ELEN_6885_HW4_Part_4

November 25, 2019

#ELEN 6885 Reinforcement Learning Coding Assignment (Part 4)# There are a lot of official and unofficial tutorials about Tensorflow, and there are also many open-source projects written in Tensorflow. You can refer to those resources according to your interest. In this part of homework 4, only knowledge of Deep Reinforcement Learning and basic programming skills will be needed.

Please put your code into the block marked by
############################
YOUR CODE STARTS HERE
YOUR CODE ENDS HERE
############################
Normally you don't need to edit anything outside of the block. If you do want to edit something, please use a similar manner to mark you edits.

[1]:
```
import numpy as np
import tensorflow as tf

# DQN
class DQN:
  def __init__(
      self,
      actions_num,
      state_size,
      learning_rate = 0.001,
      gamma = 0.99,
      epsilon_min = 0.05,
      epsilon_start = 0.9,
      replace_target_iter = 300,
      memory_size = 500,
      batch_size = 2,
      epsilon_increment = None,
  ):
      self.actions_num = actions_num
      self.state_size = state_size
      self.lr = learning_rate
      self.gamma = gamma
      self.epsilon_min = epsilon_min
      self.replace_target_iter = replace_target_iter
      self.memory_size = memory_size
```

```python
      self.batch_size = batch_size
      self.epsilon_increment = epsilon_increment
      self.epsilon = epsilon_start if epsilon_increment is not None else self.epsilon_min
      self.save_model_path = './weights/DQN_model.ckpt'
      self.memory_counter = 0

      # learned steps counter
      self.steps_counter = 0

      # initialize memory [s, a, r, s_, done]
      self.memory = np.zeros((self.memory_size, state_size * 2 + 3))

      # build target_net and q_net
      self.build_net()
      t_params = tf.get_collection('target_net_params')
      q_params = tf.get_collection('q_net_params')
      self.replace_target = [tf.assign(t, q) for t, q in zip(t_params, q_params)]

      # gpu setting
      config = tf.ConfigProto(log_device_placement=False, allow_soft_placement=True)
      config.gpu_options.per_process_gpu_memory_fraction = 0.6
      self.sess = tf.Session(config=config)

      self.sess.run(tf.global_variables_initializer())

  def build_net(self):
    # build q_net
    self.state = tf.placeholder(tf.float32, [None, self.state_size], name='state')
    self.q_target = tf.placeholder(tf.float32, [None, self.actions_num], name='Q_target')
    with tf.variable_scope('q_net'):
      # c_names(collections_names) are the collections to store variables
      c_names, neurons_layer_1, w_initializer, b_initializer = \
        ['q_net_params', tf.GraphKeys.GLOBAL_VARIABLES], 100, \
        tf.random_normal_initializer(0., 0.3), tf.constant_initializer(0.1)

      # layer 1
      with tf.variable_scope('layer_1'):
        w_layer_1 = tf.get_variable('w_layer_1', [self.state_size, neurons_layer_1], initializer=w_initializer, collections=c_names)
        b_layer_1 = tf.get_variable('b_layer_1', [1, neurons_layer_1], initializer=b_initializer, collections=c_names)
```

```python
        layer_1 = tf.nn.relu(tf.matmul(self.state, w_layer_1) + b_layer_1)

      # layer 2
      with tf.variable_scope('layer_2'):
        w_layer_2 = tf.get_variable('w_layer_2', [neurons_layer_1, self.actions_num], initializer=w_initializer, collections=c_names)
        b_layer_2 = tf.get_variable('b_layer_2', [1, self.actions_num], initializer=b_initializer, collections=c_names)
        self.q_value = tf.matmul(layer_1, w_layer_2) + b_layer_2

    with tf.variable_scope('loss'):
      self.loss = tf.reduce_mean(tf.squared_difference(self.q_target, self.q_value))
    with tf.variable_scope('train'):
      self._train_op = tf.train.AdamOptimizer(self.lr).minimize(self.loss)

    # build target_net
    self.state_t = tf.placeholder(tf.float32, [None, self.state_size], name='state_t')    # input
    with tf.variable_scope('target_net'):
      # c_names(collections_names) are the collections to store variables
      c_names = ['target_net_params', tf.GraphKeys.GLOBAL_VARIABLES]

      # layer 1
      with tf.variable_scope('layer_1'):
        w_layer_1 = tf.get_variable('w_layer_1', [self.state_size, neurons_layer_1], initializer=w_initializer, collections=c_names)
        b_layer_1 = tf.get_variable('b_layer_1', [1, neurons_layer_1], initializer=b_initializer, collections=c_names)
        layer_1 = tf.nn.relu(tf.matmul(self.state_t, w_layer_1) + b_layer_1)

      # layer 2

      ############################
      # YOUR CODE STARTS HERE

      with tf.variable_scope('layer_2'):
        w_layer_2 = tf.get_variable('w_layer_2', [neurons_layer_1, self.actions_num], initializer=w_initializer, collections=c_names)
        b_layer_2 = tf.get_variable('b_layer_2', [1, self.actions_num], initializer=b_initializer, collections=c_names)
        self.q_next = tf.matmul(layer_1, w_layer_2) + b_layer_2

      # YOUR CODE ENDS HERE
      ############################
```

```python
  def store_transition(self, s, a, r, s_, done):
    s=s.reshape(-1)
    s_=s_.reshape(-1)
    transition = np.hstack((s, [a, r], s_, done))

    # replace the old memory with new observations
    index = self.memory_counter % self.memory_size
    self.memory[index, :] = transition

    self.memory_counter += 1

  def choose_action(self, observation):
    # to have batch dimension when fed into tf placeholder
    observation = observation[np.newaxis, :]
    # epsilon-greedy
    if np.random.uniform() > self.epsilon:
      action_values = self.sess.run(self.q_value, feed_dict={self.state: observation})
      action = np.argmax(action_values)
    else:
      action = np.random.randint(0, self.actions_num)
    return action

  def learn(self):
    # replace target parameters every once a while
    if self.steps_counter % self.replace_target_iter == 0:
      self.sess.run(self.replace_target)

    # sample a batch from the memory
    if self.memory_counter > self.memory_size:
      sample_index = np.random.choice(self.memory_size, size=self.batch_size)
    else:
      sample_index = np.random.choice(self.memory_counter, size=self.batch_size)
    batch_memory = self.memory[sample_index, :]

    q_next, q_value = self.sess.run(
      [self.q_next, self.q_value],
      feed_dict={
        self.state_t: batch_memory[:, -self.state_size-1:-1],  # fixed params
        self.state: batch_memory[:, :self.state_size],  # newest params
      })

    # calculate q_target
    q_target = q_value.copy()

    # only change the action-values of this batch, because we only calculate loss on the batch observations
```

```
    batch_index = np.arange(self.batch_size, dtype=np.int32)
    act_index = batch_memory[:, self.state_size].astype(int)
    reward = batch_memory[:, self.state_size + 1]
    done = batch_memory[:, -1]
    ############################
    # YOUR CODE STARTS HERE

    for batch in batch_index:
      q_target[batch, act_index[batch]] = reward[batch] + (0 if done[batch]␣
↪else (self.gamma * np.max(q_next[batch,:],axis=0)))

    # YOUR CODE ENDS HERE
    ############################

    # train q_net
    _, self.cost = self.sess.run([self._train_op, self.loss],
                                 feed_dict={self.state: batch_memory[:, :self.
↪state_size],
                                            self.q_target: q_target})
    # change epsilon
    self.epsilon = self.epsilon - self.epsilon_increment if self.epsilon > self.
↪epsilon_min else self.epsilon_min
    self.steps_counter += 1

  def store(self):
    saver = tf.train.Saver()
    saver.save(self.sess, self.save_model_path)

  def restore(self):
    saver = tf.train.Saver()
    saver.restore(self.sess, self.save_model_path)
```

[2]:
```
import gym
# cart pole gym environment
env = gym.make("CartPole-v0")
env._max_episode_steps = 500
# state and action space
print(env.action_space)
print(env.observation_space)
# observation
env.reset()
# state, reward, done, info
print(env.step(1))
```

```
Discrete(2)
Box(4,)
(array([ 0.01000824,  0.2117136 ,  0.02151308, -0.25150414]), 1.0, False, {})
```

[3]:
```python
# play the game and train the network
np.set_printoptions(threshold=np.inf)
episode_length_set = []
tf.reset_default_graph()
total_time_steps = 100000

RL = DQN(actions_num = 2, gamma = 0.99,
         state_size = 4, epsilon_start = 1,
         learning_rate = 1e-3, epsilon_min = 0.01,
         replace_target_iter = 100, memory_size = 5000,
         epsilon_increment = 0.00001,)

new_state = env.reset()
done = False
episode_length_counter = 0
for step in range(total_time_steps):
  ############################
  # YOUR CODE STARTS HERE

    if done:
        done = False
        new_state = env.reset()
        episode_length_set.append(episode_length_counter)
        print(step,episode_length_counter)
        episode_length_counter = 0
        continue
    else:
        # take action
        action = RL.choose_action(new_state)
        old_state = new_state
        new_state, reward, done, info = env.step(action)
        RL.store_transition(old_state, action, reward, new_state, done)

  # YOUR CODE ENDS HERE
  ############################

    if step > 200:
        RL.learn()
    episode_length_counter += 1
    if episode_length_counter == 500:
        RL.store()
# RL.store()
```

```
29 29
44 14
89 44
119 29
```

148 28
160 11
177 16
189 11
203 13
219 15
234 14
300 65
343 42
368 24
381 12
394 12
416 21
428 11
440 11
455 14
478 22
493 14
505 11
537 31
561 23
574 12
596 21
610 13
626 15
643 16
671 27
698 26
716 17
729 12
745 15
760 14
777 16
797 19
812 14
828 15
866 37
897 30
910 12
928 17
938 9
949 10
968 18
985 16
1003 17
1016 12
1039 22
1057 17

1108	50
1128	19
1143	14
1166	22
1183	16
1199	15
1216	16
1240	23
1265	24
1283	17
1304	20
1326	21
1380	53
1408	27
1419	10
1449	29
1464	14
1480	15
1507	26
1528	20
1543	14
1570	26
1591	20
1614	22
1670	55
1690	19
1717	26
1739	21
1785	45
1828	42
1843	14
1856	12
1879	22
1904	24
1916	11
1927	10
1944	16
1966	21
1980	13
1997	16
2008	10
2025	16
2047	21
2063	15
2077	13
2091	13
2114	22
2179	64

2193	13
2225	31
2248	22
2261	12
2305	43
2339	33
2366	26
2385	18
2397	11
2413	15
2426	12
2448	21
2479	30
2499	19
2515	15
2541	25
2558	16
2584	25
2599	14
2609	9
2637	27
2666	28
2712	45
2775	62
2846	70
2859	12
2889	29
2927	37
2942	14
2962	19
3000	37
3020	19
3043	22
3054	10
3080	25
3109	28
3126	16
3178	51
3195	16
3223	27
3234	10
3246	11
3268	21
3285	16
3298	12
3330	31
3356	25
3377	20

3404	26
3421	16
3434	12
3449	14
3498	48
3516	17
3536	19
3564	27
3578	13
3591	12
3605	13
3621	15
3657	35
3676	18
3742	65
3788	45
3802	13
3835	32
3853	17
3874	20
3896	21
3923	26
3939	15
3951	11
3972	20
3987	14
4004	16
4045	40
4076	30
4101	24
4122	20
4142	19
4167	24
4189	21
4208	18
4222	13
4246	23
4281	34
4300	18
4324	23
4428	103
4448	19
4458	9
4469	10
4486	16
4501	14
4518	16
4538	19

4552	13
4574	21
4599	24
4618	18
4647	28
4658	10
4679	20
4705	25
4777	71
4793	15
4813	19
4850	36
4867	16
4894	26
4922	27
4939	16
4954	14
4969	14
4981	11
5002	20
5021	18
5034	12
5053	18
5071	17
5094	22
5114	19
5144	29
5166	21
5188	21
5203	14
5226	22
5239	12
5265	25
5293	27
5309	15
5343	33
5372	28
5389	16
5403	13
5433	29
5462	28
5478	15
5501	22
5531	29
5549	17
5565	15
5580	14
5608	27

5629	20
5656	26
5682	25
5702	19
5727	24
5749	21
5771	21
5794	22
5804	9
5825	20
5854	28
5881	26
5910	28
5919	8
5958	38
5992	33
6017	24
6032	14
6058	25
6086	27
6122	35
6142	19
6160	17
6176	15
6193	16
6216	22
6234	17
6271	36
6295	23
6316	20
6345	28
6367	21
6379	11
6417	37
6432	14
6473	40
6528	54
6541	12
6585	43
6605	19
6667	61
6680	12
6692	11
6711	18
6735	23
6746	10
6778	31
6798	19

6836	37
6869	32
6879	9
6890	10
6907	16
6927	19
6945	17
6965	19
6975	9
7002	26
7014	11
7031	16
7062	30
7128	65
7148	19
7161	12
7195	33
7212	16
7344	131
7384	39
7406	21
7433	26
7455	21
7475	19
7508	32
7526	17
7561	34
7597	35
7615	17
7634	18
7647	12
7661	13
7688	26
7699	10
7756	56
7773	16
7793	19
7803	9
7819	15
7894	74
7913	18
7936	22
7953	16
7974	20
8004	29
8021	16
8042	20
8120	77

8135	14
8176	40
8192	15
8221	28
8235	13
8253	17
8267	13
8290	22
8308	17
8330	21
8371	40
8412	40
8435	22
8458	22
8488	29
8511	22
8531	19
8547	15
8587	39
8607	19
8638	30
8654	15
8705	50
8723	17
8766	42
8780	13
8798	17
8819	20
8871	51
8919	47
8937	17
8953	15
8971	17
8987	15
9002	14
9029	26
9070	40
9088	17
9148	59
9185	36
9203	17
9224	20
9238	13
9255	16
9272	16
9303	30
9336	32
9348	11

9378 29
9403 24
9455 51
9484 28
9513 28
9532 18
9547 14
9561 13
9584 22
9619 34
9641 21
9660 18
9735 74
9746 10
9791 44
9806 14
9838 31
9858 19
9887 28
9943 55
9973 29
10016 42
10048 31
10079 30
10099 19
10135 35
10169 33
10180 10
10228 47
10257 28
10272 14
10312 39
10328 15
10344 15
10355 10
10382 26
10401 18
10413 11
10450 36
10475 24
10498 22
10512 13
10551 38
10566 14
10579 12
10594 14
10619 24
10635 15

10661	25
10674	12
10717	42
10728	10
10742	13
10756	13
10799	42
10824	24
10885	60
10911	25
10944	32
11007	62
11023	15
11062	38
11125	62
11146	20
11162	15
11181	18
11198	16
11225	26
11266	40
11285	18
11298	12
11324	25
11376	51
11405	28
11433	27
11482	48
11511	28
11523	11
11579	55
11593	13
11637	43
11667	29
11683	15
11707	23
11723	15
11738	14
11768	29
11793	24
11820	26
11831	10
11853	21
11886	32
11901	14
11913	11
11965	51
11984	18

12002	17
12031	28
12046	14
12066	19
12105	38
12155	49
12180	24
12198	17
12210	11
12222	11
12247	24
12269	21
12285	15
12307	21
12319	11
12381	61
12397	15
12418	20
12456	37
12467	10
12493	25
12525	31
12542	16
12563	20
12580	16
12619	38
12641	21
12677	35
12727	49
12760	32
12790	29
12816	25
12830	13
12863	32
12890	26
12902	11
12917	14
12958	40
12977	18
12994	16
13054	59
13097	42
13159	61
13225	65
13240	14
13267	26
13302	34
13348	45

13377	28
13392	14
13426	33
13442	15
13520	77
13549	28
13583	33
13621	37
13654	32
13712	57
13727	14
13775	47
13796	20
13829	32
13922	92
13953	30
13991	37
14045	53
14060	14
14110	49
14148	37
14167	18
14205	37
14256	50
14274	17
14316	41
14337	20
14350	12
14364	13
14398	33
14431	32
14464	32
14502	37
14526	23
14577	50
14599	21
14639	39
14661	21
14703	41
14717	13
14739	21
14764	24
14784	19
14796	11
14816	19
14846	29
14873	26
14901	27

14920	18
14941	20
14989	47
15034	44
15132	97
15149	16
15183	33
15217	33
15240	22
15260	19
15276	15
15317	40
15333	15
15393	59
15422	28
15433	10
15454	20
15479	24
15500	20
15522	21
15540	17
15588	47
15638	49
15657	18
15671	13
15693	21
15796	102
15850	53
15878	27
15889	10
15909	19
15921	11
15935	13
15955	19
15987	31
16025	37
16067	41
16080	12
16109	28
16123	13
16152	28
16180	27
16225	44
16259	33
16319	59
16375	55
16403	27
16456	52

16474	17
16492	17
16505	12
16555	49
16601	45
16618	16
16636	17
16653	16
16685	31
16742	56
16758	15
16777	18
16809	31
16828	18
16840	11
16883	42
16897	13
16921	23
16968	46
17002	33
17026	23
17089	62
17145	55
17184	38
17211	26
17236	24
17252	15
17275	22
17300	24
17317	16
17331	13
17400	68
17452	51
17505	52
17518	12
17562	43
17586	23
17648	61
17709	60
17769	59
17796	26
17806	9
17823	16
17845	21
17870	24
17895	24
17918	22
17944	25

17956	11
17996	39
18037	40
18055	17
18082	26
18097	14
18119	21
18172	52
18217	44
18233	15
18267	33
18285	17
18337	51
18408	70
18452	43
18507	54
18547	39
18602	54
18623	20
18652	28
18669	16
18704	34
18730	25
18751	20
18778	26
18813	34
18837	23
18870	32
18890	19
18936	45
18980	43
19002	21
19030	27
19066	35
19084	17
19129	44
19196	66
19264	67
19281	16
19299	17
19324	24
19385	60
19431	45
19473	41
19516	42
19546	29
19559	12
19637	77

19671	33
19806	134
19835	28
19875	39
19914	38
20015	100
20041	25
20104	62
20129	24
20160	30
20248	87
20262	13
20304	41
20320	15
20449	128
20487	37
20528	40
20578	49
20652	73
20671	18
20785	113
20847	61
20859	11
20885	25
20901	15
20924	22
20984	59
21070	85
21116	45
21139	22
21152	12
21172	19
21198	25
21264	65
21357	92
21386	28
21416	29
21435	18
21494	58
21521	26
21543	21
21572	28
21619	46
21641	21
21694	52
21758	63
21790	31
21853	62

21900	46
21916	15
21970	53
22015	44
22037	21
22078	40
22119	40
22132	12
22206	73
22244	37
22258	13
22375	116
22392	16
22480	87
22502	21
22587	84
22609	21
22639	29
22674	34
22694	19
22713	18
22751	37
22776	24
22793	16
22821	27
22833	11
22844	10
22872	27
22956	83
22999	42
23051	51
23148	96
23267	118
23296	28
23391	94
23404	12
23434	29
23472	37
23576	103
23630	53
23644	13
23665	20
23680	14
23763	82
23830	66
23870	39
23889	18
23943	53

23961	17
23999	37
24016	16
24029	12
24092	62
24138	45
24154	15
24263	108
24311	47
24335	23
24381	45
24423	41
24443	19
24461	17
24495	33
24552	56
24592	39
24614	21
24661	46
24672	10
24743	70
24794	50
24832	37
24876	43
24898	21
24956	57
25007	50
25020	12
25043	22
25073	29
25097	23
25140	42
25191	50
25220	28
25255	34
25307	51
25340	32
25375	34
25395	19
25468	72
25529	60
25558	28
25582	23
25596	13
25613	16
25631	17
25655	23
25673	17

25717	43
25756	38
25820	63
25847	26
25979	131
25999	19
26035	35
26075	39
26142	66
26212	69
26259	46
26342	82
26384	41
26500	115
26615	114
26628	12
26681	52
26728	46
26753	24
26882	128
26954	71
26981	26
26994	12
27007	12
27075	67
27101	25
27134	32
27173	38
27245	71
27303	57
27433	129
27605	171
27691	85
27709	17
27727	17
27744	16
27814	69
27841	26
27863	21
27945	81
27961	15
28008	46
28061	52
28079	17
28090	10
28104	13
28166	61
28195	28

28212	16
28251	38
28262	10
28296	33
28330	33
28408	77
28427	18
28464	36
28570	105
28614	43
28751	136
28834	82
28924	89
28971	46
29061	89
29081	19
29103	21
29207	103
29228	20
29355	126
29382	26
29450	67
29525	74
29671	145
29822	150
29877	54
29966	88
30005	38
30027	21
30173	145
30269	95
30353	83
30433	79
30498	64
30513	14
30534	20
30553	18
30585	31
30609	23
30634	24
30739	104
30768	28
30804	35
30893	88
30981	87
31127	145
31203	75
31356	152

31425	68
31524	98
31543	18
31564	20
31661	96
31764	102
31832	67
31909	76
31928	18
32029	100
32048	18
32080	31
32172	91
32214	41
32229	14
32271	41
32418	146
32583	164
32608	24
32652	43
32691	38
32704	12
32748	43
32881	132
32995	113
33148	152
33248	99
33272	23
33328	55
33353	24
33510	156
33528	17
33713	184
33844	130
33867	22
33932	64
34074	141
34105	30
34127	21
34212	84
34282	69
34422	139
34551	128
34596	44
34658	61
34762	103
34826	63
34916	89

34992	75
35018	25
35158	139
35295	136
35344	48
35357	12
35457	99
35531	73
35631	99
35769	137
35866	96
35962	95
36104	141
36169	64
36243	73
36311	67
36373	61
36482	108
36613	130
36688	74
36713	24
36831	117
36904	72
36957	52
37008	50
37120	111
37169	48
37196	26
37236	39
37369	132
37451	81
37475	23
37503	27
37577	73
37646	68
37736	89
37795	58
37826	30
37870	43
37934	63
38011	76
38131	119
38182	50
38254	71
38312	57
38342	29
38368	25
38554	185

38805	250
38945	139
38999	53
39041	41
39086	44
39117	30
39143	25
39164	20
39201	36
39310	108
39420	109
39511	90
39667	155
39779	111
39821	41
39985	163
40011	25
40164	152
40350	185
40459	108
40632	172
40688	55
40884	195
40906	21
40925	18
41040	114
41074	33
41128	53
41209	80
41418	208
41543	124
41704	160
41733	28
41800	66
41881	80
41934	52
41957	22
42021	63
42047	25
42071	23
42132	60
42235	102
42438	202
42485	46
42596	110
42641	44
42755	113
42916	160

43058	141
43236	177
43305	68
43409	103
43423	13
43476	52
43576	99
43630	53
43854	223
43954	99
43985	30
44038	52
44104	65
44277	172
44321	43
44429	107
44682	252
44707	24
44730	22
44751	20
44890	138
45030	139
45112	81
45132	19
45261	128
45493	231
45512	18
45727	214
45899	171
45945	45
46148	202
46354	205
46400	45
46501	100
46515	13
46662	146
46807	144
46940	132
47124	183
47267	142
47369	101
47560	190
47612	51
47641	28
47734	92
47887	152
47937	49
48083	145

48234	150
48398	163
48695	296
48904	208
49053	148
49261	207
49325	63
49667	341
49784	116
49876	91
49962	85
50128	165
50210	81
50324	113
50442	117
50470	27
50661	190
50781	119
50864	82
50918	53
51143	224
51335	191
51365	29
51513	147
51796	282
51929	132
52194	264
52287	92
52334	46
52708	373
52973	264
53196	222
53429	232
53830	400
53982	151
54030	47
54383	352
54687	303
54753	65
55179	425
55250	70
55415	164
55647	231
55855	207
56061	205
56286	224
56606	319
56912	305

57413	500
57694	280
57932	237
58389	456
58582	192
58646	63
58920	273
59338	417
59768	429
60269	500
60334	64
60693	358
61062	368
61144	81
61279	134
61349	69
61620	270
62121	500
62228	106
62727	498
63085	357
63360	274
63575	214
63857	281
64358	500
64516	157
64880	363
64917	36
65092	174
65536	443
65850	313
65914	63
66415	500
66757	341
67006	248
67164	157
67391	226
67644	252
67830	185
68094	263
68295	200
68526	230
68727	200
68891	163
69086	194
69334	247
69556	221
69786	229

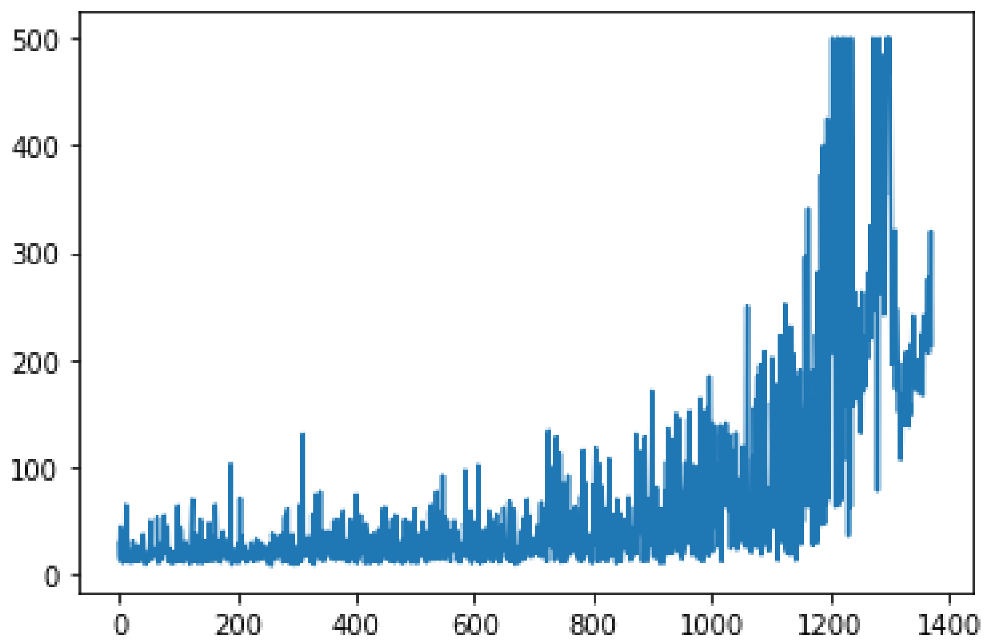
69919	132
70101	181
70365	263
70536	170
70720	183
70903	182
71109	205
71310	200
71486	175
71676	189
71900	223
72102	201
72384	281
72635	250
72912	276
73144	231
73470	325
73691	220
73953	261
74250	296
74496	245
74997	500
75267	269
75596	328
75887	290
76289	401
76715	425
77184	468
77263	78
77661	397
78066	404
78567	500
78873	305
79249	375
79735	485
80133	397
80395	261
80722	326
81088	365
81331	242
81641	309
81998	356
82354	355
82830	475
83331	500
83832	500
84301	468
84802	500

85303	500
85804	500
86221	416
86544	322
86741	196
86947	205
87186	238
87387	200
87674	286
87899	224
88221	321
88395	173
88644	248
88871	226
89070	198
89262	191
89414	151
89612	197
89778	165
89886	107
90012	125
90181	168
90321	139
90484	162
90631	146
90771	139
90932	160
91089	156
91291	201
91500	208
91666	165
91868	201
92021	152
92160	138
92319	158
92468	148
92619	150
92777	157
92993	215
93184	190
93426	241
93607	180
93815	207
93991	175
94172	180
94374	201
94548	173
94719	170

```
94890 170
95070 179
95245 174
95426 180
95615 188
95792 176
95993 200
96218 224
96386 167
96605 218
96847 241
97063 215
97282 218
97491 208
97718 226
97935 216
98165 229
98442 276
98649 206
98881 231
99143 261
99393 249
99714 320
99929 214
```

[5]:
```python
from matplotlib import pyplot as plt
plt.plot(episode_length_set)
```

[5]:
```
[<matplotlib.lines.Line2D at 0x639194a10>]
```



[6]:
```
# test our network
tf.reset_default_graph()
RL = DQN(actions_num = 2, gamma = 1,
         state_size = 4, epsilon_start = 1,
         learning_rate = 1e-3, epsilon_min = 0,
         replace_target_iter = 100, memory_size = 5000,
         epsilon_increment = None,)
# load saved parameters
RL.restore()
# run 100 trails and print how long can the agent hold the cart pole for each␣
 ↪trail
for i in range(100):
  ############################
  # YOUR CODE STARTS HERE

  RL.epsilon = 0
  new_state = env.reset()
  done = False
  episode_length_counter = 0
  for step in range(500):
    if(done):
      print(step, episode_length_counter)
      break
    action = RL.choose_action(new_state)
    new_state, reward, done, info = env.step(action)
    episode_length_counter += 1
```

```
    if(episode_length_counter == 500):
      print(step, episode_length_counter)

  # YOUR CODE ENDS HERE
  ############################
```

```
INFO:tensorflow:Restoring parameters from ./weights/DQN_model.ckpt
499 500
499 500
499 500
491 491
499 500
424 424
499 500
499 500
499 500
499 500
499 500
499 500
436 436
499 500
499 500
499 500
499 500
416 416
499 500
499 500
370 370
424 424
499 500
499 500
499 500
447 447
499 500
499 500
499 500
499 500
394 394
374 374
499 500
499 500
435 435
499 500
499 500
499 500
499 500
```

499	500
499	500
457	457
499	500
499	500
499	500
499	500
411	411
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
446	446
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
475	475
499	500
499	500
420	420
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500
499	500

```
499 500
499 500
499 500
465 465
499 500
499 500
499 500
499 500
382 382
483 483
499 500
432 432
499 500
```

You may find that the episode length doesn't stably improve as more training time is given. You can read chapter 3.2 of this paper https://arxiv.org/pdf/1711.07478.pdf if you are interested.