Neural Networks
Lab Assignment 1

Karan Praharaj

January 31, 2022

CODE

1. (a) Neural Network Hyperparameters

[144]:
```
# To ignore sklearn warnings
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

[145]:
```
from sklearn.datasets import load_digits
from sklearn import metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

[43]:
```
digits = load_digits()
X = digits.data
y = digits.target
```

[44]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
 ↪random_state=42)
```

[46]:
```
# Standardize

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_std = scaler.transform(X_train)
X_test_std = scaler.transform(X_test)
```

[86]:
```
# Setting two different values for the following hyperparameters:

num_hidden_layers = [1,2]
neurons_per_layer = [20,50]
activation_fn = ['relu', 'tanh']

# Setting the following hyperparameters to constant values:
batch_size = 32
num_epochs = 10
optimizer = 'adam'
```

[ ]:
```
# Training models using all possible combinations of the hyperparameters.

nn_models = []

for layers in num_hidden_layers:
    for neurons in neurons_per_layer:
        for activation in activation_fn:
            mlp = MLPClassifier(activation=activation, hidden_layer_sizes =␣
 ↪[neurons]*layers,
                                max_iter=num_epochs, batch_size=batch_size,␣
 ↪solver=optimizer)
            mlp.fit(X_train_std, y_train)
            nn_models.append(mlp)
```

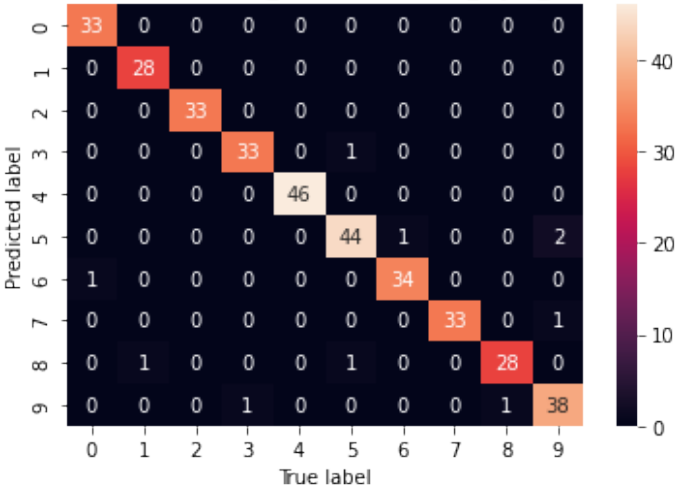
[122]:
```
# Evaluation of the models by accuracy

for i, model in enumerate(nn_models):
    print(model)
    y_pred = model.predict(X_test_std)
    print("Accuracy :", metrics.accuracy_score(y_test, y_pred))
    print("----------------------------------------")
    print()
```

```
MLPClassifier(batch_size=32, hidden_layer_sizes=[20], max_iter=10)
Accuracy : 0.9305555555555556
----------------------------------------

MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[20],
              max_iter=10)
Accuracy : 0.925
----------------------------------------

MLPClassifier(batch_size=32, hidden_layer_sizes=[50], max_iter=10)
Accuracy : 0.9611111111111111
----------------------------------------
```

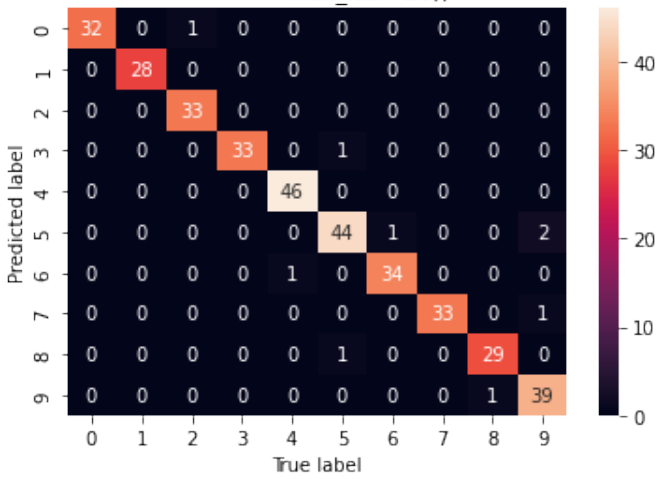
```
MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[50],
              max_iter=10)
Accuracy : 0.9611111111111111
----------------------------------------

MLPClassifier(batch_size=32, hidden_layer_sizes=[20, 20], max_iter=10)
Accuracy : 0.9444444444444444
----------------------------------------

MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[20, 20],
              max_iter=10)
Accuracy : 0.9416666666666667
----------------------------------------

MLPClassifier(batch_size=32, hidden_layer_sizes=[50, 50], max_iter=10)
Accuracy : 0.9666666666666667
----------------------------------------

MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[50, 50],
              max_iter=10)
Accuracy : 0.9722222222222222
----------------------------------------
```

[79]:
```python
# Confusion matrix

for i in range(len(nn_models)):
    y_pred = nn_models[i].predict(X_test_std)
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(i)
    plt.title(f"Heatmap for Model {i+1} ({nn_models[i]})")
    sns.heatmap(cm, annot = True)
    plt.xlabel('True label')
    plt.ylabel('Predicted label')
```

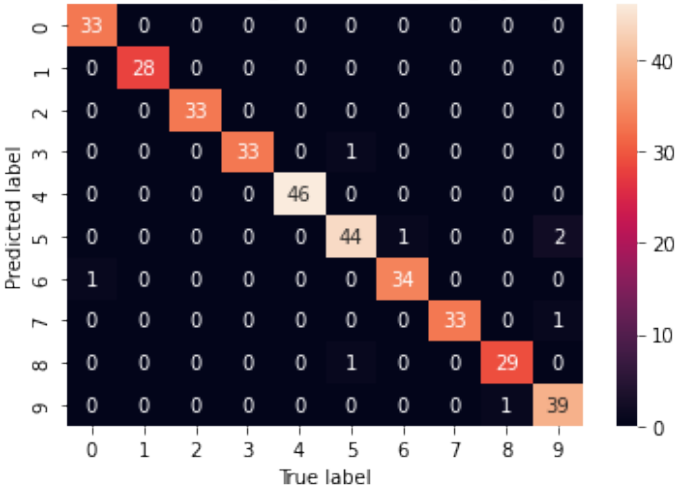
Heatmap for Model 1 (MLPClassifier(batch_size=32, hidden_layer_sizes=[100], max_iter=20))



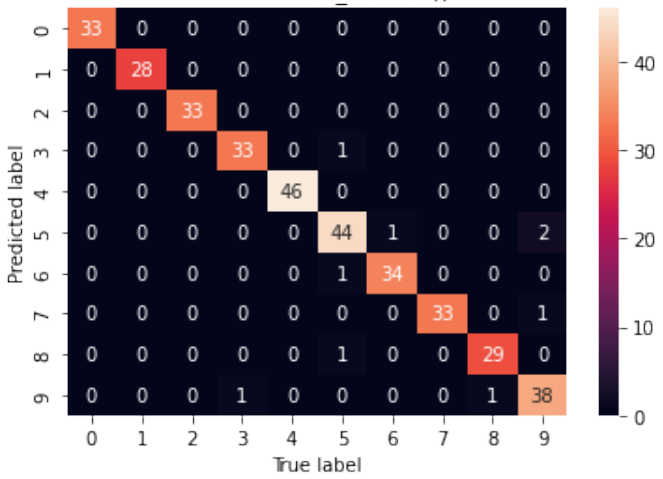
Heatmap for Model 2 (MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[100], max_iter=20))



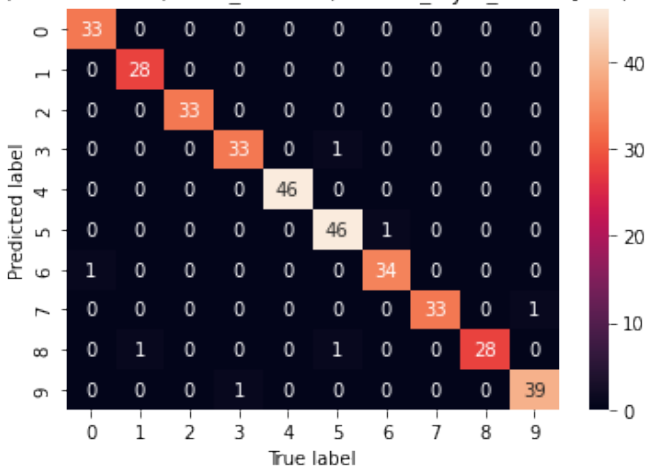
Heatmap for Model 3 (MLPClassifier(batch_size=32, hidden_layer_sizes=[200], max_iter=20))



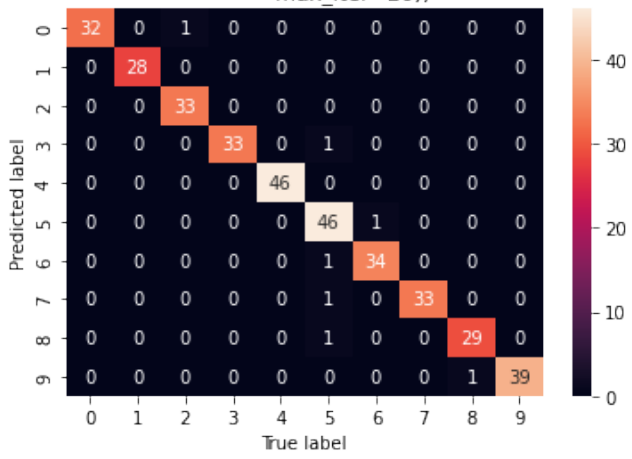
Heatmap for Model 4 (MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[200], max_iter=20))

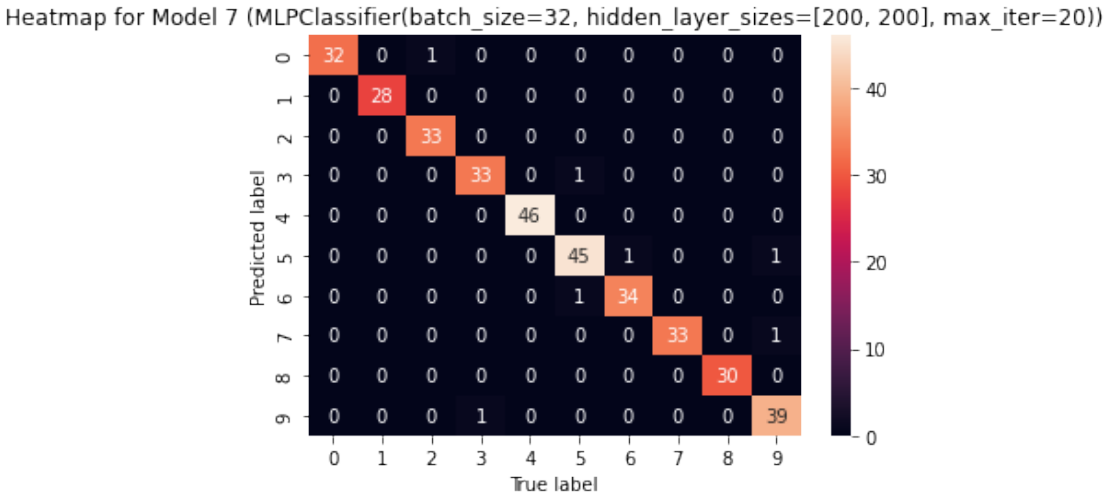


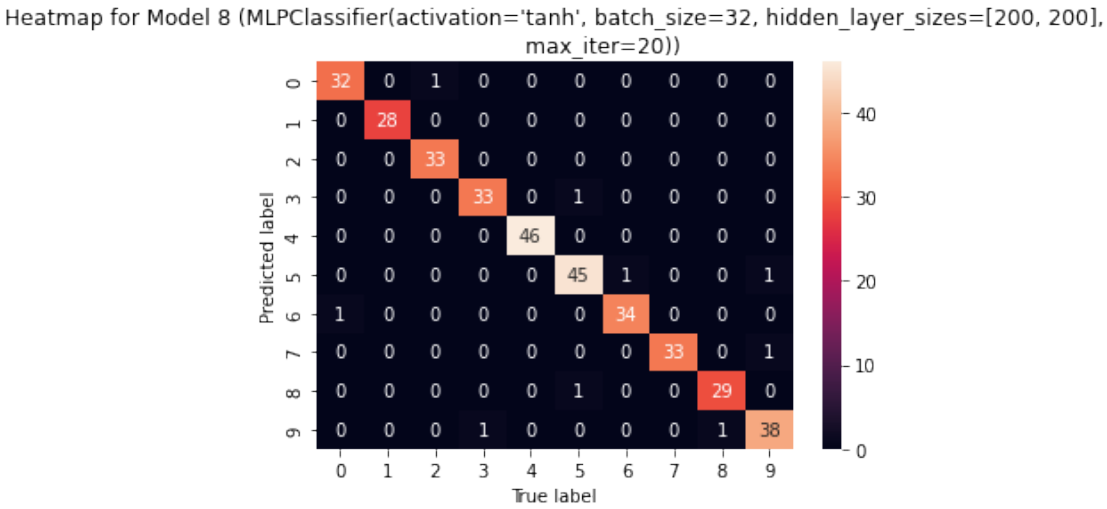
Heatmap for Model 5 (MLPClassifier(batch_size=32, hidden_layer_sizes=[100, 100], max_iter=20))



Heatmap for Model 6 (MLPClassifier(activation='tanh', batch_size=32, hidden_layer_sizes=[100, 100], max_iter=20))







[ ]:

2. (a) Impact of Training Duration and Training Data

[90]:
```
# Loading Data
# (We don't need to do this again because we will use the same data, but showing␣
 ↪the step anyway for thoroughness.)

digits = load_digits()
X = digits.data
y = digits.target
```

[92]:
```
X.shape
```

[92]:
```
(1797, 64)
```

[117]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,␣
 ↪random_state=42)
```

[116]:
```
# Standardize

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_std = scaler.transform(X_train)
X_test_std = scaler.transform(X_test)
```

[123]:
```
# Collecting subsets (25%, 50%, 75%) of the full training set by sampling.

x_25, x_50, x_75 = [], [], []
y_25, y_50, y_75 = [], [], []

for i in np.random.choice(len(X_train_std), int(len(X_train_std)*0.25)):
    x_25.append(X_train_std[i])
    y_25.append(y_train[i])

for i in np.random.choice(len(X_train_std), int(len(X_train_std)*0.5)):
    x_50.append(X_train_std[i])
    y_50.append(y_train[i])

for i in np.random.choice(len(X_train_std), int(len(X_train_std)*0.75)):
    x_75.append(X_train_std[i])
    y_75.append(y_train[i])
```

[125]:
```
len(x_25), len(x_50), len(x_75), len(X_train)
```

[125]:
```
(359, 718, 1077, 1437)
```

[190]:
```
x_subsets = {0.25:x_25, 0.5:x_50, 0.75:x_75, 1.0:X_train_std}
y_subsets = {0.25:y_25, 0.5:y_50, 0.75:y_75, 1.0:y_train}
```

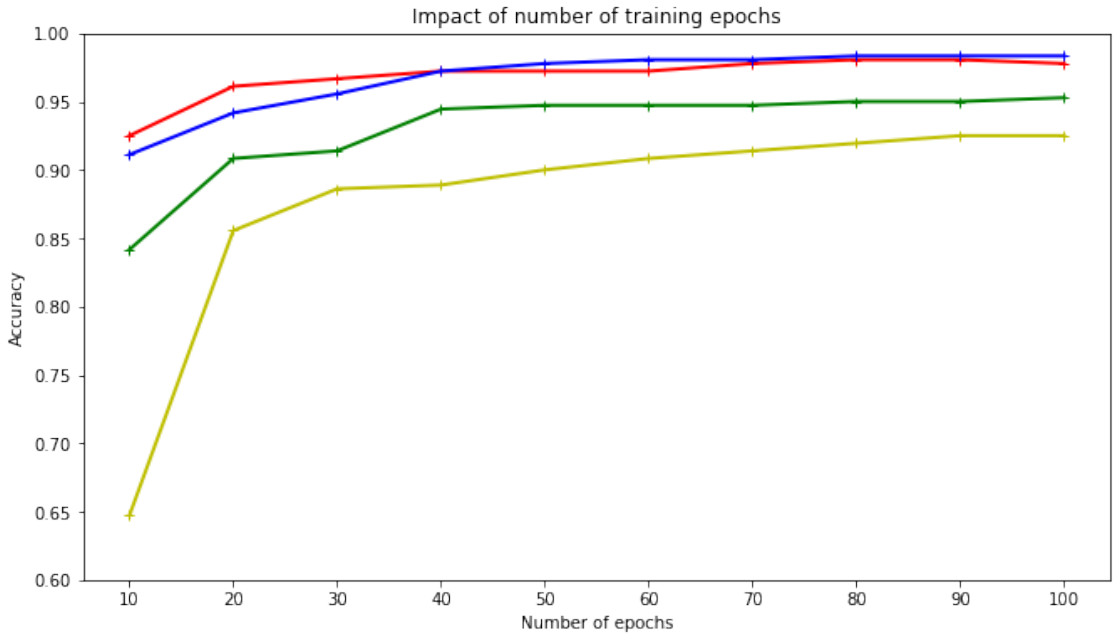
[191]:
```
scores = {}
epochs_list = []
```

[192]:
```
# Setting hyperparameters for training.

num_hidden_layers = 2
```

```
neurons_per_layer = 20
activation = 'relu'
batch_size = 32
optimizer = 'sgd'
```

[193]:
```
# Training

start_num_epochs = 10
end_num_epochs = 110
increment = 10

for epochs in range(start_num_epochs, end_num_epochs, increment):
    for subset in x_subsets:
        mlp = MLPClassifier(max_iter=epochs, random_state=42)
        mlp.fit(x_subsets[subset], y_subsets[subset])
        acc_score = mlp.score(X_test_std, y_test)
        if subset in scores:
            scores[subset].append(acc_score)
        else:
            scores[subset] = [acc_score]
    epochs_list.append(epochs)
```

[194]:
```
plt.figure(figsize=(11,6))
plt.plot(epochs_list, scores[1.0], "r-+", linewidth = 2)
plt.plot(epochs_list, scores[0.75], "b-+", linewidth = 2)
plt.plot(epochs_list, scores[0.5], "g-+", linewidth = 2)
plt.plot(epochs_list, scores[0.25], "y-+", linewidth = 2)
plt.xlabel("Number of epochs")
plt.ylabel("Accuracy")
plt.xticks(np.arange(10, 110, 10))
plt.yticks(np.arange(0.6, 1.05, 0.05))
plt.title("Impact of number of training epochs")
plt.show()
```



WRITE-UP

Report your methods, results and analysis.

1. (b) Neural Network Hyperparameters

- We used the multi-layer perceptron class from sklearn (MLPClassifier) to write our neural networks. The dataset used was the Optical recognition of handwritten digits dataset, offered on sklearn.datasets. The data set contains images of hand-written digits. It has a total of 10 classes where each class corresponds to digits between 0 to 9. The dataset has 1797 examples and 64 features in total.

  The parameters used to train the model were: Number of hidden layers = $\{1, 2\}$ Number of neurons per hidden layer = $\{20, 50\}$ activation_fn = $\{relu, tanh\}$

  The above parameters were allowed to vary. The following hyperparameters were kept constant: batch_size = 32 num_epochs = 10 optimizer = *adam*

- We report the results in terms of accuracy scores on the test set. The results obtained for every tested model are as follows. Table provided as well (after the list).

1. **{activation='relu', hidden_layer_sizes=[20]}** Accuracy : 0.9305555555555556
2. **{activation='tanh', hidden_layer_sizes=[20]}** Accuracy : 0.925
3. **{activation='relu', hidden_layer_sizes=[50]}** Accuracy : 0.9611111111111111
4. **{activation='tanh', hidden_layer_sizes=[50]}** Accuracy : 0.9611111111111111

5. **{activation='relu', hidden_layer_sizes=[20, 20]}** Accuracy : 0.9444444444444444
6. **{activation='tanh', hidden_layer_sizes=[20, 20]}** Accuracy : 0.9416666666666667
7. **{activation='relu', hidden_layer_sizes=[50, 50]}** Accuracy : 0.9666666666666667
8. **{activation='tanh', hidden_layer_sizes=[50, 50]}** Accuracy : 0.9722222222222222

Activation	1st layer neurons	2nd layer neurons	Accuracy (test)
relu	20	-	0.931
tanh	20	-	0.925
relu	50	-	0.961
tanh	50	-	0.961
relu	20	20	0.944
tanh	20	20	0.942
relu	50	50	0.967
tanh	50	50	0.972

- We observe respectable accuracy scores, even with a grid-search space that was not very exhaustive. The models seem to tend to underfitting everytime they are trained using only one hidden layer rather than two. This can be interpreted as the trained neural network not being sufficiently complex. When the number of hidden layers are increased to two, there is an increase in accuracy seen. (comparing with training using same number of neurons in the one-layer case). i.e., increasing number of layers while keeping everything else the same gives better accuracy scores.

  Observing the number of neurons per layer in isolation, we see a general trend of increasing accuracy scores with an increasing number of neurons. It is not surprising that the highest accuracy score was observed in the case that had the highest possible number of layers, with the highest number of neurons in each layer. This also indicates that the model was still learning, and had not converged or plateaud at the end of 10 epochs. Training for more epochs is expected to bring even better results.

  Looking at the heatmaps, we see that no model does terribly on any one class specifically. The few wrong predictions are spread out, but notably the digit that was found to be the most difficult by the models was "9". We see that for "9" there are 2-3 false predictions by a few models.

[ ]:

2. (b) Impact of Training Duration and Training Data

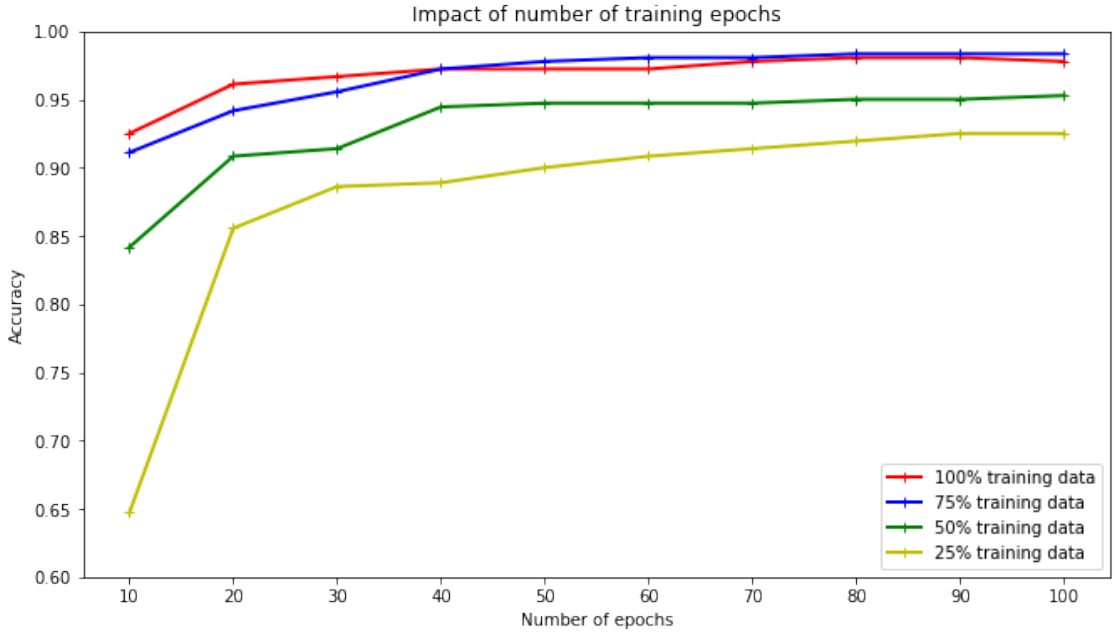
- We (again) used the multi-layer perceptron class from sklearn (MLPClassifier) to write our neural networks. The dataset used was the Optical recognition of handwritten digits dataset, offered on sklearn.datasets. The data set contains images of hand-written digits. It has a total of 10 classes where each class corresponds to digits between 0 to 9. The dataset has 1797 examples and 64 features in total.

  The parameters used to train the model were as follows: batch_size = 32 optimizer = *sgd* activation_fn = *relu* Number of hidden layers = 2 Number of neurons per hidden layer = 20 Number of epochs = 100

The above hyperparameters were kept constant. We collected the scores at an increment of 10 epochs, for a total of 100 epochs.

- Showing the plot that visualizes the performance for each of the four training data subsets:

[195]:
```
plt.figure(figsize=(11,6))
plt.plot(epochs_list, scores[1.0], "r-+", linewidth = 2, label='100% training␣
 ↪data')
plt.plot(epochs_list, scores[0.75], "b-+", linewidth = 2, label='75% training␣
 ↪data')
plt.plot(epochs_list, scores[0.5], "g-+", linewidth = 2, label='50% training␣
 ↪data')
plt.plot(epochs_list, scores[0.25], "y-+", linewidth = 2, label='25% training␣
 ↪data')
plt.legend(loc='lower right')
plt.xlabel("Number of epochs")
plt.ylabel("Accuracy")
plt.xticks(np.arange(10, 110, 10))
plt.yticks(np.arange(0.6, 1.05, 0.05))
plt.title("Impact of number of training epochs")
plt.show()
```



- When the model is trained on 100% of the data, the performance over 100 epochs in terms of test accuracy is not very different from the case when it is trained on 75% of the data. This suggests that the model reaches a point of diminishing returns at one stage, and does not have significant gains when trained on more data. For 50% and 25%, we see that the model has a progressively lower performance. This stands to reason, and aligns with intuition

because training on lesser data mostly results in a lesser ability to generalize.

Another observation is that performance in terms of test accuracy improves with a longer training duration. That is, training for a higher number of epochs increases performance in general. It is clear from the plot that the lowest accuracy scores were obtained in the early stages of training when the model wasn't trained for many epochs, but the highest scores were seen towards the end of the maximum number of epochs (100).

In general, the trend seems to be : *training data* $\uparrow$, *test accuracy* $\uparrow$ *training data* $\downarrow$, *test accuracy* $\downarrow$

training duration $\uparrow$, *test accuracy* $\uparrow$ *training duration* $\downarrow$, *test accuracy* $\downarrow$