Neural Networks - intro

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 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

In [3]:
```
num_layers = [1,2,3,4,5]
scores = []
for num_layer in num_layers:

    # build model and evaluate
    score =
    scores.append(score)

# plot scores
```

```
  Cell In[3], line 6
    score =
            ^
SyntaxError: invalid syntax
```

In [4]:
```
!pip3 install tensorflow keras
```

```
Requirement already satisfied: tensorflow in /Users/test/opt/anaconda
3/lib/python3.9/site-packages (2.12.0)
Requirement already satisfied: keras in /Users/test/opt/anaconda3/lib
/python3.9/site-packages (2.12.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=
4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /Users/test/opt/anacon
da3/lib/python3.9/site-packages (from tensorflow) (4.22.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 i
n /Users/test/opt/anaconda3/lib/python3.9/site-packages (from tensorf
low) (0.32.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /Users/test/opt
/anaconda3/lib/python3.9/site-packages (from tensorflow) (0.4.0)
Requirement already satisfied: libclang>=13.0.0 in /Users/test/opt/an
aconda3/lib/python3.9/site-packages (from tensorflow) (16.0.0)
Requirement already satisfied: setuptools in /Users/test/opt/anaconda
3/lib/python3.9/site-packages (from tensorflow) (65.6.3)
Requirement already satisfied: opt-einsum>=2.3.2 in /Users/test/opt/a
naconda3/lib/python3.9/site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /Users/test/opt
/anaconda3/lib/python3.9/site-packages (from tensorflow) (1.53.0)
Requirement already satisfied: jax>=0.3.15 in /Users/test/opt/anacond
a3/lib/python3.9/site-packages (from tensorflow) (0.4.8)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in
/Users/test/opt/anaconda3/lib/python3.9/site-packages (from tensorflo
w) (2.12.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in /Users/test
/opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (2.12.1)
Requirement already satisfied: absl-py>=1.0.0 in /Users/test/opt/anac
onda3/lib/python3.9/site-packages (from tensorflow) (1.4.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /Users/test/opt
/anaconda3/lib/python3.9/site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: six>=1.12.0 in /Users/test/opt/anacond
a3/lib/python3.9/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: numpy<1.24,>=1.22 in /Users/test/opt/a
naconda3/lib/python3.9/site-packages (from tensorflow) (1.23.5)
Requirement already satisfied: typing-extensions>=3.6.6 in /Users/tes
t/opt/anaconda3/lib/python3.9/site-packages (from tensorflow) (4.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /Users/test/opt
/anaconda3/lib/python3.9/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /Users/test/opt/anacond
a3/lib/python3.9/site-packages (from tensorflow) (3.7.0)
Requirement already satisfied: flatbuffers>=2.0 in /Users/test/opt/an
aconda3/lib/python3.9/site-packages (from tensorflow) (23.3.3)
Requirement already satisfied: astunparse>=1.6.0 in /Users/test/opt/a
naconda3/lib/python3.9/site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: termcolor>=1.1.0 in /Users/test/opt/an
aconda3/lib/python3.9/site-packages (from tensorflow) (2.2.0)
Requirement already satisfied: packaging in /Users/test/opt/anaconda3
/lib/python3.9/site-packages (from tensorflow) (22.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /Users/test/opt/
anaconda3/lib/python3.9/site-packages (from astunparse>=1.6.0->tensor
flow) (0.38.4)
Requirement already satisfied: scipy>=1.7 in /Users/test/opt/anaconda
3/lib/python3.9/site-packages (from jax>=0.3.15->tensorflow) (1.10.0)
```

```
Requirement already satisfied: ml-dtypes>=0.0.3 in /Users/test/opt/an
aconda3/lib/python3.9/site-packages (from jax>=0.3.15->tensorflow)
(0.0.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in /Users/test/o
pt/anaconda3/lib/python3.9/site-packages (from tensorboard<2.13,>=2.1
2->tensorflow) (2.17.1)
Requirement already satisfied: werkzeug>=1.0.1 in /Users/test/opt/ana
conda3/lib/python3.9/site-packages (from tensorboard<2.13,>=2.12->ten
sorflow) (2.2.2)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /Use
rs/test/opt/anaconda3/lib/python3.9/site-packages (from tensorboard
<2.13,>=2.12->tensorflow) (1.0.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /Users/test/opt/anaconda3/lib/python3.9/site-packages (from tensor
board<2.13,>=2.12->tensorflow) (0.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /User
s/test/opt/anaconda3/lib/python3.9/site-packages (from tensorboard<2.
13,>=2.12->tensorflow) (1.8.1)
Requirement already satisfied: requests<3,>=2.21.0 in /Users/test/opt
/anaconda3/lib/python3.9/site-packages (from tensorboard<2.13,>=2.12-
>tensorflow) (2.28.1)
Requirement already satisfied: markdown>=2.6.8 in /Users/test/opt/ana
conda3/lib/python3.9/site-packages (from tensorboard<2.13,>=2.12->ten
sorflow) (3.4.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /Users/test/
opt/anaconda3/lib/python3.9/site-packages (from google-auth<3,>=1.6.3
->tensorboard<2.13,>=2.12->tensorflow) (5.3.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /Users/test/o
pt/anaconda3/lib/python3.9/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.13,>=2.12->tensorflow) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in /Users/test/opt/anaco
nda3/lib/python3.9/site-packages (from google-auth<3,>=1.6.3->tensorb
oard<2.13,>=2.12->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /Users/tes
t/opt/anaconda3/lib/python3.9/site-packages (from google-auth-oauthli
b<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in /Users/test
/opt/anaconda3/lib/python3.9/site-packages (from markdown>=2.6.8->ten
sorboard<2.13,>=2.12->tensorflow) (4.11.3)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/tes
t/opt/anaconda3/lib/python3.9/site-packages (from requests<3,>=2.21.0
->tensorboard<2.13,>=2.12->tensorflow) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /Users/test/opt/
anaconda3/lib/python3.9/site-packages (from requests<3,>=2.21.0->tens
orboard<2.13,>=2.12->tensorflow) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /Users/test/o
pt/anaconda3/lib/python3.9/site-packages (from requests<3,>=2.21.0->t
ensorboard<2.13,>=2.12->tensorflow) (1.26.14)
Requirement already satisfied: idna<4,>=2.5 in /Users/test/opt/anacon
da3/lib/python3.9/site-packages (from requests<3,>=2.21.0->tensorboar
d<2.13,>=2.12->tensorflow) (3.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /Users/test/opt/a
naconda3/lib/python3.9/site-packages (from werkzeug>=1.0.1->tensorboa
rd<2.13,>=2.12->tensorflow) (2.1.1)
Requirement already satisfied: zipp>=0.5 in /Users/test/opt/anaconda3
/lib/python3.9/site-packages (from importlib-metadata>=4.4->markdown>
```

```
=2.6.8->tensorboard<2.13,>=2.12->tensorflow) (3.11.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /Users/test/op
t/anaconda3/lib/python3.9/site-packages (from pyasn1-modules>=0.2.1->
google-auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /Users/test/opt/ana
conda3/lib/python3.9/site-packages (from requests-oauthlib>=0.7.0->go
ogle-auth-oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow)
(3.2.2)
```

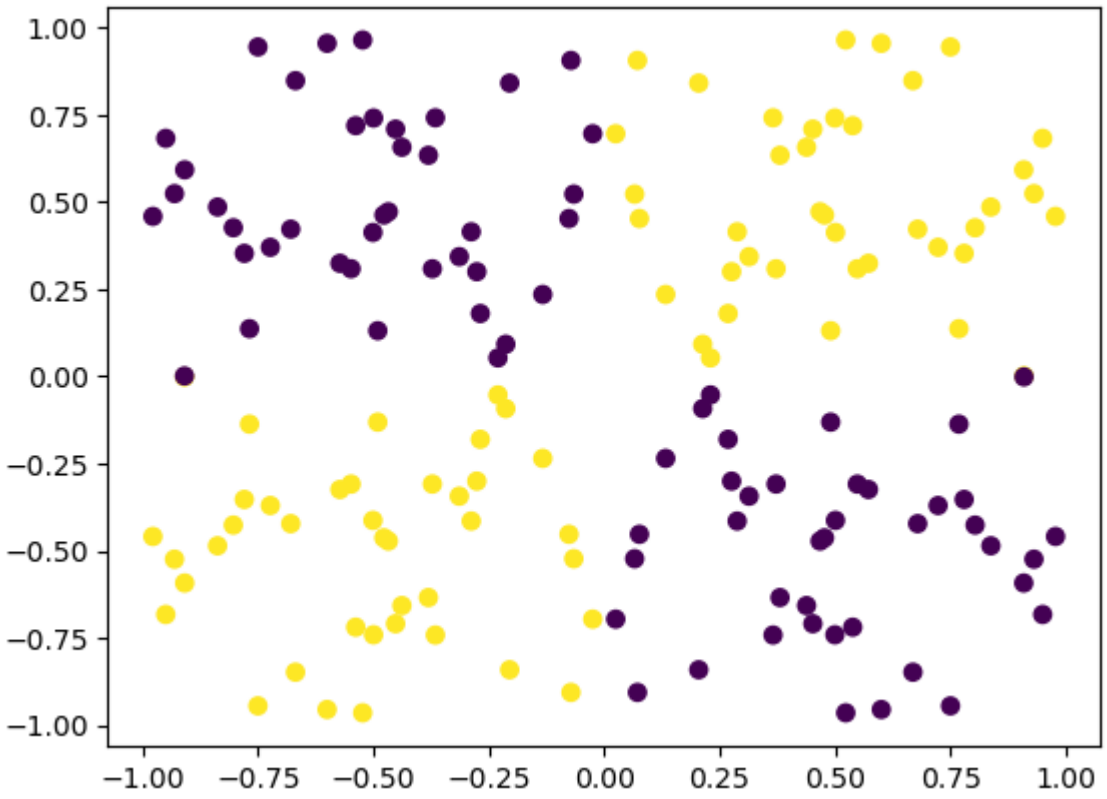
In [5]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD  #Stochastic Gradient Descent

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

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

In [6]:
```
n = 40
xx = np.random.random((n,1))
yy = np.random.random((n,1))
```

In [7]:
```
X = np.array([np.array([xx,-xx,-xx,xx]),np.array([yy,-yy,yy,-yy])]).re
y = np.array([np.ones([2*n]),np.zeros([2*n])]).reshape(4*n)
```

In [9]:
```
plt.scatter(*zip(*X), c=y)
```

Out[9]: <matplotlib.collections.PathCollection at 0x7fbbf08c8df0>



1.1.A One Layer with Two Neurons

In [20]:
```
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
print(model1.predict(X).reshape(4*n))

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

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

```
Epoch 1/400
80/80 [==============================] - 0s 583us/step - loss: 0.6974
Epoch 2/400
80/80 [==============================] - 0s 557us/step - loss: 0.6967
Epoch 3/400
80/80 [==============================] - 0s 550us/step - loss: 0.6961
Epoch 4/400
80/80 [==============================] - 0s 599us/step - loss: 0.6957
Epoch 5/400
80/80 [==============================] - 0s 546us/step - loss: 0.6951
Epoch 6/400
80/80 [==============================] - 0s 545us/step - loss: 0.6946
Epoch 7/400
80/80 [==============================] - 0s 545us/step - loss: 0.6942
Epoch 8/400
80/80 [==============================] - 0s 546us/step - loss: 0.6938
Epoch 9/400
80/80 [==============================] - 0s 568us/step - loss: 0.6935
Epoch 10/400
```

1.1.B Two Layers with Two Neurons

In [19]:
```
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(X).reshape(4*n))

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

plt.scatter(*zip(*X), c=model2.predict(X).reshape(4*n))
```

```
Epoch 1/400
80/80 [==============================] - 1s 658us/step - loss: 0.6958
Epoch 2/400
80/80 [==============================] - 0s 630us/step - loss: 0.6957
Epoch 3/400
80/80 [==============================] - 0s 730us/step - loss: 0.6955
Epoch 4/400
80/80 [==============================] - 0s 615us/step - loss: 0.6952
Epoch 5/400
80/80 [==============================] - 0s 635us/step - loss: 0.6948
Epoch 6/400
80/80 [==============================] - 0s 613us/step - loss: 0.6945
Epoch 7/400
80/80 [==============================] - 0s 595us/step - loss: 0.6944
Epoch 8/400
80/80 [==============================] - 0s 590us/step - loss: 0.6942
Epoch 9/400
80/80 [==============================] - 0s 585us/step - loss: 0.6940
Epoch 10/400
```

1.1.C Three Layers with Two Neurons

In [16]:
```
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(X).reshape(4*n))

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

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

```
Epoch 1/400
80/80 [==============================] - 0s 670us/step - loss: 0.6938
Epoch 2/400
80/80 [==============================] - 0s 622us/step - loss: 0.6939
Epoch 3/400
80/80 [==============================] - 0s 600us/step - loss: 0.6939
Epoch 4/400
80/80 [==============================] - 0s 589us/step - loss: 0.6939
Epoch 5/400
80/80 [==============================] - 0s 581us/step - loss: 0.6939
Epoch 6/400
80/80 [==============================] - 0s 585us/step - loss: 0.6941
Epoch 7/400
80/80 [==============================] - 0s 586us/step - loss: 0.6938
Epoch 8/400
80/80 [==============================] - 0s 585us/step - loss: 0.6940
Epoch 9/400
80/80 [==============================] - 0s 585us/step - loss: 0.6940
Epoch 10/400
```

1.1.D Four Layers with Two Neurons

In [17]:
```
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(X).reshape(4*n))

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

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

```
Epoch 1/400
80/80 [==============================] - 0s 685us/step - loss: 0.6949
Epoch 2/400
80/80 [==============================] - 0s 611us/step - loss: 0.6953
Epoch 3/400
80/80 [==============================] - 0s 611us/step - loss: 0.6951
Epoch 4/400
80/80 [==============================] - 0s 628us/step - loss: 0.6951
Epoch 5/400
80/80 [==============================] - 0s 623us/step - loss: 0.6951
Epoch 6/400
80/80 [==============================] - 0s 626us/step - loss: 0.6952
Epoch 7/400
80/80 [==============================] - 0s 611us/step - loss: 0.6954
Epoch 8/400
80/80 [==============================] - 0s 634us/step - loss: 0.6950
Epoch 9/400
80/80 [==============================] - 0s 634us/step - loss: 0.6954
Epoch 10/400
```

1.1.E Five Layers with Two Neurons

In [18]:
```
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(X).reshape(4*n))

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

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

```
Epoch 1/400
80/80 [==============================] - 0s 703us/step - loss: 0.6964
Epoch 2/400
80/80 [==============================] - 0s 628us/step - loss: 0.6936
Epoch 3/400
80/80 [==============================] - 0s 633us/step - loss: 0.6920
Epoch 4/400
80/80 [==============================] - 0s 631us/step - loss: 0.6900
Epoch 5/400
80/80 [==============================] - 0s 629us/step - loss: 0.6880
Epoch 6/400
80/80 [==============================] - 0s 660us/step - loss: 0.6860
Epoch 7/400
80/80 [==============================] - 0s 628us/step - loss: 0.6838
Epoch 8/400
80/80 [==============================] - 0s 609us/step - loss: 0.6807
Epoch 9/400
80/80 [==============================] - 0s 619us/step - loss: 0.6778
Epoch 10/400
```

1.2.A One Layer with Three Neurons

In [22]:
```
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(X).reshape(4*n))
scores1_3 = model1_3.evaluate(X, y)
```

```
Epoch 1/400
80/80 [==============================] - 0s 607us/step - loss: 0.7456
Epoch 2/400
80/80 [==============================] - 0s 559us/step - loss: 0.7355
Epoch 3/400
80/80 [==============================] - 0s 574us/step - loss: 0.7276
Epoch 4/400
80/80 [==============================] - 0s 553us/step - loss: 0.7211
Epoch 5/400
80/80 [==============================] - 0s 560us/step - loss: 0.7160
Epoch 6/400
80/80 [==============================] - 0s 553us/step - loss: 0.7123
Epoch 7/400
80/80 [==============================] - 0s 540us/step - loss: 0.7090
Epoch 8/400
80/80 [==============================] - 0s 539us/step - loss: 0.7059
Epoch 9/400
80/80 [==============================] - 0s 555us/step - loss: 0.7039
Epoch 10/400
```

1.2.B Two Layers with Three Neurons

In [23]:
```
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(X).reshape(4*n))
scores2_3 = model2_3.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 663us/step - loss: 0.7088
Epoch 2/400
80/80 [==============================] - 0s 603us/step - loss: 0.7017
Epoch 3/400
80/80 [==============================] - 0s 592us/step - loss: 0.6965
Epoch 4/400
80/80 [==============================] - 0s 583us/step - loss: 0.6925
Epoch 5/400
80/80 [==============================] - 0s 592us/step - loss: 0.6889
Epoch 6/400
80/80 [==============================] - 0s 595us/step - loss: 0.6865
Epoch 7/400
80/80 [==============================] - 0s 587us/step - loss: 0.6831
Epoch 8/400
80/80 [==============================] - 0s 576us/step - loss: 0.6804
Epoch 9/400
80/80 [==============================] - 0s 590us/step - loss: 0.6770
Epoch 10/400
```

1.2.C Three Layers with Three Neurons

In [24]:
```
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(X).reshape(4*n))
scores3_3 = model3_3.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 651us/step - loss: 0.6995
Epoch 2/400
80/80 [==============================] - 0s 602us/step - loss: 0.6986
Epoch 3/400
80/80 [==============================] - 0s 587us/step - loss: 0.6948
Epoch 4/400
80/80 [==============================] - 0s 619us/step - loss: 0.6973
Epoch 5/400
80/80 [==============================] - 0s 593us/step - loss: 0.6969
Epoch 6/400
80/80 [==============================] - 0s 607us/step - loss: 0.6951
Epoch 7/400
80/80 [==============================] - 0s 594us/step - loss: 0.6966
Epoch 8/400
80/80 [==============================] - 0s 587us/step - loss: 0.6937
Epoch 9/400
80/80 [==============================] - 0s 600us/step - loss: 0.6960
Epoch 10/400
```

1.2.D Four Layers with Three Neurons

In [25]:
```
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(X).reshape(4*n))
scores4_3 = model4_3.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 685us/step - loss: 0.6973
Epoch 2/400
80/80 [==============================] - 0s 611us/step - loss: 0.6963
Epoch 3/400
80/80 [==============================] - 0s 620us/step - loss: 0.6963
Epoch 4/400
80/80 [==============================] - 0s 611us/step - loss: 0.6957
Epoch 5/400
80/80 [==============================] - 0s 614us/step - loss: 0.6957
Epoch 6/400
80/80 [==============================] - 0s 609us/step - loss: 0.6943
Epoch 7/400
80/80 [==============================] - 0s 613us/step - loss: 0.6942
Epoch 8/400
80/80 [==============================] - 0s 613us/step - loss: 0.6938
Epoch 9/400
80/80 [==============================] - 0s 616us/step - loss: 0.6931
Epoch 10/400
```

1.2.E Five Layers with Three Neurons

In [26]:
```
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(X).reshape(4*n))
scores5_3 = model5_3.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 738us/step - loss: 0.7131
Epoch 2/400
80/80 [==============================] - 0s 635us/step - loss: 0.6921
Epoch 3/400
80/80 [==============================] - 0s 627us/step - loss: 0.6809
Epoch 4/400
80/80 [==============================] - 0s 626us/step - loss: 0.6720
Epoch 5/400
80/80 [==============================] - 0s 671us/step - loss: 0.6645
Epoch 6/400
80/80 [==============================] - 0s 732us/step - loss: 0.6582
Epoch 7/400
80/80 [==============================] - 0s 717us/step - loss: 0.6505
Epoch 8/400
80/80 [==============================] - 0s 718us/step - loss: 0.6443
Epoch 9/400
80/80 [==============================] - 0s 746us/step - loss: 0.6360
Epoch 10/400
```

In [27]:
```
df_3 = np.array([scores1_3, scores2_3, scores3_3, scores4_3, scores5_3
```

Out[27]:
```
array([0.16243769, 0.31481263, 0.02783206, 0.026833  , 0.02379581])
```

1.3.A One Layer with Four Neurons

In [28]:
```
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(X).reshape(4*n))
scores1_4 = model1_4.evaluate(X, y)
```

```
Epoch 1/400
80/80 [==============================] - 0s 694us/step - loss: 0.6979
Epoch 2/400
80/80 [==============================] - 0s 552us/step - loss: 0.6939
Epoch 3/400
80/80 [==============================] - 0s 551us/step - loss: 0.6906
Epoch 4/400
80/80 [==============================] - 0s 569us/step - loss: 0.6879
Epoch 5/400
80/80 [==============================] - 0s 541us/step - loss: 0.6850
Epoch 6/400
80/80 [==============================] - 0s 548us/step - loss: 0.6823
Epoch 7/400
80/80 [==============================] - 0s 541us/step - loss: 0.6795
Epoch 8/400
80/80 [==============================] - 0s 572us/step - loss: 0.6771
Epoch 9/400
80/80 [==============================] - 0s 553us/step - loss: 0.6743
Epoch 10/400
```

1.3.B Two Layers with Four Neurons

In [29]:
```
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(X).reshape(4*n))
scores2_4 = model2_4.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 625us/step - loss: 0.7286
Epoch 2/400
80/80 [==============================] - 0s 588us/step - loss: 0.7109
Epoch 3/400
80/80 [==============================] - 0s 583us/step - loss: 0.6992
Epoch 4/400
80/80 [==============================] - 0s 585us/step - loss: 0.6898
Epoch 5/400
80/80 [==============================] - 0s 568us/step - loss: 0.6838
Epoch 6/400
80/80 [==============================] - 0s 561us/step - loss: 0.6786
Epoch 7/400
80/80 [==============================] - 0s 567us/step - loss: 0.6730
Epoch 8/400
80/80 [==============================] - 0s 571us/step - loss: 0.6684
Epoch 9/400
80/80 [==============================] - 0s 583us/step - loss: 0.6634
Epoch 10/400
```

1.3.C Three Layers with Four Neurons

In [30]:
```
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(X).reshape(4*n))
scores3_4 = model3_4.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 682us/step - loss: 0.7343
Epoch 2/400
80/80 [==============================] - 0s 654us/step - loss: 0.7161
Epoch 3/400
80/80 [==============================] - 0s 695us/step - loss: 0.7050
Epoch 4/400
80/80 [==============================] - 0s 626us/step - loss: 0.6965
Epoch 5/400
80/80 [==============================] - 0s 635us/step - loss: 0.6890
Epoch 6/400
80/80 [==============================] - 0s 635us/step - loss: 0.6821
Epoch 7/400
80/80 [==============================] - 0s 741us/step - loss: 0.6766
Epoch 8/400
80/80 [==============================] - 0s 690us/step - loss: 0.6703
Epoch 9/400
80/80 [==============================] - 0s 691us/step - loss: 0.6626
Epoch 10/400
```

1.3.D Four Layers with Four Neurons

In [31]:
```
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(X).reshape(4*n))
scores4_4 = model4_4.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 687us/step - loss: 0.7146
Epoch 2/400
80/80 [==============================] - 0s 659us/step - loss: 0.7061
Epoch 3/400
80/80 [==============================] - 0s 626us/step - loss: 0.7027
Epoch 4/400
80/80 [==============================] - 0s 615us/step - loss: 0.6993
Epoch 5/400
80/80 [==============================] - 0s 604us/step - loss: 0.6967
Epoch 6/400
80/80 [==============================] - 0s 616us/step - loss: 0.6960
Epoch 7/400
80/80 [==============================] - 0s 622us/step - loss: 0.6945
Epoch 8/400
80/80 [==============================] - 0s 625us/step - loss: 0.6933
Epoch 9/400
80/80 [==============================] - 0s 618us/step - loss: 0.6914
Epoch 10/400
```

1.3.E Five Layers with Four Neurons

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(X).reshape(4*n))
scores5_4 = model5_4.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 722us/step - loss: 0.6961
Epoch 2/400
80/80 [==============================] - 0s 623us/step - loss: 0.6776
Epoch 3/400
80/80 [==============================] - 0s 633us/step - loss: 0.6688
Epoch 4/400
80/80 [==============================] - 0s 628us/step - loss: 0.6594
Epoch 5/400
80/80 [==============================] - 0s 621us/step - loss: 0.6508
Epoch 6/400
80/80 [==============================] - 0s 625us/step - loss: 0.6418
Epoch 7/400
80/80 [==============================] - 0s 624us/step - loss: 0.6343
Epoch 8/400
80/80 [==============================] - 0s 617us/step - loss: 0.6273
Epoch 9/400
80/80 [==============================] - 0s 624us/step - loss: 0.6196
Epoch 10/400
```

In [33]:
```
df_4 = np.array([scores1_4, scores2_4, scores3_4, scores4_4, scores5_4
```

Out[33]:
```
array([0.08339994, 0.04223741, 0.02577953, 0.02063147, 0.02191829])
```

1.4 Compare how tanh, sigmoid, softplus, and relu effect the loss after 400 epochs.

In [36]:
```
df_3
```

Out[36]:
```
array([0.16243769, 0.31481263, 0.02783206, 0.026833  , 0.02379581])
```

In [37]:
```
df_4
```

Out[37]:
```
array([0.08339994, 0.04223741, 0.02577953, 0.02063147, 0.02191829])
```

In the second array above, 1.3.D- the model with 4 layers and 4 neurons- displays the lowest loss.

1.4.A Tanh

In [39]:
```
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(4, activation='tanh')) #fifth 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(X).reshape(4*n))
scores_tanh = model4_4.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 1s 757us/step - loss: 0.6987
Epoch 2/400
80/80 [==============================] - 0s 628us/step - loss: 0.6945
Epoch 3/400
80/80 [==============================] - 0s 621us/step - loss: 0.6909
Epoch 4/400
80/80 [==============================] - 0s 620us/step - loss: 0.6879
Epoch 5/400
80/80 [==============================] - 0s 625us/step - loss: 0.6840
Epoch 6/400
80/80 [==============================] - 0s 625us/step - loss: 0.6801
Epoch 7/400
80/80 [==============================] - 0s 652us/step - loss: 0.6759
Epoch 8/400
80/80 [==============================] - 0s 632us/step - loss: 0.6706
Epoch 9/400
80/80 [==============================] - 0s 628us/step - loss: 0.6640
Epoch 10/400
```

1.4.B Sigmoid

In [40]:
```
model_sig = Sequential()
model_sig.add(Dense(4, input_dim=2, activation='sigmoid'))  #first lay
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(X).reshape(4*n))
scores_sig = model_sig.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 719us/step - loss: 0.7570
Epoch 2/400
80/80 [==============================] - 0s 633us/step - loss: 0.7200
Epoch 3/400
80/80 [==============================] - 0s 613us/step - loss: 0.7048
Epoch 4/400
80/80 [==============================] - 0s 632us/step - loss: 0.6985
Epoch 5/400
80/80 [==============================] - 0s 622us/step - loss: 0.6958
Epoch 6/400
80/80 [==============================] - 0s 615us/step - loss: 0.6951
Epoch 7/400
80/80 [==============================] - 0s 620us/step - loss: 0.6945
Epoch 8/400
80/80 [==============================] - 0s 618us/step - loss: 0.6945
Epoch 9/400
80/80 [==============================] - 0s 621us/step - loss: 0.6941
Epoch 10/400
```

1.4.C Softplus

In [41]:
```
model_soft = Sequential()
model_soft.add(Dense(4, input_dim=2, activation='softplus'))  #first l
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(X).reshape(4*n))
scores_soft = model_soft.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 778us/step - loss: 0.7125
Epoch 2/400
80/80 [==============================] - 0s 784us/step - loss: 0.6965
Epoch 3/400
80/80 [==============================] - 0s 661us/step - loss: 0.6937
Epoch 4/400
80/80 [==============================] - 0s 696us/step - loss: 0.6949
Epoch 5/400
80/80 [==============================] - 0s 660us/step - loss: 0.6937
Epoch 6/400
80/80 [==============================] - 0s 690us/step - loss: 0.6951
Epoch 7/400
80/80 [==============================] - 0s 676us/step - loss: 0.6949
Epoch 8/400
80/80 [==============================] - 0s 685us/step - loss: 0.6954
Epoch 9/400
80/80 [==============================] - 0s 687us/step - loss: 0.6948
Epoch 10/400
```

1.4.D Relu

In [42]:
```
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(X).reshape(4*n))
scores_relu = model_relu.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 0s 708us/step - loss: 0.6931
Epoch 2/400
80/80 [==============================] - 0s 674us/step - loss: 0.6930
Epoch 3/400
80/80 [==============================] - 0s 620us/step - loss: 0.6929
Epoch 4/400
80/80 [==============================] - 0s 621us/step - loss: 0.6927
Epoch 5/400
80/80 [==============================] - 0s 637us/step - loss: 0.6925
Epoch 6/400
80/80 [==============================] - 0s 625us/step - loss: 0.6925
Epoch 7/400
80/80 [==============================] - 0s 626us/step - loss: 0.6923
Epoch 8/400
80/80 [==============================] - 0s 634us/step - loss: 0.6922
Epoch 9/400
80/80 [==============================] - 0s 627us/step - loss: 0.6921
Epoch 10/400
```

In [43]:
```
scores_tanh
```

Out[43]: 0.02507190778851509

In [44]:
```
scores_sig
```

Out[44]: 0.6931575536727905

In [45]:
```
scores_soft
```

Out[45]: 0.3082212507724762

In [46]:
```
scores_relu
```

Out[46]: 0.10025534778833389

Tanh wins.

1.5 Other Optimizers

In [51]:
```
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(X).reshape(4*n))
scores_adam = model_adam.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 1s 892us/step - loss: 0.7278
Epoch 2/400
80/80 [==============================] - 0s 952us/step - loss: 0.7099
Epoch 3/400
80/80 [==============================] - 0s 746us/step - loss: 0.6996
Epoch 4/400
80/80 [==============================] - 0s 814us/step - loss: 0.6953
Epoch 5/400
80/80 [==============================] - 0s 726us/step - loss: 0.6902
Epoch 6/400
80/80 [==============================] - 0s 740us/step - loss: 0.6883
Epoch 7/400
80/80 [==============================] - 0s 707us/step - loss: 0.6792
Epoch 8/400
80/80 [==============================] - 0s 871us/step - loss: 0.6714
Epoch 9/400
80/80 [==============================] - 0s 736us/step - loss: 0.6617
Epoch 10/400
```

.0171 Loss! Nice!!

1.5.B Ftrl Optimizer

In [55]:
```
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(X).reshape(4*n))
scores_ftrl = model_ftrl.evaluate(X, y) # evaluate the model
```

```
Epoch 1/400
80/80 [==============================] - 1s 1ms/step - loss: 0.6932
Epoch 2/400
80/80 [==============================] - 0s 915us/step - loss: 0.6932
Epoch 3/400
80/80 [==============================] - 0s 774us/step - loss: 0.6932
Epoch 4/400
80/80 [==============================] - 0s 901us/step - loss: 0.6932
Epoch 5/400
80/80 [==============================] - 0s 804us/step - loss: 0.6932
Epoch 6/400
80/80 [==============================] - 0s 847us/step - loss: 0.6932
Epoch 7/400
80/80 [==============================] - 0s 772us/step - loss: 0.6932
Epoch 8/400
80/80 [==============================] - 0s 683us/step - loss: 0.6932
Epoch 9/400
80/80 [==============================] - 0s 728us/step - loss: 0.6932
Epoch 10/400
80/80 [==============================] - 0s 670us/step - loss: 0.6932
```

In [52]:
```
scores_tanh
```

Out[52]: 0.02507190778851509

In [53]:
```
scores_adam
```

Out[53]: 0.01713491976261139

In [56]:
```
scores_ftrl
```

Out[56]: 0.6931471228599548

Adam Optimizer wins. Ftrl Optimizer was not good.

2 BYOD - I will use Heart.csv from Programming and Data Managment

In [57]:
```
# Load the dataset
dataset = pd.read_csv('heart.csv', index_col=False)
dataset
```

Out[57]:

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

303 rows × 14 columns

X / Y

In [58]:
```
X = dataset.iloc[:,0:13]
Y = dataset.iloc[:,13]
```

Two Layer Model

In [76]:
```
model = Sequential()
model.add(Dense(24, input_dim=13, activation='tanh'))
model.add(Dense(24, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Create model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit 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], scores2[1]*100))
```

```
Epoch 1/400
31/31 [==============================] - 1s 949us/step - loss: 0.6853
- accuracy: 0.5479
Epoch 2/400
31/31 [==============================] - 0s 910us/step - loss: 0.6703
- accuracy: 0.5710
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6770 -
accuracy: 0.5677
Epoch 4/400
31/31 [==============================] - 0s 876us/step - loss: 0.6521
- accuracy: 0.6139
Epoch 5/400
31/31 [==============================] - 0s 862us/step - loss: 0.6586
- accuracy: 0.5644
Epoch 6/400
31/31 [==============================] - 0s 912us/step - loss: 0.6468
- accuracy: 0.6007
Epoch 7/400
```

86.14% Accuracy

Four Layer Model

In [78]:
```
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'))
# Create model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit 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], scores4[1]*100))
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 0.6478 -
accuracy: 0.6469
Epoch 2/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6010 -
accuracy: 0.6865
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5885 -
accuracy: 0.7228
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5819 -
accuracy: 0.7096
Epoch 5/400
31/31 [==============================] - 0s 945us/step - loss: 0.5805
- accuracy: 0.7195
Epoch 6/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5973 -
accuracy: 0.7063
Epoch 7/400
```

87.46%

Six Layer Model

In [80]:
```
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'))
# Create model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit 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], scores6[1]*100))
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 0.6854 -
accuracy: 0.5710
Epoch 2/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6560 -
accuracy: 0.6139
Epoch 3/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6363 -
accuracy: 0.6601
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6229 -
accuracy: 0.6337
Epoch 5/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6151 -
accuracy: 0.6502
Epoch 6/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6123 -
accuracy: 0.7096
Epoch 7/400
```

84.49%

In [81]:
```
scores2
```

Out[81]: [0.3336445689201355, 0.8613861203193665]

In [82]:
```
scores4
```

Out[82]: [0.2560209035873413, 0.8745874762535095]

In [83]:
```
scores6
```

Out[83]: [0.3952580690383911, 0.8448845148086548]

Four Layer Model is the winner.

.256 Loss and 87.46% Accuracy.

4 layers, 36 neurons

In [85]:
```
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=['
# 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], scores4_36[1]*100))
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 0.6931 -
accuracy: 0.5677
Epoch 2/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6326 -
accuracy: 0.6766
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6216 -
accuracy: 0.6799
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6266 -
accuracy: 0.6502
Epoch 5/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6308 -
accuracy: 0.6700
Epoch 6/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6259 -
accuracy: 0.6799
Epoch 7/400
```

84.16%

4 layers, 48 neurons

In [87]:
```
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=['
# 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], scores4_48[1]*100))
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 0.6670 -
accuracy: 0.6403
Epoch 2/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6170 -
accuracy: 0.6799
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6284 -
accuracy: 0.6568
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5969 -
accuracy: 0.6766
Epoch 5/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6086 -
accuracy: 0.6469
Epoch 6/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6028 -
accuracy: 0.6799
Epoch 7/400
```

86.14%

4 layers with 36 neurons - Sigmoid

In [88]:
```
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=['
# 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], scores4_36_sig[1]*100))
```

```
Epoch 1/400
31/31 [==============================] - 2s 1ms/step - loss: 0.7383 -
accuracy: 0.5116
Epoch 2/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6912 -
accuracy: 0.5446
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6896 -
accuracy: 0.5446
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6907 -
accuracy: 0.5446
Epoch 5/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6907 -
accuracy: 0.5446
Epoch 6/400
31/31 [==============================] - 0s 932us/step - loss: 0.6878
- accuracy: 0.5446
Epoch 7/400
```

90.76%!

4 layers, 36 neurons - Softplus

In [91]:
```
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=['
# 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], scores4_36_soft[1]*100
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 4.3980 -
accuracy: 0.5611
Epoch 2/400
31/31 [==============================] - 0s 2ms/step - loss: 0.7761 -
accuracy: 0.6337
Epoch 3/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6492 -
accuracy: 0.6601
Epoch 4/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6602 -
accuracy: 0.6568
Epoch 5/400
31/31 [==============================] - 0s 2ms/step - loss: 0.6911 -
accuracy: 0.6337
Epoch 6/400
31/31 [==============================] - 0s 1ms/step - loss: 0.5986 -
accuracy: 0.6766
Epoch 7/400
```

86.47%

4 layers, 36 neurons - Relu

In [92]:
```
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=['
# 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], scores4_36_relu[1]*100
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 1.4347 -
accuracy: 0.5644
Epoch 2/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6742 -
accuracy: 0.6436
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6176 -
accuracy: 0.6667
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6442 -
accuracy: 0.6700
Epoch 5/400
31/31 [==============================] - 0s 964us/step - loss: 0.7821
- accuracy: 0.6073
Epoch 6/400
31/31 [==============================] - 0s 993us/step - loss: 0.5818
- accuracy: 0.7030
Epoch 7/400
```

87.79%

In [93]:
```
scores4
```

Out[93]: [0.2560209035873413, 0.8745874762535095]

In [94]:
```
scores4_36
```

Out[94]: [0.35283219814300537, 0.8415841460227966]

In [95]:
```
scores4_48
```

Out[95]: [0.335470587015152, 0.8613861203193665]

Four Layer (again) is the winner.

.256 Loss and 87.46% Accuracy.

BOOM

In [98]:
```
# 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'))
# Create model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit 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], scores4_36_sig[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'))
# Create model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit 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], scores4_36_soft[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'))
# Create model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit 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], scores4_36_relu[1]*100
```

```
Epoch 1/400
31/31 [==============================] - 1s 1ms/step - loss: 0.7084 -
accuracy: 0.4983
Epoch 2/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6904 -
accuracy: 0.5446
Epoch 3/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6895 -
accuracy: 0.5446
Epoch 4/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6902 -
```

```
accuracy: 0.5446
Epoch 5/400
31/31 [==============================] - 0s 1ms/step - loss: 0.6896 -
accuracy: 0.5446
Epoch 6/400
31/31 [==============================] - 0s 957us/step - loss: 0.6896
- accuracy: 0.5446
Epoch 7/400
```

In [99]:
```
scores4_36
```

Out[99]: [0.35283219814300537, 0.8415841460227966]

In [100]:
```
scores4_36_sig
```

Out[100]: [0.2684692144393921, 0.9009901285171509]

In [101]:
```
scores4_36_soft
```

Out[101]: [0.05914897099137306, 0.9834983348846436]

In [102]:
```
scores4_36_relu
```

Out[102]: [0.18851086497306824, 0.933993399143219]

The winner, overall, is 4 Layers with 36 Neurons and Softplus.

.059 Loss and 98.34% Accuracy.

In [ ]: