Development of LSTM and GRU layers in TMVA

About Me

Basic Information

Name	Surya S Dwivedi
University	Indian Institute of Technology, Kharagpur
Major	Computer Science and Engineering
Email	surya2191997@gmail.com
Github Handle	surya2191997
Blog	surya2191997.github.io
Timezone	IST (UTC +5:30)
Contact No	+91 8436930888

Background

I am currently a final year undergraduate student pursuing Computer Science and Engineering at Indian Institute of Technology, Kharagpur. I have significant research experience in Machine Learning, Deep Learning and Computer Vision. During my third year internship at Adobe Research, we developed a system that creates a new banner based on multiple user inspirations using deep CNNs to extract the template. Our work was published at the ACM IUI 2019, Los Angeles and we also filed a US patent for the same. As part of my final year thesis, I am working on reducing execution latency of deep learning algorithms. An idea I am currently exploring is distilling knowledge from state of the art LSTM networks to CNN networks for text classification tasks, thus improving the execution latency (by exploiting parallelism in CNNs). We plan to submit our work in EMNLP 2019.

I have been programming for about 4 years now. I am comfortable with programming in C, C++, Java and Python. I have regularly been programming in these languages as part of hackathons, course projects and internships. I've used C++ mainly for competitive programming contests where in I got comfortable with STL. I had represented our college at the ACM ICPC Chennai Regionals in 2017. I've also used C++ along with OpenCV in some computer vision and machine learning projects such as coin count and 2d image stitching. I've used Java in developing an android app for tracking our institute buses and also in my course project for software lab. Most of my course assignments have been in python. I had also used python for programming during my internship at Adobe Research. I am quite comfortable using git and github for version control. I use Sublime Text 3 as my editor for programming and Mac OS X as my operating system.

I am also a robotics enthusiast and have been a software team member of the Swarm Robotics research group. This group aims to design a group of robots exhibiting swarm intelligence to achieve objectives such as navigation and mapping. I worked mainly on the vision module of the project, specifically the implementation and testing of simultaneous localization and mapping(SLAM) algorithms in C++. I also helped develop a swarm simulator for testing path planning algorithms which was also written in C++. Our group recently participated in the Robotics and Unmanned Systems Exposition organised by the Ministry of Defence, Govt. of India and bagged the 2nd position at the national level among 1088 participating teams.

My educational background is sufficient for this project. I have taken courses on Machine Learning, Deep Learning, Algorithms and Software Engineering as part of my curriculum which equips me well to tackle this project. Apart from this I have also taken various online course such as courses CS231n and CS224n on deep learning by Stanford University. Here is the link to my CV.

Platform details

OS : MacOS X Yosemite (10.10.5)

Editor : Sublime Text 3

Project Overview

This project is about development of Long Short Term Memory(LSTM) and Gated Recurrent Unit(GRU) layers in TMVA, both of which belong to a general class of neural networks called the Recurrent Neural Networks(RNN). These layers have many important applications in the realm of data analysis for particle physics experiments. As an example, LSTMs can be used for track reconstruction of charged particles in the Large Hadron Collider(LHC). They can also be used for analyzing the voltage time series from the electronic monitoring system present in superconducting LHC magnets.

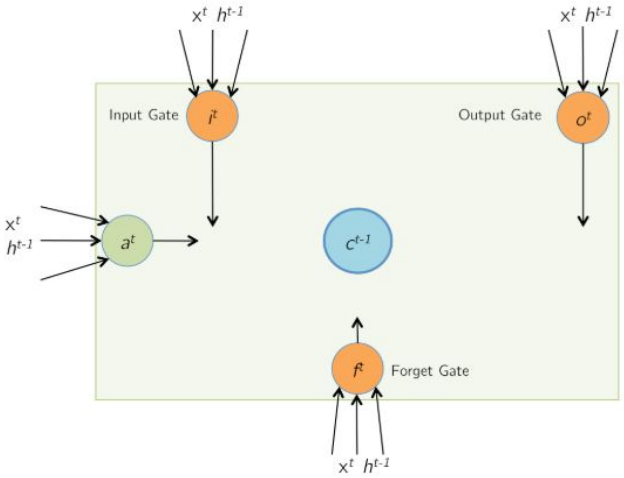
Deliverables

- Implementation of Forward and Backward passes for LSTM and GRU layer types for both CPU and GPU architectures
- Testing of Forward and Backward passes for LSTM and GRU layer types for both CPU and GPU architectures using numerical gradient check
- Implementation of ONNX operators for RNN, LSTM and GRU layers (in collaboration with the GSoC project on implementation of ONNX operators)
- Training and Testing the layers on Electromagnetic Calorimeter(ECAL) dataset.

Details of Project

LSTM and GRU layer overview

LSTMs are recurrent neural networks that are capable of learning long-term dependencies. They do not suffer from the vanishing gradient problem that is present in case of vanilla RNNs. The detailed computations involved in the forward pass of an LSTM cell are explained below:



$$a^t = tanh(W_c x^t + U_c h^{t-1}) = tanh(\hat{a}^t)$$

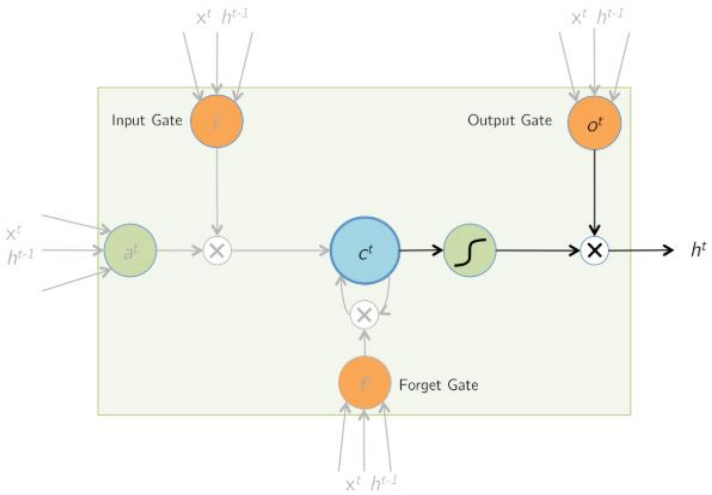
$$i^t = \sigma(W_i x^t + U_i h^{t-1}) = \sigma(\hat{i}^t)$$

$$f^t = \sigma(W_f x^t + U_f h^{t-1}) = \sigma(\hat{f}^t)$$

$$o^t = \sigma(W_o x^t + U_o h^{t-1}) = \sigma(\hat{o}^t)$$

At time t, the input to the LSTM cell is the input vector at that instant (x^t) and the output from the previous step (h^{t-1}). The hadamard product of the outputs of the input gate and activation gate (i.e. $i^t \odot a^t$) helps to add new information that is relevant to the current time step. The hadamard product of the outputs of the forget gate and the previous cell state (i.e. $f^t \odot c^{t-1}$) helps to forget the previous information and remember only the relevant information. The new cell state is an element wise sum of these two:

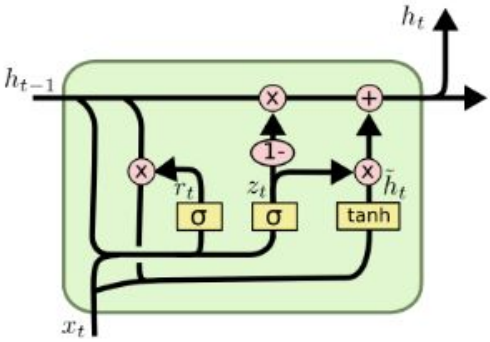
$$c^t = i^t \odot a^t + f^t \odot c^{t-1}$$



Finally, the output of the cell is computed as follows:

$$h^t = o^t \odot tanh(c^t)$$

Gated Recurrent Units(GRUs) are variants of LSTMs that are internally simpler and faster than vanilla LSTMs . In contrast to LSTMs which have 3 gates (i.e. Input, output and forget), GRUs have only 2 gates (i.e. update and reset). The detailed computations involved in the forward pass for a GRU cell are explained below:



$$z^t = \sigma(W_z x^t + U_z h^{t-1})$$

$$r^t = \sigma(W_r x^t + U_r h^{t-1})$$

$$\hat{h}^t = tanh(W_h(r^t \odot h^{t-1}) + U_h x^t)$$

$$h^t = (1 - z^t) \odot h^{t-1} + z^t \odot \hat{h}^t$$

The GRU combines the forget and input gates of LSTM into one single update gate. It also merges the cell state and the hidden state.

Implementation Overview

The RNN layer has been defined as a class `TBasicRNNLayer<Architecture_t>` in TMVA. Following similar conventions, I shall be designing classes for LSTM and GRU layers. Following are some of the tentative methods and variables for each of the classes:

`TBasicLSTMLayer<Architecture_t>:`

Methods

- `GetTimeSteps() : size_t`
- `GetStateSize() : size_t`
- `GetInputSize() : size_t`
- `InitState()    : void`
- `CellForward()  : void`
- `Forward()      : void`
- `CellBckward()  : void`
- `Backward()     : void   (and so on..)`

Variables

- `fTimeSteps    : size_t`
- `fStateSize    : size_t`
- `fInputGateWeights  : Matrix_t`
- `fForgetGateWeights : Matrix   (and so on..)`

`TBasicGRULayer<Architecture_t>:`

Methods

- `GetTimeSteps() : size_t`
- `GetStateSize() : size_t`
- `GetInputSize() : size_t`
- `InitState()    : void`
- `CellForward()  : void`
- `Forward()      : void`
- `CellBackward()  : void`
- `Backward()     : void   (and so on..)`

Variables

- `fTimeSteps    : size_t`
- `fStateSize    : size_t`
- `fUpdateGateWeights : Matrix_t`
- `fResetGateWeights  : Matrix   (and so on..)`

Testing Forward and Backward passes using Numerical Gradient Check

For testing the forward and backward passes for both the layers, I shall be implementing numerical gradient check. I will be verifying the gradients computed by the backpropagation algorithm with the gradients computed from numerical differentiation. This test covers both forward and backward propagation through the network as the backpropagation algorithm uses values computed during forward pass also. An overview of how this can be done is as follows :

$$\frac{\partial E}{\partial w} = \frac{E_{final} - E_{initial}}{\Delta w}$$

We increment a particular value w in the weight matrix by a small amount Δw. We then perform a forward pass with the modified weight matrix. Let the final loss after this forward pass is E_{final} and the initial loss(with the original weight matrix) is $E_{initial}$. Then the partial derivative of the loss function E wrt w is given by the above equation. We then verify this value with the corresponding term in the gradient matrix computed after the backward pass. We perform this process for every value in the weight matrix. This test can be implemented as a separate folder for each layer type (LSTM and GRU) and for each platform (cpu and gpu), in the test module in TMVA repository just like the RNN layer.

Implementation and Testing for CPU and GPU platforms

LSTM and GRU layer functionalities will be implemented for all 3 platform types in TMVA i.e. CPU, GPU and Reference. The implementation of LSTM layer is discussed here, GRU layer will be implemented similarly. The methods such as `CellBackward()` and `CellForward()` will be implemented as templates as follows:

- `template<typename Architecture_t>`
  `auto TBasicLSTMLayer<Architecture_t>::CellBackward(..)`
- `template<typename Architecture_t>`
  `auto TBasicLSTMLayer<Architecture_t>::CellForward(..)`

These methods will return `Architecture_t::LSTMLayerBackward()` and `Architecture_t::LSTMLayerForward()`. Hence, within this method the specialised function written for a particular platform will be called. For example, for `CellBackward()` function following functions will be called for cpu and gpu platforms respectively:

- `TCpuMatrix<AFloat> & TCpu<AFloat>::LSTMLayerBackward()`
- `TCudaMatrix<AFloat> & TCuda<AFloat>::LSTMLayerBackward()`

These functions will in turn be implemented in a file LSTMpropogation.cxx 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. I shall also be implementing tests for all the 3 architecture types for both the layers(LSTM and GRU).

Implementation of ONNX operators for RNN, LSTM and GRU layers

ONNX(Open Neural Network Exchange Format) is an open format that is used for representing Deep Learning models. It can describe models generated from any mainstream Deep Learning Frameworks such as TensorFlow, PyTorch etc. The models in ONNX are defined using operators, where in each operator represents a fundamental operation on the tensor in a computational graph. I'll be implementing the RNN, LSTM and GRU ONNX operators in TMVA in collaboration with the other GSoC project on implementation of ONNX operators. The `RTensor` class will be used as standard interface for the input/output tensors of the operators to be implemented. An overview of the equations of the operators to be implemented is as under:

- RNN : (Default: f=Tanh)

```
-Ht = f(Xt*(Wi^T) + Ht-1*(Ri^T) + Wbi + Rbi)
```

- LSTM: (Default: f=Sigmoid, g=Tanh, h=Tanh)

```
- it = f(Xt*(Wi^T) + Ht-1*(Ri^T) + Pi (.) Ct-1 + Wbi + Rbi)
- ft = f(Xt*(Wf^T) + Ht-1*(Rf^T) + Pf (.) Ct-1 + Wbf + Rbf)
- ct = g(Xt*(Wc^T) + Ht-1*(Rc^T) + Wbc + Rbc)
- Ct = ft (.) Ct-1 + it (.) ct
- ot = f(Xt*(Wo^T) + Ht-1*(Ro^T) + Po (.) Ct + Wbo + Rbo)
- Ht = ot (.) h(Ct)
```

- GRU: (Default: f=Sigmoid, g=Tanh)

```
- zt = f(Xt*(Wz^T) + Ht-1*(Rz^T) + Wbz + Rbz)

- rt = f(Xt*(Wr^T) + Ht-1*(Rr^T) + Wbr + Rbr)

- ht = g(Xt*(Wh^T) + (rt (.) Ht-1)*(Rh^T) + Rbh + Wbh) # default, when
                                linear_before_reset = 0

- ht = g(Xt*(Wh^T) + (rt (.) (Ht-1*(Rh^T) + Rbh)) + Wbh) # when linear_before_reset
                                                != 0
                        - Ht = (1 - zt) (.) ht + zt (.) Ht-1
```

For more details on notations, attributes, output types etc. for these operators the link on ONNX operator schemas can be seen.

Timeline

My summer vacations start from 1st May 2019. I have no other commitments during the entire GSoC period as my college ends after this semester, and I am not joining any job (as I'll be leaving India for my Masters in September). 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.

Pre-GSoC

- I have already gone through a major portion of the TMVA code base for DNN and RNN modules. I would like to inspect remaining part of it. I would also like to explore the BLAS and CUDA libraries a bit more.
- Gain some experience with the `RTensor` class as I shall be using it for implementing the ONNX operators
- Contacting the student selected for the GSoC project on implementation of ONNX operators and planning with him the timeline for the implementation of LSTM, GRU and RNN operators with the help of mentors
- Procuring the Electromagnetic Calorimeter(ECAL) dataset that will be used for training and testing the implemented layers

Week 1

- Finalising the class design for LSTM layer
- Implementing LSTM Forward pass for CPU and Reference

Week 2 & 3

- Implementing LSTM Backward pass for CPU and Reference
- Finalising the class design for GRU layer
- Implementing GRU Forward pass for CPU and Reference
- Implementing GRU Backward pass for CPU and Reference

Week 4

- Implementing Numerical Gradient check for both layers(LSTM and GRU) for CPU and Reference
- Testing the forward and backward passes for both layers(LSTM and GRU) for CPU and Reference

Before **Phase 1** Evaluations, The LSTM and GRU layers would have been implemented and tested for CPU

Week 5

- Implementing LSTM Forward and Backward pass for GPU
- Implementing GRU Forward and Backward pass for GPU

Week 6, 7 & 8

- Implementing Numerical Gradient check for both layers(LSTM and GRU) for GPU
- Testing the forward and backward passes for both layers(LSTM and GRU) for GPU
- Training and Testing the layers implemented on an ML dataset (ECAL)

Before **Phase 2** Evaluations, the complete implementation and testing of LSTM and GRU layers would be completed along with training and testing on an ML dataset.

Week 9

- Start collaborating with the other student on implementation of ONNX operators for RNN, LSTM and GRU
- Familiarise with the basics of ONNX and the RTensor class in TMVA

Week 10

- Implementing the “RNN” ONNX operator

Week 11 & 12

- Implementing the “LSTM” and “GRU” ONNX operator

Week 13

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

Post GSoC

I would like to continue contributing to TMVA even after my GSoC period.My immediate goals after GSoC would be the following :

- Implement some optimized variants of the LSTM and GRU layers (like peephole LSTMs)
- Implement some more complex ONNX operators like convolution

Experience with TMVA library

For completing the selection task, I had read the TMVA Quick Start section of TMVA user guide in detail. This helped me in understanding concepts like TMVA Factory object, signal and background trees etc. that are used for training, testing and evaluation of a model in TMVA. I also built the ROOT framework from source in my PC and ran tutorials for classification and regression in TMVA. I thereafter read the code for classes in the DNN package and also the MethodDL class. With this base, I started solving the problems for the selections task. For implementing RNN for classification, I read the code for RNN in TMVA and used my knowledge of basics that I acquired from reading the user guide. These experiences helped me in understanding the TMVA codebase, coding conventions followed and the overall code structure.

References

- LSTM Forward and Backward Pass
- Understanding LSTM networks
- ONNX operator schemas
- ONNX
- GSOC Blog (Particularly for numerical gradient check)