Gradient Descent

Optimization

This problem comes up very often in real world application since for most problem we can rephrase most problem in to optimization problem. For example, what's an optimal amount of sugar, flour, butter, egg to make a cake? You can think about the amount of ingredient as the variable($x, y, z, \ldots$) and the deliciousness of cake as the cost function f. You may be able to measure it by having 1000 people taste it and rate it. You want to adjust all the ingredients such that f is maximized. It may not be 100% but we just want to maximize it.[1]

Moreover, since each test takes a long time and cost a lot, you need a long time to bake and you need to spend money on the raw materials. You want to maximize it in a few iterations.

Minimize VS Maximize

Before we go on and get more technical. It should be noted that minimization and maximization is essentially the same problem. For example, the problem of maximize the cake rating(f) by adjusting all the ingredients is the same problem as minimize the negative of the cake rating($-f$). So, we will only talk about minimization since turning a maximization problem into a minimization problem is trivial. Just add a minus sign.

For example, the problem of finding x that maximize

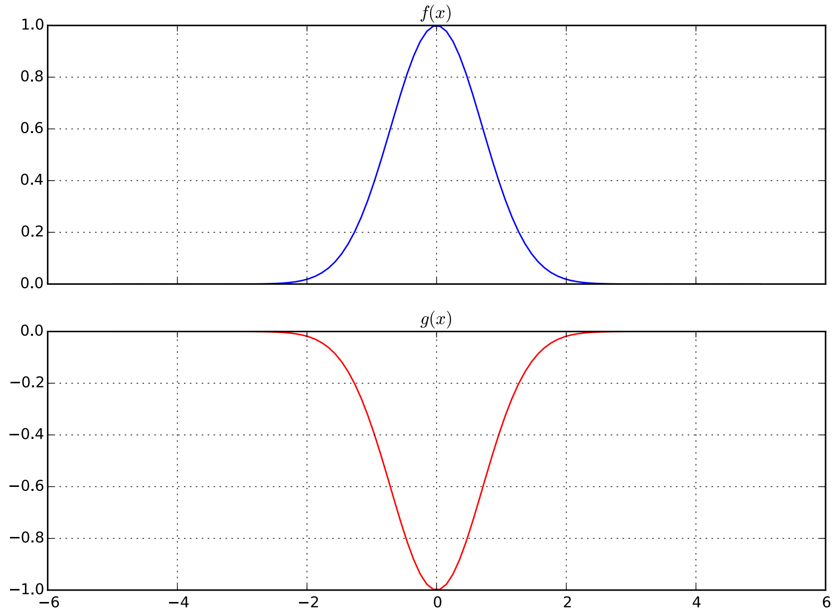
$$f(x) = e^{-x^2} \tag{1}$$

is the same thing as finding x that minimize

$$g(x) = -f(x) = -e^{-x^2} \tag{2}$$

The figure below shows the two function side by side. The location of the maximum of $f(x)$ is the location of the minimum of $g(x)$.

[1] The process of making cake depends on a lot more parameters than this. How we mix and how we bake also make a differnce. Some of these variables are not parametric. Ex: whether to put all the ingredient in a copper bowl or an glass bowl. We will deal with those kind of variables next week.



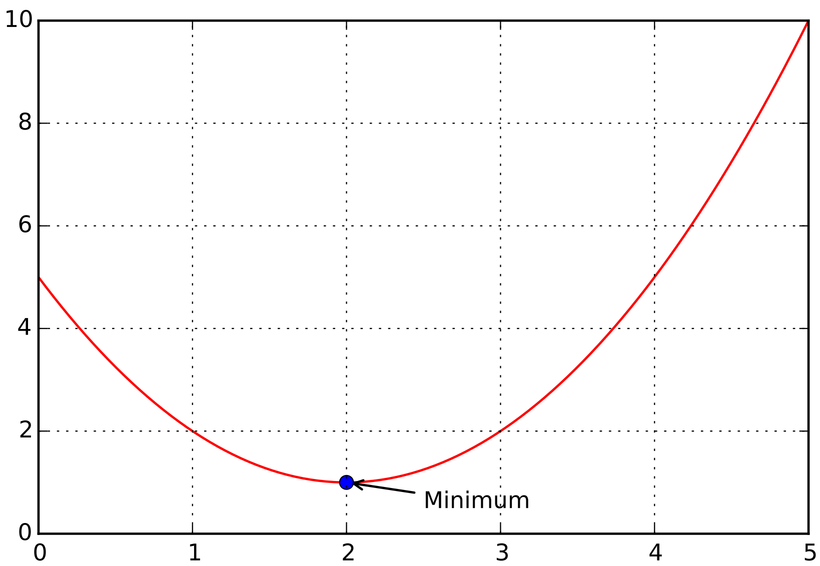
Simple Case

Let us start with 1 Dimension and grasp the concept.

Let us consider a function

$$f(x) = (x-2)^2 + 1$$

The function looks like



This function is easy to do since you can just take derivative and solve the equation. But let us pretend that we can't do that for a moment.

Gradient descent is simple.

1. Begin with a guess.

2. Figure out which direction we should go.

3. Figure out how far we should go to.

4. Go.

5. Repeat.

Direction

Let us apply this with 1 Dimension problem. Suppose that our first guess is at $x = 4$. We want to figure out which way we should go. This can be done by considering the derivative.

At $x = 4$ the derivative is positive. This means that if we walk to the right of $x = 4$. The value of the function will increase(at least locally).

But, since we want to minimize the function that means we should go in the opposite direction of the derivative. Concisely,

$$x_{n+1} = x_n - d\frac{f'(x_n)}{|f'(x_n)|}. \tag{3}$$

The ratio $\dfrac{f'(x)}{|f'(x)|}$ is just an expression that give +1 if $f'(x) > 0$ and -1 if $f'(x) < 0$. That is

$$\frac{f'(x)}{|f'(x)|} = \begin{cases} +1 & \text{if } f'(x) > 0 \\ -1 & \text{if } f'(x) < 0 \end{cases}$$

Furthermore the minus sign serves the purpose of flipping direction.

- If $f'(x)$ is positive, the function is increasing and we know that we should walk to the left hence the minus sign.

- If $f'(x)$ is negative, the function is decreasing and we know that we should walk to the right hence the minus sign.

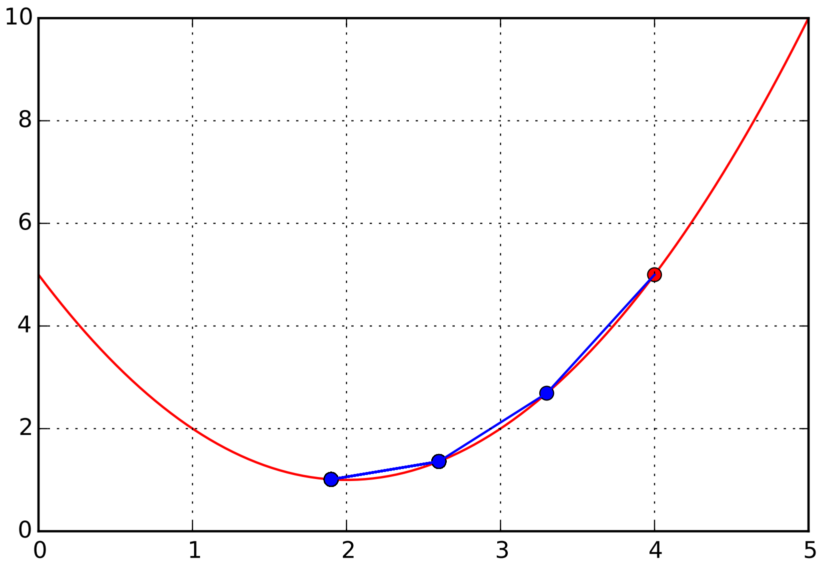
So, now at least we know that our new guess x_{n+1} should be something to the left of the current guess x_n. The question is how much to the left should we walk. This is captured in the parameter d in the Equation 3.

Constant Step

Let us think about the parameter d. Our first attempt is make d a constant. Let's make our $d = 0.17$ and see what happen.

Iteration	x	$f(x)$
0	4	5
1	3.3	2.69
2	2.6	1.36
3	1.9	1.01
4	2.6	1.36
5	1.9	1.01
6	2.6	1.36

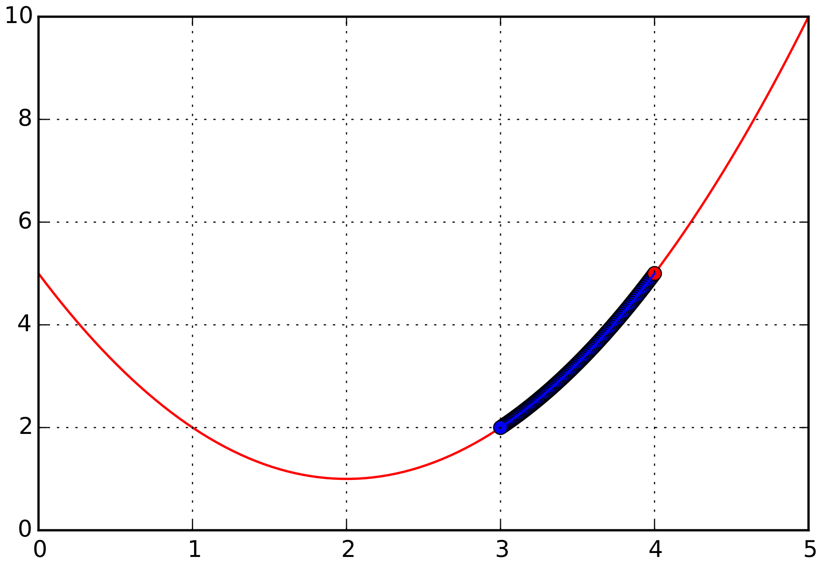
You can see that after a few iterations our x will keep socillating around the minimum. Our move will keep over stepping the true minimum as shown in the figure below.



Dynamic Step Size.

Constant step is clearly not working. But, one may think that since it's osciallting, we just need to make d small. Since the oscillating amplitude is the same as d, we will still keep stepping over the minimum but only by a small amount.

This sounds really good, but there is a little problem. It takes so many step to reach the minimum. We waste so many step when we are far away form the minimum. The figure below shows what happen when $d = 0.01$ after 100 steps. We are not even close to the minimum yet and we are walking very slowly toward the minimum. (We will eventually reach the minimum after about 200 steps)



So, we want to do something that is the mix of the two. We want the followings

1. The step size should be small when we are near the minimum so that we get an accurate answer.
2. The step size should be large when we are far away from the minimum so that we get to the minimum quickly.

This means that we need to detect how far we are from the minimum. One good candidate is the magnitude of the derivative $|f'(x)|$. The magnitude of the derivative is zero when we are at the minimum and some non zero value when we are far from the minimum. Anything proportional to $|f'(x)|$ behaves the way we want. Let us modify Equation using

$$d = \lambda |f'(x)|$$

with $\lambda > 0$.Plugging this in we have

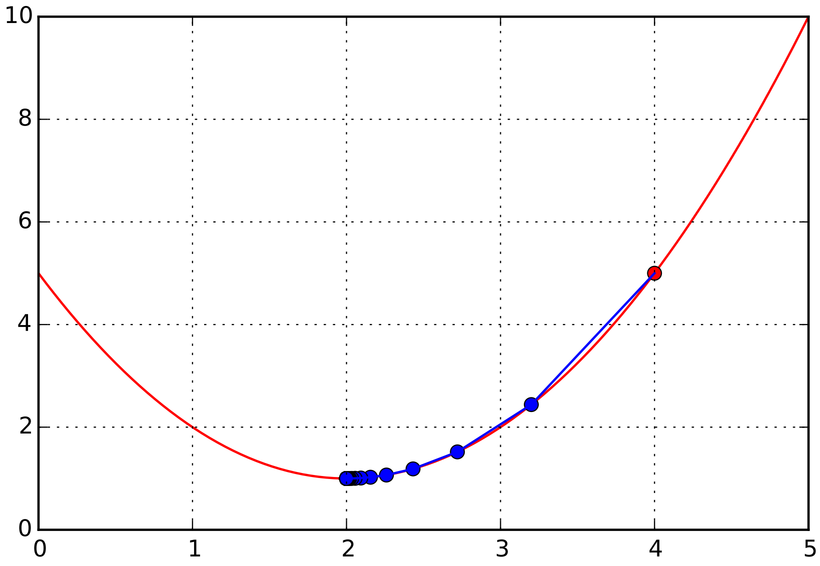
$$x_{n+1} = x_n - \lambda |f'(x)| \times \frac{f'(x)}{|f'(x)|}.$$

Therefore,

$$x_{n+1} = x_n - \lambda f'(x). \tag{4}$$

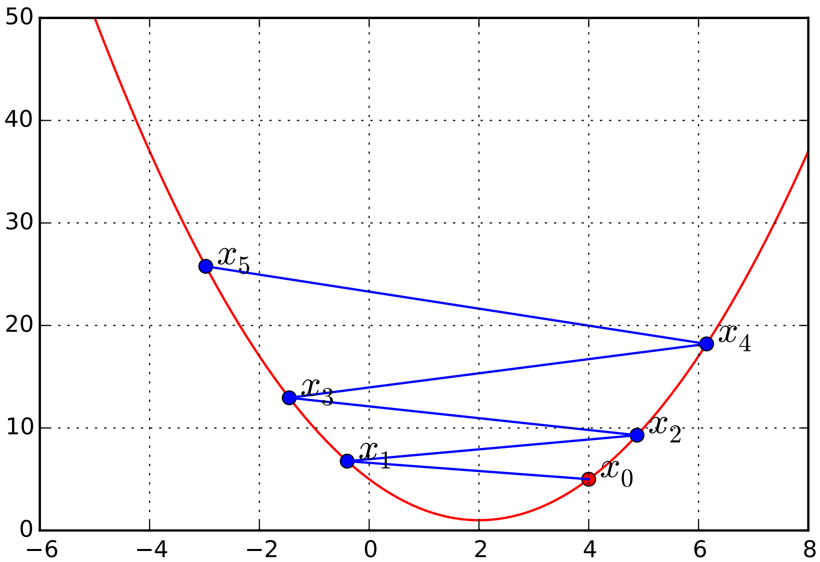
Eventhough there are so many functions that behave the way we want, this one is arguably the best since it has a nice cancellelation. The formula even look nicer than Equation .

The parameter λ is called the *learning rate*. Learning rate is the parameter that dictates how fast we are reaching the minimum. Let us first try with value of $\lambda = 0.2$. This is typically what we do, some small number less than 1.



From the picture above, we can see that we get the expected behavior. We walk fast when we are faraway from the minimum and we walk slowly when we are near the minimum. After abount 50 iterations we get to the minimum with accurcy beyond the machine accuracy.

Let us try the same thing with a big value of $\lambda = 1.1$ for the first few steps.



We can see that this is not going anywhere since we keep over stepping the minimum way too far and the more we do this the worse it becomes. This problem can also be fixed if we allow λ to be the function of iteration number and it gets smaller and smaller. But, usually we are better off by just changing the value of λ.

What we do in real world application is that we play with the value lambda. We typically try with a small value less than 1 like 0.1 or 0.2 first and see if it reaches a sensible value.

Gradient Descent

The method we discuss in the previous section is for 1 dimension only which we have a whole lot of other method which converge much faster. The attractiveness of the method described above comes when we do multidimensional problem.

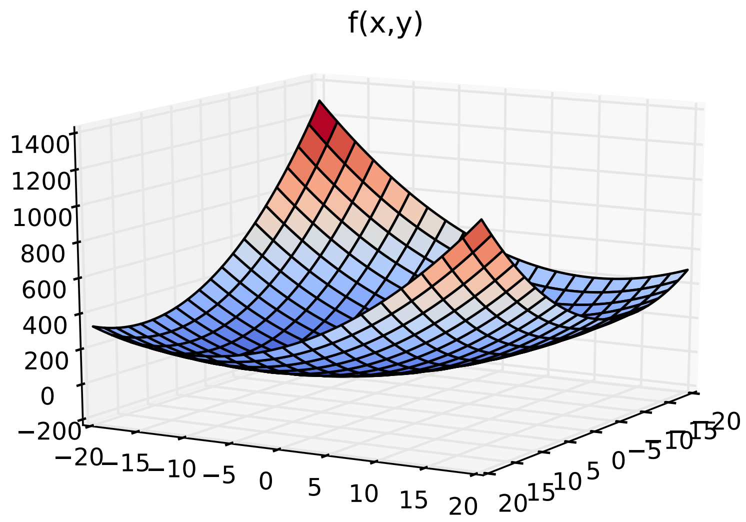
Let us recap the algorithm

1. Begin with a guess.
2. Figure out which direction we should go.
3. Figure out how far we should go to.
4. Go.
5. Repeat.

We just need to do this in multidimension setting. Let us start with 2. The idea is the same. Let us consider the following function [2]

$$f(x, y) = (x-2)^2 + xy + y * 2 + 1 \tag{5}$$

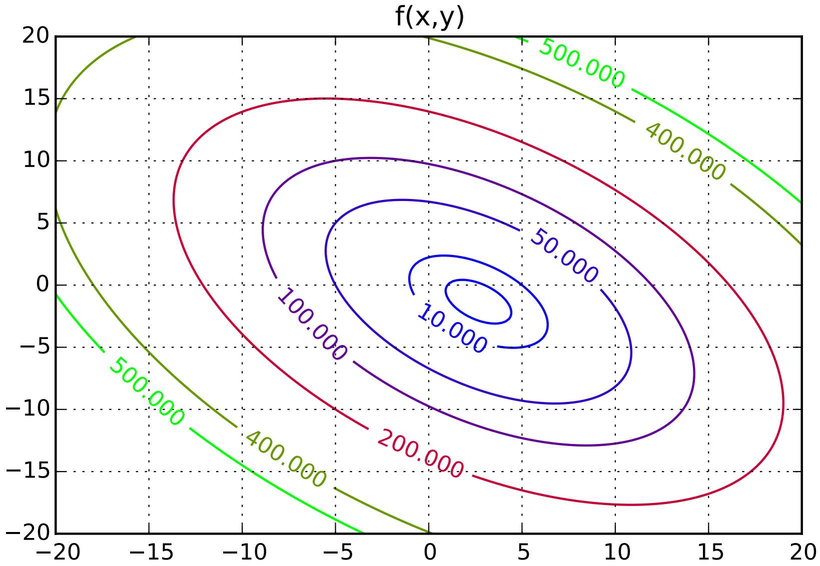
The 3d plot of the function is shown below.



We can see qualitatively that there is a some sort of a minimum in the function. We want to reach the bottom of the function.

Contour plot gives us a more quantative information about the function. This is shown below

[2] The +1 at the end will not change the location of minimum. Sometimes we just get rid it.



We can see that there is a minimum around $(x, y) = 2.5, -2.5$.

Something from Vector Calculus

Recall gradient from multivariable calculus.[3]

$$\nabla f(x,y) = \frac{\partial}{\partial x} f(x,y)\hat{x} + \frac{\partial}{\partial y} f(x,y)\hat{y} \tag{6}$$

$$= f_{,x}(x,y)\hat{x} + f_{,y}(x,y)\hat{y} \tag{7}$$

The very neat property from this gradient is that it is best direction to walk(locally) if we want to get the value of the function to increase the fastet. This property can be understood easily gemetrically.

First, from Taylor expansion we know that the change in function if we move on x direction by Δx and on y direction by Δy is given by

$$\Delta f(x,y) = f_{,x}\Delta x + f_{,y}\Delta y \tag{8}$$

This looks awfully close to the gradient. Recall one more thing from freshmen physics: the dot product. With the dot product, the above expression can be written as.

$$\Delta f(x,y) = (f_{,x}\hat{x} + f_{,y}\hat{y}) \cdot (\Delta x\hat{x} + \Delta y\hat{y}) \tag{9}$$

$$= \nabla f \cdot \Delta\vec{r} \tag{10}$$

We introduce a new variable $\Delta\vec{r}$. Geometrically this is just the vector of our displacement. This is what we get to choose: the direction of our walking.

[3] If we have more variables, we just need to do the sum of all the partial derivative for all variables.

One more thing about the dot product: it depends on the length of and the angle between the two vectors. The dot product in the expression above is maximum when the two align. That means the function will increase the most when our direction of displacement is along the gradient.

On the other hand, if we want the minimum from the dot product. We just need to make the two vectors point in the opposite direction. This means that if we want to mimize the function we should walk in the oppsite direction to the gradient. Thus the unit vector of the direction we should walk is $\frac{\nabla f}{|\nabla f|}$

The update rule for the minimization then reads

$$\vec{x}_{n+1} = \vec{x}_n - d \times \frac{\nabla f}{|\nabla f|}. \tag{11}$$

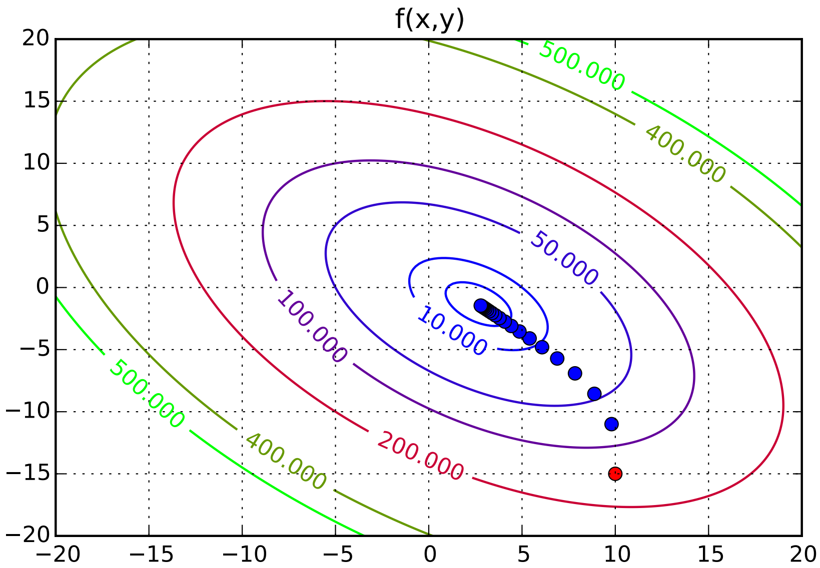
This looks very similar to the update rule for 1 Dimension shown in Equation 3.

We can then use the same idea for dynamic value of d. We need something large when we are faraway from the minimum and we want something small when we are near the minimum. Similar to 1D case, the choice is $d = \lambda |\nabla f|$. Thus the update rule with dynamic stepping is

$$\vec{x}_{n+1} = \vec{x}_n - \lambda \nabla f. \tag{12}$$

Writing this as a vector make it so easy to generalize to higher dimension.

If we apply the update rule, to the cost function shown in Equation 5 starting from $\vec{x}_0 = (10, -15)$. The figure below show the path it takes when $\lambda = 0.2$.



Some Notes regarding Gradient Descent

As we mention before that there are many variants of this algorithm. What we learn in the previous section is just one version of it. There are many more of it you can google for BFGS or DFP method. They get a bit smarter in picking the step size but the idea is very similar to what we discuss above.

Second thing that is important is that this method doesn't guarantee convergence as you have seen from the case where learning rate is large.

What we talked about here is an unconstrained optimization. The problem becomes much harder when we have constrained. This is however a well-studied problem. If you want to learn more search for convex optimization.

Some Application

Minimization is quite straight forward. But the real art lies in finding what to minimize and that requires the real understanding of the problem and some math skills. Let us look at a couple examples.

Least Square fit Revisited.

We are now equipped with a very powerful tool: a method to minimize anyf function of any number of parameters. This means that if we can phrase the problem in terms of (unconstrained) optimization problem, we are done.

Let us first try it on an old problem of least square fit. Recall least square fit problem is that we want to model the data using

$$\tilde{y} = mx + c$$

and our job is to find m and c such that our prediction is collectively close to the real data. In other words, we want to minimize the following cost function

$$cost(m, c) = \sum_{i=1}^{n} (mx_i + c - y_i)^2 \tag{13}$$

where (x_i, y_i) are the data point for $i = 1 \ldots n$.

Even though the expression in Equation 13 looks quite scary. It is just a function of m and c. We plug in m and c and the function gives a number back. All we need to do is find m and c that minimize this function.

Seperating Line

Sometimes your target value(y) is not an bounded one($-\infty, \infty$). For example, given a person's the height, the weight and the waistline, what is the probability that the person is a female. Whatever function we comeup with it better give something from 0 to 1. In a lot of applications, we want to predict a probability given a feature vector, your prediction better be between $(0, 1)$. Our goal is to find a function that when you we put in height, weight, waist line and it return 1 if the person is a female and 0 if the person is a male.

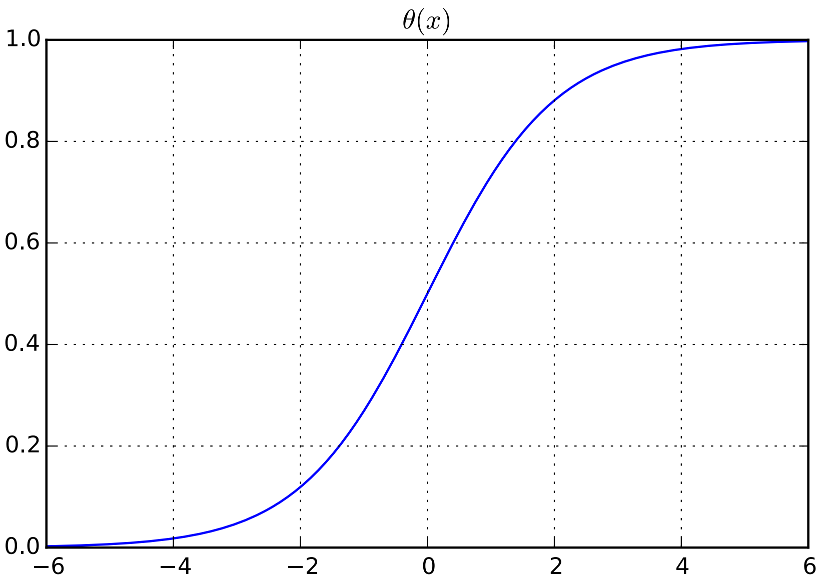
$$F(\text{height}, \text{weight}, \text{waist line}) = \begin{cases} 1 & \text{if female} \\ 0 & \text{if male} \end{cases}$$

Logistic Function

Linear Regression is cleary a bad choice since the output from linear function $y = mx + c$ can go from anywhere $(-\infty, \infty)$. If you recall from the homework, there is a way to turn and unbounded region into bounded one. In the homework we used arctan function. We are going to use something different this time. Let us introduce logistic function(θ); sometimes called sigmoid function [4].

$$\theta(s) = \frac{1}{1 + e^{-s}} \tag{14}$$

Let us look at the plot of the function.



The function turns $-\infty \to 0$ and it turns $+\infty \to 1$. You can actually see mathematically. When $s \to \infty$, the exponential term is 0 and you get $1/1$. When $s \to -\infty$, the exponent term is ∞ and you get $1/\infty = 0$. [5]

One note on the implementation of logistic function. If you try to implement this function and try to calculate $\theta(-2000)$, you will find yourself with some obcure range

[4]This is one of the thing worth committing to our neuron: the trick of turning unbounded interval to bounded one. You will find yourself in a situation where you need to do this so many times.

[5]We could adjust the slope as it pass thorough 0 by scaling s with some constant. But, we don't really need this.

error. This is because the e^{2000} is a really big number. But as you know the function is 0 for all intents and purposes for number that big. So, it is good idea to just put an if statement. That if s is less than some number then just return 0.

Of course, this is not the only function that does the job. But this function has a really nice property that makes our life much simpler. It can be verified easily that

$$\theta(-s) = 1 - \theta(s) \tag{15}$$

All in all, we have a function that turns unbounded range to bounded range

$$(-\infty, \infty) \xrightarrow{\text{Logistic}} (0, 1) \tag{16}$$

If you want other bound interval, you can just shift and scale the logistic function. For example, star rating normally goes from 1 to 5. So you can just multiply logistic function by 4 and add 1.

$$(-\infty, \infty) \xrightarrow{\text{Logistic}} (0, 1) \xrightarrow{4z+1} (1, 5) \tag{17}$$

Linear Function

Before we get to our punch line let us write the linear function

$$\tilde{y}(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots$$

in a cool form using dot product:

$$\tilde{y}(x) = \vec{w} \cdot \vec{x}' \tag{18}$$

where $\vec{x}'$ is a padded version of $\vec{x}$ that is

$$\vec{x} = [x_1, x_2, x_3, x_4, \ldots] \rightarrow \vec{x}' = [\mathbf{1}, x_1, x_2, x_3, x_4, \ldots]. \tag{19}$$

The vector $\vec{w}$ is called the weight vector.

In 1D linear regression the first element is the intercept and the second element is just slope. You can see this my multiplying $\vec{w}$ and $\vec{x}'$

$$\tilde{y}(x) = \vec{w} \cdot \vec{x}' = w_0 + w_1 x_1 = c + m x_1$$

Normally when people use this notation, we drop the prime and just write

$$y = \vec{w} \cdot \vec{x}$$

with implicit padding on $\vec{x}$. So from now on if you see $\vec{w} \cdot \vec{x}$, $\vec{x}$ is implicitly padded.

This linear function turns our input $\vec{x}$ in to an unbounded range.

$$\vec{x} \xrightarrow{\text{Linear}} (-\infty, \infty)$$

Combining the Two

This means that we can just combine linear regression with logistic function to come up with a function that returns something from 0 to 1. The idea looks like the following

$$\vec{x} \xrightarrow{\text{Linear}} (-\infty, \infty) \xrightarrow{\text{Logistic}} (0, 1) \tag{20}$$

Thus, our guess for the probability of $\vec{x}$ looks like

$$\tilde{P}(\vec{x}; \vec{w}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x})}} \tag{21}$$

remember that $\vec{x}$ is implicitly padded. This is a prediction of a vector $\vec{x}$(our features) given the weight vector $\vec{w}$. The vector $\vec{x}$ is our input. Our model is parametrized by $\vec{w}$.

That means for our problems of predicting the probability becomes the problem of finding $\vec{w}$ that optimize the "correctness" of our prediction. This looks scary but if the vector is just a bunch of numbers.

First Attempt

You could do something like square error like we did before

$$cost(\vec{w}) = \sum_{i=1}^{i=n} (y_i - \tilde{P}(\vec{x_i}; \vec{w}))^2 \tag{22}$$

This would work. It has the desired behavior: if $\tilde{P}$ always gives the right answer the cost would be really low. If $\tilde{P}$ get the wrong answer most of the time the cost function would be really high.

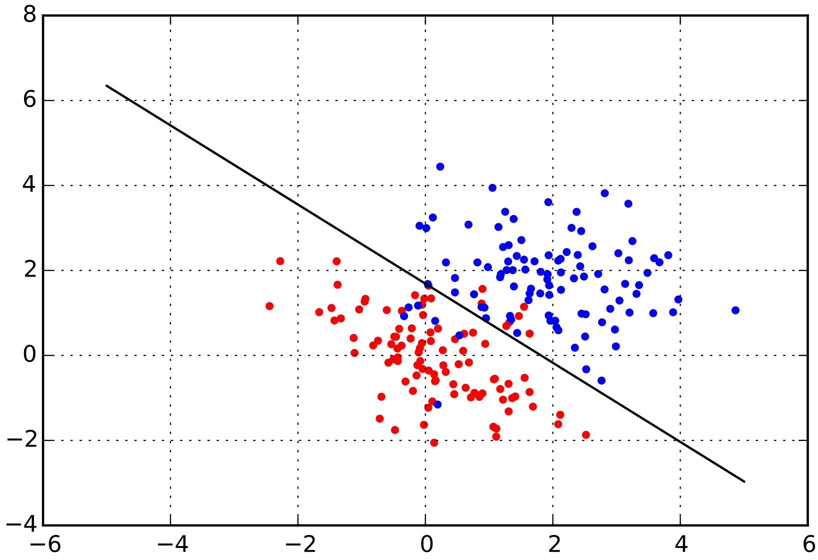
Once we found $\vec{w}$, we can draw the line. This line should be the equation for $\vec{w} \cdot \vec{x} = 0$. For example for 2 features $x = [x_1, x_2]$ the equation we want to plot is

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

or given x_1 you can find x_2 by

$$x_2 = -\frac{1}{w_2}(w_1 x_1 + w_0)$$

The result is shown in the figure below.



This is quite amazing that we start with a problem we sort or know how to do it by hand. With some math we formulate it as optimization problem and we got amazing result. Formulating your problem as an optimization is an art.

There is another way to make this more meaningful statistically. You can read that in logistic regression notes. You will get to use what you learn in discrete math.