Policy Gradients + Planning

Rajbir Kataria, Zhizhong Li, and Tanmay Gupta

Background

- Action-value function using parameters $\boldsymbol{\theta}$

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- Policy was generated from the ***Q(s,a)***

$$\pi(s) = argmax_a Q(s,a)$$

- We will focus on parameterizing the policy directly:

$$\pi_\theta(s,a) = \mathbb{P}\left[a \mid s, \theta\right]$$

Overview

- Motivation
- Policy Gradients
  - REINFORCE
    - Simple Statistical Gradient-Following Algorithms for. Connectionist Reinforcement Learning
  - Actor-critic methods: REINFORCE + e.g. Q-learning
    - Asynchronous Advantage Actor-Critic (A3C)

- Model-based learning
  - Planning
    - Value Iteration Networks
- Applications
    - Recurrent Models of Visual Attention
    - End-to-end Learning of Action Detection from Frame Glimpses in Videos
    - Alpha-Go

Motivation: Iterated Rock-Paper-Scissors

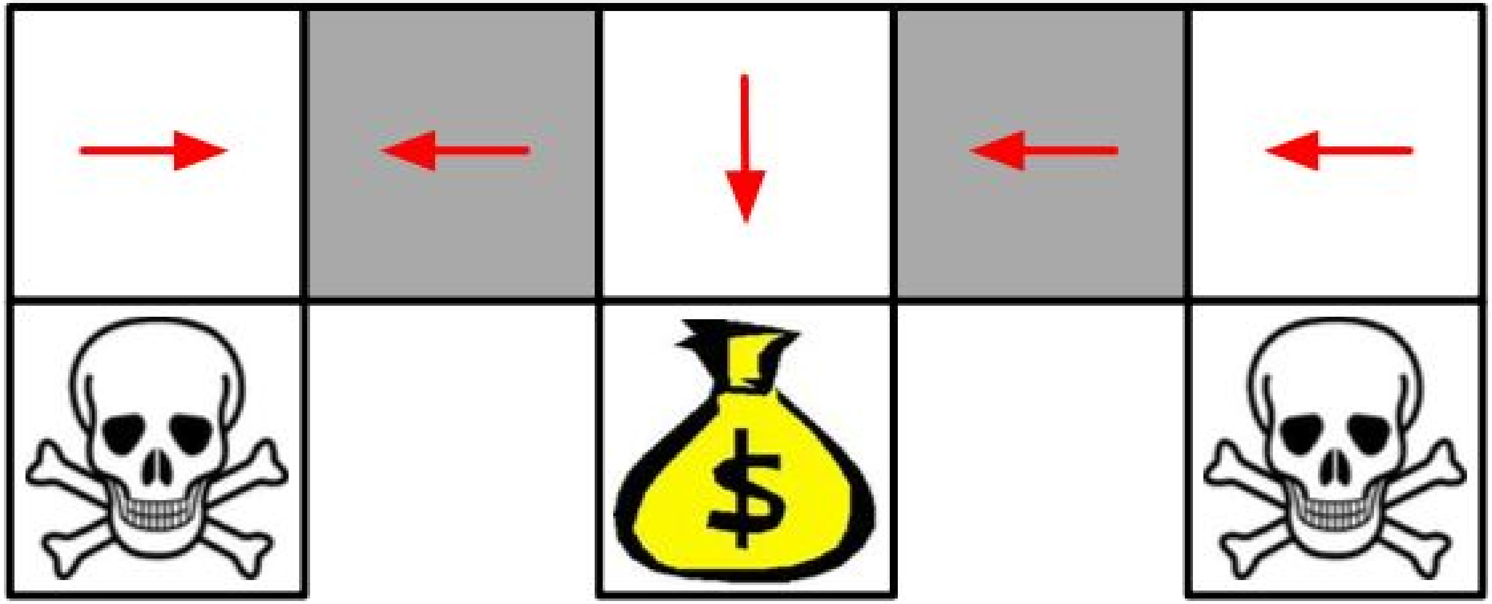
- Consider value-function based policies for iterated rock-paper-scissors

- Optimal Policy? **Random**

Slide from David Silver

Motivation: Aliased Gridworld

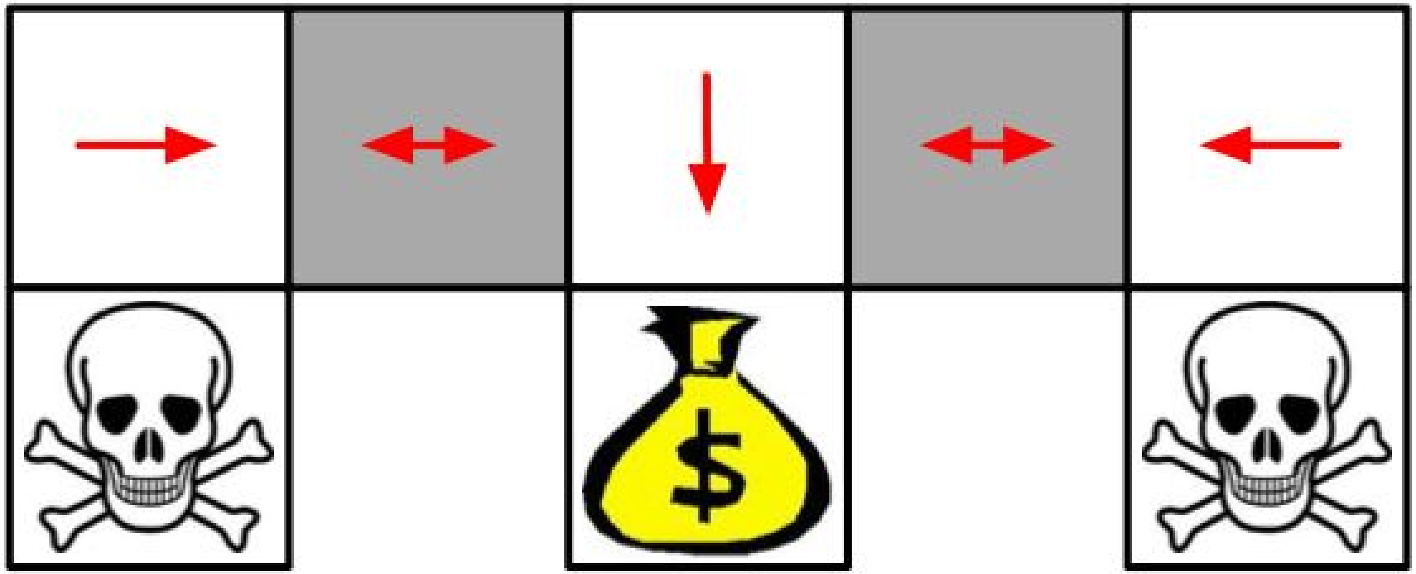
- The agent cannot distinguish the grey states



- Optimal deterministic policy?
  - Move **Left** in both grey states
  - Move **Right** in both grey states

Slide from David Silver

Motivation: Aliased Gridworld



- An optimal policy will randomly move E or W in grey states

$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$
$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- Policy-based RL can learn the optimal **stochastic policy**!

Slide from David Silver

Policy-Based RL

- **Advantages:**
  - **Can learn stochastic policies that are useful for POMDP environments**
  - Effective in high-dimensional or continuous action spaces
  - Better convergence properties

- **Disadvantages:**
  - Evaluating a policy is typically inefficient and high variance --- naive Monte Carlo sampling

Slide from David Silver

Policy Optimization

- Policy based reinforcement learning is an optimization problem
- Find **θ** that maximizes **J(θ)**
- Some approaches do not use gradient
  - **Hill climbing**

Slide from David Silver

Evolution Strategies - Hill Climbing

- At every iteration (“generation”)
  - Population of parameter vectors (“genotypes”) is perturbed (“mutated”)
  - Objective function value (“fitness”) is evaluated

- Highest scoring parameter vectors are then recombined to form the population for the next generation

- **Gradient Free!**

Salimans et al. Evolution Strategies as a Scalable Alternative to Reinforcement Learning arXiv:1703.03864v1

Evolution Strategies - Hill Climbing

- Highly parallelizable

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
    for each worker $i = 1, \ldots, n$ **do**
        Sample $\epsilon_i \sim \mathcal{N}(0, I)$
        Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
    end for
    Send all scalar returns F_i from each worker to every other worker
    for each worker $i = 1, \ldots, n$ **do**
        Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$
        Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
    end for
end for

Salimans et al. Evolution Strategies as a Scalable Alternative to Reinforcement Learning arXiv:1703.03864v1

Evolution Strategies - Results

Evolution Strategies as an Alternative for Reinforcement Learning

Game	DQN	A3C FF, 1 day	ES FF, 1 hour
Alien	570.2	182.1	994.0
Amidar	133.4	283.9	112.0
Assault	3332.3	3746.1	1673.9
Asterix	124.5	6723.0	1440.0
Asteroids	697.1	3009.4	1562.0
Atlantis	76108.0	772392.0	1267410.0
Bank Heist	176.3	946.0	225.0
Battle Zone	17560.0	11340.0	16600.0
Beam Rider	8672.4	13235.9	744.0
Berzerk	NaN	1433.4	686.0
Bowling	41.2	36.2	30.0
Boxing	25.8	33.7	49.8
Breakout	303.9	551.6	9.5
Centipede	3773.1	3306.5	7783.9
Chopper Command	3046.0	4669.0	3710.0
Crazy Climber	50992.0	101624.0	26430.0
Demon Attack	12835.2	84997.5	1166.5
Double Dunk	-21.6	0.1	0.2
Enduro	475.6	-82.2	95.0
Fishing Derby	-2.3	13.6	-49.0
Freeway	25.8	0.1	31.0
Frostbite	157.4	180.1	370.0
Gopher	2731.8	8442.8	582.0
Gravitar	216.5	269.5	805.0

Salimans et al. Evolution Strategies as a Scalable Alternative to Reinforcement Learning arXiv:1703.03864v1

Policy Optimization

- Policy based reinforcement learning is an optimization problem

- Find **θ** that maximizes **$J(\theta)$**

- Some approaches do not use gradient
  - **Hill climbing**
  - Genetic algorithms

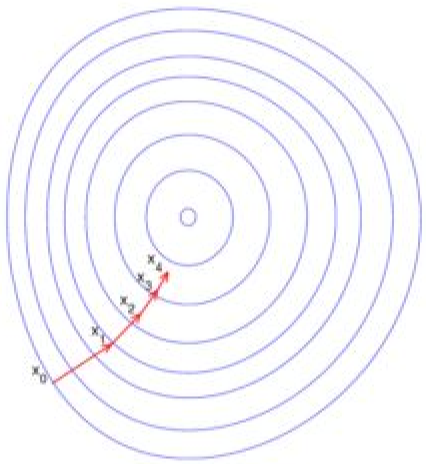
- Greater efficiency often possible using gradient
  - **Gradient Descent**
  - Quasi-newton

- From now on, we focus primarily on **Gradient Descent**

Slide from David Silver

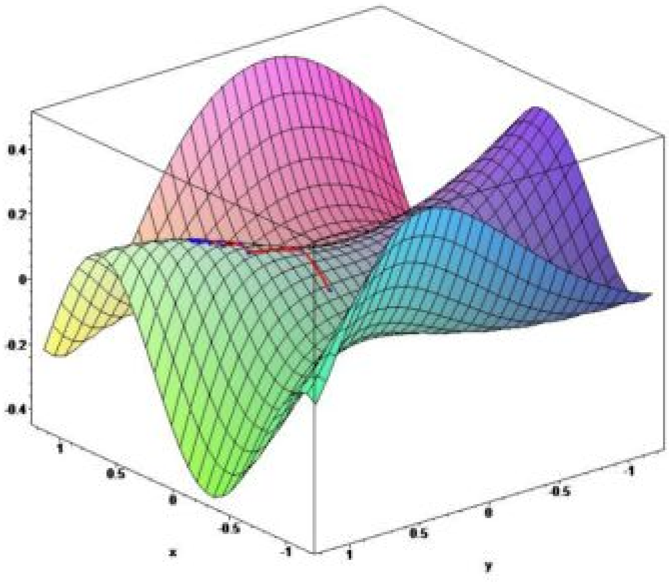
Policy Gradient

- Let $J(\theta)$ be any policy objective function



- Policy gradient algorithms search for a local maximum in $J(\theta)$

$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$



- Where $\nabla_\theta J(\theta)$ is the policy gradient
    - α is a step-size parameter

Slide from David Silver

Policy Gradient Theorem

$$\boxed{J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]}$$

$$\begin{aligned}
J(\theta) &= E_{\pi_\theta}[R] \\
\nabla_\theta J(\theta) &= \nabla_\theta E_{\pi_\theta}[R] \\
&= \nabla_\theta \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) R^{\pi_\theta}(s,a) \\
&= \sum_{s \in S} d(s) \sum_{a \in A} \nabla_\theta \pi_\theta(a|s) R^{\pi_\theta}(s,a) \\
&= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) \nabla_\theta log \pi_\theta(a|s) R^{\pi_\theta}(s,a) \\
&= E_{\pi_\theta}[\nabla_\theta log \pi_\theta(a|s) R^{\pi_\theta}(s,a)]
\end{aligned}$$

$$\boxed{\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log \pi_\theta(a|s) R^{\pi_\theta}(s,a)]}$$

REINFORCE

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)R^{\pi_\theta}(s,a)]$$

- Maximizing J is non-trivial
  - Expectation over high-dimensional action sequences

function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T-1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t)\ R$
    end for
  end for
  return θ
end function

$$\nabla_\theta J = \sum_{t=1}^{T} \mathbb{E}_{p(s_{1:T};\theta)}\left[\nabla_\theta \log \pi(u_t|s_{1:t};\theta)R\right] \approx \frac{1}{M}\sum_{i=1}^{M}\sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i|s_{1:t}^i;\theta)R^i$$

Williams et al. Simple Statistical Gradient-Following Algorithms for. Connectionist Reinforcement Learning. Machine Learning, 8(3):229-256, 1992

Connection with value learning: Actor-critic methods

Motivation: PG vs value functions

- Q-learning: learns $Q(s,a)$ (action-value function)

- PG: directly learns policy $\pi(s,a)$
  - Pro:

    Better convergence

    Can learn stochastic policy

    Get action directly; compact
  - Con:

    suffers from high variance when training

Motivation: PG vs value functions

- Q-learning: learns $Q(s,a)$ (action-value function)
- PG: directly learns policy $\pi(s,a)$
  - Con: suffers from high variance when training

... reduce variance?

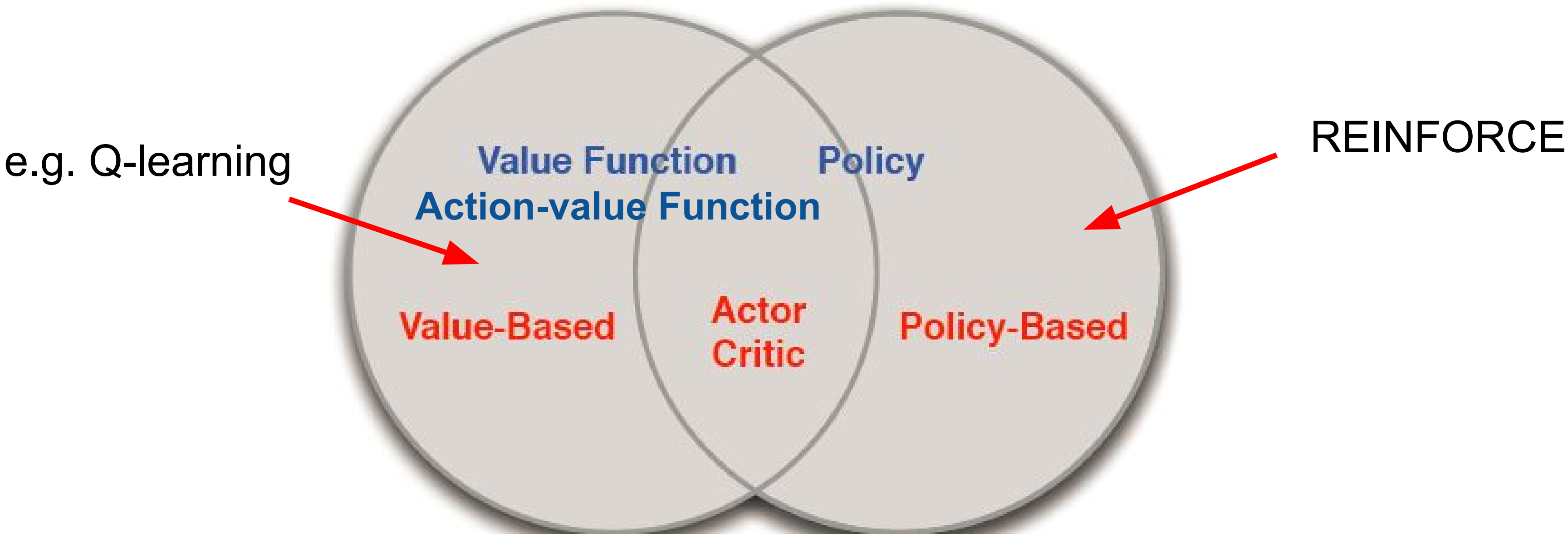


Figure from David Silver

Method outline

- Use Q to reduce variance
  - Recall gradient descent in PG:

Future return from experience (real world samples)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, R\right] \quad \text{REINFORCE}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, Q\;(s,a)\right] \quad \text{Q Actor-Critic}$$

Also future return! Learned along with π

  - Many equivalent forms

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, A\;(s,a)\right] \quad \text{Advantage Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, \delta\right] \quad \text{TD Actor-Critic}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, \delta e\right] \quad \text{TD}(\lambda)\ \text{Actor-Critic}$$

$$G_\theta^{-1}\nabla_\theta J(\theta) = w \quad \text{Natural Actor-Critic}$$

Slide from David Silver

Example (1): A3C

Asynchronous Advantage Actor-Critic

- Bias from Q actor-critic

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q\ (s,a)\right] \qquad \text{Q Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A\ (s,a)\right] \qquad \text{Advantage Actor-Critic}$$

  - Encourages action if $Q(s,a)$ is large
  - Should encourage good action on state
    (not just random actions that happen on good state)

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Example (1): A3C

Asynchronous Advantage Actor-Critic

- Advantage actor-critic

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q\ (s,a)\right] \quad \text{Q Actor-Critic}$$
$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A\ (s,a)\right] \quad \text{Advantage Actor-Critic}$$

  - Only counts the advantage (return minus baseline)

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

(in practice) $= r + \gamma V\ (s') - V\ (s)$

Encourage doing better than "baseline"

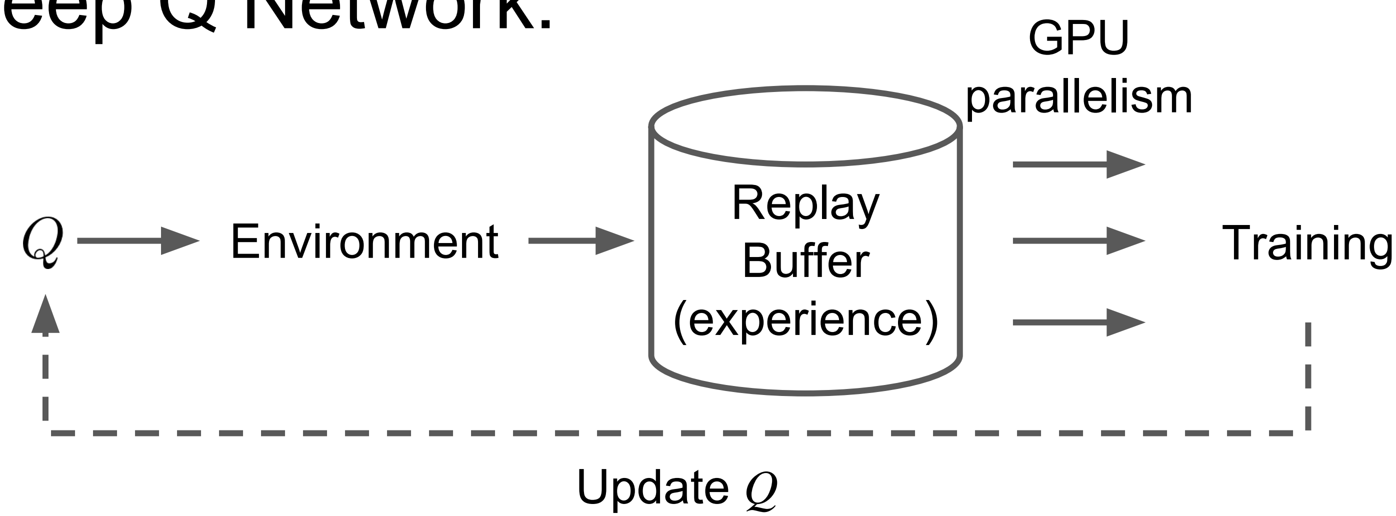
  - Reduces variance
  - Learn Q_w , V_v normally; learn π by replacing R with A.

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Example (1): A3C

Asynchronous Advantage Actor-Critic

- Deep Q Network:



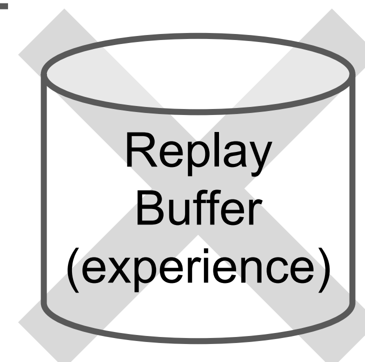
  - (to reduce correlation in training data -- crucial for DQN)
- Experience from past policy
  - Applies to off-policy learning only
  - Cannot apply to e.g. actor-critic!

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Example (1): A3C

Asynchronous Advantage Actor-Critic

- Asynchronous RL:



  - (Also reduces correlation in training data!)

- Experience is on-policy

* cf. T. Salimans et al. Evolution Strategies as a Scalable Alternative to RL

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Example (1): A3C

- Implementation details
  - Use k-step estimate of advantage

$$A(s_t, a_t) = \underbrace{\sum_{i=0}^{k-1} \gamma^i R_{t+i} + \gamma^k V(s_{t+k})}_{\text{Estimate @ future time step}} - \underbrace{V(s_t)}_{\text{Baseline return}}$$

Reward obtained

Estimate @ future time step

Baseline return

  - Actor/critic share some layers
  - Entropy regularization
  - Asynchronous RMSProp

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Example (1): A3C

Playing racing simulator *TORCS*

Example (1): A3C

- Results on Atari games (averaged)

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	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%

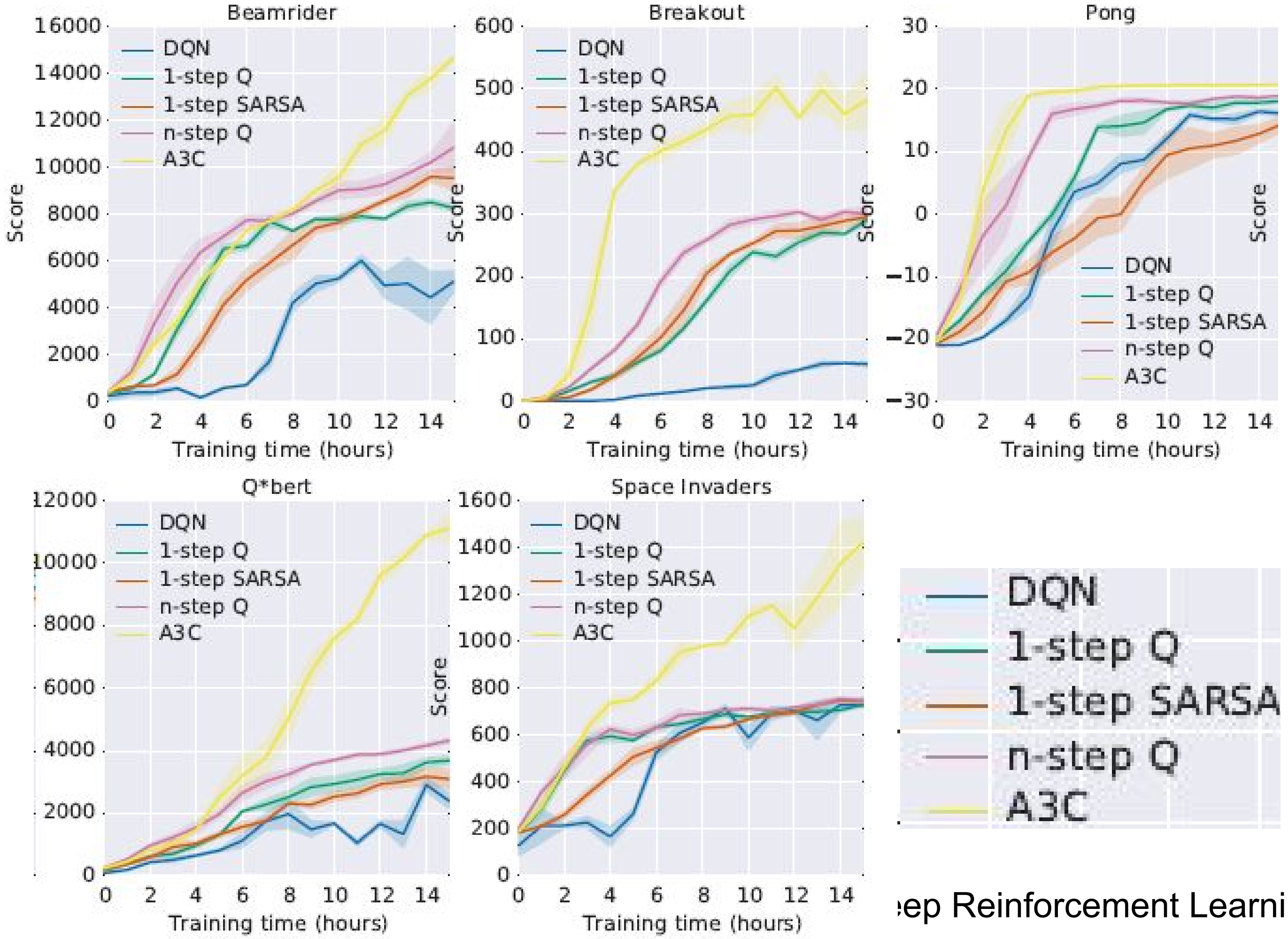
Human normalized scores

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Example (1): A3C

- Results

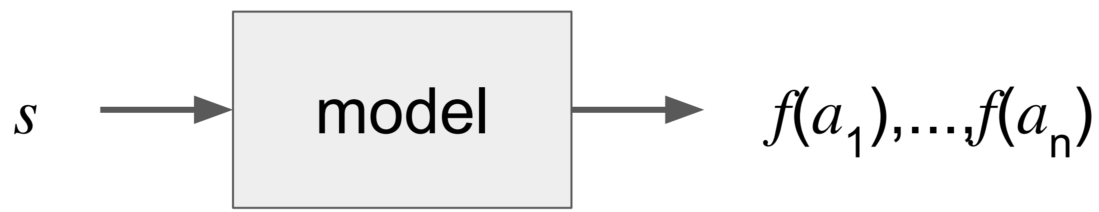
Score w.r.t. Training time (hrs).
Note: hyperparameter fiddling may be at play



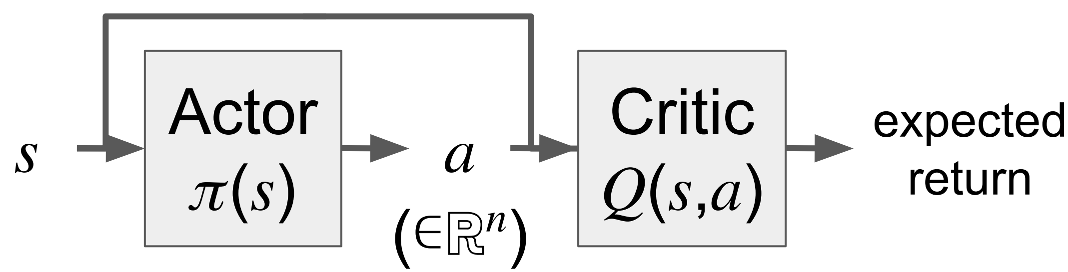
ep Reinforcement Learning. ICML 2016

Example (2): Continuous control

Before: model $Q(s,a)$ or $\pi(s,a)$ by enumerating a



- When a is continuous...
  - Actor-critic!



$$J = \mathbb{E}\left[Q(s, \pi(s))\right]$$

$$\frac{\partial J}{\partial \theta_\pi} \approx \mathbb{E}\left[\frac{\partial Q}{\partial a}\frac{\partial \pi}{\partial \theta_\pi}\right]$$

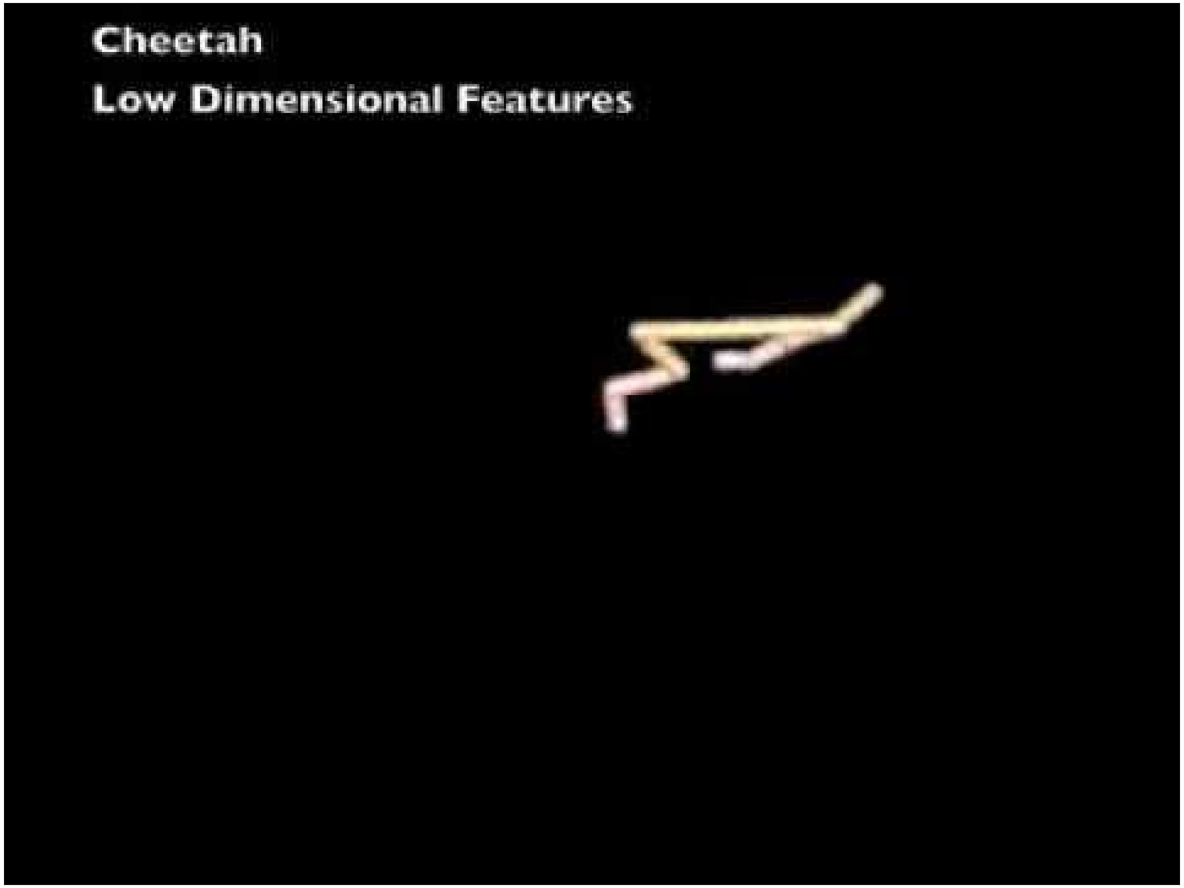
Update π
to maximize Q

Fit Q normally

Lillicrap et al. Continuous Control with Deep Reinforcement Learning. ICLR 2016

Example (2): Continuous control

Simulated control tasks



Planning

The story so far

- Model-free RL
  - ***Q-Learning / Sarsa*:**

    Learn action-value function **directly from experience**

  - ***Policy Gradient:***

    Learn policy **directly from experience**

The story so far

- Model-based RL
    - Learn a **model** of the environment
    - Use the model to learn policy/value function

Planning

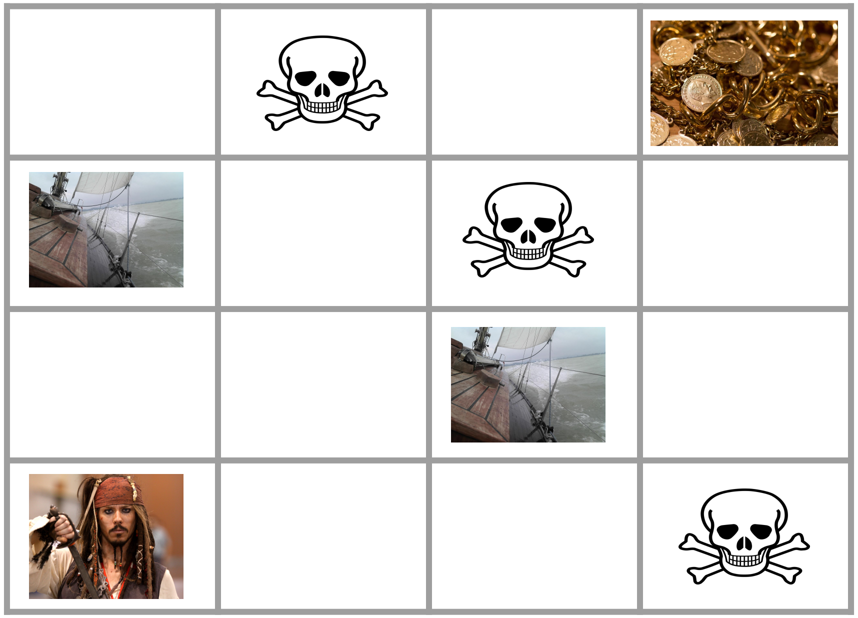
Why Plan?

- Simulation cheaper than real interaction
- Speed up learning
- Generalize to new environments
- Predict a future even

Why Plan?

- Simulation cheaper than real interaction
  - **Planning based Q-Learning**
- Speed up learning
  - **Dyna-Q**
- Generalize to new environments
  - **Value Iteration Networks**
- Predict a future even
  - **The Predictron**

Simplest Model-based RL

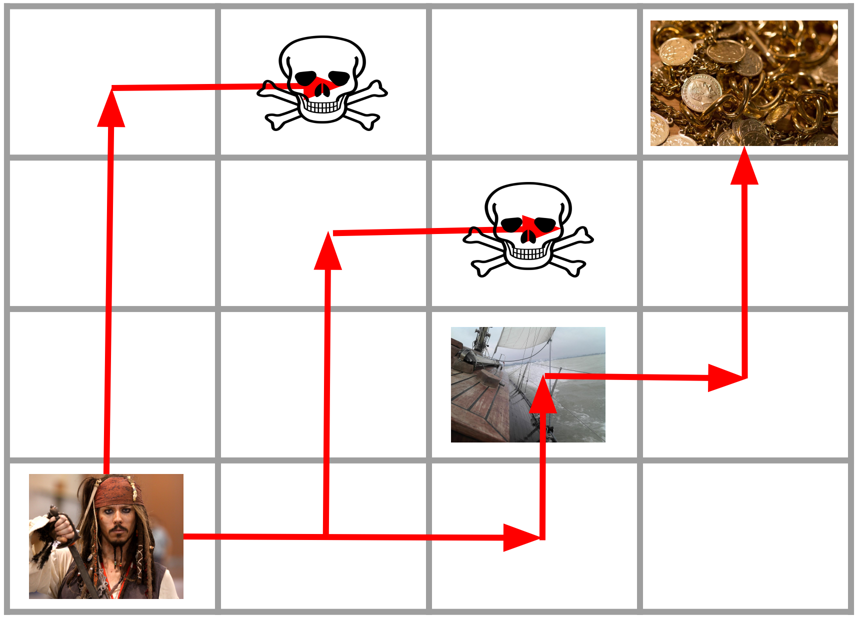


Simplest Model-based RL

MDP with Unknown

- Rewards $R(s)$
- Transition Probabilities $P(s'|s, a)$

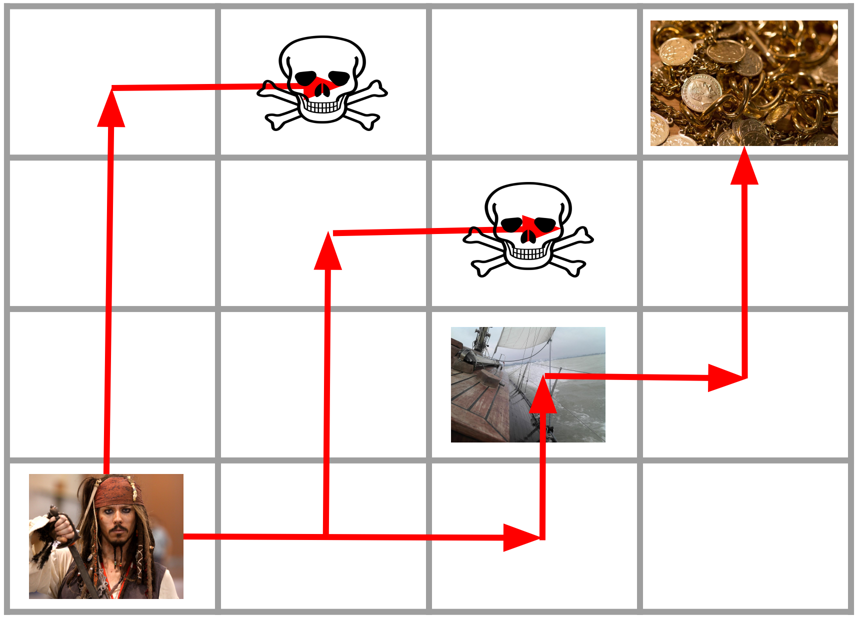
Simplest Model-based RL



Solution:

- Gain experience $\{s_1, r_1, a_1, s_2, r_2, a_2, \cdots\}$

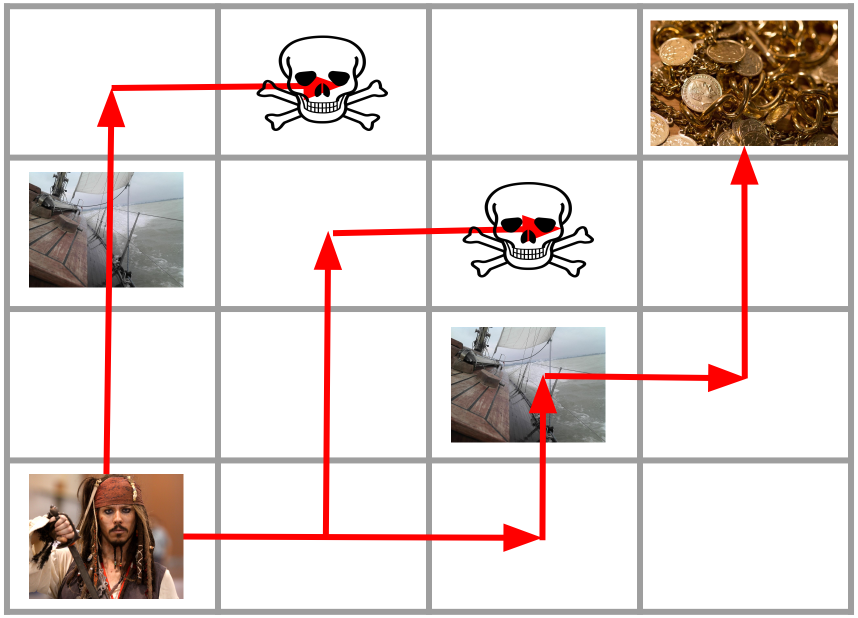
Simplest Model-based RL



Solution:

- Gain experience $\{s_1, r_1, a_1, s_2, r_2, a_2, \cdots\}$
- Estimate model $R(s) = \frac{1}{N(s)} \sum_t r_t \mathbf{1}[s_t = s]$

Simplest Model-based RL



Solution:

- Gain experience $\{s_1, r_1, a_1, s_2, r_2, a_2, \cdots\}$
- Estimate model $R(s) = \frac{1}{N(s)} \sum_t r_t \mathbf{1}[s_t = s]$

$$P(s'|s,a) = \frac{1}{N(s,a)} \sum_t \mathbf{1}[(s_t, a_t, s_{t+1}) = (s, a, s')]$$

Simplest Model-based RL

Use the estimated MDP to get optimal policy/value function

- Value Iteration
- Policy Iteration

$$V^*(s) = R(s) + \max_a \sum_{s'} P(s'|s,a)V^*(s')$$
$$\pi^*(s) = \mathrm{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

Sampling-based Planning with Q Learning

Given: An estimated MDP

Algorithm:

1. Randomly sample a state and action, (s_t, a_t)
2. Sample $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
3. Update Q function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R(s) + \max_a Q(s_{t+1}, a') - Q(s_t, a_t)]$$

4. Repeat

Learning from **simulated** experience

What if the model is incorrect?

Dyna-Q

Learning from both **real** and **simulated** experience

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

(a) $S \leftarrow$ current (nonterminal) state

(b) $A \leftarrow \varepsilon$-greedy(S, Q)

(c) Execute action A; observe resultant reward, R, and state, S'

(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S,A)\right]$

(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)

(f) Repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

$R, S' \leftarrow Model(S,A)$

$Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S,A)\right]$

D. Silver. RL course Lecture 8 http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

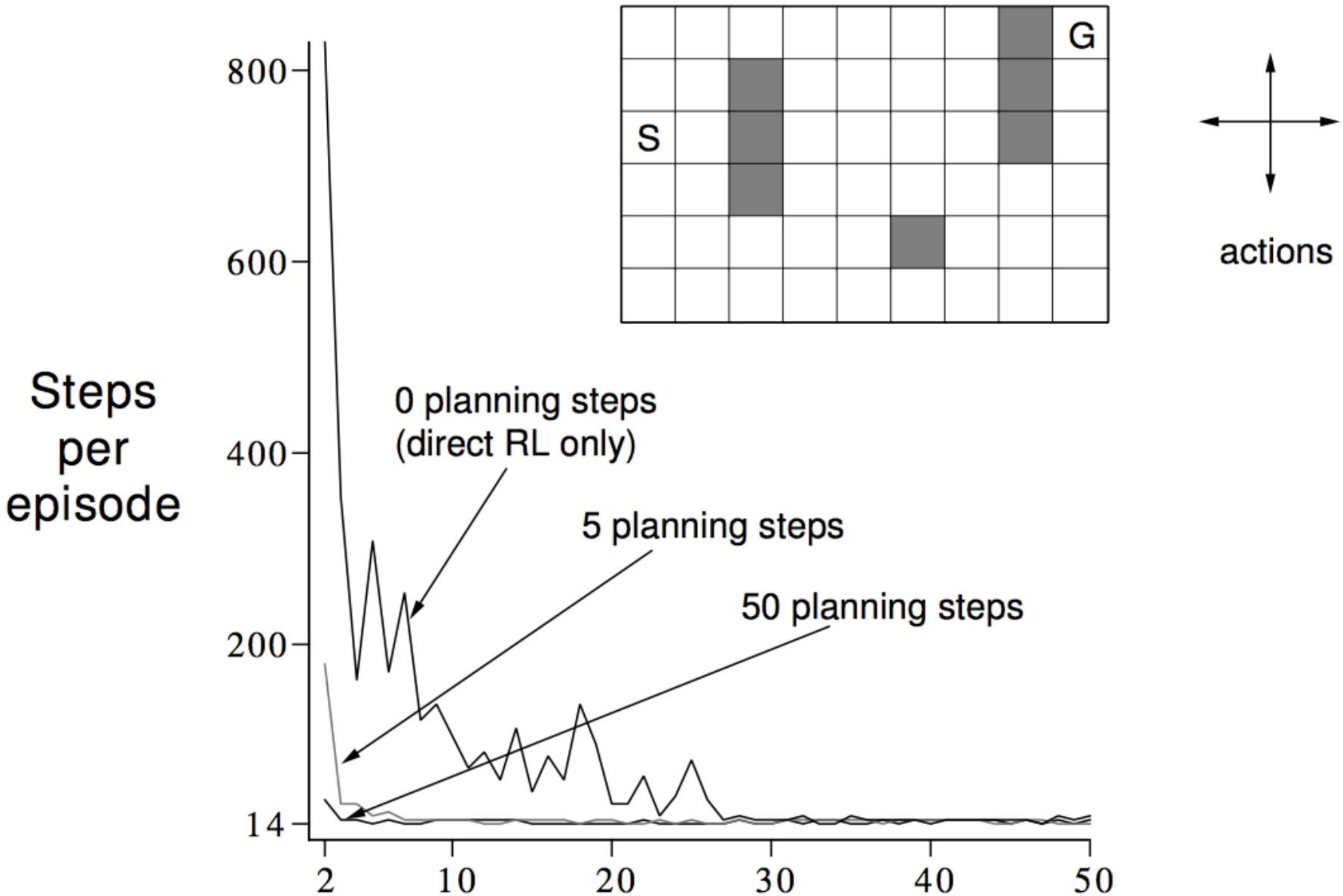
Dyna-Q

Learning from both **real** and **simulated** experience

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon\text{-greedy}(S,Q)$
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$

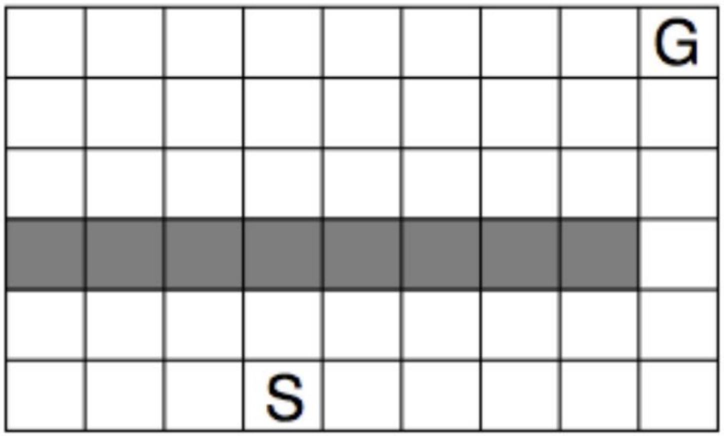
D. Silver. RL course Lecture 8 http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

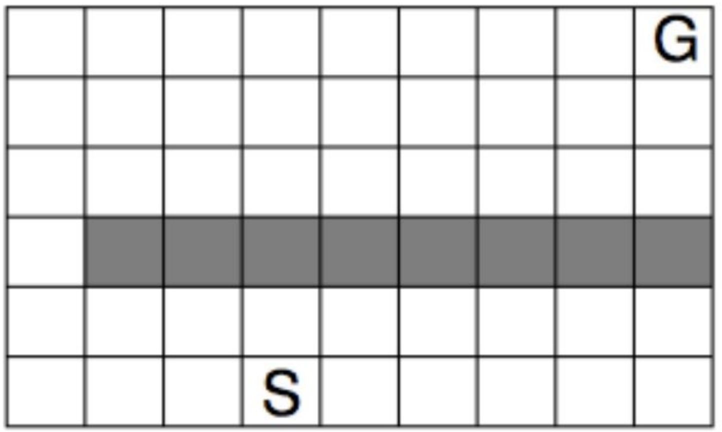
Dyna-Q



D. Silver. RL course Lecture 8 http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Generalization to novel environments





Learn Optimal
Policy / Value Function

Learn Optimal
Policy / Value Function

D. Silver. RL course Lecture 8 http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Generalization to novel environments

Policies trained using traditional CNNs are **Reactive**

$$\phi(s) \xrightarrow{\text{Q Network / Policy Network}} \pi_{re}(a|\phi(s))$$

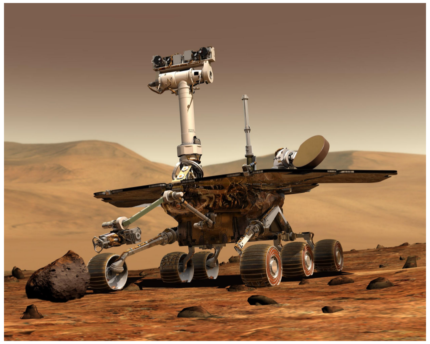
State Representation

Policy

Learning to **React**

vs

Learning to **Plan**

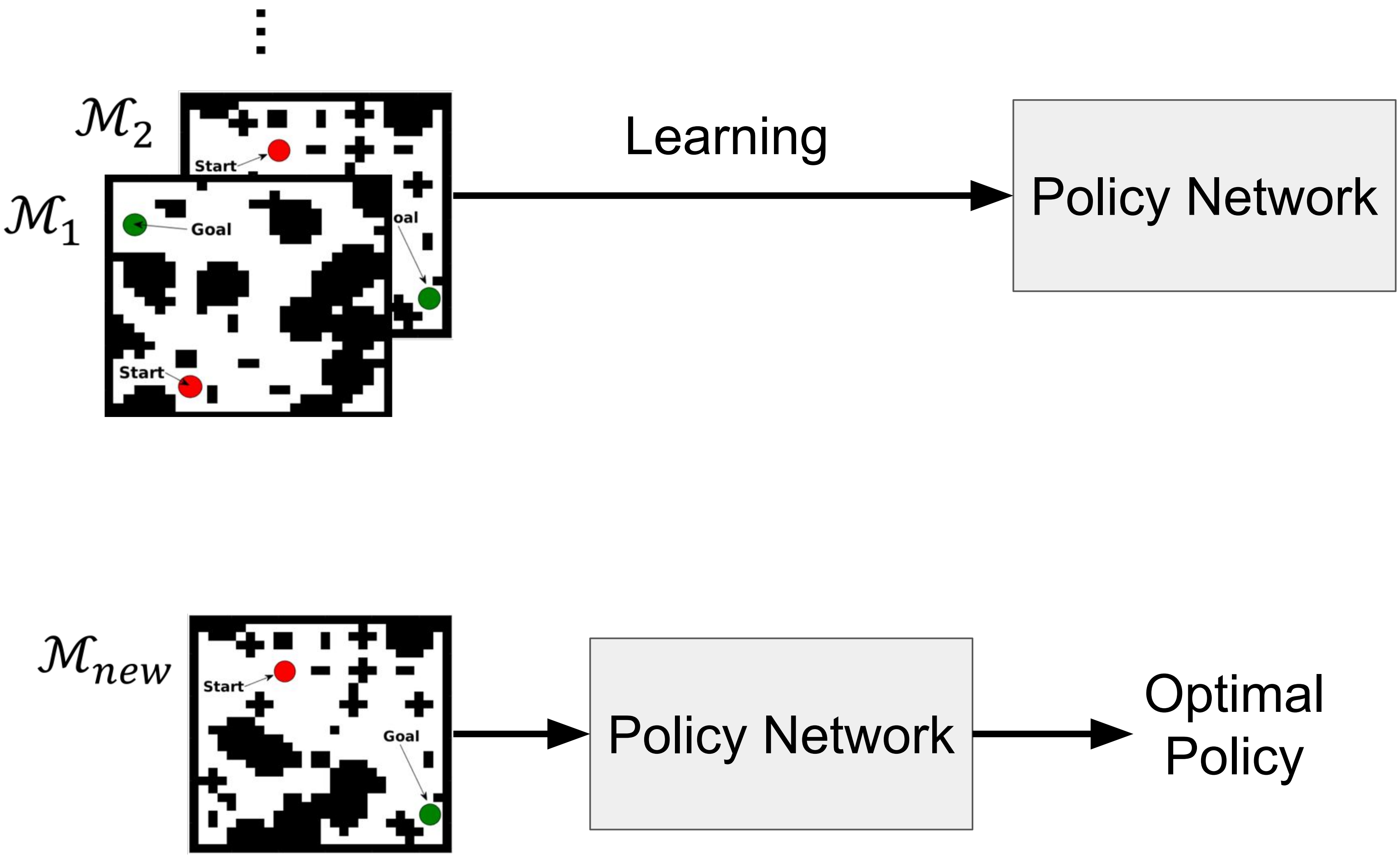


Value Iteration Network

Best Paper NIPS 2016

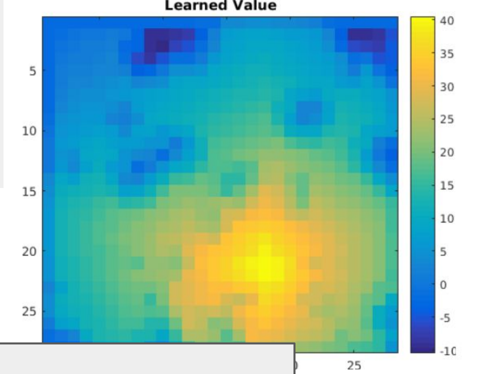
Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Value Iteration Networks



Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Value Iteration Networks

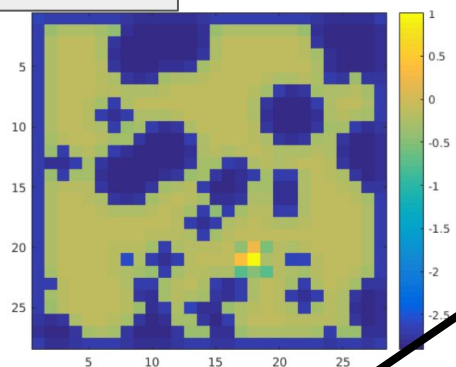


(Estimate of the real new M)

MDP $\bar{\mathcal{M}}$

Planning!

Solve using Value Iteration $\bar{V}^*$

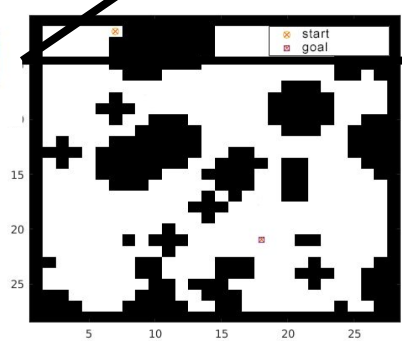


& transition

Select relevant information (Attention)

$\phi(s)$

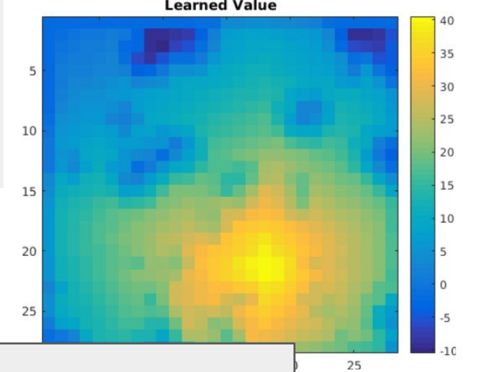
"State ovservation"



$\pi_{re}(a|\phi(s))$

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

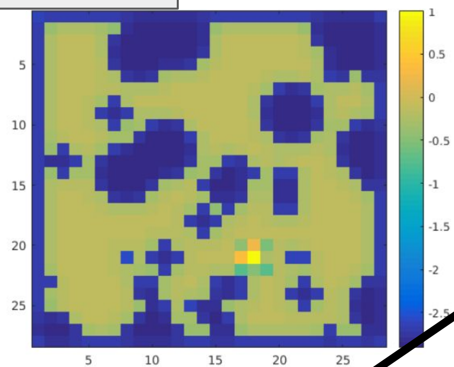
Value Iteration Networks



MDP $\bar{\mathcal{M}}$

Planning!

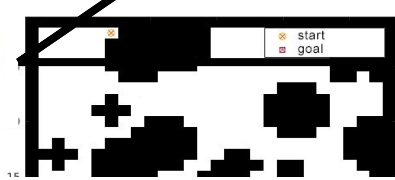
Solve using Value Iteration $\bar{V}^*$



& transition

Select relevant information (Attention)

$\phi(s)$



$$\pi_{re}(a|\phi(s), \bar{V}^*)$$

Make it **End-to-End Differentiable**

Questions?

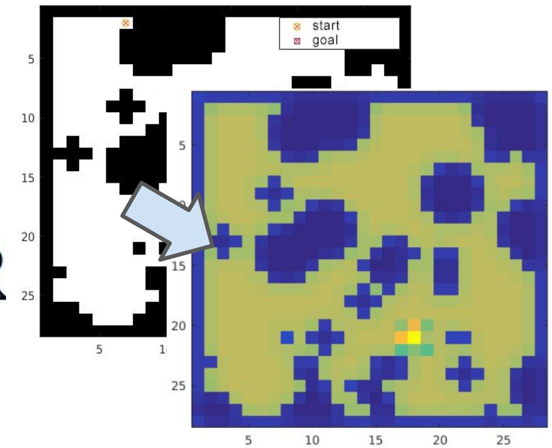
'orks." NIPS. 2016.

Value Iteration Networks

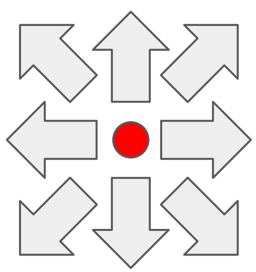
MDP $\bar{\mathcal{M}}$

Rewards:

$$f_R(\phi(s);\theta_R) \in \mathbb{R}$$



Transition probabilities (*same for all maps!*):



$\phi(s)$

$$\text{Let } s' = s + \Delta s$$

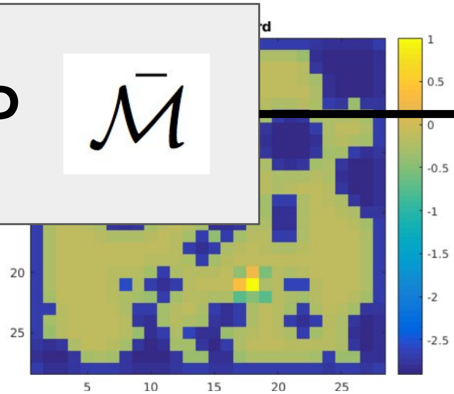
$$P(s'|s,a) = \quad P(\Delta s|a) \qquad \text{if } s' \in nbr(s)$$

$$P(s'|s,a) = \qquad 0 \qquad \text{if } s' \notin nbr(s)$$

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

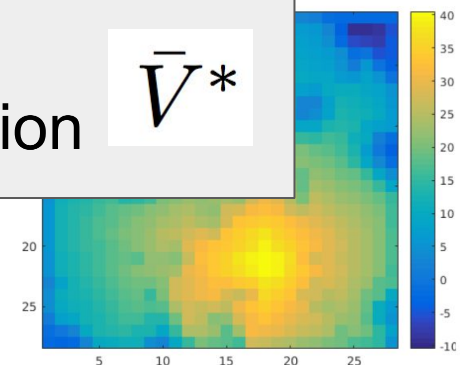
Value Iteration Networks

MDP $\bar{\mathcal{M}}$



Planning!

Solve using Value Iteration $\bar{V}^*$



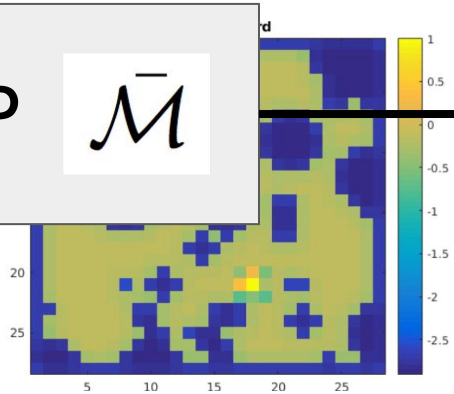
$$Q(s,a) = R(s) + \sum_{s'} P(s'|s,a)V(s')$$

$$V(s) = \max_a Q(s,a)$$

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

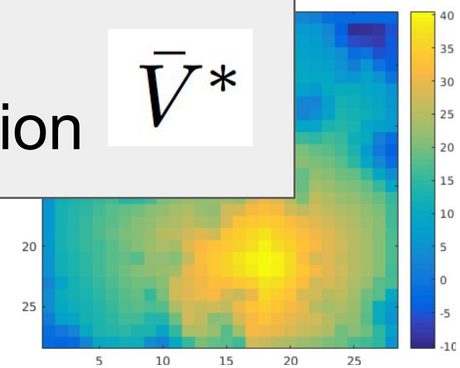
Value Iteration Networks

MDP $\bar{\mathcal{M}}$



Planning!

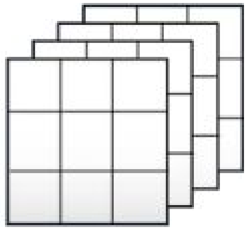
Solve using Value Iteration $\bar{V}^*$



$$Q(s,a) = R(s) + \boxed{\sum_{s' \in nbr(s)} \boxed{P(\Delta s | a)} V(s')}$$

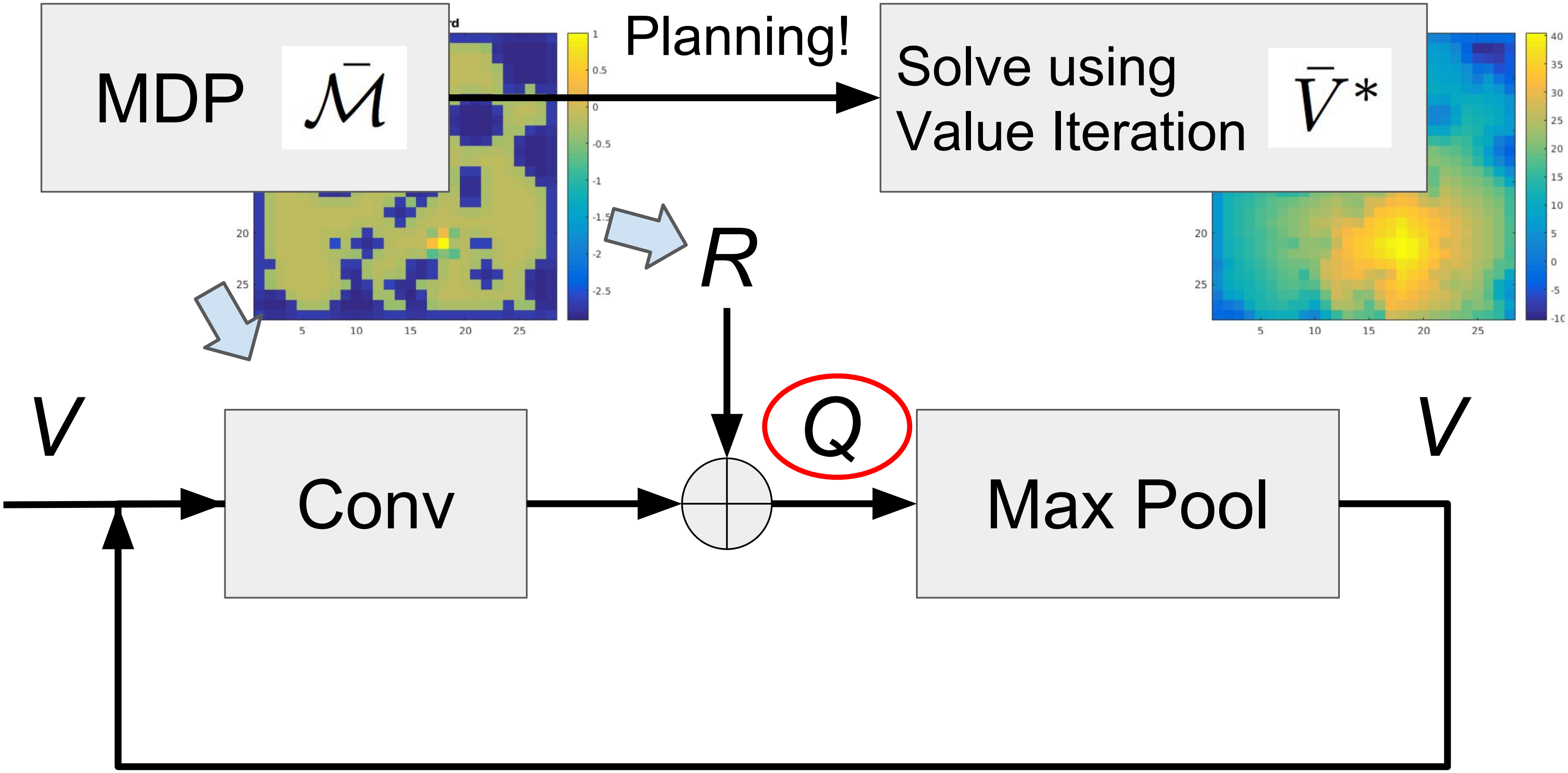
$$V(s) = \max_a Q(s,a)$$

Conv Kernel $\bar{P}$



Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Value Iteration Networks



R: m×n×1
V: m×n×1
Q: m×n×a
Conv: 3×3×a

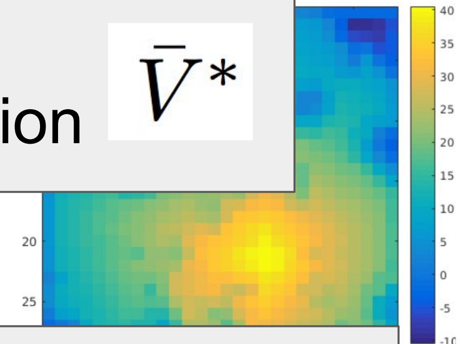
Questions?

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

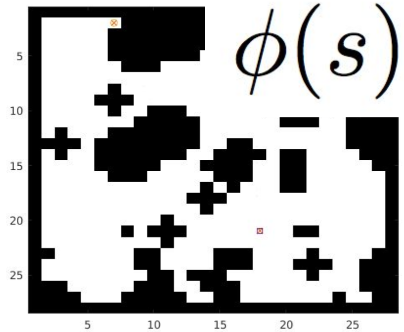
Value Iteration Networks

Attention:
Select $\bar{Q}^*(s, \cdot) \in \mathbb{R}^{|\mathcal{A}|}$

Solve using Value Iteration $\bar{V}^*$



Select relevant information (Attention)



$\phi(s)$

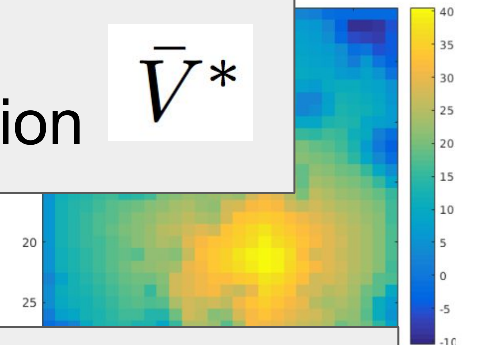
$$\pi_{re}(a|\phi(s), \bar{V}^*)$$

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

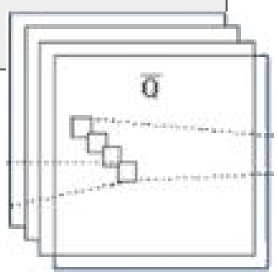
Value Iteration Networks

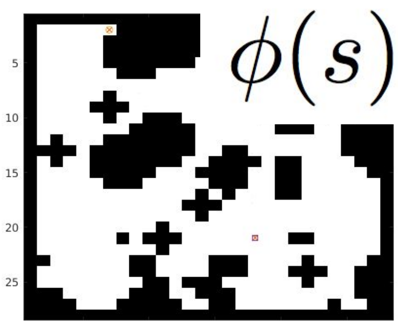
Attention:
Select $\bar{Q}^*(s,\cdot) \in \mathbb{R}^{|\mathcal{A}|}$

Solve using Value Iteration $\bar{V}^*$



Select relevant information (Attention)





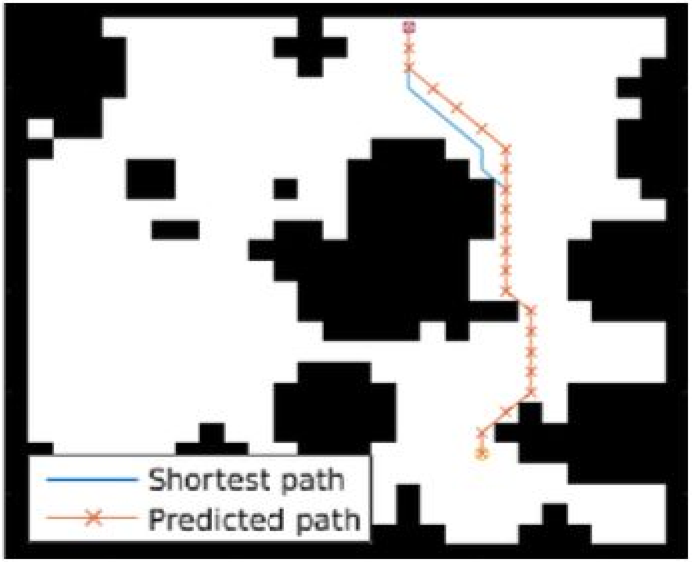
$\phi(s)$

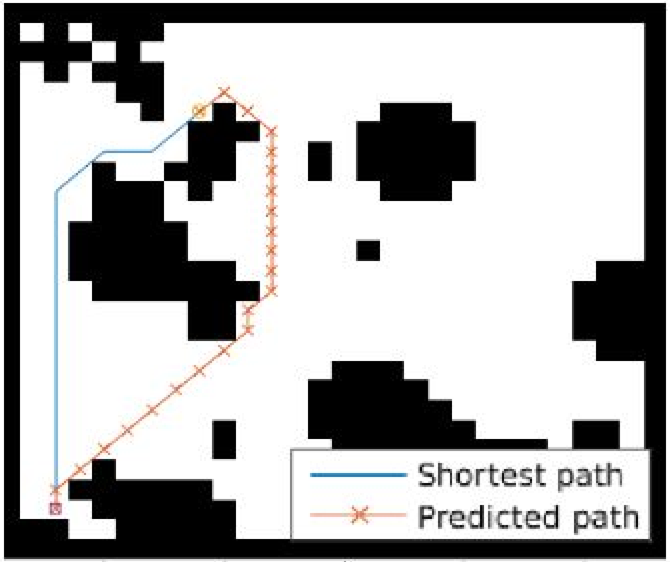
$\pi_{re}(a|\phi(s), \bar{Q}^*(s,\cdot))$

Questions?

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Grid World Experiment



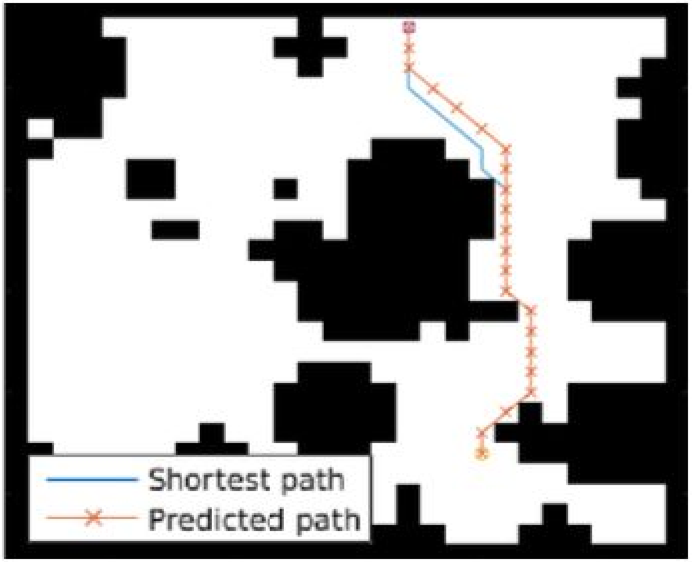


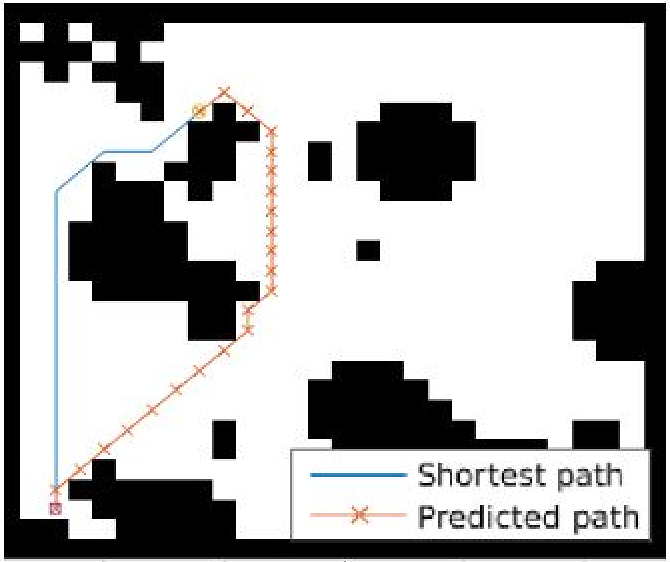
Success Rate	VIN	CNN	FCN
8x8	99.6%	97.9%	97.3%
16x16	99.3%	87.6%	88.3%
28x28	97%	74.2%	76.6%

(DQN)

(dense pixelwise classification)

Grid World Experiment

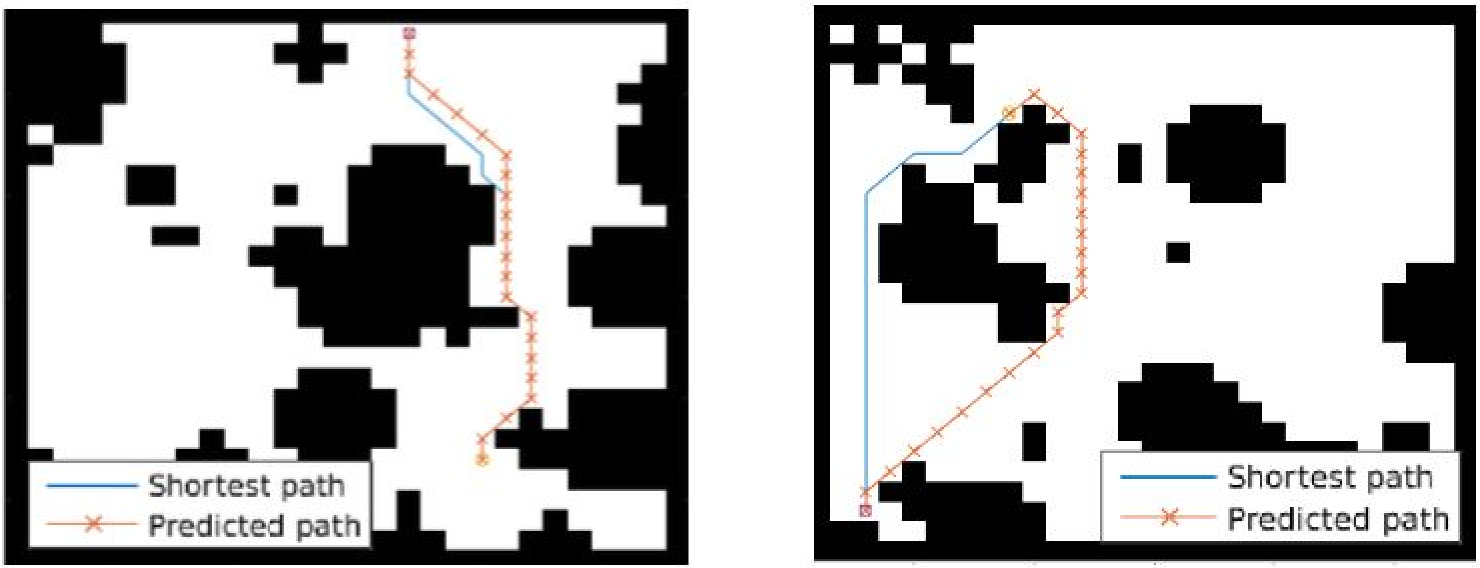




Success Rate	VIN	CNN	FCN
8x8	99.6%	97.9%	97.3%
16x16	99.3%	87.6%	88.3%
28x28	97%	74.2%	76.6%

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

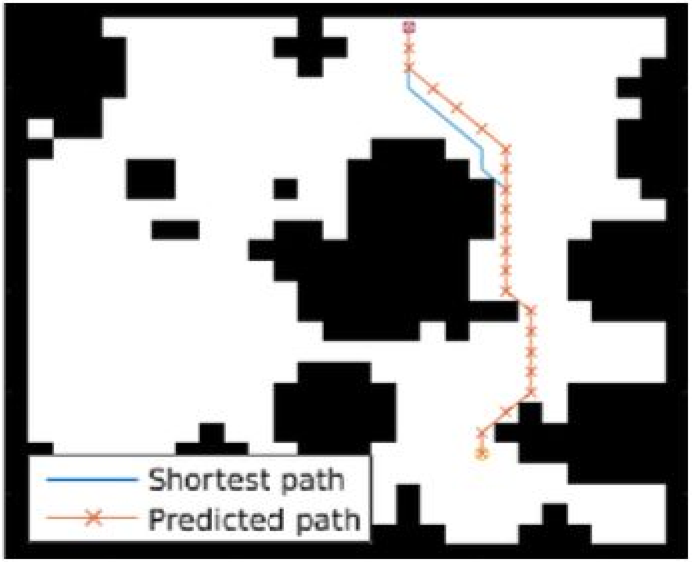
Grid World Experiment

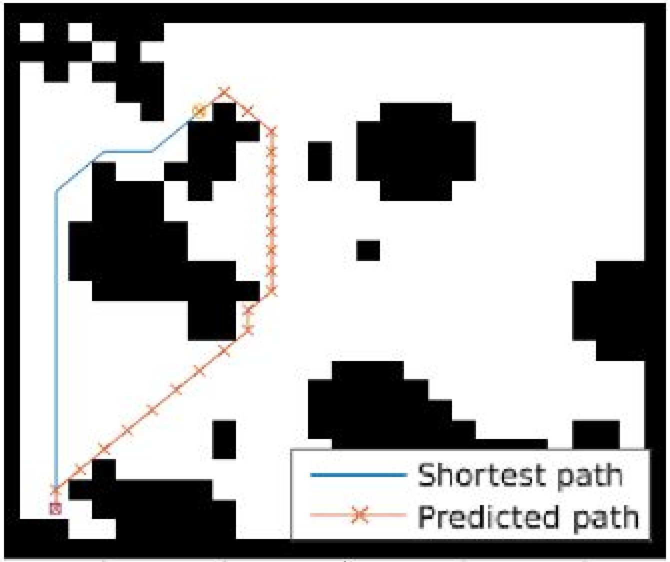


Success Rate	VIN	CNN	FCN
8x8	99.6%	97.9%	97.3%
16x16	99.3%	87.6%	88.3%
28x28	97%	74.2%	76.6%

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Grid World Experiment

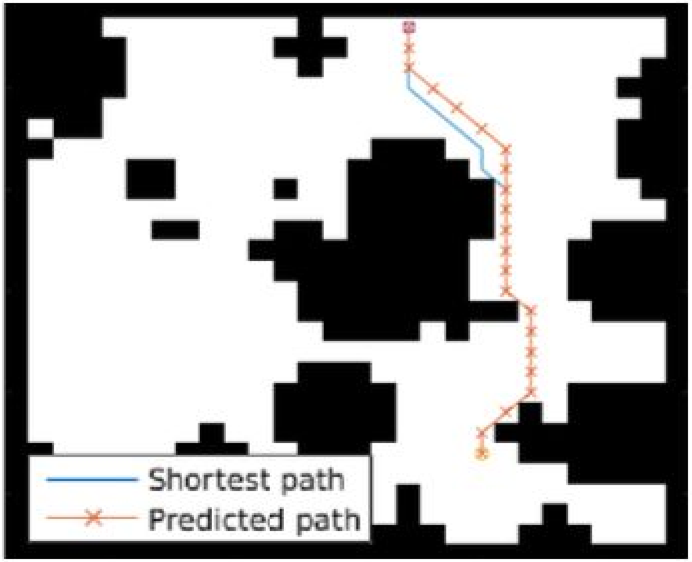


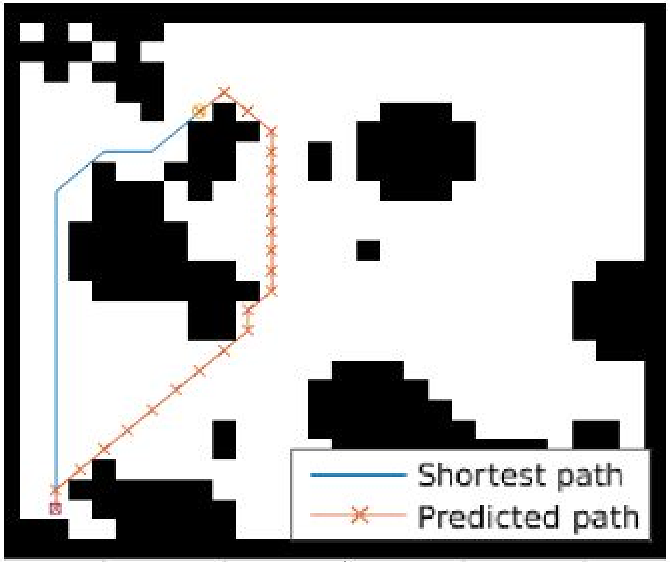


Success Rate	VIN	CNN	FCN
8x8	99.6%	97.9%	97.3%
16x16	99.3%	87.6%	88.3%
28x28	97%	74.2%	76.6%

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Grid World Experiment

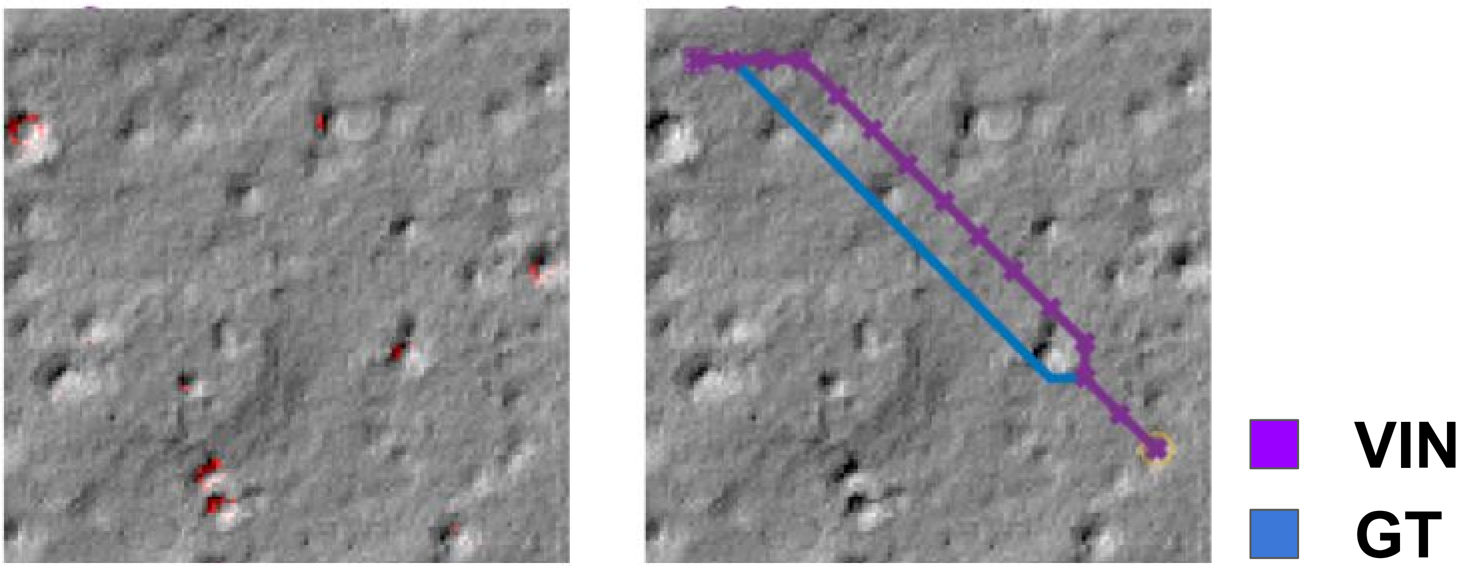




Success Rate	VIN	CNN	FCN
8x8	99.6%	97.9%	97.3%
16x16	99.3%	87.6%	88.3%
28x28	97%	74.2%	76.6%

Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

Mars Rover Experiment



Rover needs to avoid elevation angles greater than 10 degrees.
Elevation needs to be inferred from the input image.

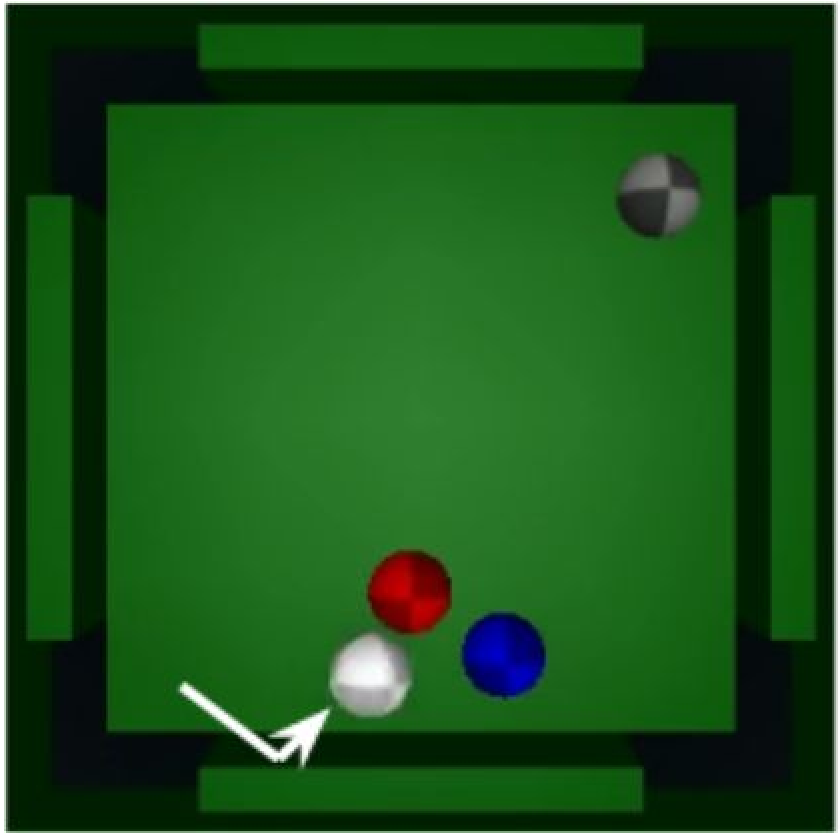
Tamar, Aviv, et al. "Value iteration networks." NIPS. 2016.

The Predictron: End-to-End Learning and Planning

David Silver et. al

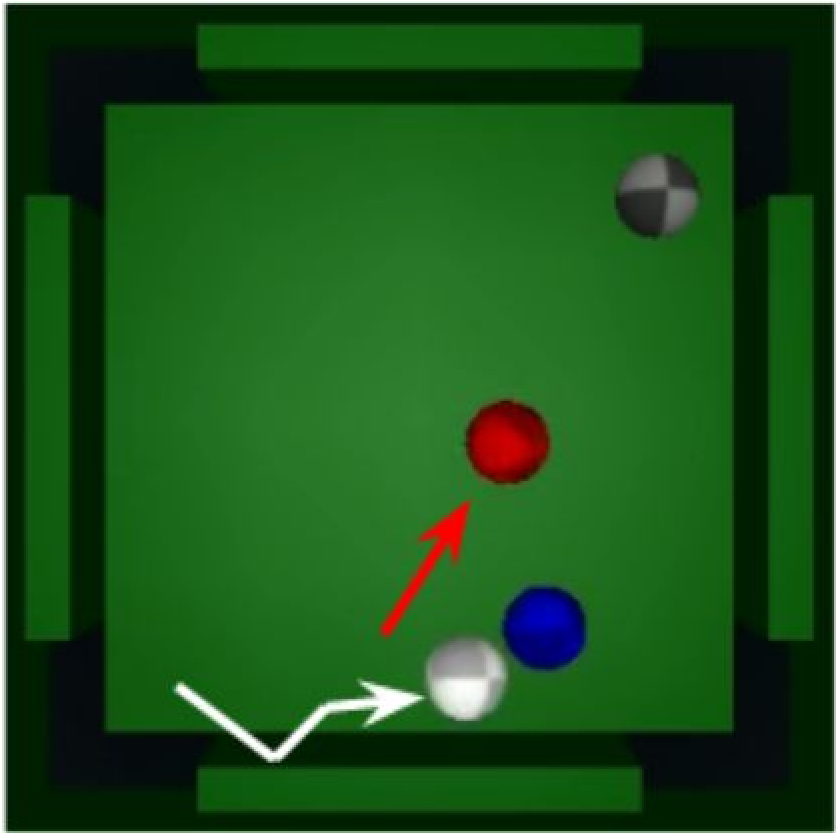
Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Motivation



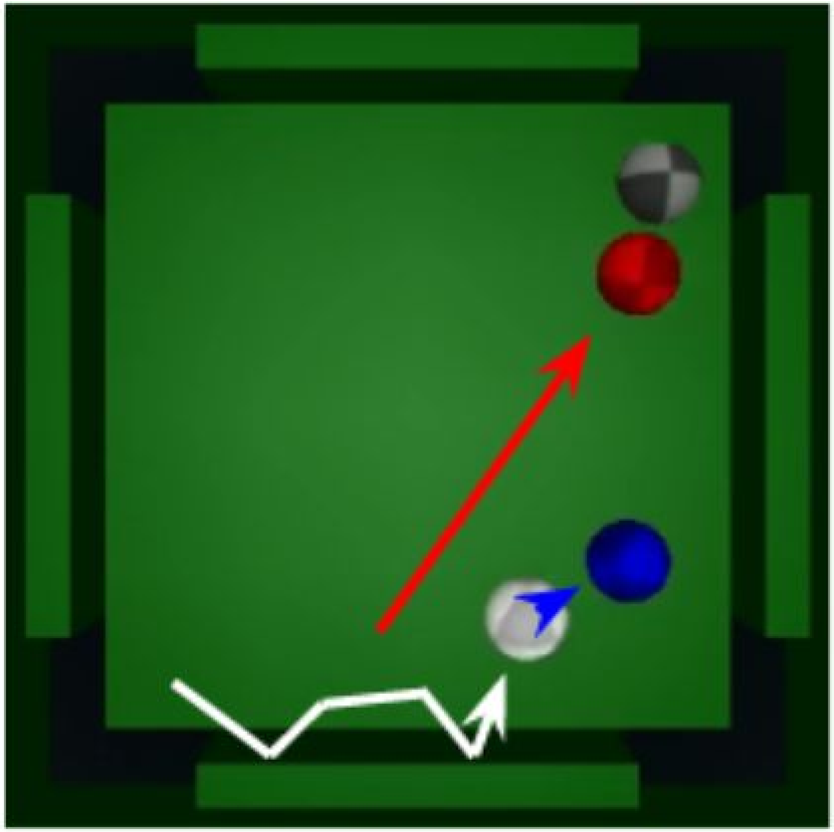
Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Motivation



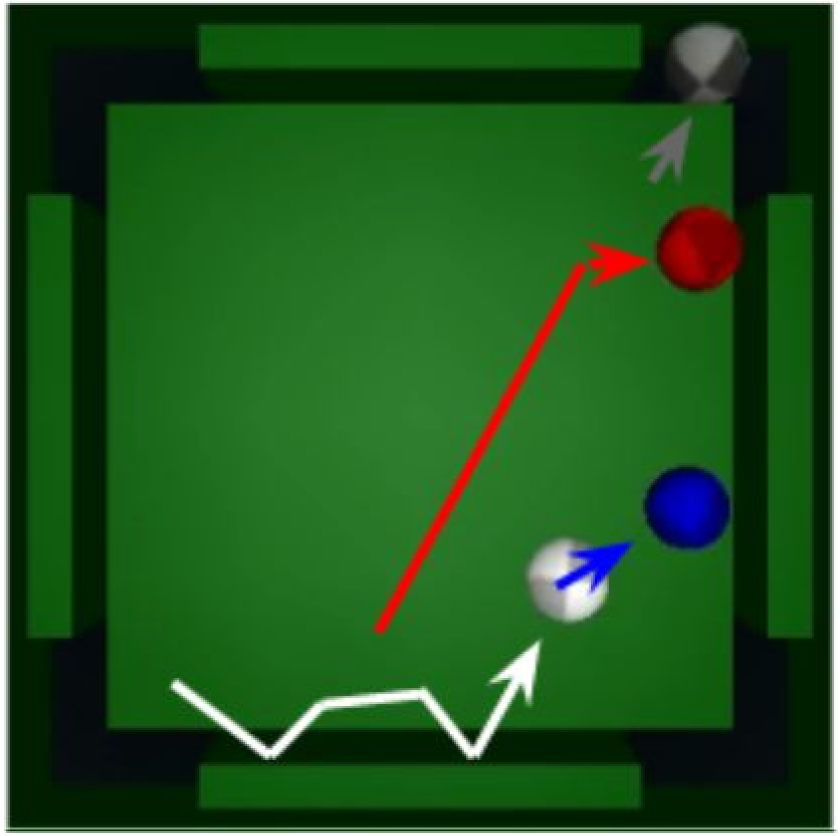
Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Motivation



Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

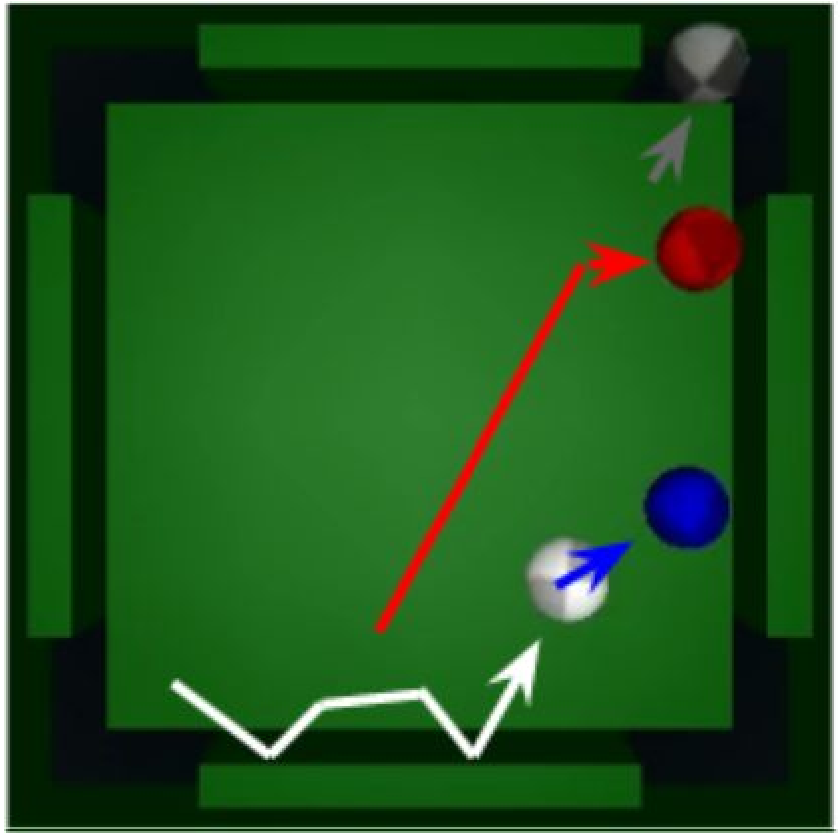
Motivation



Current deep classification/regression nets cannot **unfold into the future** for making predictions

Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Motivation



Predictron: An architecture for prediction tasks with inbuilt ***planning*** computation

Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Architecture motivated by MRP

Imagine a **Markov Reward Process** with:

1. Initial state set as **Input**

$$s_0 = I$$

2. Network for **value of a state**

$$v_i = v(s_i; \theta_v)$$

3. Network for **state transition**

$$s_{i+1}, r_{i+1}, \gamma_{i+1} = m(s_i, \beta; \theta_m)$$

1-step Preturn: $g_1 = r_1 + \gamma_1 v_1$

Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Architecture motivated by MRP

Imagine a **Markov Reward Process** with:

1. Initial state set as **Input**

$$s_0 = I$$

2. Network for **value of a state**

$$v_i = v(s_i; \theta_v)$$

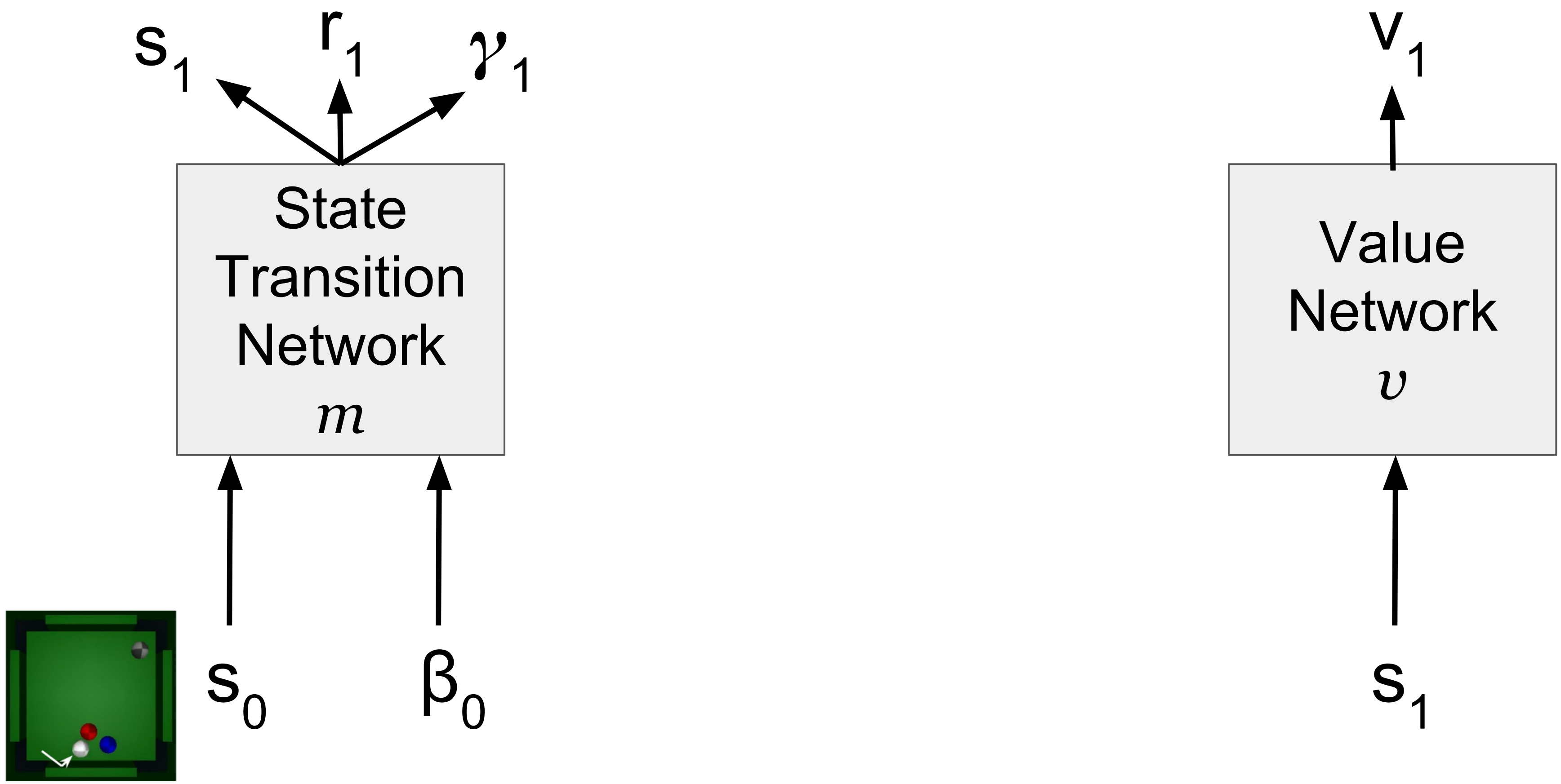
3. Network for **state transition**

$$s_{i+1}, r_{i+1}, \gamma_{i+1} = m(s_i, \beta; \theta_m)$$

2-step Preturn: $g_2 = r_1 + \gamma_1(r_2 + \gamma_2 v_2)$

Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

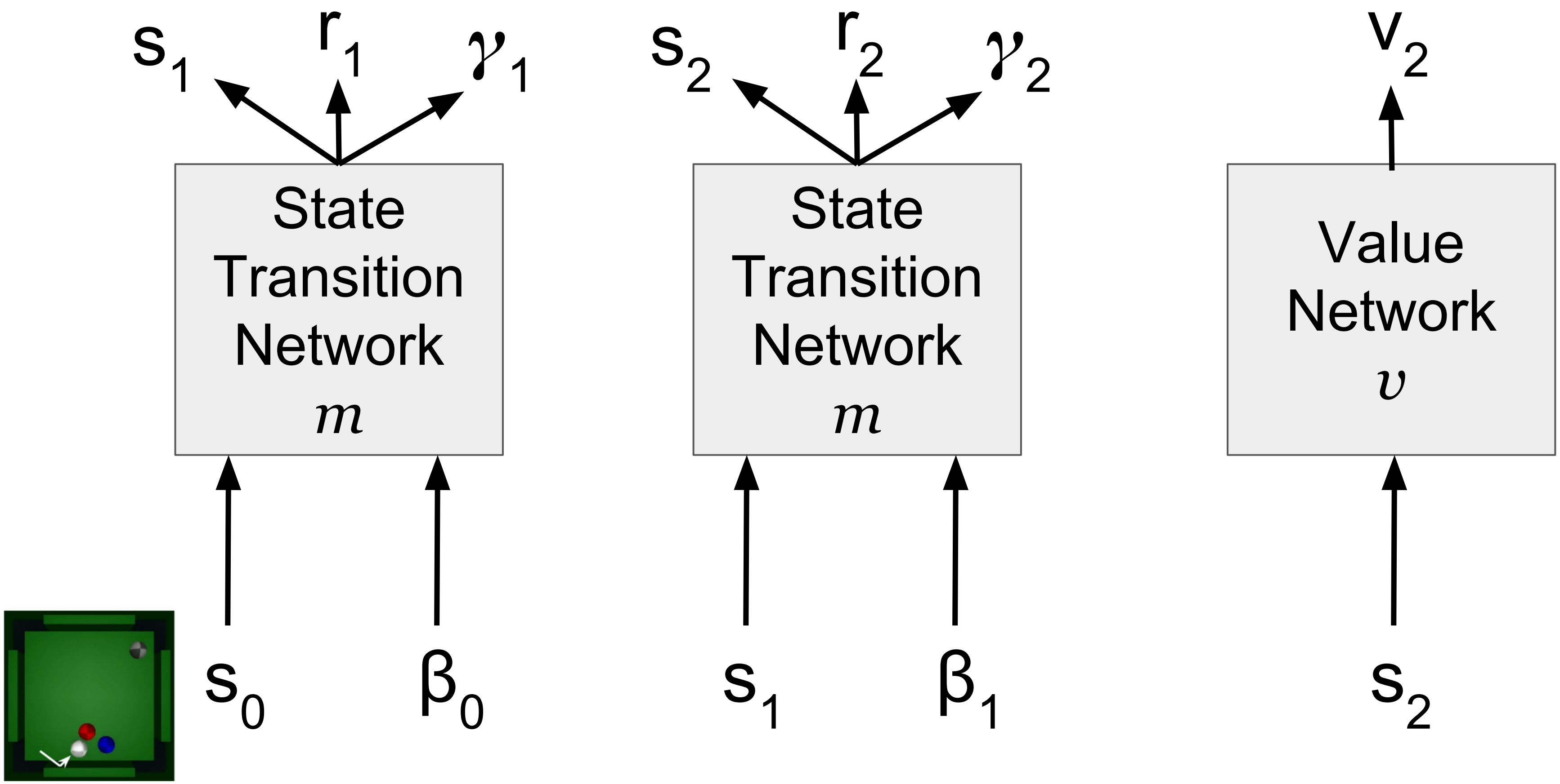
Inference



1-step Preturn:

$$g_1 = r_1 + \gamma_1 v_1$$

Inference



2-step Preturn:

$$g_2 = r_1 + \gamma_1(r_2 + \gamma_2 v_2)$$

Inference

k-step Predictron output is a Monte-Carlo estimate of expected k-step Preturns

$$E_m[g_k|s]$$

Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

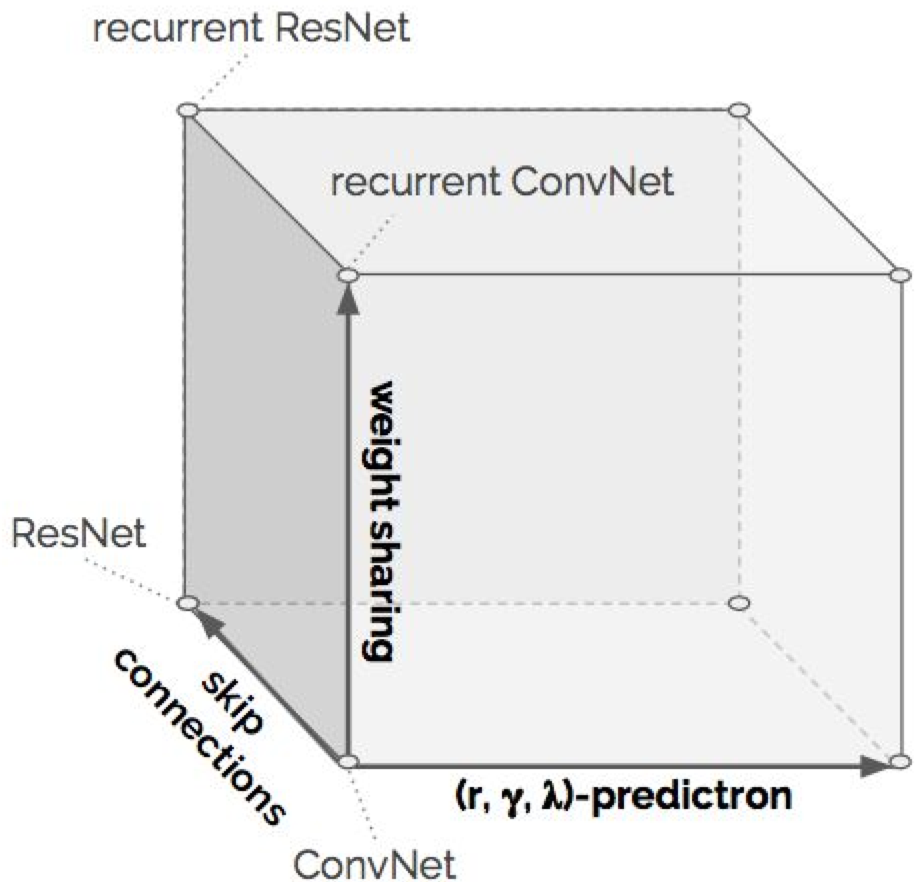
Learning

$$\mathcal{L}(\theta_m, \theta_v; s) = \frac{1}{2} \| \boxed{E_p[g|s]} - E_m[g_k|s] \|^2$$

Real Environment

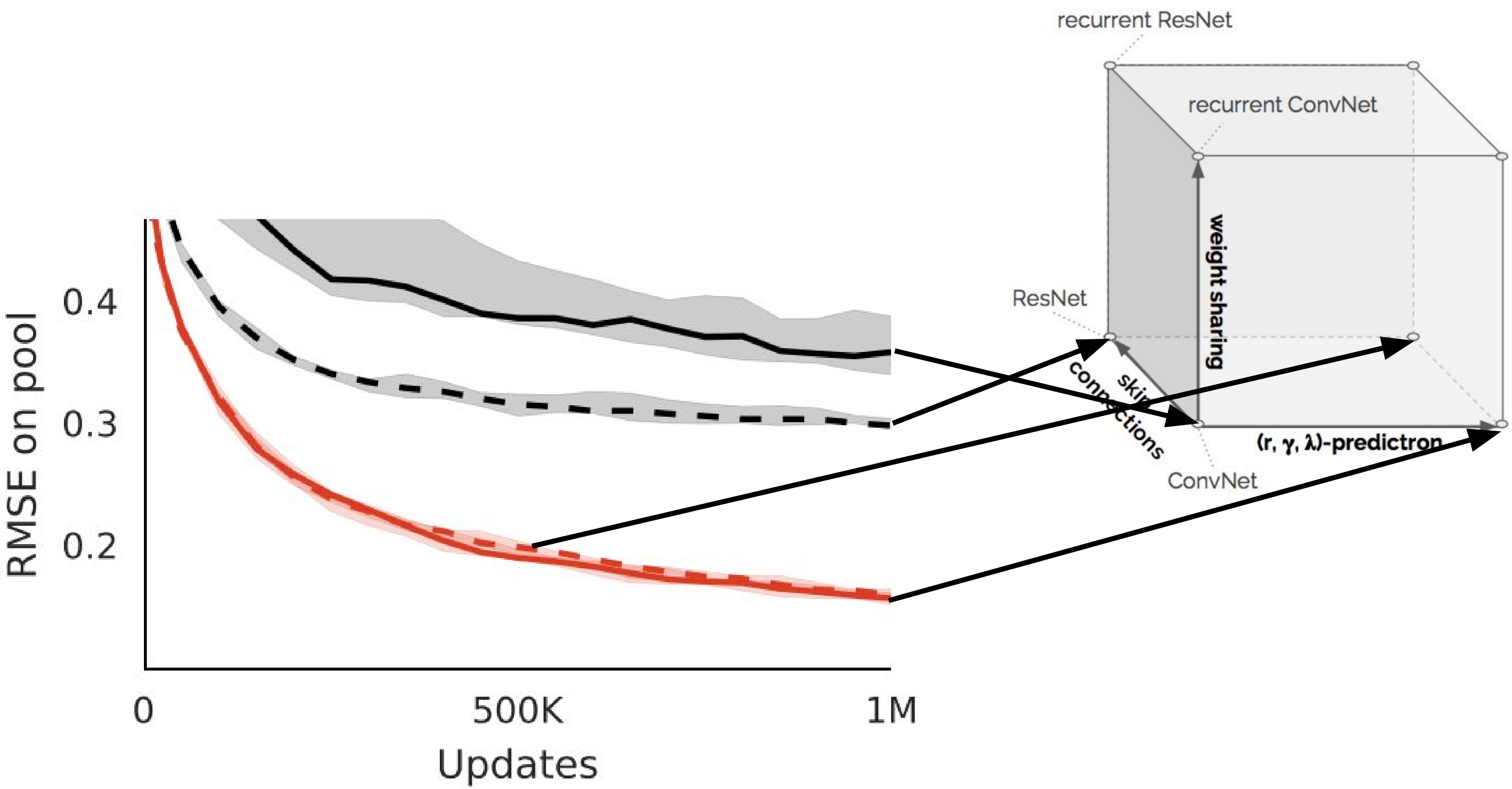
Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Experiments



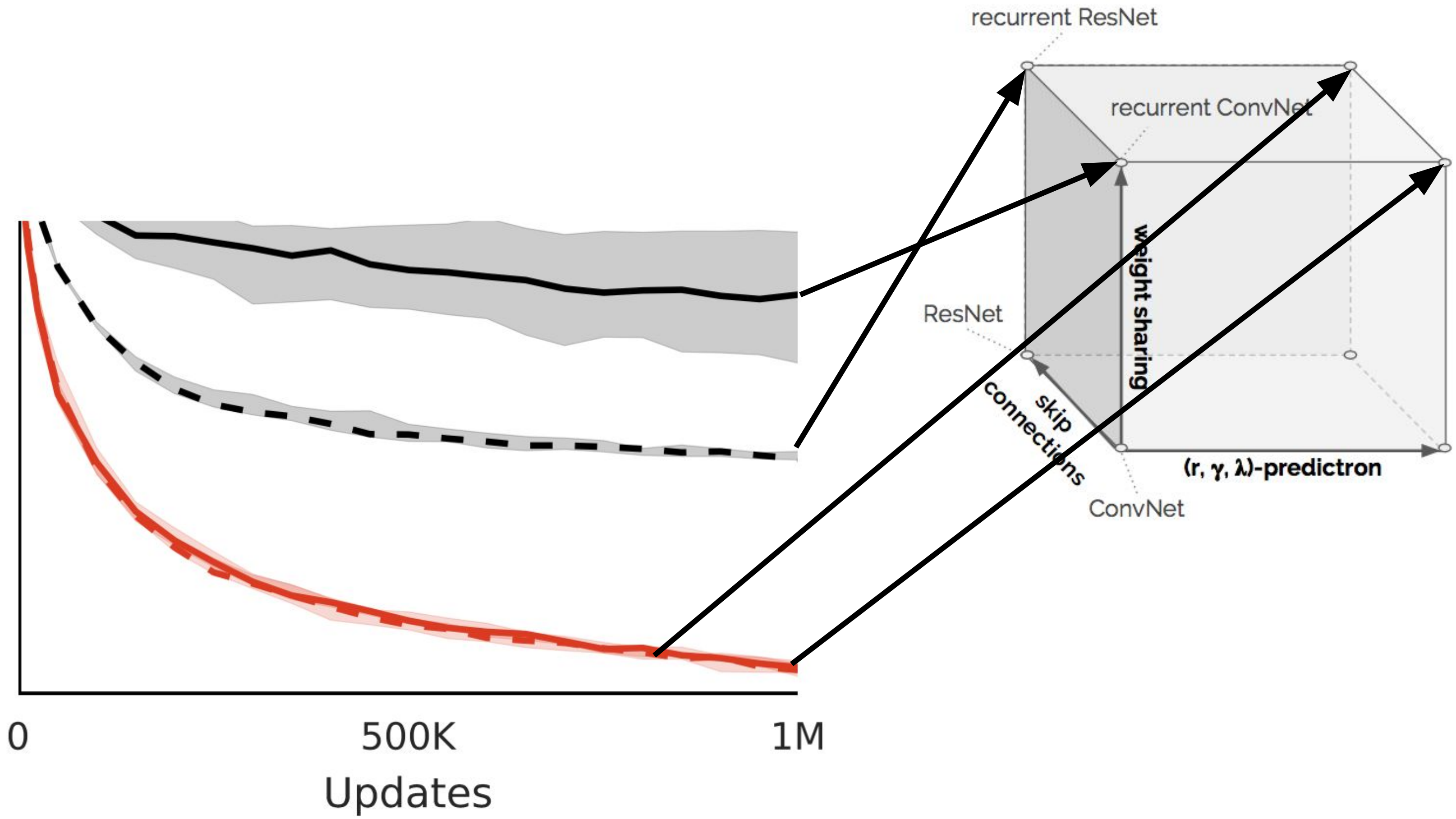
Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Experiments



Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Experiments



Silver, David, et al. "The predictron: End-to-end learning and planning." arXiv:1612.08810 (2016).

Summary

- Policy Gradients
  - Stochastic;
  - Better properties; variance in training
  - Maximize expected returns
  - Actor-critic methods: Using value-networks to reduce variance
  - A3C: parallel environments decorrelates training data

- Model-based learning
  - Planning helps learning by modeling environment
  - Dyna: new data from model
  - Value Iteration Networks: generalization
  - Predictron: reason about future

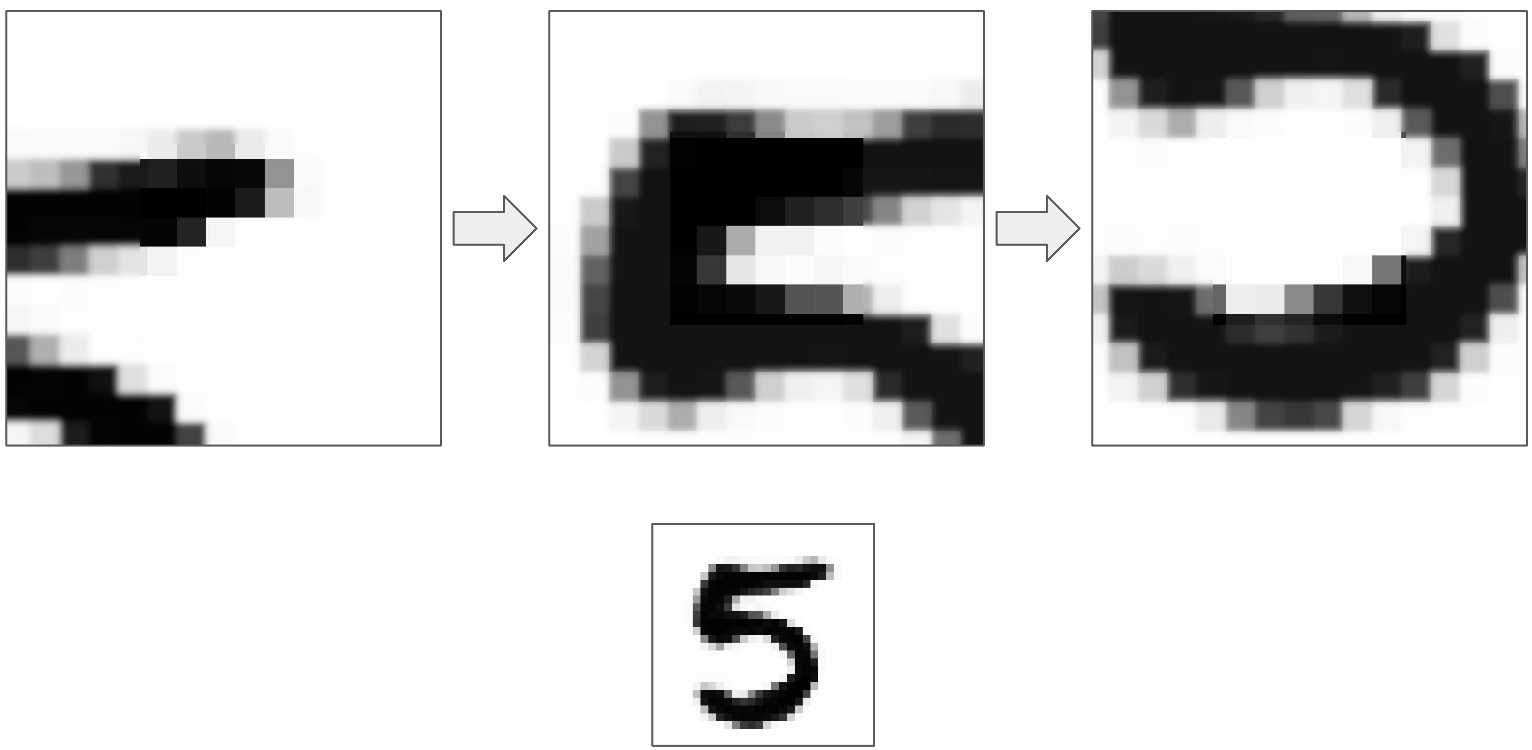
Applications

Recurrent Models of Visual Attention

Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu

Motivation

- **Task**: Classify digits in MNIST
- **Motivation**: Full image convolution is expensive!
  - Humans focus attention selectively on parts of an image
  - Combine information from different fixations over time

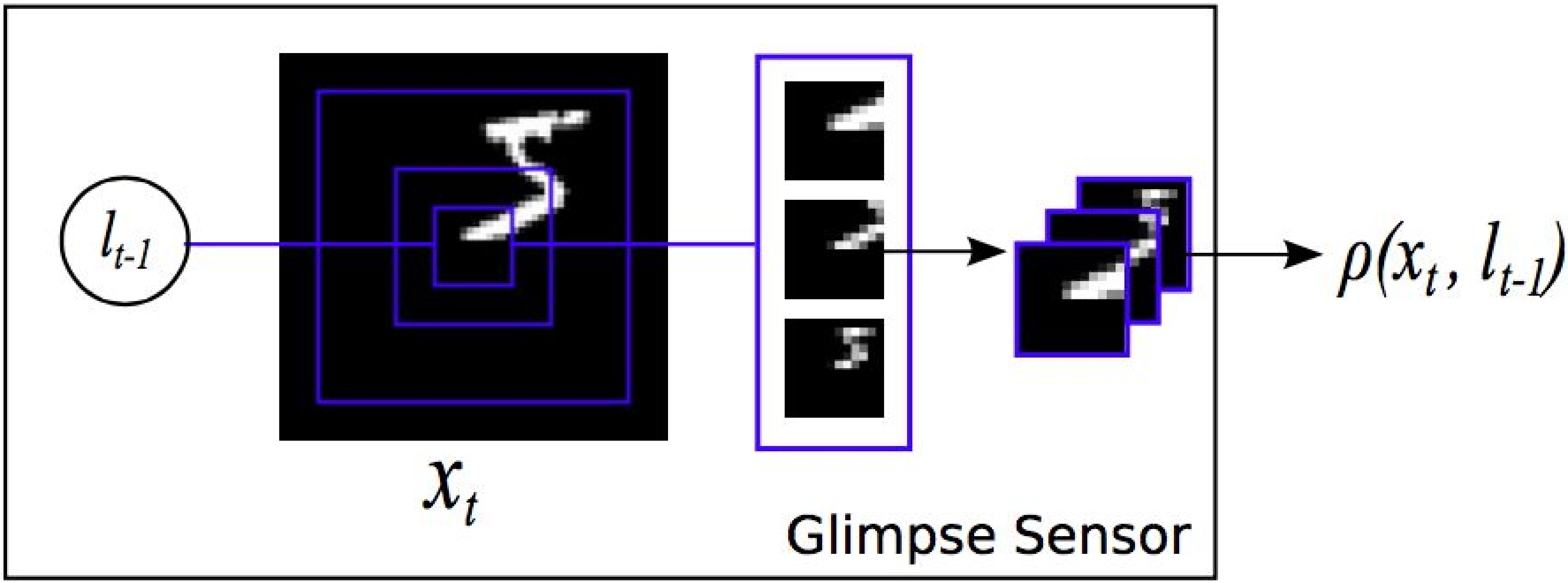


Overview

- True state of the environment is unobserved
  - **Glimpses** can be seen as a partial view of the state
- **State:** $h_t = f_h(h_{t-1}, g_t; \theta_h)$
- **Actions:**
  - Location: $l_t \sim p(.|f_l(h_t; \theta_l))$
  - An environment action: $a_t \sim p(.|f_a(h_t; \theta_a))$
- **Reward**: Cross-Entropy Loss
- Agent needs to learn a **stochastic** policy
  - Policy π is defined by the **Location Network** in the RNN

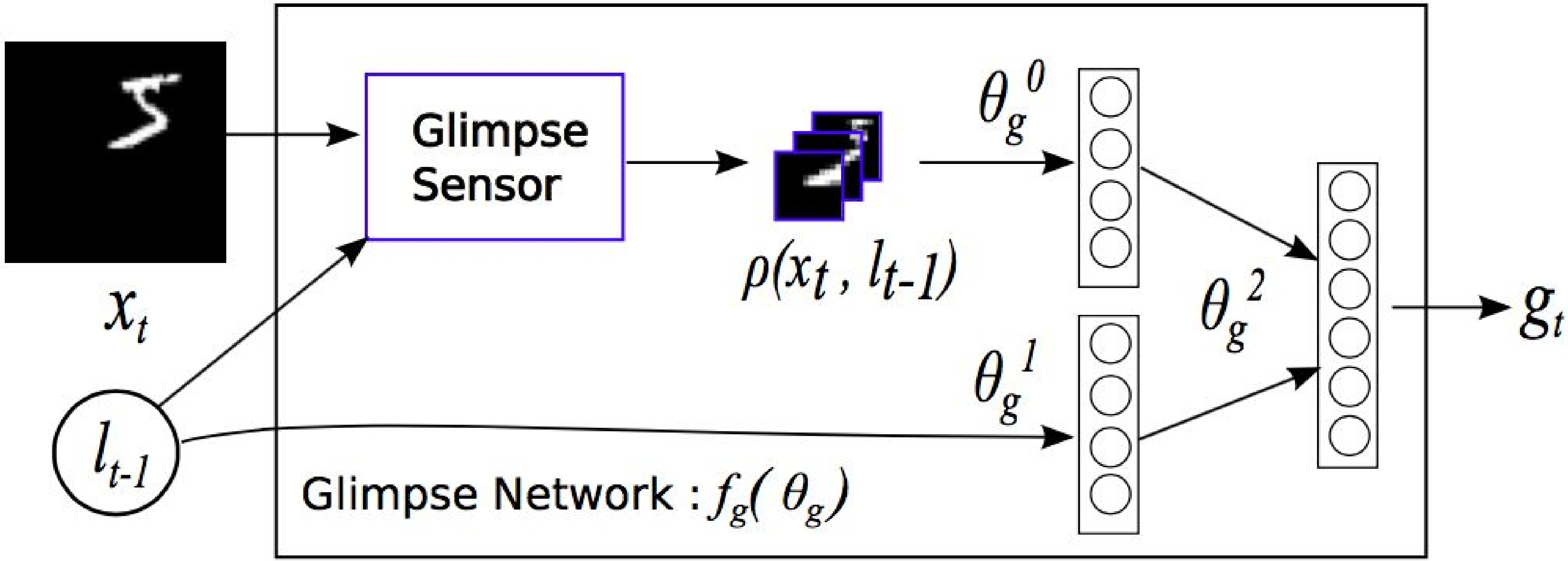
Glimpse

- Retina-like representation $\rho(x_t, l_{t-1})$
  - Contains multiple resolution patches
- Centered at location l_{t-1} of image x_t

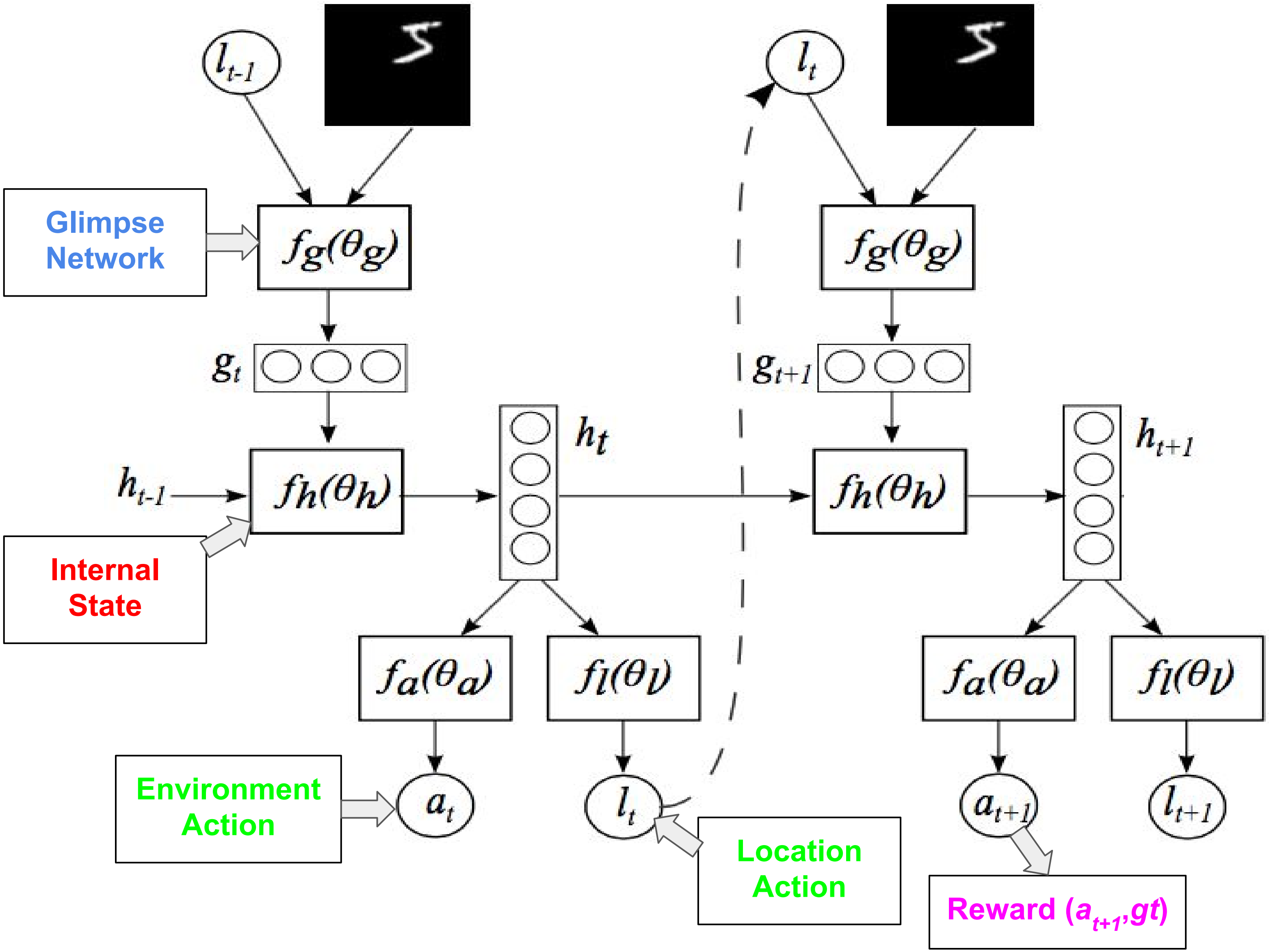


Glimpse Network

- $\rho(x_t, l_{t-1})$ and l_{t-1} are mapped into a hidden space



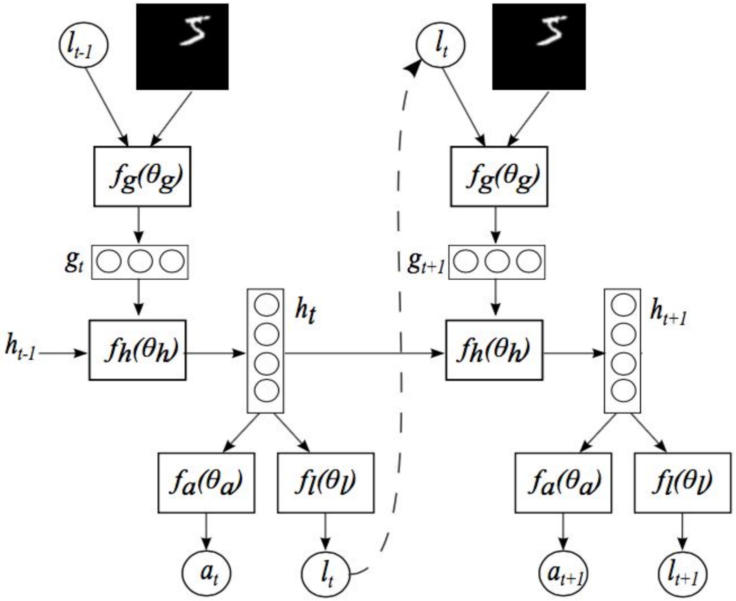
Model Architecture



Training

"OK, I've shown you the ropes, given you the low down, and gotten you up to speed. All that's left is actually training you."

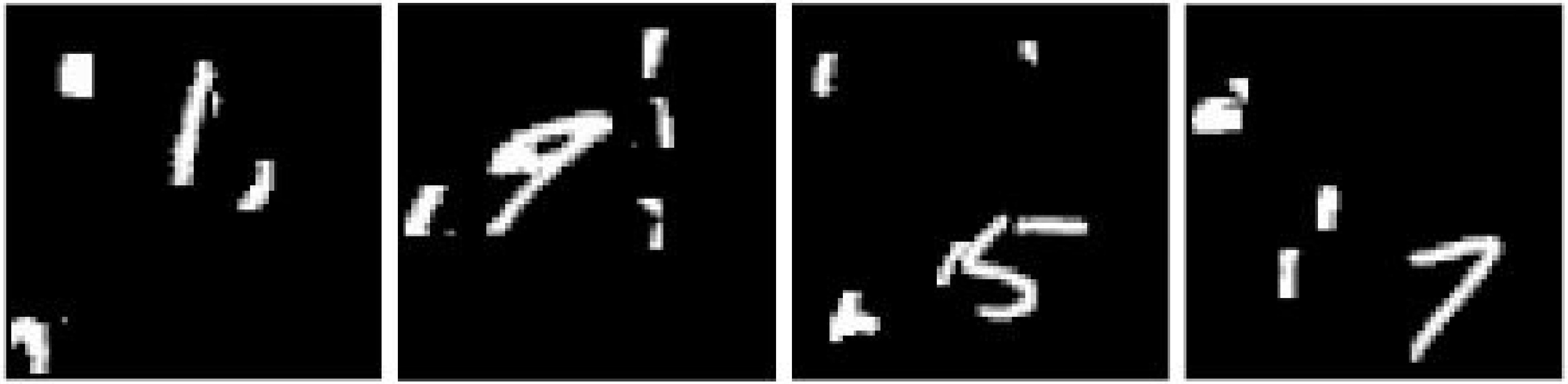
Training



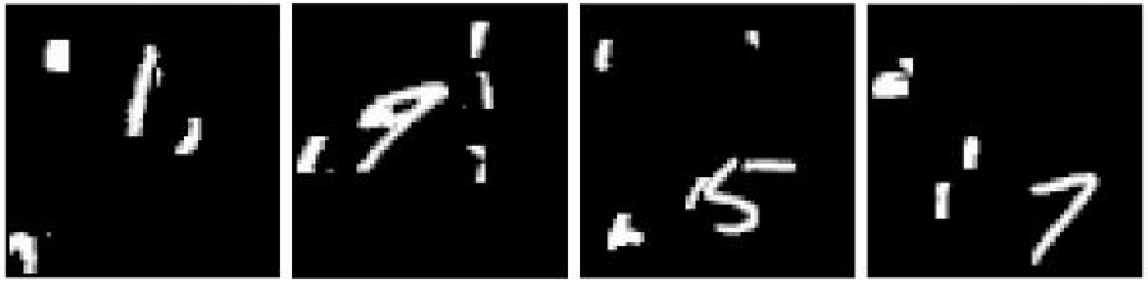
- Parameters of the agent are: $\theta = \{\theta_g, \theta_h, \theta_a\}$
  - Can be trained using standard backpropagation
- **RL Objective**: Maximize the reward given by: $J(\theta) = E[R]$
  - Can maximize $J(\theta)$ using **REINFORCE**

$$\nabla_\theta J = \sum_{t=1}^{T} \mathbb{E}_{p(s_{1:T};\theta)} \left[\nabla_\theta \log \pi(u_t | s_{1:t}; \theta) R \right] \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i | s_{1:t}^i; \theta) R^i$$

Results



Results



(a) 60x60 Cluttered Translated MNIST

Model	Error
FC, 2 layers (64 hiddens each)	28.58%
FC, 2 layers (256 hiddens each)	11.96%
Convolutional, 2 layers	8.09%
RAM, 4 glimpses, 12×12, 3 scales	4.96%
RAM, 6 glimpses, 12×12, 3 scales	4.08%
RAM, 8 glimpses, 12×12, 3 scales	4.04%
RAM, 8 random glimpses	14.4%

(b) 100x100 Cluttered Translated MNIST

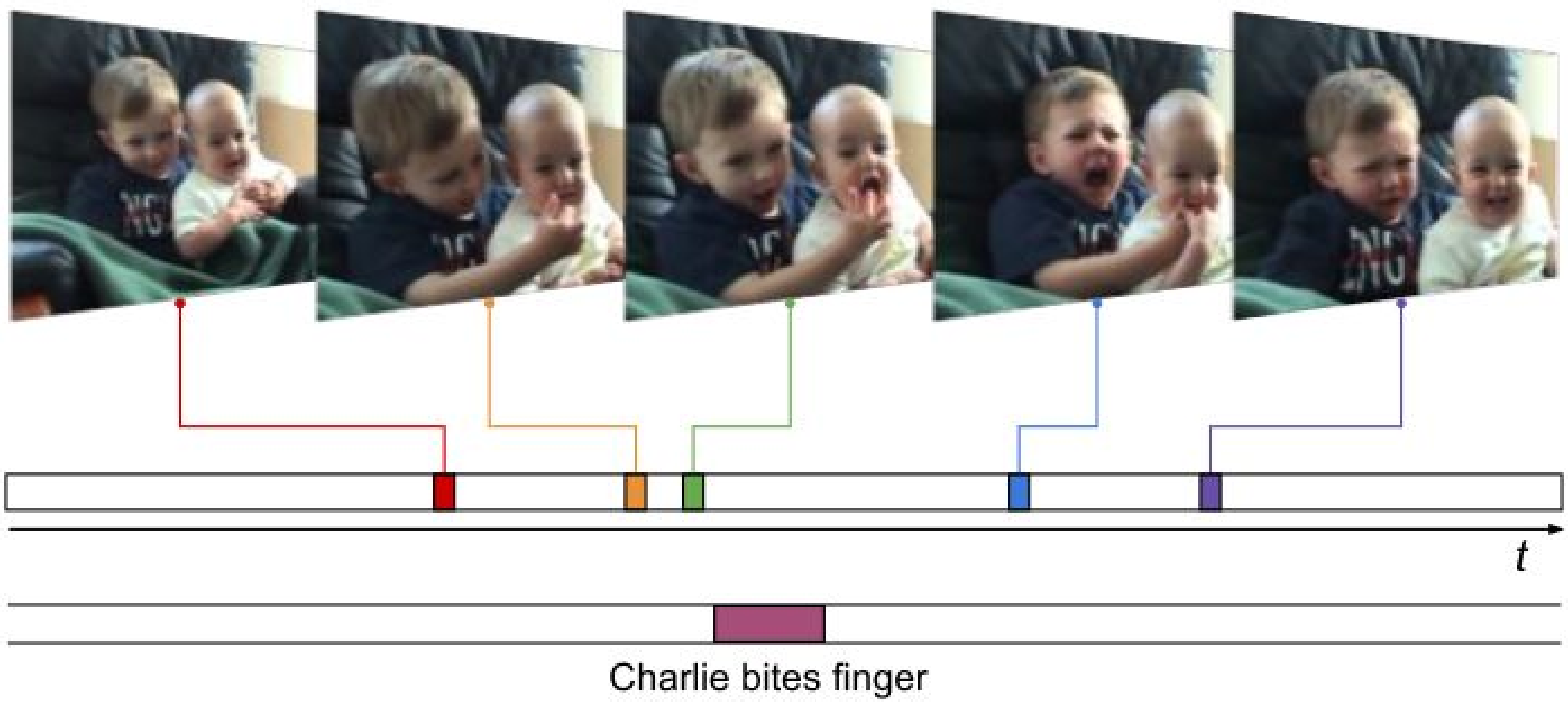
Model	Error
Convolutional, 2 layers	14.35%
RAM, 4 glimpses, 12×12, 4 scales	9.41%
RAM, 6 glimpses, 12×12, 4 scales	8.31%
RAM, 8 glimpses, 12×12, 4 scales	8.11%
RAM, 8 random glimpses	28.4%

End-to-end Learning of Action Detection from Frame Glimpses in Videos

Serena Yeung, Olga Russakovsky, Greg Mori, Li Fei-Fei

Motivation

- **Task**: Detect and classify moments in an untrimmed video
- **Motivation:** Looking at all frames in a video is slow!
  - Process of detecting actions is one of observation and refinement

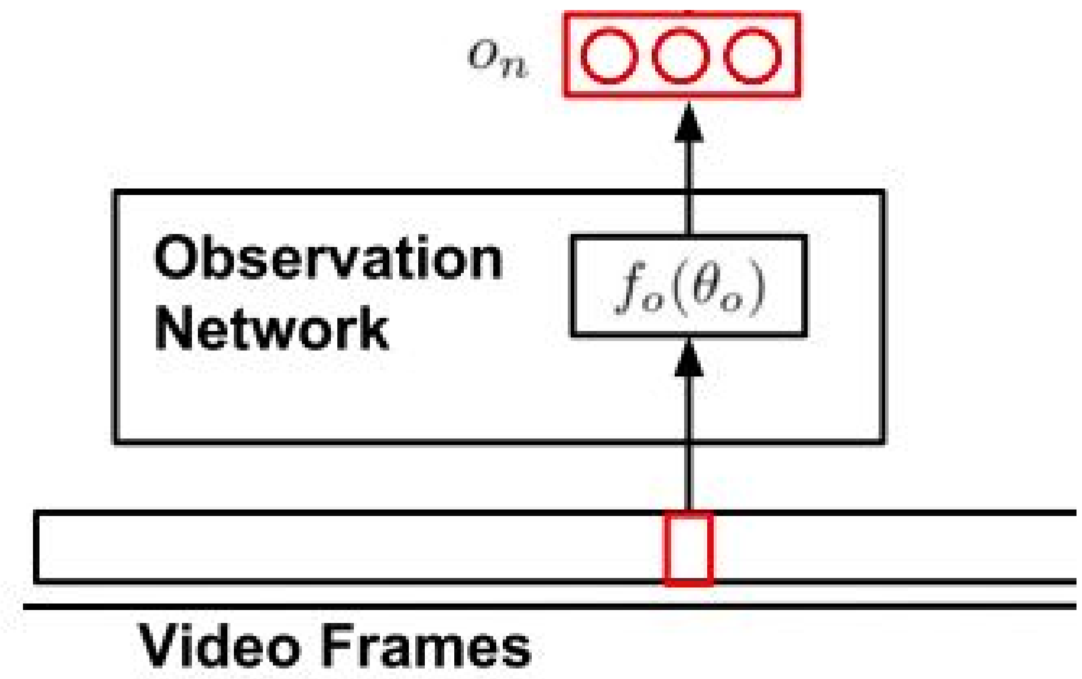


Overview

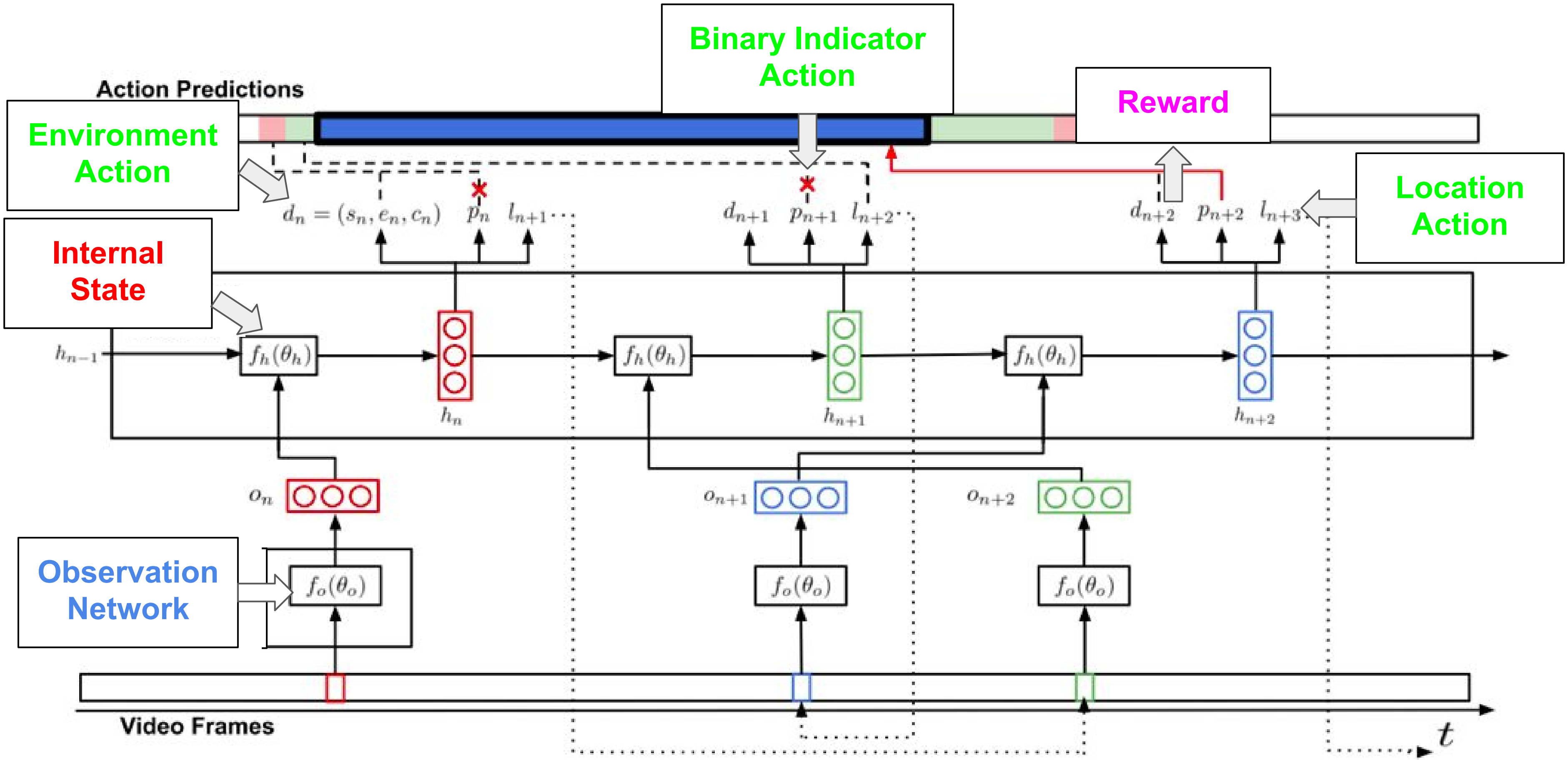
- True state of the environment is unobserved
  - **Observation Network** can be seen as a partial view of the state
- **State:** $h_n = f_h(h_{n-1}, o_n; \theta_h)$
- **Actions:**
  - Candidate detection: $d_n = f_d(h_n; \theta_d)$
  - Binary indication: $p_n = f_p(h_n; \theta_p)$
  - Temporal location: $l_{n+1} = f_l(h_n; \theta_l)$
- **Reward**: $R_N = \begin{cases} R_0 & \text{if } M > 0 \text{ and } N_p = 0 \\ N_+ R_+ + N_- R_- & \text{otherwise} \end{cases}$
- Agent needs to learn a **stochastic** policy
  - Policy π is defined by the **Location Network** in the RNN

Observation Network

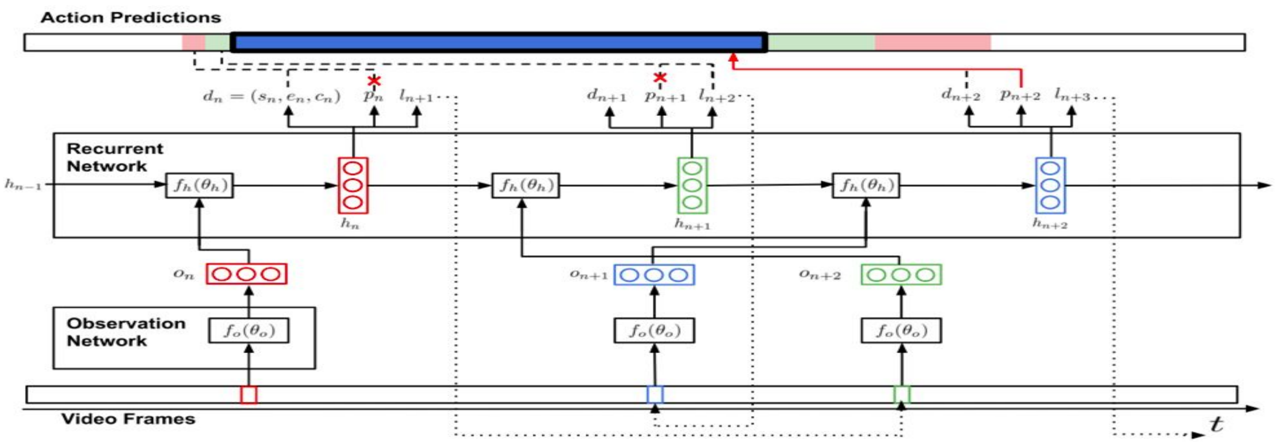
- Observes a single video frame at each timestep and encodes the frame and it's location into a feature vector o_n
  - Inspired by the **Glimpse network**



Model Architecture



Training

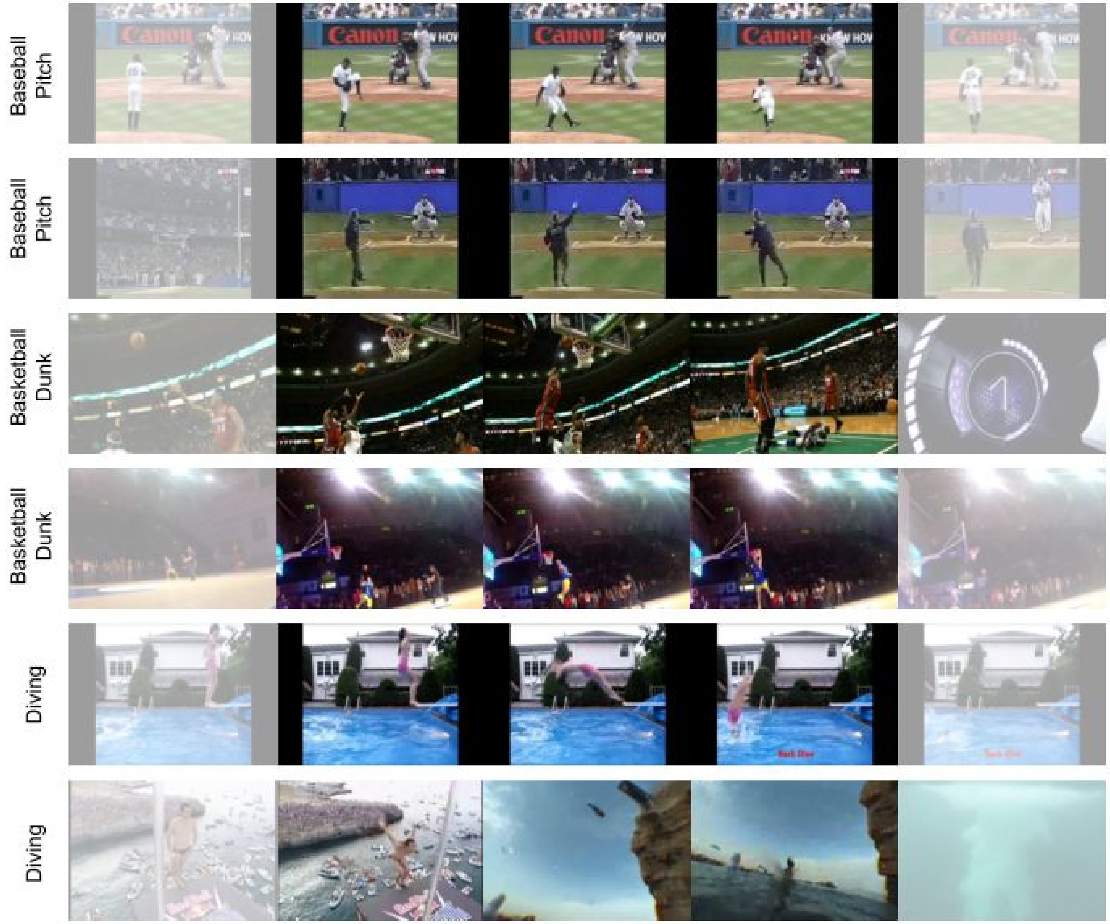


- Parameters of the agent are: $\theta = \{\theta_o, \theta_h, \theta_d\}$
  - Can be trained using standard backpropagation
- **RL Objective**: Maximize the reward given by: $J(\theta) = E[R]$
  - $L(D) = \sum_n L_{cls}(d_n) + \gamma \sum_n \sum_m \mathbb{1}[y_{nm} = 1] L_{loc}(d_n, g_m)$
  - Can maximize $J(\theta)$ using **REINFORCE**

$$\nabla_\theta J = \sum_{t=1}^{T} \mathbb{E}_{p(s_{1:T};\theta)} \left[\nabla_\theta \log \pi(u_t | s_{1:t}; \theta) R\right] \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^i | s_{1:t}^i; \theta) R^i$$

Results - I

- THUMOS 14’ Dataset
    - Correct Predictions



Results - II

	[23]	Ours		[23]	Ours
Baseball Pitch	8.6	**14.6**	Hamm. Throw	**34.7**	28.9
Basket. Dunk	1.0	**6.3**	High Jump	17.6	**33.3**
Billiards	2.6	**9.4**	Javelin Throw	**22.0**	20.4
Clean and Jerk	13.3	**42.8**	Long Jump	**47.6**	39.0
Cliff Diving	**17.7**	15.6	Pole Vault	**19.6**	16.3
Cricket Bowl.	9.5	**10.8**	Shotput	11.9	**16.6**
Cricket Shot	2.6	**3.5**	Soccer Penalty	**8.7**	8.3
Diving	4.6	**10.8**	Tennis Swing	3.0	**5.6**
Frisbee Catch	1.2	**10.4**	Throw Discus	**36.2**	29.5
Golf Swing	**22.6**	13.8	Volley. Spike	1.4	**5.2**
mAP				14.4	**17.1**

- **Key Takeaways**:
  - Accuracy is comparable to state-of-the-art
  - Less frames observed

AlphaGo:
A bit of everything

(but mostly plain PG + planning)

https://www.youtube.com/watch?v=4D5yGiYe8p4

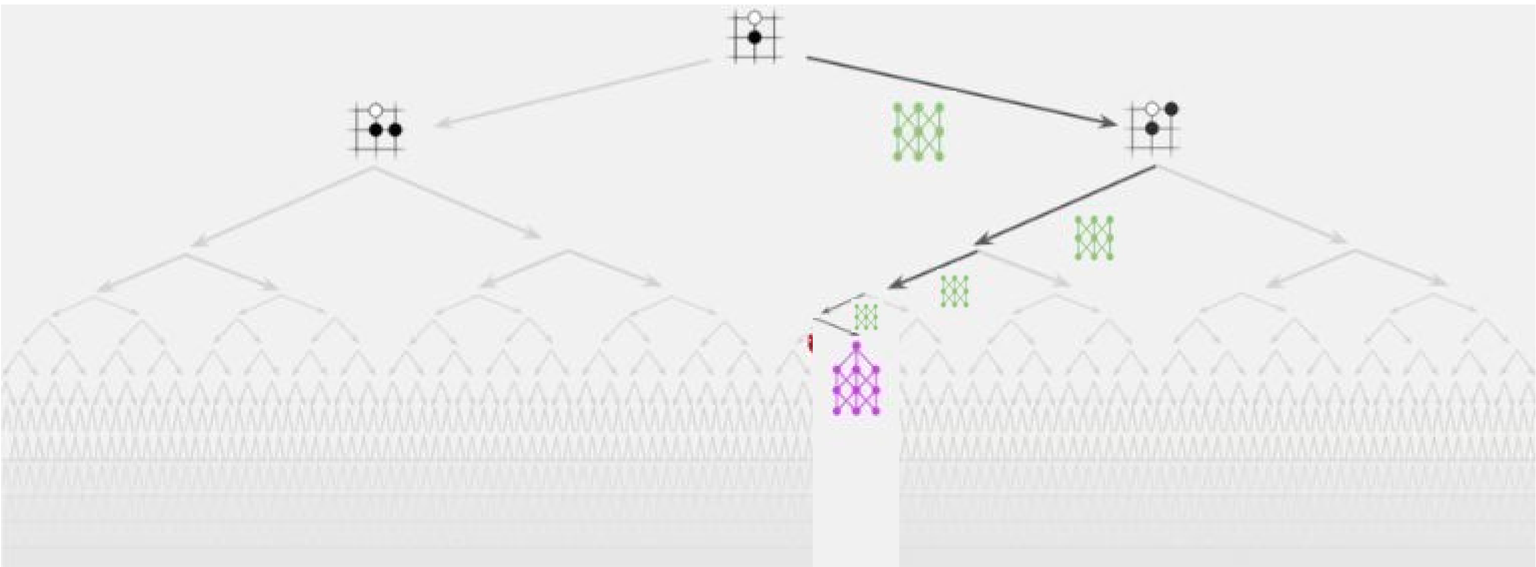
Thanks!

AlphaGo slides

Background: Monte-Carlo Tree Search

Another planning method.

- Sample future paths using stochastic policy
  - Biased towards reasonable moves
  - The predictron paper may do this if they modeled the environment $\mathbb{P}(s'|s,a)$.

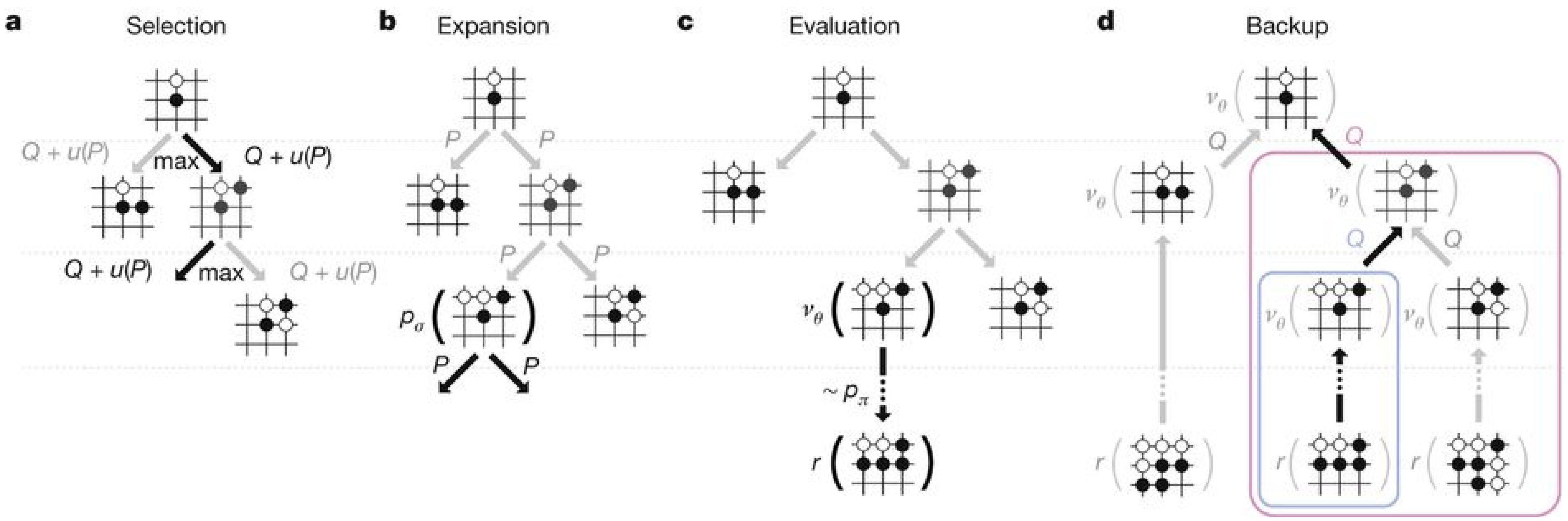


(talk) D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. ICML Workshop 2016

Background: Monte-Carlo Tree Search

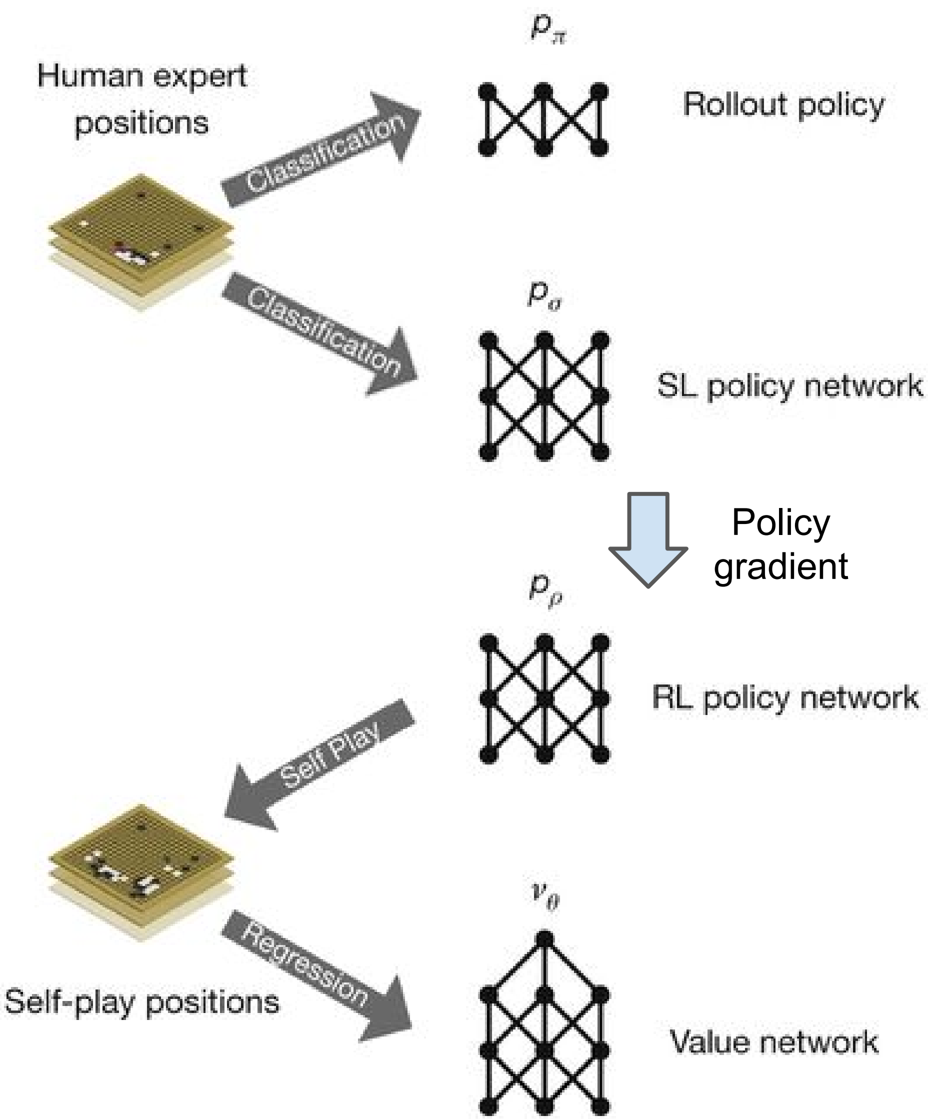
Deterministic environment version.

1. Select path according to π plus exploration
2. Expand leaf node s (compute children and their $\mathbb{P}(\cdot)$)
3. Evaluate $V(s)$ by rolling out (play till the end)
4. Backup: update $Q(s,a)$ along the path (count)



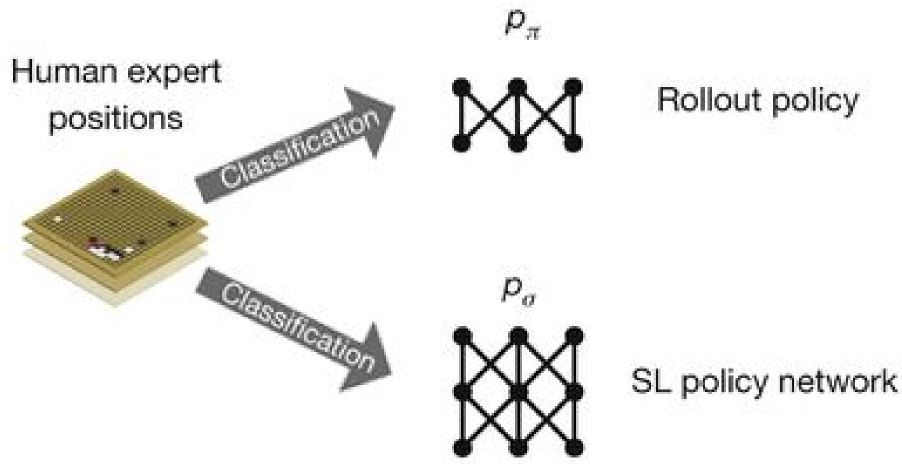
D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models overview



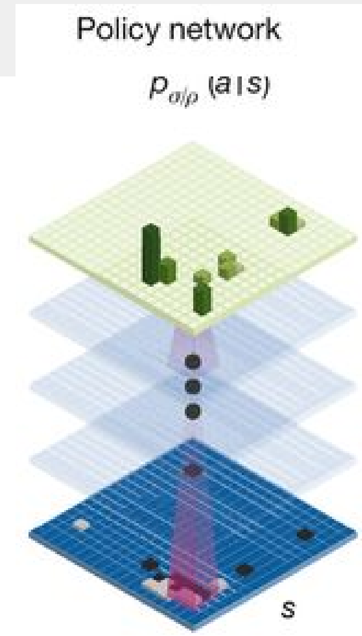
D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models



1-layer network (fast!)

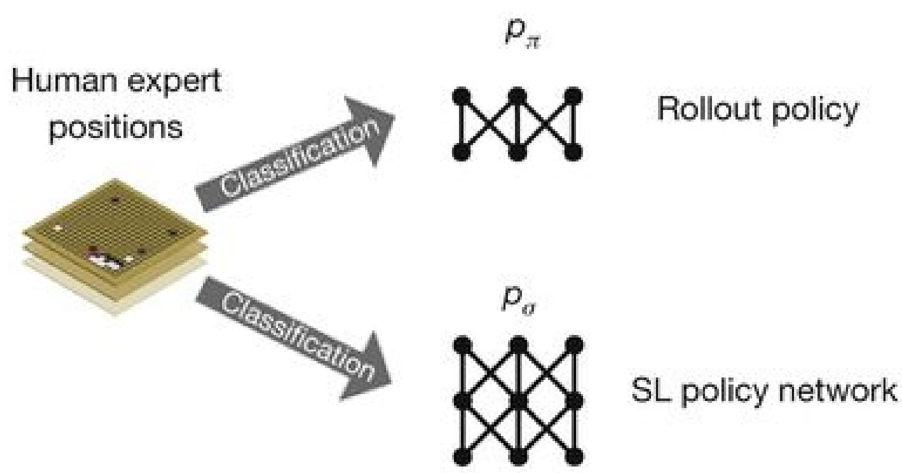
13-layer CNN



- Supervised learning
  - On human expert moves
  - One small (very fast rollout)
    - 2μs; 24.2% accuracy
  - One deeper
    - 57% accuracy w/ handcrafted features;
    - 55.7% using only raw board + past move

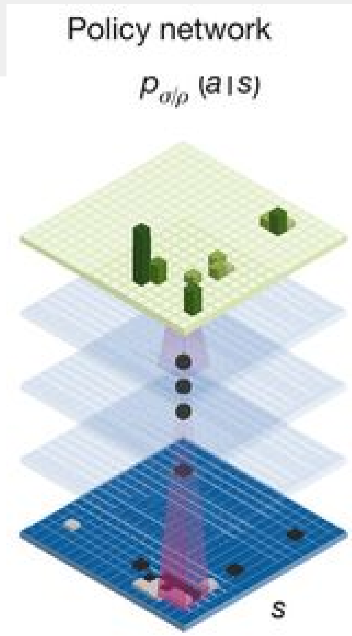
D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models



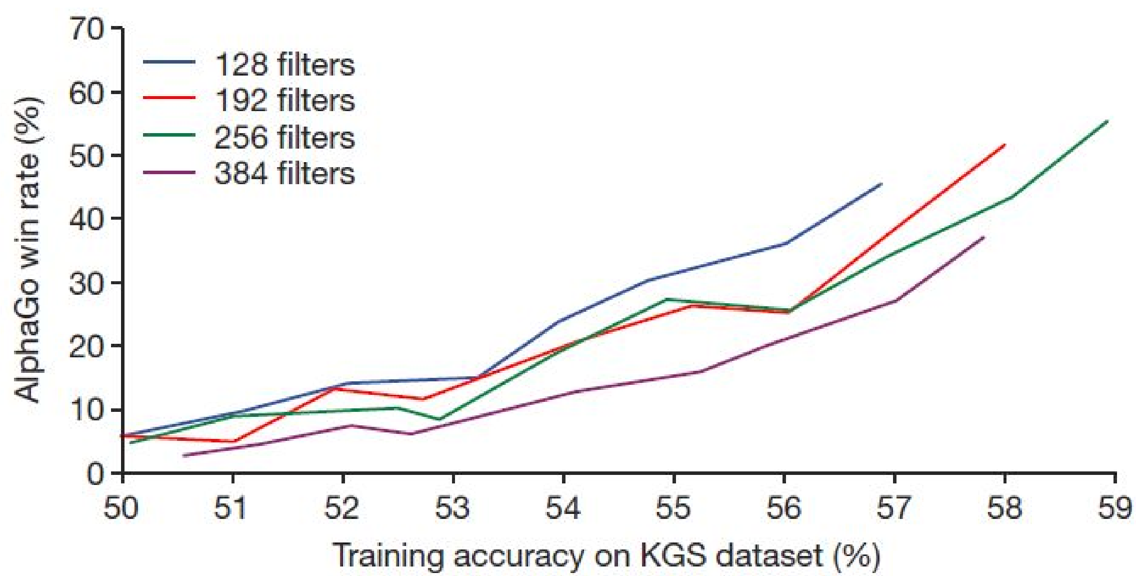
1-layer network (fast!)

13-layer CNN



Importance of classification accuracy

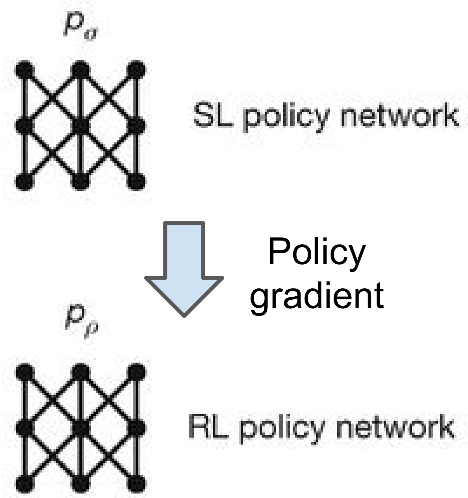
(win rate against final AlphaGo)



D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models

- Policy gradient
  - Improve SL policy to RL policy
    - Playing against its past iterations (less overfitting)

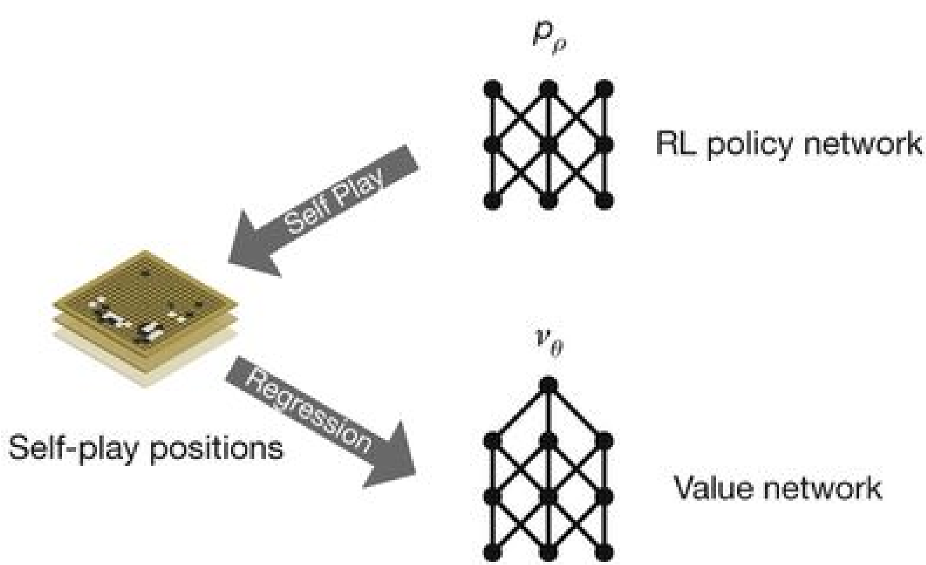


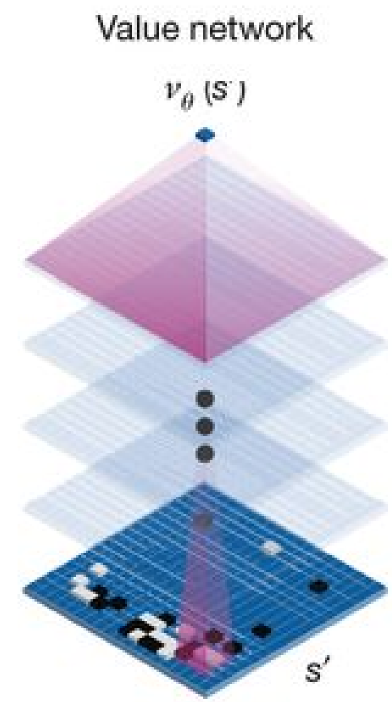
  - Training: PG w/o discount (rewards R_{win} = +1; R_{lose}= -1)
- Wins 80% against SL policy
  - 85% to Pachi (open source s-o-t-a)
  - Ranks ~ 3 amateur dan

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models

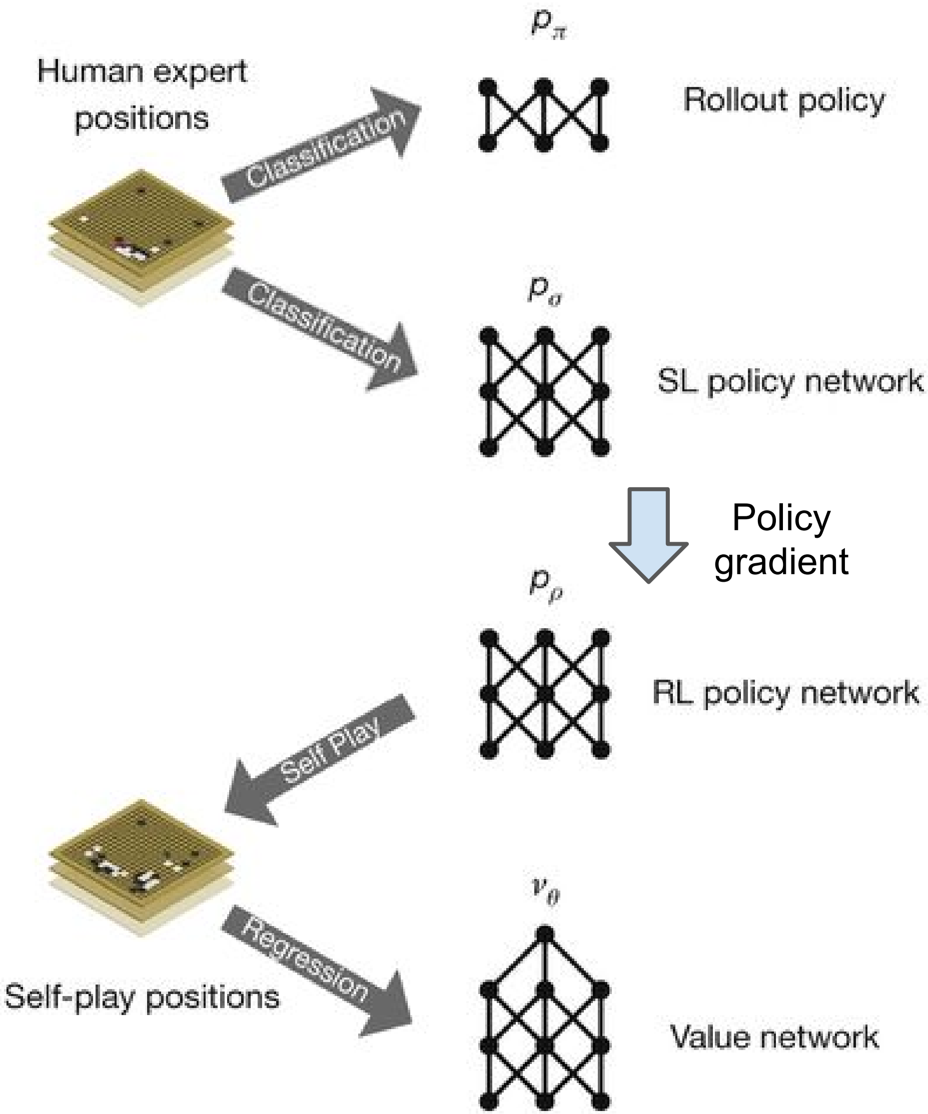
- Value network: evaluate the win-rate of state
  - Use self-play instead of human moves (<u>less overfit</u>)
  - Under "optimal policy" (the RL one)
  - David: "perhaps the key of AlphaGo dev."
    - (first strong state evaluator)





D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo models recap



Fast policy

Human-like policy

"Optimal" policy

Any policy can play go

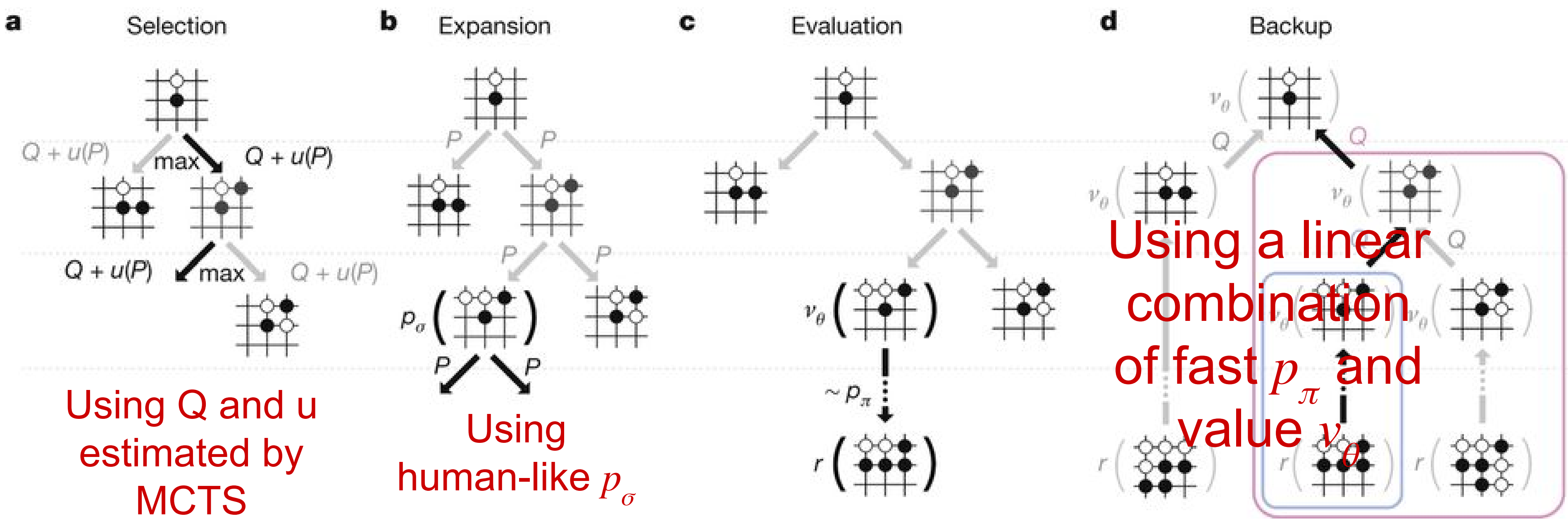
Value according to "optimal" policy

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

Putting everything together w/ MCTS

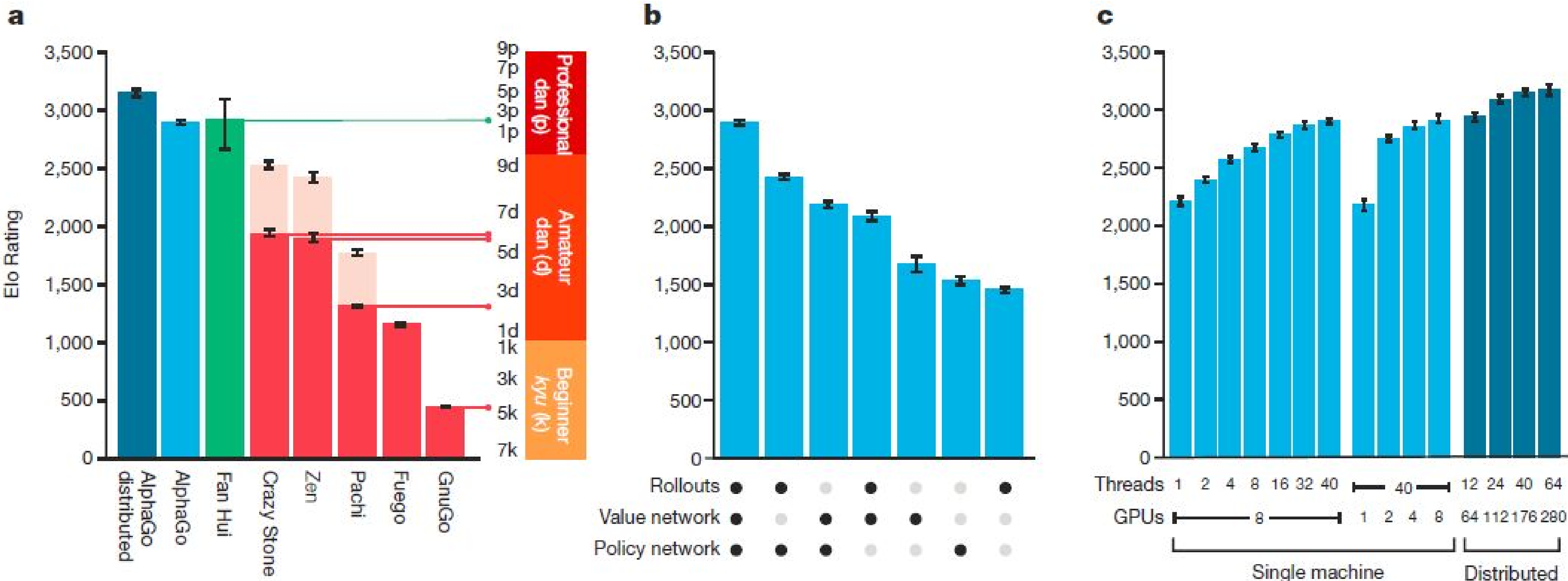
Deterministic environment version.

1. Select path by maximizing estimated Q and exploration *u*
2. Expand leaf node s (compute children's $\mathbb{P}(\cdot)$ using p_σ)
3. Evaluate $V(s)$ by rolling out (using fast p_π and value v_θ)
4. Backup: update $Q(s,a)$ along the path (using count)



D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587

AlphaGo results



AlphaGo vs human or related work

Ablation study of the components

Ablation study of distributed MCTS

D. Silver. Mastering the game of Go with Deep Neural Networks and Tree Search. Nature, vol. 529 issue 7587