Neural Network Notes

Xiao (Sean) Zhan

May 2020

1 Introduction

This document includes basic concepts of a neural network, gradient descent, and back-propagation by examining how an algorithm recognizes hand written digits between 0 and 9.

2 Definitions

1. **Neuron** - Something that holds a number n. In this example, $n \in [0, 1]$
2. **Activation** - The number n that a neuron holds
3. **Layer** - A collection of neurons operating together. Layers that are deeper in the network handle increasingly specific tasks.

3 Problem Setup

We want to design an algorithm that recognizes hand written digits. Say our input is a 28 by 28 picture. That's $28 \times 28 = 784$ pixels. Think of each pixel as a neuron. Then the number n that the neuron holds represents the brightness of that pixel, which is also that **activation** of that neuron.

Input layer: these 784 neurons representing the photo of a handwritten digit.
Output layer: 10 neurons labels 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 representing the possible outcome of the algorithm.

4 Constructing the Neural Network

There are two **hidden layers** between the input layer and the output layer. Note that the activation in one layer determines the activation in the next layer.

If we had a trained network, then activation in the input layer will be passed onto the next layer, then the next layer, then the output layer. In the output layer, the

neuron with the highest activation will be the output. For example, if the neuron that corresponds to the digit 3 has the highest activation value, then the algorithm will "guess" that the handwritten digit is 3.

In our specific example of training a computer to recognize handwritten digits, the **first hidden layer** will recognize large components of a digit. For example, 9 will be broken into a line and a circle. The **second hidden layer** will recognize smaller components within the large components that are already recognized by the first hidden layer.

Connect all the neurons in the input layer p_j to the neurons in the first hidden layer q_k, where p_j and q_k denote the activation of the neurons. Assign a weight $w_{k,j}$ to each edge. This weight will determine how much an input activation value matters comparing to other input activation values. Thus, the activation of a neuron q_k in the first hidden layer will be

$$q_k = \sigma(b_k + \sum_{n=1}^{784} w_{k,n} \cdot p_n) \tag{1}$$

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2}$$

Note that the function $\sigma(x)$ helps keep the weighted sum between 0 and 1.
b_k is the **bias**. b_k determines the threshold at which the neuron becomes activated. It can be viewed as another adjustment parameter.

Note that we can also write the above equation (1) into matrix form:

$$\begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_k \end{bmatrix} = \sigma(\begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,j} \\ w_{2,1} & w_{2,2} & \dots & w_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \dots & w_{k,j} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_j \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}) \tag{3}$$

Note that this same formula applies for the interaction between the first hidden layer and the second hidden layer, and the second hidden layer and the output layer.

Goal: Teach the algorithm to adjust the parameters $w_{k,j}$ and b_k so that the output is what we expect.

5 Learning

First, all the parameters $w_{k,j}$ and b_k are initialized randomly. These parameters will then produce an output layer that's expectantly inaccurate at all. Thus, we need to find out the difference between the output layer and the answer that we are expecting.

Say that the output layers gives us:

$$R = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{10} \end{bmatrix}$$

while we expect

$$R' = \begin{bmatrix} r'_1 \\ r'_2 \\ \vdots \\ r'_{10} \end{bmatrix}$$

The cost function finds the difference between R and R'. The cost function is defined as follows:

$$C(W, B) = \sum_{n=1}^{10} (r_n - r'_n)^2 = c$$

where W is all the weights, B is all the biases.

We want to minimize this function. Specifically, we want to minimize the average cost over many many training trials. Note that the cost function takes in all the weights and biases as inputs because the algorithm outputs the output layer based on those inputs. The table clearly shows the comparison between our neural network and the cost function:

	Input	Output	Parameters
Neural Network	784 pixels	output layer	w, b
Cost Function	w, b	cost c	trials

So how do we modify our initial weights and biases to minimize cost? We use **Gradient Descent**, which basically means that after each trial, we add $-\nabla C(W, B)$ to the original W, B. Note that we use backpropagation to calculate $\nabla C(W, B)$. We will discuss how backpropagation works later. Let's arrange all the parameters in matrix P. Thus, after each trial, we update:

$$P' = P - \nabla C(P) \tag{4}$$

In this way, after each trial, we become closer and closer to the local minimum of the cost function. Note that the rate at which we approach the minimum is proportional to the negative of the gradient at each trail so that we don't over correct (the rate at which we approach the mimum would be much smaller when we are near the local minimum).

5.1 Backpropagation

We have the following three matrices:

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_i \end{bmatrix} \quad R = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{10} \end{bmatrix} \quad R' = \begin{bmatrix} r'_1 \\ r'_2 \\ \vdots \\ r'_{10} \end{bmatrix}$$

Where V represents the second hidden layer, R represents the output layer, and R' is the expected value of the output layer.

Assume that $r'_1 = 1$ while $r'_2 = r'_3 = ... = r'_10 = 0$. Then, we need to adjust the parameters such that R matches R'. There are three ways in which we can adjust:

1. Change bias b
2. Change w_{kj}
3. Change V

Note that we cannot directly change V. However, we would need to keep track of how V would be changed so that in our recursive steps of backpropagation, we can adjust the weights and biases accordingly in our first hidden layer. Thus, for each v (refer back to equation (1) to see how different factors change r'), we need to keep track of the **change in bias** b, **change in** w_{kj}, and **change in** v. We add these changes for each v. As a result, we have a matrix

$$V' = \begin{bmatrix} v'_1 \\ v'_2 \\ \vdots \\ v'_i \end{bmatrix}$$

where each entry is the desired change in its corresponding v. We then propagate backwards to see what changes need to be made to the first hidden layer. This is a **recursive** process. We need to calculate backpropagation for each training exmaple. At the end of each recursive function, we store the values of the changes in weights and biases as a matrix P.

We perform a batch of size 10 of trials, then we take the average of all the 10 P's, call it Q_1. This matrix would be a multiple of $\nabla C(P)$, so we use equation (4) to update all the parameters and run another batch of size 10 of trials. We keep updating the parameters in this fashion until the cost function approaches a local minimum.

6 Reference

3Blue1Brown Neural Network series.