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/)

In [34]:
```
!pip3 install tensorflow keras
```

```
Requirement already satisfied: tensorflow in /Users/obelisk/anaconda3
/lib/python3.10/site-packages (2.13.0)
Requirement already satisfied: keras in /Users/obelisk/anaconda3/lib/
python3.10/site-packages (2.13.1)
Requirement already satisfied: tensorboard<2.14,>=2.13 in /Users/obel
isk/anaconda3/lib/python3.10/site-packages (from tensorflow) (2.13.0)
```

```
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 i
n /Users/obelisk/anaconda3/lib/python3.10/site-packages (from tensorf
low) (0.32.0)
Requirement already satisfied: six>=1.12.0 in /Users/obelisk/anaconda
3/lib/python3.10/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /Users/obelisk/an
aconda3/lib/python3.10/site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: termcolor>=1.1.0 in /Users/obelisk/ana
conda3/lib/python3.10/site-packages (from tensorflow) (2.3.0)
Requirement already satisfied: astunparse>=1.6.0 in /Users/obelisk/an
aconda3/lib/python3.10/site-packages (from tensorflow) (1.6.3)
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/obelisk/anaconda
3/lib/python3.10/site-packages (from tensorflow) (4.23.4)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /Users/obelisk/
anaconda3/lib/python3.10/site-packages (from tensorflow) (1.56.0)
Requirement already satisfied: wrapt>=1.11.0 in /Users/obelisk/anacon
da3/lib/python3.10/site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: packaging in /Users/obelisk/anaconda3/
lib/python3.10/site-packages (from tensorflow) (22.0)
Requirement already satisfied: numpy<=1.24.3,>=1.22 in /Users/obelisk
/anaconda3/lib/python3.10/site-packages (from tensorflow) (1.23.5)
Requirement already satisfied: typing-extensions<4.6.0,>=3.6.6 in /Us
ers/obelisk/anaconda3/lib/python3.10/site-packages (from tensorflow)
(4.4.0)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /Users/obelisk/
anaconda3/lib/python3.10/site-packages (from tensorflow) (0.4.0)
Requirement already satisfied: h5py>=2.9.0 in /Users/obelisk/anaconda
3/lib/python3.10/site-packages (from tensorflow) (3.7.0)
Requirement already satisfied: absl-py>=1.0.0 in /Users/obelisk/anaco
nda3/lib/python3.10/site-packages (from tensorflow) (1.4.0)
Requirement already satisfied: setuptools in /Users/obelisk/anaconda3
/lib/python3.10/site-packages (from tensorflow) (65.6.3)
Requirement already satisfied: tensorflow-estimator<2.14,>=2.13.0 in
/Users/obelisk/anaconda3/lib/python3.10/site-packages (from tensorflo
w) (2.13.0)
Requirement already satisfied: flatbuffers>=23.1.21 in /Users/obelisk
/anaconda3/lib/python3.10/site-packages (from tensorflow) (23.5.26)
Requirement already satisfied: libclang>=13.0.0 in /Users/obelisk/ana
conda3/lib/python3.10/site-packages (from tensorflow) (16.0.0)
Requirement already satisfied: google-pasta>=0.1.1 in /Users/obelisk/
anaconda3/lib/python3.10/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /Users/obelisk/a
naconda3/lib/python3.10/site-packages (from astunparse>=1.6.0->tensor
flow) (0.38.4)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in /Use
rs/obelisk/anaconda3/lib/python3.10/site-packages (from tensorboard<2
.14,>=2.13->tensorflow) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in /Users/obelisk/anac
onda3/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13->ten
```

```
sorflow) (3.4.1)
Requirement already satisfied: werkzeug>=1.0.1 in /Users/obelisk/anac
onda3/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13->ten
sorflow) (2.2.2)
Requirement already satisfied: requests<3,>=2.21.0 in /Users/obelisk/
anaconda3/lib/python3.10/site-packages (from tensorboard<2.14,>=2.13-
>tensorflow) (2.28.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /Users/obelisk/anaconda3/lib/python3.10/site-packages (from tensor
board<2.14,>=2.13->tensorflow) (0.7.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /Users/obelis
k/anaconda3/lib/python3.10/site-packages (from tensorboard<2.14,>=2.1
3->tensorflow) (2.21.0)
Requirement already satisfied: urllib3<2.0 in /Users/obelisk/anaconda
3/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorboa
rd<2.14,>=2.13->tensorflow) (1.26.14)
Requirement already satisfied: rsa<5,>=3.1.4 in /Users/obelisk/anacon
da3/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorb
oard<2.14,>=2.13->tensorflow) (4.9)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /Users/obelis
k/anaconda3/lib/python3.10/site-packages (from google-auth<3,>=1.6.3-
>tensorboard<2.14,>=2.13->tensorflow) (0.2.8)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /Users/obeli
sk/anaconda3/lib/python3.10/site-packages (from google-auth<3,>=1.6.3
->tensorboard<2.14,>=2.13->tensorflow) (5.3.1)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /Users/obe
lisk/anaconda3/lib/python3.10/site-packages (from google-auth-oauthli
b<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow) (1.3.1)
Requirement already satisfied: certifi>=2017.4.17 in /Users/obelisk/a
naconda3/lib/python3.10/site-packages (from requests<3,>=2.21.0->tens
orboard<2.14,>=2.13->tensorflow) (2022.12.7)
Requirement already satisfied: charset-normalizer<3,>=2 in /Users/obe
lisk/anaconda3/lib/python3.10/site-packages (from requests<3,>=2.21.0
->tensorboard<2.14,>=2.13->tensorflow) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /Users/obelisk/anacond
a3/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorboar
d<2.14,>=2.13->tensorflow) (3.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /Users/obelisk/an
aconda3/lib/python3.10/site-packages (from werkzeug>=1.0.1->tensorboa
rd<2.14,>=2.13->tensorflow) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /Users/obelisk
/anaconda3/lib/python3.10/site-packages (from pyasn1-modules>=0.2.1->
google-auth<3,>=1.6.3->tensorboard<2.14,>=2.13->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /Users/obelisk/anac
onda3/lib/python3.10/site-packages (from requests-oauthlib>=0.7.0->go
ogle-auth-oauthlib<1.1,>=0.5->tensorboard<2.14,>=2.13->tensorflow) (3
.2.2)
```

In [2]:
```
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
```

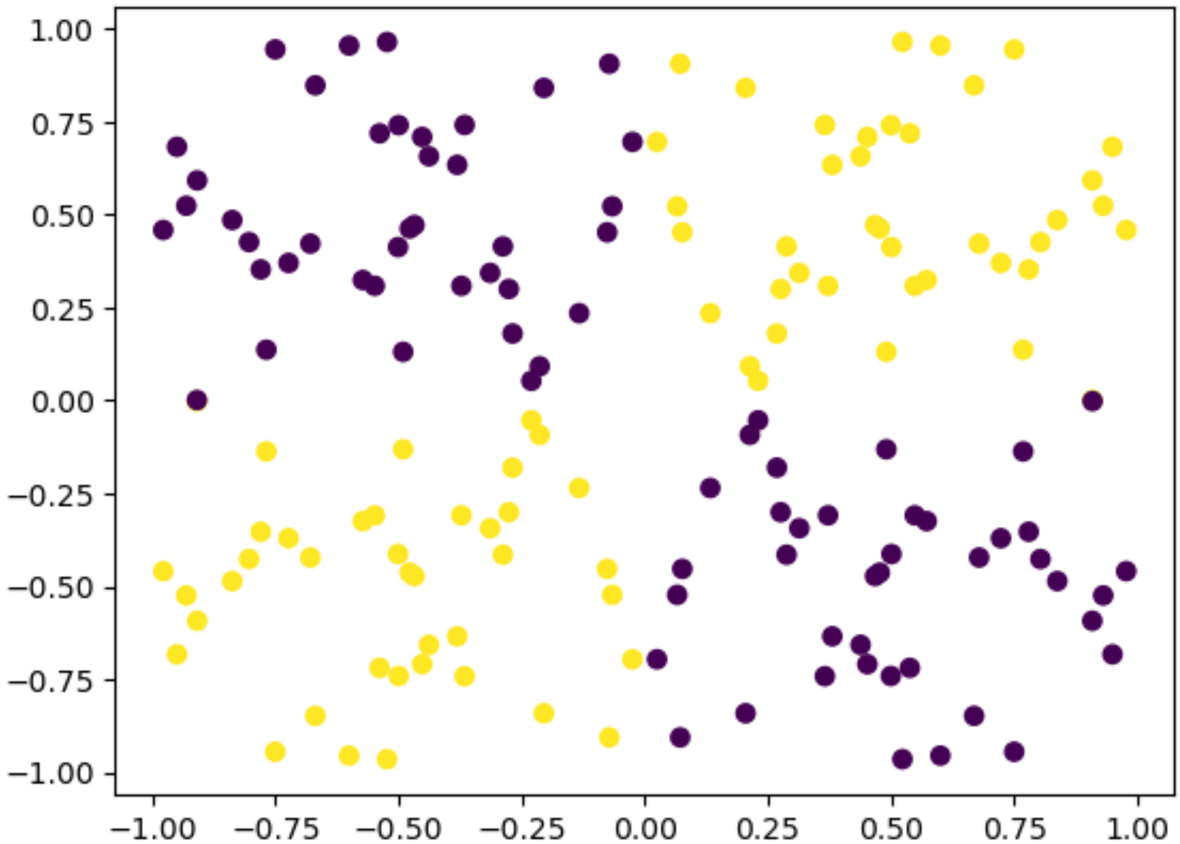
```
2023-07-16 14:23:05.935864: I tensorflow/core/platform/cpu_feature_gu
ard.cc:182] This TensorFlow binary is optimized to use available CPU
instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
```

In [3]:
```
n = 40
xx = np.random.random((n,1))
yy = np.random.random((n,1))
```

In [4]:
```
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 [5]:
```
plt.scatter(*zip(*X), c=y)
```

Out[5]: <matplotlib.collections.PathCollection at 0x7fda2b1930d0>



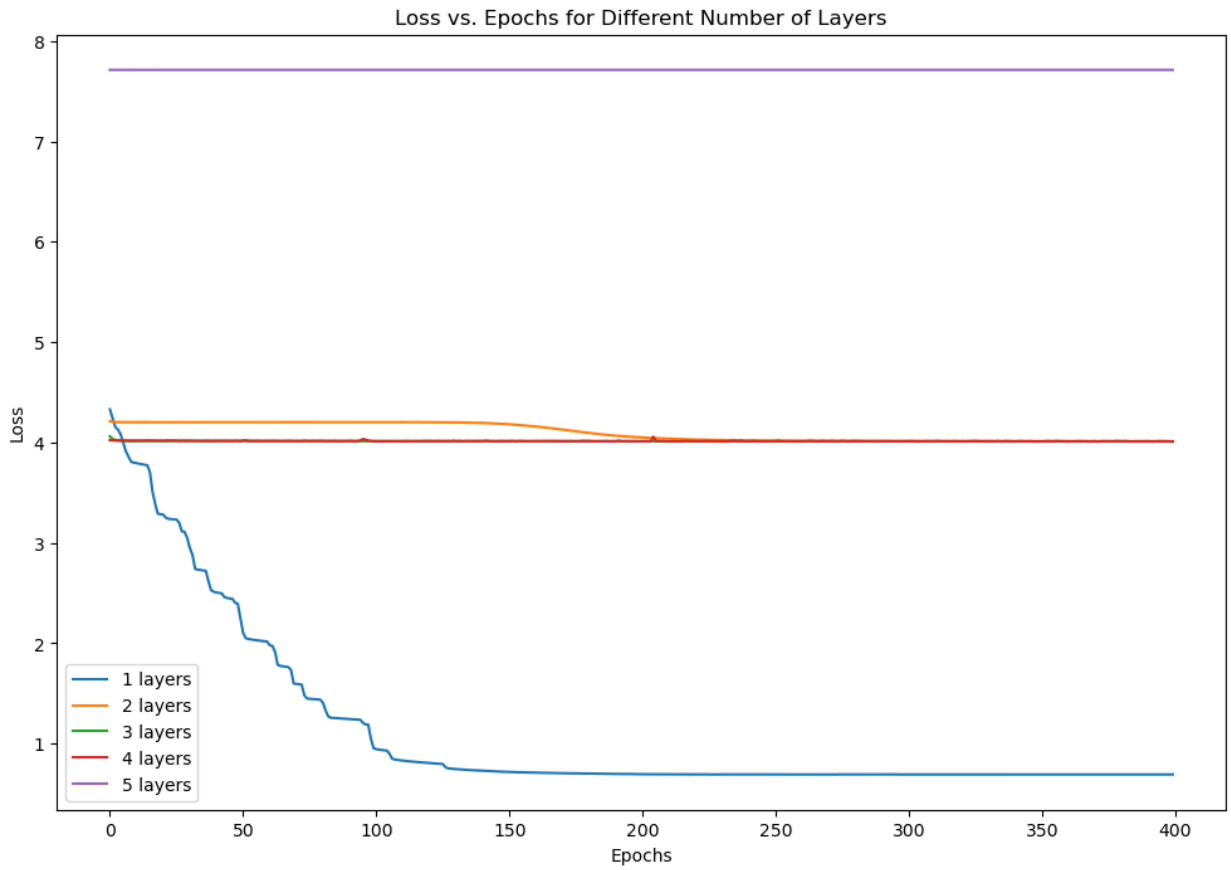
In [6]:

```
num_layers = [1,2,3,4,5]
scores = []
plt.figure(figsize=(12, 8))
model = Sequential()
model.add(Dense(2, input_dim=2, activation='tanh'))
sgd = SGD(learning_rate=0.1)

for num_layer in num_layers:
    # for the first iteration, the model will only have the base layer
    if num_layer > 1:
        model.add(Dense(2, activation='tanh'))

    model.compile(loss='binary_crossentropy', optimizer='Adam')

    history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

    plt.plot(history.history['loss'], label=f'{num_layer} layers')

    model.summary()

    score = model.evaluate(X, y)
    scores.append(score)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs for Different Number of Layers')
plt.legend()
plt.show()
printout = [item for sublist in zip(num_layers, scores) for item in su
print("2 Neurons per layer:", printout)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

=================================================================
Total params: 6 (24.00 Byte)
Trainable params: 6 (24.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.6931
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 2)                 6
```

```
=================================================================
Total params: 12 (48.00 Byte)
Trainable params: 12 (48.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.0120
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 2)                 6

 dense_2 (Dense)             (None, 2)                 6

=================================================================
Total params: 18 (72.00 Byte)
Trainable params: 18 (72.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.0095
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 2)                 6

 dense_2 (Dense)             (None, 2)                 6

 dense_3 (Dense)             (None, 2)                 6

=================================================================
Total params: 24 (96.00 Byte)
Trainable params: 24 (96.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.0093
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 6

 dense_1 (Dense)             (None, 2)                 6

 dense_2 (Dense)             (None, 2)                 6
```

```
 dense_3 (Dense)             (None, 2)                 6

 dense_4 (Dense)             (None, 2)                 6

=================================================================
Total params: 30 (120.00 Byte)
Trainable params: 30 (120.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 7.7125
```




```
2 Neurons per layer: [1, 0.693149745464325, 2, 4.012002944946289, 3, 4.009472846984863, 4, 4.009335041046143, 5, 7.7124738693237305]
```

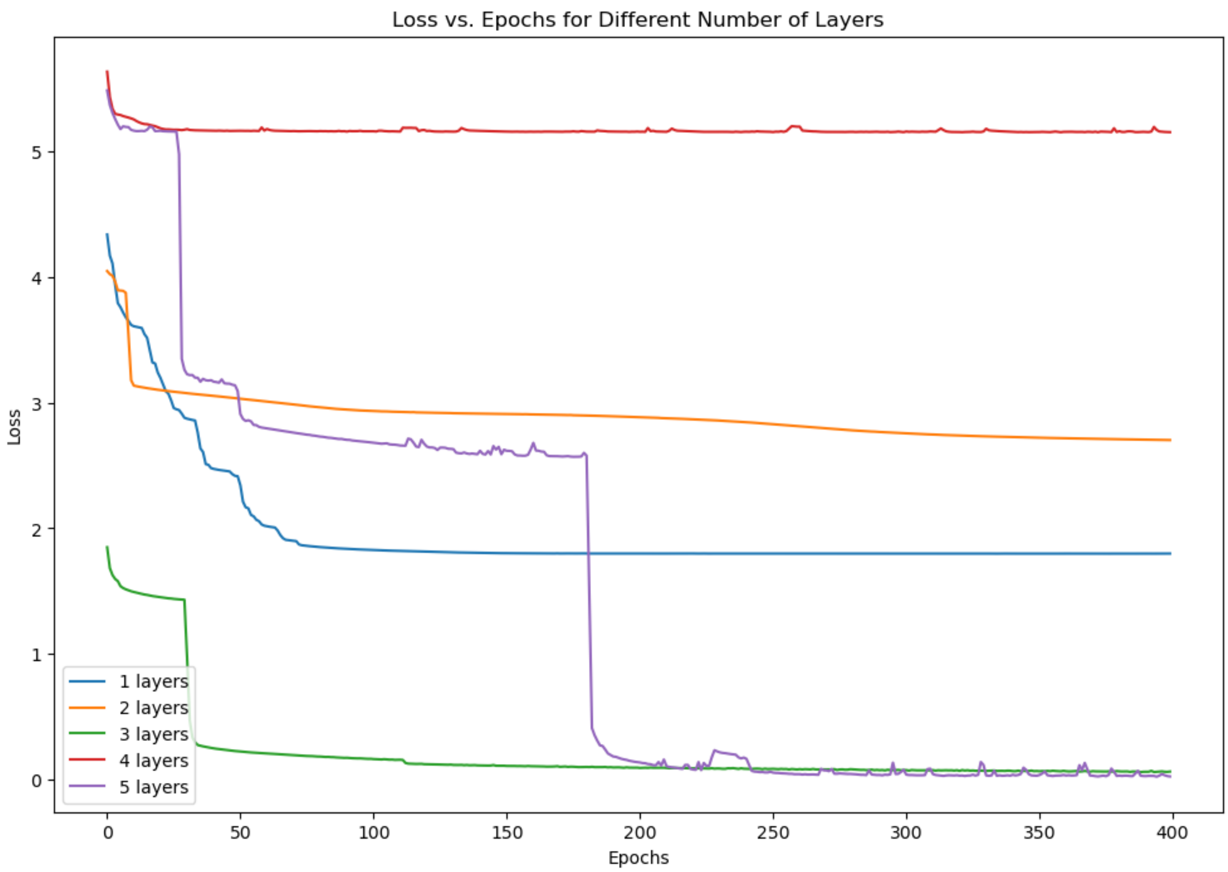
In [7]:

```
scores2 = []
plt.figure(figsize=(12, 8))
model = Sequential()
model.add(Dense(3, input_dim=2, activation='tanh'))

for num_layer in num_layers:

    if num_layer > 1:
        model.add(Dense(3, activation='tanh'))

    model.compile(loss='binary_crossentropy', optimizer='Adam')

    history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

    plt.plot(history.history['loss'], label=f'{num_layer} layers')

    model.summary()

    score = model.evaluate(X, y)
    scores2.append(score)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs for Different Number of Layers')
plt.legend()
plt.show()
printout2 = [item for sublist in zip(num_layers, scores2) for item in
print("2 Neurons per layer:", printout)
print("3 Neurons per layer:", printout2)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 3)                 9

=================================================================
Total params: 9 (36.00 Byte)
Trainable params: 9 (36.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 1.7975
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 3)                 9

 dense_6 (Dense)             (None, 3)                 12
```

```
=================================================================
Total params: 21 (84.00 Byte)
Trainable params: 21 (84.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 2ms/step - loss: 2.7003
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 3)                 9

 dense_6 (Dense)             (None, 3)                 12

 dense_7 (Dense)             (None, 3)                 12

=================================================================
Total params: 33 (132.00 Byte)
Trainable params: 33 (132.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0573
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 3)                 9

 dense_6 (Dense)             (None, 3)                 12

 dense_7 (Dense)             (None, 3)                 12

 dense_8 (Dense)             (None, 3)                 12

=================================================================
Total params: 45 (180.00 Byte)
Trainable params: 45 (180.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 5.1497
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 3)                 9

 dense_6 (Dense)             (None, 3)                 12

 dense_7 (Dense)             (None, 3)                 12
```

```
 dense_8 (Dense)             (None, 3)                 12

 dense_9 (Dense)             (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0218
```




```
2 Neurons per layer: [1, 0.693149745464325, 2, 4.012002944946289, 3,
4.009472846984863, 4, 4.009335041046143, 5, 7.7124738693237305]
3 Neurons per layer: [1, 1.797528862953186, 2, 2.700294017791748, 3,
0.057295072823762894, 4, 5.149728775024414, 5, 0.021826880052685738]
```

Over all the 3 neuron layers appear to perform generally better to the 2 neuron layers. The loss scores are lower than the 2 neuron layers. The best performance so far is 5 layer 3 neuron model with a loss of 0.022.

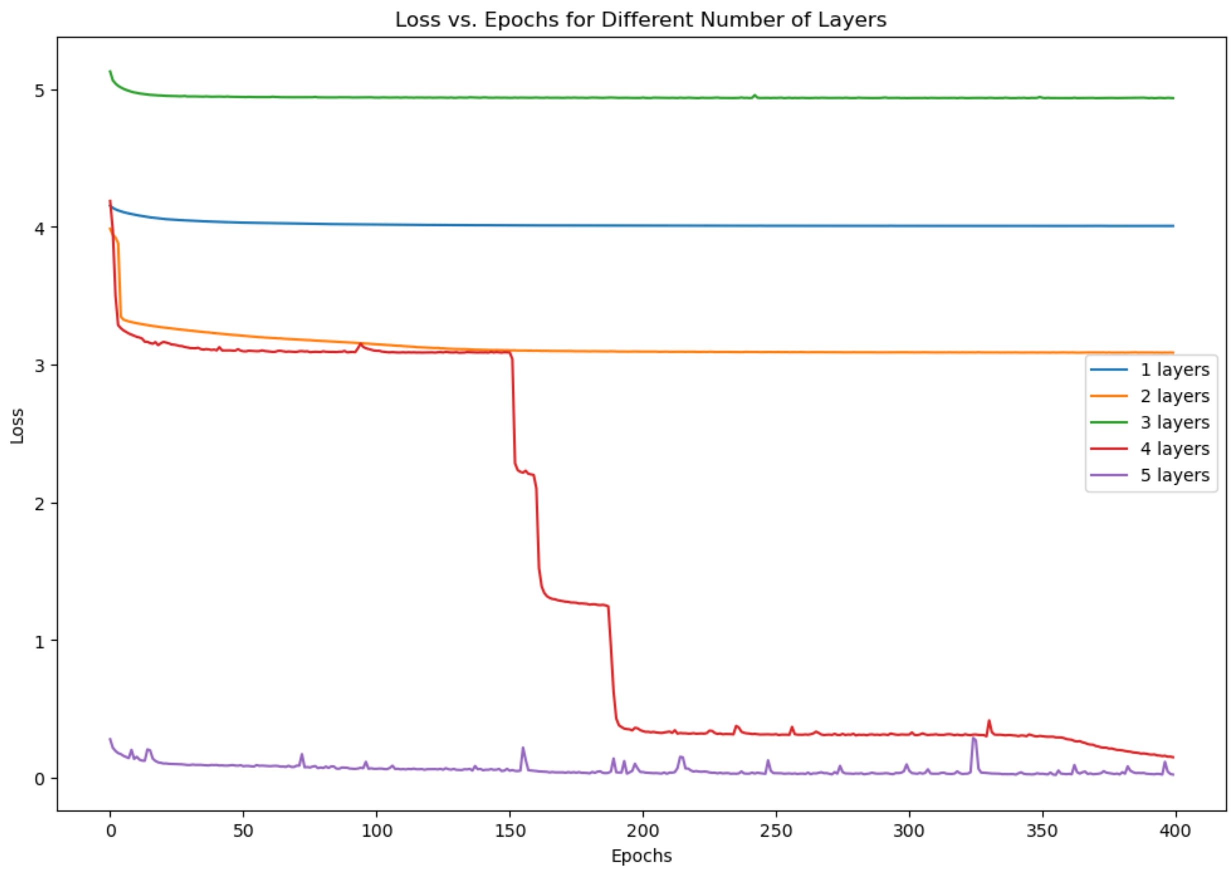
In [8]:

```
scores3 = []
plt.figure(figsize=(12, 8))
model = Sequential()
model.add(Dense(4, input_dim=2, activation='tanh'))

for num_layer in num_layers:

    if num_layer > 1:
        model.add(Dense(4, activation='tanh'))

    model.compile(loss='binary_crossentropy', optimizer='Adam')

    history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

    plt.plot(history.history['loss'], label=f'{num_layer} layers')

    model.summary()

    score = model.evaluate(X, y)
    scores3.append(score)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs for Different Number of Layers')
plt.legend()
plt.show()
printout3 = [item for sublist in zip(num_layers, scores3) for item in
print("2 Neurons per layer:", printout)
print("3 Neurons per layer:", printout2)
print("4 Neurons per layer:", printout3)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 4)                 12

=================================================================
Total params: 12 (48.00 Byte)
Trainable params: 12 (48.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.0063
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 4)                 12

 dense_11 (Dense)            (None, 4)                 20
```

```
=================================================================
Total params: 32 (128.00 Byte)
Trainable params: 32 (128.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 3.0856
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 4)                 12

 dense_11 (Dense)            (None, 4)                 20

 dense_12 (Dense)            (None, 4)                 20

=================================================================
Total params: 52 (208.00 Byte)
Trainable params: 52 (208.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.9333
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 4)                 12

 dense_11 (Dense)            (None, 4)                 20

 dense_12 (Dense)            (None, 4)                 20

 dense_13 (Dense)            (None, 4)                 20


=================================================================
Total params: 72 (288.00 Byte)
Trainable params: 72 (288.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.1448
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 4)                 12

 dense_11 (Dense)            (None, 4)                 20

 dense_12 (Dense)            (None, 4)                 20
```

```
 dense_13 (Dense)            (None, 4)                 20

 dense_14 (Dense)            (None, 4)                 20

=================================================================
Total params: 92 (368.00 Byte)
Trainable params: 92 (368.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0231
```




```
2 Neurons per layer: [1, 0.693149745464325, 2, 4.012002944946289, 3,
4.009472846984863, 4, 4.009335041046143, 5, 7.7124738693237305]
3 Neurons per layer: [1, 1.797528862953186, 2, 2.700294017791748, 3,
0.057295072823762894, 4, 5.149728775024414, 5, 0.021826880052685738]
4 Neurons per layer: [1, 4.006335735321045, 2, 3.0856049060821533, 3,
4.933342456817627, 4, 0.14478500187397003, 5, 0.023132342845201492]
```

Generally similar results to the 3 neuron models here. Seems to have more noise in the results of the loss scores.

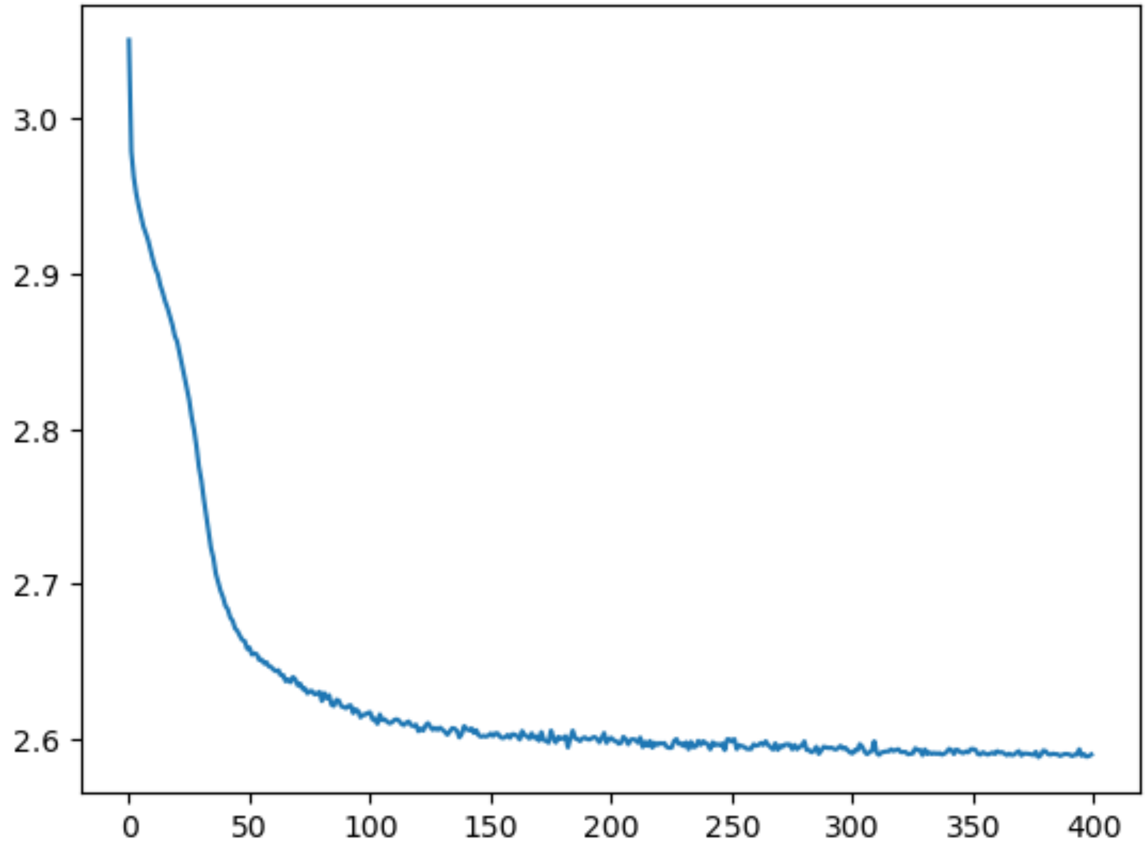
Best results are 3 neurons and 5 layers from this iteration with a loss score of 0.022. Even after switching to the Adam optimizer from SGD I am seeing somewhat erratic results between each execution of the for-loop.

In [9]:

```
# tanh

model = Sequential()

model.add(Dense(3, input_dim=2, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))

model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Tanh')

model.summary()

score_tanh = model.evaluate(X, y)

print("Tanh Loss Score:", score_tanh)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 3)                 9

 dense_16 (Dense)            (None, 3)                 12

 dense_17 (Dense)            (None, 3)                 12

 dense_18 (Dense)            (None, 3)                 12

 dense_19 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

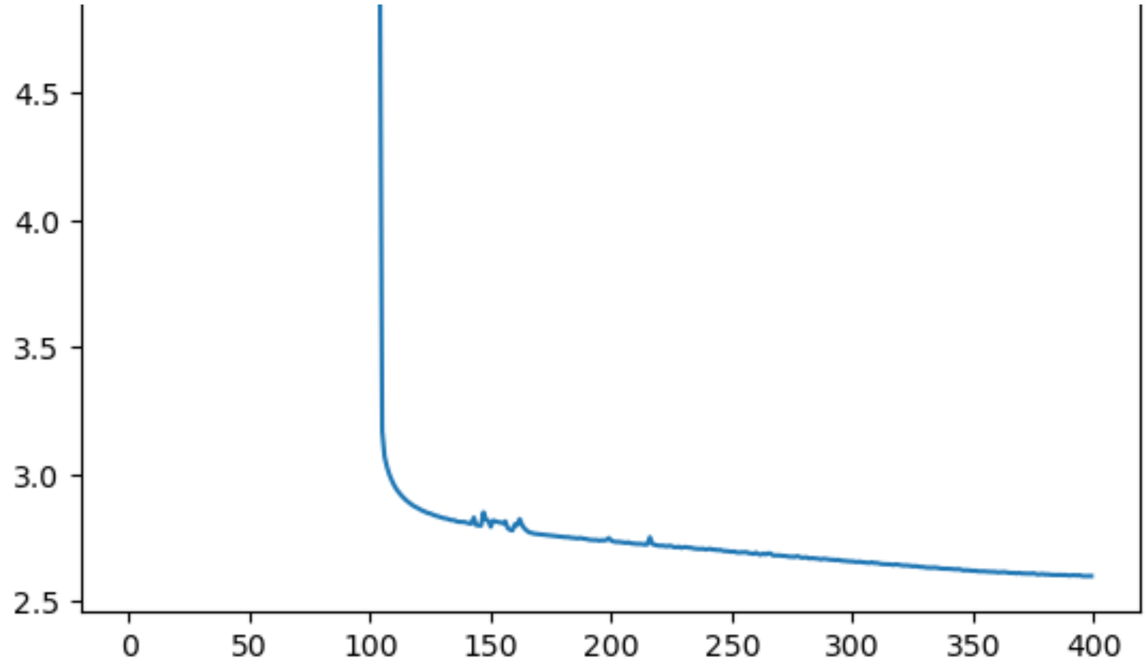
```
5/5 [==============================] - 0s 1ms/step - loss: 2.5861
Tanh Loss Score: 2.5861058235168457
```



In [10]:

```
# sigmoid

model = Sequential()

model.add(Dense(3, input_dim=2, activation='sigmoid'))
model.add(Dense(3, activation='tanh')) # If I have more than 1 sigmoic
model.add(Dense(3, activation='tanh')) # If I have all sigmoid layers
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))


model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Sigmoid')

model.summary()

score_sigmoid = model.evaluate(X, y)

print("Sigmoid Loss Score:", score_sigmoid)
```

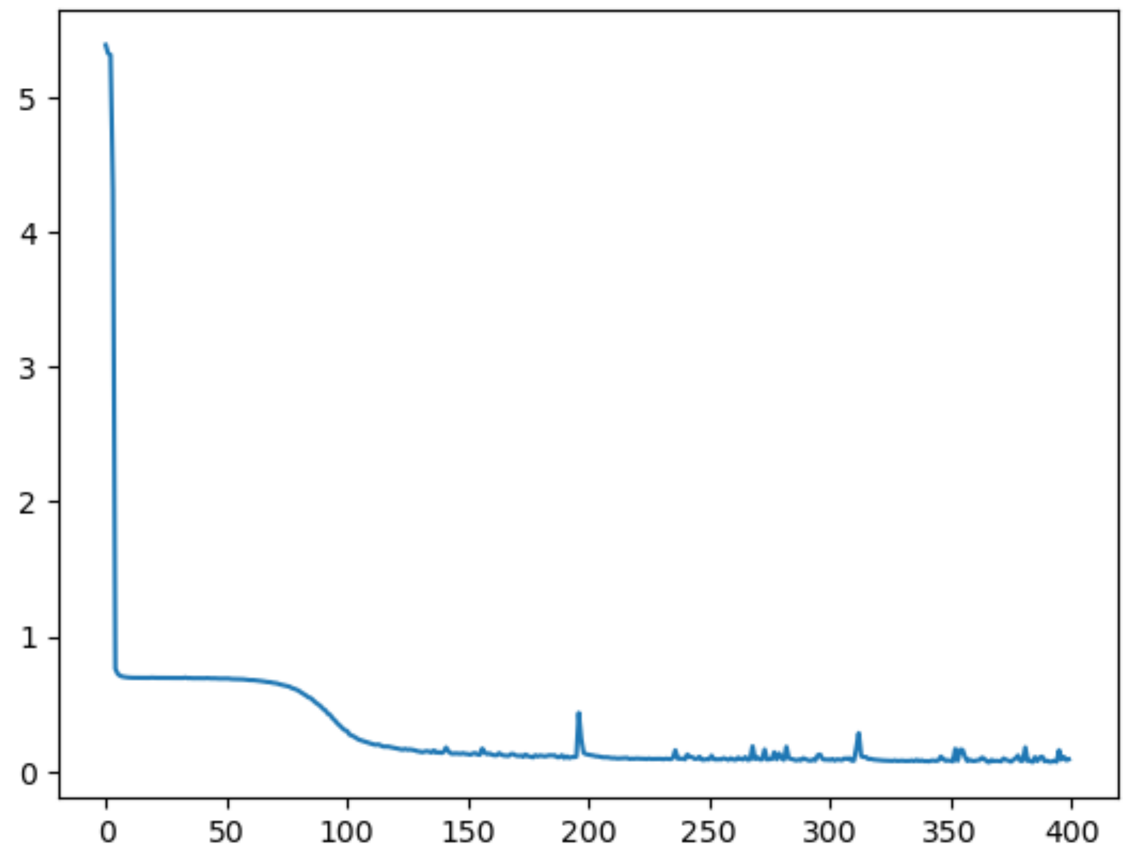
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 3)                 9

 dense_21 (Dense)            (None, 3)                 12

 dense_22 (Dense)            (None, 3)                 12

 dense_23 (Dense)            (None, 3)                 12

 dense_24 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 2.5952
Sigmoid Loss Score: 2.595248222351074
```



In [12]:
```
# softplus

model = Sequential()

model.add(Dense(3, input_dim=2, activation='softplus'))
model.add(Dense(3, activation='softplus'))
model.add(Dense(3, activation='softplus'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='softplus'))

model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Softplus')

model.summary()

score_soft_plus = model.evaluate(X, y)

print("Softplus Loss Score:", score_soft_plus)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_30 (Dense)            (None, 3)                 9

 dense_31 (Dense)            (None, 3)                 12
```

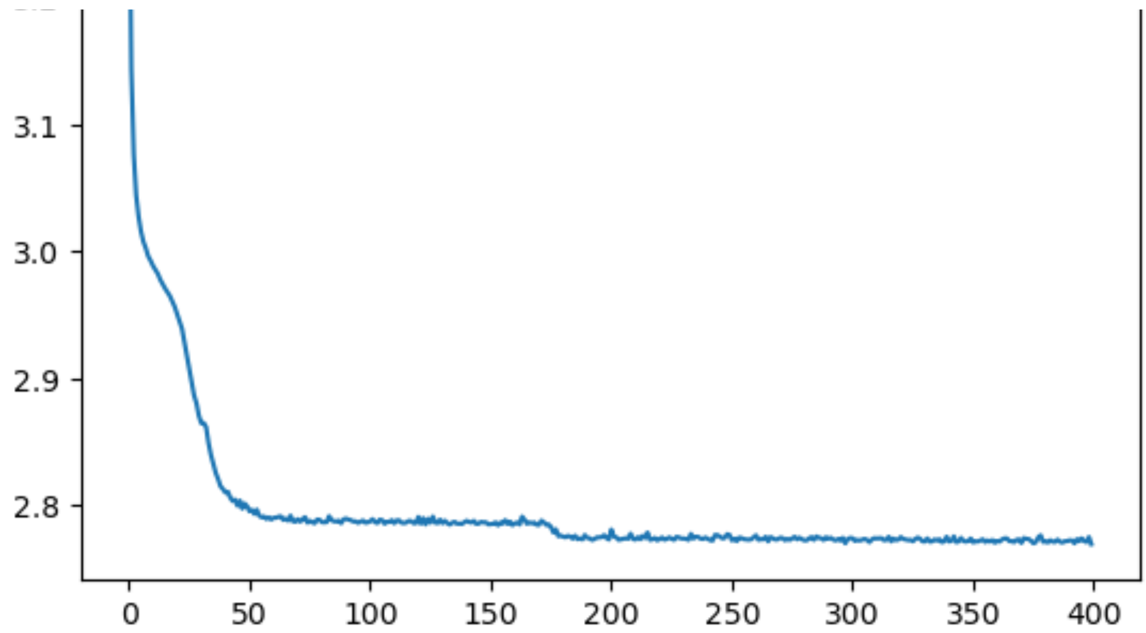
```
 dense_32 (Dense)            (None, 3)                 12

 dense_33 (Dense)            (None, 3)                 12

 dense_34 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0670
Softplus Loss Score: 0.06703799217939377
```



In [14]:

```
# relu

model = Sequential()

model.add(Dense(3, input_dim=2, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(3, activation='relu'))
model.add(Dense(3, activation='tanh')) # If I have 4 relu weird things
model.add(Dense(3, activation='tanh')) # If I have all relu weird thin

model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Relu')

model.summary()

score_relu = model.evaluate(X, y)

print("Relu Loss Score:", score_relu)
```

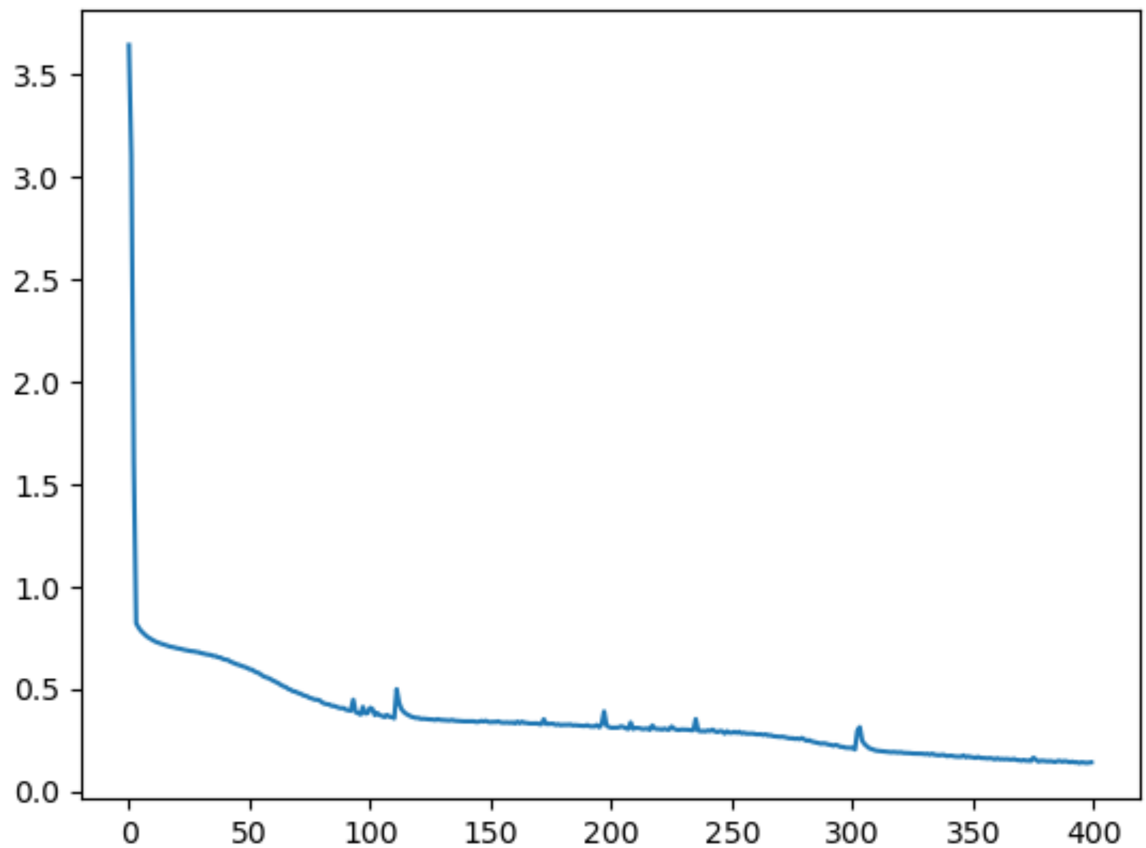
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_40 (Dense)            (None, 3)                 9

 dense_41 (Dense)            (None, 3)                 12

 dense_42 (Dense)            (None, 3)                 12

 dense_43 (Dense)            (None, 3)                 12

 dense_44 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 2.7737
Relu Loss Score: 2.773733139038086
```



In [15]:
```
# softmax

model = Sequential()

model.add(Dense(3, input_dim=2, activation='softmax'))
model.add(Dense(3, activation='tanh')) # Two softmax layers was less c
model.add(Dense(3, activation='tanh')) # If I have all softmax layers 
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))

model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Softmax')

model.summary()

score_softmax = model.evaluate(X, y)

print("Softmax Loss Score:", score_softmax)
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_45 (Dense)            (None, 3)                 9         
                                                                 
 dense_46 (Dense)            (None, 3)                 12        
                                                                 
 dense_47 (Dense)            (None, 3)                 12        
```

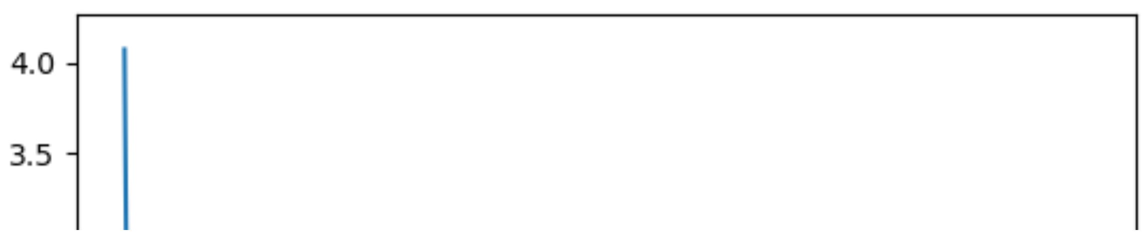
```
 dense_48 (Dense)            (None, 3)                 12

 dense_49 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.1529
Softmax Loss Score: 0.15287908911705017
```

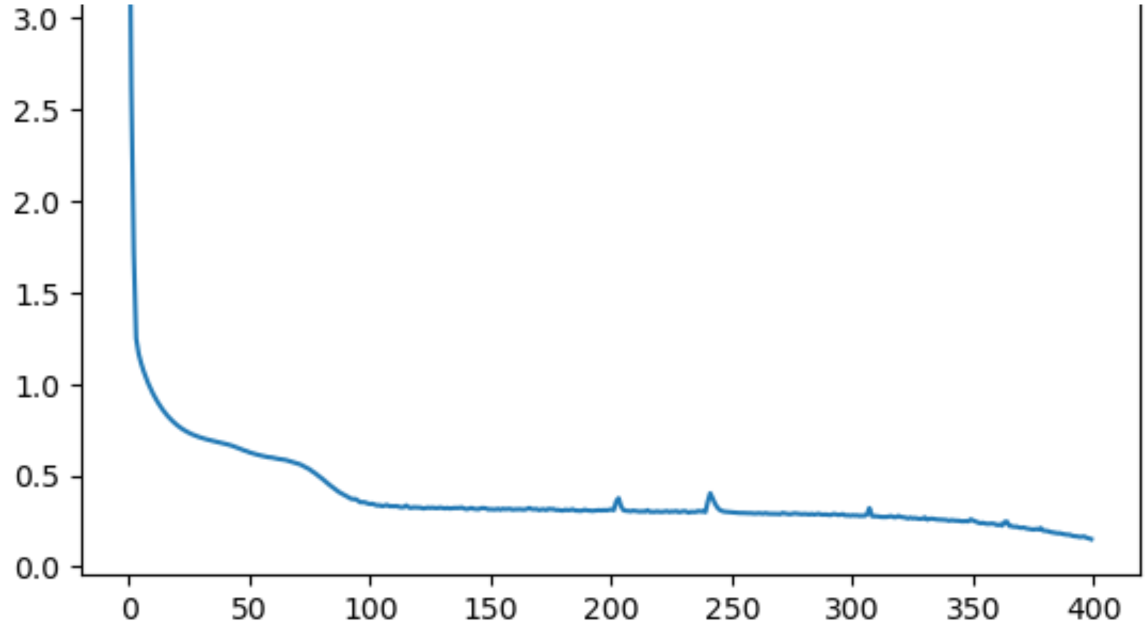


In [16]:

```
# softsign

model = Sequential()

model.add(Dense(3, input_dim=2, activation='softsign'))
model.add(Dense(3, activation='softsign'))
model.add(Dense(3, activation='softsign'))
model.add(Dense(3, activation='softsign'))
model.add(Dense(3, activation='tanh')) # 4 softsign layers with 1 tanh

model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Softsign')

model.summary()

score_softsign = model.evaluate(X, y)

print("Softsign Loss Score:", score_softsign)
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_50 (Dense)            (None, 3)                 9

 dense_51 (Dense)            (None, 3)                 12

 dense_52 (Dense)            (None, 3)                 12

 dense_53 (Dense)            (None, 3)                 12

 dense_54 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.1448
Softsign Loss Score: 0.14476850628852844
```



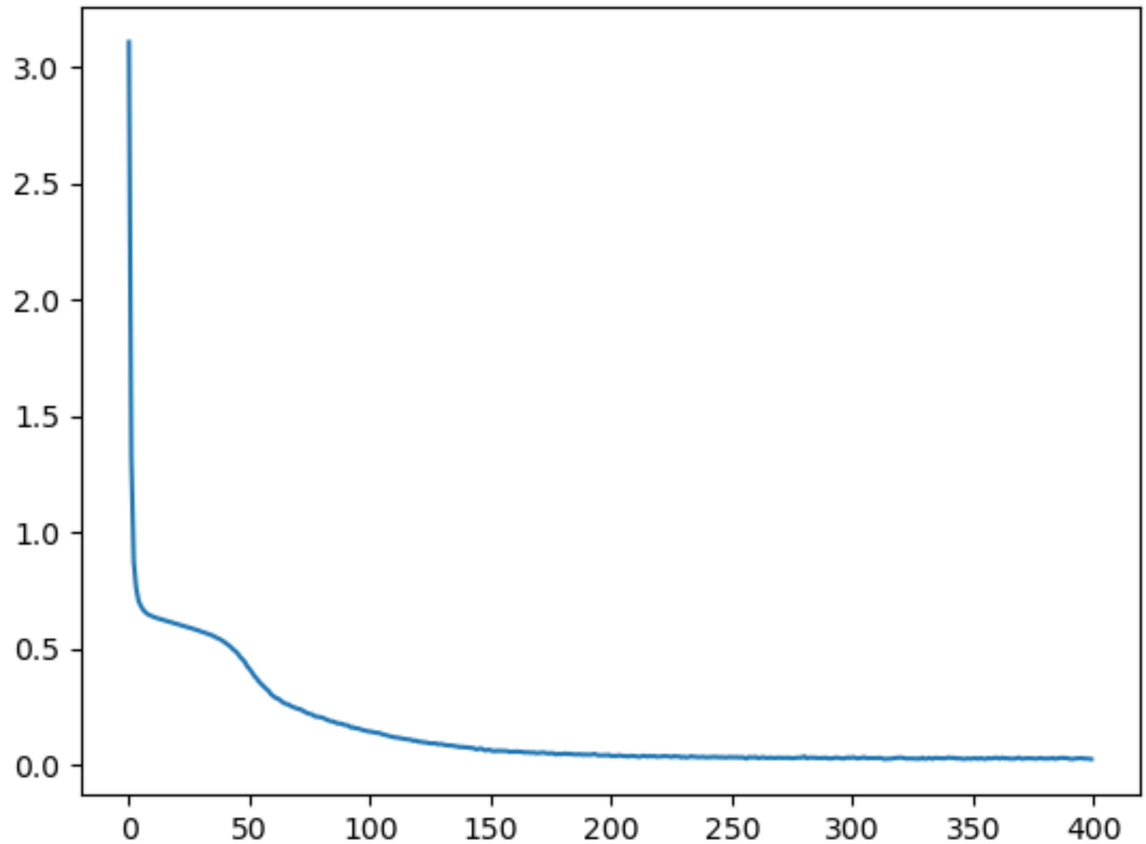


In [17]:

```
# elu

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='Adam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Elu')

model.summary()

score_elu = model.evaluate(X, y)

print("Elu Loss Score:", score_elu)
```

```
Model: "sequential_11"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_55 (Dense)            (None, 3)                 9

 dense_56 (Dense)            (None, 3)                 12

 dense_57 (Dense)            (None, 3)                 12
```

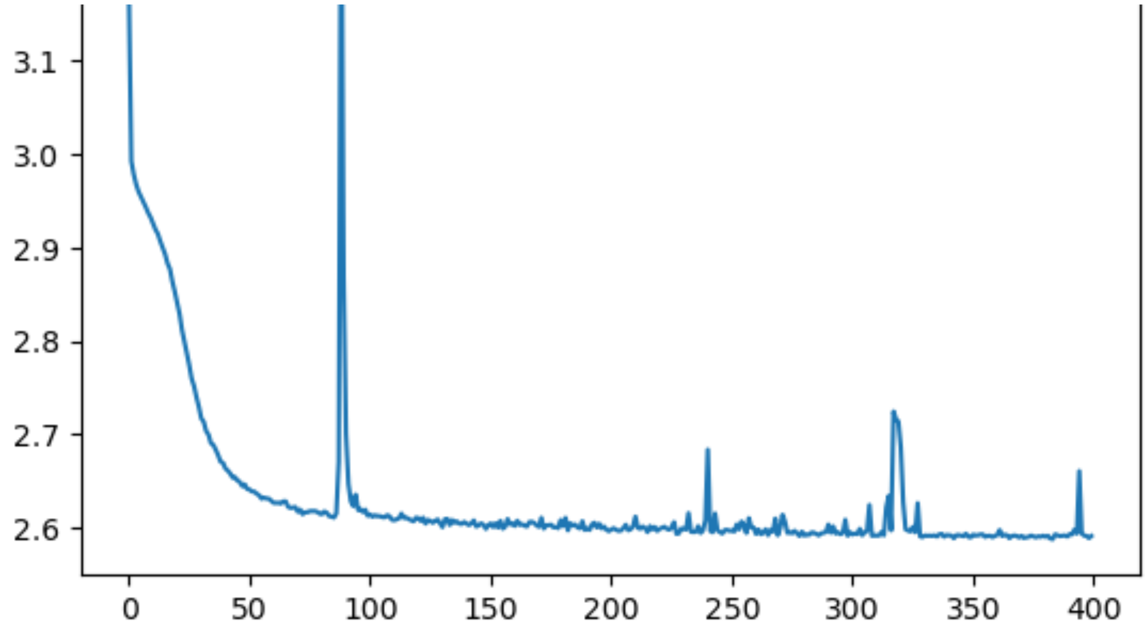
```
 dense_58 (Dense)            (None, 3)                 12

 dense_59 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0192
Elu Loss Score: 0.019223075360059738
```



In [18]:

```
# selu

model = Sequential()

model.add(Dense(3, input_dim=2, activation='selu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with selu

model.compile(loss='binary_crossentropy', optimizer='sgd')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Selu')

model.summary()

score_selu = model.evaluate(X, y)

print("Selu Loss Score:", score_selu)
```

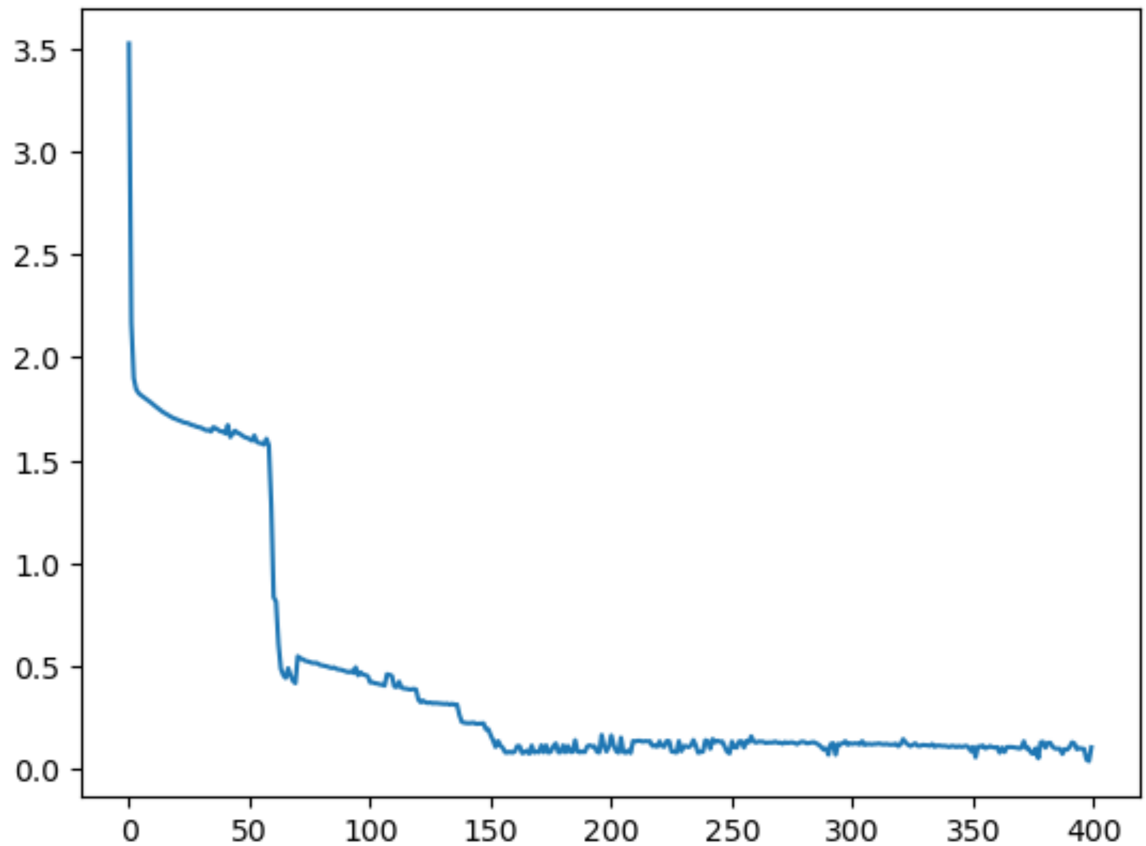
```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_60 (Dense)            (None, 3)                 9

 dense_61 (Dense)            (None, 3)                 12

 dense_62 (Dense)            (None, 3)                 12

 dense_63 (Dense)            (None, 3)                 12

 dense_64 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 2ms/step - loss: 2.5857
Selu Loss Score: 2.5857462882995605
```



In [19]:
```
# RMSprop

from keras.optimizers import RMSprop

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='RMSprop')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='RMSprop')

model.summary()

score_rmsprop = model.evaluate(X, y)

print("RMSprop Loss Score:", score_rmsprop)
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_65 (Dense)            (None, 3)                 9

 dense_66 (Dense)            (None, 3)                 12
```

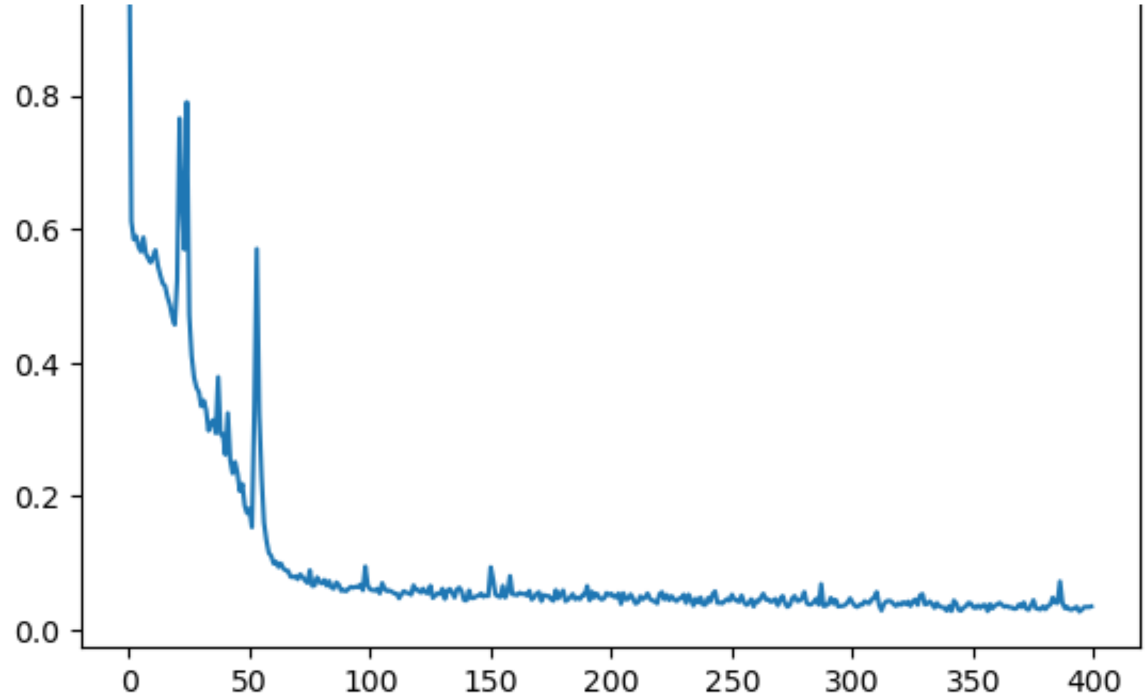
```
 dense_67 (Dense)            (None, 3)                 12

 dense_68 (Dense)            (None, 3)                 12

 dense_69 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0671
RMSprop Loss Score: 0.067109614610672
```



In [20]:

```python
# SGD

from keras.optimizers import Adam

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='sgd')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='SGD')

model.summary()

score_SGD = model.evaluate(X, y)

print("SGD Loss Score:", score_SGD)
```

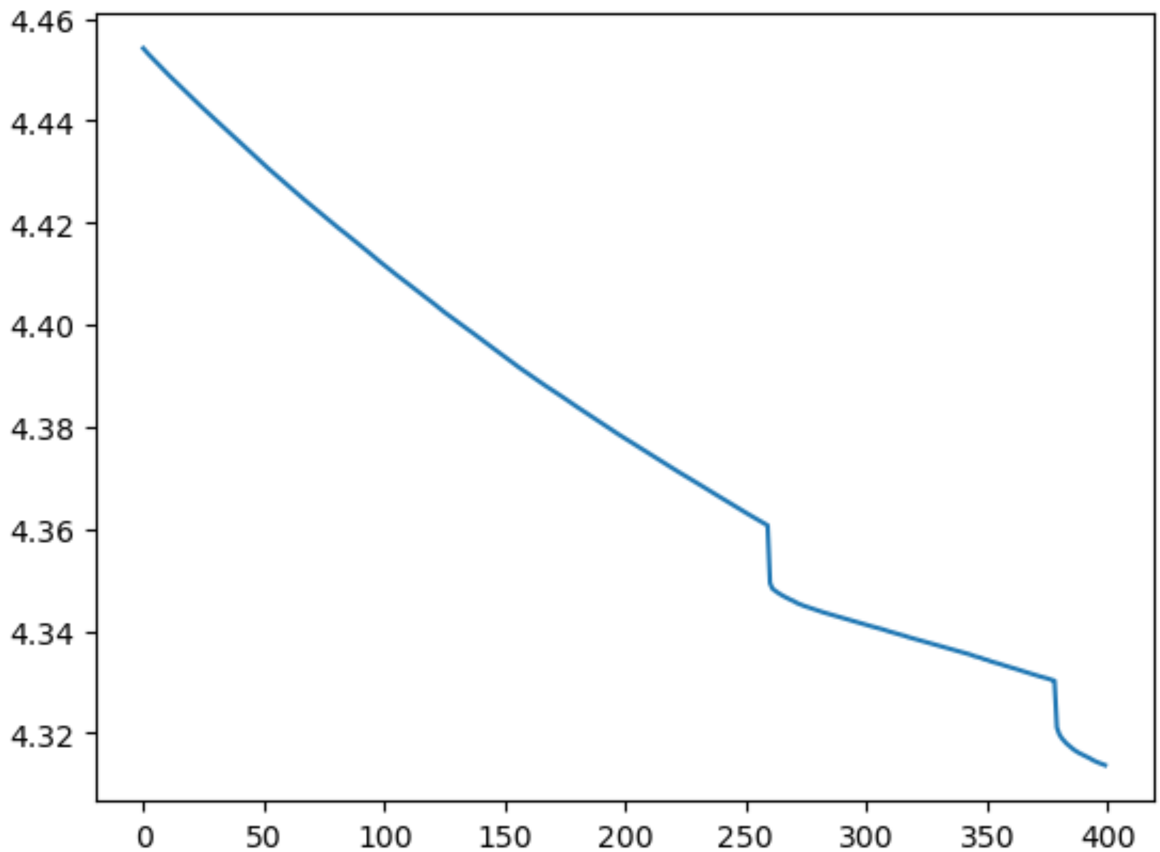
```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_70 (Dense)            (None, 3)                 9

 dense_71 (Dense)            (None, 3)                 12

 dense_72 (Dense)            (None, 3)                 12

 dense_73 (Dense)            (None, 3)                 12

 dense_74 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0232
SGD Loss Score: 0.023236945271492004
```



In [21]:
```python
# Adadelta

from keras.optimizers import Adadelta

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='Adadelta')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Adadelta')

model.summary()

score_Adadelta = model.evaluate(X, y)

print("Adadelta Loss Score:", score_Adadelta)
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_75 (Dense)            (None, 3)                 9
```

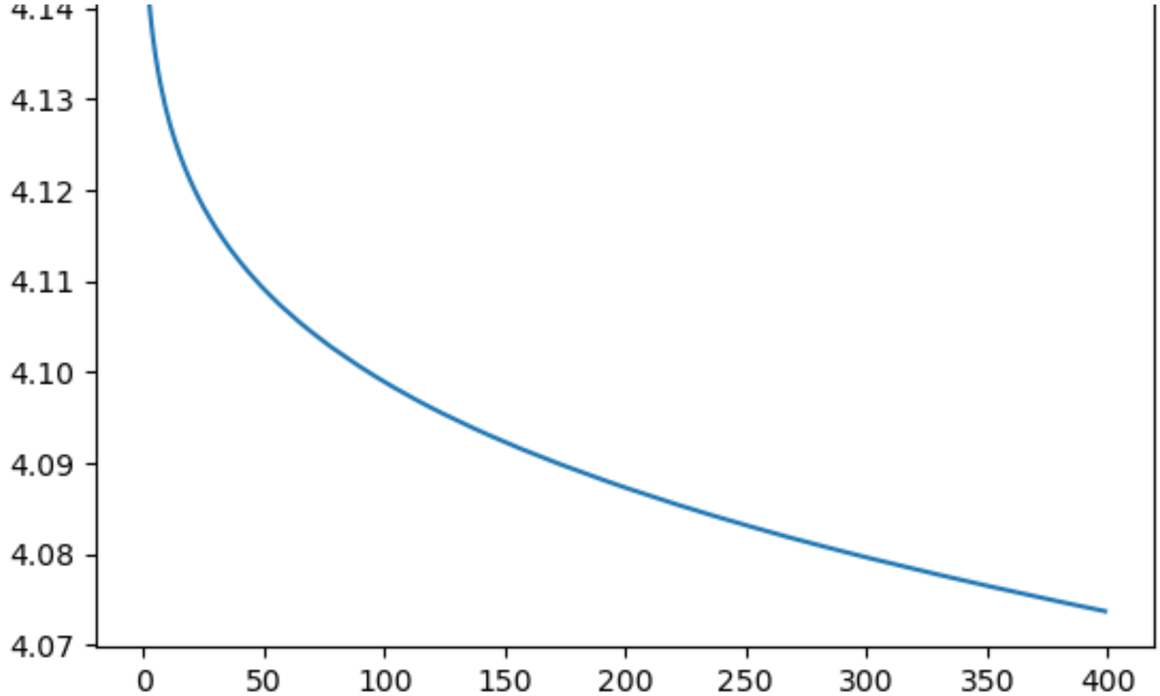
```
 dense_76 (Dense)            (None, 3)                 12

 dense_77 (Dense)            (None, 3)                 12

 dense_78 (Dense)            (None, 3)                 12

 dense_79 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.3135
Adadelta Loss Score: 4.313548564910889
```



In [22]:

```python
# Adagrad

from keras.optimizers import Adagrad

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='Adagrad')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Adagrad')

model.summary()

score_Adagrad = model.evaluate(X, y)

print("Adagrad Loss Score:", score_Adagrad)
```

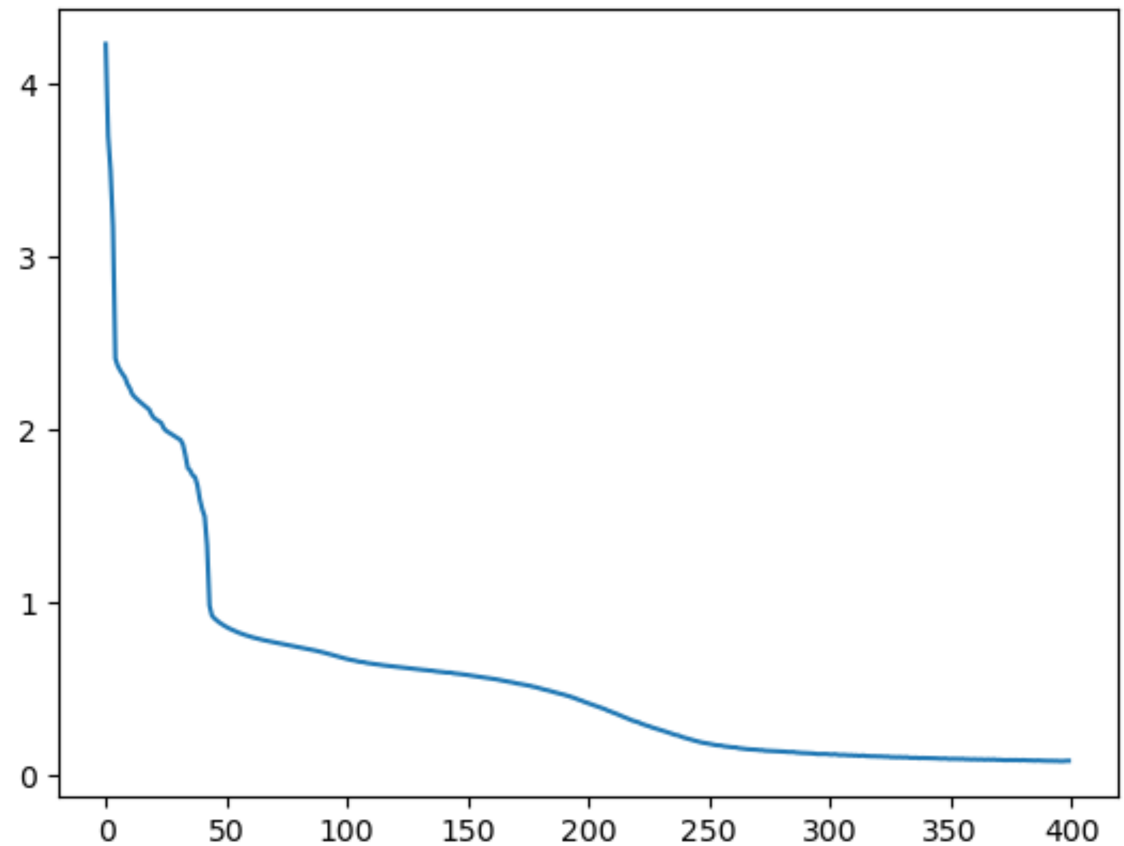
```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_80 (Dense)            (None, 3)                 9

 dense_81 (Dense)            (None, 3)                 12

 dense_82 (Dense)            (None, 3)                 12

 dense_83 (Dense)            (None, 3)                 12

 dense_84 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 4.0737
Adagrad Loss Score: 4.073662757873535
```



In [23]:
```
# Adamax

from keras.optimizers import Adamax

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='Adamax')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Adamax')

model.summary()

score_Adamax = model.evaluate(X, y)

print("Adamax Loss Score:", score_Adamax)
```

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_85 (Dense)            (None, 3)                 9
```

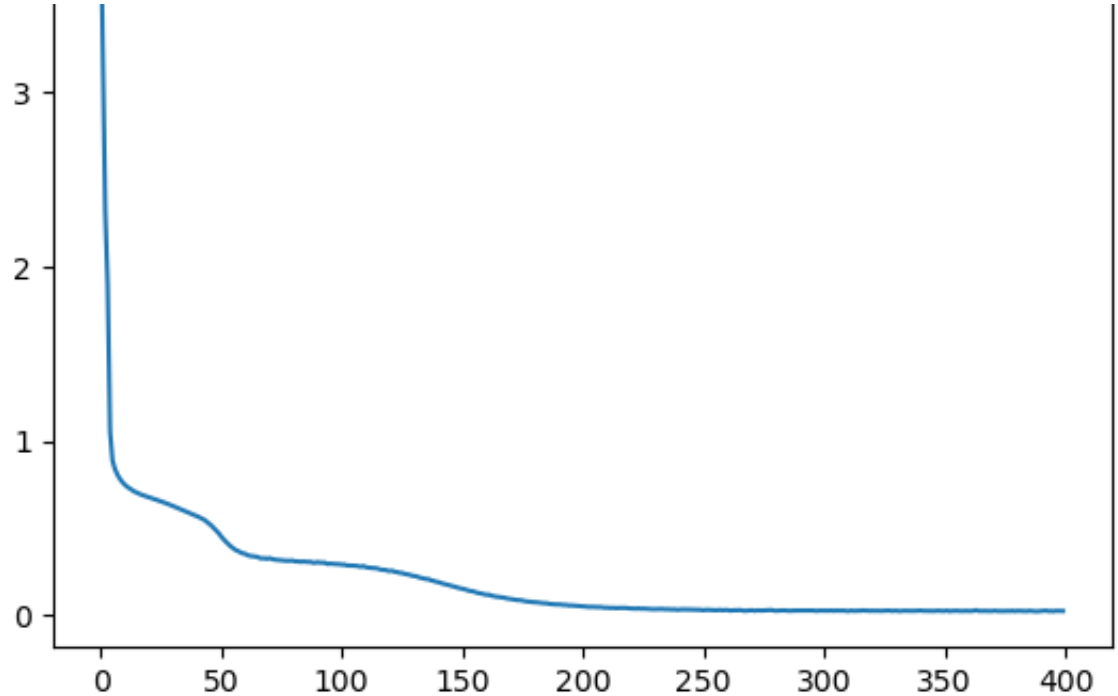
```
 dense_86 (Dense)            (None, 3)                 12

 dense_87 (Dense)            (None, 3)                 12

 dense_88 (Dense)            (None, 3)                 12

 dense_89 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0772
Adamax Loss Score: 0.07723230123519897
```



In [27]:

```
# Nadam

from keras.optimizers import Nadam

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='Nadam')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Nadam')

model.summary()

score_Nadam = model.evaluate(X, y)

print("Nadam Loss Score:", score_Nadam)
```

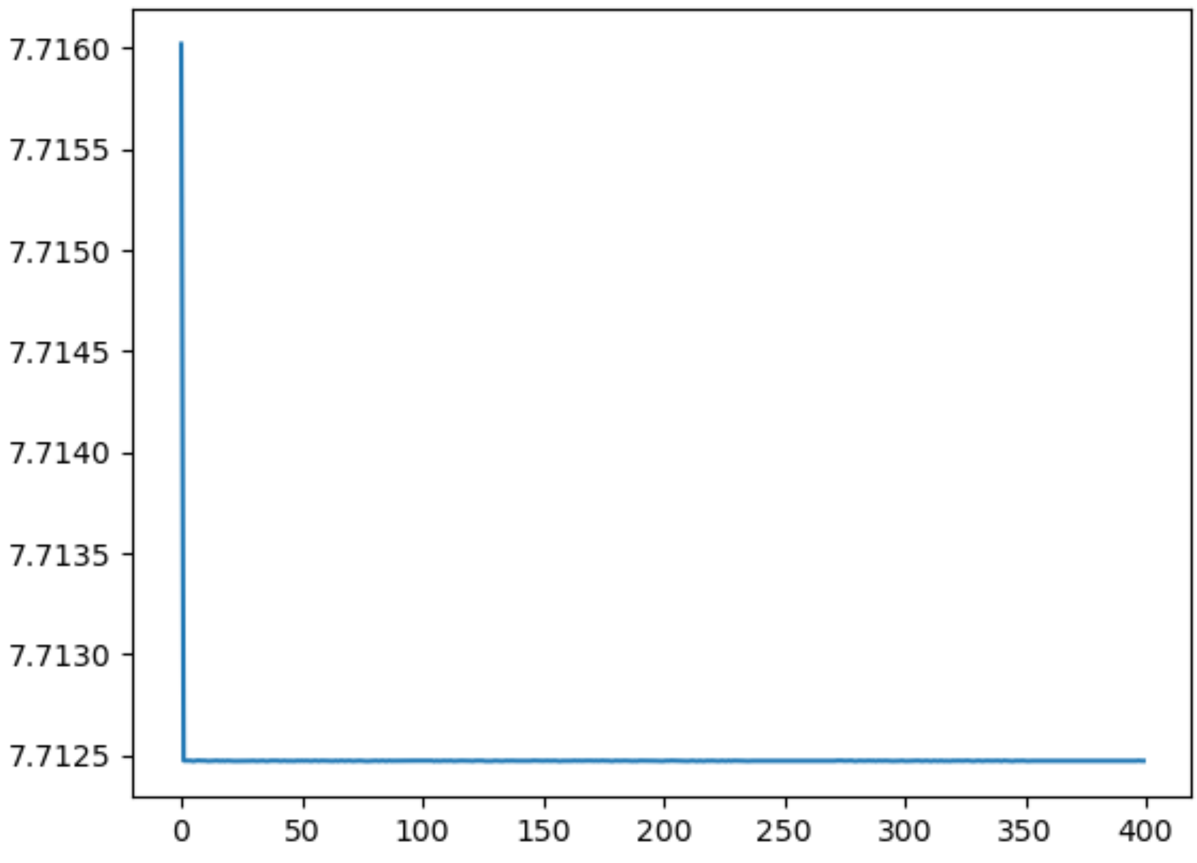
```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_103 (Dense)           (None, 3)                 9

 dense_104 (Dense)           (None, 3)                 12

 dense_105 (Dense)           (None, 3)                 12

 dense_106 (Dense)           (None, 3)                 12

 dense_107 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.0171
Nadam Loss Score: 0.017082305625081062
```



In [25]:
```
# Ftrl

from keras.optimizers import Ftrl

model = Sequential()

model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

model.compile(loss='binary_crossentropy', optimizer='Ftrl')

history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='Ftrl')

model.summary()

score_Ftrl = model.evaluate(X, y)

print("Ftrl Loss Score:", score_Ftrl)
```

```
Model: "sequential_19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_93 (Dense)            (None, 3)                 9
```

```
 dense_94 (Dense)            (None, 3)                 12

 dense_95 (Dense)            (None, 3)                 12

 dense_96 (Dense)            (None, 3)                 12

 dense_97 (Dense)            (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 7.7125
Ftrl Loss Score: 7.7124738693237305
```



In [26]:

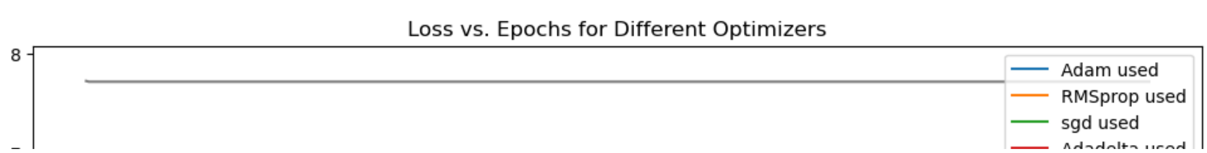
```
# All Optimizers on One Graph

        # As noted elsewhere the results between looped execution and

scores_opt = []
optimizers = ['Adam','RMSprop','sgd','Adadelta','Adagrad','Adamax','Na
plt.figure(figsize=(12, 8))
model = Sequential()
model.add(Dense(3, input_dim=2, activation='elu'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh'))
model.add(Dense(3, activation='tanh')) # Diminishing returns with elu.

for opt in optimizers:

    model.compile(loss='binary_crossentropy', optimizer=opt)

    history = model.fit(X, y, batch_size=2, epochs=400, verbose=0)

    plt.plot(history.history['loss'], label=f'{opt} used')

    model.summary()

    score = model.evaluate(X, y)
    scores_opt.append(score)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs for Different Optimizers')
plt.legend()
plt.show()
printout_opt = [item for sublist in zip(optimizers, scores_opt) for it
print("Optimizers:", printout_opt)
```

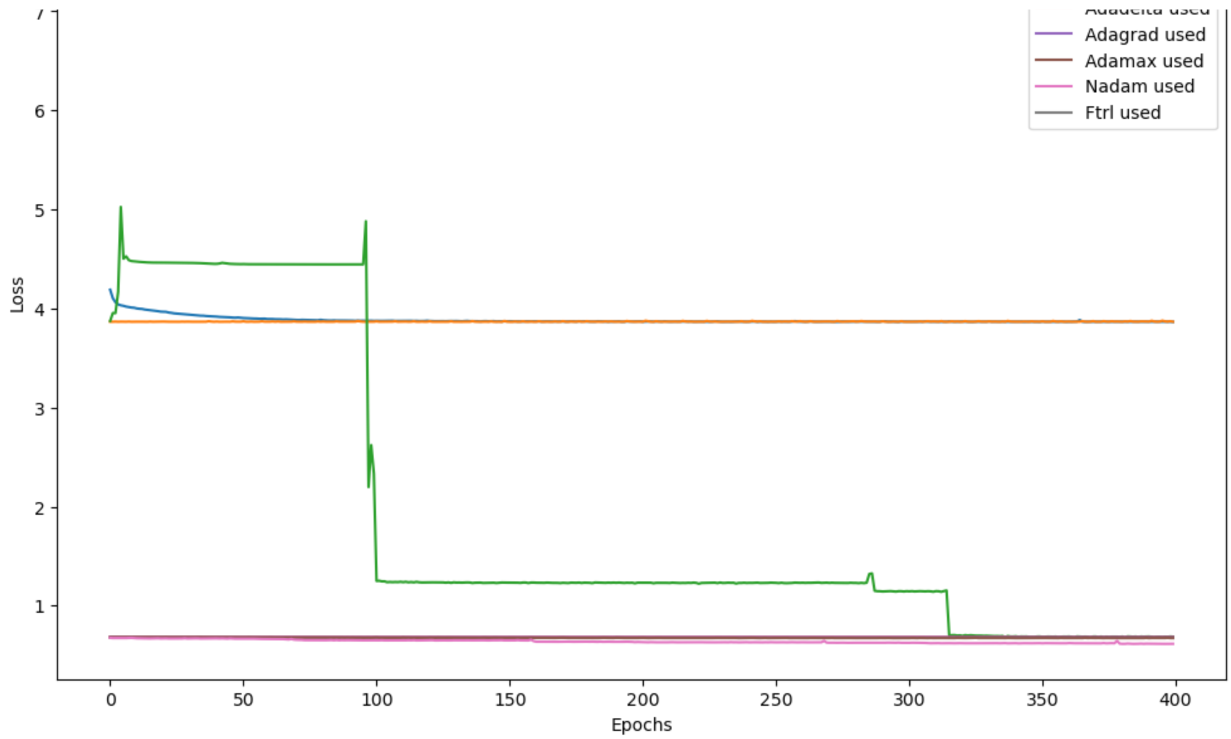
```
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
```

```
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 3.8654
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 3.8650
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.6815
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9
```

```
 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.6807
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.6801
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
```

```
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 0.6711
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________

5/5 [==============================] - 0s 1ms/step - loss: 0.6133
Model: "sequential_20"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_98 (Dense)            (None, 3)                 9

 dense_99 (Dense)            (None, 3)                 12

 dense_100 (Dense)           (None, 3)                 12

 dense_101 (Dense)           (None, 3)                 12

 dense_102 (Dense)           (None, 3)                 12

=================================================================
Total params: 57 (228.00 Byte)
Trainable params: 57 (228.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
5/5 [==============================] - 0s 1ms/step - loss: 7.7125
```






```
Optimizers: ['Adam', 3.8653903007507324, 'RMSprop', 3.865039825439453
, 'sgd', 0.6815272569656372, 'Adadelta', 0.6806966066360474, 'Adagrad
', 0.6800661683082581, 'Adamax', 0.6711133718490601, 'Nadam', 0.61329
35881614685, 'Ftrl', 7.7124738693237305]
```

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/)

In [29]:
```
# BYOD Import

heart = pd.read_csv('../data/heart+disease/processed.cleveland.data',
                    names=["age", "sex", "cp", "trestbps", "chol", "fb
                           "exang", "oldpeak", "slope", "ca", "thal",

heart_clean = heart[pd.to_numeric(heart["ca"], errors='coerce').notnul


heart_clean = heart_clean[pd.to_numeric(heart_clean["oldpeak"], errors

heart_clean = heart_clean[pd.to_numeric(heart_clean["thal"], errors='c

heart_clean["heart_disease"] = heart_clean['num'].astype(bool)
heart_clean["heart_disease"],heart_clean['num']
heart_clean = heart_clean.dropna()
```

In [30]:
```
heart_clean
```

Out[30]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	r
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
297	57.0	0.0	4.0	140.0	241.0	0.0	0.0	123.0	1.0	0.2	2.0	0.0	7.0	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	

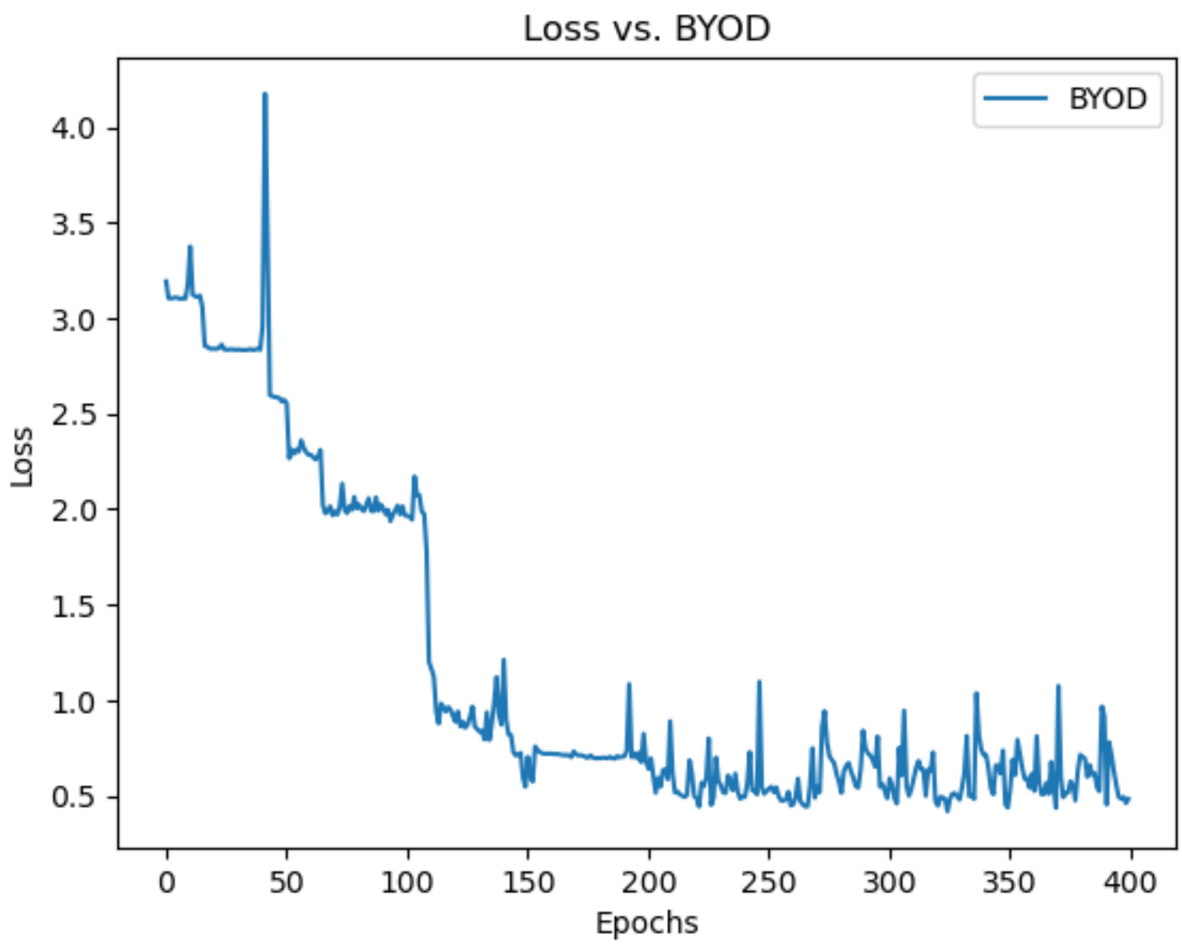
297 rows × 15 columns

In [31]:
```
# Setting up the target and dataset.

xray = heart_clean.copy()
yankee = heart_clean['heart_disease'].copy()
xray = xray.drop(columns=['num', 'heart_disease'], axis=1)

xray = np.asarray(xray).astype('float32') # Issues with 'float' data t
```

In [32]:
```
# NN Model Build

byod = Sequential()

byod.add(Dense(24, input_dim=13, activation='elu')) # Exponential Line
#byod.add(Dense(12, activation='sigmoid')) # From what I can see sigmo
byod.add(Dense(24, activation='selu')) # Scaled Exponential Linear Uni
byod.add(Dense(24, activation='softsign')) # Softmax converts a vector
                                           # Softsign activation funct
byod.add(Dense(24, activation='tanh'))

byod.compile(loss='binary_crossentropy', optimizer='Nadam') # Much lik
                                                            # Nadam is
history = byod.fit(xray, yankee, batch_size=2, epochs=400, verbose=0)

plt.plot(history.history['loss'], label='BYOD')

byod.summary()

score_byod = byod.evaluate(xray, yankee)

plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. BYOD')
plt.legend()
plt.show()
print("BYOD Loss Score:", score_byod)
```

```
Model: "sequential_23"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_112 (Dense)           (None, 24)                336

 dense_113 (Dense)           (None, 24)                600

 dense_114 (Dense)           (None, 24)                600

 dense_115 (Dense)           (None, 24)                600
```

```
=================================================================
Total params: 2136 (8.34 KB)
Trainable params: 2136 (8.34 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
10/10 [==============================] - 0s 891us/step - loss: 0.4335
```




```
BYOD Loss Score: 0.4335453510284424
```

0.43 loss score was the best I was able to optimize for with the data I brought. The Heart dataset only has 297 observations once it has been cleaned which is a fairly small training size for a neural net. With that in mind I am fairly happy with the results.

In [73]:
```
model = Sequential()

model.add(Dense(2, input_dim=2, activation='tanh'))  #sigmoid, relu
model.add(Dense(2, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
model.add(Dense(1,input_dim=2, activation='sigmoid'))


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

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

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

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learn
ing_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legac
y.SGD.

Epoch 1/400
80/80 [==============================] - 0s 1ms/step - loss: 0.7131
Epoch 2/400
80/80 [==============================] - 0s 820us/step - loss: 0.7043
Epoch 3/400
80/80 [==============================] - 0s 790us/step - loss: 0.6985
Epoch 4/400
80/80 [==============================] - 0s 772us/step - loss: 0.6949
Epoch 5/400
80/80 [==============================] - 0s 805us/step - loss: 0.6925
Epoch 6/400
80/80 [==============================] - 0s 754us/step - loss: 0.6908
Epoch 7/400
80/80 [==============================] - 0s 737us/step - loss: 0.6898
Epoch 8/400
80/80 [==============================] - 0s 713us/step - loss: 0.6888
```

In [52]:
```
print(model.predict_proba(X).reshape(4*n))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[52], line 1
----> 1 print(model.predict_proba(X).reshape(4*n))

AttributeError: 'Sequential' object has no attribute 'predict_proba'
```

In [53]:
```
scores = model.evaluate(X, y)
scores, model.metrics_names
```

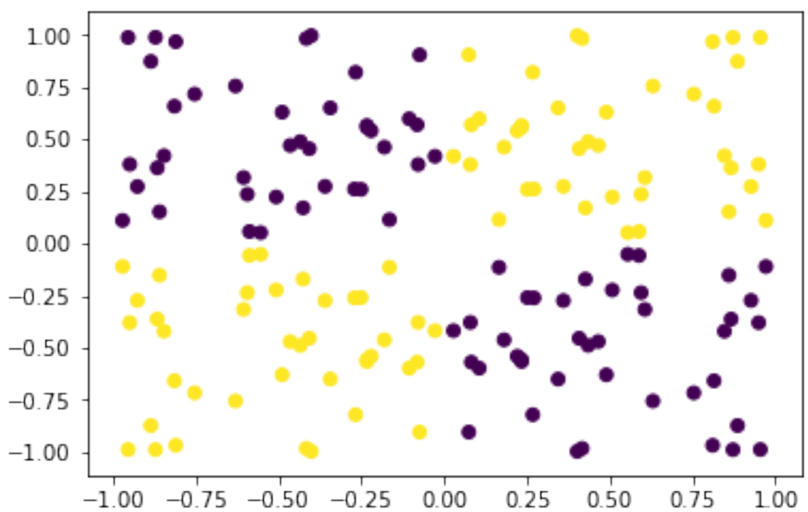
```
5/5 [==============================] - 0s 1ms/step - loss: 2.3497
```

Out[53]: (2.349733829498291, ['loss'])

In [127]:
```
plt.scatter(*zip(*X), c=model.predict_classes(X))
```

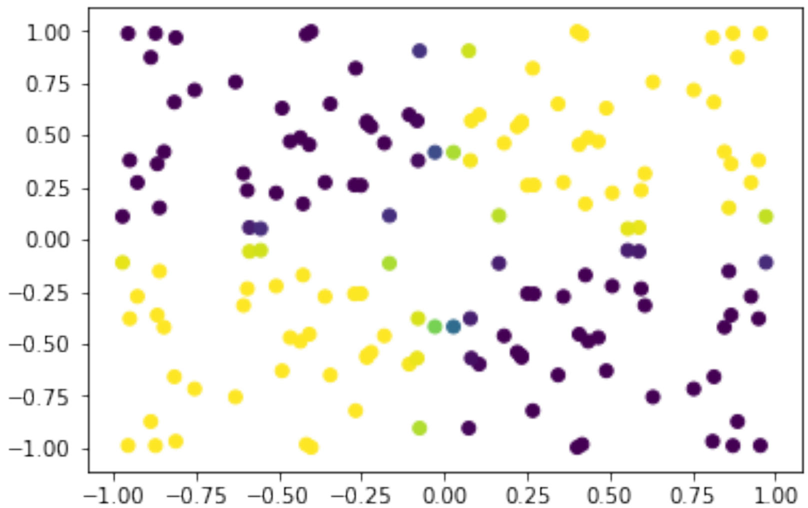
```
 32/160 [=====>........................] - ETA: 0s
```

Out[127]: <matplotlib.collections.PathCollection at 0x121aed828>



In [128]:
```
plt.scatter(*zip(*X), c=model.predict(X))
```

Out[128]: <matplotlib.collections.PathCollection at 0x120f4ee80>



Using Diabetes data

[http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data (http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data)](http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data)

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

In [135]:
```
# load pima indians dataset
dataset = np.loadtxt("../data/pima-indians-diabetes.data", delimiter="
# split into input (X) and output (Y) variables
Z = dataset[:,0:8]
W = dataset[:,8]
```

In [144]:
```
dataset.head()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[144], line 1
----> 1 dataset.head()

AttributeError: 'numpy.ndarray' object has no attribute 'head'
```

In [136]:
```
# create model
model = Sequential()
model.add(Dense(16, input_dim=8, activation='tanh'))
model.add(Dense(16, activation='tanh'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
# Fit the model
model.fit(Z, W, epochs=1000, batch_size=10)
# evaluate the model
scores = model.evaluate(Z, W)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Epoch 1/1000
768/768 [==============================] - 1s - loss: 0.6821 - acc: 0
.5820
Epoch 2/1000
768/768 [==============================] - 0s - loss: 0.6273 - acc: 0
.6536
Epoch 3/1000
768/768 [==============================] - 0s - loss: 0.6122 - acc: 0
.6719
Epoch 4/1000
768/768 [==============================] - 0s - loss: 0.6111 - acc: 0
.6680
Epoch 5/1000
768/768 [==============================] - 0s - loss: 0.6065 - acc: 0
.6862
Epoch 6/1000
768/768 [==============================] - 0s - loss: 0.6049 - acc: 0
.6745
Epoch 7/1000
```

Type *Markdown* and LaTeX: α^2

In [ ]:

```
```