Deep-Q Network Learning

Why Deep ? ⇒

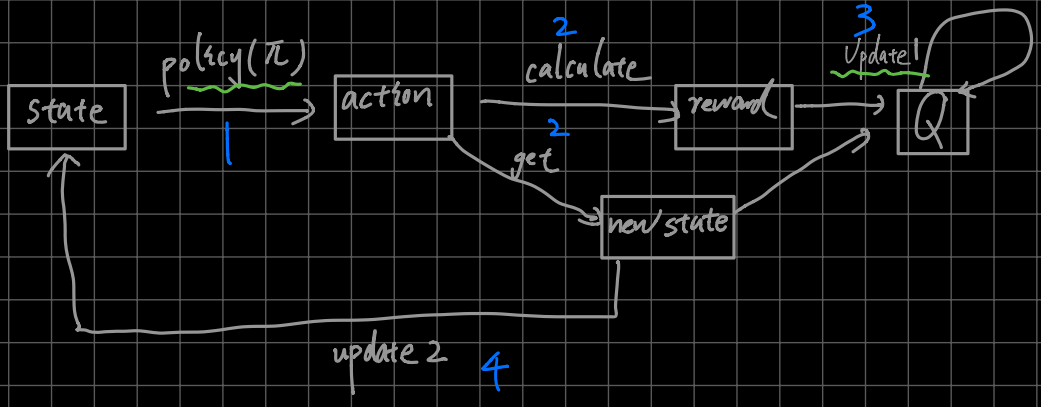
当 Action Space, state space 大到一定规模之后, Q-Learning 使用的 Q-table 方法便不能应对 (无法得到一张具有如此之大储存空间的 Q-table.)

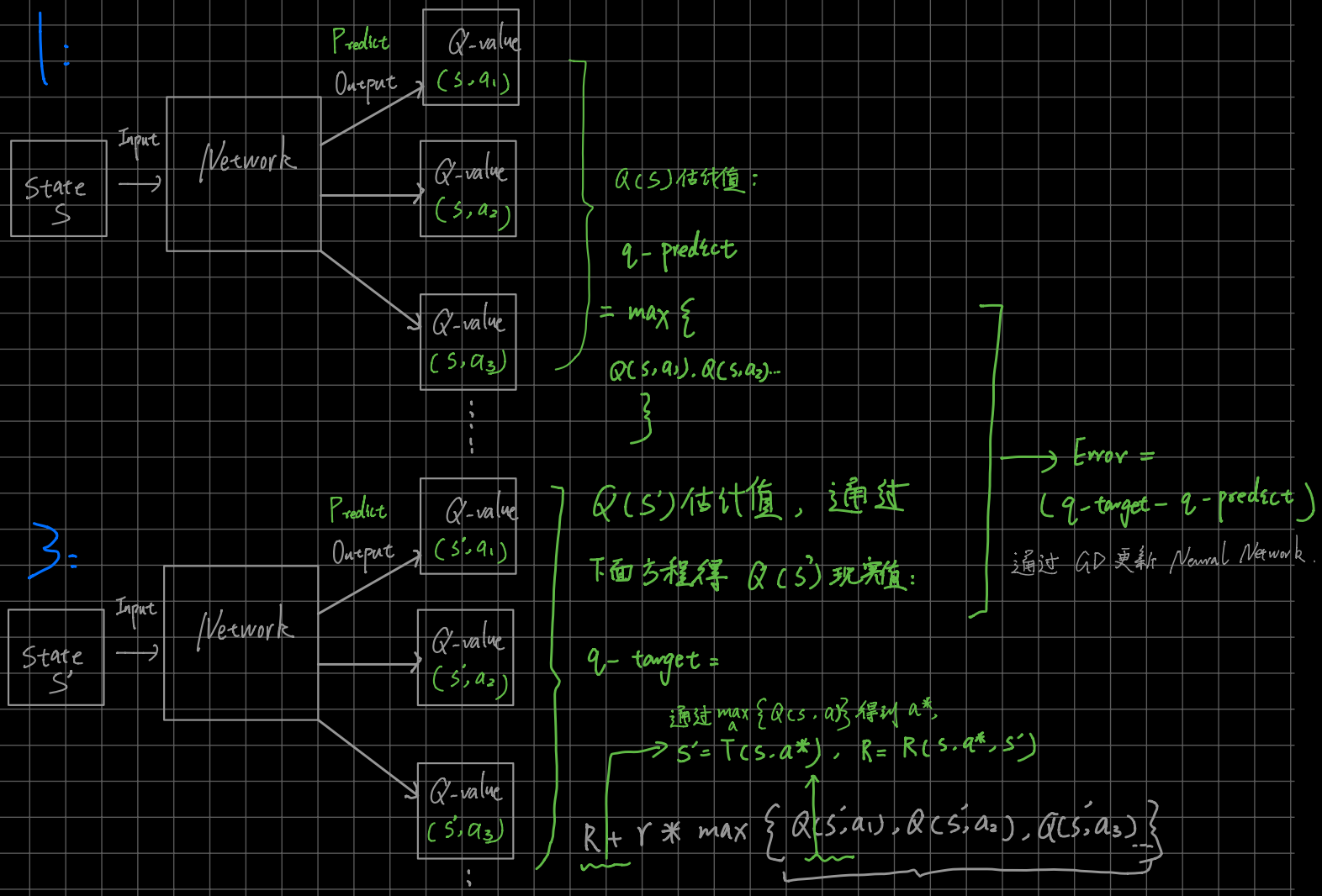
Solve: Deep Neural Network. ① 构建 NN, 使得其输入为 s, 输出为 A, 可以完成 choose_action

② 构建 NN, 使得其输入为 S、A, 输出为 Q, 可以完成 Policy_update.

How ? ⇒

DQN 仍采用与 Q-Learning 相同的流程: (除了上述涉及 Q-table 的二部分由 NN 替代之外)





Key points ? ⟹ Experience Replay + Fixed Q-targets
(DQN 两大利器)

Experience Replay (经验回放):

作用: 解决 Sample 特性 (High correlated and non-stationary.) 带来的难收敛问题。

实质: 将 online → semi-offline 用 memory 储样本, 通过随机采样去除相关性.

Fixed Q-target

作用: Seperate Target Network from Q-network. 使训练震荡发散可能性降低, 更加稳定.

震荡发散: 若 y 使用同一个 Q-network, 则使 Q 大的样本, y 也会大, 则震荡发散可能性增大.

⟹ 用估计值去逼近一个不断更新的 target label, 训练是肯定不稳定的. 在 RL 中, 训练时 label 是已知的, 因此是让估计值朝着固定的目标更新优化的. 而 RL 没有 label, 因此我们训练时把"假定的" label 固定不变, 让估计值去逼近它. 这就是 Fixed Q-target 思想来源.

DQN Algorithm :

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ ⇒ eval network

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$ ⇒ target network

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For t = 1,T **do**

① 引入原因：增加模型探索能力

With probability ε select a random action a_t action selection

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$ ⇒ ε-greedy policy

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 $\phi_{t+1} = \phi(s_{t+1})$

② Experience Replay

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D samples

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

③ target Network ⇒ 通过 Target Network 选择样本标签

方法：SGD

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ 更新Q-network权重

模型优化的目标

Every C steps reset $\hat{Q} = Q$ 每隔C步更新target network

③

End For

End For

1: Initialization
- Replay Memory (D): capacity N
- Eval Network (Q): random weights θ
- Target Network ($\hat{Q}$): $\theta^- = \theta$

2: For episode in EPISODE:

$s_1 = \{x_1\}$

$\phi_1 = \phi(s_1)$

step_counter = 0

3: while not Is_terminated:

4: a_t = choose_action(s_t) ⟶ ε-greedy:

$$a_t = \begin{cases} \text{np.random.choice(ACTION)} \\ \arg\max_a Q(\phi(s_t), a; \theta) \end{cases}$$

5: $r_t, x_{t+1} = \text{get\_env\_feedback}(s_t, a_t)$

↳ reward bias ⇒ Dueling Network

6: $s_{t+1} = s_t$

$\phi_{t+1} = \phi(s_{t+1})$

7: Experience Replay

- Store $(\phi_t, a_t, r_t, \phi_{t+1})$ (transition) in D
- Sample transition batches from D ⇓ randomly

↳ ⇒ prioritized replay

8: Policy update:

$$y_t = \begin{cases} r_j & \Rightarrow s_{t+1} = \text{"terminated"} \\ r_t + \gamma \cdot \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{, otherwise} \end{cases}$$

(ϕ_{j+1}: next state)

↳ upward bias ⇒ Double Q-Network

9: Gradient descent:

$$\text{error} = \underbrace{\left(y_t - Q(\phi_t, a_t; \theta)\right)^2}_{\text{q-predict}}$$

(y_t: q-target)

$$\theta = \theta - \alpha \frac{\partial \text{error}}{\partial \theta}$$

10: step_counter += 1

if step_counter == C:

$\theta^- = \theta \longrightarrow \hat{Q} = Q$

Advanced DQN

DQN三大改进

- Double Q-Network: solve upward-bias problem
- Prioritized Experience Replay: solve random choose mini-batch problem
  ① 变相增加样本 ② 独立于训练过程中状态的影响
- Dueling Network: solve reward bias problem

$Q(s,a) \rightarrow V(s)$ (标量)

$Q(s,a) \rightarrow A(s,a)$: Advantage (矢量)

Double DQN:

Problem:

$$q\text{-target}_t = R_{t+1} + \gamma \cdot \max_a \hat{Q}(S_{t+1}, a; \theta_t^-)$$

$\hat{Q}$ ⇓ target_net

Upward-bias: 由于 $\hat{Q}(S_{t+1}, a; \theta_t^-)$ 本身就有误差，而估计 q-target 时采用 max policy，因此每次向着最大误差的方向改进神经网络，导致 Overestimate.

Solve ⇒

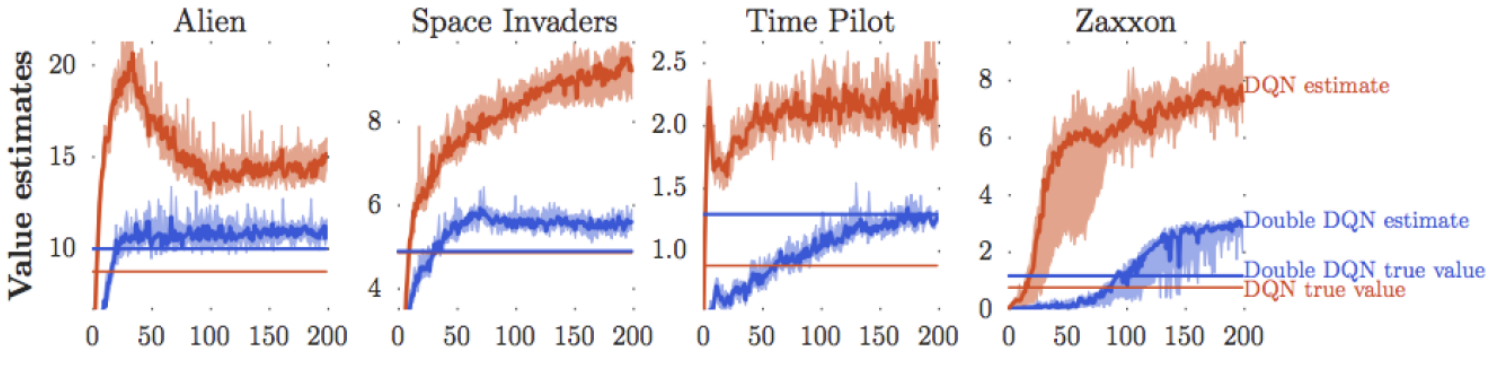
$$q\text{-target}_t = R_{t+1} + \gamma \cdot \hat{Q}(S_{t+1}, \underbrace{\arg\max_a Q(S_{t+1}, a; \theta_t)}; \theta_t^-)$$

⇓

通过 eval_net 估计 action values，通过 np.argmax(action values, axis=1) 选出 action index，再通过 target_net 估计 q-target

Function: Solve upward bias problem，使得 action values 的估计值下降，接近中心。

- Q value is usually over-estimated



Dueling DQN:

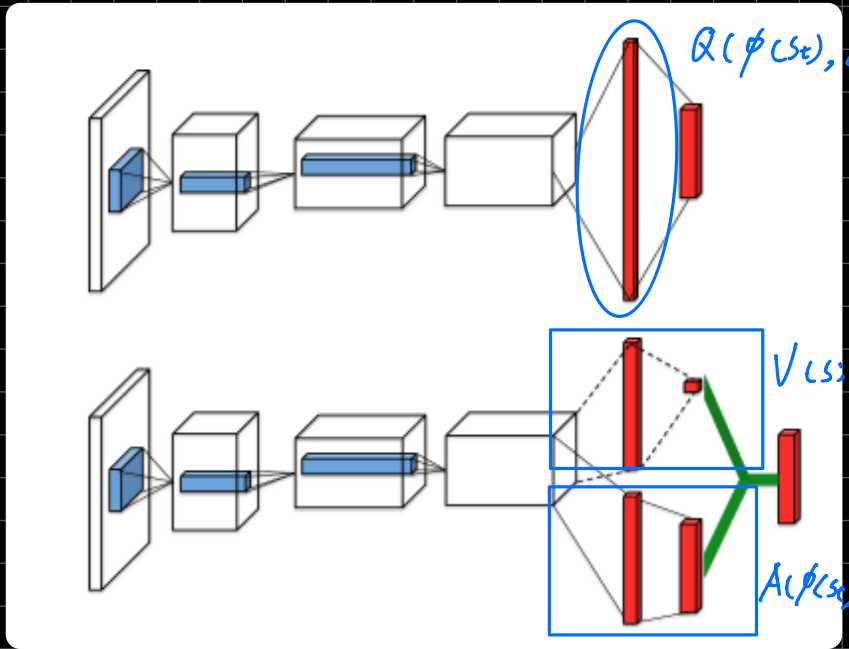
Problem: Reward bias. 在游戏中，存在很多对action不敏感的状态（S → S'，不受采取action的影响），在这些状态下计算 action value func 意义 没有 state value func 大。因此若 state value >> action value，此时Q value取决于 state value

要使用Q value来选取best action，需去除state value的影响，这样才能体现action value的作用。因此采用Dueling network.

Solve ⇒

Original:

Dueling:



Original DQN: $Q = \underbrace{Q}_{\text{eval-net}}(\phi(s_t), a; \theta)$, predict the value of each action, and choose the max Q value action

Dueling DQN: $\underbrace{Q(s_t, a; \theta, \alpha, \beta)}_{\text{predicted } Q \text{ value}} = \underbrace{V(s; \theta, \beta)}_{\text{Value}} + \underbrace{A(s, a; \theta, \alpha)}_{\text{advantage of: action from current state}}$

⇒ 实际应用中，采取 average 操作，使 advantage 更加稳定，便于优化。

$$Q(s_t, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \underbrace{\left[A(s, a; \theta, \alpha) - \frac{1}{|A|} \cdot \sum_{a'} A(s, a'; \theta, \alpha)\right]}_{\Downarrow}$$

self.V + self.A - np.reduce_mean(self.A, axis=1, keep_dims=True)

Function: ① 提高 DQN 收敛速度

② 降低 model variance，使得 model 泛化性能更好，更稳定。