TMVA Tutorial

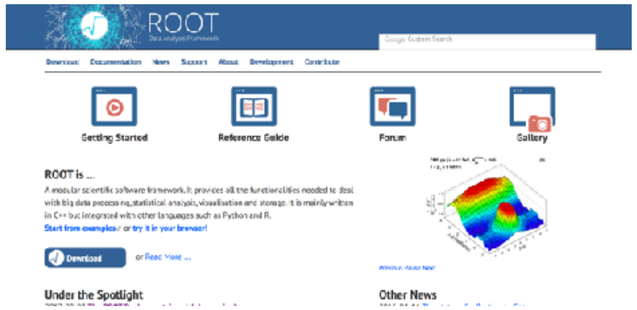
L. Moneta (CERN EP-SFT)

INSIGHTS Workshop on Statistics and Machine Learning

17-21 September 2018, CERN

ROOT

- ROOT is a software toolkit which provides building blocks for:
- Data processing
- Data analysis
- Data visualisation
- Data storage



- ROOT is written mainly in C++ (C++11 standard)
- Bindings for Python are provided.

http://root.cern.ch

- Adopted in High Energy Physics and other sciences (but also industry)
- ~250 PetaBytes of data in ROOT format on the LHC Computing Grid
- Fits and parameters' estimations for discoveries (e.g. the Higgs)
- Thousands of ROOT plots in scientific publications

TMVA

- ROOT Machine Learning tools are provided in the package TMVA (Toolkit for MultiVariate Analysis)
- Provides a set of algorithms for standard HEP usage
- Used in LHC experiment production and in several analyses
  - several publications produced using TMVA
- Development done in collaboration with CERN experiments and HEP community
- HEP Software Foundation (HSF) community:
  - Machine Learning white paper
  - importance of providing internal machine learning software tools for HEP

Key Features of TMVA

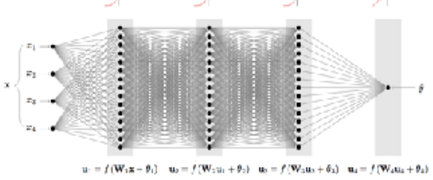
- Facilitates HEP research, from detector to analysis
  - best suitable for HEP analysis with direct connection to ROOT I/O
  - written in C++
- Good performance (makes use of GPU and CPU parallelisation)
- Stability of interfaces
- Easy to use
- Long term support
- **Challenge in integrating new algorithms**
  - Machine learning world evolves very fast
- **Several features added recently** (e.g. deep learning)
- **Interfaces to integrate external tools** easily (from Python and R)

TMVA Methods

Available methods in the old version (up-to 2015):

- Rectangular cut optimisation
- Projective likelihood estimation (PDE approach)
- Multidimensional probability density estimation (PDE - range-search approach)
- Multidimensional k-nearest neighbour classifier
- Linear discriminant analysis (H-Matrix and Fisher discriminants)
- Function discriminant analysis (FDA)
- Artificial neural networks (various implementations)
- Boosted/Bagged decision trees
- Predictive learning via rule ensembles (RuleFit)
- Support Vector Machine (SVM)

New Features

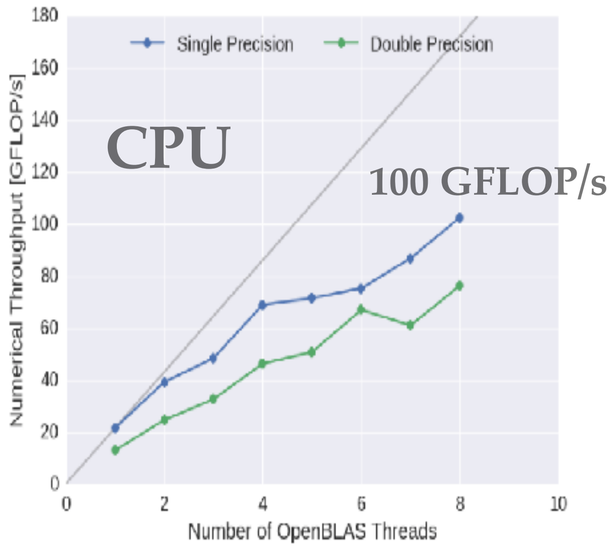


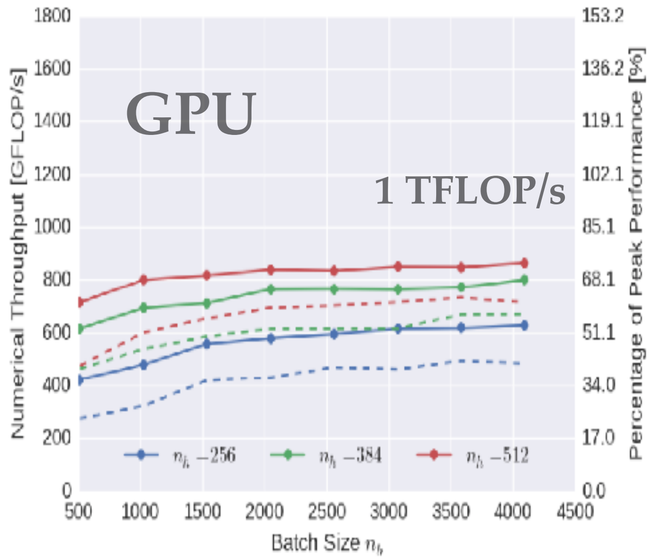
New major features added since 2016 and available in the latest ROOT versions:

- Deep Learning
  - support for parallel training on CPU and GPU (with CUDA and OpenCL)
- Cross Validation and Hyper-parameter optimisation
- Improved BDT:
  - new loss functions for regression
  - improved performances with multi-thread parallelisations
- Interfaces to external ML library in R and Python
- Interactive training and visualization for Jupyter notebooks

Deep Learning in TMVA

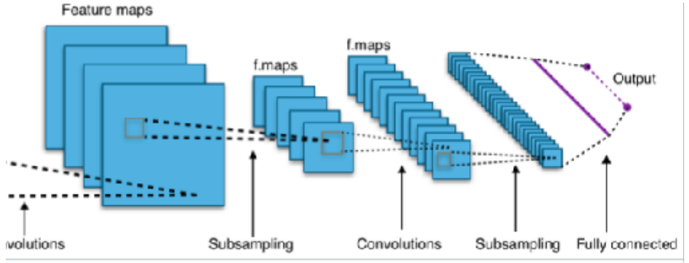
- Available for dense layers since ROOT 6.08
- extended the design in 6.14 to a new module supporting different layer types
  - parallel evaluation on CPU
    - implementation using OpenBlas and Intel TBB library
  - GPU support using CUDA
  - Excellent performance and high numerical throughput
    - see the presentation from S. Pfreundschuh

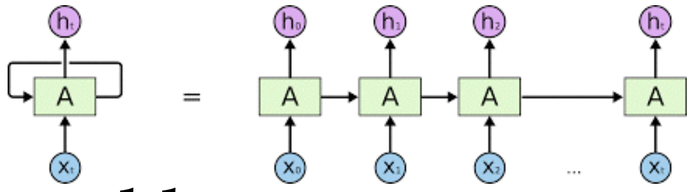




New Deep Learning Features

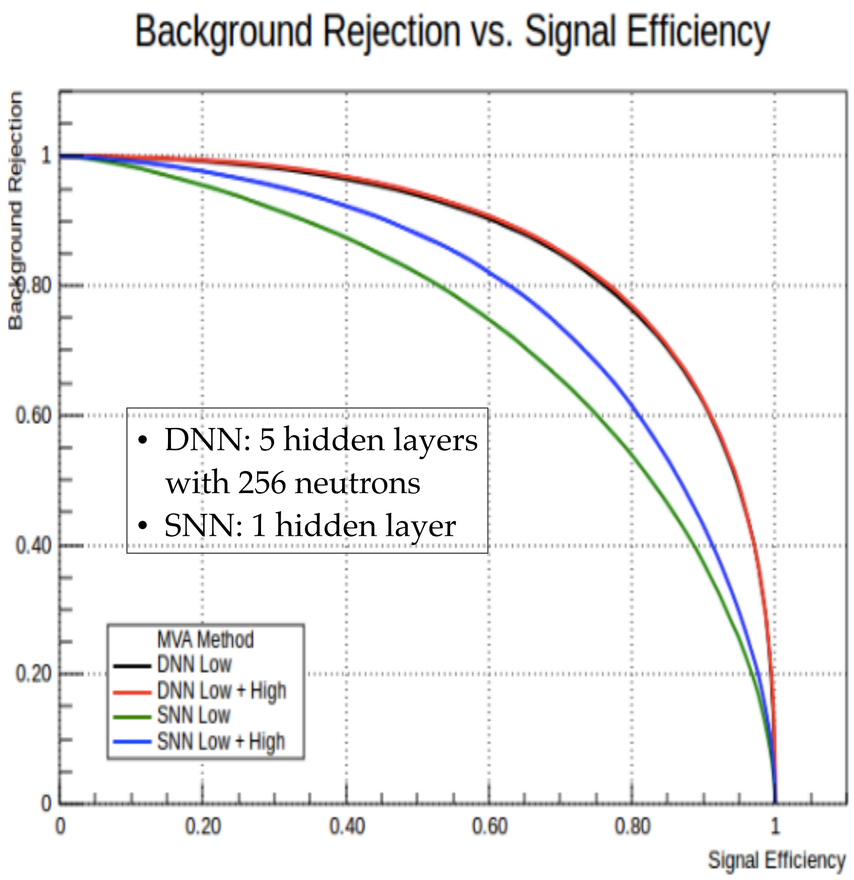
- Available in latest ROOT version 6.14 and/or ROOT master
  - Extended Deep Learning Module with support for
    - Dense Layer
    - Convolutional Layer
      - powerful for image data sets
    - Recurrent Layer
      - useful for time dependent data
  - GPU support for Dense and Convolutional layers
  - with several optimisers available (e.g. SGD, ADAM, …)





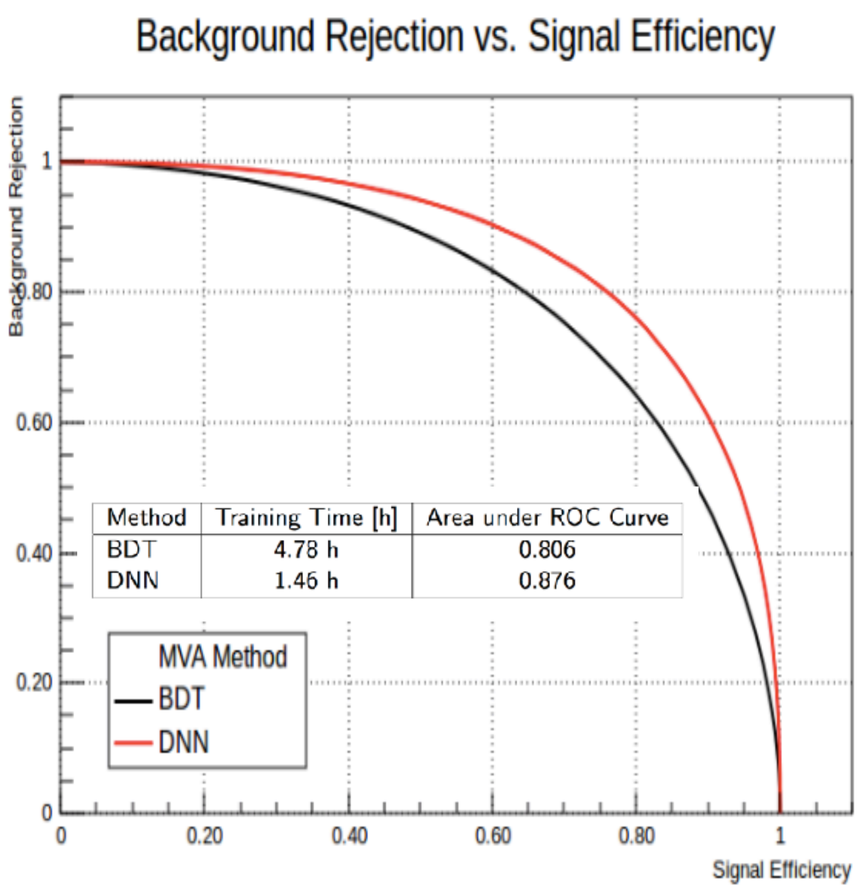
Deep Learning Performance

DNN vs Standard ANN



- DNN: 5 hidden layers with 256 neutrons
- SNN: 1 hidden layer

DNN vs BDT



Method	Training Time [h]	Area under ROC Curve
BDT	4.78 h	0.806
DNN	1.46 h	0.876

- Using Higgs public dataset (from UCI) with 11M events
- Significant improvements compared to shallow networks and BDT

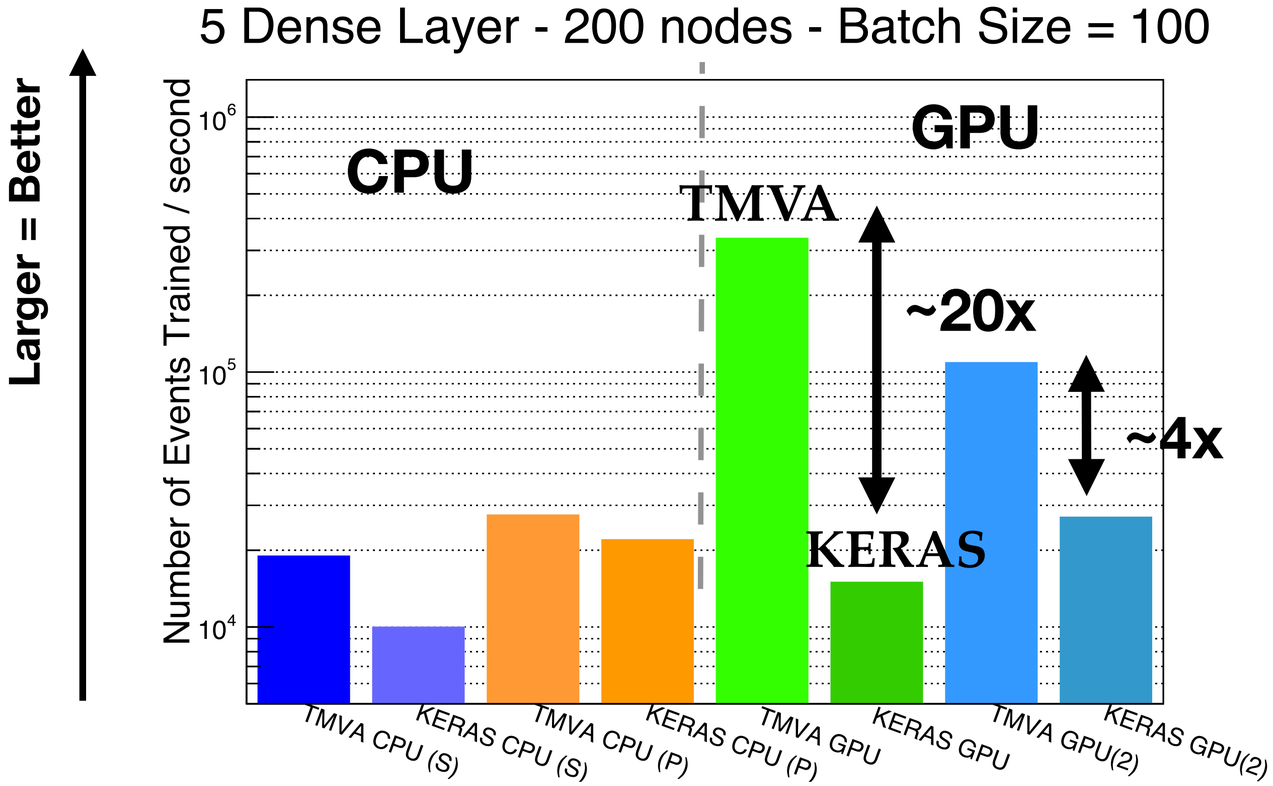
DNN Training Performance

Training time — Dense networks

- Higgs UCI dataset with 11M Events
- TMVA vs. Keras/Tensorflow
- "Out-of-the-box" performance

Excellent TMVA performance !

- How does it scale ?
  e.g. increasing batch size ?



(S) — Single threaded

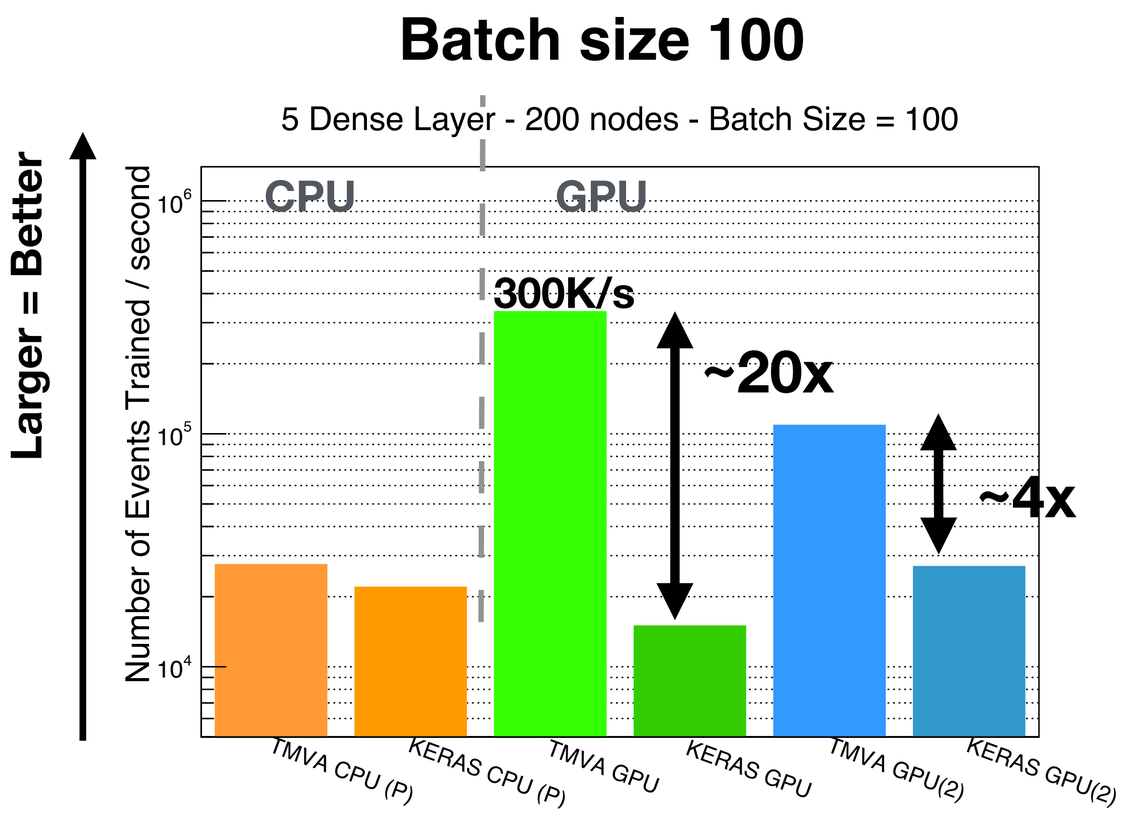
(P) — Intel Xeon E5-2683 (28 core)

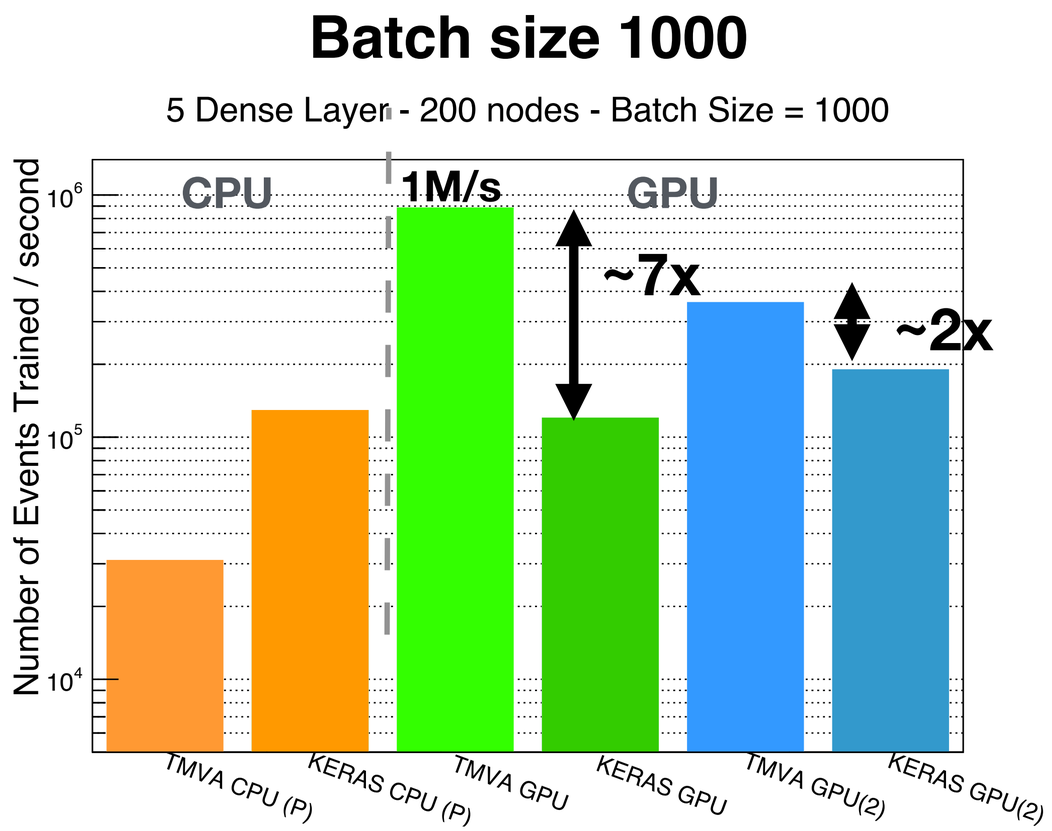
GPU — GTX1080Ti

GPU(2) — GTX980

DNN Training Performance

TMVA vs. Keras/Tensorflow on CPU and GPU using a typical HEP dataset





- Key difference is GPU utilisation
  - Tensorflow optimised for large operations

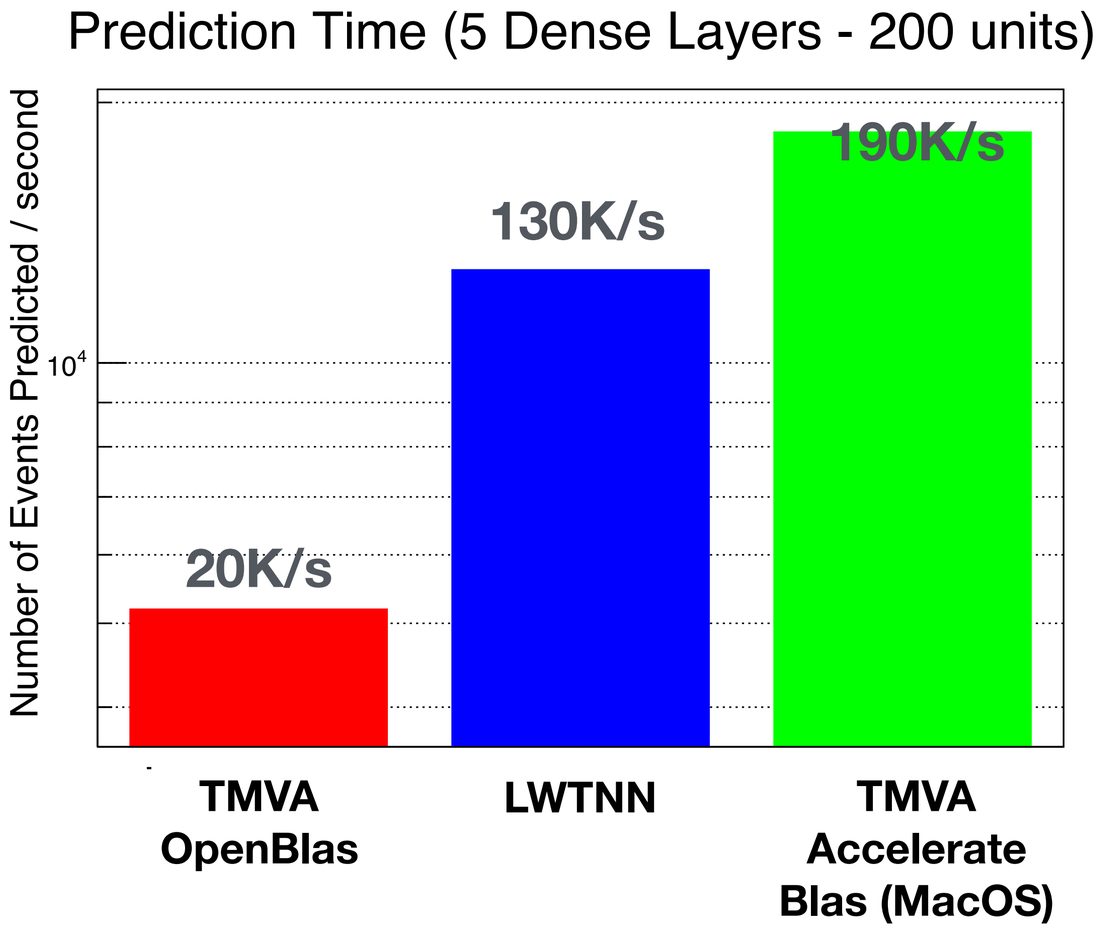
CPU— Intel Xeon E5-2683 (28 core)

GPU — GTX1080Ti

GPU(2) — GTX980

Evaluation Performance

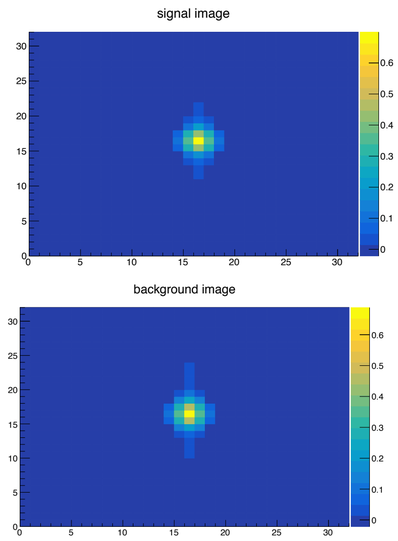
- **Single event evaluation time** for 5 layer network
  - For time critical applications — e.g. on-line reconstruction
  - Fast! 1.5 times speedup over specialised libraries like LWTNN when using optimised Blas library exploiting vectorisation
- For batched evaluation, same story as training

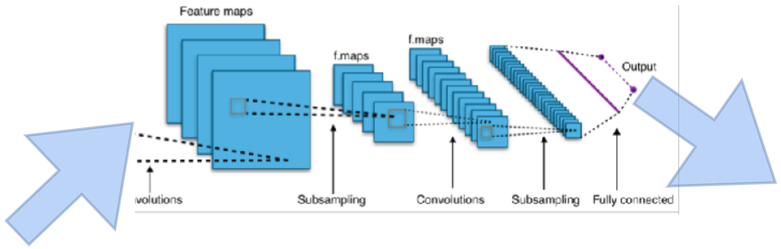


Convolutional Neural Network

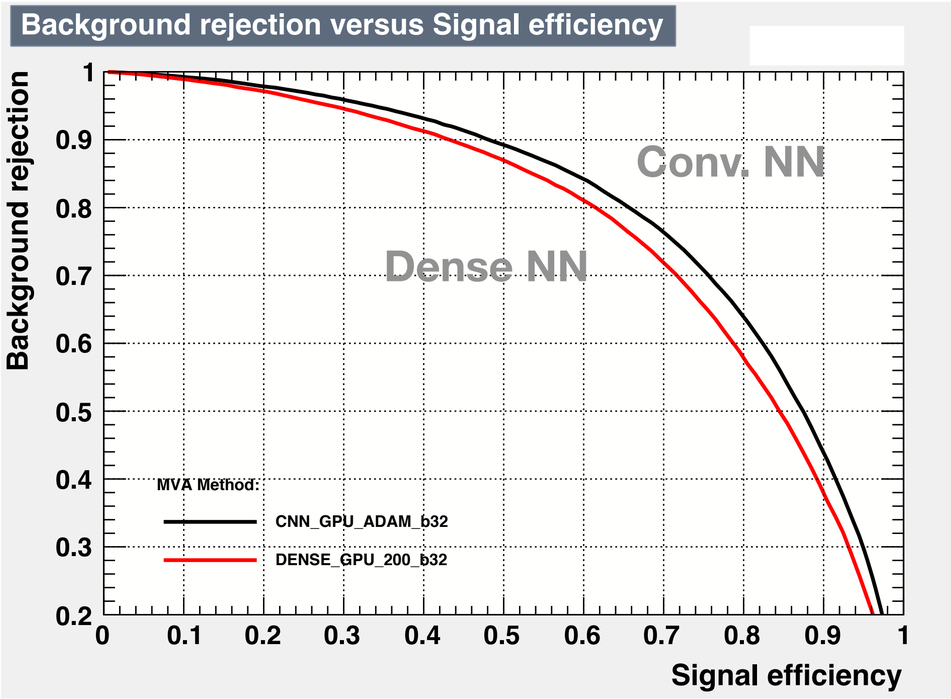
- Available in latest ROOT version (6.14)
- Supporting CPU parallelization, GPU is now also available
  - parallelisation and code optimisation is essential
- Image dataset from simulated particle showers from an electromagnetic calorimeter
  - distinguish electron from photon showers

input 32x32 images





Convolutional + Pooling + Dense layers

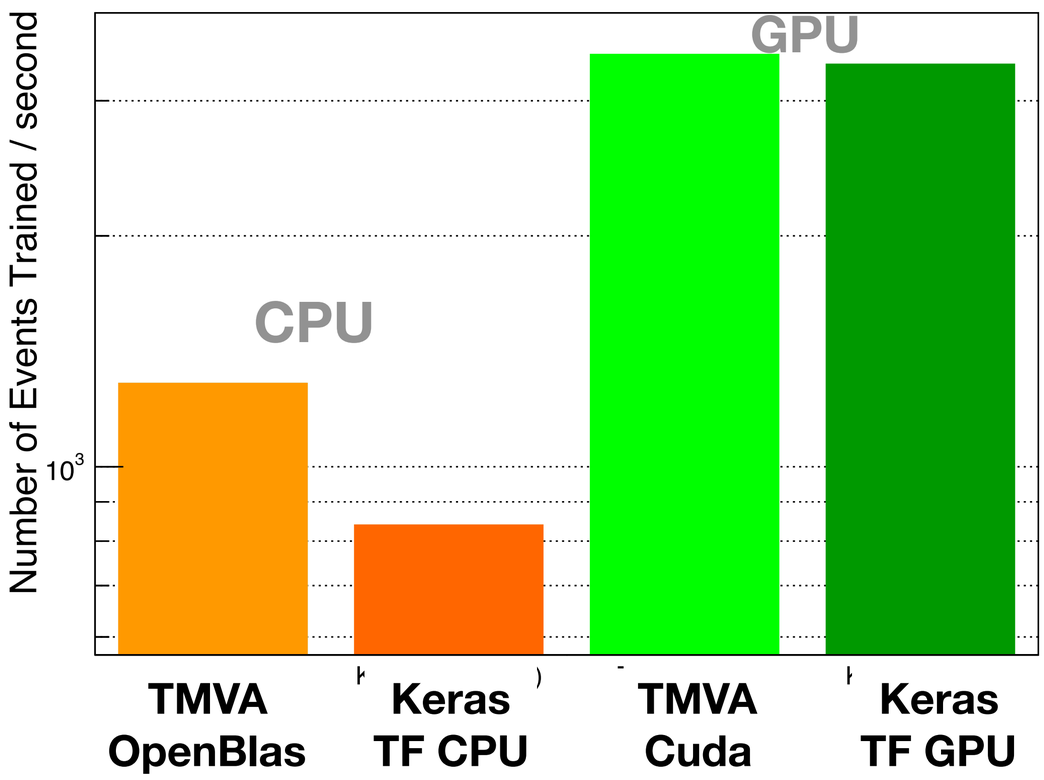


CNN Training Performance

CNN performance for TMVA CPU and GPU

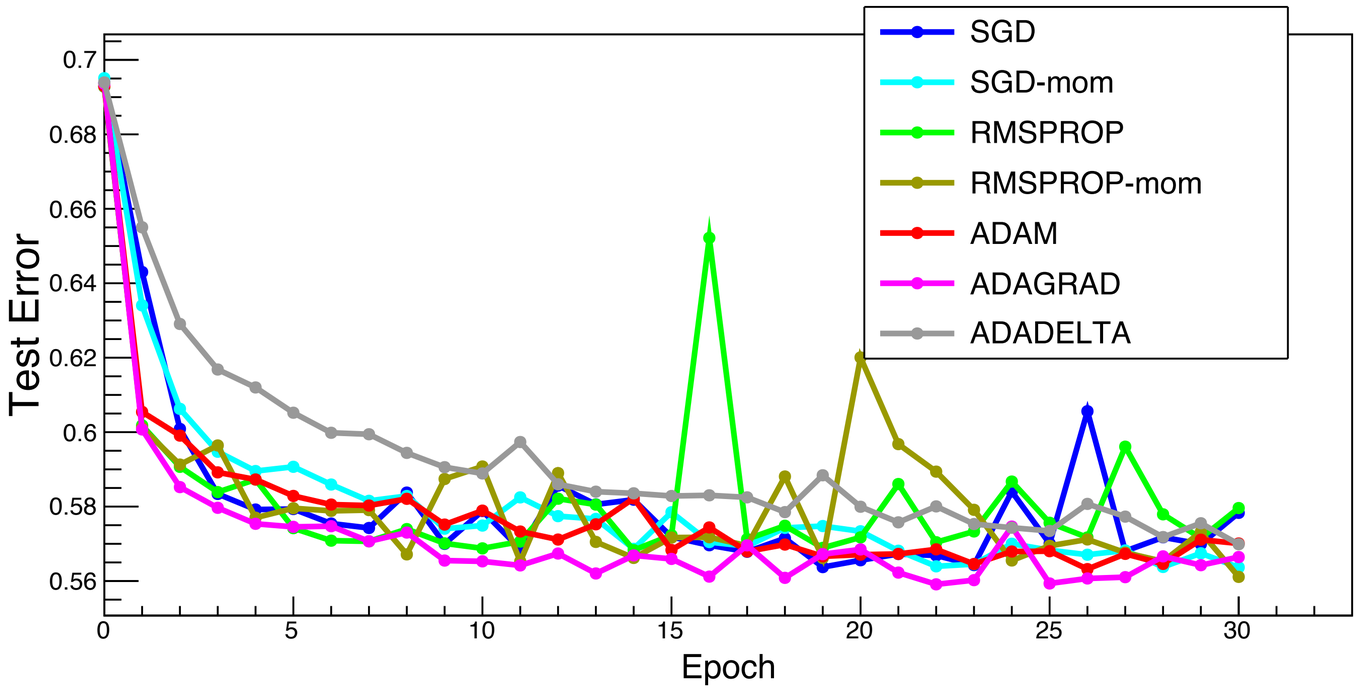
- Simulated particle showers from electromagnetic calorimeter image dataset
- TMVA GPU is now available in ROOT master
- again excellent TMVA performance for typical HEP networks !
- Code run already at same speed as Keras/Tensorflow on small/medium batch sizes
  - further optimisations are possible

4 Convolutional layers, Batch size = 32



New Deep Learning Optimizers

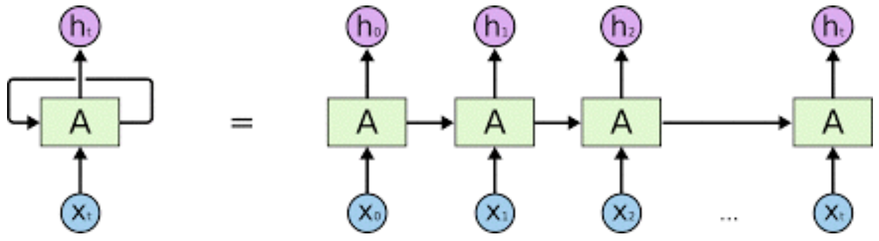
- Integrated in TMVA master new deep learning optimisers
  - In addition to SGD (Stochastic Gradient Descent) added
    - support acceleration using momentum
    - ADAM (new default)
    - ADADelta
    - ADAGrad
    - RMSProp



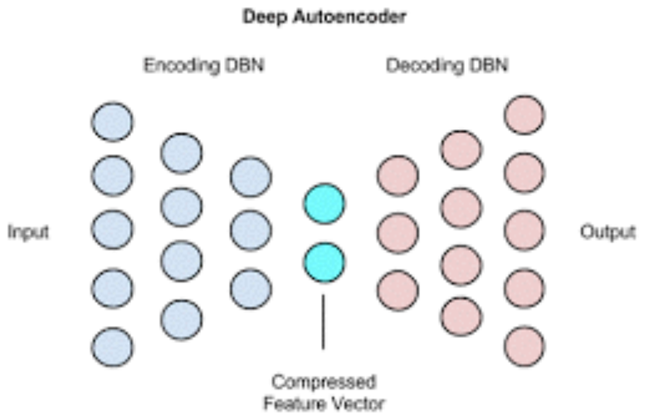
With these new optimisers need less epochs (iterations) to converge !

Other Deep Learning Developments

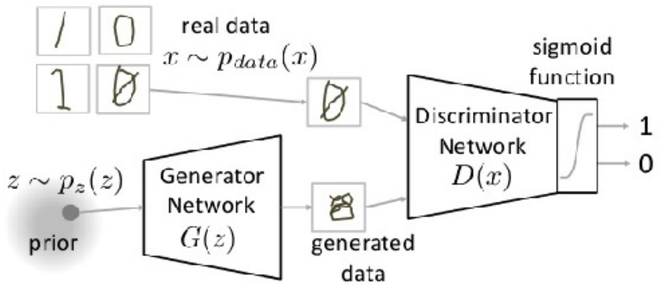
- **Recurrent Neural Network**
  - useful for time-dependent data
  - first version available in 6.14
  - extending to support LSTM layer



- **Deep Auto Encoder**
  - dimensionality reduction (pre-processing tool)
  - unsupervised tool (e.g. for anomaly detection)
  - usable also for generating models, Variational Auto Encoder (VAE)



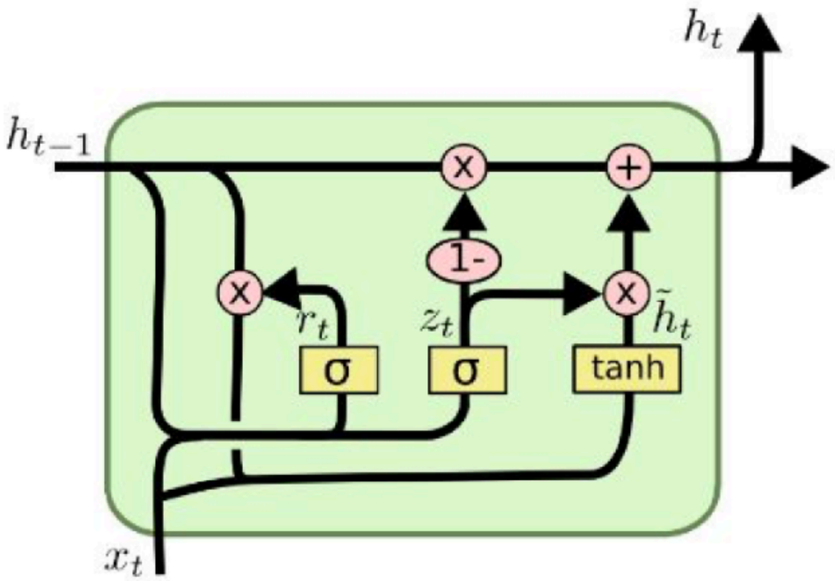
- **Generative Adversarial Network (GAN)**
  - model generation tool (fast simulation)



GSOC projects during this summer for these new developments

LSTM

- RNN suffers from problem to preserve long recurrence memory vs to the short ones
  - gradient may explode or vanish due to recursive relations (e.g. $\mathbf{s}^t = \mathbf{W}\,\mathbf{s}^{t-1} \rightarrow \mathbf{s}^t = \mathbf{W}^t\mathbf{s}^1$)
- LSTM cells is a modified RNN cells introducing gates to prevent this problem and preserving better long term memories



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

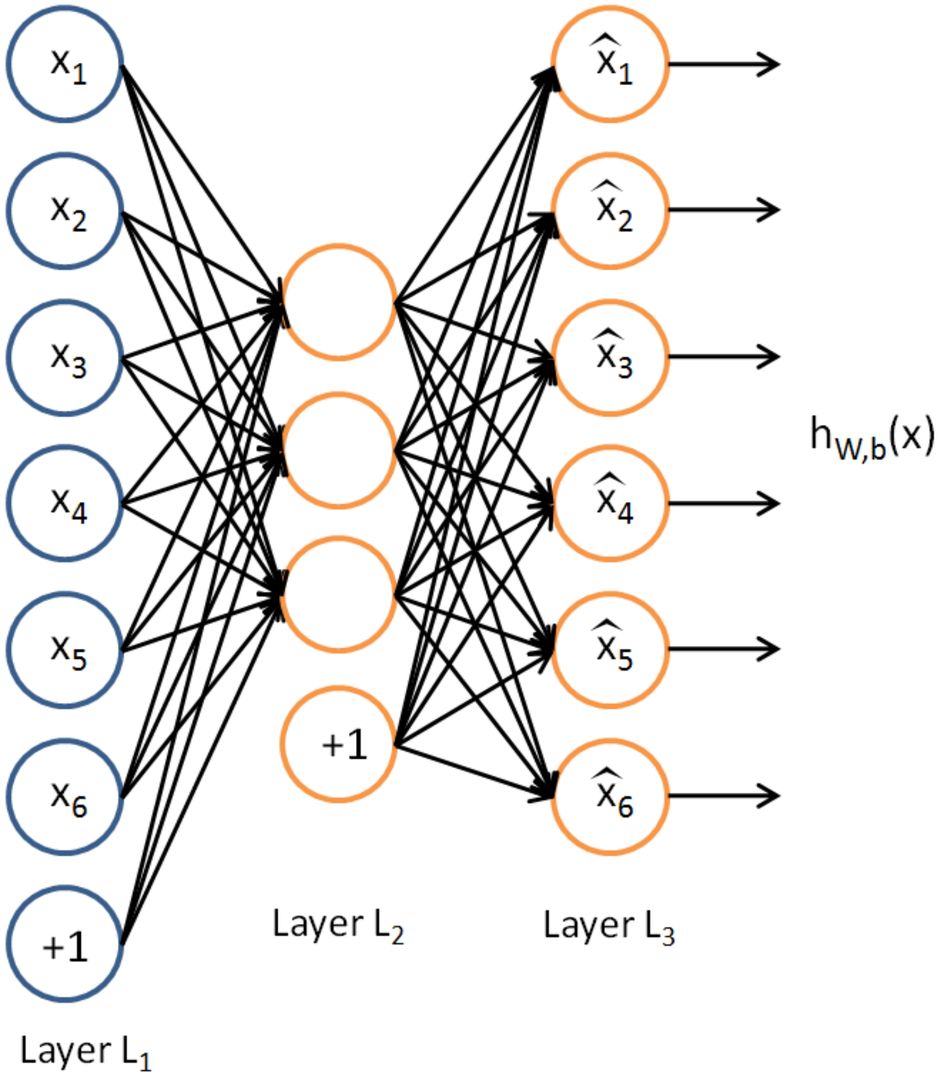
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

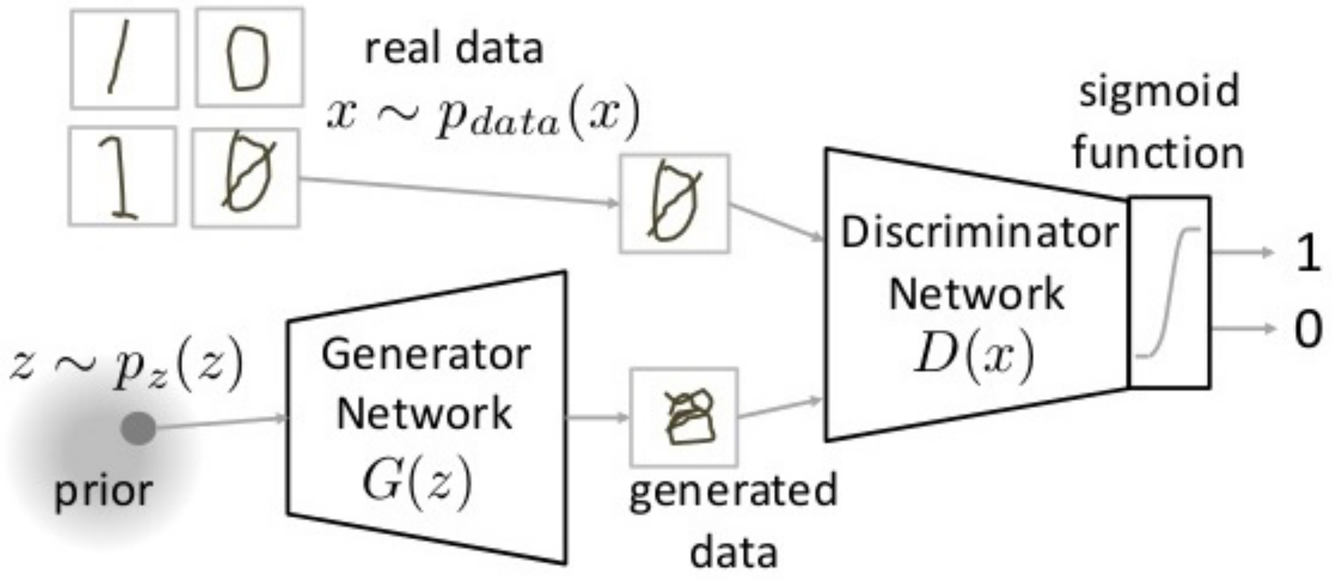
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Deep Autoencoder

- An unsupervised neural network
- Trained by setting the target values y_i equal to the inputs x_i
- Can be used for **dimensionality reduction** or **anomaly detection**
  - and as a generator (**variational auto-encoders**)



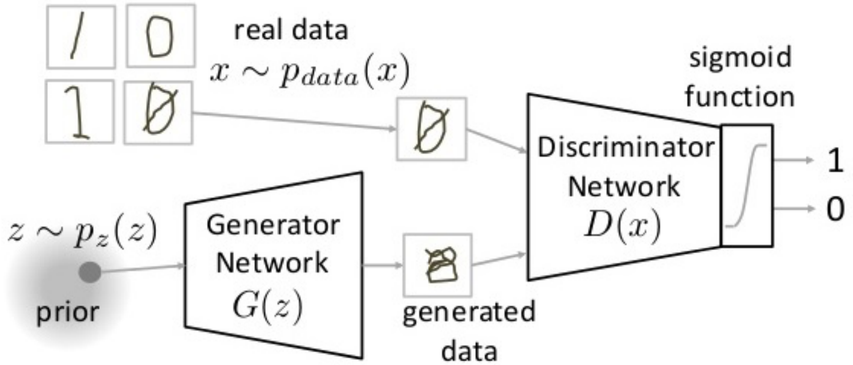
GAN: Generative Adversarial



- **Generator network**:
  - output data from a random input G(x)
- **Discriminator network**:
  - discriminate the generated data from real ones
  - output probability D(x) that data are from real input

GAN Optimization

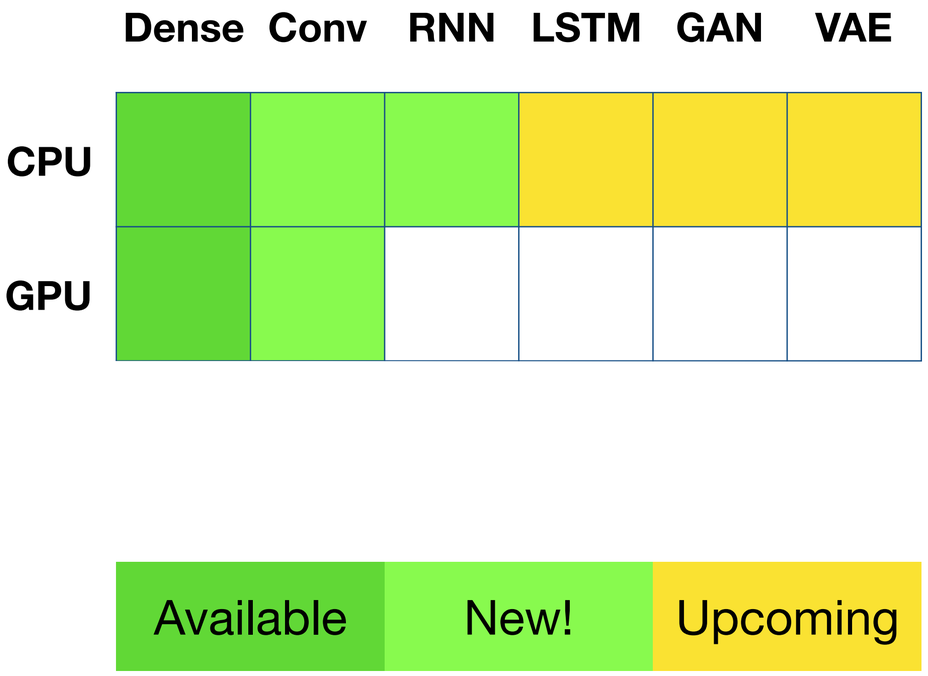
- Want to find discriminator parameters such that data coming from training sample and real one are as similar as possible
- Find generator parameters that make random (fake) generated data unlikely
  - classified by the discriminator as fake.
  - minimize for G cross-entropy log(1 - D(G(z))
- **Optimization of a GAN is then a min-max player game**



$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Summary Deep Learning

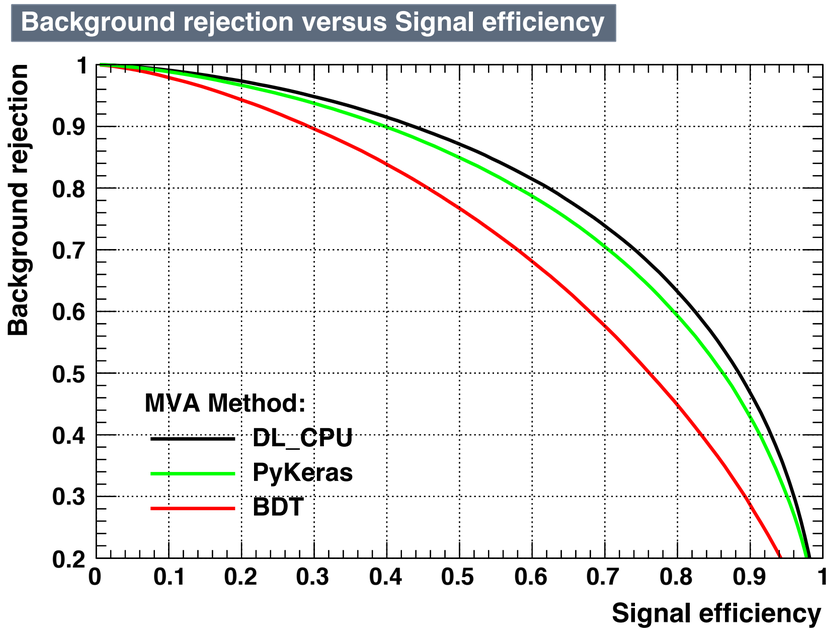
- Recent additions
  - Convolutional and recurrent layers
  - new optimisers complementing SGD
- Development ongoing!
  - LSTM (and also GRU) layers
  - GAN and VAE for event generation



TMVA Interfaces

External tools are available as additional methods in TMVA and they can be trained and evaluated as any other internal ones.

- **RMVA: Interface to ML methods in R**
  - c50, xgboost, RSNNS, e1071
- **PYMVA: Interface to Python ML**
  - **scikit-learn**
    - with RandomForest, Gradiend Tree Boost, Ada Boost
  - **Keras** (Theano + Tensorflow)
    - support model definition in Python and then training and evaluation in TMVA
  - working on direct mapping from ROOT tree to Numpy arrays
    - see Stefan's presentation



Example PyMVA with Keras

Define the Keras model in Python

Define model for Keras

In [5]:
```
# Define model
model = Sequential()
model.add(Dense(32, init='glorot_normal', activation='relu',
        input_dim=numVariables))
model.add(Dropout(0.5))
model.add(Dense(32, init='glorot_normal', activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2, init='glorot_uniform', activation='softmax'))

# Set loss and optimizer
model.compile(loss='categorical_crossentropy', optimizer=Adam(),
        metrics=['categorical_accuracy',])

# Store model to file
model.save('model.h5')

# Print summary of model
model.summary()
```

Book the method as any others of TMVA

Book methods

Just run the cells that contain the classifiers you want to try.

In [6]:
```
# Keras interface with previously defined model
factory.BookMethod(dataloader, ROOT.TMVA.Types.kPyKeras, 'PyKeras',
        'H:!V:VarTransform=G:FilenameModel=model.h5:'+\
        'NumEpochs=10:BatchSize=32:'+\
        'TriesEarlyStopping=3')
```

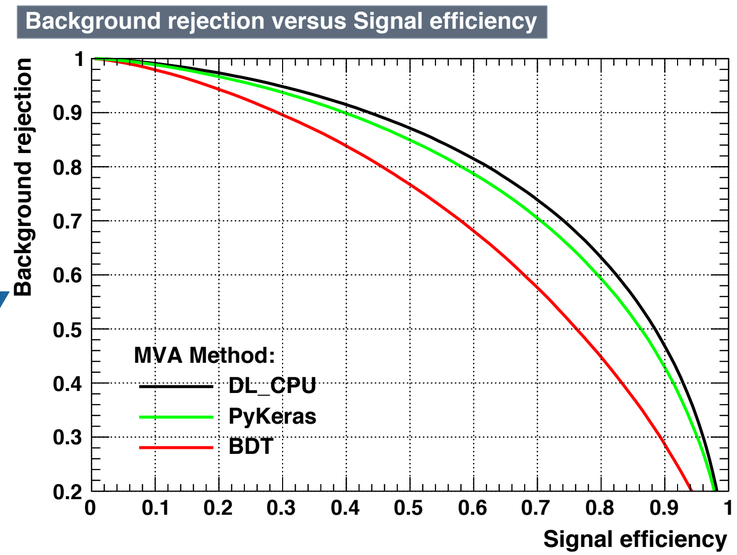
Out[6]:
```
<ROOT.TMVA::MethodPyKeras object ("PyKeras") at 0x77e48b0>
```

PyMVA with Keras

Train,Test and Evaluate inside TMVA (using TMVA::Factory)

Run training, testing and evaluation

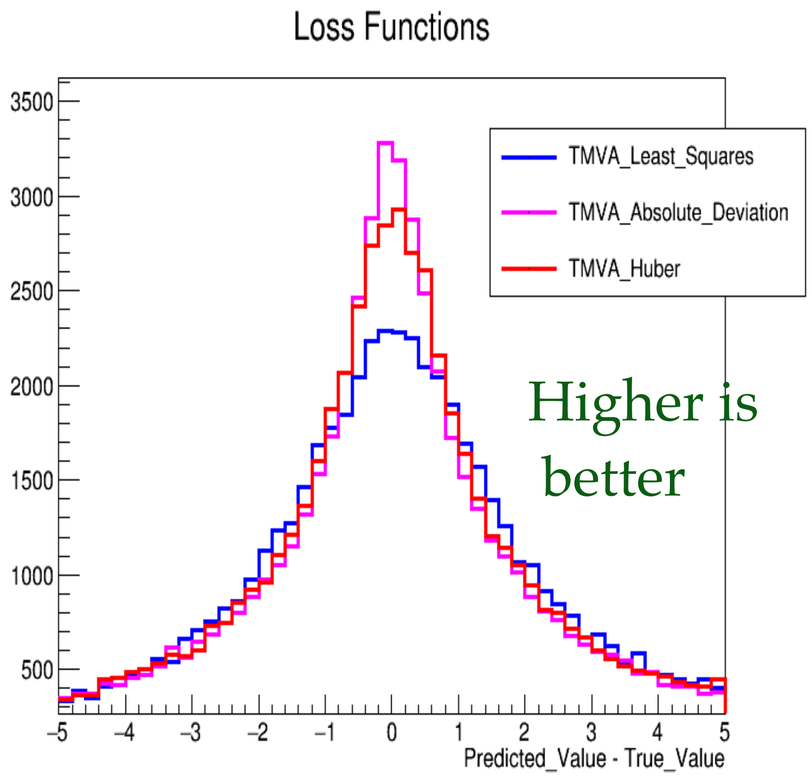
```
In [8]: factory.TrainAllMethods()

        Factory                  : Train all methods

In [9]: factory.TestAllMethods()

        Factory                  : Test all methods
        Factory                  : Test method: PyKeras
                                 :

In [10]: factory.EvaluateAllMethods()

        Factory                  : Evaluate all methods
        Factory                  : Evaluate classifier: PyKeras
                                 :
```



Examine result with TMVA GUI

Regression in TMVA

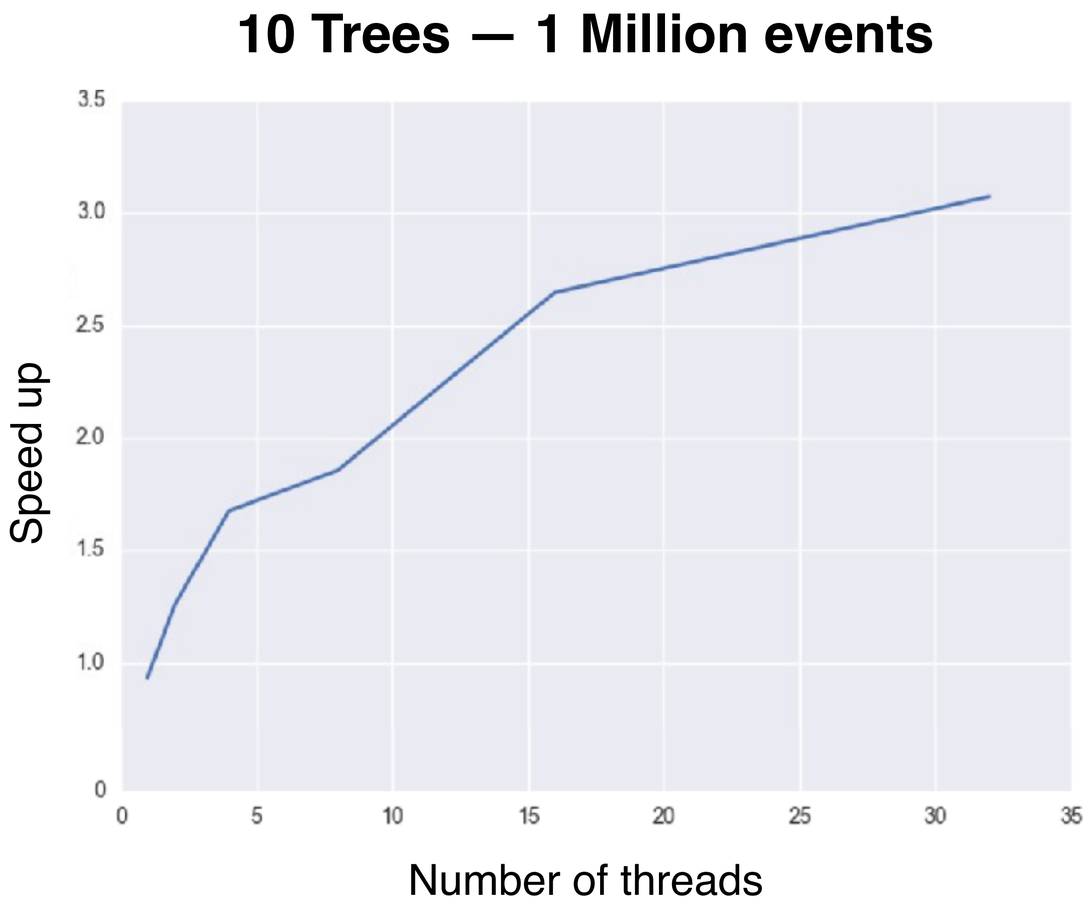
- **New Regression Features**:
  - Loss function
    - Huber (default)
    - Least Squares
    - Absolute Deviation
    - Custom Function



Important for regression performance

Boosted Decision Tree

- Boosting is serial ⟶ Can't construct all trees in parallel
- Training time speed up ~1.6x with 4 threads approaching ~3x asymptotically
- To use, just add ROOT::EnableImplicitMT() to your code



Original slide by Andrew Carnes

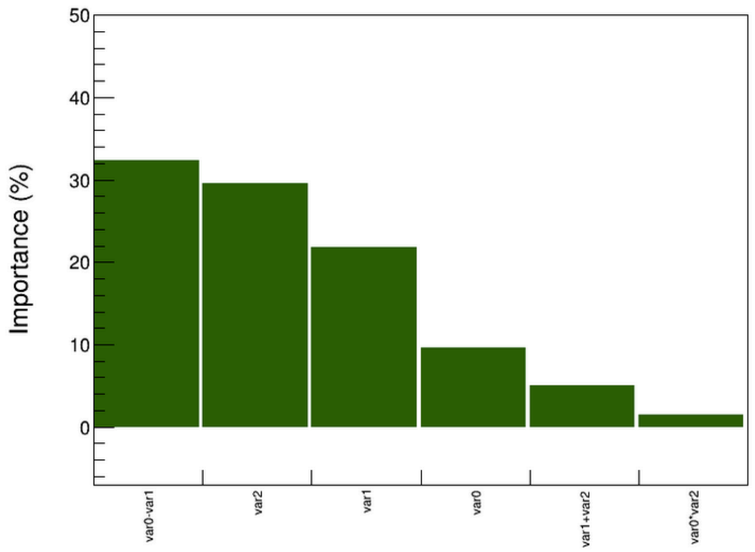
Feature Importance

- Ranks the importance of features based on contribution to classifier performance
  - A stochastic algorithm independent of classifier choice

$$FI(X_i) = \sum_{S \subseteq V : X_i \in S} F(S) \times W_{X_i}(S)$$

$$W_{X_i}(S) \equiv 1 - \frac{F(S - \{X_i\})}{F(S)}$$

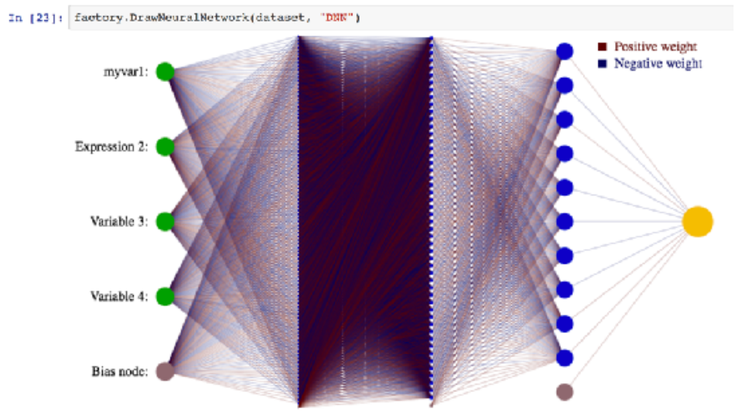
- Feature set {V}
- Feature subset {S}
- Classifier Performance F(S)

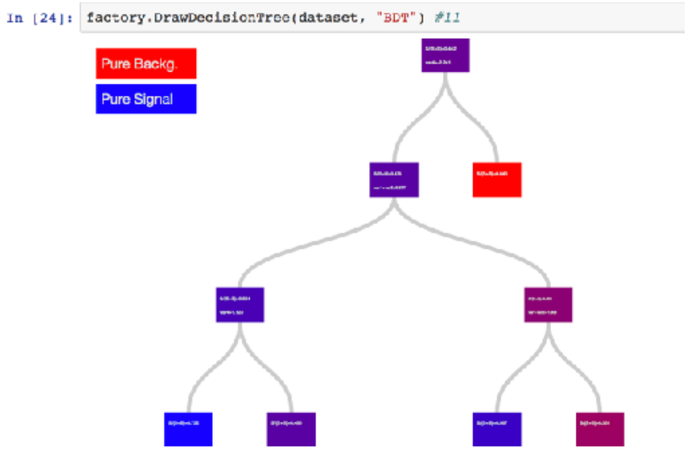


TMVA Jupyter Integration

New Python package for using TMVA in Jupyter notebook (jsmva)

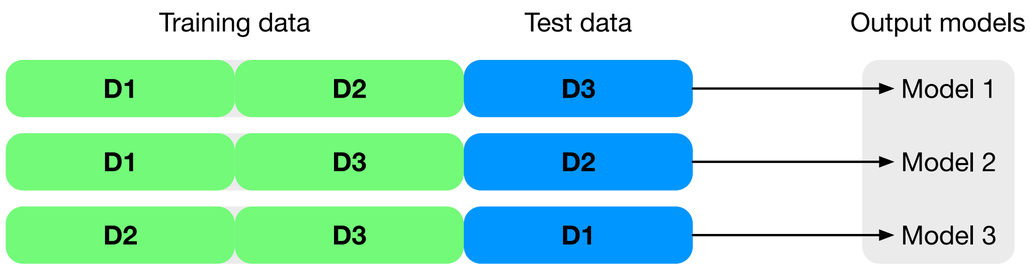
- Improved Python API for TMVA functions
- Visualisation of BDT and DNN
- Enhanced output and plots (e.g. ROC plots)
- Improved interactivity (e.g. pause/resume/stop of training)
- see example in SWAN gallery
  https://swan.web.cern.ch/content/machine-learning

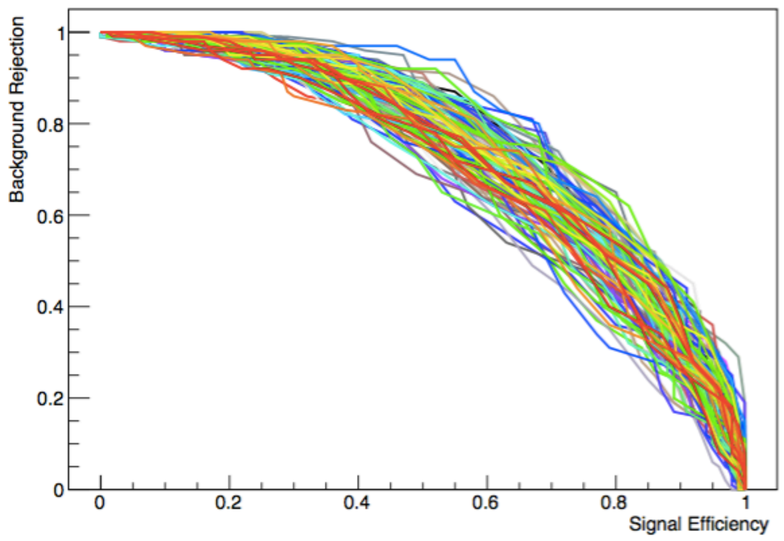




Cross Validation in TMVA

- TMVA supports k-fold cross-validation





  - Integration with TMVA analysis tools (e.g. GUI)
- Hyper-parameter tuning
  - find optimised parameters (BDT-SVM)
- Parallel execution of folds in CV
  - using multi-processes execution in on a single node
  - foreseen to provide parallelisation in a cluster using Spark or MPI

Future Developments

- Our aim is to provide to the users community efficient physics workflows
  - tools for efficient
    - data loading (using new RDataFrame)
    - integration with external ML tools
    - training of commonly used architectures
    - deployment and inference of trained models
- TMVA efficiently connects input data to ML algorithms
  - we are defining new user interfaces

Building ML models

Build model

- Solid baseline of ML methods
- Integration of (cutting-edge) external ML packages
- Mix-and-match between packages

- **ML baseline:** Methods of current TMVA
- **Key points:**
  - Modern interface
  - Modularity
  - Interoperability with numpy ≡ Interoperability with external ML packages

```
import ROOT
import numpy as np

# Read a ROOT file
df = ROOT.RDataFrame("tree", "file.root")

# Access data as numpy arrays and build training dataset
x_sig = df.Filter("a>b && c!=d").AsNumpy()
x_bkg = df.Filter("e+f==g && h==i").AsNumpy()
x = numpy.stack([x_sig, x_bkg])
y = numpy.stack([np.ones(len(x_sig)), np.zeros(len(x_bkg)])

# Build TMVA model
bdt = ROOT.TMVA.BDT(num_trees=500, depth=3)
bdt.Fit(x, y)
bdt.Save("parameters.root")

# Build sklearn model
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x, y)
```

Available in ROOT 6.14

Future

External package

Apply trained ML model

Apply model

- High throughput inference
- Fully accessible from C++
- Plug-and-play for different models

- **Key points:**
  - Fast inference, especially event-by-event
  - Being accessible from C++
  - Loading parameters of externally trained models
  - Interaction with RDataFrame

```
int main() {
  // Load TMVA and models trained with external packages
  auto bdt = ROOT::TMVA::BDT("parameters.root");
  auto nn = ROOT::TMVA::Keras("parameters.h5");

  // Perform single prediction
  vector<float> x = {1.0, 2.0, 3.0, 4.0};
  vector<float> y = bdt.Predict(x);

  // Append method responses to a ROOT dataframe
  auto df = ROOT::RDataFrame("events", "some_file.root");

  vector<string> vars = {"var1", "var2", "var3", "var4"};
  auto df_response = df.Define("response_bdt", bdt, vars)
                       .Define("response_nn", nn, vars);

  // Analyze the result
  auto h_bdt = df_response.Filter("response_bdt>0.5")
                          .Histo1D("mass");
  auto h_nn = df_response.Filter("response_nn>0.5")
                         .Histo1D("mass");
  h_bdt.Draw("histo");
  n_nn.Draw("same");
}
```

Future

Summary

Machine learning methods

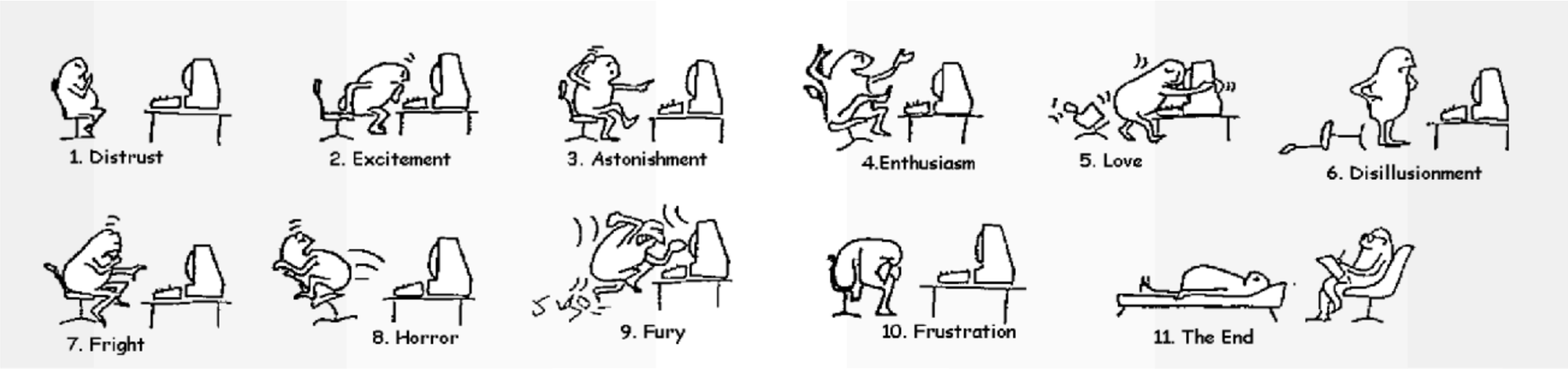
- Dense, Convolutional and Recurrent networks in TMVA
- Excellent training + evaluation time performance
- Training in parallel Boosted Decision Trees

Workflow improvements

- Cross validation analysis and parallelisation

Future:

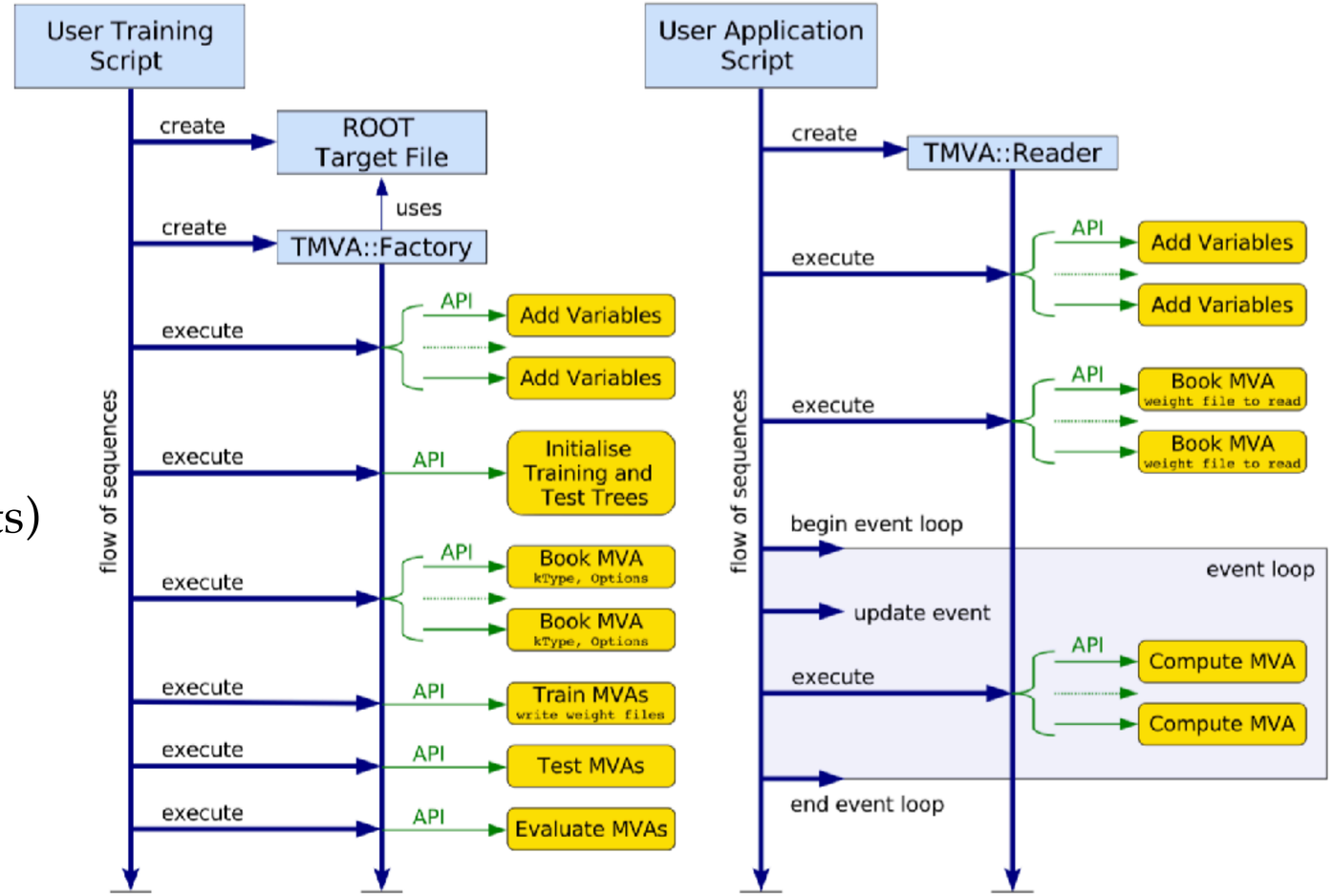
- Efficient physics workflows connecting input data to algorithms
  - integration with new RDataFrame and mapping to Numpy
  - fast deployment and inference of trained models


1. Distrust
2. Excitement
3. Astonishment
4.Enthusiasm
5. Love
6. Disillusionment
7. Fright
8. Horror
9. Fury
10. Frustration
11. The End

Using TMVA

TMVA Workflow

- Reading input data
- Select input features and preprocessing
- **Training**
  - find optimal classification or regression parameters using data with known labels (e.g. signal and background MC events)
- **Testing**
  - evaluate performance of the classifier in an independent test sample
  - compare different methods
- **Application**
  - apply classifier/regressor to real data where labels are not known



TMVA Workflow Features

TMVA supports:

- input data from ROOT Trees or ASCII data (e.g. csv)
- pre-selection cuts on input data
- event weights (negative weights for some methods)
- various method for splitting training/test samples
- k-fold cross-validation and hyper-parameter optimisation
- algorithm to identify importance of input variables
- GUI for output evaluation and analysis

TMVA Session

- Create Factory
- Create DataLoader class
- Add variables/target using the DataLoader
- Initialize input Trees
- Book MVA methods
- Train/Test/Evaluate using
- Save output and train methods

We will see better with a real example
(e.g. TMVAClassification.C tutorial)

TMVA DataLoader

- DataLoader is a new class that allows greater flexibility when working with datasets. It is an interface to
  - load the datasets
    - root files (TTrees) but can be extended to other types
  - add variables

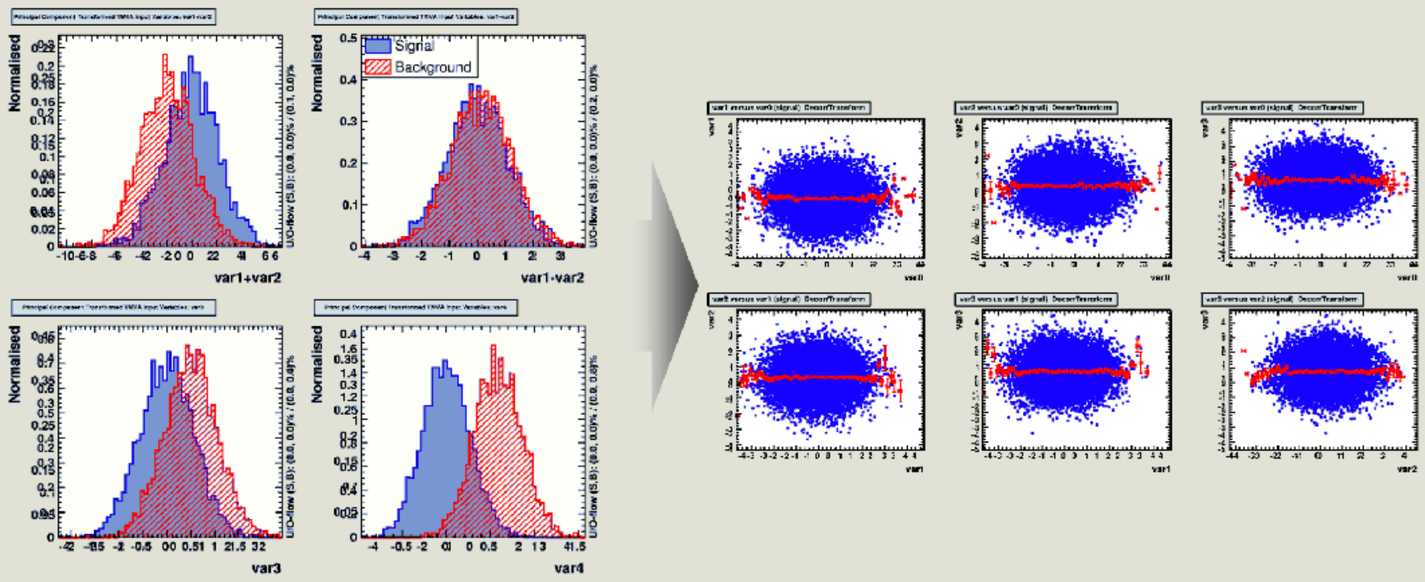
- TMVA Factory links DataLoader with a specific MVA method when booking

```
factory->BookMethod( DataLoader *loader, Types::EMVA theMethod,
                     const char * methodTitle, const char *option = "" );
```

- Obtained desired flexibility in de-coupling methods/dataset/variables

Pre-processing of the Input

- Example: decorrelation of variable before training can be useful



- Note that in cases with non-Gaussian distributions and/or nonlinear correlations decorrelation may do more harm than any good

Several others pre-processing available (see Users Guide)

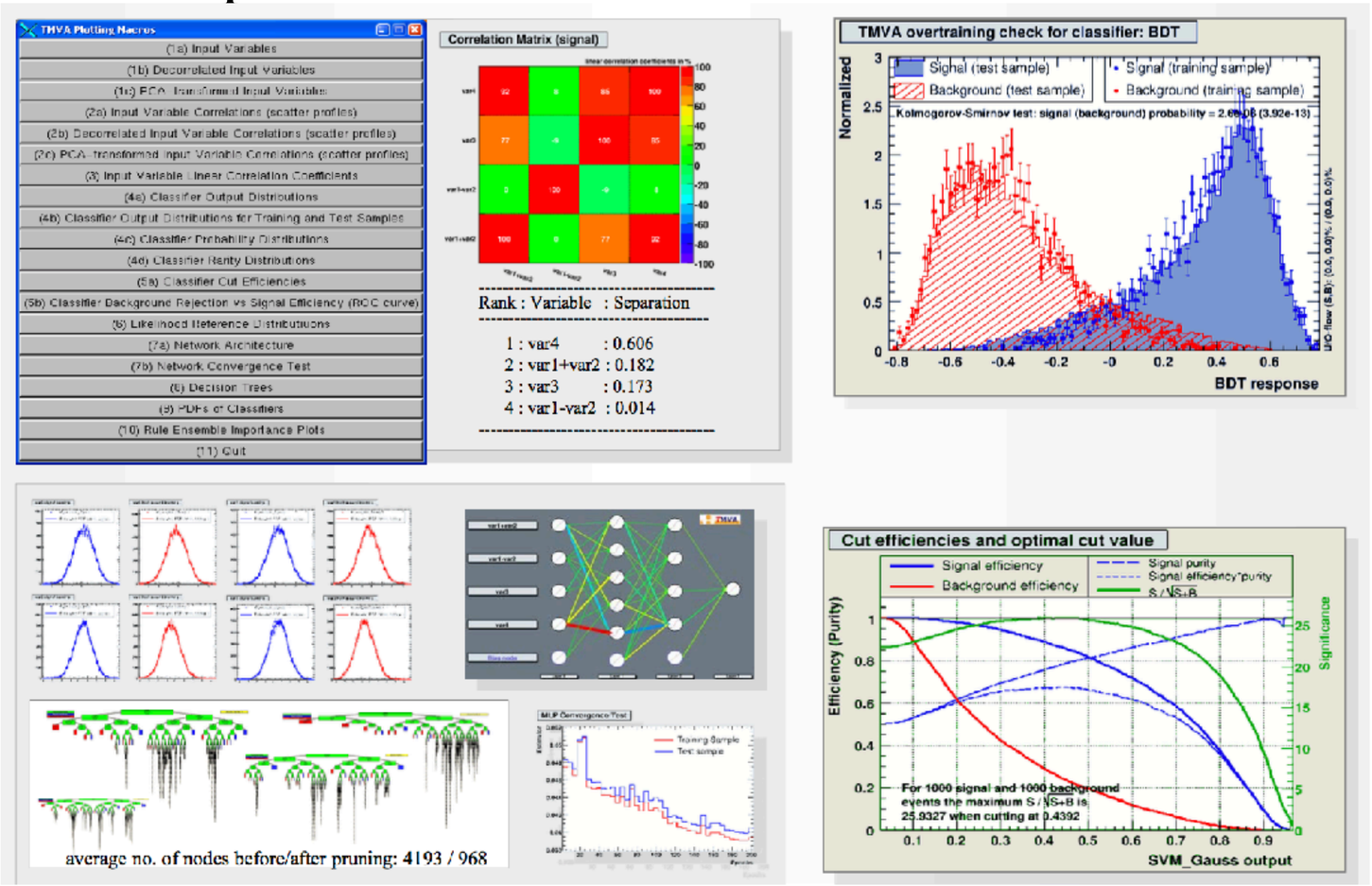
Available Preprocessing

- List of available pre-processing in TMVA
  - Normalization
  - Decorrelation (using Cholesky decomposition)
  - Principal Component Analysis
  - Uniformization
  - Gaussianization

- Can be selected individually for each single method (when booking)

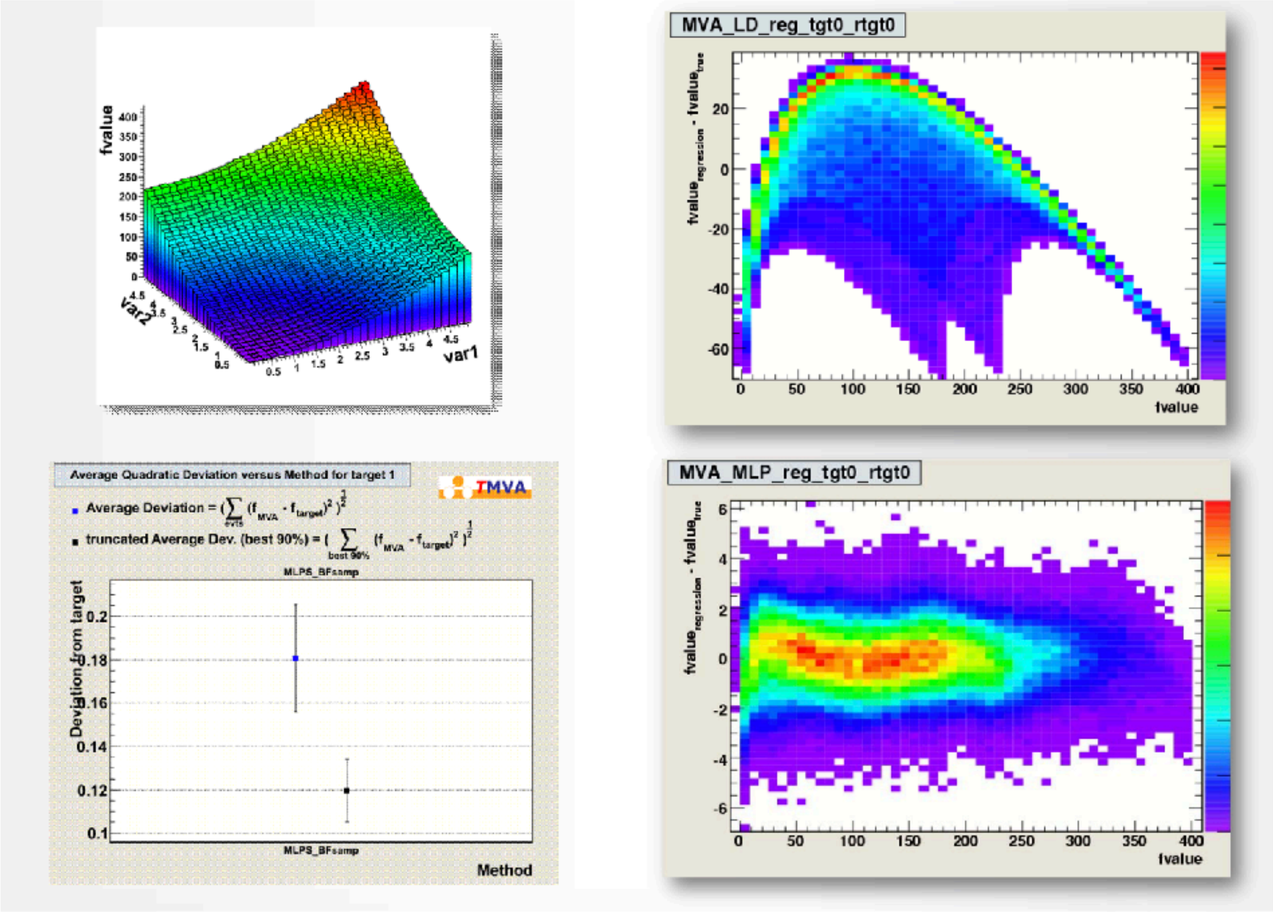
TMVA GUI

At the end of training + test phase, TMVA produces an output file that can be examined with a special GUI (**TMVAGui**)



TMVA Regression GUI

A dedicated GUI exists for regression (**TMVARegGui**)



SWAN: Data Analysis as a Service

- *Interface:*
  - Jupyter Notebooks
- *Goals:*
  - Analysis only with a web browser
    - Platform independent ROOT-based data analysis
    - Calculations, input and results “in the Cloud”
  - Easy sharing of scientific results: plots, data, code
    - Storage is crucial: mass & synchronised
  - Integration with other analysis ecosystems: R, Python, ...

Interface: The Notebook

Text

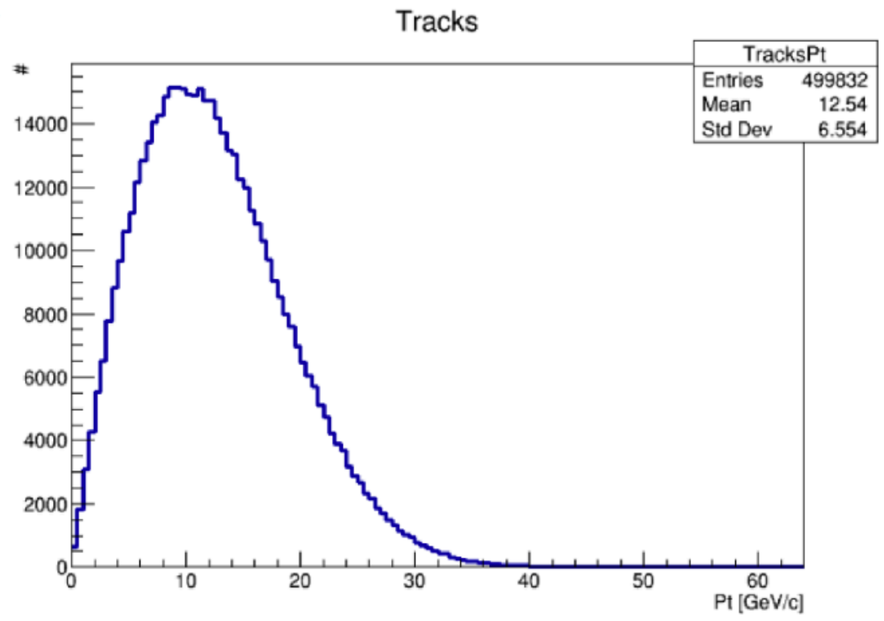
Code

Graphics

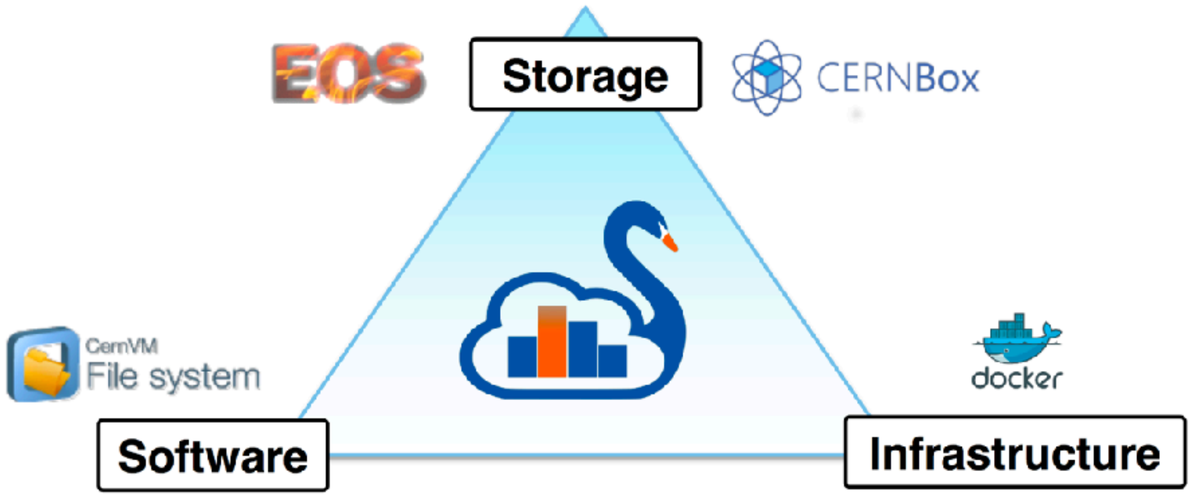
Access TTree in Python using PyROOT and fill a histogram

Loop over the TTree called "events" in a file located on the web. The tree is accessed with the dot operator. Same holds for the access to the branches: no need to set them up - they are just accessed by name, again with the dot operator.

In [1]:
```
import ROOT

f = ROOT.TFile.Open("http://indico.cern.ch/event/395198/material/0/0.root");
h = ROOT.TH1F("TracksPt","Tracks;Pt [GeV/c];#",128,0,64)
for event in f.events:
    for track in event.tracks:
        h.Fill(track.Pt())
c = ROOT.TCanvas()
h.Draw()
c.Draw()
```



SWAN Building Blocks

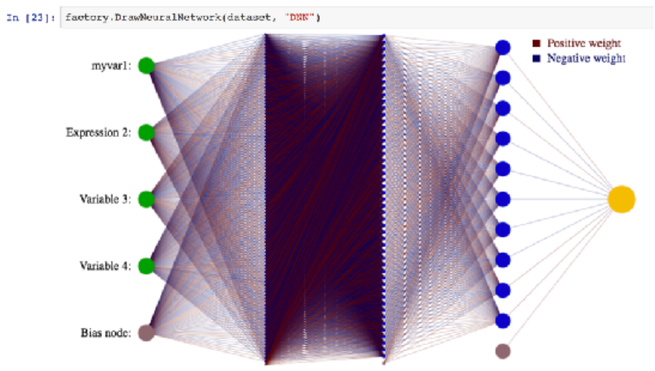


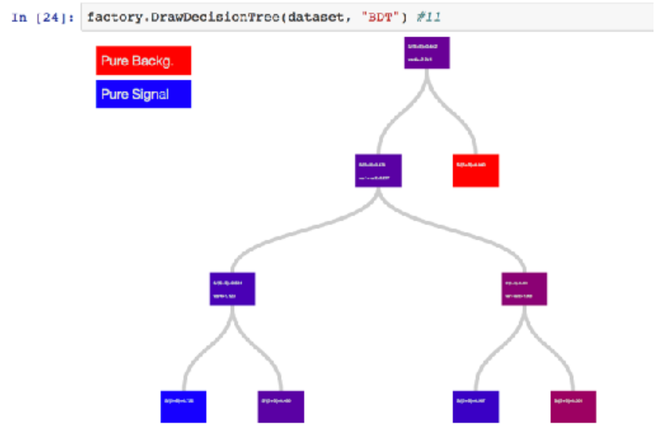
- Docker: single thin image, not managed by the user!
- CVMFS: configurable environment via “views”
  - CERN and LCG software
- CERNBox: custom user software environment

TMVA Jupyter Integration

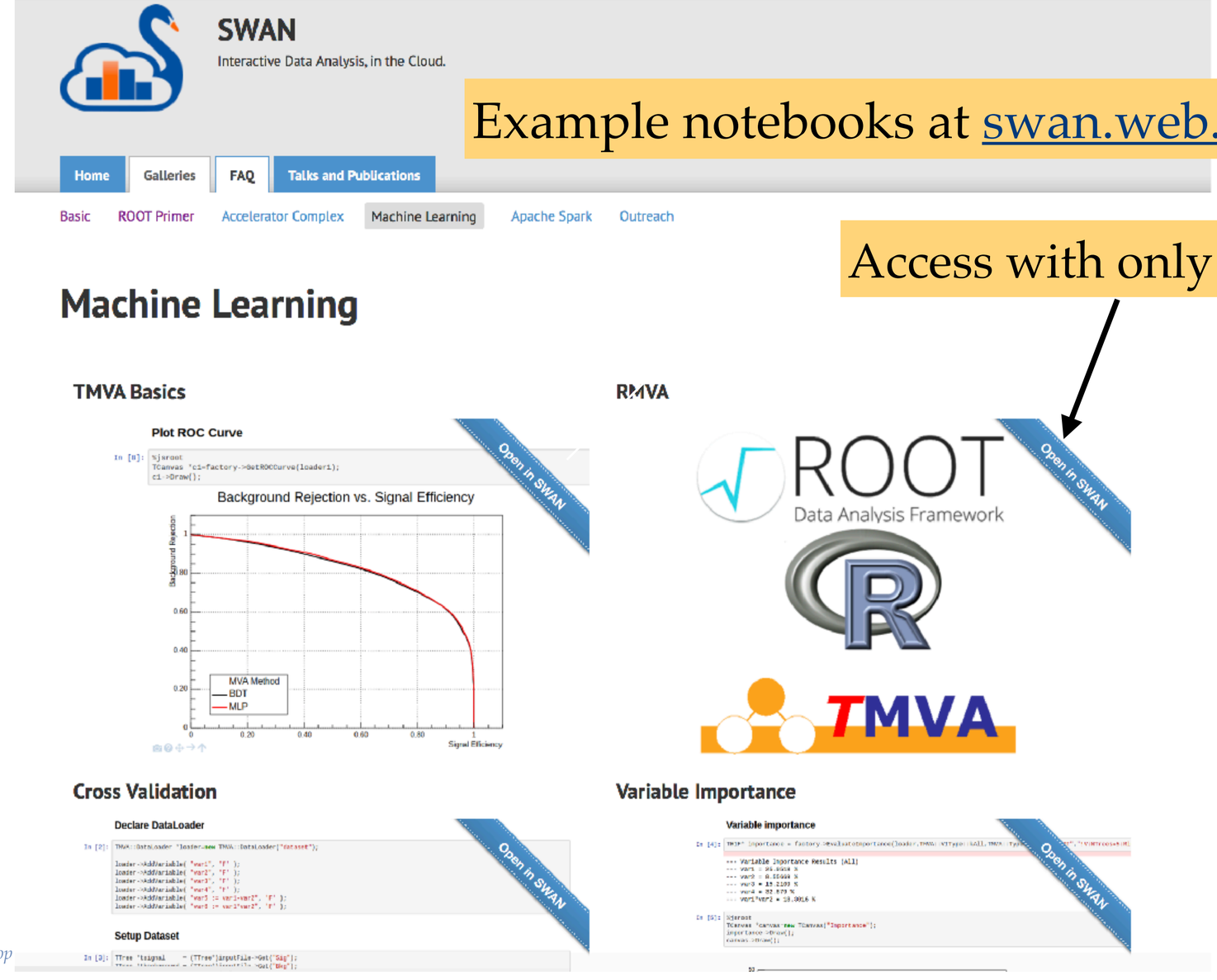
New Python package for using TMVA in Jupyter notebook (jsmva)

- Improved Python API for TMVA functions
- Visualisation of BDT and DNN
- Enhanced output and plots (e.g. ROC plots)
- Improved interactivity (e.g. pause/resume/stop of training)
- see example in SWAN gallery
  https://swan.web.cern.ch/content/machine-learning



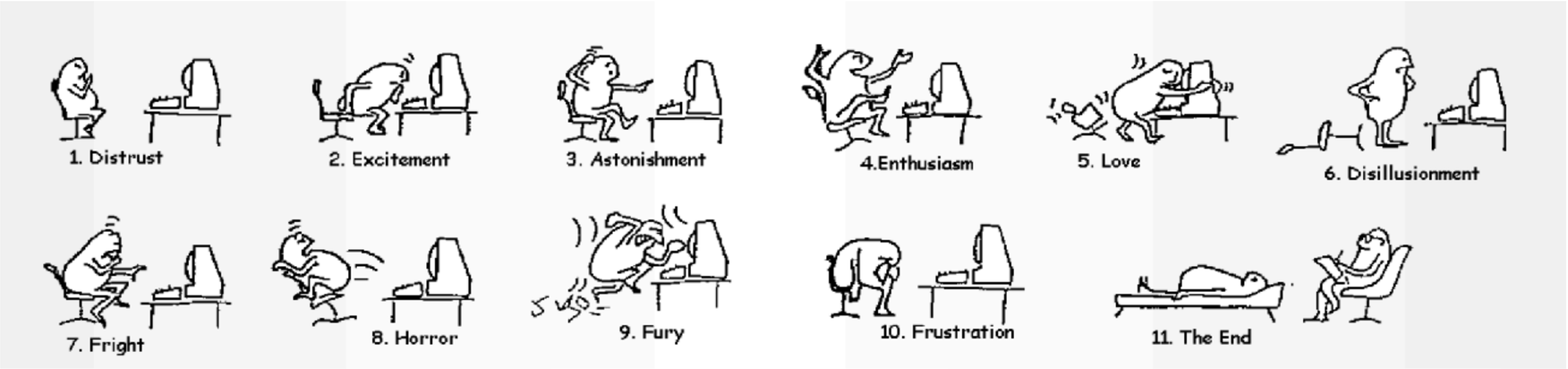


SWAN ML Gallery



Example notebooks at swan.web.cern.ch

Access with only a click



Let's start using TMVA

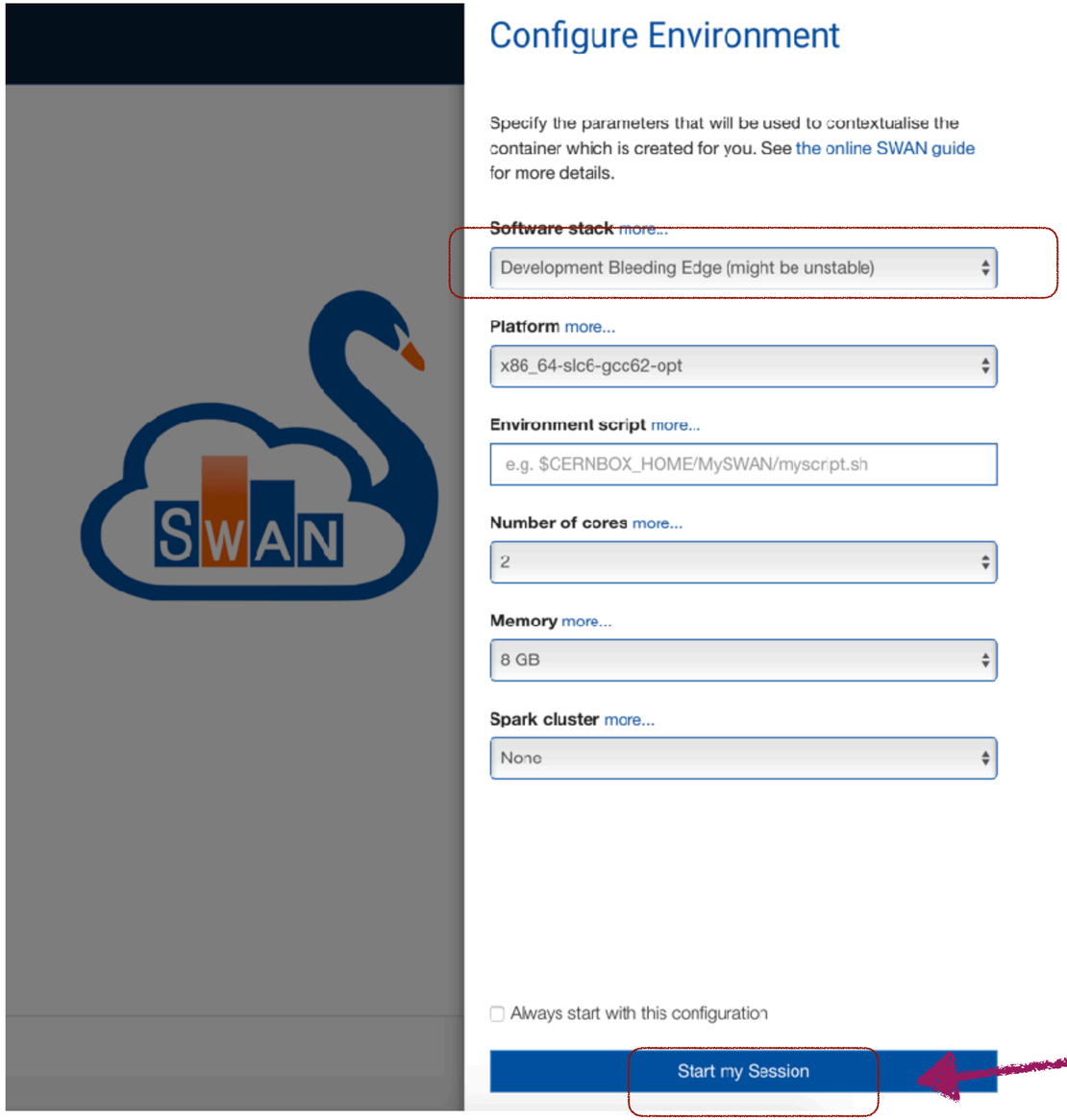
TMVA Tutorial

- **Run tutorial on notebook**
  - use **SWAN**
    - go to swan.cern.ch

  - **or running local notebooks**
    - root —notebook

If you don't have CERN account for using SWAN please contact me
Some temporary account can be made available
But before please feel the online form available **here**

Starting SWAN

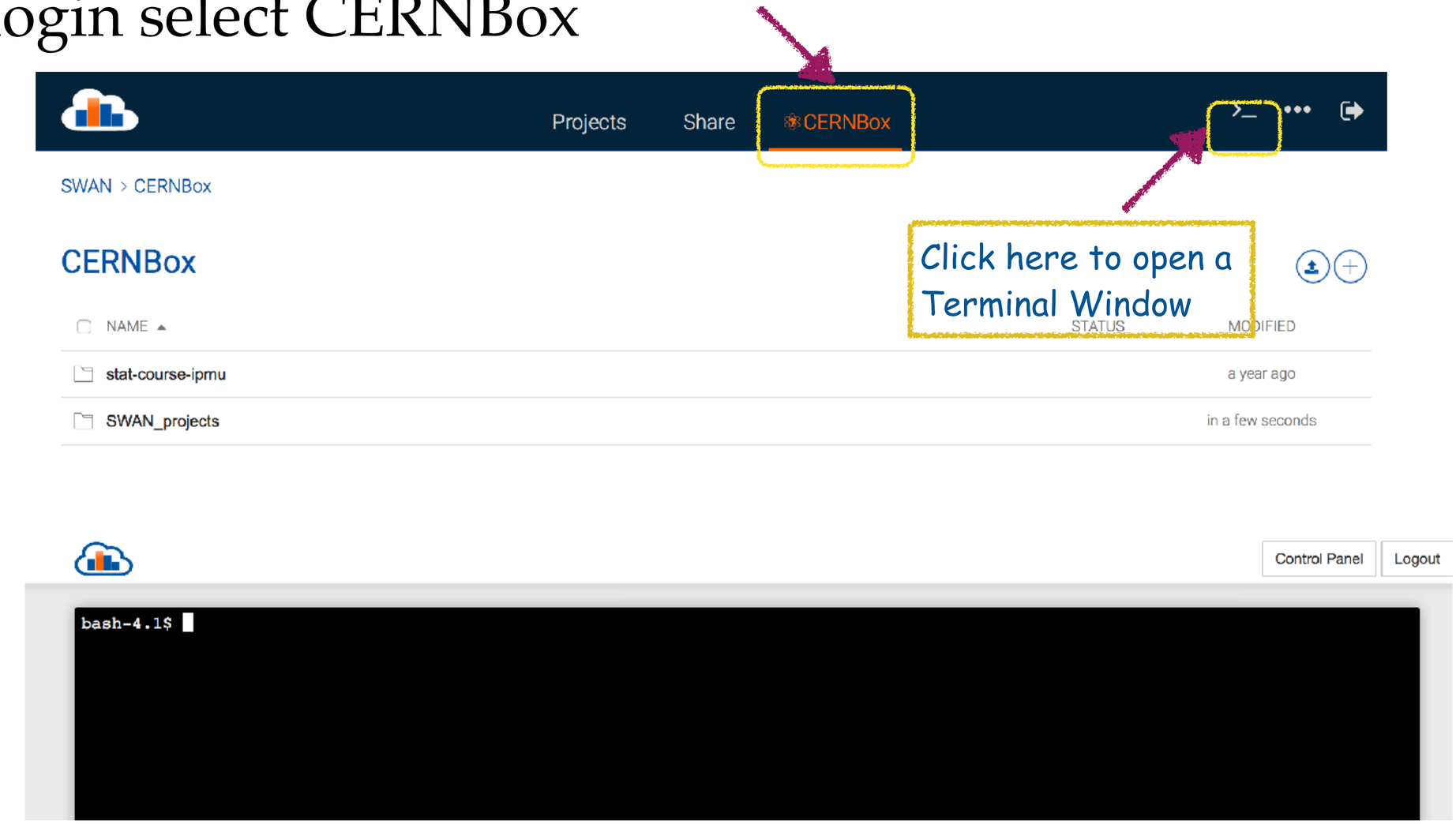


Select to use new Deep Learning

Click here to start

Starting a Terminal in SWAN

After login select CERNBox



Getting the Notebooks

- **Clone the git repository of the tutorials**
  https://github.com/lmoneta/tmva-tutorial.git
  - **git clone** https://github.com/lmoneta/tmva-tutorial.git

If directory already exists delete it before or update its git repository doing:
git fetch; git checkout -b IML-tutorial-2018 origin/IML-tutorial-2018; git pull

```
bash-4.1$ git clone  https://github.com/lmoneta/tmva-tutorial.git
Initialized empty Git repository in /eos/user/s/statlr11/tmva-tutorial/.git/
remote: Counting objects: 144, done.
remote: Compressing objects: 100% (67/67), done.
remote: Total 144 (delta 60), reused 77 (delta 35), pack-reused 42
Receiving objects: 100% (144/144), 22.01 MiB | 6.81 MiB/s, done.
Resolving deltas: 100% (74/74), done.
bash-4.1$
bash-4.1$ cd tmva-tutorial/
bash-4.1$
bash-4.1$ cd notebooks
bash-4.1$
bash-4.1$
```

- Go back to SWAN Home page and select the directory **tmva-tutorial/tutorial_INSIGHTS**
  - Start using the notebooks

Notebooks

CERNBox

NAME

- stat-course-ipmu
- SWAN_projects
- 1. tmva-tutorial

tmva-tutorial

NAME

- 2. notebooks
- tutorial_IML2017
- tutorial_Lisbon
- README.md

tutorial_INSIGHTS

NAME

- 3. TMVA_Classification.ipynb
- TMVA_CNN_Classification.ipynb
- Click here to rename, delete, etc.
- TMVA_Higgs_Classification.ipynb
- TMVA_HyperparameterOptim.ipynb
- TMVA_Reader.ipynb
- TMVA_Reader_py.ipynb
- TMVA_Regression.ipynb
- TMVA_VariableImportance.ipynb
- TMVAGuiPlots.ipynb
- TMVAGuiROC.ipynb

TMVA Classification

TMVA_Classification
(autosaved)

FILE EDIT VIEW INSERT CELL KERNEL HELP Not Trusted ROOT C++

Markdown

TMVA Classification Example

Declare Factory

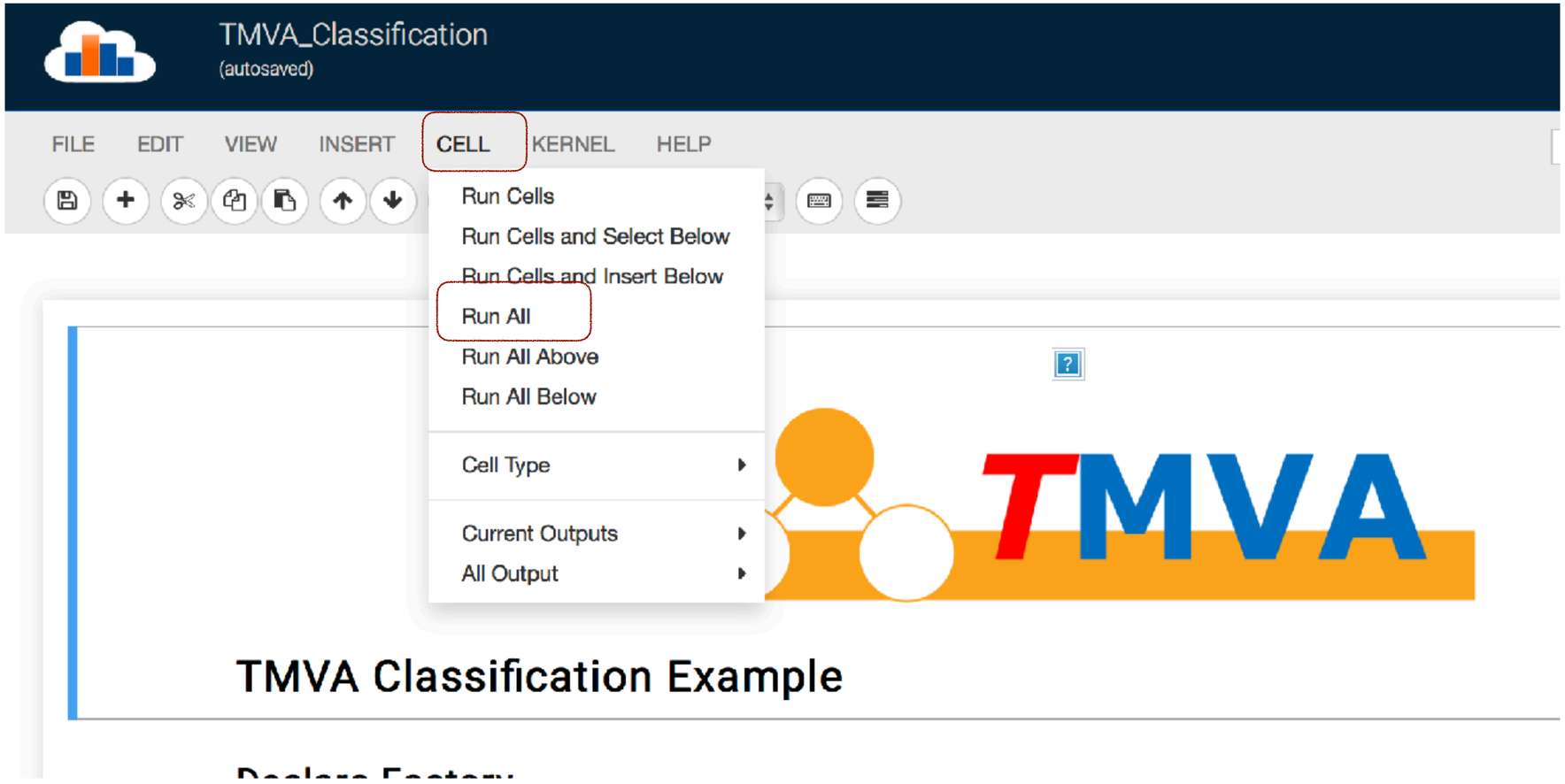
Create the Factory class. Later you can choose the methods whose performance you'd like to investigate.

The factory is the major TMVA object you have to interact with. Here is the list of parameters you need to pass

- The first argument is the base of the name of all the output weightfiles in the directory weight/ that will be created with the method parameters
- The second argument is the output file for the training results
- The third argument is a string option defining some general configuration for the TMVA session. For example all TMVA output can be suppressed by removing the "!" (not) in front of the "Silent" argument in the option string

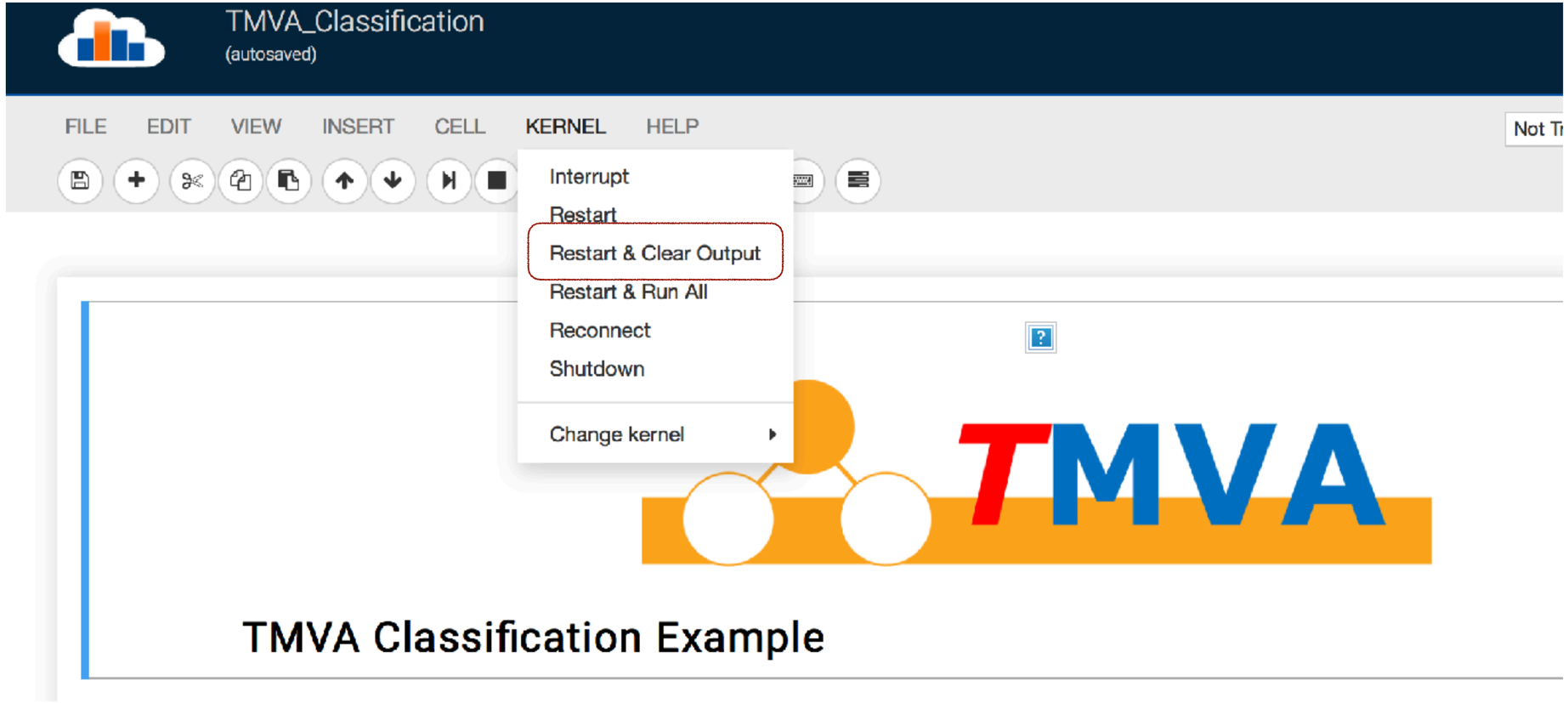
In [1]:
```
TMVA::Tools::Instance();
```

SWAN Commands: Run Cells



For running all cells

SWAN Commands: Kernel



For restarting Kernel in case of errors

Conclusions

- Very active development happening in TMVA
  - several new features released recently
  - and even more expected in a near future
    - thanks to many student contributions (e.g. from Google Summer of Code)
- Strong competition, but hopefully still good reasons for continuing using TMVA !
- Feedback from users essential
  - best way to contribute is with Pull Request in GitHub https://github.com/root-project/root
  - ROOT Forum for user support with a category dedicated to TMVA https://root.cern/forum
  - JIRA for reporting ROOT bugs: https://sft.its.cern.ch/jira
  - or just contact us (TMVA developers) directly for any questions or issues

TMVA Contributors

Lorenzo Moneta	Algorithm development, Integration and support
Sergei Gleyzer	Analyzer Tools, Algorithm Development
Omar Zapata Mesa	PyMVA, RMVA, Modularity, Parallelization and Integration
Kim Albertsson	Multi-class for BDT, cross validation/evaluation and support
Stefan Wunsch	Keras Interface, integration, improved data handling
Peter Speckmeyer	Deep Learning CPU
Simon Pfreundschuh (GSOC 2016)	Deep Learning CPU and GPU
Vladimir Ilievski (GSOC 2017)	New Deep Learning module, Convolutional layers
Saurav Shekkar (GSOC 2017)	New Deep Learning module and Recurrent layers
Manos Stergiadis (GSOC 2018)	GPU support for CNN
Ravi Selvam (GSOC 2018)	New optimisers for deep learning
Adrian Bevan, Tom Stevenson	SVMs, Cross-Validation, Hyperparameter Tuning
Attila Bagoly (GSOC 2016)	Jupyter Integration, Visualization, Output
Paul Seyfert	Performance optimization
Andrew Carnes	Regression, Loss Functions, BDT Parallelization
Anurshee Rankawat, Siddhartha Rao, Harsit Prasad	GSOC 2008 projects: GAN, VAE and LSTM

And with continued invaluable contributions from Andreas Hoecker, Helge Voss, Eckhard v.Thorne, Jörg Stelzer