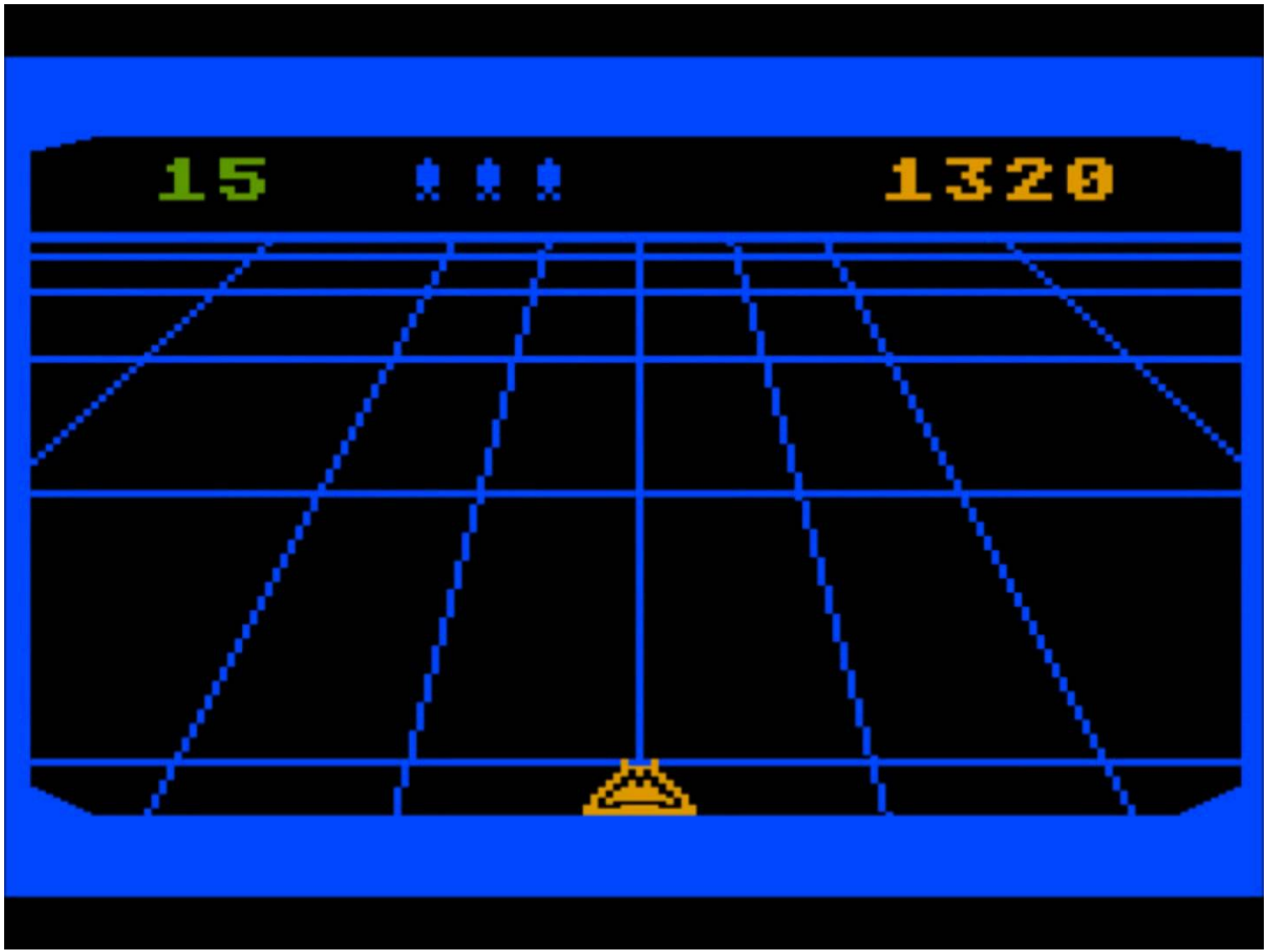
Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves,
Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller

February 18, 2015

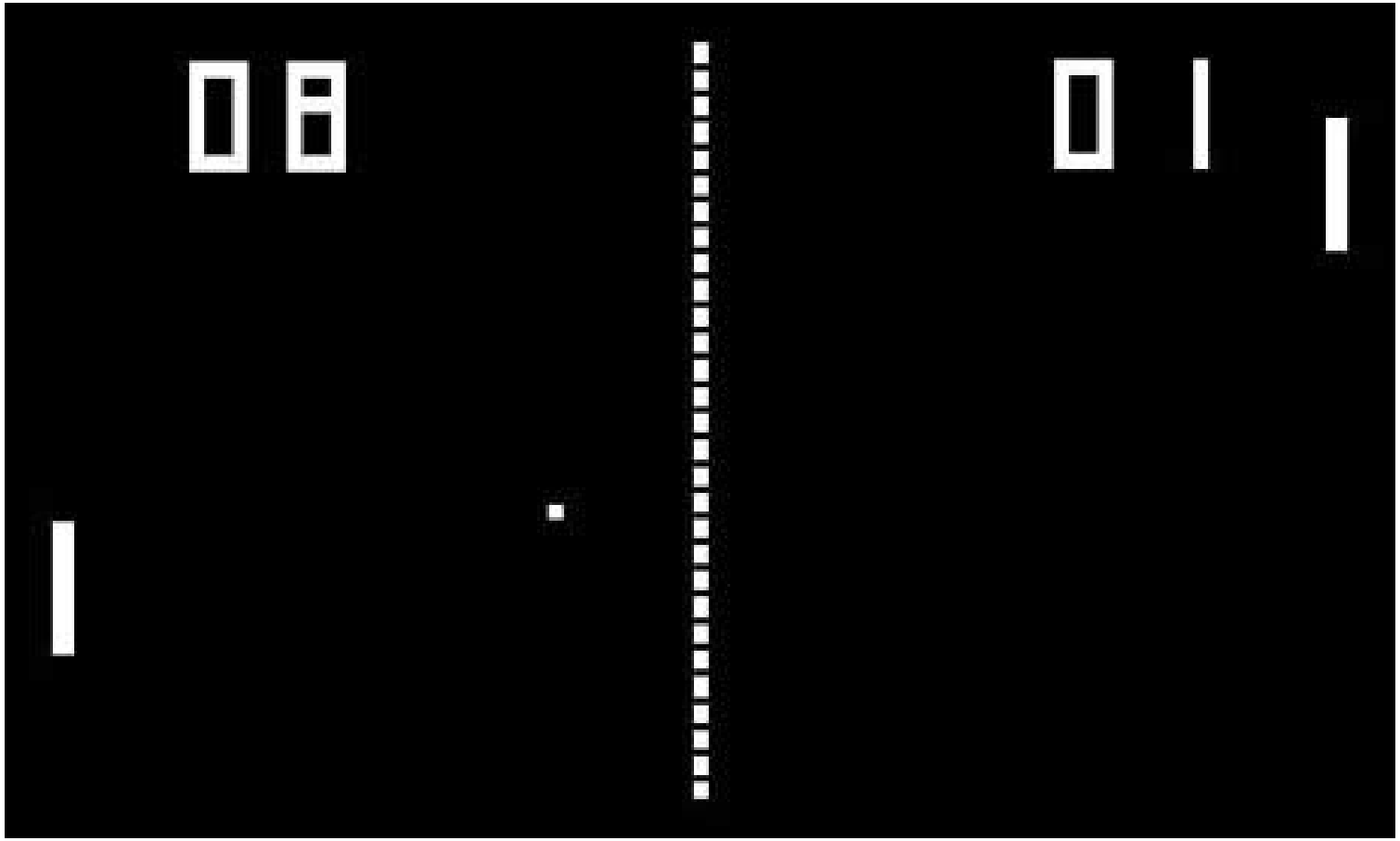
Atari Games



Atari Games

Atari Games

Atari Games



Atari Games

Atari Games

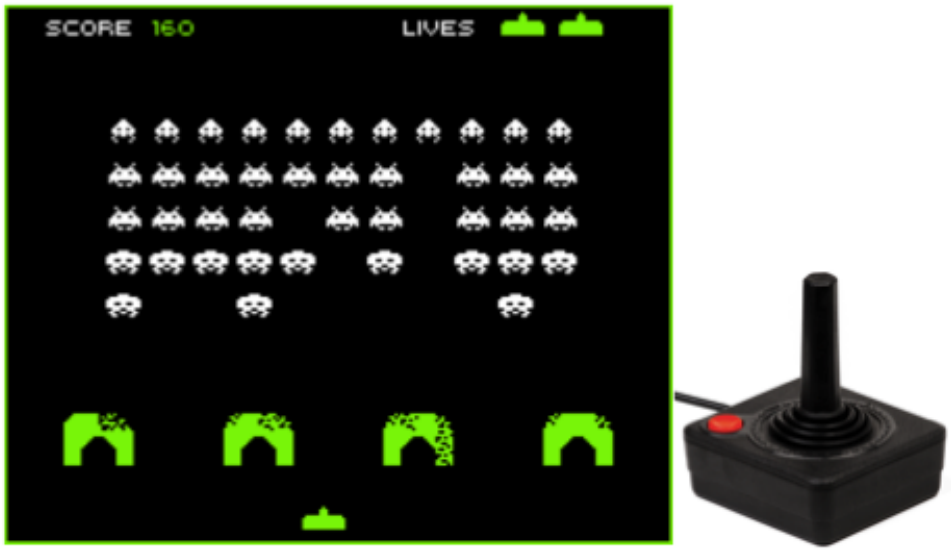

15460
OXYGEN
ACTIVISION
Play Seaquest at http://2600online.com/Seaquest.html

Atari Games

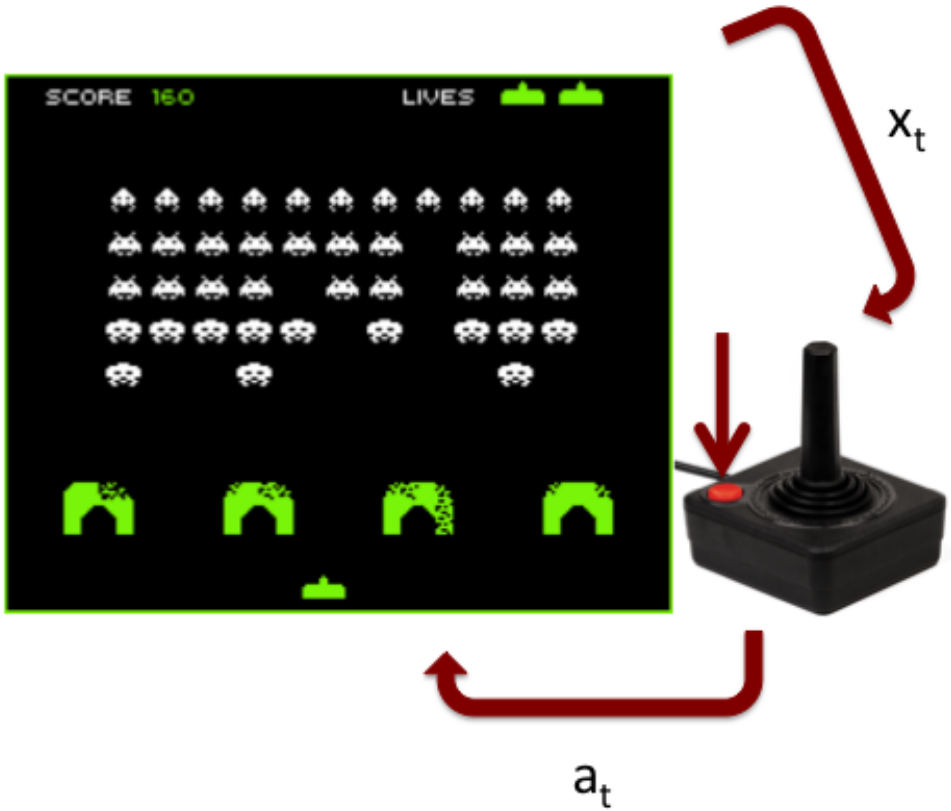
Problem Statement

1. Development of a deep learning model that learns to play Atari games.
2. The only input this model receives is the Atari emulator's output.
3. They use convolutional neural networks trained with a variant of Q-learning whose input is raw pixels and output is a value function estimating future rewards.

The Model



The Model



The Model

Sequence of states

- It is difficult to make sense of the visual input from only one frame. So the learning algorithm will take a sequence of images and actions as the state.
- $s_t = x_1, a_1, x_2, a_2, ..., a_{t-1}, x_t$
- They define the future discounted return at time t as:

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

- They define the optimal action-value function:

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

Bellman Equation and Function Approximation

- The Bellman equation is defined as:

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

- We would like to approximate $Q*(s,a) \approx Q(s,a;\theta)$
- We can use a neural network and fit it to the data by minimizing at each iteration:

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

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

Model fitting

1. The gradient of the loss function is:

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

2. We can instead compute Stochastic Gradient Descent
3. Instead of computing the expectation we only sample from the distribution of future states, we get Q-learning.

Preprocessing

- Atari frames are 210x160 pixel images in 128 color palette. So for each x_t image, they define a preprocessing stage where they turn images into a gray-scale, down-sampling the image to a 110x84 image. Then they crop the image down to a 84x84 region.
- Given a sequence of images and actions : $s_t = x_1, a_1, ...a_{t-1}, x_t$ they define $\phi(s_t)$ as only considering the last four frames and processing the corresponding images.
- For the neural network, they use an architecture in which there is a separate output unit for each action, and the state representation is the only input.

Deep Q-Learning with Experience Replay

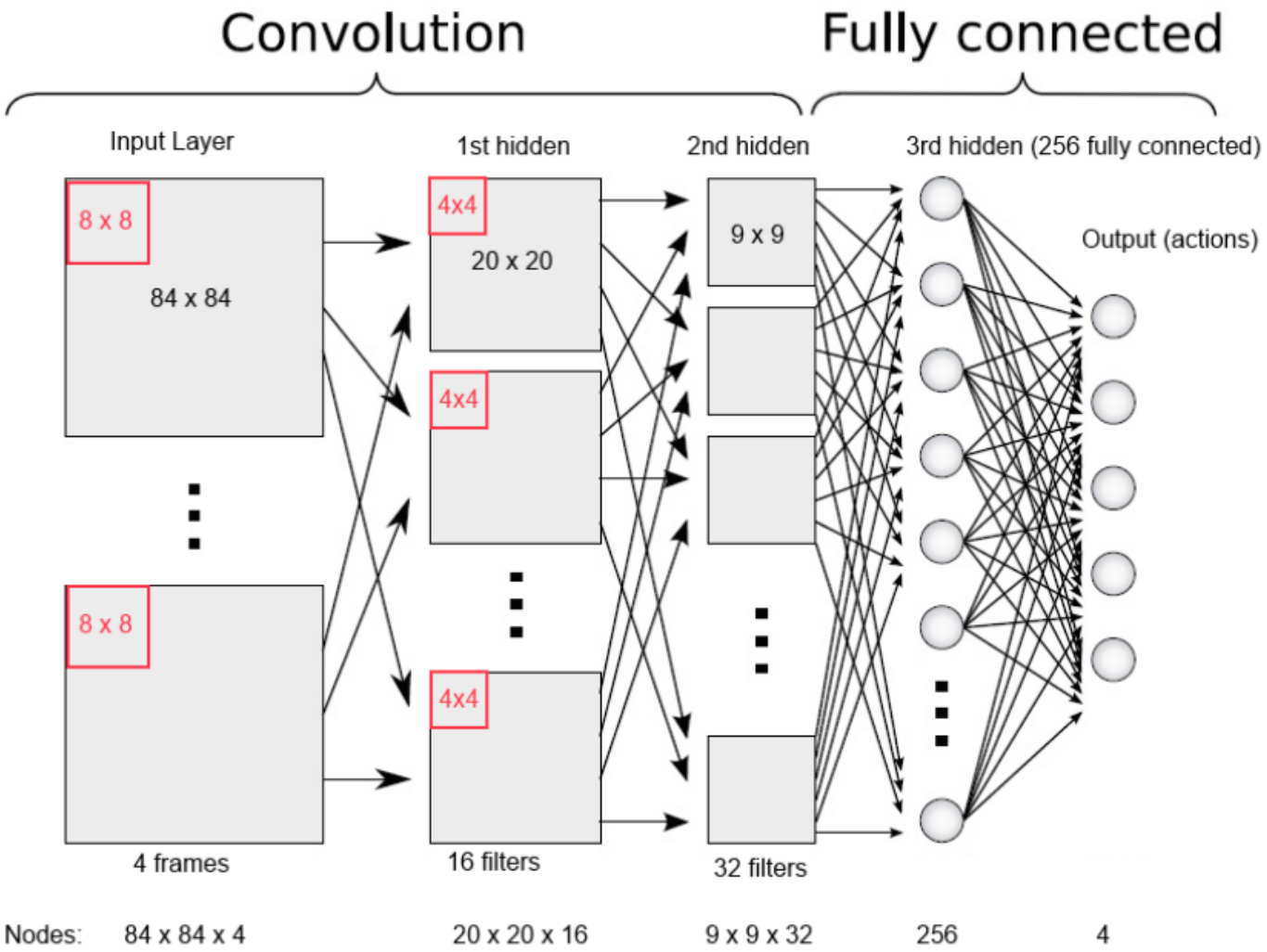
Algorithm 1 Deep Q-Learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N.
Initialize action-value function $\mathcal{Q}$ with random weights.
for *episode* $= 1$ to M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocess $\phi_1 = \phi(s_1)$.
  for $t = 1$ to T **do**
    With probability ϵ select a random action a_t.
    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

$$y_j = \begin{cases} r_j & \text{: for terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} \mathcal{Q}(\phi_{j+1}, a'; \theta) & \text{: o.w.} \end{cases}$$

    Perform a gradient descent step on $(y_j - \mathcal{Q}(\phi_j, a_j; \theta))^2$
  end for
end for

Architecture



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.

Comparison with previous algorithms and human players

	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 $\epsilon = 0.05$.

Epsilon Greedy

- Using epsilon-greedy may not be an issue for the absence of really delayed consequences.
- In general only optimizing for close time frames may be sufficient to optimize rewards.

Replay Memory

- For neural networks and other supervised learning algorithms it is important for the samples to be independent.
- By sampling from previous runs of the algorithm, they are able to break the correlation between states.