Reinforcement Learning

Aldo Faisal with contributions
by Ed Johns and Paul Bilokon

Imperial College London

July 2021 – Version 7.01

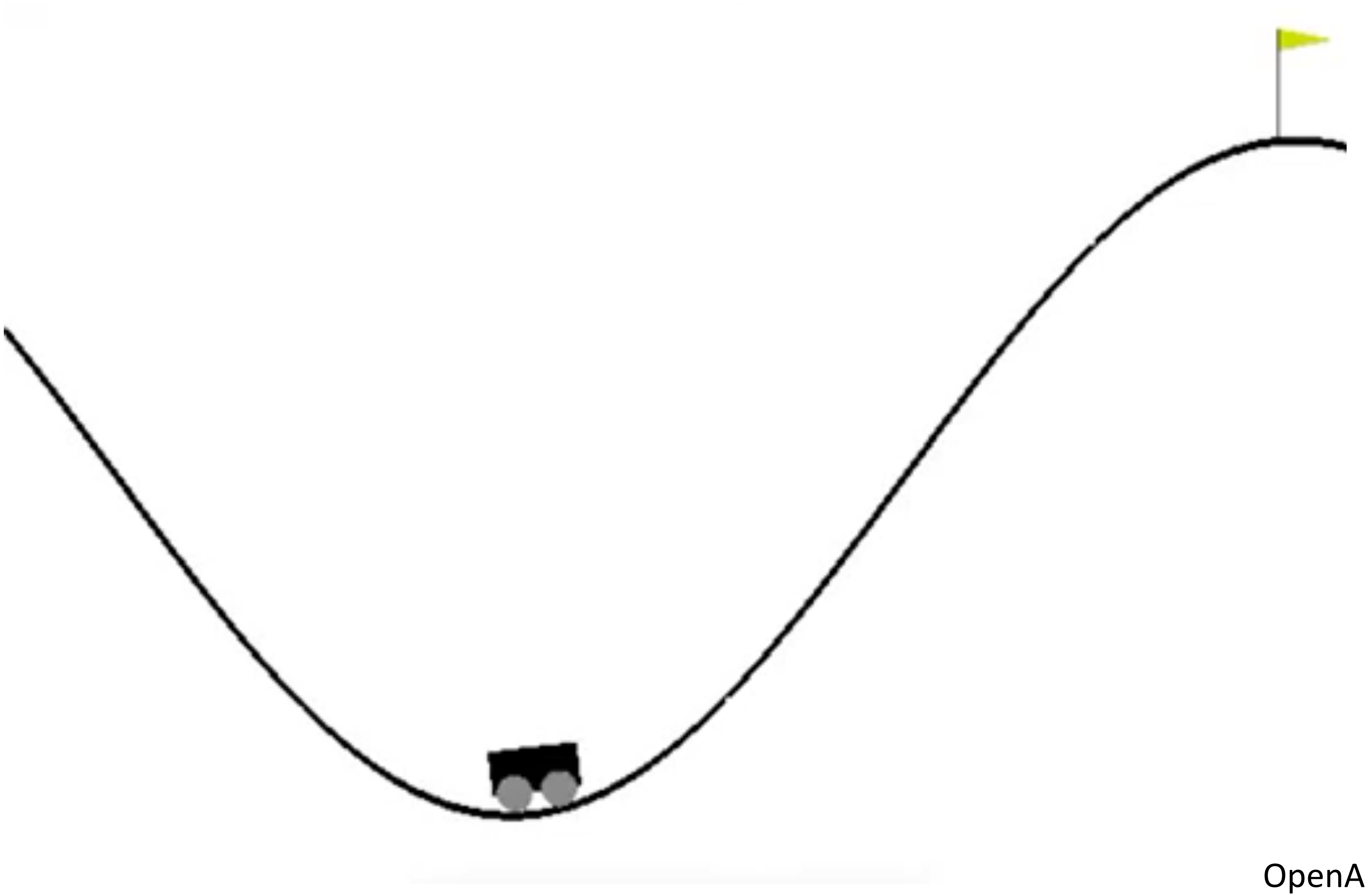
Section overview

Deep Q Learning

1. Motivation
2. Reinforcement Learning 101
3. Lets go Markov
4. Markov Decision Process
5. Dynamic Programming

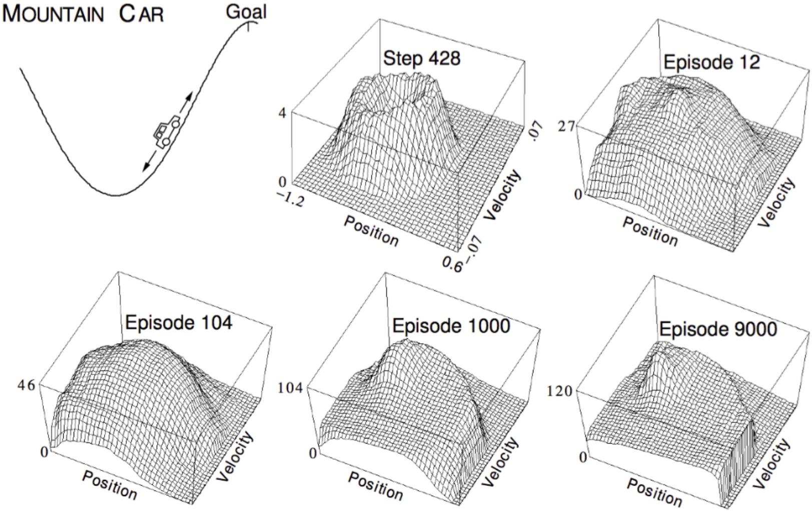
8. Function Approximation
9. Deep Q Learning
   - Putting the Deep in Reinforcement Learning
   - Policy Gradients
   - Mathematical interlude: Gaussian Distribution
   - Back to Policy Gradients
10. Actor-Critic Methods

Mountain Car Video

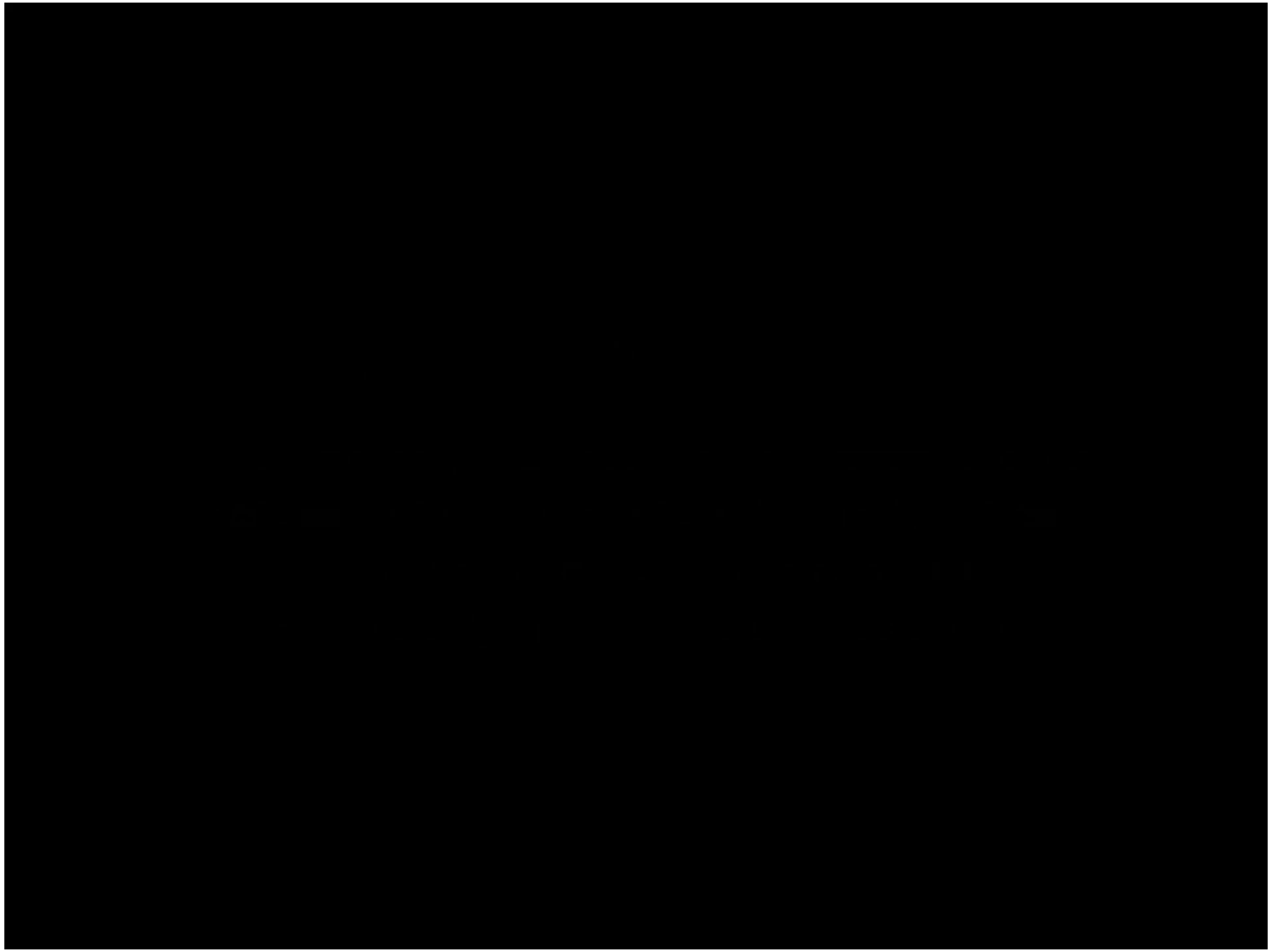


OpenAI Gym (YouTube)

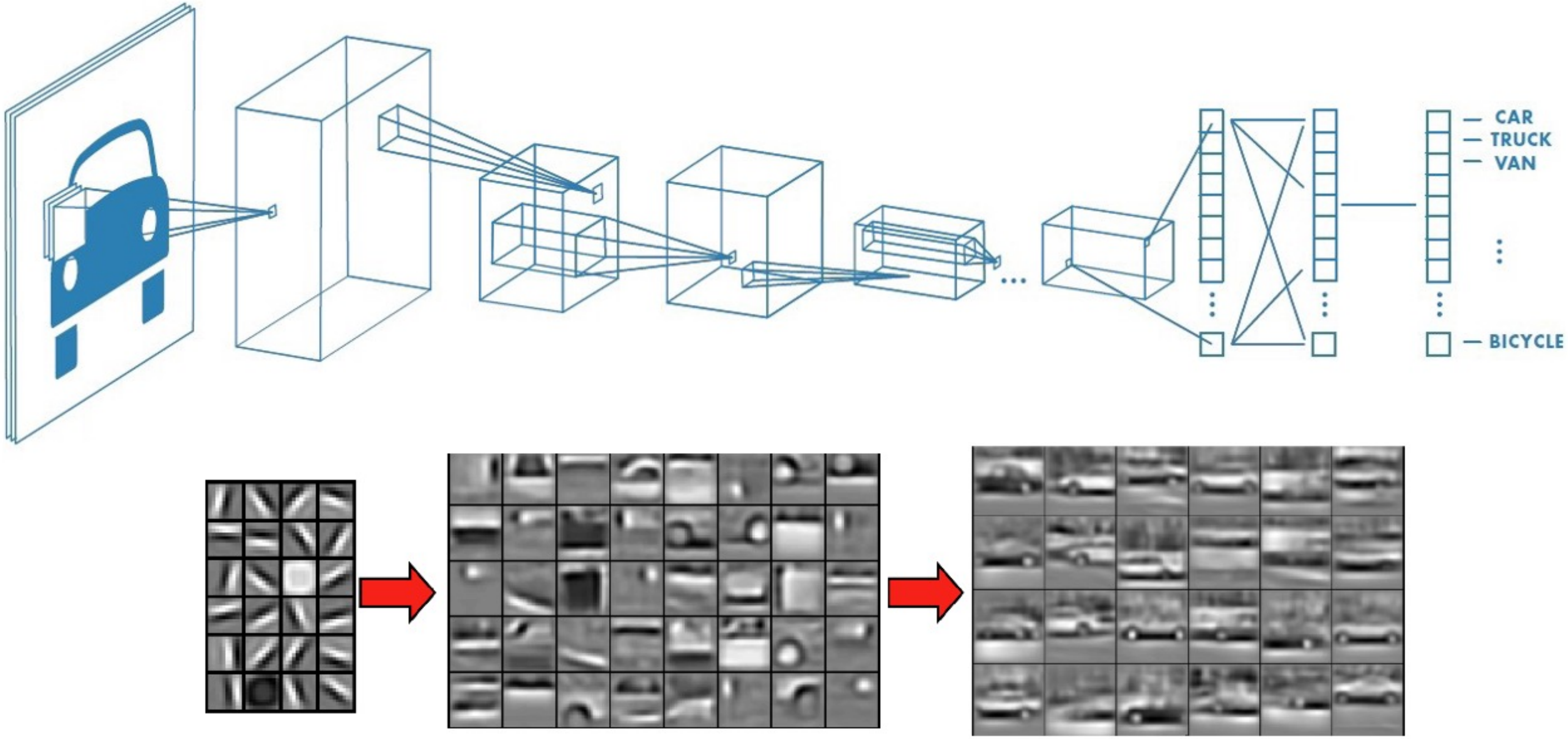
Example: Mountain Car example



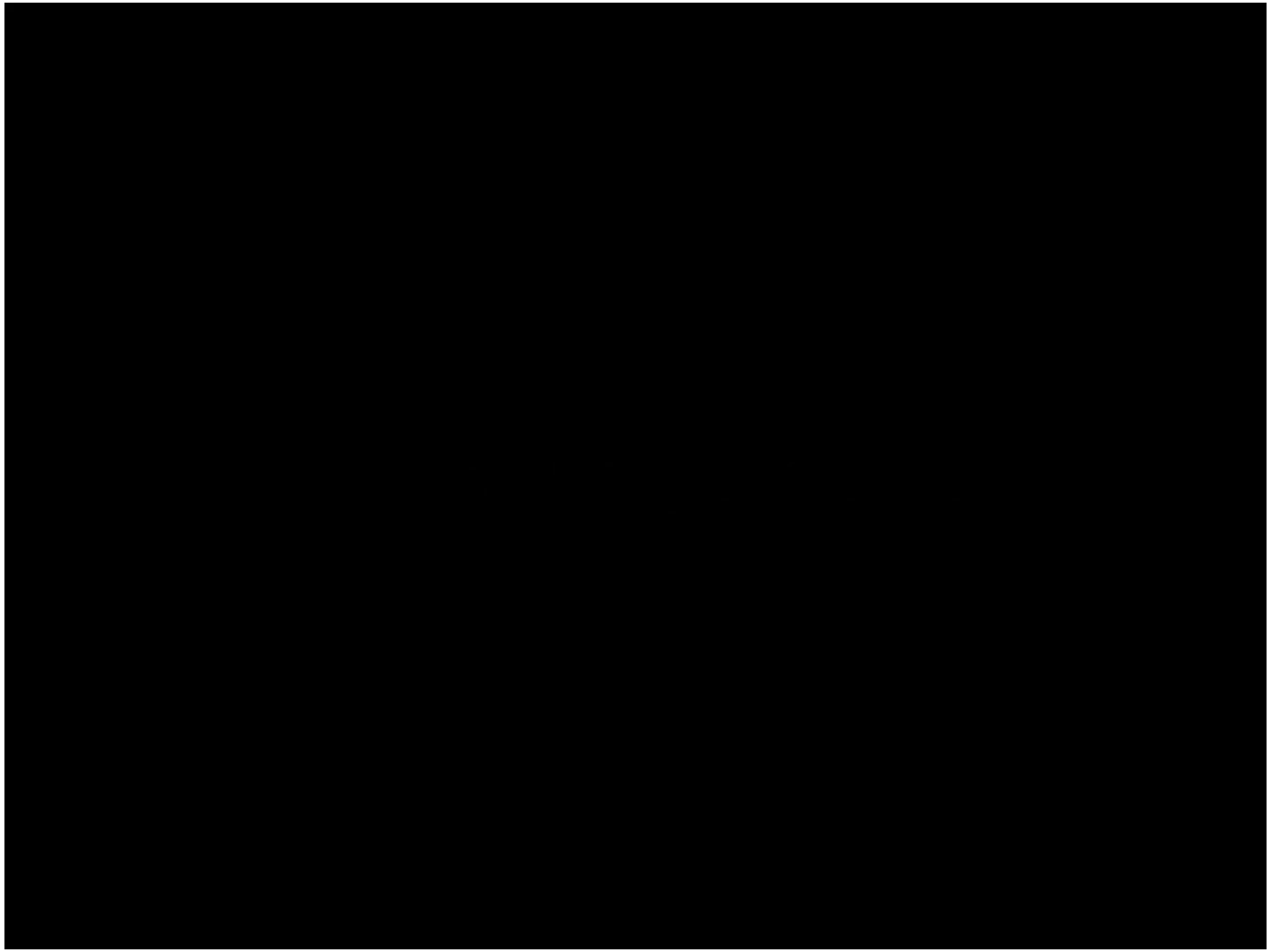
Space Invaders



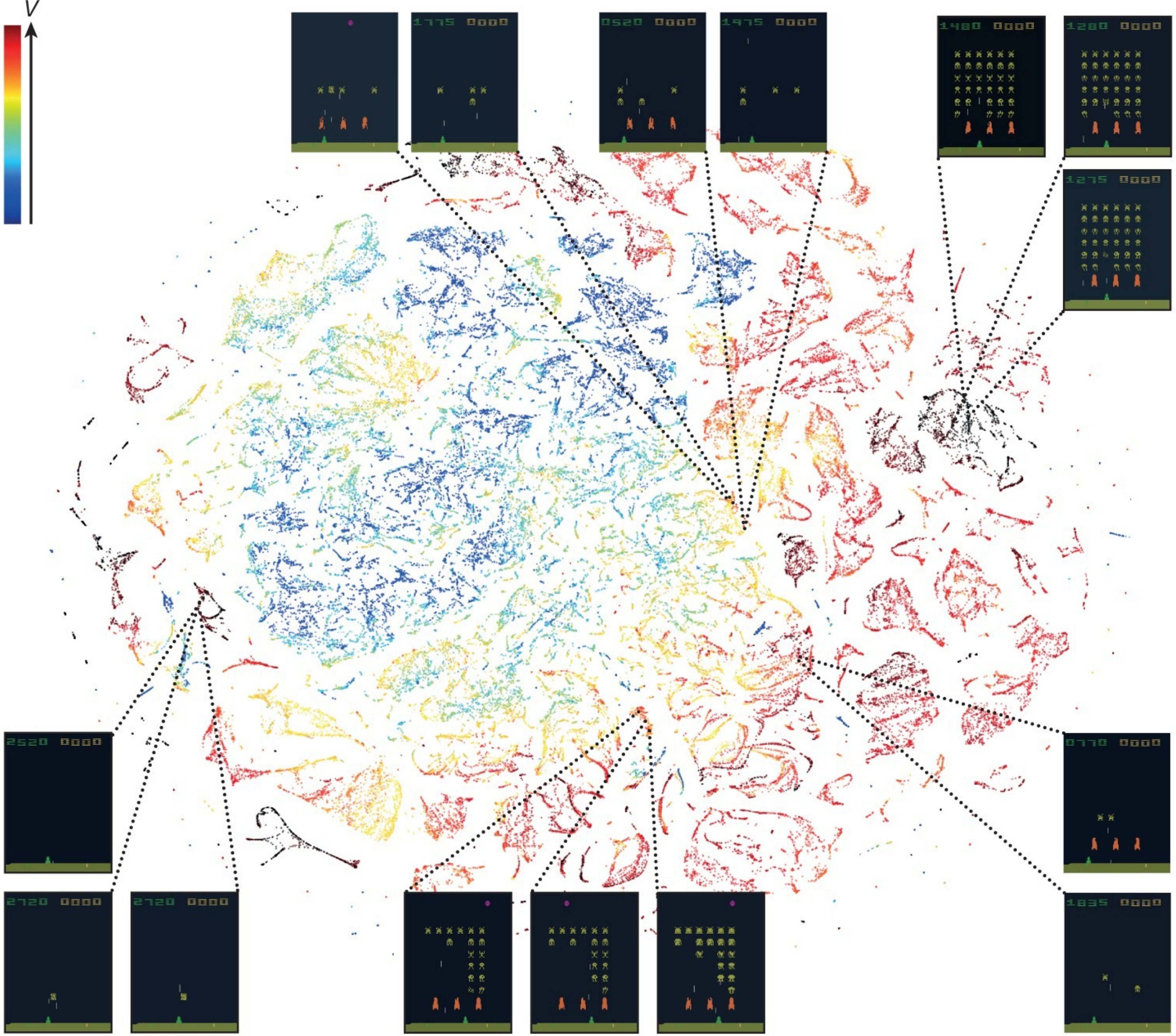
DeepMind (YouTube)


CAR
TRUCK
VAN
BICYCLE

Breakout



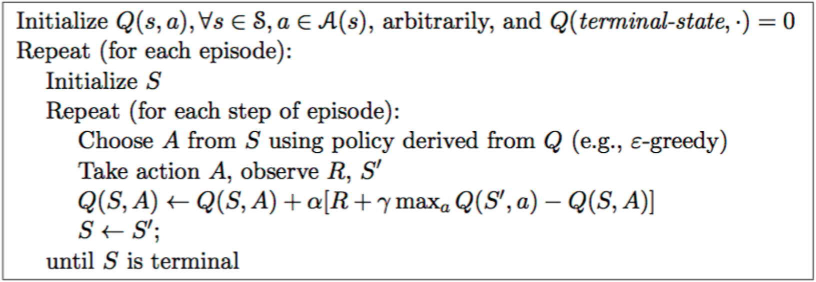
DeepMind (YouTube)


V
1775 0000
0520 0000
1975 0000
1480 0000
1280 0000
1275 0000
2520 0000
2720 0000
2720 0000
0770 0000
1835 0000

Off-Policy Function Approximation TD: The Deep RL triad

- Function approximation: A powerful, scalable way of generalising from a state space much larger than the memory and computational resources available (e.g., linear function approximation or CNNs). Leads to Deep RL.
- Bootstrapping: Update targets that include existing estimates (as in dynamic programming or TD methods) rather than relying exclusively on actual rewards and complete returns (as in MC methods). Leads to Deep TD Learning.
- Off-policy: training Training on a distribution of transitions other than that produced by the target policy. Sweeping through the state space and updating all states uniformly, as in dynamic programming, does not respect the target policy and is an example of off-policy training. Leads to Deep Q Learning.

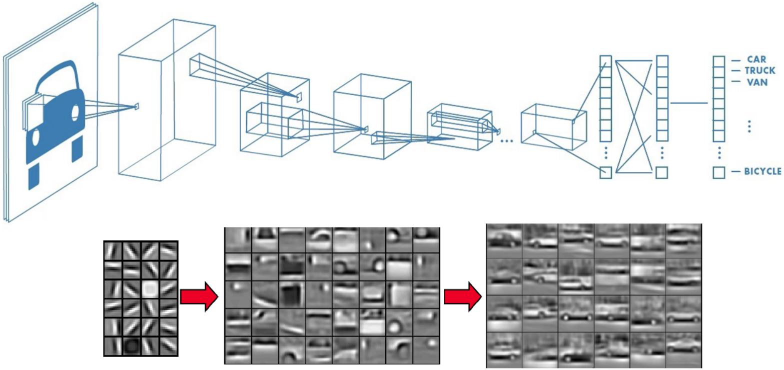
Q-Learning vs Lin FA Q-Learning



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$;
    until $S$ is terminal
```

Features (programmed not learned): Deep Learning

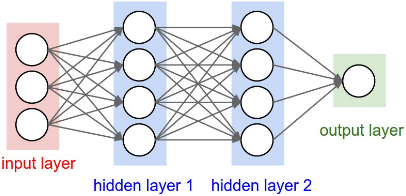
- Neural networks (NNs) are powerful function approximators



ATTENTION: Neural Networks are normally Supervised Learners

Features (programmed not learned): Deep Learning

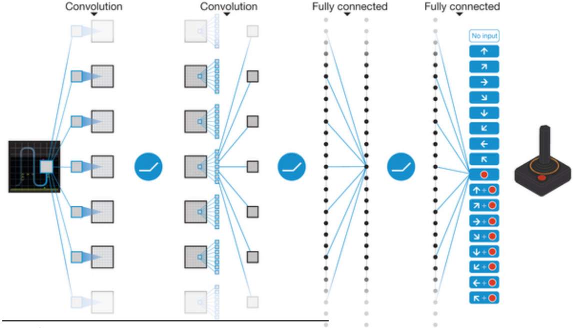
- Neural networks (NNs) are powerful function approximators
- NNs can learn features directly from (loads of) data
- Stacking layers enables learning hierarchical features (e.g. decomposing images into meaningful constituents)
- Deep learning is a subset of representation learning
- We can use it to replace hand-engineering of state space features with learning features from state data



Example: Atari DQN[3]

There are many function approximators, e.g.

- $Q(s, a)$ is approximated by a neural network
- Process raw pixels with convolutional layers for feature extraction
- Actions are from a discrete set (joystick commands)



[3] Mnih et al (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533

DQN: Bringing Deep Learning into RL I

In principle it sounds easy: use convolutional neural networks to link screen shots to values.

Given our TD Q-learning update:

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a') - Q(s_t, a)] \tag{57}$$

we want to learn $Q(s_{t+1}, a'; w)$ as a parametrised function (a neural network) with parameters w.

We can define the TD error as our learning target that we want to reduce to zero.

$$\text{TD error}(w) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a'; w) - Q(s_t, a; w) \tag{58}$$

DQN: Bringing Deep Learning into RL II

Taking the gradient of E wrt w we obtain:

$$\Delta w = \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; w) - Q(s_t, a; w)]\nabla_w Q(s_t, a; w) \tag{59}$$

However, the ATARI playing problem required more engineering to solve properly

1. Experience Replay
2. Target Network
3. Clipping of Rewards
4. Skipping of Frames

DQN: Experience Replay I

The CNN is easily overfitting the latest experienced episodes and the network becomes worse at dealing with different experiences of the game world:

1. Complication 1: Inefficient use of interactive experience
   - Training deep neural networks requires many updates, each has its own transition
   - But now each state transition is used only once, then discarded
   - so we would need to constantly revisit the same state transitions to train.

   This is inefficient and slow.
2. Complication 2: Experience distribution
   - Interactive learning produces training samples that are highly correlated (agents recent actions reflect recent policy)
   - The probability distribution of the input data changes

DQN: Experience Replay II

- The network forgets transitions that were in the not so recent past (overwriting past memory, in neuroscience known as "extinction")
- Moreover, because networks are monolithic, changes in one part of the network can have side-effects on other parts of the network [Why is this not a problem for table-based RL?]

Solution: Experience Replay[4] (alternative use Hierarchical RL).

- Experiences (traces) are stored and replayed in mini-batches to train the neural networks on more than just the last episode.
- Instead of running Q-learning on each state/action pairs as they occur during simulation or actual experience, the experience replay buffer system stores the traces sampled.

DQN: Experience Replay III

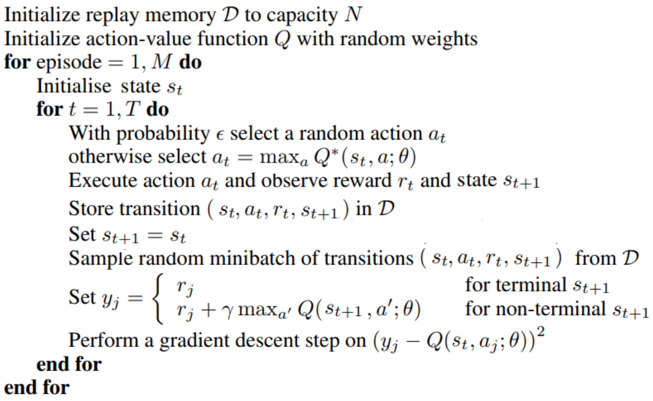
- Mini-batches are only feasible when running multiple passes with the same data is somewhat stable with respect to the samples. I.e. the transitions should low variance/entropy for the next immediate outcomes (reward, next state) given the same state, action pair.
- Replaying past data is also a more efficient use of previous experience, by learning (training the neural network) with it multiple times. This is key when gaining real-world experience is costly (e.g.simulation time) as the Q-learning updates are incremental and do not converge quickly.

DQN: Experience Replay IV

The learning phase is thereby separated from gaining experience, and based on taking random samples from this mini-batch table. DQN interleaves the two processes - acting and learning - because improving the policy will lead to a different behaviour that we should explore actions as these are closer to optimal ones (GPI). In summary

1. Reduction of correlation between experiences in updating DNN – mini-batches effectively make the samples more independently and identically distributed.
2. Increased learning speed with mini-batches
3. Reusing/Replaying past transitions to avoid catastrophic forgetting of associated rewards

DQN: Experience Replay V



```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise state $s_t$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(s_t, a; \theta)$
        Execute action $a_t$ and observe reward $r_t$ and state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
        Set $s_{t+1} = s_t$
        Sample random minibatch of transitions $(s_t, a_t, r_t, s_{t+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{t+1} \\ r_j + \gamma \max_{a'} Q(s_{t+1}, a'; \theta) & \text{for non-terminal } s_{t+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(s_t, a_j; \theta))^2$
    end for
end for
```

[4] Lin, L. J. (1993). Reinforcement learning for robots using neural networks (No. CMU-CS-93-103). Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

DQN: Target Network I

Whenever we run an udpate step on a Q-network's state we also update "near by" states (monlithic architecture + generalisation ability of CNNs).

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a)] \tag{60}$$

Issue: Unstable training resulting from bootstrapping a continuous state space representation

- We may have an unstable learning process, because changes to $Q(s, a)$ change $Q(s', a)$ (with $s - s' \approx \delta$) which changes $Q(s, a)$ on the next CNN update and then in turn $Q(s', a)$ forth. This can lead to a resonance effect

DQN: Target Network II

- The effect may result in a run-away bias from bootstrapping and subsequently dominating the system numerically, causing the estimated Q values to diverge.
- In calculating the TD error calculation, the target function is changing to frequently when using convolutional neural networks.

Solution: Slow things down (as resonance dampener)

DQN: Target Network III

Using a separate target network Q', updated every fewer time (e.g. every 1000 steps) with a copy of the latest learned weight parameters, controls this stability (relaxation time).

1. Initialise two Q networks: a main Q-network (Q), and a target network (Q')
2. When calculating the TD-error, use Q', not Q
3. Infrequently set $Q' = Q$
4. This gives the highly fluctuating Q time to settle (we can measure this so called "relaxation time" empirically and set it accordingly) before updating Q'

DQN: Clipping rewards I

Clipping rewards:

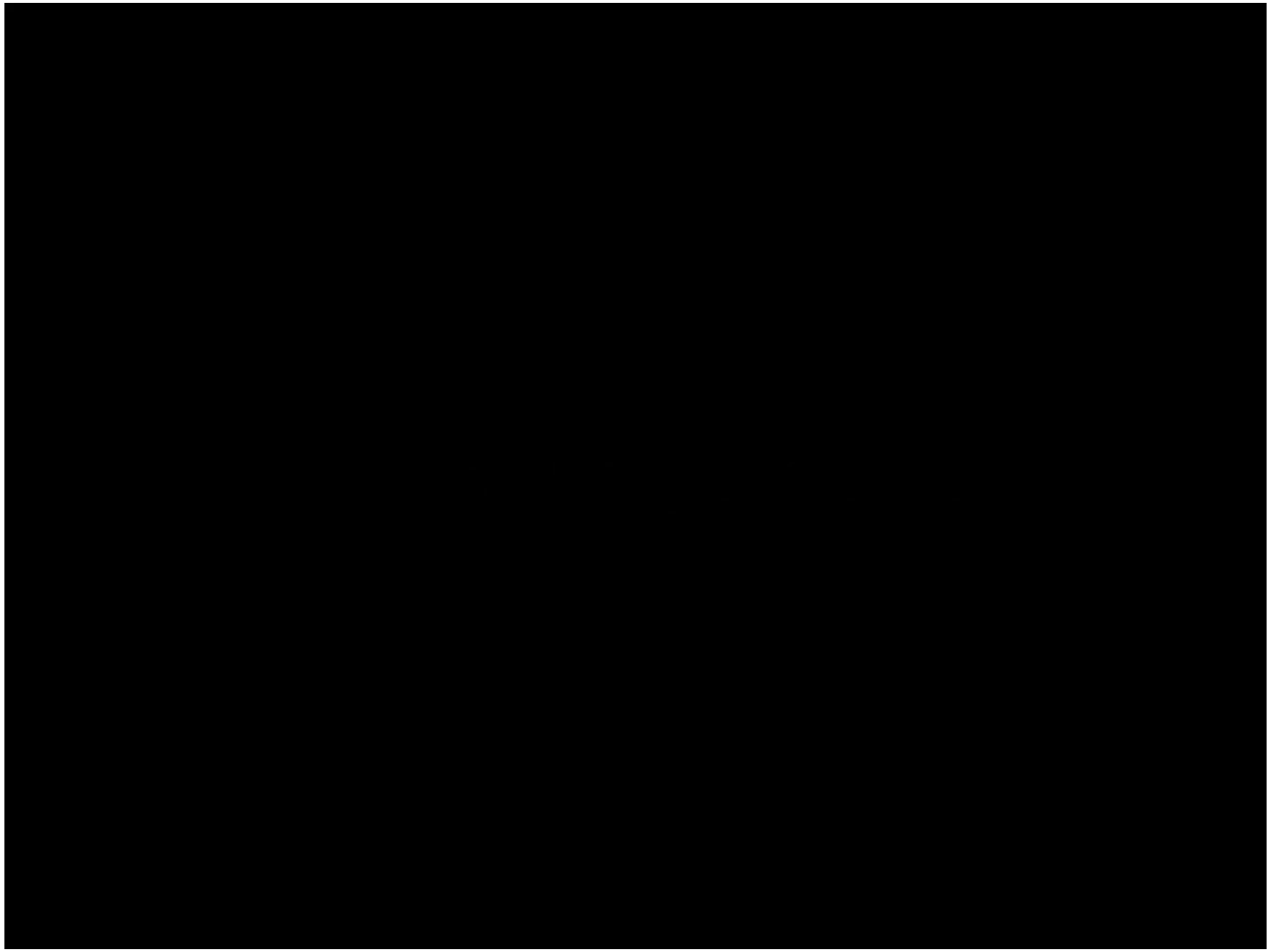
- Each ATARI game has different score scales. For example, in Pong, players can get 1 point when wining the play. Otherwise, players get -1 point. However, in Space Invaders, players get 10-30 points when defeating invaders. This difference would make training unstable.
- Clipping Rewards clips the rewards, with all positive rewards are set $+1$ and all negative rewards are set -1.

DQN: Skipping frames I

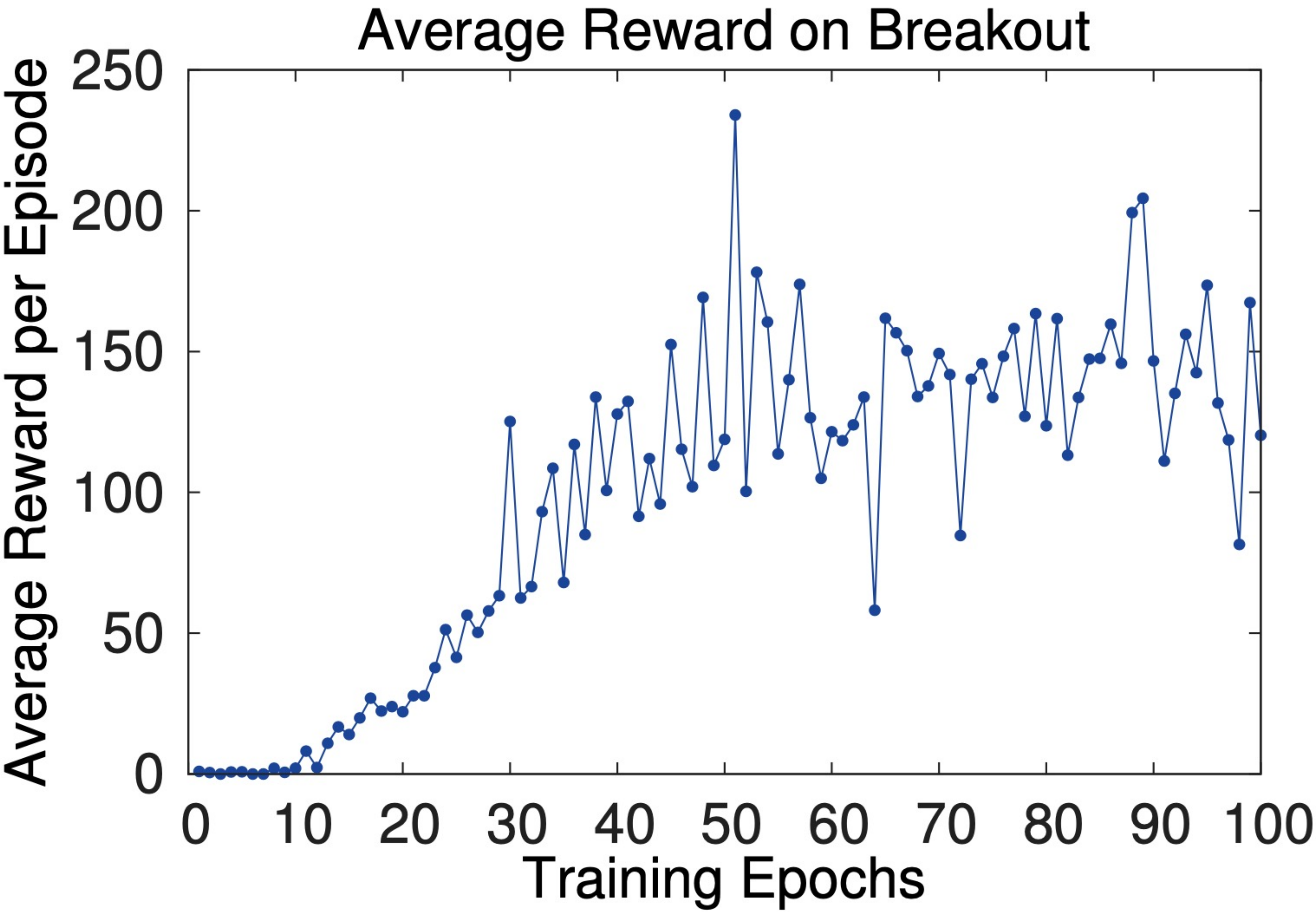
Skipping frames:

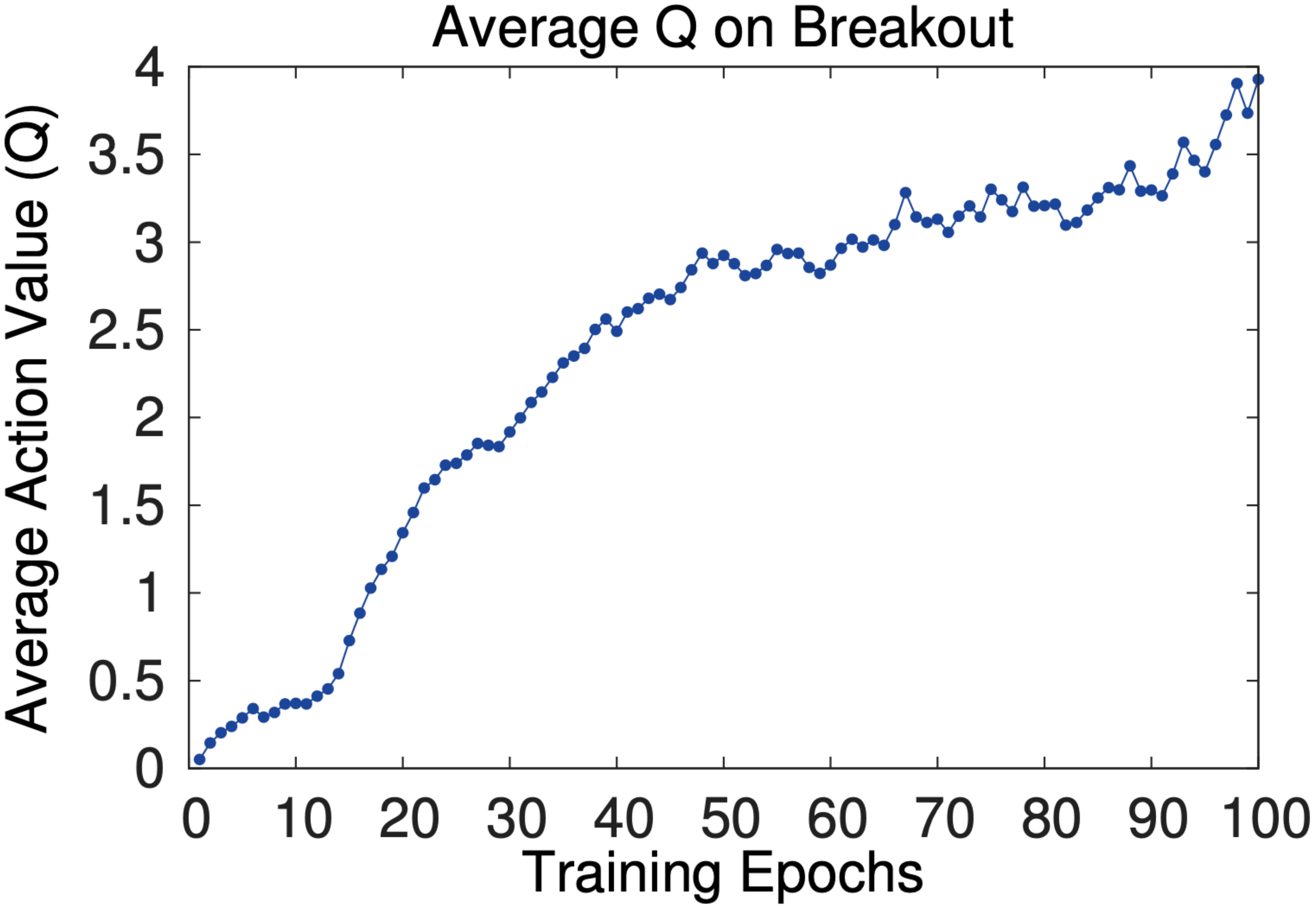
- Computer can simulate games (e.g. the ATARI simulator does is at 60 Hz) much faster than a human player would be able to react to the game, and this implies that the video game design and the required actions can be slower.
- Frame skipping uses only every 4 video game frames (i.e. 15 Hz), the past 4 frames as inputs. Reducing computational cost and accelerating training times.

Breakout



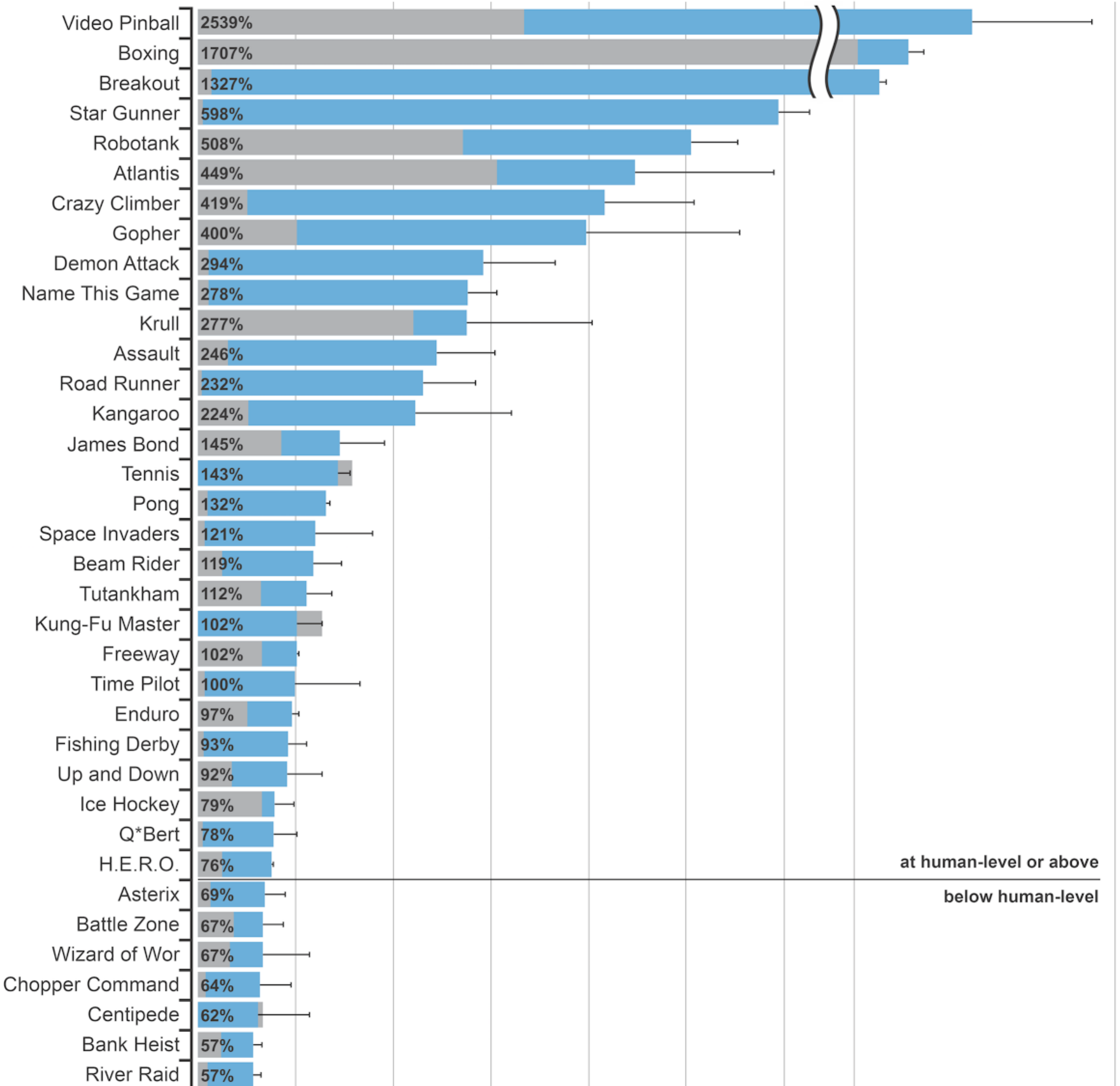
DeepMind (YouTube)

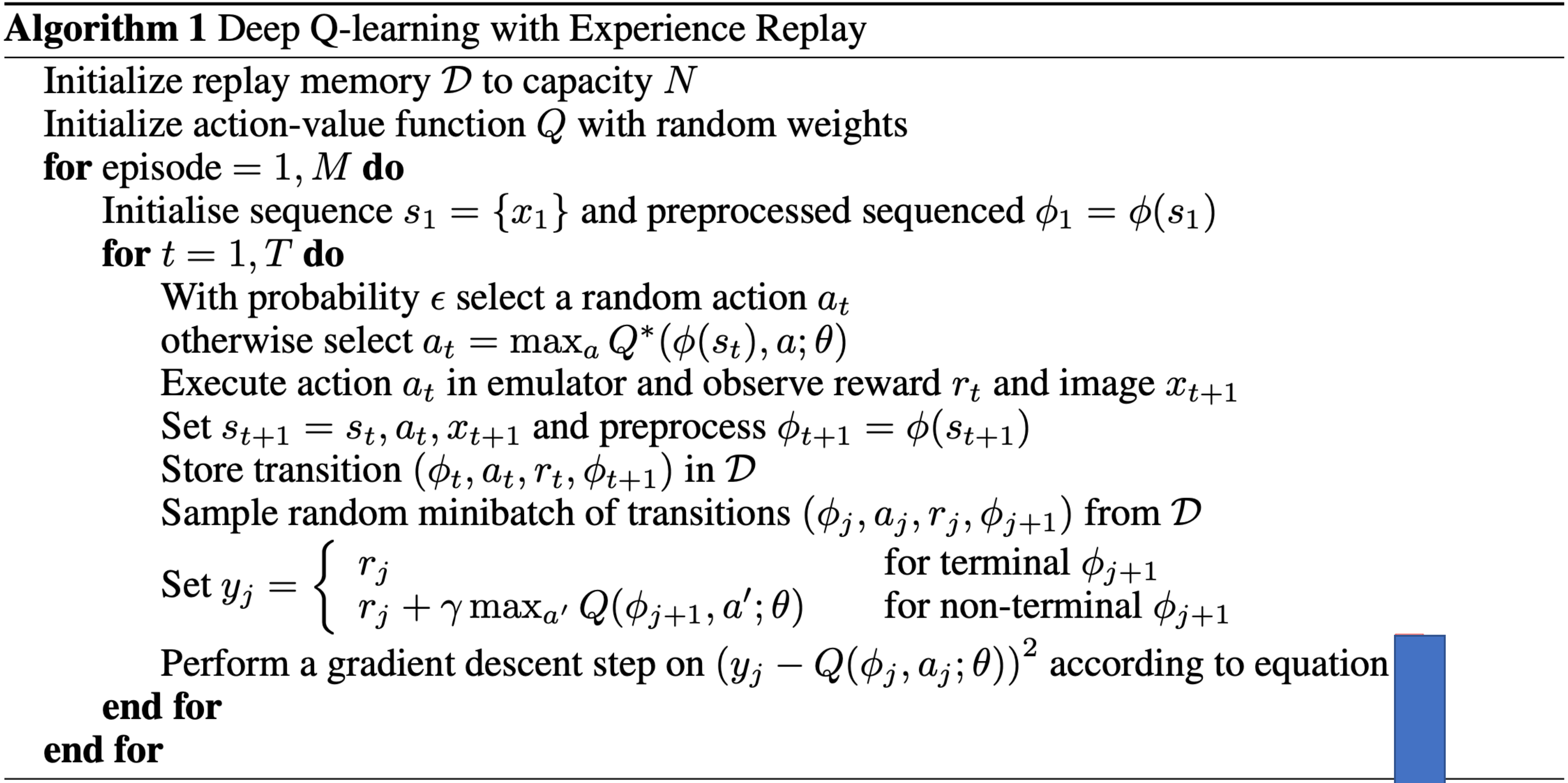

Average Reward on Breakout
Average Reward per Episode
250
200
150
100
50
0
0
10
20
30
40
50
60
70
80
90
100
Training Epochs


Average Q on Breakout
Average Action Value (Q)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
10
20
30
40
50
60
70
80
90
100
Training Epochs

5
3

120
OXYGEN
ACTIVISION


Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*Bert 78%
H.E.R.O. 76%
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
at human-level or above
below human-level



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 $\epsilon$ 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
    end for
end for
```

$$\nabla_{\theta_i} L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$