CS 178: Machine Learning & Data Mining

Homework 3: Due Friday, 23 February 2024 (11:59 PM)

Instructions

This homework (and many 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 modify these starter Jupyter notebooks to complete your assignment and to write your report. You may add additional cells (containing either code or text) as needed. 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***). Before submitting, ensure that your submission is complete, all text and code is legible (i.e. not cut off), and the pdf includes page breaks (i.e. not one very long page). We 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.

If you have any questions/concerns about using Jupyter notebooks, ask us on Piazza. There you can also find additional instructions on how to convert to a .pdf.

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 in Code (65 points)
  - Problem 2.1: Setting up Data (5 points)
  - Problem 2.2: Vary Amount of Data (20 points)
  - Problem 2.3: Learning Curves (10 points)
  - Problem 2.3: Tuning your Neural Network (30 points)
- Statement of Collaboration (5 points)

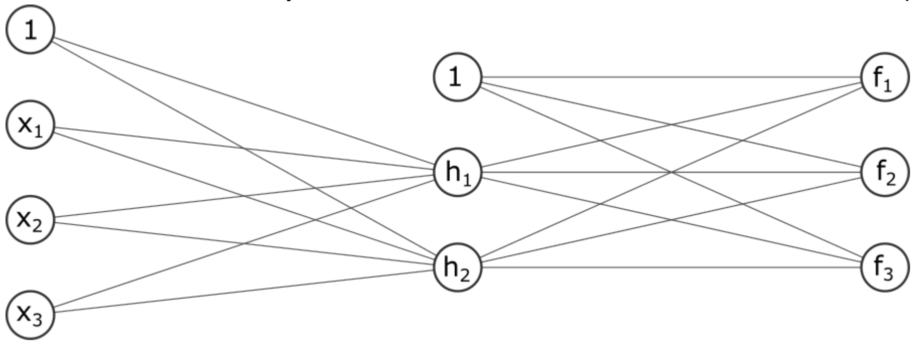
Before we get started, let's import some libraries that you will make use of in this assignment. Make sure that you run the code cell below in order to import these libraries.

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.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

import warnings
warnings.filterwarnings('ignore')

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

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 $\mathbf{x}$, as well as weights W for the hidden layer and weights B for the output layer. For example, w_{12} is the weight connecting input x_1 to hidden unit h_2. This network uses the ReLU activation function for the hidden layer, and uses the softmax activation function for the output layer.



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

$$W = \begin{bmatrix} w_{01} & w_{11} & w_{21} & w_{31} \\ w_{02} & w_{12} & w_{22} & w_{32} \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & 6 \\ 2 & 1 & 1 & 3 \end{bmatrix}$$

$$B = \begin{bmatrix} \beta_{01} & \beta_{11} & \beta_{21} \\ \beta_{02} & \beta_{12} & \beta_{22} \\ \beta_{03} & \beta_{13} & \beta_{23} \end{bmatrix} = \begin{bmatrix} 6 & -1 & 0 \\ 5 & 0 & 2 \\ 2 & 1 & 1 \end{bmatrix}$$

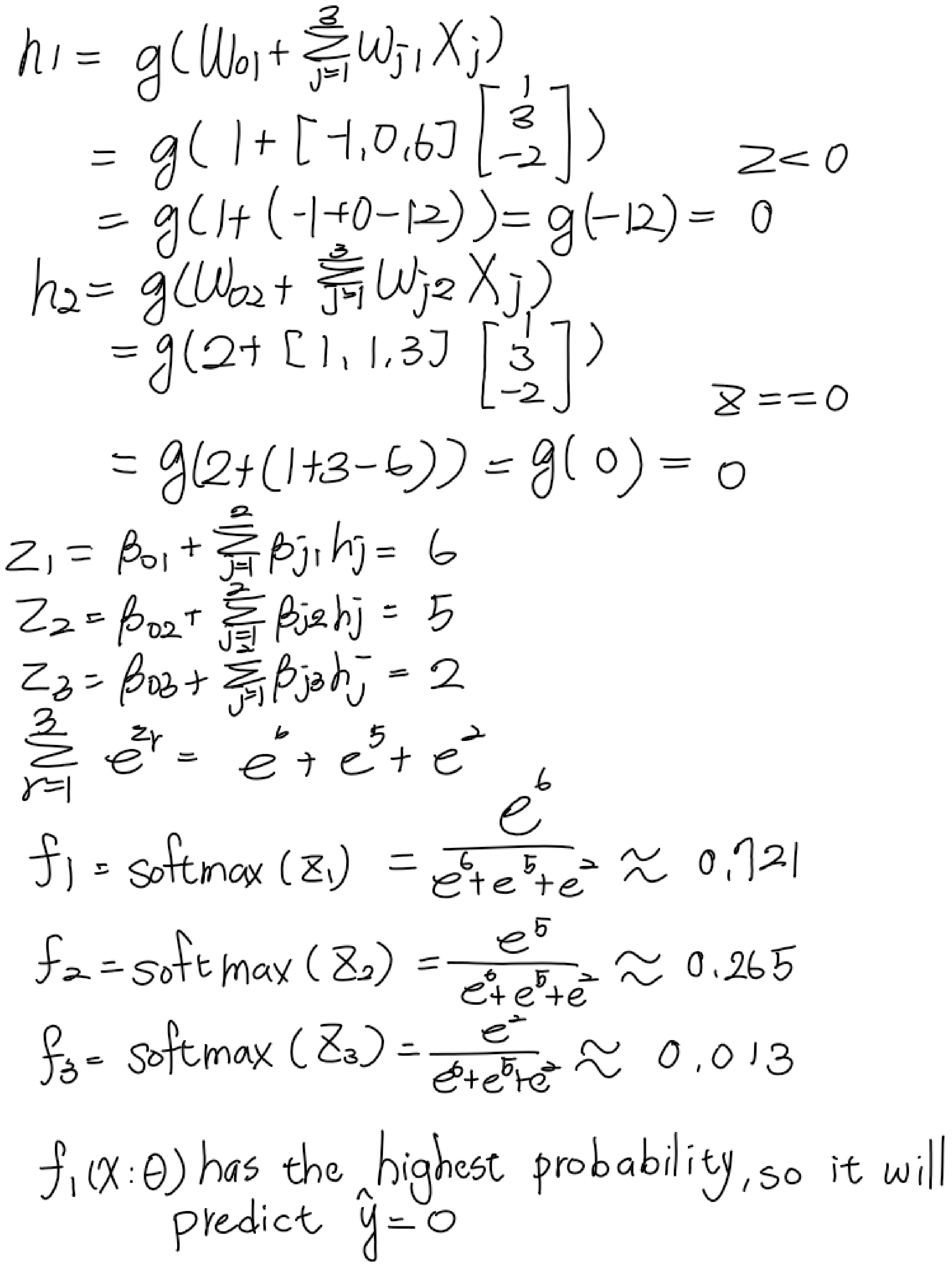
Problem 1.1 (10 points): Forward Pass

- Given the inputs and weights above, compute the values of the hidden units h_1, h_2 and the outputs f_1, f_2, 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.**

In [2]:
```
from IPython.display import Image, display
```

In [4]:
```
display(Image(filename="hw3-1.1.png"))
```



$$h_1 = g(W_{01} + \sum_{j=1}^{3} W_{j1} X_j)$$
$$= g(1 + [-1, 0, 6] \begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix}) \qquad z < 0$$
$$= g(1 + (-1 + 0 - 12)) = g(-12) = 0$$
$$h_2 = g(W_{02} + \sum_{j=1}^{3} W_{j2} X_j)$$
$$= g(2 + [1, 1, 3] \begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix}) \qquad z == 0$$
$$= g(2 + (1 + 3 - 6)) = g(0) = 0$$
$$z_1 = \beta_{01} + \sum_{j=1}^{2} \beta_{j1} h_j = 6$$
$$z_2 = \beta_{02} + \sum_{j=1}^{2} \beta_{j2} h_j = 5$$
$$z_3 = \beta_{03} + \sum_{j=1}^{2} \beta_{j3} h_j = 2$$
$$\sum_{r=1}^{3} e^{z_r} = e^6 + e^5 + e^2$$
$$f_1 = \text{softmax}(z_1) = \frac{e^6}{e^6 + e^5 + e^2} \approx 0.721$$
$$f_2 = \text{softmax}(z_2) = \frac{e^5}{e^6 + e^5 + e^2} \approx 0.265$$
$$f_3 = \text{softmax}(z_3) = \frac{e^2}{e^6 + e^5 + e^2} \approx 0.013$$

$f_1(x:\theta)$ has the highest probability, so it will predict $\hat{y} = 0$

- 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.

In [5]:
```
sumExp = np.exp(6)+np.exp(5)+np.exp(2)
loss_y1 = -np.log(np.exp(5)/sumExp)
print(f"When the true label for the input x is  $y$=1, the value of loss function
loss_y2 = -np.log(np.exp(2)/sumExp)
print(f"When the true label for the input x is  $y$=2, the value of loss function
```

```
When the true label for the input x is  y=1, the value of loss function is 1.32
65626412674703.
When the true label for the input x is  y=2, the value of loss function is 4.32
656264126747.
```

Problem 1.3 (10 points): Network Size

- Suppose we change our network so that there are 12 hidden units instead of 2. How many total weights and biases would there be in our new network?

W will increase from (2, 4) to (12, 4)

B will increase from (3,3) to (12+1(bias unit), 3) so total is 4*12 + 13*3 = 87 weights

Total is 1*12 + 1*3 = 15 biases

Problem 2: Neural Networks in Code

In the second problem of this 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 (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html) for the corresponding documentation. Although there are specialized Python libraries for neural networks, like TensorFlow (https://www.tensorflow.org/) and PyTorch (https://pytorch.org/), we'll stick with scikit-learn as you're already familiar with this library.

In this problem, we'll be working with the MNIST dataset, which we already saw in Homework 1. As a reminder, this is an image classification dataset, where each image is a hand-written digit. Take a look at Homework 1 to remind yourself what this dataset looks like.

Problem 2.1: Setting up the data (5 points)

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

- Use the scikit-learn class `StandardScaler` to standardize both the training and testing features. Remember that you should only fit the `StandardScaler` on the training data, and *not* the testing data.

In [2]:
```
# 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)
```

In [3]:
```
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.1, random_state=seed, shuffle=True
scaler = StandardScaler()
X_tr = scaler.fit_transform(X_tr)
X_te = scaler.transform(X_te)
```

Problem 2.2: Varying the amount of training data (20 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 learning rate of 0.001
- Use a batch size of 256
- **Make sure to set `random_state=seed` .**

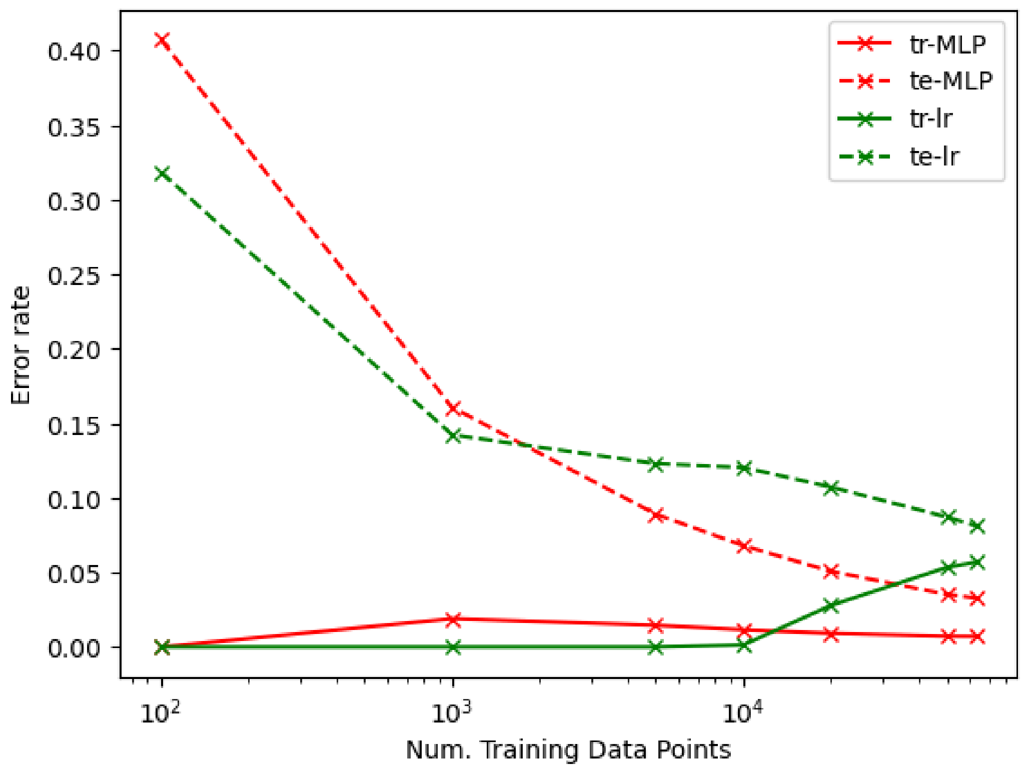
Your task is to implement the following:

- Train an MLP model (with the above hyperparameter settings) using the first `n_tr` feature vectors in `X_tr` , where `n_tr = [100, 1000, 5000, 10000, 20000, 50000, 63000]` . You should use the `MLPClassifier` class from scikit-learn in your implementation.
- Train a logistic regression classifier (with the default settings in sklearn) using the first `n_tr` feature vectors in `X_tr` , where `n_tr = [100, 1000, 5000, 10000, 20000, 50000, 63000]` .You should use the `LogisticRegression` class from scikit-learn in your implementation. **Make sure to use the argument `random_state=seed` for reproducibility.**
- Create a plot of the training error and testing error for both the logistic regression and MLP models as a function of the number of training data points. Be sure to include an x-label, y-label, and legend in your plot. Use a log-scale on the x-axis. Give a short (one or two sentences) description of what you see in your plot.

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 `n_tr` values, and only run your code with all of the `n_tr` values given once you are certain your code is working.

In [7]:
```
figure, axes = plt.subplots(1)
n_tr = [100, 1000 , 5000, 10000, 20000, 50000, 63000]
MLPtrain_error = []
MLPtest_error = []
for i in n_tr:
    MLPclf = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', solver= 'sgd', learning
    y_trpred = MLPclf.predict(X_tr[:i])
    traccuracy = accuracy_score(y_tr[:i], y_trpred)
    MLPtrain_error.append(1-traccuracy)

    y_tepred = MLPclf.predict(X_te)
    teaccuracy = accuracy_score(y_te, y_tepred)
    MLPtest_error.append(1-teaccuracy)
    #print(f"{i}: is finished")

LRtrain_error = []
LRtest_error = []
for i in n_tr:
    LRclf = LogisticRegression(random_state=seed).fit(X_tr[:i], y_tr[:i])
    y_trpred = LRclf.predict(X_tr[:i])
    traccuracy = accuracy_score(y_tr[:i], y_trpred)
    LRtrain_error.append(1-traccuracy)

    y_tepred = LRclf.predict(X_te)
    teaccuracy = accuracy_score(y_te, y_tepred)
    LRtest_error.append(1-teaccuracy)
    #print(f"{i}: is finished")


axes.semilogx(n_tr , MLPtrain_error,label='tr-MLP', color="red", marker='x')
axes.semilogx(n_tr, MLPtest_error,label='te-MLP', color="red", linestyle='--', marker='x')
axes.semilogx(n_tr ,LRtrain_error,label='tr-lr', color="green", marker='x')
axes.semilogx(n_tr,LRtest_error,label='te-lr', color="green", linestyle='--', marker='x')
axes.set_xlabel('Num. Training Data Points')
axes.set_ylabel('Error rate')
axes.legend()
```

Out[7]:
```
<matplotlib.legend.Legend at 0x222215a4b10>
```



Problem 2.3: Learning Curves (10 points)

One hyperparameter that can have a significant effect on the performance of your model 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)
- 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).
- Use a batch size of 256
- **Make sure to set `random_state=seed`.**

Your task is to:

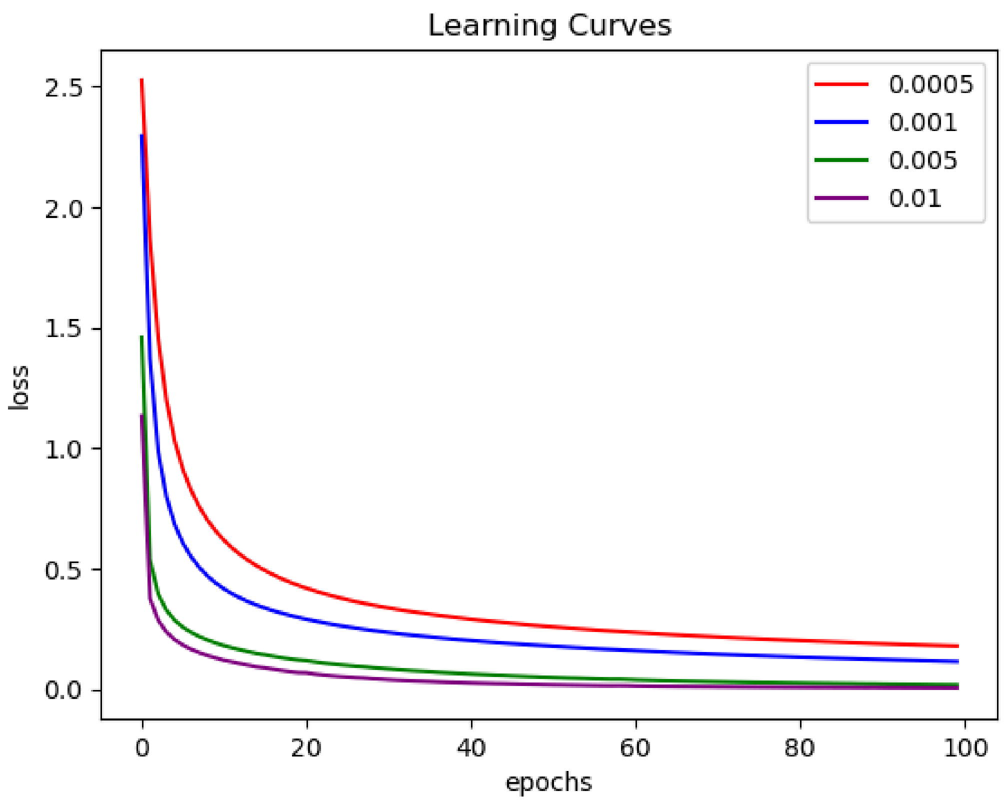
- Train a neural network with the above settings, but vary the learning rate in `lr = [0.0005, 0.001, 0.005, 0.01]`.
- Create a plot showing the loss on the training set as a function of the training epoch (i.e. the x-axis corresponds to training iterations) for each learning rate above. You should have 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 2.4.

In [4]:
```python
# Create validation sets from the second 10k images in X_tr
X_val = X_tr[10000:20000]
y_val = y_tr[10000:20000]

# Create a smaller training set with the first 10k images in X_tr
X_tr = X_tr[:10000]
y_tr = y_tr[:10000]
```

In [109]:
```
figure, axes = plt.subplots(1)
lr = [0.0005, 0.001, 0.005, 0.01]
colors= ['red', 'blue', 'green', 'purple']
for i in range(4):
    clf = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', solver= 'sgd',n_iter_no_ch
                        learning_rate= 'constant' ,learning_rate_init = lr[i], batch_size=256, 
                         .fit(X_tr, y_tr)
    axes.plot(clf.loss_curve_ ,label= f'{lr[i]}', color= colors[i])
axes.set_xlabel('epochs')
axes.set_ylabel('loss')
axes.set_title('Learning Curves')
axes.legend()
```

Out[109]:
```
<matplotlib.legend.Legend at 0x237b90fa410>
```



From this graph, a higher learning rate has a lower loss value at iteration 0, and it causes the loss value to decrease faster.

Problem 2.4: Tuning a Neural Network (30 points)

As you saw in Problem 2.2, 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
- Regularization strength
- Activation functions
- Batch size
- 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 2.3.

To receive full credit for this problem, you will need to tune your network hyperparameters until you achieve an error rate smaller than 5% on the validation data. However, tuning neural networks can be a difficult task, and you may not be able to achieve this target error rate. Hence, you will receive most of the credit for this problem as long as you train at least ten different neural networks with 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`.

In [17]:
```
test1 = ['1', '64' ,'0.01', 'relu', '256', '0.0001']
clf = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', solver= 'sgd', n_iter_no_chang
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=256, ra
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test1.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test1.append(str(valaccuracy))
print(valaccuracy)
```

```
1.0
0.9417
```

In [18]:
```
test2 = ['1', '64' ,'0.01', 'relu', '128', '0.0001']
clf = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', solver= 'sgd', n_iter_no_chang
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=128, ra
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test2.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test2.append(str(valaccuracy))
print(valaccuracy)
```

```
1.0
0.9445
```

In [19]:
```
test3 = ['1', '64' ,'0.01', 'logistic', '64', '0.0001']
clf = MLPClassifier(hidden_layer_sizes=(64,), activation= 'logistic', solver= 'sgd', n_iter_no_
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=64, ra
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test3.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test3.append(str(valaccuracy))
print(valaccuracy)
```

```
0.9996
0.9315
```

In [20]:
```
test4 = ['1', '128' ,'0.01', 'logistic', '32', '0.005']
clf = MLPClassifier(hidden_layer_sizes=(128,), activation= 'logistic', solver= 'sgd', n_iter_no
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size= 32, ra
                    alpha=0.005).fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test4.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test4.append(str(valaccuracy))
print(valaccuracy)
```

```
1.0
0.9396
```

In [21]:
```
test5 = ['1', '64' ,'0.01', 'relu', '32', '0.0001']
clf = MLPClassifier(hidden_layer_sizes=(64,), activation='relu', solver= 'sgd', n_iter_no_chang
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=32, rar
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test5.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test5.append(str(valaccuracy))
print(valaccuracy)
```

```
1.0
0.9453
```

In [22]:
```
test6 = ['1', '128' ,'0.01', 'relu', '32', '0.005']
clf = MLPClassifier(hidden_layer_sizes=(128,), activation='relu', solver= 'sgd', n_iter_no_char
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=32, rar
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test6.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test6.append(str(valaccuracy))
print(valaccuracy)
```

```
1.0
0.9508
```

In [23]:
```python
test7 = ['1', '150' ,'0.01', 'relu', '32', '0.05']
clf = MLPClassifier(hidden_layer_sizes=(150,), activation='relu', solver= 'sgd', n_iter_no_char
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=32, rar
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test7.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test7.append(str(valaccuracy))
print(valaccuracy)
```

```
1.0
0.9583
```

In [24]:
```python
test8 = ['1', '256' ,'0.01', 'relu', '32', '0.1']
clf = MLPClassifier(hidden_layer_sizes=(256,), activation='relu', solver= 'sgd', n_iter_no_char
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=32, rar
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test8.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test8.append(str(valaccuracy))
print(valaccuracy)
```

```
0.9993
0.9571
```

In [25]:
```python
test9 = ['1', '64' ,'0.01', 'tanh', '32', '0.05']
clf = MLPClassifier(hidden_layer_sizes=(64,), activation='tanh', solver= 'sgd', n_iter_no_chang
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=32, rar
                        .fit(X_tr, y_tr)

y_trpred = clf.predict(X_tr)
traccuracy = accuracy_score(y_tr, y_trpred)
test9.append(str(traccuracy))
print(traccuracy)
y_valpred = clf.predict(X_val)
valaccuracy = accuracy_score(y_val, y_valpred)
test9.append(str(valaccuracy))
print(valaccuracy)
```

```
0.9997
0.9408
```

```
In [26]:  test10 = ['1', '64' ,'0.01', 'tanh', '64', '0.1']
          clf = MLPClassifier(hidden_layer_sizes=(64,), activation='tanh', solver= 'sgd', n_iter_no_chang
                                 learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=64, ran
                                  .fit(X_tr, y_tr)

          y_trpred = clf.predict(X_tr)
          traccuracy = accuracy_score(y_tr, y_trpred)
          test10.append(str(traccuracy))
          print(traccuracy)
          y_valpred = clf.predict(X_val)
          valaccuracy = accuracy_score(y_val, y_valpred)
          test10.append(str(valaccuracy))
          print(valaccuracy)
```

```
0.9996
0.9382
```

```
In [27]:  data = [
              ['Test 1']+test1,
              ['Test 2']+test2,
              ['Test 3']+test3,
              ['Test 4']+test4,
              ['Test 5']+test5,
              ['Test 6']+test6,
              ['Test 7']+test7,
              ['Test 8']+test8,
              ['Test 9']+test9,
              ['Test 10']+test10
          ]

          # Define headers

          headers = ["Test#", "# Layer","# Hidden nodes","Learning Rate", "Activation functions", "Batch

          # Generate Markdown table
          markdown_table = "| " + " | ".join(headers) + " |\n"
          markdown_table += "| " + " | ".join(["---"] * len(headers)) + " |\n"

          for row in data:
              markdown_table += "| " + " | ".join(row) + " |\n"


          from IPython.display import display, Markdown

          display(Markdown(markdown_table))
```

Test#	# Layer	# Hidden nodes	Learning Rate	Activation functions	Batch size	Regularization strength	Accuracy on the training	Accuracy on the validation
Test 1	1	64	0.01	relu	256	0.0001	1.0	0.9417
Test 2	1	64	0.01	relu	128	0.0001	1.0	0.9445
Test 3	1	64	0.01	logistic	64	0.0001	0.9996	0.9315
Test 4	1	128	0.01	logistic	32	0.005	1.0	0.9396
Test 5	1	64	0.01	relu	32	0.0001	1.0	0.9453
Test 6	1	128	0.01	relu	32	0.005	1.0	0.9508
Test 7	1	150	0.01	relu	32	0.05	1.0	0.9583
Test 8	1	256	0.01	relu	32	0.1	0.9993	0.9571
Test 9	1	64	0.01	tanh	32	0.05	0.9997	0.9408
Test 10	1	64	0.01	tanh	64	0.1	0.9996	0.9382

Test 7 has an error rate smaller than 5% on the validation data, which is 95.83% accuracy rate. Using these hyperparameters to train the data , we can get the accuracy of testing set: 95.43%.

In [28]:
```
#test 7
clf = MLPClassifier(hidden_layer_sizes=(150,), activation='relu', solver= 'sgd', n_iter_no_char
                       learning_rate= 'constant' ,learning_rate_init = 0.01, batch_size=32, rai
                        .fit(X_tr, y_tr)

y_tepred = clf.predict(X_te)
teaccuracy = accuracy_score(y_te, y_tepred)
print(f"Accuracy of testing set: {teaccuracy}")
```

```
Accuracy of testing set: 0.9542857142857143
```

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. If you did not collaborate you can simply write N/A.

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 Piazza) 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 Piazza, etc.). Especially after you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.

N/A