Gradient Descent

Road Map

- Morning
  - Optimization problems.
  - Cost functions for goodness of fit.
  - Gradient descent.
- Afternoon
  - Stochastic gradient descent.
  - Demo.

Optimization - Problem Motivation

- What are some common examples of optimization problems?

Optimization - Problem Motivation

- What are we optimizing for in data science?
  - Models for predicting some output.
  - We would like to predict test data correctly.
  - Models have parameters.
  - Choose a measure of closeness(goodness of fit).
  - Choose parameters to optimize closeness.

Cost Functions

- Examples:
  - Linear Regression: RSS

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

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

  - Logistic Regression: Log-likelihood

$$J(\theta) = \frac{1}{n}\ln p(\vec{y}|X;\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i \ln h_\theta(x_i) + (1 - y_i)\ln(1 - h_\theta(x_i)))$$

Gradient Descent

Road Map

- Morning
  - Optimization problems.
  - Cost functions for goodness of fit.
  - Gradient descent.
    - Intuition.
    - Mathematical definition.
    - Examples.
    - Pseudocode.
    - Convergence criteria.
    - Cases when gradient descent works.

Gradient Descent

- Explore a neighborhood of parameters.
- Go in the direction of steepest descent.

Gradient Descent

- In one dimension: $J(\beta) = \beta^2$
  - Calculate the direction of steepest descent:
    $$\frac{dJ}{d\beta}$$
  - Choose a learning rate: ϵ
  - Repeatedly update parameters:
    $$\beta^{t+1} = \beta^t - \epsilon \frac{dJ}{d\beta}(\beta^t)$$
  - What happens when the step size is too large?

Gradient Descent

- In n-dimensions
  - The gradient is the direction of the steepest ascent:

$$\nabla J = \frac{\partial J}{\partial \beta_1}\hat{e}_1 + \frac{\partial J}{\partial \beta_2}\hat{e}_2 + \cdots + \frac{\partial J}{\partial \beta_n}\hat{e}_n$$

  - Choose learning rate.
  - Take a step:

$$\vec{\beta}^{t+1} = \vec{\beta}^{t} - \epsilon \nabla J$$

  - Repeat.

Recap

- Optimization problems are everywhere.
- Cost functions:
  - e.g. goodness of fit measures.
- Gradient descent:
  - Go in the direction of better fit.
- When do you stop?

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) #train on old parameters not old ones.
    for beta in params: #for each component update.
        new_params[beta] -= learning_rate*gradient(beta, params)
```

Note that the parameters are updated on the previous iterations gradient!

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

Convergence Criteria

- When a set number of iterations is done.
  (May not have converged.)
- When the percent change is small enough:

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

- When the cost function is flat enough:

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

When to use Gradient Descent

- When cost function are differentiable.
- When there is only one global optimum.
- Global optimum is guaranteed when the cost function is globally convex.
- When features have similar scales.
- When asymptotic answer is acceptable.

Summary

- Morning
  - Optimization problems.
  - Cost functions for goodness of fit.
  - Gradient descent.
    - Intuition.
    - Mathematical definition.
    - Examples.
    - Pseudocode.
    - Convergence criteria.
    - Cases when gradient descent works.

Stochastic Gradient Descent

Road Map

- Afternoon
  - When is gradient descent bad?
  - Stochastic gradient descent(SGD).
    - Intuition.
    - Expected cost and gradient.
    - The algorithm.
    - Convergence criteria.
    - Variants of stochastic gradient descent(SGD).
    - Demo

When is doing gradient descent bad?

- Memory constraints:
  - Data doesn’t fit in memory.
- Takes long time to compute cost function over many rows.
- “Online” setting: data keeps coming in

Solution

- Train on subsets of your data!

SGD Algorithm

- Sample a data point without replacement.
- For each data point, do a step of gradient descent:

$$\beta \to \beta - \epsilon \nabla J_i(\beta)$$

Is it going to work?

- Why should/shouldn’t we expect stochastic gradient descent to work?

Expected Direction is Correct

- Cost function is expected cost per observation:

$$J(\beta) = E[J_i(\beta)] = \sum_{i=1}^{n} \frac{1}{n} J_i(\beta)$$

- The gradient and expected gradient are also the same.

$$\nabla J(\beta) = E[\nabla J_i(\beta)] = \sum_{i=1}^{n} \frac{1}{n} \nabla J_i(\beta)$$

Gradient Descent Algorithm

(Pseudo)Python:

```
new_params = dict((i, 0) for i in xrange(k)) # initialize k parameters
while not has_converged:
    for i in shuffled_data:
        params = copy(new_params) #train on old parameters not new ones.
        for theta in params: #for each component update.
            new_params[theta] -= learning_rate*gradient(i,theta, params)
```

Stopping Criteria

- Can’t just wait until a random jump is flat or doesn’t improve the cost.
- You may want to take a moving average of these criteria. $T_{old} \to pT_{current} + (1-p)T_{old}$
- You may also cutoff iterations.

Pros and Cons of SGD

- Converges faster on average than batch GD
- Can optimize over a changing cost function (e.g. online setting)
- Can oscillate around optimum.

Variants of Gradient Descent/SGD

- “Online” SGD uses each observation as it’s collected
  - Ex. every time a new transaction occurs, update your fraud model with that transaction
  - Can optionally discard old observations

Variants of Gradient Descent/SGD

- “Batch” is another name for plain vanilla GD

- “Minibatch” SGD uses random subset of data
  - If the entire dataset doesn’t fit in memory, train on random subset in each iteration.

  - This is like the sample average of the gradient.

Which Variant to Use?

- In practice, SGD is often preferred because it requires less memory and computation.
- But small batches may reduce the variance of your steps.

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

Demo

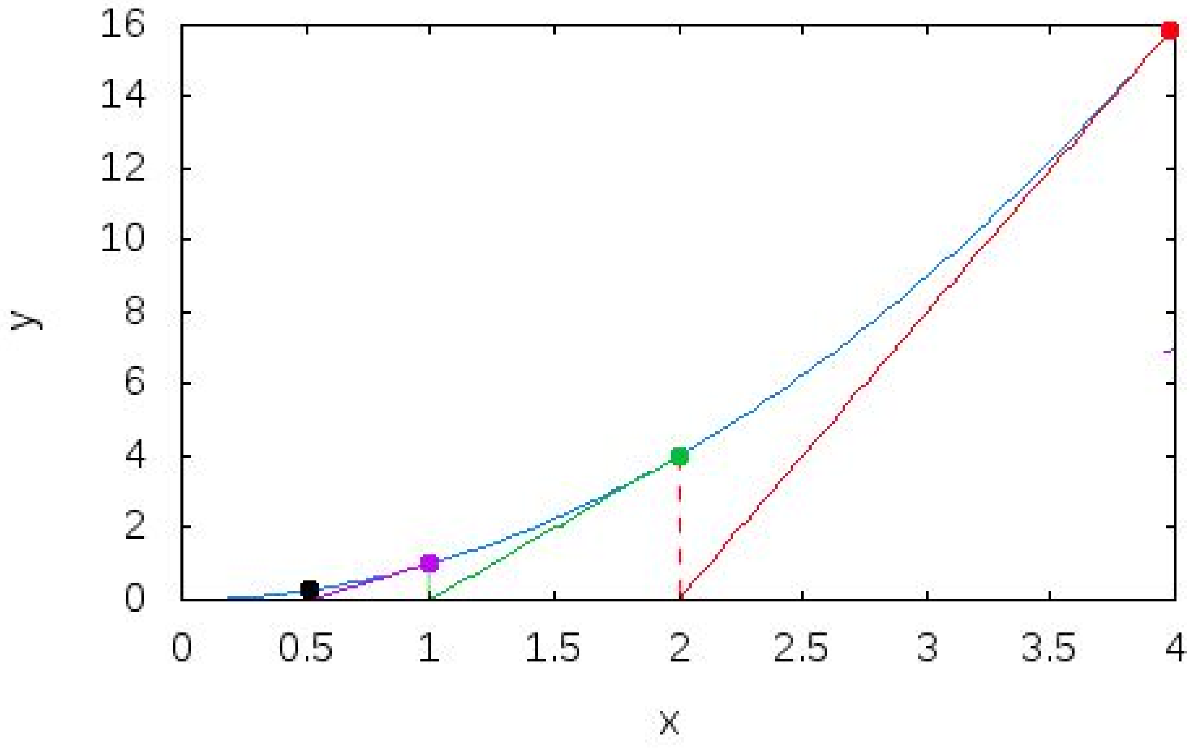
Summary

- Afternoon
  - When is gradient descent bad?
  - Stochastic gradient descent(SGD).
    - Intuition.
    - Expected cost and gradient.
    - The algorithm.
    - Convergence criteria.
    - Variants of stochastic gradient descent(SGD).
    - Demo

Newton-Raphson Method

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

Newton’s 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:

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

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