Deep reinforcement learning

Hado van Hasselt

DeepMind

Big picture

History

Big picture

- Industrial revolution (1750 - 1850) and Machine Age (1870 - 1940)
    - Implement **repetitive manual solutions** with machines

History

Big picture

- Industrial revolution (1750 - 1850) and Machine Age (1870 - 1940)
    - Implement **repetitive manual solutions** with machines
- Digital revolution (1960 - now) and Information Age
    - Implement **repetitive mental solutions** with machines

History

Big picture

- Industrial revolution (1750 - 1850) and Machine Age (1870 - 1940)
  - Implement **repetitive manual solutions** with machines
- Digital revolution (1960 - now) and Information Age
  - Implement **repetitive mental solutions** with machines

In both cases: have to come up with solution first

History

Big picture

- Industrial revolution (1750 - 1850) and Machine Age (1870 - 1940)
  - Implement **repetitive manual solutions** with machines
- Digital revolution (1960 - now) and Information Age
  - Implement **repetitive mental solutions** with machines

In both cases: have to come up with solution first

- AI revolution
  - We only specify the goal, **solutions are found autonomously**

Artificial intelligence

Big picture

- Symbolic GOFAI
  - Conclusions are derived, but rules are programmed and static
  - Hand-picked knowledge formalism & level of abstraction
  - Hard to deal with messy data and uncertainty

Artificial intelligence

Big picture

- Symbolic GOFAI
  - Conclusions are derived, but rules are programmed and static
  - Hand-picked knowledge formalism & level of abstraction
  - Hard to deal with messy data and uncertainty

- Classic statistics
  - Analyse data
  - We make decisions based on analysis

Artificial intelligence

Big picture

- Symbolic GOFAI
  - Conclusions are derived, but rules are programmed and static
  - Hand-picked knowledge formalism & level of abstraction
  - Hard to deal with messy data and uncertainty

- Classic statistics
  - Analyse data
  - We make decisions based on analysis

- True AI should learn to make decisions autonomously

Reinforcement learning

Reinforcement learning

A framework for making decisions

- RL provides a general-purpose framework for making decisions

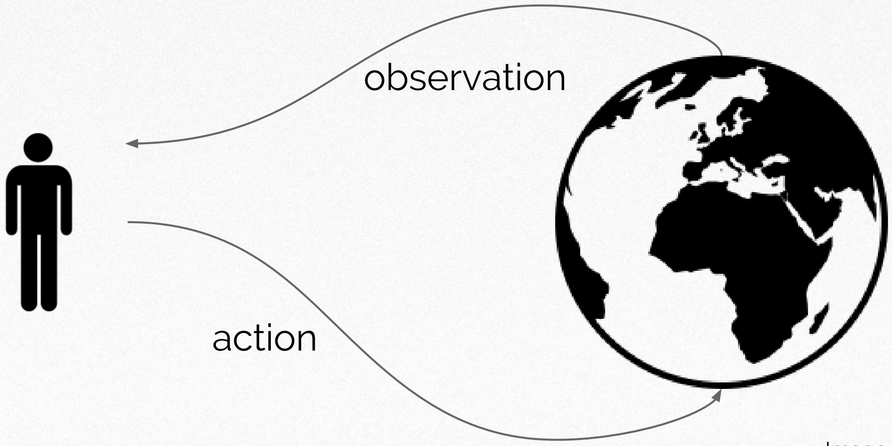


Image credits - AIGA Collection, Martin Vanco

Reinforcement learning

A framework for making decisions

- RL provides a general-purpose framework for making decisions
  - RL is about **learning to act**
  - Each action can alter the **state** of the world, and can result in **reward**
  - Goal: **optimize future rewards** (which may be internal to the agent)

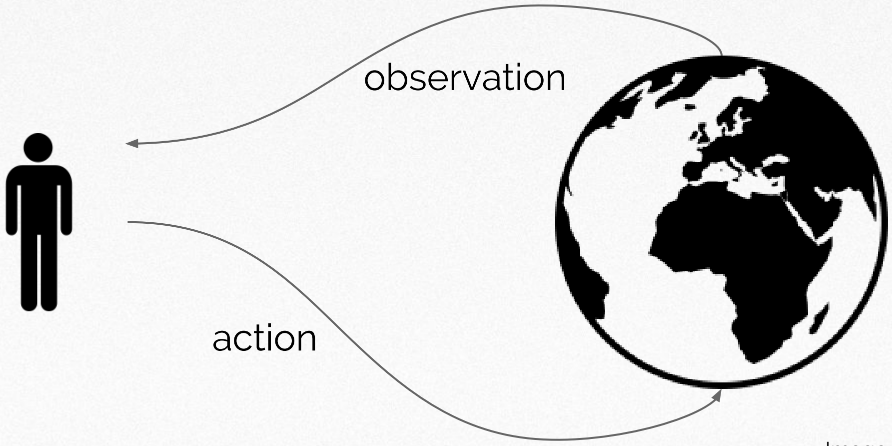


Image credits - AIGA Collection, Martin Vanco

Reinforcement learning

Examples

- Examples of reinforcement learning domains:
  - Video games (including Atari)
  - Board games (including the game of Go)
  - Robotics
  - Recommender systems
  - ...

Reinforcement learning

Examples

- Examples of reinforcement learning domains:
  - Video games (including Atari)
  - Board games (including the game of Go)
  - Robotics
  - Recommender systems
  - ...

- Essentially, problems that involves making decisions and/or making predictions about the future

Approaches to reinforcement learning

- The goal is to learn a policy of behaviour
- (At least) three possibilities:
  - Learn policy directly
  - Learn values of each action - infer policy by inspection
  - Learn a model - infer policy by planning

Approaches to reinforcement learning

- The goal is to learn a policy of behaviour
- (At least) three possibilities:
    - Learn policy directly
    - Learn values of each action - infer policy by inspection
    - Learn a model - infer policy by planning
- Agents therefore typically have at least one of these components:
    - Policy - maps current state to action
    - Value function - prediction of value for each state and action
    - Model - agent's representation of the environment.

Reinforcement learning

Components

- Policy : $\pi(s) = a$
- Value : $Q(s, a) \approx \mathbb{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + \ldots \mid S_t = s, A_t = a\right]$
- Model : $m(s, a) \approx \mathbb{E}\left[S_{t+1} \mid S_t = s, A_t = a\right]$

Reinforcement learning

Components

- Policy : $\pi(s) = a$
- Value : $Q(s, a) \approx \mathbb{E}\left[R_{t+1} + R_{t+2} + R_{t+3} + \ldots \mid S_t = s, A_t = a\right]$
- Model : $m(s, a) \approx \mathbb{E}\left[S_{t+1} \mid S_t = s, A_t = a\right]$

- All components are functions
- We need to represent and learn these functions

Deep reinforcement learning

Deep reinforcement learning

Use **deep learning** to learn
policies, **values**, and/or **models**
to use in a reinforcement learning domain

Deep reinforcement learning

- Reinforcement learning provides: a framework for making decisions
- Deep learning provides: tools to learn components

Deep reinforcement learning

- Reinforcement learning provides: a framework for making decisions
- Deep learning provides: tools to learn components

AI = RL + DL ?

- Concretely, we implement RL components with deep neural networks

Deep Q Networks

Q-learning

An algorithm to learn values

- The optimal value function fulfills:

$$Q^*(s,a) = \mathbb{E}\left[R_{t+1} + \max_b Q^*(S_{t+1}, b) \mid S_t = s, A_t = a\right]$$

(Bellman, 1957)

Q-learning

An algorithm to learn values

- The optimal value function fulfills:

$$Q^*(s, a) = \mathbb{E}\left[R_{t+1} + \max_b Q^*(S_{t+1}, b) \mid S_t = s, A_t = a\right]$$

(Bellman, 1957)

- We can turn this into a TD algorithm:

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(S_{t+1}, a) - Q_t(S_t, A_t)\right)$$

(Watkins 1989)

Q-learning

An algorithm to learn values

- By learning **off-policy** about the policy that is currently greedy, Q-learning can approximate the **optimal value function Q^***
- With Q^*, we have an optimal policy:

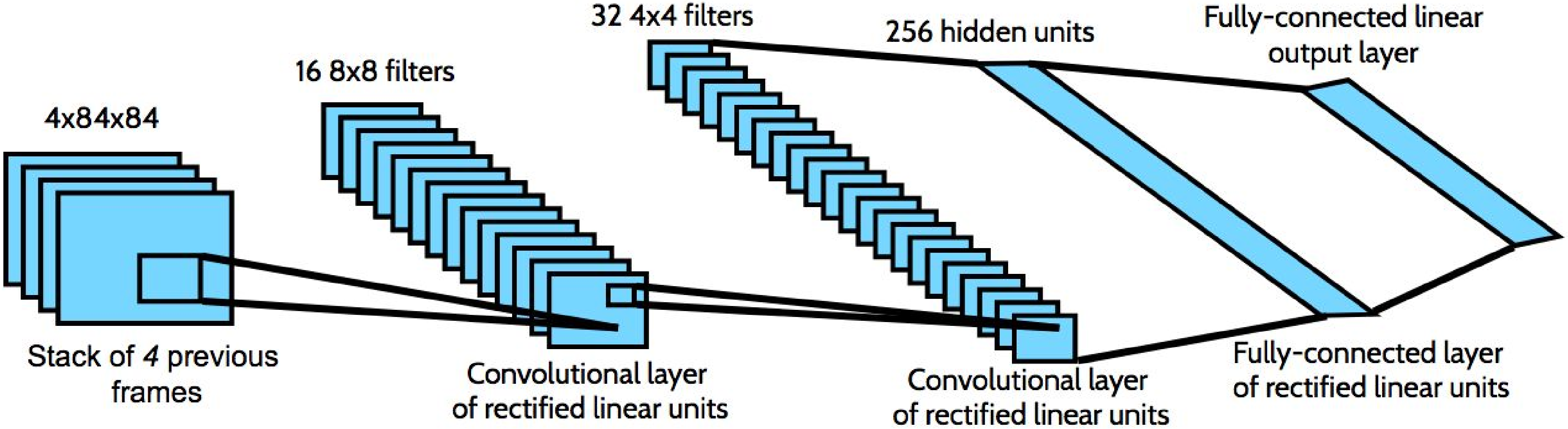
$$\pi^*(s) = \operatorname{argmax} Q^*(s, .\,)$$

DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

- Learns to play video games simply by playing
- Can learn Q function by Q-learning

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$



DQN

- Aside: we can phrase the update as a **loss**

$$\text{minimize} \quad \frac{1}{2}\|y - q(s, a; \theta)\|_2 \quad \text{where, e.g.,} \quad y = R_{t+1} + \gamma \max_a q(S_{t+1}, a; \theta)$$

- Typically, we consider the target *y* as constant, and ignore the dependence on the parameters
  - E.g., in TensorFlow you might use placeholders, or a stop_gradient
  - Interpretation: *y* is an estimate for (off-policy) expected return E[$G_t \mid \pi, a$]
  - Then just update towards this estimate

DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

- Learns to play video games simply by playing
- Can learn Q function by Q-learning

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

- Core components of DQN include:
  - Target networks (Mnih et al. 2015)

$$\Delta \boldsymbol{w} = \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{w}^-) - Q(S_t, A_t; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(S_t, A_t; \boldsymbol{w})$$

  - Experience replay (Lin 1992): replay previous tuples (s, a, r, s')

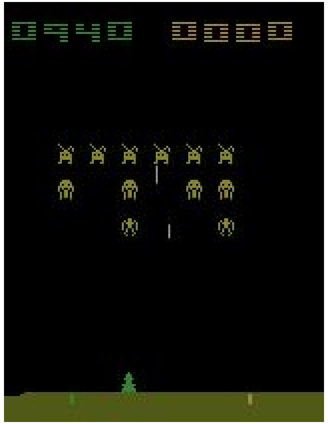
Target Network Intuition

(Slide credit: Vlad Mnih)

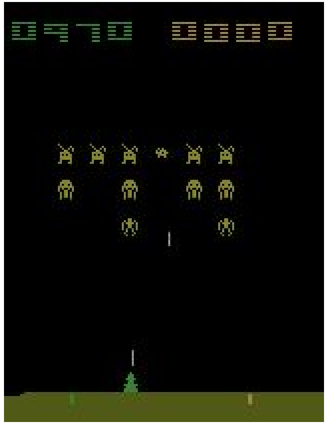
- Changing the value of one action will change the value of other actions and similar states.
- The network can end up chasing its own tail because of bootstrapping.
- Somewhat surprising fact - bigger networks are less prone to this because they alias less.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r \sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i) \right)^2$$

s



s'



“Human-Level Control Through Deep Reinforcement Learning”, Mnih, Kavukcuoglu, Silver et al. (2015)

Experience replay

- Idea: store experiences, learn from them more than once
  - In Nature DQN, sample uniformly, see each sample 4 times on average
- Benefits:
  - More data efficient
  - Learning resembles supervised learning more (deep learning likes this)

DQN

(Mnih, Kavukcuoglu, Silver, et al., Nature 2015)

- Many later improvements to DQN
    - Double Q-learning (van Hasselt 2010, van Hasselt et al. 2015)
    - Prioritized replay (Schaul et al. 2016)
    - Dueling networks (Wang et al. 2016)
    - Asynchronous learning (Mnih et al. 2016)
    - Adaptive normalization of values (van Hasselt et al. 2016)
    - Better exploration (Bellemare et al. 2016, Ostrovski et al., 2017, Fortunato, Azar, Piot et al. 2017)
    - … many more …

Experience replay

- We can view the replay as an **empirical** (non-parametric) **model**
- Can we query this model more cleverly?
- Yes:
    - Sample non-uniformly: prioritized replay really helps! (Schaul et al. 2016)
    - Can even 'plan' - episodic control (Blundell, et al. 2016, Pritzel et al. 2017)

Prioritized Experience Replay

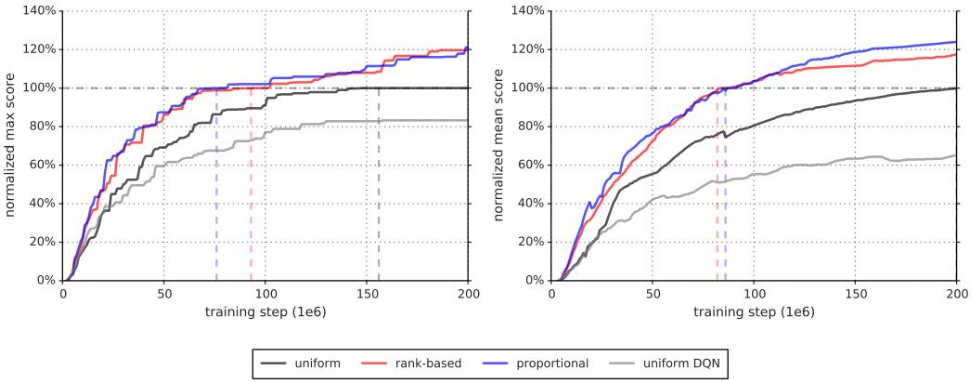
(Slide credit: Vlad Mnih)

- Replaying all transitions with equal probability is highly suboptimal.
- Replay transitions in proportion to absolute Bellman error:

$$\left| r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right|$$

- Leads to much faster learning.

	DQN		Double DQN (tuned)		
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57



"Prioritized Experience Replay", Schaul et al. (2016)

Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)

DQN:

$$\Delta \boldsymbol{w} = \alpha \left(r + \max_{a'} Q(s', a'; \boldsymbol{w}^{-}) - Q(s, a; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(s, a; \boldsymbol{w})$$

Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)

DQN:

$$\Delta \boldsymbol{w} = \alpha \left(r + \max_{a'} Q(s', a'; \boldsymbol{w}^-) - Q(s, a; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(s, a; \boldsymbol{w})$$

=

$$\Delta \boldsymbol{w} = \alpha \left(r + Q(s', \arg\max_{a'} Q(s', a'; \boldsymbol{w}^-); \boldsymbol{w}^-) - Q(s, a; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(s, a; \boldsymbol{w})$$

Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)

DQN:

$$\Delta \boldsymbol{w} = \alpha \left(r + \max_{a'} Q(s', a'; \boldsymbol{w}^{-}) - Q(s, a; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(s, a; \boldsymbol{w})$$

=

$$\Delta \boldsymbol{w} = \alpha \left(r + Q(s', \arg\max_{a'} Q(s', a'; \boldsymbol{w}^{-}); \boldsymbol{w}^{-}) - Q(s, a; \boldsymbol{w}) \right) \nabla_{\boldsymbol{w}} Q(s, a; \boldsymbol{w})$$

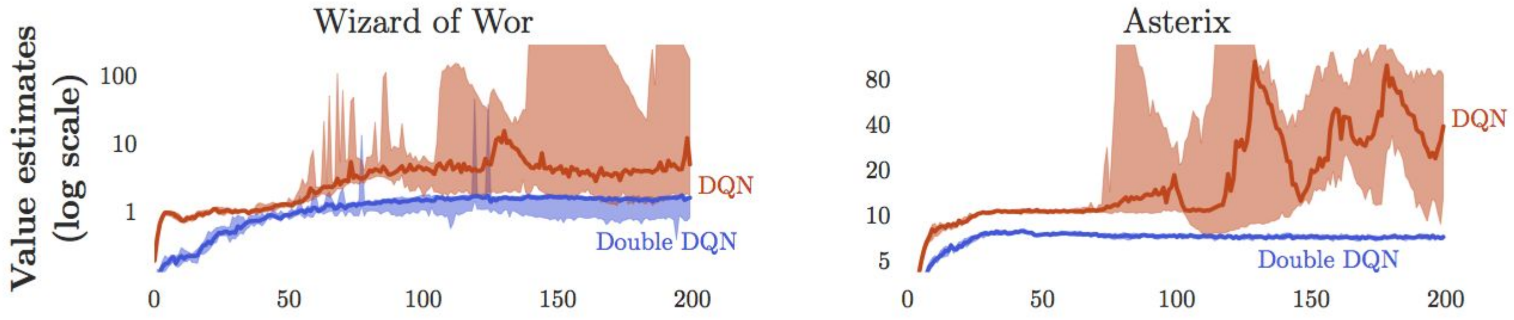
Double DQN:

$$\Delta \boldsymbol{w} = \alpha (r + Q(s', \arg\max_{a'} Q(s', a'; \boldsymbol{w}); \boldsymbol{w}^{-}) - Q(s, a)) \nabla_{\boldsymbol{w}} Q(s, a; \boldsymbol{w})$$

Idea: decorrelate selection and evaluation to mitigate overestimation

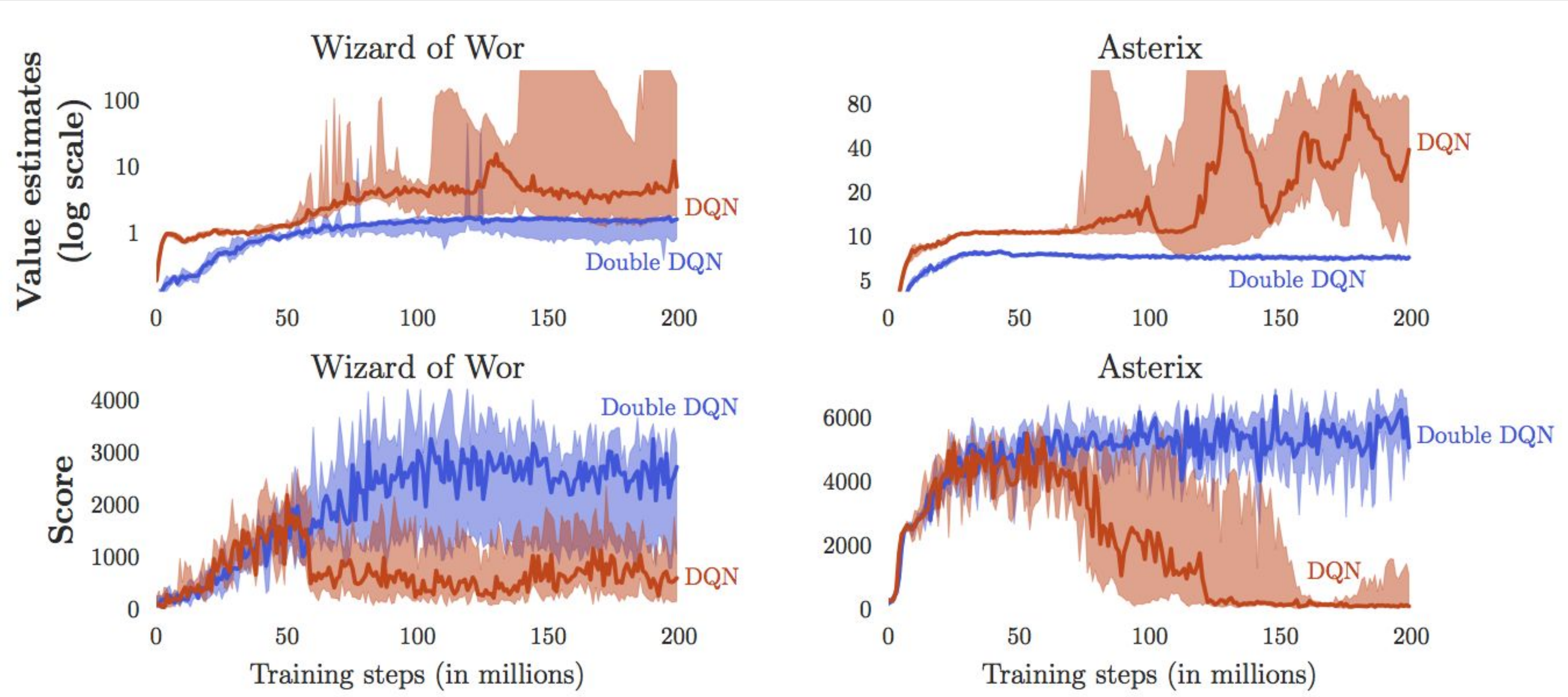
Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)



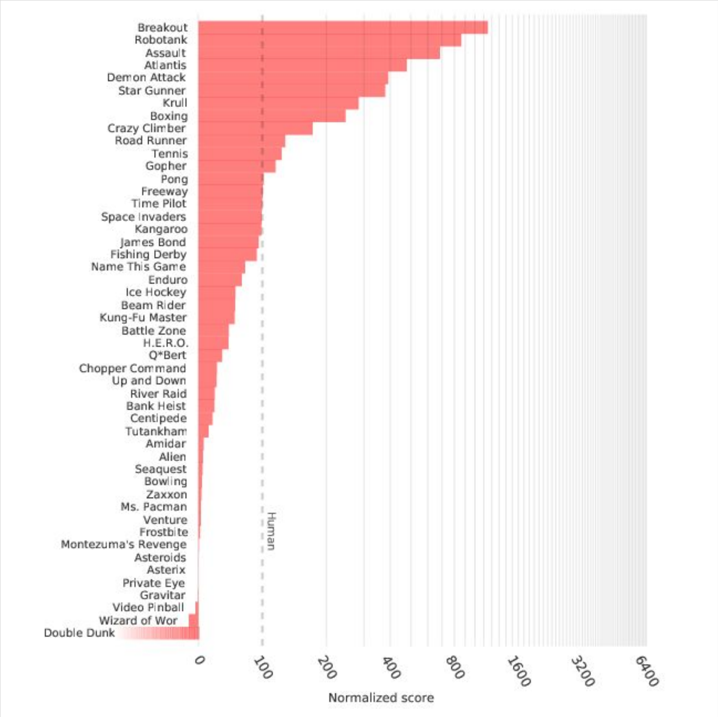
Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)



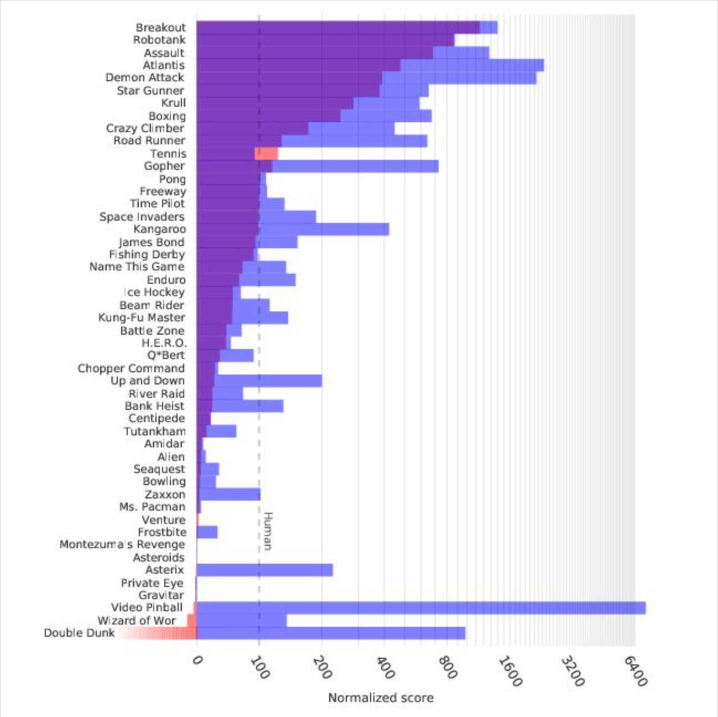
Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)



Double DQN

(van Hasselt, Guez, Silver, AAAI 2015)



DeepMind

Insights

- The take-home message is:
    - Be aware of the properties of your learning algorithms
    - Track and analyse statistics
    - If you understand what the problem is, a solution is sometimes very simple

Insights

- The take-home message is:
  - Be aware of the properties of your learning algorithms
  - Track and analyse statistics
  - If you understand what the problem is, a solution is sometimes very simple
- **RL-aware DL** and **DL-aware RL**
  - Target networks, experience replay: DL-aware RL
  - Next up, dueling networks: RL-aware DL

Dueling DQN

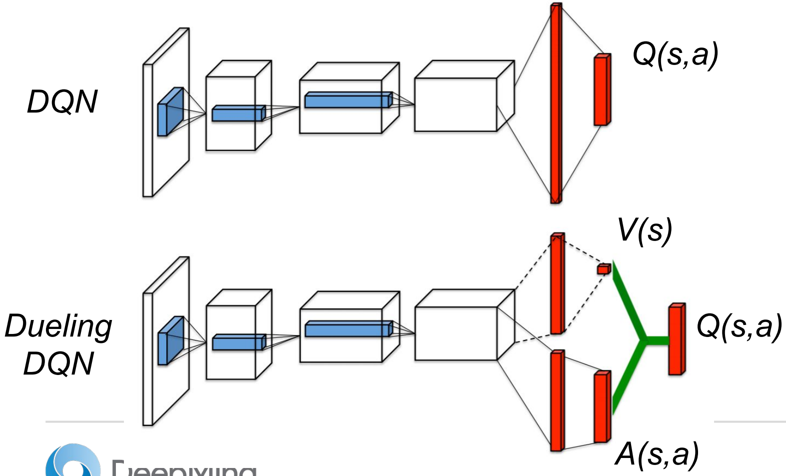
(Slide credit: Vlad Mnih)

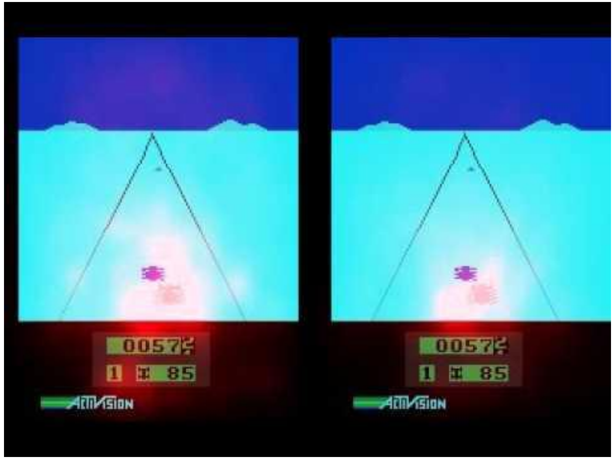
- Value-Advantage decomposition of Q:

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

- Dueling DQN (Wang et al., 2015):

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a=1}^{|\mathcal{A}|} A(s,a)$$





Atari Results

	30 no-ops		Human Starts	
	Mean	**Median**	**Mean**	**Median**
Prior. Duel Clip	**591.9%**	**172.1%**	**567.0%**	**115.3%**
Prior. Single	434.6%	123.7%	386.7%	112.9%
Duel Clip	**373.1%**	**151.5%**	**343.8%**	**117.1%**
Single Clip	341.2%	132.6%	302.8%	114.1%
Single	307.3%	117.8%	332.9%	110.9%
Nature DQN	227.9%	79.1%	219.6%	68.5%

"Dueling Network Architectures for Deep Reinforcement Learning", Wang et al. (2016)

Rewards

Defining optimality

- A task is defined by its rewards
  - Atari: change in score
  - Go: win (+1) or lose (-1)

Rewards

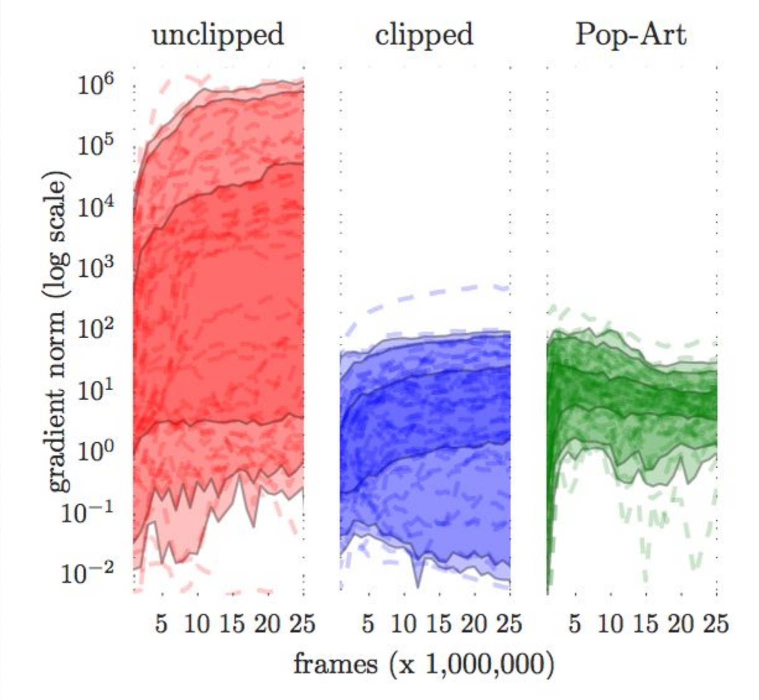
Defining optimality

- A task is defined by its rewards
  - Atari: change in score
  - Go: win (+1) or lose (-1)
- In DQN, all rewards were clipped to [-1, 1]
  - This helps learning
  - But it also changes the objective

Adaptive normalization

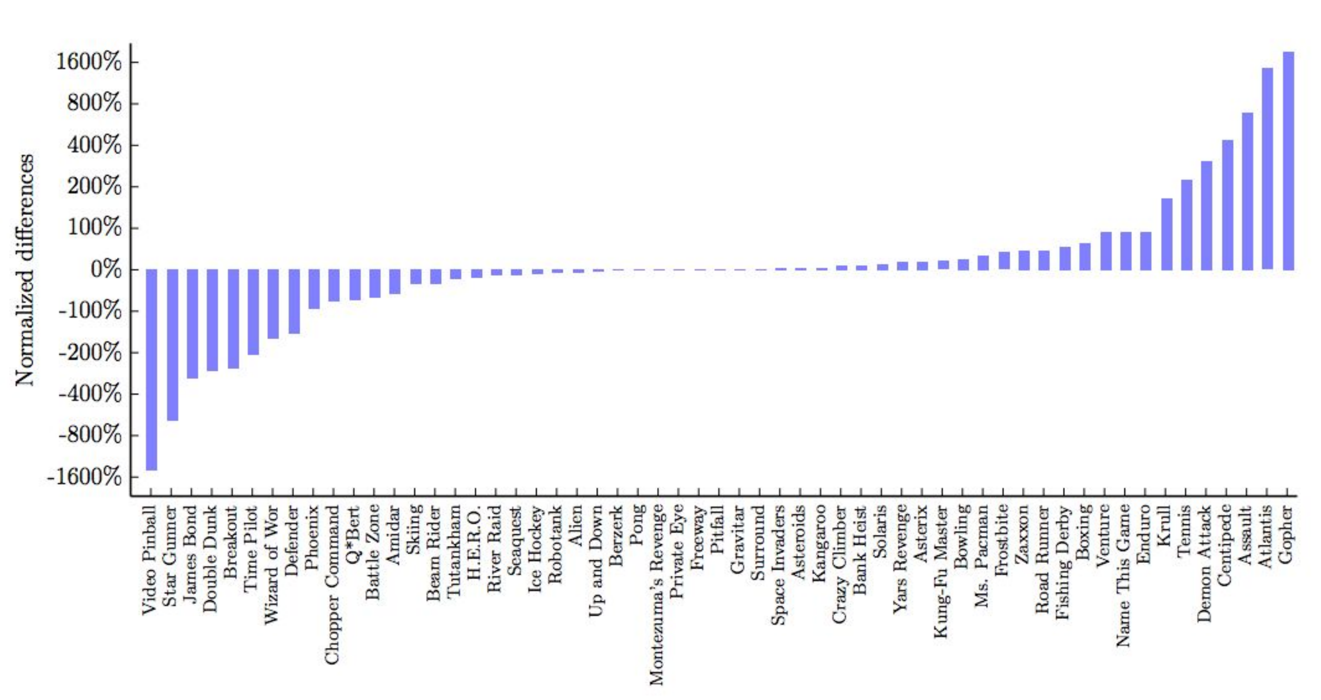
(van Hasselt et al. NIPS 2016)

- Optimization algorithms like normalized updates
- Clipping rewards is one solution, but we can do better
- We tried **adaptive target normalization** (algorithm is called Pop-Art)



Adaptive normalization

(van Hasselt et al. NIPS 2016)



Unclipping rewards

Videos at: hadovanhasselt.com/2016/08/17/atari-videos/

Policy gradients and actor-critic methods

Several slides adapted from Vlad Mnih

Policy Gradient

- We can often do better if the policy is differentiable.
  - Optimize the performance with gradient descent.
- The goal is to compute the gradient of the objective:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots\right]$$

- How can we compute this when rewards aren't differentiable?
- It turns out that there is a simple unbiased estimate of this gradient.

Contextual Bandit Policy Gradient

- Consider the simple one-step MDP (contextual bandit) setting.
- Start states are distributed according to d and episodes are one step long.

$$
\begin{aligned}
\nabla_\theta \mathbb{E}[R(S,A)] &= \nabla_\theta \sum_s d(s) \sum_a \pi_\theta(a|s) R(s,a) \\
&= \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(a|s) R(s,a) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} R(s,a) \\
&= \sum_s d(s) \sum_a \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s) R(s,a) \\
&= \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S,A)]
\end{aligned}
$$

Likelihood ratio trick

Contextual Bandit Policy Gradient

- The gradient of the expected reward is given by:

$$\nabla_\theta \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_\theta \log \pi_\theta(A|S) R(S, A)]$$

- We can approximate this with samples and update the policy using SGD:

$$\theta_{t+1} = \theta_t + \alpha R_{t+1} \nabla_\theta \log \pi_{\theta_t}(A_t|S_t)$$

Policy Gradient Theorem

- A more general result applies to full multi-step MDPs.
- For all differentiable policies:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right]$$

where expectation is over states and actions.

"Policy gradient methods for reinforcement learning with function approximation", Sutton et al. (2000)

- There is an easy sample-based approximation (REINFORCE):

$$\nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

where

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

"Simple statistical gradient-following algorithms for connectionist reinforcement learning", Williams (1992)

Variance Reduction

- The REINFORCE gradient suffers from high variance.
- Subtracting a **baseline** keeps the gradient unbiased and reduces the variance:

  $$\nabla_\theta \log \pi_\theta(a_t|s_t)\,(G_t - b(s_t))$$

- The state value function *V(s)* is a good choice for a baseline.
- Leads to a very intuitive form of update:

  $$\nabla_\theta \log \pi_\theta(a_t|s_t)\,(G_t - v(s_t))$$

- → Increase probability when action was better than expected

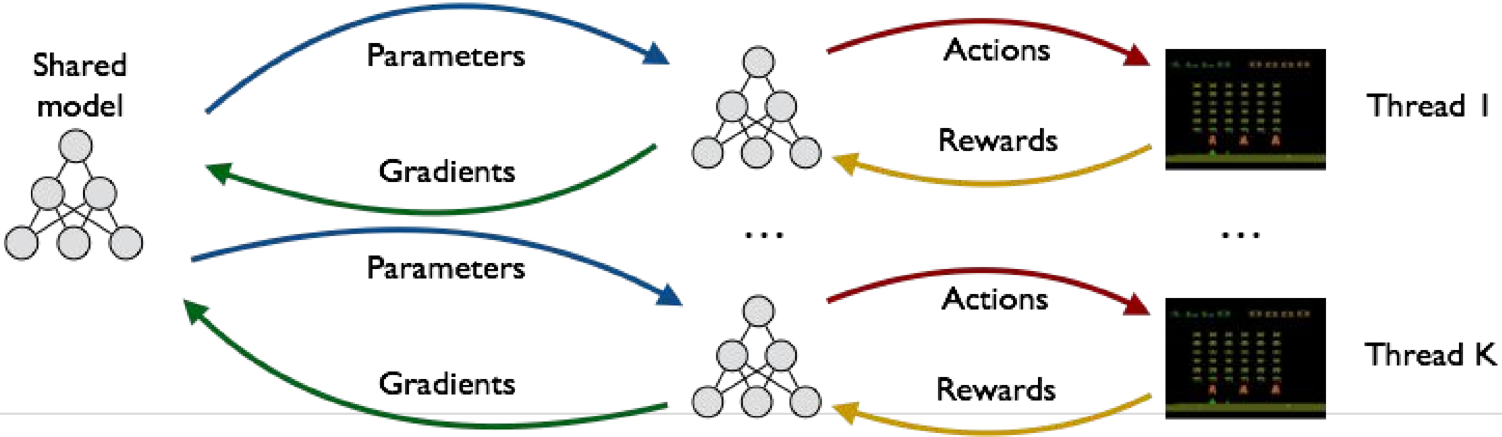
“Simple statistical gradient-following algorithms for connectionist reinforcement learning“, Williams (1992)

Practical Deep Policy Gradient

- How can policy-based methods be implemented efficiently with neural networks?
- DQN uses replay, but standard PG methods are on-policy:
    - Require samples from the current policy.
    - Good off-policy PG methods have since been developed:
        - See ACER (Wang et al., 2016) and PGQL (O'Donoghue et al., 2016).
    - Idea: sample from replay, but adapt the updates so that expected gradient looks as if we use the current policy

AsyncRL

- Asynchronous training of RL agents:
  - Parallel actor-learners implemented using **CPU threads** and shared parameters.
  - Online **Hogwild!**-style asynchronous updates (Recht et al., 2011, Lian et al., 2015).
  - No replay? Parallel actor-learners have a similar stabilizing effect.
  - Choice of RL algorithm: on-policy or off-policy, value-based or policy-based.



"Asynchronous Methods for Deep Reinforcement Learning", Mnih et al. (2016)

Asynchronous 1-step Q-Learning

- Parallel actor-learners compute online 1-step update

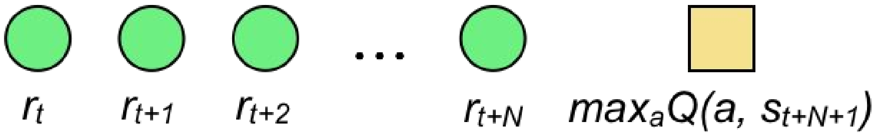
$$y \leftarrow r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

$$\Delta\theta \leftarrow \Delta\theta + \frac{\partial \left(y - Q(s, a; \theta)\right)^2}{\partial \theta}$$

- Gradients accumulated over minibatch before update

Asynchronous N-step Q-Learning

- Q-learning with a uniform mixture of backups of length 1 through N.



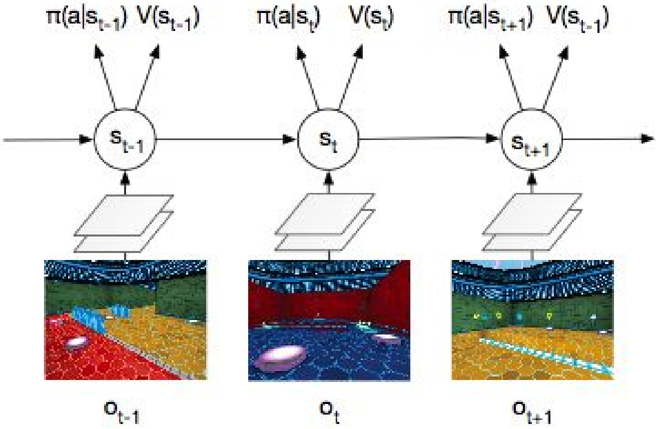
$$y \leftarrow \sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q(s_{t+N}, a'; \theta^-)$$

$$\Delta\theta \leftarrow \Delta\theta + \frac{\partial \left(y - Q(s_t, a_t; \theta)\right)^2}{\partial \theta}$$

- Variation of "Incremental multi-step Q-learning" (Peng & Williams, 1995).

Async Advantage Actor-Critic (A3C)

- The agent learns a **policy** and a state **value function**
- Uses bootstrapped n-step returns to reduce variance
- The policy gradient multiplied by an estimate of the advantage.
  - Similar to Generalized Advantage Estimation (Schulman et al, 2015).



$$\nabla_\theta \log \pi(a_t|s_t,\theta)\left(\sum_{k=0}^{N}\gamma^k r_{t+k} + \gamma^{N+1}V(s_{t+N+1}) - V(s_t)\right)$$

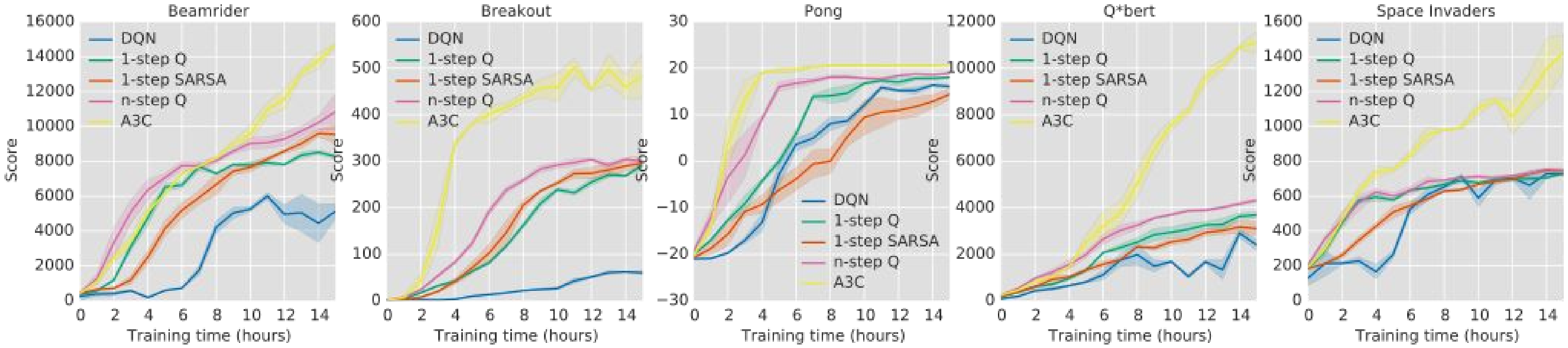
- Train value with n-step TD learning
- You can think of this as minimizing:

$$\left(\sum_{k=0}^{N}\gamma^k r_{t+k} + \gamma^{N+1}V(s_{s_{t+N+1}};\theta^-) - V(s_t;\theta)\right)^2$$

“Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al. (2016)

AsyncRL - Learning Speed

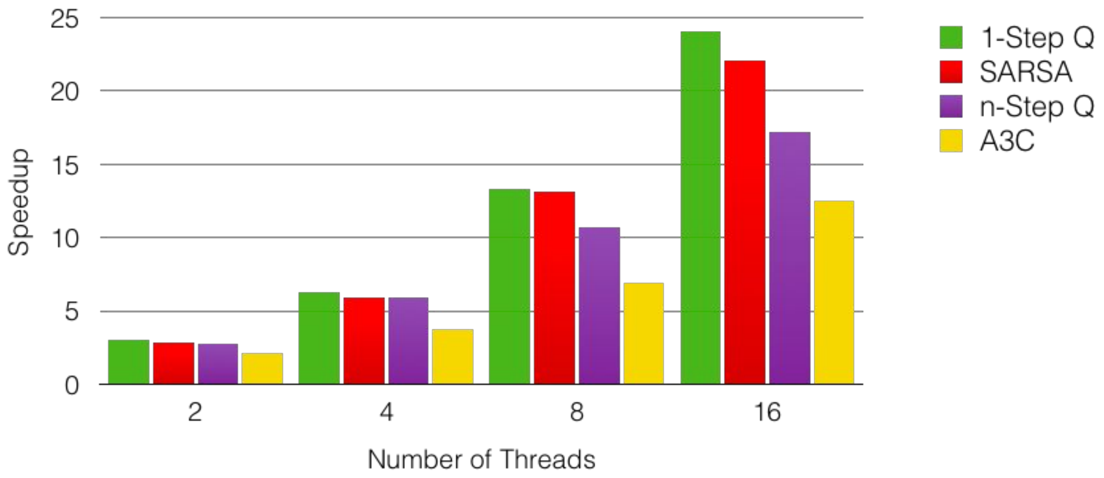
- Asynchronous methods trained on 16 CPU cores compared to DQN (blue) trained on a K40 GPU.
- n-step methods can be much faster than single step methods.
- Async advantage actor-critic tends to dominate the value-based methods.



“Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al. (2016)

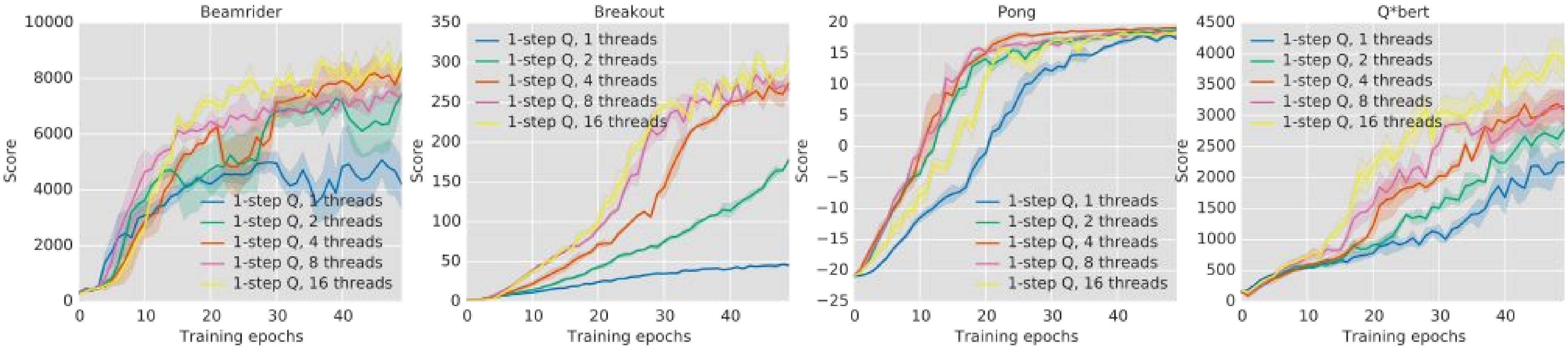
AsyncRL - Scalability

- Average speedup from using K threads to reach a reference score averaged over 7 Atari games.
- **Super-linear** speed-up for 1-step methods.



Data Efficiency of 1-Step Q-learning

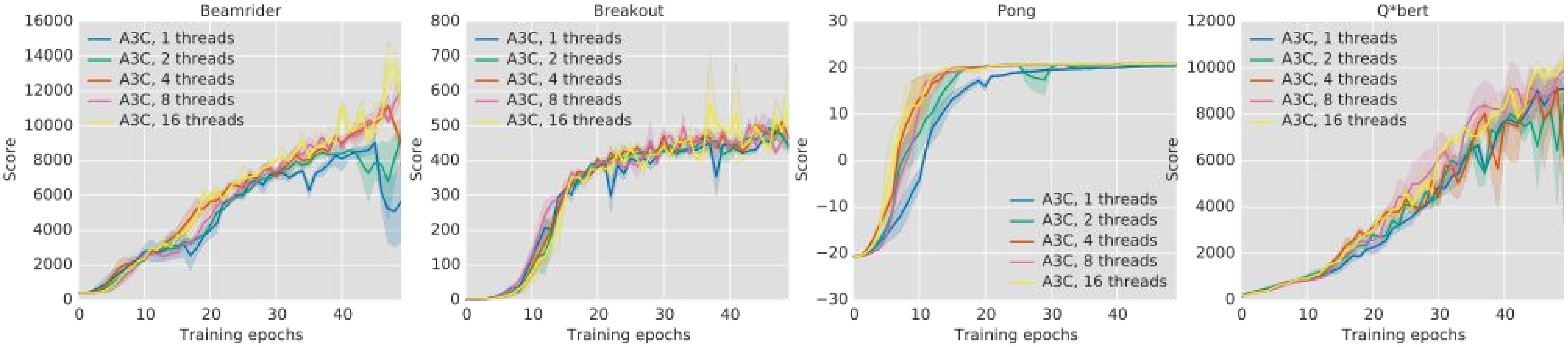
- Better **data efficiency** from more threads + speedup from parallel training
  - 1 thread (blue) 16 threads (yellow)



DeepMind

Data Efficiency of A3C

- No data-efficiency gains. Sub-linear speedup from parallel training.
  - 1 thread (blue) 16 threads (yellow)



A3C - ATARI Results

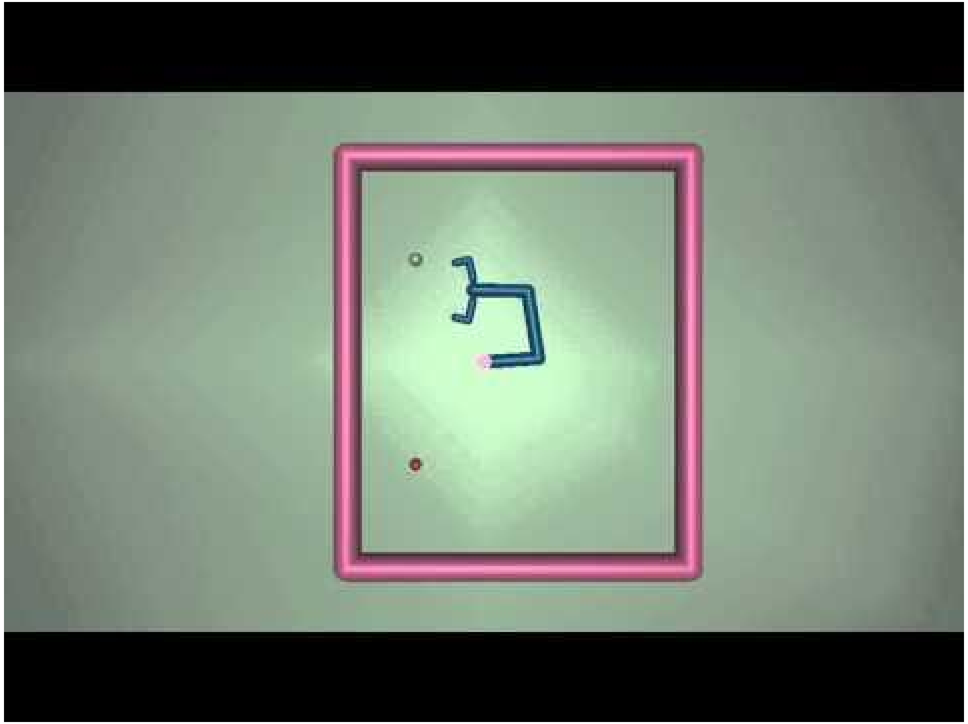
Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorilla	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

“Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al. (2016)

A3C - Procedural Maze Navigation in 3D

“Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al. (2016)

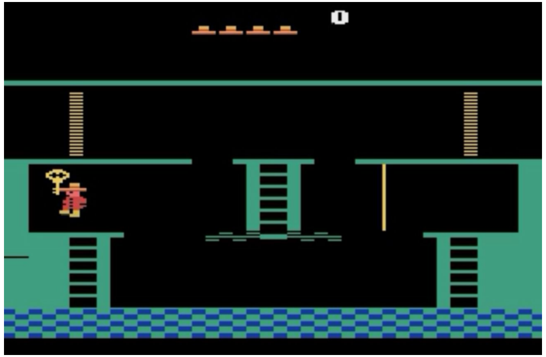
A3C - Continuous Control

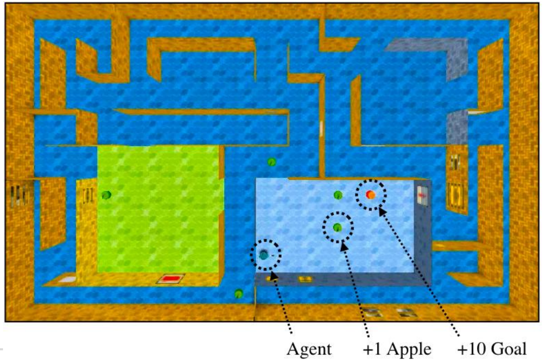


“Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al. (2016)

Unsupervised Reinforcement Learning

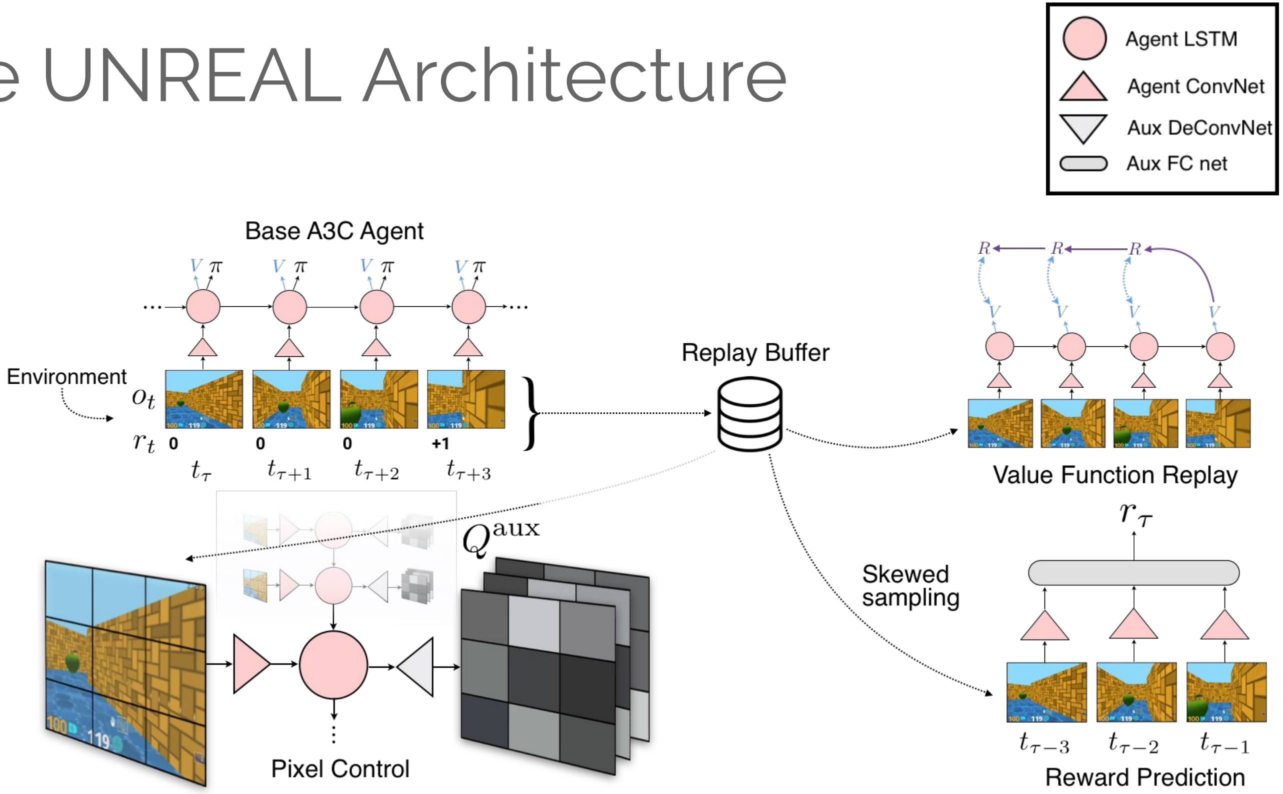
- The best deep RL methods are still very data hungry. Especially with **sparse rewards**.
- Obvious solution - Learn about the environment.
- Augment an RL agent with **auxiliary prediction and control tasks** to improve data efficiency.
- The UNREAL agent - UNsupervised REinforcement and Auxiliary Learning.
  - “Reinforcement Learning with Unsupervised Auxiliary Tasks”, (Jaderberg et al. 2017)





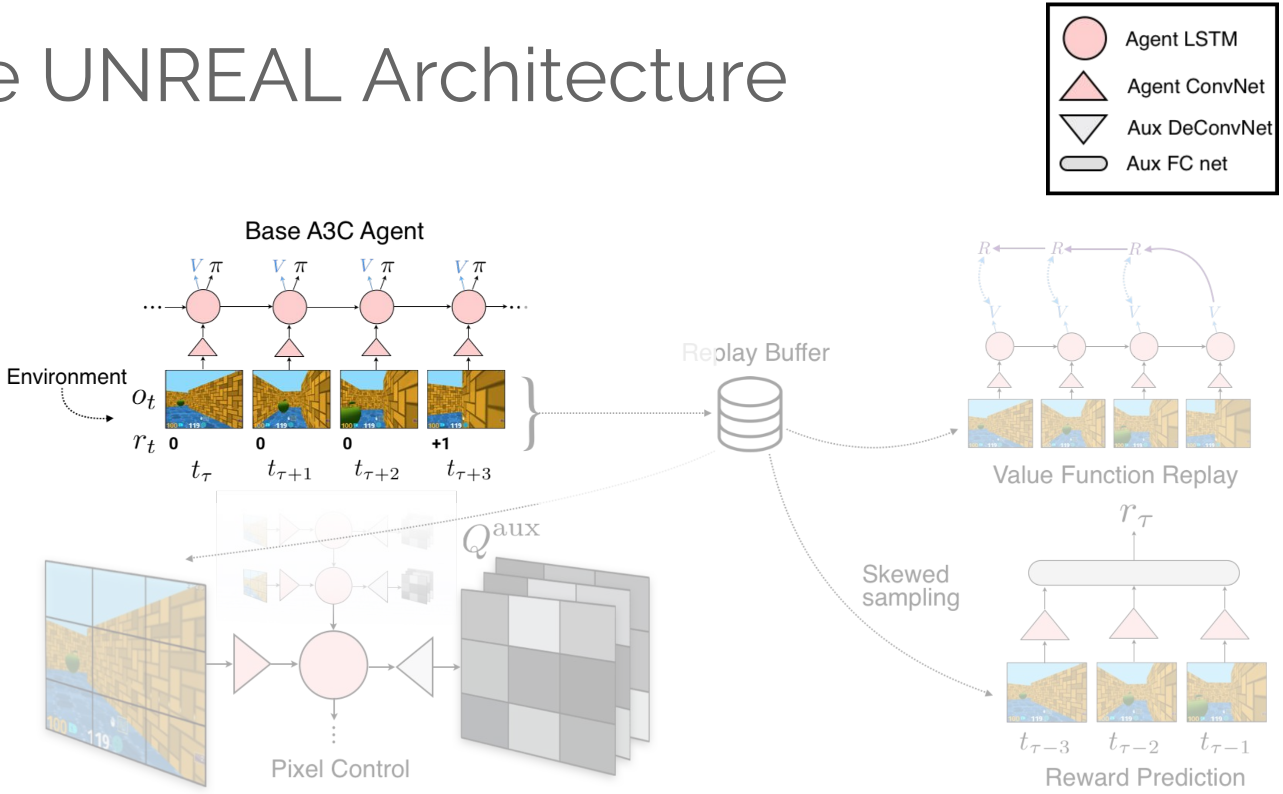
The UNREAL Architecture

- UNREAL augments an LSTM A3C agent with 3 auxiliary tasks.
- Can be used on top of DQN, DDPG, TRPO or other agents.



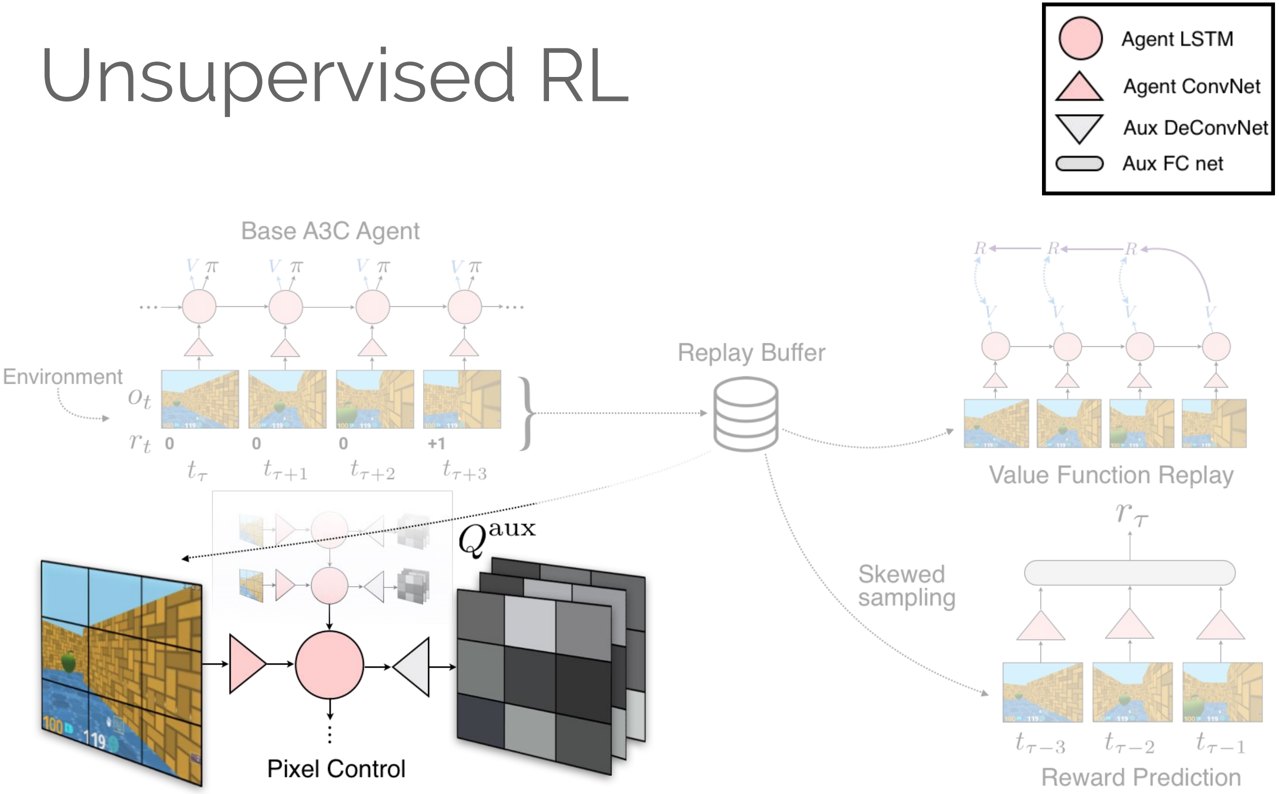
The UNREAL Architecture

- Base A3C LSTM agent learns from the environment's scalar reward signal.
- UNREAL acts using the base A3C agent's policy.



Unsupervised RL

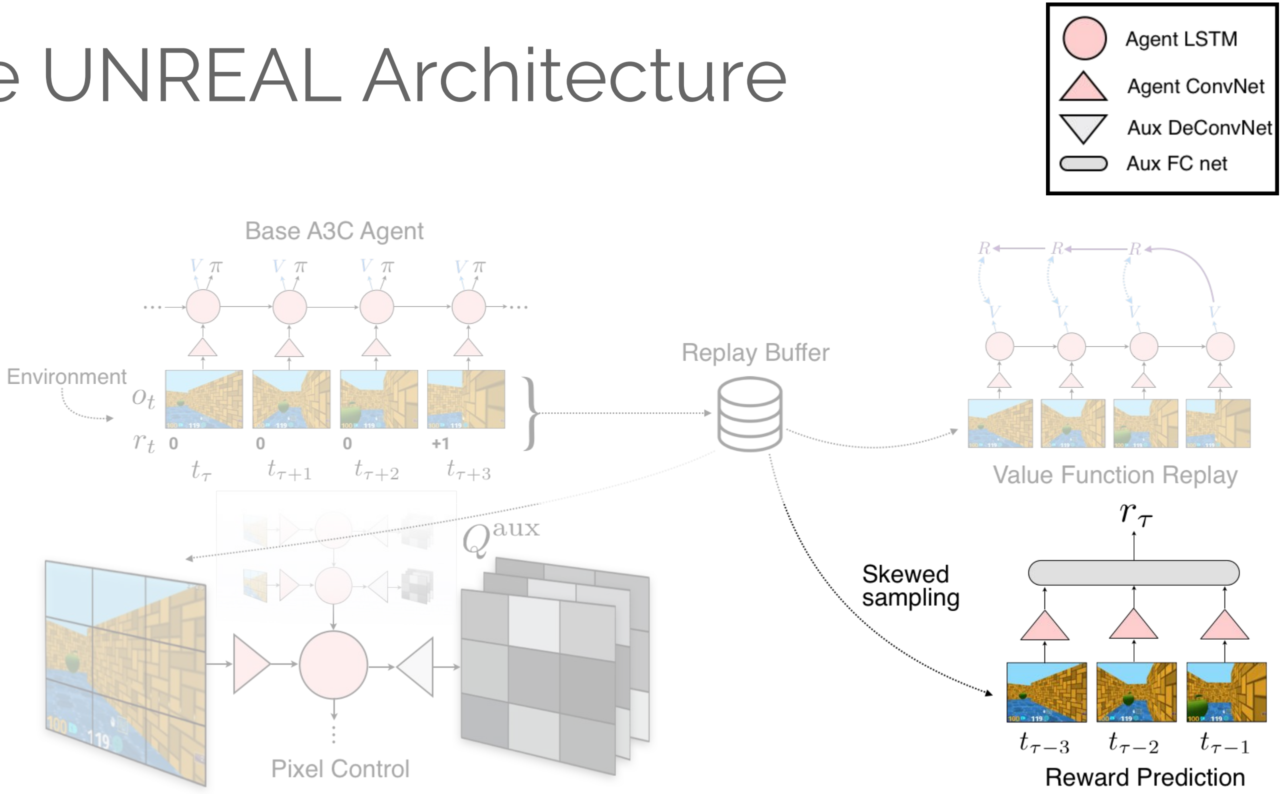
- Augment A3C with many **auxiliary control tasks**.
- Pixel control - learn to maximally change parts of the screen.
- Feature control (not used by UNREAL) - learn to control the internal representations.



The UNREAL Architecture

Focusing on rewards:

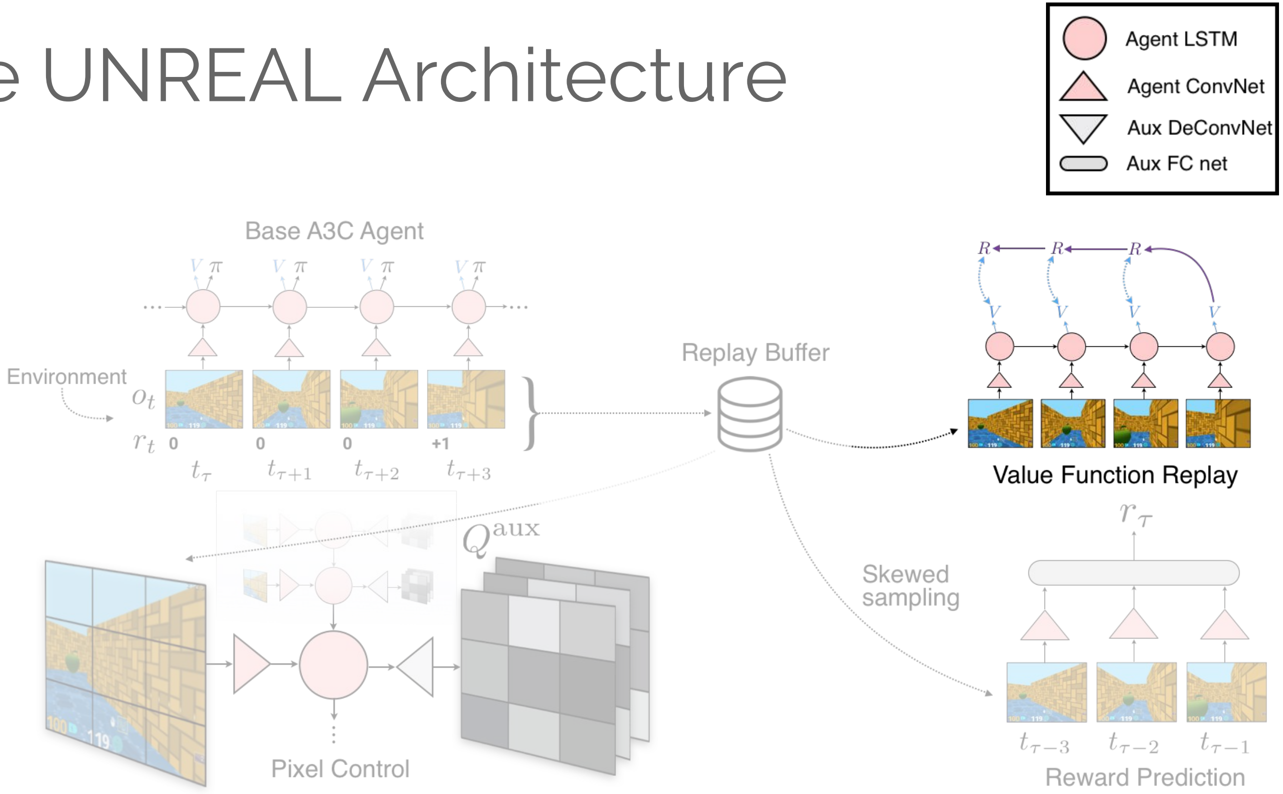
- Rebalanced reward prediction.
- Shape the agent's CNN by classifying whether a sequence of frames will lead to reward.
- No need to worry about off-policy learning.



The UNREAL Architecture

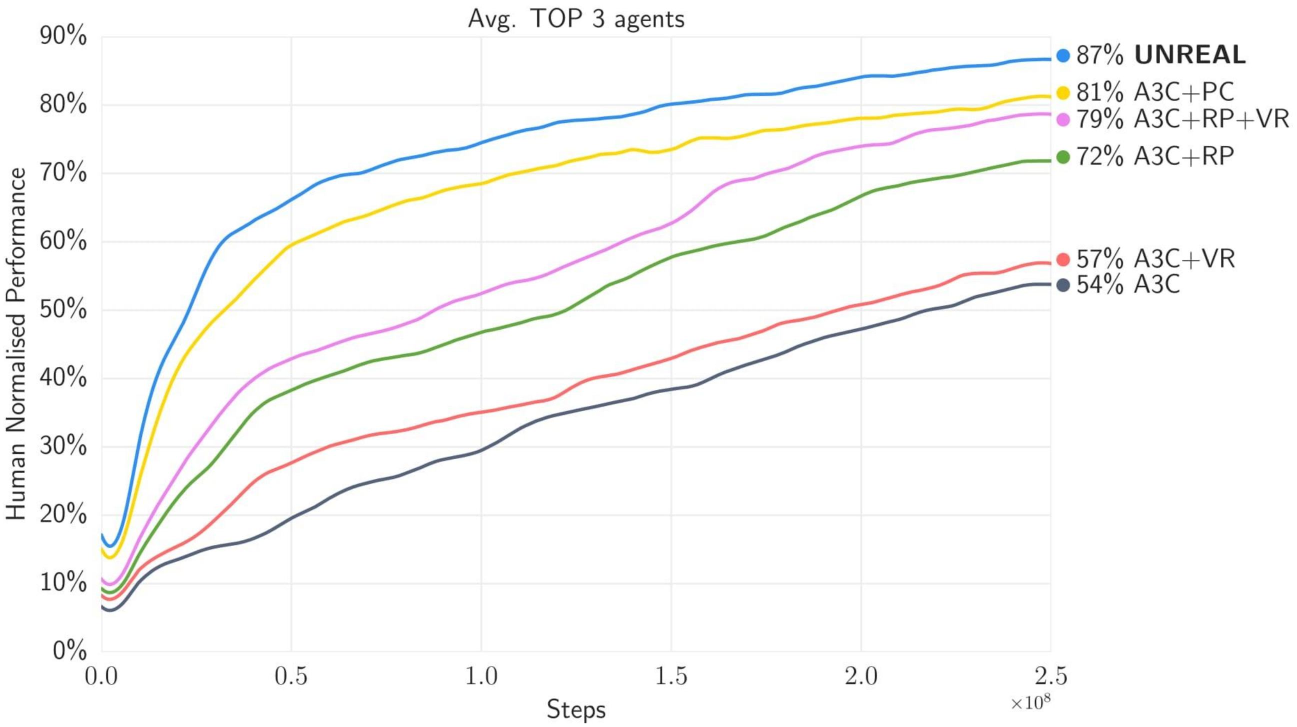
Focusing on rewards:

- Value function replay.
- Faster learning of the value function.



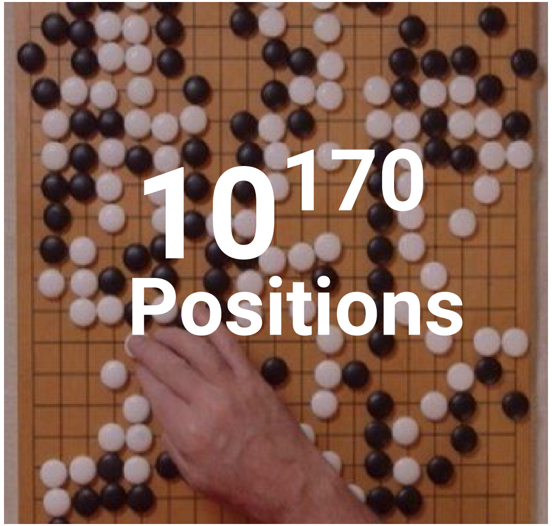
DeepMind Lab Results

- Average human-normalized performance on 13 3D environments from DeepMind Lab.
- Tasks include random maze navigation and laser tag.
- Roughly a 10x improvement in data efficiency over A3C.
- 60% improvement in final performance.



AlphaGo

Baduk in numbers

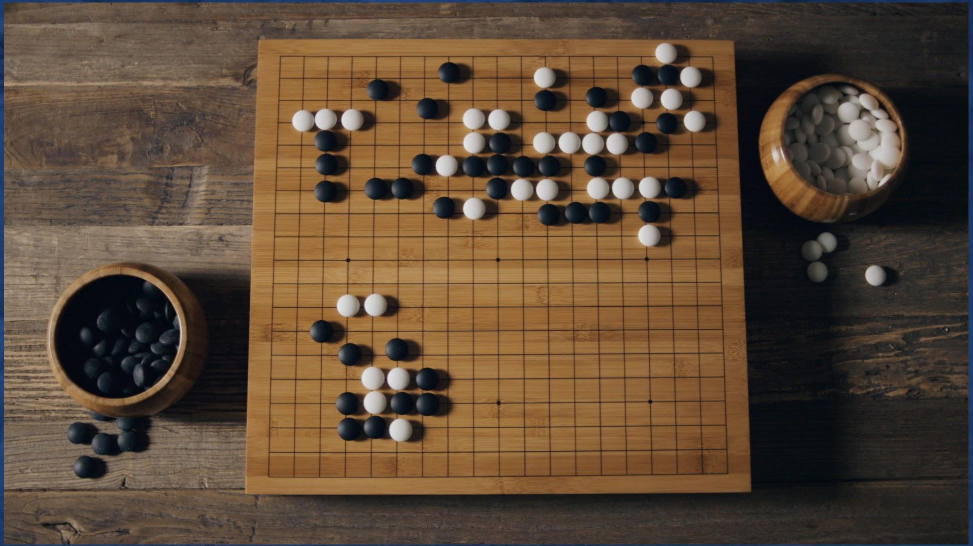


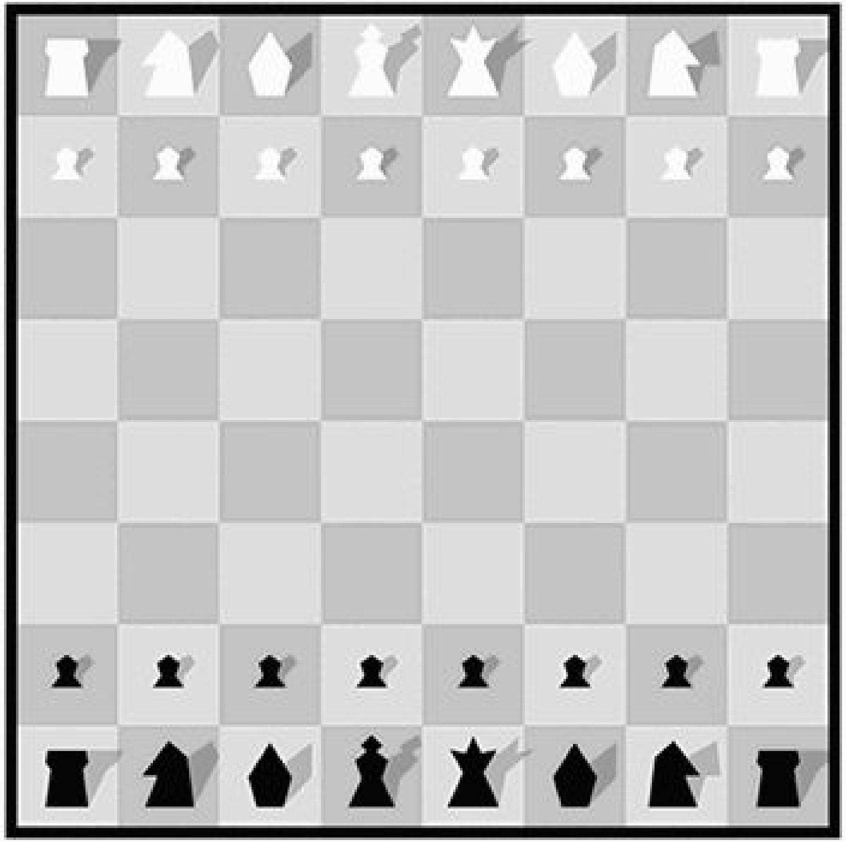
Why is Baduk hard for computers to play?

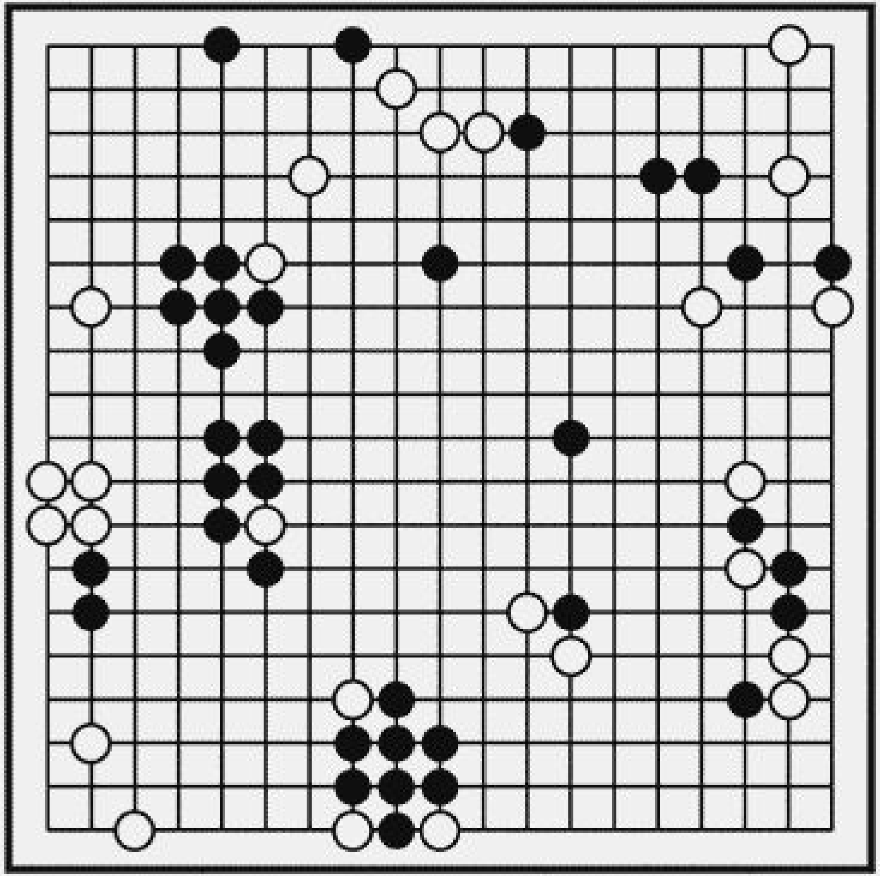
Game tree complexity = b^d

Brute force search intractable:

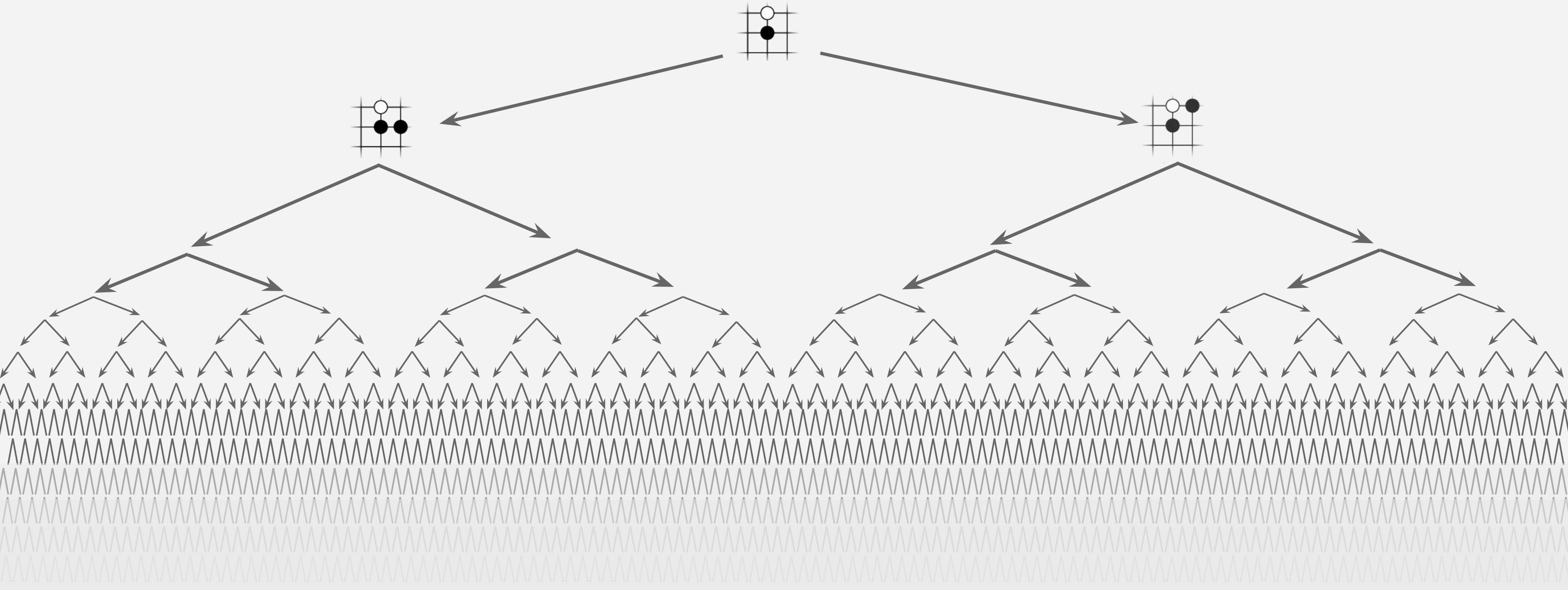
1. Search space is huge
2. “Impossible” for computers to evaluate who is winning



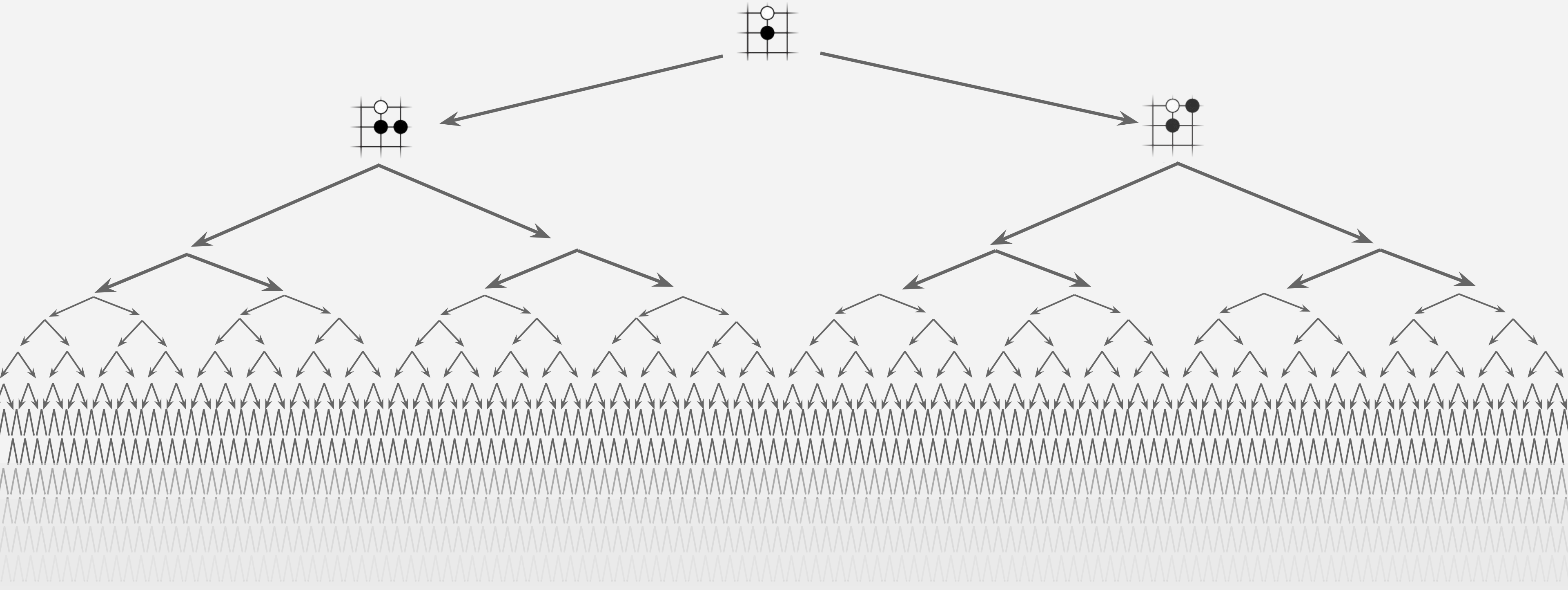




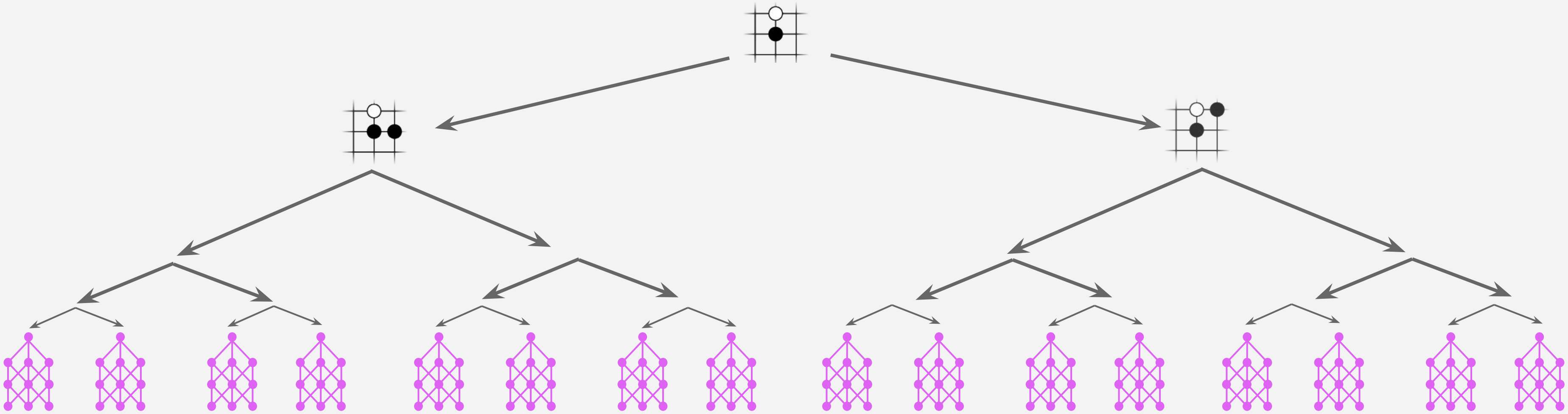
Exhaustive search



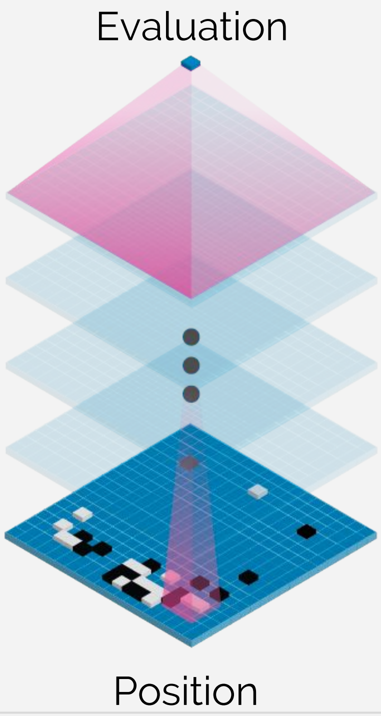
Reducing depth with value network

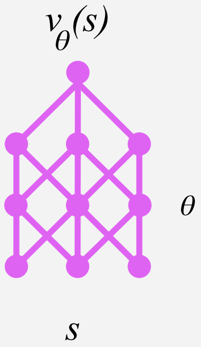


Reducing depth with value network

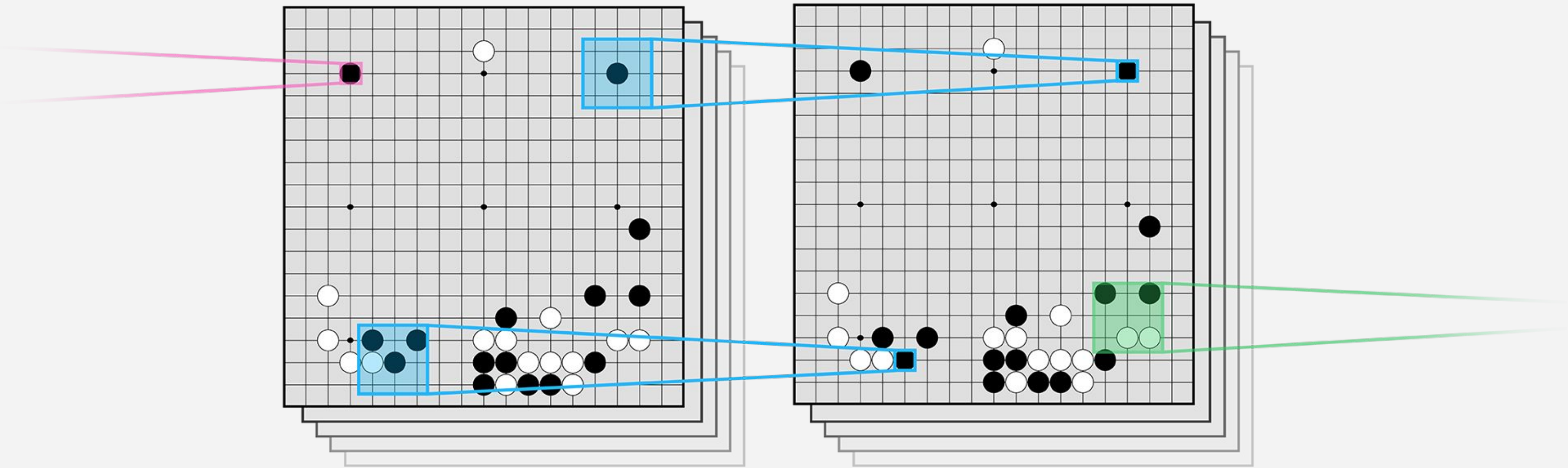


Value network

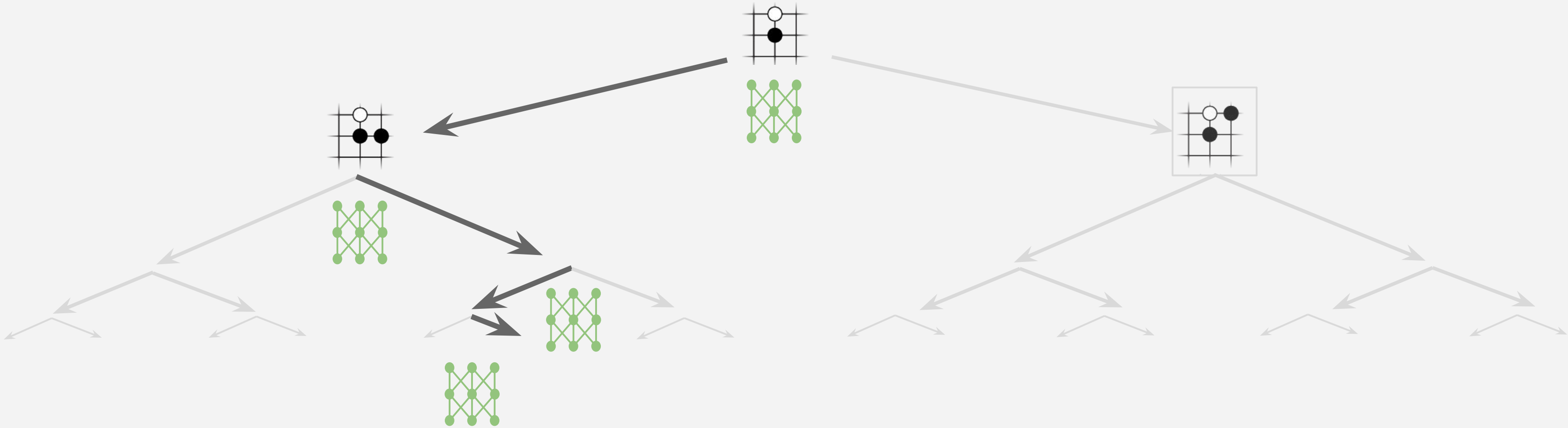




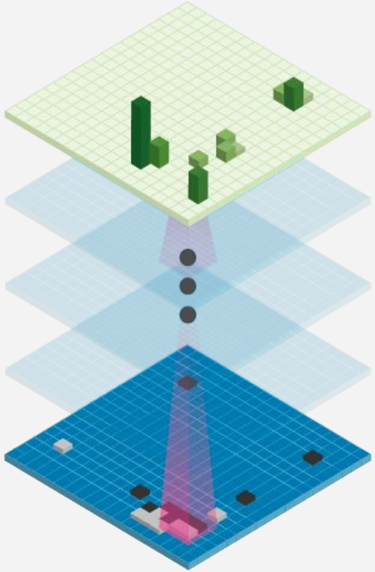
Convolutional neural network

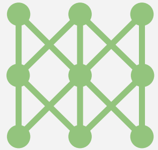


Reducing breadth with policy network

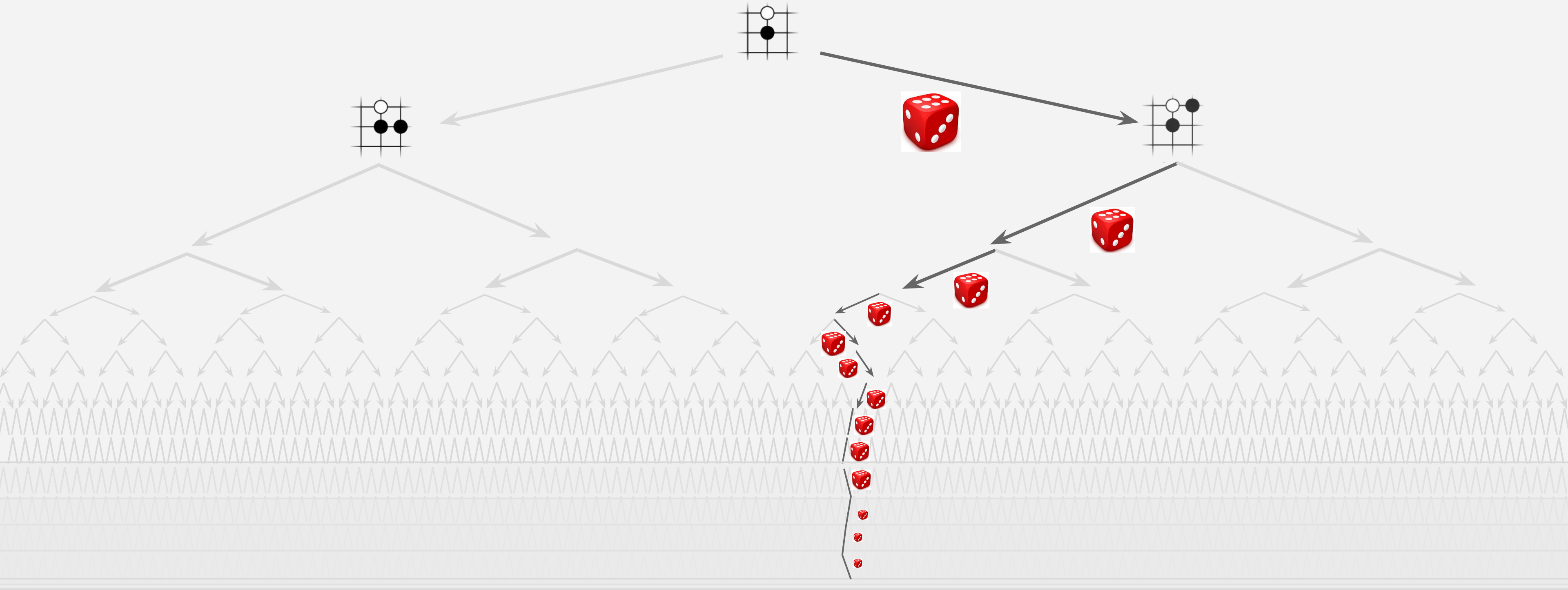


Policy network

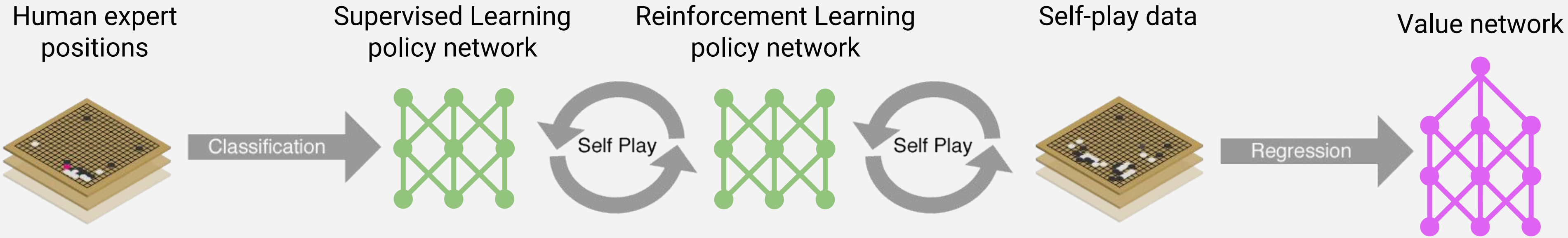




Monte-Carlo rollouts



Neural network training pipeline



Internal Testing	Calibration	External Testing
AlphaGo (May 2017)	*Wins 3/3 Matches* →	 Ke Jie (9p) World number 1
AlphaGo (Mar 2016)	*Wins 4/5 Matches* →	 Lee Sedol (9p) Top player of past decade
AlphaGo (Oct 2015)	*Wins 5/5 Matches* →	 Fan Hui (2p) 3-times reigning Euro Champion

Planning with learned models

Learning models

Motivation

- We discussed learning policies and values
- What about models?

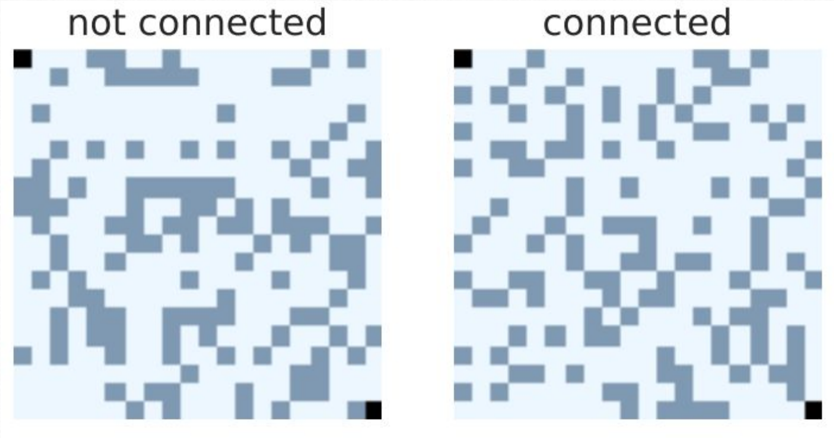
Learning models

Motivation

- We discussed learning policies and values
- What about models?
- Models would allow us to plan
  - Planning is useful in combinatorial and compositional domains
  - Trade off local compute to trying to store everything
  - Would allow us to use great planning algorithms

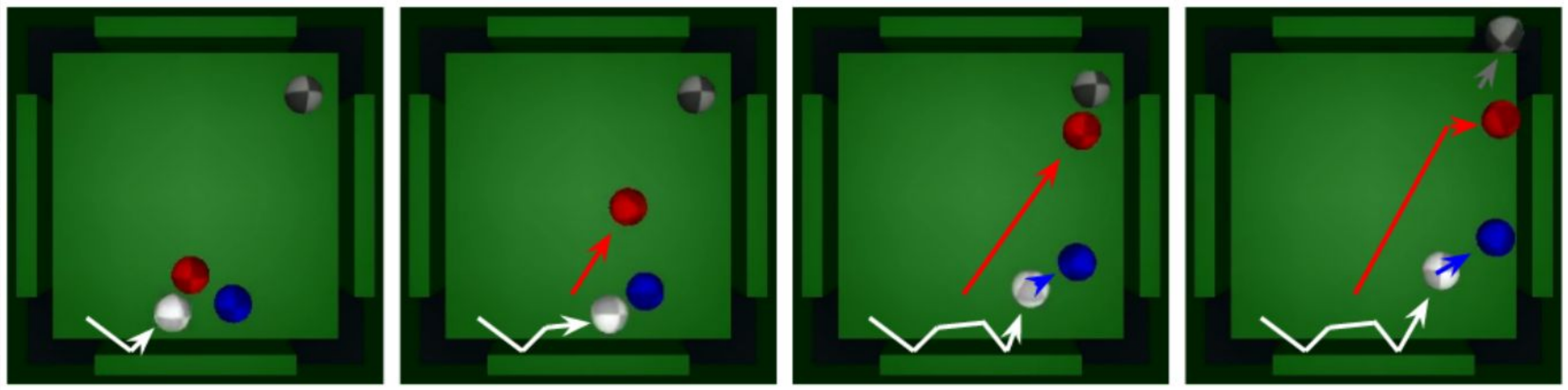
Example

Random Mazes



Example

Pool



Learning models

Complexities

- Learning models from raw inputs is hard
  - What should our model capture - pixels?
  - Objectives do not match: potentially focus on irrelevant details

Learning models

Complexities

- Learning models from raw inputs is hard
    - What should our model capture - pixels?
    - Objectives do not match: potentially focus on irrelevant details
- What to do with an imprecise model?
    - Many planning algorithms assume model is perfect

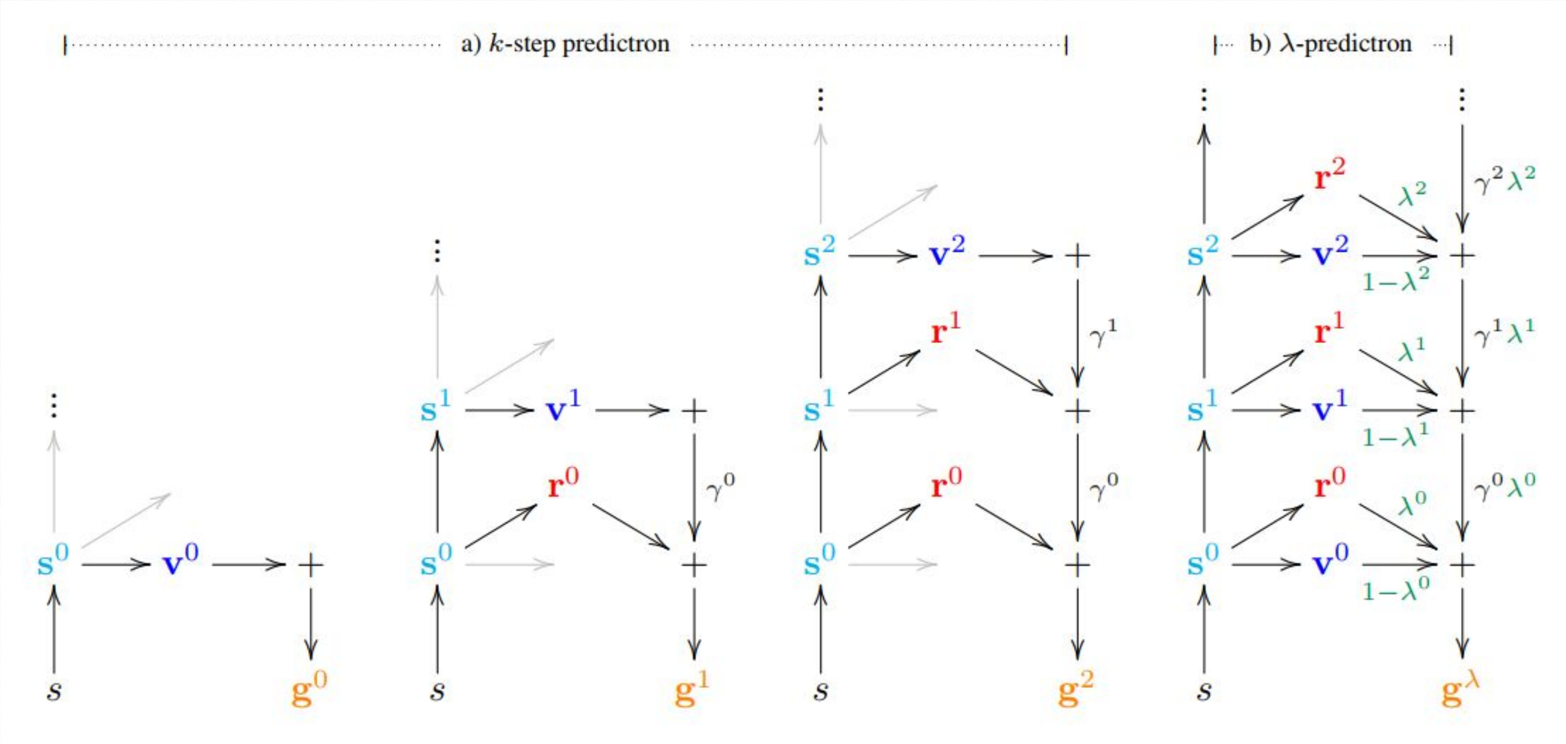
The Predictron

(Silver, van Hasselt, Hessel, Schaul, Guez, et al., 2016)

- Main idea: learn an **abstract model**
- The model should be **good for planning**
- But it does not have to match the real dynamics
    - See also "Value iteration networks" (Tamar et al., 2016)

The Predictron

(Silver, van Hasselt, Hessel, Schaul, Guez, et al., 2016)



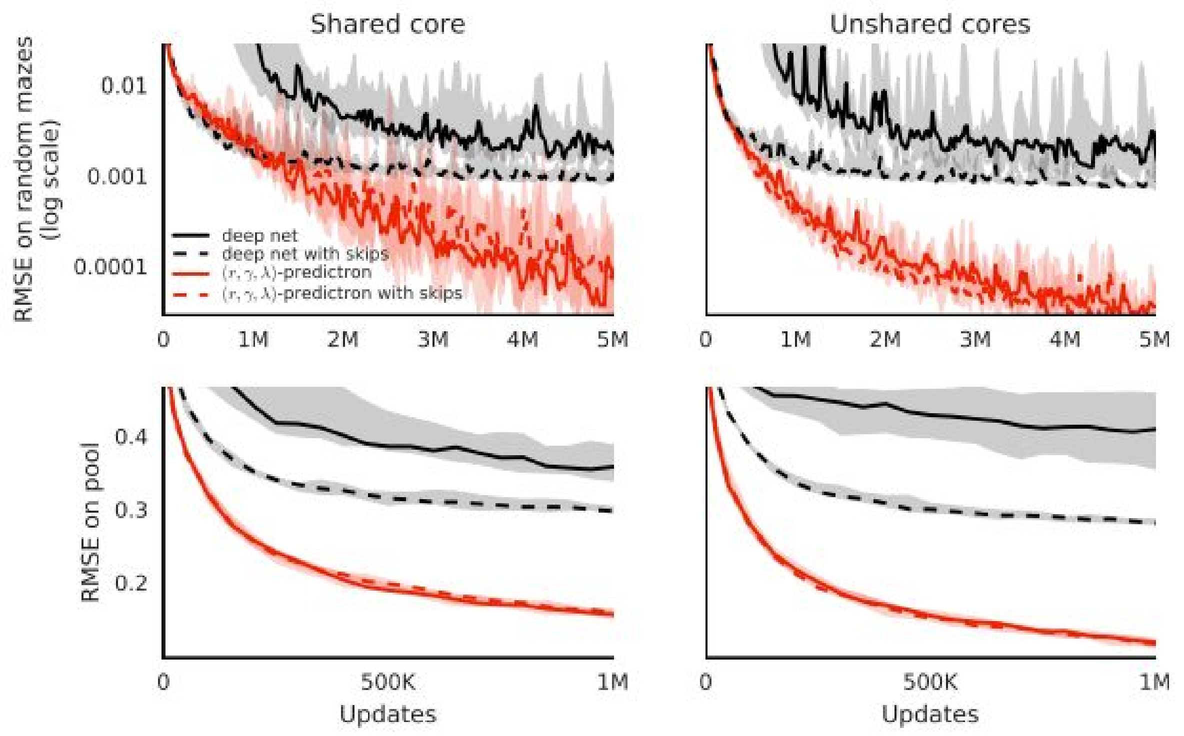
The Predictron

Learning abstract models

- Idea: compute looks like planning, but we do not have a separate model-learning objective
- Instead, the goal is to optimize the outcome of planning with the learnt model
- Then, learn all components end-to-end
- A model is learnt, because by construction a model exists
- But model-semantics (e.g., what does each state mean?) is not prefixed

The Predictron

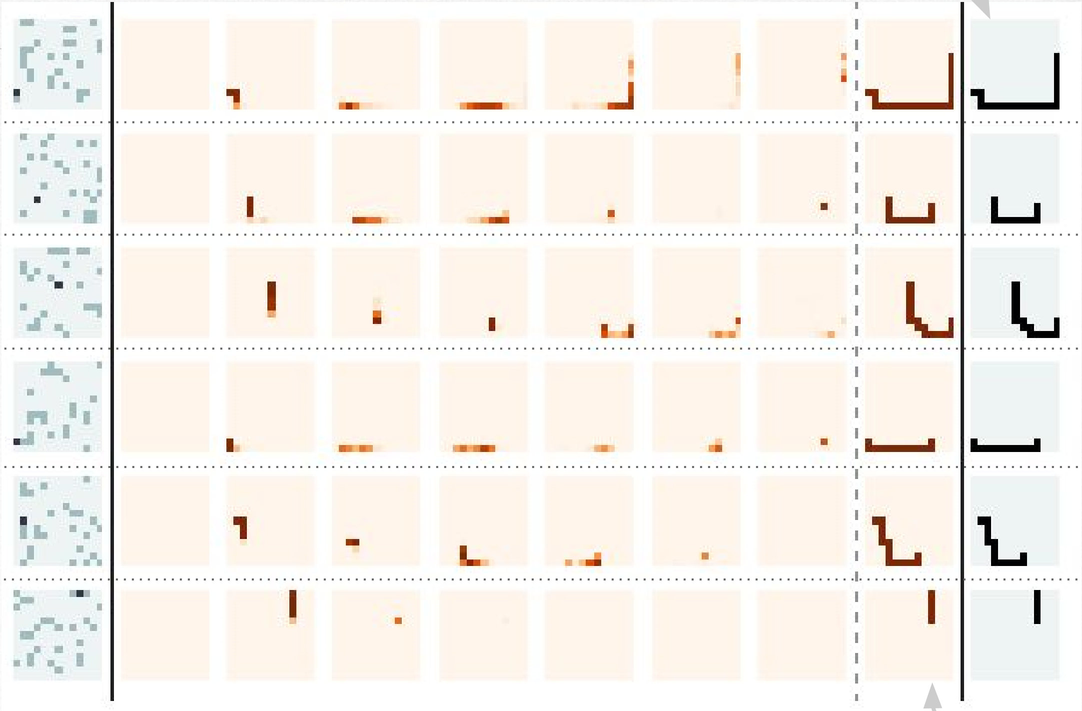
Learning abstract models



The Predictron

Trajectory prediction with the abstract model

- **Left:** Random maze +start position
- **Right:** Trajectory for some policy: this is the target
- **Middle:** Internal partial plans appear in the predictron representation
- Partial trajectories were **not** in the data
- Internal plans compose sequentially into full trajectories



THANK YOU