neural_network

May 4, 2023

[ ]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
from keras import Sequential, layers, optimizers, models
from random import randint

#* for small network, using GPU will not provide a lot of speed improvement ->␣
 ↪the following line limits the machine to use only CPU instead
tf.config.set_visible_devices([], 'GPU')
```

1 Data Processing

data from UCI Machine Learning Repositories: https://archive.ics.uci.edu/ml/datasets/SkillCraft1+Master+Table

source: Thompson JJ, Blair MR, Chen L, Henrey AJ (2013) Video Game Telemetry as a Critical Tool in the Study of Complex Skill Learning. PLoS ONE 8(9): e75129.

[ ]:

```
data = pd.read_csv('../data/SkillCraft1_Dataset.csv', na_values='?')
data = data.drop(['GameID'], axis=1)

filtered_data = data[data[data.columns].notnull().all(1)] # filter out any row␣
 ↪that contains missing value
filtered_data
```

[ ]:

```
      LeagueIndex   Age  HoursPerWeek  TotalHours       APM  SelectByHotkeys  \
0               5  27.0          10.0      3000.0  143.7180         0.003515
1               5  23.0          10.0      5000.0  129.2322         0.003304
2               4  30.0          10.0       200.0   69.9612         0.001101
3               3  19.0          20.0       400.0  107.6016         0.001034
4               3  32.0          10.0       500.0  122.8908         0.001136
…             …     …             …           …         …                …
3335            4  20.0           8.0       400.0  158.1390         0.013829
3336            5  16.0          56.0      1500.0  186.1320         0.006951
3337            4  21.0           8.0       100.0  121.6992         0.002956
3338            3  20.0          28.0       400.0  134.2848         0.005424
3339            4  22.0           6.0       400.0   88.8246         0.000844
```

```
      AssignToHotkeys  UniqueHotkeys  MinimapAttacks  MinimapRightClicks  \
0            0.000220              7        0.000110            0.000392
1            0.000259              4        0.000294            0.000432
2            0.000336              4        0.000294            0.000461
3            0.000213              1        0.000053            0.000543
4            0.000327              2        0.000000            0.001329
…                 …              …              …                 …
3335         0.000504              7        0.000217            0.000313
3336         0.000360              6        0.000083            0.000166
3337         0.000241              8        0.000055            0.000208
3338         0.000182              5        0.000000            0.000480
3339         0.000108              2        0.000000            0.000341

      NumberOfPACs  GapBetweenPACs  ActionLatency  ActionsInPAC  \
0         0.004849         32.6677        40.8673        4.7508
1         0.004307         32.9194        42.3454        4.8434
2         0.002926         44.6475        75.3548        4.0430
3         0.003783         29.2203        53.7352        4.9155
4         0.002368         22.6885        62.0813        9.3740
…             …               …              …            …
3335      0.003583         36.3990        66.2718        4.5097
3336      0.005414         22.8615        34.7417        4.9309
3337      0.003690         35.5833        57.9585        5.4154
3338      0.003205         18.2927        62.4615        6.0202
3339      0.003099         45.1512        63.4435        5.1913

      TotalMapExplored  WorkersMade  UniqueUnitsMade  ComplexUnitsMade  \
0                   28     0.001397                6               0.0
1                   22     0.001193                5               0.0
2                   22     0.000745                6               0.0
3                   19     0.000426                7               0.0
4                   15     0.001174                4               0.0
…                 …            …              …                …
3335                30     0.001035                7               0.0
3336                38     0.001343                7               0.0
3337                23     0.002014                7               0.0
3338                18     0.000934                5               0.0
3339                20     0.000476                8               0.0

      ComplexAbilitiesUsed
0                 0.000000
1                 0.000208
2                 0.000189
3                 0.000384
4                 0.000019
…                     …
3335              0.000287
```

```
3336                0.000388
3337                0.000000
3338                0.000000
3339                0.000054

[3338 rows x 19 columns]
```

2 Model Construction

[ ]:

```
predict = 'LeagueIndex'
rank: dict = {1: 'Bronze', 2: 'Silver', 3: 'Gold', 4: 'Platinum', 5: 'Diamond',␣
 ↪6: 'Master', 7: 'GrandMaster', 8: 'Professional'}

x = np.array(filtered_data.drop([predict], axis=1))
y = np.array(filtered_data[predict])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)
```

[ ]:

```
model = Sequential()
model.add(layers.Dense(16, input_shape=(18,), activation='sigmoid'))
model.add(layers.Dense(8, activation='sigmoid'))
# model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(8, activation='softmax')) #? Dense(8) works
model.summary()

#! using optimizer = 'adam' does not work for M1 architecture
model.compile(optimizer=optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

model.save('../model/8LeagueSkills_NeuralNetwork.h5')
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                304

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 8)                 72

=================================================================
Total params: 512
Trainable params: 512
Non-trainable params: 0
-----------------------------------------------------------------
Epoch 1/10

2023-05-04 01:10:20.963508: W
tensorflow/tsl/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz

94/94 [==============================] - 0s 619us/step - loss: 2.0410 -
accuracy: 0.1771
Epoch 2/10
94/94 [==============================] - 0s 509us/step - loss: 1.8722 -
accuracy: 0.1984
Epoch 3/10
94/94 [==============================] - 0s 503us/step - loss: 1.7758 -
accuracy: 0.2423
Epoch 4/10
94/94 [==============================] - 0s 503us/step - loss: 1.7183 -
accuracy: 0.2603
Epoch 5/10
94/94 [==============================] - 0s 783us/step - loss: 1.6763 -
accuracy: 0.3093
Epoch 6/10
94/94 [==============================] - 0s 494us/step - loss: 1.6412 -
accuracy: 0.3259
Epoch 7/10
94/94 [==============================] - 0s 481us/step - loss: 1.6041 -
accuracy: 0.3322
Epoch 8/10
94/94 [==============================] - 0s 484us/step - loss: 1.5644 -
accuracy: 0.3435
Epoch 9/10
94/94 [==============================] - 0s 489us/step - loss: 1.5230 -
accuracy: 0.3425
Epoch 10/10
94/94 [==============================] - 0s 490us/step - loss: 1.4944 -
accuracy: 0.3472
11/11 [==============================] - 0s 677us/step - loss: 1.5438 -
accuracy: 0.3024
Test accuracy: 0.30239519476890564
```

3 Prediction

```
[ ]: model: Sequential = models.load_model('../model/8LeagueSkills_NeuralNetwork.h5')
     predictions = model.predict(x_train)

     for _ in range(10):
         i = randint(0, len(predictions))
         print(f'Prediction: {rank[np.argmax(predictions[i])]}, Actual:␣
       ↪{rank[y_train[i]]}')
```

```
94/94 [==============================] - 0s 403us/step
Prediction: Diamond, Actual: Diamond
Prediction: Diamond, Actual: Platinum
Prediction: Diamond, Actual: Master
Prediction: Diamond, Actual: Diamond
Prediction: Diamond, Actual: Diamond
Prediction: Silver, Actual: Silver
Prediction: Gold, Actual: Gold
Prediction: Diamond, Actual: Diamond
Prediction: Diamond, Actual: Platinum
Prediction: Diamond, Actual: Diamond
```

3.1 4 Leagues Prediction

```
[ ]: fourLeague_data = filtered_data.copy()

     rank: dict = {1: 'Bronze-Silver', 2: 'Gold-Platinum', 3: 'Diamond-Master', 4:␣
       ↪'GrandMaster-Professional'}
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 2, 'LeagueIndex'] = 1
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 3, 'LeagueIndex'] = 2
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 4, 'LeagueIndex'] = 2
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 5, 'LeagueIndex'] = 3
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 6, 'LeagueIndex'] = 4
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 7, 'LeagueIndex'] = 4
     fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 8, 'LeagueIndex'] = 4

     predict = 'LeagueIndex'

     x = np.array(fourLeague_data.drop([predict], axis=1))
     y = np.array(fourLeague_data[predict])

     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

     model = Sequential()
     model.add(layers.Dense(16, input_shape=(18,), activation='sigmoid'))
     model.add(layers.Dense(8, activation='sigmoid'))
     # model.add(layers.Dense(8, activation='relu'))
     model.add(layers.Dense(5, activation='softmax')) #? Dense(4) doesn't work
```

```python
model.summary()

#! using optimizer = 'adam' does not work for M1 architecture
model.compile(optimizer=optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

model.save('../model/4LeagueSkills_NeuralNetwork.h5')
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 16)                304

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 5)                 45

=================================================================
Total params: 485
Trainable params: 485
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 16)                304

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 5)                 45

=================================================================
Total params: 485
Trainable params: 485
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
94/94 [==============================] - 0s 608us/step - loss: 1.5841 -
accuracy: 0.2400
Epoch 2/10
94/94 [==============================] - 0s 496us/step - loss: 1.4240 -
accuracy: 0.4078
```

```
Epoch 3/10
94/94 [==============================] - 0s 497us/step - loss: 1.3608 -
accuracy: 0.4078
Epoch 4/10
94/94 [==============================] - 0s 502us/step - loss: 1.3274 -
accuracy: 0.4078
Epoch 5/10
94/94 [==============================] - 0s 497us/step - loss: 1.2878 -
accuracy: 0.4078
Epoch 6/10
94/94 [==============================] - 0s 576us/step - loss: 1.2517 -
accuracy: 0.4075
Epoch 7/10
94/94 [==============================] - 0s 1ms/step - loss: 1.2158 - accuracy:
0.4081
Epoch 8/10
94/94 [==============================] - 0s 573us/step - loss: 1.1729 -
accuracy: 0.4095
Epoch 9/10
94/94 [==============================] - 0s 511us/step - loss: 1.1373 -
accuracy: 0.4291
Epoch 10/10
94/94 [==============================] - 0s 506us/step - loss: 1.1082 -
accuracy: 0.4670
11/11 [==============================] - 0s 665us/step - loss: 1.0933 -
accuracy: 0.4731
Test accuracy: 0.473053902387619
```

[ ]:

```python
predictions = model.predict(x_train)

for _ in range(10):
    i = randint(0, len(predictions))
    print(f'Prediction: {rank[np.argmax(predictions[i])]}, Actual:␣
 ↪{rank[y_train[i]]}')
```

```
94/94 [==============================] - 0s 434us/step
Prediction: Gold-Platinum, Actual: GrandMaster-Professional
Prediction: Gold-Platinum, Actual: Bronze-Silver
Prediction: Gold-Platinum, Actual: Bronze-Silver
Prediction: Gold-Platinum, Actual: Bronze-Silver
Prediction: Gold-Platinum, Actual: Diamond-Master
Prediction: Gold-Platinum, Actual: Gold-Platinum
Prediction: Diamond-Master, Actual: GrandMaster-Professional
Prediction: Gold-Platinum, Actual: GrandMaster-Professional
Prediction: Gold-Platinum, Actual: GrandMaster-Professional
Prediction: Diamond-Master, Actual: Gold-Platinum
```

3.2 3 Leagues Prediction

[ ]:

```
threeLeague_data = filtered_data.copy()

rank: dict = {1: 'Bronze-Silver-Gold', 2: 'Platinum-Diamond-Master', 3:␣
 ↪'GrandMaster-Professional'}
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 2, 'LeagueIndex'] = 1
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 3, 'LeagueIndex'] = 1
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 4, 'LeagueIndex'] = 2
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 5, 'LeagueIndex'] = 2
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 6, 'LeagueIndex'] = 2
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 7, 'LeagueIndex'] = 3
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 8, 'LeagueIndex'] = 3

predict = 'LeagueIndex'

x = np.array(threeLeague_data.drop([predict], axis=1))
y = np.array(threeLeague_data[predict])

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

model = Sequential()
model.add(layers.Dense(16, input_shape=(18,), activation='sigmoid'))
model.add(layers.Dense(8, activation='sigmoid'))
# model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(4, activation='softmax')) #? Dense(3) doesn't work
model.summary()

#! using optimizer = 'adam' does not work for M1 architecture
model.compile(optimizer=optimizers.Adam(),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

model.save('../model/3LeagueSkills_NeuralNetwork.h5')
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 16)                304

 dense_7 (Dense)             (None, 8)                 136

 dense_8 (Dense)             (None, 4)                 36
```

```
=================================================================
Total params: 476
Trainable params: 476
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 16)                304

 dense_7 (Dense)             (None, 8)                 136

 dense_8 (Dense)             (None, 4)                 36

=================================================================
Total params: 476
Trainable params: 476
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
94/94 [==============================] - 0s 702us/step - loss: 0.8374 -
accuracy: 0.6654
Epoch 2/10
94/94 [==============================] - 0s 602us/step - loss: 0.7036 -
accuracy: 0.6654
Epoch 3/10
94/94 [==============================] - 0s 983us/step - loss: 0.6362 -
accuracy: 0.7004
Epoch 4/10
94/94 [==============================] - 0s 613us/step - loss: 0.5888 -
accuracy: 0.7490
Epoch 5/10
94/94 [==============================] - 0s 622us/step - loss: 0.5570 -
accuracy: 0.7600
Epoch 6/10
94/94 [==============================] - 0s 589us/step - loss: 0.5343 -
accuracy: 0.7790
Epoch 7/10
94/94 [==============================] - 0s 513us/step - loss: 0.5162 -
accuracy: 0.7886
Epoch 8/10
94/94 [==============================] - 0s 508us/step - loss: 0.5030 -
accuracy: 0.7923
Epoch 9/10
94/94 [==============================] - 0s 521us/step - loss: 0.4982 -
accuracy: 0.7889
Epoch 10/10
```

```
94/94 [==============================] - 0s 582us/step - loss: 0.4933 -
accuracy: 0.7893
11/11 [==============================] - 0s 730us/step - loss: 0.4267 -
accuracy: 0.7904
Test accuracy: 0.7904191613197327
```

```
[ ]: predictions = model.predict(x_train)

     for _ in range(10):
         i = randint(0, len(predictions))
         print(f'Prediction: {rank[np.argmax(predictions[i])]}, Actual:␣
      ↪{rank[y_train[i]]}')
```

```
94/94 [==============================] - 0s 439us/step
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Bronze-Silver-Gold, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
```

3.3 2 Leagues Prediction

```
[ ]: twoLeague_data = filtered_data.copy()

     rank: dict = {1: 'Bronze-Silver-Gold-Platinum', 2:␣
      ↪'Diamond-Master-GrandMaster-Professional'}
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 2, 'LeagueIndex'] = 1
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 3, 'LeagueIndex'] = 1
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 4, 'LeagueIndex'] = 1
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 5, 'LeagueIndex'] = 2
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 6, 'LeagueIndex'] = 2
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 7, 'LeagueIndex'] = 2
     twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 8, 'LeagueIndex'] = 2

     predict = 'LeagueIndex'

     x = np.array(twoLeague_data.drop([predict], axis=1))
     y = np.array(twoLeague_data[predict])

     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

     model = Sequential()
     model.add(layers.Dense(16, input_shape=(18,), activation='sigmoid'))
```

```python
model.add(layers.Dense(8, activation='sigmoid'))
# model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(3, activation='softmax')) #? Dense(2) doesn't work
model.summary()

#! using optimizer = 'adam' does not work for M1 architecture
model.compile(optimizer=optimizers.Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)

test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

model.save('../model/2LeagueSkills_NeuralNetwork.h5')
```

```
Model: "sequential_3"
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 16)                304

 dense_10 (Dense)            (None, 8)                 136

 dense_11 (Dense)            (None, 3)                 27

=================================================================
Total params: 467
Trainable params: 467
Non-trainable params: 0
-----------------------------------------------------------------
-----------------------------------------------------------------
 Layer (type)                Output Shape              Param #
=================================================================
 dense_9 (Dense)             (None, 16)                304

 dense_10 (Dense)            (None, 8)                 136

 dense_11 (Dense)            (None, 3)                 27

=================================================================
Total params: 467
Trainable params: 467
Non-trainable params: 0
-----------------------------------------------------------------
Epoch 1/10
94/94 [==============================] - 0s 804us/step - loss: 0.7693 -
accuracy: 0.5320
```

```
Epoch 2/10
94/94 [==============================] - 0s 506us/step - loss: 0.6785 -
accuracy: 0.6109
Epoch 3/10
94/94 [==============================] - 0s 514us/step - loss: 0.6098 -
accuracy: 0.7617
Epoch 4/10
94/94 [==============================] - 0s 506us/step - loss: 0.5581 -
accuracy: 0.7733
Epoch 5/10
94/94 [==============================] - 0s 507us/step - loss: 0.5237 -
accuracy: 0.7776
Epoch 6/10
94/94 [==============================] - 0s 497us/step - loss: 0.5007 -
accuracy: 0.7766
Epoch 7/10
94/94 [==============================] - 0s 501us/step - loss: 0.4841 -
accuracy: 0.7853
Epoch 8/10
94/94 [==============================] - 0s 497us/step - loss: 0.4716 -
accuracy: 0.7919
Epoch 9/10
94/94 [==============================] - 0s 497us/step - loss: 0.4677 -
accuracy: 0.7933
Epoch 10/10
94/94 [==============================] - 0s 500us/step - loss: 0.4647 -
accuracy: 0.7909
11/11 [==============================] - 0s 687us/step - loss: 0.5031 -
accuracy: 0.7695
Test accuracy: 0.7694610953330994
```

[ ]:

```
predictions = model.predict(x_train)

for _ in range(10):
    i = randint(0, len(predictions))
    print(f'Prediction: {rank[np.argmax(predictions[i])]}, Actual:␣
 ↪{rank[y_train[i]]}')
```

```
94/94 [==============================] - 0s 400us/step
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
```

```
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
```