Name: Chad Markwell

Provide a code to pull your data, ideally from a raw source, and prepare it for model training.

The following is a link to the google sheets document where I did a lot of of the normalization and turning the raw data into something that can be imported and used a lot easier
https://docs.google.com/spreadsheets/d/1nDqoVzDJoELt0uuj-tc5PtUzxHw0YXZzvhVkqs4g0Ag/edit?usp=sharing

```
import urllib.request as urllib
import numpy as np

#data was originally obtained from

#read in data
url = "https://raw.githubusercontent.com/markw1ce/CPS580Project/8e838119a56901bc7aa16380c0ccb
file = urllib.urlopen(url)
data=np.loadtxt(file, delimiter=',')



#this is used to comfirm the data I was getting was what I expected
print("begining Data")
print(type(data))
print(data.shape)
print(data)
#print(data[0][3])


#i need to add normalization here for reducing the year value as well as converting the team
for i in data:
  i[3]=i[3]-1871
print("datainfo after normalization of year")#this shows my year adjusted correctly
print(data.shape)
print(data)
print(data[0][3])
print(data[1][3])

#this is how we insert https://numpy.org/doc/stable/reference/generated/numpy.insert.html
#this is how we delete https://numpy.org/doc/stable/reference/generated/numpy.delete.html
data2 = np.arange(23160970)
data2 = data2.reshape([38926, 595])

d=0
for j in data:
  #this half does the away team
  i = 1
```

```
  while i <= 248:
    if(j[5]==i):
      j=np.insert(j,6,1)
    else:
      j=np.insert(j,6,0)
    i += 1
  #this half does the home team
  i = 1
  while i <= 248:
    if(j[4]==i):
      j=np.insert(j,6,1)
    else:
      j=np.insert(j,6,0)
    i += 1
  j=np.append(j[:5],j[6:]) #this cuts out position 5
  j=np.append(j[:4],j[5:]) #this cuts out position 4
  data2[d]=j
  d=d+1

data=data2
print(j)
print("data after normalization of teams")#this shows my year adjusted correctly
print(data.shape)
print(data)
print(data[0])
print(data[0][3])
print(data[1][3])
```

```
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
```

```
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
  0.   0.   0.   0.   0.   0.   0.]
data after normalization of teams
(38926, 595)
[[0 0 1 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 ...
 [0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]]
[0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0]
1
2
```

Spliting the Data

```
model3data=data
print(model3data[0][3])
print(data[0][3])
#shuffle the data so its spread out
np.random.shuffle(data)
#print(data.shape)
#print(data)

#this is for seperating the data from the labels
dataLabels = data[:,:3]
dataInfo = data[:,3:]
print("seperating Data labels and info")
print(dataLabels.shape)
print(dataLabels)
```

```
print(dataInfo.shape)
print(dataInfo)
```

```
splitIndex = int(0.4 * data.shape[0])
trainingDataLabels= dataLabels[splitIndex:, :]
trainingDataInfo= dataInfo[splitIndex:, :]
testDataLabels= dataLabels[:splitIndex, :]
testDataInfo= dataInfo[:splitIndex, :]
print("shape after split of training and testing")
print(trainingDataLabels.shape)
print(trainingDataInfo.shape)
print(testDataLabels.shape)
print(testDataInfo.shape)
```

```
#spliting test data into test data and training data
splitIndex = int(0.5 * testDataInfo.shape[0])
validationDataLabels= testDataLabels[splitIndex:, :]
validationDataInfo= testDataInfo[splitIndex:, :]
testDataLabels= testDataLabels[:splitIndex, :]
testDataInfo= testDataInfo[:splitIndex, :]
print("spliting validation and test data")
print(validationDataLabels.shape)
print(validationDataInfo.shape)
print(testDataLabels.shape)
print(testDataInfo.shape)
```

```
1
1
seperating Data labels and info
(38926, 3)
[[0 0 1]
 [1 0 0]
 [1 0 0]
 ...
 [0 1 0]
 [0 1 0]
 [1 0 0]]
(38926, 592)
[[108   0   0 ...   0   0   0]
 [ 98   0   0 ...   0   0   0]
 [137   0   0 ...   0   0   0]
 ...
 [133   0   0 ...   0   0   0]
 [129   0   0 ...   0   0   0]
 [130   0   0 ...   0   0   0]]
shape after split of training and testing
```

```
(23356, 3)
(23356, 592)
(15570, 3)
(15570, 592)
spliting validation and test data
(7785, 3)
(7785, 592)
(7785, 3)
(7785, 592)
```

Nieve approches

These are the nieve approches which could make good references to tell how my model is doing

- one nieve approach is just always guessing one of the three options
  - guesing Home Team win will have a 48.64% accuracy
  - guesing away Team win will have a 28.18% accuracy
  - guesing draw have a 23.18% accuracy

Model1

concept: The concept for this model is to be one of the most basic models i could get. It includes some basics its a sequential model we have some basic relu activated layers and my output layer has 3 options for a home win an away win and a draw. The model is being checksed for accuracy using rmsprop as my optimizer and catigorical crossentrapy as my loss function.

Result: the max accuracy i seem to get is 57% but validation just seems to jump a lot after a point

```
from keras import models
from keras import layers
from keras import optimizers
import matplotlib.pyplot as plt

NUM_EPOCHS = 25
MIDDLE_LAYER_SIZE = 200

model = models.Sequential()

model.add(layers.Dense(MIDDLE_LAYER_SIZE, activation='relu'))
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(1024,activation='relu'))
model.add(layers.Dense(3, activation="softmax"))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                metrics=['accuracy'])
```

```
history = model.fit(trainingDataInfo, trainingDataLabels, epochs=NUM_EPOCHS, batch_size=32,va
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['accuracy', 'validation'], loc='upper left')
plt.show()
```

```
Epoch 1/25
730/730 [==============================] - 20s 26ms/step - loss: 1.1064 - accuracy: 0.47
Epoch 2/25
730/730 [==============================] - 13s 18ms/step - loss: 1.0501 - accuracy: 0.48
Epoch 3/25
730/730 [==============================] - 12s 16ms/step - loss: 1.0515 - accuracy: 0.48
Epoch 4/25
730/730 [==============================] - 18s 24ms/step - loss: 1.0487 - accuracy: 0.48
Epoch 5/25
730/730 [==============================] - 14s 19ms/step - loss: 1.0233 - accuracy: 0.50
Epoch 6/25
730/730 [==============================] - 12s 16ms/step - loss: 0.9963 - accuracy: 0.53
Epoch 7/25
730/730 [==============================] - 17s 23ms/step - loss: 0.9791 - accuracy: 0.54
Epoch 8/25
730/730 [==============================] - 14s 19ms/step - loss: 0.9688 - accuracy: 0.54
Epoch 9/25
730/730 [==============================] - 14s 19ms/step - loss: 0.9611 - accuracy: 0.55
Epoch 10/25
730/730 [==============================] - 18s 24ms/step - loss: 0.9586 - accuracy: 0.55
Epoch 11/25
```

Model 2

concept: The concept for this model is about the same as the first I really just wanted to see how making the layers sigmoid would affect the results.

Result: this seemed to negatively affect my results as my accuracy barely increased since the first epoch with a max of 53%. Also validation data never goes up at first and jumps around a lot afterwards.

```
NUM_EPOCHS = 25
MIDDLE_LAYER_SIZE = 200

model = models.Sequential()

model.add(layers.Dense(MIDDLE_LAYER_SIZE, activation='sigmoid'))
model.add(layers.Dense(1024,activation='sigmoid'))
model.add(layers.Dense(512,activation='sigmoid'))
model.add(layers.Dense(256,activation='sigmoid'))
model.add(layers.Dense(3, activation="softmax"))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                metrics=['accuracy'])

history = model.fit(trainingDataInfo, trainingDataLabels, epochs=NUM_EPOCHS, batch_size=32,va
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.legend(['accuracy', 'validation'], loc='upper left')
plt.show()
```

```
Epoch 1/25
730/730 [==============================] - 17s 22ms/step - loss: 1.0679 - accuracy: 0.47
Epoch 2/25
730/730 [==============================] - 14s 19ms/step - loss: 1.0586 - accuracy: 0.48
Epoch 3/25
730/730 [==============================] - 16s 22ms/step - loss: 1.0557 - accuracy: 0.48
Epoch 4/25
730/730 [==============================] - 18s 25ms/step - loss: 1.0545 - accuracy: 0.48
Epoch 5/25
730/730 [==============================] - 14s 19ms/step - loss: 1.0526 - accuracy: 0.48
Epoch 6/25
730/730 [==============================] - 17s 23ms/step - loss: 1.0502 - accuracy: 0.48
Epoch 7/25
730/730 [==============================] - 17s 23ms/step - loss: 1.0475 - accuracy: 0.48
```

model 3

concept: I would like to try out taking away the tournament as a variable as I question how relevent it is to predicting a result

---

Result: the max accuracy i seem to get is 56% but validation just seems to jump a little after a point but less than the original one.

```
730/730 [==============================] - 14s 19ms/step - loss: 0.9773 - accuracy: 0.5
```

```
model3data=data
model3data=model3data[:,:498]

#print(model3data.shape)
#print(model3data[0])#this was used to comfirm the split went correctly
#this is for seperating the data from the labels
model3dataLabels = model3data[:,:3]
model3dataInfo = model3data[:,3:]

splitIndex = int(0.4 * model3data.shape[0])
model3trainingDataLabels= model3dataLabels[splitIndex:, :]
model3trainingDataInfo= model3dataInfo[splitIndex:, :]
model3testDataLabels= model3dataLabels[:splitIndex, :]
model3testDataInfo= model3dataInfo[:splitIndex, :]

#spliting test data into test data and training data
splitIndex = int(0.5 * model3testDataInfo.shape[0])
model3validationDataLabels= model3testDataLabels[splitIndex:, :]
model3validationDataInfo= model3testDataInfo[splitIndex:, :]
model3testDataLabels= model3testDataLabels[:splitIndex, :]
model3testDataInfo= model3testDataInfo[:splitIndex, :]

#the following is just model 1 so i can directally compare the results
NUM_EPOCHS = 25
MIDDLE_LAYER_SIZE = 200

model = models.Sequential()
```

```
model.add(layers.Dense(MIDDLE_LAYER_SIZE, activation='relu'))
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(1024,activation='relu'))
model.add(layers.Dense(3, activation="softmax"))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                metrics=['accuracy'])

history = model.fit(model3trainingDataInfo, model3trainingDataLabels, epochs=NUM_EPOCHS, batc
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['accuracy', 'validation'], loc='upper left')
plt.show()
```

```
Epoch 1/25
730/730 [==============================] - 16s 21ms/step - loss: 1.1106 - accuracy: 0.47
Epoch 2/25
730/730 [==============================] - 12s 16ms/step - loss: 1.0523 - accuracy: 0.48
Epoch 3/25
730/730 [==============================] - 15s 21ms/step - loss: 1.0490 - accuracy: 0.48
Epoch 4/25
730/730 [==============================] - 16s 22ms/step - loss: 1.0359 - accuracy: 0.49
Epoch 5/25
730/730 [==============================] - 12s 16ms/step - loss: 1.0023 - accuracy: 0.52
Epoch 6/25
730/730 [==============================] - 17s 23ms/step - loss: 0.9846 - accuracy: 0.53
Epoch 7/25
730/730 [==============================] - 14s 20ms/step - loss: 0.9745 - accuracy: 0.54
Epoch 8/25
730/730 [==============================] - 12s 16ms/step - loss: 0.9656 - accuracy: 0.54
Epoch 9/25
730/730 [==============================] - 15s 20ms/step - loss: 0.9616 - accuracy: 0.55
Epoch 10/25
730/730 [==============================] - 16s 22ms/step - loss: 0.9602 - accuracy: 0.55
Epoch 11/25
730/730 [==============================] - 12s 16ms/step - loss: 0.9597 - accuracy: 0.55
Epoch 12/25
730/730 [==============================] - 17s 23ms/step - loss: 0.9554 - accuracy: 0.55
Epoch 13/25
730/730 [==============================] - 15s 21ms/step - loss: 0.9512 - accuracy: 0.55
Epoch 14/25
730/730 [==============================] - 12s 17ms/step - loss: 0.9518 - accuracy: 0.55
Epoch 15/25
730/730 [==============================] - 18s 24ms/step - loss: 0.9504 - accuracy: 0.55
Epoch 16/25
```

Model 4

concept: I would like to try out editing the amounts that the validation training and test data have to see if I can adjust it so that maybe having more training data makes it work better

---

Result: oddly enough this doesnt effect the accuracy much the best I can get is still just around 56% accuracy with validation jumping a lot more as a well

```
Epoch 22/25
```

```
model4data=data

#this is for seperating the data from the labels
model4dataLabels = model4data[:,:3]
model4dataInfo = model4data[:,3:]

model4splitIndex = int(0.2 * model4data.shape[0])
model4trainingDataLabels= model4dataLabels[model4splitIndex:, :]
model4trainingDataInfo= model4dataInfo[model4splitIndex:, :]
model4testDataLabels= model4dataLabels[:model4splitIndex, :]
model4testDataInfo= model4dataInfo[:model4splitIndex, :]
```

```
#spliting test data into test data and training data
model4splitIndex = int(0.5 * model4testDataInfo.shape[0])
model4validationDataLabels= model4testDataLabels[model4splitIndex:, :]
model4validationDataInfo= model4testDataInfo[model4splitIndex:, :]
model4testDataLabels= model4testDataLabels[:model4splitIndex, :]
model4testDataInfo= model4testDataInfo[:model4splitIndex, :]

#the following is just model 1 so i can directally compare the results
NUM_EPOCHS = 25
MIDDLE_LAYER_SIZE = 200

model = models.Sequential()

model.add(layers.Dense(MIDDLE_LAYER_SIZE, activation='relu'))
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(1024,activation='relu'))
model.add(layers.Dense(3, activation="softmax"))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                metrics=['accuracy'])

print(trainingDataInfo.shape)
print(testDataInfo.shape)
print(validationDataInfo.shape)

history = model.fit(model4trainingDataInfo, model4trainingDataLabels, epochs=NUM_EPOCHS, batc
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['accuracy', 'validation'], loc='upper left')
plt.show()
```

```
(23356, 592)
(7785, 592)
(7785, 592)
Epoch 1/25
974/974 [==============================] - 20s 20ms/step - loss: 1.0906 - accuracy: 0.47
Epoch 2/25
974/974 [==============================] - 19s 19ms/step - loss: 1.0500 - accuracy: 0.48
Epoch 3/25
974/974 [==============================] - 18s 19ms/step - loss: 1.0436 - accuracy: 0.48
Epoch 4/25
974/974 [==============================] - 20s 20ms/step - loss: 1.0039 - accuracy: 0.52
Epoch 5/25
974/974 [==============================] - 17s 17ms/step - loss: 0.9816 - accuracy: 0.53
Epoch 6/25
974/974 [==============================] - 21s 21ms/step - loss: 0.9713 - accuracy: 0.54
Epoch 7/25
974/974 [==============================] - 15s 16ms/step - loss: 0.9623 - accuracy: 0.54
Epoch 8/25
974/974 [==============================] - 17s 17ms/step - loss: 0.9564 - accuracy: 0.55
Epoch 9/25
974/974 [==============================] - 21s 21ms/step - loss: 0.9518 - accuracy: 0.55
Epoch 10/25
974/974 [==============================] - 16s 16ms/step - loss: 0.9523 - accuracy: 0.55
Epoch 11/25
974/974 [==============================] - 22s 23ms/step - loss: 0.9475 - accuracy: 0.56
Epoch 12/25
974/974 [==============================] - 15s 16ms/step - loss: 0.9458 - accuracy: 0.55
Epoch 13/25
974/974 [==============================] - 19s 20ms/step - loss: 0.9439 - accuracy: 0.55
Epoch 14/25
974/974 [==============================] - 19s 19ms/step - loss: 0.9421 - accuracy: 0.56
Epoch 15/25
974/974 [==============================] - 19s 19ms/step - loss: 0.9419 - accuracy: 0.56
Epoch 16/25
974/974 [==============================] - 20s 21ms/step - loss: 0.9415 - accuracy: 0.56
Epoch 17/25
974/974 [==============================] - 18s 18ms/step - loss: 0.9392 - accuracy: 0.56
Epoch 18/25
974/974 [==============================] - 20s 21ms/step - loss: 0.9366 - accuracy: 0.56
Epoch 19/25
974/974 [==============================] - 17s 18ms/step - loss: 0.9372 - accuracy: 0.56
Epoch 20/25
974/974 [==============================] - 21s 21ms/step - loss: 0.9351 - accuracy: 0.56
Epoch 21/25
974/974 [==============================] - 17s 17ms/step - loss: 0.9349 - accuracy: 0.56
Epoch 22/25
974/974 [==============================] - 21s 22ms/step - loss: 0.9307 - accuracy: 0.57
Epoch 23/25
974/974 [==============================] - 16s 16ms/step - loss: 0.9300 - accuracy: 0.56
Epoch 24/25
974/974 [==============================] - 22s 23ms/step - loss: 0.9340 - accuracy: 0.56
```

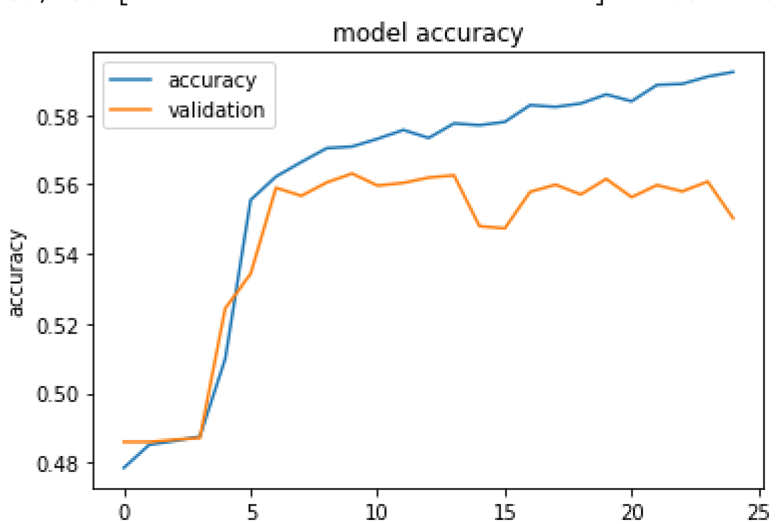
Model 5

concept: I want to look at the variety of optimizers and see if any of them work better than RMSprop

Result: Most of the optimizers seemed to not work at all or not work well, but Nadam seems to work the best with my highest accuracy at 58% and with valitation only jumping a little.

```
NUM_EPOCHS = 25
MIDDLE_LAYER_SIZE = 200

model = models.Sequential()

model.add(layers.Dense(MIDDLE_LAYER_SIZE, activation='relu'))
model.add(layers.Dense(256,activation='relu'))
model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(1024,activation='relu'))
model.add(layers.Dense(3, activation="softmax"))

#Adagrad #as an optimizer didnt work
#SGD as a optimizer didnt work
#Adam as an optimizer did not work
#Adadelta as a optimizer did not work
#Ftrl as an optimizer did not work

#Adamax seems to work with an accuracy arount 57% and little jumps in validation
#rmsprop as an optimizer does work
#Nadam as an optimizer works really well
model.compile(optimizer='Nadam', loss='categorical_crossentropy', metrics=['accuracy']) #this

history = model.fit(trainingDataInfo, trainingDataLabels, epochs=NUM_EPOCHS, batch_size=32,va
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['accuracy', 'validation'], loc='upper left')
plt.show()
```

```
730/730 [==============================] - 21s 28ms/step - loss: 1.0475 - accuracy: 0.48
Epoch 5/25
730/730 [==============================] - 17s 23ms/step - loss: 1.0088 - accuracy: 0.56
Epoch 6/25
730/730 [==============================] - 20s 27ms/step - loss: 0.9499 - accuracy: 0.55
Epoch 7/25
730/730 [==============================] - 18s 25ms/step - loss: 0.9367 - accuracy: 0.56
Epoch 8/25
730/730 [==============================] - 19s 26ms/step - loss: 0.9279 - accuracy: 0.56
Epoch 9/25
730/730 [==============================] - 20s 27ms/step - loss: 0.9233 - accuracy: 0.57
Epoch 10/25
730/730 [==============================] - 17s 23ms/step - loss: 0.9170 - accuracy: 0.57
Epoch 11/25
730/730 [==============================] - 21s 29ms/step - loss: 0.9120 - accuracy: 0.57
Epoch 12/25
730/730 [==============================] - 16s 22ms/step - loss: 0.9101 - accuracy: 0.57
Epoch 13/25
730/730 [==============================] - 19s 26ms/step - loss: 0.9052 - accuracy: 0.57
Epoch 14/25
730/730 [==============================] - 19s 26ms/step - loss: 0.9018 - accuracy: 0.57
Epoch 15/25
730/730 [==============================] - 20s 28ms/step - loss: 0.8985 - accuracy: 0.57
Epoch 16/25
730/730 [==============================] - 17s 24ms/step - loss: 0.8956 - accuracy: 0.57
Epoch 17/25
730/730 [==============================] - 21s 28ms/step - loss: 0.8933 - accuracy: 0.58
Epoch 18/25
730/730 [==============================] - 15s 21ms/step - loss: 0.8912 - accuracy: 0.58
Epoch 19/25
730/730 [==============================] - 15s 21ms/step - loss: 0.8856 - accuracy: 0.58
Epoch 20/25
730/730 [==============================] - 21s 29ms/step - loss: 0.8837 - accuracy: 0.58
Epoch 21/25
730/730 [==============================] - 17s 23ms/step - loss: 0.8823 - accuracy: 0.58
Epoch 22/25
730/730 [==============================] - 19s 26ms/step - loss: 0.8789 - accuracy: 0.58
Epoch 23/25
730/730 [==============================] - 16s 22ms/step - loss: 0.8758 - accuracy: 0.58
Epoch 24/25
730/730 [==============================] - 23s 31ms/step - loss: 0.8738 - accuracy: 0.59
Epoch 25/25
730/730 [==============================] - 16s 22ms/step - loss: 0.8707 - accuracy: 0.59
```



epoch

7m 42s completed at 7:31 PM