Playing Atari with Deep RL

Jaloliddin Boymakhammadov

April 15, 2025

Bocconi
AI&Neuroscience
Student
Association

Outline

Contributions of this Paper

- RL agents have achieved success in various domains, but have mostly been limited to fully observed, low-dimensional state spaces.
- This paper develops a novel artificial agent (deep Q-network) capable of learning successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning.
- Deep Q-network received only pixel inputs from 49 Atari 2600 games, using the same architecture and hyperparameters:
  - Surpassed performance of all previous algorithms.
  - Achieved performance comparable to professional human testers.
- First artificial agent capable of excelling at a diverse set of challenging tasks.

Model Overview

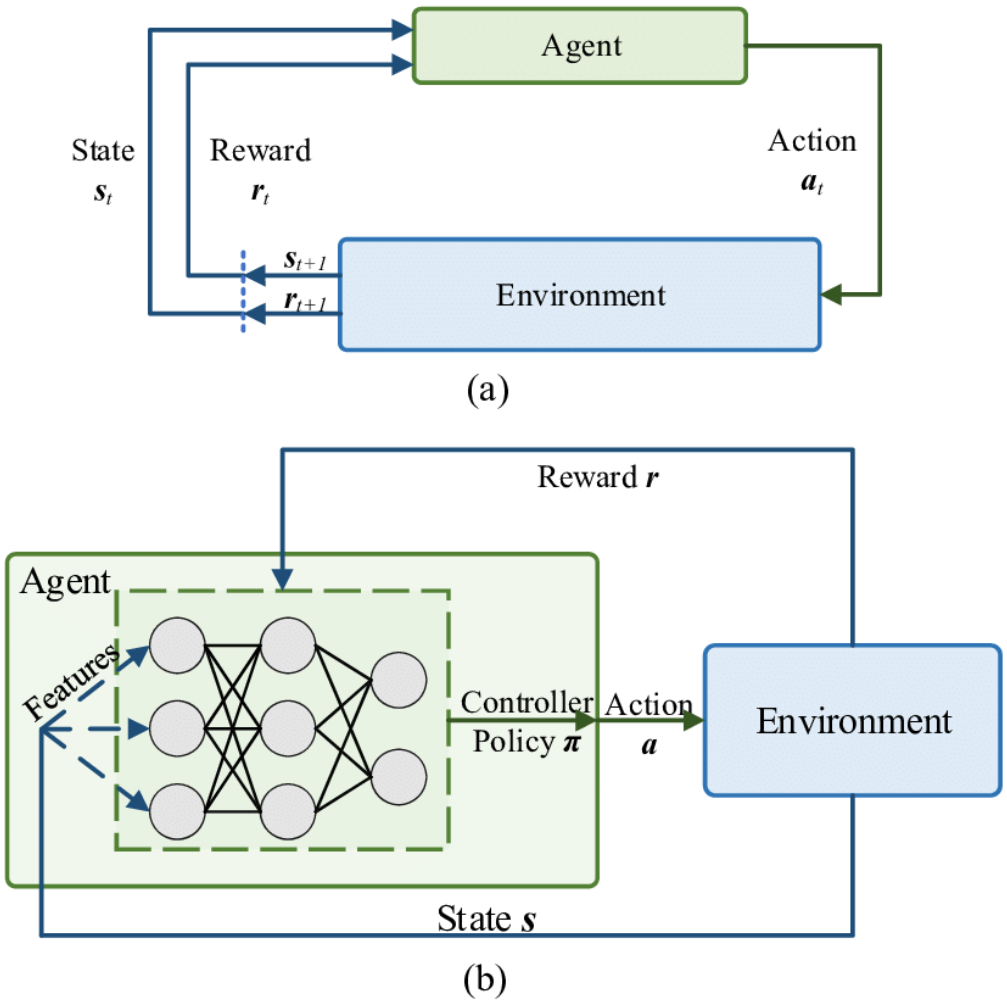


Figure: (a) Reinforcement learning architecture. (b) Deep reinforcement learning architecture with deep neural networks replacing the traditional Q-table.

Model details: General overview

- Agent aims to maximize future discounted cumulative reward by approximating the optimal action - value function

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi]$$

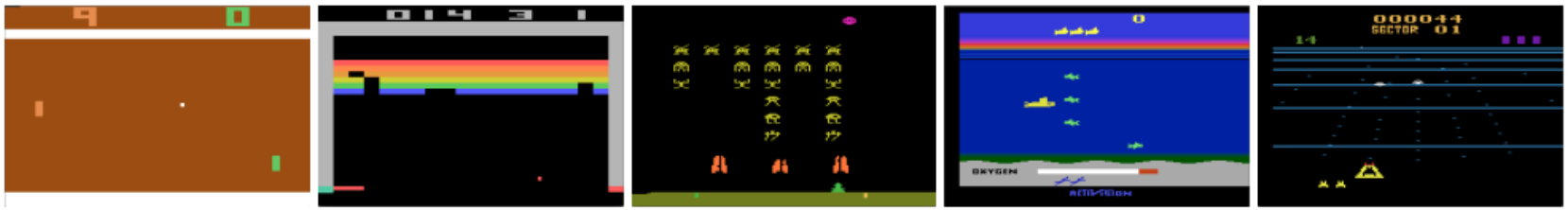


Figure: Screen shots from five Atari 2600 Games: (Left-to-right) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

Bellman Equation and Reinforcement Learning Intuition

Bellman Equation for Optimal Action - Value Function

The Bellman equation for the optimal action - value function Q^* is:

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

Value Iteration Concept

$$Q_{i+1}(s, a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q_i(s', a')|s, a]$$

By repeatedly applying this update, the Q function converges to the optimal Q^* over time. However, in practice, directly applying this to large or continuous state - action spaces is challenging, which is why function approximators (like neural networks) are employed.

Deep Q - Learning Basics

Q - Learning Update Rule

The core update rule in Q - learning is given by:

$$Q^{\text{new}}(s_k, a_k) = Q^{\text{old}}(s_k, a_k) + \alpha \left(r_k + \gamma \max_a Q(s_{k+1}, a) - Q^{\text{old}}(s_k, a_k) \right)$$

The term $r_k + \gamma \max_a Q(s_{k+1}, a)$ is the TD target estimate.

Parameterization with Neural Network

We approximate the Q function using a neural network:

$$Q(s, a) \approx Q(s, a, \theta)$$

where θ represents the parameters of the neural network. This allows us to handle large or continuous state and action spaces that traditional tabular Q - learning cannot manage effectively.

Problems with non-linear approximation

Reinforcement learning (RL) has distinct challenges when viewed from a deep - learning perspective:

- **Reward Signal Characteristics**: Deep learning models need large amounts of hand-labelled training data. On the other hand, RL must learn from sparse, noisy, and delayed scalar rewards (Credit Assignment Problem).
- **Data Independence Assumption**: Deep learning assumes independent data samples, but RL deals with highly correlated states.
- **Data Distribution**: RL's data distribution changes as the agent learns, conflicting with deep learning's assumption of a fixed data distribution.

Q - Learning with Function Approximation

Loss Function in the Context of Function Approximation

In Q - learning with a function approximator (such as a neural network in Deep Q - Learning), the loss function is:

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

Here, samples (s, a, r, s') are drawn uniformly at random from a replay buffer D. The network has parameters θ_i for the current network, and θ_i^- represents the parameters of a target network.

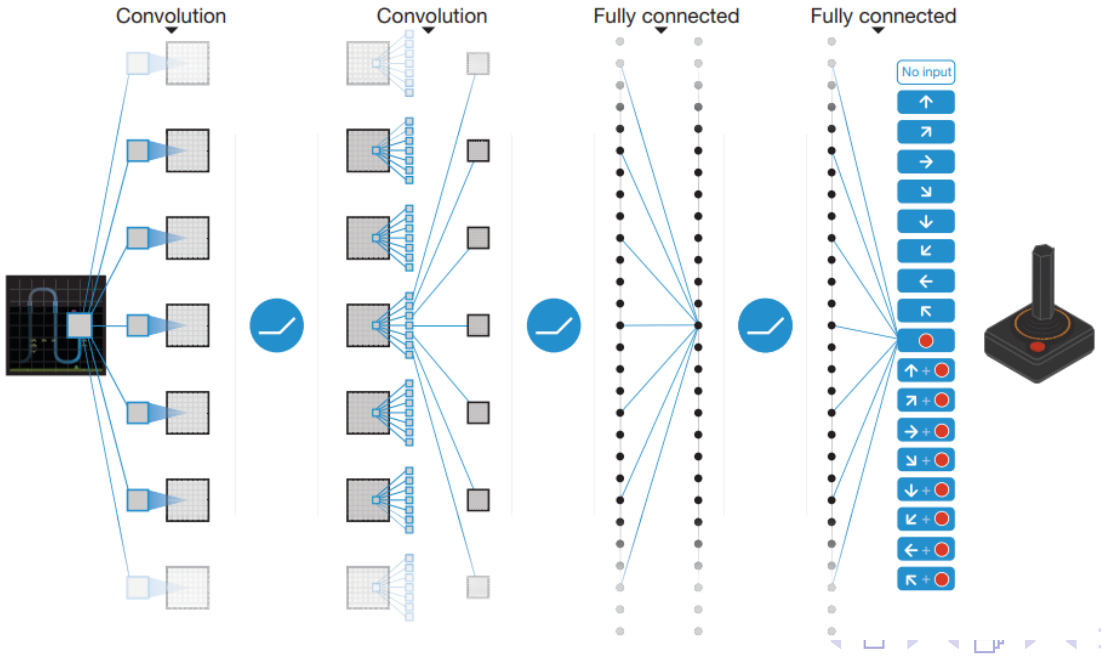
Remark

The target network's parameters are updated less frequently (usually every C steps) to stabilize the training process.

Model Architecture

- Input: $84 \times 84 \times 4$ image frames
- Convolutional layers:
  - 32 filters, 8×8 size, stride 4
  - 64 filters, 4×4 size, stride 2
  - 64 filters, 3×3 size, stride 1
- Fully-connected layers:
  - 512 rectifier (ReLU) units

- Outputs correspond to predicted Q-values for each possible action.
- Given the input state, Q-values for all actions can be computed in a single forward pass.



Deep Q-learning with Experience Replay Algorithm (Part 1)

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 $a_t =_a Q(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Deep Q-learning with Experience Replay Algorithm (Part 2)

Algorithm 1 deep Q - learning with experience replay (cont.)

10: Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

11: Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

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

13: 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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

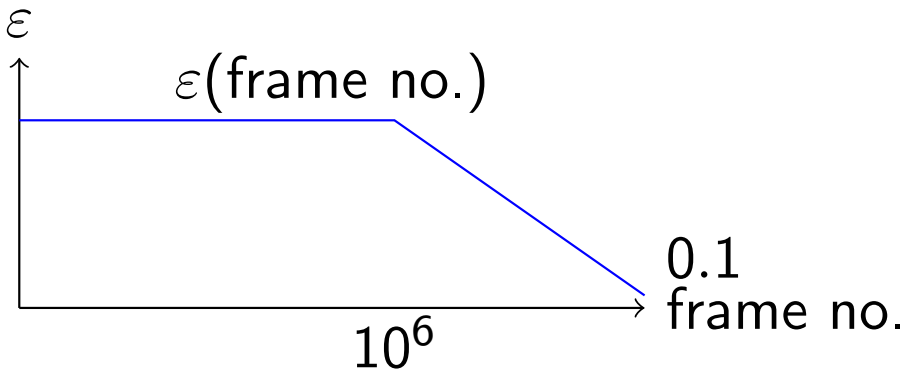
Every C steps reset $\hat{Q} = Q$

End For

End For

Training details

- Separate network for each game. Same architecture, learning algorithm and hyperparameter settings (informally selected due to high computational cost)
- Clipped positive rewards at 1 and negative rewards at -1, leaving 0 rewards unchanged. Facilitates same learning rate across multiple games but fails to differentiate between rewards of different magnitude.



Feature	Value
Behavior policy	ε-greedy
# training frames	50 million (~38 days)
Replay memory	1 million frames
Frame skipping	Yes ($k = 4$)

Evaluation Procedure

- Networks were evaluated 30 times per game for (up to 5 min each time) with different initial random conditions and an ε-greedy policy with $\varepsilon = 0.05$ to minimize the possibility of overfitting during evaluation.
- Random agent chose random actions at 10 Hz (every sixth frame), repeating its last action on intervening frames.
- Professional human tester used the same 60 Hz muted emulator without pausing, saving or reloading games. The human reward was averaged from around 20 episodes of each game (up to 5min each time), following around 2 h of practice playing each game.