Paper review: Deep Q-Networks

“Playing Atari with Deep Reinforcement Learning”

Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, Riedmiller

NIPS Deep Learning Workshop 2013

Desi R. Ivanova, 24 March 2021

Deep/Prob seminar

UNIVERSITY OF OXFORD

Overview

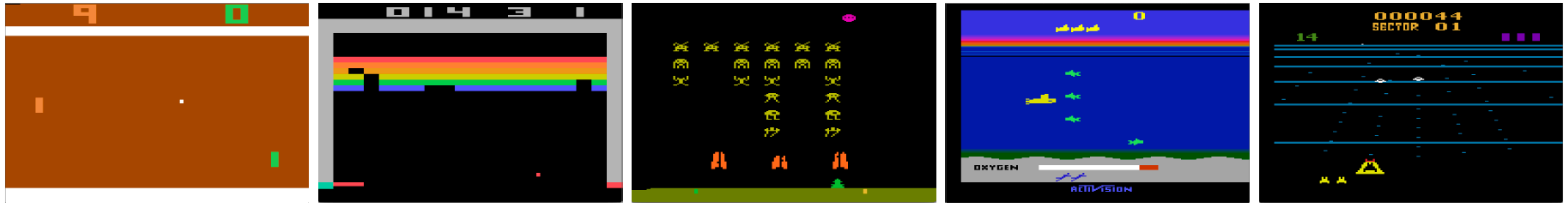


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Source: Mnih et al. 2013

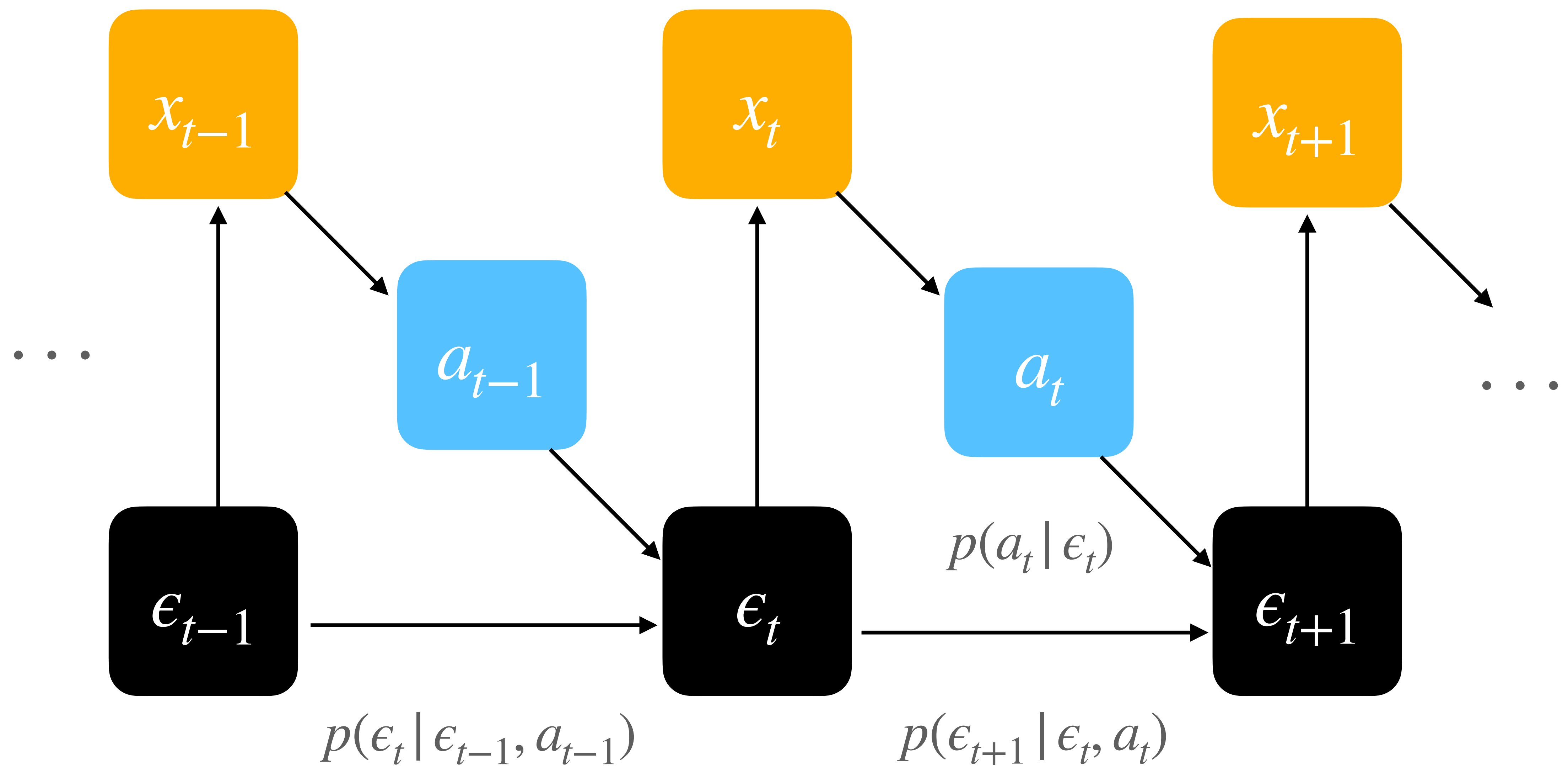
- The **first deep learning** model to learn control policies **directly** from high-dimensional **sensory input** (i.e. images) using reinforcement learning.
- Prior to this work: hand-engineered features, incorporating significant prior knowledge about the problem with simple (e.g. linear) value functions

The RL problem

At each time step t:

- Agent chooses an action a_t from a set of legal actions $\mathscr{A} = \{1,\ldots,K\}$ (K is small, between 4 and 18)
- Emulator, E, modifies **internal state** ϵ_t
- Observation $x_t \in \mathbb{R}^d$ vector representing current screen
- Agent receives a reward r_t
    - Depends on the game, can be very sparse

Partially observed MDP



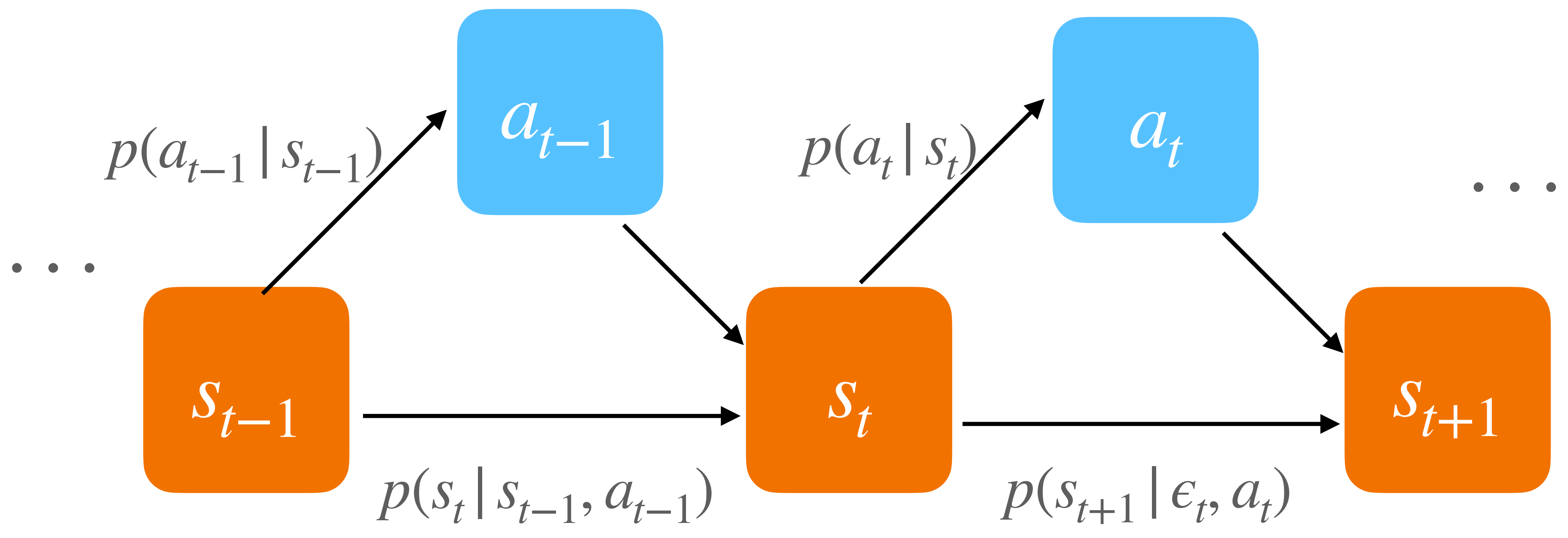
States are unobserved

The RL problem

- Consider sequences of actions and observations, $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$
- Each sequence $s_t \in \mathcal{S}$ is considered as a distinct state
- Large MDP but fully observed and finite
- Goal: select actions to maximise (discounted) future reward defined as

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

Fully observed (finite) MDP



Q-learning

Maximising future rewards

- Has been around for a long time but not combined with DL before
- Q-function maps {state, action} pairs to **future** reward $\begin{aligned} Q : \mathcal{S} \times \mathcal{A} &\to \mathbb{R} \\ (s, a) &\mapsto R \end{aligned}$
- Optimal Q-function defined as the max expected reward achievable by any policy π after observing state sequence s and taking an action a:

$$Q^*(s, a) := \max_{\pi} \mathbb{E}\left[R_t \,|\, s_t = s, a_t = a, \pi\right]$$

- Obeys the Bellman equation

$$Q^*(s, a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s', a') \,|\, s, a\right]$$

Under conditions (usually not satisfied) value iteration algorithms converge to the optimal Q^*

Approximate Q^* with a NN

- $Q^*(s, a) \approx Q(s, a; \theta)$, where θ are weights of a neural network, called the Q-network
    - Several options to parametrise Q (discussed later)
- Q-network is trained by minimising a sequence of loss functions $L_i(\theta_i)$

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho}\left[(y_i - Q(s, a; \theta_i))^2\right]$$

"Almost" like supervised regression problem→ can autodiff with respect to θ

"behaviour distribution" (in this case it is ϵ-greedy)

where $y_i = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) \,|\, s, a\right]$

BUT: the target depends on the network weights

Main problem
DL vs RL

- DL assumes data samples are **iid**.
- In RL: sequence of highly **correlated** states (think consecutive images in an Atari game); and data distribution **changes** as policy changes.
- Solution: **Experience replay** (Long-Ji Lin, 1993)
  - Store agent's experiences (transitions) at each time step $e_t := (s_t, a_t, r_t, s_{t+1})$
  - Replay memory: $\mathscr{D} = e_1, \ldots, e_N$ containing experiences over many episodes
  - $\mathscr{D}$ is updated periodically

Deep Q-learning

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

1million frames

Second "target network" added in a later version of the paper

Each x_t consists of 4 frames
ϕ : RGB to grayscale, downsampling, cropping,

Annealed from 1 to 0.1

ϵ-greedy strategy

"Frame-skipping": agent sees and select actions on every k^{th} frame

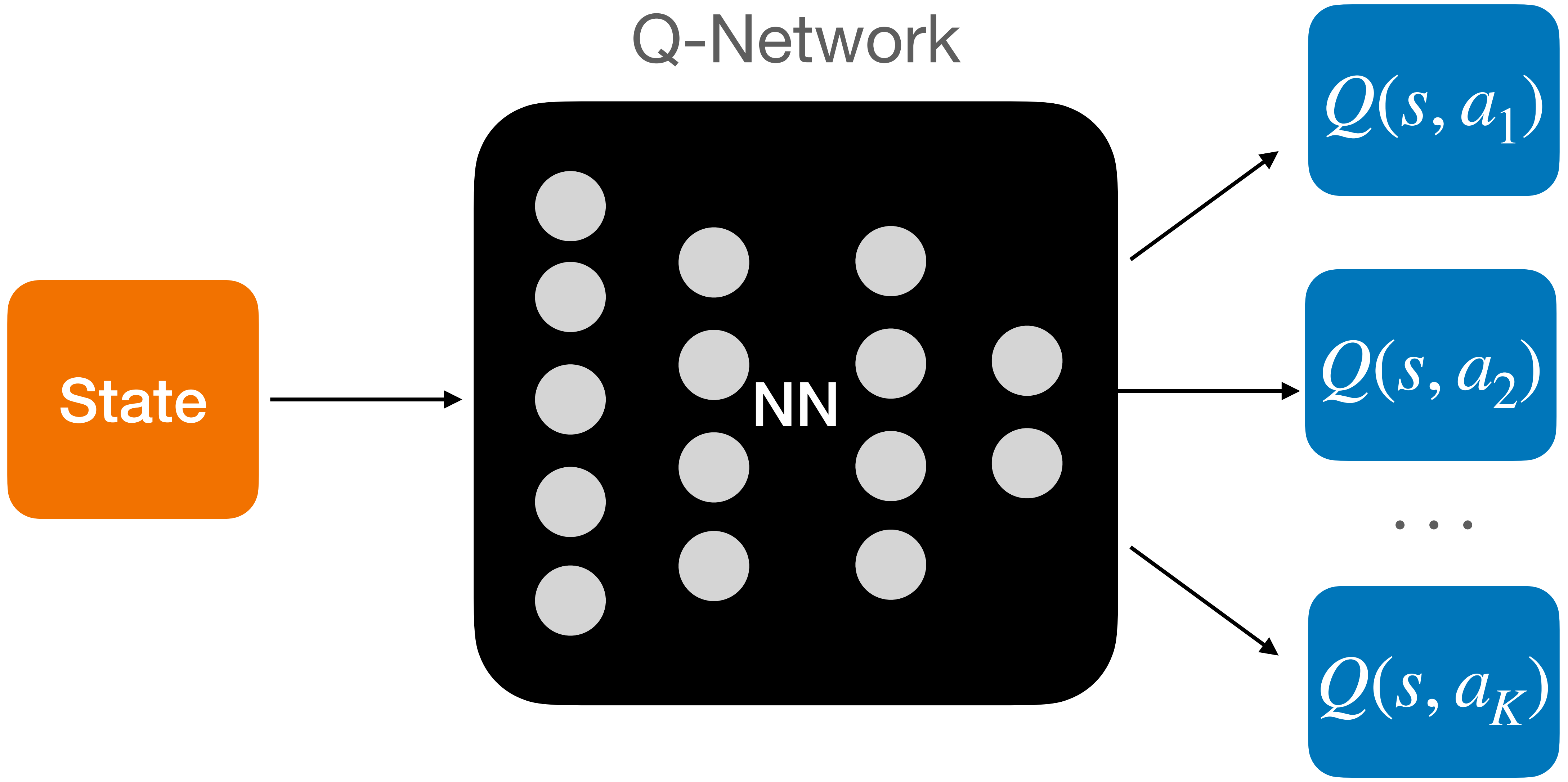
FIFO queue

batch size=32

Source: Mnih et al. 2013

Parametrising Q and NN Architecture

- One option is to input state and action, output $Q(s, a)$ (scalar)
  - Computational cost scales linearly with number of possible actions
- Instead: input state only, output a vector length K



- Convolutional NN taking 4 consecutive preprocessed images as inputs $(84 \times 84 \times 4)$
- 2 convolutional layers with ReLU activations
- Fully connected layers with ReLU activation
- Output layer consisting of a single neutron per valid action (between 4 and 18 for different games)
  - The same architecture and hyper-parameters are used in all 7 games, no game-specific information was incorporated.

Visualising the Value Function

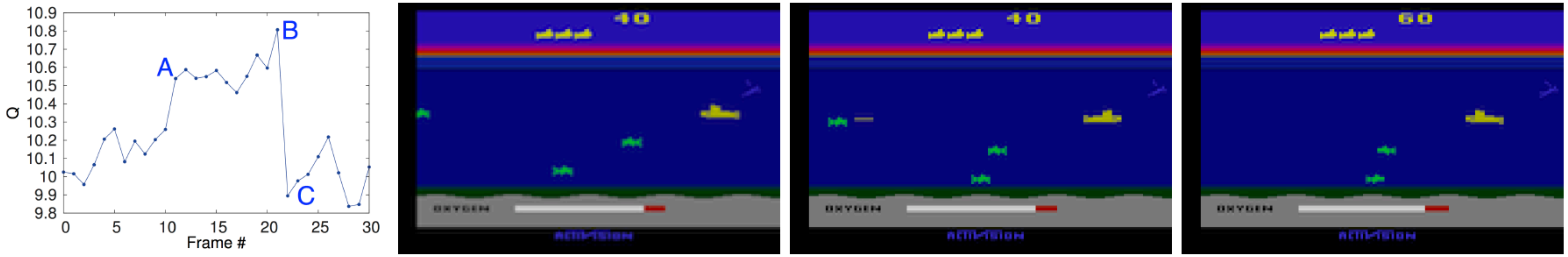


Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

Source: Mnih et al. 2013

Evaluation results

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

Source: Mnih et al. 2013