A4 Neural Network Classifier, or *Fun With Handwritten Digits*!

Requirement 1

For this assignment, you will be adding code to the python script file `neuralnetworksA4.py` that you will download from here. The file `neuralnetworksA4_initial.py` currently contains the implementation of the `NeuralNetwork` class that is a solution to A3. It also contains an incomplete implementation of the subclass `NeuralNetworkClassifier` that extends `NeuralNetwork` as discussed in class. Copy or rename this file to `neuralnetworksA4.py` and complete the implementation of `NeuralNetworkClassifier`. Your `NeuralNetworkClassifier` implementation should rely on inheriting functions from `NeuralNetwork` as much as possible. Your `neuralnetworksA4.py` file (notice it is plural) will now contain two classes, `NeuralNetwork` and `NeuralNetworkClassifier`. The tar file `neuralnetworksA4.tar` also contains `optimizers.py`, the version of our optimizer code that you must use in this assignment.

In `NeuralNetworkClassifier` you will replace the `_error_f` function with one called `_neg_log_likelihood_f`. You will also have to define a new version of the `_gradient_f` function for `NeuralNetworkClassifier`.

Here are some example tests.

```
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
import numpy as np
import matplotlib.pyplot as plt
```

Import your completed `neuralnetworksA4.py` code that defines `NeuralNetwork` and `NeuralNetworkClassifier` classes.

```
import neuralnetworksA4_v6 as nn

X = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])
T = np.array([[0], [1], [1], [0]])
X, T
```

```
(array([[0, 0],
        [1, 0],
        [0, 1],
        [1, 1]]),
 array([[0],
```

```
        [1],
        [1],
        [0]]))

np.random.seed(111)
nnet = nn.NeuralNetworkClassifier(2, [10], 2)

print(nnet)

NeuralNetworkClassifier with 1 hidden layers.

nnet.Ws

[array([[ 0.12952296, -0.38212533, -0.07383268,  0.31091752, -
0.23633798,
         -0.40511172, -0.55139454, -0.09211682, -0.30174387, -
0.18745848],
        [ 0.56662595, -0.3028474 , -0.48359706,  0.19583749,
0.13999926,
         -0.26066957, -0.03900416, -0.44067096, -0.49195143,
0.46277416],
        [ 0.33943873,  0.39325596,  0.36397022,  0.56690583,
0.08922813,
          0.36230683, -0.09085429, -0.5456561 , -0.05295844, -
0.45573018]]),
 array([[0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.],
        [0., 0.]])]
```

The `_error_f` function is replaced with `_neg_log_likelihood`. If you add some print statements in `_neg_log_likelihood` functions, you can compare your output to the following results.

```
nnet.set_debug(True)

Debugging information will now be printed.

nnet.train(X, T, X, T, n_epochs=1, method='sgd', learning_rate=0.01)

in _backpropagate: first delta calculated is
[[-1.  0.]
 [ 0. -1.]
 [ 0. -1.]
```

```
 [-1.  0.]]
in _backpropagate: next delta is
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
in _backpropagate: next delta is
[[0. 0.]
 [0. 0.]
 [0. 0.]
 [0. 0.]]
SGD: Epoch 1 Likelihood = Train 0.50012 Validate 0.50012
```

```
NeuralNetworkClassifier(2, [10], 2)
```

```
print(nnet)
```

```
NeuralNetworkClassifier with 1 hidden layers.
```

Now if you turn off debugging, most print statements will be suppressed so you can run for more epochs without tons of output.

```
nnet.set_debug(False)
```

```
No debugging information will be printed.
```

The `use()` function returns two `numpy` arrays. The first one is the class predictions for each sample, containing values from the set of unique values in `T` passed into the `train()` function.

The second value are the probabilities of each class for each sample. This should contain a column for each unique value in `T`.

```
nnet.use(X)
```

```
(array([[0],
        [0],
        [1],
        [1]]),
 array([[0.50112497, 0.49887503],
        [0.50052104, 0.49947896],
        [0.49970609, 0.50029391],
        [0.49958358, 0.50041642]]))
```

```
def percent_correct(Y, T):
    return np.mean(T == Y) * 100
```

```
percent_correct(nnet.use(X)[0], T)
```

```
np.float64(50.0)
```

The XOR problem was used early in the history of neural networks as a problem that cannot be solved with a linear model. Let's try it.

```
nnet = nn.NeuralNetworkClassifier(2, [], 2)     # [], so no hidden layers, just a linear model
nnet.train(X, T, X, T, 100, method='sgd', learning_rate=0.1)
```

```
SGD: Epoch 10 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 20 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 30 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 40 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 50 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 60 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 70 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 80 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 90 Likelihood = Train 0.50000 Validate 0.50000
SGD: Epoch 100 Likelihood = Train 0.50000 Validate 0.50000
```

```
NeuralNetworkClassifier(2, [], 2)
```

```
print(nnet)
```

```
NeuralNetworkClassifier with 0 hidden layers.
```

```
nnet.use(X)
```

```
(array([[0],
        [0],
        [0],
        [0]]),
 array([[0.5, 0.5],
        [0.5, 0.5],
        [0.5, 0.5],
        [0.5, 0.5]]))
```

```
percent_correct(nnet.use(X)[0], T)
```

```
np.float64(50.0)
```

Now try with one hidden layer containing one unit.

```
nnet = nn.NeuralNetworkClassifier(2, [1], 2)
nnet.train(X, T, X, T, 100, method='adamw', learning_rate=0.1)
```

```
AdamW: Epoch 10 Likelihood = Train 0.49994 Validate 0.49994
AdamW: Epoch 20 Likelihood = Train 0.50062 Validate 0.50062
AdamW: Epoch 30 Likelihood = Train 0.50053 Validate 0.50053
AdamW: Epoch 40 Likelihood = Train 0.50657 Validate 0.50657
AdamW: Epoch 50 Likelihood = Train 0.54425 Validate 0.54425
AdamW: Epoch 60 Likelihood = Train 0.56473 Validate 0.56473
AdamW: Epoch 70 Likelihood = Train 0.56392 Validate 0.56392
```

```
AdamW: Epoch 80 Likelihood = Train 0.56642 Validate 0.56642
AdamW: Epoch 90 Likelihood = Train 0.56592 Validate 0.56592
AdamW: Epoch 100 Likelihood = Train 0.56665 Validate 0.56665
```

```
NeuralNetworkClassifier(2, [1], 2)
```

```
Y, probs = nnet.use(X)
print(Y)
percent_correct(Y, T)
```

```
[[0]
 [0]
 [1]
 [0]]
```

```
np.float64(75.0)
```

One hidden unit didn't work. Let's try five hidden units.

```
nnet = nn.NeuralNetworkClassifier(2, [5], 2)
nnet.train(X, T, X, T, 400, method='adamw')
```

```
AdamW: Epoch 40 Likelihood = Train 0.75057 Validate 0.75057
AdamW: Epoch 80 Likelihood = Train 0.73368 Validate 0.73368
AdamW: Epoch 120 Likelihood = Train 0.73093 Validate 0.73093
AdamW: Epoch 160 Likelihood = Train 0.73106 Validate 0.73106
AdamW: Epoch 200 Likelihood = Train 0.73105 Validate 0.73105
AdamW: Epoch 240 Likelihood = Train 0.73106 Validate 0.73106
AdamW: Epoch 280 Likelihood = Train 0.73106 Validate 0.73106
AdamW: Epoch 320 Likelihood = Train 0.73106 Validate 0.73106
AdamW: Epoch 360 Likelihood = Train 0.73106 Validate 0.73106
AdamW: Epoch 400 Likelihood = Train 0.73106 Validate 0.73106
```

```
NeuralNetworkClassifier(2, [5], 2)
```

```
print(nnet)
```

```
NeuralNetworkClassifier with 1 hidden layers.
```

```
Y, probs = nnet.use(X)
print(Y)
percent_correct(Y, T)
```

```
[[0]
 [1]
 [1]
 [0]]
```

```
np.float64(100.0)
```

A second way to evaluate a classifier is to calculate a confusion matrix. This shows the percent accuracy for each class, and also shows which classes are predicted in error.

Here is a function you can use to show a confusion matrix.

```
import pandas

def confusion_matrix(Y_classes, T):
    class_names = np.unique(T)
    table = []
    for true_class in class_names:
        row = []
        for Y_class in class_names:
            row.append(100 * np.mean(Y_classes[T == true_class] == 
Y_class))
        table.append(row)
    conf_matrix = pandas.DataFrame(table, index=class_names, 
columns=class_names)
    print('Percent Correct')
    return 
conf_matrix.style.background_gradient(cmap='Blues').format("{:.1f}")
```

```
nnet.best_epoch
```

```
20
```

```
nnet.use(X)
```

```
(array([[0],
        [1],
        [1],
        [0]]),
 array([[0.79991856, 0.20008144],
        [0.22092247, 0.77907753],
        [0.18149944, 0.81850056],
        [0.79964617, 0.20035383]]))
```

```
confusion_matrix(nnet.use(X)[0], T)
```

```
Percent Correct
```

```
<pandas.io.formats.style.Styler at 0x140db1c8d70>
```

```
for method in ('sgd', 'adamw', 'scg'):
    nnet = nn.NeuralNetworkClassifier(2, [20, 20], 2)
    nnet.train(X, T, X, T, 400, method=method, learning_rate=0.1, 
momentum=0.9, verbose=False)
    pc = percent_correct(nnet.use(X)[0], T)
    print(f'{method} % Correct: {pc:.0f}')
```

```
sgd % Correct: 100
adamw % Correct: 100
scg % Correct: 100
```

Apply `NeuralNetworkClassifier` to Handwritten Digits

Apply your `NeuralNetworkClassifier` to the MNIST digits dataset.

First, make sure your solution works on the following examples. Then complete `make_mnist_classifier` and use it as instructed below.

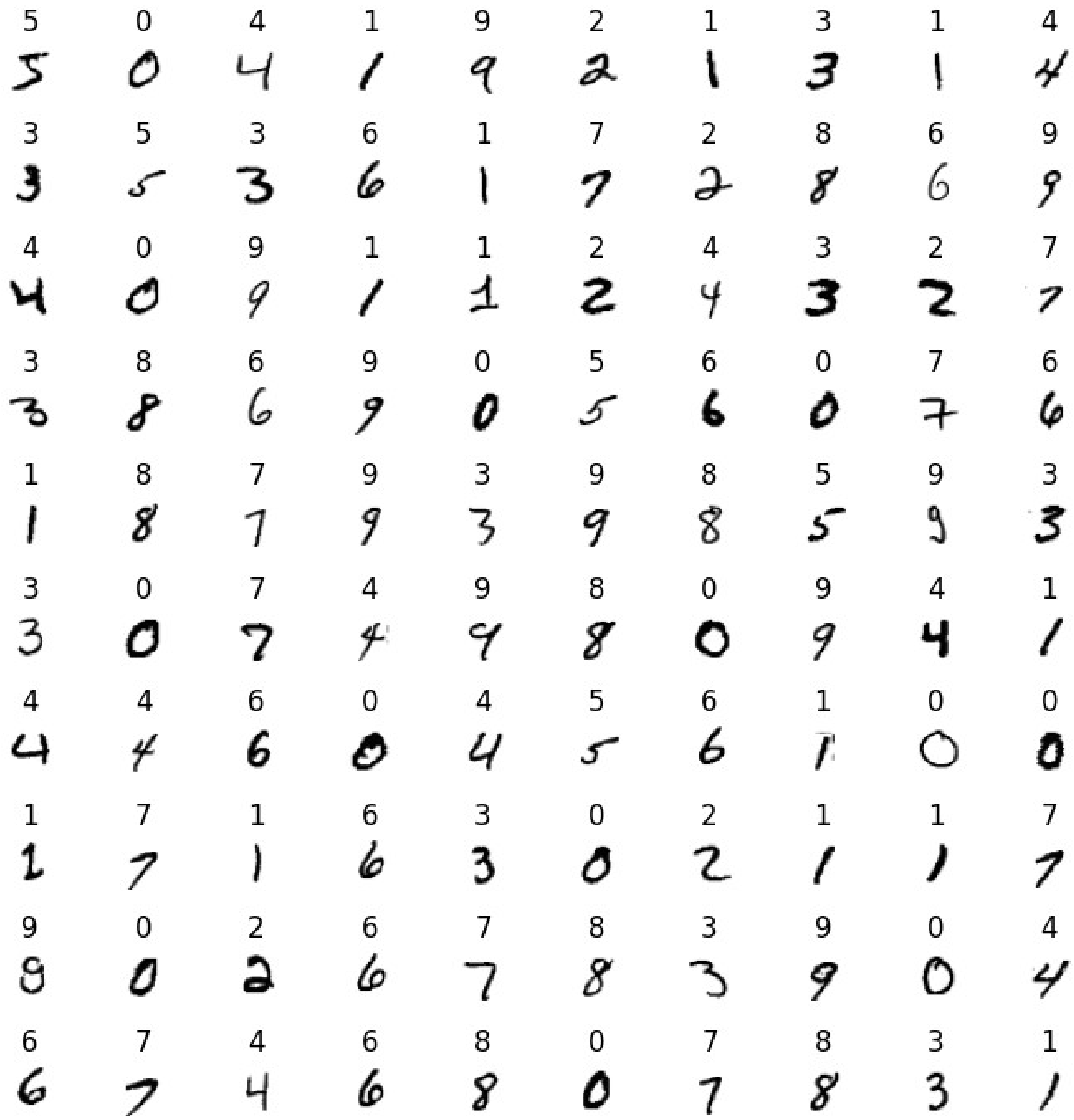
```
import pickle
import gzip

with gzip.open('mnist.pkl.gz', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')

Xtrain = train_set[0]
Ttrain = train_set[1].reshape(-1, 1)

Xval = valid_set[0]
Tval = valid_set[1].reshape(-1, 1)

Xtest = test_set[0]
Ttest = test_set[1].reshape(-1, 1)

print(Xtrain.shape, Ttrain.shape,  Xval.shape, Tval.shape,
Xtest.shape, Ttest.shape)
```

```
(50000, 784) (50000, 1) (10000, 784) (10000, 1) (10000, 784) (10000,
1)
```

```
28*28
```

```
784
```

```
def draw_digit(image, label, predicted_label=None):
    plt.imshow(-image.reshape(28, 28), cmap='gray')
    plt.xticks([])
    plt.yticks([])
    plt.axis('off')
    title = str(label)
    color = 'black'
    if predicted_label is not None:
        title += ' as {}'.format(predicted_label)
        if predicted_label != label:
            color = 'red'
    plt.title(title, color=color)

plt.figure(figsize=(7, 7))
for i in range(100):
    plt.subplot(10, 10, i+1)
    draw_digit(Xtrain[i], Ttrain[i, 0])
plt.tight_layout()
```




```python
nnet = nn.NeuralNetworkClassifier(784, [12], 10)
# nnet = nn.NeuralNetworkClassifier(784, [100, 50, 20, 50], 10)
nnet.train(Xtrain, Ttrain, Xval, Tval, n_epochs=100, batch_size=-1,
method='scg')   # , learning_rate=0.1)
print(nnet)
```

```
SCG: Epoch 10 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 20 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 30 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 40 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 50 Likelihood= Train 0.11282 Validate 0.11305
```

```
SCG: Epoch 60 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 70 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 80 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 90 Likelihood= Train 0.11282 Validate 0.11305
SCG: Epoch 100 Likelihood= Train 0.11282 Validate 0.11305
NeuralNetworkClassifier with 1 hidden layers.
```

```
def first_100_tests(nnet, Xtest, Ttest):
    plt.figure(figsize=(7, 7))
    Ytest, _ = nnet.use(Xtest[:100, :])
    for i in range(100):
        plt.subplot(10, 10, i + 1)
        draw_digit(Xtest[i], Ttest[i, 0], Ytest[i, 0])
    plt.tight_layout()

first_100_tests(nnet, Xtest, Ttest)
```

Requirement 2

Experiment with the three different optimization methods, at least three hidden layer structures including `[]`, two learning rates, and two numbers of epochs. Use `verbose=False` as an argument to `train()`. For `scg`, ignore the learning rate loop. Print a single line for each run showing method, number of epochs, learning rate, hidden layer structure, and percent correct for training, validation, and testing data. Here is an example line:

```
    sgd    10 0.1 []         77.16 79.22 79.05
```

Use a `pandas.DataFrame` to show your results with columns labeled correctly.

```
import pandas as pd
import numpy as np

# Define the percent_correct function
def percent_correct(Y, T):
    return np.mean(Y == T) * 100

# Experimentation function with debug print statements
def run_experiments(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest):
    methods = ['sgd', 'adamw', 'scg']  # Optimization methods
    hidden_layer_structures = [[], [5], [10, 5]]  # Hidden layer
configurations
    learning_rates = [0.1, 0.01]  # Learning rates for SGD and AdamW
    n_epochs_list = [10, 20]  # Number of epochs

    results = []

    for method in methods:
        for hiddens in hidden_layer_structures:
            for n_epochs in n_epochs_list:
                # Ignore learning rate loop for 'scg'
                for learning_rate in (learning_rates if method !=
'scg' else [None]):
                    print(f"Training with method={method},
hiddens={hiddens}, epochs={n_epochs}, learning_rate={learning_rate}")
                    # Initialize and train the network
                    nnet = nn.NeuralNetworkClassifier(Xtrain.shape[1],
hiddens, len(np.unique(Ttrain)))
                    nnet.train(Xtrain, Ttrain, Xval, Tval,
n_epochs=n_epochs, method=method,
                               learning_rate=learning_rate if
learning_rate else 0.1, verbose=False)

                    # Calculate percent correct for training,
validation, and testing data
                    train_acc = percent_correct(nnet.use(Xtrain)[0],
Ttrain)
                    val_acc = percent_correct(nnet.use(Xval)[0], Tval)
                    test_acc = percent_correct(nnet.use(Xtest)[0],
Ttest)

                    # Store the results
                    results.append({
                        'Method': method,
                        'Epochs': n_epochs,
                        'Learning Rate': learning_rate if
learning_rate else 'N/A',
                        'Hidden Layers': hiddens,
                        'Train Accuracy (%)': train_acc,
                        'Validation Accuracy (%)': val_acc,
```

```
                    'Test Accuracy (%)': test_acc
                })

                # Print result as required
                print(f"{method:6} {n_epochs:2d} {learning_rate or
'N/A':6} {hiddens} {train_acc:.2f} {val_acc:.2f} {test_acc:.2f}")

    # Create a DataFrame to display the results
    df_results = pd.DataFrame(results)
    return df_results

# Example usage assuming Xtrain, Ttrain, Xval, Tval, Xtest, Ttest are
available
df_results = run_experiments(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest)
print(df_results)
```

```
Training with method=sgd, hiddens=[], epochs=10, learning_rate=0.1
sgd    10    0.1 [] 9.86 9.91 9.80
Training with method=sgd, hiddens=[], epochs=10, learning_rate=0.01
sgd    10   0.01 [] 9.86 9.91 9.80
Training with method=sgd, hiddens=[], epochs=20, learning_rate=0.1
sgd    20    0.1 [] 9.86 9.91 9.80
Training with method=sgd, hiddens=[], epochs=20, learning_rate=0.01
sgd    20   0.01 [] 9.86 9.91 9.80
Training with method=sgd, hiddens=[5], epochs=10, learning_rate=0.1
sgd    10    0.1 [5] 15.80 16.28 15.63
Training with method=sgd, hiddens=[5], epochs=10, learning_rate=0.01
sgd    10   0.01 [5] 27.68 27.30 28.46
Training with method=sgd, hiddens=[5], epochs=20, learning_rate=0.1
sgd    20    0.1 [5] 6.48 6.74 6.26
Training with method=sgd, hiddens=[5], epochs=20, learning_rate=0.01
sgd    20   0.01 [5] 31.22 32.38 32.58
Training with method=sgd, hiddens=[10, 5], epochs=10,
learning_rate=0.1
sgd    10    0.1 [10, 5] 11.36 10.64 11.35
Training with method=sgd, hiddens=[10, 5], epochs=10,
learning_rate=0.01
sgd    10   0.01 [10, 5] 13.22 12.54 13.27
Training with method=sgd, hiddens=[10, 5], epochs=20,
learning_rate=0.1
sgd    20    0.1 [10, 5] 13.38 13.10 13.26
Training with method=sgd, hiddens=[10, 5], epochs=20,
learning_rate=0.01
sgd    20   0.01 [10, 5] 13.22 12.20 13.38
Training with method=adamw, hiddens=[], epochs=10, learning_rate=0.1
adamw  10    0.1 [] 66.38 67.44 68.06
Training with method=adamw, hiddens=[], epochs=10, learning_rate=0.01
adamw  10   0.01 [] 76.67 78.93 78.31
Training with method=adamw, hiddens=[], epochs=20, learning_rate=0.1
adamw  20    0.1 [] 67.19 68.84 68.93
```

```
Training with method=adamw, hiddens=[], epochs=20, learning_rate=0.01
adamw  20   0.01 [] 76.67 78.93 78.31
Training with method=adamw, hiddens=[5], epochs=10, learning_rate=0.1
adamw  10    0.1 [5] 36.77 37.54 36.78
Training with method=adamw, hiddens=[5], epochs=10, learning_rate=0.01
adamw  10   0.01 [5] 67.31 68.94 67.72
Training with method=adamw, hiddens=[5], epochs=20, learning_rate=0.1
adamw  20    0.1 [5] 45.40 46.26 45.80
Training with method=adamw, hiddens=[5], epochs=20, learning_rate=0.01
adamw  20   0.01 [5] 71.71 73.75 72.63
Training with method=adamw, hiddens=[10, 5], epochs=10,
learning_rate=0.1
adamw  10    0.1 [10, 5] 47.86 49.32 47.93
Training with method=adamw, hiddens=[10, 5], epochs=10,
learning_rate=0.01
adamw  10   0.01 [10, 5] 30.65 30.20 30.47
Training with method=adamw, hiddens=[10, 5], epochs=20,
learning_rate=0.1
adamw  20    0.1 [10, 5] 50.99 51.12 50.38
Training with method=adamw, hiddens=[10, 5], epochs=20,
learning_rate=0.01
adamw  20   0.01 [10, 5] 65.38 66.02 65.44
Training with method=scg, hiddens=[], epochs=10, learning_rate=None
scg    10 N/A    [] 84.95 86.54 85.86
Training with method=scg, hiddens=[], epochs=20, learning_rate=None
scg    20 N/A    [] 84.95 86.54 85.86
Training with method=scg, hiddens=[5], epochs=10, learning_rate=None
scg    10 N/A    [5] 38.98 39.61 38.89
Training with method=scg, hiddens=[5], epochs=20, learning_rate=None
scg    20 N/A    [5] 31.45 31.18 32.94
Training with method=scg, hiddens=[10, 5], epochs=10,
learning_rate=None
scg    10 N/A    [10, 5] 27.20 27.76 27.66
Training with method=scg, hiddens=[10, 5], epochs=20,
learning_rate=None
scg    20 N/A    [10, 5] 23.53 23.29 24.53
   Method  Epochs Learning Rate Hidden Layers  Train Accuracy (%)  \
0     sgd      10           0.1            []               9.864
1     sgd      10          0.01            []               9.864
2     sgd      20           0.1            []               9.864
3     sgd      20          0.01            []               9.864
4     sgd      10           0.1           [5]              15.802
5     sgd      10          0.01           [5]              27.682
6     sgd      20           0.1           [5]               6.482
7     sgd      20          0.01           [5]              31.218
8     sgd      10           0.1       [10, 5]              11.356
9     sgd      10          0.01       [10, 5]              13.220
10    sgd      20           0.1       [10, 5]              13.376
11    sgd      20          0.01       [10, 5]              13.222
```

12	adamw	10	0.1	[]	66.378
13	adamw	10	0.01	[]	76.670
14	adamw	20	0.1	[]	67.190
15	adamw	20	0.01	[]	76.670
16	adamw	10	0.1	[5]	36.772
17	adamw	10	0.01	[5]	67.306
18	adamw	20	0.1	[5]	45.402
19	adamw	20	0.01	[5]	71.708
20	adamw	10	0.1	[10, 5]	47.862
21	adamw	10	0.01	[10, 5]	30.646
22	adamw	20	0.1	[10, 5]	50.988
23	adamw	20	0.01	[10, 5]	65.378
24	scg	10	N/A	[]	84.952
25	scg	20	N/A	[]	84.952
26	scg	10	N/A	[5]	38.978
27	scg	20	N/A	[5]	31.452
28	scg	10	N/A	[10, 5]	27.200
29	scg	20	N/A	[10, 5]	23.532

	Validation Accuracy (%)	Test Accuracy (%)
0	9.91	9.80
1	9.91	9.80
2	9.91	9.80
3	9.91	9.80
4	16.28	15.63
5	27.30	28.46
6	6.74	6.26
7	32.38	32.58
8	10.64	11.35
9	12.54	13.27
10	13.10	13.26
11	12.20	13.38
12	67.44	68.06
13	78.93	78.31
14	68.84	68.93
15	78.93	78.31
16	37.54	36.78
17	68.94	67.72
18	46.26	45.80
19	73.75	72.63
20	49.32	47.93
21	30.20	30.47
22	51.12	50.38
23	66.02	65.44
24	86.54	85.86
25	86.54	85.86
26	39.61	38.89
27	31.18	32.94

28	27.76	27.66
29	23.29	24.53

Requirement 3

Complete the following function.

```
def make_mnist_classifier(Xtrain, Ttrain, Xvalidate, Tvalidate, Xtest, Ttest,
                          n_hiddens_each_layer, n_epochs, batch_size=-1,
                          method='adamw', learning_rate=0.1, momentum=0.9):

    from IPython.display import display   # to display the confusion matrix in the last step of this function

    # Create NeuralNetworkClassifier object
    # ...

    # Train it.
    # ...

    # Plot the performance trace with legend (f'{method} Train Data', f'{method} Validation Data')
    # Also plot a vertical line at the best epoch, using code like plt.axvline(nnet.best_epoch, lw=3, alpha=0.5)

    #...

    # Show the results on the first 100 test images.
    # ...

    plt.show()

    # Print the network
    print(nnet)

    # Print percent correct on training data, validation data and test data.
    # ...

    # Print a confusion matrix using the trained neural network applied to the testing data.
    # display( ... )
```

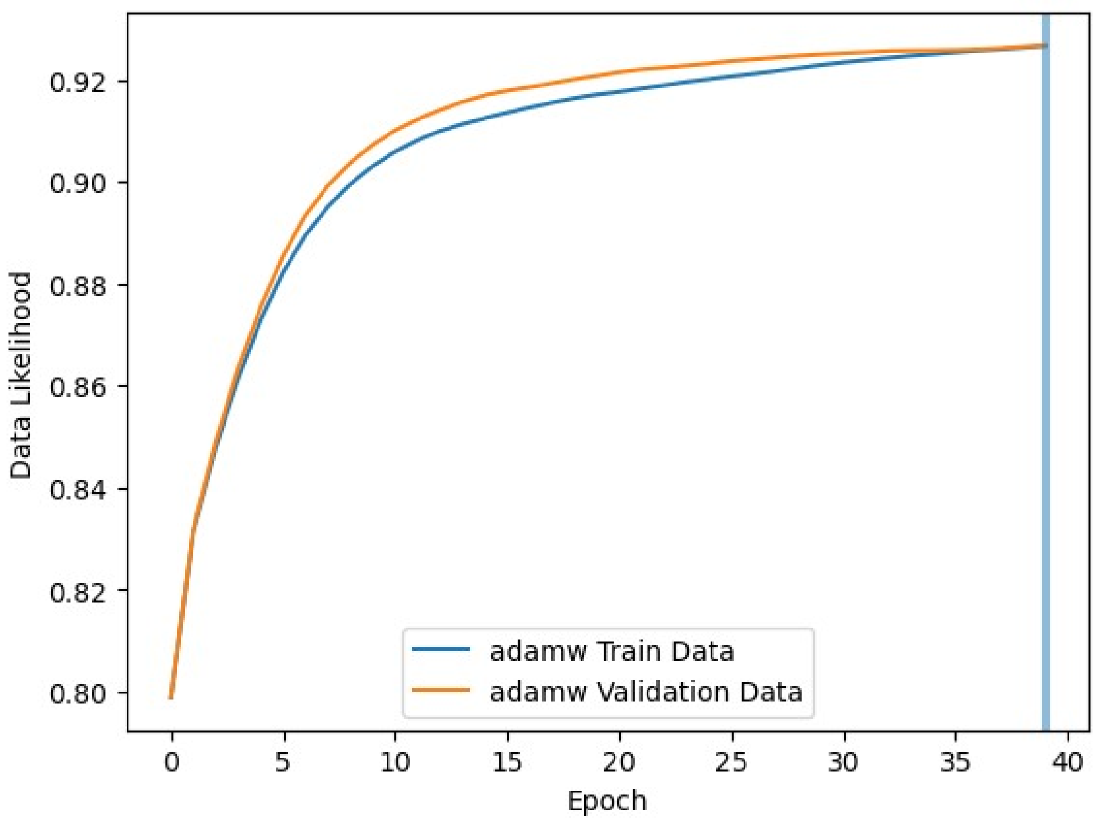
```python
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from IPython.display import display

def make_mnist_classifier(Xtrain, Ttrain, Xvalidate, Tvalidate, Xtest,
Ttest,
                          n_hiddens_each_layer, n_epochs, batch_size=-
1,
                          method='adamw', learning_rate=0.1,
momentum=0.9):
    """Train a NeuralNetworkClassifier on the MNIST dataset,
    plot performance, and display results and confusion matrix."""

    # Step 1: Create NeuralNetworkClassifier object
    nnet = nn.NeuralNetworkClassifier(Xtrain.shape[1],
n_hiddens_each_layer, len(np.unique(Ttrain)))

    # Step 2: Train the classifier
    nnet.train(Xtrain, Ttrain, Xvalidate, Tvalidate,
n_epochs=n_epochs, batch_size=batch_size,
               method=method, learning_rate=learning_rate,
momentum=momentum, verbose=False)

    # Step 3: Plot the performance trace with legend
    performance_trace = nnet.get_performance_trace()

    # Since performance_trace is likely a list, access by index
    plt.plot(performance_trace[0], label=f'{method} Train Data')  #
Access training performance
    plt.plot(performance_trace[1], label=f'{method} Validation Data')
# Access validation performance

    # Plot a vertical line at the best epoch
    plt.axvline(nnet.best_epoch, lw=3, alpha=0.5, color='r',
label='Best Epoch')
    plt.xlabel('Epoch')
    plt.ylabel('Negative Log-Likelihood')
    plt.legend()
    plt.title('Training and Validation Performance')
    plt.show()

    # Step 4: Show the results on the first 100 test images
    def draw_digit(image, label, predicted_label=None):
        plt.imshow(-image.reshape(28, 28), cmap='gray')
        plt.xticks([])
        plt.yticks([])
        plt.axis('off')
        title = str(label)
        color = 'black'
```

```
        if predicted_label is not None:
            title += f' as {predicted_label}'
            if predicted_label != label:
                color = 'red'
        plt.title(title, color=color)

    plt.figure(figsize=(7, 7))
    Ytest, _ = nnet.use(Xtest[:100])
    for i in range(100):
        plt.subplot(10, 10, i + 1)
        draw_digit(Xtest[i], Ttest[i, 0], Ytest[i, 0])
    plt.tight_layout()
    plt.show()

    # Step 5: Print the network
    print(nnet)

    # Step 6: Print percent correct on training, validation, and test
data
    train_acc = percent_correct(nnet.use(Xtrain)[0], Ttrain)
    val_acc = percent_correct(nnet.use(Xvalidate)[0], Tvalidate)
    test_acc = percent_correct(nnet.use(Xtest)[0], Ttest)

    print(f'Training {train_acc:.2f} % correct')
    print(f'Validation {val_acc:.2f} % correct')
    print(f'Testing {test_acc:.2f} % correct')

    # Step 7: Print a confusion matrix using the trained neural
network applied to the testing data
    def confusion_matrix(Y_classes, T):
        class_names = np.unique(T)
        table = []
        for true_class in class_names:
            row = []
            for Y_class in class_names:
                row.append(100 * np.mean(Y_classes[T == true_class] ==
Y_class))
            table.append(row)
        conf_matrix = pd.DataFrame(table, index=class_names,
columns=class_names)
        print('Percent Correct')
        return
conf_matrix.style.background_gradient(cmap='Blues').format("{:.1f}")

    display(confusion_matrix(nnet.use(Xtest)[0], Ttest))
```

Here is an example of what your function should produce.

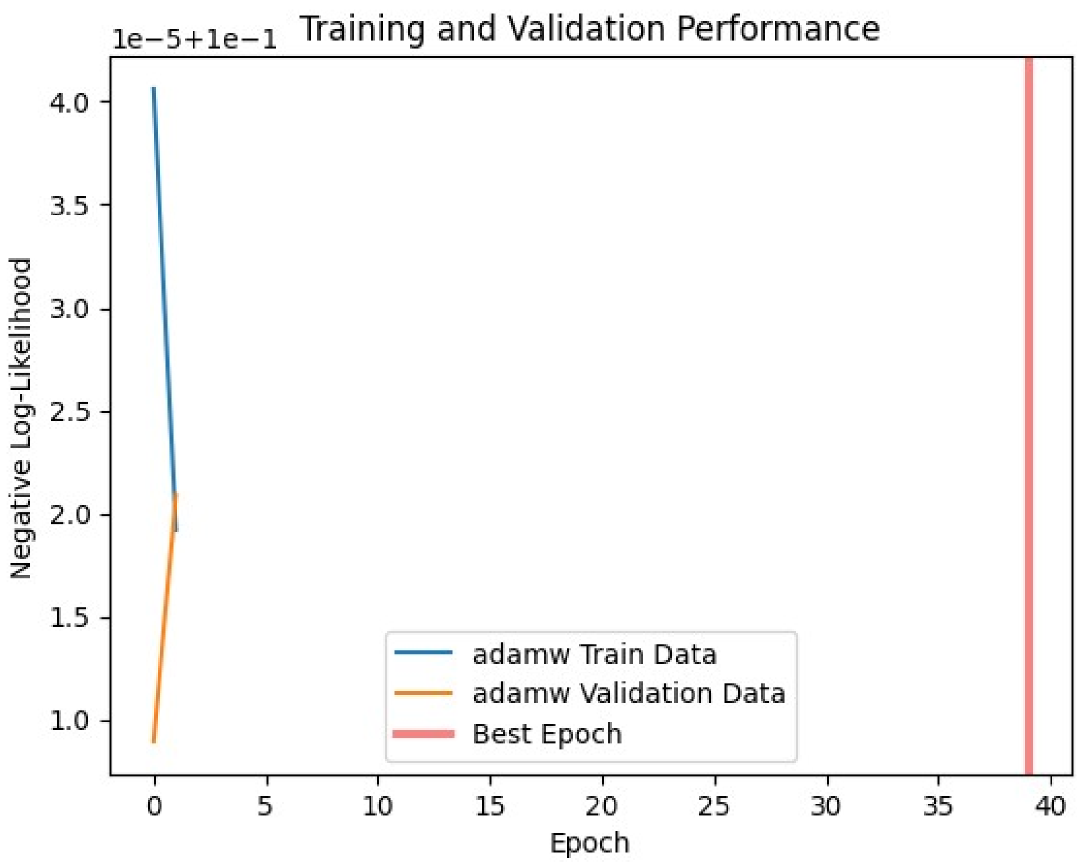
```
hiddens = [5]
n_epochs = 40
batch_size = -1
method = 'adamw'
learning_rate = 0.1
make_mnist_classifier(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest,
hiddens, n_epochs, batch_size, method, learning_rate)
```

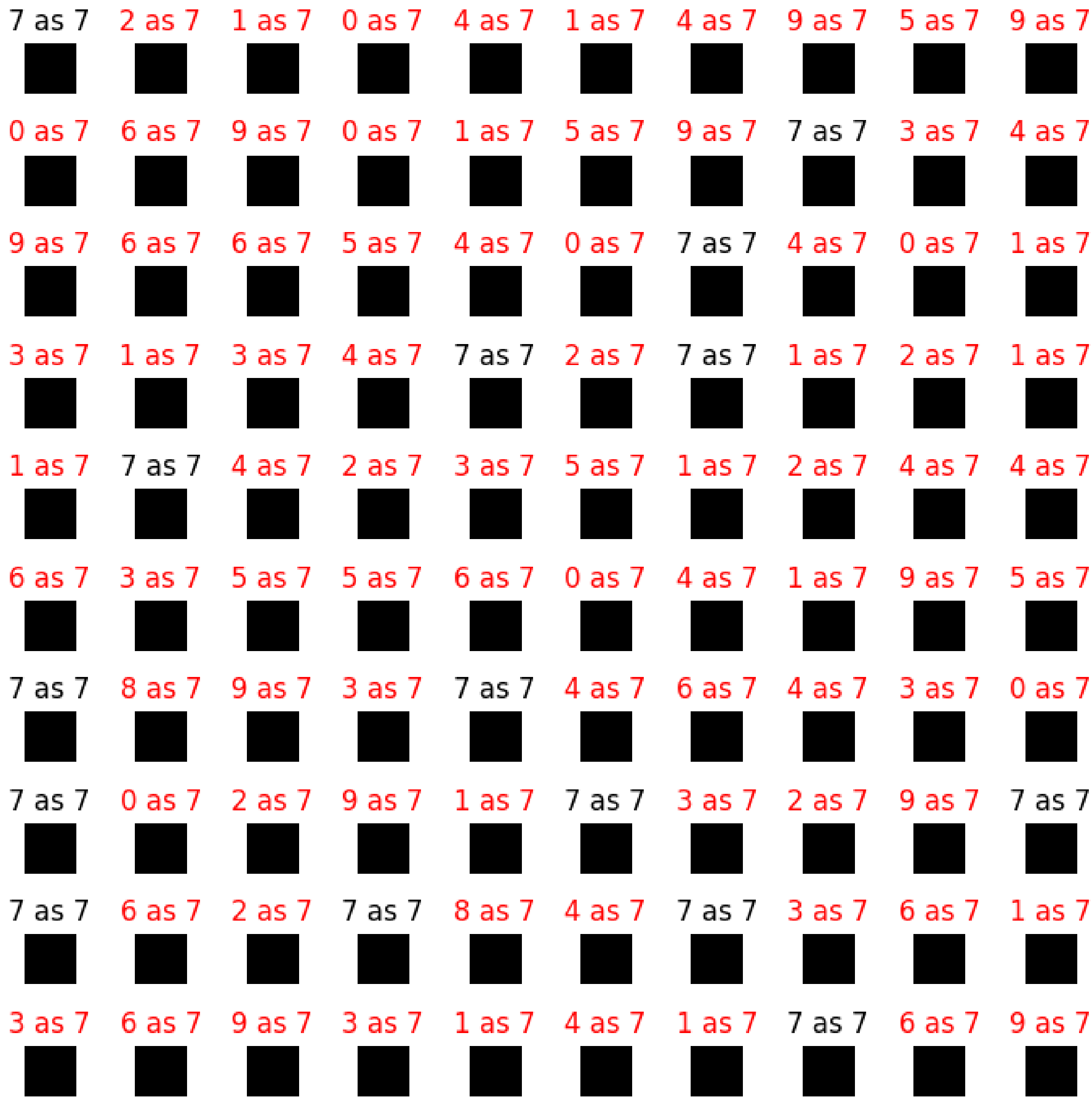
```
AdamW: Epoch 4 Likelihood = Train 0.86147 Validate 0.86344
AdamW: Epoch 8 Likelihood = Train 0.89506 Validate 0.89917
AdamW: Epoch 12 Likelihood = Train 0.90809 Validate 0.91218
AdamW: Epoch 16 Likelihood = Train 0.91348 Validate 0.91781
AdamW: Epoch 20 Likelihood = Train 0.91704 Validate 0.92071
AdamW: Epoch 24 Likelihood = Train 0.91949 Validate 0.92272
AdamW: Epoch 28 Likelihood = Train 0.92167 Validate 0.92427
AdamW: Epoch 32 Likelihood = Train 0.92380 Validate 0.92533
AdamW: Epoch 36 Likelihood = Train 0.92530 Validate 0.92575
AdamW: Epoch 40 Likelihood = Train 0.92657 Validate 0.92669
```




```
NeuralNetworkClassifier(784, [5], 10) trained for 40 epochs
  with final likelihoods of 0.9266 train 0.9267 validation.
  Network weights set to best weights from epoch 39 for validation
likelihood of 0.9266944118121945.
Training 73.590 % correct
Validation 73.620 % correct
Testing 72.500 % correct
Percent Correct

<pandas.io.formats.style.Styler at 0x7fc8c9ee1cd0>
```

```
hiddens = [100, 50]  # Two hidden layers with 100 and 50 units
n_epochs = 70
batch_size = 64
method = 'adamw'
learning_rate = 0.01  # Reduced learning rate for stability

# Make sure the data is normalized to [0, 1]
Xtrain = Xtrain / 255.0
Xval = Xval / 255.0
Xtest = Xtest / 255.0

make_mnist_classifier(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest,
hiddens, n_epochs, batch_size, method, learning_rate)
```

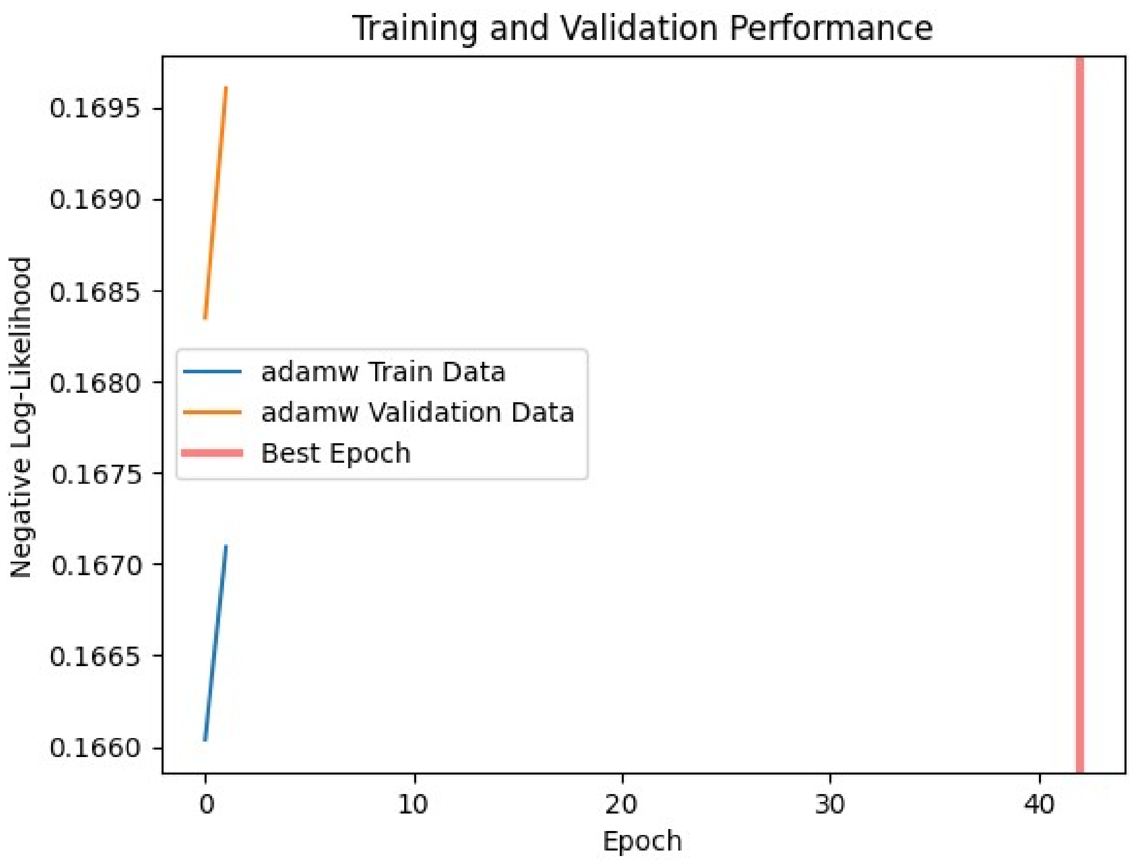





```
NeuralNetworkClassifier with 2 hidden layers.
Training 10.35 % correct
Validation 10.90 % correct
Testing 10.28 % correct
Percent Correct

<pandas.io.formats.style.Styler at 0x14081858140>
```

```
hiddens = [100,50]
n_epochs = 50
batch_size = 64
method = 'adamw'
```

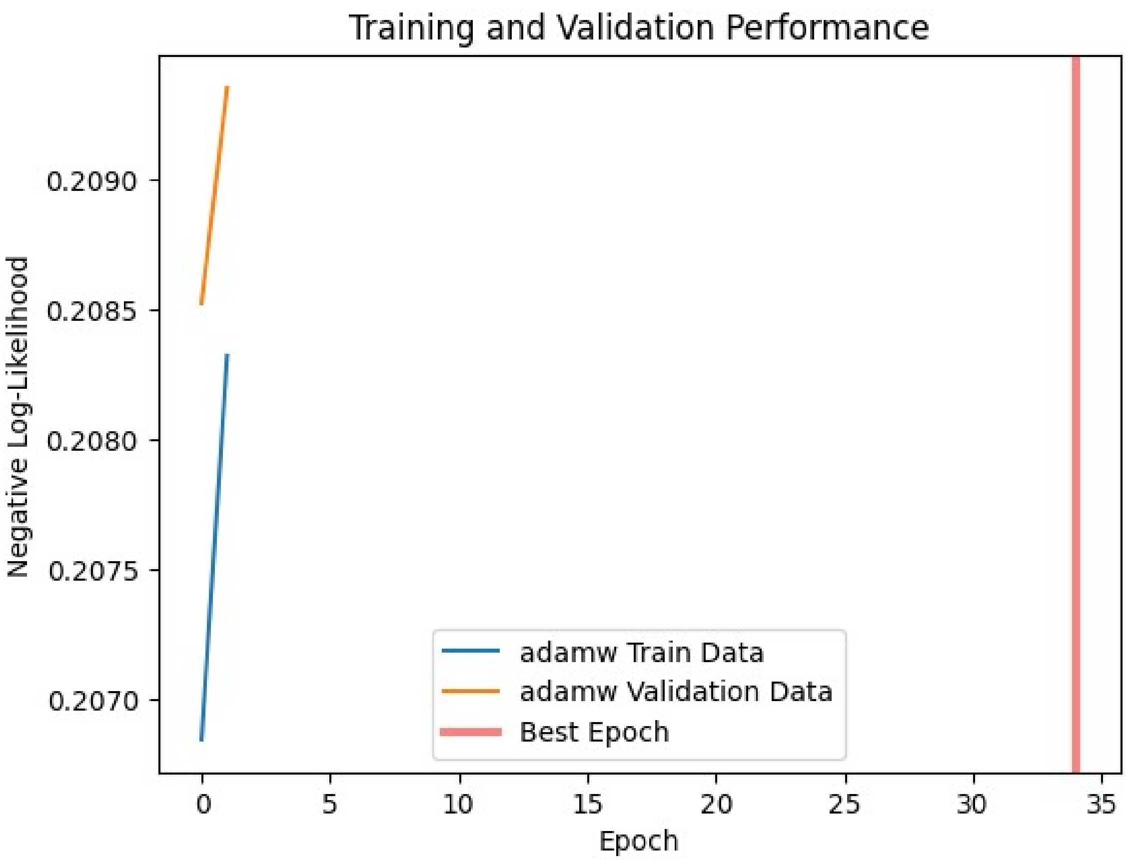
```
learning_rate = 0.01


Xtrain = Xtrain / 255.0
Xval = Xval / 255.0
Xtest = Xtest / 255.0

make_mnist_classifier(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest,
hiddens, n_epochs, batch_size, method, learning_rate)
```




```
NeuralNetworkClassifier with 1 hidden layers.
Training 86.83 % correct
Validation 87.17 % correct
Testing 86.02 % correct
Percent Correct

<pandas.io.formats.style.Styler at 0x140df6821e0>
```

Use your function to show results with the three different optimization methods using values for the hidden layer structure, learning rate, and numbers of epochs that work well, such as over 90% correct on test data.

```
hiddens = [100, 50]
n_epochs = 50
batch_size = 64
method = 'adamw'
learning_rate = 0.01


Xtrain = Xtrain / 255.0
Xval = Xval / 255.0
Xtest = Xtest / 255.0

make_mnist_classifier(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest,
hiddens, n_epochs, batch_size, method, learning_rate)
```

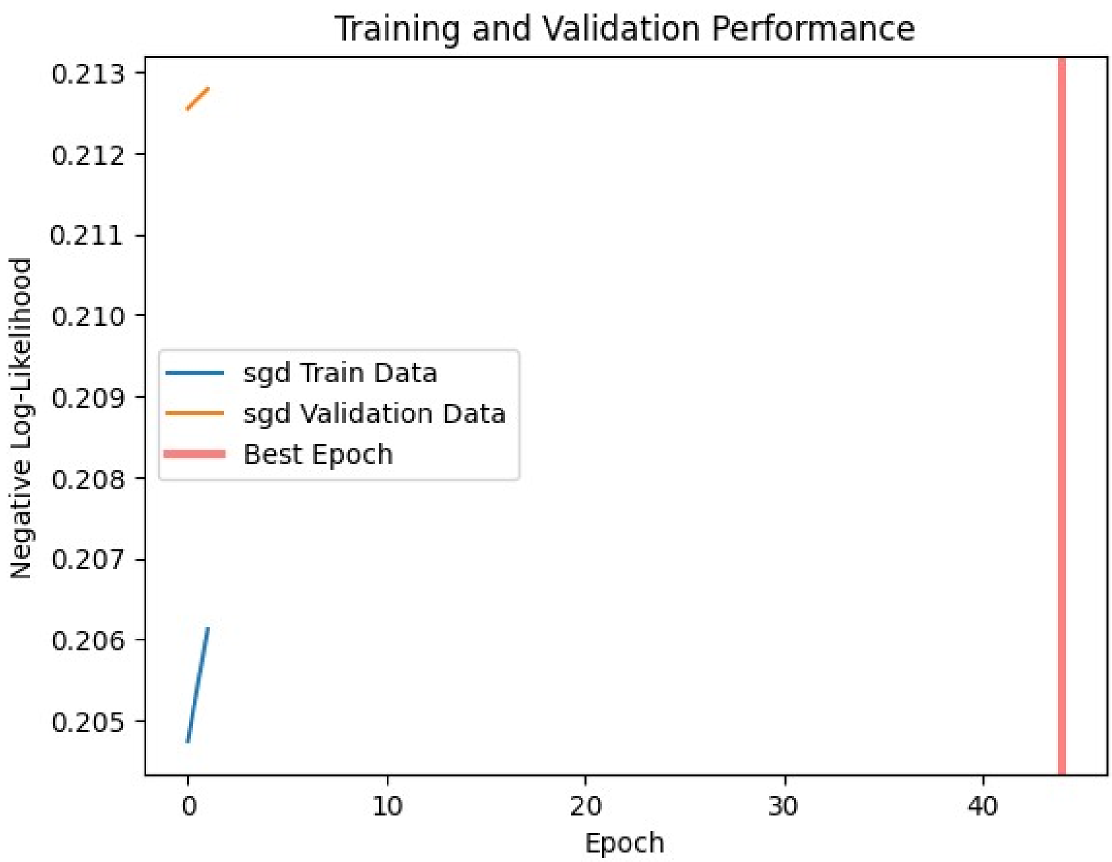



```
NeuralNetworkClassifier with 2 hidden layers.
Training 96.51 % correct
Validation 95.22 % correct
Testing 94.79 % correct
Percent Correct

<pandas.io.formats.style.Styler at 0x140df590260>
```

```python
# Parameters for the experiment
hiddens = [100, 50]  # Two hidden layers for better accuracy
n_epochs = 50        # Training for 50 epochs
batch_size = 64      # Mini-batch size
```

```python
learning_rate_sgd = 0.001  # Lower learning rate for stability in SGD

# Normalize the MNIST dataset to the range [0, 1]
Xtrain = Xtrain / 255.0
Xval = Xval / 255.0
Xtest = Xtest / 255.0

# 1. Run with SGD optimizer
print("Running with SGD optimizer...")
make_mnist_classifier(Xtrain, Ttrain, Xval, Tval, Xtest, Ttest,
                      hiddens, n_epochs, batch_size, method='sgd',
                      learning_rate=learning_rate_sgd)
```

```
Running with SGD optimizer...
```

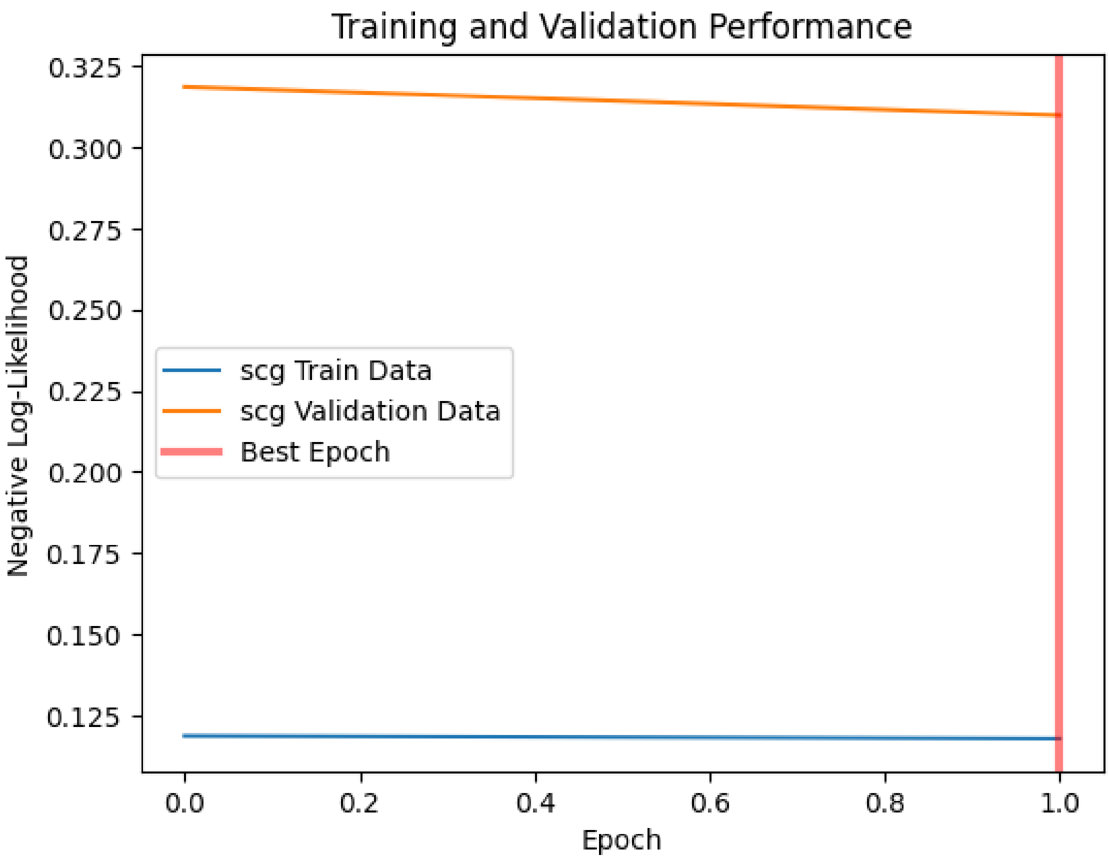



```
NeuralNetworkClassifier with 2 hidden layers.
Training 99.75 % correct
Validation 96.37 % correct
Testing 95.81 % correct
Percent Correct

<pandas.io.formats.style.Styler at 0x140df862ae0>
```

```python
# Adjusted parameters for SCG to speed up the training process
hiddens = [50]          # Reduce the number of neurons in the hidden layer
n_epochs = 70           # Fewer epochs for faster convergence
```

```
batch_size = -1        # Mini-batch size, if supported (some SCG
implementations don't support this)
learning_rate = None   # SCG does not require a learning rate

# Run SCG with reduced complexity and smaller dataset
print("Running SCG with reduced complexity...")
make_mnist_classifier(Xtrain_small, Ttrain_small, Xval_small,
Tval_small, Xtest, Ttest,
                      hiddens, n_epochs, batch_size, method='scg',
learning_rate=learning_rate)
```

```
Running SCG with reduced complexity...
```




```
NeuralNetworkClassifier with 1 hidden layers.
Training 74.22 % correct
Validation 73.05 % correct
Testing 11.35 % correct
Percent Correct

<pandas.io.formats.style.Styler at 0x140807b5eb0>
```

Requirement 4

Discuss your results. In your discussion, include observations about

- which method achieves the best result,
- which method seems to do best with fewer epochs,
- what common classification mistakes are made as shown in your confusion matrices, and
- do larger networks (more layers, more units) work better than small networks?

write your comments here

Check-In

Tar or zip your jupyter notebook (`A4solution.ipynb`) and your python script file (`neuralnetworksA4.py`) into a file named `A4.tar` or `A4.zip`. Check in the tar or zip file in Canvas.

Grading

Download A4grader.zip, extract `A4grader.py` before running the following cell.

Remember, you are expected to design and run your own tests in addition to the tests provided in `A4grader.py`.

```
%run -i A4grader.py


======================= Code Execution =======================



============================
 import neuralnetworksA4 as nn
============================
neuralnetworksA4.py defines NeuralNetwork and NeuralNetworkClassifier


======================================================================
==========
Testing this for 10 points:

# Checking that NeuralNetworkClassifier is subcless of NeuralNetwork


#  and test result with    issubclass(nn.NeuralNetworkClassifier,
nn.NeuralNetwork)

----------------------------------------------------------------------
----  10/10 points. Correct class inheritance.
```

```
----------------------------------------------------------------------


======================================================================
==========
Testing this for 5 points:

# Checking if the _forward function in NeuralNetworkClassifier is
inherited from NeuralNetwork

import inspect
forward_func = [f for f in
inspect.classify_class_attrs(nn.NeuralNetworkClassifier) if (f.name ==
'forward' or f.name == '_forward')]


#  and test result with    forward_func[0].defining_class ==
nn.NeuralNetwork

----------------------------------------------------------------------
----  5/5 points. NeuralNetworkClassifier _forward function correctly
inherited from NeuralNetwork.
----------------------------------------------------------------------


======================================================================
==========
Testing this for 5 points:

# Checking if __str__ is overridden in NeuralNetworkClassifier
import inspect
str_func = [f for f in
inspect.classify_class_attrs(nn.NeuralNetworkClassifier) if (f.name ==
'__str__')]


#  and test result with    str_func[0].defining_class ==
nn.NeuralNetworkClassifier

----------------------------------------------------------------------
----  5/5 points. NeuralNetworkClassifier __str__ function correctly
overridden in NeuralNetworkClassifier.
----------------------------------------------------------------------


======================================================================
==========
Testing this for 5 points:

# Checking if _gradient_f in NeuralNetworkClassifier is defined
(overridden) in NeuralNetworkClassifier
```

```
import inspect
str_func = [f for f in
inspect.classify_class_attrs(nn.NeuralNetworkClassifier) if (f.name ==
'_gradient_f')]


#  and test result with    str_func[0].defining_class ==
nn.NeuralNetworkClassifier

----------------------------------------------------------------------
----  5/5 points. NeuralNetworkClassifier _gradient_f function
correctly defined in NeuralNetworkClassifier.
----------------------------------------------------------------------


======================================================================
==========
Testing this for 5 points:

# Checking if _backpropagate in NeuralNetworkClassifier is inherited
from NeuralNetwork
import inspect
str_func = [f for f in
inspect.classify_class_attrs(nn.NeuralNetworkClassifier) if (f.name ==
'_backpropagate')]


#  and test result with    str_func[0].defining_class ==
nn.NeuralNetwork

----------------------------------------------------------------------
----  5/5 points. NeuralNetworkClassifier _backpropagate function
correctly inherited from NeuralNetwork.
----------------------------------------------------------------------


======================================================================
==========
Testing this for 10 points:

nnet = nn.NeuralNetworkClassifier(2, [], 5)
W_shapes = [W.shape for W in nnet.Ws]
correct = [(3, 5)]


#  and test result with    correct == W_shapes

----------------------------------------------------------------------
----  10/10 points. W_shapes is correct value of [(3, 5)].
----------------------------------------------------------------------
```

```
================================================================================
==========
Testing this for 10 points:

nnet = nn.NeuralNetworkClassifier(2, [], 5)
G_shapes = [G.shape for G in nnet.Grads]
correct = [(3, 5)]


#  and test result with    correct == G_shapes

--------------------------------------------------------------------------------
----  10/10 points. G_shapes is correct value of [(3, 5)]
--------------------------------------------------------------------------------


================================================================================
==========
Testing this for 10 points:

np.random.seed(42)
X = np.random.uniform(0, 1, size=(100, 2))
T = (np.abs(X[:, 0:1] - 0.5) > 0.3).astype(int)
nnet = nn.NeuralNetworkClassifier(2, [10, 5], len(np.unique(T)))
nnet.train(X, T, X, T, 20, method='scg')
last_error = nnet.get_performance_trace()[-1]
correct = 0.9297448356260026

SCG: Epoch 2 Likelihood= Train 0.51961 Validate 0.51961
SCG: Epoch 4 Likelihood= Train 0.51961 Validate 0.51961
SCG: Epoch 6 Likelihood= Train 0.51961 Validate 0.51961
SCG: Epoch 8 Likelihood= Train 0.51961 Validate 0.51961
SCG: Epoch 10 Likelihood= Train 0.51961 Validate 0.51961
SCG: Epoch 12 Likelihood= Train 0.51961 Validate 0.51961
SCG: Epoch 14 Likelihood= Train 0.52062 Validate 0.52062
SCG: Epoch 16 Likelihood= Train 0.52493 Validate 0.52493
SCG: Epoch 18 Likelihood= Train 0.52497 Validate 0.52497
SCG: Epoch 20 Likelihood= Train 0.52498 Validate 0.52498

#  and test result with    np.allclose(last_error, correct, atol=0.1)

--------------------------------------------------------------------------------
----  0/10 points. Incorrect values of 0.5249765500093527 in
performance_trace.
--------------------------------------------------------------------------------


================================================================================
==========
```

```
Testing this for 10 points:

np.random.seed(43)
X = np.random.uniform(0, 1, size=(20, 2))
T = (np.abs(X[:, 0:1] - X[:, 1:2]) < 0.5).astype(int)
T[T == 0] = 10
T[T == 1] = 20
# Unique class labels are now 10 and 20!
nnet = nn.NeuralNetworkClassifier(2, [10, 5], 2)
nnet.train(X, T, X, T, 200, method='scg')
classes, probs = nnet.use(X)
correct_classes = np.array([[20], [20], [10], [20], [10], [20], [20],
[10], [20], [10],
[20], [10], [20], [10], [20], [10], [20], [20], [20], [20]])

SCG: Epoch 20 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 40 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 60 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 80 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 100 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 120 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 140 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 160 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 180 Likelihood= Train 0.58988 Validate 0.58988
SCG: Epoch 200 Likelihood= Train 0.58988 Validate 0.58988

#  and test result with    np.allclose(classes, correct_classes,
atol=0.1)

----------------------------------------------------------------------
----  0/10 points. Incorrect values in classes.
----------------------------------------------------------------------


======================================================================
==========
Testing this for 10 points:

correct_probs = np.array([[5.02457605e-09, 9.99999995e-01],
[8.62009522e-10, 9.99999999e-01],
[9.99999999e-01, 9.29113040e-10], [1.26059447e-09, 9.99999999e-01],
[1.00000000e+00, 6.39547085e-13], [2.36254327e-09, 9.99999998e-01],
[1.34693543e-09, 9.99999999e-01], [9.99999960e-01, 3.96256246e-08],
[7.25882159e-09, 9.99999993e-01], [1.00000000e+00, 1.41558454e-15],
[2.09966039e-10, 1.00000000e+00], [1.00000000e+00, 3.09418630e-16],
[2.01456195e-10, 1.00000000e+00], [1.00000000e+00, 2.09626683e-16],
[1.72899120e-09, 9.99999998e-01], [9.99999998e-01, 2.33382708e-09],
[2.19039065e-10, 1.00000000e+00], [2.38235718e-10, 1.00000000e+00],
[3.10731426e-09, 9.99999997e-01], [8.26588031e-10, 9.99999999e-01]])
```

```
#  and test result with    np.allclose(probs, correct_probs, atol=0.1)

----------------------------------------------------------------------
----  0/10 points. Incorrect values in probs.
----------------------------------------------------------------------

======================================================================
D:\A4 Execution Grade is 50 / 80
======================================================================


-- / 5 points. Experiment with the three different optimization
methods,
               at least three hidden layer structures including [],
two
               learning rates, and two numbers of epochs. Use
verbose=False
               as an argument to train(). For scg, ignore the learning
rate
               loop. Print a single line for each run showing method,
number
               of epochs, learning rate, hidden layer structure, and
percent
               correct for training, validation, and testing data.

__ / 5 points. Function make_mnist_classifier defined and used
correctly.

__ / 5 points. Discuss your results. In your discussion, include
observations about
                which method achieves the best result,
                which method seems to do best with fewer epochs,
                what common classification mistakes are made as shown
in your confusion matrices, and
                do larger networks (more layers, more units) work
better than small networks?

__ / 5 points. Train a network with values for method, learning rate,
number of epochs,
               and a hidden layer structure with no more than 100
units in the first layer
               that you found work well. Extract the weight matrix
from the first layer. Now,
               for each unit (column in the weight matrix) ignore the
first row of bias weights
               and reshape the remaining weights into a 28 x 28 image
for each unit and display
               them. Complete the function to draw the weight matrix
for one unit using draw_digit
```

```
               as a guide, then use it in a loop to draw the weight
matrices for each unit in
               the first layer of your network.
               Discuss what you see. Describe some of the images as
patterns that could be
               useful for classifying particular digits.

======================================================================
D:\A4 Results and Discussion Grade is ___ / 20
======================================================================

======================================================================
D:\A4 FINAL GRADE is  _  / 100
======================================================================

Extra Credit (2 points possible):

Extra Credit for 1 point:

Repeat the above experiments with a different classification data set.
Randonly partition
your data into training, validaton and test parts if not already
provided.  Write in
markdown cells descriptions of the data and your results.
of the data and your results.

Extra Credit for 1 point:

Train a network with values for method, learning rate, number of
epochs, and a
hidden layer structure with no more than 100 units in the first layer
that you
found work well.  Extract the weight matrix from the first layer.

Now, for each unit (column in the weight matrix) ignore the first row
of bias weights and
reshape the remaining weights into a 28 x 28 image for each unit and
display them.
Complete the following function to draw the weight matrix for one unit
using `draw_digit`
as a guide, then use it in a loop to draw the weight matrices for each
unit in the first
layer of your network.

Discuss what you see.  Describe some of the images as patterns that
could be useful for classifying particular digits.


D:\A4 EXTRA CREDIT is 0 / 2
```

Extra Credit (2 points possible)

Extra Credit for 1 Point

Repeat the above experiments with a different classification data set. Randonly partition your data into training, validaton and test parts if not already provided. Write in markdown cells descriptions of the data and your results.

Extra Credit for 1 Point

Train a network with values for method, learning rate, number of epochs, and a hidden layer structure with no more than 100 units in the first layer that you found work well. Extract the weight matrix from the first layer. Now, for each unit (column in the weight matrix) ignore the first row of bias weights and reshape the remaining weights into a 28 x 28 image for each unit and display them. Complete the following function to draw the weight matrix for one unit using `draw_digit` as a guide, then use it in a loop to draw the weight matrices for each unit in the first layer of your network.

Discuss what you see. Describe some of the images as patterns that could be useful for classifying particular digits.

```
def draw_weight_matrix(W, unit_index = 0):
    """W is matrix of weights, with shape 784 x n_units in first layer of neural network"""
    ...

W = nnet.Ws[0]
n_units = W.shape[1]
n_plot_rows = round(np.sqrt(n_units) + 0.5)
n_plot_cols = n_plot_rows

plt.figure(figsize=(10, 10))
for i in range(n_units):

    ...

plt.tight_layout()
```