Training Linear Regression

Ch 2.4

Train the Model with Gradient Descent Algorithm

Content

- Gradient Descent
- Implementing Gradient Descent
- Gradient Descent Intuition
- Learning Rate
- Gradient Descent for Linear Regression
- Running Gradient Descent
- Optional Lab: Gradient Descent

Gradient Descent Algorithm

- Gradient descent is an iterative optimization algorithm commonly used in machine learning to minimize the cost (loss) function *J(w, b)* by adjusting model parameters in the direction of the steepest descent of the cost surface.
- Learning how Gradient Descent works is one of the main building block s of how machine learning works

Where do we use it

1. Linear Regression: Gradient descent is used to optimize the weights of a linear regression model to minimize the mean squared error (MSE) between predicted and actual values.
2. Logistic Regression: Gradient descent is used to optimize the weights of a logistic regression model to minimize the loss and improve its ability to classify data points.
3. Neural Networks: Gradient descent is the primary algorithm for training neural networks. It is used to adjust the weights and biases of the network's connections to minimize the loss function and improve its performance on a given task.
4. Support Vector Machines (SVMs): Gradient descent is used to optimize the hyperparameters of SVMs to find the optimal decision boundary that maximizes the margin between the two classes.
5. Regularization Techniques: Gradient descent is often used in conjunction with regularization techniques, such as L1 and L2 regularization, to prevent overfitting and improve the generalization ability of machine learning models.

Review of linear regression

Have some function J(w,b), cost function,
Want to minimize J(w,b) with respect to w and b:

$$\min_{w,b} J(w,b)$$

How is that implemented:

1. Start with some initial values for w, b, for example: w= 0, b=0
2. Keep changing w, b to reduce J(w,b)
3. Keep changing until it settle at or near a minimum value.
4. There is a possibilty of having more than 1 local minimum

What can we do with Gradient Descent

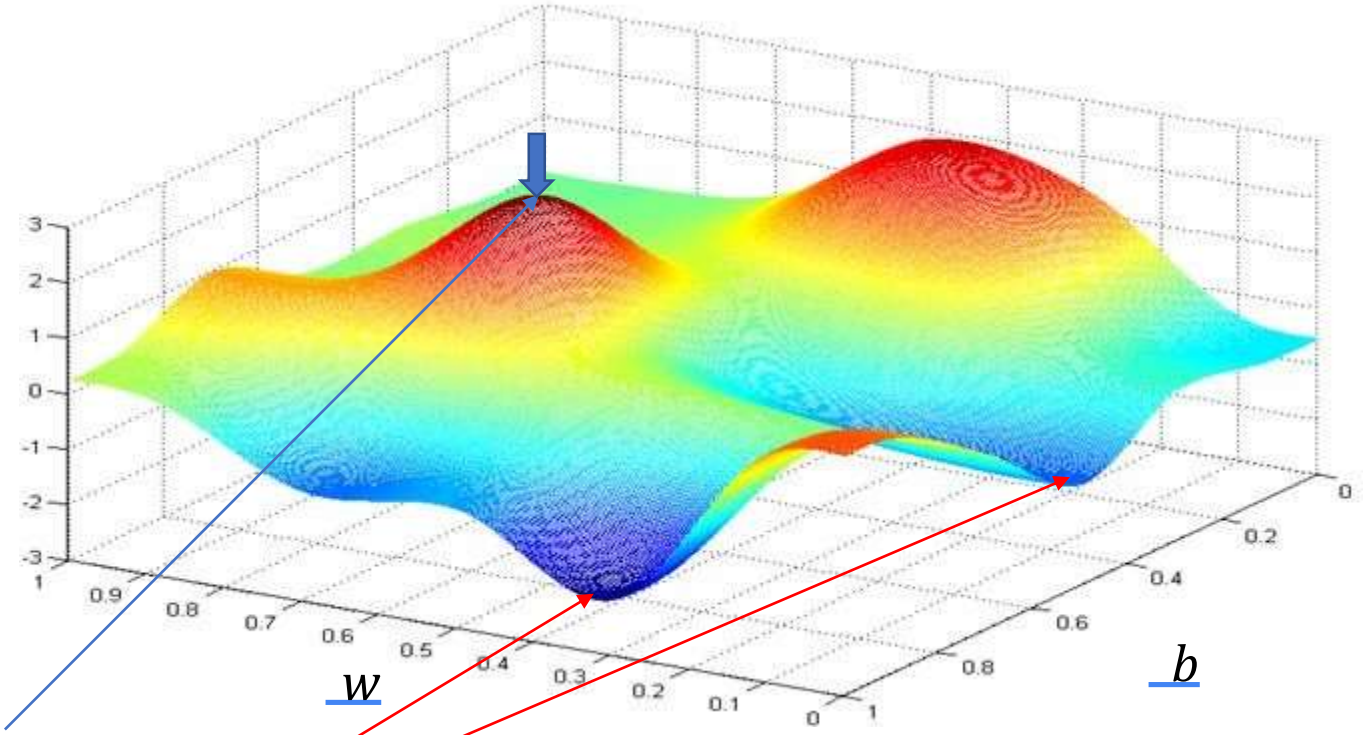
Can be use for J(w,b), or any other function, even with large number of parameters:

$$\min_{w_1, \ldots, w_n, b} J \quad (w_1, w_2, \ldots, w_n, b)$$

Gradient Descent can help us find the values of w_n and b that give the minimum value for J(w_1, w_2, w_3, ... w_n, b)

Graphical Representation of Gradient Descent

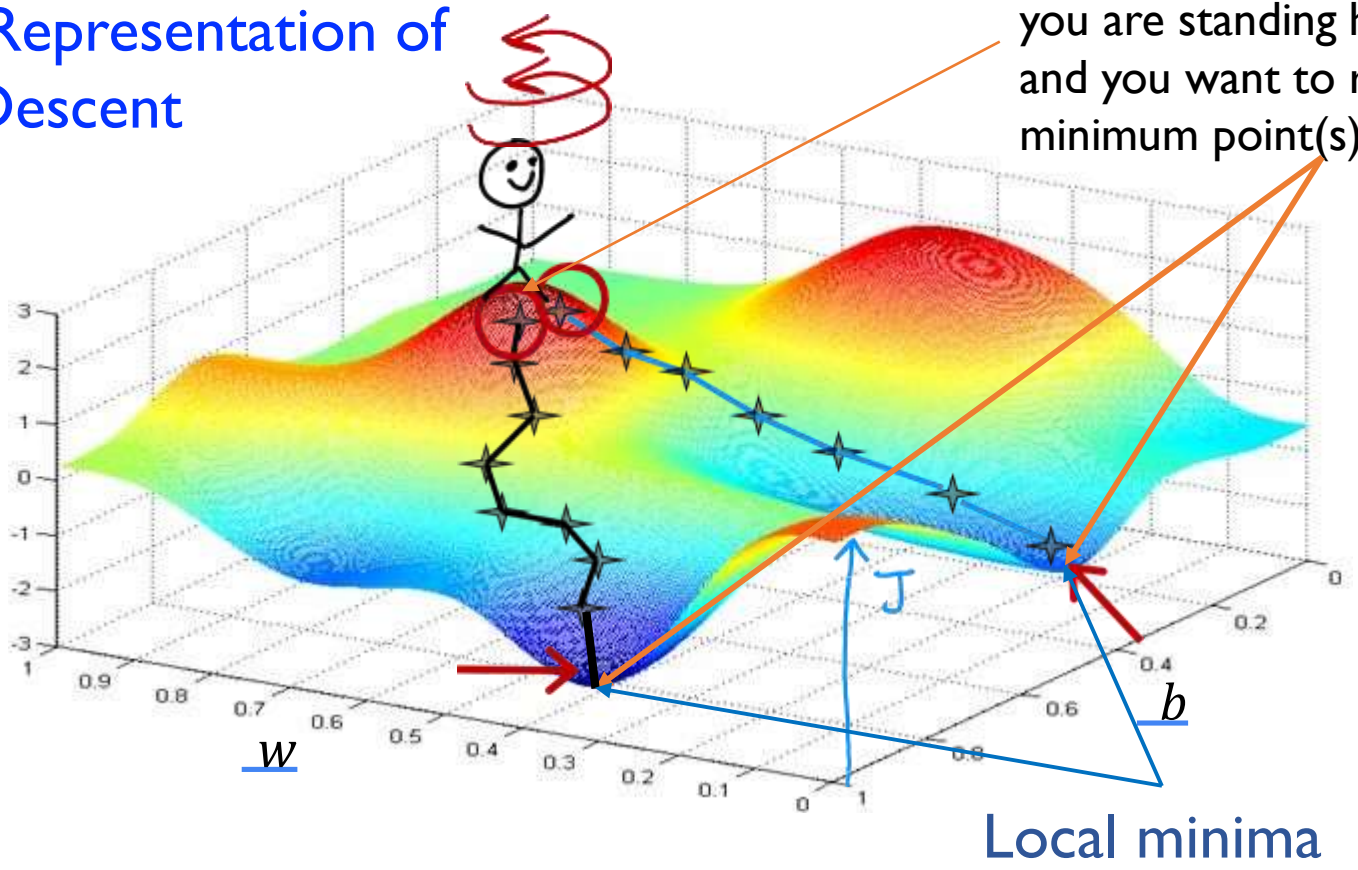
$J(w, b)$



you are standing here
and you want to reach the
minimum point(s)

Graphical Representation of Gradient Descent

$J(w, b)$



Starting at different points migh lead to different minima which means different values for w, b

What do we conclude

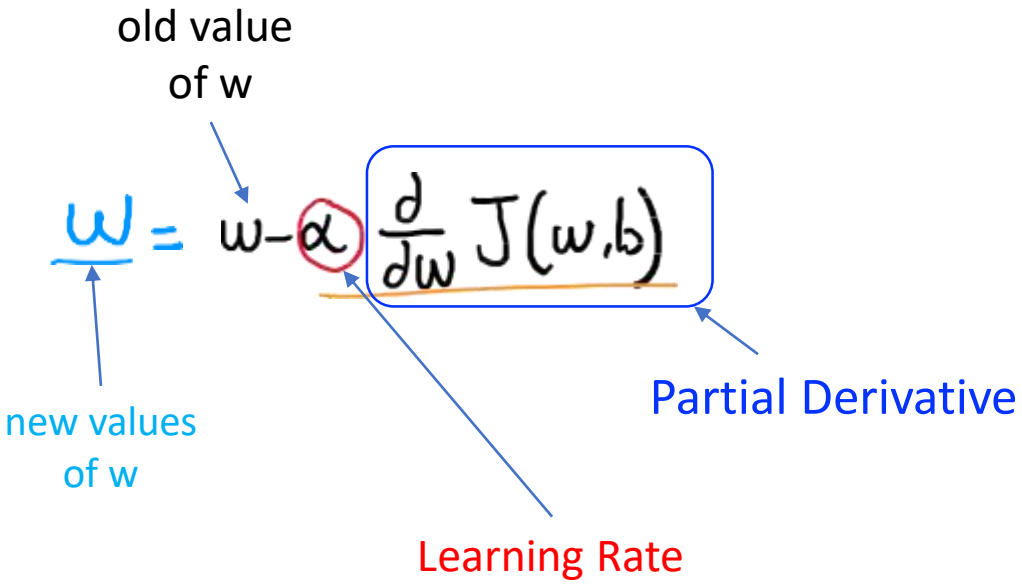
- To find the regression model, we need to find the values of w and b, such that $y(x) = wx + b$
- To find the w, and b, we use the cost function J(w,b), which calculates the difference between True values and Predicted values
- To find the values of w and b, we minimize J(w,b) by changing the values of w and b, so we search for w, b that make the cost function J(w,b) minimum.
- One of the best way to find w and b is to use the Gradient Descent algorithm, that was graphical described in this part

In the upcoming section we look at the mathematical expressions of Gradient Descent

Implementing Gradient Descent

in Mathematical Expressions

Gradient descent algorithm: the expression

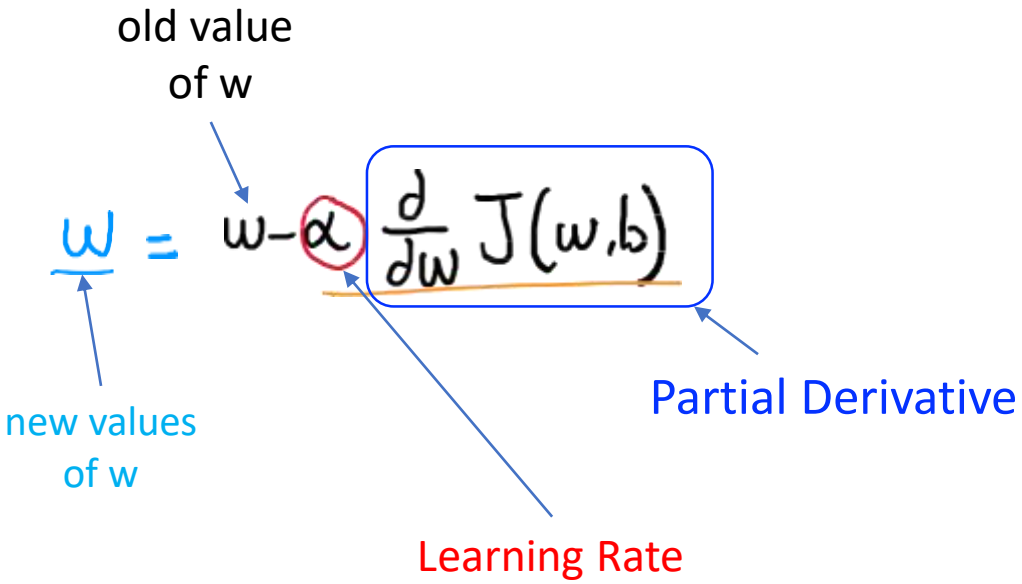


The equal sign (=) in the expression is the assignment operator, which assign the right value to the left variable.

e.g. a = a + 1

means take the old value of a and add 1 to it and assign the value to the new a.

Gradient descent algorithm: the expression

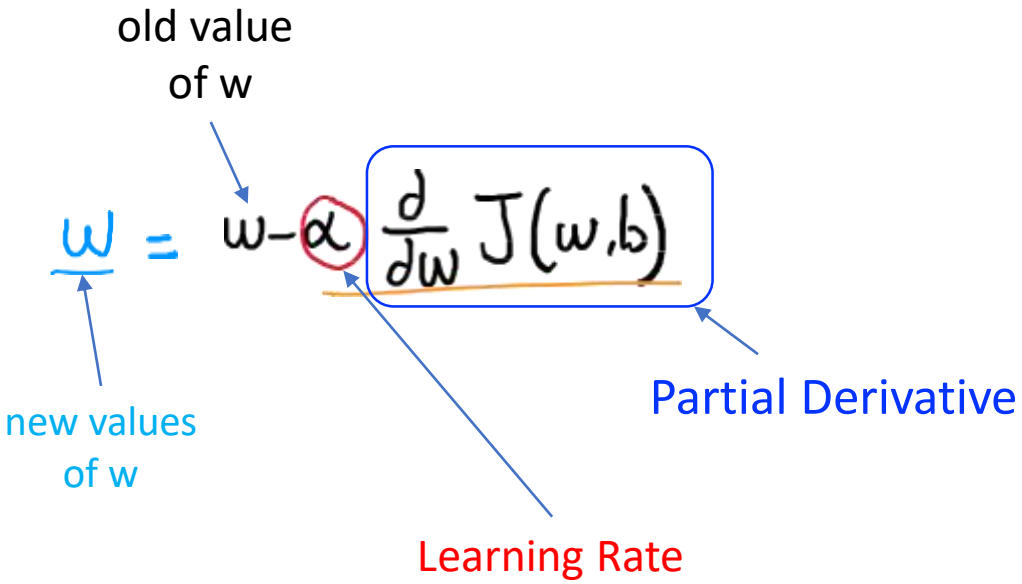


$b = b - \alpha \frac{d}{db} J(w,b)$

Simultaneously update w and b

The learning rate α (alpha) in gradient descent is a crucial hyperparameter that controls the step size at each iteration. A higher learning rate leads to larger updates, while a lower learning rate leads to smaller updates.

Gradient descent algorithm: the expression



$$b = b - \alpha \frac{d}{db} J(w,b)$$

Simultaneously update w and b

The Partial Derivative:

The partial derivative term indicates the direction and magnitude of the steepest descent of the cost function,

(i.e. the slope), guiding the algorithm's step towards minimizing the cost function $J(w,b)$.

Gradient descent algorithm

Repeat until convergence

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

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

$$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

$$tmp\_b = b - \alpha \frac{\partial}{\partial b} J(w, b)$$

$$w = tmp\_w$$

$$b = tmp\_b$$

Simultaneously update w and b

- When updating w and b simultaneously, the algorithm considers the combined effect of both parameters on the error.
- This ensures that the updates are coordinated and lead to an overall improvement in the model's fit.
- These update rules are applied simultaneously, meaning that both w and b are adjusted in the same iteration.

$(w, b) = (w - \alpha * \partial J/\partial w, b - \alpha * \partial J/\partial b)$

What do we conclude ...

What are w and b?

- w (weight): Represents the slope of the line in linear regression. It indicates how much the predicted value changes as the input value changes.
- b (bias): Represents the intercept of the line in linear regression. It indicates the predicted value when the input value is zero.

What are the update equations?

- $w = w - \alpha * \partial E/\partial w$: This equation updates the weight (w) based on the partial derivative of the loss function ($\partial E/\partial w$) and the learning rate (α).
- $b = b - \alpha * \partial E/\partial b$: This equation updates the bias (b) based on the partial derivative of the loss function ($\partial E/\partial b$) and the learning rate (α).

What do we conclude

What do the partial derivatives mean?

- The partial derivatives ($\partial E/\partial w$ and $\partial E/\partial b$) indicate the direction and magnitude of the steepest descent of the loss function.
- In simpler terms, they tell us which way to adjust the parameters (w and b) to reduce the error more effectively.

What does the learning rate (α) do?

- The learning rate (α) controls the step size of the updates.
- A larger learning rate leads to faster convergence but may cause overshooting the minimum.
- A smaller learning rate leads to slower convergence but is less likely to overshoot the minimum.

What is the overall goal?

- The goal of these equations is to iteratively adjust the parameters (w and b) to minimize the loss function, which ultimately improves the model's fit to the data.
- In essence, these equations are the core of the gradient descent algorithm, enabling it to optimize the parameters of a model and achieve better predictions.

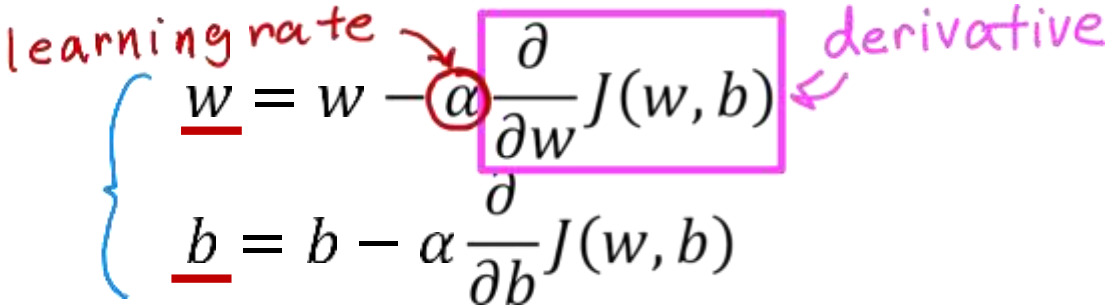
To understand how updates get occured, we need to review the partial derivative

Gradient Descent Intuition

This section explain the role of the learning rate and the derivative terms in deciding the updates of w and b

Gradient descent algorithm

repeat until convergence {



Just to make the calculation simple we will use one parameter only (w)

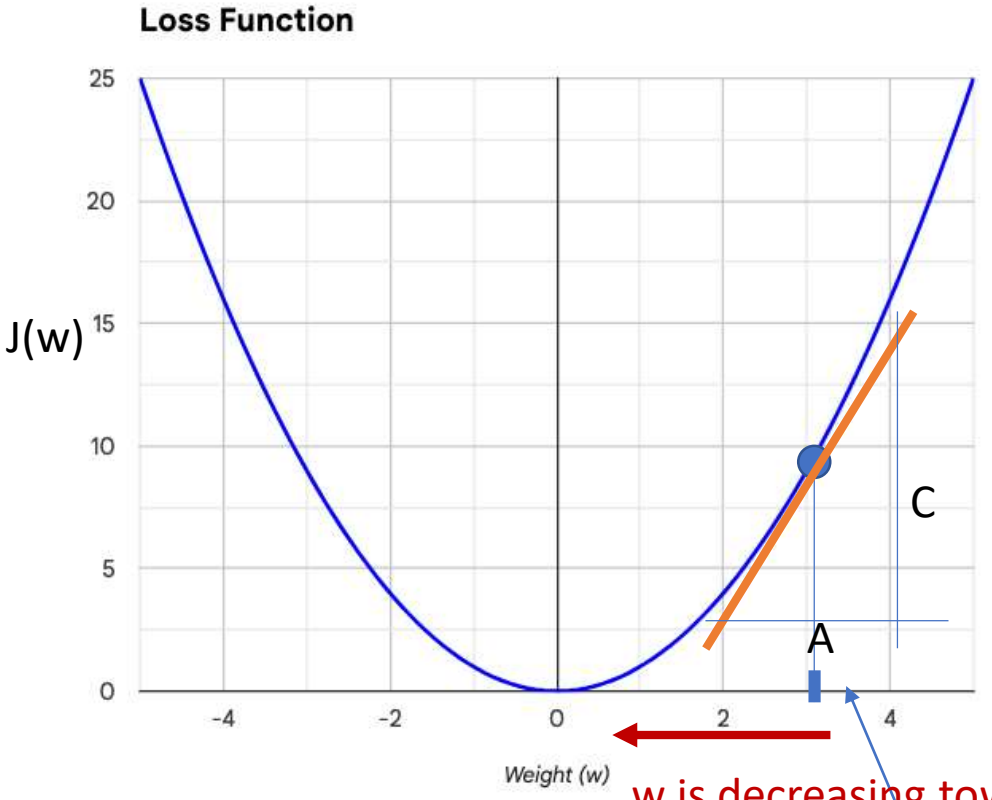
$J(w)$

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

$\min_w J(w)$

The learning rate determines how big is the step we take when we update w and b

The impact of the derivative on the parameters updates



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

Derivative = slope of J(w) = C/A = +

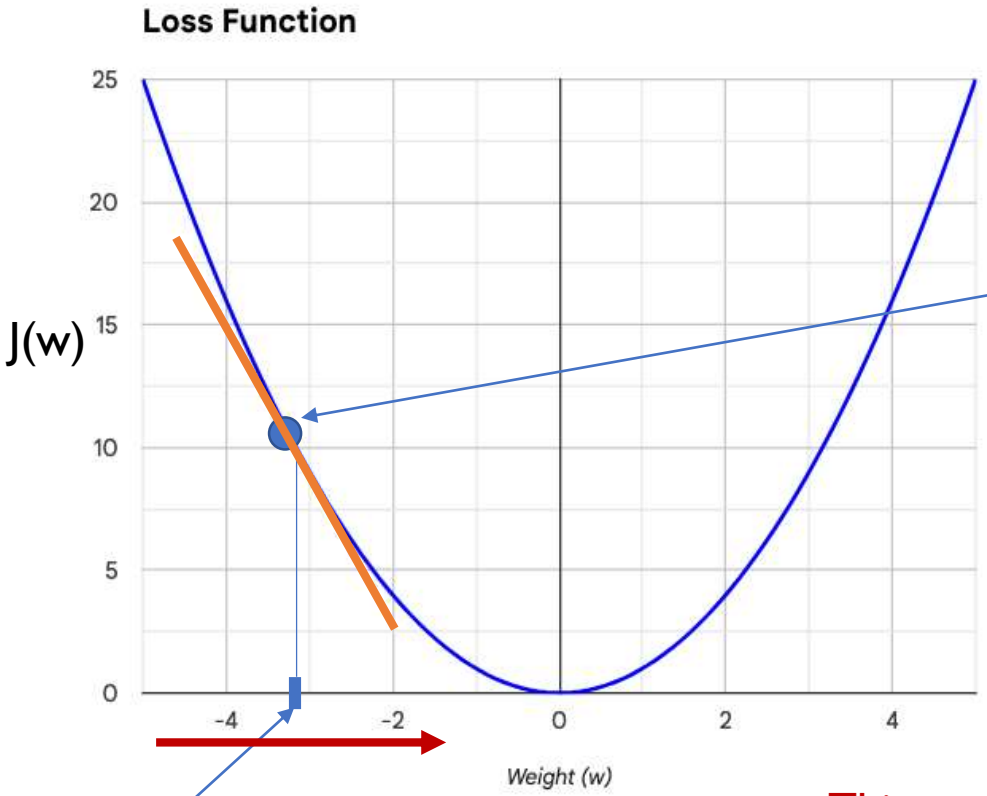
$w = w - \alpha$. Derivative(positive number)

α: is always positive number

→ w(new) = w(old) – positive value

→ w will move to the left, in the direction of the minimum

The impact of the derivative on the parameters updates



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

Derivative = slope of J(w) = - (negative)

w = w - a. (negative number)

a: is always positive number

→ w(new) = w(old) + positive value

→ w will move to the right, in the direction of the minimum

w is increasing towards the min

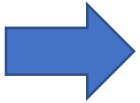
initialize w at this point

This means that the gradient descent is doing something resonable, i.e. moving with J(w) towards the min.

To conclude ...

- This derivative indicates the sensitivity of the loss function to changes in *w*. A positive derivative implies that increasing w will reduce the loss, while a negative derivative suggests that decreasing w will reduce the loss.
- The weight update rule, also known as the gradient descent algorithm, utilizes the derivative to adjust w in the direction that minimizes the loss.
- In summary, the derivative of the loss function serves as a guiding signal for updating the weight parameter, ensuring that the weight moves towards the value that minimizes the overall loss.

But what is the role of the learning rate a ?



Learning Rate

$w = w - \alpha * dJ/dw$

where: α is the learning rate, which controls the step size of the update

The main question becomes, how fast we shall be moving towards the minimum, or what determines the speed of change towards the min.

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

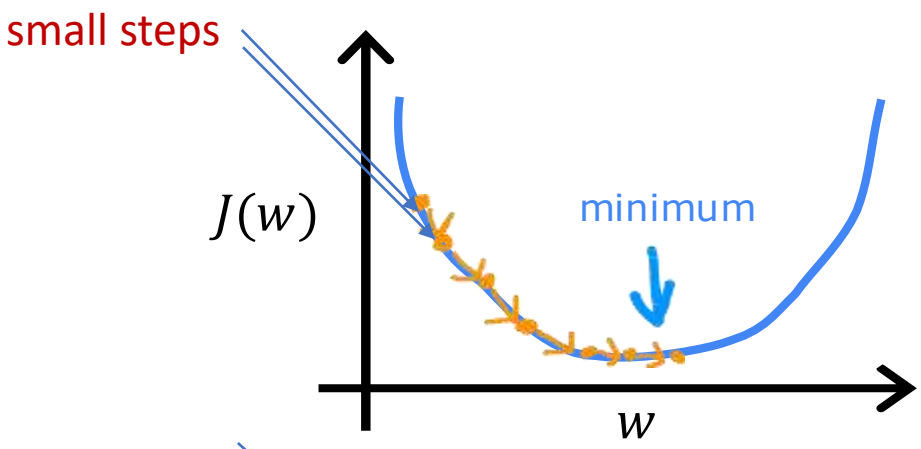
If α is too small…

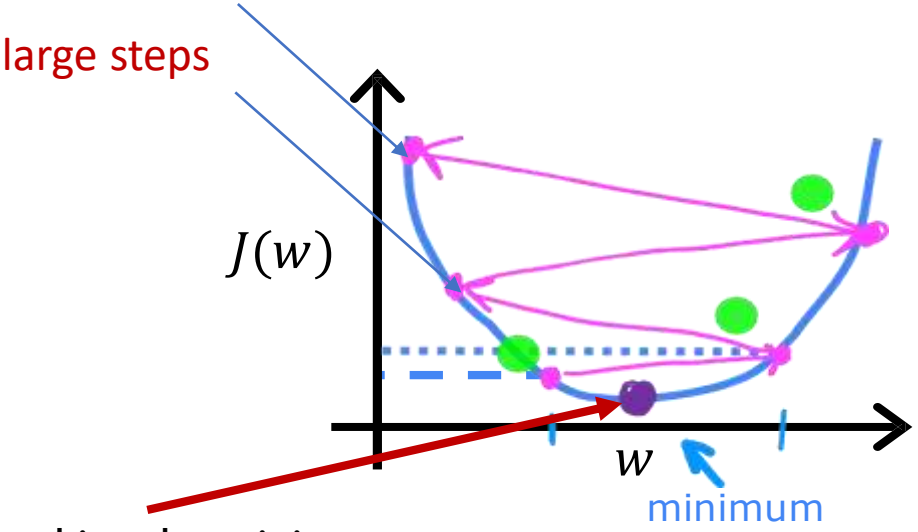
Gradient descent may be slow, and it might take long time to converge

If α is too large…

Gradient descent may:

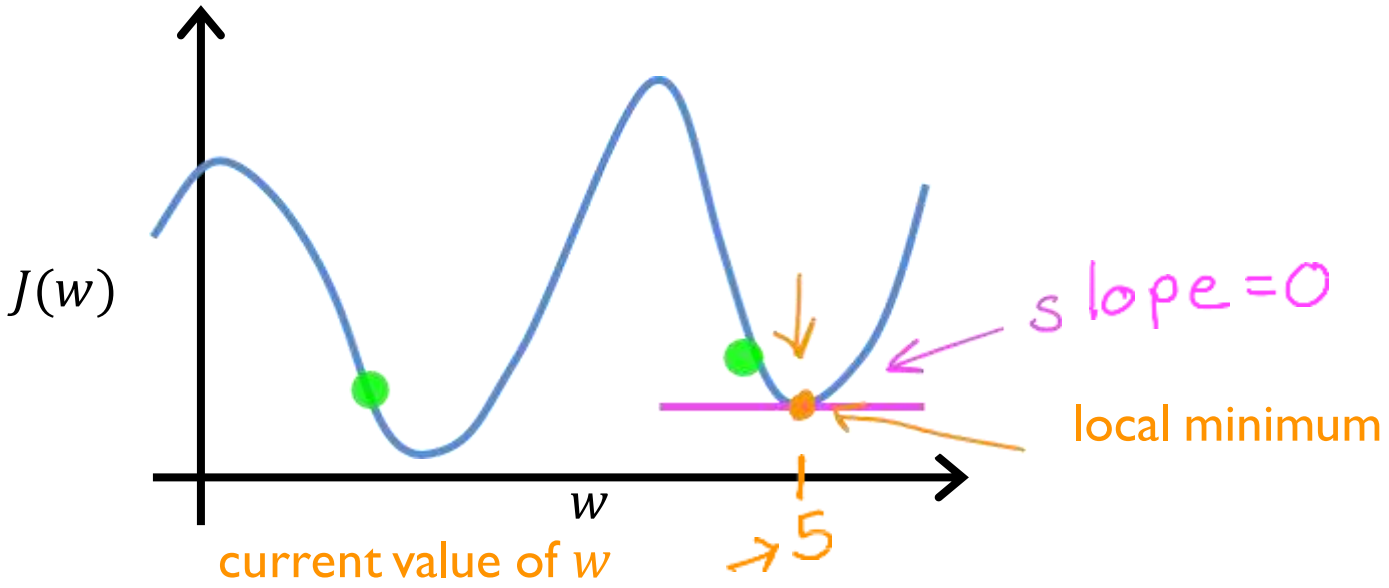
- Overshoot, never reach minimum
- Fail to converge, which means diverge





the algorithm skips the minimum

What if the value of w is at the minimum ??



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

0.1

At the minimum point the slope = 0

w = w – a (derivative)

w = w – a (0)

w = w

If the algorithm reaches the minimum, then further steps will do nothing

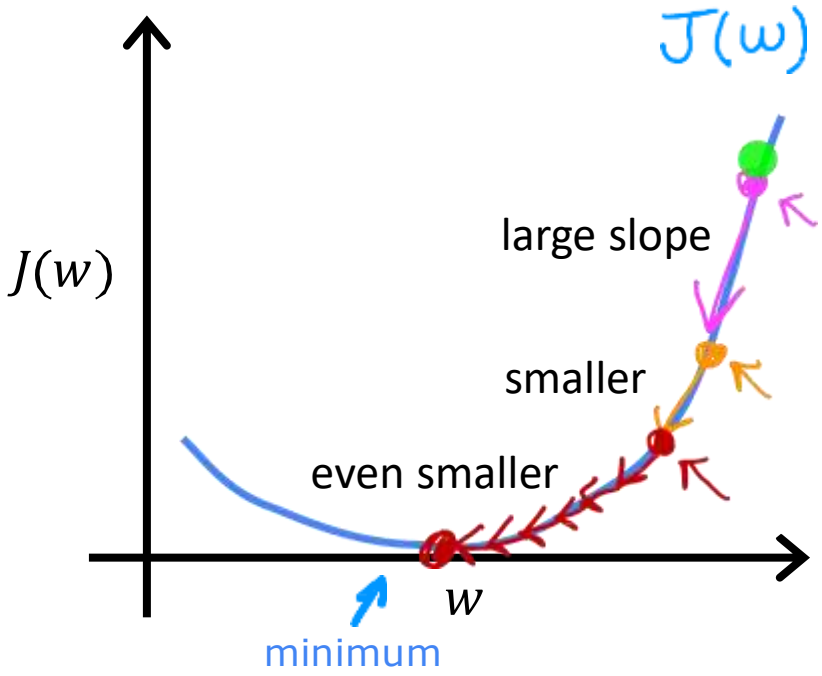
Can reach local minimum with fixed learning rate

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

smaller

not as large

large



Near a local minimum,

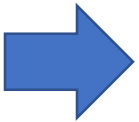
- Derivative becomes smaller
- Update steps become smaller

Can reach minimum without decreasing learning rate

To conclude ...

- The learning rate α plays a crucial role in the convergence process of model parameters, influencing the speed and stability of the optimization algorithm. It determines the step size of the adjustments made to the model parameters during each iteration of the training process.
- Optimal learning rate: The optimal learning rate strikes a balance between speed and stability, guiding the algorithm towards the desired minimum while avoiding oscillations or getting stuck. Finding the optimal learning rate requires careful experimentation and hyperparameter tuning.

So far we applied the gradient descent in general. So lets see how it can be applied to linear regression



Gradient Descent for Linear Regression

We have discussed the linear regression model,
and then developed the concept of the cost function,
and then develop the gradient descent to find the minimum,
now we will bring all these together

Linear regression model

$f_{w,b}(x) = wx + b$

Cost function

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

Gradient descent algorithm

repeat until convergence

{

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

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

}

how do we get these formulas ... Calculus

$$\frac{\partial}{\partial w} J(w,b) = \frac{d}{dw} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{d}{dw} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

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

$$\frac{\partial}{\partial b} J(w,b) = \frac{d}{db} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{d}{db} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

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

no $x^{(i)}$

The linear Model

$$\hat{y}^{(i)} = w \cdot x^{(i)} + b$$

The cost function

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

the cost function as a function of $x^{(i)}$

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

the derivative of the cost function with respect to w

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

the derivative of the cost function with respect to b

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

Gradient descent algorithm

repeat until convergence {

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

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

}

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

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

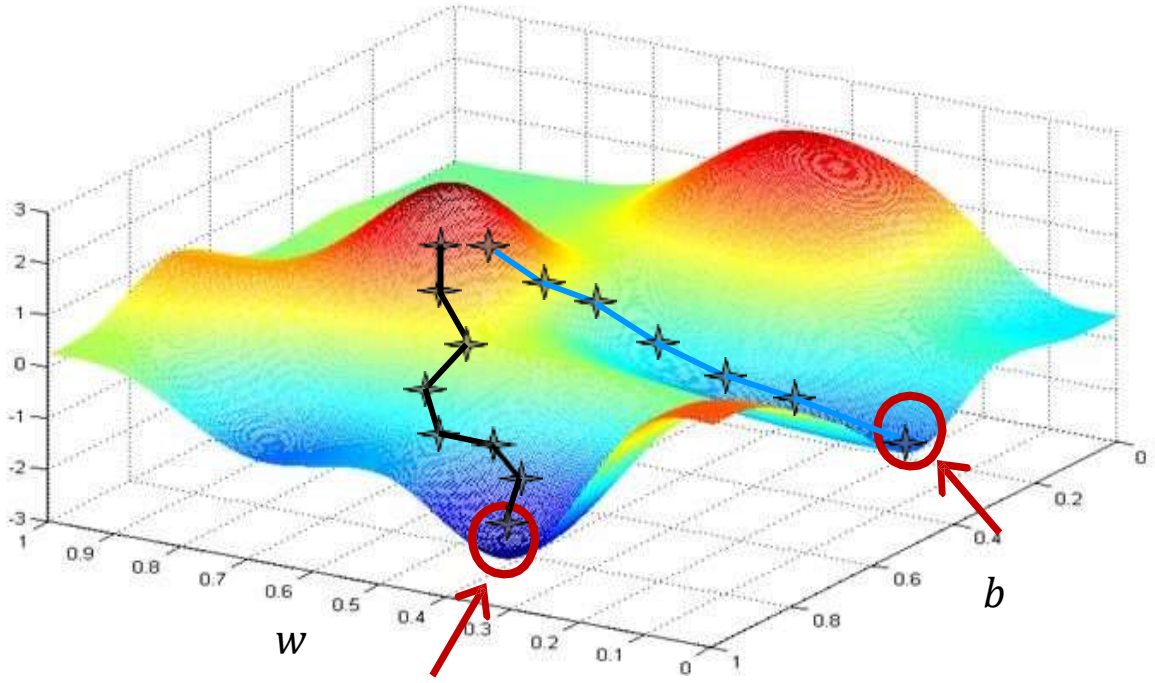
Update w and b simultaneously

$f_{w,b}(x^{(i)}) = wx^{(i)} + b$

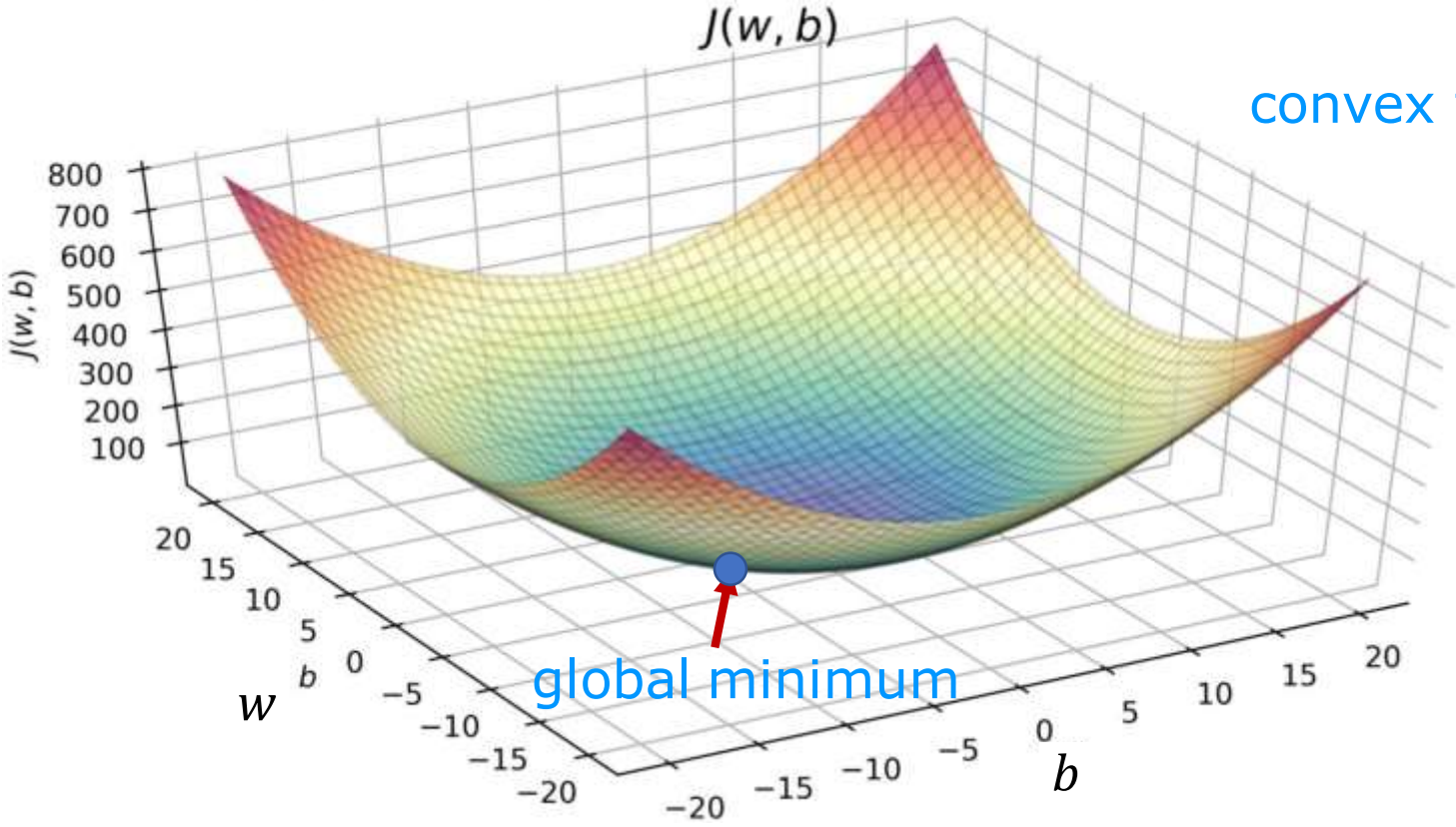
More than one local minimum

When we have more than one local minimum, then the minimum that the algorithm converges to depends on the value that we started at

$J(w, b)$



Squared Error Cost Function

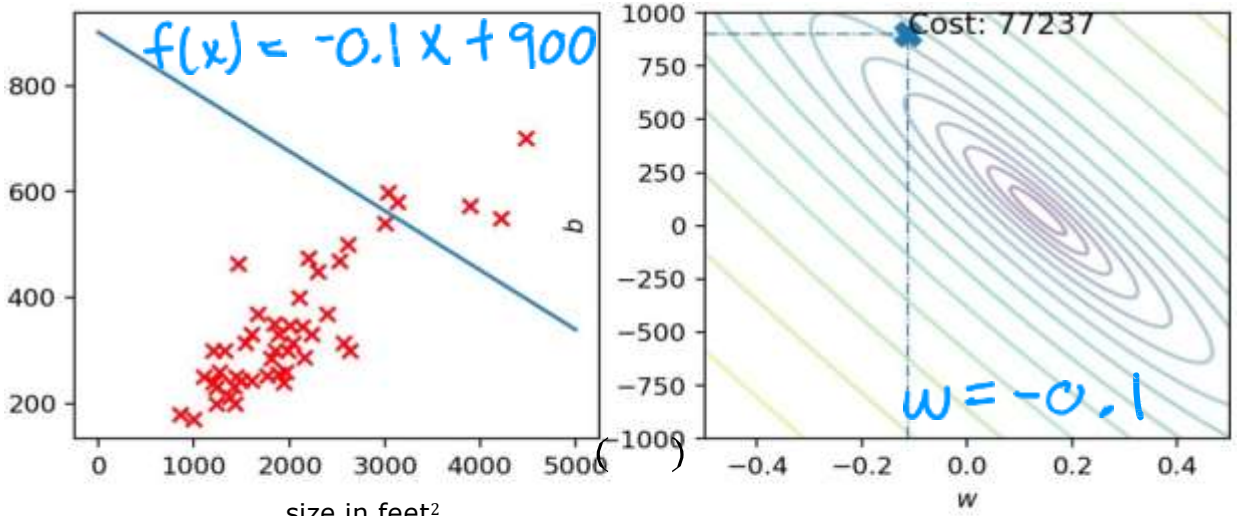


There is only one minimum for the squared error cost function since it is a convex function

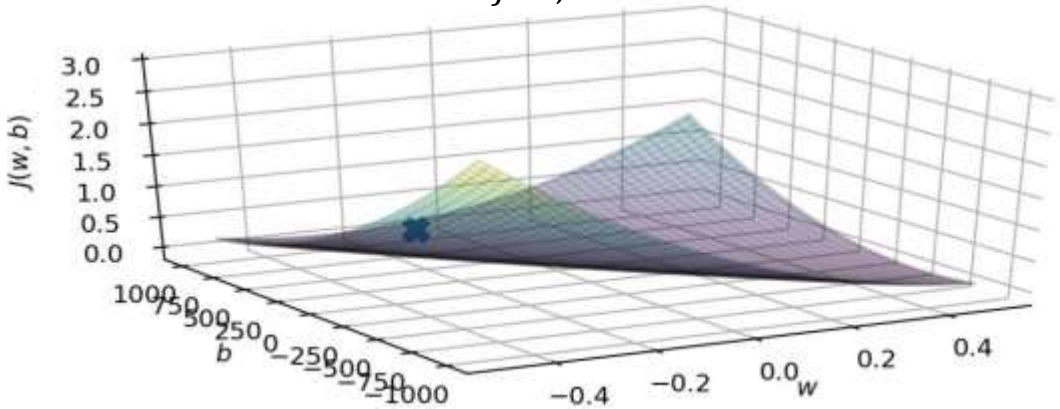
Running Gradient Descent

$f_{w,b}(size)$

$J(w,b)$

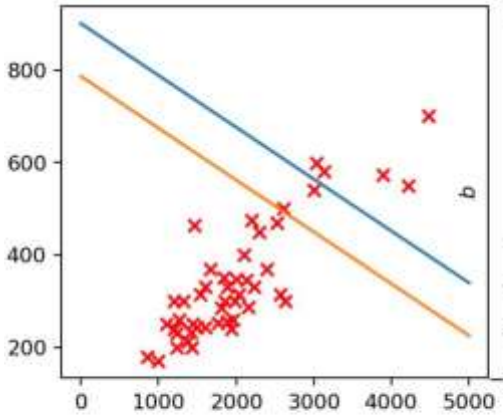


$J(w,b)$



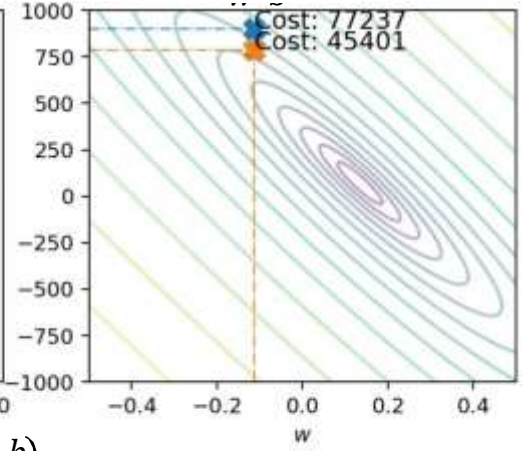
$f_{w,b}(size)$

price in $1000's

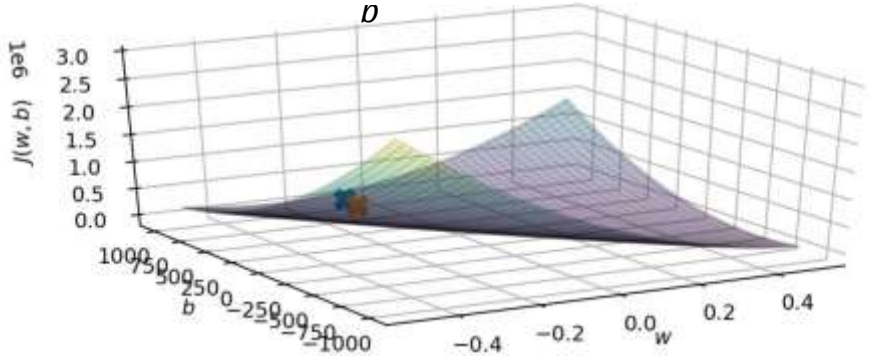


size in feet^2

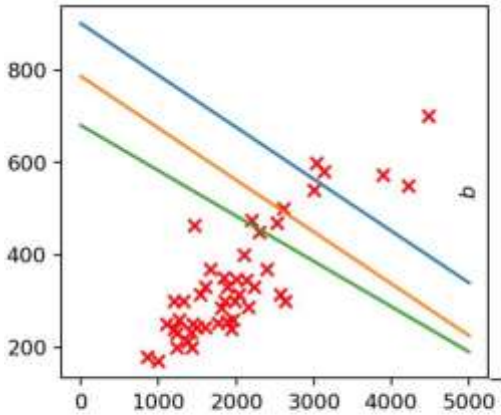
$J(w,b)$



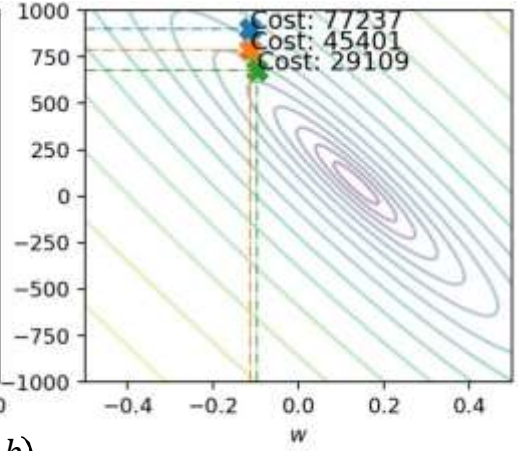
$J(w,b)$



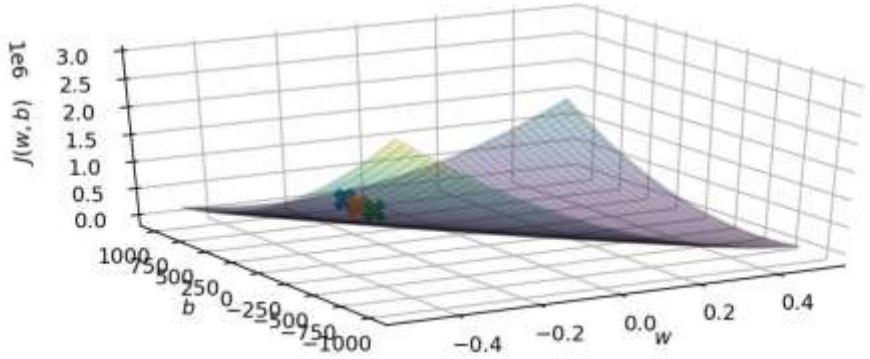
$f_{w,b}(size)$

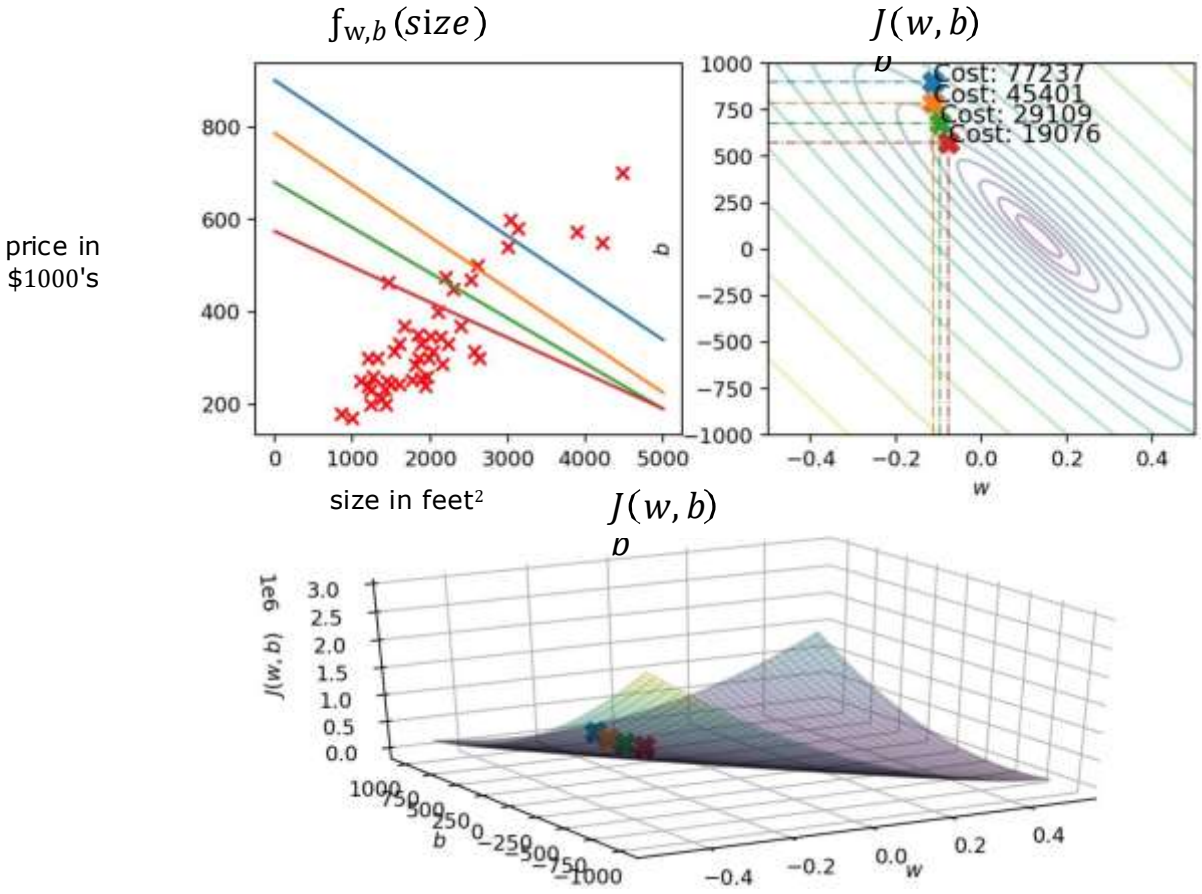


$J(w,b)$



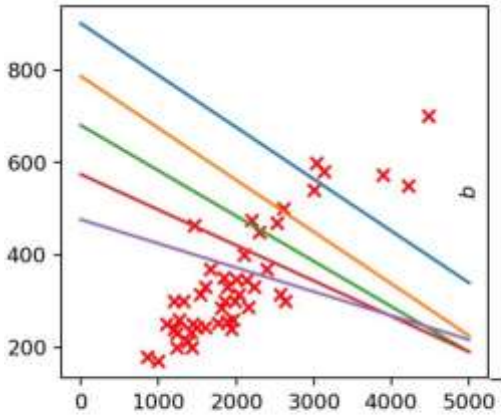
$J(w,b)$




$f_{w,b}(size)$
$J(w,b)$
price in $1000's
size in feet²
$J(w,b)$
Cost: 77237
Cost: 45401
Cost: 29109
Cost: 19076
b
w

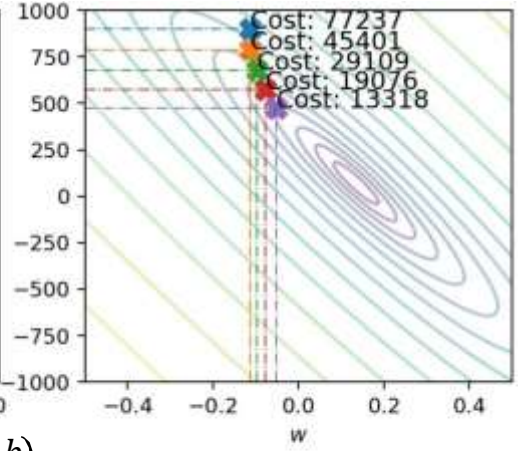
$f_{w,b}(size)$

price in $1000's

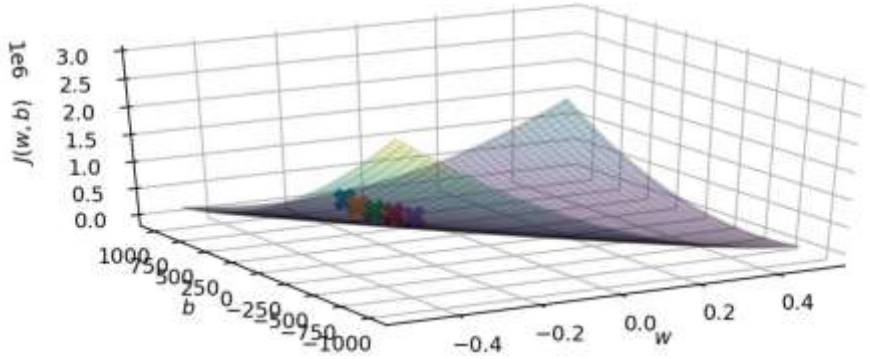


size in feet^2

$J(w,b)$

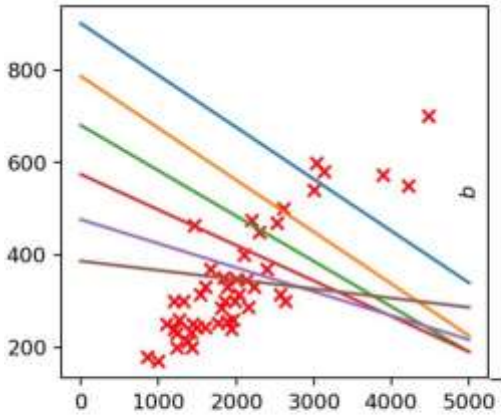


$J(w,b)$



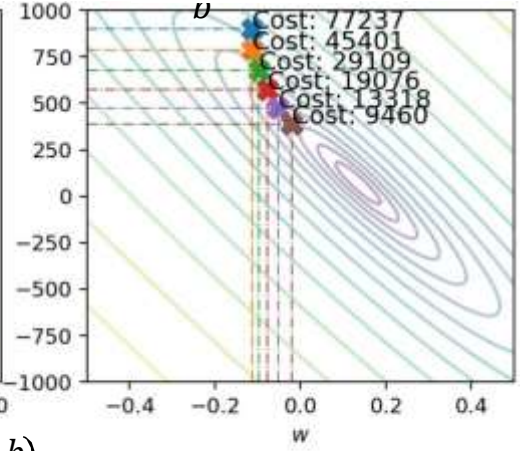
$f_{w,b}(size)$

price in $1000's

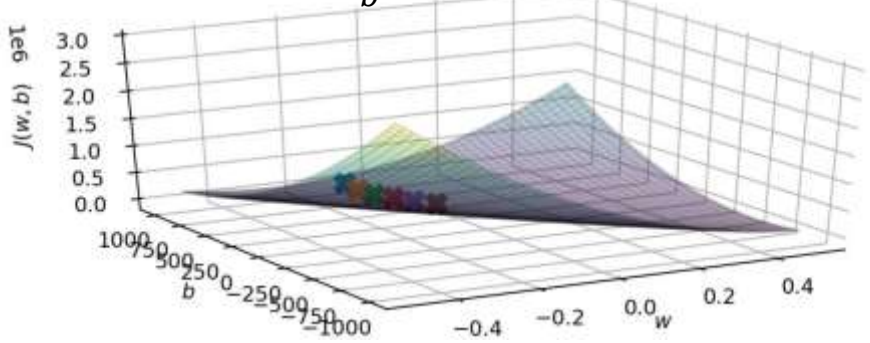


size in feet^2

$J(w,b)$

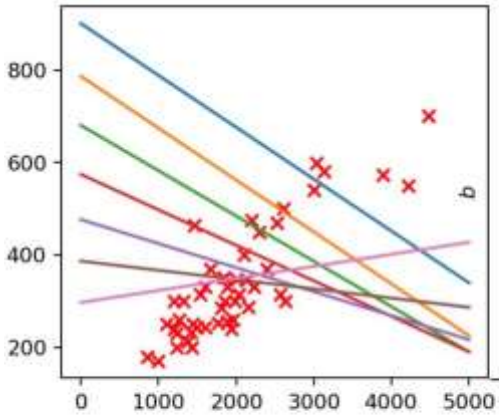


$J(w,b)$



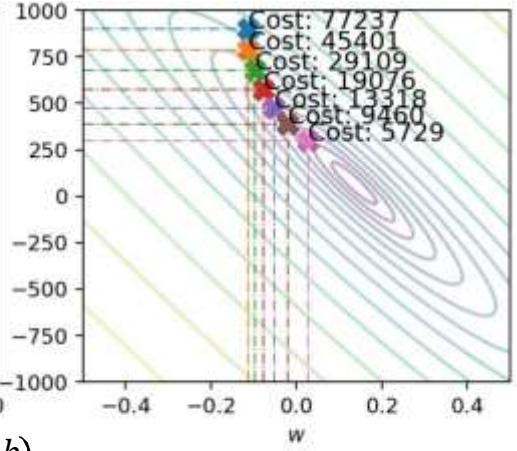
$f_{w,b}(size)$

price in $1000's

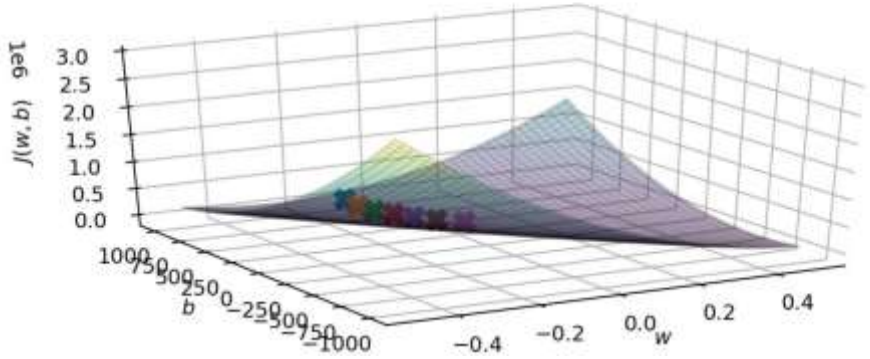


size in feet^2

$J(w,b)$

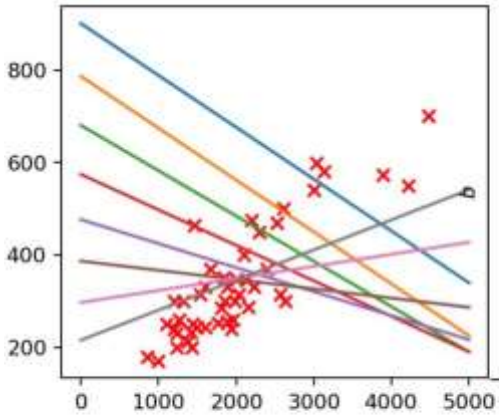


$J(w,b)$



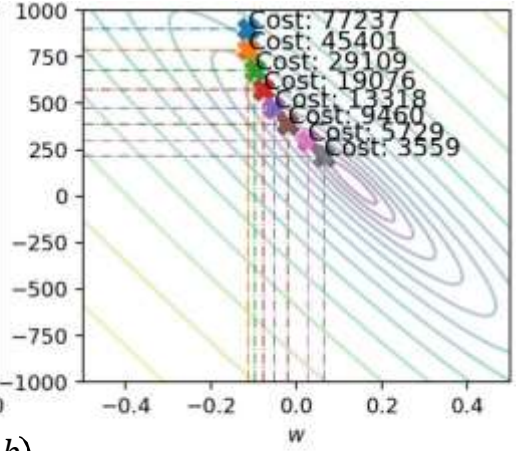
$f_{w,b}(size)$

price in $1000's

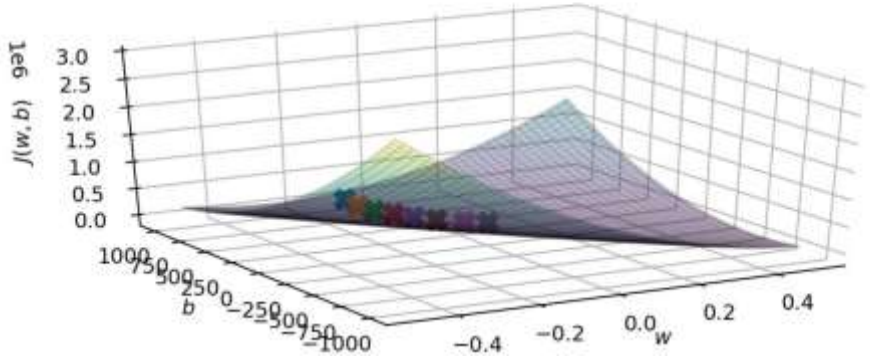


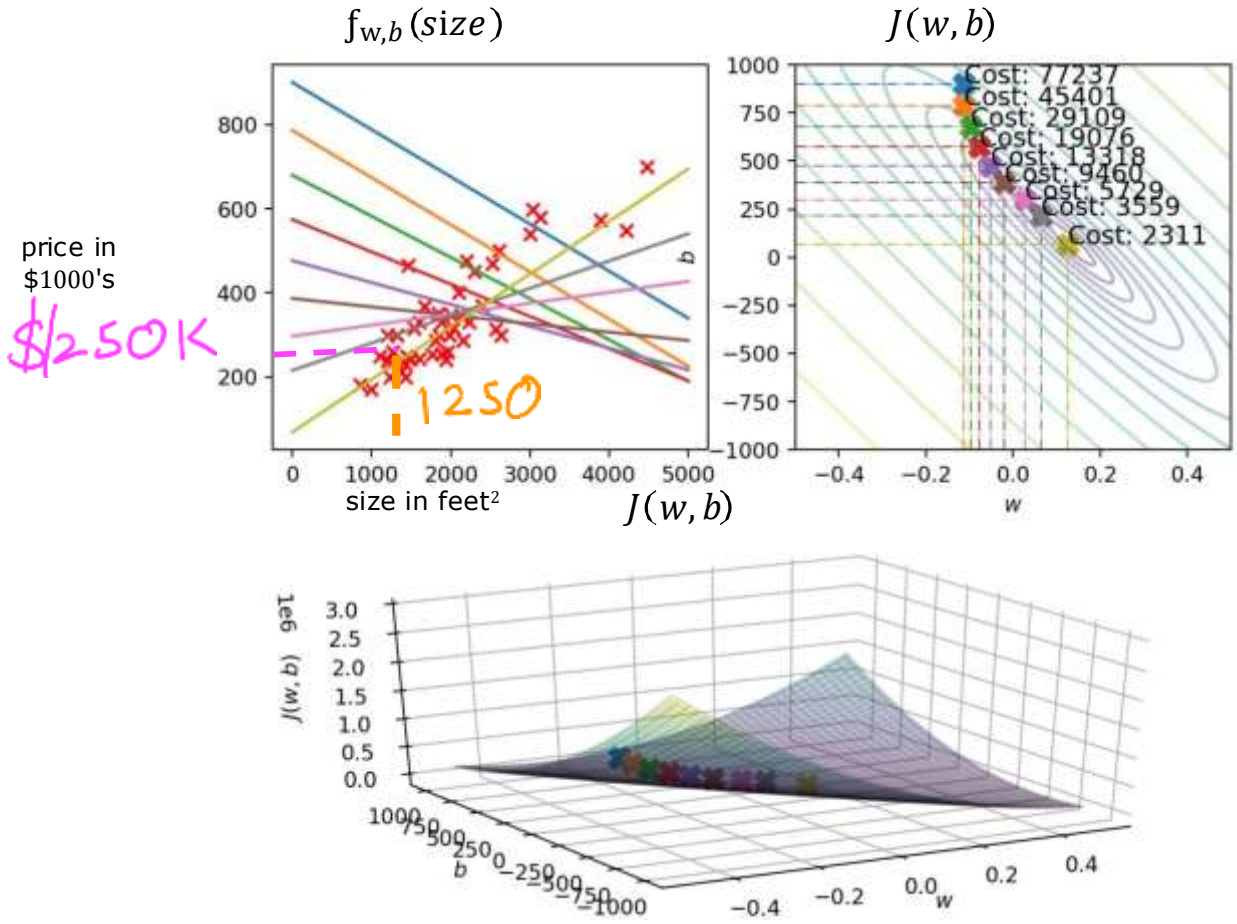
size in feet^2

$J(w,b)$



$J(w,b)$




$f_{w,b}(size)$
$J(w,b)$
price in $1000's
$250K
1250
size in feet²
$J(w,b)$
Cost: 77237
Cost: 45401
Cost: 29109
Cost: 19076
Cost: 13318
Cost: 9460
Cost: 5729
Cost: 3559
Cost: 2311
b
w
1e6
J(w,b)

"Batch" gradient descent

other gradient descent: subsets

"Batch": Each step of gradient descent uses all the training examples.

	x size in feet^2	y price in \$1000's
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
...	...	...
(47)	3210	870

m = 47

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