WiDS 4.0: AI-Phabet

Visharad Srivastava

16 February 2025

Contents

1 Acknowledgement

I would like to thank Aryan Kayande, my mentor, for giving me the oppurtunity to work on this project.

2 Introduction

The aim of this project was to recognize digits (and later letters) from an image, more specifically of 28x28 resolution. It was 4 weeks long, with an optional week 5 that I chose not to pursue. The code necessary was developed in Python, using Google Colab, and trained from the MNIST and EMNIST databases.

3 Learning Phase: Weeks 1-3

3.1 Week 1: Libraries

The first week consisted of an introduction to some libraries later used in the project: Matplotlib, NumPy, Pandas and TensorFlow (later used only for importing the MNIST database).
The resources I used for these included the official documentations of Matplotlib and NumPy.

3.1.1 NumPy

Topics learnt from NumPy consisted of basics such as creation of basic arrays using methods such as full, linspace and zeros, reshaping an array, matrix multiplication and the Hadamard product of two matrices, which is the elementwise product of two matrices.

3.1.2 Matplotlib

The main aim was to be able to visualise data, thus only the matplotlib.pyplot library was used. The key concepts learned were scatter plots (later used for weight visualisation and representing the 28x28 images) and bar graphs (comparing values of the various outputs).

3.1.3 Pandas

From this library, I learned how to create dataframes from CSV files and extract data from them. This was used to extract data from the EMNIST database for

3.2 Week 2: Basics of ML Algorithms

3.2.1 Linear Regression

This part consisted of learning two key machine learning algorithms: linear regression and logistic regression. This was done using a Coursera 3-week course hosted by Andrew Ng. Only the videos were accessible to me; so no practical implementations were done during this phase.
The course consisted firstly of linear regression, a model to predict a linear function based on specific plot points (x_i, y_i):

$$y = mx + c$$

Initially, random variables m and c were chosen, which were then input into the following cost function:

$$C = \frac{1}{2n}\sum_{i=1}^{n}(y(x_i) - y_i)^2$$

where $y(x_i) = mx_i + c$. The next step is to implement a method called gradient descent, which works as follows: A learning rate α is pre-decided, after which the values of m and c are changed as follows:

$$(m_{j+1}, c_{j+1}) = (m_j, c_j) - \alpha \nabla C(m_j, c_j)$$

where (m_j, c_j) are the values calculated in the j^{th} iteration of this algorithm. Furthermore,

$$\nabla C = \left(\frac{\partial C}{\partial m}, \frac{\partial C}{\partial n}\right)$$

3.2.2 Binary Classification Problems

The next part was logistic regression, which focuses on classification problems rather than linear fit problems. The chosen function here is generally a sigmoid function rather than a linear one.
The procedure for binary classification problems is as follows. A linear function x is initially calculated; it is then passed as an argument to the sigmoid function

$$y = \sigma(x) = \frac{1}{1 + e^{-x}}$$

After this, a predetermined threshold value is used to determine the final prediction. The cost function here is defined as logarithmic rather than linear:

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i$$

where

$$C_i = \begin{cases} -\log(1-y) & \text{if } y_i = 0, \\ -\log(y) & \text{if } y_i = 1 \end{cases}$$

where y is the predicted outcome and y_i is the actual value.

3.3 Week 3: Feedforward Neural Networks

In this week, I learned about the final model that I was going to implement. The following is the model I used. The learning consisted, for me, mainly of two resources: 3b1b's YouTube playlist and (especially) Chapters 1 and 2 of Michael Nielsen's online book on neural networks and deep learning.
This is a big step-up from the previous week's material; the playlist was assigned to transition smoothly between weeks 2 and 3. Here, there are multiple inputs x for a single output y. Each x is characterised by its weight and each y by its bias. Also, there are multiple layers in the model. Each output y of the previous layer acts as another input x for the next layer.
Learning in this week consisted of

1. the model on which this kind of network is based,
2. calculation of the cost function and
3. a technique to compute gradient descent called backpropagation, which involves the use of linear algebra to compute the required cost function gradient ∇C, and applying it to the large number of weights and biases.

4 Implementation Phase: Week 4

The final week consisted of two parts; the first to recognize digits and the next to recognize letters. Since both programs have the same structure, both are described together.

4.1 Loading of Data

Digits This was done from the tensorflow.keras library. The initial data consisted of 70,000 digits out of which 54,000 were used for training, 6,000 for validation and 10,000 for testing.

Letters A ZIP file was downloaded from the given GitHub repository, uploaded from Drive and extracted to program contents. Being a CSV file, it was then read into a Pandas DataFrame and copied into NumPy arrays. There are 372,450 alphabets out of which 300,000 were used for training, 10,000 for validation and 62,450 for testing.

4.2 Visualisation

Each dataset consisted of 784 greyscale pixel values (rearranged as 28x28 NumPy arrays) and an output. Visualisation was done using matplotlib.pyplot.imshow on a random number/letter from the training set.

4.3 Model

The model consisted of 4 layers (labelled 0-3).
Following are the variables used in the program.

i Size of the input.

h_1 Size of the first hidden layer.

h_2 Size of the second hidden layer.

o Size of the output layer.

l Learning rate.

e Number of epochs; the usage of this variable is different in both programs.

Following are key matrices used in the program. The names may differ from the actual program, but I have tried to keep them consistent.

Weights The weights and biases were characterized by matrices $W^1 \in \mathbb{R}^{h_1 \times i}, W^2 \in \mathbb{R}^{h_2 \times h_1}, W^3 \in \mathbb{R}^{o \times h_2}$ for weights, where each element w^i_{jk} represented the weight of the k^{th} element of the $(i-1)^{th}$ layer for the j^{th} element of the i^{th} layer. The elements of W^i were initialized using the normal distribution method from NumPy.

Biases There were also bias vectors defined as $b^1 \in \mathbb{R}^{h_1,1}, b^2 \in \mathbb{R}^{h_2,1}, b^3 \in \mathbb{R}^{o,1}$. They are initialised as zero vectors.

Input I_{inp} is the input matrix of size $i \times 1$ and the sigmoid function applied to the matrix is the sigmoid of the individual elements.

The curve used to normalize the linear output for each neuron to $[0, 1]$ was the sigmoid curve.

4.3.1 Epoch Definition

I have used the epoch variable e differently for both programs.

Digits 1 epoch corresponds to training the model over the entire training set, i.e. 54,000 digits. Thus, training over the entire set would take 1 epoch.

Letters 1 epoch corresponds to training the model over 10,000 of the 300,000 letters; i.e. training over the full set would take 30 epochs.

4.4 Loop

1. Forward Propagation: The prediction matrix y_p was calculated as follows:

$$a_1 = \sigma(W^1 I_{inp} + b^1),$$
$$a_2 = \sigma(W^2 a_1 + b^2),$$
$$a_3 = y_p = \sigma(W^3 a_2 + b^3),$$

   Additionally, z_i is defined as $z_i = \sigma^{-1}(a_i)$ for $i \in \{1, 2, 3\}$.

2. Cost Function: This was defined as

$$C = \frac{1}{2} \sum_{m=1}^{10} ((y_p)_m - (y_a)_m)^2$$

   where $(y_p)_m$ is the m^{th} element of y_p and $(y_a)_m$ is the m^{th} element of the actual output matrix.

3. Backpropagation: The following was done:

$$\Delta b^3 = (y_p - y_a) \otimes \sigma'(z_3),$$
$$\Delta b^2 = ({W^3}^T \Delta b_3) \otimes \sigma'(z_2),$$
$$\Delta b^1 = ({W^2}^T \Delta b_2) \otimes \sigma'(z_1),$$
$$\Delta W^3 = \Delta b_3 a_2^T,$$
$$\Delta W^2 = \Delta b_2 a_1^T,$$
$$\Delta W^1 = \Delta b_1 I_{inp}^T,$$

   where $\otimes$ represents the Hadamard product. Then, $W_i' = W_i - \alpha \Delta W_i$ and $b_i' = b_i - \alpha \Delta b_i$ were performed.

4. Validation: To display the results (not used for analysing), a separate validation set (of 6,000 images) was used.

4.5 Testing

Forward propagation was applied on a random digit/letter from the testing sets, and the values of neurons were presented as a bar graph (matplotlib.pyplot.bar) alongside the visualised character.

5 Accuracy and Cost Analysis

The key variables here are the learning rate l and the number of epochs e. Other variables are h_1 and h_2. So, this section involves analysis of accuracy A and cost C as $A(l, e)$ and $C(l, e)$.
Here are the definitions of accuracy and (average) cost.

Accuracy I have defined accuracy as the percentage of datasets of the validation set being 'valid'. If an image was 'valid', it meant that the model guessed the correct answer with the corresponding neuron value ≥ 0.99.

Cost The cost is that of the last digit/letter of the *training* set of each epoch.

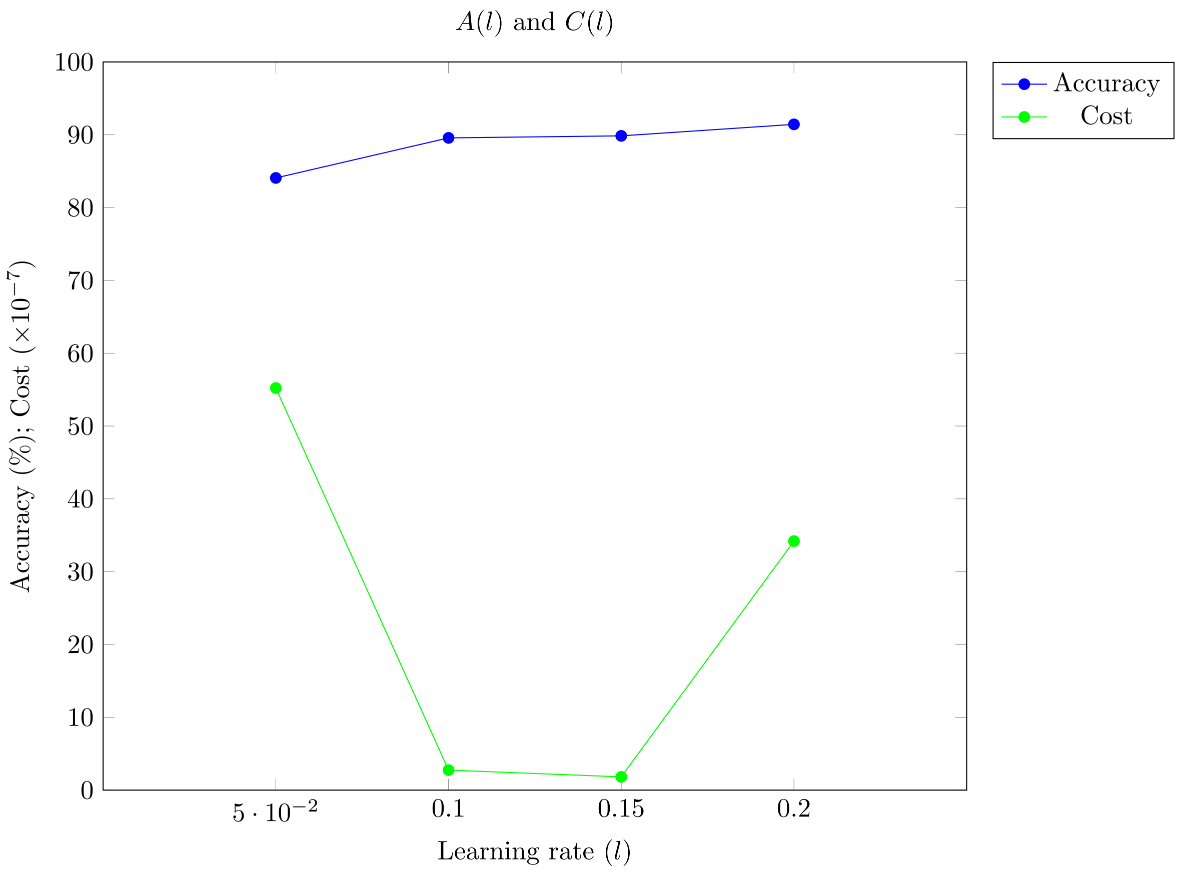
Here, I will be comparing changes in learning rate to changes in cost and accuracy. The other three variables shall remain constant during this comparison.

5.1 Part 1: Digit Recognition

The final values of the four variables were:

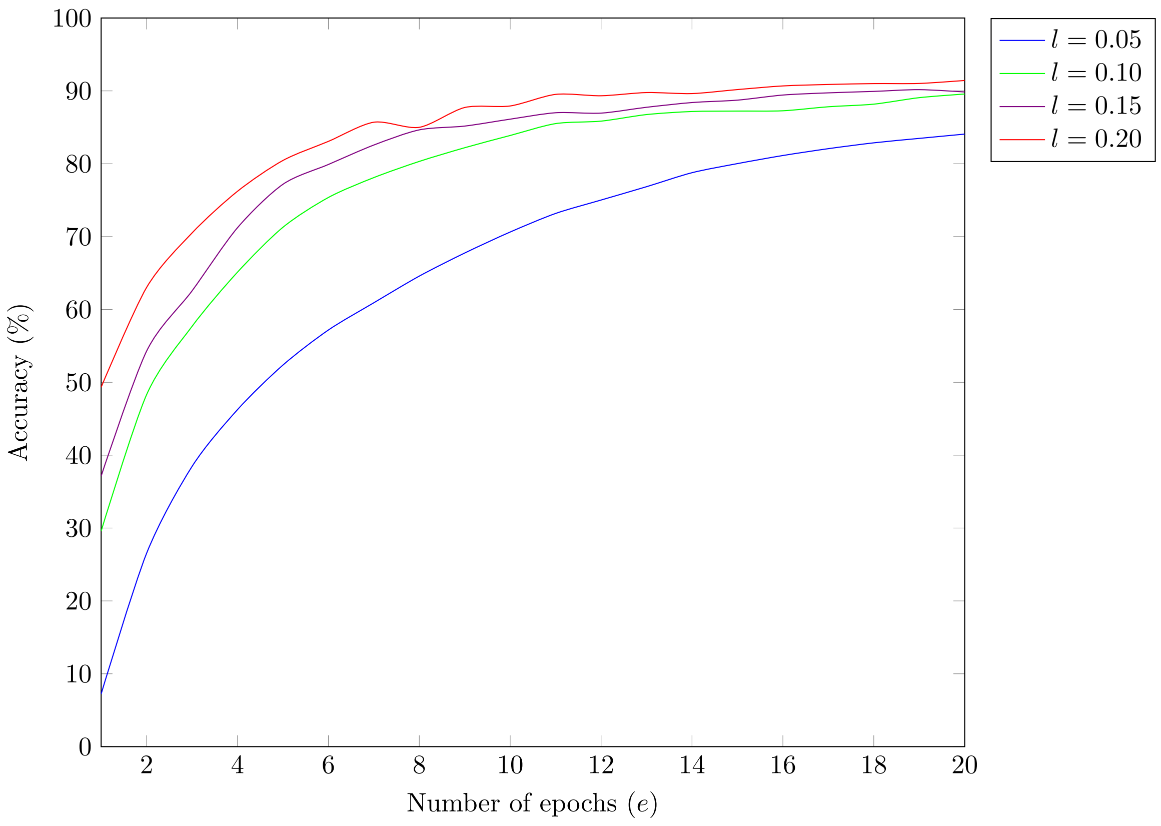
$$h_1 = 196, h_2 = 49, l = 0.15, e = 20$$

and the resulting accuracy was 89.85%.

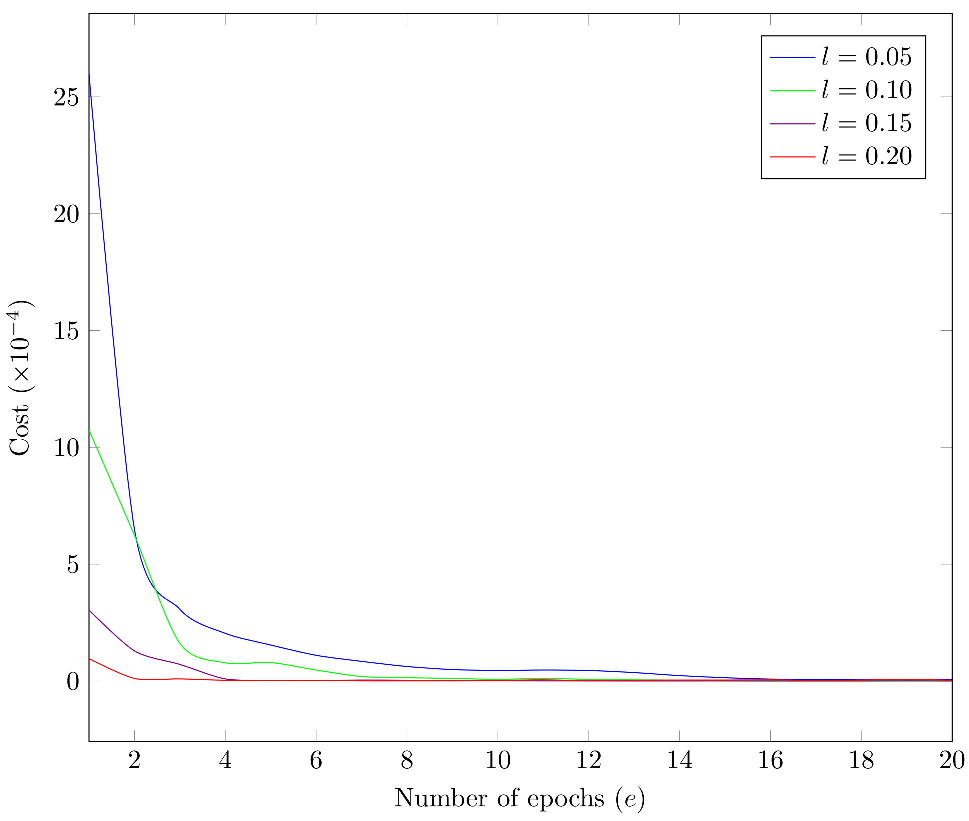


Here's a comparison of how the values of accuracy and cost changed over the epochs.

A over the epochs



C over the epochs

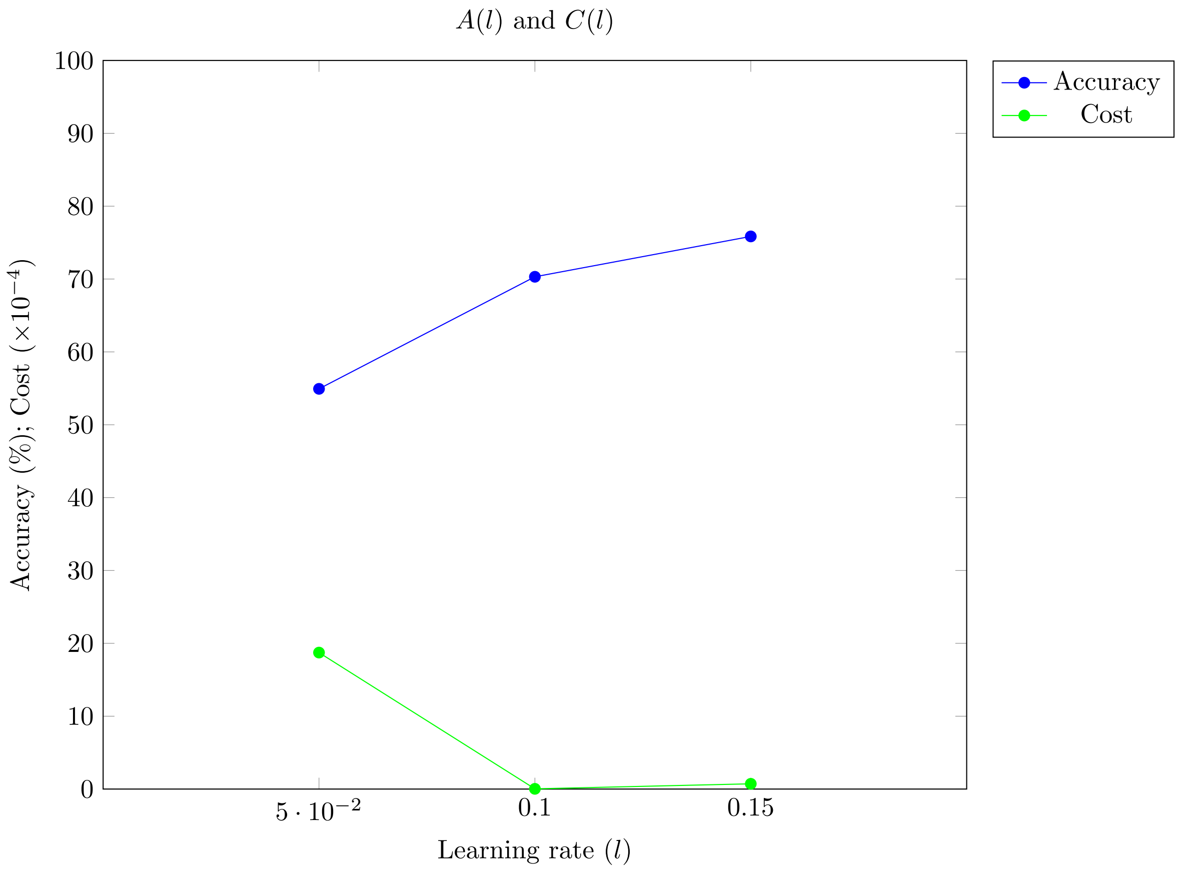


5.2 Part 2: Letter Recognition

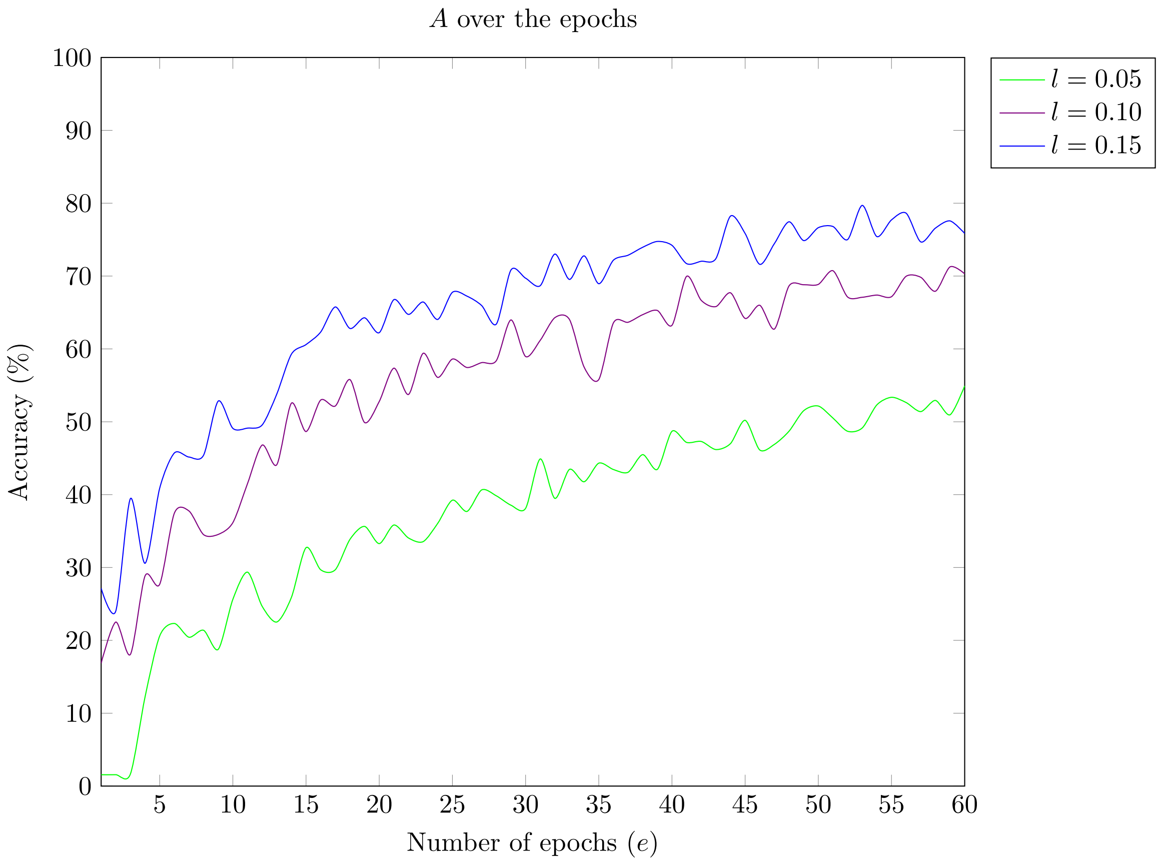
The final values of the four variables were:

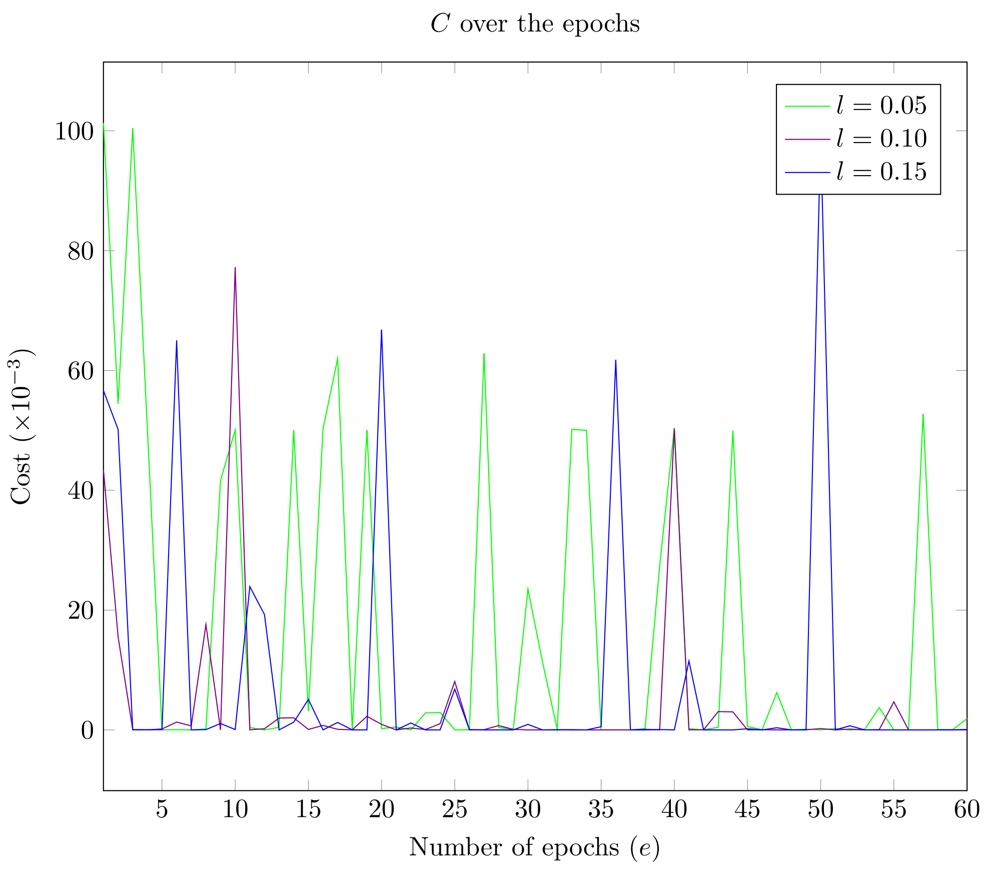
$$h_1 = 420, h_2 = 250, l = 0.1, e = 60$$

and the resulting accuracy was 70.31%.



Here's a comparison of how the values of accuracy and cost changed over the epochs.





A point to note: The reason for all the spikes in the cost function and the wiggly graph for accuracy is probably due to the training sets in the epochs not being the same. It is important to note that $l = 0.10$ had the least number of spikes; hence it was the case of choice.

6 Challenges Faced; Shortcomings

1. When I watched 3b1b's playlist initially, I expected the size of the hidden layers to be of size 16, which was way too less. I initially thought the problem to be that of a higher than normal learning rate; however after increasing the sizes of the hidden layers the program's performance improved drastically (almost a 9% to 90% accuracy jump).
2. The reason the variable *c_temp* is equated to the loss function after every iteration is because of a mistake on my end while writing the program; *c_temp* was initially meant to display the average cost over the training period. It was later decided to repurpose the variable for the average cost over the validation period; so I had to transfer 2 sections of the program from the training to the validation loop and ended up just transferring 1. The variable then finally displayed the cost of the *last* dataset of every epoch.
3. The alphabet case exhibited a much lower accuracy; I have not found the reason for it unfortunately.

7 Experience

This was a good learning experience for me. It involved a lot of waiting, especially for the epochs to run, but the efforts were worth it. It was nice to be able to somewhat demystify the process of machine learning; and also to realize the heavy usage of calculus and linear algebra. This was also something I had mentioned in my SOP; most of the orientations/workshops I had attended (most recently in October, one on OpenCV) in my first semester involved only the demonstration part but lacked in code explanation and what actually goes on behind the scenes. Hence, I am grateful to receive the opportunity to be part of such a project. Overall, it was an enjoyable experience.