Machine Learning
Reinforcement Learning

Dr Chris Willcocks
Department of Computer Science

Durham University

Lecture Overview

Durham University

Recap

- GANs, AEs, Latent spaces, UNet….
  - Mostly differentiable problems up until now
- Synaptic plasticity

Today's lecture: Reinforcement Learning

*"**Reinforcement**"*

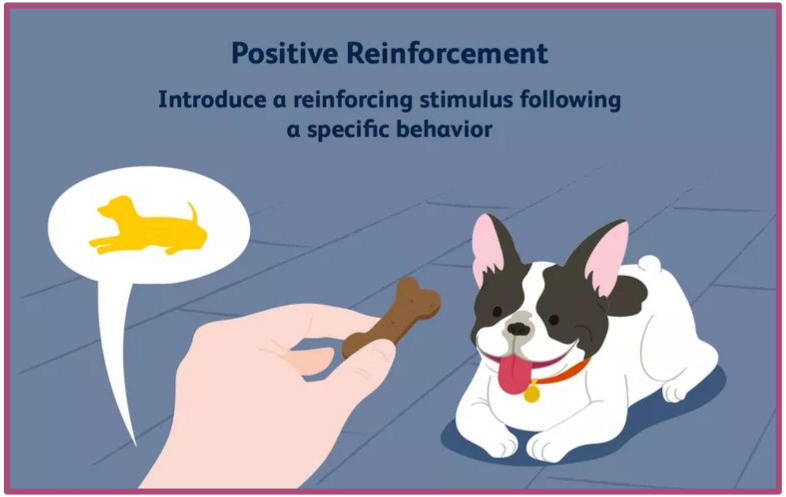


Illustration by Joshua Seong, Verywell

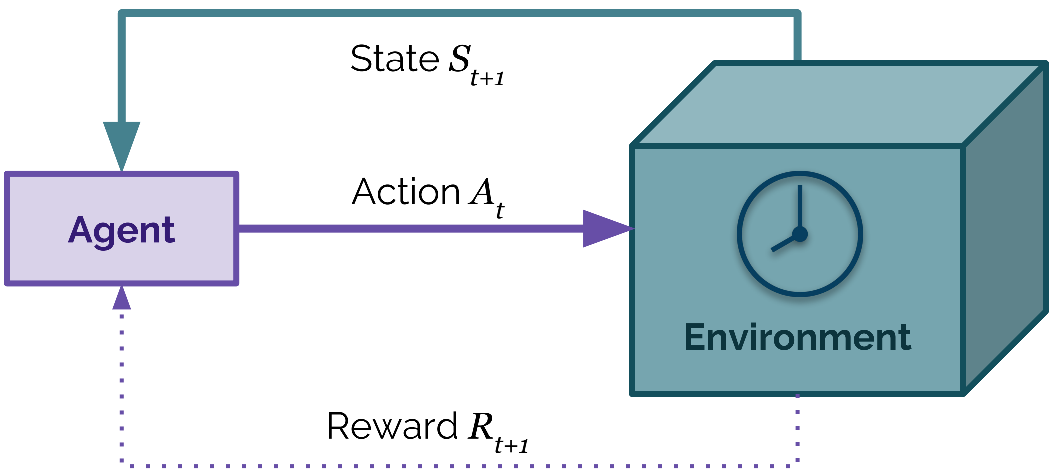
- "The action or process of reinforcing or strengthening"
- "The process of encouraging or establishing a belief or pattern of behaviour"

Further Reading: Reinforcement Learning: An Introduction Lilian Weng David Silver
(if you're interested in learning more about this field)

Reinforcement Learning

Given an agent in an unknown environment, learn how to take **smart actions** such as to **maximize cumulative rewards**

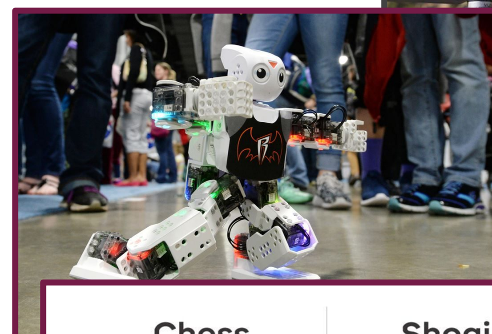
- There is **no supervisory signal**, only a notion of reward R_t at step t
- Feedback is **not immediate**
- **Time** is important (sequential, data is not *IID*)
- Agents actions affect subsequent environment state data

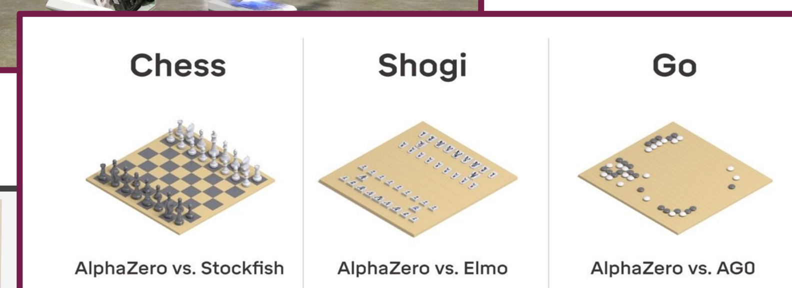


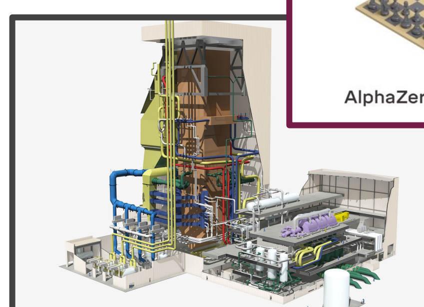
Examples

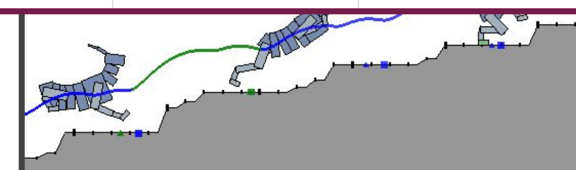
Control behaviour of complex systems

- Defeat champion at Go
- Do stunt manoeuvres in drones
- Beat humans at video games
- Find most efficient robot walking strategy over complex terrain
- Control power stations
- Self-driving cars
- Robotic cook dinner
- Investing in stock market
- ....









DeepMind Technologies, Ltd

Learning to drive and flip pancakes

Wayve: Learning to drive in a day

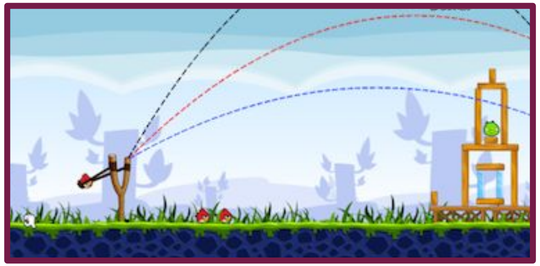
Learning to flip pancakes

Key Concepts

Interaction between **agent** and **environment** through sequential time steps $t = 1, 2, ..., T$. The agent *learns* about the environment, it learns the optimal **policy** and makes decisions about how to choose the next action.

Interaction sequence is described by an **episode** (**trajectory**) consisting of **states**, **actions**, and **rewards** at given time steps: S_t, A_t and R_t:

$$S_1, A_1, R_2, S_2, A_2, ..., S_T$$



There are many categories of RL algorithm, generally they are:

- **Model-based:** The agent learns a model of the environment
- **Model-free:** The agent learns which actions to take without an environment model
- **On-policy:** Learning on the job
- **Off-policy:** Learning optimal policy independent of current policy being executed

Key Concepts
Modelling the environment's transition and reward

The **model** describes the environment

1. Transition probability function P
2. Reward function R

Transition steps: $< s, a, s', r >$

https://www.placesyoullsee.com/25-most-dangerous-hiking-trails-in-the-world/

$$P(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

$$R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r | s, a)$$

Key Concepts
Policy and value

- The agent's "policy" π describes which action it should take in a given state.

  It can be deterministic: $a = \pi(s)$

  or stochastic:

  $$a = \pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

- The value function estimates the future reward for a state or an action. The future reward is called the **return**:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Key Concepts
Value functions (continued)

- The value function estimates the future reward for a **state** or an **action**.

The **state**-value for a state s is the expected return at time t

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

The **action**-value (Q-value, or “quality”) for a state-action pair is similarly:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

The **advantage** is the difference between the action-value and state-value:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

Key Concepts
Optimal value and policy

Durham University

The optimal value function $V_*(s)$ produces the maximum return

$$V_*(s) = \max_\pi V_\pi(s), Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

The optimal policy π_* is the policy that corresponds to the optimal value function

$$\pi_* = \arg\max_\pi V_\pi(s), \pi_* = \arg\max_\pi Q_\pi(s,a)$$

where $V_{\pi_*}(s) = V_*(s)$ and $Q_{\pi_*}(s,a) = Q_*(s,a)$

Markov Decision Process

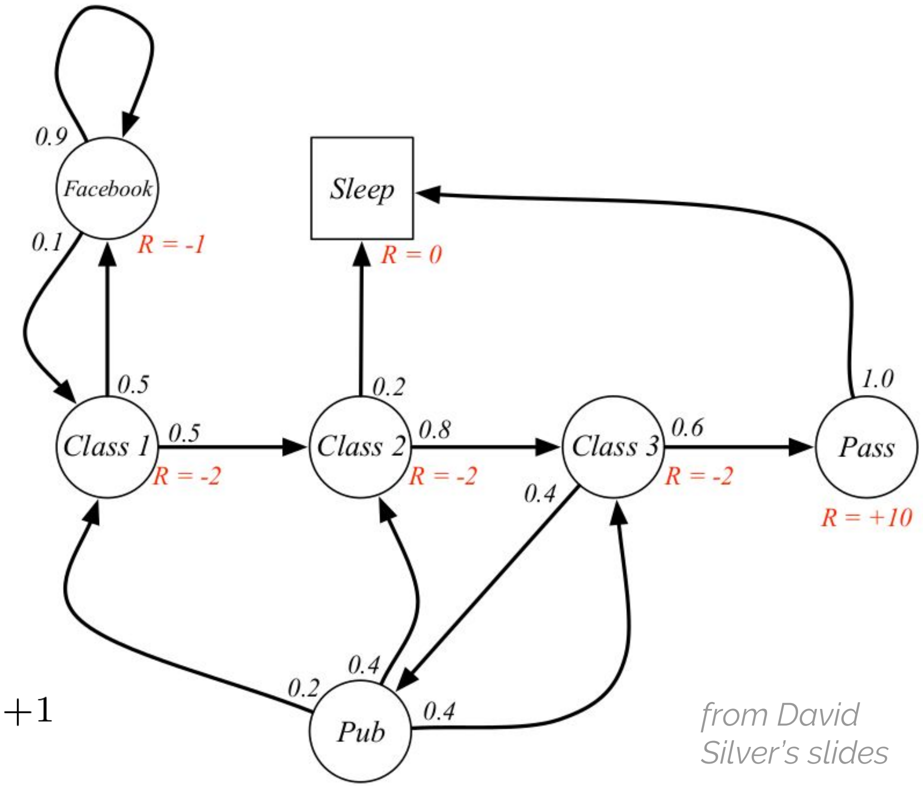
Durham University

Most *reinforcement* models as **Markov decision processes** $\mathcal{M} = <\mathcal{S}, \mathcal{A}, P, R, \gamma>$

1. Set of states $\mathcal{S}$
2. Set of actions $\mathcal{A}$
3. Transition function $P(s'|s,a)$
4. Reward function $R(s,a,s')$
5. Discount factor $\gamma \in [0,1]$

- Initial state s_0
- Discount factor $\gamma \in [0,1]$ for return G_t recall

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$



from David Silver's slides

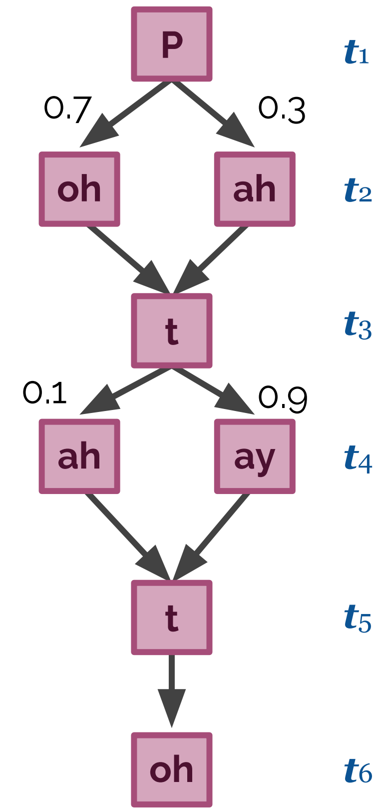
Markov States

"The future is independent of the past given the present"

A state $\mathcal{S}_t$ is *Markov* if and only if

$$\mathbb{P}[\mathcal{S}_{t+1}|\mathcal{S}_t] = \mathbb{P}[\mathcal{S}_{t+1}|\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_t]$$

- The state captures all information from the history
- Once the state is known, the history can be thrown away
- The state is a sufficient statistic of the future



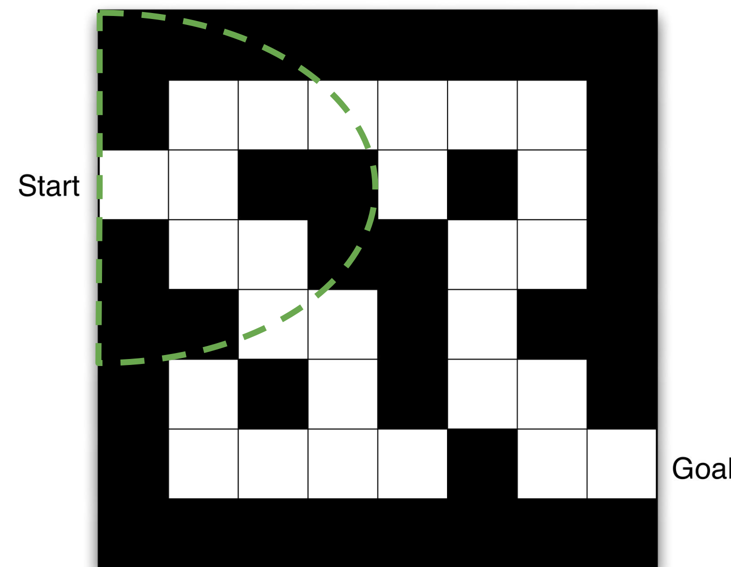
How to make decisions?

- Select actions to *maximise total future reward*
- Actions may have long term consequences
- Reward may be delayed
- Sometimes sacrifice of immediate reward is necessary for long-term reward
  - Refuelling a racing car

Each step t

1. Agent
   a. Executes $\boldsymbol{A}_t$
   b. Receives observation $\boldsymbol{O}_t$
   c. Receives reward $\boldsymbol{R}_t$
2. Environment
   a. Receives $\boldsymbol{A}_t$
   b. Emits observation $\boldsymbol{O}_{t+1}$
   c. Emits reward $\boldsymbol{R}_{t+1}$

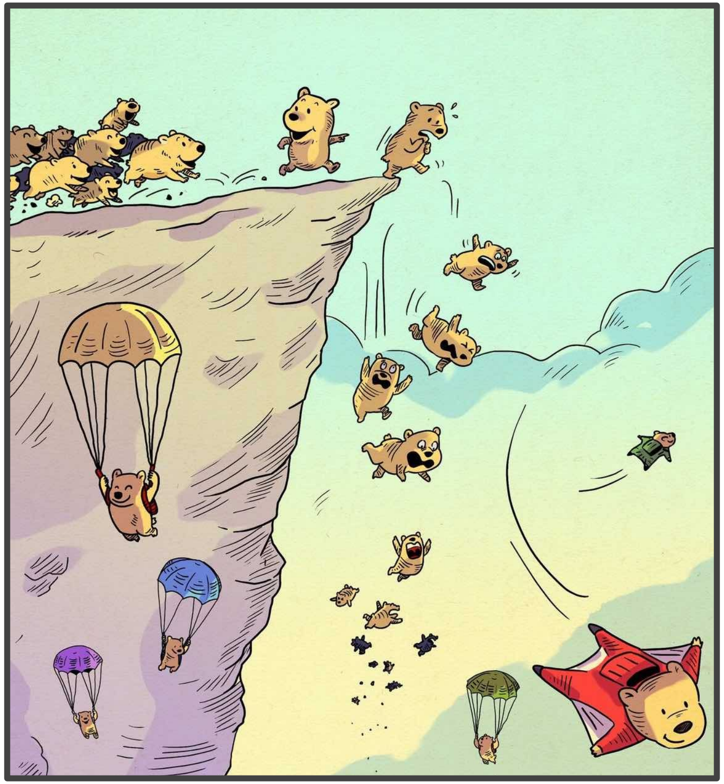
Partial observability



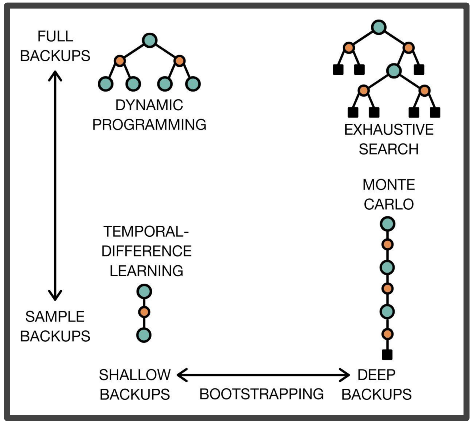
from slides by David Silver

Common Approaches

Durham University



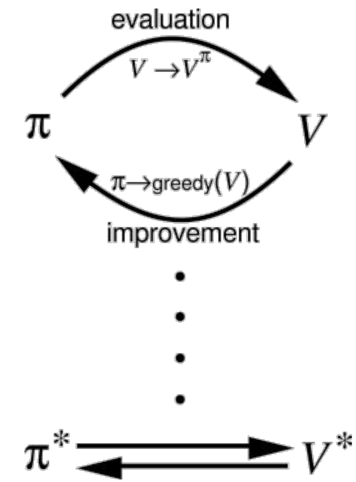
- Dynamic Programming
- Monte-Carlo methods
- Temporal-Difference Learning
- Policy Gradients
- Evolution Strategies



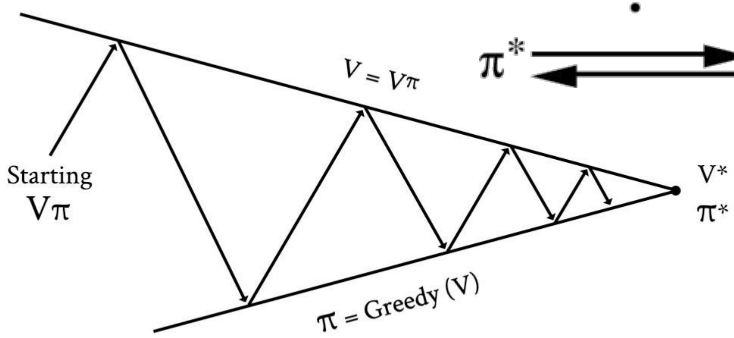
http://blog.otoro.net/2017/10/29/visual-evolution-strategies/

Common Approaches
Dynamic Programming

If we have complete information about the environment, and know $P(s'|s,a)$ and $R(s,a,s')$, we can use Dynamic Programming to directly solve MDP's by applying the Bellman Optimality Equations, which show that:



$$
\begin{aligned}
V(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma V(S_{t+1}) | S_t = s]
\end{aligned}
$$

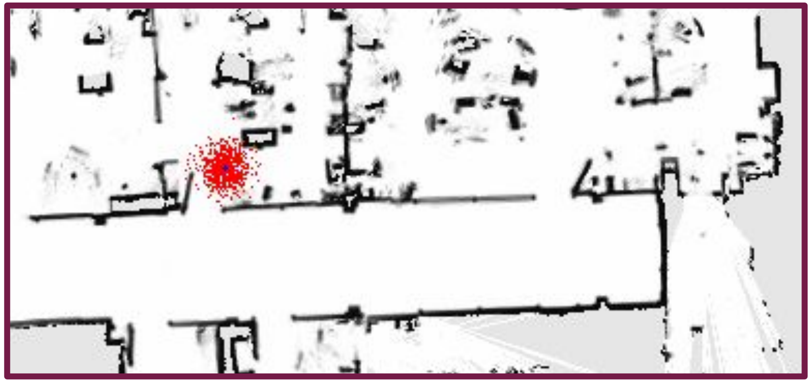


We can then iteratively evaluate the value function and improve the policy

$$\pi_0 \xrightarrow{\text{eval}} V_{\pi_0} \xrightarrow{\text{refine}} \pi_1 \xrightarrow{\text{eval}} V_{\pi_1} \xrightarrow{\text{refine}} \pi_2 \xrightarrow{\text{eval}} \dots \xrightarrow{\text{refine}} \pi_* \xrightarrow{\text{eval}} V_*$$

Common Approaches
Monte-Carlo Methods

Monte-Carlo (MC) methods simply learn through experience without modelling the environment dynamics.



Typical approach:

1. Improve the policy greedily with respect to current value function

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

2. Generate new trajectory with new policy π with ε-greedy policy.
3. Estimate the new state-action return Q, where

$$q_\pi(s,a) = \frac{\sum_{t=1}^{T} \left(1[S_t = s, A_t = a] \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}\right)}{\sum_{t=1}^{T} 1[S_t = s, A_t = a]}$$

Common Approaches
Temporal-Difference Learning

Temporal-Difference (TD) learning is where we slowly move the value function $V(S_t)$ towards $R_{t+1} + \gamma V(S_{t+1})$ which is the TD target by some learning rate hyper parameter α

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\textit{TD Target}} - V(S_t))$$

- No reliance on rewards
- No complete returns (incomplete episodes/trajectories)

Bootstrapping update targets with regard to existing estimates, rather than relying on actual rewards and complete returns.

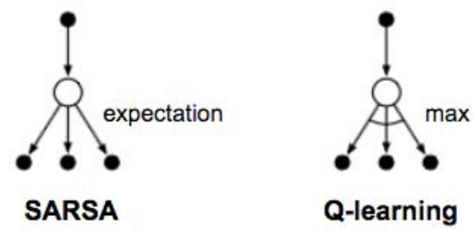
Common Approaches

Temporal-Difference Learning: **Q-Learning**

The previous slide also applies for estimating the return of state-action pairs:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$$

But what if we did an **off-policy** approach, instead of sampling SARSA with **ε**-greedy, estimating Q_* out of the best values independent of the current policy?



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t))$$

Memorizing $Q_*(.)$ for all state-action pairs is large/expensive. So we use a function approximator $Q(s, a; \theta)$ aka a **Deep Neural Network**.

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

Common Approaches
TD Learning: ***Deep*** **Q-Learning**

[PDF] Playing Atari with Deep Reinforcement Learning - University of ...
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
by V Mnih - Cited by 2031 - Related articles
However **reinforcement learning** presents several challenges from a **deep** ...
This **paper** demonstrates that a convolutional neural network can overcome

---

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    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 $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
  end for
end for

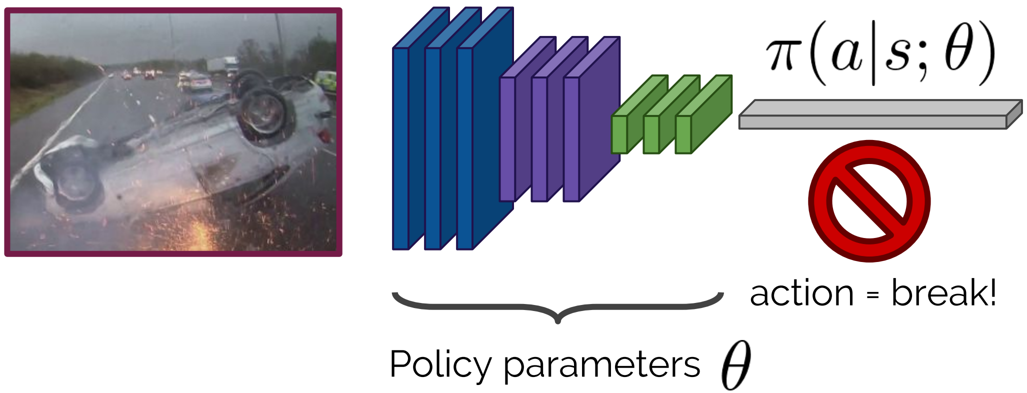
Achievement unlocked! You can now play Atari games!

Common Approaches

Policy Gradients

Previously, we've tried to learn the state/action value and choose actions.

What if we want to learn the policy function $\pi(a|s;\theta)$ with respect to some parameters θ?



How would we train this?

Common Approaches

Policy Gradients

First, we define the reward function

$$\mathcal{J}(\theta) = V_{\pi_\theta}(S_1) = \mathbb{E}_{\pi_\theta}[V_1]$$

To do gradient ascent, we need the partial derivative of the reward function with respect to the parameters, e.g. *numerically*

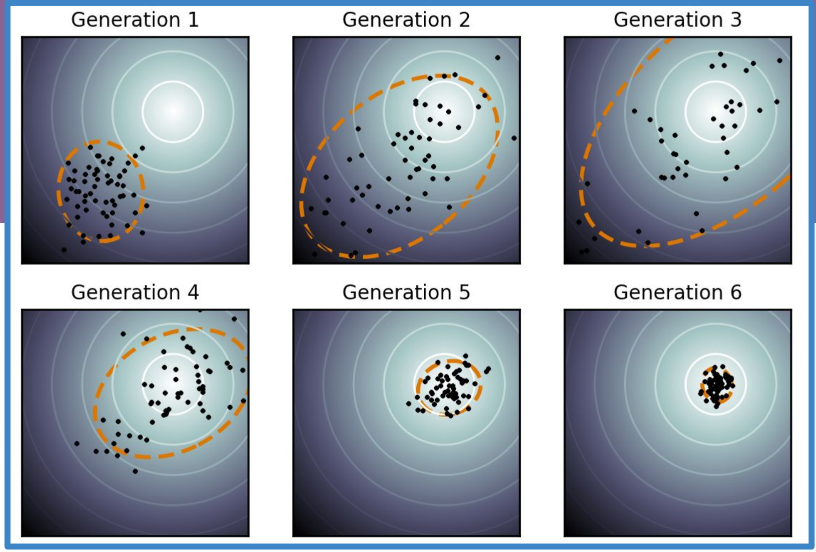
$$\frac{\partial \mathcal{J}(\theta)}{\partial \theta_k} \approx \frac{\mathcal{J}(\theta + \epsilon u_k) - \mathcal{J}(\theta)}{\epsilon}$$

where

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}[\nabla \ln \pi(a|s,\theta) Q_\pi(s,a)]$$

see **REINFORCE** and **Actor-Critic,** which are full learning algorithms that do this.
[Link to an excellent write-up by Lilian Weng] on policy-gradient algorithms.

Common Approaches
Evolution Strategies



If we satisfy:

1. Solutions able to freely interact with the environment and see what they do
2. Fitness for any solution can be evaluated

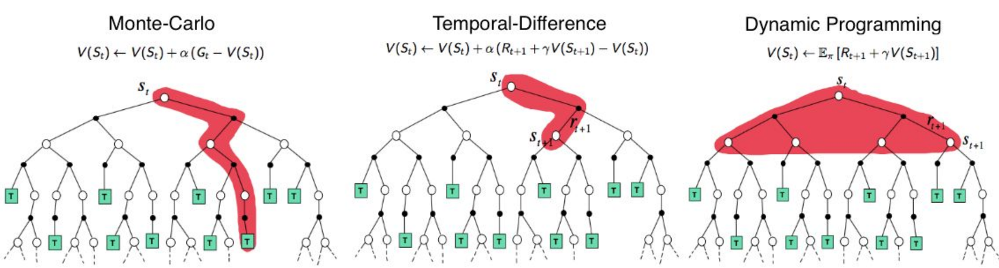
… then we can use **evolution strategies** (e.g. CMA-ES, Genetic Algorithm) using a non MDP-based approach without value approximation.

We assume a prior distribution over the policy parameters θ (e.g. a multivariate Gaussian) and sample the gradient accordingly

$$\nabla_\theta \mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma\epsilon) = \frac{1}{\sigma}\mathbb{E}_{\epsilon \sim N(0,I)}[F(\theta + \sigma\epsilon)\epsilon]$$

Take away points

- Reinforcement learning is a huge field still in its infancy...
    - There are lots of ways to build RL agents
    - There are lots of constraints in different applications
    - Quite expensive (compute, training examples)
    - **Exploitation vs exploration**
- We've only dipped our feet into it!



...where we're currently at

Appendix

Derivations

Durham University

1. Continuous form of policy reward function

$$
\begin{aligned}
\mathcal{J}(\theta) &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta) Q_\pi(s,a) \\
\nabla\mathcal{J}(\theta) &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \nabla\pi(a|s;\theta) Q_\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta) \frac{\nabla\pi(a|s;\theta)}{\pi(a|s;\theta)} Q_\pi(s,a) \\
&= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi(a|s;\theta) \nabla \ln \pi(a|s;\theta) Q_\pi(s,a) \\
&= \mathbb{E}_{\pi_\theta}[\nabla \ln \pi(a|s;\theta) Q_\pi(s,a)]
\end{aligned}
$$

2. Lerp value-return for TD-learning

$$
\begin{aligned}
V(S_t) &\leftarrow (1-\alpha)V(S_t) + \alpha G_t \\
V(S_t) &\leftarrow V(S_t) + \alpha(G_t - V(S_t)) \\
V(S_t) &\leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))
\end{aligned}
$$

Appendix

Notation 1/3

s	state
a	action
$\mathcal{S}$	set of all nonterminal states
$\mathcal{S}^+$	set of all states, including the terminal state
$\mathcal{A}(s)$	set of actions possible in state s
$\mathcal{R}$	set of possible rewards
t	discrete time step
T	final time step of an episode
S_t	state at t
A_t	action at t
R_t	reward at t, dependent, like S_t, on A_{t-1} and S_{t-1}
G_t	return (cumulative discounted reward) following t

Appendix

Notation 2/3

π	policy, decision-making rule
$\pi(s)$	action taken in state s under *deterministic* policy π
$\pi(a \mid s)$	probability of taking action a in state s under *stochastic* policy π
$p(s', r \mid s, a)$	probability of transitioning to state s', with reward r, from s, a
$v_\pi(s)$	value of state s under policy π (expected return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s, a)$	value of taking action a in state s under policy π
$q_*(s, a)$	value of taking action a in state s under the optimal policy
$V_t(s)$	estimate (a random variable) of $v_\pi(s)$ or $v_*(s)$
$Q_t(s, a)$	estimate (a random variable) of $q_\pi(s, a)$ or $q_*(s, a)$
$\hat{v}(s,\mathbf{w})$	approximate value of state s given a vector of weights $\mathbf{w}$
$\hat{q}(s, a,\mathbf{w})$	approximate value of state–action pair s, a given weights $\mathbf{w}$
$\mathbf{w}, \mathbf{w}_t$	vector of (possibly learned) *weights* underlying an approximate value function
$\mathbf{x}(s)$	vector of features visible when in state s

Appendix

Notation 3/3

δ_t	temporal-difference error at t (a random variable, even though not upper case
$E_t(s)$	eligibility trace for state s at t
$E_t(s, a)$	eligibility trace for a state–action pair
$\mathbf{e}_t$	eligibility trace vector at t
γ	discount-rate parameter
ε	probability of random action in ε-greedy policy
α, β	step-size parameters
λ	decay-rate parameter for eligibility traces

David Silver's Slides
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Lilian Weng's Blog
https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html

Other notations used in some fields

State	s	x
Action	a	u
Reward	r	$-c$