

Course 1: Supervised Machine Learning

These course notes were developed using lectures/material from the [ML Specialization](#) from Andrew Ng. The visuals and equations are adopted from the original slides. I simply hope that the notes serve as accompanying study material. Find all my notes for this course in the [ML Course Notes repo](#).

Introduction to ML

- [Overview of ML](#)
 - [Supervised vs. Unsupervised ML](#)
 - [Regression Modeling](#)
 - [Training with Gradient Descent](#)
-

Week 1 - Introduction to ML

Overview of ML

Machine Learning (ML) is a subfield of Artificial Intelligence (AI). It deals with training learning algorithms to deal with all sorts of predictive problems and tasks.

According to a study by [McKinsey](#), AI and machine learning is estimated to create an additional \$US 13 trillion of value annually by the year 2030.

ML is already creating major changes in the software sector. However, there is a lot of progress happening in automotive, transportation, retail, and other sectors as well.

Because of the massive untapped opportunities across many sectors, there is high unfulfilled demand for machine learning engineers and related skillsets. This course aims to teach the fundamentals of machine learning. It also emphasizes on how to put the knowledge into practice.

Supervised vs. Unsupervised ML

What is machine learning?

Arthur Samuel defined machine learning as the *"field of study that gives computers the ability to learn without being explicitly programmed."*

He trained a computer on tens of thousands games of checkers. The computer program got so good that it eventually was able to achieve sufficient skills to challenge a respectable amateur.

Here are the main types of machine learning algorithms:

- Supervised learning
- Unsupervised learning
- Recommender systems
- Reinforcement learning

Supervised learning (Part 1)

In supervised learning you train an algorithm given inputs X to predict output Y . The algorithm learns by given it the right answers.

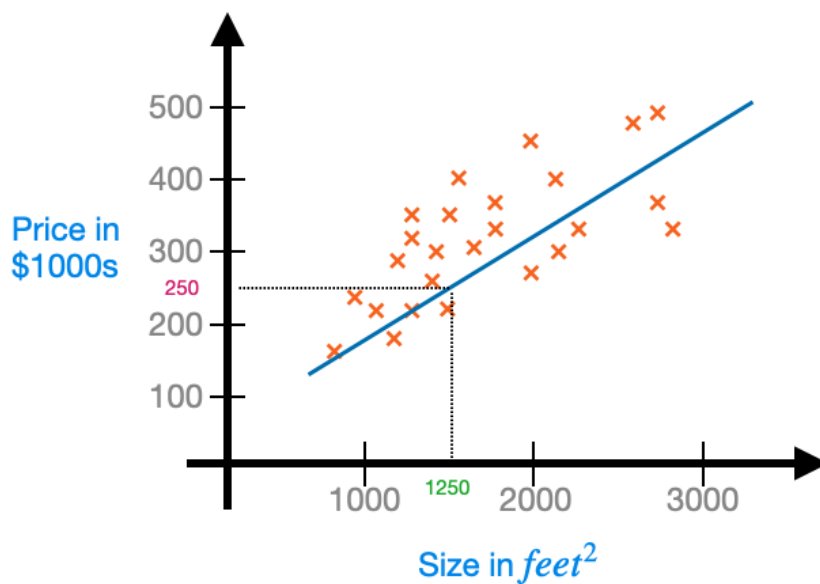
Below are some examples of applications where supervised learning is being used.

Applications	Input (X)	Output (Y)
Spam Filtering	Email	→ Spam (0/1)
Speech Recognition	Audio	→ Text transcripts
Machine Translation	English	→ Spanish
Online Advertising	Ad, User info	→ Click (0/1)
Self-driving Car	Image, radar info	→ Position of other cars
Visual Inspection	Image of phone	→ Defect (0/1)

Let's look at an example of a classical machine learning problem. For instance, let's use regression to tackle house price prediction:

- You have a dataset with prices of the houses and the house size in feet
- Now you would like to predict the price of the house given a new house size
- You can train a model to be able to predict that price
- The regression model (demonstrated in figure below) fits a line through the dataset points and that's used as the predictor
- You can fit different lines through the data which gives you different model behaviors and predictions
- The learning algorithm we are using is called regression, where the goal is to predict a number from infinitely many possible outputs (e.g., house prices)

Below is an example of a model fitting a line through the data points which is used as the predictor:



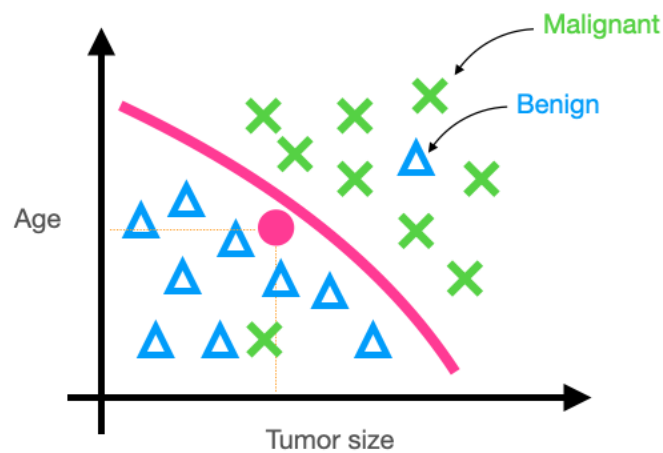
Supervised Learning (Part 2)

Another type of problem that involves supervised learning is classification. The difference compared to regression is that in classification we are trying to predict a smaller number of outputs or categories.

As an example, let's think about a classification model that's trained to perform breast cancer detection. The two possible outputs are **malignant** and **benign**. A dataset for this could include tumor size and the diagnosis.

Classification models predict categories and they don't have to be numeric.

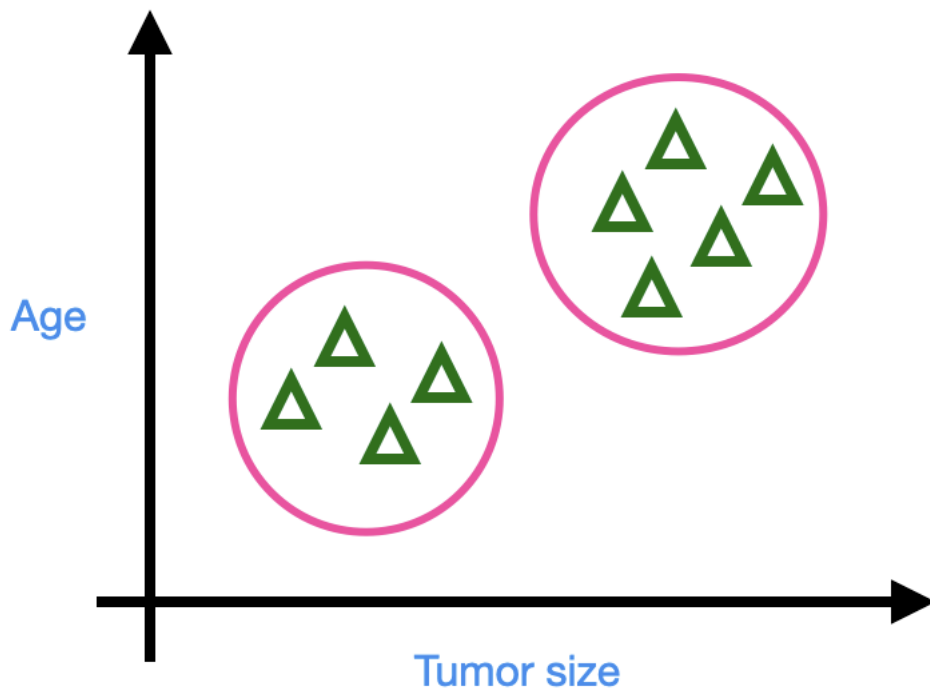
In classification, you can also have two or more inputs. For instance, below we are using age and tumor size. The learning algorithm fits a *boundary line* through the data which helps to output the final diagnosis.



Unsupervised Learning (Part 1)

The most widely used form of machine learning after supervised learning is unsupervised learning. When dealing with unsupervised learning you are dealing with data that isn't associated with any output labels y . As an example, imagine you were given data with *age* and *tumor size* without any indicator of whether the tumor was benign or malignant.

The task with unsupervised learning is basically to find some structure in the data or something interesting without requiring those labels upfront. So in the example below an algorithm would be able to detect two clusters of data points that could represent two different groups. That algorithm is referred to as clustering.



Some examples of unsupervised learning are as follows:

- Google News clustering the daily news by topics
- Clustering genetic or DNA data
- Grouping of customer profiles

Unsupervised Learning (Part 2)

A more formal definition of unsupervised learning is as follows: Data only comes with inputs x , but not output labels y . The algorithm is then designed with the goal to find structure in the data provided.

One example of unsupervised learning is called **clustering**, where the goal is to group similar data points together.

Another example is **anomaly detection**, where the goal is to find unusual data points such as unusual events in financial transactions.

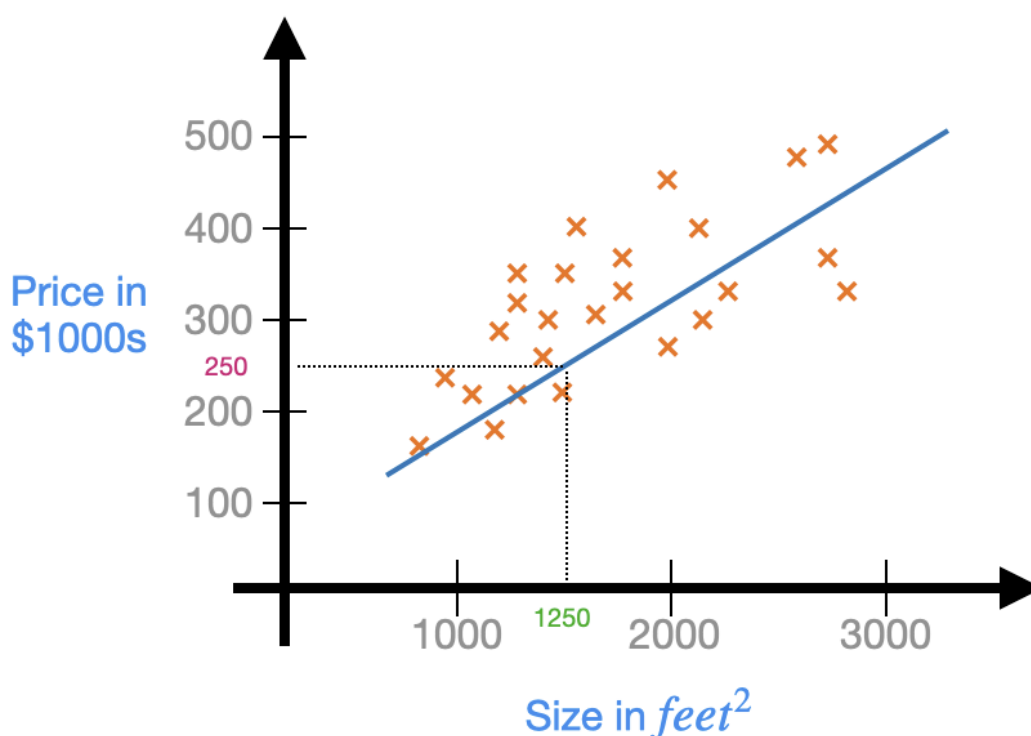
Another popular form of unsupervised learning is **dimensionality reduction**. The goal here is to compress the data using fewer dimensions without losing too much information.

Regression Model

Linear Regression Model (Part 1)

To better understand linear regression we can take a closer look at the example of a model that predicts house prices. Let's say you were given a dataset with **size** and **prices**. You can build a model based on linear regression — this model fits a line through the data.

If we look at the figure below, we can see the data points and the line that was fitted. Given that, we can see that for new house size **1250**, the model will predict something like **\$250K**.



This regression model predicts numbers and it is a form of supervised learning.

Some key terminology to understand:

- Training set: the data used to train the model
- Input variable: represents the input variable/feature and is usually denoted as x
- Target variable: represents the target variable and is usually denoted as y
- m is the number of training examples
- (x, y) is the a single training example
- $(x^{(i)}, y^{(i)})$ represents the $i^{(th)}$ training example

Below is an example of a sample dataset containing housing prices and size in feet squared:

	size in feet (x)	price in \$1000's (y)
1	2104	400
2	1416	232
3	1534	315
4	852	178
...	...	...
50	3210	870

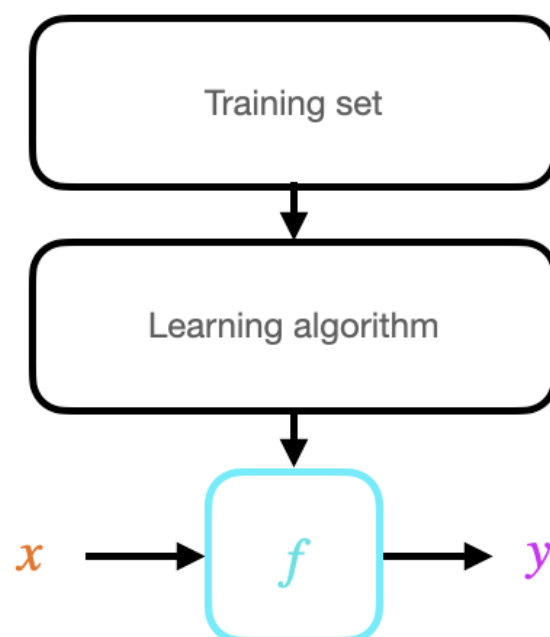
where $x^{(1)} = 2104$ and $y^{(1)} = 400$. Therefore, $(x^{(1)}, y^{(1)}) = (2104, 400)$.

Linear Regression Model (Part 2)

When designing a linear regression model the following are the important components:

- First, you have a training set which represents your features and target variables
- You feed that through a learning algorithm which can include a model (represented by a function f)

- x data is fed through the model which outputs the prediction or estimated value $\hat{y}$
- f can be represented as a linear function $f(x) = wx + b$
- w and b are called the parameters of the model
- When you are dealing with one single feature x in linear regression it is referred to as univariate linear regression



Cost Function Formula

Recall that for linear regression the model function is defined as follows:

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

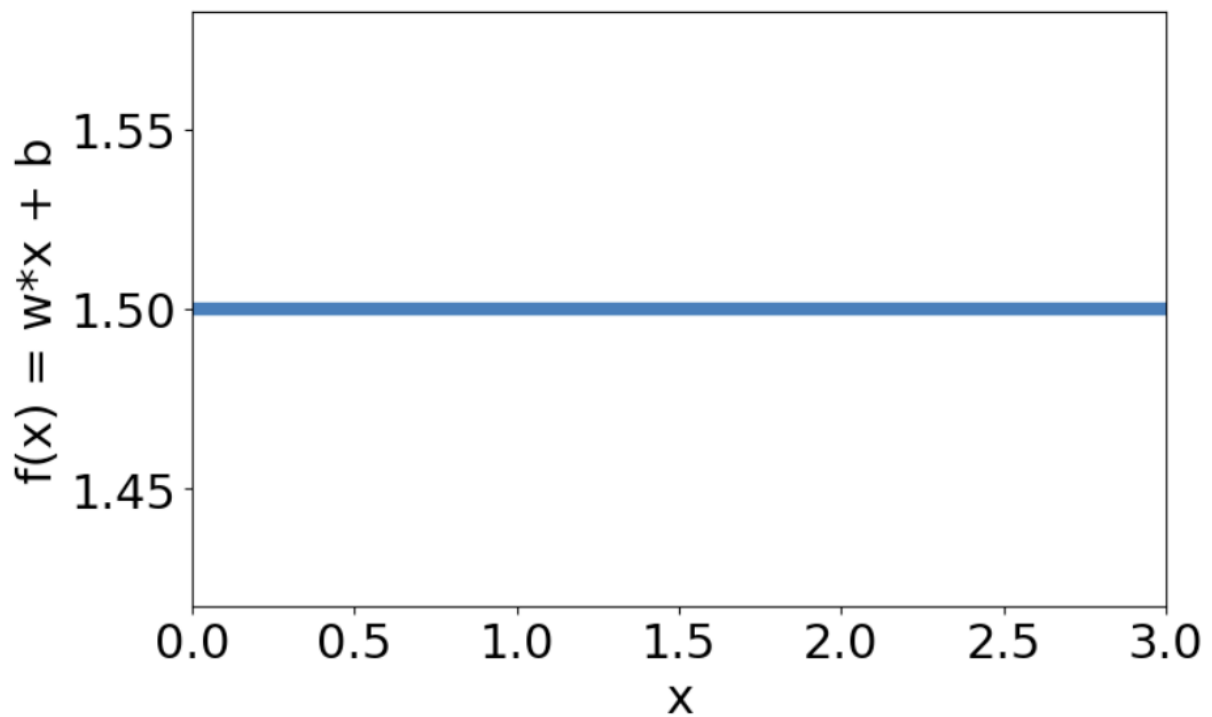
where w and b are the parameters of the model.

To demonstrate how the parameters determine the function f we can take a look at some plots in a chart.

Example 1:

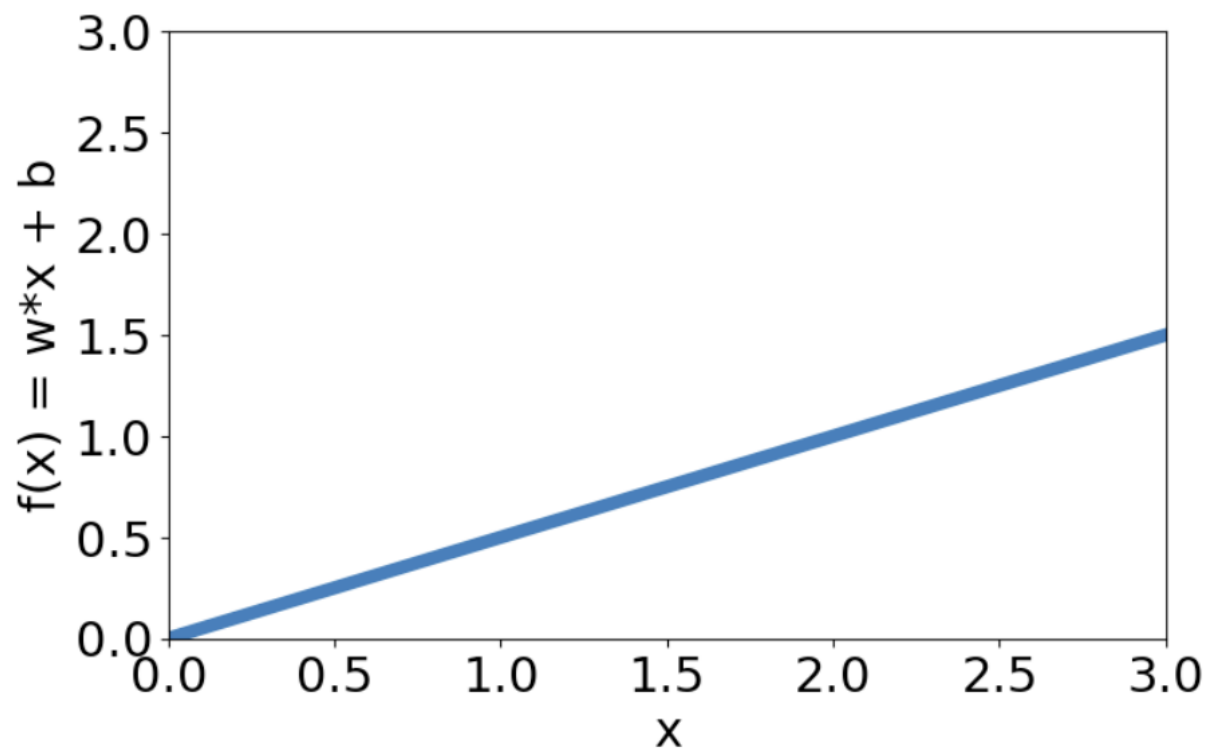
- $w = 0$
- $b = 1.5$

Given $f(x) = wx + b$, the plot looks as follows:

**Example 2:**

- $w = 0.5$
- $b = 0$

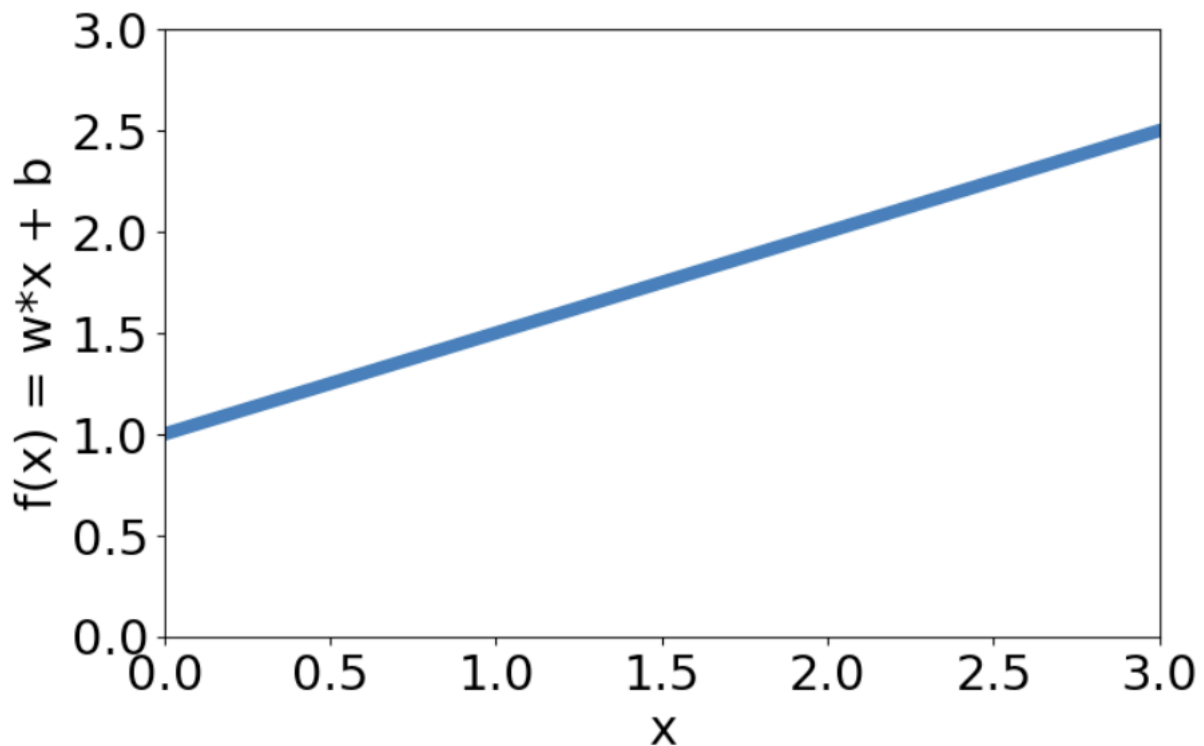
Given $f(x) = wx + b$, the plot looks as follows:



Example 3:

- $w = 0.5$
- $b = 1$

Given $f(x) = wx + b$, the plot looks as follows:



The function f is important because it gives us the prediction $\hat{y}$. We then take that prediction and compare with the original target to compute what's referred to as the **cost function**. The purpose of the cost function is to allow the model to optimize for values of w and b that reduces the errors of the model. In other words, find w and b where $\hat{y}^{(i)}$ is close to $y^{(i)}$ for all $(x^{(i)}, y^{(i)})$.

A widely used cost function in linear regression is known as **squared error cost function** and is defined as follows:

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

where m is the number of training examples and $f_{w,b}(x^{(i)}) = \hat{y}^{(i)}$.

Cost Function Intuition

Recall that to measure how well the choice of the parameters w and b fits the training data, you define the cost function J . J measures the difference between the model's predictions and the actual true values for y . So the overall goal is to minimize J as a function of w and b . This is defined as:

$$\underset{w,b}{\text{minimize}} J(w, b)$$

To get a better intuition of the cost function, let's take a look at an example and plot it.

To do this, we can simplify our prediction function f to be:

$$f_w(x) = wx$$

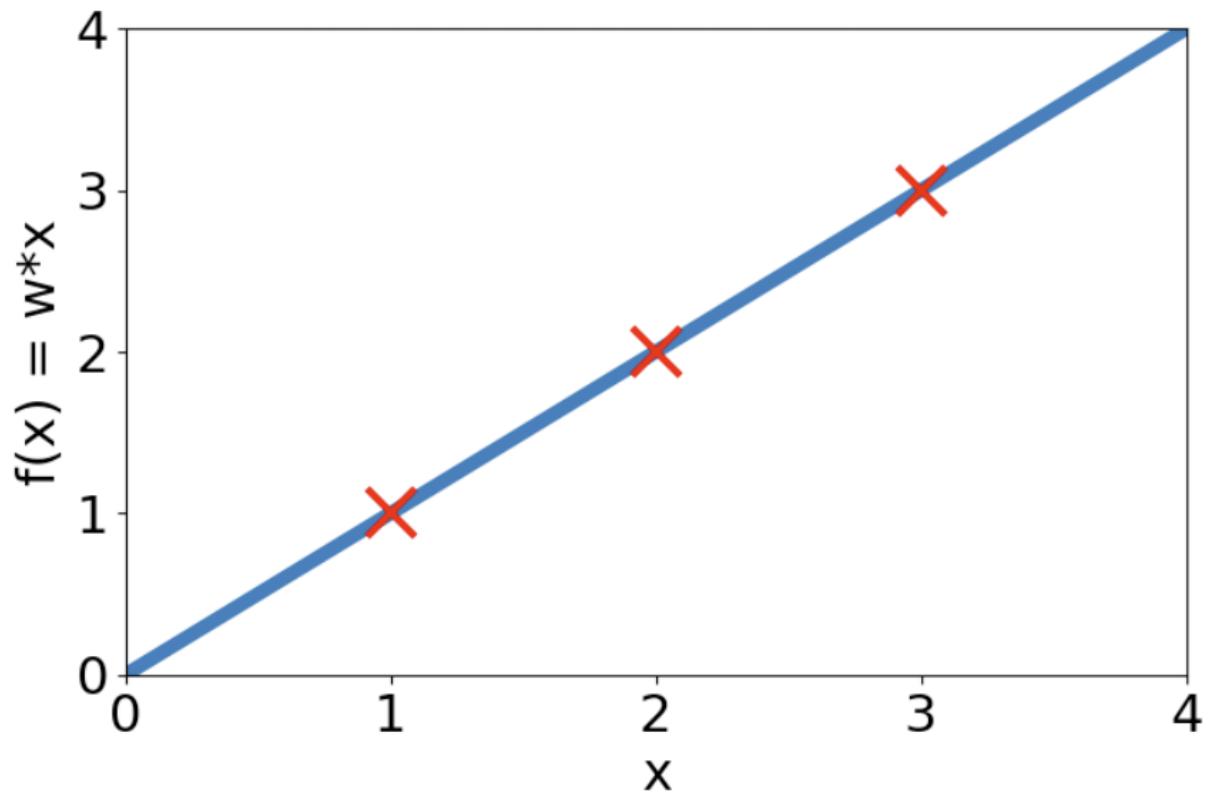
We have set $b = 0$. So this gives us the following cost function:

$$J(w) = \frac{1}{2m} \sum_{i=1}^m \left(f_w(x^{(i)}) - y^{(i)} \right)^2$$

Now we want to minimize the following:

$$\underset{w}{\text{minimize}} J(w)$$

As an example, let's plot what a minimal prediction function looks like when we plot examples on a chart and $w = 1$:



So now we want to calculate the cost J when $w = 1$. Recall the cost function is defined as follows:

$$J(w) = \frac{1}{2m} \sum_{i=1}^m \left(f_w(x^{(i)}) - y^{(i)} \right)^2$$

With substitution we end up with:

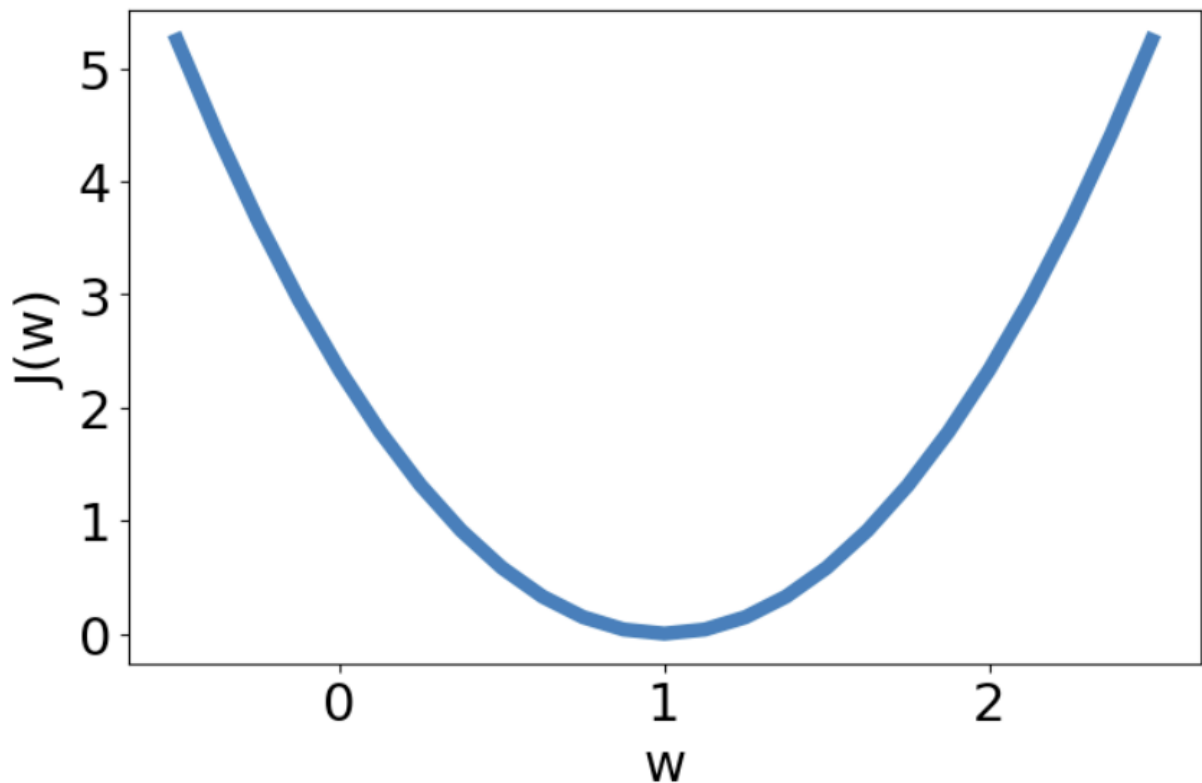
$$J(w) = \frac{1}{2m} \sum_{i=1}^m \left(wx^{(i)} - y^{(i)} \right)^2$$

If you look at the example above, we plotted the following three data points representing value pairs of x and y : `[1, 1], [2, 2], [3, 3]`

If we replace the values in the cost function J we end up with:

$$J(w) = \frac{1}{2m} (0^2 + 0^2 + 0^2)$$

Feel free to work that out in details. Given those calculations we can now plot the function J with respect to the w parameter. Let's try to use many different values of w given the same data points as before. The plot will look something like what's shown below:



Use the code below to plot the function:

```
import torch
import numpy as np

def plot_cost_function(w, m, x, y):
    j_w = [ (1/(2*m)) * torch.sum((w * x) - y)**2) for w in w ]
    plt.plot(w, j_w, '-', linewidth=6)

    plt.xlabel('w')
    plt.ylabel('J(w)')
    plt.show()
    return
```

```
w = torch.tensor([x for x in np.linspace(-0.5, 2.5, 25)], dtype=float)
x = torch.tensor([1.0, 2.0, 3.0], dtype=float)
y = torch.tensor([1.0, 2.0, 3.0], dtype=float)
m = 3.0

plot_cost_function(w, m, x, y)
```

One interesting observation you will note here is that when $w = 1$ we saw a straight line through the data points in our previous chart, which means that a perfect line was fitted through the data points. In fact, when $w = 1$ it produces $J(w) = 0$. So in this problem we want to choose w that minimizes $J(w)$, which in this example turns out to be $w = 1$.

So far, we have the following definitions:

Model:

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

Parameters:

$$w, b$$

Cost Function:

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

Learning objective:

$$\underset{w,b}{\text{minimize}} J(w, b)$$

You can also use this general form to visualize the cost function. However, it does require a bit more effort and different tools. The course has a great [video](#)

[explainer](#) and visual examples of how the cost function looks. It uses contour plots which are a popular way to visualize the cost function.

Train the model with Gradient Descent

Gradient Descent Overview

Gradient descent is one of the most important and widely used algorithms to train machine learning and deep learning models.

The gradient descent algorithm is a very simple concept. Let's say we have some function $J(w, b)$ we want to minimize. We start with w, b , which you can initialize to zero. You then keep changing w, b to reduce $J(w, b)$ until the algorithm settles as or gets near a minimum.

In the gradient descent algorithm, you can adjust the weight by using the following update formula:

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

where α is the learning rate which controls how big of a step you take in the gradient descent procedure. $\frac{\partial}{\partial w} J(w, b)$ defines in which direction you want to take the step. You do the same assignment procedure on the b parameter:

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

So the algorithm updates those parameters until convergence where w and b don't change much.

Gradient Descent Intuition

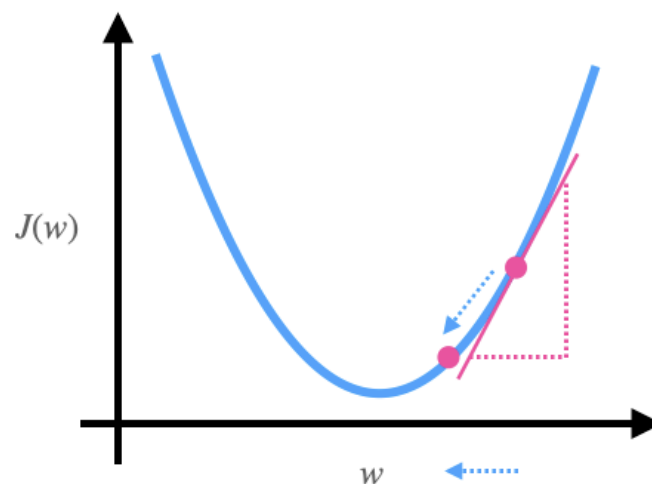
To get a better intuition about what the updates above do and how it helps the algorithm converge, we can take a look at the simplified version of the parameter update on w .

Given:

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

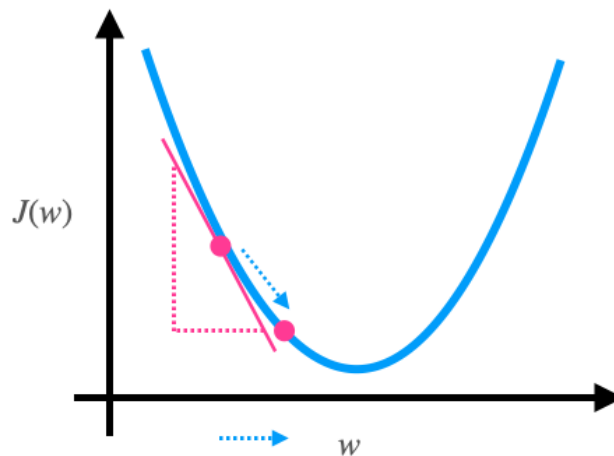
If $\frac{\partial}{\partial w} J(w) > 0$, i.e., a positive number, then w gets smaller as illustrated below:

$$w = w - \alpha \cdot (\text{positive number})$$



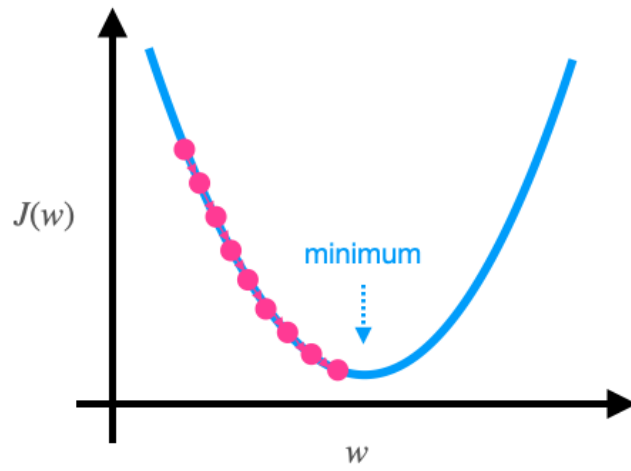
If $\frac{\partial}{\partial w} J(w) < 0$, i.e., a negative number then w increases as illustrated below:

$$w = w - \alpha \cdot (\text{negative number})$$

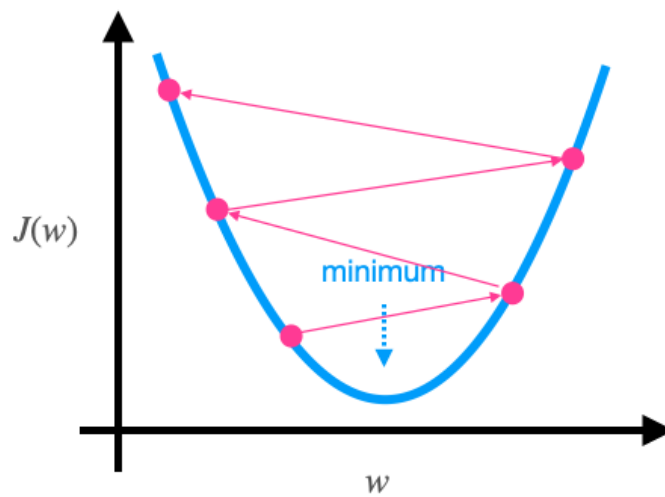


Another important consideration when using the gradient descent algorithm is to choose the right value for the learning rate. A poor choice of α might make the algorithm inefficient. If α is too small, then the steps taken by the algorithm will also be small which makes the algorithm slow. If α is too large, the algorithm could be very close to the minimum and miss it entirely because of a big step taken. With too large learning rates the algorithms runs the risk of overshooting and never reach minimum.

The illustration below shows that if α is too small, gradient descent may be too slow.

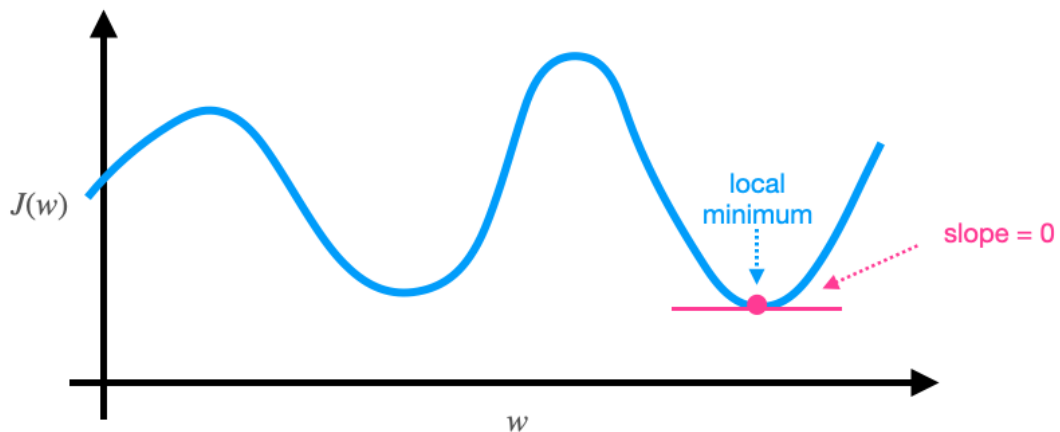


On the other hand, if α is too large, gradient descent may overshoot, never reach minimum or fail to converge or even diverge:



Note that if you are already at a local minimum the parameters won't change.

When $\frac{\partial}{\partial w} J(w) = 0$, w remains unchanged.



As we approach a local minimum, the derivative value gets closer to zero. With the gradient descent algorithm, update steps become smaller until you reach that local minimum even with a fixed learning rate.

To be continued...