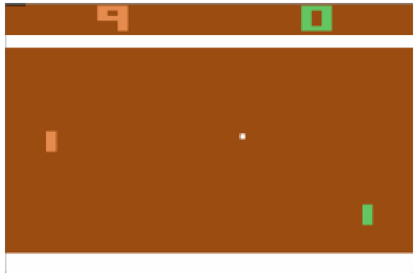
Human-level control through deep reinforcement learning

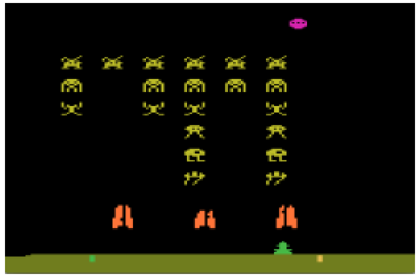
(Nature, 2015)

• Introduction	(2 slides)
• **RL basics**	(4 slides)
• **Q-value iteration**	(4 slides)
• **DQN = Deep Q-network**	(3 slides)
• Results	(1 slide)
• **Discussion/conclusions**	(3 slides)
• Code/tips	(1 slide)

DQN (overview)

- Mnih et al. introduced *Deep* Q-Network (*DQN*) algorithm, applied it to ATARI games
- Used deep learning / ConvNets, published in early stages of deep learning craze
- Popularized ATARI (Bellemare et al., 2013) as RL benchmark

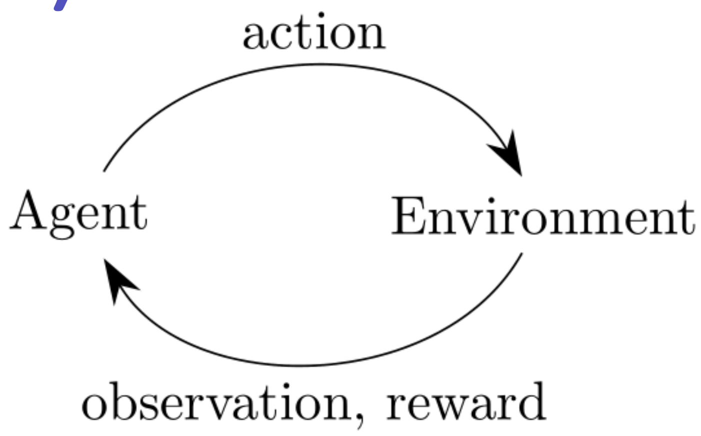





- Outperformed baseline methods, which used hand-crafted features

	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

What is Reinforcement Learning (RL)?



- ML ~ concerned with taking sequences of actions
- Try to maximize cumulative reward

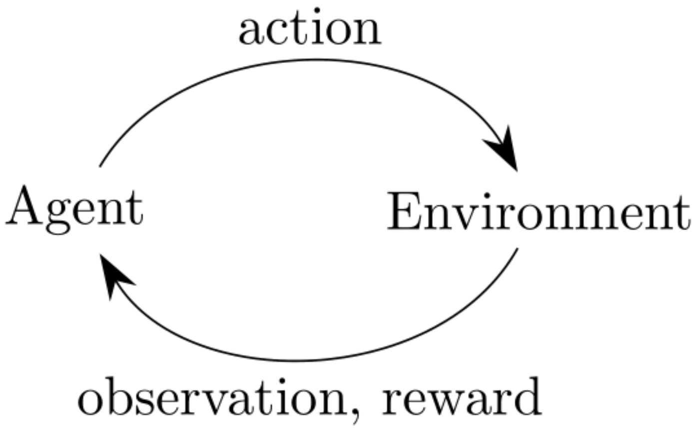
Robotics
Observation: camera images, joint angles
Actions: joint torques
Rewards: stay balanced, navigate, serve humans

Inventory management
Observation: current inventory levels
Actions: number of units of each item to purchase
Rewards: profit

Machine translation (sequential prediction)
Observation: words in source language
Action: emit word in target language
Reward: sentence-level accuracy metric

- Deep RL - use NN to approximate functions
  - Policies (select next action)
  - Value functions (measure goodness of states or state-action pairs)
  - Dynamics Models (predict next states and rewards)

Basic RL



- Markov Decision Process (MDP)
  - **S**: state space – set of states of the environment
  - **A**: action space – set of actions, which the agent selects from at each time step
  - **P**(r,s'|s,a) – a transition probability distribution (emit reward r and transition to s')

- Goal
  - find a policy π which maps states to actions
  - can be stochastic $\pi(a|s)$ or deterministic $a = \pi(s)$

Compared to supervised learning:

- don't have full access to function to be optimized
- must query it through interaction
- Interacting with a *stateful* world (input depends on previous actions)

Discounted setting

- Discount factor $0 \leq \gamma < 1$
- Preference for present rewards compared to future rewards
- Effective time horizon = $1 + \gamma + \gamma^2 + \cdots = 1/(1-\gamma)$

- Discounted problem can be obtained from original, by adding transitions to a "sink state"

$$\tilde{P}(s' \mid s, a) = \begin{cases} P(s' \mid s, a) \text{ with probability } \gamma \\ \text{sink state with probability } 1-\gamma \end{cases}$$

Basic concepts

- Policy optimization
  - maximize expected reward with respect to policy π

$$\underset{\pi}{\text{maximize}}\, \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$$

- Policy evaluation
  - compute expected **return** for fixed policy π
    - return = sum of future rewards in an episode
    - discounted return: $r_t + \gamma\, r_{t+1} + \gamma^2 r_{t+2} + \ \ldots$
    - undiscounted return: $r_t +\ r_{t+1} + \cdots +\ r_{T-1} + V(s_T)$
  - state value function

$$V^\pi(s) = \mathbb{E}_\pi\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s\right] \\ = \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a)\right]$$

  - state-action value function / Q function

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s, a_0 = a\right]$$

Bellman equations

$$
\begin{aligned}
Q^\pi(s_0, a_0) &= \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} \left[r_0 + \gamma V^\pi(s_1) \right] \\
&= \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} \left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} \left[Q^\pi(s_1, a_1) \right] \right] \\
&= \mathbb{E}_{s_1, a_1 \ldots, s_k, a_k \mid s_0, a_0} \left[r_0 + \gamma r_1 + \cdots + \gamma^{k-1} r_{k-1} + \gamma^k Q^\pi(s_k, a_k) \right]
\end{aligned}
$$

- Bellman backup operator $[\mathcal{T}^\pi Q](s_0, a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0, a_0)} \left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi} \left[Q(s_1, a_1) \right] \right]$
- Q^π is a fixed point of this operator $\mathcal{T}^\pi Q^\pi = Q^\pi$
- AND if we apply it repeatedly to any initial Q, the series converges to Q^π

$$
Q, \; \mathcal{T}^\pi Q, \; (\mathcal{T}^\pi)^2 Q, \; (\mathcal{T}^\pi)^3 Q, \; \cdots \to Q^\pi
$$

Finding the optimal Q

- Let π^* denote an optimal policy
- Define $Q^* = Q^{\pi^*}$, which satisfies $Q^*(s,a) = \max_\pi Q^\pi(s,a)$
- π^* satisfies $\pi^*(s) = \arg\max_a Q^*(s,a)$

- Then Bellman equation $$Q^\pi(s_0,a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0,a_0)}\left[r_0 + \gamma \mathbb{E}_{a_1 \sim \pi}\left[Q^\pi(s_1,a_1)\right]\right]$$
  becomes $$Q^*(s_0,a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0,a_0)}\left[r_0 + \gamma \max_{a_1} Q^*(s_1,a_1)\right]$$

  (optimal strategy is to select a1 that maximizes the expected value)

$$[\mathcal{T}Q](s_0,a_0) = \mathbb{E}_{s_1 \sim P(s_1 \mid s_0,a_0)}\left[r_0 + \gamma \max_{a_1} Q(s_1,a_1)\right] \qquad Q,\ \mathcal{T}Q,\ \mathcal{T}^2Q,\ \cdots \to Q^*$$

Q-value iteration

Algorithm 1 Q-Value Iteration

Initialize $Q^{(0)}$

for $n = 0, 1, 2, \ldots$ until termination condition **do**

  $Q^{(n+1)} = \mathcal{T}Q^{(n)}$

end for

$$[\mathcal{T}Q](s,a) = \mathbb{E}_{s_1}\left[r_0 + \gamma \max_{a_1} Q(s_1, a_1) \,\middle|\, s_0 = s, a_0 = a \right]$$

- we can compute greedily max Q(s,a)
- without knowing transition probabilities (model-free RL)

Q-Value iteration + Function Approximation: *Batch Method*

Parameterize Q-function with a neural network Q_θ

Backup estimate $\widehat{\mathcal{T}Q}_t = r_t + \max_{a_{t+1}} \gamma Q(s_{t+1}, a_{t+1})$

To approximate $Q \leftarrow \widehat{\mathcal{T}Q}$, solve minimize$_\theta \sum_t \left\| Q_\theta(s_t, a_t) - \widehat{\mathcal{T}Q}_t \right\|^2$

can compute estimator of backup without knowing P using **OFF-POLICY** data, a single trasition (s,a,r,s')

Algorithm 2 Neural-Fitted Q-Iteration (NFQ)[1]

Initialize $\theta^{(0)}$.
for $n = 0, 1, 2, dots$ **do**
  Run policy for K timesteps using some policy $\pi^{(n)}$.
  $\theta^{(n+1)} = \text{minimize}_\theta \sum_t \left(\widehat{\mathcal{T}Q_{\theta^{(n)}}}_t - Q_\theta(s_t, a_t) \right)^2$
end for

Q-Value iteration + Function Approximation *Online/Incremental Method*

Algorithm 3 Watkins' Q-learning / Incremental Q-Value Iteration

Initialize $\theta^{(0)}$.

for $n = 0, 1, 2, dots$ **do**

  Run policy for K timesteps using some policy $\pi^{(n)}$.

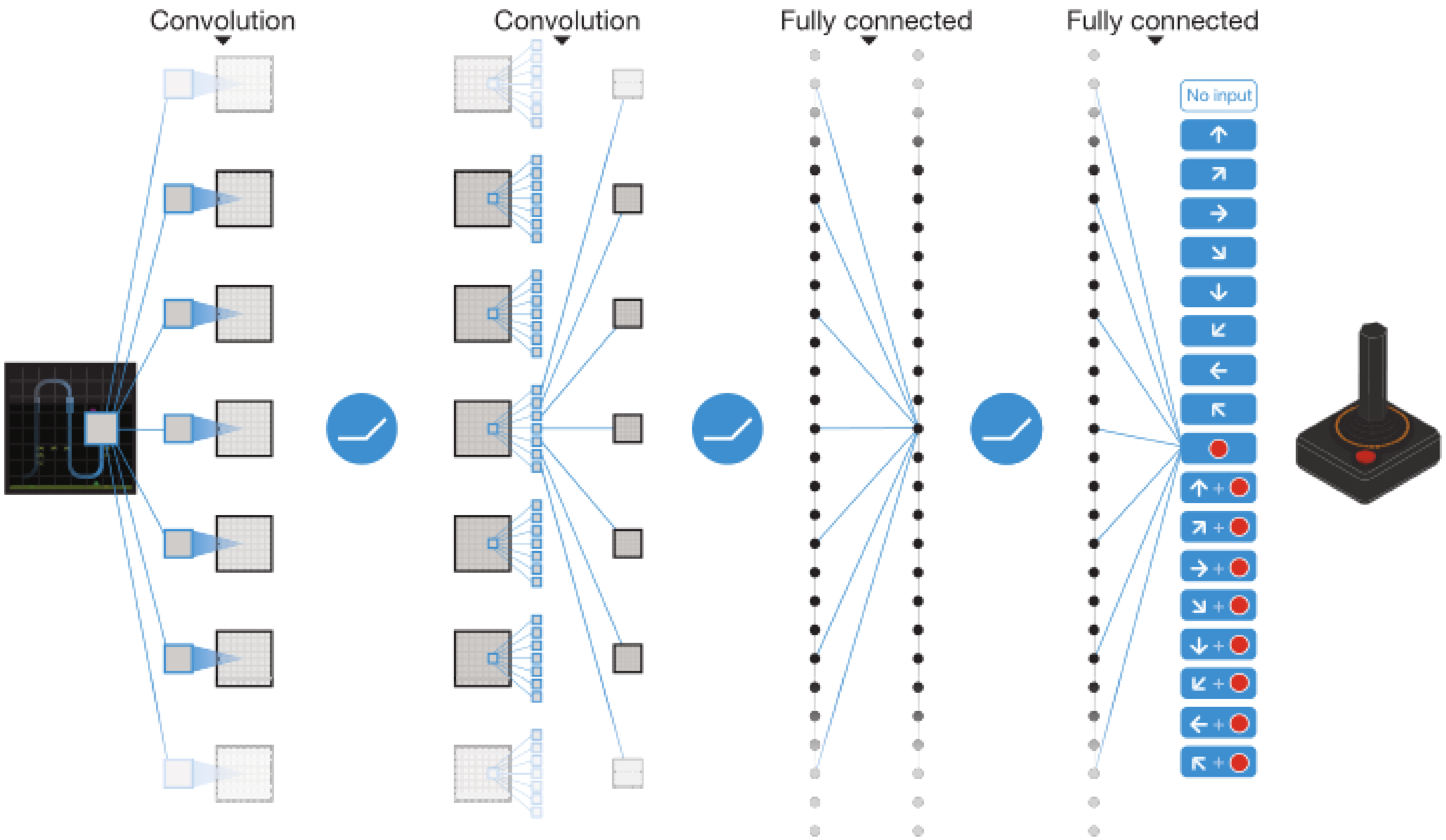
  $g^{(n)} = \nabla_\theta \sum_t \left(\widehat{\mathcal{T}Q}_t - Q_\theta(s_t, a_t)\right)^2$

  $\theta^{(n+1)} = \theta^{(n)} - \alpha g^{(n)}$ (SGD update)

end for

DQN (network)

Input image
84x84x4



32 filters
8x8, stride 4

64 filters
3x3, stride 1

512 units

4-18 units

DQN (algorithm)

- Hybrid of online + batch Q-value iteration
- Transition (s,a,r,s')
  ϕ is processed s
- Interleaves optimization with data collection

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 $(\phi_t, a_t, r_t, \phi_{t+1})$ 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

Key concepts

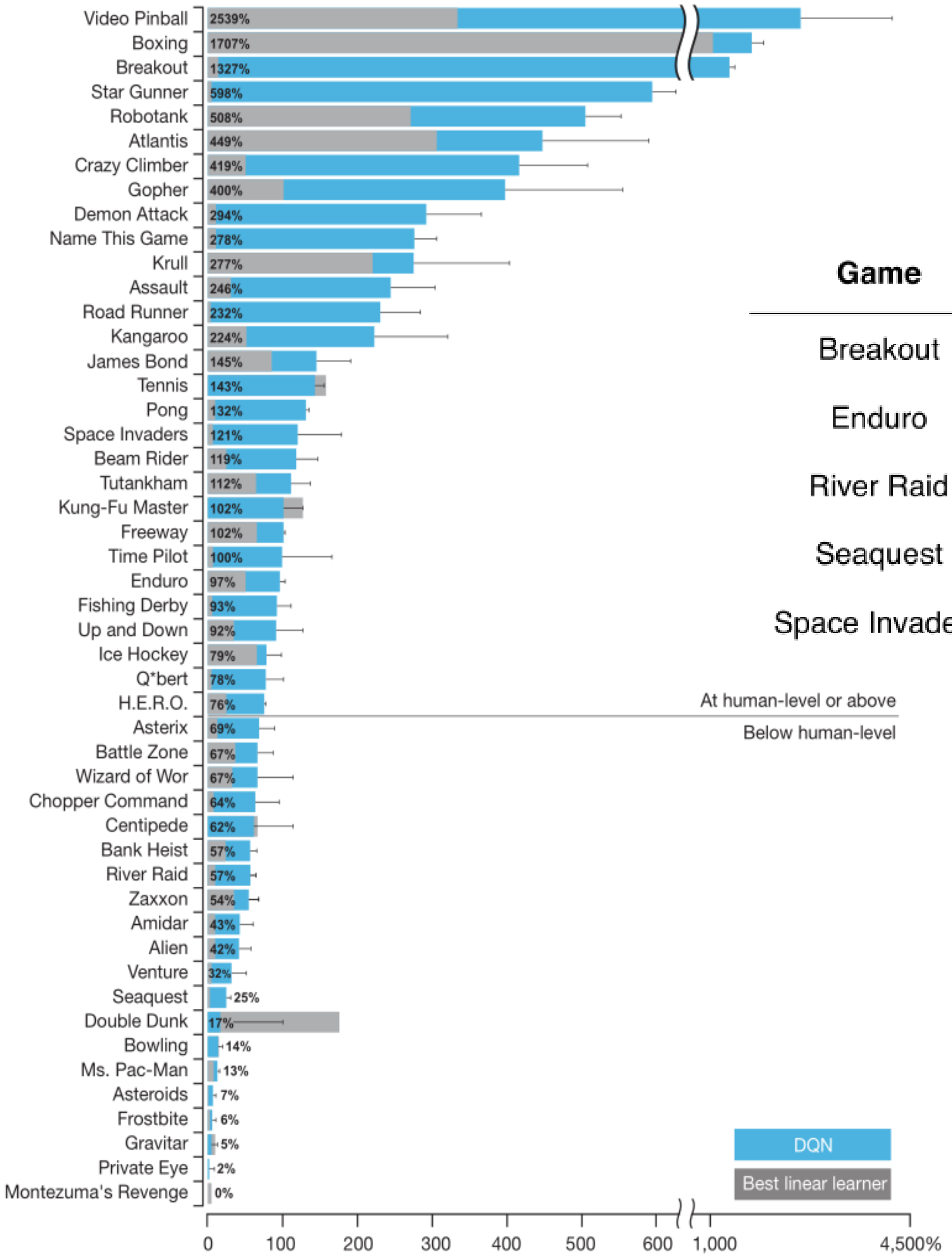
- Replay memory = history of last N transitions
  - Is it valid? yes: Q-function backup can be done with off-policy data
  - Each transition (s,a,r,s') seen many times

    => better data efficiency, reward propagation

    (example)
  - History contains data form many past policies, derived from $Q^{(n)}, Q^{(n-1)}, Q^{(n-2)}$, … and changes slowly => increased stability

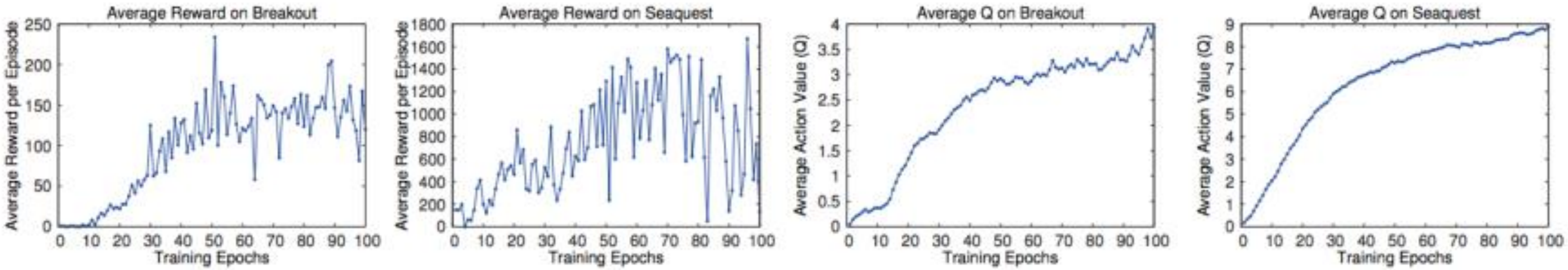
- Target network
  - Gets updated infrequently, while current Q is updated constantly by SGD
  - Why not use Q as backup target?
  - Fixed target $\mathrm{T}Q^{(n)}$ easier to reach than chasing a moving target

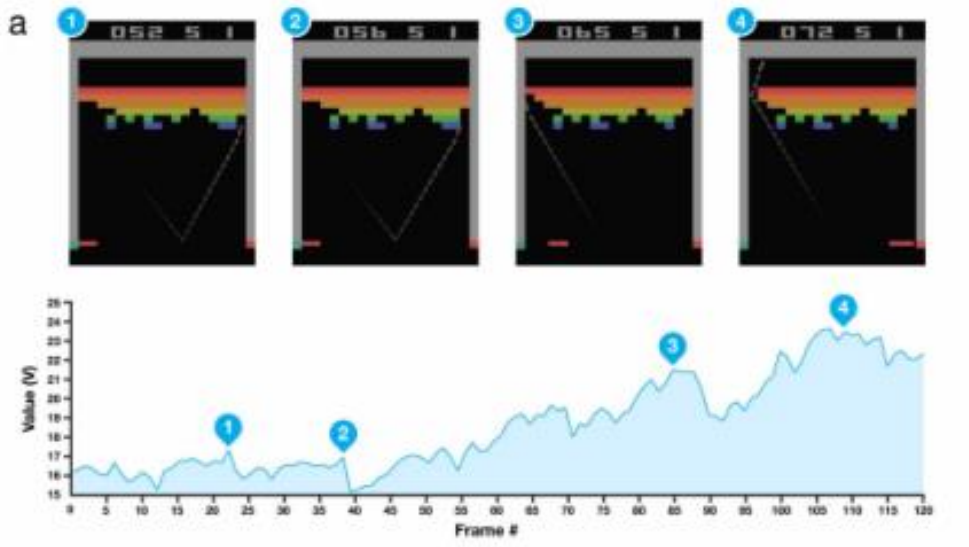
Results

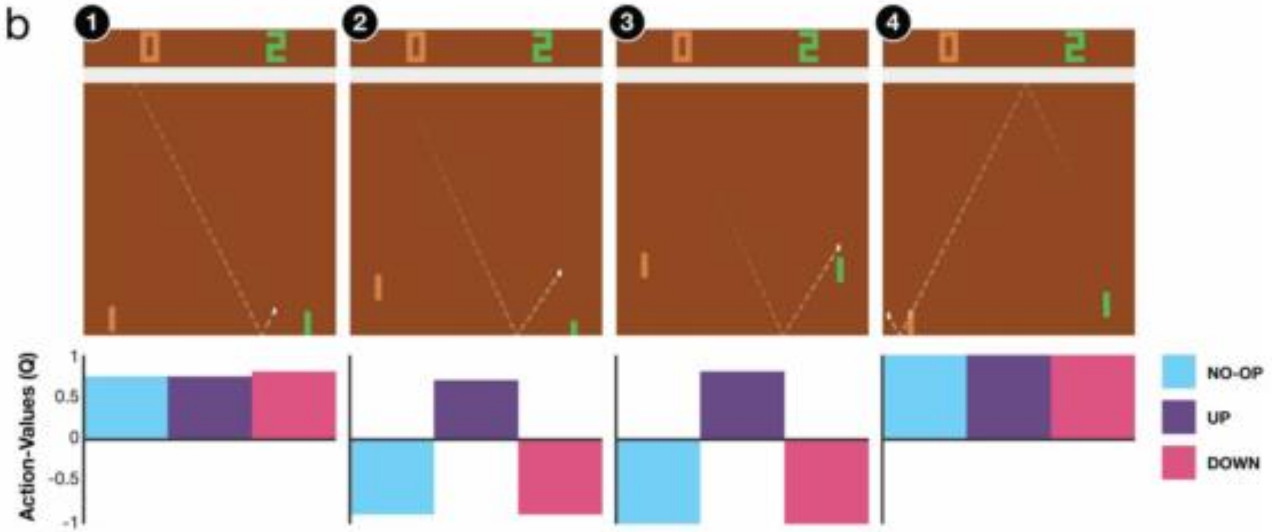


Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

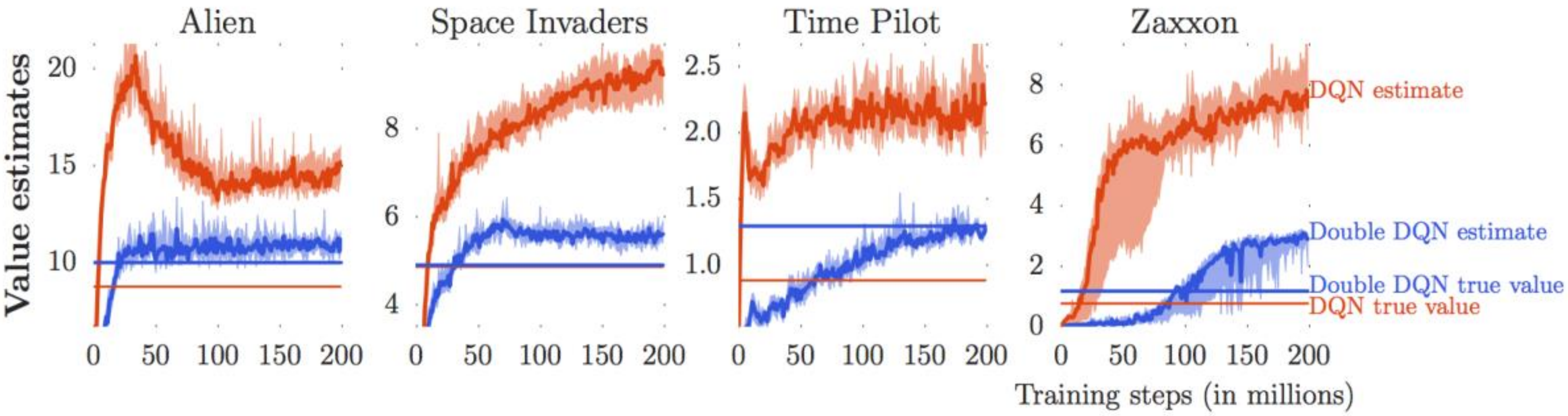
Are Q-values Meaningful?







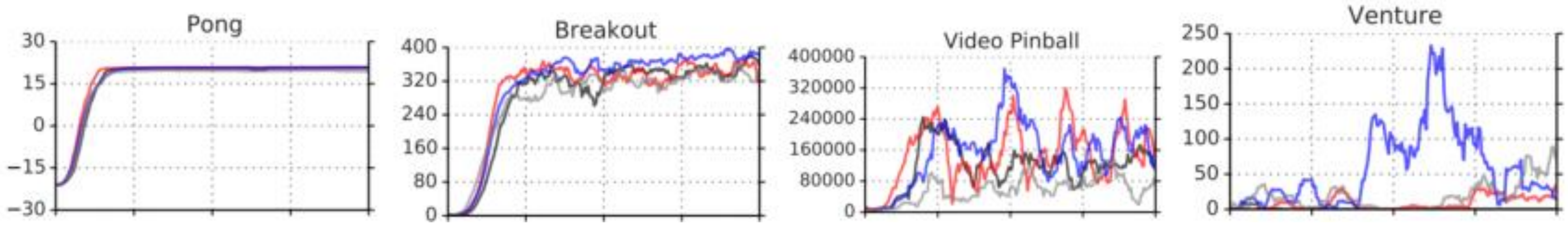
… but:



- Q function severely overestimating returns!
  - Q value can shift a lot without much impact to the loss
  - Differences between Q of different actions more important than their absolute value!

Conclusions

- Single architecture – control policies 49 different environments (same algorithm, network architecture and hyperparameters)
- Minimal prior knowledge, only pixels and score as input
- Not equally reliable on all tasks



- Improvements:
  - Double DQN (closer DQN estimates to true values)
  - Dueling nets (separately estimate V and A) $Q(s, a) = V(s) + A(s, a)$
  - Prioritized replay (favor *important* time steps – with large gradients)

Some practical tips

- Large replay buffers (~1 M) improve robustness
- Memory efficiency is key! use uint8, don't duplicate data etc.
- Converges slowly: ATARI often needs 10-40M frames (hours on a GPU) >> random policy
- Double DQN (2 lines change) but significant improvement
- Run at least 2-3 different seeds when experimenting
- Learning rate scheduling is important (high rates in initial exploration)

<u>**CODE:**</u>

Torch https://github.com/kuz/DeepMind-Atari-Deep-Q-Learner
Keras: Theano/Tensorflow https://yanpanlau.github.io/2016/07/10/FlappyBird-Keras.html
DL4J: Scala https://rubenfiszel.github.io/posts/rl4j/2016-08-24-Reinforcement-Learning-and-DQN.html