12.duel-recurrent-q-learning-agent

September 29, 2021

[1]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

[2]:
```
df = pd.read_csv('../dataset/GOOG-year.csv')
df.head()
```

[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2016-11-02	778.200012	781.650024	763.450012	768.700012	768.700012	1872400
1	2016-11-03	767.250000	769.950012	759.030029	762.130005	762.130005	1943200
2	2016-11-04	750.659973	770.359985	750.560974	762.020020	762.020020	2134800
3	2016-11-07	774.500000	785.190002	772.549988	782.520020	782.520020	1585100
4	2016-11-08	783.400024	795.632996	780.190002	790.510010	790.510010	1350800

[3]:
```
from collections import deque
import random

class Agent:

    LEARNING_RATE = 0.003
    BATCH_SIZE = 32
    LAYER_SIZE = 256
    OUTPUT_SIZE = 3
    EPSILON = 0.5
    DECAY_RATE = 0.005
    MIN_EPSILON = 0.1
    GAMMA = 0.99
```

```
    MEMORIES = deque()
    MEMORY_SIZE = 300

    def __init__(self, state_size, window_size, trend, skip):
        self.state_size = state_size
        self.window_size = window_size
        self.half_window = window_size // 2
        self.trend = trend
        self.skip = skip
        tf.reset_default_graph()
        self.INITIAL_FEATURES = np.zeros((4, self.state_size))
        self.X = tf.placeholder(tf.float32, (None, None, self.state_size))
        self.Y = tf.placeholder(tf.float32, (None, self.OUTPUT_SIZE))
        cell = tf.nn.rnn_cell.LSTMCell(self.LAYER_SIZE, state_is_tuple = False)
        self.hidden_layer = tf.placeholder(tf.float32, (None, 2 * self.
↪LAYER_SIZE))
        self.rnn,self.last_state = tf.nn.dynamic_rnn(inputs=self.X,cell=cell,
                                                     dtype=tf.float32,
                                                     initial_state=self.
↪hidden_layer)
        tensor_action, tensor_validation = tf.split(self.rnn[:,-1],2,1)
        feed_action = tf.layers.dense(tensor_action, self.OUTPUT_SIZE)
        feed_validation = tf.layers.dense(tensor_validation, 1)
        self.logits = feed_validation + tf.subtract(feed_action,tf.
↪reduce_mean(feed_action,axis=1,keep_dims=True))
        self.cost = tf.reduce_sum(tf.square(self.Y - self.logits))
        self.optimizer = tf.train.AdamOptimizer(learning_rate = self.
↪LEARNING_RATE).minimize(self.cost)
        self.sess = tf.InteractiveSession()
        self.sess.run(tf.global_variables_initializer())

    def _memorize(self, state, action, reward, new_state, dead, rnn_state):
        self.MEMORIES.append((state, action, reward, new_state, dead,␣
↪rnn_state))
        if len(self.MEMORIES) > self.MEMORY_SIZE:
            self.MEMORIES.popleft()

    def _construct_memories(self, replay):
        states = np.array([a[0] for a in replay])
        new_states = np.array([a[3] for a in replay])
        init_values = np.array([a[-1] for a in replay])
        Q = self.sess.run(self.logits, feed_dict={self.X:states, self.
↪hidden_layer:init_values})
        Q_new = self.sess.run(self.logits, feed_dict={self.X:new_states, self.
↪hidden_layer:init_values})
        replay_size = len(replay)
```

```
        X = np.empty((replay_size, 4, self.state_size))
        Y = np.empty((replay_size, self.OUTPUT_SIZE))
        INIT_VAL = np.empty((replay_size, 2 * self.LAYER_SIZE))
        for i in range(replay_size):
            state_r, action_r, reward_r, new_state_r, dead_r, rnn_memory =␣
↪replay[i]
            target = Q[i]
            target[action_r] = reward_r
            if not dead_r:
                target[action_r] += self.GAMMA * np.amax(Q_new[i])
            X[i] = state_r
            Y[i] = target
            INIT_VAL[i] = rnn_memory
        return X, Y, INIT_VAL

    def get_state(self, t):
        window_size = self.window_size + 1
        d = t - window_size + 1
        block = self.trend[d : t + 1] if d >= 0 else -d * [self.trend[0]] +␣
↪self.trend[0 : t + 1]
        res = []
        for i in range(window_size - 1):
            res.append(block[i + 1] - block[i])
        return np.array(res)

    def buy(self, initial_money):
        starting_money = initial_money
        states_sell = []
        states_buy = []
        inventory = []
        state = self.get_state(0)
        init_value = np.zeros((1, 2 * self.LAYER_SIZE))
        for k in range(self.INITIAL_FEATURES.shape[0]):
            self.INITIAL_FEATURES[k,:] = state
        for t in range(0, len(self.trend) - 1, self.skip):
            action, last_state = self.sess.run([self.logits,self.last_state],
                                               feed_dict={self.X:[self.
↪INITIAL_FEATURES],
                                                                  self.hidden_layer:
↪init_value})
            action, init_value = np.argmax(action[0]), last_state
            next_state = self.get_state(t + 1)

            if action == 1 and initial_money >= self.trend[t]:
                inventory.append(self.trend[t])
                initial_money -= self.trend[t]
                states_buy.append(t)
```

```
                print('day %d: buy 1 unit at price %f, total balance %f'% (t,␣
↪self.trend[t], initial_money))

            elif action == 2 and len(inventory):
                bought_price = inventory.pop(0)
                initial_money += self.trend[t]
                states_sell.append(t)
                try:
                    invest = ((close[t] - bought_price) / bought_price) * 100
                except:
                    invest = 0
                print(
                    'day %d, sell 1 unit at price %f, investment %f %%, total␣
↪balance %f,'
                    % (t, close[t], invest, initial_money)
                )

            new_state = np.append([self.get_state(t + 1)], self.
↪INITIAL_FEATURES[:3, :], axis = 0)
            self.INITIAL_FEATURES = new_state
        invest = ((initial_money - starting_money) / starting_money) * 100
        total_gains = initial_money - starting_money
        return states_buy, states_sell, total_gains, invest


    def train(self, iterations, checkpoint, initial_money):
        for i in range(iterations):
            total_profit = 0
            inventory = []
            state = self.get_state(0)
            starting_money = initial_money
            init_value = np.zeros((1, 2 * self.LAYER_SIZE))
            for k in range(self.INITIAL_FEATURES.shape[0]):
                self.INITIAL_FEATURES[k,:] = state
            for t in range(0, len(self.trend) - 1, self.skip):

                if np.random.rand() < self.EPSILON:
                    action = np.random.randint(self.OUTPUT_SIZE)
                else:
                    action, last_state = self.sess.run([self.logits,
                                                        self.last_state],
                                                        feed_dict={self.X:[self.
↪INITIAL_FEATURES],
                                                                           self.hidden_layer:
↪init_value})
                    action, init_value = np.argmax(action[0]), last_state
```

```
                next_state = self.get_state(t + 1)

                if action == 1 and starting_money >= self.trend[t]:
                    inventory.append(self.trend[t])
                    starting_money -= self.trend[t]

                elif action == 2 and len(inventory) > 0:
                    bought_price = inventory.pop(0)
                    total_profit += self.trend[t] - bought_price
                    starting_money += self.trend[t]

                invest = ((starting_money - initial_money) / initial_money)
                new_state = np.append([self.get_state(t + 1)], self.
↪INITIAL_FEATURES[:3, :], axis = 0)
                self._memorize(self.INITIAL_FEATURES, action, invest, new_state,
                               starting_money < initial_money, init_value[0])
                self.INITIAL_FEATURES = new_state
                batch_size = min(len(self.MEMORIES), self.BATCH_SIZE)
                replay = random.sample(self.MEMORIES, batch_size)
                X, Y, INIT_VAL = self._construct_memories(replay)

                cost, _ = self.sess.run([self.cost, self.optimizer],
                                        feed_dict={self.X: X, self.Y:Y,
                                                   self.hidden_layer: INIT_VAL})
                self.EPSILON = self.MIN_EPSILON + (1.0 - self.MIN_EPSILON) * np.
↪exp(-self.DECAY_RATE * i)

            if (i+1) % checkpoint == 0:
                print('epoch: %d, total rewards: %f.3, cost: %f, total money:␣
↪%f'%(i + 1, total_profit, cost,
                                                                              ␣
↪  starting_money))
```

[4]:
```
close = df.Close.values.tolist()
initial_money = 10000
window_size = 30
skip = 1
batch_size = 32
agent = Agent(state_size = window_size,
              window_size = window_size,
              trend = close,
              skip = skip)
agent.train(iterations = 200, checkpoint = 10, initial_money = initial_money)
```

```
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7f2873435940>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
```

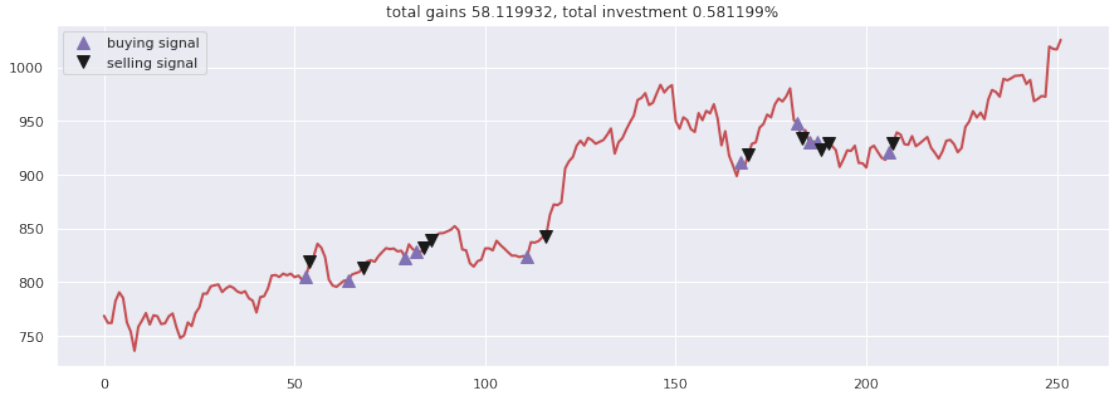
```
WARNING:tensorflow:From <ipython-input-3-976c717fc00c>:35: calling reduce_mean
(from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be
removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
epoch: 10, total rewards: 1303.755127.3, cost: 0.204159, total money:
2622.175109
epoch: 20, total rewards: 1332.510133.3, cost: 2.512769, total money:
11332.510133
epoch: 30, total rewards: 167.034789.3, cost: 0.204751, total money:
10167.034789
epoch: 40, total rewards: 885.269897.3, cost: 0.095390, total money: 8848.889892
epoch: 50, total rewards: 312.624996.3, cost: 0.415782, total money:
10312.624996
epoch: 60, total rewards: 220.209960.3, cost: 0.119438, total money:
10220.209960
epoch: 70, total rewards: 407.794859.3, cost: 0.983801, total money: 8417.984861
epoch: 80, total rewards: 200.149718.3, cost: 0.235913, total money: 9226.819701
epoch: 90, total rewards: 87.564821.3, cost: 0.034903, total money: 8097.894838
epoch: 100, total rewards: 1056.600041.3, cost: 0.286240, total money:
11056.600041
epoch: 110, total rewards: 537.204957.3, cost: 0.140037, total money:
7610.014955
epoch: 120, total rewards: 263.944828.3, cost: 0.535866, total money:
9247.304813
epoch: 130, total rewards: 387.030092.3, cost: 0.352989, total money:
8396.590090
epoch: 140, total rewards: 207.069887.3, cost: 0.474047, total money:
10207.069887
epoch: 150, total rewards: -119.230104.3, cost: 0.301262, total money:
9880.769896
epoch: 160, total rewards: 21.299804.3, cost: 0.709494, total money:
10021.299804
epoch: 170, total rewards: 241.145077.3, cost: 0.486697, total money:
10241.145077
epoch: 180, total rewards: 5.329770.3, cost: 0.447255, total money: 7042.329770
epoch: 190, total rewards: 126.395198.3, cost: 0.240739, total money:
9107.125178
epoch: 200, total rewards: 91.499876.3, cost: 0.259028, total money: 8055.119871
```

[5]:
```
states_buy, states_sell, total_gains, invest = agent.buy(initial_money =␣
↪initial_money)
```

```
day 53: buy 1 unit at price 805.020020, total balance 9194.979980
day 54, sell 1 unit at price 819.309998, investment 1.775108 %, total balance
10014.289978,
day 64: buy 1 unit at price 801.340027, total balance 9212.949951
day 68, sell 1 unit at price 813.669983, investment 1.538667 %, total balance
```

```
10026.619934,
day 79: buy 1 unit at price 823.210022, total balance 9203.409912
day 82: buy 1 unit at price 829.080017, total balance 8374.329895
day 84, sell 1 unit at price 831.909973, investment 1.056832 %, total balance
9206.239868,
day 86, sell 1 unit at price 838.679993, investment 1.157907 %, total balance
10044.919861,
day 111: buy 1 unit at price 823.559998, total balance 9221.359863
day 116, sell 1 unit at price 843.190002, investment 2.383555 %, total balance
10064.549865,
day 167: buy 1 unit at price 911.710022, total balance 9152.839843
day 169, sell 1 unit at price 918.590027, investment 0.754626 %, total balance
10071.429870,
day 182: buy 1 unit at price 947.799988, total balance 9123.629882
day 183, sell 1 unit at price 934.090027, investment -1.446504 %, total balance
10057.719909,
day 185: buy 1 unit at price 930.500000, total balance 9127.219909
day 187: buy 1 unit at price 930.390015, total balance 8196.829894
day 188, sell 1 unit at price 923.650024, investment -0.736161 %, total balance
9120.479918,
day 190, sell 1 unit at price 929.359985, investment -0.110709 %, total balance
10049.839903,
day 206: buy 1 unit at price 921.289978, total balance 9128.549925
day 207, sell 1 unit at price 929.570007, investment 0.898743 %, total balance
10058.119932,
```

[6]:
```
fig = plt.figure(figsize = (15,5))
plt.plot(close, color='r', lw=2.)
plt.plot(close, '^', markersize=10, color='m', label = 'buying signal',␣
 ↪markevery = states_buy)
plt.plot(close, 'v', markersize=10, color='k', label = 'selling signal',␣
 ↪markevery = states_sell)
plt.title('total gains %f, total investment %f%%'%(total_gains, invest))
plt.legend()
plt.show()
```



[ ]: