Artificial Neural Network Project 3

Kristen Mabry

Introduction

This report discusses the implementation of an Artificial Neural Network algorithm to classify 10 pixelated letters based on 8 different moment values. Different variables of the algorithm were adjusted to observe their effects on the success rate of the classification. These cases and the data used will be covered before presenting and analyzing the results.

Background and Implementation

Artificial Neural Network Algorithm

An artificial neural network (ANN) algorithm represents a mathematical model of neurons in the brain. Inputs are multiplied by weights and added up at the nodes with several layers of nodes making up the network. In between each layer, the nodes go through an activation function to make the transformation nonlinear. The activation function for this project was the sigmoid function,

$$f(x) = \frac{1}{1 + e^{-\lambda x}}, \tag{1}$$

with λ having a default value of 2.

To train the network, gradient descent is used with back propagation to adjust the weights until the error in the output is a minimum. The outputs can vary based on how the network is trained. In the network for this project, one-hot encoding was used but alternate encodings could be used as well.

To perform gradient descent, the weights are adjusted based on the current error and repeated until one of three conditions are met. These three conditions are the current error being less than the maximum error, the current change in the weights being less than the minimum change, and the maximum number of iterations reached. The first condition relates to finding the minimum of the cost function, J(w), defined as

$$J(w) = \frac{1}{2}\left|\left|t_k - z_k\right|\right|^2 \tag{2}$$

where t_k is the expected output and z_k is the current output, and the second condition makes sure that the algorithm is making progress towards the minimum of the cost function. The last condition is there in case the minimum is never reached.

The weights are adjusted by subtracting the product of the learning rate, η, and the partial derivative of the cost function with respect to the weights. For the hidden weights, w_{ji}, and the outer weights, w_{kj}, the change in weights comes out to

$$\Delta w_{ji} = -\eta \frac{\partial J}{\partial w_{ji}} = \eta x_i \delta_j = \eta \left[\sum_{k=1}^{c} w_{kj} \delta_k \right] f'(net_j) x_i \quad (3)$$

and

$$\Delta w_{kj} = -\eta \frac{\partial J}{\partial w_{kj}} = \eta y_i \delta_k = \eta (t_k - z_k) f'(net_k) x_i \quad (4)$$

with

$$\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj} \delta_k \quad (5)$$

and

$$\delta_k = (t_k - z_k) f'(net_k). \quad (6)$$

The variable, x_i, is the input, net_j the hidden layer before the activation function is applied, y_j the hidden layer after the activation function is applied, and net_k the outer layer before the activation function is applied.

The location where the weights are adjusted determines whether the network is trained by pattern or by epoch. Each iteration, the data is randomly shuffled and looped through to determine the change in weights. Training by epoch has the weights being adjusted after all the training data has been looped through. An extra variable is needed to store the change in weights to add at the end. Training by pattern has the weights being adjusted after every item in the training data. For both epoch and pattern, J(w) is added up after every item in the training data.

For this project, the ANN algorithm was implemented using python along with the numpy, pandas, sklearn, and matplotlib libraries. The input was a 1-dimensional vector of length 8 based on the features described in the next section, and the output used one-hot encoding which resulted in a 1-dimensional vector of length 10. The initial values of the weights were given and were made up of random values between -0.5 and 0.5. The network was set up to handle 2 layers, and the base case was trained by epoch with 4 nodes in the hidden layer. The variable values used at the start are summarized in Table 1.

Table 1 Initial Variable Values

Variable	Value
Learning Rate	0.1
Number of Hidden Nodes	4
Maximum # of Iterations	2000
Minimum Change	0.005
Max Error	16
Training By	Epoch
Bias	None
Momentum	None

To evaluate the success of the algorithm, confusion matrices were constructed. This matrix is a grid with the algorithm's classification on one axis and the actual class on the other. The goal is for the non-zero numbers to show up only on the upper left to lower right diagonal, meaning the algorithm was correct. The incorrect classifications are added up in an extra row and column called the error columns. The error I column represents how many inputs were classified incorrectly while the error II row represents what the most popular incorrect choice was. The sum of either the row or column – or the total error – is in the intersecting box in the lower right corner. By dividing the total error by the total number of data points in the evaluation set, the error rate can be achieved. The inverse of that would be the success rate.

Training and Evaluation Sets

The training set contained 100 rows with 10 of each letter. This set, as well as the evaluation sets, had 8 moment values for each data point. There were two sets of evaluation data – one with 100 rows and one with 200 rows. These sets also had an even distribution of letters and identified the letter for each row which allowed the success rate of the algorithm to be calculated.

There were 10 letters to be classified which made up the classes: 'a', 'c', 'e', 'm', 'n', 'o', 'r', 's', 'x', and 'z.' Each letter had the same 8 moment values which made up the features. These moments are m_{00}, μ_{02}, μ_{11}, μ_{02}, μ_{03}, μ_{12}, μ_{21}, and μ_{30}. While these values were already given in the text file, they could be calculated using the equations

$$m_{pq} = \sum_j \left\{ \sum_i \{ i^p j^q X(i,j) \} \right\} \tag{7}$$

and

$$\mu_{pq} = \sum_j \left\{ \sum_i \{ (i - i_{mean})^p (j - j_{mean})^q X(i,j) \} \right\} \tag{8}$$

where

$$i_{mean} = \frac{m_{10}}{m_{00}} \tag{9}$$

and

$$j_{mean} = \frac{m_{01}}{m_{00}}. \tag{10}$$

Variables p and q relate to the coordinates if the pixels making up each letter were put into a grid. X(i, j) would be 1 if a black pixel exists and 0 if it does not. These values relate to different properties of the letters such as center of mass, skew, and standard distribution horizontally and vertically.

Classifier Results

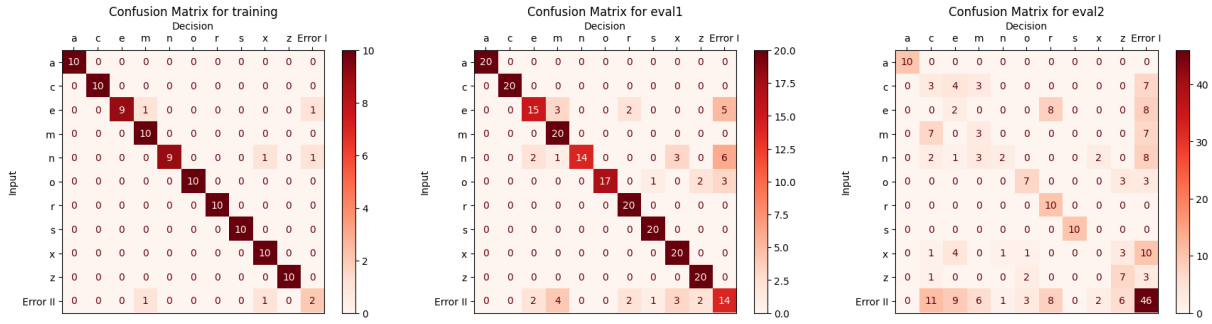
The first case run, based on the parameters in Table 1, was used as the comparison for all other cases. The results shown in this section are a random sample from training, so slightly

different results could happen if the training was run again due to the element of randomness with the shuffling of the data before each epoch. To display the results, confusion matrices were created and success rates calculated.

Confusion Matrices

Confusion matrices were constructed with error I and error II columns which allowed an easy visualization of the success rate and which characters were being confused with each other. Each case has confusion matrices for the training data and both evaluation sets.

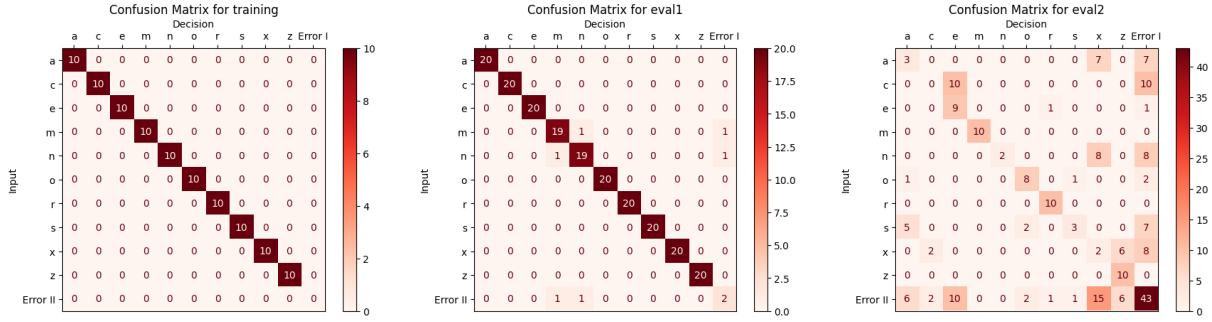
The matrices for the base case can be seen in Fig. 1. The matrix on the left is for the training data, and the next two matrices are for the evaluation sets.



(a) (b) (c)

Fig. 1 Confusion matrices for training by epoch (a) training data (b) evaluation set 1 (c) evaluation set 2

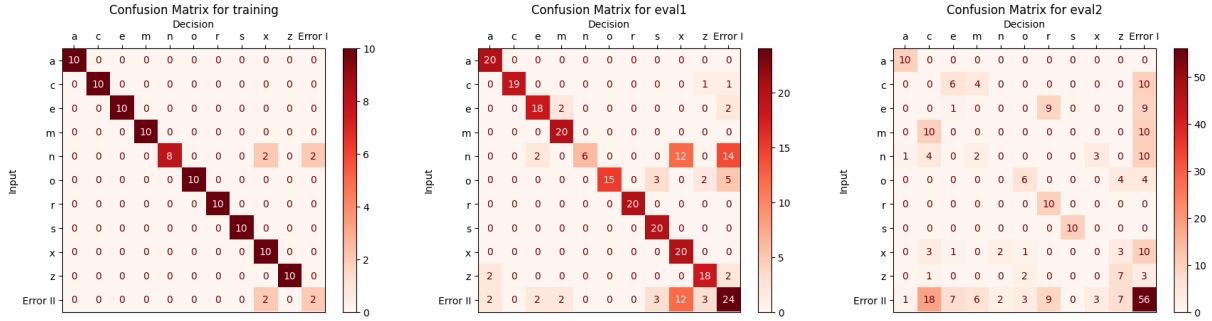
The training by pattern confusion matrices are displayed in Fig. 2. The results were slightly more accurate.



(a) (b) (c)

Fig. 2 Confusion matrices for training by pattern (a) training data (b) evaluation set 1 (c) evaluation set 2

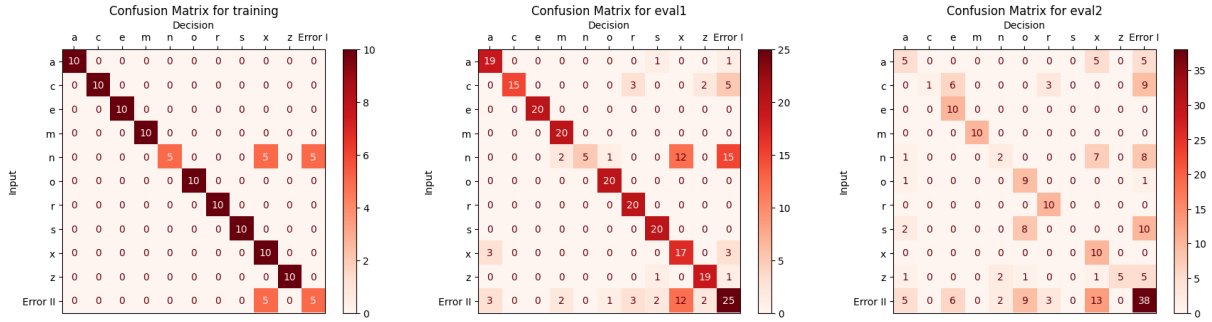
The next case used momentum which added part of the previous update to the weights in the next iteration. The α value, which determined the weight of the previous update, was 0.1. Fig. 3 shows the confusion matrices for this case.



(a) (b) (c)

Fig. 3 Confusion matrices for momentum (a) training data (b) evaluation set 1 (c) evaluation set 2

In Fig. 4, the confusion matrices for the case with bias can be seen. The bias was added to the input layer with a value of 1.



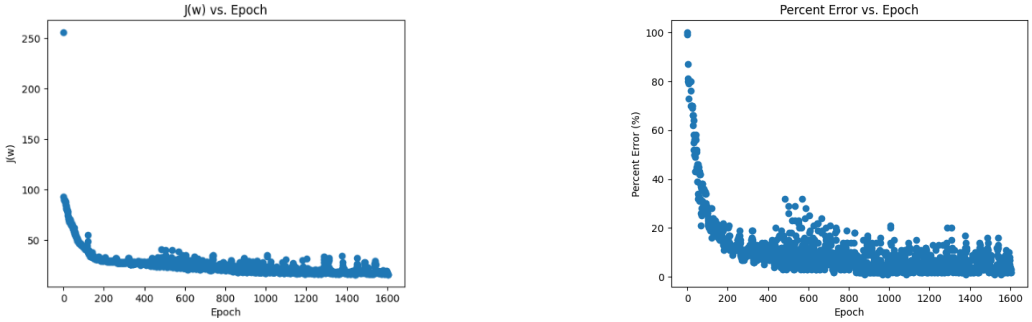
(a) (b) (c)

Fig. 4 Confusion matrices for bias (a) training data (b) evaluation set 1 (c) evaluation set 2

The lambda in the activation function was changed to view its effect. The values used were 0.5, 1, 2.5, and 3. The confusion matrices for these cases can be found in Appendix A. The learning rate was adjusted to 0.01, 0.05, 0.5, and 1. These matrices can be found in Appendix B. Lastly, the effect of the number of hidden nodes was tested with 2, 3, 5, and 6 nodes. These matrices are in Appendix C.

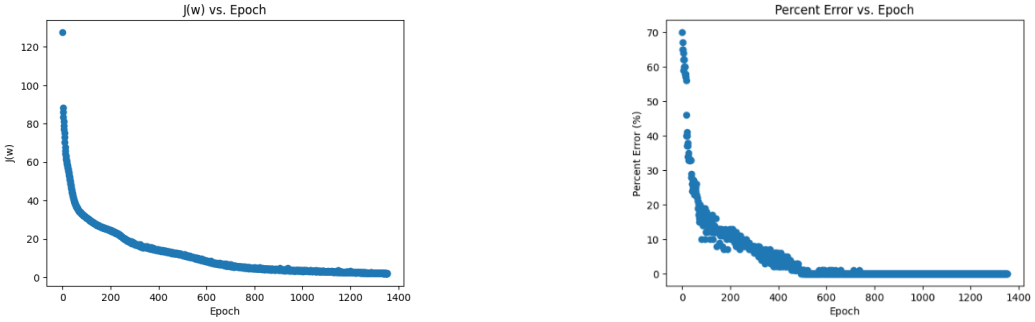
J(w) and Percent Error

The cost function, J(w), and the percent error of the training data were both plotted after each epoch in training to see how the network became more accurate as time went on. It also gives insight into how many epochs were needed to train the network. The plots for training by epoch can be seen in Fig. 5. Fig. 6 is for train by pattern, Fig. 7 for momentum, and Fig. 8 for bias. The plots for the other cases can be found in the Appendix.



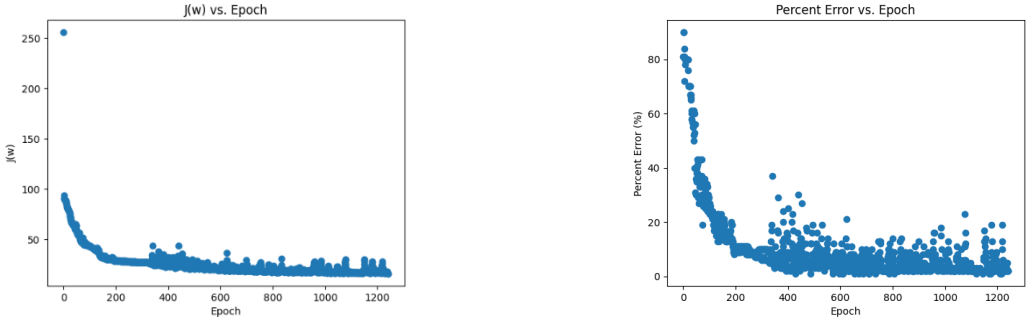
(a) (b)

Fig. 5 Error over epochs while training for train by epoch (a) J(w) (b) percent error



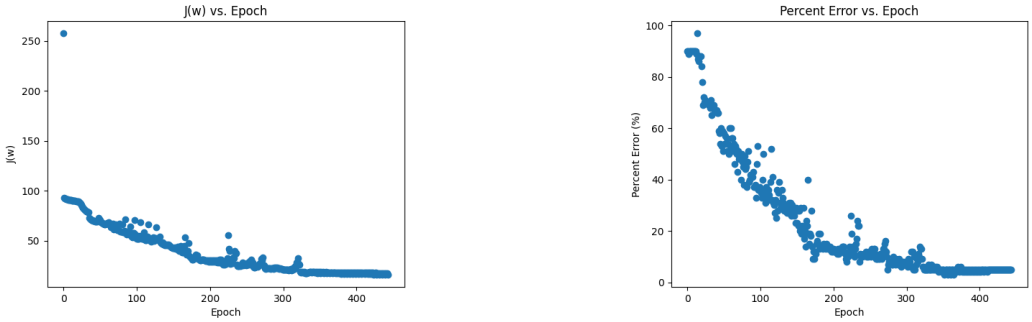
(a) (b)

Fig. 6 Error over epochs while training for train by pattern (a) J(w) (b) percent error



(a) (b)

Fig. 7 Error over epochs while training for momentum (a) J(w) (b) percent error



(a) (b)

Fig. 8 Error over epochs while training for bias (a) J(w) (b) percent error

Success Rates

The success rates were calculated by taking the total error in the confusion matrices and dividing it by the number of rows in the data set. The rates for the biggest changes are summarized in Table 2. The number of epochs performed before the training completed, as seen in the cost function and percent error plots in the previous section, are also shown.

Table 2 Success Rates for Initial Cases

Case	*Epoch (Base)*	*Pattern*	*Momentum*	*Bias*
Number of Epochs	1600	1350	1250	450
Training	98.0%	100.0%	98.0%	95.0%
Evaluation 1	93.0%	99.0%	88.0%	87.5%
Evaluation 2	54.0%	57.0%	44.0%	62.0%

Table 3 shows the success rates for the different values of lambda in the activation function.

Table 3 Success Rates for Different Lambdas

Lambda	*0.5*	*1*	*2 (Base)*	*2.5*	*3*
Number of Epochs	1600	1000	1600	2000	1400
Training	90.0%	98.0%	98.0%	92.0%	97.0%
Evaluation 1	85.0%	99.0%	93.0%	79.0%	81.0%
Evaluation 2	59.0%	54.0%	54.0%	50.0%	50.0%

The different learning rates have their success rates summarized in Table 4.

Table 4 Success Rates for Different Learning Rates

Learning Rate	*0.01*	*0.05*	*0.1 (Base)*	*0.5*	*1*
Number of Epochs	140	750	1600	65	1
Training	42.0%	96.0%	98.0%	10.0%	10.0%
Evaluation 1	47.0%	87.0%	93.0%	10.0%	10.0%
Evaluation 2	44.0%	49.0%	54.0%	10.0%	10.0%

The success rates for the different number of hidden nodes are shown in Table 5.

Table 5 Success Rates for Different Number of Hidden Nodes

Number of Hidden Nodes	*2*	*3*	*4 (Base)*	*5*	*6*
Number of Epochs	2000	2000	1600	110	160
Training	45.0%	83.0%	98.0%	98.0%	90.0%
Evaluation 1	40.5%	72.0%	93.0%	90.0%	86.5%
Evaluation 2	33.0%	38.0%	54.0%	45.0%	68.0%

Analysis

In general, the accuracy of the training data is significantly higher than the accuracy for the evaluation sets. This could be due to overfitting especially since most cases took over 1000 epochs to train.

Epoch vs. Pattern

Training by pattern proved to be more accurate than by epoch. Evaluation set 1 was better by 6%, and evaluation set 2 was better by 3%. The J(w) was also able to go below 2 for pattern whereas the J(w) for epoch was only able to go below 16. Looking at the J(w) plots, the one for train by pattern was a smooth curve while the one for train by epoch constantly went back up before reaching a new minimum.

Lambda in Activation Function

As lambda increased, the results became less accurate. When testing the smaller values, lambda of 1 was more accurate than a lambda of 2, but a lambda of 0.5 was less accurate again. This suggests that the optimal value is somewhere around 1 for this problem.

Learning Rate

The larger learning rates both resulted in 10% accuracy overall. This is because it was taking too large of jumps and missing the cost function's minimum every iteration. The learning rate of 0.01 had poor results with accuracy around 40%. Based on the low number of epochs, it likely stopped because the change was too small. The learning rate of 0.05 was only slightly less accurate than the base case of 0.1 which could mean it got stuck in a local minimum. The default learning rate was therefore the best one out of the cases tested.

Number of Hidden Nodes

The smaller number of hidden nodes did worse than the larger number. The drawbacks of a larger number of nodes are that there's less than 1 test data point for each weight which would make it more difficult to accurately train; however, a smaller number of nodes gives the network less flexibility which caused a bigger issue in this case. While the larger number of nodes generally performed worse than the base case, 6 hidden nodes showed an improvement for the second evaluation set by 14%.

Momentum vs. No Momentum

Adding momentum caused the accuracy to go down by up to 10%. As seen by the J(w) and percent error plots, this could be because the momentum was causing the gradient descent to keep going past the minimum once it was reached. This may be needed if the learning rate is small, so the algorithm doesn't stop in a local minimum. Since the default learning rate used did not seem to have that issue, momentum did not help in this case.

Bias vs. No Bias

Having a biased input caused the accuracy of the first evaluation set to go down but the second evaluation set to go up. Knowing from previous projects that the data in the second

evaluation set is much closer together, the bias was able to separate them more. The clear separation also allowed for the training to complete much faster with only 450 epochs run instead of 1600.

Share with Partner

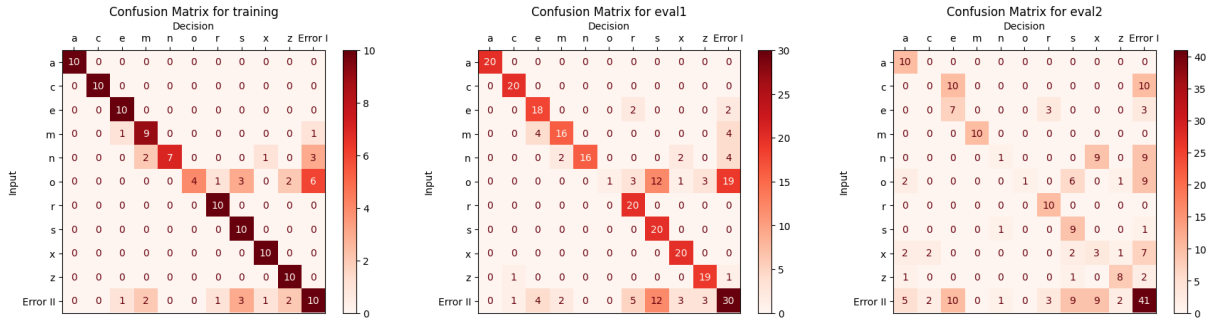
My partner for this project was Dmitry. I liked how he organized the functions into a class and did all the initialization of loading and normalizing the data in the initialization function. He also used numpy arrays instead of matrices which I learned that it allowed for more built-in functions to be used while I had to find workarounds for my code.

Conclusion

The artificial neural network was successfully implemented to identify the 10 letter classes using 8 features consisting of central moments. Different methods were used and variables altered to see their effects on the training of the neural network. Overall, this method has proven to be more accurate than the previous methods of k-nearest neighbor and Bayesian classifiers, and the ANN algorithm could be further improved by finding the best values for the learning rate and activation function lambda and using momentum, bias, and train by pattern where appropriate.

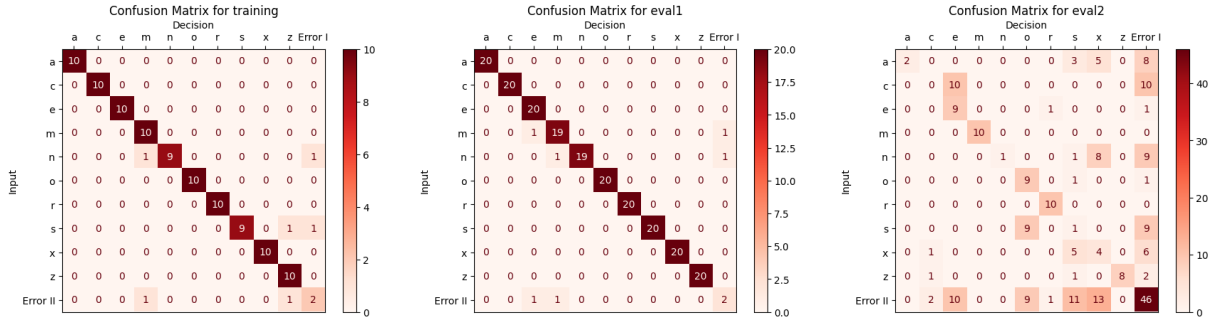
Appendix

Appendix A



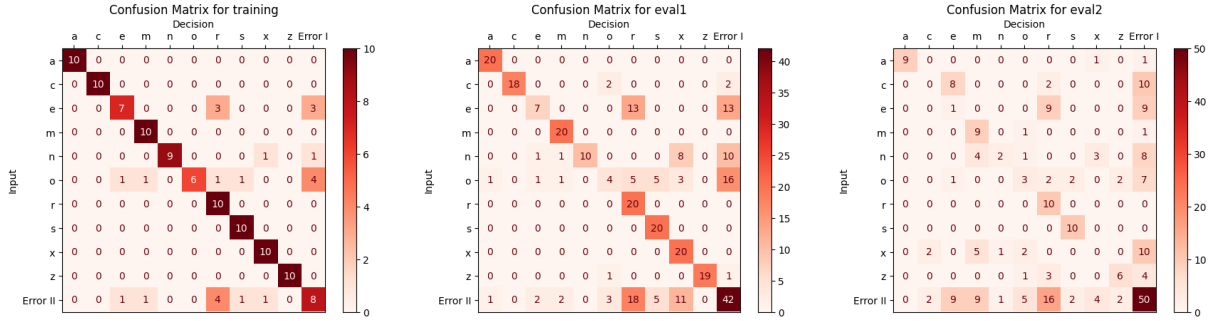
(a) (b) (c)

Fig. 9 Confusion matrices for lambda = 0.5 (a) training data (b) evaluation set 1 (c) evaluation set 2



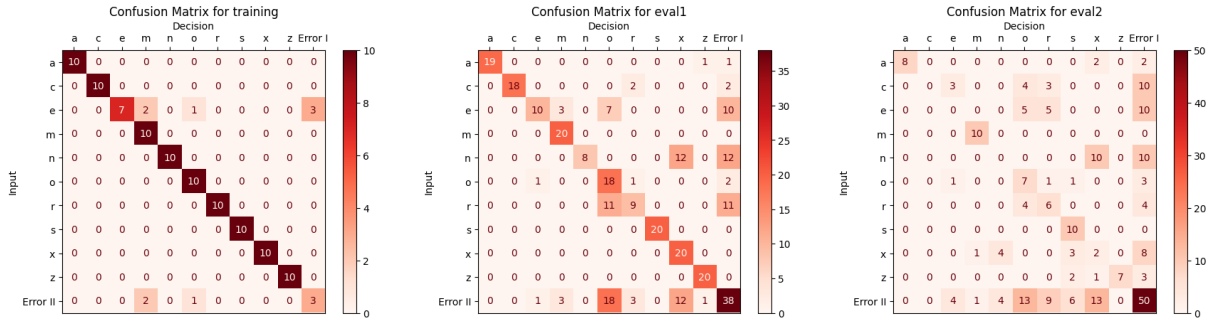
(a) (b) (c)

Fig. 10 Confusion matrices for lambda = 1.0 (a) training data (b) evaluation set 1 (c) evaluation set 2



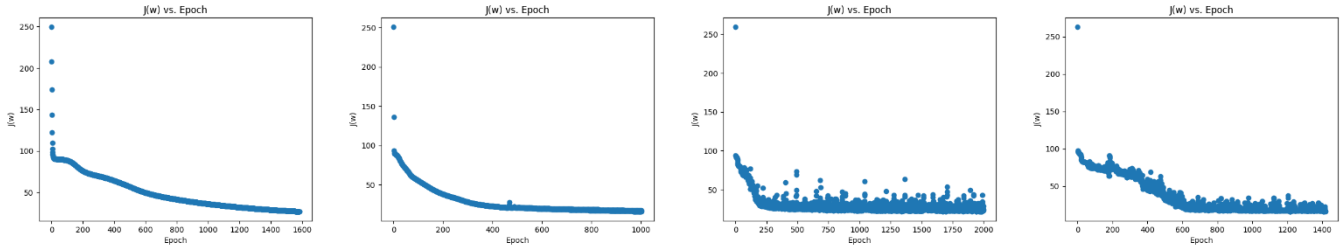
(a) (b) (c)

Fig. 11 Confusion matrices for lambda = 2.5 (a) training data (b) evaluation set 1 (c) evaluation set 2



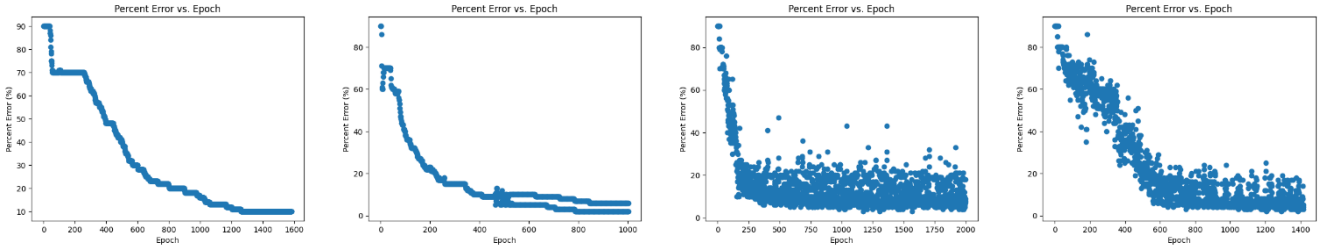
(a) (b) (c)

Fig. 12 Confusion matrices for lambda = 3.0 (a) training data (b) evaluation set 1 (c) evaluation set 2



(a) (b) (c) (d)

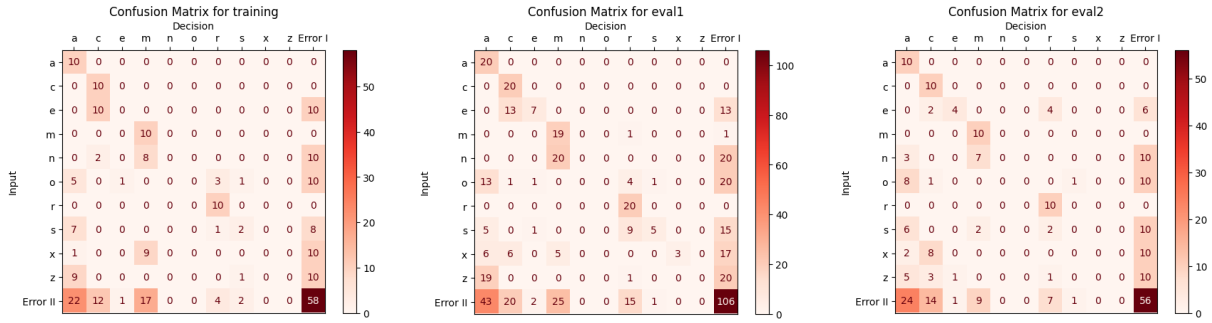
Fig. 13 J(w) vs. epoch for lambdas (a) 0.5 (b) 1.0 (c) 2.5 (d) 3.0



(a) (b) (c) (d)

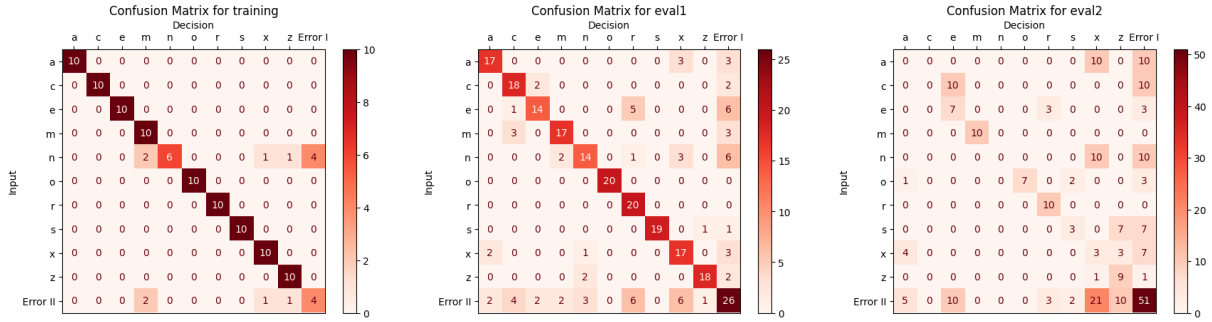
Fig. 14 Percent error vs. epoch for lambdas (a) 0.5 (b) 1.0 (c) 2.5 (d) 3.0

Appendix B



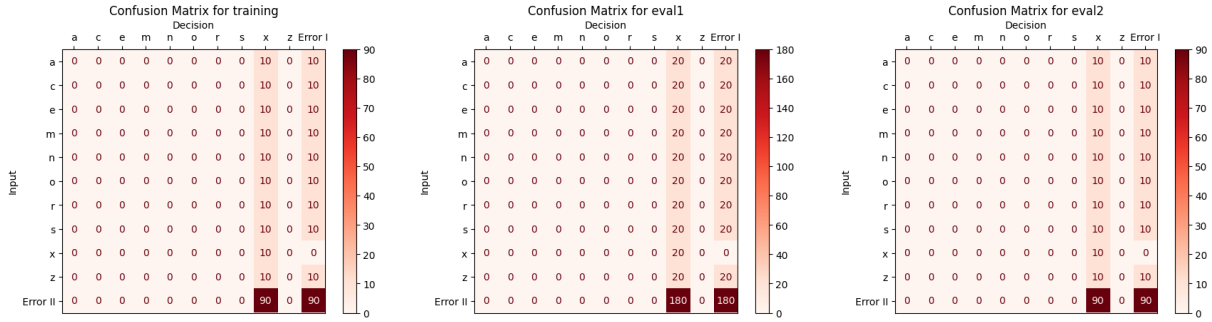
(a) (b) (c)

Fig. 15 Confusion matrices for learning rate = 0.01 (a) training data (b) evaluation set 1 (c) evaluation set 2



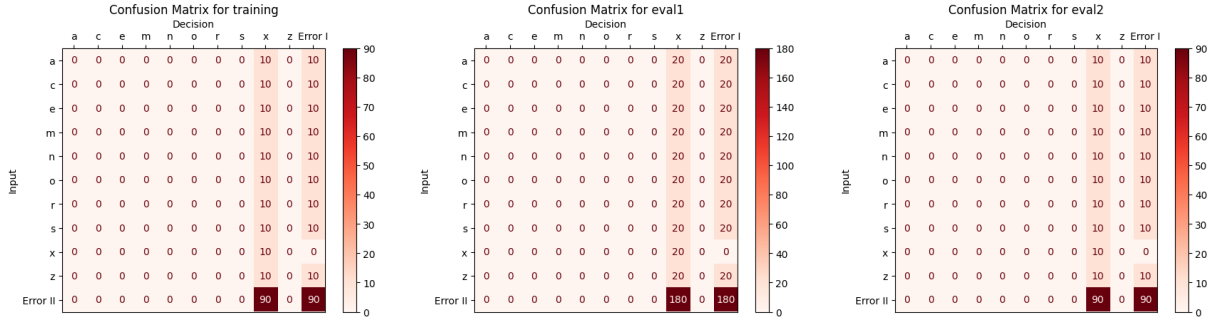
(a) (b) (c)

Fig. 16 Confusion matrices for learning rate = 0.05 (a) training data (b) evaluation set 1 (c) evaluation set 2



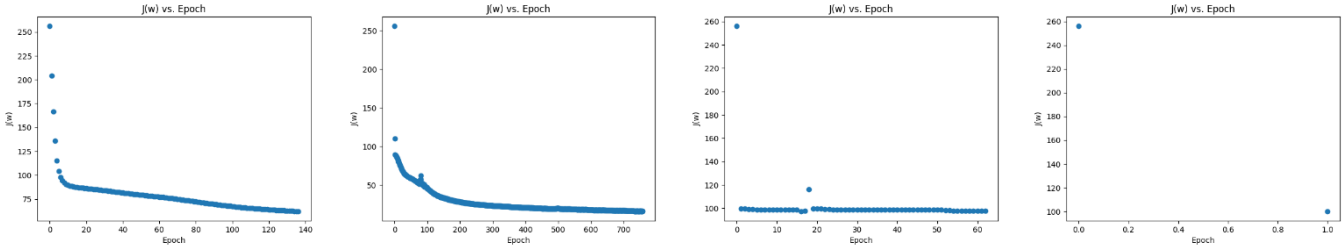
(a) (b) (c)

Fig. 17 Confusion matrices for learning rate = 0.5 (a) training data (b) evaluation set 1 (c) evaluation set 2



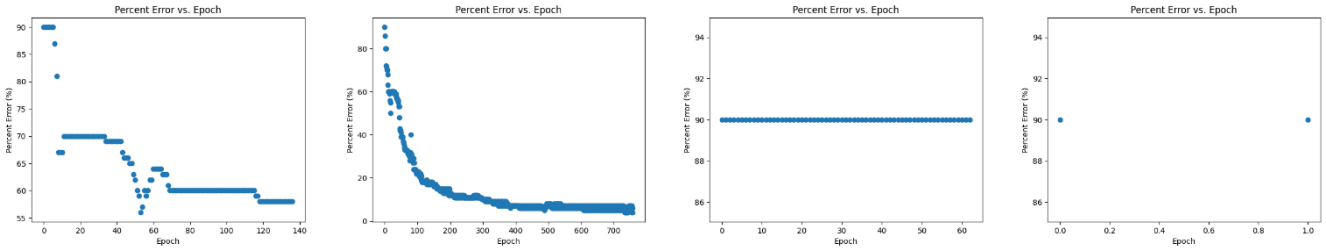
(a) (b) (c)

Fig. 18 Confusion matrices for learning rate = 1.0 (a) training data (b) evaluation set 1 (c) evaluation set 2



(a) (b) (c) (d)

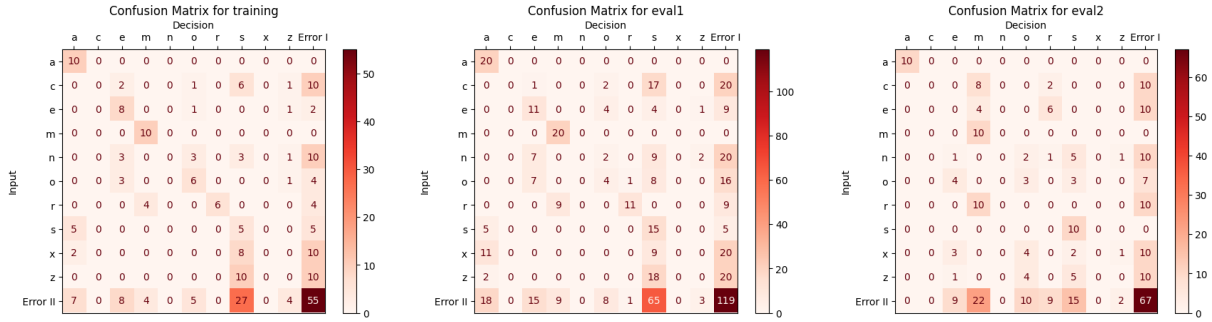
Fig. 19 J(w) vs. epoch for learning rates (a) 0.01 (b) 0.05 (c) 0.5 (d) 1.0



(a) (b) (c) (d)

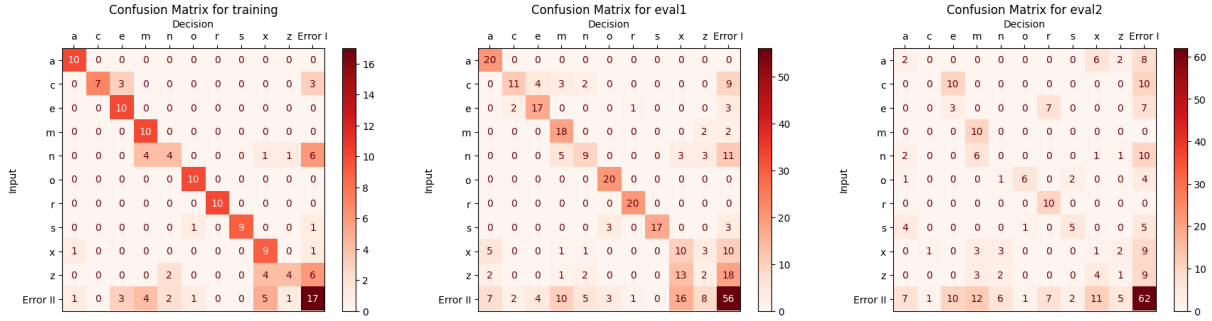
Fig. 20 Percent error vs. epoch for learning rates (a) 0.01 (b) 0.05 (c) 0.5 (d) 1.0

Appendix C



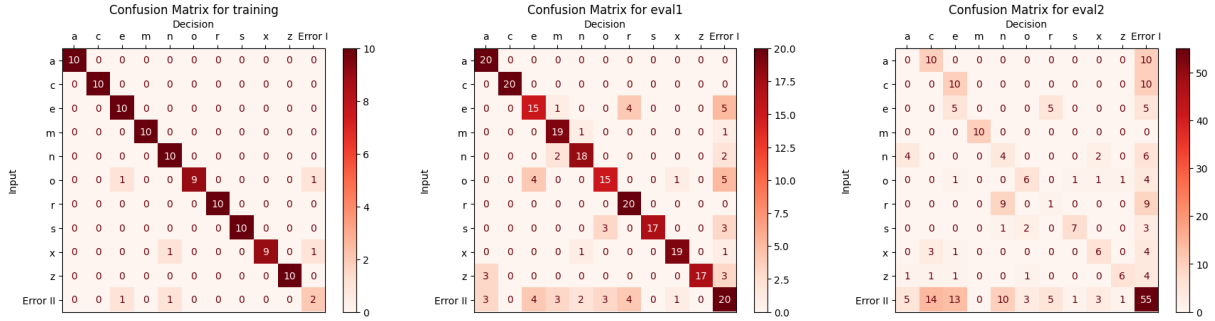
(a) (b) (c)

Fig. 21 Confusion matrices for 2 hidden nodes (a) training data (b) evaluation set 1 (c) evaluation set 2



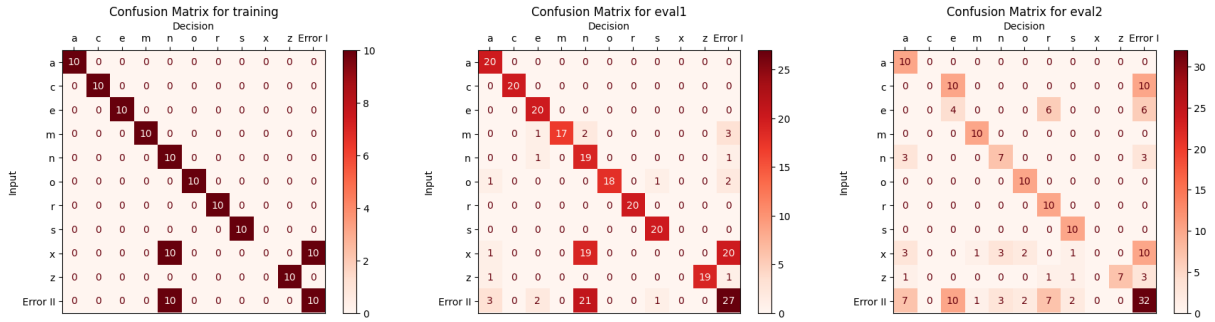
(a) (b) (c)

Fig. 22 Confusion matrices for 3 hidden nodes (a) training data (b) evaluation set 1 (c) evaluation set 2



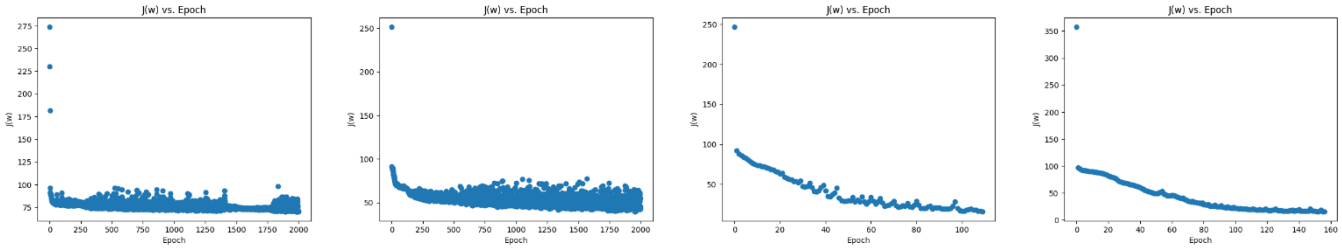
(a) (b) (c)

Fig. 23 Confusion matrices for 4 hidden nodes (a) training data (b) evaluation set 1 (c) evaluation set 2



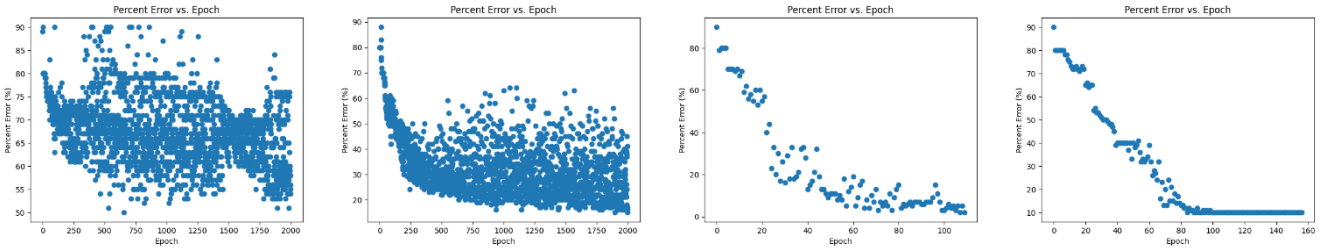
(a) (b) (c)

Fig. 24 Confusion matrices for 6 hidden nodes (a) training data (b) evaluation set 1 (c) evaluation set 2



(a) (b) (c) (d)

Fig. 25 J(w) vs. epoch for hidden nodes (a) 2 (b) 3 (c) 5 (d) 6



(a) (b) (c) (d)

Fig. 26 Percent error vs. epoch for hidden nodes (a) 2 (b) 3 (c) 5 (d) 6