cs273-hw4-Abdul_Kalam_Syed

November 3, 2025

1 CS273A Homework 4

1.1 Due: Monday November 3th, 2025 (11:59pm)

1.2 Instructions

This homework (and subsequent ones) will involve data analysis and reporting on methods and results using Python code. You will submit a **single PDF file** that contains everything to Gradescope. This includes any text you wish to include to describe your results, the complete code snippets of how you attempted each problem, any figures that were generated, and scans of any work on paper that you wish to include. It is important that you include enough detail that we know how you solved the problem, since otherwise we will be unable to grade it.

Your homeworks will be given to you as Jupyter notebooks containing the problem descriptions and some template code that will help you get started. You are encouraged to use these starter Jupyter notebooks to complete your assignment and to write your report. This will help you not only ensure that all of the code for the solutions is included, but also will provide an easy way to export your results to a PDF file (for example, doing *print preview* and *printing to pdf*). I recommend liberal use of Markdown cells to create headers for each problem and sub-problem, explaining your implementation/answers, and including any mathematical equations. For parts of the homework you do on paper, scan it in such that it is legible (there are a number of free Android/iOS scanning apps, if you do not have access to a scanner), and include it as an image in the Jupyter notebook.

Double check that all of your answers are legible on Gradescope, e.g. make sure any text you have written does not get cut off.

If you have any questions/concerns about using Jupyter notebooks, ask us on EdD. If you decide not to use Jupyter notebooks, but go with Microsoft Word or LaTeX to create your PDF file, make sure that all of the answers can be generated from the code snippets included in the document. ### Summary of Assignment: 100 total points - Problem 1: A Small Neural Network (30 points) - Problem 1.1: Forward Pass (10 points) - Problem 1.2: Evaluate Loss (10 points) - Problem 1.3: Network Size (10 points) - Problem 2: Neural Networks on MNIST (35 points) - Problem 2.1: Varying the Amount of Training Data (15 points) - Problem 2.3: Optimization Curves (10 points) - Problem 2.3: Tuning your Neural Network (10 points) - Problem 3: Convolutional Networks (30 points) - Problem 3.1: Model structure (10 points) - Problem 3.2: Training (10 points) - Problem 3.3: Evaluation (5 points) - Problem 3.4: Comparing predictions (5 points) - Statement of Collaboration (5 points)

Before we get started, let's import some libraries that you will make use of in this assignment. Make sure that you run the code cell below in order to import these libraries.

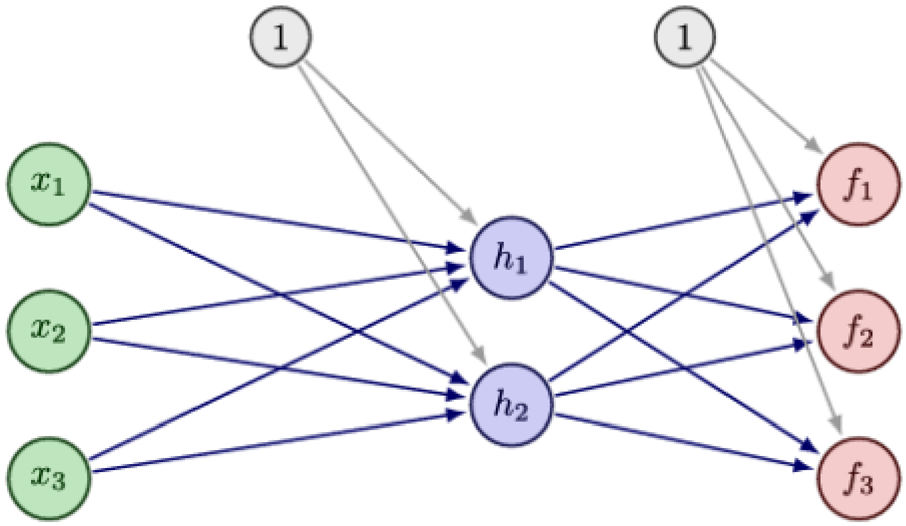
Important: In the code block below, we set `seed=1234`. This is to ensure your code has reproducible results and is important for grading. Do not change this. If you are not using the provided Jupyter notebook, make sure to also set the random seed as below.

Important: Do not change any codes we give you below, except for those waiting for you to complete. This is to ensure your code has reproducible results and is important for grading.

[2]:
```python
import numpy as np
import matplotlib.pyplot as plt

import torch

from IPython import display

from sklearn.datasets import fetch_openml             # common data set access
from sklearn.preprocessing import StandardScaler      # scaling transform
from sklearn.model_selection import train_test_split # validation tools
from sklearn.metrics import accuracy_score

from sklearn.neural_network import MLPClassifier     # scikit's MLP

import warnings
warnings.filterwarnings('ignore')

# Fix the random seed for reproducibility
# !! Important !! : do not change this
seed = 1234
np.random.seed(seed)
torch.manual_seed(seed);
```

1.3 Problem 1: A Small Neural Network

Consider the small neural network given in the image below, which will classify a 3-dimensional feature vector $\mathbf{x}$ into one of three classes ($y = 0, 1, 2$):



You are given an input to this network $\mathbf{x}$,

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} = \begin{bmatrix} 1 & 3 & -2 \end{bmatrix}$$

as well as weights W for the hidden layer and weights B for the output layer.

$$W = \begin{bmatrix} w_{01} & w_{11} & w_{21} & w_{31} \\ w_{02} & w_{12} & w_{22} & w_{32} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 5 \\ 2 & 1 & 1 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} \beta_{01} & \beta_{11} & \beta_{21} \\ \beta_{02} & \beta_{12} & \beta_{22} \\ \beta_{03} & \beta_{13} & \beta_{23} \end{bmatrix} = \begin{bmatrix} 4 & -1 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \end{bmatrix}$$

For example, w_{12} is the weight connecting input x_1 to hidden node h_2; w_{01} is the constant (bias) term for h_1, etc.

This network uses the ReLU activation function for the hidden layer, and uses the softmax activation function for the output layer.

Answer the following questions about this network.

1.3.1 Problem 1.1 (10 points): Forward Pass

- Given the inputs and weights above, compute the values of the hidden units h_1, h_2 and the outputs f_0, f_1, f_2. You should do this by hand, i.e. you should not write any code to do the calculation, but feel free to use a calculator to help you do the computations.
- You can optionally use LaTeX in your answer on the Jupyter notebook. Otherwise, write your answer on paper and include a picture of your answer in this notebook. In order to include an image in Jupyter notebook, save the image in the same directory as the .ipynb file and then write `![caption](image.png)`. Alternatively, you may go to Edit –> Insert Image at

the top menu to insert an image into a Markdown cell. **Double check that your image is visible in your PDF submission.**

- What class would the network predict for the input **x**?

Abdul Kalam Syed ① 11/03/2025

CS273A :- Homework #4

Problem 1 :-

$$x = [x_1, x_2, x_3] = [1 \;\; 3 \;\; -2]$$

$$W = \begin{bmatrix} w_{01} & w_{11} & w_{21} & w_{31} \\ w_{02} & w_{12} & w_{22} & w_{32} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 5 \\ 2 & 1 & 1 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} \beta_{01} & \beta_{11} & \beta_{21} \\ \beta_{02} & \beta_{12} & \beta_{22} \\ \beta_{03} & \beta_{13} & \beta_{23} \end{bmatrix} = \begin{bmatrix} 4 & -1 & 0 \\ 3 & 0 & 2 \\ 2 & 1 & 1 \end{bmatrix}$$

Problem 1.1 :- Forward Pass

Formula to find the hidden layer before activation :-

$$r_{lj} = \sum_i w_{lij} h_{(l-1)i} + b_{lj}, \quad h_{lj} = a_l(r_{lj})$$

$$r_j = w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 + w_{0j}$$

$$h_1 = r_1 = -1\cdot 1 + 0\cdot 3 + 5\cdot(-2) + 1 = -1+0-10+1 = -10$$

$$h_2 = r_2 = 1\cdot 1 + 1\cdot 3 + 2\cdot(-2) + 2 = 1+3-4+2 = 2$$

(without activation) $h_1 = -10$, $h_2 = 2$

With activation :-

$$h_j = \text{ReLU}(r_j) = \max(0, r_j)$$

$$h_1 = \max(0, -10) = 0$$

$$h_2 = \max(0, 2) = 2$$

$$h_1 = 0, \; h_2 = 2$$

2

Formula to find output before activation :-

$$s_k = \sum_j \beta_{jk} h_j + \beta_{0k}$$

$s_0 = -1 \cdot 0 + 0 \cdot 2 + 4 = 4$
$s_1 = 0 \cdot 0 + 2 \cdot 2 + 3 = 7$
$s_2 = 1 \cdot 0 + 1 \cdot 2 + 2 = 4$

Applying softmax :

$$f_k = \frac{e^{s_k}}{\sum_k e^{s_k}}$$

$$f_0 = \frac{e^4}{e^4 + e^7 + e^4} = 0.045$$

$$f_1 = \frac{e^7}{e^4 + e^7 + e^4} = 0.909$$

$$f_2 = \frac{e^4}{e^4 + e^7 + e^4} = 0.045$$

$h_1 = 0, \ h_2 = 2$
$f_0 = 0.045, \ f_1 = 0.909, \ f_2 = 0.045$

1.3.2 Problem 1.2 (10 points): Evaluate Loss

Typically when we train neural networks for classification, we seek to minimize the log-loss function. Note that the output of the log-loss function is always nonnegative (≥ 0), but can be arbitrarily large (you should pause for a second and make sure you understand why this is true).

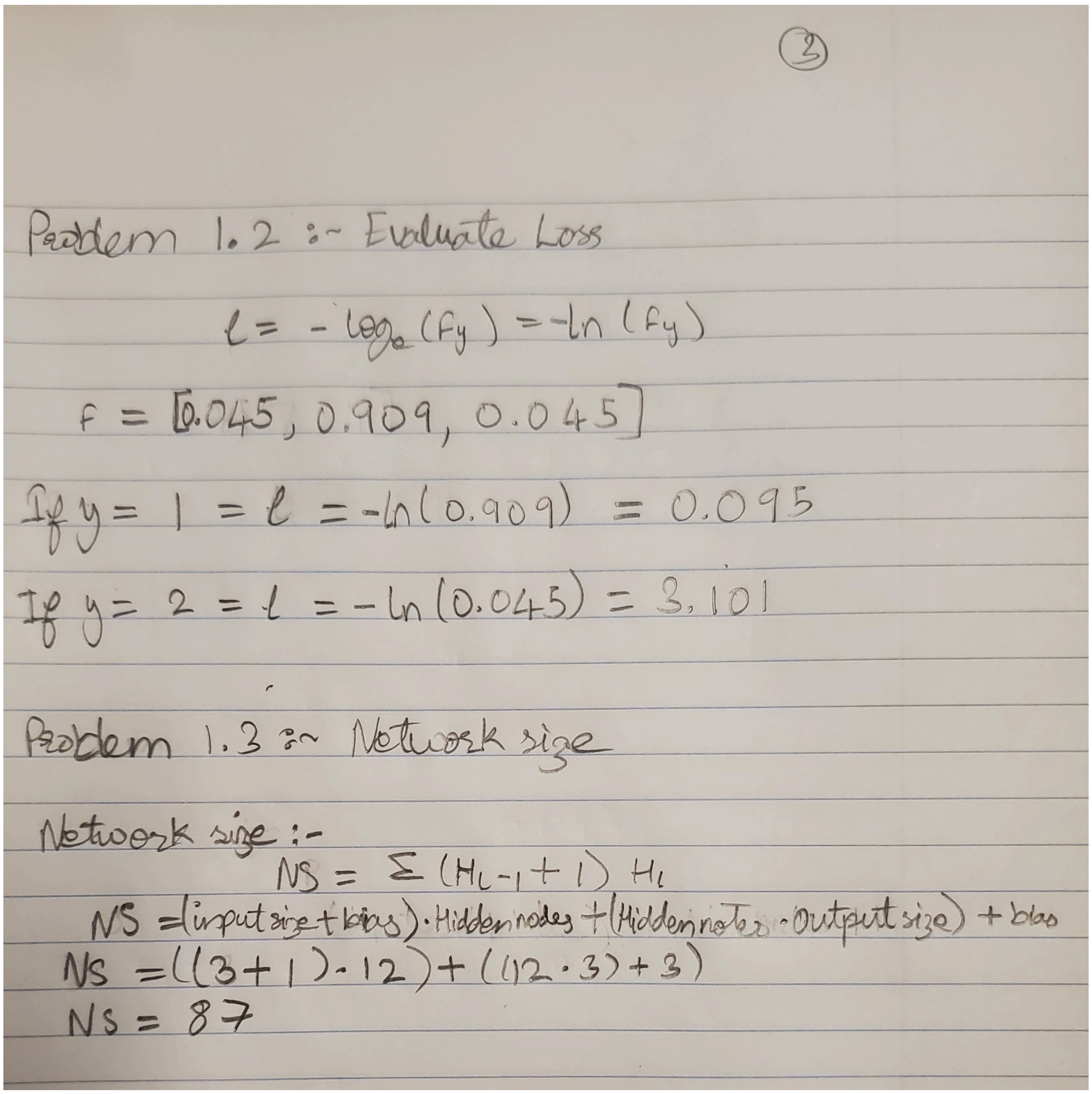
- Suppose the true label for the input $\mathbf{x}$ is $y = 1$. What would be the value of our loss function based on the network's prediction for $\mathbf{x}$?

- Suppose instead that the true label for the input $\mathbf{x}$ is $y = 2$. What would be the value of our loss function based on the network's prediction for $\mathbf{x}$?

You are free to use numpy / Python to help you calculate this, but don't use any neural network libraries that will automatically calculate the loss for you.

1.3.3 Problem 1.3 (10 points): Network Size

- Suppose we change our network so that there are 12 hidden nodes instead of 2. How many total parameters (weights and biases) are in our new network?



3

Problem 1.2 :- Evaluate Loss

$\ell = -\log_e(f_y) = -\ln(f_y)$

$f = [0.045, 0.909, 0.045]$

If $y = 1 = \ell = -\ln(0.909) = 0.095$

If $y = 2 = \ell = -\ln(0.045) = 3.101$

Problem 1.3 :- Network size

Network size :-

$NS = \Sigma (H_{l-1} + 1) H_l$

$NS = (\text{input size} + \text{bias}) \cdot \text{Hidden nodes} + (\text{Hidden nodes} \cdot \text{Output size}) + \text{bias}$

$NS = ((3+1) \cdot 12) + ((12 \cdot 3) + 3)$

$NS = 87$

<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

1.4 Problem 2: Neural Networks on MNIST

In this part of the assignment, you will get some hands-on experience working with neural networks. We will be using the scikit-learn implementation of a multi-layer perceptron (MLP). See here for the corresponding documentation. Although there are specialized Python libraries for neural networks, like TensorFlow and PyTorch, in this problem we'll just use scikit-learn since you're already familiar with it.

1.4.1 Problem 2.0: Setting up the Data

First, we'll load our MNIST dataset and split it into a training set and a testing set. Here you are given code that does this for you, and you only need to run it.

We will use the scikit-learn class `StandardScaler` to standardize both the training and testing features. Notice that we **only** fit the `StandardScaler` on the training data, and *not* the testing data.

[3]:
```
# Load the features and labels for the MNIST dataset
# This might take a minute to download the images.
X, y = fetch_openml('mnist_784', as_frame=False, return_X_y=True)

# Convert labels to integer data type
y = y.astype(int)
```

[4]:
```
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.1,␣
↪random_state=seed, shuffle=True)
```

[5]:
```
scaler = StandardScaler()
scaler.fit(X_tr)
X_tr = scaler.transform(X_tr)     # We can forget about the original values &␣
↪work
X_te = scaler.transform(X_te)     #  just with the transformed values from here
```

1.4.2 Problem 2.1: Varying the amount of training data (15 points)

One reason that neural networks have become popular in recent years is that, for many problems, we now have access to very large datasets. Since neural networks are very flexible models, they are often able to take advantage of these large datasets in order to achieve high levels of accuracy. In this problem, you will vary the amount of training data available to a neural network and see what effect this has on the model's performance.

In this problem, you should use the following settings for your network: - A single hidden layer with 64 hidden nodes - Use the ReLU activation function - Train the network using stochastic gradient descent (SGD) and a constant learning rate of 0.001 - Use a batch size of 256 - **Make sure to set `random_state=seed`.**

Your task is to implement the following: - Train an MLP model (with the above hyperparameter settings) using the first `m_tr` feature vectors in `X_tr`, where `m_tr = [100, 1000, 5000, 10000, 20000, 50000, 63000]`. You should use the `MLPClassifier` class from scikit-learn in your implementation. - Create a plot of the training error and testing error for your MLP model as a function of the number of training data points. For comparison, also plot the training and test error rates we found when we trained a logistic regression model on MNIST (these values are provided below). Again, be sure to include an x-label, y-label, and legend in your plot and use a log-scale on the x-axis. - Give a short (one or two sentences) description of what you see in your plot. Do you think that more data (beyond these 63000 examples) would continue to improve the model's performance?

Note that training a neural network with a lot of data can be **a slow process**. Hence, you should be careful to implement your code such that it runs in a reasonable amount of time. One recommendation is to test your code using only a small subset of the given `m_tr` values, and only run your code with the larger values of `m_tr` once you are certain your code is working. (For reference, it took about 20 minutes to train all models on a quad-core desktop with no GPU.)

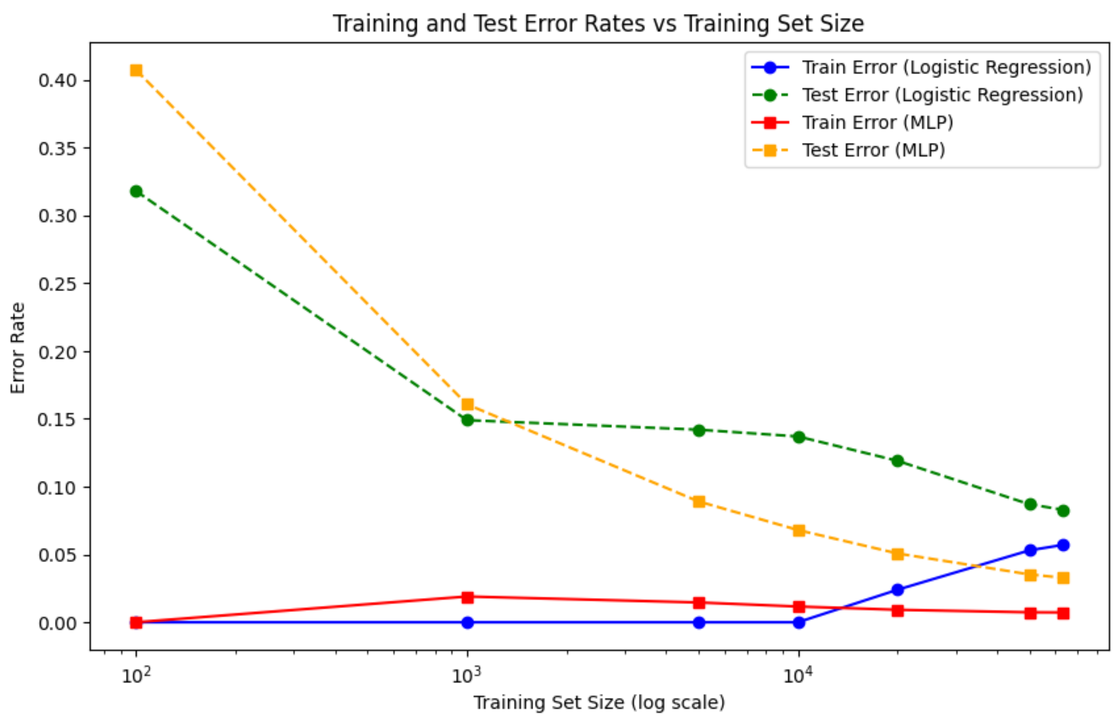
[15]:
```python
import time         # helpful if you want to track execution time
tic = time.time()

train_sizes = [100, 1000, 5000, 10000, 20000, 50000, 63000]

tr_err_mlp = []
te_err_mlp = []

for m_tr in train_sizes:
    ### YOUR CODE STARTS HERE

    mlp = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', solver='sgd', learning_rate_init=0.001, batch_size=256, random_state=seed)
    mlp.fit(X_tr[:m_tr], y_tr[:m_tr])
    y_tr_pred = mlp.predict(X_tr[:m_tr])
    y_te_pred = mlp.predict(X_te)
    tr_err = 1 - accuracy_score(y_tr[:m_tr], y_tr_pred)
    te_err = 1 - accuracy_score(y_te, y_te_pred)
    tr_err_mlp.append(tr_err)
    te_err_mlp.append(te_err)

    print(f'Total elapsed time: {time.time()-tic}')

    ### YOUR CODE ENDS HERE
```

```
Total elapsed time: 0.7713367938995361
Total elapsed time: 3.8418829441070557
Total elapsed time: 17.95932960510254
Total elapsed time: 47.957062005996704
Total elapsed time: 107.29102897644043
Total elapsed time: 261.04254722595215
Total elapsed time: 456.8697319030762
```

[16]:
```python
# When plotting, use these (rounded) values from the similar logistic regression problem solution:
tr_err_lr = np.array([0.   , 0.   , 0.   , 0.   , 0.024, 0.053, 0.057])
te_err_lr = np.array([0.318, 0.149, 0.142, 0.137, 0.119, 0.087, 0.083])
```

[17]:
```python
## YOUR CODE HERE (PLOTTING)
plt.figure(figsize=(10,6))
plt.plot(train_sizes, tr_err_lr, 'o-', label='Train Error (Logistic␣
↪Regression)', color='blue')
plt.plot(train_sizes, te_err_lr, 'o--', label='Test Error (Logistic␣
↪Regression)', color='green')
plt.plot(train_sizes, tr_err_mlp, 's-', label='Train Error (MLP)', color='red')
plt.plot(train_sizes, te_err_mlp, 's--', label='Test Error (MLP)',␣
↪color='orange')
plt.xscale('log')
plt.xlabel('Training Set Size (log scale)')
plt.ylabel('Error Rate')
plt.title('Training and Test Error Rates vs Training Set Size')
plt.legend()
```

[17]: <matplotlib.legend.Legend at 0x16cd22763d0>



[ ]:
```python
# DISCUSS
# MLP starts with a higher error rate on small datasets compared to logistic␣
↪regression, but as the training set size increases, MLP's error rate␣
↪decreases more significantly,
```

```
# outperforming logistic regression on larger datasets. Traning error for LR␣
↪also increases slightly with more data, probably due to overfitting, while␣
↪MLP's training error remains low.
# This indicates that MLP benefits more from larger amounts of data.
```

1.4.3 Problem 2.2: Optimization Curves (10 points)

One hyperparameter that can have a significant effect on the optimization of your model, and thus its performance, is the learning rate, which controls the step size in (stochastic) gradient descent. In this problem you will vary the learning rate to see what effect this has on how quickly training converges as well as the effect on the performance of your model.

In this problem, you should use the following settings for your network: - A single hidden layer with 64 hidden nodes - Use the ReLU activation function - Train the network using stochastic gradient descent (SGD) - Use a batch size of 256 - Set `n_iter_no_change=100` and `max_iter=100`. This ensures that all of your networks in this problem will train for 100 epochs (an *epoch* is one full pass over the training data). - Make sure to set `random_state=seed`.

Your task is to: - Train a neural network with the above settings, but vary the learning rate in `lr = [0.0005, 0.001, 0.005, 0.01]`. - Create a plot showing the training loss as a function of the training epoch (i.e. the x-axis corresponds to training iterations) for each learning rate above. You should have a single plot with four curves. Make sure to include an x-label, a y-label, and a legend in your plot. (Hint: `MLPClassifier` has an attribute `loss_curve_` that you likely find useful.) - Include a short description of what you see in your plot.

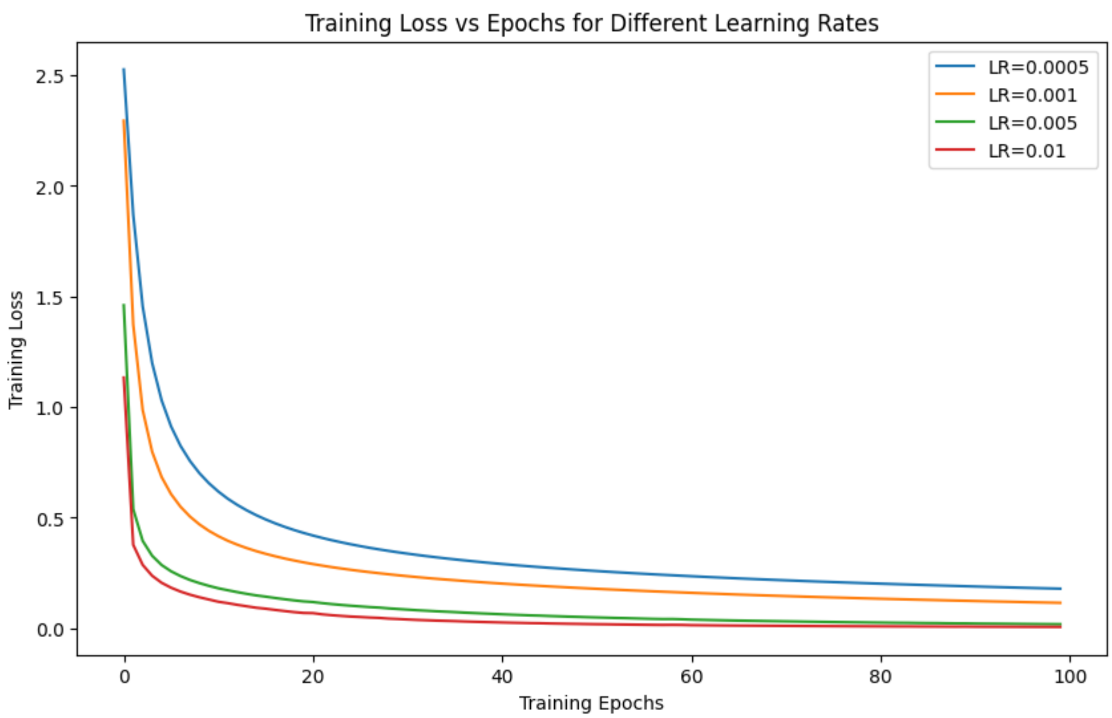
Important: To make your code run faster, you should train all of your networks in this problem on only the first 10,000 images of `X_tr`. In the following cell, you are provided a few lines of code that will create a small training set (with the first 10,000 images in `X_tr`) and a validation set (with the second 10,000 images in `X_tr`). You will use the validation later in Problem 3.3.

[6]:
```
# Create a smaller training set with the first 10,000 images in X_tr
#   along with a validation set from images 10,000 - 20,000 in X_tr

X_val = X_tr[10000:20000] # Validation set
y_val = y_tr[10000:20000]

X_tr = X_tr[:10000]       # From here on, we will only use these smaller sets,
y_tr = y_tr[:10000]       #  so it's OK to discard the rest of the data
```

[ ]:
```
learning_rates = [0.0005, 0.001, 0.005, 0.01]

err_curves = []
plt.figure(figsize=(10,6))

for lr in learning_rates:
    ### YOUR CODE STARTS HERE
```

```
    mlp = MLPClassifier(hidden_layer_sizes=(64,), activation='relu',␣
↪solver='sgd', learning_rate_init=lr, batch_size=256, n_iter_no_change=100,␣
↪max_iter=100, random_state=seed)
    mlp.fit(X_tr, y_tr)
    err_curves.append(mlp.loss_curve_)
    plt.plot(mlp.loss_curve_, label=f'LR={lr}')


    ### YOUR CODE ENDS HERE

plt.xlabel('Training Epochs')
plt.ylabel('Training Loss')
plt.title('Training Loss vs Epochs for Different Learning Rates')
plt.legend()
```

[ ]: <matplotlib.legend.Legend at 0x1c799066c10>



[ ]:
```
# DISCUSS
# MLP with learning rate 0.0005 and 0.001 start with a higher training loss and␣
↪decrease slowly, indicating slower learning.
```

```
# In contrast, learning rates of 0.005 and 0.01 show a rapid decrease in
↪training loss, with 0.01 achieving the lowest loss fastest and staying
↪consistently low.
```

1.4.4 Problem 2.3: Tuning a Neural Network (10 points)

As you saw in Problem 3.2, there are many hyperparameters of a neural network that can possibly be tuned in order to try to maximize the accuracy of your model. For the final problem of this assignment, it is your job to tune these hyperparameters.

For example, some hyperparameters you might choose to tune are: - Learning rate - Depth/width of the hidden layers - Regularization strength - Activation functions - Batch size in stochastic optimization - etc.

To do this, you should train a network on the training data `X_tr` and evaluate its performance on the validation set `X_val` – your goal is to achieve the highest possible accuracy on `X_val` by changing the network hyperparameters. **Important: To make your code run faster, you should train all of your networks in this problem on only the first 10,000 images of `X_tr`.** This was already set up for you in Problem 3.2.

Try to find settings that enable you to achieve an error rate smaller than 5% on the validation data. However, tuning neural networks can be difficult; if you cannot achieve this target error rate, be sure to try at least five different neural networks (corresponding to five different settings of the hyperparameters).

In your answer, include a table listing the different hyperparameters that you tried, along with the resulting accuracy on the training and validation sets `X_tr` and `X_val`. Indicate which of these hyperparameter settings you would choose for your final model, and report the accuracy of this final model on the testing set `X_te`.

[8]:
```
configs = [
    {'hidden_layer_sizes': (64,), 'lrn': 0.1, 'batch_size': 256, 'alpha': 0.
↪0001, 'activation': 'relu'},
    {'hidden_layer_sizes': (128,), 'lrn': 0.001, 'batch_size': 128, 'alpha': 0.
↪0001, 'activation': 'relu'},
    {'hidden_layer_sizes': (64), 'lrn': 0.01, 'batch_size': 64, 'alpha': 0.
↪0001, 'activation': 'relu'},
    {'hidden_layer_sizes': (64,), 'lrn': 0.005, 'batch_size': 256, 'alpha': 0.
↪0001, 'activation': 'relu'},
    {'hidden_layer_sizes': (64,), 'lrn': 0.01, 'batch_size': 256, 'alpha': 0.
↪0005, 'activation': 'relu'},
    {'hidden_layer_sizes': (128,), 'lrn': 0.005, 'batch_size': 128, 'alpha': 0.
↪0005, 'activation': 'tanh'},
    {'hidden_layer_sizes': (64,), 'lrn': 0.1, 'batch_size': 256, 'alpha': 0.
↪0001, 'activation': 'tanh'},
    {'hidden_layer_sizes': (128,), 'lrn': 0.005, 'batch_size': 128, 'alpha': 0.
↪0005, 'activation': 'logistic'},
    {'hidden_layer_sizes': (64,), 'lrn': 0.001, 'batch_size': 256, 'alpha': 0.
↪0001, 'activation': 'logistic'},
```

```
    {'hidden_layer_sizes': (64,), 'lrn': 0.001, 'batch_size': 256, 'alpha': 0.
 ↪0001, 'activation': 'identity'}
]

results = []

for cf in configs:
    mlp = MLPClassifier(hidden_layer_sizes=cf['hidden_layer_sizes'],
                        learning_rate_init=cf['lrn'],
                        batch_size=cf['batch_size'],
                        alpha=cf['alpha'],
                        activation=cf['activation'],
                        n_iter_no_change=100,
                        max_iter=100,
                        random_state=seed)
    mlp.fit(X_tr, y_tr)
    y_val_pred = mlp.predict(X_val)
    tr_err = 1 - accuracy_score(y_tr, mlp.predict(X_tr))
    val_err = 1 - accuracy_score(y_val, y_val_pred)
    results.append((cf, tr_err, val_err))

# Display the results in a table
for i, (cf, tr_err, val_err) in enumerate(results):
    print(f"Config {i+1}: {cf}, Train Error: {tr_err:.4f}, Validation Error:␣
 ↪{val_err:.4f}")
```

```
Config 1: {'hidden_layer_sizes': (64,), 'lrn': 0.1, 'batch_size': 256, 'alpha':
0.0001, 'activation': 'relu'}, Train Error: 0.0035, Validation Error: 0.0591
Config 2: {'hidden_layer_sizes': (128,), 'lrn': 0.001, 'batch_size': 128,
'alpha': 0.0001, 'activation': 'relu'}, Train Error: 0.0000, Validation Error:
0.0486
Config 3: {'hidden_layer_sizes': 64, 'lrn': 0.01, 'batch_size': 64, 'alpha':
0.0001, 'activation': 'relu'}, Train Error: 0.0001, Validation Error: 0.0520
Config 4: {'hidden_layer_sizes': (64,), 'lrn': 0.005, 'batch_size': 256,
'alpha': 0.0001, 'activation': 'relu'}, Train Error: 0.0000, Validation Error:
0.0489
Config 5: {'hidden_layer_sizes': (64,), 'lrn': 0.01, 'batch_size': 256, 'alpha':
0.0005, 'activation': 'relu'}, Train Error: 0.0000, Validation Error: 0.0516
Config 6: {'hidden_layer_sizes': (128,), 'lrn': 0.005, 'batch_size': 128,
'alpha': 0.0005, 'activation': 'tanh'}, Train Error: 0.0000, Validation Error:
0.0691
Config 7: {'hidden_layer_sizes': (64,), 'lrn': 0.1, 'batch_size': 256, 'alpha':
0.0001, 'activation': 'tanh'}, Train Error: 0.0683, Validation Error: 0.1144
Config 8: {'hidden_layer_sizes': (128,), 'lrn': 0.005, 'batch_size': 128,
'alpha': 0.0005, 'activation': 'logistic'}, Train Error: 0.0000, Validation
Error: 0.0605
Config 9: {'hidden_layer_sizes': (64,), 'lrn': 0.001, 'batch_size': 256,
'alpha': 0.0001, 'activation': 'logistic'}, Train Error: 0.0005, Validation
```

```
Error: 0.0728
Config 10: {'hidden_layer_sizes': (64,), 'lrn': 0.001, 'batch_size': 256,
'alpha': 0.0001, 'activation': 'identity'}, Train Error: 0.0104, Validation
Error: 0.1196
```

[20]:
```python
# I would choose Config 2: {'hidden_layer_sizes': (128,), 'lrn': 0.001,
↪'batch_size': 128, 'alpha': 0.0001, 'activation': 'relu'}, Train Error: 0.
↪0000, Validation Error: 0.0486
# As it provides the lowest validation error while maintaining a perfect
↪training accuracy, indicating good generalization without overfitting.

mlp_final = MLPClassifier(hidden_layer_sizes=(128,), learning_rate_init=0.001,
↪batch_size=128, alpha=0.0001, activation='relu', n_iter_no_change=100,
↪max_iter=100, random_state=seed)
mlp_final.fit(X_tr, y_tr)
y_te_pred = mlp_final.predict(X_te)
te_err_final = np.mean(y_te != y_te_pred)
print(f'Final Model Test Error: {te_err_final:.4f}')
```

```
Final Model Test Error: 0.0497
```

1.5 Problem 3: Torch and Convolutional Networks

In this problem, we will train a small convolutional neural network and compare it to the "standard" MLP model you built in Problem 2. Since `scikit` does not support CNNs, we will implement a simple CNN model using `torch`.

The `torch` library may take a while to install if it is not yet on your system. It should be pre-installed on ICS Jupyter Hub (`hub.ics.uci.edu`) and Google CoLab, if you prefer to use those.

1.5.1 Problem 3.0: Defining the CNN

First, we need to define a CNN model. This is done for you; it consists of one convolutional layer, a pooling layer to down-sample the hidden nodes, and a standard fully-connected or linear layer. It contains methods to calculate the 0/1 loss as well as the negative log-likelihood, and trains using the Adam variant of SGD.

It can (optionally) output a real-time plot of the training process at each epoch, if you would like to assess how it is doing.

[22]:
```python
import torch
torch.set_default_dtype(torch.float64)

class myConvNet(object):
    def __init__(self):
        # Initialize parameters: assumes data size! 28x28 and 10 classes
        self.conv_ = torch.nn.Conv2d(1, 16, (5,5), stride=2)  # Be careful when
↪declaring sizes;
```

```
        self.pool_ = torch.nn.MaxPool2d(3, stride=2)          # inconsistent␣
↪sizes will give you
        self.lin_ = torch.nn.Linear(400,10)                   # hard-to-read␣
↪error messages.

    def forward_(self,X):
        """Compute NN forward pass and output class probabilities (as tensor)␣
↪"""
        r1 = self.conv_(X)              # X is (m,1,28,28); R is (m,16,24,24)/2␣
↪= (m,16,12,12)
        h1 = torch.relu(r1)             #
        h1_pooled = self.pool_(h1)      # H1 is (m,16,12,12), so H1p is␣
↪(m,16,10,10)/2 = (m,16,5,5)
        h1_flat = torch.nn.Flatten()(h1_pooled)  # and H1f is (m,400)
        r2 = self.lin_(h1_flat)
        f  = torch.softmax(r2,axis=1)   # Output is (m,10)
        return f

    def parameters(self):
        return list(self.conv_.parameters())+list(self.pool_.
↪parameters())+list(self.lin_.parameters())

    def predict(self,X):
        """Compute NN class predictions (as array) """
        m,n = X.shape
        Xtorch = torch.tensor(X).reshape(m,1,int(np.sqrt(n)),int(np.sqrt(n)))
        return self.classes_[np.argmax(self.forward_(Xtorch).detach().
↪numpy(),axis=1)]    # pick the most probable class

    def J01(self,X,y):   return (y != self.predict(X)).mean()
    def JNLL_(self,X,y): return -torch.log(self.forward_(X)[range(len(y)),y.
↪astype(int)]).mean()

    def fit(self, X,y, batch_size=256, max_iter=100, learning_rate_init=.005,␣
↪momentum=0.9, alpha=.001, plot=False):
        self.classes_ = np.unique(y)
        m,n = X.shape
        Xtorch = torch.tensor(X).reshape(m,1,int(np.sqrt(n)),int(np.sqrt(n)))
        self.loss01, self.lossNLL = [self.J01(X,y)], [float(self.
↪JNLL_(Xtorch,y))]

        optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate_init)
        for epoch in range(max_iter):                          # 1 epoch = pass␣
↪through all data
            pi = np.random.permutation(m)                      # per epoch:␣
↪permute data order
```

```
        for ii,i in enumerate(range(0,m,batch_size)):    # & split into␣
↪mini-batches
            ivals = pi[i:i+batch_size]
            optimizer.zero_grad()                            # Reset the␣
↪gradient computation
            Ji = self.JNLL_(Xtorch[ivals,:,:,:],y[ivals])
            Ji.backward()
            optimizer.step()
        self.loss01.append(self.J01(X,y))                    # track 0/1 and␣
↪NLL losses
        self.lossNLL.append(float(self.JNLL_(Xtorch,y)))

        if plot:                                             # optionally␣
↪visualize progress
            display.clear_output(wait=True)
            plt.plot(range(epoch+2),self.loss01,'b-',range(epoch+2),self.
↪lossNLL,'c-')
            plt.title(f'J01: {self.loss01[-1]}, NLL: {self.lossNLL[-1]}')
            plt.xlabel('Epochs')
            plt.ylabel('Loss')
            plt.draw(); plt.pause(.01);
```

1.5.2 Problem 3.1: CNN model structure (10 points)

How many (trainable) parameters are specified in the convolutional network? (If you like, you can count them – you can access them through each trainable element, e.g., `myConvNet().conv_._parameters['weights']` and `..._parameters['bias']`). List how many from each layer, and the total.

[ ]:
```
conv_weights = myConvNet().conv_._parameters['weight'].numel()  # 16 filters of␣
↪size 5x5x1
conv_bias = myConvNet().conv_._parameters['bias'].numel()       # 16 biases
lin_weights = myConvNet().lin_._parameters['weight'].numel()    # 10 outputs,␣
↪each with 400 inputs
lin_bias = myConvNet().lin_._parameters['bias'].numel()         # 10 biases
total_params = conv_weights + conv_bias + lin_weights + lin_bias
print(f'Convolutional Layer Weights: {conv_weights}')
print(f'Convolutional Layer Biases: {conv_bias}')
print(f'Linear Layer Weights: {lin_weights}')
print(f'Linear Layer Biases: {lin_bias}')
print(f'Total Trainable Parameters: {total_params}')
```

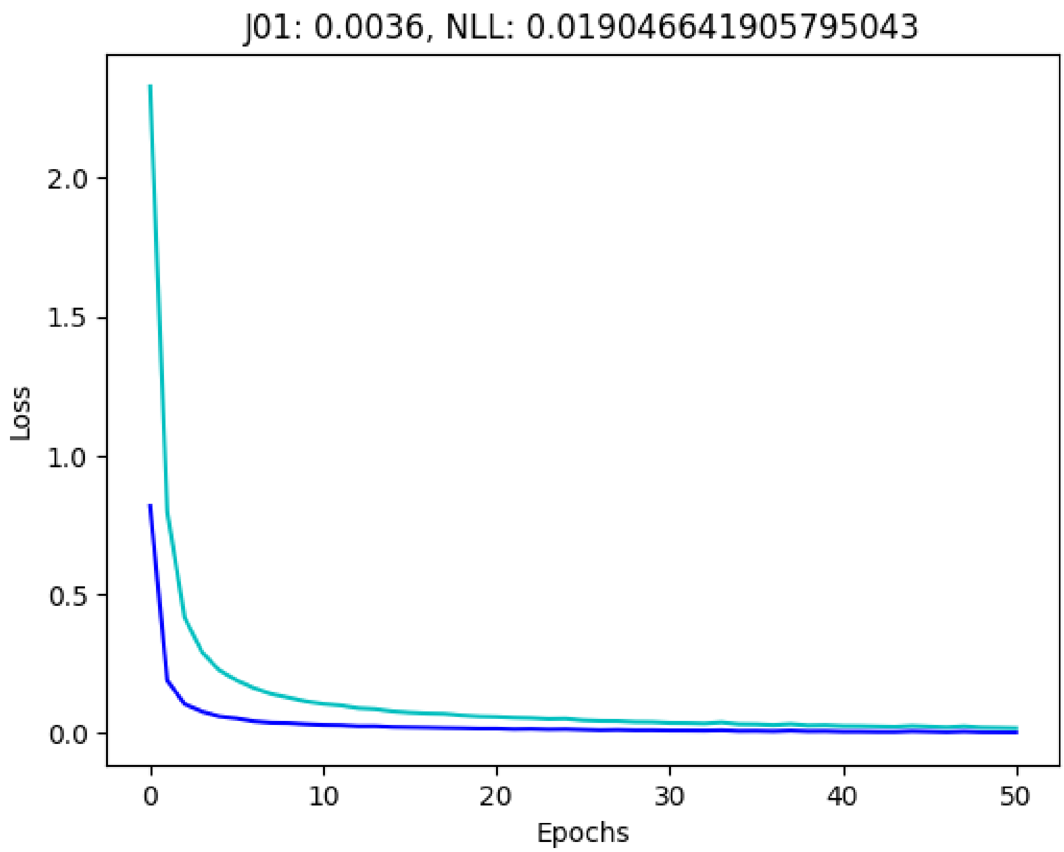
```
Convolutional Layer Weights: 400
Convolutional Layer Biases: 16
Linear Layer Weights: 4000
Linear Layer Biases: 10
Total Trainable Parameters: 4426
```

1.5.3 Problem 3.2: Training the model (10 points)

Now train your model on `X_tr`. (Note that this should now include only 10k data points.) If you like you can plot while training to monitor its progress. Train for 50 epochs, using a learning rate of .001.

(Note that my simple training process takes these arguments directly into `fit`, rather than being part of the model properties as is typical in `scikit`.)

[32]:
```
mycnn = myConvNet()

mycnn.fit(X_tr, y_tr, batch_size=256, max_iter=50, learning_rate_init=0.001,␣
 ↪plot=True)
```



1.5.4 Problem 3.3: Evaluation and Discussion (5 points)

Evaluate your CNN model's training, validation, and test error. Compare these to the values you got after optimizing your model's training process in Problem 2.3 (Tuning). Why do you think these differences occur? (Note that your answer may depend on how well your model in P2.3 did, of course.)

```
[37]: tr_err_conv = mycnn.J01(X_tr, y_tr)
val_err_conv = mycnn.J01(X_val, y_val)
te_err_conv = mycnn.J01(X_te, y_te)
print(f'Convolutional Network Train Error: {tr_err_conv:.4f}')
print(f'Validation Error: {val_err_conv:.4f}')
print(f'Test Error: {te_err_conv:.4f}')
```

```
Convolutional Network Train Error: 0.0036
Validation Error: 0.0269
Test Error: 0.0286
```

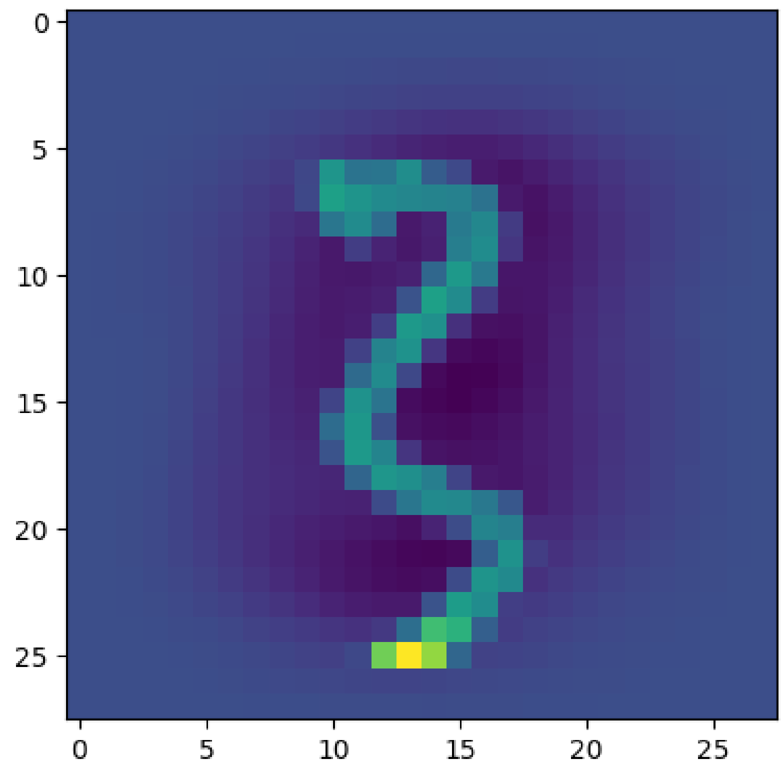
```
[ ]: # The results show that the convolutional network achieves a lower validataion␣
↪and test error compared to the best MLP model
# I think this is because the convolutional network is better suited for this␣
↪data type as it can capture spatial hierarchies in images more effectively␣
↪than a standard MLP.
```

1.5.5 Problem 3.4: Comparing Predictions (5 points)

Consider the "somewhat ambiguous" data point `X_val[592]`. Display the data point (it will look a bit weird since it is already normalized). Then, use your trained `MLPClassifier` model to predict the class probabilities. If there are other classes with non-negligible probability, are they plausible? Similarly, find the class probabilities predicted by your CNN model. Compare the two models' uncertainty.

```
[ ]: plt.imshow(X_val[592].reshape(28,28))
y_val_592_mlp_probs = mlp_final.predict_proba(X_val[592].reshape(1,-1))
y_val_592_cnn_probs = mycnn.forward_(torch.tensor(X_val[592].
↪reshape(1,1,28,28))).detach().numpy()
print("MLP Class Probabilities:", y_val_592_mlp_probs)
print("CNN Class Probabilities:", y_val_592_cnn_probs)
```

```
MLP Class Probabilities: [[1.39961095e-03 5.34368410e-03 4.21983820e-02
1.87217652e-01
  2.02463219e-09 3.56159088e-02 1.39960152e-02 1.52629637e-01
  3.25139196e-01 2.36459912e-01]]
CNN Class Probabilities: [[4.02330999e-05 1.63470236e-06 1.29745911e-02
9.74088256e-01
  9.22011495e-07 2.83147150e-05 2.87275542e-05 1.83957174e-03
  1.09641243e-02 3.36251394e-05]]
```



Based on the displayed image, which is a three, the MLP model struggles to classify the number correctly. I has a higher probability of the number being 8 or 9 than 3. The CNN performs very well and it is near 100% confident that it is a 3. Both these models were trained on the same about of data, and the probability just shows how much better CNN is than MPL at image classification.

<img src="data:image/svg+xml,%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-8%22%20standa

1.5.6 Statement of Collaboration (5 points)

It is **mandatory** to include a Statement of Collaboration in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using EdD) to discuss the task descriptions, requirements, bugs in my code, and the relevant technical content before they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to EdD, etc.). Especially after you have started working on the assignment, try to restrict the discussion to EdD as much as possible, so that there is no doubt as to the extent of your collaboration.

I have not collaborated with anyone on this assignment.