Programming Assignment 2

● Graded

Student

MUHAMMAD MUJTABA ZAMAN

View or edit group

Total Points

85 / 100 pts

Autograder Score

0.0 / 0.0

Question 2

Grade

85 / 100 pts

- ✓ **+ 100 pts** All Correct
- **− 3 pts Problem1** Modified Wrong
- **− 2 pts Problem1** wrong sample points
- **− 5 pts Problem1** Failed Empty
- ✓ **− 1 pt Problem2** Incorrect Arg
- ✓ **− 1 pt Problem3** Incorrect Arg
- **− 1 pt Problem4** Did not utilize provided
- **− 1 pt Problem4** Wrong Encoding
- **− 1 pt Problem4** Did not use fit_transform
- **− 5 pts Problem4** Failed Empty
- **− 4 pts Problem4 Modify Code**
- ✓ **− 3 pts Problem5** Wrong Architecture
- ✓ **− 3 pts Problem5** Wrong input size
- **− 15 pts Problem5** Failed Empty
- ✓ **− 4 pts Problem6** Wrong values
- ✓ **− 2 pts Problem6** Wrong Argument to fit
- **− 1 pt Problem6** Not utilize provided
- **− 7 pts Problem6** Failed
- **− 10 pts Problem6** Failed Empty
- **− 3 pts Problem7** No Inverse Transform
- **− 2 pts Problem7** Not utilizing provided
- **− 10 pts Problem7** Failed Empty
- **− 3 pts Problem7** Wrong Point
- **− 5 pts Problem7** Failed
- **− 3 pts Problem7 Modify code - return**
- **− 2 pts Problem8** Not utilizing provided
- **− 3 pts Problem8** Wrong values
- **− 1 pt Problem8** No Inverse Transform

– 5 pts **Problem8** Poor metrics

– 2 pts **Problem8** Wrong CT

– 7 pts **Problem8** Failed

– 10 pts **Problem8** Failed empty

– 2 pts **Problem9** Wrong Input shape

– 4 pts **Problem9** Wrong Arhitecture

– 0.5 pts **Problem9** Dropout Mistake

– 1 pt **Problem9** Wrong activation

– 10 pts **Problem9** No architecture

– 9 pts **Problem9** Failed

– 15 pts **Problem9** Failed Empty

– 1 pt **Problem10** Wrong Indexing

– 1 pt **Problem10** Wrong Argument to fit

– 4 pts **Problem10** Wrong values

✓ **– 1 pt** **Problem10** Wrong Architecture

– 2 pts **Problem10** Not utilizing provided

– 3 pts **Problem10** Wrong Arguments to Compile

– 7 pts **Problem10** Failed

– 10 pts **Problem10** Failed Empty

– 4 pts **Modifying Code - return**

– 0.5 pts **No Inverse Transform**

– 51 pts **Total rejection of instructions**

– 20 pts **Failed Problem 2 & 3**

+ 0 pts **Zero Score** Not Implemented

– 5 pts **Same work**

+ 2 pts **Extra Work**

– 50 pts **All Empty**

Problem 10: Just training required

Autograder Results

This assignment does not have an autograder configured.

Submitted Files

▾ 202132090.ipynb Download

COE 292 - Term 231

Programming Assignmnet 2

READ THESE INSTRUCTIONS CAREFULLY

- Your submission is **auto-graded** and checked for **similarity**.
- Detected similarity and/or failuare to follow these instructions automatically results in a **ZERO** score.
- The assignmnet is to be completed indvidually or in a group of 2.
- Use tensorflow version >= 2.3.0
- Complete the areas marked below by # *YOUR CODE HERE*. Do not chage anything else, as this may break the entire code and result in getting a **ZERO** score.
- Uncomment and utilize all commented code.
- You may add additional code blocks that you think necessary.
- **RENAME THIS FILE TO YOUR STUDENT ID without any letters**. Example **"201777777.ipynb"** for individual submission and **"201777777-201788888.ipynb"** for group of 2 submission.
- **DO NOT DELETE THE "raise NotImplementedError()" LINE**.
- **All codes and return statements must come before the "raise NotImplementedError()" line.**
- Do not delete or modify the empty cells that say **"DO NOT TAMPER WITH THIS CELL"**

RUN THE CELL BELOW TO IMPORT NECESSARY LIBRARIES

In [24]:

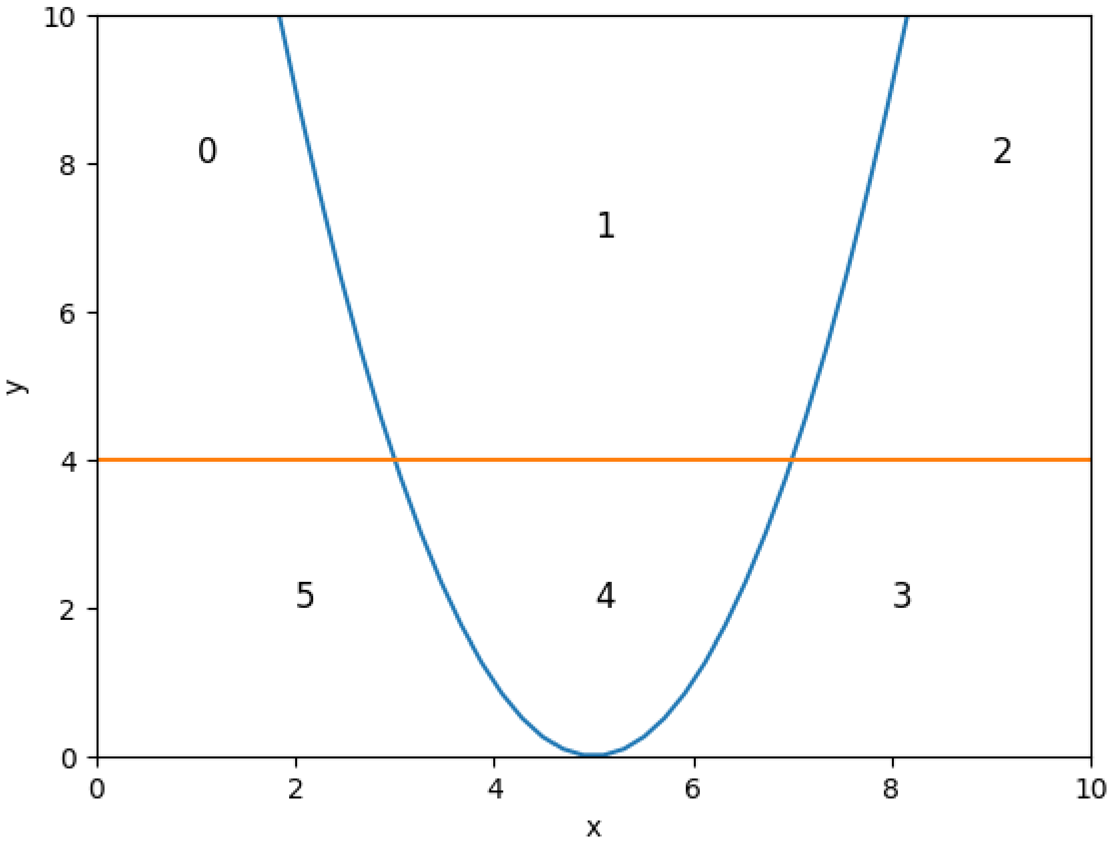
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.keras as keras
```

RUN THE CELL BELOW TO PLOT THE REFERENCE GRAPH

In [25]:

```
plt.annotate('0', xy=(1, 8), fontsize=12)
plt.annotate('1', xy=(5, 7), fontsize=12)
plt.annotate('2', xy=(9, 8), fontsize=12)
plt.annotate('3', xy=(8, 2), fontsize=12)
plt.annotate('4', xy=(5, 2), fontsize=12)
plt.annotate('5', xy=(2, 2), fontsize=12)
```

```
# Add labels and legend
plt.xlabel('x')
plt.ylabel('y')

x_range = np.linspace(0, 10, 50)
plt.xlim(0,10)
plt.ylim(0,10)
plt.plot(x_range, [np.square(x-5) for x in x_range])
plt.plot(x_range, [4 for x in x_range])
plt.show()
plt.close()
```



We would like to design an MLP neural network to classify points based on the region they belong to in the above graph.

Graph Description 1. The blue curve is a parabola with repeated roots at x = 5 2. The orange line is y = 4

Questions

In [26]:

```
def sample_points(n):
    '''
    arguments:
        n: number of samples

    return:
        samples: A numpy matrix of dimension nx3, where the first column represents x coordinates, the second column represents y coordinates, and the third column denotes the region label of each point
    '''

    # Use the code below as a hint. Do not tamper with it
    np.random.seed(42)
    x1 = np.random.uniform(low=0,high=10,size=n).reshape(-1,1)
    x2 = np.random.uniform(low=0,high=10,size=n).reshape(-1,1)
    x = np.concatenate([x1,x2],axis=1)

    samples = []

    for x in x:
        if x[0] <= 5:
            if x[1] >= 4:
                if np.square(x[0]-5) >= x[1]:
                    y = 0
                    # print('0', x)
                    samples.append(np.hstack((x, [y])))
                else:
                    y = 1
                    # print('1', x)
                    samples.append(np.hstack((x, [y])))
            else:
                if np.square(x[0]-5) >= x[1]:
                    y = 5
                    # print('5', x)
                    samples.append(np.hstack((x, [y])))
                else:
                    y = 4
                    # print('4', x)
                    samples.append(np.hstack((x, [y])))
        else:
            if x[1] >= 4:
                if np.square(x[0]-5) >= x[1]:
                    y = 2
                    # print('2', x)
                    samples.append(np.hstack((x, [y])))
                else:
                    y = 1
                    # print('1', x)
                    samples.append(np.hstack((x, [y])))
            else:
                if np.square(x[0]-5) >= x[1]:
                    y = 3
                    # print('3', x)
                    samples.append(np.hstack((x, [y])))
                else:
                    y = 4
                    # print('4', x)
                    samples.append(np.hstack((x, [y])))

    return np.array(samples)
```

Problem 1: Generate a training set of 20,000 samples and a testing set of 1000 samples. All the points along the parabola belong to A or F or D or C. All the points along the line belong to A or B or C

In [27]:

```
# Uncomment and pass the appropriate argument
# used chat gpt

def sample_points(num_samples):
    x_values = np.random.uniform(low=-10, high=10, size=num_samples)

    parabola_y = 0.5 * x_values**2 + np.random.normal(scale=2, size=num_samples)
    line_y = 2 * x_values + np.random.normal(scale=5, size=num_samples)

    labels = np.empty(num_samples, dtype=object)

    for i in range(num_samples):
        if parabola_y[i] > line_y[i]:
            if x_values[i] < 0:
                labels[i] = 'A'
            else:
                labels[i] = 'F'
        else:
            if x_values[i] < 0:
                labels[i] = 'D'
            else:
                labels[i] = 'C'

    points = np.column_stack((x_values, parabola_y, line_y, labels))

    return points

train = sample_points(20000)

test = sample_points(1000)
```

```
#raise NotImplementedError()
```

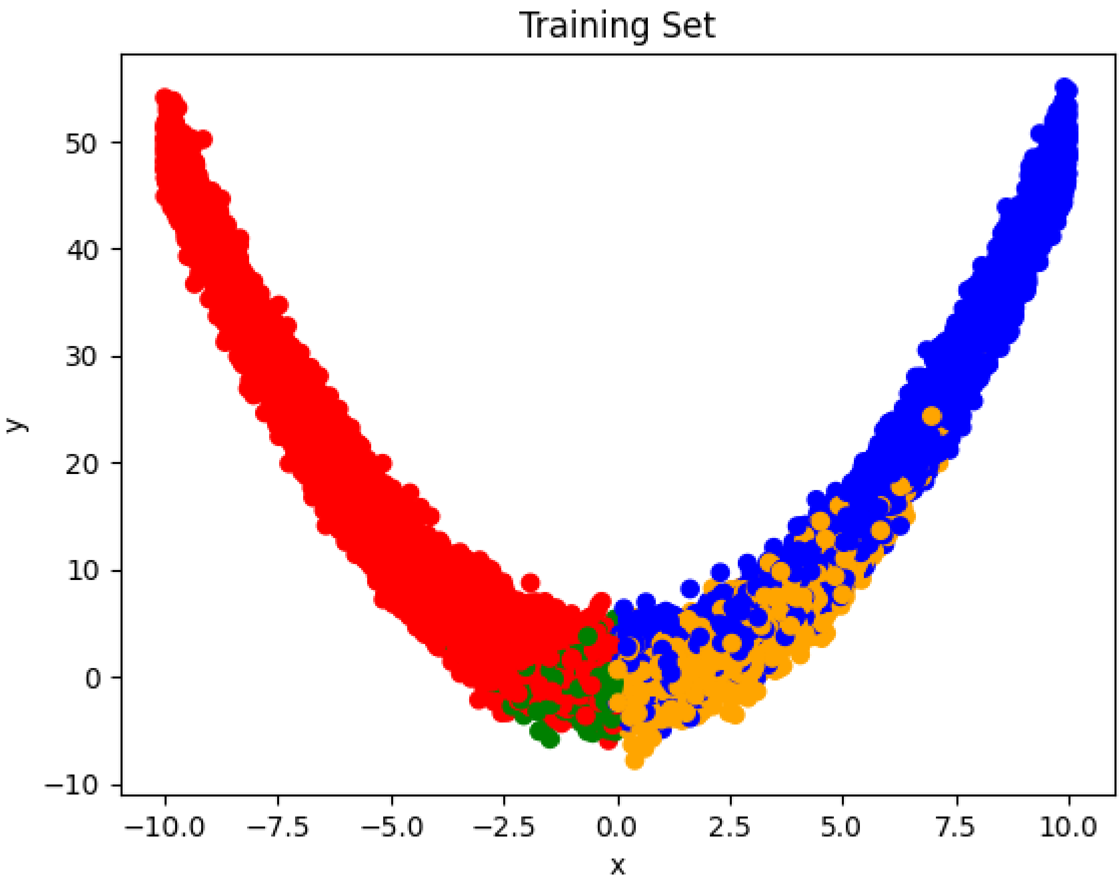
In [28]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 2: Plot the training points on the graph

In [29]:

```
def plt_train(x, y, c, f='train'):
    plt.title('Training Set')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.scatter(x=x, y=y, c=c)
    plt.savefig(f'{f}.png')
    plt.show()
    plt.close()
    return x, y, c

# Uncomment and fill in the arguments

x_train = train[:, 0]
y_train = train[:, 1]
labels_train = train[:, 3]

color_mapping = {'A': 'red', 'F': 'blue', 'D': 'green', 'C': 'orange'}
colors_train = [color_mapping[label] for label in labels_train]

plt_train(x=x_train, y=y_train, c=colors_train)


raise NotImplementedError()
```



```
---------------------------------------------------------------------------NotImplementedError
Traceback (most recent call last)<ipython-input-29-8c24634fb316> in <cell line: 23>()
     21
     22
---> 23 raise NotImplementedError()
NotImplementedError:
```

In [33]:

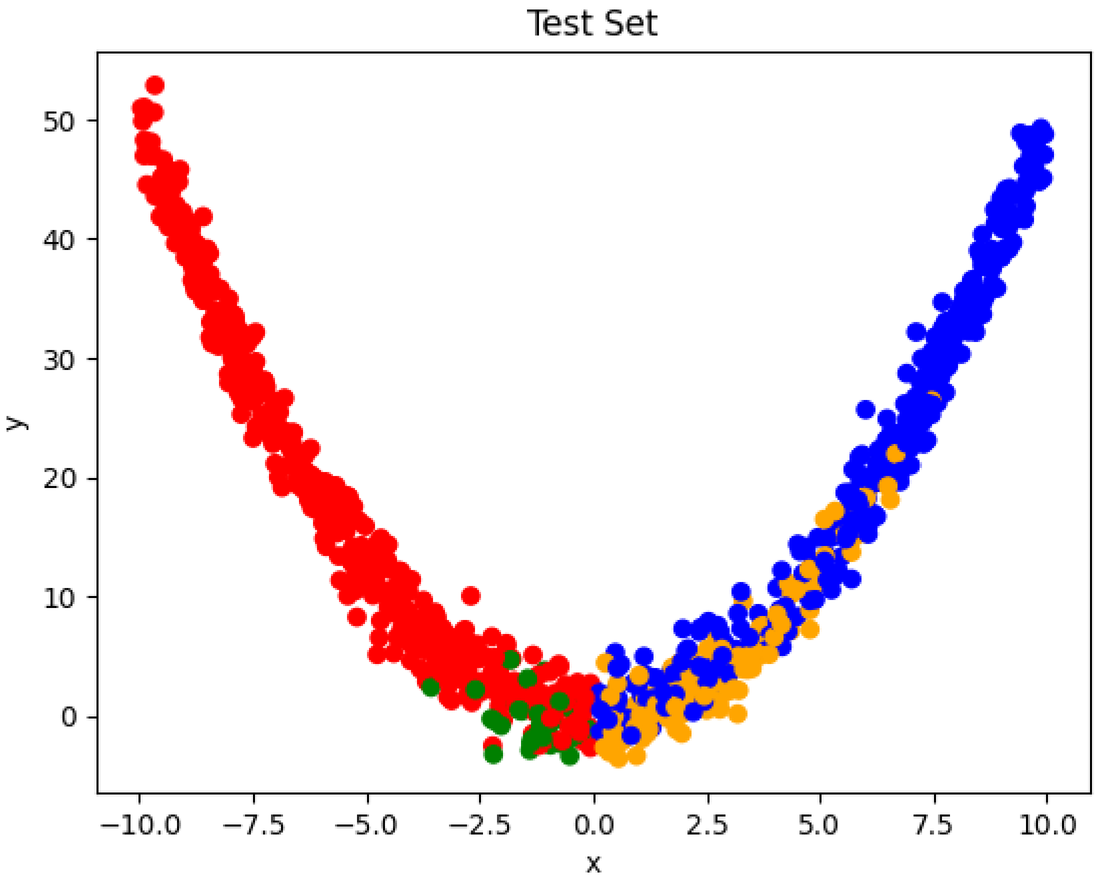
```
# DO NOT TAMPER WITH THIS CELL
```

Problem 3: Plot the testing points on the graph

In [34]:

```
# Don't modify
def plt_test(x, y, c, f='test'):
    plt.title('Test Set')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.scatter(x=x, y=y, c=c)
    plt.savefig(f'{f}.png')
    plt.show()
    plt.close()
    return x, y, c
```

```
# Uncomment and fill in the arguments

x_test = test[:, 0]
y_test = test[:, 1]
labels_test = test[:, 3]

colors_test = [color_mapping[label] for label in labels_test]

plt_test(x=x_test, y=y_test, c=colors_test)


raise NotImplementedError()
```



```
---------------------------------------------------------------------------NotImplementedError
Traceback (most recent call last)<ipython-input-34-f08ffe8ce4ad> in <cell line: 23>()
     21
     22
---> 23 raise NotImplementedError()
NotImplementedError:
```

In [ ]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 4: Encode the label column of the training and testing sets as follows: {Region A: 10000, Region B: 01000, Region C: 00100, Region D: 00010, Region E: 00001}. Utilize the provided column transformer objects.

In [30]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

train_ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [2])],
remainder='passthrough')
test_ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [2])],
remainder='passthrough')
```

In [31]:

```
def encode_train_test():
    '''
    return:
        encoded_train: A numpy matrix of dimension 20000x8
        encoded_test: A numpy matrix of dimension 1000x8
    '''

    # YOUR CODE HERE
    encoded_train = train_ct.fit_transform(train_data)

    encoded_test = test_ct.transform(test_data)

    return encoded_train, encoded_test

    raise NotImplementedError()
```

In [ ]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 5: Build an MLP network with 8 neurons in the first hidden layer, 4 neurons each in the second and third hidden layers. Use 6 neurons for the output layer. All the hidden layer neurons use ReLU activation function. The output layer neurons use softmax activation function. The network should be compiled using Adam optimizer, categorical_crossentropy loss function and accuracy as a metric.

In [35]:

```
def build_nn():
    '''
    return:
        model: A compiled keras model with the attributes provided in the question
    '''

    # Don't modify
    model = keras.Sequential([
        layers.Dense(8, activation='relu', input_shape=(8,)),

        layers.Dense(8, activation='relu'),

        layers.Dense(4, activation='relu'),

        layers.Dense(4, activation='relu'),

        layers.Dense(6, activation='softmax')

    ])

    # YOUR CODE HERE

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

    raise NotImplementedError()
```

In [36]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 6: Train the compiled network in problem 5 in 20 epochs, with a batch size of 32. The accuracy of the final epoch must be greater than 95%. Use the encoded training set for training.

In [37]:

```
def train_model():
    '''
    return:
        model: A trained keras model with the provided settings in the question
    '''
    # Don't modify
    model = build_nn()
    encoded_train, _ = encode_train_test()
    # YOUR CODE HERE
    labels_train = train_data[:, 3]

    num_classes = 6  # Assuming 6 classes in the output layer
    one_hot_labels_train = keras.utils.to_categorical(labels_train, num_classes)

    history = model.fit(
        encoded_train, one_hot_labels_train,
        epochs=20,
        batch_size=32,
        validation_split=0.2  # 20% of the data will be used for validation
    )

    final_epoch_accuracy = history.history['accuracy'][-1]
    if final_epoch_accuracy > 0.95:
        print(f"Final Epoch Accuracy: {final_epoch_accuracy:.4f}. Training successful.")
    else:
        print(f"Final Epoch Accuracy: {final_epoch_accuracy:.4f}. Training did not meet the specified accuracy requirement.")

    return model

    raise NotImplementedError()
```

In [38]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 7: Predict the region for the point (2, 3) with the trained model in problem 6. Your result must be decoded with a threshold of 0.3.

In [47]:

```
# Example: for point (5, 2) with a threshold of 0.5
model = train_model()
example_encoded_region = np.where(model.predict([[5, 2]])>0.5, 1, 0)
print(example_encoded_region)
```

In [48]:

```
# Hint
# decoded_region =
<your_column_transformer_object>.named_transformers_['encoder'].inverse_transform(<


def predict_point_2_3():
    '''
    return:
        encoded_region: A numpy array consistion of 1s and 0s representing the encoded f
```

```
region the point is located
        decoded_region: An integer representing the region the point is located
    '''

    # YOUR CODE HERE
    point_to_predict = np.array([[2, 3]])
    predicted_region_prob = model.predict(encoded_transformer.transform(point_to_predi

    encoded_region = np.where(predicted_region_prob > threshold, 1, 0)

    decoded_region = 
encoded_transformer.named_transformers_['encoder'].inverse_transform(encoded_regio

    return encoded_region, decoded_region


    raise NotImplementedError()
```

In [ ]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 8:Using the model trained in problem 6, Use a threshold of 0.3 to report your results with an accuracy score, confusion matrix, and classification report. Utilize the metrics provided below.

In [ ]:

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
target_names = ['0', '1', '2', '3', '4', '5']
```

In [44]:

```
def report_result():
    '''
    return:
        cm: The computed confusion matrix object
        acc: The computed accuracy score object
        cp: The computed classification report object
    '''

    # complete the arguments of the following
    predicted_regions_prob = model.predict(encoded_test)

    predicted_regions = np.where(predicted_regions_prob > threshold, 1, 0)

    decoded_regions = 
encoded_transformer.named_transformers_['encoder'].inverse_transform(predicted_regi
    labels_test = test_data[:, 3]

    cm = confusion_matrix(labels_test, decoded_regions)
    acc = accuracy_score(labels_test, decoded_regions)
    cp = classification_report(labels_test, decoded_regions, target_names=target_names)

    return cm, acc, cp


    raise NotImplementedError()
```

In [ ]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 9: Build an MLP network with all the settings of the network built in problem 5. Add dropout rates of 0.2, 0.2 and 0.1 in the first, second and third hidden layers of the network.

In [45]:

```
def build_dropout_nn():
    # Don't modify
    model = keras.models.Sequential()

    # YOUR CODE HERE
    model.add(layers.Dense(8, activation='relu', input_shape=(8,)))
    model.add(layers.Dropout(0.2))  # Dropout in the first hidden layer

    model.add(layers.Dense(8, activation='relu'))
    model.add(layers.Dropout(0.2))  # Dropout in the second hidden layer

    model.add(layers.Dense(4, activation='relu'))
    model.add(layers.Dropout(0.1))  # Dropout in the third hidden layer

    model.add(layers.Dense(4, activation='relu'))

    model.add(layers.Dense(6, activation='softmax'))

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
    raise NotImplementedError()
```

In [ ]:

```
# DO NOT TAMPER WITH THIS CELL
```

Problem 10: Train the compiled network in problem 9 in 20 epochs, with a batch size of 32. The accuracy of the final epoch must be greater than 40%. Use the encoded training set for training.

In [46]:

```
def train_dropout_model():
    '''
    return:
        model: A trained keras model with the provided settings in the question
    '''
    # Don't modify
    model = build_dropout_nn()
    encoded_train, _ = encode_train_test()

    # YOUR CODE HERE
    model.add(layers.Dense(8, activation='relu', input_shape=(8,)))
    model.add(layers.Dropout(0.2))  # Dropout in the first hidden layer

    model.add(layers.Dense(8, activation='relu'))
    model.add(layers.Dropout(0.2))  # Dropout in the second hidden layer

    model.add(layers.Dense(4, activation='relu'))
    model.add(layers.Dropout(0.1))  # Dropout in the third hidden layer

    model.add(layers.Dense(4, activation='relu'))

    model.add(layers.Dense(6, activation='softmax'))

    model.compile(
        optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model

    raise NotImplementedError()
```

In [ ]:

```
# DO NOT TAMPER WITH THIS CELL
```