Neural Networks From Scratch: The Core Mathematics of a Neural Network

Shlomo Stept

December 30, 2022

Disclaimer : This is an extrapolation of Samson Zhang's video.

1 Overview

The following tutorial explores the mathematics of a Neural Network at a low level, to examine how a Neural Network works, how it makes predictions, and how it learns and improves itself

This is a implementation of a Basic 3 Neural Network that is trained on the MNIST dataset to accurately categorize Handwritten Digit's.

1.1 A Quick Look at the MNIST dataset:

The Dataset we will be working with is the NMIST Handwriten Digits dataset. The full dataset contains 60,000 training images and 10,000 testing images, in the accompanying code only 42,000 images are used. They can be download here.

1. Each Image in the dataset is $28\,rows \times 28\,columns = 784\,pixels$:

$$\underset{28\,Rows \times 28\,Columns}{\begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,28} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,28} \\ \vdots & \vdots & \ddots & \vdots \\ p_{28,1} & p_{28,2} & \cdots & p_{28,28} \end{bmatrix}} \underset{Each\ row\ of\ 28\ values\ (i.e.\ columns)\ is\ concatonated}{\rightarrow \begin{bmatrix} p_{1,1} & \cdots & p_{1,28} \end{bmatrix} + \cdots + \begin{bmatrix} p_{28,1} & \cdots & p_{28,28} \end{bmatrix}} \quad = \quad \underset{to\ get\ a\ 784\ value\ Vector}{\begin{bmatrix} p_1 & p_2 & \cdots & p_{784} \end{bmatrix}}$$

- Note : In most sources we denote data with rows and columns using R^m, where (1) m - denotes the rows, and (2) n denotes the columns

2. Now we can take a look at the Dataset in its entirety - knowing that we have 42,000 images and that each image is represented as a 784 Value Vector, we get the full dataset - $\overline{X}$.

$$\overline{X} = \underset{42,000\ Rows\ \times\ 784\ Columns}{\begin{bmatrix} \text{——} & Image^{(1)} & \text{——} \\ \text{——} & Image^{(2)} & \text{——} \\ & \vdots & \\ \text{——} & Image^{(42,000)} & \text{——} \end{bmatrix}}$$

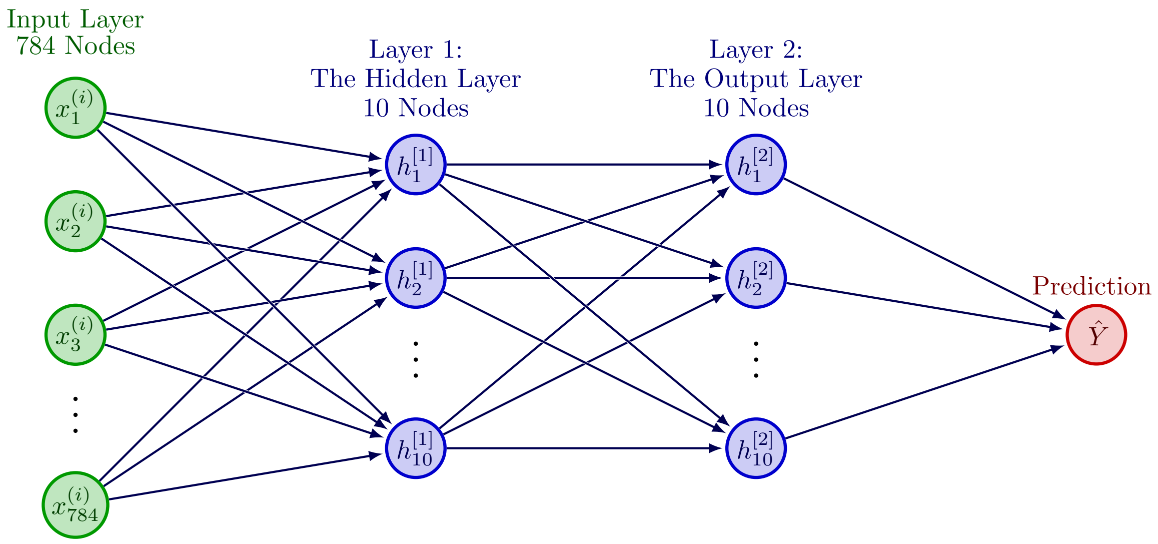
- But to input one image at a time to the column of 784 input nodes of the Neural Network, this matrix must be transposed in order that each column is an image - rather than each row being one image.

$$\overline{X}^T = \underbrace{\begin{bmatrix} | & | & & | \\ Image^{(1)} & Image^{(2)} & \cdots & Image^{(42,000)} \\ | & | & & | \end{bmatrix}}_{784\ Rows\ \times\ 42,000\ Columns}$$

3. The dataset contains handwritten digits, so it follows that there are 10 possible options for each image (digit), Hence there there are 10 classes in the set of possible categories :

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

1.2 The Neural network :



1. The Input layer takes in a single image of 784 pixels: $\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{784} \end{bmatrix}$
2. First we Process the Input Layer to obtain Layer 1 as follows:
   (a) The Image is multiplied by $W^{[1]}$ and, a bias term is added.
   (b) then *ReLU* is applied to the results.
   (c) Putting this all together, Input Layer → Layer 1 : $ReLU(\overline{X}^T W^{[1]} + b^{[1]}) = A^{[1]}$
3. Next we Process the Layer 1 to obtain Layer 2 - the output layer as follows:
   (a) Layer 1 is multiplied by $W^{[2]}$ and, a bias term is added.
   (b) Then the $softmax$ function is applied to the results - to obtain a likelihood score for each of the categories, and
   (c) Putting this all together, Layer 1 → Layer 2 : $SoftMax(A^{[1]}W^{[2]} + b^{[2]})$
4. Finally, the prediction is taken from the Output layer as the Category corresponding to the row with the highest $softmax$ value.

- Notes:

1. The arrows connecting the layers, represents the mathematical multiplications additions and application of functions - that occurs to obtain a layer from the one preceding it. The layers are simply the end result of these calculations.

2 Learning the Correct Weights and Biases

The primary goal of Machine Learning in general is to use data to learn an accurte mapping between an input and the correct output.
- the output can be a label, a value a category, and many more.

In this example, the task of the Nerual Network is to properly classify images of handwritten digits. The Nerual Network if trained properly, will modify its Weights and Biases so that when an image not seen before is input into the neural network, it will correctly identify which digit (0,1...9) has been written.

In order for the network to Learn the proper weights, (1) First it must use its network to process the image , and compute a prediction of which digit it represents. However the initial assignments that are made for the Neural Networks Weights and Biases, will not be the perfect ones, so (2) Second - the network must use some mechanism, that will enable it to learn from its mistakes and update the weights and biases to improve its performance.

This- learning from its mistakes and Updating - is done in three stages :

1. Forward Propagation - which processes the input and outputs a prediction.
2. Backwards Propagation - which (1) uses the Network's current prediction, and the ground truth labeled data, to calculate the error, and also calculates (2) how much each of the weights and biases - from each layer - contributed to the final prediction.
3. Then these calculations (error and % contribution) are then used to update the Weights and Biases.
4. Then Repeat.

– Now we will explore each stage in more detail.

2.1 Forward Propagation:

Lets walk through each step of a forward propagation one at a time:

2.1.1 Input Layer $\rightarrow$ Layer 1

1. $A^{[0]} = \overline{X}$: Our initial starting point is really $\overline{X}^T$, which has 784 Rows $\times$ 42,000 Columns

2. $Z^{[1]} = W^{[1]} \cdot A^{[0]} + b^{[1]}$

   - Where :

$$\underset{\substack{W^{[0]} \\ 10 \times 784}}{\begin{bmatrix} W^{[1]}_{1,1} & \dots & W^{[1]}_{1,784} \\ W^{[1]}_{2,1} & \dots & W^{[1]}_{2,784} \\ \vdots & \ddots & \vdots \\ W^{[1]}_{10,1} & \dots & W^{[1]}_{10,784} \end{bmatrix}} \cdot \underset{\substack{A^{[0]} \\ 784 \times \text{m}}}{\begin{bmatrix} A^{[0]}_{1,1} & A^{[0]}_{1,2} & \dots & A^{[0]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ A^{[0]}_{784,1} & A^{[0]}_{784,2} & \dots & A^{[0]}_{784,m} \end{bmatrix}} + \underset{\substack{b^{[1]} \\ 10 \times 1}}{\begin{bmatrix} b^{[1]}_1 \\ b^{[1]}_2 \\ \vdots \\ b^{[1]}_{10} \end{bmatrix}} = \underset{\substack{Z^{[1]} \\ 10 \times \text{m}}}{\begin{bmatrix} Z^{[1]}_{1,1} & Z^{[1]}_{1,2} & \dots & Z^{[1]}_{1,m} \\ Z^{[1]}_{2,1} & Z^{[1]}_{2,2} & \dots & Z^{[1]}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ Z^{[1]}_{10,1} & Z^{[1]}_{10,2} & \dots & Z^{[1]}_{10,m} \end{bmatrix}}$$

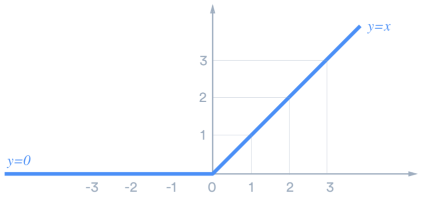
Note:

(a) $W^{[1]}$ is the matrix between the Input-Layer and Layer 1- visually this is represented with the connections between the nodes of the Input and First layers.

(b) $Z^{[1]}$ is the output of (a) Multiplying $W^{[1]}$ (the first weights) and $A^{[0]}$ (the input) - which results in a $10 \times m$ matrix, - and then adding a bias vector $b^{[1]}$ of size 10×1 to each of the resulting columns.

3. $A^{[1]} = g(Z^{[1]}) = ReLU(Z^{[1]})$

$$\text{Where } ReLU(x) = \begin{cases} x\,,\ for\ x > 0 \\ 0\,,\ otherwise \end{cases}$$



- This $A^{[1]}$ is the input to Hidden Layer 1 - visually its represented as the Hidden Layer 1 nodes.

Notes:

(a) At this point you might be wondering why are we applying the ReLU function to our results, before we get to Layer 1 ($A^{[1]}$) ?

(b) The Use of the Function ReLU (Rectified Linear Unit), a non-linear function, is the core of what makes Neural Networks Special, and different from older Machine Learning Tactics.

(c) The Special property of ReLU, the core of why we use this, and what differentiates a neural network from a standard Machine learning Algorithm such as Linear or Logistic Regression, is the Non-Linearity of the ReLU function.

(d) Without the use of ReLU (or another non-linear function) our neural network would just be a standard linear function. There is nothing inherently wrong with linear functions, however many linear machine learning algorithms already exists, and these algorithms tend to level out before reaching perfection (typically they have a maximum accuracy of 70%-90%). The intuition behind Neural Networks, is that some data is non-linear, therefore employing algorithms that utilize linear relationships will always fall short of perfection. However some pattern does exist in the data - no matter how non-linear it is - if the non-linearity is taken into account. This is where ReLu and Neural Networks come in, by utilizing a model which has the non-linearity built into it, Neural Networks are able to improve performance in almost all applications.

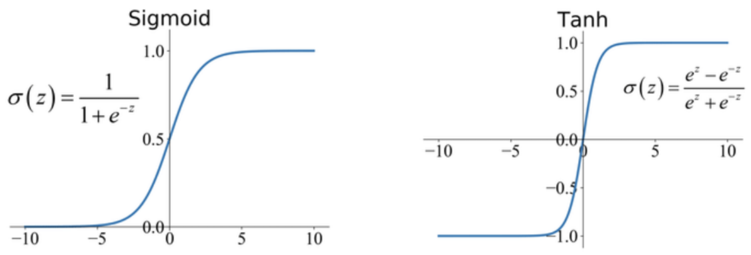


Figure 1: Other non-linear functions include Sigmoid and Tanh

2.1.2 Layer 1 $\rightarrow$ Layer 2

1. $Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]}$

   - Where :

$$\underset{\substack{W^{[2]} \\ 10 \times 10}}{\begin{bmatrix} W^{[2]}_{1,1} & \dots & W^{[2]}_{1,10} \\ \vdots & \ddots & \vdots \\ W^{[2]}_{10,1} & \dots & W^{[2]}_{10,10} \end{bmatrix}} \cdot \underset{\substack{A^{[1]} \\ 10 \times \text{m}}}{\begin{bmatrix} A^{[1]}_{1,1} & A^{[1]}_{1,2} & \dots & A^{[1]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ A^{[1]}_{10,1} & A^{[1]}_{10,2} & \dots & A^{[1]}_{10,m} \end{bmatrix}} + \underset{\substack{b^{[1]} \\ 10 \times 1}}{\begin{bmatrix} b^{[2]}_1 \\ b^{[2]}_2 \\ \vdots \\ b^{[2]}_{10} \end{bmatrix}} = \underset{\substack{Z^{[2]} \\ 10 \times \text{m}}}{\begin{bmatrix} Z^{[2]}_{1,1} & Z^{[2]}_{1,2} & \dots & Z^{[2]}_{1,m} \\ Z^{[2]}_{2,1} & Z^{[2]}_{2,2} & \dots & Z^{[2]}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ Z^{[2]}_{10,1} & Z^{[2]}_{10,2} & \dots & Z^{[2]}_{10,m} \end{bmatrix}}$$

   Notes:

   (a) $W^{[2]}$ is the matrix we multiply between the Layer 1 and Layer 2 - visually this is represented with all the connections between all nodes in the input and first layer. and we multiple t

   (b) $Z^{[1]}$ is the output of applying (a) a matrix multiplication between $W^{[2]}$ and $A^{[1]}$ to the input which results in a $10 \times m$ matrix, and then adding a bias vector $b^{[2]}$ of size 10×1 to each of the resulting columns.

2. $A^{[2]} = softmax(Z^{[2]})$

   Where :

   (a) $softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$

   (b) so for $X = \begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \end{bmatrix}$ $softmax(\begin{bmatrix} 1.3 \\ 5.1 \\ 2.2 \end{bmatrix}) = \begin{bmatrix} \frac{e^{1.3}}{e^{1.3}+e^{5.1}+e^{2.2}} \\ \frac{e^{5.1}}{e^{1.3}+e^{5.1}+e^{2.2}} \\ \frac{e^{2.2}}{e^{1.3}+e^{5.1}+e^{2.2}} \end{bmatrix} = \begin{bmatrix} 0.020 \\ 0.903 \\ 0.0497 \end{bmatrix}$

   Notes:

   i. Why exactly do we use the softmax function on the last layer of 10 nodes?

   ii. Background : When an image in input into the neural network, the goal is to obtain a correct classification of number the handwritten digit represents. The Neural Network outputs a 10 row vector where each row represents a number from 0-9. Logically each row of the 10 row vector, represents the likelihood of that class being the one the handwritten digit represents. Without modifying the vector output by the Neural Network, the network would essentially be using the argmax activation function - where the maximum value is used as our predicted digit. Softmax is a softer, more probabilistic function, that is much more useful when learning the proper weights and biases, and here's why.

   iii. Given a 3 value vector $\begin{bmatrix} 1.3 & 5.1 & 2.2 \end{bmatrix}^T$, using the standard argmax function, all values except for the largest would be 0, resulting in the vector $\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$, however the first step of the networks learning procedure calculates the error of the prediction versus the real ground truth value. This is done by converting each ground truth value into its one hot encoding vector.
   – A one hot encoding of a value is a vector which contains $0's$ in all locations except for the row which represents its value - for example the one hot encoding of the number 1 is : $\begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^T$

   iv. So when computing the loss/error using two mostly zeroed out vectors, at most two of the rows will contain loss values that can be used when updating, this looses all the information encoded in the remaining values output by the Network.

v. The softmax function is a probabilistic function, which gives the highest weight to the largest value, while still computing a probability for all the other values. This output vector can then be used in the loss function to update the weights associated with the incorrect predictions.

vi. Without the use of softmax, all the information encoded in the final prediction vector would be lost and could not be used during the learning stage -back propagation- to improve our network. This is why softmax is so useful in classification problems and why it is used in this Neural Network.

2.2 Backward Propagation:

The purpose of backwards propagation:

The initial choice of values for the weights and biases - randomly chosen values between -0.5 and 0.5 - will not be an accurate representation of the optimal values for the wights a biases, that can be used to correctly classify an input image. Thus the goal of backwards propagation is to improve the incorrect weights and biases.

At the end of a Forward propagation the Network outputs a prediction using its current weights and biases. With the knowledge of the ground truth (i.e. the correct category for an image) the Network can update its weights and biases, in order to improve its classification accuracy.

1. In general:

   (a) First the prediction is computed and compared with the ground truth, to determine the magnitude of the error.

   (b) Then the Network is Traversed backwards starting from the final layer, and for each layer a determination is made regarding, the degree to which its weights and biases contributed to the incorrect/correct prediction.

   This % contribution and the magnitude of the error is then used by the network to adjust its associated weights and biases, to improve its performance.

2.2.1 Layer 2 → Layer 1

1. $dZ^{[2]} = A^{[2]} - \hat{Y}$

   - Where :

$$\underset{\substack{A^{[2]} \\ 10 \times \text{m}}}{\begin{bmatrix} A^{[2]}_{1,1} & A^{[2]}_{1,2} & \dots & A^{[2]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ A^{[2]}_{10,1} & A^{[2]}_{10,2} & \dots & A^{[2]}_{10,m} \end{bmatrix}} - \underset{\substack{\hat{Y} \\ 10 \times \text{m}}}{\begin{bmatrix} \hat{Y}_{1,1} & \hat{Y}_{1,2} & \dots & \hat{Y}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{Y}_{10,1} & \hat{Y}_{10,2} & \dots & \hat{Y}_{10,m} \end{bmatrix}} = \underset{\substack{dZ^{[2]} \\ 10 \times \text{m}}}{\begin{bmatrix} dZ^{[2]}_{1,1} & dZ^{[2]}_{1,2} & \dots & dZ^{[2]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[2]}_{10,1} & dZ^{[2]}_{10,2} & \dots & dZ^{[2]}_{10,m} \end{bmatrix}}$$

   (a) Since softmax is used to transform the output Vector into class probabilities for each of the 10 categories. Each of the m columns of the matrix $A^{[2]}$ is a column vector of probabilities :

$$A^{[2]}_{column\ 1} = \begin{bmatrix} 0.05 \\ 0.928 \\ 0.0002 \\ 0.019 \\ \vdots \end{bmatrix}$$

   (b) And each images ground truth class is transformed into a one-hot-encoding column vector corresponding to its class :

$$\hat{Y}_{column\ 1} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

(c) This enables the network to update the weights and biases for all the images - for all the classes.

2. $dW^{[2]} = \frac{1}{m} \times dZ^{[2]} \cdot A^{[1]^T}$

- Where :

$$\underset{\substack{dZ^{[2]} \\ 10 \times m}}{\begin{bmatrix} dZ^{[2]}_{1,1} & dZ^{[2]}_{1,2} & \dots & dZ^{[2]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[2]}_{10,1} & dZ^{[2]}_{10,2} & \dots & dZ^{[2]}_{10,m} \end{bmatrix}} \cdot \underset{\substack{A^{[1]^T} \\ m \times 10}}{\begin{bmatrix} A^{[1]^T}_{1,1} & \dots & A^{[1]^T}_{1,10} \\ A^{[1]^T}_{2,1} & \dots & A^{[1]^T}_{2,10} \\ \vdots & \ddots & \vdots \\ A^{[1]^T}_{m,1} & \dots & A^{[1]^T}_{m,10} \end{bmatrix}} = \underset{\substack{dW^{[2]} \\ 10 \times 10}}{\begin{bmatrix} dW^{[2]}_{1,1} & \dots & dW^{[2]}_{1,10} \\ \vdots & \ddots & \vdots \\ dW^{[2]}_{10,1} & \dots & dW^{[2]}_{10,10} \end{bmatrix}}$$

Notes:

(a) Step 1 calculates $dZ^{[2]}$ the magnitude of the difference between the networks prediction, and the ground truth.

(b) Now the network attempts to determine the extent to which $W^{[2]}$ and $b^{[2]}$ contributed to the correct or incorrect prediction, regardless, we can use the information of final output, along with a determination of the extent to which $W^{[2]}$ and $b^{[2]}$ contributed to these predictions to update $W^{[2]}$ and $b^{[2]}$, to improve their performance.

(c) Visually : $\begin{bmatrix} dZ^{[2]}_{1,1} & \cdots & dZ^{[2]}_{1,m} \end{bmatrix} \cdot \begin{bmatrix} A^{[1]^T}_{1,1} \\ \vdots \\ A^{[1]^T}_{m,1} \end{bmatrix} = dZ^{[2]}_{1,1} A^{[1]^T}_{1,1} + ... + dZ^{[2]}_{1,m} A^{[1]^T}_{m,1}$

Where:

i. The row $(dZ^{[2]}_i)$ represents the error for all the images corresponding to the category row i represents.

ii. The col $A^{[1]}_{:,j}$, has no clear mapping but should correspond to the values for a single class for each of the m images.

iii. This multiplication will result in the sum of the error for the weights corresponding to class i given by $W^{[2]}$

(warning - this isn't 100% accurate but aids in understanding whats taking place within the Neural Network)

3. $db^{[2]} = \frac{1}{m}\sum_{n=0}^{m} dZ^{[2]}$

   - Where :

$$\underset{\substack{dZ^{[2]} \\ 10\ \times\ \text{m}}}{\begin{bmatrix} dZ^{[2]}_{1,1} & dZ^{[2]}_{1,2} & \dots & dZ^{[2]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[2]}_{10,1} & dZ^{[2]}_{10,2} & \dots & dZ^{[2]}_{10,m} \end{bmatrix}} \quad \textit{and} \quad \underset{\substack{db^{[2]} \\ \text{m}\ \times\ 10}}{\begin{bmatrix} db^{[2]}_{1,1} \\ db^{[2]}_{2,1} \\ \vdots \\ db^{[2]}_{10,1} \end{bmatrix}} \quad \textit{and} \ db^{[2]}_{1,1} = \sum_{j=1}^{m} \begin{bmatrix} dZ^{[2]}_{1,j} & \dots & dZ^{[2]}_{1,j} \end{bmatrix}$$

   Notes:

   (a) Each row of the $dZ^{[2]}$ matrix, is summed up, thus each row has a single value. The aim for the bias term to incorporate the average error/missing-value the matrix $W^{[2]}$ has for each row of nodes.

2.2.2 Layer 1 → Input Layer

1. $dZ^{[1]} = (W^{[2]^{[T]}} \cdot dZ^{[2]}) \cdot g^{'}(Z^{[1]})$

   - Where :

$$\underset{\substack{dZ^{[2]} \\ 10\ \times\ \text{m}}}{\begin{bmatrix} dZ^{[1])}_{1,1} & dZ^{[1]}_{1,2} & \dots & dZ^{[1]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[1]}_{10,1} & dZ^{[1]}_{10,2} & \dots & dZ^{[1]}_{10,m} \end{bmatrix}} = \left(\underset{\substack{W^{[2]^{[T]}} \\ 10\ \times\ 10}}{\begin{bmatrix} W^{[2]^{[T]}}_{1,1} & \dots & W^{[2]^{[T]}}_{1,10} \\ \vdots & \ddots & \vdots \\ W^{[2]^{[T]}}_{10,1} & \dots & W^{[2]^{[T]}}_{10,10} \end{bmatrix}} \cdot \underset{\substack{dZ^{[2]}) \\ 10\ \times\ \text{m}}}{\begin{bmatrix} dZ^{[2]}_{1,1} & dZ^{[2]}_{1,2} & \dots & dZ^{[2]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[2]}_{10,1} & dZ^{[2]}_{10,2} & \dots & dZ^{[2]}_{10,m} \end{bmatrix}} \right) \underset{10\ \times\ \text{m}}{\cdot\ g^{'}(Z^{[1])})}$$

   Notes:

   (a) The value of error $dZ^{[1]}$ aim to represent the magnitude of the error associated with layer 1.

   (b) Visually : $\begin{bmatrix} W^{[2]^{[T]}}_{1,1} & \dots & W^{[2]^{[T]}}_{1,10} \end{bmatrix} \cdot \begin{bmatrix} dZ^{[2]}_{1,1} \\ \vdots \\ dZ^{[2]}_{10,1} \end{bmatrix} = \mathrm{W}^{[2]^{[T]}}_{1,1} dZ^{[2]}_{1,1} + ... + W^{[2]^{[T]}}_{1,10} dZ^{[2]}_{10,1}$

   Where:

   i. The row vector $W^{[2]^{T}}$ represents a single classes weight vector in $W^{[2]}$
   ii. The column vector $dZ^{[2]}$ represents the magnitude of the error in the Networks Prediction for that class.
   iii. and $g^{'}(Z^{[1]})$ is used to determine which nodes the ReLU function zeroed out, since the network cannot attribute any error values to the nodes that contributed nothing to the original prediction.
   iv. In summary, other words $dZ^{[1]}$ is determining the magnitude of the error contributed by layer-1.

   (warning - this isn't 100% accurate but aids in understanding whats taking place within the Neural Network)

2. $dW^{[1]} = \frac{1}{m} \times dZ^{[1]} \cdot A^{[0]^T}$

   - Where :

$$\underset{\substack{dZ^{[1]} \\ 10 \times \text{m}}}{\begin{bmatrix} dZ^{[1]}_{1,1} & dZ^{[1]}_{1,2} & \dots & dZ^{[1]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[1]}_{10,1} & dZ^{[1]}_{10,2} & \dots & dZ^{[1]}_{10,m} \end{bmatrix}} \cdot \underset{\substack{A^{[0]^T} \\ \text{m} \times 784}}{\begin{bmatrix} A^{[0]^T}_{1,1} & A^{[0]^T}_{1,2} & \dots & A^{[0]^T}_{1,784} \\ A^{[0]^T}_{2,1} & A^{[0]^T}_{2,2} & \dots & A^{[0]^T}_{2,784} \\ \vdots & \vdots & \ddots & \vdots \\ A^{[0]^T}_{m,1} & A^{[0]^T}_{m,2} & \dots & A^{[0]^T}_{m,784} \end{bmatrix}} = \underset{\substack{dW^{[1]} \\ 10 \times 784}}{\begin{bmatrix} dW^{[1]}_{1,1} & dW^{[1]}_{1,2} & \dots & dW^{[1]}_{1,784} \\ \vdots & \vdots & \ddots & \vdots \\ dW^{[1]}_{10,1} & dW^{[1]}_{10,2} & \dots & dW^{[1]}_{10,784} \end{bmatrix}}$$

   Notes:

   (a) Step 1 calculates $dZ^{[1]}$ the magnitude of the error associated with layer 1.

   (b) Now the network attempts to determine the extent to which $W^{[1]}$ and $b^{[1]}$ contributed to the correct or incorrect prediction, regardless, we can use the information of the middle layer, along with a determination of the extent to which $W^{[1]}$ and $b^{[1]}$ contributed to these predictions to update $W^{[2]}$ and $b^{[2]}$, to improve their performance.

   (c) Visually : $\begin{bmatrix} dZ^{[1]}_{1,1} & \cdots & dZ^{[1]}_{1,m} \end{bmatrix} \cdot \begin{bmatrix} A^{[0]^T}_{1,1} \\ \vdots \\ A^{[0]^T}_{m,1} \end{bmatrix} = dZ^{[1]}_{1,1} A^{[0]^T}_{1,1} + ... + dZ^{[1]}_{1,m} A^{[0]^T}_{m,1}$

   Where:

   i. The row $(dZ^{[1]}_i)$ represents the magnitude of the error for the row i in $W^{[1]}$

   ii. The col $A^{[0]}_{:,j}$, corresponds to the values for a single class for each of the m images.

   iii. This multiplication will result in the sum of the error for the weights corresponding to class i given by $W^{[1]}$

3. $db^{[1]} = \frac{1}{m} \sum_{n=0}^{m} dZ^{[1]}$

   - Where :

$$\underset{\substack{dZ^{[2]} \\ 10 \times \text{m}}}{\begin{bmatrix} dZ^{[1]}_{1,1} & dZ^{[1]}_{1,2} & \dots & dZ^{[1]}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ dZ^{[1]}_{10,1} & dZ^{[1]}_{10,2} & \dots & dZ^{[1]}_{10,m} \end{bmatrix}} \quad \textit{to get} \quad \underset{\substack{db^{[2]} \\ \text{m} \times 10}}{\begin{bmatrix} db^{[1]}_{1,1} \\ db^{[1]}_{2,1} \\ \vdots \\ db^{[1]}_{10,1} \end{bmatrix}} \quad : \quad db^{[1]}_{1,1} \; = \; \sum_{j=1}^{m} \begin{bmatrix} dZ^{[1]}_{1,j} & \dots & dZ^{[1]}_{1,j} \end{bmatrix}$$

   (a) Here each row of the $dZ^{[1]}$ matrix, is summed up, thus each row has a single value. The aim for the bias term to incorporate the average error/missing-value the matrix $W^{[1]}$ has for each row of nodes.

2.2.3 Putting it all together:

Finally the values calculated during the backwards propagation are used to update the weights a biases of the Neural Network.

1. $W^{[1]} = W^{[1]} - \alpha \cdot dW^{[1]}$
2. $b^{[1]} = b^{[1]} - \alpha \cdot db^{[1]}$
3. $W^{[2]} = W^{[2]} - \alpha \cdot dW^{[2]}$
4. $b^{[2]} = b^{[2]} - \alpha \cdot db^{[2]}$

    (a) Note -α is a hyperparameter - a value which is set manually and not learned via training the network. typically values for α tend to be between $0.1 - 0.0001$. As a rule of thumb for ever decimal place reduction in the learning rate, the network requires 10 times the the number of updates (epochs) to achieve the same accuracy.

    For more details on learning rate - see wikipedia

2.3 Final Notes

By repeatedly running this process of (1) forward propagation (2) propagation (3) updating weights and biases, the Neural Network should be able to achieve an accuracy of around 90%

For a step by step code implementation see : Neural Networks from Scratch - Using Only Numpy