Human-level Control through Deep Reinforcement Learning

Mnih et al, 2015

The Aim

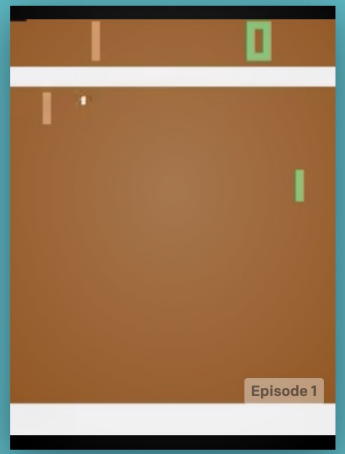
- To create an agent that would perform well in different, challenging tasks
- The task for the paper is a set of 49 games from the Atari 2600 series.
- The games to be used are provided as an emulator by OpenAI gym.
- The agent is to be trained using Deep Q-Learning (more on this later)

Pong-v0

Maximize your score in the Atari 2600 game Pong. In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from $\{2, 3, 4\}$.

The game is simulated through the Arcade Learning Environment [ALE], which uses the Stella [Stella] Atari emulator.

[ALE] *MG Bellemare, Y Naddaf, J Veness, and M*



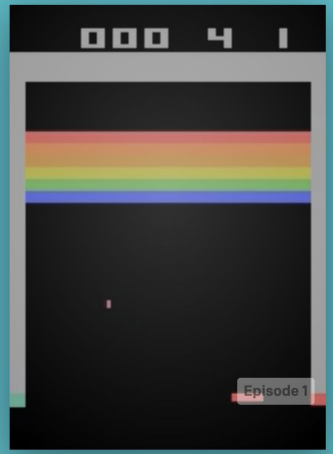
RandomAgent on Pong-v0

Breakout-v0

Maximize your score in the Atari 2600 game Breakout. In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of k frames, where k is uniformly sampled from $\{2, 3, 4\}$.

The game is simulated through the Arcade Learning Environment [ALE], which uses the Stella [Stella] Atari emulator.

[ALE] *MG Bellemare, Y Naddaf, J Veness, and M*



RandomAgent on Breakout-v0

Agent and Emulator

- **Agent:** The agent selects an action from the list of possible actions (A1, ...Ak). This it does by using a policy.
- The policy improves by training, which involves playing the game a large number of times. This is done using the **emulator,** whose task is:
  - Give the image of the screen during gameplay (this will help us train the policy approximated as a CNN)
  - Give the game score. This will be used in the reward function for performing SGD.
- The agent's goal is to maximize future rewards. The future rewards are discounted by a factor of 0.99 in the paper, and the expression is:

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

Action-value function Q

- An optimal action-value function (returning the maximum possible future reward after taking an action ‘a’ from a state ‘s’ must follow the Bellman Equation (here, Q* denotes the optimal value function, and π denotes the policy (a mapping of states to actions)).

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

The Reinforcement Learning Problem

01 While it is theoretically possible to use the Bellman Equation to develop an optimal policy by working bottom's up, it is completely infeasible.

02 The Universal Function Theorem says that neural nets can effectively approximate any mathematical function, and Deep RL uses a NN to emulate Q.

03 Thus, the general reinforcement learning problem includes an action policy, which we run on the environment, and iteratively improve using the rewards.

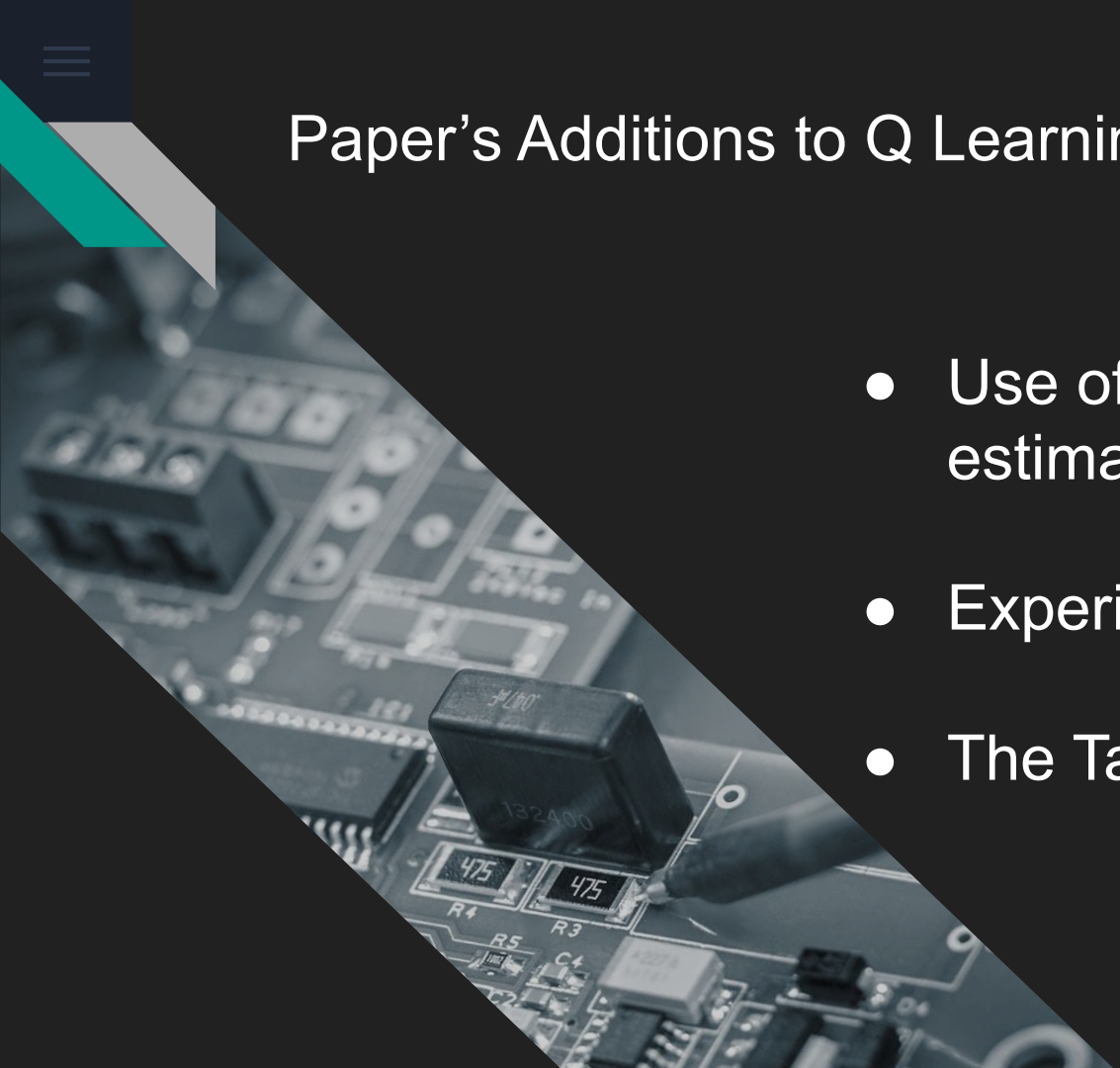
The Loss Function

- The loss function is used to update the policy. Since we're approximating the policy using a neural network, the loss function (Huber Loss is used in the paper) helps us perform SGD to change the parameters (weights of the NN in this case (actually filters in CNN)).

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

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

Paper’s Additions to Q Learning



- Use of CNNs for Q value estimations.
- Experience Replay
- The Target Network

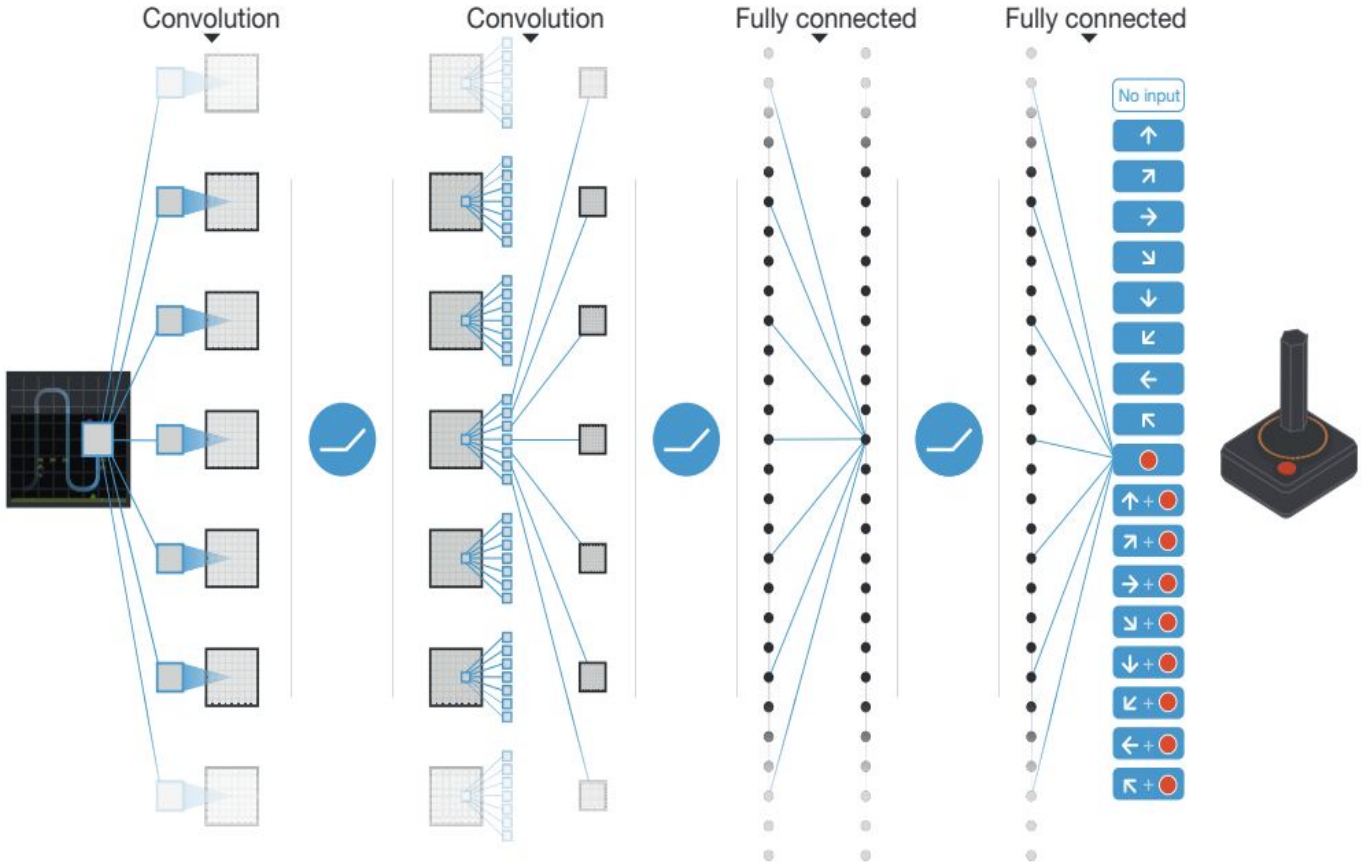


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

Experience Replay

- Store a history of **100000** most recent trajectories (the states, actions, rewards and next states).
- Draw random minibatch of samples from this history during training. This separates experience with learning, and thus
- The purpose of experience replay is that it prevents the agent from learning from only recent experiences.
- Thus, with better use of previous experience, and the independence of data, experience replay leads to better convergence.

Second Target Network (Q-hat)

- This is a heuristic technique used by the paper to stabilize the network's learning by decreasing the chances of divergence.
- Instead of the main network Q (which we're improving), we use a separate network Q-hat to generate the targets on each update.
- Periodically, set Q-hat to be the updated main Q, thus incorporating learning into generation too.
- The reason why it makes sense to do this is that it solves the "**Dog chasing it's own tail**" problem

Training Details

- RMSProp Optimizer is used in the paper. We did some study on this and found that while **Adam works better**, it was not invented during that time.
- Annealing (like SA from AI) is done on the epsilon value from **1 to 0.1** in the first **100000 frames,** after which epsilon is set to 0.1.
- The network is trained for about **50 mn** frames till reward saturates (we were not able to do that much with our resources, hence we only went till 1.5 mn frames, and observed the increasing rewards.
- Atari images from gym are 210x160x3, which has a lot of superfluous data, hence the paper rescales them to 84x84 and grayscales.
- Stack the 4 most recent frames to give velocity information.

Training Details (continued)

- Both **rewards and gradients** were **clipped at [-1, 1]**. We experimented with and without these constraints. Clipping of gradients was done by using a **“Huber Loss”** for calculating loss values
- Instead of every frame, the paper selects an action on every 4th frame, thus this makes it less expensive to run the emulator for one step than to have the agent select an action.
- Hyperparameter search was done using “informal search” and not any formal grid search algorithm due to the curse of dimensionality.

THE ALGO

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 $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

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

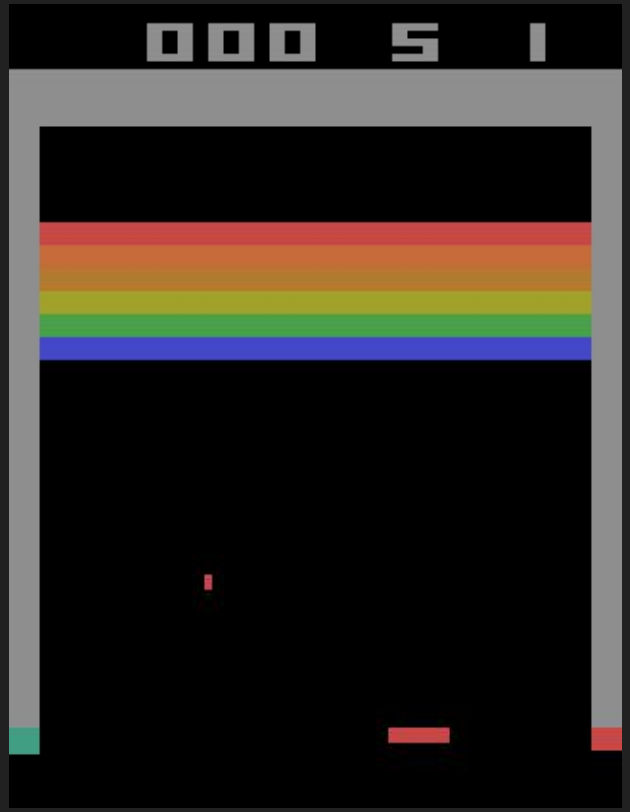
Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

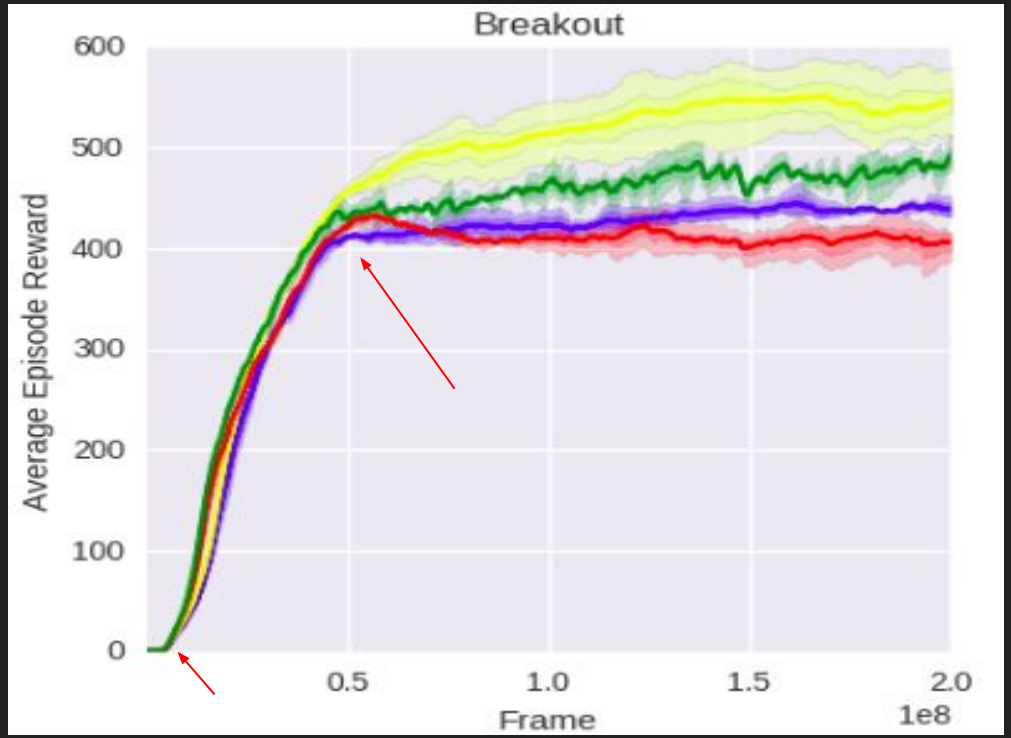
End For

Our Results (mid training GIFs)

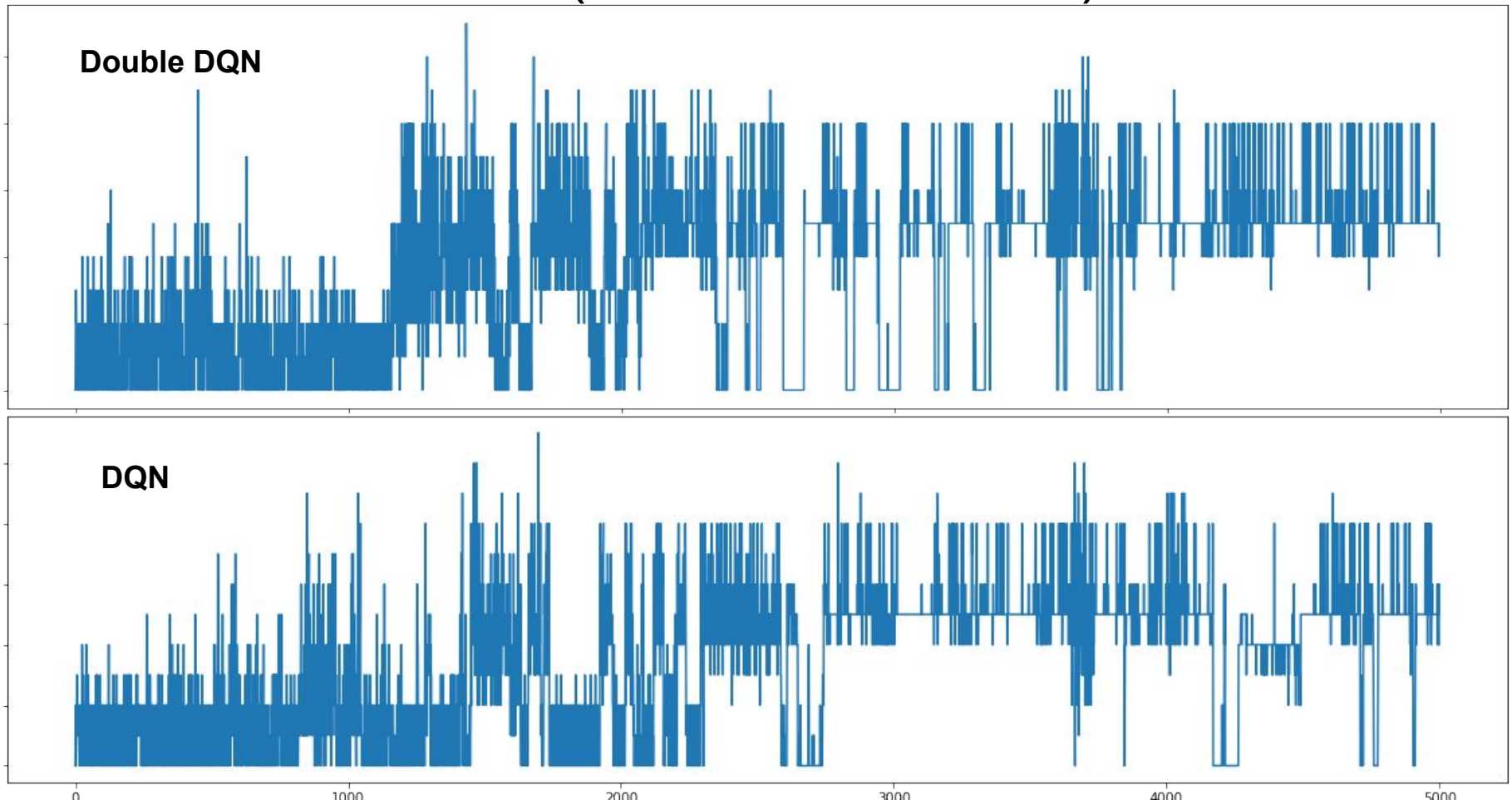


The Paper’s Results (Fully Trained GIFs)

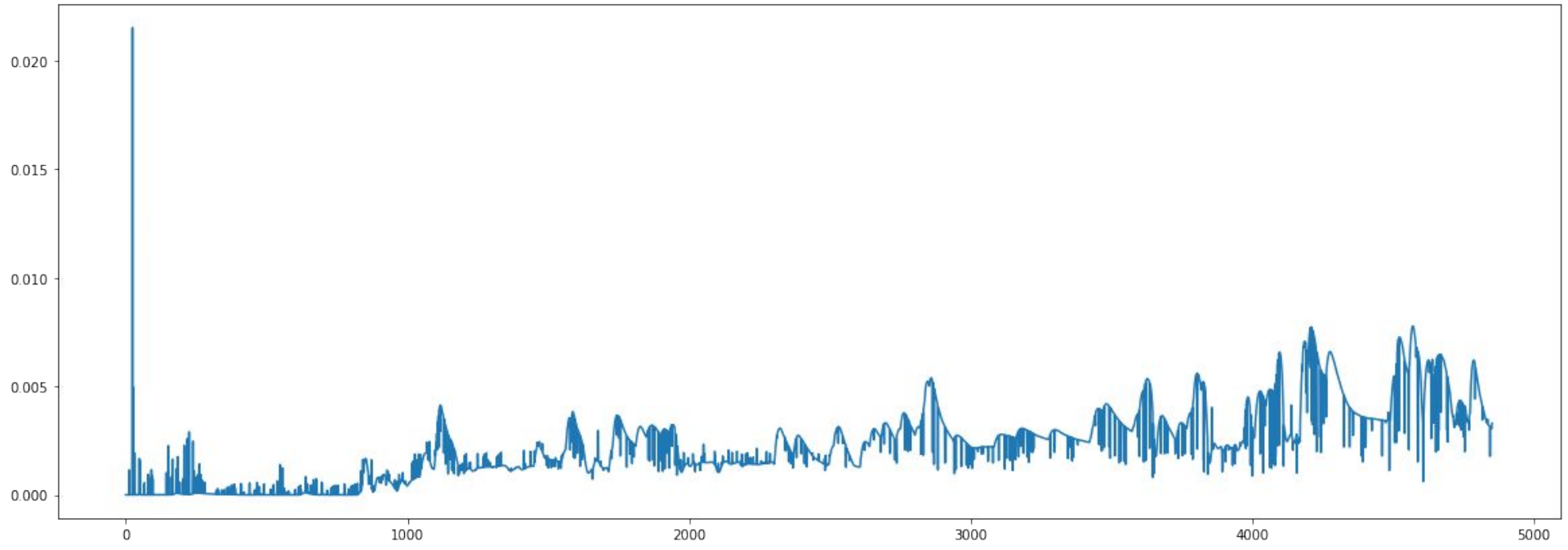
Understanding the Difference



Our Results (rewards over time)



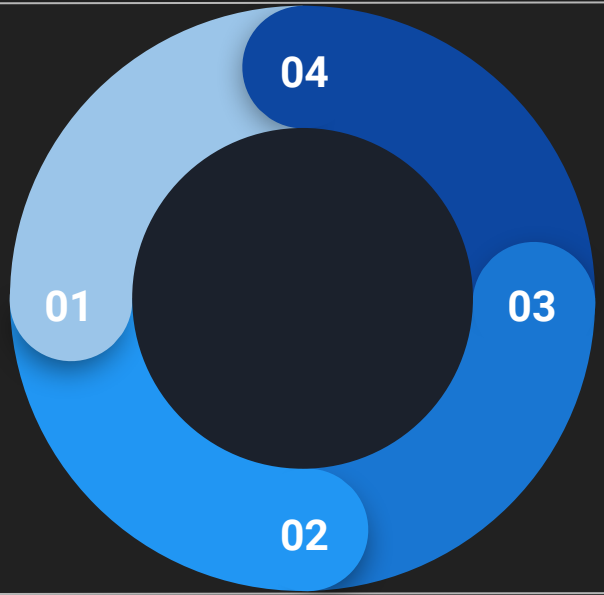
Our Results (losses over frames)



Further Improvements

Multi-step Returns

Noisy Networks



Prioritised Experience Replay

Dueling Networks Architecture


GAME OVER
Thank you for playing