EN2550 – ASSIGNMENT 04

1) Part 1

a.

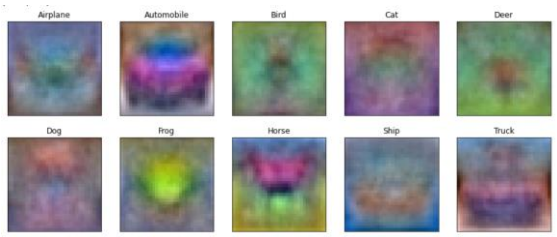
The bias term (b1) which is found here is combined to the weight matrix (w1) and the additional row of ones is inserted to the x_train dataset in order to ease out the matrix multiplication operations carried out.

```
#updating the x_train and x_test
x_train_new = np.concatenate((np.ones((Ntr,1)), x_train), axis = 1)
x_test_new = np.concatenate((np.ones((Nte,1)), x_test), axis = 1)

#updating the w1 by concatenating with the bias vector
w1 = np.concatenate((b1.reshape(1, K), w1), axis = 0)
```

As a linear classifier is used in the exercise the relevant hypothesis derivation and loss calculations in the forward pass and the back propagation algorithm which is implemented by gradient descent I shown here. The model is iterated for 300 epochs in order to optimize with an initial learning rate which is decayed in each operation along with regularization.

```
# FORWARD PASS

h_theta = np.matmul(x_train_new, w1) #calculating the hypothesis

loss = (1/(2*m))*np.sum((h_theta - y_train)**2) + (reg/(2*m))*np.sum(w1**2) #mean sum of squared errors.
#Here the bias term is not considered for regularitation. This summing goves the row wise summation

# BACKWARD PASS (IMplementing Gradient Descent)

#differentiate w.r.t w1
d_loss = (1/m)*np.matmul(x_train_new.T, (h_theta - y_train))+ (reg/m)*w1
w1 = w1 - lr*d_loss
#-------------------------------------------------------------
```

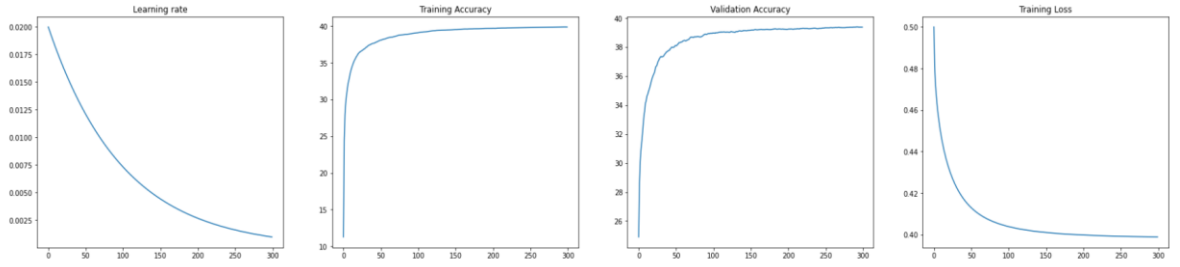
b. The weights matrix W as 10 images;



c.

Initial learning rate = 2.e-2

```
Training_accuracy:  39.867999999999995
Test_accuracy:  39.379999999999995
```



2) Part 2

a. The sigmoid function can be implemented as follows;

```
def sigmoid(x, w1, b1):
    sgm = 1/(1.0 + np.exp(-x.dot(w1) - b1))
    return sgm
```

The 2-layer neural network consist of 200 hidden layers and is iterated for 300 epochs for optimization using gradient descent.

The forward pass;

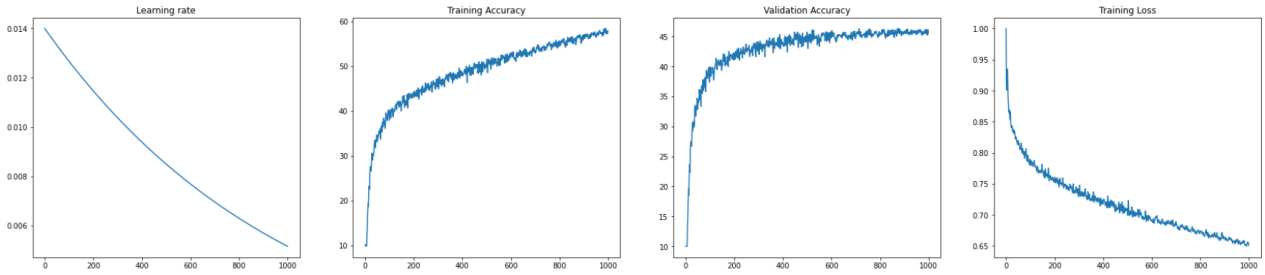
```
h = sigmoid(x,w1,b1)
y_pred = h.dot(w2) + b2
loss = 1./batch_size*np.square(y_pred - y).sum() + reg * (np.sum(w2*w2) + np.sum(w1*w1))
```

The backward pass;

```
dy_pred = 1./batch_size*2.0*(y_pred - y) #this is the partial derivative of loss w.r.t dy_pred
dw2 = h.T.dot(dy_pred) + reg*w2
db2 = dy_pred.sum(axis = 0)
dh = dy_pred.dot(w2.T)
dw1 = x.T.dot(dh*h*(1-h)) + reg*w1
db1 = (dh*h*(1-h)).sum(axis = 0)
w1 = w1 -  lr*dw1
w2 = w2 - lr*dw2
b1 = b1 - lr*db1
b2 = b2 - lr*db2
lr = lr*lr_decay
```

b. The initial learning rate was set to 1.4e-2 and the preceding accuracies and losses obtained were as follows.

```
Training_accuracy:  57.926
Test_accuracy:  46.03
Learning Rate: 0.005147735946793495
Testing Loss:  0.7397296123274232
```



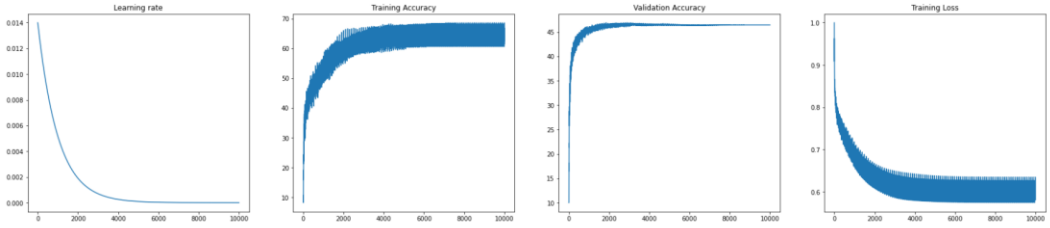
3) Part 3

a.

The following results were obtained when stochastic gradient descent is run with batch sizes of 500.

```
Training Done!!
Loss:  0.6038600571490382
Training_accuracy:  62.6
Test_accuracy:  46.44
Learning Rate: 6.324268436786747e-07
Testing Loss:  0.7395193459330253
```

These results were obtained for an initial learning rate of 1.4e-2 and a regularization factor of 5e-6.



b.

- There is significant decrease in the training time consumed in the cases of training the models using traditional gradient descent and stochastic gradient descent. The reason behind is unlike in traditional methods of gradient descent optimization the whole batch of training data is not considered in each iteration, but only a pre-defined batches of data is randomly selected. The optimization is carried out for each mini-batch and the whole process is iterated finite times.

- There is a increment in Training accuracy in the stochastic gradient descent which is obvious for the reason that stochastic gradient descent is capable of reaching a global minimum in the loss function more than traditional gradient descent operation which is having a probability of being stuck at a local minima in optimization.

- The curves of training accuracies and training losses in the stochastic gradient descent shows some significant ripple in the graphs. The reason behind might be the local minima which is fount at every single batch leading to sudden deviations in the accuracies and the Loss functions

4) Part 4

a.

Layer	Output shape	Learnable parameters
C32	(1,30,30,32)	3×3×32 + 32 = 320
Max Pooling	(1,15,15,32)	0
C64	(1,13,13,64)	3×3×64×31 + 64 = 18496
Max Pooling	(1,6,6,64)	0
C64	(1,4,4,64)	3×3×64×64 + 64 = 36928
Max Pooling	(1,2,2,64)	0
Flatten	(1,256)	0
F64	(1,64)	256×64 + 64 = 16448
F10	(1,10)	64×10 + 10 = 650

Therefore the total learnable parameter count = 72842

b. Learning rate = 0.05
Learning rate decay = 1e-5
Momentum = 0.8

c.

- Training loss - 0.065
- Training accuracy - 0.9798
- Testing accuracy - 0.978699

The code used for the implementing of the sequential model can be shown as follows;

```
#construct the sequential model
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)))
model.add(layers.MaxPool2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPool2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPool2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(10))
#compiling the model-------------------------------------------------------------------------------------------------
sgd = SGD(lr = 0.05, decay = 1e-5, momentum = 0.8)
model.compile(optimizer = 'sgd',loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
print(model.summary)
#Enable visualization for tensorboard--------------------------------------------------------------------------------
log_dir = ".\\logs\\fit\\" + datetime.datetime.now().strftime("%Y%m%d - %H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
#Training------------------------------------------------------------------------------------------------------------
model.fit(x_train, y_train, epochs = 5, batch_size = 50, callbacks=[tensorboard_callback])
#evaluation----------------------------------------------------------------------------------------------------------
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
```

5) ATTACHMENTS

All the attachments for the code along with their outputs can be visited here:

- https://github.com/praneethperera123/E2550_Assignment-04