THE UNIVERSITY OF HONG KONG

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE

COMP7409 Machine Learning in Trading and Finance

Date: August 8, 2022 Time: 6:30-8:30pm

INSTRUCTIONS:

a. Candidates are permitted to bring to the examination 5 sheets of A4-sized paper with printed/written notes on both sides.
b. Internet searching, crowdsourcing from group messages, online forums or social media, etc. are strictly forbidden.
c. This paper has 4 questions. Answer ALL questions.
d. Total mark is 100.
e. All answers should be handwritten on self-prepared blank papers. Typing and writing answers on digital devices are not accepted.
f. Write your university number clearly at the beginning of your answer script. DO NOT write down your name.
g. At the end of the examination, submit to OLEX a single pdf file containing your answer script.
h. Students SHOULD NOT access any online/electronic/printed/handwritten materials, except for the electronic examination question paper and the 5 sheets of A4-sized paper with printed/written notes, during the examination.
i. Only approved calculators as announced by the Examinations Secretary can be used in the examination. It is the candidates' responsibility to ensure that their calculator operates satisfactorily, and candidates must record the name and type of the calculator used on the front page of your answer script.

1.

(a) (8%) Write a python program that inputs three integers r, c, s (one integer at a line), and then constructs two numpy 2D arrays A and B, both have r rows and c columns, and A contains the integer sequence s, s+1, … starting from left to right, and top to bottom, and B contains the square-root of the elements of A. For example, given the input:

```
3
4
64
```

A should be equal to the numpy 2D array

```
array([[64, 65, 66, 67],
       [68, 69, 70, 71],
       [72, 73, 74, 75]])
```

and B should be equal to

```
array([[8.        , 8.06225775, 8.1240384 , 8.18535277],
       [8.24621125, 8.30662386, 8.36660027, 8.42614977],
       [8.48528137, 8.54400375, 8.60232527, 8.66025404]])
```

and the program should print the sum of these two arrays, which is

```
array([[72.        , 73.06225775, 74.1240384 , 75.18535277],
       [76.24621125, 77.30662386, 78.36660027, 79.42614977],
       [80.48528137, 81.54400375, 82.60232527, 83.66025404]])
```

(b) (7%) Give a single Python statement that creates and assigns the following DataFrame to the variable `table`:

```
table
```

	year	team	wins	draws	losses
0	2018	HKU	30	6	2
1	2019	HKU	28	7	3
2	2020	HKU	32	4	2
3	2018	CU	29	5	4
4	2019	CU	32	4	2
5	2020	CU	26	7	5
6	2018	UST	21	8	9
7	2019	UST	17	10	11
8	2020	UST	19	8	11

(c) (5%) Apply the `groupby` method to `table` to get the data structure `teams_wins`, which groups the wins in terms of teams such that

- `teams_wins.groups` gives

  ```
  {'CU': [3, 4, 5], 'HKU': [0, 1, 2], 'UST': [6, 7, 8]}
  ```

  , and
- `teams_wins.sum()` gives

  ```
  team
  CU     87
  HKU    90
  UST    57
  Name: wins, dtype: int64
  ```

(d) (5%) Make use of `teams_wins` to write a single Python statement that gives the team with the largest number of wins, i.e.

```
team
HKU    90
Name: wins, dtype: int64
```

2. (25%) In the lecture, we gave the following program that trains a LSTM model to predict the Heng Seng Index for the next day. Modify the program to train a new LSTM model to predict the average Heng Seng Index of the following three days. For example, if today is 01/06/2010, then the model should predict the average of the indexes on 02/06/2010, 03/06/2010 and 04/06/2010.

```
start = datetime.strptime('2004-01-02','%Y-%m-%d')
end = datetime.strptime('2009-6-30','%Y-%m-%d')
df = data.DataReader('^HSI',start=start, end=end, data_source='yahoo')
df = df['Close']
```

```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
rdata = scaler.fit_transform(np.array(df).reshape(-1,1))
training_size=int(len(rdata)*0.65)
train_period,test_period=rdata[:training_size],rdata[training_size:]


from keras.preprocessing.sequence import TimeseriesGenerator
train = TimeseriesGenerator(train_period, train_period, length=100, batch_size=1000)
test = TimeseriesGenerator(test_period, test_period, length=100, batch_size=1000)
X_train, y_train = list(train)[0][0], list(train)[0][1]
X_test, y_test = list(test)[0][0], list(test)[0][1]


import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM

model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(50))
model.add(Dense(1))
model.compile(loss='mean_squared_error',optimizer='adam')

model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=10,batch_size=64,verbose=False)
```

```
train_pred = scaler.inverse_transform(model.predict(X_train))
test_pred = scaler.inverse_transform(model.predict(X_test))
```

Note: You don't need to import the necessary libraries in your program.

Hint: Following gives a snapshot of `X_train` and `y_train` constructed by the above program.

```
X_train
```

```
array([[[0.08871696],
        [0.09857878],
        [0.10007803],
        ...,
        [0.03506963],
        [0.0491641 ],
        [0.0555969 ]],

       [[0.09857878],
        [0.10007803],
        [0.10594915],
        ...,
        [0.0491641 ],
        [0.0555969 ],
        [0.05953342]],

       [[0.10007803],
        [0.10594915],
        [0.10817019],
        ...,
        [0.0555969 ],
        [0.05953342],
        [0.05504925]],

       ...,

       [[0.468481  ],
        [0.47611556],
        [0.47139483],
        ...,
        [0.58487261],
        [0.57445292],
        [0.5828842 ]],

       [[0.47611556],
        [0.47139483],
        [0.46154018],
        ...,
        [0.57445292],
        [0.5828842 ],
        [0.59622206]],

       [[0.47139483],
        [0.46154018],
        [0.44412024],
        ...,
        [0.5828842 ],
        [0.59622206],
        [0.59978559]]])
```

```
y_train
```

```
array([[0.05953342],
       [0.05504925],
       [0.05970322],
       [0.04655311],
       [0.05103823],
       [0.06575528],
       [0.06659273],
       [0.06638859],
       [0.07040056],
       [0.06911949],
       [0.05364728],
       [0.05239527],
       [0.05776957],
       [0.05395158],
       [0.04295476],
       [0.04247292],
       [0.04267512],
       [0.05786146],
       [0.05891802],
       [0.0593573 ],
       [0.05556931],
       [0.06376697],
       [0.0605924 ],
       [0.06213955],
       [0.06368618],
       [0.06543648],
       [0.05573623],
       [0.05972788],
       [0.05918363],
       [0.05373242],
       [0.04669342],
       [0.04701175],
       [0.05280695],
       [0.05801968],
```

3. (25%) We have studied the program below for balancing a CartPole

Cart Pole

The performance of the program is very poor (it can last only ten to twenty steps most of the times), simply because it takes random actions and does not consider the current state = (Cart Position, Cart Velocity, Pole Angle, Pole Angular Velocity) when deciding the next action. Propose some idea to improve the performance of the program, and then give a complete program (by modifying the program below) that implements this idea. You don't need to apply complicated idea like Q_learning; just simple and effective idea will be enough to get high marks.

```
import gym
env = gym.make('CartPole-v0')


env.seed(100)
env.action_space.seed(100)
random.seed(100)
np.random.seed(100)


def run_one_episode(env):
  state = env.reset()
  for step in range(200):
    a = random.randint(0,1)
    state, reward, done, info = env.step(a)
    if done:
      break
    #env.render()
    #time.sleep(0.05)
    print(f'step={step:2d} | state={state} | action={a} | reward={reward}')
  return step


nosteps = run_one_episode(env)
if nosteps >= 200:
  print(f'SUCCESS')
else:
  print(f'FAIL')
print(f'nosteps = {nosteps}')
env.close()
```

4. (a) (10%) Consider the following call option for a particular stock in which the current stock price is $400, the strike price is $360, and it will expire after 50 days. Assume that the volatility rate is 18%, risk-free rate is 4%. Determine the price of this option using the Black-Stroke equation given in the lecture:

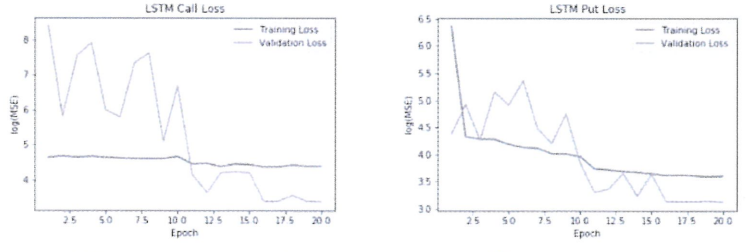
$$C = S \times N(d_1) - Ke^{-rT} \times N(d_2)$$

where $d_1 = \frac{\ln(\frac{S}{K}) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}}$, $d_2 = d_1 - \sigma\sqrt{T}$, and

You may use in your calculation the following table for the Cumulative Distribution Function of the normal distribution N(.).

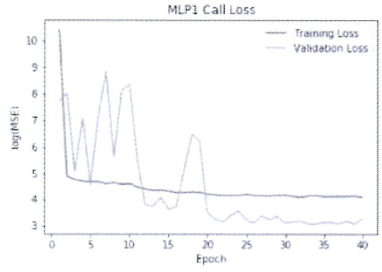
x	0	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0	0.5	0.50399	0.50798	0.51197	0.51595	0.51994	0.52392	0.5279	0.53188	0.53586
0.1	0.53983	0.5438	0.54776	0.55172	0.55567	0.55962	0.56356	0.56749	0.57142	0.57535
0.2	0.57926	0.58317	0.58706	0.59095	0.59483	0.59871	0.60257	0.60642	0.61026	0.61409
0.3	0.61791	0.62172	0.62552	0.6293	0.63307	0.63683	0.64058	0.64431	0.64803	0.65173
0.4	0.65542	0.6591	0.66276	0.6664	0.67003	0.67364	0.67724	0.68082	0.68439	0.68793
0.5	0.69146	0.69497	0.69847	0.70194	0.7054	0.70884	0.71226	0.71566	0.71904	0.7224
0.6	0.72575	0.72907	0.73237	0.73565	0.73891	0.74215	0.74537	0.74857	0.75175	0.7549
0.7	0.75804	0.76115	0.76424	0.7673	0.77035	0.77337	0.77637	0.77935	0.7823	0.78524
0.8	0.78814	0.79103	0.79389	0.79673	0.79955	0.80234	0.80511	0.80785	0.81057	0.81327
0.9	0.81594	0.81859	0.82121	0.82381	0.82639	0.82894	0.83147	0.83398	0.83646	0.83891
1	0.84134	0.84375	0.84614	0.84849	0.85083	0.85314	0.85543	0.85769	0.85993	0.86214
1.1	0.86433	0.8665	0.86864	0.87076	0.87286	0.87493	0.87698	0.879	0.881	0.88298
1.2	0.88493	0.88686	0.88877	0.89065	0.89251	0.89435	0.89617	0.89796	0.89973	0.90147
1.3	0.9032	0.9049	0.90658	0.90824	0.90988	0.91149	0.91308	0.91466	0.91621	0.91774
1.4	0.91924	0.92073	0.9222	0.92364	0.92507	0.92647	0.92785	0.92922	0.93056	0.93189
1.5	0.93319	0.93448	0.93574	0.93699	0.93822	0.93943	0.94062	0.94179	0.94295	0.94408
1.6	0.9452	0.9463	0.94738	0.94845	0.9495	0.95053	0.95154	0.95254	0.95352	0.95449
1.7	0.95543	0.95637	0.95728	0.95818	0.95907	0.95994	0.9608	0.96164	0.96246	0.96327
1.8	0.96407	0.96485	0.96562	0.96638	0.96712	0.96784	0.96856	0.96926	0.96995	0.97062
1.9	0.97128	0.97193	0.97257	0.9732	0.97381	0.97441	0.975	0.97558	0.97615	0.9767

(b) (15%) In the paper "Option Pricing with Deep Learning" by Ke and Yang, the authors showed the performance of three ML models LSTM, MLP1, MLP2 by giving the following graphs on their Mean Squared Error (MSE):

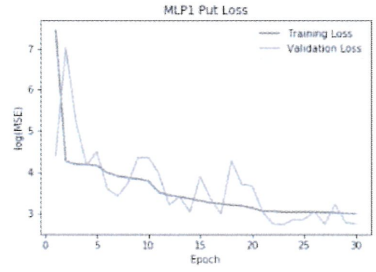


(a) LSTM trained on call option data. (b) LSTM trained on put option data.

Figure 6: MSE over training of LSTM models. Note the logarithmic y-axis.

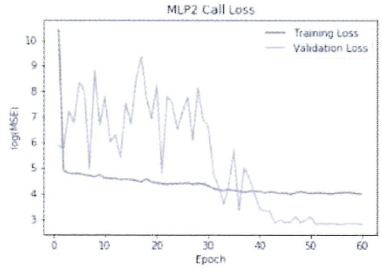


(a) MLP1 trained on call option data.

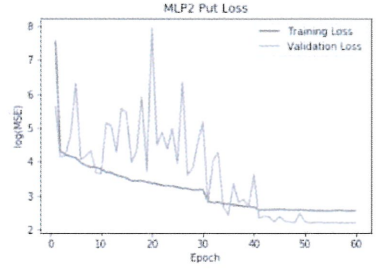


(b) MLP1 trained on put option data.

Figure 4: MSE over training of MLP1 models. Note the logarithmic y-axis.



(a) MLP2 trained on call option data.



(b) MLP2 trained on put option data.

Figure 5: MSE over training of MLP2 models. Note the logarithmic y-axis.

From these graphs draw as many conclusions as you can on the performance of these three models. Note that this is an open-ended question, and there is not definitely right or wrong answers. Your marks will be depending on the number of conclusions you draw, and the supporting reasons you give for each of your conclusions.

*** END OF PAPER ***