11 - Neural Networks

Part 1 - XOR

1. Using the XOR dataset below, train (400 epochs) a neural network (NN) using 1, 2, 3, 4, and 5 hidden layers (where each layer has only 2 neurons). For each n layers, store the resulting loss score along with n. Plot the results to find what the optimal number of layers is.
2. Repeat the above with 3-neuron and 4-neuron Hidden layers. How do these results compare to the 2 neuron layers?
3. 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/)](https://keras.io/activations/))
4. Instead of `SGD` try other optimizers and report on the loss score. ([https://keras.io/optimizers/ (https://keras.io/optimizers/)](https://keras.io/optimizers/))

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

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

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

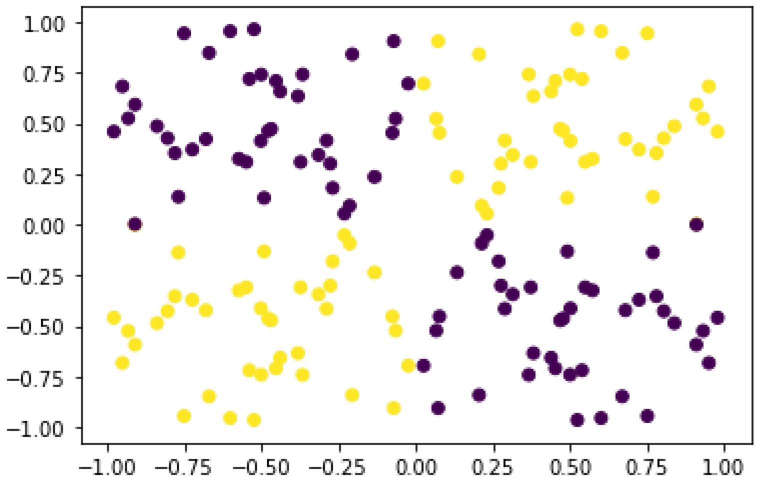
```
Using TensorFlow backend.
```

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*r
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 0x170b56a8b08>



1 layer, 2 neurons

In [5]:
```
model1 = Sequential()

model1.add(Dense(2, input_dim=2, activation='tanh'))  #first layer
model1.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1)
model1.compile(loss='binary_crossentropy', optimizer='sgd')

model1.fit(X, y, batch_size=2, epochs=400) #160/4 = 40 per epoch.  we fit the dat
print(model1.predict_proba(X).reshape(4*n))

scores = model1.evaluate(X, y)  # evaluate the model
```

```
80/80 [==============================] - 0s 1ms/step - loss: 0.6875
Epoch 36/400
80/80 [==============================] - 0s 947us/step - loss: 0.6872
Epoch 37/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6868
Epoch 38/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6864
Epoch 39/400
80/80 [==============================] - 0s 898us/step - loss: 0.6858
Epoch 40/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6855
Epoch 41/400
80/80 [==============================] - 0s 972us/step - loss: 0.6851
Epoch 42/400
80/80 [==============================] - 0s 972us/step - loss: 0.6845
Epoch 43/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6838
Epoch 44/400
80/80 [==============================] - 0s 960us/step - loss: 0.6833
Epoch 45/400
```

In [6]:
```
scores1 = model1.evaluate(X, y)
scores1, model1.metrics_names
```

```
5/5 [==============================] - 0s 1ms/step - loss: 0.3173
```

Out[6]:
```
(0.3173014521598816, ['loss'])
```

plt.scatter(*zip*(X), c=model1.predict(X).reshape(4*n))

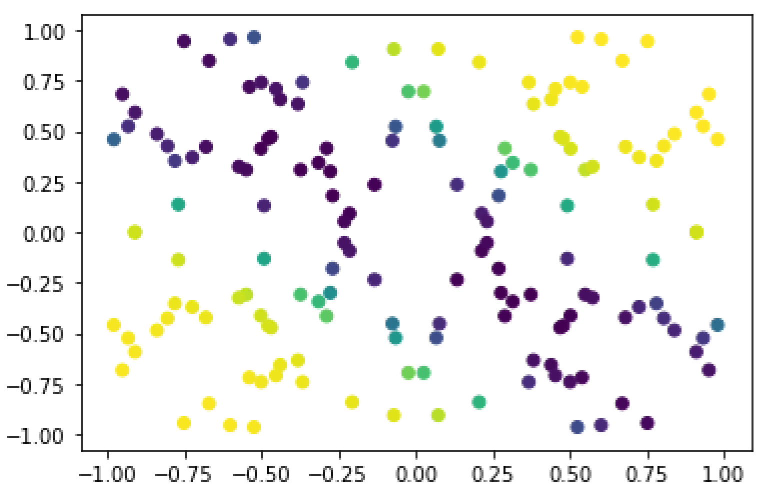
2 layers, 2 neurons

In [7]:
```python
model2 = Sequential()

model2.add(Dense(2, input_dim=2, activation='tanh'))  #first layer
model2.add(Dense(2, activation='tanh')) #second layer
model2.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1)
model2.compile(loss='binary_crossentropy', optimizer='sgd')

model2.fit(X, y, batch_size=2, epochs=400)
print(model2.predict_proba(X).reshape(4*n))

scores2 = model2.evaluate(X, y) # evaluate the model
```

```
 0.74615055 0.90717804 0.10576588 0.9549526  0.9662323  0.7780062
 0.9678076  0.9685021  0.09452382 0.8860277  0.96701056 0.15198445
 0.96692574 0.96899426 0.96730506 0.92001367 0.9433472  0.5926153
 0.70934296 0.92005116 0.9503858  0.962932   0.9061436  0.67016655
 0.90433824 0.5030033  0.18561235 0.201395   0.08401108 0.12929755
 0.34707493 0.07295185 0.05253154 0.8838269  0.05291629 0.09291396
 0.08042651 0.1475657  0.09748438 0.29919767 0.0612832  0.90606844
 0.0556474  0.09977868 0.17404866 0.7151205  0.17144898 0.08196738
 0.06314722 0.07592601 0.10373479 0.05765411 0.28540272 0.07764047
 0.07306796 0.051173   0.6616037  0.1381262  0.05158785 0.05084795
 0.19007108 0.12680581 0.6015444  0.05204031 0.07836878 0.05181479
 0.16408855 0.23152715 0.0761663  0.1504196  0.3839514  0.06720281
 0.05546361 0.8802028  0.05568403 0.08379501 0.09201756 0.1717977
 0.0873971  0.27158833 0.05757689 0.911978   0.05952567 0.08952773
 0.20236537 0.7062851  0.15367556 0.09465659 0.06941399 0.08612427
 0.1210503  0.05470872 0.26152593 0.07144606 0.06751639 0.05136165
 0.650609   0.15948981 0.05366191 0.05156997 0.16964638 0.14803526
 0.6234674  0.05142802 0.08920226 0.05132195]
5/5 [==============================] - 0s 1ms/step - loss: 0.3159
```

In [8]:
```python
plt.scatter(*zip(*X), c=model2.predict(X).reshape(4*n))
```

Out[8]: <matplotlib.collections.PathCollection at 0x170b6dcd9c8>



3 layers, 2 neurons

In [9]:
```
model3 = Sequential()

model3.add(Dense(2, input_dim=2, activation='tanh'))  #first layer
model3.add(Dense(2, activation='tanh')) #second layer
model3.add(Dense(2, activation='tanh')) #third layer
model3.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1)
model3.compile(loss='binary_crossentropy', optimizer='sgd')

model3.fit(X, y, batch_size=2, epochs=400)
print(model3.predict_proba(X).reshape(4*n))

scores3 = model3.evaluate(X, y) # evaluate the model

plt.scatter(*zip(*X), c=model3.predict(X).reshape(4*n))
```

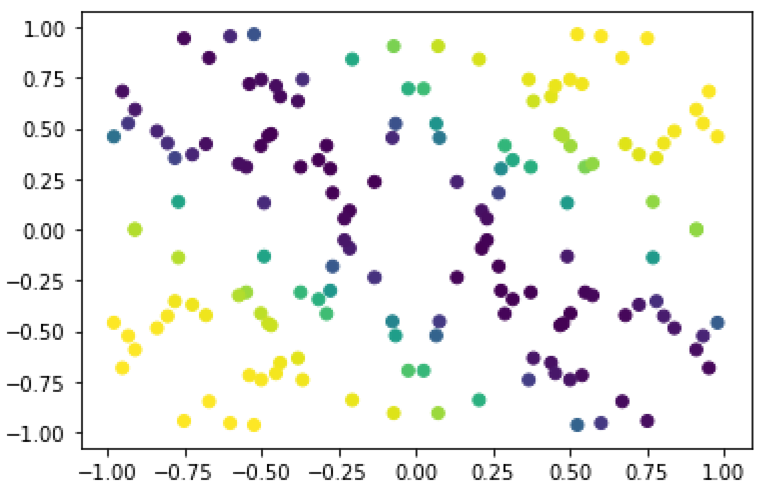
```
Epoch 45/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4727
Epoch 46/400
80/80 [==============================] - 0s 2ms/step - loss: 0.4695
Epoch 47/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4677
Epoch 48/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4653

Epoch 49/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4634
Epoch 50/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4619
Epoch 51/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4597
Epoch 52/400
80/80 [==============================] - 0s 1ms/step - loss: 0.4581
Epoch 53/400
80/80 [==============================] - 0s 2ms/step - loss: 0.4567
Epoch 54/400
```

4 layers, 2 neurons

In [12]:
```python
model4 = Sequential()

model4.add(Dense(2, input_dim=2, activation='tanh'))  #first layer
model4.add(Dense(2, activation='tanh')) #second layer
model4.add(Dense(2, activation='tanh')) #third layer
model4.add(Dense(2, activation='tanh')) #fourth layer
model4.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1)
model4.compile(loss='binary_crossentropy', optimizer='sgd')

model4.fit(X, y, batch_size=2, epochs=400)
print(model4.predict_proba(X).reshape(4*n))

scores4 = model4.evaluate(X, y) # evaluate the model

plt.scatter(*zip(*X), c=model4.predict(X).reshape(4*n))
```

```
 0.5551992  0.04853067 0.06210914 0.04820016]
5/5 [==============================] - 0s 1ms/step - loss: 0.3103
```

Out[12]: <matplotlib.collections.PathCollection at 0x170b93952c8>

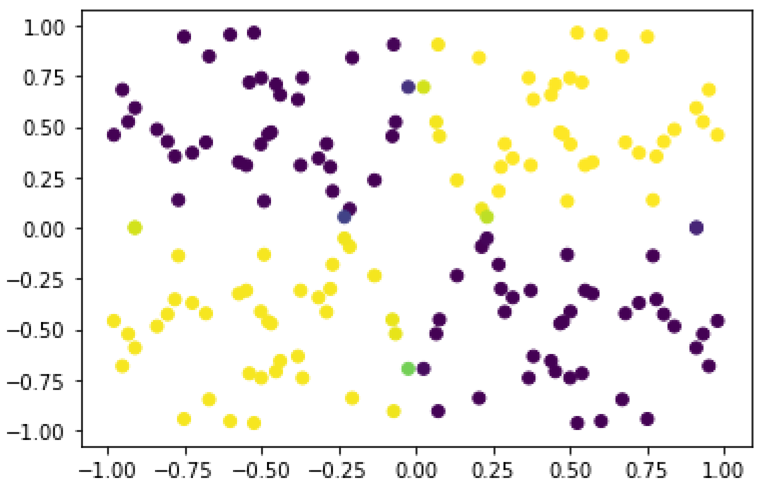


5 layers, 2 neurons

In [14]:
```python
model5 = Sequential()

model5.add(Dense(2, input_dim=2, activation='tanh'))  #first layer
model5.add(Dense(2, activation='tanh')) #second layer
model5.add(Dense(2, activation='tanh')) #third layer
model5.add(Dense(2, activation='tanh')) #fourth layer
model5.add(Dense(2, activation='tanh')) #fifth layer
model5.add(Dense(1, activation='sigmoid'))

sgd = SGD(lr=0.1)
model5.compile(loss='binary_crossentropy', optimizer='sgd')

model5.fit(X, y, batch_size=2, epochs=400)
print(model5.predict_proba(X).reshape(4*n))

scores5 = model5.evaluate(X, y) # evaluate the model

plt.scatter(*zip(*X), c=model5.predict(X).reshape(4*n))
```

```
 0.01390851 0.01134229 0.01162797 0.01133832]
5/5 [==============================] - 0s 1ms/step - loss: 0.0482
```

Out[14]: `<matplotlib.collections.PathCollection at 0x170b935bb08>`



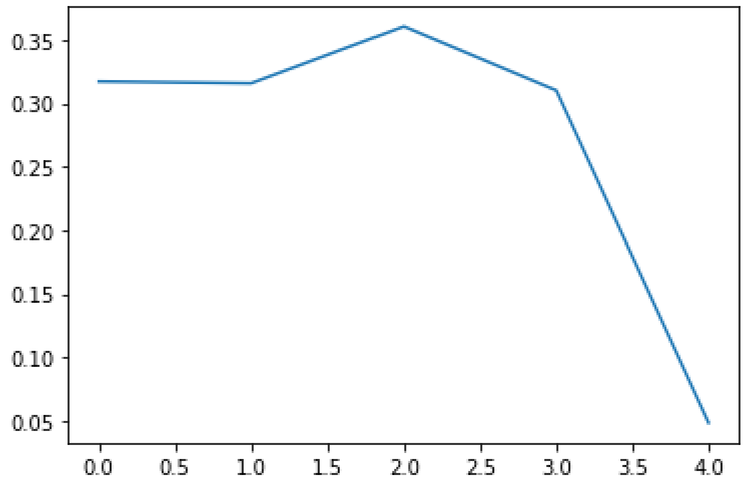
Plotting the results

In [15]:
```python
df = np.array([scores1, scores2, scores3, scores4, scores5])
df
```

Out[15]: `array([0.31730145, 0.31588522, 0.36067224, 0.31026211, 0.0482264 ])`

In [16]:
```
plt.plot(df)
```

Out[16]: [<matplotlib.lines.Line2D at 0x170b9537208>]



Part 1 - continued

2. Repeat the above with 3-neuron and 4-neuron Hidden layers. How do these results compare to the 2 neuron layers?

3-neuron models

In [18]:
```
# 1 layer
model1_3 = Sequential()
model1_3.add(Dense(3, input_dim=2, activation='tanh'))  #first layer
model1_3.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model1_3.compile(loss='binary_crossentropy', optimizer='sgd')
model1_3.fit(X, y, batch_size=2, epochs=400)
print(model1_3.predict_proba(X).reshape(4*n))
scores1_3 = model1_3.evaluate(X, y)

# 2 layers
model2_3 = Sequential()
model2_3.add(Dense(3, input_dim=2, activation='tanh'))  #first layer
model2_3.add(Dense(3, activation='tanh')) #second layer
model2_3.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model2_3.compile(loss='binary_crossentropy', optimizer='sgd')
model2_3.fit(X, y, batch_size=2, epochs=400)
print(model2_3.predict_proba(X).reshape(4*n))
scores2_3 = model2_3.evaluate(X, y) # evaluate the model

# 3 layers
model3_3 = Sequential()
model3_3.add(Dense(3, input_dim=2, activation='tanh'))  #first layer
model3_3.add(Dense(3, activation='tanh')) #second layer
model3_3.add(Dense(3, activation='tanh')) #third layer
model3_3.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model3_3.compile(loss='binary_crossentropy', optimizer='sgd')
model3_3.fit(X, y, batch_size=2, epochs=400)
print(model3_3.predict_proba(X).reshape(4*n))
scores3_3 = model3_3.evaluate(X, y) # evaluate the model

# 4 layers
model4_3 = Sequential()
model4_3.add(Dense(3, input_dim=2, activation='tanh'))  #first layer
model4_3.add(Dense(3, activation='tanh')) #second layer
model4_3.add(Dense(3, activation='tanh')) #third layer
model4_3.add(Dense(3, activation='tanh')) #fourth layer
model4_3.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model4_3.compile(loss='binary_crossentropy', optimizer='sgd')
model4_3.fit(X, y, batch_size=2, epochs=400)
print(model4_3.predict_proba(X).reshape(4*n))
scores4_3 = model4_3.evaluate(X, y) # evaluate the model

# 5 layers
model5_3 = Sequential()
model5_3.add(Dense(3, input_dim=2, activation='tanh'))  #first layer
model5_3.add(Dense(3, activation='tanh')) #second layer
model5_3.add(Dense(3, activation='tanh')) #third layer
model5_3.add(Dense(3, activation='tanh')) #fourth layer
model5_3.add(Dense(3, activation='tanh')) #fifth layer
model5_3.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model5_3.compile(loss='binary_crossentropy', optimizer='sgd')
```

```
model5_3.fit(X, y, batch_size=2, epochs=400)
print(model5_3.predict_proba(X).reshape(4*n))
scores5_3 = model5_3.evaluate(X, y) # evaluate the model
```

```
 0.9930178  0.8335678  0.9727447  0.9949606  0.9957796  0.7714665
 0.9964323  0.9963763  0.94380397 0.9933458  0.99600005 0.98400676
 0.99631226 0.99663216 0.9962824  0.99385124 0.9942194  0.9906001
 0.9929278  0.9938419  0.99475217 0.9952544  0.99090946 0.99287117
 0.9934825  0.99242556 0.01773462 0.00457767 0.00425583 0.00451839
 0.00447074 0.00427061 0.00436348 0.01159886 0.00487137 0.00426149
 0.00442171 0.00445125 0.00424767 0.02885684 0.00425577 0.8132191
 0.00758037 0.00425449 0.00440139 0.25771248 0.004262   0.00433728
 0.04112327 0.00464192 0.00436077 0.00457802 0.00425279 0.00427502
 0.00427595 0.00430402 0.00452453 0.01799223 0.00441068 0.00431323
 0.00424302 0.00440365 0.01283097 0.00426871 0.00459141 0.00425866
 0.00450107 0.00763521 0.00663856 0.00755537 0.00688237 0.00705406
 0.00713941 0.00488397 0.00890303 0.00705037 0.00725171 0.00713745
 0.00624803 0.00497016 0.00544262 0.7473177  0.01698297 0.00681308
 0.00680828 0.01970136 0.008091   0.00677314 0.10041916 0.008481
 0.00678292 0.00427267 0.00831807 0.00761643 0.0073989  0.00677887
 0.00482103 0.05042472 0.00726756 0.00684345 0.00636938 0.00693497
 0.02555111 0.00607866 0.00820667 0.00586253]
5/5 [==============================] - 0s 1ms/step - loss: 0.0331
```

In [19]:
```
df_3 = np.array([scores1_3, scores2_3, scores3_3, scores4_3, scores5_3])
df_3
```

Out[19]:
```
array([0.16349804, 0.09726368, 0.31269667, 0.02364234, 0.03312378])
```

4-neuron models

In [20]:
```python
# 1 layer
model1_4 = Sequential()
model1_4.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model1_4.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model1_4.compile(loss='binary_crossentropy', optimizer='sgd')
model1_4.fit(X, y, batch_size=2, epochs=400)
print(model1_4.predict_proba(X).reshape(4*n))
scores1_4 = model1_4.evaluate(X, y)

# 2 layers
model2_4 = Sequential()
model2_4.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model2_4.add(Dense(4, activation='tanh')) #second layer
model2_4.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model2_4.compile(loss='binary_crossentropy', optimizer='sgd')
model2_4.fit(X, y, batch_size=2, epochs=400)
print(model2_4.predict_proba(X).reshape(4*n))
scores2_4 = model2_4.evaluate(X, y) # evaluate the model

# 3 layers
model3_4 = Sequential()
model3_4.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model3_4.add(Dense(4, activation='tanh')) #second layer
model3_4.add(Dense(4, activation='tanh')) #third layer
model3_4.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model3_4.compile(loss='binary_crossentropy', optimizer='sgd')
model3_4.fit(X, y, batch_size=2, epochs=400)
print(model3_4.predict_proba(X).reshape(4*n))
scores3_4 = model3_4.evaluate(X, y) # evaluate the model

# 4 layers
model4_4 = Sequential()
model4_4.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model4_4.add(Dense(4, activation='tanh')) #second layer
model4_4.add(Dense(4, activation='tanh')) #third layer
model4_4.add(Dense(4, activation='tanh')) #fourth layer
model4_4.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model4_4.compile(loss='binary_crossentropy', optimizer='sgd')
model4_4.fit(X, y, batch_size=2, epochs=400)
print(model4_4.predict_proba(X).reshape(4*n))
scores4_4 = model4_4.evaluate(X, y) # evaluate the model

# 5 layers
model5_4 = Sequential()
model5_4.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model5_4.add(Dense(4, activation='tanh')) #second layer
model5_4.add(Dense(4, activation='tanh')) #third layer
model5_4.add(Dense(4, activation='tanh')) #fourth layer
model5_4.add(Dense(4, activation='tanh')) #fifth layer
model5_4.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model5_4.compile(loss='binary_crossentropy', optimizer='sgd')
```

```
model5_4.fit(X, y, batch_size=2, epochs=400)
print(model5_4.predict_proba(X).reshape(4*n))
scores5_4 = model5_4.evaluate(X, y) #evaluate the model
```

```
 9.9881852e-01 9.9642766e-01 9.9855804e-01 9.9859798e-01 9.9867213e-01
 9.9854243e-01 9.9465132e-01 9.9863374e-01 9.9848831e-01 9.9864197e-01
 9.4312429e-04 2.1353662e-03 1.0594726e-03 1.8286109e-03 3.0544996e-03
 1.1482537e-03 1.2022555e-03 9.4985962e-04 1.0387003e-03 1.0983944e-03
 1.5938282e-03 2.0766854e-03 1.0147691e-03 1.0327697e-03 9.6467137e-04
 4.5830736e-01 1.3356507e-03 1.0603964e-03 2.5803149e-03 8.6001158e-03
 1.1397600e-03 2.4036169e-03 5.7303011e-03 1.3760328e-03 2.3299456e-03
 8.4057450e-04 1.0600984e-03 1.3822913e-03 1.2838244e-03 1.1356473e-03
 8.8140368e-04 3.3524036e-03 1.0711551e-03 1.1507273e-03 9.8606944e-04
 2.1381974e-03 1.1148363e-02 9.9623203e-04 1.4131963e-03 9.6538663e-04
 2.1280646e-03 3.7652850e-03 2.2040904e-03 3.5057664e-03 3.7088990e-03
 2.4352670e-03 2.6372671e-03 2.4102032e-03 2.9561222e-03 2.3311377e-03
 3.1637549e-03 3.4661293e-03 2.0642281e-03 2.4262369e-03 1.9360185e-03
 6.2009609e-01 7.9893768e-03 2.2216439e-03 3.4881830e-03 1.4453262e-02
 2.4994612e-03 3.3524334e-03 7.3363870e-02 3.3100843e-03 3.3559799e-03
 1.7821193e-03 2.3247898e-03 2.8271079e-03 2.6907921e-03 2.4204850e-03
 1.7154813e-03 1.7299235e-02 2.4747849e-03 2.4637878e-03 2.0032823e-03
 3.3796132e-03 1.9443780e-02 2.1184087e-03 3.2892823e-03 2.0761788e-03]
5/5 [==============================] - 0s 1ms/step - loss: 0.0207
```

In [21]:
```
df_4 = np.array([scores1_4, scores2_4, scores3_4, scores4_4, scores5_4])
df_4
```

Out[21]:
```
array([0.15961495, 0.03270616, 0.02293862, 0.02196405, 0.02074153])
```

Part 1 - continued

3. 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/)](https://keras.io/activations/))

In [28]:
```
df
```

Out[28]:
```
array([0.31730145, 0.31588522, 0.36067224, 0.31026211, 0.0482264 ])
```

In [29]:
```
df_3
```

Out[29]:
```
array([0.16349804, 0.09726368, 0.31269667, 0.02364234, 0.03312378])
```

In [30]:
```
df_4
```

Out[30]:
```
array([0.15961495, 0.03270616, 0.02293862, 0.02196405, 0.02074153])
```

The model with 5 layers and 4 neurons resulted in lowest loss. Will use to compare with other models

In [32]:
```
model5_4 = Sequential()
model5_4.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model5_4.add(Dense(4, activation='tanh')) #second layer
model5_4.add(Dense(4, activation='tanh')) #third layer
model5_4.add(Dense(4, activation='tanh')) #fourth layer
model5_4.add(Dense(4, activation='tanh')) #fifth layer
model5_4.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model5_4.compile(loss='binary_crossentropy', optimizer='sgd')
model5_4.fit(X, y, batch_size=2, epochs=400)
print(model5_4.predict_proba(X).reshape(4*n))
scores_tanh = model5_4.evaluate(X, y) # evaluate the model

# sigmoid
model_sig = Sequential()
model_sig.add(Dense(4, input_dim=2, activation='sigmoid'))  #first layer
model_sig.add(Dense(4, activation='sigmoid')) #second layer
model_sig.add(Dense(4, activation='sigmoid')) #third layer
model_sig.add(Dense(4, activation='sigmoid')) #fourth layer
model_sig.add(Dense(4, activation='sigmoid')) #fifth layer
model_sig.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model_sig.compile(loss='binary_crossentropy', optimizer='sgd')
model_sig.fit(X, y, batch_size=2, epochs=400)
print(model_sig.predict_proba(X).reshape(4*n))
scores_sig = model_sig.evaluate(X, y) # evaluate the model

# softplus
model_soft = Sequential()
model_soft.add(Dense(4, input_dim=2, activation='softplus'))  #first layer
model_soft.add(Dense(4, activation='softplus')) #second layer
model_soft.add(Dense(4, activation='softplus')) #third layer
model_soft.add(Dense(4, activation='softplus')) #fourth layer
model_soft.add(Dense(4, activation='softplus')) #fifth layer
model_soft.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model_soft.compile(loss='binary_crossentropy', optimizer='sgd')
model_soft.fit(X, y, batch_size=2, epochs=400)
print(model_soft.predict_proba(X).reshape(4*n))
scores_soft = model_soft.evaluate(X, y) # evaluate the model

# relu
model_relu = Sequential()
model_relu.add(Dense(4, input_dim=2, activation='relu'))  #first layer
model_relu.add(Dense(4, activation='relu')) #second layer
model_relu.add(Dense(4, activation='relu')) #third layer
model_relu.add(Dense(4, activation='relu')) #fourth layer
model_relu.add(Dense(4, activation='relu')) #fifth layer
model_relu.add(Dense(1, activation='sigmoid'))
sgd = SGD(lr=0.1)
model_relu.compile(loss='binary_crossentropy', optimizer='sgd')
model_relu.fit(X, y, batch_size=2, epochs=400)
print(model_relu.predict_proba(X).reshape(4*n))
scores_relu = model_relu.evaluate(X, y) # evaluate the model
```

```
80/80 [==============================] - 0s 1ms/step - loss: 0.6654
Epoch 11/400
```

```
Epoch 11/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6584
Epoch 12/400
80/80 [==============================] - 0s 935us/step - loss: 0.6519
Epoch 13/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6437
Epoch 14/400
80/80 [==============================] - 0s 997us/step - loss: 0.6380
Epoch 15/400
80/80 [==============================] - 0s 972us/step - loss: 0.6311
Epoch 16/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6251
Epoch 17/400
80/80 [==============================] - 0s 985us/step - loss: 0.6204
Epoch 18/400
80/80 [==============================] - 0s 947us/step - loss: 0.6154
Epoch 19/400
80/80 [==============================] - 0s 1ms/step - loss: 0.6114
Epoch 20/400
```

In [33]:
```
scores_tanh
```

Out[33]: 0.020530756562948227

In [34]:
```
scores_sig
```

Out[34]: 0.6931532621383667

In [36]:
```
scores_soft
```

Out[36]: 0.0738680362701416

In [37]:
```
scores_relu
```

Out[37]: 0.022404631599783897

Part 1 - continued

4. Instead of `SGD` try other optimizers and report on the loss score. ([https://keras.io/optimizers/ (https://keras.io/optimizers/)](https://keras.io/optimizers/))

will use the `tanh` model with `sgd` and compare to other optimizers

In [40]:

```python
# adam optimizer
from tensorflow import keras
from tensorflow.keras import layers

model_adam = Sequential()
model_adam.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model_adam.add(Dense(4, activation='tanh')) #second layer
model_adam.add(Dense(4, activation='tanh')) #third layer
model_adam.add(Dense(4, activation='tanh')) #fourth layer
model_adam.add(Dense(4, activation='tanh')) #fifth layer
model_adam.add(Dense(1, activation='sigmoid'))
adam = keras.optimizers.Adam(learning_rate=0.01)
model_adam.compile(loss='binary_crossentropy', optimizer='adam')
model_adam.fit(X, y, batch_size=2, epochs=400)
print(model_adam.predict_proba(X).reshape(4*n))
scores_adam = model_adam.evaluate(X, y) # evaluate the model
```

```
 9.9904478e-01 9.9630034e-01 9.9829805e-01 9.9843597e-01 9.9879003e-01
 9.9820125e-01 9.9532622e-01 9.9849010e-01 9.9807596e-01 9.9849379e-01
 6.7308545e-04 2.2098422e-04 3.5175681e-04 2.3555756e-04 2.9137731e-04
 3.6633015e-04 2.7906895e-04 2.9352307e-04 1.6680360e-04 3.6087632e-04
 2.7540326e-04 2.7349591e-04 3.4230947e-04 5.1289797e-04 2.9397011e-04
 6.9679499e-01 5.6260824e-04 3.5387278e-04 3.3193827e-04 6.6405833e-03
 3.6600232e-04 3.8492680e-04 7.0707500e-03 2.0051003e-04 3.6174059e-04
 2.7599931e-04 3.5184622e-04 3.8340688e-04 3.7926435e-04 3.1465292e-04
 3.2043457e-04 3.6230683e-04 2.1570921e-04 3.0982494e-04 3.3518672e-04
 3.1182170e-04 4.6333671e-04 2.4867058e-04 2.0974874e-04 2.2357702e-04
 8.1494451e-04 1.4898777e-03 1.2208223e-03 1.4412105e-03 1.3707280e-03
 1.2377501e-03 1.3180673e-03 1.0784864e-03 1.8896461e-03 1.2298226e-03
 1.3602674e-03 1.3792813e-03 1.2030005e-03 8.5097551e-04 1.1889935e-03
 6.0804570e-01 5.0607920e-03 1.2210608e-03 1.3224185e-03 5.6765676e-03
 1.2376010e-03 1.2777746e-03 2.5666893e-02 1.5205145e-03 1.2945831e-03
 9.9918246e-04 1.2273490e-03 1.2497008e-03 1.2467206e-03 1.2682676e-03
 9.9945068e-04 5.7137609e-03 1.3881326e-03 1.2759566e-03 1.1903048e-03
 1.3345480e-03 6.1666965e-03 1.2716353e-03 1.4884174e-03 1.2865663e-03]
5/5 [==============================] - 0s 997us/step - loss: 0.0194
```

In [41]:
```
# Ftrl optimizer
model_ftrl = Sequential()
model_ftrl.add(Dense(4, input_dim=2, activation='tanh'))  #first layer
model_ftrl.add(Dense(4, activation='tanh')) #second layer
model_ftrl.add(Dense(4, activation='tanh')) #third layer
model_ftrl.add(Dense(4, activation='tanh')) #fourth layer
model_ftrl.add(Dense(4, activation='tanh')) #fifth layer
model_ftrl.add(Dense(1, activation='sigmoid'))
Ftrl = keras.optimizers.Adam(learning_rate=0.01)
model_ftrl.compile(loss='binary_crossentropy', optimizer='Ftrl')
model_ftrl.fit(X, y, batch_size=2, epochs=400)
print(model_ftrl.predict_proba(X).reshape(4*n))
scores_ftrl = model_ftrl.evaluate(X, y) # evaluate the model
```

```
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.49967048 0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.49967048 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.49967048 0.4996705  0.4996705  0.49967048
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.4996705
 0.4996705  0.49967054 0.4996705  0.49967054 0.49967054 0.49967054
 0.4996705  0.4996705  0.4996705  0.4996705  0.49967054 0.49967054
 0.4996705  0.4996705  0.4996705  0.49967054 0.4996705  0.4996705
 0.49967054 0.4996705  0.49967054 0.49967054 0.4996705  0.49967054
 0.49967054 0.4996705  0.4996705  0.49967054 0.49967054 0.4996705
 0.4996705  0.4996705  0.4996705  0.4996705  0.4996705  0.49967054
 0.49967054 0.4996705  0.49967054 0.4996705 ]
5/5 [==============================] - 0s 1ms/step - loss: 0.6931
```

In [42]:
```
scores_tanh
```

Out[42]: 0.020530756562948227

In [43]:
```
scores_adam
```

Out[43]: 0.019363850355148315

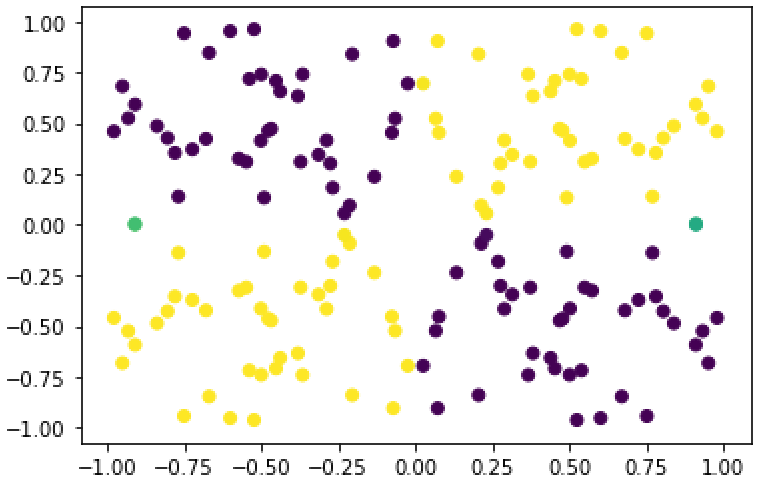
In [44]:
```
scores_ftrl
```

Out[44]: 0.6931473612785339

Plotting the best model (5 layers, 4 neurons, tanh, adam optimizer)

In [45]:
```
plt.scatter(*zip(*X), c=model_adam.predict(X).reshape(4*n))
```

Out[45]:
```
<matplotlib.collections.PathCollection at 0x170c29d6348>
```



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

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

Using Heart Disease data

[https://www.kaggle.com/volodymyrgavrysh/heart-disease?select=heart.csv (https://www.kaggle.com/volodymyrgavrysh/heart-disease?select=heart.csv)](https://www.kaggle.com/volodymyrgavrysh/heart-disease?select=heart.csv)

Using 13 factors to predict diagnosis of heart disease

Attribute Information:
Age: Age
Sex: Sex (1 = male; 0 = female)
ChestPain: Chest pain (typical, asymptotic, nonanginal, nontypical)
RestBP: Resting blood pressure
Chol: Serum cholestoral in mg/dl

Fbs: Fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
RestECG: Resting electrocardiographic results
MaxHR: Maximum heart rate achieved
ExAng: Exercise induced angina (1 = yes; 0 = no)
Oldpeak: ST depression induced by exercise relative to rest
Slope: Slope of the peak exercise ST segment
Ca: Number of major vessels colored by flourosopy (0 - 3)
Thal: (3 = normal; 6 = fixed defect; 7 = reversable defect)
target: AHD - Diagnosis of heart disease (1 = yes; 0 = no)

In [178]:

```
# Load dataset
dataset = pd.read_csv('../data/heart2.csv', index_col=False)
dataset
```

Out[178]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

In [179]:

```
# split into input (X) and output (Y) variables
X = dataset.iloc[:,0:13]
Y = dataset.iloc[:,13]
```

In [183]:
```python
# create models with 2, 4, and 6 layers
model = Sequential()
model.add(Dense(24, input_dim=13, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores2 = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# 4 layers
model = Sequential()
model.add(Dense(24, input_dim=13, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores4 = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# 6 layers
model = Sequential()
model.add(Dense(24, input_dim=13, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores6 = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
y: 0.7129
Epoch 174/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5339 - accurac
y: 0.7327
Epoch 175/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5175 - accurac
y: 0.7294
Epoch 176/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5319 - accurac
y: 0.7162
Epoch 177/400
```

```
31/31 [==============================] - 0s 2ms/step - loss: 0.5366 - accurac
y: 0.6898
Epoch 178/400
31/31 [==============================] - ETA: 0s - loss: 0.5023 - accuracy:
0.80 - 0s 2ms/step - loss: 0.5314 - accuracy: 0.7195
Epoch 179/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5432 - accurac
y: 0.7459
Epoch 180/400
```

In [184]:
```
scores2
```

Out[184]:
```
[0.3897251784801483, 0.7788779139518738]
```

In [185]:
```
scores4
```

Out[185]:
```
[0.30145254731178284, 0.8811880946159363]
```

In [186]:
```
scores6
```

Out[186]:
```
[0.38527220487594604, 0.8448845148086548]
```

4 layer model had lowest loss score. Will explore different number of neurons

In [187]:

```python
# 4 layers, 36 neurons
model = Sequential()
model.add(Dense(36, input_dim=13, activation='tanh'))
model.add(Dense(36, activation='tanh'))
model.add(Dense(36, activation='tanh'))
model.add(Dense(36, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores4_36 = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))


# 4 layers, 48 neurons
model = Sequential()
model.add(Dense(48, input_dim=13, activation='tanh'))
model.add(Dense(48, activation='tanh'))
model.add(Dense(48, activation='tanh'))
model.add(Dense(48, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores4_48 = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Epoch 251/400
31/31 [==============================] - 0s 2ms/step - loss: 0.3401 - accurac
y: 0.8482
Epoch 252/400
31/31 [==============================] - 0s 1ms/step - loss: 0.3601 - accurac
y: 0.8284
Epoch 253/400
31/31 [==============================] - 0s 2ms/step - loss: 0.3693 - accurac
y: 0.8548
Epoch 254/400
31/31 [==============================] - 0s 1ms/step - loss: 0.3539 - accurac
y: 0.8449
Epoch 255/400
31/31 [==============================] - 0s 2ms/step - loss: 0.3589 - accurac
y: 0.8350
Epoch 256/400
31/31 [==============================] - 0s 1ms/step - loss: 0.3515 - accurac
y: 0.8548
Epoch 257/400
31/31 [==============================] - 0s 2ms/step - loss: 0.3400 - accurac
```

In [188]:
```
scores4
```
Out[188]:
```
[0.30145254731178284, 0.8811880946159363]
```
In [189]:
```
scores4_36
```
Out[189]:
```
[0.27431437373161316, 0.8415841460227966]
```
In [190]:
```
scores4_48
```
Out[190]:
```
[0.3702474534511566, 0.8646864891052246]
```

4 layer with 36 neurons still has lowest loss score. Will adjust activation

```
In [191]: # 4 layers, 36 neurons, sigmoid
model = Sequential()
model.add(Dense(36, input_dim=13, activation='sigmoid'))
model.add(Dense(36, activation='sigmoid'))
model.add(Dense(36, activation='sigmoid'))
model.add(Dense(36, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores4_36_sig = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# 4 layers, 36 neurons, softplus
model = Sequential()
model.add(Dense(36, input_dim=13, activation='softplus'))
model.add(Dense(36, activation='softplus'))
model.add(Dense(36, activation='softplus'))
model.add(Dense(36, activation='softplus'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores4_36_soft = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))

# 4 layers, 36 neurons, relu
model = Sequential()
model.add(Dense(36, input_dim=13, activation='relu'))
model.add(Dense(36, activation='relu'))
model.add(Dense(36, activation='relu'))
model.add(Dense(36, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=400, batch_size=10)
# evaluate the model
scores4_36_relu = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
31/31 [==============================] - 0s 1ms/step - loss: 0.5948 - accurac
y: 0.6964
Epoch 18/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5727 - accurac
y: 0.7294
Epoch 19/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5689 - accurac
y: 0.7195
Epoch 20/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5574 - accurac
y: 0.7228
Epoch 21/400
```

```
31/31 [==============================] - 0s 2ms/step - loss: 0.5528 - accurac
y: 0.7261
Epoch 22/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5480 - accurac
y: 0.7261
Epoch 23/400
31/31 [==============================] - 0s 2ms/step - loss: 0.5377 - accurac
y: 0.7294
```

In [192]:
```
scores4_36
```
Out[192]:
```
[0.27431437373161316, 0.8415841460227966]
```

In [193]:
```
scores4_36_sig
```
Out[193]:
```
[0.2899039089679718, 0.8745874762535095]
```

In [194]:
```
scores4_36_soft
```
Out[194]:
```
[0.0405021533370018, 0.9900990128517151]
```

In [195]:
```
scores4_36_relu
```
Out[195]:
```
[0.15602846443653107, 0.933993399143219]
```

Conclusion:

The model with 4 layers, 36 neurons, and activation with `softplus` , was the best performing model with 0.0405 loss and 0.99 accuracy