10.01 - Large Scale Machine Learning

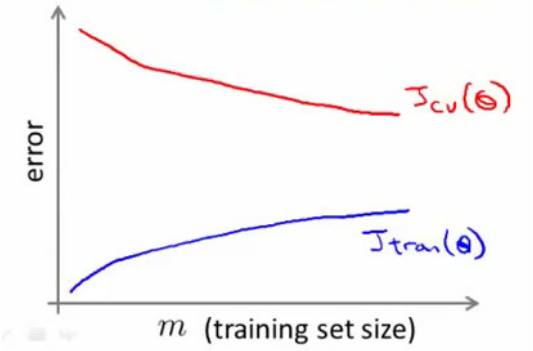

Figure 1: Learning curves with high variance

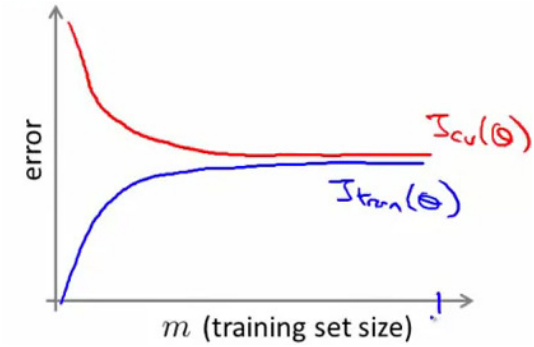

Figure 2: Learning curves with high bias

1 Learning With Large Datasets

One of the reasons why algorithms are getting better is to simply get more data.

> "It's not who has the best algorithm that wins. It's who has the most data."

Learning with large datasets comes with a price: computational time (they run very slow). A solution for that is to simply take a randomized subset of the whole data. We can plot the learning curves to see if that works. For example, in fig. 1, we can see that getting more data is likely to help when we have high variance. On the other hand, in fig. 2 we can see that, when we have high bias, the amount of data doesn't matter after a certain point. In this last case, we can think about adding more features/more hidden layer units.

2 Stochastic Gradient Descent

This algorithm is a modification to the standard gradient descent (called "Batch gradient descent", as it uses all the data on every iteration) that allows us to train better on large scale datasets.

In this new algorithm, we modify J_{train} to look like this (which is the same as the batch one but written in a different way):

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)})) \tag{2}$$

We can see a description of the algorithm in algorithm 1.

It basically goes example by example, each time fitting the θ values a little bit better. What batch did was to have a sumation for every example in the data, here we do the descent example by example.

In conclusion, stochastic can run much faster in huge datasets at the cost of giving an approximation of the local minimum instead of the exact local minimum.

Algorithm 1 Stochastic gradient descent

```
Randomly shuffle dataset.

repeat
    for i ← 1, m do                    ▷ m is the amount of examples in the dataset
        for j ← 1, n do                ▷ n is the amount of θ to fit
            θ_j := θ_j − α(h_θ(x^(i)) − y^(i)) x_j^(i)
        end for
    end for
until amount of times                  ▷ Hyperparameter: if dataset is large 1 may be enough
```

3 Mini-Batch Gradient Descent

- Batch gradient descent = use all m examples in each iteration.
- Stochastic gradient descent = use 1 example in each iteration.
- Mini-Batch gradient descent = use b examples in each iteration.

b is the mini-batch size, with a typical size of 2-100. It might be faster if the implementation is well vectorized.

4 Stochastic Gradient Descent Convergence

Back when we were using batch, we could plot $J_{train}(\theta)$ as a function of the number of iterations of gradient descent and check whether it was always decreasing or not.

In stochastic, we can't do that, though, as in order to plot $J_{train}(\theta)$ we need to iterate over all m. What we can do is, during learning, compute the *cost* of the actual example before updating θ. Then, every 1000 iterations (say), we can plot the *cost* averaged over the last 1000 examples. This graphic can be noisy and not decrease in some points, as we can see in fig. 3. If we see in the graph that the line isn't decreasing, we could try to increase the amount of iteration we average (fig. 4).

4.1 Learning rate tunning

Stochastic gradient descent gets near the local minimum and then wanders around, never converging. If we want to try to make it converge, we can slowly decrease α over time, for example with eq. (3). This comes at the cost of tunning extra hyperparameters.

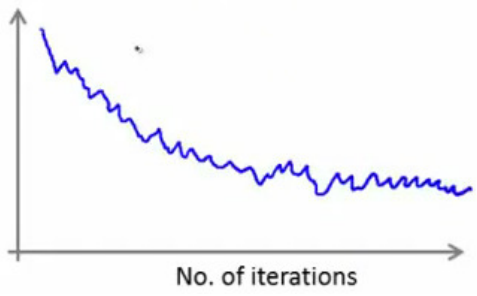

Figure 3: Ideal plot of the stochastic gradient descent

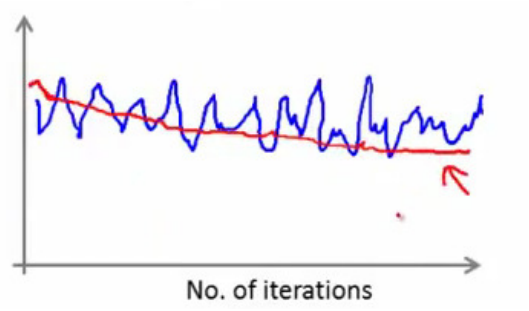

Figure 4: Noisy plot (blue) vs plot with a larger number of iterations (red) of the stochastic gradient descent

$$\alpha = \frac{const1}{iterationNumber + const2} \tag{3}$$

5 Online learning

This style of learning happens when we have a constant stream of data to learn from, ie a website.

An example of that would be a shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ($y = 1$), sometimes not ($y = 0$).

Features x capture properties of user, of origin/destination and asking price. We want to learn $p(y = 1|x; \theta)$ to optimize price.

We will use algorithm 2 for that. Basically, it's very similar to stochastic gradient descent but we discard the user data after using it. A nice property of this algorithm is that it adapts if your users change over time.

6 Map Reduce and Data Parallelism

Sometimes we just have so much data that we can't train using a single computer. We can use Map Reduce in order to solve that problem.

The idea is: imagine that we have 4 machines and are using batch gradient descent. We can have each of the machines compute 1/4 of the sum needed for a gradient descent step and then

Algorithm 2 Online Learning

while forever **do**
    Wait until a new example comes in from the stream.
    Get the example x and y.
    for $j \leftarrow 1, n$ **do** ▷ n is the amount of θ to fit
        $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j$
    end for
end while

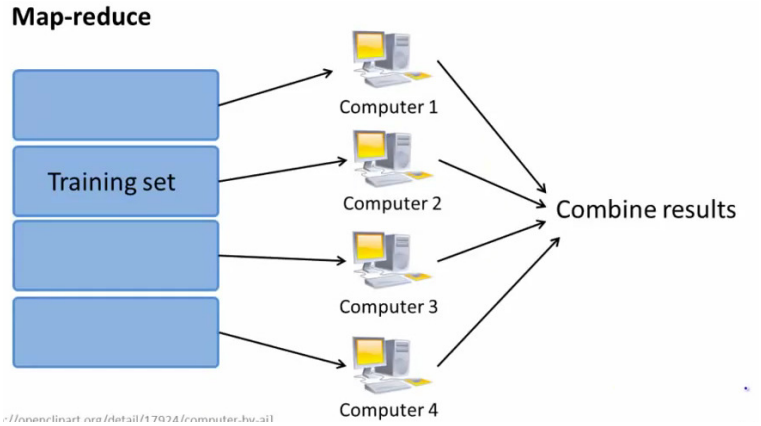

Figure 5: Scheme of a map reduce solution

collaborate with the result of the partial sums.

We can look at that visually in fig. 5.

Many learning algorithms can be expressed as computing sums of cuntions over the training set. This are the types of algorithms that can be run with map-reduce. A good place to look for is the sums over all the dataset ($\sum_{i=1}^{m}$), which can be parallelized.

Map reduce can even be useful in a single computer. We can simply send each job to a core instead of another computer.