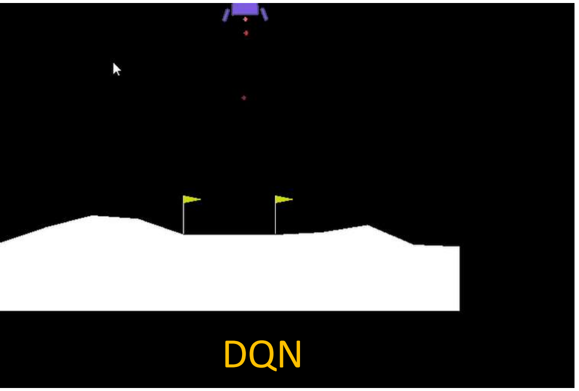
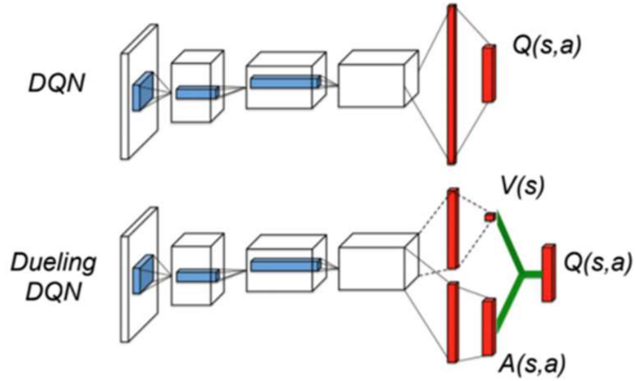
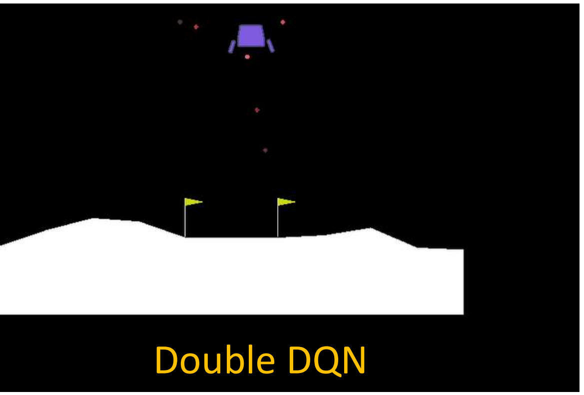


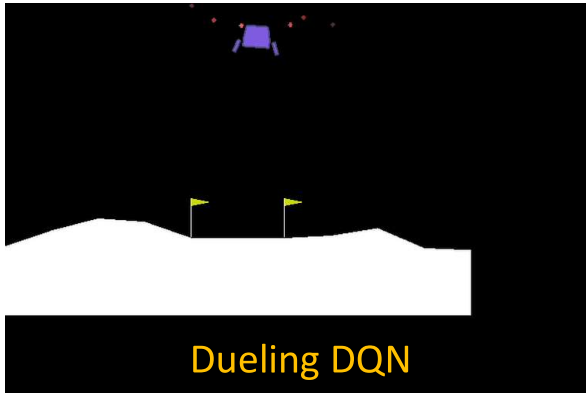


軟式計算期末專題

三種DQN強化學習方式在Lunar Lander上的PyTorch應用比較

Albert Wu,　June 2020





DQN
(Nature 2015)

Two Q networks are used:
local (original 2013) &
target (updated every C steps)

Batch size = 64
C = 4
in this presentation

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \boxed{\theta^-}\right) & \text{otherwise} \end{cases}$

Target Q-network

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \boxed{\theta}\right)\right)^2$ with respect to the network parameters θ

Q-network (local)

    Every C steps reset $\hat{Q} = Q$

Neural Network in DQN for Lunar Lander

Input

8 real values:

State

horizontal coordinate: **s[0]**
vertical coordinate: **s[1]**
horizontal speed: **s[2]**
vertical speed: **s[3]**
angle: **s[4]**
angular speed: **s[5]**
Indicator(1st leg has contact): **s[6]**
Indicator(2nd leg has contact): **s[7]**

```
Indicator(event) = 1, if event is true
                 = 0, otherwise
```

```
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size,
seed, fc1_units=64, fc2_units=64):
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

Neural Network in PyTorch

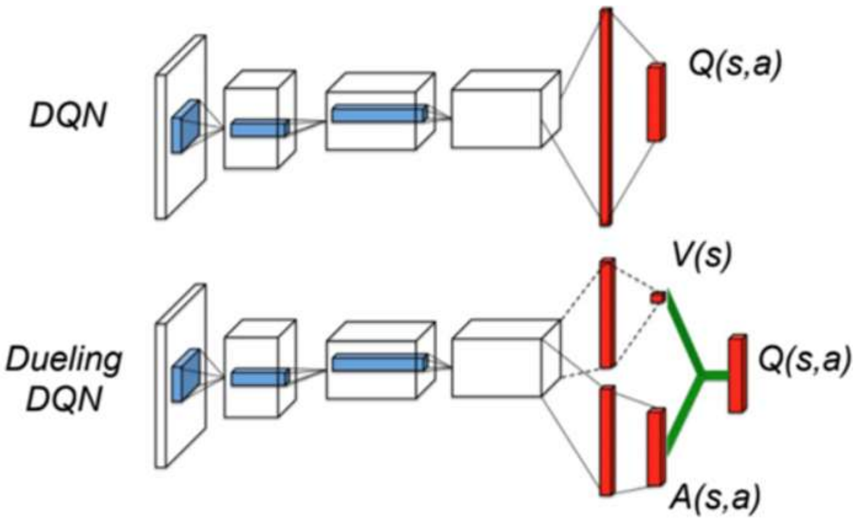
Output

4 real values,
each for one of 4 Actions

Action

- 0 - Do nothing
- 1 - Fire left engine
- 2 - Fire main engine
- 3 - Fire right engine

Neural Networks




```python
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size, seed, fc1_units=64, fc2_units=64):
        super(QNetwork, self).__init__()
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.fc3 = nn.Linear(fc2_units, action_size)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

DQN

Double DQN

```python
class QNetwork(nn.Module):
    def __init__(self, state_size, action_size, seed, fc1_units=64, fc2_units=64):
        super(QNetwork, self).__init__()
        self.action_size = action_size
        self.seed = torch.manual_seed(seed)
        self.fc1 = nn.Linear(state_size, fc1_units)
        self.fc2 = nn.Linear(fc1_units, fc2_units)
        self.adv = nn.Linear(fc2_units, action_size)
        self.val = nn.Linear(fc2_units, 1)

    def forward(self, state):
        x = F.relu(self.fc1(state))
        x = F.relu(self.fc2(x))
        adv = self.adv(x)
        val = self.val(x).expand(x.size(0), self.action_size)

        x = val + adv - adv.mean(1).unsqueeze(1).expand(x.size(0), self.action_size)
        return x
```

A: advantage

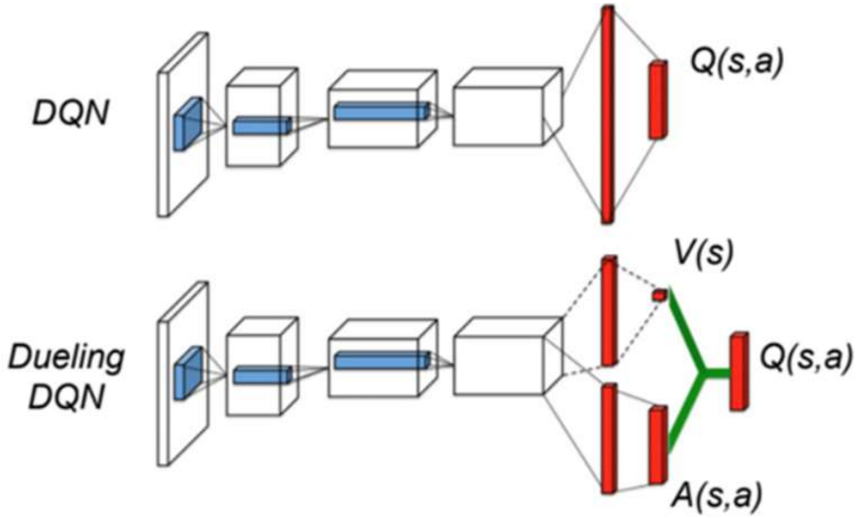
V: value

Dueling DQN

Dueling DQN

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

Deep Q-Learning



Q_targets

$$Y_t^{\mathrm{DQN}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t^-)$$

```
def learn(self, experiences, gamma):

    states, actions, rewards, next_states, dones = experiences

    Q_expected = self.qnetwork_local(states).gather(1, actions)

    Q_targets_next = self.qnetwork_target(next_states).detach().max(1)[0].unsqueeze(1)
    Q_targets = rewards + (gamma * Q_targets_next * (1 - dones))

    loss = F.mse_loss(Q_expected, Q_targets)
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

DQN

Dueling DQN

Double DQN uses different **learning** algorithm than DQN and dueling DQN

```
def learn(self, experiences, gamma):

    states, actions, rewards, next_states, dones = experiences

    Q_expected = self.qnetwork_local(states).gather(1, actions)

    actions_value = self.qnetwork_local.forward(next_states)
    next_action = torch.unsqueeze(torch.max(actions_value, 1)[1], 1)
    next_q = self.qnetwork_target.forward(next_states).gather(1, next_action)
    Q_targets = rewards + GAMMA * next_q * (1-dones)

    loss = F.mse_loss(Q_expected, Q_targets)
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

Double DQN

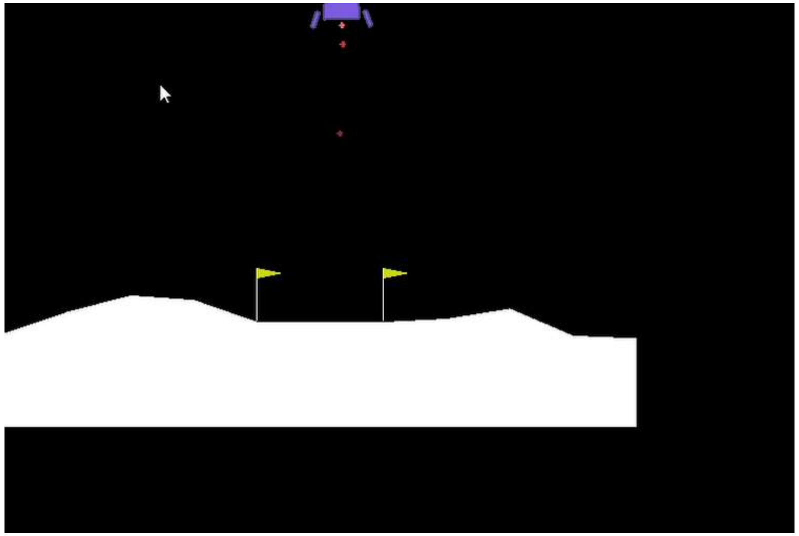
$$\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$$

$$Y_t^{\mathrm{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \boxed{\operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)}; \boldsymbol{\theta}'_t)$$

Q_targets

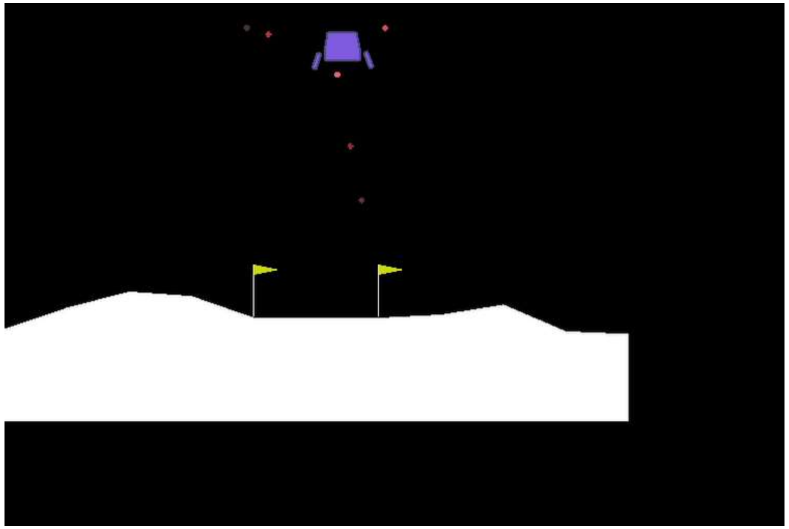
Lunar Lander測試

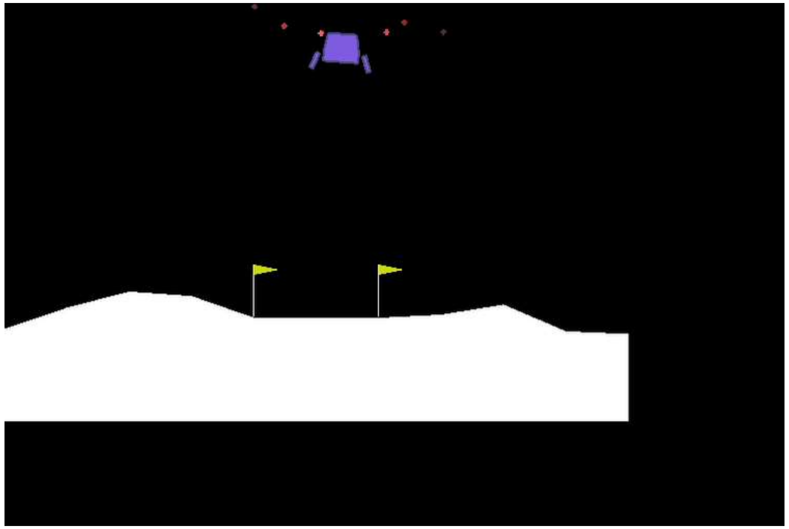
訓練1500回合
歷經1.5小時以後



DQN

Double DQN





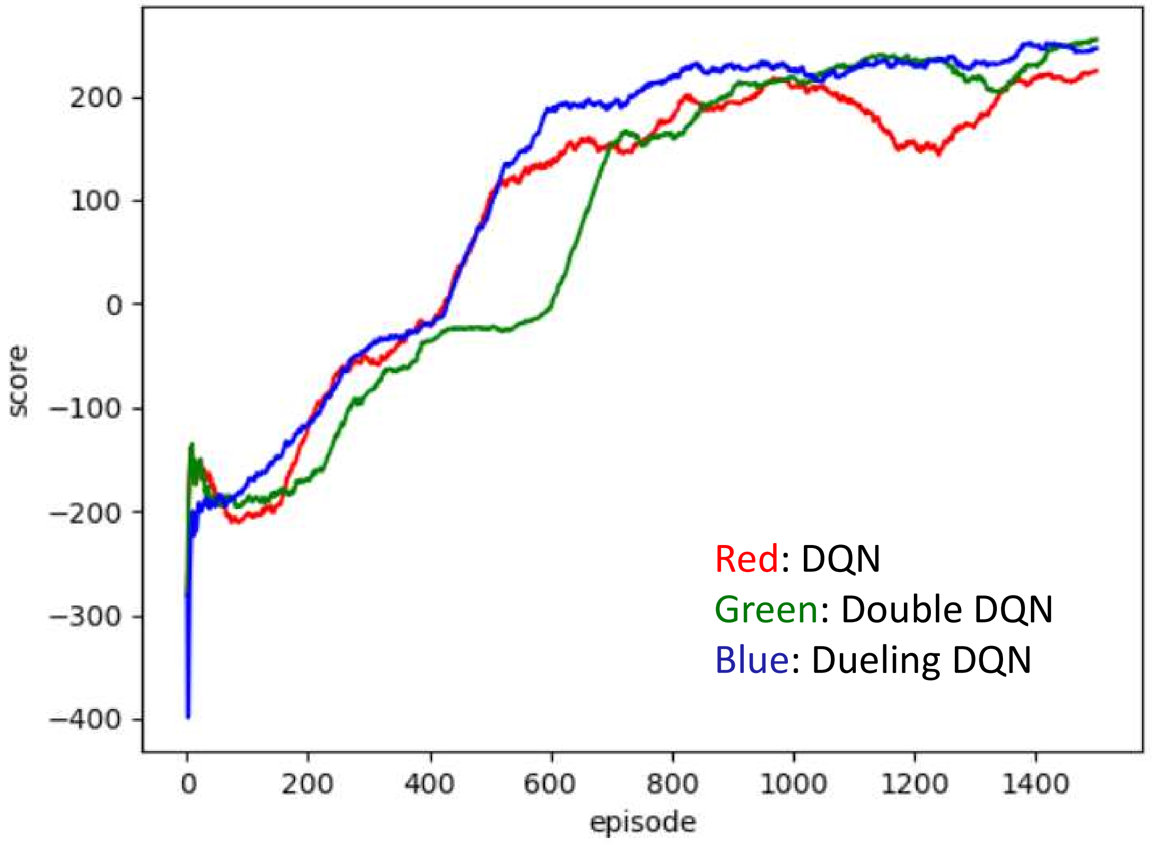
Dueling DQN

Conclusions: Average Score vs #Episodes

1. Dueling DQN has larger scores and faster convergence.

2. Double DQN has smaller scores initially with (improved) less over-estimates.

3. Both double and dueling DQN's are more robust than DQN.



References

1. V. Mnih, etc., “Playing Atari with Deep Reinforcement Learning”, arXiv:1312.5602, 2013.
2. V. Mnih, etc., “Human-level control through deep reinforcement learning”, Nature, 2015.
3. H.V. Hasselt, A. Guez, D. Silver, “Deep reinforcement learning with double Q-learning”, arXiv:1509.06461, 2015.
4. Z. Wang, etc., “Dueling network architectures for deep reinforcement learning”, arXiv:1511.06581, 2015.
5. https://github.com/udacity/deep-reinforcement-learning/tree/master/dqn/solution
6. 高揚, 葉振斌, 強化學習**(RL)**：使用**PyTorch**徹底精通, 深智數位, 2020.
7. 小川雄太郎, 實戰人工智慧之深度強化學習：使用**PyTorch x Python**, 碁峰, 2019.
8. S. Ravichandiran, 用**Python**實作強化學習：使用**TensorFlow**與**OpenAI Gym**,碁峰, 2019.