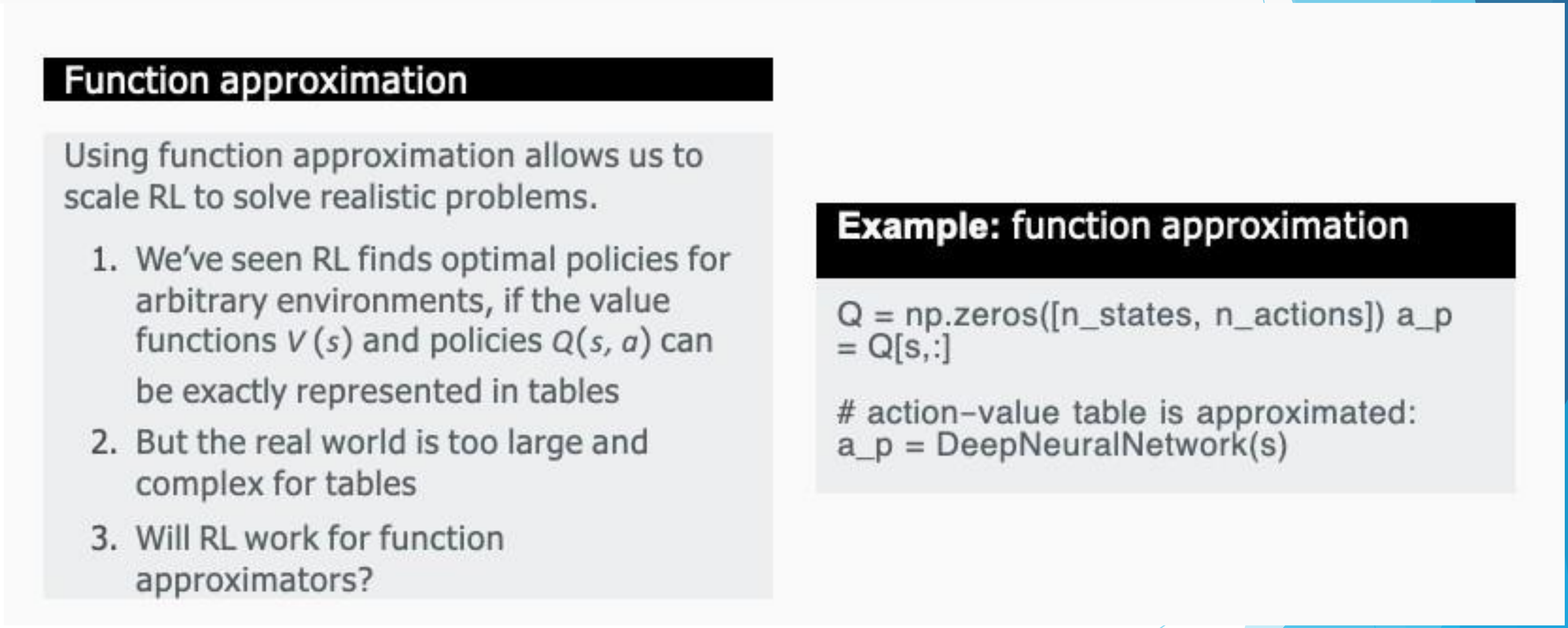
Lecture-7
Function approximation

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Function Approximation



Function approximation

Using function approximation allows us to scale RL to solve realistic problems.

1. We've seen RL finds optimal policies for arbitrary environments, if the value functions $V(s)$ and policies $Q(s, a)$ can be exactly represented in tables
2. But the real world is too large and complex for tables
3. Will RL work for function approximators?

Example: function approximation

```
Q = np.zeros([n_states, n_actions]) a_p
= Q[s,:]

# action–value table is approximated:
a_p = DeepNeuralNetwork(s)
```

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Function Approximation

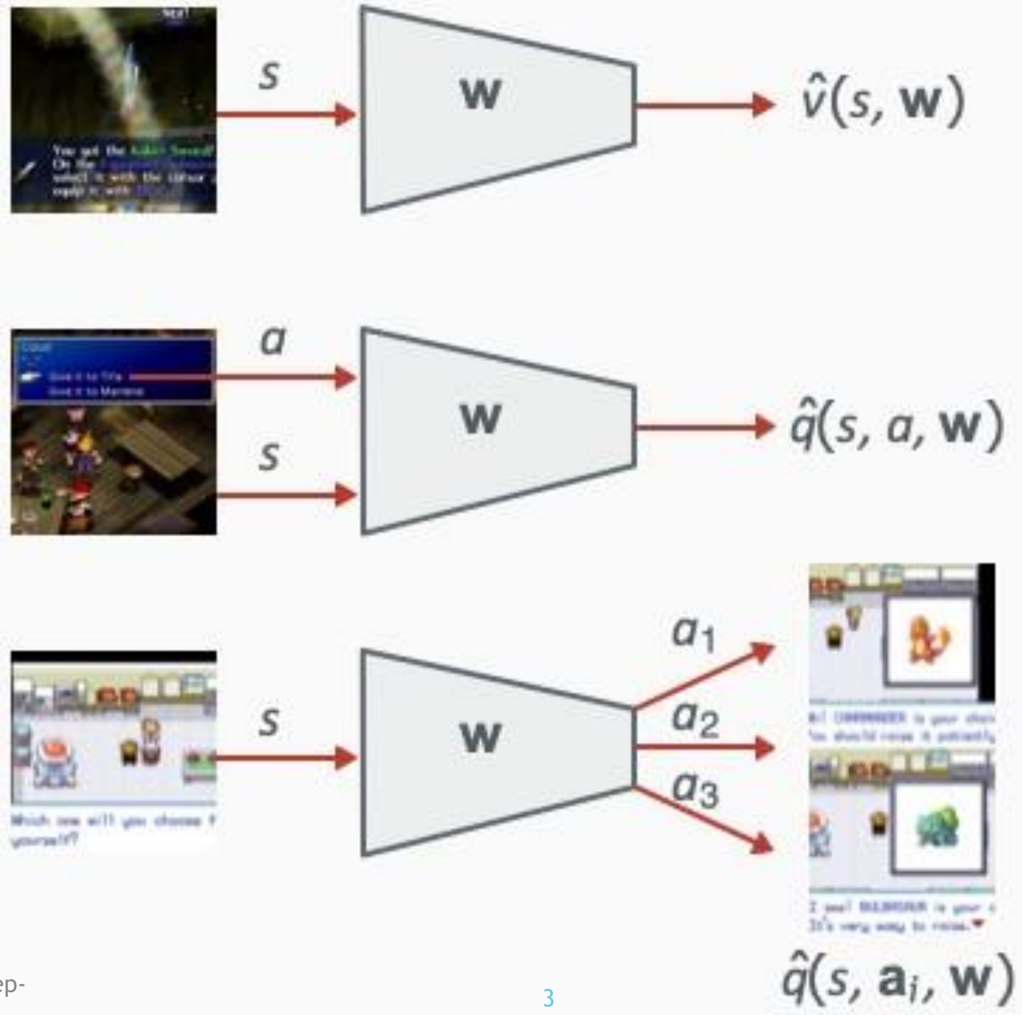
Definition: Function approximation

There are too many states/actions to fit into memory, which are too slow to process. Therefore we estimate the value function:

$$\hat{v}(S, \mathbf{w}) \approx v_\pi(S),$$

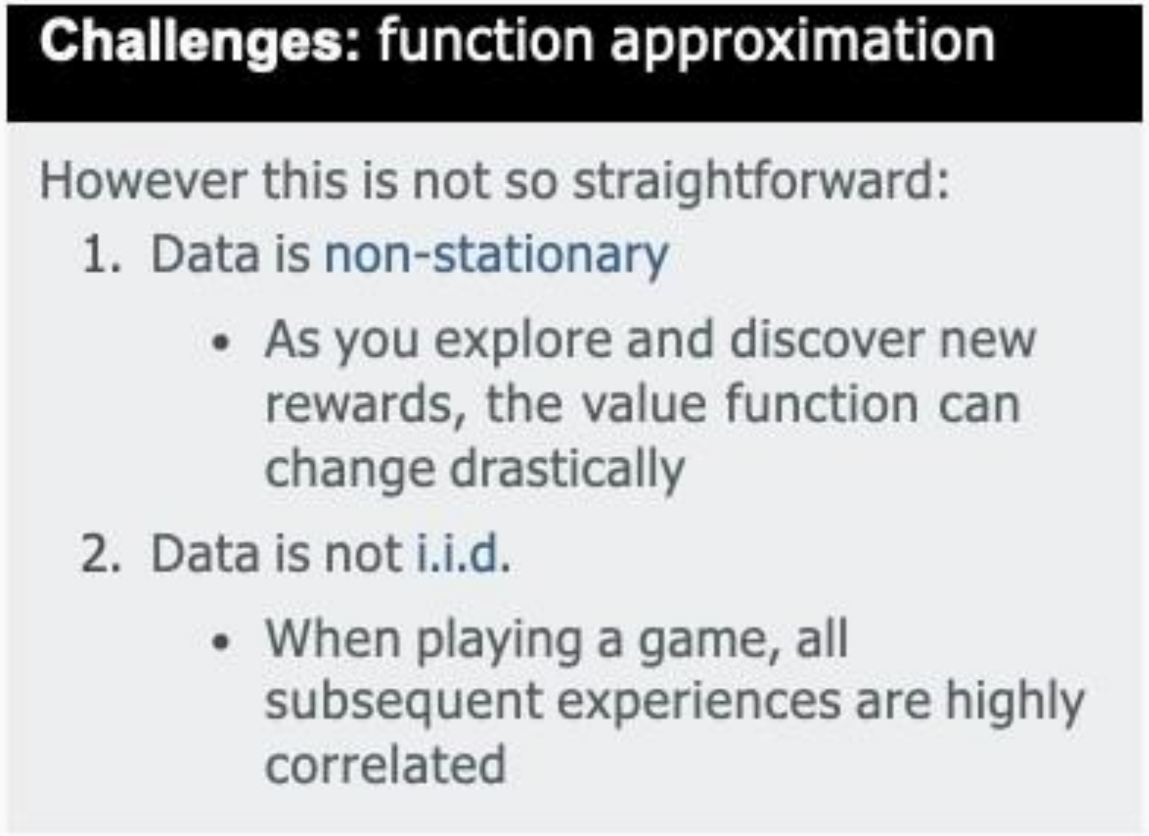
or for control we'd do:

$$\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A),$$



https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Function Approximation Challenges



Challenges: function approximation

However this is not so straightforward:

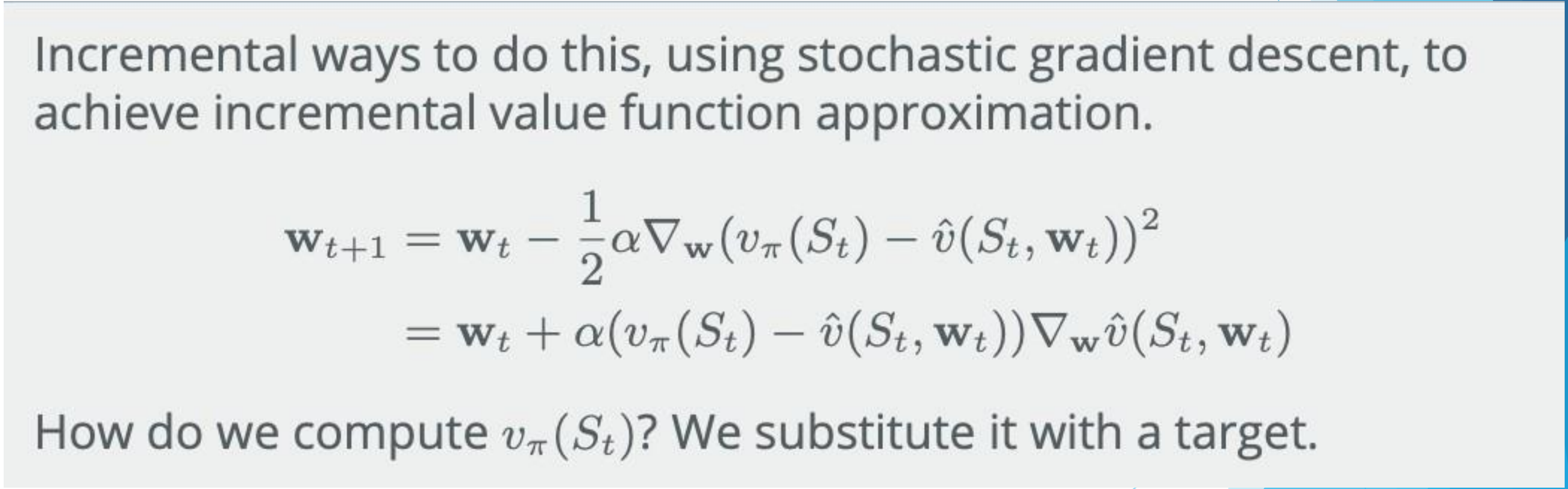
1. Data is non-stationary
    - As you explore and discover new rewards, the value function can change drastically
2. Data is not i.i.d.
    - When playing a game, all subsequent experiences are highly correlated

Example: fog of war

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Incremental methods- stochastic gradient descent for prediction

Incremental SGD for prediction



Incremental ways to do this, using stochastic gradient descent, to achieve incremental value function approximation.

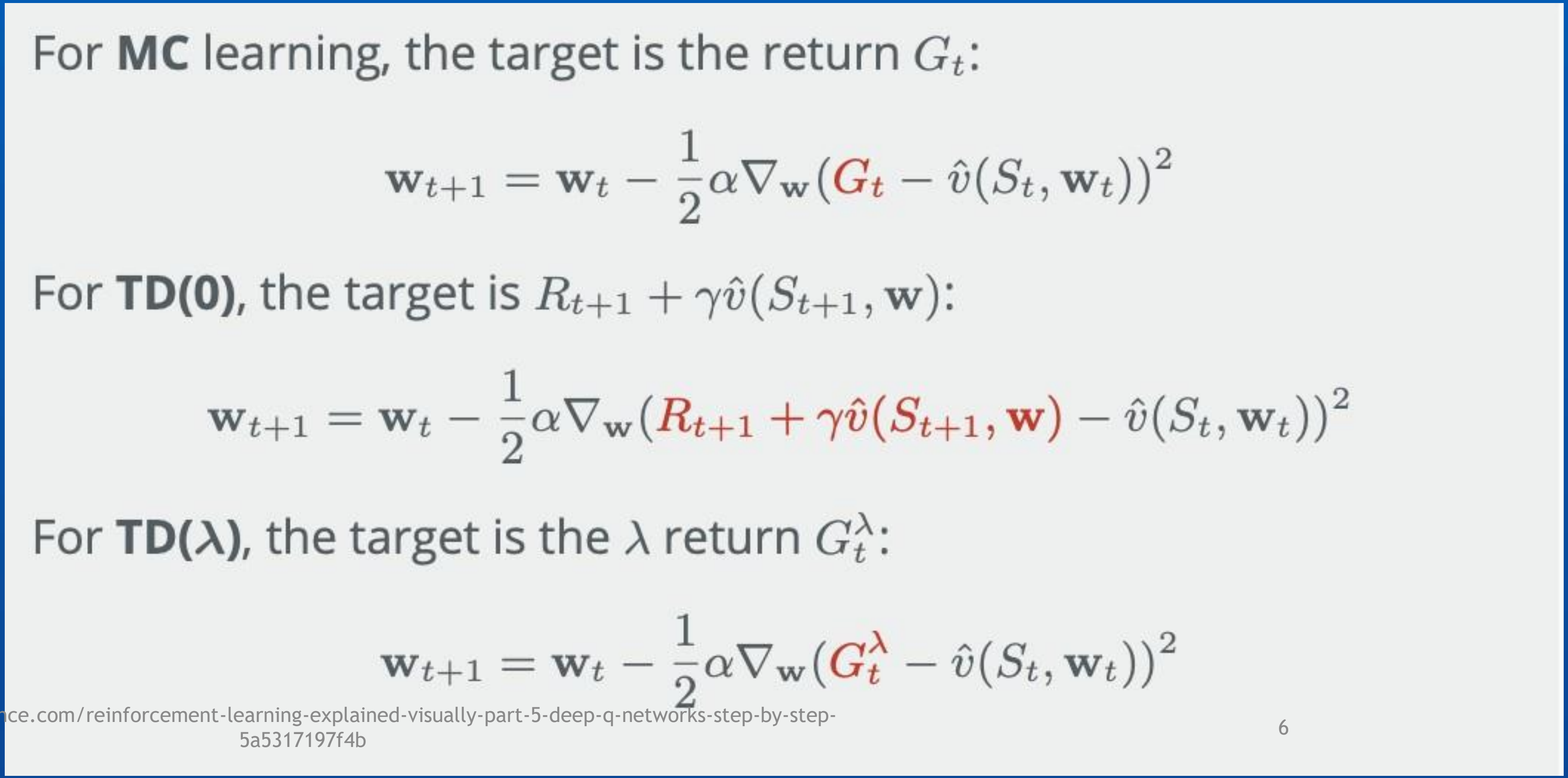
$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}\left(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right)^2 \\ &= \mathbf{w}_t + \alpha\left(v_\pi(S_t) - \hat{v}(S_t, \mathbf{w}_t)\right)\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

How do we compute $v_\pi(S_t)$? We substitute it with a target.

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Incremental methods- stochastic gradient descent for prediction

substituting $V_\pi(S_t)$



For **MC** learning, the target is the return G_t:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}\left(G_t - \hat{v}(S_t, \mathbf{w}_t)\right)^2$$

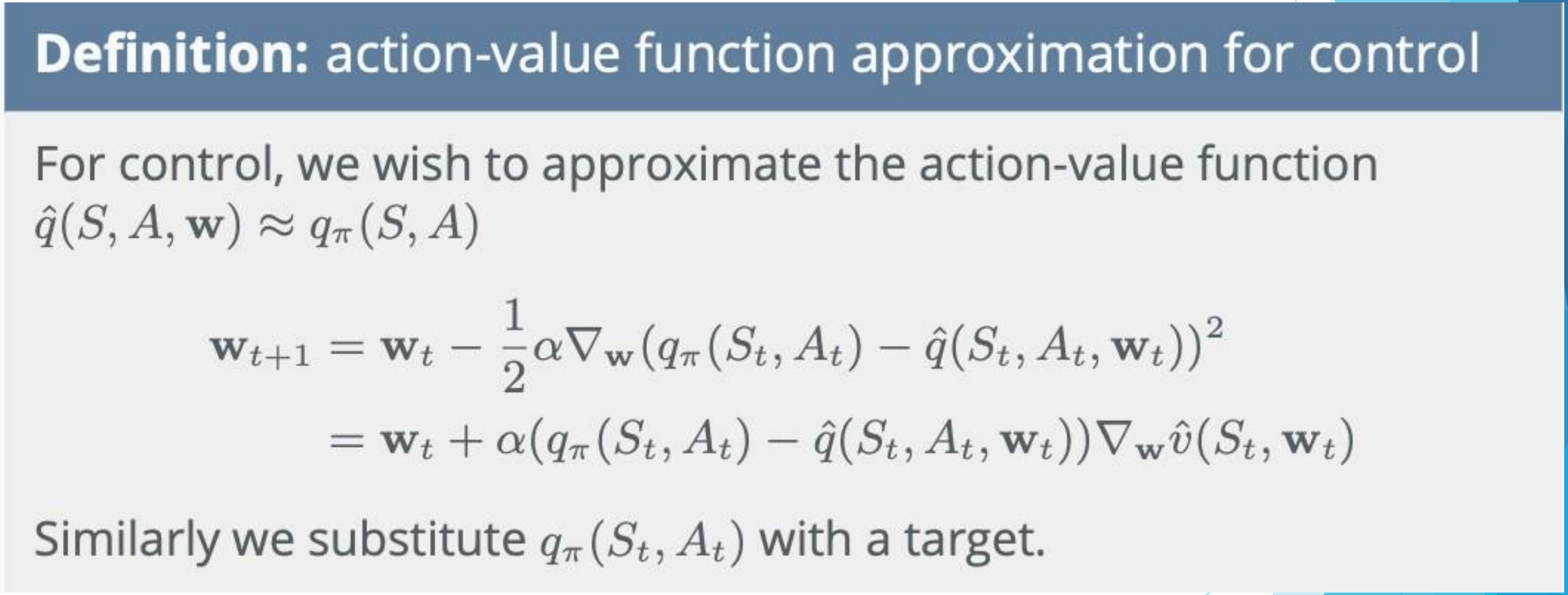
For **TD(0)**, the target is $R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w})$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}\left(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}_t)\right)^2$$

For **TD(λ)**, the target is the λ return G_t^λ:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}\left(G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t)\right)^2$$

Incremental methods- stochastic gradient descent for control



Definition: action-value function approximation for control

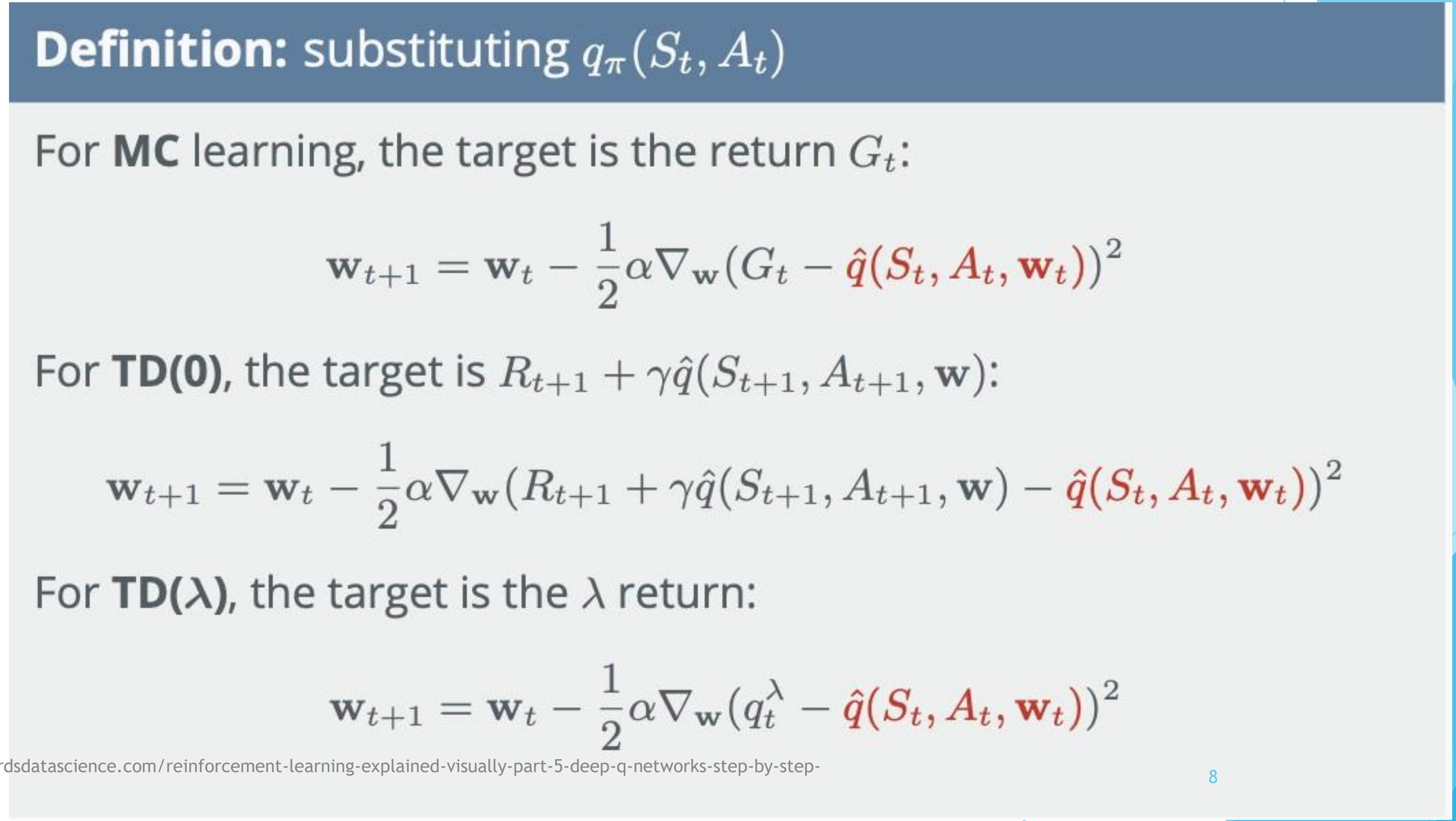
For control, we wish to approximate the action-value function $\hat{q}(S, A, \mathbf{w}) \approx q_\pi(S, A)$

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}(q_\pi(S_t, A_t) - \hat{q}(S_t, A_t, \mathbf{w}_t))^2 \\ &= \mathbf{w}_t + \alpha(q_\pi(S_t, A_t) - \hat{q}(S_t, A_t, \mathbf{w}_t))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w}_t)\end{aligned}$$

Similarly we substitute $q_\pi(S_t, A_t)$ with a target.

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Incremental methods- stochastic gradient descent for control



Definition: substituting $q_\pi(S_t, A_t)$

For **MC** learning, the target is the return G_t:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}(G_t - \hat{q}(S_t, A_t, \mathbf{w}_t))^2$$

For **TD(0)**, the target is $R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w})$:

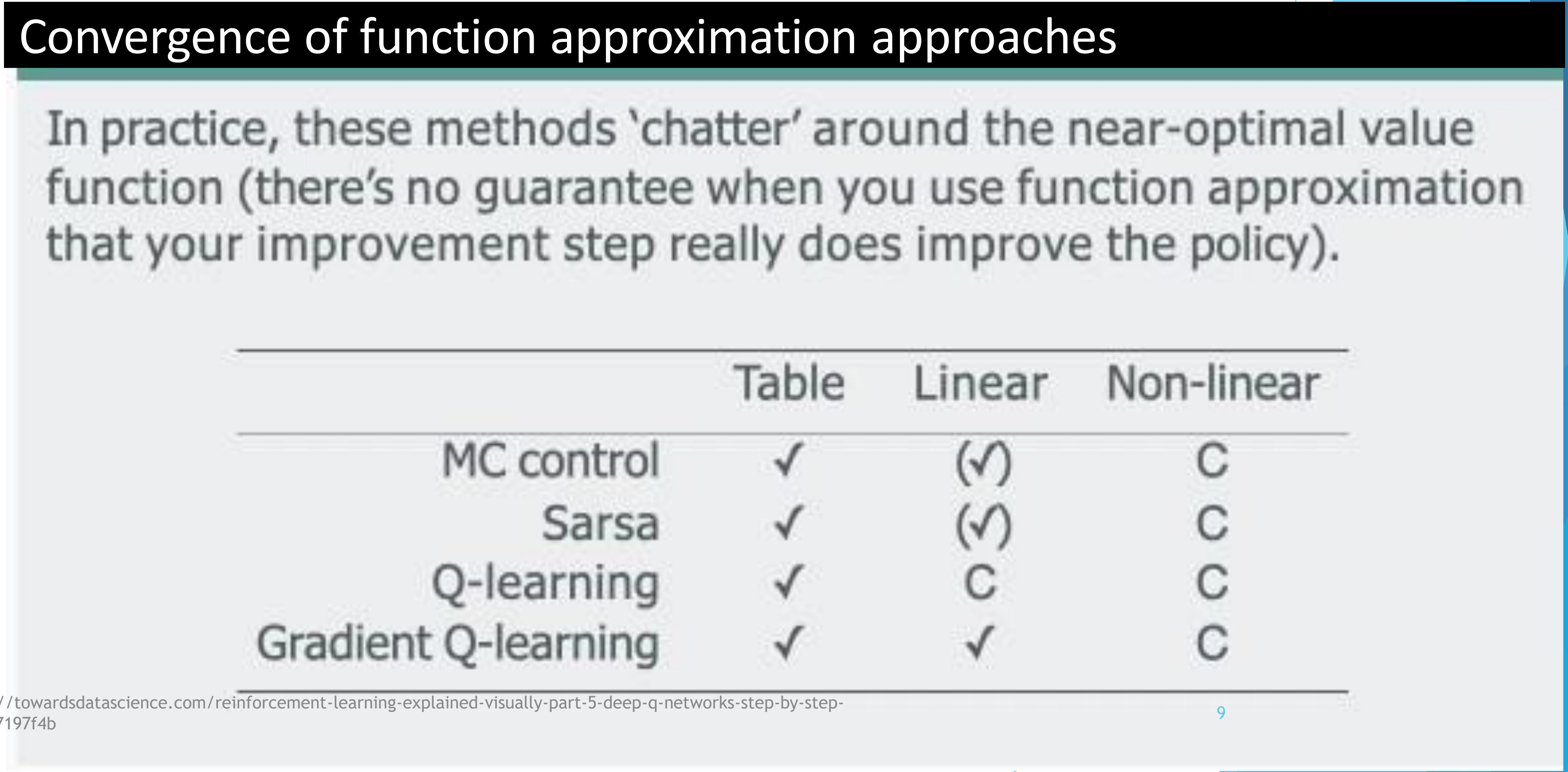
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w}_t))^2$$

For **TD(λ)**, the target is the λ return:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{2}\alpha\nabla_{\mathbf{w}}(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}_t))^2$$

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Incremental methods- convergence



Convergence of function approximation approaches

In practice, these methods 'chatter' around the near-optimal value function (there's no guarantee when you use function approximation that your improvement step really does improve the policy).

	Table	Linear	Non-linear
MC control	✓	(✓)	C
Sarsa	✓	(✓)	C
Q-learning	✓	C	C
Gradient Q-learning	✓	✓	C

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

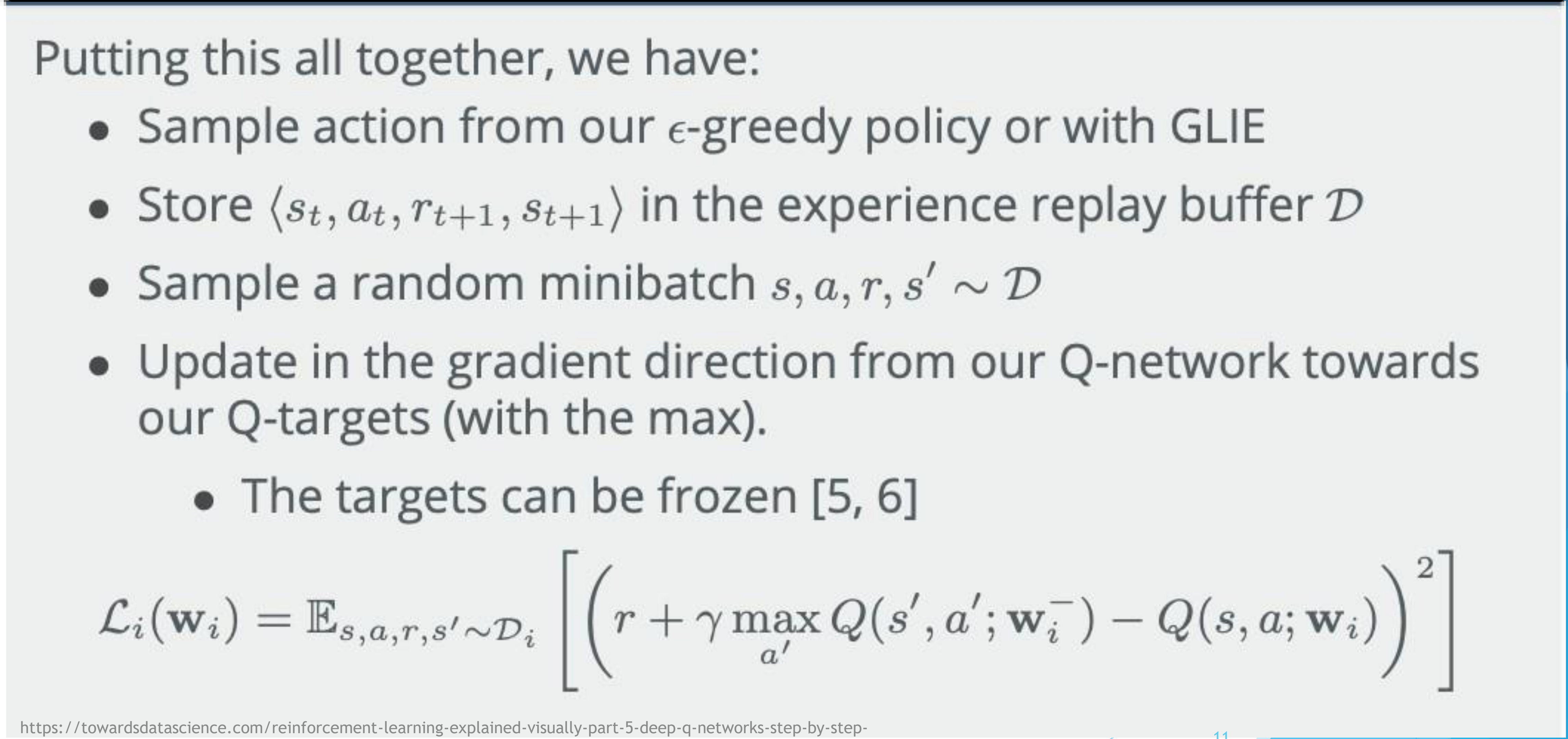
Batch learning- experience

Incremental methods have several problems:

1. They are not sample efficient
2. They have strongly correlated updates that break the i.i.d. assumptions of popular SGD algorithms
3. They may rapidly forget rare experiences that would be useful later on

Experience replay and prioritized experience replay address these issues by storing experiences and reusing them. These can be sampled uniformly or prioritized to replay important transitions more frequently.

Batch learning- experience replay with double Q-learning



Putting this all together, we have:

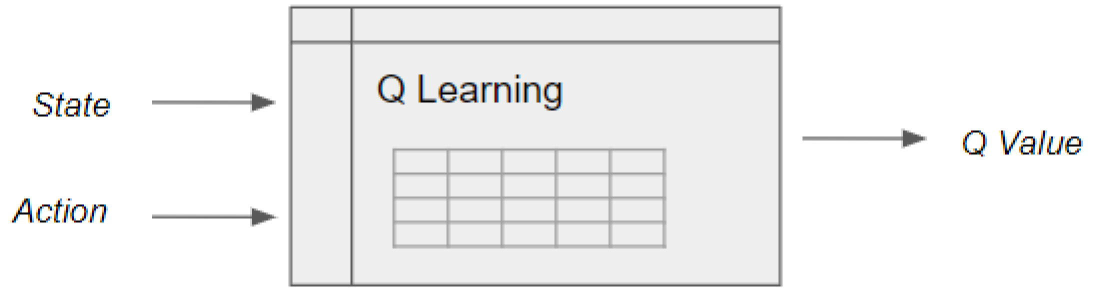
- Sample action from our ϵ-greedy policy or with GLIE
- Store $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ in the experience replay buffer $\mathcal{D}$
- Sample a random minibatch $s, a, r, s' \sim \mathcal{D}$
- Update in the gradient direction from our Q-network towards our Q-targets (with the max).
    - The targets can be frozen [5, 6]

$$\mathcal{L}_i(\mathbf{w}_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; \mathbf{w}_i^-) - Q(s, a; \mathbf{w}_i)\right)^2\right]$$

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Overview of Deep Q Networks

- Q Learning builds a Q-table of State-Action values, with dimension *(s, a)*, where *s* is the number of states and *a* is the number of actions. Fundamentally, a Q-table maps state and action pairs to a Q-value.



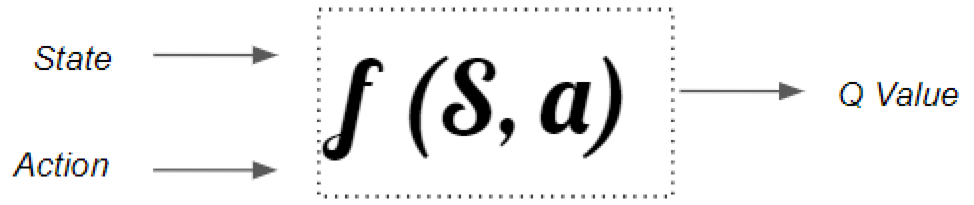
Q Learning looks up state-action pairs in a Q table

- However, in a real-world scenario, the number of states could be huge, making it computationally intractable to build a table.

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Use a Q-Function for real-world problems

- To address this limitation we use a Q-function rather than a Q-table, which achieves the same result of mapping state and action pairs to a Q value.

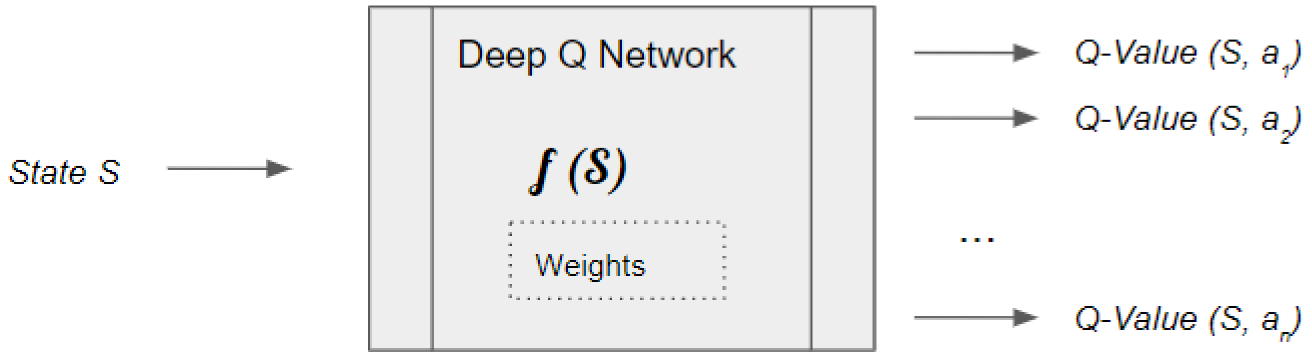


A state-action function is required to handle real-world scenarios with a large state space.

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Neural Networks are the best Function Approximators

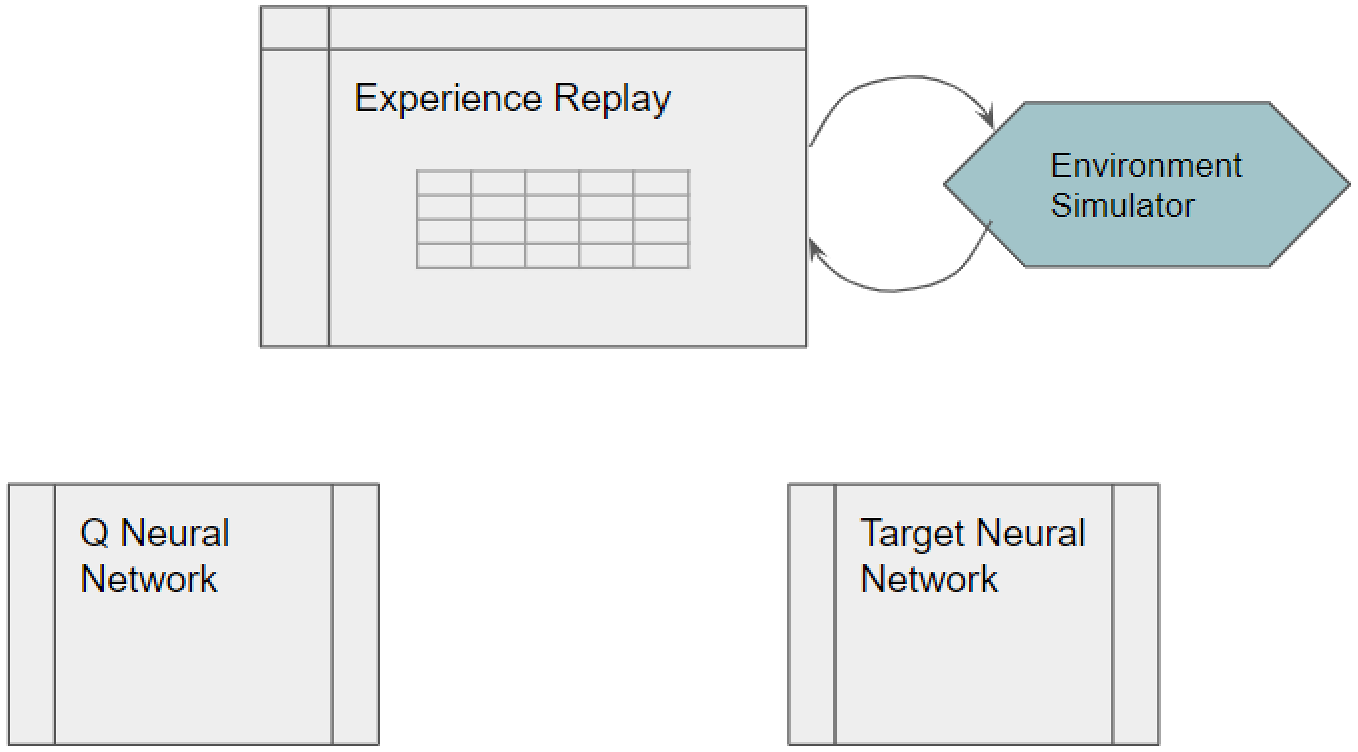
- Since neural networks are excellent at modeling complex functions, we can use a neural network, which we call a Deep Q Network, to estimate this Q function.
- This function maps a state to the Q values of all the actions that can be taken from that state.



- It learns the network's parameters (weights) such that it can output the Optimal Q values

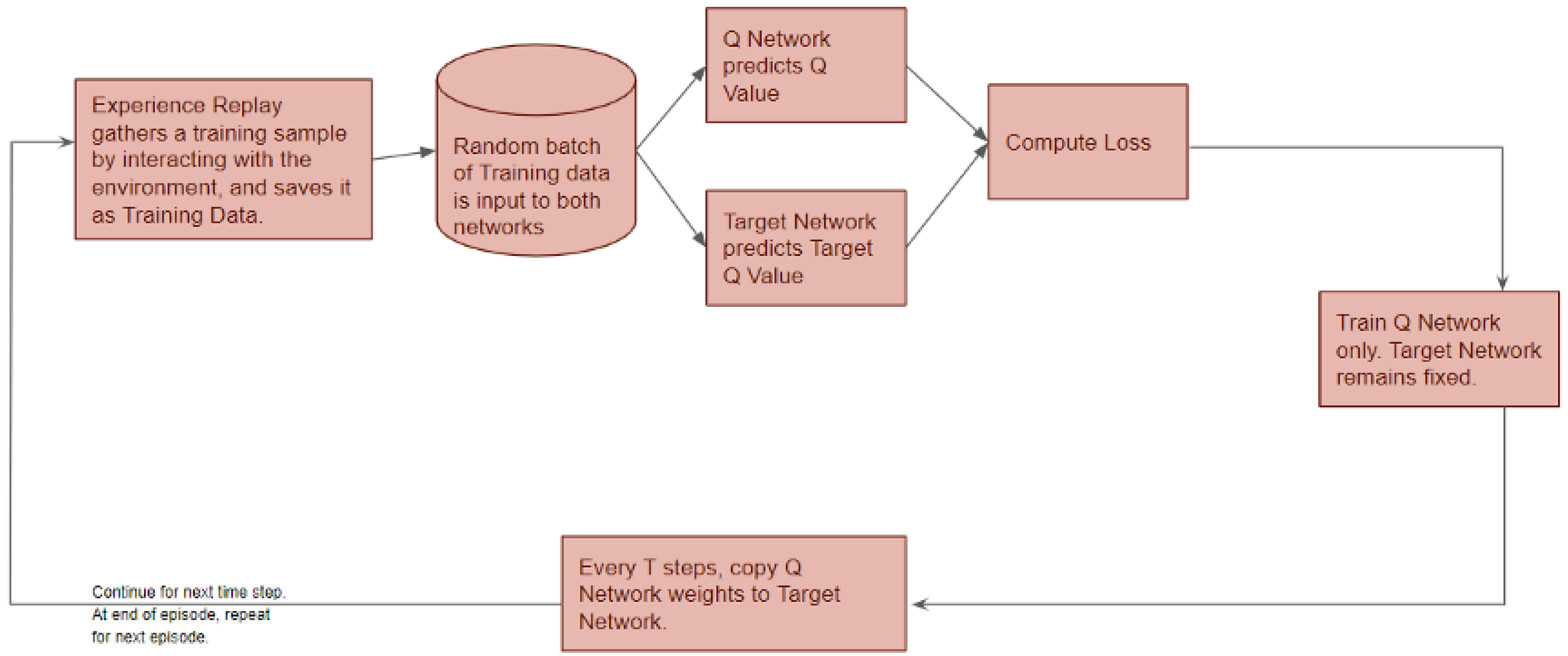
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

DQN Architecture Components



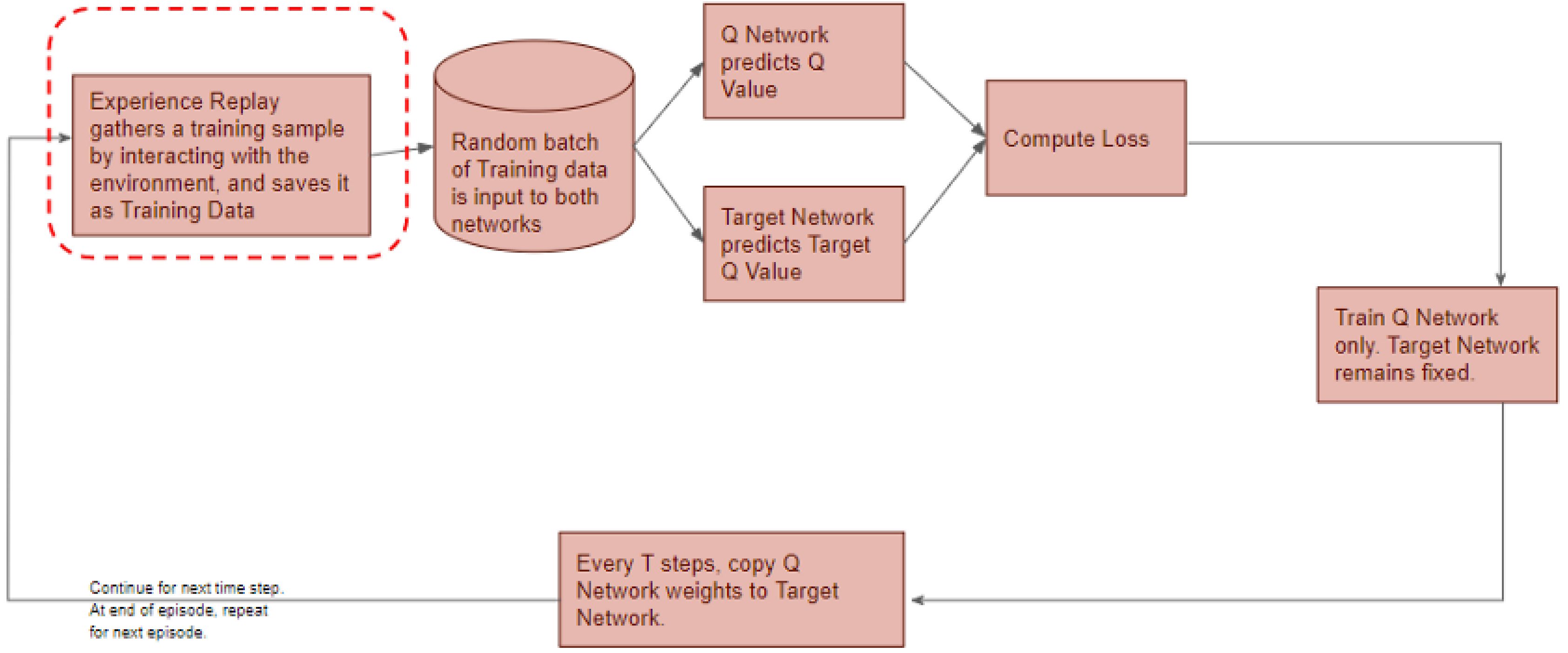
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

High-level DQN Workflow


Experience Replay gathers a training sample by interacting with the environment, and saves it as Training Data.
Random batch of Training data is input to both networks
Q Network predicts Q Value
Target Network predicts Target Q Value
Compute Loss
Train Q Network only. Target Network remains fixed.
Every T steps, copy Q Network weights to Target Network.
Continue for next time step. At end of episode, repeat for next episode.

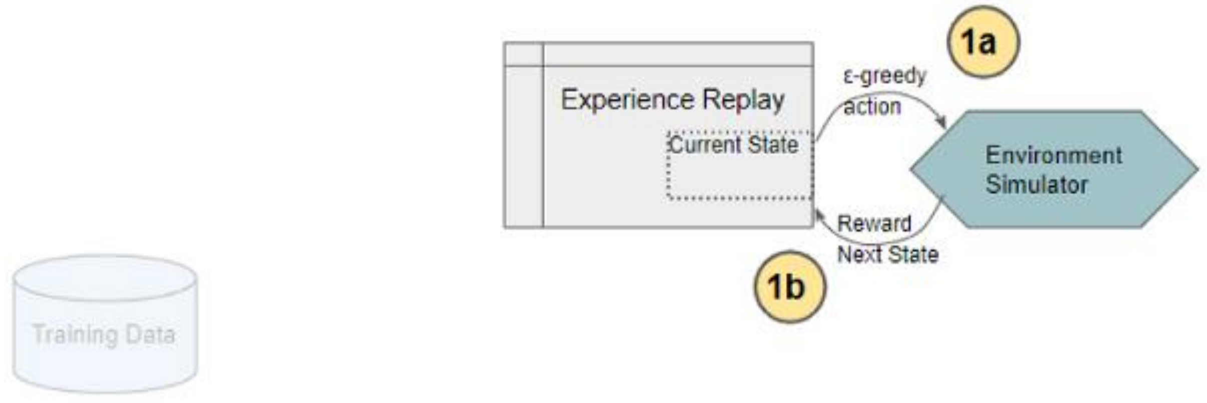
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Gather Training Data


Experience Replay gathers a training sample by interacting with the environment, and saves it as Training Data
Random batch of Training data is input to both networks
Q Network predicts Q Value
Target Network predicts Target Q Value
Compute Loss
Train Q Network only. Target Network remains fixed.
Every T steps, copy Q Network weights to Target Network.
Continue for next time step. At end of episode, repeat for next episode.

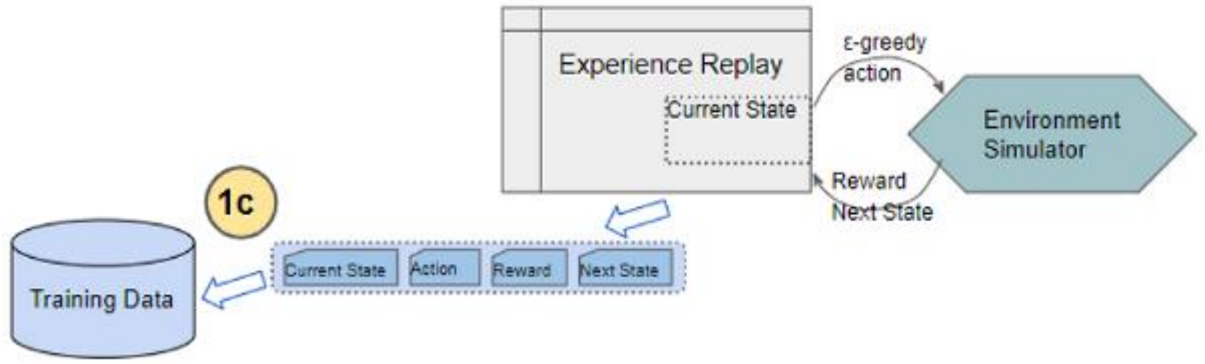
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Experience Replay selects an ε-greedy action from the current state, executes it in the environment, and gets back a reward and the next state.

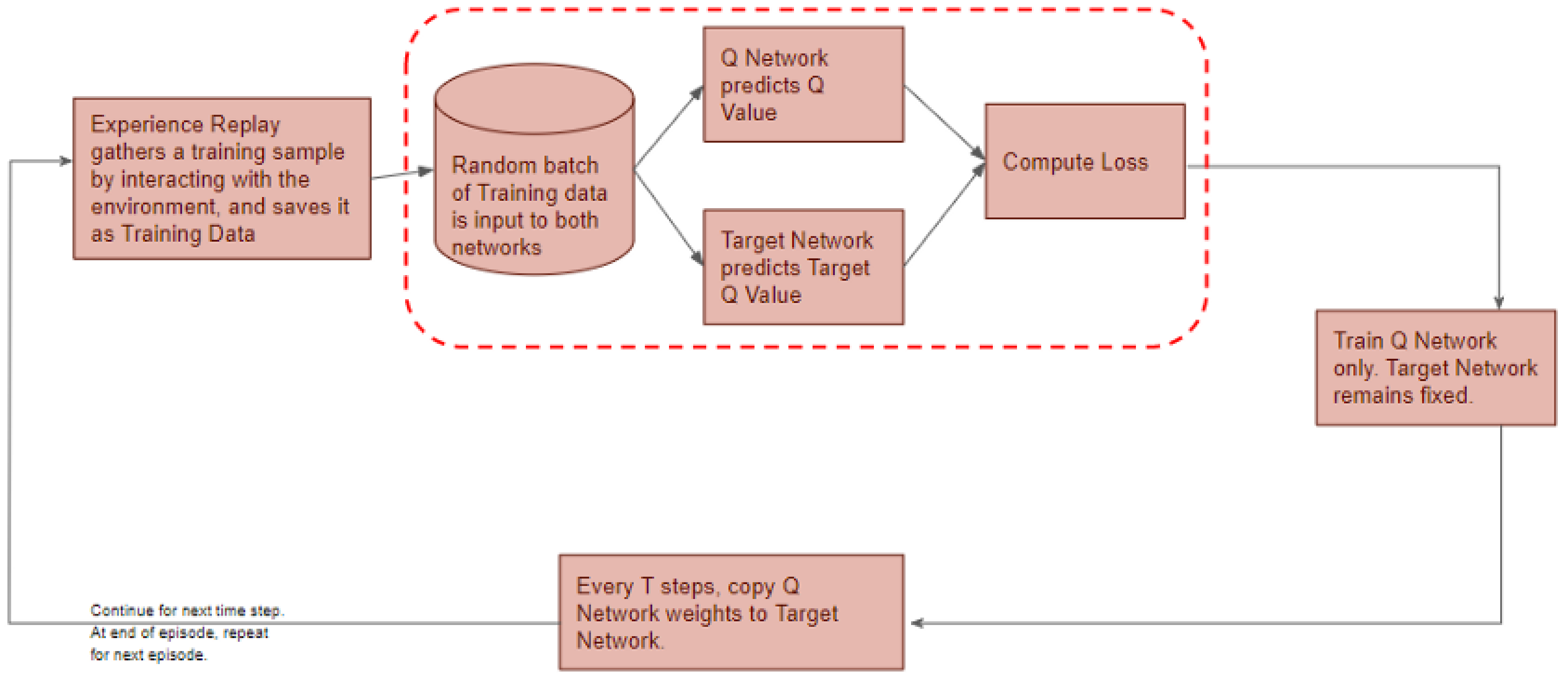


https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

It saves this observation as a sample of training data.

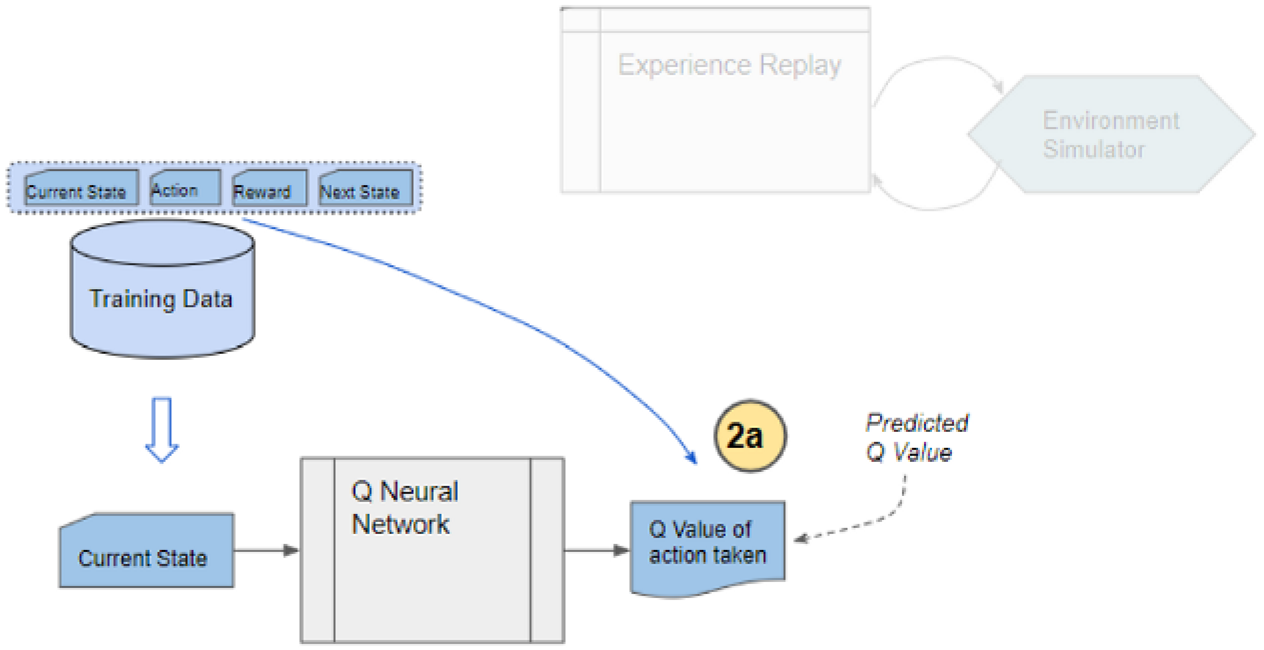


Let’s see the next phase of the flow.


Experience Replay gathers a training sample by interacting with the environment, and saves it as Training Data
Random batch of Training data is input to both networks
Q Network predicts Q Value
Target Network predicts Target Q Value
Compute Loss
Train Q Network only. Target Network remains fixed.
Every T steps, copy Q Network weights to Target Network.
Continue for next time step. At end of episode, repeat for next episode.

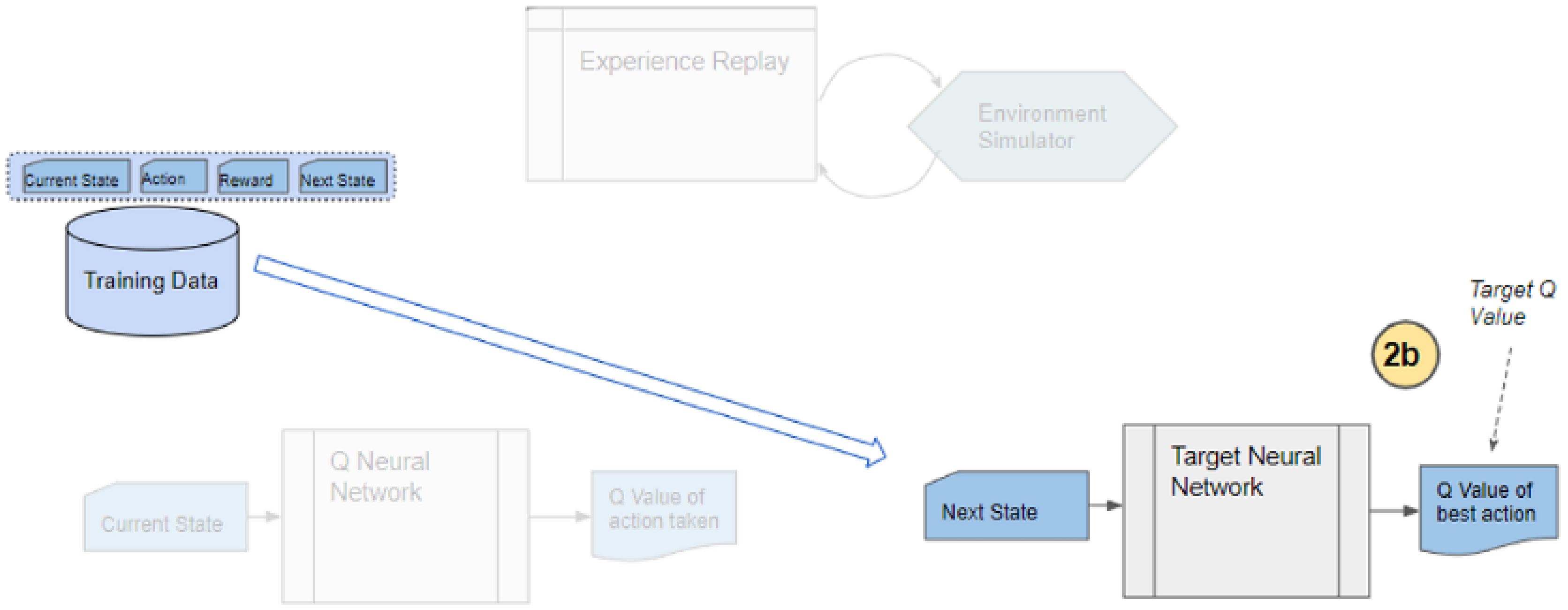
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Q Network predicts Q-value



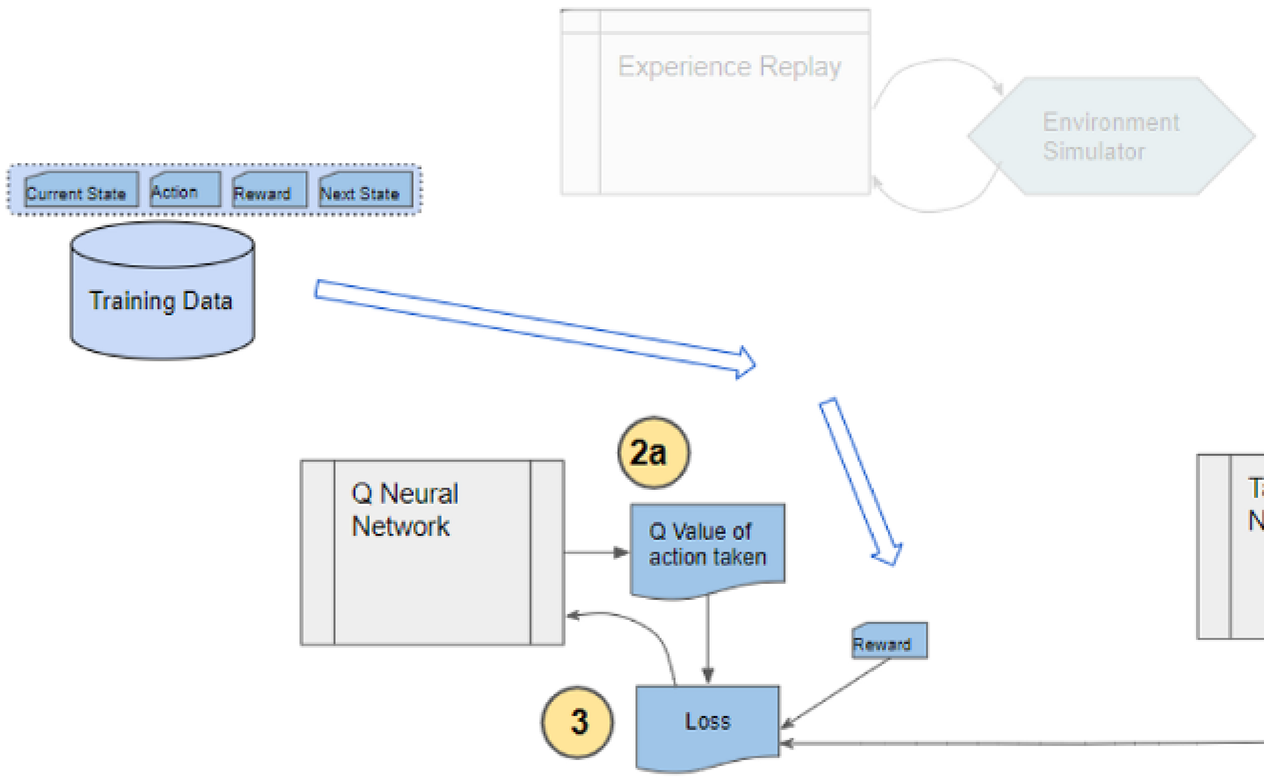
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Target Network predicts Target Q-value



https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Compute Loss and Train Q Network

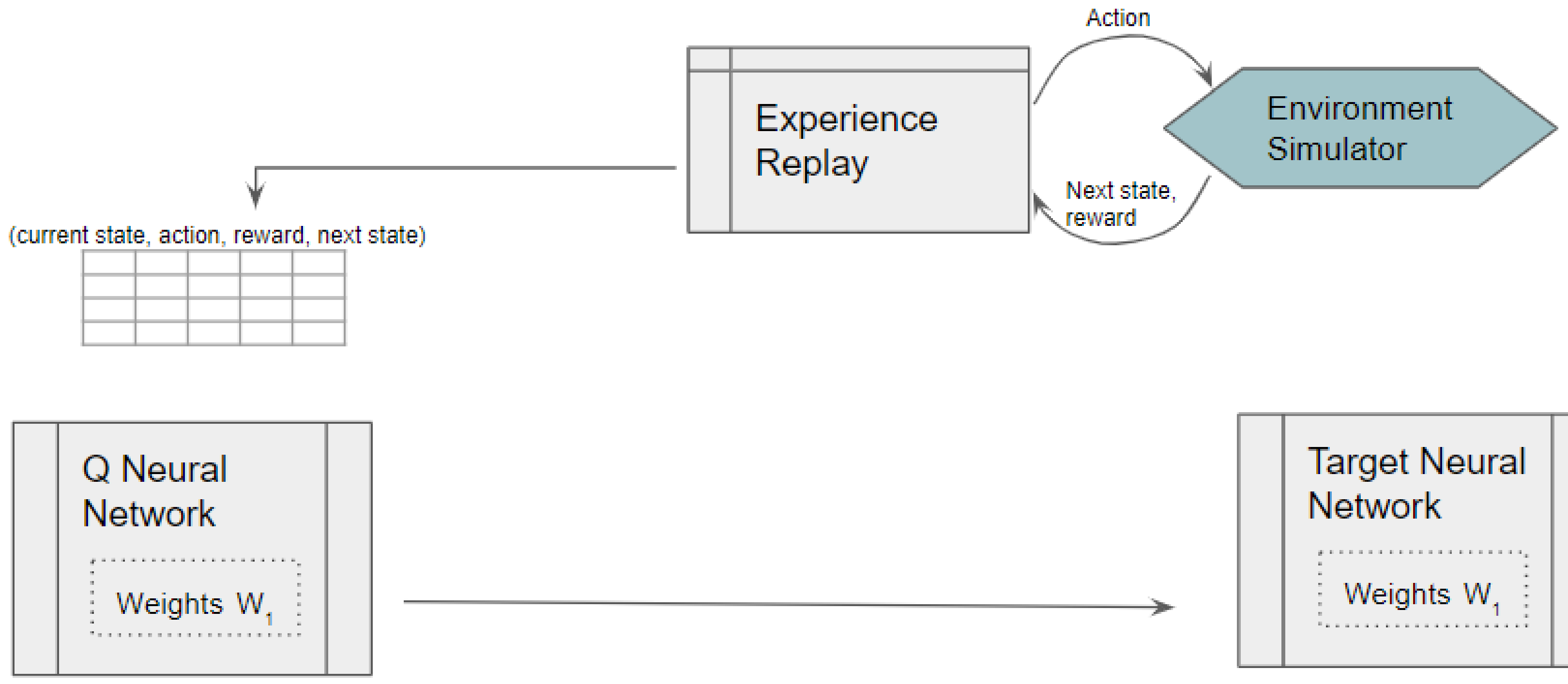


Why do we need Experience Replay?

- All of the actions and observations that the agent has taken from the beginning (limited by the capacity of the memory, of course) are stored.
- Then a batch of samples is randomly selected from this memory.
- This ensures that the batch is ‘shuffled’ and contains enough diversity from older and newer samples (eg. from several regions of the factory floor and under different conditions) to allow the network to learn weights that generalize to all the scenarios that it will be required to handle.

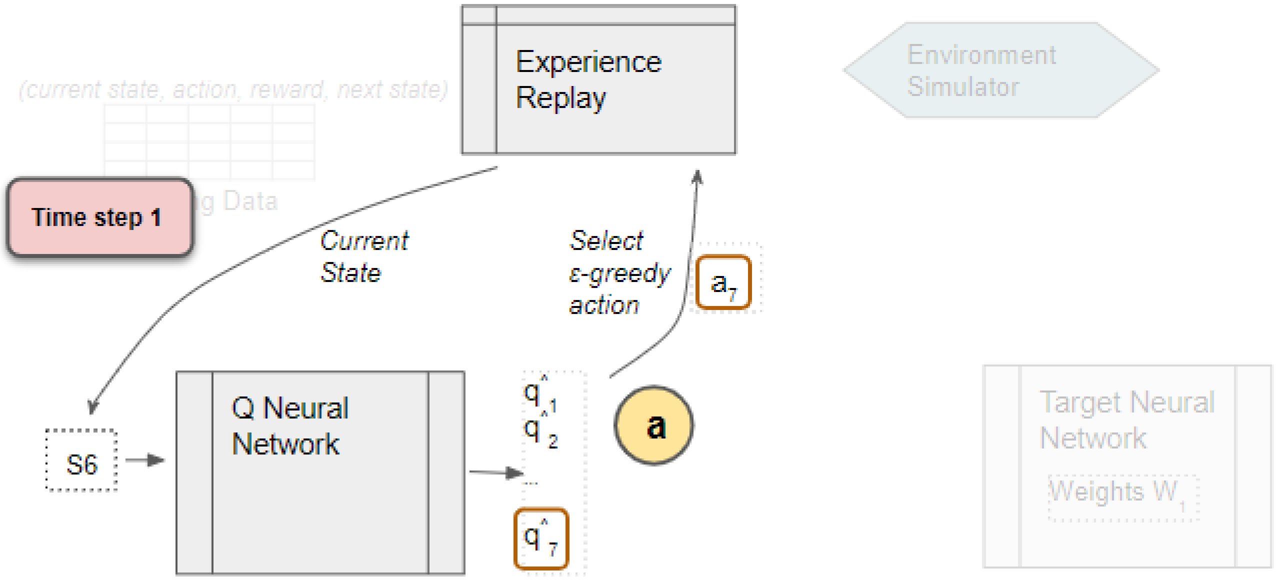
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Overall Flow--1. Initialization



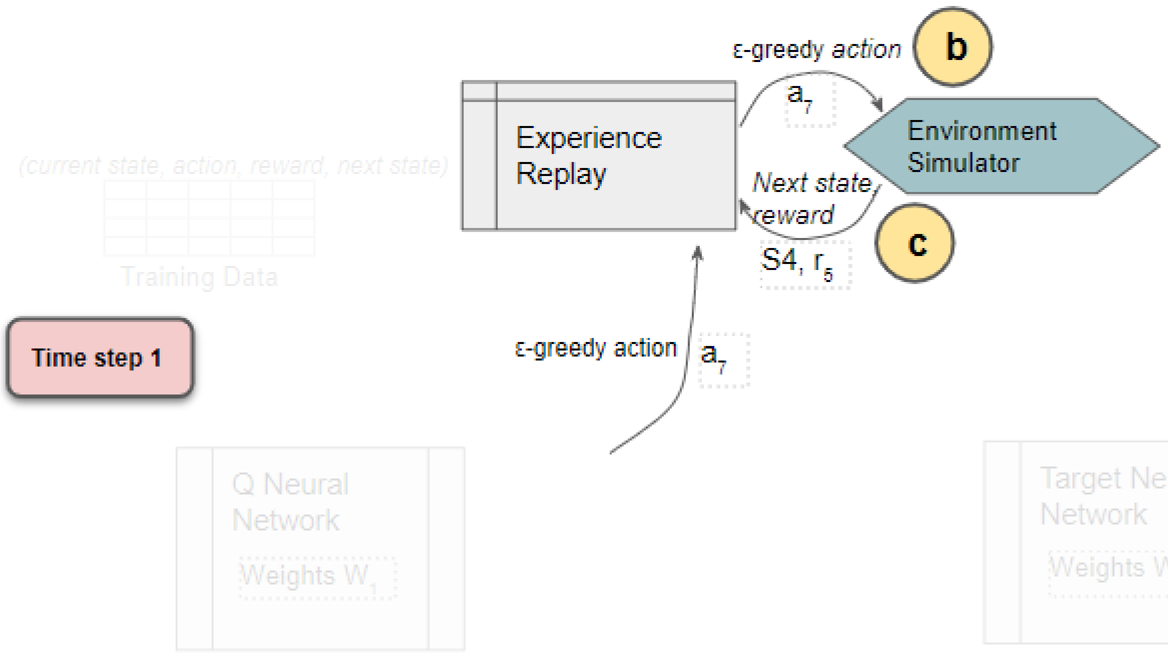
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

2. Experience Replay



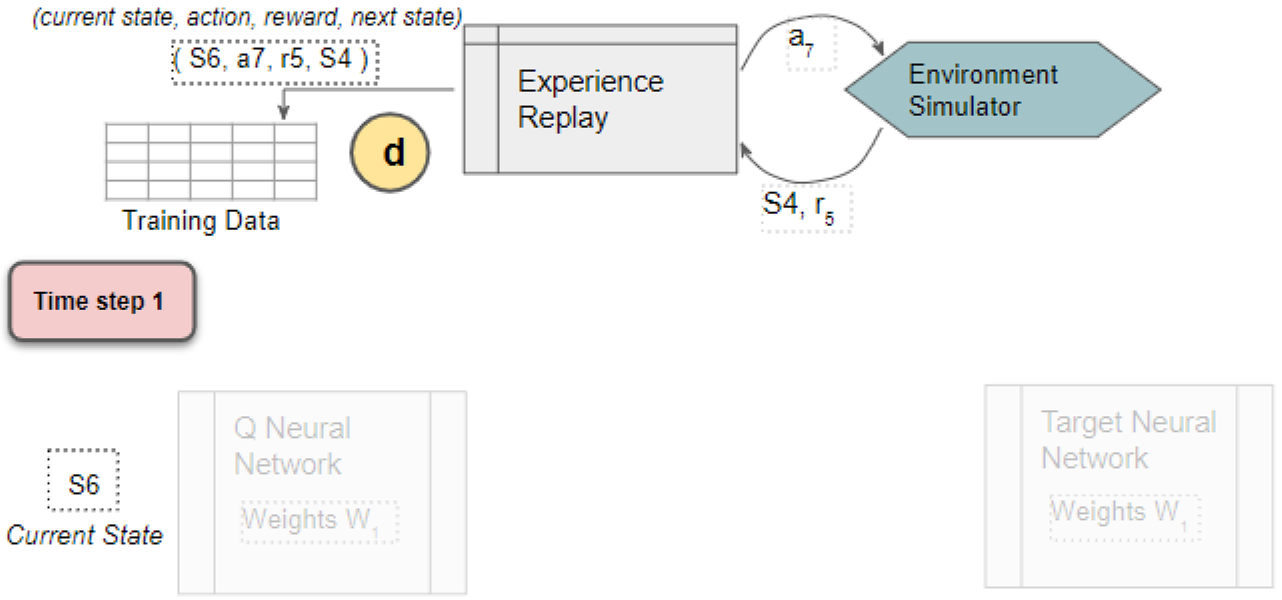
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Experience Replay executes the ε-greedy action and receives the next state and reward



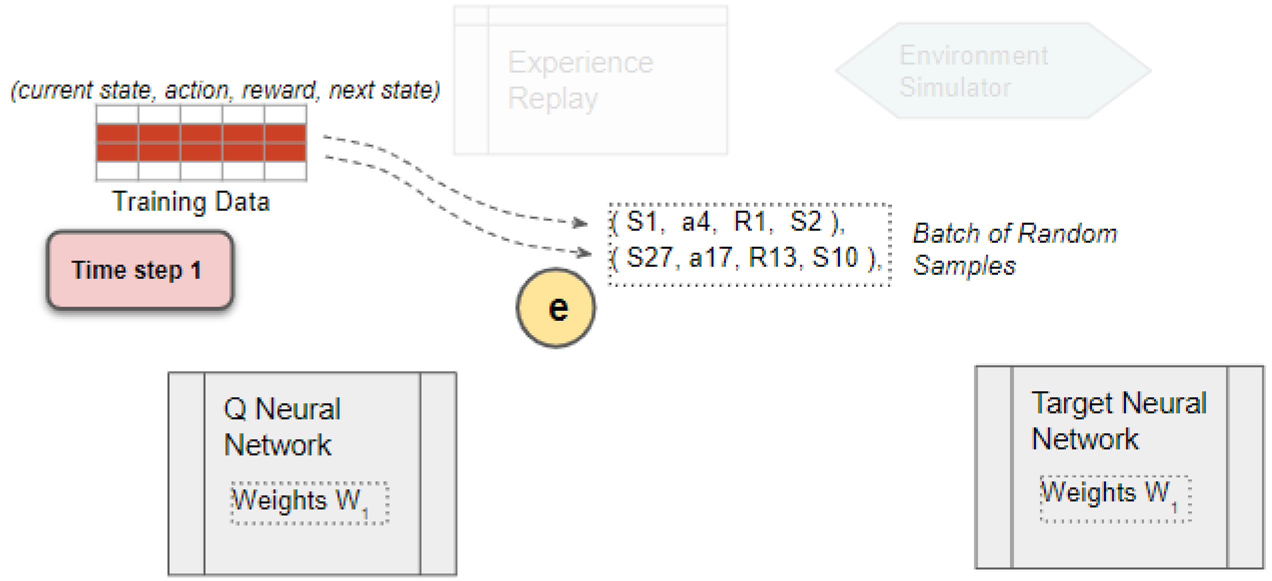
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Each such result is a sample observation which will later be used as training data.



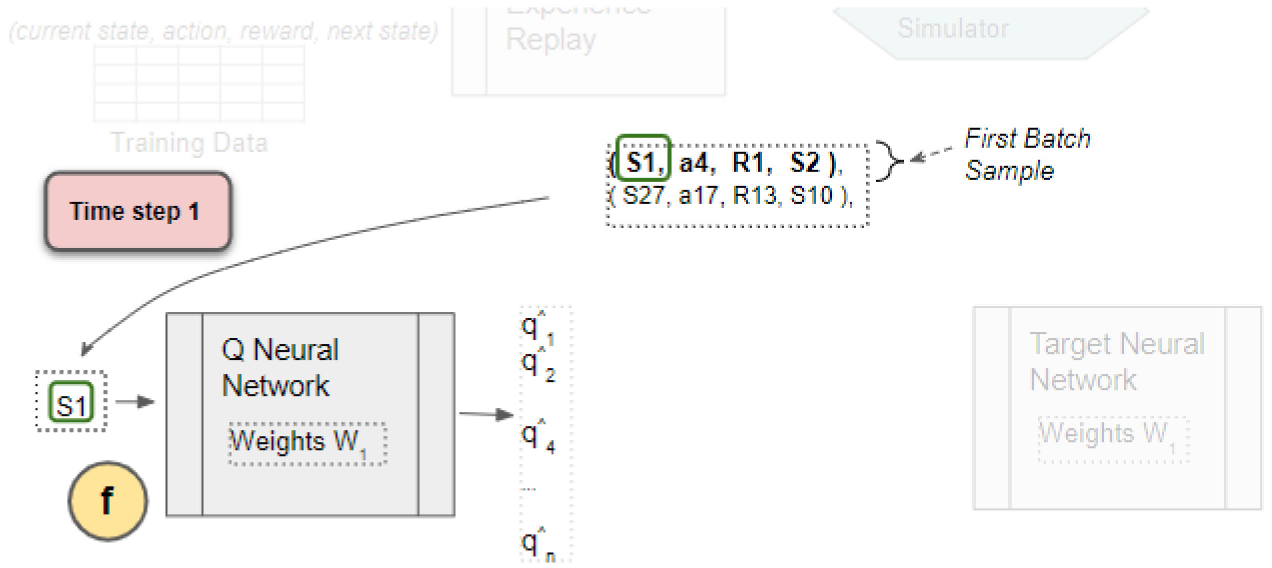
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

3. Select random training batch



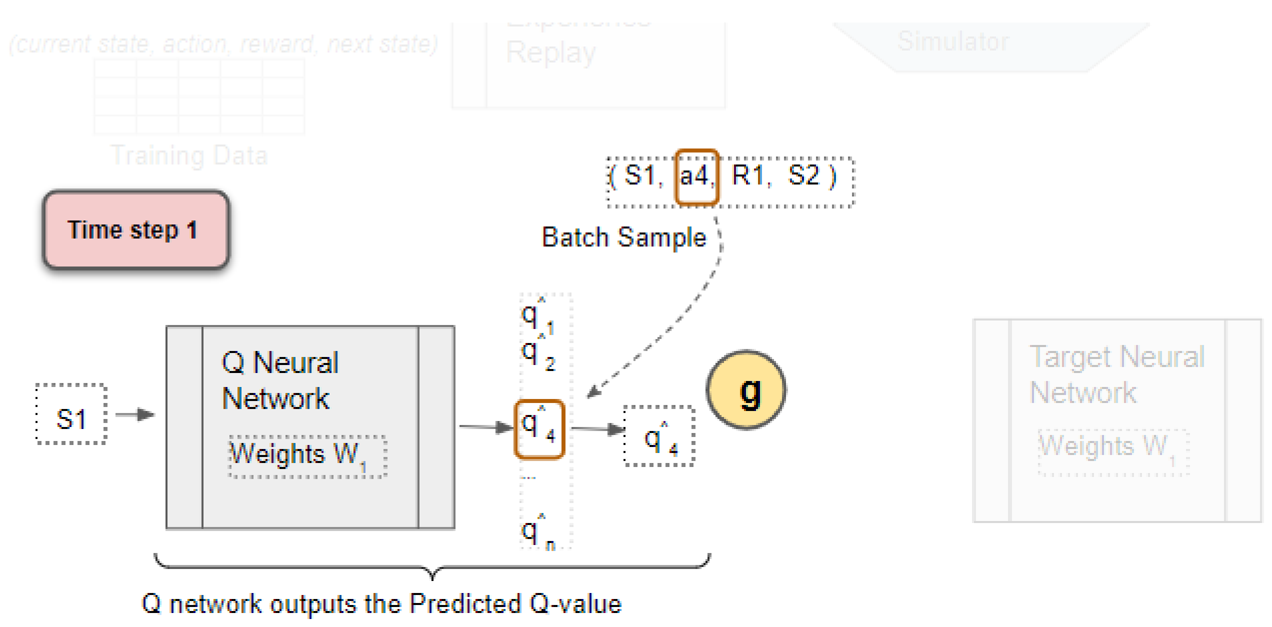
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Use the current state from the sample as input to predict the Q values for all actions



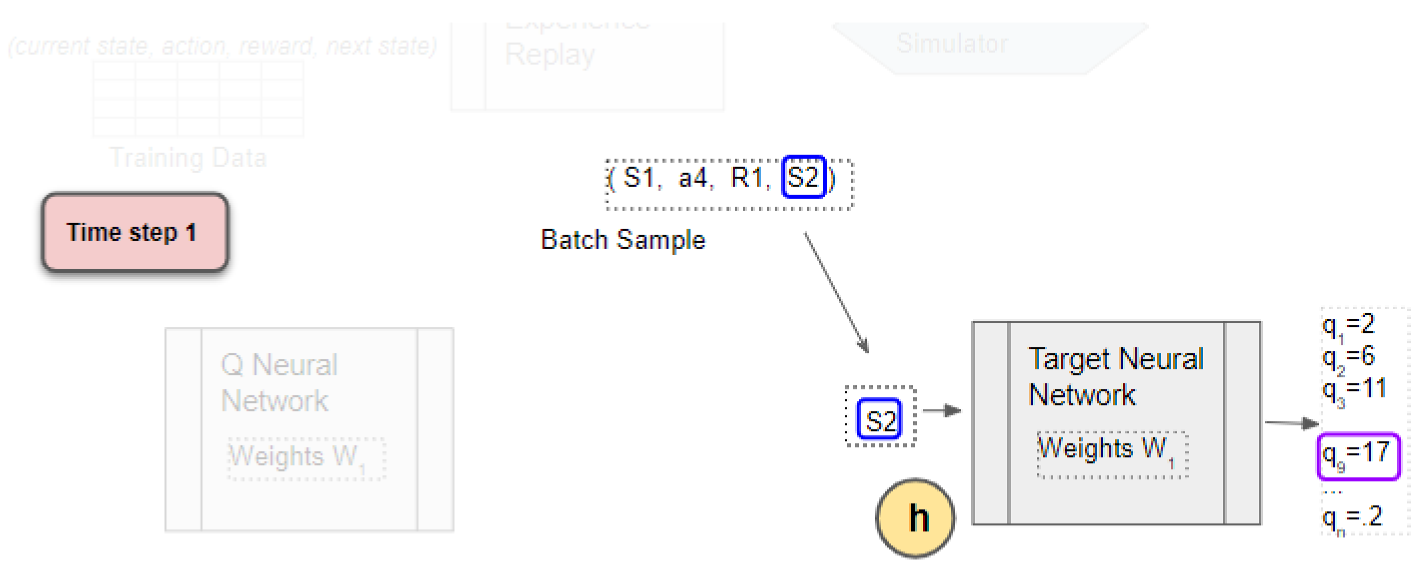
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

4. Select the Predicted Q-value



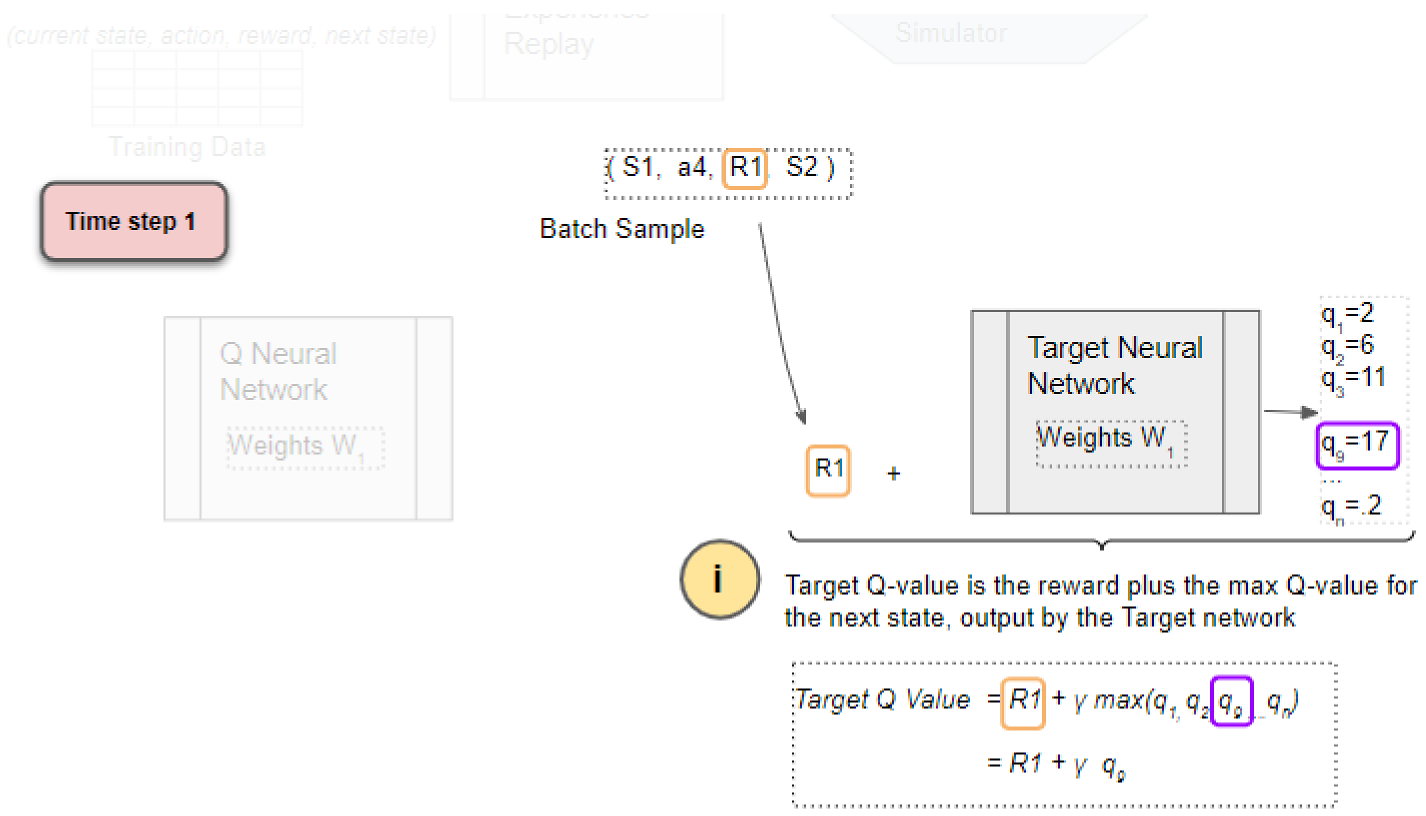
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Use the next state from the sample as input to the Target network



https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

5. Get the Target Q Value

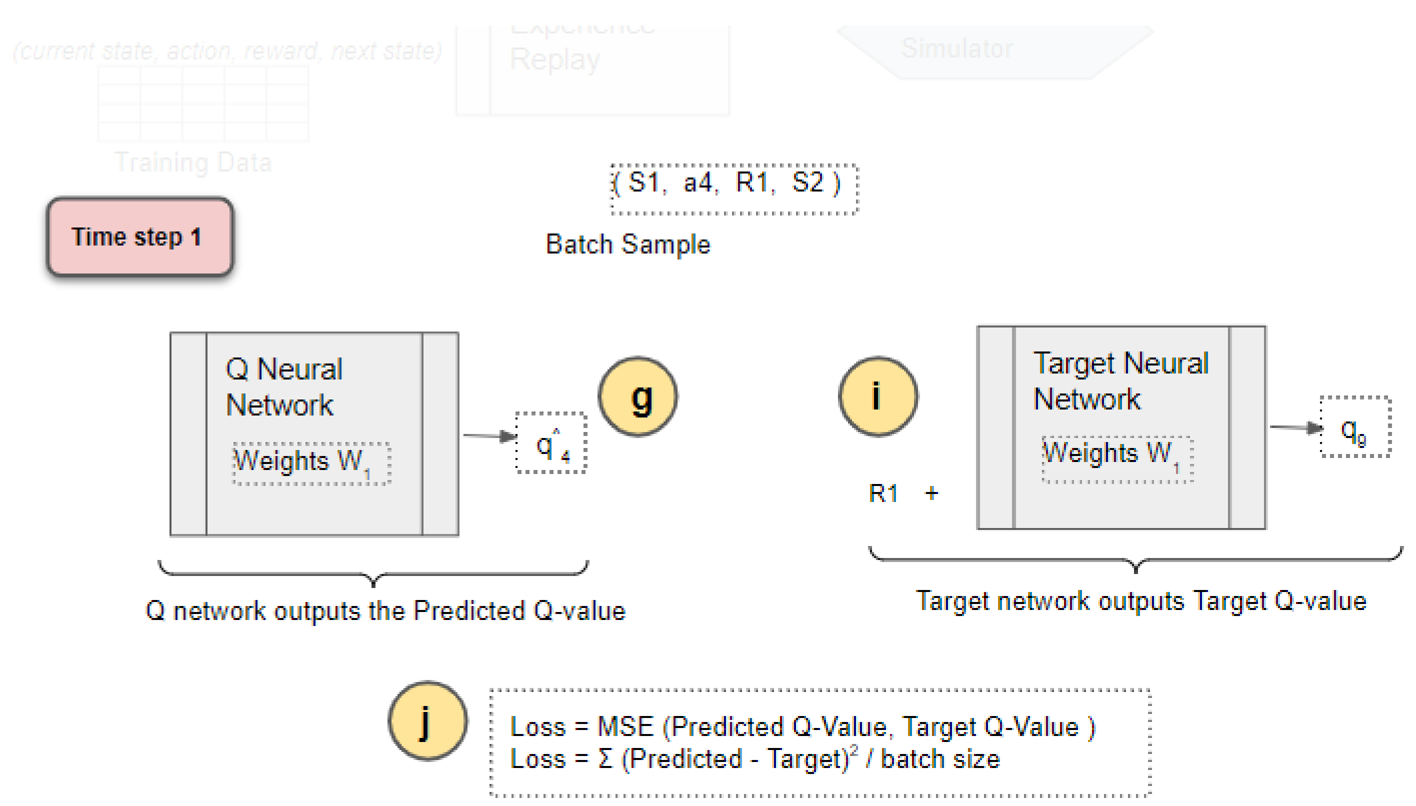


Target Q-value is the reward plus the max Q-value for the next state, output by the Target network

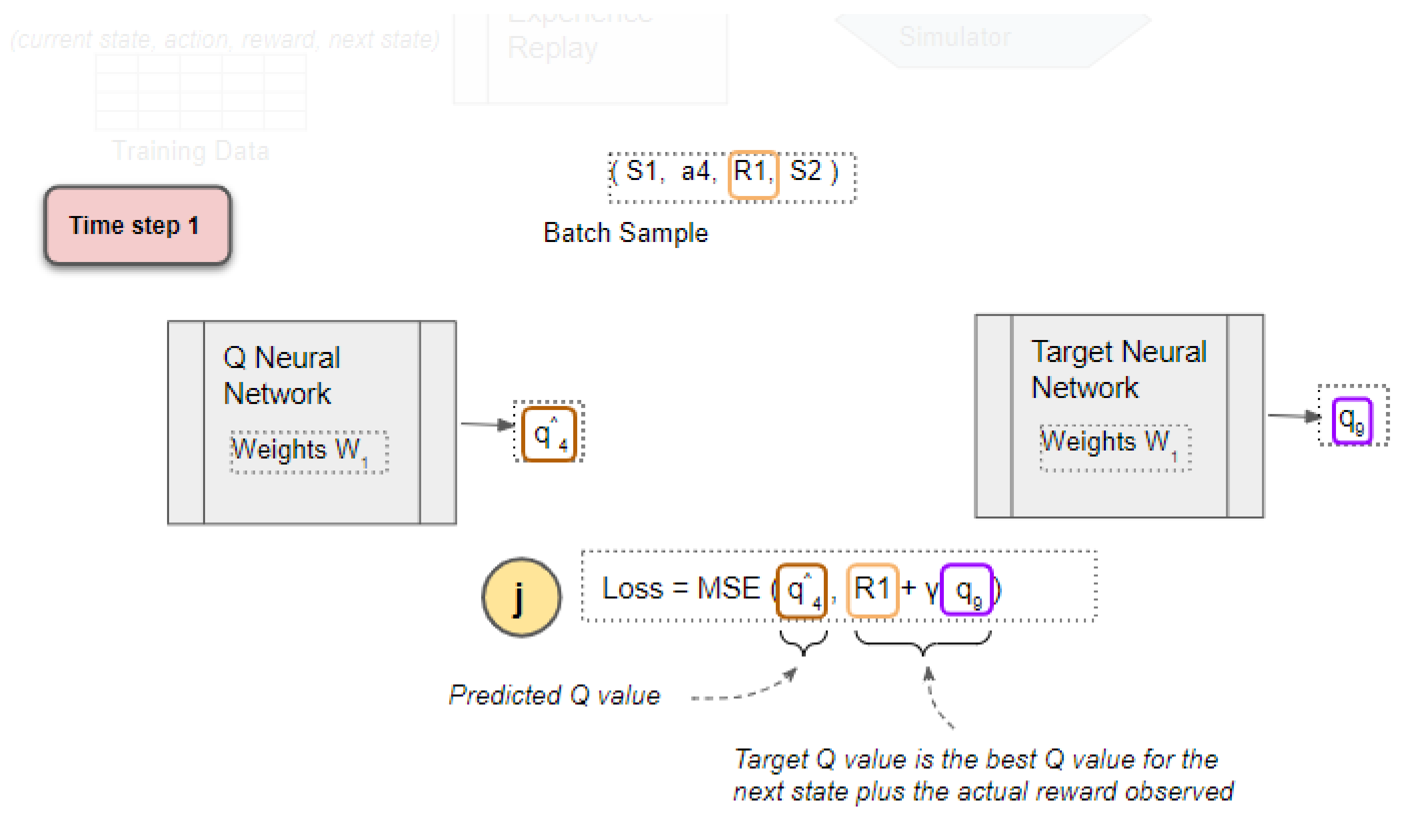
$$\text{Target Q Value} = \boxed{R1} + \gamma \max(q_1, q_2 \boxed{q_9} \ldots q_n)$$
$$= R1 + \gamma\ q_9$$

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

6. Compute Loss

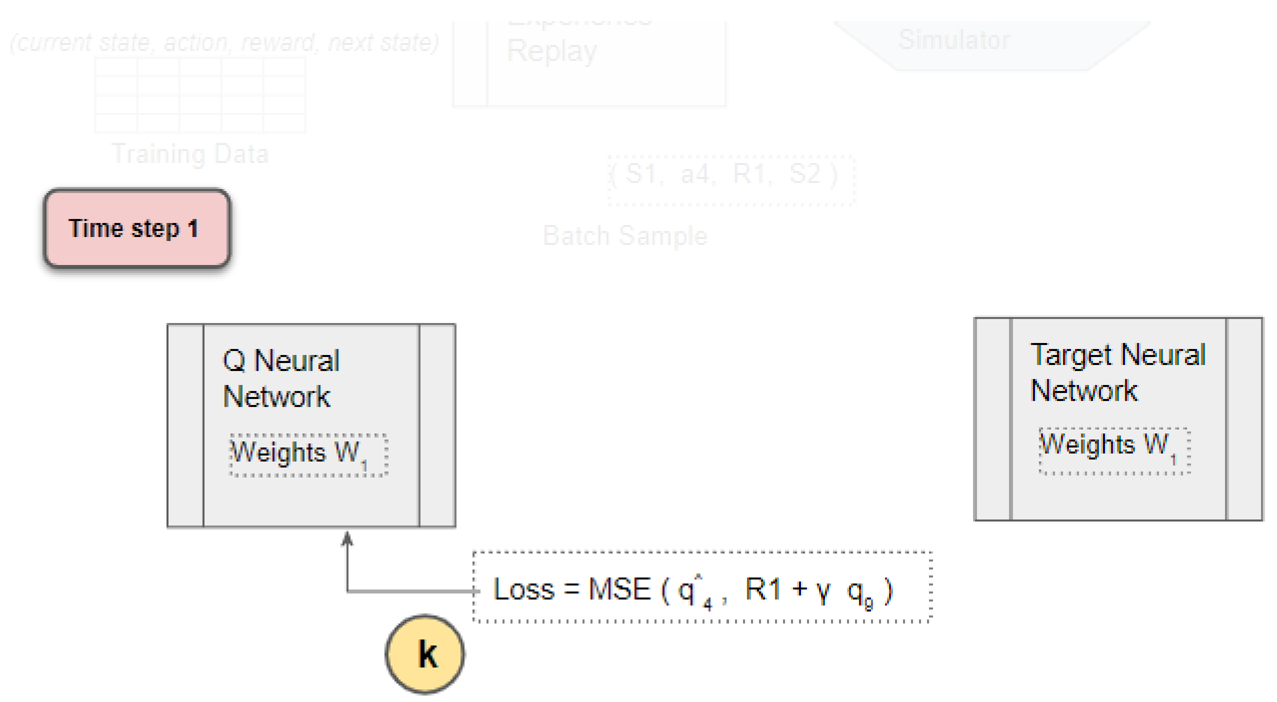

(current state, action, reward, next state)
Experience Replay
Simulator
Training Data
(S1, a4, R1, S2)
Batch Sample
Time step 1
Q Neural Network
Weights W1
q̂4
g
i
R1 +
Target Neural Network
Weights W1
q9
Q network outputs the Predicted Q-value
Target network outputs Target Q-value
j
Loss = MSE (Predicted Q-Value, Target Q-Value)
Loss = Σ (Predicted - Target)² / batch size

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b



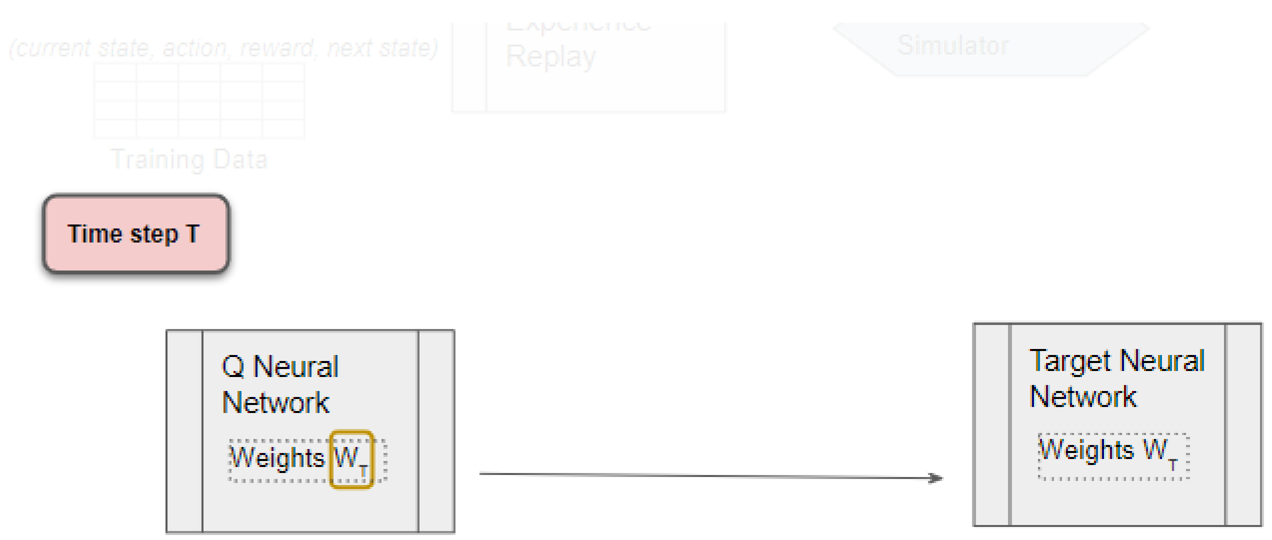
https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

7. Back-propagate Loss to Q-Network



https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

8. After T time-steps, copy Q Network weights to Target Network



https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b

Summary

- The DQN works in a similar way as Q-Learning.
- Since it is a neural network, it uses a Loss function rather than an equation.
- It also uses the Predicted (ie. Current) Q Value, Target Q Value, and observed reward to compute the Loss to train the network, and thus improve its predictions.

https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b