Assignment 1

Problem1:

After using MLP to classify the CIFAR-10 dataset into 10 classes, we should modify some parameters to obtain an optimal model to classify the data. The parameters should be modified are: (1) No. of epochs, (2) batch size, (3) No. of nodes in a layer and No. of layers, (4) learning rate (5) activation function, (6) dropout rate.

I modified the parameters 25 times to compare the accuracy and loss to check which parameter influences the model most and generate an optimal model for this case. Appendix A is about the details about the entire process. Appendix B is the code for optimal model I selected.

For the optimal model (state 11) as the table in appendix A, the six parameters are: epochs=30, batch size=512, # layers= 4, the nodes for each hidden layer being 512, 512, learning rate=0.001, “relu” being hidden layer activation function, softmax being output layer activation function, dropout rate=0.2.

*Figure 1: Accuracy for 25 states*

From the figure above, when I adjusted some parameters, the accuracy will turn to around 0.1-- average accuracy for each class, which means the parameters are not suitable for the classification model. State 13 and state 14 is because of learning rate which are 0.05 and 0.1 respectively. And model 18 and model 19 is because of the activation functions. I used “sigmoid” function to be the output layer activation function. state 18 referred to “relu” as hidden layers activation function, and state 19 used “softplus” as hidden layers activation functions. What’s more, from the figure 1, the highest accuracy happened when I made the epochs equal 100 and the activation functions for hidden layers are “softplus”, meanwhile other parameters are the same as the optimal model (state11). But it cannot be the optimal model, because it is overfitting, the difference between train\_acc and val\_acc is 0.2051 which is much more than others.

Figure 2: loss for 25 states

From the figure above, the difference between train\_loss and val\_loss is not too much. The spike occurred when I made the learning rate being 0.005 and 0.01, which means the value of learning rate is not suitable for this case. And the loss of other models is relatively acceptable because I didn’t set a larger epoch. From figure 3 and figure 4 we

During the modifying process, we had to make some trade-off consideration to choose the optimal parameters for the case with the relatively higher accuracy, less computing time, and less loss. After comparing the 25 states I chose the values of parameters as above. The accuracies of the 25 models are around 0.5, which is not a good value for a model to classification. But it is hard to improve the solution. The accuracy of state 11 is 0.4914 which is nearly the highest value among the 25 models. And the computing time is about 340 seconds which is not much more than other models. And the difference between the train\_acc and val\_acc is not relatively greater than others. So it is not obviously but a little bit overfitting for the model. We can use cross validation to reduce the impact of overfitting, also we can use some regularization methods to avoid overfitting through adjusting some parameters of “Dense”.

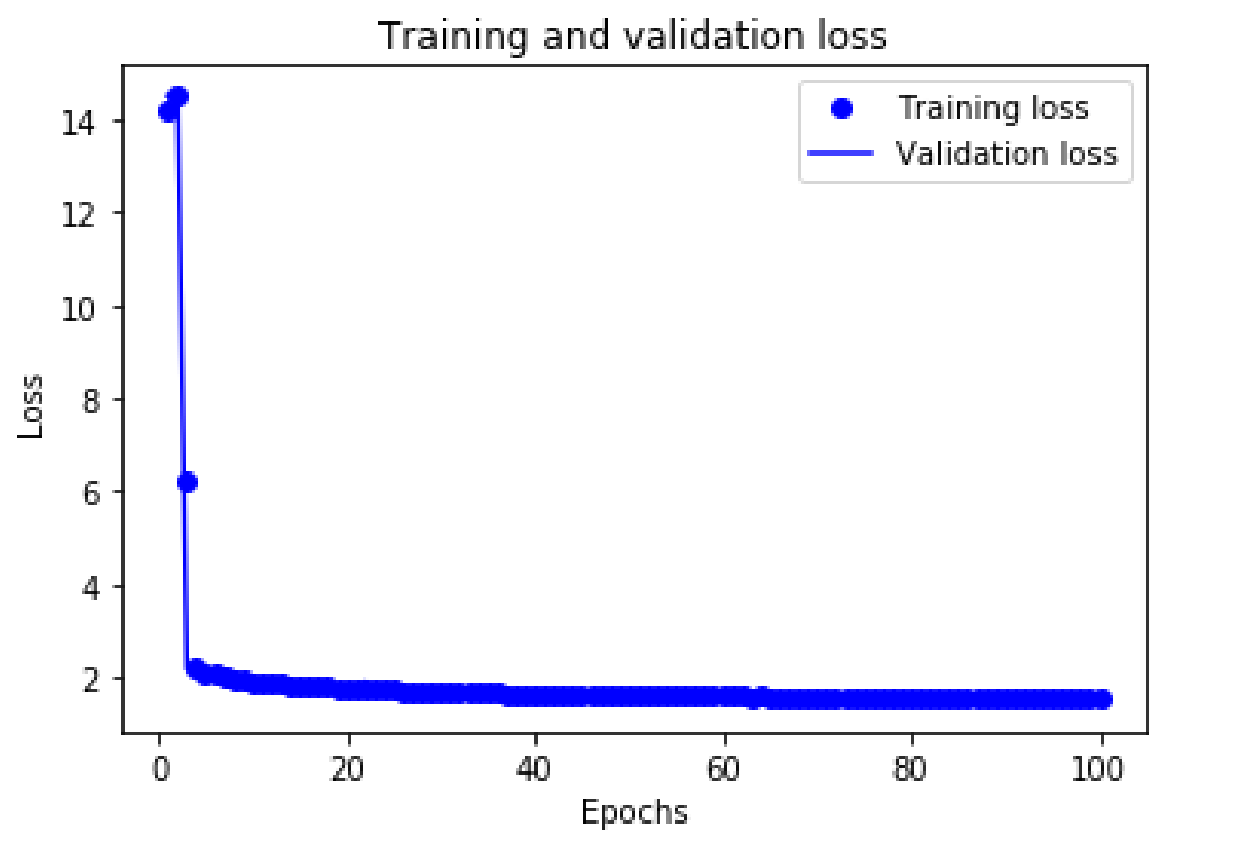
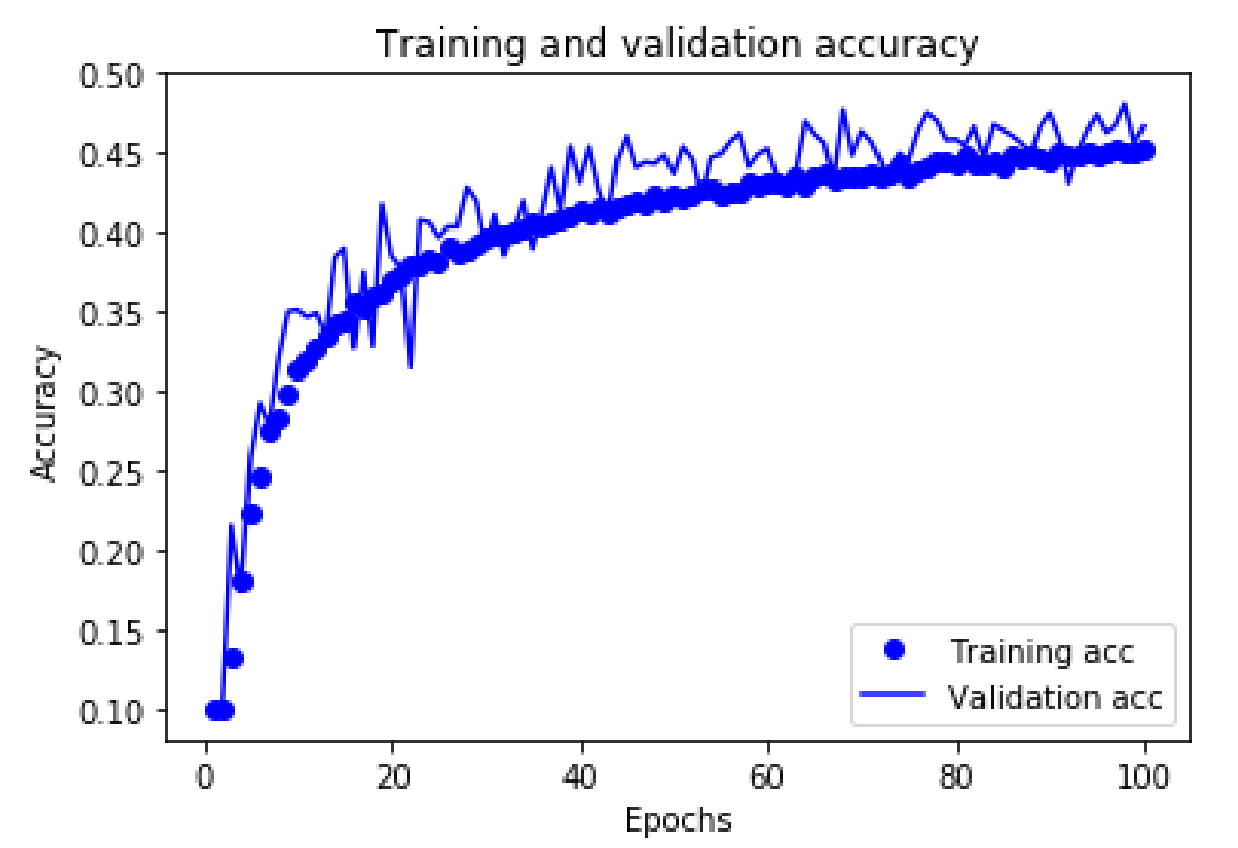
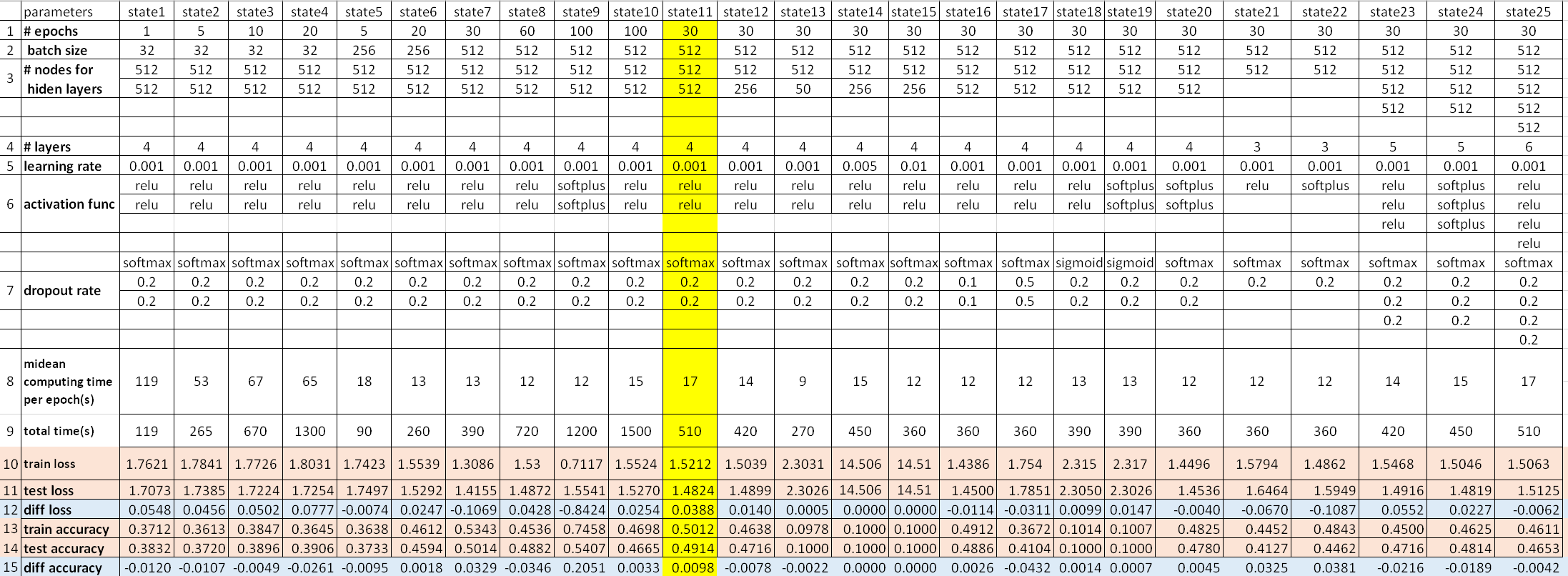
 

Figure 3: loss for 100 epochs (state 10) Figure 4: accuracy for 100 epochs (state 10)

 **Appendix A**

**Appendix B**

Code for optimal model:

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import cifar10

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras import optimizers

from keras.optimizers import RMSprop

import os

(x\_train, y\_train),(x\_test, y\_test) = cifar10.load\_data()

print('x\_train shape: ', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

epochs= 30

batch\_size = 512

num\_classess =10

x\_train = x\_train.reshape(50000,3072)

x\_test = x\_test.reshape(10000,3072)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train = x\_train/255

x\_test = x\_test/255

print('x\_train shape: ', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

y\_train= keras.utils.to\_categorical(y\_train, num\_classess)

y\_test = keras.utils.to\_categorical(y\_test, num\_classess)

model= Sequential()

model.add(Dense(512, activation='relu', input\_shape=(32\*32\*3,)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(num\_classess, activation='softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy',

optimizer=optimizers.RMSprop(lr=0.001),

metrics=['accuracy'])

history= model.fit(x\_train, y\_train,

batch\_size= batch\_size,

epochs=epochs,

verbose=1,

validation\_data=(x\_test,y\_test))

score= model.evaluate(x\_test, y\_test, verbose=0)

print( 'test loss:', score[0])

print('test accuracy:', score[1])

import matplotlib.pyplot as plt

history\_dict=history.history

history\_dict.keys()

history\_dict = history.history

loss\_values = history\_dict['loss']

val\_loss\_values = history\_dict['val\_loss']

epochs = range(1, len(loss\_values) + 1)

plt.plot(epochs, loss\_values, 'bo', label='Training loss')

plt.plot(epochs, val\_loss\_values, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

acc = history\_dict['acc']

val\_acc = history\_dict['val\_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()