Optimization Methods

Optimization - Problem Motivation

- General case: find minimum or maximum of a function
- **Example 1**: find parameters of model to maximize likelihood of observed data
- **Example 2**: minimize squared distance between predicted and actual values

Cost Functions

- Many machine learning methods rely on optimizing a "cost function," for example:
  - Linear Regression: $J(\theta) = \sum_{i=1}^{n} (h_\theta(x_i) - x_i)^2$
  - Logistic Regression: $\ln p(\vec{y}|X;\theta) = \sum_{i=1}^{n} (y_i \ln h_\theta(x_i) + (1 - y_i) \ln(1 - h_\theta(x_i)))$
- Cost functions are the mathematical definition of your machine learning goal

Finding an Optimum

- Method from calculus: find where derivative equals zero
- This can be done numerically rather than analytically

The Gradient

- The gradient is the direction of steepest ascent:

$$\nabla f = \frac{\partial f}{\partial x}\hat{i} + \frac{\partial f}{\partial y}\hat{j} + \frac{\partial f}{\partial z}\hat{k}$$

Gradient Descent

- Popular method for optimizing cost functions
- Follows line of steepest descent on cost surface to find minimum

Gradient Descent Algorithm

- Summary: repeatedly take steps down the gradient until the cost function converges

Gradient Descent Algorithm

Mathematical Description:

Repeat until convergence:

$$\theta_{i+1,j} = \theta_{i,j} - \alpha \frac{\partial}{\partial \theta_{i,j}} J\left(\vec{\theta_i}\right)$$

α=learning rate, i=iteration, j=feature

Gradient Descent Algorithm

(Pseudo)Python:

```
new_params = dict((i, 0) for i in xrange(k)) # initialize k parameters
while not has_converged:
    params = copy(new_params)
    for theta in params:
        new_params[theta] -= learning_rate*gradient(theta, params)
```

Note that the parameters are updated simultaneously!

Think about how this could be written in numpy without loops!

Gradient Descent Convergence

Choices of convergence criteria:

- max number of iterations
- change in cost function

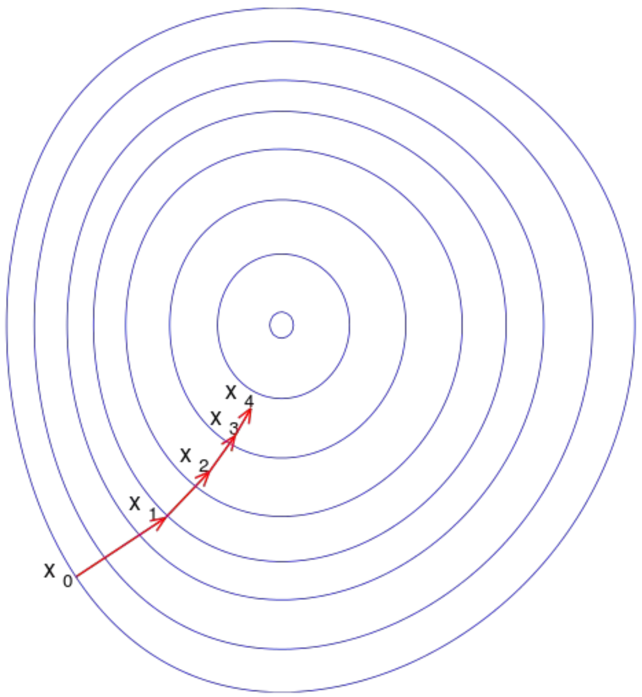
  $$\left(cost_{new} - cost_{old}\right)/cost_{old} < \epsilon$$

- magnitude of gradient

  $$|\nabla f| < \epsilon$$

Gradient Descent - 3D Graphical Example

See IPython Notebook



Properties of Gradient Descent

- Requires differentiable and convex cost function
- Only finds global optimum on globally convex functions
- Convergence asymptotically

Stochastic Gradient Descent (SGD)

- What if your data is too big to fit in memory?
- What if computation time is very limited?
- What if you're continually getting more data?
- Use SGD!

SGD Algorithm

- SGD computes cost function using only one randomly chosen observation at a time
- Otherwise the same algorithm as gradient descent
- Ex. linear regression:

Instead of this:

$$J(\theta) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

Use this:

$$J(\theta) = (y_i - h_\theta(x_i))^2$$

Variants of Gradient Descent/SGD

- “Batch” is another name for plain vanilla GD
- “Online” SGD uses each observation as it’s collected
  - Ex. every time a new transaction occurs, update your fraud model with that transaction
- “Minibatch” SGD uses random subset of data
  - If the entire dataset doesn’t fit in memory, train on random subset in each iteration

Properties of SGD

- Converges faster on average than batch GD
- Can oscillate around optimum

Which Variant to Use?

- In practice, SGD is often preferred because it requires less memory and computation

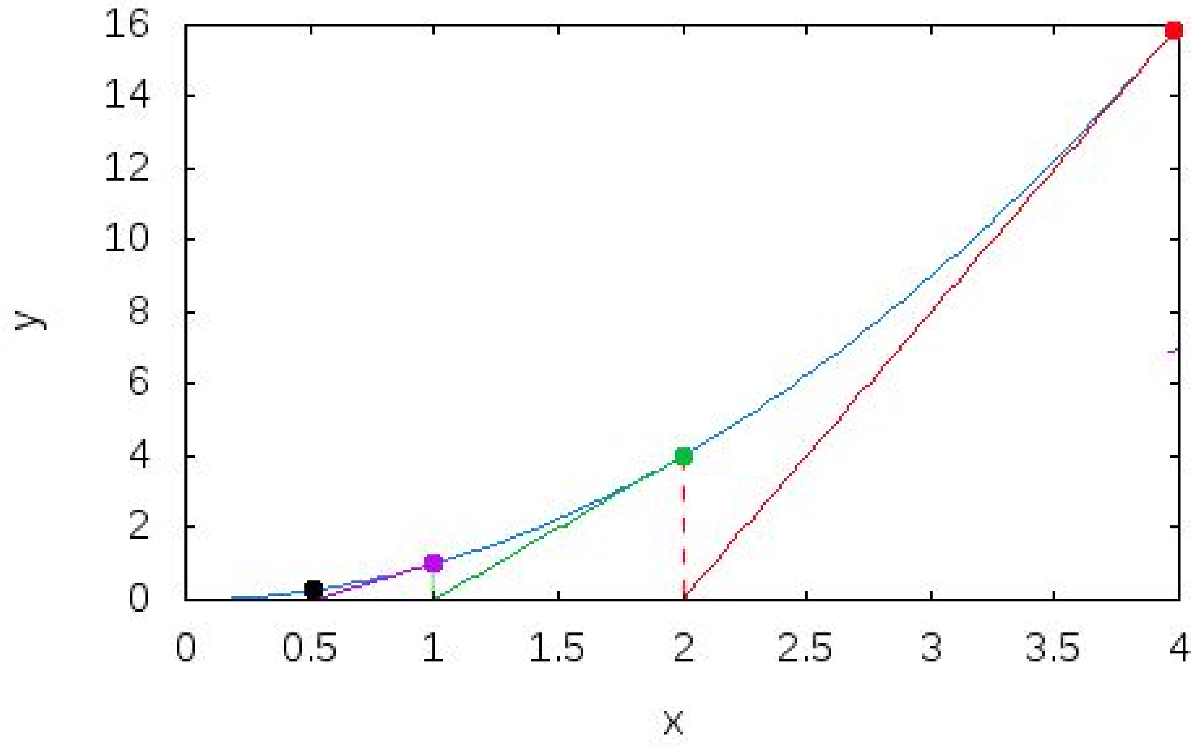
See papers:

- *"Large-Scale Machine Learning with Stochastic Gradient Descent"* *http://leon.bottou.org/publications/pdf/compstat-2010.pdf*
- *"The general inefficiency of batch training for gradient descent learning"* *http://axon.cs.byu.edu/papers/Wilson.nn03.batch.pdf*

Newton-Raphson Method

- Optimization technique similar to gradient descent
- Root-finding method applied to cost function’s first derivative
- Sometimes just called “Newtons Method”

Newtons Method - Graphical Example



Newton-Raphson Method

Mathematical Description:

$$\text{while } J'(x) > \text{threshold:}$$

$$\theta_{i+1} = \theta_i - \frac{J'(\theta_i)}{J''(\theta_i)}$$

Python:

```
while f_prime(x) > threshold and iterations < max_iter:
    x = x - f_prime(x)/f_double_prime(x)
```

Comparison with Gradient Descent

Gradient Descent:

$$\theta_{i+1} = \theta_i - \alpha J'(\theta_i)$$

Newton-Raphson:

$$\theta_{i+1} = \theta_i - \frac{J'(\theta_i)}{J''(\theta_i)}$$

Newton-Raphson in Higher Dimensions

$$\mathbf{x}_{n+1} = \mathbf{x}_n - [Hf(\mathbf{x}_n)]^{-1} \nabla f(\mathbf{x}_n)$$

Newtons Method vs Gradient Descent

- When Newton's Method works, it often takes many fewer iterations by accounting for 2nd order information
- Inverting the Hessian matrix can be computationally costly or impossible if the matrix is singular
- Newton can diverge with bad initial guess
- Key takeaway: there is no universally best optimization method