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/)

In [8]:
```
# part 1
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD,Adam
import tensorflow as tf

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


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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Input In [8], in <cell line: 2>()
      1 # part 1
----> 2 from keras.models import Sequential
      3 from keras.layers import Dense
      4 from keras.optimizers import SGD,Adam

ModuleNotFoundError: No module named 'keras'
```

In [9]:
```
n = 40
xx = np.random.random((n,1))
yy = np.random.random((n,1))

X = np.array([np.array([xx,-xx,-xx,xx]),np.array([yy,-yy,yy,-yy])]).reshape(2,4*r
y = np.array([np.ones([2*n]),np.zeros([2*n])]).reshape(4*n)
plt.scatter(*zip(*X), c=y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [9], in <cell line: 2>()
      1 n = 40
----> 2 xx = np.random.random((n,1))
      3 yy = np.random.random((n,1))
      5 X = np.array([np.array([xx,-xx,-xx,xx]),np.array([yy,-yy,yy,-yy])]).res
hape(2,4*n).T

NameError: name 'np' is not defined
```

In [10]:
```
num_layers = [1, 2, 3, 4, 5]  # different layers
scores_2 = []  # 2 nuerons
for num_layer in num_layers:

    # build model and evaluate
    model = Sequential()
    model.add(Dense(2, input_dim=2, activation="tanh"))
    for _ in range(num_layer-1):
        model.add(Dense(2, activation="tanh"))
    model.add(Dense(1, activation="sigmoid"))

    sgd = SGD(lr=0.1)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy']

    model.fit(X, y, batch_size=2, epochs=400)
    score = model.evaluate(X, y)
    scores_2.append(score[1])  # save accuracy
plt.plot(num_layers, scores_2)
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      2 scores_2 = []  # 2 nuerons
      3 for num_layer in num_layers:
      4
      5     # build model and evaluate
----> 6     model = Sequential()
      7     model.add(Dense(2, input_dim=2, activation="tanh"))
      8     for _ in range(num_layer-1):

NameError: name 'Sequential' is not defined
```

In [11]:
```
scores_2 # layer_num = 5
```

Out[11]:
```
[0.862500011920929,
 0.856249988079071,
 0.862500011920929,
 0.8687499761581421,
 0.7875000238418579]
```

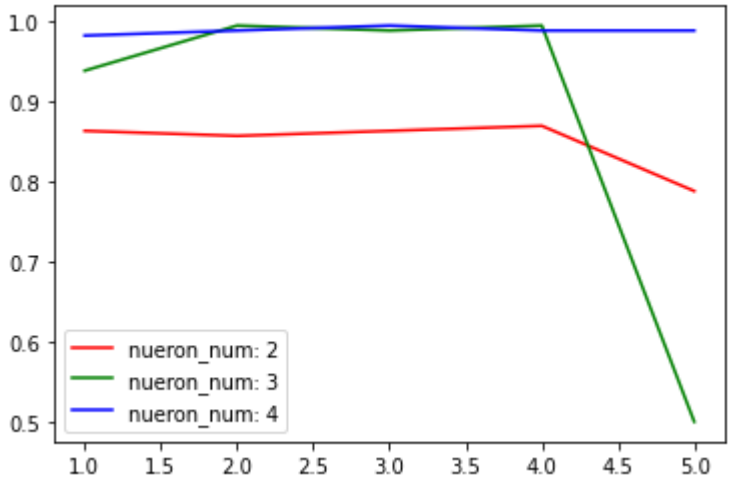
In [5]:
```
num_layers = [1, 2, 3, 4, 5]  # different layers
scores_3 = []  # nueron_num = 3
for num_layer in num_layers:

    # build model and evaluate
    model = Sequential()
    model.add(Dense(3, input_dim=2, activation="tanh"))
    for _ in range(num_layer - 1):
        model.add(Dense(3, activation="tanh"))
    model.add(Dense(1, activation="sigmoid"))

    sgd = SGD(lr=0.1)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy']

    model.fit(X, y, batch_size=2, epochs=400)
    score = model.evaluate(X, y)
    scores_3.append(score[1])  # save accuracy

print(scores_3)
plt.plot(num_layers, scores_3)
plt.show()
```

```
Epoch 43/400
80/80 [==============================] - 0s 929us/step - loss: 0.1935 - accur
acy: 0.9125
Epoch 44/400
80/80 [==============================] - 0s 936us/step - loss: 0.1891 - accur
acy: 0.9187
Epoch 45/400
80/80 [==============================] - 0s 998us/step - loss: 0.1961 - accur
acy: 0.9312
Epoch 46/400
80/80 [==============================] - 0s 1ms/step - loss: 0.1974 - accurac
y: 0.9125
Epoch 47/400
80/80 [==============================] - 0s 934us/step - loss: 0.1895 - accur
acy: 0.9250
Epoch 48/400
80/80 [==============================] - 0s 945us/step - loss: 0.1872 - accur
acy: 0.9000
Epoch 49/400
80/80 [==============================] - 0s 953us/step - loss: 0.1810 - accur
```

In [6]:
```
num_layers = [1, 2, 3, 4, 5]  # different layers
scores_4 = []  # nueron_num = 4
for num_layer in num_layers:

    # build model and evaluate
    model = Sequential()
    model.add(Dense(4, input_dim=2, activation="tanh"))
    for _ in range(num_layer - 1):
        model.add(Dense(4, activation="tanh"))
    model.add(Dense(1, activation="sigmoid"))

    sgd = SGD(lr=0.1)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy']

    model.fit(X, y, batch_size=2, epochs=400)
    score = model.evaluate(X, y)
    scores_4.append(score[1])  # save accuracy

print(scores_4)
plt.plot(num_layers, scores_4)
plt.show()
```

```
Epoch 1/400

/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/gradient
_descent.py:108: UserWarning: The `lr` argument is deprecated, use `learning_
rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

In [12]:
```
plt.figure()
plt.plot(num_layers, scores_2, "r", label="nueron_num: 2")
plt.plot(num_layers, scores_3, "g", label="nueron_num: 3")
plt.plot(num_layers, scores_4, "b", label="nueron_num: 4")
plt.legend()
plt.show()
```



the most optimal configuraion

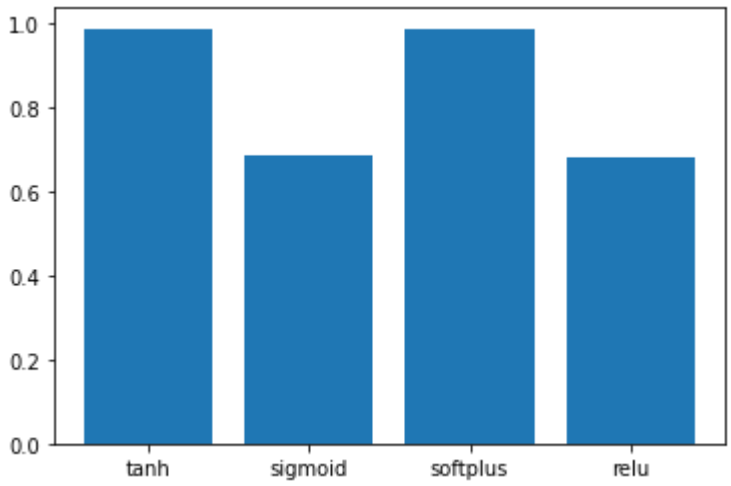
- 3 hidden layers, 4 nuerons per layer
- 2 or 4 hidden layers, 3 nuerons per layer

In [ ]:
```
# different activations tanh, sigmoid,softplus and relu
# 3 hidden layers, 4 nuerons per layer
activations = ['tanh', 'sigmoid', 'softplus', 'relu']
scores = []
for activation in activations:

    # build model and evaluate
    model = Sequential()
    model.add(Dense(4, input_dim=2, activation=activation))  # 3 hidden layer
    model.add(Dense(4, activation=activation))
    model.add(Dense(4, activation=activation))
    model.add(Dense(1, activation="sigmoid"))  # output layer

    sgd = SGD(lr=0.1)
    model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy']

    model.fit(X, y, batch_size=2, epochs=400)
    score = model.evaluate(X, y)
    scores.append(score[1])  # save accuracy

print(scores)
```

In [19]:
```
scores
```

Out[19]:
```
[0.987500011920929, 0.6875, 0.987500011920929, 0.6812499761581421]
```

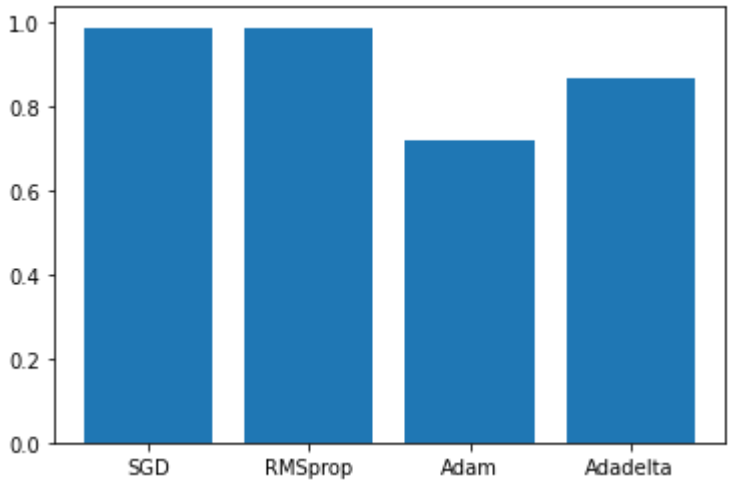
In [21]:
```
plt.bar(np.array(activations), np.array(scores))
plt.show()  # tanh or softplus are better
```



In [22]:
```
# different optimizer SGD, Adam, RMSprop and Adadelta
# 3 hidden layers, 4 nuerons per layer, activation = tanh
from keras.optimizers import SGD, RMSprop, Adam, Adadelta
optimizers = [SGD(lr=0.1), RMSprop(lr=0.1), Adam(lr=0.1), Adadelta(lr=0.1)]
scores = []
for optimizer in optimizers:

    # build model and evaluate
    model = Sequential()
    model.add(Dense(4, input_dim=2, activation='tanh'))  # 3 hidden layer
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(4, activation='tanh'))
    model.add(Dense(1, activation="sigmoid"))  # output layer

    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accu

    model.fit(X, y, batch_size=2, epochs=400)
    score = model.evaluate(X, y)
    scores.append(score[1])  # save accuracy

print(scores)
```

```
Epoch 1/400

/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/gradient
_descent.py:108: UserWarning: The `lr` argument is deprecated, use `learning_
rate` instead.
  super(SGD, self).__init__(name, **kwargs)
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/rmsprop.
py:135: UserWarning: The `lr` argument is deprecated, use `learning_rate` ins
tead.
  super(RMSprop, self).__init__(name, **kwargs)
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/adam.py:
110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instea
d.
  super(Adam, self).__init__(name, **kwargs)
/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/adadelt
a.py:77: UserWarning: The `lr` argument is deprecated, use `learning_rate` in
stead.
  super(Adadelta, self).__init__(name, **kwargs)
```

In [26]:
```
plt.bar(['SGD', 'RMSprop', 'Adam', 'Adadelta'], scores)
plt.show()  # SGD and RMSprop are better
```



part 1: final setup

- layer_num: 3
- nuerons_per_layer: 4
- activation: tanh
- optimizer: SGD

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 [ ]:
```
# part 2
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD, Adam

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

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

In [ ]:
```
from sklearn.datasets import load_iris
X,y = load_iris(return_X_y=True)
X,y
```

Out[8]:
```
(array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
```

In [ ]:
```
X.shape, y.shape
```

Out[9]:
```
((150, 4), (150,))
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[11]:
```
((105, 4), (45, 4), (105,), (45,))
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='relu', use_bias=True))
model.add(Dense(6, activation='relu', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.01), me
model.fit(X_train, y_train, batch_size=5, epochs=100)

score = model.evaluate(X_test, y_test)
score
```

```
Epoch 1/100
21/21 [==============================] - 1s 1ms/step - loss: 1.3379 - accurac
y: 0.3429
Epoch 2/100
21/21 [==============================] - 0s 1ms/step - loss: 0.9442 - accurac
y: 0.6286
Epoch 3/100
21/21 [==============================] - 0s 1ms/step - loss: 0.7758 - accurac
y: 0.6857
Epoch 4/100
21/21 [==============================] - 0s 1ms/step - loss: 0.5901 - accurac
y: 0.7048
Epoch 5/100
21/21 [==============================] - 0s 1ms/step - loss: 0.4574 - accurac
y: 0.8857
Epoch 6/100
21/21 [==============================] - 0s 1ms/step - loss: 0.3569 - accurac
y: 0.9429
Epoch 7/100
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='relu', use_bias=True))
model.add(Dense(6, activation='relu', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=SGD(lr=0.01), met
model.fit(X_train, y_train, batch_size=5, epochs=100)

score = model.evaluate(X_test, y_test)
score, model.metrics_names
```

```
Epoch 1/100
21/21 [==============================] - 0s 1ms/step - loss: 1.8590 - accurac
y: 0.5333
Epoch 2/100
21/21 [==============================] - 0s 988us/step - loss: 0.6313 - accur
acy: 0.7524
Epoch 3/100
21/21 [==============================] - 0s 1ms/step - loss: 0.5402 - accurac
y: 0.7619
Epoch 4/100
21/21 [==============================] - 0s 1ms/step - loss: 0.4981 - accurac
y: 0.7905
Epoch 5/100
21/21 [==============================] - 0s 1ms/step - loss: 0.4538 - accurac
y: 0.8476
Epoch 6/100
21/21 [==============================] - 0s 1ms/step - loss: 0.4436 - accurac
y: 0.8190
Epoch 7/100
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='relu', use_bias=True))
model.add(Dense(6, activation='relu', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.001), m
model.fit(X_train, y_train, batch_size=5, epochs=100)

score = model.evaluate(X_test, y_test)
score
```

```
Epoch 1/100

/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/adam.py:
110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instea
d.
  super(Adam, self).__init__(name, **kwargs)
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='tanh', use_bias=True)) # activa
# model.add(Dense(8, activation='tanh', use_bias=True))
model.add(Dense(6, activation='tanh', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.001), m
model.fit(X_train, y_train, batch_size=5, epochs=100)

score = model.evaluate(X_test, y_test)
score, model.metrics_names
```

```
Epoch 1/100
21/21 [==============================] - 0s 1ms/step - loss: 1.5217 - accurac
y: 0.3238
Epoch 2/100
21/21 [==============================] - 0s 1ms/step - loss: 1.3029 - accurac
y: 0.3238
Epoch 3/100
21/21 [==============================] - 0s 1ms/step - loss: 1.1363 - accurac
y: 0.3238
Epoch 4/100
21/21 [==============================] - 0s 1ms/step - loss: 0.9826 - accurac
y: 0.4190
Epoch 5/100
21/21 [==============================] - 0s 2ms/step - loss: 0.8415 - accurac
y: 0.7619
Epoch 6/100
21/21 [==============================] - 0s 1ms/step - loss: 0.7357 - accurac
y: 0.7048
Epoch 7/100
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='relu', use_bias=True)) # activa
model.add(Dense(8, activation='relu', use_bias=True))  # more layers, same result
model.add(Dense(6, activation='relu', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.001), m
model.fit(X_train, y_train, batch_size=5, epochs=100)

score = model.evaluate(X_test, y_test)
score, model.metrics_names
```

```
Epoch 1/100
21/21 [==============================] - 0s 1ms/step - loss: 1.2976 - accurac
y: 0.3143
Epoch 2/100
21/21 [==============================] - 0s 1ms/step - loss: 1.1217 - accurac
y: 0.3143
Epoch 3/100
21/21 [==============================] - 0s 1ms/step - loss: 0.9880 - accurac
y: 0.4095
Epoch 4/100
21/21 [==============================] - 0s 1ms/step - loss: 0.8839 - accurac
y: 0.6381
Epoch 5/100
21/21 [==============================] - 0s 1ms/step - loss: 0.8093 - accurac
y: 0.6381
Epoch 6/100
21/21 [==============================] - 0s 1ms/step - loss: 0.7538 - accurac
y: 0.6381
Epoch 7/100
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='relu', use_bias=True))
model.add(Dense(6, activation='relu', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.001), m
model.fit(X_train, y_train, batch_size=5, epochs=200)  # more epochs, get same re

score = model.evaluate(X_test, y_test)
score, model.metrics_names
```

```
Epoch 1/200

/usr/local/lib/python3.7/dist-packages/keras/optimizers/optimizer_v2/adam.py:
110: UserWarning: The `lr` argument is deprecated, use `learning_rate` instea
d.
  super(Adam, self).__init__(name, **kwargs)
```

In [ ]:
```
model = Sequential()
model.add(Dense(12, input_shape=(4,), activation='relu', use_bias=True))
model.add(Dense(6, activation='relu', use_bias=True))
model.add(Dense(3, activation='softmax'))

model.compile(loss="sparse_categorical_crossentropy", optimizer=Adam(lr=0.001), m
model.fit(X_train, y_train, batch_size=10, epochs=100)  # larger batch_size, wors

score = model.evaluate(X_test, y_test)
score
```

```
Epoch 1/100
53/53 [==============================] - 1s 1ms/step - loss: 0.8559 - accurac
y: 0.4095
Epoch 2/100
53/53 [==============================] - 0s 910us/step - loss: 0.7231 - accur
acy: 0.6286
Epoch 3/100
53/53 [==============================] - 0s 1ms/step - loss: 0.6143 - accurac
y: 0.8095
Epoch 4/100
53/53 [==============================] - 0s 1ms/step - loss: 0.5501 - accurac
y: 0.8095
Epoch 5/100
53/53 [==============================] - 0s 1ms/step - loss: 0.5074 - accurac
y: 0.7810
Epoch 6/100
53/53 [==============================] - 0s 943us/step - loss: 0.4588 - accur
acy: 0.8476
Epoch 7/100
```

optimal setup

- layer_num: 3
- activation: relu or tanh. softmax for classify
- learning_rate: 0.001
- optimizer: Adam
- batch_size: 5
- epoch: 100