Programming Assignment – 2 Report

Group – 22 (CSE 574)

Members:

1) Tejas Kalsait – 50483754
2) Tilak Sharma – 50479412
3) Hrishikesh Kakkad – 50466654

Table of Content

1) Snapshots of Functions preprocess(), sigmoid(), nnObjFunction(), nnPredict()

1) Preprocess() function

```
mat = loadmat('mnist_all.mat')  # loads the MAT object as a Dictionary

train_data = []
test_data = []
train_label = []
test_label = []
for i in range(10):
    train_i = mat['train' + str(i)]
    test_i = mat['test' + str(i)]
    for ex in train_i:
        train_data.append(ex)
        train_label.append(i)
    for test_ex in test_i:
        test_data.append(test_ex)
        test_label.append(i)

train_data = np.array(train_data) / 255
# train_data = train_data[:, ~np.all(train_data[1:] == train_data[:-1], axis=0)]
test_data = np.array(test_data) / 255
# test_data = test_data[:, ~np.all(test_data[1:] == test_data[:-1], axis=0)]
train_label = np.array(train_label)
test_label = np.array(test_label)

train_label = np.reshape(train_label, (len(train_label), 1))
test_label = np.reshape(test_label, (len(test_label), 1))

train_with_label = np.hstack([train_data, train_label])
test_with_label = np.hstack([test_data, test_label])

np.random.shuffle(train_with_label)
np.random.shuffle(test_with_label)

validation_data = train_with_label[-10000:, :-1]
validation_label = train_with_label[-10000:, -1].reshape(10000, 1)

train_data = train_with_label[:50000, :-1]
train_label = train_with_label[:50000, -1].reshape(50000, 1)
```

2) Sigmoid()

```python
def sigmoid(z):
    """# Notice that z can be a scalar, a vector or a matrix
    # return the sigmoid of input z"""

    return 1.0 / (1.0 + np.exp(-z))
```

3) nnObjFunction()

```python
n_input, n_hidden, n_class, training_data, training_label, lambdaval = args

w1 = params[0:n_hidden * (n_input + 1)].reshape((n_hidden, (n_input + 1)))
w2 = params[(n_hidden * (n_input + 1)):].reshape((n_class, (n_hidden + 1)))

training_data = np.hstack(
    [
        training_data,
        np.ones([training_data.shape[0], 1])
    ]
)
hidden1_values = sigmoid(np.matmul(training_data, w1.transpose()))

hidden1_values = np.hstack([
    hidden1_values,
    np.ones([hidden1_values.shape[0], 1])
])

out_values = sigmoid(np.matmul(hidden1_values, w2.transpose()))

# One hot encoding
truth_labels = np.zeros([training_label.shape[0], out_values.shape[1]])
for i in range(truth_labels.shape[0]):
    truth_labels[i, int(training_label[i])] = 1

delta_l = out_values - truth_labels
grad_w2 = np.dot(delta_l.T, hidden1_values)

grad_w2 = (np.add((lambdaval * w2), grad_w2)) / training_data.shape[0]

sum_delta_weight2 = np.dot(delta_l, w2[:, :-1])
one_minus_z_dot_z = (1.0 - hidden1_values[:, :-1]) * hidden1_values[:, :-1]
lft_part = sum_delta_weight2 * one_minus_z_dot_z
grad_w1 = np.dot(lft_part.T, training_data)

grad_w1 = (np.add((lambdaval * w1), grad_w1)) / training_data.shape[0]

obj_grad = np.concatenate((grad_w1.flatten(), grad_w2.flatten()), 0)

tot_err = (-np.add(
    np.dot(
        truth_labels.flatten(),
        np.log(out_values).flatten().T
    ),
    np.dot(
        1 - truth_labels.flatten(),
        np.log(1 - out_values).flatten().T
    )
) / training_data.shape[0])

regularization = lambdaval * np.add(np.sum(w1 ** 2), np.sum(w2 ** 2)) / (2 * training_data.sha

obj_val = tot_err + regularization

print(obj_val)

return obj_val, obj_grad
```

4) nnPredict()

```python
labels = np.empty([data.shape[0], 1])

data = np.hstack([data, np.ones([data.shape[0], 1])])

hidden1_values = sigmoid(np.dot(data, w1.transpose()))
hidden1_values = np.hstack([
    hidden1_values,
    np.ones([hidden1_values.shape[0], 1])
])

out_values = sigmoid(np.dot(hidden1_values, w2.T))

for index in range(0, out_values.shape[0]):
    labels[index] = np.argmax(out_values[index])

return labels
```

Next: Relation between Regularization, Accuracy, and Hidden Neurons on MNIST Data (nnScript.py)

2) Relation between Regularization, Accuracy, and Hidden Neurons on MNIST Data (nnScript.py)

Machine Learning is a highly iterative task with lots of trial and error to find the optimal Hyperparameters. For this assignment we wrote a script that will train different models with Different Hyperparameters.

We decided to increment the Regularization parameters starting from 0 to 60 with increments of 10. Therefore $\lambda = \{0, 10, 20, 30, 40, 50, 60\}$

We were given 784 image features in the MNIST dataset. These features (inputs) would converge to the hidden layers neurons and finally pass on to the 10 neurons in the output layer. For the number of Hidden neurons (n), we decided to start with 2 and increment it with the exponents of 2 until the accuracy starts to decrease again. Therefore

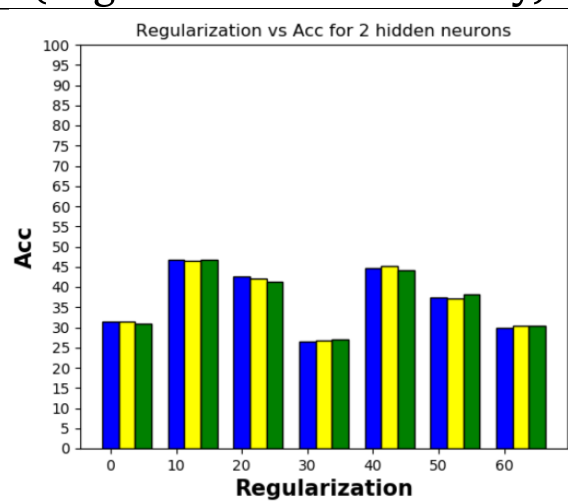
$n = \{2, 4, 8, 16, 32, 64, 128\}$.

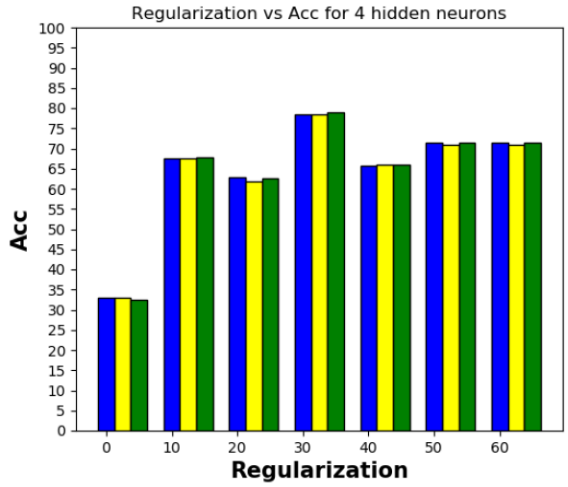
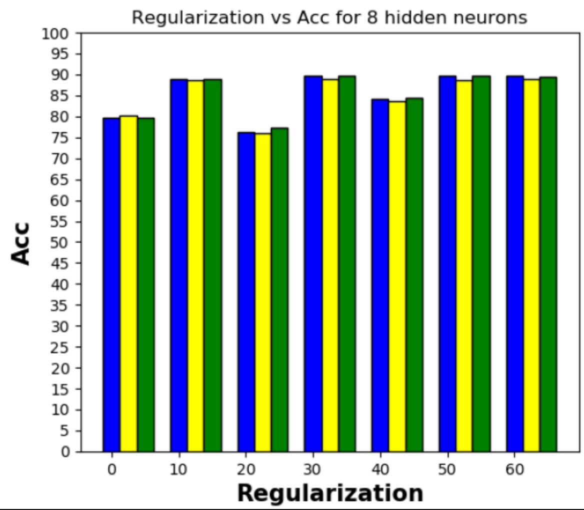
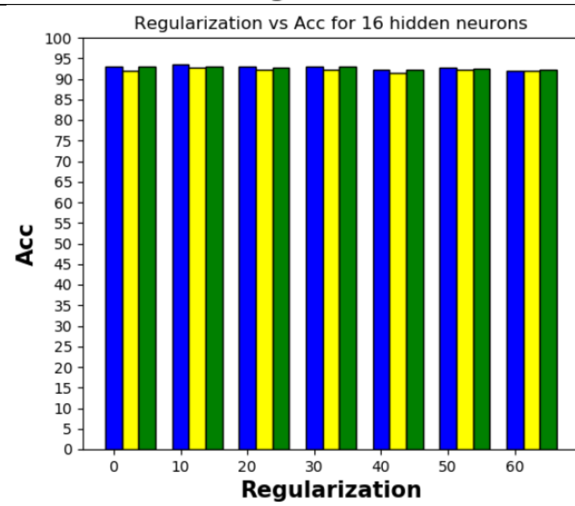
Graph label:

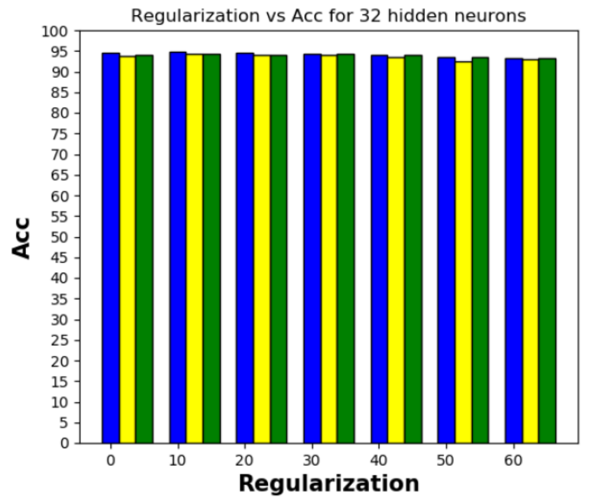
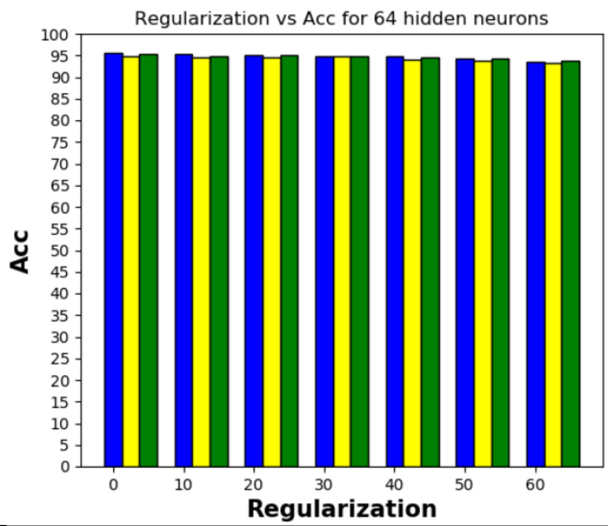
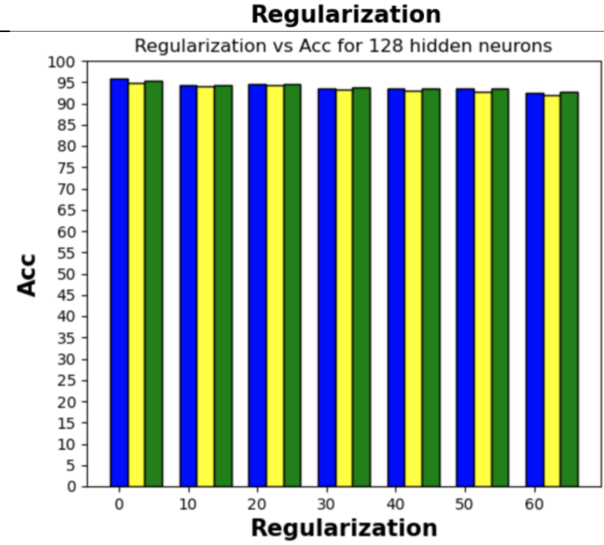
Blue: Accuracy on Training data

Yellow: Accuracy on Validation data

Green: Accuracy on Test data

# Hidden Neurons	Regularization (λ) Range	Graph (Regularization vs Accuracy)	Highest Accuracy	Training time
2	0-60		Test: 44.5% Validation: 45.1% Test: 44.2%	For the model with Highest accuracy on Test and Train dataset: 00:00:10

4	0-60		Test: 78.3% Validation: 78.5% Test: 79.0%	For the model with Highest accuracy on Test and Train dataset: 00:00:13
8	0-60		Test: 89.8% Validation: 88.7% Test: 89.7%	For the model with Highest accuracy on Test and Train dataset: 00:00:15
16	0-60		Test: 93.4% Validation: 92.6% Test: 93.0%	For the model with Highest accuracy on Test and Train dataset: 00:00:21

32	0-60		Test: 94.4% Validation: 93.7% Test: 93.9%	For the model with Highest accuracy on Test and Train dataset: 00:00:23
64	0-60		Test: 95.6% Validation: 94.8% Test: 95.30%	For the model with Highest accuracy on Test and Train dataset: 00:00:30
128	0-60		Test: 95.7% Validation: 94.7% Test: 95.24%	For the model with Highest accuracy on Test and Train dataset: 00:00:44

Observations:

We see that, with only 2 hidden neurons, the highest accuracy we get is 44.2% because 2 neurons are not enough to extract 784 image features.

Whereas, as we increase the number of hidden neurons, the accuracy increases. For example, the maximum accuracy on the test dataset we observe is 95.3% with 64 hidden neurons and no regularization. The accuracy starts to decrease again as we increase the hidden neurons further.

Underfitting vs Overfitting:

Let's take the models with 64 hidden neurons as an example.

For $\lambda = 0$,

Test accuracy: 95.69%
Train Accuracy: 95.30%
Difference: 0.39%

For $\lambda = 60$,

Test accuracy:93.56%
Train accuracy:92.81%
Difference: 0.75%

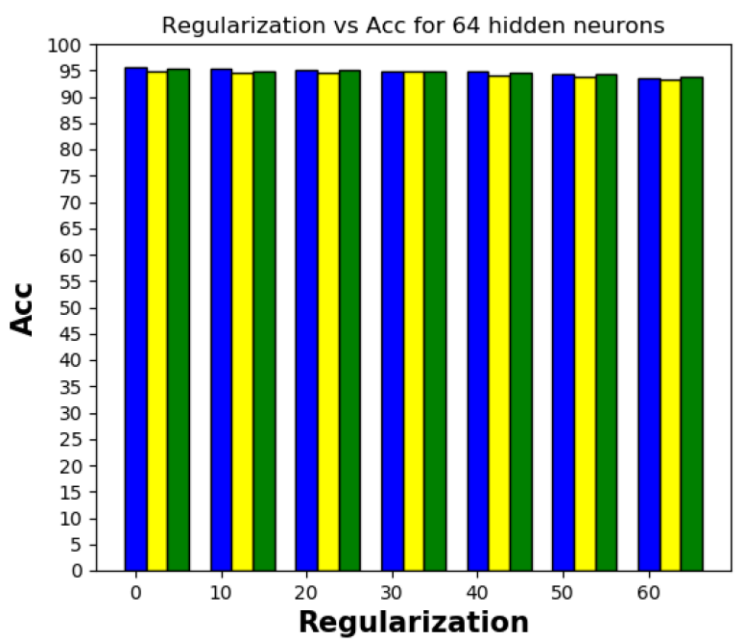
As we can observe, the difference between the test accuracy and training accuracy increases as we increase the λ value. The trade-off between the bias and variance need to be calculated to check for underfitting and overfitting of the model on the training data.

3) Choosing appropriate lambda value to avoid overfitting and underfitting

We observe that, for hidden neurons higher that 32, $\lambda = \{0\}$ gives the highest accuracy.

The accuracy decreases as we increase the λ value.

For example, Lets consider the graph of $n = 64$ and deduce the effect of increasing the lambda value.



Regularization	Test Accuracy
0	95.30%
10	94.87%
20	95.04%
30	94.71%
40	94.62%
50	94.22%

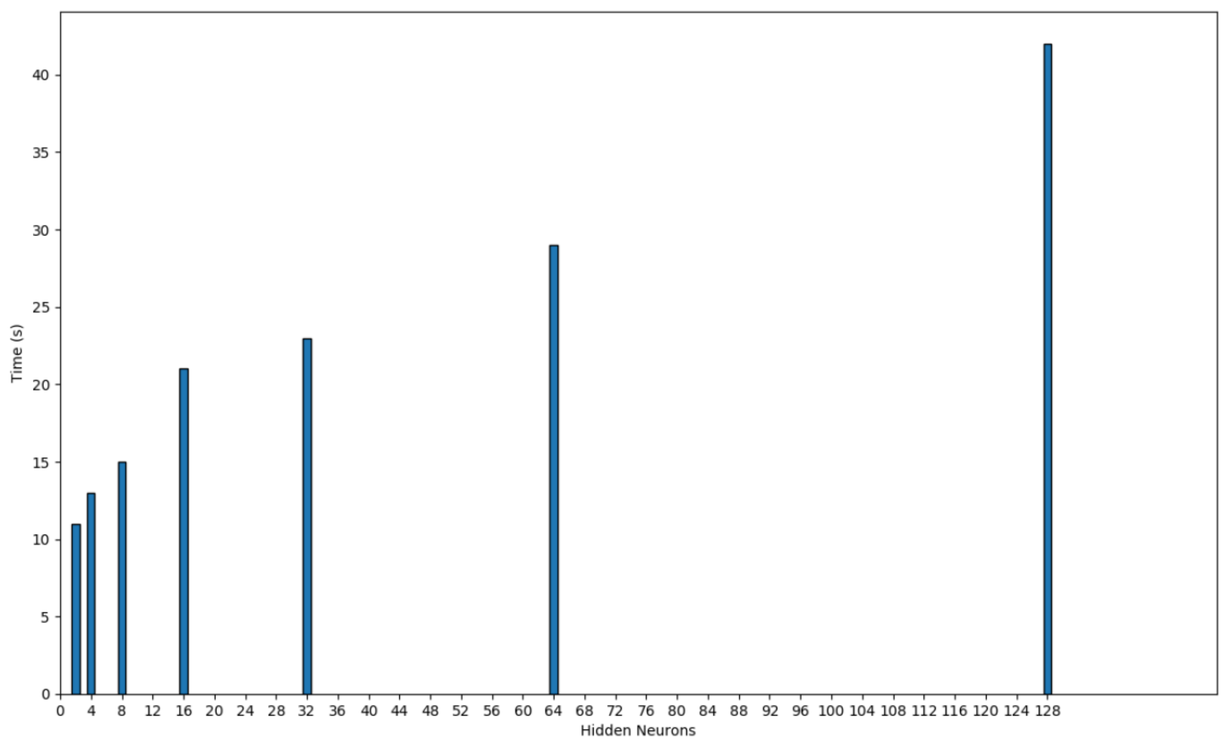
60	93.69%

The bias of the model is increasing (underfitting) as we increase the regularization. Hence, reducing it’s accuracy on the test data.

4) Hidden Units vs Training time

Note:

All the models are trained on the CPU of Apple’s M1 Pro System on Chip (SoC).



Comparing the model with highest accuracy in $n = \{2, 4, 8, 16, 32, 64, 128\}$ hidden units and training time.

# Neurons in hidden layer	Training time (Highest accuracy model)
2	00:00:10
4	00:00:13
8	00:00:15
16	00:00:21
32	00:00:23
64	00:00:30
128	00:00:44

5) Comparing the results of deep neural network and neural network with one hidden layer on the CelebA data set. (Shallow vs Deep Network)

CelebA dataset	**Shallow Network**	**Deep Network**
Training time	00:00:14	00:02:39
Accuracy	85.80%	90.00%

Hyperparameters for Shallow Network (facennscript.py):

N_hidden: 4

Regularization (λ): 0

Hyperparameters for Deep Network (facennscript.py):

Number of layers: 3 (Input -> H1 -> H2 -> Output)

N_hidden_1: 1024

N_hidden_2: 512

Learning_rate: 0.008

Epochs: 50

Batch_Size: 100

Next: How did we choose the Hyperparameters!

6) How to choose Hyperparameters for facennscript.py

N_hidden: 4

We tried with different n_hidden values with increments as 2^n. n = {2, 4, 8, 16, 32, 64, 128}. After n = 4, the training time increases drastically but there is no significant improvement in the accuracy of the model. To reduce the dimensions and training time, we decided to choose n_hidden = 4 which gave us an accuracy of 85.80% on the testing dataset.

Regularization (λ): 0

Since there is no high variance in our model (overfitting), we don't need to add a regularization parameter to our model. We tried with a small regularization value (5) and the model started to have high bias (underfitting) which resulted in lower accuracy.

7) Accuracy on Handwritten digits and CelebA test Data

1) Handwritten Digits

	Test data Accuracy	Training time
Shallow Network	95.3%	00:00:30
Deep Network	95.8%	00:02:45
CNN architecture	98.7%	00:13:06

2) CelebA

	Test data Accuracy	Training Time
Shallow Network	85.80%	00:00:14
Deep Network	90.00%	00:02:39

8) Results from Convolutional Neural Network

Note:

We chose to test the Deep Neural Network and Convolutional Neural Network architectures on the MNIST dataset to demonstrate the difference in training time.

Result after training for 9 epochs:

```
Confusion Matrix:
[[ 964    0    0    0    0    3    7    1    5    0]
 [    0 1099    4    2    2    1    5    0   22    0]
 [   13    1  936   16   11    3    9   11   30    2]
 [    4    0    8  928    0   25    0   11   26    8]
 [    1    1    4    0  929    0   14    2    3   28]
 [    9    1    3   18    3  823   14    1   15    5]
 [   12    3    1    0    5   15  917    1    4    0]
 [    0    4   24    5    5    1    0  940    6   43]
 [    9    0    3   14    8   15    7    8  898   12]
 [    8    5    6    9   15    7    0   16    9  934]]
Epochs 9
time usage: 0:13:06
Accuracy: 98.7%, Avg loss: 0.037329
```

Training time for MNIST dataset: 00:13:06

Accuracy: 98.7%

Minimized loss: 0.037

Comparing CNN architecture and Flattened out Neural Network on Handwritten Digits dataset:

Note: We use the same Hyperparameters for better comparison (**Epochs: 9**)

	Deep Neural Network	**CNN Architecture**
Test Accuracy	95.8%	98.7%
Training Time	00:02:45	00:13:06