Introduction to Computer Networks

Assignment 3: Deep Q-Networks

소프트웨어학과 김승현

2014310279

1. Development environments

- **Mac OS X (10.15.1)**
- **Python 3.7.5**
- **Tensorflow 2.0.0**
- **Keras 2.3.0**
- **Pip 19.3.1**

2. Design

a. Parameter

Discount rate	Replay memory	Learning rate	Learning starts	Optimizer
0.99	50000	0.001	1000	RMSprop

network hidden_size	batch_size	target_ replace_iter	multi_step	Learning _step
32	128	100	3	20

Batch_size, learning_stpe등은 batch sample이 골고루 학습 될 수 있도록 고려하여 설정하였다.

b. _build_network()

```
def _build_network(self):
    # Local 네트워크 및 target 네트워크를 설정
    model = models.Sequential()
    model.add(layers.Dense(self.hidden_size, activation='relu', input_dim= (self.state_size) ))
    model.add(layers.Dense(self.hidden_size, activation='relu'))
    model.add(layers.Dense(self.action_size, activation='linear'))
    model.compile(loss='mse', optimizer=keras.optimizers.RMSprop(lr=LEARNING_RATE))
    return model
```

Keras model을 이용하여 network를 구성하였다. Activationd은 hidden 레이어의 경우 relu, output 레이어는 linear를 사용하였고, optimizer는 RMSprop을 사용하였다.

c. _learn()

```
# episode 시작
while not done:
    action = self.predict(state)
    next_state, reward, done, _ = self.env.step(action)
    tmp_reward = reward
    if next_state[0] >= 0.5:
        tmp_reward = 10
    if next_state[0] > -0.4 :
        tmp_reward = -0.1 + next_state[0]
    if step_count % render_count == 0:
        self.env.render()
```

초반에 빠르게 목표에 도달할수 있도록 reward shapeing을 해줬다. 목표에 도달하지 못하고 얻는 reward가 지나치게 큰 경우 학습이 잘 되지 않기 때문에, 목표 도달에 도움을 주면서도 학습 자체를 방해하지 않도록 알맞은 reward를 설정해 주었다. 최종 결과에는 shaping한 reward가 들어가지 않도록 train시에만 tmp_reward를 사용한다.

```
if episode % 1 == 0:
    if len(self.memory) > LEARNING_STARTS:
        for _ in range(self.learning_step):
            mini_batch = random.sample(self.memory, self.batch_size)
            self.train_minibatch(mini_batch)
            self.memory_counter +=1
    if self.memory_counter >= self.target_replace_iter:
        self.memory_counter = 0
        self.target_network.set_weights(self.local_network.get_weights())
```

Replay memory가 충분히 차면 memry에서 batch사이즈만큼 mini_batch를 랜덤 샘플링 후 learning_step 만큼 학습을 진행 한다. 또한 target_replace_iter에 맞춰 target_network를 업데이트 한다.

d. _train_minibatch()

```
def train_minibatch(self, mini_batch):
    # mini batch를 받아 policy를 update
    samples = np.array(mini_batch)
    state, action, reward, next_state, done = np.hsplit(samples, 5)
    sampe_states = np.concatenate((np.squeeze(state[:])), axis = 0)
    sampe_states = np.reshape(sampe_states, (self.batch_size, 2))
    sample_rewards = reward.reshape(self.batch_size,).astype(float)
    Q = self.local_network.predict(sampe_states)
    sampe_nstates = np.concatenate((np.squeeze(next_state[:])), axis = 0)
    sampe_nstates = np.reshape(sampe_nstates, (self.batch_size, 2))
    dones = np.concatenate(done).astype(bool)
    not_dones = (dones^1).astype(float)
    dones = dones.astype(float)
    if self.double_q == True:
        target_next_q = self.target_network.predict(sampe_nstates)
        double_action = np.argmax(self.local_network.predict(sampe_nstates), axis=1)
        next_q = target_next_q[(np.arange(self.batch_size), double_action)]
        Q[(np.arange(self.batch_size), action.reshape(self.batch_size,).astype(int))] =
    else:
        next_q = self.target_network.predict(sampe_nstates).max(axis = 1)
        Q[(np.arange(self.batch_size), action.reshape(self.batch_size,).astype(int))] =
    self.local_network.fit(sampe_states, Q, epochs=1, verbose=0)
```

빠른 학습을 위하여 minibatch내에서 for문으로 하나씩 학습시키지 않고, numpy를 활용하여 한꺼번에 학습이 가능하도록 preprocessing 해 주었다.

3. 추가 구현

1) Double DQN

```
if self.double_q == True:
    target_next_states = self.target_network.predict(next_states)
    double_action = np.argmax(self.local_network.predict(next_states), axis=1)
    next_q = target_next_states[(np.arange(self.batch_size), double_action)]
    targets[(np.arange(self.batch_size), action.reshape(self.batch_size,).astype(int))] = rewards + dones + (rewards + next_q * DISCOUNT_RATE)*notdones
```

Mini_batch를 트레이닝 할 때, next_q를 구하는 방법만 살짝 바꿔 DDQN을 구현하였다.

2) Multistep DQN

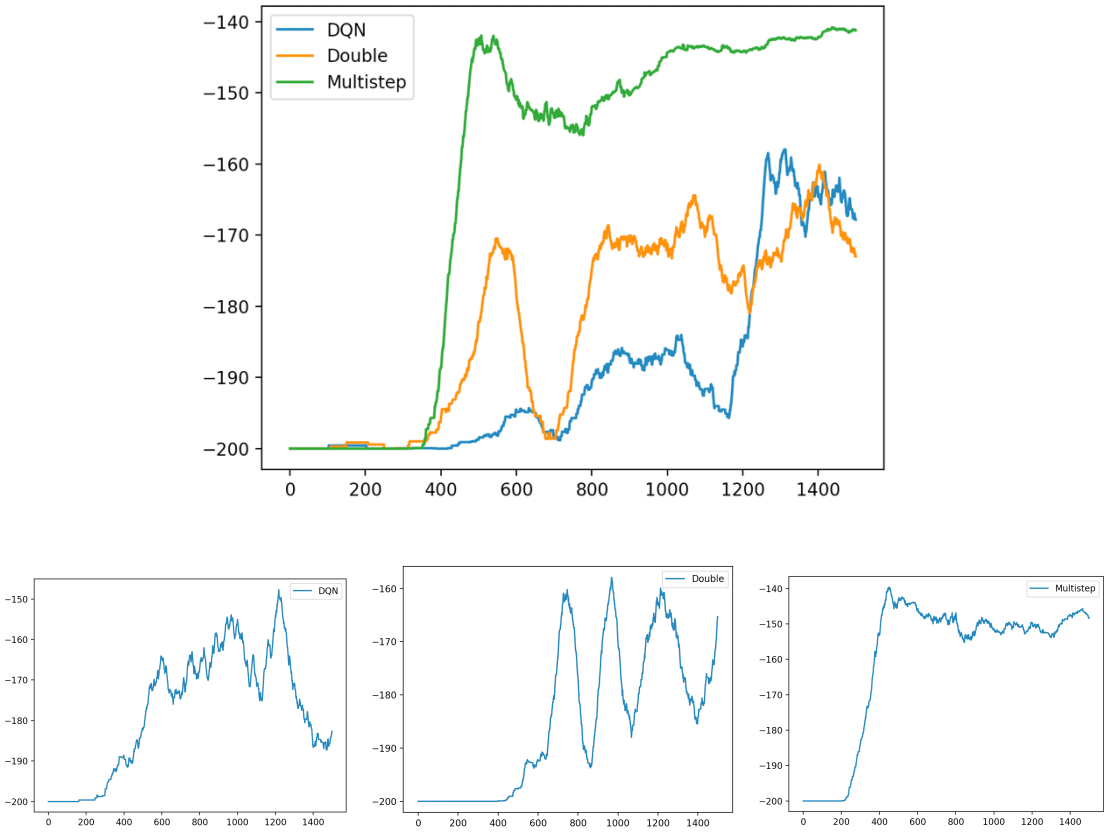
```
if self.multistep == True:
    nstep = self.n_steps
    len_tmpmem = len(multi_state)
    if done:
        for i in range(len_tmpmem):
            if i == (len_tmpmem-nstep):
                nstep -= 1
            for j in range(nstep-1):
                multi_reward[i] += multi_reward[i+j+1] * (DISCOUNT_RATE ** (j+1))
            multi_next_state[i] = multi_state[i+nstep]
            self.memory.append((multi_state[i], multi_action[i], multi_reward[i], multi_next_state[i], multi_done[i]))
    else :
        multi_state.append(state)
        multi_action .append(action)
        multi_reward.append(reward)
        multi_next_state.append(next_state)
        multi_done.append(done)
```

Train 함수를 따로 건드리지 않고, reward와 next_state를 step수에 맞춰서 조정하도록 데이터를 처리하였다. Train 시에는 업데이트된 reward와 next_state를 기준으로 값을 계산한다.

3) PER DQN

구현하지 못하였다.

4. Experimentation results



Multi-step의 경우 첫 성공 이후에는 급격하게 성공률이 증가하는 모습을 보이지만, 첫 성공 타이밍이 늦는 경우가 가끔 있었으며, DDQN과 DQN은 늦어도 400번 이전에는 성공하지만 성공 이후에도 성공률이 확 좋아지지 않아서 지속적으로 성공하지 못해 결과 값이 잘 나오지 않는 것 같다.