HW4

SiCheng Yi

1: Understanding Neural Networks

1. Complete the first Feed Forward iteration of the training process. Show all parts of your work for both the Hidden and Output Layer. Report this iteration's calculated Y. [10pts]

```python
# Import required libraries :
import numpy as np
# Define input features :
input_features = np.array([[0,0],[0,1],[1,0],[1,1]])
print (input_features.shape)
print (input_features)
# Define target output :
target_output = np.array([[0,1,1,1]])
# Reshaping our target output into vector :
target_output = target_output.reshape(4,1)
print(target_output.shape)
print (target_output)
# Define weights :
# 6 for hidden layer
# 3 for output layer
# 9 total
weight_hidden = np.array([[0.1,0.1,0.5],
 [0.9,0.1,0.9]])
weight_output = np.array([[0.2],[0.3],[0.4]])
# Learning Rate :
lr = 0.05
# Sigmoid function :
def sigmoid(x):
 return 1/(1+np.exp(-x))
```

```
[0.9420574]
```

2. Complete the first Backpropagation iteration of the training process. Show all parts of your work for updating each weight. Report the updated set of weights, w 1 - w 9 . [10pts]

```python
# Update Weights
 weight_hidden -= lr * derror_wh
 weight_output -= lr * derror_dwo # TODO: Verify update from

# Final hidden layer weight values :
print(weight_hidden)
# Final output layer weight values :
print(weight_output)
# Predictions :
#Taking inputs :
single_point = np.array([1,1])
#1st step :
result1 = np.dot(single_point, weight_hidden)
#2nd step :
result2 = sigmoid(result1)
#3rd step :
result3 = np.dot(result2,weight_output)
#4th step :
result4 = sigmoid(result3)
print(result4)
#==================================================
#Taking inputs :
single_point = np.array([0,0])
#1st step :
result1 = np.dot(single_point, weight_hidden)
#2nd step :
result2 = sigmoid(result1)
#3rd step :
result3 = np.dot(result2,weight_output)
#4th step :
result4 = sigmoid(result3)
print(result4)
#======================================================
#Taking inputs :
single_point = np.array([1,0])
#1st step :
result1 = np.dot(single_point, weight_hidden)
#2nd step :
result2 = sigmoid(result1)
#3rd step :
result3 = np.dot(result2,weight_output)
#4th step :
result4 = sigmoid(result3)
print(result4)
```

```
[[ 1.44174355 -2.78777929  2.67735506]
 [ 1.88143805 -2.73493536  2.71768941]]
[[ 0.71645927]
 [-7.69080285]
 [ 2.83769964]]
```

3. Complete a second Feed Forward iteration. Show all parts of your work for both the Hidden and Output Layer. Report this iteration's calculated Y and specifically state how your prediction's accuracy has changed, if at all. [10pts]

I completed the second feed forward iterations and in my observation the accuracy remained the same as well as the value for the calculated Y.

2: Programming Neural Networks

1. Using the Iris dataset, implement a Keras classifier that uses a neural network model with one input layer, four hidden layers, and an output layer. The input layer has four neurons, each hidden layer has four neurons, and the output layer has three neurons. Each layer should use the sigmoid activation function. Evaluate the classifier with KFold validation across five splits with shuffling. Run the evaluation 10 times, and report the performance mean and standard deviation for each run. Alongside the performance metrics, take a screenshot of your model's implementation in your source code. [15pts]

```python
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
#   Model :   A Model with Hidden Layers (sigmoid) #
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
def AlternativeModel1():
    """ A sequential Keras model that has an input layer, four
        hidden layers, and an output layer."""
    model = Sequential()
    model.add(Dense(4, input_dim=4, activation='sigmoid', name='layer_1'))
    model.add(Dense(4, activation='sigmoid', name='layer_2'))
    model.add(Dense(4, activation='sigmoid', name='layer_3'))
    model.add(Dense(4, activation='sigmoid', name='layer_4'))
    model.add(Dense(3, activation='sigmoid', name='output_layer'))

    # Don't change this!
    model.compile(loss="categorical_crossentropy",
                  optimizer="adam",
                  metrics=['accuracy'])
    return model


# Model Evaluations
# Below, we build KerasClassifiers using our model definitions.

# - - Model - -
estimator = KerasClassifier(
        build_fn=AlternativeModel1,
        epochs=200, batch_size=20,
        verbose=0)

kfold = KFold(n_splits=5, shuffle=True) #kfold with 5 splits and shuffling

print("- - - - - - - - - - - - - - -")

for i in range(0,10):
    results = cross_val_score(estimator, X_train, y_train, cv=kfold)
    print("(KFOLD MODEL : RUN " + str(i) +") Performance: mean: %.2f%% std: (%.2f%%)" % (results.mean()*100, results.std()*100))
```

(KFOLD MODEL : RUN 0) Performance: mean: 22.86% std: (3.56%)
(KFOLD MODEL : RUN 1) Performance: mean: 27.62% std: (5.55%)
(KFOLD MODEL : RUN 2) Performance: mean: 27.62% std: (9.71%)
(KFOLD MODEL : RUN 3) Performance: mean: 26.67% std: (16.11%)
(KFOLD MODEL : RUN 4) Performance: mean: 34.29% std: (9.23%)
(KFOLD MODEL : RUN 5) Performance: mean: 33.33% std: (6.02%)
(KFOLD MODEL : RUN 6) Performance: mean: 29.52% std: (5.55%)
(KFOLD MODEL : RUN 7) Performance: mean: 23.81% std: (5.22%)
(KFOLD MODEL : RUN 8) Performance: mean: 32.38% std: (8.19%)
(KFOLD MODEL : RUN 9) Performance: mean: 28.57% std: (4.26%)

2. Following your implementation in the prior question, you decide that it makes more sense to simplify the model with two layers. However, you still need to identify the appropriate set of hyperparameters for your neural network. Implement a neural network model that allows you to configure the number of neurons and the activation functions in the network's hidden layers. Use Scikit-Learn's GridSearchCV function to identify the optimal hyperparameters. [15pts]

Best: 0.809524 using {'activation_func': 'relu', 'neurons': 30}
0.561905 (0.203818) with: {'activation_func': 'linear', 'neurons': 1}
0.657143 (0.163299) with: {'activation_func': 'linear', 'neurons': 2}
0.780952 (0.117417) with: {'activation_func': 'linear', 'neurons': 5}
0.504762 (0.211677) with: {'activation_func': 'linear', 'neurons': 10}
0.476190 (0.110246) with: {'activation_func': 'linear', 'neurons': 15}
0.685714 (0.209956) with: {'activation_func': 'linear', 'neurons': 20}
0.628571 (0.209956) with: {'activation_func': 'linear', 'neurons': 25}
0.438095 (0.158794) with: {'activation_func': 'linear', 'neurons': 30}
0.390476 (0.107750) with: {'activation_func': 'sigmoid', 'neurons': 1}
0.409524 (0.155329) with: {'activation_func': 'sigmoid', 'neurons': 2}
0.552381 (0.210388) with: {'activation_func': 'sigmoid', 'neurons': 5}
0.419048 (0.168762) with: {'activation_func': 'sigmoid', 'neurons': 10}
0.457143 (0.141902) with: {'activation_func': 'sigmoid', 'neurons': 15}
0.304762 (0.013469) with: {'activation_func': 'sigmoid', 'neurons': 20}
0.314286 (0.000000) with: {'activation_func': 'sigmoid', 'neurons': 25}
0.628571 (0.245781) with: {'activation_func': 'sigmoid', 'neurons': 30}
0.561905 (0.176640) with: {'activation_func': 'tanh', 'neurons': 1}
0.523810 (0.203818) with: {'activation_func': 'tanh', 'neurons': 2}
0.800000 (0.101686) with: {'activation_func': 'tanh', 'neurons': 5}
0.571429 (0.163299) with: {'activation_func': 'tanh', 'neurons': 10}
0.514286 (0.207348) with: {'activation_func': 'tanh', 'neurons': 15}
0.609524 (0.097124) with: {'activation_func': 'tanh', 'neurons': 20}
0.457143 (0.123443) with: {'activation_func': 'tanh', 'neurons': 25}
0.628571 (0.207348) with: {'activation_func': 'tanh', 'neurons': 30}
0.457143 (0.163299) with: {'activation_func': 'relu', 'neurons': 1}
0.571429 (0.176126) with: {'activation_func': 'relu', 'neurons': 2}
0.514286 (0.145686) with: {'activation_func': 'relu', 'neurons': 5}
0.533333 (0.175093) with: {'activation_func': 'relu', 'neurons': 10}
0.628571 (0.222539) with: {'activation_func': 'relu', 'neurons': 15}
0.580952 (0.088320) with: {'activation_func': 'relu', 'neurons': 20}
0.304762 (0.035635) with: {'activation_func': 'relu', 'neurons': 25}
0.809524 (0.117417) with: {'activation_func': 'relu', 'neurons': 30}

(a) Below, include a screenshot of your model's implementation that clearly shows your Sequential() Keras model and its ability to configure the number of neurons and the activation functions in both hidden layers. [5pts]

```python
def DynamicModel(neurons=1, activation_func='sigmoid'):
    """ A sequential Keras model that has an input layer, one
        hidden layer with a dymanic number of units, and an output layer."""
    model = Sequential()
    model.add(Dense(4, input_dim=4, activation='sigmoid', name='layer_1'))
    model.add(Dense(4, activation='sigmoid', name='layer_2'))
    model.add(Dense(3, activation='sigmoid', name='output_layer'))

    # Don't change this!
    model.compile(loss="categorical_crossentropy",
                  optimizer="adam",
                  metrics=['accuracy'])
    return model


# Evaluation + HyperParameter Search
# Below, we build KerasClassifiers using our model definitions.
model = KerasClassifier(
    build_fn=DynamicModel,
    epochs=200,
    batch_size=20,
    verbose=0)
```

(b) State the optimal number of neurons and choice of activation function for both layers as observed via GridSearchCV. [5pts]

The optimal number of 'neurons': 30 with the 'activation_func': 'relu' .

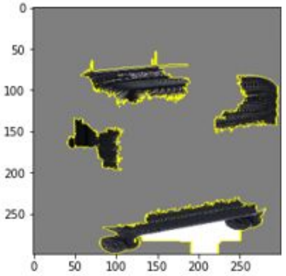
(c) Re-run the GridSearchCV technique on your neural network again, and you will find that your output may suggest a different set of hyperparameters perform best. With this variability in mind, what steps could you take with GridSearchCV to know that you've truly reached the optimal set of hyperparameters? [5pts]

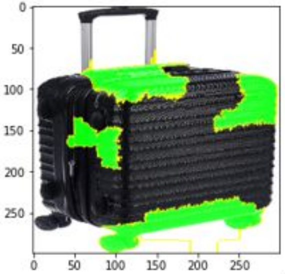
I have re-ran the GridSearchCV technique and found out that it somehow suggests different hyperparameters for my model and having that in mind and to tackle this thing, I suggest that we should increase the number of epochs for the search to find the optimal hyperparameters for us and don't settle or get stuck at local minimum situation.

3: Interpreting Neural Networks

1. **Top five predictions and probs**

('n04265275', 'space_heater', 0.62474257)
('n04041544', 'radio', 0.042016856)
('n04040759', 'radiator', 0.041702427)
('n04442312', 'toaster', 0.019159246)
('n04111531', 'rotisserie', 0.016038848)

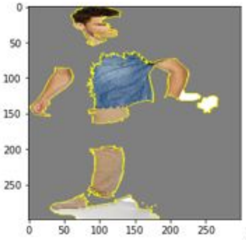


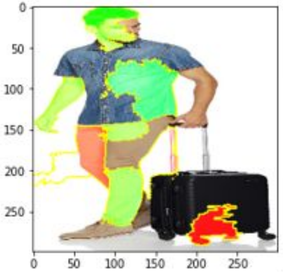


According to the boundaries area the classification is not well defined.

2. Top five predictions and probs

('n03594734', 'jean', 0.12457314)
('n02963159', 'cardigan', 0.090886645)
('n04517823', 'vacuum', 0.07377569)
('n02769748', 'backpack', 0.031545497)
('n03980874', 'poncho', 0.011480665)

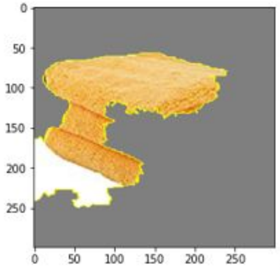


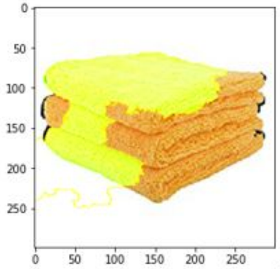


According to the boundaries area the classification is well defined.

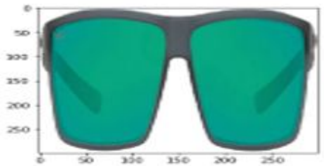
3. Top 5 predictions and probs

('n02808304', 'bath_towel', 0.9750167)
('n04344873', 'studio_couch', 0.0030538419)
('n03223299', 'doormat', 0.0014694714)
('n04599235', 'wool', 0.0011654152)
('n04209239', 'shower_curtain', 0.0011083777)



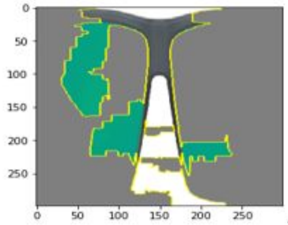


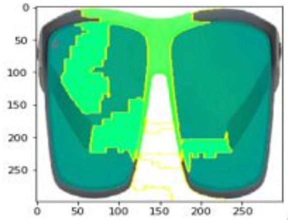
According to the boundaries area the classification is well defined.



4. Top five predictions and probs

('n04355933', 'sunglass', 0.86909425)
('n04356056', 'sunglasses', 0.12334232)
('n04357314', 'sunscreen', 0.0001756017)
('n03680355', 'Loafer', 0.00012325066)
('n03793489', 'mouse', 9.077392e-05)

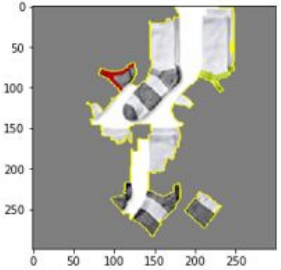


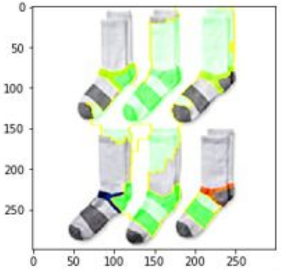


According to the boundaries area the classification is well defined.

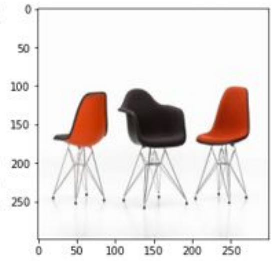
5. Top five predictions and probs

('n03197337', 'digital_watch', 0.51810086)
('n04579432', 'whistle', 0.1383834)
('n02966687', "carpenter's_kit", 0.036015928)
('n04372370', 'switch', 0.026099157)
('n04019541', 'puck', 0.02062653)



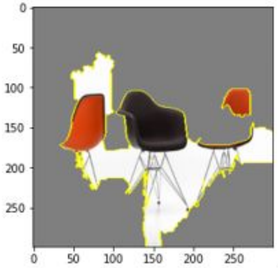


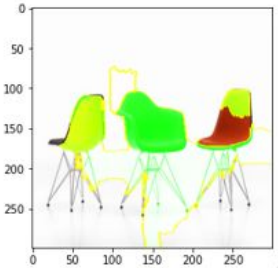
According to the boundaries area the classification is not well defined.



6. Top five predictions and probs

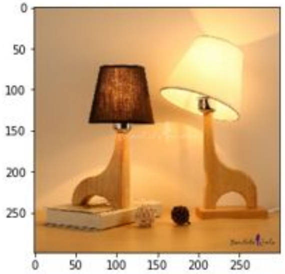
('n04099969', 'rocking_chair', 0.76237077)
('n03376595', 'folding_chair', 0.073354386)
('n03201208', 'dining_table', 0.0040207515)
('n04344873', 'studio_couch', 0.0030291225)
('n03047690', 'clog', 0.0029680138)



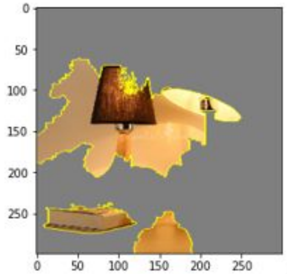


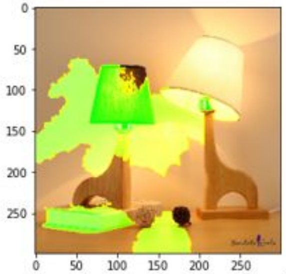
According to the boundaries area the classification is well defined.

7. Top five predictions and prob

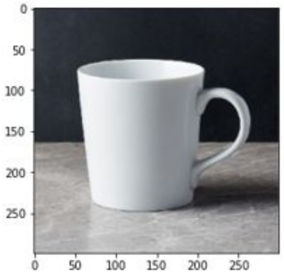


('n04380533', 'table_lamp', 0.81699556)
('n03637318', 'lampshade', 0.15963711)
('n04286575', 'spotlight', 0.00041560034)
('n02870880', 'bookcase', 0.00021121703)
('n03196217', 'digital_clock', 0.00016094094)



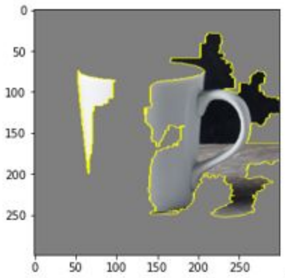


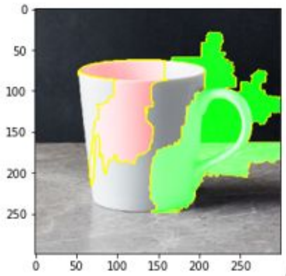
According to the boundaries area the classification is well defined.



8. **Top five predictions and prob**

('n03063599', 'coffee_mug', 0.55225056)
('n07930864', 'cup', 0.39505714)
('n03950228', 'pitcher', 0.020055579)
('n03733805', 'measuring_cup', 0.004896856)
('n04560804', 'water_jug', 0.0038265265)

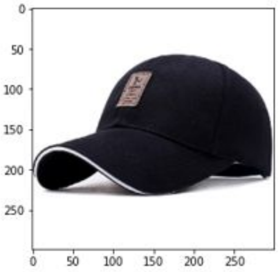


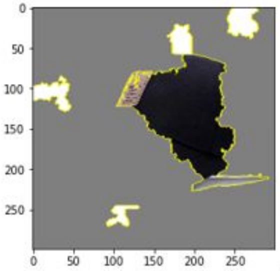


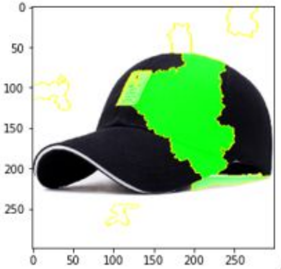
According to the boundaries area the classification is well defined.

9. *Top five predictions and probs*

('n02802426', 'basketball', 0.1819118)
('n03127747', 'crash_helmet', 0.08874585)
('n09835506', 'ballplayer', 0.041522685)
('n02807133', 'bathing_cap', 0.037733246)
('n03803284', 'muzzle', 0.026507074)



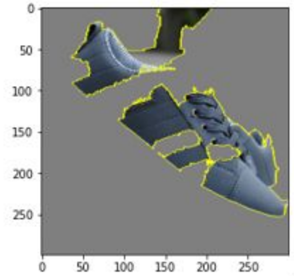


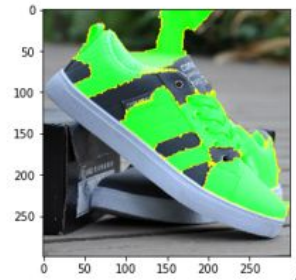


According to the boundaries area the classification is not well defined.

10. Top five predictions and probs

('n04120489',
'running_shoe',0.94716185)
('n04133789', 'sandal', 0.0032043813)
('n04200800', 'shoe_shop',0.0010591904)
('n03047690', 'clog', 0.0010585441)
('n04254777', 'sock', 0.00090397295)





According to the boundaries area the classification is well defined.

3. Based on your observations in Question 1 and 2, is the Inception model interpretable? Are there types of images that seem to fail or succeed? Rationalize your answer with Miller's definition of "Good Explanations".

Based on the observations in question 1 and 2 for ten observations , we can say that the inception model is pretty much interpretable.
There are different images of the different products found on amazon that we checked for the lime model and there we have found that about 1/3rd of the images seem to fail while the other images seem to succeed .For Ex:

The image 7 of the lamp is predicted accurately but contradict to that image 5 of socks is not predicted right.