Bringing Keras into basf2 with Frugally Deep

Stefan Wallner
(swallner@mpp.mpg.de)

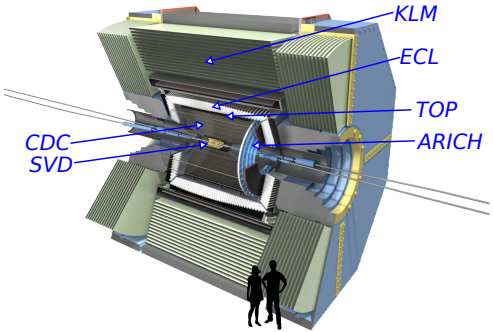
Max Planck Institute for Physics

Machine Learning: Python <-> C++ infrastructure
May 15, 2024

The Project: Particle Identification at Belle II

The Project: Particle Identification at Belle II

- In general six particle species hypotheses considered
  - e, μ, π, K, p, d
- Information from six subdetectors used for PID
  - SVD, CDC, TOP, ARICH, ECL, KLM
  - Different subdetectors cover different kinematic regions
- For each hypothesis h a likelihood is calculated from the information of each subdetector D
  - 36 likelihoods $\mathcal{L}^D(h)$



Pure-likelihood approach

$$\mathcal{L}(h) = \prod_D \mathcal{L}^D(h)$$

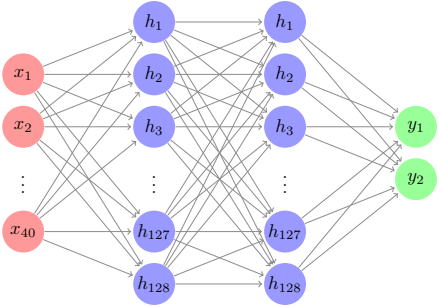
Limitations

- Likelihoods $\mathcal{L}^D(h)$ require modeling
- Correlations among $\mathcal{L}^D(h)$ not taken into account

Our approach

- Combine high-level information:
  - $\mathcal{L}^D(h)$
  - Track momentum: $|\vec{p}|$, $\cos\theta$, ϕ
  - Track charge
- using a neural network to predict a classification variable for the considered hypotheses

- Focus on K/π separation
  - ➥ Network predicts only the probabilities of the K and π hypotheses
- "Simple" Multilayer Perceptron with PReLU/Softmax activation function



The Application: Bringing the Network to Analysts

The Application: Bringing the Network to Analysts

Training

- Training samples can be loaded in Python
- Building and training the network in Python using Keras

Application

- Raw data reconstruction, access to data for analysts, high-level reconstruction, fitting, ...
  provided by basf2 analysis framework
  - Documentation and source code publicly available [Comput.Softw.Big Sci. 3 (2019) 1]
  - Written in C++ with a Python frontend heavily used in analysis
  - ➥ Requires to evaluate the network from C++ code

Implementation

- Use frugally deep library to evaluate trained networks from C++

The Application: Bringing the Network to Analysts

Training

- Training samples can be loaded in Python
- Building and training the network in Python using Keras

Application

- Raw data reconstruction, access to data for analysts, high-level reconstruction, fitting, ... provided by basf2 analysis framework
  - Documentation and source code publicly available [Comput.Softw.Big Sci. 3 (2019) 1]
  - Written in C++ with a Python frontend heavily used in analysis
  - ➥ Requires to evaluate the network from C++ code

Implementation

- Use frugally deep library to evaluate trained networks from C++

The Application: Bringing the Network to Analysts

Training

- Training samples can be loaded in Python
- Building and training the network in Python using Keras

Application

- Raw data reconstruction, access to data for analysts, high-level reconstruction, fitting, ... provided by basf2 analysis framework
  - Documentation and source code publicly available [Comput.Softw.Big Sci. 3 (2019) 1]
  - Written in C++ with a Python frontend heavily used in analysis
  - ➥ Requires to evaluate the network from C++ code

Implementation

- Use frugally deep library to evaluate trained networks from C++

Frugally Deep and Basf2

Frugally Deep

- Open source (MIT license) C++ library developed mainly by Tobias Hermann
- Allows to evaluate (predict) trained Keras/Tensorflow models in C++
- Small header-only library written in modern and pure C++
- Very easy to integrate and use
- Much smaller binary size than linking against TensorFlow
- Active development
  - Currently recommends Tensorflow 2.16.1
  - We use it with Tensorflow 2.13
- Depends on FunctionalPlus (Tobias Hermann), Eigen, and json header-only libraries

Supported Layer Types

- Supports a large set of layer types, but not all keras layer types (see github)
  - `Dense`, `Flatten`, `BatchNormalization`, ...
  - `ReLU`, `PReLU`, `Sigmoid`, `Softmax`, ...
  - `Conv1D/2D`, `MaxPooling1D/2D/3D`, `AveragePooling1D/2D/3D`, ...
  - `Add`, `Concatenate`, `Reshape`...
  - `Attention`, ...
- Interface for prediction from
  - Sequential models
  - Complex computational graphs created with functional API
- Also supports
  - Multiple inputs and outputs, residual connections, nested models, custom layers, ...

Supported Layer Types

- Supports a large set of layer types, but not all keras layer types (see github)
  - `Dense`, `Flatten`, `BatchNormalization`, ...
  - `ReLU`, `PReLU`, `Sigmoid`, `Softmax`, ...
  - `Conv1D/2D`, `MaxPooling1D/2D/3D`, `AveragePooling1D/2D/3D`, ...
  - `Add`, `Concatenate`, `Reshape`...
  - `Attention`, ...
- Interface for prediction from
  - Sequential models
  - Complex computational graphs created with functional API
- Also supports
  - Multiple inputs and outputs, residual connections, nested models, custom layers, ...

Limitations

- Some layer types not supported (see github)
  - `Conv3D`, `Conv2DTranspose`, `RNN`, ...
- No GPU support
- Only single core per prediction
  - Multiple predictions can run in parallel

Limitations

- Some layer types not supported (see github)
  - `Conv3D`, `Conv2DTranspose`, `RNN`, ...
- No GPU support
- Only single core per prediction
  - Multiple predictions can run in parallel

Bring your network into C++ code in 3 steps (see github)

1. Prepare your Keras network
   - Build and train the network
   - Save it to a single file `.keras` file with `model.save`
2. Convert keras network to frugally deep json format
   - Using frugally deep Python converter script
3. Load and evaluate your network in C++

Load the network in C++ [Code]

- Include C++ frugally deep header

```
#include <fdeep/fdeep.hpp>
```

- Load the network from the json file

```
const auto model = fdeep::load_model("fdeep_model.json");
```

- In basf2, "calibrations" are stored in a central data base
  - ➥ Need to load network from json string stored in data base, not from json file
  - ➥ Possible in frugally deep

```
const auto model = fdeep::read_model_from_string(model_json_string);
```

Load the network in C++ [Code]

- Include C++ frugally deep header

```
#include <fdeep/fdeep.hpp>
```

- Load the network from the json file

```
const auto model = fdeep::load_model("fdeep_model.json");
```

- In basf2, "calibrations" are stored in a central data base
  - ➥ Need to load network from json string stored in data base, not from json file
  - ➥ Possible in frugally deep

```
const auto model = fdeep::read_model_from_string(model_json_string);
```

Prepare input [Code]

- Frugally deep model requires frugally deep tensor as input
- Can be created from a `std::vector<float> input`

```
const auto inputFdeep = fdeep::tensor(fdeep::tensor_shape(input.size()), input);
```

- You can use `double` instead of `float` precision by defining

```
#define FDEEP_FLOAT_TYPE double
```

before including frugally deep

Prepare input [Code]

- Frugally deep model requires frugally deep tensor as input
- Can be created from a `std::vector<float> input`

```
const auto inputFdeep = fdeep::tensor(fdeep::tensor_shape(input.size()), input);
```

- You can use `double` instead of `float` precision by defining

```
#define FDEEP_FLOAT_TYPE double
```

before including frugally deep

Evaluate the network [Code]

- Evaluate the network

```
const auto result = m_model->predict({inputFdeep});
```

- and obtain the results

```
probabilities[pdgCode] = result.front().get(fdeep::tensor_pos(outputIndex));
```

Interlude: Pytorch

- First implementation of the neural network was in PyTorch
- Needed to convert PyTorch model to keras to use in with furgally deep
- Could not find a single working PyTorch $\rightarrow$ keras converter
  - ➥ Converted the network by hand to keras

Validation and Performance

Validation and Performance

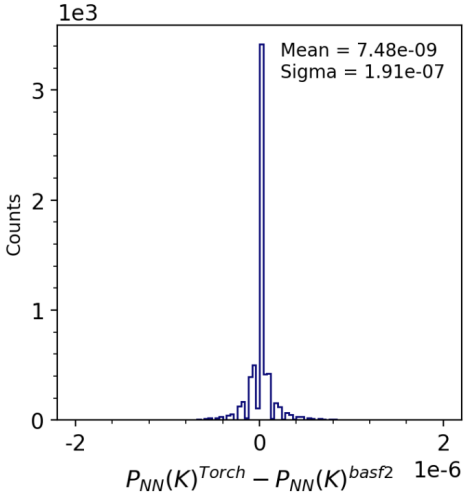
Testing

- Frugally deep verifies that the C++ network works the same as the Python network by running tests when loading a model (can be switched off)
- Comparison of network output in Python and C++ implementation yielded consistent results

Validation and Performance

Testing

- Frugally deep verifies that the C++ network works the same as the Python network by running tests when loading a model (can be switched off)
- Comparison of network output in Python and C++ implementation yielded consistent results



$0 < P_{\mathrm{NN}}(K) < 1$ is neural network output

Validation and Performance

Performance

- No detailed performance studies performed
- Measured the increase in execution time per track evaluation in basf2
  - This includes the network evaluation, but also input gathering, input pre processing, ...
  - For three different network sizes
- Finally used medium-sized network was sufficiently fast evaluated

	Small	Medium	Large
Node per hidden layer	64	128	640
Parameters	7k	20k	400k
Execution time	0.07 ms/call	0.07 ms/call	0.20 ms/call
Increase of total execution time[1]	$\lessapprox 7\,\%$	$\lessapprox 7\,\%$	$\lessapprox 17\,\%$

[1] Compared to execution time of minimal analysis script. Relative increase in total execution time for full-scale analysis scripts is even smaller.

Conclusion

Frugally deep

- Very easy to use header-only library to evaluate keras models in C++
- Small coding overhead $\Rightarrow$ fast implementation
- Supports basic layer types from keras and also some advanced types
  - However, not all keras layer types supported
- Evaluation sufficiently fast for our requirements
  - No GPU or multi-threading support
  - No detailed performance studies performed
- Used for two projects (PIDNN, BelleNbarMVAModule) in basf2

➥ From our applications, frugally deep is very well suited for "small" and simple networks
(might also be well suited for large and complex networks)

Backup

Outline