Lecture 3
Neural Networks: from The Ground up

By
Dr. Muhammad Alrabeiah
Electrical Engineering Dept., KSU
Spring 2022

February 1st, 2022

Disclaimer These notes are still under development, and they have not been subjected to proper review and revision.

Contents

1 Artificial Neurons **3**

1 Rosenblatt's Perceptron 3

2 Artificial Neuron: The Building Block 5

2.1 Activation functions 6

2 Multilayer Perceptron Networks **8**

1 Multilayer Perceptron Networks 8

1.1 A Single Layer 9

1.2 Multiple layers 10

2 Regression and Classification with Neural Networks 11

3 Universal Approximation Property 11

3.1 Bird's-eye View 12

3.2 Universal Approximation Theorem for Neural Networks 13

3.3 Expressiveness of MLP networks 13

Chapter 1

Artificial Neurons

Suggested reading:
Section 1.7 of Chapter 4 in [1] and Sections 1.1 and 1.2 of Chapter 1 in [2].

We have, so far, focused on some fundamental tasks in supervised machine learning, i.e., regression and classification, and we have discussed some learning algorithms to perform those tasks. All those algorithms, in general, share one defining characteristic, which is their linearity. Real-world problems, however, can rarely be handled with "linear algorithms," and, thus, we need more powerful algorithms to deal with those complex problems. "artificial neural network" is the algorithm of choice in this lecture and the rest of this course to deal with complex real-world problems. This chapter aims to lay the necessary groundwork for discussions on neural networks (and, later, on deep learning); it discusses the basic architecture of artificial neurons, classical and modern.

1 Rosenblatt's Perceptron

One of the early attempts to build an artificial neuron is that of Frank Rosenblatt, an American psychologist. His idea on modeling neurons has led to the design and implementation of the first learning machine [1], which implemented a simple neural network using dedicated hardware. His work and interests in artificial intelligence and neural networks has recently earned him the title of "father of deep learning"[3]—Of course along with doctors Hinton, LeCun, and Bengio.

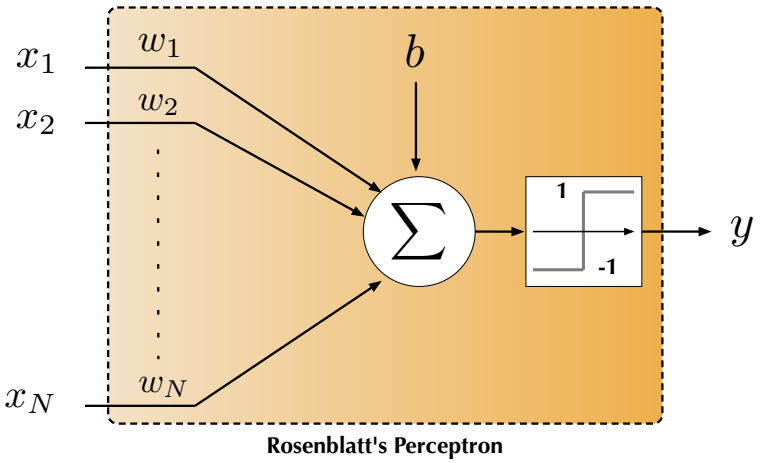

Figure 1.1: Architecture of the Rosenblatt's perceptron

Following the short stroll down history lane, we will take a step towards a more technical discussion on artificial neural networks. The *perceptron algorithm* is recognized by many as the first artificial neuron, and for obvious reason, it is commonly referred to as the *Rosenblatt's perceptron*. The algorithm is a type of linear classifier that learns a decision boundary for binary classification tasks. Figure 1.1 depicts a schematic for the perceptron. Its architecture is composed of an augmentation of two functions: (i) a linear transformation, and (ii) a thresholding function. Mathematically, the perceptron function could be expressed by

$$y = \varphi\left(b + \sum_{i=1}^{N} w_i x_i\right) = \varphi(\mathbf{w}^T\mathbf{x} + b), \tag{1.1}$$

where

$$\varphi(\mathbf{w}^T\mathbf{x} + b) = \begin{cases} +1, & \mathbf{w}^T\mathbf{x} + b \geq 0 \\ -1, & \mathbf{w}^T\mathbf{x} + b < 0 \end{cases} \tag{1.2}$$

As Equation 1.1 indicates, the perceptron is a way to learn a linear decision boundary. It differs from the least squares classifier in the way it learns that boundary. We learned in Lecture 2 Chapter 2 that least squares learns a classifier using a closed-form solution which is the normal equation; however, the perceptron follows an iterative approach to learn the decision boundary. In each iteration, it uses the *mis-classified* inputs in the previous iteration to correct the position of the line, i.e., the parameters $\mathbf{w}$ and w_0. The loss function of the

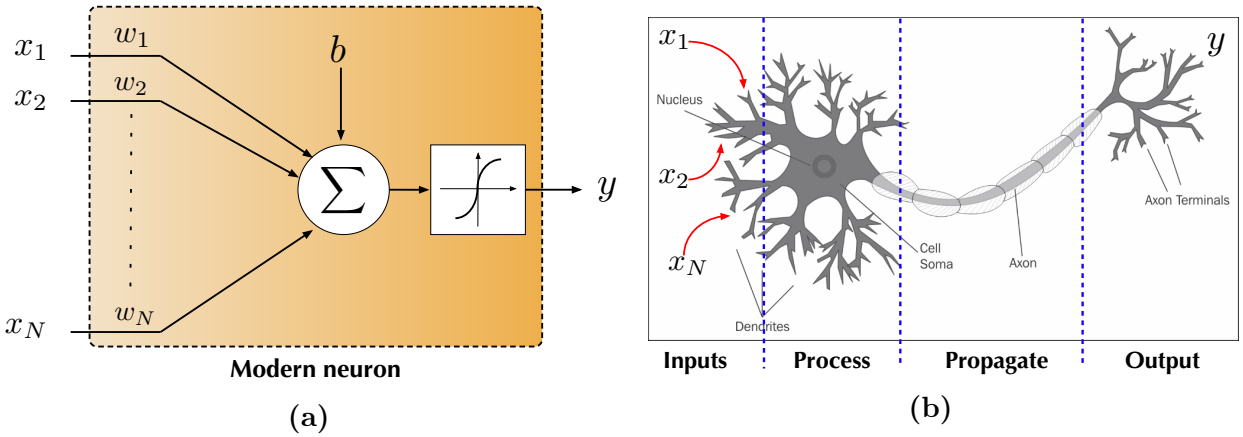

Figure 1.2: (a) Architecture of the modern neuron, and (b) A drawing of a pyramidal neuron

algorithm is also very different from that of the least squares classifier. It is expressed by

$$\min_{\mathbf{w},b} \mathcal{L} = \min_{\mathbf{w},b} -\sum_{i\in\mathcal{M}} (\mathbf{w}^T\mathbf{x} + b)t \tag{1.3}$$

where $\mathcal{M}$ is the set of mis-classified inputs. Note that the loss in Problem 1.3 is always positive (Why is that?). The perceptron algorithm attempts to minimize the loss by iteratively adjusting the parameters $\mathbf{w}$ and b so that the set $\mathcal{M}$ shrinks over iterations. It does so using a *gradient descent* algorithm, but let's not get ahead of ourselves; we will discuss gradient descent some time around Lecture 4, inshallah.

2 Artificial Neuron: The Building Block

The perceptron algorithm, and architecture, has a major limitation, its inability to deal with non-linearly separable patterns. This drawback restricts the use of Rosenblatt's perceptron to simple tasks—as we know, linear classification is not common in real-world problems. To deal with such issue, artifficial neurons tweak the perceptron architecture to include a *differentiable activation function* that replaces the thresholding function (see Figure 1.2a), and, then, they use this neuron to build neural networks. We will leave the discussion about neural network to the next chapter, and focus on the artificial neuron here. Equation 1.1 still models the neuron operation, but with a different function, i.e., $\varphi(.)$.

The artificial neuron mimics the architecture of an actual biological neuron, but not the

functionality [2]. Let's try to untangle that sentence with the help of Figure 1.2b. This is called a pyramidal neuron cell, and it is the most common type of cortical neurons. Artificial neurons are inspired by the architecture of the pyramidal cell. The input terminals in an artificial neuron model dendrites in the cell. The *synaptic* weights, *bais*, and activation function (i.e., $\mathbf{w}$, b, and $\varphi(.)$, respectively) model the nucleus and axon. Finally, the output terminal models a single axon terminal. The resemblance between the two stops right here; the operation performed by the artificial neuron does not quite model that of the pyramidal cell. The artificial neuron basically applies a linear combiner followed by the activation function while the pyramidal cell performs a slightly more complex operation that produces *spikes* at the output instead of a constant value.

We will not spend more time dissecting biological neurons and highlighting similarities and differences with artificial ones, for this will stray us away from the subject of the course. Instead, we will explore types of activation functions that are dominant in the field of neural networks and deep learning.

2.1 Activation functions

It is a non-linear operation applied on the output of the combiner. It serves two objectives:

1. Provides control over the range of the output (activation range).
2. It helps expand the representational ability of neural networks beyond linearly-separable patterns.

Below are three of the most popular activations:

- **Sigmoid function:** this is one of the oldest choices for activation. It performs a *squashing* operation to the output of the linear combiner such that is guarantees a neuron output y that ranges from 0 to 1. Mathematically, it is given by

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

- **Hyperbolic tangent function (tanh):** this activation still performs a squashing operation on the output of the linear combiner, but it has a larger range compared to

the sigmoid function, going from -1 to 1. It is expressed as

$$\varphi(x) = \frac{e^x - e^{-1}}{e^x + e^{-1}} \tag{1.5}$$

- **Rectified Linear Unit (ReLU):** this function is one of the most popular in modern-day deep neural networks. There are a few reasons for that, and we will discuss the most important of those in Lecture 5. However, for now, we need to learn that ReLU does not subscribe to the squashing operation as the former two functions do. Formally, ReLU is described by

$$\varphi(x) = \max\{0, x\} \tag{1.6}$$

The above three activations are only the most popular, yet they are not the most advanced. Designing non-linearities is a major research direction in neural networks and deep learning, for they determine important properties like the universal approximation property, which we will touch upon at the end of this lecture. One important point to bring up here is that all three activations are not *parameterized*; they are not learnable. This is a common feature among most state-of-the-art activation functions, Leaky ReLU and Exponential Linear Unit (ELU) to name two.

Chapter 2

Multilayer Perceptron Networks

Suggested readings:
Sections 2 to 4 of Chapter 6 in [4] and Section 12 of Chapter 4 in [2].

In the pervious chapter, we have learned about the artificial neuron which is the building block of Artificial Neural Networks (ANNs). This means we have the necessary background to dive into a lengthy albeit exciting discussion on ANNs. After all, they are the core of this course and modern deep learning. The discussion will be spread across three lectures, and the start here will be with Multilayer Perceptron (MLP) networks, which are also known as Fully-Connected (FC) or dense networks. MLP could be considered the most basic and, to some extent, most generic form of neural networks.

1 Multilayer Perceptron Networks

There is quite little that one can do using a single neuron, for it is only suitable to deal with linear regression and classification problems. A method to circumvent that limitation is to stack neurons into a layer and stack multiple layers to construct a "network." This is exactly what we are going to be doing in this chapter.

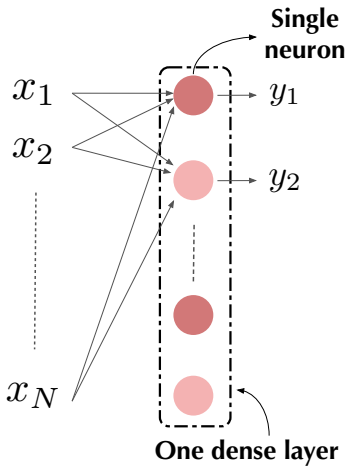

Figure 2.1: Single fully-connected (dense) layer

1.1 A Single Layer

A single layer in a neural network is the result of stacking several neurons. The number of neurons stacked to form a layer is usually referred to as th *breadth* of the layer and we will denoted by Q. Each of those neurons sees the whole input vector all at once, and, hence, this layer is commonly referred to as fully-connected or dense layer—we will use these names exchangeably throughout the course. Figure 2.1 depicts an illustration of a dense layer. Other defining features for a dense layer are the distinct parameters and biases and the matching activation functions. The q-th neuron in a layer has its own parameter vector $\mathbf{w}_q$ and b_q that are different form those of other neurons, and it applies activation function $\varphi(.)$ that is the same across all neurons. Mathematically, we can express the operation of a single layer as follows

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_Q \end{bmatrix} = \varphi\left(\begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \vdots \\ \mathbf{w}_Q^T \end{bmatrix} \mathbf{x} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_Q \end{bmatrix}\right) \tag{2.1}$$

$$\mathbf{y} = \varphi(\mathbf{W}^T\mathbf{x} + \mathbf{b}), \tag{2.2}$$

where $\varphi(.)$ here is an element-wise activation function that guarantees all neurons apply the same activation; $\mathbf{W} \in \mathbb{R}^{N\times Q}$ is the matrix encompassing all the weights of a layer, i.e., each

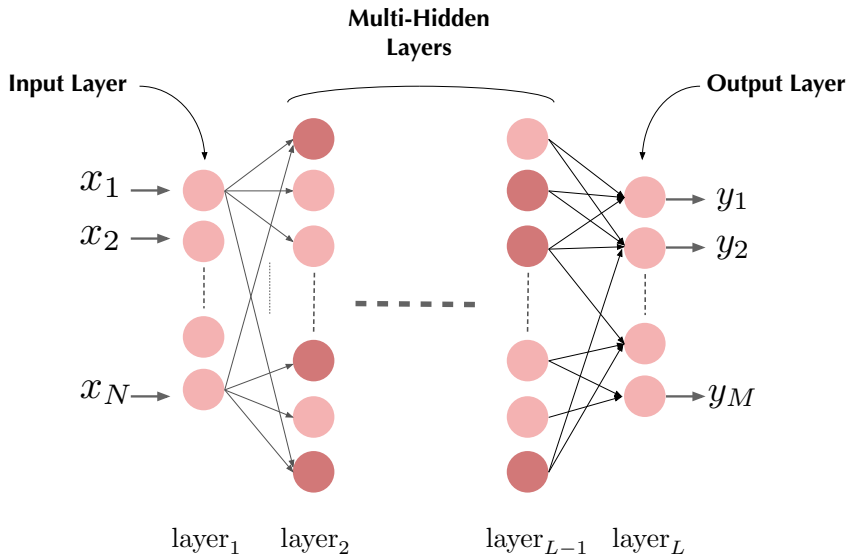

Figure 2.2: Schematic of multilayer perceptron network

column $\mathbf{w}_q$ belongs to one neuron; $\mathbf{b}$ is a vector of Q biases, one per neuron; $\mathbf{x}$ is the input vector; and $\mathbf{y}$ is the output vector.

1.2 Multiple layers

Stacking neurons in a single layer could be argued to be useless for non-linear tasks (whether classification or regression), for the linearity are the same and are fixed, i.e., non-parametric. This limits their ability to represent different non-linear input-output relations—more about this in Section 3.1. Hence, it is common to use multiple layers and build a neural network. These layers are usually stacked sequentially like those in Figure 2.2 where the output of one layer is the input of the other one. The number of stacked layers is commonly referred to as the *depth*. In the literature of neural networks, the layer seeing the input $\mathbf{x}$ is either referred to as the "input layer" or "first hidden layer," and in this course, we will use the former as illustrated in Figure 2.2. The last layer, just before the output, is referred to as the output layer. We express the input-output relation as follows

$$\mathbf{y} = \varphi_L\left(\mathbf{W^{(L)}}^T\varphi_{L-1}\left(\mathbf{W^{(L-1)}}^T\varphi_{L-2}\left(\dots\mathbf{W^{(2)}}^T\varphi_1\left(\mathbf{W^{(1)}}^T\mathbf{x}+\mathbf{b}^{(1)}\right)+\mathbf{b}^{(2)}\right)+\mathbf{b}^{(L-1)}\right)+\mathbf{b}^{(L)}\right), \tag{2.3}$$

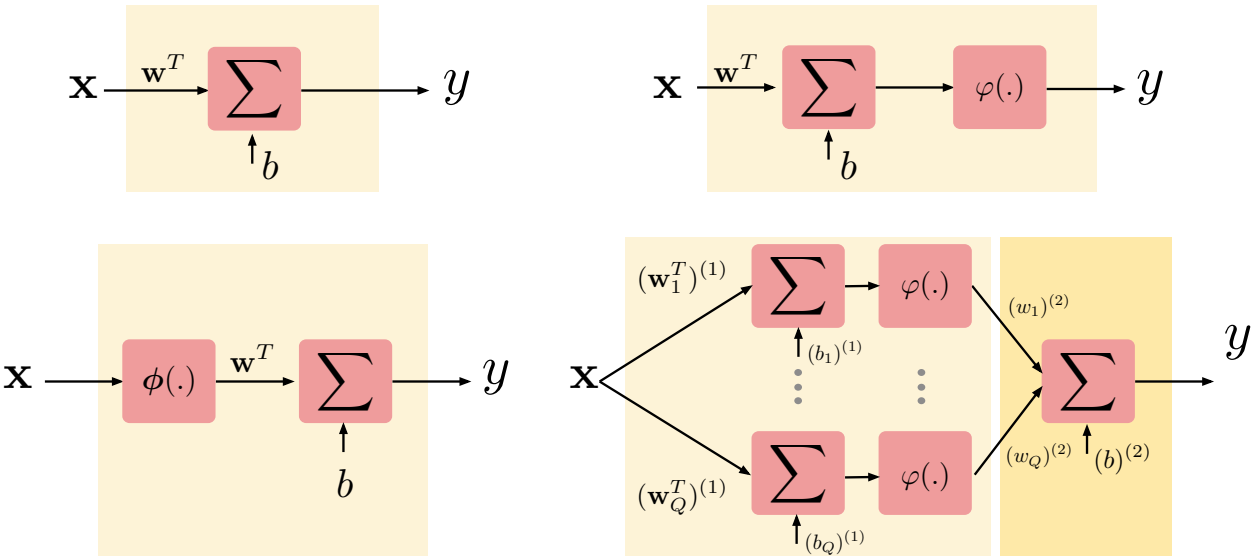

Figure 2.3: illustration of input-output and output-parameters relations in all models discussed from Lecture 1 till now.

which encompasses the sequential nature of neural networks. It is very obvious that Equation 2.3 is quite missy and not pleasant to the eye. Hence, we will re-write it in the following compact form

$$\mathbf{y} = (\varphi_L \circ \varphi_{L-1} \circ \cdots \circ \varphi_1)(\mathbf{x}), \tag{2.4}$$

where each $\varphi_i,\ \forall i \in \{1, \ldots, L\}$, encloses its own $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$.

The stacking of these layers is what gives neural networks the ability to learn non-linear functions. We will explore this ability further in the next section when we touch upon the universal approximation theorem for neural networks.

2 Regression and Classification with Neural Networks

3 Universal Approximation Property

All the learning algorithms discussed prior to this lecture is limited to learning linear patterns, whether in regression or classification. Only in Assignment 1 have we discussed an approach for learning non-linear patterns using linear regression models and basis functions. Neural networks is another way to do that, and a very powerful one for that matter. We are not

going to delve into proving why it is powerful and how—not in a formal way anyhow—but we will try to develop an intuitive understanding to their *representational abilities*. This discussion will lead us to the universal approximation property of neural networks, which is one way to explain their popularity and superiority over other algorithms.

3.1 Bird's-eye View

Let's consider Figure 2.3, which depicts an illustration of most of the models we have encountered so far. We will discuss each one below.

- **Model 1:** The top-left model is a graph representation of the linear model with which we started this course. One could notice that this model is equivalent to a modern neuron with a linear activation function. Because of its linear architecture, it is incapable of representing anything but linear x-y relations.

- **Model 2:** To deal with such limitation, the bottom-left model introduces a non-linear non-parameterized transformation to the observed variables right before they are passed to the linear combiner (a weighted sum with a bias will be called weight combine henceforth). The transformation is referred to as the basis function, and it is one way to deal with non-linear x-y relations without jeopardizing the linearity of the model. The catch with such model is also its point of strength, *the reliance on non-parametric functions*. These functions are determined by the machine learning engineer (hyper-parameter) and not derived from the data. These functions give raise to handcrafted (or engineered) feature extraction, which has been the core for classical machine learning. Overall, such feature extraction approach has been shown to be inferior to feature learning approaches, more will be said soon.

- **Model 3:** Top-right model shows the modern neuron we discussed in Chapter 1 Section 2. It represents a form of symmetry to Model 2; the basis function is removed and we are back to feeding the input to a linear combiner. The output of the combiner is, then, passed through a non-linear activation function, which results in a non-linear relation between both input and output as well as output and parameters. This again presents a way to overcome the linear model shortcoming. However, similar to Model 2, the point of strength is also a point of weakness. The non-linear activation is non-parametric

and it limits what input-output relations could be represented (Think of a sigmoid and what patterns it could represent!)

- **Model 4:** The bottom-right model in Figure 2.3 represents a natural evolution for Models 2 and 3; it gets the best of both worlds, non-linear feature extraction and non-linear activation outputs. The first layer (light yellow box) deploys several neurons to produce a hidden variable, which has a non-linear relation with the input. The second layer (dark yellow box) is a simple parametric linear combiner that learns to combine the elements of the hidden variables. This layered architecture helps capture non-linear input-output relations and maintain a mix of linear and non-linear parameter relations. We can think of the first layer as a parametrized basis function that could be learned by training while the second layer acts as a simple combiner. This architecture is the core of fully-connected networks, and it brings about an interesting property, the universal approximation ability. We will briefly talk about it in the next section.

3.2 Universal Approximation Theorem for Neural Networks

The ability of FC networks to represent non-linear pattern is consolidated by the universal approximation theorem [5]. We are not going to dive deep into the inner workings of that theorem as it does not have direct impact on designing neural networks (the focus of this course). It provides a vessel for proving the powerful representational capability of neural networks. We are, instead, going to state the main conclusion of the theorem. A two-layer network with non-linear activations, like sigmoid and ReLU, in the first layer and linear ones in the second layer can learn any continuous and smooth function given the right first layer breadth. This means by varying the breadth of the first layer, a two-layer network *have the ability to represent any function.* Of course, there are some conditions the functions and the choice of activation need to satisfy, but we will not discuss them here.

3.3 Expressiveness of MLP networks

Bibliography

[1] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[2] S. Haykin, *Neural Networks and Learning Machines*. Prentice Hall, 2009. [Online]. Available: https://books.google.com.sa/books?id=K7P36lKzI_QC

[3] C. C. Tappert, "Who is the father of deep learning?" in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2019, pp. 343–348.

[4] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[5] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.