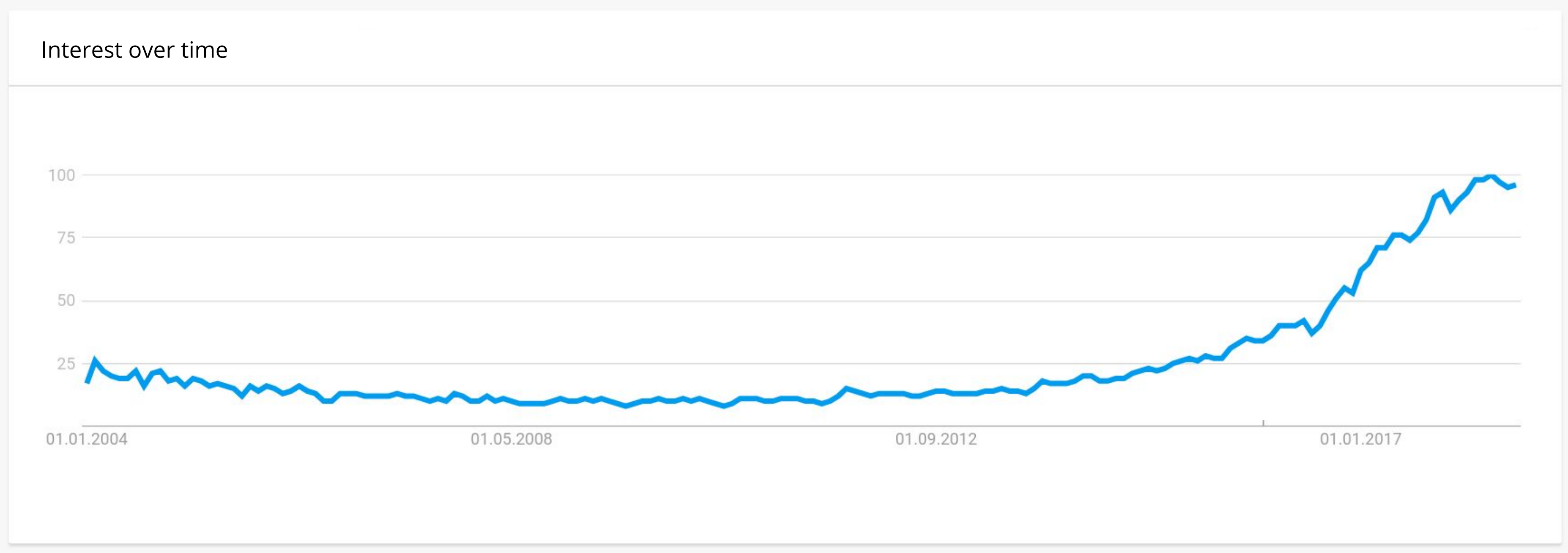
TMVA in the Future

Adapting to the Modern Machine-Learning Landscape

Stefan Wunsch (stefan.wunsch@cern.ch) for the ROOT team

ROOT
Data Analysis Framework
https://root.cern

What has changed?



[Popularity of the term “machine learning” on Google]

The machine-learning workflow

Collect data

Events of physics processes

Energy deposits in detector cells

…

Load data

Transport data from physical device (HDD, file server, …) to your environment (Python runtime, …)

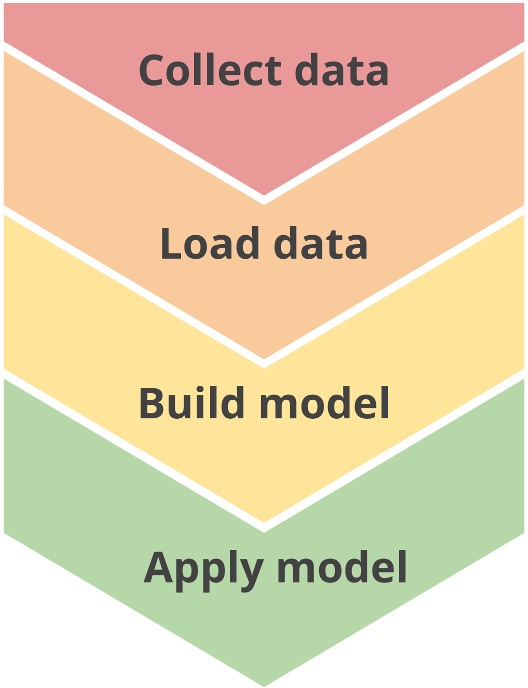
Build model

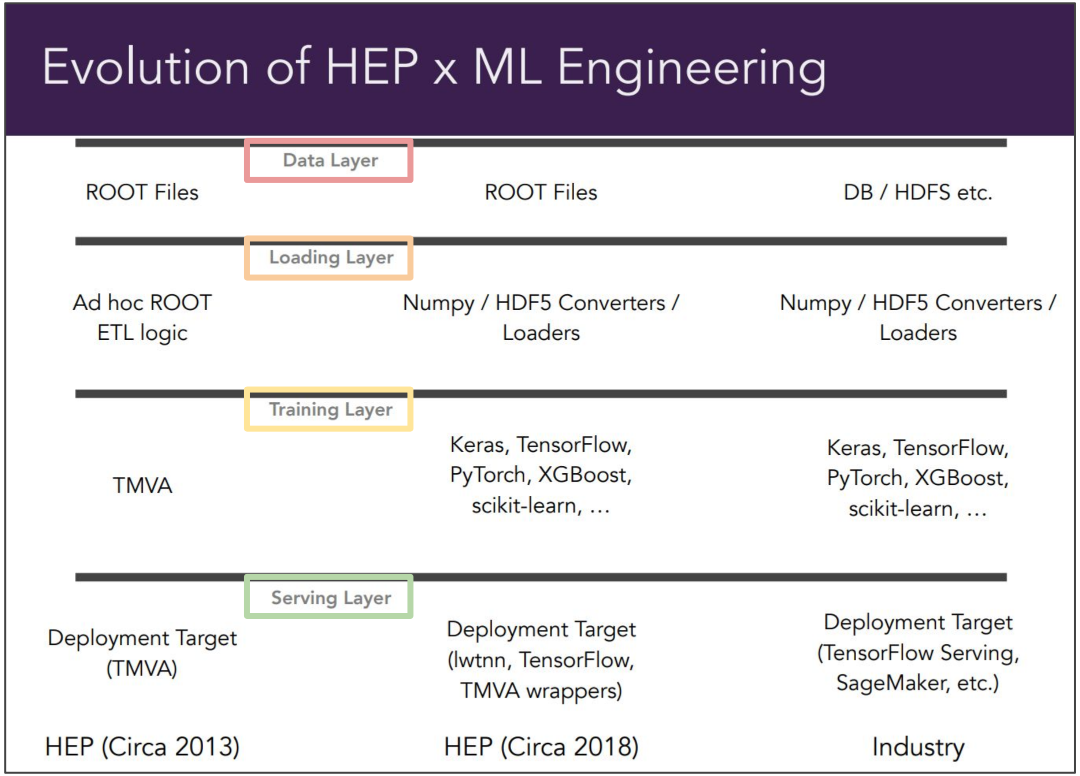
Fit the free parameters of your model to data (weights of a NN, cuts defining trees in a BDT, …)

Apply model

Apply trained model to new data (trigger, event classification, jet tagging, …)

Evolution of the ML landscape





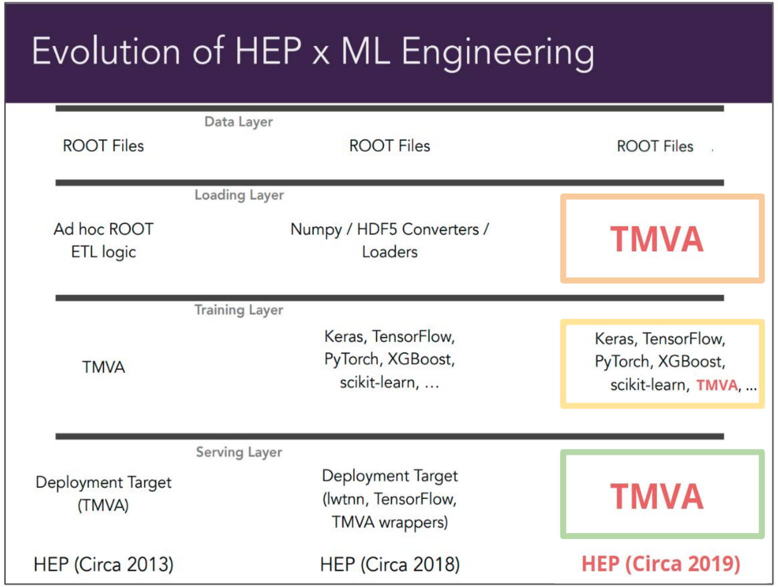
"Overview of ML in HEP" by Luke De Oliveira at the 2nd IML workshop in April 2018

Our vision for TMVA

Evolution of HEP x ML Engineering

	HEP (Circa 2013)	HEP (Circa 2018)	**HEP (Circa 2019)**
Data Layer	ROOT Files	ROOT Files	ROOT Files
Loading Layer	Ad hoc ROOT ETL logic	Numpy / HDF5 Converters / Loaders	**TMVA**
Training Layer	TMVA	Keras, TensorFlow, PyTorch, XGBoost, scikit-learn, …	Keras, TensorFlow, PyTorch, XGBoost, scikit-learn, **TMVA**, …
Serving Layer	Deployment Target (TMVA)	Deployment Target (lwtnn, TensorFlow, TMVA wrappers)	**TMVA**

Key ingredients



Load data

- Load data from many sources
- Filter data
- Define new variables
- Access data easily from Python

Build model

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

Apply model

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

Loading data with RDataFrame

Load data

- Load data from many sources
- Filter data
- Define new variables
- Access data easily from Python

- **Key tool:** ROOT dataframes
- **Sources:**
  - ROOT
  - CSV
  - (xAOD)
  - (SQLite)
- **Remote file access:**
  - xRootD
  - Davix

```python
import ROOT

# Read a remote ROOT file via http
df = ROOT.RDataFrame(
  "Events",
  "http://root.cern.ch/files/NanoAOD_DoubleMuon_CMS2011OpenData.root")

# Reduce on the desired events
df_reduced = df.Filter("nMuon>=2")

# Define needed variables
df_newvar = df_reduced.Define("Muon_pt_leading", "Sorted(Muon_pt)[0]")

# Access data as numpy array
data = df_newvar.AsNumpy()

# Feed to any ML package
import awesome_ml
model = awesome_ml.Model()
model.fit(data)
```

Available in ROOT 6.14

Future

Enrico’s talk about declarative analysis in ROOT

On the way …

Memory adoption of data from C++ containers with numpy arrays

```python
import ROOT
import numpy

# Standard vector from C++ side of the application
x = ROOT.std.vector("float")((1, 2, 3))

# View on data as numpy array via memory adoption (zero copy)
numpy_array = numpy.asarray(x)
```

Available in ROOT 6.14

Read flat TTree as `numpy.array`

```python
import ROOT

# Open remote file via http
file = ROOT.TFile.Open("http://root.cern.ch/files/tmva_class_example.root")

# Get tree with data
tree = file.Get("TreeS")

# Read data in tree as numpy.array
numpy_array = tree.AsMatrix(["var1", "var2", "var3", "var4"])
```

Available in ROOT 6.14

[Enric's talk about PyROOT]

[Talk about memory adoption with numpy]

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

```python
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

On the way …

C++ container for multi-dimensional arrays

C++

```cpp
#include "ROOT/RTensor.hpp"
RTensor<float> x({2, 2});
x(0,0) = 1;
x(1,1) = 1;
cout << x << endl;
// Returns:
// { {1, 0},
//   {0, 1} }
```

Future

Python

```python
import ROOT
import numpy
x = numpy.array([[1, 0],
                 [0, 1]])
y = ROOT.AsTensor(x) # zero copy!
z = numpy.asarray(y) # zero copy!
(x == z).all()
# Returns:
# True
```

Future

- Key feature for
  - design of modern C++ interfaces for ML, e.g., for batches or image data as input
  - interoperability with numpy as C++-side object

[RTensor proposal talk]

Apply trained ML model

Apply model

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

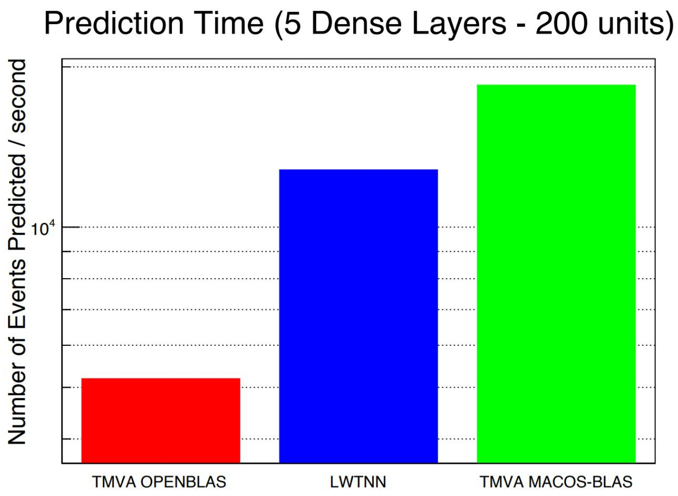
- **Key points:**
  - Fast inference, especially event-by-event
  - Being accessible from C++
  - Loading parameters of externally trained models
  - Interaction with RDataFrame

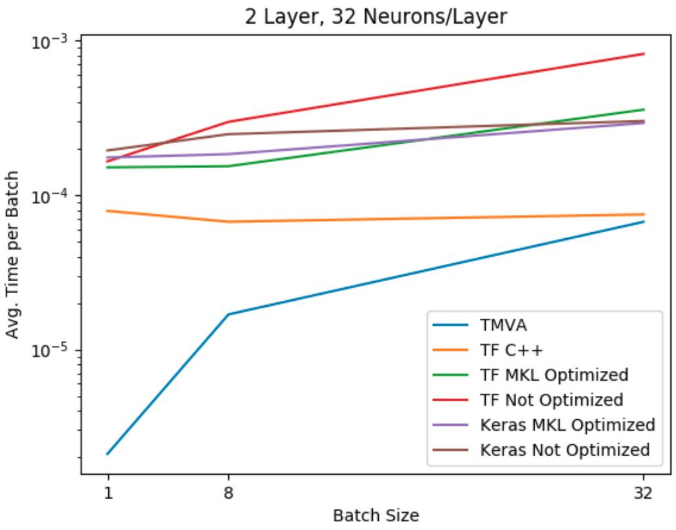
```
int main() {
  // Load TMVA and models trained with external packages
  auto bdt = ROOT::TMVA::BDT("parameters.root");
  auto nn = ROOT::TMVA::Keras("parameters.h5");

  // 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

On the way ...

Fast event-by-event inference with TMVA's neural network implementation





[CHEP talk by Kim Albertsson]

[Work by Alexandru Burlacu]

Summary

Adapting to the modern machine-learning landscape

≡

Provide tools being important in the modern ML workflow in HEP

High-throughput inference

Solid baseline of ML methods

Python and numpy support

TMVA

ROOT dataframe support

Modular features

Modern C++ interfaces

Integration of external packages

Backup

Our vision for TMVA

Evolution of HEP x ML Engineering

	HEP (Circa 2013)	HEP (Circa 2018)	**HEP (Circa 2019)**
Data Layer	ROOT Files	ROOT Files	ROOT Files
Loading Layer	Ad hoc ROOT ETL logic	Numpy / HDF5 Converters / Loaders	**TMVA**
Training Layer	TMVA	Keras, TensorFlow, PyTorch, XGBoost, scikit-learn, …	Keras, TensorFlow, PyTorch, XGBoost, scikit-learn, **TMVA**, …
Serving Layer	Deployment Target (TMVA)	Deployment Target (lwtnn, TensorFlow, TMVA wrappers)	**TMVA**

- **T**oolkit for **M**ulti-**V**ariate **A**nalysis:
  - Focus on *supporting* users using ML in HEP
  - Glue between HEP and ML
- **Modularity**
  - Features as separated tools
  - Mix-and-match with external packages
  - Supports parallelism
- **Interoperability**
  - ML framework independent tools
  - Excellent support equally for C++ and the Python ecosystem