Topics in Particle Physics and Astroparticles II

Guilherme Soares

Laboratory of Instrumentation and Experimental Particle Physics

24 March 2023

Index

Machine Learning Template

The relevant template can be found under Template_ML.ipynb, under the usual github page.

We will be utilizing a different SHiP dataset than the one utilized for Template_Data_Analysis.ipynb, which is named kinematic$_v$*ariables*$_B$*ack*.*datandkinematic*$_v$*ariables*$_H$*NLtoPimu*.*datundertheDatadirectory*.

The first thing to do in ML is to define the datasets that we want to use when training the Neural Networks.

Machine Learning Template - Training and Validation Samples

Our **datasets should always be normalized**. While we are able to identify that a mass of 1 GeV is relatively big but a total momentum of 1 GeV is rather small, the algorithm cannot not do that. So we create a transformation pipeline, which is an object that transforms all data applied to it.

In our case we can to perform 2 transformations : **Standard Scaler** and **Principal Component Analysis**. The first re-scales all features to have a mean of 0 and a standard deviation of 1 ; the second performs a basis rotation to the features in order to leave them as independent of each other as possible, while maximizing each the variance within each new feature.

While the second transformation might not be intuitive to us, it is effective and the algorithm does not understand units or physical features, but is vastly better than us at performing multi-dimensional analysis regardless of the data format.

Machine Learning Tutorial - Training and Validation Samples

Now we need to **define** what **features** we will give as **input to the NN**.

We should not be lazy and feed everything we have to the NN. Some features are highly correlated, and others provide no discernible difference between signal and background.

Choose wisely as providing too much data in the best of cases increases the necessary computing power requirements, and in the worst case is like pilling extra straws on a haystack when we are trying to find a needle, "confusing" our algorithm.

After we selected the variables to use, we feed the pipeline our **full dataset** (signal+background), as this allows the **normalization** to **always** be **the same**.

Once we normalize the features that we want, we then separate the data back into the Signal and Background for ease of processing.

Machine Learning Tutorial - Training and Validation Samples

Defining our Training and Validation Samples

Since the algorithm defines the optimization of a certain set of weights through minimizing a loss function, providing samples with unbalanced sizes might skew the NN's evaluation of its solution towards one. To avoid this we define two **training samples with similar dimensions** : one of signal and one of background.

One should always reserve a significant amount of events to validate the NN, in order to avoid **overfitting**. In our case, we save 20% of the smallest sample for validation and define the other one accordingly.

All that is left to do is to transform our samples into arrays : 1 for development (training), 1 for validation (X_class) and 1 for testing, and 1 with the score for the development set and another for the validation set (Y_class) (can also do it for the testing set).

Machine Learning Tutorial - Training Options

Now we need to define some options regarding our MLP. We present here some basic options, where we define our **loss function** as being the aforementioned **binary cross-entropy**, while utilizing the standard **Adam** tensorflow-keras optimizer.

As for the remaining primary features, we set as default **100 epochs**, which is a small number that allows for very fast, but not optimal results, with **batch sizes of 100**.

We will define a NN for binary classification, since we want to separate between a signal sample and a background sample. Here we define the **depth of the hidden layers** as being 2 layers, with the first one having a **width of 25 neurons**, which **decrease by a step of 5** at each subsequent layer.

These are all default values that work but can be tweaked to provide more optimal results.

Machine Learning Tutorial - Training Options

After giving the general structure of the network, we need to **define the shape of the input and output layers**, as well as **initiate every layer** and **specify the activation functions** for each neuron.

We add layers through tensorflow : :keras : :Sequential.add(Dense()), where the arguments for Dense() must include the number of neurons (here given by width), the kernel_initializer, that sets the default values for the neuron's weights, and the activation functions, which we set as the commonly seen ReLU. Notice that when we create the first layer we need to add the number of inputs through input_shape.

The last thing to note is that there are some Sequential.add(Dropout(x)) functions spread throughout the network. This function makes it so that a fraction x of the information is lost between layers. This is helpful to avoid early overfitting.

While reducing the dimension and complexity of the NN can also avoid overfitting, it can make it so that the near-optimal solution found is not as good as it could be, and the Dropout function allows for more complex NNs to have similar training loss scores to the validation ones for longer, hopefully providing a better final result.

Machine Learning Tutorial - Training the Model

The rest of the training code involves the Callbacks section, that defines some criteria on how and when we want to stop the training, and what data we want to save, both for the weights as well as for the history of the training.

Here we are choosing to save both the values of the loss function on the training sample, as well as for the validation sample.

Training the Neural Network

We start by defining a folder where we want to save our relevant data, such as the weights. **Always remember to alter this and keep a log of your runs**, so that you do not accidentally overwrite previous weights with good results!!!

Additionally, notice that we are choosing to save the set of weights that provides the best loss function result with the validation sample, and not the training one. Recall that overfitting can and always happens, given enough time.

When training your NN, always try to avoid early stops in the first iterations so you know what to expect.

Always check the loss function evolution plot and try to create NNs that maintain close training and validation loss values close for as long as possible.

Machine Learning Tutorial - Results

If you close your notebook after training a NN and want to revisit it later there is no problem. You just need to initiate your NN again (with the same configurations, so keep a good log of your work), open the files you saved from your training, and load the model again.

A standard check of your ML results is the **ROC curve** (Receiver Operating Characteristic). This is simply a plot of the true positive rate (TPR) as a function of the false positive rate (FPR), and gives a visual representation of the rate of true positives versus the rate of false positives as you increase the score threshold to determine what is or is not a signal event.

The closer the **area under the ROC curve** (AUC) is to 1, the better the separation between your samples.

Machine Learning Tutorial - Results

In order to obtain the NN scores for a given dataset, you just need to use the function **model.predict(data)**. This will return a pointer with an array for each output (which is usually one for each class unless you are doing a binary classification, where there is only 1 output), with each entry corresponding to the score attributed to each event.

Before applying the data to your NN always remember to select the correct features and normalize your data properly.

The only thing left is to establish a threshold to determine what we consider signal and what we consider background, and repeat the whole process again and again until we are happy with the selection efficiencies for both signal (as high as possible) and background (as low as possible) that our NNs provide.

Notes on the 1st Project

NOT grades of the first project

In general, projects were good

CMS is a general purpose experiment → Very unlikely to have very precise measurements

Best measurements will be related to high mass particles (Higgs, Z, W)

Decay width measured value should be very suspicious

Decay width comes from the decay probability

Decay probability depends on the final state

Full decay width Γ is calculated based on the sum of all decay modes

You are only looking at $X \to \mu^{\pm}\mu^{\mp}$

Z Boson References : OPAL Measurement ; LEP Combined Measurements