# Deep networks for model-based reinforcement learning

UPenn CIS 522 Guest Lecture | April 2020

Timothy Lillicrap
Research Scientist, DeepMind & UCL

# Outline

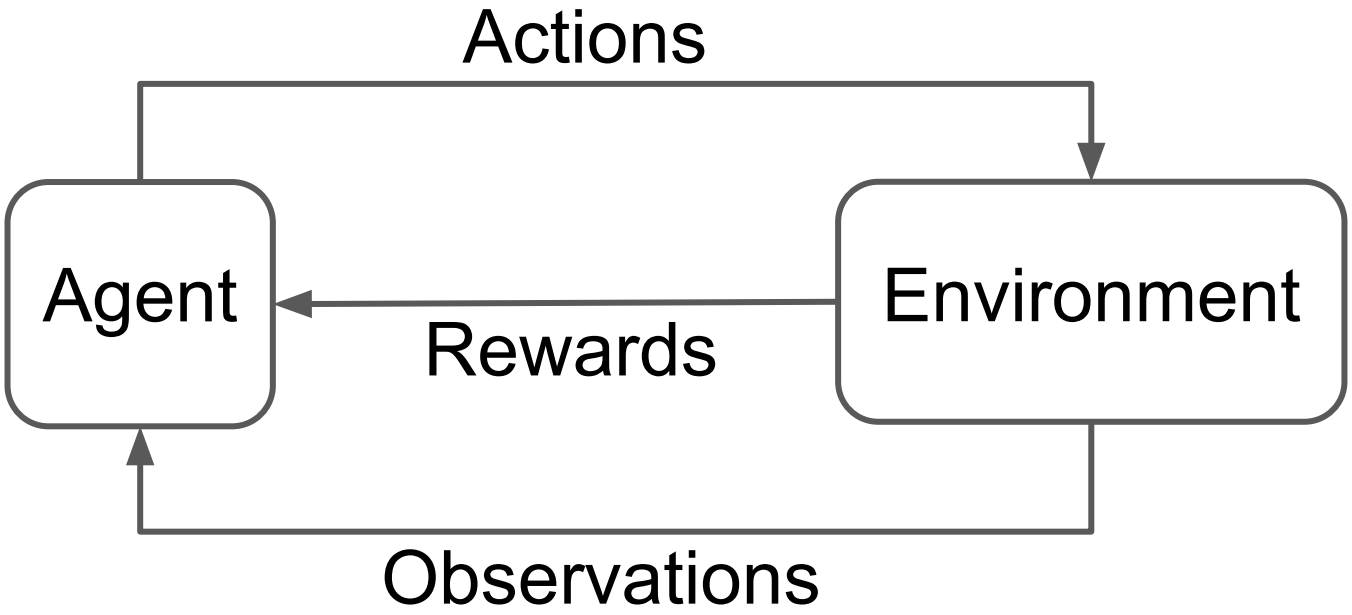
- Understand what is meant by model-free and model-based (and why this distinction is blurry).

- Trace the path from *AlphaGo* to *AlphaGoZero* to *AlphaZero* to *MuZero*

  David Silver, Aja Huang, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Karen Simonyan & the AlphaZero Group

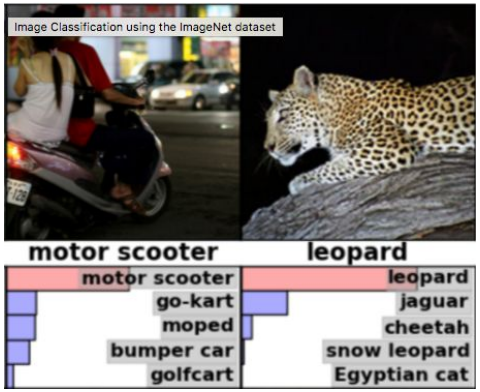
- Discuss some of the limitations of these approaches and where the field might go next.

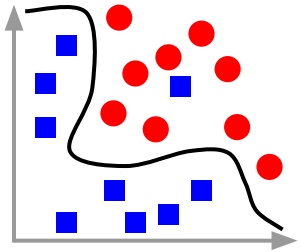
# What is Reinforcement Learning?



# What is Reinforcement Learning?

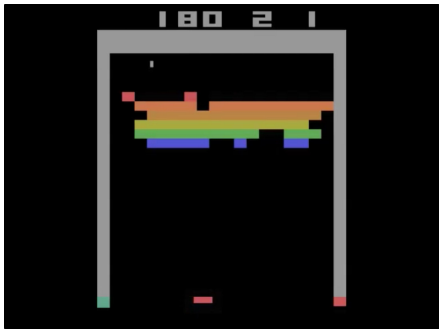
## Supervised Learning

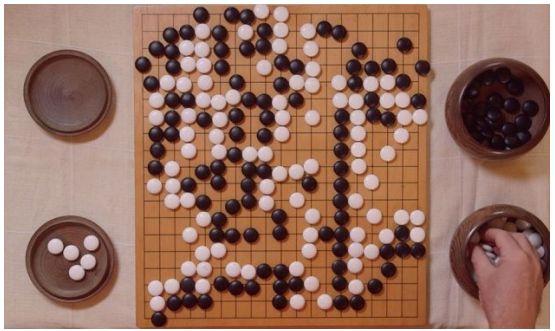


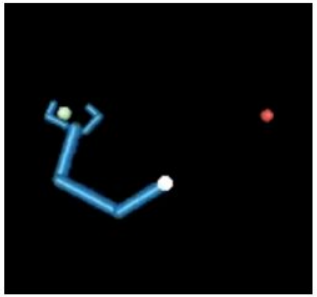



**Fixed dataset**

## Reinforcement Learning

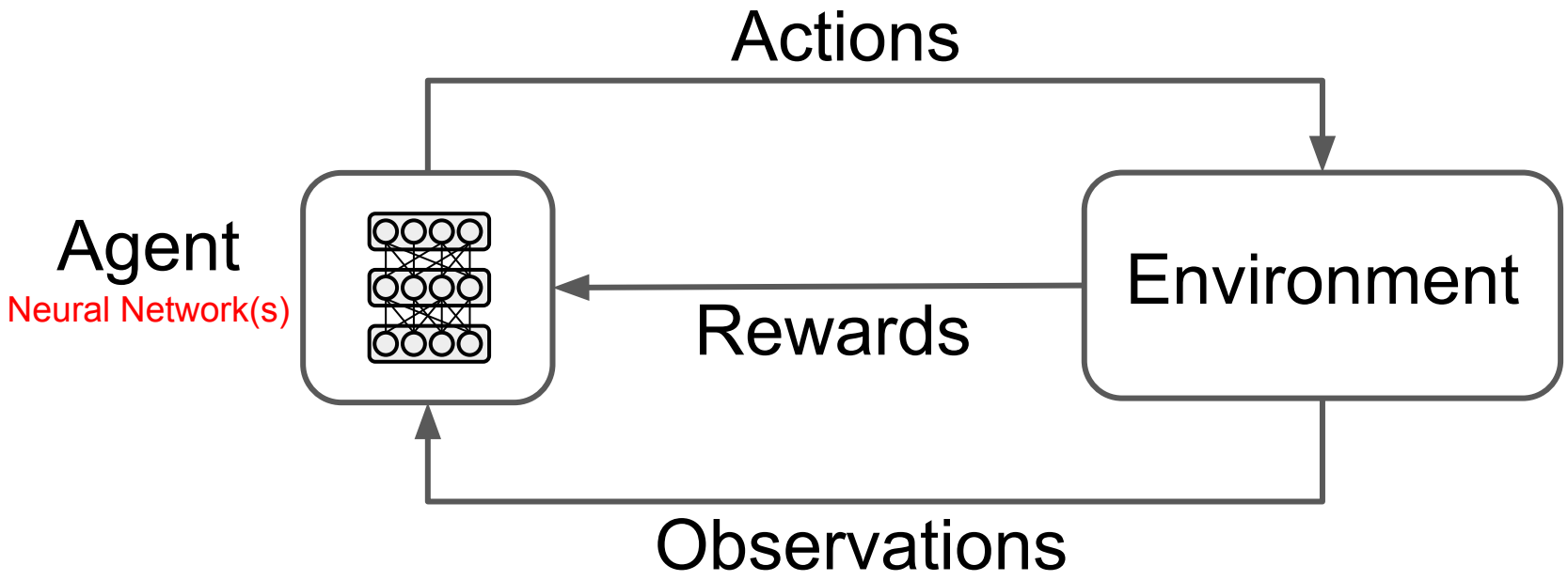


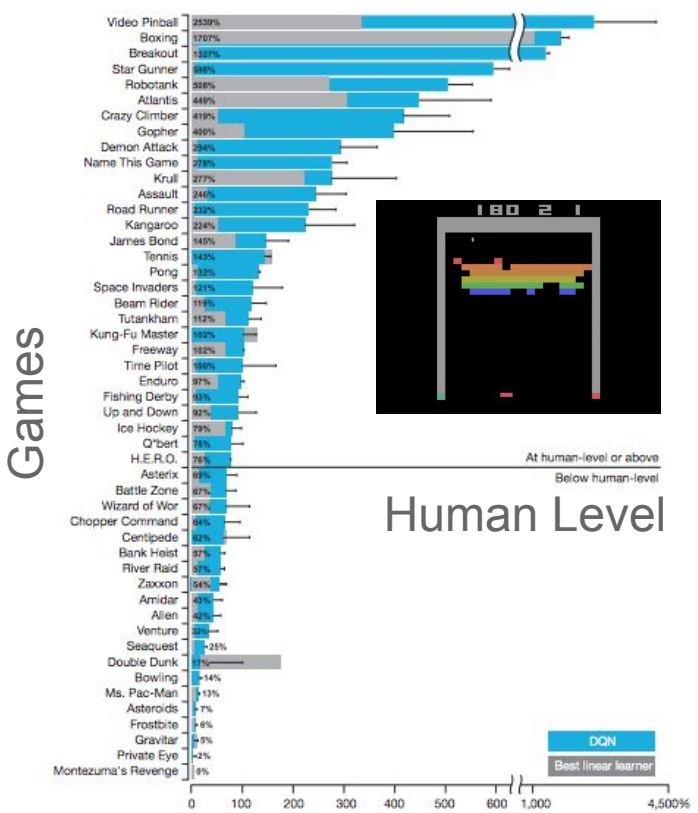


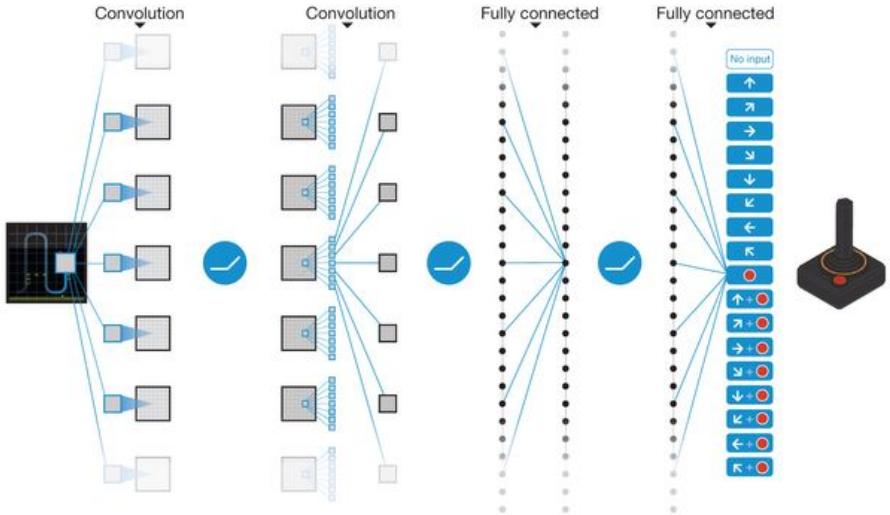
**Data depends on actions taken in environment**

# What is Deep Reinforcement Learning?



# Playing Atari with a Deep Network (DQN)




Same hyperparameters for all games!

Mnih et al., *Nature* 2015

# Current State and Limitations of Deep RL

We can now solve virtually *any single task/problem* for which we can:

(1) Formally specify and query the reward function.

(2) *Explore sufficiently* and collect *lots of data*.

# Current State and Limitations of Deep RL

We can now solve virtually *any single task/problem* for which we can:

(1) Formally specify and query the reward function.

(2) *Explore sufficiently* and collect *lots of data*.

What remains challenging:

(1) Learning when a reward function is difficult to specify.

(2) Data efficiency & multi-task / transfer learning.

# Current State and Limitations of Deep RL

We can now solve virtually *any single task/problem* for which we can:

(1) Formally specify and query the reward function.

(2) *Explore sufficiently* and collect *lots of data*.

What remains challenging:

(1) Learning when a reward function is difficult to specify.

(2) Data efficiency & multi-task / transfer learning.

# Plan with a Model?

Training a model is an obvious way to introduce structure into our agents:

$$\rho(s_{t+1}|s_t, a_t)$$

- Data Efficiency
- Allows us to Spend Compute for Performance
- Generalization and Transfer

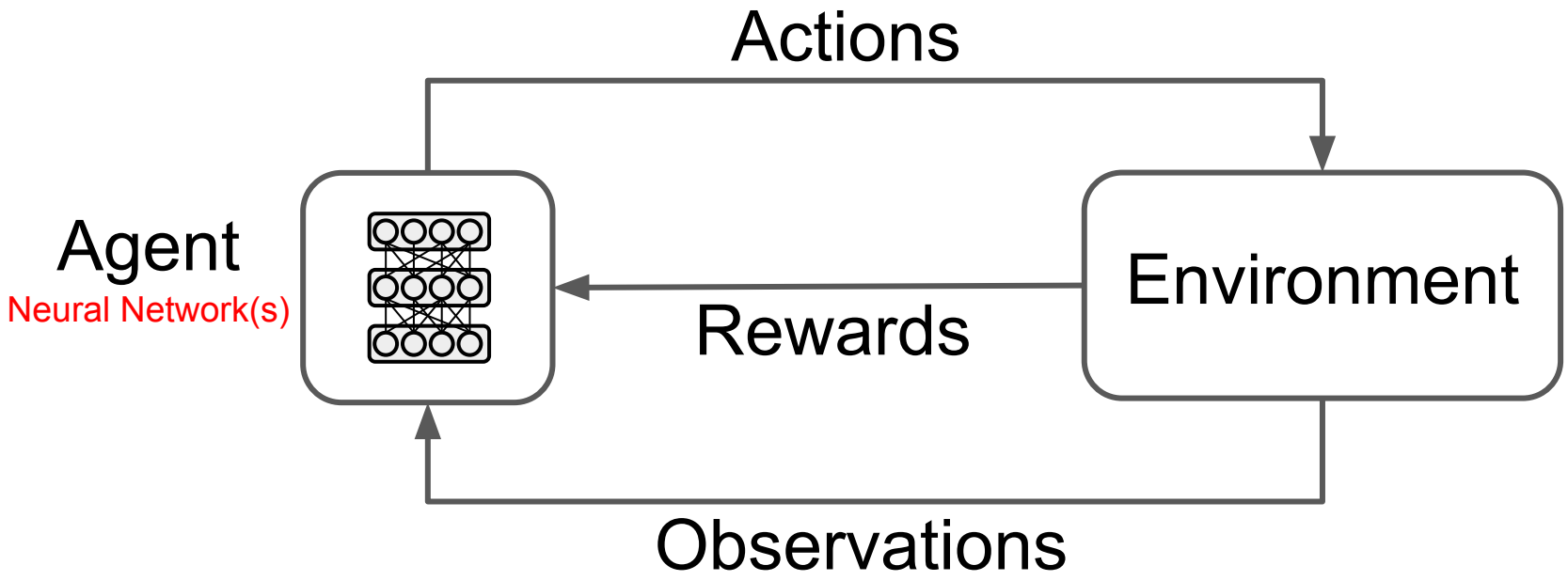
# Planning with Learned Models has Been Tricky

Why is it dangerous to plan with a learned model?:

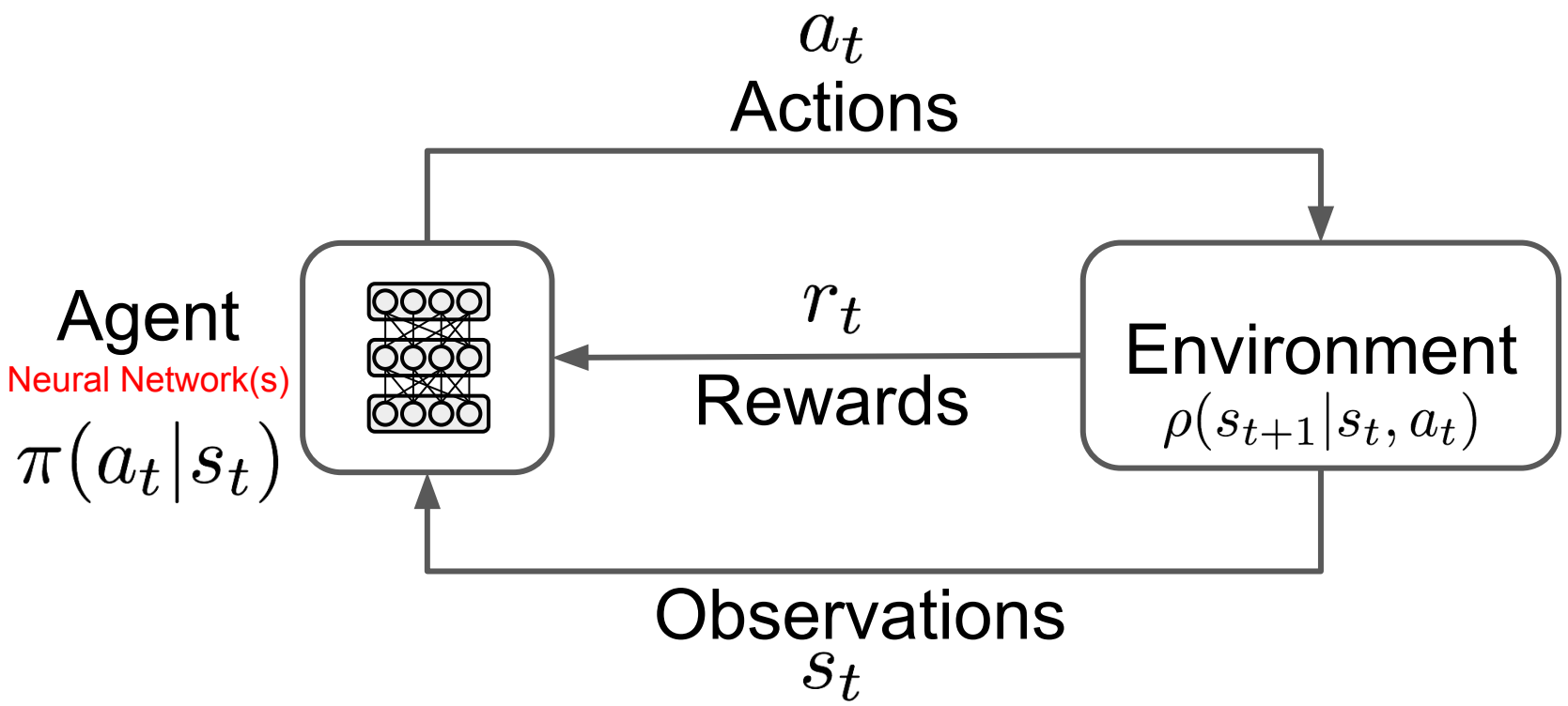
Planning is inherently a maximization operation that looks for a sequence of actions that make the expected return better.

Planning will search for and exploit imperfections in a learned model!

# Deep Reinforcement Learning



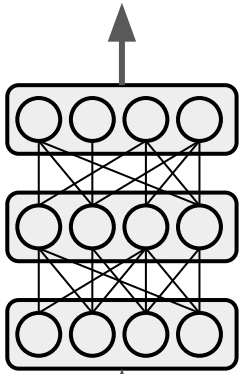
# Formalizing the Agent-Environment Loop



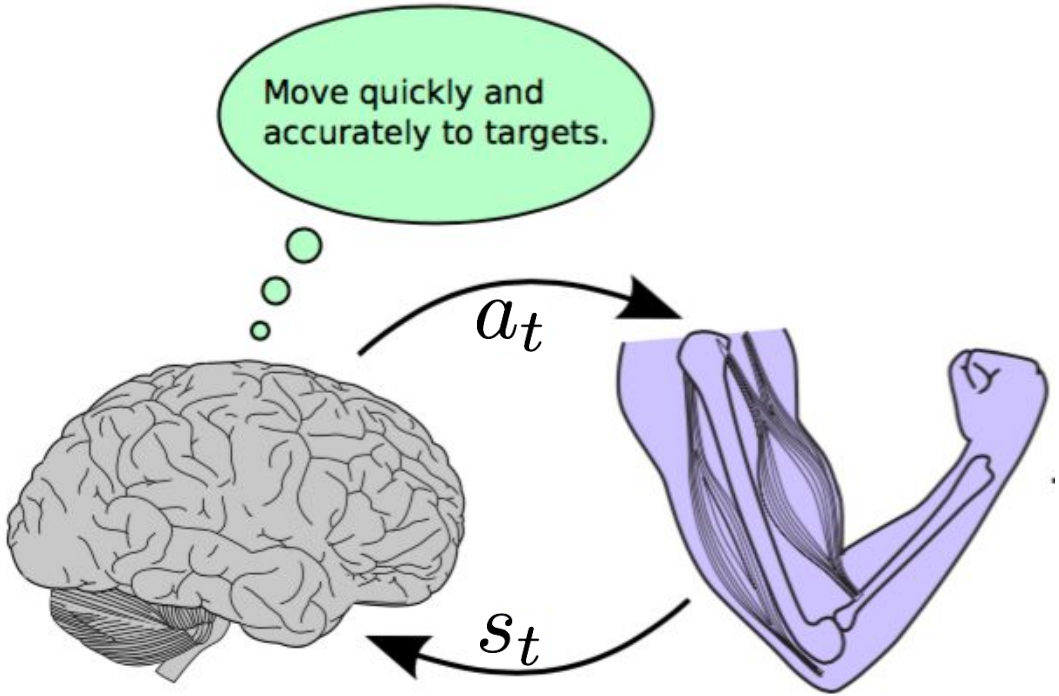
# Formalizing the Agent-Environment Loop

$$a_t \sim \pi_\theta(a_t|s_t)$$

$$\theta = \begin{bmatrix} W_1 \\ W_0 \end{bmatrix}$$



$s_t$



# A Single Trial

$$
\begin{array}{ccccc}
r_0, & r_1, & r_2, & \ldots, & r_T \\
a_0, & a_1, & a_2, & \ldots, & a_T \\
\pi(a_0|s_0), & \pi(a_1|s_1), & \pi(a_2|s_2), & \ldots, & \pi(a_T|s_T) \\
s_0, & s_1, & s_2, & \ldots, & s_T \\
 & \rho(s_1|s_0,a_0), & \rho(s_2|s_1,a_1), & \ldots, & \rho(s_T|s_{T-1},a_{T-1})
\end{array}
$$

Time ⟶

Probability of trajectory $\tau$

$$p_\theta(\tau) = \rho(s_0) \prod_{t=0}^{T} \rho(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

# Measuring Outcomes

Return for a single trial:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$$

Objective function:

$$J(\theta) = \int_{\mathbb{T}} p_\theta(\tau) R(\tau) d\tau$$

# Update Parameters with the Policy Gradient

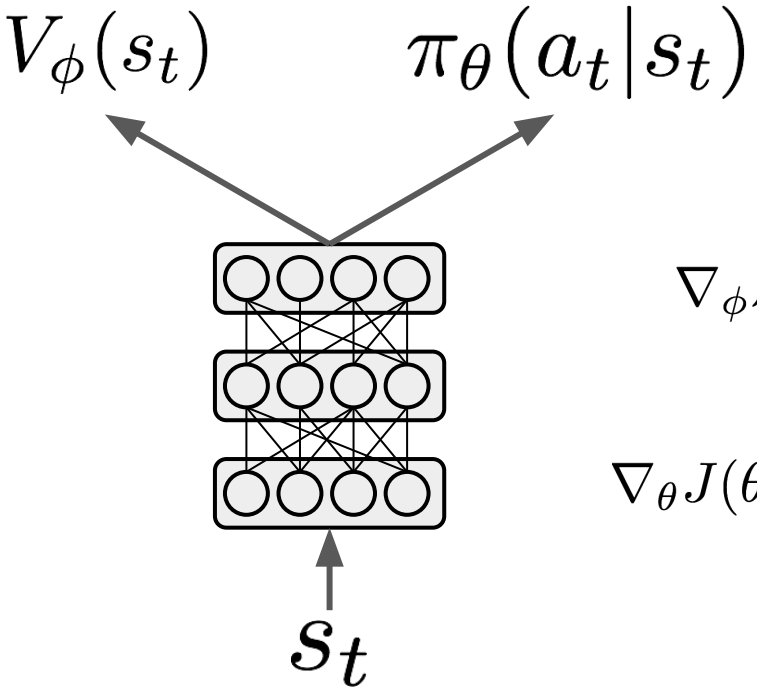
1. Sample a trajectory by rolling out the policy:

$$\tau \sim \begin{matrix} r_0, & r_1, & r_2, & ..., & r_T \\ a_0, & a_1, & a_2, & ..., & a_T \\ \pi(a_0|s_0), & \pi(a_1|s_1), & \pi(a_2|s_2), & ..., & \pi(a_T|s_T) \\ s_0, & s_1, & s_2, & ..., & s_T \\ & \rho(s_1|s_0,a_0), & \rho(s_2|s_1,a_1), & ..., & \rho(s_T|s_{T-1},a_{T-1}) \end{matrix}$$

2. Compute an estimate of the policy gradient and update network parameters:

$$\theta_{i+1} = \theta_i + \eta \nabla_\theta \hat{J(\theta)}\Big|_{\theta=\theta_i}$$

# Combating Variance: Advantage Actor-Critic

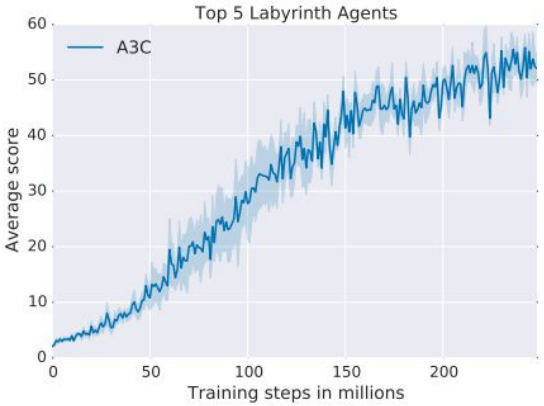



$$\nabla_\phi \mathcal{L} = \sum_{t=0}^{T} \nabla_\phi (R_t - V_\phi(s_t))^2$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[ \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) \right]$$

Mnih et al., *ICML* 2016

# Training Neural Networks with Policy Gradients

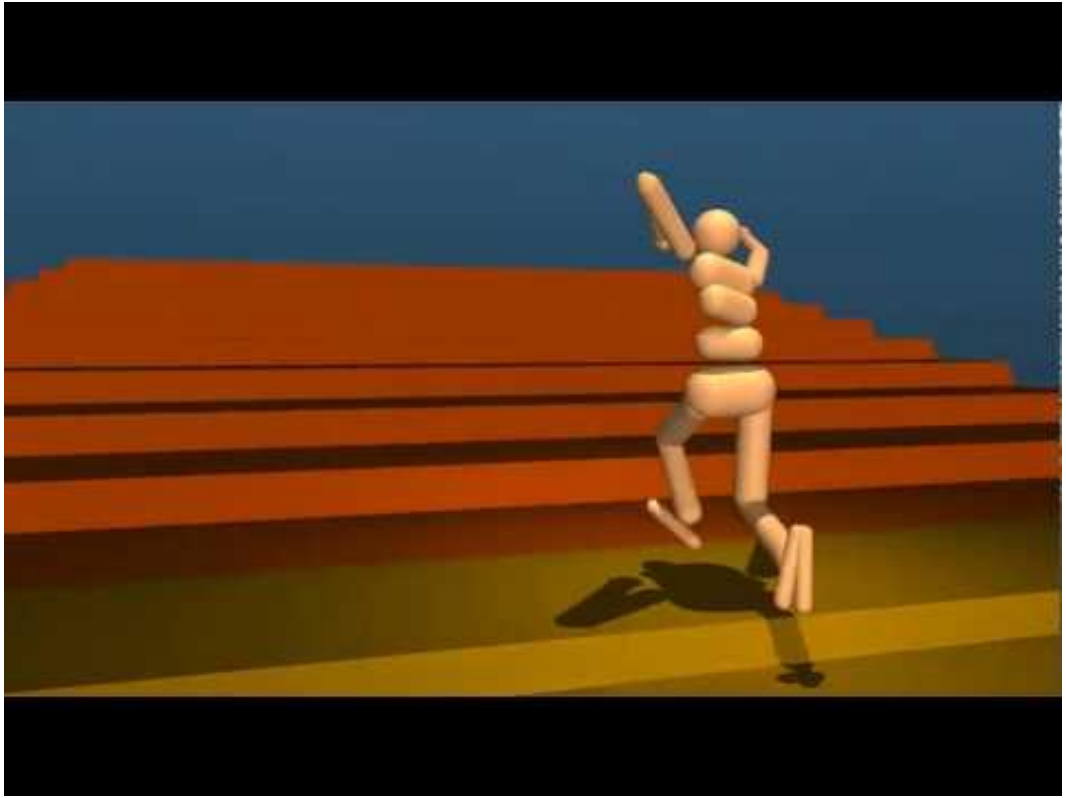



100s of Millions of steps!

The policy gradient has high variance.

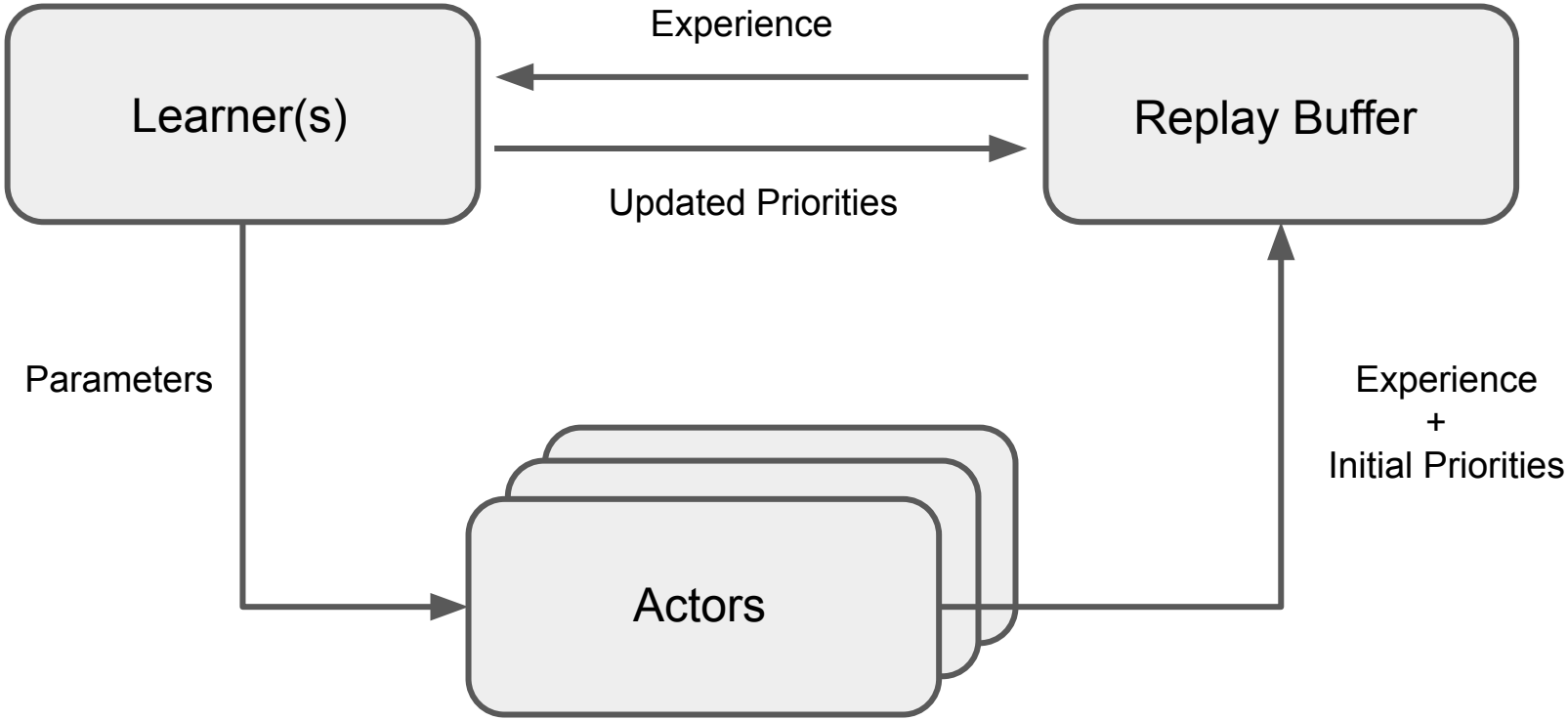

Mnih et al., *ICML* 2016

# Proximal Policy Gradient for Flexible Behaviours



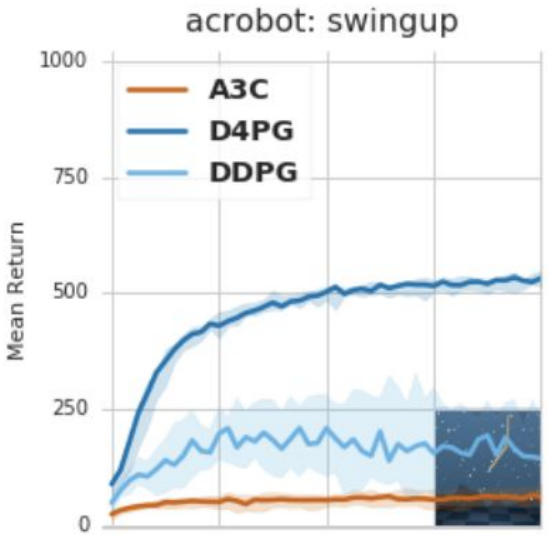
Heess et al., 2017

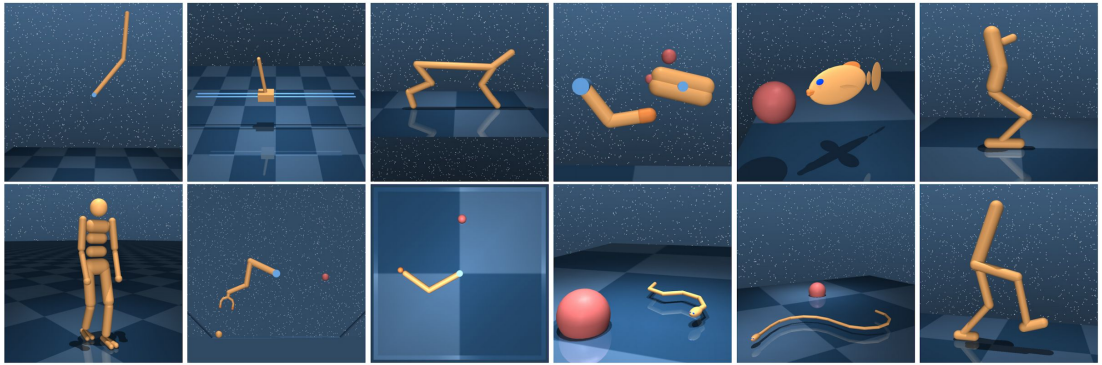
# Off-policy Approaches with Replay?




Horgan et al., 2017 & Schaul et al. 2015 + Many Others

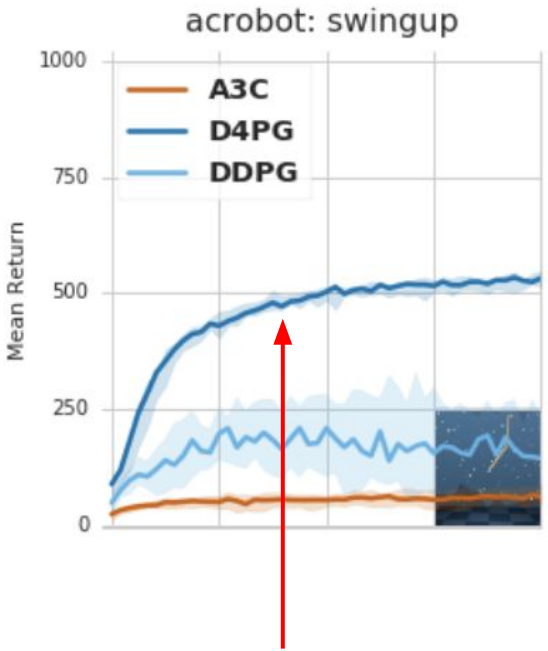
# Off-Policy Learning with D4PG



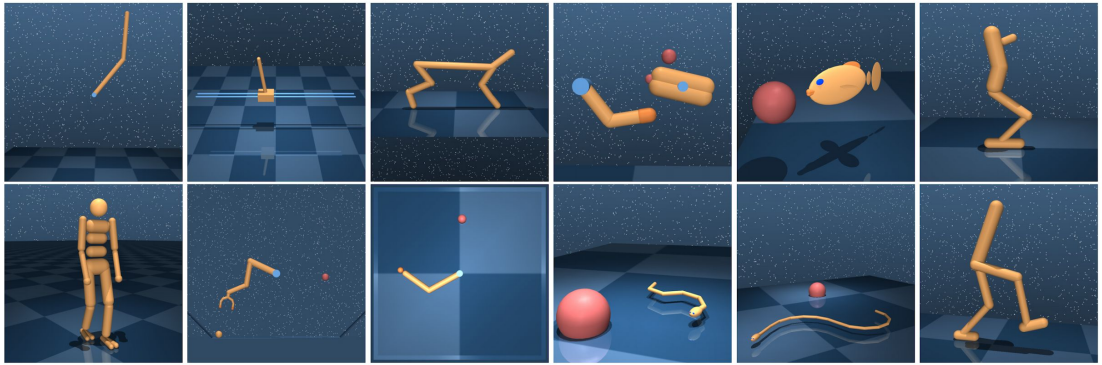



Hoffman, Barth-Maron et al., 2017; Tassa et al. 2018

# Off-Policy Learning with D4PG




Still 10,000s of Episodes



Hoffman, Barth-Maron et al., 2017; Tassa et al. 2018

# Plan with a Model?

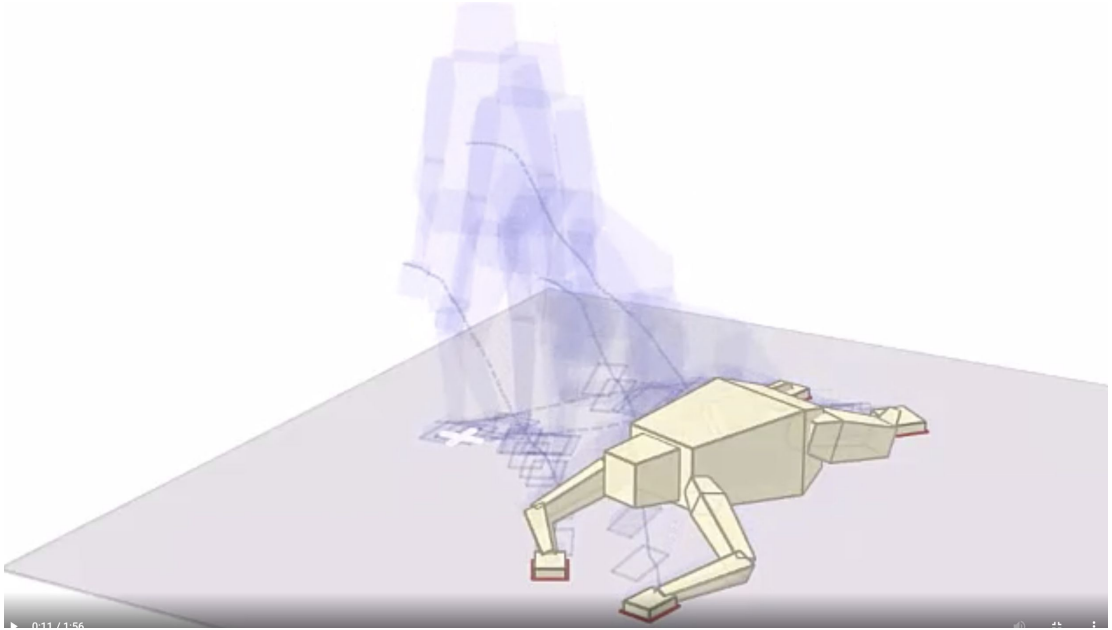
Training a model is an obvious way to introduce structure into our agents:

$$\rho(s_{t+1}|s_t, a_t)$$

- Data Efficiency
- Allows us to Spend Compute for Performance
- Generalization and Transfer

# Plan with a Model?

Planning with a model is a compelling idea and has produced stunning results, especially in simulated physics:

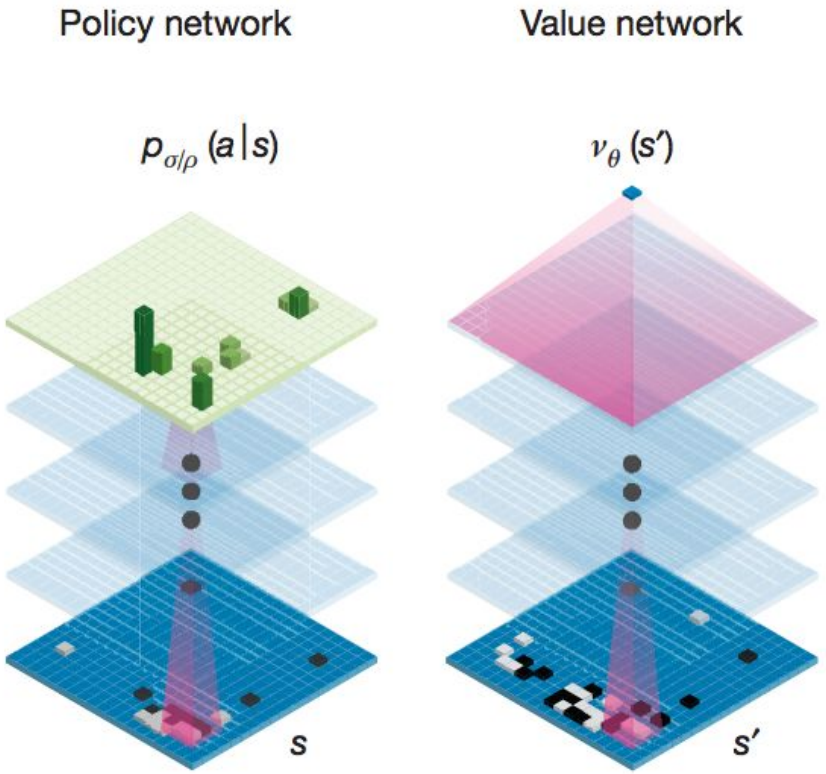


Mordatch et al, SIGGRAPH 2012.

# Combining Deep Networks with Models & Planning

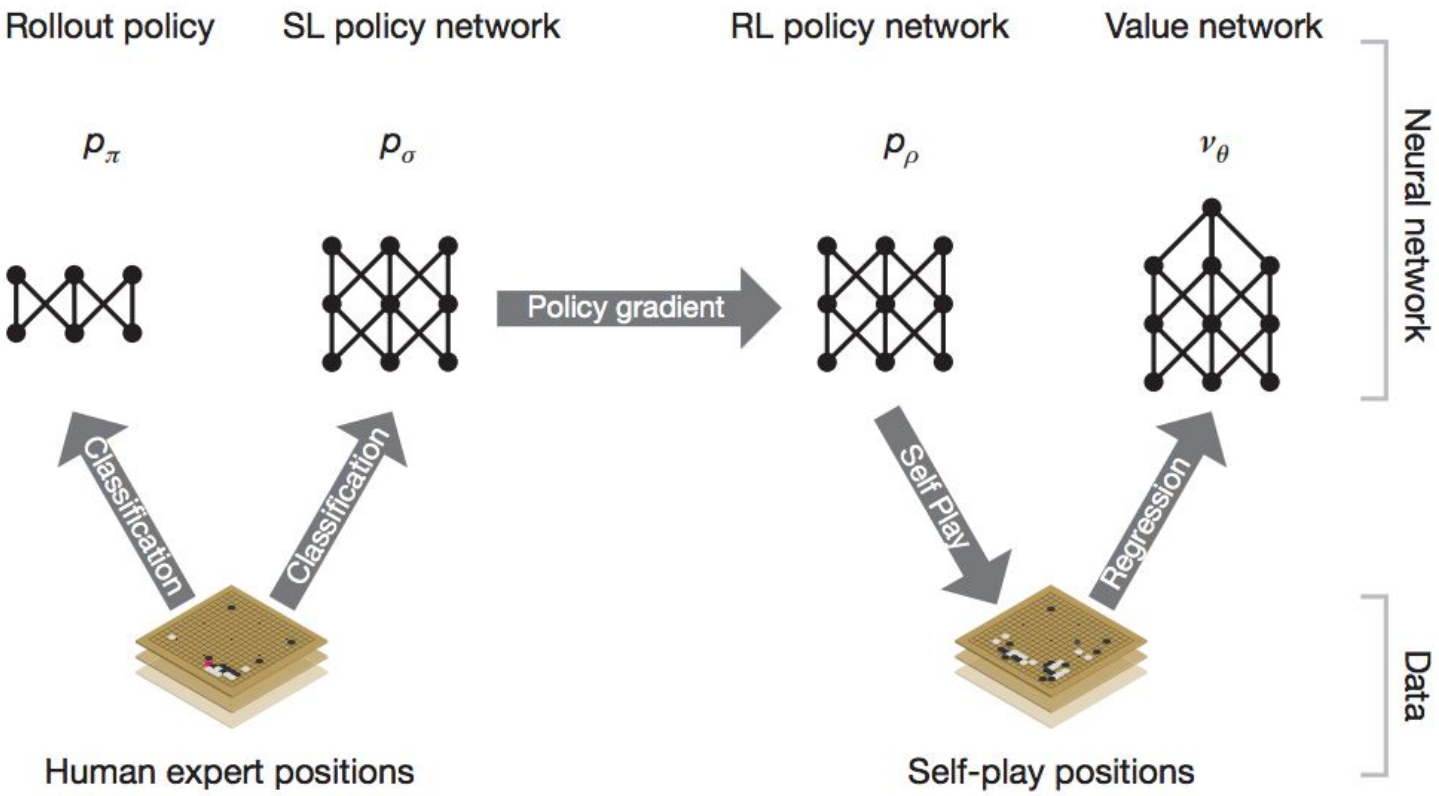
$$\rho(s_{t+1}|s_t, a_t)$$

Use environment model to plan!



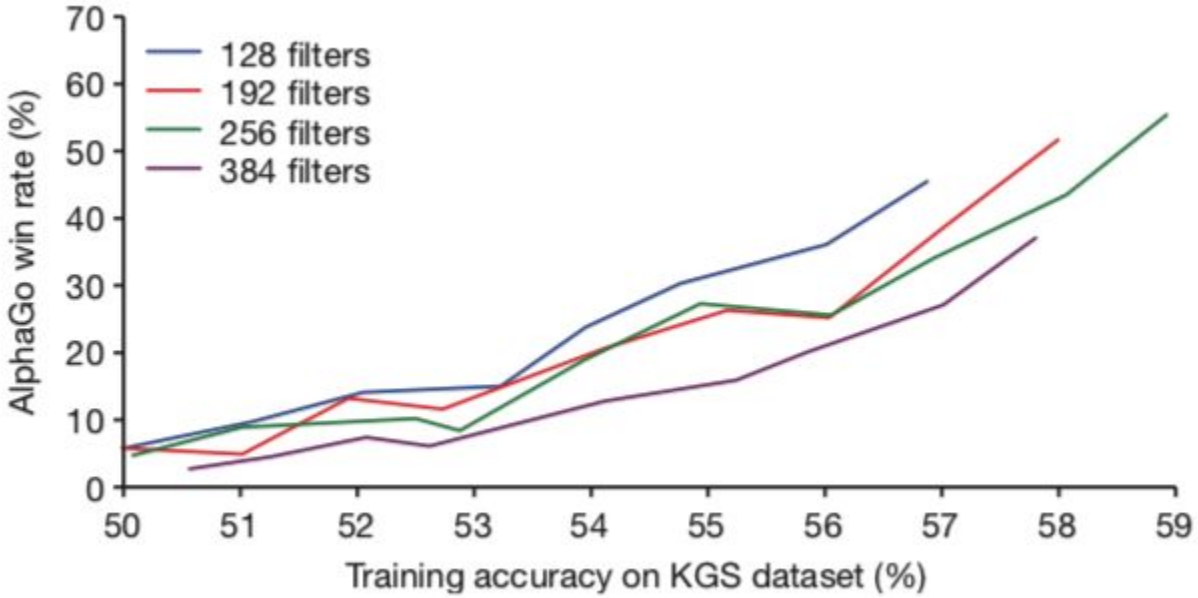

Silver, Huang et al., *Nature*, 2016

# Training Policy and Value Networks



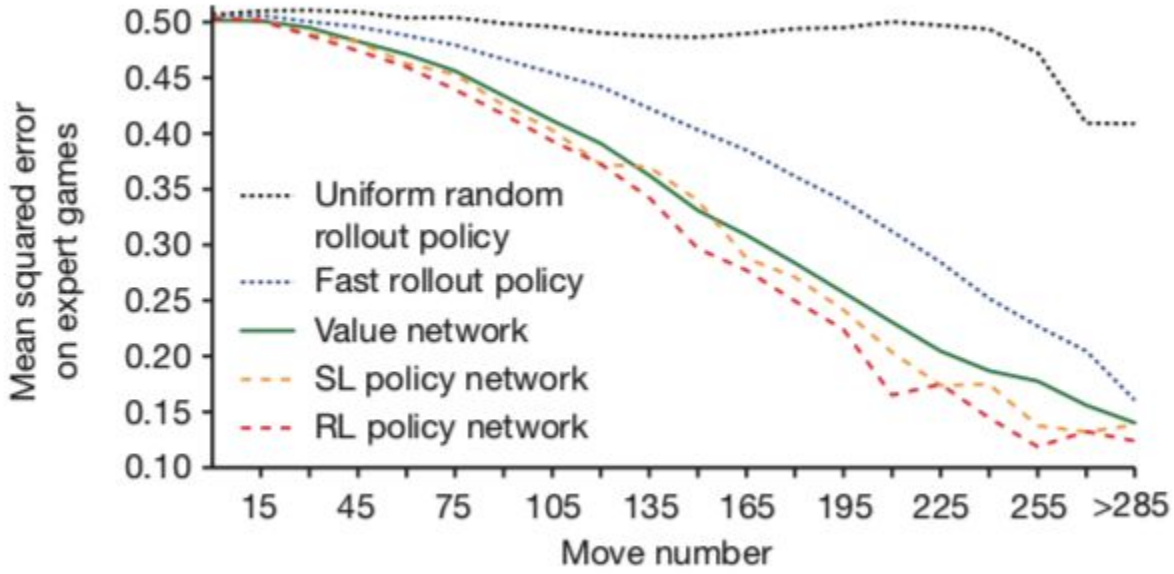

Silver, Huang et al., *Nature*, 2016

# Small Accuracy Diffs Lead to Large Winrate Diffs



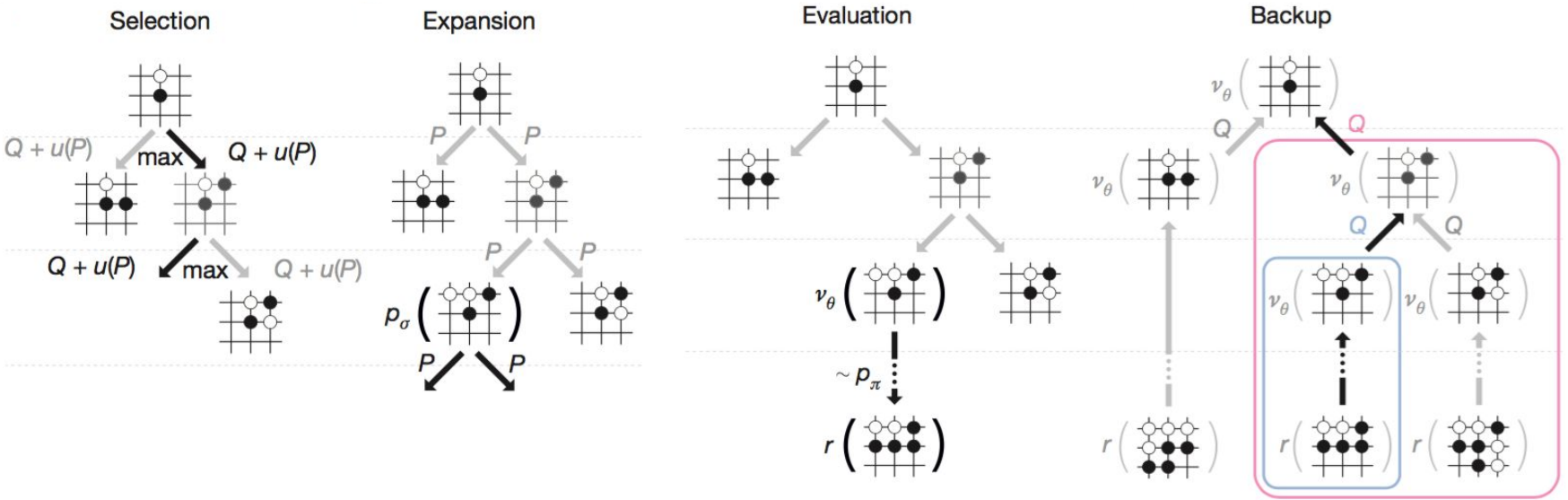

Silver, Huang et al., *Nature*, 2016

# Rollouts of Strong Networks Predict Value Better




Silver, Huang et al., *Nature*, 2016

# Planning with an Environment Model & MCTS




Silver, Huang et al., *Nature*, 2016

# A Quick Look at MCTS

***Selection:***
Traverse the existing search tree, selecting which edge to follow via: $\arg\max_a(Q(s_t, a) + u(s_t, a))$

Keep going until you reach a leaf node.

***Expansion:***
The leaf position is evaluated by the prior network and the probabilities are stored as the prior P(s,a)
Other counts and values are initialized to 0.

***Evaluation:***
Compute the value of the reached leaf position using the value function and fast rollout network

***Backup:***
Step backward through the traversed search path.
At each step, update the visit counts: N(s,a) += 1
Also at each step, update the estimated value: Q(s,a) += leaf value

Silver, Huang et al., *Nature*, 2016

# A Quick Look at MCTS

MCTS efficiently trades exploration and exploitation over the course of search:

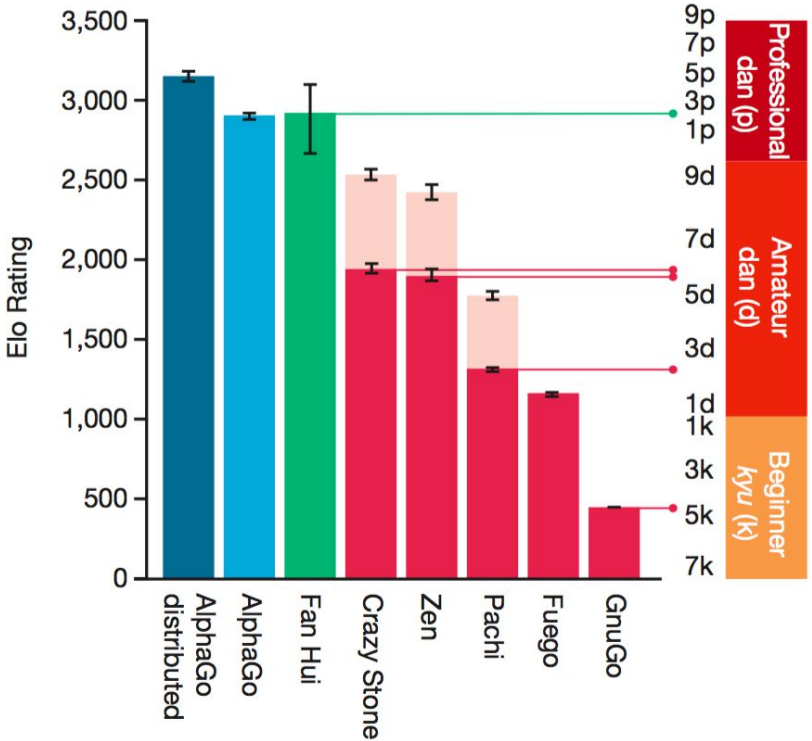
$$\arg\max_a (Q(s_t, a) + u(s_t, a))$$

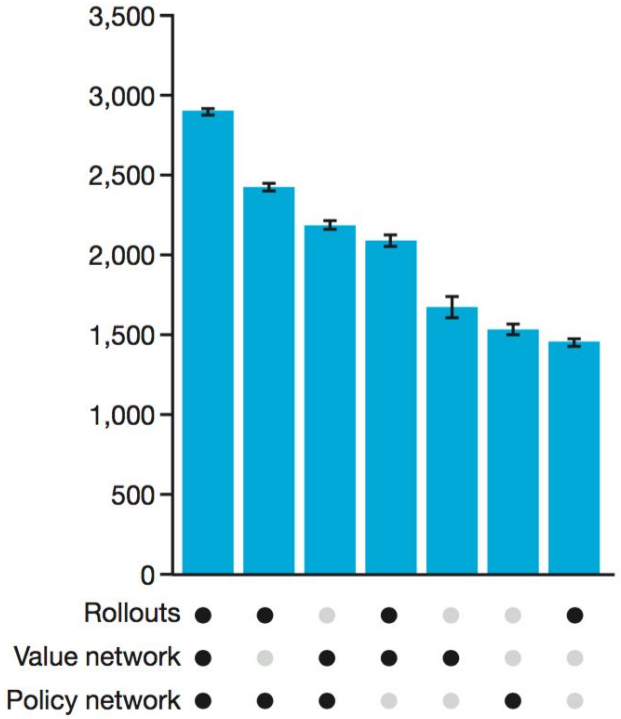
$$u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$$

- Initially prefers actions with higher prior and low counts.
- Asymptotically prefers actions with high estimated value.
- UCB variant algorithm developed in the context of multi-armed bandits.

Silver, Huang et al., *Nature*, 2016

# Planning with an Environment Model

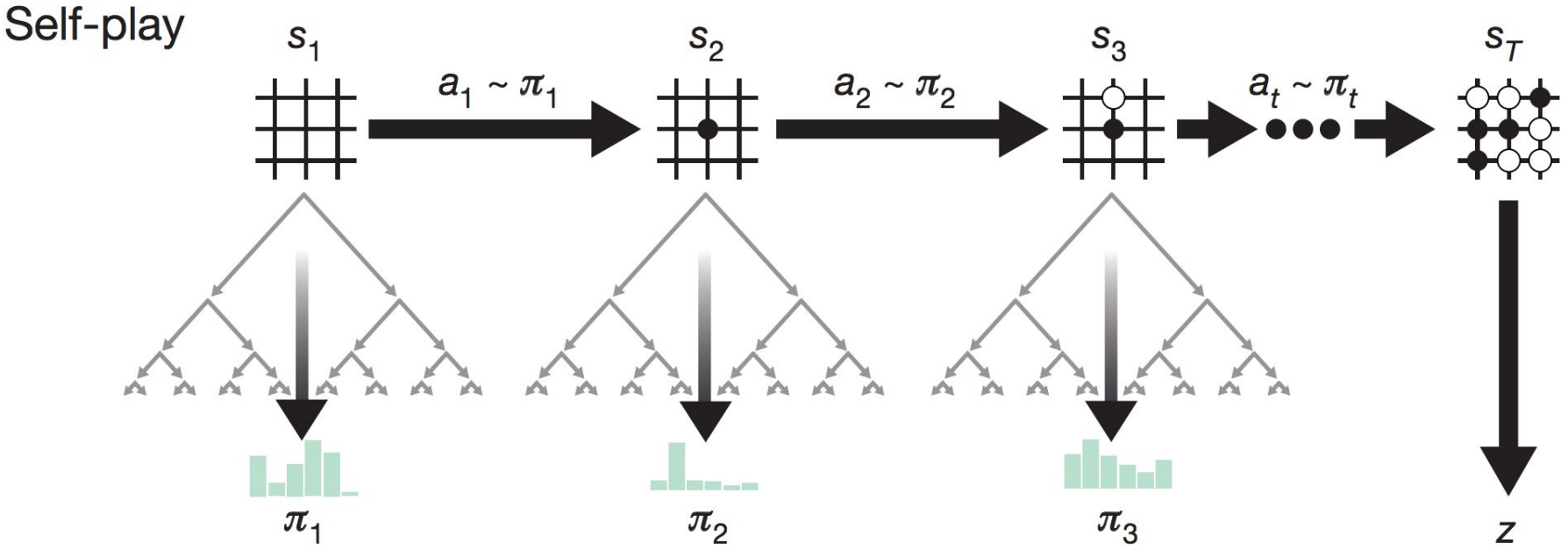



$$p(a \text{ defeats } b) = (1 + 10^{c_{\text{elo}}(e(b) - e(a))})^{-1}$$

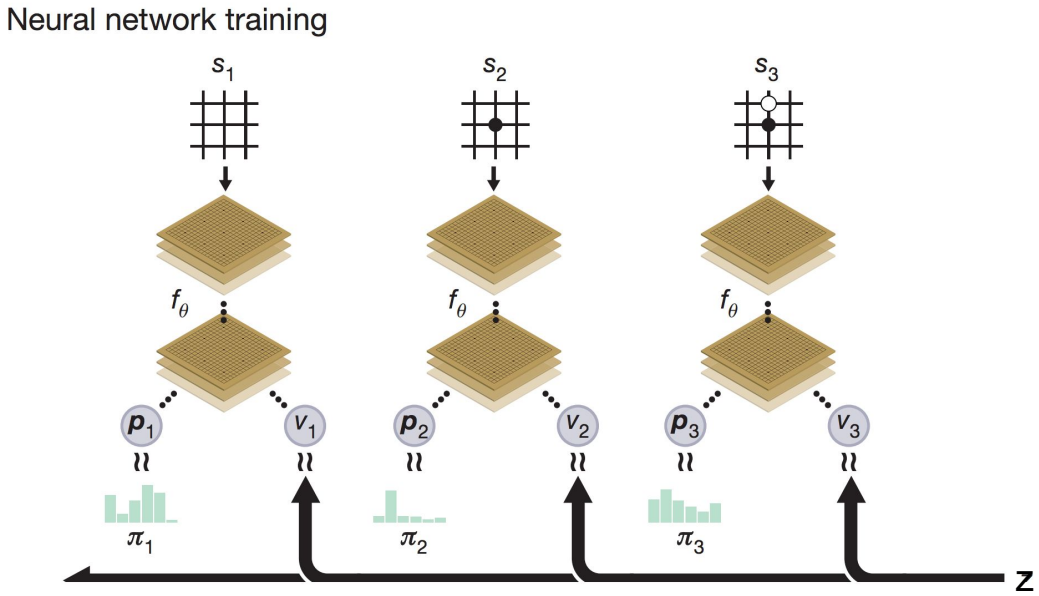
Silver, Huang et al., *Nature*, 2016

# Playing Go with Without Human Knowledge




Silver, Schrittwieser, Simonyan, et al. *Nature*, 2017

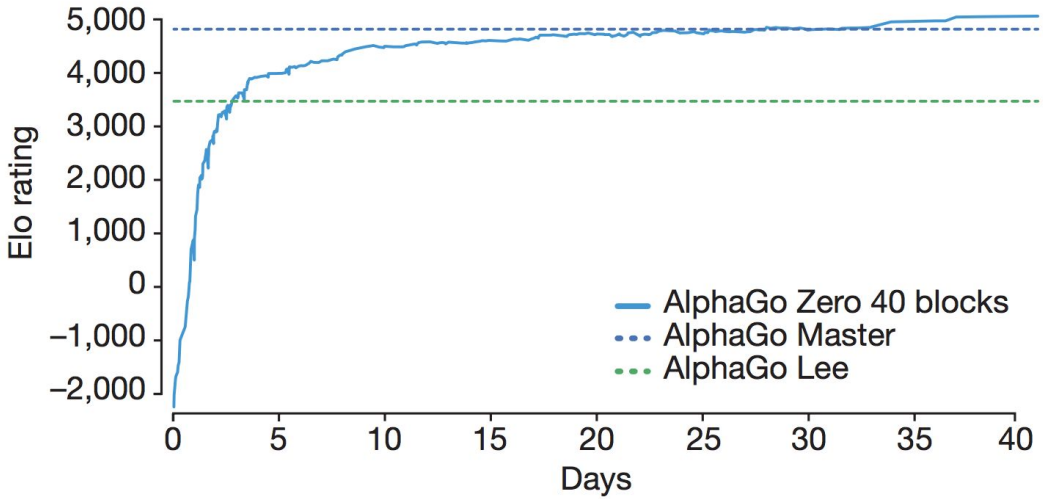
# Playing Go with Without Human Knowledge

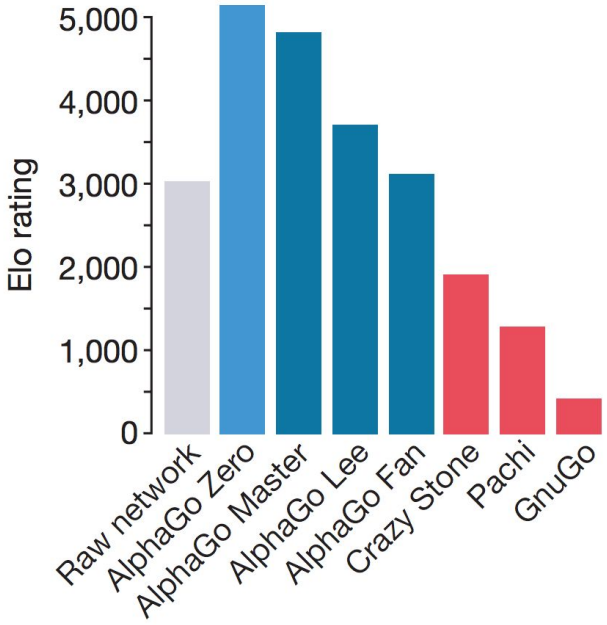



$$(\boldsymbol{p}, v) = f_\theta(s) \quad \text{and} \quad l = (z - v)^2 - \boldsymbol{\pi}^{\mathrm{T}} \log \boldsymbol{p} + c\|\theta\|^2$$

Silver, Schrittwieser, Simonyan, et al. *Nature*, 2017

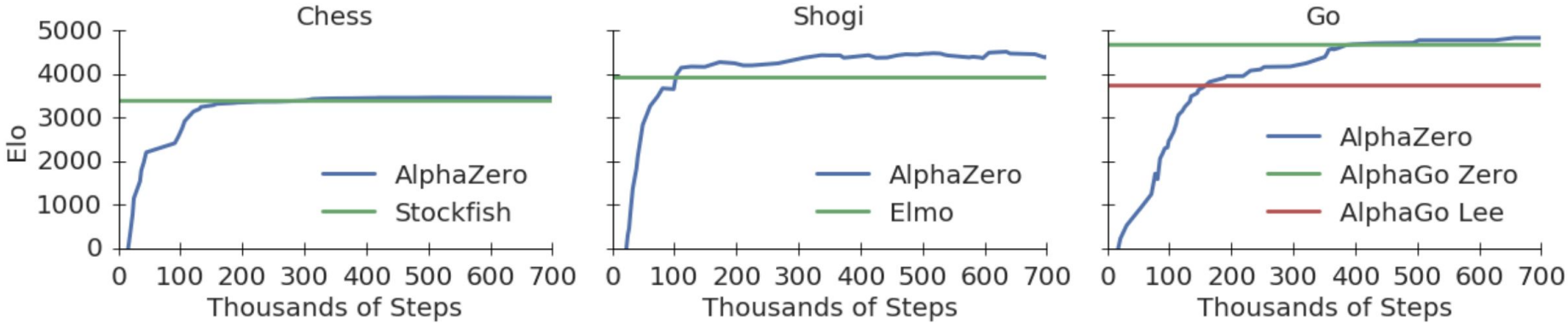
# Playing Go with Without Human Knowledge




Silver, Schrittwieser, Simonyan, et al. *Nature*, 2017

# Playing Go, chess and shogi with Deep RL



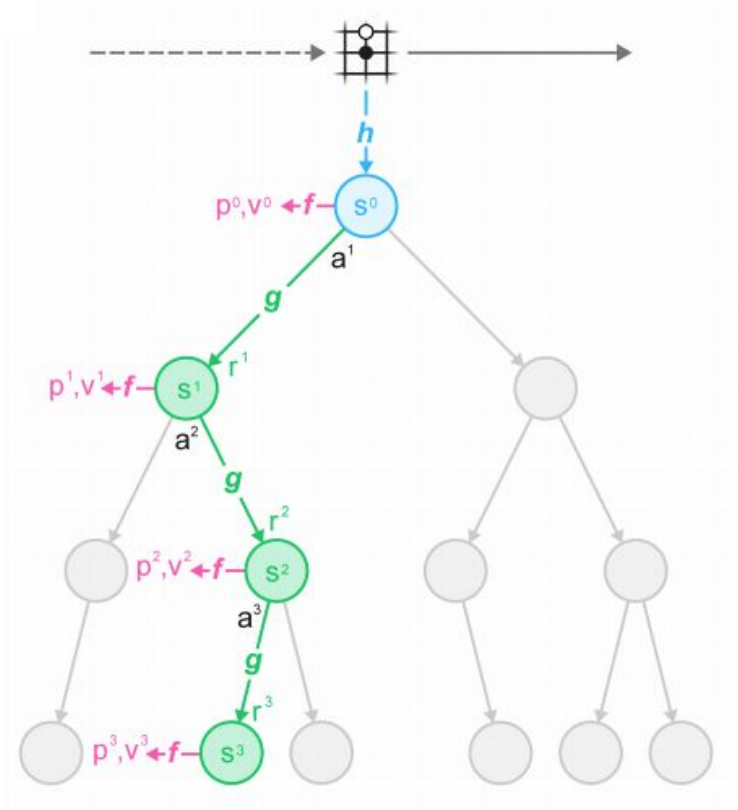

Silver, Hubert, Schrittwieser, et al. *Nature*, 2017

# Chess, Go, shogi & Atari with a Learned Model

- AlphaGo & AlphaZero use perfect models of the environment
- MuZero builds a model of the environment that can be used to search
- Unlike most model-based work MuZero does not try to model environment transitions
- Rather, it focuses predicting the future reward, value, and policy
- These are the essential components required by MCTS

Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

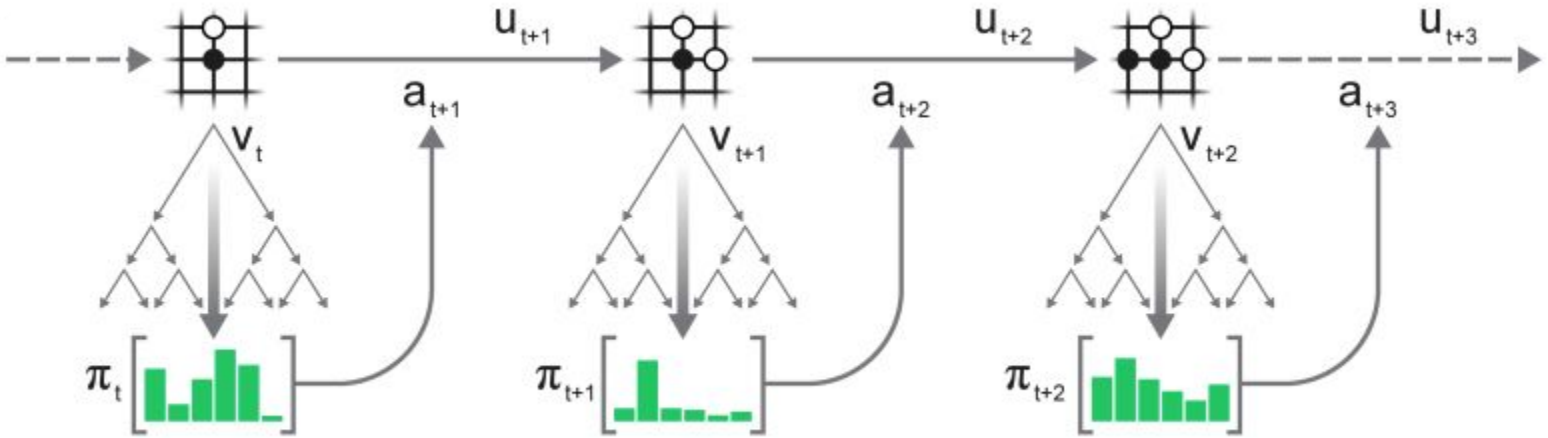
# *MuZero*: Model Design




Note that the transposition tables used in *AlphaGo* and *AlphaZero* are not possible in *MuZero*.

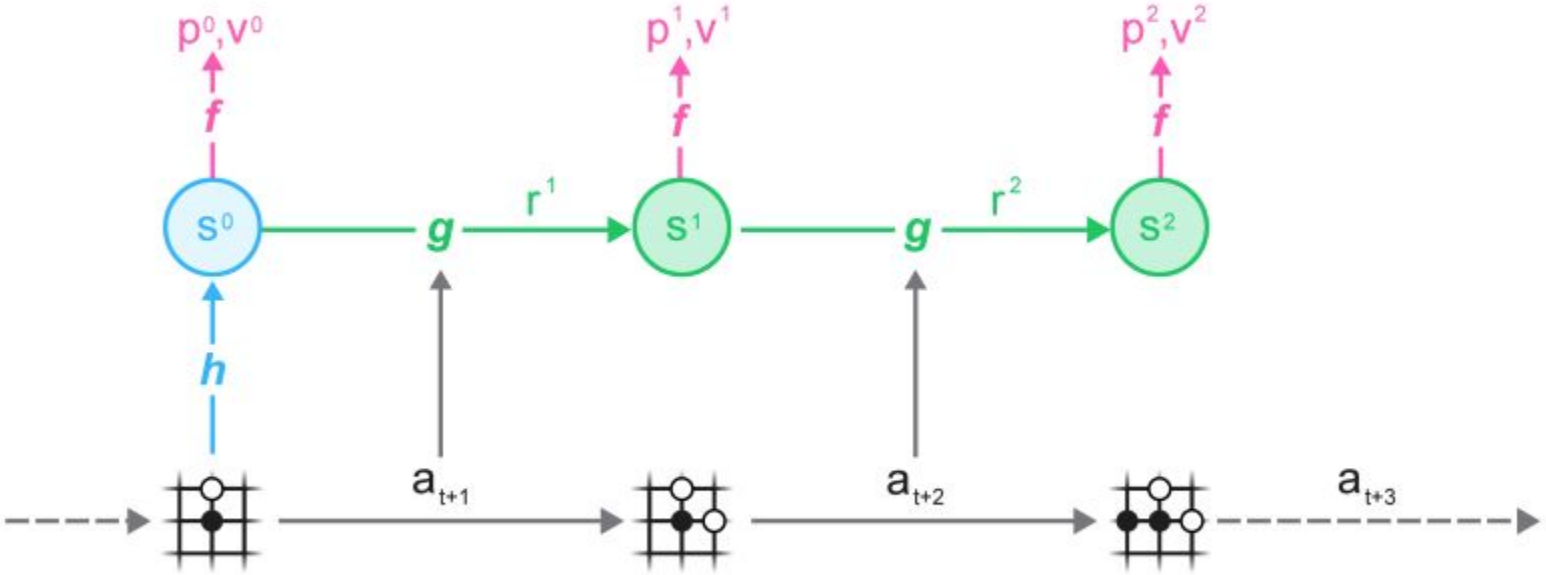
Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# *MuZero*: How it acts



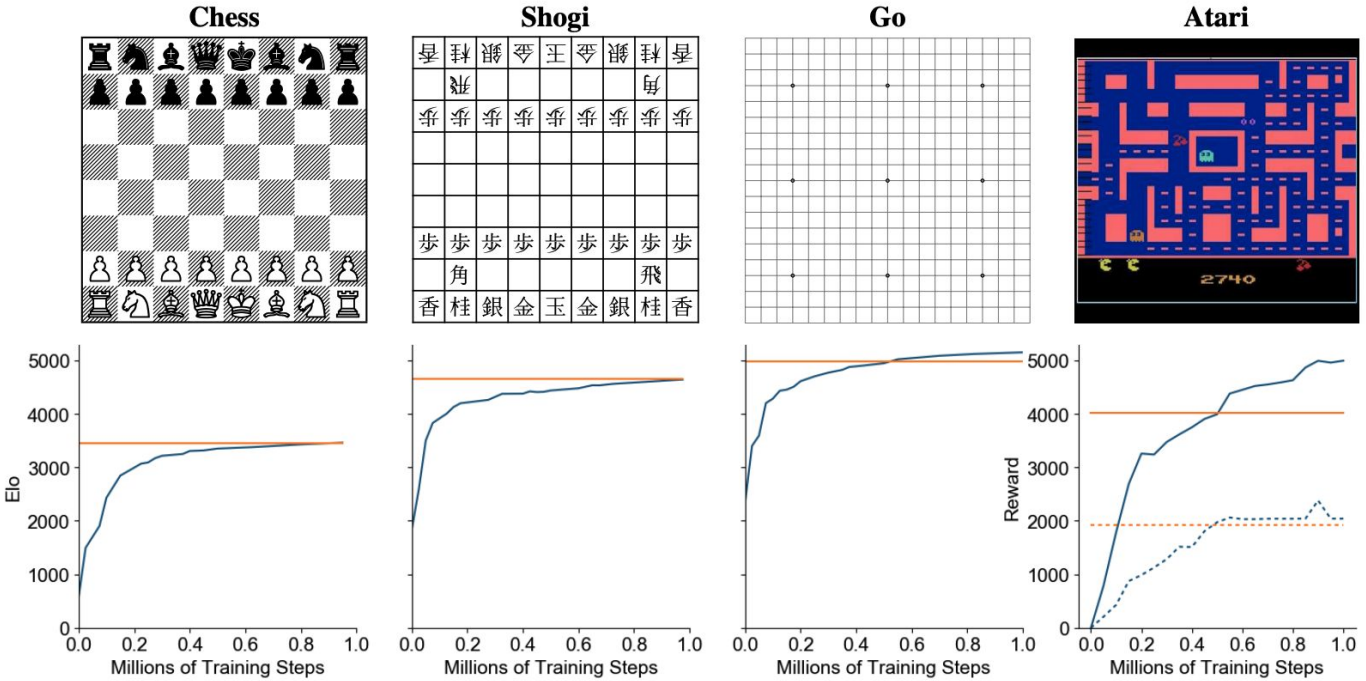

Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# *MuZero*: How it learns




Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# *MuZero*: Main Results




Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# *Muzero:* Differences from *AlphaGo* / *AlphaZero*

>PUCT variant

>Single player games can emit intermediate rewards at any timestep

- Backups are generalized to incorporate rewards along the path

$$G^k = \sum_{\tau=0}^{l-1-k} \gamma^\tau r_{k+1+\tau} + \gamma^{l-k} v^l$$

- *Muzero* computes normalized Q values using empirical min/max

$$Q(s^{k-1}, a^k) := \frac{N(s^{k-1}, a^k) \cdot Q(s^{k-1}, a^k) + G^k}{N(s^{k-1}, a^k) + 1}$$

$$N(s^{k-1}, a^k) := N(s^{k-1}, a^k) + 1$$

$$\overline{Q}(s^{k-1}, a^k) = \frac{Q(s^{k-1}, a^k) - \min_{s,a \in Tree} Q(s, a)}{\max_{s,a \in Tree} Q(s, a) - \min_{s,a \in Tree} Q(s, a)}$$

Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# *MuZero*: Results & Reanalyze

| Agent | Median | Mean | Env. Frames | Training Time | Training Steps |
|---|---|---|---|---|---|
| Ape-X [18] | 434.1% | 1695.6% | 22.8B | 5 days | 8.64M |
| R2D2 [21] | 1920.6% | 4024.9% | 37.5B | 5 days | 2.16M |
| *MuZero* | **2041.1%** | **4999.2%** | 20.0B | 12 hours | 1M |
| IMPALA [9] | 191.8% | 957.6% | 200M | – | – |
| Rainbow [17] | 231.1% | – | 200M | 10 days | – |
| UNREAL[a] [19] | 250%[a] | 880%[a] | 250M | – | – |
| LASER [36] | 431% | – | 200M | – | – |
| *MuZero Reanalyze* | **731.1%** | **2168.9%** | 200M | 12 hours | 1M |

Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# *Muzero Reanalyze:*

> Revisit past timesteps from the replay buffer

> Rerun MCTS on these past states using fresher network parameters

> Recompute targets for the policy and value function

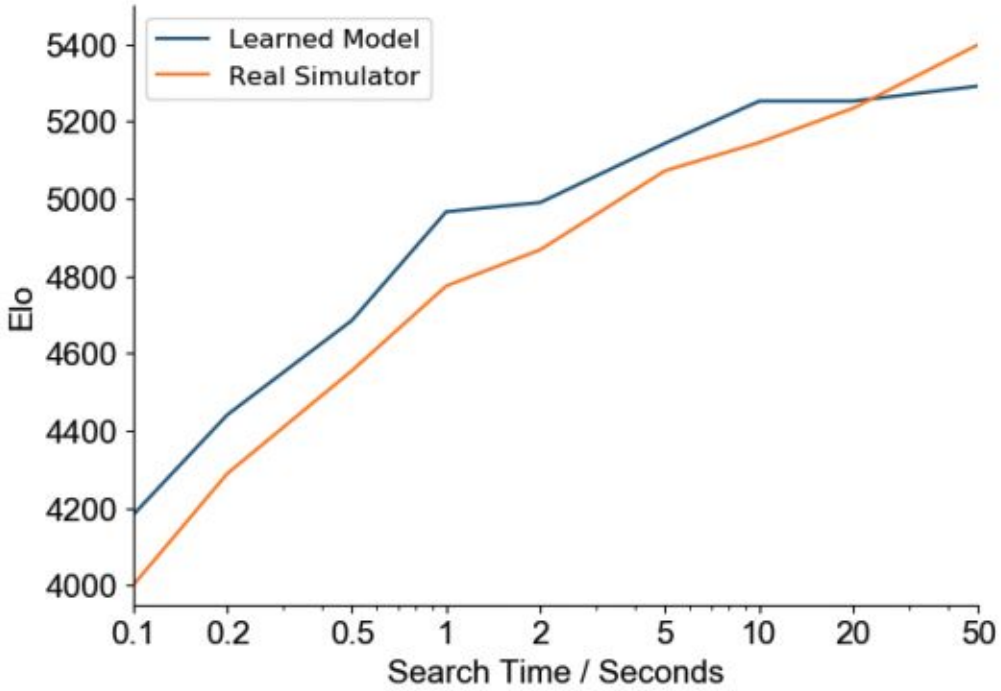
> Sample states from the replay buffer more of (2 instead of 0.1)

> Scale down value target loss (to ¼) versus 1 for policy head

> Alter n-steps used in backing up value (5 instead of 10)

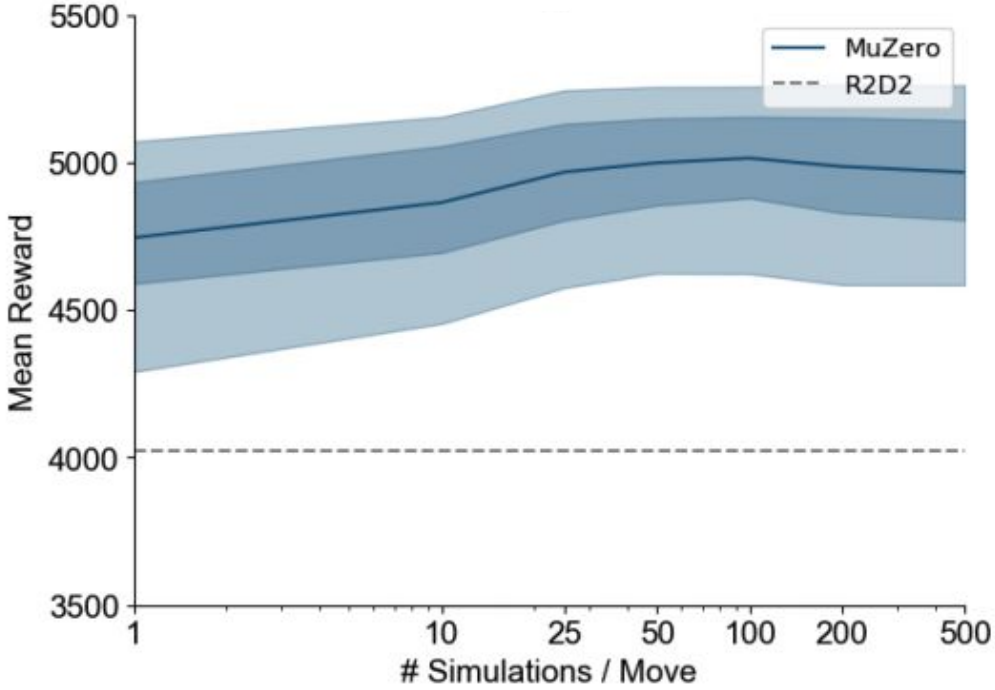
Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# Comparison of Real versus Learned Simulator




Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# Test-Time Sims Increase Atari Performance

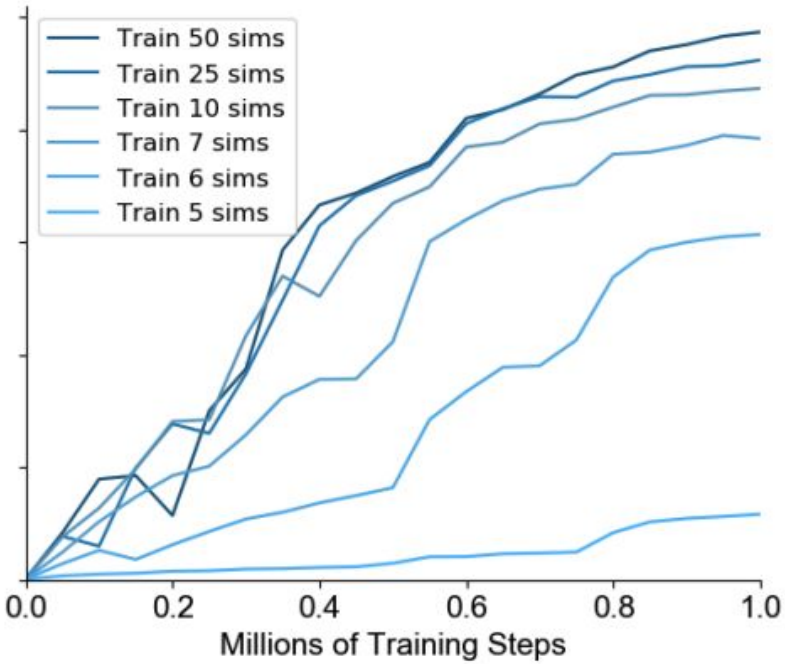



Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# Comparison with Model-free at Scale


Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# Effect of Simulations on Training Performance




Schrittwieser, Antonoglou, Hubert, et al. *arXiv*, 2020

# Conclusions

- Model-based algorithms hold the promise of addressing many of the limitations in current deep reinforcement learning approaches.

- We have working algorithms that allow *planning in unknown environments*.

- Much work remains to be done to understand these kinds of algorithms and they're relationship with model-free approaches --- especially off-policy approaches.

- The question of how to *discover* useful reward functions looms large.

# Questions?

Work with:

David Silver, Thomas Hubert, Julian Schrittwieser,
Ioannis Antonoglou, Karen Simonyan
& the AlphaZero Group

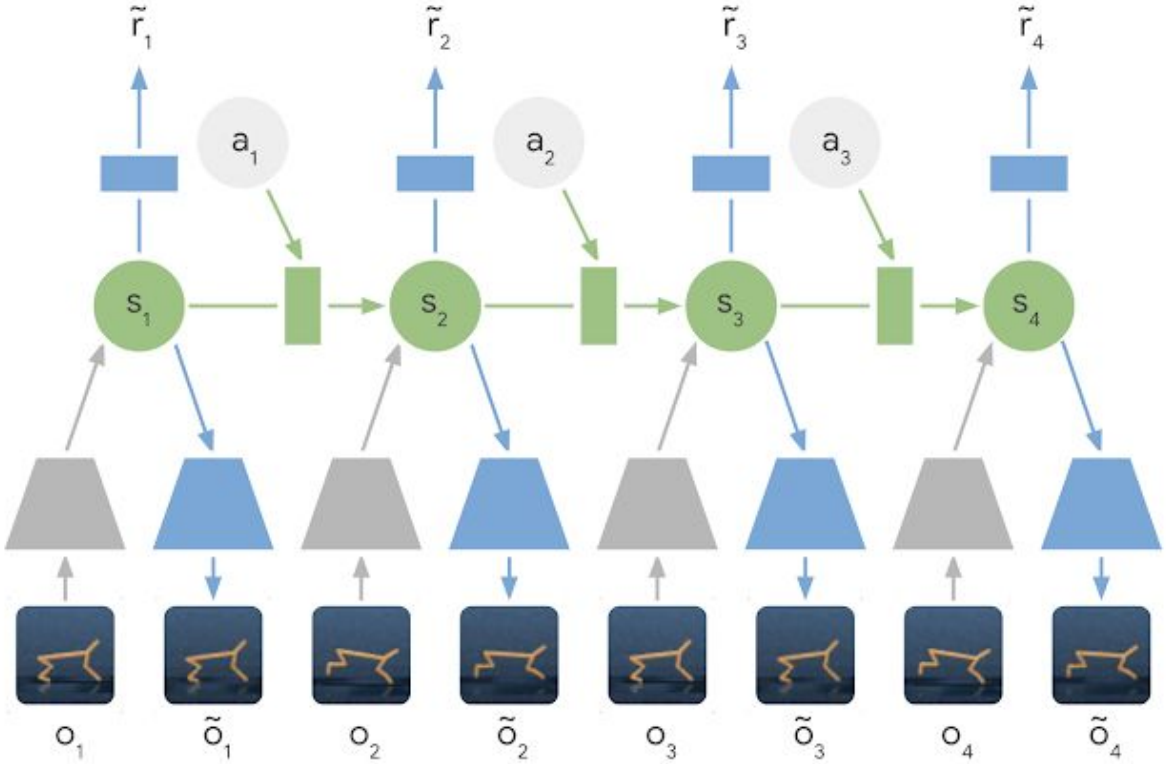
Danijar Hafner, Ian Fischer, Ruben Villegas,
David Ha, Honglak Lee, James Davidson

Matt Hoffman, Gabriel Barth-Maron, Yuval Tassa

&

Many others at DeepMind

# Planning with Learned Models of the World?




Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Planning with Learned Models has Been Tricky

Why is it dangerous to plan with a learned model?:

Planning is inherently a maximization operation that looks for a sequence of actions that make the expected return better.

Planning will search for and exploit imperfections in a learned model!

# Planning with Learned Models in State Space

**Algorithm 1** Our model-based MPC algorithm '*PETS*':

Initialize data $\mathbb{D}$ with a random controller for one trial.
**for** Trial $k = 1$ to $K$ **do**
  Train a *PE* dynamics model $\widetilde{f}$ given $\mathbb{D}$.
  **for** Time $t = 0$ to TaskHorizon **do**
    **for** Actions sampled $\boldsymbol{a}_{t:t+T} \sim \text{CEM}(\cdot)$, 1 to NSamples **do**
      Propagate state particles $\boldsymbol{s}_\tau^p$ using *TS* and $\widetilde{f}|\{\mathbb{D}, \boldsymbol{a}_{t:t+T}\}$.
      Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(\boldsymbol{s}_\tau^p, \boldsymbol{a}_\tau)$
      Update CEM$(\cdot)$ distribution.
    Execute first action $\boldsymbol{a}_t^*$ (only) from optimal actions $\boldsymbol{a}_{t:t+T}^*$.
    Record outcome: $\mathbb{D} \leftarrow \mathbb{D} \cup \{\boldsymbol{s}_t, \boldsymbol{a}_t^*, \boldsymbol{s}_{t+1}\}$.

Probabilistic Ensembles with Trajectory Sampling

Chua, Calandra, McAllister, Levine, *arXiv*, 2018

# Planning with Learned Models in State Space

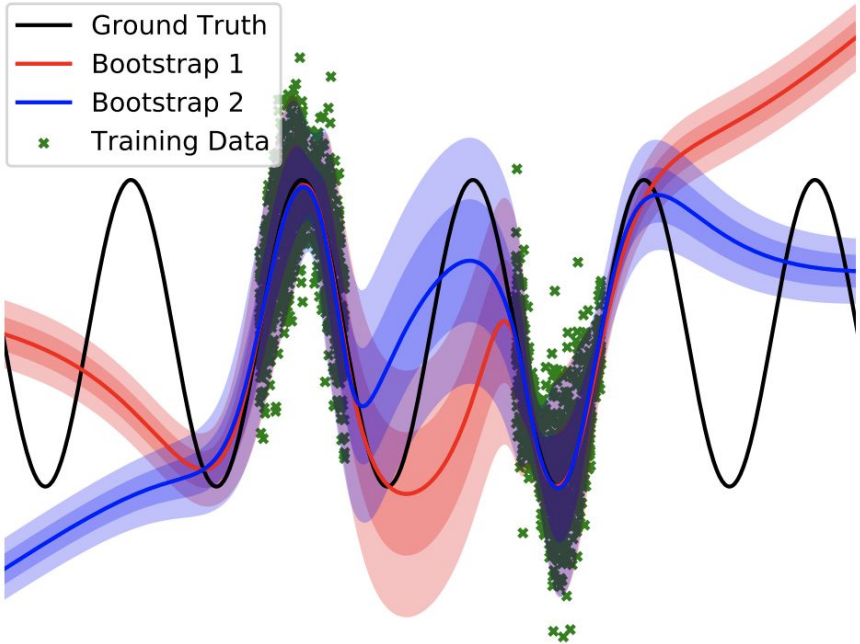
**Algorithm 1** Our model-based MPC algorithm '*PETS*':

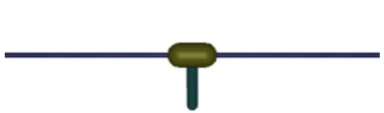
1: Initialize data $\mathbb{D}$ with a random controller for one trial.
2: **for** Trial $k = 1$ to $K$ **do**
3: Train a *PE* dynamics model $\tilde{f}$ given $\mathbb{D}$. Train Model
4: **for** Time $t = 0$ to TaskHorizon **do**
5: **for** Actions sampled $\boldsymbol{a}_{t:t+T} \sim \text{CEM}(\cdot)$, 1 to NSamples **do**
6: Propagate state particles $\boldsymbol{s}_\tau^p$ using $TS$ and $\tilde{f}|\{\mathbb{D}, \boldsymbol{a}_{t:t+T}\}$.
7: Evaluate actions as $\sum_{\tau=t}^{t+T} \frac{1}{P} \sum_{p=1}^{P} r(\boldsymbol{s}_\tau^p, \boldsymbol{a}_\tau)$ Plan
8: Update CEM$(\cdot)$ distribution.
9: Execute first action $\boldsymbol{a}_t^*$ (only) from optimal actions $\boldsymbol{a}_{t:t+T}^*$.
10: Record outcome: $\mathbb{D} \leftarrow \mathbb{D} \cup \{\boldsymbol{s}_t, \boldsymbol{a}_t^*, \boldsymbol{s}_{t+1}\}$. Record Data

Probabilistic Ensembles with Trajectory Sampling

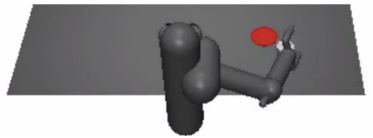
Chua, Calandra, McAllister, Levine, *arXiv*, 2018

# Planning with Learned Models in State Space

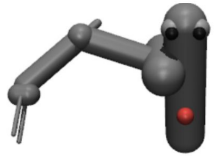


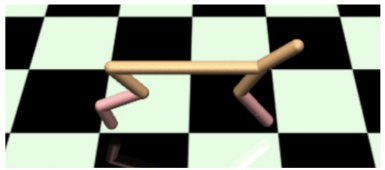
(a) Cartpole



(b) 7-dof Pusher



(c) 7-dof Reacher

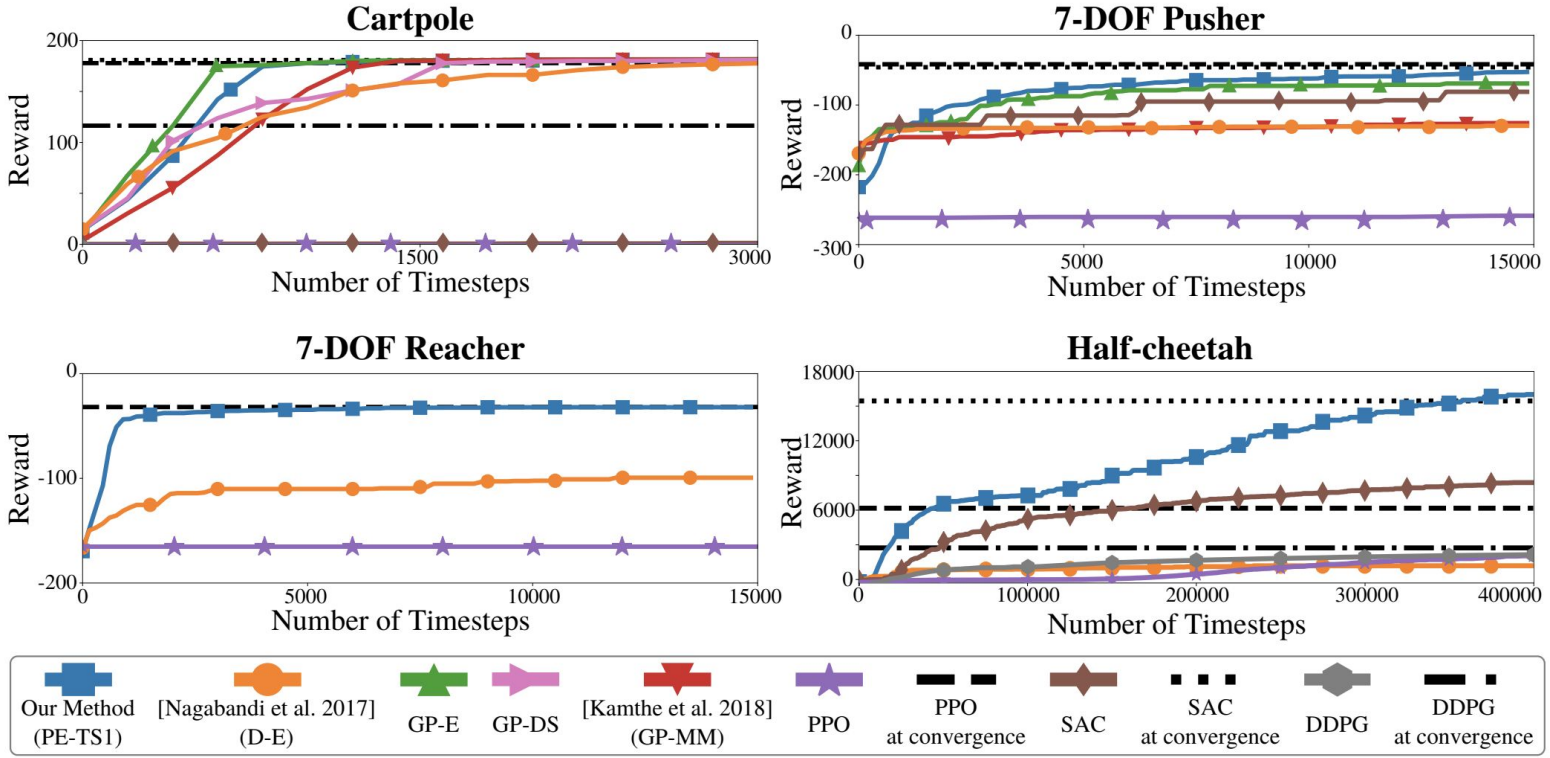


(d) Half-cheetah

Probabilistic Ensembles with Trajectory Sampling

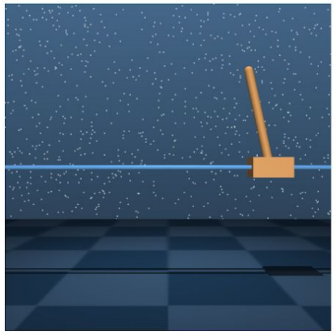
Chua, Calandra, McAllister, Levine, *arXiv*, 2018

# Planning with Learned Models in State Space

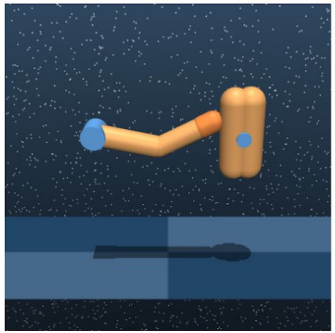



Chua, Calandra, McAllister, Levine, *arXiv*, 2018

# Learning Latent Dynamics for Planning from Pixels?

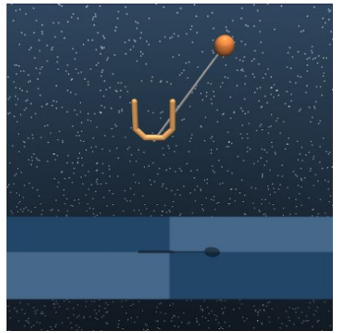


(a) Cartpole

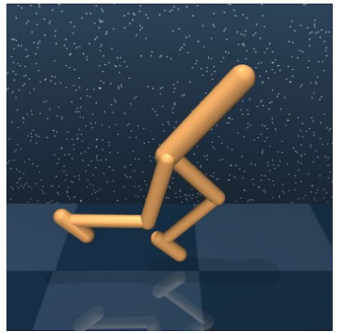


(b) Finger

(c) Cheetah



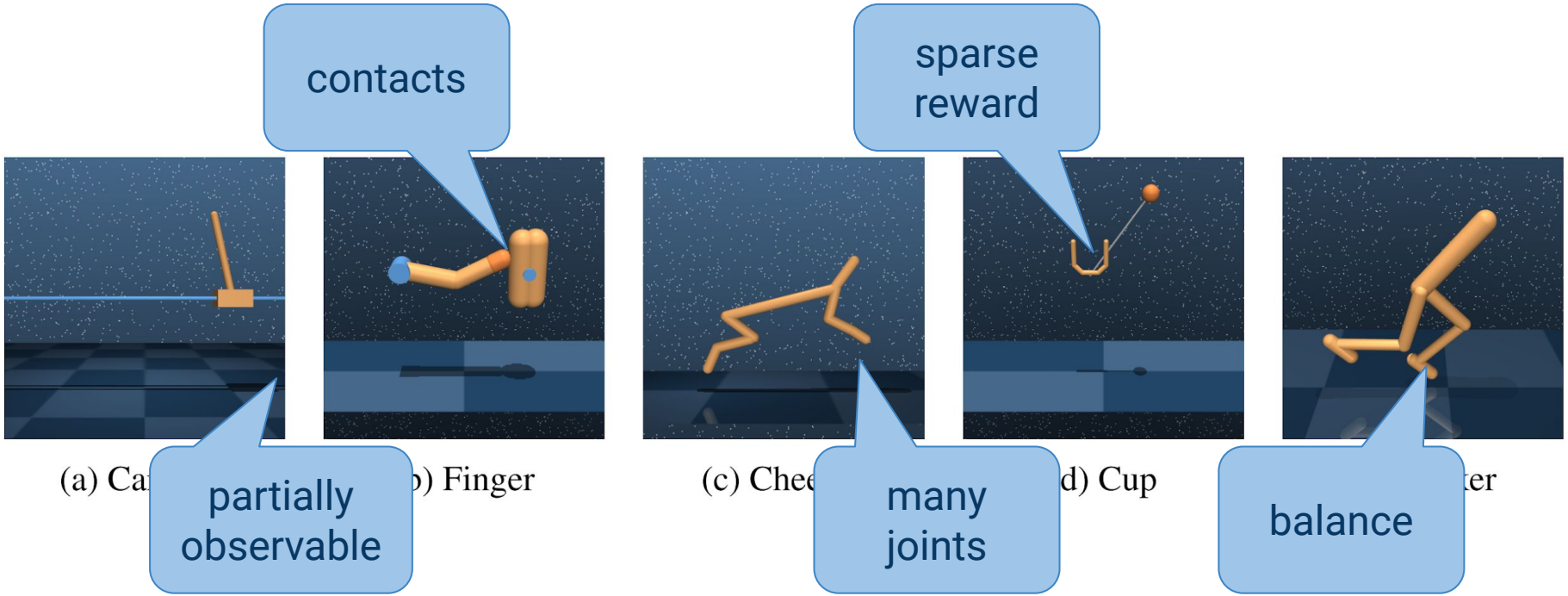
(d) Cup



(e) Walker

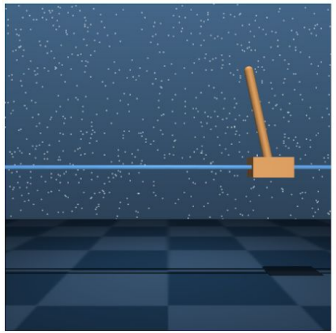
Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Continuous Control from Image Observations

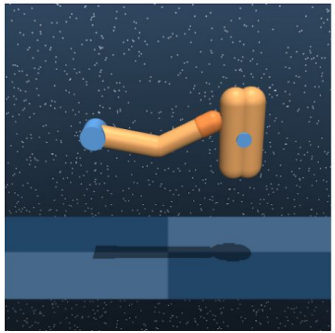



Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Continuous Control from Image Observations

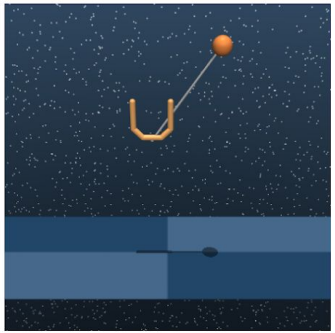


(a) Cartpole

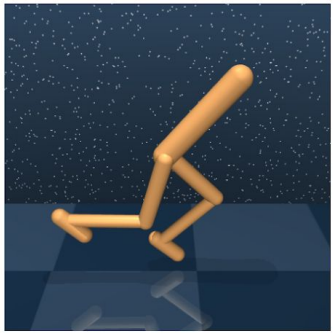


(b) Finger

(c) Cheetah



(d) Cup

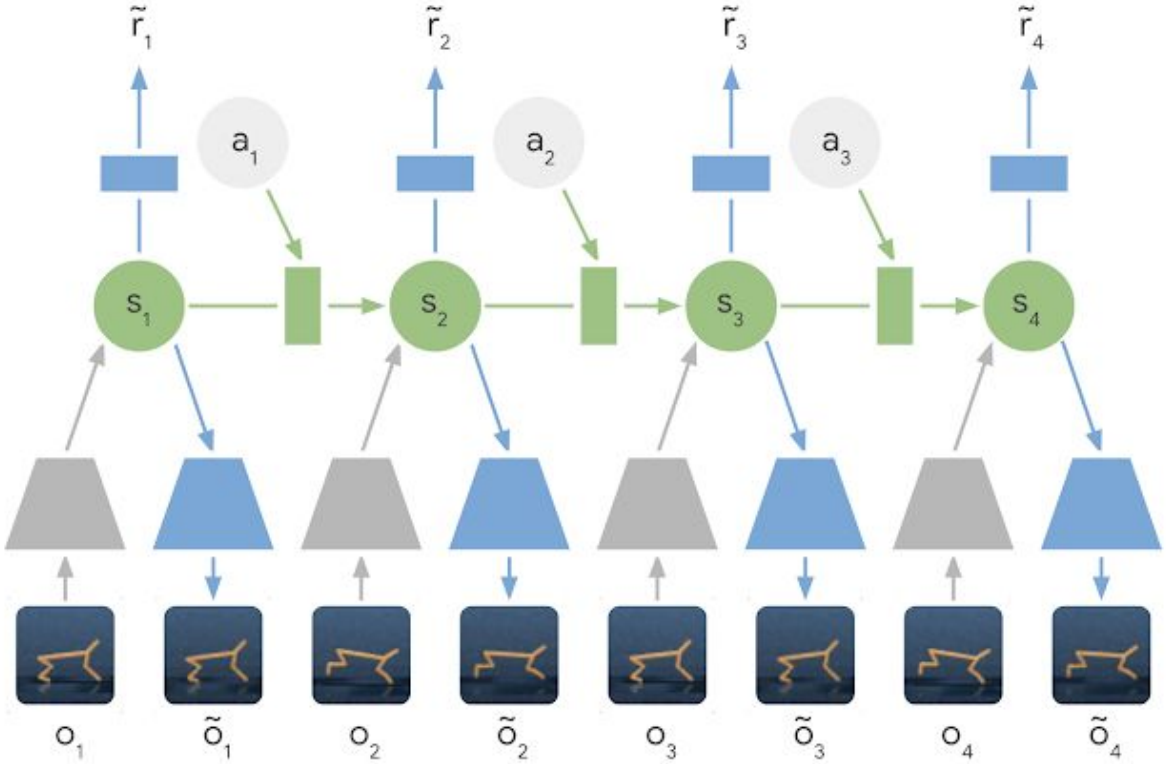


(e) Walker

Model-free algorithms can solve
these tasks but need 1,000s of episodes

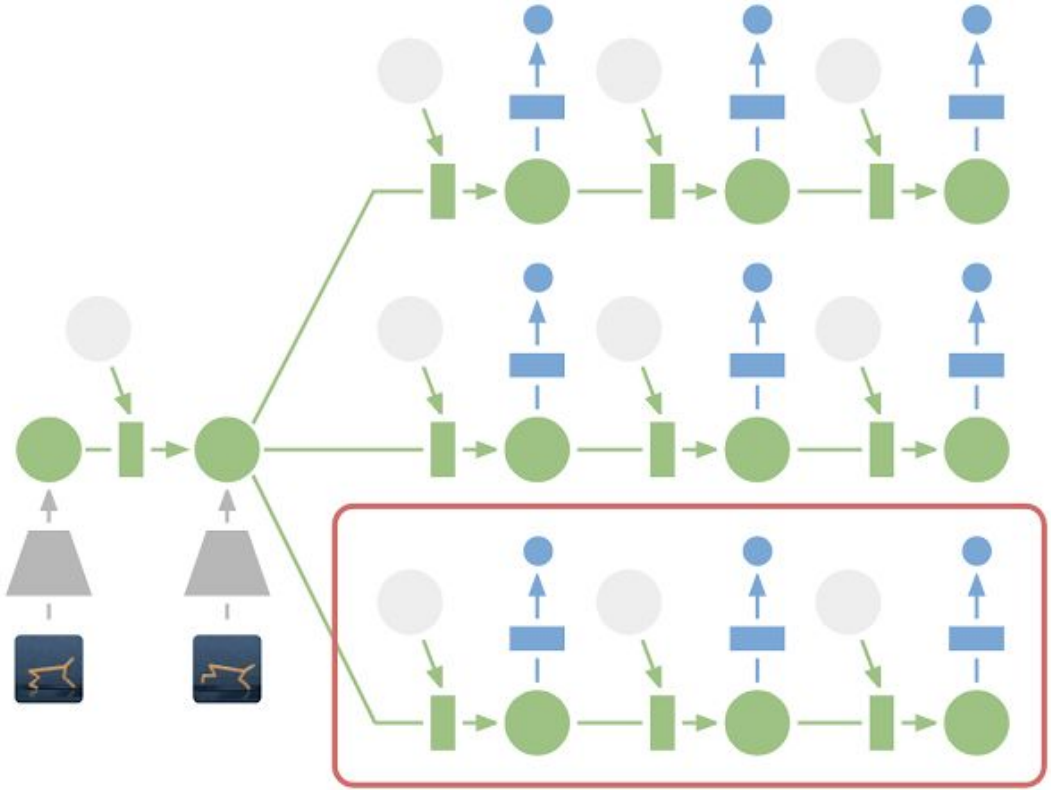
Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Learning Latent Dynamics for Planning from Pixels



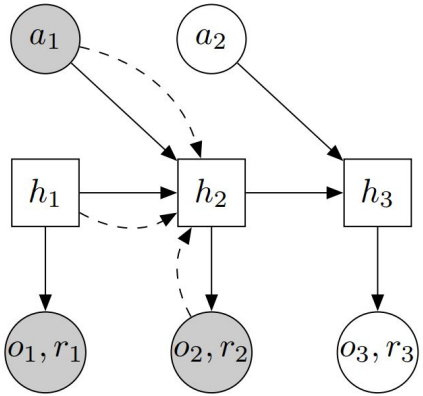

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Learning Latent Dynamics for Planning from Pixels

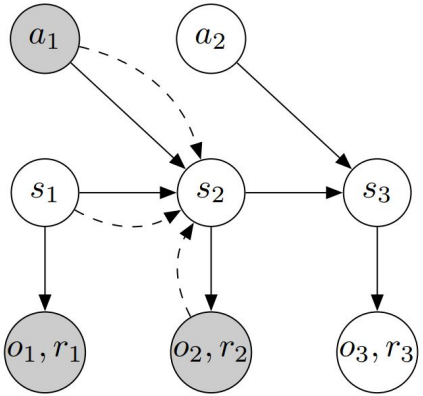


Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

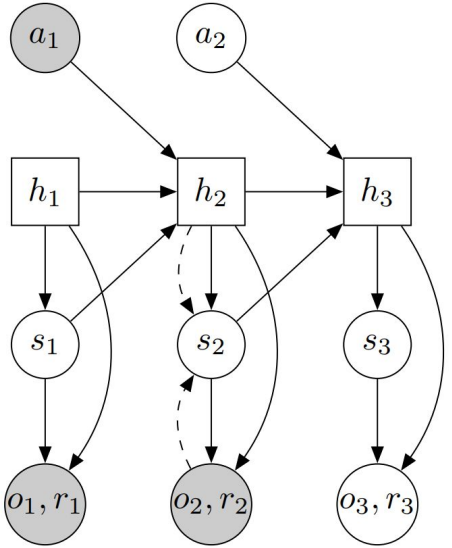
# Recurrent State-Space Model




(a) Deterministic model (RNN)



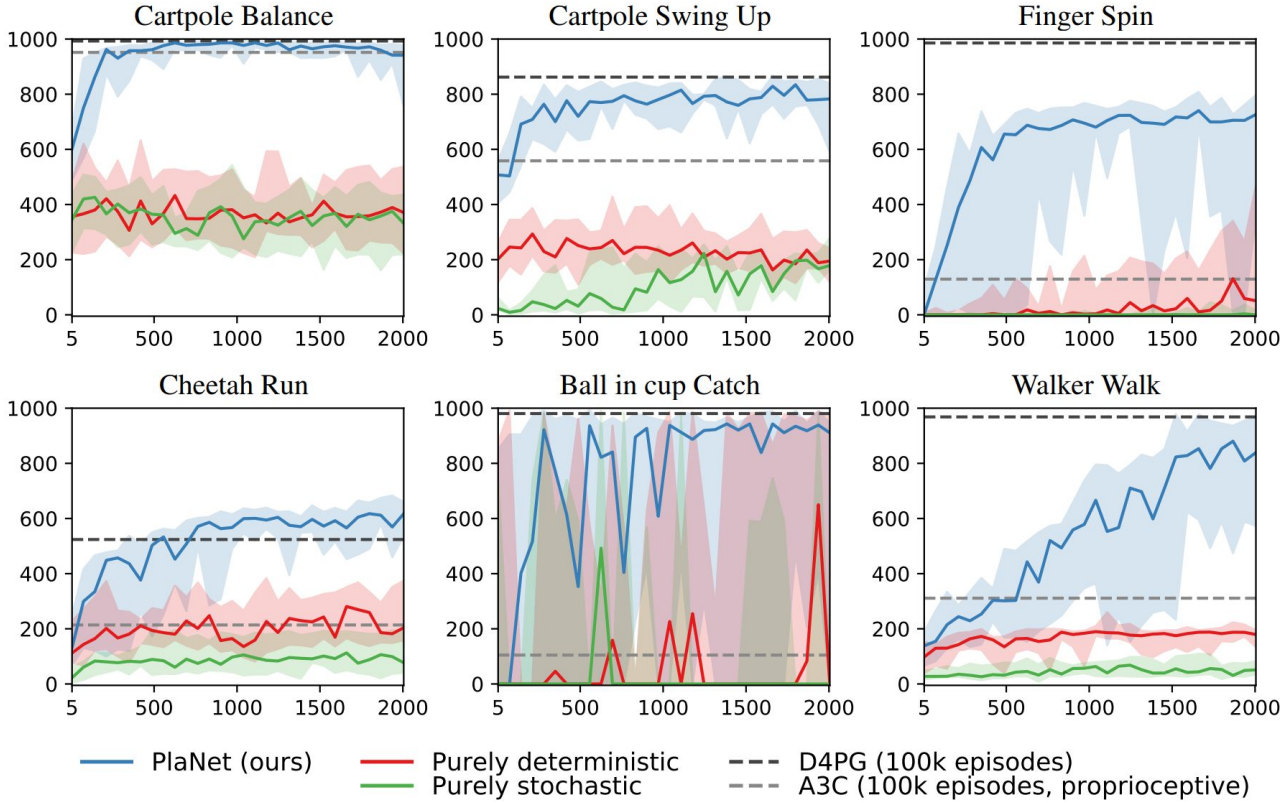

(b) Stochastic model (SSM)




(c) Recurrent state-space model (RSSM)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Comparison of Model Designs




Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Comparison to model-free agents (A3C & D4PG)

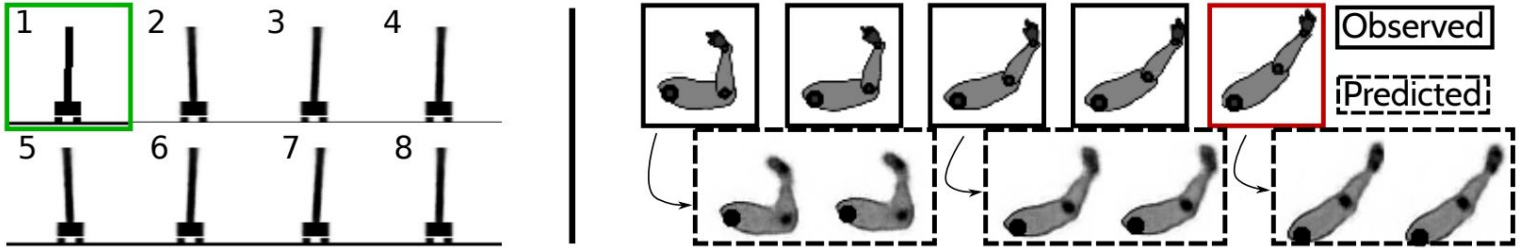
| Method | Modality | Episodes | Cartpole Balance | Cartpole Swingup | Finger Spin | Cheetah Run | Ball in cup Catch | Walker Walk |
|---|---|---|---|---|---|---|---|---|
| A3C | proprioceptive | 100,000 | 952 | 558 | 129 | 214 | 105 | 311 |
| D4PG | pixels | 100,000 | 993 | 862 | 985 | 524 | 980 | 968 |
| PlaNet (ours) | pixels | 2,000 | 986 | 831 | 744 | 650 | 914 | 890 |
| CEM + true simulator | simulator state | 0 | 998 | 850 | 825 | 656 | 993 | 994 |
| Data efficiency gain PlaNet over D4PG (factor) | | | 100 | 180 | 16 | 50+ | 20 | 11 |

(Training time 1 day for PlaNet and 1 day for model-free on a single GPU. We did not parallelize data collection yet.)

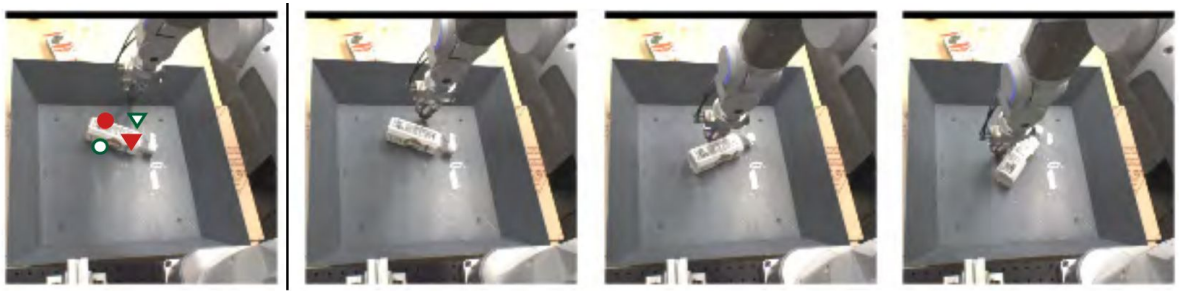
Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018


505 Episodes

# Previous Model-Based Approaches



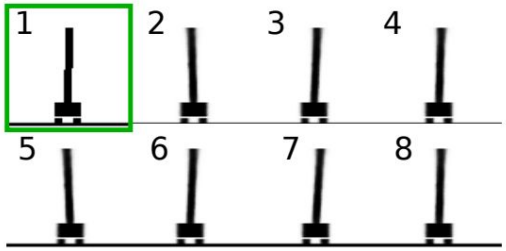

Watter et al., 2015, Banijamali et al. 2017, Zhang et al. 2017

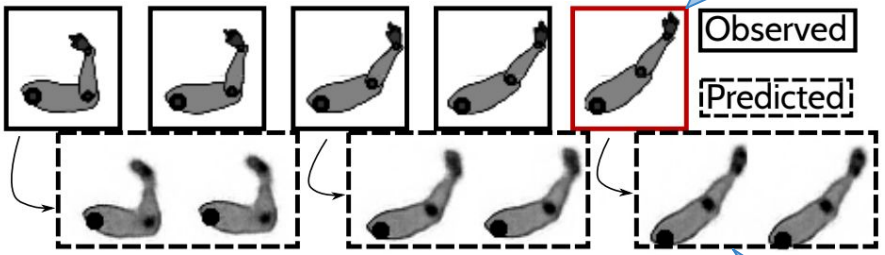


Agrawal et al., 2016; Finn & Levine, 2016; Ebert et al., 2018

# Previous Model-Based Approaches

latent space but Markov images

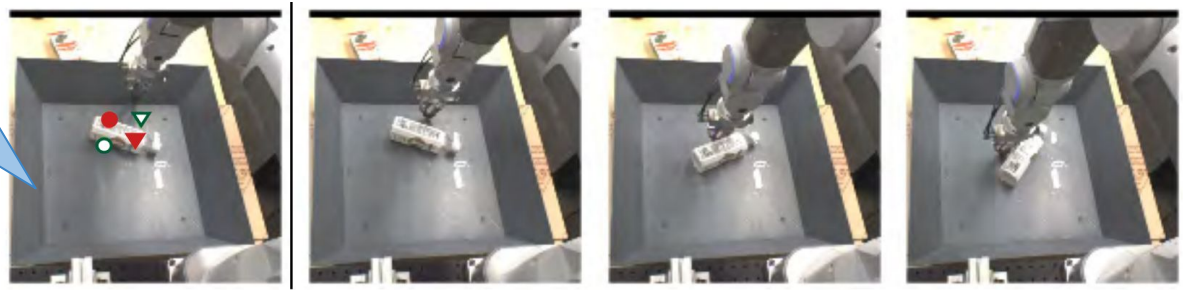



Watter et al., 2015, Banijamali et al. 2017, Zhang et al. 2017

simple behavior and visuals

visually complex but simple controls

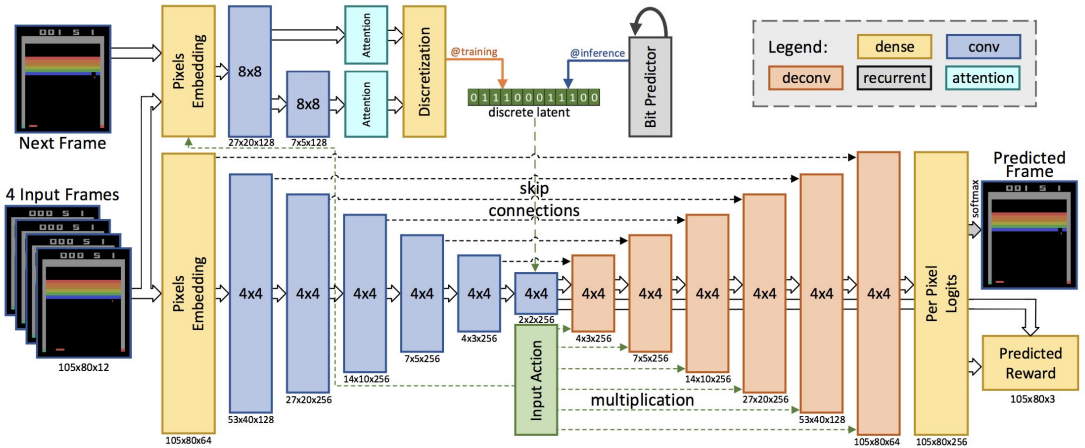


Agrawal et al., 2016; Finn & Levine, 2016; Ebert et al., 2018

# Previous Model-Based Approaches

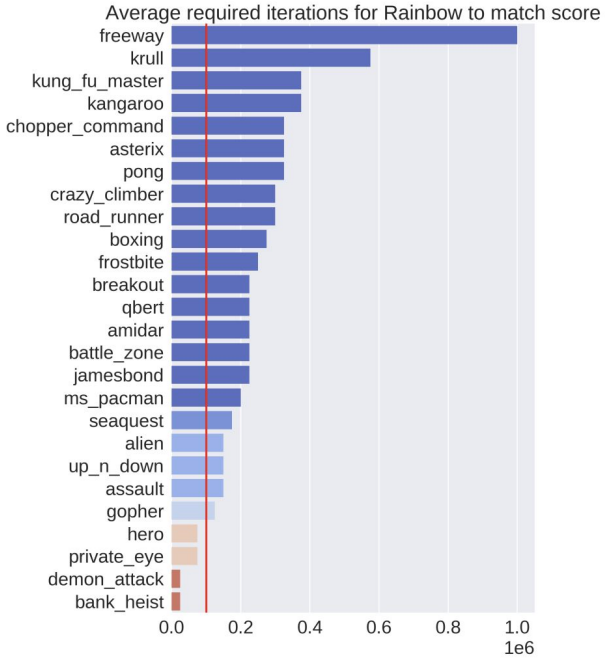
**Model Based Reinforcement Learning for Atari**

Łukasz Kaiser [*1] Mohammad Babaeizadeh [*23] Piotr Miłoś [*45] Błażej Osiński [*453] Roy H Campbell [2]
Konrad Czechowski [4] Dumitru Erhan [1] Chelsea Finn [1] Piotr Kozakowski [4] Sergey Levine [1] Ryan Sepassi [1]
George Tucker [1] Henryk Michalewski [45]




Kaiser et al., *arXiv*, 2019

# Previous Model-Based Approaches




**Algorithm 1:** Pseudocode for SimPLe

Initialize policy $\pi$
Initialize model parameters $env'$
Initialize empty set $\mathbf{D}$
**while** not done **do**
  ▷ collect observations from real env.
  **while** not enough observations **do**
    $a \leftarrow \pi(s)$
    $(s', r) \leftarrow env(a)$
    $\mathbf{D} \leftarrow \mathbf{D} \cup (s, a, r, s')$
    $s \leftarrow s'$
  **end while**
  ▷ update model using collected data.
  $\theta \leftarrow$ TRAIN_SUPERVISED$(env', \mathbf{D})$
  ▷ update policy using world model.
  $\pi \leftarrow$ TRAIN_RL$(\pi, \theta)$
**end while**

[2]Specifically, for the final evaluation we selected games which achieved non-random results using our method or the Rainbow algorithm using 100K interactions.

Kaiser et al., *arXiv*, 2019

**Algorithm 1:** Deep Planning Network (PlaNet)

**Input :**

| | | | |
|---|---|---|---|
| $R$ | Action repeat | $p(s_t \mid s_{t-1}, a_{t-1})$ | Transition model |
| $S$ | Seed episodes | $p(o_t \mid s_t)$ | Observation model |
| $C$ | Collect interval | $p(r_t \mid s_t)$ | Reward model |
| $B$ | Batch size | $q(s_t \mid o_{\leq t}, a_{<t})$ | Encoder |
| $L$ | Chunk length | $p(\epsilon)$ | Exploration noise |
| $\alpha$ | Learning rate | | |

Initialize dataset $\mathcal{D}$ with $S$ random seed episodes.
Initialize model parameters $\theta$ randomly.
**while** *not converged* **do**
  `// Model fitting`
  **for** *update step* $s = 1..C$ **do**
    Draw sequence chunks $\{(o_t, a_t, r_t)_{t=k}^{L+k}\}_{i=1}^{B} \sim \mathcal{D}$ uniformly at random from the dataset.
    Compute loss $\mathcal{L}(\theta)$ from Equation 7.
    Update model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$.
  `// Data collection`
  $o_1 \leftarrow$ `env.reset()`
  **for** *time step* $t = 1..\lceil \frac{T}{R} \rceil$ **do**
    Infer belief over current state $q(s_t \mid o_{\leq t}, a_{<t})$ from the history.
    $a_t \leftarrow$ `planner`$(q(s_t \mid o_{\leq t}, a_{<t}), p)$, see Algorithm 2 in the appendix for details.
    Add exploration noise $\epsilon \sim p(\epsilon)$ to the action.
    **for** *action repeat* $k = 1..R$ **do**
      $r_t^k, o_{t+1}^k \leftarrow$ `env.step`$(a_t)$
    $r_t, o_{t+1} \leftarrow \sum_{k=1}^{R} r_t^k, o_{t+1}^R$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$

**Algorithm 2:** Latent planning with CEM

**Input :**

| | | | |
|---|---|---|---|
| $H$ | Planning horizon distance | $q(s_t \mid o_{\leq t}, a_{<t})$ | Current state belief |
| $I$ | Optimization iterations | $p(s_t \mid s_{t-1}, a_{t-1})$ | Transition model |
| $J$ | Candidates per iteration | $p(r_t \mid s_t)$ | Reward model |
| $K$ | Number of top candidates to fit | | |

Initialize factorized belief over action sequences $q(a_{t:t+H}) \leftarrow \text{Normal}(0, \mathbb{I})$.
**for** *optimization iteration* $i = 1..I$ **do**
  `// Evaluate` $J$ `action sequences from the current belief.`
  **for** *candidate action sequence* $j = 1..J$ **do**
    $a^{(j)}_{t:t+H} \sim q(a_{t:t+H})$
    $s^{(j)}_{t:t+H+1} \sim q(s_t \mid o_{1:t}, a_{1:t-1}) \prod_{\tau=t+1}^{t+H+1} p(s_\tau \mid s_{\tau-1}, a^{(j)}_{\tau-1})$
    $R^{(j)} = \sum_{\tau=t+1}^{t+H+1} \mathrm{E}[p(r_\tau \mid s^{(j)}_\tau)]$
  `// Re-fit belief to the` $K$ `best action sequences.`
  $\mathcal{K} \leftarrow \text{argsort}(\{R^{(j)}\}_{j=1}^{J})_{1:K}$
  $\mu_{t:t+H} = \frac{1}{K} \sum_{k \in \mathcal{K}} a^{(k)}_{t:t+H}, \quad \sigma_{t:t+H} = \frac{1}{K-1} \sum_{k \in \mathcal{K}} |a^{(k)}_{t:t+H} - \mu_{t:t+H}|.$
  $q(a_{t:t+H}) \leftarrow \text{Normal}(\mu_{t:t+H}, \sigma^2_{t:t+H}\, \mathbb{I})$
**return** *first action mean* $\mu_t$.

# Our Contributions

## Deep Latent Planning (PlaNet)

Purely model-based agent that solves several control tasks from images

Outperforms A3C and similar final performance to D4PG with 50x less data

## Recurrent State-Space Model (RSSM)

Combine deterministic and stochastic transition elements

Consistent rollouts, handles partial observability, harder for planner to exploit

## Latent Overshooting

We need accurate multi-step reward predictions for planning

Generalize ELBO to train latent sequence models on multi-step predictions

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Deep Planning Network Overview

**Problem:**

- To plan in unknown envs the agent must learn the dynamics from experience
- Must iteratively collect more data as the agent gets better to visit more states
- Individual images don't convey full state of the environment

**Solution:**

- Start from a few seed episodes collected using random actions
- Train latent dynamics model (think sequential VAE for images and rewards)
- Collect one episode using planning in latent space every few model updates
- For planning, infer belief over state from past observation sequence

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Recurrent State-Space Model

Deterministic models (RNN, LSTM, GRU, etc)

- ✓ Can remember information over many time steps
- ✗ Cannot predict multiple futures as needed for partial observability
- ✗ Easy for planner to exploit inaccuracies

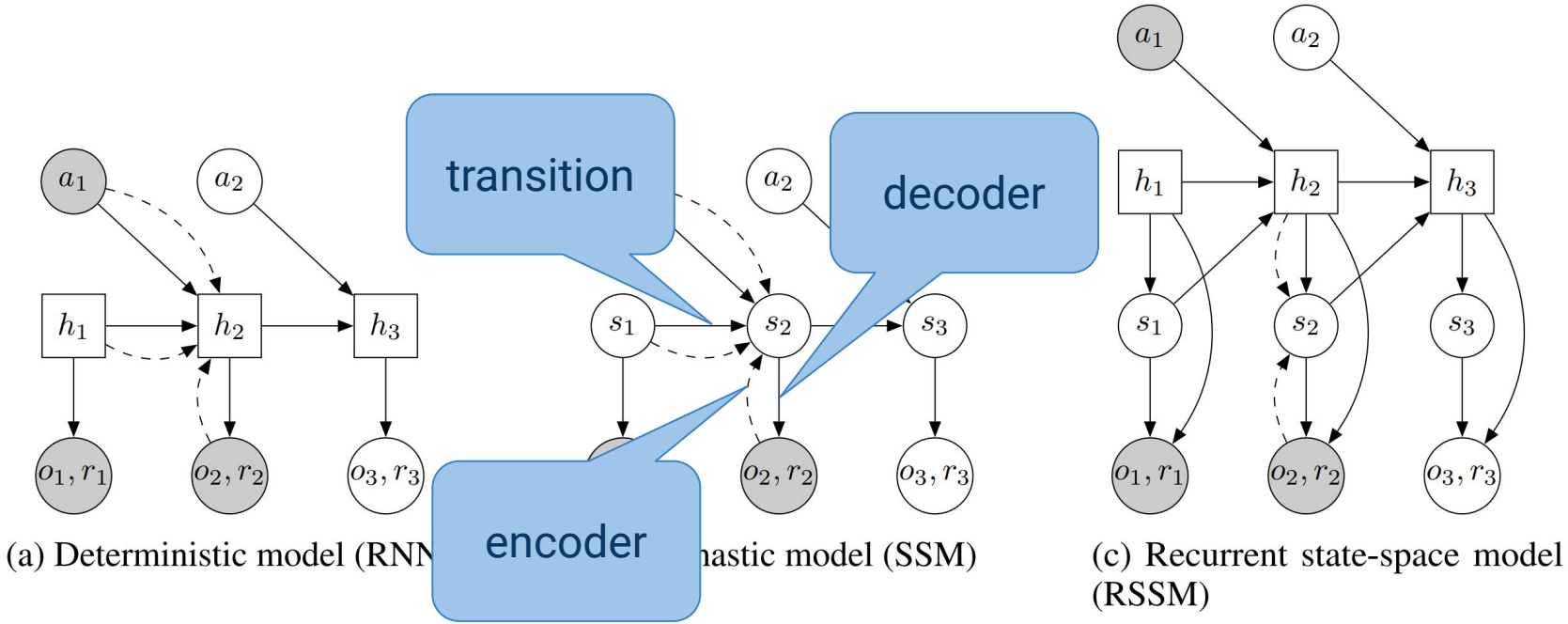
Stochastic models (Gaussian SSM, MoG SSM, etc)

- ✓ Can prediction multiple futures
- ✓ Creates safety margin around the planning objective
- ✗ Difficult to optimize: often too random, does not remember over many steps

Solution: Both deterministic and stochastic transition elements

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Recurrent State-Space Model

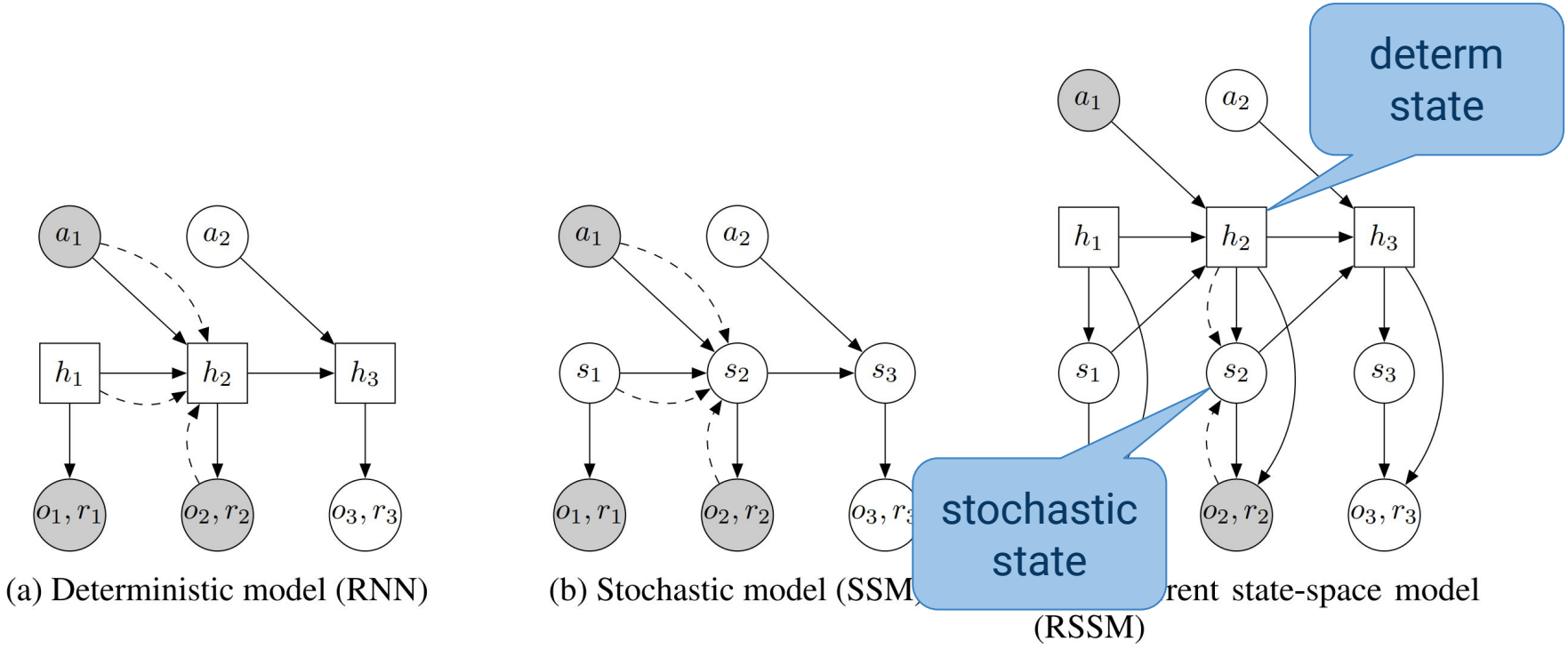



(a) Deterministic model (RNN ... hastic model (SSM)

(c) Recurrent state-space model (RSSM)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Recurrent State-Space Model




(a) Deterministic model (RNN)

(b) Stochastic model (SSM) ... rent state-space model (RSSM)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Latent Overshooting

Problem

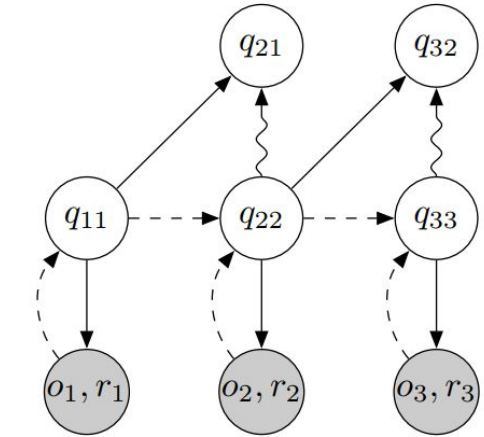
- Perfect one-step predictions would mean perfect multi-step predictions
- However, model has **limited capacity** and restricted posterior distribution
- In general, best one-step fit **does not coincide** with best multi-step fit
- Want to train on all multi-step predictions, but this is expensive for images

Solution

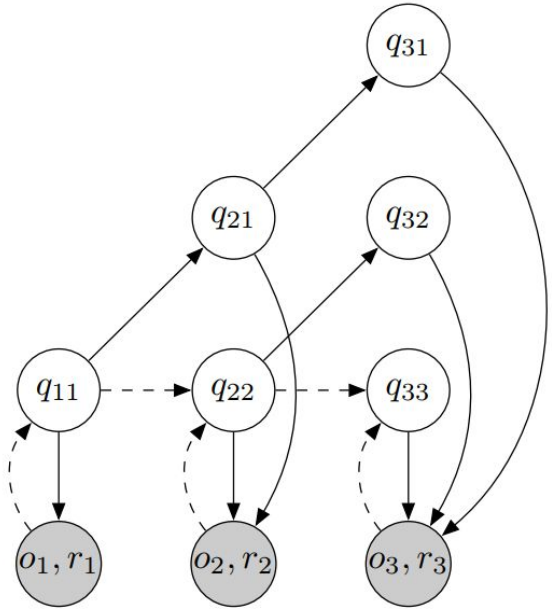
- Generalize ELBO objective for latent sequence models to multi-step
- Keep only KL-terms in latent space and thus fast to compute
- Yields fast and effective regularizer that encourages long-term predictions

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

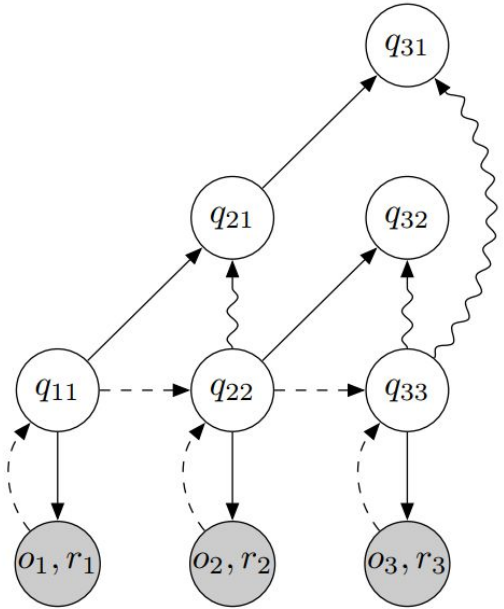
# Latent Overshooting




(a) Standard variational bound



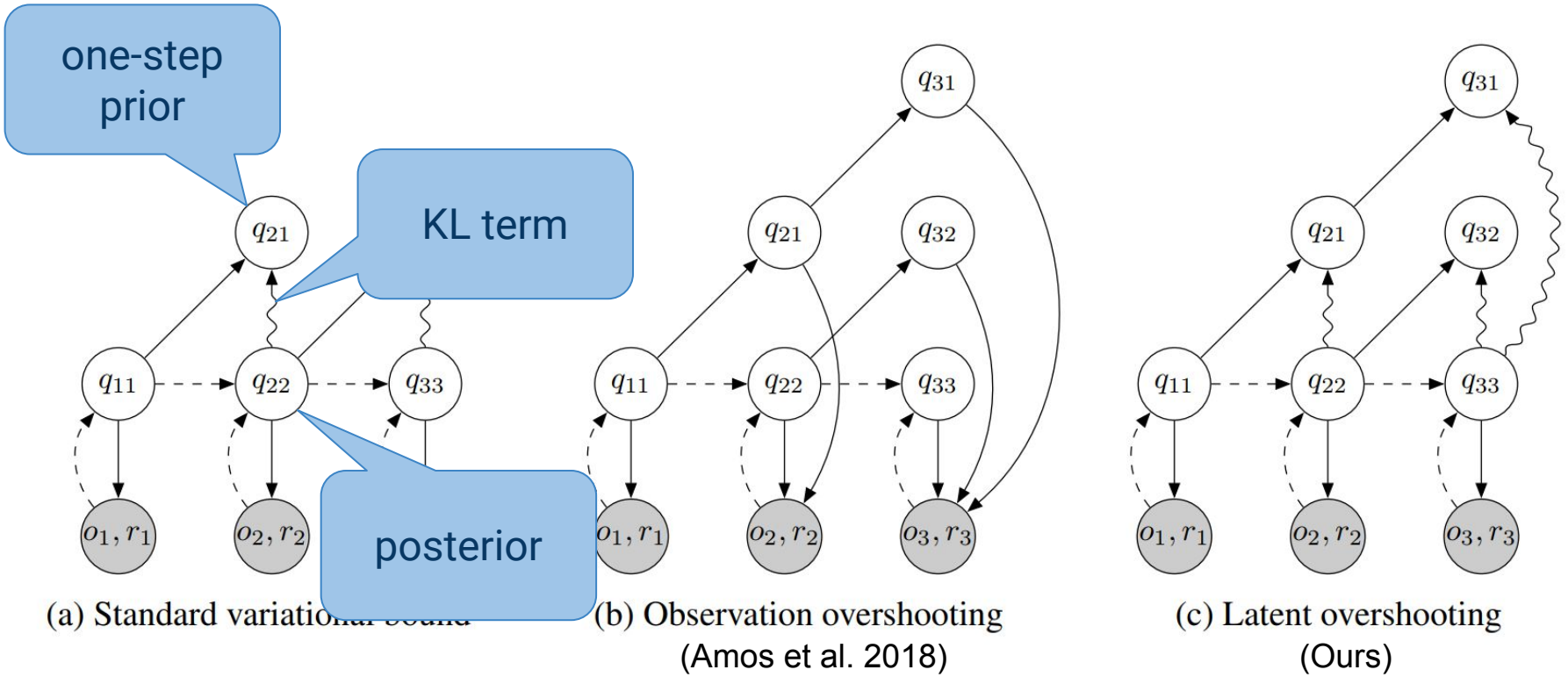

(b) Observation overshooting
(Amos et al. 2018)




(c) Latent overshooting
(Ours)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Latent Overshooting



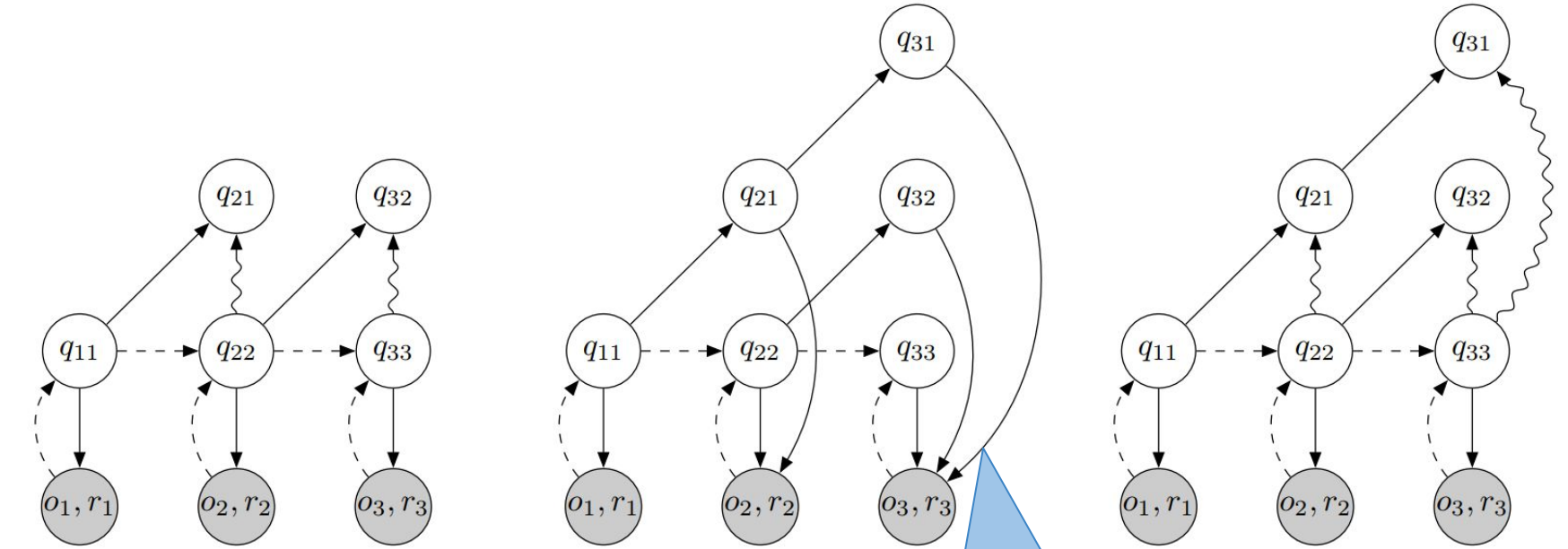

(a) Standard variational bound

(b) Observation overshooting
(Amos et al. 2018)

(c) Latent overshooting
(Ours)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Latent Overshooting



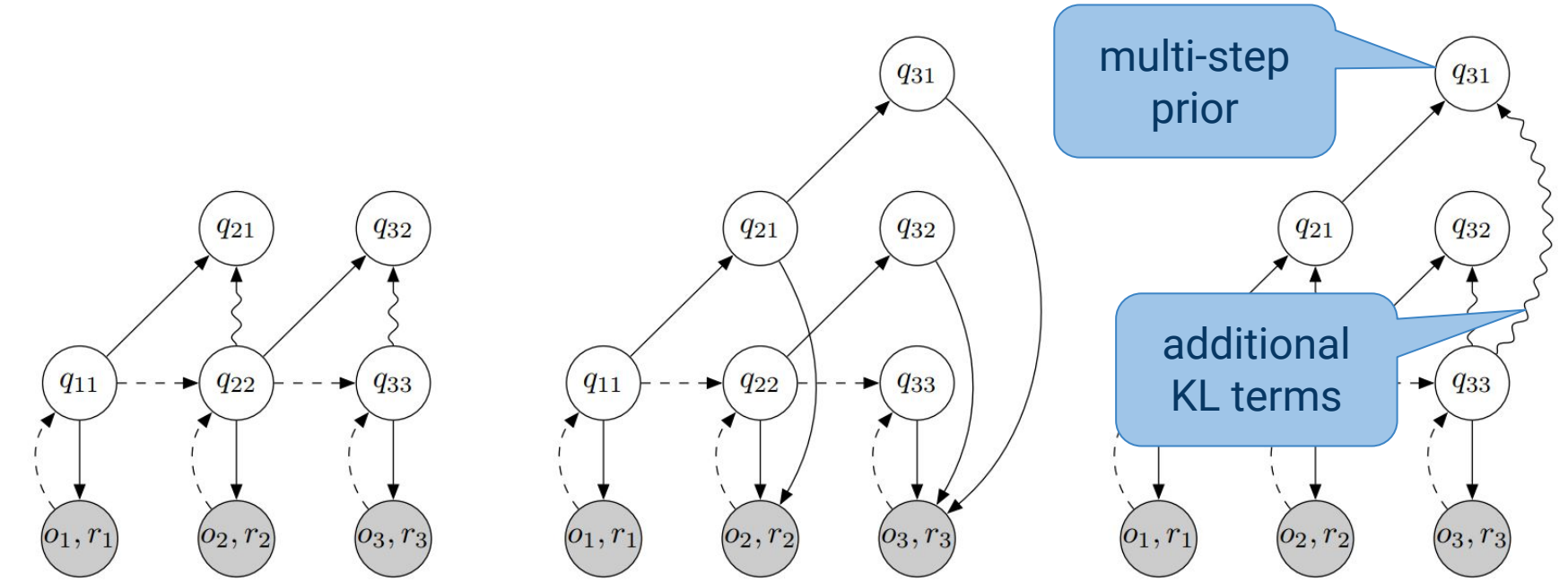

(a) Standard variational bound

(b) Observati
(Amos

(c) Latent overshooting
(Ours)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Latent Overshooting




(a) Standard variational bound

(b) Observation overshooting (Amos et al. 2018)

(c) Latent overshooting (Ours)

Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Latent Overshooting

$$\ln p(o_{1:T} \mid a_{1:T}) \geq$$

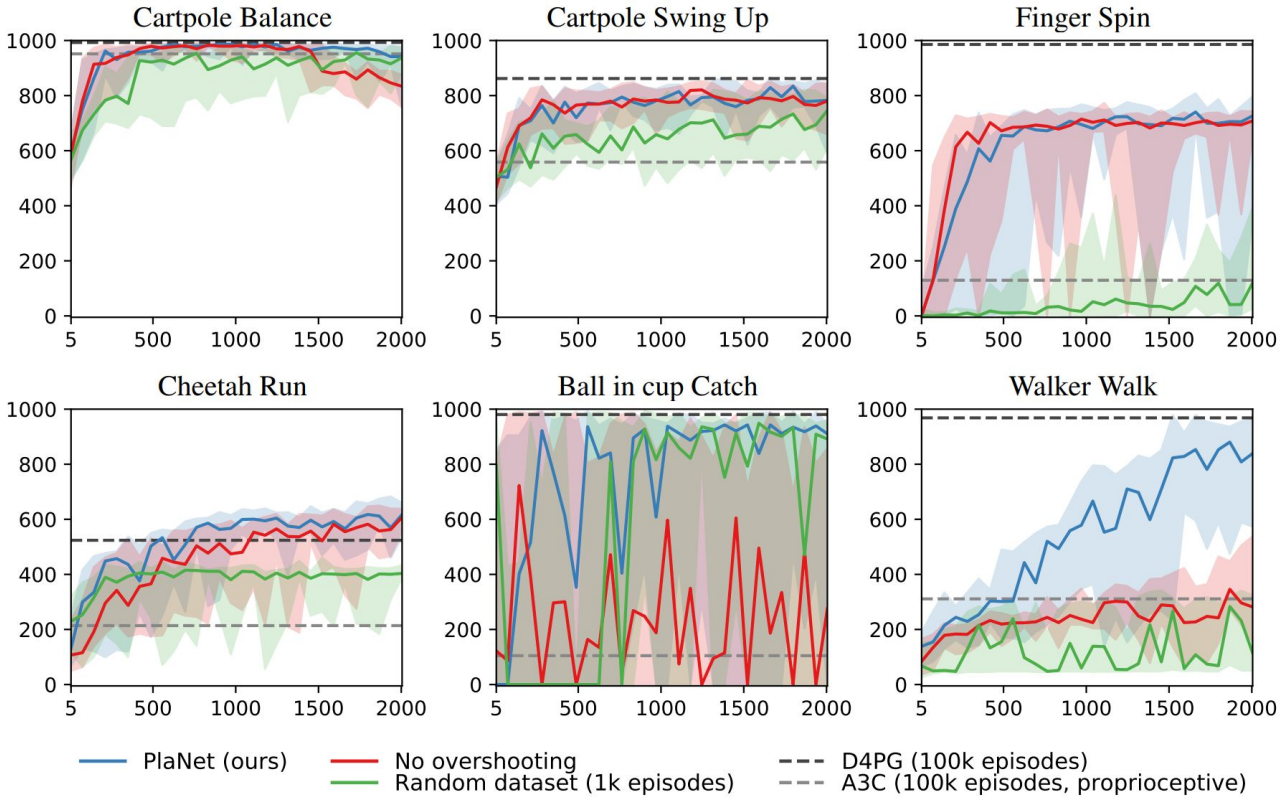
$$\sum_{t=1}^{T} \Bigg( \underbrace{\mathrm{E}_{q(s_t \mid o_{\leq t}, a_{<t})}\left[\ln p(o_t \mid s_t)\right]}_{\text{reconstruction}}$$

$$\underbrace{- \frac{1}{D} \sum_{d=1}^{D} \beta_d \, \mathrm{E}_{q(s_{t-1} \mid o_{\leq t-d}, a_{<t-1})} \left[ \mathrm{KL}\left[ q(s_t \mid o_{\leq t}, a_{<t}) \parallel p(s_t \mid s_{t-1}, a_{t-1}) \right] \right] \Bigg)}_{\text{latent overshooting}}$$

average of ELBOs with different priors

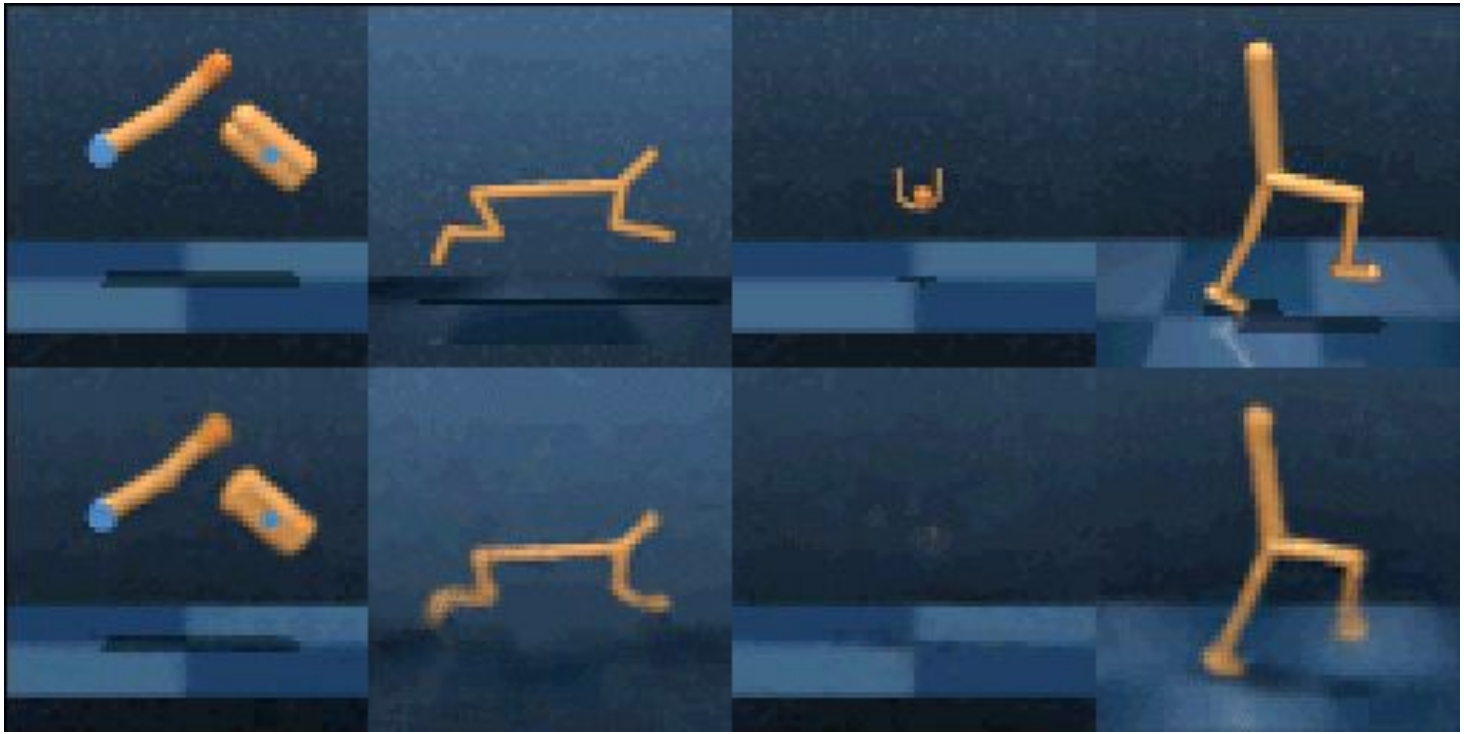
Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# Comparison of Agent Designs




Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018

# One Agent on All Tasks:



Hafner, Lillicrap, Fischer, Villegas, Ha, Lee, Davidson, *arXiv*, 2018