16.actor-critic-recurrent-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  \
0  2016-11-02  778.200012  781.650024  763.450012  768.700012  768.700012
1  2016-11-03  767.250000  769.950012  759.030029  762.130005  762.130005
2  2016-11-04  750.659973  770.359985  750.560974  762.020020  762.020020
3  2016-11-07  774.500000  785.190002  772.549988  782.520020  782.520020
4  2016-11-08  783.400024  795.632996  780.190002  790.510010  790.510010

    Volume
0  1872400
1  1943200
2  2134800
3  1585100
4  1350800
```

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

class Actor:
    def __init__(self, name, input_size, output_size, size_layer):
        with tf.variable_scope(name):
            self.X = tf.placeholder(tf.float32, (None, None, input_size))
            self.hidden_layer = tf.placeholder(tf.float32, (None, 2 *␣
↪size_layer))
            cell = tf.nn.rnn_cell.LSTMCell(size_layer, state_is_tuple = False)
            self.rnn,self.last_state = tf.nn.dynamic_rnn(inputs=self.X,␣
↪cell=cell,
                                                        dtype=tf.float32,
```

```
                                                     initial_state=self.
↪hidden_layer)
            self.logits = tf.layers.dense(self.rnn[:,-1], output_size)

class Critic:
    def __init__(self, name, input_size, output_size, size_layer,␣
↪learning_rate):
        with tf.variable_scope(name):
            self.X = tf.placeholder(tf.float32, (None, None, input_size))
            self.Y = tf.placeholder(tf.float32, (None, output_size))
            self.hidden_layer = tf.placeholder(tf.float32, (None, 2 *␣
↪size_layer))
            self.REWARD = tf.placeholder(tf.float32, (None, 1))
            feed_critic = tf.layers.dense(self.X, size_layer, activation = tf.
↪nn.relu)
            cell = tf.nn.rnn_cell.LSTMCell(size_layer, state_is_tuple = False)
            self.rnn,self.last_state = tf.nn.dynamic_rnn(inputs=self.X,␣
↪cell=cell,
                                                     dtype=tf.float32,
                                                     initial_state=self.
↪hidden_layer)
            feed_critic = tf.layers.dense(self.rnn[:,-1], output_size,␣
↪activation = tf.nn.relu) + self.Y
            feed_critic = tf.layers.dense(feed_critic, size_layer//2,␣
↪activation = tf.nn.relu)
            self.logits = tf.layers.dense(feed_critic, 1)
            self.cost = tf.reduce_mean(tf.square(self.REWARD - self.logits))
            self.optimizer = tf.train.AdamOptimizer(learning_rate).
↪minimize(self.cost)

class Agent:

    LEARNING_RATE = 0.001
    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
    COPY = 1000
    T_COPY = 0

    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.INITIAL_FEATURES = np.zeros((4, self.state_size))
        self.skip = skip
        tf.reset_default_graph()
        self.actor = Actor('actor-original', self.state_size, self.OUTPUT_SIZE, self.LAYER_SIZE)
        self.actor_target = Actor('actor-target', self.state_size, self.OUTPUT_SIZE, self.LAYER_SIZE)
        self.critic = Critic('critic-original', self.state_size, self.OUTPUT_SIZE, self.LAYER_SIZE, self.LEARNING_RATE)
        self.critic_target = Critic('critic-target', self.state_size, self.OUTPUT_SIZE,
                                    self.LAYER_SIZE, self.LEARNING_RATE)
        self.grad_critic = tf.gradients(self.critic.logits, self.critic.Y)
        self.actor_critic_grad = tf.placeholder(tf.float32, [None, self.OUTPUT_SIZE])
        weights_actor = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope='actor')
        self.grad_actor = tf.gradients(self.actor.logits, weights_actor, -self.actor_critic_grad)
        grads = zip(self.grad_actor, weights_actor)
        self.optimizer = tf.train.AdamOptimizer(self.LEARNING_RATE).apply_gradients(grads)
        self.sess = tf.InteractiveSession()
        self.sess.run(tf.global_variables_initializer())

    def _assign(self, from_name, to_name):
        from_w = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope=from_name)
        to_w = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, scope=to_name)
        for i in range(len(from_w)):
            assign_op = to_w[i].assign(from_w[i])
            self.sess.run(assign_op)

    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 _select_action(self, state):
        if np.random.rand() < self.EPSILON:
```

```
            action = np.random.randint(self.OUTPUT_SIZE)
        else:
            prediction = self.sess.run(self.actor.logits, feed_dict={self.actor.
↪X:[state]})[0]
            action = np.argmax(prediction)
        return action

    def _construct_memories_and_train(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.actor.logits, feed_dict={self.actor.X: states,
                                                          self.actor.hidden_layer:␣
↪init_values})
        Q_target = self.sess.run(self.actor_target.logits, feed_dict={self.
↪actor_target.X: states,
                                                                       self.
↪actor_target.hidden_layer: init_values})
        grads = self.sess.run(self.grad_critic, feed_dict={self.critic.X:
↪states, self.critic.Y:Q,
                                                             self.critic.
↪hidden_layer: init_values})[0]
        self.sess.run(self.optimizer, feed_dict={self.actor.X:states, self.
↪actor_critic_grad:grads,
                                                  self.actor.hidden_layer:␣
↪init_values})

        rewards = np.array([a[2] for a in replay]).reshape((-1, 1))
        rewards_target = self.sess.run(self.critic_target.logits,
                                       feed_dict={self.critic_target.X:
↪new_states,self.critic_target.Y:Q_target,
                                                  self.critic_target.
↪hidden_layer: init_values})
        for i in range(len(replay)):
            if not replay[0][-2]:
                rewards[i] += self.GAMMA * rewards_target[i]
        cost, _ = self.sess.run([self.critic.cost, self.critic.optimizer],
                                feed_dict={self.critic.X:states, self.critic.Y:
↪Q, self.critic.REWARD:rewards,
                                           self.critic.hidden_layer:␣
↪init_values})
        return cost

    def get_state(self, t):
        window_size = self.window_size + 1
        d = t - window_size + 1
```

```python
        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):

            if np.random.rand() < self.EPSILON:
                action = np.random.randint(self.OUTPUT_SIZE)
            else:
                action, last_state = self.sess.run([self.actor.logits,
                                                  self.actor.last_state],
                                                  feed_dict={self.actor.X:[self.INITIAL_FEATURES],
                                                             self.actor.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 (self.T_COPY + 1) % self.COPY == 0:
                    self._assign('actor-original', 'actor-target')
                    self._assign('critic-original', 'critic-target')

                if np.random.rand() < self.EPSILON:
                    action = np.random.randint(self.OUTPUT_SIZE)
                else:
                    action, last_state = self.sess.run([self.actor.logits,
                                                  self.actor.last_state],
                                                  feed_dict={self.actor.X:[self.
↪INITIAL_FEATURES],
                                                             self.actor.
↪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])
                batch_size = min(len(self.MEMORIES), self.BATCH_SIZE)
                self.INITIAL_FEATURES = new_state
                replay = random.sample(self.MEMORIES, batch_size)
                cost = self._construct_memories_and_train(replay)
                self.T_COPY += 1
                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
0x7f46cd19b6d8>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7f46cd102ef0>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7f46ccc7ce10>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell_impl.LSTMCell object at
0x7f46cc5685f8>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
epoch: 10, total rewards: 1158.549991.3, cost: 0.046632, total money:
4247.099979
epoch: 20, total rewards: 466.185119.3, cost: 0.035100, total money: 5537.135131
epoch: 30, total rewards: 477.615173.3, cost: 0.330107, total money: 975.775206
epoch: 40, total rewards: 1200.205012.3, cost: 0.215860, total money:
```

```
10180.934992
epoch: 50, total rewards: 283.615237.3, cost: 0.116108, total money: 3314.845217
epoch: 60, total rewards: 324.265078.3, cost: 0.435482, total money: 9334.585085
epoch: 70, total rewards: 587.429873.3, cost: 0.749076, total money: 4785.129884
epoch: 80, total rewards: 1248.729918.3, cost: 0.167420, total money: 663.739866
epoch: 90, total rewards: 520.270204.3, cost: 0.006982, total money: 9503.630189
epoch: 100, total rewards: 195.270142.3, cost: 0.153058, total money:
10195.270142
epoch: 110, total rewards: 74.399840.3, cost: 0.350105, total money:
10074.399840
epoch: 120, total rewards: 2842.805359.3, cost: 0.074852, total money:
7832.085327
epoch: 130, total rewards: 509.049985.3, cost: 0.053447, total money:
8518.609983
epoch: 140, total rewards: -2.900205.3, cost: 0.015182, total money: 8979.989810
epoch: 150, total rewards: 93.080022.3, cost: 0.008775, total money:
10093.080022
epoch: 160, total rewards: 89.794983.3, cost: 0.107893, total money:
10089.794983
epoch: 170, total rewards: 222.045106.3, cost: 0.189179, total money:
10222.045106
epoch: 180, total rewards: -57.619995.3, cost: 0.002425, total money:
8925.739990
epoch: 190, total rewards: 21.009889.3, cost: 0.005919, total money:
10021.009889
epoch: 200, total rewards: 201.354980.3, cost: 0.002352, total money:
10201.354980
```

[5]:

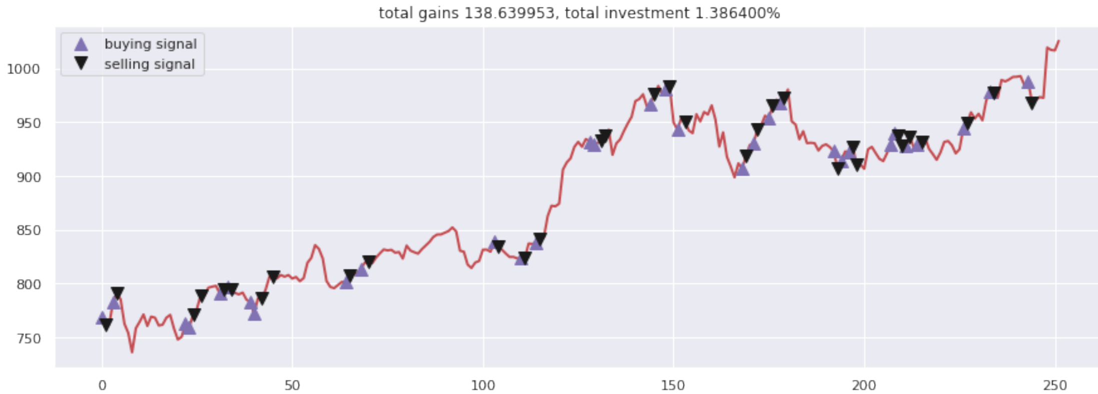
```
states_buy, states_sell, total_gains, invest = agent.buy(initial_money =
initial_money)
```

```
day 0: buy 1 unit at price 768.700012, total balance 9231.299988
day 1, sell 1 unit at price 762.130005, investment -0.854691 %, total balance
9993.429993,
day 3: buy 1 unit at price 782.520020, total balance 9210.909973
day 4, sell 1 unit at price 790.510010, investment 1.021059 %, total balance
10001.419983,
day 22: buy 1 unit at price 762.520020, total balance 9238.899963
day 23: buy 1 unit at price 759.109985, total balance 8479.789978
day 24, sell 1 unit at price 771.190002, investment 1.137017 %, total balance
9250.979980,
day 26, sell 1 unit at price 789.289978, investment 3.975708 %, total balance
10040.269958,
day 31: buy 1 unit at price 790.799988, total balance 9249.469970
day 32, sell 1 unit at price 794.200012, investment 0.429947 %, total balance
10043.669982,
day 33: buy 1 unit at price 796.419983, total balance 9247.249999
day 34, sell 1 unit at price 794.559998, investment -0.233543 %, total balance
```

```
10041.809997,
day 39: buy 1 unit at price 782.789978, total balance 9259.020019
day 40: buy 1 unit at price 771.820007, total balance 8487.200012
day 42, sell 1 unit at price 786.900024, investment 0.525051 %, total balance
9274.100036,
day 45, sell 1 unit at price 806.650024, investment 4.512712 %, total balance
10080.750060,
day 64: buy 1 unit at price 801.340027, total balance 9279.410033
day 65, sell 1 unit at price 806.969971, investment 0.702566 %, total balance
10086.380004,
day 68: buy 1 unit at price 813.669983, total balance 9272.710021
day 70, sell 1 unit at price 820.450012, investment 0.833265 %, total balance
10093.160033,
day 103: buy 1 unit at price 838.549988, total balance 9254.610045
day 104, sell 1 unit at price 834.570007, investment -0.474627 %, total balance
10089.180052,
day 110: buy 1 unit at price 824.320007, total balance 9264.860045
day 111, sell 1 unit at price 823.559998, investment -0.092198 %, total balance
10088.420043,
day 114: buy 1 unit at price 838.210022, total balance 9250.210021
day 115, sell 1 unit at price 841.650024, investment 0.410399 %, total balance
10091.860045,
day 128: buy 1 unit at price 932.169983, total balance 9159.690062
day 129: buy 1 unit at price 928.780029, total balance 8230.910033
day 131, sell 1 unit at price 932.219971, investment 0.005363 %, total balance
9163.130004,
day 132, sell 1 unit at price 937.080017, investment 0.893644 %, total balance
10100.210021,
day 144: buy 1 unit at price 966.950012, total balance 9133.260009
day 145, sell 1 unit at price 975.599976, investment 0.894562 %, total balance
10108.859985,
day 148: buy 1 unit at price 980.940002, total balance 9127.919983
day 149, sell 1 unit at price 983.409973, investment 0.251796 %, total balance
10111.329956,
day 151: buy 1 unit at price 942.900024, total balance 9168.429932
day 153, sell 1 unit at price 950.760010, investment 0.833597 %, total balance
10119.189942,
day 168: buy 1 unit at price 906.690002, total balance 9212.499940
day 169, sell 1 unit at price 918.590027, investment 1.312469 %, total balance
10131.089967,
day 171: buy 1 unit at price 930.090027, total balance 9200.999940
day 172, sell 1 unit at price 943.830017, investment 1.477275 %, total balance
10144.829957,
day 175: buy 1 unit at price 953.419983, total balance 9191.409974
day 176, sell 1 unit at price 965.400024, investment 1.256533 %, total balance
10156.809998,
day 178: buy 1 unit at price 968.150024, total balance 9188.659974
day 179, sell 1 unit at price 972.919983, investment 0.492688 %, total balance
```

```
10161.579957,
day 192: buy 1 unit at price 922.900024, total balance 9238.679933
day 193, sell 1 unit at price 907.239990, investment -1.696829 %, total balance
10145.919923,
day 194: buy 1 unit at price 914.390015, total balance 9231.529908
day 196: buy 1 unit at price 922.219971, total balance 8309.309937
day 197, sell 1 unit at price 926.960022, investment 1.374688 %, total balance
9236.269959,
day 198, sell 1 unit at price 910.979980, investment -1.218797 %, total balance
10147.249939,
day 207: buy 1 unit at price 929.570007, total balance 9217.679932
day 208: buy 1 unit at price 939.330017, total balance 8278.349915
day 209, sell 1 unit at price 937.340027, investment 0.835872 %, total balance
9215.689942,
day 210, sell 1 unit at price 928.450012, investment -1.158273 %, total balance
10144.139954,
day 211: buy 1 unit at price 927.809998, total balance 9216.329956
day 212, sell 1 unit at price 935.950012, investment 0.877336 %, total balance
10152.279968,
day 214: buy 1 unit at price 929.080017, total balance 9223.199951
day 215, sell 1 unit at price 932.070007, investment 0.321823 %, total balance
10155.269958,
day 226: buy 1 unit at price 944.489990, total balance 9210.779968
day 227, sell 1 unit at price 949.500000, investment 0.530446 %, total balance
10160.279968,
day 233: buy 1 unit at price 978.890015, total balance 9181.389953
day 234, sell 1 unit at price 977.000000, investment -0.193077 %, total balance
10158.389953,
day 243: buy 1 unit at price 988.200012, total balance 9170.189941
day 244, sell 1 unit at price 968.450012, investment -1.998583 %, total balance
10138.639953,
```

[6]:
```python
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()
```


total gains 138.639953, total investment 1.386400%
buying signal
selling signal
1000
950
900
850
800
750
0
50
100
150
200
250

[ ]: