New Machine Learning Developements in ROOT/TMVA

L. Moneta (CERN EP-SFT)

on behalf of the TMVA development team

ROOT Users Workshop

10-15 September 2018, Sarayevo

TMVA

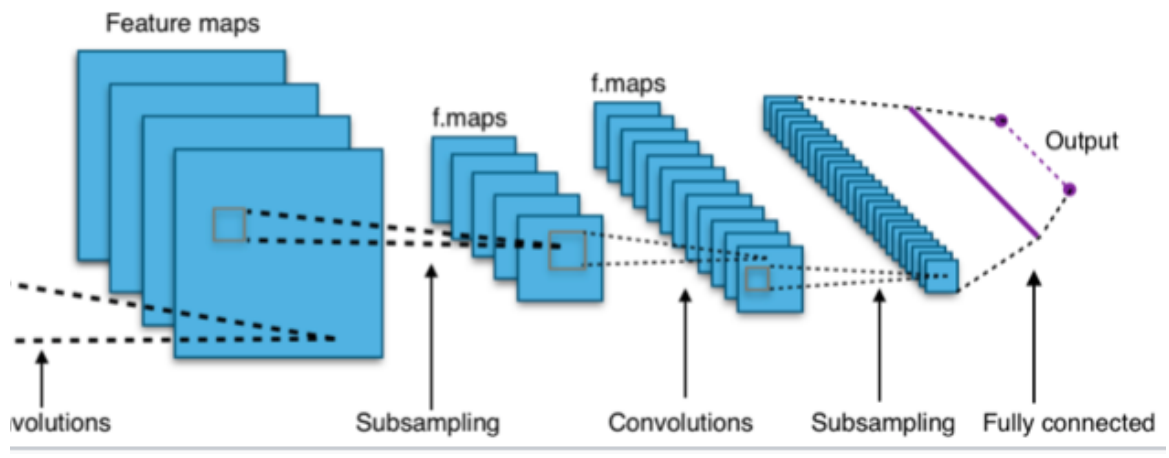
- ROOT Machine Learning tools are provided in the package TMVA (Toolkit for MultiVariate Analysis)
- Provides a set of algorithms for standard HEP usage
- Used in LHC experiment production and in several analyses
- Key features
  - Facilitates HEP research, from detector to analysis
  - Easy to use, good performance
  - Long term support
  - Several features added recently (e.g. deep learning)
- Development done in collaboration with CERN experiments and HEP community

Outline

- New machine learning methods for TMVA
  - Deep learning library
    - current status
    - performance tests with comparison to Keras/Tensorflow
- A look into the future
  - planned new developments
- Summary and conclusions

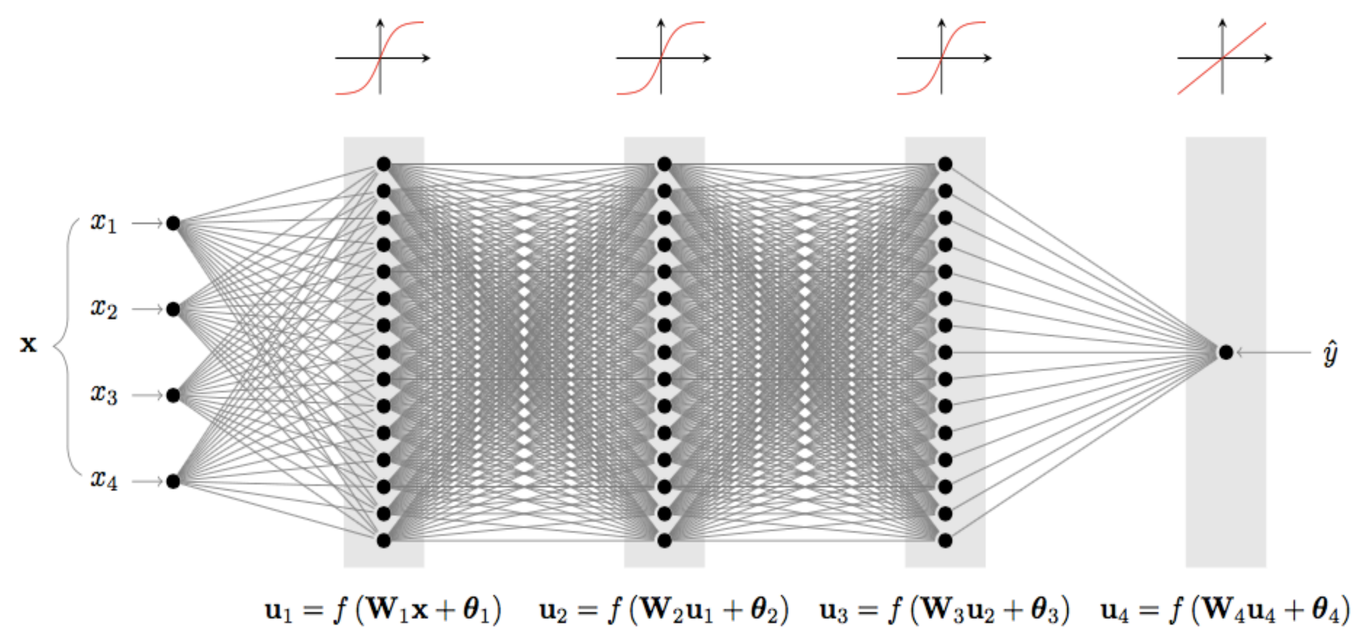
Several New Developments in TMVA

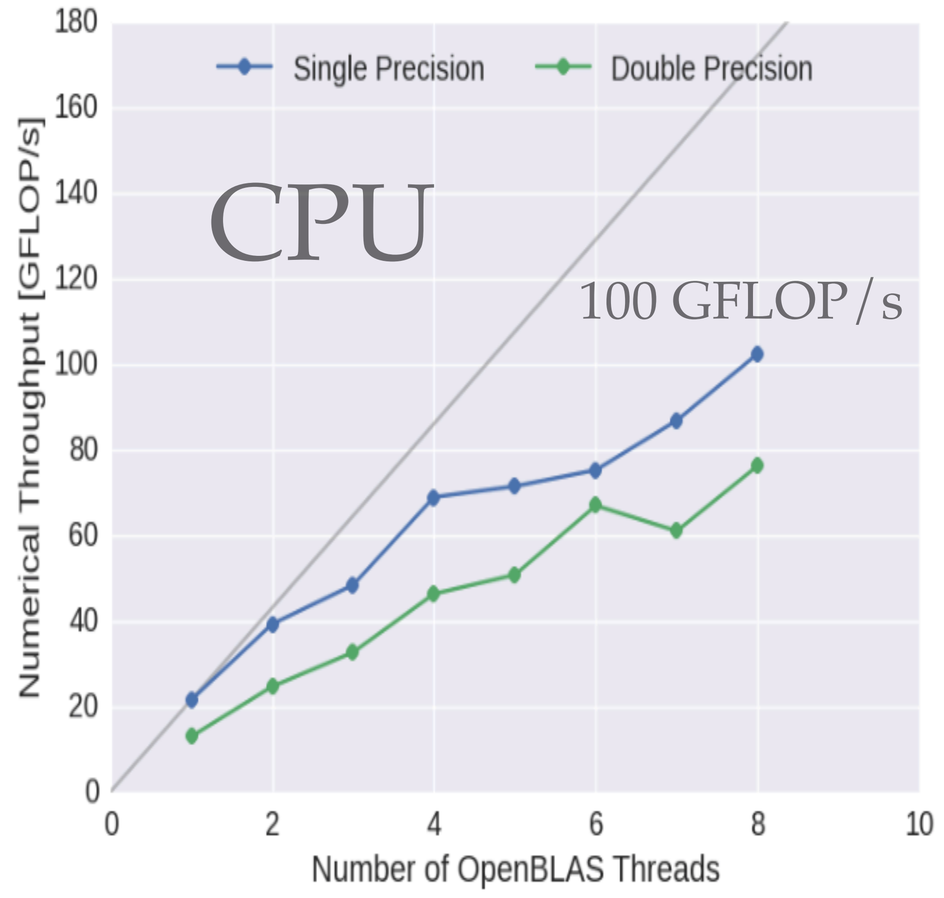
- New features available in latest ROOT version 6.14:
  - Deep Learning Module with support for
    - Dense Layer
    - Convolutional Layer
    - Recurrent Layer
  - improved BDT performance using multi-thread parallelisation
  - improved Cross Validation
- And also available since ROOT 6.12:
  - new interfaces to external tools (scikit-learn, Keras, R)

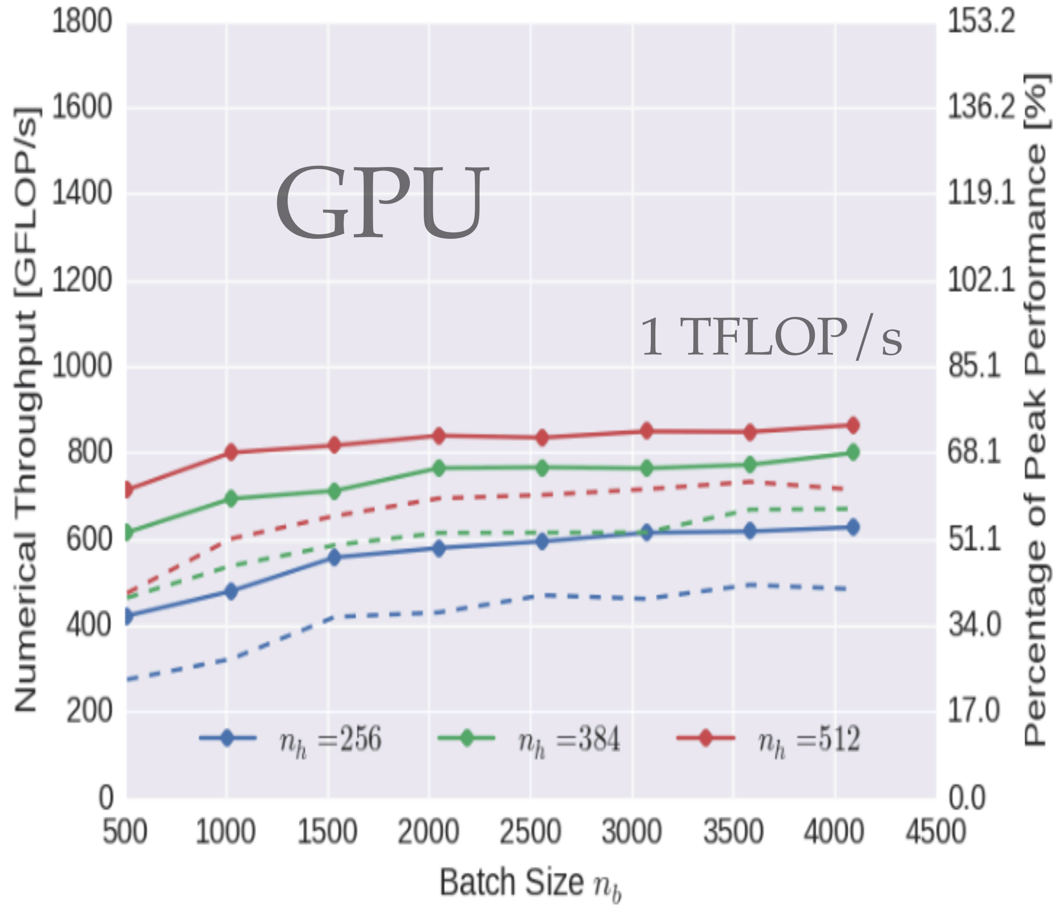


Deep Learning in TMVA

- Deep Learning library in ROOT/TMVA
  - parallel evaluation on CPU
    - implementation using OpenBlas and Intel TBB library
  - GPU support using CUDA
  - Excellent performance and high numerical throughput





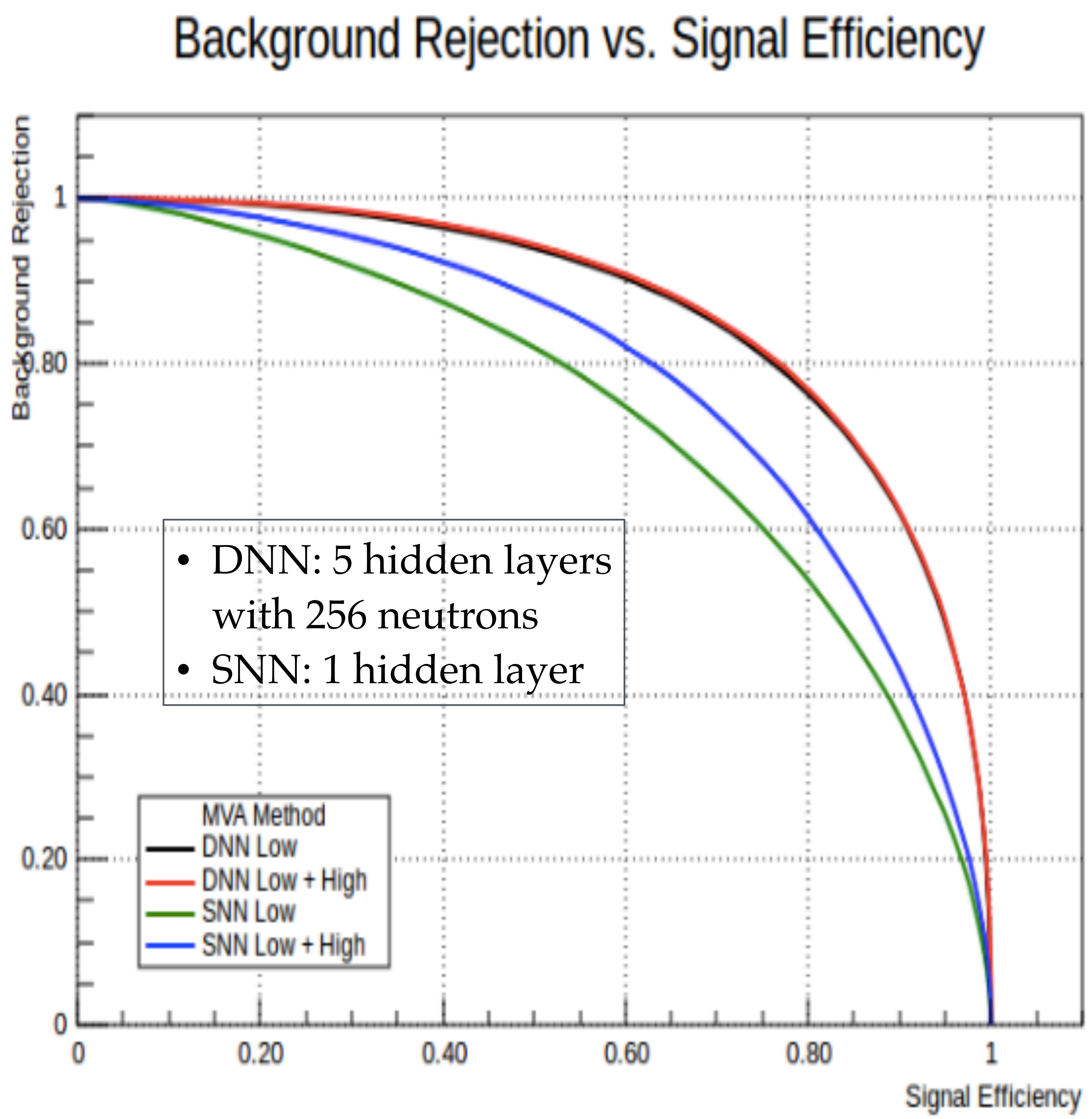


- For more information see
  - [https://indico.cern.ch/event/565647/contributions/2308666/attac ... va_dr]

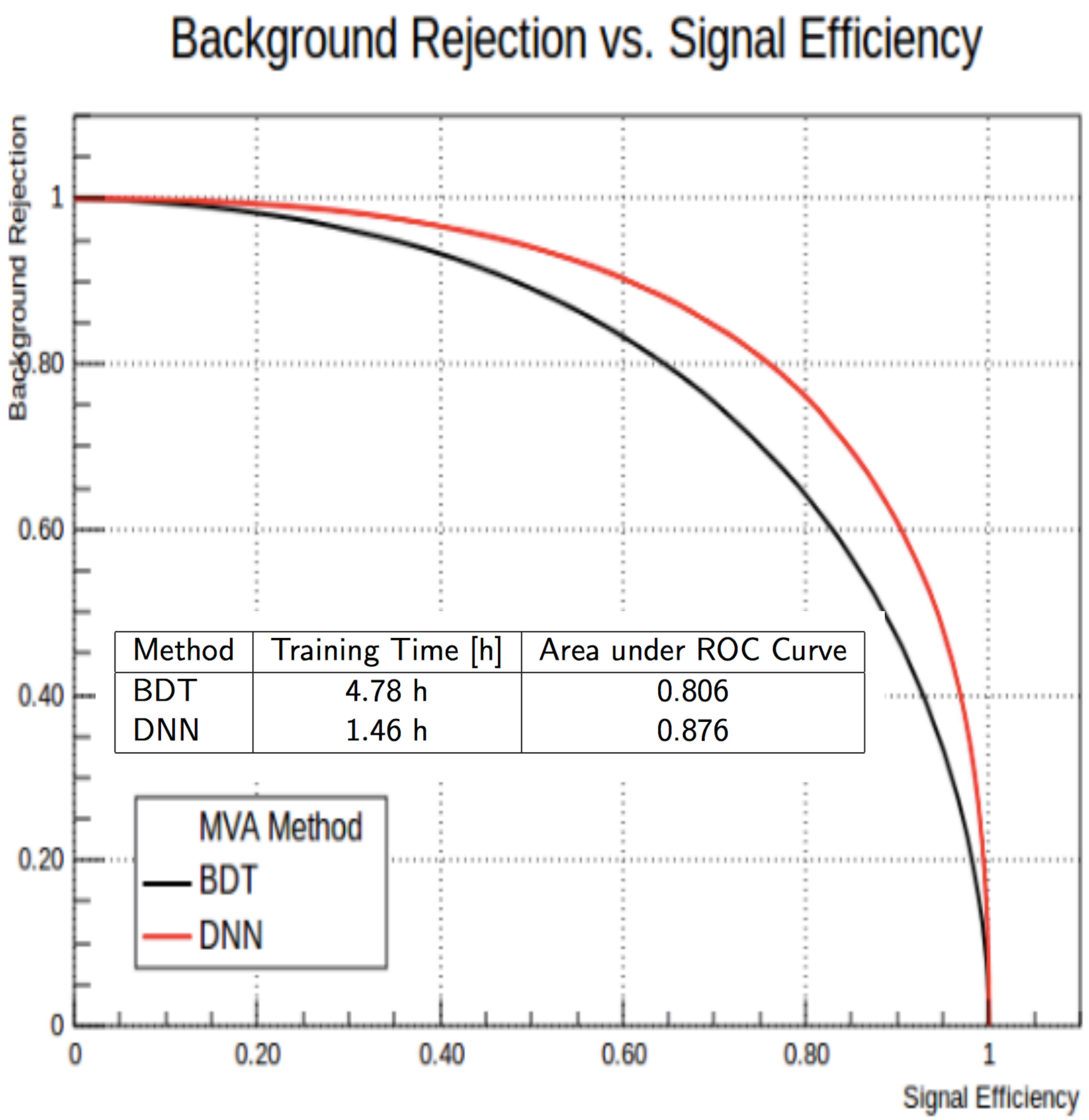
- Available for dense layers since ROOT 6.08 but extended the design in 6.14 to a new module supporting different layer types

Deep Learning Performance

DNN vs Standard ANN



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

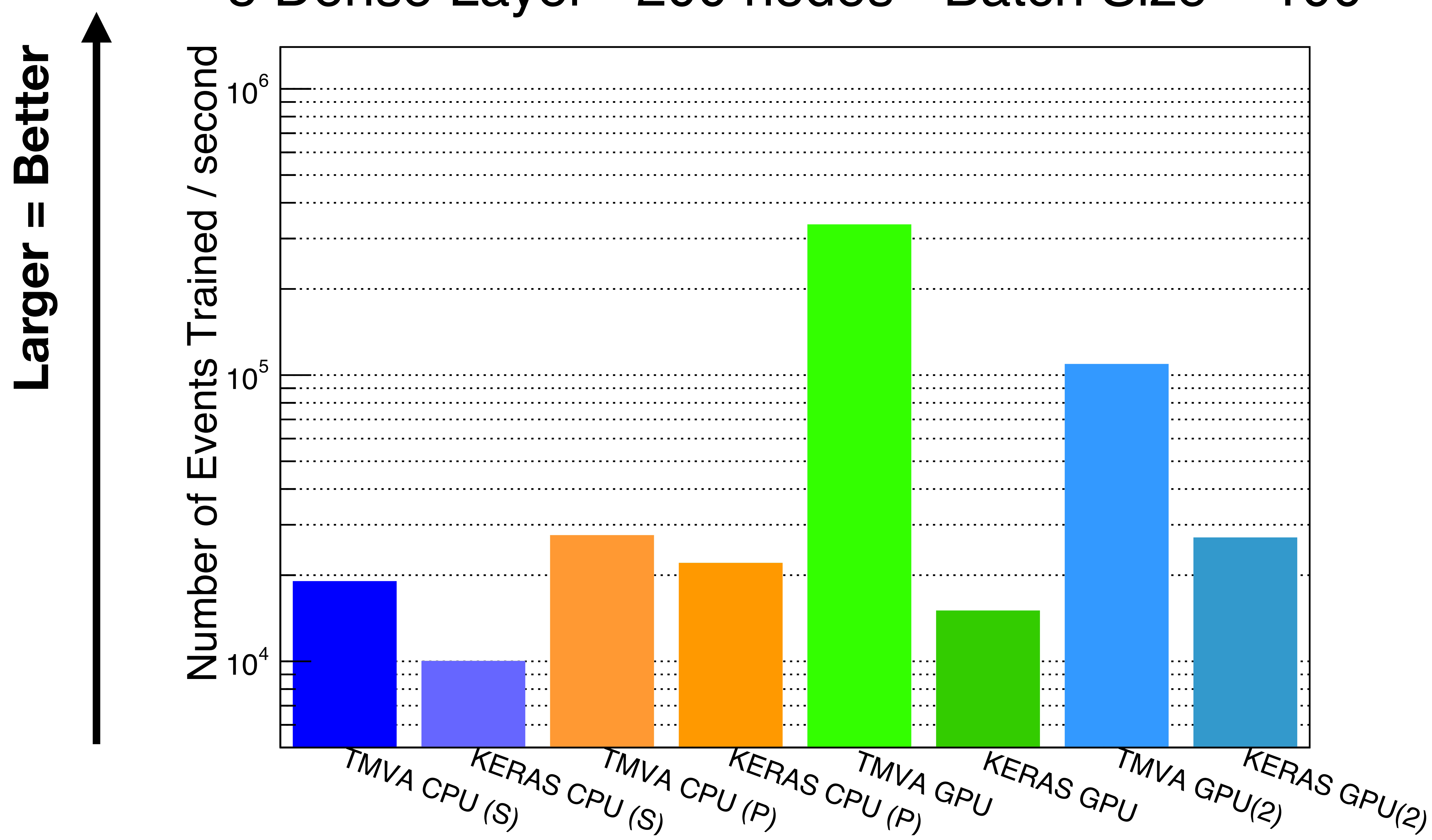
DNN Training Performance

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?



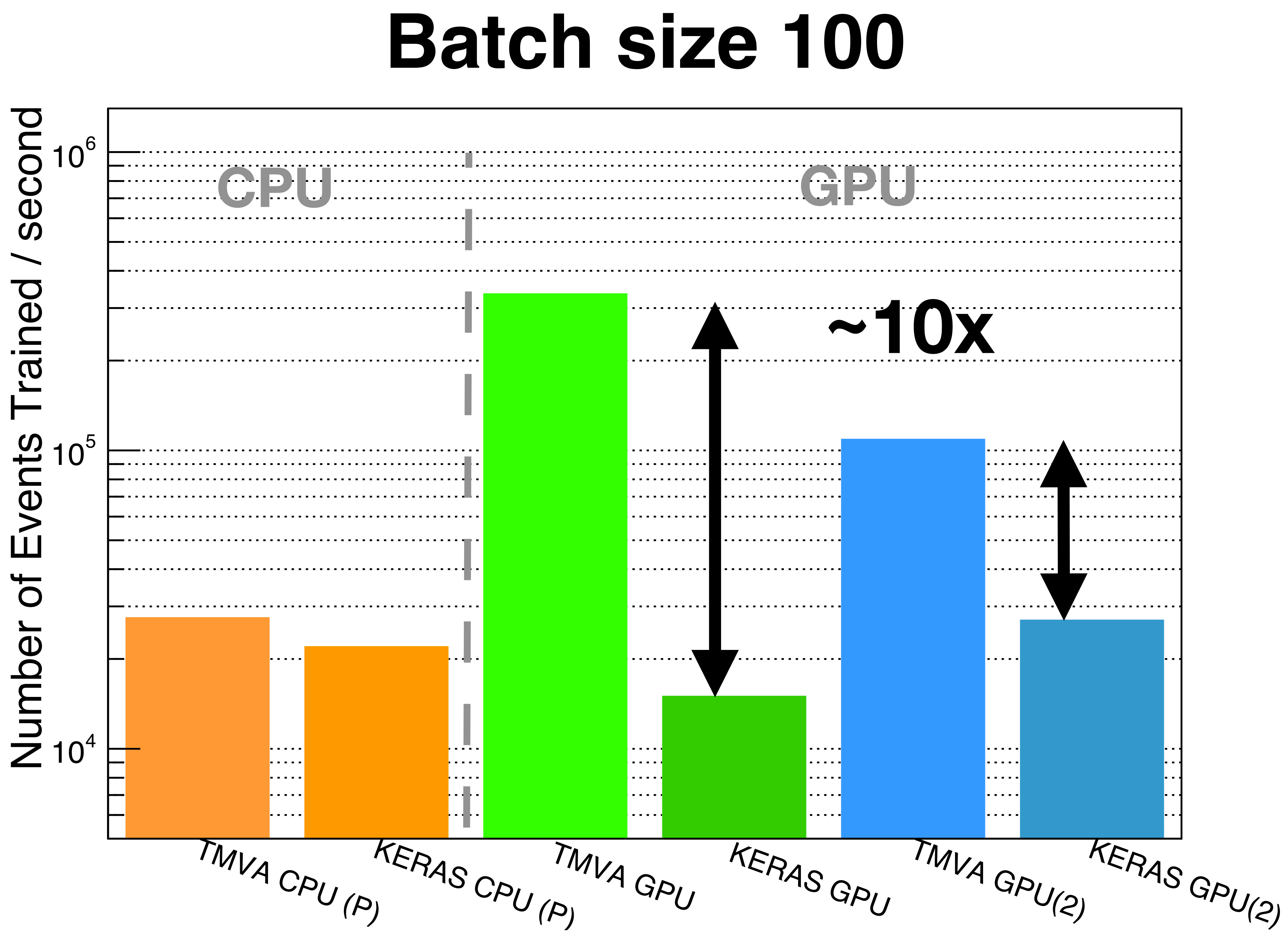
(S) — Single threaded

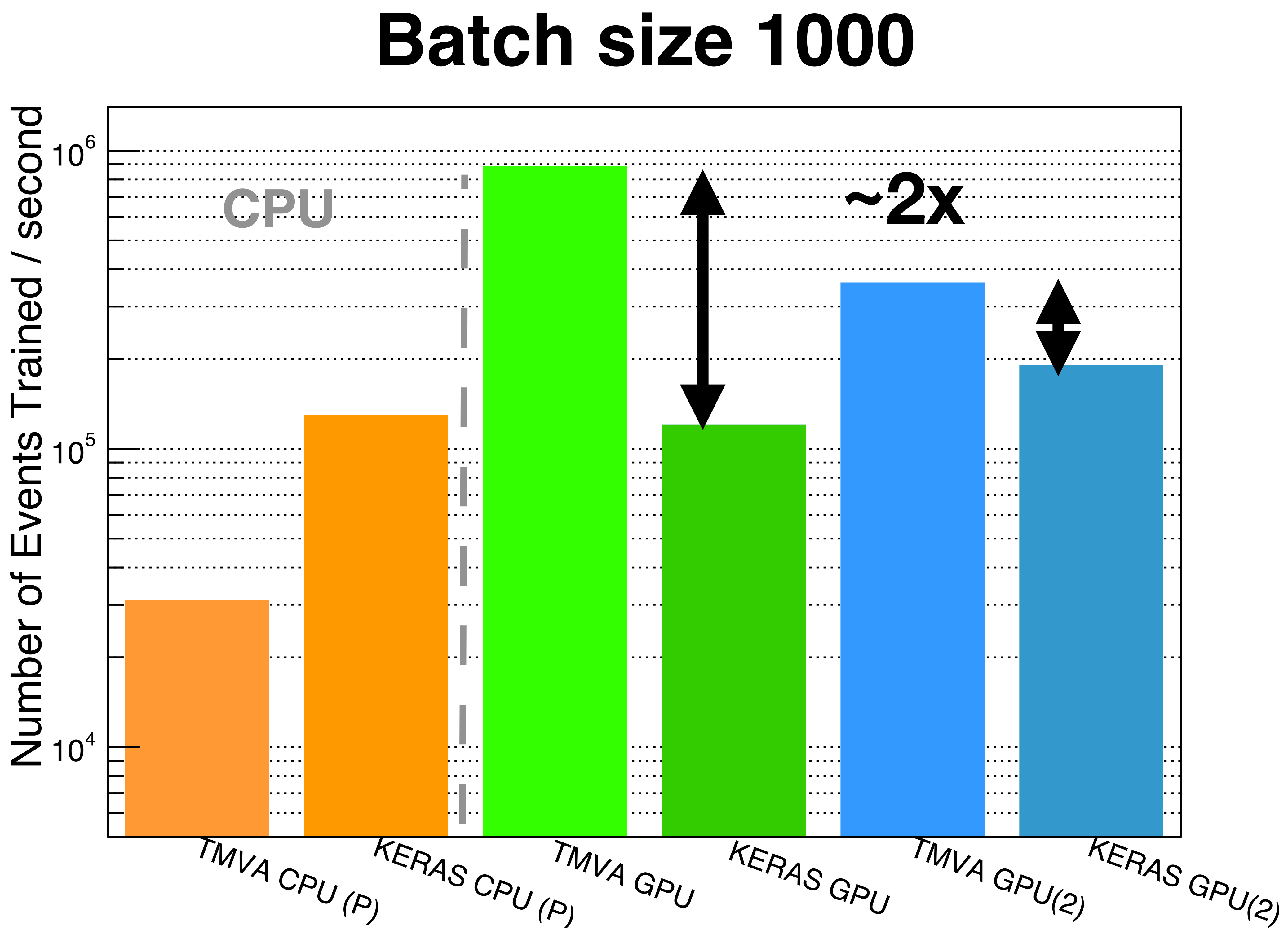
(P) — Intel Xeon E5-2683 (28 core)

GPU — GTX1080Ti

GPU(2) — GTX980

DNN Training Performance





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

(S) — Single threaded

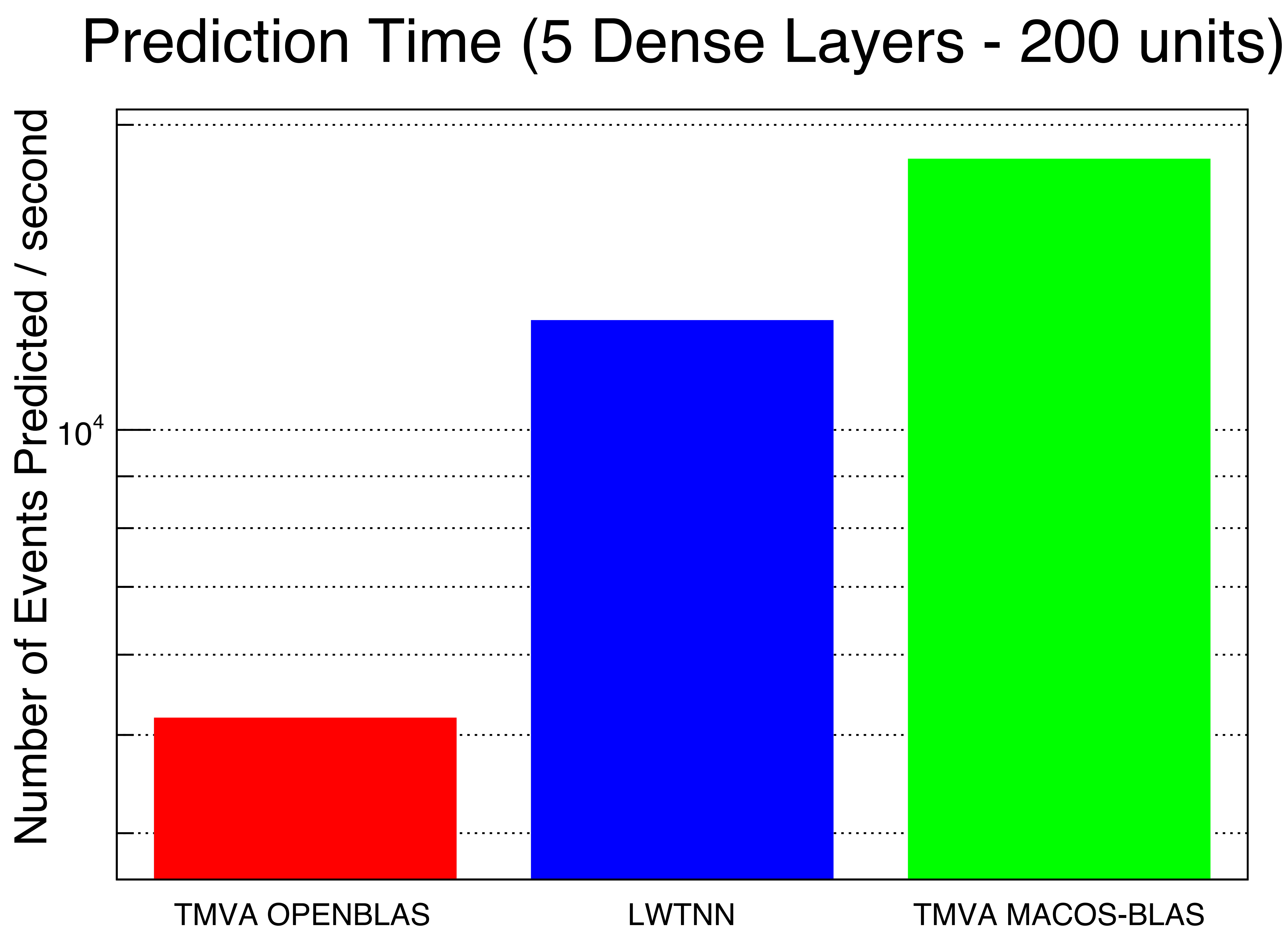
(P) — Intel Xeon E5-2683 (28 core)

GPU — GTX1080Ti

GPU(2) — GTX980

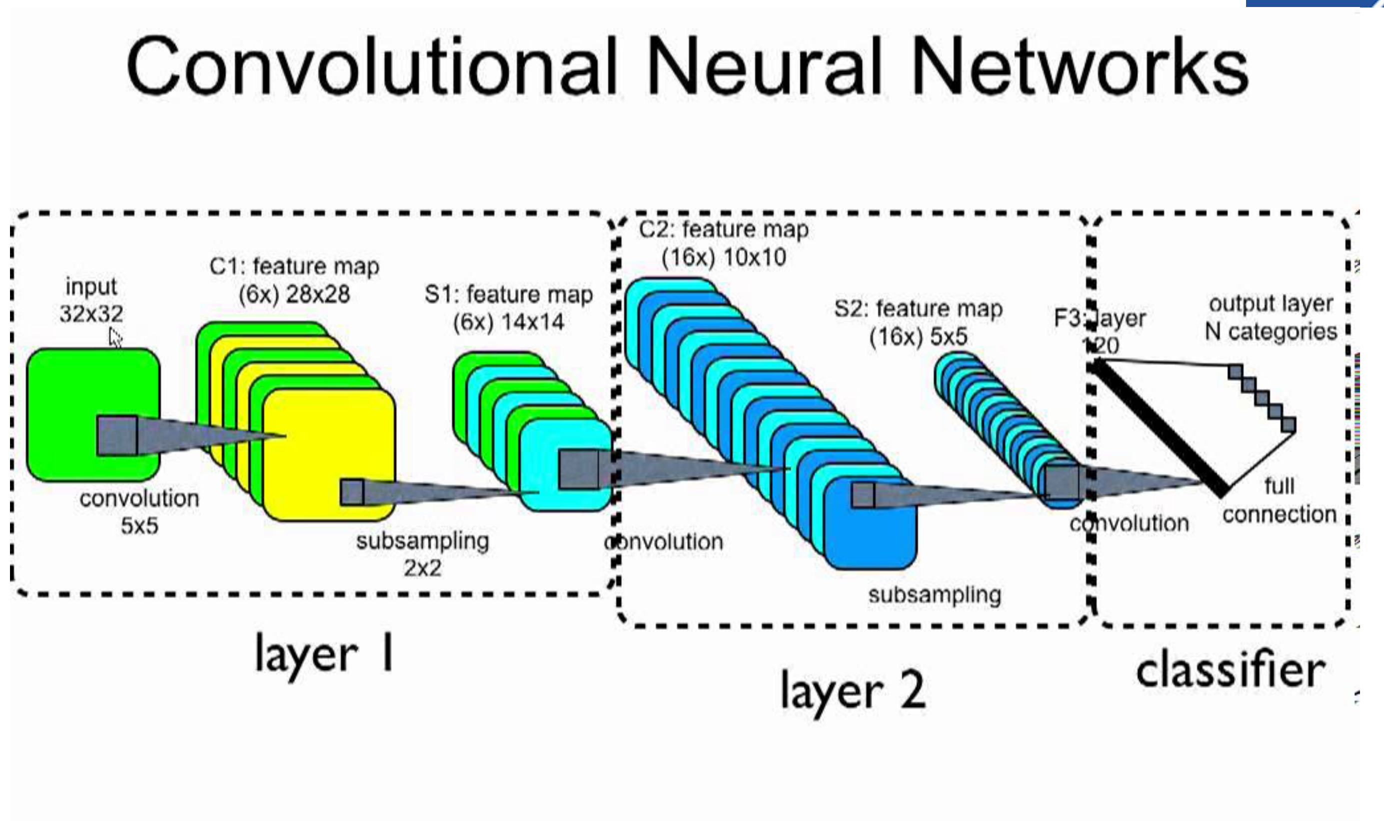
DL Evaluation Performance

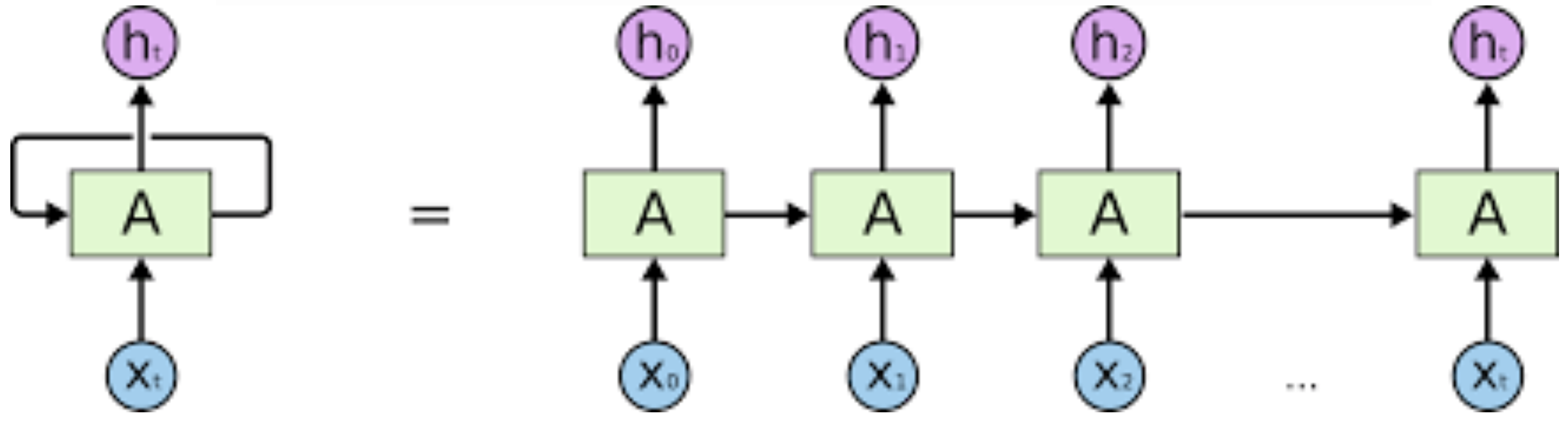
- **Single event evaluation time** for 5 layer network
  - For time critical applications — e.g. on-line reconstruction
  - Fast! 1.5 times speedup over specialised libraries like LWTNN when using optimised Blas library exploiting vectorisation
- For batched evaluation, same story as training

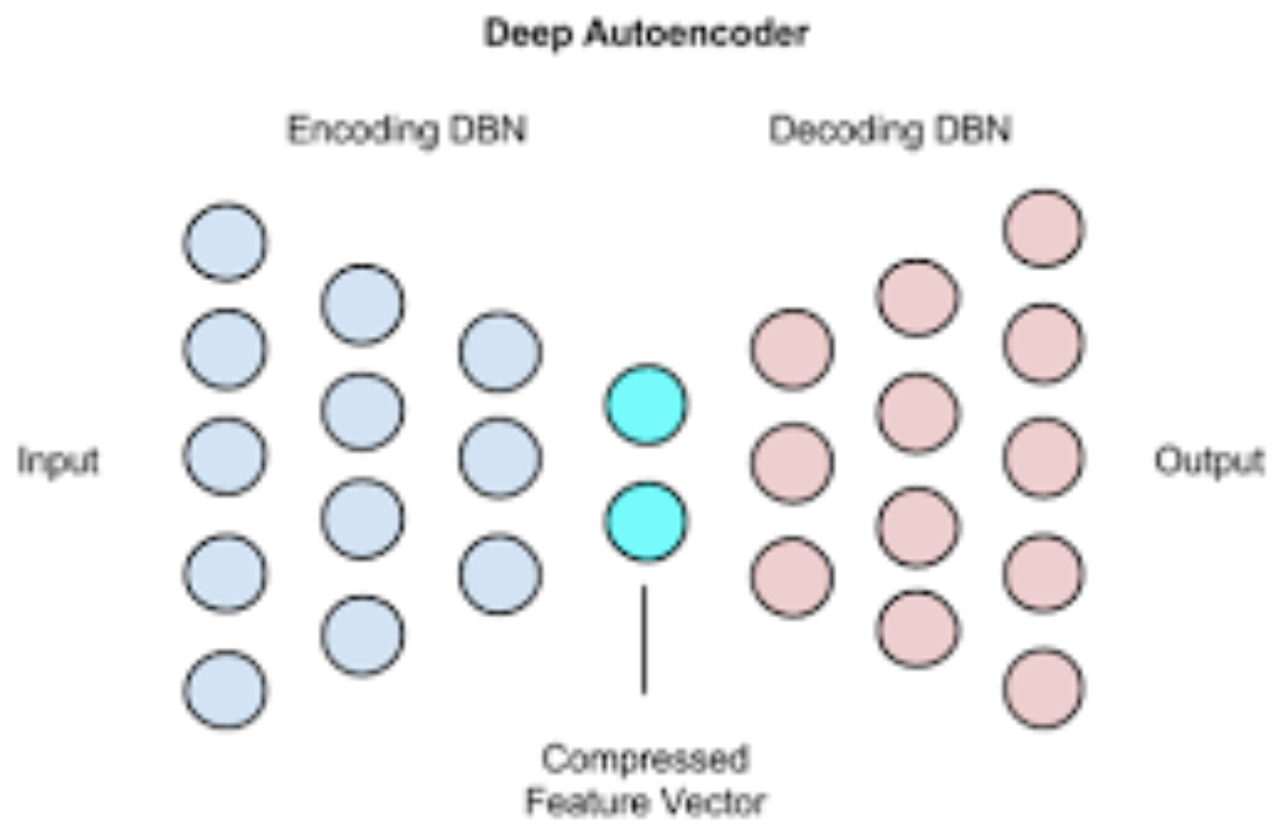


New Developments

- Extended Deep Neural Network classes
  by adding:
  - Convolutional Neural Network
    - very powerful for image data sets
  - Recurrent Neural Network
    - useful for time-dependent data
- Working also on auto-encoders and generative adversarial networks

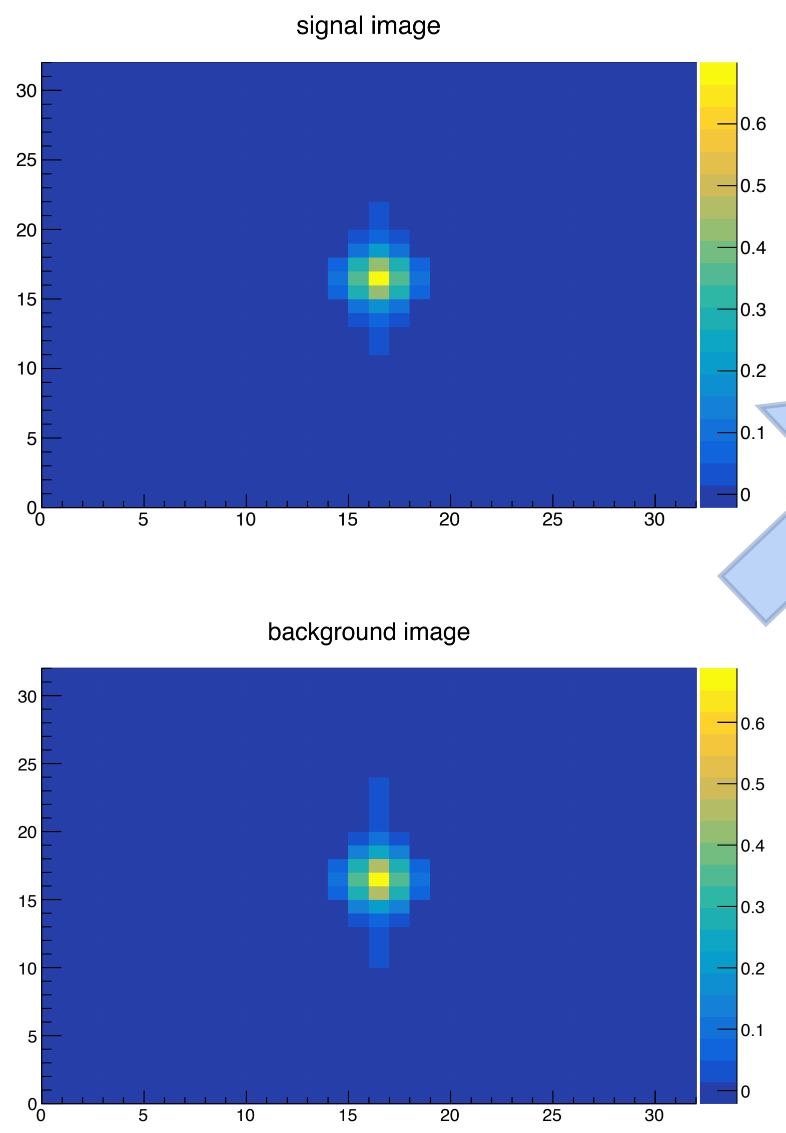




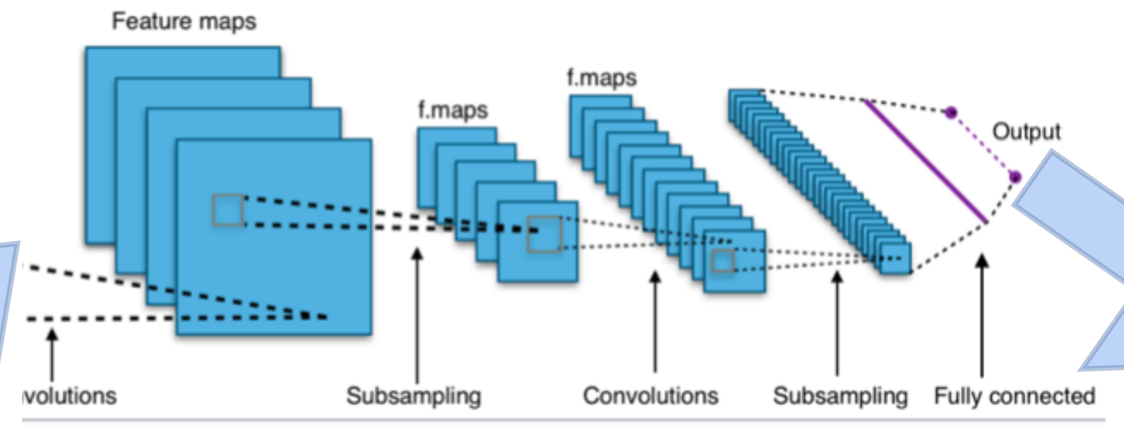


Convolutional Neural Network

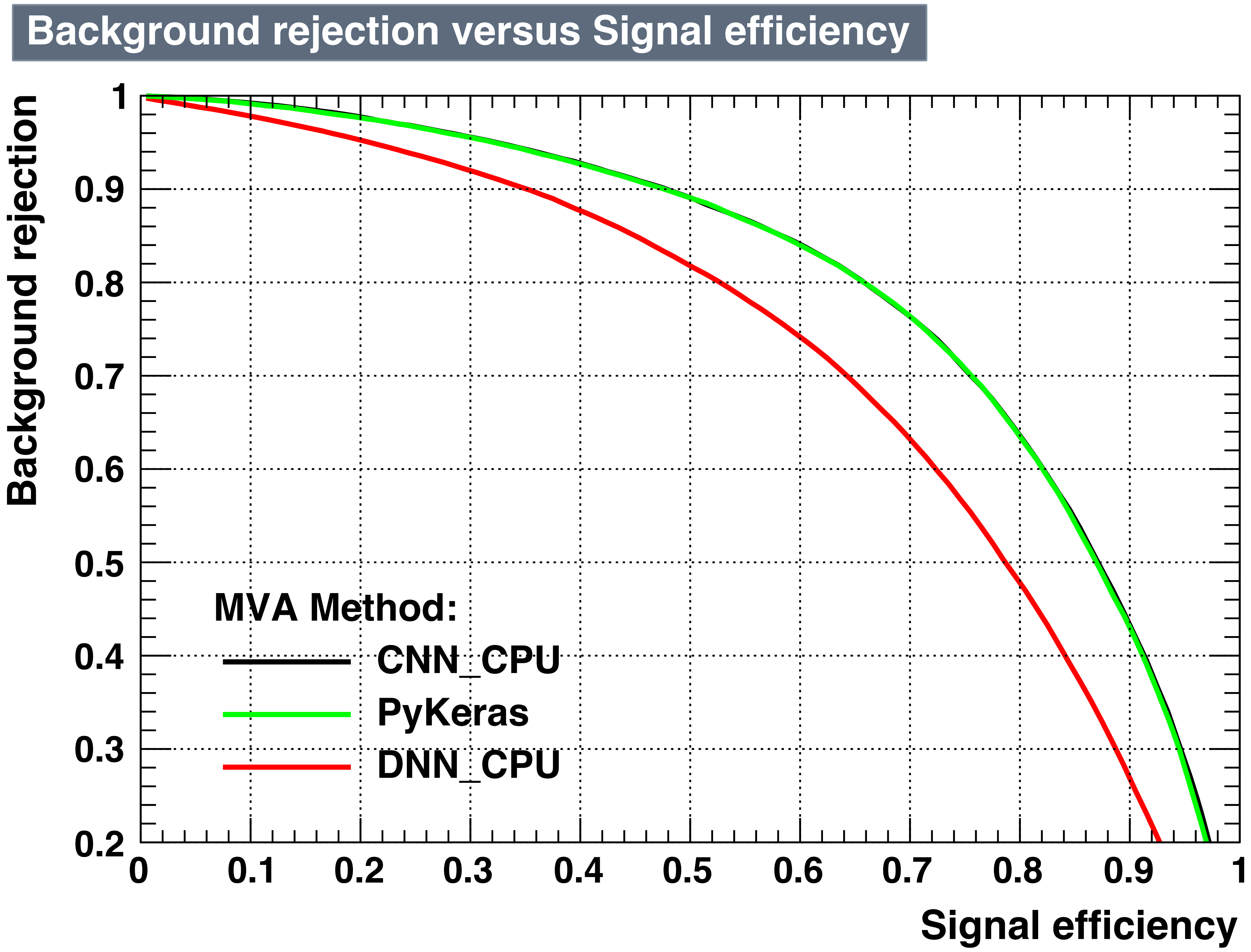
- Available in latest ROOT release (6.14)
- Supporting CPU parallelization, GPU is available in ROOT master
  - parallelisation and code optimisation is essential



input 32x32 images



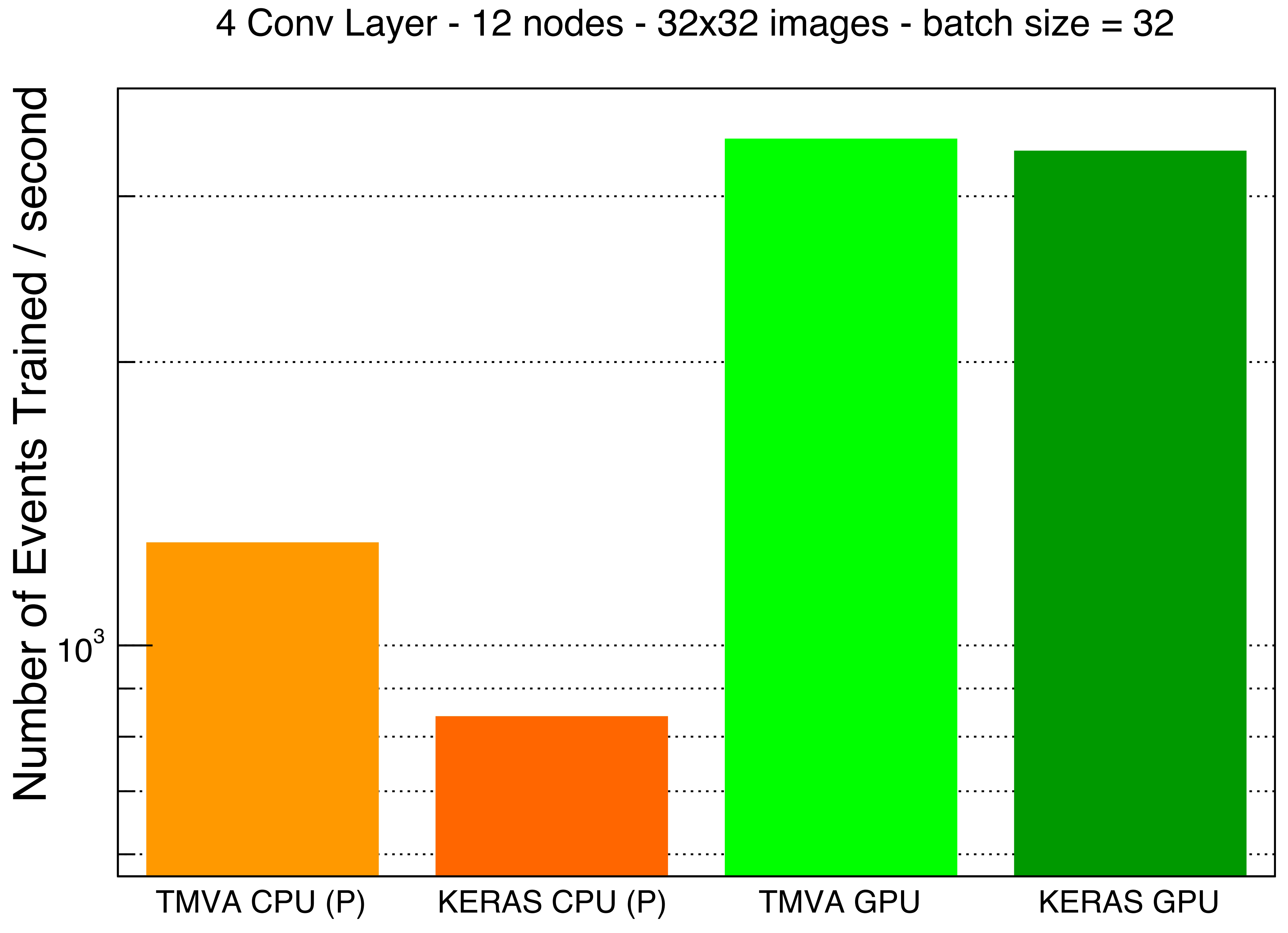
Convolutional + Pooling +
Dense layers



CNN Training Performance

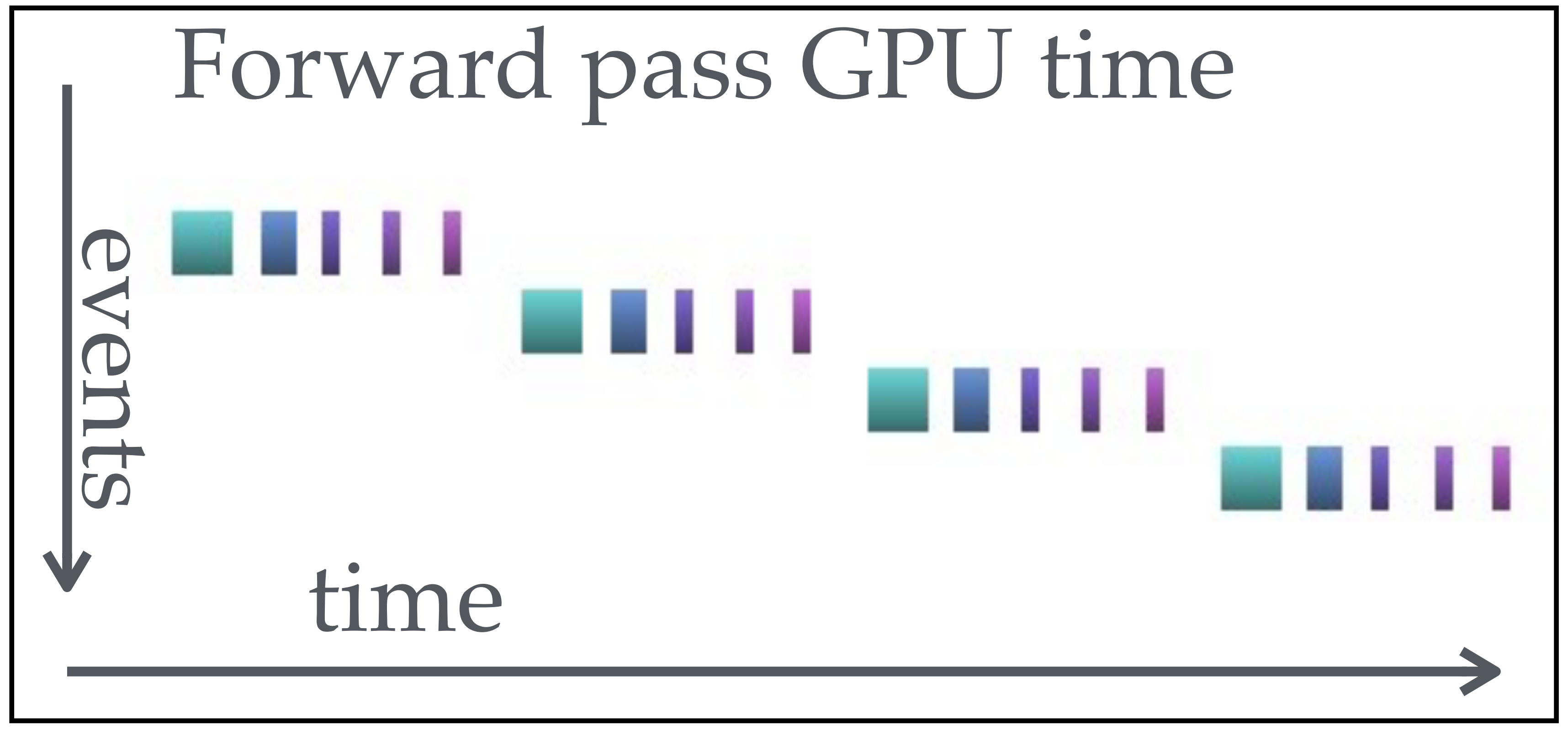
CNN performance for TMVA CPU and GPU

- 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



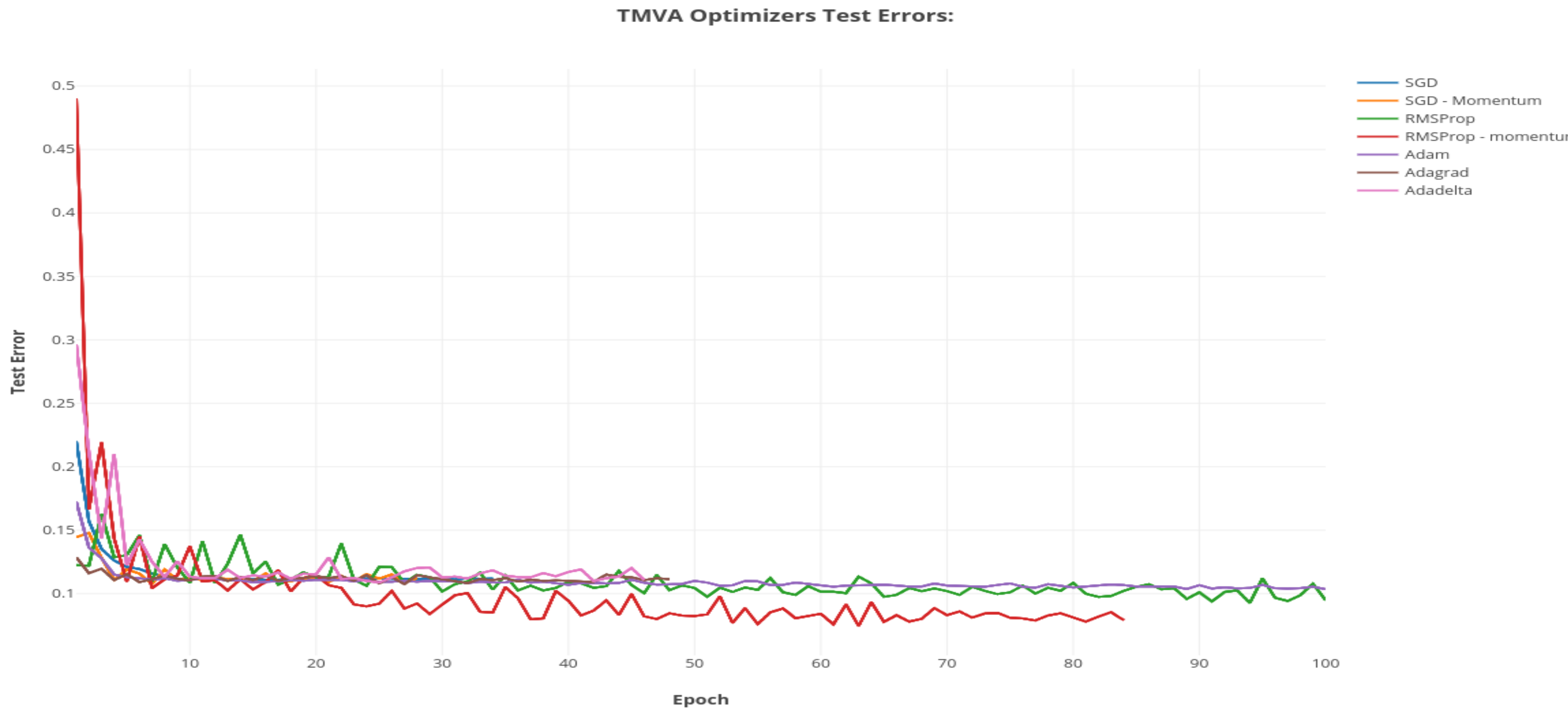
CNN Developments

- GSOC student developed GPU implementation
  - excellent performances already obtained
  - further optimisation are possible using parallelisation within batches
    - require modifying used data structure for GPU
      - using a tensor with (batch size x depth x image size)
      - perform loop for events in a batch in the GPU
  - Further planned developments
    - support for 1D and 3D convolutions
    - implement transpose (inverse) convolution (for generative models)



NEW DL Optimizers

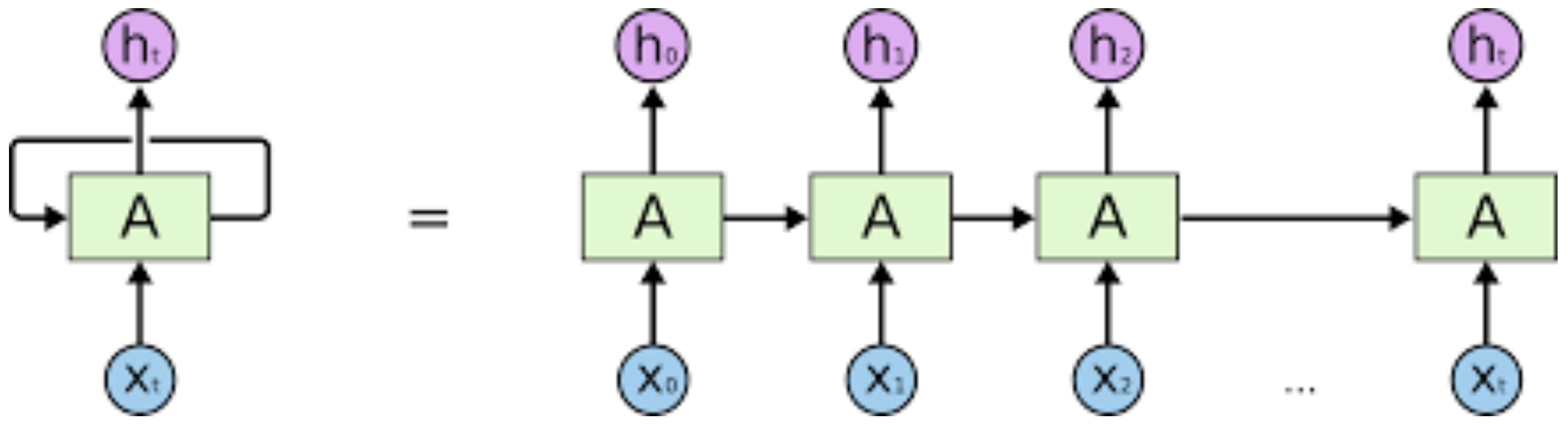
- Integrated in TMVA master new deep learning optimisers
  - In addition to SGD added
    - support acceleration for SGD using momentum
    - ADAM (new default)
    - ADADelta
    - ADAGrad
    - RMSProp

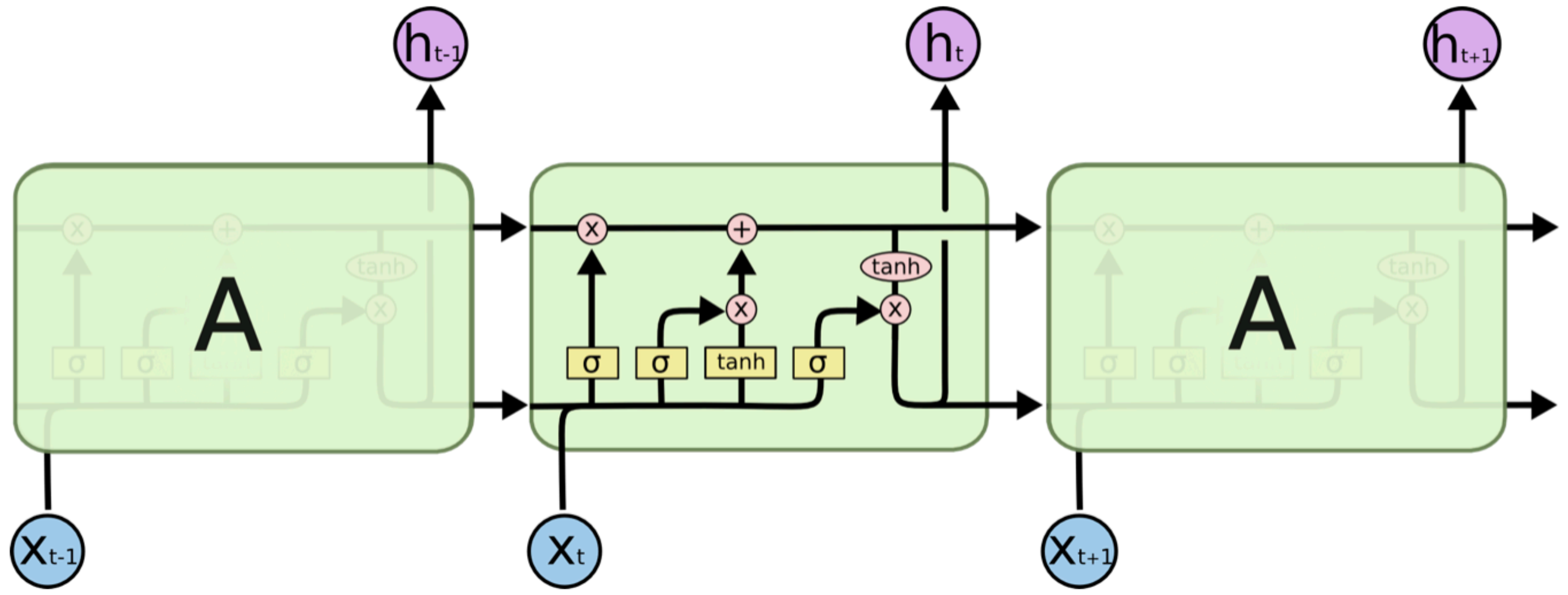


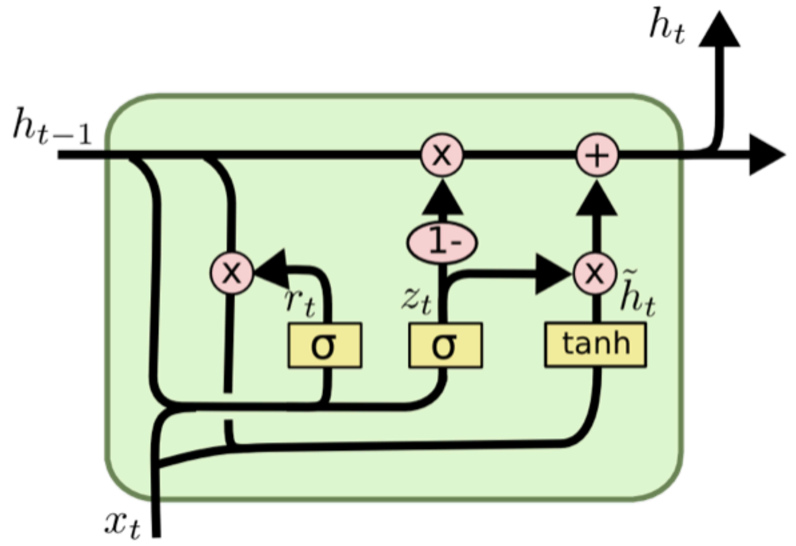
With these new optimisers we need less epochs (iterations) to converge !

Recurrent Network

- Added in 6.14 first implementation of a recurrent layer
  - GSOC 2017 project
- RNN are very useful for time depend data
  - several applications in HEP (e.g. flavour tagging)
- 2018 GSOC project for developing a LSTM layer
  - LSTM (Long Short Term Memory) can cope with long term dependencies in the sequence
- Work is not completed, but plan to complete and integrate first version before end of the year
- Once LSTM layer is available also GRU (Gated Recurrent Unit) can be implemented

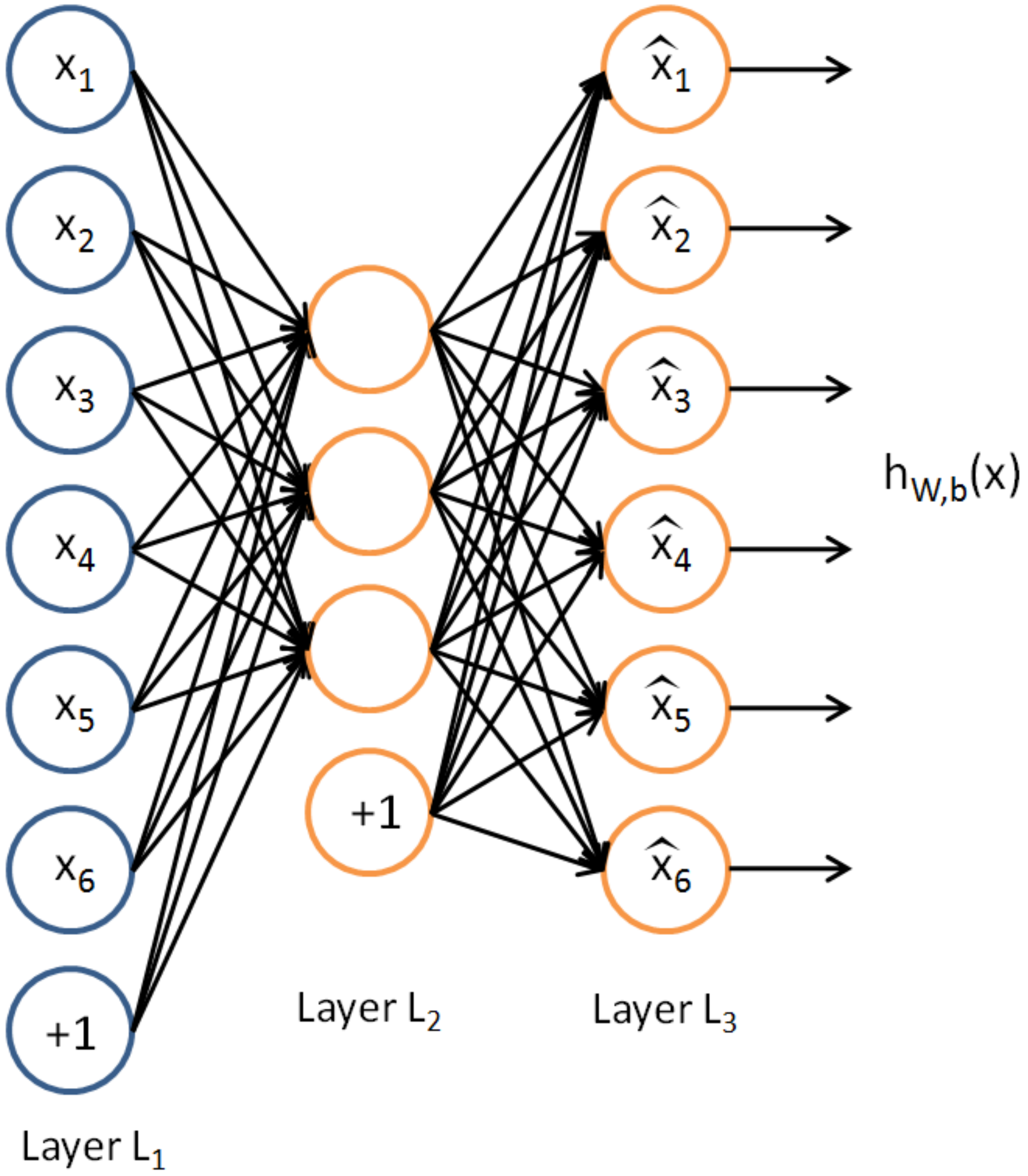






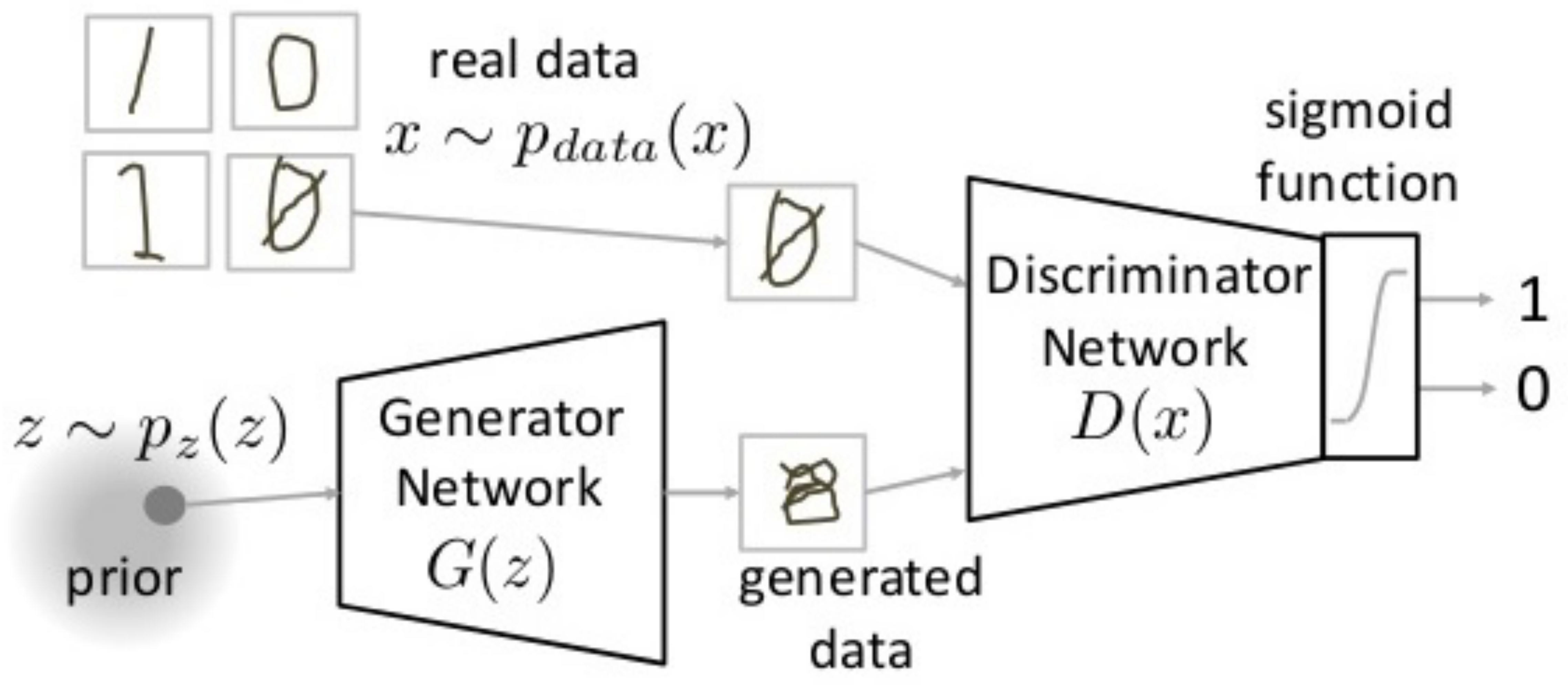
Deep Auto-encoder

- A neural network trained to learn the input data
  - Unsupervised machine learning methods
  - Useful for dimensionality reduction or anomaly detection
  - Can be used also as a generator
    - Variational Auto-encoder
- GSOC project on developing auto-encoders
  - implemented Kullback-Leiber divergence
  - MethodAE class for building auto-encoders
- Plan to integrate it in TMVA



Generative Adversarial Model

- GSOC project on developing a class for creating and training GAN based on the current TMVA DL library
- MethodGAN class
  - plan to complete and integrate in the ROOT master in the next months

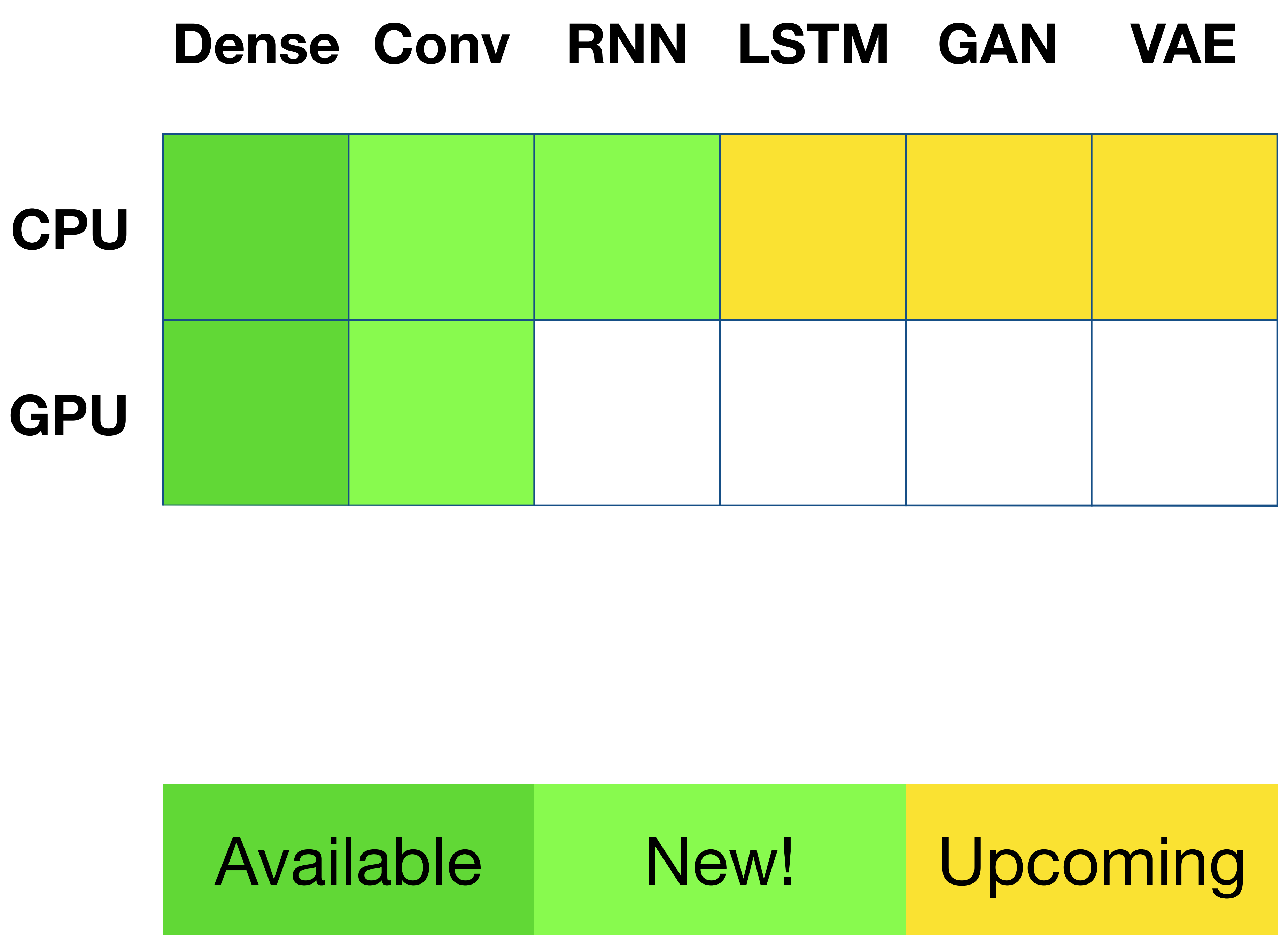


Training of GAN can be difficult, min-max game optimisation

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

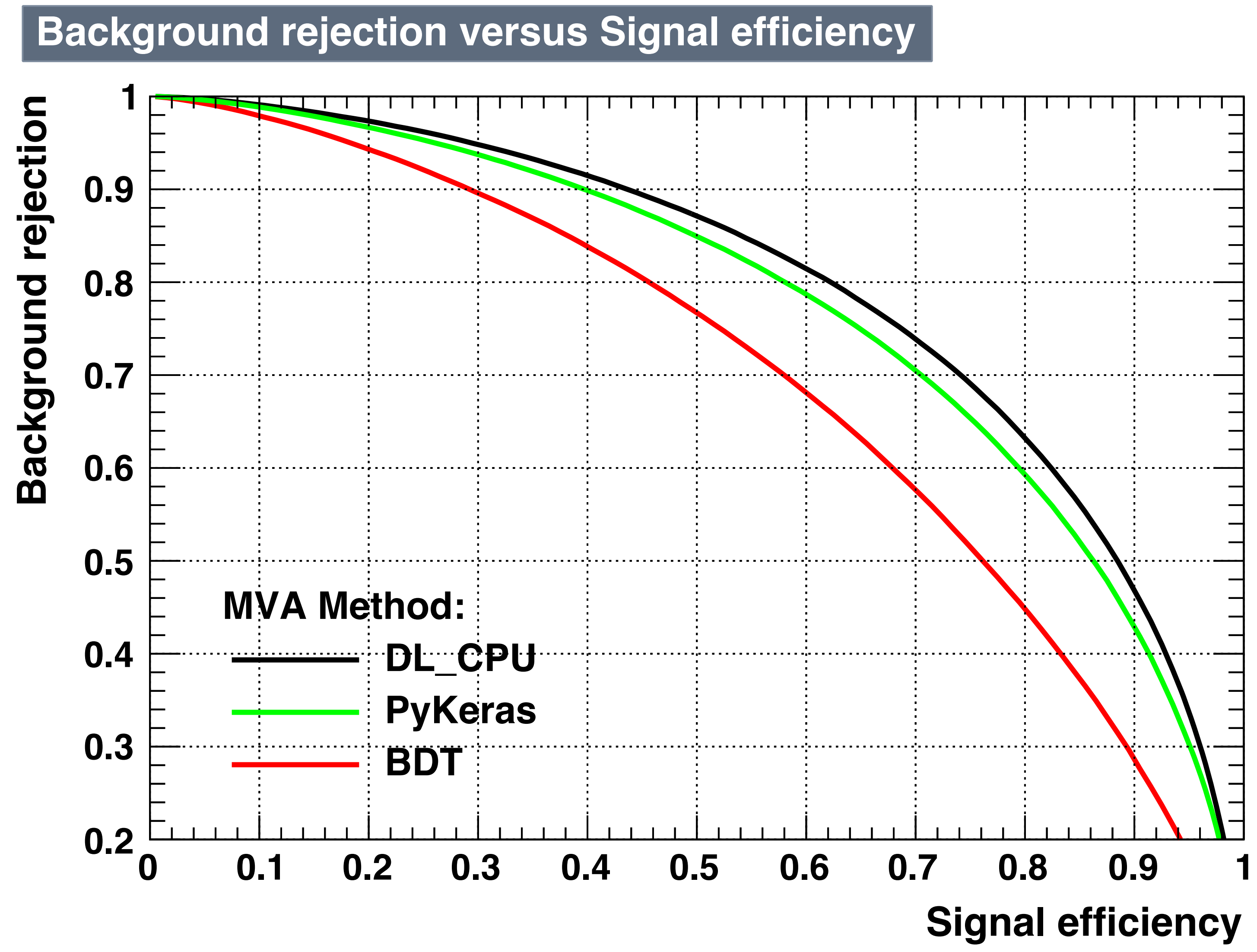
Summary Deep Learning

- 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.

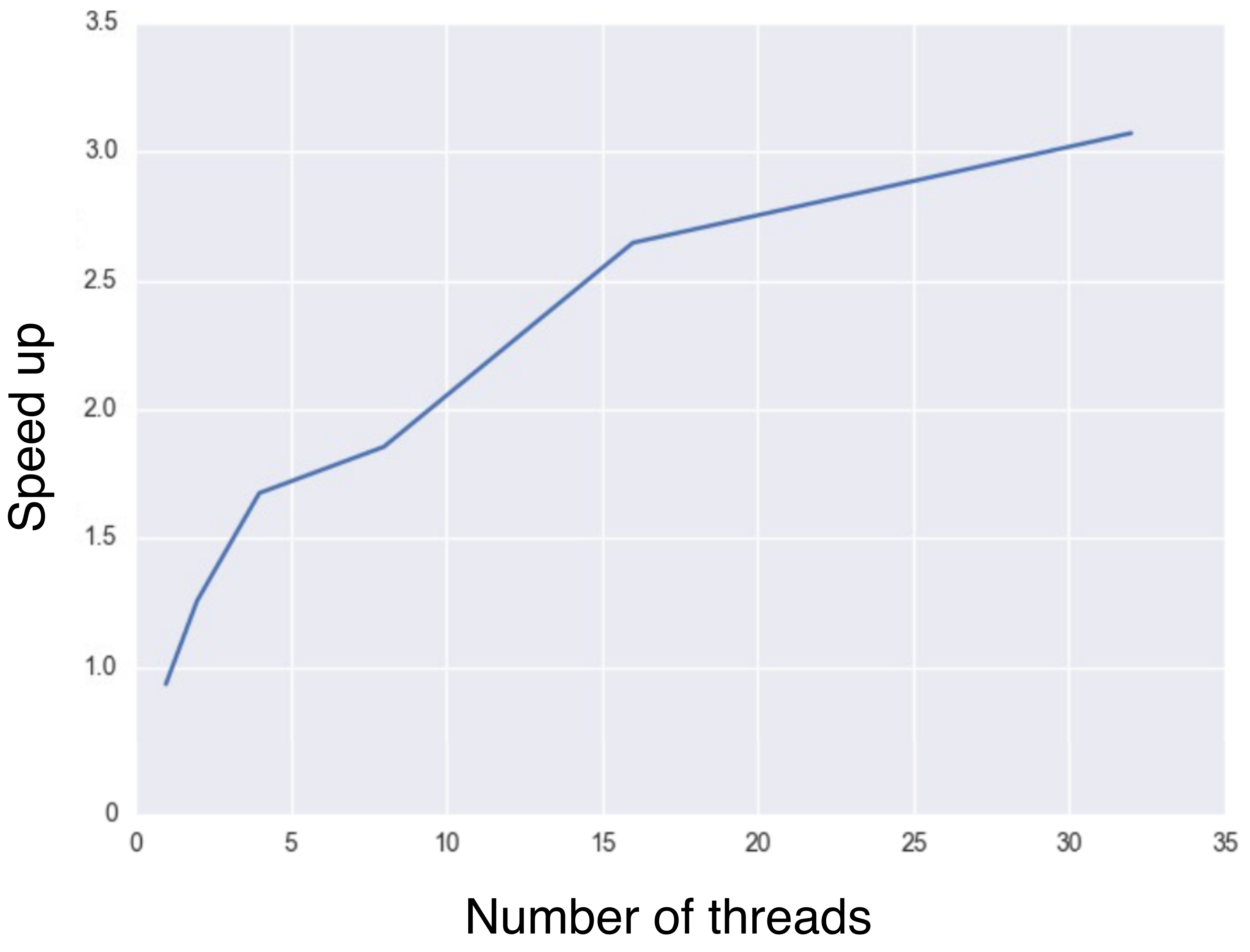


- **RMVA**: **Interface to Machine Learning methods in R**
  - c50, xgboost, RSNNS, e1071
- **PYMVA**: **Python Interface**
  - **scikit-learn**
    - with RandomForest, Gradiend Tree Boost, Ada Boost
  - **Keras** (Theano + Tensorflow)
    - support model definition in Python and then training and evaluate in TMVA
- Input data are copied now internally from input ROOT trees to TMVA data structure and then to Numpy arrays
  - working on direct mapping from ROOT tree to Numpy arrays
    - see Stefan W. presentation

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

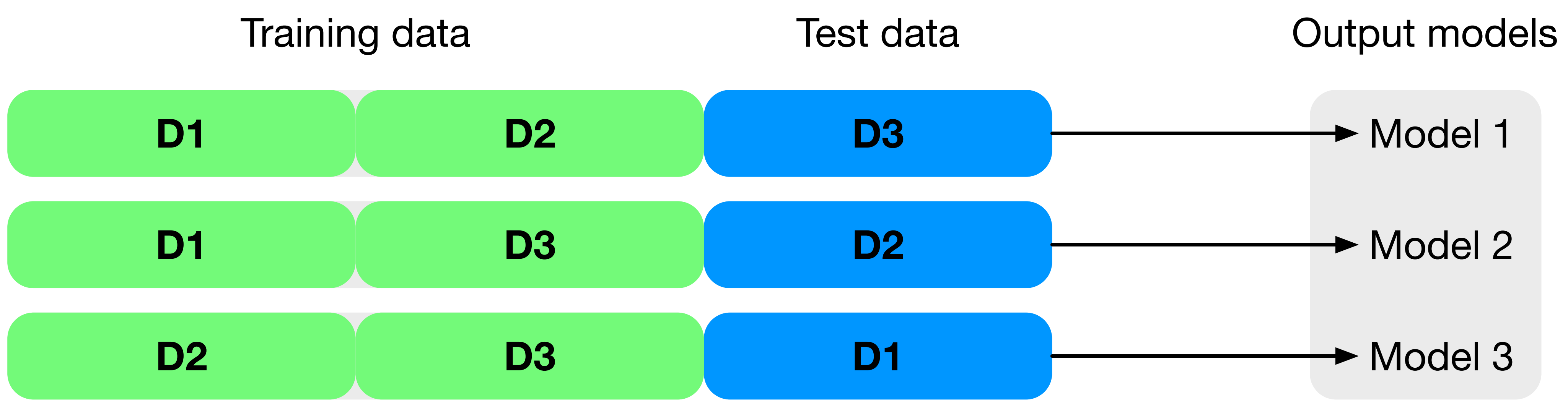
10 Trees — 1 Million events

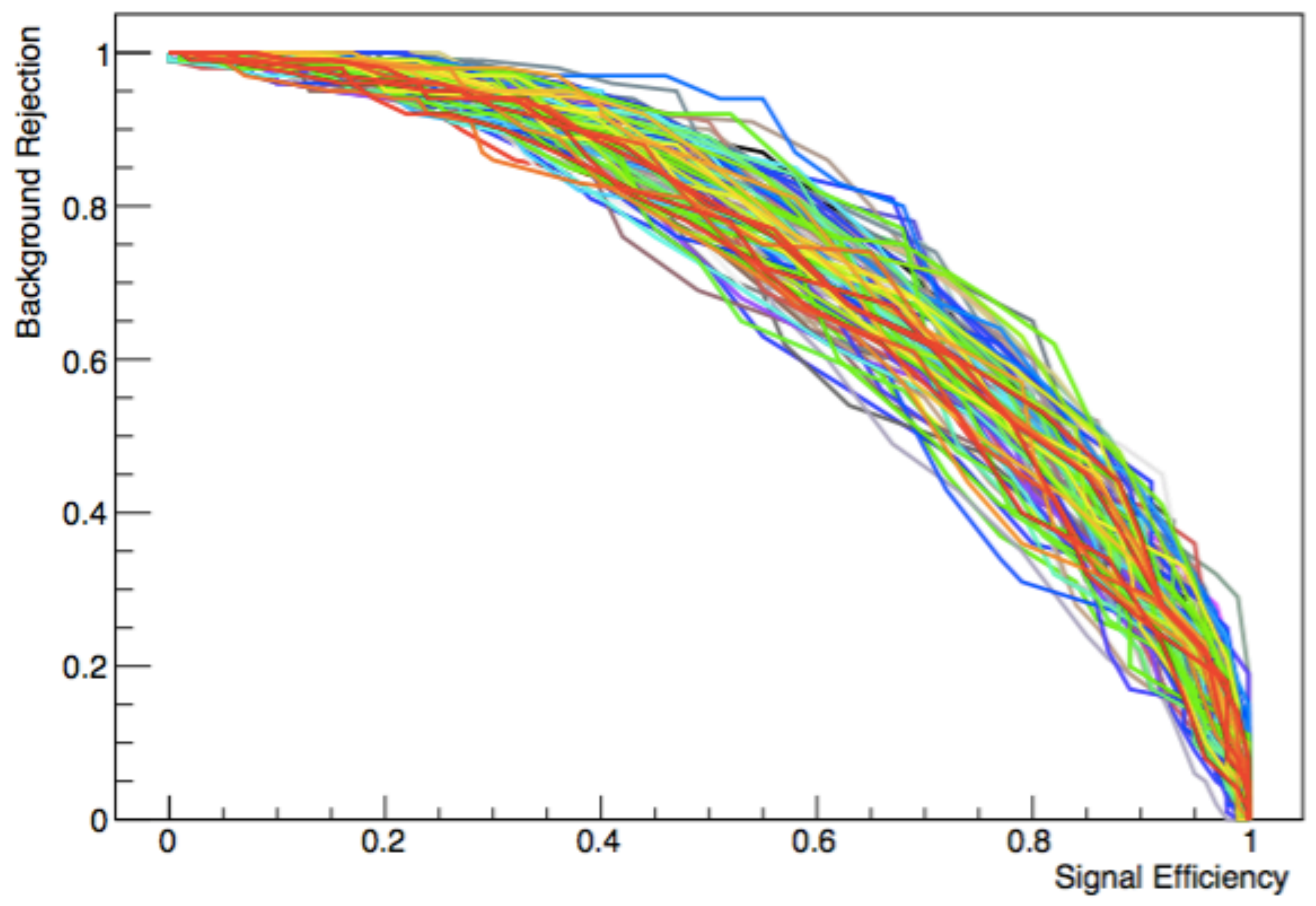


Original slide by Andrew Carnes

Cross Validation in TMVA

- TMVA supports k-fold cross-validation





  - Integration with TMVA analysis tools (e.g. GUI)
  - support for "CV in application"
- 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
- See Kim A. presentation

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 (see Stefan W. presentation)

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

Conclusions

- Very active development happening in TMVA
  - several new features released recently and more expected for next release
  - thanks to many student contributions (e.g. from Google Summer of Code)
- Users contributions and feedback from users are essential
  - ROOT is an open source project
  - 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.ch/phpBB3/
- 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

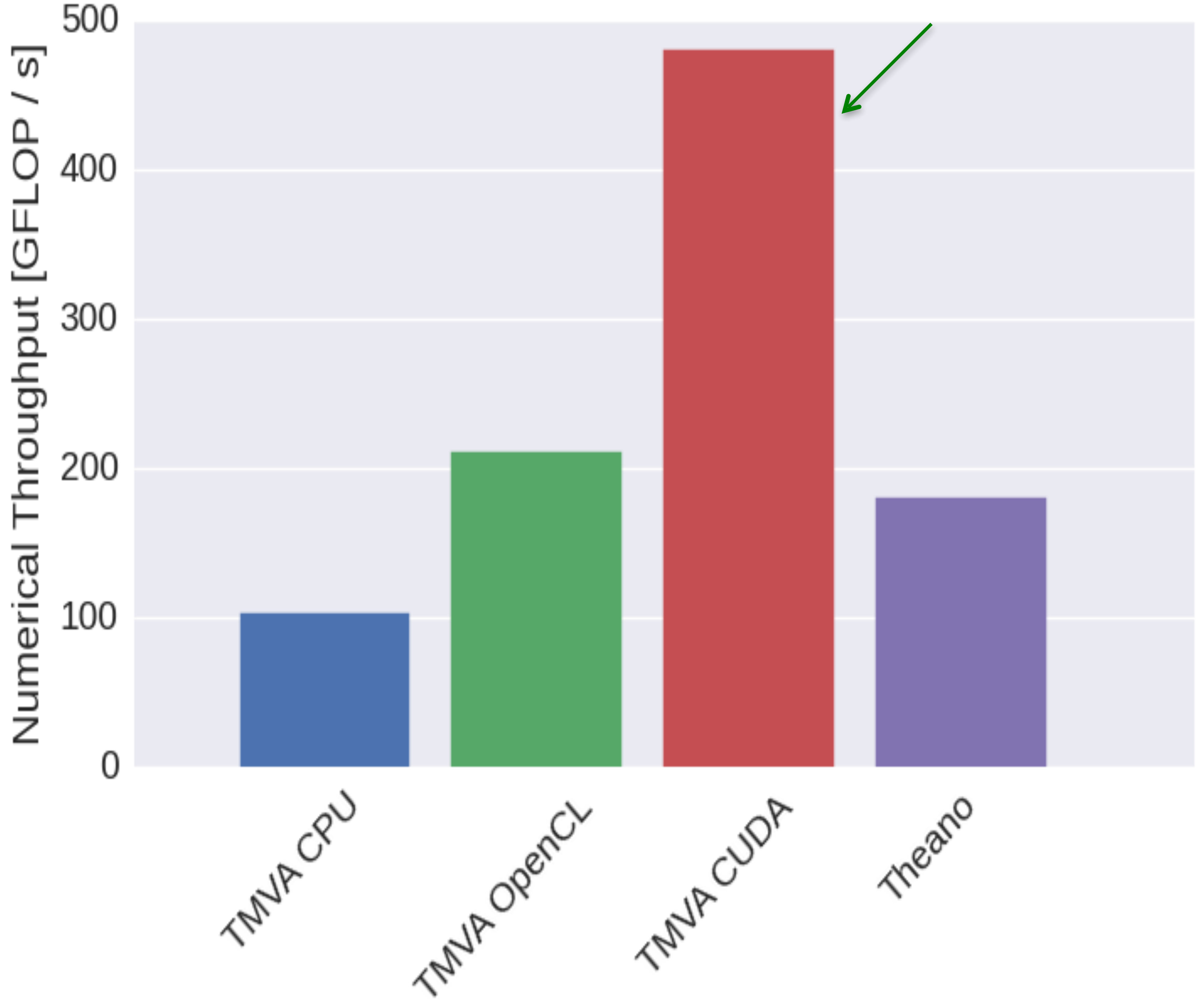
Lorenzo Moneta	Algorithm development, Integration and support
Sergei Gleyzer	Analyzer Tools, Algorithm Development
Omar Zapata Mesa	PyMVA, RMVA, Modularity, Parallelization and Integration
Kim Albertsson	Multi-class for BDT, cross validation/evaluation and support
Stefan Wunsch	Keras Interface, integration, 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 2018)	GAN, VAE and LSTM

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

Backup Slides

Deep Learning Performance

2.7 * Theano



Network:

- 20 input nodes,
- 5 hidden layers with 256 nodes each,
- *tanh* activation functions,
- squared error loss
- batch size = 1024
- Single precision

Training Data:

- Random data from a linear mapping $\mathbb{R}^n \rightarrow \mathbb{R}$

Excellent throughput compared to Theano on same GPU

Example PyMVA with Keras

Define model for Keras

Define the Keras model in Python

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