Deep Reinforcement Learning

David Silver, Google DeepMind

Reinforcement Learning: AI = RL

RL is a general-purpose framework for artificial intelligence

- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal

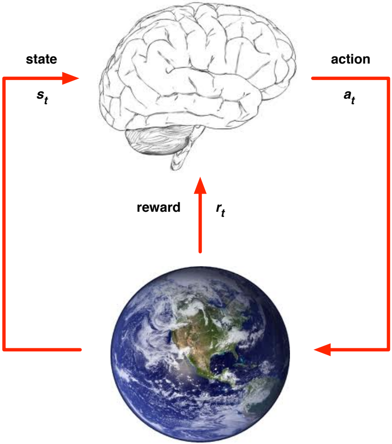
RL in a nutshell:

- Select actions to maximise future reward

We seek a single agent which can solve any human-level task

- The essence of an intelligent agent

Agent and Environment



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

Examples of RL

- Control physical systems: walk, fly, drive, swim, ...
- Interact with users: retain customers, personalise channel, optimise user experience, ...
- Solve logistical problems: scheduling, bandwidth allocation, elevator control, cognitive radio, power optimisation, ..
- Play games: chess, checkers, Go, Atari games, ...
- Learn sequential algorithms: attention, memory, conditional computation, activations, ...

Policies and Value Functions

- Policy π is a behaviour function selecting actions given states

$$a = \pi(s)$$

- Value function $Q^{\pi}(s, a)$ is expected total reward from state s and action a under policy π

$$Q^{\pi}(s, a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

"How good is action a in state s?"

Approaches To Reinforcement Learning

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Model-based RL

- Build a transition model of the environment
- Plan (e.g. by lookahead) using model

Deep Reinforcement Learning

- Can we apply deep learning to RL?
- Use deep network to represent value function / policy / model
- Optimise value function / policy /model end-to-end
- Using stochastic gradient descent

Bellman Equation

- Value function can be unrolled recursively

$$\begin{aligned} Q^{\pi}(s, a) &= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right] \\ &= \mathbb{E}_{s'}\left[r + \gamma Q^{\pi}(s', a') \mid s, a\right] \end{aligned}$$

- Optimal value function $Q^*(s, a)$ can be unrolled recursively

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

- Value iteration algorithms solve the Bellman equation

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

Deep Q-Learning

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

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

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Stability Issues with Deep RL

Naive Q-learning oscillates or diverges with neural nets

1. Data is sequential
   - Successive samples are correlated, non-iid
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-Networks

DQN provides a stable solution to deep value-based RL

1. Use experience replay
   - Break correlations in data, bring us back to iid setting
   - Learn from all past policies
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
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Optimise MSE between Q-network and Q-learning targets, e.g.

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

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 w^-

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

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

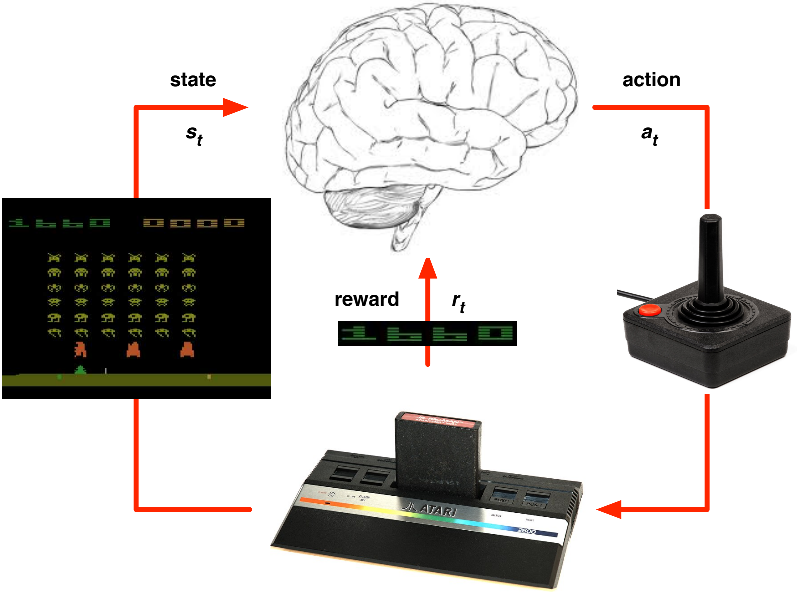
$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

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

Stable Deep RL (3): Reward/Value Range

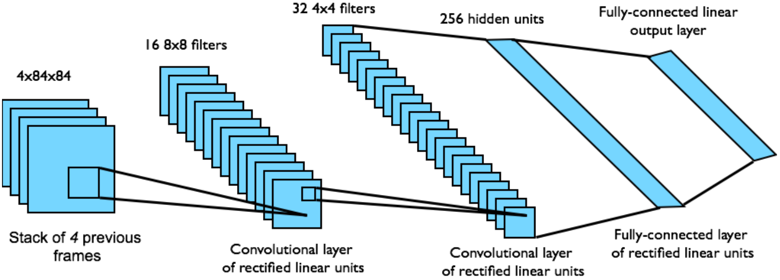
- DQN clips the rewards to $[-1, +1]$
- This prevents Q-values from becoming too large
- Ensures gradients are well-conditioned
- Can't tell difference between small and large rewards

Reinforcement Learning in Atari



DQN in Atari

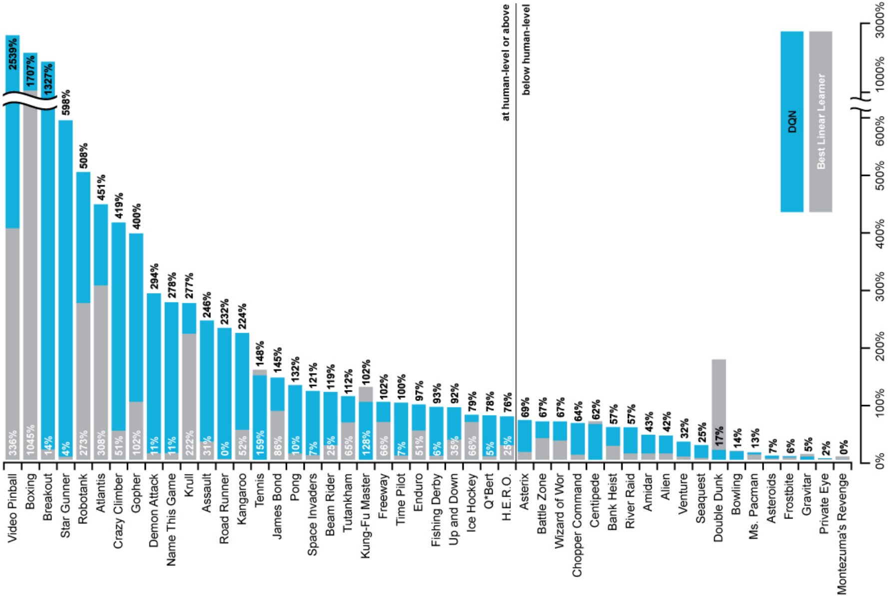
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

[Mnih et al.]

DQN Results in Atari



DQN Demo

How much does DQN help?

DQN

	Q-learning	Q-learning + Target Q	Q-learning + Replay	Q-learning + Replay + Target Q
Breakout	3	10	241	**317**
Enduro	29	142	831	**1006**
River Raid	1453	2868	4103	**7447**
Seaquest	276	1003	823	**2894**
Space Invaders	302	373	826	**1089**

Normalized DQN

- Normalized DQN uses true (unclipped) reward signal
- Network outputs a scalar value in "stable" range,

$$U(s, a, w) \in [-1, +1]$$

- Output is scaled and translated into Q-values,

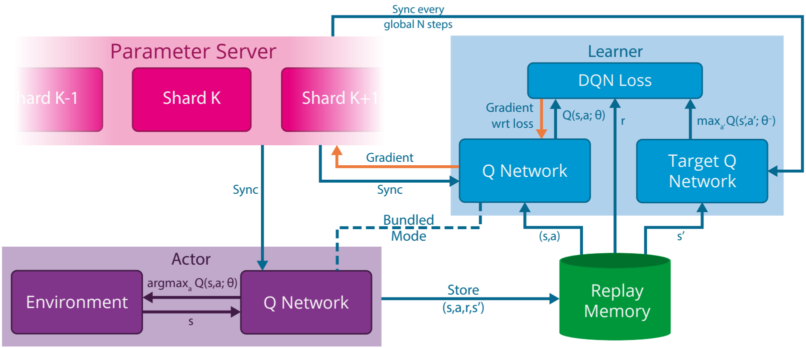
$$Q(s, a, w, \sigma, \pi) = \sigma U(s, a, w) + \pi$$

- π, σ are adapted to ensure $U(s, a, w) \in [-1, +1]$
- Network parameters w are adjusted to keep Q-values constant

$$\sigma_1 U(s, a, w_1) + \pi_1 = \sigma_2 U(s, a, w_2) + \pi_2$$

Demo: Normalized DQN in PacMan

Gorila (GOogle ReInforcement Learning Architecture)



- Parallel acting: generate new interactions
- Distributed replay memory: save interactions
- Parallel learning: compute gradients from replayed interactions
- Distributed neural network: update network from gradients

Stable Deep RL (4): Parallel Updates

Vanilla DQN is unstable when applied in parallel. We use:

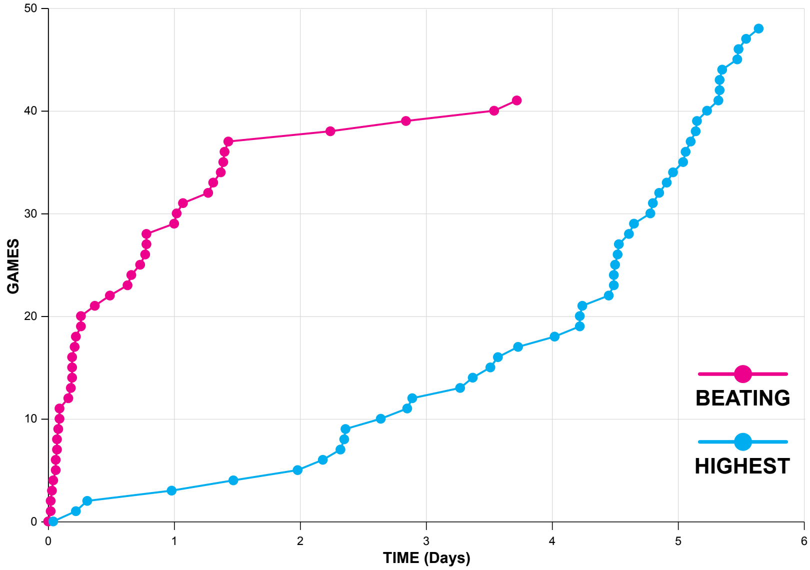
- Reject stale gradients
- Reject outlier gradients $g > \mu + k\sigma$
- AdaGrad optimisation

Gorila Results

Using 100 parallel actors and learners

- Gorila significantly outperformed Vanilla DQN
  - on 41 out of 49 Atari games
- Gorila achieved x2 score of Vanilla DQN
  - on 22 out of 49 Atari games
- Gorila matched Vanilla DQN results 10x faster
  - on 38 out of 49 Atari games

Gorila DQN Results in Atari: Time To Beat DQN



Deterministic Policy Gradient for Continuous Actions

- Represent deterministic policy by deep network $a = \pi(s, u)$ with weights u
- Define objective function as total discounted reward

$$J(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + ...\right]$$

- Optimise objective end-to-end by SGD

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s\left[\frac{\partial Q^{\pi}(s, a)}{\partial a}\frac{\partial \pi(s, u)}{\partial u}\right]$$

  - Update policy in the direction that most improves Q
  - i.e. Backpropagate critic through actor

Deterministic Actor-Critic

Use two networks: an actor and a critic

- Critic estimates value of current policy by Q-learning

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s'), w) - Q(s, a, w)\right)\frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Actor updates policy in direction that improves Q

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s\left[\frac{\partial Q(s, a, w)}{\partial a}\frac{\partial \pi(s, u)}{\partial u}\right]$$

Deterministic Deep Actor-Critic

- Naive actor-critic oscillates or diverges with neural nets
- DDAC provides a stable solution

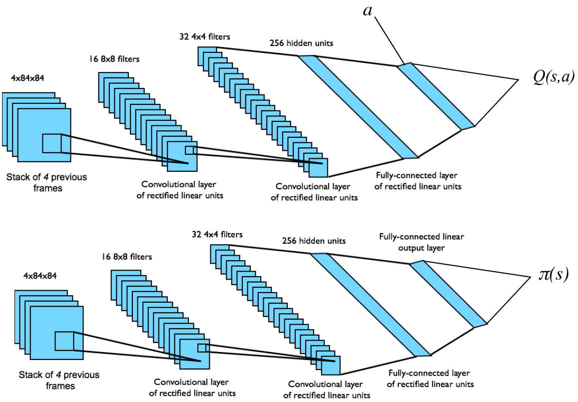
1. Use experience replay for both actor and critic
2. Use target Q-network to avoid oscillations

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma Q(s', \pi(s'), w^-) - Q(s,a,w) \right) \frac{\partial Q(s,a,w)}{\partial w} \right]$$

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\frac{\partial Q(s,a,w)}{\partial a} \frac{\partial \pi(s,u)}{\partial u} \right]$$

DDAC for Continuous Control

- End-to-end learning of control policy from raw pixels s
- Input state s is stack of raw pixels from last 4 frames
- Two separate convnets are used for Q and π
- Physics are simulated in MuJoCo



[Lillicrap et al.]

DDAC Demo

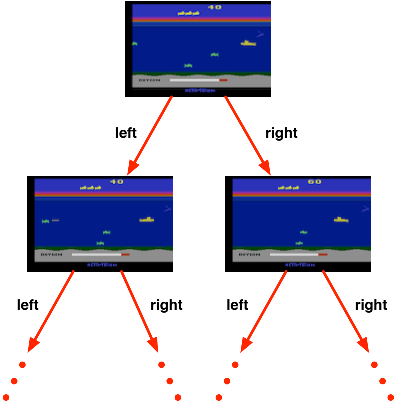
Model-Based RL

Learn a transition model of the environment

$$p(r, s' \mid s, a)$$

Plan using the transition model

- e.g. Lookahead using transition model to find optimal actions



Deep Models

- Represent transition model $p(r, s' \mid s, a)$ by deep network
- Define objective function measuring goodness of model
- e.g. number of bits to reconstruct next state (Gregor et al.)
- Optimise objective by SGD

DARN Demo

Challenges of Model-Based RL

Compounding errors

- Errors in the transition model compound over the trajectory
- By the end of a long trajectory, rewards can be totally wrong
- Model-based RL has failed (so far) in Atari

Deep networks of value/policy can "plan" implicitly

- Each layer of network performs arbitrary computational step
- n-layer network can "lookahead" n steps
- Are transition models required at all?

Deep Learning in Go

Monte-Carlo search

- Monte-Carlo search (MCTS) simulates future trajectories
- Builds large lookahead search tree with millions of positions
- State-of-the-art 19×19 Go programs use MCTS
- e.g. First strong Go program *MoGo*

(Gelly et al.)

Convolutional Networks

- 12-layer convnet trained to predict expert moves
- Raw convnet (looking at 1 position, no search at all)
- Equals performance of MoGo with 10^5 position search tree

(Maddison et al.)

Program	Accuracy
Human 6-dan	$\sim$ 52%
12-Layer ConvNet	55%
8-Layer ConvNet*	44%
Prior state-of-the-art	31-39%

Program	Winning rate
GnuGo	97%
MoGo (100k)	46%
Pachi (10k)	47%
Pachi (100k)	11%

*Clarke & Storkey

Conclusion

- RL provides a general-purpose framework for AI
- RL problems can be solved by end-to-end deep learning
- A single agent can now solve many challenging tasks
- Reinforcement learning $+$ deep learning $=$ AI

Questions?

"The only stupid question is the one you never ask" -*Rich Sutton*