Xiaobing Li, PhD

An ensemble Tensorflow CNN model on Spark

Summary *Tensorflow is a popular deep learning library for GPU and CPUs but the current version is not scalable. Here I proposed a simple design to train multiple tensorflow CNN models on a Spark cluster with random data samples from MNIST datasets. The final prediction was made by taking the majority vote of all the parallel models. In the early stage testing, this ensemble model improved the accuracy of predictions and reduced the running time of the whole training process, comparing to a single-core model.*

In Nov. 2015, Google released Tensorflow, a deep learning library for GPU and CPUs. It soon becomes a popular framework for the deep learning research and development. To train a complicated deep learning network with a huge dataset, it will take a high-quality GPU computer hours or days. Although a modern GPU can have hundreds of cores, a huge dataset will always be a burden to a GPU computer. At the same time, a traditional company, like Amex, has huge Hadoop/Spark clusters for big data but no GPU clusters. If we can run tensorflow models on current Hadoop/spark nodes, it will be a huge save for our current resources.

The idea to combine tensorflow and Spark is not new as Databricks[1] and Yahoo[2] have proposed their ways to realize it. However, my idea of distributing multiple CNN models on Spark to do an ensemble learning is new and highly scalable.

Inspired by RandomForest, I built multiple tensorflow CNN models for Spark nodes and randomly distributed a subset of the training data to each model for training. The final predictions for a test set were made by taking the majority vote of all the models.

I developed this design and implemented the code in my private time and did some early stage testing. The results were quite promising. The ensemble model has better performances in terms of speed and accuracy comparing to an identical single-core model. The model/design is still in an un-matured stage. Further optimizations and modifications are required.

Environment

The single-core model was running on a 16-core, 128G-memory linux/unix server. The ensemble model was running on a Spark YARN cluster consisted of four of similar servers. The available executors were 75 and I used the default 2G memory for each executor (more detailed information skipped).

The code was written in Python, with tensorflow, numpy, pandas and pyspark libraries used. The ensemble model was deployed on the cluster by using spark-submit with parameters.

Model Implementation

To simplify the comparison, I created a CNNmodel class with identical parameters for the CNN graph in the class constructor. The model was consisted of two convolutional layers with ReLU activation and max-pooling and two fully-connected layers. An Adam optimizer was used with a learning rate 0.0001 and no regularization was applied at the moment (detailed parameter values skipped).

To estimate the influence of data volume on models, I extracted the MNIST train dataset into a plain text format and combined the images and labels in a matrix. The size of the file was about 1GB. I also made another train dataset with 10-time duplications of the original MNIST. The size of this folder was about 11GB.

For the single-core model, I simply called a Tensorflow session to run the full MNIST training dataset and make the predictions for 1000 records from the test dataset. The model was running with a batch-size 50. This part was done by a train() function.

For the ensemble model, I first loaded the training data into a RDD. After parsing, I randomly split the datasets into 100 partitions, which then were mapped to a similar train() function as above. So, each node would only train 1% of the whole dataset. In practical, I would repeat the training 2-5 epochs to achieve a better accuracy.

Result

I did some early stage testing without any optimization. Below are some results:

Data volume   The single-core model can load 1G training data without problem, but it was frozen with 11G data. The ensemble mode had no problem to load both datasets. Due to the characteristics of HDFS, this model can deal with even 10x or 100x more data without any problem. It was expected and not surprising.

Performance   The single-core model was trained on 1G data for 7min29secs with an accuracy at 84.2% (this value is very bad. Again, no optimization was done at the moment.) . The ensemble model finished a 2-epoch training on 11G data in 5min21s with an accuracy at 85.1%; finished a 3-epoch training on 11G data in 6min4s with an accuracy 90.5% and a 5-epoch training in 10min35s with an accuracy 94.7%.

Conclusion

The early results are quite promising. With huge datasets, the ensemble model not only can finish the training for the whole dataset in a short period of time, the ensemble nature will also boost the accuracy of prediction.

In the next step, I will fix some issues that I found during the testing and do more optimizations both on the model parameters and parallel computing. The current result is still not ideal but this model has its own advantage especially for huge datasets. The same architecture can be applied to a distributed GPU clusters as well.

References

[1] https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html

[2] https://github.com/yahoo/TensorFlowOnSpark