Advanced Machine Learning and Autonomous Agents

Reinforcement Learning III

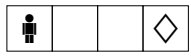
Jesse Read

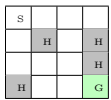
Version: February 12, 2025

Agenda Today

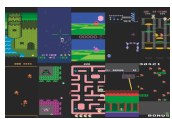
- Deep Q-Learning (DQN) because tabular Q-learning doesn't scale up
  We will see vanilla DQN and two improvements
- Policy-Gradient because policy search doesn't scale up
  We will look at REINFORCE and a few improvements
- Actor-Critic methods: We will combine both approaches – and lead into the state-of-the-art (DDPG, PPO, etc).

In other words, today we want to move from something like this ...





to something like this ...



Introduction

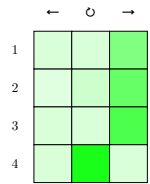
Reminder: Value-Based Methods

In value-based methods, we want to obtain

$$Q(s, a) \approx q^*(s, a)$$

since it gives our optimal policy (and optimal action to take):

$$a^* = \pi_*(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q^*(s, a)$$



Tabular methods include:

- Monte Carlo methods.
- Q-Learning (Off-policy Temporal Difference)
- SARSA (On-policy Temporal Difference)

Disadvantage of Q-Tables

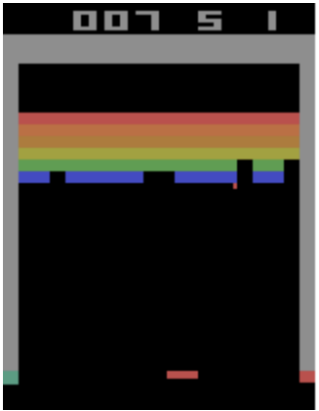
For example (where $\mathcal{A} = \{a_1, a_2\}$, $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$),

Q	a_1	a_2
s_1		
s_2		
s_3		
s_4		

Producing and updating a Q-table is not suitable when,

- large state space (up to $|\mathcal{S}| \times |\mathcal{A}|$ values)
- continuous state space or action space (no *table* possible!)

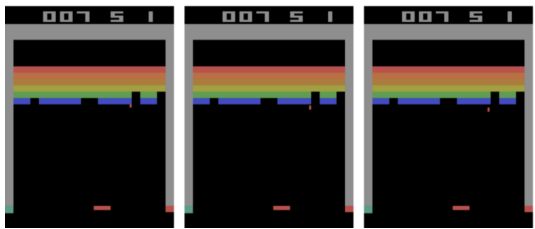
Breakout (1976)



See also: [1]

- $|\mathcal{S}| = 8^{84 \cdot 84}$ (8 colours, 84×84 screen)
- $|\mathcal{A}| = 2$ (left,right).

Recall need for Markov property! Is the ball coming or going?



A solution: frame stacking (now, $|\mathcal{S}| = 8^{84 \cdot 84 \cdot 3}$).

Doom (1993)

- $\mathcal{A} = \{\texttt{left}, \texttt{right}, \texttt{shoot}, \texttt{move}\}$
- resolution 320×200, virtually infinite state space.

[2]

Good news: Graphics cards capacity have scaled faster than game *mechanics*; the same game can be played on lower resolution at the cost of aesthetics (– the AI agent doesn't mind!).

So, we just need an efficient representation for the state space ...

Deep Q-Learning

A Reminder: Q-Learning (off-policy TD control)

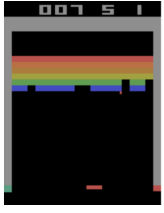
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

where we reduce the temporal difference error between target (empirical reward + estimate); and estimate; learning rate α. An estimate based on an estimate (bootstrap)!

For a given state we get vector $\widehat{\boldsymbol{q}} = Q(s, :)$ The greedy policy:

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q(s, a) = \operatorname*{argmax}_{a} \widehat{q}_a$$

Problem: Where to fit this s into a Q-table?

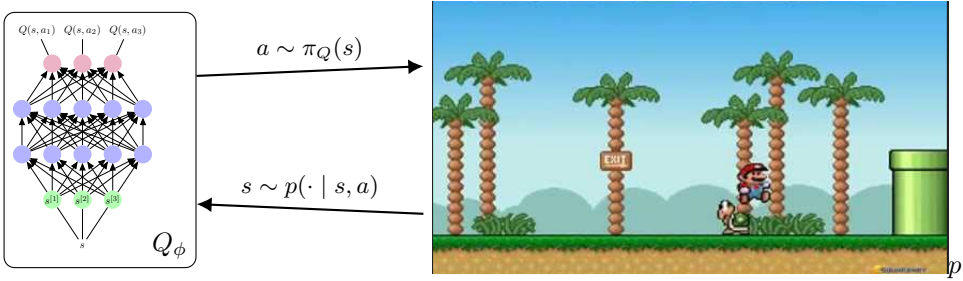


A Deep Representation of the Q-table

Replace $Q(s, :)$ with neural network $Q_\phi(s)$ of parameters ϕ, having one for each action;

$$\hat{\boldsymbol{q}} = Q_\phi(s) \approx [Q(s, a_1), \ldots, Q(s, a_{|\mathcal{A}|})]$$
$$Q(s, a) = \hat{q}_a$$

where a is the a-th output (action).



Deep Q-Learning (DQN): The Q-Update

Q Learning:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

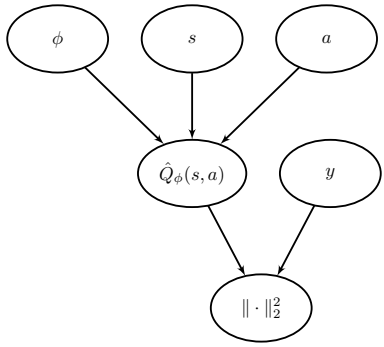
Deep Q-Learning

$$\phi = \phi + \alpha [\underbrace{(R_t + \gamma \max_{a \in \mathcal{A}} [Q_\phi(S_{t+1})]_a)}_{\text{target}} - \underbrace{[Q_\phi(S_t)]_{A_t}}_{\text{pred.}}] \nabla_\phi [Q_\phi(S_t)]_{A_t}$$

Note: we are using squared error

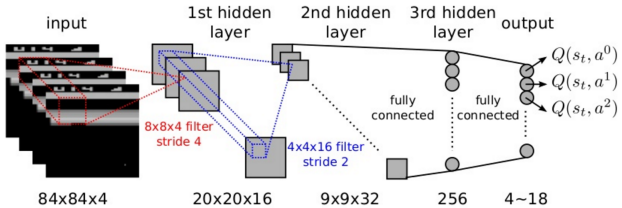
$$E(\phi) = (\text{target} - \text{pred.})^2$$

Computational Graph



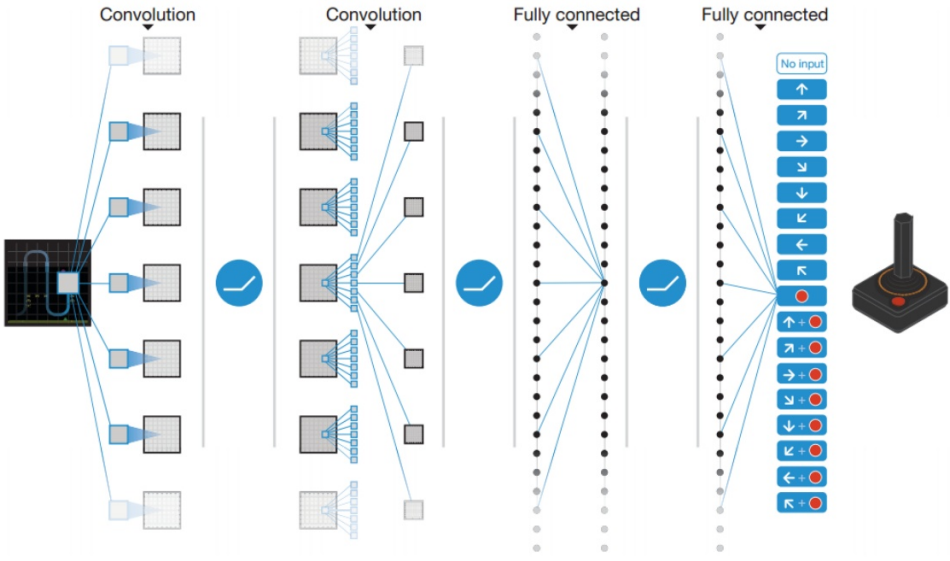
Assume target is fixed. It's not(!!) but just assume *for now*.

Example Architecture



A Deep Q Learning Network [3]

On Breakout: Deep Q-learning agent can be trained in approximately 40 hours on a Tesla K80 GPU ($\approx$ \$5000 in 2019)


Convolution
Convolution
Fully connected
Fully connected
No input

[4]

Deep Q-Learning – Vanilla Version

procedure DQN
  Initialise deep neural network Q_ϕ of parameters ϕ
  $s_1 \sim p(\cdot)$ ▷ Initial state of environment
  for $t = 1, \ldots, T$ **do**
    $a_t \sim \pi_\phi(\cdot \mid s_t)$ ▷ e.g., ϵ-greedy (on Q_ϕ)
    $r_t, s_{t+1} \sim p(\cdot, \cdot \mid s_t, a_t)$ ▷ Step in environment
    $y \leftarrow r_t + \gamma \max_a Q_\phi(s_{t+1}, a)$ ▷ Set the target
    $\widehat{y} \leftarrow Q_\phi(s_t, a_t)$ ▷ Set the estimate
    $g \leftarrow \nabla_\phi (y - \widehat{y})^2$ ▷ Get gradient
    $\phi \leftarrow \phi - \alpha g$ ▷ Learning update

Major Issue 1): Correlated Data

And its solution: Replay Buffer

Problem: Input data is highly correlated; samples are not iid. This can cause overfitting, forgetting, bias the network to learn just a replay.

Solution: A Replay Buffer; store many (s_t, a_t, r_t, s_{t+1}) tuples and sample from the buffer randomly during training.

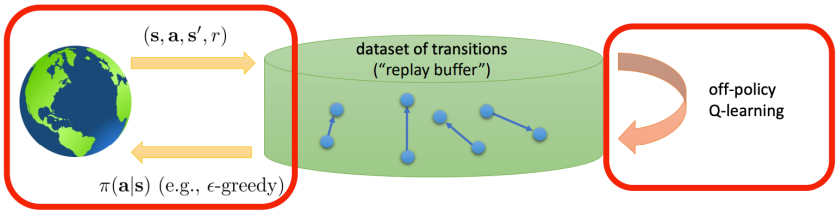


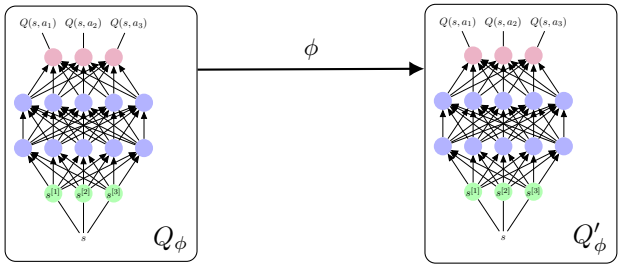
Image Credits: Sergey Levine

Major Issue 2): Moving Target

And its solution: Target Network

Problem: We are chasing a non-stationary target! Imagine in linear regression, y_t being different each time we reference it!

Solution: A separate Target Network; learning more slowly (update every so often).



Deep Q-Learning with Replay Buffer and Target Network

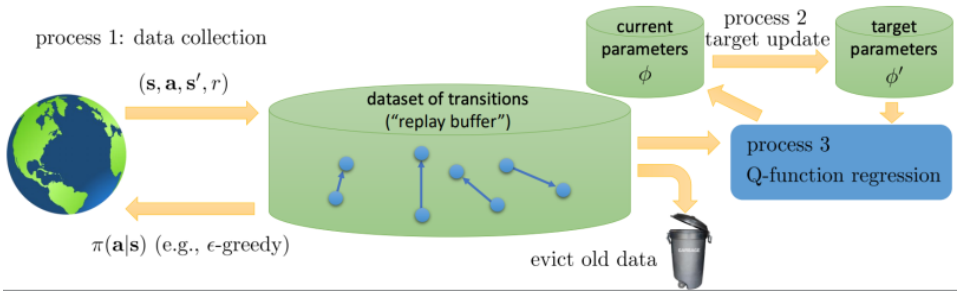


Image credit: Sergey Levine

Deep Q-Learning with Replay Buffer and Target Network

procedure DDQN'

Initialise replay buffer $\mathcal{D}$

Initialise deep neural networks Q_ϕ and $Q_{\phi'}$ (with $\phi' \leftarrow \phi$)

$s_1 \sim p(\cdot)$ ▷ Initial state of environment

for $t = 1, \ldots, T$ **do**

$a_t \sim \pi_\phi(\cdot \mid s_t)$ ▷ e.g., ϵ-greedy (on Q_ϕ)

$s_{t+1} \sim p(\cdot, \cdot \mid s_t, a_t)$ ▷ Step in environment

$\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, r_t, s_{t+1})$ ▷ Update buffer

$(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$ ▷ Sample from buffer

$y \leftarrow r_t + \gamma \max_a Q_{\phi'}(s_{t+1}, a)$ ▷ Set the target

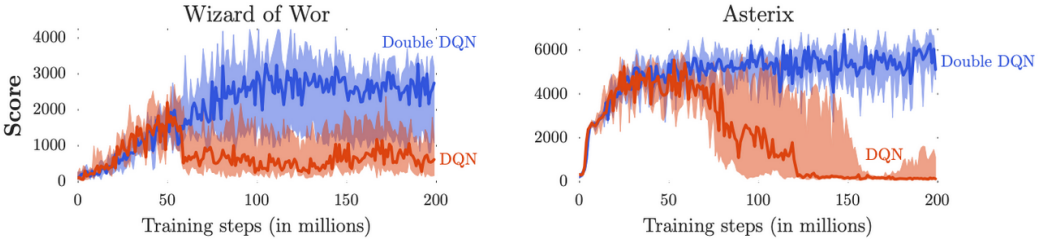
$\widehat{y} \leftarrow Q_\phi(s_t, a_t)$ ▷ Set the estimate

$g \leftarrow \nabla_\phi (y - \widehat{y})^2$ ▷ Get gradient

$\phi \leftarrow \phi - \alpha g$ ▷ Learning update

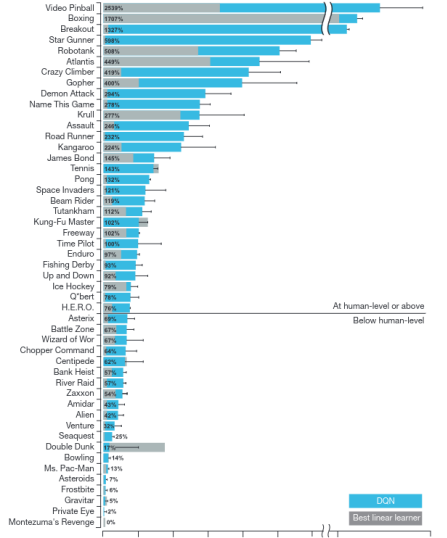
$\phi' \leftarrow \phi$ ▷ Update target network (sometimes)

The Effectiveness of DDQN (vs DQN)



source: [1]

The Effectiveness of Deep Q-Learning



source: [2], p3

Deep Q Learning For Everything?

DQN works so well on Atari games!

Deep Q Learning For Everything?

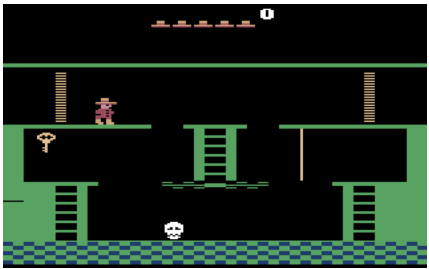
DQN works so well on Atari games!

Clarification: DQL works well on *some* Atari games.

Of the 49 games (identified by Mnih et al. 2015) all are fully observable and Markovian (given the last 4 frames of input)!

Also. . .

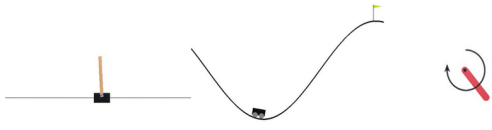
- Games are not everything (DQN often not very robust to a range of problems).
- Also, DQN is a bit ‘dodgy’ (not ‘real’ gradient descent)
- Cannot deal natively with continuous action spaces (e.g., robotic control)
- Struggles with high dimensional action spaces
- . . . with highly-stochastic environments, and
- . . . with sparse rewards



Policy Gradient

Policy Search (A Reminder)

Continuous state/action spaces:



Given a parametric policy,

$$a_t = \pi_\theta(s_t)$$

and performance metric (where policy π_θ interacts with environment p)

$$J(\theta) \approx \mathbb{E}_{\tau \sim p_\theta}[G]$$

we can search directly in parameter space, i.e., policy search:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmax}}\, J(\theta)$$

Many off-the-shelf optimization tools for this kind (no gradient) of search!

Does not work well in high-parameter spaces.

Policy Gradient Methods

Under performance metric $J(\theta)$, we want to do gradient ascent:

$$\theta \leftarrow \theta + \alpha \widehat{\nabla_\theta J(\theta)}$$

where $\widehat{\nabla_\theta J(\theta)}$ approximates $\nabla_\theta J(\theta)$.

With policy gradient methods: use gradient-based optimization!

Requirement 1: Stochastic Parametric Policy

Require a stochastic policy, parametrized by parameters θ:

$$\pi_\theta(a|s)$$

Example of Parametrized Stochastic Policy

where $a \in \{1, 2\}$, $s \in \mathbb{R}^d$, $\theta \in \mathbb{R}^d$, define policy

$$\pi_\theta(a|s) = \begin{cases} \sigma(\theta^\top s) & a = 1 \\ 1 - \sigma(\theta^\top s) & a = 2 \end{cases}$$

In practice, require $0 < \pi_\theta(a|s) < 1$ (should not be deterministic).

Requirement 2: Differentiable Policy

Require that the policy is differentiable everywhere, i.e., we have

$$\nabla_\theta \pi_\theta(a|s)$$

Example of Differentiable Policy

With the previously exemplified policy:

$$\nabla_\theta \pi_\theta(a|s) = \begin{cases} \sigma(1-\sigma)^\top s & a = 1 \\ -\sigma(1-\sigma)^\top s & a = 2 \end{cases}$$

Performance Metric $J(\theta)$, Episodic Case

Recall: We want to do gradient ascent on

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta}[G] = \mathbb{E}_{\tau \sim p_\theta}\left[\sum_{t=1}^{H} r(S_t, A_t)\right] \tag{1}$$

(let p_θ be the stocasticity of policy π_θ in environment p; trajectory τ over horizon H).

$$\nabla J(\theta) = \mathbb{E}_{(x_i, y_i) \sim p_\star}\left[\sum_{i=1}^{N} \ell(x_i, y_i)\right] \quad \triangleright \text{ In supervised learning, e.g., } \ell = p_\theta(y_i | x_i)$$

$$\nabla J(\theta) = \mathbb{E}_{\{(s_t, a_t)\} \sim p_\theta}\left[\sum_{t=1}^{H} r(s_t, a_t)\right] \quad \triangleright \text{ policy gradient?}$$

Problem: $J(\theta)$ depends both on the actions $a_t \sim \pi_\theta$, *and* the distribution of states $d_\theta = \int p_\theta(s' \mid s, a)\, \mathrm{d}a, s'$, yet both are determined by θ! So the policy changes the state distribution (in an unknown way).

How to proceed?

The Policy Gradient Theorem

Policy Gradient Theorem

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d_\theta(s) \sum_{a \in \mathcal{A}} Q^{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(a|s)$$

where, if $\eta(s)$ is the expected number of time steps spent on state s under policy π_θ, then $d_\theta(s) = \frac{\eta(s)}{\sum_s \eta(s)}$ is a distribution of states under π_θ; and Q^{π_θ} is the true (unknown) Q-function wrt π_θ.

The PG theorem thus gives us an analytical expression for $\nabla_\theta J(\theta)$, avoiding any derivative of transition dynamics!

But we're trying to *avoid* having to deal with Q-tables (Q^{π_θ})!

The Log-Derivative Trick and Others

Continuing (we're trying to get rid of Q^{π_θ} and d_θ) ...

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} d_\theta(s) \sum_{a\in\mathcal{A}} Q^{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a|s) && \triangleright \text{ Policy Gradient Theorem} \\
&= \sum_{s\in\mathcal{S}} d_\theta(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} && \triangleright \text{ since } f = \frac{\pi_\theta f}{\pi_\theta} \\
&= \sum_{s\in\mathcal{S}} d_\theta(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s) && \triangleright \text{ where } \nabla \ln f = \frac{\nabla f}{f} \\
&= \mathbb{E}_{s\sim d_\theta, a\sim\pi_\theta(\cdot|s)} \left[Q^{\pi_\theta}(s,a) \nabla_\theta \ln \pi_\theta(a|s) \right] && \triangleright \text{ since } \sum_x p(x) f(x) = \mathbb{E}_x f(x) \\
&= \mathbb{E}_{\tau\sim p_\theta} \left[G_t \nabla_\theta \ln \pi_\theta(a|s) \right] && \triangleright \text{ because } \mathbb{E}_{\tau\sim p_\theta}[G_t \mid s_t, a_t] = Q^{\pi_\theta}(s_t, a_t)
\end{aligned}$$

REINFORCE (Monte Carlo Policy Gradient)

Recall: We want to do stochastic gradient ascent using $\nabla_\theta J(\theta)$.
We have thus far:

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}\left[G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)\right]$$

which avoids dealing with Q^π and d_θ directly.

What to do with $\mathbb{E}_{\tau\sim p_\theta}$?

REINFORCE (Monte Carlo Policy Gradient)

Recall: We want to do stochastic gradient ascent using $\nabla_\theta J(\theta)$.
We have thus far:

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}\left[G_t \nabla_\theta \ln \pi_\theta(a_t|s_t)\right]$$

which avoids dealing with Q^π and d_θ directly.

What to do with $\mathbb{E}_{\tau\sim p_\theta}$?

Enter Monte Carlo: Replace $\mathbb{E}$ with average over N samples (trajectories) $\tau \sim p_\theta$ (horizon H);

$$\widehat{\nabla_\theta J(\theta)} = \frac{1}{N}\sum_{i=1}^{N}\left[\left(\sum_{t=1}^{H} G_t^{(i)} \nabla_\theta \ln \pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\right]$$

i.e., using empirical gain/returns $G_t^{(i)}$.

1: **procedure** REINFORCE
2: Initialise deep neural network π_θ
3: **for** Episode $i = 1, \ldots, N$ **do**
4: Generate $\tau \sim p_\theta$
5: **for** $t = 1, \ldots, H$ **do**
6: $G_t \leftarrow \sum_{t'=t}^{H} R_{t'}$ ▷ empirical return
7: $\widehat{\nabla_\theta J(\theta)} = G_t \nabla \ln \pi_\theta(a_t|s_t)$ ▷ gradient
8: $\theta \leftarrow \theta + \alpha \widehat{\nabla_\theta J(\theta)}$ ▷ gradient ascent

where

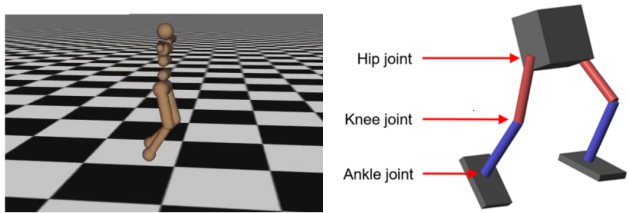
$$\nabla \ln \pi_\theta(a_t|s_t) = \frac{\nabla \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}$$

i.e., the gradient of the probability of taking action a_t over the probability of taking action a_t; it points in the direction in space Θ that most increases the probability of a_t from state s_t again in the future, and we step that way, $\propto G_t$; i.e., move in direction of highest G_t.

Recall: where π_θ is a deep neural network (Deep Policy Gradient)

Early Successes of Deep Policy Gradient

Stable legged walking has been a long standing control problem. DPG methods have shown some success here.



Kumar et al., 2018; www

Kumar et al. 2018: 41 hours of training on NVIDIA GeForce GTX 1050 Ti GPU, in simulation.

Good to know: Dealing with partial observations is no problem for policy gradient.

REINFORCE with Baseline

Typical of Monte Carlo methods: converge slowly; high variance.

Note that the $\nabla \pi_\theta$ does not depend on the reward trajectory $\{R_t\}$, but these rewards add a lot of variance to the Monte Carlo sampling as they appear in the return G_t.

We may use a baseline $b(s_t)$ to reduce variance.

$$\theta \leftarrow \theta + \alpha(G_t - b(s_t))\nabla \ln \pi_\theta(a_t|s_t)$$

Any $b(s_t)$ that does not vary with actions a_t is valid as a baseline (i.e., if $b(s_t)$ is independent of the policy parameters θ, it does not increase bias).

REINFORCE with Baseline

Typical of Monte Carlo methods: converge slowly; high variance.

Note that the $\nabla \pi_\theta$ does not depend on the reward trajectory $\{R_t\}$, but these rewards add a lot of variance to the Monte Carlo sampling as they appear in the return G_t.

We may use a baseline $b(s_t)$ to reduce variance.

$$\theta \leftarrow \theta + \alpha(G_t - b(s_t))\nabla \ln \pi_\theta(a_t|s_t)$$

Any $b(s_t)$ that does not vary with actions a_t is valid as a baseline (i.e., if $b(s_t)$ is independent of the policy parameters θ, it does not increase bias).

An example baseline (the estimated value of state s_t):

$$b(s_t) = \widehat{V}_\phi(s_t)$$

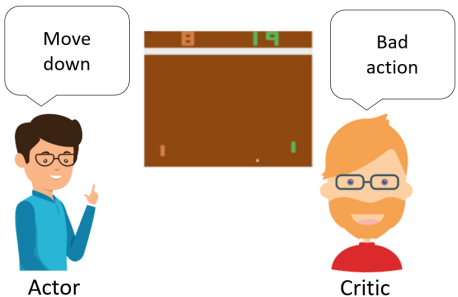
(we now also need to update ϕ in the loop).

Actor-Critic Methods

Actor-Critic Methods

Actor (policy; acts/behaves) $a_t \sim \pi_\theta(s_t)$

Critic (value function; critiques), e.g., $V_\phi(s_t)$ or $Q_\phi(s_t, a_t)$ – helps the agent to understand how good or bad different states and actions are in terms of expected return.



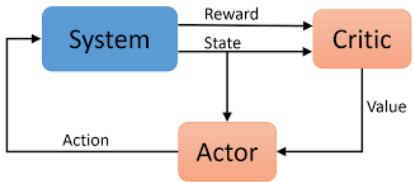


Image source: Wang et al., and [5]

1: **procedure** AC
2: Initialise deep neural networks π_θ and V_ϕ
3: **for** Episode $i = 1, \ldots, N$ **do**
4: $s_1 \sim p(\cdot)$ ▷ Initial state of environment
5: **for** $t = 1, \ldots, H$ **do**
6: $a_t \sim \pi_\theta(s_t)$
7: $r_t, s_{t+1} \sim p(\cdot, \cdot \mid s_t, a_t)$ ▷ Step in environment
8: $\delta \leftarrow r_t + V_\phi(s_{t+1}) - V_\phi(s_t)$ ▷ TD error
9: $\phi \leftarrow \phi + \alpha_\phi \delta \nabla_\phi V_\phi(s_t)$ ▷ Update Critic
10: $\theta \leftarrow \theta + \alpha_\theta V_\phi(s_t) \nabla_\theta \ln \pi_\theta(a_t|s_t)$ ▷ Update Actor

Advantage Actor-Critic (A2C)

$$Q^\pi(s,a) = V^\pi(s) + A^\pi(s,a)$$
$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

i.e., the advantage of taking action a over some average action.

Advantage Actor Critic (A2C):

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \ln \pi_\theta(a|s) A_\phi(s,a)\right]$$

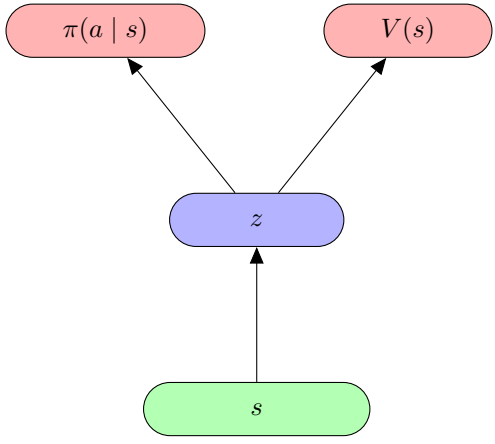
with advantage function

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

and, good to notice: $Q(s,a) = r(s,a) + V(s')$ (we only need to model V_ϕ!)

We should now update A_ϕ (and updates) into the AC algorithm.

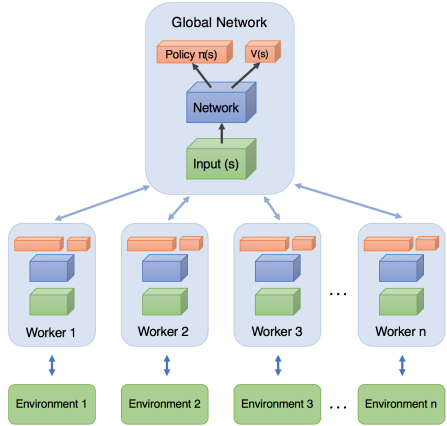
Also good to notice: efficient shared representation is possible;



Asyncronous Advantage Actor Critic (A3C)

A3C: multiple independent agents (own parameters) interacting with a different copy of the environment in parallel; asynchronously.

After n steps, update, then reset parameters to those of the global network.



Issues with Policy Gradient Methods

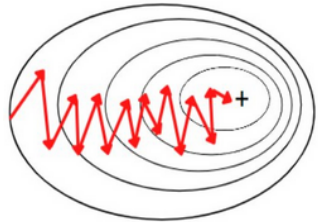


Image: huggingface.co

- Big policy updates are risky
- . . . and leads to instability/brittleness

Summary and Further Development (Towards PPO)

$$= \mathbb{E}_{\tau \sim p_\theta} \left[G \nabla_\theta \ln \pi_\theta(a \mid s) \right] \quad \triangleright \text{ Vanilla DPG}$$
$$= \mathbb{E}_{\tau \sim p_\theta} \left[(G - b(s)) \nabla_\theta \ln \pi_\theta(a \mid s) \right] \quad \triangleright \text{ with baseline}$$
$$= \mathbb{E}_{\tau \sim p_\theta} \left[\nabla_\theta \ln \pi_\theta(a \mid s) A_\phi \right] \quad \triangleright \text{ Advantage AC}$$
$$= \mathbb{E}_{(s_t, a_t) \sim p_{\theta_k}} \left[r_t(\theta) \nabla_\theta \ln \pi_\theta(a \mid s) \hat{A}_t \right] \quad \triangleright \text{ With importance weight } r_t(\theta) = \frac{\pi_\theta}{\pi_{\theta_k}}$$

where π_{θ_k} the policy that we collected recent samples from, and π_θ the policy we want to refine. The Proximal Policy Optimization (PPO) objective:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}_{(s_t, a_t) \sim p_{\theta_k}} \left[\min \left(r_t(\theta) \hat{A}_t,\ \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right]$$

Summary and Other Methods

Other Policy-Gradient and Actor-Critic Methods

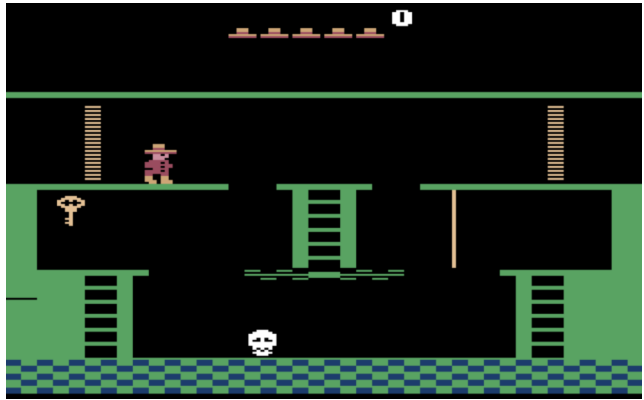
- Trust Region Policy Optimization (TRPO)
  - AC architecture
  - 'Trust regions' via constraints (measured as KL divergence between old and new policies)
- Proximal Policy Optimization (PPO)
  - A clipped surrogate objective function (instead of trust region)
  - Simpler computation/implementation (than TRPO)
  - Industry standard, used in, e.g., ChatGPT
- Deep Deterministic Policy Gradient (DDPG)
  - Combines DPG with DQN via deterministic policy
  - Allows for continuous action spaces
  - Off-policy (unlike standard PG), like DQN
  - Add noise for exploration
- Soft Actor Critic (SAC)
  - DDPG with maximum-entropy regularization
  - seeks to maximize the entropy of the policy (as well as rewards) to stabilise learning

Deep Reinforcement Learning Beyond Games

- Robotics and computer vision
- Energy – adaptive decision control
- Healthcare – diagnosis
- Education – task recommendation
- Transport – traffic signal control
- Finance – trading, portfolio/risk management
- Computer systems – resource management
- Machine learning – hyperparameter tuning

Open Challenges

There remain many open challenges for reinforcement learning.



Montezuma's Revenge – Notoriously Difficult for RL Agents [7]

Consider alternatives such as Imitation Learning.

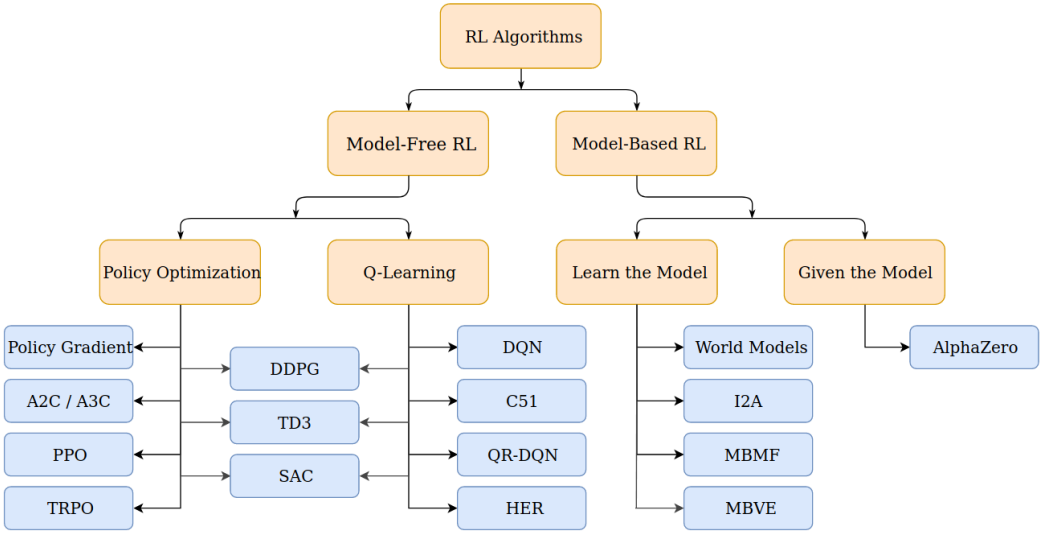
Summary

What we have looked at today:

- Critic: e.g., TD Control – Deep Q-Learning
- Actor: e.g., Deep Policy Gradient – REINFORCE
- Actor-Critic: e.g., A2C, A3C, ...

Value-based vs Policy-based RL?

- Policy optimization methods: we are optimizing directly for what we want (the policy). Often stable and reliable.
- Q-learning (value-based) methods: optimize for agent performance via Q_ϕ. Can fail often/are less stable. But, Q-learning gets more out of each sample; better data reuse.



Modern Machine Learning Methods [8]

Advanced Machine Learning and Autonomous Agents

Reinforcement Learning III

Jesse Read

Version: