Introduction to Deep Learning - Homework 1

Josh Green

Problem 1: K-Nearest Neighbors (KNN)

This first problem asks us to use a KNN to recognize handwritten digits.

Dataset Information

The MNIST dataset can be downloaded using the `download_mnist.py` script. This script generates an `mnist.pkl` file that contains the data converted to numpy arrays. The MNIST dataset consists of:

- `x_train`: A 60,000x784 numpy array containing flattened training images.
- `y_train`: A 1x60,000 numpy array that corresponds to the true label of the corresponding training images.
- `x_test`: A 10,000x784 numpy array where each row contains flattened versions of test images.
- `y_test`: A 1x10,000 numpy array where each component is the true label of the corresponding test images.

There are four compressed files representing the above sets of numpy arrays:

- `train-images-idx3-ubyte.gz`: contains the `x_train` data
- `train-labels-idx1-ubyte.gz`: contains the `y_train` data
- `t10k-images-idx3-ubyte.gz`: contains the `x_test` data
- `t10k-labels-idx1-ubyte.gz`: contains the `y_test` data

Source Code for `download_mnist.py`

```
import numpy as np
from urllib import request
import gzip
import pickle

filename = [
    ["training_images", "train-images-idx3-ubyte.gz"],
    ["test_images", "t10k-images-idx3-ubyte.gz"],
    ["training_labels", "train-labels-idx1-ubyte.gz"],
    ["test_labels", "t10k-labels-idx1-ubyte.gz"]
]

def download_mnist():
    base_url = "https://ossci-datasets.s3.amazonaws.com/mnist/"
    for name in filename:
        print("Downloading " + name[1] + "...")
        request.urlretrieve(base_url + name[1], name[1])
    print("Download complete.")

def save_mnist():
```

```
    mnist = {}
    for name in filename[:2]:
        with gzip.open(name[1], 'rb') as f:
            mnist[name[0]] = np.frombuffer(f.read(), np.uint8,
offset=16).reshape(-1, 28*28)
    for name in filename[-2:]:
        with gzip.open(name[1], 'rb') as f:
            mnist[name[0]] = np.frombuffer(f.read(), np.uint8, offset=8)
    with open("mnist.pkl", 'wb') as f:
        pickle.dump(mnist, f)
    print("Save complete.")

def init():
    download_mnist()
    save_mnist()

def load():
    with open("mnist.pkl", 'rb') as f:
        mnist = pickle.load(f)
    return mnist["training_images"], mnist["training_labels"],
mnist["test_images"], mnist["test_labels"]

if __name__ == '__main__':
    init()
```

KNN Information

The `knn.py` script imports the `mnist.pkl` data, looping through a specified data set performing distance calculations. Then, the k-nearest neighbors are calculated to classify the image.

There are two distance calculations that can be used:

- L1 distance is the Manhattan distance, defined as $$ D = \sum_{i=1}^{n} |x_i - y_i|$$
- L2 distnace is the Euclidian distance, defined as $$ D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$ **NOTE:** The knn.py script implements the L2 distance formula

Source Code for `knn.py`

```
import math
import numpy as np
from download_mnist import load
import operator
import time

x_train, y_train, x_test, y_test = load()
x_train = x_train.reshape(60000, 28, 28)
x_test  = x_test.reshape(10000, 28, 28)
x_train = x_train.astype(float)
x_test = x_test.astype(float)

def kNNClassify(newInput, dataSet, labels, k):
```

```
    result = []
    for test_sample in newInput:
        distances = []
        for train_image in dataSet:
            distance = np.sqrt(np.sum((test_sample - train_image) ** 2)) #
L2 Distance
            distances.append(distance)

        k_neighbors = np.argsort(distances)[:k]
        k_labels = labels[k_neighbors]
        predicted_label = np.bincount(k_labels).argmax()
        result.append(predicted_label)
    return result

start_time = time.time()
outputlabels = kNNClassify(x_test[0:20], x_train, y_train, 7)
result = y_test[0:20] - outputlabels
result = (1 - np.count_nonzero(result) / len(outputlabels))
print("---classification accuracy for knn on mnist: %s ---" % result)
print("---execution time: %s seconds ---" % (time.time() - start_time))
```

Output from `knn.py`

The output from `knn.py` is shown below for a k value of 7 and for 20 images classified.

```
---classification accuracy for knn on mnist: 1.0 ---
---execution time: 4.4463958740234375 seconds ---
```

KNN Tester Information

The `knn_tester.py` script is a modified version of `knn.py` that runs the knn with both L1 and L2 distances as well as with different k values.

NOTE: `knn_tester.py` is classifying 1000 images compared to `knn.py` only classifying 20 images. This was done to get a better representation of the accuracy of the model, and is also why the reported times by the scripts are different.

`knn_tester.py` Script

```
import math
import numpy as np
from download_mnist import load
import operator
import time

x_train, y_train, x_test, y_test = load()
x_train = x_train.reshape(60000,28,28)
x_test  = x_test.reshape(10000,28,28)
x_train = x_train.astype(float)
```

```
x_test = x_test.astype(float)

def kNNClassify(newInput, dataSet, labels, k, distance_metric='L2'):
    result=[]
    for test_sample in newInput:
        distances = []

        for train_image in dataSet:
            if distance_metric == 'L2':
                distance = np.sqrt(np.sum((test_sample - train_image) ** 2)) #L2 Distance
            else:
                distance = np.sum(np.abs(test_sample - train_image)) #L1 Distance
            distances.append(distance)

        k_neighbors = np.argsort(distances)[:k]
        k_labels = labels[k_neighbors]
        predicted_label = np.bincount(k_labels).argmax()
        result.append(predicted_label)
    return result

# Test both L1 and L2 distances
for distance_type in ['L1', 'L2']:
    print(f"\nTesting {distance_type} Distance:")
    print("=" * 50)

    # Loop through different k values
    for k in range(1, 11):
        start_time = time.time()
        outputlabels = kNNClassify(x_test[0:1000], x_train, y_train, k, distance_type)
        result = y_test[0:1000] - outputlabels
        accuracy = (1 - np.count_nonzero(result)/len(outputlabels))
        execution_time = time.time() - start_time

        print(f"k={k}:")
        print(f"---classification accuracy for knn on mnist: {accuracy:.4f} ---")
        print(f"---execution time: {execution_time:.2f} seconds ---")
        print("-" * 50)
```

Output from `knn_tester.py`

```

Testing L1 Distance:
==================================================
k=1:
---classification accuracy for knn on mnist: 0.9500 ---
---execution time: 123.90 seconds ---
--------------------------------------------------
```

```
k=2:
---classification accuracy for knn on mnist: 0.9430 ---
---execution time: 126.50 seconds ---
--------------------------------------------------
k=3:
---classification accuracy for knn on mnist: 0.9530 ---
---execution time: 128.91 seconds ---
--------------------------------------------------
k=4:
---classification accuracy for knn on mnist: 0.9460 ---
---execution time: 128.22 seconds ---
--------------------------------------------------
k=5:
---classification accuracy for knn on mnist: 0.9510 ---
---execution time: 128.85 seconds ---
--------------------------------------------------
k=6:
---classification accuracy for knn on mnist: 0.9490 ---
---execution time: 127.98 seconds ---
--------------------------------------------------
k=7:
---classification accuracy for knn on mnist: 0.9460 ---
---execution time: 128.89 seconds ---
--------------------------------------------------
k=8:
---classification accuracy for knn on mnist: 0.9420 ---
---execution time: 128.22 seconds ---
--------------------------------------------------
k=9:
---classification accuracy for knn on mnist: 0.9410 ---
---execution time: 128.45 seconds ---
--------------------------------------------------
k=10:
---classification accuracy for knn on mnist: 0.9340 ---
---execution time: 129.16 seconds ---
--------------------------------------------------

Testing L2 Distance:
==================================================
k=1:
---classification accuracy for knn on mnist: 0.9620 ---
---execution time: 149.86 seconds ---
--------------------------------------------------
k=2:
---classification accuracy for knn on mnist: 0.9480 ---
---execution time: 150.23 seconds ---
--------------------------------------------------
k=3:
---classification accuracy for knn on mnist: 0.9620 ---
---execution time: 150.35 seconds ---
--------------------------------------------------
k=4:
---classification accuracy for knn on mnist: 0.9640 ---
---execution time: 149.90 seconds ---
```

```
--------------------------------------------------
k=5:
---classification accuracy for knn on mnist: 0.9610 ---
---execution time: 150.04 seconds ---
--------------------------------------------------
k=6:
---classification accuracy for knn on mnist: 0.9590 ---
---execution time: 150.51 seconds ---
--------------------------------------------------
k=7:
---classification accuracy for knn on mnist: 0.9620 ---
---execution time: 152.00 seconds ---
--------------------------------------------------
k=8:
---classification accuracy for knn on mnist: 0.9580 ---
---execution time: 150.28 seconds ---
--------------------------------------------------
k=9:
---classification accuracy for knn on mnist: 0.9520 ---
---execution time: 149.62 seconds ---
--------------------------------------------------
k=10:
---classification accuracy for knn on mnist: 0.9540 ---
---execution time: 150.26 seconds ---
--------------------------------------------------
```

Observations on L1 and L2 Distances and Accuracy as K Changes

After running the `knn_tester.py` script, it can be observed that the accuracy improved and the runtime increased when using the L2 distance formula. From my small sample size, the reported times don't show a correlation between accuracy and an increased K value. As K increases, we initially see the accuracy increase, but we eventually get to a point where we start considering neighbors that are unrelated to an image we are looking at, which can account for the lower accuracy as K gets closer to 10.

Problem 2: Linear Classifier

The second problem asks us to train a linear classifier to recognize handwritten digits.

The `linear_classifier.py` script implements a linear classifier to accomplish handwriting recognition. This linear classifier uses "Cross Entropy" for the loss function and employs "Random Search" to find the parameters W. Afterwards, the accuracy of the linear classifier is tested using the MNIST testing set.

Source Code for `linear_classifier.py`

```
import numpy as np
from download_mnist import load
from sklearn.preprocessing import OneHotEncoder
import time
```

```
# Load MNIST dataset
x_train, y_train, x_test, y_test = load()
x_train = x_train.reshape(60000, -1) / 255.0
x_test = x_test.reshape(10000, -1) / 255.0

# One-hot encode labels
encoder = OneHotEncoder(sparse_output=False)
y_train_onehot = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_onehot = encoder.transform(y_test.reshape(-1, 1))

# Hyperparameters
num_classes = 10
num_features = x_train.shape[1]
learning_rate = 0.1
num_epochs = 100
batch_size = 128

# Initialize weights
W = np.random.randn(num_features, num_classes) / np.sqrt(num_features)
start_time = time.time()

# Training loop
for epoch in range(num_epochs):
    # Shuffle training data
    indices = np.random.permutation(len(x_train))
    x_train_shuffled = x_train[indices]
    y_train_shuffled = y_train_onehot[indices]

    # Mini-batch training
    for i in range(0, len(x_train), batch_size):
        batch_x = x_train_shuffled[i:i + batch_size]
        batch_y = y_train_shuffled[i:i + batch_size]

        # Forward pass
        scores = batch_x @ W
        exp_scores = np.exp(scores - np.max(scores, axis=1,
keepdims=True))
        probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

        # Backward pass
        dscores = probs - batch_y
        dW = batch_x.T @ dscores

        # Update weights
        W -= learning_rate * dW / batch_size

    # Evaluate on test set
    if (epoch + 1) % 10 == 0:
        test_scores = x_test @ W
        test_preds = np.argmax(test_scores, axis=1)
        accuracy = np.mean(test_preds == y_test)
        print(f'Epoch {epoch + 1}, Test accuracy: {accuracy:.4f}')

print("---execution time: %s seconds ---" % (time.time() - start_time))
```

```
```

Results for `linear_classifier.py`

```
Epoch 10, Test accuracy: 0.9195
Epoch 20, Test accuracy: 0.9228
Epoch 30, Test accuracy: 0.9230
Epoch 40, Test accuracy: 0.9255
Epoch 50, Test accuracy: 0.9248
Epoch 60, Test accuracy: 0.9252
Epoch 70, Test accuracy: 0.9243
Epoch 80, Test accuracy: 0.9255
Epoch 90, Test accuracy: 0.9251
Epoch 100, Test accuracy: 0.9259
---execution time: 13.739806652069092 seconds ---
```