**Ben Walczak**

**Modeling the Speedup of using Varying Number of Threads for a Neural Network Classification Algorithm**

**Problem:**

The problem I will be exploring is testing the training speed of a neural network classification algorithm on varying numbers of thread counts per CPU and varying sample sizes of data. Understanding how the training speed changes according to thread count and sample size may provide insight as to when it will be useful to introduce more threads. This may help a company when attempting to allocate a discrete amount of resources for computing.

One of the most popular modules/frameworks used to build neural networks is TensorFlow. This particular module also has the advantage of controlling either the number of threads or the number of GPU’s while training a model. Therefore, TensorFlow will be used due to its popularity and capabilities of parallelization.

**Approach:**

The first step to this problem is to download all the necessary modules in Python required to compute this project. These modules include Pandas, Matplotlib, Time, NumPy, and TensorFlow. TensorFlow in particular requires many other modules and dependencies. Luckily, we will not be focusing on using the GPU version of TensorFlow as this module requires additional software that needs to be configured to the GPU’s. The latest version of TensorFlow, 1.13.0-rc2 should work fine to replicate this project.

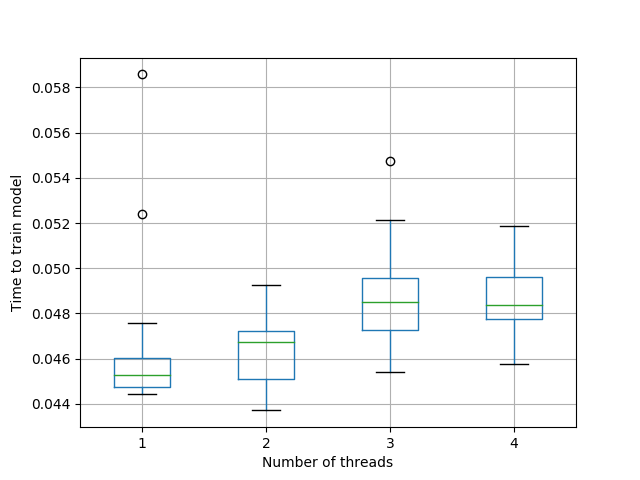
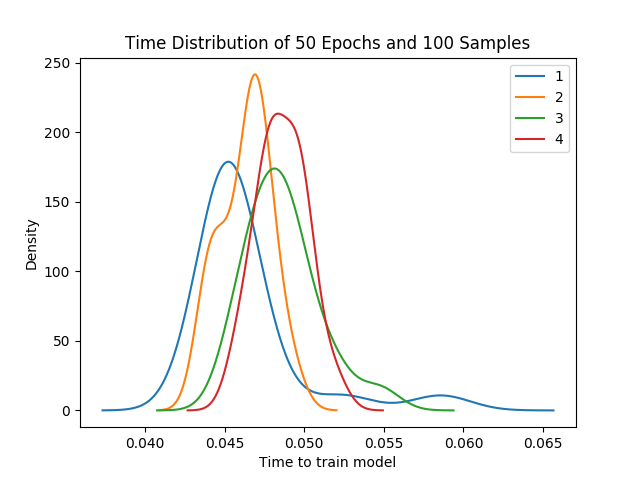
The neural networks that will be trained need data, so the next step will be to generate random data. The model will train at the same speed regardless of the distribution. Therefore, it will be unnecessary to shape the data in any way. Four columns of random data will be used to train the model to predict or classify the fifth column of 1’s and 0’s. The sample size of this randomly generated data will vary. Sample sizes to be used are N = 100, 1000, 10000, 100000, and 1000000. It is expected that a speedup will gradually occur at larger and larger sample sizes because it is known that communication can lower speedup at lower sample sizes.

After the data has been generated it must be prepared and built into the model. Once this is done, we can start the training process, but before we do so we will automate the process to test varying numbers of CPU threads and sample sizes. To acquire a more accurate evaluation of time, 20 iterations of training the neural network will be recorded for each thread count and sample size. This will give us a distribution of training times that will more accurately portray the general training time of the neural network. The number of threads to be used in training will be 1, 2, 3, and 4. Additional threads may be added in the future improvements of this project.

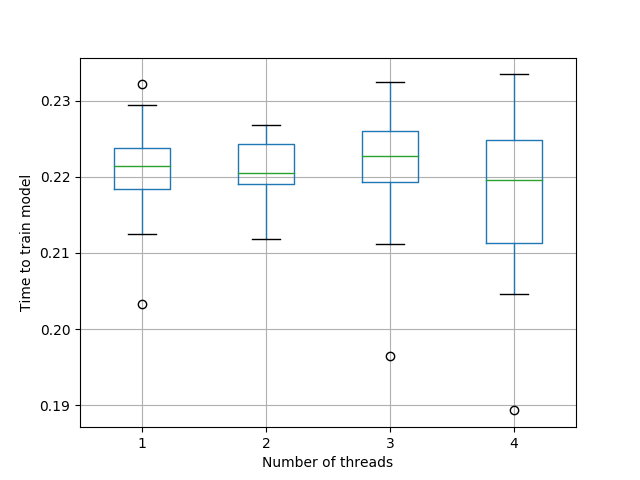
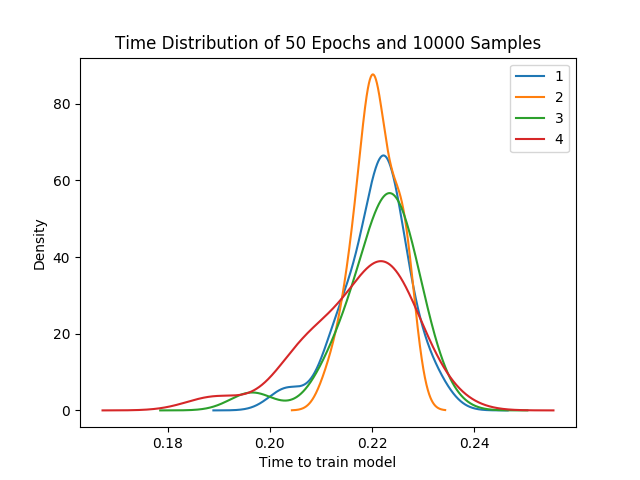
**Analysis:**

Time distributions will be displayed for each thread count (1, 2, 3, or 4) and its sample size alongside boxplots, which provide another view of the distribution. The blue, yellow, green, and red distributions are thread counts 1, 2, 3, and 4 respectively.

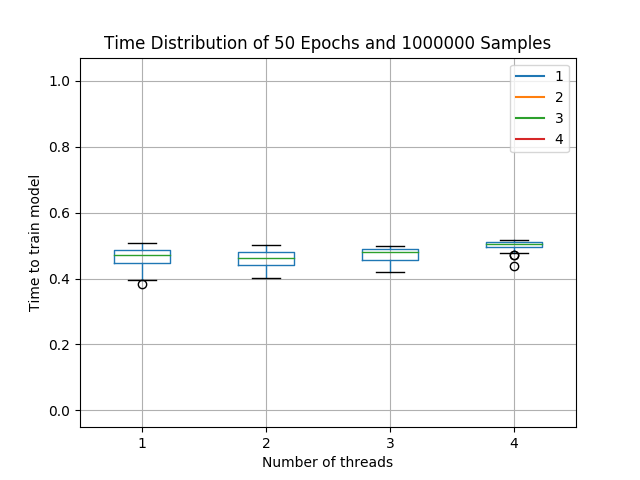
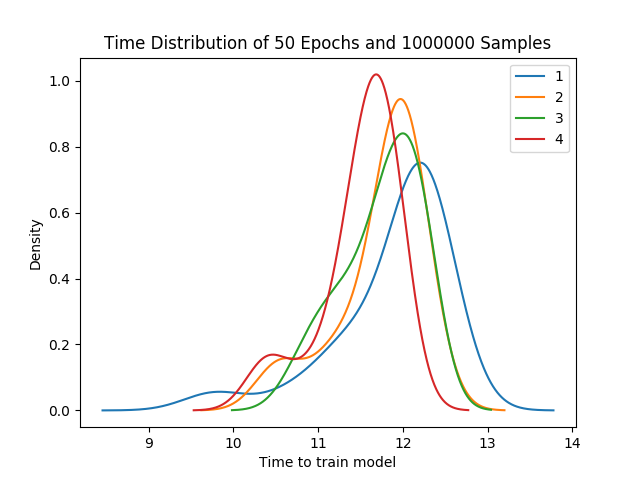
N = 100



N = 10000



N = 1000000



**Results:**

Below are the mean times for all iterations of a given thread count and sample size. The table that follows displays the speedup, calculated using the mean time for a single thread divided by the mean time of the number of threads used.

Mean time per thread and sample size:

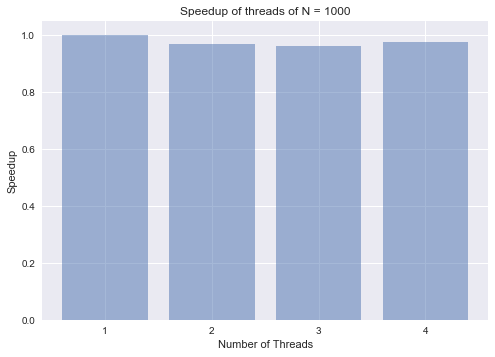
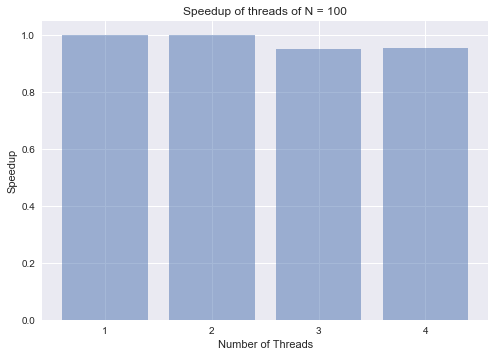
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N samples | 1 Threads | 2 Threads | 3 Threads | 4 Threads |
| 100 | 0.046 | 0.046 | 0.049 | 0.049 |
| 1000 | 0.090 | 0.093 | 0.094 | 0.093 |
| 10000 | 0.221 | 0.221 | 0.221 | 0.217 |
| 100000 | 1.505 | 1.571 | 1.549 | 1.619 |
| 1000000 | 11.877 | 11.736 | 11.739 | 11.488 |

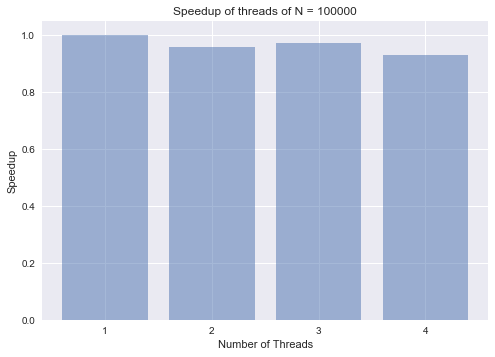
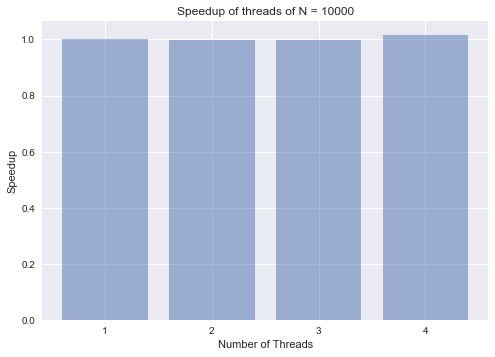
Speed up per mean time values:

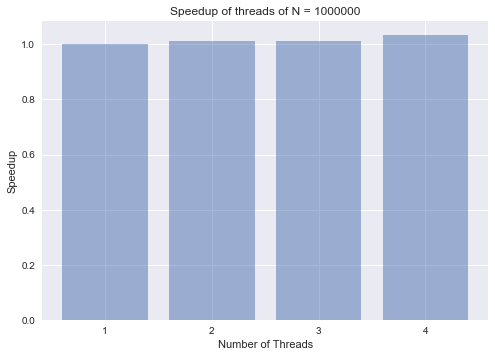
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N samples | 1 Threads | 2 Threads | 3 Threads | 4 Threads |
| 100 | 1.0 | 0.999 | 0.953 | 0.953 |
| 1000 | 1.0 | 0.968 | 0.962 | 0.976 |
| 10000 | 1.0 | 0.999 | 0.998 | 1.016 |
| 100000 | 1.0 | 0.957 | 0.971 | 0.930 |
| 1000000 | 1.0 | 1.012 | 1.012 | 1.034 |

Although there are hints of possible speed up at sample size 10,000, it does not become evident that a speedup from additional threads actually occurs until the sample size becomes 1,000,000.

Below are bar graphs of each speedup per thread count of each sample size:







Although the differences between each bar in each bar graph is relatively small, it does illustrate how the speedup varies from thread count to thread count. As mentioned previously, sample size of 1,000,000 is of interest because this is when speedup surpasses 1.0.

**Interpretation:**

Although it was interesting and exciting to finally see the speedup surpass 1.0 for the last sample size, the speed up is still relatively small. It is not significant enough for one to consider varying number of threads while training a neural network classification model. Perhaps, adding additional threads and sample sizes may yield interesting results in further replications of this experiment.

Initially, I had very different plans for this project. In the beginning, I wanted to test how varying the number of GPU’s affected training speed of neural networks. However, this proved difficult due to many configuration issues that TensorFlow presents. The coding part of this project was not the difficult part. Rather it was configuring TensorFlow with my local hardware or the hardware of the Everest server. Overall, I learned valuable information regarding how to use TensorFlow, how to install/configure TensorFlow or modules in general, how to better automate some functions, how to visualize results, how to calculate the speedup of a given algorithm, as well as many other things.