18.05.2023
Berk Demirkan
21802124

EEE443/543 Neural Networks – Mini Project

Question 1

Part A: Pre-processing

After converting the images to grayscale and normalizing, it is apparent that a lot more of the information is available after processing. Even though some of the color images seem to be of a single color, after processing, more information can be extracted and used while training the autoencoder neural network. Both the RGB and grayscale images can be seen below.

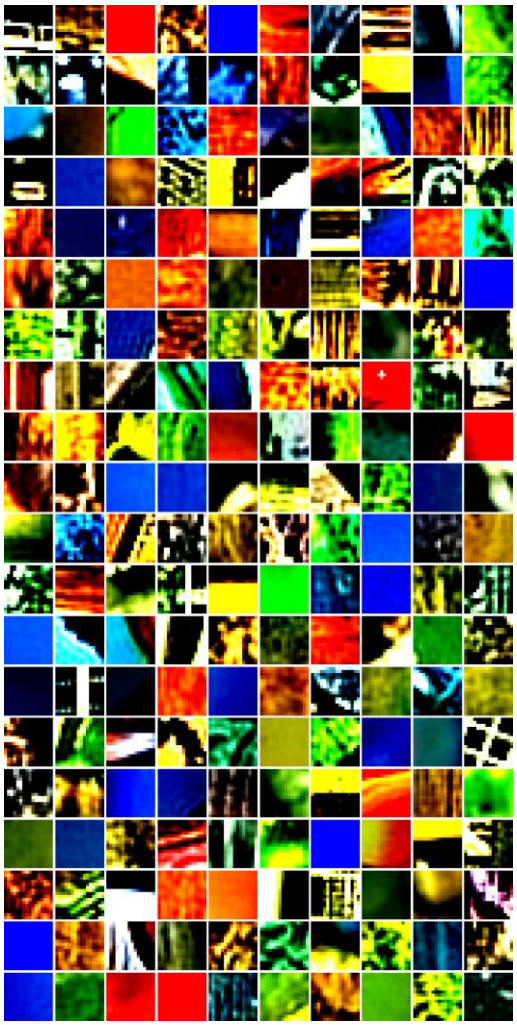


Figure 1: 200 Randomly chosen RGB images

Figure 2: 200 Randomly chosen grayscale images

Part B: Neural Network

To train the neural network, we are given the following cost function:

$$J_{ae} = \underbrace{\frac{1}{2N}\sum_{i=1}^{N}\|d(m) - o(m)\|^2}_{J_1} + \underbrace{\frac{\lambda}{2}\left[\sum_{b=1}^{L_{hid}}\sum_{a=1}^{L_{in}}\left(W_{a,b}^{(1)}\right)^2 + \sum_{c=1}^{L_{out}}\sum_{b=1}^{L_{hid}}\left(W_{b,c}^{(1)}\right)^2\right]}_{J_2} + \underbrace{\beta\sum_{b=1}^{L_{hid}} KL(\rho|\hat{\rho}_b)}_{J_3}$$

The first term is a fundamental mean squared error (MSE) formula that forms the cost of the system. It is important to note that the desired output of the system is the input. The second term is the Tykhonov regularization formula which punishes large weight values and prevents overfitting the model regulated by the λ. The last term is the KL divergence function applied to the average of the activations of the hidden layer, which enforces that only some portion of the weight is active at a time, regulated by β.

The steps to calculate the backpropagation gradients are as follows.

$$\frac{\partial J_{ae}}{\partial A_2} = -\frac{1}{N}(d(m) - o(m))$$
$$\frac{\partial J_{ae}}{\partial Z_2} = \frac{\partial J_{ae}}{\partial A_2} * \sigma(A_2)$$
$$\frac{\partial J_{ae}}{\partial A_1} = W_2^T \cdot A_2$$
$$\frac{\partial J_{ae}}{\partial Z_1} = \frac{\partial J_{ae}}{\partial A_1} * \sigma(A_1)$$

$$\frac{\partial J_{ae}}{\partial W_1} = \frac{\partial J_{ae}}{\partial Z_1} \cdot d(m) + \lambda * W_1$$
$$\frac{\partial J_{ae}}{\partial b_1} = \sum \frac{\partial J_{ae}}{\partial Z_1}$$

$$\frac{\partial J_{ae}}{\partial W_2} = \frac{\partial J_{ae}}{\partial Z_2} \cdot A_1 + \lambda * W_1$$
$$\frac{\partial J_{ae}}{\partial b_2} = \sum \frac{\partial J_{ae}}{\partial Z_2}$$

A simple mini-batch gradient descent algorithm was chosen as the solver. The mini-batch functionality helps the model converge faster, requiring at most five hundred iterations. After trial and error, the determined parameters are as follows.

$\lambda = 5 * 10^{-4}$
$\beta = 0.05$
$\rho = 0.02$
$\eta = 0.01$ (Learning Rate)
$\alpha = 0.95$ (Momentum Rate)
Batch Size = 64
Epochs = 500

Part C: Hidden Layer Weights

This part asks us to visualize the hidden layer weights as separate images. As can be seen from Figure 3, these images are representative of specific features from the input dataset. We can also see some repetition in the weights, but others are very different. This can indicate that the number of neurons in the hidden layer is chosen correctly. Furthermore, it can be guessed that many of these weights will be active simultaneously. Thus, it is not very sparse.

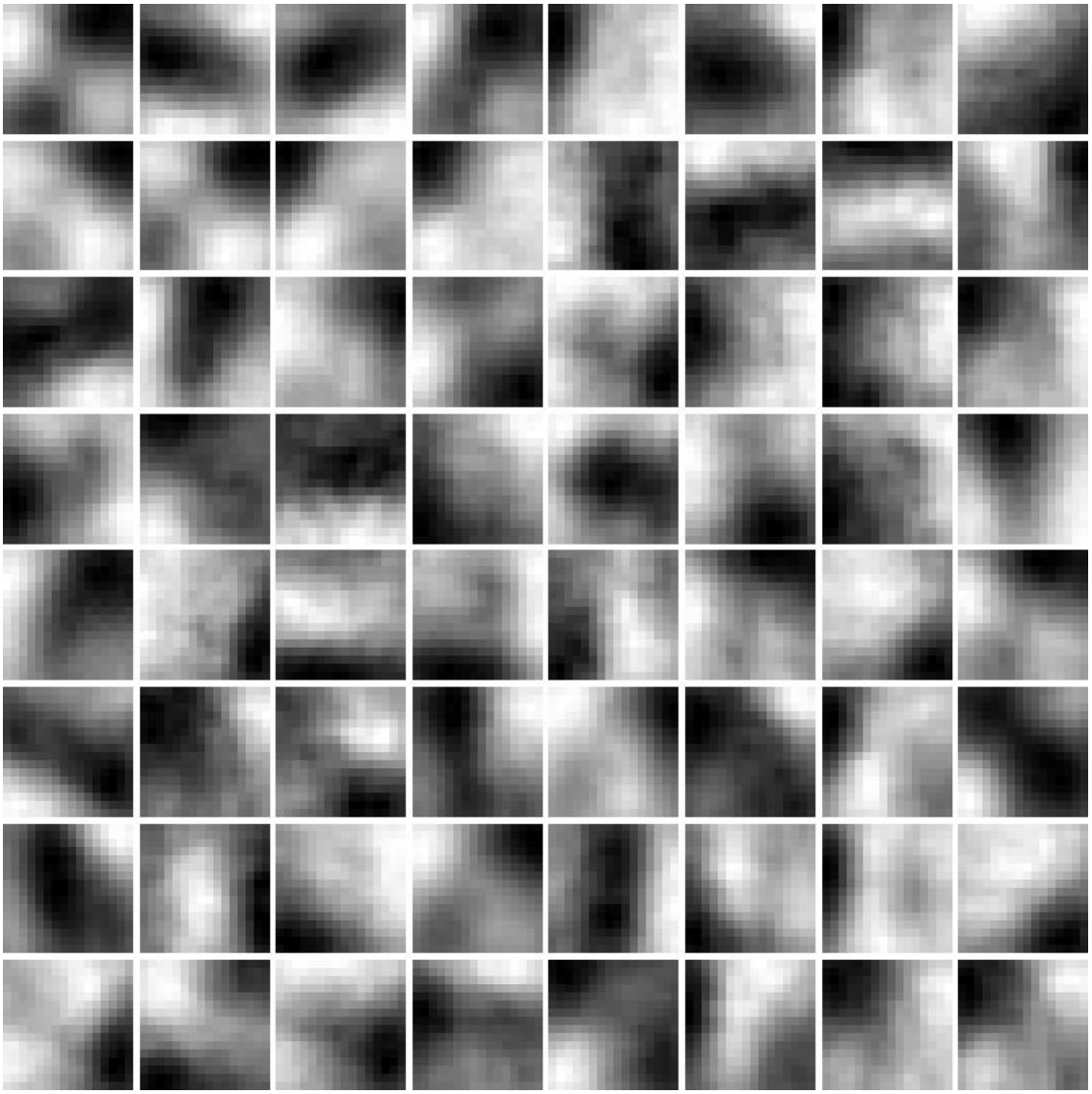


Figure 3: Hidden Layer Weights as Images

Part D: L_{hid} and λ

The network was run for three different values of L_{hid} and λ for a total of 9 different combinations. The other parameters were left as they were. The respective images can be seen below.

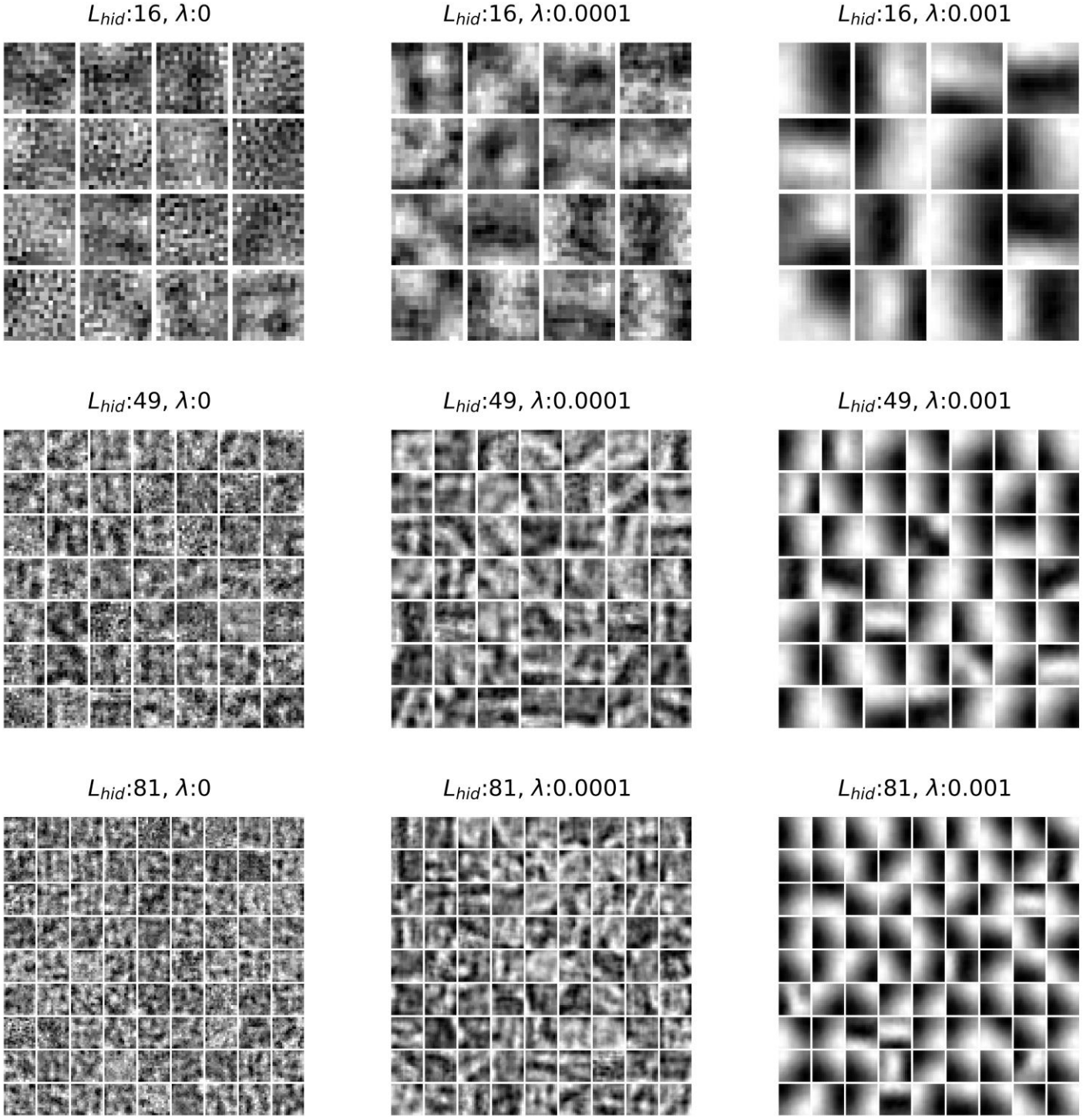


Figure 4: Images From Different Combinations of Hyperparameters

asdasd

Some observations are as follows:

- As the hidden layer size L_{hid} is increased, more features are extracted. This is an important conclusion for the autoencoder.
- Unfortunately, the increase in L_{hid} also creates more chances for the autoencoder to extract redundant features, thus proving no additional benefit. This is especially apparent in the image with $L_{hid} = 81$ and $\lambda = 0.001$. While the high value of λ was used to prevent overfitting, it caused the autoencoder to generate the same weights many times.

- The hyperparameter λ is used to control the L2 regularization term of the cost function. With $\lambda = 0$, overfitting occurs in every case. Even though this helps with sparsity, the extracted features are not helpful. With $\lambda = 0.0001$, some overfitting still happens, but the extracted features are much higher quality. With $\lambda = 0.001$, there is no overfitting, but the network has simplified the weights too much and thus extracted many redundant features.

The hyperparameter L_{hid} should be kept as high as possible, and the sparsity hyperparameter ρ should be adjusted to keep the network sparse with higher L_{hid} values. The regularization hyperparameter λ should be kept around 0.0005 to extract high-quality and sparse features.

Question 2

This question asks us to create a simple NLP-like neural network that can guess the next word when given three words. Using a word embedding matrix, we will convert the word indexes into usable vectors. This matrix will help us create the necessary shape for our neural network.

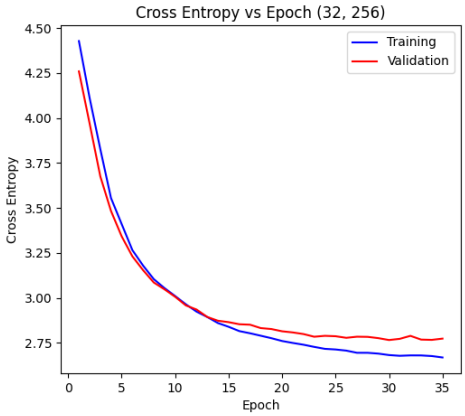
In an ideal world, we would be using the argmax function; however, since the derivative of the argmax function is undefined in a large portion of the numerical space, we will use the softmax function. The output of the softmax function will then be used to get the words of the highest likelihood by sorting them in the appropriate axis.

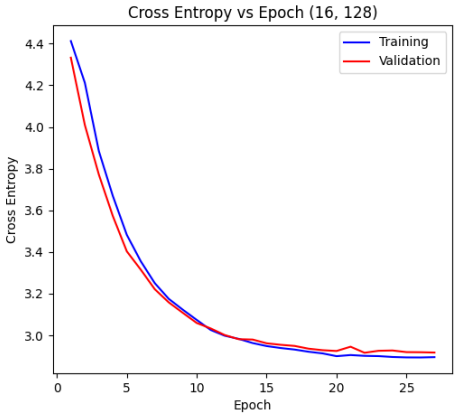
Part A: The Network

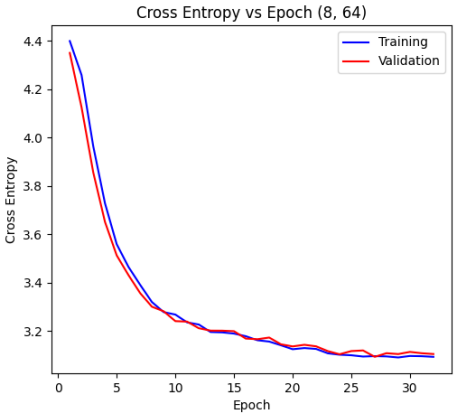
After many iterations of trial and error, the optimal parameters were determined as follows.

$\eta = 0.01$ (Learning Rate)
$\alpha = 0.8$ (Momentum Rate)
Batch Size = 200
Maximum Epochs = 50

On average, the network with (32, 256) seems to be the one with the lowest error. Such high sizes would normally tend toward overfitting, but with the early stopping implemented, this is mostly prevented. The Cross-Entropy vs Epoch graphs can be seen in the figures below.







Figures 5, 6, 7: Cross Entropy vs Loss

Part B: Predictions

The function was called for all 3 cases of (D, P).

```
's all for ['me', 'the', 'us', 'them', 'you', '.', 'what', 'show', '?', 'him']
, he said ['.', ',', 'today', 'yesterday', 'of', '?', 'last', 'so', ':', 'at']
nt know what ['to', 'it', "'s", 'they', 'he', 'i', 'was', 'that', 'is', 'we']
nt say what ['they', 'you', 'he', 'i', 'we', 'to', 'it', '?', '.', "'s"]
do nt have ['to', 'any', 'much', 'it', 'one', 'a', 'that', 'the', 'time', '.']
```

Figure 8: Predictions (32, 256)

```
you never know ['what', '.', 'when', 'how', 'where', ',', 'it', 'that', 'who', 'more']
is time to ['go', 'be', 'do', 'see', 'know', 'get', 'play', 'think', 'make', 'come']
though he would ['nt', 'not', 'do', 'have', 'be', 'get', 'come', 'see', 'take', 'say']
is what your ['is', '.', 'it', 'children', 'life', 'people', ',', '?', 'of', 'own']
end before the ['next', 'world', 'first', 'war', 'day', 'country', 'new', 'game', 'police', 'show']
```

Figure 9: Predictions (16, 128)

```
and they may ['not', 'do', 'be', 'get', 'have', 'go', '.', 'nt', 'say', 'know']
just does , ['he', 'she', 'i', 'it', 'but', 'too', 'and', 'you', 'not', 'no']
this is what ['we', 'i', 'you', 'they', 'it', 'he', 'this', 'the', 'people', "'s"]
will they be ['?', '.', ',', 'back', 'the', 'out', 'here', 'there', 'on', 'more']
want her to ['do', 'go', 'be', 'work', 'see', 'school', 'the', 'it', 'get', '.']
```

Figure 10: Predictions (8, 64)

In the figure above, 10 predictions for each of the five samples are obtained. Although the predictions of any of these networks are sometimes incorrect, they are mostly sensible.

Question 3

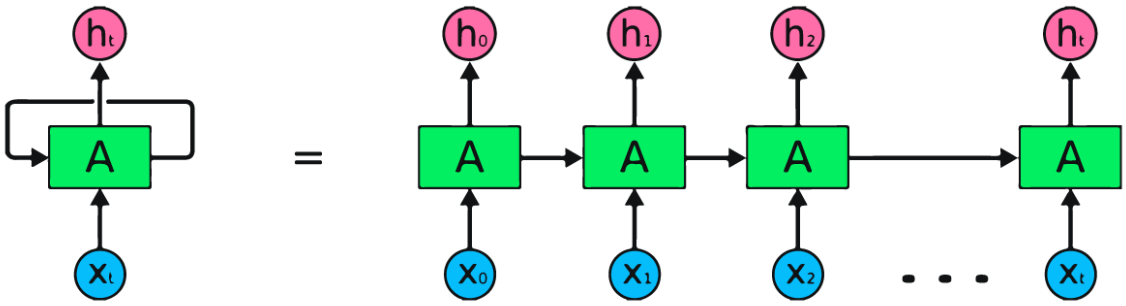


Figure 11: Unrolled Recurrent Neural Network

In this question, we are tasked with creating three types of recurrent neural networks. A Recurrent Neural Network is a neural network that works with time series data. In the requested neural network example, one iteration's recurrent layer output is added to the input of the recurrent layer for the next iteration. Due to the iterative nature of such networks, they can take large amounts of time and computing power to train.

Part A: Simple Recurrent Layer

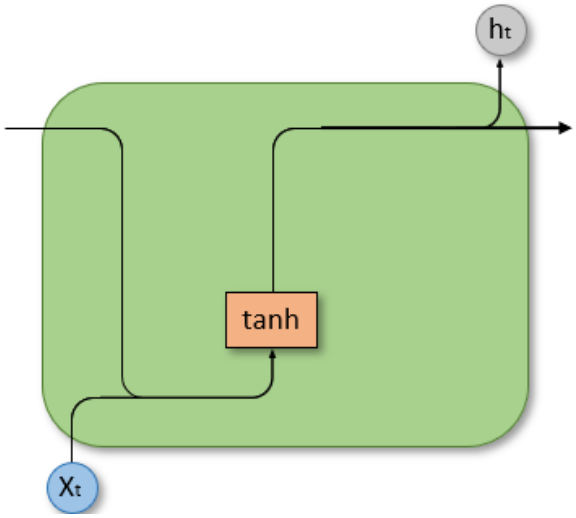


Figure 12: Simple Recurrent Unit

In the figure above, the horizontal line that comes from the right and goes toward the left is the memory of the network. As can be seen, this unit is straightforward and only has a single activation function and a single weight matrix. This is because memory and the actual input are regarded as outputs of different neurons and are processed together.

On such a recurrent network, as the time series is processed, the gradient is multiplied by itself several times, which causes weight values larger than 1 to "explode" towards infinity and values smaller than 1 "vanish" towards zero. This phenomenon is called the vanishing/exploding gradients problem. For example, the given dataset has 150 time samples, and as such, a weight value of 0.9 becomes $0.9^{150} = 1.39 \cdot 10^{-7}$ by the end of the time series. This causes the network to be volatile and unpredictable.

The MLP section of the neural network has one layer with a size of 83 and another layer with a size of 51. These layer sizes were chosen to map linearly between the recurrent layer size of 128 and the output layer size of 6.

The final parameters are as follows:
$\eta = 0.0001$ (Learning Rate)
$\alpha = 0.95$ (Momentum Rate)
Batch Size = 50
Maximum Epochs = 50

The final "Cross Entropy vs. Epoch "graph for the simple RNN can be seen below.

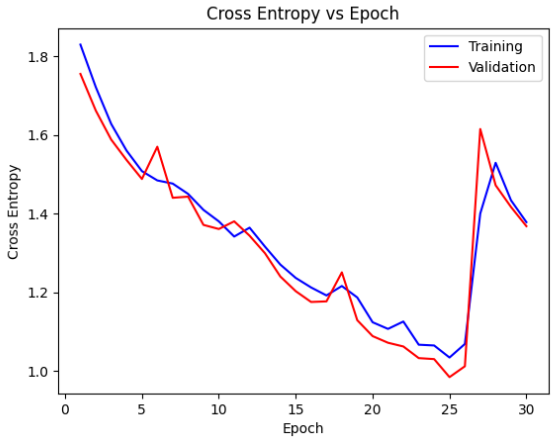


Figure 13: Cross Entropy vs. Epoch for RNN

The confusion matrices for both the training and the test sets are given below.

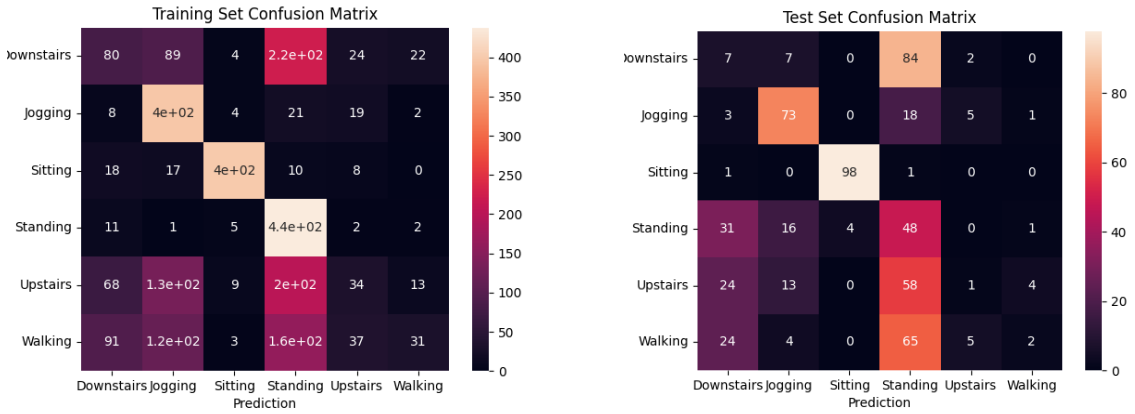


Figure 14, 15: Confusion Matrices for Simple RNN + 2 Layer MLP

As can be seen from the cost vs. epoch graph, the system is unstable. In addition, while the confusion matrices have some resemblance toward correctness, it is far from satisfactory. More epochs and a lower learning rate help; however, with 15s per epoch, waiting on thousands of iterations would be unfeasible.

$$Accuracy_{Train} = 51.04\%$$
$$Accuracy_{Test} = 38.17\%$$

Part B: Long-Short-Term Memory

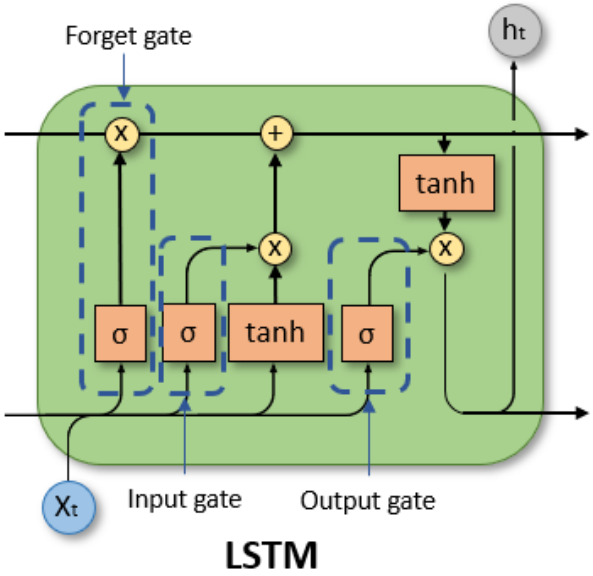


Figure 16: Long-Short-Term Memory Recurrent Unit

In the figure above, a depiction of a single LSTM unit is given. Such LSTM units are created to solve or mitigate the issue of vanishing/exploding gradients. First, the forget gate determines how much long-term memory will be retained (or forgotten). Next, the input gate, with the candidate, determines the effect of the new input on long-term memory. Finally, the output gate determines how much of the newly calculated long-term memory will be redirected as the layer's output and the short-term memory for the next iteration.

Due to the number of gates, the network complexity is increased three times. This causes the training time to be much longer than the simple recurrent neural network.

The MLP layer sizes are kept as (83, 51) to be as consistent as possible.

The final parameters are as follows:
$\eta = 0.0001$ (Learning Rate)
$\alpha = 0.95$ (Momentum Rate)
Batch Size = 200
Maximum Epochs = 50

The "Cross-Entropy vs. Epoch" graph for the LSTM can be seen in the figure below.

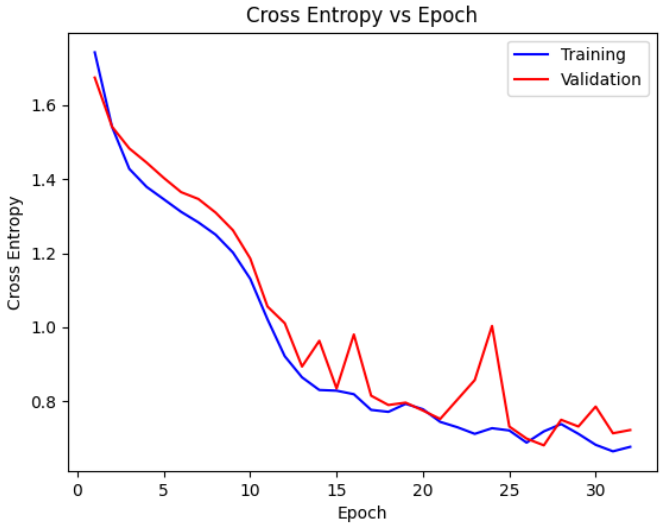


Figure 17: Cross Entropy vs Epoch graph for LSTM

The confusion matrices are as follows:

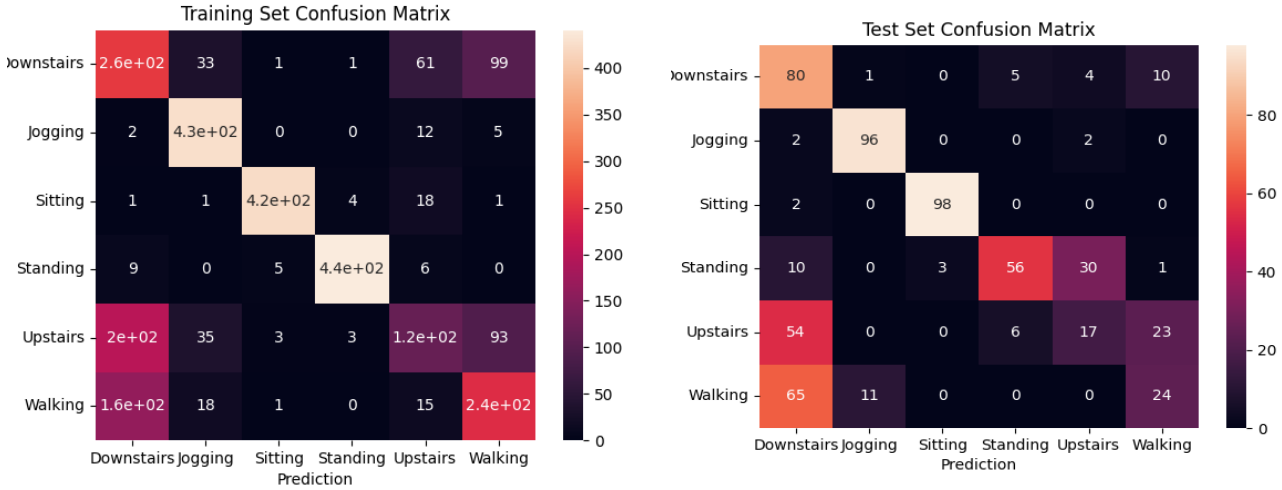


Figure 18, 19: Confusion Matrices for LSTM RNN + 2 Layer MLP

As can be seen, the LSTM performs much better than the simple RNN. The network is much more stable while training. Although confusion matrices are imperfect, they look much better than the simple RNN case. At 55s per epoch, they were a nightmare to train, however.

$$Accuracy_{Train} = 71.37$$
$$Accuracy_{Test} = 57.33$$

Part C: Gated Recurrent Units

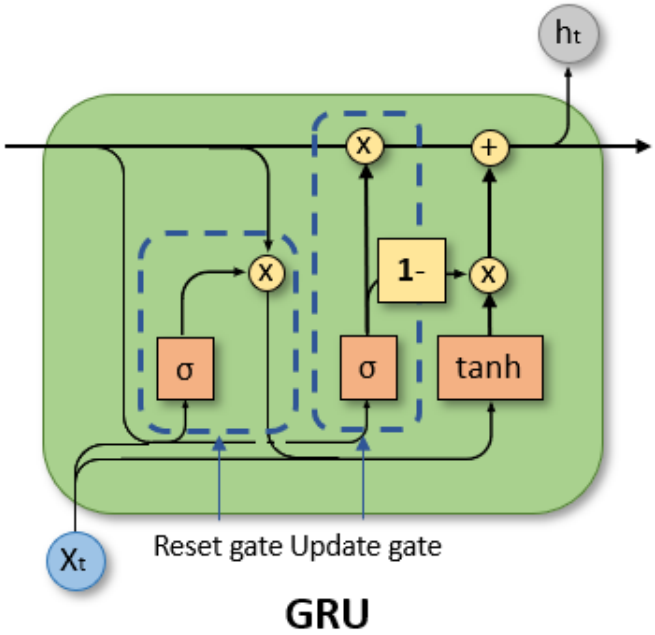


Figure 20: Gated Recurrent Unit

The figure above depicts a single gated recurrent unit (GRU). GRU is the newest recurrent unit architecture in recurrent neural networks. This new recurrent unit was developed to create a simpler recurrent unit than the LSTM recurrent unit. The reset gate is similar to the forget gate of the LSTM and is used to decide how much past information to forget. The update gate is the analog of the forget and input gate and controls how much information to forget and how much to add.

The GRU has only two gates, decreasing the time and resources required to train the network compared to the LSTM's three gates.

The MLP layer sizes are again kept as (83, 51).

The final parameters have not changed and are as follows:
$\eta = 0.0001$ (Learning Rate)
$\alpha = 0.95$ (Momentum Rate)
Batch Size = 200
Maximum Epochs = 50

The "Cross-Entropy vs. Epoch" graph for the GRU can be seen in the figure below.

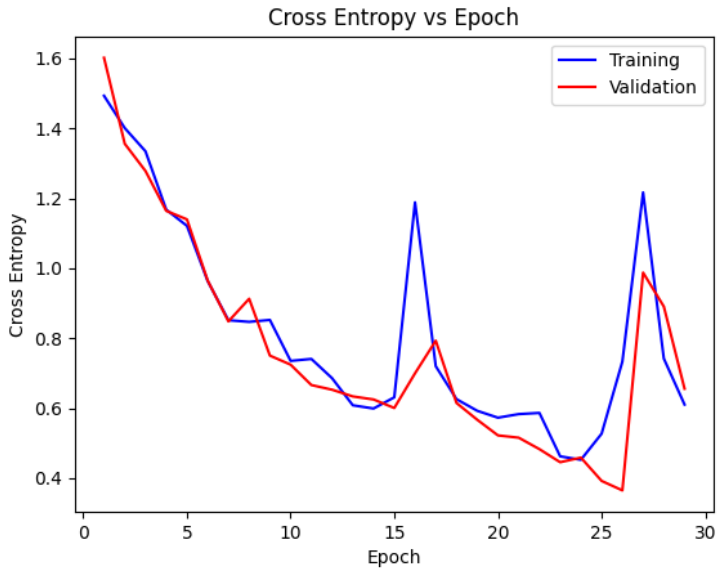


Figure 21: Cross Entropy vs Epoch graph for GRU

The confusion matrices are as follows:

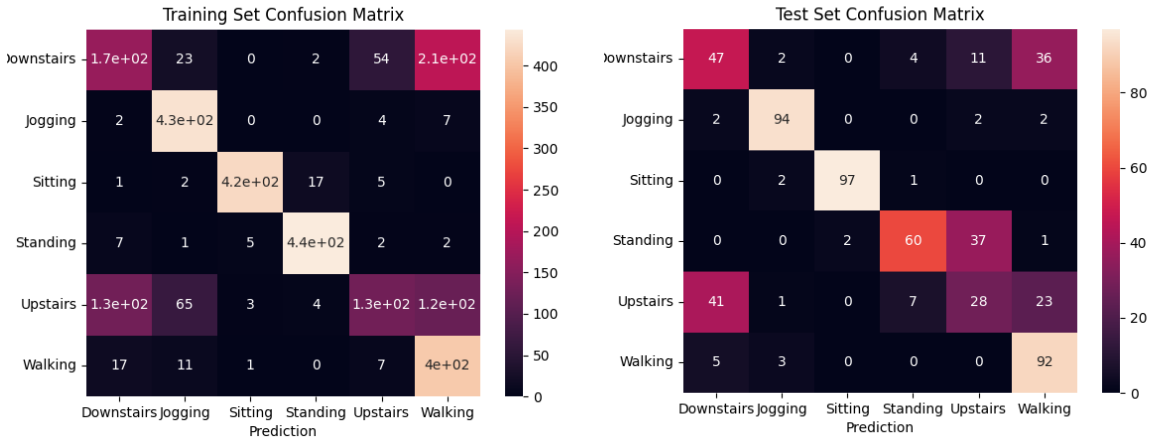


Figure 18, 19: Confusion Matrices for GRU RNN + 2 Layer MLP

GRU performed better than LSTM as well. Moreover, at 30s per epoch, it is the most helpful network yet.

$$Accuracy_{Train} = 74.15$$
$$Accuracy_{Test} = 69.67$$