TMVA Deep Learning Developments - Inference Code Generation for Batch Normalization

Mentors: Sitong An, Ashish Kshirsagar, Lorenzo Moneta

GSoC 2021, Aaradhya Saxena

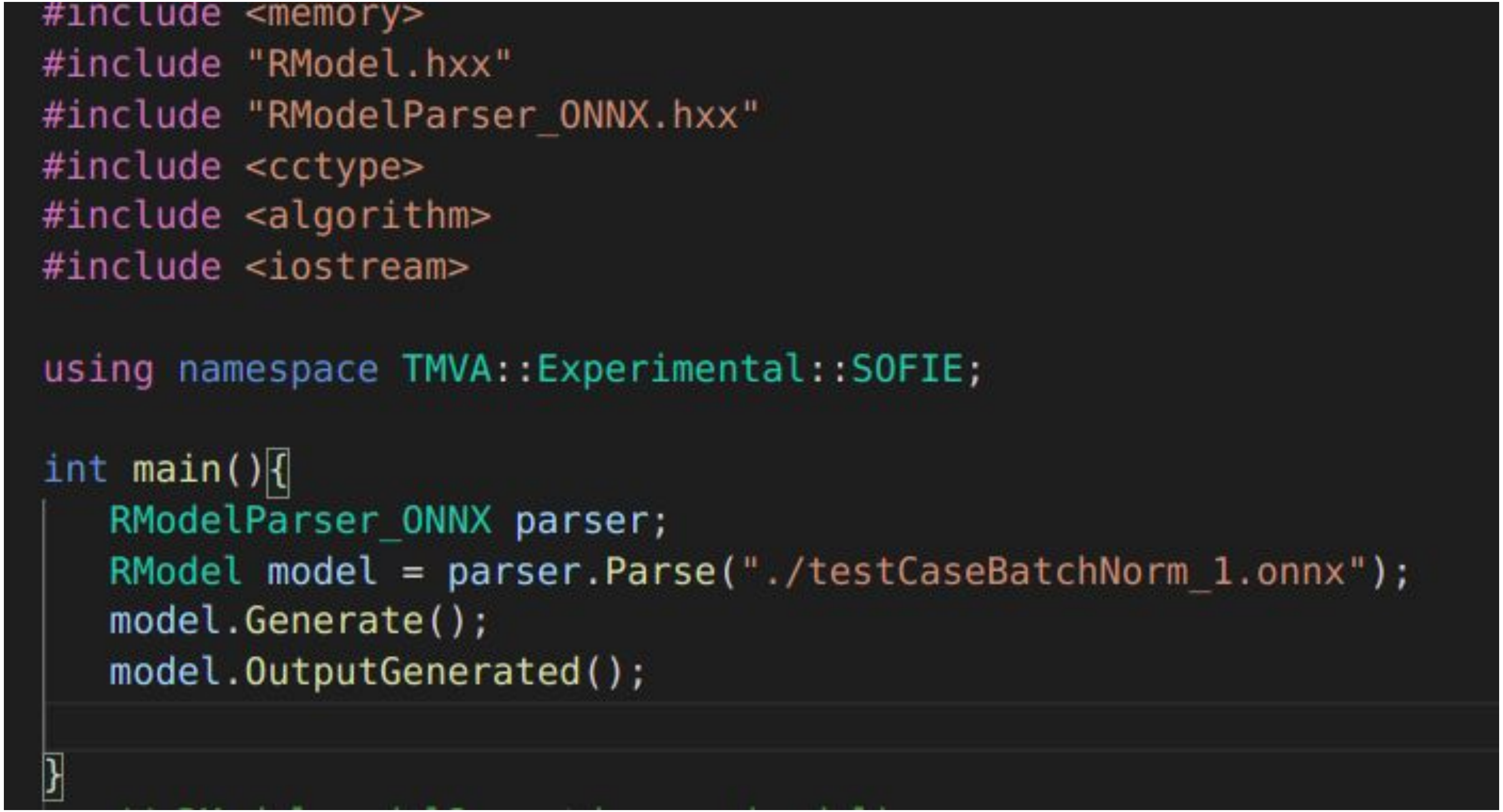
Introduction

Development of a fast inference system in TMVA that takes takes a trained ONNX model as input and produces compilation-ready standalone C++ scripts as output.

These scripts will then provide users an easy way to deploy their deep learning models in their physics software and analysis frameworks.

GOAL

Implementation of Batch Normalization and Instance Normalization operator as defined by the ONNX operator standards in the code generation format.

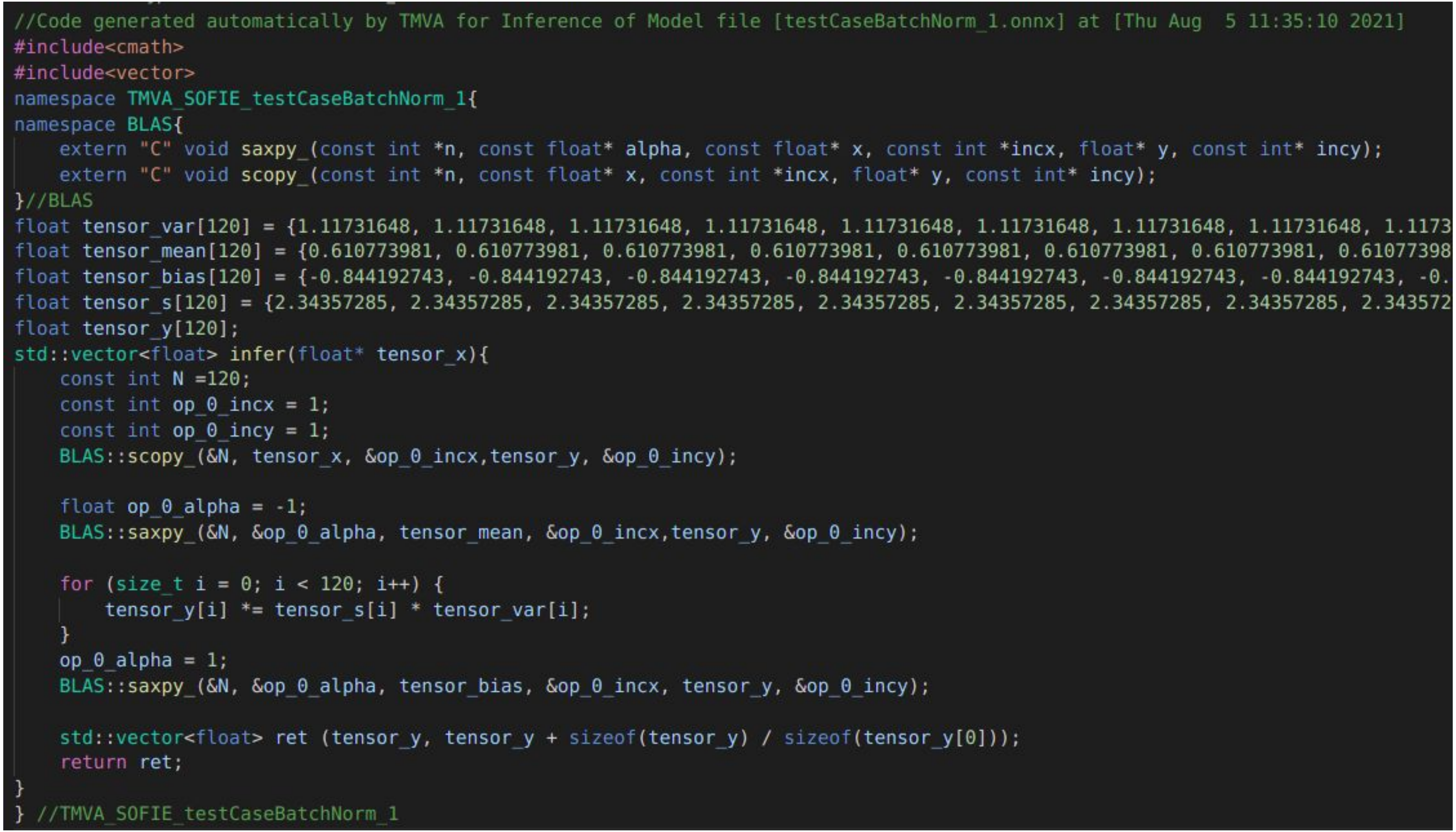


```
#include <memory>
#include "RModel.hxx"
#include "RModelParser_ONNX.hxx"
#include <cctype>
#include <algorithm>
#include <iostream>

using namespace TMVA::Experimental::SOFIE;

int main(){
   RModelParser_ONNX parser;
   RModel model = parser.Parse("./testCaseBatchNorm_1.onnx");
   model.Generate();
   model.OutputGenerated();

}
```

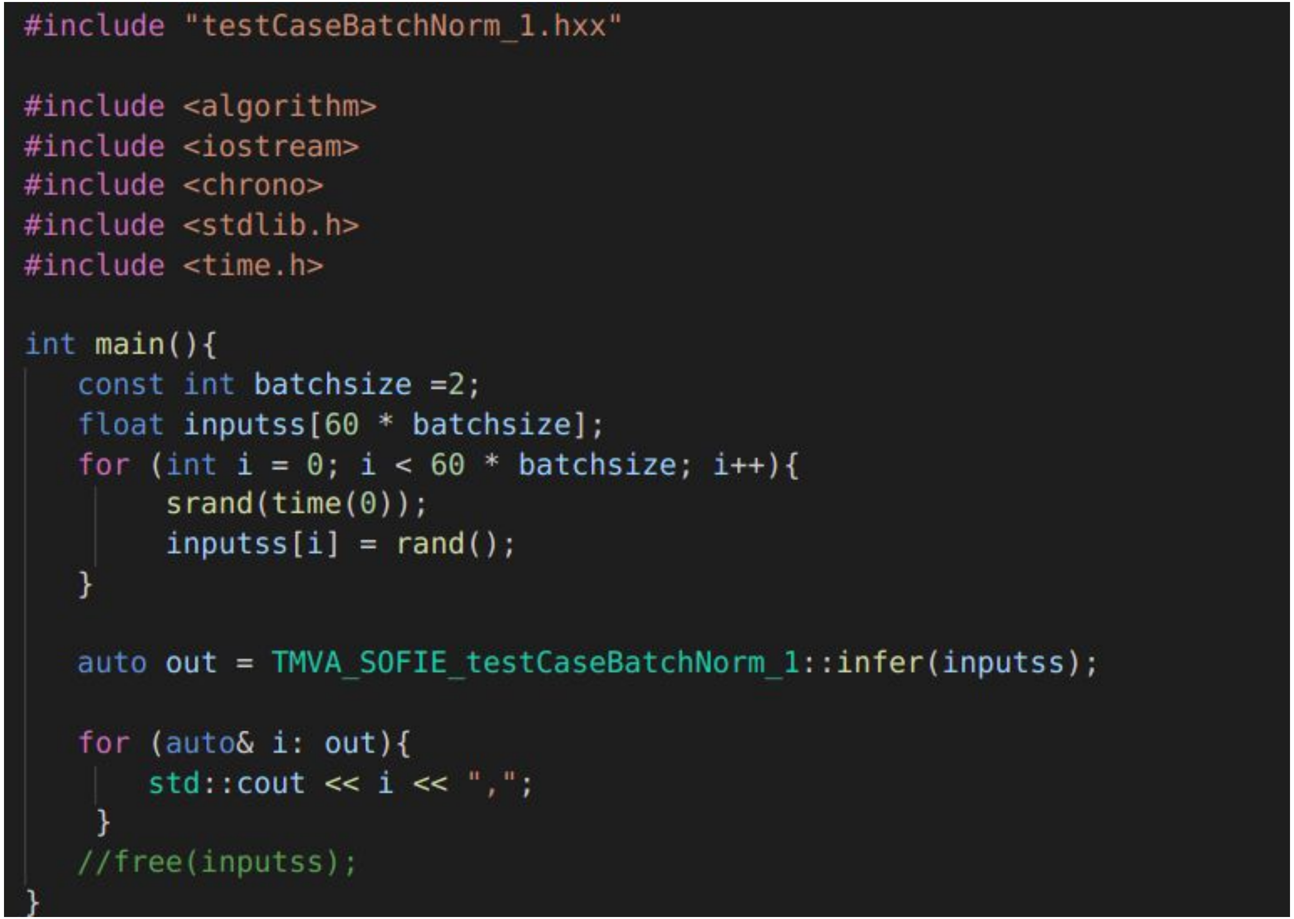


```
//Code generated automatically by TMVA for Inference of Model file [testCaseBatchNorm_1.onnx] at [Thu Aug  5 11:35:10 2021]
#include<cmath>
#include<vector>
namespace TMVA_SOFIE_testCaseBatchNorm_1{
namespace BLAS{
    extern "C" void saxpy_(const int *n, const float* alpha, const float* x, const int *incx, float* y, const int* incy);
    extern "C" void scopy_(const int *n, const float* x, const int *incx, float* y, const int* incy);
}//BLAS
float tensor_var[120] = {1.11731648, 1.11731648, 1.11731648, 1.11731648, 1.11731648, 1.11731648, 1.11731648, 1.11731648, 1.1173
float tensor_mean[120] = {0.610773981, 0.610773981, 0.610773981, 0.610773981, 0.610773981, 0.610773981, 0.610773981, 0.61077398
float tensor_bias[120] = {-0.844192743, -0.844192743, -0.844192743, -0.844192743, -0.844192743, -0.844192743, -0.844192743, -0.
float tensor_s[120] = {2.34357285, 2.34357285, 2.34357285, 2.34357285, 2.34357285, 2.34357285, 2.34357285, 2.34357285, 2.343572
float tensor_y[120];
std::vector<float> infer(float* tensor_x){
    const int N =120;
    const int op_0_incx = 1;
    const int op_0_incy = 1;
    BLAS::scopy_(&N, tensor_x, &op_0_incx,tensor_y, &op_0_incy);

    float op_0_alpha = -1;
    BLAS::saxpy_(&N, &op_0_alpha, tensor_mean, &op_0_incx,tensor_y, &op_0_incy);

    for (size_t i = 0; i < 120; i++) {
        tensor_y[i] *= tensor_s[i] * tensor_var[i];
    }
    op_0_alpha = 1;
    BLAS::saxpy_(&N, &op_0_alpha, tensor_bias, &op_0_incx, tensor_y, &op_0_incy);

    std::vector<float> ret (tensor_y, tensor_y + sizeof(tensor_y) / sizeof(tensor_y[0]));
    return ret;
}
} //TMVA_SOFIE_testCaseBatchNorm_1
```



```
#include "testCaseBatchNorm_1.hxx"

#include <algorithm>
#include <iostream>
#include <chrono>
#include <stdlib.h>
#include <time.h>

int main(){
   const int batchsize =2;
   float inputss[60 * batchsize];
   for (int i = 0; i < 60 * batchsize; i++){
        srand(time(0));
        inputss[i] = rand();
   }

   auto out = TMVA_SOFIE_testCaseBatchNorm_1::infer(inputss);

   for (auto& i: out){
       std::cout << i << ",";
    }
   //free(inputss);
}
```

TASKS

- ✓ Batch-Normalization Operator
  - ✓ ROperator_BN()
  - ✓ TypeInference()
  - ✓ ShapeInference()
  - ✓ Initialize()
  - ✓ Generate()
  - ✓ make_ROperator_BN()
  - ✓ Write tests
  - ✓ Optimize using blas
- ✓ Instance-Normalization Operator
  - ✓ ROperator_IN()
  - ✓ TypeInference()
  - ✓ ShapeInference()
  - ✓ Initialize()
  - ✓ Generate()
  - ✓ make_ROperator_IN()
  - ✓ Write tests
  - ✓ Optimize using blas

Thank you !

Report:

https://docs.google.com/document/d/1qtoglXoyIfsngJVpidsTegpQlybpxd6mFH6doP4myLo/edit?usp=sharing

https://github.com/sitongan/TMVAFastInferencePrototype/pull/5

https://github.com/sitongan/TMVAFastInferencePrototype/pull/7