Faculty of Sciences Tetouan
Mathematics Department
2024-2025

Introduction to Neural Networks: Lecture 2: Feedforward Neural Networks

Pr. Youness EL YAZIDI

1 Introduction

Introduction

Introduction

Feedforward : Information passes from the input data x through some intermediate steps (hidden layers), to the output y (The direct sense).

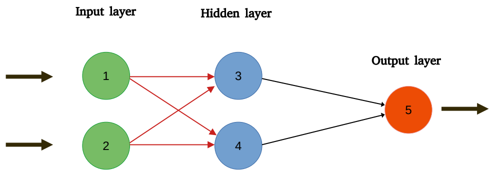
Definition

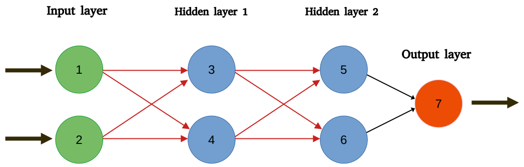
A Feedforward Neural Network (FNN) defines a mapping f to approximate the target; $y = f(x;\theta)$. This feedforward neural network is trained with an optimization technique, so that y results in the best approximation.

FNN are a type of artificial neural network where connections between the nodes do not form a cycle. They are the simplest type of artificial neural network.

Architecture of Feedforward Networks:

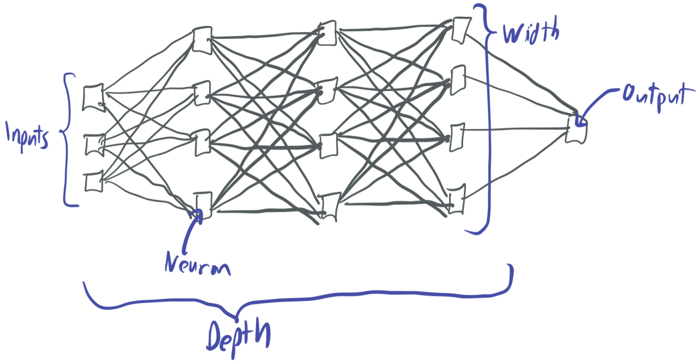
- Consist of an input layer, one or more hidden layers, and an output layer.
- Neurons in each layer are fully connected to neurons in the next layer.





Definition

- The **length** of the chain of functions in a neural network is called its depth.
- The dimensionality of the hidden layers of a neural network is called its **width**



How FNN works?

The process of a FNN involves two stages: the feedforward phase and the backpropagation phase.

Feedforward Phase:

The input data is fed into the network, and it propagates forward through the network. At each hidden layer, the weighted sum of the inputs is calculated and passed through an activation function, which introduces non-linearity into the model. This process continues until the output layer is reached, and a prediction is made.

Introduction

Backpropagation Phase:

Once a prediction is made, the error (difference between the predicted output and the actual output) is calculated. This error is then propagated back through the network, and the weights are adjusted to minimize this error. The process of adjusting weights is typically done using a gradient descent optimization algorithm.

Feedforward phase

Feedforward phase

Forward pass involves computing the output of each neuron in the network from the input layer to the output layer, uses an activation function to determine the output of each neuron.
The loss function computes the error between the predictions and the actual targets.

Activation function

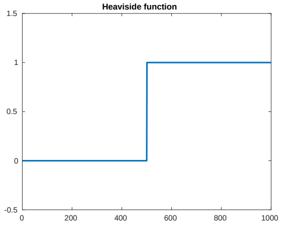
Taking the activation function we have seen in the first lecture, that we can write as

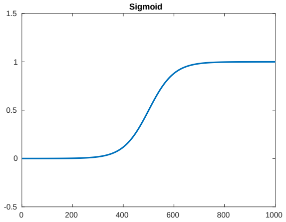
$$y = f(g(x) - \theta) = \begin{cases} 1, & \text{if } g(x) - \theta \geq 0 \\ 0, & \text{if } g(x) - \theta < 0 \end{cases}$$

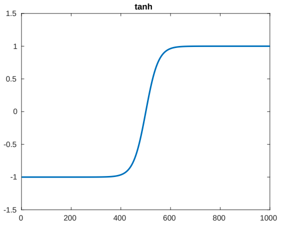
this function is exactly the so-called Heaviside function. Which is defined by as next

$$H(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

Feedforward phase







$$H(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases} \qquad f(z) = \frac{1}{1+e^{-z}} \qquad f(z) = \tanh(z)$$

- The outputs of Sigmoid or Logistic function range from 0 to 1 and are often interpreted as probabilities
- tanh is a rescaling of sigmoid function, such that outputs range from -1 to 1. There is horizontal stretching as well.

Feedforward phase

• The **sigmoid** function is usually used in output layer of a binary classification, where result is either 0 or 1. The prediction can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.

Feedforward phase

• The **sigmoid** function is usually used in output layer of a binary classification, where result is either 0 or 1. The prediction can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.
• The **tanh** function is usually used in hidden layers of a neural network as it's values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.

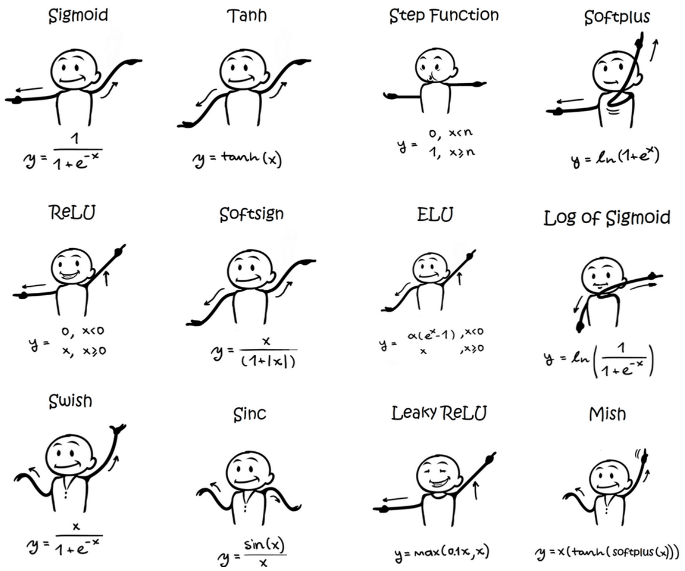
Feedforward phase

• The **sigmoid** function is usually used in output layer of a binary classification, where result is either 0 or 1. The prediction can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.
• The **tanh** function is usually used in hidden layers of a neural network as it's values lies between -1 to 1 hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in centering the data by bringing mean close to 0. This makes learning for the next layer much easier.
• The activation that works almost always better than **sigmoid** function is **Tanh** function. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.

Feedforward phase

• The activation function decides whether a neuron should be activated or not.
• The purpose is to introduce non-linearity into the output of a neuron.
• A neural network without an activation function is just a linear regression model.
• The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Feedforward phase


Sigmoid
$y = \frac{1}{1+e^{-x}}$
Tanh
$y = \tanh(x)$
Step Function
$y = \begin{cases} 0, & x<n \\ 1, & x \geq n \end{cases}$
Softplus
$y = \ln(1+e^{x})$
ReLU
$y = \begin{cases} 0, & x<0 \\ x, & x \geq 0 \end{cases}$
Softsign
$y = \frac{x}{(1+|x|)}$
ELU
$y = \begin{cases} \alpha(e^{x}-1), & x<0 \\ x, & x \geq 0 \end{cases}$
Log of Sigmoid
$y = \ln\left(\frac{1}{1+e^{-x}}\right)$
Swish
$y = \frac{x}{1+e^{-x}}$
Sinc
$y = \frac{\sin(x)}{x}$
Leaky ReLU
$y = \max(0.1x, x)$
Mish
$y = x(\tanh(\text{softplus}(x)))$

Loss function

In optimization context, the function used to evaluate a candidate solution is referred to as the objective (called also cost or loss) function, which we seek to maximize or minimize.
In neural networks, we seek to minimize the error between the constructed $\hat{y}$ and the given output y.

Cost/Loss/Objective function

The cost function reduces all the various good and bad aspects of a possibly complex system down to a single number, a scalar value, which allows candidate solutions to be ranked and compared.

Feedforward phase

• The choice of cost function is tightly coupled with the choice of output unit.

• A simple kind of output unit is one based on an affine transformation with no non-linearity:

$$\hat{y} = W^T h(x;\theta) + b$$

- $\hat{y}$ is the output of the whole model, or $f(x;\theta)$,
- $h(x;\theta)$ is the output of the final hidden layer.

• The cost function is a measure of how well our algorithm performs.

• Training is performed through minimizing the cost function.

Mean Absolute Error

$$J(w) = \frac{1}{N}\sum_{k=1}^{N} |\hat{y}_k - y_k|$$

It also has some disadvantages; as the average distance approaches 0, gradient descent optimization will not work, as the function's derivative at 0 is undefined

- a convex function (it has a global minimum)
- suitable for gradient based optimization

Mean Squared Error

$$J(w) = \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

This function has numerous properties that make it especially suited for calculating loss:

- a convex function (it has a global minimum)
- suitable for gradient based optimization

Cross Entropy (Log Loss)

$$J(w) = \frac{1}{N}\sum_{k=1}^{N}\Big[-y_k\log(\hat{y}_k) - (1-y_k)\log(1-\hat{y}_k)\Big]$$

- a convex function
- suitable for gradient based optimization

Backpropagation phase

Backpropagation phase

Backpropagation

is a technique used to train neural networks, it works by propagating the error of the learning backward through its layers to adjust weights and biases in a way that minimizes the loss function.

Backpropagation: backward propagation of errors.

Descent gradient algorithm

Neural networks aim to learn a mapping from inputs to outputs by minimizing a loss function $L(\theta)$, which measures how far the network's predictions are from the true outputs. θ represents the parameters (weights and biases) of the network.

Descent gradient algorithm

Neural networks aim to learn a mapping from inputs to outputs by minimizing a loss function $L(\theta)$, which measures how far the network's predictions are from the true outputs. θ represents the parameters (weights and biases) of the network. Gradient descent algorithm uses the derivative of the loss function with respect to the parameters θ to determine the direction in which the parameters should move in order to reduce the cost.

For each parameter θ_i

$$\theta_i = \theta_i - \eta \frac{\partial L}{\partial \theta_i}$$

η is the learning rate (a small, positive scalar that controls the step size).
$\frac{\partial L}{\partial \theta_i}$ Gradient of the loss function with respect to the parameter θ_i.

Backpropagation phase

- By computing gradients, backpropagation allows neural networks to learn patterns from data.

Backpropagation phase

• By computing gradients, backpropagation allows neural networks to learn patterns from data.

• Backpropagation allows gradients to be computed efficiently in $O(n)$ time for networks with nn parameters.

Backpropagation phase

- By computing gradients, backpropagation allows neural networks to learn patterns from data.
- Backpropagation allows gradients to be computed efficiently in $O(n)$ time for networks with nn parameters.
- Most deep learning models rely on backpropagation to train multi-layer networks.

Limitations

- **Vanishing/Exploding Gradients**: In deep networks, gradients can become very small (vanish) or very large (explode), hindering training.

Limitations

• **Vanishing/Exploding Gradients**: In deep networks, gradients can become very small (vanish) or very large (explode), hindering training.
• **Computational Intensity**: Requires significant computational resources for large networks.

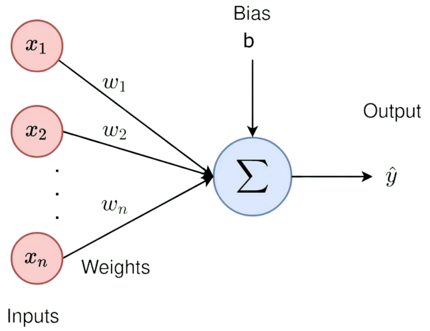
Limitations

• **Vanishing/Exploding Gradients**: In deep networks, gradients can become very small (vanish) or very large (explode), hindering training.
• **Computational Intensity**: Requires significant computational resources for large networks.
• **Dependence on Data**: Highly sensitive to the quality and quantity of training data.

Linear Regression example

Linear Regression example

Let's consider a simple network with a single layer composed from one unit.



Linear Regression example

We aim to model a simple linear relationship:

$$y = wx + b$$

where x is the **input**, y is the **target output**, w and b (**weight** and **bias**) are the parameters to learn.
Forward phase: for a data points (x, y) we compute

1. the predicted output:

$$\hat{y} = wx + b$$

2. the loss using Mean Squared Error (MSE):

$$L = \frac{1}{N}(\hat{y} - y)^2$$

Linear Regression example

Backpropagation phase: using the chain rule:

1. Compute the gradient of the loss with respect to $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

Linear Regression example

Backpropagation phase: using the chain rule:

1. Compute the gradient of the loss with respect to $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

2. Propagate to the parameters:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w} = (\hat{y} - y)x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = (\hat{y} - y)$$

Linear Regression example

Backpropagation phase: using the chain rule:

1. Compute the gradient of the loss with respect to $\hat{y}$:

$$\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$

2. Propagate to the parameters:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w} = (\hat{y} - y)x$$
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b} = (\hat{y} - y)$$

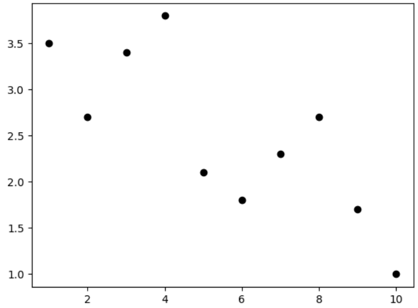
3. Update the parameters using gradient descent:

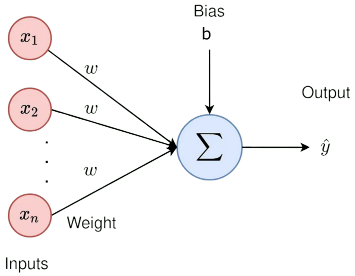
$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

$$b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

Linear Regression example

x =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y =[3.5, 2.7, 3.4, 3.8, 2.1, 1.8, 2.3,2.7, 1.7, 1]





We assume that $w = 1$, $b = 1$ and we set the learning rate at $\eta = 0.01$.

Linear Regression example

Iteration 1:

Forward Pass:

$$\hat{y} = 1 \cdot x + 1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].$$

Linear Regression example

Iteration 1:

Forward Pass:

$$\hat{y} = 1 \cdot x + 1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].$$

Compute the loss:

$$L = \frac{1}{10} \sum_{i=1}^{10} (\hat{y}_i - y_i)^2.$$

Linear Regression example

Iteration 1:

Forward Pass:

$$\hat{y} = 1 \cdot x + 1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].$$

Compute the loss:

$$L = \frac{1}{10} \sum_{i=1}^{10} (\hat{y}_i - y_i)^2.$$

Substitute:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{10} [-1.5, 0.3, 0.6, 1.2, 3.9, 5.2, 5.7, 6.3, 8.3, 10].$$

Linear Regression example

Iteration 1:

Forward Pass:

$$\hat{y} = 1 \cdot x + 1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].$$

Compute the loss:

$$L = \frac{1}{10} \sum_{i=1}^{10} (\hat{y}_i - y_i)^2.$$

Substitute:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{10}[-1.5, 0.3, 0.6, 1.2, 3.9, 5.2, 5.7, 6.3, 8.3, 10].$$

Backward Pass:

$$\frac{\partial L}{\partial w} = \frac{1}{5} \sum_{i=1}^{10} (\hat{y}_i - y_i) x_i = 64, 28 \quad \text{and} \quad \frac{\partial L}{\partial b} = \frac{1}{5} \sum_{i=1}^{10} (\hat{y}_i - y_i) = 7, 8.$$

Linear Regression example

Iteration 1:

Forward Pass:

$$\hat{y} = 1 \cdot x + 1 = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].$$

Compute the loss:

$$L = \frac{1}{10} \sum_{i=1}^{10} (\hat{y}_i - y_i)^2.$$

Substitute:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{10}[-1.5, 0.3, 0.6, 1.2, 3.9, 5.2, 5.7, 6.3, 8.3, 10].$$

Backward Pass:

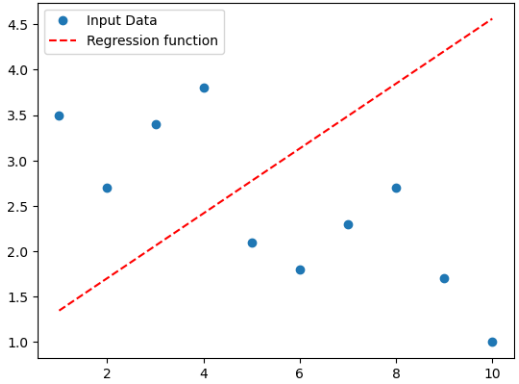
$$\frac{\partial L}{\partial w} = \frac{1}{5} \sum_{i=1}^{10} (\hat{y}_i - y_i) x_i = 64,28 \quad \text{and} \quad \frac{\partial L}{\partial b} = \frac{1}{5} \sum_{i=1}^{10} (\hat{y}_i - y_i) = 7,8.$$

Substitute values to compute gradients.

Parameter Updates:

$$w = w - 0.01 \cdot \frac{\partial L}{\partial w} = 1 - 0,6428 = 0,3572 \quad \text{and} \quad b = b - 0.01 \cdot \frac{\partial L}{\partial b} = 1 - 0,078 = 0,922.$$

Linear Regression example



Linear Regression example

Iteration 2:

Forward Pass:

$$\hat{y} = 0.3572 \cdot x + 0.922 = [1.3472, 1.7044, 2.0616, 2.4188, 2.776, 3.1332, 3.4904, 3.8476, 4.2048, 4.562].$$

Linear Regression example

Iteration 2:

Forward Pass:

$\hat{y} = 0.3572 \cdot x + 0.922 = [1.3472, 1.7044, 2.0616, 2.4188, 2.776, 3.1332, 3.4904, 3.8476, 4.2048, 4.562].$

Compute the loss:

$$L = \frac{1}{10} \sum_{i=1}^{10} (\hat{y}_i - y_i)^2.$$

Linear Regression example

Iteration 2:

Forward Pass:

$\hat{y} = 0.3572 \cdot x + 0.922 = [1.3472, 1.7044, 2.0616, 2.4188, 2.776, 3.1332, 3.4904, 3.8476, 4.2048, 4.562].$

Compute the loss:

$$L = \frac{1}{10}\sum_{i=1}^{10}(\hat{y}_i - y_i)^2.$$

Substitute:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{10}[-2.1528, -0.9956, -1.3384, -1.3812, 0.676, 1.3332, 1.1904, 1.1476, 2.5048, 3.562].$$

Linear Regression example

Iteration 2:

Forward Pass:

$\hat{y} = 0.3572 \cdot x + 0.922 = [1.3472, 1.7044, 2.0616, 2.4188, 2.776, 3.1332, 3.4904, 3.8476, 4.2048, 4.562].$

Compute the loss:

$$L = \frac{1}{10} \sum_{i=1}^{10} (\hat{y}_i - y_i)^2.$$

Substitute:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{10}[-2.1528, -0.9956, -1.3384, -1.3812, 0.676, 1.3332, 1.1904, 1.1476, 2.5048, 3.562].$$

Backward Pass:

$$\frac{\partial L}{\partial w} = \frac{1}{5} \sum_{i=1}^{10} (\hat{y}_i - y_i) x_i = 14.6744 \quad \text{and} \quad \frac{\partial L}{\partial b} = \frac{1}{5} \sum_{i=1}^{10} (\hat{y}_i - y_i) = 0.9092.$$

Substitute values to compute gradients.

Linear Regression example

Iteration 2:

Forward Pass:

$\hat{y} = 0.3572 \cdot x + 0.922 = [1.3472, 1.7044, 2.0616, 2.4188, 2.776, 3.1332, 3.4904, 3.8476, 4.2048, 4.562].$

Compute the loss:

$$L = \frac{1}{10}\sum_{i=1}^{10}(\hat{y}_i - y_i)^2.$$

Substitute:

$$\frac{\partial L}{\partial \hat{y}} = \frac{2}{10}[-2.1528, -0.9956, -1.3384, -1.3812, 0.676, 1.3332, 1.1904, 1.1476, 2.5048, 3.562].$$

Backward Pass:

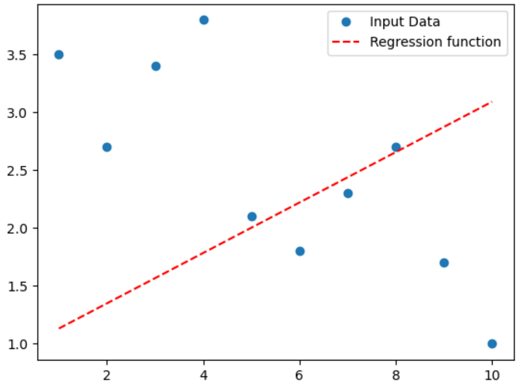
$$\frac{\partial L}{\partial w} = \frac{1}{5}\sum_{i=1}^{10}(\hat{y}_i - y_i)x_i = 14.6744 \quad \text{and} \quad \frac{\partial L}{\partial b} = \frac{1}{5}\sum_{i=1}^{10}(\hat{y}_i - y_i) = 0.9092.$$

Substitute values to compute gradients.

Parameter Updates:

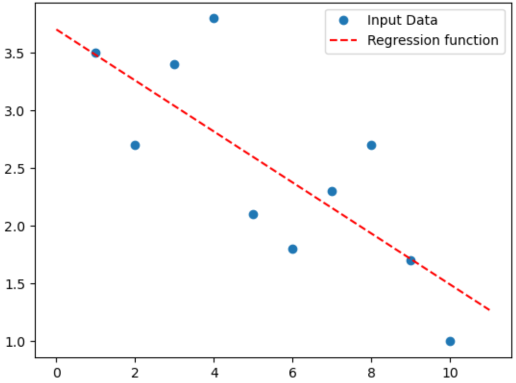
$$w = w - 0.01 \cdot \frac{\partial L}{\partial w} = 0,3572 - 0,146744 = 0,217936 \quad \text{and} \quad b = b - 0.01 \cdot \frac{\partial L}{\partial b} = 0,922 - 0,09092 = 0,914268.$$

Linear Regression example



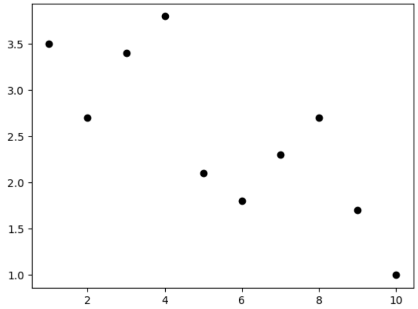
Linear Regression example

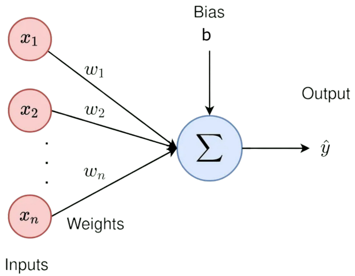
After 1000 iteration we obtain $w = -0.22115$ and $b = 3.70474$



Exercise

We retake the last example but this time we multiple weights.
x =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and y =[3.5, 2.7, 3.4, 3.8, 2.1, 1.8, 2.3,2.7, 1.7, 1]

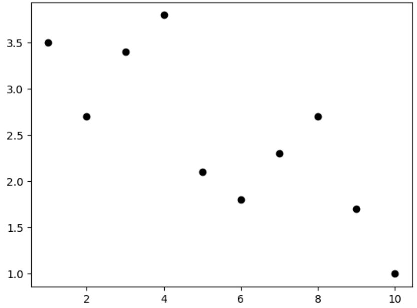


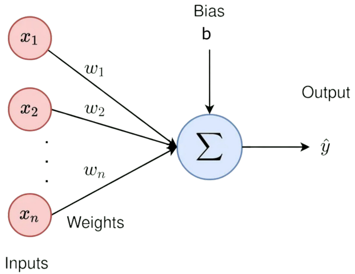


We assume that $w = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $b = 1$ and the learning rate is $\eta = 0.01$.
Perform the first three iteration.

Exercise

We retake the last example but this time with a nonlinear activation function $g(z) = \frac{1}{1+e^{-z}}$.
x =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and y =[3.5, 2.7, 3.4, 3.8, 2.1, 1.8, 2.3,2.7, 1.7, 1]





We assume that $w = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $b = 1$ and the learning rate is $\eta = 0.01$. Perform the first three iteration.