Development of 3D CNN in TMVA

About Me

Basic Information

Name	Surya S Dwivedi
University	The University of Texas at Austin
Major	Computer Science
Email	surya2191997@gmail.com
Github Handle	surya2191997
Blog	surya2191997.github.io
Timezone	USA, Central Standard Time (CST)
Contact No	+1(512) 228 9336

Background

I am currently a Master's student at the Department of Computer Science, The University of Texas at Austin where I intend to specialize in Systems and Machine Learning. Prior to joining UT, I graduated with a Bachelor's in CS from Indian Institute of Technology, Kharagpur. I understand the TMVA codebase very well, especially the DNN part of it, as I was a GSoC student in TMVA the previous year, and successfully implemented and tested two new modules(LSTM and GRU) in TMVA.

I have been programming for about 5 years now. I am comfortable with programming in C++ and Python. I have regularly been programming in these languages as part of hackathons, course projects and internships. My educational background is sufficient for the projects, I understand convolutions really well, both mathematically and as applied to Machine Learning. Here is a link to my resume: Resume

Platform details

OS : MacOS X Yosemite (10.10.5)

Editor : Sublime Text 3

Project Overview

This project is about the development of 3D CNN functionality in TMVA. I shall be developing both 3D convolution as well as 3D pooling layers. 3D CNNs have very promising applications in particle physics, such as imaging calorimetry and particle tracking, allowing physicists to use new techniques to identify particles and search for new physics. In addition they can be used as Generative Adversarial Networks for fast imaging detector simulations.

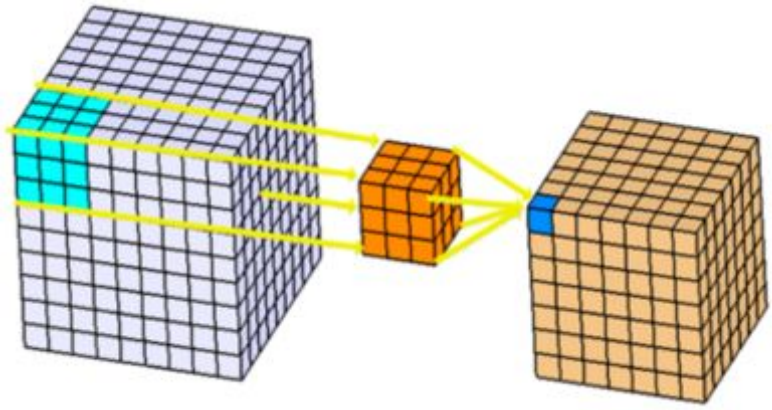
Deliverables

- Production-ready 3D CNNs implementation
- Convolution code optimization for CPU (For 2D as well as 3D)
- GPU support for training
- Testing on a real world dataset
- Numerical gradient check

Details of Project

3D CNN overview

A 3D CNN is nothing but 3D convolution and 3D pooling layers stacked on top of each other with activation functions in between them. A convolution, mathematically speaking, is a function which takes in as input a signal and an impulse and outputs the 'response' of the signal under the impulse. In ML, we usually deal with only discrete convolutions. In a discrete convolution we have a filter (impulse) which is applied to every position of the input(signal) to produce an output. To apply a filter basically means, taking the sum of products of all input values and filter values at corresponding positions. Now what makes a convolution 1D, 2D or 3D is what dimensions the filter is allowed to move in. If a filter is allowed to move only along one axis we have 1D convolution, along two axis we have the 2D convolution, and along all 3 axis we have the 3D convolution. 3D pooling uses similar ideas, instead of taking sum of products at every filter location we take maximum or average.



Implementation of 3D Conv and 3D Pool layers

The implementation will follow the 2D CNN. For the 2D CNN, there is a module 'CNN' with ConvLayer.h and MaxPoolLayer.h files. Similarly a 3D CNN module will be made with Conv3DLayer.h and MaxPool3DLayer.h files. An overview of the 3D Convolution class is as follows :

```
template <typename Architecture_t>

class T3DConvLayer : public VGeneralLayer<Architecture_t>
```

Methods

- `forward(input) : void`
- `backward(gradient_backwards, activation_backwards) : void`

Variables

- `fFilterDepth : size_t`
- `fFilterHeight : size_t`
- `fFilterWidth : size_t`
- `fStrideX : size_t`
- `fStrideY : size_t`
- `fStrideZ : size_t`
- `fPaddingX : size_t`
- `fPaddingY : size_t`
- `fPaddingZ : size_t`

- noOffilters : size_t (and so on…)

An overview of the 3D Pooling class is as follows . We support both average as well as max pool layers :

```
template <typename Architecture_t>

class T3DPoolLayer : public VGeneralLayer<Architecture_t>
```

Methods

- forward(input) : void
- backward(gradient_backwards, activation_backwards) : void

Variables

- fFilterDepth : size_t
- fFilterHeight : size_t
- fFilterWidth : size_t
- fStrideX : size_t
- fStrideY : size_t
- fStrideZ : size_t
- fPaddingX : size_t
- fPaddingY : size_t
- fPaddingZ : size_t
- type : bool (max or avg)
- noOffilters : size_t (and so on…)

Implementation for CPU and GPU platforms

3D CNN functionalities will be implemented for all 3 platform types in TMVA i.e. CPU, GPU and Reference. The methods such as `Backward()` and `Forward()` will be implemented as templates as follows:

- `template<typename Architecture_t>`
  `auto T3DConvLayer<Architecture_t>::Backward(..)`
- `template<typename Architecture_t>`
  `auto T3DConvLayer<Architecture_t>::Forward(..)`

These methods will in turn call `Architecture_t::3DConvLayerBackward()` and `Architecture_t::3DConvLayerForward()`. Hence, within this method the specialised function written for a particular platform will be called. For example, for `Backward()` function following functions will be called for cpu and gpu platforms respectively:

- `TCpuMatrix<AFloat> & TCpu<AFloat>::3DConvLayerBackward()`
- `TCudaMatrix<AFloat> & TCuda<AFloat>::3DConvLayerLayerBackward()`

These functions will in turn be implemented in the file propagation.cxx (that also has 2D CNN operations) in the corresponding module for each of the platform types using the low level functionalities (such as hadamard product, column sum, matrix multiplication etc.) that is optimized for that platform type.

Backpropagation Test: Comparing Numerical and Analytical Gradient

I'll also implement a test for verifying that the backward pass for 3D convolution and 3D pooling layers is working correctly. The test will compare the gradients obtained through backpropagation and gradients obtained numerically, i.e we perturb the input by a small amount, forward pass through the network and see the change in output to compute the gradient.

Convolution/Pooling Code Optimization for CPU (Both for 2D and 3D Convolution Layers)

The current implementation of convolution iterates over each example of the batch to do forward/backward pass. This can probably be made more efficient, by passing the batch data in the form of tensor to BLAS library to do the computation, which is optimized for CPU operations. This can be done for both 3D as well as 2D convolution/pooling layers.

Timeline

My summer vacations start from 15th May 2020. I have no other commitments during the entire GSoC period. So I will be easily able to give 40-50 hours per week. I will also maintain a blog so that my mentors can regulate my work, and also monitor my progress.

Week 1

- Finalising the class design for 3D Conv layer
- Finalising the class design for 3D Pool layer
- Implementing Forward pass for CPU for both layers

Week 2 & 3

- Implementing Forward pass for ref for both layers

Week 4

- Implementing Forward pass for gpu for both layers

Before **Phase 1** Evaluations, forward pass will be implemented for all architectures

Week 5

- Implementing Backward pass for ref for both layers

Week 6, 7 & 8

- Implemented Backward pass for cpu and gpu for both layers

- Numerical Gradient Check

Before **Phase 2** Evaluations, the complete implementation 3d Conv layers would be completed (forward + backward pass)

Week 9

- Test the 3D CNN end to end layer on a real dataset
- Try to optimize CPU code using BLAS library

Week 10

- Wrap up and Buffer Time
- Write Documentation of the work done in GSoC Period,

Experience with TMVA library

I already have a lot of experience with TMVA, as I was a student the previous year and successfully implemented the GRU and LSTM layers (which to be honest, is way more typical and math heavy than convolutions !)

References

- 3D CNN