DEEP REINFORCEMENT LEARNING

RL FOR THE MODERN AGE….

BACKGROUND INFORMATION….

WHY DEEP RL?

- You can frame many problems in terms of a discrete set of States, however almost all “real-world” data is continuous.
- We therefore need to use some continuous function in order to learn from continuous data
- Neural Network are learnable continuous function!

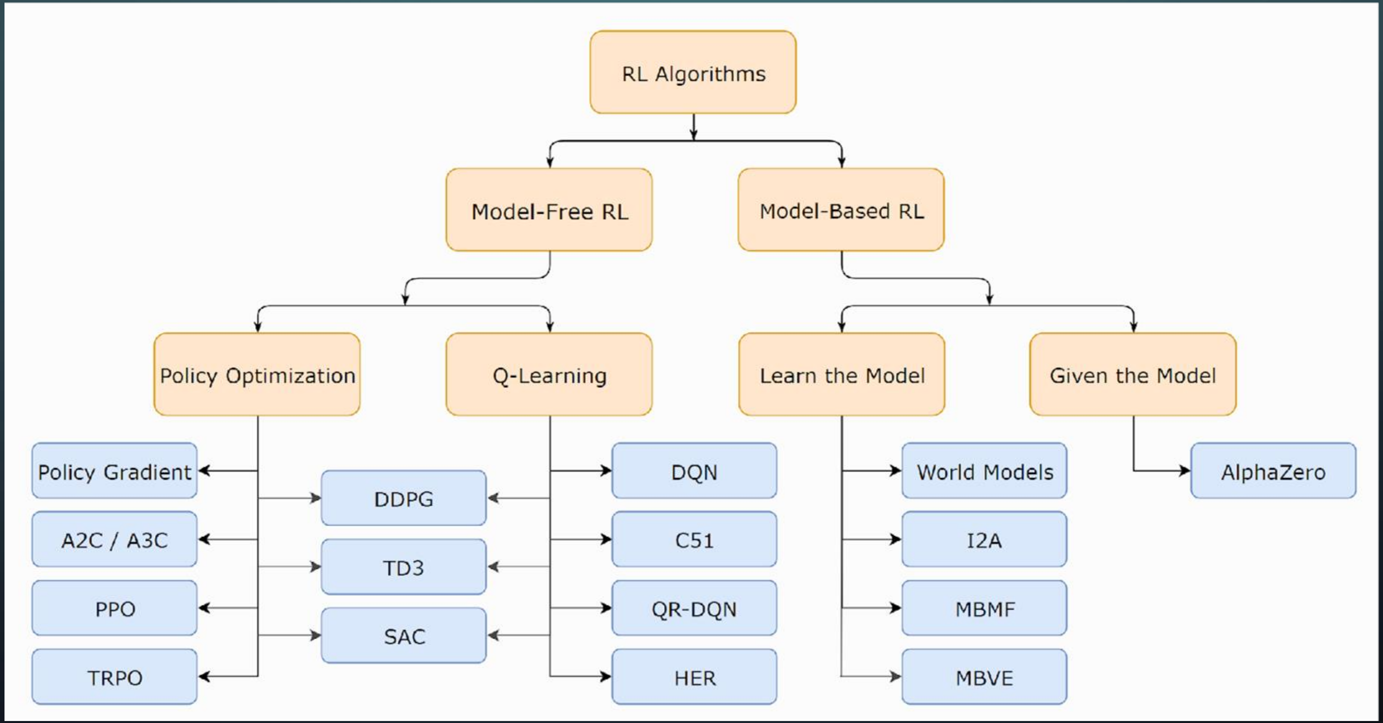
PROBLEMS WITH USING NEURAL NETS….

- RL the training signal is often very sparse
  - Rewards are often only given once, or far apart during a rollout
- Training signals are often dependant of each other
  - You only get some rewards after completing an initial step which gets a reward.
    - In a video game, to get to the second enemy you need to kill the first enemy first, if you never kill the first enemy you never see the second.
- The training data distribution is non-stationary
  - As the agent gets better it is likely to progress further into a task, getting itself into previously unseen States.
    - To get to the second level in a video game, you need to complete the first level!

PROBLEMS WITH USING NEURAL NETS….

- Unlike a Look-Up table, you can’t change the stored variable for a single State only, Neural Networks are parameterised equations, if you change the values of the parameters you change the outputs for ALL States!
- Neural Networks require a LOT of data to train, and can easily over-fit to a small dataset!

CURRENT LANDSCAPE OF DEEP RL



DEEP Q NETWORKS (DQN)

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

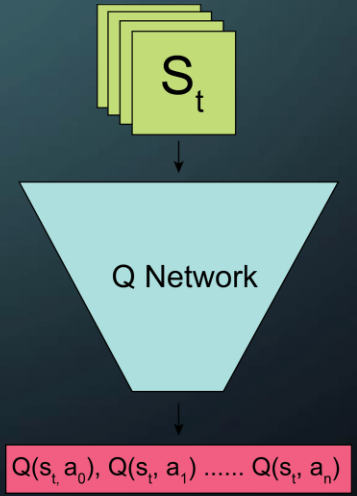
Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The

DEEP Q NETWORKS (DQN)

- One of the first instances of creating a robust and “general” Deep RL algorithm
- The “Q-Value” table is replaced with a Convolutional “Q Network”

Q(a, s)	a_0	a_1	a_2	.....	a_n
s_0					
s_1					
s_2					
s_3					
s_4					
⋮					
s_m					



CHALLENGES WITH DEEP Q NETWORKS (DQN)

To train a “Deep Q Network” we need to turn the Q-Value update into a loss function:

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s,a;\theta_i\right)\right)^2\right]$$

$$\text{where } y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

We can see that target y_i is the just the Q-Value update and the total loss is MSE between the current Q-Value and the update.

Here we also notice a bit of a problem! Our “target” for the Q-Value, is constructed from the output of our Q-Network! This can lead to instabilities.

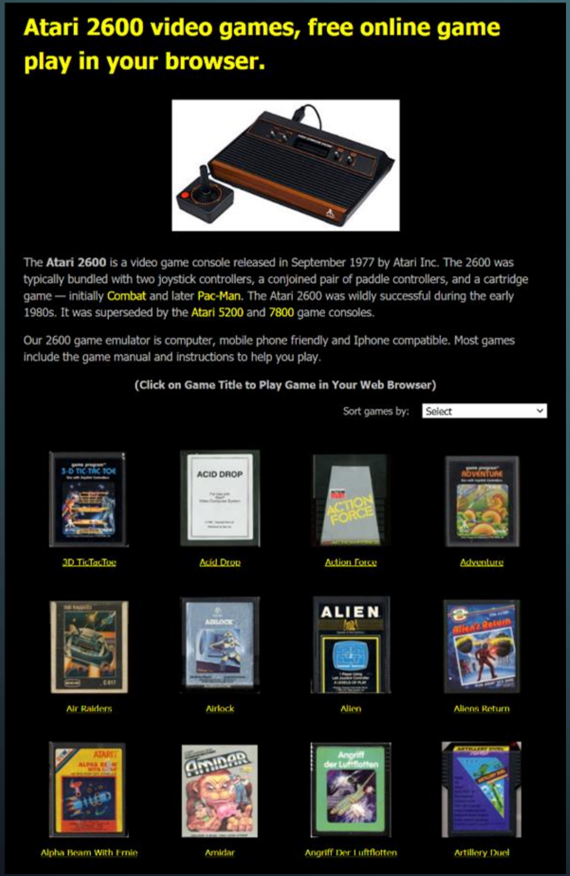
DEEP Q NETWORKS (DQN) - Solutions

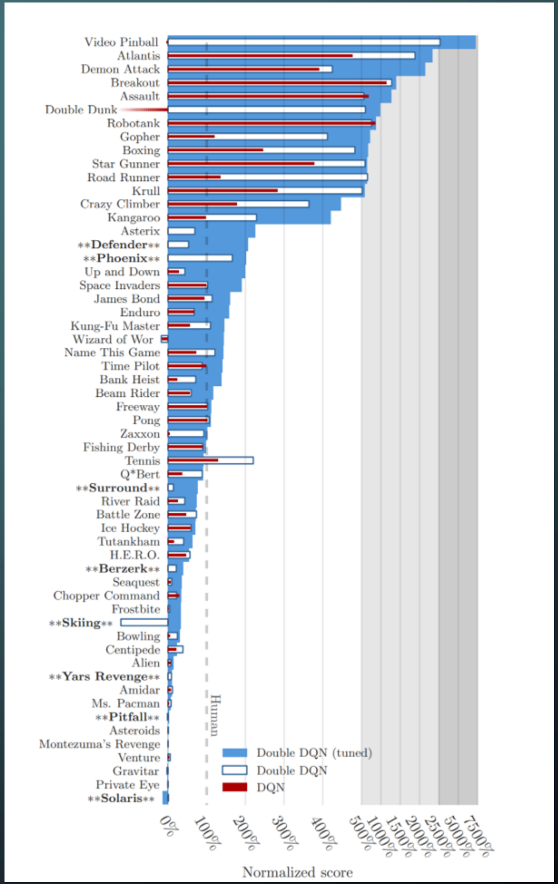
- Non-Stationary/Noisy data and dependant training signals
  - Experience Replay Buffer
    - A FIFO buffer that collects and holds each “transition” experience (s_t, a_t, r_t, s_{t+1})
    - ~10,000 - 1,000,000 transitions
    - At training time we perform batched gradient descent over the average loss
- Sparse Rewards/Overfitting
  - A relatively small network

DEEP Q NETWORKS (DQN)

Algorithm 1 Deep Q-learning with Experience Replay

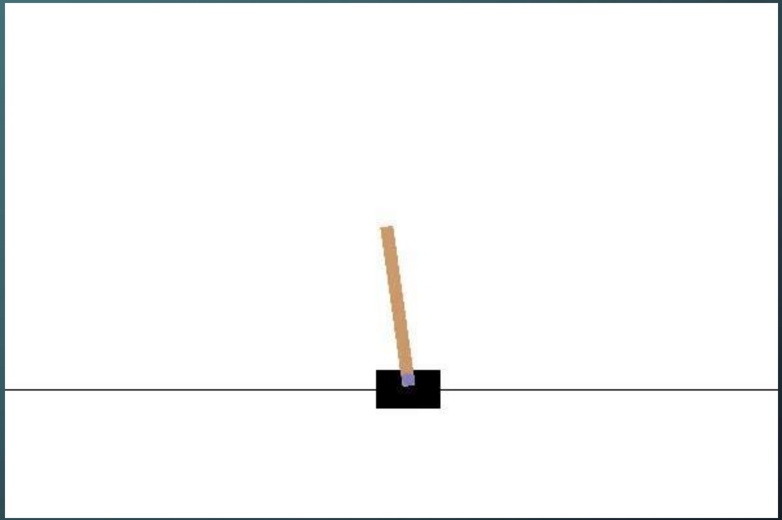
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise 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


Atari 2600 video games, free online game play in your browser.
The Atari 2600 is a video game console released in September 1977 by Atari Inc. The 2600 was typically bundled with two joystick controllers, a conjoined pair of paddle controllers, and a cartridge game — initially Combat and later Pac-Man. The Atari 2600 was wildly successful during the early 1980s. It was superseded by the Atari 5200 and 7800 game consoles.
Our 2600 game emulator is computer, mobile phone friendly and Iphone compatible. Most games include the game manual and instructions to help you play.
(Click on Game Title to Play Game in Your Web Browser)
Sort games by:
Select
3D TicTacToe
Acid Drop
Action Force
Adventure
Air Raiders
Airlock
Alien
Aliens Return
Alpha Beam With Ernie
Amidar
Angriff Der Luftflotten
Artillery Duel


Video Pinball
Atlantis
Demon Attack
Breakout
Assault
Double Dunk
Robotank
Gopher
Boxing
Star Gunner
Road Runner
Krull
Crazy Climber
Kangaroo
Asterix
Defender
Phoenix
Up and Down
Space Invaders
James Bond
Enduro
Kung-Fu Master
Wizard of Wor
Name This Game
Time Pilot
Bank Heist
Beam Rider
Freeway
Pong
Zaxxon
Fishing Derby
Tennis
Q*Bert
Surround
River Raid
Battle Zone
Ice Hockey
Tutankham
H.E.R.O.
Berzerk
Seaquest
Chopper Command
Frostbite
Skiing
Bowling
Centipede
Alien
Yars Revenge
Amidar
Ms. Pacman
Pitfall
Asteroids
Montezuma's Revenge
Venture
Gravitar
Private Eye
Solaris
Human
Double DQN (tuned)
Double DQN
DQN
0%
100%
200%
300%
400%
500%
1000%
1500%
2000%
2500%
5000%
7500%
Normalized score

CODE EXAMPLE

CARTPOLE



Deep Reinforcement Learning with Double Q-learning

Hado van Hasselt and **Arthur Guez** and **David Silver**
Google DeepMind

Abstract

The popular Q-learning algorithm is known to overestimate action values under certain conditions. It was not previously known whether, in practice, such overestimations are common, whether they harm performance, and whether they can generally be prevented. In this paper, we answer all these

exploration technique (Kaelbling et al., 1996). If, however, the overestimations are not uniform and not concentrated at states about which we wish to learn more, then they might negatively affect the qual
and Schwartz (1993) give
leads to suboptimal polici

Dueling Network Architectures for Deep Reinforcement Learning

Ziyu Wang ZIYU@GOOGLE.COM
Tom Schaul SCHAUL@GOOGLE.COM
Matteo Hessel MTTHSS@GOOGLE.COM
Hado van Hasselt HADO@GOOGLE.COM
Marc Lanctot LANCTOT@GOOGLE.COM
Nando de Freitas NANDODEFREITAS@GMAIL.COM

cesses
ement
ıs use

In spite of this, most of the approaches for RL use standard neural networks, such as convolutional networks, MLPs, LSTMs and autoencoders. The focus in these recent advances has been on designing improved control and RL algorithms, or simply on incorporating existing neural net-

Hindsight Experience Replay

Marcin Andrychowicz[*], **Filip Wolski**, **Alex Ray**, **Jonas Schneider**, **Rachel Fong**, **Peter Welinder**, **Bob McGrew**, **Josh Tobin**, **Pieter Abbeel**[†], **Wojciech Zaremba**[†]
OpenAI

Abstract

th sparse rewards is one of the biggest challenges in Reinforcement
RL). We present a novel technique called *Hindsight Ex*
vs sample-efficient learning from rewards which are s
re avoid the need for complicated reward engineering
an arbitrary off-policy RL algorithm and may be se

Rainbow: Combining Improvements in Deep Reinforcement Learning

Matteo Hessel DeepMind
Joseph Modayil DeepMind
Hado van Hasselt DeepMind
Tom Schaul DeepMind
Georg Ostrovski DeepMind
Will Dabney DeepMind
Dan Horgan DeepMind
Bilal Piot DeepMind
Mohammad Azar DeepMind
David Silver DeepMind

Abstract

The deep reinforcement learning community has made several independent improvements to the DQN algorithm. However, it is unclear which of these extensions are complementary and can be fruitfully combined. This paper examines six extensions to the DQN algorithm and empirically studies their combination. Our experiments show that the combination provides state-of-the-art performance on the Atari 2600 benchmark, both in terms of data efficiency and final performance. We also provide results from a detailed ablation study that shows the contribution of each component to overall per-

Distributional Reinforcement Learning with Quantile Regression

Will Dabney DeepMind
Mark Rowland University of Cambridge*
Marc G. Bellemare Google Brain
Rémi Munos DeepMind

Abstract

In reinforcement learning an agent interacts with the environment by taking actions and observing the next state and reward. When sampled probabilistically, these state transitions, rewards, and actions can all induce randomness in the ob-

the-art on the suite of benchmark Atari 2600 games (Bellemare et al. 2013).

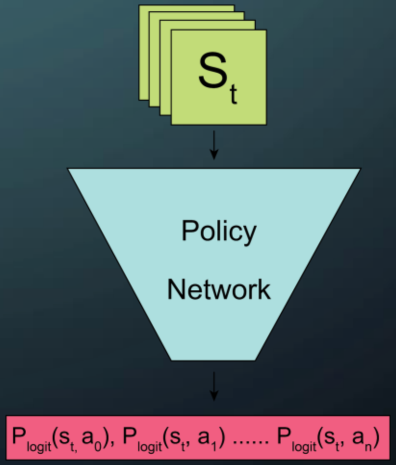
One of the theoretical contributions of the C51 work was a proof that the distributional Bellman operator is a contraction in a maximal form of the Wasserstein metric between probability distributions. In this context, the Wasserstein

DEEP POLICY METHODS - REINFORCE

Using some of the lessons from Deep Value-Based methods, we can start to implement some of the Policy methods as well!

Unlike DQN's we don't need a replay buffer as policy gradient methods are not as sensitive to non-stationary data (we don't learn a Q-Value).

$P(a, s)$	a_0	a_1	a_2	.....	a_n	$\Sigma_i^n P(a_i, S)$
s_0	$P(a_0, s_0)$	$P(a_1, s_0)$	$P(a_2, s_0)$		$P(a_n, s_0)$	1
s_1						
s_2						
s_3						
s_4						
⋮						
s_m						



DEEP POLICY METHODS - REINFORCE

We can see the REINFORCE loss function

$$L_{\theta}^{PG} = \mathbb{E}_{\pi}\left[log(\pi_{\theta}(s_t, a_t))R_t\right]$$

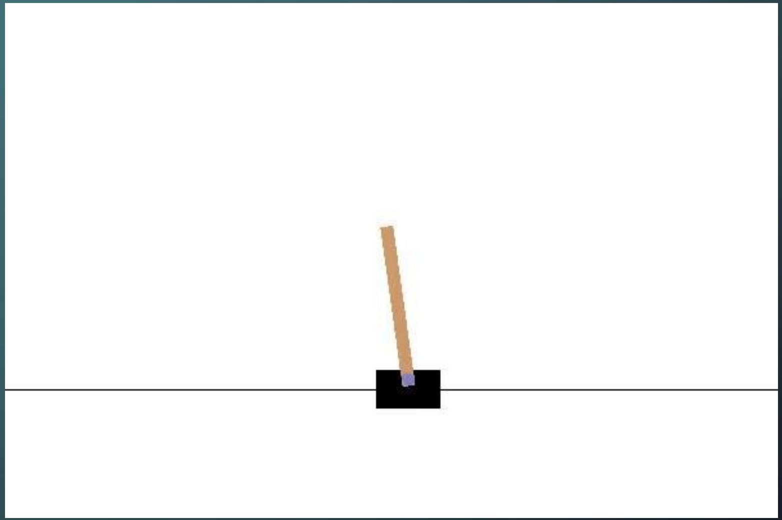
An important difference here is that we want to MAXIMISE this value!

We want to INCREASE the log-probability if the Returns are positive etc...

Note the loss is the expectation over the current policy, so we do not use examples/trajectories from other policies (we can use a replay buffer but all the experiences need to be from the same policy- aka no update).

CODE EXAMPLE

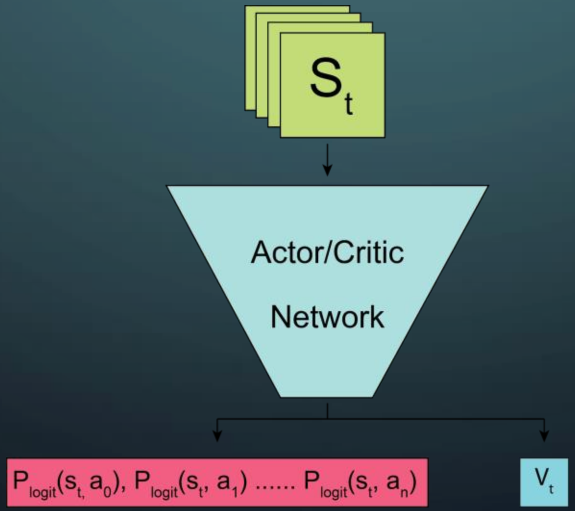
[CARTPOLE]
[OPENAI GYM]



ACTOR-CRITIC METHODS

Similar to simple policy gradient methods except we now have both an Actor (Policy) and a Critic (Value) network.

Most of the time we use a single, combined Actor/Critic Network.



ACTOR-CRITIC POLICY GRADIENTS

The loss for the Policy is similar to before, except we now calculate the Advantage.

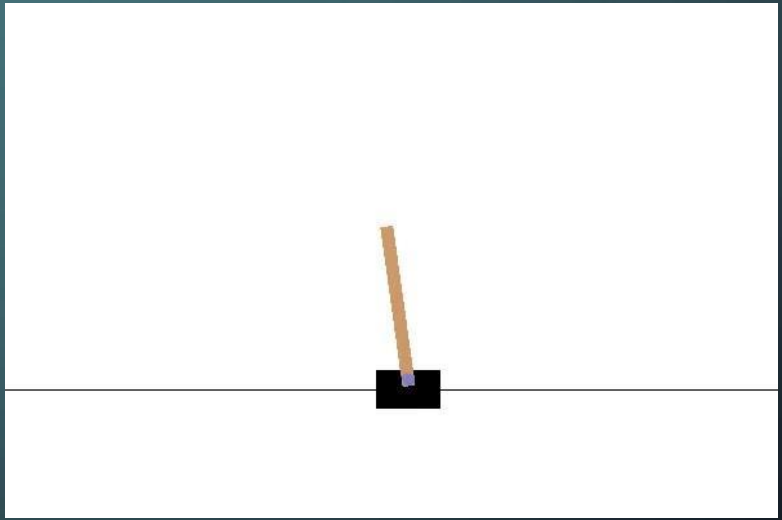
$$A_t = R_t - V_t$$

$$L_\theta^{PG} = \mathbb{E}_\pi \left[log(\pi_\theta(s_t, a_t)) A_t \right]$$

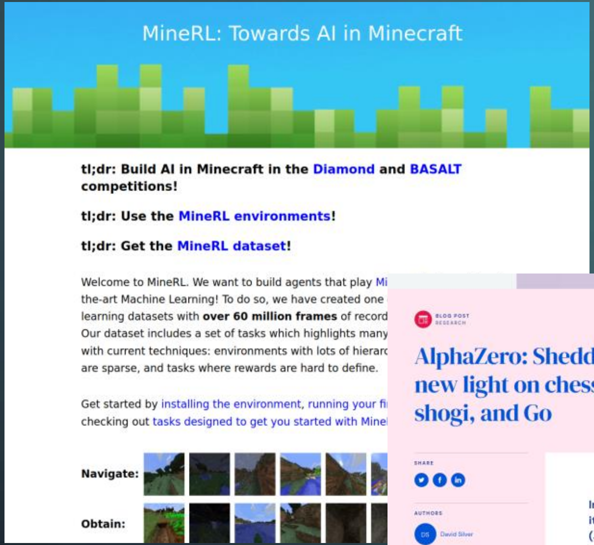
We also need to update the Value function, we can do this with either Monte Carlo or TD updates.

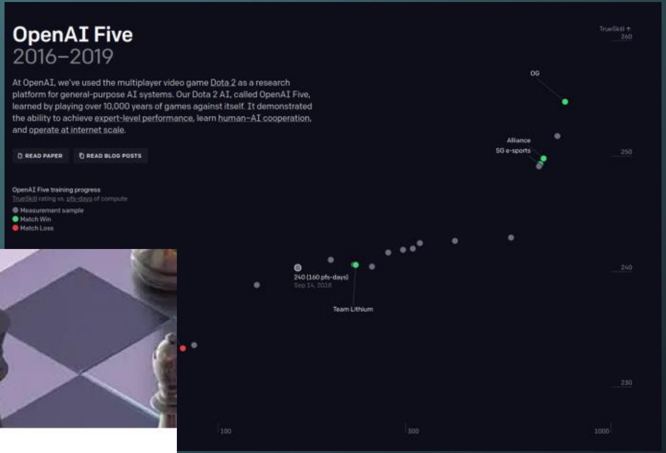
CODE EXAMPLE

CARTPOLE

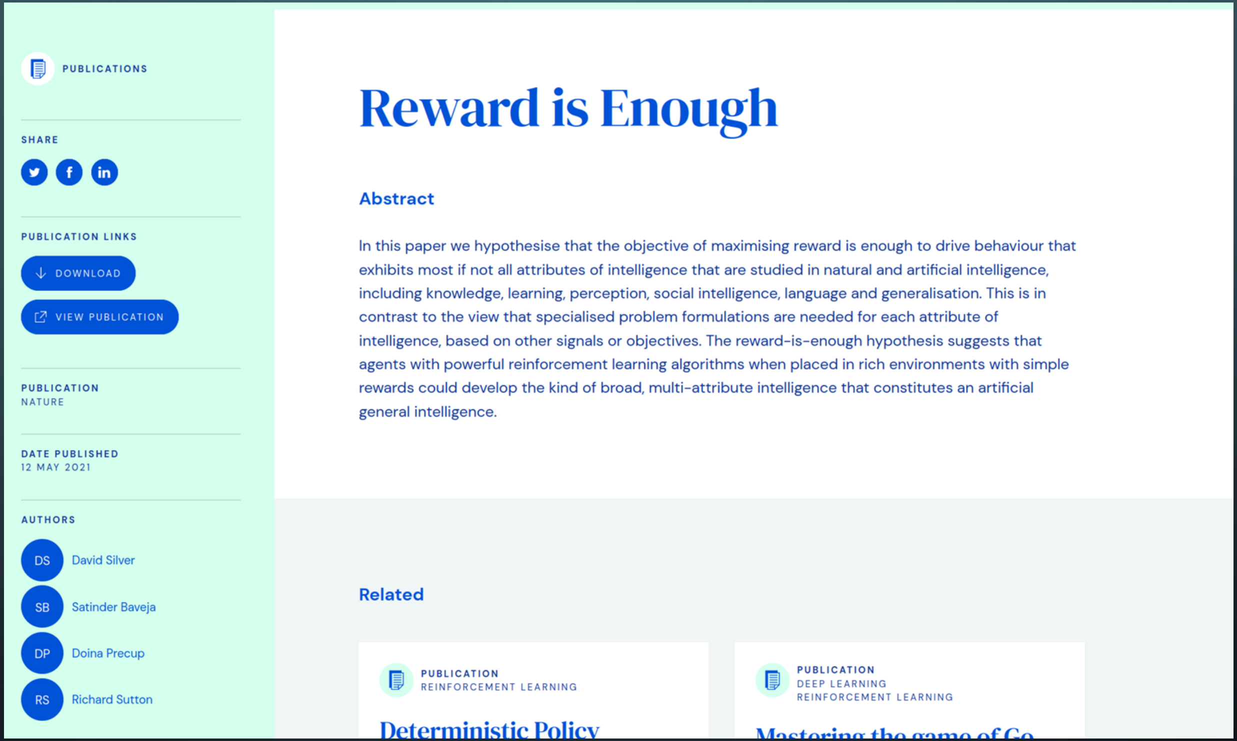


RL IN THE WILDE!





CAN RL DO IT ALL?



PUBLICATIONS

SHARE

PUBLICATION LINKS

DOWNLOAD

VIEW PUBLICATION

PUBLICATION
NATURE

DATE PUBLISHED
12 MAY 2021

AUTHORS

DS David Silver

SB Satinder Baveja

DP Doina Precup

RS Richard Sutton

Reward is Enough

Abstract

In this paper we hypothesise that the objective of maximising reward is enough to drive behaviour that exhibits most if not all attributes of intelligence that are studied in natural and artificial intelligence, including knowledge, learning, perception, social intelligence, language and generalisation. This is in contrast to the view that specialised problem formulations are needed for each attribute of intelligence, based on other signals or objectives. The reward-is-enough hypothesis suggests that agents with powerful reinforcement learning algorithms when placed in rich environments with simple rewards could develop the kind of broad, multi-attribute intelligence that constitutes an artificial general intelligence.

Related

PUBLICATION
REINFORCEMENT LEARNING

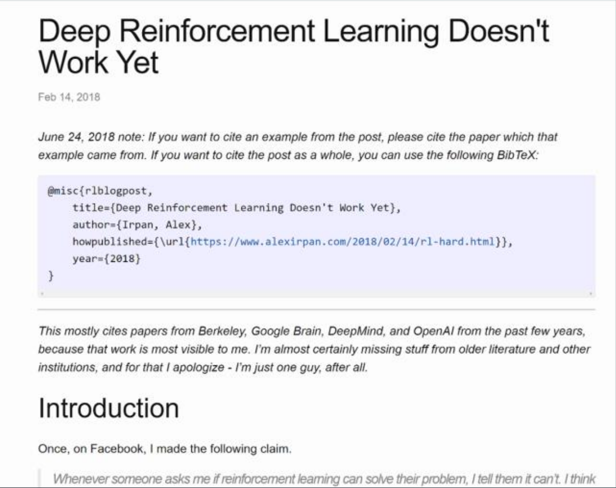
Deterministic Policy

PUBLICATION
DEEP LEARNING
REINFORCEMENT LEARNING

Mastering the game of Go

PROBLEMS WITH DEEP RL…

Deep RL does sound like the perfect solution for any problem! However there are some big obstacles getting in the way of mass adoption….



Deep Reinforcement Learning Doesn't Work Yet

Feb 14, 2018

June 24, 2018 note: If you want to cite an example from the post, please cite the paper which that example came from. If you want to cite the post as a whole, you can use the following BibTeX:

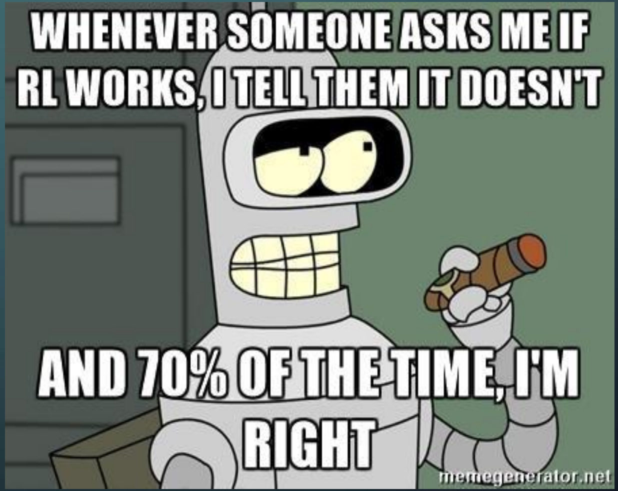
```
@misc{rlblogpost,
    title={Deep Reinforcement Learning Doesn't Work Yet},
    author={Irpan, Alex},
    howpublished={\url{https://www.alexirpan.com/2018/02/14/rl-hard.html}},
    year={2018}
}
```

This mostly cites papers from Berkeley, Google Brain, DeepMind, and OpenAI from the past few years, because that work is most visible to me. I'm almost certainly missing stuff from older literature and other institutions, and for that I apologize - I'm just one guy, after all.

Introduction

Once, on Facebook, I made the following claim.

> *Whenever someone asks me if reinforcement learning can solve their problem, I tell them it can't. I think*



[Deep Reinforcement Learning Doesn't Work Yet](https://www.alexirpan.com/2018/02/14/rl-hard.html)

TRAINING TIME

One of the most common problems is the time it takes to train a Deep RL agent.
For example the "OpenAI Five" and Dota2 playing agent was train with over 10,000 years of game experience!!

Now imagine trying to train a robot in the real world….
You could try to train in a super fast simulation and transfer onto a real robot, however there is another problem….

GENERALIZATION

You may have noticed one big problem with the way we train and evaluate our Deep RL agents. We train them in some environment, and then test them….. in the same environment!

A lot of recent research has shown that when testing a Deep RL agent in a slightly different environment on the same task, performance is very poor. To get a high performing and generalizable agent you not only need to provide huge amounts of data but huge amounts of variation.

[Procgen]()

OTHER PROBLEMS...

- RL can be very sensitive to:
  - Random initialization
  - Hyperparameters
  - Environment experiences
- Training can be unstable and agents can completely “collapse”.
  - Usually, to assess the performance of an Deep RL algorithm, many agents are trained and their performance averaged together.
- Infrastructure for training is usually more complicated than the actual RL algorithm.
  - This is why you see a lot of work in the this area by large research groups who can afford to turn a game/simulation into a training bed for RL.

OTHER PROBLEMS...

- Hard to define a good Reward signal
    - RL will do whatever is easiest to get the reward
        - “Misaligned” agents
        - Reward hacking

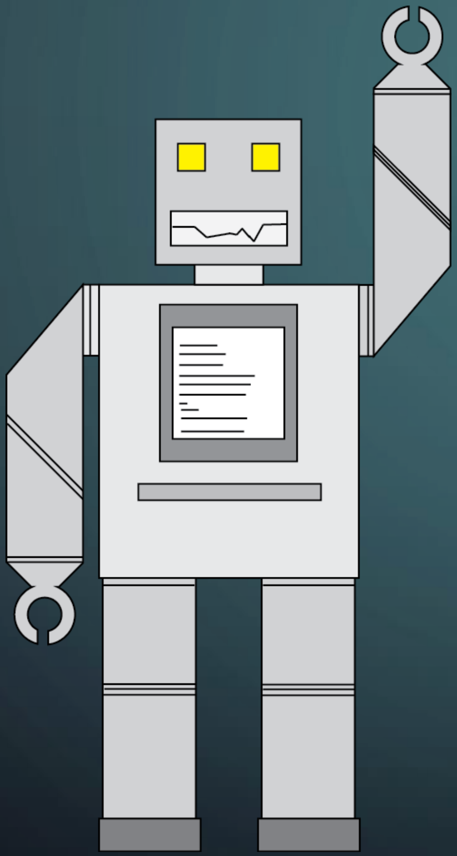
AI SAFETY?

How do we safely use Deep RL and AI in general, without knowing exactly how it will behave?

- The field of AI safety tries to come up with potential ways in which AI systems can:
    - Avoid doing what we want
    - Change its own goals
    - Become misaligned

And hopefully solve them.

[AI Safety Gridworlds]



THANK YOU!