Human-level control through deep reinforcement learning

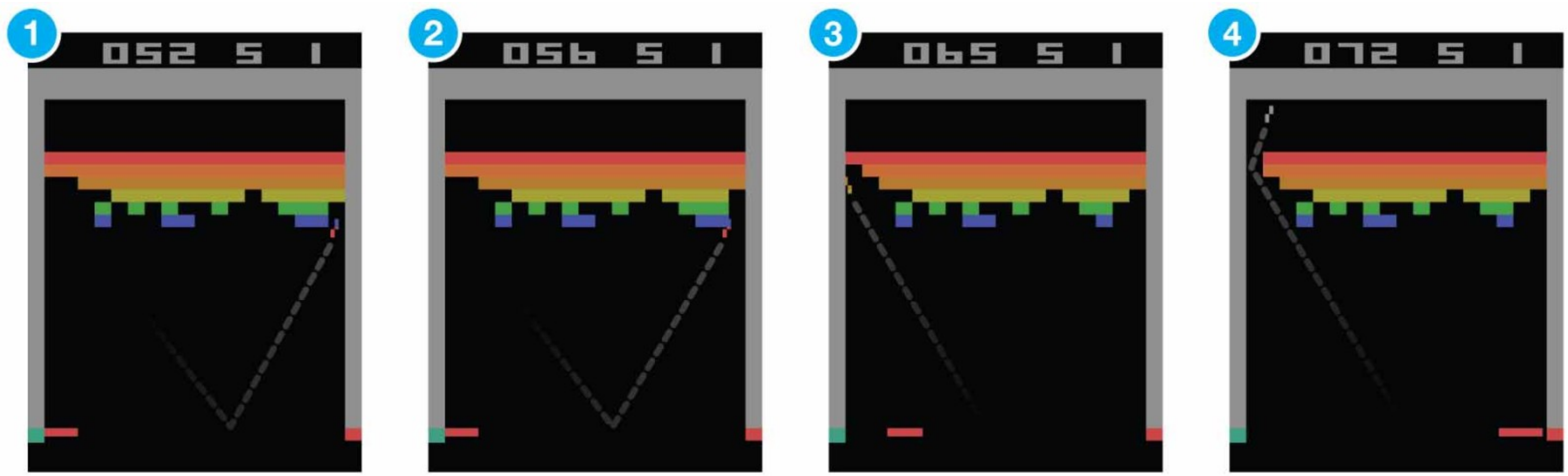
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

Agenda

- Problem
- Action-value function Q
- Paper’s additions to Q learning
- Training details
- Algorithm
- Results
- Summary and discussion

Problem

- Create an agent that would perform different, varied and challenging tasks
- Tasks: 49 games in the Atari 2600 platform



Agent and emulator

- Agent selects action a_t from $\{a_1, \ldots, a_K\}$ possible game actions
- Emulator executes action and returns:
  - Image for the current screen x_t
  - Game score → Δ game score for reward r_t
- Agent's goal: maximize future rewards $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$
  - Discounted by γ = 0.99

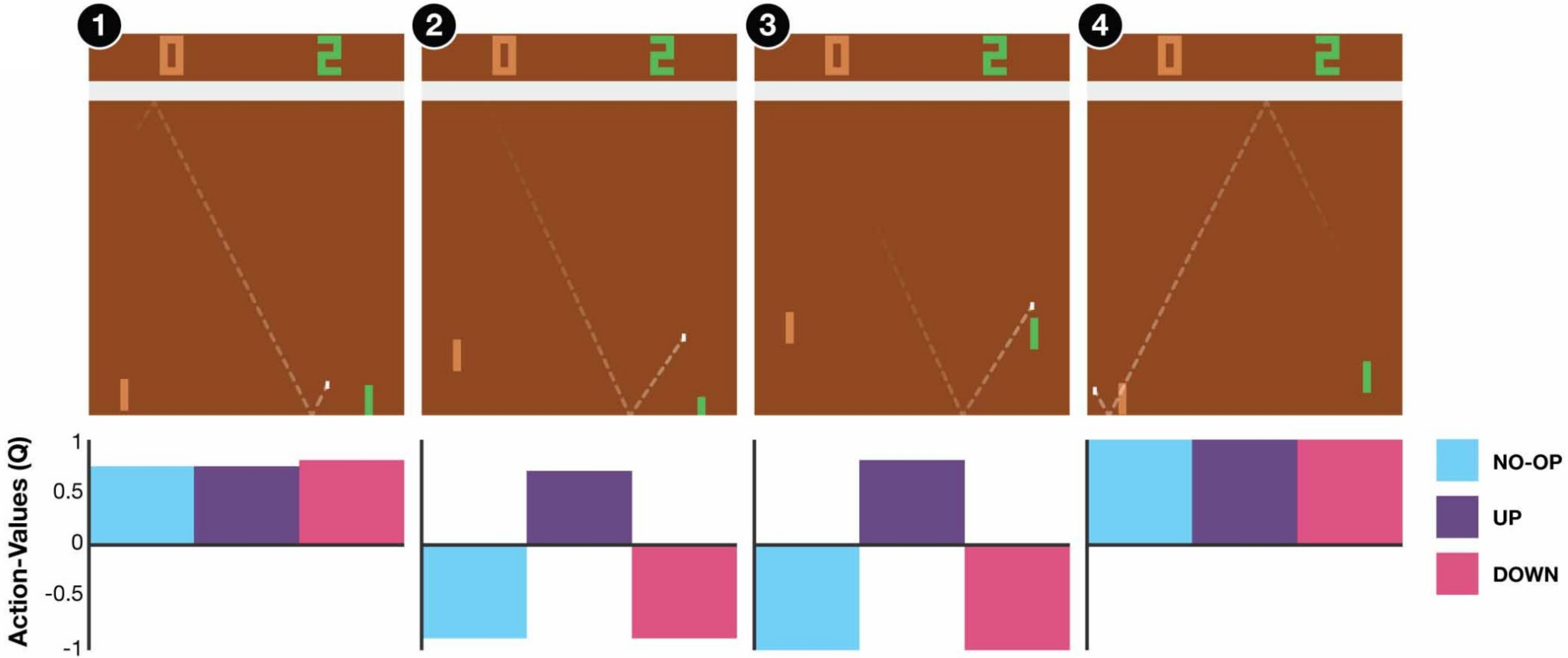
Action-value function *Q*

- Optimal action-value function

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

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

- Observed states: *s* this step, *s*' next step
- Action taken: *a* this step, *a*' next step
- π: policy mapping states to actions


1
2
3
4
0
2
Action-Values (Q)
1
0.5
0
-0.5
-1
NO-OP
UP
DOWN

Q-network: approximator for Q^*

- Instead of optimal target values

$$r + \gamma \max_{a'} Q^*(s', a')$$

use approximate target values

$$r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$$

- θ_i^-: parameters from some previous iteration
- Q-network: neural network function approximator with weights θ

Loss function

For iteration *i*

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

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

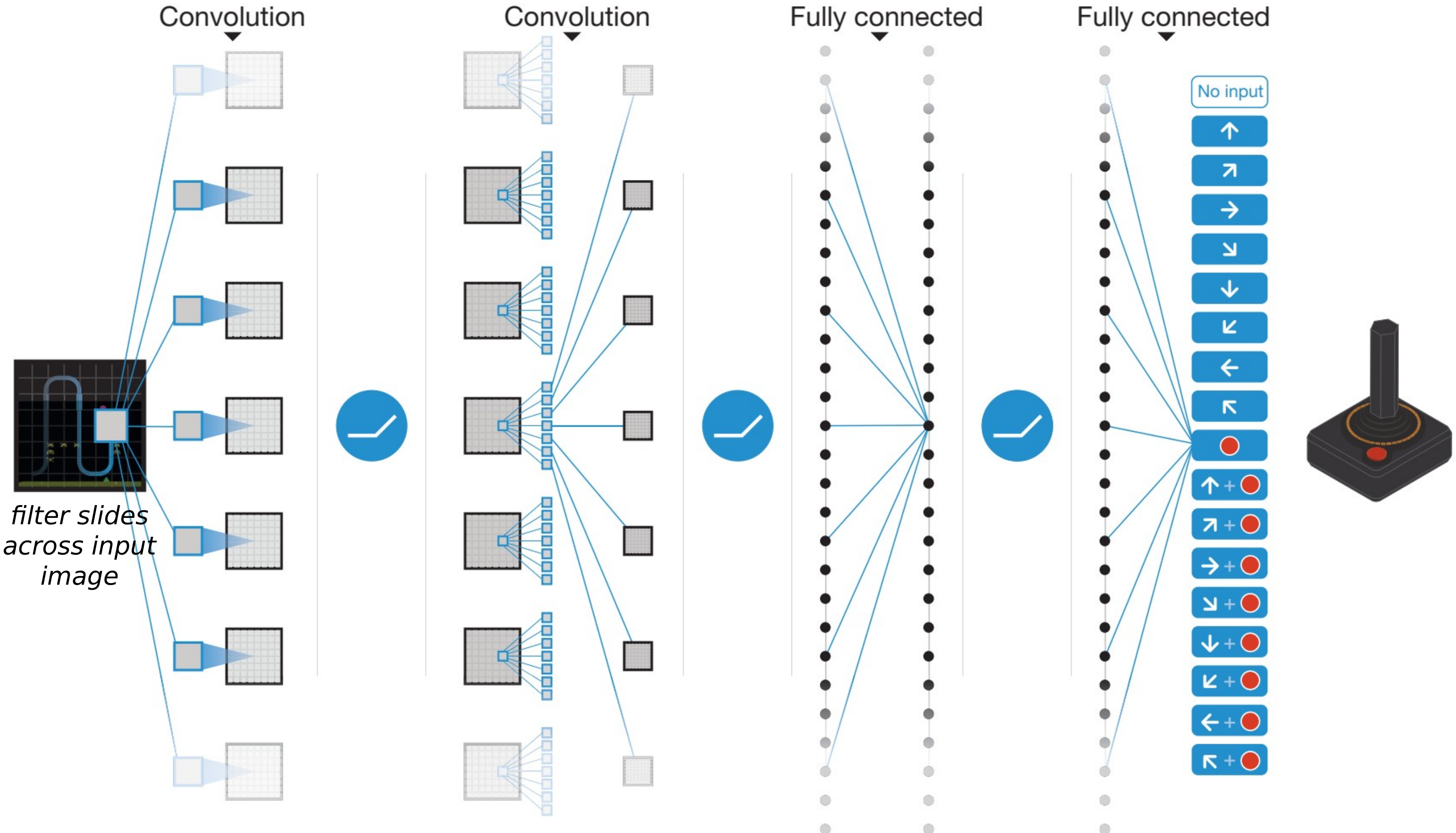
Paper’s additions to Q-learning

1. Convolutional network

2. Experience replay

3. Second target network

Input
84 × 84 pixels
× 4 frames
preprocessed image

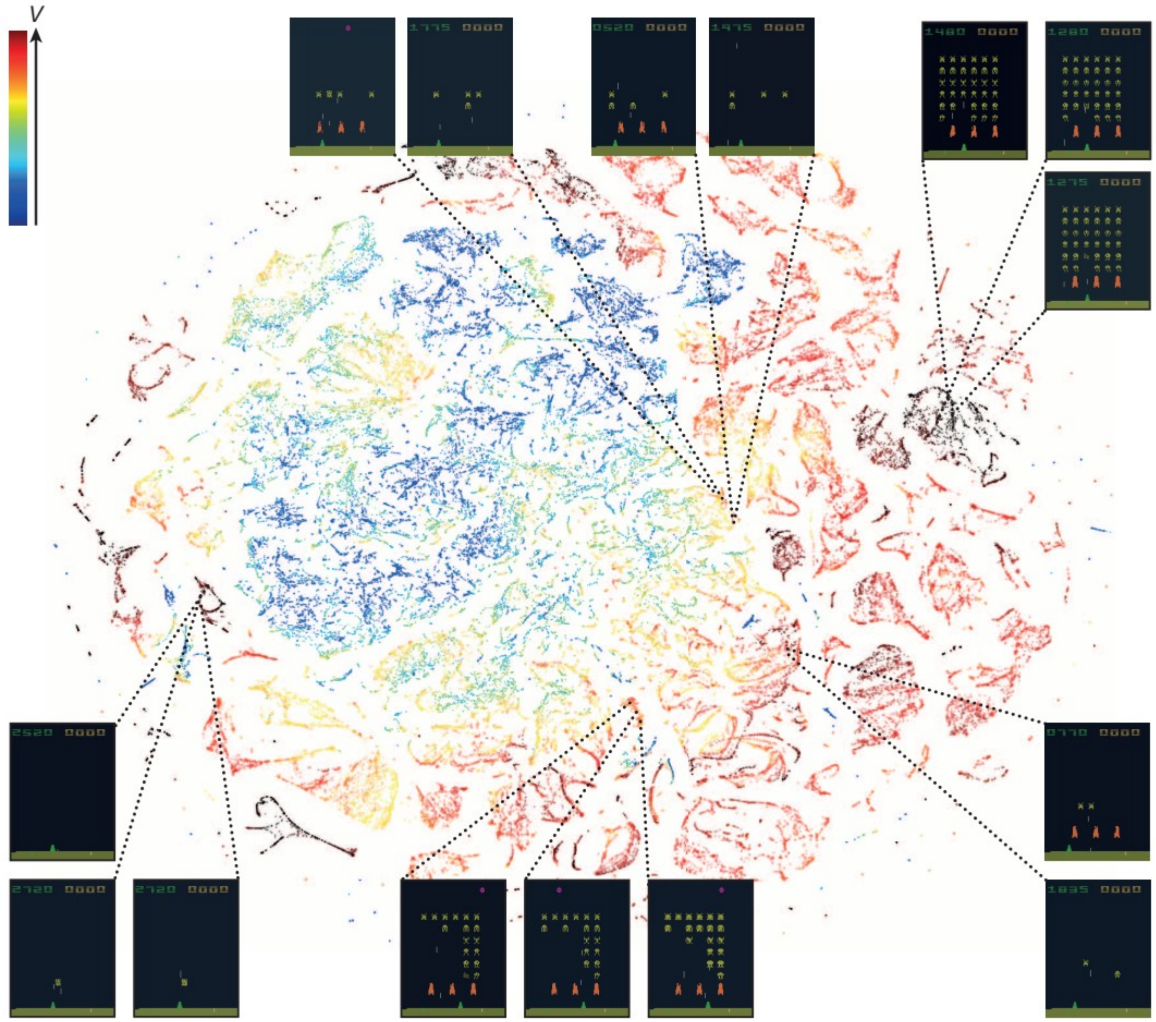


32 8 × 8 filters, stride 4

64 4 × 4 filters, stride 2

512 units

# of units = # of game actions



- Values of game states in the last hidden layer
- t-SNE
- After 2 hours of play
- Nearby points = perceptively similar states

2. Experience replay

- Store a history of 1 mln most recent (*s*, *a*, *r*, *s*') in replay memory
- Draw random minibatch of samples during training
- Stabilizes learning: prevents the agent from learning from only recent experiences
  – Which are highly correlated with current state
- Start populating after 50,000 initial random moves

3. Second target network $\hat{Q}$

- Use $\hat{Q}$ to generate targets y_j on each update
  - Instead of the main network Q
- Periodically set $\hat{Q} = Q$
- Stabilizes learning: makes divergence less likely
  - An update that increases $Q(s_t, a_t)$ often also increases $Q(s_{t+1}, a)\ \forall\, a$
  - So increases $y_j \rightarrow$ possible divergence

Effect of experience replay and target network

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

Training

- RMSProp, minibatch size 32
- ε-greedy policy
  - ε annealed from 1 to 0.1 in the first 1 mln frames
  - $\varepsilon = 0.1$ afterwards
- Trained for 50 million frames (≈38 days of playing)

Preprocess input

- Atari 2600 frames are 210 × 160 color images
- Preprocess:
  1. Max of each pixel color value over frame at t and $t - 1$ to remove flickering
  2. Rescale to 84 × 84 and convert to grayscale
  3. Stack 4 most recent frames

Other training elements

- Both rewards and errors clipped at [-1; 1]
- Select action on every 4^{th} frame, instead of every frame
  - Selected action repeated on skipped frames
  - Less expensive to run the emulator for one step than to have the agent select an action
  - The fastest a human can press the “fire” button is every 6^{th} frame
- Hyperparameters selected in “informal search” (not e.g. grid search)

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

    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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

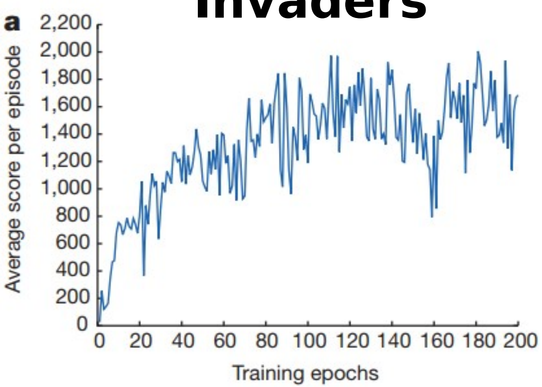
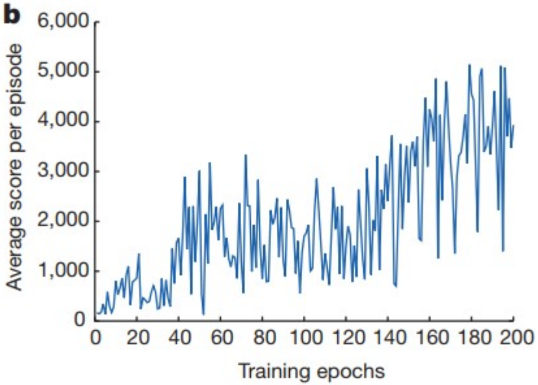
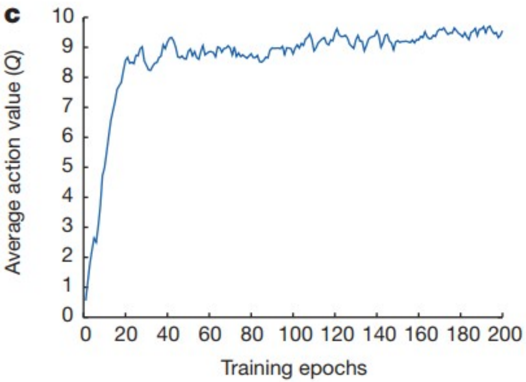
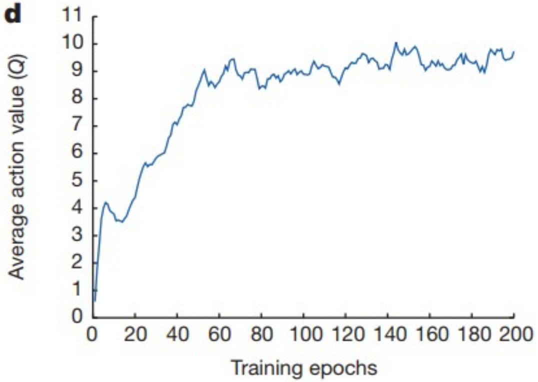
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    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 θ

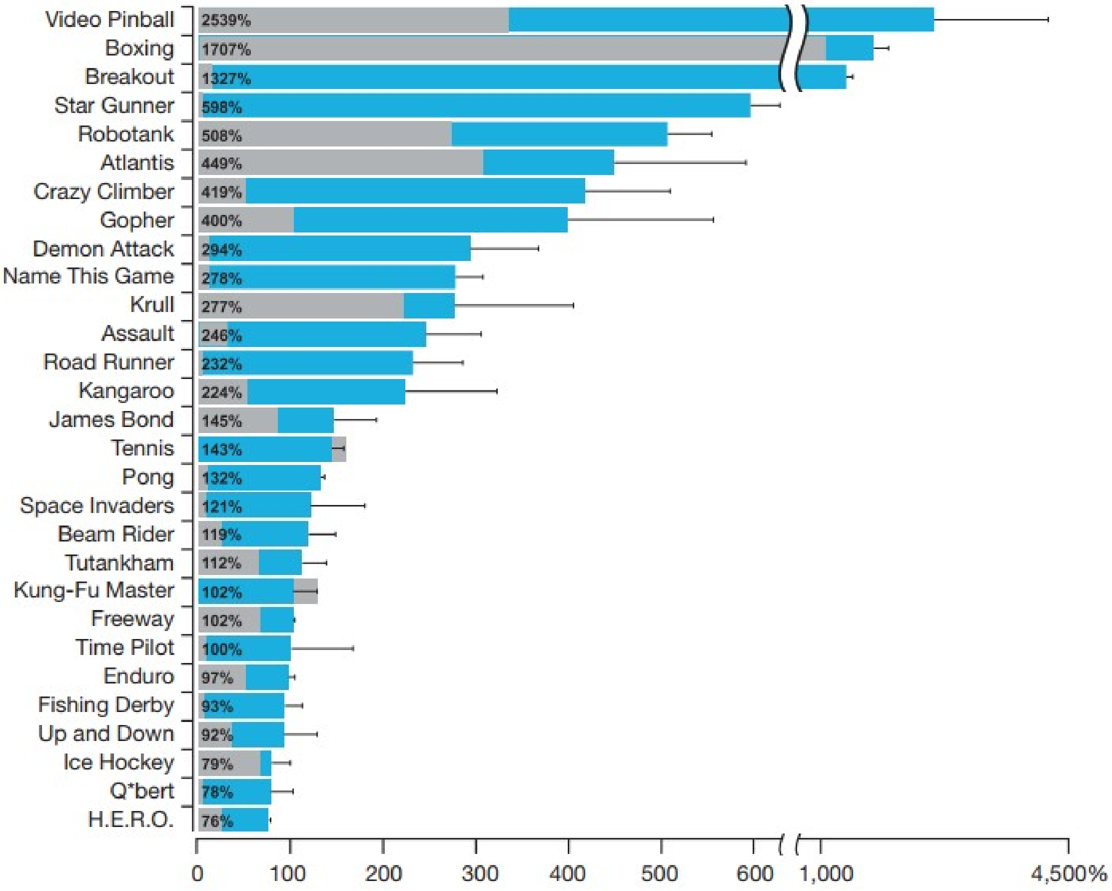
    Every C steps reset $\hat{Q} = Q$

  End For

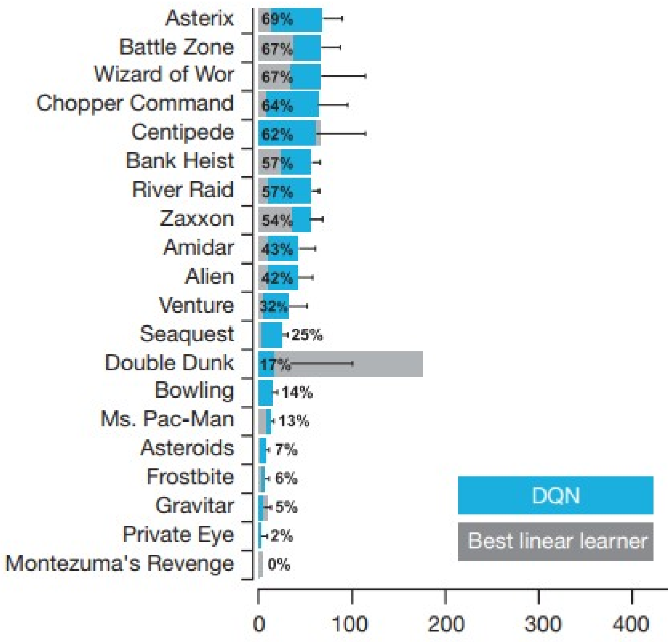
End For

	Space Invaders	**Seaquest**
Average score (ε=0.05, 520k frames)		
Average Q value (on holdout set of states)		
		

At human level or above



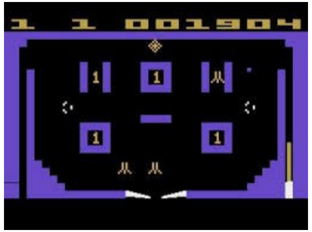
Below human level



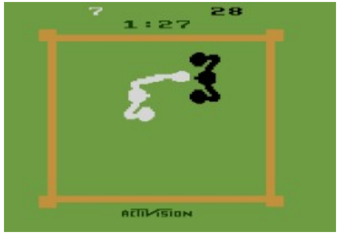
- 0%: random play
- 100%: professional human games tester
- Error bars: s.d. on 30 evaluation episodes
- Linear learner: single linear layer

Evaluation

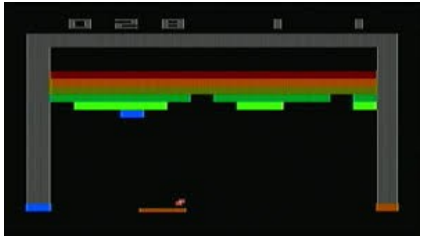
- Trained agents played each game 30 times for up to 5 mins each
  - Random number of initial "no-op"s up to 30



Pinball

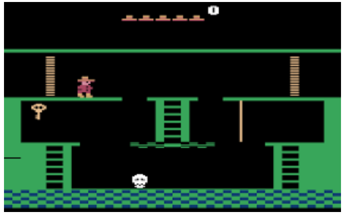


Boxing



Breakout

Star Gunner



Montezuma's Revenge

Private Eye

Gravitar

Ms. Pac-Man

Paper summary

- Combined Q-learning with a conv net
  - Able to use high-dimensional input
- Added techniques to stabilize learning:
  - Experience replay
  - Second network
- Result: generalizable agent
  - Learns in various different tasks (=Atari games)
  - Beats human performance in 29 of the 49 games

Discussion points

- Some games are more manageable for the agent than others. Why?
- What other tasks can this agent be applied to?
  - Anything in your line of work?
  - Would the architecture need to be tweaked?
- The agent starts with no clue about the environment it works in. How can this knowledge be used?
- Reward is highest game score. What other reward functions can be used here?
- 49 games = 49 trained agents. What can be done to generalize these agents?