The Fascinating Maths Behind Neural Networks

Adrian Fang

1 Introduction

The concept of Neural Networks stem from solving seemingly simplistic problems that do not seem feasible from a programming perspective. Consider the case of recognising handwritten digits. As simple a task it may seem, there are too many cases to consider when attempting to write an algorithm. Using a Neural Network on the other hand can condense the otherwise thousands of lines and rules dedicated to classification into a concrete algorithm that will learn how to solve the problem. Using a combination of statistics, probability, calculus and linear algebra, the gateway to creating a neural network is opened.

2 The Architecture of Simple Neural Networks

2.1 As a Whole

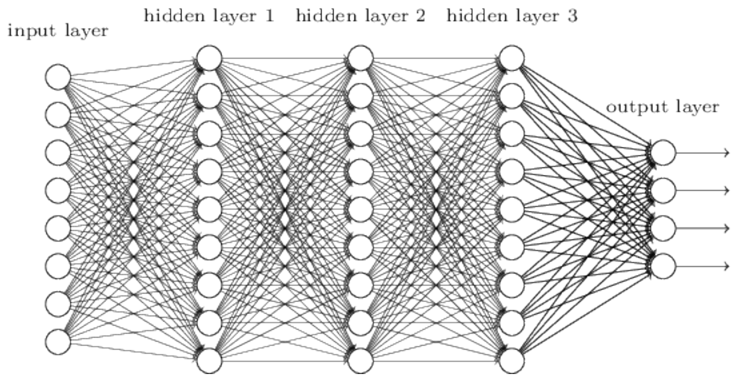
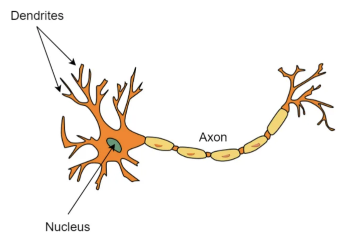


Figure 2.1 - General Diagram of Multi-Layer Neural Network

Every Neural Network consists of input layers, calculation or hidden layers, followed by a final output layer with a solution to the objective. Every neuron is connected with every neuron in the next layer, leading up to the output. The network leverages weights and biases to make predictions, which are modified in the training process.

2.2 Individual Neurons and Making Predictions

Individual neurons are modelled on how scientists believe neurons in a brain work. Neurons are interconnected with every other neuron, sending information back and forth to progressively learn and make predictions based on the task at hand. Consider the following two figures, representing a biological and artificial neuron respectively:



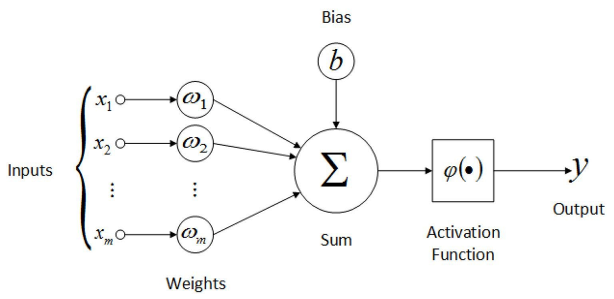


Figure 2.2 & Figure 2.3 - Diagrams of Biological and Artificial Neurons

Every neuron in a layer is represented in the form of matrices, where layers take the shape of $(m, 1)$, where m is the amount of neurons in the layer. For every neuron in the previous layer to pass into the next, specific weights and biases are used to modify the values which are then passed through an activation function. These weights and biases are also represented using matrices. Weights for any given layer take the shape (n, m), where m is the amount of neurons in the current layer, with n representing the amount of neurons in the next. A generalised formula is used to represent the transformation of a neuron into a proceeding layer:

$$z_i = \begin{cases} w_i x + b_i & i = 0 \\ w_i a_{i-1} + b_i & i \neq 0 \end{cases} \tag{1}$$

Since w_i, a_{i-1} and x are matrices, dot product multiplication is required to multiply them. A piecewise function is used to differentiate between the input matrix x and previously calculated neurons a_{i-1} which would be used after conducting operations on the original values. Afterwards, the bias value is added to every value in the newly created matrix. These individual bias values should match the shape of the new matrix, otherwise addition is impossible. An activation function is then used to mitigate the linear relationship that is caused by the multiplication of weights and the original input values. Suppose that the activation function be represented as Q:

$$a_i = Q(z_i) \tag{2}$$

This process repeats throughout the entire neural network until reaching the output. This procedure is known as forward propagation, which is how the model makes predictions. The output layer of the model depends on the task, which may be returning probabilities of classifications of multiple types of objects.

2.3 Learning With Gradient Descent

Forward propagation is how the model makes predictions, however does not enable it to make accurate predictions without training. This is the role of backward propagation, which is forward propagation but in reverse. Rather than moving from the input layer to the output, backward propagation begins from the output layer and travels to the input. Gradient descent depends on a loss function that is to be minimised. The loss function takes two parameters: the true labels and predicted labels of the model's output layer. In the scope of representing handwritten digits, the Categorical Cross-Entropy Loss function is used, shown below:

$$L = -\sum_{i=1}^{n} \left(y_i log\left(\hat{y_i}\right)\right) \tag{3}$$

Note that log in this case has base e, which could otherwise be written as the natural logarithm ln. Let n represent the amount of output neurons, y_i represent the truth label and $\hat{y_i}$ represent the predicted labels. Loss is used to calculate error in the model, and is not to be mistaken as a direct inverse to accuracy. The objective of a neural network is to minimise the loss of the model, and hence maximise accuracy. This is where Gradient Descent is used, which is an optimisation algorithm to obtain the global minimum of a function by in this case adjusting weights and biases to find the global minimum of loss. Generally, to find the derivative of loss with respect to weights and biases in certain layers, there are two different scenarios: backpropagation of the final layer and for every other. Once these derivatives have been calculated, they are subtracted from weights and biases to improve the model's accuracy.

3 Cross-Entropy Softmax Loss Backpropagation

3.1 The Derivative of Loss in the Final Layer

Since Neural Networks utilise multivariable functions, partial derivatives are used to assess the change in loss. Backward propagation aims to find $\frac{\partial L}{\partial w_{i,j}}$ and $\frac{\partial L}{\partial b_{i,j}}$, but will require $\frac{\partial L}{\partial z_j}$. Consider the chain rule of differentiation, which will be used to find both derivatives:

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{i,j}} \tag{4}$$

$$\frac{\partial L}{\partial b_{i,j}} = \frac{\partial L}{\partial z_j} \cdot \frac{\partial z_j}{\partial b_{i,j}} \tag{5}$$

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial \hat{y_i}} \cdot \frac{\partial \hat{y_i}}{\partial z_j} \tag{6}$$

Let's consider the cross entropy loss function as L, which the softmax function as the output layer's activation function:

$$\hat{y_i} = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \tag{7}$$

Notice how there are two variables to represent indices (i,j). This stems from neurons being connected with every other neuron in the previous layer. To represent this change, there are two separate indices to represent the ith index or neuron in the output layer, with j being used as an index for the previous. Let the following Jacobian Matrix demonstrate this distinction, though it does not relate to the values of the neural network:

$$J = \begin{bmatrix} \nabla f_1 \\ \nabla f_2 \\ \nabla f_3 \\ \vdots \\ \nabla f_n \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_m} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \tag{8}$$

The weights and bias matrices are similar to Jacobian matrices, which contain different values and indices based on the layer and neuron. The two separate variables will also result in two cases in the process of differentiation: one where $i = j$, and one where $i \neq j$. First, let's derive $\frac{\partial \hat{y_i}}{\partial z_i}$ to account for the first case:

$$\frac{\partial \hat{y_i}}{\partial z_i} = \frac{\partial}{\partial z_i} \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \tag{9}$$

Using the quotient rule of differentiation:

$$\frac{\partial \hat{y_i}}{\partial z_i} = \frac{\sum_{k=1}^{n} e^{z_k} \cdot \frac{\partial}{\partial z_i} e^{z_i} - e^{z_i} \cdot \frac{\partial}{\partial z_i} \sum_{k=1}^{n} e^{z_k}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \tag{10}$$

The differentiated sum equates to e_{z_i} due to every other variable in the array being independent of z_i, hence their individual derivatives are zero.

$$\frac{\partial \hat{y_i}}{\partial z_i} = \frac{\sum_{k=1}^{n} e^{z_k} \cdot e^{z_i} - e^{z_i} \cdot e^{z_i}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \tag{11}$$

$$\frac{\partial \hat{y}_i}{\partial z_i} = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \left(\frac{\sum_{k=1}^{n} e^{z_k} - e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \right) \tag{12}$$

$$\frac{\partial \hat{y}_i}{\partial z_i} = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \left(1 - \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \right) \tag{13}$$

We can further simplify this equation by substituting $\hat{y}_i$.

$$\frac{\partial \hat{y}_i}{\partial z_i} = \hat{y}_i \left(1 - \hat{y}_i\right) \tag{14}$$

For the next case, consider the differentiation of $\hat{y}_i$ with respect to z_j.

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{\partial}{\partial z_i} \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}} \tag{15}$$

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{\sum_{k=1}^{n} e^{z_k} \cdot \frac{\partial}{\partial z_j} e^{z_i} - e^{z_i} \cdot \frac{\partial}{\partial z_j} \sum_{k=1}^{n} e^{z_k}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \tag{16}$$

Since z_i is independent of z_j, $\frac{\partial}{\partial z_j} e^{z_i} = 0$.

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{\sum_{k=1}^{n} e^{z_k} \cdot 0 - e^{z_i} \cdot e^{z_j}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \tag{17}$$

$$\frac{\partial \hat{y}_i}{\partial z_j} = -\frac{e^{z_i} \cdot e^{z_j}}{\left(\sum_{k=1}^{n} e^{z_k}\right)^2} \tag{18}$$

$$\frac{\partial \hat{y}_i}{\partial z_j} = -\frac{e^{z_i} \cdot e^{z_j}}{\left(\sum_{k=1}^{n} e^{z_k} \cdot \sum_{k=1}^{n} e^{z_k}\right)} \tag{19}$$

$$\frac{\partial \hat{y}_i}{\partial z_j} = -\hat{y}_i \cdot \hat{y}_j \tag{20}$$

Now, to find the derivative of Loss with respect to z_j. This process will be slightly different, and will have account for both cases. The equation for Cross Entropy Loss is shown again:

$$L = -\sum_{i=1}^{n} \left(y_i log\left(\hat{y}_i\right)\right) \tag{21}$$

This sum will have to be split, which will separate the jth index to account for $i = j$, where the remaining elements of the sum will account for $i \neq j$:

$$L = -\left(\sum_{i=1, i\neq j}^{n} \left(y_i log\left(\hat{y}_i\right)\right) + y_j log(\hat{y}_j) \right) \tag{22}$$

$$L = -\sum_{i=1, i\neq j}^{n} \left(y_i log\left(\hat{y}_i\right)\right) - y_j log(\hat{y}_j) \tag{23}$$

Loss can now be differentiated. Since $\hat{y_i}$ and $\hat{y_j}$ represent the softmax function, implicit differentiation is required. The sums in the denominator of the softmax function consist of z_j and z_i, which means that the results of the softmax functions in both cases are dependent on z_j and therefore means that their derivatives are not zero.

$$\frac{\partial L}{\partial z_j} = \frac{\partial}{\partial z_j}\left(-\sum_{i=1, i\neq j}^{n} \left(y_i log\left(\hat{y_i}\right)\right) - y_j log(\hat{y_j})\right) \tag{24}$$

$$\frac{\partial L}{\partial z_j} = -\frac{\partial}{\partial z_j}\sum_{i=1, i\neq j}^{n} \left(y_i log\left(\hat{y_i}\right)\right) - \frac{\partial}{\partial z_j} y_j log(\hat{y_j}) \tag{25}$$

$$\frac{\partial L}{\partial z_j} = -\sum_{i=1, i\neq j}^{n} \left(\frac{y_i}{\hat{y_i}} \cdot \frac{\partial \hat{y_i}}{\partial z_j}\right) - \frac{y_j}{\hat{y_j}} \cdot \frac{\partial \hat{y_j}}{\partial z_j} \tag{26}$$

Using the equations derived earlier for $\frac{\partial \hat{y_i}}{\partial z_j}$ and $\frac{\partial \hat{y_i}}{\partial z_i}$, the partial derivatives can be substituted.

$$\frac{\partial L}{\partial z_j} = -\sum_{i=1, i\neq j}^{n} \left(\frac{y_i}{\hat{y_i}} \cdot -\hat{y_i} \cdot \hat{y_j}\right) - \frac{y_j}{\hat{y_j}} \cdot \hat{y_j}(1-\hat{y_j}) \tag{27}$$

$$\frac{\partial L}{\partial z_j} = \sum_{i=1, i\neq j}^{n} \left(y_i \cdot \hat{y_j}\right) - y_j(1-\hat{y_j}) \tag{28}$$

$$\frac{\partial L}{\partial z_j} = \sum_{i=1, i\neq j}^{n} \left(y_i \cdot \hat{y_j}\right) - y_j + (y_j \cdot \hat{y_j}) \tag{29}$$

The function in the sum matches the rightmost term of the equation for $i = j$, meaning that $(y_j \cdot \hat{y_j})$ can be inserted into the sum to account for all i, where i can be equal to j. Additionally, since $\hat{y_j}$ is independent of i, it can be factored out of the sum.

$$\frac{\partial L}{\partial z_j} = \sum_{i=1}^{n} \left(y_i \cdot \hat{y_j}\right) - y_j \tag{30}$$

$$\frac{\partial L}{\partial z_j} = \hat{y_j}\sum_{i=1}^{n} y_i - y_j \tag{31}$$

Since y_i is a one-hot vector, the sum will cancel out to 1.

$$\frac{\partial L}{\partial z_j} = \hat{y_j} - y_j \tag{32}$$

Marvellously, the solution to such a complex expression is spectacularly simple. By combining this equation with $\frac{\partial z_j}{\partial w_{i,j}}$, $\frac{\partial L}{\partial w_{i,j}}$ can be solved. Since index j will not be zero in the case of working with the output layer, the original case from the piecewise function can be neglected.

$$\frac{\partial z_j}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \left(w_i a_{j-1} + b_i \right) \tag{33}$$

$$\frac{\partial z_j}{\partial w_{i,j}} = a_{j-1} \tag{34}$$

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial z_j} \cdot \left(a_{j-1} \right)^T \tag{35}$$

Calculating the bias derivative is trivial, given how it follows the same procedure as the combination of $\frac{\partial z_j}{\partial w_{i,j}}$ and $\frac{\partial L}{\partial z_j}$, however uses $\frac{\partial z_j}{\partial b_{i,j}}$. Although the function for z is a piecewise, these cases are neglected since partial differentiation with respect to the bias weights will cancel them out.

$$\frac{\partial z_j}{\partial b_{i,j}} = \frac{\partial}{\partial b_{i,j}} \left(w_i x + b_i \right) = 1 \tag{36}$$

$$\frac{\partial L}{\partial b_{i,j}} = \frac{\partial L}{\partial z_j} \tag{37}$$

With equations for every derivative obtained, the first step in adjusting weights and biases can be completed.

3.2 The Derivative of Loss in Input and Hidden Layers

The process to find the derivative of loss in every other layer is not as arduous as the final. The following are the conventional formulae for the Softmax Cross-Entropy derivatives:

$$\frac{\partial L}{\partial z_j} = Q' \left(z_j \right) \cdot \left(w_{j+1} \right)^T \cdot \frac{\partial L}{\partial z_{j+1}} \tag{38}$$

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z_j} \cdot \left(a_{j-1} \right)^T \tag{39}$$

$$\frac{\partial L}{\partial b_j} = \frac{\partial L}{\partial z_j} \tag{40}$$

Similar to the derivatives of the final layer, these derivatives represent the change in loss with respect to the weights and biases for weight calculation. To align with the implementation of this paper, slightly different formulae are used due to the opposite direction of traversal when iterating through every hidden and input layer compared to the conventional method. Let k represent the amount of layers.

$$\frac{\partial L}{\partial z_j} = Q' \left(z_{k-(j+1)} \right) \cdot \left(w_{k-j} \right)^T \cdot \frac{\partial L}{\partial z_{1+(k-j)}} \tag{41}$$

$$\frac{\partial L}{\partial w_{i,j}} = \begin{cases} \frac{\partial L}{\partial z_j} \cdot \left(a_{k-(j+2)}\right)^T & k-(j+2) \neq 0 \\ \frac{\partial L}{\partial z_j} \cdot x^T & k-(j+2) = 0 \end{cases} \tag{42}$$

$$\frac{\partial L}{\partial b_{i,j}} = \frac{\partial L}{\partial z_j} \tag{43}$$

Now, backpropagation for every layer is possible; both in theory and in code.

3.3 Weight Adjustments

Weight adjustments are conducted after calculating the necessary derivatives. For every single layer, the following formulae are conducted to adjust said weights, with some learning rate α:

$$w_i := w_i - \alpha \cdot \frac{\partial L}{\partial w_{i,j}} \tag{44}$$

$$b_i := b_i - \alpha \cdot \frac{\partial L}{\partial b_{i,j}} \tag{45}$$

4 Training Loop and Pseudocode

With every single equation required for the function the neural network derived, these formulae can be leveraged in an algorithm. First observe the pseudocode for the backward process, which is how the model adjusts weights.

Algorithm 1 Backward Process

```
$\hat{y} \leftarrow a$
$L \leftarrow -\sum_{i=1}^{n} (y_i log(\hat{y_i}))$
for $i = 0$ To $n_L - 1$ do
    if $i = 0$ then
        $\frac{\partial L}{\partial z_i} \leftarrow \hat{y_n} - y_n$
    else
        $\frac{\partial L}{\partial z_i} \leftarrow (w_{n-i})^T \cdot \frac{\partial L}{\partial z_{n-i}} \cdot Q'(z_{n-(i+1)})$
    end if
    $\frac{\partial L}{\partial w_i} \leftarrow \frac{\partial L}{\partial z_{n-i}} \cdot \left(a_{n-(i+2)}\right)^T$
    $\frac{\partial L}{\partial b_i} \leftarrow \frac{\partial L}{\partial z_i}$
    $w_{n-(i+1)} \leftarrow w_{n-(i+1)} - \alpha \cdot \frac{\partial L}{\partial w_i}$
    $b_{n-(i+1)} \leftarrow b_{n-(i+1)} - \alpha \cdot \frac{\partial L}{\partial b_i}$
end for
```

To make predictions with the weights and biases, the model conducts the forward process. Observe the following pseudocode which shows the workings of forward propagation.

Algorithm 2 Forward Process

```
for $i = 0$ To $n_L - 1$ do
    if $i = 0$ then
        $z_i \leftarrow w_i x + b_i$
    else
        $z_i \leftarrow w_i a_{i-1} + b_i$
    end if
    $a_i \leftarrow Q(z_i)$
end for
```

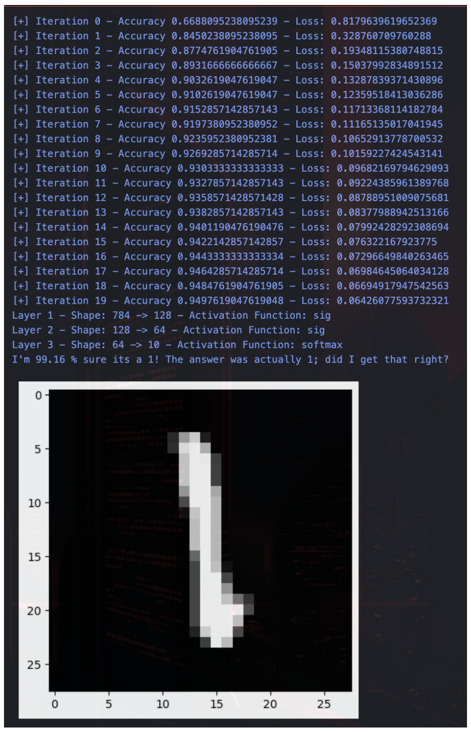
To fully train the model, the forward and backward processes are combined into a training loop. Since the batch size is only 1, the algorithm is quite slow.

Algorithm 3 Training Algorithm

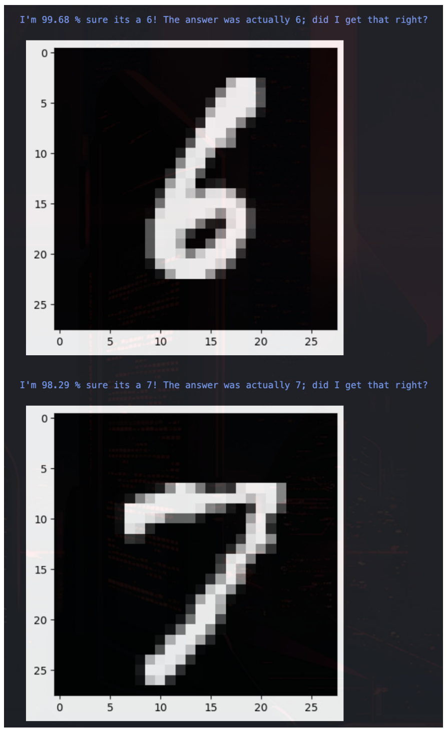
```
Ensure: $\lim_{I \to \infty} L = 0$
  $E \leftarrow k$
  $n_L \leftarrow p$
  $n \leftarrow n_L - 1$
  $n_I \leftarrow q$
  $I \leftarrow 0$
  for $I = 0$ To $E - 1$ do
      for $J = 0$ To $n_I - 1$ do
          Conduct the forward process (Algorithm 2)
          Conduct the backward process (Algorithm 1)
      end for
  end for
```

5 In Practice

With every equation and a general algorithm to follow, an actual implementation is more than within reach. For this paper, an implementation has been written to solve the problem of classifying simple handwritten digits. In practice, the implementation of the pseudocode and formulae are demonstrated in the following images. The model has been trained for 20 epochs, with 4 layers of 784, 128, 64 and 10 respectively. Although accuracy will eventually converge, it can be increased by increasing the number of layers and neurons per hidden layer.



```
[+] Iteration 0 - Accuracy 0.6688095238095239 - Loss: 0.8179639619652369
[+] Iteration 1 - Accuracy 0.8450238095238095 - Loss: 0.328760709760288
[+] Iteration 2 - Accuracy 0.8774761904761905 - Loss: 0.19348115380748815
[+] Iteration 3 - Accuracy 0.8931666666666667 - Loss: 0.15037992834891512
[+] Iteration 4 - Accuracy 0.9032619047619047 - Loss: 0.13287839371430896
[+] Iteration 5 - Accuracy 0.9102619047619047 - Loss: 0.12359518413036286
[+] Iteration 6 - Accuracy 0.9152857142857143 - Loss: 0.11713368114182784
[+] Iteration 7 - Accuracy 0.9197380952380952 - Loss: 0.11165135017041945
[+] Iteration 8 - Accuracy 0.9235952380952381 - Loss: 0.10652913778700532
[+] Iteration 9 - Accuracy 0.9269285714285714 - Loss: 0.10159227424543141
[+] Iteration 10 - Accuracy 0.9303333333333333 - Loss: 0.09682169794629093
[+] Iteration 11 - Accuracy 0.9327857142857143 - Loss: 0.09224385961389768
[+] Iteration 12 - Accuracy 0.9358571428571428 - Loss: 0.08788951009075681
[+] Iteration 13 - Accuracy 0.9382857142857143 - Loss: 0.08377988942513166
[+] Iteration 14 - Accuracy 0.9401190476190476 - Loss: 0.07992428292308694
[+] Iteration 15 - Accuracy 0.9422142857142857 - Loss: 0.07632216792377 5
[+] Iteration 16 - Accuracy 0.9443333333333334 - Loss: 0.07296649840263465
[+] Iteration 17 - Accuracy 0.9464285714285714 - Loss: 0.06984645064034128
[+] Iteration 18 - Accuracy 0.9484761904761905 - Loss: 0.06694917947542563
[+] Iteration 19 - Accuracy 0.9497619047619048 - Loss: 0.06426077593732321
Layer 1 - Shape: 784 -> 128 - Activation Function: sig
Layer 2 - Shape: 128 -> 64 - Activation Function: sig
Layer 3 - Shape: 64 -> 10 - Activation Function: softmax
I'm 99.16 % sure its a 1! The answer was actually 1; did I get that right?
```



```
I'm 99.68 % sure its a 6! The answer was actually 6; did I get that right?
```

```
I'm 98.29 % sure its a 7! The answer was actually 7; did I get that right?
```