Deep Reinforcement Learning With Double Q-Learning

Hado van Hasselt
Arthur Guez
David Silver

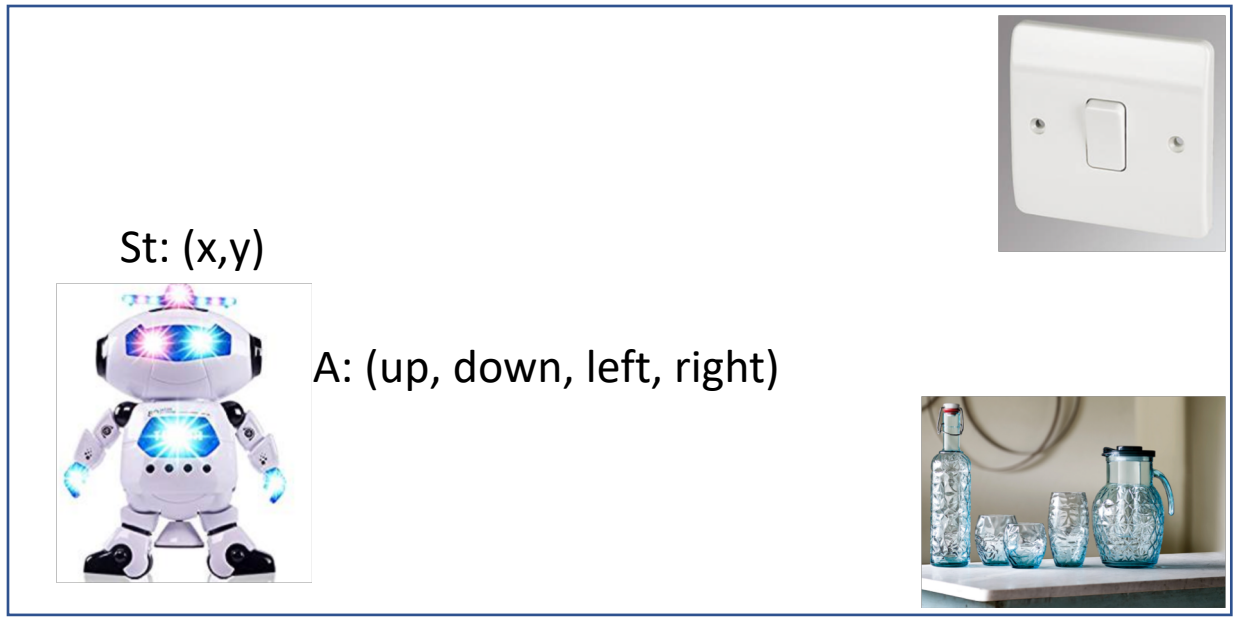
Presented By: Sachin Rana
Machine Learning Engineer @ Dessa

Highlights

- The popular Q-learning algorithm overestimate action values
- Double Q-learning reduces overestimations and can work on large-scale problems

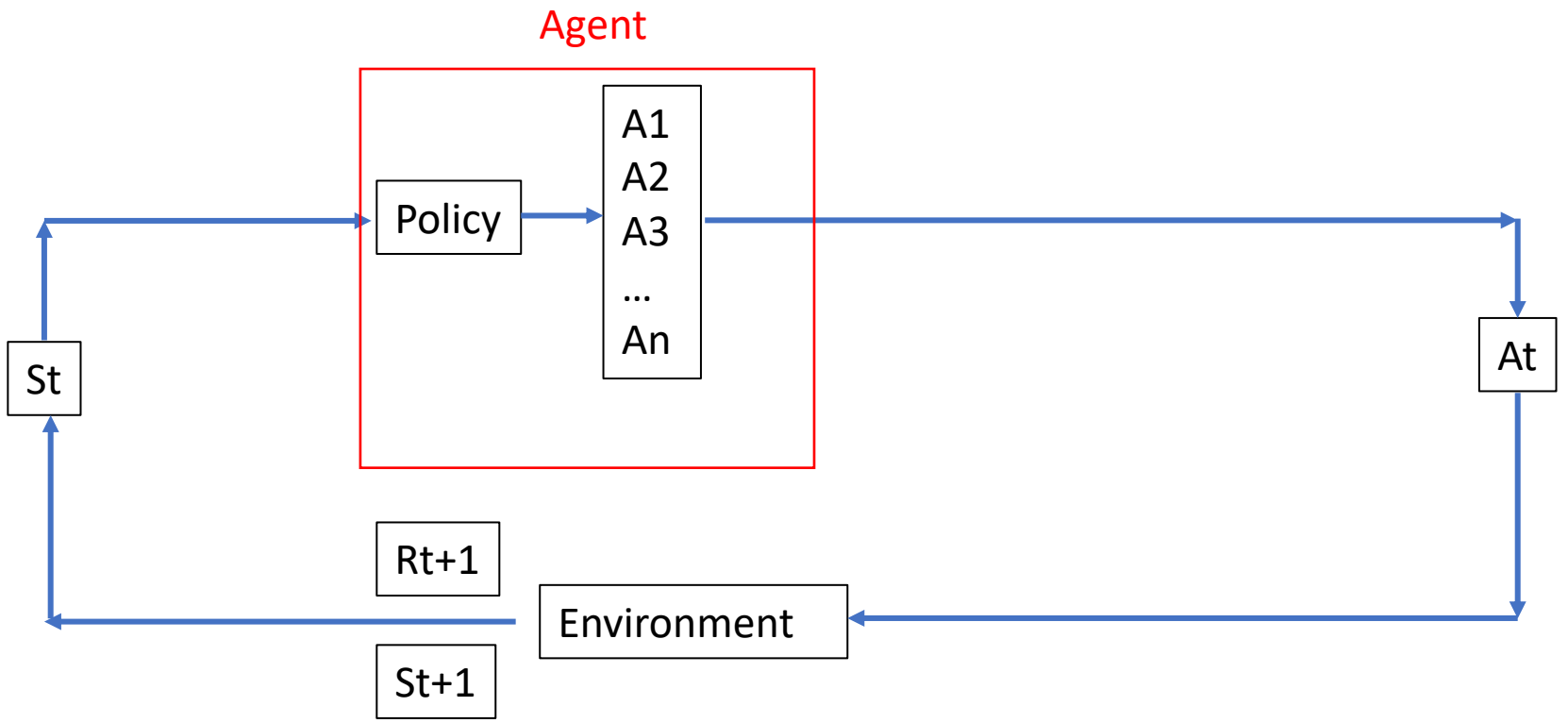
Reinforcement Learning Basics

- State - a representation/ features of environment
- Action – a discrete action to take at any given time
- Immediate Reward – result of taking any action
  $(S_t, A_t) \rightarrow Env \rightarrow R_{t+1}, S_{t+1}$
- Total Future Reward (Q-value):
  - $Q_t = R_{t+1} + R_{t+2} + \cdots R_{t+T}$
  - $Q_t = R_{t+1} + Q_{t+1}$
- Discounted Future Reward:
  - $Q_t = R_{t+1} + \gamma . Q_{t+1}$



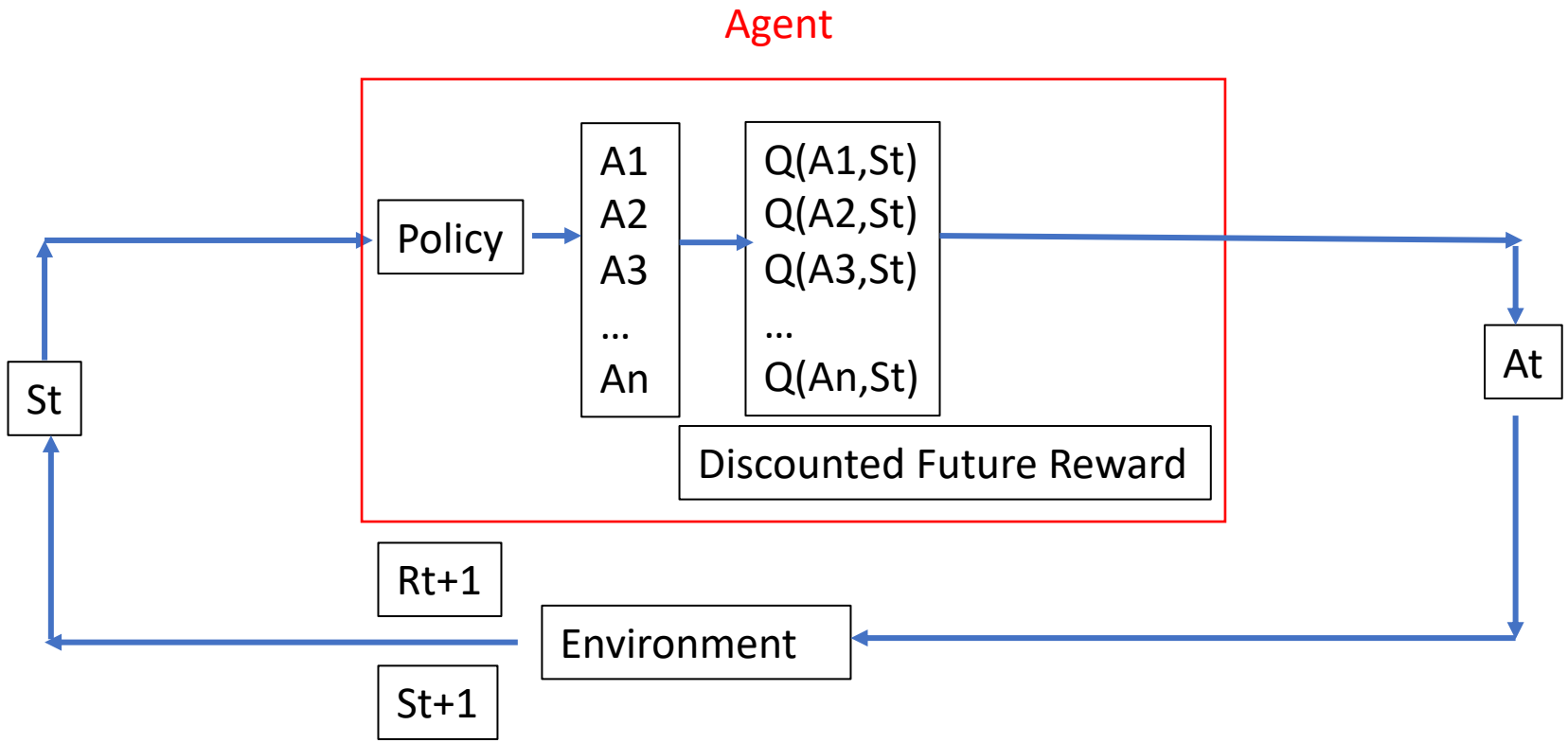
[Reinforcement Learning Video Link]

Reinforcement Learning (What is Policy?)



- Policy: what action to take given any state?
- Goal: learn the best policy to take the best action -> that gives the maximum future discounted reward.

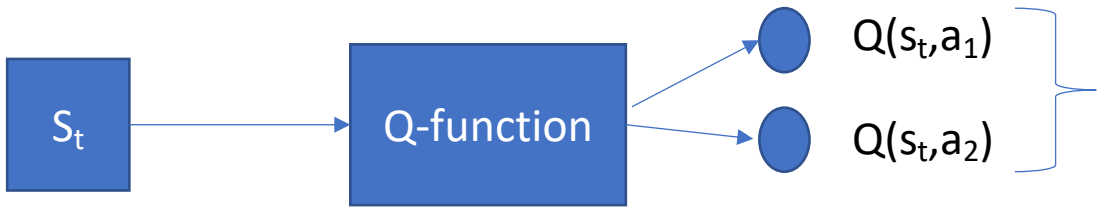
Reinforcement Learning (What is Q-learning?)



- Q-values are the expected future rewards
- Following greedy policy: the agent takes the actions which have the highest Q-value

Universal Q-Learning

- We can approximate Q-values using an arbitrary function:

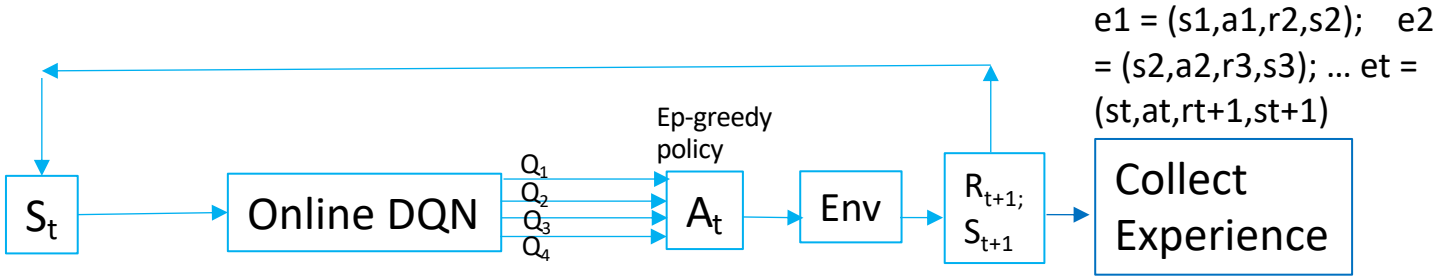


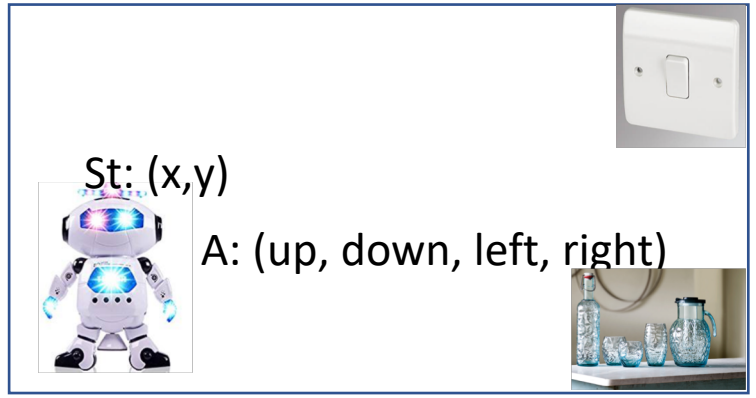
Exploitation: Choose the action with highest Q-values

Exploration: Choose a random action

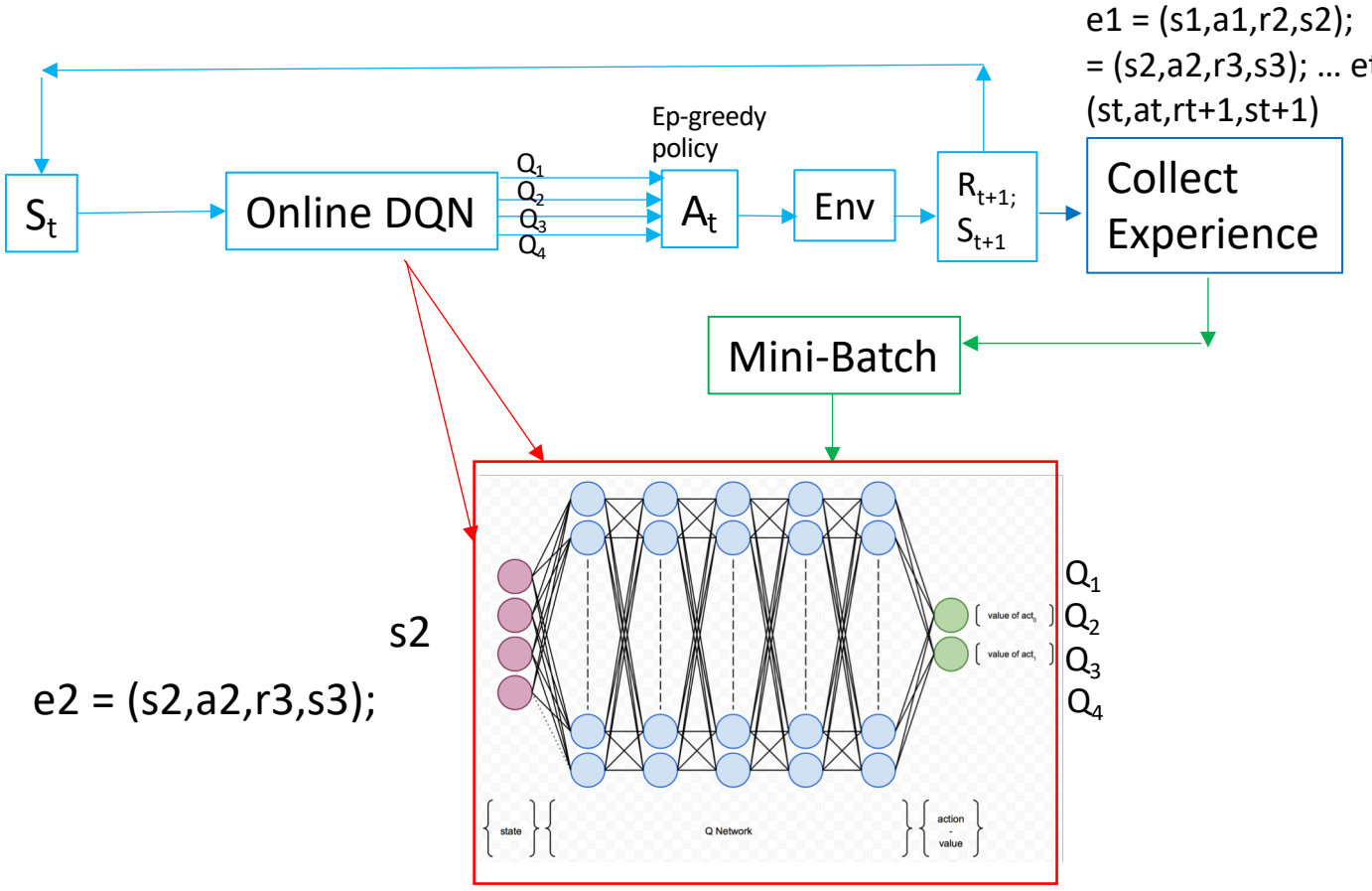
- Loss function: $Loss = \left[Y_t^Q - Q(S_t, A_t; \theta_t)\right]^2$
- Target Q-value: $Y_t^Q \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)$. This is Bellman equation
- Parameter update: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^Q - Q(S_t, A_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t)$.

Deep Q-Learning (Minh et al (2015))



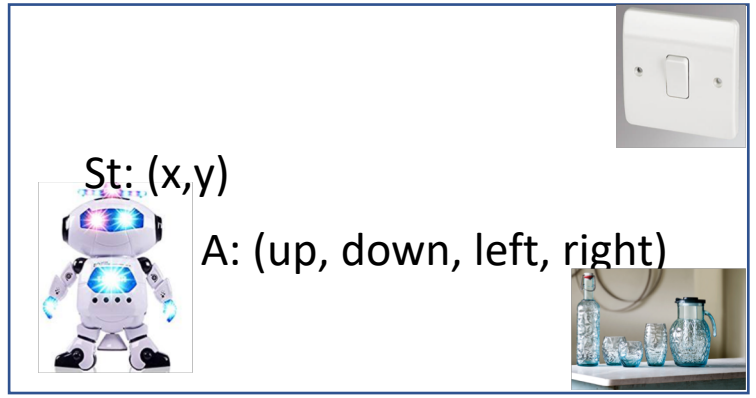


Deep Q-Learning (Minh et al (2015))

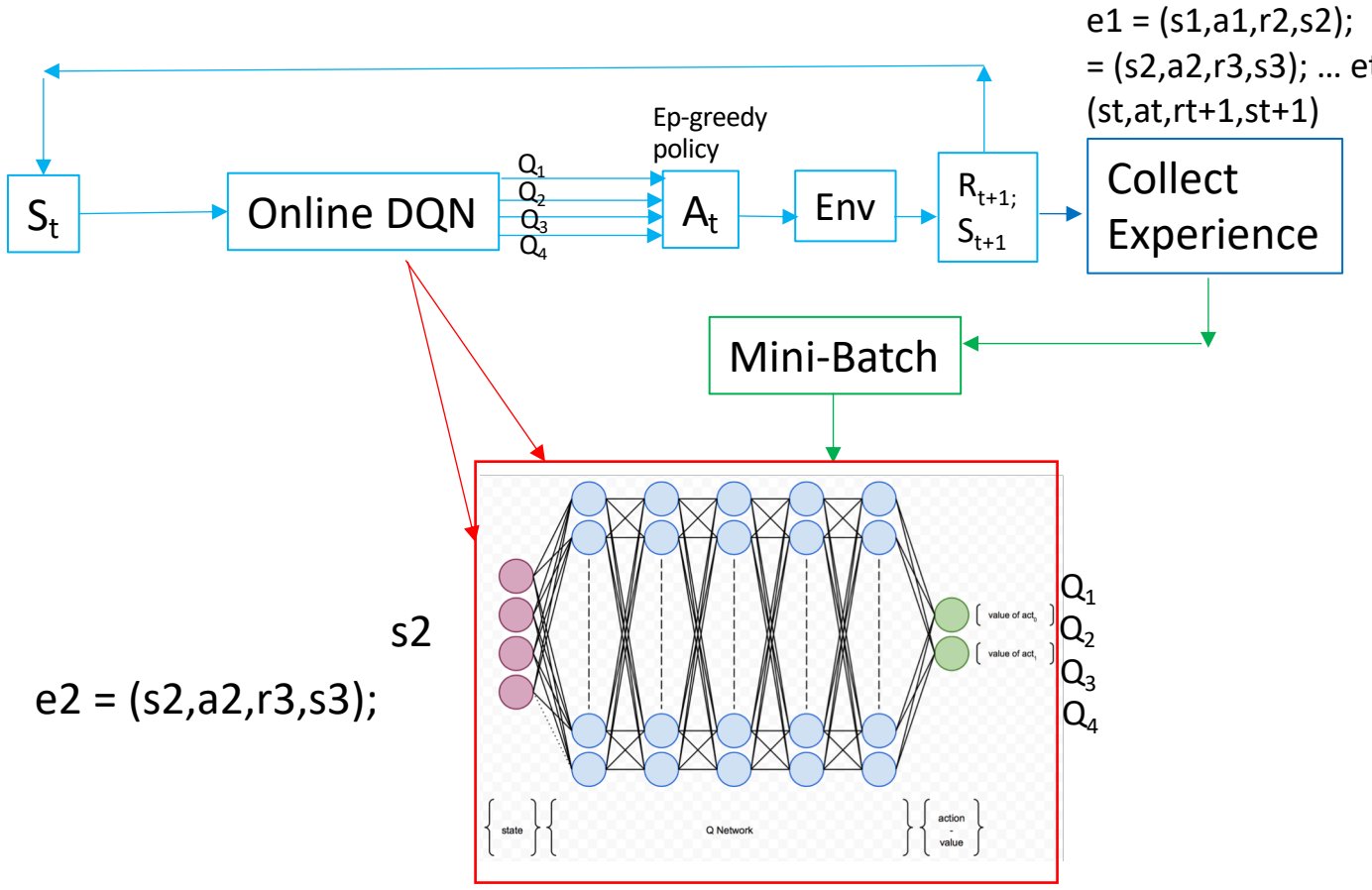


Online Deep Q-Network

$$Loss = \left[Y_t^{DQN} - Q(A_t)\right]^2$$

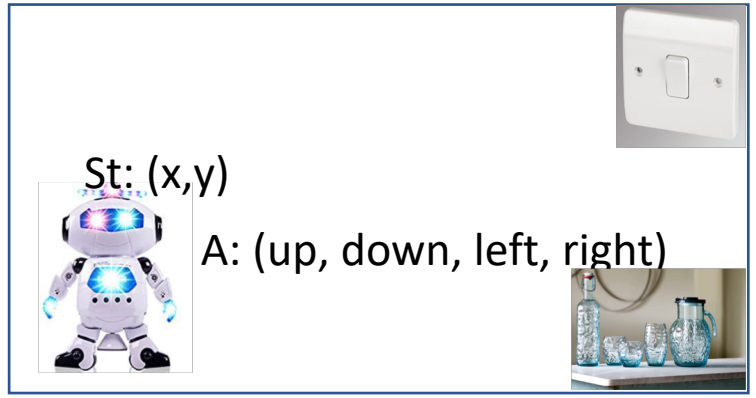


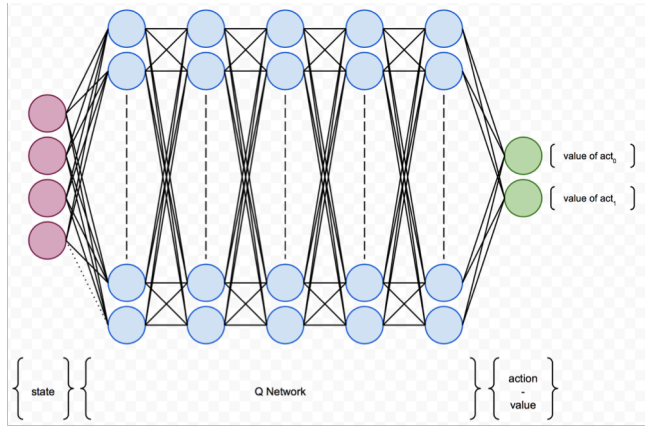
Deep Q-Learning (Minh et al (2015))



Online Deep Q-Network

$$Loss = \left[Y_t^{DQN} - Q(A_t)\right]^2$$



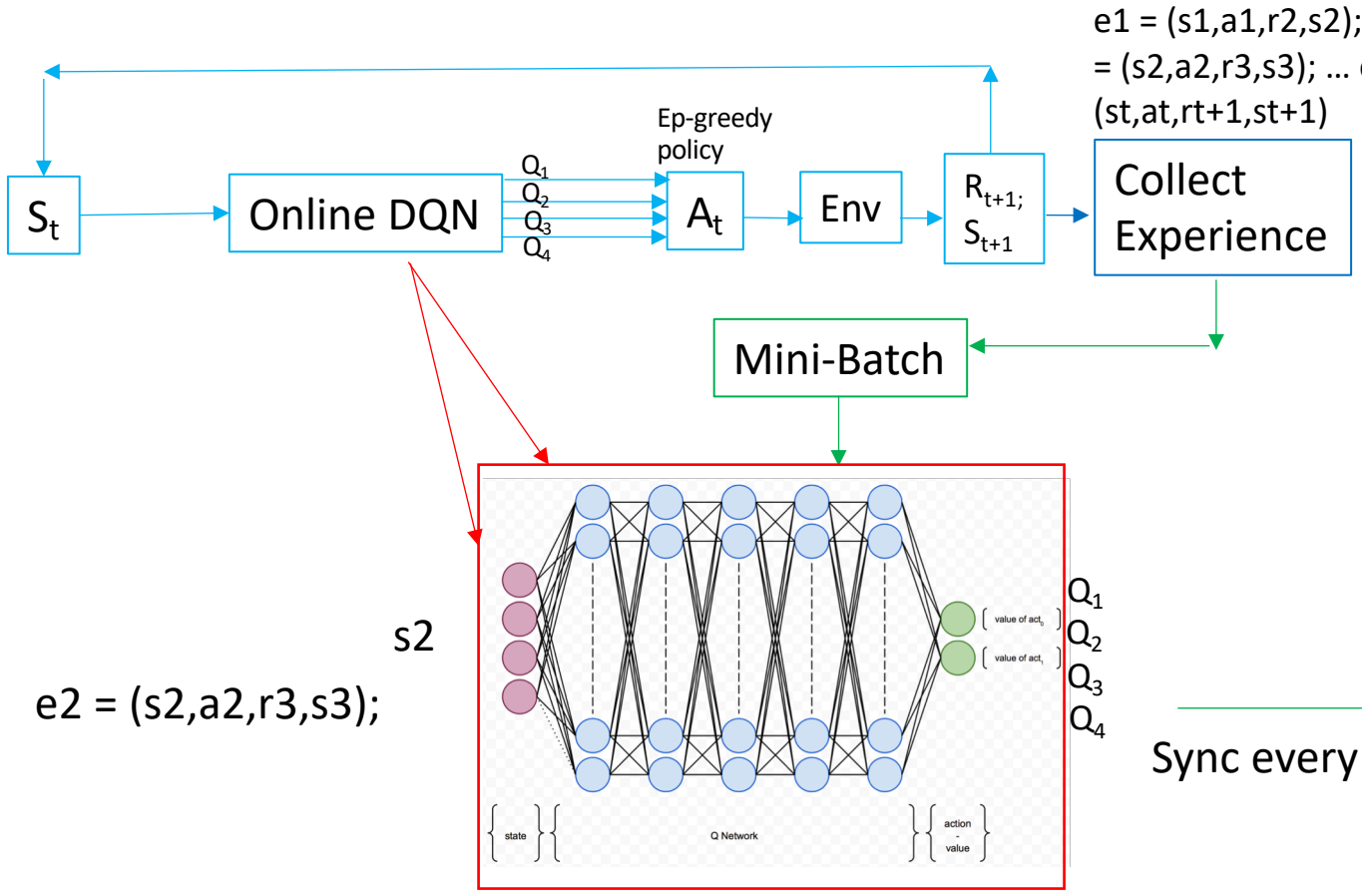


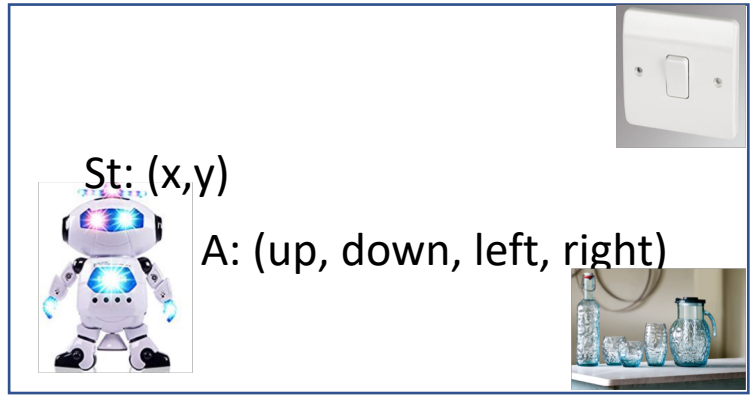
Target Deep Q-Network

$$Y_t^{DQN} = R_{t+1} + \gamma . max_a Q'(a)$$

1. Choose Action
2. Get its Q-value

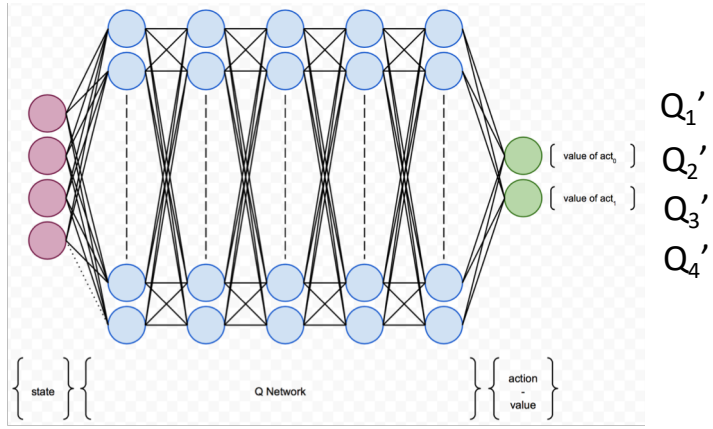
Deep Q-Learning (Minh et al (2015))





Sync every C Steps

s3

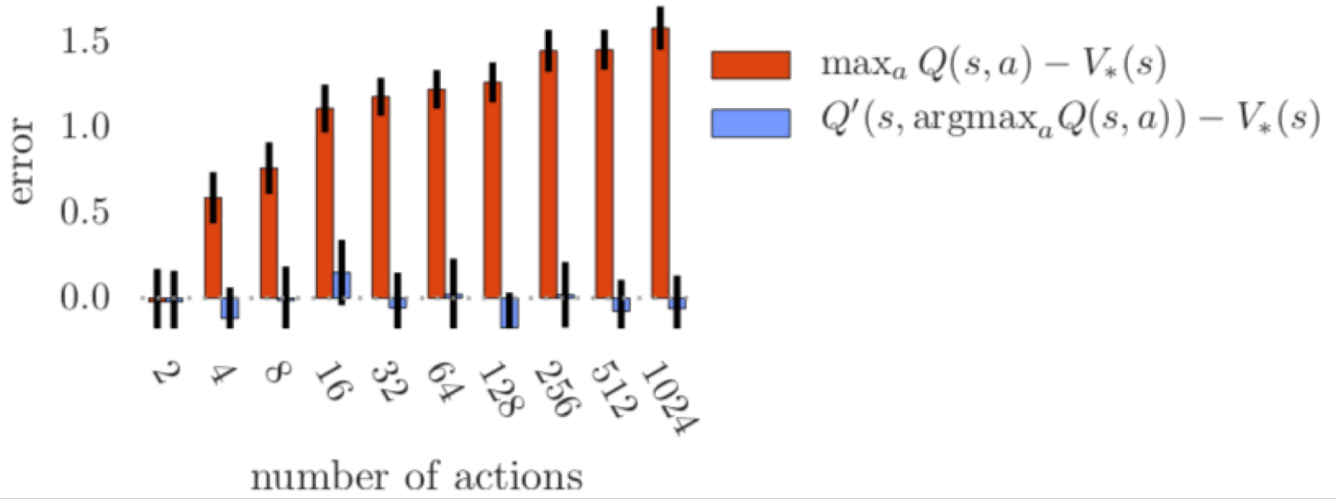


1. Choose Action
2. Get its Q-value

Reasons for Over-Estimation

- Insufficient flexible function approximation
- Noise or Stochasticity in rewards and/ or environment
- But, this paper shows the overestimation will occur even without these factors due to inherent estimation errors

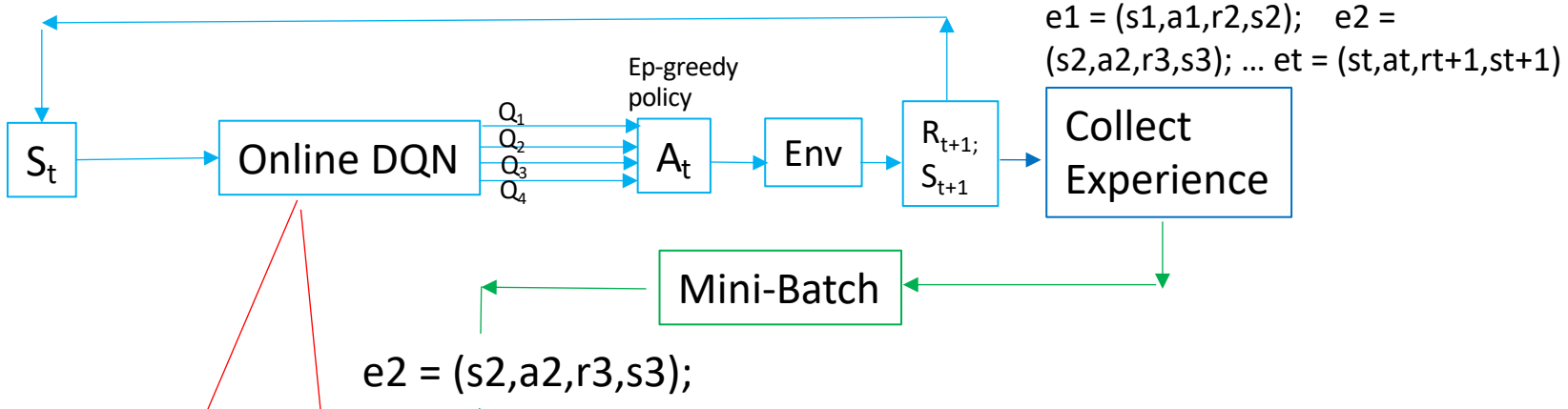
Theorem 1. *Consider a state s in which all the true optimal action values are equal at $Q_*(s,a) = V_*(s)$ for some $V_*(s)$. Let Q_t be arbitrary value estimates that are on the whole unbiased in the sense that $\sum_a (Q_t(s,a) - V_*(s)) = 0$, but that are not all correct, such that $\frac{1}{m}\sum_a (Q_t(s,a) - V_*(s))^2 = C$ for some $C > 0$, where $m \geq 2$ is the number of actions in s. Under these conditions, $\max_a Q_t(s,a) \geq V_*(s) + \sqrt{\frac{C}{m-1}}$. This lower bound is tight. Under the same conditions, the lower bound on the absolute error of the Double Q-learning estimate is zero. (Proof in appendix.)*

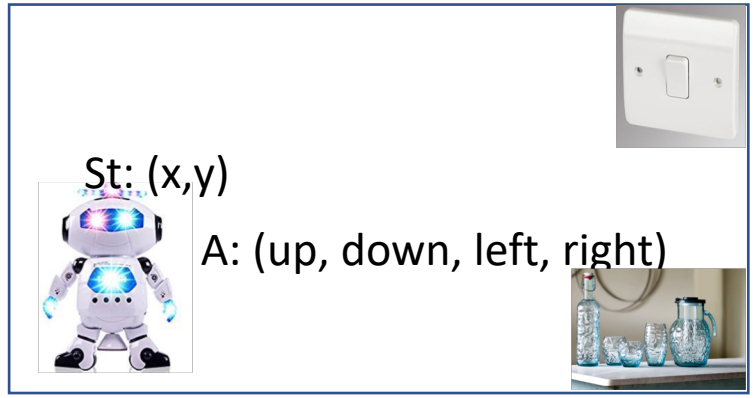


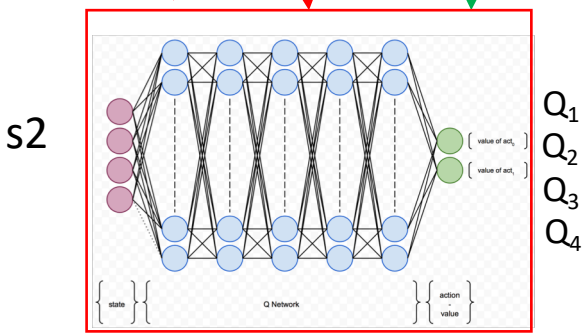
If $Q_t(s,a) - Q_*(s,a)$ has uniform random distribution in [-1,1], then

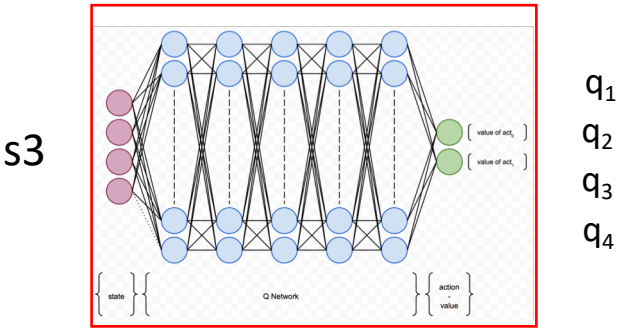
$$Q_t(s,a) \geq Q_*(s,a) + \frac{m-1}{m+1}$$

Double Deep Q-Learning

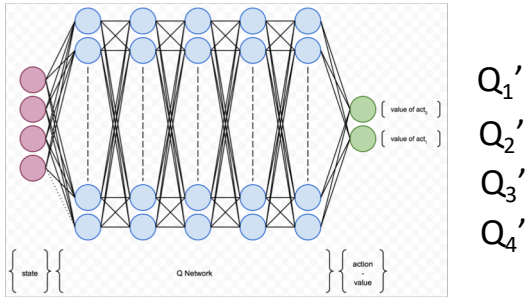








Sync every C Steps



$$Loss = \left[Y_t^{DDQN} - Q(A_t)\right]^2$$

Choose action: $A_{t+1} = argmax_a q(a)$

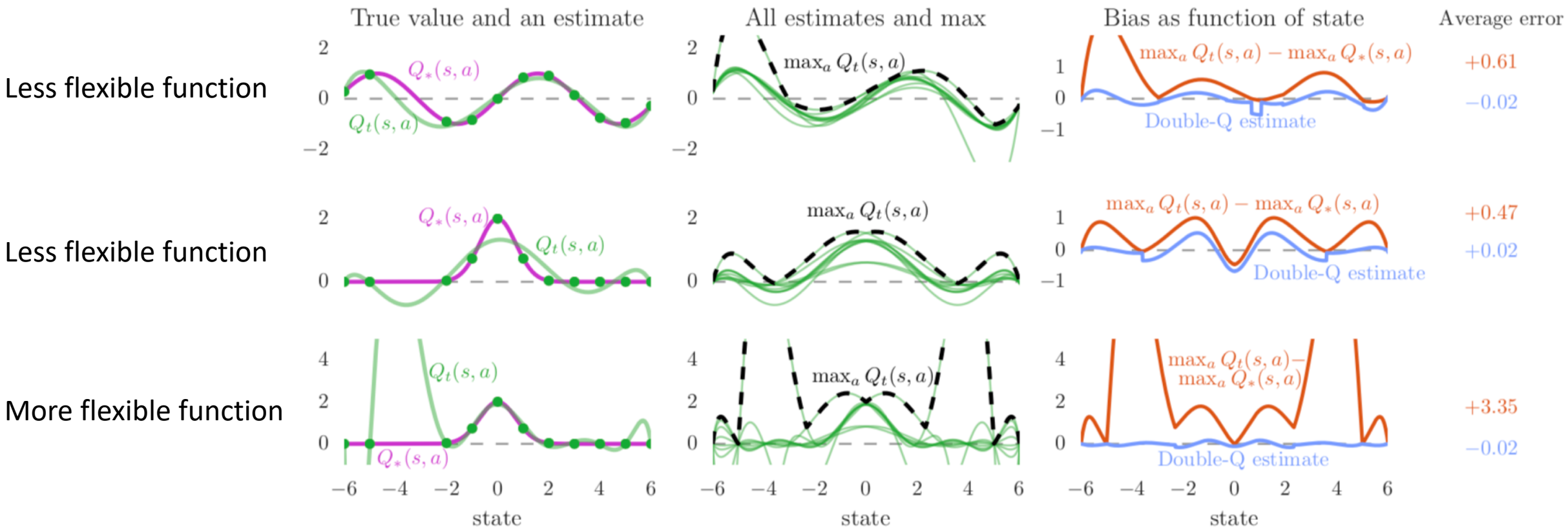
$$Y_t^{DDQN} = R_{t+1} + \gamma\, Q'(A_{t+1})$$

Get the Q-value

Break

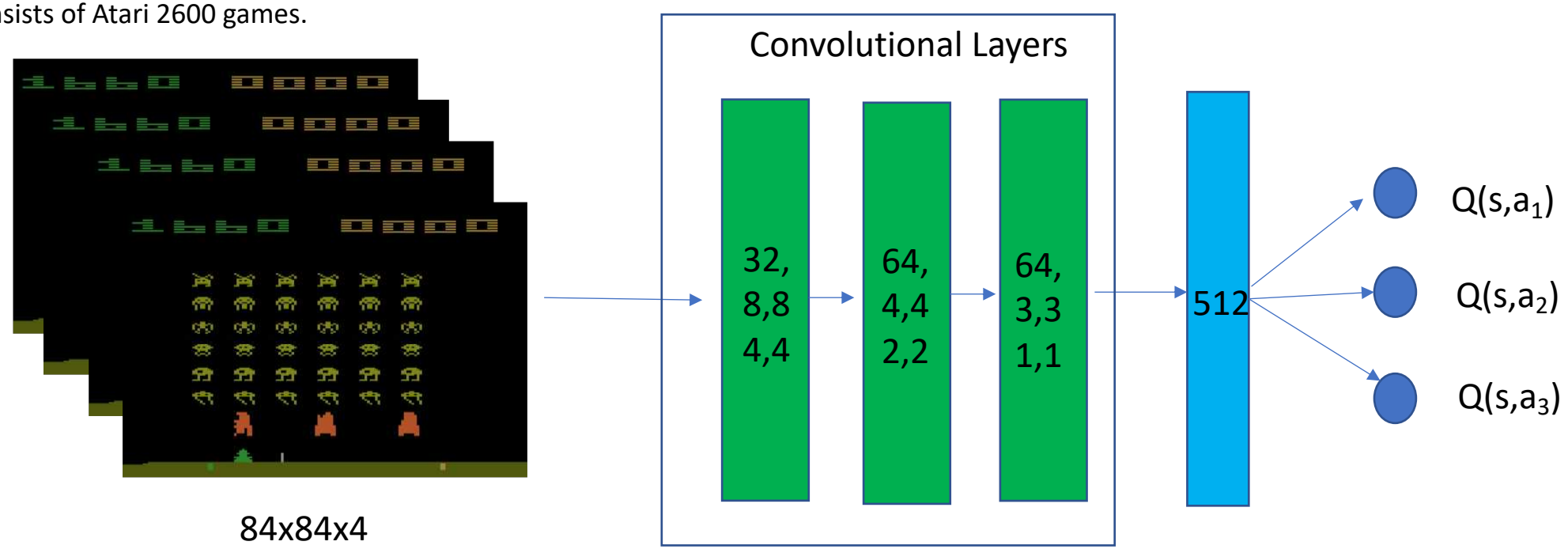
- After break, we will discuss
  - The results of q-learning vs. double q-learning

Q-learning Vs Double Q-learning



Empirical Results

- Testbed consists of Atari 2600 games.

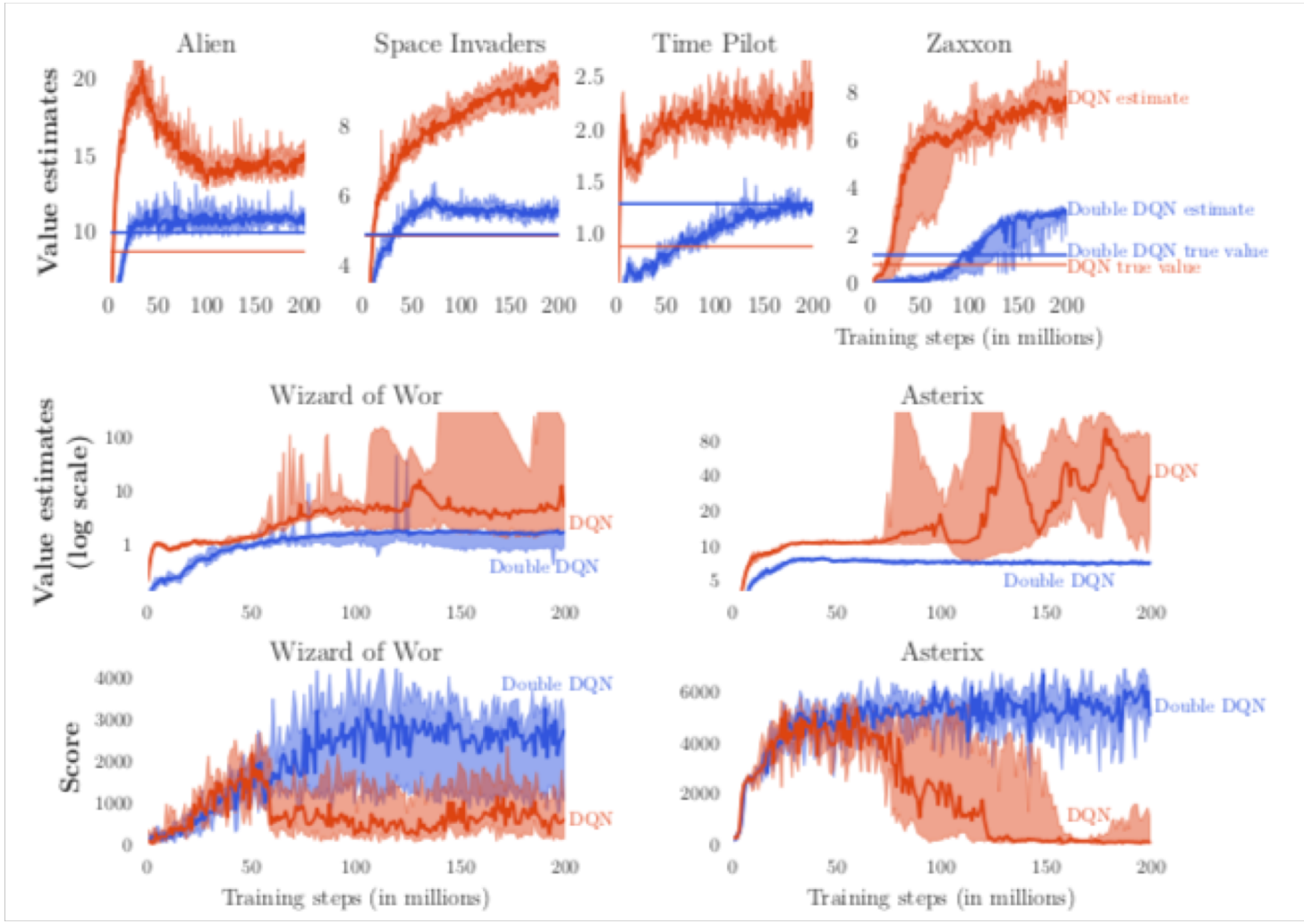


- Trained for 200M frames (or approx. 1 week)
- Backprop using RMSProp algorithm
- Discount rate = 0.99
- learning rate = 0.00025, steps between target network update = 10,000, steps = 50M, size of memory = 1M tuples, mini-batch size =32, network update interval = 4, epsilon decreasing linearly from 1 to 0.1 over 1M steps.

DQN Vs DDQN in 6 different Atari Games

- 6 different random seeds as starting point for Atari games
- The horizontal orange and blue lines represent actual discounted value of the best learned policy over many several episodes
- More precisely, the horizontal line estimates are calculated as follows:

$$\frac{1}{T}\sum_{t=1}^{T}\underset{a}{\operatorname{argmax}}\, Q(S_t, a; \boldsymbol{\theta})$$



Normalized scores on 57 Atari Games

- Tested for 100 episodes per game

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}.$$

- Epsilon = 0.05
- Scores are averaged over 100 episodes

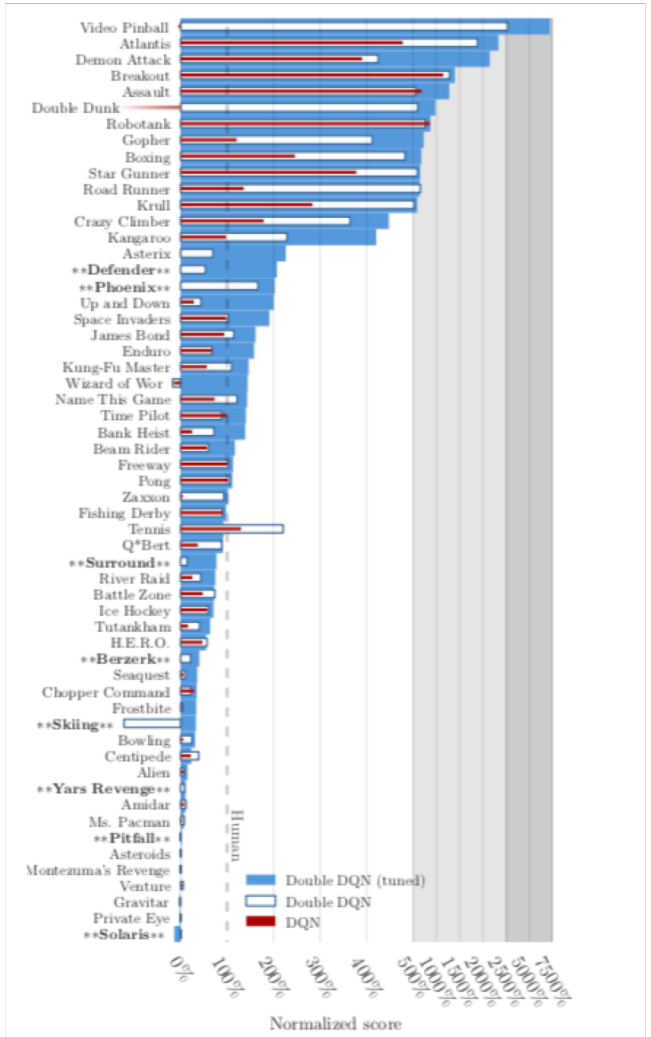


Figure 4: Normalized scores on 57 Atari games, tested for 100 episodes per game with human starts. Compared to Mnih et al. (2015), eight games additional games were tested. These are indicated with stars and a bold font.

Conclusion

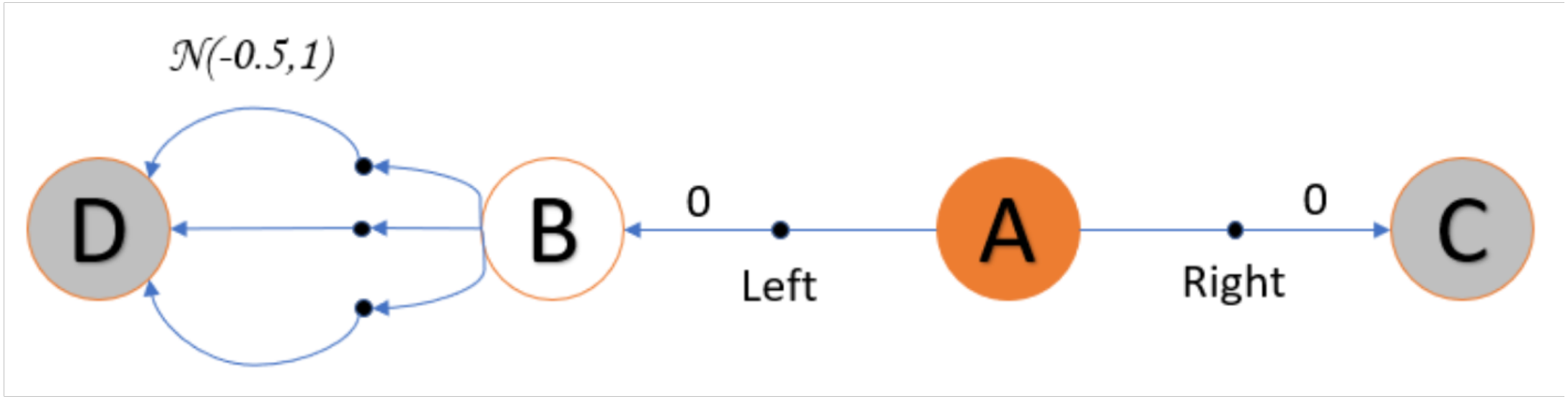
- It is shown why Q-learning can be overoptimistic
- Overestimations can occur in any scenario
- Double Q-learning reduces the overestimations by avoiding the max operation on target Q-function
- Double Q-learning is only a slight modification to the Q-leaning
- Double Q-learning finds better policies obtaining new state-of-the-art results on the Atari 2600 domain

Discussion Points

- Instead of synching networks after C steps, why not randomly select which network is online and which one is target?
- Question: Would actor-critic outperform double DQN?
- Instead of randomly choosing a mini-batch, why don't we carefully choose those batches of experiences from which our Deep Q-Network will learn the most (i.e. better sampling techniques)
- What is the effect of choosing deep network on the speed of learning?
- Why have two different networks (i.e. online and target)?

Questions

Example of Overestimation

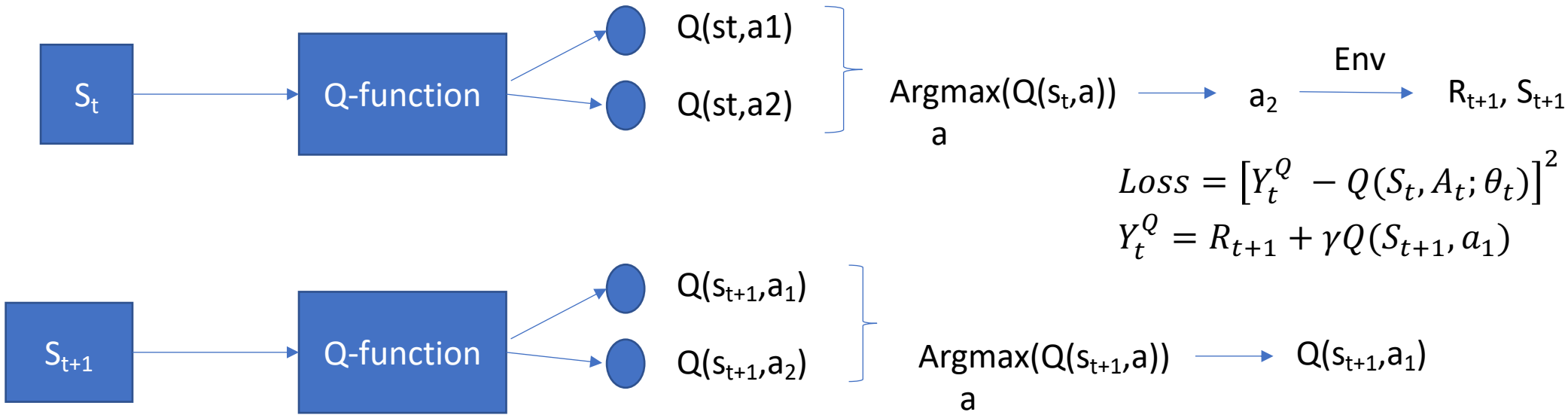


https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3

Universal Q-Learning

Skip this one

- At any time t, we are in state S_t



$$Loss = \left[Y_t^Q - Q(S_t, A_t; \theta_t)\right]^2$$

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, a_1)$$

Two problems:

1) Using the same network for calculating the Q(s,a) and Q*(s,a)
2) The consecutive states may be correlated?

Use a different network

Memory replay

Policy

- A policy is a function that maps a given state to probabilities of selecting each possible action from that state. We will use the symbol π to denote a policy.

	A1	A2	A3	A4
S1	P11	P12	P13	P14
S2	P21	P22	P23	P24
S3	P31	P32	P33	P34

Double Q-Learning

May be unnecessary.
May move to appendix

- Decouple selection of action from value of action

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

Q-value for action a that corresponds to max Q-value

$$Y_t^{\mathrm{Q}} = R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t) .$$

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

Offline function approx.

Action selection via Online function approx.

- There are two sets of Q-functions Q and Q'. The function Q is used to select the best action given any state and Q' is used to evaluate its Q-value.

Full DQN Algorithm

given replay memory $\mathcal{D}$ with capacity N

initialize Q-networks Q, $\hat{Q}$ with same random weights θ

repeat until timeout

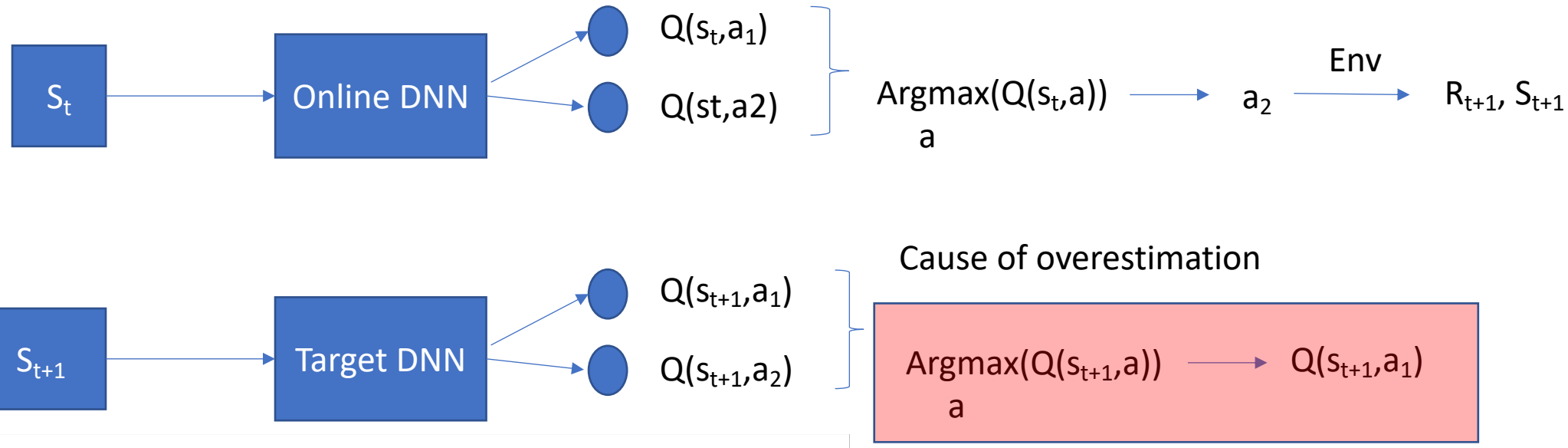
initialize frame sequence $s_1 = \{x_1\}$ and preprocessed state $\phi_1 = \phi(s_1)$

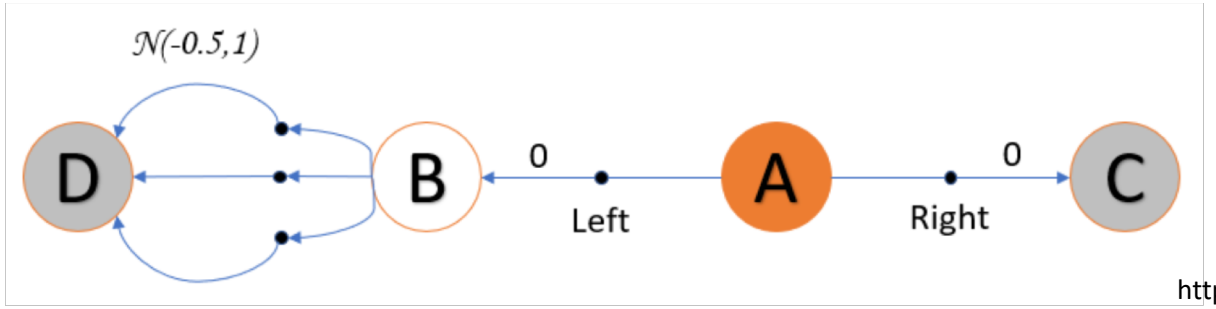
for $t = 1, \ldots, T$

1. select action $a_t = \begin{cases} \max_a Q(\phi(s_t), a; \theta) & \text{w.p. } 1-\epsilon \\ \text{random action} & \text{otherwise} \end{cases}$
2. execute action a_t and observe reward r_t and frame x_{t+1}
3. append $s_{t+1} = (s_t, a_t, x_{t+1})$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
4. store experience $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
5. uniformly sample minibatch $(\phi_j, a_j, r_j, \phi_{j+1}) \sim \mathcal{D}$
6. set $y_j = \begin{cases} r_j & \text{if } \phi_{j+1} \text{ terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta) & \text{otherwise} \end{cases}$
7. perform gradient descent step for Q on minibatch
8. every C steps reset $\hat{Q} = Q$

Universal DQN-Learning

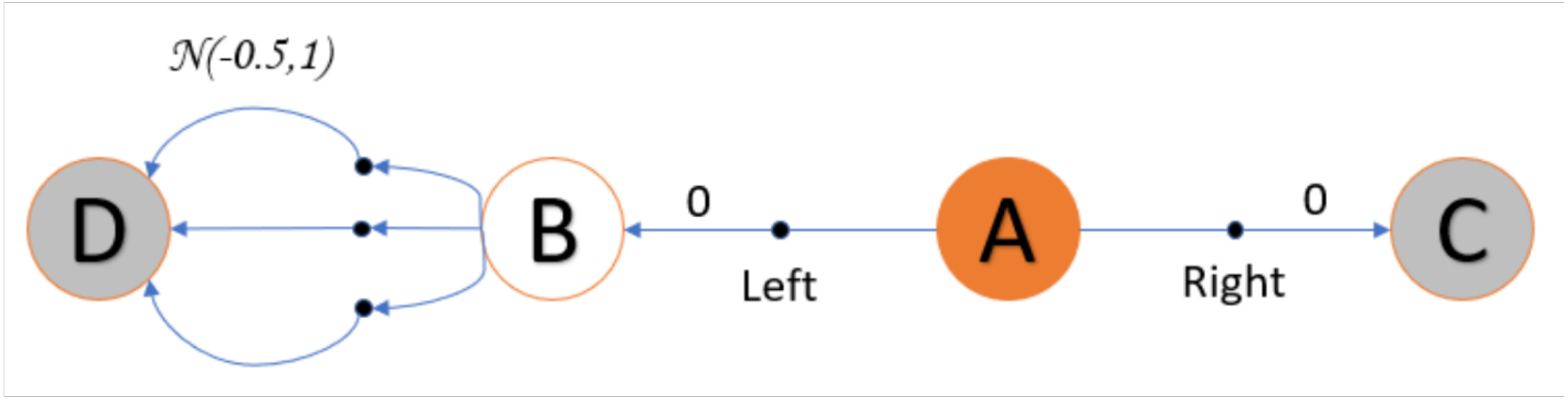
- At any time t, we are in state S_t





https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3

Example of Overestimation



https://towardsdatascience.com/double-q-learning-the-easy-way-a924c4085ec3