Google DeepMind

Deep Reinforcement Learning

Matteo Hessel
Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L
www.decoding-intelligence.com

Google DeepMind

01 - RL Introduction

Matteo Hessel
Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L
www.decoding-intelligence.com

The problem of intelligence

Artificial Intelligence (AI) aims to reproduce, *in software*, the most salient features of *human intelligence*

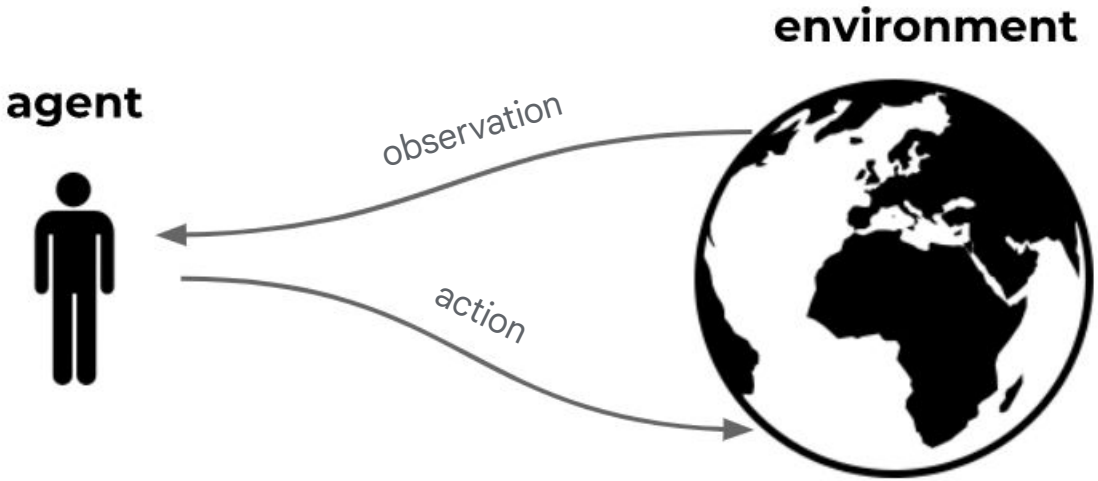
- Learning
- Deductive (logical) reasoning
- Inductive (probabilistic) reasoning
- Planning
- Visual and Language understanding
- Theory of mind

Intelligence is the computational part of the ability to achieve goals.

McCarthy,
What is intelligence?

Google DeepMind

Agent and environment



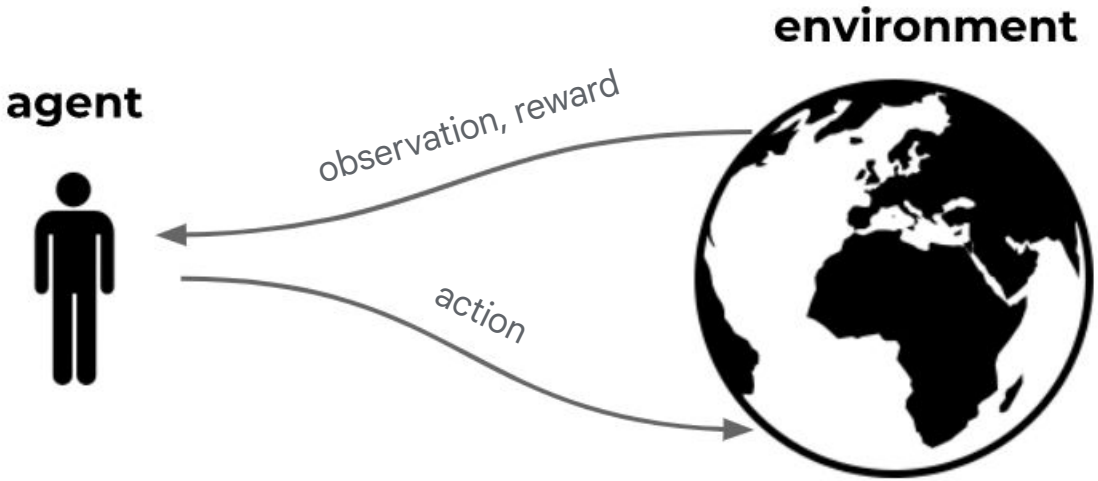
“

All that we mean by goals can be thought of as maximization of the expected value of the cumulative sum of a scalar signal (a reward)

Rich Sutton
John McCarthy’s Definition of Intelligence
The reward hypothesis

Google DeepMind

The RL interface



The RL objective

Let's assume an *observation* O_t and *reward* R_t are provided to the agent on every timestep *t*

The agent's behaviour is represented by a **policy** $\pi(A_t|H_t)$

- a mapping from the history of previous observations $H_t=[O_0, O_1, ..., O_{t-1}]$ onto the next action A_t

The agent's objective is to find an **optimal policy** $\pi^*(A_t|H_t)$

- that maximises the discounted sum of future rewards, or **return**:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

State representation

An environment is said **fully observable**

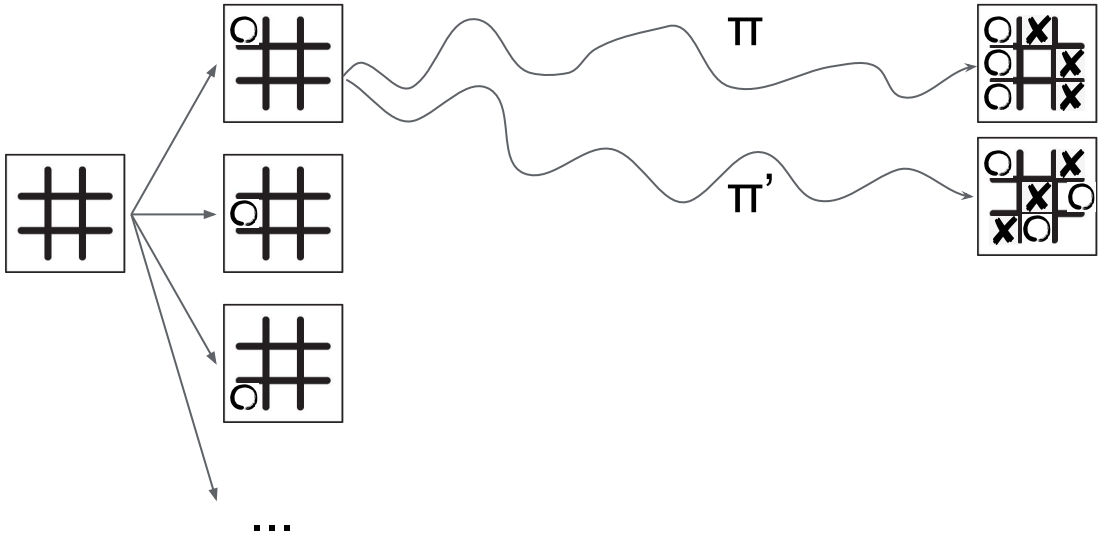
- when the optimal policy $\pi^*(A_t|H_t)$ depends only on the last observation received by the agent .

In general, this is not the case and the agent will need to summarise the past history into a *state* $S_t = f(H_t)$

- In a fully observable environment this is trivial as $S_t = O_t$

This state representation $S_t = f(H_t)$ has to be learned jointly with the policy $\pi(A_t|S_t)$ mapping states to behavior

Credit assignment



Value functions

Value functions estimate the return that you can expect from any state *s* onwards

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s]$$

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

Optimal value functions assume the return is collected under optimal policy
And satisfy the recursive **bellman optimality** equation

$$q^*(S_t, A_t) = E[R_{t+1} + \gamma \max_a q^*(S_{t+1}, a)]$$

Q-learning

Solving the bellman optimality equation solves the entire RL problem
As it's easy to construct an optimal policy from optimal action values:

$$a^* = argmax_a \; q^*(s, a)$$

Temporal difference learning allows us to do so **iteratively** by

- Randomly initialising the agent's estimates of the values
- Updating the values after each interaction with the environment
  so that the bellman equation is satisfied a bit better on the latest data

To reduce the error on the bellman optimality equation,
Q-learning updates the estimates towards a **bootstrap target**

$$Q(S, A) \rightarrow R + \gamma \max_{a'} Q(S', a')$$

Tabular RL is not enough

In tabular RL we learn values/policies/models as **look-up tables**

- Each state (and/or action) value is estimated independently from each other
- This is not scalable
- Many problems have huge state spaces (and/or action spaces)

We need instead to treat values/policies/models as **functions** to be approximated

- Curve fitting
- Generalisation

Google DeepMind

02 - Deep RL

Matteo Hessel
Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L
www.decoding-intelligence.com

Value function approximation

Values are ultimately just **functions**

that map states to scalar estimates of the cumulative reward

$$v : s \rightarrow v(s)$$

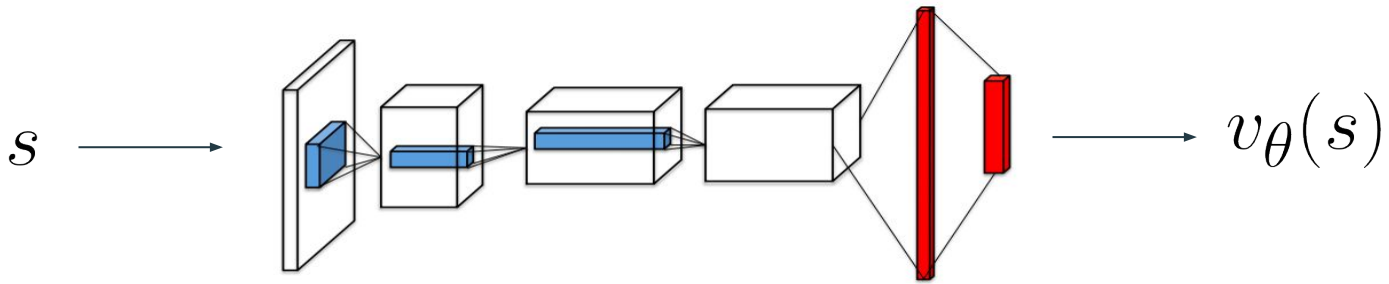
$$q : s, a \rightarrow q(s, a)$$

The problem of values estimation thus reduces to:

- Picking a function class such that values are parametrized by some weights ϑ
- Finding suitable values for these weights ϑ

Value function approximation - II

Typically we will assume that the **function class** is a neural network



Value function approximation - III

Goal: In both cases the problem of learning values becomes an optimisation problem

$$L(\theta) = E_{\pi,d}[(v_\pi(S) - v_\theta(S))^2]$$

Where we want to find the parameters θ that minimise the value prediction error.

Value function approximation - IV

Main tool: the simplest thing is to optimise this via gradient descent

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta) = \alpha E_{\pi,d}[(v_\pi(S) - v_\theta(S))\nabla v_\theta(S))]$$

By taking small steps in the directions of the negative gradient

Value function approximation - V

Problem: evaluating the gradient $E_{\pi,d}[(v_\pi(S) - v_\theta(S))\nabla v_\theta(S))]$ is hard

Solution:

1) sample the expectation by computing the update on the states we encounter

2) approximate the value target by using a suitable bootstrap target from classic RL

Deep Q-learning

We easily obtain the **deep Q-learning** algorithm

by using $G = R + \gamma \max_{a'} \; q_\theta(S', a')$

Resulting in the update $\Delta\theta = \alpha(R + \gamma \max_{a'} \; q_\theta(S',) - q_\theta(S, A))\nabla q_\theta(S, A))$

Pseudo-losses

For consistency with DL notation, you'll see the updates written as gradients of a **pseudo loss**

E.g. Q-learning:

$$L(\theta) = \frac{1}{2}(||R + \gamma \max_{a'} q_\theta(S', a')|| - q_\theta(S, A))^2$$

Where the target is surrounded by a **stop-gradient** sign

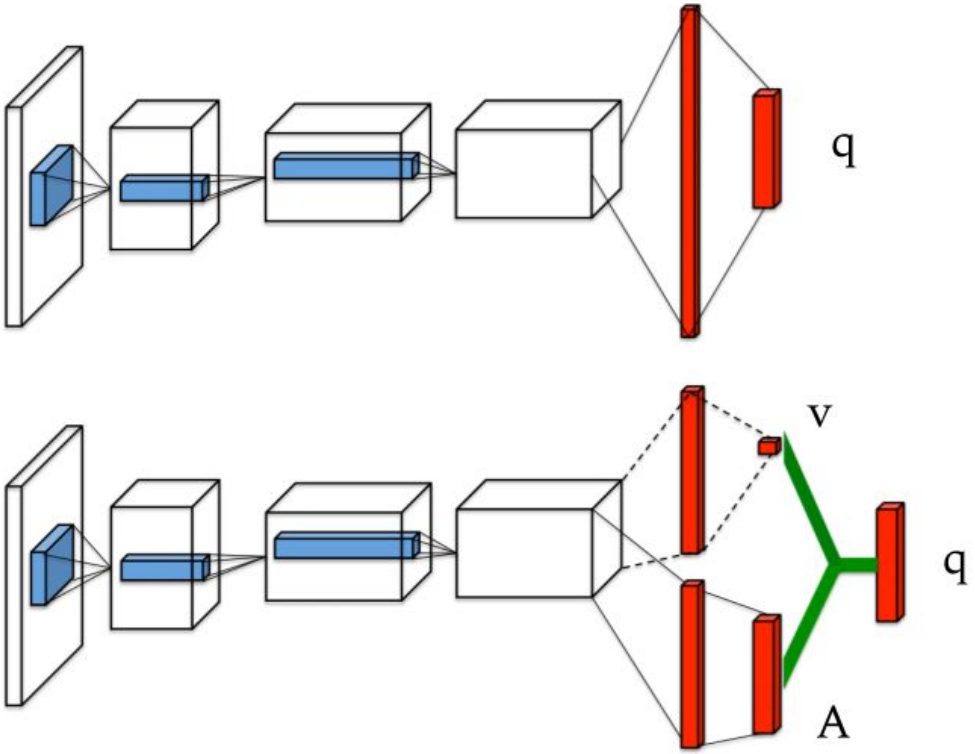
Meaning that we ignore the dependency on θ of the target itself

Choosing the function class: value functions

The difficulty of the optimisation problem often depends on the “function class”

- Deep learning research gives us a lot of smart architectures suitable for different kind of inputs
  - MLPs
  - ConvNets
  - Transformers
- You *can* use any of these out of the box
- Usually however you can do better in DL by having suitable inductive biases in the network
- Can we design RL-aware deep learning architectures?

Choosing the function class: value functions



Google DeepMind

Choosing the function class: state representation

As discussed, state representations also needs to be learned

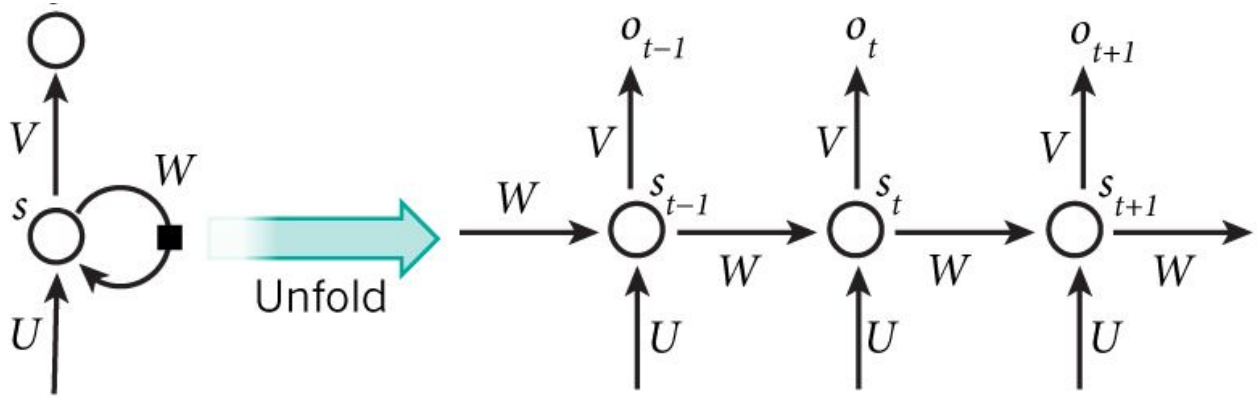
- What is a good function class for these?
- A crucial requirement for a good state representation is to be **incremental**:

  $f : S_t, O_t \rightarrow S_{t+1}$

  So that the cost of computing the state is **constant** during the agent's lifespan

Recurrent state representation - I

This is exactly the functional form of **recurrent neural networks** (RNNs)



Recurrent state representation - II

Deep learning provides many possible concrete instances to choose from

- Long Short Term Memory networks (LSTMs)
- Gated Recurrent Units (GRUs)

Optimisation

Problem: stochastic gradient descent assumes that the data is i.i.d

- In RL data comes from interacting with the environment
- Consecutive updates are strongly correlated

This can cause instabilities

Experience Replay

A popular solution is to store experience in an experience replay buffer

- then compute updates on data sampled uniformly from the buffer
- this allows to break correlations between consecutive updates
- allows to improve data efficiency by allowing to reuse samples more than once

Stability of RL algorithms under function approximation

Introducing function approximation can make classic RL algorithms unstable

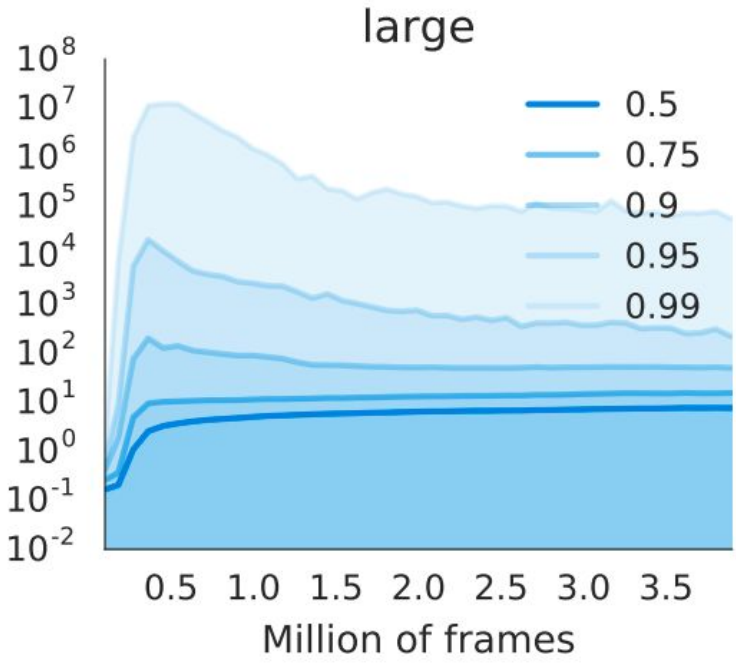
- For instance parameters might **diverge** (to $+\infty$) when using the “Deadly Triad”
  - Bootstrapping
  - Off-policy learning
  - Function approximation

This is for instance the case with deep Q-learning

Soft divergence

In practice we don’t see unbounded divergence.

We see a **soft-divergence** where values grow to unreasonably high values but then converge back to sensible estimates.



Inappropriate generalisation

The reason for the deadly triad is **inappropriate generalization**

- When updating the value of a state $q_\theta(S, A)$
- you might push in the same direction the max value of the next state $\max_{a'} q_\theta(S', a')$
- And therefore the bootstrap target $G = R + \gamma \max_{a'} q_\theta(S', a')$
- Which in turn will push up further the value of the original state $q_\theta(S, A)$

When **on-policy** this self corrects because you immediately see the next state and correct

When **off-policy** this can cause a uncontrolled feedback loop that causes divergence

Target networks

Target Networks are a common remedy to this type of inappropriate generalization

- We can keep two copies of the neural network used to estimate Q values:
- The parameters of one are updated at each step, and is used to act on each step
- The parameters of the other are a slow copy of the first one.
- The “slow” network is then used to construct the bootstrap target

This reduces the opportunity for feedback loops.

N-step Q-learning

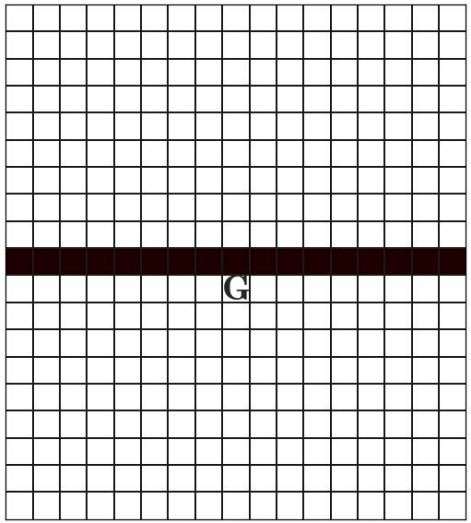
N-step learning also reduces the impact of this generalisation

- Instead of bootstrapping after a single step, you can do so after N, e.g. with N=2

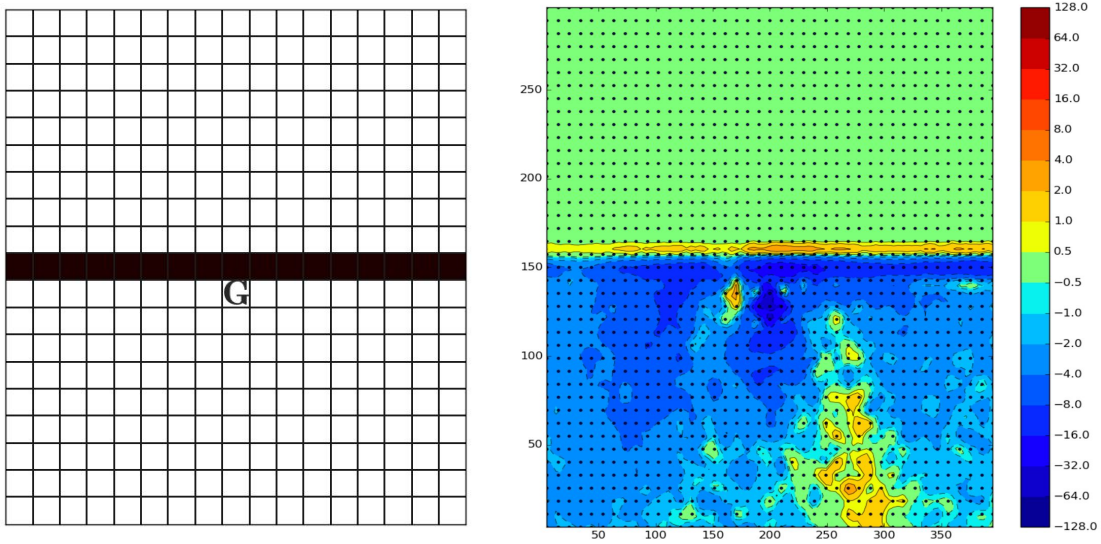
$$Q'(S, A) \rightarrow R + \gamma R' + \gamma^2 \max_a Q(S'', a)$$

- The bootstrap target matters less (due to the gamma^2 discounting)
- State S'' is (usually) more different from S since more stuff has happened in between so updates to the values in S are less likely to change S''
- Also speeds up propagating information through state spacw

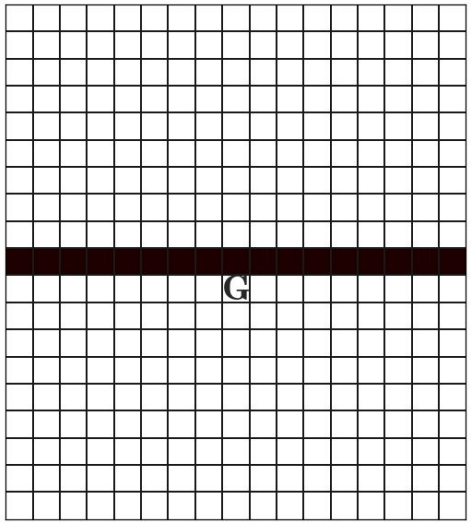
Leakage propagation

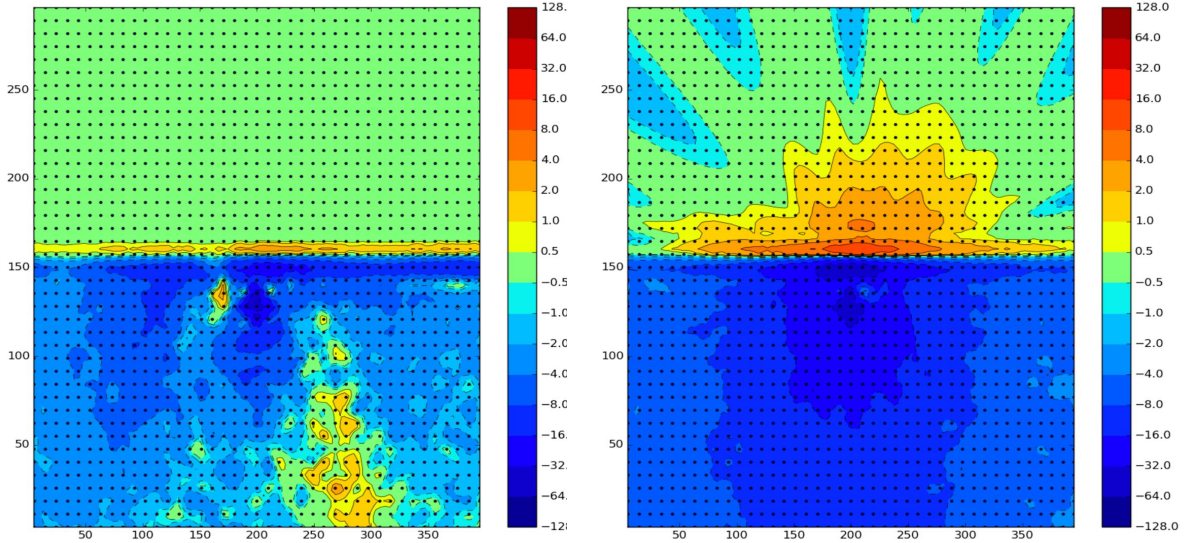


Leakage propagation



Leakage propagation





Google DeepMind

Poverty of the stimulus

The quality of the state representation is **critical**, but

- At the beginning of training when the agent doesn't see many rewards the signal is very limited
  - So it's hard to learn a good representation
- Conversely, without a good representation is hard to learn from the limited signal we see
  - So the behaviour doesn't improve and we continue to see few rewards

How do we help the agent break out of this loop?

Auxiliary tasks

The RL problem provides the agent with 2 things on each step:

- Rewards
- Observations

We should make use of both! For instance, we can share the state representation with **auxiliary tasks** constructed from the stream of observations

- **Reconstruct** the observation
- **Predict** the next (or the previous?) observation
- Construct **pseudo rewards** from the observations that we learn additional values and policies for:
    - Pixel control
    - Feature control
    - Learn at multiple time-scales (e.g. learn values under different discount factors)

Scaling up

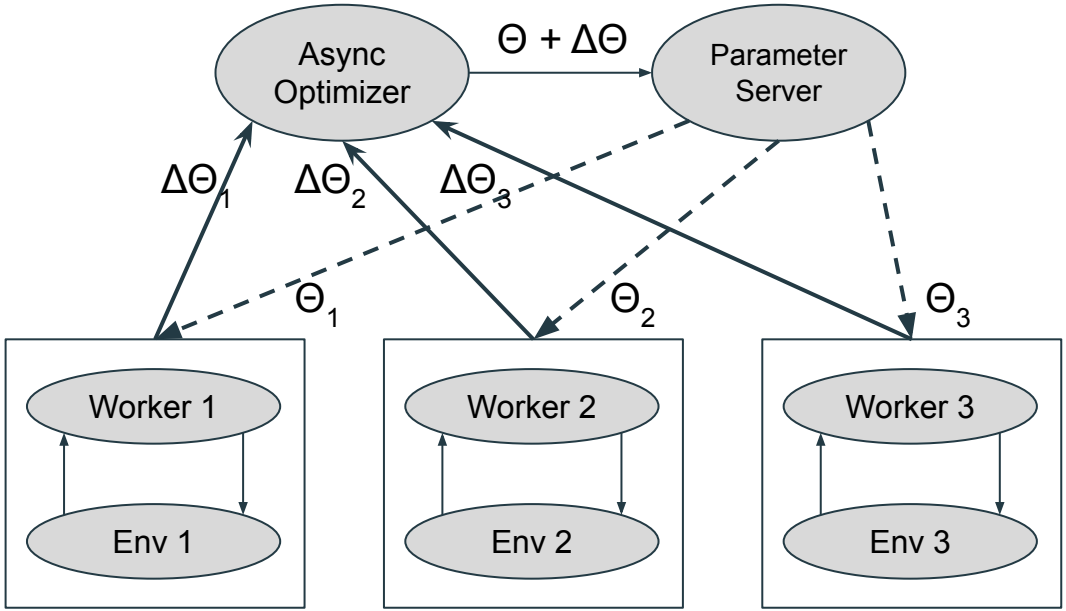
The basic agent-environment interaction loop is inherently sequential

Much of the power of Deep Learning comes from the fact that it's highly scalable

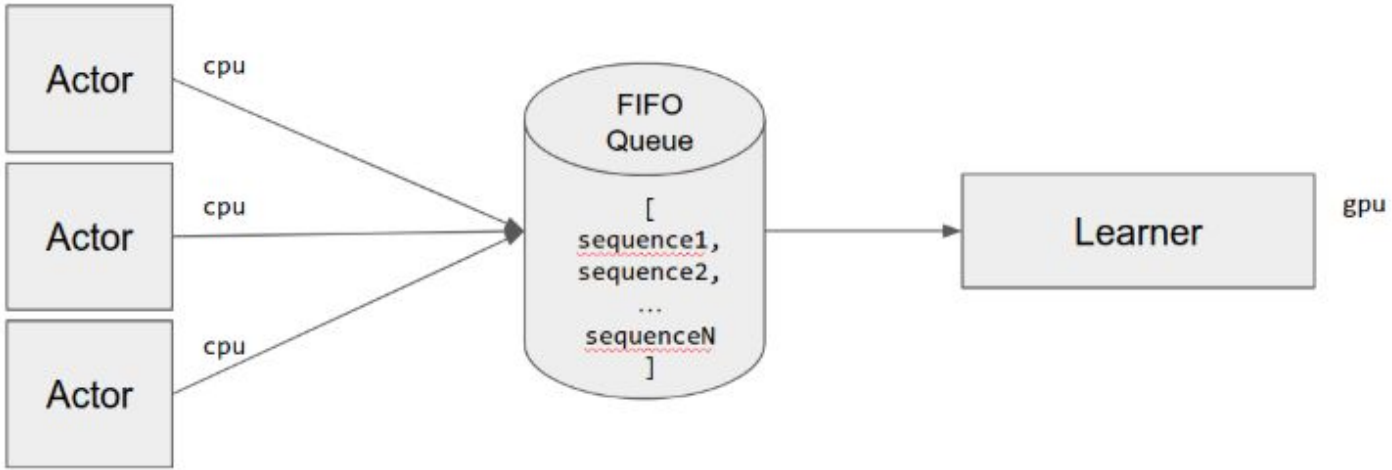
- By allowing to process large batches of data in parallel

How can we scale up reinforcement learning?

Async RL



Actor-learner decomposition



Google DeepMind

Google DeepMind

03 - Deep RL Applications

Matteo Hessel
Research Engineer @ Google DeepMind
Honorary Lecturer @ UCL
Organizer @ M2L
www.decoding-intelligence.com

Deep RL & Nuclear Fusion

Nuclear fusion promises a potential future of unlimited clean energy without radioactive waste.

Requires unprecedented technical challenges

- Contain plasma at 100s of millions degrees Celsius using powerful super-magnets

DeepMind and EPFL showed that an RL algorithm could learn in a simulator to control the plasma

And the learned policy transferred successfully to controlling an **actual small scale fusion reactor**

Google DeepMind

Deep RL & Inventory Management

Inventory management is the process by which a company chooses how much to stock in its warehouses of each product it sells.

- If you don't stock enough you can't provide the product to your clients in a timely manner
- If you stock too much you incur an expense due to unsold items taking space in the warehouses

Amazon showed that they could to train an RL agent to choose when to order products from the suppliers so as to maximise availability and minimising the amount of product held in stock at any one time.

Google DeepMind

Deep RL & LLM Alignment

Large **language models** are trained to predict the next token on huge amounts of text data collected from books and from the internet.

Out of the box these models often

- struggle with understanding human intentions and holding conversations
- they are prone to outputting toxic content (e.g. sexist, racist)

RLHF has emerged as the leading way to fine tune LLMs to be more suitable for deployment

- The model is asked to generate a dataset of possible utterances/dialogs
- Human raters are asked to pick the preferred response
- A reward model is trained to be consistent with these preferences

A standard RL algorithm (e.g. PPO) is used to optimize the LLM to maximise this inferred reward

The generality of Deep RL

What do all these problems have in common?

- Almost nothing
- Except that the agent
  - Must make a sequence of decision over time
  - So as to maximise some feedback of how well its doing

But this is just the very general definition of the **RL problem**

- We can use the same deep RL algorithms to solve all of these problems!

The bitter lesson of AI

Of course in some cases domain specific algorithms can be more efficient in the short term

But focusing on general problems and solutions has the biggest impact long term

Try to avoid repeating the **bitter lesson** of AI:

1. AI researchers always try to build knowledge into their agents
2. This always helps in the short term
3. In the long run leads to diminishing returns and may even inhibits further progress
4. breakthrough eventually arrives by an approach based on learning and search.

Thank you.

Google DeepMind

Google DeepMind