Scaling to Huge Datasets & Online Learning

Gradient ascent won't scale to today's huge datasets. Each iteration of gradient ascent requires a full pass over each row of input data for each feature in order to calculate the partial derivative (contribution to the gradient) for each feature and to use it to update the coefficient for that feature. Each iteration is very time consuming when there is a lot of data (especially if it does not all fit in RAM, so we end up reading from disk).

Data sets can be huge. Think about 4.8 billion web pages. Think about the 500 million tweets that are posted a day.

You tube shows 5 billion video views a day. They have a Machine Learning algorithm that figures out what ad to show for each of those views. So that ML algorithm must make the choice 5 billion times per day. Moreover, it must choose the ad within milliseconds so the ad is shown in a timely manner. Finally, it must be updated as user choices change to account for the 300 hours of new video that is posted each minute.

Timeline of scalable machine learning & stochastic gradient
Over time, Machine Learning has improved as data sets have gotten larger.

1996 – Small Data

- Relatively small data sets
- Complex models to pull as much accuracy as possible out of the small amount of data
    - Kernels
    - Graphical Models

2006 – Big Data

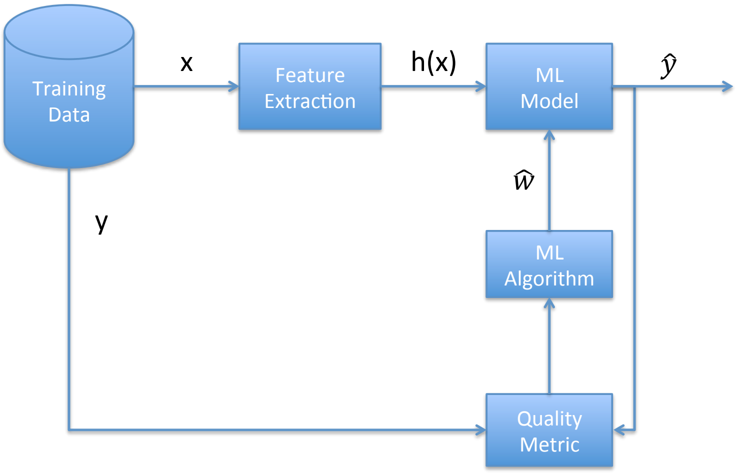
- Data sets have gotten very large
- Models get simpler so they can run fast enough to use so much data.
    - Logistic Regression
    - Matrix Factorization
- 2009, Halevy, Norvig, Pereira, “The Unreasonable Effectiveness of Data” describes how a large amount of data yields better accuracy even with simpler models

2016 – Bigger Data

- Back to complex algorithms to improve accuracy
    - Boosted Trees
    - Tensor factorization
    - Deep Learning
    - Massive Graphical models
- Parallelism, GPUs and computing clusters to make the complex algorithms fast enough

In moving to massive data sets, our basic ML flow does not change;

Machine Learning Workflow



In order to scale to massive data sets, we need to make a change to our algorithm. We move from Gradient Ascent to Stochastic Gradient ascent. This turns out to be a relatively small change. Rather than pass over all rows of data to calculate the gradient, we pass over some subset, which is different on each pass, and use that subset to calculate the partial differential (contribute to the gradient) for each feature and to update each coefficient.

This makes some sense. For really massive data sets, we can still have a lot of data in a subset. If we choose a different subset each time, we will avoid overfitting on that subset.

Scaling ML with stochastic gradient

Why gradient ascent won't scale

Gradient Ascent Algorithm involves calculating the gradient of the likelihood function and using that to update the coefficients.

The partial derivative of the log-likelihood with respect for feature j is given by;

$$\frac{\partial l(w)}{\partial w_j} = \sum_{i=1}^{N} h_j(x_i)(1[y_i = +1] - P(y = +1|x_i, w))$$

N is the number of data points (rows of feature matrix H)

i is the index of the data (the ith data input, ith row of feature matrix H)

$h_j(x_i)$ is the jth feature of the ith data input

$P(y = +1|x_i, w)$ is the prediction that x_i is positive for the given set of coefficients w.

$1[y_i = +1]$ is the indicator function. This outputs 1 if the known output value y_i is labeled as +1 and 0 if y_i is labeled as -1.

$$indicator(y_i) = 1[y_i = +1] = \begin{cases} 1 \; if \; y_i = 1 \\ 0 \; if \; y_i = -1 \end{cases}$$

This is the contribution of data point (x_i, y_i) to the partial derivative for feature j:

$$\frac{\partial l_i(w)}{\partial w_j} = h_j(x_i)(1[y_i = +1] - P(y = +1|x_i, w))$$

We must sum those contributions over all the data points to calculate the contribution to the gradient of feature j. We must then do that for all the other features and sum the partials to compute the full gradient. The whole algorithm is:

```
While not converged:
  $\hat{w}^{(t+1)} \leftarrow \hat{w}^{(t)} + \eta \nabla l\left(\hat{w}^{(t)}\right)$
```

Rather than look at this in terms of the number of operations, we can simplify and look at it as time.

Compute contribution to the gradient of one data point	# of data points	Time to compute one step of the gradient
1 millisecond	1000	1 second
1 second	1000	16.7 minutes
1 millisecond	10 million	2.8 hours
1 millisecond	10 billion	115.7 days

Stochastic gradient- Learning one data point at a time

In normal gradient ascent, we use all data points to calculate the gradient.

$$\frac{\partial l(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial l_i(w)}{\partial w_j}$$

In stochastic gradient ascent, we can use a single point as an approximation of the gradient;

$$\frac{\partial l(w)}{\partial w_j} \approx \frac{\partial l_i(w)}{\partial w_j}$$

With each iteration of the algorithm, we pick a different single point to approximate the gradient.

The original algorithm was:

```
Gradient Ascent for Logistic Regression

At t=1, initialize $\hat{w}^{(1)} = 0$, or some other smart choice.
while $\left\|\nabla l(\hat{w}^{(t)})\right\| > \varepsilon$ do
  for j = 0..D
    partial[j]=$\frac{\partial l(w)}{\partial w_j} = \sum_{i=1}^{N} h_j(x_i)\big(1[y_i = +1] - P(y = +1|x_i, w^{(t)})\big)$
    $\hat{w}^{(t+1)} = \hat{w}^{(t)} + \eta \times partial[j]$
  t = t + 1
```

The stochastic gradient ascent with a single point is:

```
Stochastic Gradient Ascent for Logistic Regression

At t=1, initialize $\hat{w}^{(1)} = 0$, or some other smart choice.
while $\left\|\nabla l(\hat{w}^{(t)})\right\| > \varepsilon$ do
  for i = 1 to N
    for j = 0..D
      partial[j]≈ $\frac{\partial l_i(w)}{\partial w_j} = h_j(x_i)\big(1[y_i = +1] - P(y = +1|x_i, w^{(t)})\big)$
      $\hat{w}^{(t+1)} = \hat{w}^{(t)} + \eta \times partial[j]$
  t = t + 1
```

We have introduced a loop around the feature loop that visits each data point. So the calculation of the partial now involves only one data point. This changes how fast we can update the gradient;

Compute contribution to the gradient of one data point	# of data points	Time to compute one step of the gradient	Time to compute on step of Stochastic gradient
1 millisecond	1000	1 second	1 millisecond
1 second	1000	16.7 minutes	1 second
1 millisecond	10 million	2.8 hours	1 millisecond
1 millisecond	10 billion	115.7 days	1 millisecond

Of course, convergence will now requires us to calculate many more gradients, but each is much cheaper.

Comparing gradient to stochastic gradient

		Total time to convergence for large data		
Algorithm	Time per iteration	In theory	In practice	Sensitivity to Parameters
Gradient	slow for large data	slower	often slower	moderate
Stochastic Gradient	always fast	faster	often faster	very high

Stochastic Gradient Ascent conversion much faster than normal Gradient Ascent, but it does not converge smoothly; there will be considerable oscillation around the optimum. Given enough iterations, Gradient Ascent will converge as well.

The lack of smooth convergence presents a practical problem for Stochastic Gradient Ascent. However, this allows us to scale to billions of data points. Because all data does not need to be in memory in order to update the gradient (only a single row of data needs to be in memory), even a desktop computer can handle large data sets (larger than the amount of ram available).

Understanding why stochastic gradient works

Why would stochastic gradient ever work?

Let's start with thinking about how normal Gradient Ascent works. The moving along the gradient gives us the 'best', most direct way to the maximum that we are searching for. The gradient is the direction of 'steepest' ascent. When we move along the gradient, we start at coefficients $w^{(T)}$ and move to coefficients $w^{(T+1)}$ such that, for the log likelihood function;

$$l\left(w^{(T+1)}\right) > l\left(w^{(T)}\right)$$

So by moving along the gradient we have improved our likelihood. Moreover, by moving along the gradient, we have improved the likelihood optimally – we have chosen the best path up the hill. The gradient direction is the sum of the contributions from all of our data points;

$$\frac{\partial l(w)}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial l_i(w)}{\partial w_j}$$

However, moving directly along the gradient is not the only path that improves the likelihood. We can move along any number of non-optimal paths but still improve the likelihood. In effect, we are taking a less direct path up the hill, but we are still going up;

$$l(w') > l(w^{(T)})$$

This is what we are doing in Stochastic Gradient Ascent, we are choosing a lot o non-optimal paths, but in general this will improve our likelihood, so long as we choose enough paths, because we know that the sum of all these paths leads us to the maximum.

$$\frac{\partial l(w)}{\partial w_j} \approx \frac{\partial l_i(w)}{\partial w_j}$$

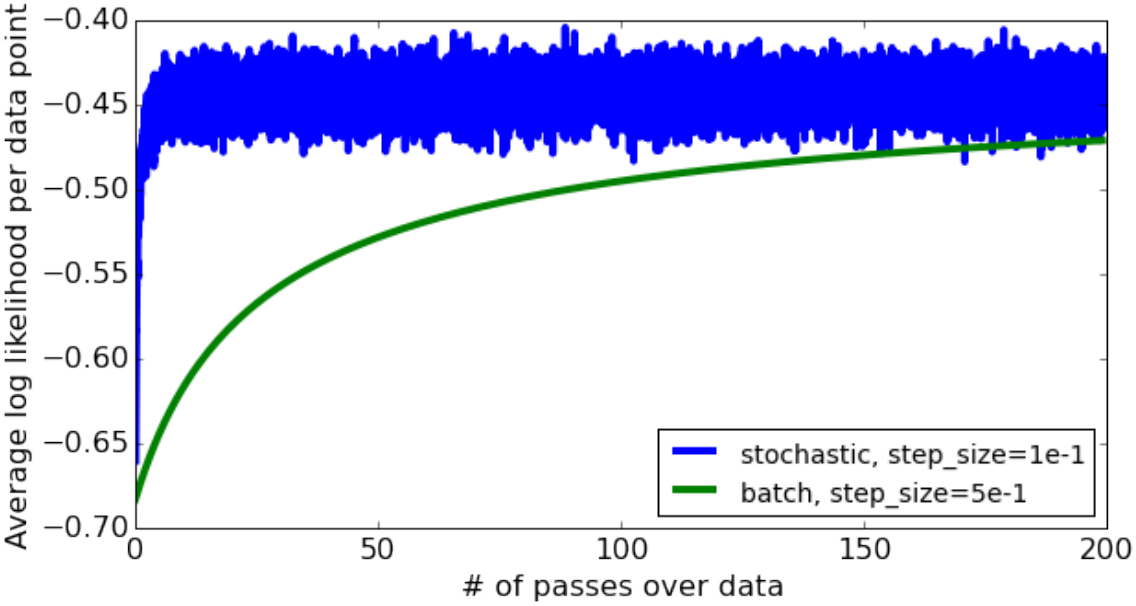
Even though some paths may not go up, on average we will go up because most paths go up. So Stochastic Gradient Ascent chooses one point and moves in that direction, which is usually up the hill, but not always. But by choosing enough puts, we take a meandering walk up the hill.

Convergence paths

Normal Gradient Ascent converges smoothly because it is taking the optimal path (the path of steepest ascent). Stochastic Gradient Ascent converges in a noisy fashion because it takes non-optimal steps, some of which may even go in the wrong direction, but which do ascend on average.

Stochastic Gradient Ascent can converge on the maximum much more quickly than normal Gradient Ascent because it updates that gradient on each point, rather than needing all points.

Both of these aspects of convergence can be seen in the plot below. (Note – the batch like uses a batch size of B==N, so it is a normal gradient ascent).



In the example above, after 20 passes over the data, the stochastic gradient ascent is at or near convergence. This is because it is updating the gradient for each data point, while normal gradient ascent only updates after a full pass over the data.

$$\text{number\_of\_gradient\_updates} = (\text{number\_of\_passes} * \text{N}) \ / \ \text{B}$$

Where B is the batch size.

So if we have 50,000 data points and 20 passes over the data set

normal gradient ascent (B==N),
number_gradient_updates = 50000 * 20 / 50000 = 20 gradient updates

Stochastic gradient ascent (B==1),
number_gradient_updates = 50000 * 20 / 1 = 1,000,000 gradient updates

Stochastic gradient - Practical tricks

Shuffle data before running stochastic gradient

The Stochastic Gradient algorithm uses each row of data in turn to update the gradient. We are hoping that on average we will ascent up the hill towards the maximum. However, if our data has some sort of implicit ordering, this will negatively affect the algorithm. At an extreme, what if we had the data sorted so that all positive reviews came before negative reviews? In that case, even if most reviews are negative, we might converge on an answer of +1 because we never get to see the other data. This can happen with any feature column – if users are sorted by country, or last name or age, then this will affect the outcome.

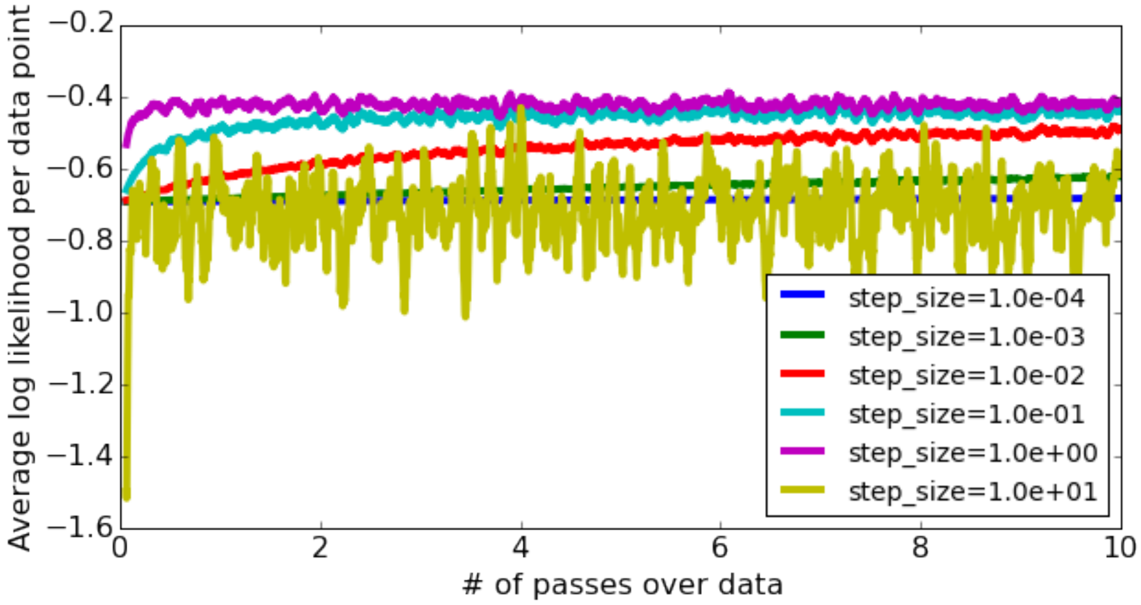
To avoid this, we always shuffle the data before we begin so the rows are in random order.

```
Stochastic Gradient Ascent for Logistic Regression

**Shuffle the data rows**
At t=1, initialize $\widehat{w}^{(1)} = 0$, or some other smart choice.
while $\|\nabla l(\widehat{w}^{(t)})\| > \varepsilon$ do
  for i = 1 to N
    for j = 0..D
      partial[j]≈ $\frac{\partial l_i(w)}{\partial w_j} = h_j(x_i)\big(1[y_i = +1] - P(y = +1|x_i, w^{(t)})\big)$
      $\widehat{w}^{(t+1)} = \widehat{w}^{(t)} + \eta \times partial[j]$
  t = t + 1
```

Choosing step size

Stochastic Gradient is very sensitive to the step size η. If η is too small, convergence will be very slow. If η is too large, then oscillations became very large and it may not converge to the best answer. This can be seen in the plot below. Very small step size (1x10^{-4}) does not converge. Neither does the very large step size (1x10^{1}). The plot shows that a step size of 1x10^{0} (or 1) performs the best – it converges and has the highest average log likelihood at all points on the graph.



- Picking η requires a lot of trial and error – much more than with normal gradient ascent.
- Try several values exponentially spaced.
    - Plot learning curves to find
        - an η that is too large
        - an η that is too small
- Advanced: Use a step size that decreases with each iteration t. This is very important for stochastic gradient;

$$\eta_t = \frac{\eta_0}{t}$$

Don't trust last coefficients

Stochastic Gradient, even when it is converges, oscillates noisily around the answer. So any single point can be higher or lower. Our last iteration is not 'the answer' like it is in normal gradient ascent. Instead, we use an average of the gradient steps – all the gradient steps T, as our final predicted coefficients;

$$\hat{w} = \frac{1}{T}\sum_{t=1}^{T} w^{(t)}$$

So our final predicted coefficients are the average of all of our predictions.

TODO: add optional lessons 4,5,6

Online machine learning - Fitting models from streaming data

The online machine learning task

Batch Learning

Up until now, we have been doing batch learning. We have a large dataset, and we learn a model from it, then use that model to make predictions.

Online Learning

In many applications, data points arrive over time. We must train the model as the data arrives so that our subsequent predictions use the latest iteration of the model coefficients. For example, and ad network want to know what is the best ad to show you. Each time a page is shown to a user, the ad network serves ads. If the user clicks on one of the ads, the ad server is notified and it can now label that ad as positive – the user likes that ad. It can label the ad and relearn a new set of coefficients. If the ad server can update the model very quickly, then it may be able to use the new model on the user's next page view.

Using stochastic gradient for online learning

Stochastic gradient is well suited to the online learning task.

- It only needs to keep track of the number of iterations and the last average set of coefficients – it does not need to keep all the data.
- This allows updates to be very fast.

However, because stochastic gradient oscillates so much, it may not be trusted in all situations. Most companies will actually run lots of small batch learning jobs using fresh data in order to update their models.

Summarizing scaling to huge datasets and online learning
An area for scaling to huge data sets that we have not talked about is parallelism;

- Multicore processors – requires parallel machine learning algorithms
- Cluster computing – requires distributed machine learning algorithms