rl-project-code-nio

May 4, 2023

[1]:

```
#Importing the libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

[2]:

```
import time
import copy
import numpy as np
import pandas as pd
import chainer
import chainer.functions as F
import chainer.links as L
from plotly import tools
from plotly.graph_objs import *
from plotly.offline import init_notebook_mode, iplot, iplot_mpl
init_notebook_mode()
```

[3]:

```
#reading the data
data = pd.read_csv("C:\\Users\\madha\\OneDrive\\Desktop\\RL project\\NIO.csv")
data['Date'] = pd.to_datetime(data['Date'])
data = data.set_index('Date')
print(data.index.min(), data.index.max())
data.head()
```

```
2020-01-02 00:00:00 2022-12-29 00:00:00
```

[3]:

```
            Open  High   Low  Close  Adj Close     Volume
Date
2020-01-02  4.10  4.10  3.61   3.72       3.72  103740100
2020-01-03  3.50  3.90  3.48   3.83       3.83   82892400
2020-01-06  4.19  4.24  3.66   3.68       3.68  106619700
2020-01-07  3.70  3.73  3.21   3.24       3.24  106336400
2020-01-08  3.14  3.49  3.13   3.39       3.39   65118100
```

Splitting the data in train and test model

[4]:

```
date_split = '2022-04-01'
train = data[:date_split]
test = data[date_split:]
```

```
len(train), len(test)
```

[4]: (568, 188)

[5]:
```
def plot_train_test(train, test, date_split):

    data = [
        Candlestick(x=train.index, open=train['Open'], high=train['High'],␣
↪low=train['Low'], close=train['Close'], name='train'),
        Candlestick(x=test.index, open=test['Open'], high=test['High'],␣
↪low=test['Low'], close=test['Close'], name='test')
    ]
    layout = {
         'shapes': [
             {'x0': date_split, 'x1': date_split, 'y0': 0, 'y1': 1, 'xref':␣
↪'x', 'yref': 'paper', 'line': {'color': 'rgb(1,1,0)', 'width': 1}}
         ],
        'annotations': [
            {'x': date_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper',␣
↪'showarrow': False, 'xanchor': 'left', 'text': ' test data'},
            {'x': date_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper',␣
↪'showarrow': False, 'xanchor': 'right', 'text': 'train data '}
        ]
    }
    figure = Figure(data=data, layout=layout)
    iplot(figure)
    #display(HTML(figure.to_html()))
```

[6]:
```
plot_train_test(train, test, date_split)
```

[7]:
```
class Environment:

    def __init__(self, data, history_t=90):
        self.data = data
        self.history_t = history_t
        self.reset()

    def reset(self):
        self.t = 0
        self.done = False
        self.profits = 0
        self.positions = []
        self.position_value = 0
        self.history = [0 for _ in range(self.history_t)]
        return [self.position_value] + self.history # obs

    def step(self, act):
        reward = 0

        # act = 0: stay, 1: buy, 2: sell
        if act == 1:
            self.positions.append(self.data.iloc[self.t, :]['Close'])
        elif act == 2: # sell
            if len(self.positions) == 0:
                reward = -1
            else:
                profits = 0
                for p in self.positions:
                    profits += (self.data.iloc[self.t, :]['Close'] - p)
                reward += profits
                self.profits += profits
                self.positions = []

        # set next time
        self.t += 1
        self.position_value = 0
        for p in self.positions:
            self.position_value += (self.data.iloc[self.t, :]['Close'] - p)
        self.history.pop(0)
        self.history.append(self.data.iloc[self.t, :]['Close'] - self.data.
 ↪iloc[(self.t-1), :]['Close'])

        # clipping reward
        if reward > 0:
            reward = 1
        elif reward < 0:
            reward = -1

        return [self.position_value] + self.history, reward, self.done # obs,␣
 ↪reward, done
```

[8]:

```
env = Environment(train)
print(env.reset())
for _ in range(3):
    pact = np.random.randint(3)
    print(env.step(pact))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.10999999999999988], -1, False)
([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.10999999999999988, -0.14999999999999999], -1, False)
([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.10999999999999988, -0.14999999999999999,
-0.43999999999999995], 0, False)
```

Applying the DQN model

[9]:

```python
# DQN

def train_dqn(env):

    class Q_Network(chainer.Chain):

        def __init__(self, input_size, hidden_size, output_size):
            super(Q_Network, self).__init__(
                fc1 = L.Linear(input_size, hidden_size),
                fc2 = L.Linear(hidden_size, hidden_size),
                fc3 = L.Linear(hidden_size, output_size)
            )

        def __call__(self, x):
            h = F.relu(self.fc1(x))
            h = F.relu(self.fc2(h))
            y = self.fc3(h)
            return y

        def reset(self):
            self.zerograds()

    Q = Q_Network(input_size=env.history_t+1, hidden_size=100, output_size=3)
    Q_ast = copy.deepcopy(Q)
    optimizer = chainer.optimizers.Adam()
    optimizer.setup(Q)

    epoch_num = 50
    step_max = len(env.data)-1
    memory_size = 200
    batch_size = 20
    epsilon = 1.0
    epsilon_decrease = 1e-3
    epsilon_min = 0.1
    start_reduce_epsilon = 200
    train_freq = 10
    update_q_freq = 20
    gamma = 0.99
    show_log_freq = 5

    memory = []
    total_step = 0
    total_rewards = []
    total_losses = []

    start = time.time()
    for epoch in range(epoch_num):

        pobs = env.reset()
        step = 0
        done = False
        total_reward = 0
        total_loss = 0

        while not done and step < step_max:

            # select act
            pact = np.random.randint(3)
            if np.random.rand() > epsilon:
                pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))
                pact = np.argmax(pact.data)

            # act
            obs, reward, done = env.step(pact)

            # add memory
            memory.append((pobs, pact, reward, obs, done))
            if len(memory) > memory_size:
                memory.pop(0)

            # train or update q
            if len(memory) == memory_size:
                if total_step % train_freq == 0:
                    shuffled_memory = np.random.permutation(memory)
                    memory_idx = range(len(shuffled_memory))
                    for i in memory_idx[::batch_size]:
                        batch = np.array(shuffled_memory[i:i+batch_size])
                        b_pobs = np.array(batch[:, 0].tolist(), dtype=np.
↪float32).reshape(batch_size, -1)
                        b_pact = np.array(batch[:, 1].tolist(), dtype=np.int32)
                        b_reward = np.array(batch[:, 2].tolist(), dtype=np.
↪int32)
```

```
                        b_obs = np.array(batch[:, 3].tolist(), dtype=np.
↪float32).reshape(batch_size, -1)
                        b_done = np.array(batch[:, 4].tolist(), dtype=np.bool)

                        q = Q(b_pobs)
                        maxq = np.max(Q_ast(b_obs).data, axis=1)
                        target = copy.deepcopy(q.data)
                        for j in range(batch_size):
                            target[j, b_pact[j]] =␣
↪b_reward[j]+gamma*maxq[j]*(not b_done[j])
                        Q.reset()
                        loss = F.mean_squared_error(q, target)
                        total_loss += loss.data
                        loss.backward()
                        optimizer.update()

                if total_step % update_q_freq == 0:
                    Q_ast = copy.deepcopy(Q)

            # epsilon
            if epsilon > epsilon_min and total_step > start_reduce_epsilon:
                epsilon -= epsilon_decrease

            # next step
            total_reward += reward
            pobs = obs
            step += 1
            total_step += 1

        total_rewards.append(total_reward)
        total_losses.append(total_loss)

        if (epoch+1) % show_log_freq == 0:
            log_reward = sum(total_rewards[((epoch+1)-show_log_freq):])/
↪show_log_freq
            log_loss = sum(total_losses[((epoch+1)-show_log_freq):])/
↪show_log_freq
            elapsed_time = time.time()-start
            print('\t'.join(map(str, [epoch+1, epsilon, total_step, log_reward,␣
↪log_loss, elapsed_time])))
            start = time.time()

    return Q, total_losses, total_rewards
```

[10]:
```
Q, total_losses, total_rewards = train_dqn(Environment(train))
```

```
C:\Users\madha\AppData\Local\Temp\ipykernel_3192\515023404.py:74:
```

```
VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.

C:\Users\madha\AppData\Local\Temp\ipykernel_3192\515023404.py:82:
DeprecationWarning:

`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning,
use `bool` by itself. Doing this will not modify any behavior and is safe. If
you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations


5       0.0999999999999992      2835    -31.0   13367.827018247172
18.211485385894775
10      0.0999999999999992      5670    -14.6   394245.34012479783
27.150068759918213
15      0.0999999999999992      8505    -150.4  494851.8175792694
24.27080273628235
20      0.0999999999999992      11340   -138.6  53307.133459949495
22.52271318435669
25      0.0999999999999992      14175   -124.4  18333.73999013901
22.55620002746582
30      0.0999999999999992      17010   -87.4   13681.400502169132
22.050071477890015
35      0.0999999999999992      19845   -80.6   6073.248950099945
22.67339825630188
40      0.0999999999999992      22680   -63.4   3513.8264394462108
23.218868255615234
45      0.0999999999999992      25515   -49.0   2317.257956920564
23.792641401290894
50      0.0999999999999992      28350   -43.0   1462.8681479632855
23.048948049545288
```

[11]:

```python
def plot_loss_reward(total_losses, total_rewards):

    figure = tools.make_subplots(rows=1, cols=2, subplot_titles=('DQN Loss', '␣
  ↪DQN Reward'), print_grid=False)
    figure.append_trace(Scatter(y=total_losses, mode='lines',␣
  ↪line=dict(color='skyblue')), 1, 1)
    figure.append_trace(Scatter(y=total_rewards, mode='lines',␣
  ↪line=dict(color='orange')), 1, 2)
    figure['layout']['xaxis1'].update(title='epoch')
    figure['layout']['xaxis2'].update(title='epoch')
    figure['layout'].update(height=400, width=900, showlegend=False)
    iplot(figure)
```

[12]:

```
plot_loss_reward(total_losses, total_rewards)
```

```
C:\Users\madha\anaconda3\lib\site-packages\plotly\tools.py:461:
DeprecationWarning:

plotly.tools.make_subplots is deprecated, please use
plotly.subplots.make_subplots instead
```

[13]:

```
def plot_train_test_by_q(train_env, test_env, Q, algorithm_name):

    # train
    pobs = train_env.reset()
    train_acts = []
    train_rewards = []

    for _ in range(len(train_env.data)-1):

        pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))
        pact = np.argmax(pact.data)
        train_acts.append(pact)

        obs, reward, done = train_env.step(pact)
        train_rewards.append(reward)

        pobs = obs

    train_profits = train_env.profits

    # test
    pobs = test_env.reset()
    test_acts = []
    test_rewards = []

    for _ in range(len(test_env.data)-1):

        pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))
        pact = np.argmax(pact.data)
        test_acts.append(pact)

        obs, reward, done = test_env.step(pact)
        test_rewards.append(reward)

        pobs = obs
    test_profits = test_env.profits

    # plot
    train_copy = train_env.data.copy()
    test_copy = test_env.data.copy()
    train_copy['act'] = train_acts + [np.nan]
    train_copy['reward'] = train_rewards + [np.nan]
    test_copy['act'] = test_acts + [np.nan]
    test_copy['reward'] = test_rewards + [np.nan]
    train0 = train_copy[train_copy['act'] == 0]
    train1 = train_copy[train_copy['act'] == 1]
    train2 = train_copy[train_copy['act'] == 2]
    test0 = test_copy[test_copy['act'] == 0]
    test1 = test_copy[test_copy['act'] == 1]
    test2 = test_copy[test_copy['act'] == 2]
    act_color0, act_color1, act_color2 = 'black', 'green', 'red'

    data = [
        Candlestick(x=train0.index, open=train0['Open'], high=train0['High'], low=train0['Low'], close=train0['Close'], increasing=dict(line=dict(color=act_color0)), decreasing=dict(line=dict(color=act_color0))),
        Candlestick(x=train1.index, open=train1['Open'], high=train1['High'], low=train1['Low'], close=train1['Close'], increasing=dict(line=dict(color=act_color1)), decreasing=dict(line=dict(color=act_color1))),
        Candlestick(x=train2.index, open=train2['Open'], high=train2['High'], low=train2['Low'], close=train2['Close'], increasing=dict(line=dict(color=act_color2)), decreasing=dict(line=dict(color=act_color2))),
        Candlestick(x=test0.index, open=test0['Open'], high=test0['High'], low=test0['Low'], close=test0['Close'], increasing=dict(line=dict(color=act_color0)), decreasing=dict(line=dict(color=act_color0))),
        Candlestick(x=test1.index, open=test1['Open'], high=test1['High'], low=test1['Low'], close=test1['Close'], increasing=dict(line=dict(color=act_color1)), decreasing=dict(line=dict(color=act_color1))),
        Candlestick(x=test2.index, open=test2['Open'], high=test2['High'], low=test2['Low'], close=test2['Close'], increasing=dict(line=dict(color=act_color2)), decreasing=dict(line=dict(color=act_color2)))
    ]
    title = '{}: train s-reward {}, profits {}, test s-reward {}, profits {}'.format(
        algorithm_name,
        int(sum(train_rewards)),
        int(train_profits),
        int(sum(test_rewards)),
        int(test_profits)
    )
    layout = {
        'title': title,
        'showlegend': False,
         'shapes': [
             {'x0': date_split, 'x1': date_split, 'y0': 0, 'y1': 1, 'xref':␣
 ↪'x', 'yref': 'paper', 'line': {'color': 'rgb(0,0,0)', 'width': 1}}
         ],
        'annotations': [
            {'x': date_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'left', 'text': ' test data'},
            {'x': date_split, 'y': 1.0, 'xref': 'x', 'yref': 'paper',␣
 ↪'showarrow': False, 'xanchor': 'right', 'text': 'train data '}
        ]
    }
    figure = Figure(data=data, layout=layout)
    iplot(figure)
```

[14]:

```
plot_train_test_by_q(Environment(train), Environment(test), Q, 'DQN')
```

Applying the Double DQN model

[15]:

```
# Double DQN

def train_ddqn(env):

    class Q_Network(chainer.Chain):

        def __init__(self, input_size, hidden_size, output_size):
            super(Q_Network, self).__init__(
                fc1 = L.Linear(input_size, hidden_size),
                fc2 = L.Linear(hidden_size, hidden_size),
                fc3 = L.Linear(hidden_size, output_size)
            )

        def __call__(self, x):
            h = F.relu(self.fc1(x))
            h = F.relu(self.fc2(h))
            y = self.fc3(h)
            return y

        def reset(self):
            self.zerograds()
```

```
Q = Q_Network(input_size=env.history_t+1, hidden_size=100, output_size=3)
Q_ast = copy.deepcopy(Q)
optimizer = chainer.optimizers.Adam()
optimizer.setup(Q)

epoch_num = 50
step_max = len(env.data)-1
memory_size = 200
batch_size = 50
epsilon = 1.0
epsilon_decrease = 1e-3
epsilon_min = 0.1
start_reduce_epsilon = 200
train_freq = 10
update_q_freq = 20
gamma = 0.99
show_log_freq = 5

memory = []
total_step = 0
total_rewards = []
total_losses = []

start = time.time()
for epoch in range(epoch_num):

    pobs = env.reset()
    step = 0
    done = False
    total_reward = 0
    total_loss = 0

    while not done and step < step_max:

        # select act
        pact = np.random.randint(3)
        if np.random.rand() > epsilon:
            pact = Q(np.array(pobs, dtype=np.float32).reshape(1, -1))
            pact = np.argmax(pact.data)

        # act
        obs, reward, done = env.step(pact)

        # add memory
        memory.append((pobs, pact, reward, obs, done))
        if len(memory) > memory_size:
                memory.pop(0)

            # train or update q
            if len(memory) == memory_size:
                if total_step % train_freq == 0:
                    shuffled_memory = np.random.permutation(memory)
                    memory_idx = range(len(shuffled_memory))
                    for i in memory_idx[::batch_size]:
                        batch = np.array(shuffled_memory[i:i+batch_size])
                        b_pobs = np.array(batch[:, 0].tolist(), dtype=np.
↪float32).reshape(batch_size, -1)
                        b_pact = np.array(batch[:, 1].tolist(), dtype=np.int32)
                        b_reward = np.array(batch[:, 2].tolist(), dtype=np.
↪int32)
                        b_obs = np.array(batch[:, 3].tolist(), dtype=np.
↪float32).reshape(batch_size, -1)
                        b_done = np.array(batch[:, 4].tolist(), dtype=np.bool)

                        q = Q(b_pobs)
                        """ <<< DQN -> Double DQN
                        maxq = np.max(Q_ast(b_obs).data, axis=1)
                        === """
                        indices = np.argmax(q.data, axis=1)
                        maxqs = Q_ast(b_obs).data
                        """ >>> """
                        target = copy.deepcopy(q.data)
                        for j in range(batch_size):
                            """ <<< DQN -> Double DQN
                            target[j, b_pact[j]] =␣
↪b_reward[j]+gamma*maxq[j]*(not b_done[j])
                            === """
                            target[j, b_pact[j]] = b_reward[j]+gamma*maxqs[j,␣
↪indices[j]]*(not b_done[j])
                            """ >>> """
                        Q.reset()
                        loss = F.mean_squared_error(q, target)
                        total_loss += loss.data
                        loss.backward()
                        optimizer.update()

                if total_step % update_q_freq == 0:
                    Q_ast = copy.deepcopy(Q)

            # epsilon
            if epsilon > epsilon_min and total_step > start_reduce_epsilon:
                epsilon -= epsilon_decrease
            # next step
            total_reward += reward
            pobs = obs
            step += 1
            total_step += 1

        total_rewards.append(total_reward)
        total_losses.append(total_loss)

        if (epoch+1) % show_log_freq == 0:
            log_reward = sum(total_rewards[((epoch+1)-show_log_freq):])/
 ↪show_log_freq
            log_loss = sum(total_losses[((epoch+1)-show_log_freq):])/
 ↪show_log_freq
            elapsed_time = time.time()-start
            print('\t'.join(map(str, [epoch+1, epsilon, total_step, log_reward,␣
 ↪log_loss, elapsed_time])))
            start = time.time()

    return Q, total_losses, total_rewards
```

[16]:

```
Q, total_losses, total_rewards = train_ddqn(Environment(train))
```

```
C:\Users\madha\AppData\Local\Temp\ipykernel_3192\1579113142.py:74:
VisibleDeprecationWarning:

Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If
you meant to do this, you must specify 'dtype=object' when creating the ndarray.

C:\Users\madha\AppData\Local\Temp\ipykernel_3192\1579113142.py:82:
DeprecationWarning:

`np.bool` is a deprecated alias for the builtin `bool`. To silence this warning,
use `bool` by itself. Doing this will not modify any behavior and is safe. If
you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations


5       0.0999999999999992      2835    -33.4   41.554668059386316
14.612796068191528
10      0.0999999999999992      5670    3.6     79.02428655717522
14.437743902206421
15      0.0999999999999992      8505    4.2     93.92526926249266
15.615903377532959
20      0.0999999999999992      11340   4.6     14.700800946727394
13.950337886810303
```

```
25      0.0999999999999992      14175   15.4    7.628558929357678
15.192245721817017
30      0.0999999999999992      17010   13.8    6.4270844518207015
15.193782091140747
35      0.0999999999999992      19845   10.8    8.912814674107358
14.709294557571411
40      0.0999999999999992      22680   15.8    7.5310753562022
15.371719360351562
45      0.0999999999999992      25515   16.4    6.759689093055203
16.17582130432129
50      0.0999999999999992      28350   11.4    5.152364396560006
14.769146203994751
```

[17]:

```python
def plot_loss_reward(total_losses, total_rewards):

    figure = tools.make_subplots(rows=1, cols=2, subplot_titles=(' Double DQN␣
 ↪Loss', ' Double DQN Reward'), print_grid=False)
    figure.append_trace(Scatter(y=total_losses, mode='lines',␣
 ↪line=dict(color='skyblue')), 1, 1)
    figure.append_trace(Scatter(y=total_rewards, mode='lines',␣
 ↪line=dict(color='orange')), 1, 2)
    figure['layout']['xaxis1'].update(title='epoch')
    figure['layout']['xaxis2'].update(title='epoch')
    figure['layout'].update(height=400, width=900, showlegend=False)
    iplot(figure)
```

[18]:

```python
plot_loss_reward(total_losses, total_rewards)
```

```
C:\Users\madha\anaconda3\lib\site-packages\plotly\tools.py:461:
DeprecationWarning:

plotly.tools.make_subplots is deprecated, please use
plotly.subplots.make_subplots instead
```

[19]:

```python
plot_train_test_by_q(Environment(train), Environment(test), Q, 'Double DQN')
```

[ ]:

```python

```