CHAPTER 3

Multilayer Perceptron

Before you start learning about multilayered perceptron, you need to get a big-picture view of artificial neural networks. That's what I'll start with in this chapter.

Artificial Neural Network

An *artificial neural network* (ANN) is a computational network (a system of nodes and the interconnection between nodes) inspired by biological neural networks, which are the complex networks of neurons in human brains (see Figure 3-1). The nodes created in the ANN are supposedly programmed to behave like actual neurons, and hence they are artificial neurons. Figure 3-1 shows the network of the nodes (artificial neurons) that make up the artificial neural network.

© Navin Kumar Manaswi 2018
N. K. Manaswi, *Deep Learning with Applications Using Python*,
https://doi.org/10.1007/978-1-4842-3516-4_3

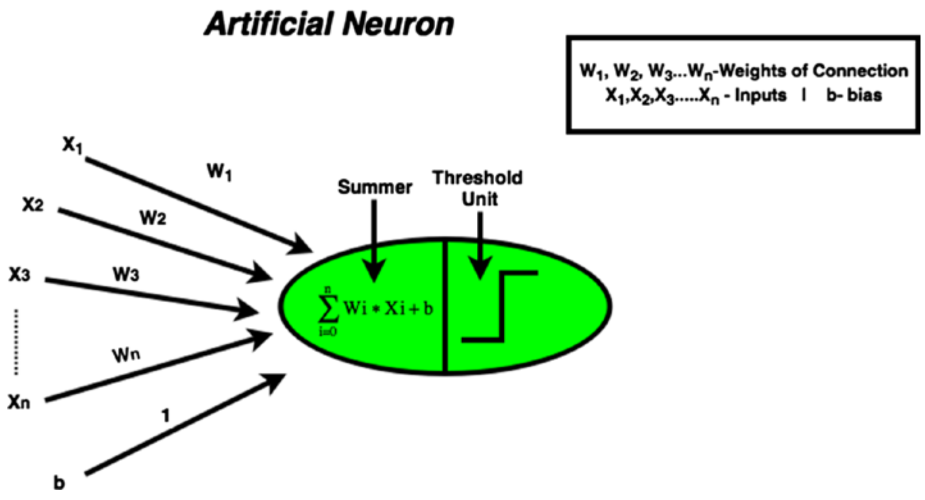

Artificial Neuron
$W_1, W_2, W_3 \ldots W_n$-Weights of Connection
$X_1, X_2, X_3 \ldots X_n$ - Inputs | b- bias
X_1
X_2
X_3
X_n
b
W_1
W_2
W_3
W_n
1
Summer
Threshold Unit
$\sum_{i=0}^{n} Wi * Xi + b$

Figure 3-1. *Artificial neural network*

The number of layers and the number of neurons/nodes per layer can be the main structural component of an artificial neural network. Initially, the weights (representing the interconnection) and bias are not good enough to make the decision (classification, etc.). It is like the brain of a baby who has no prior experience. A baby learns from experiences so as to be a good decision-maker (classifier). Experiences/data (labeled) helps the neural network of brains tune the (neural) weights and bias. The artificial neural network goes through the same process. The weights are tuned per iteration to create a good classifier. Since tuning and thereby getting the correct weights by hand for thousands of neurons is very time-consuming, you use algorithms to perform these duties.

That process of tuning the weights is called *learning* or *training*. This is the same as what humans do on a daily basis. We try to enable computers to perform like humans.

Let's start exploring the simplest ANN model.

A typical neural network contains a large number of artificial neurons called *units* arranged in a series of different layers: input layer, hidden layer, and output layer (Figure 3-2).

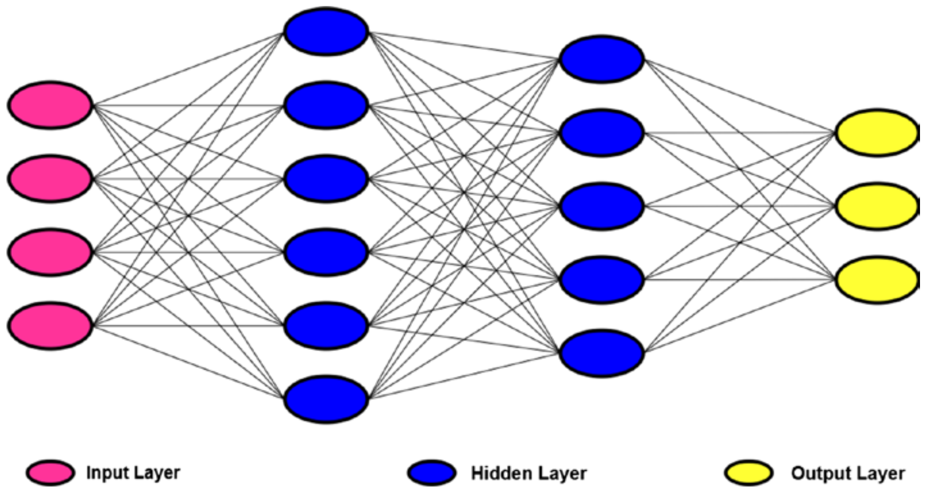


Figure 3-2. *Neural network*

Neural networks are connected, which means each neuron in the hidden layer is fully connected to every neuron in the previous input layer and to its next output layer. A neural network learns by adjusting the weights and biases in each layer iteratively to get the optimal results.

Single-Layer Perceptron

A *single-layer perceptron* is a simple linear binary classifier. It takes inputs and associated weights and combines them to produce output that is used for classification. It has no hidden layers. Logistic regression is the single-layer perceptron.

Multilayer Perceptron

A *multilayer perceptron* (MLP) is a simple example of feedback artificial neural networks. An MLP consists of at least one hidden layer of nodes other than the input layer and the output layer. Each node of a layer other than the input layer is called a *neuron* that uses a nonlinear activation function such as sigmoid or ReLU. An MLP uses a supervised learning technique called *backpropagation* for training, while minimizing the loss function such as cross entropy. It uses an optimizer for tuning parameters (weight and bias). Its multiple layers and nonlinear activation distinguish an MLP from a linear perceptron.

A multilayer perceptron is a basic form of a deep neural network.

Before you learn about MLPs, let's look at linear models and logistic models. You can appreciate the subtle difference between linear, logistic, and MLP models in terms of complexity.

Figure 3-3 shows a linear model with one input (X) and one output (Y).

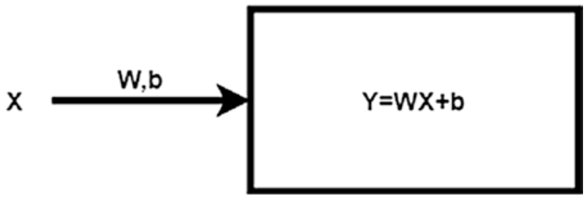


Figure 3-3. *Single-input vector*

The single-input model has a vector X with weight W and bias b. The output, Y, is WX + b, which is the linear model.

Figure 3-4 shows multiple inputs (X1 and X2) and one output (Y).

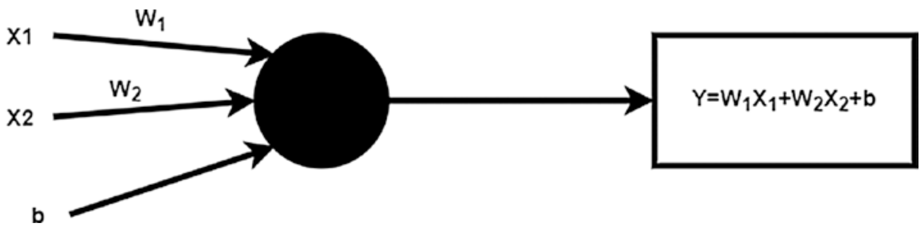


Figure 3-4. *Linear model*

This linear model has two input features: X1 and X2 with the corresponding weights to each input feature being W1, W2, and bias b. The output, Y, is W1X1 + W2X2 + b.

Logistic Regression Model

Figure 3-5 shows the learning algorithm that you use when the output label Y is either 0 or 1 for a binary classification problem. Given an input feature vector X, you want the probability that Y = 1 given the input feature X. This is also called as a *shallow* neural network or a *single-layer* (no hidden layer; only and output layer) neural network. The output layer, Y, is σ (Z), where Z is WX + b and σ is a sigmoid function.

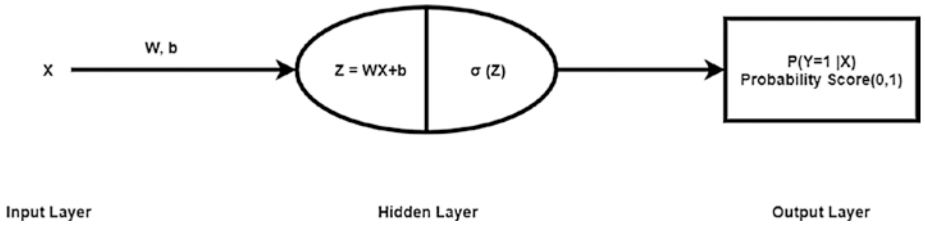


Figure 3-5. *One input (X) and one output (Y)*

Figure 3-6 shows the learning algorithm that you use when the output label Y is either 0 or 1 for a binary classification problem.

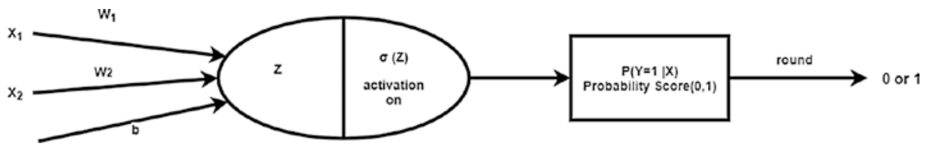


Figure 3-6. *Multiple inputs (X1 and X1) and one output (Y)*

Given input feature vectors X1 and X2, you want the probability that Y = 1 given the input features. This is also called a *perceptron*. The output layer, Y, is σ (Z), where Z is WX + b.

$$\begin{bmatrix} X1 \\ X2 \end{bmatrix} \rightarrow \begin{bmatrix} W1 & W2 \\ W3 & W4 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix} \rightarrow \sigma\left(\begin{bmatrix} W1 * X1 + W2 * X2 + b1 \\ W3 * X1 + W4 * X2 + b2 \end{bmatrix} \right)$$

Figure 3-7 shows a two-layer neural network, with a hidden layer and an output layer. Consider that you have two input feature vectors X1 and X2 connecting to two neurons, X1' and X2'. The parameters (weights) associated from the input layer to the hidden layer are w1, w2, w3, w4, b1, b2.

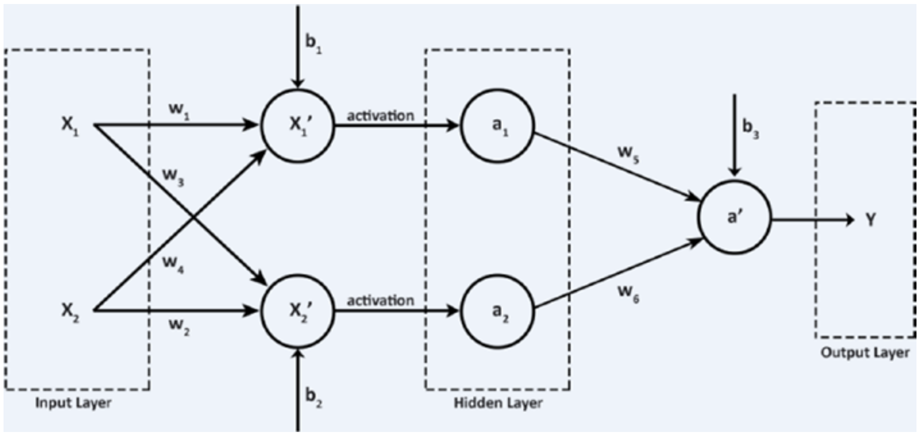


Figure 3-7. *Two-layer neural network*

X1′ and X2′ compute the linear combination (Figure 3-8).

$$\begin{bmatrix} X1' \\ X2' \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w3 & w4 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix}$$

(2×1)(2×2)(2×1)(2×1) is the dimension of the input and hidden layers.

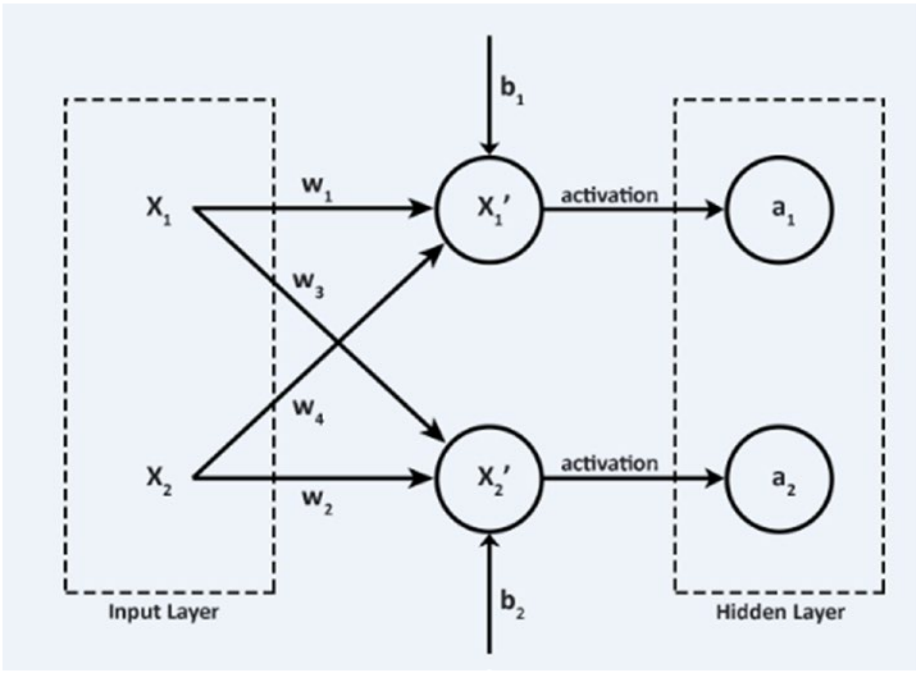


***Figure 3-8.** Computation in the neural network*

The linear input X1′ and X2′ passes through the activation unit a1 and a2 in the hidden layer.

a1 is σ (X1′) and a2 is σ(X2′), so you can also write the equation as follows:

$$\begin{bmatrix} a1 \\ a2 \end{bmatrix} = \sigma \begin{bmatrix} X1' \\ X2' \end{bmatrix}$$

The value forward propagates from the hidden layer to the output layer. Inputs a1 and a2 and parameters w5, w6, and b3 pass through the output layer a' (Figure 3-9).

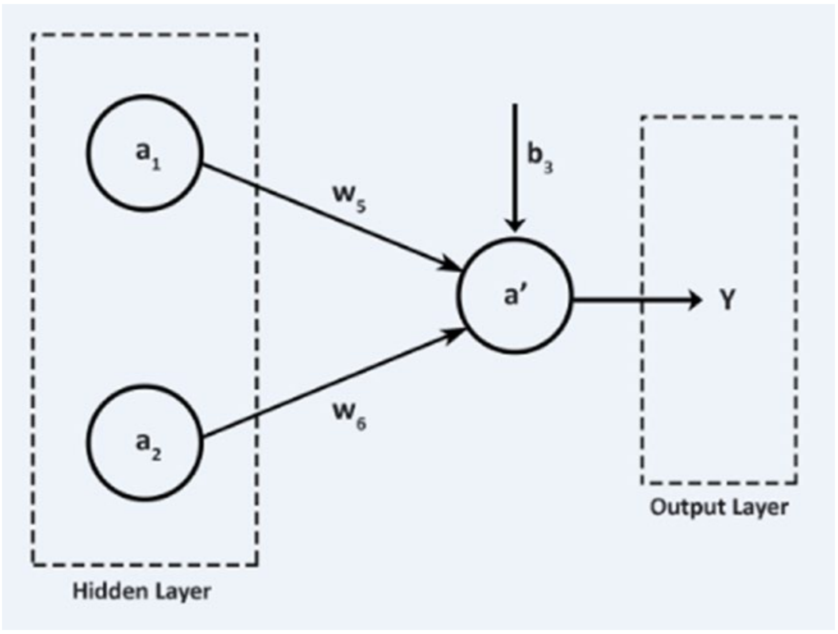


Figure 3-9. *Forward propagation*

$\text{a}' = \begin{bmatrix} w5 & w6 \end{bmatrix} \begin{bmatrix} a1 \\ a2 \end{bmatrix} + \begin{bmatrix} b3 \end{bmatrix}$ creates a linear combination of (w5*a1 + w6*a2) + b3, which will pass through a nonlinear sigmoid function to the final output layer, Y.

$$y = \sigma(\text{a}')$$

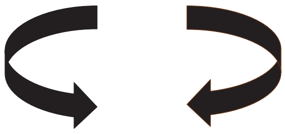
Let's say the initial model structure in one dimension is Y = w*X + b, where the parameters w and b are weights and bias.

Consider the loss function L(w, b) = 0.9 for the initial value of the parameters w = 1 and b = 1. You get this output: y = 1*X+1 & L(w ,b) = 0.9.

The objective is to minimize the loss by adjusting the parameters w and b. The errors will be backpropagated from the output layer to the hidden layer to the input layer to adjust the parameter through a learning rate and optimizer. Finally, we want to build a model (regressor) that can explain Y in terms of X.

To start the process of build a model, we initialize weight and bias. For convenience, w = 1, b = 1 (Initial value), (optimizer) stochastic gradient descent with learning rate (α = 0.01).

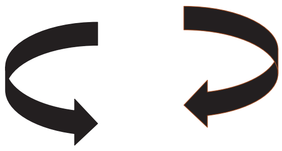
Here is step 1: Y = 1 * X + 1.



1.20 0.35

The parameters are adjusted to w = 1.20 and b = 0.35.

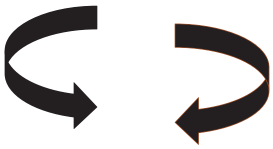
Here is step 2: Y1 = 1.20*X + 0.35.



1.24 0.31

The parameters are adjusted to w = 1.24 and b = 0.31.

Here is step 3: Y1 = 1.24*X + 0.31.



1.25 0.30

After some iterations, the weight and bias become stable. As you see, the initial changes are high while tuning. After some iterations, the change is not significant.

L(w, b) gets minimized for w = 1.26 and b = 0.29; hence, the final model becomes the following:

$$Y = 1.26 * X + 0.29$$

Similarly, in two dimensions, you can consider the parameters, weight matrix and bias vector.

Let's assume that initial weight matrix and bias vector as $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

You iterate and backpropagate the error to adjust w and b.

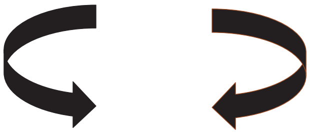
$Y = W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * [X] + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is the initial model. Weight matrix (2x2) and bias matrix(2x1) are tuned in each iteration. So, we can see change in weight and bias matrices

Here is step 1:

$$W = \begin{bmatrix} 0.7 & 0.8 \\ 0.6 & 1.2 \end{bmatrix}, B = \begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix}$$

Here is step 2:

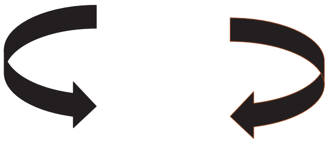
$$\begin{bmatrix} 0.7 & 0.8 \\ 0.6 & 1.2 \end{bmatrix} \begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix}$$



$$W = \begin{bmatrix} 0.6 & 0.7 \\ 0.4 & 1.3 \end{bmatrix}, B = \begin{bmatrix} 2.8 \\ 3.8 \end{bmatrix}$$

Here is step 3:

$$\begin{bmatrix} 0.6 & 0.7 \\ 0.4 & 1.3 \end{bmatrix} \begin{bmatrix} 2.8 \\ 3.8 \end{bmatrix}$$



You can notice change in weight matrix(2x2) and bias matrix(2x1) in the iteration.

$$W = \begin{bmatrix} 0.5 & 0.6 \\ 0.3 & 1.3 \end{bmatrix}, B = \begin{bmatrix} 2.9 \\ 4.0 \end{bmatrix}$$

The final model after w and b are adjusted is as follows:

$$\mathrm{Y} = \begin{bmatrix} 0.4 & 0.5 \\ 0.2 & 1.3 \end{bmatrix} * \left[X\right] + \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$$

In this chapter, you learned how weight and bias are tuned in each iteration while keeping the aim of minimization of loss functions. That is done with the help of optimizers such as stochastic gradient descent.

In this chapter, we have understood ANN and MLP as the basic deep learning model. Here, we can see MLP as the natural progression from linear and logistic regression. We have seen how weight and bias are tuned in every iteration which happens in backpropagation. Without going into details of backpropagation, we have seen the action/result of backpropagation. In next two chapters, we can learn how to build MLP models in TensorFlow and in keras.