KIT-Fakultät für Physik
Institut für Experimentelle Teilchenphysik
Dr. Matthias Schröder
Sebastian Wieland

Karlsruher Institut für Technologie

# Teilchenphysik 2 — W/Z/Higgs an Collidern

## Sommersemester 2019

## Exercises No. 7

Discussion on July 16, 2019

---

In this exercise, we will study and train neural networks (NNs).

The training of complicated NNs is quite time consuming. Fortunately, when working with NNs, many calculations can be parallelised. Nowadays one uses the shader processing units of graphical processing units (GPUs) for the parallelisation. In our exercise, you will be given access to a dedicated computer with a powerful GPU (with the fitting name "gpumonster").

The exercise will be a computer exercise, and the first part will be done during class time on July 16th ("Präsenzübung"). The second part is do be done at home during the following week, and the results will be discussed class time on July 23rd.

For the exercise you will need a computer with the possibility to use `ssh`. This is native on linux and mac. On Windows you can use putty (https://www.putty.org/) or for example the new Windows 10 Subsystem for linux (https://docs.microsoft.com/en-us/windows/wsl/about; worth a look if you do not know this yet).

Therefore, **you are asked to bring a laptop with a working Wifi access and `ssh` capabilities!** You can work in small groups of up to three persons, and it is sufficient to have one laptop per group.

The first problem (setup) can already be done before the class. But it is brief and you can also do it during the class. You do not need to work on any other problems before the class on July 16th!

All further instructions will be provided during the class and the remainder of this exercise sheet.

## Exercise 1: Setting up the environment

First, connect to the "gpumonster" with `ssh` using for example the following command

```
ssh -Y higgsuserN@gpumonster.ekp.kit.edu
```

where `N` is your assigned group number in the class. Please use the `-Y` flag to allow forwarding of graphical output. The password for the first login is `HiggsForTheMasses` and needs to be changed after the initial login. This can be done via `passwd`.

Once connected, navigate to your home directory (`/local/scratch/` if you are not already in there. Then, in this directory, clone the following git repository and checkout the branch for the exercise using the following command:

```
git clone -b TP2-WZH https://github.com/kit-cn-cms/DRACO-MLfoy.git
```

The scripts and material you will need are all provided with the repository.

To actually train and evaluate the NNs we will use rather simple python scripts. These implement functionalities from KERAS (https://www.tensorflow.org/) using Google's Tensorflow (https://www.tensorflow.org/ in the backend, which will run on an NVidia GPU using the CUDA software. This is already installed on the gpumonster but you need to install some missing python packages. To do this follow the steps explained in the DRACO-MLfoy repository https://github.com/kit-cn-cms/DRACO-MLfoy/tree/TP2-WZH:

```
pip install --user uproot==3.2.7
pip install --user uproot-methods==0.2.6
pip install --user awkward==0.4.2
export KERAS_BACKEND=tensorflow
pip install numpy==1.15.4
```

**If you encounter any problems or you have questions or suggestions, do not hesitate to contact** `sebastian.wieland@kit.edu` .

## Exercise 2: Separating $t\bar{t}H$ and $t\bar{t}$ with an DNN

Now we want to apply neural networks to a particle physics task, specifically the classification of an collision event recorded by the CMS detector as coming either from $t\bar{t}H$ or from $t\bar{t}$ production. This is a difficult task as will be explained in the lecture.

To train the classifier we will need known and labelled $t\bar{t}H$ and $t\bar{t}$ events. In our case, we use simulated events which were simulated with the Powheg MC generator and

the Pythia8 parton shower. The events were simulated for a centre-of-mass energy of 13 TeV. After the Powheg+Pythia steps, the events are run through a simulation of the CMS detector to get realistic observables as also seen in the CMS detector. Finally, particles are reconstructed (using the particle-flow algorithm), and jets are clustered from them. In the end, we have ROOT ntuples with reconstructed jets and leptons, from which we can compute higher-level features.

The simulated events are weighted such that the sum of weights of simulated events in a specific phase space corresponds to the expected number of real events in this phase space.

- Why are the events weighted?

On top of that, the weights of each sample ($t\bar{t}H$ or $t\bar{t}$) are normalised such that the sum of all weights in the sample equals 1.

- Why are the weights normalised to 1?

Finally, we apply an event selection where we require events to have exactly 1 lepton, $\geq 6$ jets of which $\geq 3$ have to be b-tagged.

- Why this selection?

In this exercise, the feature vector consists of the values of the reconstructed observables such as the $p_T$ of the various jets and leptons and many more. The full list of the 21 input features can be seen in this file:
`variable_sets/variables_challenge.py`.

KERAS itself can not handle the superior ROOT data format. Therefore, a preprocessing is necessary to convert into pandas dataframes (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html). This has already been taken care of for you ;)

We will start with a simple NN defined in
`train_scripts/net_configs.py`. Take a look at the code.

- What is the architecture of the network?
- Discuss dropout, L2 regularisation and early stopping during class!

The training itself is invoked by the python script `train_challenge_tt.py`. Most functionality is handled via command line options. Have a look at the available options.

Now run the script via:

```
python train_challenge_tt.py -v variables_challenge  --binary
  --signalclass ttH -n example_config_challenge -o challenge_test
```

The printout will show the loss and accuracy for each epoch evaluated on the training and the test datasets. It will also indicate when early stopping was invoked. After the training has finished it will report the ROC-AUC values for both the train and test dataset. *Reminder: The Receiver-Operator-Characteristic (ROC) is the curve one gets when plotting the background-rejection efficiency versus the signal-efficiency for different cuts on the classifier output. The area under the curve can be used as a measure how well the distributions of signal and background are separated.*

The script will also create a directory in the workdir with the name specified with the `-o` option, which contains various plots. Take a look at these plots:

- What is plotted?
- Did overtraining occur?

**Exercise 3: Higgs-Challenge**
Train the two best DNNs to separate $t\bar{t}H$ events from

- $t\bar{t}$ + jets events
- $t\bar{t}$ + $b\bar{b}$ events.

Consider following questions:

- What's the difference between the two cases?
- Why could it be beneficial to train a neural net to only separate the $t\bar{t}H$ signal from a subset of the backgrounds?
- Can a multiclassification help?

**Rules of the challenge**

- Your task is to try to change the architecture, the optimiser, the dropout and other parameters (and generally play around with the code) to achieve the best ROC-AUC values possible.
- Select suitable InputVariables. You can plot the variables for signal and a given background using

  ```
  python plotInputVariables_challenge_xx.py
    -o plots_challenge -v variables_challenge,
  ```

  where xx is either `tt` or `ttbb`.
- You're only allowed to use **10 input variables** for each DNN!!!
- Hint: To avoid overwriting previous results, make use of the `-o` option.
- Deliver your best performing NNs by providing at least the created `checkpoints` directories (better: the whole `workdir` one)
- Inform Sebastian (sebastian.wieland@kit.edu) of the location by 23:59 on july 21nd.
- We encourage you to work together in groups.
- The winner will be announced in class on July 23rd. There are of course prizes to be won!