Midterm Report

Nicholas Wotton

6 December 2019

Chapter 1

Exploring Neural Networks

1.1 Defining a Neural Network

A neural network is a type of function. It is initialized specifically with input-output pairs. Given sets of these input-output pairs, the function changes to try and replicate the relationship between them. For each pair it makes a best guess at the relationship between them, updating with each new pair. The idea is to create a trained network that can replicate a function given an input. Ideally, complex functions for which there are no mathematical closed solutions can be solved using this network.

The network is made up of layers of Neurons. A layer can have any number of neurons and a network can contain any number of layers. For example, let us create a neural network with 1 layer with 1 neuron. The network guesses at what factor the input, x needs to be scaled by and what constant, if any, needs to be added to achieve $f(x)$. This is just the problem

$$ax + b = f(x) \tag{1.1}$$

Where we are solving for a and b. If we add another layer of 1 neuron to Equation 1.1, our probelm expands to

$$(a_1x + b_1)a_2 + b_2 = f(x) \tag{1.2}$$

Where a_1, a_2 and b_1, b_2 refer to the neuron and constant for each layer respectively. Multiple layers allow for multiple scalars and constants to be employed by the network trying to guess the function. Notice specifically how the second layer's a_2 term, the scalar, multiplies the constant from the first layer. This can allow for approximation of non-linear functions.

However, layers of single neurons are not typically used. Adding more neurons in a layer allows for more combinations to be tried at once, that is every neuron in the first layer can try a different scalar. Theoretically, any function could be approximated with a Neural Network using only a single layer com-

posed of a massive number of neurons. The main problem with this is the computational mass and required time.

Figure 1.1 shows a more complex version of a simple neural network.

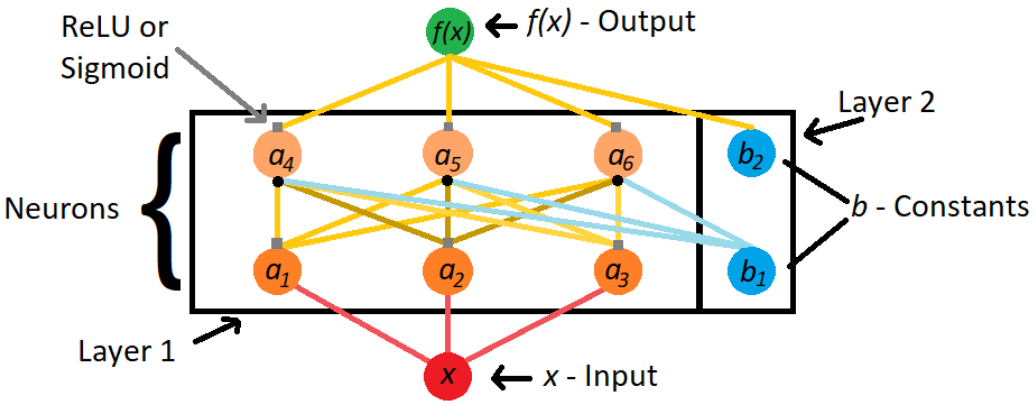

Figure 1.1: A Basic Neural Network Diagram

Walking through the diagram, we begin with our input-output pair, $(x, f(x))$. Each a_i in the first layer modifies x in some way. Then, $a_i x$ passes through an activation function. The two most common types are ReLU and Sigmoid. They are

$$ReLU : \phi(x) = max\{0, x\}$$

$$Sigmoid : \phi(x) = \frac{1}{1 + e^{-x}}$$

After $a_i x$ passes through the activation function, the constant is added and the whole thing is submitted to the next layer. Each neuron in layer 2 then modifies this $a_i x + b_1$ in a different way. Finally, each product of the neurons of the second layer pass through the activation function again. This leads them to a single value. During the training of the model, this value, $N(x)$ is then compared to the true given value, $f(x)$. The loss is assessed and adjustments are made using an optimizer function to minimize the loss.

Mathematically, we can represent this using linear algebra:

$$\phi_2 \left(\left(\phi_1 \left(\begin{bmatrix} x \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \right) + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \right) \begin{bmatrix} a_4 \\ a_5 \\ a_6 \end{bmatrix} \right) + \begin{bmatrix} b_4 \end{bmatrix} = \begin{bmatrix} f(x) \end{bmatrix} \tag{1.3}$$

1.2 An Example of a Linear Function and Neural Network

Here we will go through a simple example of using a neural network to reproduce a simple linear function. For our target function, we chose

$$f(x) = x + 2 \tag{1.4}$$

Next, we create the neural network. We use Python code, specifically the Pytorch package to do this:

```
#model
#nn.Linear
in_dim = 1
out_dim = 1

model = nn.Sequential(
    nn.Linear(in_dim, 30),
#    nn.ReLU(),
    nn.Linear(30, out_dim)
)
```

Notice that we have 2 layers of 30 neurons each. Notice also that we have removed the ReLU step from the network. Since our function is linear, having ReLU in the network negatively impacts the accuracy and efficiency of the model. After initializing the model, we create our training data. These are n randomly generated values that are passed through our function, $f(x)$ to create the input-output value pairs. Next, we need to define what we will use to measure accuracy in the iterative process of optimizing the network. Here we will just "cheat" by using the Mean Squared Error as our measure of improvement. That is, on each iteration we are computing

$$MSE = \sum_{i=1}^{n} (f(x) - \hat{y})^2$$

where $\hat{y}$ is the estimated value for the function from the Neural Network. Note that this is "cheating" since in more advanced circumstances we would not explicitly know the function f we are trying to replicate. The MSE and Newton's Method is then used to optimize the neural network. Typically, the training data is passed through a certain number of times to increase accuracy.

Once we have established the model and trained it, we need to test it. To do this, we randomly generate another set of points. We feed these to f and our neural network separately and compare the resulting values. The closer our predicted values to the actual values, the more accurate the network has become. The results of our optimized model are in Figure 1.2.

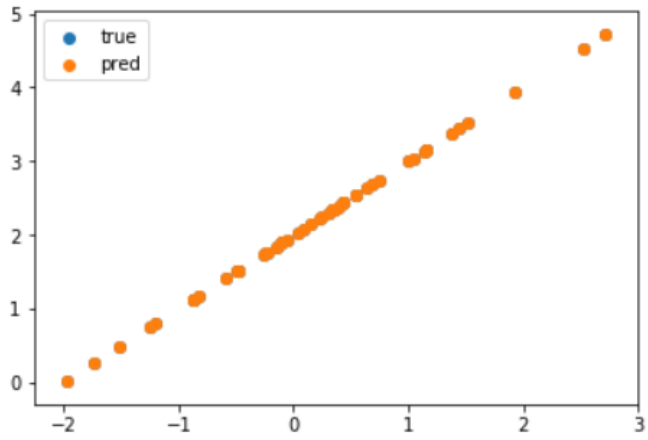

Figure 1.2: Actual and Predicted Values of $f(x) = x + 2$

Chapter 2

The Black-Scholes Model

The Balck-Scholes Model is a partial differential equation used for option pricing.

2.1 Brief Overview of Options

An option is a contract on an asset, typically a stock, that can be purchased and sold. There are two important types of options: Calls and Puts. There are also different varieties of these, but the simplest and most important for the Black-Scholes model is the European Option.

Owning an option gives the holder the right, but not the obligation, to sell or buy an asset at a pre-agreed upon price known as the Strike price, denoted K. More specifically, a European Call option gives the holder the right to buy an asset while a European Put option gives the holder the right to sell an asset. Mathemtacially, we have the Payoff for owning an option at time t is:

Call Options: $P_{Call}(t) = max\{0, S_t - K\}$

Put Options: $P_{Put}(t) = max\{0, K - S_t\}$

where S_t is the stock price at time t and K is the strike price.

Graphically, we have the payoffs for Call and Put respectively:

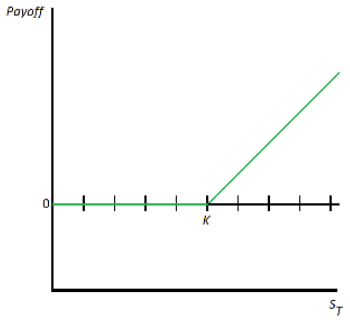

Figure 2.1: Payoff for European Call

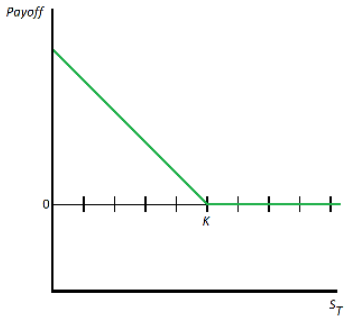

Figure 2.2: Payoff for European Put

For the Call Option, this means that if at expiry time T the price of the stock is below or equal to K, the holder will not exercise it, thus, their Payoff is \$0. However, if at expiry the price of the stock is greater than K, the holder does exercise the option, thus their Payoff is equal to the difference between S_T and K. For example, for a strike price $K = \$110$, if the stock price at time T is

- \$100, the holder makes \$0 since $K \geq \$100$.
- \$120, the holder makes $S_T - K = \$120 - \$110 = \$10$

However, for the Put option, at expiry T the holder only exercises the option if the price of the stock, S_T is less than K. In that case, their Payoff is equal to the difference between K and S_T. If the price of the stock is greater than K, then the holder does not exercise the stock, thus making \$0. Again, for example, for a strike price $K = \$110$, if the stock price at time T is

- \$100, the holder makes $\$110 - \$100 = \$10$
- \$120, the holder makes \$0 since $K \leq \$120$

Note that it is possible to sell, or short, an option. In that case, the payoff diagram is just the same as for holding the option, but reflected on the S_t axis. Mathematically, the payoffs become
Call Options: $P_{Call}(t) = min\{0, S_t - K\}$
Put Options: $P_{Put}(t) = min\{0, K - S_t\}$

2.2 The Black-Scholes Model

The Black-Scholes model assumes the distribution of the stock as lognormal. In particular, it writes

$$\ln \frac{S(T)}{S(0)} \sim \mathcal{N}((r - \frac{1}{2}\sigma^2)T, \sigma^2 T)$$

with respect to the risk neutral measure. In the above, the parameters stand for

- $S(0)$: The initial stock price
- $S(T)$: The stock price at T
- r: interest rate
- σ: volatility

The call and put price with maturity T and strike price K will be known as C_0 and P_0 given as below:

$$C_0 = S_0\Phi(d_1) - Ke^{-rT}\Phi(d_2),$$

and

$$P_0 = Ke^{-rT}\Phi(-d_2) - S_0\Phi(-d_1),$$

where the d_i are given as

$$d_1 = \frac{1}{\sigma\sqrt{(T-t)}}\left[\ln\frac{S_0}{K} + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right],$$

and

$$d_2 = \frac{1}{\sigma\sqrt{(T-t)}}\left[\ln\frac{S_0}{K} + \left(r - \frac{\sigma^2}{2}\right)(T-t)\right] = d_1 - \sigma\sqrt{(T-t)}$$

and Φ is the CDF of the Standard Normal Distribution:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{t^2}{2}}dt$$