PRNN ASSIGNMENT 2

Kalyan Reddy
SR. No. 21361

Ramesh Babu
SR. No. 20950

Abstract

This report consists the results and interpretations of a Assignment 2. Python code for this assignment is in ***PRNN_Assignment_2_Ramesh_Kalyan.zip*** *file*

1. Problem 1: SVM

SVM stands for Support Vector Machine, which is a machine learning algorithm used for classification and regression tasks. SVM works by finding the hyperplane that maximally separates the classes in the feature space.

SVM with slack formulation is also known as soft-margin SVM, which allows for some degree of misclassification by introducing slack variables that penalize the misclassified instances. This formulation is useful in cases where the data is not linearly separable or when there is noise or outliers in the data.

On the other hand, SVM without slack formulation is also known as hard-margin SVM, which aims to find the hyperplane that completely separates the classes without any misclassifications. This formulation works only when the data is linearly separable and there are no outliers or noise in the data.

The difference between SVM with and without slack formulation is that the former is more flexible and can handle data that is not perfectly separable, whereas the latter is more strict and can only be used when the data is linearly separable.

In general, SVM with slack formulation is more commonly used because it can handle a wider range of datasets and is less sensitive to outliers. However, if the data is known to be linearly separable and there are no outliers, then SVM without slack formulation can be used for faster training and better performance.

1.1. Implementation

Given a set of training data $D = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$, we aim to find a hyperplane that separates the data into two classes with maximum margin.

SVM without slack formulation:

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \text{ s.t.} \qquad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$

SVM with slack formulation:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i \qquad \xi_i \geq 0 \quad \forall i$$

In the slack formulation, ξ_i represents the slack variables and C is the regularization parameter that balances the trade-off between maximizing the margin and minimizing the classification error.

1.2. Results for p3 in A1

1.2.1 Gaussian Kernel

Gaussian Kernel is given by $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$

Without Slack: Test accuracy with parameter **gamma 0.1** : 49.64%

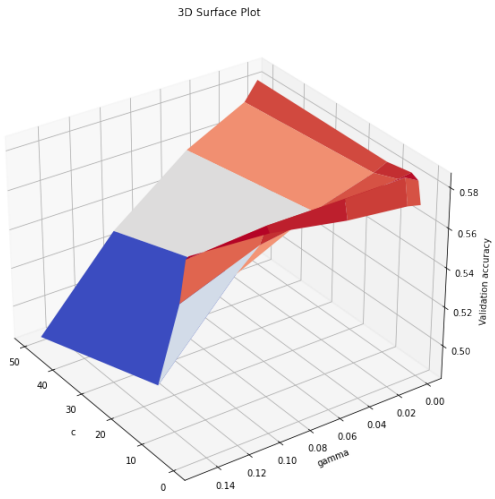


Figure 1. Grid Search plot for c,gamma and accuracy

With Slack:

- **Best Validation accuracy: 58.51 with parameters: C=1, gamma=0.01**
- **Test accuracy with best parameters found using Grid Search : 58.08%**

Plots: Grid Search Plot is given by:

		Gamma				
		0.001	**0.01**	**0.05**	**0.1**	**0.15**
C	**0.1**	57.035	58.14	58.32	58.105	58.175
	1	57.2	58.51	58.225	57.785	57.24
	3	57.42	57.975	57.465	57.18	56.87
	10	58.245	**58.51**	57.35	55.125	52.75
	50	58.49	58.23	55.875	50.98	48.49

Table 1. Validation Accuracy Grid Search Results for C and Gamma

1.2.2 Polynomial Kernel

Polynomial Kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$ where:

- x_i and x_j are two data points
- γ is a scaling factor
- r is a coefficient in the kernel
- d is the degree of the polynomial

Without Slack: Test accuracy with parameters **degree 3** , **gamma 0.001** : 56.553%

With Slack:

- **Best Validation accuracy: 57.23 with parameters: 'C': 0.1, 'gamma': 0.1, 'degree': 3**
- **Test accuracy with best parameters found using Grid Search : 57.0933%**

Grid Search for each degree:

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	19.84%	48.35%	51.86%	51.92%
	1	23.37%	51.24%	51.96%	51.93%
	2	36.68%	51.64%	51.94%	51.94%

Table 2. Validation Accuracy Grid Search Results for C and Gamma for **Degree=2**

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	19.84%	37.355%	**57.23%**	56.705%
	1	19.84%	52.59%	56.805%	46.64%
	2	19.84%	54.73%	56.79%	54.65%

Table 3. Validation Accuracy Grid Search Results for C and Gamma for **Degree=3**

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	19.84%	19.93%	49.24%	47.36%
	1	19.84%	36.42%	48.49%	23.95%
	2	19.84%	36.53%	48.15%	22.76%

Table 4. Validation Accuracy Grid Search Results for C and Gamma for **Degree=4**

1.2.3 Linear kernel

The linear kernel used in SVM is defined as:

$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j}$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ are input feature vectors. The dot product $\mathbf{x_i} \cdot \mathbf{x_j}$ is a measure of similarity between the two feature vectors. This kernel is called the linear kernel because it corresponds to a linear decision boundary in the input feature space.

Without Slack: Test accuracy with c=high value : 56.2133%

With Slack:

- **Best Validation accuracy: 57.34 with parameters: 'C': 0.0001**
- **Test accuracy with best parameters found using Grid Search : 57.03%**

Grid Search for c:

C	**Accuracy**
10^{-6}	19.82%
10^{-5}	19.82%
10^{-4}	**57.34%**
0.001	56.71%
0.01	56.44%
0.1	56.43%
1	56.43%

Table 5. Accuracy of SVM model for different values of C

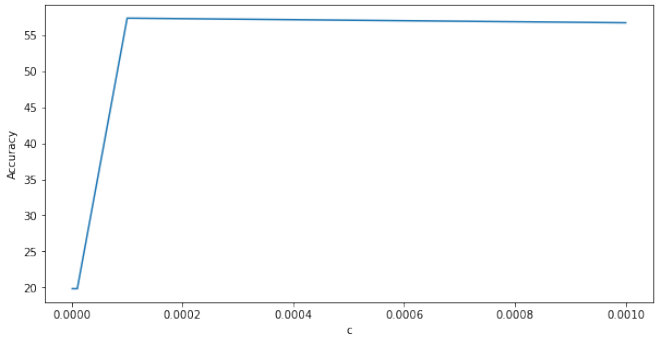


Figure 2. Grid Search plot for c and accuracy

1.3. Results for p4 in A1

1.3.1 Gaussian Kernel

Gaussian Kernel is given by $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$

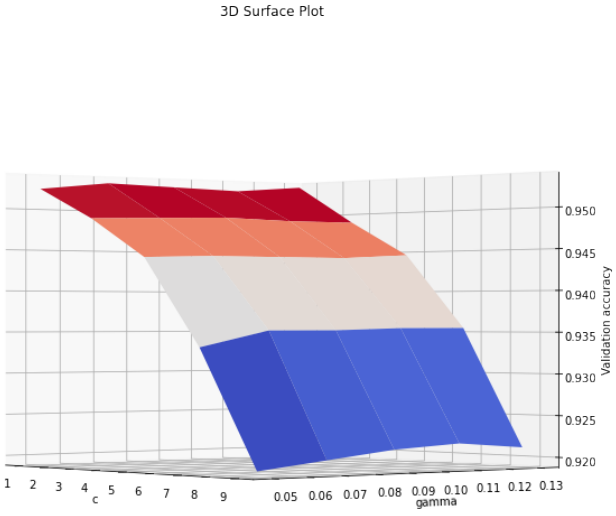


Figure 3. Grid Search plot for c,gamma and accuracy

Without Slack: Test accuracy with parameter **gamma 0.1** : 92.42%

With Slack:

- **Best Validation accuracy: 95.36 with parameters: C=9, gamma=0.05**
- **Test accuracy with best parameters found using Grid Search : 93.47%**

The grid search was conducted to find the best combination of two hyperparameters: C and gamma. The accuracy of the SVM model was evaluated for each combination of C and gamma in the grid.

The results of the grid search are presented in two tables. The first table shows the validation accuracy of the SVM model for each combination of C and gamma, with gamma values ranging from 0.1 to 0.9 and C values ranging from 1 to 9. The second table shows the validation accuracy for gamma values ranging from 0.05 to 0.13 and C values ranging from 1 to 9.

The results indicate that the performance of the SVM model is highly dependent on the values of C and gamma. In the first table, the highest accuracy of 93.95% is achieved with C=5 and gamma=0.1. In the second table, the highest accuracy of 95.36% is achieved with C=9 and gamma=0.05. It is also notable that the performance of the model generally improves as the value of gamma decreases.

		Gamma				
		0.1	**0.3**	**0.5**	**0.7**	**0.9**
C	**1**	93.83	63.05	32.88	11.17	10.26
	3	93.90	64.93	36.24	15.81	10.26
	5	93.95	64.33	35.19	15.09	10.24
	7	93.90	65.74	38.02	12.69	10.24
	9	93.93	64.98	34.26	11.88	9.428

Table 6. Validation Accuracy Grid Search Results for C and Gamma

		Gamma				
		0.05	**0.07**	**0.09**	**0.11**	**0.13**
C	**1**	95.24	94.86	94.38	93.33	91.95
	3	95.33	94.88	94.40	93.52	92.05
	5	95.31	94.90	94.40	93.52	92.14
	7	95.29	94.88	94.40	93.55	92.19
	9	**95.36**	94.88	94.45	93.55	92.12

Table 7. Validation Accuracy Grid Search Results for C and Gamma

1.3.2 Polynomial Kernel

Polynomial Kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$ where:

- x_i and x_j are two data points
- γ is a scaling factor
- r is a coefficient in the kernel
- d is the degree of the polynomial

Without Slack: Test accuracy with parameters **degree 3** , **gamma 0.001** : 91.811%

With Slack:

- **Best Validation accuracy: 95.00 with parameters: 'C': 10, 'gamma': 0.1, 'degree': 2**
- **Test accuracy with best parameters found using Grid Search : 92.6222%**

Grid Search for each degree:

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	45.97%	83.95%	90.83%	95%
	1	65%	93.88%	94.976%	95%
	10	88.88%	94.80%	**95%**	95%

Table 8. Validation Accuracy Grid Search Results for C and Gamma for **Degree=2**

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	22.88%	76.69%	94.33%	94.40%
	1	22.88%	90.97%	94.40%	94.40%
	10	25.83%	94.04%	94.40%	94.40%

Table 9. Validation Accuracy Grid Search Results for C and Gamma for **Degree=3**

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	14.47%	39.14%	93.11%	93.07%
	1	14.47%	81.26%	93.07%	93.07%
	10	14.47%	90.92%	93.07%	93.07%

Table 10. Validation Accuracy Grid Search Results for C and Gamma for **Degree=4**

1.3.3 Linear kernel

The linear kernel used in SVM is defined as:

$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j}$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ are input feature vectors. The dot product $\mathbf{x_i} \cdot \mathbf{x_j}$ is a measure of similarity between the two feature vectors. This kernel is called the linear kernel because it corresponds to a linear decision boundary in the input feature space.

Without Slack: Test accuracy with c=∞ : 90.4778%

With Slack:

- **Best Validation accuracy: 93.24 with parameters: 'C': 0.01**
- **Test accuracy with best parameters found using Grid Search : 91.47%**

Grid Search for c:

C	Accuracy
0.001	91.7381%
0.01	**93.2381%**
0.1	92.9524%
1	92.9048%
3	92.7857%
5	92.6905%

Table 11. Accuracy of SVM model for different values of C

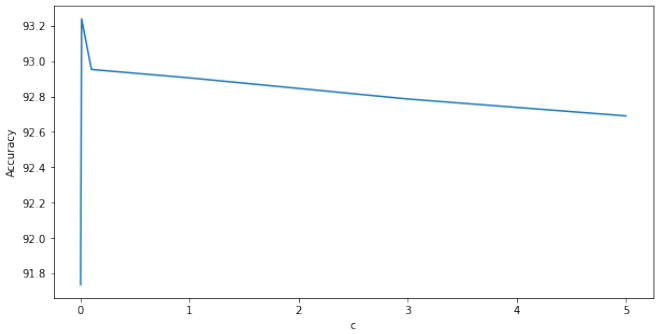


Figure 4. Grid Search plot for c and accuracy

1.4. Results for p5 in A1

1.4.1 Gaussian Kernel

Gaussian Kernel is given by $K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$

Without Slack: Test accuracy with parameter **gamma 0.01** : 88.75%

With Slack:

- **Best Validation accuracy: 91.52% with parameters: C=7, gamma=0.09**
- **Test accuracy with best parameters found using Grid Search : 90.28%**

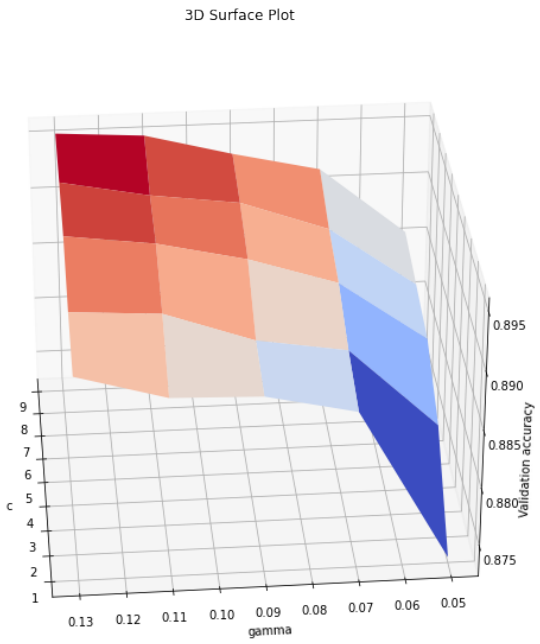


Figure 5. Grid Search plot for c,gamma and accuracy

		Gamma				
		0.1	**0.3**	**0.5**	**0.7**	**0.9**
C	**1**	88.36	89.33	89.90	90.31	90.50
	3	88.90	90.07	90.48	90.74	91.12
	5	89.17	90.24	90.71	91.00	91.21
	7	89.31	90.57	90.74	91.10	**91.45**
	9	89.35	90.47	91	91.47	91.45

Table 12. Validation Accuracy Grid Search Results for C and Gamma

		Gamma				
		0.05	**0.07**	**0.09**	**0.11**	**0.13**
C	**1**	87.29	88.02	88.38	88.48	88.57
	3	88.60	88.71	88.90	89.00	89.17
	5	88.76	88.83	89.14	89.26	89.33
	7	88.79	89.10	89.31	89.36	89.52
	9	89.00	89.14	89.40	89.50	89.57

Table 13. Validation Accuracy Grid Search Results for C and Gamma

1.4.2 Polynomial Kernel

Polynomial Kernel: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$ where:

- x_i and x_j are two data points
- γ is a scaling factor
- r is a coefficient in the kernel
- d is the degree of the polynomial

Without Slack: Test accuracy with parameters **degree 3** , **gamma 1** : 87.2889%

With Slack:

- **Best Validation accuracy: 91.29 with parameters: 'C': 10, 'gamma': 1, 'degree': 4**
- **Test accuracy with best parameters found using Grid Search : 90.0444%**

Grid Search for each degree:

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	57.21%	57.21%	73.19%	88.83%
	1	57.21%	57.21%	85.12%	89.97%
	10	57.21%	73.19%	88.83%	90.61%

Table 14. Validation Accuracy Grid Search Results for C and Gamma for **Degree=2**

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	46.59%	46.59%	50.48%	89.83%
	1	46.59%	46.59%	80.23%	90.64%
	10	46.59%	46.59%	87.45%	90.80%

Table 15. Validation Accuracy Grid Search Results for C and Gamma for **Degree=3**

		Gamma			
		0.001	**0.01**	**0.1**	**1**
C	**0.1**	36.66%	36.66%	36.66%	90.26%
	1	36.66%	36.66%	63.30%	90.76%
	10	36.66%	36.66%	83.73%	**91.29%**

Table 16. Validation Accuracy Grid Search Results for C and Gamma for **Degree=4**

1.4.3 Linear kernel

The linear kernel used in SVM is defined as:

$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j}$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ are input feature vectors. The dot product $\mathbf{x_i} \cdot \mathbf{x_j}$ is a measure of similarity between the two feature vectors. This kernel is called the linear kernel because it corresponds to a linear decision boundary in the input feature space.

Without Slack: Test accuracy with c=10^6 : 86.44%

With Slack:

- **Best Validation accuracy: 88.45 with parameters: 'C': 5**
- **Test accuracy with best parameters found using Grid Search : 87.33%**

Grid Search for c:

c	Accuracy
0.1	86.7857%
1	88.0238%
2	88.381%
3	88.381%
4	88.3571%
5	**88.4524**%
6	88.4048%
7	88.4048%
8	88.4286%
9	88.4524%
10	88.4048%
20	88.2143%

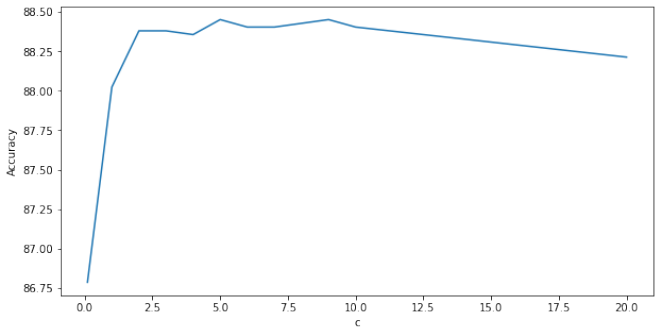


Figure 6. Grid Search plot for c and accuracy

1.5. Inferences

SVM performed poorly on p3 in A1. A reason can be the dataset may be too complex: SVM may not be able to effectively separate the classes in a very complex dataset, especially if there is a lot of overlap between the classes.

From the above results, we can observe that the accuracy without slack is slightly less than accuracy with slack.If the accuracy of SVM without slack is very slightly less than with slack, it suggests that the data being used may still be relatively easy to separate, even though it may not be perfectly separable, and there may be some noise or outliers in the data.

In SVM with slack, the algorithm allows for misclassifications in order to achieve better overall performance, but this can sometimes lead to overfitting or a decrease in performance if the amount of slack is too high. On the other hand, SVM without slack aims to find the optimal hyperplane that perfectly separates the data points without allowing any misclassifications.

When the difference in accuracy between SVM with and without slack is very small, it suggests that the optimal hyperplane that perfectly separates the data points without allowing any misclassifications is already very close to the hyperplane that allows for some misclassifications. This means that allowing for some misclassifications does not improve the performance significantly, and it is better to use the hyperplane without slack in order to avoid overfitting.

SVM performed well on MNIST. This might be because SVM is known to perform well on datasets with high-dimensional feature spaces, which is the case with MNIST. Each image in MNIST is represented by a 28x28 pixel grid, which results in 784 features (i.e., one feature for each pixel). This high number of features makes it difficult for other algorithms, such as k-nearest neighbors or decision trees, to perform well due to the "curse of dimensionality".

SVM is able to handle high-dimensional feature spaces well due to its ability to find a hyperplane that separates the data points into different classes with maximum margin. In the case of MNIST, this means that SVM is able to find a hyperplane that separates the digit images into their respective classes (i.e., 0-9) with maximum margin.

Furthermore, SVM has good generalization properties, which means that it is able to perform well on unseen test data. This is important for the MNIST dataset, which has a large number of test images that the model has not seen during training.

With PCA, The accuracy with all Kernels dropped by around 3% . In our problem the dimensions have been reduced from 784 dimensions to 10 dimnensions. While PCA can help to reduce the computational complexity of the SVM model and prevent overfitting, it can also result in the loss of important information if too few principal components are retained. This can lead to a reduction in accuracy, as the model may not have enough information to accurately distinguish between different classes.

In the case of the MNIST dataset, it is possible that the reduction in accuracy after applying PCA is due to the loss of important information about the handwritten digits. One way to mitigate this issue is to experiment with different numbers of principal components and find the optimal number that maximizes the accuracy of the SVM model.

In terms of kernel performances Gaussian Kernel had better accuracy quantitatively followed by polynomial followed by linear. In some cases though accuracies were almost similar.

It is not surprising that the Gaussian kernel performed better than the polynomial and linear kernels in terms of accuracy, as the Gaussian kernel is generally considered to be the most flexible and powerful kernel function in SVM.

The Gaussian kernel allows SVM to model complex nonlinear decision boundaries in the data by transforming the input features into a high-dimensional space, where the data may become linearly separable. This can lead to better performance on datasets that are not linearly separable in their original feature space, such as the MNIST dataset.

In contrast, the polynomial kernel is less flexible than the Gaussian kernel, as it only allows for modeling of polynomial decision boundaries. This may result in reduced accuracy on datasets where the decision boundary is highly nonlinear and cannot be accurately modeled by a polynomial function.

The linear kernel is the simplest of the three, as it assumes a linear decision boundary in the data. While it may perform well on datasets that are linearly separable, it is likely to underperform on datasets that have more complex nonlinear decision boundaries.

2. Problem 2: FLDA

FLDA is used to find a linear combination of features that maximizes the separation between classes while minimizing the variance within each class. The reduced feature space obtained through FLDA can then be used to train a linear classifier, such as logistic regression, to classify new instances.

In a multi-class classification problem, FLDA can be used to project the data onto a lower-dimensional space that captures the most important discriminatory information between the classes. The resulting reduced feature space can then be used to train a multi-class classifier, such as a support vector machine, decision tree, or neural network.

In this problem, the classifier used is the nearest mean classifier, also known as centroid classifier.The predicted class for the sample is the class whose mean is the closest to the sample. This classifier is used in the function FLDA classifier, which takes training data, training labels, test data, and the number of components to keep after performing Fisher Linear Discriminant Analysis (FLDA).

2.1. Implementation

Assuming we have K classes and N_k observations in each class k, with a total of $N = \sum_{k=1}^{K} N_k$ observations. Let $\mathbf{x}_i$ be the i-th observation, and let $\boldsymbol{\mu}_k$ be the mean vector of class k.

First, we compute the within-class scatter matrix $\mathbf{S}_W$ as follows:

$$\mathbf{S}_W = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T = \sum_{k=1}^{K} \mathbf{S}_k$$

where $\mathbf{S}_k$ is the scatter matrix for class k:

$$\mathbf{S}_k = \sum_{\mathbf{x}_i \in C_k} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$$

Next, we compute the between-class scatter matrix $\mathbf{S}_B$:

$$\mathbf{S}_B = \sum_{k=1}^{K} N_k(\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

where $\boldsymbol{\mu}$ is the overall mean vector:

$$\boldsymbol{\mu} = \frac{1}{N} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$$

The goal of FLDA is to find a linear transformation matrix $\mathbf{W}$ that maximizes the ratio of between-class scatter to within-class scatter, which can be expressed as follows:

$$\mathbf{W} = \underset{\mathbf{W}}{\operatorname{argmax}} \frac{|\mathbf{W}^T \mathbf{S}_B \mathbf{W}|}{|\mathbf{W}^T \mathbf{S}_W \mathbf{W}|}$$

The optimal solution for $\mathbf{W}$ can be obtained by solving the generalized eigenvalue problem:

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w}_i = \lambda_i \mathbf{w}_i$$

where $\mathbf{w}_i$ and λ_i are the i-th eigenvector and eigenvalue, respectively, of the matrix $\mathbf{S}_W^{-1}\mathbf{S}_B$.

The FLDA function implements Fisher's Linear Discriminant Analysis (FLDA) algorithm. FLDA is a dimensionality reduction technique that aims to maximize the separation between the classes in a dataset by projecting the data onto a lower-dimensional space. The resulting features can then be used as input to a classification algorithm.

Given an input dataset X and corresponding labels y, FLDA computes the mean and covariance of each class, as well as the total mean and scatter of the entire dataset. It then constructs two matrices: S_W, which represents the within-class scatter, and S_B, which represents the between-class scatter. The goal of FLDA is to find a linear transformation of the data that maximizes the ratio of S_B to S_W.

To accomplish this, the function computes the eigenvectors and eigenvalues of the matrix $\text{pinv}(S_W) \cdot S_B$. The top $n_{components}$ eigenvectors (i.e., those with the highest corresponding eigenvalues) are then selected as the new basis for the transformed data. The resulting transformed data is returned as X_lda, along with the corresponding eigenvectors in top_eigenvectors.

After projecting the data onto the reduced-dimensional space obtained from FLDA, the code assigns class labels to each instance in the test set based on the mean values obtained for each class. Specifically, for each instance in X_test_lda, the code calculates the Euclidean distance between the instance and the mean value of each class, and assigns the predicted class label to the mean value with the minimum distance.

2.2. Results

2.2.1 p3 in A1

Classification Accuracy on Test Data is:
57.14666666666667

Confusion Matrix is:

$$\begin{bmatrix} 1700. & 292. & 269. & 333. & 387. \\ 317. & 1866. & 253. & 290. & 314. \\ 349. & 256. & 1727. & 290. & 353. \\ 326. & 245. & 334. & 1774. & 354. \\ 394. & 303. & 369. & 400. & 1505. \end{bmatrix}$$

f1 Score of Class 1 is: 0.5604087687489698
f1 Score of Class 2 is: 0.6217927357547485
f1 Score of Class 3 is: 0.582756875316349
f1 Score of Class 4 is: 0.5797385620915034
f1 Score of Class 5 is: 0.5115567641060503

2.2.2 p4 in A1

Classification Accuracy on Test Data is:
88.49444444444444
Confusion Matrix is:

$$\begin{bmatrix} 1742. & 3. & 13. & 11. & 9. & 3. & 0. & 0. & 6. & 13. \\ 3. & 1712. & 14. & 2. & 42. & 0. & 1. & 1. & 1. & 24. \\ 2. & 0. & 1478. & 154. & 50. & 0. & 53. & 0. & 0. & 63. \\ 1. & 0. & 7. & 1730. & 52. & 0. & 5. & 0. & 1. & 4. \\ 9. & 11. & 11. & 67. & 1684. & 1. & 3. & 13. & 1. & 0. \\ 3. & 0. & 67. & 25. & 3. & 1547. & 153. & 0. & 2. & 0. \\ 10. & 0. & 181. & 137. & 12. & 302. & 1130. & 0. & 3. & 25. \\ 6. & 2. & 3. & 15. & 6. & 0. & 0. & 1669. & 48. & 51. \\ 1. & 0. & 4. & 76. & 6. & 9. & 13. & 40. & 1642. & 9. \\ 148. & 1. & 27. & 12. & 7. & 2. & 3. & 3. & 2. & 1595. \end{bmatrix}$$

F1 Score of Class 1 is: 0.9353020134228188
F1 Score of Class 2 is: 0.9702465287616889
F1 Score of Class 3 is: 0.8199722607489597
F1 Score of Class 4 is: 0.8587738893025565
F1 Score of Class 5 is: 0.917461182239172
F1 Score of Class 6 is: 0.8444323144104803
F1 Score of Class 7 is: 0.7149636191078773
F1 Score of Class 8 is: 0.9466817923993193
F1 Score of Class 9 is: 0.9366799771819738
F1 Score of Class 10 is: 0.8900669642857143

2.2.3 p5 in A1

Classification Accuracy on Test Data is:
81.74444444444444

Confusion Matrix is:

$$\begin{bmatrix} 1642 & 1 & 62 & 7 & 4 & 0 & 1 & 20 & 21 & 42 \\ 21 & 1557 & 20 & 1 & 136 & 0 & 11 & 2 & 0 & 52 \\ 2 & 2 & 1473 & 147 & 28 & 3 & 75 & 0 & 2 & 68 \\ 0 & 0 & 22 & 1703 & 58 & 3 & 4 & 4 & 1 & 5 \\ 0 & 7 & 92 & 107 & 1580 & 2 & 5 & 7 & 0 & 0 \\ 0 & 3 & 48 & 51 & 4 & 1169 & 521 & 0 & 1 & 3 \\ 10 & 0 & 185 & 203 & 6 & 311 & 1053 & 3 & 2 & 27 \\ 10 & 3 & 10 & 29 & 19 & 2 & 1 & 1586 & 86 & 54 \\ 6 & 0 & 30 & 114 & 4 & 6 & 33 & 61 & 1512 & 34 \\ 183 & 0 & 90 & 21 & 10 & 0 & 2 & 36 & 19 & 1439 \end{bmatrix}$$

F1 Score of Class 1 is: 0.8938486663037561
F1 Score of Class 2 is: 0.9232137563000297
F1 Score of Class 3 is: 0.7687891440501045
F1 Score of Class 4 is: 0.8142481472627301
F1 Score of Class 5 is: 0.8659906823787339
F1 Score of Class 6 is: 0.7093446601941747
F1 Score of Class 7 is: 0.600684540787222
F1 Score of Class 8 is: 0.9013924410343848
F1 Score of Class 9 is: 0.8780487804878049
F1 Score of Class 10 is: 0.8166855845629967

2.3. Interpretations

Inferring from the above results,

FLDA with nearest mean classifier performed well on p4 and p5 ,but performed poorly on p3.And also, we can see that before PCA the accuracy was around 89% and after PCA the accuracy dropped to 82%. This is because after performing PCA (Principal Component Analysis), the accuracy of FLDA (Fisher's Linear Discriminant Analysis) decreases due to the loss of discriminative information during dimensionality reduction. PCA projects the data onto a lower-dimensional space based on the variance of the data, without taking into account the class labels of the data. Therefore, it is possible that important discriminative information between classes may be lost during PCA.

FLDA, on the other hand, aims to project the data onto a lower-dimensional space that maximizes the separation between classes. By contrast, PCA may not necessarily maximize the separation between classes. Therefore, when the number of dimensions is further reduced using PCA before applying FLDA, it is possible that important discriminative information may be lost, leading to a decrease in accuracy.

The poor performance on p3 might be due to:

1. Non-linear separability: FLDA assumes that the data can be linearly separated into distinct classes. However, if the classes are not linearly separable, FLDA may perform poorly and produce inaccurate classification results.

2. Inadequate dimensionality reduction: FLDA reduces the dimensionality of the data by projecting it onto a lower-dimensional space. However, if the number of

dimensions is not appropriately reduced, FLDA may still produce high-dimensional data that is difficult to separate linearly, leading to poor classification performance.

3. Overfitting: FLDA can be prone to overfitting, especially when the dimensionality reduction is not appropriately regularized. Overfitting occurs when the model fits too closely to the training data and fails to generalize well to unseen data, leading to poor performance on the test set.

4. Correlated features: FLDA assumes that the features in the data are uncorrelated. If the features are highly correlated, FLDA may perform poorly as it may not be able to capture the underlying patterns in the data.

3. Problem 3: OverFitting and Regularization

The Implementation, results and Inferences are in the notebook .ipynb file.

p4: Neural Networks

The objective of this problem is to construct and experiment with various architectures of Multi-Layer Perceptron using different Activations and Loss functions and understand the working of these architectures on Kannada MNIST data (28 x 28 images of handwritten digits) (784 dims)

Data:

The given data is split into train and test datasets in 80:20 ratio

Architectures:

Config -1 : Input layer of 784 neurons followed by 1 hidden FC layer with 350 neurons. Activations – ReLU, sigmoid, linear. Loss functions – CE, MSE loss

Config -2 : Input layer of 784 neurons followed by 2 hidden FC layers with 800 and 300 neruons. Activations – ReLU, sigmoid, linear. Loss functions – CE, MSE loss

Config -3 : Input layer of 784 neurons followed by 3 hidden FC layers with 500, 250 and 100 neurons. Activations – ReLU, sigmoid, linear. Loss funcitons – CE, MSE loss

All the configurations are followed by a fully connected output layer with 10 neurons and a Softmax activation layer

Results:

Architecture	Config- 1 (CE loss function)	Config- 1 (MSE loss function)	Config- 2 (CE loss function)	Config-2 (MSE loss function)	Config- 3 (CE loss function)	Config- 3 (MSE loss function)
RelU (Train Accuracy)	100	98.58	99.98	99.64	99.68	98.51
RelU (Test Accuarcy)	98.43	97.96	98.41	98.36	98.11	98.06
Linear (Train Accuracy)	97.07	97.56	96.93	97.83	97.21	97.75
Linear (Test Accuracy)	96.22	96.88	96.08	97.02	96.94	97.29
Sigmoid (Train Accuracy)	99.84	99.66	99.79	99.03	92.04	92.94
Sigmoid (Test Accuracy)	98.28	98.16	97.88	97.47	93.55	95.25

F1 scores for each configuration with ReLU activation and CE loss are given below. *F1 scores for all the activations and loss functions with all the configurations are shown in the ipynb file*:

Config – 1:

```
f1 Score of Class  0  is:  0.9933554817275748
f1 Score of Class  1  is:  0.9916874480465502
f1 Score of Class  2  is:  0.9864141622066693
f1 Score of Class  3  is:  0.9896049896049897
f1 Score of Class  4  is:  0.9912609238451936
f1 Score of Class  5  is:  0.9692307692307691
f1 Score of Class  6  is:  0.9655751469353484
f1 Score of Class  7  is:  0.9665979381443299
f1 Score of Class  8  is:  0.9941908713692945
f1 Score of Class  9  is:  0.9949832775919732
```

Config – 2:

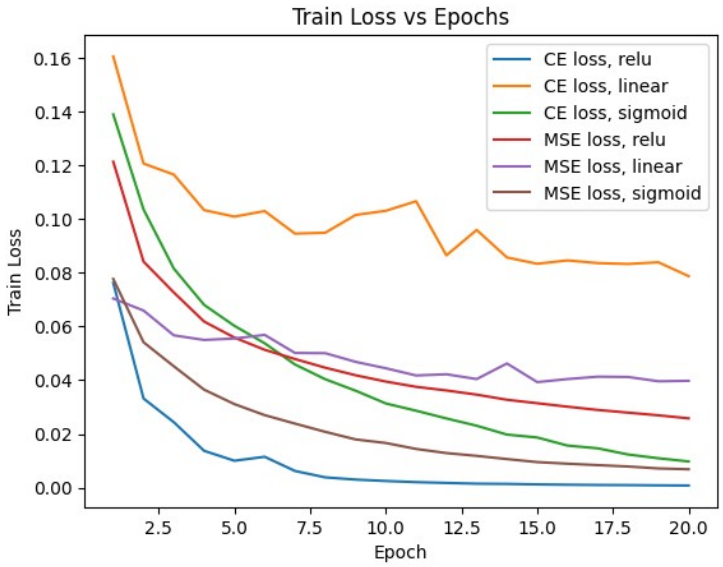
```
f1 Score of Class  0  is:  0.9896308585649108
f1 Score of Class  1  is:  0.9912463526469362
f1 Score of Class  2  is:  0.987205943045811
f1 Score of Class  3  is:  0.9891936824605153
f1 Score of Class  4  is:  0.9920997920997922
f1 Score of Class  5  is:  0.973594548551959
f1 Score of Class  6  is:  0.961864406779661
f1 Score of Class  7  is:  0.96359918200409
f1 Score of Class  8  is:  0.9962640099626401
f1 Score of Class  9  is:  0.9958228905597326
```

Config – 3:

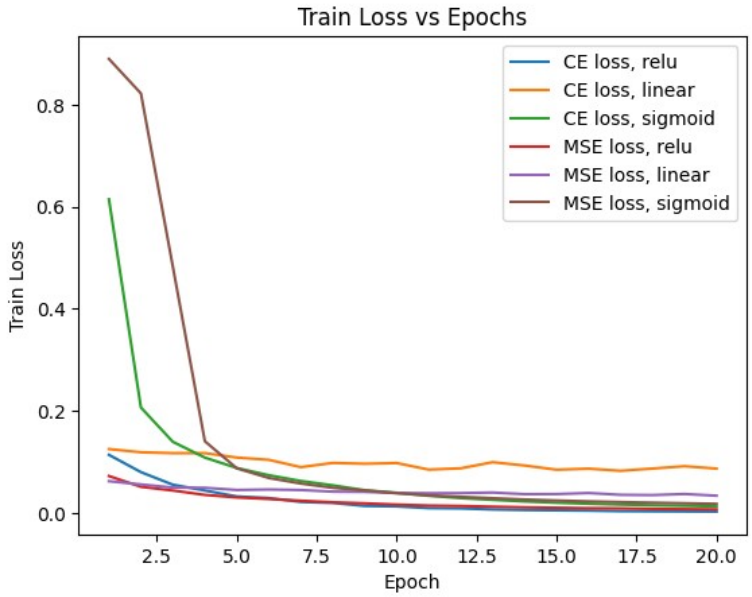
```
f1 Score of Class  0  is:  0.9892116182572614
f1 Score of Class  1  is:  0.9912536443148688
f1 Score of Class  2  is:  0.9810699588477366
f1 Score of Class  3  is:  0.9822387443205287
f1 Score of Class  4  is:  0.9908713692946057
f1 Score of Class  5  is:  0.9674935842600514
f1 Score of Class  6  is:  0.9616195698017713
f1 Score of Class  7  is:  0.9557668458040512
f1 Score of Class  8  is:  0.9958437240232751
f1 Score of Class  9  is:  0.9949874686716792
```

Plots comparing the behavior training behaviour of the architectures with various loss funtions and activations are shown below:

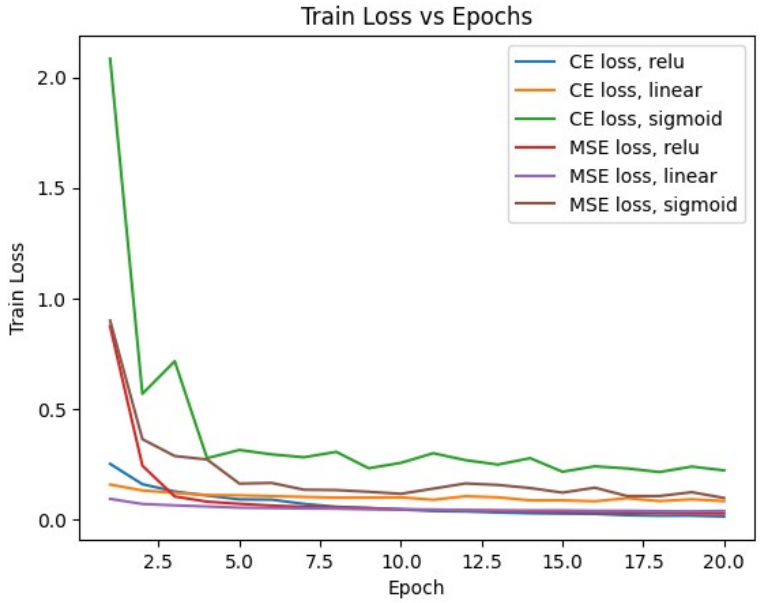
Config – 1:



Config – 2:



Config – 3:



Observations:

- Multiple learning rates were tried out for all the configurations with all loss functions and all activations. It is observed that sigmoid activation and linear activation are highly sensitive to the learning rates used in gradiend descnet and hence the model doesnt train appropriately with many learning rates and the plots that are shown above are with a particular learning rate for each case. But RelU activation perfomed well with many learning rates and the model is very easy to train.
- This could be because of the vanishing gradient problem of sigmoid activation. As the no. of neurons/ layers that are trained increase, the gradients becomes very small and the training

will become very slow in case of deep networks. Sigmoid activation, eventhough non linear, saturates at extremes, which can limit its ability to learn complex patterns.

- When linear activation is used, its similar to using a logistic regressor (due to softmax layer in the end). Hence it has limited expressive power and the model cannot learn complex patterns of the data. Linear activation can also suffer from gradient vanishing and exploding problem similar to sigmoid activation
- During training, in some cases, slight increase in the train data loss is observed at some epochs. This could be either because of high or low learning rate as the weights can overshoot the optimum or the model can get stuck in local minima and the loss can increase over time.
- Overall, it is observed that for this particular problem, all the 3 architectures performed well with any activation and loss function, *but only with proper tuning of learning rate* and RelU activation gives better performance over Sigmoid and Linear activation, in general.

p5: CNN

The objective of this problem is to construct and experiment with various architectures of CNN and study their behaviour on Kannada MNIST data (28 x 28 images of handwritten digits) (784 dims)

Data:

The given data is split into train and test data in 20:80 ratio. This is because the CNNs take a lot longer time to train with dedicated compute power.

Architectures:

Config – 1:

- 1 input layer with 784 neurons
- 1 conv layer with kernel size 3 x 3 and depth 1
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer

Config – 2:

- 1 input layer with 784 neurons
- 1 conv layer with kernel size 5 x 5 and depth 1
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer

Config – 3:

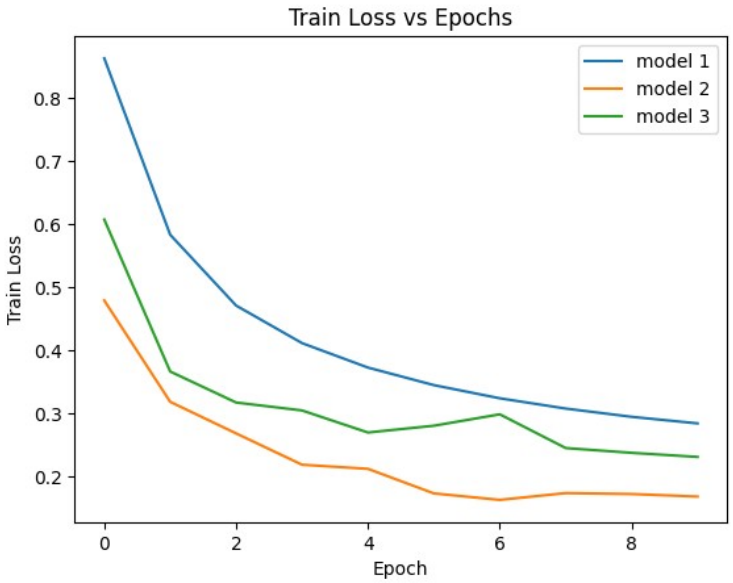
- 1 input layer with 784 neurons
- 1 conv layer with kernel size 3 x 3 and depth 2

- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer
- 1 conv layer with kernel size 3 x 3 and depth 3
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer

In all configurations, a fully connected layer of appropriate neurons is added to the hidden layers and a softmax activation layer is added in the end. Loss function used is Cross Entropy loss

Results and Observations:

Architecture	Train Accuracy	Test Accuracy
Config – 1	92.27	88.92
Config – 2	95.13	91.4
Config – 3	92.58	90.15



- The above plot shows the behaviour of 3 configurations trained for 10 epochs. We can see from the table that all the architechtures performed well. The models that are used can fit the given data very well.
- It is observed that, even for 20 % of the train data and with batch size 256, the networks took a lot of time to train. This is because the process of training a convolutional layer involves multiple operations, including matrix multiplication, activation functions, and pooling operations. Each of these operations requires significant computation, and the overall complexity of the model can add up quickly
- The training and convergence is very sensitive to learning rate used during gradient descent. It is observed that for many learning rates, the train accuracy came out to be very less or the train loss increased with epochs. Even for the learning rates that allowed convergence, after a few epochs, since proper decay was not used, the train loss increased while training sometimes.

p6: CNN (L2 regularizer, overfitting, earlystopping)

The objective of this problem is to construct a big enough CNN which can overfit the Kannada MNIST data and observe the effects of early stopping, L2 regularization and the effect of noise in input data

Data:

For early stopping, the given data is split into train, test and val datasets in 2:96:2 ratio. For overfitting, effect of L2 regularizer and the effect of input noise, the given data is split into train, test and val datasets in 0.1:98.9:1.

Architectures:

For early stopping:

- 1 input layer with 784 neurons
- 1 conv layer with kernel size 3 x 3 and depth 2
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer
- 1 conv layer with kernel size 3 x 3 and depth 3
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer

For overfitting, L2 regularizer and others:

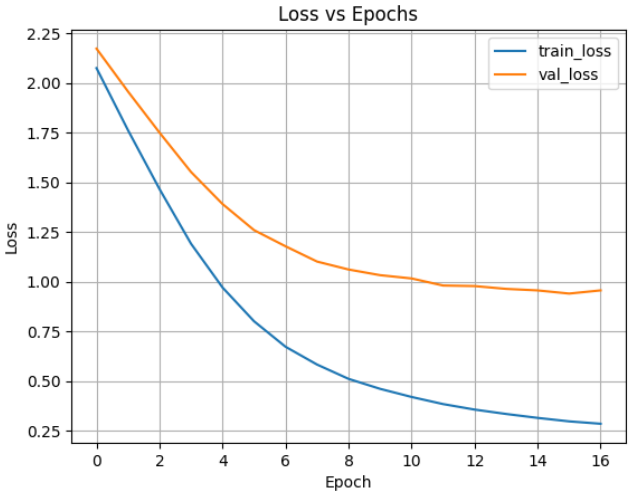
- 1 input layer with 784 neurons
- 1 conv layer with kernel size 4 x 4 and depth 2
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer
- 1 conv layer with kernel size 6 x 6 and depth 3
- 1 Max Pooling layer with kernel size 2 x 2
- 1 RelU Activation layer

In all configurations, a fully connected layer of appropriate neurons is added to the hidden layers and a softmax activation layer is added in the end. Loss function used is Cross Entropy loss

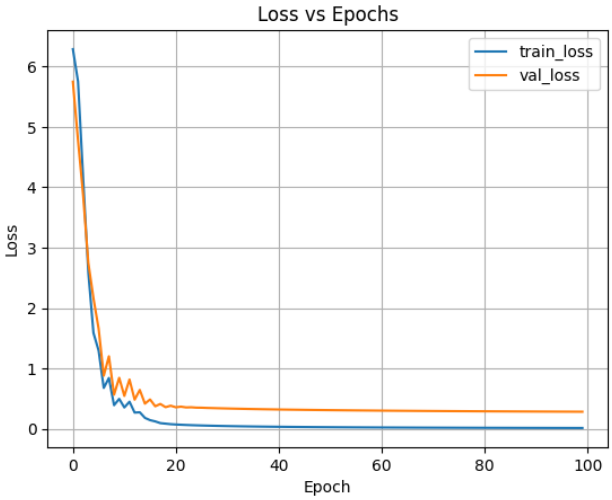
Results and Observations:

	Early stopping	Overfitting	L2 Regularizer	Noisy input
Train Accuracy	92.92	100	100	100
Val Accuracy	70.5	9.7	9.5	8.33
Test Accuracy	73.34	73.32	68.1	70.81

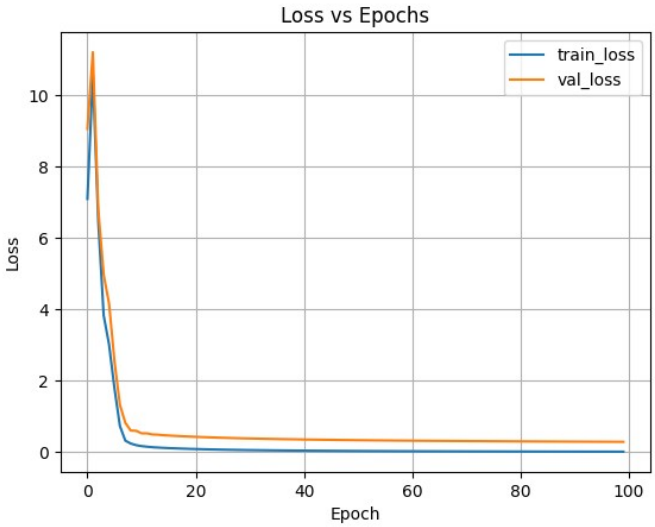
Early Stopping:



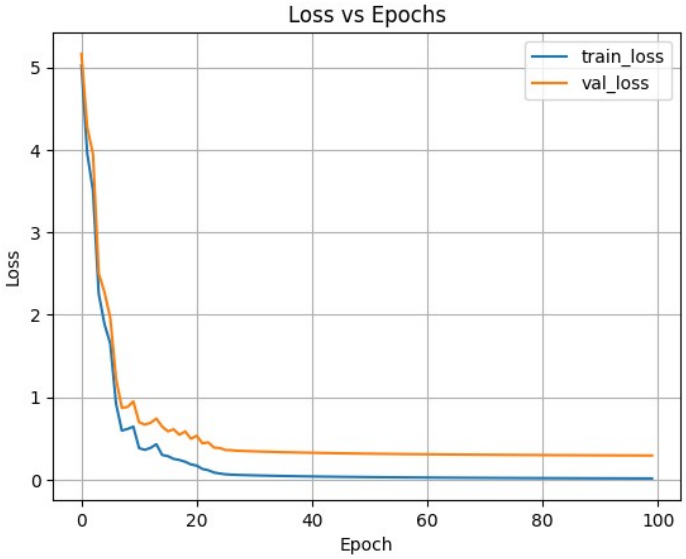
Overfitting:



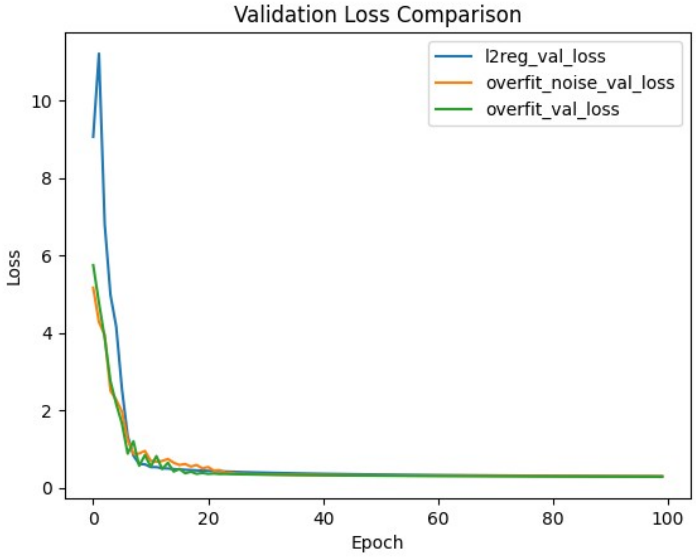
L2 Regularizer:



Noisy Input:



Validation loss comparison:



- The architecture that is used took a lot of time for training even for a very low dataset of ~ 1200 train and 1200 val dataset during early stopping. The validation loss increased at 17th epoch and the training stopped.
- Many such architectures were used along with various learning rates, but still couldn't properly overfit the data even after 100s of epochs. Training accuracy reached 100% at this point. Due to limited compute power, more complex architectures were not used. Also it is observed that for the same data and same architecture. This could be because of the random weights initialization and also the randomness involved in selection of batches during SGD step.
- L2 Regularizer and Noisy input image has the same training behaviour as shown in the plot. This is expected because Noisy input is equivalent to imposing L2 Regularizer

p7: Neural Networks (PCA_MNIST DATA)

The objective of this problem is to construct and experiment with various architectures of Multi-Layer Perceptron using different Activations and Loss functions and understand the working of these architectures on PCA_MNIST data (10 dims)

Data:

The given data is split into train and test data in 80:20 ratio

Architectures:

Config -1 : Input layer of 10 neurons followed by 1 hidden FC layer with 350 neurons. Activations – ReLU, sigmoid, linear. Loss functions – CE, MSE loss

Config -2 : Input layer of 10 neurons followed by 2 hidden FC layers with 800 and 300 neruons. Activations – ReLU, sigmoid, linear. Loss functions – CE, MSE loss

Config -3 : Input layer of 784 neurons followed by 3 hidden FC layers with 500, 250 and 100 neurons. Activations – ReLU, sigmoid, linear. Loss funcitons – CE, MSE loss

All the configurations are followed by a fully connected output layer with 10 neurons and a Softmax activation layer

Results:

Architecture	Config- 1 (CE loss function)	Config- 1 (MSE loss function)	Config- 2 (CE loss function)	Config-2 (MSE loss function)	Config- 3 (CE loss function)	Config- 3 (MSE loss function)
RelU (Train Accuracy)	96.15	96.07	96.5	96.24	95.85	95.16
RelU (Test Accuarcy)	97.03	97.39	97.31	97.15	97.42	97.09
Linear (Train Accuracy)	89.28	89.64	89.35	89.7	89.21	88.82
Linear (Test Accuracy)	92.99	93.26	93.08	93.2	92.97	92.78
Sigmoid (Train Accuracy)	93.86	93.48	95.59	94.89	95.71	95.07
Sigmoid (Test Accuracy)	96.22	96.13	96.33	96.61	96.74	96.52

F1 scores for each configuration with ReLU activation and CE loss are given below. *F1 scores for all the activations and loss functions with all the configurations are shown in the ipynb file:*

Config – 1:

```
f1 Score of Class  0  is:  0.974190905366653
f1 Score of Class  1  is:  0.9690021231422505
f1 Score of Class  2  is:  0.9908256880733944
f1 Score of Class  3  is:  0.9672667757774142
f1 Score of Class  4  is:  0.9833610648918469
f1 Score of Class  5  is:  0.9776465626318008
f1 Score of Class  6  is:  0.942857142857143
f1 Score of Class  7  is:  0.9275961936284651
f1 Score of Class  8  is:  0.9863127333056824
f1 Score of Class  9  is:  0.9836065573770492
```

Config – 2:

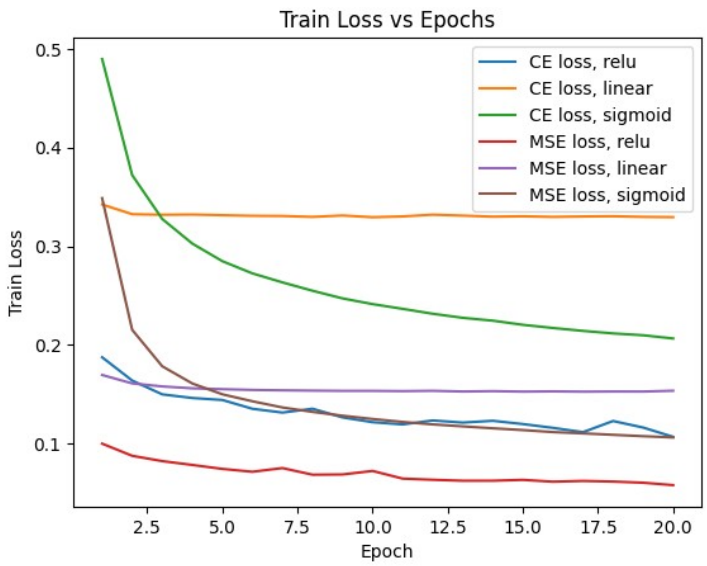
```
f1 Score of Class  0  is:  0.9733715690290864
f1 Score of Class  1  is:  0.9720812182741116
f1 Score of Class  2  is:  0.9941569282136896
f1 Score of Class  3  is:  0.976878612716763
f1 Score of Class  4  is:  0.9816971713810315
f1 Score of Class  5  is:  0.9772535804549284
f1 Score of Class  6  is:  0.9471933471933472
f1 Score of Class  7  is:  0.9396587598834789
f1 Score of Class  8  is:  0.9879015435961618
f1 Score of Class  9  is:  0.9807852965747702
```

Config – 3:

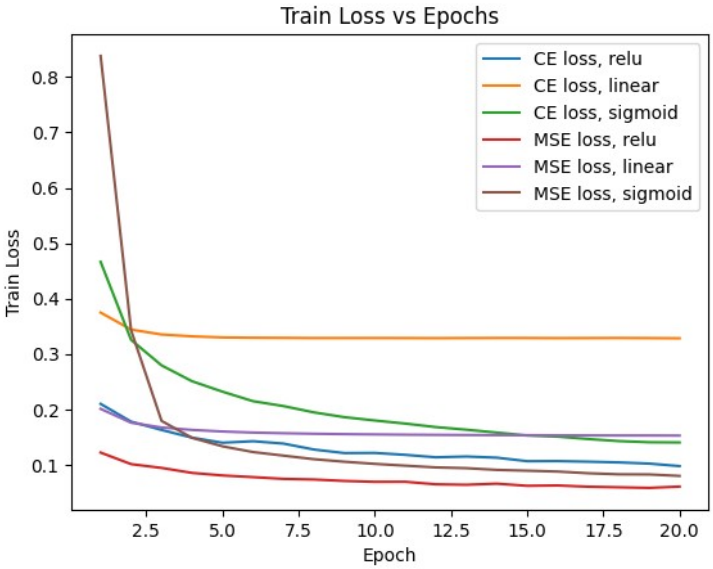
```
f1 Score of Class  0  is:  0.980865224625624
f1 Score of Class  1  is:  0.9807852965747702
f1 Score of Class  2  is:  0.9928900041823504
f1 Score of Class  3  is:  0.967609676096761
f1 Score of Class  4  is:  0.9829804898298049
f1 Score of Class  5  is:  0.9790794979079498
f1 Score of Class  6  is:  0.9506984387838949
f1 Score of Class  7  is:  0.9358266043348915
f1 Score of Class  8  is:  0.9871208973826341
f1 Score of Class  9  is:  0.9844472467423288
```

Plots comparing the behavior training behaviour of the architectures with various loss funtions and activations are shown below:

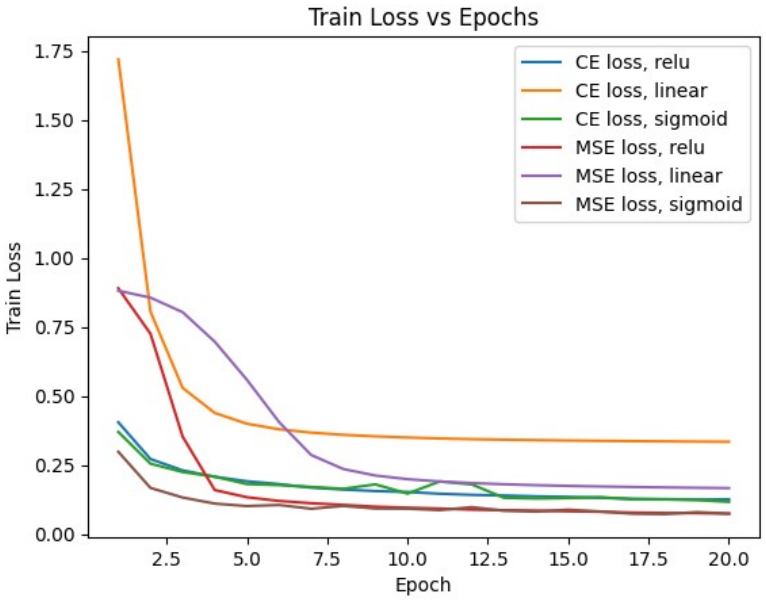
Config - 1 :



Config – 2:



Config – 3:



Observations:

- The observations for this problem are similar to that of p4. But since the dimensions of the input data are reduced, the training process is very fast but we can see that almost all the accuracies decreased because of the loss of information due to PCA. The network may not have access to all the relevant features and patterns in the data