Gradient Descent Notes – Aron Culotta

1 Gradient Descent Recipe

One pervasive form of machine learning uses the following recipe:

1. Select a model type (e.g., linear classification, logistic classification)
2. Select an **error function** that, when minimized, results in a good setting of the model parameters.
3. Analytically determine the gradient of the error function with respect to the model parameters.
4. Iteratively change the parameters by a small amount in the direction of the gradient until the (near) minimum of the error function is found.

2 Example: Linear Regression of One Variable

Consider the model $y = mx + b$. This is a simple linear equation mapping x to y. In terms of the hypothesis notation, we would write $h(x) = mx + b$.

Suppose you are presented with n of **training examples** of $< x, y >$ pairs (which we call D): $D = \{< x_1, y_1 > \ldots < x_n, y_n >\}$

The learning problem is to pick m and b that fit D the best. To determine what "best" is, we define an error function called **residual sum of squares**:

$$RSS(h, D) = \frac{1}{2}\sum_{i=1}^{|D|}(y_i - (mx_i + b))^2$$

This is the sum of the squared distances between the model's prediction ($mx_i + b$) and the truth (y_i). This is a somewhat arbitrary function that hopefully makes intuitive sense. Another important reason it is chosen is because it is easy to analytically determine its gradient.

We have now done the first two steps of the recipe. The third step requires a little bit of calculus (so close your eyes and skip to the update rules below if this is a problem):

$$\begin{aligned}
\frac{\partial RSS}{\partial m} &= \frac{1}{2}\frac{\partial}{\partial m}\sum_{i=1}^{|D|}(y_i - (mx_i + b))^2 \\
&= \frac{1}{2}\sum_{i=1}^{|D|}2(y_i - (mx_i + b))\frac{\partial}{\partial m}(y_i - (mx_i + b)) \\
&= \sum_{i=1}^{|D|}(y_i - (mx_i + b))(-x_i)
\end{aligned}$$

Recomputing the gradient with respect to b yields:

$$\frac{\partial RSS}{\partial b} = \sum_{i=1}^{|D|} -(y_i - (mx_i + b))$$

You'll notice the negative signs in front of each derivative. In fact, the gradient points "up", so to reduce the error function, we must update in the inverse direction. Thus, the final update formulae, assuming a

step-size of η, are:

$$\begin{aligned} m^{t+1} &= m^t + \eta \sum_{i=1}^{|D|} (y_i - (m^t x_i + b^t)) x_i \\ b^{t+1} &= b^t + \eta \sum_{i=1}^{|D|} (y_i - (m^t x_i + b^t)) \end{aligned}$$

where m^t is the m parameter at iteration t of gradient descent (similarly for b^t).

Thus, the pseudo-code for doing gradient descent here is:

1. Initialize m and b to random small numbers
2. Select η (e.g., 0.1)
3. for iter=0; iter < numIterations; iter++
   (a) Update: $m^{t+1} = m^t + \eta \sum_{i=1}^{|D|} (y_i - (m^t x_i + b^t)) x_i$
   (b) Update: $b^{t+1} = b^t + \eta \sum_{i=1}^{|D|} (y_i - (m^t x_i + b^t))$
   (c) If $RSS(h, D)$ has not changed by some threshold, exit.

As we discussed in class, when RSS is nearly minimized, the update may "jump over" the minimum. Once simple way to fix this is to modify the pseudo-code above as follows:

- Before updating m and b, compute the RSS that you would get if you were to update them.
- If this RSS is larger than the present RSS, reduce the step size (e.g., by multiplying η by 0.9).
- Recompute the updates and repeat.

3 Example: Linear Regression of Multiple Variables

Clearly, a model with two parameters is limited, if for no other reason than each instance can only be described by one variable (e.g., hair color). Many real-world problems have millions of variables.

The recipe is exactly the same for multiple variable regression, but we'll have to change the notation a bit. Rather than having two parameters (m, b), we'll have a vector of k parameters $\theta = \{\theta_1 \dots \theta_k\}$. Our model to classify an instance x_i is now:

$$h(x_i) = \sum_{j=1}^{k} x_{ij} \theta_j$$

where x_{ij} is the value of feature j for instance i (e.g., "Julie has blonde hair").

That sum of products between θ and x is a dot product. We can thus write the model more concisely using dot product notation:

$$h(x_i) = \theta \cdot x_i$$

Thus, the RSS becomes:

$$RSS(h, D) = \frac{1}{2} \sum_{i=1}^{|D|} (y_i - \theta \cdot x_i)^2$$

The gradient we have to compute is with respect to one of the θ variables:

$$\begin{aligned}
\frac{\partial RSS(h,D)}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \sum_{i=1}^{|D|} (y_i - \theta \cdot x_i)^2 \\
&= \sum_{i=1}^{|D|} (y_i - \theta \cdot x_i) \frac{\partial}{\partial \theta_j} (y_i - \theta \cdot x_i) \\
&= \sum_{i=1}^{|D|} (y_i - \theta \cdot x_i)(-x_{ij})
\end{aligned}$$

Notice that in the final step, the only element of θ that contributes to the gradient with respect to θ_j is θ_j (the other variables are "constants" for the purposes of computing this gradient).

This gradient should look eerily similar to the one we compute for the simple linear model of one variable. The update is then

$$\theta_j^{t+1} = \theta_j^t + \eta \sum_{i=1}^{|D|} (y_i - \theta \cdot x_i) x_{ij}$$

Examining the update rule for θ_j shows something that should make some intuitive sense: The contribution that example x_i makes to the update is a function of (1) the error the model makes on instance x_i, (2) the feature value x_{ij}. Thus, if a large error is made on x_i, a large update is required. Also, if x_{ij} is very small, it presumably did not contribute much to the mistake the model made, so its update is small.

What happened to b?

You may have noticed that the intercept term b disappeared when we moved from single to multiple variables. This is mainly for convienence. What happens in practice is that a "dummy variable" is added to all instances and set to the value one (e.g., hair=blonde, eyes=blue, dummy variable=1). Thus, the intercept just becomes another element of θ, so we don't need to compute a separate gradient, etc.