
Description

The Classifier class is a way to use a Keras deep neural network (DNN) inside a C++ script. This is achieved by calling the Python interpreter and converting values back and forth between C++ and Python. The class is designed to be high-level, so all the conversions and wrapping happen behind the doors.

Classifier relies on a valid installation of Python 2.7 including the Theano and Keras libraries, as well as the presence of a trained DNN (as a .h5 file) and an internal file called X_max.npy. See the Installation section for details

The neural network is designed to discriminate between a high energy proton and a high energy electron as detected by the BGO calorimeter on the DAMPE satellite. The variables used (see further sections) were selected to characterise a showering particle inside a calorimeter.

How to use

Dependencies

- Python
  - Python 2.7
  - Keras 2.0.4 or newer
  - Theano 0.8 or newer
- C++ / ROOT
- DAMPE software
  - DmpEvent class

To fulfill all these requirements without problem, it is recommended to use the DAMPE CVMFS package *both at compilation and run time.* Please source the package :

```
source /cvmfs/dampe.cern.ch/rhel6-64/etc/setup.sh
```

and then initialise the DAMPE framework :

```
dampe_init
```

Installation and compilation

Get the latest version of the code straight from github :

wget -nv https://raw.githubusercontent.com/david-droz/DmpAnalysis/master/cppWrapper/getFiles.sh
bash getFiles.sh

Source the cvmfs package. These must be called both at compilation time and run time :

```
source /cvmfs/dampe.cern.ch/rhel6-64/etc/setup.sh
dampe_init
```

In your compilation/makefile, include the following lines

```
-I/cvmfs/dampe.cern.ch/rhel6-64/opt/externals/python2.7/latest/build/include/python2.7 \
-I/cvmfs/dampe.cern.ch/rhel6-64/opt/releases/latest/include \
-I/cvmfs/dampe.cern.ch/rhel6-64/opt/externals/root/latest/include \
-L/cvmfs/dampe.cern.ch/rhel6-64/opt/releases/latest/lib \
-l DmpEvent \
```

See a compilation example under

```
/dampe/data3/users/ddroz/keras-wrapper/compile.sh
```

Usage in code

In code header:

```
#include "Classifier.h"
```

Initialise an instance:

```
Classifier c ;
```

To get the DNN score of a given event, use the getScore() method.

```
double eventScore = c.getScore(bgorec);
```

Several overloaded functions exist, see the code documentation below. Finally, at the end of your code, please call the finalize method for some house keeping:

```
c.finalize() ;
```

Class description and methods

Constructors

```
Classifier::Classifier(void)
```

Standard constructor. Initialises the Python interpreter and loads the associated Python script (calculateScore.py)

```
Classifier::Classifier(string)
```

Optional constructor. Argument is the name of the associated Python script

getScore methods

These methods analyse a given event and return the DNN "score" associated to the event. A score higher than 0 means it's likely an electron, and lower than 0 means it's likely a proton.
The difference between the following methods is only the format of the argument.

```
double Classifier::getScore(DmpEvent*)
double Classifier::getScore(DmpEvtBgoRec*)
```

Return the DNN classifier score of a given event, by accessing observables through DmpEvent/DmpEvtBgoRec built-in methods.

```
double Classifier::getScore(double eneLayer[14],double rmsLayer[14],
                      double hitsLayer[14], double longitudinalRMS,
                      double radialRMS, double Etot, double hits,
                      double XZslope, double YZslope)
```

Takes as input the explicit variables used by the DNN. See below.

```
double Classifier::getScore(vector <double>)
double Classifier::getScore(double[48])
```

Takes as input the full list of 48 variables. The corresponding DmpSoftware methods are, in respective order:
DmpEvtBgoRec::GetELayer(i)
DmpEvtBgoRec::GetRMS2[i]
DmpEvtBgoRec::GetLayerHits()[i]
DmpEvtBgoRec::GetRMS_l()
DmpEvtBgoRec::GetRMS_r()
DmpEvtBgoRec::GetTotalEnergy()
DmpEvtBgoRec::GetTotalHits()
DmpEvtBgoRec::GetSlopeXZ()
DmpEvtBgoRec::GetSlopeYZ()

```
void Classifier::finalize()
```

Cleans up the process by deleting pointers and closing the Python interpreter.

Technical details on neural networks

The model used is a feed-forward neural network consisting of four densely connected layers, with respectively 300, 150, 75 and 1 neurons. The activation functions are respectively ReLU, ReLU, ReLU, and None.

The training was performed on a slightly different model, including 10 % dropout on every hidden layer and a sigmoid activation at output. The optimisation was performed using Adam, minimising the binary cross-entropy loss function over 70 epochs. After training, the weights were transfered to the model described previously.

The data used for training are large sets of simulated "Monte-Carlo" data, reprocessed using the version 5.4.2 of the DAMPE Software framework. Events were cleaned up by a few basic selection cuts, and then described by a set of 47 variables. Every variable was normalised independently to fit in the 0-1 range. The training set included nearly 2 million events, split evenly into positive signal class (electrons) and negative background class (protons)

Known issues & Contact

1. Error message when running : *error while loading shared libraries*
   Solution : source the cvmfs package and initialise the dampe framework

Contact : david.droz@cern.ch