TMVA Deep Learning Developments - Inference Code Generation for Recurrent Neural Networks

Google Summer of Code 2021

Ahmat Hamdan

Mentors: Lorenzo Moneta, Sitong An

Project goal

The goal of my project develop the recurrent neural networks operators as defined by the ONNX (Open Neural Network Exchange) standards in the code generation format for the TMVA SOFIE (System for Fast-Inference Code Emit). This was done successfully over the course of the summer.

SOFIE overview

SOFIE is a deep learning inference engine that

- Takes ONNX files as input

and

- Produces a C++ script as output.

Its

- Currently under active development in the ROOT/TMVA team at CERN.

SOFIE today

- Parsing models from ONNX.
- Serialisation of models.
- Support for feedforward neural networks.
- Support for convolutional neural networks.
- and more.

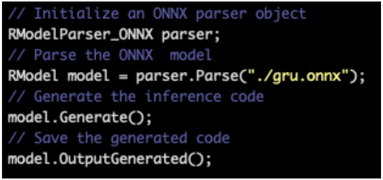
Tasks

- Development of the Recurrent Neural Network (RNN) operator.
- Development of the Long Short-term Memory (LSTM) operator.
- Development of the Gated Recurrent Unit (GRU) operator.

Implementation

- Parse the node of the RNN operator from the ONNX Graph.
- Infer the type and the shape of the output tensors.
- Check the atributes and the input tensors.
- Broadcasting the input tensors when needed.
- Generate the code implementing the forward pass of the RNN operator.

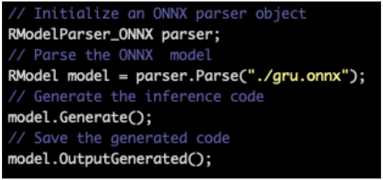
A practical example



```
// Initialize an ONNX parser object
RModelParser_ONNX parser;
// Parse the ONNX  model
RModel model = parser.Parse("./gru.onnx");
// Generate the inference code
model.Generate();
// Save the generated code
model.OutputGenerated();
```

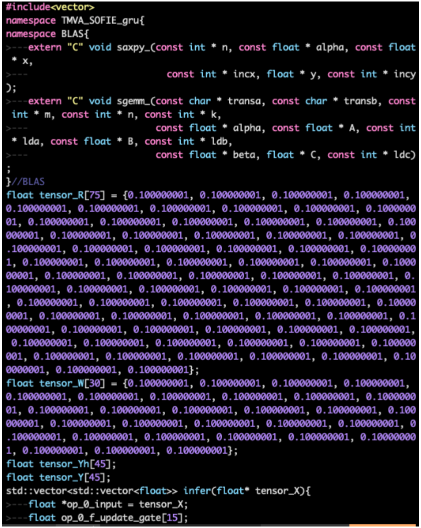
Generate the header file

A practical example



```
// Initialize an ONNX parser object
RModelParser_ONNX parser;
// Parse the ONNX  model
RModel model = parser.Parse("./gru.onnx");
// Generate the inference code
model.Generate();
// Save the generated code
model.OutputGenerated();
```

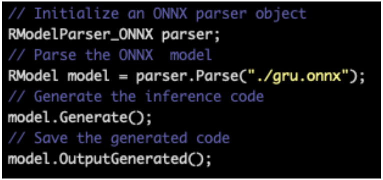
Generate the header file



```
#include<vector>
namespace TMVA_SOFIE_gru{
namespace BLAS{
>---extern "C" void saxpy_(const int * n, const float * alpha, const float
 * x,
>---                        const int * incx, float * y, const int * incy
);
>---extern "C" void sgemm_(const char * transa, const char * transb, const
 int * m, const int * n, const int * k,
>---                      const float * alpha, const float * A, const int
 * lda, const float * B, const int * ldb,
>---                      const float * beta, float * C, const int * ldc)
;
}//BLAS
float tensor_R[75] = {0.100000001, 0.100000001, 0.100000001, 0.100000001,
0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1000000
01, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100
000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0
.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10000000
1, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1000
00001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.
100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001
, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10000
0001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1
00000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001,
 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000
001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10
0000001, 0.100000001, 0.100000001};
float tensor_W[30] = {0.100000001, 0.100000001, 0.100000001, 0.100000001,
0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1000000
01, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100
000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0
.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10000000
1, 0.100000001, 0.100000001, 0.100000001};
float tensor_Yh[45];
float tensor_Y[45];
std::vector<std::vector<float>> infer(float* tensor_X){
>---float *op_0_input = tensor_X;
>---float op_0_f_update_gate[15];
```

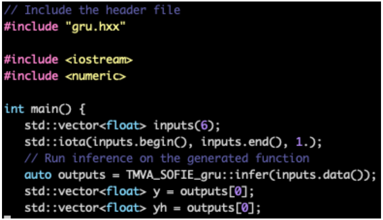
Generated code

A practical example



```
// Initialize an ONNX parser object
RModelParser_ONNX parser;
// Parse the ONNX  model
RModel model = parser.Parse("./gru.onnx");
// Generate the inference code
model.Generate();
// Save the generated code
model.OutputGenerated();
```

Generate the header file

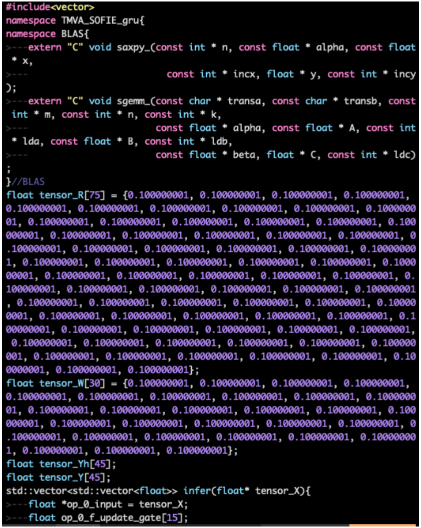


```
// Include the header file
#include "gru.hxx"

#include <iostream>
#include <numeric>

int main() {
   std::vector<float> inputs(6);
   std::iota(inputs.begin(), inputs.end(), 1.);
   // Run inference on the generated function
   auto outputs = TMVA_SOFIE_gru::infer(inputs.data());
   std::vector<float> y = outputs[0];
   std::vector<float> yh = outputs[0];
```

Run inference



```
#include<vector>
namespace TMVA_SOFIE_gru{
namespace BLAS{
>---extern "C" void saxpy_(const int * n, const float * alpha, const float
 * x,
>---                         const int * incx, float * y, const int * incy
);
>---extern "C" void sgemm_(const char * transa, const char * transb, const
 int * m, const int * n, const int * k,
>---                       const float * alpha, const float * A, const int
 * lda, const float * B, const int * ldb,
>---                       const float * beta, float * C, const int * ldc)
;
}//BLAS
float tensor_R[75] = {0.100000001, 0.100000001, 0.100000001, 0.100000001,
0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1000000
01, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100
000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0
.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10000000
1, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1000
00001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.
100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001
, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10000
0001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1
00000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001,
 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000
001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10
0000001, 0.100000001, 0.100000001};
float tensor_W[30] = {0.100000001, 0.100000001, 0.100000001, 0.100000001,
0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.1000000
01, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100
000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0
.100000001, 0.100000001, 0.100000001, 0.100000001, 0.100000001, 0.10000000
1, 0.100000001, 0.100000001, 0.100000001};
float tensor_Yh[45];
float tensor_Y[45];
std::vector<std::vector<float>> infer(float* tensor_X){
>---float *op_0_input = tensor_X;
>---float op_0_f_update_gate[15];
```

Generated code

Further developments

What's left

- Adding the tests for the LSTM operator.
- Adding the tests for the GRU operator.

Post GSoC

- Benchmarking the RNNs operators against ONNX runtime.
- Improvements to SOFIE.

Thank you!

https://github.com/axmat/TMVAFastInferencePrototype

https://github.com/axmat/TMVAInference