UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER EN TRATAMIENTO ESTADÍSTICO COMPUTACIONAL DE LA INFORMACIÓN

TRABAJO DE FIN DE MÁSTER

Aplicación de redes neuronales a la clasificación y propagación de incertidumbre de procesos físicos en el experimento CMS, CERN

Julia Vázquez Escobar

Codirectores CIEMAT

Miguel Cárdenas Montes y José María Hernández Calama

Ponente UPM

Pedro José Zufiria Zatarain

Madrid, 2020

Resumen

Los experimentos que se llevan a cabo en el ámbito de la Física de Partículas conllevan la producción de grandes cantidades de información. Para poder producir y detectar procesos de interés de baja probabilidad, que consideramos señal, se deben generar un gran número de colisiones entre partículas. Esto produce conjuntos de datos en los que la mayoría de observaciones no son de interés, se consideran fondo. Es necesario recurrir a algún procedimiento que sea capaz de diferenciar de manera eficiente entre señal y fondo.

En este trabajo se desarrolla una red neuronal con ese objetivo. Normalmente, las redes neuronales se usan de manera que ofrecen predicciones puntuales. Sin embargo, muchas ramas de la ciencia requieren resultados acompañados de medidas de incertidumbre para poder evaluar su nivel de precisión. En este proyecto, las predicciones de la red neuronal se acompañan con una estimación de su incertidumbre. Para tal fin se usan Técnicas de Regularización Estocástica aplicadas a un Perceptrón Multicapa. Existe una equivalencia entre optimizar usando redes neuronales de este tipo y redes neuronales Bayesianas, que plantean un modelo de red probabilístico. Este hecho nos permite calcular la esperanza de las predicciones y su varianza.

En este trabajo se construye una arquitectura de este tipo para identificar la producción y desintegración de pares de quarks top-antitop en colisiones de protones llevadas a cabo en el colisionador de hadrones LHC en el CERN. Se usa un conjunto de datos del experimento CMS que contiene información detallada de simulaciones de estas colisiones, con procesos tanto de señal como de fondo. La red neuronal desarrollada demuestra una excelente capacidad de discriminación entre señal y fondo.

Abstract

Particle physics experiments entail large data collection. In order to produce and detect low probability processes of interest (signal), a huge number of particle collisions must be carried out. This procedure produces huge sets of observations where most of them are of no interest (background). A mechanism able to differentiate rare signals buried in immense backgrounds is required.

In this project, a neural network is used to classify particle physics events according to their nature. Usually neural networks provide predictions as mere point estimations. However, many science fields require uncertainty measures to validate the results. Obtaining expected results with their variance would give us more information about the scope of the network. To achieve this, Stochastic Regularization Techniques are applied to a Multilayer Perceptron. There is an equivalence between optimising performance with a neural network of this kind and with a Bayesian neural network, which defines a model from a probabilistic point of view, so that uncertainty measures naturally come up. This fact allows us to calculate, not only point estimations, but expected results with their variance.

An architecture with those properties is built for the identification of the production and decay of top-antitop quark pairs in collisions of protons at the Large Hadron Collider at CERN. Datasets of detailed simulations of the signal and background processes elaborated by the CMS experiment are used. Results, stemming from the network, are presented together with uncertainty measures, showing the great discrimination power achieved.

Contents

Introduction

Particle physics is the study of fundamental constituents of matter and radiation and the interaction between them. For this purpose, experiments involving high energy particle collisions are conducted using specialised detectors that record the properties of the particles produced in the collisions. Many elementary particles do not exist under ordinary circumstances, so these experiments make it possible to create and detect them.

Typically, the processes of interest rarely occur. They can involve the discovery of a new particle, or just the study of events that hard to reproduce. The probability of these processes occurring can be so low that millions or even billions of collisions are needed to produce and detect such events. This situation gives rise to a huge volume of data to store and analyze. Indeed tens of petabytes of data are recorded every year at the Large Hadron Collider (LHC) [Cen] located in the European Centre for Particle Physics Research (CERN).

The LHC is the largest particle collider in the world. It performs this kind of experiments accelerating protons until reaching 99.9999991% the speed of light. Protons are collided about 10^9 times per second, originating 40 TB of information per second in the detector. This unmanageable amount of data is filtered in real time by specialized electronics, producing an output of about 1 GB/s or 10 petabytes of raw data per year. Evaluating datasets of this magnitude just to study concrete types of events is a tedious task and it consumes computing resources. For rare processes, the proportion of interesting events as compared to the background can be really low. For example, only one in about 10^{10} proton collisions at the LHC produces a Higgs boson, the elementary particle discovered at the LHC in 2012.

The identification of tiny amounts of signal inside a huge dataset is a complex task. It involves the analysis of multiple event features such as the number and type of particles produced, kinematic and topological distributions, etc. Neural networks can help optimizing this selection process improving the signal identification efficiency and background supression. By performing a binary classification, we can predict with high accuracy whether or not the observations collected are signal or background, obtaining a set of events, where the proportion of signal is much higher, that can be used for further physics analyses.

Estimating the uncertainty in the measurement of any physics quantity is of crucial importance. In this work, we follow [Gal16] to obtain uncertainty measures when using neural networks. Bayesian neural networks (BNN) [Mac92] are artificial neural networks whose weights are considered as random variables. We could calculate posterior probabilities for these weights, given some observed data, using a Bayesian approach, having then, uncertainty measures. The problem is that calculating these posterior probabilities becomes intractable for a network with more than one hidden layer.

An alternative approach has been proposed, using Stochastic Regularisation Techniques (SRT), more specifically dropout. It has been shown that optimisation by the use of Bayesian neural networks is equivalent to optimise by the use of regular neural networks with dropout [SHK+14].

Two results are key in this area: *i)* the expected value for the predictions of a Bayesian neural network can be estimated with various predictions from regular neural networks with dropout, and *ii)* the second moment for these predictions can be estimated analogously. Hence we can obtain uncertainty measures.

The techniques described above are applied to a dataset related to the Compact Muon Solenoid (CMS)[SC14] experiment at the LHC. The dataset consists of detailed simulations of detector reconstruction of two types of events, events containing the decay products of a top-antitop ($t\bar{t}$) quark pair, considered as signal, and another category of events (including several processes) considered as background. Features for the events are built from the data reconstructed by the detector. The goal is to discern signal $t\bar{t}$ events from the background events, which in nature occur approximately with a proportion of 1 to 400. A large suppression factor of the background, while keeping a reasonable selection efficiency on the signal is required to isolate $t\bar{t}$ events for subsequent physics studies of the properties of the top quark.

Mathematical background

In this section we describe the type of neural network that is going to be used as a classification tool, the multilayer perceptron.

In order to estimate the uncertainty of the performance of the model, an equivalence is established between Bayesian neural networks and the application of dropout to a multilayer perceptron [Gal16]. This allows us to perform inference over the predictions of the network with dropout, estimating a variance analogous to what we would obtain if we used a raw Bayesian approach.

1.1 Multilayer perceptron

We face a classification problem. Neural networks are used as a resource to solve it. More specifically, a multilayer perceptron, an artificial neural network, is the chosen tool. We briefly introduce this network in this section and explain how it works. For further information on neural networks and other deep learning techniques, the book [GBC16] is referred.

In this section, we explore the key aspects of a multilayer perceptron. This kind of neural networks are deep learning models whose task is, given some observed data (x, y), to find a function f^* that best fits $y = f^*(x)$. Typically, some bias is imposed by defining a family of parametric functions $f(x; \omega) = f^{\omega}(x)$. An optimisation algorithm adjusts the value of ω so the best approximation for $y = f^{\omega}(x)$ is reached.

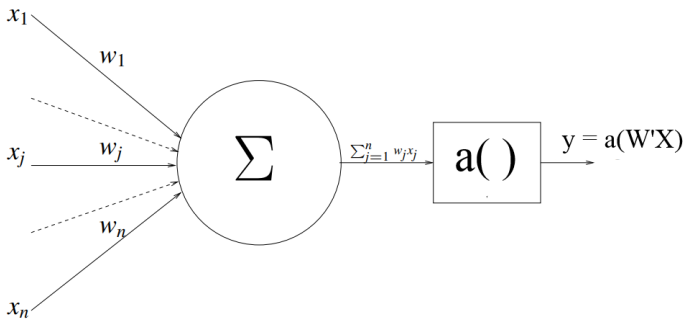


Figure 1.1: Single neuron process

A multilayer perceptron consists of one or more hidden layers each of them consisting of several neurons. In each neuron the arriving input is transformed, first linear operations are performed and then a nonlinear activation function is applied (see Fig. 1.1). The input is a multidimensional observation $X = (x_1, ..., x_n)$ or several of them if the network works in batches. It is transformed using a vector of weights $W = (w_1, ..., w_n)$, performing the scalar product $\sum_{i=1}^{n} = w_i x_i = W^T X$. The final step to obtain the output involves the application of a nonlinear function called the activation function $a()$. This could be a sigmoid [Mit97] or Relu [NH10] function, for example. The final output is therefore $y = a(W^T X)$.

The complete scheme of the network could look something like Fig. 1.2, where every neuron of the hidden layers performs the explained operations. The layers are dense which means that the neurons are fully connected with the neurons of the next layer. The objective of our work is to perform a binary classification, therefore the target variable, that should match the output, is chosen to be one dimensional, just as the example in Fig. 1.2.

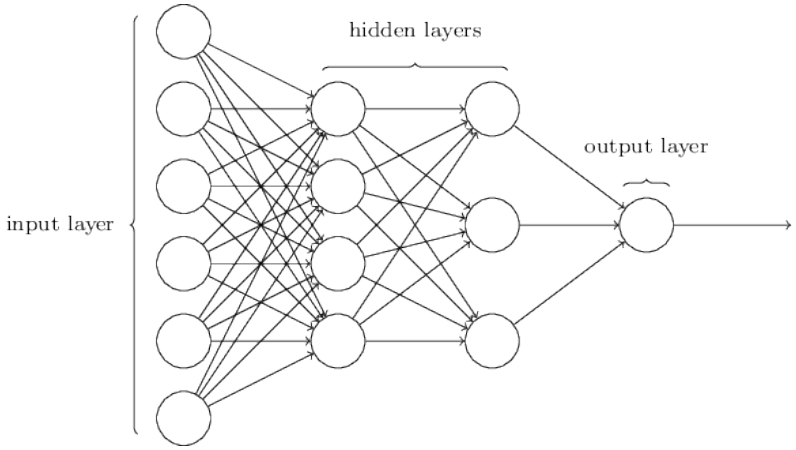


Figure 1.2: Example of a MLP

Given the input data $\underline{X} = \{X_1, \ldots, X_M\}$, we want an output as close as possible to a target variable $Y = \{y_1, \ldots, y_M\}$. In order to optimize the output, an error or loss function is defined, quantifying the difference between output and target values.

$$E(W) = \mathcal{L}(MLP(\underline{X}; W), Y), \tag{1.1}$$

where $MLP(\underline{X}; W)$ denotes the final output of the network (MLP stands for MultiLayer Perceptron).

To compute the error we use some loss function $\mathcal{L}$. This function must increase when the prediction differs from the true target Y. Some of the loss functions often used are the *mean squared error*, *mean absolute error*, *crossentropy*, etc.

The objective is therefore to find the parameters W that minimize the loss function. Since we face an optimization problem, an optimization algorithm is required. Gradient descent algorithms are the usual choice when drawing on neural networks.

An optimization algorithm of this type consists in looking for a solution in the direction of the gradient of the objective function. We start the procedure in a random feasible solution. Then, the gradient of the objective function is computed. Since the term $MLP(\underline{X}; W)$ can be extremely complex, the gradient of the objective function may turn tricky to compute. In [RHW86], backpropagation is described as a tool to obtain this gradient.

The solution is updated following the equation $W_{t+1} = W_t - \epsilon \nabla_W f(W_t)$[1] so it goes in the direction that makes the objective function f decrease. The update is repeated until some specified criteria is met. There are several options. If we choose reaching a minimum as ending condition we may end in a local minimum. We can also set a maximum of steps or a minimal difference between consecutive solutions. The specific method used is mentioned in chapter 3.

1.2 Bayesian neural networks

In our work, we would like to deliver the predictions of the neural network with their estimated uncertainty. In physics it is crucial to assess the precision of a measurement. The content of this section is based on the thesis work in [Gal16] where an equivalence between Bayesian neural networks and ordinary networks with dropout is achieved. The author demonstrates that the variance and expected value of the predictions that a Bayesian networks would provide can be estimated performing various predictions with an artificial network with dropout.

1.2.1 Bayesian model

In order to study neural networks from a probabilistic point of view, we use Bayesian modelling. Given training data $X = \{x_1, \ldots, x_N\}$ and some corresponding target variable $Y = \{y_1, \ldots, y_N\}$, we would like to find an equation to model their relationship, $y = f^{\omega}(x)$, function f depending on some parameters ω. If we consider the parameters ω as random variables it is natural to wonder which is their probability distribution. Taking the Bayesian approach, the posterior probability of ω conditioned to the observed data can be calculated using some *prior* distribution $p(\omega)$, through the Bayes' theorem:

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega}) p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})} \tag{1.2}$$

A problem arises when the posterior distribution $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ cannot be obtained analytically. An alternative is to define a family of parametric functions $q_\theta(\boldsymbol{\omega})$ whose structure is

[1] ϵ is the step size parameter, adequately chosen for a good optimising process.

treatable. Since we want this distribution to be as close as possible to the original distribution, we try to minimize the Kullback–Leibler (KL) divergence w.r.t. θ [Kul59] between both:

$$\mathrm{KL}\left(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\right) = \int q_\theta(\boldsymbol{\omega}) \log \frac{q_\theta(\boldsymbol{\omega})}{p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})} \mathrm{d}\boldsymbol{\omega}, \tag{1.3}$$

which is only defined when $q_\theta(\boldsymbol{\omega})$ is absolutely continuous with regard to $p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})$. We denote $q_\theta^*(\omega)$ as the solution that minimizes 1.3. This solution would allow us to obtain a predictive distribution:

$$p\left(\mathbf{y}^*|\mathbf{x}^*,\mathbf{X},\mathbf{Y}\right) \approx \int p\left(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega}\right) q_\theta^*(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} =: q_\theta^*\left(\mathbf{y}^*|\mathbf{x}^*\right). \tag{1.4}$$

We point out that minimizing eq. 1.3 is equivalent to maximizing the expression:

$$\mathcal{L}_{\mathrm{VI}}{}^2(\theta) := \int q_\theta(\boldsymbol{\omega}) \log p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} - \mathrm{KL}\left(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega})\right). \tag{1.5}$$

1.2.2 Bayesian Neural Networks

Following these concepts, Bayesian Neural Networks were first introduced in the '90s. They can be understood as probabilistic neural networks. They are distinguished for inferring distributions over the models' weights. The network parameters ω are established as random variables so we can consider obtaining uncertainty measures. Consider, for example, the multilayer perceptron and think of the weights of each neuron as random variables.

Referring to the posterior probability defined in eq. 1.2, we could now ask ourselves the following: given some known data (X, Y), what distribution do the parameters of the Bayesian neural network follow? Gaussian prior distributions are often placed over the weights of the network yet the posterior probability is, in most cases, not tractable due to the complexity of the expression of the output, in terms of the input and weights.

Different approaches have been taken in an attempt to give functionality to these neural networks. An example is the work [HvC93], in which the authors assume independence of each weight from all the others in the network. This results in a factorised distribution $p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})$ when we try to approximate it.

Recalling the variational inference techniques characterized in eq. 1.3, we can express the divergence as in eq. 1.6. The goal is to find the optimum $q_\theta^*(\omega)$ that minimizes this quantity:

[2]We denote the expression as $\mathcal{L}_{\mathrm{VI}}$ because it is obtained through a *Variational Inference* procedure [JGJS99]. While the Bayesian approach tries to calculate the probability of interest by marginalizing it, the variational inference approach suggests an optimisation problem.

$$\begin{aligned}\mathrm{KL}\left(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\right) &\propto -\int q_\theta(\boldsymbol{\omega})\log p(\mathbf{Y}|\mathbf{X},\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}\left(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\right) \\ &= -\sum_{i=1}^{N}\int q_\theta(\boldsymbol{\omega})\log p\left(\mathbf{y}_i|\mathbf{f}^\omega\left(\mathbf{x}_i\right)\right)\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}\left(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\right) = \mathcal{L}_{\mathrm{VI}}(\theta).\end{aligned} \tag{1.6}$$

Still, evaluating this expression can be difficult. The terms $\int q_\theta(\boldsymbol{\omega})\log p\left(\mathbf{y}_i|\mathbf{f}^\omega\left(\mathbf{x}_i\right)\right)\mathrm{d}\boldsymbol{\omega}$ are not tractable for BNNs with more than one single hidden layer. Even though we could, obtaining these for every data observation, (x_i,y_i) $\forall i = 1,...,N$, becomes computationally costly.

The author solves this last problem by sampling the data. The expression 1.6 becomes 1.7. The integral is only evaluated for a selection of indexes $i \in S$, being S a subset of the original data of size M.

$$\hat{\mathcal{L}}_{\mathrm{VI}}(\theta) := -\frac{N}{M}\sum_{i\in S}\int q_\theta(\boldsymbol{\omega})\log p\left(\mathbf{y}_i|\mathbf{f}^\omega\left(\mathbf{x}_i\right)\right)\mathrm{d}\boldsymbol{\omega} + \mathrm{KL}\left(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\right). \tag{1.7}$$

This last expression is an unbiased stochastic estimator of $\mathcal{L}_{\mathrm{VI}}(\theta)$, $\mathbb{E}\left[\hat{\mathcal{L}}_{\mathrm{VI}}(\theta)\right] = \mathcal{L}_{\mathrm{VI}}(\theta)$. If we use a stochastic optimiser to optimise expression 1.7, we would thus obtain a local optimum of 1.5 [RM51].

To evaluate the terms $\int q_\theta(\boldsymbol{\omega})\log p\left(\mathbf{y}_i|\mathbf{f}^\omega\left(\mathbf{x}_i\right)\right)\mathrm{d}\boldsymbol{\omega}$, Monte Carlo integration techniques are considered. A final expression 1.8 is obtained by using a *pathwise derivative* estimator. It also meets the condition $\mathbb{E}\left[\hat{\mathcal{L}}_{\mathrm{MC}}(\theta)\right] = \mathcal{L}_{\mathrm{VI}}(\theta)$:

$$\widehat{\mathcal{L}}_{\mathrm{MC}}(\theta) = -\frac{N}{M}\sum_{i\in S}\log p\left(\mathbf{y}_i|\mathbf{f}^\omega\left(\mathbf{x}_i\right)\right) + \mathrm{KL}\left(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\right). \tag{1.8}$$

The work in [Rub81] leads to an equivalence between optimising $\widehat{\mathcal{L}}_{\mathrm{MC}}(\theta)$ and $\mathcal{L}_{\mathrm{VI}}(\theta)$.

Distributions for the predictions follow eq. 1.4. The integral involved is also approximated with Monte Carlo integration techniques:

$$\begin{aligned}\tilde{q}_\theta\left(\mathbf{y}^*|\mathbf{x}^*\right) := \frac{1}{T}\sum_{t=1}^{T} p\left(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega}_t\right) &\underset{T\to\infty}{\longrightarrow} \int p\left(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega}\right) q_\theta(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega} \\ &\approx \int p\left(\mathbf{y}^*|\mathbf{x}^*,\boldsymbol{\omega}\right) p(\boldsymbol{\omega}|\mathbf{X},\mathbf{Y})\mathrm{d}\boldsymbol{\omega} \\ &= p\left(\mathbf{y}^*|\mathbf{x}^*,\mathbf{X},\mathbf{Y}\right),\end{aligned} \tag{1.9}$$

where $\hat{\boldsymbol{\omega}}_t \sim q_\theta(\boldsymbol{\omega})$.

The bayesian model for the network easily accomodates the possibility of obtaining uncertainty when working with these models, but the problem is that, most often, it is not tractable. Stochastic regularisation techniques [Gal16] are presented as an alternative to obtain predictions with uncertainty measures.

1.3 Dropout technique

The regularization technique we are going to use is *Dropout* [SHK+14]. Deep learning models are likely to overfit the training data set quickly. Dropout is a technique often used to obtain models that are not overfitted. It consists of randomly shutting down values, set them to 0, when training the model.

To explain its use, we consider a single hidden layer neural network. We sample two binary vectors $\widehat{\epsilon}_1, \widehat{\epsilon}_2$. $\widehat{\epsilon}_1$ must have the same dimension as the input and $\widehat{\epsilon}_2$ the same as the output. We select two parameters $0 \leq p_i \leq 1$ $i = 1, 2$. The elements of $\widehat{\epsilon}_i$ are null with probability p_i, they take the value 1 otherwise. The input data x is transformed $\hat{\mathbf{x}} = \mathbf{x} \odot \widehat{\epsilon}_1$[3]. The output of the layer is also transformed, $\mathbf{h} = \sigma\left(\hat{\mathbf{x}}\mathbf{M}_1 + \mathbf{b}\right)$ to $\widehat{\mathbf{h}} = \mathbf{h} \odot \widehat{\epsilon}_2$. The final output of the model would be $\hat{\mathbf{y}} = \widehat{\mathbf{h}}\mathbf{M}_2$[4].

Since we use a multilayer perceptron, we apply this procedure to each hidden layer. This way we are providing some random noise to the final output, mimicking Bayesian neural networks, where the stochasticity comes from the fact that the model parameters are considered random variables. Let's compare both of them more thoroughly.

We can express the output of the model with dropout (with only one layer) as:

$$\begin{aligned}
\hat{\mathbf{y}} &= \hat{\mathbf{h}}\mathbf{M}_2 \\
&= \left(\mathbf{h} \odot \hat{\boldsymbol{\epsilon}}_2\right)\mathbf{M}_2 \\
&= \left(\mathbf{h} \cdot \operatorname{diag}\left(\hat{\boldsymbol{\epsilon}}_2\right)\right)\mathbf{M}_2 \\
&= \mathbf{h}\left(\operatorname{diag}\left(\hat{\boldsymbol{\epsilon}}_2\right)\mathbf{M}_2\right) \\
&= \sigma\left(\hat{\mathbf{x}}\mathbf{M}_1 + \mathbf{b}\right)\left(\operatorname{diag}\left(\hat{\boldsymbol{\epsilon}}_2\right)\mathbf{M}_2\right) \\
&= \sigma\left(\left(\mathbf{x} \odot \hat{\boldsymbol{\epsilon}}_1\right)\mathbf{M}_1 + \mathbf{b}\right)\left(\operatorname{diag}\left(\hat{\boldsymbol{\epsilon}}_2\right)\mathbf{M}_2\right) \\
&= \sigma\left(\mathbf{x}\left(\operatorname{diag}\left(\hat{\boldsymbol{\epsilon}}_1\right)\mathbf{M}_1\right) + \mathbf{b}\right)\left(\operatorname{diag}\left(\hat{\boldsymbol{\epsilon}}_2\right)\mathbf{M}_2\right)
\end{aligned}$$

The estimated weights would be $\widehat{\mathbf{W}}_1 := \operatorname{diag}\left(\widehat{\epsilon}_1\right)\mathbf{M}_1$ and $\widehat{\mathbf{W}}_2 := \operatorname{diag}\left(\widehat{\epsilon}_2\right)\mathbf{M}_2$, we denote the output as $\hat{\mathbf{y}} = \sigma\left(\mathbf{x}\widehat{\mathbf{W}}_1 + \mathbf{b}\right)\widehat{\mathbf{W}}_2 =: \mathbf{f}^{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}}(\mathbf{x})$, in the next lines we use $\widehat{\omega} = \{\widehat{\mathbf{W}}_1, \widehat{\mathbf{W}}_2, \mathbf{b}\}$ to shorten the expressions.

The loss function for this model could be this one:

$$\widehat{\mathcal{L}}_{\text{dropout}}\left(\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}\right) := \frac{1}{M}\sum_{i \in S}\left\|\mathbf{y}_i - \hat{\mathbf{y}}_i\right\|^2 + \lambda_1\left\|\mathbf{M}_1\right\|^2 + \lambda_2\left\|\mathbf{M}_2\right\|^2 + \lambda_3\|\mathbf{b}\|^2, \tag{1.10}$$

[3] $\odot$ indicates the element-wise product.

[4] We denote the weights as M to distinguish deterministic values from the weights considered as random variables (W) or estimations ($\widehat{W}$).

where we take the average square error into account but also the complexity of the model. S is a subset of the data of size M. According to [TLS89],we can replace the average square error by the negative log-likehood scaled by a constant.

$$\widehat{\mathcal{L}}_{\text{dropout}}\ (\mathbf{M}_1, \mathbf{M}_2, \mathbf{b}) = -\frac{1}{M\tau}\sum_{i\in S}\log p\left(\mathbf{y}_i|\mathbf{f}^{g(\theta,\hat{\varepsilon}_1)}(\mathbf{x})\right) + \lambda_1\left\|\mathbf{M}_1\right\|^2 + \lambda_2\left\|\mathbf{M}_2\right\|^2 + \lambda_3\|\mathbf{b}\|^2, \tag{1.11}$$

where $p\left(\mathbf{y}|\mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x})\right) = \mathcal{N}\left(\mathbf{y}; \mathbf{f}^{\mathbf{M}_1,\mathbf{M}_2,\mathbf{b}}(\mathbf{x}), \tau^{-1}I\right)$ with τ^{-1} observation noise.

We recall the expression 1.8 and its adding term $\mathrm{KL}\left(q_\theta(\boldsymbol{\omega})\|p(\boldsymbol{\omega})\right)$. Since $p(\omega)$ is a prior distribution, we can choose it so the next relation holds:

$$\frac{\partial}{\partial\theta}\mathrm{KL}\left(q_\theta(\omega)\|p(\omega)\right) = \frac{\partial}{\partial\theta}N\tau\left(\lambda_1\left\|\mathrm{M}_1\right\|^2 + \lambda_2\left\|\mathrm{M}_2\right\|^2 + \lambda_3\|\mathrm{b}\|^2\right). \tag{1.12}$$

Then, the derivatives of the expressions 1.8 and 1.10 follow this relation:

$$\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{dropout}}(\theta) = \frac{1}{N\tau}\frac{\partial}{\partial\theta}\widehat{\mathcal{L}}_{\text{MC}}(\theta). \tag{1.13}$$

Consequently, we can optimise over both loss functions obtaining the same results. In the mentioned reference [Gal16], two equivalent optimisation algorithms are also constructed. Summarizing, optimising an artificial neural network, built as explained, with dropout is equivalent to approximately performing estimations over Bayesian neural networks. The main conclusion is that a dropout network constructed as indicated has all the properties that a neural network viewed as a probabilistic model possesses.

1.3.1 Achieving uncertainity measures

We can now present two important results of [Gal16] that allow us to perform inference over a network with dropout.

We first recall the approximate predictive distribution 1.4:

$$q_\theta^*\left(\mathbf{y}^*|\mathbf{x}^*\right) = \int p\left(\mathbf{y}^*|\mathbf{f}^\omega\left(\mathbf{x}^*\right)\right) q_\theta^*(\omega)\mathrm{d}\omega,$$

where $\omega = \{\mathbf{W}_i\}_{i=1}^L$ are the weights as random variables for a network with L layers, $f^\omega\left(x^*\right)$ is the output of the network and $q_0^*(\omega)$ is the optimum for 1.8.

We can estimate the first moment of the output as:

Proposition 1.3.1 *Given* $p\left(\mathbf{y}^*|\mathbf{f}^\omega\left(\mathbf{x}^*\right)\right)=\mathcal{N}\left(\mathbf{y}^*;\mathbf{f}^\omega\left(\mathbf{x}^*\right),\tau^{-1}\mathbf{I}\right)$ *for some* $\tau>0, \mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}\left[\mathbf{y}^*\right]$ *can be estimated with the unbiased estimator*

$$\widetilde{\mathbb{E}}\left[\mathbf{y}^*\right]:=\frac{1}{T}\sum_{t=1}^{T}\mathbf{f}^{\widehat{\omega}_t}\left(\mathbf{x}^*\right)\underset{T\to\infty}{\longrightarrow}\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}\left[\mathbf{y}^*\right] \tag{1.14}$$

with $\widehat{\omega}_t\sim q_\theta^*(\omega)$.

Proof:

$$\begin{aligned}\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}\left[\mathbf{y}^*\right]&=\int\mathbf{y}^*q_\theta^*\left(\mathbf{y}^*|\mathbf{x}^*\right)\mathrm{d}\mathbf{y}^*\\&=\iint\mathbf{y}^*\mathcal{N}\left(\mathbf{y}^*;\mathbf{f}^\omega\left(\mathbf{x}^*\right),\tau^{-1}\mathbf{I}\right)q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\mathrm{d}\mathbf{y}^*\\&=\int\left(\int\mathbf{y}^*\mathcal{N}\left(\mathbf{y}^*;\mathbf{f}^\omega\left(\mathbf{x}^*\right),\tau^{-1}\mathbf{I}\right)\mathrm{d}\mathbf{y}^*\right)q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\\&=\int\mathbf{f}^\omega\left(\mathbf{x}^*\right)q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\end{aligned}$$

giving the unbiased estimator $\widetilde{\mathbb{E}}\left[\mathbf{y}^*\right]:=\frac{1}{T}\sum_{t=1}^{T}\mathbf{f}^{\widehat{\omega}_t}\left(\mathbf{x}^*\right)$ following Monte Carlo integration techniques as in 1.8, this time with T samples.

□

We can also estimate the second moment.

Proposition 1.3.2 *Given can be* $p\left(\mathbf{y}^*|\mathbf{f}^\omega\left(\mathbf{x}^*\right)\right)=\mathcal{N}\left(\mathbf{y}^*;\mathbf{f}^\omega\left(\mathbf{x}^*\right),\tau^{-1}\mathbf{I}\right)$ *for some* $\tau>0$, $\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}\left[\left(\mathbf{y}^*\right)^T\left(\mathbf{y}^*\right)\right]$ *estimated with the unbiased estimator*

$$\widetilde{\mathbb{E}}\left[\left(\mathbf{y}^*\right)^T\left(\mathbf{y}^*\right)\right]:=\tau^{-1}\mathbf{I}+\frac{1}{T}\sum_{t=1}^{T}\mathbf{f}^{\widehat{\omega}_t}\left(\mathbf{x}^*\right)^T\mathbf{f}^{\widehat{\omega}_t}\left(\mathbf{x}^*\right)\underset{\bar{T}\to\infty}{\longrightarrow}\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}\left[\left(\mathbf{y}^*\right)^T\left(\mathbf{y}^*\right)\right] \tag{1.15}$$

with $\widehat{\boldsymbol{\omega}}_t\sim q_\theta^*(\boldsymbol{\omega})$ *and* $\mathbf{y}^*,\mathbf{f}^{\widehat{\omega}_t}\left(\mathbf{x}^*\right)$ *row vectors (thus the sum is over the outer-products).*

PROOF:

$$\begin{aligned}\mathbb{E}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right] &= \int\left(\int(\mathbf{y}^*)^T(\mathbf{y}^*)\,p(\mathbf{y}^*|\mathbf{x}^*,\omega)\,\mathrm{d}\mathbf{y}^*\right)q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\\ &= \int\left(\mathrm{Cov}_{p(\mathbf{y}^*|\mathbf{x}^*,\omega)}[\mathbf{y}^*] + \mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*,\omega)}[\mathbf{y}^*]^T\,\mathbb{E}_{p(\mathbf{y}^*|\mathbf{x}^*,\omega)}[\mathbf{y}^*]\right)q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\\ &= \int\left(\tau^{-1}\mathbf{I} + \mathbf{f}^\omega(\mathbf{x}^*)^T\mathbf{f}^\omega(\mathbf{x}^*)\right)q_\theta^*(\boldsymbol{\omega})\mathrm{d}\boldsymbol{\omega}\end{aligned}$$

giving the unbiased estimator $\widetilde{\mathbb{E}}\left[(\mathbf{y}^*)^T(\mathbf{y}^*)\right] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^T \mathbf{f}^{\hat{\omega}_t}(\mathbf{x}^*)^T\mathbf{f}^{\hat{\omega}_t}(\mathbf{x}^*)$ following again MC integration with T samples. □

Having the second moment we can obtain the variance of the prediction as:

$$\widetilde{\mathrm{Var}}[\mathbf{y}^*] := \tau^{-1}\mathbf{I} + \frac{1}{T}\sum_{t=1}^T \mathbf{f}^{\hat{\boldsymbol{\omega}}t}(\mathbf{x}^*)^T\mathbf{f}^{\hat{\omega}_t}(\mathbf{x}^*) - \widetilde{\mathbb{E}}[\mathbf{y}^*]^T\widetilde{\mathbb{E}}[\mathbf{y}^*] \underset{T\to\infty}{\to} \mathrm{Var}_{q_\theta^*(\mathbf{y}^*|\mathbf{x}^*)}[\mathbf{y}^*] \tag{1.16}$$

this is the sum of the sample variance of T stochastic predictions with dropout and the inverse model precision. This precision is obtained with the following relation:

$$\tau = \frac{(1-p)l^2}{2N\lambda} \tag{1.17}$$

where p is for the probability of shutting down data in the dropout process, l^2 is the variance we set for the prior distribution of the model's weights, N is the number of iterations and λ is the regularization parameter.

With these two results we can estimate the expected predictions and their variance for the multilayer perceptron with dropout applied. There's no need to use Bayesian neural networks to obtain uncertainty measures, which would led us to intractable operations. The alternative of using stochastic regularisation techniques, dropout in this case, offers an approachable solution.

Dataset

We apply the explained ideas to a CMS dataset. CMS stands for *Compact Muon Solenoid*, a particle physics experiment located in the LHC (Large Hadron Collider) at the European Centre for Particle Physics Research (CERN). The LHC accelerates protons which end up colliding head-on at high energy (close to the speed of light) to study the processes that arise from these interactions. CMS is a detector that collects the data produced in these experiments.

The dataset that is going to be used contains information about the particles produced in the proton collisions (events) for both a concrete type of reaction, that is of interest (signal), and other types of processes that we consider as background. Background processes occur with a much higher probability than signal events. The goal is to perform binary classification in order to discriminate the events of interest from the rest, efficiently suppressing the large background while losing as little signal events as possible. More information can be found in [SC14].

The signal process of interest consists in the production of a top-antitop quark pair ($t\bar{t}$) followed by the immediate decay of the top quarks (after only $5 \cdot 10^{-25}$ s). The top quark decays into a bottom quark and a W boson. In the process under study, one of the W bosons decays immediately ($3 \cdot 10^{-25}$ s) into two quarks, and the other decays into one lepton (a muon, specifically) and a neutrino (it escapes undetected).

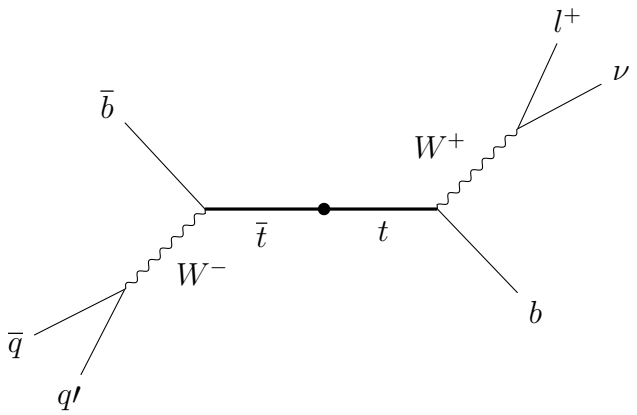


Figure 2.1: $t\bar{t}$ (semi-leptonic) decay diagram.

The detector only *sees* the remains of the process, the lepton and the four quarks[1]. Identifying the whole process from its final state particles can be challenging because there are several other reactions that have similar final states. We refer to these other processes as background and to the decay of interest ($t\bar{t}$) as signal.

We aim to suppress the background as much as we can and retain only the data related to the signal. The detector collects kinematical information of the events such as number of jets, jet momentum, bottom-quark-tagging discriminators, number of muons and others. These variables are used to discriminate the signal from the background events.

The data consists of a series of simulated observations (events) that gather the variables measured by the detector in a collision of protons. They are displayed in a matrix where each row represents one event and each column is a measured physical magnitude. The process that corresponds to each event is known so we can label the observations correspondingly. In total there are 240k simulated observations. In order to get a realistic representation of the relative proportions of the events as they occur in nature, a weight[2] for each observation is provided. This way we can represent the results weighted, which is important to appropriately combine the various components of the background and to compare the right proportions of signal and background.

Type	**Number of events**	**Weighed amount**
$t\bar{t}$ (signal)	2850	635.10
W plus jets (background)	1097337	209603.60
Drell-Yan (background)	77729	34115.51
WW (background)	4580	229.95
WZ (background)	3367	69.93
ZZ (background)	2421	16.92
Single top (background)	5684	311.62

Table 2.1: Dataset breakdown

We describe below the features of each event. In addition, some new features are added performing operations on the existing features. For each observation, we take into account up to a maximum of 6 jets and 3 muons measured by the detector. Quarks are confined inside hadrons by the nuclear strong force, and therefore after being produced turn instantly into a spray of particles called jets. Muons are leptons (fermions that do not undergo strong interaction) and their presence provides us with valuable information about the parent process.

In relation to the jets, we consider their linear momentum (projected in the transverse plane of the collision direction, p_T): *jpt1, jpt2, jpt3, jpt4, jpt5, jpt6*. When the number of detected jets is smaller than six, the corresponding variables are set to 0. We also consider the jets' pseudorapidity (η), *jeta1, jeta2, jeta3, jeta4, jeta5, jeta6*, an angular feature related

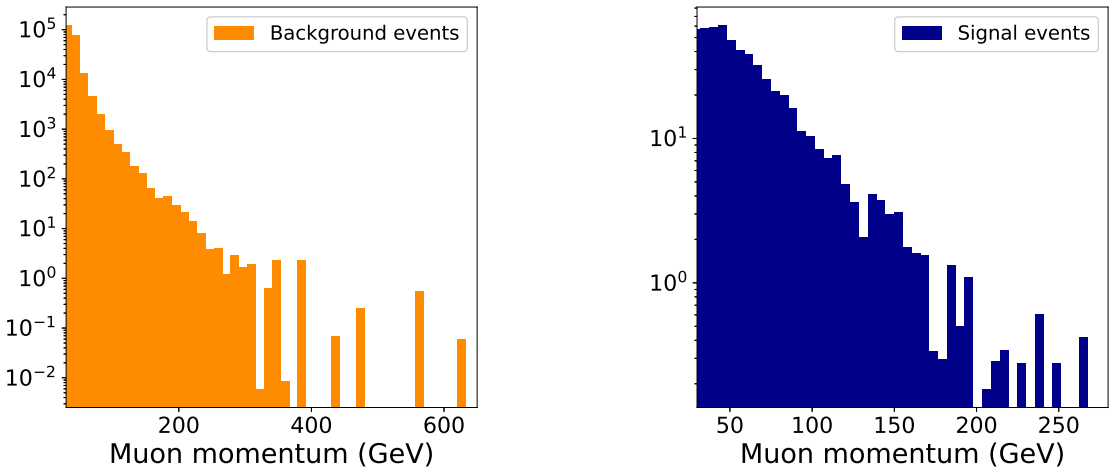
[1]The quarks turn into a collimated spray of hadron particles called a jet.

[2]An event that appears in the set may not occur in nature with the frequency represented in the data set, to correct that we use the weight.

to the polar angle of the originating quark with respect to the collision direction. Since the process of the signal involves two bottom (b) quarks, being able to know whether the jets correspond to bottom quarks or not is highly informative. A b-tagging variable for the jets is provided which is obtained from an algorithm that identifies B-hadron decays in a jet. If the b-tagging score is high it means that there is a high probability that the jet relates to a bottom quark. This way we can discriminate between jets associated with bottom quarks and other types. The corresponding b-tagging features are named *jbtag1, jbtag2, jbtag3, jbtag4, jbtag5, jbtag6*.

To characterize the muons we also consider their transverse momentum *mupt1, mupt2, mupt3* and their pseudorapidity *mueta1, mueta2, mueta3*. It is of interest to know if a detected muon is located within the cone of particles of a jet or if it comes from the decay of a W boson, in which case it is detected isolated. To quantify this, an isolation feature accompanies every muon. It meausures the activity of the detector around the direction of the muon. If the value is large it means that the muon is not isolated, it comes from a jet, whereas if it is small the muon is detected isolated, it comes from a boson decay. These features are named *muiso1, muiso2, muiso3*.

From the original dataset we also take a feature that indirectly measures the presence of a neutrino in the event. Neutrinos hardly interact with matter and escape the detector unseen. However, the transverse momentum of the neutrino can be inferred from the total momentum of all the other particles originated in the collision by means of transverse linear momentum conservation. The corresponding feature is called *met*.



(a) Background events

(b) Signal events

Figure 2.2: Momentum distribution for the first muon

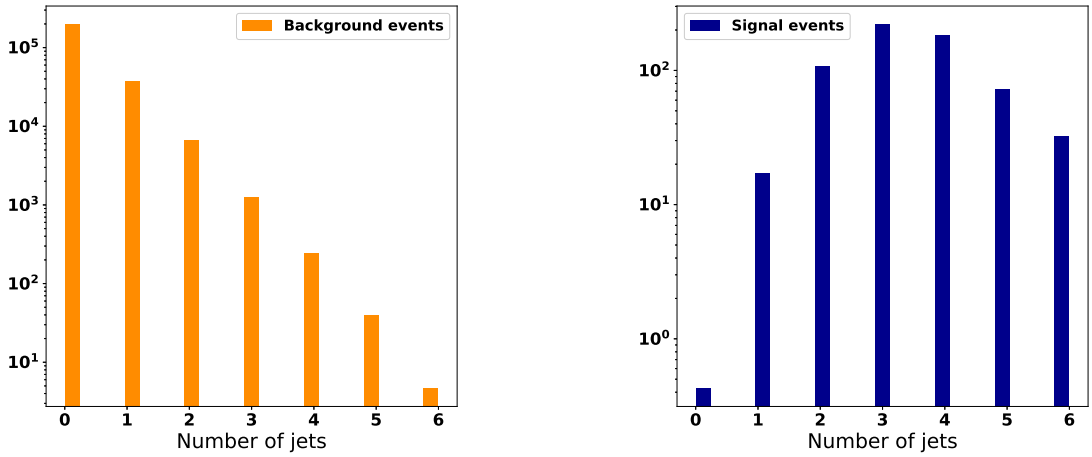
Since the variables are simulated, there are no missing values. Furthermore, the simulated dataset has been constructed carefully pre-selecting clean samples of every process type. Nevertheless, we inspected the distributions of the features to look for anomalies. For example, for the momentum of the first muon *mupt1* there are no strange cases. We

can observe some higher values, (see Figures 2.2a and 2.2b), but they are on an acceptable range and provide necessary information so they are not considered outliers. The rest of the features present the same behaviour, we did not detect outliers in any of them.

In addition to these features, that are included without modification from the original dataset, some others are calculated. For each muon, the relative isolation is obtained as the quotient between the isolation feature and the momentum, $reliso1 = muiso1/mupt1$; *reliso2, reliso3* are defined likewise. The number of muons (*nmuons*) is calculated having into account the non-zero momenta. We also obtain the number of isolated muons, *nisomuons*.

In order to characterize jet detections, we count the number of jets for each observation (*njets*). Besides, we count the number of b-tagged jets, *nbtag*.

Some of these features highlight the differences between the events' nature. For example, we can visualize the number of jets detected for background events (Fig. 2.3a) versus signal events (Fig. 2.3b). The differences are remarkable.

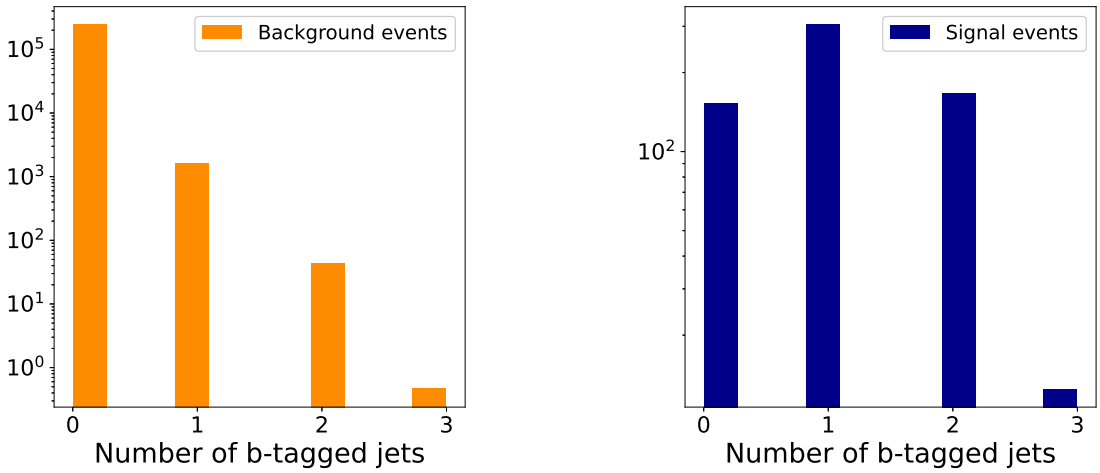


(a) Background events

(b) Signal events

Figure 2.3: Number of jets detected

Another interesting feature to note is the number of b-tagged jets, *nbtag*, displayed in Fig. 2.4. Again, the distributions for signal events and background are clearly different. While the background events have mostly no b-tagged jets (the count for 0 b-tagged jets is two orders of magnitude larger than the next one), in the case of signal events it is more uniformly distributed between 0, 1 or 2 b-tagged jets. The expected value for this feature in the signal case is 2, according to the Fig. 2.1. The differences between the obtained distribution and the expected one are due to the fact that some jets are not tagged properly or detected at all. The b-tagging algorithm is not perfect and some times misses some true b-jets or even wrongly tags non-b jets (those in the count 3 in Fig. 2.4b).



(a) Background events

(b) Signal events

Figure 2.4: Number of b-jets detected

We have in total 35 features that describe the events, plus the event weight which is not used for the network training, but for testing the trained model with a dataset whose composition reflects reality. We label the data, creating the target variable for the neural network, the signal events take the value 1 and the background events the value 0. Just for informative purposes, another variable that stores the specific process of each event (WW, W plus jets, single top ...) is created. It allows us to check the performance of the model for the various components of the background separately.

Summarizing, the dataset consists of 35 input variables, 1 target variable and 2 identifying variables, not used for training nor predicting.

Results

Results from applying the concepts outlined in chapter 1 to the dataset described in chapter 2 are presented in this chapter. A neural network is used to predict the target variable, whether an event originates from a $t\bar{t}$ process or not. We rely on the results of Prop. 1.3.1 and Prop. 1.3.2 to offer expected predictions and their variance, not only point estimates.

To implement the process, Google colab notebooks are used. The library *TensorFlow* [AAB+15], and more concretely, the module *keras* are the tools used for the network building and training. The code developed is available in https://github.com/juliavazquez3/github-upload.git.

3.1 Procedure

We split the data in a training set and a test set. 30% of observations are reserved for testing the models and the rest is used for training them. We sample the 30% randomly to avoid any bias.

The training set of observations has a proportion of only 3.5%[1] of signal events. This can entail a bad training because the network works in batches of observations, so in many of them there might not appear any signal events. If this happens frequently, the network's weights are updated so that the identification of signal events is disfavoured.

Better results are obtained if the training set is undersampled. Background events are eliminated randomly to obtain a set consisting in 40% signal events and 60% background. Training the network with the undersampled set avoids the aforementioned problem. The predictions obtained are more reliable this way.

The final sets are used for training and testing the networks. A multilayer perceptron with dropout is built, consisting of dense layers with dropout applied before them. First a dropout layer, then the first hidden layer, which is dense, then dropout is performed again, another hidden one, dropout, the last hidden layer and the output layer. As explained before in chapter 1, the dense layer consists of a series of neurons that are fully connected to the neurons of the next layer. By adding dropout, we are randomly shutting down some neurons. The simplified scheme depicted in Fig. 3.2 only represents the layers arrangement. In the

[1] This proportion is higher than the natural one. This is because these events are not weighed to the real proportions.

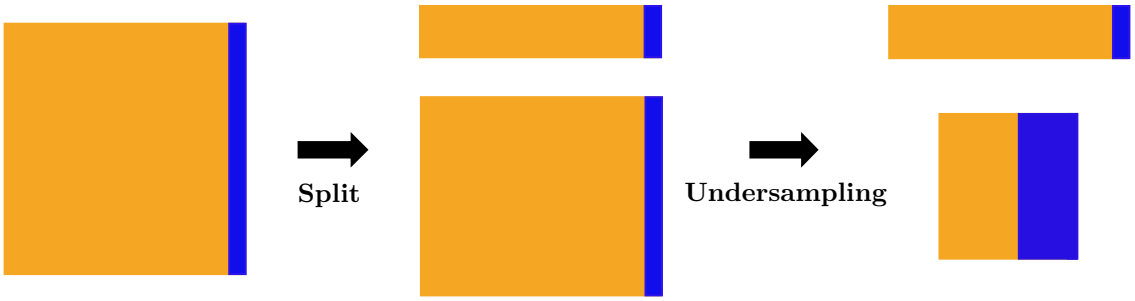


Figure 3.1: Diagram displaying the data processing

actual network, the first dense layer has 64 neurons, the second has 32 neurons and the last one 16. The choice for the number of dense layers and their neurons is due to previous experience in building neural networks for binary classification. Additionally, variations of these parameters were tested showing that this combination offers the best results.

The number of epochs chosen for the network is 50. It is small enough so that we do not end up with an overfitted model and a larger number would not improve the fit much, just would increase the running time. The batch size is set to 128. The activation function used for the neurons in the hidden layers is the rectified linear unit ($ReLU$ [NH10]), compared with other activaction fuctions it is more computationally efficient. The activation function used for the neuron in the output layer was a sigmoid [Mit97], appropriate for binary classification.

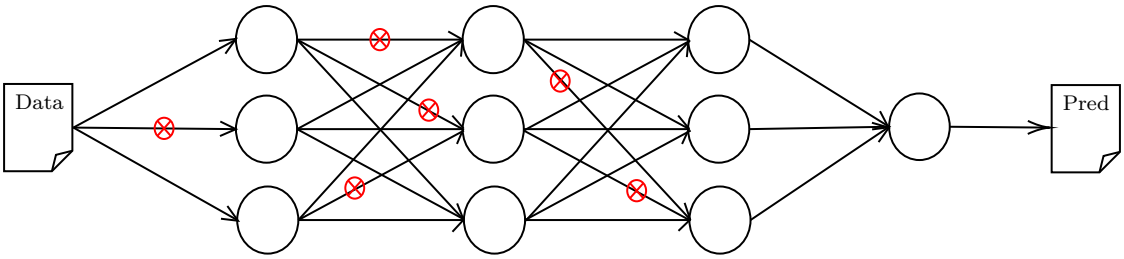


Figure 3.2: Scheme of the network architecture

To train the network, a binary cross-entropy loss function was chosen, whose expression is:

$$\text{Loss} = \sum_i -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)), \tag{3.1}$$

where y_i is the true label for the event i and $\hat{y}_i$ is the prediction provided by the network for that event. We also specify a regularization parameter λ in each dense layer so that the computed error takes into account the complexity of the model, as in eq 1.10, where the norm of the weights plays a role in the loss function. These terms are added to the expression 3.1 to form the complete loss function. This regularization parameter is chosen $\lambda = 0.001$. If we set a higher regularization parameter, the predictions of the model significantly decrease in goodness of fit, and with this quantity we don't have overfitting problems.

An algorithm to implement the network training is necessary. The module of *TensorFlow*, *keras*, offers a variety of them. We choose an *Adam* optimizer, details of how it works can be found in [KB15].

The probability of shutting down a neuron when applying dropout is fixed to 0.1. For the purpose of evaluating the robustness of the procedure, alternative dropout parameters were tested. Instead of 0.1, the values 0.15 and 0.2 were set as dropout probability. There were no remarkable changes in the results. The observed variation does not follow a tendency against the parameter and it is within a small range. Therefore, only results regarding to 0.1 as dropout parameter are presented.

In order to estimate the uncertainty of the predictions, we obtain 10 different predictions by running the network training from scratch 10 times.

Since 10 different models are trained, we obtain 10 different predictions for the test dataset. Following the result of Prop. 1.3.1, we calculate an expected prediction by averaging the T predictions $\overline{y} = \frac{1}{T}\sum_r \hat{y}_r$ [2]. In addition, We used Prop. 1.3.2 to obtain a variance as given in eq. 1.16. We add τ^{-1} as defined in eq. 1.17 to the variance estimator $S^2 = \frac{1}{T}\sum_r (\hat{y}_r - \overline{y})^2$.

3.2 Training performance

Accuracy and loss metrics are measured for both training and test datasets[3]. The values of these metrics are stored for each epoch of the training of each network. These metrics are of interest for evaluating the scope of the resulting models. Since these values are similar for every training (we train 10 models), and we only want to study the process qualitatively, only the loss and accuracy evolution of one of the models is represented.

It can be seen in Fig. 3.3 that the accuracy increases slowly and smoothly with the training process. Precisely, the model progresses in such a way that it fits the training events better and better. The test data has a more noisy plot but it also seems to have an increasing tendency.

The model does not take into account the test events when evolving, hence the accuracy the model provides for these observations is noisier. The accuracy still increases with the epochs for the test set which means that the model is not overfitted, thus it will generalize nicely.

In the loss plot (Fig. 3.3), for the train data, it can be seen how it descends, first with a high slope, then the decrease slows down. Again, the behaviour for the test data is more variable but it also has a decreasing tendency. The loss for the train data is larger than the test loss, which indicates that the model is not overfitted.

[2] We denote the prediction corresponding to the model r as $\hat{y}_r$ for $r = 1, ..., T$.

[3] Even though the accuracy and loss are measured for the test events, for every epoch of the training, these events are not used in the weights updates, the test events do not influence the model outcome.

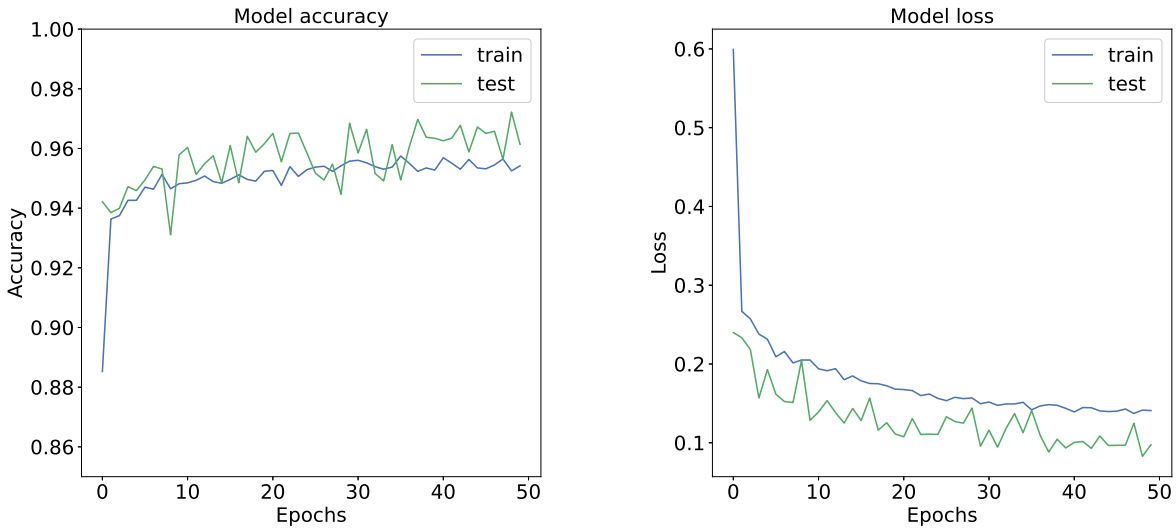


Figure 3.3: Accurary and Loss metrics for one of the models.

It could be expected to have a correspondence between the accuracy and the loss in the form of symmetric plots but we must remember that the loss includes regularization terms that the accuracy measurements do not.

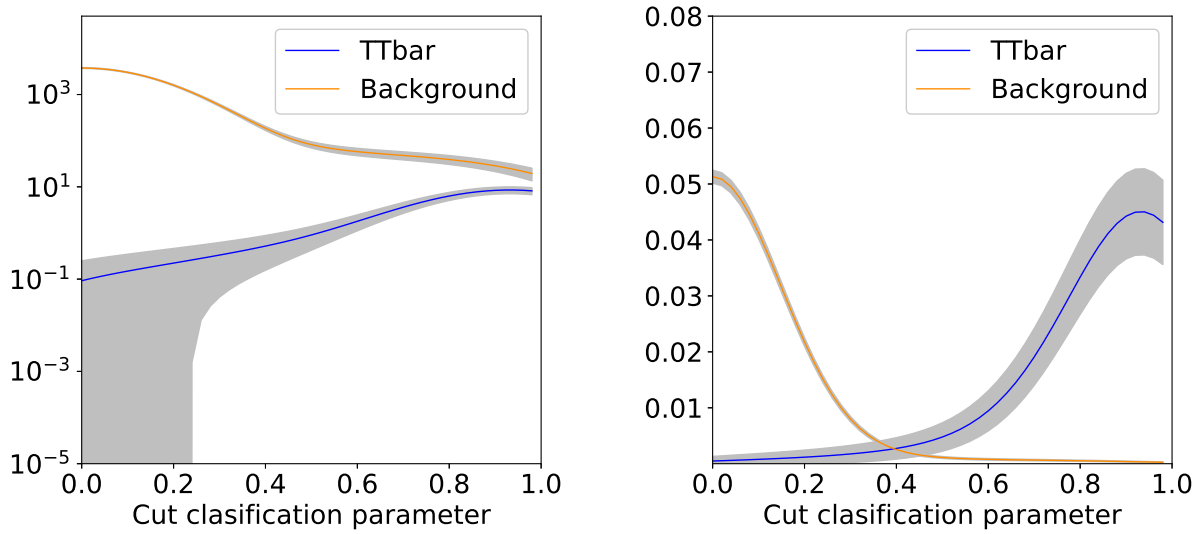
For both the accuracy and the loss metrics of the training data, there is a decrease in the slope of the tendency towards the end. The plot stabilizes as the number of epochs augments. We could increase the number of epochs in the network training, since there are not overfitting problems, but the small improvement in goodness of fit would not compensate the computational cost.

3.3 Classification performance

Using the test dataset, we present in this section the evaluation of the classification performance of the network. The various metrics calculated below are presented together with the estimation of their uncertainty. This way, the precision of the results can be assessed.

3.3.1 Classification parameter distribution

We show in Fig 3.4[4] the distribution of the predicted values of the classification parameter together with its estimated variance, for both the signal and the background processes. We describe in appendix A the detailed technical procedure to produce the uncertainty bands. The distribution for the background has been constructed using the event weights that allow



(a) Events weighed to reality

(b) Normalized distributions

Figure 3.4: Distribution of the predicted classification parameter for the signal and the background test datasets separately. The left figure reflects the relative normalization between signal and background as occurring in nature (note the log scale), while the right figure shows the probability distribution functions (normalized to area 1) of the classification parameter.

to mix the various processes contributing to the background according to their proportions in nature.

Fig. 3.4 shows how the background events tend to have lower score for the classification parameter while for signal events higher scores are more probable (closer to 1). In Fig. 3.4a signal and background have been weighted so that their relative proportion is the one observed in nature. We can therefore view the figure as how the model would operate for a real set of events. Background events are much more copious, but for values of the classification parameter close to 1, the number of signal and background events tend to even out. The y axis has a logarithmic scale so that the signal events are visible.

In Fig. 3.4b the distributions have been normalized to unity area so that they represent the probability distribution functions of the classification parameter. It can be better appreciated the distinct shape of the distributions for both types of events. The background tends to have a prediction closer to 0 and the signal closer to 1, creating a valley in between. It is qualitatively perceptible that the network is able to discriminate very well between signal and background.

Regarding the variance band, the variance of the signal distribution is larger. The fact that the test data set contains a large proportion of background compared to signal events may contribute to this. The predictions delivered by our model are more erratic for the signal case, but still, within the variance limits, the signal is clearly differentiated from the background.

[4]The label *TTbar* in the figure corresponds to the decay $t\bar{t}$ considered signal as explained.

3.3.2 True and false positive rates

We consider the True Positive Rate (TPR) and False Positive Rate (FPR) metrics to evaluate the quality of the predictions of the model in the test dataset[5]. TPR is the ratio between the number of signal events predicted as signal over all tested signal events. FPR is the ratio between the number of background events predicted as signal over all tested background events.

$$\text{TPR} = \frac{\text{TP}}{\text{CP}} \quad \text{FPR} = \frac{\text{FP}}{\text{CN}}^{6} \tag{3.2}$$

Since 10 models were trained, we have obtained the expected average of the TPR and FPR predictions and their variance, instead of point estimations. Calculation details are provided in the appendix B.

Although the true label of the events is either 0 or 1, we have chosen the classification parameter to be a continuous real number in the range $[0, 1]$. This allows us to study TPR and FPR as a function of the value of the classification parameter, taken as a threshold to select a sample of signal and background events. The goal is to enrich the signal+background sample in signal events, keeping the level of background as low as possible.

It is usual to define the so-called tight, medium and loose *working points* that correspond to FPR values (background selection efficiency) of 0.1%, 1% and 10% respectively. Typically, tighter working points result in an increased ratio of signal over background but at the price of decreasing the signal selection efficiency (TPR). The choice of the working point depends on the details of the physics analysis of the signal process. In some cases a high signal purity is needed (tight working point) and in other cases a large number of signal events is required (loose working point).

We represent in Fig. 3.5 TPR and 1-FPR as a function of the value of the predicted classification parameter. This value is used as a threshold to predict an event in the test dataset either as signal (classification parameter value larger than the threshold) or background (otherwise). TPR is built comparing the prediction with the test events tagged as signal, while FPR is built using the test events tagged as background.

The TPR and 1-FPR curves correspond to the average values obtained with 10 trainings of the network. The band around the curves represents the estimated uncertainties (variances). Uncertainty bands are very small, so the predictions of the model are expected to be quite stable.

The form of the TPR and 1-FPR curves show that the model discriminates very well between signal and background events. The ideal situation would be to have a very small slope all throughout the plot and then a drastic drop to 0 near to small values of the cut

[5] In the context of binary classification we label signal events as 1 (positive events) and background as 0 (negative events).

[6] We denote as TP the True Positives predicted and FP the False Positives. CP stands for Condition Positives and CN for Condition Negatives.

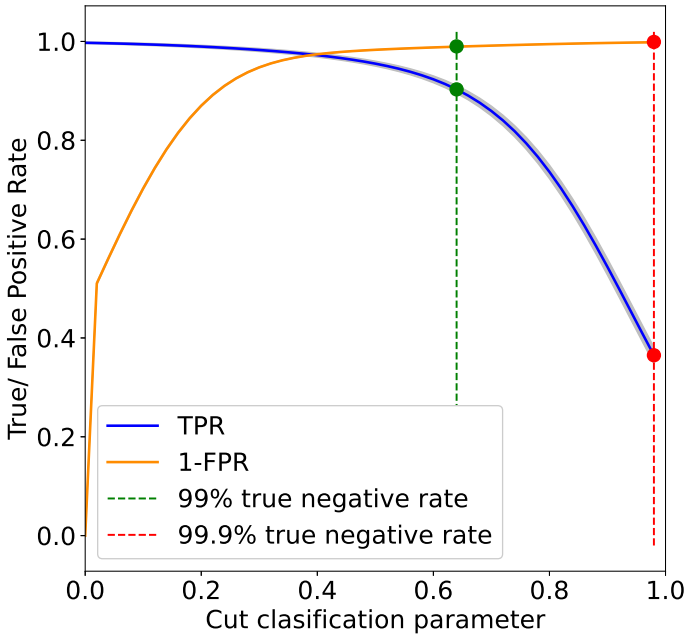


Figure 3.5: TPR and 1-FPR plots

classification parameter for the 1-FPR curve, and near to 1 for the TPR curve. The actual situation is not far from the ideal case.

A good indicator of the classification performance is the point where both curves meet. The closer to 1 the better the performance. In our case, the crossing point is about 0.97, which is a great result. It means that for the corresponding value of the classification parameter cut, we can correctly predict 97% of true signal and background events.

For a sufficiently low value of the classification parameter threshold (0.64), a large background suppression can be achieved with a high signal selection efficiency. A 99% background suppression (FPR=0.01 or 1-FPR=0.99) corresponds to an average signal efficiency equal to 90.3% (TPR in the range $[0.895, 0.911]$ considering the uncertainty). This is the medium working point referred above. A tighter background suppression (99.9% or 1-FPR=0.999) yields an average signal selection efficiency of 36.5% (TPR in the range of $[0.348, 0.382]$ taking into account the uncertainty). This tight working point is set with a threshold of the classification parameter of 0.98. Both working points are represented in Fig. 3.5 as vertical lines crossing the TPR and 1-FPR curves.

When choosing the tight working point, a fair fraction of the signal events (about one third) is kept while the background is reduced at the per mill level! If a less stringent reduction of the background level is adequate for the subsequent physics analysis of the data (at the percent level with the medium working point), we can keep about 90% of the signal.

In addition to the analysis of the performance achieved with the network in the identification of signal (TPR) and background (FPR) events, it is of interest to calculate the

relative proportion of signal and background events expected in real data for the various working points (defined by cuts in the classification parameter). Usually, the background is orders of magnitude larger than the signal, so even with small FPR values and large TPR, the proportion of the signal might still be small to be seen compared to the remaining background.

The test data set, weighted to fit the real proportions of the various processes in nature, consists of 73k observations, where only 0.26% of them are signal events. The proportion of signal events increases as the working point gets tighter but at the price of discarding true signal events. For the working point defined by TPR= 0.97 (1-FPR = 0.97), the set of test events classified as signal consists of 7.7% of the total condition signal events. This represents an increase of 30 in the proportion of signal events with the small penalty of losing 3% of them.

For the medium working point (1-FPR= 0.99, TPR= 0.90 ± 1, considering the TPR variance) the proportion of signal events increases to $18.9 \pm 0.1\%$. For the tight working point (1-FPR= 0.999, TPR= 0.365 ± 0.015), the proportion goes to of 48.6 $\pm$1.1%%, so the events data selected in this way would contain approximately the same number of signal and background events.

As seen, the continuous classification parameter offers a number of possibilities to select the final data sample, that varies in the proportion of signal to background events and in the number of signal events. Depending on the particular physics analysis of data data, a loose working point can be chosen (when it is important to keep a large number of signal events) or a tighter working point when the purity of the sample if signal events is important.

Anyhow, as seen above, the neural network is able to suppress a large fraction of the background while keeping most of the signal. We emphasize the discrimination power of our neural network. Furthermore, the estimation of the uncertainty on the performance of the network allows to calibrate the reliability of the result.

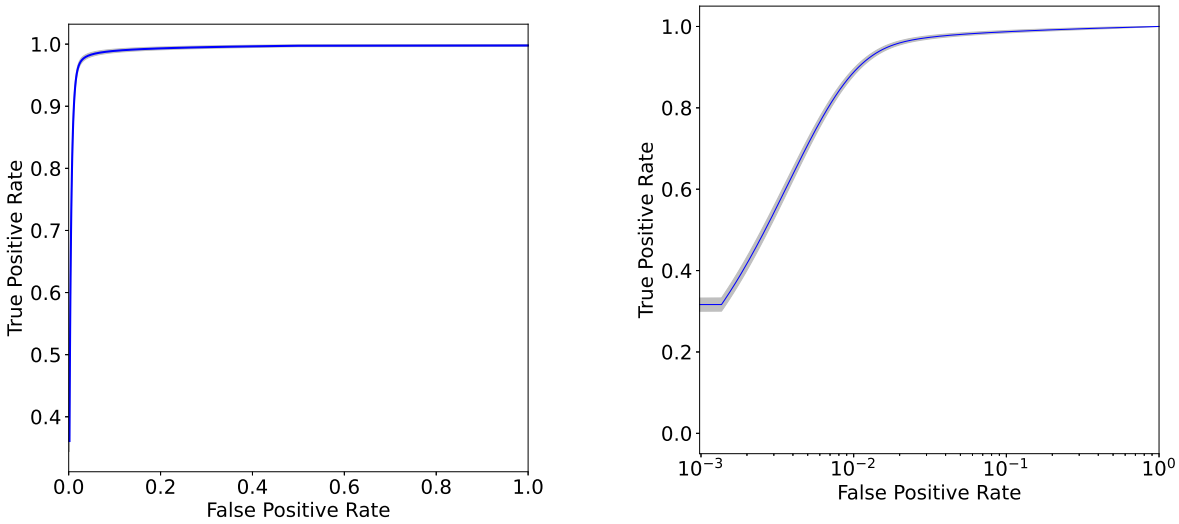
3.3.3 The ROC curve

Another way of presenting the results is through the so-called Receiver Operation Characteristic (ROC) curve. It is a graphic representation where the TPR values are plotted against the FPR ones. Fig. 3.6 shows the results applying our model to the test data.

Both plots, 3.6a and 3.6b, correspond to the same ROC curve, with the only difference being that in Fig. 3.6b the abscissa axis is presented in logarithmic scale so that very small values of FPR can be better appreciated.

The shaded area in the plots correspond to the TPR variance. The FPR variance is so small it would not be noticeable in the graphics. The TPR uncertainty is also quite small and only noticeable in the logarithmic scale plot.

A helpful measure is the area under this curve, *auc*. This is interpreted as the probability of the model predicting a higher value for a random signal event than for a random



(a) ROC curve in linear scale.

(b) ROC curve with the x axis in logarithmic scale.

Figure 3.6: ROC curves

background event (here signal events are labeled as 1 and background as 0). An area of 0.5 would mean that our model classifies as well as a random classifier. The higher the *auc* value the better our model classifies.

The area under the curve, for the central values, is 0.990. We obtain essentially the same value if we use the external curves given by the uncertainty band. This very high value tells us that there is a high probability (close to 1) that our model makes a higher prediction for a random signal event than for a random background event. This allows us to discriminate between both of them.

3.4 Summary and conclusions

The huge volume of data that particle physics experiments have to deal with motivates the problem faced in this project. When trying to produce and detect rare events, huge amounts of observations must be made, most of them of no interest. It has been shown how neural networks can offer an efficient solution for the processing of vast data volumes in the search for tiny signals.

In this study, the rare process considered as signal is the production and decay of a $t\bar{t}$ quark-pair. Any other type of final state resulting from the collisions of two protons is considered as background. A dataset of the CMS experiment at CERN, consisting on simulations of proton-proton collisions is used.

Applying binary classification by means of a neural network, allows us to efficiently suppress most of the background maintaining a large proportion of the signal. Results are presented with uncertainty measures, following the approach in [Gal16]. Expected values together with their variance are provided, not only point estimations. Therefore, the results provide reliable information about the network scope. This method to estimate the performance uncertainty of a neural network applied to particle physics data is innovative.

A continuous classifier (between 0 and 1) is constructed. Background events tend to score low values of the classification parameter, while signal events concentrate at high values. Selecting events above certain threshold of the classification parameter allows to select samples of events with different signal to background proportions. For a 99% background suppression we obtain a $90 \pm 1\%$ signal selection efficiency. For a more severe background suppression requirement of 99.9%, the signal efficiency achieved by our model is $36.5 \pm 1.5\%$, still keeping a fair proportion of signal events.

In conclusion, the neural network built is this work is able to very efficiently discriminate between signal and background events. This allows to apply the model to real data to select a sample enriched in signal events that can be used to study the physical properties of the top quark. Our approach additionally provides the estimation of the uncertainty in the performance of the model. This is key in the measurement of any physical process.

This project opens the possibility of applying machine learning techniques to the classification of processes of interest in collision data collected by particle physics experiments.

Appendices

A: Distribution obtention

We want to present the counts of events against their predicted target with uncertainty measures. We do this by displaying expected bin values and their variance. We suppose that the prediction of an event follows a normal distribution, the mean of the distribution is the average prediction previously calculated and the standard deviation the square root of the prediction variance. We assume, then, $\hat{y}_i \sim N(\overline{y}_i, \mathrm{Var}_i)$ where $i = 1, ..., S$, having S events to be labeled.

We divide the space of prediction, which is $[0, 1]$, into a desired number of bins R. The fact that the prediction for an event falls into a bin can be viewed as if it followed a Bernouilli distribution $Be(p_{ij})$, where p_{ij} is the probability of an event e_i to fall into a bin b_j (delimited by $[b_{j1}, b_{j2}]$) calculated as $p_{ij} = P\left[b_{j1} \leq X_i \leq b_{j2}\right]$ given $X_i \sim N(\overline{y}_i, \mathrm{Var}_i)$ $i = 1, ..., S$, $j = 1, ..., R$.

The number of events with predicted value in a bin will follow, therefore, a sum of Bernouilli distributions. We estimate the expected value of each bin and its variance. The weights of each event w_i is taken into account when computing these quantities.

$$\mathbb{E}[b_j] = \sum_{i=1}^{S} p_{ij} w_i \quad \mathrm{Var}(b_j) = \sum_{i=1}^{S} p_{ij}(1 - p_{ij}) w_i^2$$

following the properties $\mathbb{E}\left[aX + bY\right] = a\mathbb{E}\left[X\right] + b\mathbb{E}\left[Y\right]$, where X and Y are random variables and a and b are constants, and $\mathrm{Var}(aX + bY) = a^2 \mathrm{Var}(X) + b^2 \mathrm{Var}(Y)$ if X and Y are independent from each other.

B: Metrics obtention

Similarly, expected values and variance are calculated for TPR and FPR. The probability of an event i to be classified as signal, given a cut classification parameter a is calculated as $p_i(a) = P\left[-\infty \leq X_i \leq a\right]$ given $X_i \sim N(\overline{y}_i, \mathrm{Var}_i)$ $i = 1, ..., S$. If an event is classified as signal or not is modeled as a Bernouilli distribution $Be(p_i(a))$. As a reminder, the TPR value for a cut point a is the ratio between the true signal events predicted as such and all the condition signal events. Therefore, we can model the TPR as a sum of Bernouillis multiplied by a constant. In an equivalent way the FPR, ratio between true background events classified as signal and condition background, is obtained. The expected values and the variance are estimated.

$$\mathbb{E}[\mathrm{TPR}(a)] = \frac{1}{|N_s|} \sum_{i \in N_s} p_i(a) w_i \quad \mathrm{Var}(\mathrm{TPR}(a)) = \frac{1}{|N_s|^2} \sum_{i \in N_s} p_i(a)(1 - p_i(a)) w_i^2$$

$$\mathbb{E}[\mathrm{FPR}(a)] = \frac{1}{|N_b|} \sum_{i \in N_b} p_i(a) w_i \quad \mathrm{Var}(\mathrm{FPR}(a)) = \frac{1}{|N_b|^2} \sum_{i \in N_b} p_i(a)(1 - p_i(a)) w_i^2$$

where N_s is the set of condition signal events and N_b the set of condition background.

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015, Software available from tensorflow.org.

[Cen] CERN Data Centre, *Processing lhc data*, https://home.cern/science/computing/processing-what-record.

[Gal16] Yarin Gal, *Uncertainty in Deep Learning*, Ph.D. thesis, University of Cambridge, 2016.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, Cambridge, MA, USA, 2016, http://www.deeplearningbook.org.

[HvC93] Geoffrey E. Hinton and Drew van Camp, *Keeping the neural networks simple by minimizing the description length of the weights*, 5–13.

[JGJS99] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul, *An introduction to variational methods for graphical models*, Mach. Learn. **37** (1999), no. 2, 183–233.

[KB15] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, CoRR **abs/1412.6980** (2015).

[Kul59] Solomon Kullback, *Information theory and statistics*, Wiley, 1959.

[Mac92] David J. C. MacKay, *A practical bayesian framework for backpropagation networks.*, Neural Computation **4** (1992), no. 3, 448–472.

[Mit97] Thomas M. Mitchell, *Artificial neural networks*, Machine Learning, McGraw-Hill, Inc., 1997, pp. 81–126.

[NH10] Vinod Nair and Geoffrey E. Hinton, *Rectified linear units improve restricted boltzmann machines*, Proceedings of the 27th International Conference on International Conference on Machine Learning (Madison, WI, USA), ICML'10, Omnipress, 2010, p. 807–814.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Learning Representations by Back-propagating Errors*, Nature **323** (1986), no. 6088, 533–536.

[RM51] H. Robbins and S. Monro, *A stochastic approximation method*, Annals of Mathematical Statistics **22** (1951), 400–407.

[Rub81] Donald B. Rubin, *The bayesian bootstrap*, Ann. Statist. **9** (1981), no. 1, 130–134.

[SC14] Schmidt A. Sander C., *CMS data analysis tutorial*, September 2014, http://ippog.org/resources/2012/cms-hep-tutorial.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research **15** (2014), 1929–1958.

[TLS89] Tishby, Levin, and Solla, *Consistent inference of probabilities in layered networks: predictions and generalizations*, 403–409 vol.2.