A simple ML framework setup

Dinos

January 17, 2018

INFN Lecce

Useful links

Miniconda: https://conda.io/docs/index.html

Keras: https://keras.io/

Scikit-learn: http://scikit-learn.org/stable/

RootNumpy: http://scikit-hep.org/root_numpy/index.html

Numpy: http://www.numpy.org/

SLAC workshop: https://indico.cern.ch/event/615994/page/10587-software-setups-and-data-access

Setup

Host machine setup

On the terminal you are running, you will need to create a **.theanorc** file in your home directory, containing the following:

```
[global]
device = cpu
floatX = float32
force_device=True
[gcc]
cxxflags=-march=corei7
```

I tried yesterday to get the framework running on lxplus but got too many installation errors. Try with your institute's nodes.

I'm using bash for in my shell but there should be no problem to use other types

Miniconda Installation

1. Download the Miniconda package

   ```
   wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
   ```

2. Install

   ```
   sh Miniconda3-latest-Linux-x86_64.sh
   ```

3. make sure that

   ```
   export PATH="/afs/cern.ch/user/k/kbachas/miniconda3/bin:$PATH"
   ```

   is under your login script (e.g bashrc or bash_login)

4. After the conda installation is complete, close the terminal and reopen it
5. Add installation channels

   ```
   conda config --add channels https://conda.anaconda.org/NLeSC
   ```

Setup a Conda environment

To create a Conda environment simply type:

```
conda create --name=myAIenv
```

To activate this environment, use:

```
source activate myAIenv
```

This will now give you a characteristic `(myAIenv)` in front of your prompt

To deactivate an active environment, use:

```
source deactivate
```

Change the file `.keras/keras.json` to read:

```
{
"epsilon": 1e-07,
"floatx": "float32",
"image_data_format": "channels_last",
"backend": "theano",
"image_dim_ordering": "tf"
}
```

Install remaining and ML related packages I

Activate your environment first!

```
source activate AIenv
```

Install whatever you need with `conda install` expressions

```
conda install root=6 python=2 mkl jupyter numpy scipy matplotlib
scikit-learn h5py rootpy root-numpy pandas scikit-image theano
pydot
```

and then through `pip`

```
pip install keras
```

Troubleshooting(Francesco's recipe!):

Go in `miniconda3/envs/[nameoftheenvironmet]/lib`

and in `miniconda3/lib`

and relink `libstdc++.so` and `libstdc++.so.6` to `libstdc++.so.6.0.24`

Install remaining and ML related packages II

```
rm libstdc++.so.6 libstdc++.so
ln -s libstdc++.so.6 ibstdc++.so.6.0.24
ln -s libstdc++.so ibstdc++.so.6.0.24
```

then go ahead and install the remaining packages

```
pip install scikit-optimize
pip install seaborn
```

and finally:

```
export MKL_THREADING_LAYER=GNU
```

Install ML Code

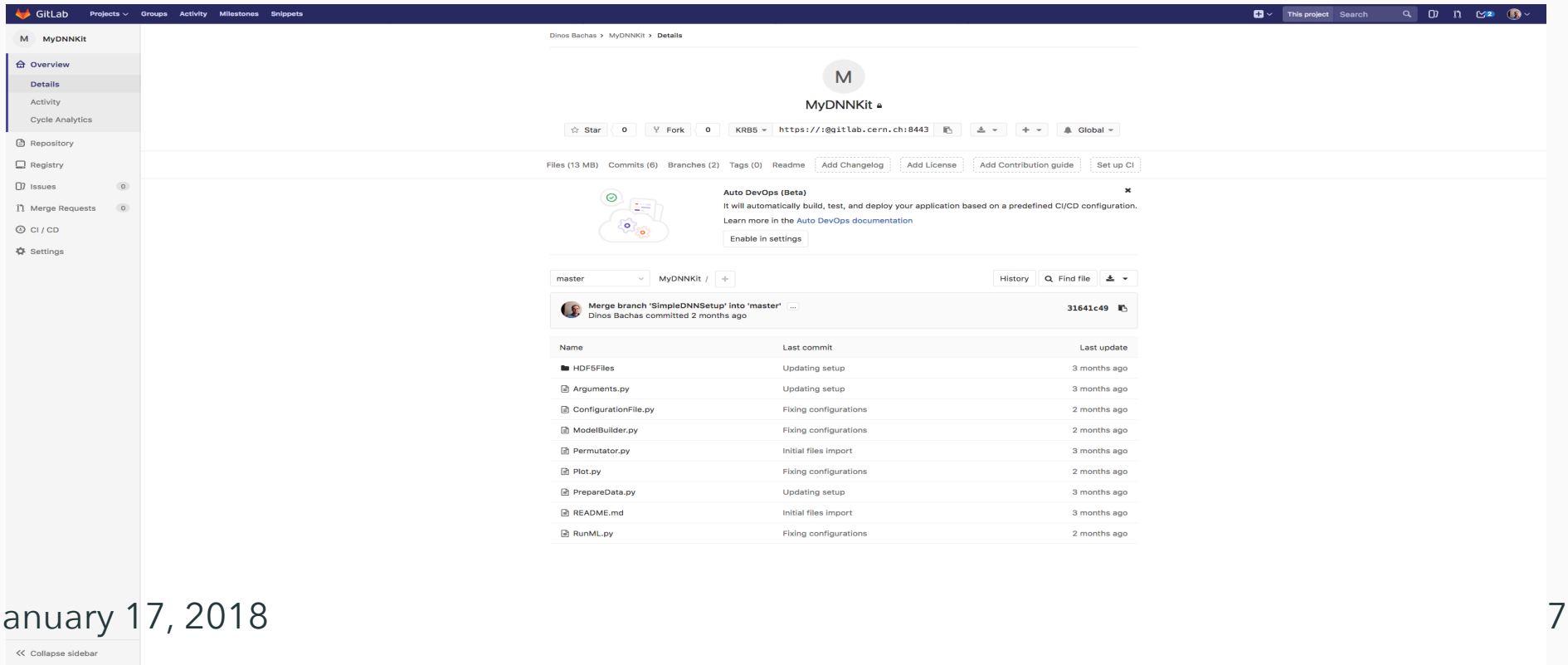
Download MyDNNKit package

Our Lecce code lives under git here:

`https://gitlab.cern.ch/kbachas/MyDNNKit`

To download it simply type:

```
git clone ssh://git@gitlab.cern.ch:7999/kbachas/MyDNNKit.git
TutorialDNNKit
```



MyDNNKit Elements I

RunML.py

Driving python file. Holds the configuration and basic routines to be run:

1. Reads the ConfigurationFile.py
2. controls if hdf5 Files should be created
3. Pre-Process the Data
4. Defines if Train should be run

MyDNNKit Elements II

ConfigurationFile.py

- Defines I/O paths, I/O files
- Defines Variable set, Parameters for `"Width" ,"Depth" ,"Epochs"`
- Creates permutations out of the Parameters and VarSet

MyDNNKit Elements III

ModelBuilder.py

Holds the basic DNN building method with dropout by default to 20%

```
def BuildDNN(N_input,width,depth):
```

N_input: Number of input variables

width: Number of NN nodes

depth: Number of hidden layers to use

Permutator.py

Provides the actual permutations in ConfigurationFile.py

MyDNNKit Elements iv

PrepareData.py

Holds method to:

- Create HDF5 files from the flat Ntuples(=can be the output of CxAODReader)
- Randomize the data
- Prepare sample to have [50:50] Signal and Background. Backgrounds are of equal fractions themselves

Run the DNN

Create HDF5 data out of flat ntuples I

First, in the ConfigurationFile.py change:

```
InputMLNtuplePath="/afs/le.infn.it/user/k/kbachas/ML/MyDNNKit
/InputMLNtuples/"
```

to

```
InputMLNtuplePath="/afs/le.infn.it/user/k/kbachas/public
/InputMLNtuples/"
```

and

HDF5Path to

```
HDF5Path ="[yourpath]/TutorialDNNKit/HDF5Files/"
```

Create HDF5 data out of flat ntuples II

python RunML.py -c -s 1000000

will create HDF5 files for each channel, split it in 1M events each and save to `HDF5Path`

- **-c:** CreateHDF5Files=True
- **-s N:** Split in N events

Mixing and shuffling of data.

python RunML.py -p

- Uses the hdf5 data from `HDF5Files` directory to create Panda Dataframes.
- Mixing signal and background with equal proportions. Backgrounds get mixed by equal proportions as well. Signal N events drives the size of the mixed S+B sample.
- The Panda Dataframe is saved as a 'pickle file' `MixData_PD.pkl` under `HDF5Files`

Build and Train the DNN

python RunML.py -t -y <N>

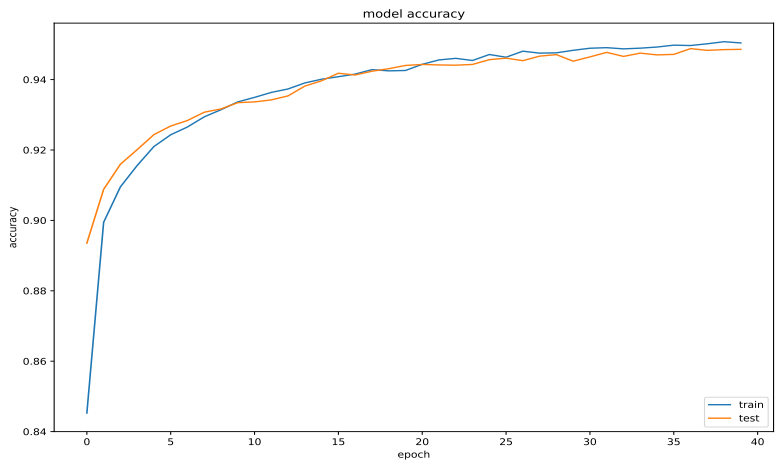
- Runs on the .pkl file
- DNN model building is driven by the option `-y <N>`, where `N` is the number of the parameter set which defines the NN
- `-y -1` will dump all available parameter combinations
- If no training is specified then the code tries to load a pre-existing model of the same -y Set

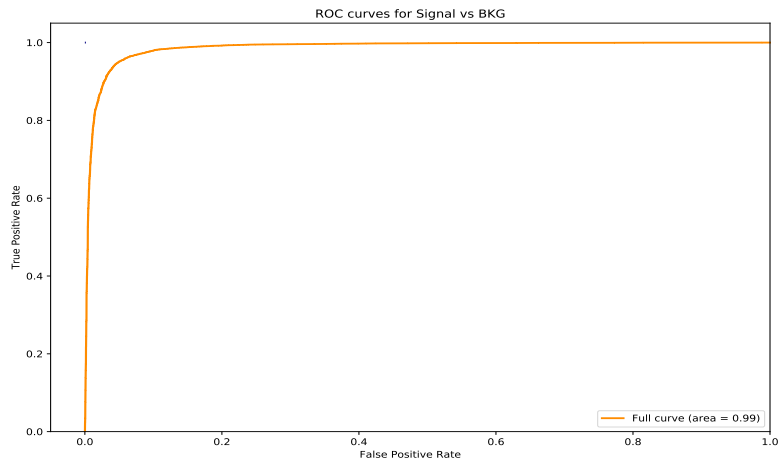
```
Params={ "Width":[32,64]
         ,"Depth":[2,3,4]
         ,"Epochs": [40,60,80]
         ,"VarSet": [ i for i in xrange(0,len(SelectedFields))]
         }

SelectedFields = [
['lep1_pt','lep1_eta','lep1_phi','lep1_E','lep2_pt','lep2_eta','lep2_phi','lep2_E','fatjet_pt',
'fatjet_eta','fatjet_phi','fatjet_E','fatjet_C2','fatjet_D2','MET','Zll_mass','Zll_pt'],
['lep1_pt','lep2_pt','fatjet_pt','lep1_E','lep2_E','fatjet_E','Zll_pt','MET']]
```

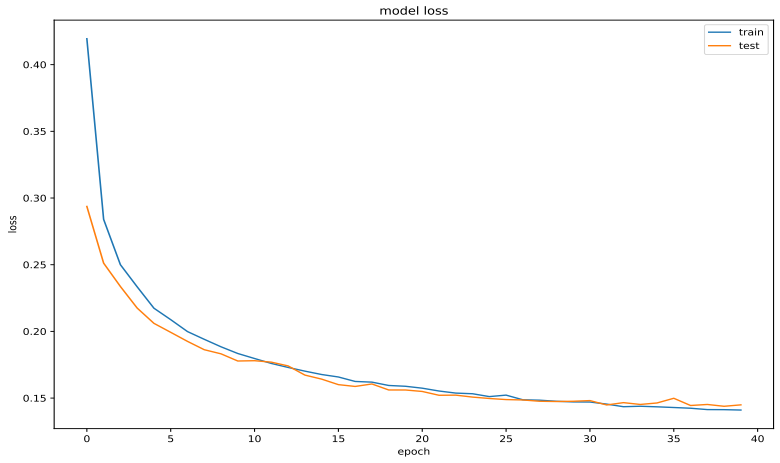
Example

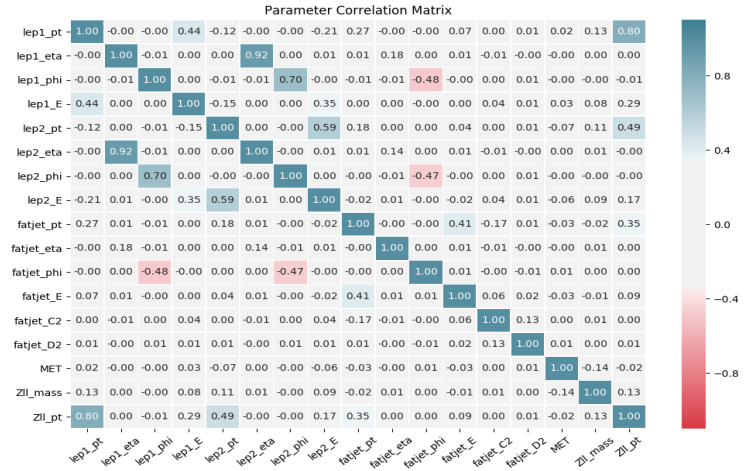
```
(AIenv) [wn-21|13:53]~/TutorialDNNKit >>> python RunML.py -y -1 -t
Using Theano backend.

Using config file: ConfigurationFile.py
Using HyperParamSet:  -1
Will Train Network

HyperParameter Scan:  54 possible combinations.
SetList:
0 : {'Epochs': 40, 'Width': 32, 'Depth': 2, 'VarSet': 0}
1 : {'Epochs': 40, 'Width': 64, 'Depth': 2, 'VarSet': 0}
2 : {'Epochs': 40, 'Width': 32, 'Depth': 2, 'VarSet': 1}
3 : {'Epochs': 40, 'Width': 64, 'Depth': 2, 'VarSet': 1}
4 : {'Epochs': 40, 'Width': 32, 'Depth': 2, 'VarSet': 2}
5 : {'Epochs': 40, 'Width': 64, 'Depth': 2, 'VarSet': 2}
6 : {'Epochs': 60, 'Width': 32, 'Depth': 2, 'VarSet': 0}
7 : {'Epochs': 60, 'Width': 64, 'Depth': 2, 'VarSet': 0}
8 : {'Epochs': 60, 'Width': 32, 'Depth': 2, 'VarSet': 1}
...
```

Output


model accuracy
accuracy
epoch
train
test


ROC curves for Signal vs BKG
True Positive Rate
False Positive Rate
Full curve (area = 0.99)


model loss
loss
epoch
train
test


Parameter Correlation Matrix
lep1_pt
lep1_eta
lep1_phi
lep1_E
lep2_pt
lep2_eta
lep2_phi
lep2_E
fatjet_pt
fatjet_eta
fatjet_phi
fatjet_E
fatjet_C2
fatjet_D2
MET
Zll_mass
Zll_pt