Advanced Optimization

We have an alternative view of what gradient descent is doing. We have some cost function J and we want to minimize it.

Cost function $J(\theta)$. Want $min_\theta J(\theta)$

Give θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$

Gradient Descent:

Repeat $\{$

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

$\}$

One algorithm we can use to compute the cost function and the derivative partial of the cost function is gradient descent. But gradient descent isn't the only algorithm we can use. And there are other algorithms, more advanced, more sophisticated ones, that, if we only provide them a way to compute the cost function and the derivative partial of the cost function, then these are different approaches to optimize the cost function for us.

Optimization algorithms:

- Gradient Descent
- Conjugate gradient
- BFGS (Broyden-Fletcher-Goldfarb-Shanno)
- L-BFGS (Limited memory - BFGS)

Conjugate gradient BFGS and L-BFGS are examples of more sophisticated optimization algorithms that need a way to compute $J(\theta)$, and need a way to compute the derivatives of $J(\theta)$, and can then use more sophisticated strategies than gradient descent to minimize the cost function.

The three new optimization algorithms have some properties:

Advantages:

- No need to manually pick α
- Often faster than gradient descent.
- Can be used successfully without understanding their complexity

Disvantages

- More complex

We can think of these algorithms as having a **clever inter-loop**. And, in fact, they have a clever inter-loop called a line search algorithm that automatically tries out different values for the learning rate alpha and **automatically picks a good learning rate alpha** so that it can even pick a different learning rate for every iteration. And so then we don't need to choose it ourself. These algorithms actually do more sophisticated things than just pick a good learning rate, and so they often end up converging much faster than gradient descent.

Video Question: Suppose you want to use an advanced optimization algorithm to minimize the cost function for logistic regression with parameters θ_0 and θ_1. You write the following code:

```
function [jVal, gradient] = costFunction(theta)
  jVal = % code to compute J(theta)
  gradient(1) = CODE#1 % derivative for theta_0
  gradient(2) = CODE#2 % derivative for theta_1
```

What should CODE#1 and CODE#2 above compute?

- CODE#1 and CODE#2 should compute $J(\theta)$.
- CODE#1 should be theta(1) and CODE#2 should be theta(2).

  > CODE#1 should compute $\frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^{(i)})-y^{(i)})\cdot x_0^{(i)}](=\frac{\partial}{\partial\theta_0}J(\theta))$, and CODE#2 should compute $\frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^{(i)})-y^{(i)})\cdot x_1^{(i)}](=\frac{\partial}{\partial\theta_1}J(\theta))$.

- None of the above.

Summary

"Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent. We suggest that you should not write these more sophisticated algorithms yourself (unless you are an expert in numerical computing) but use the libraries instead, as they're already tested and highly optimized. Octave provides them.

We first need to provide a function that evaluates the following two functions for a given input value θ:

$J(\theta)$

$\frac{\partial}{\partial\theta_j}J(\theta)$

We can write a single function that returns both of these:

```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()"

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, opt
ions);
```

We give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.