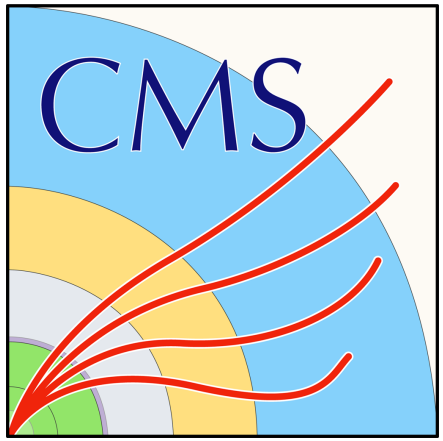
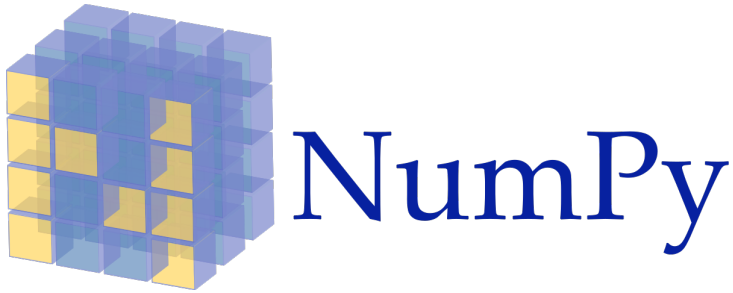
Deep learning in high energy physics Exercises: EOS PhD days

Willem Verbeke



Software tools



- https://www.tensorflow.org/
- https://keras.io/
- https://numpy.org/
- https://github.com/scikit-hep/uproot

Uproot basics

Feeding training data to a neural network in Keras uses a line as follows:

```
model.fit( train_data, train_targets, ... )
```

- train_data is a multidimensional numpy array containing input information for each event (last dimension of array is number of events)
- train_targets are the values the model has to predict
- most **modern machine learning libraries take numpy arrays** as input!
- **uproot** is a convenient library for **reading root trees into numpy arrays**

Example of listing the trees, and converting branches of a tree to arrays in uproot:

```
f = uproot.open('file.root')
print( f.keys() )
t = f['tree']
pt_array = t.array('pt') #returns numpy array
array_dict = t.arrays() #returns dict of numpy arrays
```

Exercise 1

- List all the trees in the root file GluGluHToWWTo2L2Nu_M125.root **using uproot**.
- Open the tree 'Events', and list all the branches in the tree.
- Compute the mean and standard deviation of the branch 'mtw1' (tip : np.mean and np.std respectively compute the mean and standard deviation of an array)
- Gradient descent tends to be much more numerically stable if input variables are normalized. Make an array representing the branch 'mtw1', and normalize it (center it at 0 and make it have unit variance)

Solutions available as python script and Jupyter notebook:

- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/solution_exercise_1.py
- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/solution_exercise_1.ipynb

Exercise 2

- For the signal (GluGluHToWWTo2L2Nu_M125.root) and both of the backgrounds (DYJetsToLL_M-10to50-LO.root and DYJetsToLL_M-50-LO.root) make a dictionary containing arrays representing all the following branches: 'ptll', 'mth', 'uperp', 'upara', 'ptTOT_cut', 'mTOT_cut'
- For training a neural network on the given dataset we will need a 2D array containing the input variables for each event. Write a function that converts an array of dictionaries to a 2D array of dimensions (number of variables, number of events). (Tip: You can use np.expand_dims to expand the dimensionality of an array, and np.concatenate to concatenate arrays along an existing dimension)
- Make 2D arrays representing all the input variables for signal and background events. For the backgrounds, merge the arrays.

Solutions in the form of a python script, and a jupyter notebook guiding the exercise are available:

- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/guide_exercise_2.ipynb
- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/solution_exercise_2.py

Exercise 3

- Each event has an associated weight which should be taken into account when training a machine learning algorithm (for instance processes with a higher cross sections should typically have larger weights). The weights in the given dataset are obtained by multiplying the following branches: 'XSWeight', 'SFweight2l', 'LepSF2l__ele_mvaFall17V1Iso_WP90__mu_cut_Tight_HWWW', 'LepCut2l__ele_mvaFall17V1Iso_WP90__mu_cut_Tight_HWWW', 'GenLepMatch2l', 'METFilter_MC'
- Read in the weight branches as arrays for each of the backgrounds and the signal. Multiply them to obtain a single array representing the weights for each event (do this for signal and backgrounds). Merge the weight arrays for the backgrounds.
- For numerical stability of the neural network training it is better if the weights have a scale of the order one. Divide all the weights by their mean for signal and background events. (Note that this has to be done after merging the array of weights for the background.)
- During training we will also need targets or 'labels' for each event. Make arrays representing the targets for each event : 0 for signal and 1 for background.

Solutions in the form of a python script, and a jupyter notebook guiding the exercise are available:

- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/guide_exercise_3.ipynb
- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/solution_exercise_3.py

Exercise 4

- When training a machine learning algorithm the available data sample should be split in 3 pieces, namely training, validation and test sets. The validation set is used for evaluating the performance of particular hyperparameters, and after these are fixed the model should be checked on the test set.
- Before splitting a dataset it has to be randomized so all orthogonal samples come from the same distribution. Randomly shuffle the weights, input arrays and labels for signal and background. Note that they need to be randomized simultaneously (each event must have its correct corresponding weight and label!).
- Write a function to split an array in three, with a given validation and test fraction.
- Split the randomized weights, labels and inputs for signal and background into training, validation, and test sets.

Solutions in the form of a python script, and a jupyter notebook guiding the exercise are available:

- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/guide_exercise_4_and_5.ipynb
- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/solution_exercise_4.py

Exercise 5

- For training a neural network we need a training and validation set given as arrays containing both signal and background events in random order.
- Merge the training, validation and test sets for signal and background (inputs, labels and weights).
- Randomize each of the arrays again (keep in mind that inputs, labels and weights are shuffled simultaneously). (Careful readers might think that the randomization done in Exercise 4 was not needed, but this is not the case since we want the ability to make plots showing the signal and background training and validation sets separately. When initially doing the splitting we already need randomization to avoid any bias.)
- Now we are ready for training the neural network. Copy the code from the solution of exercise 5 for training a neural network and evaluating its performance (lines 131 and below). Train the neural network and study the output plots.

Solutions in the form of a python script, and a jupyter notebook guiding the exercise are available:

- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/guide_exercise_4_and_5.ipynb
- https://github.com/wverbeke/DeepLearningTutorialLouvain/blob/master/solution_exercise_5.py

Exercise 6 (optional, advanced)

- In the previous exercises you have always read the entire dataset directly into memory. For many other datasets this will not be possible, as they are too large. In this case a **data generator** has to be written.
- Write generator functions yielding training, validation and test data, and train the neural network from the previous exercise using these generator functions. To do this efficiently, one should read in chunks of the data into memory, subsequently generate batches, and when the current chunk is exhausted read in a new one. Reading batches one-by-one tends to be very inefficient since an entire basket in the root file will have to be decompressed to yield each batch. (see https://github.com/scikit-hep/uproot). When deciding the size of the chunks to read in, it is a good idea to estimate the size of a decompressed event and compare this to the available memory of the system. Larger chunks will result in a faster generator.
- When a root file does not fit into memory you can not randomize it like in the previous exercises! Now write a script to randomize a given root file. You must come up with an algorithm to do this that does not involve reading random events from the input root file, since this will be slow to the extent that it is fully unusable as each event will need a basket decompression.

The following code I developed and use for training neural networks might be a guide to this exercise:

- https://github.com/wverbeke/deepLearning/blob/master/DeepSets/PFNSampleGenerator.py
- https://github.com/wverbeke/deepLearning/tree/master/randomizer