CNN

May 17, 2020

[1]:
```
import os
import pandas as pd
import numpy as np
```

Test on a single year to ensure the method works.

[2]:
```
df = pd.read_csv('../core/tensors/games/2015.csv', header=None)
```

[3]:
```
df
```

[3]:

	0	1	2	3	4	5	6	7	8	9	...	531	\
0	0.446721	0.40	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.136311	
1	0.467213	0.40	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.212168	
2	0.409836	0.30	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.298806	
3	0.487705	0.45	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.185214	
4	0.508197	0.55	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.112437	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2423	0.500000	0.50	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.001155	
2424	0.508197	0.50	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.041201	
2425	0.385246	0.40	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.006931	
2426	0.487705	0.30	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.033115	
2427	0.590164	0.60	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.118598	

	532	533	534	535	536	537	538	\
0	0.008721	0.140351	0.089219	0.101562	0.164557	1.0	0.164568	
1	0.007267	0.098246	0.070632	0.085938	0.065823	1.0	0.160971	
2	0.015988	0.049123	0.237918	0.414062	0.225316	1.0	0.167266	
3	0.021802	0.129825	0.052045	0.250000	0.096203	1.0	0.178058	
4	0.011628	0.056140	0.070632	0.085938	0.091139	1.0	0.163669	
...	...	...	...	...	...	...	...	
2423	0.000000	0.000000	0.007435	0.000000	0.000000	1.0	0.103417	
2424	0.000000	0.000000	0.085502	0.000000	0.020253	1.0	0.114209	
2425	0.000000	0.000000	0.022305	0.000000	0.000000	1.0	0.107014	
2426	0.000000	0.000000	0.115242	0.000000	0.002532	1.0	0.064748	
2427	0.000000	0.003509	0.271375	0.015625	0.002532	1.0	0.075540	

	539	540
0	0.116185	1.0

```
1     0.084971  1.0
2     0.124277  1.0
3     0.150867  0.0
4     0.076879  1.0
...        ...  ...
2423  0.049711  0.0
2424  0.062428  1.0
2425  0.051445  1.0
2426  0.046243  1.0
2427  0.047399  0.0

[2428 rows x 541 columns]
```

[4]:
```
game0 = list(df.iloc[0])
```

[5]:
```
len(game0)
```

[5]:
```
541
```

[6]:
```
y0 = int(game0[-1])
```

[7]:
```
y0
```

[7]:
```
1
```

[8]:
```
type(game0)
```

[8]:
```
list
```

[9]:
```
game0 = game0[:-1]
```

[10]:
```
game0 = np.reshape(game0, (18, 30, 1))
```

[11]:
```
game0.shape
```

[11]:
```
(18, 30, 1)
```

[12]:
```
game1 = list(df.iloc[1])
```

[13]:
```
y1 = int(game1[-1])
```

[14]:
```
game1 = game1[:-1]
```

[15]:
```
game1 = np.reshape(game1, (18, 30, 1))
```

[16]:
```
games = [game0, game1]
```

[17]:
```
rows = len(games)
cols = len(games[0])
fors = len(games[0][0])
last = len(games[0][0][0])
```

[18]:
```
(rows, cols, fors, last)
```

[18]: (2, 18, 30, 1)

[19]:
```
game2 = list(df.iloc[2])
y2 = int(game2[-1])
game2 = game2[:-1]
game2 = np.reshape(game2, (18, 30))
```

[20]:
```
games.append(game2)
```

[21]:
```
rows = len(games)
cols = len(games[0])
fors = len(games[0][0])
(rows, cols, fors)
```

[21]: (3, 18, 30)

We've shown that this is an effective method of getting games as 3D tables from the .csv files, as well as combining them into a list of tables. We can scale this to get a list of every game, and this will be our CNN input.

[22]:
```
df.shape
```

[22]: (2428, 541)

[23]:
```
X = []
y = []
for index in range(0, df.shape[0]):
    game = list(df.iloc[index])
    y_sample = int(game[-1])
    game = game[:-1]
    game = np.reshape(game, ([18, 30, 1]))
    X.append(game)
    y.append(y_sample)
X = np.array(X)
y = np.array(y)
```

[24]:
```
y = y[..., np.newaxis]
```

[25]:
```
X.shape
```

[25]: (2428, 18, 30, 1)

[26]:
```
y.shape
```

[26]: (2428, 1)

[27]:
```
rows = len(games)
cols = len(games[0])
fors = len(games[0][0])
last = len(games[0][0][0])
(rows, cols, fors, last)
```

[27]: (3, 18, 30, 1)

[28]:
```
len(y)
```

[28]: 2428

[29]:
```
input_shape = (18, 30, 1)
```

Now we can extrapolate this to pull all games from all years into one list. This will be the last step before running our predictive model.

[30]:
```
X = []
y = []
for year in range(1919, 2020):
    df = pd.read_csv('../core/tensors/games/{}.csv'.format(year), header=None)
    for index in range(0, df.shape[0]):
        game = list(df.iloc[index])
        y_sample = int(game[-1])
        game = game[:-1]
        game = np.reshape(game, ([18, 30, 1]))
        X.append(game)
        y.append(y_sample)
```

[31]:
```
X = np.array(X)
y = np.array(y)
```

[32]:
```
y = y[..., np.newaxis]
```

[33]:
```
rows = len(X)
cols = len(X[0])
fors = len(X[0][0])
last = len(X[0][0][0])
(rows, cols, fors, last)
```

[33]: (176687, 18, 30, 1)

[34]:
```
len(y)
```

[34]:
```
176687
```

[35]:
```
from sklearn.model_selection import train_test_split
```

[36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[37]:
```
len(X_train)
```

[37]:
```
141349
```

[38]:
```
X_train.shape
```

[38]:
```
(141349, 18, 30, 1)
```

[39]:
```
len(X_test)
```

[39]:
```
35338
```

[40]:
```
len(y_train)
```

[40]:
```
141349
```

[41]:
```
y_train.shape
```

[41]:
```
(141349, 1)
```

[42]:
```
len(y_test)
```

[42]:
```
35338
```

[43]:
```
y_train
```

[43]:
```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [1],
       [1]])
```

Building the CNN

[44]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Dropout, Flatten
```

[45]:
```
image_shape=(18, 30, 1)
```

[46]:
```
epochs = 4000
batch_size = 16
loss_param = 'binary_crossentropy'
optimizer_param = 'adam'
stop_monitor = 'val_loss'
metric = 'accuracy'
stop_patience = 20
```

[47]:
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(1, 1),
                 input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(1, 1)))

model.add(Conv2D(filters=32, kernel_size=(3, 3),
                 input_shape=image_shape, activation='tanh'))
model.add(MaxPool2D(pool_size=(3, 3)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=64, kernel_size=(3, 3),
                 activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3),
                 activation='relu', padding="same"))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss=loss_param, optimizer=optimizer_param,
              metrics=[metric])
```

[48]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 18, 30, 32)        64
```

```
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 18, 30, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 28, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 9, 32)          0
_________________________________________________________________
dropout (Dropout)            (None, 5, 9, 32)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 5, 9, 64)          18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 4, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 2, 4, 128)         73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 1, 2, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 2, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 167,713
Trainable params: 167,713
Non-trainable params: 0
_________________________________________________________________
```

[49]:
```
from tensorflow.keras.callbacks import EarlyStopping
```

[50]:
```
early_stop = EarlyStopping(monitor=stop_monitor, patience=stop_patience)
```

[51]:
```
results = model.fit(X_train, y_train, epochs=epochs,
                            validation_data=(X_test, y_test),
                            callbacks=[early_stop]
                   )
```

```
Train on 141349 samples, validate on 35338 samples
Epoch 1/4000
141349/141349 [==============================] - 189s 1ms/sample - loss: 0.6857
- accuracy: 0.5518 - val_loss: 0.6808 - val_accuracy: 0.5633
Epoch 2/4000
141349/141349 [==============================] - 200s 1ms/sample - loss: 0.6839
- accuracy: 0.5568 - val_loss: 0.6831 - val_accuracy: 0.5560
```

```
Epoch 3/4000
141349/141349 [==============================] - 195s 1ms/sample - loss: 0.6835
- accuracy: 0.5562 - val_loss: 0.6825 - val_accuracy: 0.5595
Epoch 4/4000
141349/141349 [==============================] - 179s 1ms/sample - loss: 0.6836
- accuracy: 0.5566 - val_loss: 0.6811 - val_accuracy: 0.5591
Epoch 5/4000
141349/141349 [==============================] - 177s 1ms/sample - loss: 0.6834
- accuracy: 0.5576 - val_loss: 0.6816 - val_accuracy: 0.5570
Epoch 6/4000
141349/141349 [==============================] - 183s 1ms/sample - loss: 0.6835
- accuracy: 0.5579 - val_loss: 0.6823 - val_accuracy: 0.5572
Epoch 7/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6832
- accuracy: 0.5580 - val_loss: 0.6825 - val_accuracy: 0.5627
Epoch 8/4000
141349/141349 [==============================] - 184s 1ms/sample - loss: 0.6831
- accuracy: 0.5585 - val_loss: 0.6816 - val_accuracy: 0.5615
Epoch 9/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6832
- accuracy: 0.5582 - val_loss: 0.6836 - val_accuracy: 0.5626
Epoch 10/4000
141349/141349 [==============================] - 186s 1ms/sample - loss: 0.6832
- accuracy: 0.5589 - val_loss: 0.6819 - val_accuracy: 0.5623
Epoch 11/4000
141349/141349 [==============================] - 183s 1ms/sample - loss: 0.6829
- accuracy: 0.5583 - val_loss: 0.6825 - val_accuracy: 0.5622
Epoch 12/4000
141349/141349 [==============================] - 186s 1ms/sample - loss: 0.6829
- accuracy: 0.5588 - val_loss: 0.6813 - val_accuracy: 0.5596
Epoch 13/4000
141349/141349 [==============================] - 187s 1ms/sample - loss: 0.6829
- accuracy: 0.5566 - val_loss: 0.6816 - val_accuracy: 0.5620
Epoch 14/4000
141349/141349 [==============================] - 186s 1ms/sample - loss: 0.6830
- accuracy: 0.5577 - val_loss: 0.6818 - val_accuracy: 0.5612
Epoch 15/4000
141349/141349 [==============================] - 178s 1ms/sample - loss: 0.6830
- accuracy: 0.5586 - val_loss: 0.6831 - val_accuracy: 0.5614
Epoch 16/4000
141349/141349 [==============================] - 178s 1ms/sample - loss: 0.6828
- accuracy: 0.5584 - val_loss: 0.6828 - val_accuracy: 0.5634
Epoch 17/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6829
- accuracy: 0.5576 - val_loss: 0.6812 - val_accuracy: 0.5614
Epoch 18/4000
141349/141349 [==============================] - 177s 1ms/sample - loss: 0.6828
- accuracy: 0.5585 - val_loss: 0.6817 - val_accuracy: 0.5595
```

```
Epoch 19/4000
141349/141349 [==============================] - 180s 1ms/sample - loss: 0.6829
- accuracy: 0.5584 - val_loss: 0.6834 - val_accuracy: 0.5610
Epoch 20/4000
141349/141349 [==============================] - 181s 1ms/sample - loss: 0.6829
- accuracy: 0.5584 - val_loss: 0.6830 - val_accuracy: 0.5531
Epoch 21/4000
141349/141349 [==============================] - 178s 1ms/sample - loss: 0.6829
- accuracy: 0.5593 - val_loss: 0.6813 - val_accuracy: 0.5588
```

[52]:

```
losses = model.history.history
losses['loss'] = np.asarray(losses['loss'])
losses['val_loss'] = np.asarray(losses['val_loss'])
final_number_of_epochs = len(losses['loss'])
min_loss = losses['loss'].min()
mean_loss = losses['loss'].mean()
final_loss = losses['loss'][-1]
min_val_loss = losses['val_loss'].min()
mean_val_loss = losses['val_loss'].mean()
final_val_loss = losses['val_loss'][-1]

def get_model_summary():
    output = []
    model.summary(print_fn=lambda line: output.append(line))
    return str(output).strip('[]')


summary = get_model_summary()

record = {
    'Epochs': final_number_of_epochs,
    'Batch_Size': batch_size,
    'Loss_Func': loss_param,
    'Optimizer': optimizer_param,
    'Early_Stop_Monitor': stop_monitor,
    'Early_Stop_Patience': stop_patience,
    'Min_Loss': min_loss,
    'Mean_Loss': mean_loss,
    'Final_Loss': final_loss,
    'Min_Val_Loss': min_val_loss,
    'Mean_Val_Loss': mean_val_loss,
    'Final_Val_Loss': final_val_loss,
    'Model': summary
}

new_data = pd.DataFrame(record, index=[0])
```

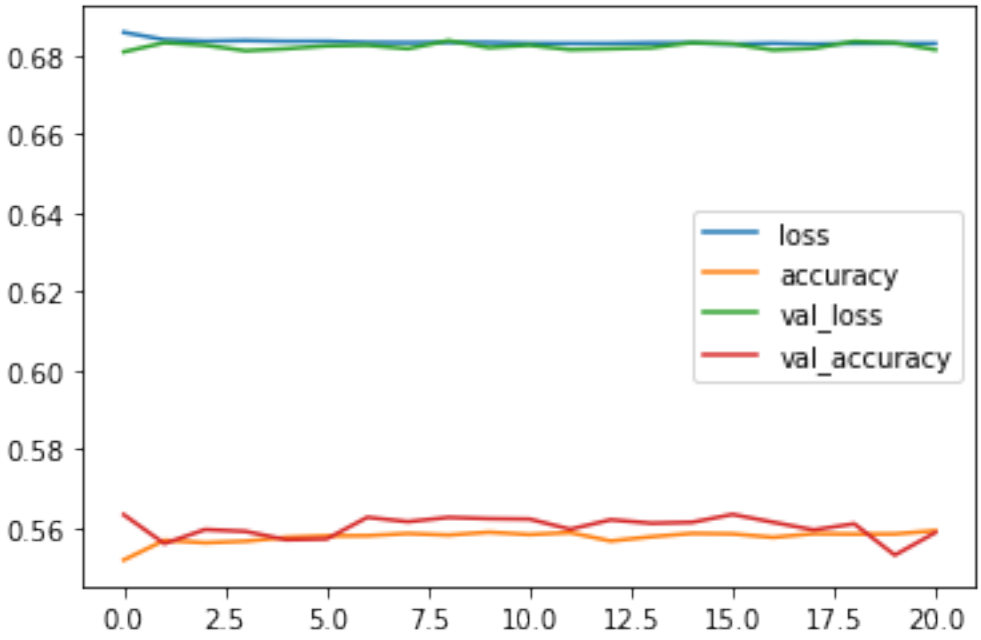
```
if os.path.exists('../core/records/game_predictions.csv'):
    df_records = pd.read_csv('../core/records/game_predictions.csv')
    df_records = df_records.append(new_data)
else:
    df_records = pd.DataFrame(new_data)

df_records.to_csv('../core/records/game_predictions.csv',
                  index=False, float_format='%g')


model.save('../core/models/model_games.h5')
```

Model Evaluation

[53]:
```
losses = pd.DataFrame(model.history.history)
```

[54]:
```
losses.plot()
```

[54]: <matplotlib.axes._subplots.AxesSubplot at 0x197e53250>



[ ]: