University at Buffalo
School of Engineering and Applied Sciences

CSE 574 – Intro to Machine Learning
Fall 2022

Assignment #2

Prepared by

Antarpreet Kaur - 50486342
Kamala Krishnan - 50478764
Sakshamdeep Singh - 50475857

November 16, 2022

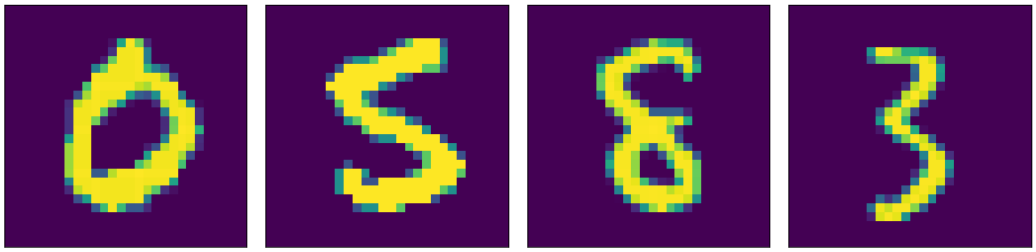
TABLE OF CONTENTS

1. Datasets

We have the following two datasets.

d. MNIST dataset

- The dataset consists of 60,000 training samples which we divide into 50,000 training samples and 10,000 validation samples. The dataset also has 10,000 test samples. Each data sample is a 28*28 pixels image.
- Visualizing a few data samples:



- We load the data and preprocess it so that we only take relevant features into account.
- There are a total of 784 features out of which we only take **717** into account.
- We have a neural network model with a single hidden layer.

e. CelebA Dataset

- CelebA dataset is a large-scale face attributes dataset with more than 200K celebrity images, each with 40 attribute annotations.
- We have a subset of the data with a total of 26407 samples. Each sample has 2376 feature columns.
- Each sample image is a 54*44 pixels matrix.
- Each sample is associated with a label value which signifies whether a celebrity is wearing glasses in the image.
- Visualizing a few data samples:

2. Scripts

a. nnScript.py

- In the nnScript.py, we build a neural network with a single hidden layer and train it on **MNIST** dataset.
- We change the hyperparameters to find the ones that best fit the validation data.

b. facennScript.py

- In this script, we build the same neural network with single hidden layer, but we train it on the **CelebA** dataset.
- Hyperparameters:
  i. Number of hidden nodes = 256
  ii. Regularization constant = 10

c. deepnnScript.py

- In this script, we build 5 neural network models with 1, 2, 3, 4, and 5 layers respectively. We train all the models on **CelebA** dataset.
- We also train the single layer model on **MNIST** dataset to compare with CNN model.
- Hyperparameters:
  i. Number of training iterations: 1, 9, 90, 900
  ii. Learning rate: 1e-4
  iii. Train batch size: 64
  iv. Number of nodes in layer 1 = 256
  v. Number of nodes in layer 2 = 128
  vi. Number of nodes in layer 3 = 64
  vii. Number of nodes in layer 4 = 32
  viii. Number of nodes in layer 5 = 16

d. cnnScript.py

- In this script, we have built a CNN model with two convolutional layers and one fully connected layer. We train the model on **MNIST** dataset.
- Hyperparameters:
  i. Number of training iterations: 1, 9, 90, 900
  ii. Learning rate: 1e-4
  iii. Train batch size: 64

3. Choosing the hyper-parameter for Neural Network

- For our first neural network model in nnScript.py, we have the following two hyperparameters:
    - Number of nodes in hidden layer: [7, 14, 21, 28, 35, **42**, 49]
    - Regularization Constant, λ: [0, **10**, 20, 30, 40, 50, 60]

- *We train our model on all the sets of hyperparameters and check the validation set accuracy. The model and hyperparameters with the highest accuracy on the validation set are chosen.*

- In our case, we get the best validation set accuracy using 42 hidden nodes and a regularization constant of 10. The validation set accuracy achieved is 95.10 %. Using these hyperparameters, the test set accuracy is 95.09%.

- **Optimal λ = 10**
- **Optimal number of hidden nodes = 42**

This relation of hyperparameters with accuracy and training time is shown in the following four graphs.

a. Graphs

i. Accuracy vs number of hidden nodes for optimal lambda, λ = 10

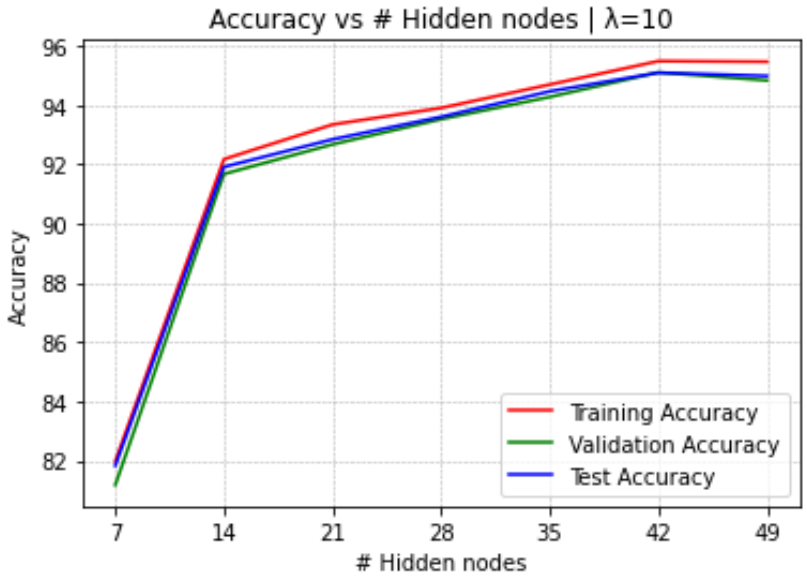


Figure 1: For a fixed regularization constant, λ, as the number of nodes in hidden layers increases, the validation set accuracy increases till a certain point. After that, the model starts to overfit, and the validation set accuracy decreases.

ii. Accuracy vs λ for the optimal number of hidden nodes = 42

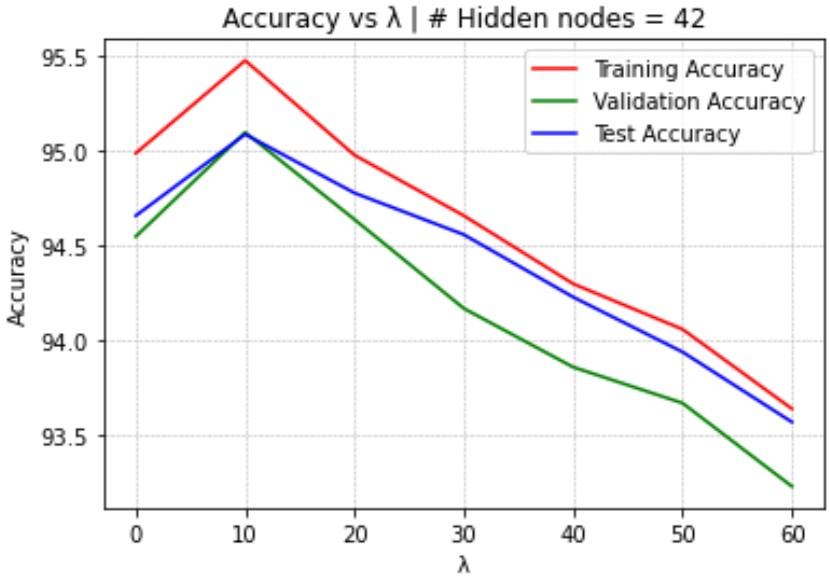


Figure 2: For a fixed number of hidden nodes, as we increase the regularization constant, the accuracy increases by preventing overfitting but after some time the regularization starts to inhibit the model's ability to learn.

iii. Training time vs number of hidden nodes for optimal, λ = 10

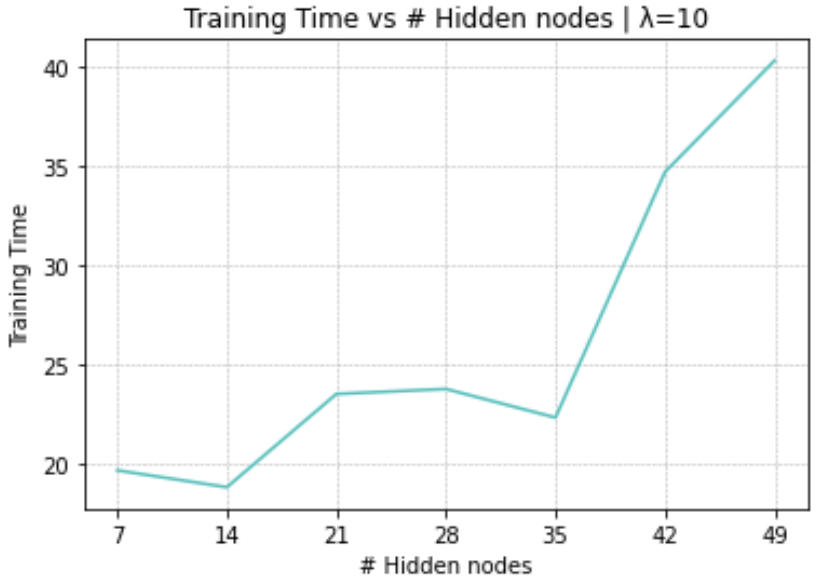


Figure 3: For a fixed regularization constant, as we increase the number of hidden nodes, we increase the total number of learning parameters thereby increasing the training time

iv. Training time vs λ for the optimal number of hidden nodes = 42

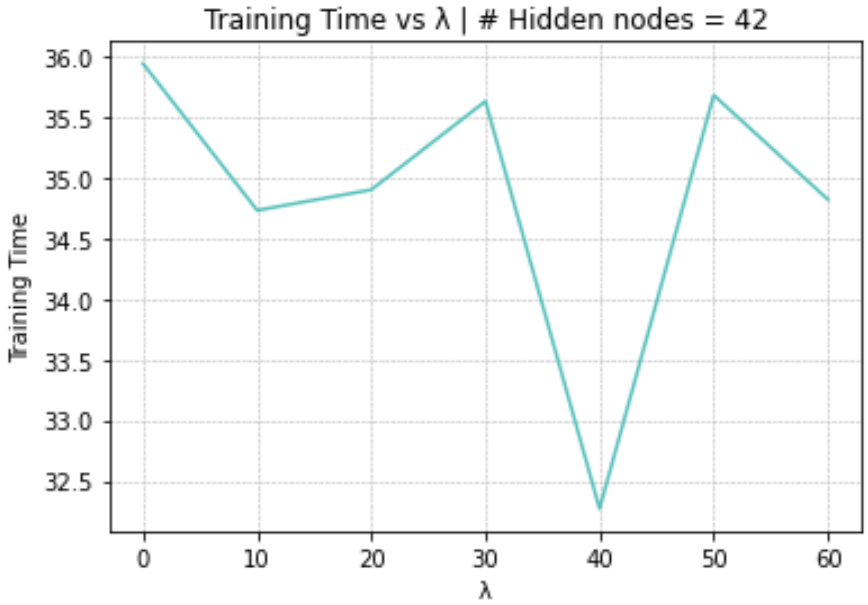


Figure 4: For a fixed number of hidden nodes, by changing the value of the regularization constant, we don't change the total number of learning parameters. Hence, the training time is more or less the same for a fixed number of hidden nodes.

b. Observations:

- As we increase the number of hidden nodes,
  - The validation accuracy increases till a certain level
  - After that, the model starts overfitting on the training data
  - The model training time increases

- As we increase the value of the regularization constant,
  - The model is prevented from overfitting
  - But a huge regularization constant also inhibits the model's ability to learn
  - The model training time is not affected

4. **Accuracy of classification method on the handwritten digits test data (nnScript.py)**

    - Optimal Hyperparameters:
        - Number of hidden nodes = 42
        - Regularization constant, $\lambda = 10$

    - Training Set Accuracy: 95.48%

    - Validation Set Accuracy: 95.10%

    - Test Set Accuracy: 95.09%

5. **Accuracy of classification method on the CelebA data set (facennScript.py)**

    - For the facennScript.py, we use a neural network with a single hidden layer with 256 nodes. The value of the regularization constant, λ, is 10. Using these hyperparameters we get the following accuracies.

    - Training set Accuracy: 83.36%

    - Validation set Accuracy: 81.83%

    - Test set Accuracy: 83.57%

    - Training Time: 1m 16s

6. **Comparison of neural network with a deep neural network (using PyTorch) in terms of accuracy and training time (deepnnScript.py)**

- **Hyperparameters:**
  - learning_rate = 0.0001
  - training_epochs = 50
  - batch_size = 100

# Hidden Layers	Model	Test Accuracy (in %)	Average Loss	Training Time
1 (256)	Neural Network	81.83	-	1m 16s
1 (256)	PyTorch Deep NN	70.4	0.604	1m 15s
2 (256, 128)	PyTorch Deep NN	67.9	0.671	1m 21s
3 (256, 128, 64)	PyTorch Deep NN	62.9	0.682	1m 28s
4 (256, 128, 64, 32)	PyTorch Deep NN	50.0	0.691	1m 26s
5 (256, 128, 64, 32, 16)	PyTorch Deep NN	50.0	0.693	1m 29s

- **Observations:**

a. Single layer neural network performs better than the PyTorch Deep neural network.
b. It is because the deep NN is overfitting the training data.
c. For deep NN, as the number of hidden layers increases,
   i. Training time increases to learn a greater number of weight parameters.
   ii. Test Accuracy decreases due to overfitting.
   iii. Average cross-entropy loss increases due to overfitting.

7. **Results from the convolutional neural network in terms of accuracy and training time (also comparisons between CNN and DNN models)**

- **CNN Model**
  - **a.** The model has 2 convolutional layers, 2 max pooling and 2 fully connected linear layers.
  - **b.** The total number of trainable parameters = 242,062.
  - **c.** Below is the architecture of our CNN model.

```
==========================================================
Layer (type:depth-idx)          Output Shape          Param #
==========================================================
├─Conv2d: 1-1                  [16, 28, 28]          416
├─MaxPool2d: 1-2               [16, 14, 14]          --
├─Conv2d: 1-3                  [36, 14, 14]          14,436
├─MaxPool2d: 1-4               [36, 7, 7]            --
├─Linear: 1-5                  [128]                 225,920
├─Linear: 1-6                  [10]                  1,290
==========================================================
Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0
Total mult-adds (M): 3.36
==========================================================
```

- **Deep Neural Network model**
  - a. The Deep NN model has two fully connected linear layers.
  - b. There are a total of 203,530 training parameters.
  - **c.** Below is the architecture of the DNN model.

```
==========================================================
Layer (type:depth-idx)          Output Shape          Param #
==========================================================
├─Linear: 1-1                  [1, 256]              200,960
├─Linear: 1-2                  [10]                  2570
==========================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
Total mult-adds (M): 0.24
==========================================================
```

Model Performance:

- Both models have similar number of training parameters ~ 220,000
- Hyperparameters are equal for both the models:
  a. Learning rate = 1e-4
  b. Train batch size = 64
  c. Test batch size = 256

CNN Model:

# Iterations	Accuracy (in %)	Average Loss	Training Time
1	94.0	0.204	9s
9	98.7	0.041	1m 24s
90	99.1	0.039	15m 54s
900	98.8	0.166	2h 40m 10s

DNN Model:

# Iterations	Accuracy (in %)	Average Loss	Training Time
1	10.1	2.309	8s
9	11.3	2.300	1m 13s
90	86.2	0.614	11m 12s
900	92.8	0.250	1h 54m 35s

- **Observations:**
  i. On the MNIST dataset, we see that the CNN model has higher accuracy than the simple neural network.
  ii. Even though both the DNN and CNN model have comparatively similar number of parameters to learn, CNN outperforms the DNN model by a lot.
  iii. CNN model converges much faster than the DNN model.
  iv. Convolutional neural networks take the **spatial structure of the image** into account, thereby learning good features like edges and corners. The model is therefore better at discriminating between two images based on the features that it learned which leads to higher accuracy on the unseen data than the other models.
  v. After a certain number of iterations, both models start to overfit on the training data and hence does not provide any increase in validation set accuracy.
  vi. Since both models have similar number of training parameters, the training time for same number of iterations is similar.

- **Sample confusion matrix after 1 iteration:**

```
[[ 958    0    4    0    0    2    9    2    5    0]
 [   0 1114    4    2    0    1    3    0   11    0]
 [  12    2  961   10   12    0    6   12   17    0]
 [   3    2   18  931    0   10    0   15   26    5]
 [   1    3    5    0  925    0   11    1    2   34]
 [   8    4    4   29    9  786   18    2   25    7]
 [   8    3    2    0    9    8  925    1    2    0]
 [   1    9   31    3    4    0    0  945    2   33]
 [   8    4    5   16   11    8   12   11  889   10]
 [  11    5    6   10   22    3    0   19    8  925]]
```

- **Sample confusion matrix after 1 iteration (graphically):**

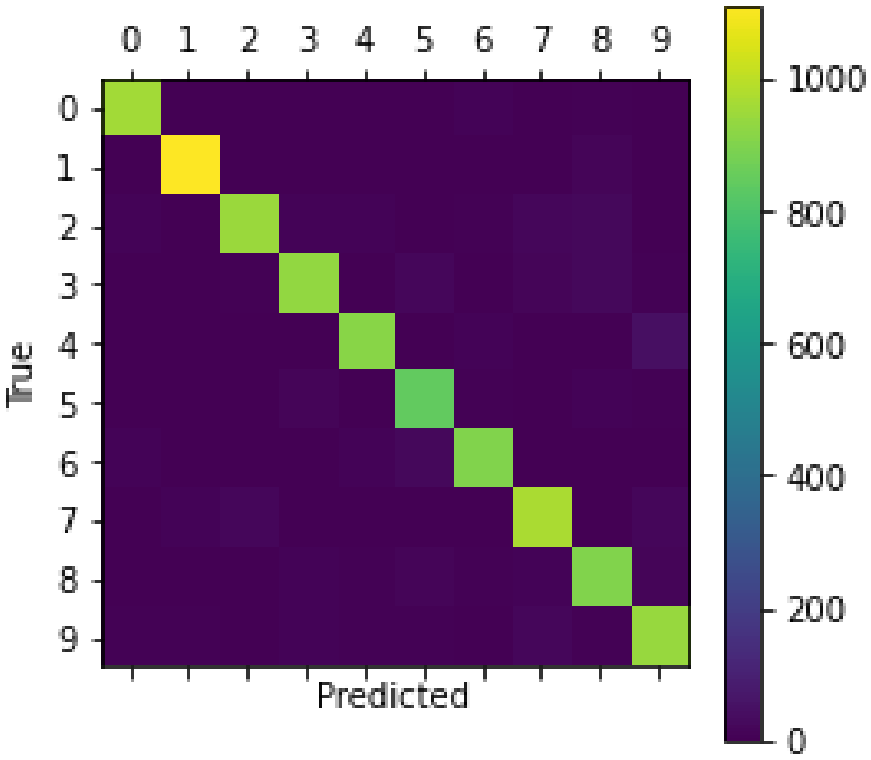
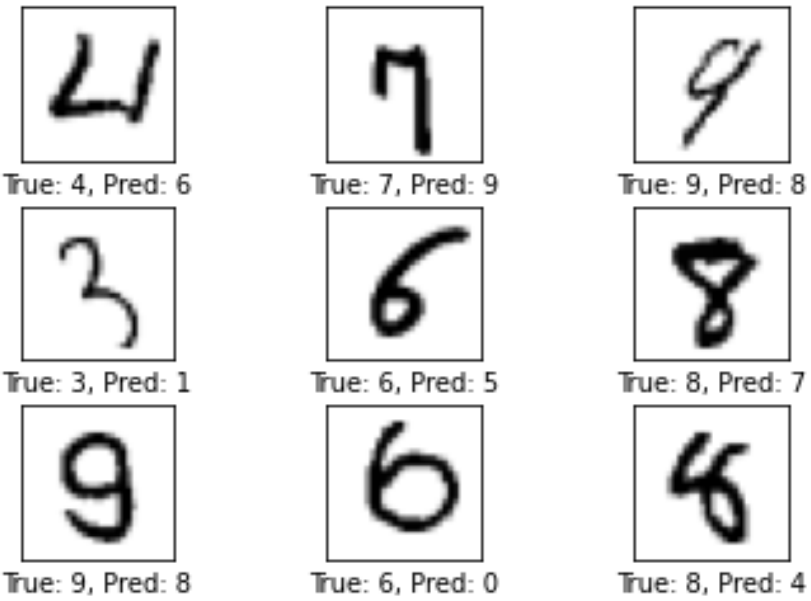


Figure 5: Confusion matrix on MNIST dataset

- **Example errors:**



~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~