ENSC 813: Deep Learning Systems in Engineering

Kumar Abhishek

Assignment 2

June 23, 2021

Student ID: 301371793

Question 1

Analyzing the digits dataset

The digits dataset contains 1797 images, each of 8 x 8 resolution, of handwritten digits 0 through 9.

Therefore, the target labels are {0, 1, …, 9}.

We count the number of images belonging to each class and they are as follows:

Class	Count
0	178
1	182
2	177
3	183
4	181
5	182
6	181
7	179
8	174
9	180

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

Partitioning the dataset into training and testing splits

In order to prepare the dataset for binary classification, we select 2 classes whose images will be used to train this digit classifier. We choose the digits ‘2’ and ‘7’ because they are quite similar except the tail of 2. However, the user can choose any 2 digits by specifying them in the variables “class0” and “class1”.

Next, we specify the fraction of the dataset to be held out for testing. We choose to set aside 40% 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 2 classes specified above (‘2’ and ‘7’ in our case).
- Change the target labels for these indices to 0 and 1 for class0 and class1 respectively.
- 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 logistic regression classifier (neuron)

Identifying the hyperparameters

The logistic regression classifier has been defined in **utils.py** as a custom class named **LogisticRegressionClassifier**. 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.
- **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 5 learning rates, 4 regularization parameters (which includes 0 indicating no regularization), 2 batch sizes and 2 number of epochs. Taking the Cartesian product, we are scanning over 80 possible sets of hyperparameter values (5 x 4 x 2 x 2 = 80).

Hyperparameter	**Set of possible values**
lr (learning rate)	{1e-1, 1e-2, 1e-3, 1e-4, 1e-5}
reg_param (regularization parameter)	{1, 0.1, 0.01, 0.}
batch_size (batch size for SGD)	{50, 100}
n_epochs (number of training epochs)	{100, 1000}

Table 2: Set of hyperparameter values to choose from.

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 240 classifiers (80 sets of hyperparameters x 3-fold cross validation = 240 classifiers) for choosing the best hyperparameters. We choose the F1 score as the scoring function as it yields a better representation of performance for both the classes. The set of hyperparameter values with the highest F1 score is chosen as the best set.

The best set of hyperparameter values are listed in Table 3 and these yield a mean F1 score of 1.0.

n_epochs	**lr**	**batch_size**	**reg_param**
1000	0.1	50	1

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

Figure 1 shows a graphical visualization of the 80 hyperparameter value combinations. Red dots depict the mean F1 scores and the blue bars denote the corresponding standard deviations. The complete results of the grid search are listed in Table 4, where for each hyperparameter value combination, we report the mean and the standard deviation of the F1 scores.

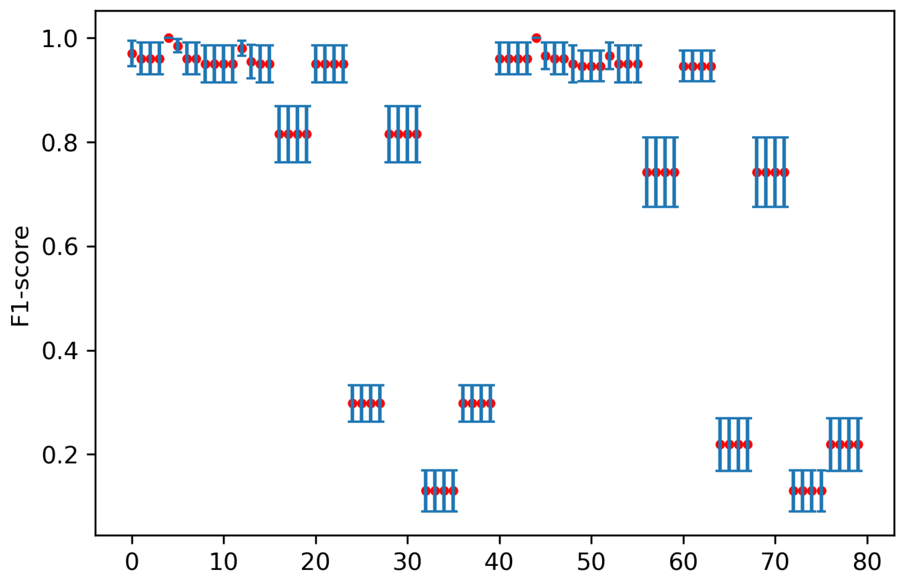

Figure 1: F1 score means (red dots) and standard deviations (blue error bars) for all 80 hyperparameter value combinations.

n_epochs	lr	batch_size	reg_param	F1 score: Mean	F1 score: Std. dev.
100	0.1	50	1	0.9703	0.0248
100	0.1	50	0.1	0.9606	0.0306
100	0.1	50	0.01	0.9606	0.0306
100	0.1	50	0	0.9606	0.0306
1000	0.1	50	1	1	0
1000	0.1	50	0.1	0.9849	0.0128
1000	0.1	50	0.01	0.9606	0.0306
1000	0.1	50	0	0.9606	0.0306
100	0.01	50	1	0.9502	0.0352
100	0.01	50	0.1	0.9502	0.0352
100	0.01	50	0.01	0.9502	0.0352
100	0.01	50	0	0.9502	0.0352
1000	0.01	50	1	0.9801	0.0141
1000	0.01	50	0.1	0.9549	0.0324
1000	0.01	50	0.01	0.9502	0.0352
1000	0.01	50	0	0.9502	0.0352
100	0.001	50	1	0.8153	0.0542
100	0.001	50	0.1	0.8153	0.0542

100	0.001	50	0.01	0.8153	0.0542
100	0.001	50	0	0.8153	0.0542
1000	0.001	50	1	0.9502	0.0352
1000	0.001	50	0.1	0.9502	0.0352
1000	0.001	50	0.01	0.9502	0.0352
1000	0.001	50	0	0.9502	0.0352
100	0.0001	50	1	0.2979	0.0351
100	0.0001	50	0.1	0.2979	0.0351
100	0.0001	50	0.01	0.2979	0.0351
100	0.0001	50	0	0.2979	0.0351
1000	0.0001	50	1	0.8153	0.0542
1000	0.0001	50	0.1	0.8153	0.0542
1000	0.0001	50	0.01	0.8153	0.0542
1000	0.0001	50	0	0.8153	0.0542
100	0.00001	50	1	0.1302	0.0396
100	0.00001	50	0.1	0.1302	0.0396
100	0.00001	50	0.01	0.1302	0.0396
100	0.00001	50	0	0.1302	0.0396
1000	0.00001	50	1	0.2979	0.0351
1000	0.00001	50	0.1	0.2979	0.0351
1000	0.00001	50	0.01	0.2979	0.0351
1000	0.00001	50	0	0.2979	0.0351
100	0.1	100	1	0.9606	0.0306
100	0.1	100	0.1	0.9606	0.0306
100	0.1	100	0.01	0.9606	0.0306
100	0.1	100	0	0.9606	0.0306
1000	0.1	100	1	1	0
1000	0.1	100	0.1	0.9653	0.0255
1000	0.1	100	0.01	0.9606	0.0306
1000	0.1	100	0	0.9606	0.0306
100	0.01	100	1	0.9502	0.0352
100	0.01	100	0.1	0.946	0.0292
100	0.01	100	0.01	0.946	0.0292
100	0.01	100	0	0.946	0.0292
1000	0.01	100	1	0.9653	0.0255

1000	0.01	100	0.1	0.9502	0.0352
1000	0.01	100	0.01	0.9502	0.0352
1000	0.01	100	0	0.9502	0.0352
100	0.001	100	1	0.742	0.067
100	0.001	100	0.1	0.742	0.067
100	0.001	100	0.01	0.742	0.067
100	0.001	100	0	0.742	0.067
1000	0.001	100	1	0.946	0.0292
1000	0.001	100	0.1	0.946	0.0292
1000	0.001	100	0.01	0.946	0.0292
1000	0.001	100	0	0.946	0.0292
100	0.0001	100	1	0.2188	0.0508
100	0.0001	100	0.1	0.2188	0.0508
100	0.0001	100	0.01	0.2188	0.0508
100	0.0001	100	0	0.2188	0.0508
1000	0.0001	100	1	0.742	0.067
1000	0.0001	100	0.1	0.742	0.067
1000	0.0001	100	0.01	0.742	0.067
1000	0.0001	100	0	0.742	0.067
100	0.00001	100	1	0.1302	0.0396
100	0.00001	100	0.1	0.1302	0.0396
100	0.00001	100	0.01	0.1302	0.0396
100	0.00001	100	0	0.1302	0.0396
1000	0.00001	100	1	0.2188	0.0508
1000	0.00001	100	0.1	0.2188	0.0508
1000	0.00001	100	0.01	0.2188	0.0508
1000	0.00001	100	0	0.2188	0.0508

Table 4: F1 scores (means and standard deviations) for all 80 hyperparameter value combinations.

Training the classifier with the best hyperparameters

We train the logistic 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 (1000 epochs) for both the training and the testing datasets in Figure 2.

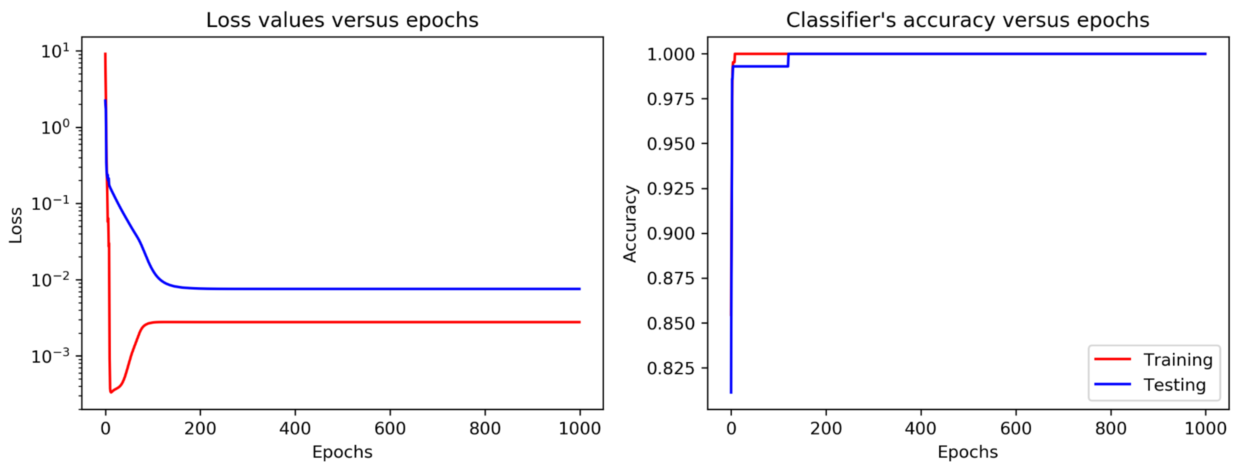

Figure 2: The classifier’s loss and accuracy plots over the course of training.

As we can see in Figure 2, the loss values for the training and the testing datasets are both very small: ~10^{-3} and ~10^{-2} respectively. On the other hand, the classifier’s accuracy for both the training and the testing datasets are 1.0, meaning a 100% classification accuracy on both the datasets. This shows that the classifier has been successfully trained.

Visualizing the classifier’s parameters

We visualize the classifier’s parameters: weights and bias to see how they converge to their optimum values. Since the input is 64-dimensional, the classifier has 64 weights and 1 bias.

Figure 3 and Figure 4 show the classifier’s weights and bias as they are updated over the course of training. In Figure 3, we show 64 plots, one for each weight parameter. They have been presented in an 8 x 8 grid for improved readability. We observe that as training progresses, the weights converge to the best values. Similarly, in Figure 4, we observe that the bias of the classifier converges to the optimum value.

Since we use an L_2 regularization to the classifier’s weights to improve the generalization performance, it is also interesting to visualize how the regularization affects the weights. To do so, we visualize the L_2-norm of the weights of the classifier in Figure 5. We observe that L_2 regularization helps the training by reducing the L_2-norm of the weights as the training progresses.

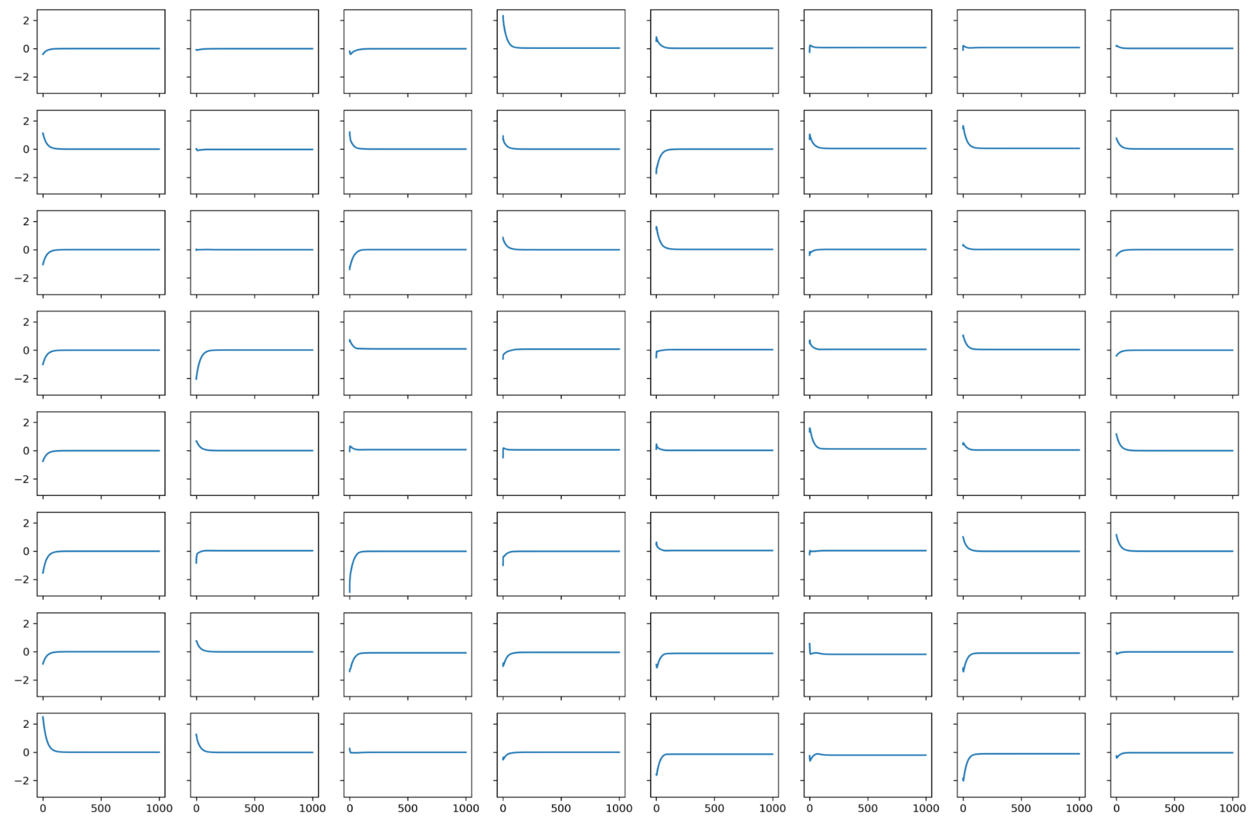

Figure 3: Visualizing the weights of the classifier versus the training epochs. The 64 weights are arranged in an 8 x 8 grid for improved readability. The Y-axis denotes the weight values and the X-axis denotes the number of training epochs.

Finally, to gain an intuitive understanding of how the logistic regression classifier makes its binary decision, we plot its weights as a sequence of frames, each frame representing its weights arranged in the form of an 8 x 8 image (thus 64 pixels, each corresponding to a weight parameter). Since we train the classifier for 1000 epochs, we visualize the weights for every 10th epoch. This resulting sequence of frames has been saved as a video named "**Q1_6.avi**" in the "**SavedFigs/**" directory. Figure 6 shows some representative frames from the video. We show the weights from the initialization, the 100th, the 400th, and the last epochs.

Notice how as the training progresses, the classifier learns to assign the largest weight (darkest color) to the region at the bottom right of the image. If we look at images of handwritten '2' and '7', the only location where they differ is the "tail" at the bottom of the digit '2'. Therefore, since the classifier learns to assign the highest weight to that region, this conclusively proves that the classifier has been trained successfully.

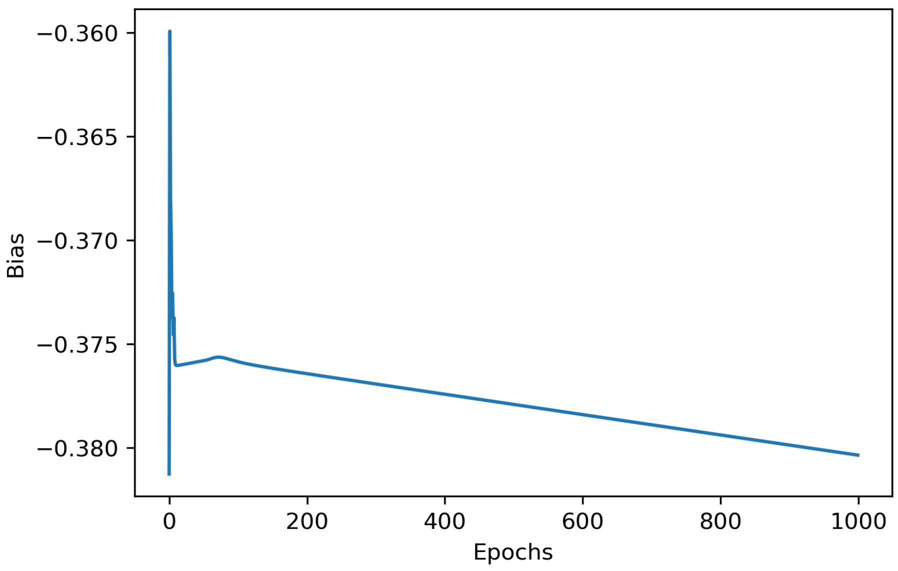

Figure 4: Visualizing the bias of the classifier versus the training epochs.

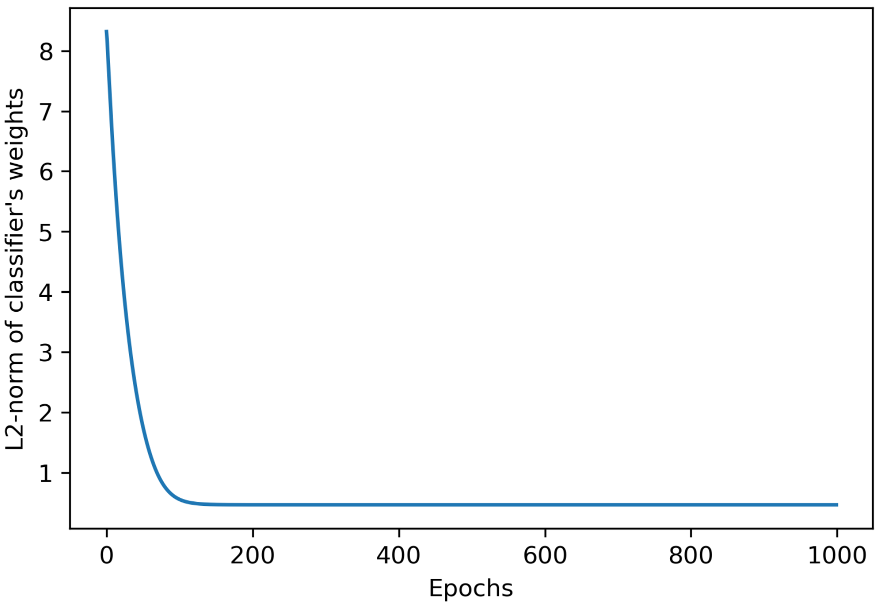

Figure 5: Visualizing the L_2-norm of the classifier's weights versus the training epochs.

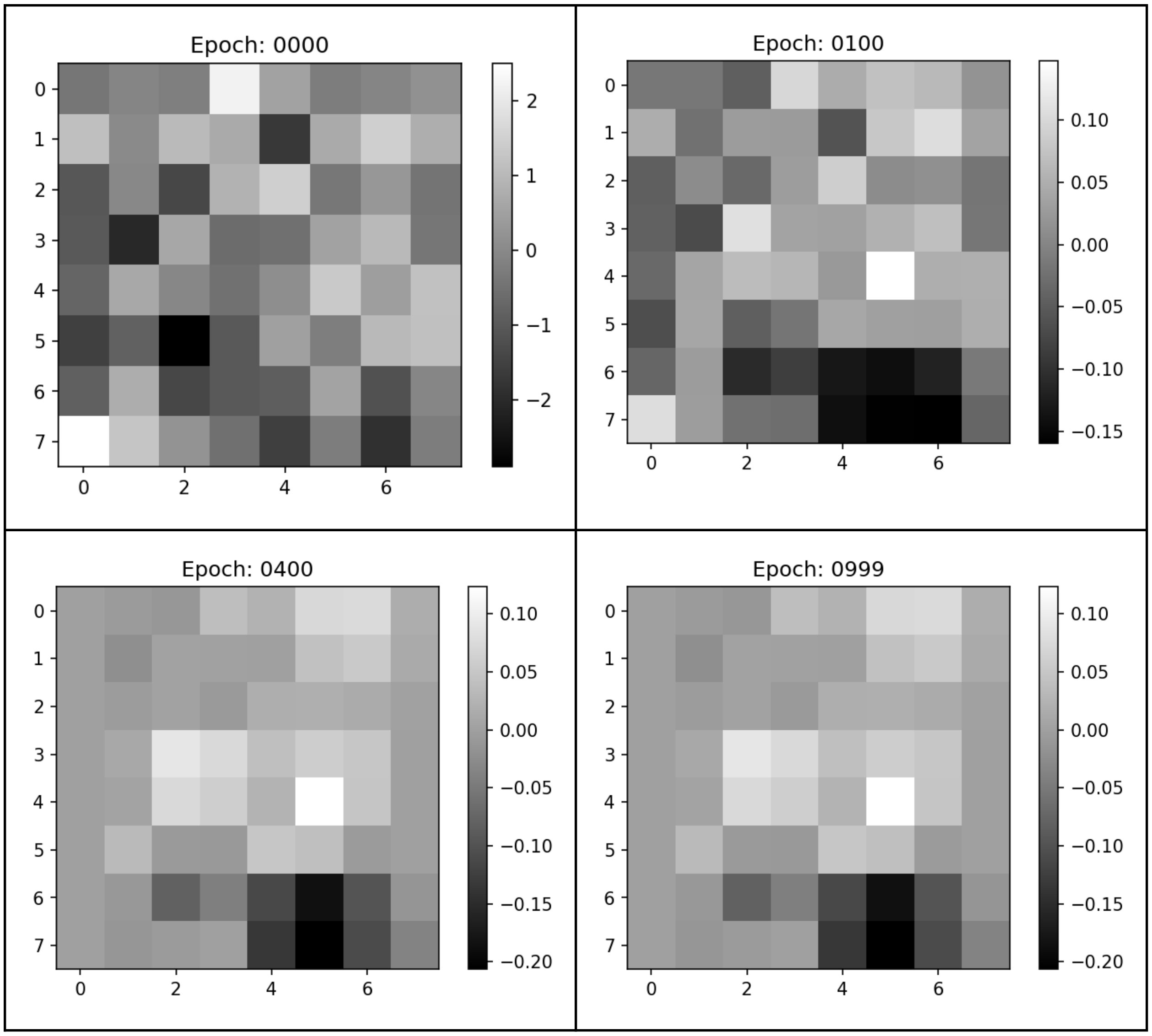

Figure 6: Representative frames from the video showing the classifier's weights being updated.

Evaluating on the testing partition

In order to assess the generalization performance of the trained classifier, we evaluate it on the testing partition (40% of the original dataset) that we had held out while splitting the data. We report the accuracy, the precision, the recall, the sensitivity, the specificity, and the F1 score of the classifier in Table 5, and observe that the classifier obtains a 100% score for all metrics.

Accuracy	Precision	Recall	Sensitivity	Specificity	F1 score
1.0	1.0	1.0	1.0	1.0	1.0

Table 5. Evaluating the digits dataset classifier on the testing partition.

Analyzing the bias-variance tradeoff

We recall from the lecture that if the classifier exhibits a low training error but a large testing error (or equivalently, good performance on the training set and poor performance on the testing set), it exhibits a high variance and is said to have overfit to the training dataset.

On the other hand, if the classifier exhibits high training and test errors (or equivalently, poor performance on both training and test sets), it exhibits a high bias and is said to have underfit to the training dataset.

Now, let us look at one instance of both these scenarios and analyze the findings.

A model with a high variance

Figure 7 shows a scenario where the trained model has a high variance. This model has been trained with hyperparameters: {**'n_epochs'**: 100, **'lr'**: 1e-1, **'reg_param'**: 0, **'batch_size'**: 10}.

We see from the plots that the loss values for the training set have become much smaller (~1e-6) as compared to the best classifier trained above (as shown in Figure 2), whereas the loss values for the testing set remain comparatively much larger.

Similarly, the classifier achieves 100% accuracy on the training set in less than 10 epochs, whereas the performance on the testing set is considerably lower and does not improve even as training continues.

Both these plots indicate that the classifier has overfit to the training dataset and is an example of high variance. This can be explained by the combination of a large learning rate and no regularization, which causes the model to overfit to the training dataset.

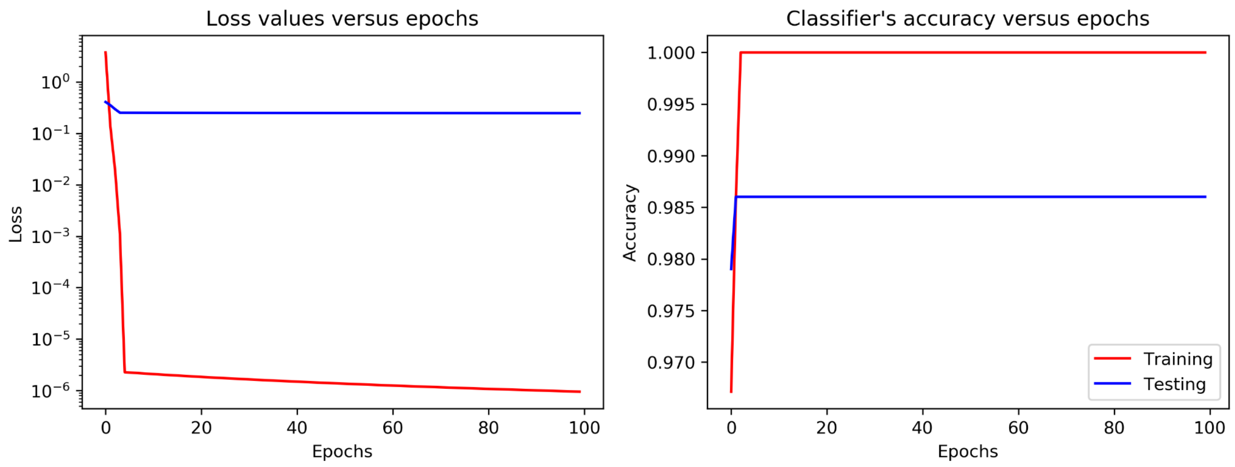

Figure 7: Loss and accuracy plots for a classifier with a high variance.

A model with a high bias: Example 1

Figure 8 shows a scenario where the trained model has a high bias. This model has been trained with hyperparameters: {**'n_epochs'**: 100, **'lr'**: 1e-3, **'reg_param'**: 1, **'batch_size'**: 100}.

We see from the plots that the loss values for both the training set and the testing set remain quite large even towards the end of training.

Similarly, the classifier's accuracy on both training and testing sets never approaches high values and flattens around 80%.

Both these plots indicate that the classifier has underfit to the training dataset and is an example of high bias. This can be explained by the combination of a small learning rate and the presence of regularization, which causes the model to learn very slowly and therefore it underfits to the training dataset.

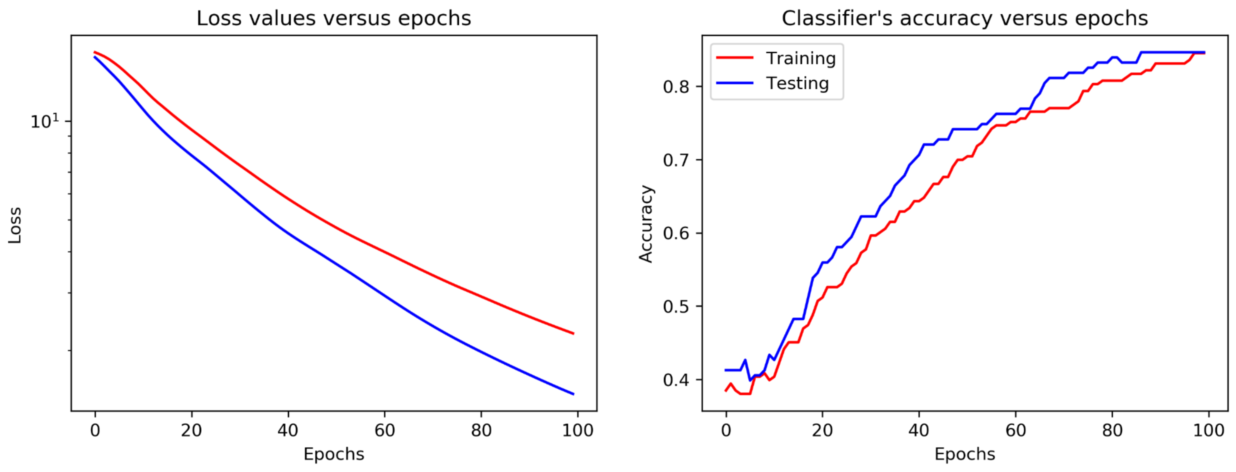

Figure 8: Loss and accuracy plots for a classifier with a high bias.

A model with a high bias: Example 2

Figure 9 shows a scenario where the trained model has a very high bias. This model has been trained with hyperparameters: {**'n_epochs'**: 100, **'lr'**: 1e-4, **'reg_param'**: 1, **'batch_size'**: 100}.

We see from the plots that the loss values for both the training set and the testing set are very large (> 10) even towards the end of training.

Similarly, the classifier's accuracy on both training and testing sets is very poor and is less than 50% (meaning it is worse than random guessing for a binary classification task, since for a binary classification task, a random guess yields 50% accuracy).

Both these plots indicate that the classifier has extremely underfit to the training dataset and is an example of very high bias. This can be explained by the combination of a very small learning rate and the presence of regularization, which causes the model to learn very slowly and therefore it underfits to the training dataset.

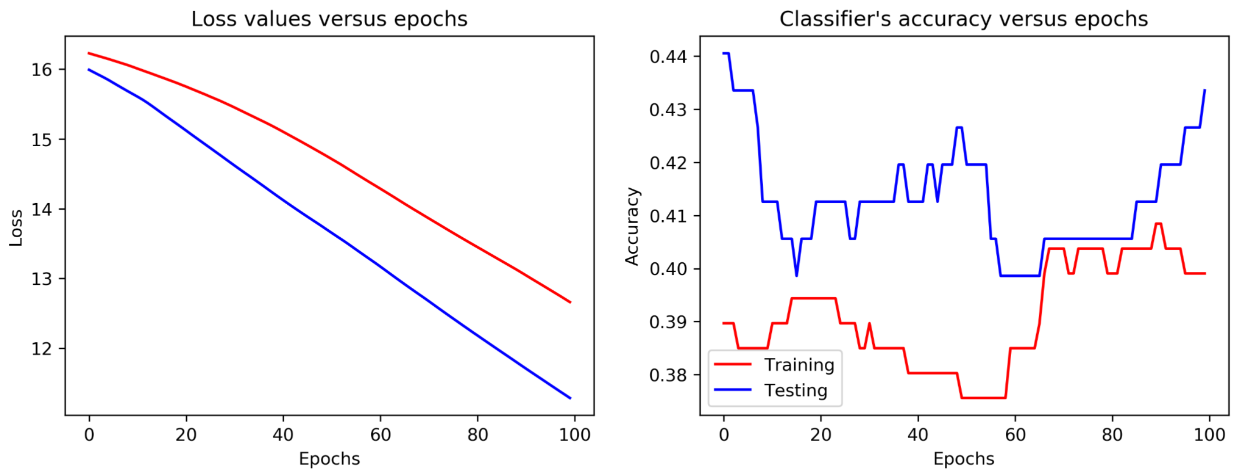

Figure 9: Loss and accuracy plots for a classifier with a very high bias.

Question 2

Analyzing the MNIST dataset

The digits dataset contains 70000 images, each of 28 x 28 resolution, of handwritten digits 0 through 9.

Therefore, the target labels are {0, 1, …, 9}.

We count the number of images belonging to each class and they are as follows:

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

Table 6: Class-wise population of the MNIST dataset.

Partitioning the dataset into training and testing splits

As we did with the digits dataset above, in order to prepare the dataset for binary classification, we select 2 classes whose images will be used to train this digit classifier. We choose the digits ‘2’ and ‘7’ because they are quite similar except the tail of 2. However, the user can choose any 2 digits by specifying them in the variables “class0” and “class1”.

Next, we specify the fraction of the dataset to be held out for testing. We choose to set aside 40% 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 2 classes specified above (‘2’ and ‘7’ in our case).
- Change the target labels for these indices to 0 and 1 for class0 and class1 respectively.
- 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 logistic regression classifier (neuron)

Identifying the hyperparameters

Similar to our solution for the digits dataset, we use the logistic regression classifier which has been defined in **utils.py** as a custom class named **LogisticRegressionClassifier**. As before, 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.
- **n_epochs**: The number of epochs for which the gradient descent optimization is performed.

Training a classifier with no regularization

We first train a digit classifier with no regularization of the classifier’s weights. Therefore, we use the hyperparameters as shown in Table 7.

n_epochs	lr	batch_size	reg_param
500	0.001	1000	0

Table 7: Best hyperparameter values for a classifier trained with no weight regularization.

The hyperparameters in Table 6 are optimal for a classifier without regularization because in the absence of regularization, a higher value of either the learning rate or the number of training epochs will most likely cause the model to overfit to the training dataset and thus yield poor generalization performance when evaluated on the testing dataset.

Therefore, we train the logistic regression classifier with these hyperparameters, and 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 for both the training and the testing datasets in Figure 10.

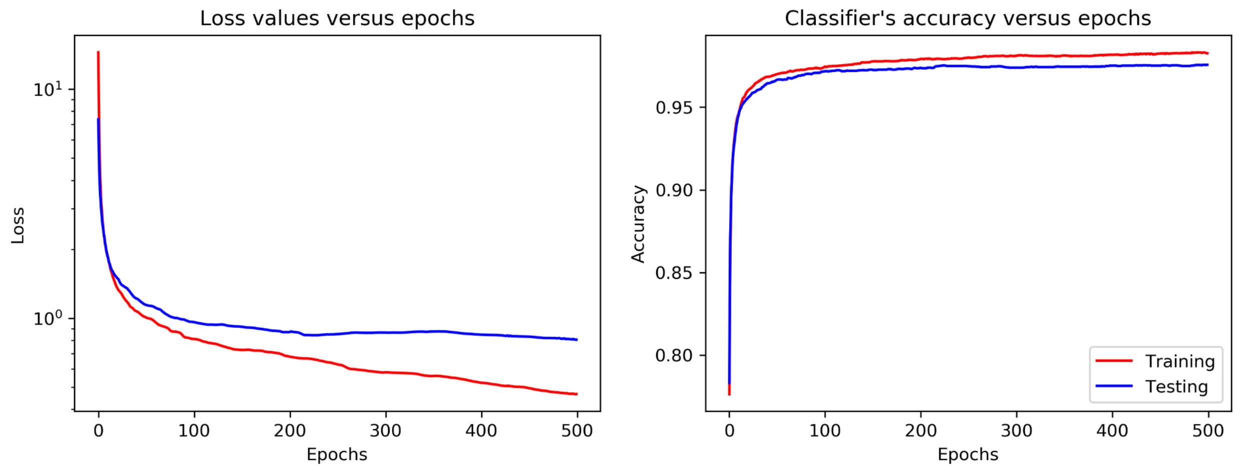

Figure 10: The classifier's loss and accuracy plots over the course of training when trained without regularization.

As we can see in Figure 10, the loss values for the training and the testing datasets both converge. On the other hand, the classifier's accuracy for both the training and the testing datasets are quite high (> 95% accuracy). This shows that the classifier has been successfully trained.

Finally, we report the classification performance metrics for this classifier in Table 8.

Accuracy	Precision	Recall	Sensitivity	Specificity	F1 score
0.9757	0.9770	0.9753	0.9753	0.9760	0.9762

Table 8. Evaluating the MNIST dataset classifier (trained with no regularization) on the testing partition.

Training a classifier with appropriate regularization

Next, we train a digit classifier with an appropriate amount (neither too high nor too low) of regularization of the classifier's weights. Therefore, we use the hyperparameters as shown in Table 8.

n_epochs	lr	batch_size	reg_param
1000	0.001	1000	0.1

Table 8: Best hyperparameter values for a classifier trained with no weight regularization.

The hyperparameters in Table 8 are optimal because in the presence of regularization, the number of training epochs can be larger as compared to the scenario with no regularization, since the presence of regularization will help avoid overfitting to the training dataset.

Therefore, we train the logistic regression classifier with these hyperparameters, and 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 for both the training and the testing datasets in Figure 11.

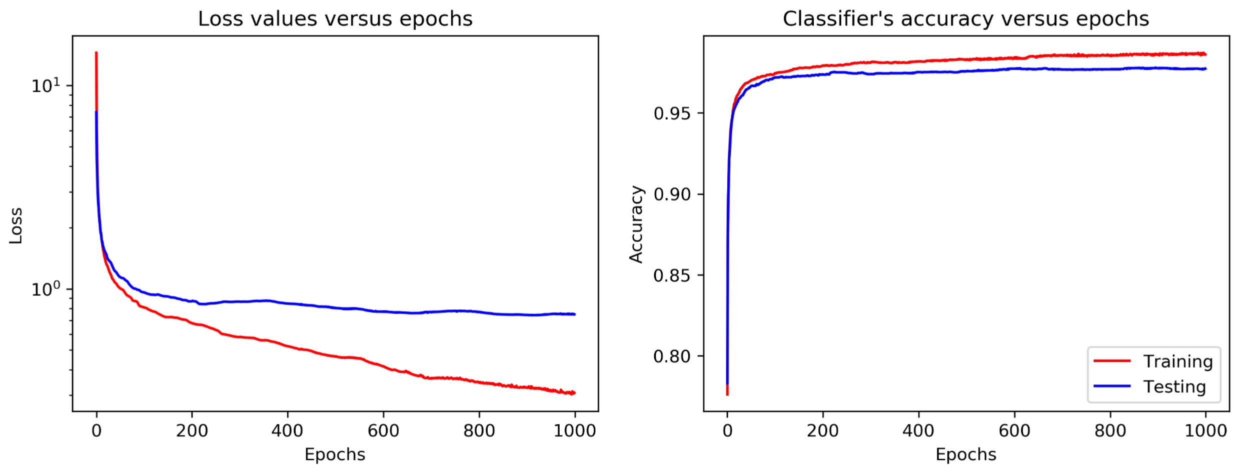

Figure 11: The classifier's loss and accuracy plots over the course of training when trained with an appropriate amount of regularization.

As we can see in Figure 11, the loss values for the training and the testing datasets both converge. On the other hand, the classifier's accuracy for both the training and the testing datasets are quite high (> 95% accuracy). This shows that the classifier has been successfully trained.

Finally, we report the classification performance metrics for this classifier in Table 9.

Accuracy	**Precision**	**Recall**	**Sensitivity**	**Specificity**	**F1 score**
0.9772	0.9774	0.9781	0.9781	0.9764	0.9777

Table 9. Evaluating the MNIST dataset classifier (trained with an appropriate amount of regularization) on the testing partition.

Training a classifier with high regularization

Finally, we train a digit classifier with a high regularization of the classifier's weights. Therefore, we use the hyperparameters as shown in Table 10.

n_epochs	lr	batch_size	reg_param
1000	0.001	1000	10

Table 10: Best hyperparameter values for a classifier trained with no weight regularization.

The hyperparameters in Table 10 are optimal because except the regularization parameter, everything else is the same as that listed in Table 8, and we set the regularization parameter to 10, which is a high value.

Therefore, we train the logistic regression classifier with these hyperparameters, and 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 for both the training and the testing datasets in Figure 12.

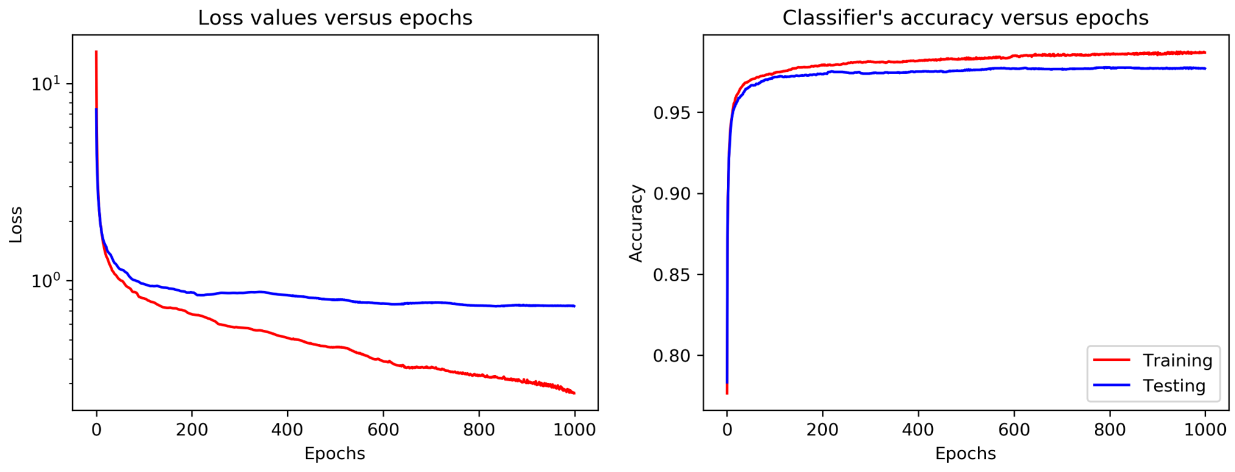

Figure 12: The classifier's loss and accuracy plots over the course of training when trained with a high regularization.

As we can see in Figure 12, the loss values for the training and the testing datasets both converge. On the other hand, the classifier's accuracy for both the training and the testing datasets are quite high (> 95% accuracy). This shows that the classifier has been successfully trained.

Finally, we report the classification performance metrics for this classifier in Table 11.

Accuracy	Precision	Recall	Sensitivity	Specificity	F1 score
0.9771	0.9780	0.9770	0.9770	0.9771	0.9775

Table 11. Evaluating the MNIST dataset classifier (trained with a high regularization) on the testing partition.

Comparing the performance of the three classifiers

Table 12 shows a comparison of the classification performance metrics of the three classifiers trained above. For each metric, the value in bold denotes the classifier which achieved the highest value. We see that the classifier with no regularization performs the worst comparatively, whereas the classifier trained with an appropriate amount of regularization performs the best among all of them.

Regularization	Accuracy	Precision	Recall	Sensitivity	Specificity	F1 score
None	0.9757	0.9770	0.9753	0.9753	0.9760	0.9762
Medium	**0.9772**	0.9774	**0.9781**	**0.9781**	0.9764	**0.9777**
High	0.9771	**0.9780**	0.9770	0.9770	**0.9771**	0.9775

Table 12. Comparing the performance of the three classifiers (trained with different amounts of weight regularization) on the testing partition of the dataset.

Analyzing the bias-variance tradeoff

Similar to our analysis with the digits dataset above, let us look at examples of scenarios where the classifier exhibits a high variance or a high bias.

A model with a high variance

Figure 13 shows a scenario where the trained model has a high variance. This model has been trained with hyperparameters: {**'n_epochs'**: 1000, **'lr'**: 1e-1, **'reg_param'**: 0, **'batch_size'**: 1000}.

We see from the plots that the loss values for the training set have become very small (< 1e-14) as compared to the best model trained above (as shown in Figure 11), whereas the loss values for the testing set remain comparatively much larger (~1e0).

Similarly, the model achieves 100% accuracy on the training set in ~200 epochs, whereas the performance on the testing set is comparatively lower and does not improve even as training continues. Instead, we see that the testing accuracy decreases after ~500 epochs.

Both these plots indicate that the model has overfit to the training dataset and is an example of high variance. This can be explained by the combination of a large learning rate, a large number of training epochs, and no regularization, which causes the model to overfit to the training dataset.

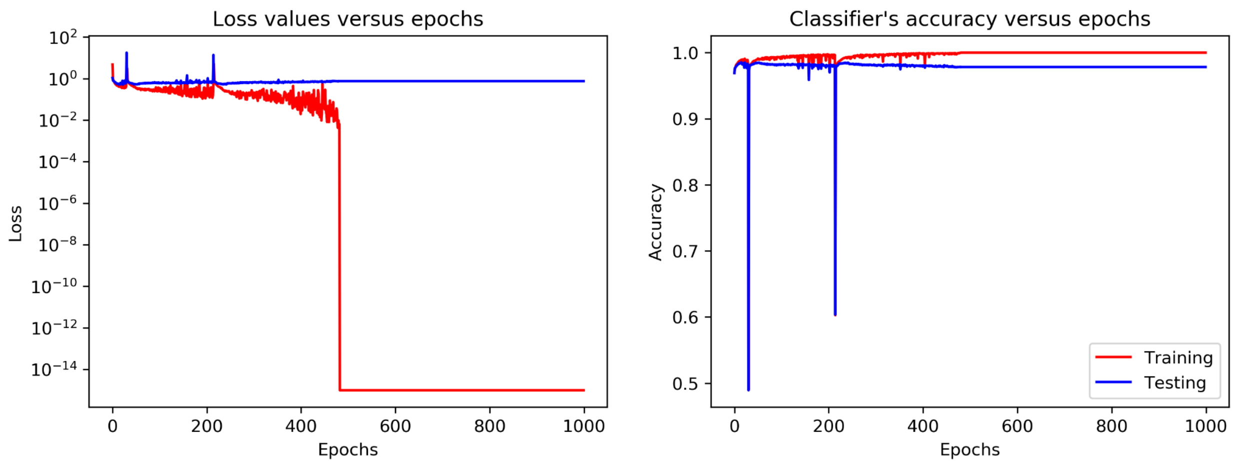

Figure 13: Loss and accuracy plots for a classifier with a high variance.

A model with a high bias

Figure 14 shows a scenario where the trained model has a high bias. This model has been trained with hyperparameters: {**'n_epochs'**: 100, '**lr**': 1e-5, **'reg_param'**: 1, **'batch_size'**: 1000}.

We see from the plots above that the loss values for both the training set and the testing set remain quite large (~1e1) even towards the end of training.

Similarly, the model's accuracy on both training and testing sets never approaches high values and only remains around 70%-80%.

Both these plots indicate that the model has underfit to the training dataset and is an example of high bias. This can be explained by the combination of a very small learning rate, a large number of training epochs, and the presence of regularization, which causes the model to learn very slowly and therefore it underfits to the training dataset.

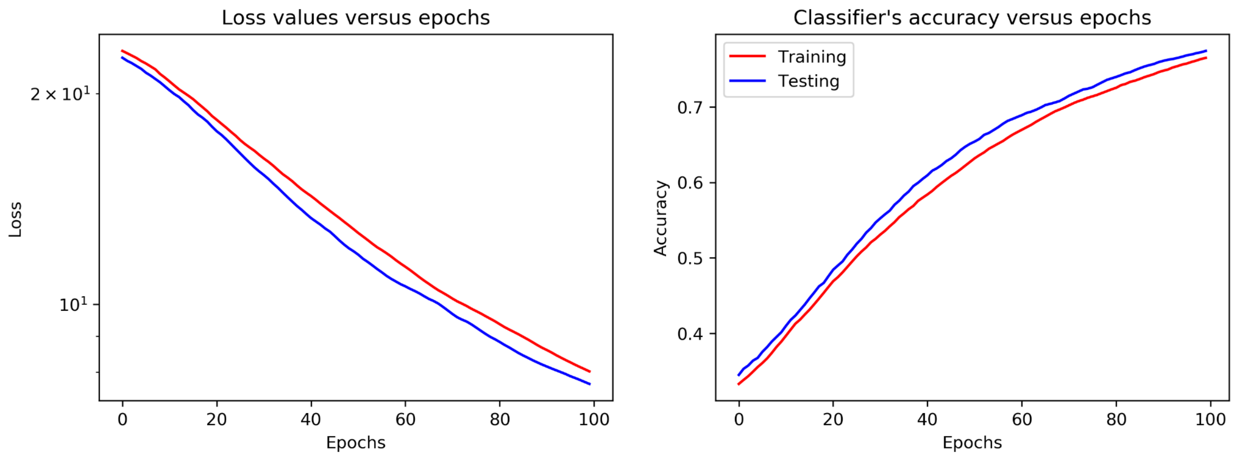

Figure 14: Loss and accuracy plots for a classifier with a high bias.