An Undergraduate's Explanation of the Multilayer Perceptron: Mathematical Concepts and a Python3 Implementation

Jared Frazier*†

2022
January

1 Preamble

The purpose of the present document is to explain and implement the major mathematical constructs/concepts behind feedforward neural networks, specifically the multilayer perceptron. This includes the layers that compose such networks, the cost (aka loss, error, or objective) function and activation functions, the forward pass through the network, the computation of gradients via backpropagation (a concept that is often "handwaved" to the extreme or explained in so much detail as to be utterly confusing–at least in the author's experience), and the update of model parameters via mini-batch stochastic gradient descent. If the ideas such as *layer* and *backpropagation* are entirely unfamiliar to you, then the author encourages you to visit 3Blue1Brown's Deep Learning YouTube Series [1] and peruse the first few chapters of texts such as *Deep Learning* (free, online) [2], *Neural Networks and Deep Learning* (free, online) [3], *Hands-on Machine Learning with Scikit-Learn, TensorFlow and Keras 2ed* (buy) [4], and/or *Deep Learning with Python 2ed* (buy) [5]. The present document is not intended to be a comprehensive overview of neural networks nor an extremely in-depth explanation but rather a document that highlights certain concepts that the author found confusing or ambiguous when he was learning about neural networks.

The format of explanations in the present document will essentially alternate between mathematics and concrete implementations using the Python3 programming language with the NumPy library. Note that the implementations here are not intended to be optimal. If you would like optimal implementations, the author encourages you to use a machine learning API such as TensorFlow or PyTorch, the tutorials for which will abstract and make easily usable many of the concepts elucidated in the present document.

*Department of Computer Science (2021-2022), Middle Tennessee State University
†Not endorsed by or affiliated with any of the authors or entities associated with references

2 Introduction

The neural network (function approximator) is just a chain of geometric transformations (functions) each parametrized by a weight matrix $W \in \mathcal{R}^{n_h \times n_x}$ and a bias vector $b \in \mathcal{R}^{n_h}$ on the input vector $x \in \mathcal{R}^{n_x}$. The geometric transformations of the neural network are encapsulated by connecting layers (e.g., dense/fully connected layers) together. A neural network has L total layers and the current layer l receives the output from the previous layer $(l-1)$. Note that n_x is the number of features (or independent variables) in the input and n_h is the number of hidden units in the current layer. The following subsections will breifly elucidate the claims in of the first setnence of this section and the corresponding notation.

2.1 Parametrized Functions

The author assumes you know what a function is; however, the term *parametrized* is one that appears often in deep learning literature and should be well-understood by the student. Consider a generic quadratic function [6] as

$$f(x) = ax^2 + bx + c \tag{1}$$

The *variable* x is an *argument* to the function f that has *parameters* a, b, and c. The parameters determine the behavior of the function (e.g., the steepness of slope, intercepts, shape, etc.) while the variable can take on some range of values. When a variable that takes on a particular value is passed as an argument to the function with defined parameters, the result is some other value y if $y = f(x)$. This explanation of a function should not be anything new; however, the *parameters* are quantities of particular interest for neural networks since the parameters are the quantities that are *learned* by the neural network over time. What it means to learn parameters will be explaned later.

A neural network can be denoted as a function h with parameters W and b of a variable x. This statement can be compactly written as $h_{W,b}(x)$. The subscript with W and b means that the weights W and biases b are parameters of the neural network h. The claim that a neural network is a chain of functions is useful later during the updating of the parameters of the network. But to briefly illustrate the idea of chaining functions, the generic quadratic function in Equation 1 is decomposed into units called *atomic functions.*

$$g_a(x) = ax^2 \tag{2}$$

$$u_b(x) = bx \tag{3}$$

$$f_{a,b,c}(x) = g_a(x) + u_b(x) + c \tag{4}$$

Decomposing functions into their constituent atomic functions is useful for applying rules of calculus–the basis of parameter learning via the backpropagation algorithm illustrated later.

2.2 Operand Types

The input x is not a single value as is conceived in the elementary formulations described above. Rather, the input x is a list of values referred to as a *column vector*. Each element of the vector is the value a particular feature, or independent variable, could take on. The shape of the vector x is important to understand since the functions and operations performed by the neural network (dot product, Hadamard product, addition, etc.) restrict their vector/matrix operands to particular shapes. When using the term *vector*, the author is always referring to a *column vector* unless otherwise specified. Also, note that $x \in \mathcal{R}^{n_x}$ indicates that x is a vector with n_x elements and the j^{th} is a real number. For example, the below vector x is shown and a common vector operation known as transposition (converts a *column vector* to a *row vector* and is denoted with a superscript of $\top$) is also shown.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{(n_x)} \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{(n_x-1)} \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{(n_x-1)} \end{bmatrix}^\top \tag{5}$$

Many programming languages assign the first element of a vector the index 0; this notation is shown above in addition to the more standard mathematical notation where the first element begins with the index 1. For the remainder of this document, the author will use the index 0 assumption since the author's implementation of the the neural network will use the Python programming language. If you wish to implement the same algorithms in a language such as R or Wolfram Mathematica, be wary of this index discrepancy. Consequently, with index beginning at 0, the last index of a vector with n_x elements will be $(n_x - 1)$... and woe is the programmer who commits an off-by-one error.

Lastly, a matrix W represents the weight of edges between the k^{th} input neuron of n_x total input neurons (i.e., neurons in the previous layer) and the j^{th} hidden neuron of n_h^l total hidden neurons. A weight matrix looks similar to the vector, except rather than having a single column, a matrix has a rows and columns–looking like a table. Vectors can be referred to as rank-1 tensors, matrices as rank-2 tensors, so and so forth for multiple index "lists" in higher dimensions. A sample weight matrix $W \in \mathcal{R}^{n_h \times n_x}$ is shown below.

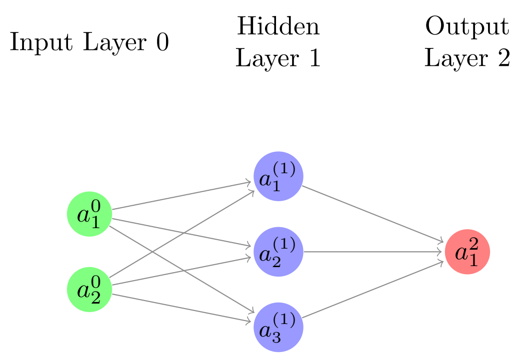
$$\begin{bmatrix} W_{00} & W_{01} & W_{02} & \cdots & W_{0(n_x-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ W_{(n_h-1)0} & W_{(n_h-1)1} & W_{(n_h-1)2} & \cdots & W_{(n_h-1)(n_x-1)} \end{bmatrix} \tag{6}$$

3 The Multilayer Perceptron

Here the author defines the operations that occur for a multilayer perceptron (MLP). Note that the MLP can sometimes refer to any class of feedforward neural network, that is a network that applies affine transformations and activation functions to input from a previous layer in the network.

The affine transformation, which is the most fundamental transformation of the densely/fully connected layers that exist in the MLP is simply $W^l_{jk} a^{l-1}_k + b^l_j$. Here, the activation a^{l-1}_k denotes the activation of the k^{th} neuron of the previous layer l. Importantly, the input layer has no activation function ϕ associated with it. Moreover, the inner dimensions of the matrix product Wx match, that is the subscripts k are "adjacent" to one another. While you may observe that the activation vector $a \in \mathcal{R}^{n_a}$ is clearly not a matrix, numerical libraries will often treat a vector $v \in \mathcal{R}^{n_v}$ as equivalent to a matrix with a single column (i.e., $V \in \mathcal{R}^{n_v \times 1}$) for the purposes of performing fast matrix-matrix calculations.

Until now the discussion of the MLP has been entirely in abstract mathematical notation, so now a visual of a single layer (meaning single hidden layer) MLP is shown.



(7)

3.1 Single Hidden Layer Neural Network

$$\begin{aligned} NeuralNet_\theta(X) &= \sigma(ReLU(XW^{[1]} + b^{[1]})W^{[2]} + b^{[2]}) \\ &= \sigma(g(ReLU(w(X)))) \\ A^L &= NeuralNet_\theta(X) \qquad \text{Activation matrix } A \text{ for last layer } L \end{aligned} \tag{8}$$

where σ, $ReLU$, g, and w are defined as follows:

$$\begin{aligned} \sigma(t) &= \frac{1}{1+e^{-t}} \\ ReLU(t) &= max(0,t) \\ g_{\theta^{[2]}}(A) &= AW^{[2]} + b^{[2]} \\ w_{\theta^{[1]}}(A) &= AW^{[1]} + b^{[1]} \\ u_{\theta^{[l]}}(A) &= AW^{[l]} + b^{[l]} \qquad \text{General form of } g \text{ and } w \text{ for } l^{th} \text{ layer} \end{aligned} \tag{9}$$

3.2 Neural Network Prediction (Forward Pass)

$$\hat{y} \leftarrow NeuralNet_{\theta}(X) \tag{10}$$

3.3 Mean Squared Error Loss Function

$$\begin{aligned}\mathcal{L}_{\theta}(X) &= \frac{1}{N}\sum_{i=1}^{N}(\hat{y}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}(a^{(i)} + y^{(i)})^2 \quad a^{(i)} \text{ is the activation vector of the last layer } L \text{ for the } i^{th} \text{ input}\end{aligned} \tag{11}$$

3.4 Gradient Update

$$\theta_i \leftarrow \theta_i - \eta(\nabla_{\theta}\mathcal{L}_{\theta}(\hat{y}, y)) \tag{12}$$

References

[1] 3Blue1Brown: Grant Sanderson. *But What is a Neural Network — Chapter 1, Deep Learning*. YouTube. 2017. URL: `https://www.youtube.com/watch?v=aircAruvnKk`.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: `http://www.deeplearningbook.org`.

[3] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: `http://neuralnetworksanddeeplearning.com/index.html`.

[4] Aurelien Geron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly, 2020.

[5] Francois Chollet. *Deep Learning with Python, Second Edition*. Manning Publications, 2021.

[6] nmasanta (https://math.stackexchange.com/users/623924/nmasanta). *What is the difference between variable, argument and parameter?* Mathematics Stack Exchange. URL: `https://math.stackexchange.com/q/3562026`.