Human-level control through deep reinforcement learning

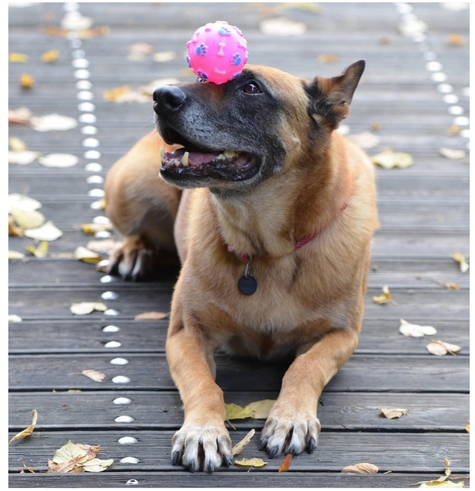
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, elen King, Dharshan Kumaran, Daan Wierstra, Shane Legg& Demis Hassabis

Presented By: Bharath Kollanur

What is Reinforcement Learning?

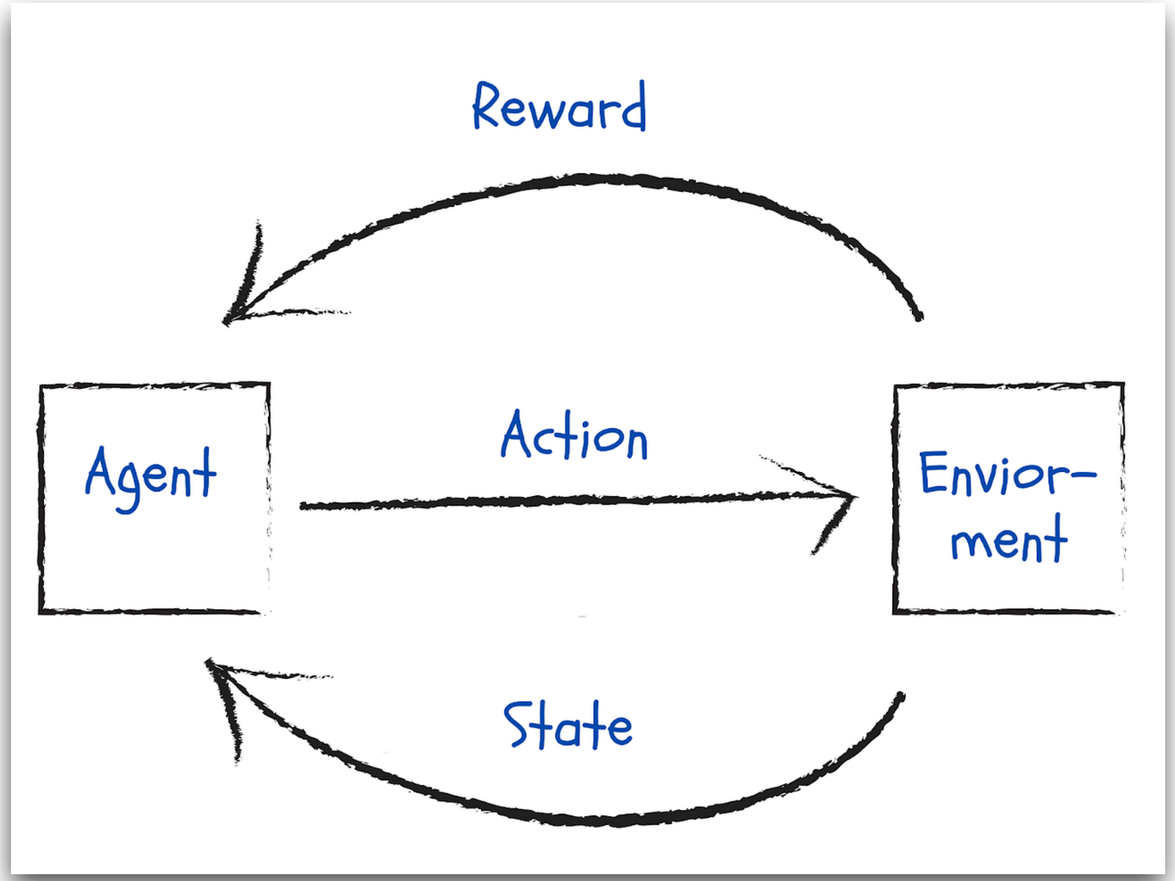
Reinforcement Machine Learning

When the kid tries to touch the fire anyway and *learns* that it is not good to touch fire!



Formalising Reinforcement Learning

- **Agent:** The learner or decision-maker that interacts with the environment to achieve a goal.
- **Environment:** The external system with which the agent interacts. It provides feedback on the agent's actions in the form of rewards.
- **State (S):** A representation of the current environment situation observed by the agent.
- **Action (A)**: The set of all possible moves or decisions the agent can make at any state.
- **Reward (R)**: Feedback from the environment after the agent acts. It indicates how good or bad the action was.
- **Policy (π)**: The strategy used by the agent to decide actions based on the current state.
- **Goal:** To maximize the cumulative reward over time.



Mathematical Formulation of Reinforcement Learning

- **Agent's Goal:**
  - The **goal** of the agent is to choose actions (a) in each state (s) such that it **maximises the cumulative future reward**.
- **Cumulative Future Reward:**

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots.$$

  Here r_t is the reward received at time t and γ is the discount factor.

- **Action Value Function:**
  - The **action-value function** ($Q^*(s, a)$) represents the **maximum expected cumulative reward** that the agent can achieve:

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

  - π**:** A policy is a mapping from states to actions, defining how the agent chooses actions.

Limitations of Handcrafted Features in Traditional RL

- **Handcrafted Features:**
  - In traditional reinforcement learning, experts manually design these features based on their understanding of the problem domain.
- **Limitations:**
  - They are often task-specific and do not generalize well to new problems or environments.
  - Designing them can be time-consuming and labour-intensive.
  - They may fail to capture the complexity of high-dimensional data, like raw image pixels or unstructured data

Solution: Deep Q-Networks

- The **deep Q-network (DQN)** introduced in this paper eliminates the need for handcrafted features by directly processing raw sensory inputs (e.g., raw pixel images).
- It uses deep convolutional neural networks to automatically learn hierarchical feature representations that are relevant to the task.

Traditional RL Working

- In traditional RL, the **action-value function** $Q(s, a)$represents the **expected cumulative reward** starting from state s, taking action a, and then following a specific policy.

$$Q(s, a) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots . \mid s_t = s, a_t = a]$$

- **Tabular Representation:** $Q(s, a)$ is stored as a **table** for every state s and action a.
- **Update Rule (Q-Learning):** $Q(s, a)$ is updated using the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$$

- **Limitations:**
  - **State Explosion:** For large or continuous state-action spaces (e.g., images or robotics), storing a $Q(s, a)$ table becomes infeasible.
- The **Deep Q-Network (DQN)** solves the scalability issues by approximating $Q(s, a)$ using a neural network.

Key Differences Between Normal RL and DQN

Aspect	Normal RL	Deep Q-Network (DQN)
Representation of *Q(s,a)*	Tabular (explicitly stores values for all states and actions).	Neural network (approximates *Q(s,a)* using weights).
Scalability	Limited to small state-action spaces.	Scales to large, high-dimensional spaces (e.g., images).
Feature Engineering	Requires handcrafted features.	Learns features automatically from raw inputs.
Learning Process	Updates *Q(s,a)* directly using the Bellman equation.	Minimizes a loss function to train the neural network.

RL with Neural Networks is Unstable

- **Correlations in Observations:**
  - In reinforcement learning, the agent interacts with an environment step by step, so the data (observations or states) it collects are highly correlated.
  - For example, if you're playing a video game, the next frame of the game is very similar to the current frame.
- **Action Value Instability:**
  - The **policy** decides which actions the agent takes based on the current $Q(s, a)$.
  - As the neural network learns and updates $Q(s, a)$, the policy changes.
  - The constant change in the data distribution can destabilize the training process because the neural network has to adapt to continuously changing data.
- **Prediction Instability:**
  - In Q-learning, the **target value** for the neural network is:

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

  - The problem is that this target value y is computed using the same neural network that is being updated to predict $Q(s, a)$.

Addressing the Instabilities

- **Experience Replay:**
  - Instead of training the neural network on data collected in the most recent steps, all experiences (*s, a, r, s'*) are stored in a replay buffer.
  - During training, the algorithm randomly samples experiences from this buffer.
- **Target Network:**
  - The **target value** $y = r + \gamma \max_{a'} Q(s', a')$ is calculated using a separate copy of the neural network called the **target network**.
  - This target network is a clone of the main network but is updated less frequently
- **How do these methods work together?**
  - **Experience Replay** smooths the training data, reducing bias and ensuring that the neural network learns from a diverse set of past experiences.
  - The **Target Network** stabilizes the learning targets, preventing rapid oscillations and divergence during training.

Deep Q - Network

- **Loss Function:**
  - The **loss function** measures the error between the predicted $Q(s, a)$ and the target value y:

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

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

    r: Immediate reward.

    γ: Discount factor, which determines the importance of future rewards $(0 \leq \gamma \leq 1)$.

    $\max_{a'} Q(s', a'; \theta_i^-)$: Maximum predicted value for the next state s', computed using the **target network**.

- The loss ensures the network learns to make the predicted value $Q(s, a; \theta_i)$ closer to the target y.

Algorithm

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

Evaluation

- **Test Environment:**
  - 49 Atari 2600 Games.
- **Performance Highlights:**
  - Outperformed best RL methods in **43 out of 49 Atari games**.
  - Achieved ≥75% of human performance in 29 games.
- **Insights:**
  - Same network architecture, hyperparameters, and training algorithm used across all games.
  - Capable of generalizing across diverse games with minimal prior knowledge.
  - Demonstrated **human-like learning** from raw sensory data.
  - Struggled with games requiring **long-term planning** (e.g., Montezuma's Revenge).

Evaluation

Comparison of games scores obtained by DQN agents with methods from the literature and a professional human games tester

Game	Random Play	Best Linear Learner	Contingency (SARSA)	Human	DQN (± std)	Normalized DQN (% Human)
Alien	227.8	939.2	103.2	6875	3069 (±1093)	42.7%
Amidar	5.8	103.4	183.6	1676	739.5 (±3024)	43.9%
Assault	222.4	628	537	1496	3359(±775)	246.2%
Asterix	210	987.3	1332	8503	6012 (±1744)	70.0%
Asteroids	719.1	907.3	89	13157	1629 (±542)	7.3%
Atlantis	12850	62687	852.9	29028	85641(±17600)	449.9%
Bank Heist	14.2	190.8	67.4	734.4	429.7 (±650)	57.7%
Battle Zone	2360	15820	16.2	37800	26300 (±7725)	67.6%
Beam Rider	363.9	929.4	1743	5775	6846 (±1619)	119.8%
Bowling	23.1	43.9	36.4	154.8	42.4 (±88)	14.7%
Boxing	0.1	44	9.8	4.3	71.8 (±8.4)	1707.9%
Breakout	1.7	5.2	6.1	31.8	401.2 (±26.9)	1327.2%
Centipede	2091	8803	4647	11963	8309(±5237)	63.0%
Chopper Command	811	1582	16.9	9882	6687 (±2916)	64.8%
Crazy Climber	10781	23411	149.8	35411	114103 (±22797)	419.5%
Demon Attack	152.1	520.5	0	3401	9711 (±2406)	294.2%
Double Dunk	-18.6	-13.1	-16	-15.5	-18.1 (±2.6)	17.1%
Enduro	0	129.1	159.4	309.6	301.8 (±24.6)	97.5%
Fishing Derby	-91.7	-89.5	-85.1	5.5	-0.8 (±19.0)	93.5%
Freeway	0	19.1	19.7	29.6	30.3 (±0.7)	102.4%
Frostbite	65.2	216.9	180.9	4335	328.3 (±250.5)	6.2%
Gopher	257.6	1288	2368	2321	8520 (±3279)	400.4%

Evaluation

The effects of replay and separating the target Q-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

Comparison of DQN performance with linear function approximator

Game	DQN	Linear
Breakout	316.8	3.00
Enduro	1006.3	62.0
River Raid	7446.6	2346.9
Seaquest	2894.4	656.9
Space Invaders	1088.9	301.3

Team Work

Rishab and I worked together to create discussion slides and also helped me with the slides.


Q
A

Discussion Questions

- **DQ1:** DQN performed well across 49 games with minimal prior knowledge. What features of the architecture enable this generalization, and how might this approach extend to other real-world applications?
- **DQ2:** The paper highlights challenges in games requiring long-term planning (e.g., Montezuma's Revenge). What modifications or enhancements could make DQN more effective in such scenarios?


THANK
YOU