CIS5300 - Speech and Language Processing - Neural Networks - Chapter 7 Notes

Jonathon Delemos - Chris Callison Burch

July 18, 2025

0.1 Abstract

Neural networks are a fundamental computational tool for language processing, and a very old one. They are called neural because their origins lie in the McCulloch-Pitts neuron (McCulloch and Pitts, 1943), a simplified model of the biological neuron as a kind of computing element that could be described in terms of propositional logic. But the modern use in language processing no longer draws on these early biological inspirations. Instead, a modern neural network is a network of small computing units, each of which takes a vector of input values and produces a single output value. In this chapter we introduce the neural net applied to classification

0.2 7.1 - Units

The building block of a neural network is a single computational unit. At it's heart, a neural unit is taking a weighted sum of it's inputs, ith one additional term in the sum called a bias term.
A lot of the information here is covered in Chapter 4. By **nonlinear**, we mean the function cannot be expressed as a straight light in the input space. With nonlinear activation functions, you can bend and curve the decision boundary, cpature complex patterns like XOR, and represent images, language, and time-series data in meaningful ways.

Review Sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

In practice, the sigmoid function is not commonly used as an activation function. A function that is very similar but almost always better is tanh function, the tanh functionis a variant that ranges from -1 to +1. TanH is a tangent hyperbolic function.n

$$y = tanh(z) = \frac{e^z - e^z}{e^z + e^{-z}}$$

The simplest activation function, and perhaps the most commonly used, is the rectified linear unit, also called the ReLU. It's just the same as z whe z is positive, and 0 otherwise.

$$y = ReLU(z) = max(z, 0)$$

Activation functions compute nonlinear outputs from each neuron. These outputs form activation vectors that are passed forward through the network. During training, the network adjusts its weights by backpropagating errors, using the activation function derivatives to compute gradients.

Activation Units

Think of these a little bit like decision gates. If the score is too low, the gate shuts. If the score is high enough, the gate opens and the package goes to the next part of the factory.

- ReLU: If positive, do this.
- Sigmoid: If the score is high, open it fully, if low, keep it shut.
- Tanh: Let some positive or negative flow through - cap the extremes.

0.3 7.2 - The XOR Problem

Using a single-layer neural network as our system for determining 1 or 0, we cannot compute the correct results for XOR as XOR isn't linearly seperable. The solution is to use a multi-layer neural network, one with a hiddden layer that combines multiple linear and nonlinear transformations.

This is a system that allows for intermediate decisions.

Logic-Based XOR Computation

The XOR function can be expressed using basic logic gates as:

$$\text{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

Two-Layer Network

Layer 1: Compute the following intermediate values:

$$h_1 = x_1 \wedge \neg x_2$$
$$h_2 = \neg x_1 \wedge x_2$$

Layer 2: Final output using OR:

$$y = h_1 \vee h_2$$

Truth Table

x_1	x_2	$\neg x_1$	$\neg x_2$	$h_1 = x_1 \wedge \neg x_2$	$h_2 = \neg x_1 \wedge x_2$	$y = h_1 \vee h_2$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

In summary, we say that *xor* is not a linearly seperable function.

0.4 7.3 - Feedforward Neural Networks

A feedforward network is a multilayer network in wich the units are connected with no cycles; the outputs from units in each layer are passed to units in the next higher layer, and no inputs are passed back to lower layers.

- input unit
- hidden unit
- output unit

These are the foundations of machine learning. We use input layers, hidden layers, and output layers to determine the final result of our decision.
The output of the hidden layer, the vector **h** is:

$$h = \sigma(W \cdot x + b)$$

Like the hidden layer, the output layer has a weight matrix **U**. The weight matrix is multiplied by the input parameter **hidden layer *h***.

$$z = \mathbf{Uh}$$

Here are the final equations for a feedforward network with a single hidden layer, which takes input vector $\mathbf{x}$, outputs a probability distribution $\mathbf{y}$, and is parameterized by weights $\mathbf{W}$ and $\mathbf{U}$ and a bias vector $\mathbf{b}$.

$$h = \sigma(W \cdot x + b) z = \mathbf{U}\mathbf{h} y = softmax(z)$$