CHAPTER 3

Feed Forward Neural Networks

In this chapter we will cover some key concepts around feedforward neural networks. These concepts will serve as a foundation as we cover more technical topics in depth.

At an abstract level, a Neural Network can be thought of as a function $f_\theta : x \to y$, which takes an input $x \in \mathbb{R}^n$ and produces an output $y \in \mathbb{R}^m$, and whose behavior is parameterized by $\theta \in \mathbb{R}^p$. So for instance, f_θ could be simply $y = f_\theta(x) = \theta \cdot x$.

We will start by looking at the structure of a neural network, followed by how they are trained and used for making predictions.

Unit

A unit is the basic building of a neural network; refer to Figure 3-1. The following points should be noted:

1. A unit is a function that takes as input a vector $x \in \mathbb{R}^n$ and produces a scalar.
2. A unit is parameterized by a weight vector $w \in \mathbb{R}^n$ and a bias term denoted by b.
3. The output of the unit can be described as

$$f\left(\sum_{i=1}^{n} x_i \cdot w_i + b\right)$$

   where $f : \mathbb{R} \to \mathbb{R}$ is referred to as an activation function.
4. A variety of activation functions may be used, as we shall see later in the chapter; generally, it's a non-linear function.

N. Ketkar, *Deep Learning with Python*, DOI 10.1007/978-1-4842-2766-4_3

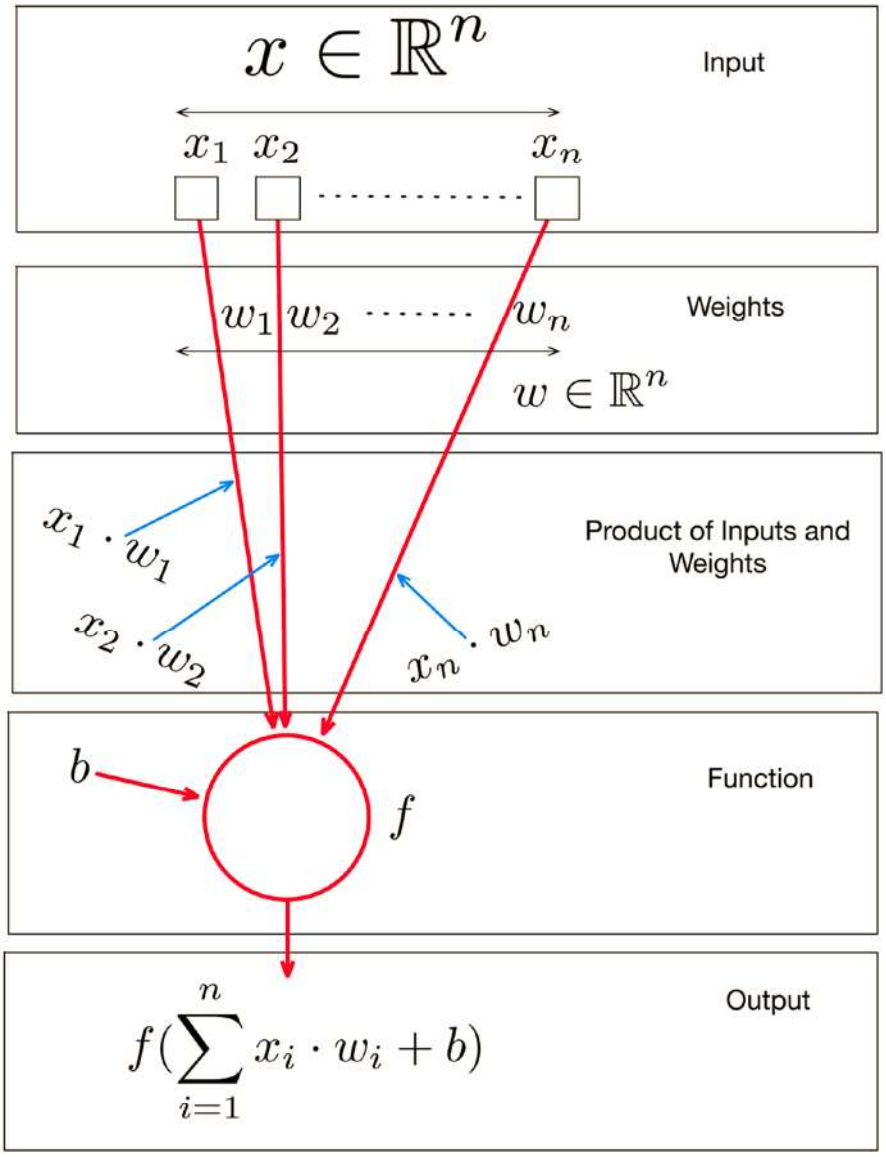

Figure 3-1. *A unit in a neural network*

Overall Structure of a Neural Network

Neural Networks are constructed using the unit as a basic building block (introduced earlier); refer to Figure 3-2.

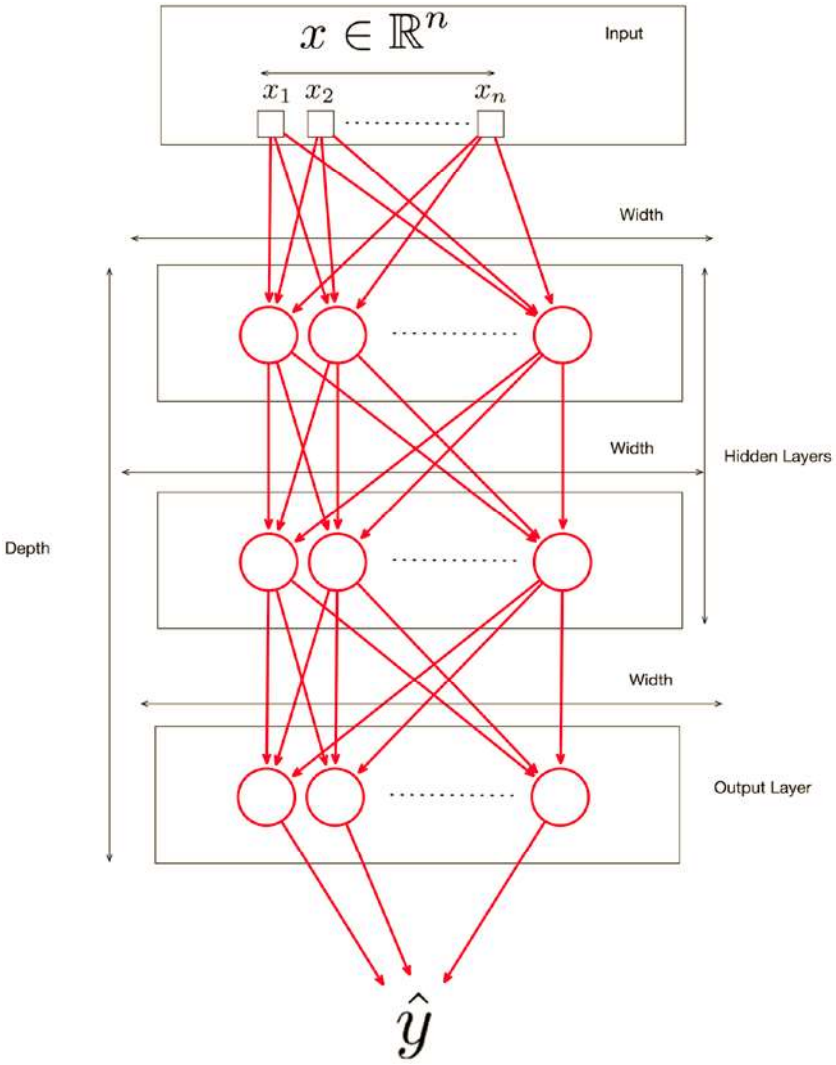

Figure 3-2. *Structure of a Neural Network*

The following points are to be noted:

1. Units are organized as layers, with every layer containing one or more units.
2. The last layer is referred to as the output layer.
3. All layers before the output layers are referred to as hidden layers.
4. The number of units in a layer is referred to as the width of the layer.

5. The width of each layer need not be the same, but the dimension should be aligned, as we shall see later in the chapter.
6. The number of layers is referred to as the depth of the network. This is where the notion of *deep* (as in deep learning) comes from.
7. Every layer takes as input the output produced by the previous layer, except for the first layer, which consumes the input.
8. The output of the last layer is the output of the network and is the prediction generated based on the input.
9. As we have seen earlier, a neural network can be seen as a function $f_\theta : x \rightarrow y$, which takes as input $x \in \mathbb{R}^n$ and produces as output $y \in \mathbb{R}^m$ and whose behavior is parameterized by $\theta \in \mathbb{R}^p$. We can now be more precise about θ; it's simply a collection of all the weights w for all the units in the network.
10. Designing a neural network involves, amongst other things, defining the overall structure of the network, including the number of layers and the width of these layers.

Expressing the Neural Network in Vector Form

Let us now take a look at the layers of a Neural Network and their dimensions in a bit more detail. Refer to Figure 3-3; the following points should be noted:

1. If we assume that the dimensionality of the input is $x \in \mathbb{R}^n$ and the first layer has p_1 units, then each of the units has $w \in \mathbb{R}^n$ weights associated with them. That is, the weights associated with the first layer are a matrix of the form $w_1 \in \mathbb{R}^{n \times p_1}$. While this is not shown in the diagram, each of the p_1 units also has a bias term associated with it.
2. The first layer produces an output $o_1 \in \mathbb{R}^{p_1}$ where $o_i = f\left(\sum_{k=1}^{n} x_k \cdot w_k + b_i\right)$. Note that the index k corresponds to each of the inputs/weights (going from $1 \ldots n$) and the index i corresponds to the units in the first layer (going from $1..p_1$).
3. Let us now look at the output of the first layer in a vectorised notation. By vectorised notation we simply mean linear algebraic operations like vector matrix multiplications and computation of the activation function on a vector producing a vector (rather than scalar to scalar). The output of the first layer can be represented as $f(x \cdot w_1 + b_1)$. Here we are treating the input $x \in \mathbb{R}^n$ to be of dimensionality $1 \times n$, the weight matrix w_1 to be of dimensionality $n \times p_1$, and the bias term to be a vector of dimensionality $1 \times p_1$. Notice then that $x \cdot w_1 + b$ produces a vector of dimensionality $1 \times p_1$ and the function f simply transforms each element of the vector to produce $o_1 \in \mathbb{R}^{p_1}$.
4. A similar process follows for the second layer that goes from $o_1 \in \mathbb{R}^{p_1}$ to $o_2 \in \mathbb{R}^{p_2}$. This can be written in vectorised form as $f(o_1 \cdot w_2 + b_2)$. We can also write the entire computation up to layer 2 in vectorised form as $f(f(x \cdot w_1 + b_1) \cdot w_2 + b_2)$.

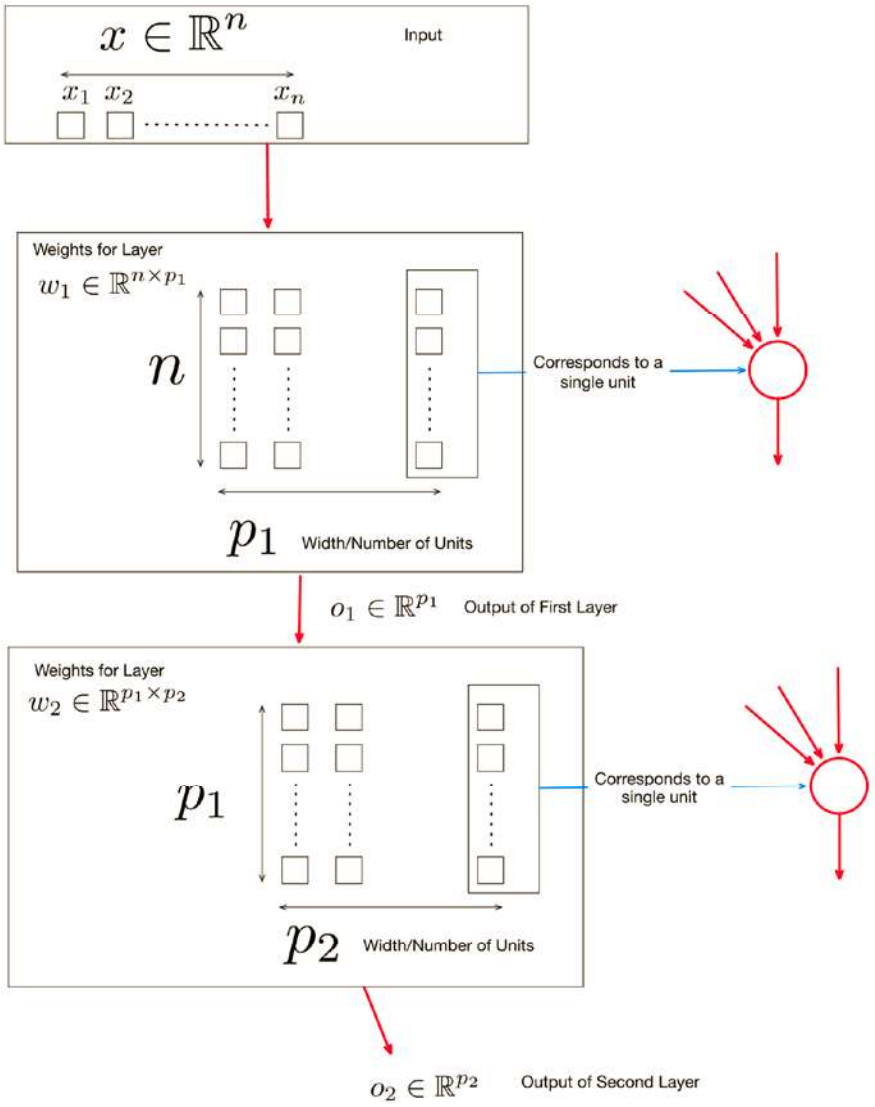

Figure 3-3. *Expressing the Neural Network in Vector Form*

Evaluating the output of the Neural Network

Now that we have looked at the structure of a Neural Network, let's look at how the output of the neural network can be evaluated against labeled data. Refer to Figure 3-4. The following points are to be noted:

1. We can assume that our data has the form $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$ where $x \in \mathbb{R}^n$ and $y \in \{0, 1\}$, which is the target of interest (currently this is binary, but it may be categorical or real valued depending on whether we are dealing with a multi-class or regression problem, respectively).
2. For a single data point we can compute the output of the Neural Network, which we denote as $\hat{y}$.

3. Now we need to compute how good the prediction of our Neural Network $\hat{y}$ is as compared to y. Here comes the notion of a loss function.
4. A loss function measures the disagreement between $\hat{y}$ and y which we denote by l. There are a number of loss functions appropriate for the task at hand: binary classification, multi-classification, or regression, which we shall cover later in the chapter (typically derived using Maximum Likelihood).
5. A loss function typically computes the disagreement between $\hat{y}$ and y over a number of data points rather than a single data point.

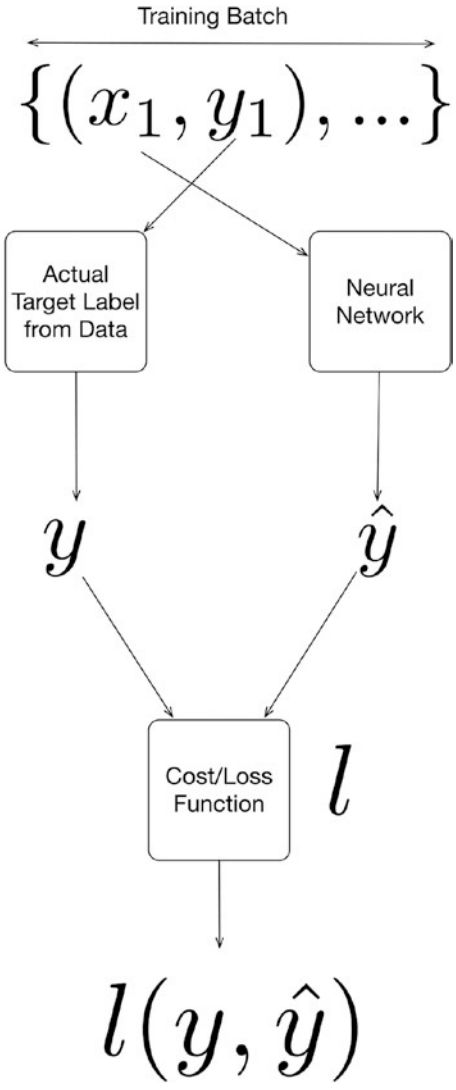

Figure 3-4. *Loss/Cost function and the computation of cost/loss w.r.t, a neural network*