Neural Networks - intro

Part 1 - XOR

1. Using the XOR dataset below, train (400 epochs) a neural network (NN) using 2, 3, 4, and 5 hidden layers (where each layer has only 2 neurons). For each n layers, store the resulting accuracy along with n. Plot the results to find what the optimal number of layers is.
2. Repeat the above with 3 neurons in each Hidden layers. How do these results compare to the 2 neuron layers?
3. Repeat the above with 4 neurons in each Hidden layers. How do these results compare to the 2 and 3 neuron layers?
4. Using the most optimal configuraion (n-layers, k-neurons per layer), compare how `tanh`, `sigmoid`, `softplus` and `relu` effect the loss after 400 epochs. Try other Activation functions as well (https://keras.io/activations/ (https://keras.io/activations/))
5. Again with the most optimal setup, try other optimizers (instead of `SGD`) and report on the loss score. (https://keras.io/optimizers/ (https://keras.io/optimizers/))

Part 2 - BYOD (Bring your own Dataset)

Using your own dataset, experiment and find the best Neural Network configuration. You may use any resource to improve results, just reference it.

While you may use any dataset, I'd prefer you didn't use the diabetes dataset used in the lesson.

https://stackoverflow.com/questions/34673164/how-to-train-and-tune-an-artificial-multilayer-perceptron-neural-network-using-k (https://stackoverflow.com/questions/34673164/how-to-train-and-tune-an-artificial-multilayer-perceptron-neural-network-using-k)

https://keras.io/ (https://keras.io/)

Part 1

1-A. generating XOR dataset

In [1]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD  #Stochastic Gradient Descent

import numpy as np
# fix random seed for reproducibility
np.random.seed(7)

import matplotlib.pyplot as plt
%matplotlib inline
```

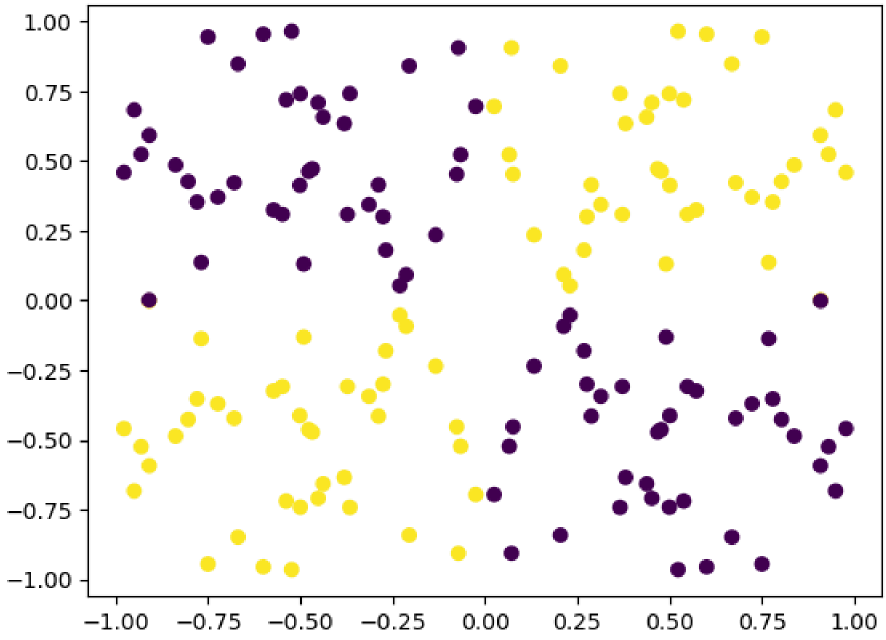
```
WARNING:tensorflow:From C:₩Users₩kcosm₩anaconda3₩Lib₩site-packages₩keras₩src₩losses.py:2976: The name t
f.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross
_entropy instead.
```

In [2]:
```
n = 40
xx = np.random.random((n,1))
yy = np.random.random((n,1))
```

In [3]:
```
X = np.array([np.array([xx,-xx,-xx,xx]),np.array([yy,-yy,yy,-yy])]).reshape(2,4*n).T
y = np.array([np.ones([2*n]),np.zeros([2*n])]).reshape(4*n)
```

In [4]:
```
plt.scatter(*zip(*X), c=y)
```

Out[4]: <matplotlib.collections.PathCollection at 0x16cc95640d0>



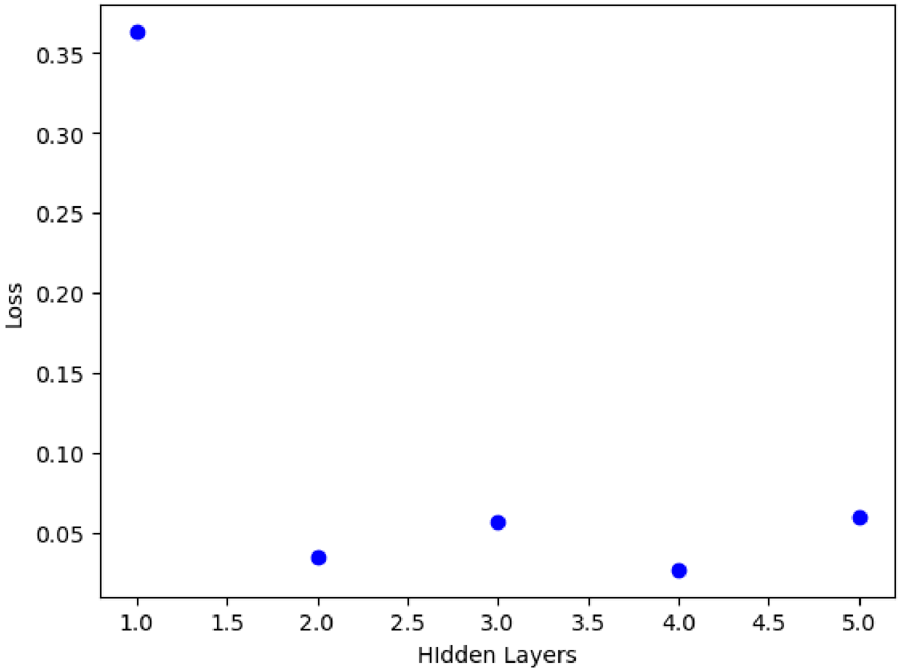
1-B. Train Neural Network with 2 neurons

In [5]:
```
num_layers = [1,2,3,4,5]

# set SGD for optimizer
sgd = SGD(learning_rate=0.1)
```

In [6]:
```
# define 'scores1_2' for scores 2-neuron model in assignment 1
scores1_2 = []

# define 'layer_size' to designate the number of neurons
layer_size = 2

for num_layer in num_layers:

    # define 'model' as sequential model
    model = Sequential()

    # add input layer (input dimension =2)
    model.add(Dense(2, input_dim=2, activation='tanh'))

    # add hiddel layers for the designated number of layers (num_layer)
    for _ in range(num_layer):
        model.add(Dense(layer_size, activation='tanh'))

    # add output layer
    # I set sigmoid function for activation, because the model did not fit well when using 'tanh' (loss va
    model.add(Dense(1, activation='sigmoid'))

    # complie and fit the model
    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, y, batch_size=1, epochs=400)

    # calculate score for each case and add scores
    score = model.evaluate(X, y)
    scores1_2.append(score)

    # print length of layers and scores to check if the model is well generated
    print(len(model.layers), scores1_2)
```

```
Epoch 392/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0604
Epoch 393/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0604
Epoch 394/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0604
Epoch 395/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0603
Epoch 396/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0603
Epoch 397/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0603
Epoch 398/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0603
Epoch 399/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0603
Epoch 400/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0603
5/5 [==============================] - 0s 3ms/step - loss: 0.0599
```

In [7]:
```
# Plot the result
plt.scatter(num_layers, scores1_2, c='b')
plt.xlabel('HIdden Layers')
plt.ylabel('Loss')
plt.show()

print(scores1_2)
```




```
[0.3633359968662262, 0.03465220332145691, 0.0565946027636528, 0.02665257453918457, 0.0599004924297332 76]
```

Result of 1-B:

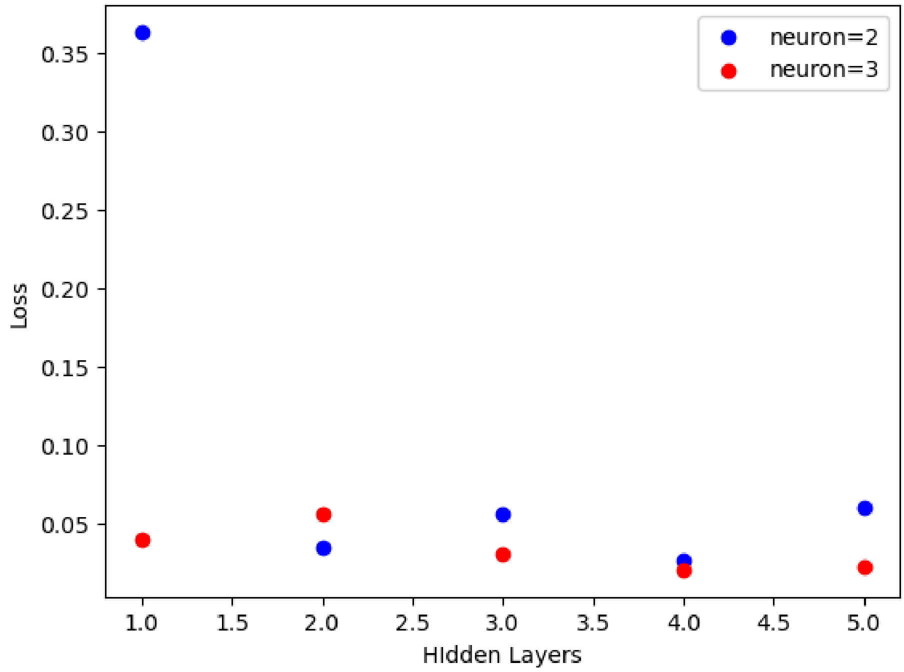
- 4 hidden layers are optimal for 2-neuron models

1-C. Train Neural Network with 3 neurons

In [8]:
```python
# Repeat the same process as 2 neurons. Only change layer_size from 2 to 3

# define 'scores1_3' for scores of 3-neuron model in assignment 1
scores1_3 = []
layer_size = 3

for num_layer in num_layers:

    model = Sequential()
    model.add(Dense(layer_size, input_dim=2, activation='tanh'))

    for _ in range(num_layer):
        model.add(Dense(layer_size, activation='tanh'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, y, batch_size=1, epochs=400)

    score = model.evaluate(X, y)
    scores1_3.append(score)
    print(len(model.layers), scores1_3)
```

```
160/160 [==============================] - 0s 2ms/step - loss: 0.0342
Epoch 393/400
160/160 [==============================] - 0s 3ms/step - loss: 0.0338
Epoch 394/400
160/160 [==============================] - 0s 3ms/step - loss: 0.0286
Epoch 395/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0321
Epoch 396/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0287
Epoch 397/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0309
Epoch 398/400
160/160 [==============================] - 0s 3ms/step - loss: 0.0301
Epoch 399/400
160/160 [==============================] - 0s 3ms/step - loss: 0.0306
Epoch 400/400
160/160 [==============================] - 0s 3ms/step - loss: 0.0269
5/5 [==============================] - 0s 3ms/step - loss: 0.0225
7 [0.03947985172271285, 0.05617266148328781, 0.030184149742126465, 0.020512768998742104, 0.022529361
```

In [9]:
```
# Plot the result
plt.scatter(num_layers, scores1_2, c='b')
plt.scatter(num_layers, scores1_3, c='r')
plt.legend(['neuron=2', 'neuron=3'])
plt.xlabel('HIdden Layers')
plt.ylabel('Loss')
plt.show()

print(scores1_2)
print(scores1_3)
```




```
[0.3633359968662262, 0.03465220332145691, 0.05659460276365 28, 0.02665257453918457, 0.0599004924297332 76]
[0.039479851722717285, 0.056172661483287 81, 0.030184149742126465, 0.020512768998742104, 0.02252936176955
7]
```

Result of 1-C:

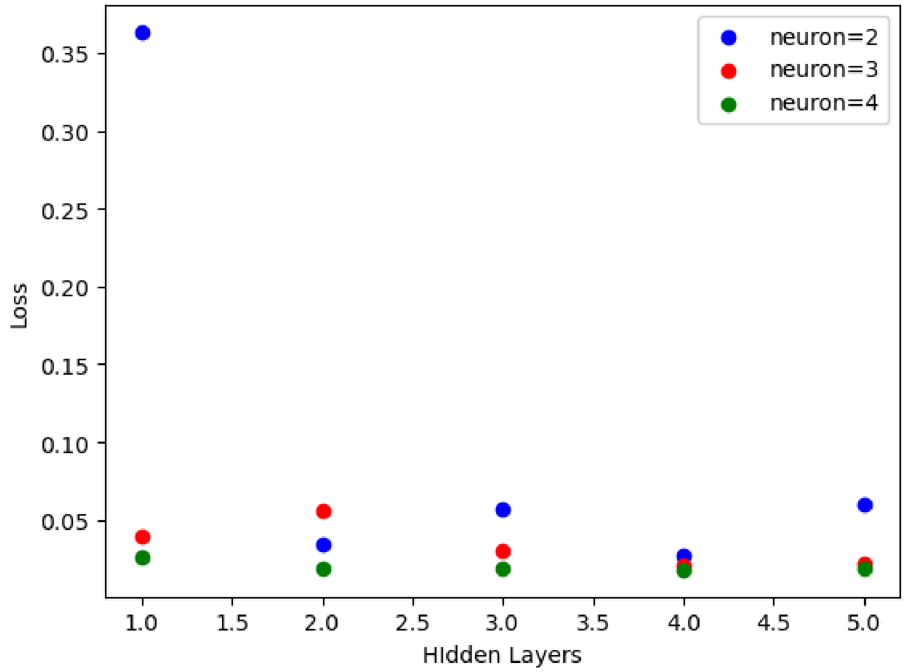
- 4 hidden layers are optimal for 3-neuron models
- 3-neuron models outperform 2-neuron models in all hidden layers except 2 hiddel layers

1-D. Train Neural Network with 4 neurons

In [10]:
```
# Repeat the same process as 2 and 3 neurons. Only change layer_size into 4

# define 'scores1_4' for scores of 4-neuron model in assignment 1
scores1_4 = []
layer_size = 4

for num_layer in num_layers:

    model = Sequential()
    model.add(Dense(layer_size, input_dim=2, activation='tanh'))

    for _ in range(num_layer):
        model.add(Dense(layer_size, activation='tanh'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, y, batch_size=1, epochs=400)

    score = model.evaluate(X, y)
    scores1_4.append(score)
    print(len(model.layers), scores1_4)
```

```
160/160 [==============================] - 0s 2ms/step - loss: 0.0215
Epoch 393/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0216
Epoch 394/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0215
Epoch 395/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0214
Epoch 396/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0216
Epoch 397/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0217
Epoch 398/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0216
Epoch 399/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0213
Epoch 400/400
160/160 [==============================] - 0s 2ms/step - loss: 0.0214
5/5 [==============================] - 0s 3ms/step - loss: 0.0184
7 [0.026274342089891434, 0.018895685672760 01, 0.01870402693748474, 0.01799273118376732, 0.01843993738
```

In [11]:
```
# Plot the result
plt.scatter(num_layers, scores1_2, c='b')
plt.scatter(num_layers, scores1_3, c='r')
plt.scatter(num_layers, scores1_4, c='g')
plt.legend(['neuron=2', 'neuron=3', 'neuron=4'])
plt.xlabel('HIdden Layers')
plt.ylabel('Loss')
plt.show()

print(scores1_2)
print(scores1_3)
print(scores1_4)
```




```
[0.3633359968662262, 0.03465220332145691, 0.0565946027636528, 0.02665257453918457, 0.059900492429733276]
[0.039479851722717285, 0.05617266148328781, 0.030184149742126465, 0.020512768998742104, 0.02252936176955
7]
[0.026274342089891434, 0.01889568567276001, 0.01870402693748474, 0.01799273118376732, 0.0184399373829364
78]
```

Result of 1-D

- 4 hidden layers are optimal for 4-neuron models (Loss = 0.01799..)
- 4-neuron models outperform 2-neuron and 3-neuron models in all hidden layers

Overall Result of 1-B to 1-D : 4 neuron-model with 4 hidden layers is the optimal configuration

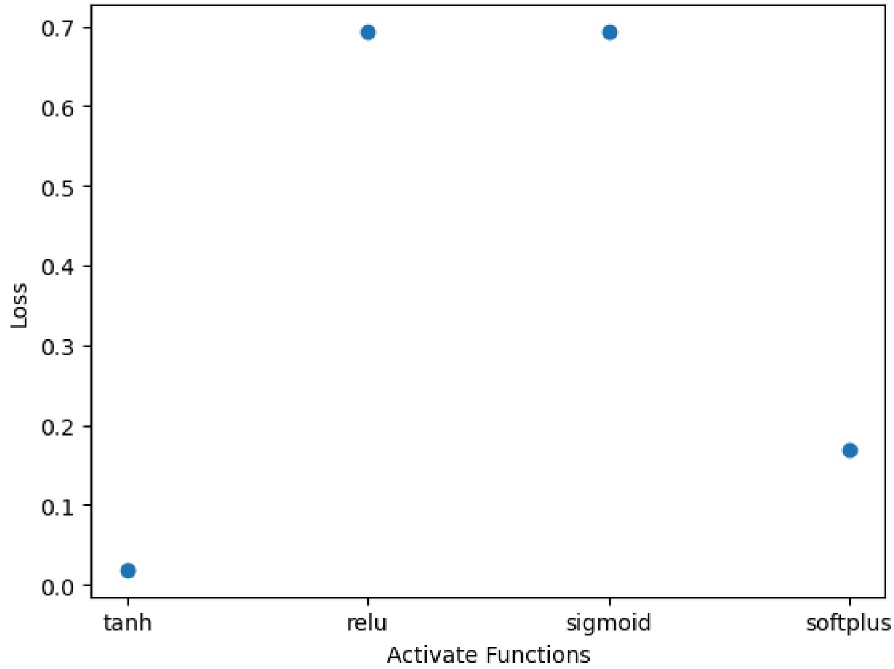
- 1 batch size, 400 epoch
- tanh function for input and hidden layers
- sigmoid function for output layer

1-E. Compare activate functions

In [12]:
```
# define four activate functions as 'activate_functions'
activate_functions = ['tanh', 'relu', 'sigmoid', 'softplus']

# define 'scores1_af' for scores of activate functions in assignment 1
scores1_af = []

# use 'for' loop to acquire score for each function
for af in activate_functions:

    # generate the optimal sequential model(4-neuron, 4 hidden layers) for each activate function.
    model = Sequential()
    model.add(Dense(4, input_dim=2, activation=af))
    model.add(Dense(4, activation=af))
    model.add(Dense(4, activation=af))
    model.add(Dense(4, activation=af))
    model.add(Dense(4, activation=af))
    # keep activation function of the output layer as 'sigmoid'
    # because the model did not fit well when using other activation functions (loss soars above 7.0)
    model.add(Dense(1, activation='sigmoid'))

    # compile and fit
    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, y, batch_size=1, epochs=400)

    score = model.evaluate(X, y)
    scores1_af.append(score)
    print(len(model.layers), scores1_af)
```

```
Epoch 392/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2213
Epoch 393/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2268
Epoch 394/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2416
Epoch 395/400
160/160 [==============================] - 0s 3ms/step - loss: 0.2344
Epoch 396/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2219
Epoch 397/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2130
Epoch 398/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2167
Epoch 399/400
160/160 [==============================] - 0s 2ms/step - loss: 0.2072
Epoch 400/400
160/160 [==============================] - 0s 2ms/step - loss: 0.1939
5/5 [==============================] - 0s 3ms/step - loss: 0.1683
6 [0.018594099208712578, 0.6931766271591187, 0.6931482553482056, 0.1683395802974701]
```

In [13]:
```
#Plot the result
plt.scatter(activate_functions, scores1_af)
plt.xlabel('Activate Functions')
plt.ylabel('Loss')
plt.show()

print(scores1_af)
```




```
[0.018594099208712578, 0.6931766271591187, 0.6931482553482056, 0.1683395802974701]
```

Result of 1-E:

- 'tanh' function had the best performance of the four activation functions (Loss = 0.01859..)

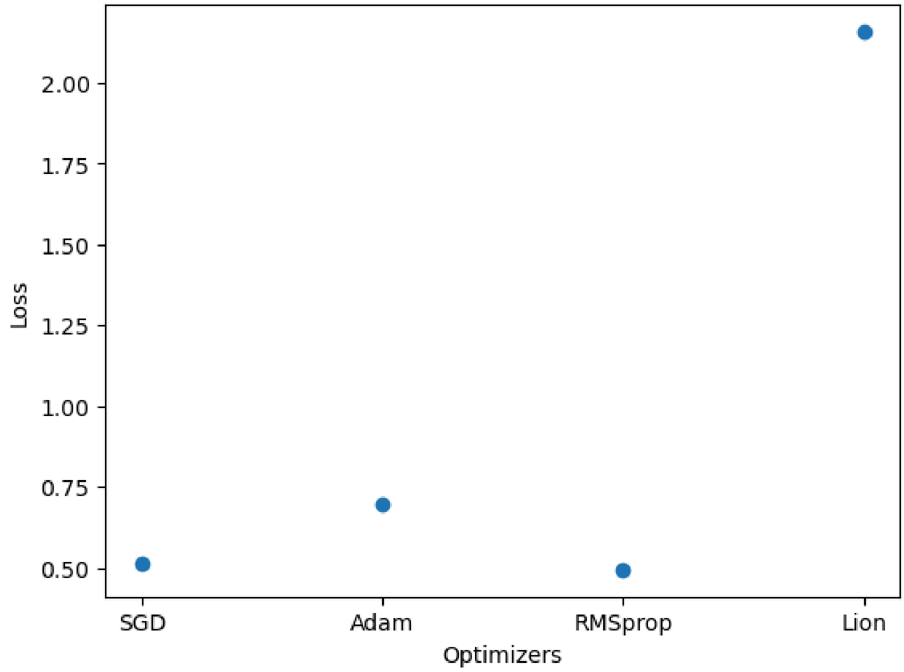
1-F. Compare optimizers

In [14]:
```
# import three new optimizers
from keras.optimizers import Adam, RMSprop, Lion

# define three new optimizers
Adam = Adam(learning_rate=0.1)
RMSprop = RMSprop(learning_rate=0.1)
Lion = Lion(learning_rate=0.1)
```

In [15]:
```python
# define four optimizers as 'activate_functions'

optimizers = [sgd, Adam, RMSprop, Lion]

# define 'scores1_op' for scores of optimizers in assignment 1
scores1_op = []

for op in optimizers:

    # generate the optimal sequential model (4-neuron, 4 hidden layers, activation = 'tanh')
    model = Sequential()
    model.add(Dense(4, input_dim=2, activation='tanh'))
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(1, activation= 'sigmoid'))

    # compile and fit for each optimizer
    model.compile(loss='binary_crossentropy', optimizer=op)
    model.fit(X, y, batch_size=1, epochs=400)

    # generate scores
    score = model.evaluate(X, y)
    scores1_op.append(score)
    print(len(model.layers), scores1_op)
```

```
Epoch 392/400
160/160 [==============================] - 0s 2ms/step - loss: 0.9445
Epoch 393/400
160/160 [==============================] - 0s 2ms/step - loss: 0.9565
Epoch 394/400
160/160 [==============================] - 0s 2ms/step - loss: 1.0003
Epoch 395/400
160/160 [==============================] - 0s 2ms/step - loss: 0.8254
Epoch 396/400
160/160 [==============================] - 0s 2ms/step - loss: 0.8219
Epoch 397/400
160/160 [==============================] - 0s 2ms/step - loss: 0.8802
Epoch 398/400
160/160 [==============================] - 0s 2ms/step - loss: 0.9595
Epoch 399/400
160/160 [==============================] - 0s 2ms/step - loss: 1.2155
Epoch 400/400
160/160 [==============================] - 0s 2ms/step - loss: 0.8178
5/5 [==============================] - 0s 2ms/step - loss: 2.1575
6 [0.5101855397224426, 0.6957978010177612, 0.49299997091293335, 2.1574771404266357]
```

In [16]:
```
#Plot the result
plt.scatter(['SGD', 'Adam', 'RMSprop', 'Lion'], scores1_op)
plt.xlabel('Optimizers')
plt.ylabel('Loss')
plt.show()

print(scores1_op)
```




```
[0.5101855397224426, 0.6957978010177612, 0.49299997091293335, 2.1574771404266357]
```

Result of 1-F:

- The optimizer 'RMSProp' outperforms other optimizers (Loss = 0.4929...)

Overall Result : Optimal Configuration of Assignment 1 (XOR Dataset)

- 4-neuron and 4 hidden layers
- 'tanh' activation function
- 'RMSProp' optimizer

Part 2 - BYOD (Bring your own Dataset)

Using your own dataset, experiment and find the best Neural Network configuration. You may use any resource to improve results, just reference it.

While you may use any dataset, I'd prefer you didn't use the diabetes dataset used in the lesson.

2-A. generating dataset

- Dataset from UCI: Wine Quality ([https://archive.ics.uci.edu/dataset/186/wine+quality (https://archive.ics.uci.edu/dataset/186/wine+quality)](https://archive.ics.uci.edu/dataset/186/wine+quality))
- Classifying good wine (score >= 6) and normal wine (score <6)

In [17]:
```
import pandas as pd

# load dataset
df = pd.read_csv("winequality-red.csv")

# convert quality value into binary value to make classification model
# values from 6 to 10 are transformed into value 1(good wine), and values from 1 to 5 are transformed into
df['quality'] = df.quality.between(6,10).astype(int)

df.head()
```

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0

In [18]:
```
# define X, Y, and transfrom dataframe into numpy array

X = df.drop(['quality'], axis=1).to_numpy()
Y = df['quality'].to_numpy()
X.shape, Y.shape
```

Out[18]: ((1599, 11), (1599,))

2-B. Find optimal number of neurons and layers

- repeat the same process as assignment 1 (1-B to 1-D)
- because there are 11 input variables, extend number of neurons and layers
- number of layers : 1 to 10
- number of neurons : 5, 10, 15

In [19]:
```
num_layers = [1,2,3,4,5,6,7,8,9,10]

# set SGD for optimizer
sgd = SGD(learning_rate=0.1)
```

In [20]:
```python
# define 'scores2_5' for scores 5-neuron model in assignment 2
scores2_5 = []
layer_size = 5

for num_layer in num_layers:

    # generate sequential model and add input layer
    model = Sequential()
    model.add(Dense(layer_size, input_dim=11, activation='tanh'))

    # add hidden layers
    for _ in range(num_layer):
        model.add(Dense(layer_size, activation='tanh'))

    # add output layers
    model.add(Dense(1, activation='sigmoid'))

    # compile and print
    # set batch_size=10 to use 10% of datset as samples
    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, Y, batch_size=10, epochs=200)

    # calculate scores
    score = model.evaluate(X, Y)
    scores2_5.append(score)
    print(len(model.layers), scores2_5)
```

```
Epoch 192/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6912
Epoch 193/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6914
Epoch 194/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6914
Epoch 195/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6911
Epoch 196/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6913
Epoch 197/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6912
Epoch 198/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6911
Epoch 199/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6913
Epoch 200/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6913
50/50 [==============================] - 0s 2ms/step - loss: 0.6908
```

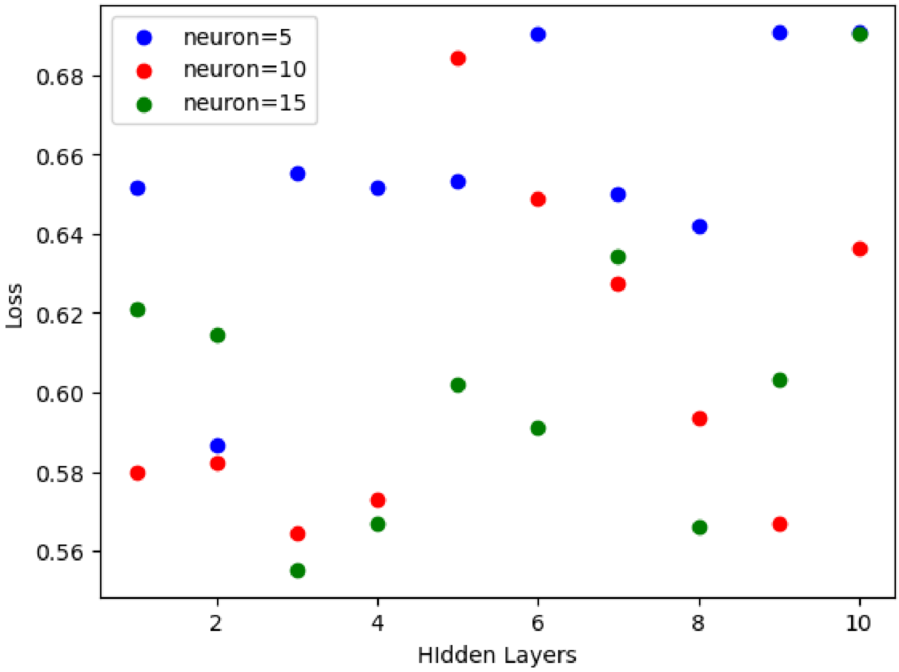
In [21]:
```
# Same process with the previous cell
# define 'scores2_10' for scores 10-neuron model in assignment 2
scores2_10 = []
layer_size = 10

for num_layer in num_layers:

    model = Sequential()
    model.add(Dense(layer_size, input_dim=11, activation='tanh'))

    for _ in range(num_layer):
        model.add(Dense(layer_size, activation='tanh'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, Y, batch_size=10, epochs=200)

    score = model.evaluate(X, Y)
    scores2_10.append(score)
    print(len(model.layers), scores2_10)
```

```
Epoch 192/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6092
Epoch 193/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6142
Epoch 194/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6053
Epoch 195/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6023
Epoch 196/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5986
Epoch 197/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6070
Epoch 198/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6135
Epoch 199/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6137
Epoch 200/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6055
50/50 [==============================] - 1s 3ms/step - loss: 0.6363
```

In [22]:
```python
# Same process with the previous cell
# define 'scores2_15' for scores 15-neuron model in assignment 2
scores2_15 = []
layer_size = 15

for num_layer in num_layers:

    model = Sequential()
    model.add(Dense(layer_size, input_dim=11, activation='tanh'))

    for _ in range(num_layer):
        model.add(Dense(layer_size, activation='tanh'))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, Y, batch_size=10, epochs=200)

    score = model.evaluate(X, Y)
    scores2_15.append(score)
    print(len(model.layers), scores2_15)
```

```
Epoch 192/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6905
Epoch 193/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6901
Epoch 194/200
160/160 [==============================] - 1s 5ms/step - loss: 0.6908
Epoch 195/200
160/160 [==============================] - 1s 4ms/step - loss: 0.6909
Epoch 196/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6913
Epoch 197/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6908
Epoch 198/200
160/160 [==============================] - 1s 3ms/step - loss: 0.6909
Epoch 199/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6909
Epoch 200/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6911
50/50 [==============================] - 1s 4ms/step - loss: 0.6903
12 [0.6210130453109741, 0.6145344972610474, 0.5552312731742859, 0.566909670829773, 0.601902246475219
```

In [23]:
```
plt.scatter(num_layers, scores2_5, c='b')
plt.scatter(num_layers, scores2_10, c='r')
plt.scatter(num_layers, scores2_15, c='g')
plt.legend(['neuron=5', 'neuron=10', 'neuron=15'])
plt.xlabel('HIdden Layers')
plt.ylabel('Loss')
plt.show()

print(scores2_5)
print(scores2_10)
print(scores2_15)
```




```
[0.6515727639198303, 0.5866639018058777, 0.6554192304611206, 0.6515476703643799, 0.6530188918113708, 0.6
902503371238708, 0.6500072479248047, 0.6418786644935608, 0.6906289458274841, 0.6907546520233154]
[0.5796726942062378, 0.5824112296104431, 0.5644353628158569, 0.5727626085281372, 0.6844196319580078, 0.6
489644646644592, 0.6275237202644348, 0.5934621691703796, 0.5667896866798401, 0.6362775564193726]
[0.6210130453109741, 0.6145344972610474, 0.5552312731742859, 0.566909670829773, 0.6019022464752197, 0.59
09478664398193, 0.6342706084251404, 0.566011905670166, 0.6032663583755493, 0.6902809143066406]
```

Result of 2-B:

- 15-neuron model with 3 hidden layers is the optimal configuration (Loss = 0.5552..)

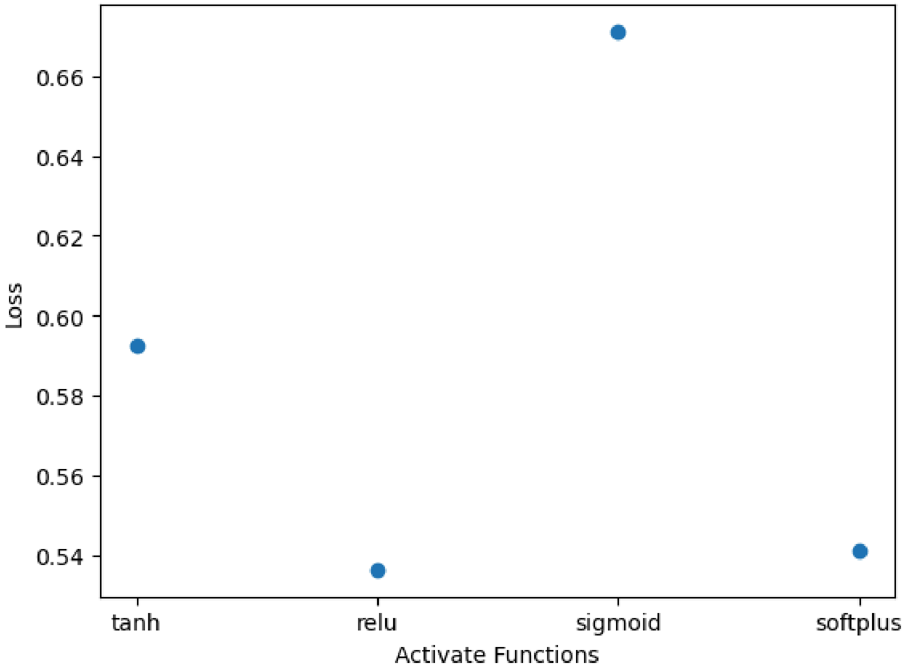
2-C. Find optimal activate function

- repeat the same process as assignment 1 (1-E)

In [24]:
```
# define four activate functions as 'activate_functions'
activate_functions = ['tanh', 'relu', 'sigmoid', 'softplus']

# define 'scores2_af' for scores of activate functions in assignment 2
scores2_af = []

# use 'for' loop to acquire score for each function
for af in activate_functions:

    # generate the optimal sequential model(15-neuron, 3 hidden layers) for each activate function.
    model = Sequential()
    model.add(Dense(15, input_dim=11, activation=af))
    model.add(Dense(15, activation=af))
    model.add(Dense(15, activation=af))
    model.add(Dense(15, activation=af))

    # keep activation function of the output layer as 'sigmoid'
    model.add(Dense(1, activation='sigmoid'))

    # compile and fit
    model.compile(loss='binary_crossentropy', optimizer='sgd')
    model.fit(X, Y, batch_size=10, epochs=200)

    score = model.evaluate(X, Y)
    scores2_af.append(score)
    print(len(model.layers), scores2_af)
```

```
160/160 [==============================] - 0s 2ms/step - loss: 0.5619
Epoch 193/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5572
Epoch 194/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5575
Epoch 195/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5547
Epoch 196/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5553
Epoch 197/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5555
Epoch 198/200
160/160 [==============================] - 0s 2ms/step - loss: 0.5534
Epoch 199/200
160/160 [==============================] - 0s 3ms/step - loss: 0.5597
Epoch 200/200
160/160 [==============================] - 0s 3ms/step - loss: 0.5600
50/50 [==============================] - 0s 3ms/step - loss: 0.5411
5 [0.5924168825149536, 0.5362298488616943, 0.6712279915809631, 0.541130542755127]
```

In [25]:
```
#Plot the result
plt.scatter(activate_functions, scores2_af)
plt.xlabel('Activate Functions')
plt.ylabel('Loss')
plt.show()

print(scores2_af)
```




```
[0.5924168825149536, 0.5362298488616943, 0.6712279915809631, 0.541130542755127]
```

Result of 2-C:

- relu is the optimal activate function (Loss = 0.5362..)

2-D. Find optimal optimizer

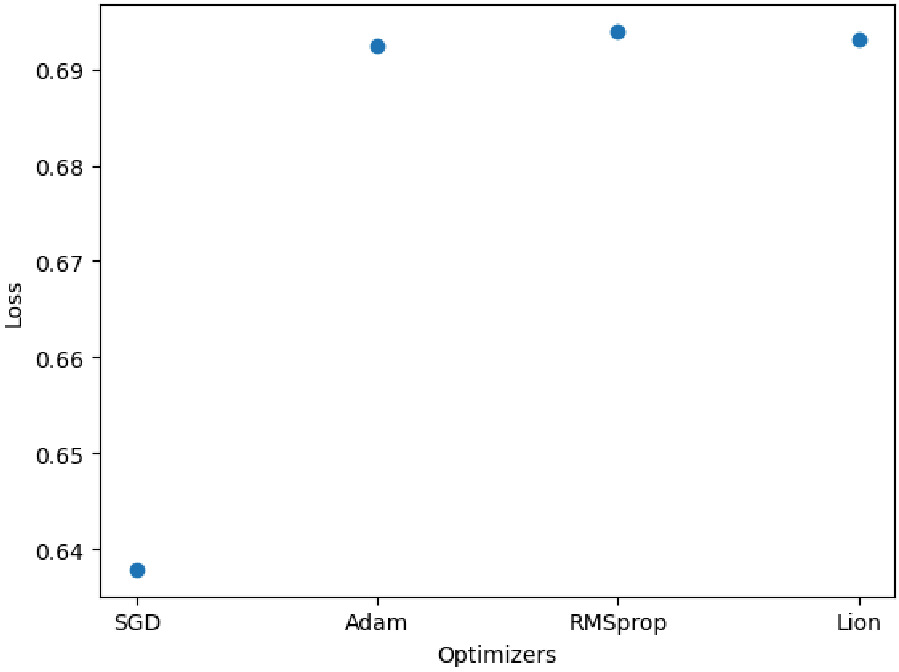
- repeat the process as assignment 1 (1-F)

In [32]:
```
# because optimizer 'SGD', 'Adam', 'RMSprop' led to error, I imported legacy optimizers
# The error message was :'The optimizer cannot recognize variable dense_373/kernel:0

from keras.optimizers.legacy import SGD, Adam, RMSprop
from keras.optimizers import Lion

# define three new optimizers
sgd = SGD(learning_rate=0.1)
adam = Adam(learning_rate=0.1)
rmsprop = RMSprop(learning_rate=0.1)
lion = Lion(learning_rate=0.1)
```

In [33]:
```
# define four optimizers as 'activate_functions'

optimizers = [sgd, adam, rmsprop, lion]

# define 'scores2_op' for scores of optimizers in assignment 2
scores2_op = []

for op in optimizers:

   # generate the optimal sequential model (15-neuron, 3-hidden layer, activation = 'relu')
    model = Sequential()
    model.add(Dense(15, input_dim=11, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(15, activation='relu'))
    model.add(Dense(15, activation='relu'))

    # keep activation function of the output layer as 'sigmoid'
    model.add(Dense(1, activation= 'sigmoid'))

    # compile and fit for each optimizer
    model.compile(loss='binary_crossentropy', optimizer=op)
    model.fit(X, Y, batch_size=10, epochs=200)

    # generate scores
    score = model.evaluate(X, Y)
    scores2_op.append(score)
    print(len(model.layers), scores2_op)
```

```
Epoch 192/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6958
Epoch 193/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6946
Epoch 194/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6969
Epoch 195/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6978
Epoch 196/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6973
Epoch 197/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6996
Epoch 198/200
160/160 [==============================] - 0s 2ms/step - loss: 0.6986
Epoch 199/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6964
Epoch 200/200
160/160 [==============================] - 0s 3ms/step - loss: 0.6984
50/50 [==============================] - 0s 2ms/step - loss: 0.6931
5 [0.6378671526908875, 0.6923683285713196, 0.6939593553543091, 0.6931474208831787]
```

In [34]:
```
#Plot the result
plt.scatter(['SGD', 'Adam', 'RMSprop', 'Lion'], scores2_op)
plt.xlabel('Optimizers')
plt.ylabel('Loss')
plt.show()

print(scores2_op)
```




```
[0.6378671526908875, 0.6923683285713196, 0.6939593553543091, 0.6931474208831787]
```

Result of 2-D:

- SGD is the best optimizer (Loss = 0.6379..)

Overall Result : Optimal Configuration of Assignment 2 (Wine Quality Dataset)

- 15-neuron and 3 hidden layers
- 'relu' activation function
- 'SGD' optimizer