, is in a region not "seen" by the model, because it is far from the training data $\mathcal{D}_{train} = (\mathbf{X}, \mathbf{y})$, then we should expect epistemic uncertainty to be high. This is specifically because the model has little to no knowledge about the relationships in that data region. If we had an infinite amount of data spread out throughout the whole of sampling space of the data, then we should expect this form of uncertainty to be zero. We can use Equation (5.84) to obtain a measure of epistemic uncertainty.

Mixture Variance The variance of a Gaussian mixture is the component addition of the aleoteric estimate from Equation (5.83) and the epistemic estimates from (see Equation (5.84))

Mixture Variance

$$\sigma_{mix} = \sqrt{\sigma_a^2 + \sigma_e^2}. \tag{5.85}$$

Figure 5.5 shows an example of a Gaussian mixture with the aleoteric, epistemic and mixture variances depicted.

5.8 Prior Distributions

Neural networks can be described as black box models, in the sense that it is difficult to understand exactly why a certain structure of weights manages to approximate a function, and often there can be millions of adjustable parameters. Networks used for practical purposes may also number in the millions in terms of weights and biases, making interpretability futile. However, being able to decipher the inner workings of a network is not always necessary for a deterministic network. But for Bayesian inference one of the main challenges is to find a prior that best suits the problem at hand. As the prior distribution is set on the weights it may be hard to obtain an intuitive understanding of exactly what makes a good prior.

In Bayesian inference we should necessarily pick a prior that mirror our prior beliefs. There is *a lot* that can be said about priors within the field of Bayesian inference. There are mainly two broad categories of priors: *informative prior* and *non-informative priors*.

Non-Informative Prior When one has no prior information available, then non-informative priors are various ways to quantify the most naive assumptions that can be made about a problem. There are several approaches that can be used for a non-informative prior, such as maximum entropy. For example, a six-sided dice could for all we know be heavier on one of the sides, but without any prior knowledge we may rely on the maximum entropy, which would yield a uniform distribution with $p(x_i) = \frac{1}{6}$. This can be shown by using the principles of Section 5.3.1, specifically Equation (5.18) (See [41] for derivation).

Informative Prior In this study we shall use informative priors and empirical priors, the latter is covered in Section 5.9. An informative prior is as its name suggest, a prior constructed based on prior information about a problem or related problems, but

before seeing the data. Specifically, it concerns picking a prior distribution that has a preference for certain domains of the parameter space. The isotropic Gaussian that we encountered in Equation (4.19) and Equation (5.4) is an example of such a prior, as it is has a preference for values around $\mathbf{w} = \mathbf{0}$. This is the most widely used prior in BNNs [64], which is not just a lazy coincidence, but stems from the fact that if one plots the weight distribution of a trained deterministic neural network they have a tendency to be symmetric around zero with the tails dropping off in what may resemble a Gaussian. We shall adopt this type of prior in our experiments and try out a range of different variances τ^2.

5.8.1 Fat-Tailed Prior

Continuing the discussion from the previous section. Is it indeed reasonable to assume that neural network weights are normally distributed? In [64] they challenge this assumption and demonstrates, although for classification, that the distribution of weights depends on what sort of neural network is implemented, such as a convolutional neural network or a fully connected neural network. The neural network structure we implement in this thesis is a feed-forward neural network, and in this case they suggest that fat tailed prior distributions that are symmetric about zero are generally superior to isotropic Gaussians. There are multiple ways one can instate a fat tailed prior, examples are the student's t-distribution, Laplace distribution, or a Gaussian mixture distribution. Univariate types of the two latter ones are depicted in Figure 5.6, however we shall only use the Laplace distribution, in addition to the Gaussian distribution, in this thesis.

In the context of the above discussion it is worth noting that in a BNN we implement a distribution of weights for every parameter location in the network, that would analogously be just a single value in a DNN. Thus as the distributions of a BNN evolve during training they may be completely different at one location of the network architecture versus another location. Therefore the distribution containing all the single valued MLE estimates from a DNN only serves to give an idea of the *average* of all distributions of a BNN, but that is indeed a good starting point for a general prior.

Laplace Distribution The Laplace distribution is listed in the appendix, and depicted in figure 5.6. The Laplace distributon's equation resembles the Gaussian distribution, but as previously mentioned it uses the L1 norm as an argument for the exponential instead of the L2 norm. As a consquence of using the L2 norm it can also be considered two exponential distribution concatenated at zero. The Laplace distribution satisfies our desire of wanting fatter tails in our prior than a single Gaussian distribution can provide. Note that when we use it as a prior distributions it is not minimized, so the behaviour of the gradient, as described in the context of the MAE cost function in Section 3.3, is not of concern.

5.9 Deterministic Pretraining

Until now we have assumed the weights of the prior to be non-trainable, essentially serving as initial conditions. It is possible to make the weights of a prior trainable while training the BNN, which is called *type 2 maximum likelihood* or *empirical Bayes* method [65]. This violates the principle in Bayesian inference in which the prior should be independent of the data. We have already seen an example of an empirical approach

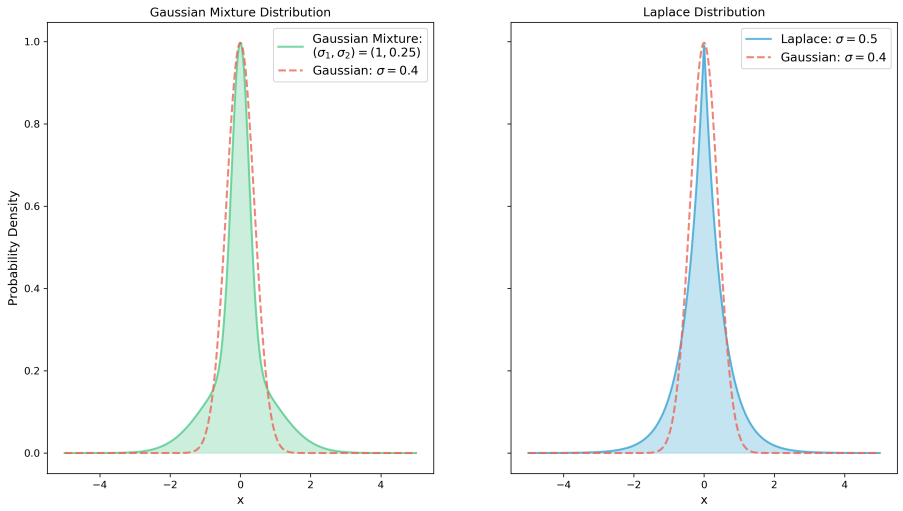


Figure 5.6: Univariate Gaussian mixture distribution and univariate Laplace distribution, both with $\mu = 0$ and standard deviations and scale parameter (both as σ) as listed in the figure. The two components of the Gaussian Mixture are weighed equally. Univariate Gaussian distributions are illustrated for comparison.

in Section 4.7, where we substituted the posterior for the MAP estimate, and where the uncertainty in posterior predictive distribution was mainly provided by the likelihood. Learning the prior would be "more Bayesian" than a MAP-learning approach, as we would still attempt to approximate the true posterior. In Bayesian inference an empirical Bayes approach is usually attempted to simplify difficult inference problems at the cost of potentially underestimating the uncertainty. For a BNN on the other hand a trainable prior does not in an obvious way simplify the problem [42], it may in fact make it more difficult to assess. As part of the training procedure the prior parameters may fit early on to parameters that minimize the complexity cost, $\mathbb{KL}[q(\mathbf{w}|\boldsymbol{\theta})\|p(\mathbf{w})] = \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})}\left[\log \frac{q(\mathbf{w}|\boldsymbol{\theta})}{p(\mathbf{w})}\right]$, by maximizing the denominator. The variational posterior on the other hand is also the base at which the likelihood obtains its samples, so it has more "to consider". As such the prior parameters may minimize initially the complexity cost to such an extent that the complexity cost becomes insensitive to changes in the posterior parameters, breaking the network's ability to learn.

However, we shall attempt a different variant of Empirical bayes more suited to a variational BNN. Successful convergence of the training procedure of a BNN is highly sensitive to prior and posterior weight initialization, so that despite running a training session for a large number of epochs the model may still fail to predict anything reasonable. The uncertainty on predictions might be giving an accurate picture of the misplaced means of the distribution, but this is of no use if the BNN does not manage to predict any relationship between the feature data, so the design matrix $\mathbf{X}$, and the target vector $\mathbf{y}$. Instead of allowing the prior weight to vary during training as aforementioned, we may make an estimated guess placing the prior weight initialization in the proximity of the optimal weights. These optimal weights are estimated by MLE, that is, from the weights of a DNN. This modeling approach constitutes a two-stage hierarchical method. We follow the scheme outlined in [65], named *MOdel Priors with*

Empirical Bayes using DNN (MOPED)[5].

Choosing a MFVI (see Section 5.6.1) setup for a BNN, where the Gaussian covariances are zero, means a BNN will have exactly twice as many weights as a DNN with the same architecture. Taking this as our base we outline the procedure as follows

1. A DNN is first trained according to the methods of Chapter 3. The DNN architecture of nodes and layers is equivalent to the MFVI BNN architecture which it will initialize at the next stage. This means that the DNN should have the same number of weights $\mathbf{w}_{MLE}$ as there are means $\boldsymbol{\mu}$ in the BNN. We minimize the DNN using the likelihood cost (see 5.4.2), meaning that we peform a negative log-likelihood of the MFVI Gaussian likelihood as in Section 3.3 with a variable variance, giving an MSE-like cost function.

2. Next we initialize the prior distributions as a factorized Gaussian,

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}_{\text{MLE}}, \tau^2\mathbf{I}), \tag{5.86}$$

   where typically $\tau^2 = 1$ but we may try other values as well. A fat tailed prior (see Section 5.8.1) will also be tried, specifically the Laplace distribution, also with $\boldsymbol{\mu} = \mathbf{w}_{MLE}$.

3. Finally the initial values for the variational posterior is set as

$$q(\mathbf{w}|\boldsymbol{\theta}) = \mathcal{N}\left(\mathbf{w}_{\text{MLE}}.\boldsymbol{\sigma}\right) \tag{5.87}$$

   Here $\boldsymbol{\sigma} = \log(1 + \exp(\boldsymbol{\rho}))$ and the elements ρ_i of $\boldsymbol{\rho}$ are sampled according to

$$\rho_i \sim \mathcal{N}(\bar{\rho}, \Delta\rho), \tag{5.88}$$

   where $\bar{\rho}$ and $\Delta\rho$ are hyperparameters. We shall also use the second scheme proposed by the authors of MOPED. They argue that the initialization in Equation (5.88) can be too restrictive for complex architectures with many parameters, as the hyperparameters are the same across all weights. The second scheme proposes the following initialization,

$$\boldsymbol{\rho} = \delta|\mathbf{w}_{\text{MLE}}|. \tag{5.89}$$

   Where the bars signify the absolute value to ensure positivity. The coefficent δ is a hyperparameter, called the *initial pertubation*, which is the same for all variances. This is the scheme we will use in this thesis and present in the results in Chapter 7

We note that this procedure is more computationally costly than just a BNN as it needs to train both a DNN and a BNN, however the BNN will persumably converge faster than without pretraining. The DNN is also suitable for hyperparameter optimization, such as choosing the best learning rate and activation function. The optimal configuration of these choice might not be the same for the DNN and the BNN, thus leading to more quantities to be adjusted. This is not necessarily a significant issues as it is not essential that the DNN produces the best possible results. The DNN is supposed to provide initial values in a domain of parameter space that ensures convergence of the BNN to optimal weight parameters.

[5] A name giving even "sparticles" and "strangeness" competition.

Part III

Experiments and Results

Chapter 6

Data Generation and Analysis

In this chapter we will cover the data generation process, how we prepare the data, what uncertainty based performance measures are used, and lastly some notes on efficiency of computation. All scripts needed to train, test and plot results from the BNN models are available at https://github.com/perdimit/master_thesis.

6.1 Data Generation

The data used to train, validate and test BNN models comes from the supersymmetric model MSSM-4, covered in Section 1.8.1. In the same section we outlined how we initialize the majority of the free parameters in the MSSM to constant values that are such that most sparticles decouples for current experiments. The four remaining parameters that will be varied are $M_1, M_2, \mu,$ tan and β. These will serve as our features at the first step of the process, however, for the data generation in `Prospino` and for the BNN training we use derived features, specifically the relevant mixing matrix elements (see Section 1.7.6) and derived masses. The target variables will be the NLO cross sections σ_{NLO} of the electroweakino pair production, specifically we shall focus on the cross sections of the following lightest neutralinos and charginos $\widetilde{\chi}_1^0$, $\widetilde{\chi}_2^0$, $\widetilde{\chi}_1^+$, and specifically the cross sections

$$\sigma(pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^+), \qquad \sigma(pp \to \widetilde{\chi}_1^0 \widetilde{\chi}_1^0), \tag{6.1}$$

where we have chosen to only study the process with the positive chargino χ_1^+. The reasoning behind the choice of targets is firstly that the former of the two cross sections is being actively searched for with the ATLAS detector [2], and the latter is the potential LSP production process that can be reasoned to exist by detection of the former. These two interactions have already been encountered in Figure 1.4. The second reason is simply that we will have to restrict our study for the purpose of focusing on the performance of the BNN, along with providing a proof of concept of its ability (good or bad).

Sampling To generate data for the MSSM-4 the four parameters were first sampled. The choice of distribution to sample from follows Bayesian reasoning (see Section 4.2), that is, the sampling distribution is a prior probability distribution $p(x)$ that accounts for prior beliefs about the problem. In this case a non-informative prior is chosen, as described in Section 5.8, specifically the uniform distribution. We noted previously

Parameter	Minimum	Maximum	Priors
M_1	-1 TeV	1 TeV	uniform
M_2	0 TeV	1 TeV	uniform
μ	-1 TeV	1 TeV	uniform
$\tan\beta$	1	70	uniform

Table 6.1: Sampling intervals of the four MSSM-4 parameters. All other parameters of the MSSM-4 are fixed and described in Section 1.8.1.

that this distribution can be derived using the principle of maximum entropy with a normalizing condition as shown with equation Equation (5.18). It should be noted that more informed choices can be made regarding the form of the sampling distributions of the relevant parameters (see [15]). The physical information about the problem is introduced through the limits of the prior distribution. The choices of parameter intervals are summarized in Table 6.1. The intervals of the bino, wino and Higgs mass parameters M_1, M_2, and μ, respectively, are chosen such that they will yield neutralino and chargino masses at the electroweak scale up to $\sim 0,1$ TeV. This sets the focus on lighter neutralinos and charginos. The reasoning for this choice is that it mimics the physics of what can be expected at the Large Hadron Collider, where heavy electroweakinos are expected to have very small cross sections. Note that both M_1 and M_2 can take on negative parameter values, but a phase rotation of the fields in the Lagrangian in Equation (1.42) allows one of the two signs to be remove without altering the physics. It is thus arbitrary which of the two parameters we hold strictly positive, and the arbitrary choice falls on M_2. The three mass parameters are, as is apparent from Table 6.1, dimensionful parameters and will depend on the renormalization scale. This is set to to $Q = 1$ TeV, as already noted in Section 1.8.1. The remaining parameter $\tan\beta$ is restricted by multiple conditions, where if violated produces nonphysical models. The conditions stems from the fact that masses of quarks and lepton are dependent not only on their Yukawa couplings in the superpotential, but also on $\tan\beta$ through the relevant mass matrices [14]. At tree level we have of most relevance

$$m_t = y_t v \sin\beta, \quad m_b = y_b v \cos\beta, \quad m_\tau = y_\tau v, \cos\beta, \tag{6.2}$$

where m_t, m_b are the top and bottom quarks and m_τ tau lepton. For the coupling y_t to not become excessively large above the electroweak scale $\tan\beta \geq 1$ is reasonable, depending on the top quark mass m_t. Similarly requiring that y_b and y_τ also is kept at bay a rough upper bound $\tan\beta < 70$ should do [15].

Spectrum Calculations To calculate the supersymmetric particle spectra the programming package `SOFTSUSY 4.1.5` [66] was used. The fixed parameters of the MSSM-4 were first set. Next, a single calculation is conducted by taking one sample for each of the four variable parameters from their respective uniform distributions and passing them to `SOFTSUSY 4.1.5`. This then yields all relevant masses, couplings and mixing matrices, which are written to a SLHA-file [67] corresponding to one data point (to parse a SLHA-file the program `PySLHA` is used [68]). Subsequently the SLHA-file is fed as input to `Prospino 2.1` which then calculates the LO and NLO cross sections. We note that all cross section calculations are performed with $\sqrt{s} = 13$ TeV center of mass energy. The data contains in total 12101 SLHA files, each with a multitude of cross sections of various final state particles. However, as already mentioned we restrict our study to the two in Equation (6.1). This is a small data set relative to the data sets used in parameter scans with the purpose of searching for new physics. The number of data points is severely limited by the expense of calculating cross sections, That said,

Target	Features		
Cross Section	Masses	Neutralino Mixing	Chargino Mixing
$\sigma\left(pp \to \widetilde{\chi}_1^0 \widetilde{\chi}_1^0\right)$	$m_{\widetilde{\chi}_1^0}$	$N_{11}, N_{12}, N_{13}, N_{14}$	
$\sigma\left(pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^+\right)$	$m_{\widetilde{\chi}_2^0}, m_{\widetilde{\chi}_1^+}$	$N_{21}, N_{22}, N_{23}, N_{24}$	V_{11}, V_{12}

Table 6.2: The targets and derived features that were used to generate the data, and used for cross section predictions with variational BNN.

it should be sufficient for our objective of evaluating the quality of a BNN model for the purpose of supersymmetry predictions.

Derived Features Note that the initial four features are not used explicitly to produce the training data, as using Lagrangian parameters explicitly would make the resulting cross section scale-dependent. As such, to reproduce the cross section results from `Prospino 2.1` in a BNN, as we shall get to in Chapter 7, it is reasonable to use the derived features that go into the cross section calculation to establish the feature-target relationship, namely the masses and mixing matrix elements. We summarize the features used for the two cross sections in Table 6.2. One could suggest that we just use every feature available to us, that is, both the original MSSM-4 parameters and the derived features. However, MSSM-4 parameters are strongly correlated with the derived features, such that by using all available features it may produce *multicollinearity*. This means essentially what we have already suggested, that the features are highly linearly dependent. Multicollinearity can lead to issues if it is sufficiently pronounced, such as reducing our ability to assess the predictive power of each individual feature, so plotting the mass against the predicted cross section values would not necessarily give the true dependence. However, it does not reduce a model's ability to predict for the feature set as a whole, and more importantly, neural networks are robust against any problems associated with *linear relationships* thanks to its non-linear nature. Additionally this is a BNN, and a common approach to reduce multicollinearity is regularization, which exists naturally in a Bayesian model as described in Chapter 4. Apart from it being interesting to see if multicollinearity would even be a problem, we see no other gain in introducing information to the model that should already exist in the derived features, and hence we stick to the derived features in Table 6.2.

Preprocessing The sampling ranges chosen will, except for $\tan\beta$, include samples with masses close to zero. `Prospino 2.1` has a limitation in that it does not allow the calculation of cross sections with final state particles that together have a mass below the $Z-$mass, here set to $m_Z = 91.188$ GeV. This is because it is written to avoid the consideration of Feynman diagrams with an on-shell Z. The lightest neutralino can have a mass below half that of the Z mass, $m_{\chi_1^0} < m_Z/2$, such that these cross sections will simply output the unphysical $\sigma_{LO/NLO} = -1$. The corresponding SLHA files are removed from the data set such that the remaining data has 10503 data points.

Sanity Check In Figure 6.1 we have plotted a histogram of the parameter samples, and as expected it mostly follows the uniform distribution. However, note that it parts from the uniform distribution towards parameter values close to zero, as in this region

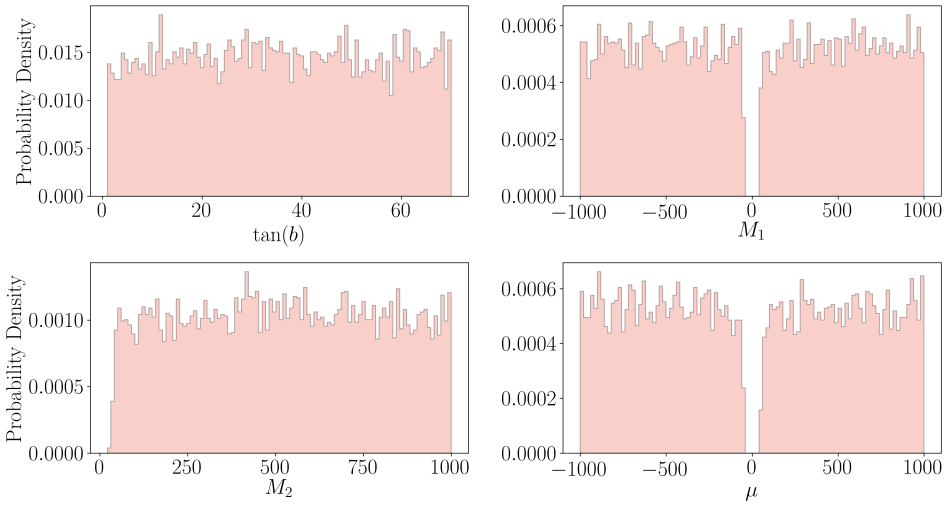


Figure 6.1: Sampling distribution of parameters M_1, M_2, μ and $\tan\beta$. All four parameters are sampled from a uniform distributions with various limits described in Section 6.1. Unphysical values have been removed producing the drop off seen close to zero for all but $\tan\beta$.

the parameters yield the unphysical cross sections described above and is removed here. To get an intuition for the relationship between the initial features of the data and the targets we show in Figure 6.2 scatter plots with the two targets against the initial features. An interesting aspect of the plots is that we can expect $\tan\beta$ to lend the least amount of help as a feature in terms of predictive power, as a correlation is hard to spot.

One may also spot a small number of data points that are up to several orders of magnitude below the bulk of the data. These points are not erroneous calculations, but the result of mere chance due to accidental calculations. As a neural network generally functions better the more data it is fed, we should not expect it to perform well in data scarce territory. It is common practice in machine learning to remove such outliers (or gather more data), however, for our purpose of evaluating a BNN these points may serve as an informal test of the quality of the uncertainty (bad predictions should give high uncertainty).

Furthermore, we plot the masses of various electroweakinos to see if `SOFTSUSY 4.1.5` has produced them in the electroweak energy range, and indeed it has as seen in Figure 6.3. Additionally we see that the four neutralinos are ordered by steadily increasing mass (as defined in Section 1.7.6) and that the lightest of them is lighter than the lightest chargino, which is good or else the LSP would be charged and not an eligible WIMP candidate.

6.2 Validation

In Chapter 2 we elaborated on the necessity of splitting the data set into a training and test set. We have done just that, and the test set amounts to 20% of the total data. Additionally we mentioned the need for a validation set: Here we shall use *two* validation sets, and as we noted in Chapter 2, we have one set for the early stopping

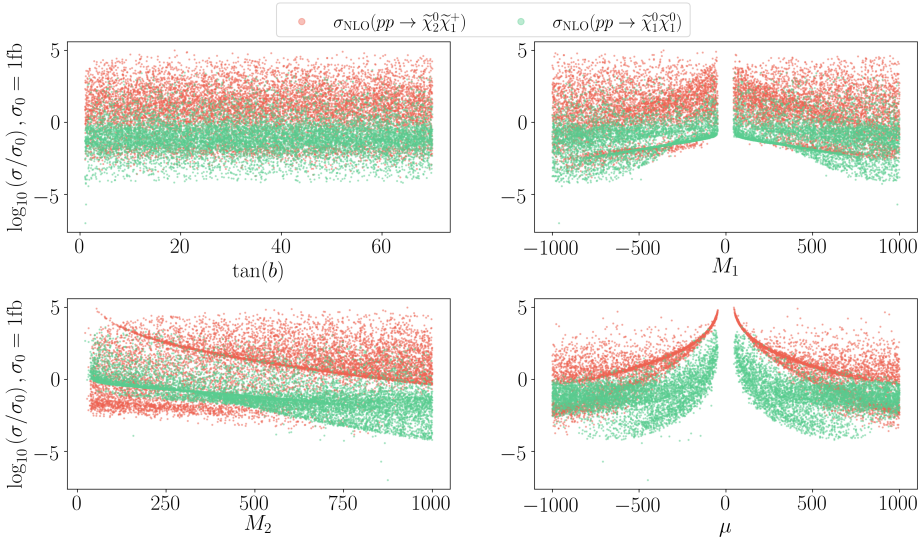


Figure 6.2: Scatter plot between the four parameters of the MSSM-4 and the two cross sections of interest.

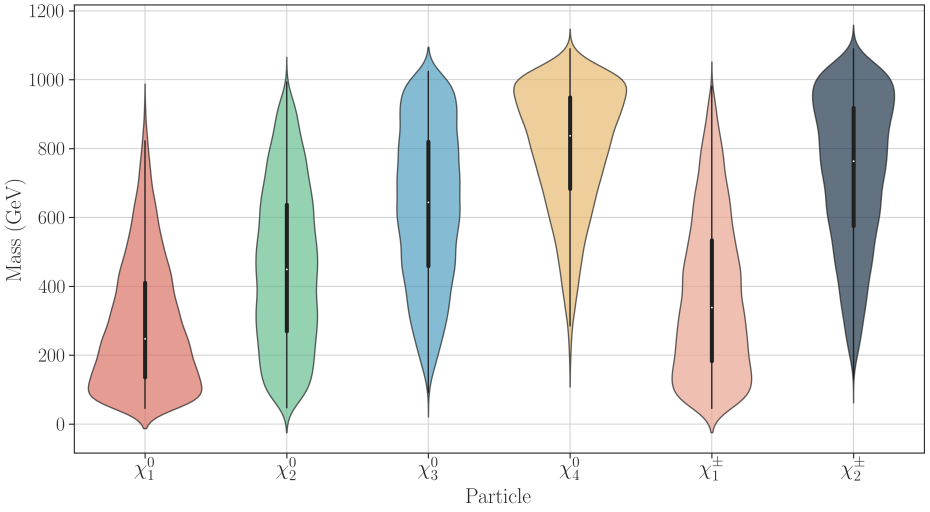


Figure 6.3: Violin plot showing the distributions of neutralino and chargino masses in the data. The various masses results from various parameter configurations and `SOFTSUY 4.1.5` calculations.

procedure which we shall abbreviate as "m", so the early stopping feature data would then be $\mathbf{X}_{\text{es}}$. Additionally we use one validation set for model validation, which we shall abbreviate as simply "m", so we have $\mathbf{X}_{\text{m}}$. By model validation we mean the process of searching for the optimal configuration of hyperparameters (including posterior and prior initializations, and annealing schemes), which we here denote $\boldsymbol{\lambda}_i$. We summarize the fraction of the total data in each data set in Table 6.3.

The overall process of training, validating and testing will proceed as follows

1. For a given $\boldsymbol{\lambda}_i$ we train our model on $\mathbf{X}_{\text{train}}$, and after each epoch we run our model

Data set	$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{es}}$	$\mathcal{D}_{\text{m}}$	$\mathcal{D}_{\text{test}}$
Fraction	50%	15%	15%	20%

Table 6.3: The fraction of the total data set $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ that each subset receives.

(a feedforward pass) on $\mathbf{X}_{\text{es}}$ and calculate the ELBO[1] as a metric for validation, specifically we calculate $\text{ELBO}(\mathbf{y}_{\text{es}}, \hat{\mathbf{y}}, \mathbf{w})$ sampling the resulting BNN *once*. Note, this calculation is in addition to the ELBO loss calculation on each iteration. As explained in Section 2.5, we set a patience to indicate the number of epochs we allow the algorithm to continue without achieving new ELBO lows.

2. When the training run is done we calculate the ELBO on the second validation set, $\text{ELBO}(\mathbf{y}_{\text{m}}, \bar{\mathbf{y}}, \mathbf{w})$, but now with a full set of 2000 $\hat{\mathbf{y}}_i$ samples from the BNN. These are then averaged over for each data point giving $\bar{\mathbf{y}}$. Given the size of this data set, the total number of samples amounts to 3.15×10^6. We store the score and execute training again, but now with new hyperparameters $\boldsymbol{\lambda}_{i+1}$.

3. When we have finished evaluating different BNN models we choose the model with the best performance on the second validation set. Now as the validation data has been "seen" we can join the sets into a final training set $\mathcal{D}_{\text{train}_2} = \mathcal{D}_{\text{train}} + \mathcal{D}_{\text{es}} + \mathcal{D}_{\text{m}}$. However, after we have randomized the new training set, we need to once again create a new $\mathcal{D}_{\text{es}}$, as the size of the new training data set is larger thus we cannot simply use the optimal epoch obtained from validation. Finally we train BNN once more with early stopping and the optimal hyperparameter configurations and evaluate the final model on $\mathcal{D}_{\text{test}} = (\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}})$.

Ideally we would have used *cross-validation* [24], however, since we have a massive number of hyperparameters to attend to (we will summarize them in Chapter 7) we considered this unfeasible. The drawback to this choice is that performance metrics on the different data sets will be more sensitive to the random draw of the data. This could be mitigated by removing points that are highly deviant from the bulk of the data, but as we argued in Section 6.1 we will keep them in. A way we may increase the probability that the data sets are representative is by reducing the share of training points and consequentially increase it for the other sets. This is exactly the thought behind choosing 50% training data, as opposed to the standard 70% or 80% (see Chapter 2).

6.3 Scaling

It is common practice in machine learning to scale the data before training, and thus this section could fit well into Chapter 2, but as the motivation for our specific scaling implementation involves concepts from all the chapters that followed, we introduce it here. We implemented two types of data scaling which we will now present.

Feature scaling The first type of scaling is motivated by our use of gradient descent based algorithms. These algorithms are dependent on the scale of the features, and thus the group of scaling methods used in this context are usually categorized under

[1] Because of its succinctness we will use the term ELBO from here on when referring to the variational free energy, even though they have a sign difference

the name *feature scaling*. The simple gradient descent formula in Equation (3.14) relies on the gradient of the cost function in order to make a step in parameter space. Each weight is updated according to its corresponding gradient component, that is its partial derivative. The components corresponding to the first layer will depend on the feature data through partials such as $\frac{\partial(\sum_{i=1}^{n_x} w_{ij}x_i)}{\partial w_i j}$, where x_i is the feature input, so that if two features differ greatly in scale the size of their corresponding gradient steps will differ greatly as well, and the resulting effect is slower convergence. Feature scaling normalizes the feature data so that different features reside on the same scale. The classic illustration is imagining an elliptical cost surface with dimensions w_0 and w_1, and the center being the global minimum. If the algorithm happens to traverse along the major axis then this may be costly in terms of iterations used until convergence. By scaling we transform the error surface closer to a circular shape, where all paths to the center are equally long.

The feature scaling technique that we found most suited for our problem is called *min-max scaling*, also known as *min-max normalization* or simply *rescaling*. This technique transforms the feature data into the interval $(0, 1)$ by transforming every $\mathbf{X}_j$ independently, where j runs over every feature column of data. So for a given feature column j every data point x_i in that column is transformed as

Min-Max Scaling

$$x_i' = \mathrm{M}(x_i) = \frac{x_i - \min(\mathbf{X}_j)}{\max(\mathbf{X}_j) - \min(\mathbf{X}_j)}, \tag{6.3}$$

where the prime denotes a scaled quantity. Another popular scaling technique is called *standardization* and transforms each x_i in $\mathbf{X}_j$ according to

$$x_i' = \frac{x_i - \mu_j}{\sigma_j}, \tag{6.4}$$

where μ_j and σ_j are the means and standard deviations of each feature column, respectively. However, we found min-max scaling to be superior in terms of performance and is thus the method used in the results of the next chapter. We omit the comparison as it is not deemed too relevant to the overarching topic of BNNs.

Another reason feature scaling is particularly appropriate for our BNNs, is that our cost function is not merely an MSE cost function, but the ELBO cost function. Generally, cost functions that involve regularization either through a Bayesian prior term or deterministic regularization techniques will have constant terms that treat all variable weights equally, in our case the constant term is represented by the constant prior $p(\mathbf{w})$ and the variable surrogate posterior $q(\mathbf{w} \mid \theta)$, both in $L^C = \mathbb{KL}[q(\mathbf{w} \mid \theta) \| p(\mathbf{w})]$. Hence, if the constant weights of the prior is to treat all variable weights of the posterior equally, the variable weights should all be updated with respect to the same scale.

Target Scaling After having applied feature scaling it is not essential to many machine learning problems to scale the target variables y. For a simple cost function such as the MSE the network will still tune its weights in its effort to minimize the cost $\mathcal{C} = \sum_{i=1}^{n} \left(y_i - \hat{y}(\mathbf{x}_i, \mathbf{w})\right)^2$, regardless of the scale of the features. However, it may be wise to scale the targets in cases where a target variable has a large spread of values in $\mathbf{y}$, so as to stabilize the learning process by avoiding large cost gradients and consequentially large and erratic updates to the weights. As can be seen in Figure 6.4 we do

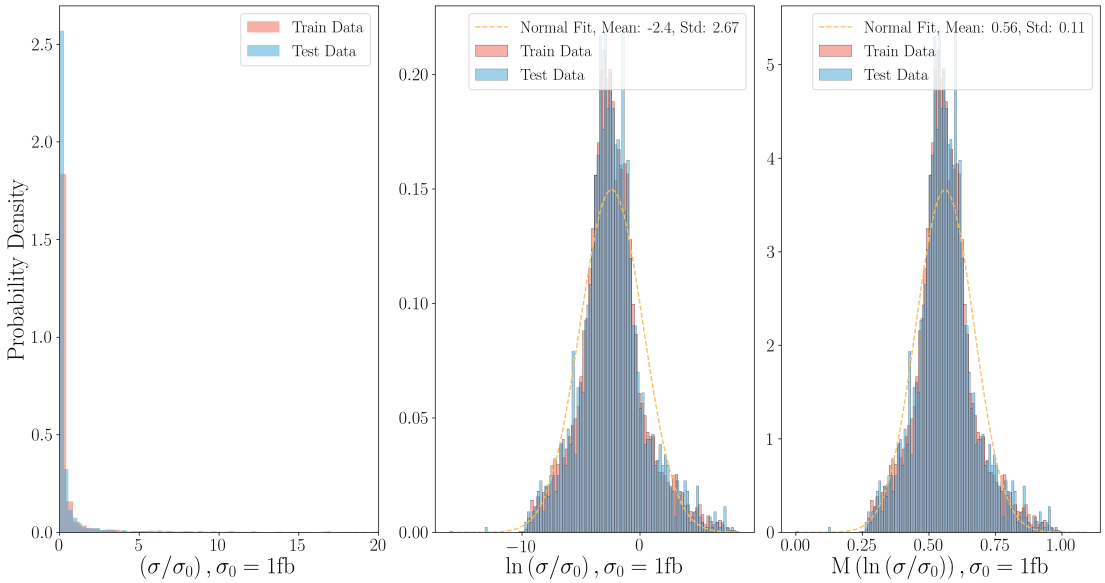


Figure 6.4: Figure shows the transformation of the targets. Left-hand panel shows original data, but truncated at 20 fb. In reality it goes up to ~ 4250 fb. The center-panel shows after a natural log transform, and right-panel shows the min-max scaling. Normal curves have been fitted to the two right most panel with μ and σ calculated from the data.

indeed have a large spread in our target values and they follow a largely asymmetrically distribution. In fact for $\sigma(pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0)$ the difference between the smallest cross section and the largest cross section in our data is 10 orders of magnitude, with the smallest being $\sigma_{\text{min}}(\widetilde{\chi}_1^0\widetilde{\chi}_1^0)$, $\approx 10^{-7}$ fb and the largest $\sigma_{\text{max}}(\widetilde{\chi}_1^0\widetilde{\chi}_1^0)$, $\approx 4.25 \cdot 10^3$ fb. We may add that another possible motivation for target scaling is caused by the ELBO cost function having a weight independent term (the prior) in the same expression as a term akin to the MSE, such as shown in Equation (5.74). The size of the MSE is relative to the scale of the target values, while the prior is not. While a prior suited for the target scale could by principle be chosen, it is perceivable that this may be a hard task as they are heavily skewed.

We obtained the best performance by exploiting the log-normal-like behaviour of the targets[2], meaning that taking the logarithm of the target values gives approximately a normal distribution $\log(\mathbf{y}) \sim \mathcal{N}$, as can be seen in the figure just shown Figure 6.4. However, the target is not log-normal, as it does seem to have power law behaviour. We have not verified this, as what matter to us is the symmetrical normal-like transformed targets apt for BNN. Furthermore we applied the min-max scaling as in Equation (6.3) shifting the normal distribution to the $(0, 1)$ interval. The transformations were applied in the same order as written,

Target Scaling

$$\mathbf{y}' = \mathrm{M}(\log(\mathbf{y})), \tag{6.5}$$

where we use M to denote the piece-wise transformation of every element in the target column. Note, that in regards to having log-normally distributed targets; this is a

[2]However, the target distribution probably has a power law behaviour

case-specific argument, where if we had used a more informative prior for the MSSM-4 parameters, peaked at certain intervals, we would not necessarily have discovered such a convenient data distribution.

We will not report the results for the poor performance of the BNN without target scaling, we simply note that reducing the spread and distribution asymmetry of the target values turned out to be decisive in terms of performance. However, the drawback is that we cannot analyse metrics dependent on the network weights in the fb-scale as these have been fitted in the log-scale. Particularly, this means the ELBO cost function must be seen in context of scaled targets. Additionally, as we shall see in the next paragraph, the manner in which the output variances (see mixture variance in Section 5.7.2) should be inverted is not immediately obvious.

6.3.1 Inverting Scales

After having trained the BNN and made predictions, we would need to apply an inverse transformation on the predictions. Two approaches were used according to: I) whether the uncertainties were calculated through prediction intervals from the posterior predictive distribution, where multiple samples are taken from each likelihood, (see Equation (5.81), or II) if uncertainties were calculated by using the variance and mean output from the network (See Equation (5.82). The latter method utilizes the information we have about the form of the components of the posterior predictive mixture (the Gaussian at the output of a forward-pass).

Method I The first method is performed by simply undoing the transformations of Equation (6.5) in the right order

Inverse Target Scaling

$$\mathbf{y} = \exp(\mathrm{M}^{-1}(\mathbf{y}')) \tag{6.6}$$

$$\hat{\mathbf{Y}} = \exp(\mathrm{M}^{-1}(\hat{\mathbf{Y}}')), \tag{6.7}$$

where we have undone the transformation on the target data $\mathbf{y}'$ from Equation (6.5) and all the sample predictions $\hat{\mathbf{Y}}'$, which is a data-points $\times$ samples matrix of samples. Both the exponential and M^{-1} is applied piece-wise on $\hat{\mathbf{Y}}'$.[3] The inverse min-max transform is simply found by rearranging the original transform

$$\hat{y} = \mathrm{M}^{-1}\left(\hat{y}_i'\right) = [\max(\mathbf{y}) - \min(\mathbf{y})]\,\hat{y}_i' + \min(\mathbf{y}). \tag{6.8}$$

The resulting distribution of samples for a given data point will essentially be a mixture distribution of logged samples, and will therefore be asymmetrical. As the components of the mixture will often be tightly concentrated around a central tendency the mixture bears resemblance to a log-normal distribution, and the smaller the epistemic uncertainty (the variation caused by weight sampling) the closer the mixture, in our experience, was to a log-normal distribution (this was checked with quantile-quantile plots). The uncertainty is then calculated by using prediction intervals, specifically the equal-tailed interval (ETI) as described in Section 4.9. Predictions can be made using the mean, mode or the median (see Section 4.5). In the case of a log-normal distribution the median will lie between the two extremes of the mode and mean, and is thus a natural choice. In our case we have a mixture distribution that tends towards a log-normal.

[3] Note, if we had multiple targets we would have different M^{-1} for each target matrix

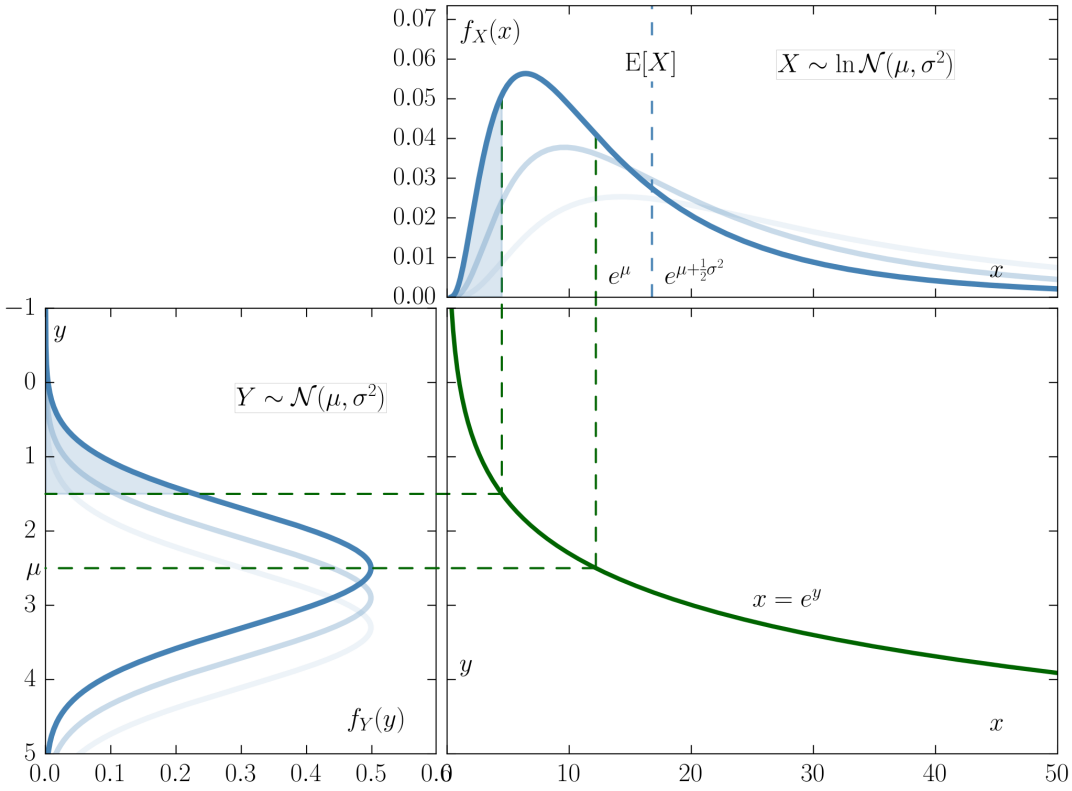


Figure 6.5: Figure showing the relationship between the mean and the median of the normal and log-normal distribution. y here corresponds to BNN output, while x is the inverted target transform. Figure taken from [69].

The mixture has a median we could use, however we shall use an approximation of the median as we will explain shortly.

Method II For the second approach, we only use the means and variances of the likelihood for faster evaluation. Consider again a single test data point i and a Gaussian likelihood at the output of the network. By taking the exponential of a single mean $\hat{y}_\mu$ (the network output), $\mu' = \exp(\hat{y}_\mu)$ we get the *geometric mean* of the individual exponential values, where the following inequality applies

$$\exp(\hat{y}_\mu) = \exp\left(\frac{1}{N}\sum_j^N \hat{y}_j\right) \leq \frac{1}{N}\sum_j^N \exp\left(\hat{y}_j\right). \tag{6.9}$$

Here the left hand side is the geometric mean and the right hand side is the arithmetic mean. The geometric mean of a normal distribution is the *median* of a log-normal distribution. Generally, for a log-normal distribution we have that

$$\mu = \exp\left(\mu' + \frac{\sigma'^2}{2}\right) \tag{6.10}$$

$$\tilde{\mu} = \exp\left(\mu'\right), \tag{6.11}$$

where μ' and σ'^2 is the mean and the variance, respectively, of the corresponding normal distribution. In Figure 6.5 we see an illustration of the relationship in Equation (6.10) and Equation (6.11). The variance about the log-normal median, which is called the

geometric variance, can be expressed by

$$\sigma^2_{\text{GM}} = \exp(\pm\sigma'^2). \tag{6.12}$$

Since we are dealing with an asymmetrical distribution we have positive/negative signs for each side of the median. For a 1σ asymmetric prediction interval around the median, $\bar{y} \pm \Delta^{\pm}_{\text{GM}}$, we have

$$\begin{aligned} \Delta^{-}_{\text{GM}} &= \exp(\mu) - \exp\left(\mu - \sigma'\right), \\ \Delta^{+}_{\text{GM}} &= \exp\left(\mu' + \sigma'\right) - \exp\left(\mu'\right). \end{aligned} \tag{6.13}$$

This may seem fine so far, however our posterior predictive distribution is not a log-normal, it is a log-transformed mixture of normals. The mixture- σ_m, epistemic- σ_e and aleoteric σ_a standard deviations presented in Section 5.7.2 are about the mean of a Gaussian mixture. We have not taken it upon ourselves to discover the analogous expressions, asymmetrically about the median of a log-normal mixture distribution. Instead we use the *geometric mixture mean* (GMM) and *geometric mixture variance* (GMV). This simply means that we follow the exact same prescription as above with the geometric mean and variance. Practically, we *first* calculate the mean of means of multiple outputs (still only considering one data point) and then we exponentiate it

$$\bar{y}_{GMM} = \exp\left(\frac{1}{N_{mc}} \sum_{i}^{N_{mc}} \hat{y}_{\mu}\right). \tag{6.14}$$

We follow the same idea for the variances σ^2_m, σ^2_e and σ^2_a, calculating each separately. That is, we first find the Gaussian mixture variances as in Section 5.7.2 *then* we find the credible intervals according to Equation (6.13).

It is important to note that GMM is *not* the median of the log-normal mixture distribution, however the GMM will be relatively close to the median when the log-normal components of the mixture are concentrated, which they mostly are. When they are not, that is, when uncertainties are great (larger variation in weight samples) the median and the GMM will be further apart. In figure Figure 6.6 we show multiple central tendencies for a simple case of a three component log-normal mixture, along with the σ_{GMM}.

Inverting Min-Max on the Variances To undo the min-max scaling on the variances we cannot simply use M^{-1} as defined in Equation (6.8). Instead we must use the scaling factor, which is the denominator of the original transform

$$\sigma = \text{M}^{-1}_{\sigma}(\sigma') = \frac{\sigma'}{\max(\mathbf{y}) - \min(\mathbf{y})}. \tag{6.15}$$

The final inverse transform of uncertainty estimates is

Inverse Standard Deviation

$$\boldsymbol{\sigma} = \exp(\text{M}^{-1}_{\sigma}(\boldsymbol{\sigma}'), \tag{6.16}$$

where $\boldsymbol{\sigma}$ is a matrix of standard deviation samples.

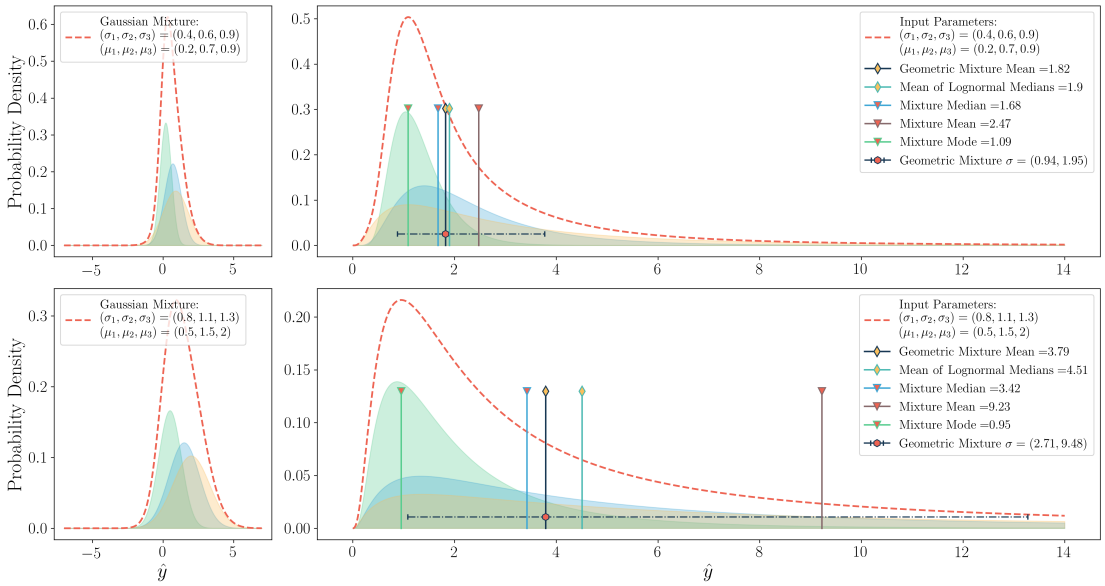


Figure 6.6: Figure showing various central tendencies along with the geometric mixture standard deviation σ_{GMM}. The log-normal components and the central tendencies are calculated using the parameters of the Gaussian mixture in the left pane.

6.4 Performance Measures

We have already presented some of the metrics used to assess performance. That is, the RMSE and R2 from Chapter 2 and the ELBO from Chapter 5. In addition we will use two more measurements to assess the quality of uncertainty prediction.

Coverage Score A measurement frequently used in in Bayesian machine learning is to to use the 95% ETI or HDI interval of the posterior predictive distribution of a validation (or test) point prediction, and in a frequentist fashion check if the true data point falls within or outside the interval. This is then repeated for all data points in the validation set, and if the model is "ideal" 95% of the true validation points should have fallen within their respective 95% intervals. If this is the case we can make the frequentist interpretation that there is a 95% probability that a new observation will lie within the 95% interval of the posterior predictive distribution, given that the validation data observed is sufficiently generalizable. If 93% of the validation points fall within the 95% intervals then by the frequentist interpretation there is a 93% probability that a new observation lies within the interval, and the uncertainties "should" be larger, that is, the posterior predictive distributions should be wider. In this case we say the model is *liberal* and is *underestimating* the uncertainty. On the other hand if 97% of the validation points fall within the 95% intervals then the model is *conservative* as the uncertainties are larger than the validation data would merit and is thus *overestimating* the uncertainty. We should note that this method, as most machine learning methods, rely on having sufficient amount of data in the validation set, and which is sufficiently representative of new data. This is inherently difficult in many dimensional feature spaces.

We will call the "falls within" for *coverage* in the subsequent discussion. We take the method described above one step further and evaluate the coverage for multiple intervals in an equidistant set of percentiles in the range $[1\%, 100\%]$. So for every interval we check

how many percent of the validation points fall within the current prediction interval. We can then plot this curve against the percentiles, and if the model is ideal for every prediction interval it should be linear from the origin to 100%, so that $y = x$ where y is the coverage percentage and x is the prediction interval percentiles. Next, we can calculate the area between the ideal line and the coverage curve. The area above the ideal line is a measure of "coservativeness" that we call *coverage area over score* (CAOS) and the area below the ideal line is a measure of "liberalness" that we call *coverage area under score* (CAUS). Each score is divided by the area above/below the ideal line

$$\text{CAOS} = \frac{\text{Area over ideal line}}{100^2/2}, \tag{6.17}$$

$$\text{CAUS} = \frac{\text{Area under ideal line}}{100^2/2}, \tag{6.18}$$

so each score is in the range $[0, 1]$. The reason why we do not create one single score, say between $(-1, 1)$, is that the model may be conservative and liberal at the same time at different percentiles. Most of the time it will be either conservative or liberal, but for models close to ideal our experience is that it may be liberal at very high percentiles, and conservative at lower percentiles. We show examples of these coverage curves that have just been described in Figure 7.4 and Figure 7.5.

Standardised Residuals To get another idea of the nature of the uncertainties, we evaluated the distribution of the *standardised residuals*. For a given uncertainty estimate, this involves point-wise comparing the uncertainty estimate $\Delta^{\pm}_{GMV}$ with the true deviation between a final model prediction $\bar{y}_i$ against the true datum y_i, for our case the validation/test set. So for a single data point $(\mathbf{x}_i, y_i) \in \mathcal{D}_{\text{m/es}}$, we calculate

$$z_i = \frac{y_i - \bar{y}_i}{\Delta^{\pm,i}_{\text{GMV}}} \tag{6.19}$$

By making the comparison for every validation point we obtain a distribution z of standardised residuals. We will use the geometric variance as the basis for our standardization. The notation $\Delta^{\pm}_{GMV}$ is shorthand for applying Δ^{+}_{GMV} when $\bar{y}_i - y_{\text{i}} > 0$ and Δ^{-}_{GMV} when $\bar{y}_i - y_{\text{i}} < 0$.

Let us for a moment assume that the residuals $\hat{\epsilon} = y - \bar{y}$ obey normality, so that

$$\hat{\epsilon} = \mu_{\hat{\epsilon}} + \sigma_{\hat{\epsilon}} \mathcal{N}(0, 1)\,. \tag{6.20}$$

The ideal situation would be that we have a model that neither over-predicts nor under-predicts. In that case we can expect $\mu_{\hat{\epsilon}} \approx 0$, and thus expect $\mathbb{E}(z) = 0$. Regardless of of normality, this is a desired outcome. Now let us use the normality assumption in the standardised residuals with zero mean

$$z = \frac{\hat{\epsilon}}{\Delta^{\pm}_{\text{GMV}}} = \frac{\sigma_{\hat{\epsilon}}}{\Delta^{\pm}_{\text{GMV}}} \mathcal{N}(0, 1)\,. \tag{6.21}$$

If the variation in the residuals can be fully explained by the uncertainty we would expect that, $\sigma_{\hat{\epsilon}}/\Delta^{\pm}_{\text{GMV}} \approx 1$. In that case the standardised residuals should have a standard normal distribution $z \sim \mathcal{N}(0, 1)$. If $\sigma_{\hat{\epsilon}}/\Delta^{\pm}_{\text{GMV}} > 1$ the uncertainty can be considered liberal as it is smaller than the actual variation in the residuals. On the other hand if $\sigma_{\hat{\epsilon}}/\Delta^{\pm}_{\text{GMV}} < 1$ it can be considered conservative as its larger than the actual variation in the residuals. Normality is not essential for the analysis of the z distribution, rather,

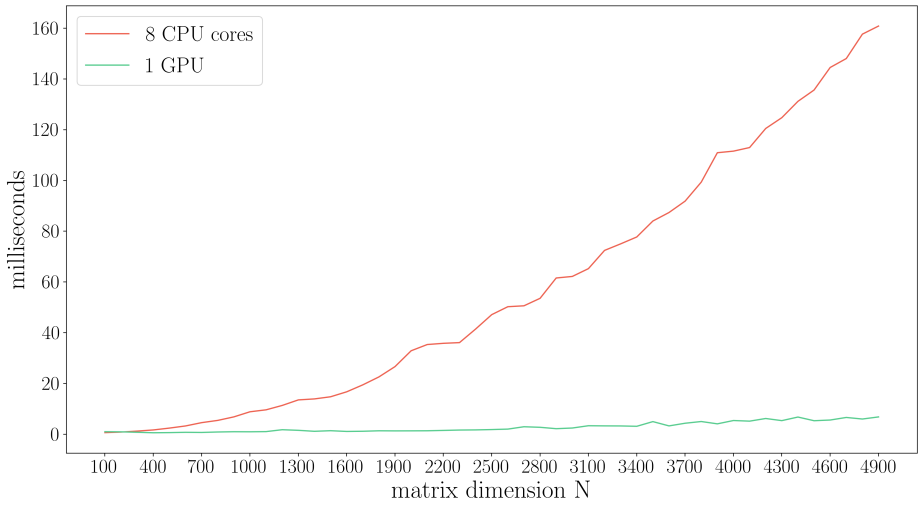


Figure 6.7: Tensorflow matrix multiplication of $(N \times N) \cdot (N \times N)$ with the use of 1 GPU and 8 CPU cores, respectively. This calculation ran on laptop hardware, GPU: `GeForce GTX 1050 Mobile`, CPU: `Intel i7-7700HQ CPU @ 2.80GHz`.

of interest is whether it indeed has a mean at zero, and what its spread is about zero. Furthermore, it is interesting in what regard it strays from normality, that is, if that is the case. When doing multiple hyperparameter tuning runs our program will report the μ_z and σ_z, for the above case with normality these would be: $\mu_z = \mu_{\hat{\epsilon}}/\Delta^{\pm}_{\text{GMV}}$ and $\sigma_z = \sigma_{\hat{\epsilon}}/\Delta^{\pm}_{\text{GMV}}$.

Now, we have three standard deviation type uncertainties σ_m, σ_e and σ_a. We will check the nature of z using each of these three separately in the expression for $\Delta^{\pm}_{\text{GMV}}$. For example, if we only assumed uncertainty from inherent data noise (aleoteric) we would expect $\sigma_{\hat{\epsilon}}/\sigma_a > 1$ as it should not alone explain the residual variation. We will also assess whether it is even necessary to include σ_a as part of σ_m as the actual level of noise in the `Prospino 2.1` training data set is small. That is, there is no added noise, only noise stemming from the stochasticity of the integration algorithm. We should note that the relationship $\sigma_m = \sqrt{\sigma_e^2 + \sigma_a^2}$ has partly broken down at the point of analysis, this is because of the geometric variance approach where they are calculated first then inverted individually. As a consequence, some degree of interpretability is lost, however we can still analyse z on a validation set assuming the use of each one, respectively.

Average Relative Test Error We will when reporting our final test results present the *average relative test error*. This is obtained using the inverse scaled mixture errors, that is, the length of the asymmetric error bars, against the true test value,

$$\text{ARTE} = \sum_i^N \frac{|\Delta^+_{\text{GMV},i} + \Delta^-_{\text{GMV},i}|}{y_i}. \tag{6.22}$$

6.5 Computation

The program written consists of multiple `Python` scripts that rely on the libraries `TensorFlow 2.2.0` [61], `TensorFlow-Probability 0.10.1`, `Scikit-learn` [70] and `SciPy`

[71]. Tensorflow is an open source library made primarily for machine learning with a particular focus on neural networks. It is developed and maintained by Google. Tensorflow allows rapid calculations by its use of stateful dataflow graphs, as well as default parallelization across multiple CPUs. The parallelization can be controlled to some extent, but we found it best to leave it at Tensorflow's default settings.

Tensorflow also provides extensive support for GPU parallelism under the *distributed training* API accessible through the module `tf.distribute`. It contains various *strategies* where the code following this thesis has `tf.distribute.MirroredStrategy` implemented. It allows the use of all GPUs on one GPU-node at the SAGA HPC used in this thesis, which amounts to four. When running on GPUs tensorflow uses CPUs for the many smaller operations requiring small amounts of memory, as CPUs are fast in the sense that they are optimized for low latency. Heavier operations such as multiplication of large matrices is handled by the GPUs as it is optimized for large bandwidth. We assessed the use of GPUs in comparison to various CPU-only configurations and found a speed up on GPUs only in very few circumstances of very wide neural networks, thus the use was not merited on larger runs, especially considering the abundance of CPU cores on the SAGA HPC. When considering the time-use of GPUs vs CPUs the relevant factor is the *overhead* when invoking GPUs, which is relatively high. That is, copying data to and from the GPUs is relatively time expensive, thus for GPU use to be advantageous over CPU the operations should be large but relatively infrequent with relatively fast use of memory. In Figure 6.7 we show the result of a short script comparing matrix multiplication run with 1 GPU versus 8 CPU cores on a laptop. The horizontal axis shows the dimensions N in the operation $(N \times N) \cdot (N \times N)$. In neural networks whatever increases the size of the matrices in matrix multiplication will increase the memory use of an operation. We can look at a simple feed-forward expression with identity activation, such as $y = \prod^{L} \mathbf{W}^{(l)} \mathbf{X}^{\intercal}$, to see that layer width and data size increases the size of the matrix and is thus the relevant factors. In our calculations we found relatively small batch-sizes to be best for convergence. Specifically we used 32 data points in a batch for most calculations, totalling 165 batches for the training data. For most calculations we used 20 nodes layer-width, and with 5 features this should give matrix multiplication at the first layer of $\mathbf{W}^{(0)} \mathbf{X}^{\intercal}_{batch} \to (20 \times 5) \cdot (5 \times 32)$. With larger data sets one would have to increase the batch-size of each batch for it to hold the same fraction of the total training data, that is, about $\frac{1}{165}$. The data-set we used in this thesis is small relative to data sets commonly used elsewhere, thus larger data sets could be expected for any future use, and the use of GPUs may therefore be warranted. We ran similar calculations with $(20 \times 5) \cdot (5 \times N)$ matrix multiplication, which indicated an N of at least ~ 3000 before GPU and CPU execution times started to cross over. Using this as a very rough estimate for batch-size and ignoring all other calculations in a BNN (of which there are many) would mean that with 165 batches a data set of circa $\mathbf{X}^{\intercal} \to (5 \times 5 \cdot 10^5)$ would merit GPU use, which the code used in this thesis allows the use of.

The time consuming part of working with variational BNNs is the hyperparameter tuning. On 4-8 CPU cores our model takes about 1-3 hours to run, depending on whether deterministic pretraining (see Section 5.9) has been used or not. However, many hyperparameter evaluations are basically a one-time test. That is, for similar data sets one should be able to confidently rule out a lot of hyperparameters based on previous runs. In order to best exploit the availability of the many CPUs on SAGA we used what is called a *job array*. Which simply means that we ran a large number of models with different hyperparameters consecutively, initiated from the same main script.

Chapter 7

Results and Discussion

In this chapter we present the results of this electroweakino and BNN endeavour. We will combine discussion with the results as we present them and finish off with an overarching discussion about the optimal approach for electroweakino cross section predictions with uncertainties using BNNs. The results can roughly be divided into four parts: 1) We first present discuss the use of the plain variational BNN (See Section 5.5.2) for $pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^+$ production process. 2) Next, we present the validation results from variational BNN with deterministic pretraining (see Section 5.9). 3) Then we present the results from variational BNN with deterministic pretraining and various annealing schemes (See Section 5.6.3). 4) We use the best model from the validation runs and present the results on the test data, and finally we discuss computation time and the standardised residuals of the test results.

For the walk-through of the steps above we will focus on the simplest case, with five features, namely the cross section for $pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^+$, while we will report the final test results of $pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^\pm$ at the end.

7.1 Validation Results

7.1.1 Plain Variational BNN

The plain variational BNN uses the ELBO cost function as its presented in Section 3.3, with a KL-weight π of the complexity cost initially set at $N_{batches}^{-1}$. For batch size $S = 32$ this is $\pi = \frac{1}{165}$.[1] Various hyperparameters were attempted, using both Gaussian and Laplace priors/posteriors, different prior variances, initial posterior variances, nodes, layers, activations, batch sizes and different annealing schemes. While there are indeed differences in performance for the various choices, none of the configurations allowed the training to get past certain local minima where training seems to be consistently stuck. All the results gave an $R^2 < 0$, which means the model is explaining very little of the variance in targets. We did obtain some sort of fitting with simple one variable functions, such as $y = x^2$, but we will not report on these results, as here too the results were abysmal, and the focus of the thesis is not such functions.

The plain variational BNN is simply put, very hard to fit. To the point that we see little value in this method on its own. The main issue, as we see it, is that it seems very

[1] Not to be confused with the irrational number π. We follow the notation of the literature.

hard to strike the right balance between the complexity cost and the likelihood cost (see Section 3.3). On the one hand, optimization of the ELBO cost function prioritizes fitting the prior through the minimization of the complexity cost over the the output variance α and the MSE part of the likelihood cost. In other words, the quality of the predictions suffer greatly. This is essentially an example of underfitting as described in Section 2.5. On the other hand, the KL-weight π can be reduced below $N_{batches}^{-1}$, so that the complexity cost is less prioritized. However, then the opposite problem occurs, that is, the optimization process prioritizes the MSE cost, giving good predictions but at the cost of neglecting the prior distribution.

We did attempt the two annealing schemes discussed in Section 5.6.3. So for example for the linear annealing scheme the idea was to let it first optimize the likelihood cost, and then at a later epoch optimize the complexity cost. However, here too a balance was difficult to strike as it would lose its previous gains on the likelihood cost. That said, even if it did not work for this case, the annealing schemes will come too good use in a moment. We also attempted to use the `DenseFlipout` layer, part of the `DenseVariational` module in `TensorFlow`, while elsewhere, `DenseVariational_v2`: the second version of the module, which does not contain the Flipout method is used. The method description of flipout is found in Chapter C as we will not focus on this method, since we did not manage to successfully use it with pretraining. Unfortunately the above discussion still stands with the use of Flipout, that is, low model performance, $R^2 < 0$. We now move on to the more uplifting results in the next section.

7.1.2 Variational BNN with Deterministic Pretraining

The model we present here is the empirical Bayes model from Section 5.9 for complex architectures. The following describes the setup and results of the validation run. Metrics shown in plots are calculated on the model validation set $\mathbf{y}_{\text{m}}$ (see Section 6.2).

Parameter Description We give here an overview of the hyperparameters in the model, and the ranges that have been tried. Note that in the coming discussion we will only report the parameter configurations that are of most interest to the discussion, as there are simply too many parameters that have been tried and tested.

- **Activation Functions**
  To reduce the number of possible configurations we used the same activation functions for both the DNN and the BNN. For a given activation function, the same one is used throughout the networks, except at the output where $\hat{y}^{\mu}$ uses the identity and $\hat{y}^{\sigma}$ uses the softplus function. The activation functions used are listed in Table 3.2. The LeakyRelu parameter was set to $\alpha = 0.15$, and the ELU parameter to $\alpha = 1$, which are considered in the default range.

- **Architecture**
  As we use mean field posterior distributions the layers and nodes of the DNN used for pretraining is the same as in the BNN. We use the same number of nodes in every hidden layer for all the numerical experiments. The weights from the DNN are used to initialize the BNN as described in Section 5.9. We mainly used architectures with 20 nodes in each hidden layer where we tested depth of $1 - 7$ hidden layers. We did check narrower layers such as 5 and 10, and wider layers such as 50 and 100 nodes, but found that the loss did not converge after running for at least 20.000 epochs. Hence all the reported experiments are from runs with

Parameter	Values
Priors	Gaussian, Laplace
σ_{prior}	$1.3310^{-4},\ 1.4510^{-4}\ 1.5810^{-4},\ 1.3310^{-3},\ 1.4510^{-3},\ 1.5810^{-3}$
δ	$10^{-6},\ 10^{-7}$

Table 7.1: The variable configurations for the models shown in Figure 7.1.

20 nodes.

- **Optimizer, Learning Rate and Batch Size**
  We use the same ADAM optimizer for all experiments and for both the DNN and the BNN. We considered different learning rates η for the DNN and the BNN, where the tested values were $\eta \in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. We found the learning rates $\eta_{\text{NN}} = 10^{-3}$, $\eta_{\text{BNN}} 10^{-4}$ to be optimal for most configurations so these are the learning rates used for all the reported experiments, hence it will not be listed from here on.

  The batch sizes $S \in \{32, 64, 128, 256\}$ were attempted, and the batch size 32 was seen as the most favourable in terms of convergence, so this is the batch size used in all the reported experiments, hence it will not be listed from here on either.

- **Epochs, Patience and Sample Size**
  For both the DNN and BNN the number of epochs was set very high at 40000, a number that is not expected to be reached. The patience was set at $3000 - 4000$, which for most runs is more than enough and the training will often run for much longer than what is required for a good result. This was done to be certain that the minimum had been reached, as for some few configuration a new minimum is reached late in the run. For all validation runs the sample size used to calculate metrics was 2000.

- **Prior and Posterior Mean Field Distributions**
  The Gaussian and Laplace distribution were tried as both the posterior mean field distribution (PMF) and the prior distribution, so giving four combinations. The prior and PMF means μ were initialized by the DNN. The prior standard deviations (scale parameter for Laplace) that were tried were in the range $\sigma \in [10^{-4}, 1]$, and the $\delta \in [10^{-7}, 10^{-3}]$.

- **KL-weight π**
  The KL-weight is for all the runs presented in this section are kept at $\pi = \frac{1}{165}$, as we found higher or lower KL-weights to tip the scales too much towards complexity cost or likelihood cost.

Performance: Activation Functions We first assess the performance of the different activation functions, where we in figure Figure 7.1 illustrate the ELBO loss (scaled), the RMSE (unscaled) and the CAOS (unscaled) on the validation set as a function of the choice of activation function. The architecture was $[5, 20, 20, 20, 20, 20, 2]$ for all models and the varying parameters and distributions, except the activation functions, is listed in Table 7.1. Furthermore a Gaussian PMF was used for all cases, and a patience of 4000.

Looking at Figure 7.1 we see that the swish activation function has the lowest ELBO validation loss, and it perfors well on the RMSE values as well. Although on the RMSE

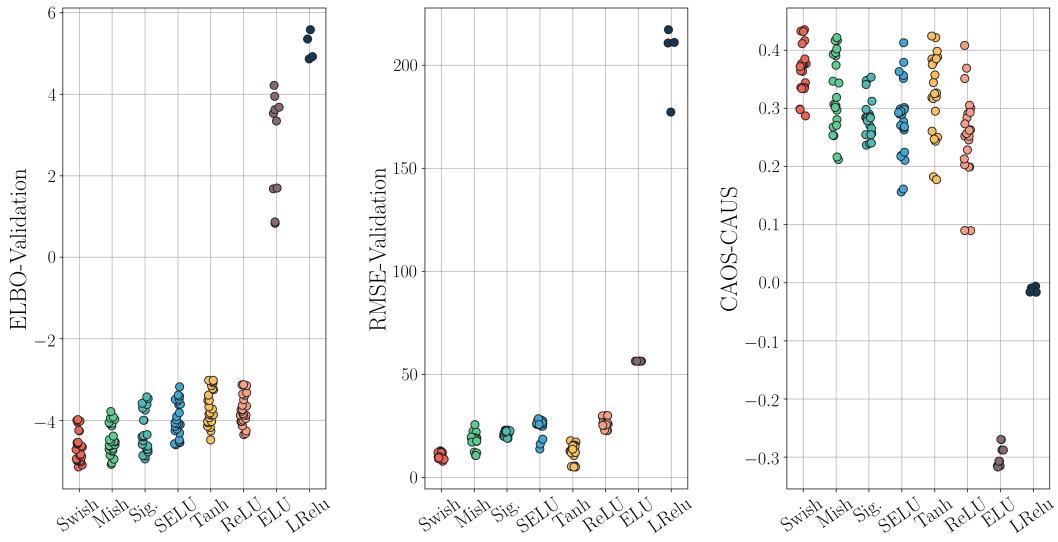


Figure 7.1: Comparison of the activation functions listed in Table 3.2 evaluated on the model validation set. Parameter variation are listed in Table 7.1.[2] For LeakyRelu(LRelu) and ELU, $\alpha = 0.15$, $\alpha = 1$, respectively. The plot has been cut at ELBO $= 6$: LRelu has ELBO values up to ELBO ~ 5600, and ELU up to ELBO ~ 70, while the other activation functions have all their values shown. Thus LRelu and ELU are "worse" then shown here. The third panel vertical axis shows the subtraction of CAUS from the CAOS.

we see that the tanh activation function also has a good fit. It is interesting to see that arguably the most popular activation function in machine learning today, the ReLU, performs worse than the sigmoid and tanh which were the hidden layer activations of choice over a decade ago.[3] The swish and mish, which are derivatives of the sigmoid and tanh, respectively, seems to be the optimal choice for these variational BNNs. We note that in the third panel we subtract CAUS from CAUS, but most points have either CAUS or CAOS equal to zero, that is, they are either liberal or conservative. Those who have non-zero values of both have one that is very small. We see that the better the activation performs with ELBO and RMSE the more conservative the model becomes, to the point that it possibly is too conservative. We shall discuss this further in the coming section. Nonetheless, we choose the swish activation for all ensuing experiments, or that is, it is the one that remains by the filtering procedures we outline in the next paragraphs.

Selected Models In Table 7.2a we show the parameters of the four best performing models in ELBO, R^2 and RMSE, obtained from multiple runs with various configurations and one "bad" performing model (Model 1) that we will soon elaborate on. The hyperparameter ranges are similar to what was seen in Section 7.1.2, but these are indeed the best performers of wider hyperparameter searches (see ranges in Section 7.1.2). The performance and uncertainty metrics of the models are listed in Table 7.2b and Table 7.2c, respectively.

The well performing models that have been selected have been filtered by the requirement that $R^2 > 0.94$ for both train, ES validation and model validation. For de-

[3] The ReLU and its derivatives came about in 2010 [72].

	Layers	Prior	σ_{prior}	δ
1	5-20-20-20-20-20-20-2	Gaussian	1.4510^{-3}	10^{-6}
2	5-20-20-20-20-20-2	Laplace	1.3310^{-4}	10^{-6}
3	5-20-20-20-20-20-2	Gaussian	1.4510^{-4}	10^{-7}
4	5-20-20-20-20-20-2	Gaussian	1.5810^{-4}	10^{-6}
5	5-20-20-20-20-20-2	Laplace	1.3310^{-4}	10^{-7}

(a) The parameters that differentiates the models.

	ELBO model	RMSE train	RMSE es	RMSE model	R2 train	R2 es	R2 model	πKL
1	-5.144	38133	44.276	5.639	0.000	0.748	0.989	0.165
2	-3.987	12.658	26.784	9.544	0.986	0.952	0.975	1.559
3	-4.838	12.972	26.540	9.244	0.985	0.951	0.976	0.719
4	-4.993	12.964	26.524	9.158	0.985	0.951	0.976	0.599
5	-4.007	13.033	26.749	8.860	0.984	0.949	0.978	1.603

(b) The performance metrics. The ELBO and πKL (the complexity cost) are values dependent on the weights of the network which are calculated in the log-scaled frame, while the remaining have been calculated in fb scale. We may note that the decimals of the largest number in this table have been truncated.

	$\sigma_{z,m}$	$\mu_{z,m}$	CAOS
1	1.5178	0.1357	0.3618
2	0.7018	-0.0886	0.2868
3	0.7029	0.0602	0.3337
4	0.7021	0.0648	0.3356
5	0.6865	-0.0642	0.3718

(c) The uncertainty related performance metrics. $\sigma_{z,m}$ and $\mu_{z,m}$ are standard deviation and mean of the standardised residuals using geometric mixture standard deviation discussed in Section 6.3.1. The standardised residual estimates for epistemic and aleoteric standard deviations have been omitted, and the CAUS has been omitted as all the models had CAUS = 0.

Table 7.2: Parameters and metrics for the four best performing models and one bad performing model (Model 1) in variational BNN with pretraining.

terministic machine learning, a model trained to where overfitting begins would have $R^2_{train} > R^2_{es} > R^2_{model}$. This is not at all the case for the models encountered through these experiments, meaning that while it may happen, we do not manage to consistently overfit to the training data, that is, reach the start of overfitting. So here we see the Bayesian experience in practice. A second requirement is that the standard residuals do not over or under predict showing systematic bias, so that $\mu_z \approx 0.0$, specifically $|\mu_{z,m}| < 0.1$

We have included the bad performing model in Table 7.2b to illustrate the issue of picking a model in the manner typical in deterministic machine learning. We see that it is the best performing model on the model validation set, but on the other two data sets it does not only perform worse, but disastrous had we wanted to use the model for anything practical. In our experience, Model 1 is an outlier in terms of performance, however, but still a good share of the models will have metric performance on the trainig set markedly lower than on validation sets (even without early stopping), such as the models in row 2-4 Table 7.2b that indeed *does* reach the limit at which overfitting begins, in that the RMSE and R2 of the training are the best performers. Validation and test sets are important, however, the results do shed light on the different nature of a Bayesian method, even an approximate one, owing to its natural regularizing effect as explained in Section 4.3.

Furthermore, the results also show that the ELBO is not the end all and be all of performance measures, as we see that the models in row 2 and 4 have comparable performance as models in row 3 and 5, but the former have a higher complexity cost which affects the ELBO. That they have a higher complexity cost should probably be expected as their surrogate posteriors are Gaussian, while the prior is Laplace. When both the posterior and prior is of the same distribution family the surrogate posterior can more easily fit to the prior.

Which one of the four best performing models we should choose is a tough decision, as their performance is very similar, and their differences might be a result of the stochastic nature of the procedure. However, one uncertainty metric sticks out, namely the CAOS in the second row of Table 7.2c. All the models are notably conservative, which is not necessarily bad as it implies that we have interval uncertainties larger than necessary, however, we should not be too conservative either as then the value of uncertainty estimates are reduced, and at the end of the day we do want to be certain about something. So the model corresponding to the second row with CAOS = 0.2868 would be our model of choice. That is, if we did not have better models in the next section. There we will also take a deeper look into the other uncertainty metric, loss curves, as well.

7.1.3 Variational BNN with Deterministic Pretraining and Annealing

In this section we pick up the thread from the previous section. The parameter descriptions and ranges from Section 7.1.2 still apply, and we keep the same patience, total epochs, learning rates and batch size. We do restrict some possibilities, for example now we only use the swish activation function, the architecture $[5, 20, 20, 20, 20, 20, 2]$ and $\delta = 10^{-6}$. As the configurations just mentioned are kept constant we not list any of these in the following results. We do, however, vary the prior, both in terms of its variance $\sigma_{\text{prior}} \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and whether it is Gaussian or Laplace, so this information is included. In the following models there is one exception to the para-

meter settings, and that is regarding the KL-weight π, which we now shall vary with exponential batch annealing (EBA) and linear epoch annealing (LEA) as explained in Section 5.6.3.

Parameter Description We first discuss the parameters of the two annealing schemes.

- **Exponential batch annealing**
  Using EBA involves the introduction of the hyperparameter κ. The values that were attempted were the κ-values depicted in Figure 5.3, that are in the range $\kappa \in (0.071, 1.33)$, additionally we also tried values $\kappa \in 0.011, 0.021$. Note that these small differences matter for the final KL-weight that is obtained, $\pi_{\text{final}} = e^{-0.011 \cdot 165} \approx 0.16$, and $\pi_{\text{final}} = e^{-0.071 \cdot 165} \approx 8 \times 10^{-6}$.

- **Linear epoch annealing**
  Using LEA involves the introduction of four hyperparameters: minimum KL-weight π_a, maximum KL-weight π_b, epoch of annealing start τ_0, and the period of annealing τ_Δ in epochs. The values that were tried were $\pi_a = \{10^{-5},\ 10^{-4}\}$, $\pi_b = \{\frac{1}{165},\ 10^{-2},\ 10^{-1}, 1\}$, $\tau_0 = \{20,\ 200,\ 1500,\ 2000,\ 4000\}$ and $\tau_\Delta = \{2000, 4000\}$.

  When reporting the linear annealing results we will report the *ELBO-metric*, which we define as the ELBO used on the Model-validation set, but always with $\pi = {N_{batches}}^{-1}$. This is so that it is easier to compare the ELBO across experiments.

For the EBA case we found that the annealing schemes with small κ gave the best results, that is around $\approx 10^{-2}$, which is considerably smaller than the $\kappa = 0.693$ used in [42]. For the LEA case we saw no difference between starting at $\tau_0 = 20$ and using $\tau_\Delta = 2000$ and $\tau_\Delta = 4000$. We did however see a difference between starting the annealing early $20 - 200$ and late $1500 - 4000$, where starting it early more consistently gave good results in terms of a lower ELBO-metric and RMSE, and higher R2 score.

For both EBA and LEA we found that prior variances at $\sigma_{\text{prior}} = \{10^{-4},\ 10^{-3}\}$ were optimal, which mirrors the results of the non-annealing case. The Gaussian priors gave overall a slightly better ELBO-metric, RMSE and R2 scores, however it had a larger tendency to systematically over-predict or under-predict, specifically $|z_\mu| \approx (0.1, 0.2)$[4], while the runs with Laplace priors consistently had $z_\mu \approx 0.0$.

In Table 7.3 we list four of the best performing models of both EBA and LEA, which have been filtered by ELBO-metric < 4.9 and $R2 > 0.95$ (for all three data sets). We have not filtered by $z_\mu \approx 0.0$, as unlike in Section 7.1.2 we want to show a model that performs well on all other metrics.

The LEA and EBA results in the Table 7.3 have similar performance metrics as compared to the non-annealing case in Table 7.2b in terms of the RMSE and R2. However, there is one thing that stands out, the complexity cost πKL, which for all the models in Table 7.3 are smaller, compared to Table 7.2b (excluding the badly performing model). This implies that for both schemes, even though they are very different, that the model fits easier to the prior without it being at the expense of the likelihood cost and thus the RMSE and R2 metrics. There is no clear pattern for the uncertainty metrics $\sigma_{z,m}$ and CAOS between the annealing and non-annealing case. However, as the annealing

[4]Meaning approximately in this range.

	Annealing	π_a	π_b	τ_0	τ_Δ	Prior	σ_{prior}
1	LEA	10^{-5}	$\frac{1}{165}$	200	2000	Gaussian	0.001
2	LEA	10^{-5}	$\frac{1}{165}$	200	2000	Laplace	0.001

(a) Hyperparameters for the LEA scheme.

	Annealing	κ	Prior	σ_{prior}
3	EBA	0.021	Gaussian	0.00015
4	EBA	0.071	Gaussian	0.0001

(b) Hyperparameters for the EBA scheme.

	ELBO model	RMSE train	RMSE es	RMSE model	R2 train	R2 es	R2 model	πKL
1	-5.2735	12.8993	26.8234	9.7495	0.9856	0.9522	0.9744	0.1754
2	-4.9517	12.7960	27.3757	9.8255	0.9861	0.9513	0.9741	0.4873
3	-5.5136	13.0663	26.6529	9.673	0.985	0.9517	0.9746	0.0458
4	-5.4797	12.9716	26.6202	9.2123	0.9848	0.9505	0.9764	0.1226

(c) Performance metrics.

	$\sigma_{z,m}$	$\mu_{z,m}$	CAOS
1	0.6654	-0.1218	0.3514
2	0.6465	-0.0346	0.4058
3	0.707	0.0384	0.3211
4	0.6941	-0.0336	0.3716

(d) Uncertainty metrics.

Table 7.3: Varied parameters and metrics for the five best performing models of variational BNN with pretraining and exponential batch annealing or linear epoch annealing. The standardised residual estimates for epistemic and aleoteric standard deviations have been omitted, and the CAUS has been omitted as all the models had CAUS = 0.

schemes are better at minimizing the KL-loss we suspect that this should translate to better uncertainties, given that the priors are sensible.

Now, let us assess which model we should pick. Model 1 over-predicts markedly more than the other models, as can be seen from $\mu_{z,m}$. If the z distribution is approximately normal the deviation from zero for Model 1 is in the same order of magnitude as the amount of variation about the origin, that is, the 1 standard deviation σ_z is in the same order of magnitude as $\mu_{z,n}$, which is too large systematic deviation too ignore, thus we do not choose Model 1. Model 2, with Laplace prior, has a relatively high KL-loss and is the most conservative of the four models. We omit this model from consideration owing to its worse complexity cost. Model 3 and 4 are similar, but Model 3 obtains the overall best ELBO and complexity cost, so we choose Model 3. However, we note that these two models are probably equivalent as multiple runs have been executed on model 4 with different seeds (random number generator) and it obtained ELBO values in the range $(-5.47, -5.51)$ By the metrics evaluated until now, we conclude that the EBA method with small κ is the superior method for this data set.

Performance Illustrations of the Optimal Model In Figure 7.2 we have plotted the loss curve obtained from the training of Model 4. It includes both the pretraining phase and the BNN phase. The NN cost function of the pretraining phase is the same as the likelihood cost function in the BNN. The ELBO curves shown are of the ELBO-metric with $\pi = \frac{1}{165}$, not the ELBO-loss. The ELBO-loss and the complexity loss has the same form as the ELBO-metric and the complexity loss of the ELBO-metric, but would be shifted up if depicted. The plot has been truncated at a vertical axis value of one. The ELBO-metric curves begin at ≈ 5000. The curves as shown here have been smoothed out by a *Savitzky–Golay filter* [73] for illustration, as in reality the curves are significantly more noisy. The kink that can be seen as the training procedure goes from NN to BNN is continuous within the first epoch of the BNN, that is, over the iterations/batches. So there is an immediate improvement on the likelihood estimate by the introduction of a Bayesian model, owing to its regularizing effect. However, as noted previously, there is a trade-off at play between the likelihood loss and the complexity loss in a variational BNN, which can be seen by the likelihood's subsequent increase.

In Figure 7.3 we show the true values from `Prospino`, that make up the model validation targets $\mathbf{y}_\mathrm{m}$, plotted against the predictions. Additionally we plot the "fast" uncertainty estimates σ_m as error bands/bars on every model validation prediction. Since the true values cover 10 orders of magnitude the inverse transformed output values of our program are unfeasible for illustration. We have thus taken the base 10 logarithm of the true and predicted values, similarly we have used the uncertainties without the inverse transformation in Equation (6.13). We convert the uncertainties from the natural logarithm to the base 10 logarithm as well.

In Figure 7.3 we remove the most uncertain predictions in order uncertainty magnitude. We do this in order to see if the removed points correspond to the largest residuals. If it does it implies that our model correctly yields uncertainties proportional to the residuals. The left panel of figure Figure 7.3 shows the true versus predicted values with one highly uncertain point removed here, this has been done for illustration as the error is simply too large to plot, but we note it is also the point with the largest residual. The right panel shows the top 20 most uncertain points removed. In Figure 7.3 we see in the legend how the removal of highly uncertain predictions improves performance seen by

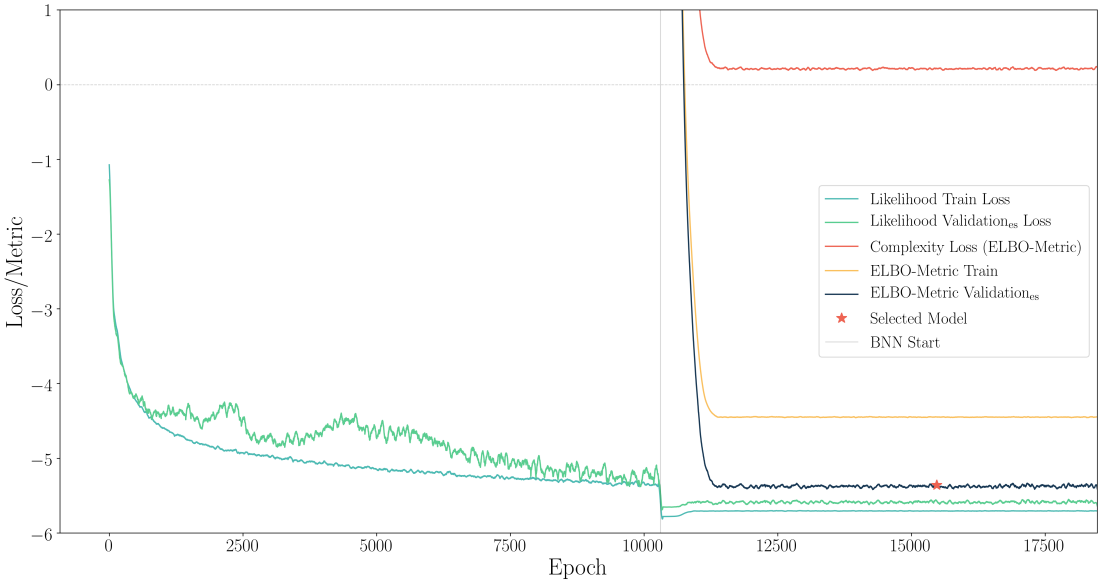


Figure 7.2: Loss plot of Model 4 in Table 7.3 with EBA scheme. First part of the plot (from the left) is the empirical pretraining, while the second part is the BNN. The actual ELBO-loss curves (variable KL-weight) used for training are not shown here, but they are similar to the ELBO-metric curves, but shifted up.

the increased R^2 score and decreases the RMSE, implying that the residuals no longer in the plot were on average worse than the rest. We see clearly from the figures that the highest uncertainties are indeed the largest residuals. This illustrates the power of the uncertainties, as these are accessible regardless of whether we know the out-of-sample true values.

For all the predictions/validation points we have coloured them according to whether the neutralinos are predominantly bino, wino or higgsino. The focus is however on the error bands and bars, so this is not too visible. The color-labeling is obtained by using Equation (1.55). The largest value among N_{11}^2, N_{21}^2 and $(N_{31}+N_{41})^2$ decides whether the point is labeled a bino, wino or higgsino, respectively. The way in which we assessed which points had the highest uncertainty was to use a variation of what is called the *coefficient of determination*, which we defined as

$$\mathrm{CV}_i = \frac{\sigma_{m,i}}{\mathrm{med}_{\hat{y}}}. \tag{7.1}$$

So for a given cross section prediction we standardise $\sigma_{m,i}$ by the median of all log-scaled predictions. The idea behind this is to measure the uncertainties independent of the true values, but still comparable across models. Note that we do not scale $\sigma_{m,i}$ by its own prediction, instead we rely on the assumption that the target distribution has been made sufficiently symmetric in the scaled frame, such that its central tendency can work as a partial standardisation quantity for all data points. Strictly speaking, our measure is *not* a coefficient of determination as these are usually properly standardised so that they reside in the range 0% to 100%. This is mostly the case for this CV, however in the left panel of figure 7.3 the one data point that was removed had CV ≈ 16.39, hence we shall not express it in percentages. In the right panel of Figure 7.3 the highest CV value of the removed points were CV ≈ 1.8 and the lowest CV ≈ 0.07, corresponding to absolute differences and uncertainties of $\hat{\epsilon} \approx 5 \times 10^{-2}$, $\sigma_m \approx 18.8$ and $\hat{\epsilon} \approx 10^{-5}$, $\sigma_m \approx 0.08$, respectively. The points were removed by the order of their CV.

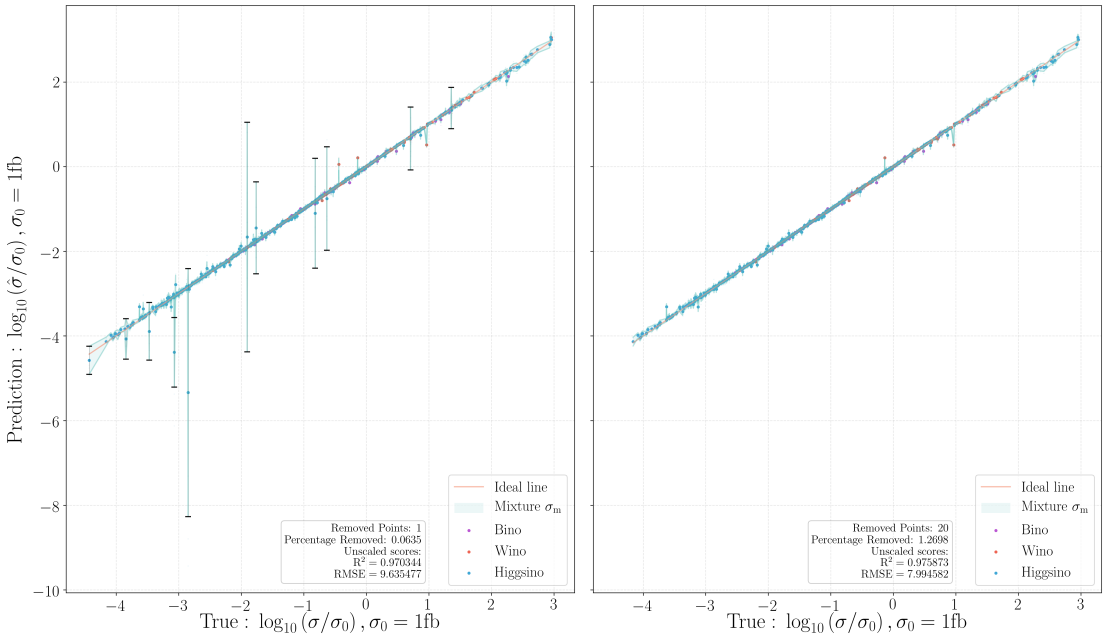


Figure 7.3: For both left and right panel, the figure shows predicted $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ cross section values on the vertical axis and the corresponding model validation cross section values from `Prospino` at the horizontal axis. In the right panel the point corresponding to the 20 highest uncertainties have been removed. Error bars and error bands are calculated by using the scaled symmetrical mixture standard deviation σ_m. Error bars are only printed for points with error bands larger than 0.5, to keep the figure uncluttered. The predictions have been coloured according to if the neutralinos are predominantly bino, wino or higgsino.

Next we look at the ETI based metrics, the CAOS and CAUS, in the right panel of Figure 7.4 together with three standard deviations of the mixture σ_m based error bands, only the latter is log-scaled. We now compare Figure 7.4 with the two plots in Figure 7.5. The only difference between the two figures is that in Figure 7.4 the coverage curve in the right panel has been calculated by sampling multiple $\hat{y}$ (see Equation (5.81)) from the network likelihoods, while in Figure 7.5 we have sampled the means of the likelihoods only $\hat{y}^\mu$ (See Equation (5.82)) and we use the epistemic uncertainty σ_e, instead of the mixture uncertainty. The plots in Figure 7.4 thus capture the spread of each likelihood, therefore accounting for the aleoteric uncertainty, while Figure 7.5 only has epistemic uncertainty. By comparing the two figures we see that without the aleoteric component the coverage curve is very liberal. We see in the right panels that when it is liberal it fails to produce relatively large error bands on the predictions that miss the true value in the left panels. To obtain reliable uncertainties it thus seem crucial to include the spread of the likelihood. This is surprising to us, as the noise from `Prospino` cross section data is negligible [5]. Therefore we suspect that the measures that includes a variance, α^2, of the network output likelihood, such as the σ_m or the ETI using $\hat{y}$, are not reliable measures of the irreducible noise of the data. Instead we believe they capture a good portion of the epistemic uncertainty, and in our case in a favourable way. What we have just described was also seen when using epistemic uncertainty σ_e only and comparing it with using the mixture uncertainty σ_m. That is, the epistemic uncertainty seems to under-estimate the uncertainties.

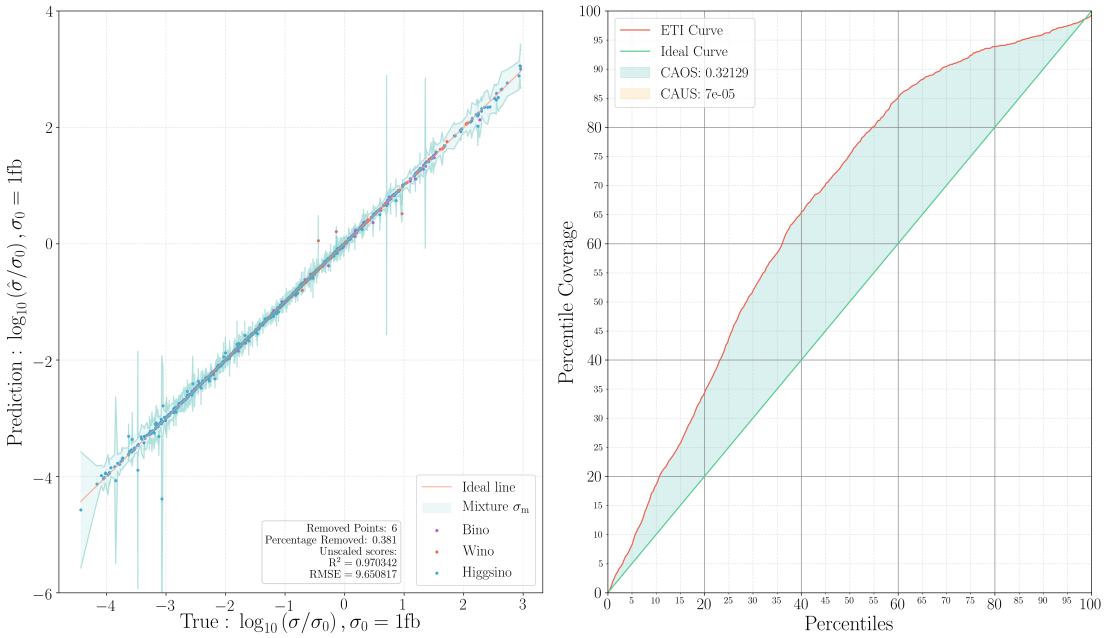


Figure 7.4: Left-hand panel shows the $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ cross section predictions against the $\mathbf{y}_m$ validation set, with mixture error bands based on σ_m. The right-hand panel shows the curve of multiple ETIs for multiple percentiles. Note, that the coverage curve is calculated in the fb scale, while the prediction versus true plot is a log-log plot. Predictions are made by likelihood samples ($\hat{y}$).

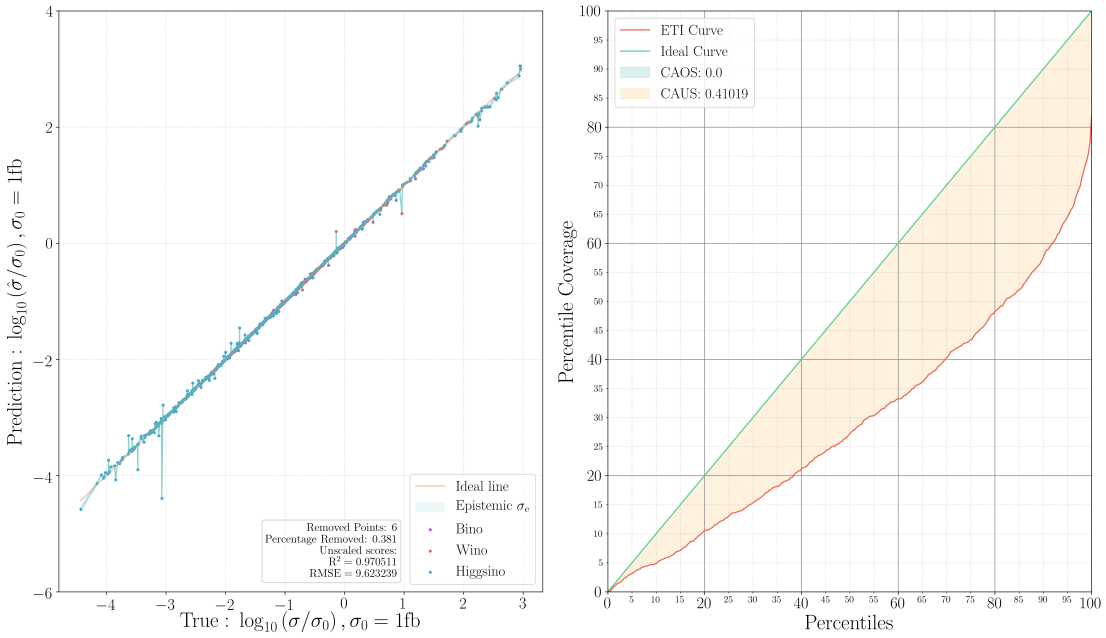


Figure 7.5: Left-hand panel shows the $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ cross section predictions against the $\mathbf{y}_m$ validation set, with epistemic error bands based on σ_e. The right-hand panel shows the the curve of multiple ETIs for the percentiles given. Note, that the coverage curve is calculated in the fb scale, while the prediction versus true plot is a log-log plot. Predictions are made by likelihood means ($\hat{y}^\mu$).

7.2 Test Results

In this section we will continue from the previous section and use the best model from the validation procedure in the $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ case to check how it performs on out-of-

ELBO test	RMSE train	RMSE test	R2 train	R2 test	πKL
-4.8928	1839317	47.1305	-0.0001	0.7232	0.0723

Table 7.4: Performance metrics for prediction on the test set with Model 4 from Table 7.3c. Their performance scores are discussed in the text.

sample data. Similarly we show the test results for the $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$, where we have performed a similar validation analysis as in Section 7.1.

Neutralino-Neutralino Cross Section Results In Table 7.4 we see the results on the training set and the test set for $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$. We do not list the parameters as these are the same as with Model 4 in Table 7.3b, with the exception that we changed the size of κ to adjust for more mini-batches. Specifically, $\kappa = \frac{0.021 \cdot 165}{214}$, so that the final KL-weight within an epoch is the same as in the validation model. In Table 7.4 we see that performance on the test set has taken a considerable step down from the performance on the validation set. This illustrates the importance of doing out of sample testing.

In Section 7.1 we made the point that reliable uncertainties is of great importance, and if removing the largest uncertainties leads to an improved model (in terms of R^2 and RMSE) then we can infer that these large uncertainties more or less correspond to the residuals that are on average larger than the bulk of residuals. However, one may argue that we violate the central principle of out-of-sample testing, as described in Chapter 2, namely that one should not modify the model after the fact. However, this is purely an exercise in assessing the quality of the uncertainties, and they would indeed all be used in supersymmetry global search and filtered accordingly as cross section theory and experiment is compared.

In Figure 7.3 we removed 20 data points giving us a CV = 0.07 for the last prediction that was removed, which reduced the RMSE from RMSE $\approx$ 9.64 to RMSE $\approx$ 8. We opt for the same here, with CV $>$ 0.07 for the test set. In Figure 7.6 we show prediction against test values for $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ in the same fashion as we did in Figure 7.3 with σ_m, but removing predictions with CV $>$ 0.07, specifically 113 prediction/test points, which constituted $\approx$ 5.4% of the test data and gave $R^2 = 0.9976$ and RMSE = 0.6864, which is a considerable improvement, implying that the uncertainties do on average indeed correspond to predictions that are far removed from the true value as the R^2 and RMSE are measures of exactly the average residuals. We emphasize that predictions would not be removed in a supersymmetry global search, they would be compared with experiment and weighted accordingly. We assess only the effect the largest uncertainties have on the metrics.

The initial performance was even worse on the training set as seen in Table 7.4. Using the same procedure with CV $>$ 0.07 on the training set, we improved the score on the train set considerably as well. This involved the removal of the 178 prediction/test points. The test and train results are summarized in Table 7.5, where we have included the average relative test error (ARTE) from Equation (6.22). The highest uncertainty of the predictions were at CV $\approx 10^{63}$, which was far higher than other uncertainties, however this was also the point with the highest residual with four orders of magnitude difference between true and predicted values.

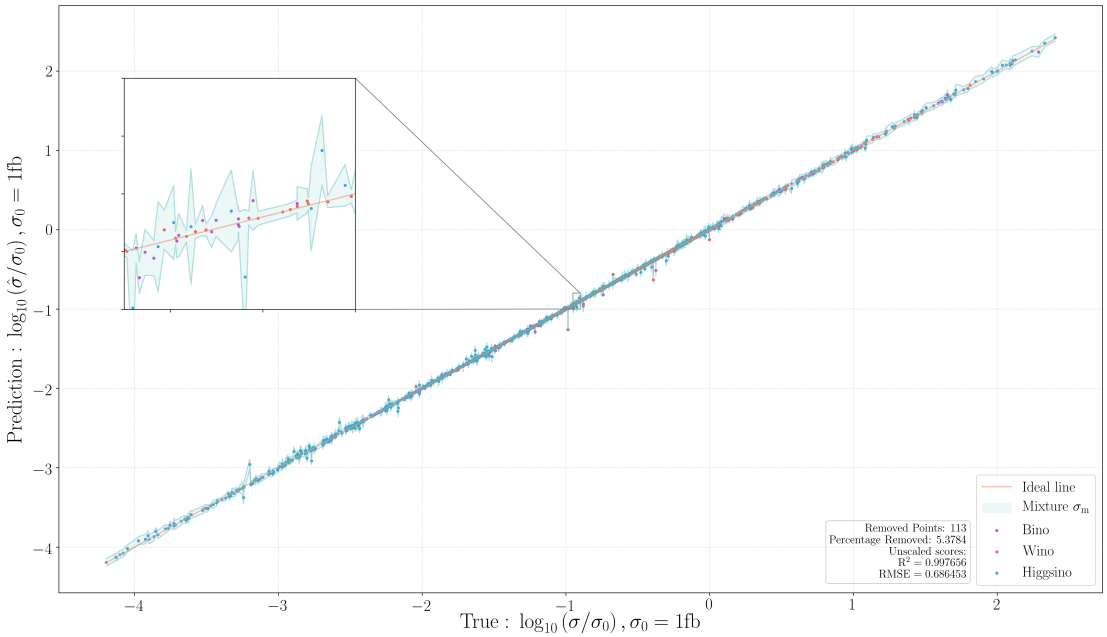


Figure 7.6: Figure shows predicted $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ cross section values on the vertical axis and the corresponding test set cross section values from `Prospino` at the horizontal axis. Error bars are calculated by using the scaled symmetrical mixture standard deviation σ_m. The predictions have been coloured according to if the neutralinos are predominantly bino, wino or higgsino.

RMSE train	RMSE test	R2 train	R2 test	ARTE test	% Removed train	% Removed test
2.93278	0.68645	0.98994	0.99765	3.33%	2.6073%	5.3784%

Table 7.5: Results test and train set predictions for $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$. The performance metrics from test run with model parameters corresponding to model number 4 from Table 7.3c, but now highly uncertain points have been left out according to their CV. Listed is also the amount of data that has been removed and the average relative test error (ARTE).

Neutralino-Chargino Cross Section Results We followed the same validation procedure for the cross section $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$ as in Section 7.1. However, we did a smaller hyperparameter search, assuming that some of what applied for $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ applied here as well. This means we used the same network architecture, batch size, learning rates, activation function, and annealing scheme (EBA). We summarize the test and train results of the final model in Table 7.6. Unlike for the $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ case, here the model chosen through validation gave very good results on the test set as well. We filtered CV > 0.07 again for the purpose of showing that the highest uncertainties correspond to the points with highest residuals, removing 243 points, constituting $\sim$ 11% of the predictions/test points. This gave $R^2 = 0.9982$, RMSE $= 125.7445$, which is a clear improvement from the results in Table 7.6, implying again that the uncertainties correspond to high residuals. Before pruning the predicitons the ARTE was close to 3000% while after ARTE $= 4.50\%$.

We turn again to error band plots. In Figure 7.7 we plot $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$ against $m_{\widetilde{\chi}_2^0}$ with σ_m at two standard deviations. For the purpose of illustration we plot points where $N_{21} < 0.75$ and $V_{11} < 0.75$, thus removing neutralinos dominated by the bino $\tilde{B}^0$,

ELBO test	RMSE train	RMSE test	R2 train	R2 test	πKL	$\sigma_{z,m}$	$\mu_{z,m}$	CAOS
-4.8928	390.4933	293.2014	0.9874	0.9936	0.1408	0.7633	-0.0399	0.4591

(a) Performance metrics

κ	Prior	σ_{prior}
0.072	Laplace	0.00015

(b) Hyperparameters

Table 7.6: Parameters, train- and test metrics for the top performing model for $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$. The standardised residual estimates for epistemic and aleoteric standard deviations have been omitted, and the CAUS has been omitted as all the models had CAUS = 0.

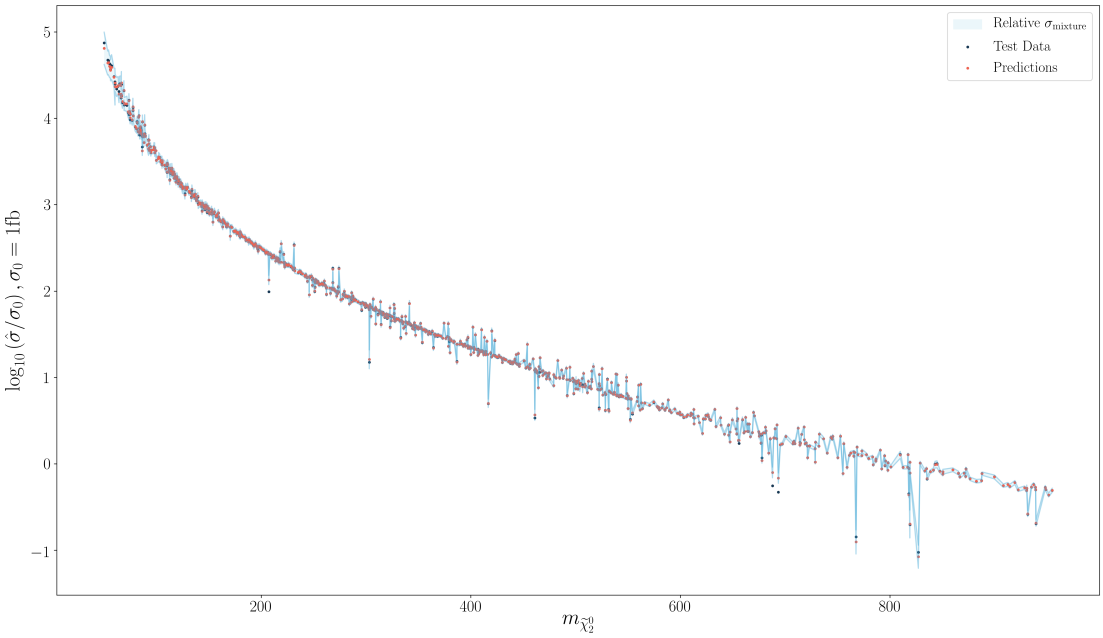


Figure 7.7: Figure shows cross section predictions and true values for $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$ against $m_{\widetilde{\chi}_2^0}$, with the 2 standard deviations of σ_m. Filtered by $N_{21} < 0.75$ and $V_{11} < 0.75$. The horizontal axis is in units of [GeV].

and charginos dominated by charged wino $\widetilde{W}^+$. We do not trim any points according to uncertainties. We see a few number of data points falling outside the error band curve, however the uncertainties are still large at these points. To assess whether the uncertainties are indeed conservative we will in the next section look at the standardised residuals as a final test.

Standardised Residuals We show here the standardised residual plots (see Section 6.4) on the test set of the two cross sections analyzed. The standardised residuals z have not been filtered, that is, no predictions have been removed. In Figure 7.8a we have plotted z for $pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0$ using the geometric mixture standard deviations $\Delta_{GMV}^{\pm}$ for the mixture, aleoteric and epistemic cases. Similarly in Figure 7.8b we have done the same for $pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+$. Unlike previous plots (except CAOS/CAUS), the uncertainties here have been transformed back to the original cross section scale. This means

Average NN Time	Average BNN Time	Average Prediction Time	Average Epochs
0:48:18	0:36:19	0:00:07	6200

Table 7.7: Average computation times for 10 models with architecture $(5, 20, 20, 20, 20, 20, 2)$. Prediction time is total time for 2000 samples for all 1575 data points, so in total 3.15×10^6 samples. Average epochs is the average number of epochs until the BNN stopped. Average epochs for the NN is not listed (usually larger). The format of the time is hours:minutes:seconds.

they are asymmetric, and they have been plotted according to whether the residual is above or below zero according to the discussion in Section 6.4. We have included a standard normal distribution, that is $\mu = 0$ and $\sigma = 1$, for comparison. It is clear from both figures, that all three measures can be considered conservative, however, it is both the aleoteric and epistemic combined that gives the most conservative measure in the residual based on the mixture uncertainty. Additionally, none of them show signs of systematically over- or under-predicting as they are centered at zero.

7.2.1 Computation Time

We shall now make some comments on run time for both model training and prediction.

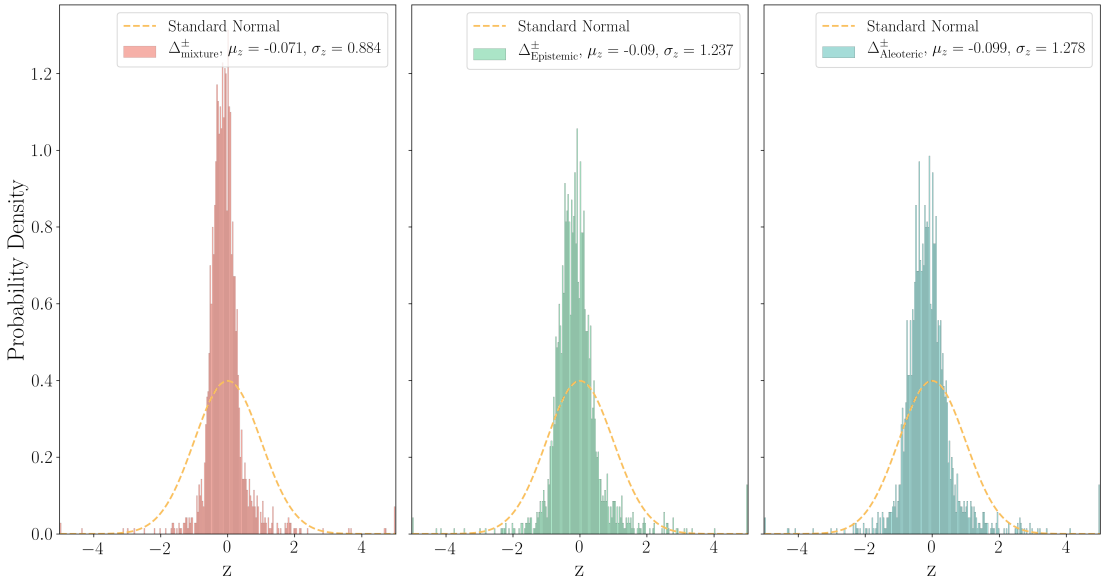
Computation Times For Training In Table 7.7 we show average computation times of one EBA hyperparameter run that involved training 10 models, seeking to predict $\sigma(pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0)$. The architecture matters a lot in terms of computation time when training, in this case it was the same for all 10 models at $(5, 20, 20, 20, 20, 20, 2)$. In Table 7.7 *prediction time* is for one execution of (2000 samples) $\times$ (1575 data points), but this operation is done for all three data sets $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{es}}$, $\mathcal{D}_{\text{m}}$. Note, this setup is inefficient, as all ten BNN models could have used the same initial NN weights, but nonetheless it lets us get an idea of the average computation time. Both the NN and the BNN had the same patience and maximum epochs setting, we see, however, in Table 7.7 that the NN has on average used longer time until early stopping than the BNN, even though the BNN is a more complex structure. The BNN is, in fact, markedly slower than the NN, but for the pretraining case the NN has already done a lot of the work for the BNN, which is why the BNN does not need to run as long as the NN. The average epochs listed is average epochs for the BNN, not the NN, but as implied by the previous discussion, the NN runs for longer and usually needs $10.000 - 20.000$ epochs until early stopping.

Hyperparameter tuning for $\sigma(pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0)$ took a lot of time, and thousands of models have been attempted. However, we found that after we had an idea of what hyperparameters worked for the first case this was highly transferable to the $\sigma(pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+)$ case, and thus we had to run far fewer hyperparameter runs until we found a working model. While more testing is needed with other data sets, this is potentially means that for practical use the amount of time needed for tuning will not become excessive enough to reduce the time efficiency advantages using a BNN has over using only `Prospino 2.1`

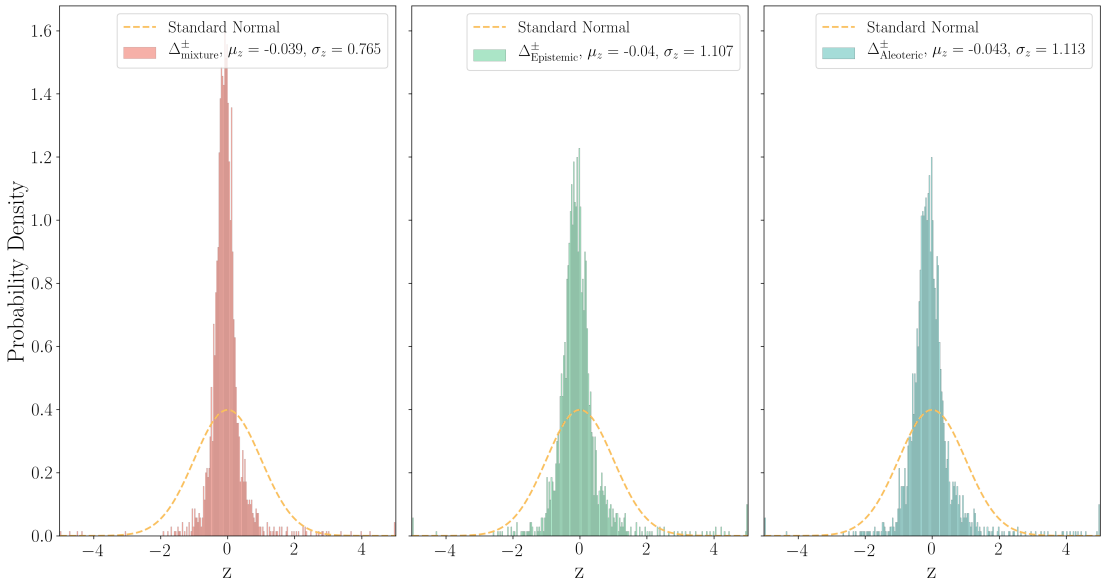
Computation Times For Prediction Finally we show in Figure 7.9 a plot of RMSE and ARTE against various sample sizes. The interval ranges from (10, 5000) in sample size, where for each point we increase the sample size by 10. The model used for

this evaluation is the test set model of the $pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^+$ cross section. The test set and predictions have been filtered by $N_{21} < 0.75$ and $V_{11} < 0.75$ and CV > 0.07, respectively. The ARTE in Figure 7.9 is about 6.5%, while in Section 7.2 we noted it was 4.5%. This is because at that point we had not filtered the mixing matrix elements. The number of data points predicted in Figure 7.9 is 1045, meaning that for the maximum sample size of 5000 the program did approximately 5 million samples in four seconds. If we consider the left panel in Figure 7.9 with the RMSE we see that the spread of scores converge for larger sample sizes. However, from 2000 to 5000 there is a relatively small change in spread, so using a sample size of 2000 might be sufficient. By this plot, we can roughly state that ~ 1000 predictions at 2000 samples each takes about 2 seconds, and assuming that it scales linearly we could produce a million SUSY predictions in ~ 30 minutes, which is considerably faster than `Prospino 2.1`. Note, that every sample here involves one sample $\hat{y}$ from a likelihood, then a new sample from not the same likelihood, but a new likelihood, and so on. The calculations involve also sampling and calculating σ_m. Had we sampled $\hat{y}^\mu$ instead of $\hat{y}$ we could reduce the number of samples to get the same RMSE as in Figure 7.9, but then we would lose the aleoteric uncertainty in the prediction samples (as shown in Figure 7.5). However, the aleoteric uncertainty is still contained in σ_m, so if we are content with σ_m the sampling efficiency can be increased further.

We should add that we obtained a huge increase in sampling efficiency by using `TensorFlow 2` graph execution for sampling. Code-wise this involves using the `TensorFlow 2` decorator `tf.function`. Note, however, code has to be rewritten from `Python` native functions to `TensorFlow 2` functions.



(a) Figure shows the standardised residuals z using the geometric uncertainties $\Delta^{\pm}_{GMV}$ on the test set for the $pp \to \widetilde{\chi}^0_1\widetilde{\chi}^0_1$ cross section. The uncertainties correspond to the mixture, epistemic and aleoteric standard deviations σ_m, σ_e and σ_a. The histograms have been truncated at ± 5 and entries that lie outside are set equal to ± 5, respectively.



(b) Figure shows the standardised residuals z using the geometric uncertainties $\Delta^{\pm}_{GMV}$ on the test set for the $pp \to \widetilde{\chi}^0_2\widetilde{\chi}^+_1$ case. The uncertainties correspond to the mixture, epistemic and aleoteric standard deviations σ_m, σ_e and σ_a. The histograms have been truncated at ± 5 and entries that lie outside are set equal to ± 5, respectively.

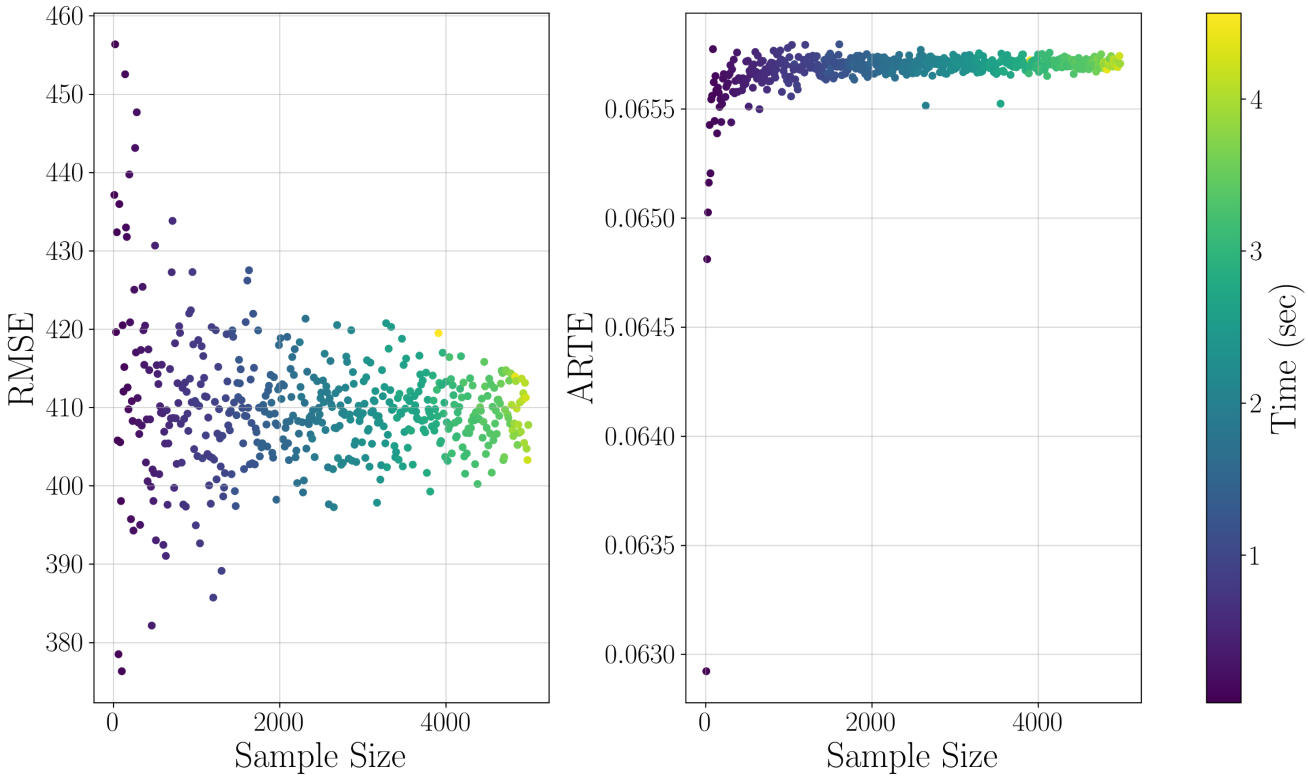


Figure 7.9: Figure show RMSE and ARTE against sample sizes. Each point corresponds to running the test model of the $\sigma(pp \to \widetilde{\chi}_2^0 \widetilde{\chi}_1^+)$ and predicting 1045 test points, so a sample size of 1 essentially means 1045 predictions. The colors denote the number of seconds it takes to run.

Conclusions

The primary goal of this thesis was to construct variational Bayesian neural network regression for speedy next-to-leading order cross section predictions in supersymmetry with reliable uncertainties. The variational Bayesian neural network (BNN) prediction computation time needed to be considerably shorter than that of the cross-section calculation software `Prospino 2.1`. This was accomplished with a sampling time of about 1000 cross section predictions per second, compared to `Prospino 2.1` that use in the order of minutes for a single prediction. The cross sections that were used as training, validation and test data were $\sigma(pp \to \widetilde{\chi}_1^0\widetilde{\chi}_1^0)$ and $\sigma(pp \to \widetilde{\chi}_2^0\widetilde{\chi}_1^+)$.

When using a variational BNN in its basic form, as described in [42], we did not succeed in making a good fit to the data, probably owing to the method's inclination to underfit. However, we found that we could get past this obstacle by using pretraining. Specifically, this involved first training a deterministic neural network with the same architecture as the BNN, and then initializing the means and standard deviations of the surrogate posterior distributions by the use of the weights and biases from the deterministic neural network.

Additionally we included the use of annealing schemes to aid the BNN during the fitting process. Using an exponential batch annealing scheme improved the BNNs ability to fit to the prior probability distribution by reducing the KL-divergence between the prior and the posterior. Two priors on the weight distribution were attempted, Gaussian and Laplace priors. While we saw some variations in the results between the two, it was not clear whether one was superior to the other.

One of our primary motivations for this to work was to obtain reliable uncertainties on the regression estimates, such that we could be able to assess the degree to which a prediction matched a true `Prospino` value without needing to actually know the true value. We used two types of uncertainties, the first was defined in a frequentist fashion, where we used the knowledge we had about the functional form of the BNN likelihood to define a measure of the spread of the posterior predictive distribution. The second method was calculating the prediction intervals of the final posterior predictive distribution, thus in a Bayesian fashion. The former of the two has an attractive feature in that it required fewer calculations than the latter, however, we believe this type of uncertainty needs to be investigated in a more rigorous mathematical manner than what was done in this thesis.

Generally the uncertainties lived up to the task of identifying when a residual was relatively large. For the frequentist type uncertainties we obtained, for both cross sections studied, standardised residuals z (See Section 6.4) that were centered on zero, $\mu_z \approx 0$. This means that the model was not prone to over or under predict. Additionally the standard deviation of the residuals was in the interval $\sigma_z = (0.7, 0.8)$ and were

tightly centered about zero, meaning that the model was generally conservative and rather systematically over-estimated the uncertainty rather than under-estimated. We saw similar behaviour by the use of our coverage metric when we included both aleoteric and epistemic uncertainties. The CAOS scores consistently scored above 0.3, or 30% of the possible conservative coverage.

Future Work and Improvement Through this proof-of-principle work we have indetified several possible directions of future research:

- **Multiple Phases of Prediction**
  We suggest producing training data in multiple phases by using the uncertainties to guide the prior probability distribution used to sample the MSSM parameters. Uncertainties over a certain threshold can be interpreted as representing areas in parameter space that are scarce, they are thus difficult for the BNN to use for prediction. To increase the accuracy we suggest these steps:

  1. Sample MSSM parameter uniformly and produce training data with `Prospino`
  2. Train the BNN and obtain uncertainties.
  3. Use the MSSM features corresponding to uncertainties above some threshold to change the parameter prior of the next training phase. Specifically, increase the probability of sampling in a region around parameters corresponding to high uncertainties.
  4. Train the BNN again with the old and new training data.

- **More Data and New Targets**
  The data set we worked with in this thesis was small relative to both what is needed for supersymmetry parameter searches and machine learning in general. For future work it would be interesting to see what a larger data set would mean in terms of learning. Additionally, only two cross sections were attempted as targets, so further quality analysis would involve checking how well the method fares with other feature–target relationships.

- **Weight Pruning**
  We have not considered weight pruning in this thesis, but it is usually the case with neural networks that after training, one can remove a large number of redundant weights with only negligible reduction of performance. The motivation is that it decreases prediction run-time. In [42] their test error is only changed from 1.24% to 1.29% when removing 95% of the BNN weights, implying there is a lot of redundancy in a trained BNN.

- **Flipout**
  In this thesis we did not use the flipout method for the pretraining case, which is essentially an extension to the method used here (See Chapter C). We believe this method would improve performance, as we managed to use it for the non-pretraining case and saw improved performance. Flipout is generally viewed in the literature as an improvement to variational BNN. The `TensorFlow` layer that provides this features is, however, of an older version of the `DenseVariational` module, and it is a bit difficult to implement it with the framework of the current code.

- **Comparing With Ground-truth**
  The method we have used should be compared with the more computationally

heavy method of Hamiltonian Monte Carlo BNN (HMC BNN) in order to compare the posterior and posterior predictive distribution with a method that samples the true posterior. This is to produce a ground-truth to assess the error of using an approximate variational method for cross section predictions. It would also be interesting to see the difference in training and prediction run times to see if an approximate method is even necessary. The package `TensorBNN` built on `TensorFlow`, is a HMC BNN package that could be used for this purpose [74].

- **Other Configurations**
  Other priors and surrogate posterior could be attempted to see if this leads to improved uncertainties or predictive performance. Additionally, there are a whole host of different approaches to improve the robustness of the variational BNN. Instead of using mean field approximation one may use structured-covariance approximations [59]. These approximations seek to increase the BNN's ability to fit the true posterior while not going all the way to full-covariance variational inference (See Section 5.6.1. Another interesting approach is to use an analog to perturbation theory, calculating corrections to the evidence lower bound [75]. Finally, using different optimization tools, e.g. instead of using the KL-divergence one could use the related α-divergence [76].

Part IV

Appendices

Appendix A

Supersymmetry formulae

A.1 Algebras

Poincaré algebra The *Poincaré algebra* is given by the following commutation relations

The Poincaré Algebra

$$\begin{aligned}
[P_\mu, P_\nu] &= 0 \\
[M_{\mu\nu}, P_\rho] &= -i\left(g_{\mu\rho}P_\nu - g_{\nu\rho}P_\mu\right) \\
[M_{\mu\nu}, M_{\rho\sigma}] &= -i\left(g_{\mu\rho}M_{\nu\sigma} - g_{\mu\sigma}M_{\nu\rho} - g_{\nu\rho}M_{\mu\sigma} + g_{\nu\sigma}M_{\mu\rho}\right),
\end{aligned} \tag{A.1}$$

where P, for all subscripts, are generators of translations, M the generators of Lorentz boosts and rotations, and g is the Minkowski metric with sign convention $(+,-,-,-)$.

Pauli Matrices The *graded Lie algebra*, L, is a direct sum of two vector spaces L_0 and L_0, $L = L_0 \oplus L_1$, with the special binary operator $\bullet$, which serves as the *grading* operator. For $x_i \in L_i$, the grading is then given as

Grading of Graded Lie Algebra

$$x_i \bullet x_j \in L_{i+j \mod 2}, \tag{A.2}$$

which means that $x_0 \bullet x_0 \in L_0, x_1 \bullet x_1 \in L_0$ and $x_0 \bullet x_1 \in L_1$, with the same pattern for L_{i+j} continuing for other indicies as a consquence of the modulo 2 operator.

Graded Lie Algebra The *Pauli matrices* are

The Pauli Matrices

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{A.3}$$

where i is the imaginary unit.

By using $\theta^A Q_A, \bar{\theta}_{\dot{A}} \bar{Q}^{\dot{A}}$ together with the anticommutation relations in Equation (1.17), one may rewrite the Super-Poincaré algebra (See Equations (1.12) to (1.15)) in terms of commutation relations

Super-Poincaré Algebra with Commutators

$$\begin{aligned} \left[\theta^A Q_A, \bar{\theta}_{\dot{B}} \bar{Q}^{\dot{B}}\right] &= 2\theta^A \sigma^\mu_{A\dot{B}} \bar{\theta}^B P_\mu \\ \left[\theta^A Q_A, \theta^B Q_B\right] &= 0 \\ \left[\bar{\theta}_{\dot{A}} \bar{Q}^A, \bar{\theta}_{\dot{B}} \bar{Q}^{\dot{B}}\right] &= 0, \end{aligned} \tag{A.4}$$

where θ_A and $\theta_{\dot{A}}$ anticommute with one another, as well as with the spinor charges Q_A and $\bar{\theta}_{\dot{A}}$.

A.2 Superspace

Scalar Superfields The general left-handed scalar superfield is

Left-handed Scalar Superfield

$$\begin{aligned} \Phi(x, \theta, \bar{\theta}) = A(x) + i\left(\theta\sigma^\mu\bar{\theta}\right)\partial_\mu A(x) - \frac{1}{4}\theta\theta\bar{\theta}\bar{\theta}\Box A(x) \\ + \sqrt{2}\theta\psi(x) - \frac{i}{\sqrt{2}}\theta\theta\partial_\mu\psi(x)\sigma^\mu\bar{\theta} + \theta\theta F(x), \end{aligned} \tag{A.5}$$

and the right-handed scaler superfield is

Right-handed Scalar Superfield

$$\begin{aligned} \Phi^\dagger(x, \theta, \bar{\theta}) = A^*(x) - i\left(\theta\sigma^\mu\bar{\theta}\right)\partial_\mu A^*(x) - \frac{1}{4}\theta\theta\bar{\theta}\bar{\theta}\Box A^*(x) \\ + \sqrt{2}\bar{\theta}\bar{\psi}(x) + \frac{i}{\sqrt{2}}\bar{\theta}\bar{\theta}\theta\sigma^\mu\partial_\mu\bar{\psi}(x) + \bar{\theta}\bar{\theta}F^*(x), \end{aligned} \tag{A.6}$$

where $\Box = -\eta^{\mu\nu}\partial_\mu\partial_\nu$ is the *d'Alembert* operator. $A(x), F(x)$ are complex scalar fields, and $\psi(x)$ $\bar{\psi}(x)$ are left-handed and right-handed Weyl spinor fields, respectively.

A.3 MSSM

Kinetic Lagrangian The kinetic terms of the MSSM are summarized by the kinetic Lagrangian

Kinetic Lagrangian

$$\begin{aligned}\mathcal{L}_{\text{kin}} =& L_i^\dagger e^{\frac{1}{2}g\sigma W-\frac{1}{2}g'B}L_i + Q_i^\dagger e^{\frac{1}{2}g_s\lambda C+\frac{1}{2}g\sigma W+\frac{1}{3}\cdot\frac{1}{2}g'B}Q_i \\ &+ \bar{U}_i^\dagger e^{\frac{1}{2}g_s\lambda C-\frac{4}{3}\cdot\frac{1}{2}g'B}\bar{U}_i + \bar{D}_i^\dagger e^{\frac{1}{2}g_s\lambda C+\frac{2}{3}\cdot\frac{1}{2}g'B}\bar{D}_i \\ &+ \bar{E}_i^\dagger e^{2\cdot\frac{1}{2}g'B}\bar{E}_i + H_u^\dagger e^{\frac{1}{2}g\sigma W+\frac{1}{2}g'B}H_u + H_d^\dagger e^{\frac{1}{2}g\sigma W-\frac{1}{2}g'B}H_d,\end{aligned} \tag{A.7}$$

where we have that g_s, g and g' are the coupling constants that pertain to the symmetry groups $SU(3)_C, SU(2)_L$ and $U(1)_Y$, respectively.

Superpotential Lagrangian The superpotential terms of the MSSM are summarized by the kinetic Lagrangian

Superpotential Lagrangian

$$\begin{aligned}W =& \mu H_u H_d + \mu_i' L_i H_u + y_{ij}^e L_i H_d \bar{E}_j + y_{ij}^u Q_i H_u \bar{U}_j + y_{ij}^d Q_i H_d \bar{D}_j \\ &+ \lambda_{ijk} L_i L_j \bar{E}_k + \lambda_{ijk}' L_i Q_j \bar{D}_k + \lambda_{ijk}'' \bar{U}_i \bar{D}_j \bar{D}_k,\end{aligned} \tag{A.8}$$

where the $SU(2)_L$ specific structure has been supressed notation-wise, so that for example $H_u^T i\sigma^2 H_d \to H_u H_d$, and equivalently for the other pairs.

Supersymmetric Field Strength Lagrangian The superpotential terms of the MSSM are summarized by the supersymmetric field strength Lagrangian

Supersymmetric Field Strength Lagrangian

$$\mathcal{L}_{\mathcal{V}} = \frac{1}{2}\,\text{Tr}\{W^A W_A\}\bar{\theta}\bar{\theta} + \frac{1}{2}\,\text{Tr}\{C^A C_A\}\bar{\theta}\bar{\theta} + \frac{1}{4}B^A B_A \bar{\theta}\bar{\theta} + \text{h.c.}, \tag{A.9}$$

with field strengths:

$$\begin{aligned} W_A &= -\frac{1}{4}\bar{D}\bar{D}e^{-W}D_A e^W, \quad W = \frac{1}{2}g\sigma^a W^a \\ C_A &= -\frac{1}{4}\bar{D}\bar{D}e^{-C}D_A e^C, \quad C = \frac{1}{2}g_s\lambda^a C^a \\ B_A &= -\frac{1}{4}\bar{D}\bar{D}D_A B, \quad B = \frac{1}{2}g'B^0 \end{aligned} \tag{A.10}$$

Appendix B

Probabilistic formulae

Here we list important probability distributions and formulae used in this thesis. The univariate Gaussian, or normal distribution, is

Univariate Gaussian

$$\mathcal{N}\left(x \mid \mu, \sigma^2\right) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}, \tag{B.1}$$

where μ is the mean, and σ^2 is the variance of the distribution. The n-dimensional generalization of the former is the multivariate Gaussian, which is

Multivariate Gaussian

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \tag{B.2}$$

where $\boldsymbol{\mu}$ is the D-dimensional vector mean, and $\boldsymbol{\Sigma}$ is the $D \times D$ covariance matrix. Along the diagonal of $\boldsymbol{\Sigma}$ are the variances σ_{ii}^2, so $\boldsymbol{\Sigma} = \mathbf{I}\boldsymbol{\sigma}^2$, with $\mathbf{I}$ being the identity, would be a diagonal covariance matrix of only variances.

The univariate Laplace distribution is,

Laplace Distribution

$$\mathcal{L}(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right), \tag{B.3}$$

where μ is the mean, and b is a scale parameter. Similar to the Gaussian, the Laplace distribution may also be generalized to n-dimensional parameter space.

The following relations are convenient for dealing with Gaussian conjugate forms of priors, likelihoods and posterior in Bayes' rule and the posterior predictive distribution.

Relations for Marginal and Conditional Gaussians

Given the marginal distribution for $\mathbf{x}$

$$p(\mathbf{x}) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}\right) \tag{B.4}$$

and the conditional distribution for $\mathbf{y}$

$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}\right) \tag{B.5}$$

we have that

$$p(\mathbf{y}) = \mathcal{N}\left(\mathbf{y} \mid \mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^{\mathrm{T}}\right) \tag{B.6}$$

$$p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\Sigma}\left\{\mathbf{A}^{\mathrm{T}}\mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}\right\}, \boldsymbol{\Sigma}\right) \tag{B.7}$$

with

$$\boldsymbol{\Sigma} = \left(\boldsymbol{\Lambda} + \mathbf{A}^{\mathrm{T}}\mathbf{L}\mathbf{A}\right)^{-1}. \tag{B.8}$$

The KL-divergence between two multivariate Gaussian (See Eq. B.2) distributions in $\mathbb{R}^n$ is

KL-Divergence Between Two Multivariate Gaussians

$$\mathbb{KL}\left(\mathcal{N}_1 \| \mathcal{N}_2\right) = \frac{1}{2}\left[\mathrm{tr}\left(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1\right) + \left(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1\right)^{\mathrm{T}}\boldsymbol{\Sigma}_2^{-1}\left(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1\right) - n + \ln\frac{|\boldsymbol{\Sigma}_2|}{|\boldsymbol{\Sigma}_1|}\right]. \tag{B.9}$$

The trace is the sum of elements along the diagonal of the matrix, $\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} a_{ii}$, while the bars signify the determinant, $|\mathbf{A}| = \sum(\pm 1)A_{1i_1}A_{2i_2}\cdots A_{Ni_N}$, where the summation runs over all products of one element from each row and column, respectively. The sign ± 1 depends on even or odd permutations, for example for a 2×2 matrix, Even: $a_{11}a_{22}$, Odd: $-a_{12}a_{21}$.

Appendix C

Flipout

A conjecture generally held in machine learning is that for stochastic gradient methods the following relationship holds: Given a loss function L_{b_t}, where b_t is the mini-batch at iteration t, the variance of the gradients is

$$var(\nabla_{\mathbf{w}} L_{b_t}) \propto \frac{n^2}{S} var(\nabla_{\mathbf{w}} L). \tag{C.1}$$

Where S is the number of samples in a mini-batch, and n is the total number of training samples in which the method samples from. For a justification of Equation (C.1) we refer to [77]. For the following discussion the important take away of Equation (C.1) is that the variance of the gradient has a linear inverse proportionality to the batch-size S. In Section 3.4.1 we implicitly discussed the effect batch size S has on the variance by making the point that SGD, that is $S = 1$, have noisy gradients. This is equivalent to stating that the gradients have high variance. Increasing the number of data points in which the gradient is calculated over by the use of mini-batch gradient descent, eases the noise, and Equation (C.1) illustrates this relationship. As we noted in Section 3.4.1, there is a trade-off at play in terms of batch-size. The variance is not an unwanted property as it reduces overfitting as described in Chapter 2, but too much variance is not advantageous either.

As mentioned in Section 5.6.3 a typical choice of batch size is $S = 32$, meaning that for every iteration of the network we calculate the gradient based on 32 data points sampled from the training data. In the Algorithm 1 we described how we perform the training process of a variational BNN, where sampling from the posterior distribution was performed by perturbing the variational parameters by a random sample ϵ. For every iteration of the algorithm we pass in one whole mini-batch of data b_t. This means that every data point in a particular mini-batch is predicted by the use of the *same* sample(for a given layer) ϵ. This stocasticity creates an addition to the variance which is *not* reduced by the number of samples S in a mini-batch. This may limit the gradient descent method's ability to converge. The method of Flipout seeks to ameliorate this issue by decorrelating the equal gradients that are implicitly within the mini-batches, such that the expected values of, $var = \mathbb{E}\|\nabla_{\mathbf{w}} L_{b_t}\|^2 - \|\mathbb{E}\nabla_{\mathbf{w}} L_{b_t}\|^2$, average out to smaller values.

We describe here the method as it is described by [78]. Two assumptions are made, that the samples ϵ are independent; and that the perturbation distribution is symmetric about zero. Both assumptions are satisfied in Algorithm 1. The following points are

implemented layer-wise, and we use matrix-notation as in Section 5.1. We assume here that the weights are sampled directly into a matrix $\mathbf{W}^{(l)}$ apt for feed-forward. The matrix has the dimensions $n^{l-1} \times n^l$, where n is the number of nodes in a given layer. Furthermore we can write $\mathbf{W}^{(l)}$ in terms of the pertubation equation (5.58)

$$\mathbf{W}^{(l)} = \mathbf{\Theta}_\mu + \Delta\mathbf{\Theta}_\sigma. \tag{C.2}$$

Where $\Delta\mathbf{\Theta}_\sigma = \mathbf{\Theta}_\sigma \circ \epsilon$. Note this is still the same operation as in Eq. (5.58), but with means and standard deviations ordered according to which nodes in the network they correspond to. In algorithm 1 we would apply these same weights for every S data points in a mini-batch.

Next let us define two random sign-vectors $\mathbf{s}_i$ and $\mathbf{r}_i$ sampled from the *Rademacher distribution*, where i runs of the data points in a mini-batch, so $i = 1, 2 \dots S$. The Rademacher distribution is a discrete probability distribution that outputs either $+1$ or -1, each with a 50% probability. The vectors are thus filled uniformly with ± 1. The vector $\mathbf{r}_i$ can be considered the input sign-vector with dimensions $n^{(l-1)} \times 1$, while $\mathbf{r}_i$ can be considered the output sign-vector with dimensions $n^{(l)} \times 1$. Because we assumed that ϵ was sampled from a distribution symmetric about zero $\Delta\mathbf{\Theta}_\sigma \circ \mathbf{r}_i\mathbf{s}_i^\intercal$ will be identically distributed to $\Delta\mathbf{\Theta}_\sigma$, as both, say a standard normal distribution, and the rademacher distribution will have 50% probability of sign $\pm$. The consequence is that Flipout gives an unbiased estimator for the gradients. As aformentioned, the matrix $\Delta\mathbf{\Theta}_\sigma$ is shared by all data in a mini-batch, but by applying the Rademacher transformation we can define unique, but equally distributed, weight pertubations for each data point, so

$$\Delta_i\mathbf{\Theta}_\sigma = \Delta\mathbf{\Theta}_\sigma \circ \mathbf{r}_i\mathbf{s}_i^\intercal \tag{C.3}$$

Let us now put the Flipout routine to use for a given layer in feed-forward propagation,

$$\begin{aligned}
\mathbf{A}_i^{(l)} &= f\left(\mathbf{W}^{(l)\intercal}\mathbf{A}_i^{(l-1)}\right) \\
&= f\left(\left(\mathbf{\Theta}_\mu^{(l)} + \Delta\mathbf{\Theta}_\sigma^{(l)} \circ \mathbf{r}_i\mathbf{s}_i^\intercal\right)^\intercal \mathbf{A}_i^{(l-1)}\right) \\
&= f\left(\mathbf{\Theta}_\mu^{(l)\intercal}\mathbf{A}_i^{(l-1)} + \left(\Delta\mathbf{\Theta}_\sigma^{(l)\intercal}\left(\mathbf{A}_i^{(l-1)} \circ \mathbf{s}_i\right)\right)\mathbf{r}_i\right).
\end{aligned} \tag{C.4}$$

Where we have omitted the bias for brevity. As before the notation $\circ$ means element-wise multiplication, so $\Delta\mathbf{\Theta}_\sigma$ and $\mathbf{r}_i\mathbf{s}_i^\intercal$ have the same dimensions. As we want to implement this operation on all data points in one go, we can vetorize it as follows: Define matrices $\mathbf{S}$ and $\mathbf{R}$, where the rows are the random sign vectors $\mathbf{s}_i$ and $\mathbf{r}_i$, respectively. They thus have S number of rows, where as always S is the number of samples in a mini-batch. The final Flipout formula is then

Flipout

$$\mathbf{A}^{(l)} = f\left(\mathbf{A}^{(l-1)}\mathbf{\Theta}_\mu^{(l)} + \left((\mathbf{A}^{(l-1)} \circ \mathbf{S})\mathbf{\Theta}_\sigma^{(l)}\right) \circ \mathbf{R}\right). \tag{C.5}$$

Bibliography

[1] S. Chatrchyan, V. Khachatryan, A. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo et al., *Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC*, *Physics Letters B* **716** (Sep, 2012) 30–61.

[2] ATLAS collaboration, G. Aad et al., *Searches for electroweak production of supersymmetric particles with compressed mass spectra in $\sqrt{s} = 13$ TeV pp collisions with the ATLAS detector*, *Phys. Rev. D* **101** (2020) 052005, [1911.12606].

[3] W. Beenakker, R. Hopker and M. Spira, *PROSPINO: A Program for the production of supersymmetric particles in next-to-leading order QCD*, hep-ph/9611232.

[4] W. Beenakker, C. Borschensky, M. Krämer, A. Kulesza, E. Laenen, S. Marzani et al., *NLO+NLL squark and gluino production cross-sections with threshold-improved parton distributions*, *Eur. Phys. J. C* **76** (2016) 53, [1510.00375].

[5] A. Buckley, A. Kvellestad, A. Raklev, P. Scott, J. V. Sparre, J. Van den Abeele et al., *Xsec: the cross-section evaluation code*, *The European Physical Journal C* **80** (Dec, 2020) .

[6] G. Johnson, *Strange beauty: Murray Gell-Mann and the revolution in twentieth-century physics*. Vintage Books, 1999.

[7] M. Badziak and K. Sakurai, *Explanation of electron and muon $g-2$ anomalies in the MSSM*, *JHEP* **10** (2019) 024, [1908.03607].

[8] Muon g-2 collaboration, B. Abi et al., *Measurement of the Positive Muon Anomalous Magnetic Moment to 0.46 ppm*, *Phys. Rev. Lett.* **126** (2021) 141801, [2104.03281].

[9] N. Aghanim, Y. Akrami, M. Ashdown, J. Aumont, C. Baccigalupi, M. Ballardini et al., *Planck 2018 results*, *Astronomy & Astrophysics* **641** (Sep, 2020) A6.

[10] A. Raklev, *Supersymmetry lecture notes.*, https://www.uio.no/studier/emner/matnat/fys/FYS5190/h19/pensumliste/notes.pdf, November, 2019.

[11] S. R. Coleman and J. Mandula, *All Possible Symmetries of the S Matrix*, *Phys. Rev.* **159** (1967) 1251–1256.

[12] R. Haag, J. T. Lopuszanski and M. Sohnius, *All Possible Generators of Supersymmetries of the s Matrix*, *Nucl. Phys. B* **88** (1975) 257.

[13] M. F. Sohnius, *Introducing supersymmetry*, vol. 128. Elsevier B.V., 1985.

[14] G. L. Kane, ed., *Perspectives on supersymmetry. Vol.2.* World Scientific Publishing Company, Incorporated, 2010.

[15] GAMBIT collaboration, P. Athron et al., *Combined collider constraints on neutralinos and charginos*, *Eur. Phys. J. C* **79** (2019) 395, [1809.02097].

[16] J. Debove, B. Fuks and M. Klasen, *Threshold resummation for gaugino pair production at hadron colliders*, *Nuclear Physics B* **842** (Jan, 2011) 51–85.

[17] M. Spira, *Gaugino pair production at hadron colliders*, *Nucl. Phys. B Proc. Suppl.* **89** (2000) 222–227.

[18] W. Beenakker, M. Klasen, M. Kramer, T. Plehn, M. Spira and P. M. Zerwas, *The Production of charginos / neutralinos and sleptons at hadron colliders*, *Phys. Rev. Lett.* **83** (1999) 3780–3783, [hep-ph/9906298].

[19] G. P. Lepage, *Vegas: An adaptive multidimensional integration program*, *Cornell Univ. Lab. Nucl. Stud., Ithaca, NY* (3, 1980) .

[20] C. Alexandrou, K. Cichy, M. Constantinou, K. Jansen, A. Scapellato and F. Steffens, *Light-Cone Parton Distribution Functions from Lattice QCD*, *Phys. Rev. Lett.* **121** (2018) 112001, [1803.02685].

[21] M. Thomson, *Modern particle physics*. Cambridge University Press, New York, 2013.

[22] P. M. Nadolsky, H.-L. Lai, Q.-H. Cao, J. Huston, J. Pumplin, D. Stump et al., *Implications of CTEQ global analysis for collider observables*, *Phys. Rev. D* **78** (2008) 013004, [0802.0007].

[23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche et al., *Mastering the game of go with deep neural networks and tree search*, *Nature* **529** (2016) 484–503.

[24] T. Hastie, R. Tibshirani and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 ed., 2009.

[25] H. I. Choi, *Machine learning, lecture notes: Model selection*, https://www.math.snu.ac.kr/~hichoi/machinelearning/lecturenotes/ModelSelection.pdf, 2017. Online accessed on 20.05.2021.

[26] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[27] M. Hjorth-Jensen, *Lecture notes: FYSSTK4155 Applied Data Analysis and Machine Learning*, https://github.com/CompPhysics/MachineLearning, 2020. Online accessed on 20-05-2021.

[28] D.-A. Clevert, T. Unterthiner and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus).*, in *ICLR (Poster)* (Y. Bengio and Y. LeCun, eds.), 2016.

[29] G. Klambauer, T. Unterthiner, A. Mayr and S. Hochreiter, *Self-normalizing neural networks*, *NIPS* (2017) .

[30] A. L. Maas, *Rectifier nonlinearities improve neural network acoustic models*, *in Proc. ICML* (2013) .

[31] P. Ramachandran, B. Zoph and Q. V. Le, *Searching for activation functions.*, in *ICLR (Workshop)*, OpenReview.net, 2018.

[32] D. Misra, *Mish: A Self Regularized Non-Monotonic Activation Function*, in *Proceedings of the British Machine Vision Conference*, 2020. 1908.08681.

[33] A. Amini, A. Soleimany, S. Karaman and D. Rus, *Spatial Uncertainty Sampling for End-to-End Control*, 1805.04829.

[34] B. Recht and C. Re, *Toward a noncommutative arithmetic-geometric mean inequality: Conjectures, case-studies, and consequences*, in *Proceedings of the 25th Annual Conference on Learning Theory* (S. Mannor, N. Srebro and R. C. Williamson, eds.), vol. 23 of *Proceedings of Machine Learning Research*, pp. 11.1–11.24, JMLR Workshop and Conference Proceedings, 25–27 Jun, 2012.

[35] G. H. T Tieleman, *Lectures notes on neural networks for machine learning, with rmsprop*, https://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf, 2012.

[36] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[37] K. P. Murphy, *Machine learning : a probabilistic perspective*. The MIT Press, 2012.

[38] E. Jaynes, E. Jaynes, G. Bretthorst and C. U. Press, *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[39] J. B. Jeremy Orloff, *Introduction to probability and statistics*, 2014. MIT OpenCourseWare, Online accessed on 20.05.2021.

[40] A. Gelman, J. B. Carlin, H. S. Stern and D. B. Rubin, *Bayesian Data Analysis*. Chapman & Hall/CRC, 2nd ed., Boca Raton, USA, 2003.

[41] D. Sivia and J. Skilling, *Data Analysis: A Bayesian Tutorial*. Oxford science publications. OUP Oxford, 2006.

[42] C. Blundell, J. Cornebise, K. Kavukcuoglu and D. Wierstra, *Weight uncertainty in neural network*, in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, pp. 1613–1622, PMLR, 07–09 Jul, 2015.

[43] V. Antun, F. Renna, C. Poon, B. Adcock and A. C. Hansen, *On instabilities of deep learning in image reconstruction and the potential costs of ai*, *Proceedings of the National Academy of Sciences* **117** (May, 2020) 30088–30095.

[44] A. D. Cobb and B. Jalaian, *Scaling hamiltonian monte carlo inference for bayesian neural networks with symmetric splitting.*, .

[45] J. Dy and A. Krause, *Variational Bayesian dropout: pitfalls and fixes*, in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80 of *Proceedings of Machine Learning Research*, PMLR, 2018.

[46] B. Lakshminarayanan, A. Pritzel and C. Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles*, in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et al., eds.), vol. 30, Curran Associates, Inc., 2017.

[47] L. Hoffmann and C. Elster, *Deep ensembles from a bayesian perspective*, May, 2021. unpublished.

[48] M. Humt, J. Lee and R. Triebel, *Bayesian optimization meets laplace approximation for robotic introspection*, *IEEE/RSJ Intelligent Conference on Intelligent Robots and Systems (IROS)* (2020) , [2010.16141].

[49] R. Zhang, C. Li, J. Zhang, C. Chen and A. G. Wilson, *Cyclical stochastic gradient mcmc for bayesian deep learning*, in *International Conference on Learning Representations*, 2020.

[50] W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov and A. G. Wilson, *A simple baseline for bayesian uncertainty in deep learning*, in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[51] Z. Zhao, A. Pozas-Kerstjens, P. Rebentrost and P. Wittek, *Bayesian deep learning on a quantum computer*, *Quantum Machine Intelligence* **1** (May, 2019) 41–51.

[52] D. Cline and M. Sarkis, *Variational Principles in Classical Mechanics*. River Campus Libraries, 2017.

[53] C. E. Shannon, *A mathematical theory of communication.*, *Bell Syst. Tech. J.* **27** (1948) 379–423.

[54] L. Reichl, *A Modern Course in Statistical Physics*. Physics textbook. Wiley, 2009.

[55] E. Jang, *A beginner's guide to variational methods: Mean-field approximation*, https://blog.evjang.com/2016/08/variational-bayes.html. Online accessed on 04.04.2021.

[56] A. Graves, *Practical variational inference for neural networks*, in *Advances in Neural Information Processing Systems* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira and K. Q. Weinberger, eds.), vol. 24, Curran Associates, Inc., 2011.

[57] G. Gundersen, *The reparameterization trick*, https://gregorygundersen.com/blog/2018/04/29/reparameterization/. Online accessed on 20.04.2021.

[58] S. Farquhar, L. Smith and Y. Gal, *Liberty or depth: Deep bayesian neural nets do not need complex weight posterior approximations*, https://oatml.cs.ox.ac.uk/blog/2020/11/29/liberty_or_depth.html, 2021.

[59] W. Lin, M. E. Khan and M. Schmidt, *Fast and simple natural-gradient variational inference with mixture of exponential-family approximations*, in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 3992–4002, PMLR, 09–15 Jun, 2019.

[60] S. Sun, G. Zhang, J. Shi and R. Grosse, *Functional variational Bayesian neural networks*, in *International Conference on Learning Representations*, 2019.

[61] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, https://www.tensorflow.org/, 2015. Software available from tensorflow.org.

[62] L. V. Jospin, W. Buntine, F. Boussaid, H. Laga and M. Bennamoun, *Hands-on bayesian neural networks – a tutorial for deep learning users*, 2020. unpublished.

[63] C. T. Lewis and C. Short, *A Latin Dictionary*. Oxford: Clarendon Press, 1879.

[64] V. Fortuin, A. Garriga-Alonso, F. Wenzel, G. Ratsch, R. E. Turner, M. van der Wilk et al., *Bayesian neural network priors revisited*, in *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

[65] R. Krishnan, M. Subedar and O. Tickoo, *Efficient priors for scalable variational inference in bayesian deep neural networks*, in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 773–777, 2019.

[66] B. C. Allanach, *SOFTSUSY: a program for calculating supersymmetric spectra*, *Comput. Phys. Commun.* **143** (2002) 305–331, [hep-ph/0104145].

[67] P. Z. Skands et al., *SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators*, *JHEP* **07** (2004) 036, [hep-ph/0311123].

[68] A. Buckley, *PySLHA: a Pythonic interface to SUSY Les Houches Accord data*, *Eur. Phys. J. C* **75** (2015) 467, [1305.4194].

[69] Wikipedia contributors, *Log-normal distribution — Wikipedia, the free encyclopedia*, 2021. [Online; accessed 3-June-2021].

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel et al., *Scikit-learn: Machine learning in Python*, *Journal of Machine Learning Research* **12** (2011) 2825–2830.

[71] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau et al., *Scipy 1.0: fundamental algorithms for scientific computing in Python*, *Nature Methods* **17** (2020) 261–272, [1907.10121].

[72] V. Nair and G. Hinton, *Rectified linear units improve restricted boltzmann machines vinod nair*, *Proceedings of ICML* **27** (06, 2010) 807–814.

[73] A. Savitzky and M. J. E. Golay, *Smoothing and differentiation of data by simplified least squares procedures.*, *Analytical Chemistry* **36** (1964) 1627–1639.

[74] B. Kronheim, M. Kuchera and H. Prosper, *Tensorbnn: Bayesian inference for neural networks using tensorflow*, 2020. Preprint at https://arxiv.org/pdf/2009.14393v2.pdf.

[75] R. Bamler, C. Zhang, M. Opper and S. Mandt, *Tightening bounds for variational inference by revisiting perturbation theory*, *Journal of Statistical Mechanics: Theory and Experiment* **2019** (Dec, 2019) 124004.

[76] Y. Li and R. E. Turner, *Rényi divergence variational inference*, in *Advances in Neural Information Processing Systems* (D. Lee, M. Sugiyama, U. Luxburg, I. Guyon and R. Garnett, eds.), vol. 29, Curran Associates, Inc., 2016.

[77] X. Qian and D. Klabjan, *The impact of the mini-batch size on the variance of gradients in stochastic gradient descent*, 2020. unpublished.

[78] Y. Wen, P. Vicol, J. Ba, D. Tran and R. Grosse, *Flipout: Efficient pseudo-independent weight perturbations on mini-batches*, in *International Conference on Learning Representations*, 2018.