Chris Culver

Jiafang Liu

Torch and TensorFlow on GPU’s - Final Report

**Introduction**

Machine learning has been making significant advances in the recent years. AI programs, such as DeepLearning have beaten human champions at their own games such as Go. On the hardware side, GPU acceleration is proving to be a very powerful tool. Although machine learning has been around for decades, it has primarily used CPU power. With GPU advances it is natural for machine learning programs to take advantage of GPU speed. This project will compare popular machine learning programs which use different platforms.

This project’s goal is to compare the most popular machine learning programs that use GPU acceleration such as TensorFlow and Torch. The programs will be performance tested to see which is better optimized for GPUs. Reliability, accuracy and usability will also be compared between each software package. The conclusion of this project will result in a comparison to assist people deciding between TensorFlow and Torch.

This report is an update on the progress of the project at this point in the semester. We will start by introducing machine learning and how it works. We then discuss the results we already have, which are applying machine learning concepts to some fairly image data sets. The results from these runs will be included. After this we explain what we are currently working on, which is running the programs on GPU’s. We lastly discuss what is left to be done in the project, and what results can be expected in our final report.

**Machine Learning Basics**

The purpose of machine learning is to literally teach the computer how to detect things in different categories. The first step of this process is called the training step. This requires a set of data whose training data and training label are known. We first want to show the computer how to correctly classify the data in the dataset. Then it will attempt to classify other pieces of data without a priori knowing the answer. To classify the data the computer can use and user defined model. Depending on the data, different models will have different success rates.

The two main model categories are linear and nonlinear models. Two linear models one could use are Logistic Regression(LR) and Linear Discriminant Analysis(LDA). Three nonlinear models that could be used are K-Nearest Neighbor(KNN), Classification and Regression Tree(CART) and Gaussian Naive Bayes(GNB). Each of these models can be trained on the data, then can be tested for accuracy. This would then inform the user how well the models work to classify unknown data.

For the facial recognition portion the model used was a Convolutional Neural Network(CNN). The basic idea behind a CNN is to take the data and pass it through several layers of differentiation functions. There are then three primary types of layers used, Convolution Layers, Pooling Layers, and Fully Connected Layers. A convolutional layer measures the neurons on local regions of the data, and takes dot products between nearby ones. A pooling layer is used to reduce the data to a smaller volume. The last layer, the fully connected layer categorizes the input into the given categories for the task. The specific details of the model we used can be found in Ref 1.

**Setup and Applications Chosen**

MNIST was first implemented on our personal machines successfully in Torch and TensorFlow. After this we were also able to run the MNIST data set on colonial one both with GPU support and without GPU support. Having accomplished this the next task was to find a more complicated machine learning application. This created much difficulty and were the majority of the project time was spent. One problem we faced is that after we finished the mnist data comparison on colonialone, we found it impossible for us the implement Age-Gender-Deep-Learning application on colonialone. Since we have no authority of root , we can not install or update any software what we need. Finally, we have to move the staff and decided use AWS as the common machine to test the our project codes. Moreover, The main issue was finding an application that could be run both in Torch and Tensorflow.

For Torch there is a largely developed application for facial recognition called OpenFace. This was attempted to install on Colonial One, but due to the large amounts of dependencies and lack of TensorFlow counterpart was abandoned. The largest difficulty was correctly installing OpenCv, which had its own dependencies. Next on the TensorFlow side a Facial Keypoint detector was implemented using the Kaggle dataset. A Torch implementation was found, and after again dealing with dependency issues was compiled. This was not the final face recognition due to runtime errors. We believe it is due to a mismatch in Torch versions, but there was no documentation for what version of Torch was necessary.

The facial recognition implemented was a two fold classification program. The categories it can detect on the face are the gender of the subject as well as its age. The age is split up into the following 10 categories, (0-2,4-6,8-13,15-20,25-32,38-43,48-53,60+). The dataset used to test this model was the AdienceFaces data set. This data set contains 19487 images which were uploaded to a social media website. This is a challenging data set because the quality and lighting of the images varies greatly.

**Results**

The first program to be run on GPU’s was the MNIST dataset. In torch, three different models were implemented to see the effect on performance and accuracy. A linear, modified linear, and convolutional neural model were implemented. They were first run on a portion of the dataset where training was performed on 2000 images and testing on 1000 images. After this was completed the code was trained on the full MNIST data set of 60000 images and tested on 10000 images. Ten separate epochs of training and testing were performed. The average of these ten runs is reported in the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Linear(Partial) | MLP(Partial) | CNN(Partial) | Linear(Full) | MLP(Full) | CNN(FULL) |
| Time(s) | 2.76 | 7.19 | 60.29 | 72.85 | 194.000 | 1524.4 |
| Accuracy(%) | 82.78 | 85.85 | 94.30 | 87.18 | 98.24 | 99.45 |

From these results some of the common machine learning concerns are shown. First of all is the time taken as it depends on the model. The more complicated the model is, the more time the model takes. There was a significant increase in time for the CNN over the linear models. This extra cost in time though is balanced out by an increase in accuracy. Each model, although more time consuming, is more accurate than its predecessor. The scalability of the code can also be seen. The full MNIST data set is 30x the size of the partial data set. While the full linear model actually took longer than 30 times as long as the partial run, the MLP and CNN had a slight speedup. Lastly it is shown that the big data aspect of machine learning helps the models be even more accurate. With more data to train on, each model was able to increase its correctness. The largest increase came to the MLP model with a 13% increase in accuracy.

The next step was to now compare the Torch and TensorFlow analysis of the MNIST dataset. The full MNIST data set was used and 10 epochs were performed and averaged over. The results are given in the table below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Linear(Torch) | Linear(TFlow) | MLP(Torch) | MLP(TFlow) | CNN(Torch) | CNN(TFlow) |
| Time(s) | 72.85 | 80.00 | 194.000 | 125.74 | 1524.4 | 282.92 |
| Accuracy(%) | 87.18 | 91.32 | 98.24 | 96.43 | 99.45 | 99.05 |

Here we see that there's is some dependence on the program used as to which is faster or more accurate. For the linear model, Torch is faster but TensorFlow is more accurate. For the MLP and CNN model, Torch is much slower but slightly more accurate. There is also a huge difference in speed between the CNN implementation in Torch and TensorFlow.

For the facial recognition project we first ran a gender classifier. This was run only on a two classification CNN model which was very computationally expensive. Just loading a trained model and guessing what one image’s gender was took on the order of seconds timewise. This is a very long time in comparison to the evaluation of MNIST data. We were able to do gender recognition in both Torch and TensorFlow. The results for an average over three different runs are summarized in the following table.

|  |  |  |
| --- | --- | --- |
| Program | Torch | TensorFlow |
| Time(minutes) | 82 | 2.7 |
| Accuracy(%) | 96.5 | 97 |

The paper by Levi[3] only quotes a maximum accuracy of approximately 80% for gender recognition. Our implementation seems to surpass this by quite a lot.

For the age recognition portion ages were split into the ten categories mentioned above. The time for the age recognition was similar to the gender although slightly longer. Again this was completed both in Torch and TensorFlow. The results for the average over three runs is summarized in the following table. Again we found better results then quoted by Levi [3].

|  |  |  |
| --- | --- | --- |
| Program | Torch | TensorFlow |
| Time(minutes) | 69 | 2.7 |
| Accuracy(%) | 95 | 99 |

**Conclusion**

The recommendation for fastest GPU machine learning code is TensorFlow. For the MNIST data set TensorFlow was slower for the linear analysis and slightly faster for the MLP model. When implementing a CNN though TensorFlow is significantly faster even for just the MNIST data set. For the facial recognition, which only uses CNN’s, TensorFlow maintains that speed increase over Torch. The CNN for the MNIST data set was 7x faster in TensorFlow while the CNN for the facial dataset was ~20x faster. This makes TensorFlow the clear choice for speed.

The recommendation for easier program to do machine learning is TensorFlow. The installation for Torch and TensorFlow for basic machine learning applications are very straightforward. However, once we began looking for more advanced applications it became clear the TensorFlow is the better program. One of the main reasons for that is that there are many excellent language modelling architectures already available in TF that you would probably have to make yourself in Torch. Another reason is that TensorFlow is installed within python. Python itself does a very nice job of handling additional packages and dependencies whether or not the user has root permissions. Torch, which is C++ based, uses as its primary interface the lua programming language. While Lua, like python has the ability to locally install packages, not nearly as many are openly available. This often requires the users to find the packaged and install them manually. This greatly inhibited the project leaving many Torch applications in disarray. If you are picking a deep learning program for ease of use TensorFlow is the way to go.

**Bibliography**

1. ***TensorFlow*. N.p., n.d. Web. 09 May 2017**
2. **"Scientific Computing for LuaJIT." *Torch*. N.p., n.d. Web. 09 May 2017**
3. **Levi, Gil, and Tal Hassncer. "Age and Gender Classification Using Convolutional Neural Networks." *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2015): n. pag. Web**