Result and Analysis

Jin Hyun Park, UIN: 633001823

3-a

* In class Kernel\_LR, bias of nn.Linear is set to False.
* After hyperparameter tuning with different learning rate, epoch, batch size, and sigma, the best hyperparameters that I have found is the following
  + Learning rate = 0.01, epoch = 100, batch size = 32, and sigma = 1
* Test accuracy = 0.987 (98.7%)

3-b

* In class RBF, bias of nn.Linear is set to False
* After hyperparameter tuning with different hidden dimension size, learning rate, epoch, batch size, and sigma, the best hyperparameters that I have found is the following
  + Hidden dim size = 32, learning rate = 0.01, epoch = 100, batch size = 128, and sigma = 1
* Test accuracy = 0.985 (98.5%)

3-c

* In class FFN, bias of nn.Linear is set to True
* After hyperparameter tuning with different hidden dimension size, learning rate, epoch, and batch size, the best hyperparameters that I have is the following:
  + Hidden dim size = 32, learning rate = 0.01, epoch = 100, batch size = 128
* Test accuracy = 0.989 (98.9%)

Summary of Question 3

|  |  |  |
| --- | --- | --- |
|  | Hidden dimension size | Test accuracy |
| Kernel LR | # of training samples | 98.7% |
| RBF | 32 | 98.5% |
| FFN | 32 | 98.9% |

* For comparison, the hidden layer dimension size is set to 32 for both RBF and FFN.
* RBF achieved similar test accuracy to Kernel LR (only 0.2% difference). This implies that RBF can perform similarly to Kernel LR with fewer parameters.
* FFN performs slightly better than RBF (only 0.4% difference). This implies that RBF can perform similarly to FFN with fewer parameters. This is possible because we used the K-means algorithm to find centroids.

4-b

|  |  |
| --- | --- |
| # of principal component (p) | Reconstruction Error (RE) |
| 32 | 129.37 |
| 64 | 85.82 |
| 128 | 45.63 |

4-d

|  |  |
| --- | --- |
| # of hidden representation (d) | Reconstruction Error (RE) |
| 32 | 132.70 |
| 64 | 86.63 |
| 128 | 46.60 |

4-e

|  |  |  |
| --- | --- | --- |
| p=d | RE of PCA | RE of AE |
| 32 | 129.37 | 132.70 |
| 64 | 85.82 | 86.63 |
| 128 | 45.63 | 46.60 |

* The reconstruction error of PCA is slightly better than AE (Auto Encoder) but the differences are very small. This is because we did not use a non-linear activation function. With a linear activation function, PCA and AE basically do the same thing.

4-f

|  |  |  |
| --- | --- | --- |
| p=d | frobeniu\_norm\_error(G, W) | frobeniu\_norm\_error(GT.G, WT.W) |
| 32 | 8.11 | 0.17 |
| 64 | 11.44 | 0.03 |
| 128 | 15.92 | 0.02 |

Note 1: It is represented as G.GT and W.WT instead of GT.G and WT.W in the following URL: [1] <http://people.tamu.edu/~sji/classes/PCA.pdf>.

Note 2: GT.G and WT.W are the same as G.GT and W.WT in my code implementation.

According to [1], a comparison of frobeniu\_norm\_error between GT.G and WT.W is a better approach than a comparison of frobeniu\_norm\_error between G and W. Unsurprisingly, it turned out that the errors were significantly reduced.

4-g

|  |  |  |
| --- | --- | --- |
| p=d | Shared weights | Non-shared weights |
| 32 | 132.70 | 130.28 |
| 64 | 86.63 | 86.43 |
| 128 | 46.60 | 46.76 |

The result of shared weights and non-shared weights is very similar. The decoding part of the network essentially learns to mirror the encoder part for the non-shared weights model.

4-h

* After hyperparameter tuning with different epochs and batch sizes, the best hyperparameters that I have found is the following (*d* is set to 64).
  + Batch size = 64 and epoch = 3000
* I used the following network structure:
  + (Encoder) layer 1 🡪 ReLU 🡪 (Encoder) layer 2 🡪 [Hidden Representation] 🡪 (Decoder) layer 3 🡪 ReLU 🡪 (Decoder) layer 4
  + For more details, please see the following two figures

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

|  |  |
| --- | --- |
|  | Reconstruction error (*d*=64) |
| PCA | 85.82 |
| Auto Encoder with shared weights | 86.63 |
| Auto Encoder with non-shared weights | 86.43 |
| Auto encoder with non-linearity and more layers | 70.67 |

The Auto Encoder with non-linearity (= non-linear activation function) and more layers outperformed the others with respect to reconstruction error. This is because it (= the last method, 4-h) becomes **manifold learning** which is an approach to non-linear dimensionality reduction. I strongly believe that with more layers and trying out different combinations of hyperparameters (such as activation functions and batch size), it will be possible to get a lower reconstruction error.