Recommending Cryptocurrency Trading Points with Deep Reinforcement Learning Approach

https://www.mdpi.com/2076-3417/10/4/1506

Why?

Cryptocurrency

- ✓ No breaking points
- ✓ Little external factors

Why?

Bitcoin : 14.4% Ethereum : 41% Litecoin : 74% (in a month)

with “Simple” model structure

Data : Hourly historical data (https://www.cryptodatadownload.com/)

Unix Timestamp	Date	Symbol	Open	High	Low	Close	Volume BTC	Volume KRW
1605855600	2020-11-20 07-AM	BTCKRW	20140000	20340000	20061000	20179000	361.41	7302968556
1605852000	2020-11-20 06-AM	BTCKRW	20095000	20140000	20048000	20140000	204.61	4112172908
1605848400	2020-11-20 05-AM	BTCKRW	20111000	20147000	20000000	20095000	264.29	5304702287
1605844800	2020-11-20 04-AM	BTCKRW	19890000	20151000	19866000	20111000	364.52	7308066180
1605841200	2020-11-20 03-AM	BTCKRW	19927000	19949000	19866000	19890000	103.52	2061272509

State : 10-row price information

Action : Buy, Sell, Hold

Reward : Sell Price – Last purchase price

Technique #1 : number_of_*

Algorithm 1. Reward Function Algorithm

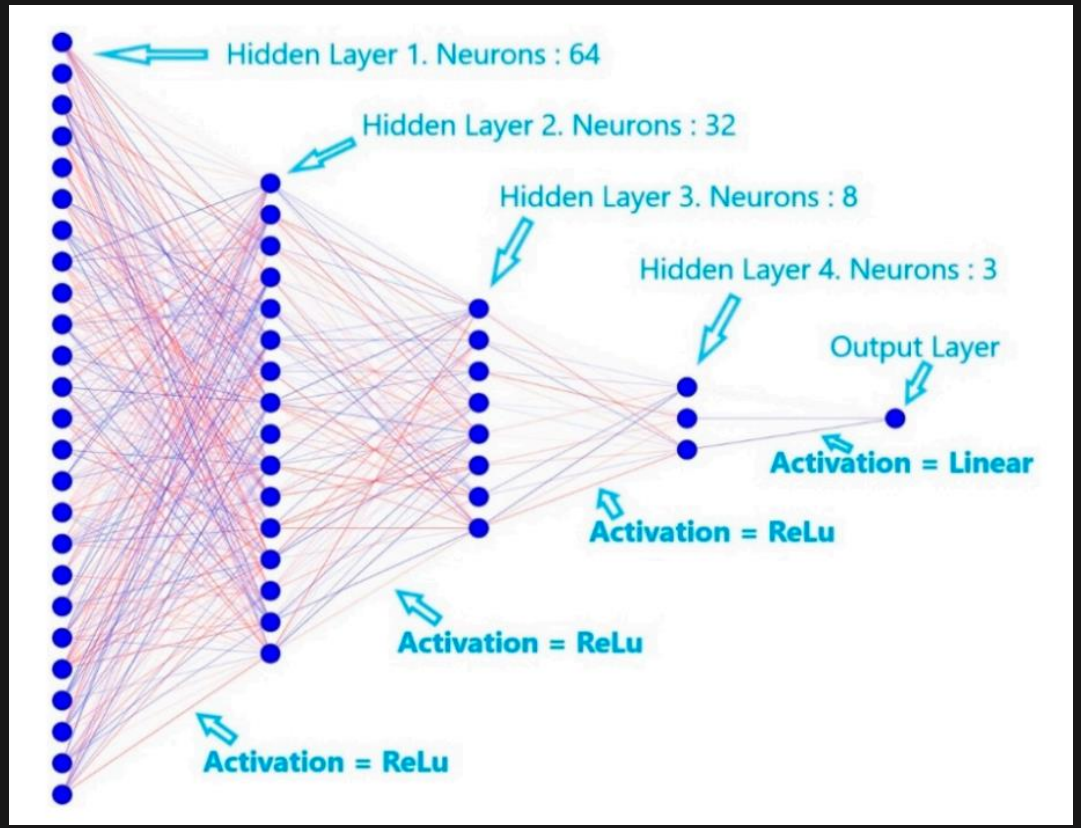
```
repeat (until data is over) // start trading
reward = 0
number_of_purchases = 0
number_of_sales = 0
number_of_holds = 0
choose one of three actions: // buy, sell, and hold
if (action = "buy"):
number_of_purchases++
// Add to the number_of_purchases one
number_of_holds = 0
// Declare number_of_holds as zero
if (number_of_purchases > 20):
Decrease the reward value
if (action = "hold"):
number_of_holds++
// Add to the number_of_holds one
if (number_of_holds > 20):
Decrease the reward value
if (action = "sell"):
number_of_holds = 0
// Declare number_of_holds as zero
number_of_purchases = 0
// Declare number_of_purchases as zero
reward ← (sell price) – (last purchase price)
end loop
```

Technique #2 : Action Threshold

Threshold > Highest Confidence indicator

→ Action : hold

Model Structure



Result

Table 3. Main information about the trading process tested with different training time.

Training Time	Invested Money for Trading ($)	Trade with	"Buy" Actions	"Sell" Actions	Transaction Fee ($)	Money after Trading ($)	Quality of Trading (%)
70-time	1 000 000	DC strategy	63	63	699.2	1 000 266	100.02 (Grew 0.02)
		hold position	0	0	0	980,637	98 (Lost 2)
80-time	1 000 000	DC strategy	150	150	1 626.9	996 214.8	Lost 0.4
		hold position	0	0	0	980 637	Lost 2
90-time	1 000 000	DC strategy	262	230	2 663.1	996 029.8	Lost 0.4
		hold position	0	0	0	980 637	Lost 2
100-time	1 000 000	DC strategy	268	218	2 677.9	994 529.5	Lost 0.6
		hold position	0	0	0	980 637	Lost 2
450-time	1 000 000	DC strategy	510	396	5 577.0	1 155 658	115.5 (Grew 15.5)
		hold position	0	0	0	980 637	98 (Lost 2)

Result

Table 4. Experiment results on Bitcoin's data.

	Invested Money ($)	Number of Actions	Money after Trading ($)	Quality of Trading %
Double Cross Strategy	48 124	14	48 563	100.9 (Grew 0.9)
Swing Trading	48 124	5	48 469	100.7 (Grew 0.7)
Scalping Trading	48 124	130	51 073	106.1 (Grew 6.1)
DRL Application	1 000 000	400	1 144 961	114.4 (Grew 14.4)

Table 5. Experiment results on Litecoin's data.

	Invested Money ($)	Number of Actions	Money after Trading ($)	Quality of Trading %
Double Cross Strategy	10 814	12	10 576	97.8 (Lost 2.2)
Swing Trading	10 814	4	11 217	103.6 (Grew 3.6)
Scalping Trading	10 814	134	13 382	123.7 (Grew 23.7)
DRL Application	10 000	642	17 467	174.6 (Grew 74.6)

Table 6. Experiment results on Ethereum's data.

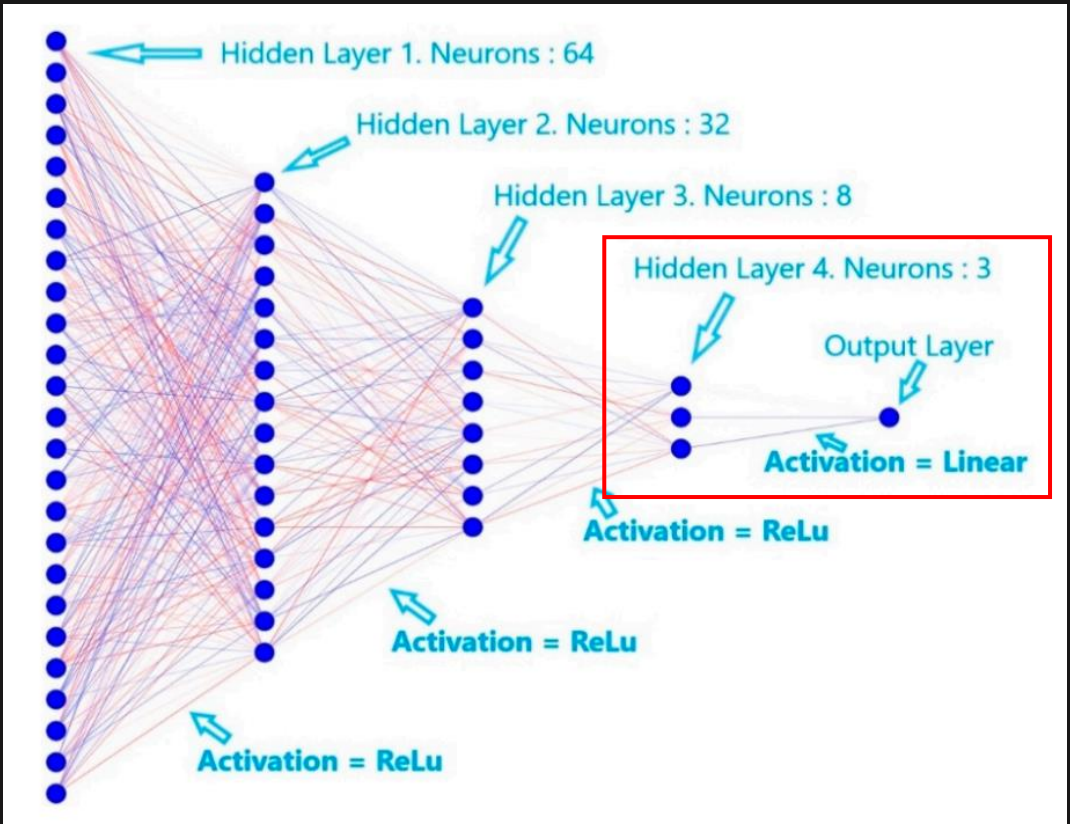
	Invested Money ($)	Number of Actions	Money after Trading ($)	Quality of Trading %
Double Cross Strategy	11 739	9	11 400	97.1 (Lost 2.9)
Swing Trading	11 739	4	10 643	90.6 (Lost 9.4)
Scalping Trading	11 739	112	11 462	97.6 (Lost 2.4)
DRL Application	10 000	294	14 140	141.4 (Grew 41.4)

Questions

Algorithm 1. Reward Function Algorithm

1: ***repeat*** *(until data is over) // start trading*
2: *reward = 0*
3: *number_of_purchases = 0*
4: *number_of_sales = 0*
5: *number_of_holds = 0*
6: *choose one of three actions: // buy, sell, and hold*
7: *if (action = "buy"):*
8: number_of_purchases++
9: *// Add to the number_of_purchases one*
10: number_of_holds = 0
11: *// Declare number_of_holds as zero*
12: *if (number_of_purchases > 20):*
13: *Decrease the reward value*
14: *if (action = "hold"):*
15: number_of_holds++
16: *// Add to the number_of_holds one*
17: *if (number_of_holds > 20):*
18: *Decrease the reward value*
19: *if (action = "sell"):*
20: number_of_holds = 0
21: *// Declare number_of_holds as zero*
22: number_of_purchases = 0
23: *// Declare number_of_purchases as zero*
24: *reward* ← *(sell price) – (last purchase price)*
25: ***end loop***

Questions


Hidden Layer 1. Neurons : 64
Hidden Layer 2. Neurons : 32
Hidden Layer 3. Neurons : 8
Hidden Layer 4. Neurons : 3
Output Layer
Activation = Linear
Activation = ReLu
Activation = ReLu
Activation = ReLu

Questions

```
with tf.device('/CPU:0'):
    for epoch in range(500):
        epoch_reward = 0
        idx = 0
        for state in tqdm(result[:-1]):

            action_raw = agent.make_action(state)

            if action_raw.any()>0.8:
                action = np.argmax(action)
            else:
                action = 2

            if action == 0: #buy
                agent.num_of_hold = 0
                agent.num_of_buy += 1
                agent.total_amount += state[-2]
                reward = 0
                if agent.num_of_buy > 20:
                    reward -= 20
            elif action == 1: #sell
                reward = (agent.total_amount - agent.num_of_buy * state[-2])/agent.num_of_buy
                agent.num_of_buy = 0
                agent.num_of_hold = 0
                agent.total_amount = 0
            else:
                agent.num_of_hold += 1
                reward = 0
                if agent.num_of_hold > 20:
                    reward -= 20
            n_state = result[idx+1]
            agent.memorize(state,action_raw,reward,0,n_state) #batch 저장
            agent.replay(1)
            epoch_reward += reward
            idx += 1
        agent.total_reward.append(epoch_reward)

#         with open('C:/Users/lchan/OneDrive - 고려대학교/Python/Quant/agent/actor_critic_1/total_reward.pickle', 'wb') as f:
#             pickle.dump(agent.total_reward, f)

94%|                                                                                | 18709/19999 [47:48<03:42,  5.79it/s]
```

Questions

- ✓ Last purchase price?
- ✓ Last node structure
- ✓ Enhancing simulation iteration
- ✓ Input State