Neural Network

Jack Schamburg

May 31, 2017

Contents

1 Manual calculation for a small neural network

1.1 Forward propagation steps

Steps for the manual calculation of the given neural network. Numbers are rounded to 3 decimal places.

Let $W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.1 & 0.1 \end{bmatrix}$

Let $W_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix}$

Let $b_1 = \begin{bmatrix} w_9 \\ w_{10} \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$

Let $b_2 = \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$

If $X = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix}$ with labels $Y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Let the $\oplus$ binary operator be defined as: add each vector element to each of the matrix elements in the same row. Let S_i denote summation of layer i (pre-sigmoid), H stand for 'hidden neuron layer' and $\hat{Y}$ be the output.

$$S_1 = W_1^T X \oplus b_1 = \begin{bmatrix} 0.1 & 0.1 \\ 0.2 & 0.1 \end{bmatrix} \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix} \oplus \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.12 & 0.13 \\ 0.13 & 0.14 \end{bmatrix}$$

Let $\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$$H = \sigma(S_1) = \begin{bmatrix} 0.530 & 0.532 \\ 0.532 & 0.535 \end{bmatrix}$$

$$S_2 = W_2^T H \oplus b_2 = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix} \begin{bmatrix} 0.530 & 0.532 \\ 0.532 & 0.535 \end{bmatrix} \oplus \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.206 & 0.207 \\ 0.259 & 0.260 \end{bmatrix}$$

$$\hat{Y} = \sigma(S_2) = \begin{bmatrix} 0.551 & 0.552 \\ 0.565 & 0.565 \end{bmatrix}$$

1.2 Back propagation steps

Steps for performing back propagation on each of the weights. Numbers displayed in section 1.2.1 are rounded to 3 decimal places and 5 decimal places in 1.2.2 and 1.2.3. Although rounded in the report, the numbers will be properly calculated using the python inline interpreter and therefore not rounded.

Let E be the mean squared error function:

$$E = \sum_{i=1}^{K} \frac{1}{K}(Y_i - \hat{Y}_i)^2$$

Where K is the number of samples; number of rows in X and Y.

1.2.1 Calculating $\frac{\partial E}{\partial W_2}$

$$\frac{\partial E}{\partial W_2} = \sum_{i=1}^{2}(\hat{Y}_i - Y_i) \cdot \frac{\partial \hat{Y}_i}{\partial W_2}$$

$$\frac{\partial E}{\partial W_2} = \sum_{i=1}^{2}(Y_i - \hat{Y}_i) \cdot \frac{\partial \hat{Y}_i}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_2}$$

Let $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$

$$\frac{\partial \hat{Y}}{\partial S_2} = \sigma'(S_2)$$

$$\frac{\partial S_2}{\partial W_2} = H$$

As matrix multiplication sums together each sample's error by definition, the sumation symbol is not needed. Also note that $\odot$ denotes elementwise multiplication

Let $\delta_3 = (\hat{Y} - Y) \odot \sigma'(S_2)$ which will be later used in section 1.2.2

$$(\hat{Y} - Y) = \begin{bmatrix} 0.551 & 0.552 \\ 0.565 & 0.565 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.449 & 0.552 \\ 0.565 & -0.435 \end{bmatrix}$$

$$\sigma'(S_2) = \begin{bmatrix} 0.247 & 0.247 \\ 0.246 & 0.246 \end{bmatrix}$$

$$\delta_3 = \begin{bmatrix} -0.449 & 0.552 \\ 0.565 & -0.435 \end{bmatrix} \odot \begin{bmatrix} 0.247 & 0.247 \\ 0.246 & 0.246 \end{bmatrix} = \begin{bmatrix} -0.111 & 0.136 \\ 0.139 & -0.107 \end{bmatrix}$$

$$\frac{\partial E}{\partial W_2} = H \cdot \begin{bmatrix} -0.111 & 0.139 \\ 0.136 & -0.107 \end{bmatrix}$$

$$\frac{\partial E}{\partial W_2} = \begin{bmatrix} 0.530 & 0.532 \\ 0.532 & 0.535 \end{bmatrix} \cdot \begin{bmatrix} -0.111 & 0.139 \\ 0.136 & -0.107 \end{bmatrix}$$

$$\frac{\partial E}{\partial W_2} = \begin{bmatrix} 0.014 & 0.017 \\ 0.014 & 0.017 \end{bmatrix}$$

1.2.2 Calculating $\frac{\partial E}{\partial W_1}$

$$\frac{\partial E}{\partial W_1} = -(Y - \hat{Y}) \cdot \frac{\partial \hat{Y}}{\partial W_1}$$

$$\frac{\partial E}{\partial W_1} = -(Y - \hat{Y}) \cdot \frac{\partial \hat{Y}}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_1}$$

$$\frac{\partial E}{\partial W_1} = \delta_3 \cdot \frac{\partial S_2}{\partial W_1}$$

$$\frac{\partial E}{\partial W_1} = \delta_3 \cdot \frac{\partial S_2}{\partial H} \cdot \frac{\partial H}{\partial W_1}$$

$$\frac{\partial E}{\partial W_1} = \delta_3 \cdot W_2 \cdot \frac{\partial H}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_1}$$

$$\frac{\partial E}{\partial W_1} = \delta_3 \cdot W_2 \cdot \sigma'(S_1) \cdot X$$

Let $\delta_2 = (W_2 \cdot \delta_3) \odot \sigma'(S1)$

$$\sigma'(S2) = \begin{bmatrix} 0.206 & 0.206 \\ 0.259 & 0.260 \end{bmatrix}$$

$$\delta_2 = (\begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} -0.111 & 0.136 \\ 0.139 & -0.107 \end{bmatrix}) \odot \begin{bmatrix} 0.206 & 0.206 \\ 0.259 & 0.260 \end{bmatrix}$$

$$\delta_2 = \begin{bmatrix} 0.003 & 0.003 \\ 0.017 & -0.008 \end{bmatrix} \odot \begin{bmatrix} 0.206 & 0.206 \\ 0.259 & 0.260 \end{bmatrix} = \begin{bmatrix} 0.001 & 0.001 \\ 0.004 & -0.002 \end{bmatrix}$$

$$\frac{\partial E}{\partial W_1} = X \cdot \delta_2^T$$

$$\frac{\partial E}{\partial W_1} = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 0.001 & 0.004 \\ 0.001 & -0.002 \end{bmatrix} = \begin{bmatrix} 0.00014 & 0.00022 \\ 0.00022 & 0.00003 \end{bmatrix}$$

1.2.3 Updating Weights

To update W_1 and W_2 we can simply substract the partial derivatives found previously multiplied by a learning rate and dividing by how many samples (so that we're updating the weights with the average of the gradients).

$\eta = 0.1, m = 2$

$$W_1' = W_1 - \frac{\eta}{m} \times \frac{\partial E}{\partial W_1}$$

$$W_1' = \begin{bmatrix} 0.1 & 0.2 \\ 0.1 & 0.1 \end{bmatrix} - \frac{0.1}{2} \times \begin{bmatrix} 0.00014 & 0.00022 \\ 0.00022 & 0.00003 \end{bmatrix} = \begin{bmatrix} 0.09999 & 0.19998 \\ 0.09998 & 0.09999 \end{bmatrix}$$

$$W_2' = W_2 - \frac{\eta}{m} \times \frac{\partial E}{\partial W_2}$$

$$W_2' = \begin{bmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{bmatrix} - \frac{0.1}{2} \times \begin{bmatrix} 0.014 & 0.017 \\ 0.014 & 0.017 \end{bmatrix} = \begin{bmatrix} 0.09862 & 0.09834 \\ 0.09862 & 0.19833 \end{bmatrix}$$

As δ_2 and δ_3 were never matrix multiplied and therefore never summed against every sample, this step must be done manually to squash it into a vector.

$$b_1' = b_1 - \frac{\eta}{m} \times \sum_{i=1}^{2} \delta_2^i$$

$$b_1' = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} - \frac{0.1}{2} \times \begin{bmatrix} 0.002 \\ 0.002 \end{bmatrix} = \begin{bmatrix} 0.09993 \\ 0.09989 \end{bmatrix}$$

$$b_2' = b_2 - \frac{\eta}{m} \times \sum_{i=1}^{2} \delta_3^i$$

$$b_2' = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix} - \frac{0.1}{2} \times \begin{bmatrix} 0.247 \\ 0.029 \end{bmatrix} = \begin{bmatrix} 0.09873 \\ 0.09841 \end{bmatrix}$$

2 Implementation in Python

2.1 Section 1 Implementation

Section 1's neural network of 2 input, 2 hidden and 2 output neurons was implemented in python3 (see *basic_neural_network.py*) using stochastic gradient descent with a minibatch size of 2 (a full batch essentially) and was trained over several thousand epochs.

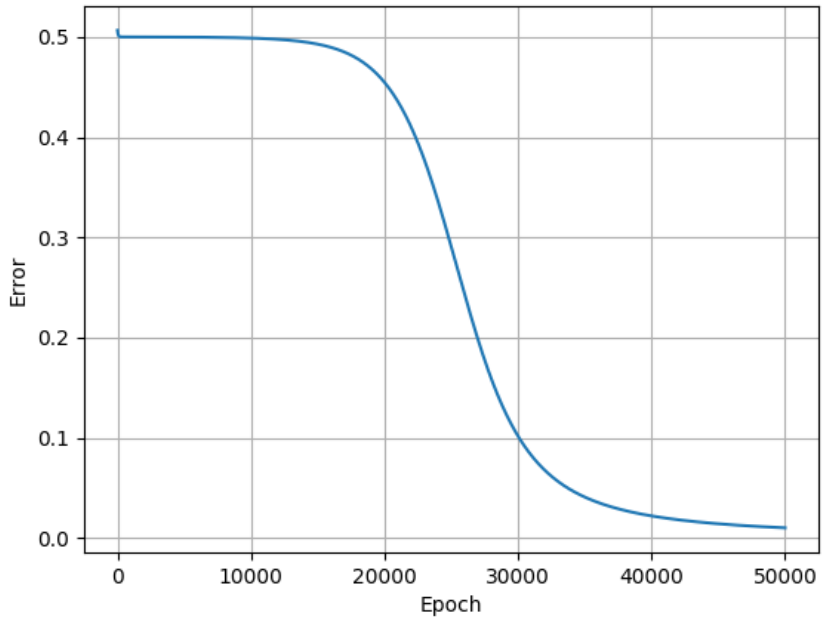

Figure 1: The mean square error of the basic neural network implementation over epochs

2.1.1 Change in weights over 3 epochs

The following table show cases how the values of each of the 2 weight matricies and 2 bias vectors change over each epoch.

Weights	**Epoch 1**	**Epoch 2**	**Epoch 3**
W_1	$\begin{bmatrix} 0.09999288 & 0.19998892 \\ 0.09998922 & 0.09999857 \end{bmatrix}$	$\begin{bmatrix} 0.09998587 & 0.19997799 \\ 0.09997859 & 0.09999736 \end{bmatrix}$	$\begin{bmatrix} 0.09997899 & 0.19996722 \\ 0.09996813 & 0.09999637 \end{bmatrix}$
W_2	$\begin{bmatrix} 0.09930886 & 0.09917138 \\ 0.09930571 & 0.19916741 \end{bmatrix}$	$\begin{bmatrix} 0.09862415 & 0.09835049 \\ 0.09861788 & 0.19834262 \end{bmatrix}$	$\begin{bmatrix} 0.09794581 & 0.09753728 \\ 0.09793647 & 0.19752554 \end{bmatrix}$
b_1	$\begin{bmatrix} 0.09992876 \\ 0.09988917 \end{bmatrix}$	$\begin{bmatrix} 0.09985872 \\ 0.0997799 \end{bmatrix}$	$\begin{bmatrix} 0.099789887 \\ 0.09967217 \end{bmatrix}$
b_2	$\begin{bmatrix} 0.09872793 \\ 0.09841132 \end{bmatrix}$	$\begin{bmatrix} 0.09746791 \\ 0.09683713 \end{bmatrix}$	$\begin{bmatrix} 0.09621986 \\ 0.09527734 \end{bmatrix}$

From the above table it is apparent that the weights of the neuron to neuron synapses as well as the biases are changing, with all of the weights being decreased over time. The reason for weights decreasing is due to the optimal weight configuration values being less than the starting weights.

2.1.2 Verifying Section 1

The calculations for the partial derivatives found in section 1 will now be verified by printing each of the gradients from the python implementation.

```
EPOCH: 1
W1 [[ 0.09999288  0.19998892]
 [ 0.09998922  0.09999857]]
W2 [[ 0.09930886  0.09917138]
 [ 0.09930571  0.19916741]]
b1 [[ 0.09992876]
 [ 0.09988917]]
b2 [[ 0.09872793]
 [ 0.09841132]]
dEdW1 [[  1.42478355e-04   2.21664739e-04]
 [  2.15688433e-04   2.86203723e-05]]
dEdW2 [[ 0.01382276  0.01657242]
 [ 0.0138859   0.01665172]]
dEdb1 [[ 0.01272073]
 [ 0.01588682]]
dEdb2 [[ 0.01272073]
 [ 0.01588682]]
EPOCH: 2
W1 [[ 0.09998587  0.19997799]
 [ 0.09997859  0.09999736]]
W2 [[ 0.09862415  0.09835049]
 [ 0.09861788  0.19834262]]
b1 [[ 0.09985872]
 [ 0.0997799 ]]
b2 [[ 0.09746791]
 [ 0.09683713]]
dEdW1 [[  1.40072557e-04   2.18534619e-04]
 [  2.12452887e-04   2.42005519e-05]]
dEdW2 [[ 0.01369419  0.01641773]
 [ 0.01375652  0.01649598]]
dEdb1 [[ 0.01260012]
 [ 0.01574183]]
dEdb2 [[ 0.01260012]
 [ 0.01574183]]
EPOCH: 3
W1 [[ 0.09997899  0.19996722]
 [ 0.09996813  0.09999637]]
W2 [[ 0.09794581  0.09753728]
 [ 0.09793647  0.19752554]]
b1 [[ 0.09978987]
 [ 0.09967217]]
b2 [[ 0.09621986]
 [ 0.09527734]]
dEdW1 [[  1.37707781e-04   2.15451369e-04]
 [  2.09276317e-04   1.98502725e-05]]
dEdW2 [[ 0.01356675  0.0162643 ]
 [ 0.01362827  0.0163415 ]]
dEdb1 [[ 0.01248056]
 [ 0.01559799]]
dEdb2 [[ 0.01248056]
 [ 0.01559799]]
```

Figure 2: Screenshot of output from the first 3 epochs of basic_neural_network.py

2.2 MINST Neural Network

Following on from the previous implementation of section 1, a neural network was implemented in python3 (see file *neural_network.py*) that given a training data set was able to learn the details of hand written digits from 0-9. This neural network has 784 input, 30 hidden and 10 output neurons. The major difference between this implementation and the previous, is that this problem is of classification instead of regression and therefore a softmax function was used instead of a sigmoid for the final output $\hat{Y}$. As the softmax function returns the probability distribution of each exclusive class, it is possible to minimize the error by maximizing the probability of a class when its matching digit appears (e.g. when it is given an 8, the optimal output of the softmax function would be 1 for the 8 class and 0 for the other classes). The following sections explores the results of the neural network as the hyper parameters were altered.

2.2.1 Initial Hyperparamater Results

The initial hyperparamters of the neural network was stated to include a mini-batch size of 20 samples, a learning rate of 3 and 30 epochs.

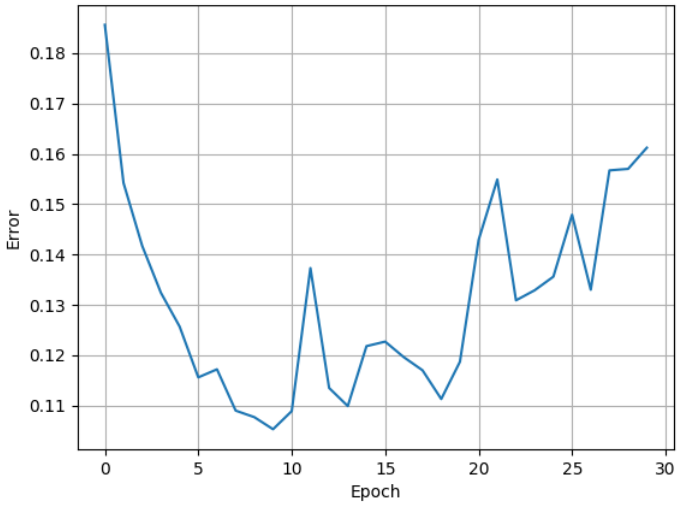

Figure 3: Error of the neural network on each epoch

The results shown above showcases underfitting between epoch 0 and epoch 9, in which the network still does not know enough and is 'learning' the details of the classes. At epoch 9, the network is at its most accurate state guessing over 90% of the test digits. After epoch 9, the network begins to overfit to the training data and learn the noise associated with it. Therefore, if these hyper parameters were to be used, the network should be trained until epoch 9 and

then be stopped. This method was used to generate the predicted labels in files PredictTestY.csv.gz and PredictTestY2.csv.gz.

2.2.2 Altering the Learning Rate

The learning rate of a neural network is responsible for how big of a 'step' the updated weights take in the direction of the global minima. The trade-offs associated with the learning rates are that if the step is too small, the network will take far too long to train, whilst having to large a learning rate will result in the algorithm 'steping over' or 'out' of the minima. These tradeoffs in the hyper-parameter are showcased below.

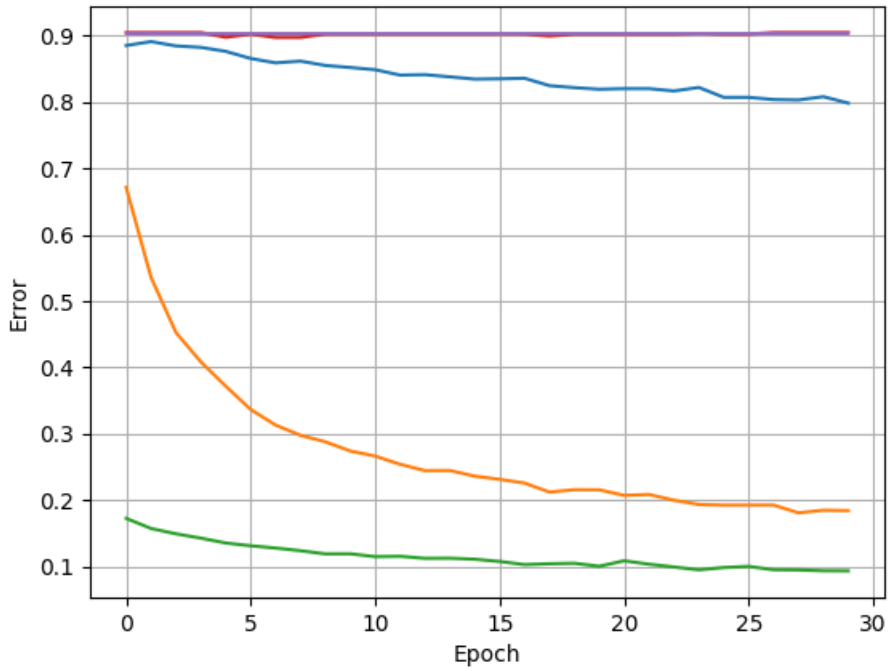

Figure 4: Comparison of each neural networks error with differing learning rates

- $\eta = 100$ - Purple
- $\eta = 10$ - Red
- $\eta = 10$ - Blue
- $\eta = 0.1$ - Yellow
- $\eta = 0.001$ - Green

This comparison highlights the tradeoff mentioned above in which networks with large learning rates almost never converge to an error of near 0 as they are taking far to large a stride back and forth and how as the learning rate becomes more sensible, the network can take its time and eventually find and travel towards the global minima.

2.2.3 Altering the Minibatch Size

The size of the minibatch is similar to previous remarks as they both ends (small or large), both have pros and cons. Online learning (minibatchsize of 1) is known to be faster but does not always learn correctly as certain steps to the global minima are 'undone' by others. Large minibatch sizes are slower to compute but yield lower error levels as a 'compromise' is made between each of the gradients in which direction to move in.

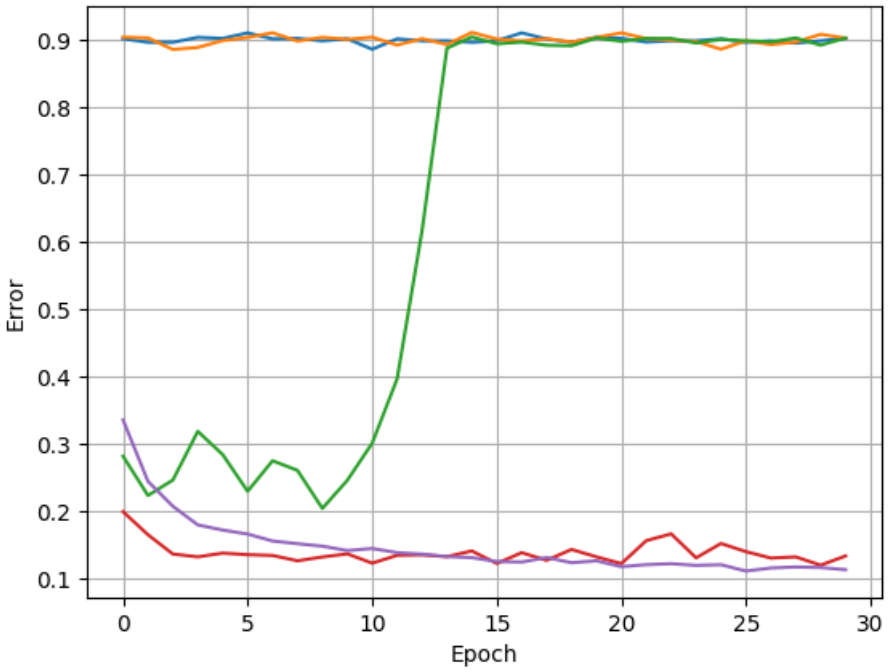

Figure 5: Comparison of each neural networks error with differing minibatch sizes

- 100 = purple
- 20 = red
- 10 = green

- 5 = yellow
- 1 = blue

2.2.4 Altering Additional Hyperparameters

Learning rates and minibatch sizes are not the only hyper-parameters that can be tweaked and adjusted to improve performance for a particular problem. For example the number of hidden layers in a network can easily be adjusted to try and improve performance.

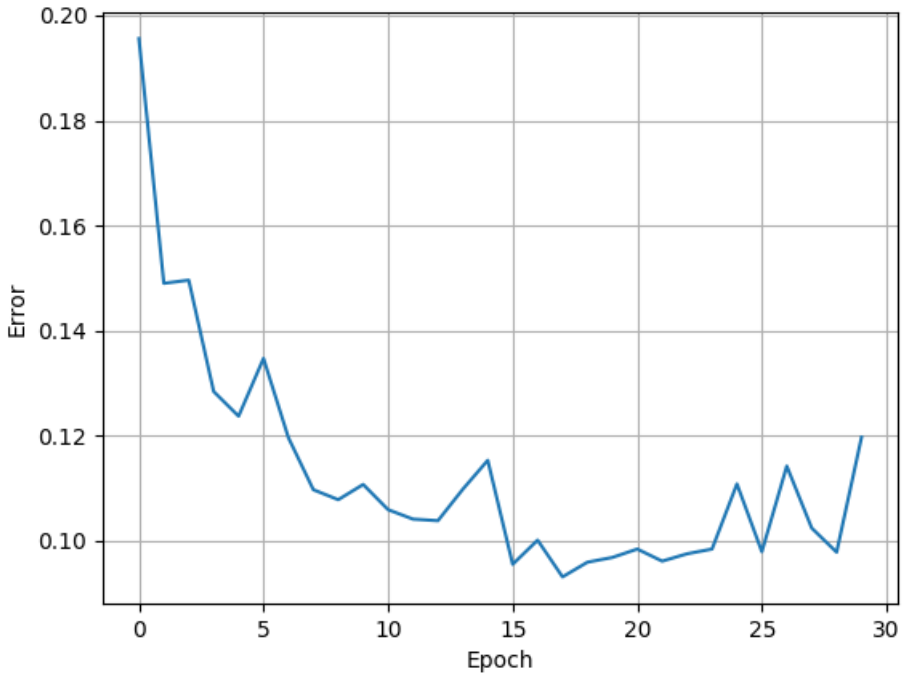

Figure 6: The neural network from section 2.2.1 but with hidden layer set to 35 neurons

As shown above, the network configuration from section 2.2.1 can be slightly changed to include 35 hidden neurons and further decrease the error from 0.09 to 0.05 (at both of their best epochs).