Human-level Control Through Deep Reinforcement Learning

Dong-Kyoung Kye

2015. 11. 17

Vehicle Intelligence Laboratory
Seoul Nat'l University

Vehicle Intelligence Laboratory

Paper information

- **Title**
  - Human-level control through deep reinforcement learning

- **Publication**
  - *Nature,* Vol. 518, Feb 2015 (citation : 97)

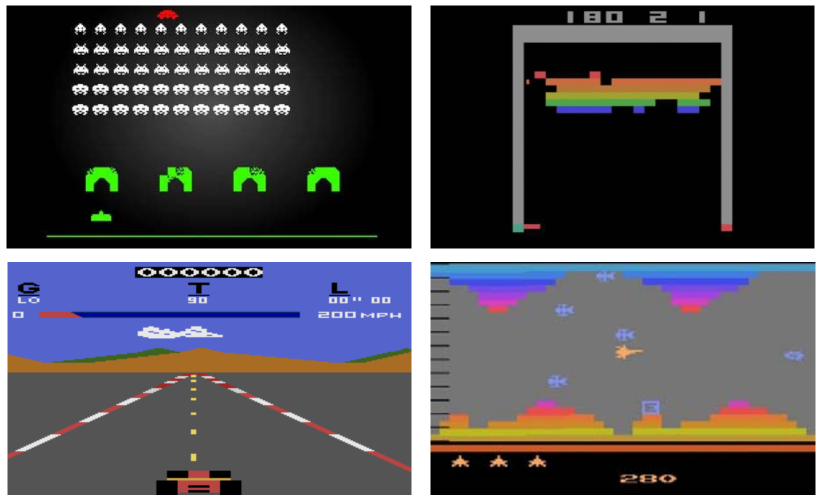
- **Authors**
  - Volodymyr Mnih et al. (Google DeepMind)

Motivation

- Automatically convert unstructured information into useful, actionable knowledge.
- Ability to learn for itself from experience.
- Robot can do stuff that maybe human don’t know how to program
- Model-free reinforcement learning
- Deep learning + Reinforcement learning!

What did they do?

- Playing Atari with deep reinforcement learning.
- Trained by deep learning “Convolutional Neural Network (CNN)”
- Input is current state (raw image sequence)
- Output is all legal action (joystick) and corresponding Q(s,a) value

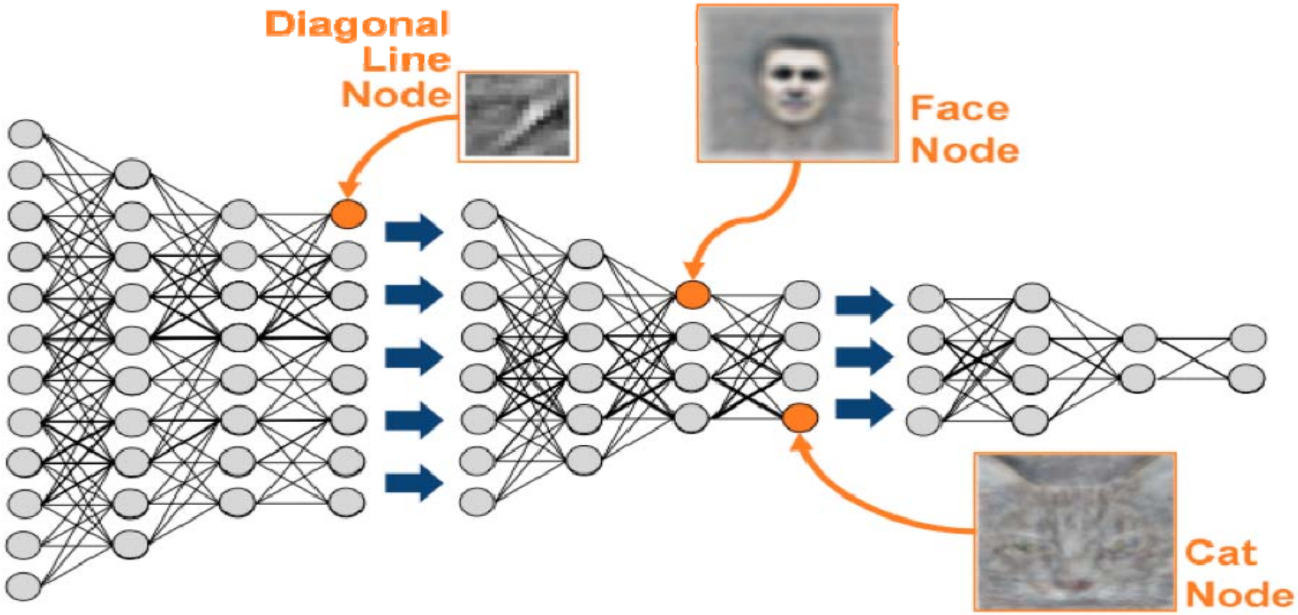


What is special?

- Input is only raw image
- Output is the action
- Game independent, same Convolutional Neural Network (CNN) for all games
- Outperform human expert players in some games

What is deep learning?

- ML algorithm that use many layers of nonlinear processing units for feature extraction and transformation.
- It is based on learning multiple levels of features or representation in each layer



What is deep learning?

TRADITIONAL APPROACH

The traditional approach uses fixed feature extractors.

→ Hand-crafted Feature Extractor → "Simple" Trainable Classifier →

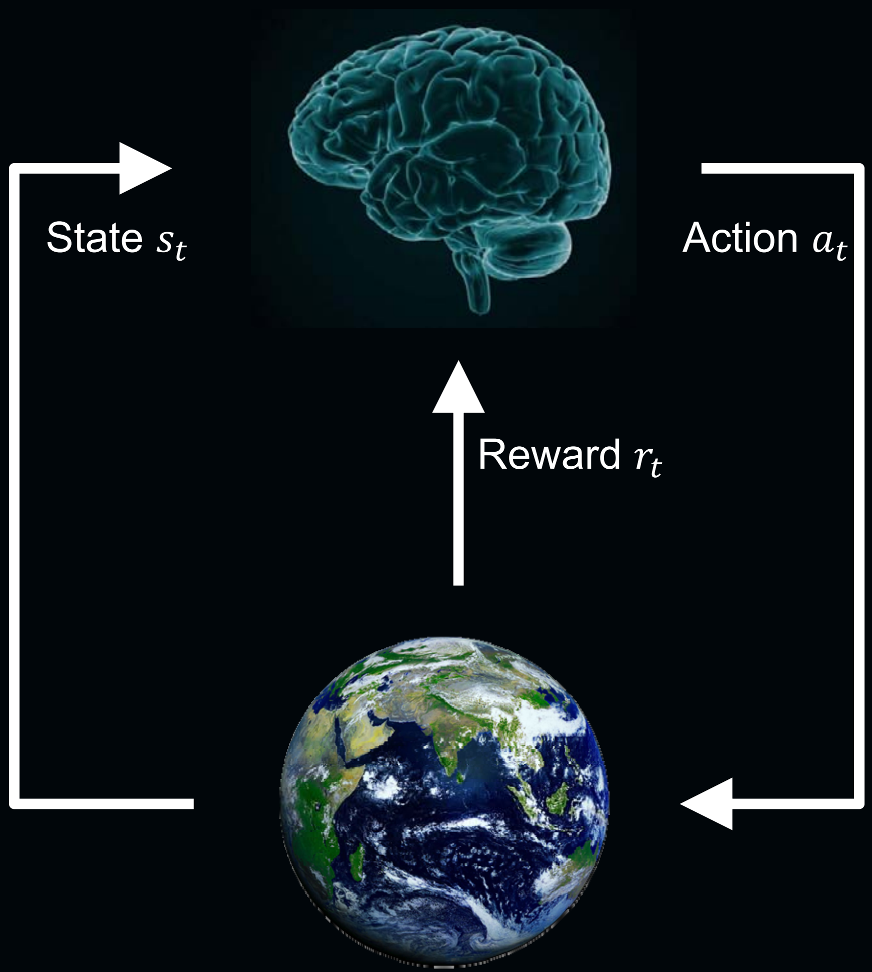
DEEP LEARNING APPROACH

Deep Learning approach uses trainable feature extractors.

→ Trainable Feature Extractor → Trainable Classifier →

“However, many deep learning algorithms mainly deal with the perception problem…”

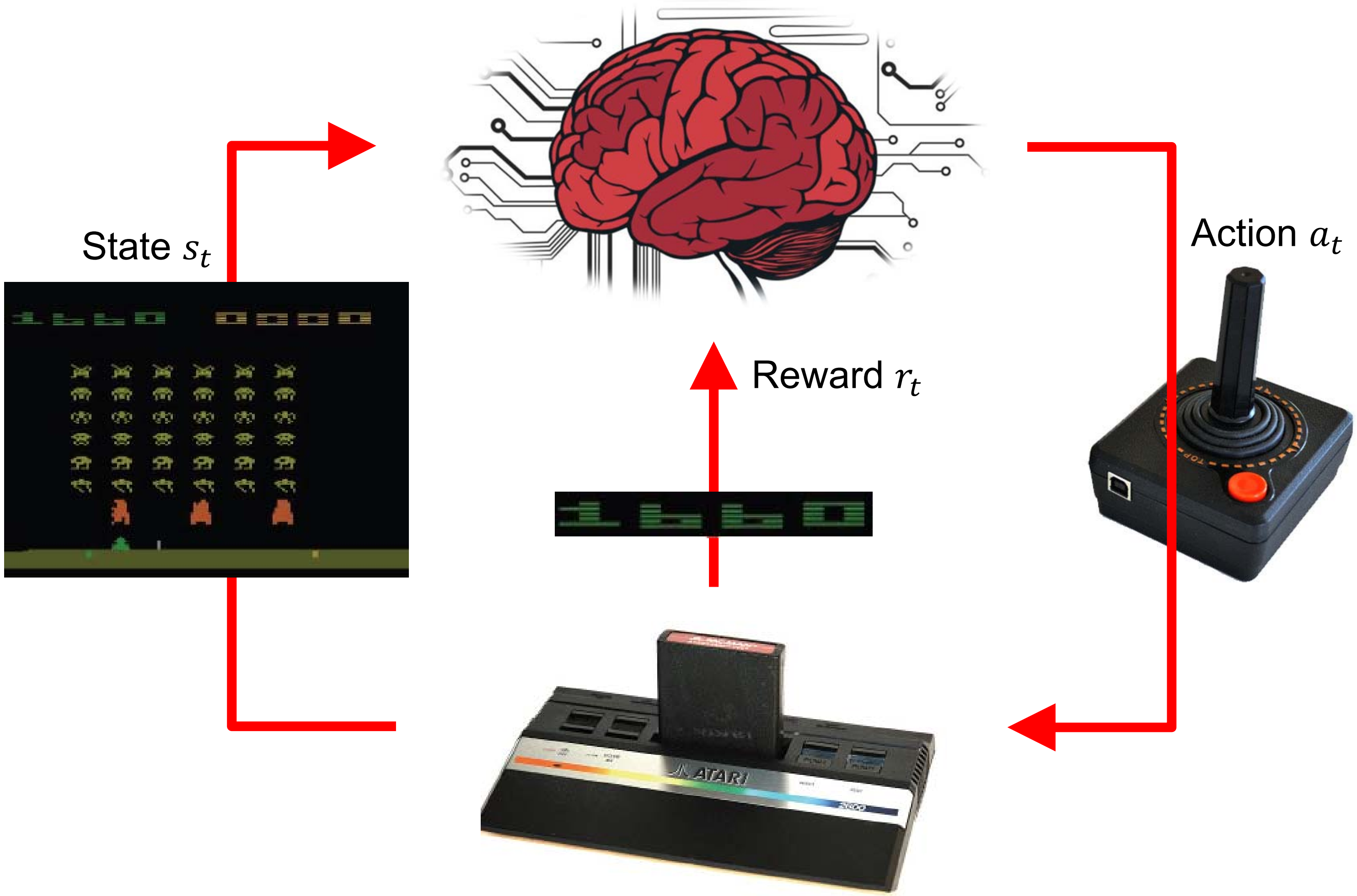
Reinforcement Learning



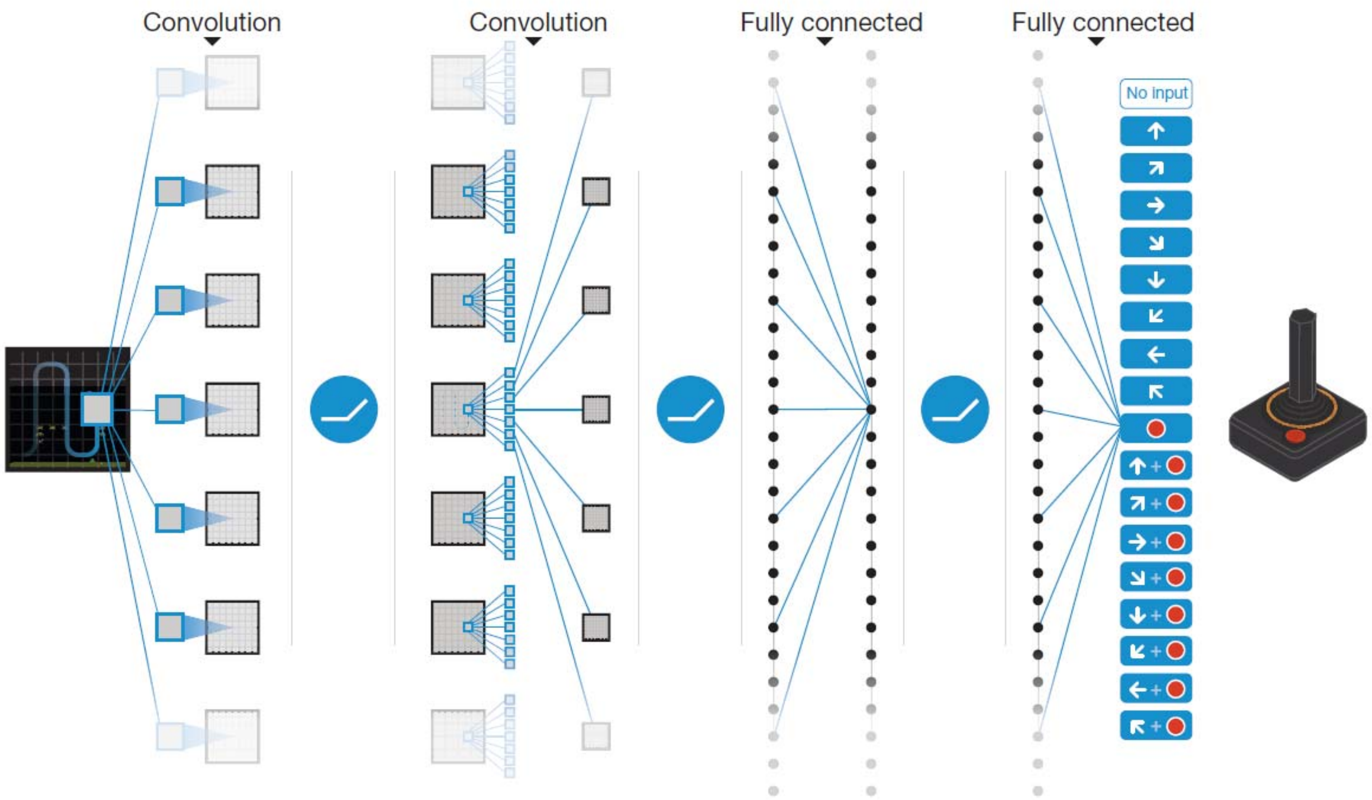
- At each step *t the agent :*
  - Receives state s_t
  - Receives scalar reward r_t
  - Executes action a_t

- The environment *:*
  - Emits state s_t
  - Emits scalar reward r_t
  - Receives action a_t

Problem Definition



Problem Definition



Action a

Value $Q(s,a)$

Bellman Equation

- The optimal policy is given by :

$$\pi_s^* = argmax_\pi U^\pi(s)$$

- Denote $U^{\pi^*}(s)$ as $U(s)$, the optimal policy chooses the action that maximizes the expected utility of the subsequent state :

$$\pi^*(s) = argmax_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

Bellman Equation

- Bellman Equation : $U(s) = R(s) + \gamma \cdot max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$
- The utility of a state is the immediate reward for that state plus expected discounted utility of the next state, assuming that the agent choose the optimal action
- $U^{\pi^*}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)]$ with $S_0 = s$, is the unique solution to Bellman equation.

  → (The expected value of state s is obtained by executing π starting in s)

Q-learning

Bellman Equation:

$$U(s) = R(s) + \gamma \cdot max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

- Q-value is defined by :

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) max_{a'} Q(s',a')$$

- The relationship between utility and Q-value is :

$$U(s) = max_a Q(s,a)$$

- The optimal policy is given by :

$$\pi^*(s) = argmax_a Q(s,a)$$

- Q-learning algorithm is used to learn this Q-value table

Q-learning

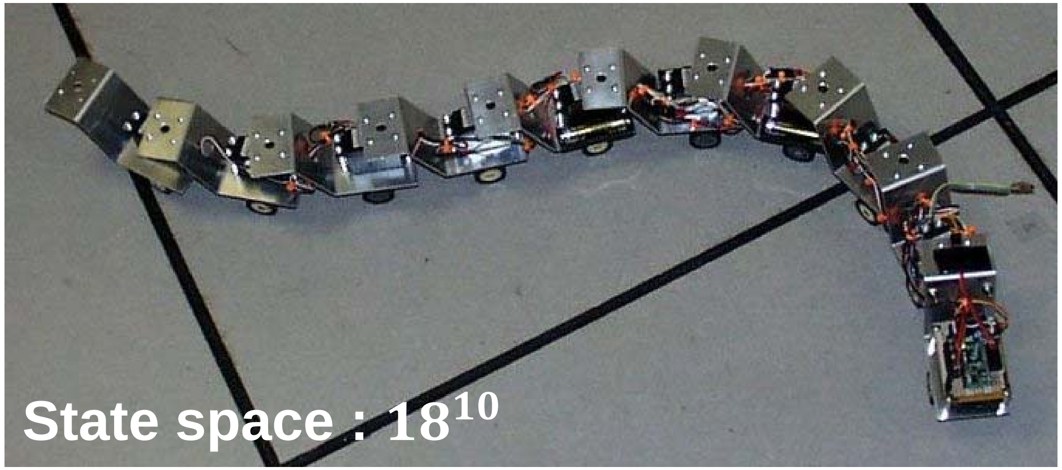
- Q : a table of Q-values indexed by state and action.

$$\boldsymbol{Q} : \boldsymbol{S} \times \boldsymbol{A} \rightarrow \mathbb{R}$$

- Before learning has started, Q returns an (arbitrary) fixed value.
- Then each time the agent selects an action, and observes a reward and a new state that may depend on both previous state and the selected action, "Q" is updated

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\substack{\text{discount} \\ \text{factor}}} \cdot \underbrace{\max_a Q_t(s_{t+1}, a)}_{\substack{\text{estimate of optimal} \\ \text{future value}}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \right)$$

Q-learning



In this case…

- ✓ It's hard to define transition probability $P(s'|s,a)$

$$U(s) = R(s) + \gamma \cdot max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$

$$\boxed{Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t) \cdot \left(R_{t+1} + \gamma \cdot \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)}$$

→ No transition probability term

Q-learning

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t) \cdot \left(\underbrace{R_{t+1} + \gamma \cdot \max_a Q_t(s_{t+1}, a)}_{\hat{Q}_t(s_t, a_t)} - Q_t(s_t, a_t) \right)$$

$$\Downarrow$$

$$\underbrace{Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t) \cdot \underbrace{\left(\hat{Q}_t(s_t, a_t) - Q_t(s_t, a_t) \right)}_{\text{Derivative the square error } (\hat{Q} - Q)^2}}_{\text{Regress } Q \text{ to } \hat{Q} \text{ using stochastic gradient descent method}}$$

Q-learning

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t) \cdot \left(\hat{Q}_t(s_t, a_t) - Q_t(s_t, a_t)\right)$$

Too many pixels (states)..

Deep Q-Learning

- **What role does Deep Learning play in RL?**
  - Provides a compact form for Q (function approximator)

$$\theta_{t+1} = \theta_t + \gamma \underbrace{\left(\hat{Q}(x_t, a_t) - Q_{\theta_t}(x_t, a_t)\right)\frac{\partial Q_\theta}{\partial \theta}}_{\text{derivative of the square error } \frac{\partial(\hat{Q}-Q)^2}{\partial \theta}}$$

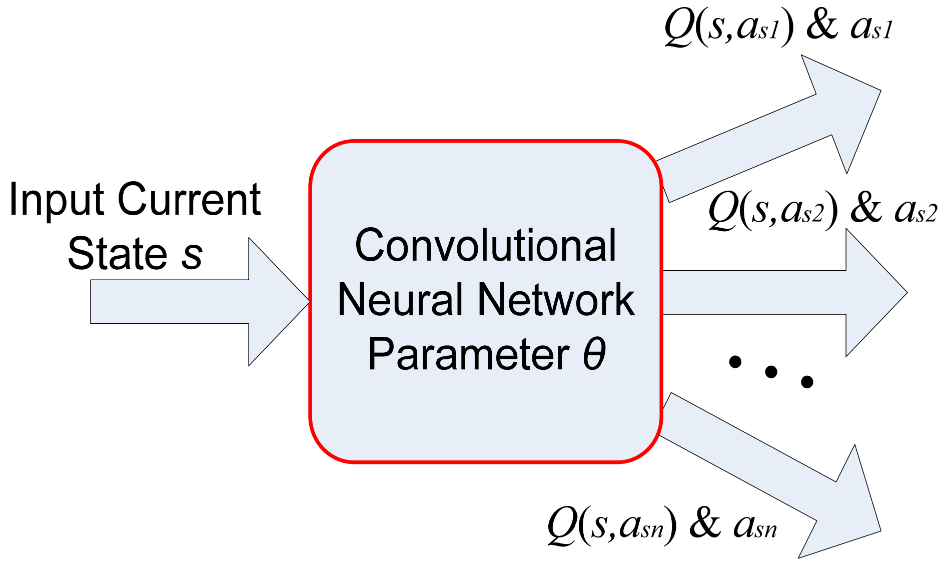
  - Parameterizing an approximate value function $Q(s, a; \theta_i)$ using deep convolutional neural network, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i.

Vehicle Intelligence Laboratory

Deep Q-Learning

- **Approach the Q-value with a convolutional neural network $\boldsymbol{Q(s,a;\theta_i)}$**

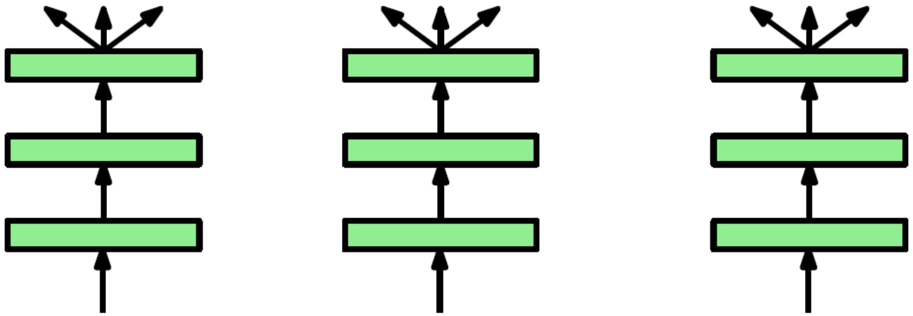
$$Q(s,a;\theta) \approx Q^*(s,a)$$



Neural network function approximator with weight θ is a 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.



$\mathbf{x}_t = 2 \qquad \mathbf{x}_{t+1} = 2 \qquad \mathbf{x}_{t+2} = 2$

Correlated samples break learning

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

    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

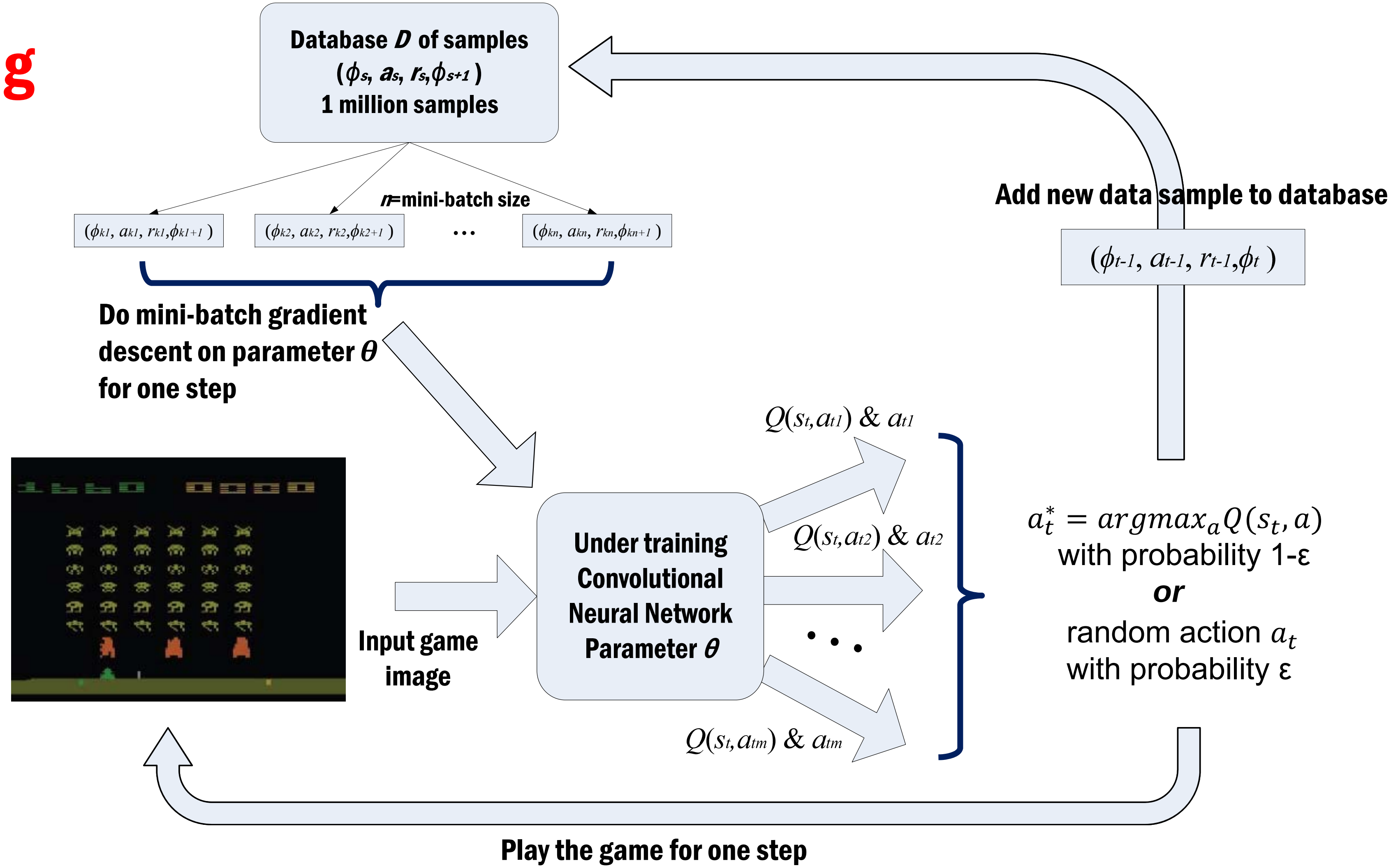


image at time t : x_t

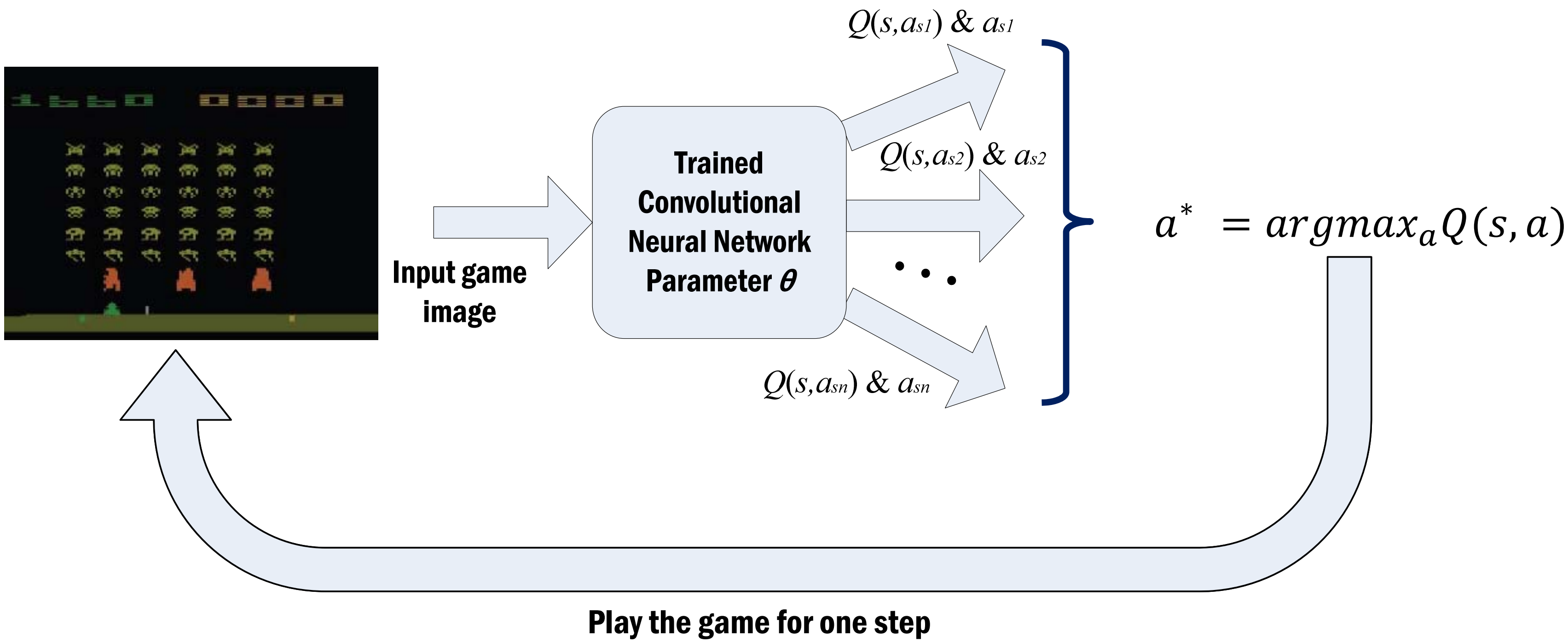
$s_t = s_{t-1}, a_{t-1}, x_t$

preprocessed sequence

$\phi_t = \phi(s_t)$

How to Train the Deep Q-Network

After Training



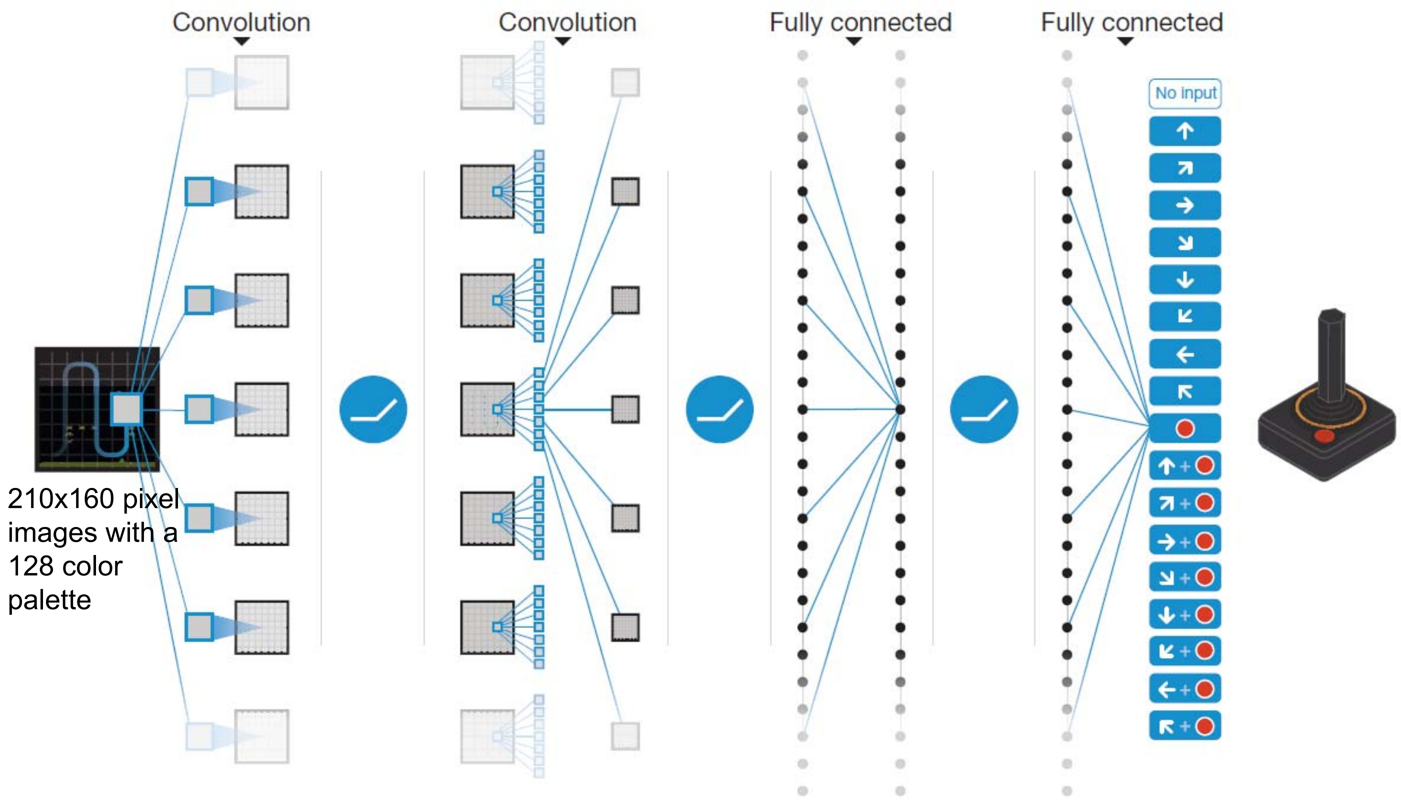
DQN in Atari

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.

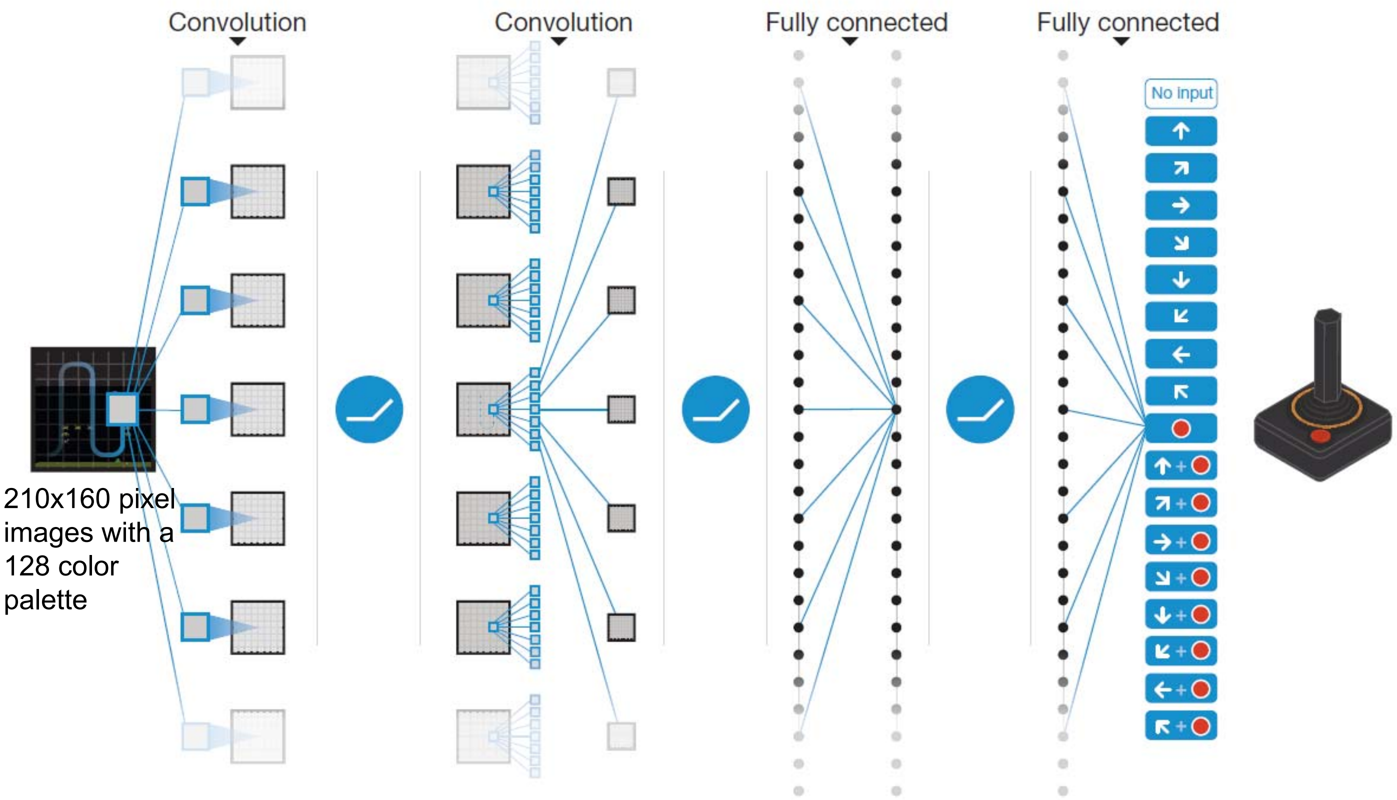
The values of all the hyperparameters were selected by performing an informal search on the games Pong, Breakout, Seaquest, Space Invaders and Beam Rider. We did not perform a systematic grid search owing to the high computational cost, although it is conceivable that even better results could be obtained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

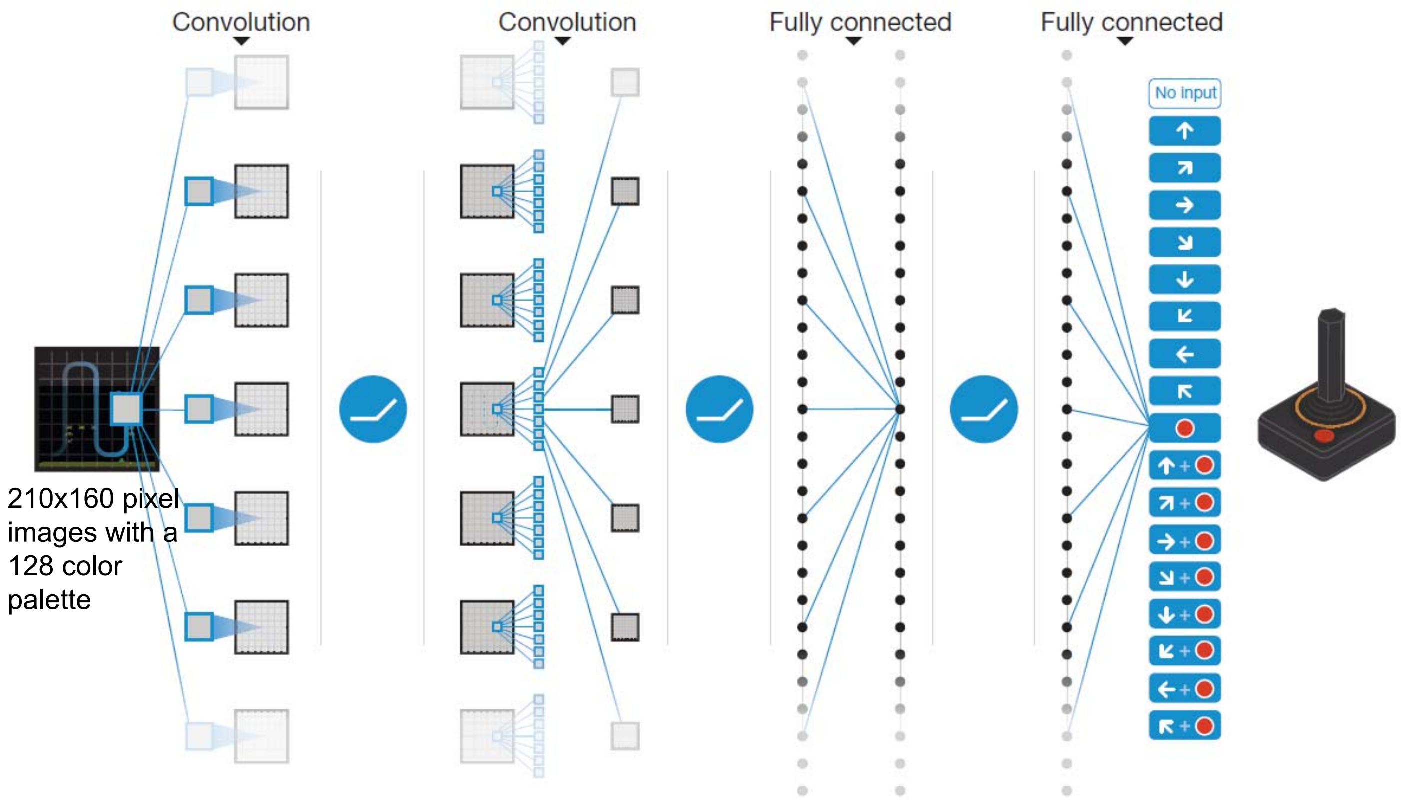


210x160 pixel images with a 128 color palette

- 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

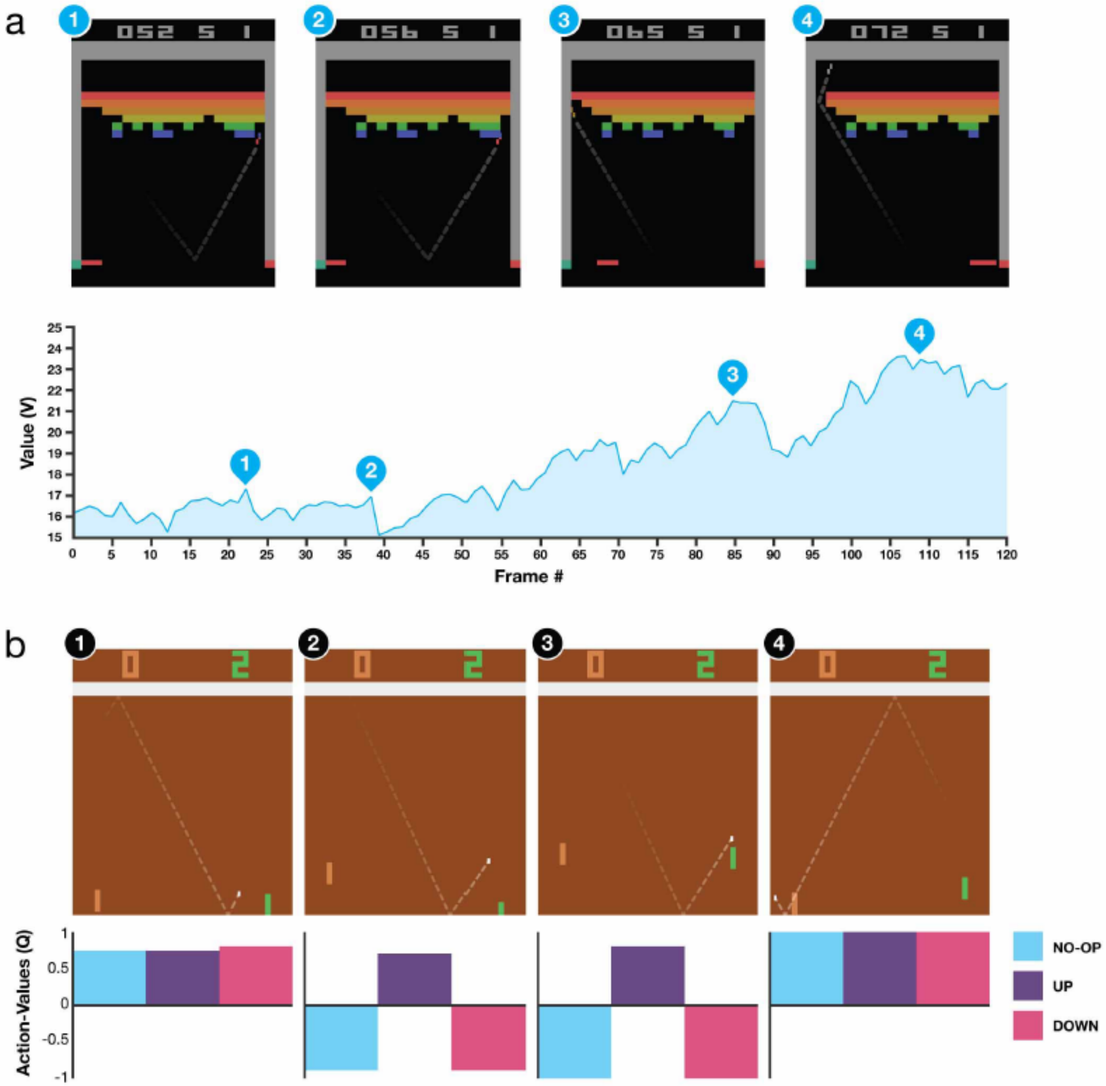
Vehicle Intelligence Laboratory

DQN in Atari

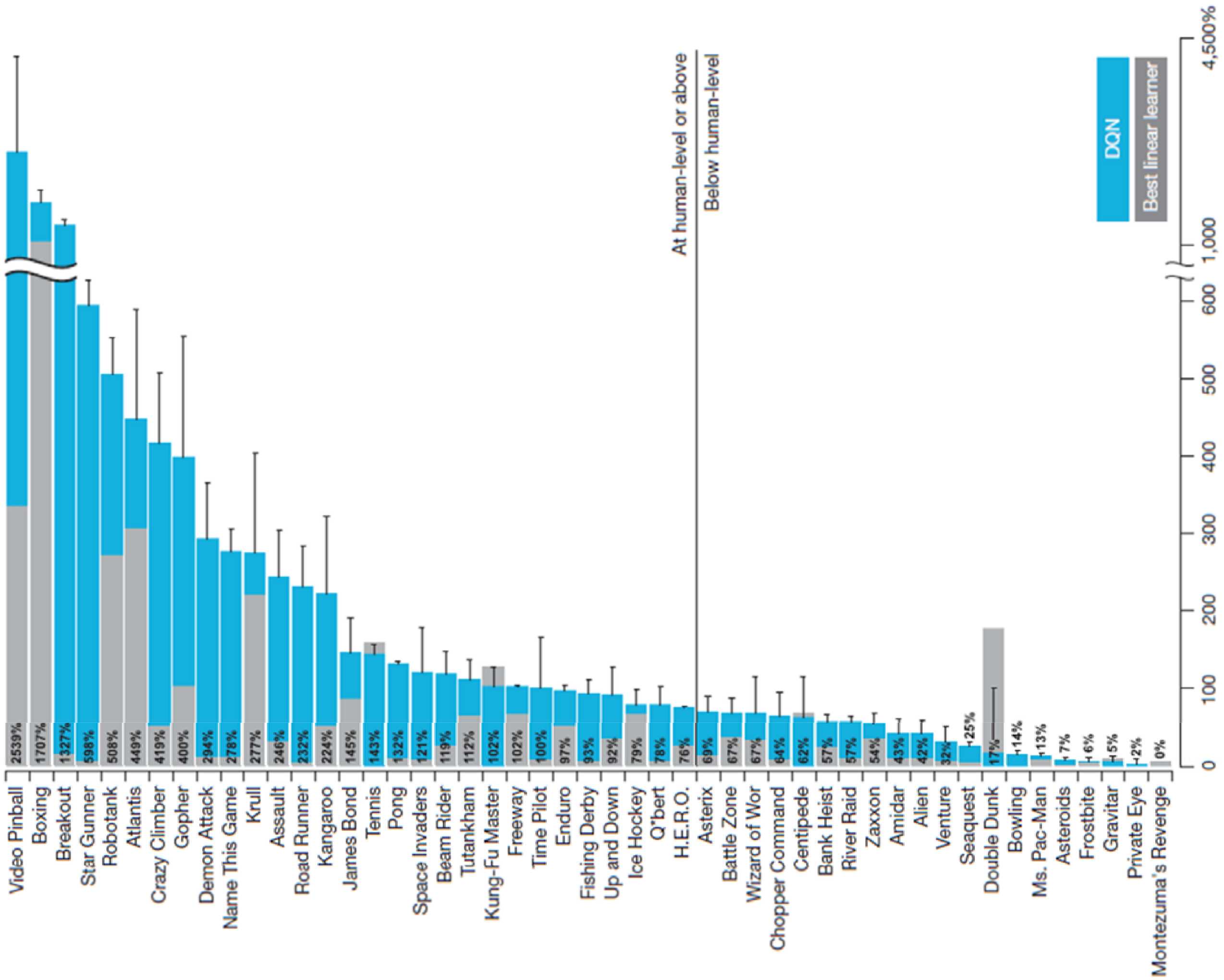


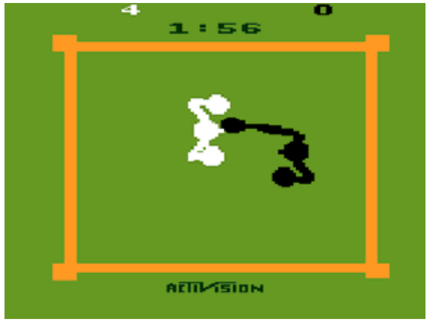
- The final hidden layer is fully-connected and consists of 512 rectifier units.
- The output layer is a fully-connected linear layer with a single output of each valid action.
- The number of valid actions varied between 4 and 18 on the games

DQN result in Atari



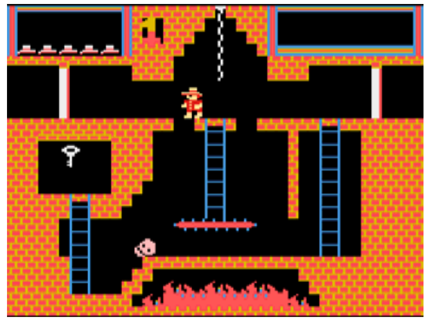
DQN result in Atari





Good results

Bad results

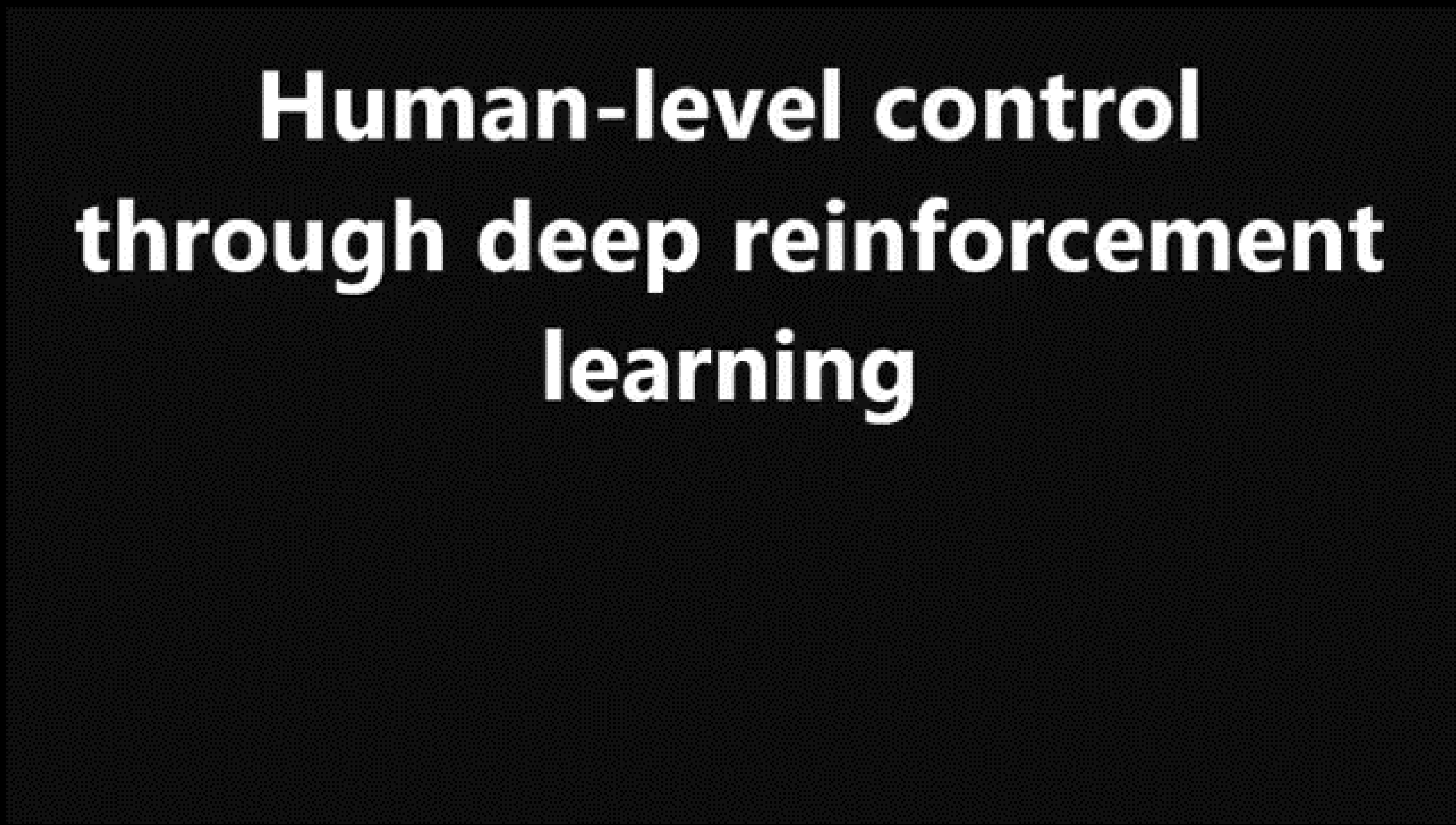


DQN result in Atari

“Seaquest” DQN gameplay

[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