Arnav Goyal

Data Engineering & Machine Learning

Winter 2024 / AISE 3010

Preface

The purpose of this document is to act as a comprehensive note for my understanding on the subject matter. I may also use references aside from the lecture material to further develop my understanding, and these references will be listed here.

This document should eventually serve as a standalone reference for learning or review of the subject matter. There is also a lot of organization within these documents, please refer to the table of contents within your PDF viewer for ease of navigation.

References

- Provided Course Material & Lecture Notes
- Neural Networks and Learning Machines - 3e - S. Haykin

The Perceptron

In this chapter we will be going over the architecture of the most basic **neural network (NN)**, called the **perceptron** [1].

[1] Also called Rosenblatt's perceptron

The Neuron

The most basic unit of a perceptron is called the **neuron** [2]. The structure of a neuron can be seen in Figure 1, each neuron possesses the following.

[2] More specifically, the McCulloch-Pitts model of a neuron

- Some **input features** $x_1 \dots x_n$
- A corresponding **weight** for each feature $w_1 \dots w_n$
- A **bias** b
- An **activation function** $\varphi(.)$
- An **output** y

A neuron essentially performs the following operation, which can be seen from the signal flow graph.

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

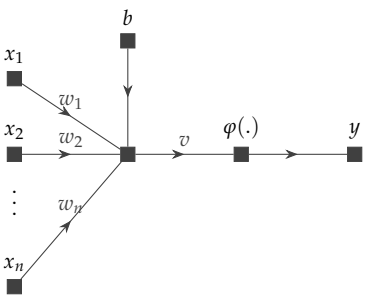

Figure 1: Signal Flow Graph of A Neuron

The Basic Perceptron

The perceptron was initially built with just a single neuron, and was limited to performing **binary classification** or **linear regression.**

- In binary classification, each training sample (X, y) contains some data $X = x_1, x_2, \ldots$ and a label $y \in \{-1, +1\}$
- In linear regression the dependent variable is $y \in \mathbb{R}$

Overall, the perceptron was created to deal with binary classification problems.

- This means that both classes c_1 and c_2 need to be **linearly seperable** [3]
- The perceptron is essentially trying to find a **linear separator** between the two classes[4], such that $\sum w_i x_i + b = 0$.

[3] We need to be able to draw a hyperplane that clearly seperates every instances from both classes, such that only instances from the same class are on a distinct side of the hyperplane

[4] Given that the activation function, $\varphi(.) \leftarrow \text{sgn}(.)$

Multi-Layered Perceptrons

The most basic perceptron was basically a single-layered NN, but now we will discuss the **multi-layered perceptron (MLP)** which uses more layers, with possibly more neurons in each layer[5].

[5] We do this to overcome the perceptrons practical limitations

Before I continue, lets clarify some terminology.

- The **input layer** is the first layer with neurons that each accept 1 feature. In total, an input layer with k neurons can accept k features.

- The **output layer** is the last layer, containing neurons that spit out one output each. In total an output layer with h neurons produces an output with dimensionality of h.
- The **hidden layers** are the other layers sandwiched in between the input and output. When we say a "an n-layered network", we are referring to the amount of hidden layers in the network.

This is illustrated in Figure 2.

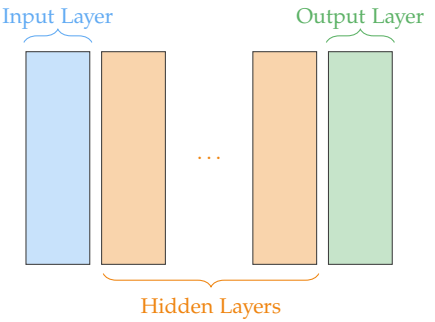

Figure 2: Layers in an MLP

In order to construct a MLP we must ensure the following.

- The activation function $\varphi_i(.)$ of each neuron i is a **differentiable** and non-linear function.
- The network contains at least one layer that is **hidden** from the input and output nodes (hidden layers).
- The network exhibits a high degree of **connectivity** through synaptic weights of the network.

These conditions ensure that the network becomes complex, but also so complex that we struggle to understand the behavior of the network.

Batch & Iterative Learning

As we work our way up to understanding the training of the MLP we should account for the two methods of teaching the MLP, called **batch learning** and **iterative learning** [6]

- Batch learning consists of presenting a batch of training examples to the network at the same time
- Iterative learning consists of presenting one training example to the network at a time.

Batch learning has the advantages of providing an accurate-estimation of the gradient vector, and allowing a parallelization of the learning process. While iterative learning is simple to implement and can utilize redundancies in the dataset.

[6] In the textbook this is called *online learning* but this was obviously confusing so I changed it for this note.

Backpropagation

Now we are going to dive into the complex algorithm called **backpropagation** which is the way modern NNs are trained. There are two cases for this algorithm

- When a neuron is an output neuron
- When a neuron isn't an output neuron

We will be starting with the former, but be warned! this is going to be a painful and math-intensive chapter.

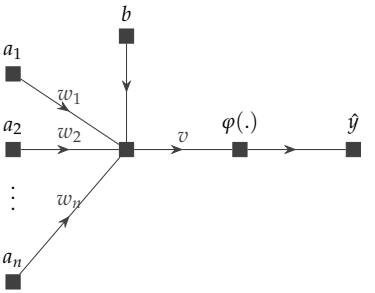

Figure 3: An Output Neuron

The Error & Cost Function

Consider a neuron j that is located in the output layer of an MLP. It has internal structure as shown by Figure 3. We also define the **error** ε, that gives a measure of how far our current output $\hat{y}$, is from our desired output y. For the purposes of this chapter:

$$\varepsilon = y - \hat{y}$$

We should also define a **cost function** ℓ as the **total error energy** of the whole network.[7]

$$\ell = \frac{1}{2}\varepsilon^2$$

[7] This is usually defined at some iteration n but this is left out for brevity.

The Output Neuron

We are concerned with finding out how the loss function ℓ changes with respect to a certain weight w_k. In other words we are tasked with finding the derivative.[8]. We can do this through the **chain rule**.

[8] We are tasked with finding $\frac{\partial \ell}{\partial w_k}$

$$\frac{\partial \ell}{\partial w_k} = \frac{\partial \ell}{\partial \varepsilon} \cdot \frac{\partial \varepsilon}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_k}$$

We can break down this derivative even further using Figure 3, resulting in Equation 1. We can even extend this to see how we can change the biases, which is given in Equation 2.

$$\frac{\partial \ell}{\partial w_k} = \frac{\partial \ell}{\partial \varepsilon} \cdot \frac{\partial \varepsilon}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v} \cdot \frac{\partial v}{\partial w_k} \tag{1}$$

$$\frac{\partial \ell}{\partial b_k} = \frac{\partial \ell}{\partial \varepsilon} \cdot \frac{\partial \varepsilon}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v} \cdot \frac{\partial v}{\partial b_k} \tag{2}$$

Analytically solving Equation 1 gives the following result:

$$\frac{\partial \ell}{\partial w_k} = -\varepsilon a_k \varphi'(v)$$

A Hidden Neuron

When a neuron j is located within a hidden layer of the MLP, there is no desired response for that neuron, accordingly the error signal for that neuron would have to be determined recursively, and work backwards from all neurons to where the neuron j is located[9].

[9] This is where the term *backpropagation* comes from