Stevens Institute of Technology

Department of Electrical Engineering

EE628: Data Acquisition/Proc II

Homework Assignment 3

Submitted by:
Hadia Hameed
CWID: 10440803

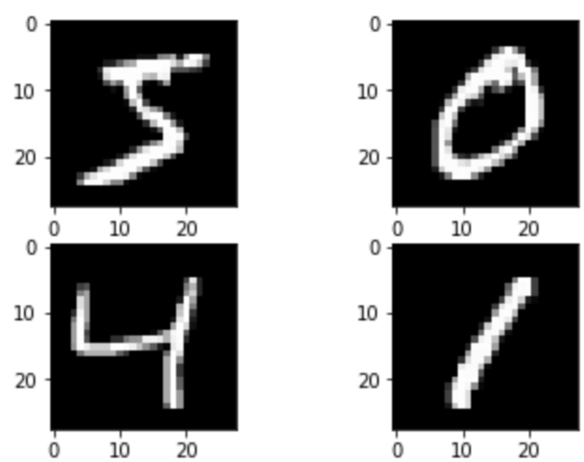
Submitted to:
Prof. Rensheng Wang

February 18, 2019

1 Problem 3: Working with MNIST dataset

1.1 Plotting a few instances of the training data:

```
# Plot ad hoc mnist instances
from keras.datasets import mnist
import matplotlib.pyplot as plt
# load (downloaded if needed) the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
# show the plot
plt.show()
```



1.2 Flattening the images:

Initially we has 6000 images each of 28x28 size. We flatten them out into 6000 vectors each of size 784x1.

In [2]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
import numpy as np

# fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')
```

In [3]:
```
X_train.shape
```

Out[3]:
```
(60000, 784)
```

1.3 Normalizing pixel values:

The pixel intensity values are between 0-255 as shown below.

In [9]:
```
X_train[0,180:200]
```

Out[9]:
```
array([170., 253., 253., 253., 253., 253., 225., 172., 253., 242., 195.,
        64.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.],
      dtype=float32)
```

So they are normalized between 0-1 range.

In [10]:
```
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
```

In [11]:
```
X_train[0,180:200]
```

Out[11]:
```
array([0.6666667 , 0.99215686, 0.99215686, 0.99215686, 0.99215686,
       0.99215686, 0.88235295, 0.6745098 , 0.99215686, 0.9490196 ,
       0.7647059 , 0.2509804 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ],
      dtype=float32)
```


1.4 One-hot encoding of the target values:

Their are 10 target classes from 0-9 (representing digits).

```
In [12]: y_train
```

```
Out[12]: array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

We do one-hot encoding, creating 6000 vectors each of size 10x1 e.g. a vector [1,0,0,0,0,0,0,0,0,0] indicates that the target class is '0', vector [0,1,0,0,0,0,0,0,0,0] indicates that the target class is '1', vector [0,0,1,0,0,0,0,0,0,0] indicates that the target class is '2' and so on.

```
In [13]: # one hot encode outputs
         y_train = np_utils.to_categorical(y_train)
         y_test = np_utils.to_categorical(y_test)
         num_classes = y_test.shape[1]
```

```
In [14]: y_train
```

```
Out[14]: array([[0., 0., 0., ..., 0., 0., 0.],
                [1., 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., 1., 0.]], dtype=float32)
```

1.5 Building a Sequential Neural Network:

We build a sequential model for a neural network with an input layer, a single hidden layer with 784 neurons and an output layer with 10 nodes. The activation function between the input and hidden layer is Rectified Linear Unit (ReLu) i.e. $A = max(0, x)\ where\ x\ is\ the\ input.$ The activation function at the output is softmax i.e. $\frac{1}{1+e^{-x}}$.

```
In [17]: # define baseline model
         def baseline_model():
          # create model
          model = Sequential()
          model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal', activation='rel
          model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
          # Compile model
          model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
          return model
```

```
In [18]: # build the model
         model = baseline_model()
```

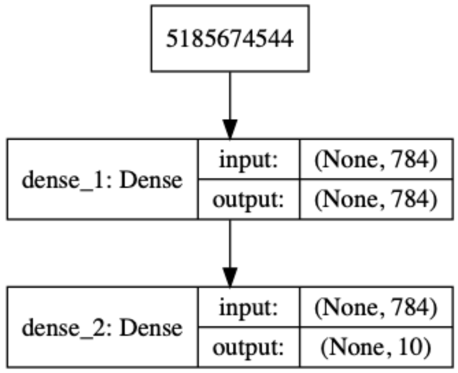


Figure 1: Sequential model for an Artificial Neural network with the input layer having 784 input nodes, a hidden layer with 784 neurons and an output layer with 10 output nodes.

1.6 Training the Neural Network:

We use epochs = 10 (number of iterations) and batch size = 200 which means for each iteration, 200 images will be randomly fed to the network for training.

```
In [31]: # Fit the model
         model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200, verbos
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 3s - loss: 0.0079 - acc: 0.9984 - val_loss: 0.0613 - val_acc: 0.9812
Epoch 2/10
 - 3s - loss: 0.0048 - acc: 0.9994 - val_loss: 0.0580 - val_acc: 0.9832
Epoch 3/10
 - 3s - loss: 0.0025 - acc: 0.9998 - val_loss: 0.0584 - val_acc: 0.9833
Epoch 4/10
 - 3s - loss: 0.0027 - acc: 0.9996 - val_loss: 0.0652 - val_acc: 0.9822
Epoch 5/10
 - 3s - loss: 0.0062 - acc: 0.9984 - val_loss: 0.0841 - val_acc: 0.9780
Epoch 6/10
 - 3s - loss: 0.0100 - acc: 0.9967 - val_loss: 0.0782 - val_acc: 0.9784
Epoch 7/10
 - 3s - loss: 0.0088 - acc: 0.9971 - val_loss: 0.0765 - val_acc: 0.9791
Epoch 8/10
 - 3s - loss: 0.0031 - acc: 0.9995 - val_loss: 0.0641 - val_acc: 0.9838
Epoch 9/10
 - 3s - loss: 8.9920e-04 - acc: 1.0000 - val_loss: 0.0647 - val_acc: 0.9841
Epoch 10/10
 - 3s - loss: 4.2943e-04 - acc: 1.0000 - val_loss: 0.0629 - val_acc: 0.9838
```

1.7 Evaluating performance on test data:

We use categorical cross-entropy as the loss function. It is given by:

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

where C is the number of classes which, in our case, is 10, t is the true label and $f(s)_i$ is the predicted output for input s.

Accuracy is 98.38%, cross-entropy loss is 0.0629 and baseline error is 100 - 98.38 = 1.62%

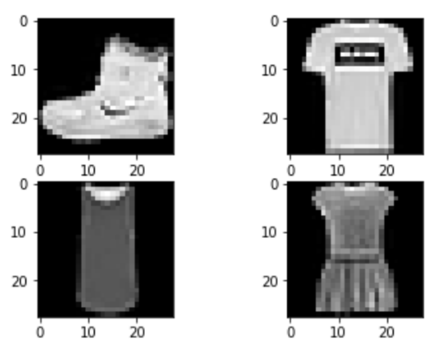
In [32]:
```
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print(scores)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```

```
[0.06293792360295525, 0.9838]
Baseline Error: 1.62%
```

2 Problem 4: Working with FASHION_MNIST dataset

2.1 Plotting a few instances of the training data:

In [36]:
```
# Plot ad hoc mnist instances
from keras.datasets import fashion_mnist
import matplotlib.pyplot as plt
# load (downloaded if needed) the FASHION_MNIST dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
# show the plot
plt.show()
```



2.2 Flattening the images:

Initially we has 6000 images each of 28x28 size. We flatten them out into 6000 vectors each of size 784x1.

```
In [7]: X_train.shape

Out[7]: (60000, 28, 28)
```

```
In [9]: from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Dropout
        from keras.utils import np_utils
        import numpy as np

        # fix random seed for reproducibility
        seed = 9
        np.random.seed(seed)
        # flatten 28*28 images to a 784 vector for each image
        num_pixels = X_train.shape[1] * X_train.shape[2]
        print("Number of images: ", str(X_train.shape[0]))
        print("Number of pixels in each image: ", str(num_pixels))

        Number of images:  60000
        Number of pixels in each image:  784
```

2.3 Normalizing pixel values:

The pixel intensity values are between 0-255 as shown below.

```
In [10]: X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
         X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

In [11]: X_train[0,180:200]

Out[11]: array([  0.,   0., 155., 236., 207., 178., 107., 156., 161., 109.,  64.,
                 23.,  77., 130.,  72.,  15.,   0.,   0.,   0.,   0.],
               dtype=float32)
```

So they are normalized between 0-1 range.

```
In [12]: # normalize inputs from 0-255 to 0-1
         X_train = X_train / 255
         X_test = X_test / 255

In [13]: X_train[0,180:200]

Out[13]: array([0.        , 0.        , 0.60784316, 0.9254902 , 0.8117647 ,
                0.69803923, 0.41960785, 0.6117647 , 0.6313726 , 0.42745098,
                0.2509804 , 0.09019608, 0.3019608 , 0.50980395, 0.28235295,
                0.05882353, 0.        , 0.        , 0.        , 0.        ],
               dtype=float32)
```


2.4 One-hot encoding of the target values:

Their are 10 target classes from 0-9.

```
In [13]: np.unique(y_train)

Out[13]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

We do one-hot encoding, creating 6000 vectors each of size 10x1 e.g. a vector [1,0,0,0,0,0,0,0,0,0] indicates that the target class is '0', vector [0,1,0,0,0,0,0,0,0,0] indicates that the target class is '1', vector [0,0,1,0,0,0,0,0,0,0] indicates that the target class is '2' and so on.

```
In [14]: # one hot encode outputs
         y_train = np_utils.to_categorical(y_train)
         y_test = np_utils.to_categorical(y_test)
         num_classes = y_test.shape[1]

In [16]: y_train

Out[16]: array([[0., 0., 0., ..., 0., 0., 1.],
                [1., 0., 0., ..., 0., 0., 0.],
                [1., 0., 0., ..., 0., 0., 0.],
                ...,
                [0., 0., 0., ..., 0., 0., 0.],
                [1., 0., 0., ..., 0., 0., 0.],
                [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

2.5 Building a Sequential Neural Network:

We build a sequential model for a neural network with an input layer, a single hidden layer with 784 neurons and an output layer with 10 nodes. The activation function between the input and hidden layer is Rectified Linear Unit (ReLu) i.e. $A = max(0, x)$ *where x is the input*. The activation function at the output is softmax i.e. $\frac{1}{1+e^{-x}}$.

```
In [17]: # define baseline model
         def baseline_model():
          # create model
          model = Sequential()
          model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal', activation='rel
          model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
          # Compile model
          model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
          return model

In [18]: # build the model
         model = baseline_model()
```

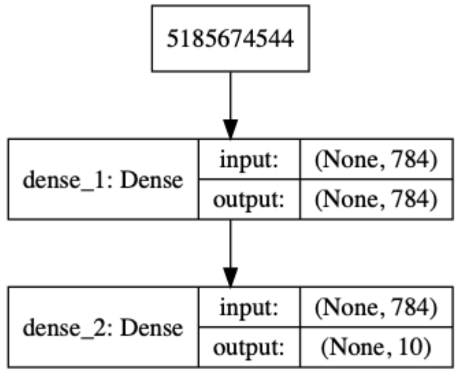



Figure 2: Sequential model for an Artificial Neural network with the input layer having 784 input nodes, a hidden layer with 784 neurons and an output layer with 10 output nodes.

2.6 Training the Neural Network:

We use epochs = 10 (number of iterations) and batch size = 200 which means for each iteration, 200 images will be randomly fed to the network for training.

```
In [20]: # Fit the model
         model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=200, verbose=2)

Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 3s - loss: 0.2186 - acc: 0.9193 - val_loss: 0.3095 - val_acc: 0.8917
Epoch 2/10
 - 3s - loss: 0.2063 - acc: 0.9232 - val_loss: 0.3129 - val_acc: 0.8896
Epoch 3/10
 - 3s - loss: 0.2020 - acc: 0.9246 - val_loss: 0.3131 - val_acc: 0.8882
Epoch 4/10
 - 3s - loss: 0.1911 - acc: 0.9288 - val_loss: 0.3400 - val_acc: 0.8857
Epoch 5/10
 - 3s - loss: 0.1818 - acc: 0.9320 - val_loss: 0.3289 - val_acc: 0.8885
Epoch 6/10
 - 3s - loss: 0.1813 - acc: 0.9324 - val_loss: 0.3283 - val_acc: 0.8899
Epoch 7/10
 - 3s - loss: 0.1739 - acc: 0.9356 - val_loss: 0.3172 - val_acc: 0.8947
Epoch 8/10
 - 3s - loss: 0.1659 - acc: 0.9382 - val_loss: 0.3427 - val_acc: 0.8854
Epoch 9/10
 - 3s - loss: 0.1595 - acc: 0.9409 - val_loss: 0.3226 - val_acc: 0.8953
Epoch 10/10
 - 3s - loss: 0.1537 - acc: 0.9443 - val_loss: 0.3191 - val_acc: 0.8950
```

2.7 Evaluating performance on test data:

We use categorical cross-entropy as the loss function. It is given by:

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

where C is the number of classes which, in our case, is 10, t is the true label and $f(s)_i$ is the predicted output for input s.

Accuracy is 89.50%, cross-entropy loss is 0.319 and baseline error is 100 - 89.50 = 10.50%

In [23]:
```
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print(scores)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```

```
[0.31908862102627755, 0.895]
Baseline Error: 10.50%
```

2.8 Tuning batch size and number of epochs:

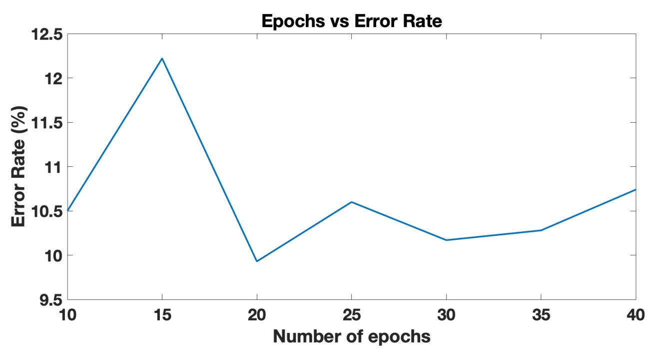


Figure 3: Change in error rate with respect to the number of epochs, with batch size fixed at 200.

Figure 4: Change in error rate with respect to batch size with number of epochs fixed at 10.

