Week 17

August 20, 2021

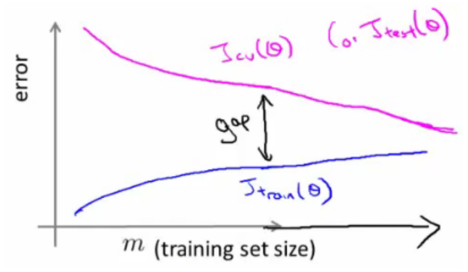
Learning with large datasets

Taking a low bias algorithm and training it on a large amount of data is one of the greatest approaches to get high performance.

- Assume we have a data collection with m = 100,000, 000.
- On such a large system, how can we train a logistic regression model?

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- As a result, for each gradient descent step, you must add up more than 100,000,000 terms.
- The first step is to inquire whether we may train on 1000 examples rather than 100 000 000.
- Pick a small sample at random. Can you build a system that performs well?



- If the gap is large, it is a high variance problem. More examples should lead to better results.
- If the gap is tiny, it is a high bias problem. More examples may not be beneficial.

Stochastic Gradient Descent

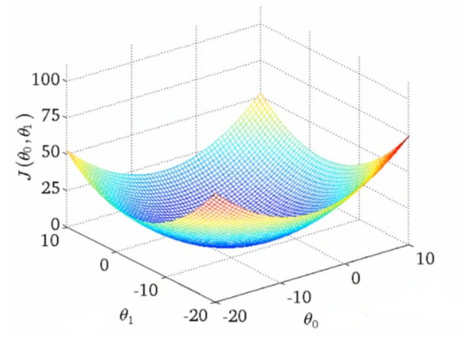
Hypothesis:

$$h_\theta(x) = \sum_{n}^{j=0} \theta_j x_j$$

Cost function:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{m}^{i=1} (h_\theta(x^{(i)}) - y^{(i)})^2$$

We get bowl shape surface plots if we plot our two parameters against the cost function:



How does gradient descent work?

Algorithm 1 Gradient Descent

```
1: while not converged do
2:     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, ..., \theta_n)$
3:     $(for\ j = 0, ..., n)$
```

Although we have just referred to it as gradient descent so far, this type of gradient descent is known as batch gradient descent. This simply implies that we examine all of the examples at the same time. Batch gradient descent is unsuitable for large datasets.

When discussing stochastic gradient descent, we will use linear regression as an algorithmic example, however the concepts apply to other algorithms as well, such as logistic regression and neural networks.

Define our cost function in a slightly different way, as follows:

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

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

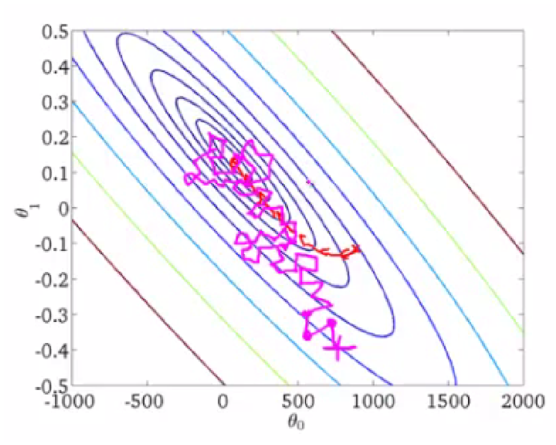
Algorithm 2 Stochastic Gradient Descent

```
1: Randomly shuffle training examples
2: for $i := 1, ..., m$ do
3:     $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
4:     $(for\ \ j = 0, ..., n)$
```

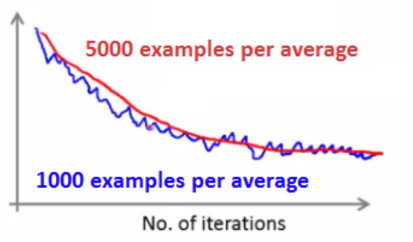
The random shuffling at the start ensures that the data is in a random sequence, preventing bias in the movement.

Although stochastic gradient descent is similar to batch gradient descent, instead of waiting for the gradient terms to be summed across all m examples, we pick only one example and make work on enhancing the parameters right away.

This means that we change the parameters on EVERY step through the data, rather than at the end of each loop over all of the data.



For our stochastic gradient descent, we might plot the cost function vs the number of iterations. We should be able to see if convergence is occurring by looking at the graphs. A smoother curve may be obtained by averaging across many (e.g. 1000 and 5000) instances.



Online learning

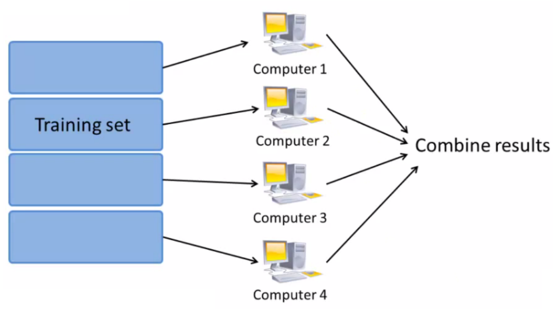
- Allows us to model problems in which there is a continuous stream of data from which an algorithm should learn.
- In the same way that stochastic gradient descent is used, slow updates are performed.
- To learn from traffic, web companies employ several sorts of online learning algorithms.

Example - product search

- Assume you own a cellphone-selling website.
- You have a user interface where the user may enter in a query such as "Android phone 1080p camera."
- We want to provide the user ten phones per query, with the phones ranked from most appealing to the user.
- We generate a feature vector (x) for each phone based on a specific user query.
- We want to determine the likelihood of a user picking a phone.
- $y = 1$ if a user clicks on a link.
- $y = 0$ otherwise.
- We learn $p(y = 1|x; \theta)$ - this is the problem of learning the predicted click through rate (CTR).
- If you can estimate the CTR for any phone, we can utilize it to show the phones with the highest likelihood first.

Map reduce and data parallelism

Some problems are simply too large for a single CPU to handle.



Parallelization can come from:

- Multiple machines.
- Multiple CPUs.
- Multiple cores in each CPU.

Certain numerical linear algebra libraries can automatically parallelize your calculations over several cores, depending on the implementation details. So, if this is the case and you have a decent vectorization implementation, you don't have to worry about local parallelization because the local libraries will handle optimization for you.

Hadoop is an example of Map Reduce implementation.