# Human-Level Control Through Deep RL

## Mnih et al

**Simar Kareer**

# Motivation

- Solving “real life” tasks
- Replicate human learning
- Perception embedding
- Generalization between tasks

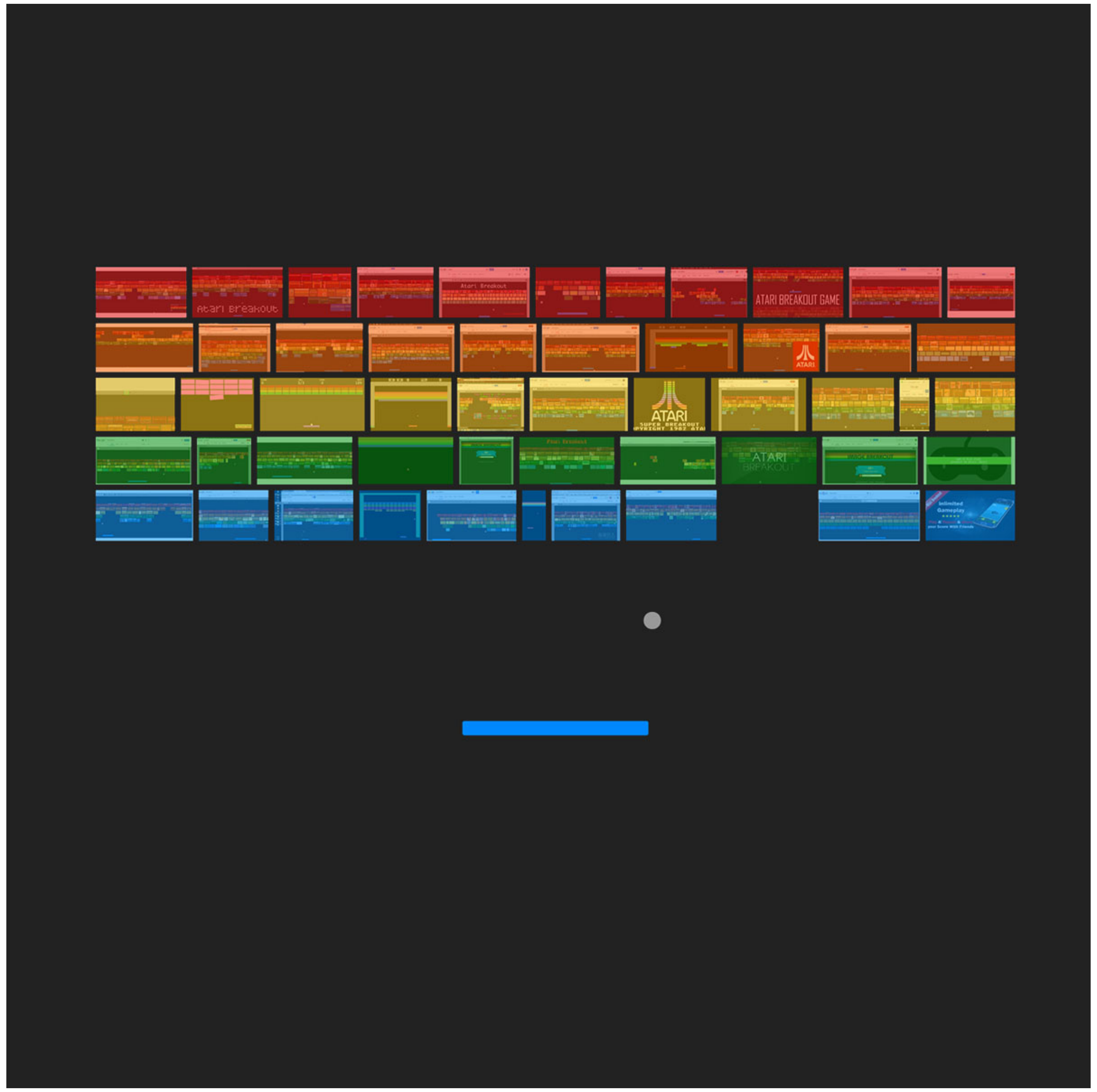


Image Credit

# Related Works

- General low dimensional RL algs
  - (Reidmiller, 2009), (Tesauro, 1995), (Diuk, 2008)
- Deep Learning Advances
  - (Bengio, 2009), (Krizhevsky, 2012)

# Method

## Q Values

- $Q^*(s, a) =$ max expected reward given $(s, a)$
- $Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \ldots \mid s_t = s, a_t = a, \pi]$
- $Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a\right]$
- $L_i\left(\theta_i\right) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right]$

# Method

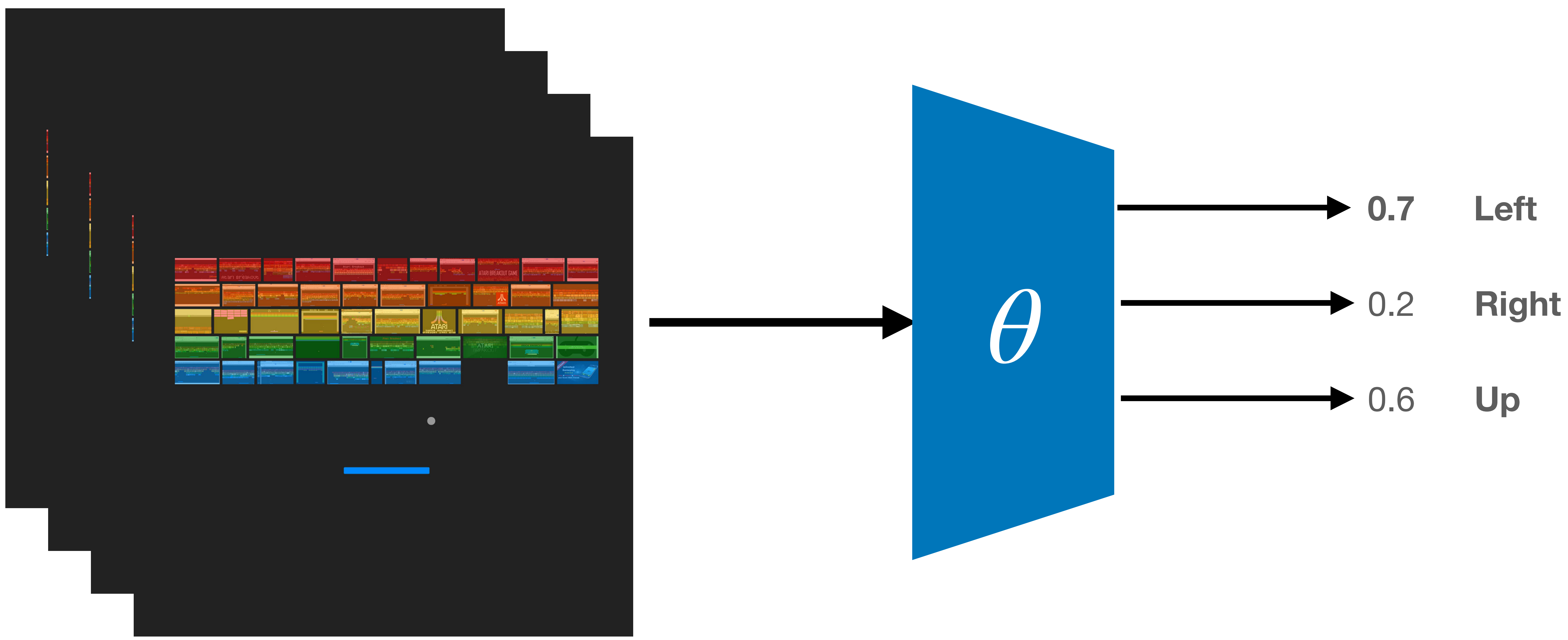
## Parametrization

- How shall we represent the Q network?

# Method

## Parametrization

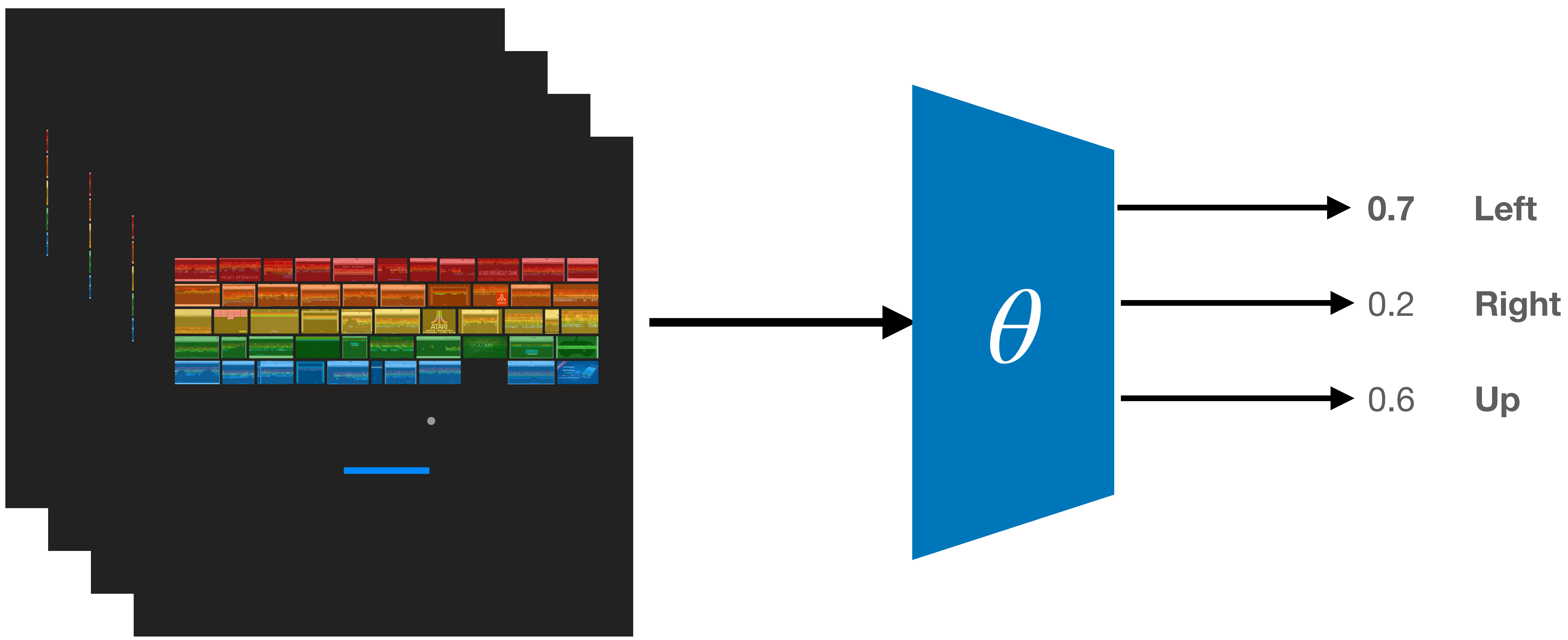
- How shall we represent the Q network?



# Method

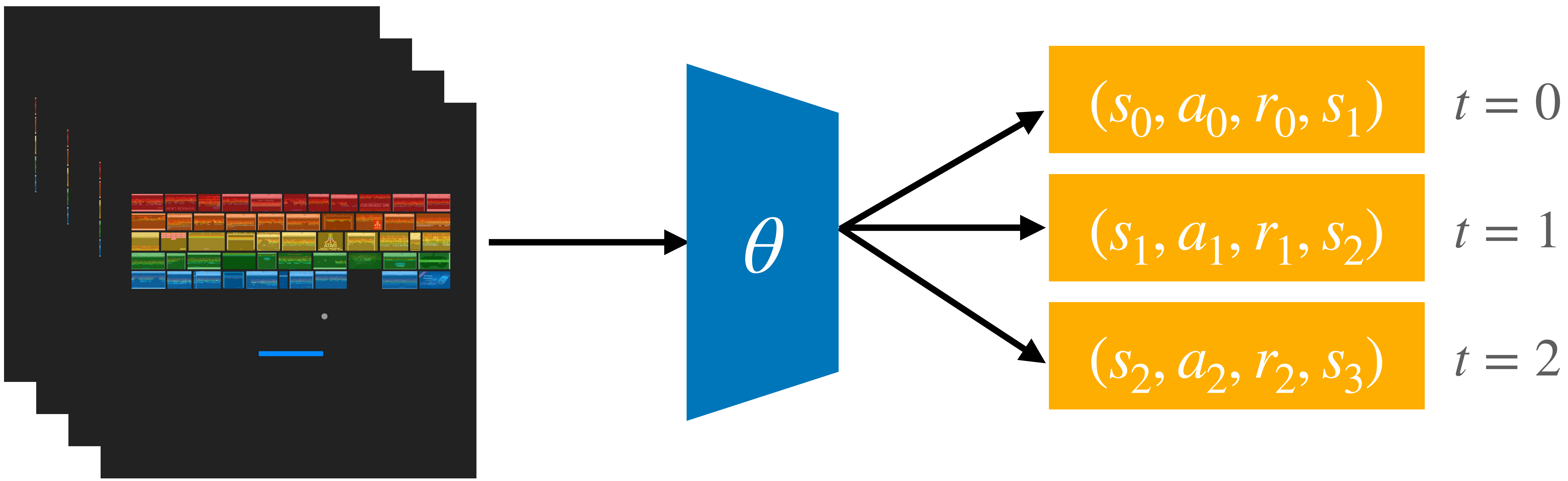
## Parametrization

- Quiz: why do we represent it in this way?



# Method

## Experience Buffer



# Method

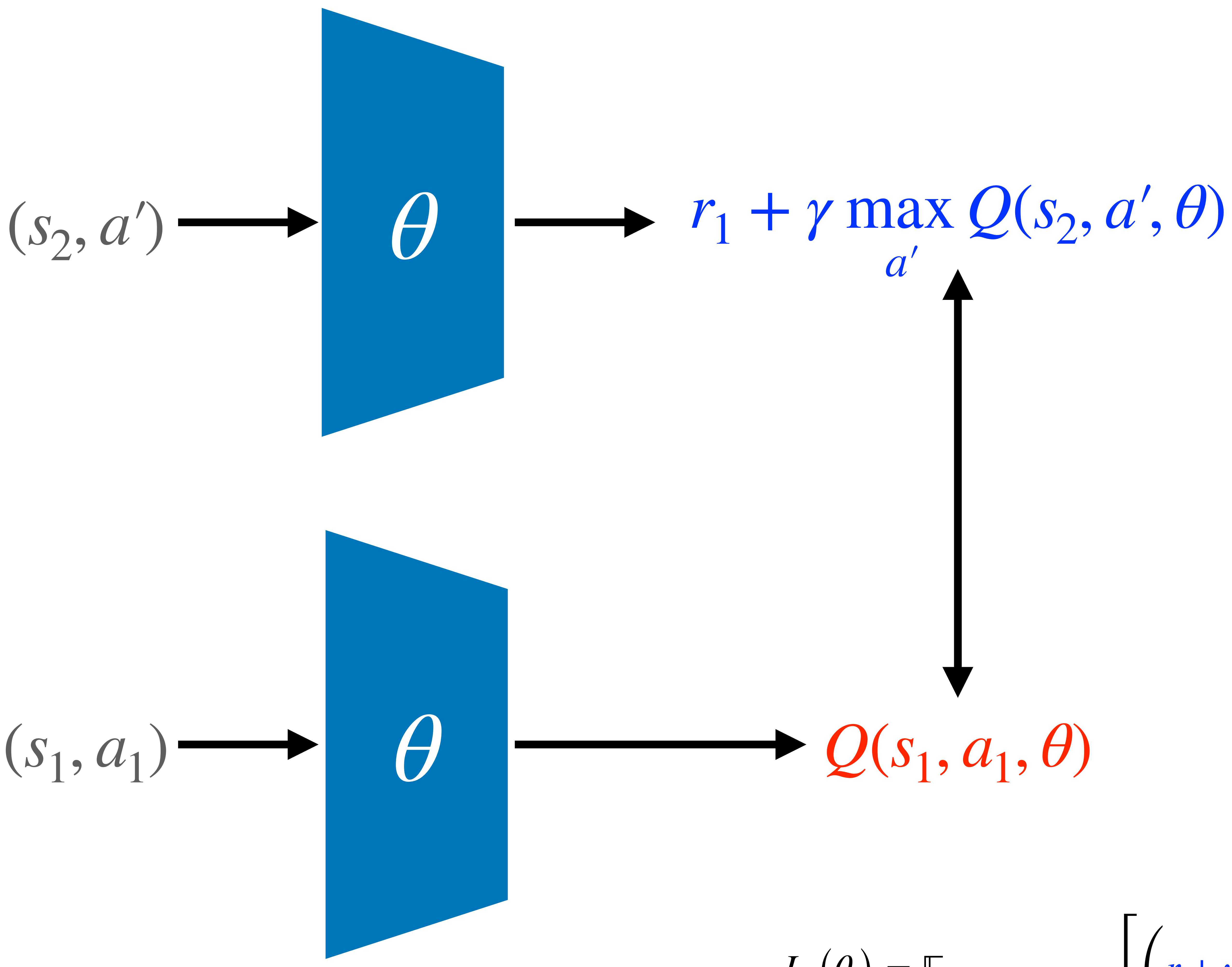
## Naive Q Update


| | |
|---|---|
| $(s_0, a_0, r_0, s_1)$ | $t = 0$ |
| $(s_1, a_1, r_1, s_2)$ | $t = 1$ |
| $(s_2, a_2, r_2, s_3)$ | $t = 2$ |

# Method

## Naive Q Update

$(s_1, a_1, r_1, s_2)$ $t = 1$




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

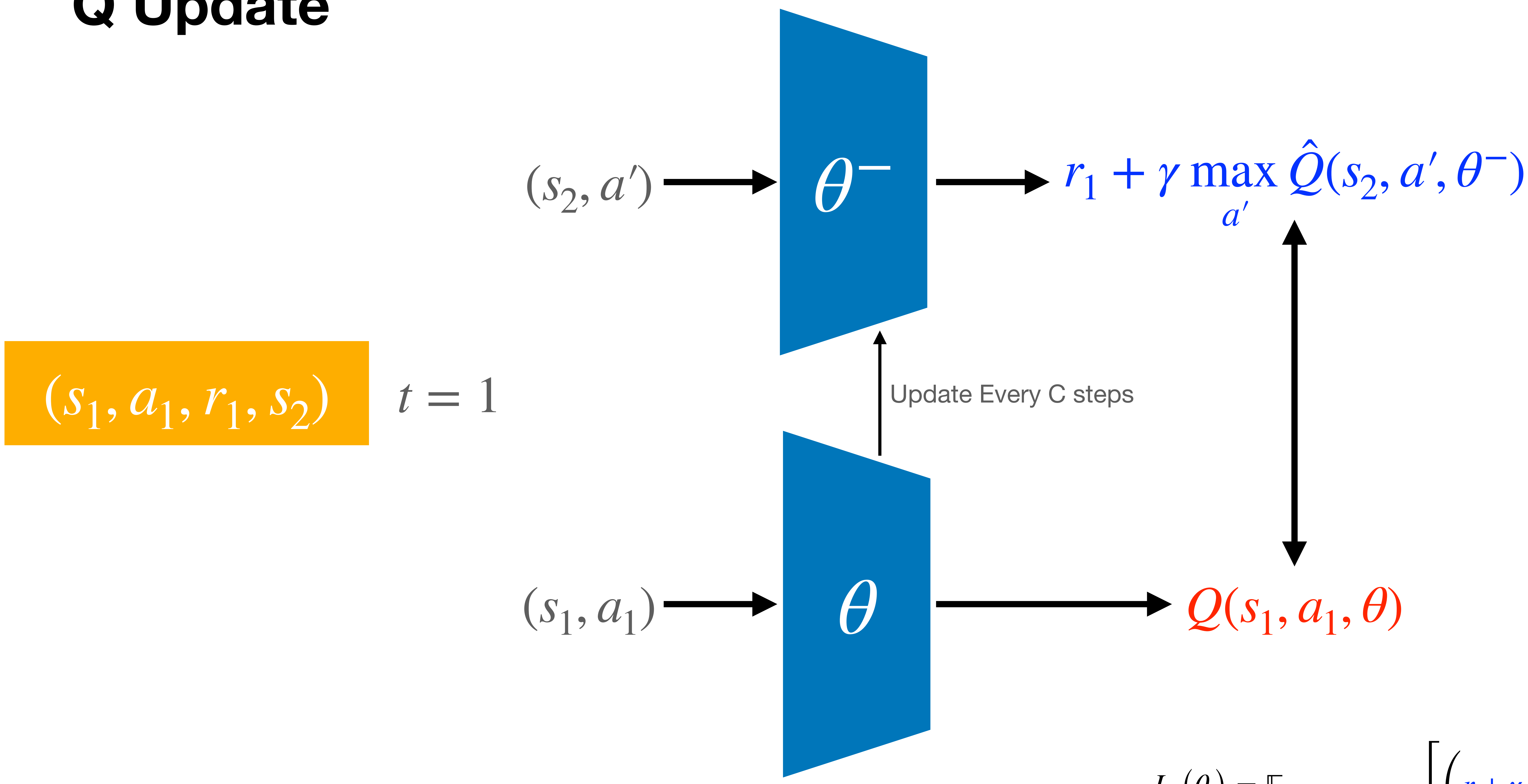
# Method

## Quiz: Naive Update

- Why might the above approach not work?
- Why do we need 2 Q networks?

# Method

## Q Update




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

# Method

## Full Algorithm

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\varepsilon$ 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 $D$
    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 $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **End For**
**End For**

# Method

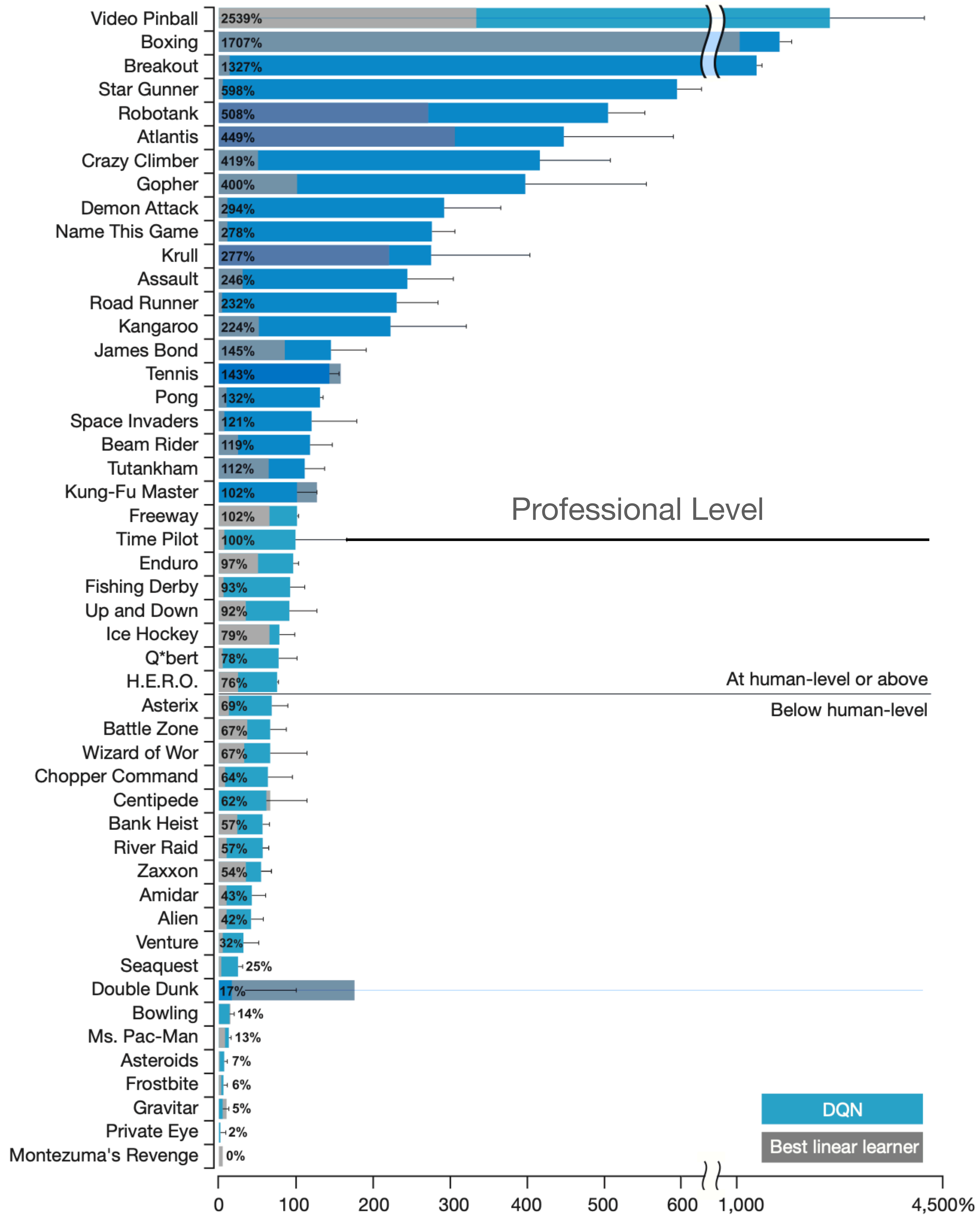
## Quiz: Gradient of Loss

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

# Results

## Performance

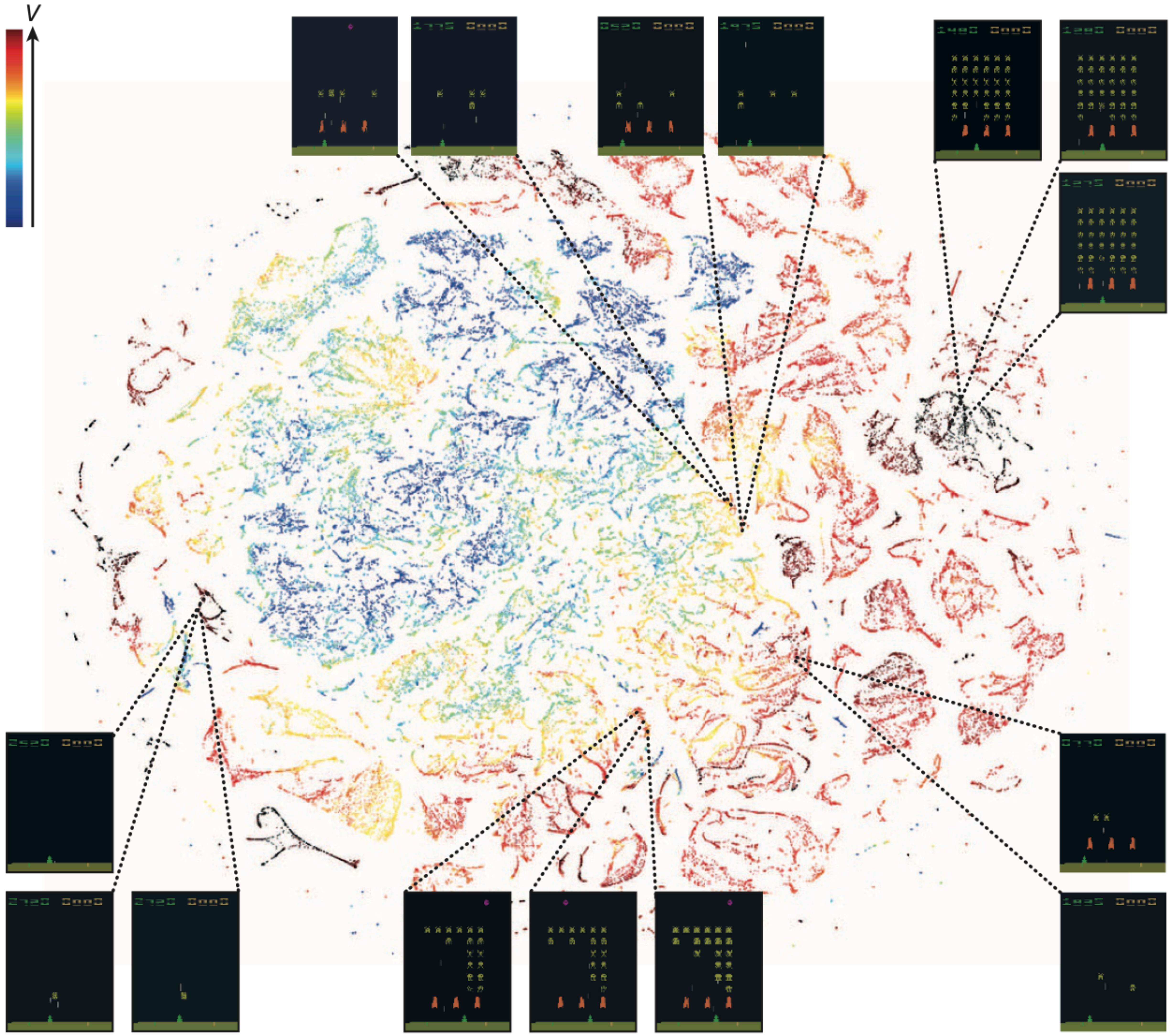
- Beats prior work on 43/49 games
- Often outperforms professional
- Montezuma's Revenge?



# Results

## Encoding

- Merge perception + reward



# Conclusion

- General framework for high dimensional games
- No handcrafted features
- NN can encode not only state, but value

# Discussion Questions

- Why could clipping rewards between [-1, 1] be detrimental, and how might we fix it?
- How might we sample more intelligently from our replay buffer?

**Training details.** We performed experiments on 49 Atari 2600 games where results were available for all other comparable methods[12,15]. A different network was trained on each game: the same network architecture, learning algorithm and hyperparameter settings (see Extended Data Table 1) were used across all games, showing that our approach is robust enough to work on a variety of games while incorporating only minimal prior knowledge (see below). While we evaluated our agents on unmodified games, we made one change to the reward structure of the games during training only. As the scale of scores varies greatly from game to game, we clipped all positive rewards at 1 and all negative rewards at −1, leaving 0 rewards unchanged. Clipping the rewards in this manner limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games. At the same time, it could affect the performance of our agent since it cannot differentiate between rewards of different magnitude. For games where there is a life counter, the Atari 2600 emulator also sends the number of lives left in the game, which is then used to mark the end of an episode during training.