Machine learning for particle physics

Anja Butter[1,2]

1 Institut für Theoretische Physik, Universität Heidelberg, Germany
2 LPNHE, Sorbonne Université, Université Paris Cité, CNRS/IN2P3, Paris, France

July 21, 2025

Abstract

Contents

1 Introduction

2 Machine learning basics

The goal of machine learning is to learn a complex function to describe some kind of data as good as possible. In that sense physicist have been doing machine learning ever since they fit a linear function to data to described classical mechanics. Nevertheless the combination of huge amounts of available data at the LHC, in cosmology and astrophysics with latest advancement in computing capacaties have made machine learning in fundamental physics an exciting research topic. The following notes will give a very brief introduction into the basics of machine learning. These notes are not meant to be self explanatory but to be used as accompanying material to a lecture.

2.1 Building and training Neural Networks

A neural networks is a parametrized function f_θ that maps input data x to an output y

$$f_\theta : x \to y \qquad \text{with} \quad x \in \mathbb{R}^{D_i}, y \in \mathbb{R}^{D_o} \,, \tag{1}$$

where $D_{i/o}$ refers to the dimension of the input and output. The network parameters are referred to as weights and typically come in big numbers (ChatGPT 4 is estimated to have around $1.8 \cdot 10^12$ parameters). The underlying encoding of neural networks is not arbitrary. They make use of the ability of GPUs to perform highly efficient matrix multiplications. A network is therefore build of multiple layers l, each corresponding to one multiplication of their respective input vector with a weight matrix W. Naively a dense neural network would apply a series of weight matrices to the input data vector.

$$x^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)} \qquad \text{with} \quad 1 \leq l \leq N_{\text{layers}}, \quad x = x^0, \quad y = x^{N_{\text{layers}}} \tag{2}$$

The bias b_l, a trainable weight vector, enables the mapping of the zero vector to a non-zero output and turns the linear functions into affine functions. However, applying a series of affine functions simply yields an affine function.

$$x^{(N)} = W^{(N)} \cdot \left(W^{(N-1)} \cdot \ldots \cdot \left(W^{(2)} \cdot \left(W^{(1)}x^{(0)} + b^{(1)}\right) + b^{(2)}\right) + \ldots + b^{(N-1)}\right) + b^{(N)} \tag{3}$$

$$= \widetilde{W}x^{(0)} + \tilde{b} \tag{4}$$

It is therefore crucial to apply a non-linear function σ, the so-called activation function, to achieve the expressibility that makes neural networks to powerful. The activation function is applied after each affine function. The output of a layer is hence given by

$$x^{(l)} = \sigma^{(l)}(W^{(l)}x^{(l-1)} + b^{(l)}) \qquad \text{with} \quad 1 \leq l \leq N_{\text{layers}}, \quad x = x^0, \quad y = x^{N_{\text{layers}}} \tag{5}$$

Note that so far the indizes have refered to the layer of the network. In order to implement a network it is useful to formulate the above equation in index notation using the usual summation convention. Eq. (6) then reads

$$x_i^{(l)} = \sigma^{(l)}(W_{ij}^{(l)}x_j^{(l-1)} + b_i^{(l)}) \qquad \text{with} \quad 1 \leq l \leq N_{\text{layers}}, \quad x = x^0, \quad y = x^{N_{\text{layers}}} \tag{6}$$

While the full equation describes a layer, for a single index i the expression corresponds to a single node or neuron, the name giver of the neural network.

We have now assembled the minimal ingredients to build a deep neural network. In practice we have to choose three hyper parameters in order to fix the architecture, given the input data: the depth of the network, ie. the number of layers N_{layers}, the width of the network, ie. the number of nodes per layer N_{nodes}, and the type of activation function. While theoretically any non-linear function would be a possible option, there are several well-performing standard choices.

- **ReLU** (Rectified Linear Unit). The ReLU function

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{7}$$

  maps every number to its non-negative component. For non negative values it serves as aan identity mapping while negative values are mapped to zero. This function is very popular due to its low computing costs and the simple form of its derivative. Obviously it should now be used in the final layer if the prediction should include negative values.
- **Sigmoid**. The sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

  maps real numbers to the interval $[0, 1]$. It is particularly usefull as a final layer in classification task where the output of the network corresponds to a probability $p \in [0, 1]$
- GeLu, Leaky ReLU,... Other activation functions can be more suitable depending on the problem. GeLu and leaky ReLU allow for a treatment of negative values. Moreover GeLU is differentiable and smooths for all real numbers.

2.2 The likelihood or How to formulate a loss function

Given the neural network we build in Sec. 2.1 the next step is to train the network so that it performs the desired task. The training of the network is governed by the loss function. The network is trained to minimize the loss function by adjusting its parameters. There are two main approaches to build a lossfunction: Likelihood based or variational inference. In this case we will focus likelihood based approaches. The goal of the training is to maximise the probability of the network parameters θ given the training data x_{train}. While we have no direct access to this information we can relate it to known quantities using Bayes' theorem.

$$p(\theta|x_{\text{train}}) = p(x_{\text{train}}|\theta)\frac{p(\theta)}{p(x_{\text{train}})} \tag{9}$$

We can now consider the three contributions individually:

- $p(x_{\text{train}})$ is called the evidence. Since it is independent of the network parameters. Since we will train the network to minimize the negative logarithm of the likelihood, the evidence simply adds a constant contribution to the training. It can hence be ignored.
- $p(\theta)$ is the prior over the network weights. There are no first principles based arguments for a correct prior. A common choice is to assume a Gaussian prior on θ. Applying the logarithm leads to a contribution proportional θ^2 to the loss term, also known as L_2—regularization. An alternative choice consists in the Bernoulli distribution which leads to the implementation of Dropout, randomly setting weights to zero during the training.
- $p(x_{\text{train}}|\theta)$ is the likelihood of the network parameters. As the only term that connects the data to the network parameters, maximizing the conditional probability of θ is often equivalent to maximizing the likelihood, as long as the regularization is not too severe.

Instead of maximizing the likelihood, it is numerically more stable to minimize the negative loglikelihood. Hence the final loss function will take the following form

$$\mathcal{L} = -\log\left(p(x_{\text{train}}|\theta)\right) - \log\left(p(\theta)\right) \tag{10}$$

Regression Given the derived form of the loss function we have to postulate a functional form of the likelihood. Motivated by the central limit theorem, we assume a Gaussian likelihood.

$$-\log p(x_{\text{train}}|\theta) = \sum_{j\leq n_{\text{train}}} \frac{|y_j - f_\theta(x_j)|^2}{2\sigma^2(x_j)} + \log(\sigma(x_j)) + \log(\sqrt{2\pi}). \tag{11}$$

Classification One of the most common tasks in machine learning is classification, ie. assigning a class label like "top jet", "QCD jet" or "lepton" to data. The output of the network should give the probability of a sample to belong to a certain class. In binary classification, the classification into two classses A and B the loss function is called binary cross-entropy. It is derived from the KL-divergence of the distribution of the two classes. It reads

$$\mathcal{L} = -\sum_{x\sim p_A} \log(f_\theta(x)) - \sum_{x\sim p_B} \log(1-f_\theta(x)) \quad \text{with} \quad 0\leq f_\theta(x)\leq 1 \tag{12}$$

$$= -\int \mathrm{d}x\, p_A(x)\, \log(f_\theta(x)) + p_B(x)\, \log(1-f_\theta(x)) \tag{13}$$

The second line makes the transition towards the continuous limit.

Instead of giving a derivation for the loss let us consider what condition will be fulfilled once we arrive at the minimum. Taking the functional derivative yields

$$\frac{\delta \mathcal{L}}{\delta f_\theta} = 0 = -\int \mathrm{d}x \, \frac{p_A(x)}{f_\theta(x)} + \frac{p_B(x)}{1-f_\theta(x)} \tag{14}$$

$$\Leftrightarrow \qquad \frac{p_A(x)}{p_B(x)} = \frac{f_\theta(x)}{1-f_\theta(x)} \tag{15}$$

Back propagation

$$\frac{d\mathcal{L}}{d\theta} = \frac{d\mathcal{L}}{dy}\frac{dy}{dz^{(n)}}\frac{dz^{(n)}}{d\sigma^{(n-1)}}\frac{d\sigma^{(n-1)}}{dz^{(n-1)}}\frac{dz^(n-1)}{d\sigma^{(n-2)}}\cdots \tag{16}$$

3 How to train better networks

Please check out the slides for details.

Stochastic gradient descent

Training, validation and test data

The learning rate and optimizers

Regularization

Initialization

Data representation

References