Train the model with gradient descent

Course	
Topic	
Type	Lecture
Done	☐
Date	@July 15, 2022
Materials	https://drive.google.com/file/d/1EQC2CnTYN7EUieKc_0v7RkATU1IZGDkO/view?usp=sharing

▼ Summary

Introduction

- Gradient descent is used all over the place in machine learning, not just for linear regression, but for training for example some of the most advanced neural network models, also called

deep learning models.

- Gradient Descent is an algorithm that you can use to try to minimize any function.
- It applies to more general functions, including other cost functions that work with models that have more than two parameters. For instance, if you have a cost function J as a function of w_1, w_2 ….. w_n and b, your objective is to minimize J over the parameters w_1 to w_n and b. In other words, you want to pick values for w_1 $through$ w_n and b, that gives you the smallest possible value of J. Gradient descent is an algorithm that you can apply to try to minimize this cost function J as well.

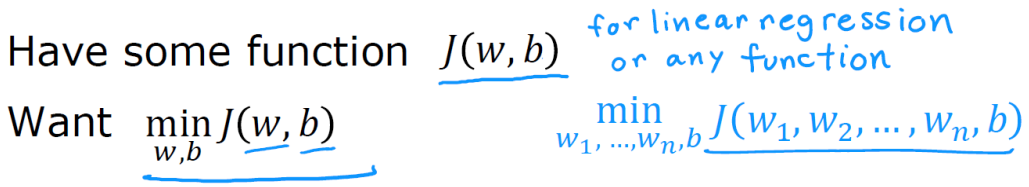

Figure 1: Gradient descent alogorithm overview

Outline for the algorithm:

- Pick some values for w, b
- Change w, b to find a smaller cost function $J(w,b)$
- Repeat the above step until we are at a local min

> 💡 It is important that a function can have more than one local minimum. Gradient descent finds the local minimum(Figure 2).

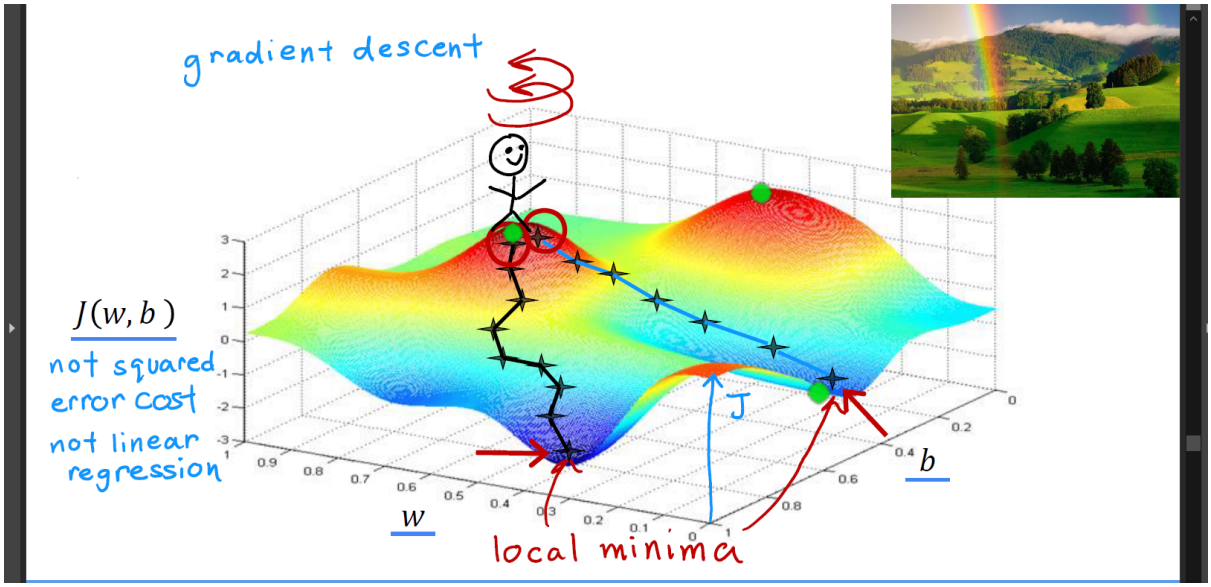

Figure 2: Different local minima for dfiferent starting positions of w and b.

Implementing Gradient Descent Algorithm:

Algorithm:

1. $tmp_b = b - \alpha \frac{\partial}{\partial b} J(w,b)$
2. $tmp_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$
3. $w = tmp_w$
4. $b = tmp_b$
5. Repeat until convergence: not much of a difference in $J(w,b)$ (You reach the point at a local minimum where the parameters w and b no longer change much with each additional step that you take)

What does α mean here?

- α is also called the learning rate.
- The learning rate is usually a small positive number between 0 and 1.
- **What Alpha does is, it basically controls how big of a step you take downhill. If Alpha is very large, then that corresponds to a very aggressive gradient descent procedure where**

you're trying to take huge steps downhill. If Alpha is very small, then you'd be taking small baby steps downhill(Refer to figure 2 for downhill reference)

- **Later on**, we will find how to find the best α.

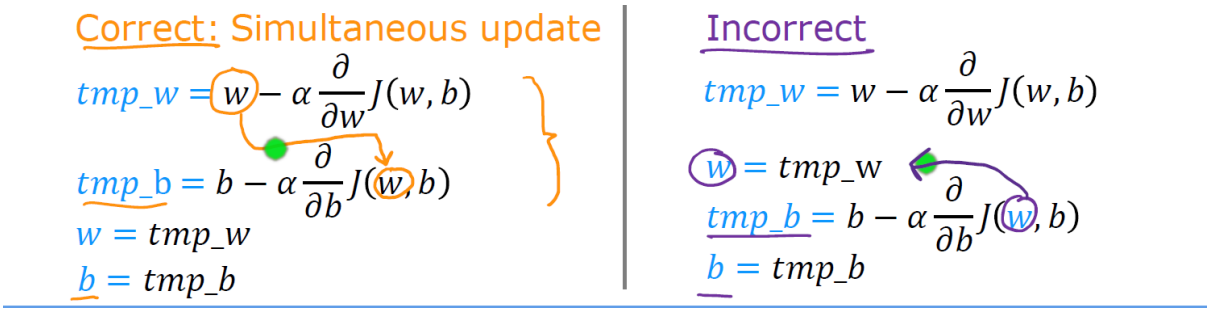

Figure 3: Correct way to calculate gradient descent.

Significance of partial derivative:

To understand the signifance, let us assume that cost function depends only on w $J(w)$.

So now,

$$w = w - \alpha \frac{\partial}{\partial w} J(w)$$

1) You're trying to minimize the cost function **J** by adjusting the parameter **w**. One can look at two-dimensional graphs of the cost function j, instead of three dimensional graphs. Let's look at what gradient descent does on just function J of w. Here on the horizontal axis is parameter w, and on the vertical axis is the cost j of w (Figure 4).

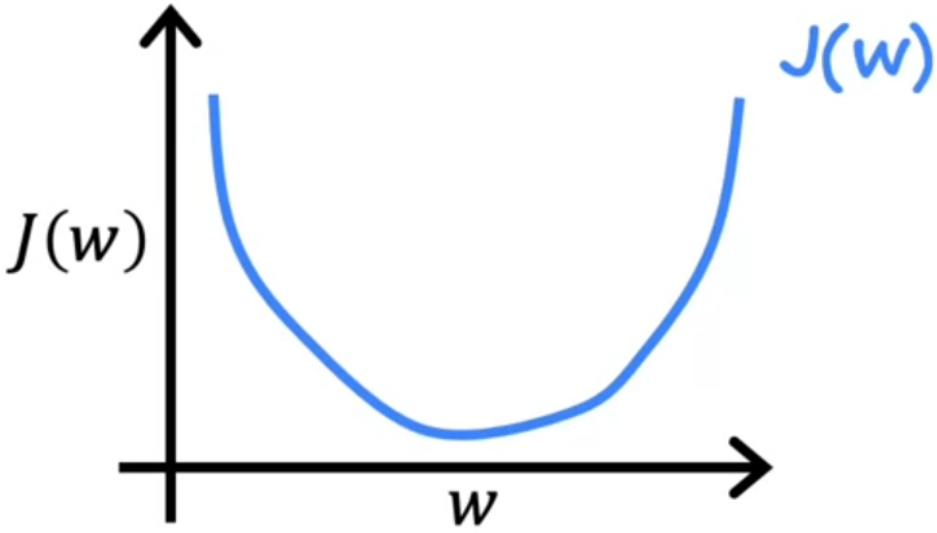

Figure 4: Cost function J as a function of w.

2) Now let us initialize gradient descent with some starting value for w. Let's initialize it at this location(marked red in Figure 5). Imagine that you start off at this point right here on the function **J**, what gradient descent will do is it will update $\boldsymbol{w = w - \alpha \frac{\partial}{\partial w} J(w)}$.

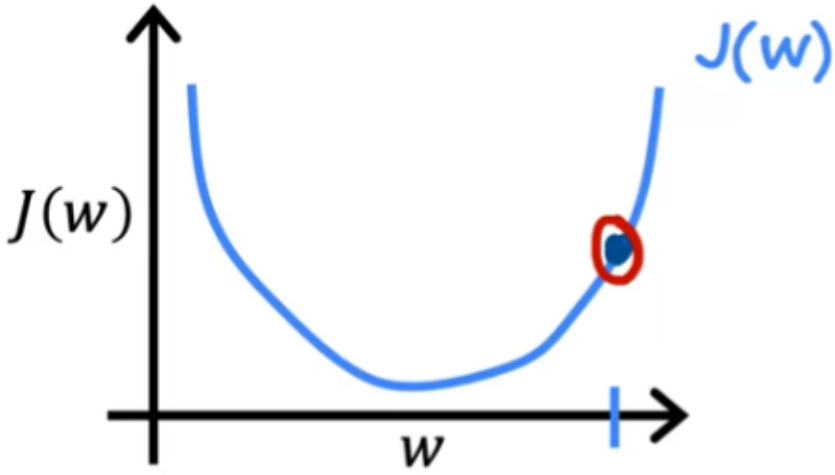

Figure 5: Starting value for w for the above example.

3) Let's look at what this derivative term here means. A way to think about the derivative at this point on the line is to draw a tangent line, which is a straight line that touches this curve at that

point. The slope of this line is the derivative of the function j at this point. To get the slope, you can draw a little triangle like this. If you compute the height divided by the width of this triangle, that is the slope. For example, this slope might be 2 over 1, for instance and when the tangent line is pointing up and to the right, the slope is positive, which means that this derivative is a positive number, so is greater than 0. The updated **w is going to be w minus the learning rate times some positive number**. The learning rate is always a positive number. If you take w minus a positive number, you end up with a new value for **w**, that's smaller. On the graph, you're moving to the left, you're decreasing the value of **w**. You may notice that this is the right thing to do if your goal is to decrease the cost **J**, because when we move towards the left on this curve, the cost **J** decreases, and you're getting closer to the minimum for **J**, which is over here. So far, gradient descent, seems to be doing the right thing.

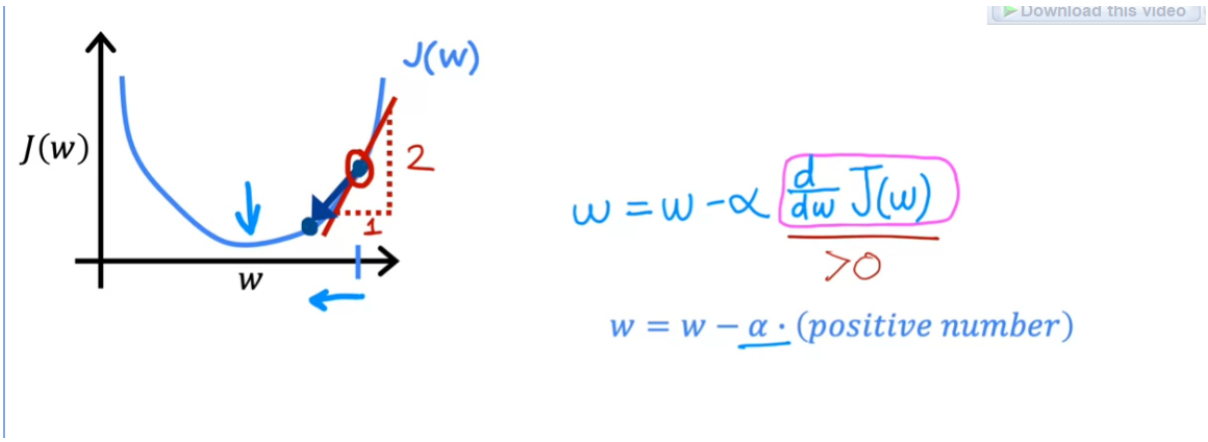

Figure 6: When the slope of the line is positive.

4) Let's take the same function $J(w)$ as above, and now let's say that you initialized gradient descent at a different location. Say by choosing a starting value for w that's over here on the left. That's this point of the function **J**. Now, the derivative term, $\frac{\partial}{\partial w}J(w)$ is the slope of this . But this tangent line is sloping down into the right. This lines sloping down into the right has a negative slope. In other words, the derivative of **J** at this point is a negative number. For instance, if you draw a triangle, then the height like this is negative 2 and the width is 1, the slope is negative 2 divided by 1, which is negative 2, which is a negative number. **When you update w, you get w minus the learning rate times a negative number.** This means you subtract from w, a negative number. But subtracting a negative number means adding a positive number, and so you end up increasing **w**. Because subtracting a negative number is the same as adding a positive number to **w**. This step of gradient descent causes **w** to increase, which means you're moving to the right of the graph and your cost **J** has decrease down to here. Again, it looks like gradient descent is doing something reasonable, is getting you closer to the minimum.

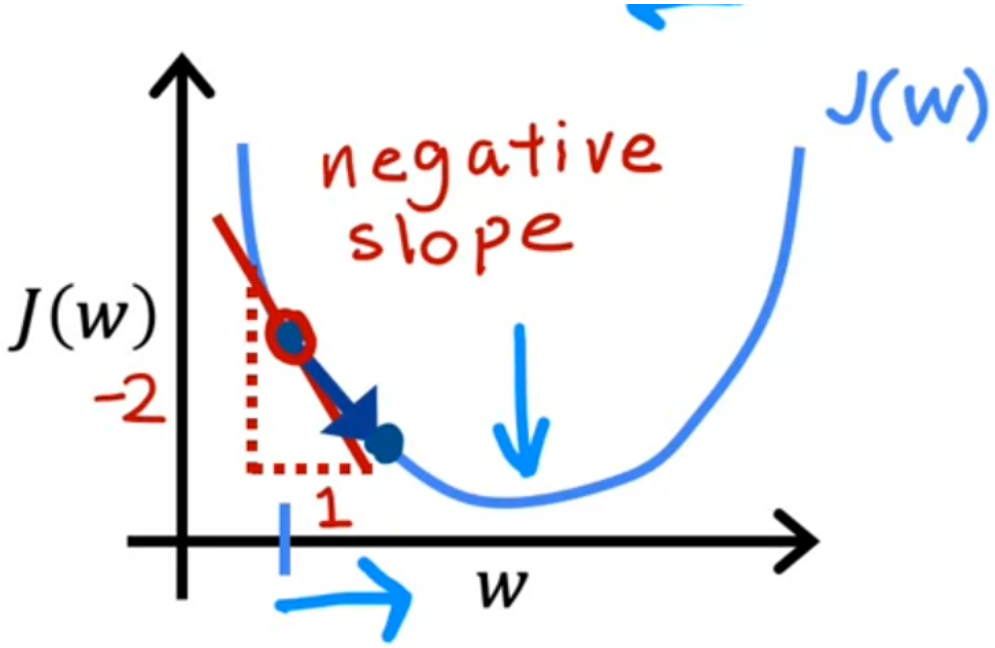

Figure 7: When the slope of the line is negative.

Learning rate α:

- The choice of the learning rate, alpha will have a huge impact on the efficiency of your implementation of gradient descent. And if alpha, the learning rate is chosen poorly rate of descent may not even work at all.

Case 1: When α too small:

Let's start grading descent at the first point(Figure 8). If the learning rate is too small, then we multiply the derivative term by some really, really small number. So you end up taking a very small baby step like that. Then from this point you're going to take another tiny tiny little baby step. But because the learning rate is so small, the second step is also just minuscule. The outcome of this process is that you do end up decreasing the cost **J** but incredibly slowly. But as you may notice you're going to need a lot of steps to get to the minimum. ***So to summarize if the learning rate is too small, then gradient descents will work, but it will be slow. It will take a very long time because it's going to take these tiny tiny baby steps***. And it's going to need a lot of steps before it gets anywhere close to the minimum.

$$w = w - \alpha \frac{d}{dw} J(w)$$

If α is too small...
Gradient descent may be slow.

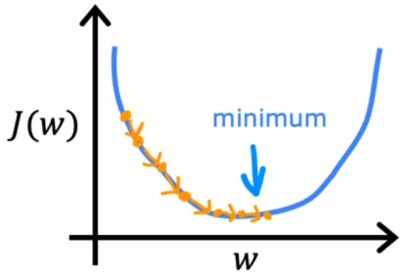

Figure 8: When the learning rate α is too small

Case 2: When α too large:

Let's say we start gradient descent with **w** at this value (Figure 9). So it's actually already pretty close to the minimum. But if the learning rate is too large then you update **w** very giant step to be all the way over here.. So you move from this point on the left, all the way to this point on the right. And now the cost has actually gotten worse. Now the derivative at this new point says to decrease W but when the learning rate is too big, then you may take another huge step with an acceleration and way overshoot the minimum again. So as you may notice you're actually getting further and further away from the minimum. ***To summarize, if the learning rate is too large, then gradient descent may overshoot and may never reach the minimum. And another way to say that is that great intersect may fail to converge and may even diverge.***

If α is too large...

Gradient descent may:
- Overshoot, never reach minimum

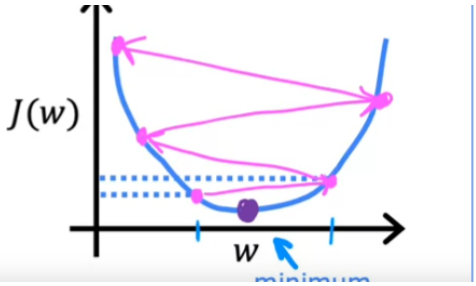

Figure 9: When the learning rate α is too large

Case 3: When you are already at a local minimum, what is α?

Let's suppose you have some cost function **J(Figure 10)** and this cost function has two local minima corresponding to the two valleys that you see here. Now let's suppose that after some number of steps of gradient descent, your parameter **w** is over here, say equal to 5. And so this is the current value of **w** which happens to be a local minimum. The slope of this line is zero and thus the derivative term here is equal to zero for the current value of **w**. And so you're grading descent update becomes **w** is updated to w - $\alpha \cdot zero$. This is the same as saying $w = w$.

So this means that if you're already at a local minimum, gradient descent leaves w unchanged. So if your parameters have already brought you to a local minimum, then further gradient descent steps to absolutely nothing. It doesn't change the parameters which is what you want because it keeps the solution at that local minimum.

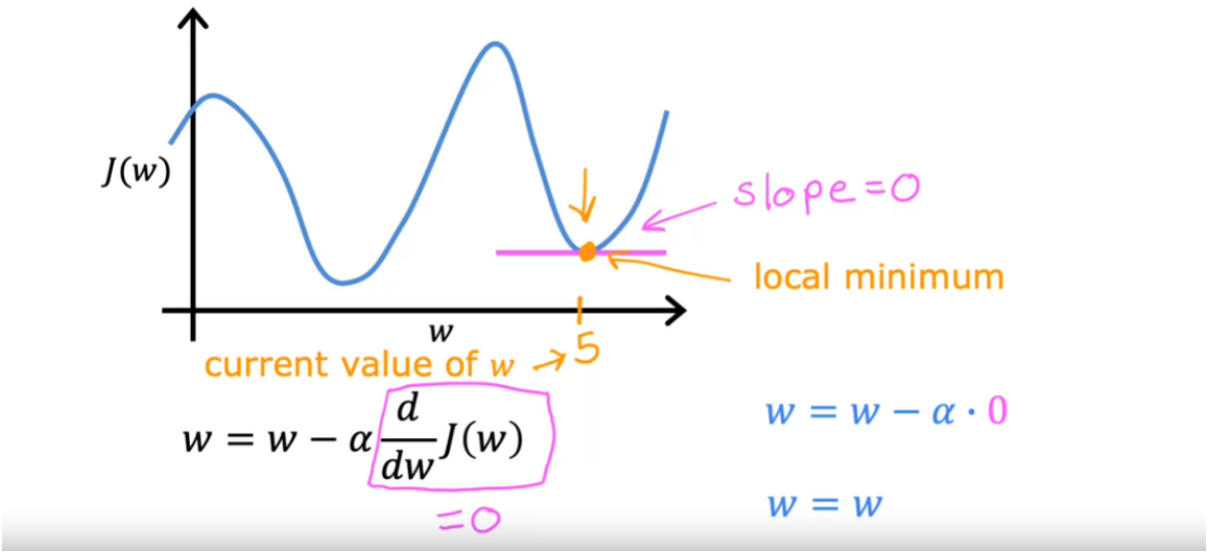

Figure 10: When the slope is 0

One call even reach local minimum with fixed learning rate α. It is explained below:

Let us look at an example(Figure 11). We want to minimize the cost function $J(w)$. Let's initialize gradient descent up here at this point((Figure 11). If we take one update step, maybe it will take us to the next point($marked\ by\ pink$). This derivative is pretty large, so gradieny descent takes a relatively big step right. Now, we're at this second point($marked\ by\ yellow$) where we take another step. Now, the slope is not as steep as it was at the first point. So the derivative isn't as large and so the next update step will not be as large as that first step. Now,at this third point ($marked\ by\ red$) the derivative is smaller than it was at the previous step and will take an even

smaller step as we approach the minimum. The derivative gets closer and closer to zero. So as we run gradient descent, eventually we're taking very small steps until we finally reach a local minimum. So just to recap, ***as we get nearer a local minimum, gradient descent will automatically take smaller steps***. Because as we approach the local minimum, the derivative automatically gets smaller. And that means the update steps also automatically gets smaller, even if the learning rate α is kept at some fixed value.

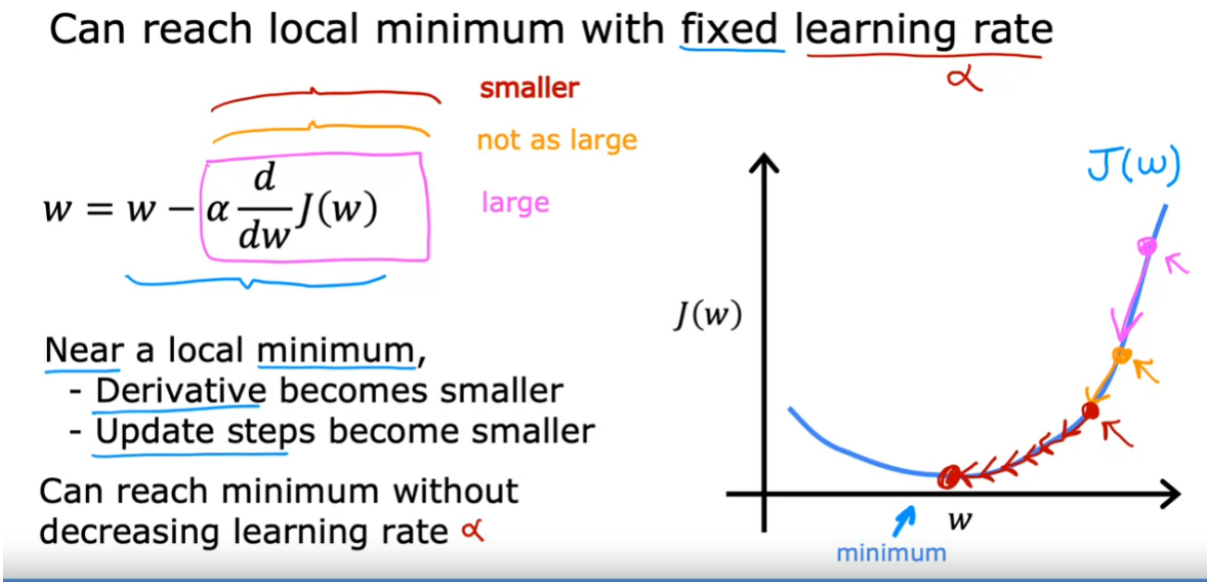

Figure 11: Reaching local minimum when α is fixed.

Gradient Descent for Linear Regression:

Gradient Descent algorithm for linear regression calculates minimum cost function by calculating values of w and b.

According to the algorithm,

$$tmp_b = b - \alpha \frac{\partial}{\partial b} J(w,b) \longrightarrow 1$$

$$tmp_w = w - \alpha \frac{\partial}{\partial w} J(w,b) \longrightarrow 2$$

So now calculating $\frac{\partial}{\partial b} J(w,b)$

We know, $J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$

Substituting in equation (1) :

$\frac{\partial}{\partial b} J(w,b) = \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$

We know $f_{w,b}(x^{(i)}) = wx^{(i)} + b$ from Regression Model

$= \frac{\partial}{\partial b} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$

$= \frac{1}{2m} \sum_{i=1}^{m} 2 \left(wx^{(i)} + b - y^{(i)}\right) \frac{\partial}{\partial b}(wx^{(i)} + b - y^{(i)})$

$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \cancel{2} \left(wx^{(i)} + b - y^{(i)}\right) (1)$

$= \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$

💡 Therefore, $\frac{\partial}{\partial \mathbf{b}} \mathbf{J}(\mathbf{w},\mathbf{b}) = \frac{1}{\mathbf{m}} \sum_{\mathbf{i=1}}^{\mathbf{m}} \left(\mathbf{f}_{\mathbf{w},\mathbf{b}}(\mathbf{x}^{(\mathbf{i})}) - \mathbf{y}^{(\mathbf{i})}\right)$

Now calculating $\frac{\partial}{\partial w} J(w,b)$

Substituting in equation (2) :

$\frac{\partial}{\partial w} J(w,b) = \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$

We know $f_{w,b}(x^{(i)}) = wx^{(i)} + b$ from Regression Mod

$= \frac{\partial}{\partial w} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$

$= \frac{1}{2m} \sum_{i=1}^{m} 2 \left(wx^{(i)} + b - y^{(i)}\right) \frac{\partial}{\partial w}(wx^{(i)} + b - y^{(i)})$

$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \cancel{2} \left(wx^{(i)} + b - y^{(i)}\right) (\mathbf{x}^{(\mathbf{i})})$

$= \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right) x^{(i)}$

💡 $\frac{\partial}{\partial \mathbf{b}} \mathbf{J}(\mathbf{w},\mathbf{b}) = \frac{1}{\mathbf{m}} \sum_{\mathbf{i=1}}^{\mathbf{m}} \left(\mathbf{f}_{\mathbf{w},\mathbf{b}}(\mathbf{x}^{(\mathbf{i})}) - \mathbf{y}^{(\mathbf{i})}\right) \mathbf{x}^{(\mathbf{i})}$

Therefore,

REPEAT UNTIL COVERGENCE

{

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)} \right)$$

}

NOTE:

- Some functions can more than one local minimum. So, it depends on the starting position of where you start applying gradient descent(Figure 12).

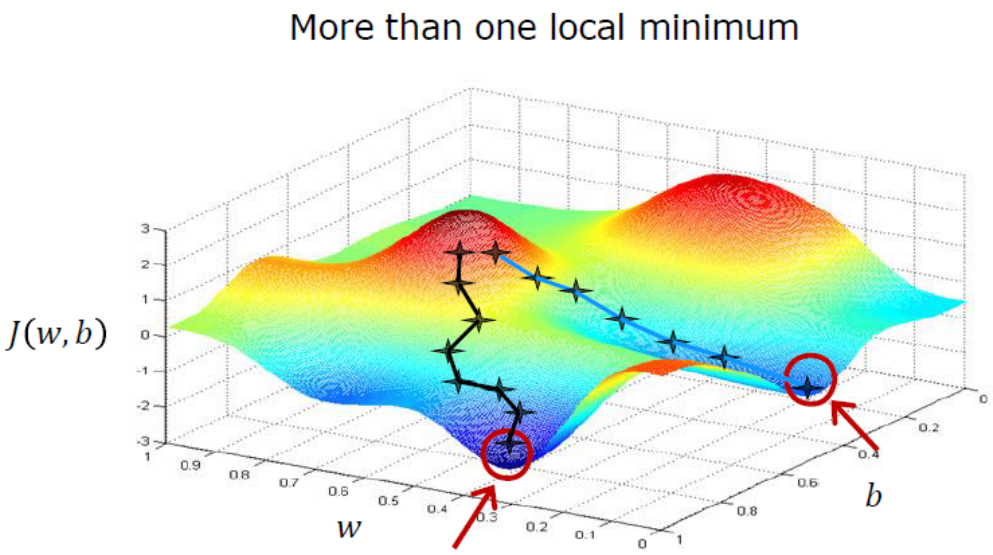

Figure 12: Function with multiple local minima

- When you're using a squared error cost function with linear regression, the cost function does not and will never have multiple local minima.
  It has a single global minimum because of the bowl-shape. The technical term for this is that this cost function is a convex function.

- Informally, a convex function is of bowl-shaped function and it cannot have any local minima other than the single global minimum.

- When you implement gradient descent on a convex function, one nice property is that so long as you're learning rate is chosen appropriately, it will always converge to the global minimum.

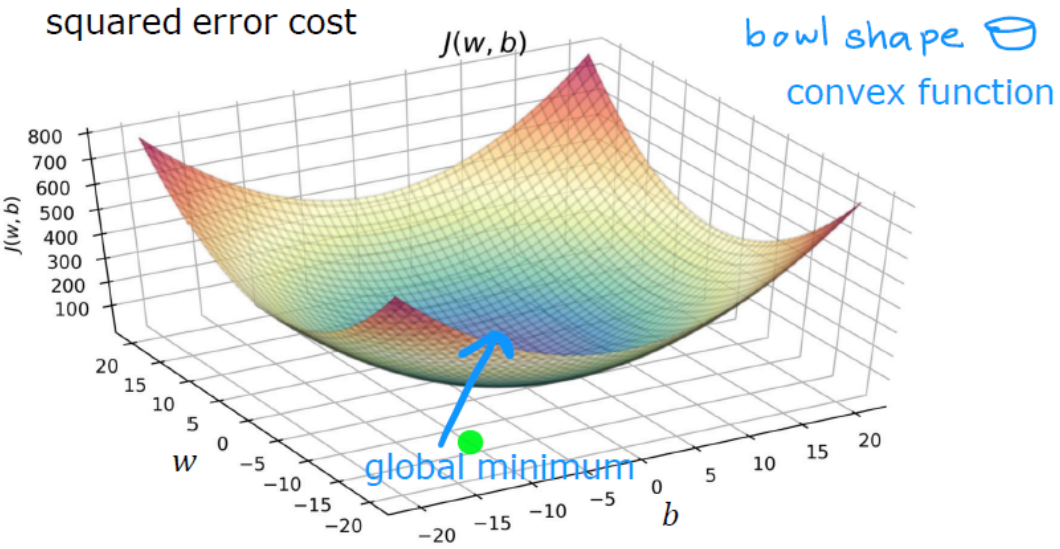

Figure 13: Square error cost function with one global minimum

Example of Gradient Descent for linear regression:

Step by step implementaion of the gradiant descent algorithm for Linear Regression:

1 →

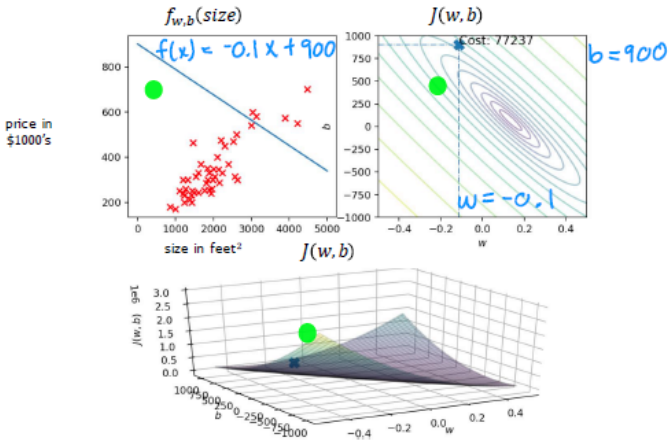

Figure 14: Illustrating 1st step of Gradient Descent for linear regression

2 →

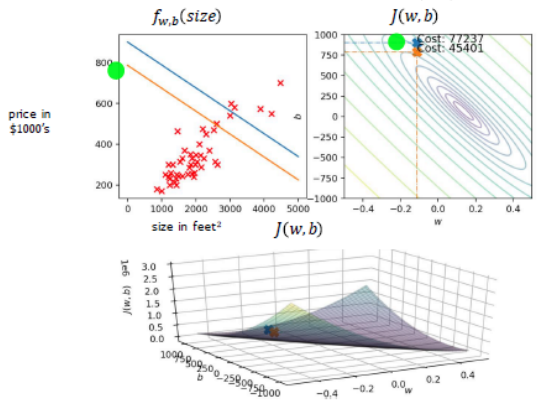

Figure 15: Illustrating 2nd step of Gradient Descent for linear regression

3 →

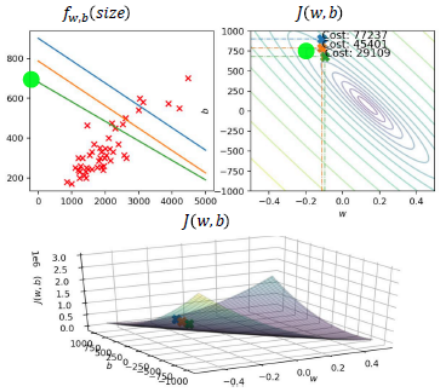

Figure 16: Illustrating 3rd step of Gradient Descent for linear regression

Last Step →

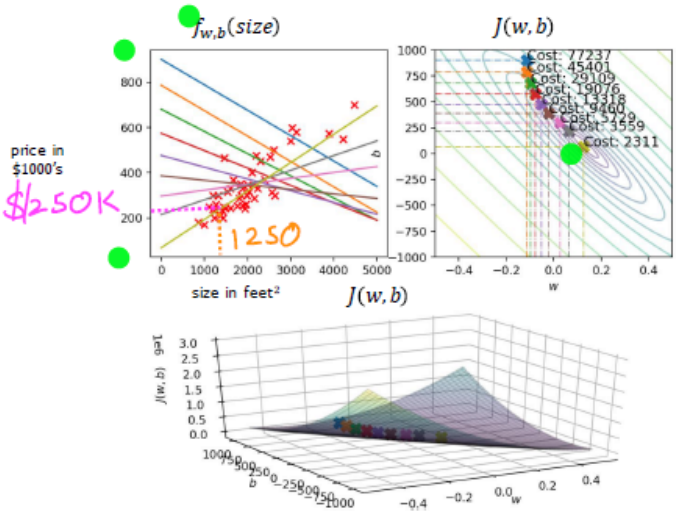

Figure 17: Illustrating last step of Gradient Descent for linear regression

Bash Gradient Descent:

- This gradient descent process is called batch gradient descent.
- The term bashed grading descent refers to the fact that on every step of gradient descent, we're looking at all of the training examples, instead of just a subset of the training data.
- So in computing grading descent, when computing derivatives , we are computing the sum from i =1 to m and bash gradient descent is looking at the entire batch of training examples at each update.

There are other versions of gradient descent that do not look at the entire training set, but instead looks at smaller subsets of the training data at each update step.

But we'll use batch gradient descent for linear regression.