Performance of Tensorflow in a Limited Data Setting

Jean Feng

May 29, 2017

1 Introduction

Tensorflow has garnered a considerable amount of enthusiasm among communities that tackle problems with big data, such as image recognition and natural language processing. However there are many other datasets that are comparatively smaller, and adoption of Tensorflow to analyze such small/medium-sized data is limited. One reason is that in this limited-data setting, smaller models tend to perform better whereas Tensorflow is a machine learning system targeted for fitting large models in large-scale environments. Thus in the limited-data setting, the performance gain of using Tensorflow over other existing machine learning frameworks is unclear.

I'm interested in determining when one should use Tensorflow versus an out-of-the-box machine learning library. In particular, I will benchmark the performance of training neural networks via Tensorflow vs. Scikit-learn. In my experiments, I vary the size and structure of the neural networks to see how the two systems behave. The results show that Tensorflow actually offers little to no performance gain over Scikit-learn. This is quite surprising since Scikit-learn's implementation of neural networks is relatively simple whereas Tensorflow goes to greater lengths to optimize performance.

Given that Tensorflow does not provide major performance benefits, the main advantage of using Tensorflow in limited-data settings is that it allows the user to have fine-grained control over the structure of the neural network and customize the training algorithm used. To illustrate this, I give a brief discussion on implementation ease in Tensorflow versus Scikit-learn. In addition, I benchmark how using a custom algorithm can affect the speed of training a neural network.

2 The Limited Data Setting

For this report, "limited data" refers to the typical dataset found in statistics and biostatistics. In these fields, the usual dataset has no more than a hundred thousand features and no more than thousands of observations. In a limited data setting, the size of the machine learning model (e.g. the number of parameters) is also limited. For instance, if one was to fit a neural network given a thousand training observations, the number of hidden nodes should probably not exceed a hundred. This is due to the well-known bias variance tradeoff. If we consider fitting a complex model with many parameters, we expect that the model will have lower approximation error, also known as bias. However more complex models also tend to overfit the data and thus have higher "variance."

The practical implication of being in the "limited data" setting is that our data and models will be able to fit on a single machine.

3 Systems

In this section, I give a brief overview of Tensorflow and Scikit-learn. I will focus on system features that affect training speeds when the algorithm is run on a single machine.

3.1 Tensorflow

TensorFlow represents an algorithm using a dataflow graph where each node corresponds to an individual mathematical operation. Nodes in the graph can store the parameters of the algorithm, which can be updated by the user. A Tensorflow application is typically split into two steps: (1) constructing the dataflow graph and (2) executing the graph. By deferring execution until the complete graph is built, Tensorflow can optimize the execution phase by using global information about the computation. For instance, if one uses a GPU, Tensorflow can push a list of consecutive operations to the GPU so that intermediate results are not sent back to the CPU. Since we are training the model on a single machine, it will be interesting to see if Tensorflow is still well-optimized for this simple setting or whether it has been optimized for larger-scale systems.

Tensorflow also stores parameters of the algorithm as Variables, which are mutable buffers. Tensorflow stores the reference handles to these buffers so that it can update the parameter values. The user can update parameter values by calling the Assign operation. In the distributed setting, this operation will propagate updates to the model parameters throughout the network. In the single-machine setting, the operation should be extremely efficient.

3.2 Scikit-learn

Scikit-learn is a machine learning library in Python and provides a simple interface for users to fit models. The MLPRegressor class in Scikit-learn fits a multi-layer neural network for regression problems. The implementation of MLPRegressor is all written in Python, so one can treat Scikit-learn as an example of a python implementation of neural networks.

4 Performance Benchmarks: Tensorflow vs. Scikit-learn

In this section, I benchmark the speed of Tensorflow and Scikit-learn in training fully-connected linear-output neural networks using gradient descent. The performance benchmark is the amount of time spent per training iteration. I vary the size and structure of the neural network to show how performance changes. As we will see, the performance of Tensorflow and Scikit-learn are actually very similar.

4.1 Benchmark Environment

Since this is a limited data setting, I present single-machine benchmarks. The experiments were run on a four-core Intel Core i7 at 2.3 GHz. I compiled Tensorflow version 1.1.0 from the source to allow use of SSE4.1, SSE4.2, and AVX instructions. The simulation code used the python API. Training was performed with 32-bit floats. I used Scikit-learn version 0.18.1. All the experiments were run on simulated data.

One may think that Scikit-learn is at a disadvantage since it does not make use of vector instructions. However, Scikit-learn provides no faster options and the practitioner must choose between the simple implementation in Scikit-learn vs. the highly-optimized Tensorflow package. These benchmark settings are meant to be reflective of this choice the practitioner must make.

Finally, note that these simulation settings only consider neural networks with 10 to 320 hidden nodes. I do not consider larger models since they are uncommon in limited data settings.

4.2 Changing the number of nodes

For the first experiment, I considered a neural network with one hidden layer and varied the number of input features p and the number of hidden nodes in the neural network. The results are given in Figure 1. As we can see, the time required for the training step increases with the number of hidden nodes and the number of input features for both systems. The difference in training time of the two

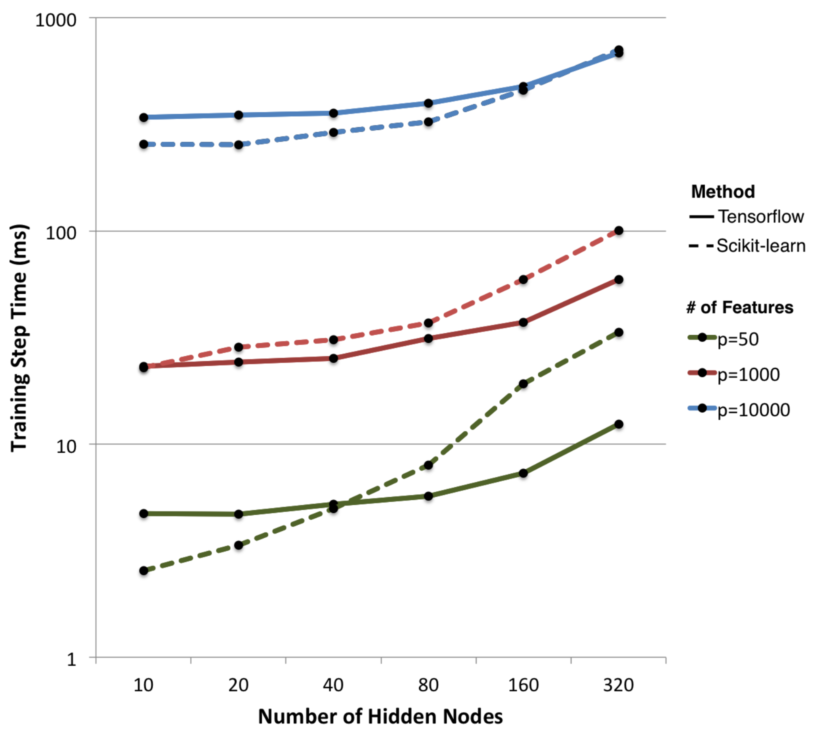

Figure 1: Training step time for a single hidden-layer neural network.

systems is minor for the three different values of p considered. The largest difference between the curves is when $p = 50$. However the training step time in this case is on the order of tens of milliseconds, so this difference is inconsequential to the user.

4.3 Varying the number of layers

In this next experiment, I varied the number of hidden layers in the neural network. The number of input features was fixed at 1000 and every hidden layer had ten nodes. As seen in Figure 2, both Tensorflow and Scikit-learn require more computation time as the number of layers increases. As before, the difference in training speed using Tensorflow and Scikit-learn is minor.

Remarks From the benchmark experiments in this section, we see that Tensorflow offers little to no performance gain compared to Scikit-learn. This is quite surprising since Scikit-learn is a relatively simple implementation of neural networks in python. Thus from a practitioner's point of view, performance is not a good reason to switch from Scikit-learn to Tensorflow.

5 Tensorflow's Flexibility

The results in the previous section show that Tensorflow is not significantly faster than Scikit-learn. Instead, I believe the main advantage of using Tensorflow is its flexibility. The system gives the user fine-grained control over model construction and training, which is very important for developing new models and testing new algorithms. In this section, we will discuss implementation ease in Tensorflow and how performance changes when using a custom algorithm.

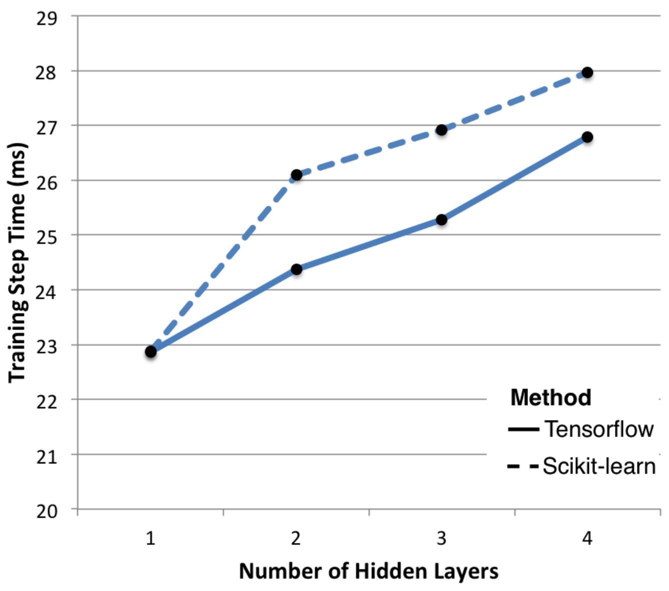

Figure 2: Training step time for multi-layer neural networks with 1000 input features and 10 nodes per hidden layer.

5.1 Implementation ease in Tensorflow

If we implement the standard fully-connected multi-layer neural network, Tensorflow requires many more lines of code compared to an out-of-the-box framework like Scikit-learn. For instance, my implementation used about 150 lines of code, whereas the same neural network in Scikit-learn requires at most twenty lines of code.

The benefits of Tensorflow are much more apparent when we want to implement a custom neural network structure or run a custom training algorithm. In this case, implementing these customizations in Tensorflow is very natural since the user has full control over the graph structure and execution. The amount of code one needs to write for a custom neural network structure/algorithm should be roughly on the same order as writing the standard neural network. On the other hand, the usual interface that Scikit-learn provides would not allow for a different neural network structure or training algorithm. The user would need to customize the library and write vastly more code than the original 20 lines.

5.2 Custom training algorithms

Given that implementing a custom training algorithm in Tensorflow is relatively straightforward, a natural question to ask is whether Tensorflow's performance is worse when one uses a custom training algorithm. In particular, we might be concerned that the training time could increase significantly if Tensorflow needs to continually propagate parameter updates to the dataflow graph.

To see how a custom algorithm affects performance, I considered an artificial training algorithm where each iteration is composed of a gradient descent step and the Assign operation that propagates new parameter values to the model. In practice, computational time is also required for determining what the new parameter values are, but this is highly dependent on the training algorithm used. In order to isolate the time to propagate parameter values to the model, I only consider this artificial training algorithm.

For the experiment, I fit a fully-connected neural network with 50 input features and a single

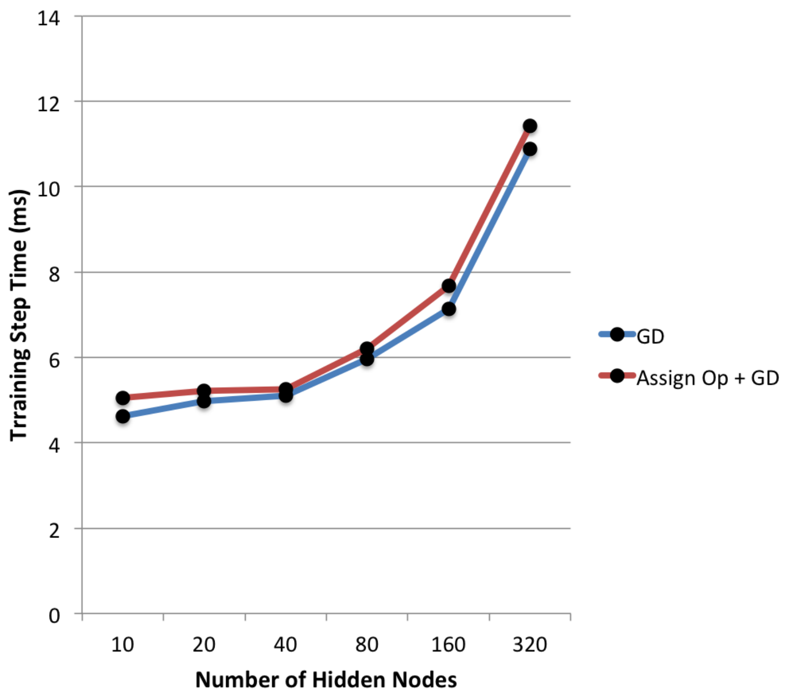

Figure 3: Training step time for using standard gradient descent (GD) vs. gradient descent with an additional Assign operation (Assign Op + GD).

hidden layer. I compare fitting the neural network using the standard gradient descent algorithm to the artificial training algorithm with a gradient step and an Assign operation. As shown in Figure 3, the training time for both training algorithms is essentially the same. Thus pushing parameter updates to the graph when running Tensorflow on a single machine is very fast. For the practitioner, this is good news since this means that using a custom algorithm does not drastically increase the training time.

6 Discussion

In this report, I compared TensorFlow and Scikit-learn in the limited-data setting to see why one should use one system over the other. My results show that the training times using the two systems are very similar. Thus switching from Scikit-learn to Tensorflow solely based on training speed is not a good reason. Instead, the main advantages of using Tensorflow are that it gives the user fine-grained control over the construction of the model and the training algorithm. In addition, if the user wants the ability to scale up their model to larger data, Tensorflow can easily scale up the computation and take advantage of GPUs or a distributed network.