Informatics Project Proposal: Machine Learning Software Frameworks

Abhijit Singh
s1788323
s1788323@ed.ac.uk

Abstract

Deep learning is being used to make breakthroughs in many areas of research. There are many new deep learning frameworks available, but we do not yet know their comparative strengths and weaknesses, especially when using complicated models. This is a proposal for a project that aims to fill this gap in the literature. We state the reasons why these frameworks are important, and give a brief review of some of the popular frameworks. We outline why the model we aim to implement is ideal for our project, and give a recap of some concepts that will be used. The general plan and objectives of this project conclude the proposal.

1 Introduction

In the last two decades there has been an exponential increase in the amount of raw information available. Until a little while back, we did not know how to analyze and draw conclusions from this vast amount of data in a reasonable amount of time. For example, in the field of space exploration, there are many observatories that collect data from space in the form of raw images or measurements made by sophisticated sensors. Then scientists spend an enormous amount of time to manually analyze this data and make meaningful conclusions. Similarly in medical diagnosis, doctors or lab technicians spend a lot of time going over the scans of individual patients. This extra time and effort that is spent on analysis can be saved if the process is automated. This is where machine learning becomes extremely useful.

In situations where we have a lot of data to process, neural networks and specifically deep learning models are exceedingly useful tools. The main advantage that deep learning models have over other models is that they can create more representations of the input data over their successive layers, which help them do a better job at tasks like classification, amongst others. We have seen machine learning technology in various fields now, such as face recognition softwares in cameras, web searching algorithms, automated chat-bots, recommendation systems on e-commerce and video streaming sites, and even medical diagnosis LeCun et al. (2015). The applications of machine learning, and particularly deep learning, are only going to increase in the future.

Back-propagation Rumelhart et al. (1986) is a key component of a deep learning model. It revolutionized the field of machine learning, and it is this step that makes neural networks so powerful. However, computing the gradients that need to be back-propagated through a network is quite tricky, especially for complicated deep learning models, and immensely time-consuming to do analytically. Hence it was necessary to have tools which would make these tasks easier, and this is why machine learning software frameworks were developed. Each framework has its own method of automatically computing any gradients that might be required. Nowadays a researcher does not need to spend any time on checking if the intricate details of their model are faultless, they can use one of the numerous frameworks available and quickly build many different models and test them.

This naturally makes us consider which framework should be used, out of the myriad options available today. At the moment, there is no definitive answer to this question. Since this is a new field with a lot of active development ongoing, there have been many significant changes in the last few years. This project will aim to compare some of the popular and most widely used frameworks on a neural language modeling task, and attempt to gain a better understanding of the pros and cons of these frameworks. This work should be beneficial to all researchers who want to know which framework is ideal for their specific task, and what hurdles they might face during the developmental phase.

The next section gives a brief overview of the current literature, and also introduces the neural language modeling task that shall be used for this project. Section 3 outlines the general plan of the project, and Section 4 succinctly states the objectives of the project.

2 A Brief Review

One of the earliest open source and comprehensive machine learning frameworks was Torch Collobert et al. (2002). It was first released in 2002, and it provides data structures for many useful components such as multi-dimensional tensors. It is a script language based on the Lua programming language, with an underlying implementation in C. When it was initially released it was used by a significant number of researchers, but many have now switched to the newer frameworks.

Theano Bergstra et al. (2010) was developed at the University of Montreal around 2008. The coding is done in Python, and it combines the convenience of using the NumPy syntax with the speed of optimized native machine language. It provided massive speed ups on processing times for large neural networks, compared to the other frameworks available at its time of release. Theano computes symbolic differentiation of complex expressions automatically and minimizes the memory usage during processing, amongst a host of other features. This is done by storing the mathematical expressions defined by the user as a graph of variables and operations, that is pruned and optimized at compilation time. Active development of Theano was stopped in September 2017, due to competing frameworks launched by many big technology companies like Google, Facebook and Microsoft.

Convolutional Architecture for Fast Feature Embedding (CAFFE) Jia et al. (2014) was developed at UC Berkeley in 2014. As the name suggests, it primarily focuses on Convolutional Neural Networks (CNN). In recent releases the developers added some support for Recurrent Neural Networks (RNN) as well, but most researchers still avoid using CAFFE when dealing with text or time series data. It is written in C++ with a Python interface.

Like Theano, Tensorflow Abadi et al. (2016) also uses a static computation graph. Its flexible architecture allows for easy deployment of computation across various platforms like desktops, GPUs and mobile devices. It was developed by the Google Brain team for internal use, but was finally released as an open source library in November 2015. It is built keeping extensibility in mind, and the framework's document claims that it scales well to multiple GPUs. MXNet Chen et al. (2015) is another deep learning framework that is backed by a big commercial company, in this case, Amazon. Its features are quite similar to Tenserflow, but the low level implementation is obviously different. It handles the embeddings of both symbolic expression and tensor operation in a unified fashion, and supports many different programming languages for its frontend, such as Python, R, Scala and Perl.

PyTorch was released in October 2016, by Facebook's Artificial Intelligence (AI) research group. It can be viewed as a Python frontend for the Torch engine. Like the other frameworks, it allows users to define mathematical functions and computes gradients automatically. However, PyTorch uses a dynamic computation graph, which makes it an easy to use and efficient framework for handling text or time series data. Due to this, many researchers have started using PyTorch even though it is a new entrant to the scene.

Keras Chollet et al. (2015) is an open source library written in Python. It was designed by an engineer at Google to enable fast experimentation with deep neural networks, and it focuses on being user-friendly, modular and extensible. It is basically a high level interface which abstracts the backend, making it quite easy for beginners. It supports various backends like Tensorflow, Theano, MXNet and Microsoft Cognitive Toolkit (CNTK) Seide and Agarwal (2016). Its ease of use and

support for many popular backends have resulted in Keras having the fastest growing user base, and being second only to Tensorflow in overall popularity, according to many surveys and reports.

With so many competitive frameworks being available, one would expect that many people have carried out comparative studies. However this is not the case. There is a dearth of such studies in the current literature, and this is one of the key reasons for undertaking this project. Of the few studies that have been done, Bahrampour et al. compared CAFFE, Neon, TensorFlow, Theano, and Torch on metrics such as extensibility, hardware utilization and speed. They used stacked autoencoders and CNNs on the MNIST dataset LeCun (1998) and the ImageNet dataset Deng et al. (2009), and a Long Short-Term Memory (LSTM) network on the IMDB review dataset Maas et al. (2011). Similarly, Shi et al. compared CAFFE, CNTK, MXNet, Tensorflow and Torch in terms of time taken for processing with different mini batch sizes, on the same datasets as used by Bahrampour et al.. However, while Bahrampour et al. only tested on one type of GPU, Shi et al. tested their architectures on three different architectures of GPUs.

There are a few key points to note after going through this brief review. Firstly, there is a vacuum of studies which compare the latest frameworks with each other. Secondly, the existing studies implement only a few standard architectures on standard datasets, for each framework. There has been no attempt to compare complicated architectures on a different dataset. Lastly, there has not been any mention of challenges faced during model building in each of the frameworks. This is a significant concern for most researchers, as a lot of time that is spent on coming up with hacks can be saved, if building models is an easy and smooth process.

Keeping the points mentioned above in mind, we choose to implement the work of Yang et al. for the proposed project. In this paper the authors argue and prove that language modeling can be equated to a matrix factorization problem, and that the expressiveness of Softmax based models is limited by a 'Softmax bottleneck'. This is important because a majority of neural language models use a Softmax layer to make the multi-class classification, and consequently all of them suffer due to this bottleneck. They propose a solution to this problem, which is an alternate to the conventional Softmax model. In the paper they term this model as 'Mixture of Softmaxes'. Yang et al. show that this new model works better than the conventional Softmax, by testing it on different datasets like the Penn Treebank (PTB) dataset Marcus et al. (1993) and the WikiText-2 dataset Merity et al. (2016).

We feel this paper is ideal to make a comparison between different frameworks because it involves building a new, and more complex model. This is important because now we can also subjectively document what roadblocks were faced during the model building phase for each framework. This will give other researchers more insight into the subtle nuances of each framework, and give them a rough idea of whether they would find it comfortable to work with a particular framework. At the same time, since the datasets being used are very big, we will have the opportunity to observe a larger difference between the performance of each framework on GPUs.

Before proceeding to explain how we will approach this project, let us briefly review what language models are. Language models are basically a probability distribution over sequences of words. We use a subset of words that have been used until then to predict what the next word is going to be. The number of previous words we use to predict the next word is called the context length. If we use just the previous word (that is, a context length of 1), the model is called a uni-gram model. Similarly, using two previous words is a bi-gram model and using n previous words makes an n-gram model. Increasing the context length also increases the complexity of the models, and this is why neural language models are important. They use continuous representations or embeddings of words to make their predictions (using neural networks, hence the name 'neural' language models). This is why they are also sometimes referred to as continuous space language models. There is extensive literature on the different kinds of language models, and the pros and cons of each.

3 Approach

Though the project is an individual piece of work, there are 4 people working on the same topic. Thus at the beginning we decide to each pick one framework, and implement the work of Yang et al. on our chosen framework. I will be using Keras with a Tensorflow backend. After we have all completed our implementations, we will compare our results so that we can compare across all the frameworks and analyze their differences.

3.1 Methods

In this project we shall primarily be using different architectures of RNNs and LSTMs.

RNNs were developed to address the shortcomings of the feed-forward neural language model approach in Bengio et al. (2003); in particular the fixed and small window size for the input. A conspicuous limitation of vanilla neural networks (and CNNs as well) is that they are too constrained. They accept a fixed-sized vector as input (perhaps images or videos) and produce a fixed-sized vector as output (usually the probabilities of different classes, when the task is multi-class classification). Moreover, this mapping is performed using a fixed number of computational steps (for example, the number of layers in the model). The main reason why RNNs are effective is that they allow us to operate over sequences of vectors: sequences in the input, the output, or in the most general case, both in put and output. Fundamentally, RNNs are neurals network that take both an extrinsic input and it's own previous state as a combined input. This way the network is able to consider past states in addition to the current state; it has a memory (see figure 1). As a result, RNNs do not use a limited size context; information can cycle inside these networks for an arbitrarily long time, although long-term dependencies are not learned easily Mikolov et al. (2010).

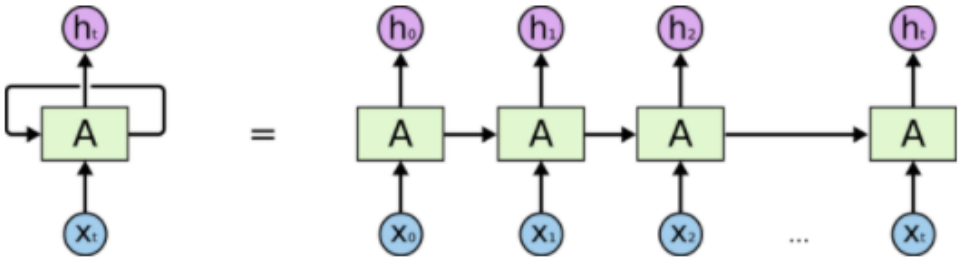


Figure 1: Diagram of an RNN. Right hand side is 'unrolled', with sequential timesteps from left to right. x are inputs, A is a hidden layer and h are outputs. Figure from Olah (2015).

For a simple RNN (an Elman network) there is an input layer x, a hidden layer s and an output layer y; this is the same as a typical single layer neural network. Critically, the hidden, or context layer, s is preserved for one timestep, such that at time t, s_{t-1} is available for computation. Input, hidden and output layers are computed by

$$x(t) = w(t) + s(t-1) \tag{1}$$

$$s_j(t) = f\left(\sum_i x_i(t) u_{ji}\right) \tag{2}$$

$$y_k(t) = g\left(\sum_j s_j(t) v_{kj}\right) \tag{3}$$

where $f(z)$ is the sigmoid function and $g(z)$ is the softmax function.

LSTMs are a special kind of RNN, which were designed to solve the long term dependency problem. As they are capable of learning long term dependencies, they work very well on a large variety of problems, and hence are widely used Olah (2015). All RNNs are in the form of a chain of repeating

modules of neural network. In standard RNNs, this repeating module has a simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four layers, interacting in a special way. RNNs are architecturally forced to consider the input of the previous state. LSTMs on the other hand, make this connection a learned gate function, meaning that the network learns how to weight the previous states given the current context. LSTMs also make the input and output dependent on learned gate functions, allowing the network to optionally and flexibly consider input, output and context (previous state) dependent on the current input (figure 2 shows this architecture). This is computed by

$$f_t = \sigma(W_f[s_{t-1}, x_t] + b_f) \tag{4}$$

$$i_t = \sigma(W_i[s_{t-1}, x_t] + b_i) \tag{5}$$

$$\widetilde{C_t} = tanh(W_C[s_{t-1}, x_t] + b_C) \tag{6}$$

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{7}$$

$$o_t = \sigma(W_o[s_{t-1}, x_t] + b_o) \tag{8}$$

$$s_t = o_t * tanh(C_t) \tag{9}$$

where f_t is the forget gate, i_t is the input gate, o_t is the output gate, $\widetilde{C_t}$ are candidate values to be propagated dependent on the input gate and C_t is the updated cell values.

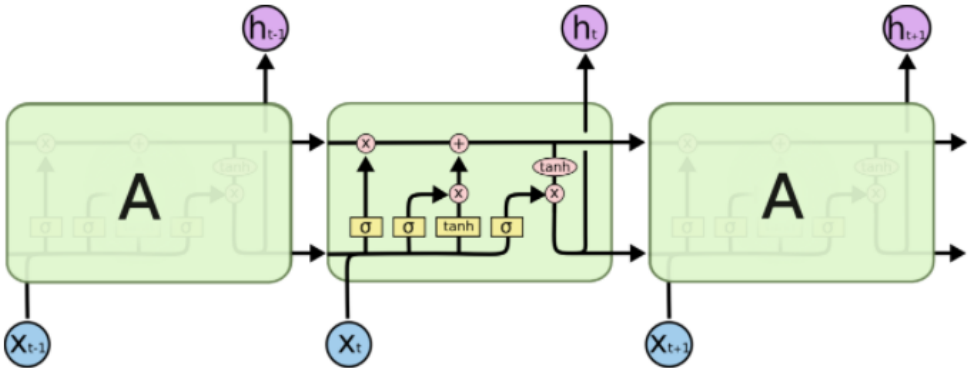


Figure 2: LSTM architecture diagram showing the gating functions (the three sigmoid functions σ). Figure from Olah (2015).

In this project, we shall use a simple LSTM layer as one of the baselines. We shall also compare the model proposed by Yang et al. against at least two other architectures. These will be finalized after a comprehensive literature survey. During the implementation, we shall be documenting how many functions were available in the existing library of the framework, and how much time we had to spend on building the functions which were unavailable, with a particular emphasis on the debugging experience. Since we want to replicate the model of Yang et al., more detail about the exact experiments we will run can be found in their paper. All the experiments will be run on GPUs, and the metrics that we are interested in are detailed in the next section.

3.2 Evaluation Metrics

We want to compare the deep learning frameworks on the following metrics:

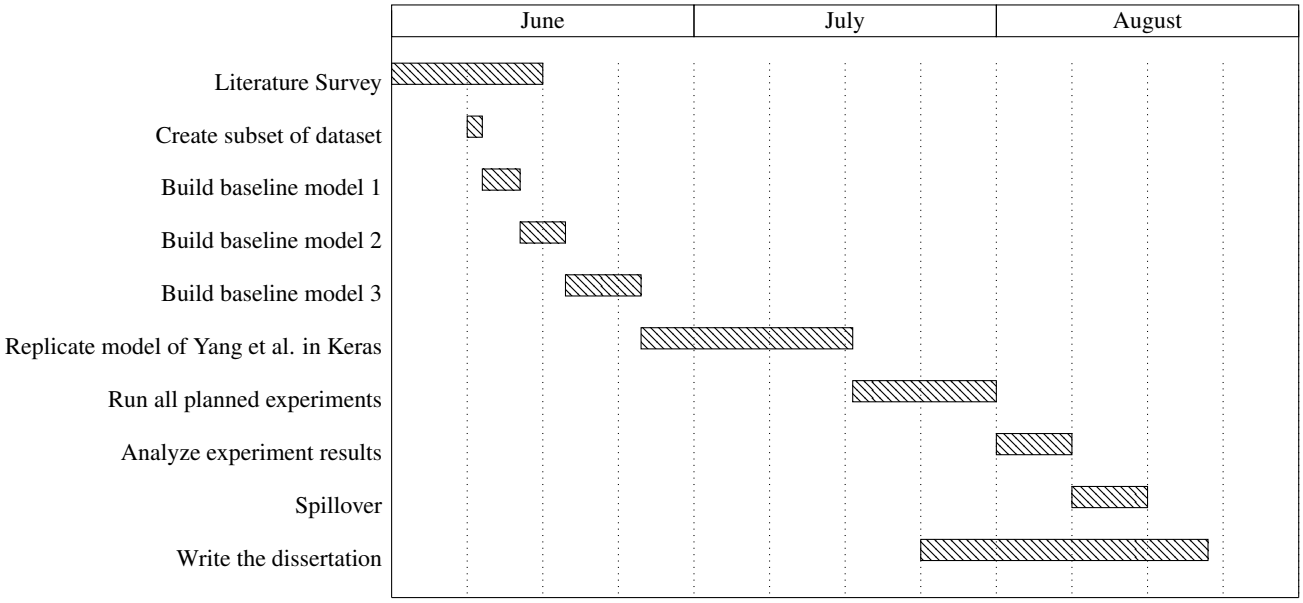
1. Time taken to load the data into memory and completing the pre-processing. Here we will also see whether always streaming the data is better than caching it to the disk.
2. Time taken for forward and backward pass of data. This will include:
   - Total training time per epoch - this is important because it is the most computationally expensive part of building a machine learning model. Apart from compute time, we can also calculate what the effective monetary cost is.
   - Total time taken to make a prediction - this is important because for many applications the prediction time needs to be extremely quick (for example, choice of online ad to be displayed).
3. How well does the framework scale with the use of multiple GPUs. This metric is especially important for training complex models with vast amounts of data.
4. How well does the framework utilize available hardware. This will mean seeing how much memory it takes on the GPUs.
5. Ease of use. This is subjective, and will include:
   - How many functions required in the models we build are available as built in functions, and how many do we have to code on our own.
   - How much time did we spend coding custom functions, and how helpful were the error messages (and material available online) during debugging.
6. Are there any significant differences in results (perplexity of the neural language model) obtained on the different frameworks.

3.3 Plan of Work

The basic plan can be summarized with the following steps:

1. Install Tensorflow and Keras, and all dependencies required.
2. Download the dataset and place it in the appropriate folder.
3. Read the paper many times to become comfortable with it.
4. Do a literature survey and confirm additional baselines.
5. Create a subset of the dataset, so that it can be used for checking if our implementation is correct. This will ensure that we don't have to use a GPU every time we test a small chunk of code during the model building phase.
6. Build and test the baseline models.
7. Replicate the work of Yang et al. in Keras (They had used PyTorch).
8. Document all hurdles faced during model building.
9. Once the model building is completed, start the planned experiments on the GPUs.
10. Repeat each experiment a set number of times (perhaps 10), and tabulate the mean and standard deviation of the results.
11. Compare results with other people, and start analyzing them.
12. If time permits, try to improve the model's performance.
13. Write the dissertation

The Gantt chart on the next page provides a more detailed timeline of the different steps.



Ample time has been allocated for the tasks that are expected to take the longest time, which include: building the main model, running the experiments, and writing the dissertation. Some tasks, which are independent of each other, are overlapping in their allocated period. For example, after the initial part of the literature survey is completed, we begin work on our baseline models. Similarly, we will begin writing those parts of the dissertation which do not depend on the experiment results well before they are completed. This will ensure that the writing of the dissertation is not hurried towards the end.

4 Objectives

The proposed project aims to compare different deep learning frameworks, on the metrics detailed in Section 3.2. In short, they are:

- Speed of processing on single GPU
- Scalability to multiple GPUs
- Hardware utilization
- Ease of use
- Perplexity of the neural language model

We aim to present a comprehensive list of advantages and disadvantages of the framework used by us, based on our obtained results and experience of working with it. Ideally, our project will help other researchers in deciding which framework would be best suited to their requirements. It may also lead to suggestions on improving the framework.

References

LeCun, Y.; Bengio, Y.; Hinton, G. *nature* **2015**, *521*, 436.

Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. *nature* **1986**, *323*, 533.

Collobert, R.; Bengio, S.; Mariéthoz, J. *Torch: a modular machine learning software library*; 2002.

Bergstra, J.; Breuleux, O.; Bastien, F.; Lamblin, P.; Pascanu, R.; Desjardins, G.; Turian, J.; Warde-Farley, D.; Bengio, Y. Theano: A CPU and GPU math compiler in Python. Proc. 9th Python in Science Conf. 2010.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional architecture for fast feature embedding. Proceedings of the 22nd ACM international conference on Multimedia. 2014; pp 675–678.

others,, et al. TensorFlow: A System for Large-Scale Machine Learning. OSDI. 2016; pp 265–283.

Chen, T.; Li, M.; Li, Y.; Lin, M.; Wang, N.; Wang, M.; Xiao, T.; Xu, B.; Zhang, C.; Zhang, Z. *arXiv preprint arXiv:1512.01274* **2015**,

others,, et al. Keras. 2015.

Seide, F.; Agarwal, A. Cntk: Microsoft's open-source deep-learning toolkit. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2016; pp 2135–2135.

Bahrampour, S.; Ramakrishnan, N.; Schott, L.; Shah, M. *arXiv preprint arXiv:1511.06435* **2015**,

LeCun, Y. *http://yann. lecun. com/exdb/mnist/* **1998**,

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. 2009; pp 248–255.

Maas, A. L.; Daly, R. E.; Pham, P. T.; Huang, D.; Ng, A. Y.; Potts, C. Learning word vectors for sentiment analysis. Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1. 2011; pp 142–150.

Shi, S.; Wang, Q.; Xu, P.; Chu, X. Benchmarking state-of-the-art deep learning software tools. Cloud Computing and Big Data (CCBD), 2016 7th International Conference on. 2016; pp 99–104.

Yang, Z.; Dai, Z.; Salakhutdinov, R.; Cohen, W. W. *arXiv preprint arXiv:1711.03953* **2017**,

Marcus, M. P.; Marcinkiewicz, M. A.; Santorini, B. *Computational linguistics* **1993**, *19*, 313–330.

Merity, S.; Xiong, C.; Bradbury, J.; Socher, R. *arXiv preprint arXiv:1609.07843* **2016**,

Bengio, Y.; Ducharme, R.; Vincent, P.; of machine learning, J. C. *Journal of machine learning ...* **2003**,

Mikolov, T.; Karafiát, M.; Burget, L.; of ..., Č. J. .... *Annual Conference of ...* **2010**,

Olah, C. Understanding LSTMs. 2015; `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.