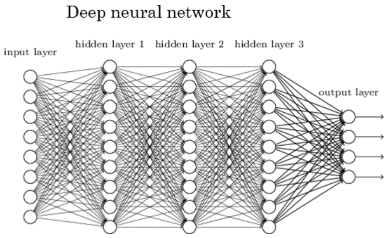
Distributed Stochastic Gradient Descent

Kevin Yang and Michael Farrell

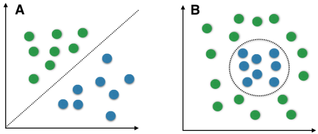
May 9, 2016

Motivation - Deep Learning

- Deep-Learning
  - Objective: Learn a complicated, non-linear function that minimizes some loss function
- Why do we need deep models?
  - The class of linear functions is inadequate for many problems.

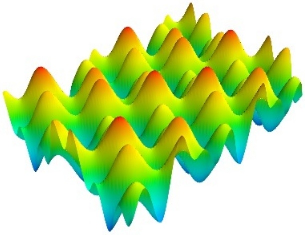


http://www.rsipvision.com/exploring-deep-learning/



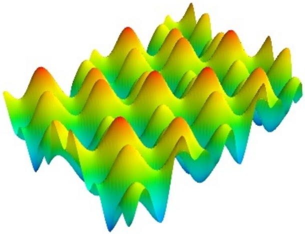
http://sebastianraschka.com/Articles/2014_naive_bayes_1.html

Motivation - Deep Learning



- How do we learn these deep models?
  - Choose a random example
  - Run the neural network on the example
  - Adjust the parameters of the network such that our loss function is minimized more than it was before
  - Repeat
- Difficulties?
  - Local Minima
  - Non-convexity
  - Neural Networks can have millions or even billions of parameters

Motivation - Deep Learning



- How do we learn these deep models?
  - Choose a random example
  - Run the neural network on the example
  - Adjust the parameters of the network such that our loss function is minimized more than it was before
  - Repeat
- Difficulties?
  - Local Minima
  - Non-convexity
  - Neural Networks can have millions or even billions of parameters

Motivation - SGD

- How do we maximize our reward function?
  - One common technique is Stochastic Gradient Descent
  - $\mathbf{w}$ is the vector of parameters for the model
  - η is the learning rate
  - $\mathbf{f}(\mathbf{w})$ is the loss function evaluated with the current parameters $\mathbf{w}$
  - $\mathbf{w} \leftarrow \mathbf{0}$

    while $\mathbf{f}(\mathbf{w})$ is not minimized **do**

        for $i = 1, n$ **do**

            $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla f(\mathbf{w})$
  - As the number of training examples, n, and the number of parameters, $|\mathbf{w}|$, increases, this algorithm quickly becomes very slow...

Motivation - Distributed SGD

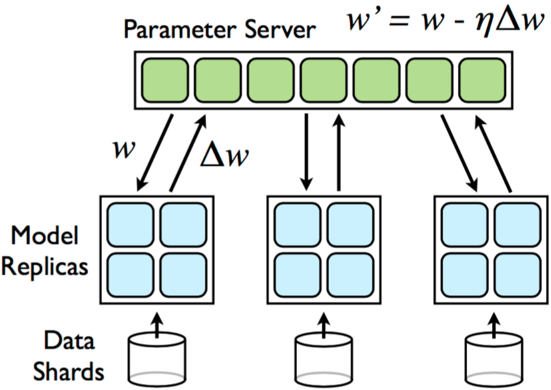
- Since some of these models take days/weeks/months to run, we would hope that we could use a distributed computing cluster in order to parallelize this process.
- Learn from Google!
  - DistBelief- 2012
    - Downpour SGD
    - Sandblaster L-BFGS
  - TensorFlow- 2015
    - gRPC

Motivation - Distributed SGD

- Since some of these models take days/weeks/months to run, we would hope that we could use a distributed computing cluster in order to parallelize this process.
- Learn from Google!
  - DistBelief- 2012
    - Downpour SGD
    - Sandblaster L-BFGS
  - TensorFlow- 2015
    - gRPC

DistBelief - Downpour SGD

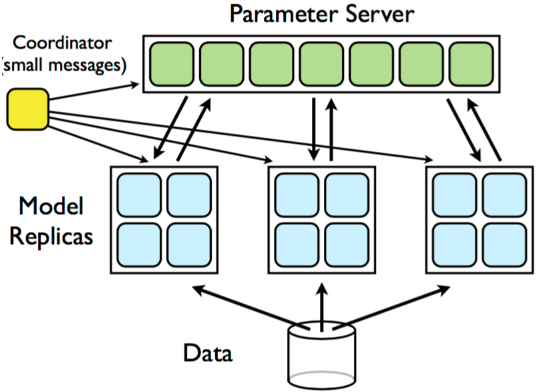
- "An asynchronous stochastic gradient descent procedure supporting a large number of model replicas." [1]



[1] Diagram taken from Dean et al. *Large Scale Distributed Deep Networks*

DistBelief - Sandblaster L-BFGS

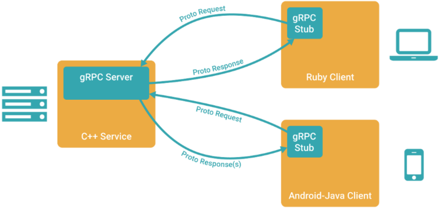
- "A framework that supports a variety of distributed batch optimization procedures, including a distributed implementation of L-BFGS" [2]



[2] Diagram taken from Dean et al. *Large Scale Distributed Deep Networks*

TensorFlow-GRPC

- Second Generation ML Model focused on distributing models to CPUs and GPUs
- Uses the high performance RPC framework (GRPC [3]) in order to communicate between separate processes
  - Uses Protocol Buffers -v3
  - C-based
  - Client-server stubs in 10+ languages and counting



[3]Diagram taken from http://www.grpc.io/

DistBelief/TensorFlow Summary

- TensorFlow is basically the second version of DistBelief that is approximately twice as fast and much more user-friendly.
- Results from DistBelief' [4]:

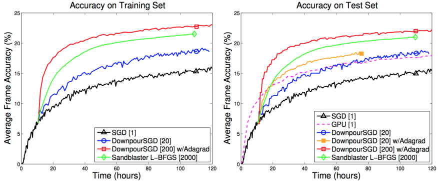


Figure 4: Left: Training accuracy (on a portion of the training set) for different optimization methods. Right: Classification accuracy on the hold out test set as a function of training time. Downpour and Sandblaster experiments initialized using the same ∼10 hour warmstart of simple SGD.

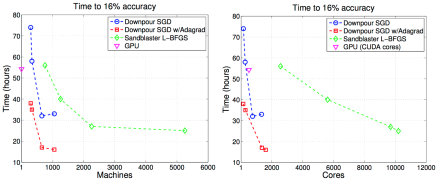


Figure 5: Time to reach a fixed accuracy (16%) for different optimization strategies as a function of number of the machines (left) and cores (right).

[4] Diagram taken from Dean et al. *Large Scale Distributed Deep Networks*

Our Project

- We frequently run into scenarios where we have a model that trains incredibly slowly on our local machines. As a consequence, we hope to benefit from additional cloud computing resources and build our own Distributed SGD system based on DistBelief and TensorFlow systems.
  - The Distributed SGD system will have the user give a function that returns the outputs of a model, a function that returns the gradients of a model, and the number of machines to train the model on.
  - Use GRPC with Protocol Buffers to communicated between processes, similar to TensorFlow.
  - Implement Downpour-SGD which seems to be the most effective model with limited resources.

Our Example

- To test our system, we're working with the Caltech 101 Computational Vision dataset [5]. In this dataset, there are about 20,000 pictures of objects in 101 categories. All of these images are around 300 x 200 pixels in size.
- We've implemented a convolutional neural net that tries to classify what object is represented in the image.

[5] L. Fei-Fei, R. Fergus and P. Perona. *Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories.*

Computational Resources

- We are using Google Cloud Compute Engine to set up VMs and run the code. To run classification on our image dataset, we're using small instances with 6GB of RAM with 2 cores. This has a rate of 7.8 cents per hour.
- On a machine of this size, running 10 epochs of gradient descent takes 56 minutes.
- To streamline things, we've preconfigured images of a parameter server and model training server that are already set up with relevant code, tools, and libraries.
- As a result, setting up and launching the compute instances necessary for model training takes only a couple lines.

Implementing Downpour-SGD

- The Downpour-SGD requires the passing of parameters and parameter updates between processes. In our example, we have 74,770,901 parameters and the size of our parameters is 0.5GB.
- Bottleneck here is the network. Parameters can be $>>$0.5Gb.
- We can leverage the fact that some of these models are extremely sparse
  - only send parameters updated
  - only update parameters every n_x times
- Explore protocol buffer streams

Large Data Sets

Protocol Buffers are not designed to handle large messages. As a general rule of thumb, if you are dealing in messages larger than a megabyte each, it may be time to consider an alternate strategy.

That said, Protocol Buffers are great for handling individual messages *within* a large data set. Usually, large data sets are really just a collection of small pieces, where each small piece may be a structured piece of data. Even though Protocol Buffers cannot handle the entire set at once, using Protocol Buffers to encode each piece greatly simplifies your problem: now all you need is to handle a set of byte strings rather than a set of structures.

Main Distributed System Challenges

- Network Issues
  - We have to deal with network latency and try to reduce transportation cost as much as possible in order for our models to train properly.
  - We would like to experiment with a couple different RPCs to optimize the speed of our system.
- Fault tolerance
  - We need to make our system as resilient as possible against failures. Because all of these machines are doing a lot of computation while running gradient descent and manipulating parameters, these systems are bound to fail with relatively high frequently.
  - Having methods in place to detect and remedy the failure of parameter servers and model replicas will be critical.