demo

May 3, 2018

1 Comparing Feed-Forward and Convolutional Neural Networks in the Task of Classifying Poker Hands

Introduction: Poker has always shown itself to be a challenging way to test one's luck and grit against odds that are exceptionally low. There are many different variants that slightly alter the odds. The poker variant 5 card draw has 2,598,960 different hand combinations in a standard 52 card deck. With this many possible hands it is easy to see how difficult it could be to get even a decent hand in a single 5 card draw. The reason that these odds do not hinder most players is that even a terrible hand has a chance to win with a good bluff. Great human poker players have learned when to bluff and how to tell when another player is bluffing. In order to compete, computerized robot players must focus on a categorical approach based on the odds and significant training. The overall aim of our project is to discover which techniques and neural networks performed the best at classifying poker hands.

Our team trained two different types of Neural Networks and compared their performance in order to assess which was superior for the problem at hand. We will be going over them and learning about them in brief in this demo.

Installation: In the event that you are the explorative type and ended up here before you read the readme, here are some quick steps to get going with this interactive demo:

Step 1: Install Jupyter here - http://jupyter.readthedocs.io/en/latest/install.html

Step 2: Install the package Treys here - https://github.com/ihendley/treys/tree/master/treys This will be used to handle our deck of cards and other related things.

Step 3: Make a local copy of this repo.

How does this Jupyter Notebook thing work?: Jupyter notebooks work almost the exact same way as a normal programming environment, with one very big yet simple difference. In a Jupyter Notebook everything is divided into modular cells, so you can just run individual segments of code instead of the entire program. All you need to do to run a cell of code is to click on it and then press cntrl-enter.

Otherwise everything should be relatively intuitive.

What is a Neural Network?: Maureen Caudill succinctly answered this question in 1989 and explained Neural Networks as: "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs". -- "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989.

Want to learn more? Visit: http://www.explainthatstuff.com/introduction-to-neural-networks.html

Data: In order to train a Neural Network you must first have a dataset to learn from. Our dataset is available within our repository here: https://github.com/CSCI4850/S18-team4-project/tree/master/data

It is simply every possible hand in poker.

Feed-Forward Neural Networks: The first type of Neural Network we trained was a Feed-Forward Neural Network. Feed-Forward Neural Networks are a type of Neural Network where the connections from unit to unit don't form a cycle.

If you want to learn more in-depth about Feed-Forward Networks, please visit: https://www.researchgate.net/publication/228394623_A_brief_review_of_feed-forward_neural_networks

Our Feed-Forward Neural Network: Let's look at our results! We won't be training the network in this demo as it can take quite a long time, but images of the entire process are provided.

If you want to see the actual file where this was done, please head to: https://github.com/CSCI4850/S18-team4-project/blob/master/FeedNN.ipynb

Okay, so let's get our data and set it up in a format our network can understand!

```
#setting attribute info, column names for data set.
# S1 refers to Suit of card #1
# C1 refers to Rank of card #1 and so on.
# see data set attributes (poker-hand.names)
column = ['S1', 'C1', 'S2', 'C2', 'S3','C3','S4','C4','S5','C5','class']
suit = ['S1','S2','S3','S4','S5']
rank = ['C1','C2','C3','C4','C5']
_class = ['class']


#labeling
poker_train = np.array((pandas.read_table(train_path,names=column,
                             #delim_whitespace=True,
                             header=None,sep =',')))
poker_tst = np.array((pandas.read_table(test_path,names=column,
                             #delim_whitespace=True,
                             header=None,sep =',')))

print(poker_train.shape,'train')
print(poker_tst.shape,'test')
```

```
(25010, 11) train
(1000000, 11) test
```

Next, we have to build our model...

Here's the part everyone always talks about, training!

And finally here you can see the results of our training!

As you can see we got 98% accuracy, which is really quite good!

In [21]:

```
#Building MODEL

#26 variables, also convert into category and do our one hot encoding.

#X = np.array(poker_train.drop('class',axis=1))
X = poker_train[:,0:10]
labels = poker_train[:,10]
#one hot encoding categorical values
#get_dummies creates dummy/indicator variables (1 or 0).
#Y2 = pandas.get_dummies(poker_train['class'].astype('category'))
Y = keras.utils.to_categorical(labels,
                               len(np.unique(labels)))
#print(Y)

#validation set
#test_X = np.array(poker_tst.drop('class',axis=1))
test_X = poker_tst[:,0:10]
olabels = poker_tst[:,10]
#test_y2 = pandas.get_dummies(poker_tst['class'].astype('category'))
test_Y = keras.utils.to_categorical(olabels,
                               len(np.unique(olabels)))

print(test_X.shape)
print(test_Y.shape)

#######    MODEL ########

model = keras.models.Sequential()

model.add(keras.layers.Dense(512,input_dim=10,activation='relu'))
model.add(keras.layers.Dense(512,activation='relu'))
model.add(keras.layers.Dense(512,activation='relu'))

model.add(keras.layers.Dense(10,activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

model.summary()
```

```
(1000000, 10)
(1000000, 10)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 512)               5632
_________________________________________________________________
dense_18 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_19 (Dense)             (None, 512)               262656
_________________________________________________________________
dense_20 (Dense)             (None, 10)                5130
=================================================================
Total params: 536,074
Trainable params: 536,074
Non-trainable params: 0
_________________________________________________________________
```

In [22]:
```
batch_size = 100
epochs = 25
history = model.fit(X,Y,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(test_X, test_Y))
```

```
Train on 25010 samples, validate on 1000000 samples
Epoch 1/25
25010/25010 [==============================] - 54s 2ms/step - loss: 0.9968 - acc: 0.5142 - val_loss: 0.9614 - val_acc:
0.5411
Epoch 2/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.9559 - acc: 0.5457 - val_loss: 0.9553 - val_acc:
0.5288
Epoch 3/25
25010/25010 [==============================] - 54s 2ms/step - loss: 0.9439 - acc: 0.5536 - val_loss: 0.9353 - val_acc:
0.5608
Epoch 4/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.9356 - acc: 0.5611 - val_loss: 0.9291 - val_acc:
0.5636
Epoch 5/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.9259 - acc: 0.5672 - val_loss: 0.9225 - val_acc:
0.5701
Epoch 6/25
25010/25010 [==============================] - 54s 2ms/step - loss: 0.9088 - acc: 0.5769 - val_loss: 0.9012 - val_acc:
0.5861
Epoch 7/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.8882 - acc: 0.5922 - val_loss: 0.8877 - val_acc:
0.5961
Epoch 8/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.8600 - acc: 0.6125 - val_loss: 0.8630 - val_acc:
0.6158
Epoch 9/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.8386 - acc: 0.6246 - val_loss: 0.8439 - val_acc:
0.6179
Epoch 10/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.8062 - acc: 0.6449 - val_loss: 0.8036 - val_acc:
0.6488
Epoch 11/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.7704 - acc: 0.6652 - val_loss: 0.8120 - val_acc:
0.6366
Epoch 12/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.7271 - acc: 0.6885 - val_loss: 0.7343 - val_acc:
0.6917
Epoch 13/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.6801 - acc: 0.7194 - val_loss: 0.7220 - val_acc:
0.6939
```

```
Epoch 14/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.6261 - acc: 0.7423 - val_loss: 0.6391 - val_acc:
0.7462
Epoch 15/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.5820 - acc: 0.7646 - val_loss: 0.5839 - val_acc:
0.7707
Epoch 16/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.5324 - acc: 0.7883 - val_loss: 0.5395 - val_acc:
0.7863
Epoch 17/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.4759 - acc: 0.8131 - val_loss: 0.5289 - val_acc:
0.7913
Epoch 18/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.4242 - acc: 0.8404 - val_loss: 0.4404 - val_acc:
0.8386
Epoch 19/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.3652 - acc: 0.8661 - val_loss: 0.4326 - val_acc:
0.8350
Epoch 20/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.3072 - acc: 0.8888 - val_loss: 0.3459 - val_acc:
0.8724
Epoch 21/25
25010/25010 [==============================] - 54s 2ms/step - loss: 0.2649 - acc: 0.9084 - val_loss: 0.3298 - val_acc:
0.8848
Epoch 22/25
25010/25010 [==============================] - 54s 2ms/step - loss: 0.2276 - acc: 0.9236 - val_loss: 0.3265 - val_acc:
0.8830
Epoch 23/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.1762 - acc: 0.9436 - val_loss: 0.2370 - val_acc:
0.9185
Epoch 24/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.1604 - acc: 0.9511 - val_loss: 0.1849 - val_acc:
0.9436
Epoch 25/25
25010/25010 [==============================] - 53s 2ms/step - loss: 0.1066 - acc: 0.9719 - val_loss: 0.1370 - val_acc:
0.9593
```

In [23]:

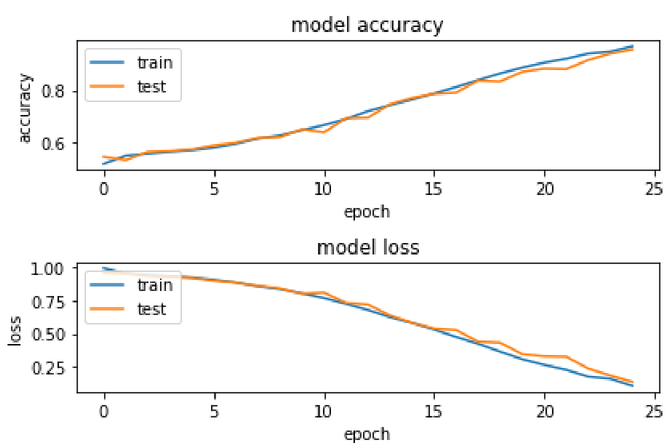
```
#   Model Accuracy
score = model.evaluate(X, Y, verbose=1)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

```
25010/25010 [==============================] - 2s 99us/step
Test loss: 0.0795012926328
Test accuracy: 0.980567772891
```

In [24]:
```
#   Plotting
plt.figure(1)
# summarize history for accuracy
plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.tight_layout()
plt.show()
```



Let's play some poker against the Feed-Forward Neural Network! : All of this work and no fun? Time to see what the above code can really do in a game against yourself in a slightly more tangible example.

Please note that the neural network knows how to classify hands, this means it can look at 5 cards and tell you if the hand is a high card, 2 of a kind, royal flush, etc.

All decision making based on what the neural network says the hand is has been hard coded.

Don't worry about the following code blocks, just click on them one at a time and then press cntrl+enter(this runs the code in the block) on each of them sequentially and the game will begin!

This can also double as a nice example of modular code in jupyter if you're new to this. (:

Setup game:

```
In [169]: from treys import Deck,Card,Evaluator
          import keras
          import pandas as pd
          import numpy as np
          from keras.models import load_model
```

```
In [170]: deck=Deck()
          evaluator=Evaluator()
          trans = {'Ah': [1,1], '2h': [1,2], '3h': [1,3], '4h': [1,4],'5h': [1,5], '6h': [1,6]
                  'As': [2,1], '2s': [2,2], '3s': [2,3], '4s': [2,4],'5s': [2,5], '6s': [2,6],
                  'Ad': [3,1], '2d': [3,2], '3d': [3,3], '4d': [3,4],'5d': [3,5], '6d': [3,6],
                  'Ac': [4,1], '2c': [4,2], '3c': [4,3], '4c': [4,4],'5c': [4,5], '6c': [4,6],
                  } #dict creation
```

Get and show initial hands:

```
In [171]: player1_hand=deck.draw(5) #draw initial hands
          nn_hand=deck.draw(5)

          player1_hand.sort()

          print ("Your hand: ")# prints initial hands
          Card.print_pretty_cards(player1_hand)
```

```
Your hand:
 [5],[6],[8],[T],[A]
```

Please specify which cards you would like to discard.

Your cards are ordered 0-4 from left to right.

```
In [172]: new_hand=[0,0,0,0,0]
          print('Please enter yes or no to indicate your choice:')
          print('Would you like to discard your card at position 0?','\n')
          zero=input()
          print('Would you like to discard your card at position 1?','\n')
          one=input()
          print('Would you like to discard your card at position 2?','\n')
```

```
    two=input()
    print('Would you like to discard your card at position 3?','\n')
    three=input()
    print('Would you like to discard your card at position 4?','\n')
    four=input()


    if(zero.lower()=='yes'):
        new_hand[0]=deck.draw(1)
    if(one.lower()=='yes'):
        new_hand[1]=deck.draw(1)
    if(two.lower()=='yes'):
        new_hand[2]=deck.draw(1)
    if(three.lower()=='yes'):
        new_hand[3]=deck.draw(1)
    if(four.lower()=='yes'):
        new_hand[4]=deck.draw(1)

    for i in range(0,5):
        if(new_hand[i]!=0):
            player1_hand[i]=new_hand[i]

    print("Your current hand is:" ,'\n')
    Card.print_pretty_cards(player1_hand)
```

```
Please enter yes or no to indicate your choice:
Would you like to discard your card at position 0?

Would you like to discard your card at position 1?

Would you like to discard your card at position 2?

Would you like to discard your card at position 3?

Would you like to discard your card at position 4?

Your current hand is:

 [5],[6],[8],[4],[9]
```

Note: The cards above will not display correctly in pdf format.

Note: You might want to wait ~20 seconds for the code in this next cell to finish...

```
In [173]: #determine what the NN thinks of the hand
          hand_for_nn=[]
          for i in range(0,5):
```

```
    hand_for_nn=(hand_for_nn+trans[Card.int_to_str(nn_hand[i])])



model = load_model('demo/ffbnn.h5') #ffbnn.h5 is feed forward NN

handarray = np.array([hand_for_nn]) #convert hand to np array for input

preds = model.predict(handarray) #calculates probabilities of each class from NN

label=np.argmax(preds)

#if nothing in hand
if (label==0):

    new_hand=deck.draw(5)#draw 5 new cards


#if one pair
if (label==1):
    nn_hand.sort() #sort arr by val


    cards_to_keep=[0,0] #array where will store the cards from hand to keep, init as
    new_hand=[0,0] #actual arr where cards will be stored as their int encodings
    count=2
    for i in range(0,4):
        current=Card.int_to_str(nn_hand[i]) #get card as str
        current_num=nn_hand[i]
        for x in range(0,5):
            if (x!=i): #if we're not on the card we are checking for a match for
                check=Card.int_to_str(nn_hand[x]) #get card to check against current
                check_num=nn_hand[x]
                if(check[0]==current[0]): #if the cards have the same value aka are
                    cards_to_keep[0]=check
                    cards_to_keep[1]=current

                    new_hand[0]=check_num
                    new_hand[1]=current_num



    draw_3=deck.draw(3)#draw 3 new cards
    new_hand=new_hand+draw_3 #add the 3 new cards to the hand, keeping the pairs

    Card.print_pretty_cards(new_hand)
    #for y in range(0,5):
     #   print(Card.int_to_str(new_hand[y]))
```

```
if (label==2): #two pairs




    nn_hand.sort() #sort arr by val

    cards_to_keep=[0,0,0,0] #array where will store the cards from hand to keep, ini
    new_hand=[0,0,0,0] #actual arr where cards will be stored as their int encodings
    count=0
    has_1_pair=0
    for i in range(0,4):
        current=Card.int_to_str(nn_hand[i]) #get card as str
        current_num=nn_hand[i]
        if(i==4):
            break
        for x in range(0,5):
            if (x!=i): #if we're not on the card we are checking for a match for
                check=Card.int_to_str(nn_hand[x]) #get card to check against current
                check_num=nn_hand[x]
                if(check[0]==current[0] and (check_num!=new_hand[0] and check_num!=ne
                    for y in range(0,4):
                        if (check_num not in new_hand): #man python is nice
                            cards_to_keep[count]=check
                            new_hand[count]=check_num
                            count+=1
                            cards_to_keep[count]=current
                            new_hand[count]=current_num
                            count+=1
                            #print(len(new_hand))


    draw_1=deck.draw(1)#draw 1 new card
    drawn=[draw_1]
    new_hand=new_hand+drawn #add the new card to the hand, keeping the pairs



if (label==3): #three of a kind
    nn_hand.sort() #sort arr by val

    cards_to_keep=[0,0,0] #array where will store the cards from hand to keep, init
    new_hand=[0,0,0] #actual arr where cards will be stored as their int encodings
    count=0
    for i in range(0,4):
        current=Card.int_to_str(nn_hand[i]) #get card as str
```

```
                current_num=nn_hand[i]
                for x in range(0,4):
                    if (x!=i): #if we're not on the card we are checking for a match for
                        check=Card.int_to_str(nn_hand[x]) #get card to check against current
                        check_num=nn_hand[x]
                        if(check[0]==current[0]): #if the cards have the same value aka are
                            cards_to_keep[0]=check
                            cards_to_keep[1]=current

                            new_hand[0]=check_num
                            new_hand[1]=current_num


                            #find third card
                            for y in range(0,5):
                                check2=Card.int_to_str(nn_hand[y])
                                if(check2[0]==current[0] and (check2[1]!=check[1] and check2
                                    store=nn_hand[y]

                                    new_hand[2]=new_hand[1]
                                    new_hand[1]=store
                                    cards_to_keep[2]=check2



            draw_2=deck.draw(2)#draw 2 new cards
            new_hand=new_hand+draw_2 #add the 2 new cards to the hand, keeping the pairs




        if(label>3):
            print("")
```

In [174]:
```
#using rankings of evaluator from Treys to determine winner

#get scores
player1_score=evaluator._five(player1_hand)
nn_score=evaluator._five(new_hand)

if (nn_score<player1_score):
    print("NN wins!")
else:
    print("YOU win!")

print ("Player1 hand: ")# prints hands
Card.print_pretty_cards(player1_hand)
```

```
        print ("NN hand: ")
        Card.print_pretty_cards(new_hand)
```

```
NN wins!
Player1 hand:
 [5],[6],[8],[4],[9]
NN hand:
 [4],[8],[Q],[9],[A]
```

Convolutional Neural Networks: Normally in neural networks the input is a vector, but in Convolutional Neural Networks the input is divided into multiple channels. This input is then used to produce a Convolutional Layer that are then trained through back-propagation.

Want to learn more about Convolutional Neural Networks? Visit: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Our Convolutional Neural Network! : Again, we won't be training any network in this demo because it can take an obscene amount of time, so screenshots of the process will be provided.

Setting up the data:

```
"""
https://archive.ics.uci.edu/ml/datasets/Poker+Hand
"""

#you will prob have to change path location to load data.
train_path = "http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-hand-training-true.data"
test_path = "http://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-hand-testing.data"

#setting attribute info, column names for data set.
# S1 refers to Suit of card #1
# C1 refers to Rank of card #1 and so on.
# see data set attributes (poker-hand.names)
column = ['S1', 'C1', 'S2', 'C2', 'S3','C3','S4','C4','S5','C5','class']
suit = ['S1','S2','S3','S4','S5']
rank = ['C1','C2','C3','C4','C5']
_class = ['class']

#labeling
poker_train = np.array((pandas.read_table(train_path,names=column,
                            #delim_whitespace=True,
                            header=None,sep =',')))
poker_tst = np.array((pandas.read_table(test_path,names=column,
                            #delim_whitespace=True,
                            header=None,sep =',')))

print(poker_train.shape,'train')
print(poker_tst.shape,'test')

#for each attribute of suit of cards we will convert var into category
#this will allow us to do one hot encoding / dummy encoding later on
#for i in suit:
#    poker_train[i] = poker_train[i].astype('category')
#    poker_tst[i] = poker_tst[i].astype('category')
# suits are now one hot encoded
#poker_train = pandas.get_dummies(poker_train)
#poker_tst = pandas.get_dummies(poker_tst)
#print(poker_tst.shape)
#print(poker_train.shape)
```

```
(25010, 11) train
(1000000, 11) test
```

Build the CNN model:

Time to do the training!

Results:

As you can see, this performed marginally worse than the Feed-Forward Neural Network.

Let's play some poker against the Convolutional Neural Network! : Time to take on another AI.

Remember it plays the game in the same way as the Feed-Forward Neural Network.

```python
#Building MODEL

#26 variables, also convert into category and do our one hot encoding.

#X = np.array(poker_train.drop('class',axis=1))
X = poker_train[:,0:10]
labels = poker_train[:,10]
#one hot encoding categorical values
#get_dummies creates dummy/indicator variables (1 or 0).
#Y2 = pandas.get_dummies(poker_train['class'].astype('category'))
Y = keras.utils.to_categorical(labels,
                               len(np.unique(labels)))
#print(Y)
#validation set
#test_X = np.array(poker_tst.drop('class',axis=1))
test_X = poker_tst[:,0:10]
olabels = poker_tst[:,10]
#test_y2 = pandas.get_dummies(poker_tst['class'].astype('category'))
test_Y = keras.utils.to_categorical(olabels,
                               len(np.unique(olabels)))
tempX = X[:,:,None]
tempY = Y[:,:,None]

tempX.reshape(X.shape[0],X.shape[1],1)
#temp.reshape(temp.shape[0],temp.shape[2],temp.shape[1])

tempY.reshape(Y.shape[0],Y.shape[1],1)

print(tempX.shape)
print(tempY.shape)

#######    MODEL ########

model = keras.Sequential()
model.add(keras.layers.Conv1D(512, kernel_size=(8),
        activation='relu',input_shape=[tempX.shape[1],
        tempX.shape[2]]))
model.add(keras.layers.Conv1D(512, (2), activation='relu'))
model.add(keras.layers.MaxPooling1D(pool_size=(2)))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512, activation='relu'))
model.add(keras.layers.Dropout(0.25))
model.add(keras.layers.Dense(tempY.shape[1], activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
        optimizer=keras.optimizers.Adam(),
        metrics=['accuracy'])
model.summary()
```

```
(25010, 10, 1)
(25010, 10, 1)
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 3, 512)            4608      
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 2, 512)            524800    
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 1, 512)            0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130      
=================================================================
Total params: 797,194
Trainable params: 797,194
Non-trainable params: 0
_________________________________________________________________
```

```
batch_size = 100
epochs = 25
history = model.fit(tempX,Y,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_split=.33)
```

```
Train on 16756 samples, validate on 8254 samples
Epoch 1/25
16756/16756 [==============================] - 10s 588us/step - loss: 1.0228 - acc: 0.4849 - val_loss: 0.9828 - val_acc: 0.4977
Epoch 2/25
16756/16756 [==============================] - 9s 544us/step - loss: 0.9846 - acc: 0.5058 - val_loss: 0.9675 - val_acc: 0.5309
Epoch 3/25
16756/16756 [==============================] - 9s 538us/step - loss: 0.9701 - acc: 0.5273 - val_loss: 0.9627 - val_acc: 0.5265
Epoch 4/25
16756/16756 [==============================] - 9s 544us/step - loss: 0.9562 - acc: 0.5428 - val_loss: 0.9479 - val_acc: 0.5476
Epoch 5/25
16756/16756 [==============================] - 9s 533us/step - loss: 0.9471 - acc: 0.5579 - val_loss: 0.9379 - val_acc: 0.5531
Epoch 6/25
16756/16756 [==============================] - 9s 552us/step - loss: 0.9351 - acc: 0.5680 - val_loss: 0.9569 - val_acc: 0.5240
Epoch 7/25
16756/16756 [==============================] - 9s 541us/step - loss: 0.9250 - acc: 0.5714 - val_loss: 0.9196 - val_acc: 0.5606
Epoch 8/25
16756/16756 [==============================] - 9s 548us/step - loss: 0.9070 - acc: 0.5864 - val_loss: 0.9369 - val_acc: 0.5550
Epoch 9/25
16756/16756 [==============================] - 9s 563us/step - loss: 0.8963 - acc: 0.5908 - val_loss: 0.8974 - val_acc: 0.5876
Epoch 10/25
16756/16756 [==============================] - 9s 545us/step - loss: 0.8779 - acc: 0.6034 - val_loss: 0.8833 - val_acc: 0.5903
Epoch 11/25
16756/16756 [==============================] - 9s 530us/step - loss: 0.8621 - acc: 0.6166 - val_loss: 0.8658 - val_acc: 0.6083
Epoch 12/25
16756/16756 [==============================] - 9s 532us/step - loss: 0.8410 - acc: 0.6263 - val_loss: 0.8580 - val_acc: 0.6069
Epoch 13/25
16756/16756 [==============================] - 9s 525us/step - loss: 0.8188 - acc: 0.6323 - val_loss: 0.8209 - val_acc: 0.6387
Epoch 14/25
16756/16756 [==============================] - 9s 541us/step - loss: 0.7883 - acc: 0.6536 - val_loss: 0.7815 - val_acc: 0.6596
Epoch 15/25
16756/16756 [==============================] - 9s 527us/step - loss: 0.7571 - acc: 0.6718 - val_loss: 0.7336 - val_acc: 0.6918
Epoch 16/25
16756/16756 [==============================] - 9s 535us/step - loss: 0.7099 - acc: 0.6970 - val_loss: 0.6826 - val_acc: 0.7160
Epoch 17/25
16756/16756 [==============================] - 9s 526us/step - loss: 0.6266 - acc: 0.7431 - val_loss: 0.6198 - val_acc: 0.7453
```

```
Epoch 18/25
16756/16756 [==============================] - 9s 524us/step - loss: 0.5380 - acc: 0.7903 - val_loss: 0.5067 - val_acc: 0.8137
Epoch 19/25
16756/16756 [==============================] - 9s 527us/step - loss: 0.4658 - acc: 0.8244 - val_loss: 0.4651 - val_acc: 0.8237
Epoch 20/25
16756/16756 [==============================] - 9s 526us/step - loss: 0.3566 - acc: 0.8704 - val_loss: 0.3380 - val_acc: 0.8805
Epoch 21/25
16756/16756 [==============================] - 9s 528us/step - loss: 0.3105 - acc: 0.8868 - val_loss: 0.2987 - val_acc: 0.8974
Epoch 22/25
16756/16756 [==============================] - 9s 525us/step - loss: 0.2575 - acc: 0.9085 - val_loss: 0.2560 - val_acc: 0.9055
Epoch 23/25
16756/16756 [==============================] - 9s 543us/step - loss: 0.2170 - acc: 0.9243 - val_loss: 0.2661 - val_acc: 0.9043
Epoch 24/25
16756/16756 [==============================] - 9s 543us/step - loss: 0.1912 - acc: 0.9346 - val_loss: 0.1737 - val_acc: 0.9437
Epoch 25/25
16756/16756 [==============================] - 10s 586us/step - loss: 0.1403 - acc: 0.9546 - val_loss: 0.1584 - val_acc: 0.9535
```

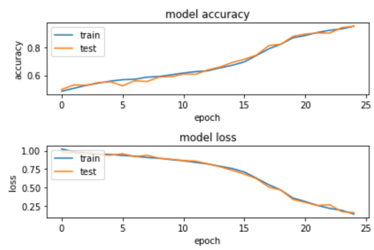
```
#   Model Accuracy
score = model.evaluate(tempX, Y, verbose=1)
print('Test loss:',score[0])
print('Test accuracy:',score[1])
```

```
25010/25010 [==============================] - 6s 249us/step
Test loss: 0.12084040802354958
Test accuracy: 0.9654538184726109
```

```python
#   Plotting
plt.figure(1)
# summarize history for accuracy
plt.subplot(211)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

# summarize history for loss

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.tight_layout()
plt.show()
```



title

```python
In [175]: from treys import Deck,Card,Evaluator
          import keras
          import pandas as pd
          import numpy as np
          from keras.models import load_model

          deck=Deck()
          evaluator=Evaluator()
          trans = {'Ah': [1,1], '2h': [1,2], '3h': [1,3], '4h': [1,4],'5h': [1,5], '6h': [1,6]
                  'As': [2,1], '2s': [2,2], '3s': [2,3], '4s': [2,4],'5s': [2,5], '6s': [2,6],
                  'Ad': [3,1], '2d': [3,2], '3d': [3,3], '4d': [3,4],'5d': [3,5], '6d': [3,6],
                  'Ac': [4,1], '2c': [4,2], '3c': [4,3], '4c': [4,4],'5c': [4,5], '6c': [4,6],
                  } #dict creation
```

```python
In [176]: player1_hand=deck.draw(5) #draw initial hands
          nn_hand=deck.draw(5)

          print ("Your hand: ")# prints initial hands
          Card.print_pretty_cards(player1_hand)
```

```
Your hand:
 [5],[6],[6],[8],[9]
```

Please specify which cards you would like to discard.
Your cards are ordered 0-4 from left to right.

```python
In [177]: new_hand=[0,0,0,0,0]
          print('Please enter yes or no to indicate your choice:')
```

```
    print('Would you like to discard your card at position 0?','\n')
    zero=input()
    print('Would you like to discard your card at position 1?','\n')
    one=input()
    print('Would you like to discard your card at position 2?','\n')
    two=input()
    print('Would you like to discard your card at position 3?','\n')
    three=input()
    print('Would you like to discard your card at position 4?','\n')
    four=input()
    if(zero.lower()=='yes'):
        new_hand[0]=deck.draw(1)
    if(one.lower()=='yes'):
        new_hand[1]=deck.draw(1)
    if(two.lower()=='yes'):
        new_hand[2]=deck.draw(1)
    if(three.lower()=='yes'):
        new_hand[3]=deck.draw(1)
    if(four.lower()=='yes'):
        new_hand[4]=deck.draw(1)

    for i in range(0,5):
        if(new_hand[i]!=0):
            player1_hand[i]=new_hand[i]

    print("Your current hand is:" ,'\n')
    Card.print_pretty_cards(player1_hand)
```

```
Please enter yes or no to indicate your choice:
Would you like to discard your card at position 0?

Would you like to discard your card at position 1?

Would you like to discard your card at position 2?

Would you like to discard your card at position 3?

Would you like to discard your card at position 4?

Your current hand is:

 [3],[6],[6],[A],[7]
```

Note: You might want to wait ~20 seconds for the code in this next cell to finish...

```
In [178]: #determine what the CNN thinks of the hand
          hand_for_nn2=[]
```

```
for i in range(0,5):
    hand_for_nn2=(hand_for_nn2+trans[Card.int_to_str(nn_hand[i])])



model2 = load_model('demo/cnn.h5') #ffbnn.h5 is feed forward NN

handarray2 = np.array([hand_for_nn2]) #convert hand to np array for input


handarray2_t = handarray2[:,:,None]

handarray2_t.reshape(handarray2.shape[0],handarray2.shape[1],1)

preds2 = model2.predict(handarray2_t) #calculates probabilities of each class from N

label=np.argmax(preds2)

#if nothing in hand
if (label==0):

    new_hand=deck.draw(5)#draw 5 new cards


#if one pair
if (label==1):
    nn_hand.sort() #sort arr by val

    cards_to_keep=[0,0] #array where will store the cards from hand to keep, init as
    new_hand=[0,0] #actual arr where cards will be stored as their int encodings
    count=2
    for i in range(0,4):
        current=Card.int_to_str(nn_hand[i]) #get card as str
        current_num=nn_hand[i]
        for x in range(0,5):
            if (x!=i): #if we're not on the card we are checking for a match for
                check=Card.int_to_str(nn_hand[x]) #get card to check against current
                check_num=nn_hand[x]
                if(check[0]==current[0]): #if the cards have the same value aka are
                    cards_to_keep[0]=check
                    cards_to_keep[1]=current

                    new_hand[0]=check_num
                    new_hand[1]=current_num



    draw_3=deck.draw(3)#draw 3 new cards
```

```
        new_hand=new_hand+draw_3 #add the 3 new cards to the hand, keeping the pairs


    if (label==2): #two pairs





        nn_hand.sort() #sort arr by val

        cards_to_keep=[0,0,0,0] #array where will store the cards from hand to keep, ini
        new_hand=[0,0,0,0] #actual arr where cards will be stored as their int encodings
        count=0
        has_1_pair=0
        for i in range(0,4):
            current=Card.int_to_str(nn_hand[i]) #get card as str
            current_num=nn_hand[i]
            if(i==4):
                break
            for x in range(0,5):
                if (x!=i): #if we're not on the card we are checking for a match for
                    check=Card.int_to_str(nn_hand[x]) #get card to check against current
                    check_num=nn_hand[x]
                    if(check[0]==current[0] and (check_num!=new_hand[0] and check_num!=ne
                        for y in range(0,4):
                            if (check_num not in new_hand): #man python is nice
                                cards_to_keep[count]=check
                                new_hand[count]=check_num
                                count+=1
                                cards_to_keep[count]=current
                                new_hand[count]=current_num
                                count+=1
                                #print(len(new_hand))


        draw_1=deck.draw(1)#draw 1 new card
        drawn=[draw_1]
        new_hand=new_hand+drawn #add the new card to the hand, keeping the pairs


    if (label==3): #three of a kind
        nn_hand.sort() #sort arr by val

        cards_to_keep=[0,0,0] #array where will store the cards from hand to keep, init
        new_hand=[0,0,0] #actual arr where cards will be stored as their int encodings
        count=0
        for i in range(0,4):
            current=Card.int_to_str(nn_hand[i]) #get card as str
```

```
            current_num=nn_hand[i]
            for x in range(0,4):
                if (x!=i): #if we're not on the card we are checking for a match for
                    check=Card.int_to_str(nn_hand[x]) #get card to check against current
                    check_num=nn_hand[x]
                    if(check[0]==current[0]): #if the cards have the same value aka are
                        cards_to_keep[0]=check
                        cards_to_keep[1]=current

                        new_hand[0]=check_num
                        new_hand[1]=current_num


                        #find third card
                        for y in range(0,5):
                            check2=Card.int_to_str(nn_hand[y])
                            if(check2[0]==current[0] and (check2[1]!=check[1] and check2
                                store=nn_hand[y]

                                new_hand[2]=new_hand[1]
                                new_hand[1]=store
                                cards_to_keep[2]=check2



        draw_2=deck.draw(2)#draw 2 new cards
        new_hand=new_hand+draw_2 #add the 2 new cards to the hand, keeping the pairs




    if(label>3):
        print("")
```

In [179]:
```
#using rankings of evaluator from Treys to determine winner

#get scores
player1_score=evaluator._five(player1_hand)
nn_score=evaluator._five(new_hand)

if (nn_score<player1_score):
    print("NN wins!")
else:
    print("YOU win!")

print ("Player1 hand: ")# prints hands
Card.print_pretty_cards(player1_hand)
```

```
        print ("NN hand: ")
        Card.print_pretty_cards(new_hand)
```

```
YOU win!
Player1 hand:
 [3],[6],[6],[A],[7]
NN hand:
 [K],[8],[3],[4],[7]
```

How do these Neural Networks perform against each other? : We ran 1000 poker games net vs net and determined which neural network performed better with our poker game, check out the results below!

```
FFNN wins: 52.1% of the time.
CNN wins: 47.6% of the time.
```

In the end, we found that both Feed-Forward and Convolutional Neural Networks performed well at the task of classifying poker hands, but had a bit more success with the Feed-Forward Neural Network.

The next step in training would be to attempt to teach a network to bet/bluff.