The Perceptron

A perceptron takes several binary inputs, x1,x2,x3,.....and produces a single binary output:

x_1

x_2 → output

x_3
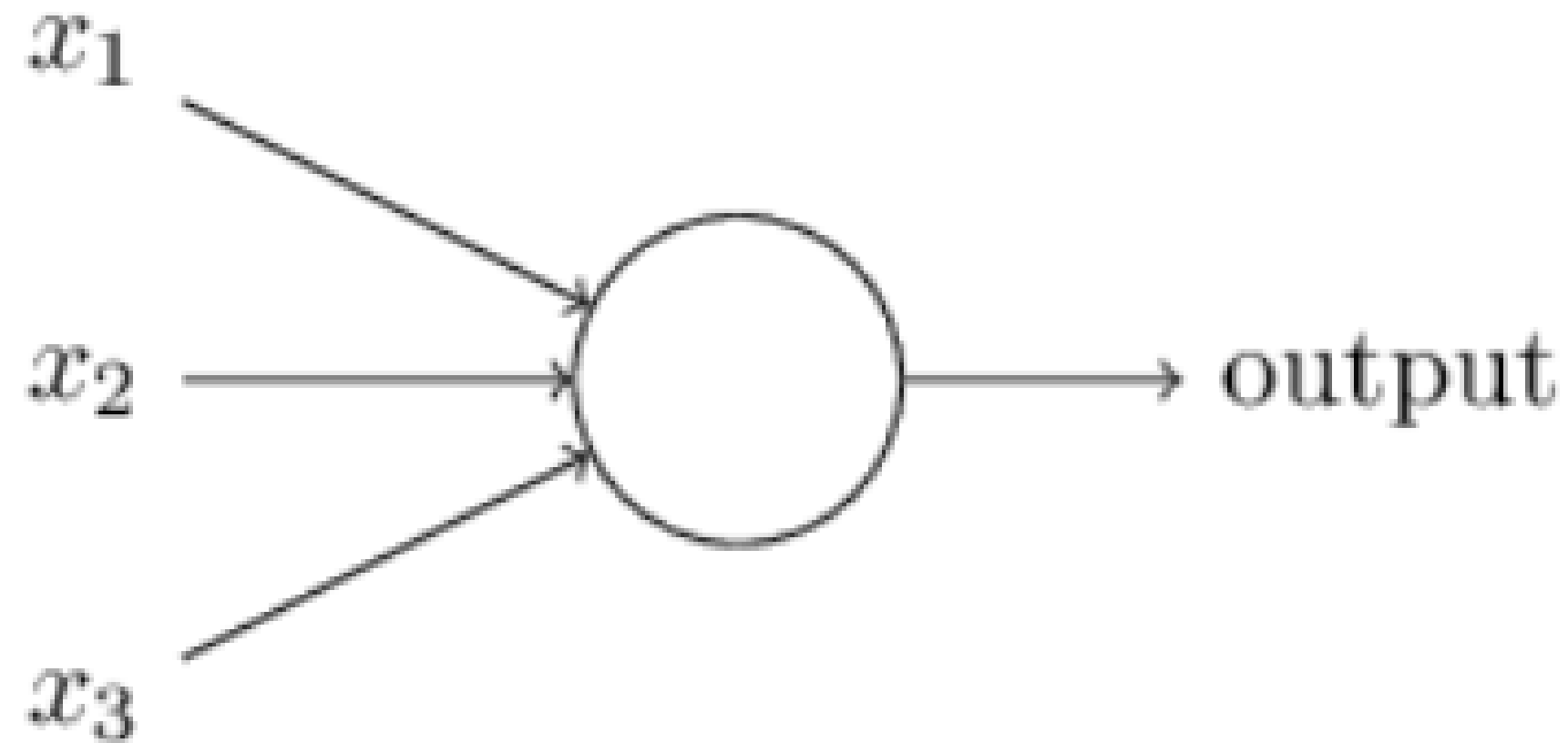
How does it make a "decision"?

Say you want to go to *Z-Square*, you weigh the importance of some factors before making a decision.

1. Is there a quiz coming up?
2. Is there an assignment?
3. Is the weather good?

You may assign different "weights" or importance to each of the factors to make a decision. A quiz the next day may have a higher weight than a bad weather.

So, say the relative weights are 0.7, 0.1, 0.1.

Say, you have a quiz, no assignments and the weather is good. You apply the perceptron as follows: - 0.7(1) + 0.1(0) + 0.1(1)

Meaning, you would rather go out if you donot have a quiz or assignments, leading to a lower weight. Similarly you would rather have good weather to go out.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

We need to set a threshold, if the output of the perceptron is lesser than it, we output a 0 and vice versa.
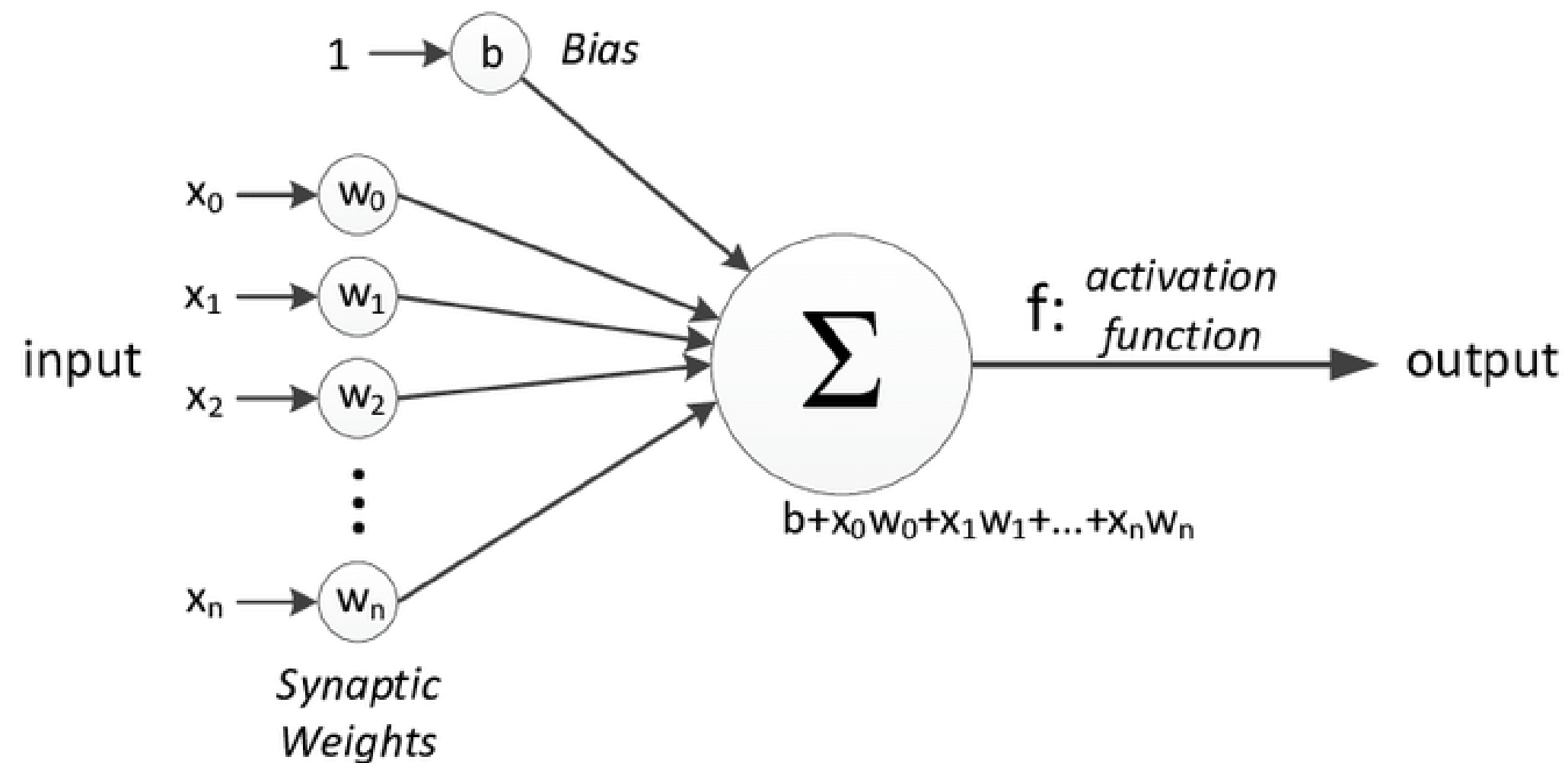
A better representaion is using the dot product w.x, and using a bias term "b"
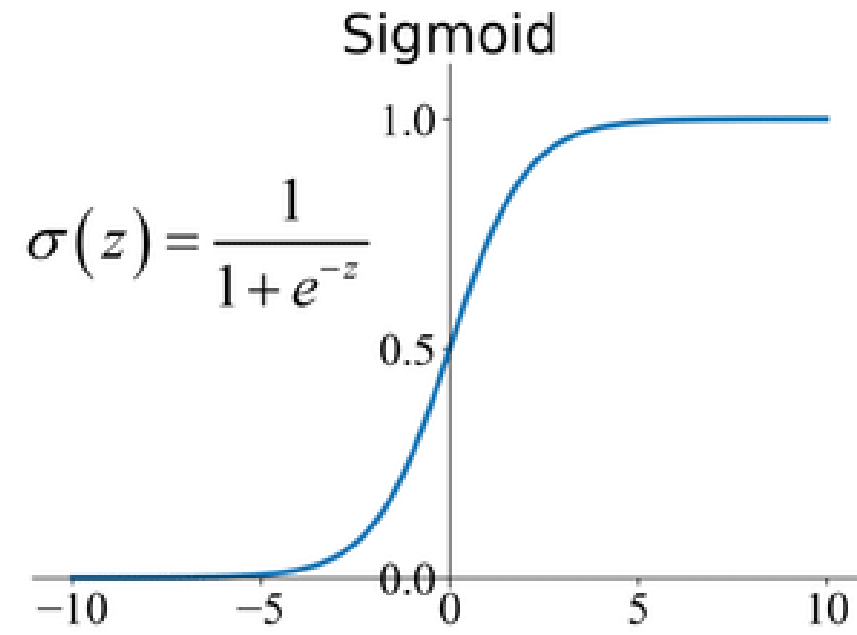
Here, w and x are represented as vectors

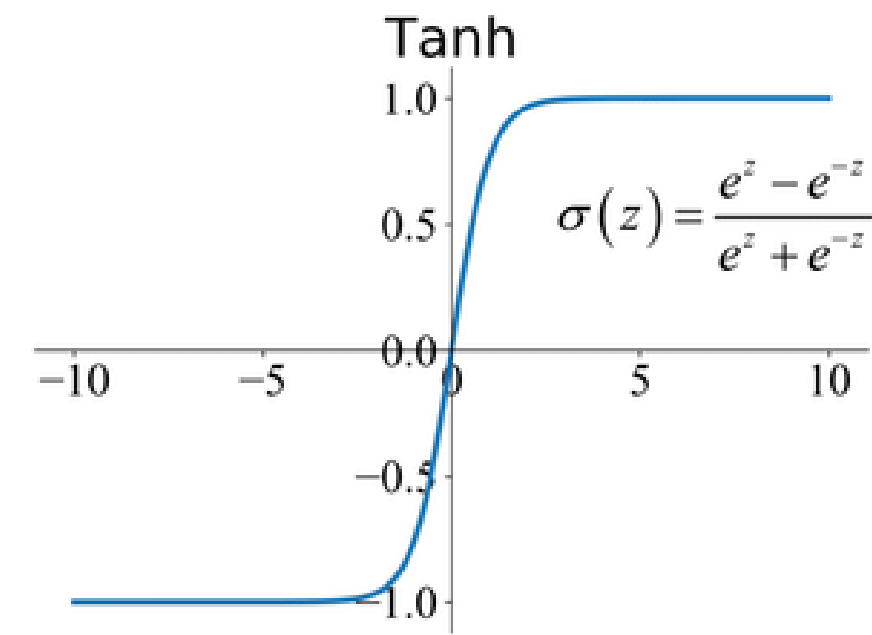$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Activation Functions

Activation functions helps us add non-linearity to the output by 1 perceptron. Each perceptron function on the basis of w.x + b, but if we add an extra step of non-linearity, the model can capture complex patterns in our dataset.
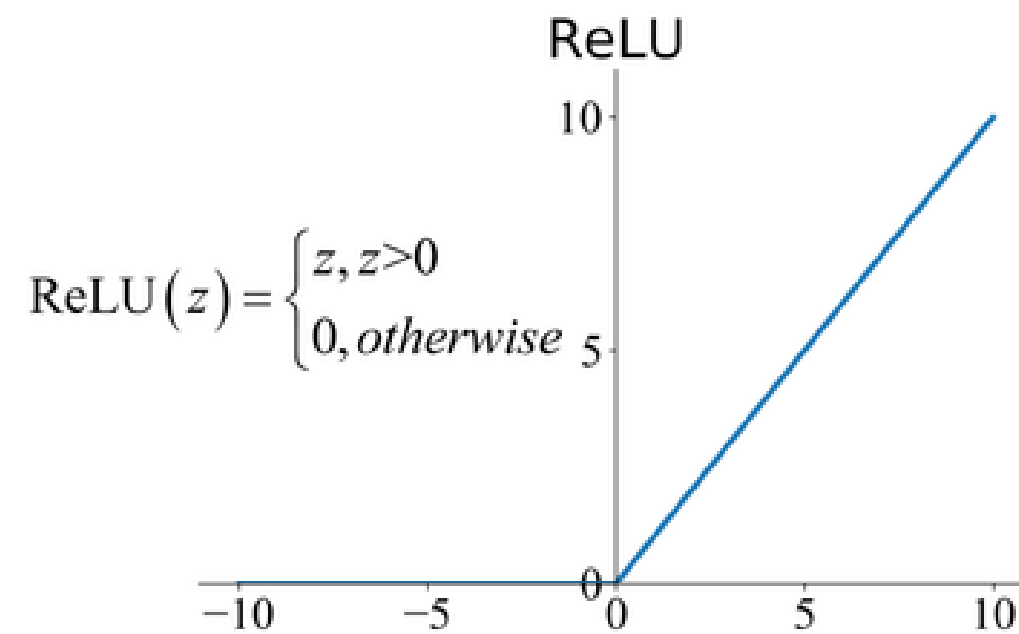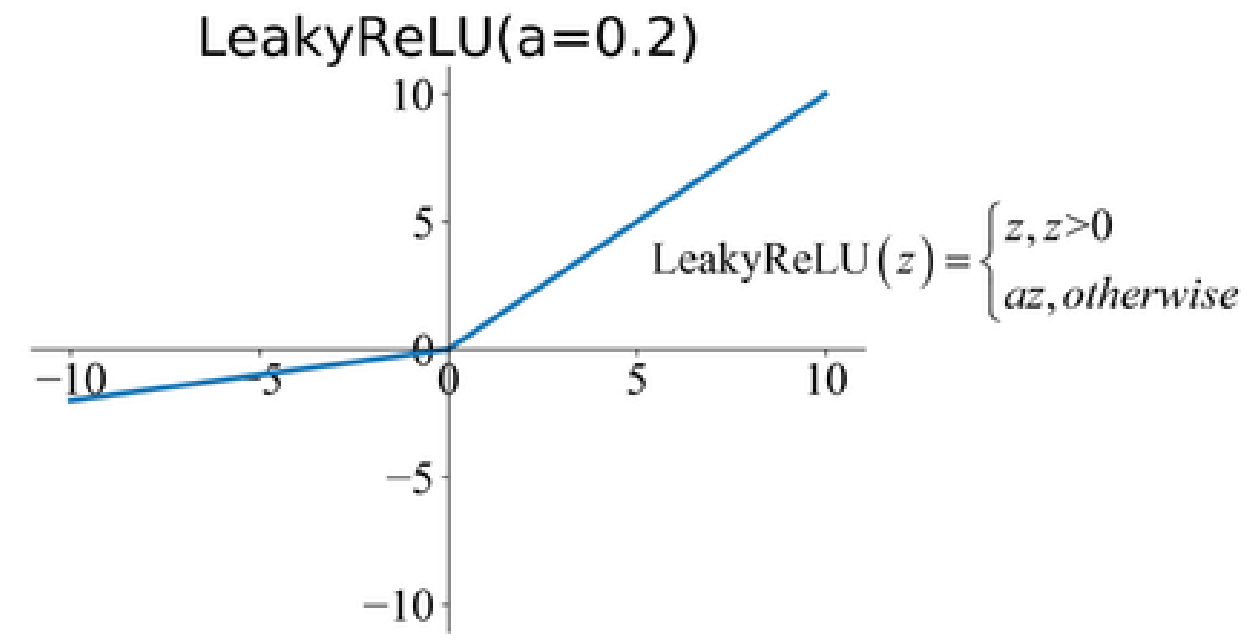
Sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

(a)

Tanh

$$\sigma(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

(b)

ReLU

$$ReLU(z) = \begin{cases} z, z > 0 \\ 0, otherwise \end{cases}$$

(c)

LeakyReLU(a=0.2)

$$LeakyReLU(z) = \begin{cases} z, z > 0 \\ az, otherwise \end{cases}$$

(d)

X is a vector of dimensions (num_samples,n_features). W is a column vector of dimensions (num_features,1). B is a real number. So equation is y = f(X.W + b).

Supervised Learning -
Linear Regression

We have a collection of labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where N is the size of the collection, $\mathbf{x}_i$ is the D-dimensional feature vector of example $i = 1, \ldots, N$, y_i is a real-valued[1] target and every feature $x_i^{(j)}$, $j = 1, \ldots, D$, is also a real number.

We want to build a model $f_{\mathbf{w},b}(\mathbf{x})$ as a linear combination of features of example $\mathbf{x}$:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b, \tag{1}$$

where $\mathbf{w}$ is a D-dimensional vector of parameters and b is a real number. The notation $f_{\mathbf{w},b}$ means that the model f is parametrized by two values: $\mathbf{w}$ and b.

We will use the model to predict the unknown y for a given $\mathbf{x}$ like this: $y \leftarrow f_{\mathbf{w},b}(\mathbf{x})$. Two models parametrized by two different pairs $(\mathbf{w}, b)$ will likely produce two different predictions when applied to the same example. We want to find the optimal values $(\mathbf{w}^*, b^*)$. Obviously, the optimal values of parameters define the model that makes the most accurate predictions.
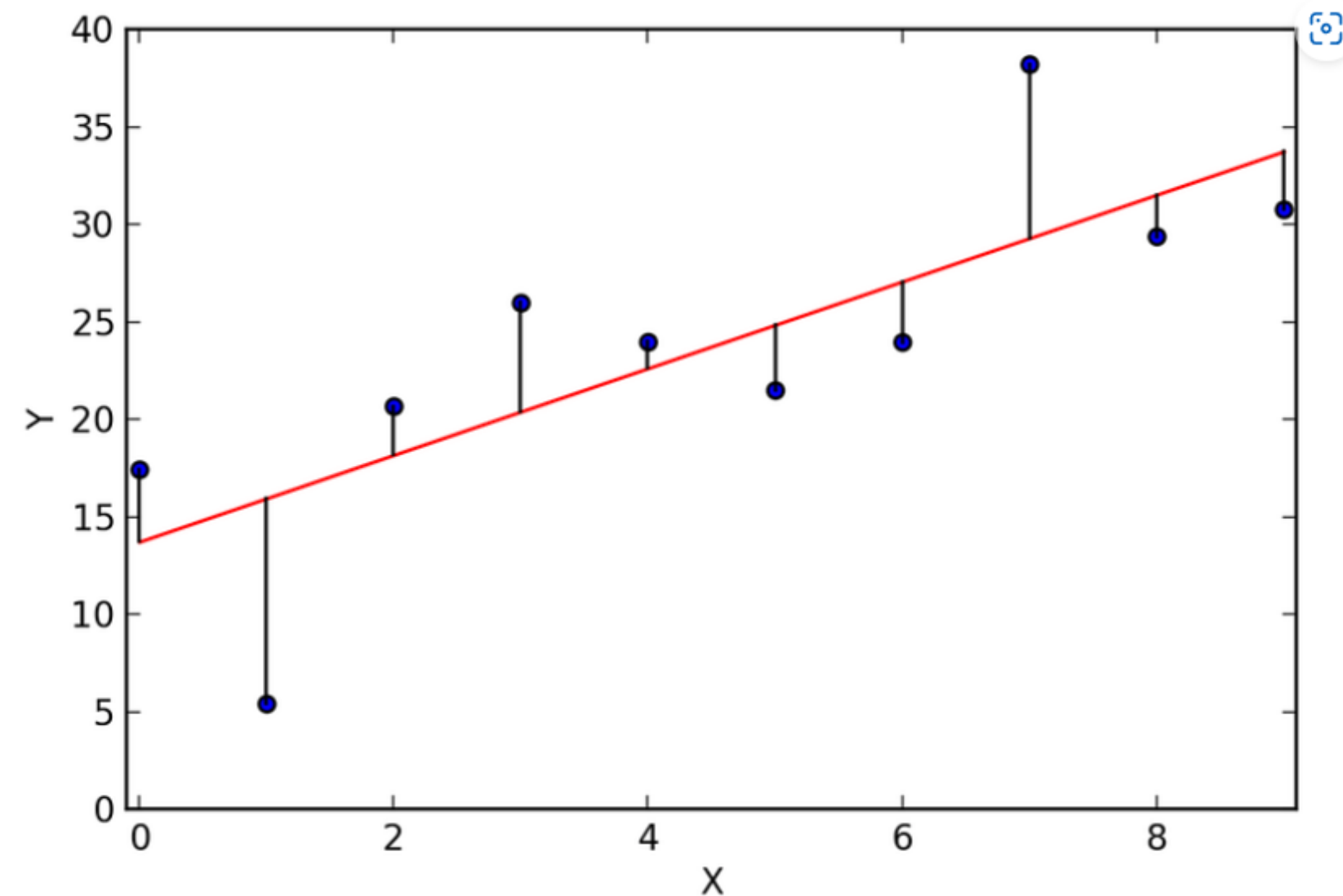
This is an example where x is 1 dimensionsal. If x would be n-dimensional, the only difference would that instead of the cartesian plane, we would be in the n-dimensional plane hyper plane.
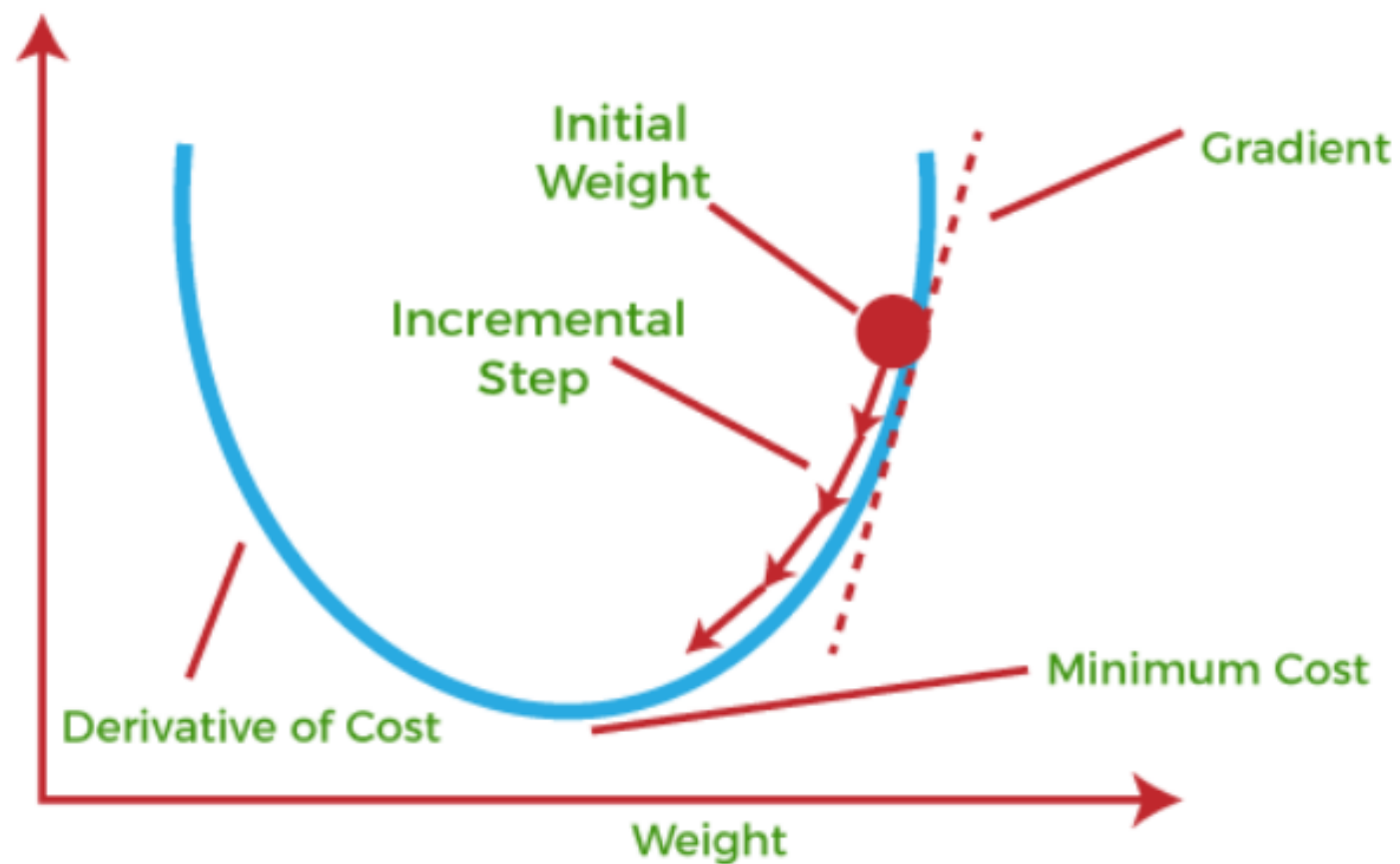
How to find w and b?

To find the line-of-best-fit, we need to do an optimization process:

$$\frac{1}{N} \sum_{i=1...N} (f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2. \tag{2}$$

In mathematics, the expression we minimize or maximize is called an objective function, or, simply, an objective. The expression $(f(\mathbf{x}_i) - y_i)^2$ in the above objective is called the **loss function**. It's a measure of penalty for misclassification of example i. This particular choice of the loss function is called **squared error loss**. All model-based learning algorithms have a loss function and what we do to find the best model is we try to minimize the objective known as the **cost function**. In linear regression, the cost function is given by the average loss, also called the **empirical risk**. The average loss, or empirical risk, for a model, is the average of all penalties obtained by applying the model to the training data.

Gradient Descent diagram showing Initial Weight, Gradient, Incremental Step, Derivative of Cost, Minimum Cost, and Weight axes.

Algorithm:

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

↓ Assignment operator.

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

→ Learning rate

(It is a number b/w 0 and 1 which controls how big a step is during gradient descent)

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

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

$$w = tmp\_w$$

$$b = tmp\_b$$

$(✓)$

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

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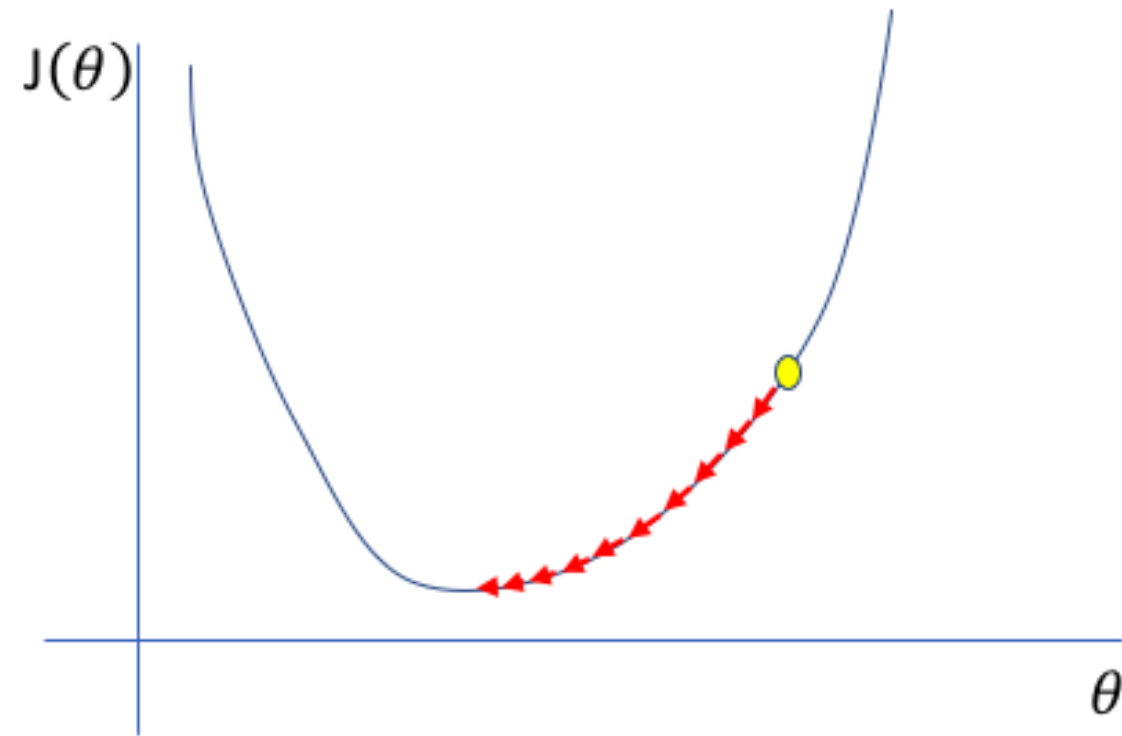
$$tmp\_b = b - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b = tmp\_b$$

$(✗)$

Too low
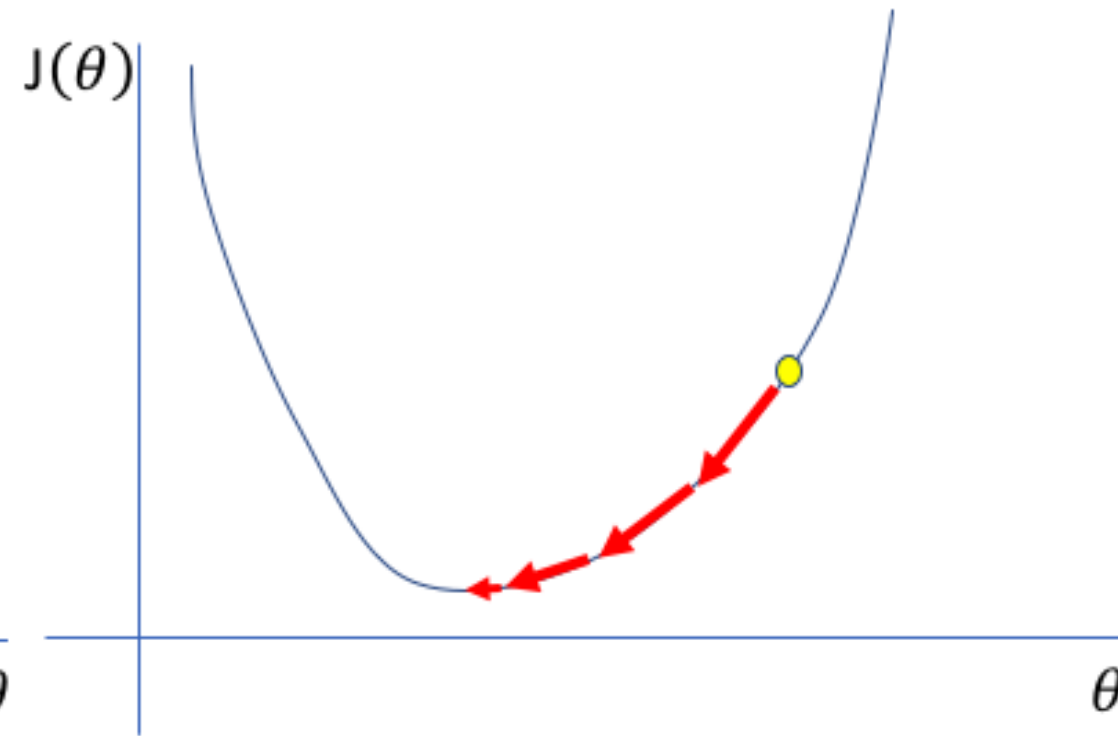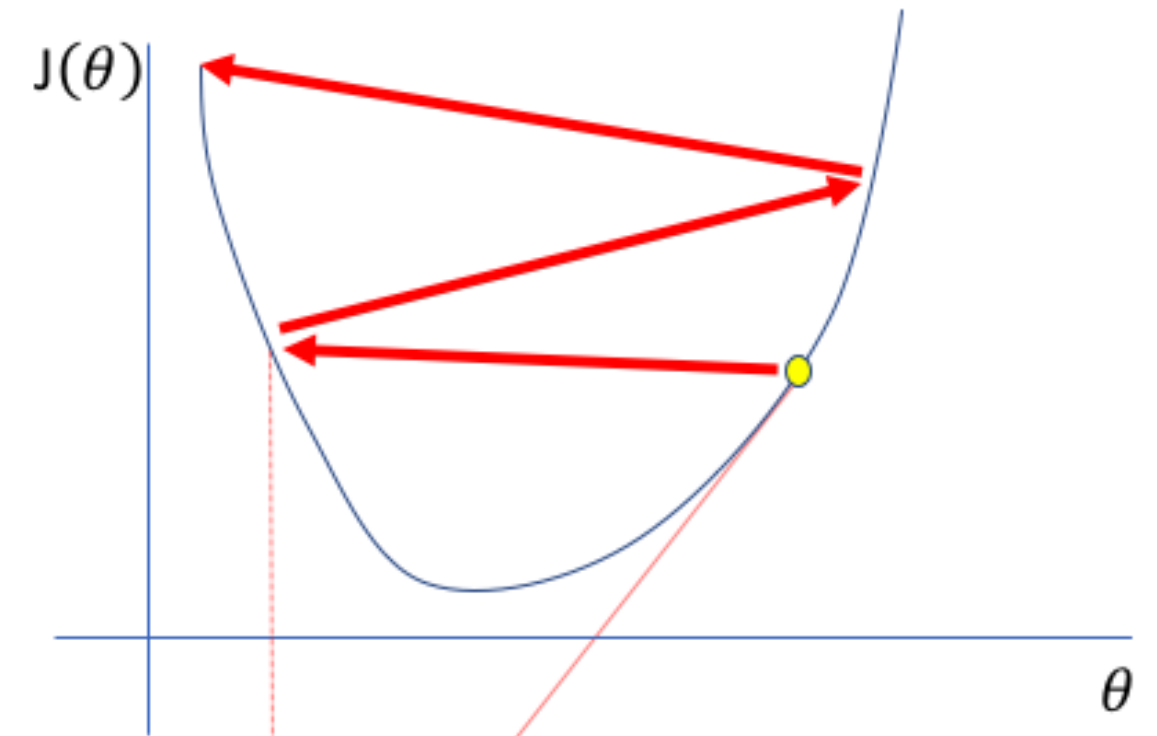
$J(\theta)$

θ

A small learning rate requires many updates before reaching the minimum point

Just right

$J(\theta)$

θ

The optimal learning rate swiftly reaches the minimum point

Too high

$J(\theta)$

θ

Too large of a learning rate causes drastic updates which lead to divergent behaviors

The Gradient Decent Algorithm

$$l = \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2.$$

We must minimize the above cost function to find optimum values of **w** and b. Gradient descent starts with calculating the partial derivative for every parameter:

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (wx_i + b));$$

$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (wx_i + b)).$$

Code:

NoteBook Link

Evaluation Metrics for Regression

- **Mean Absolute Error(MAE) :** MAE calculates the absolute difference between actual and predicted values.   **sum of (y_test - y_pred) / total_samples.**

  from sklearn.metrics import mean_absolute_error

  mean_absolute_error(y_test,y_pred)

- **Mean Squared Error(MSE)** : finds the squared difference between actual and predicted value. **sum of (y_test - y_pred)^2 / total_samples.**

  from   sklearn.metrics import mean_squared_error

  mean_squared_error(y_test,y_pred)

- **Root Mean Squared Error(RMSE) :** root over MSE

  np.sqrt(mean_squared_error(y_test,y_pred))

- **R Squared (R2) :** also known as Coefficient of Determination or Goodness of fit. R2 squared calculates how much is the regression line better than the mean line.

  R2 = 1 - (sum of (y_test - y_pred)^2)/(sum of (y_test - y_test_mean)^2)

  from sklearn.metrics import r2_score

  r2 = r2_score(y_test,y_pred)

Supervised Learning -
Logistic Regression

In logistic regression, we still want to model y_i as a linear function of $\mathbf{x}_i$, however, with a binary y_i this is not straightforward. The linear combination of features such as $\mathbf{w}\mathbf{x}_i + b$ is a function that spans from minus infinity to plus infinity, while y_i has only two possible values.
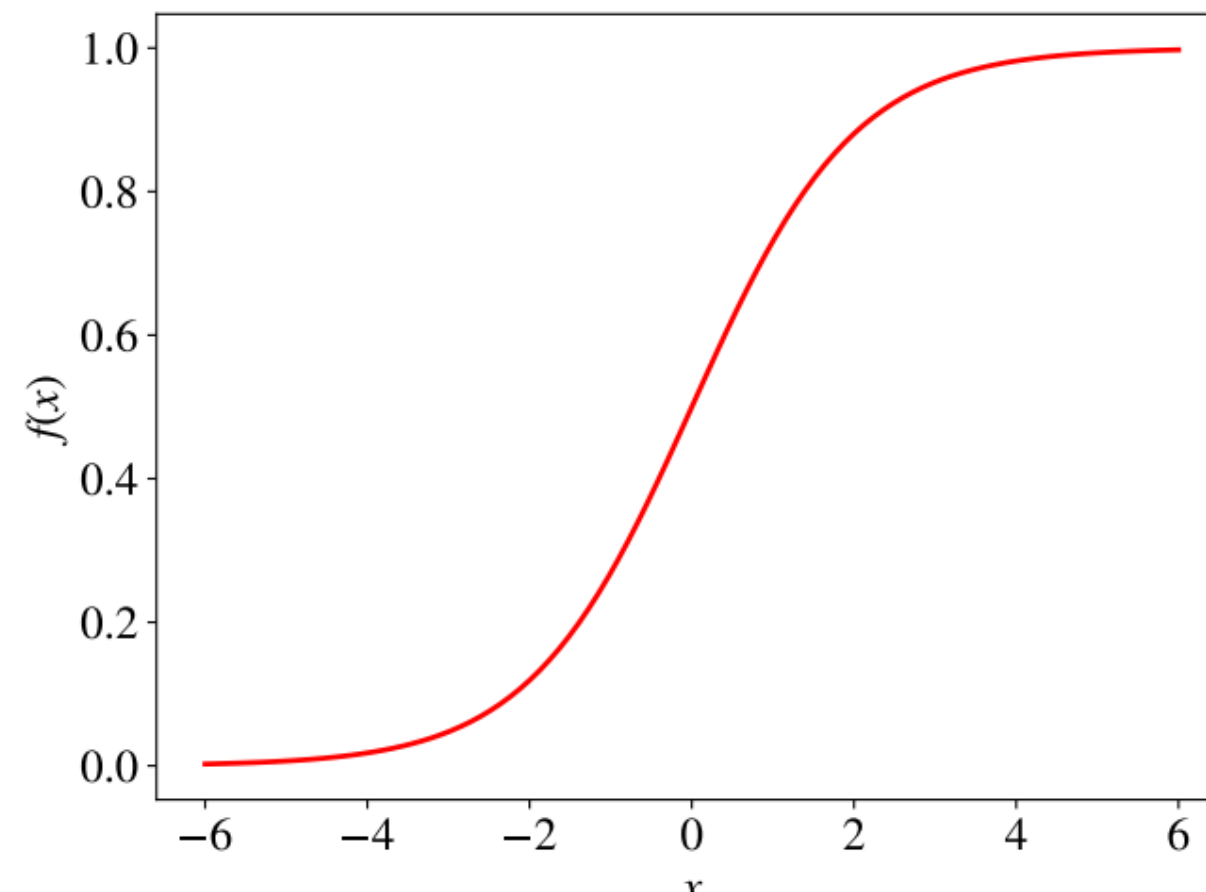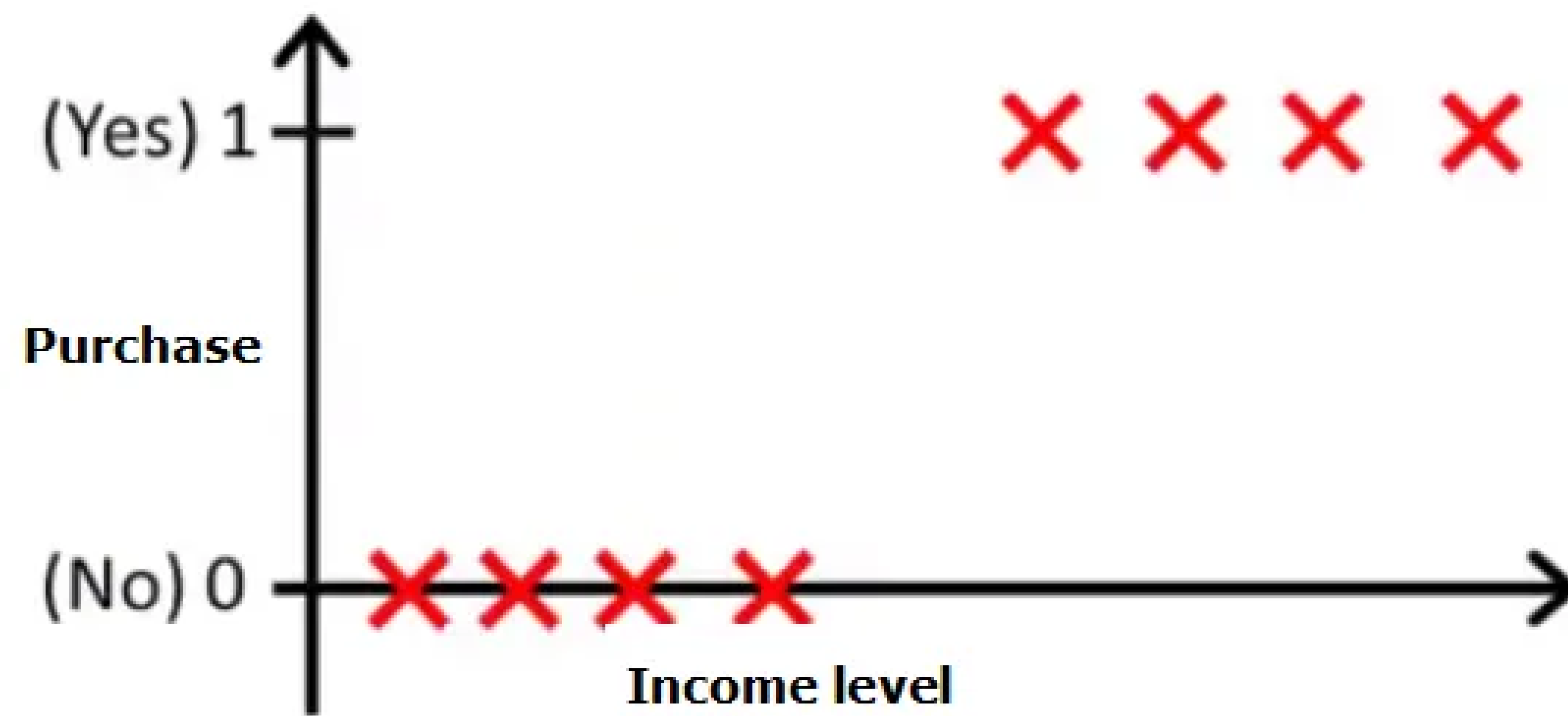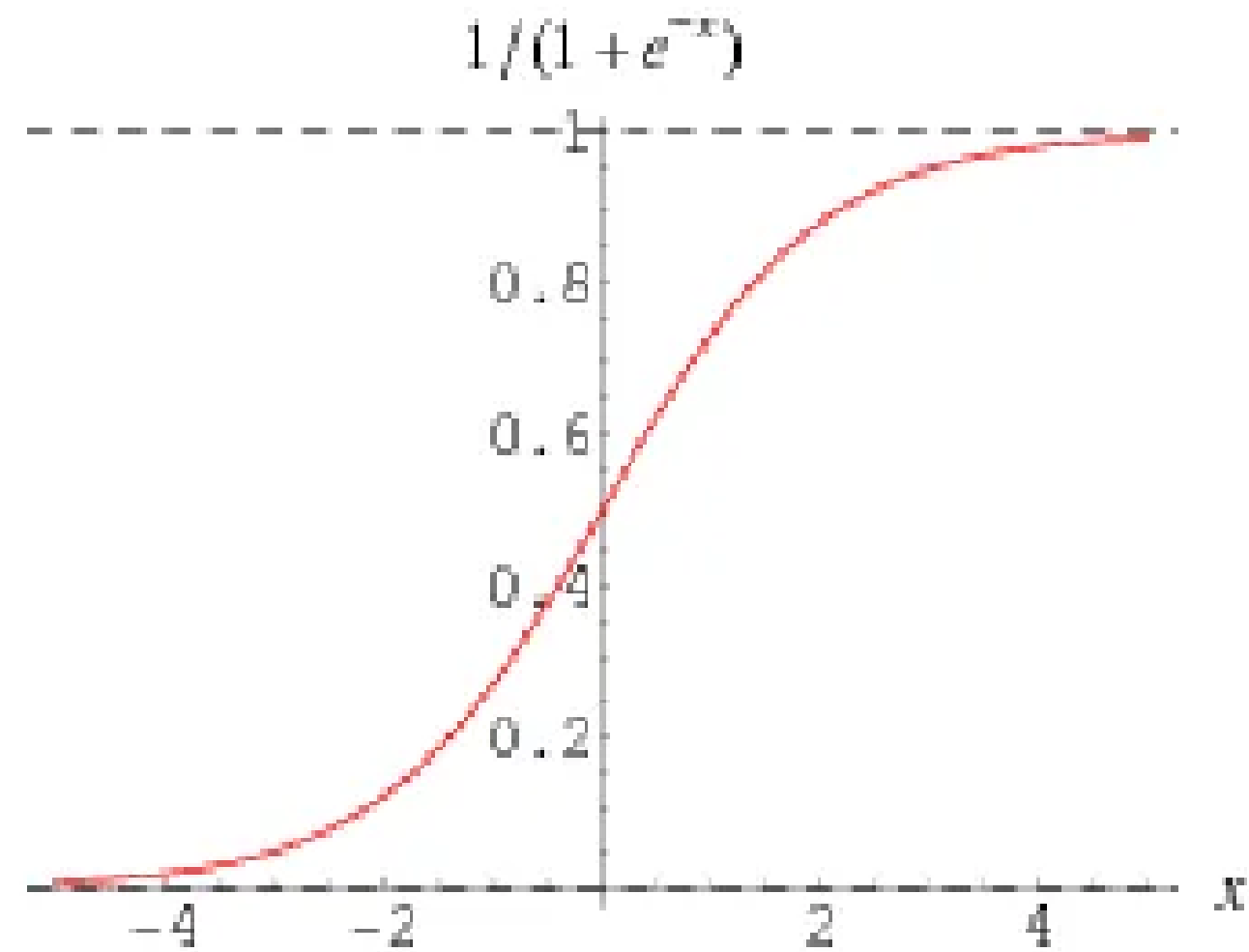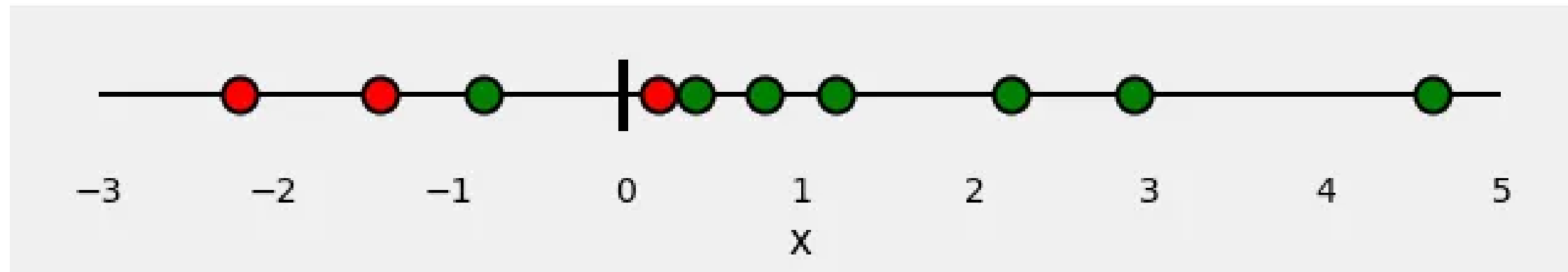
Why not Linear Regression?
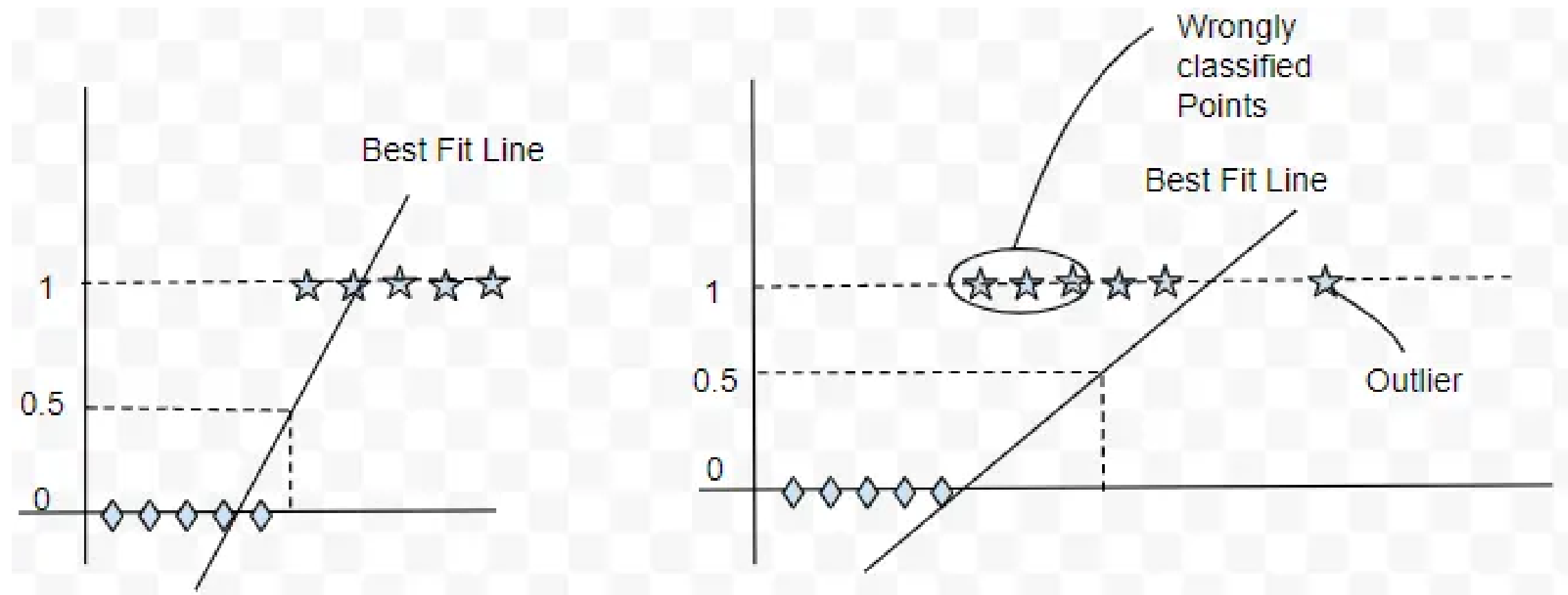
a. Outlier: If the data set has an outlier, linear regression will not perform better.

In the above two graphs it is clear that a linear line is not the ideal decision surface as it may miss-classify many values , what we ideally need is a decision surface which outputs the decision as 0 or 1/yes or no.

$$1/(1 + e^{-x})$$

If we fit a model to perform this classification, it will predict a probability of being green to each one of our points.

Given what we know about the color of the points, how can we evaluate how good (or bad) are the predicted probabilities? This is the whole purpose of the loss function! It should return high values for bad predictions and low values for good predictions.

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$
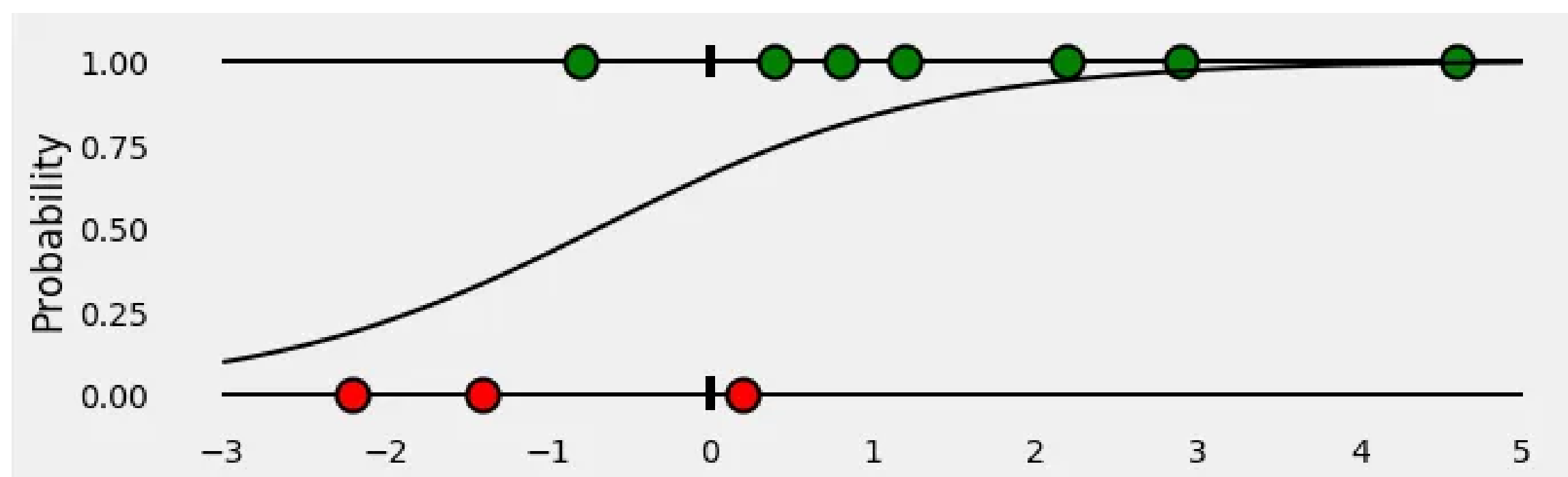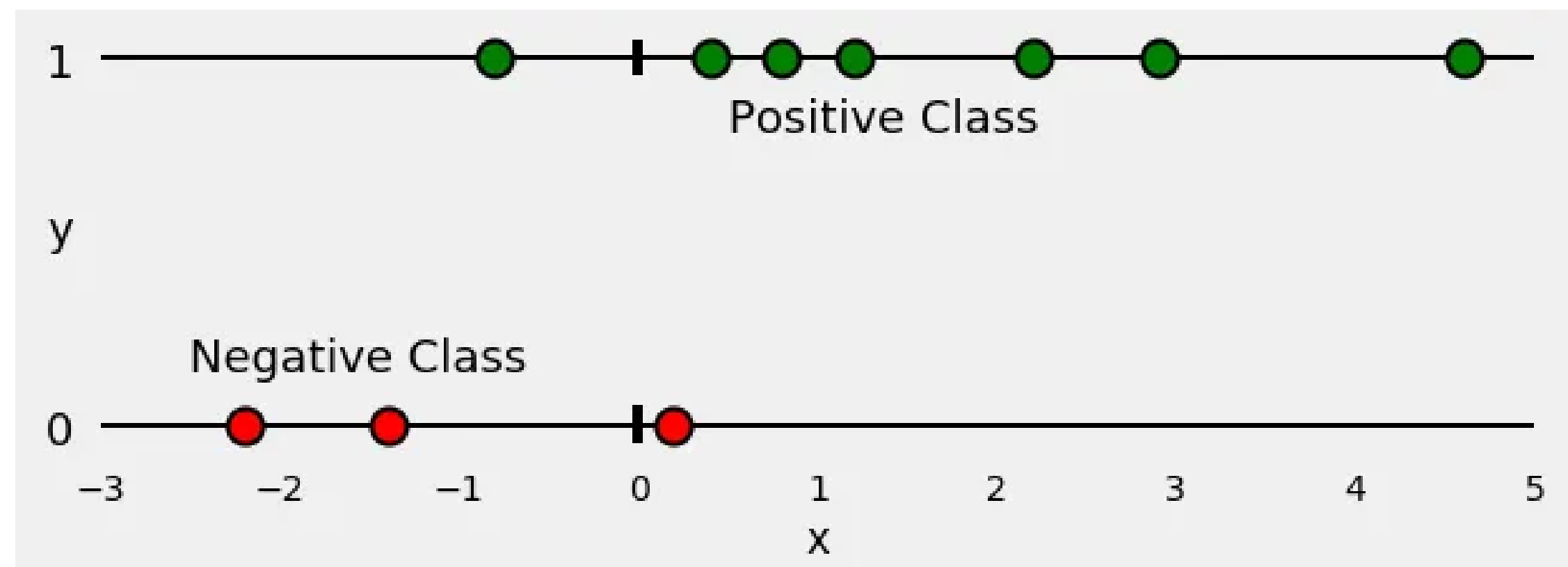
Then, for all points belonging to the positive class (green), what are the predicted probabilities given by our classifier? These are the green bars under the sigmoid curve, at the x coordinates corresponding to the points.

Log Loss
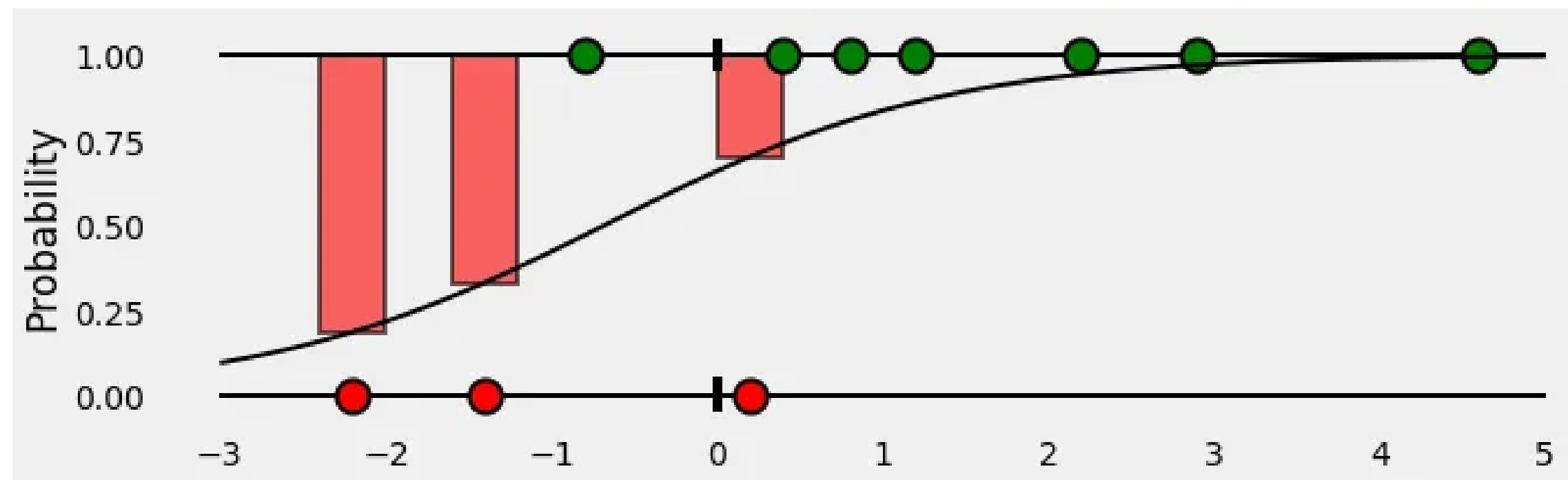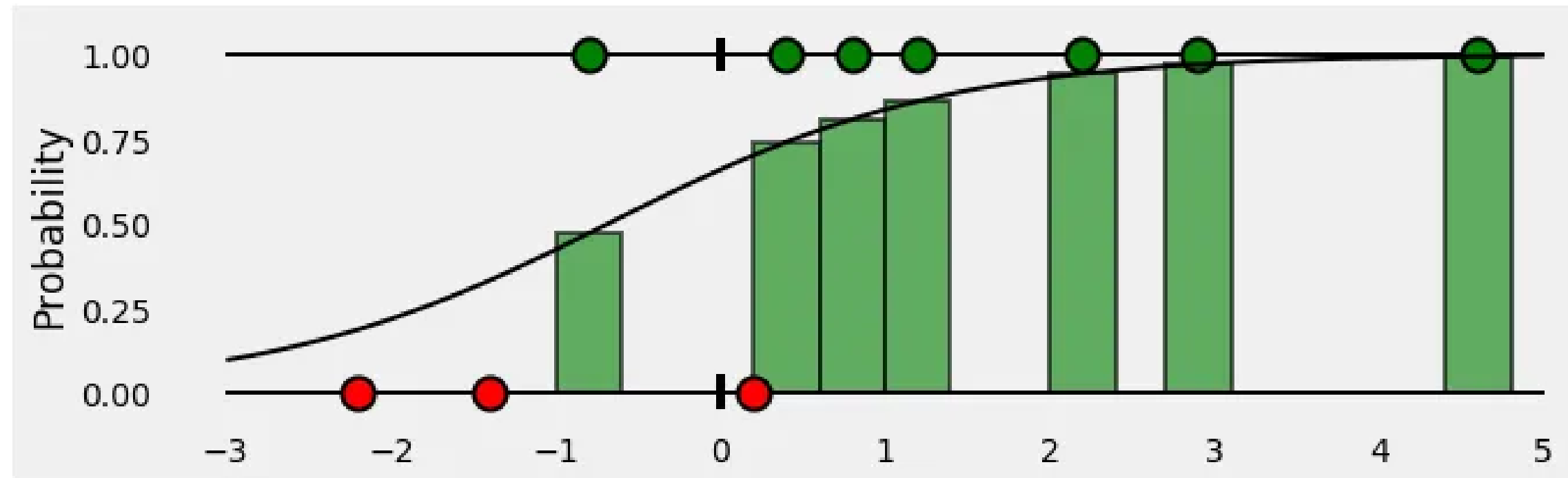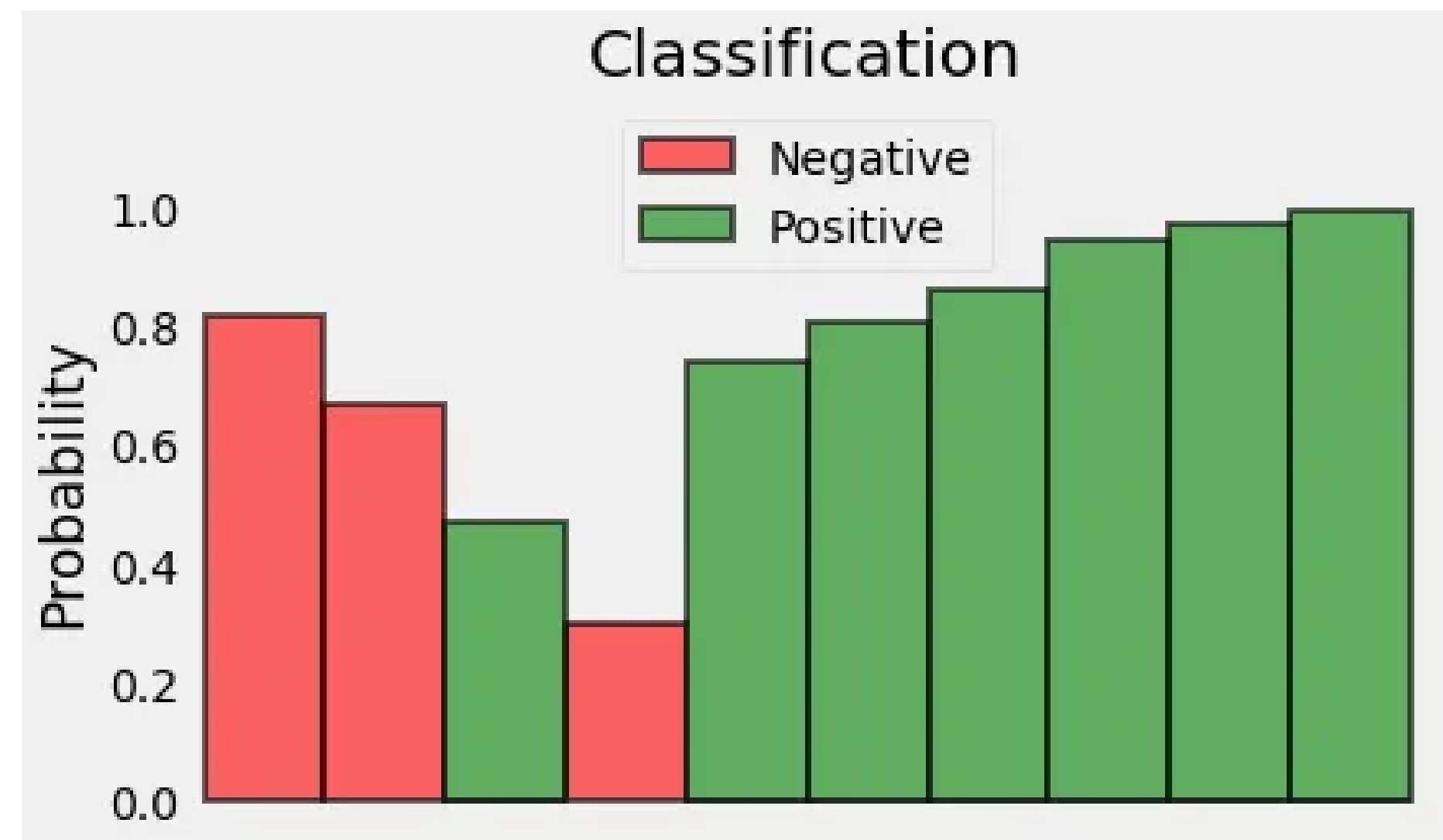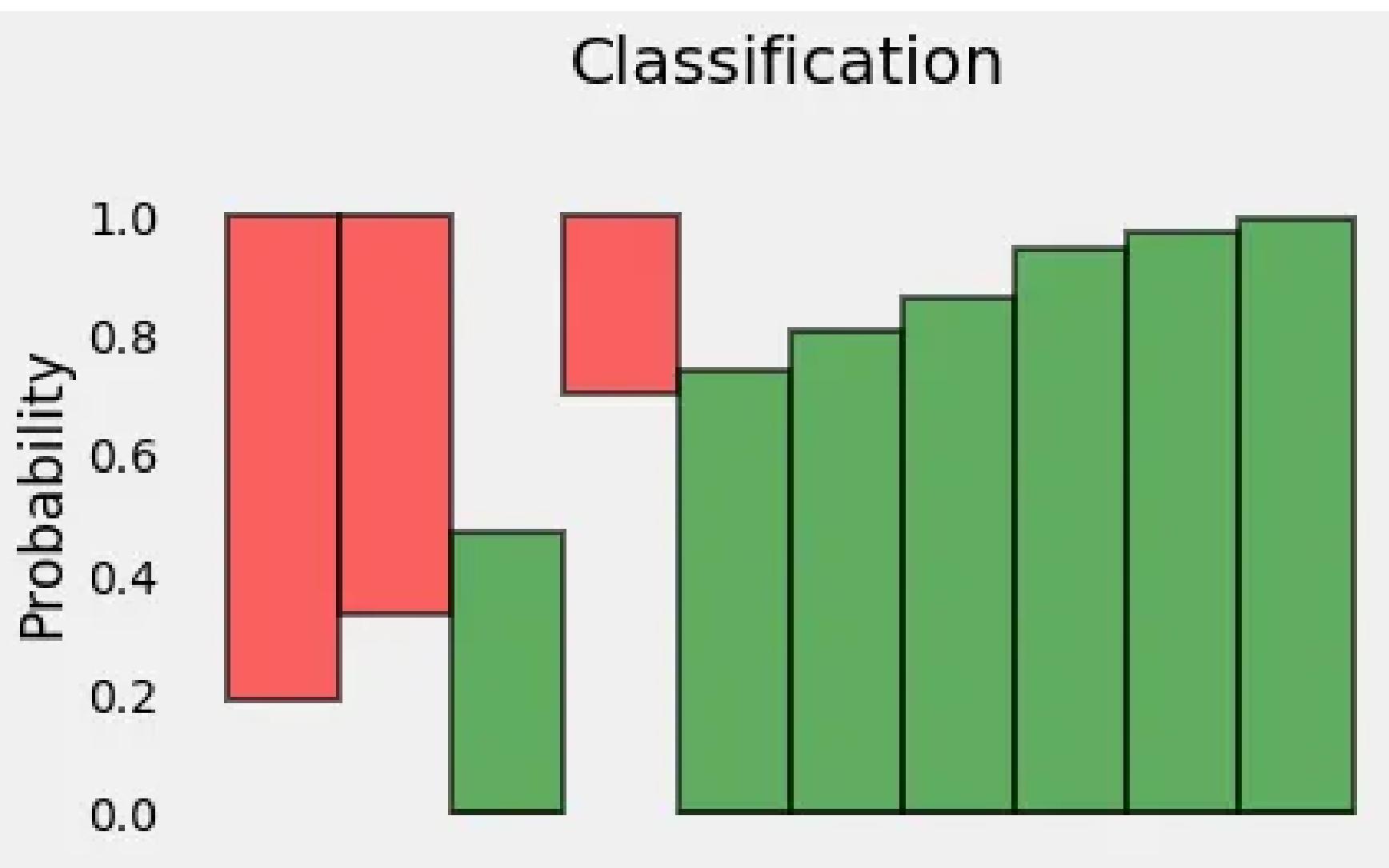
By looking at the graph of the standard logistic function, we can see how well it fits our classification purpose: if we optimize the values of $\mathbf{x}$ and b appropriately, we could interpret the output of $f(\mathbf{x})$ as the probability of y_i being positive. For e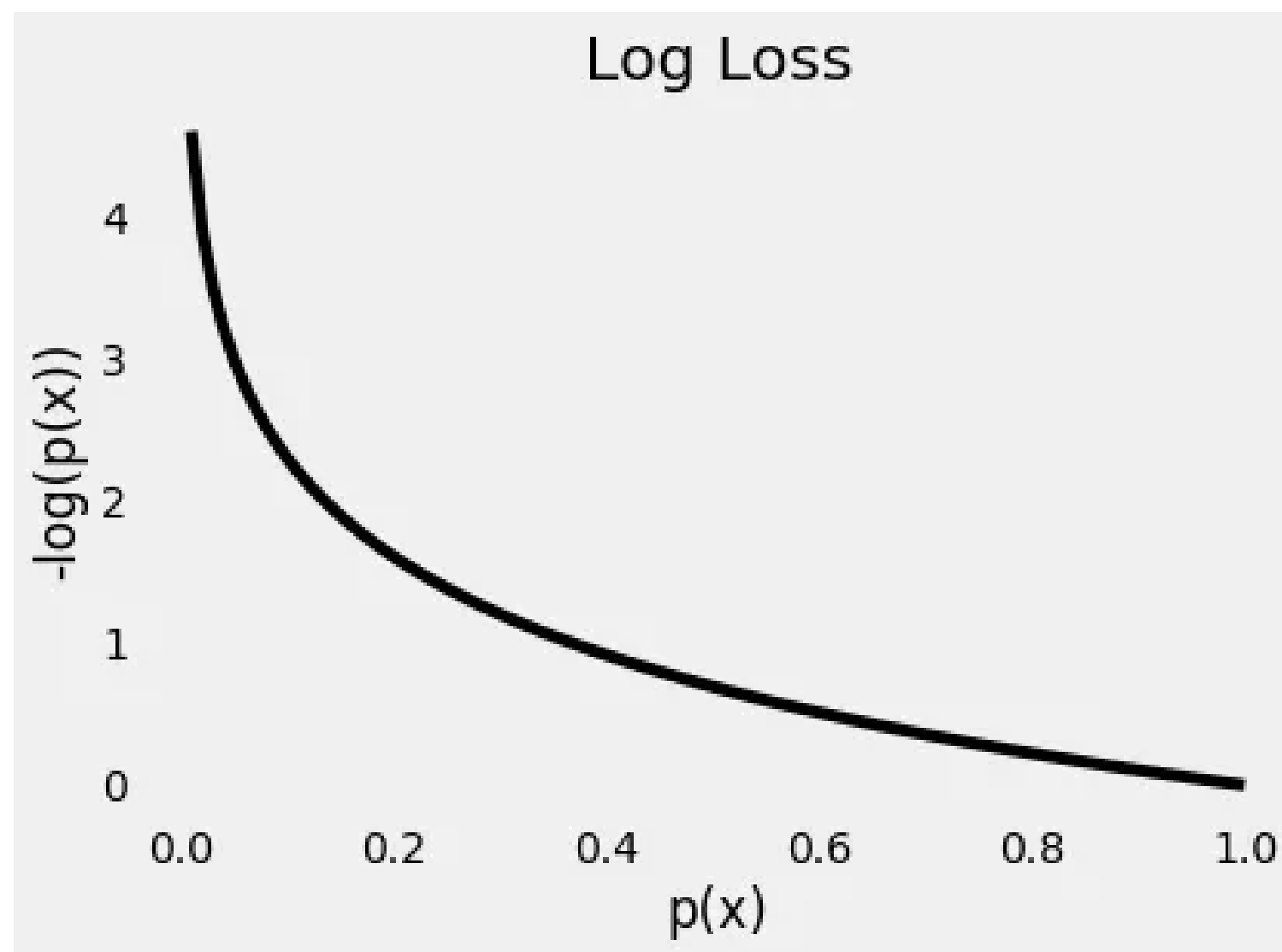xample, if it's higher than or equal to the threshold 0.5 we would say that the class of $\mathbf{x}$ is positive; otherwise, it's negative. In practice, the choice of the threshold could be different depending on the problem. We

$$f_{\mathbf{w},b}(\mathbf{x}) \overset{\text{def}}{=} \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x}+b)}}. \qquad (3)$$
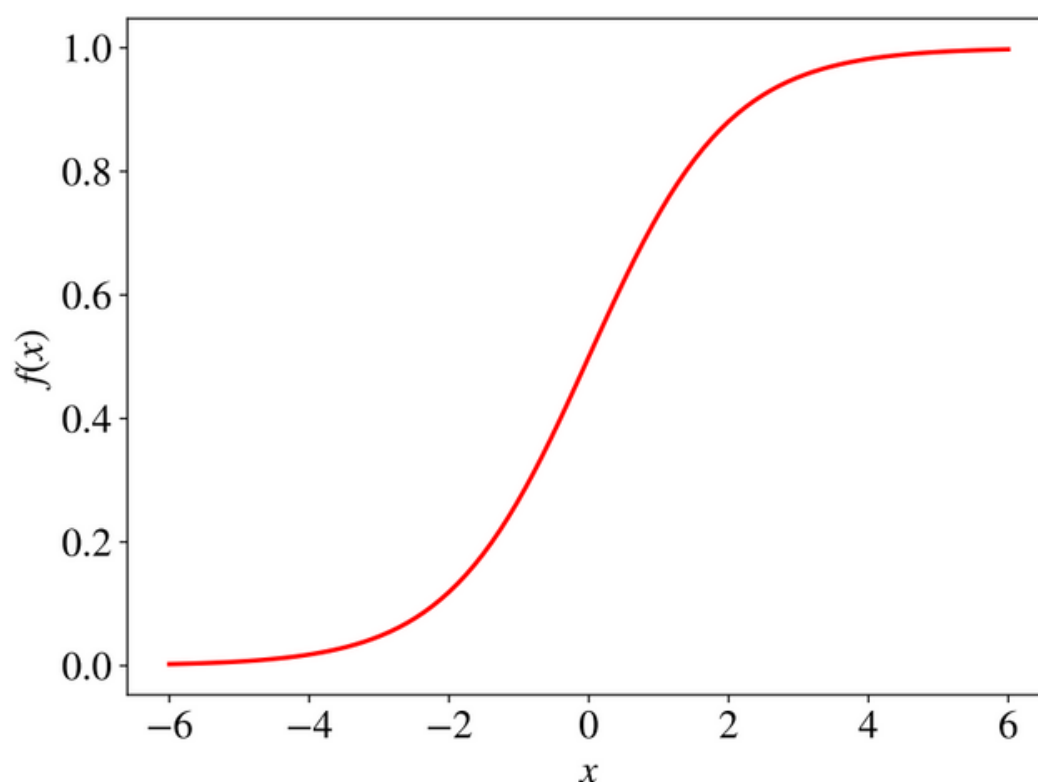
Squared error cost

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

loss $\boxed{L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})}$

linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

convex

$J(\vec{w}, b)$ — $\vec{w}, b$
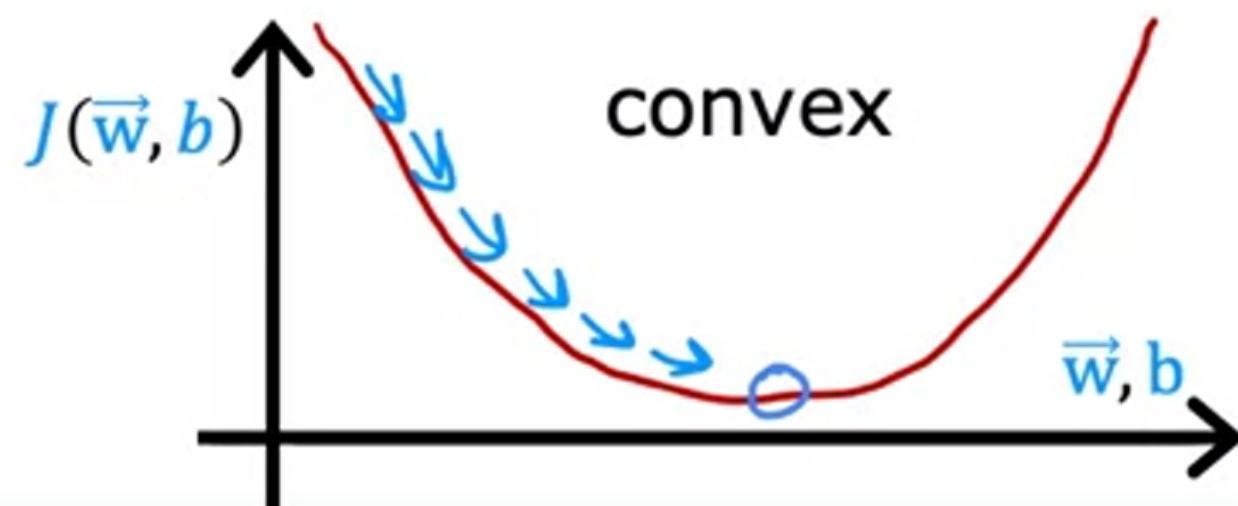
logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

non-convex

$J(\vec{w}, b)$ — $\vec{w}, b$
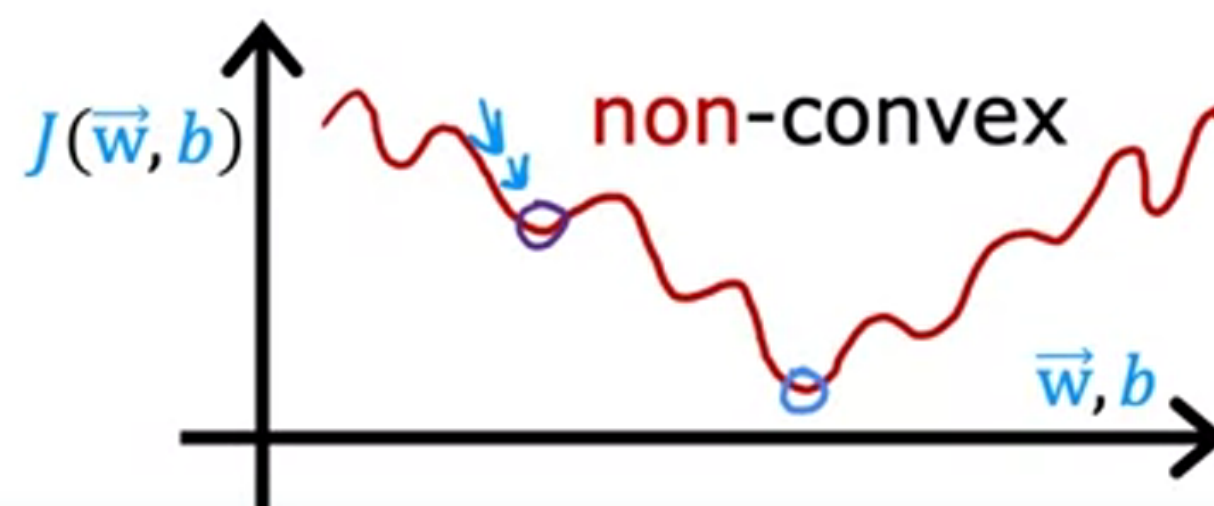
Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(\underbrace{f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}}_{loss})$$

$$= \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Simplified loss function

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$$\boxed{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)}$$

Simplified cost function

loss
$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) - (1 - y^{(i)})\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)$$

cost
$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)\right]$$

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

Gradient Descent

$\text{algorithm}:$

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

$\downarrow$ Assignment operator.

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

$\longrightarrow$ learning rate

$(\text{It is a number b/w } 0 \text{ and } 1 \text{ which controls how big a step is during gradient descent})$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\frac{\partial(J(\theta))}{\partial(\theta)} = \frac{1}{m} X^T [\; h_\theta(x) - y]$$

This derivative of cost function is with respect to weight and biases for each sample.
For calculating the total derivative we do summation over all the samples

Code:

**Notebook
Link**

Evaluation Metrics for Classification

- **Accuracy :** correct predictions / total predictions. OR (TP + TN)/(TP + TN+ FP +FN)
  sklearn.metrics.accuracy_score
- **Precision** : number of true positives divided by the number of predicted positives. (TP)/(TP+FP)
  sklearn.metrics.average_precision_score
- **Recall (Sensitivity):** number of true positives divided by the total number of actual positives. TP/(TP+FN)
- **F1 Score :** harmonic mean of precision and recall.
- **AUC-ROC**— [Read here](#)
  sklearn.metrics.roc_auc_score
- **Binary Cross Entropy Loss or Log Loss :**
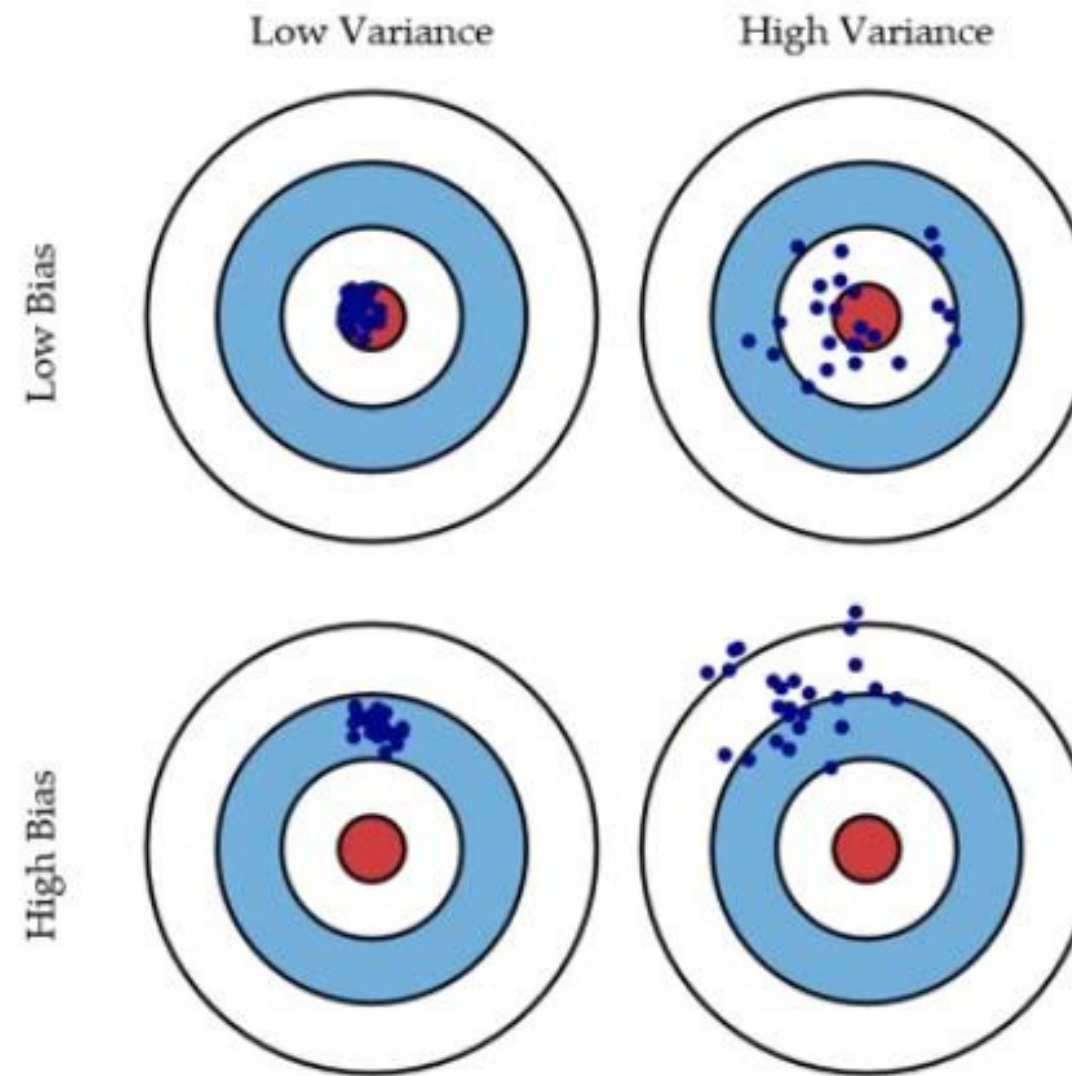 y_test ln (y_pred) + (1-y_test) ln (1- y_pred)

Confusion Matrix

Actual Values

	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

Predicted Values

sklearn.metrics.confusion_matrix

Bias and Variance in Regression Models

Let's say we have model which is very accurate, therefore the error of our model will be low, meaning a low bias and low variance as shown in first figure. All the data points fit within the bulls-eye. Similarly we can say that if the variance increases, the spread of our data point increases which results in less accurate prediction. And as the bias increases the error between our predicted value and the observed values increases.

As we add more and more parameters to our model, its complexity increases, which results in increasing variance and decreasing bias, i.e., overfitting. So we need to find out one optimum point in our model where the decrease in bias is equal to increase in variance. In practice, there is no analytical way to find this point. So how to deal with high variance or high bias?

To overcome underfitting or high bias, we can basically add new parameters to our model so that the model complexity increases, and thus reducing high bias.

Basically there are two methods to overcome overfitting,

- Reduce the model complexity
- Regularization

There are two types of regularization :

1. L1 Regularization
2. L2 Regularization or Weight Decay

The idea of regularization is to add an extra term to the cost function, a term called the regularization term. The modified cost functions are : (lambda is a parameter and theta represents the weights.)

$$\min \left(||Y - X\theta||_2^2 + \lambda ||\theta||_1 \right) \qquad \min \left(||Y - X(\theta)||_2^2 + \lambda ||\theta||_2^2 \right)$$

Ridge regression uses **L2 regularization** and is used in cases where there are many parameters or predictors that affect the outcome.

Lasso regression (Least Absolute Shrinkage and Selection Operator) uses **L1 regularization** and works better when there are fewer significant parameters or predictors involved. There is another type of regression called **ElasticNet** which is a combination of both L1 and L2 regularization. It has two parameters, alpha and L1_ratio. alpha= a + b   and L1_ratio = a/(a+b)   where a and b are weights assigned to L1 and L2 term respectively.