L2D
Deep reinforcement learning

Dr Neythen Treloar

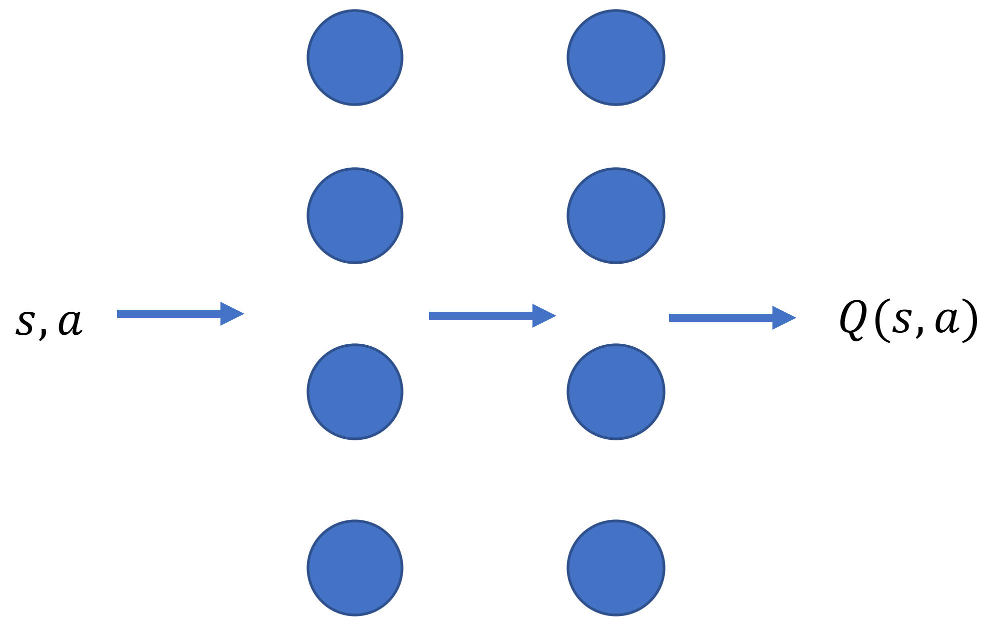
Function approximation in reinforcement learning

- So far, we have restricted ourselves to learning tabular Q-functions
- Function approximation allows us to work in continuous state-action spaces
- And allows generalization between similar states and hence application to much larger problems
- However, function approximation with off-policy temporal difference methods can lead to instability problems ("the deadly triad" pg 264 of Sutton and Barto)
- We will look at using neural networks for Q learning and how to overcome the stability issues

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

Neural networks

- We will use neural networks to learn $Q(s, a)$
- Deep neural networks are universal function approximators meaning they can learn any function with a degree of accuracy dictated by the number of neurons
- Can be trained easily by gradient descent



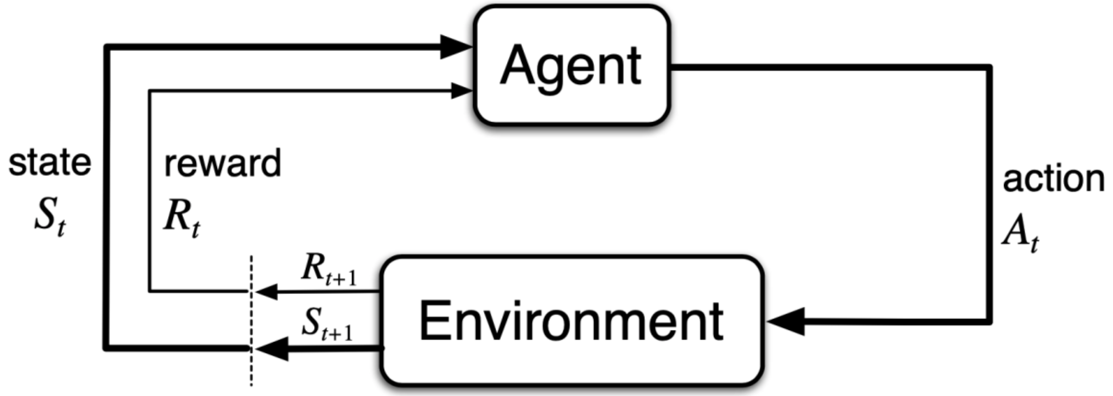
Difficulties

- Q-learning can be unstable when using neural networks
- This is for a few reasons:
    1. Temporal correlations in data mean that samples aren't independent
    2. The data distribution is non-stationary and small changes in the learned Q value can lead to large changes in the policy and hence the distribution of data
    3. The Q values and the target values can be highly correlated with each other
- We will explain these in a bit more detail (see the paper for more detail)

Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015). https://doi.org/10.1038/nature14236

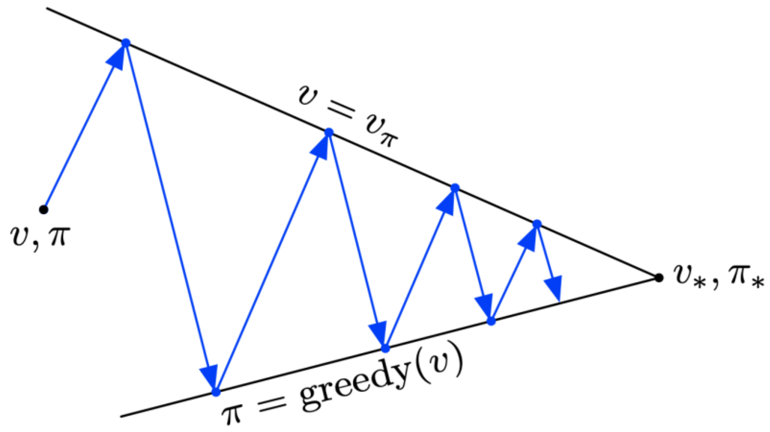
1. Temporal correlations in data mean that samples aren’t independent

- For neural networks, each sample should be independently taken
- The stream of consecutive data points are highly correlated e.g. timestep 2 will be highly dependent on timestep 1
  - $(s_1, a_1, r_1), (s_2, a_2, r_2), \dots, (s_N, a_N, r_N)$



2. The data distribution is non-stationary

- When training on a supervised learning task the data distribution is stationary e.g. for an image classification task the desired result for each image will be constant
- However, in RL the value of a state-action pair depends on the actions that you take after visiting the state-action pair
- The actions are dictated by the policy, but the policy is dependent on the value function
- Changing the value function changes the policy which changes the value function and so on, meaning that the value of a given state-action pair will change throughout training



Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: and introduction *(2nd ed.). The MIT Press.*

3. Correlations between the Q value and the target

- The Q values $Q(s, a)$ are updated according to $r + \gamma \max_a Q(s_{t+1}, a)$.
- $Q(s, a)$ and $\max_a Q(s_{t+1}, a)$ can be highly correlated with each other, especially if s and s' are very similar
- This can lead to runaway instabilities. If $Q(s, a)$ is high, $\max_a Q(s_{t+1}, a)$ will be high if they are correlated. This will further increase $Q(s, a)$ and so on

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\, (\underbrace{r_t + \gamma \max_a Q(s_{t+1}, a)}_{\text{Target}} \; - Q(s_t, a_t))$$

Solutions

1. Experience replay
2. Delayed target updates

SOLUTION 1: experience replay

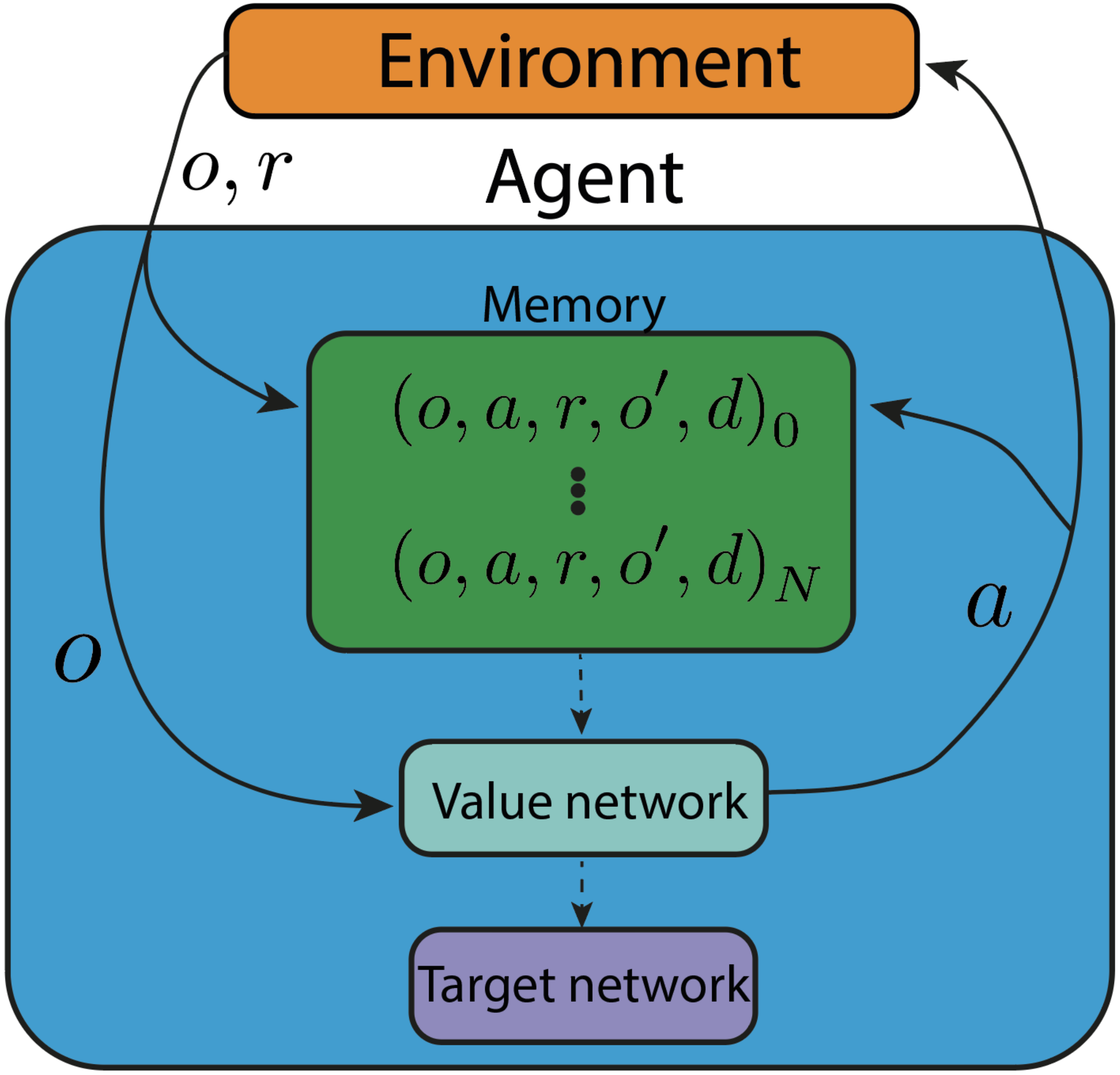
- We put all experience into a memory and randomly sample when we train the agent
  - Memory = $(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_N, a_N, r_N)$
  - Sample = $(s_{21}, a_{21}, r_{21}), (s_1, a_1, r_1), (s_5, a_5, r_5)$
- This removes the temporal correlation between samples and smooths over the changing of the data distribution

SOLUTION 2: delayed target updates

- Only update target values periodically
- We do this by using two neural networks Q and Q_{target}
- We learn values using a modified update, where Q_{target} is used to calculate the Q learning target:
  - $Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left(\underline{r_t + \gamma \max_a Q_{target}(s_{t+1}, a)} - Q(s_t, a_t) \right)$
- The weights of Q_{target} are periodically updated to those of Q with a low frequency
- This reduces the correlation between $Q(s, a)$ and $\max_a Q_{target}(s_{t+1}, a)$

DQN overview

- Incorporating these modifications leads to the deep Q network (DQN) algorithm
- We put all experience into a memory and randomly sample when we train the agent
- Experience is sampled and used to update the value network (using targets generated from the target network)
- The target network is periodically updated to match the value network.



Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015). https://doi.org/10.1038/nature14236

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

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

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Putting all this together results in the Deep Q Network algorithm (DQN).
This paper allowed us to use neural networks with RL
Since then, more sophisticated algorithms have been developed using these principles

Mnih, V., Kavukcuoglu, K., Silver, D. *et al.* Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015). https://doi.org/10.1038/nature14236