Where is modern RL at today

Success of deep RL = in domains that aren't similar to the real world (Go is structured, with few unexpected situations)

Sample inefficient -> requires simulation

Why is everyone not using q learning

- not easy to get convergence - q learning with nonlinear function approximation is not guaranteed to converge either in theory or practice
- doesn't in general converge
- sensitive to hyper parameters
- lots of local optima

rl tasks = small, uniform, goal=master, test identical to train

real world = huge, highly varied, goal=dont screw up too much, test set = unmitigated disaster of complexity

Open AI Dota

TODO

(https://www.rockpapershotgun.com/2018/07/20/ai-wizard-mike-cook-wants-openais-dota-bots-to-teach-him-not-beat-him/)

(http://www.gamesbyangelina.org/2018/06/good-luck-have-fun/)]

(https://s3-us-west-2.amazonaws.com/openai-assets/dota_benchmark_results/network_diagram_08_06_2018.pdf)

World models

TODO

Model based RL

TODO

Deep RL doesn't work yet

Blog post - Deep Reinforcment Learning Doesn't Work Yet

State of the art reinforcement learning is **sample inefficient** - we need lots of experience to learn

Tackling any problem where we don't have access to a simulator remain beyond modern RL

Domain specific algorithms often work faster & better. This is especially true if you have access to a good environment model to plan with

Requirement of a reward function - or the requirement to design one

Results can be unstable and hard to produce (this applies to a lot of scientific literature). Different random seeds can lead to dramatically different results

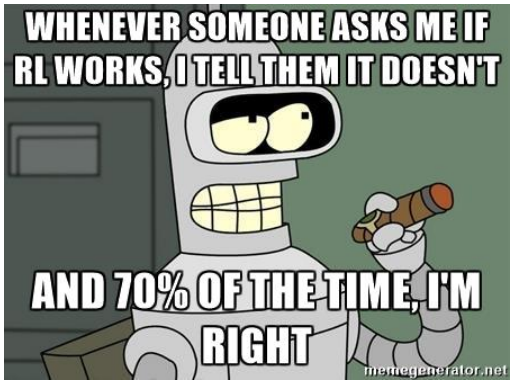

Figure 1: fig

> [Supervised learning] wants to work. Even if you screw something up you'll usually get something non-random back. RL must be forced to work. If you screw something up or don't tune something well enough you're exceedingly likely to get a policy that is even worse than random. And even if it's all well tuned you'll get a bad policy 30% of the time, just because - Andrej Karpathy (when he was at OpenAI)

Still immature in real world production systems - examples are rare

Requirements and/or nice to haves for learning

- easy to generate experience
- simple problem
- ability to introduce self play
- well defined rewards and dense

RL solution doesn't have to achieve a global optima, as long as its local optima is better than the human baseline

Modern RL is sample inefficient

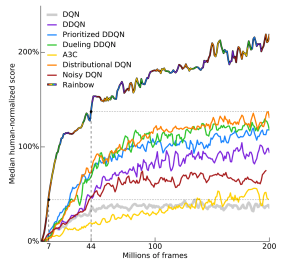

Figure 1: **Median human-normalized performance** across 57 Atari games. We compare our integrated agent (rainbow-colored) to DQN (grey) and six published baselines. Note that we match DQN's best performance after 7M frames, surpass any baseline within 44M frames, and reach substantially improved final performance. Curves are smoothed with a moving average over 5 points.

Figure 2: fig

To pass the 100% median performance

- Rainbow = 18 million frames = 83 hours of play
- Distributional DQN = 70 million
- DQN = never (even after 200 million frames!)

We can ignore sample efficiency if sampling is cheap

In the real world it can be hard or expensive to generate experience

It's not about learning time - it's about the ability to sample

Other methods often work better

Many problems are better solved by other methods

- allowing the agent access to a ground truth model (i.e. simulator)
- model based RL with a perfect model

In a similar vein, you can easily outperform DQN in Atari with off-the-shelf Monte Carlo Tree Search. Here are baseline numbers from Guo et al, NIPS 2014. They compare the scores of a trained DQN to the scores of a UCT agent (where UCT is the standard version of MCTS used today.)

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
DQN	4092	168	470	20	1952	1705	581
-best	5184	225	661	21	4500	1740	1075

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
UCT	7233	406	788	21	18850	3257	2354

Figure 3: fig

The generalizability of RL means that except in rare cases, domain specific algorithms work faster and better

Requirement of a reward function

Reward function design is difficult - need to encourage behaviour - need to be learnable

Shaping rewards to help learning can change behaviour

Unstable and hard to reproduce results

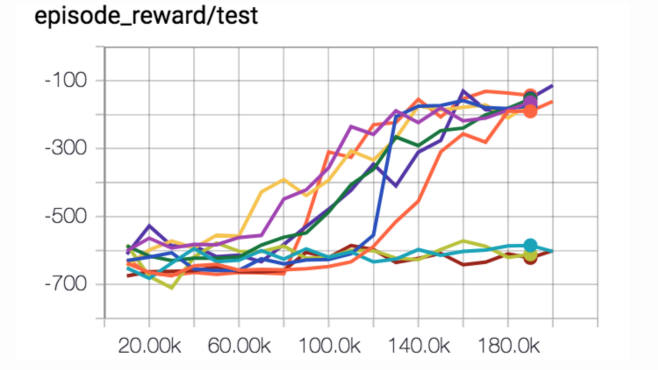

Figure 4: fig

Only difference is the random seed!

30% failure rate counts as working

Machine learning adds more dimensions to your space of failure cases

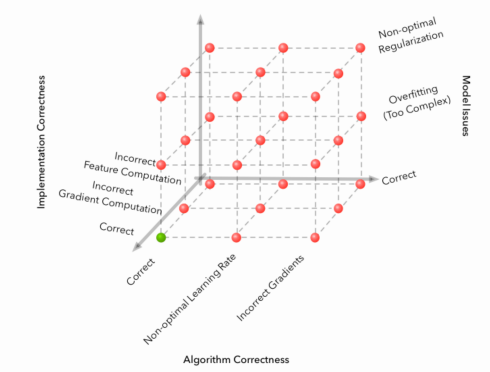

Figure 5: fig

RL adds an additional dimension - random change

A sample inefficient and unstable training algorithm heavily slows down your rate of productive research

> *[Supervised learning] wants to work. Even if you screw something up you'll usually get something non-random back. RL must be forced to work. If you screw something up or don't tune something well enough you're exceedingly likely to get a policy that is even worse than random. And even if it's all well tuned you'll get a bad policy 30% of the time, just because.*
>
> *Long story short your failure is more due to the difficulty of deep RL, and much less due to the difficulty of "designing neural networks".*
>
> Hacker News comment from Andrej Karpathy, back when he was at OpenAI

Figure 6: fig

Going forward & the future

> The way I see it, either deep RL is still a research topic that isn't robust enough for widespread use, or it's usable and the people who've gotten it to work aren't publicizing it. I think the former is more likely.

Figure 7: fig

Make learning eaiser

- ability to generate near unbounded amounts of experience
- problem is simplified into an eaiser form
- you can introduce self-play into learning
- learnable reward signal
- any reward shaping should be rich

The future

- local optima are good enough (is any human behaviour globally optimal)
- improvements in hardware help with sample inefficiency
- more learning signal - hallucinating rewards, auxillary tasks, model learning
- model learning fixes a bunch of problems - difficulty is learning one

Many things need to go right for RL to work - success stories are the exception, not the rule

Inverse reinforcement learning

Figure 8: Chelsea Finn – Berkley Deep RL Bootcamp 2017

Closing thoughts

Exploration versus exploitation

Test your models on simple problems

Reinforcement learning is sample inefficient

Deep RL is hard

Reward engineering is key