Assignment 3

Kumar Abhishek
Student ID: 301371793
ENSC 813: Deep Learning Systems in Engineering

July 9, 2021

Contents

Question 1.

Multi-class classification using softmax regression

The input features are denoted by x, and therefore the activation output of the 0th layer, $a^{[0]}$ is the same as the input. Figure 1 shows a schematic overview of the classifier's components.

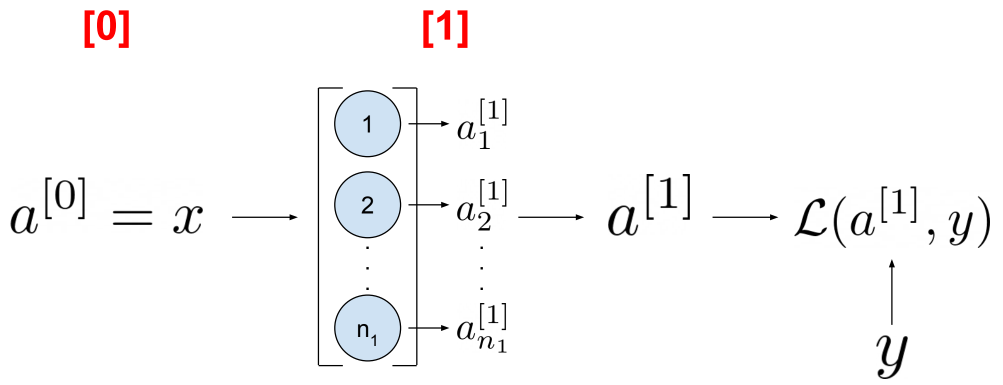
$$a^{[0]} = x. \tag{1}$$

These are passed through the 1st layer, which contains n_1 neurons, each with its own weight and bias. The weights and the biases of the 1st layer are denoted by $W^{[1]}$ and $b^{[1]}$ respectively. Therefore, the computation output of the 1st layer is given by

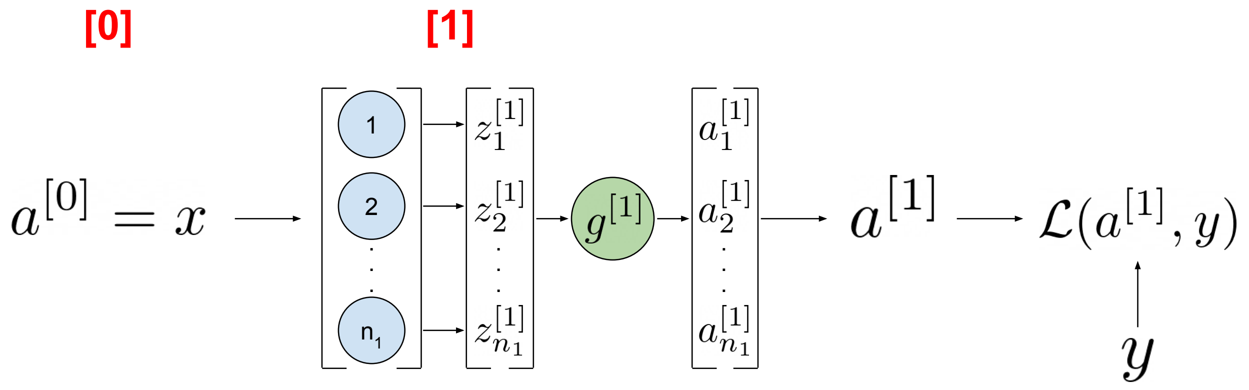
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}. \tag{2}$$

Let $g^{[1]}$ denote the activation of the 1st layer. Then, the output of the 1st layer after the activation step is given by

$$a^{[1]} = g^{[1]}\left(z^{[1]}\right). \tag{3}$$



(a) A schematic of the softmax regression classifier.



(b) Expanded schematic to explicitly show the computation and the activation outputs of the neurons.

Figure 1: A schematic of the softmax regression classifier model. Red indicates the layer indices, blue denotes the neurons, and green indicates the softmax activation.

For softmax regression, the activation of the last layer (1st layer in our case) is the softmax function, so

$$a^{[1]} = \text{softmax}\left(z^{[1]}\right), \quad \text{where} \;\; a_i^{[1]} = \frac{e^{z_i^{[1]}}}{\sum\limits_{k=1}^{n_1} e^{z_k^{[1]}}} \; \forall i \in [1, n_1]. \tag{4}$$

For clarity and notational brevity, we drop the limits of the sum and replace $e^{[\cdot]}$ with $\exp([\cdot])$. Therefore, Equation 4 can be rewritten as

$$a_i^{[1]} = \frac{\exp\left(z_i^{[1]}\right)}{\sum\limits_k \exp\left(z_k^{[1]}\right)}. \tag{5}$$

The multi-class classification loss function is given by

$$\mathcal{L}(a^{[1]}, y) = -\sum_{k=1}^{n_1} y_k \ln a_k^{[1]}, \tag{6}$$

where y_k denotes the k^{th} entry of the one-hot encoded ground truth classification label and $a_k^{[1]}$ denotes the activation output of the k^{th} neuron of the 1^{st} layer.

Backward pass:
In order to update the weights and biases of this multi-class classifier, we need to calculate the gradients of the loss $\mathcal{L}(a^{[1]}, y)$ (Equation 6) w.r.t. these, i.e., $\mathbf{dW^{[1]}}$ and $\mathbf{db^{[1]}}$ respectively, where

$$\begin{aligned} \mathbf{dW^{[1]}} &= \frac{d\mathcal{L}}{dW^{[1]}}, \\ \mathbf{db^{[1]}} &= \frac{d\mathcal{L}}{db^{[1]}}. \end{aligned} \tag{7}$$

Calculating the other derivatives:

$$\mathbf{da^{[1]}} = \frac{d\mathcal{L}}{da^{[1]}} = \begin{bmatrix} \frac{d\mathcal{L}}{da_1^{[1]}} \\ \frac{d\mathcal{L}}{da_2^{[1]}} \\ \vdots \\ \frac{d\mathcal{L}}{da_{n_1}^{[1]}} \end{bmatrix} = \begin{bmatrix} -\frac{y_1}{a_1^{[1]}} \\ -\frac{y_2}{a_2^{[1]}} \\ \vdots \\ -\frac{y_{n_1}}{da_{n_1}^{[1]}} \end{bmatrix} \tag{8}$$

$$\mathbf{dz^{[1]}} = \frac{d\mathcal{L}}{dz^{[1]}} = \frac{d\mathcal{L}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}} \tag{9}$$

Unlike other activations such as the sigmoid, the hyperbolic tangent, or the linear activation, softmax activation is different because the activation output of the k^{th} neuron, $a_k^{[1]}$, depends not only on $z_k^{[1]}$, but on computation output of all the neurons (Equation 5). Therefore, the derivative of the activation output w.r.t. the computation output, i.e., $\frac{da^{[1]}}{dz^{[1]}}$ will depend on outputs from all the neurons.

Calculating $\frac{da^{[1]}}{dz^{[1]}}$:
Since $a^{[1]}$ and $z^{[1]}$ are both vectors, the derivative of $a^{[1]}$ w.r.t. $z^{[1]}$ is the Jacobian matrix, denoted as

$$\frac{da^{[1]}}{dz^{[1]}} = \begin{bmatrix} \frac{da_1^{[1]}}{dz_1^{[1]}} & \frac{da_1^{[1]}}{dz_2^{[1]}} & \cdots & \frac{da_1^{[1]}}{dz_{n_1}^{[1]}} \\ \frac{da_2^{[1]}}{dz_1^{[1]}} & \frac{da_2^{[1]}}{dz_2^{[1]}} & \cdots & \frac{da_2^{[1]}}{dz_{n_1}^{[1]}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{da_{n_1}^{[1]}}{dz_1^{[1]}} & \frac{da_{n_1}^{[1]}}{dz_2^{[1]}} & \cdots & \frac{da_{n_1}^{[1]}}{dz_{n_1}^{[1]}} \end{bmatrix}, \tag{10}$$

whose $(i,j)^{\text{th}}$ entry is $\frac{da_i^{[1]}}{dz_j^{[1]}}$. Now, let us calculate the individual entries of this Jacobian matrix. Clearly, there are 2 cases to evaluate: **case 1:** when $i = j$ and **case 2:** when $i \neq j$.

Case 1: when $i = j$

From Equation 5, differentiating $a_i^{[1]}$ w.r.t. $z_i^{[1]}$,

$$\begin{aligned} \frac{da_i^{[1]}}{dz_i^{[1]}} &= \frac{\exp\left(z_i^{[1]}\right).\sum_k \exp\left(z_k^{[1]}\right) - \exp\left(z_i^{[1]}\right).\exp\left(z_i^{[1]}\right)}{\left(\sum_k \exp\left(z_k^{[1]}\right)\right)^2} \\ &= \frac{\exp\left(z_i^{[1]}\right).\sum_k \exp\left(z_k^{[1]}\right)}{\left(\sum_k \exp\left(z_k^{[1]}\right)\right)^2} - \frac{\exp\left(z_i^{[1]}\right).\exp\left(z_i^{[1]}\right)}{\left(\sum_k \exp\left(z_k^{[1]}\right)\right)^2} \\ &= \frac{\exp\left(z_i^{[1]}\right)}{\sum_k \exp\left(z_k^{[1]}\right)} - \left(\frac{\exp\left(z_i^{[1]}\right)}{\sum_k \exp\left(z_k^{[1]}\right)}\right)^2 \\ &= a_i^{[1]} - \left(a_i^{[1]}\right)^2 \\ &= a_i^{[1]}.\left(1 - a_i^{[1]}\right) \end{aligned}$$

$$\Rightarrow \boxed{\frac{da_i^{[1]}}{dz_i^{[1]}} = a_i^{[1]}.\left(1 - a_i^{[1]}\right)}. \tag{11}$$

Case 2: when $i \neq j$

From Equation 5, differentiating $a_i^{[1]}$ w.r.t. $z_i^{[j]}$,

$$\begin{aligned}
\frac{da_i^{[1]}}{dz_j^{[1]}} &= \frac{0.\sum_k \exp\left(z_k^{[1]}\right) - \exp\left(z_i^{[1]}\right).\exp\left(z_j^{[1]}\right)}{\left(\sum_k \exp\left(z_k^{[1]}\right)\right)^2} \\
&= -\frac{\exp\left(z_i^{[1]}\right).\exp\left(z_j^{[1]}\right)}{\left(\sum_k \exp\left(z_k^{[1]}\right)\right)^2} \\
&= -\left(\frac{\exp\left(z_i^{[1]}\right)}{\sum_k \exp\left(z_k^{[1]}\right)}\right) \cdot \left(\frac{\exp\left(z_j^{[1]}\right)}{\sum_k \exp\left(z_k^{[1]}\right)}\right) \\
&= -a_i^{[1]}.a_j^{[1]}
\end{aligned}$$

$$\Rightarrow \boxed{\frac{da_i^{[1]}}{dz_j^{[1]}} = -a_i^{[1]}.a_j^{[1]}}. \tag{12}$$

Going back to Equation 9, we have

$$\begin{aligned}
\mathbf{dz^{[1]}} &= \frac{d\mathcal{L}}{dz^{[1]}} = \frac{d\mathcal{L}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}} \\
\Rightarrow \mathbf{dz_j^{[1]}} &= \frac{d\mathcal{L}}{dz_j^{[1]}} = \frac{d\mathcal{L}}{da^{[1]}}\frac{da^{[1]}}{dz_j^{[1]}}
\end{aligned} \tag{13}$$

As explained before, $a^{[1]}$ depends on all values of $z_k^{[1]}$, we can rewrite Equation 13 as below and substitute values of various terms as calculated above:

$$
\begin{aligned}
\mathbf{dz_j^{[1]}} = \frac{d\mathcal{L}}{dz_j^{[1]}} &= \sum_k \frac{d\mathcal{L}}{da_k^{[1]}} \frac{da_k^{[1]}}{dz_j^{[1]}} \\
&= \sum_{k \neq j} \frac{d\mathcal{L}}{da_k^{[1]}} \frac{da_k^{[1]}}{dz_j^{[1]}} + \frac{d\mathcal{L}}{da_j^{[1]}} \frac{da_j^{[1]}}{dz_j^{[1]}} \\
&= \sum_{k \neq j} \left(-\frac{y_k}{a_k^{[1]}} \right) \left(-a_k^{[1]} a_j^{[1]} \right) + \left(-\frac{y_j}{a_j^{[1]}} \right) a_j^{[1]} \left(1 - a_j^{[1]} \right) \\
&= \sum_{k \neq j} y_k a_j^{[1]} - y_j + y_j a_j^{[1]} \\
&= \underbrace{\sum_{k \neq j} y_k a_j^{[1]} + y_j a_j^{[1]}}_{\text{join these terms together}} - y_j \\
&= \sum_k y_k a_j^{[1]} - y_j \\
&= a_j^{[1]} \underbrace{\sum_k y_k}_{=1} - y_j \\
&= a_j^{[1]} - y_j
\end{aligned}
$$

$$
\Rightarrow \boxed{\mathbf{dz_j^{[1]}} = a_j^{[1]} - y_j}. \tag{14}
$$

For clarification, we are able to set $\sum_k y_k = 1$ in the penultimate step above since the target labels y_ks are one hot encoded, meaning only one of the k indices is 1 and the rest are 0, thus leading to a sum of 1.

Now, calculating $\mathbf{dz^{[1]}}$, we have:

$$
\mathbf{dz^{[1]}} = \begin{bmatrix} \mathbf{dz_1^{[1]}} \\ \mathbf{dz_2^{[1]}} \\ \vdots \\ \mathbf{dz_{n_1}^{[1]}} \end{bmatrix} = \begin{bmatrix} a_1^{[1]} - y_1 \\ a_2^{[1]} - y_2 \\ \vdots \\ a_{n_1}^{[1]} - y_{n_1} \end{bmatrix} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_{n_1}^{[1]} \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n_1} \end{bmatrix} = a^{[1]} - y \tag{15}
$$

Finally, plugging all these values in the parameter gradient equations (Equation 7), we have

$$\begin{aligned}\mathbf{dW^{[1]}} = \frac{d\mathcal{L}}{dW^{[1]}} &= \underbrace{\frac{d\mathcal{L}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}}}_{\mathbf{dz^{[1]}}}\frac{dz^{[1]}}{dW^{[1]}} \\ &= \mathbf{dz^{[1]}}\underbrace{\frac{dz^{[1]}}{dW^{[1]}}}_{a^{[0]^T}} \\ &= \left(a^{[1]} - y\right) a^{[0]^T}, \quad \text{and}\end{aligned}$$

$$\begin{aligned}\mathbf{db^{[1]}} = \frac{d\mathcal{L}}{db^{[1]}} &= \underbrace{\frac{d\mathcal{L}}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}}}_{\mathbf{dz^{[1]}}}\frac{dz^{[1]}}{db^{[1]}} \\ &= \mathbf{dz^{[1]}}\underbrace{\frac{dz^{[1]}}{db^{[1]}}}_{1} \\ &= a^{[1]} - y.\end{aligned}$$

Therefore, the parameter update equations are:

$$W^{[1]} = W^{[1]} - \alpha\ \mathbf{dW^{[1]}} = W^{[1]} - \alpha\left(\left(a^{[1]} - y\right) a^{[0]^T}\right) \tag{16}$$

$$b^{[1]} = b^{[1]} - \alpha\ \mathbf{db^{[1]}} = b^{[1]} - \alpha\left(a^{[1]} - y\right) \tag{17}$$

Vectorization over a mini-batch of m training samples:
Forward pass:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}; \quad A^{[0]} = \begin{bmatrix} | & | & & | \\ a^{[0](1)} & a^{[0](2)} & \dots & a^{[0](m)} \\ | & | & & | \end{bmatrix} \tag{18}$$

$$Z^{[1]} = W^{[1]}A^{[0]} + b^{[1]}$$

$$\Rightarrow \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \dots & z^{[1](m)} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} W^{[1]}_{11} & \cdots & W^{[1]}_{1n_1} \\ \vdots & \cdots & \vdots \\ W^{[1]}_{n_1 1} & \cdots & W^{[1]}_{n_1 n_1} \end{bmatrix} \begin{bmatrix} | & | & & | \\ a^{[0](1)} & a^{[0](2)} & \dots & a^{[0](m)} \\ | & | & & | \end{bmatrix} + \begin{bmatrix} b^{[1]}_1 \\ \vdots \\ b^{[1]}_{n_1} \end{bmatrix} \tag{19}$$

$$A^{[1]} = g^{[1]}\left(Z^{[1]}\right)$$

$$\Rightarrow \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix} = g^{[1]}\left(\begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \dots & z^{[1](m)} \\ | & | & & | \end{bmatrix}\right)$$
$$= \begin{bmatrix} | & | & & | \\ g^{[1]}\left(z^{[1](1)}\right) & g^{[1]}\left(z^{[1](2)}\right) & \dots & g^{[1]}\left(z^{[1](m)}\right) \\ | & | & & | \end{bmatrix} \tag{20}$$

<u>Backward pass:</u>

$$\mathbf{dW^{[1]}} = \frac{d\mathcal{L}}{dW^{[1]}} = \left(A^{[1]} - Y\right) A^{[0]^T}$$

$$\Rightarrow \mathbf{dW^{[1]}} = \left(\begin{bmatrix} | & & | \\ a^{[1](1)} & \dots & a^{[1](m)} \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ y^{(1)} & \dots & y^{(m)} \\ | & & | \end{bmatrix}\right) \begin{bmatrix} | & & | \\ a^{[0](1)} & \dots & a^{[0](m)} \\ | & & | \end{bmatrix}^T \tag{21}$$

$$\mathbf{db^{[1]}} = \frac{d\mathcal{L}}{db^{[1]}} = A^{[1]} - Y$$

$$\Rightarrow \mathbf{db^{[1]}} = \begin{bmatrix} | & & | \\ a^{[1](1)} & \dots & a^{[1](m)} \\ | & & | \end{bmatrix} - \begin{bmatrix} | & & | \\ y^{(1)} & \dots & y^{(m)} \\ | & & | \end{bmatrix} \tag{22}$$

Question 2.

Analyzing the MNIST dataset

The MNIST dataset contains $70,000$ images, each of 28×28 resolution, of handwritten digits 0 through 9. Therefore, the target labels are $\{0, 1, \cdots, 9\}$. We count the number of images belonging to each class and they are as shown in Table 1.

Partitioning the dataset into training and testing splits

In order to prepare the dataset for multi-class classification, we select 4 classes whose images will be used to train this digit classifier. We choose the digits '0', '1', '3', and '4', because they have distinctive shapes. '0' has a circular shape, '1' has a linear shape, '3' usually has two distinct curves in its shape and '4' usually has 3 distinct lines in its shape.

Class	**Count**
0	6903
1	7877
2	6990
3	7141
4	6824
5	6313
6	6876
7	7293
8	6825
9	6958

Table 1: Class-wise population of the MNIST dataset.

Next, we specify the fraction of the dataset to be held out for testing. We choose to set aside 30% of the entire dataset for testing. However, the user can choose any other fraction by specifying it in the variable `test_frac`.

Finally, we prepare the training and the testing datasets using the following steps:

- Select only those indices of the dataset which correspond to the 4 classes specified above ('0', '1', '3', and '4' in our case).
- Change the target labels for these indices to 0, 1, 2, and 3 for the classes respectively. However, this is optional and can be skipped since in softmax regression, we work with one-hot encoded target labels.
- Use scikit-learn's inbuilt functionality to split the input data and the target labels into training and testing splits. Note that we shuffle the data for randomness and use a seed value for reproducibility. We also employ stratified sampling to ensure that the class population ratios remain the same in training and testing splits.

Training the softmax regression classifier

We use the softmax regression classifier for our multi-class classification task which has been defined in `utils.py` as a custom class named `SoftmaxClassifier`. Apart from the classifier's parameters (weights and biases), the hyperparameters associated with it are:

- `lr`: The learning rate of the gradient descent optimization.
- `reg_param`: The regularization parameter (denoted by λ in the regularized gradient descent cost equation) for L_2 regularization of the classifier's weights.
- `batch_size`: The number of samples in each mini-batch used to update the classifier's parameters in each iteration.

Hyperparameter	**Set of possible values**
`lr` (learning rate)	{1, 5e-1, 1e-1}
`reg_param` (regularization parameter)	{1, 0.1, 0.}
`batch_size` (batch size for SGD)	{1000, 5000}
`n_epochs` (number of training epochs)	{50, 100, 200}

Table 2: Set of hyperparameter values to choose from.

- `n_epochs`: The number of epochs for which the gradient descent optimization is performed.

Defining the set of hyperparameter values to choose from Based on the hyperparameters defined above, we choose 3 learning rates, 3 regularization parameters (which includes 0 indicating no regularization), 2 batch sizes and 3 number of epochs. Taking the Cartesian product, we are scanning over 54 possible sets of hyperparameter values ($3 \times 3 \times 2 \times 3 = 54$). These are as shown in Table 2.

Training and hyperparameter optimization

In order to choose the best set of hyperparameters, we employ a grid search-based hyperparameter optimization. We define a grid of hyperparameters based on the Cartesian product of the values listed in Table 2 and then train a classifier using each such element of this grid.

Furthermore, in order to further improve the generalization performance of the classifier, we also employ a K-fold cross validation along with the grid search with K set to 3. This means that for each hyperparameter value combination, the training dataset will be divided into 3 sub-partitions. Then, a classifier is trained on 2 of these sub-partitions and evaluated on the 3rd, and this is repeated for all 3 partitions, and scores are recorded for all these classifiers.

To summarize, we train a total of 162 classifiers (54 sets of hyperparameters $\times$ 3-fold cross validation = 162 classifiers) for choosing the best hyperparameters. We choose the classification accuracy as the scoring function. The set of hyperparameter values with the highest classification accuracy is chosen as the best set.

The best set of hyperparameter values are listed in Table 3 and these yield a mean classification accuracy of 0.9879.

`n_epochs`	`lr`	`batch_size`	`reg_param`
200	0.5	1000	0.1

Table 3: Best hyperparameter values obtained using grid search with 3-fold cross validation.

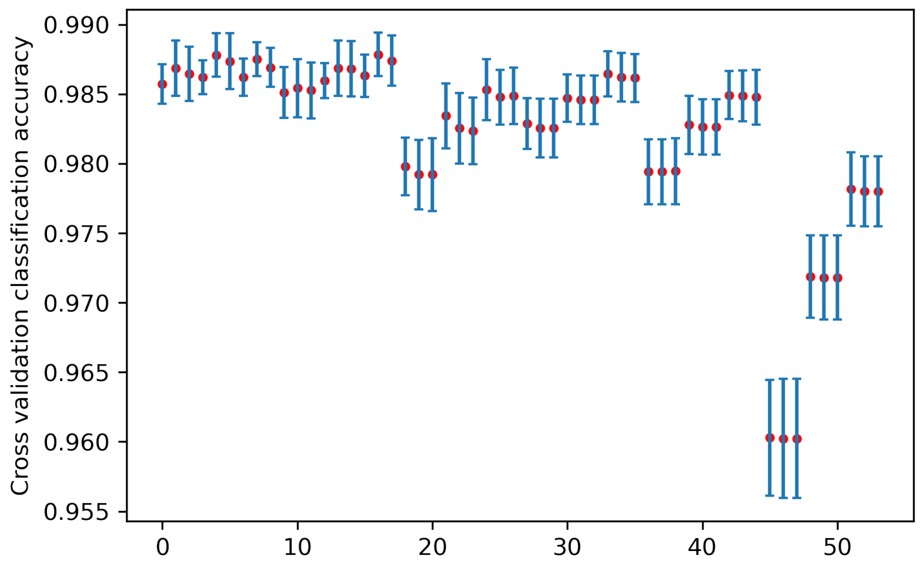


Figure 2: Classification accuracy means (red dots) and standard deviations (blue error bars) for all the 54 hyperparameter value combinations.

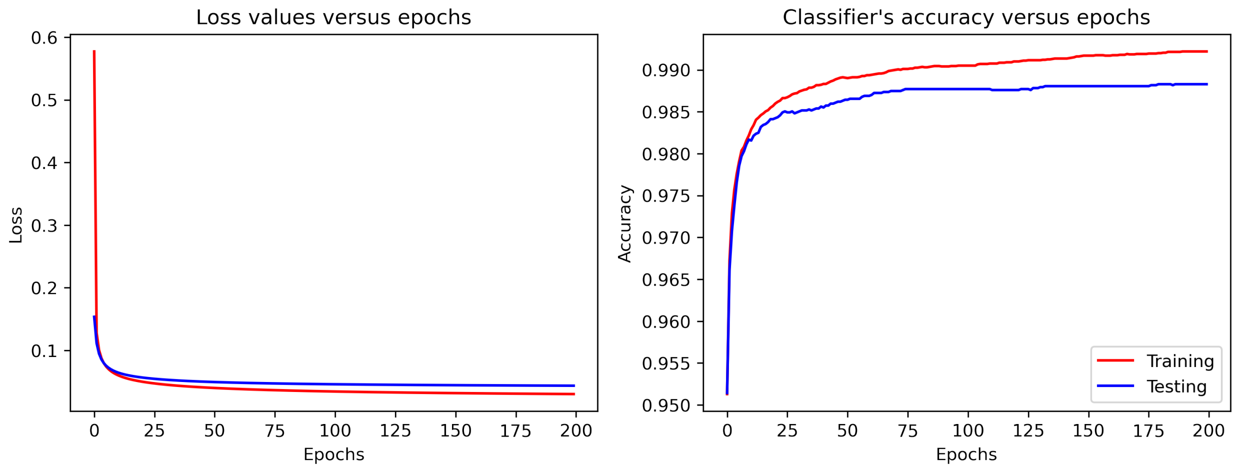


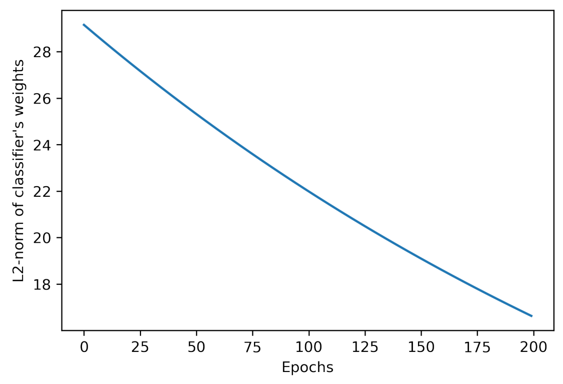
Figure 3: The classifier's loss and accuracy plots over the course of training.

Figure 2 shows a graphical visualization of the 54 hyperparameter value combinations. Red dots depict the mean F1 scores and the blue bars denote the corresponding standard deviations.

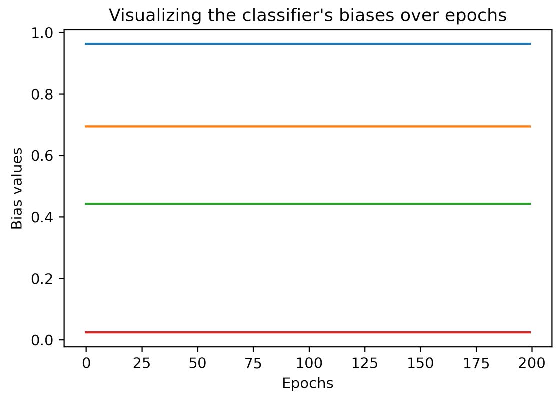
Training the classifier with the best hyperparameters

We train the softmax regression classifier using the best set of hyperparameter values as listed in Table 3.

In order to show that the training is successful, we plot the loss values and the classification accuracy of the classifier over the course of training (200 epochs) for both the training and



(a) L_2-norm of the classifier's weights over the course of training.



(b) The 4 biases of the classifier over the course of training.

Figure 4: Visualizing the weights and the biases of the classifier versus the training epochs.

the testing datasets in Figure 3.

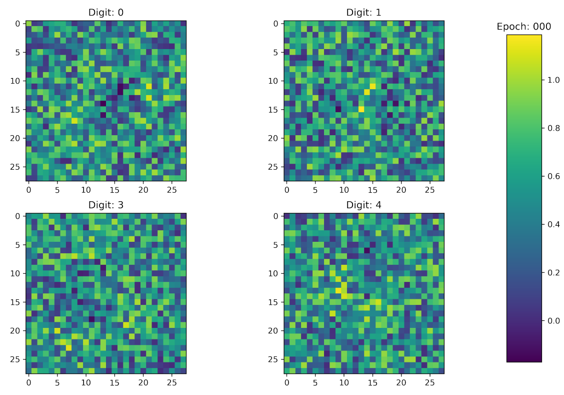
As we can see in Figure 3, the loss values for the training and the testing splits of the dataset are both very small. On the other hand, the classifier's accuracy for both the training and the testing splits are quite high: > 0.99 and > 0.985 for the training and the testing splits respectively. This shows that the classifier has been successfully trained.

Visualizing the classifier's parameters

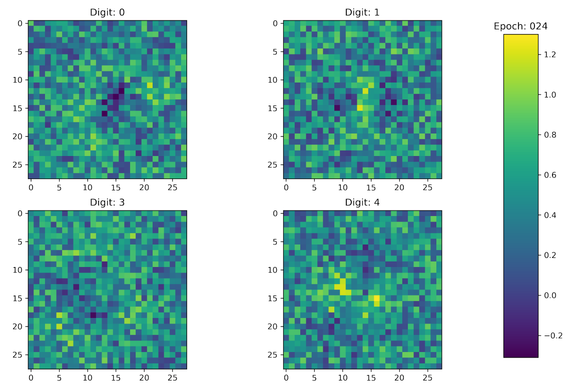
We visualize the classifier's parameters: weights and biases to see how they converge to their optimum values. Since we use an L_2 regularization to the classifier's weights to improve the generalization performance, we visualize the L_2-norm of the weights of the classifier in Figure 4 (a). We observe that L_2 regularization helps the training by reducing the L_2-norm of the weights as the training progresses. Similarly, we visualize each of the 4 biases of the classifier using a different color for each in Figure 4 (b).

Finally, to gain an intuitive understanding of how the logistic regression classifier makes its 4-way classification decision, we plot its weights as a sequence of frames, each frame representing the weights associated with each of the 4 neurons, arranged as a 2×2 grid of 28×28 images (thus 784 pixels, each corresponding to a weight parameter of a neuron). Since we train the classifier for 200 epochs, we visualize the weights for every epoch as an image. This resulting sequence of frames has been saved as a video named `Q2_5.avi` in the `SavedFigs/` directory and all the individual frames are stored in the `VideoFrames/` subdirectory. Figure 5 shows 4 representative frames from the video. We show the weights from the initialization, the 25th , the 75th , and the last epochs.

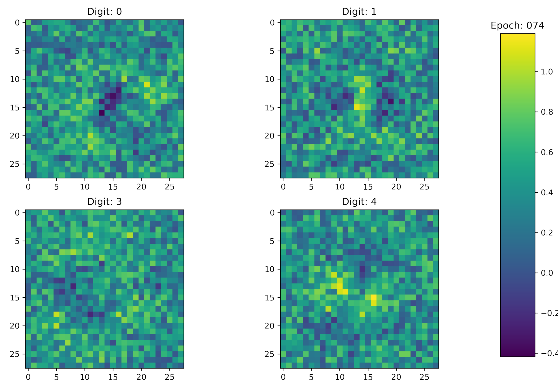
In order to gain some more insight into the classifier's reasoning, we visually inspect the final weights, i.e., after the training has completed. These are visualized in Figure 6. As can be inferred from the accompanying color bar, the brightest yellow color (top of the color bar) indicated the location (thus the weight) with the highest value. We chose the 'viridis' color



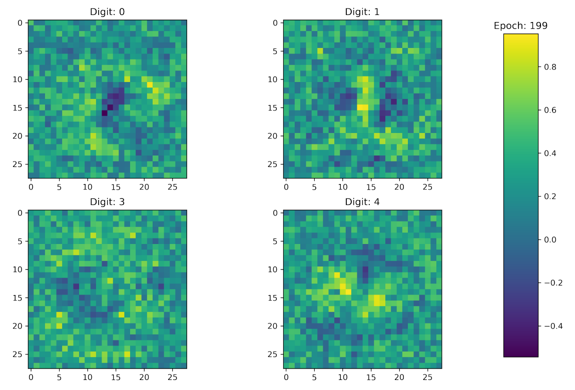
(a) Weights at initialization.



(b) Weights after the 25th epoch.



(c) Weights after the 75th epoch.



(d) Final trained weights.

Figure 5: Representative frames from the video showing the classifier's weights being updated.

map to for maximum visual contrast, but it is still difficult to isolate the most important weights.

In order to better visualize the trained weights and understand (as best as we can) the classifier's decision making process, we threshold the weights from Figure 6 to selectively visualize only some of them. In particular, for each of the 4 neurons' weights, we choose a threshold of 75% of the maximum weight value, and only set those weights to 1, and set all other weights to 0, thus selectively turning on only the highest weights. Figure 7 (a) shows the output of thresholding the weights. Next, we connect the active weights using curves (excluding outliers) and see if these lead to any insights. These are shown in Figure 7 (b).

- For digit '0', we can see that the most active weights are those at the periphery of the digit, and we can roughly connect them to yield a circular shape.
- For digit '1', barring a few outliers, we see that the weights arrange themselves in a linear shape, and we can infer that the neuron has learned the average vertical direction of the handwritten digit '1'.

- The weights for digit '3' are the most revealing as they can be joined to clearly reveal the skeleton of a handwritten digit '3'.
- For digit '4', the most active weights yield the average directions of the three lines that are used to write the digit '4'.

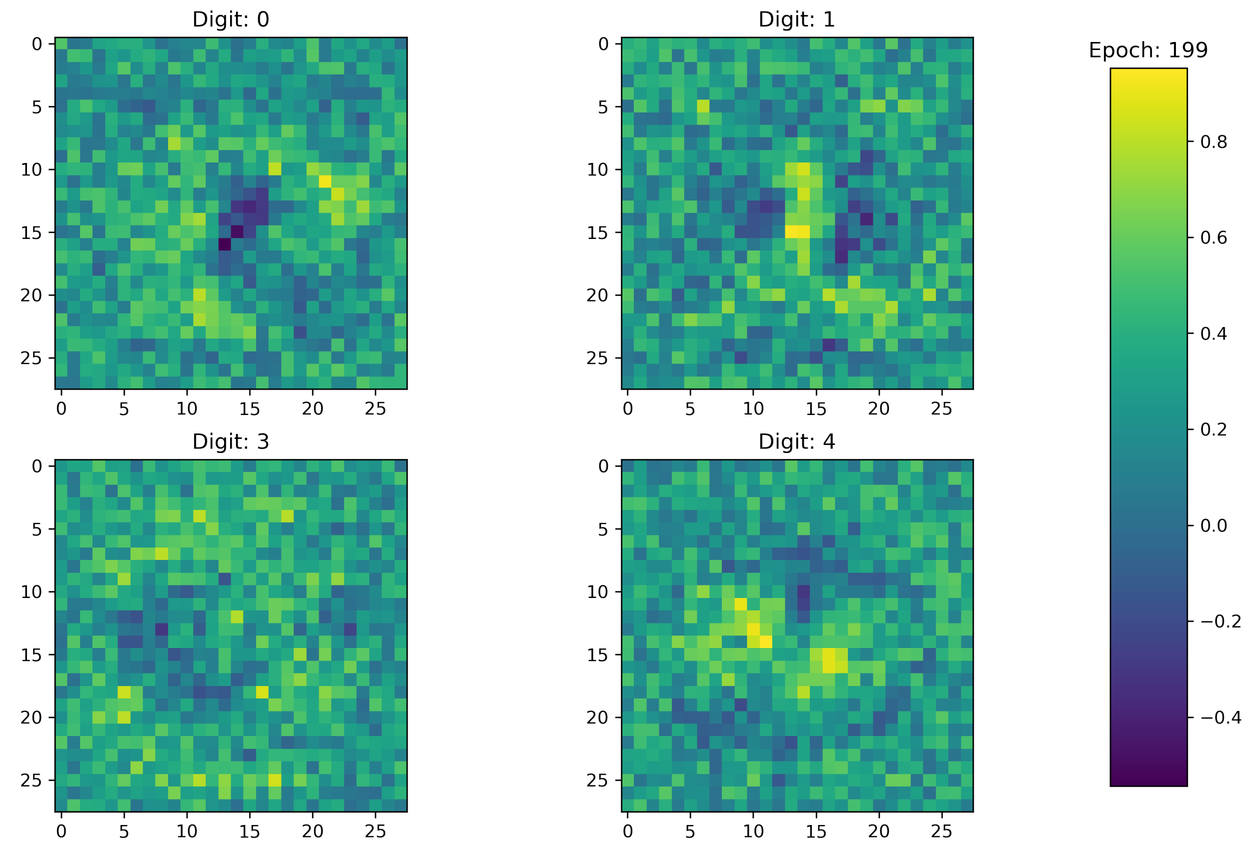
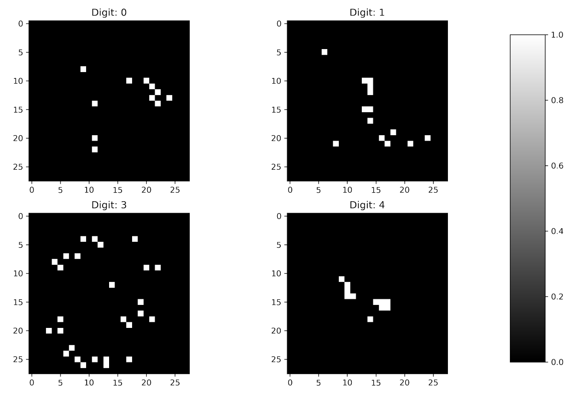


Figure 6: Final trained weights of the classifier visualized as a grid of images.

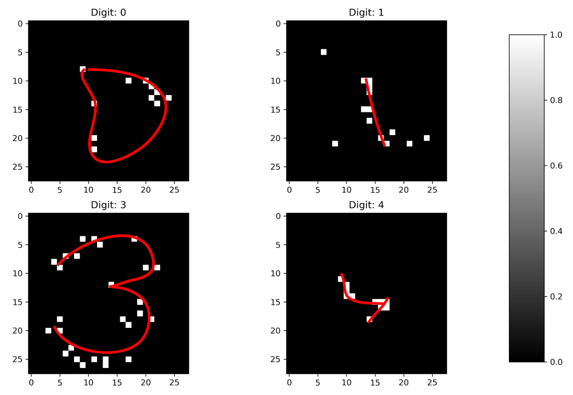
However, it must be noted that despite our best attempts to explain and understand the classifier's decision, the high-dimensional nature of the decision boundary makes it difficult to arrive at trivial explanations. Moreover, the final classification decisions are made by reasoning in the combination of the outputs of all the 4 neurons as well as their biases, so the actual decision making is highly likely to be much more complicated than our explanations here.

Evaluating on the testing partition

In order to assess the generalization performance of the trained classifier, we evaluate it on the testing partition (30% of the original dataset) that we had held out while splitting the data.



(a) Thresholding the final trained weights at 75% of the maximum weight value.



(b) Connecting the active weights using a curve to understand classifier's decisions.

Figure 7: Thresholded weights of the trained classifier and the active weights connected in order to better understand the regions of the image the classifier is focusing on while making its decisions.

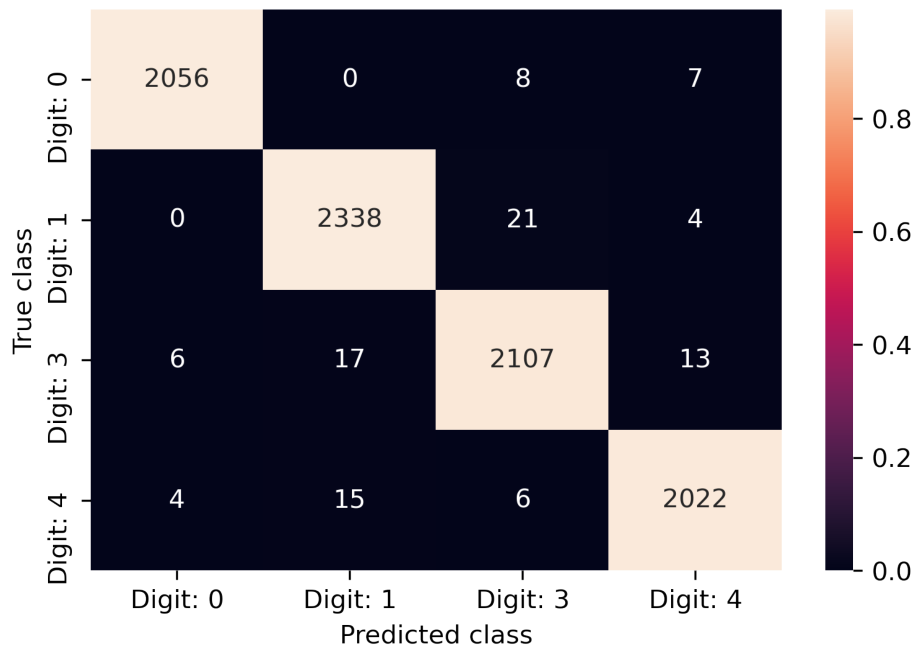


Figure 8: The confusion matrix of the classifier when evaluated on the test split of the dataset.

The confusion matrix constructed using the classifier's predictions and the ground truth labels is as shown in Figure 8. As can be inferred from the accompanying color bar, a lighter color (top of the color bar) indicates that a large fraction of images from that class has been predicted correctly. We see that our lightest colored cells (and thus the highest number of images) are along the principal diagonal of the confusion matrix, thus indicating that the

Class	**Precision**	**Recall**	**F1-score**	**Accuracy**
Digit: 0	0.9952	0.9928	0.9940	0.9928
Digit: 1	0.9865	0.9894	0.9880	0.9894
Digit: 3	0.9837	0.9832	0.9834	0.9832
Digit: 4	0.9883	0.9878	0.9880	0.9878
Average	0.9884	0.9883	0.9883	0.9883

Table 4: Evaluating the MNIST dataset classifier on the testing partition.

classifier performs really well for all the classes.

We also report the accuracy, the precision, the recall, and the F1 score of the classifier for each class as well as for the entire testing partition (macro-averaged) in Table 4, and observe that the classifier achieves excellent performance (> 0.98) for all metrics.