Human-level control through deep reinforcement learning

Presented by Bowen Xu

Acknowledgment

Slides from **Jiang Guo** available at:
http://ir.hit.edu.cn/~jguo/docs/notes/dqn-atari.pdf

Slides from **Dong-Kyoung Kye** available at: http://vi.snu.ac.kr/xe/

Towards General Artificial Intelligence

- **Playing Atari with Deep Reinforcement Learning**. *ArXiv (2013)*
  - 7 Atari games
  - The first step towards "General Artificial Intelligence"

- DeepMind got acquired by @Google (2014)

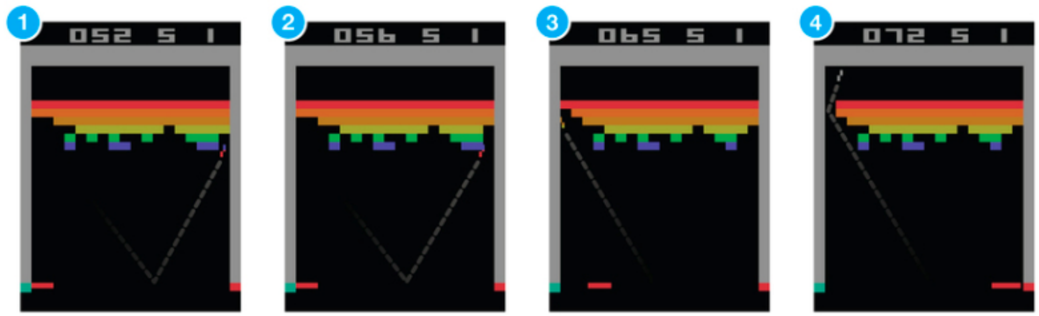
- **Human-level control through deep reinforcement learning**. *Nature (2015)*
  - 49 Atari games
  - Google patented "Deep Reinforcement Learning"

Key Concepts

- Reinforcement Learning
- Markov Decision Process
- Discounted Future Reward
- Q-Learning
- Deep Q Network
- Exploration-Exploitation
- Experience Replay
- Deep Q-learning Algorithm

Reinforcement Learning

- Example: breakout (one of the Atari games)

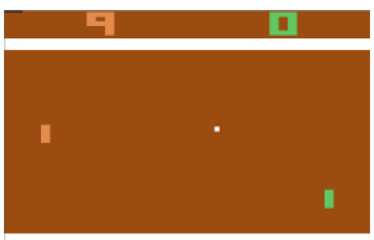


- Suppose you want to teach an agent (e.g. NN) to play this game
  - Supervised training (expert players play a million times) *That's not how we learn!*
  - **Reinforcement learning**

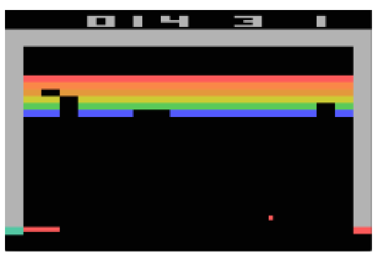
Reinforcement Learning

ML

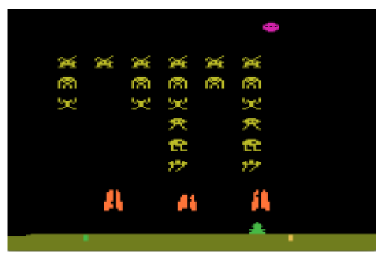
Supervised Learning	Target label for each training example
Reinforcement Learning	Sparse and time-delayed labels
Unsupervised Learning	No label at all



Pong



Breakout

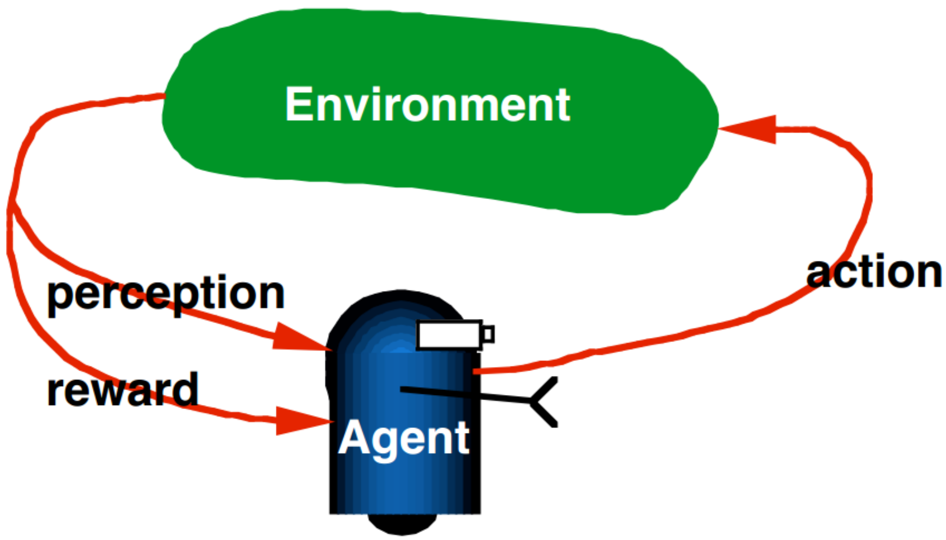


Space Invaders

Seaquest

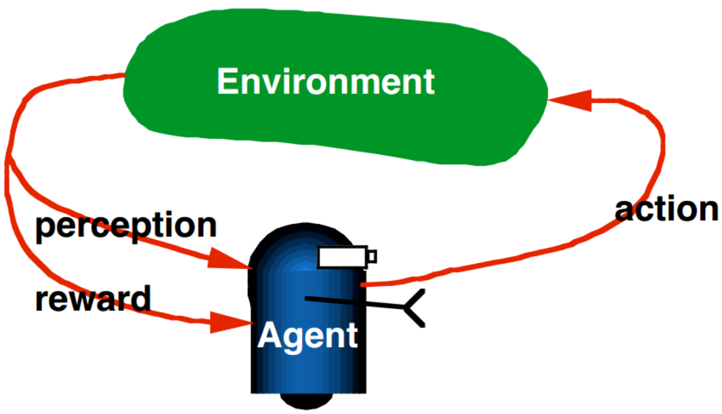
Beam Rider

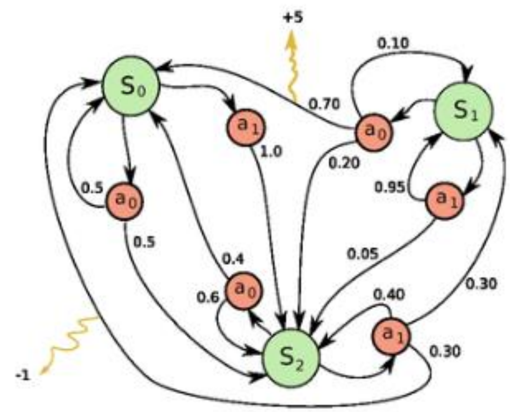
RL is Learning from Interaction



RL is like Life!

Markov Decision Process





$$s_0, a_0, r_1, s_1, a_1, r_2, \dots, s_{n-1}, a_{n-1}, r_n, s_n$$

state

action

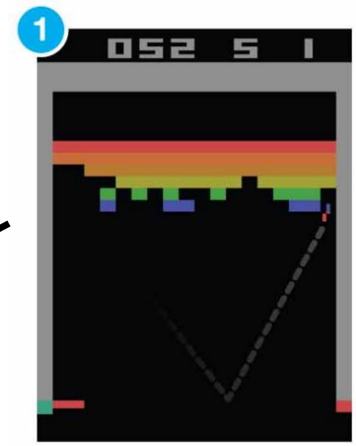
reward

Terminal state

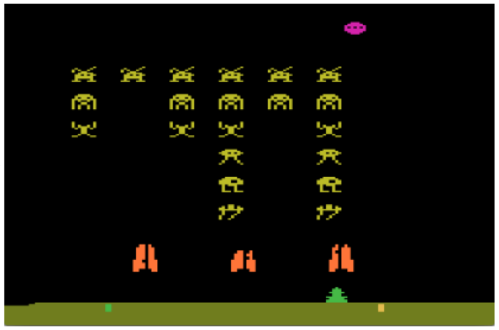
State Representation

Think about the **Breakout** game

- How to define a state?



- Location of the paddle
- Location/direction of the ball
- Presence/absence of each individual brick



Let's make it more universal!

Screen pixels

Value Function

MDP

$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{n-1}, a_{n-1}, r_n, s_n$

- Future reward

$$R = r_1 + r_2 + r_3 + \cdots + r_n$$

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

- Discounted future reward (environment is stochastic)

$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \\ &= r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \cdots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

- A good strategy for an agent would be to always choose an action that maximizes the (discounted) future reward

Value-Action Function

- We define a $Q(s, a)$ representing the maximum discounted future reward when we perform action <u>a</u> in state <u>s</u>:

$$Q(s_t, a_t) = \max R_{t+1}$$

  - **Q-function**: represents the "**Quality**" of a certain action in a given state
- Imagine you have the magical Q-function

$$\pi(s) = arg\max_{a} Q(s, a)$$

  - π is the policy

Q-Learning

- How do we get the Q-function?
  - Bellman Equation （贝尔曼公式）

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

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

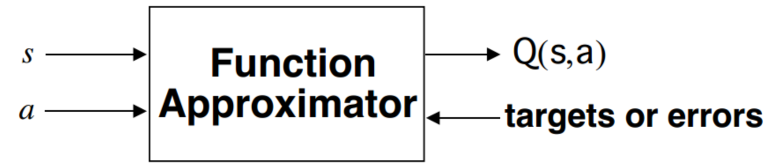
Value Iteration

Q-Learning

- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states
- Think about the **Breakout** game
  - State: screen pixels
    - Image size: $\mathbf{84 \times 84}$ (resized)
    - Consecutive **4** images
    - Grayscale with **256** gray levels

$\mathbf{256^{84\times84\times4}}$ rows in the Q-table!

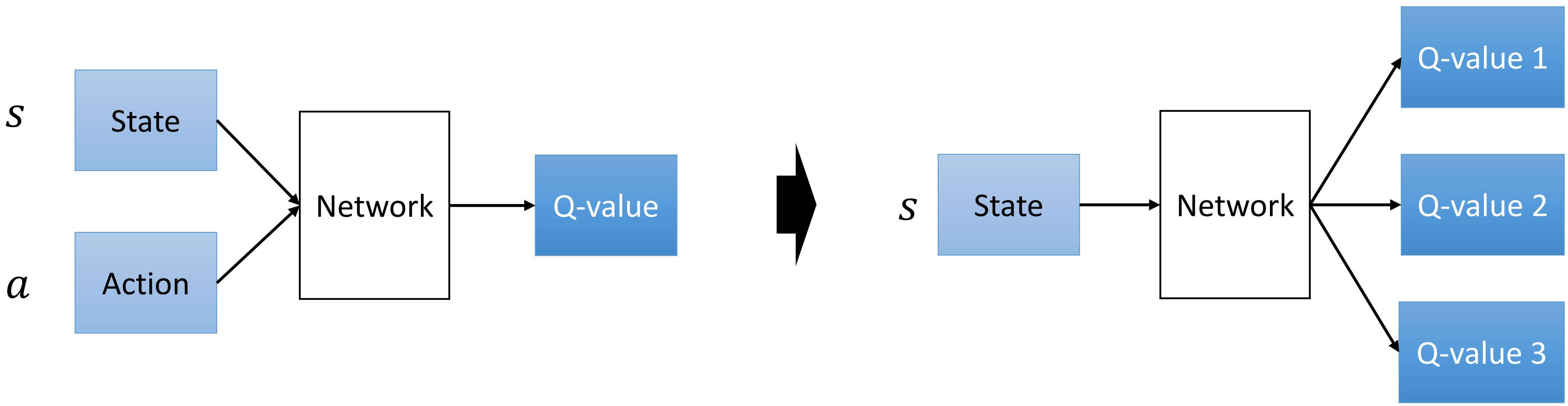
Function Approximator



- Use a function (with parameters) to approximate the Q-function

$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$

  - Linear
  - Non-linear: **Q-network**



Deep Q-Learning

- Stability issues with Deep RL
  - Naïve Q-learning oscillates or diverges with neural nets
    1. Data is sequential
       - Successive samples are correlated, non-i.i.d.
    2. Policy changes rapidly with slight changes to Q-values
       - Policy may oscillate
       - Distribution of data can swing from one extreme to another
    3. Scale of rewards and Q-values is unknown
       - Naive Q-learning gradients can be large unstable when backpropagated

Deep Q-Learning

- Deep Q-Network provides a stable solution to deep value-based RL

  1. Use experience replay
     - Break correlations in data, bring us back to i.i.d. setting
     - Learn from all past policies
     - Using off-policy Q-learning
  2. Freeze target Q-network
     - Avoid oscillations
     - Break correlations between Q-network and target
  3. Clip rewards or normalize network adaptively to sensible range
     - Robust gradients

Stable Deep RL(1) : Experience Replay

- To remove correlations, build data-set from agent's own experience
  - Take action a_t according to ε-greedy policy

    (Choose "best" action with probability 1- ε, and selects a random action with probability ε)
  - Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$ (Huge data base to store historical samples)
  - Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
  - Optimize MSE between Q-network and Q-learning targets, e.g.

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i)}_{\text{target}} - Q(s, a; \theta_i)\right)^2\right]$$

Stable Deep RL(2) : Fixed Target Q-Network

- To avoid oscillations, fix parameters used in Q-learning target
  - Compute Q-learning targets w.r.t. old, fixed parameters θ_i^-

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

  - Optimize MSE between Q-network and Q-learning targets

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

  - Periodically update fixed parameters $\theta_i^- \leftarrow \theta_i$

Stable Deep RL(3) : Reward / Value Range

- DQN clips the reward to [-1, +1]
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned

Stable Deep RL

DQN

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

How to Train the Deep Q-Network

- Loss function :

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

- Differentiating the loss function w.r.t. the weights we arrive at following gradient :

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

Do gradient descent:

$$\theta_{i+1} = \theta_i + \alpha \cdot \nabla_{\theta_i}L_i(\theta_i)$$

How to Train the Deep Q-Network

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

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

    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

How to Train the Deep Q-Network

During Training

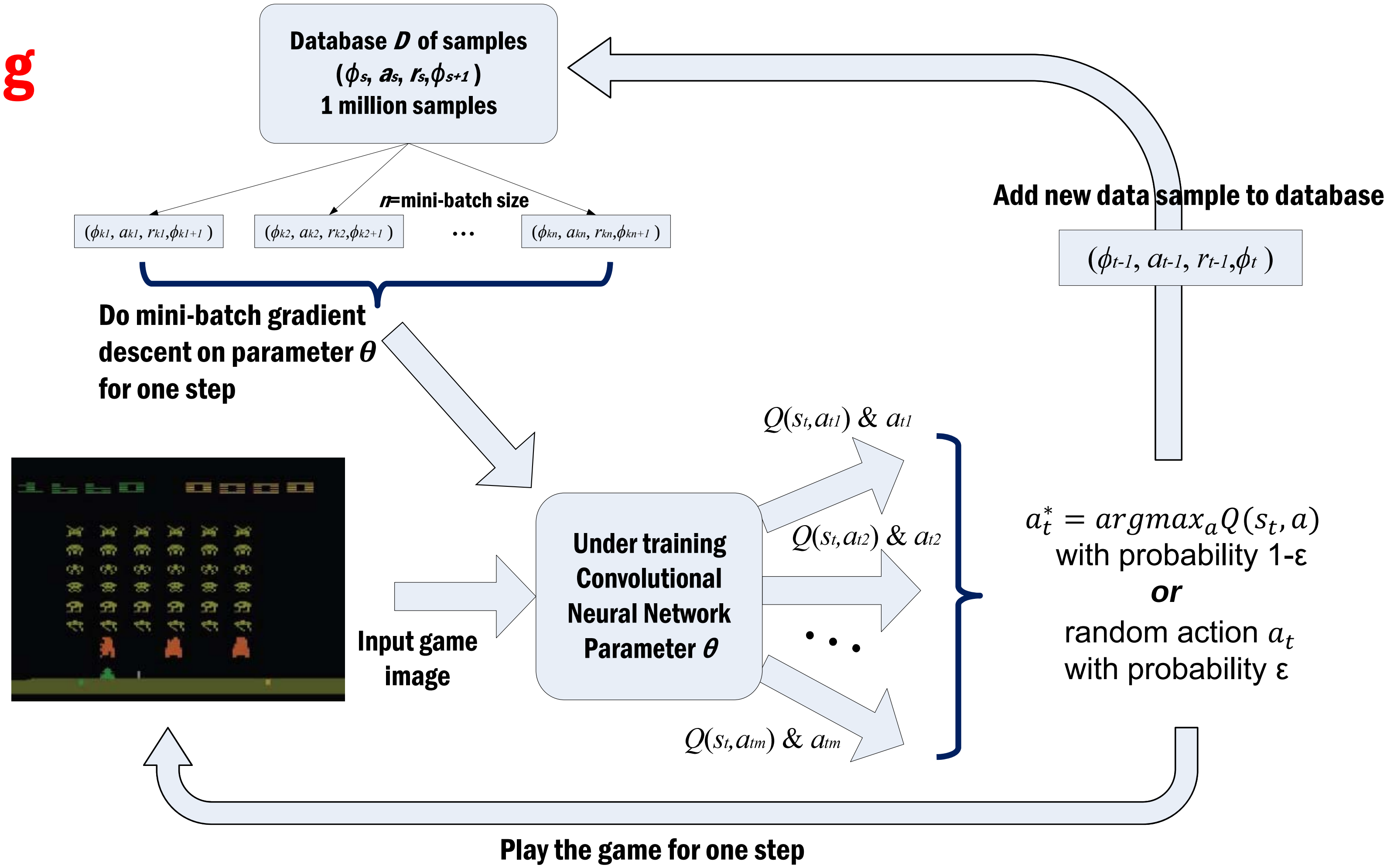


image at time t : x_t

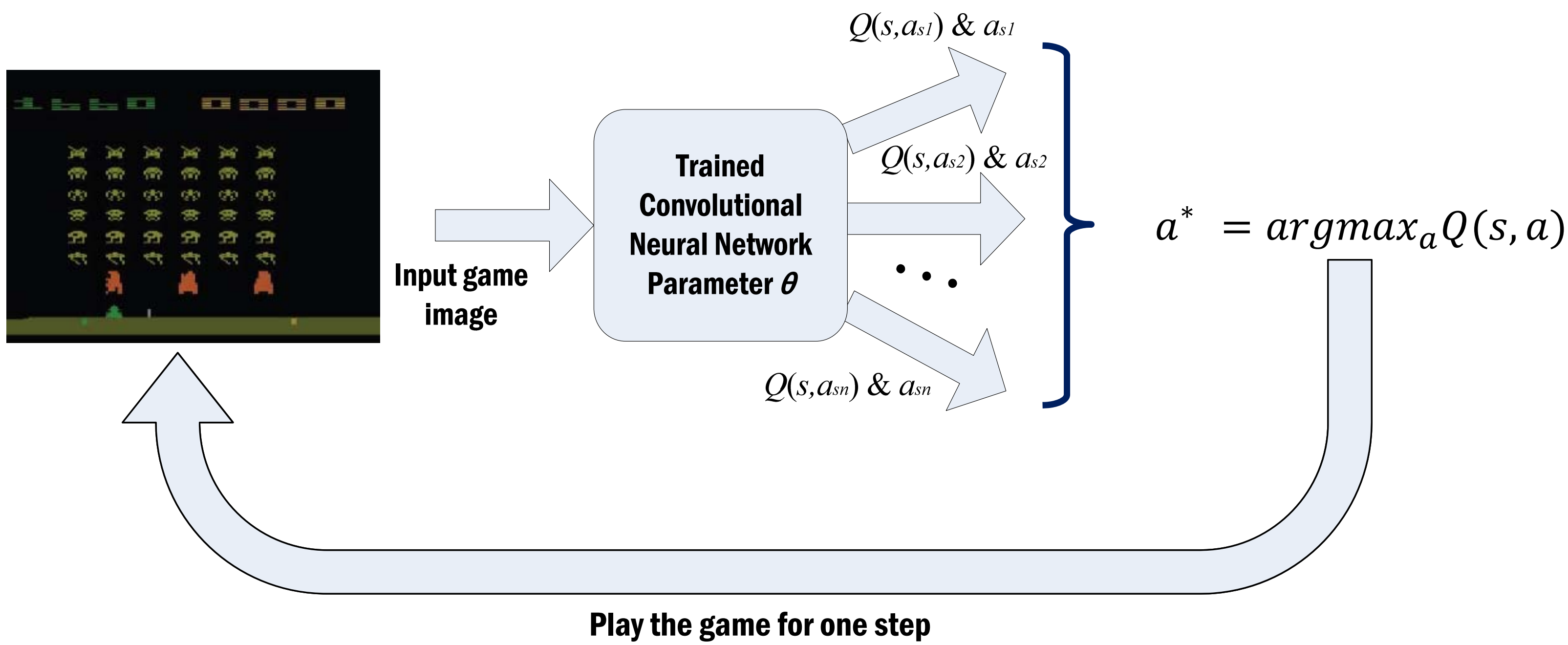
$s_t = s_{t-1}, a_{t-1}, x_t$

preprocessed sequence

$\phi_t = \phi(s_t)$

How to Train the Deep Q-Network

After Training



Vehicle Intelligence Laboratory

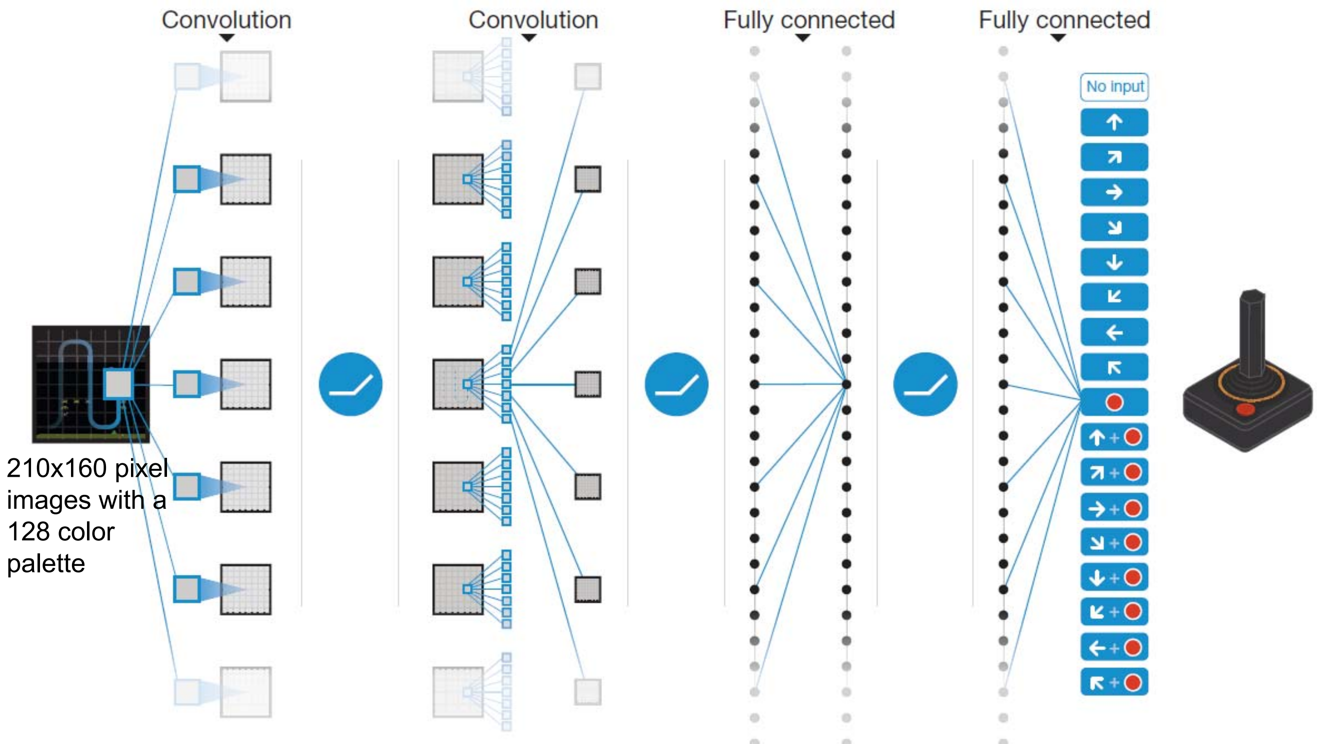
DQN in Atari

Extended Data Table 1 | List of hyperparameters and their values

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter *C* from Algorithm 1).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.
no-op max	30	Maximum number of "do nothing" actions to be performed by the agent at the start of an episode.

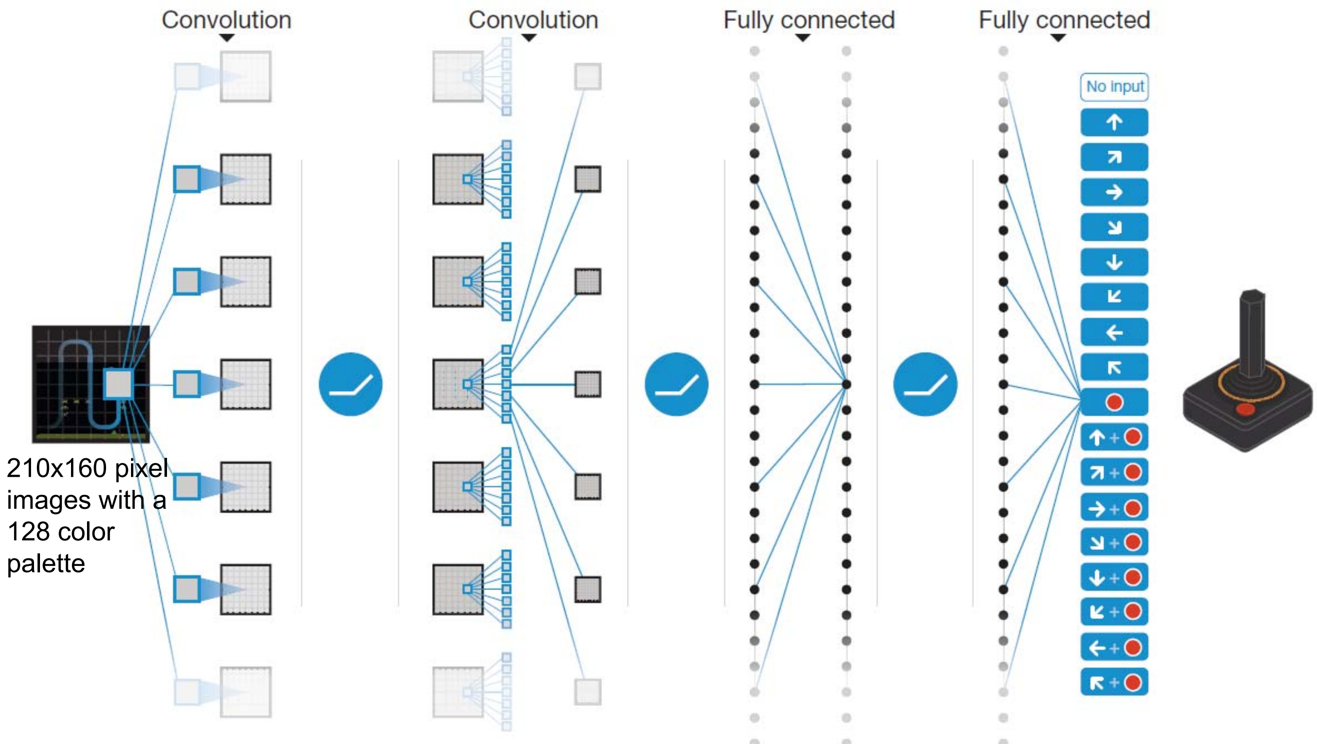
The values of all the hyperparameters were selected by performing an informal search on the games Pong, Breakout, Seaquest, Space Invaders and Beam Rider. We did not perform a systematic grid search owing to the high computational cost, although it is conceivable that even better results could be obtained by systematically tuning the hyperparameter values.

DQN in Atari



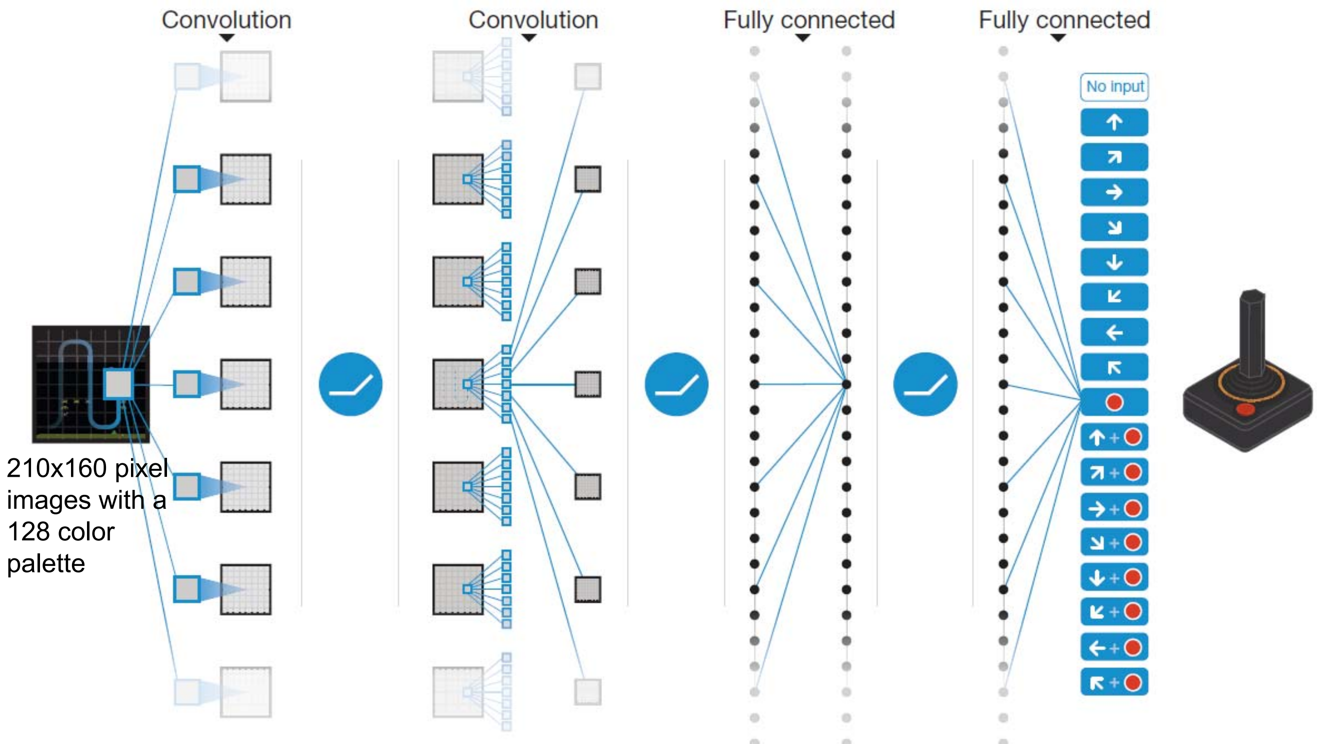
- The input to the neural network consists of an 84x84x4 image produced by the pre-processing map ϕ
- Input state is stack of raw pixels from last 4 frames

DQN in Atari



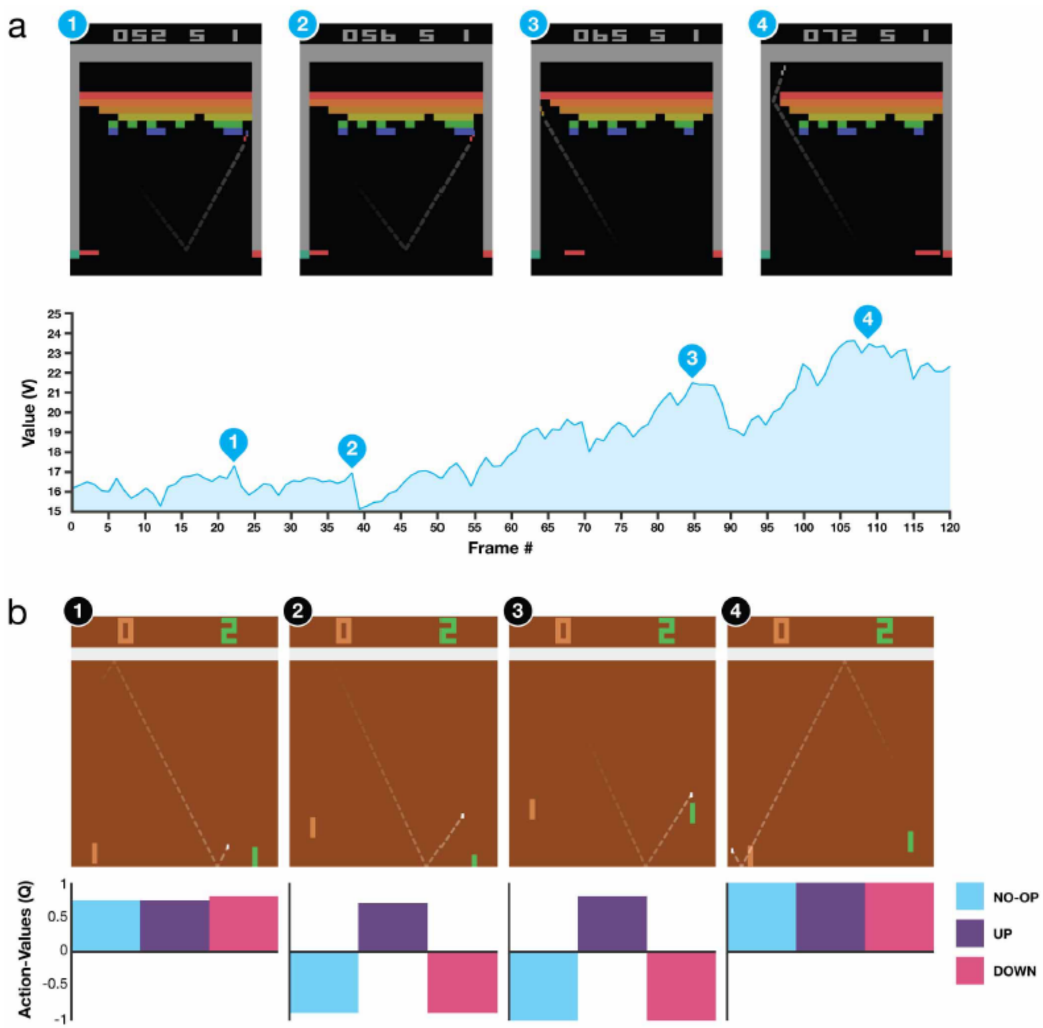
- The first hidden layer convolves 32 filters of 8x8 with stride 4 with the input image and applies a rectifier nonlinearity.
- The second hidden layer convolves 64 filters of 4x4 with stride 2.
- This is followed by a third convolutional layer that convolves 64 filters of 3x3 with stride 1

DQN in Atari

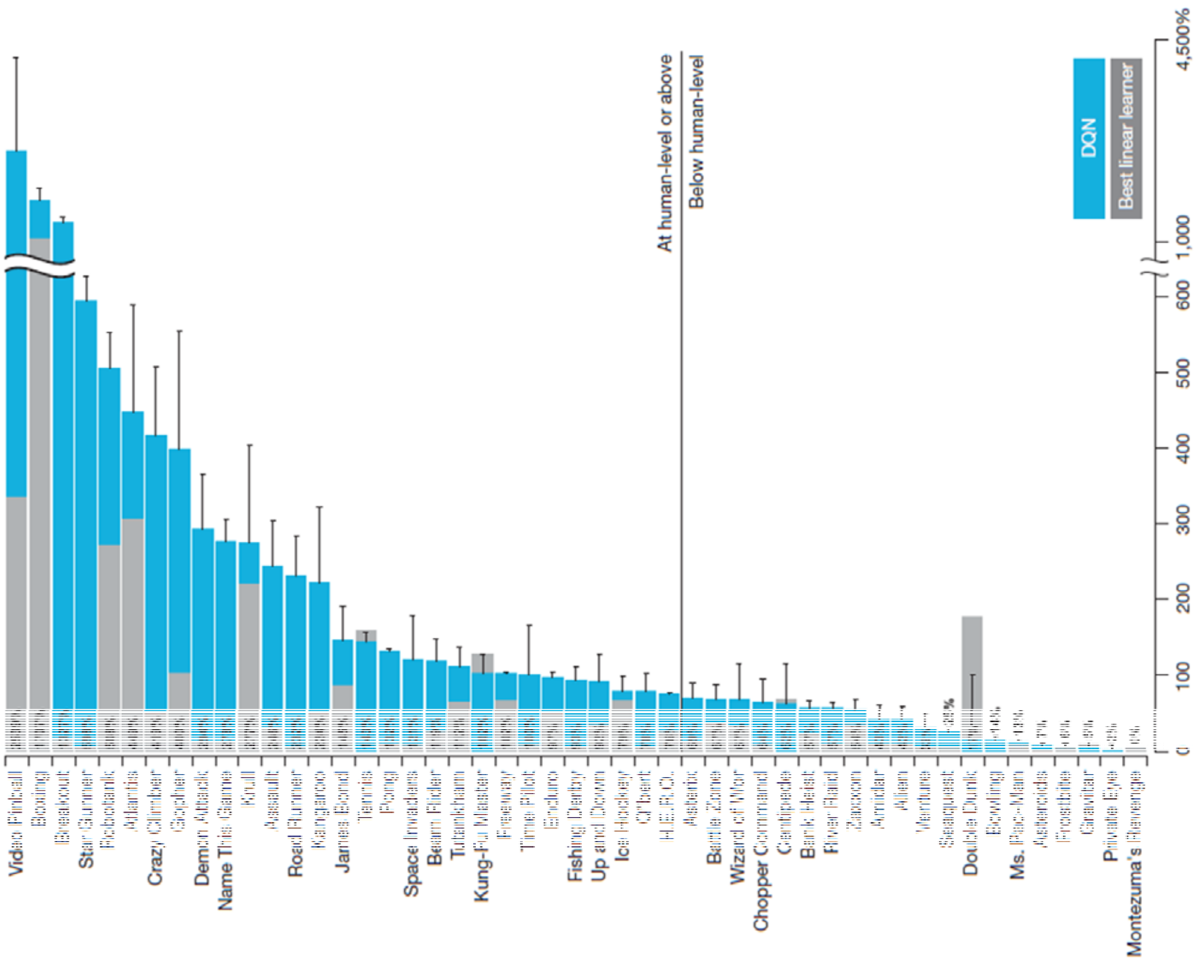


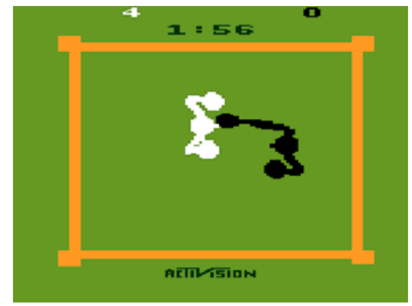
- The final hidden layer is fully-connected and consists of 512 rectifier units.
- The output layer is a fully-connected linear layer with a single output of each valid action.
- The number of valid actions varied between 4 and 18 on the games

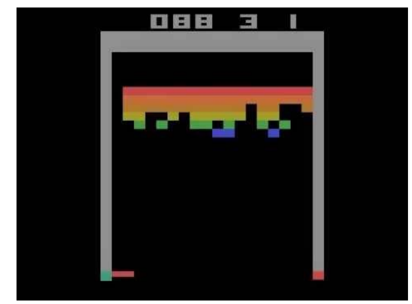
DQN result in Atari



DQN result in Atari

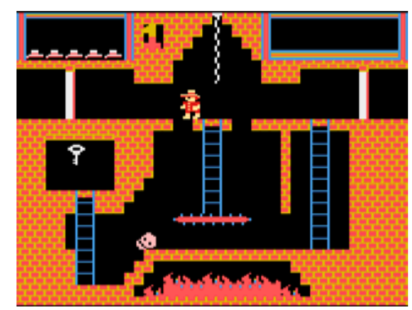






Good results

Bad results



Vehicle Intelligence Laboratory

DQN result in Atari

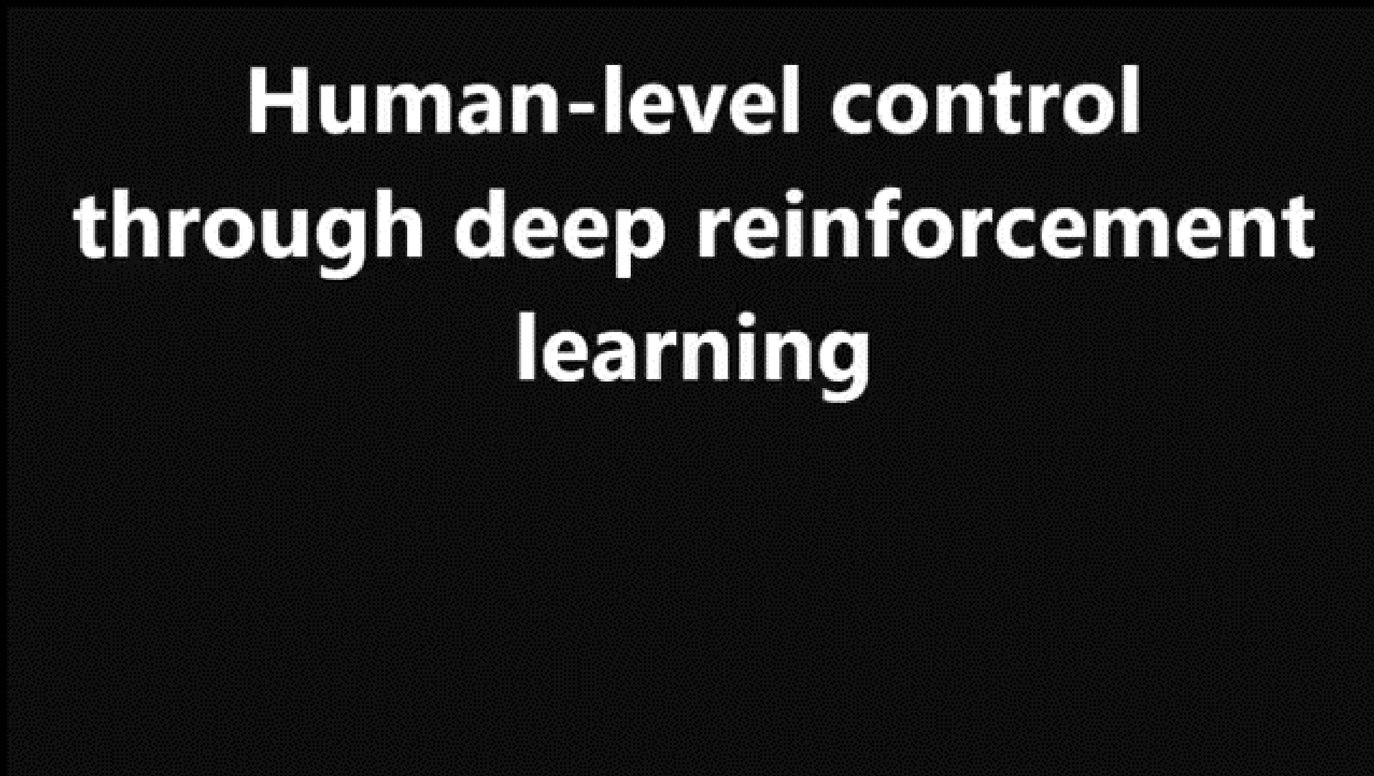
"Seaquest" DQN gameplay

Before training

peaceful swimming

[https://youtu.be/5WXVJ1A0k6Q]

DQN result in Atari



[https://youtu.be/iqXKQf2BOSE]

Conclusion

- Reinforcement learning provides a general-purpose framework for A.I.
- RL problems can be solved by end-to-end deep learning
- A single agent can now solve many challenging tasks
- Reinforcement learning + Deep learning
- Agent can do stuff that maybe human don’t know how to program