New developments for TMVA SOFIE

Ahmat Mahamat Hamdan
EP-SFT
Supervisor: Lorenzo Moneta
● Toolkit for Multivariate Analysis
● Machine Learning library integrated in ROOT
● Provides support for training, testing and evaluating machine learning models
● Development shifted to deep learning and BDT inference
ONNX

- Open Neural Networks Exchange
- Specification for machine learning models.

Advantages

- Interoperability between machine learning frameworks
- Can have implementation with hardware optimization
**SOFIE**

- System for Optimized Fast Inference Code Emit

A deep learning inference engine that
- Takes ONNX, Pytorch and Keras files as input
and produces as output
- a C++ header file with minimal dependency.
- optionally a data file for the weights

**Motivation**

- ML ecosystem mostly focuses on model training.
- Machine Learning Inference & deployment is often neglected
- Inference in Tensorflow & PyTorch
  - Supports only their own model
  - Usage of C++ environment is difficult
  - Heavy dependency
- Inference in ONNX (Open Neural Network Exchange)
  - Can use ONNXRuntime by Microsoft
  - Large dependency
  - Frequent C++ API changes
  - Difficult to integrate in HEP applications
    - Control of libraries, threads
    - Not optimized for single event evaluation
Goals of the project

● Add support for missing deep learning operators
● Implement the operators in the code generation
● Optimize the implementation on CPU with BLAS
### Supported deep learning operators

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Supported Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully connected neural networks</td>
<td>Gemm</td>
</tr>
<tr>
<td>Upsampling (Convolution)</td>
<td>Conv</td>
</tr>
<tr>
<td>Downsampling (Transposed convolution)</td>
<td>ConvTranspose</td>
</tr>
<tr>
<td>Pooling</td>
<td>MaxPool, AvergePool and GlobalAveragePool</td>
</tr>
<tr>
<td>Recurrents Neural Networks</td>
<td>RNN, LSTM and GRU</td>
</tr>
<tr>
<td>Normalization operators</td>
<td>BatchNormalization and LayerNormalization</td>
</tr>
<tr>
<td>Activation functions</td>
<td>Relu, LeakyRelu, Selu, Tanh, Softmax, Sigmoid</td>
</tr>
</tbody>
</table>

- CERN Short Term Internship
- Google Summer of Code (GSoC) 2021 with CERN-HSF
## Supported tensor manipulation operators

<table>
<thead>
<tr>
<th>Supported tensor manipulation operators</th>
<th>Supported operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensor Indexing and slicing</td>
<td>Slice and <strong>Gather</strong></td>
</tr>
<tr>
<td>Tensor Reshaping operators</td>
<td>Reshape, Flatten and Transpose</td>
</tr>
<tr>
<td>Concatenation</td>
<td><strong>Concat</strong></td>
</tr>
<tr>
<td>Casting</td>
<td><strong>Cast</strong></td>
</tr>
<tr>
<td>Unary operators</td>
<td><strong>Reciprocal, Sqrt, Neg</strong> and <strong>Exp</strong></td>
</tr>
<tr>
<td>Binary operators</td>
<td>Add, Sub, Mul, Div and Pow</td>
</tr>
<tr>
<td>Reduction operators</td>
<td><strong>ReduceMean, ReduceSumSquare and ReduceProd</strong></td>
</tr>
<tr>
<td>Others</td>
<td><strong>Shape</strong></td>
</tr>
</tbody>
</table>

---

* CERN Short Term Internship
### Some missing operators

<table>
<thead>
<tr>
<th>Category</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalization</td>
<td>MeanVarianceNormalization, GroupNormalization</td>
</tr>
<tr>
<td>Regularization</td>
<td>Dropout</td>
</tr>
<tr>
<td>Pooling</td>
<td>GlobalMaxpool</td>
</tr>
<tr>
<td>Activation functions</td>
<td>PRelu, Elu and ThresholdedRelu</td>
</tr>
</tbody>
</table>

And more in the [ONNX operators documentation](#).
Implemented operators

Transposed convolution

- Perform an upsampling operation (reverse of the Convolution)

![Transposed convolution diagram](image)
Fuse operators

- Fuse multiples nodes into a single node in the computational graph
- Implemented for ConvTranspose and Add
- The input of Add is added to the bias of ConvTranspose
Layer Normalization

- Normalize the output of a hidden layer by computing the mean and the standard deviation along a given axis
- Implemented for up to 5d tensors

\[
z = \frac{x - \mu}{\sigma}
\]

\[
\mu = \text{Mean} \\
\sigma = \text{Standard Deviation}
\]

1d Layer normalization

3d Layer normalization
Implemented operators

**Gather**

- Takes elements from an array and concatenate them
- Similar to numpy.take
- Useful for embedding

```python
>>> import numpy as np
>>> x = np.arange(10) * 2
>>> x
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
>>> indices = [0, 2, 3, 8]
>>> np.take(x, indices)
array([ 0,  4,  6, 16])
```
Implementation of the operators

**Parse the operator from the onnx file**

- Define a function to parse the corresponding node from the onnx graph
- Parse the attributes, the inputs and outputs
- Returns a unique pointer to the operator

```cpp
ParserFuncSignature ParseLayerNormalization = ([](RModelParser_ONNX &parser, const onnx::NodeProto &nodeproto)
    -> std::unique_ptr<ROperator> {  
    ETensorType input_type = ETensorType::UNDEFINED;
    const std::string input_name = nodeproto.input(0);

    ...

    std::unique_ptr<ROperator> op;

    ...
    return op;
});
```
Implementation of the operators

Code generation

- Define an operator class that inherits from ROperator
- Override the pure virtual member functions

```
template <typename T>
class ROperator_LayerNormalization : public ROperator {

  ROperator_LayerNormalization () {}  

  std::vector<std::vector<size_t>> ShapeInference (std::vector<std::vector<size_t>>) override;

  std::vector<ETensorType> TypeInference (std::vector<ETensorType>) override;

  void Initialize (RModel &) override;

  std::string GenerateInitCode () override;

  std::string Generate (std::string) override;

  std::vector<std::string> GetBlasRoutines () override;

  std::vector<std::string> GetStdLibs () override;
}
```
Improvement of the ONNX parser

ONNX parser

With the old parser
- Can’t check if an operator is supported
- Can’t get the list of supported operators
- Merging a commit that add a new operator introduces a lot of merge conflicts
- Difficult to develop new operators

With the new parser
- Use registry design pattern for the functions that parse the operators
- Faster compile time by using the pointer to implementation idiom for the map containing the functions
- Check at runtime if an operator is supported
- Get the list of supported operators
- Plugin a custom operator
- Replace a supported operator with a custom implementation
- Develop new operators with a root macro
Improvement of the ONNX parser

**New RModelParser_ONNX class**

- Use a `std::function` object to parse an operator and store it in a unique pointer to ROperator

```cpp
using ParserFuncSignature = std::function<std::unique_ptr<ROperator>(RModelParser_ONNX & /*parser*/, const onnx::NodeProto & /*nodeproto*/)>;
```

- Store the operators in a map

```cpp
std::unique_ptr<OperatorsMapImpl> fOperatorsMapImpl;
```

- Register an operator to the parser

```cpp
void RegisterOperator(const std::string &name, ParserFuncSignature func);
```

- Parse a node from the graph

```cpp
std::unique_ptr<ROperator> ParseOperator(const size_t /*index*/, const onnx::GraphProto & /*graphproto*/, const std::vector<size_t> & /*nodes*/);
```

- Parse a model from a file

```cpp
RModel Parse(std::string filename, bool verbose = false);
```
Improvement of the ONNX parser

Developing new operators with a root macro

1. Include the header files
#include "TMVA/RModel.hxx"
#include "TMVA/RModelParser_ONNX.hxx"
#include "onnx_proto3.pb.h"

2. Define the operator class
using namespace TMVA::Experimental::SOFIE;
template <typename T> struct ROperator_Sign : public ROperator

3. Define a function to parse the node
ParserFuncSignature ParseSign = [](RModelParser_ONNX &parser,
const onnx::NodeProto &nodeproto) ->
std::unique_ptr<ROperator>;

4. Main function
void SofieSign () {
    RModelParser_ONNX parser;
    // Register the Sign operator
    parser.RegisterOperator("Sign", ParseSign);
    // Parse the model
    RModel model = parser.Parse("Sign.onnx");
    // Generate and save
    model.Generate();
    model.OutputGenerated();
}

5. Execute the macro to generate the code
>>> root .x SofieSign.C

Link to a working example
Tensor broadcasting

- **Unidirectional broadcasting**: Broadcasting two tensors to the common shape
- **Multidirectional broadcasting**: Broadcast more than two tensors to the common shape
- Implemented according to numpy's broadcasting rules

Broadcast a vector of size 1 to 3

Broadcast a 1x3 matrix to 4x3
### Implementation

- **UnidirectionalBroadcastShape** returns the common shape of two tensors
  \[
  \text{std::vector}\langle\text{size}_t\rangle \text{ UnidirectionalBroadcastShape} (\text{std::vector}\langle\text{size}_t\rangle, \text{std::vector}\langle\text{size}_t\rangle);
  \]

- **MultidirectionalBroadcastShape** returns the common shape of a list of tensors
  \[
  \text{std::vector}\langle\text{size}_t\rangle \text{ MultidirectionalBroadcastShape} (\text{std::vector}\langle\text{std::vector}\langle\text{size}_t\rangle\rangle);
  \]

- **BroadcastTensor** broadcasts a tensor to the given shape
  \[
  \text{template<typename T>} \text{T}^* \text{ BroadcastTensor} (\text{const T}^* \text{ data}, \text{const std::vector}\langle\text{size}_t\rangle& \text{ shape}, \text{const std::vector}\langle\text{size}_t\rangle& \text{ targetShape})
  \]

- **UnidirectionalBroadcastTensor** handles the case where the number of dimensions are different
  \[
  \text{template<typename T>} \text{T}^* \text{ UnidirectionalBroadcast} (\text{const T}^* \text{ data}, \text{const std::vector}\langle\text{size}_t\rangle& \text{ shape}, \text{const std::vector}\langle\text{size}_t\rangle& \text{ targetShape})
  \]
Patches

- Optimize Conv for large batch size
  
  Included in [tmva][sofie] Fix ConvTranspose for multiple channels

- Fix the Expand operator and add the tests
  
  Included in [TMVA][SOFIE] Expand ONNX Operator implemented with the corresponding unit tests
Benchmark

- Convolutional neural networks

- Transposed convolution

- Recurrent Neural Networks (RNNs)

- Benchmark configuration
  - Macbookpro running MacOS12 ARM
  - ROOT master
  - ONNX Runtime version 1.13.1
  - Sofie Benchmark PR
### Pull requests

<table>
<thead>
<tr>
<th>PR</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>[TMVA][SOFIE] Add ConvTranspose operator</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Fix the Softmax operator</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Broadcast bias of Conv and ConvTranspose operators</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Broadcast bias of Conv and ConvTranspose operators</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Add basic unary operators</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Fix warnings in the Softmax operator</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE]Fuse Conv or ConvTranspose and Add</td>
<td>Closed*</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Broadcasting</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] ONNX parser with registry like pattern</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Add LayerNormalization operator</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Bring back Flatten, Squeeze, Unsqueeze and fix fused operators</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Fix SOFIE tests</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Register operators in the constructor of RModelParser ONNX</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Optimize Conv for large batch size</td>
<td>Closed*</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Add Gather operator</td>
<td>Merged</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Use GetBlasRoutines and GetStdLibs</td>
<td>Open</td>
</tr>
<tr>
<td>[TMVA][SOFIE] Nary operators with multidirectional broadcasting</td>
<td>Open</td>
</tr>
</tbody>
</table>

*Merged to master as part of another commit/PR*
In brief

- Implemented the following operators: ConvTranspose, Unary operators (Reciprocal, Sqrt, Neg and Exp), LayerNormalization, Gather and Nary operators (Max, Min, Mean and Sum).
- Fixed Softmax and Expand operators
- Implemented tensor broadcasting according to numpy’ broadcasting rules
- Restructured the ONNX parser
- Optimized the Conv operator for large batch sizes
Future plans and challenges

Future plans

- Parsing the GNNs (WIP)
- Add support for dynamic GNNs
- Add support for more ONNX operators

Challenges

- Fusing complex operators or multiple nodes into a single operator
- Broadcasting tensors inplace
- Reuse the implemented operators in the GNNs
I'm grateful to

- My supervisor Lorenzo Moneta for giving me this opportunity and guiding me throughout the internship
- The EP-SFT group especially anyone that I had the chance to interact with

I enjoyed working on my project and I hope to comeback at CERN for the long term.
Any questions?