Multi-Agent Systems

Introduction to Reinforcement Learning

Model-free Methods

Eric Pauwels (CWI & VU)

December 4, 2021

Outline

Policy gradient algorithms

Policy gradient algorithms

- optimise policy directly,
- NOT via value function (indirectly)

Ingredients:

1. Parametrised policy $\pi_\theta(a \mid s)$ (θ to be determined)
2. Objective function $J(\theta)$ to be maximised
3. Update rule: $\theta_{new} \leftarrow \theta_{old}$, specifically **gradient** ascent:

$$\theta_{new} \leftarrow \theta_{old} + \nabla_\theta J(\theta_{old})$$

Policy gradient: Objective Function

- Trajectory (episodic):

$$\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, s_2, \ldots\ldots, s_{T-1}, a_{T-1}, r_T, s_T\}$$

- Cumulative return along trajectory

$$R(\tau) := \sum_{t=1}^{T} \gamma^{t-1} r_t \qquad \text{or} \qquad R_s(\tau) := \sum_{t=s}^{T} \gamma^{t-s} r_t$$

- Objective function $J(\theta)$: Quantifying policy performance:

$$J(\theta) := \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)]$$

- **Goal**:

$$\max_\theta J(\theta)$$

Policy gradient theorem

- Goal: maximise objective function

$$J(\theta) := \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \int R(\tau)p(\tau \,|\, \theta)\, d\tau$$

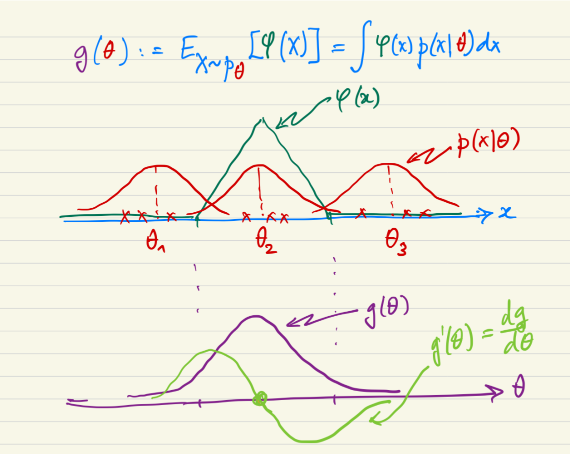
- In abstract terms (to simplify notation):

$$g(\theta) := \mathbb{E}_{X\sim p_\theta}\left[\phi(X)\right] = \int \phi(x)p(x \,|\, \theta)\, dx$$

- Need to compute derivative (to optimise):

$$\frac{d}{d\theta}g(\theta) = \frac{d}{d\theta}\int \phi(x)p(x \,|\, \theta)\, dx = \int \phi(x)\frac{d}{d\theta}\left(p(x \,|\, \theta)\right)\, dx$$

Policy gradient theorem



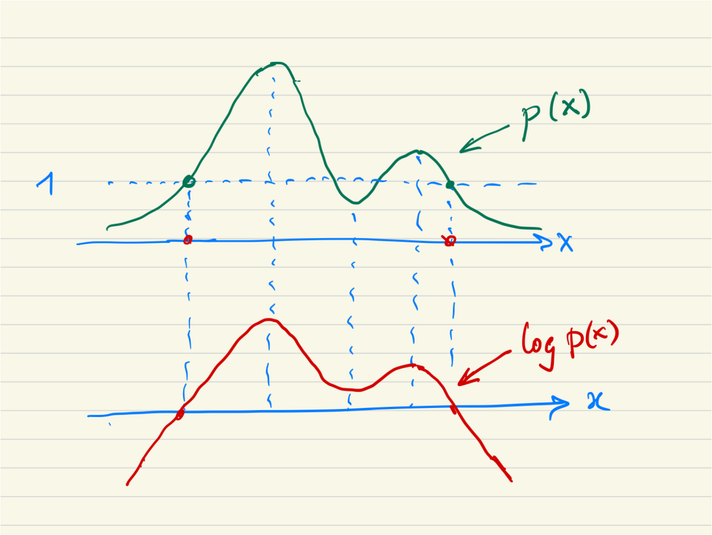
Optimal value $\theta^* = \theta_2$

Policy gradient theorem

$$
\begin{aligned}
\frac{d}{d\theta} g(\theta) &= \int \phi(x) \frac{d}{d\theta}\left(p(x \mid \theta)\right) dx \\
&= \int \phi(x) \left[\frac{\frac{d}{d\theta}\left(p(x \mid \theta)\right)}{p(x \mid \theta)} \right] p(x \mid \theta)\, dx \\
&= \int \phi(x) \left[\frac{\frac{d}{d\theta}\left(p(x \mid \theta)\right)}{p(x \mid \theta)} \right] p(x \mid \theta)\, dx \\
&= \int \phi(x) \left[\frac{d}{d\theta}\left(\log p(x \mid \theta)\right) \right] p(x \mid \theta)\, dx \\
&= \mathbb{E}_{X \sim p_\theta} \left[\phi(X) \frac{d}{d\theta}\left(\log p(x \mid \theta)\right) \right]
\end{aligned}
$$

$$
\boxed{\frac{d}{d\theta} E_{X \sim p_\theta}\left[\phi(X)\right] = \mathbb{E}_{X \sim p_\theta} \left[\phi(X) \frac{d}{d\theta}\left(\log p(X \mid \theta)\right) \right]}
$$

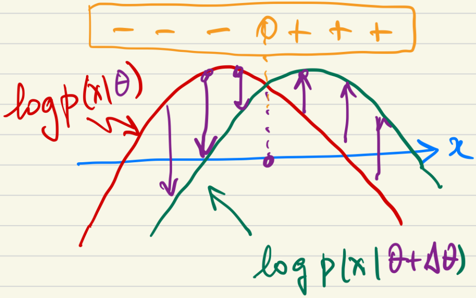
$p(x)$ and $\log p(x)$ have same local exptremes



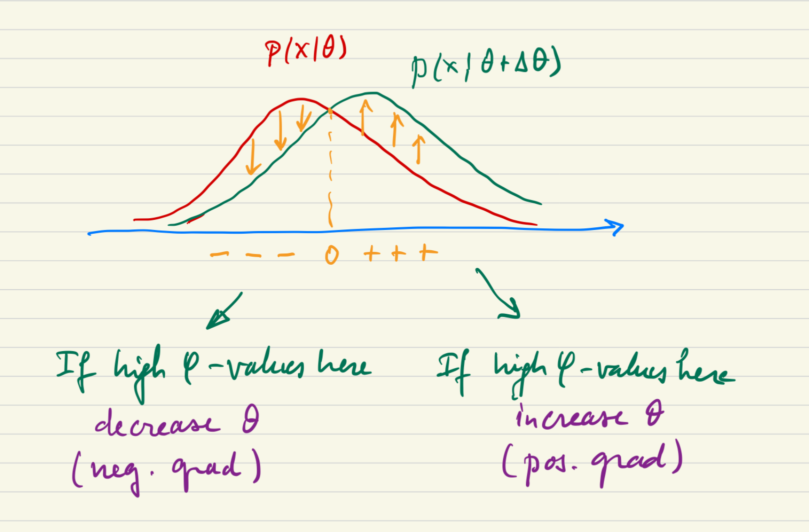
Policy gradient theorem

$$\nabla_\theta g(\theta) = \mathbb{E}_{X\sim\pi_\theta}\left[\varphi(X)\,\underbrace{\nabla_\theta \log p(X;\theta)}\right]$$

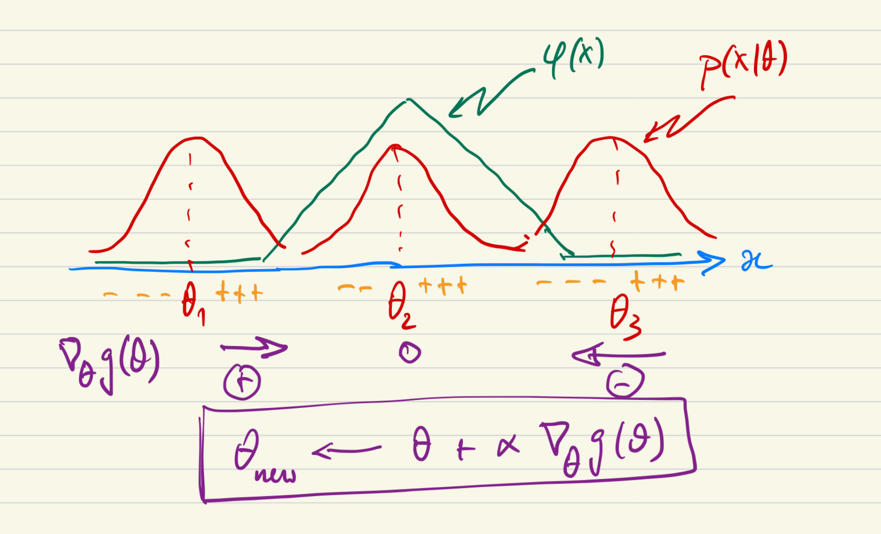
at given X
how does $\log p(X|\theta)$
change if θ increases
i.e. $\theta \to \theta+\Delta\theta$



Policy gradient theorem



Policy gradient theorem



Policy gradient theorem

General result

$$\nabla_\theta E_{X \sim p_\theta}\left[\phi(X)\right] = \mathbb{E}_{X \sim p_\theta}\left[\phi(X) \nabla_\theta \log p(X \mid \theta)\right]$$

Gradient Policy Theorem

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau) \nabla_\theta \log p(\tau \mid \theta)\right]$$

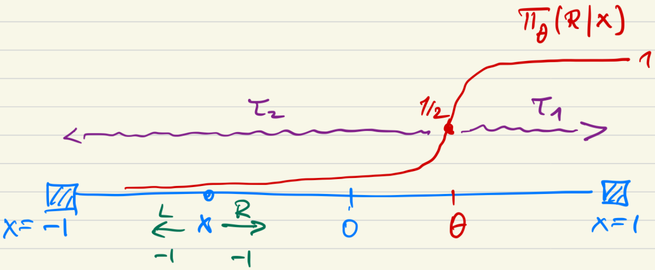
Policy gradient theorem

$$
\begin{aligned}
p(\tau \mid \theta) &= \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t) \\
\log p(\tau \mid \theta) &= \sum_{t \geq 0} [\log p(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t)] \\
\nabla_\theta \log p(\tau \mid \theta) &= \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t \mid s_t)
\end{aligned}
$$

Gradient Policy Theorem

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau) \nabla_\theta \log p(\tau \mid \theta)] \\
&= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right]
\end{aligned}
$$

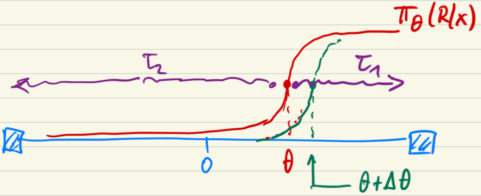
Policy gradient: Example(1)



$$R(\tau_1) = -(1-\theta)$$
$$R(\tau_2) = -(1+\theta)$$
$$\boxed{R(\tau_2) < R(\tau_1)}$$

$$p(\tau_1, \theta) = 1/2$$
$$p(\tau_2|\theta) = 1/2$$

Policy gradient: Example(2)



$$p(\tau_1|\theta+\Delta\theta) < p(\tau_1|\theta) \longrightarrow \nabla_\theta \log p(\tau_1|\theta) < 0 \quad (=-\delta)$$

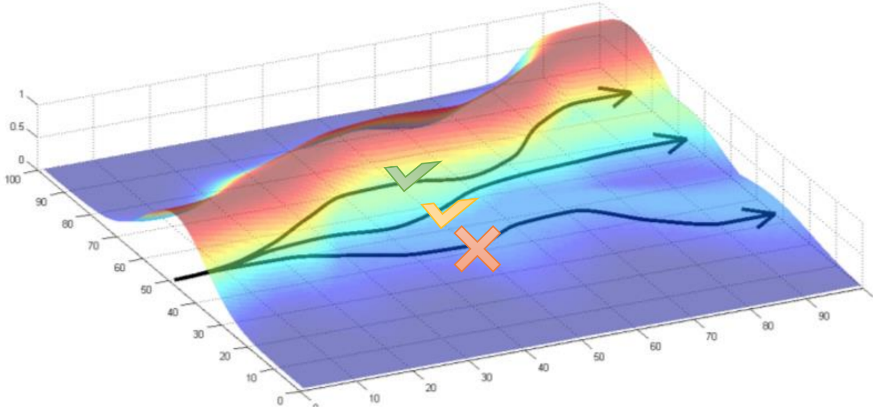
$$p(\tau_2|\theta+\Delta\theta) > p(\tau_2|\theta) \longrightarrow \nabla_\theta \log p(\tau_2|\theta) > 0 \quad (=+\delta)$$

$$\nabla_\theta J \simeq R(\tau_1)\cdot\underbrace{\nabla_\theta \log p(\tau_1|\theta)}_{-\delta} + R(\tau_2)\underbrace{\nabla_\theta \log p(\tau_2|\theta)}_{+\delta}$$

$$\simeq \delta\left[-R(\tau_1) + R(\tau_2)\right]$$

$$\simeq \delta\left[|R(\tau_1)| - |R(\tau_2)|\right] < 0 \implies \theta\downarrow$$

Policy gradient illustration



REINFORCE algorithm

Example of policy gradient algo

1. Initialise learning rate α
2. Initialise parameter θ of policy π_θ
3. **for** episode $= 1 \ldots$ NR_EPISODES:
4. Sample trajectory $\tau = \{s_0, a_0, r_1, s_1, \ldots, r_T, s_T\}$
5. Set $\nabla_\theta J(\theta) = 0$
6. # add gradient contributions along trajectory
7. **for** $t = 0, 1, \ldots, T$:
8. $R_t(\tau) = \sum_{t=s}^{T} \gamma^{t-s} r_t,$
9. $\nabla_\theta J(\theta) = \nabla_\theta J(\theta) + R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
10. # update policy parameter
11. $\theta = \theta + \alpha \nabla_\theta J(\theta)$

Improving REINFORCE algorithm

- Policy gradient estimate has high variance
  (trajectories might be very different!)

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} R_t(\tau) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

- Variance reduction by introducing action-independent baseline:

$$\nabla_\theta J(\theta) \approx \sum_{t=0}^{T} (R_t(\tau) - b(s_t)) \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

- Example: Actor-Critic algorithm:

$$R_t(\tau) = q_{\pi_\theta}(s_t, a_t) \qquad \text{and} \qquad b(s_t) = v_{\pi_\theta}(s_t)$$

Gaussian Policy as example of parametrised policy

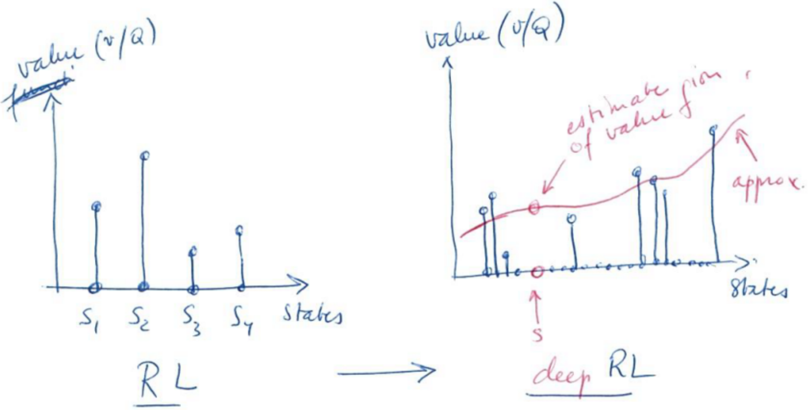
- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu(s) = \phi(s)^\top \theta$
- Variance may be fixed σ^2, or can also parametrised
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

Outline

Deep Reinforcement Learning (dRL)

- What if number of states or actions is **huge**!
- Use function approximation (i.e. generalisation) to estimate value functions in unseen states;



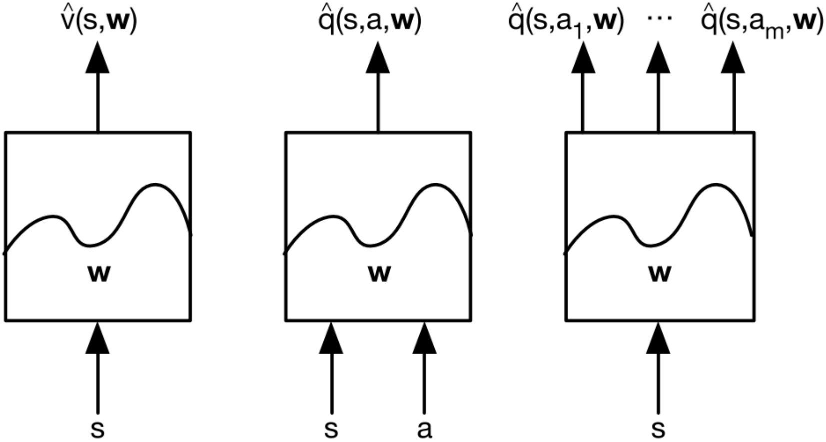
Value Function Approximation for Large RL Problems

- So far we have represented value function by a *lookup table*
  - Every state s has an entry $V(s)$
  - Or every state-action pair s, a has an entry $Q(s, a)$
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with *function approximation*

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\text{or } \hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

  - *Generalise* from seen states to unseen states
  - *Update* parameter $\mathbf{w}$ using MC or TD learning

Types of Value Function Approximation



Fitting dNN using Stochastic Gradient Descent

- Goal: find parameter vector $\mathbf{w}$ minimising mean-squared error between approximate value fn $\hat{v}(s, \mathbf{w})$ and true value fn $v_\pi(s)$

$$J(\mathbf{w}) = \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w}))^2 \right]$$

- Gradient descent finds a local minimum

$$\begin{aligned} \Delta \mathbf{w} &= -\frac{1}{2} \alpha \nabla_\mathbf{w} J(\mathbf{w}) \\ &= \alpha \mathbb{E}_\pi \left[(v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_\mathbf{w} \hat{v}(S, \mathbf{w}) \right] \end{aligned}$$

- Stochastic gradient descent *samples* the gradient

$$\Delta \mathbf{w} = \alpha (v_\pi(S) - \hat{v}(S, \mathbf{w})) \nabla_\mathbf{w} \hat{v}(S, \mathbf{w})$$

- Expected update is equal to full gradient update

Incremental Prediction Algorithms

- Have assumed true value function $v_\pi(s)$ given by supervisor
- But in RL there is no supervisor, only rewards
- In practice, we substitute a *target* for $v_\pi(s)$
  - For MC, the target is the return G_t

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

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

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w}))\nabla_{\mathbf{w}}\hat{v}(S_t, \mathbf{w})$$

DQN Overview

- Deep RL version of Q-learning:
  use experience to learn optimal $q^*(s, a)$
  - Construct dNN approximation $\hat{q}_\theta$ for q^*
  - Use experiences to train dNN $\hat{q}_\theta$
- Off-policy algorithm
- Improve sample efficiency by storing experiences in experience replay buffer
- Target network improves stability

Experience Replay Memory (ERM)

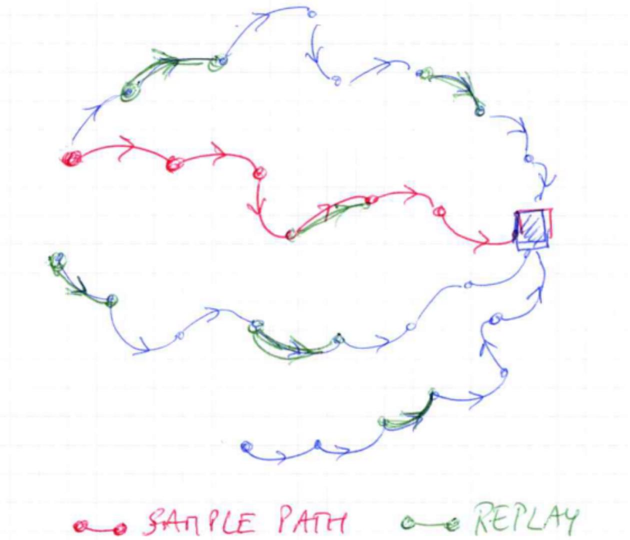
Motivation

- Recall: experience $= \{s, a, r, s'\}$
- Information-rich experiences should be used multiple times
- Experiences along trajectory are highly correlated;

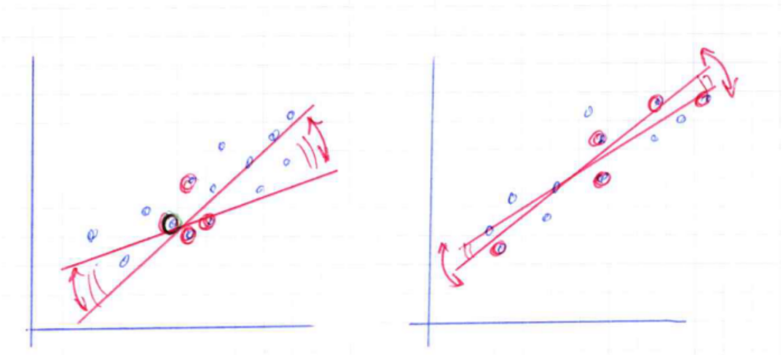
Experience Replay Memory (ERM)

- Storing: ERM stores K most recent experiences;
  - Typically: $K = 10^6$
  - Stalest data are overwritten (reflects learning)
- Training: sample batch (uniformly or prioritized) from ERM

Sample path versus experience replay



Correlated vs. uncorrelated



DQN Algorithm

1. **for** $m = 1 \ldots$ MAX_STEPS: # loop over training steps;
2. Use current ϵ-greedy policy to roll-out trajectories and store experiences $\{s, a, r, s'\}$ in ERM
3. **for** $b = 1 \ldots, B$: # $B =$ nr BATCHES per training step
4. *Sample batch of size U from ERM*
5. **for** $u = 1 \ldots U$: # $U =$ number of updates per batch
6. **for** $i = 1 \ldots N$: # $N =$ batch size
7. $y_i = r_i + \gamma \max_{a'_i} q_\theta(s'_i, a'_i)$ # Target q-values
8. $L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - q_\theta(s_i, a_i))^2$ # compute loss
9. $\theta = \theta - \alpha \nabla_\theta L(\theta)$ # Update dNN parameter θ
10. Update ϵ, α

Improving DQN

- **Target Networks**
  - Compute target values wrt. lagged parameter value φ of θ:

$$y_i = r_i + \gamma \max_{a_i'} q_\varphi(s_i', a_i')$$

  - Update $\varphi \leftarrow \theta$ from time to time;
  - Fixes target, improves stability

- **PER: Prioritised Experience Replay**
  - Not all experiences are equally informative to agent
  - Therefore, sampling uniformly from ERM is inefficient
  - Give higher priority to most "surprising" experiences (as measured by TD error).

DQN: Experience replay in Deep Q-Networks

- DQN uses experience replay and fixed Q-targets
- Store transition (s_t,a_t,r_{t+1},s_{t+1}) in replay memory D
- Sample random mini-batch of transitions (s,a,r,s′) from D
- Compute Q-learning targets w.r.t. old, fixed parameters w−
- Optimize MSE between Q-network and Q-learning targets

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

DQN: Deep Q-Network, *(Mnih et al, Nature 2015)*

- **Loss function**

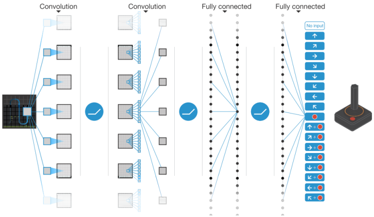
$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

- **Gradient**

$$\nabla_{\theta_i} L(\theta_i) \quad = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

DQN: Deep Q-Network, *(Mnih et al, Nature 2015)*

- Single RL architecture achieved (super)human performance on suite of classic ATARI games;



- **Key features and improvements:**
  - Function approximation based on deep NN;
  - Experience replay (data re-use and de-correlation)
  - Use of target network for stabilisation