Model-Based RL

What is model (in model-based RL) ?

- Having a model = having an ability to predict future outcome.

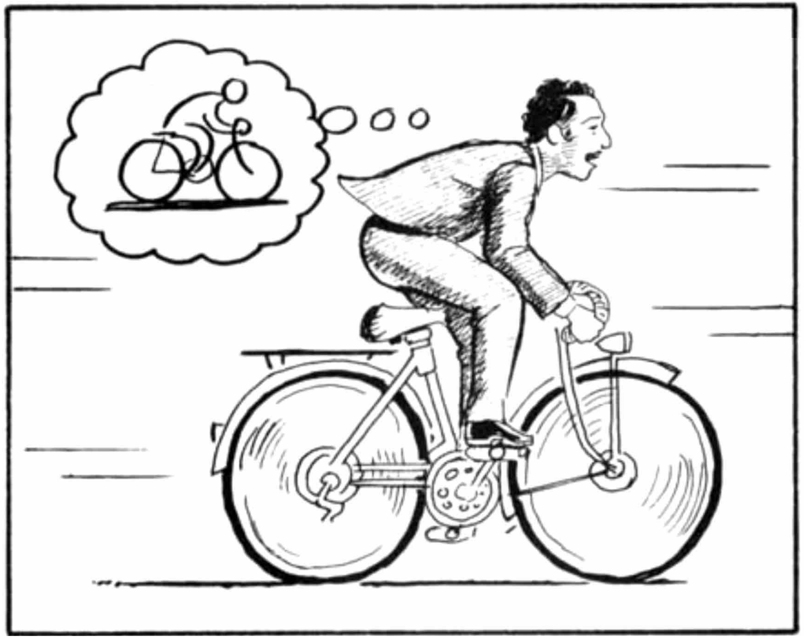


Illustration by
Ha and Schmidhuber, 2018

We use model to do planning…

- What is planning?
- Planning is imagining the future and choose the action accordingly.

Why use model-based RL ?

1. Sample Efficiency? (Debatable)
2. Transferability and Generalisability

Cite: Chelsea Fin's slide for DRL bootcamp

Data efficiency?

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, etc.)

10x

model-based deep RL
(e.g. guided policy search)

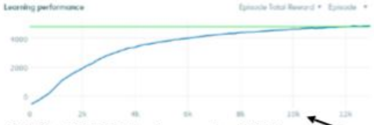
10x

model-based "shallow" RL
(e.g. PILCO)

Evolution Strategies as a Scalable Alternative to Reinforcement Learning

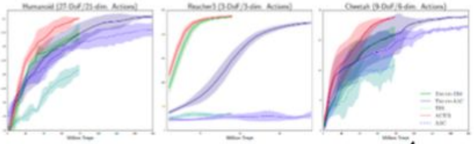
Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

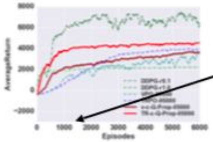
10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)



Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

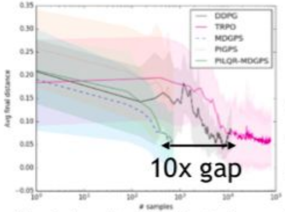
half-cheetah



Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~ 3 hours real time)

	cart-pole	cart-double-pole	unicycle
state space	$\mathbb{R}^4$	$\mathbb{R}^6$	$\mathbb{R}^{12}$
# trials	≤ 10	20–30	≈ 20
experience	≈ 20 s	≈ 60 s–90 s	≈ 20 s–30 s
parameter space	$\mathbb{R}^{305}$	$\mathbb{R}^{1816}$	$\mathbb{R}^{28}$



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

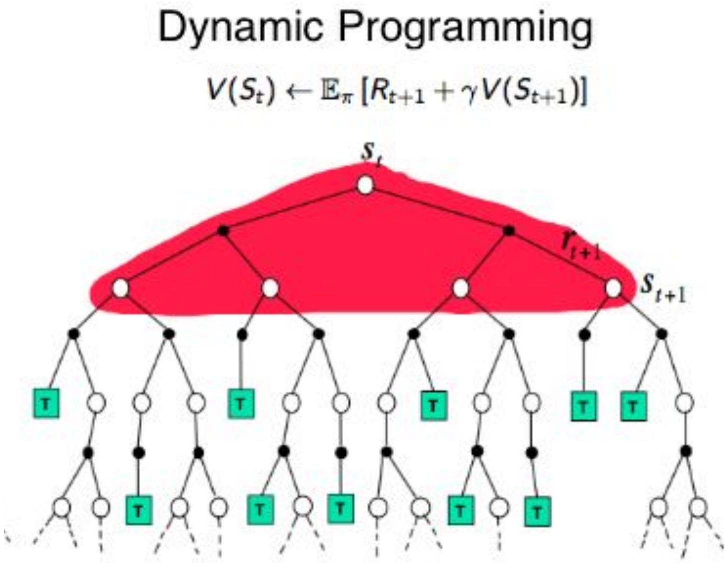
Slide from Sergey Levine

Transferability?

- Imagine same environment setting for several tasks
- Those MDPs have the same transition function.

Explicit vs Implicit Planning

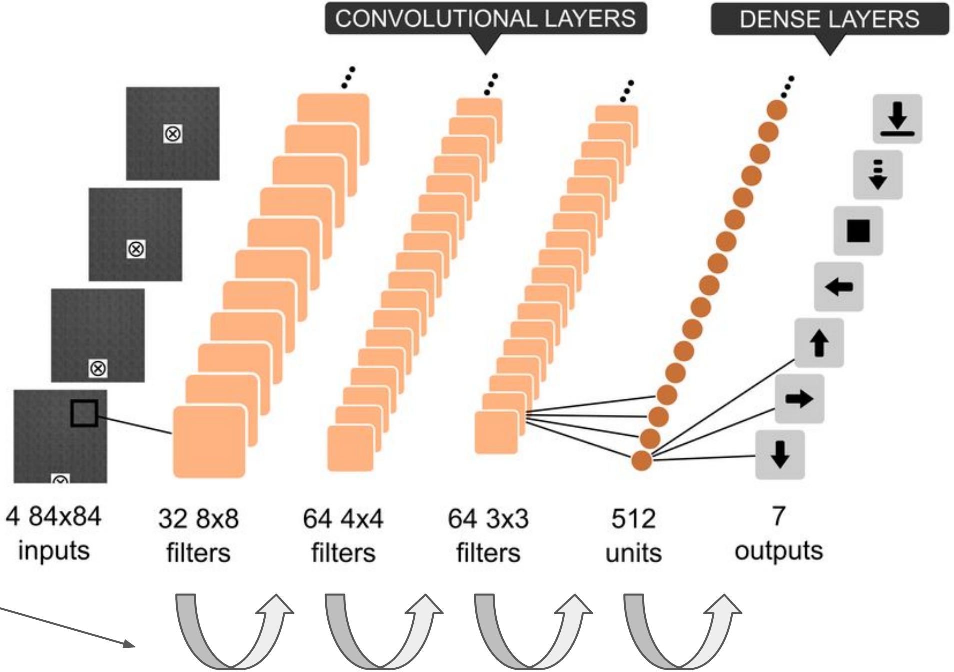
- The most obvious form of planning is to use the forward model for selecting actions (i.e. use model to come up with policy)



1. Lay out the plan
2. Choose action accordingly

Explicit vs Implicit Planning

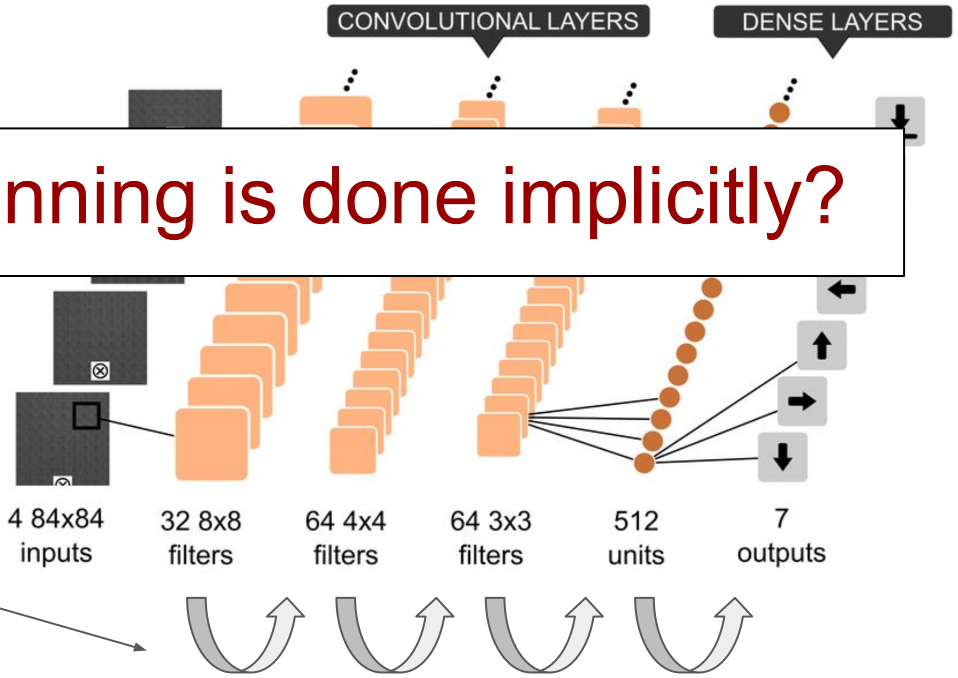
- Do you think DQN plan? How does it arrived at the Q-value?



Are there computations akin to `planning' somewhere here?

Explicit vs Implicit Planning

- Do you think DQN plan? How does it arrived at the Q-value?



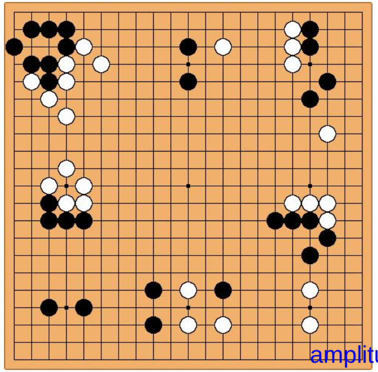
Maybe the planning is done implicitly?

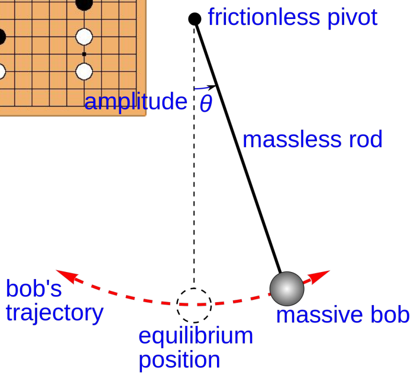
Are there computations akin to `planning' somewhere here?

When we say `model’,
we often mean **`the forward model’**

Different types of forward model

known accurate model





Estimated global model

Estimated local model

Landscape of model-based RL

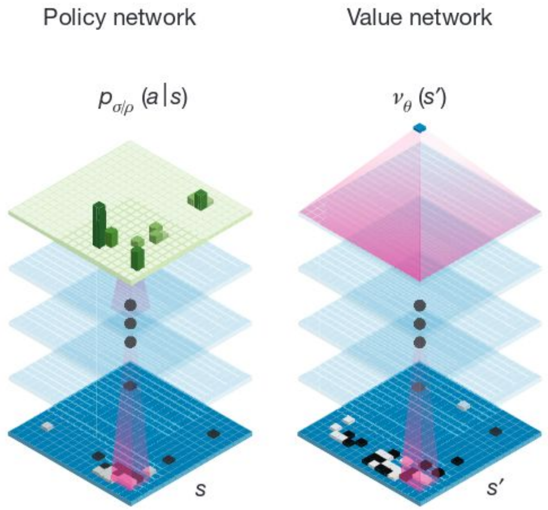
1. AlphaGo, AlphaGo Zero and AlphaZero


Mastering the game of Go with deep neural networks and tree search

David Silver[1*], Aja Huang[1*], Chris J. Maddison[1], Arthur Guez[1], Laurent Sifre[1], George van den Driessche[1], Julian Schrittwieser[1], Ioannis Antonoglou[1], Veda Panneershelvam[1], Marc Lanctot[1], Sander Dieleman[1], Dominik Grewe[1], John Nham[2], Nal Kalchbrenner[1], Ilya Sutskever[2], Timothy Lillicrap[1], Madeleine Leach[1], Koray Kavukcuoglu[1], Thore Graepel[1] & Demis Hassabis[1]

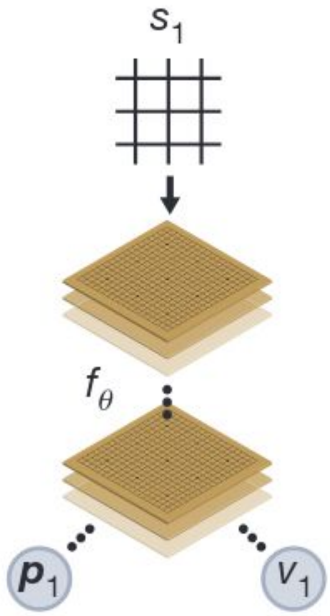
1. AlphaGo, AlphaGo Zero and AlphaZero

AlphaGo architecture



Separate convolutional neural network

AlphaGo Zero and AlphaZero architecture



Combined residual neural network

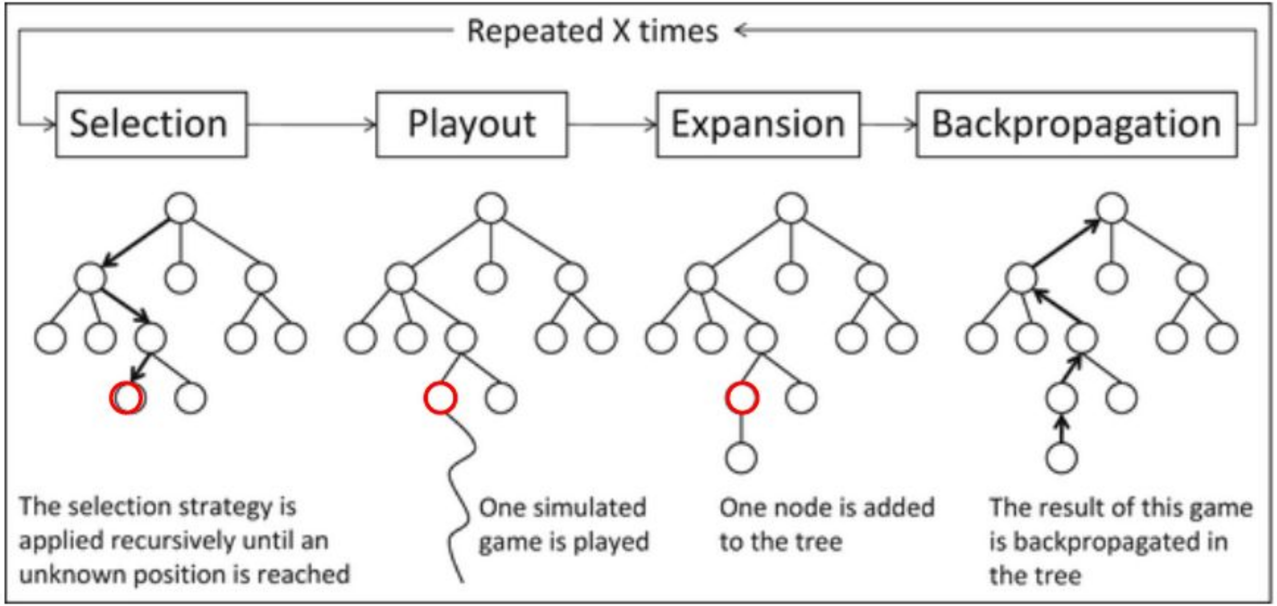
1. AlphaGo, AlphaGo Zero and AlphaZero

- Model of the environment is given (the agent has access to the game rules)

- Use modified Monte Carlo Tree Search to plan explicitly

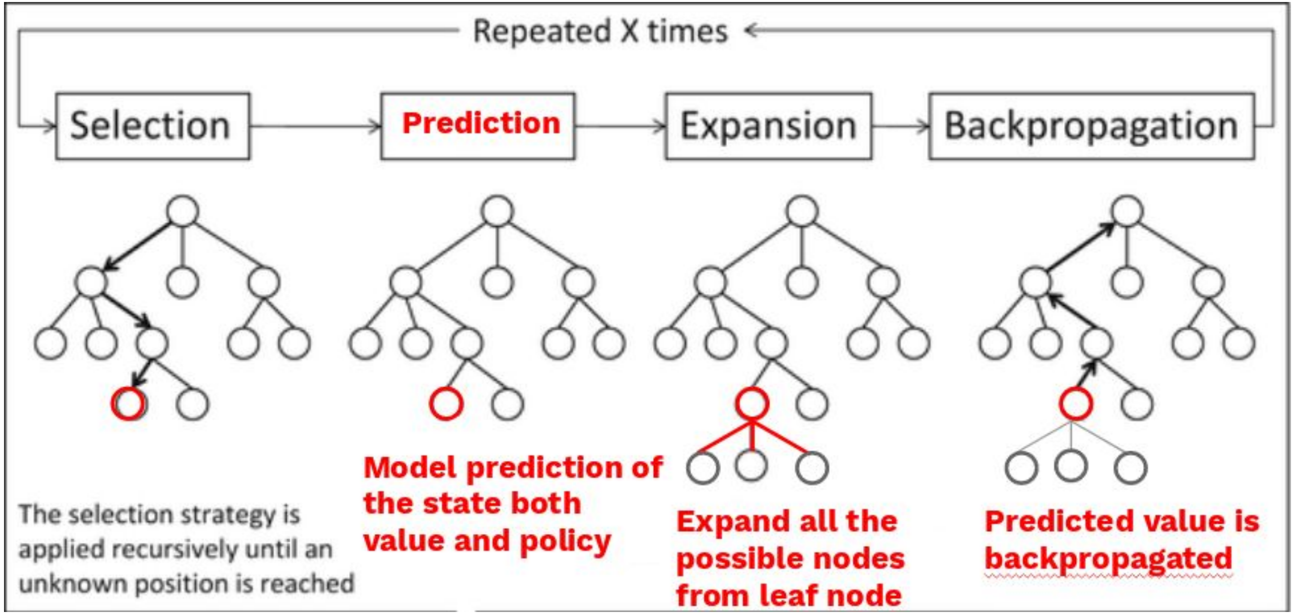
1. AlphaGo, AlphaGo Zero and AlphaZero

Original Monte Carlo Tree Search



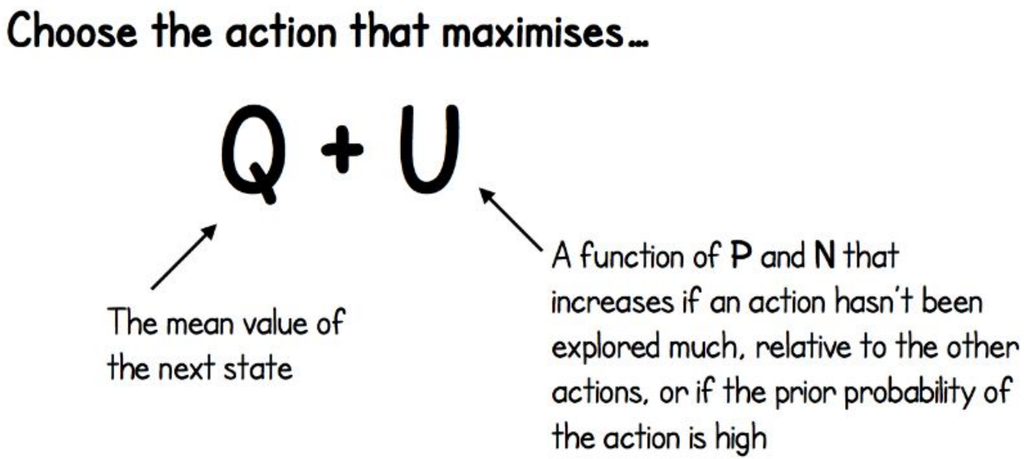
1. AlphaGo, AlphaGo Zero and AlphaZero

Modified Monte Carlo Tree Search



1. AlphaGo, AlphaGo Zero and AlphaZero

Selection strategy



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

1. AlphaGo, AlphaGo Zero and AlphaZero

Properties	AlphaGo	AlphaGo Zero	AlphaZero
Neural network architecture	Separate convolutional neural network	Combined residual neural network	
Data augmentation	Exploit board reflection and rotation to augment training data (unique to Go)		Did not use board exploitation (cannot apply to chess)
Self-play/evaluation	Evaluate against current best player to find new best player and use current best player to generate training data		Continuous training

2. PILCO

Explicit Planning

Global Model

PILCO: A Model-Based and Data-Efficient Approach to Policy Search

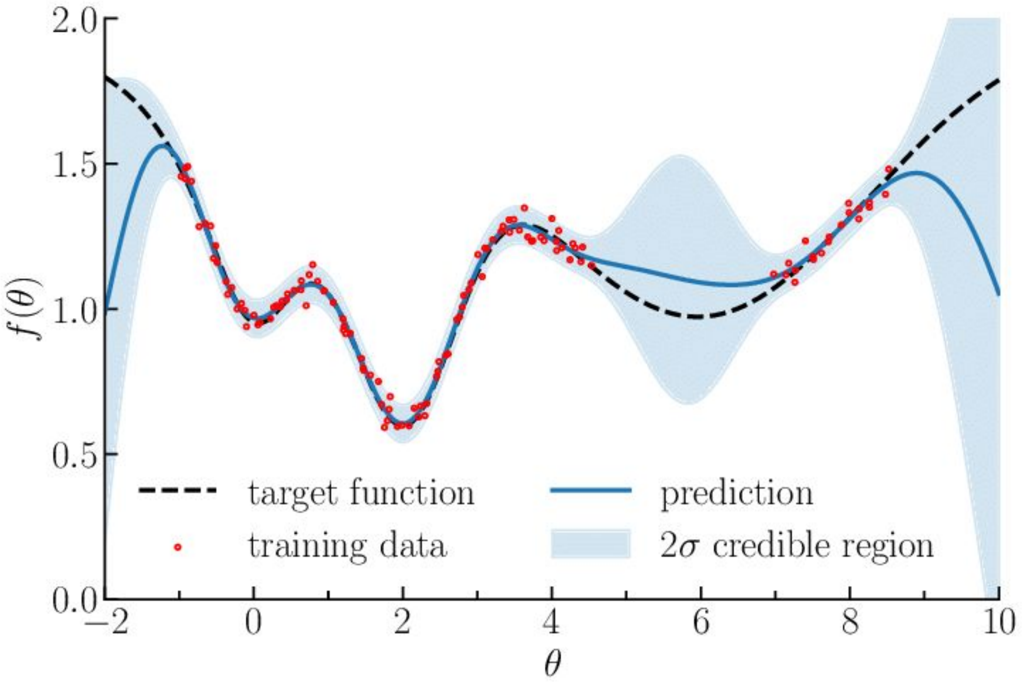
Marc Peter Deisenroth MARC@CS.WASHINGTON.EDU
Department of Computer Science & Engineering, University of Washington, USA

Carl Edward Rasmussen CER54@CAM.AC.UK
Department of Engineering, University of Cambridge, UK

2. PILCO

PILCO's dynamic model is implemented as Gaussian Process

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t \mid \mu_t, \mathbf{\Sigma}_t\right),$$
$$\mu_t = \mathbf{x}_{t-1} + \mathbb{E}_f[\Delta_t],$$
$$\mathbf{\Sigma}_t = \mathrm{var}_f[\Delta_t].$$



2. PILCO

- PILCO perform policy evaluation & policy improvement

- Policy Evaluation is done by doing Bayesian Inference to find the density of the trajectory (to draw) P(x)

- Policy Improvement is done with non-convex optimisation on the analytic gradients

2. PILCO

Algorithm 1 PILCO

init: Sample controller parameters $\theta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Apply random control signals and record data.

repeat

  Learn probabilistic (GP) dynamics model, see Sec. 2.1, using all data.

  Model-based policy search, see Sec. 2.2–2.3.

  repeat

    Approximate inference for policy evaluation, see Sec. 2.2: get $J^\pi(\theta)$, Eqs. (10)–(12), (24).

    Gradient-based policy improvement, see Sec. 2.3: get $\mathrm{d}J^\pi(\theta)/\,\mathrm{d}\theta$, Eqs. (26)–(30).

    Update parameters θ (e.g., CG or L-BFGS).

  until convergence; **return** θ^*

  Set $\pi^* \leftarrow \pi(\theta^*)$.

  Apply π^* to system (single trial/episode) and record data.

until task learned

2. PILCO

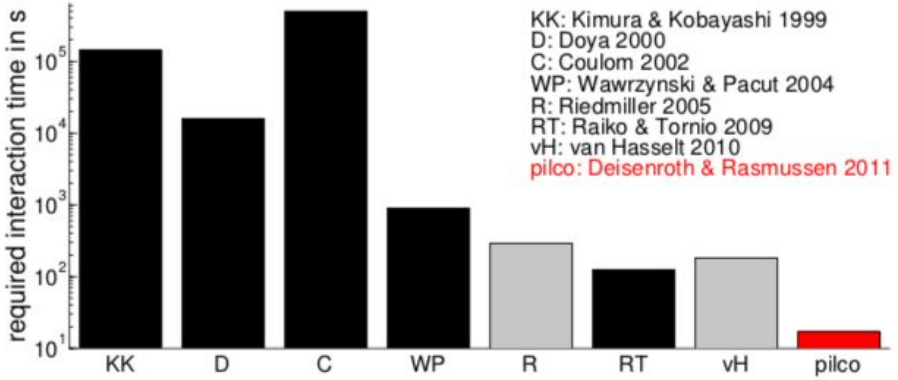


Figure 5. Data efficiency for learning the cart-pole task in the absence of expert knowledge. The horizontal axis chronologically orders the references according to their publication date. The vertical axis shows the required interaction time with the cart-pole system on a log-scale.

Figure 3. Real cart-pole system. Snapshots of a controlled trajectory of 20 s length after having learned the task. To solve the swing-up plus balancing, PILCO required only 17.5 s of interaction with the physical system.

3. World Models

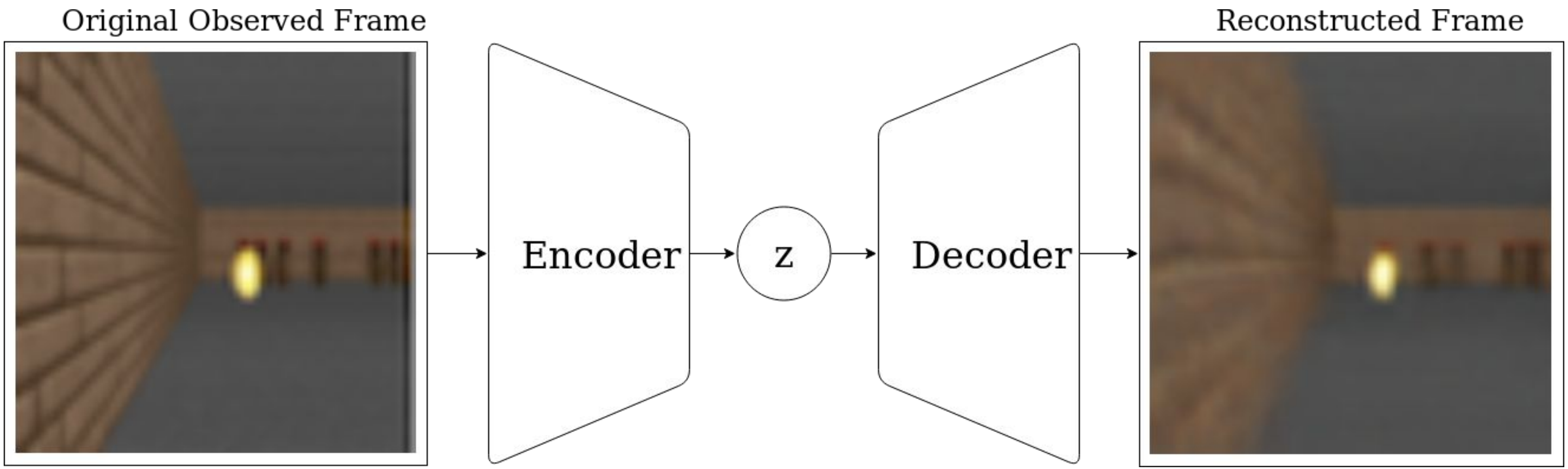
- There are 3 parts in World Models, VAE, RNN and a (linear) controller model

- Learn representation of the space (z) using VAE
- Then feed z to RNN to predict next z and use hidden state of RNN to be representation of time (h)
- Controller gets inputs z and h from VAE and RNN respectively

- The goal is to train the controller to maximize expected cumulative reward from environment

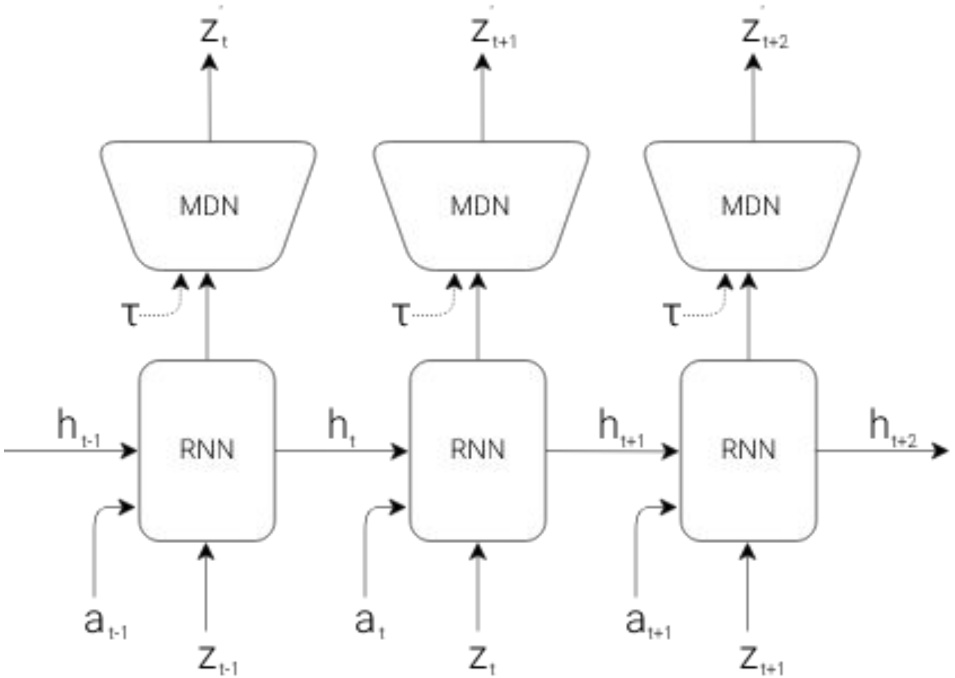
3. World Models

Learn representation of the space (z) using VAE



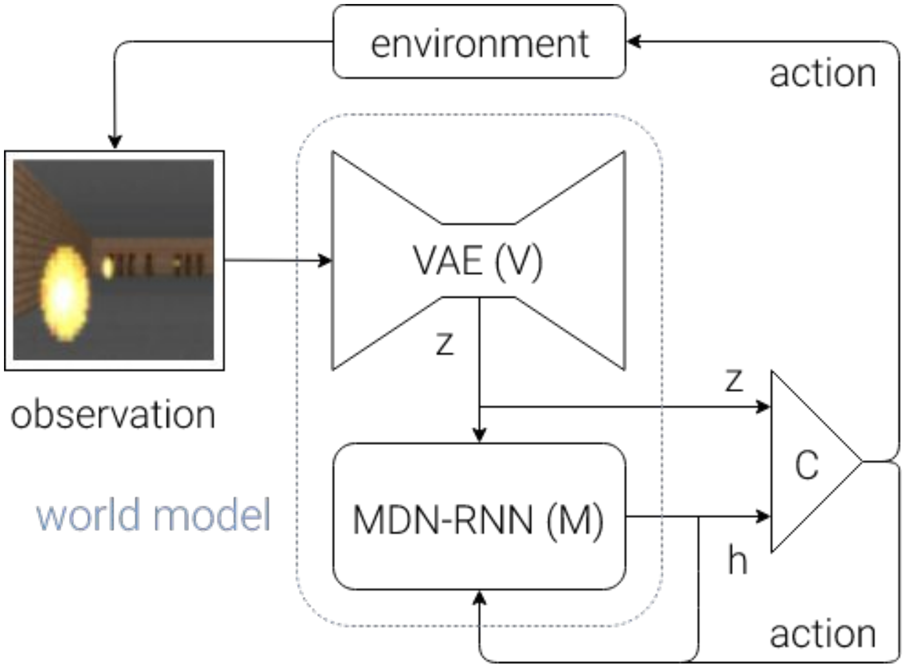
3. World Models

Learn representation of time (h) using RNN



3. World Models

Putting everything together




```python
def rollout(controller):
  ''' env, rnn, vae are '''
  ''' global variables  '''
  obs = env.reset()
  h = rnn.initial_state()
  done = False
  cumulative_reward = 0
  while not done:
    z = vae.encode(obs)
    a = controller.action([z, h])
    obs, reward, done = env.step(a)
    cumulative_reward += reward
    h = rnn.forward([a, z, h])
  return cumulative_reward
```

3. World Models

- Learned representation can be used as a virtual environment
  (the author called this “dream”)

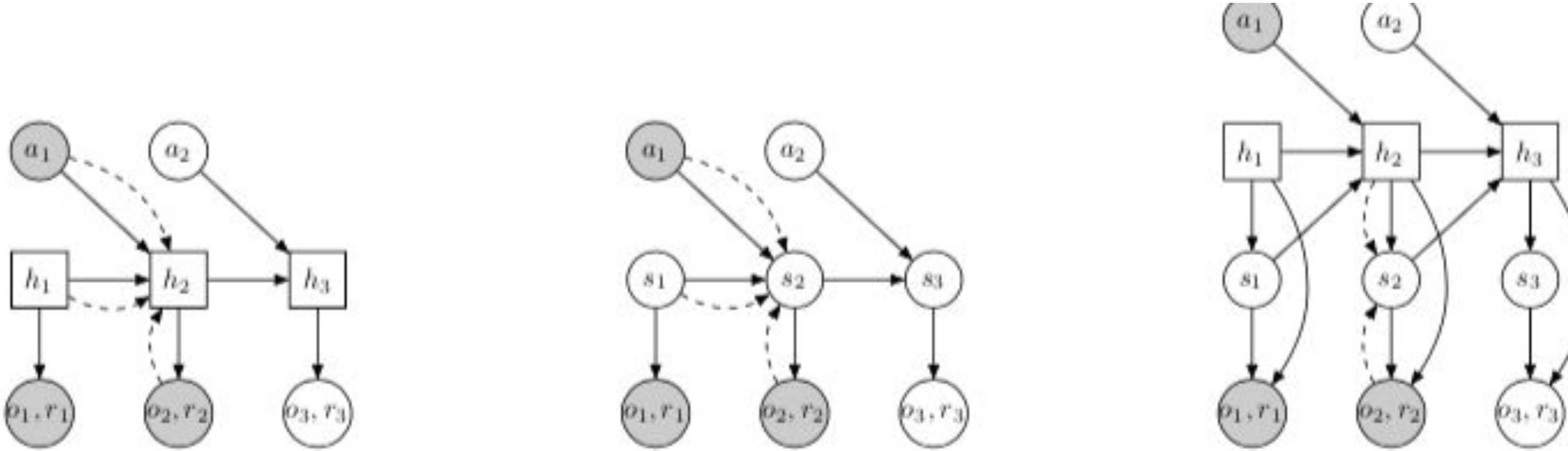
- The policy learned in actual environment can be used in dream environment
  (https://worldmodels.github.io/) (Car Racing Dreams section)

- Also, the controller policy can be learned inside the dream and transfer back to the actual environment
  (train VAE and RNN first, then train the controller in the dream)
  (https://worldmodels.github.io/) (VizDoom section)
  (The Rnn in this part has to predict if the episode is ‘done’)

3. World Models

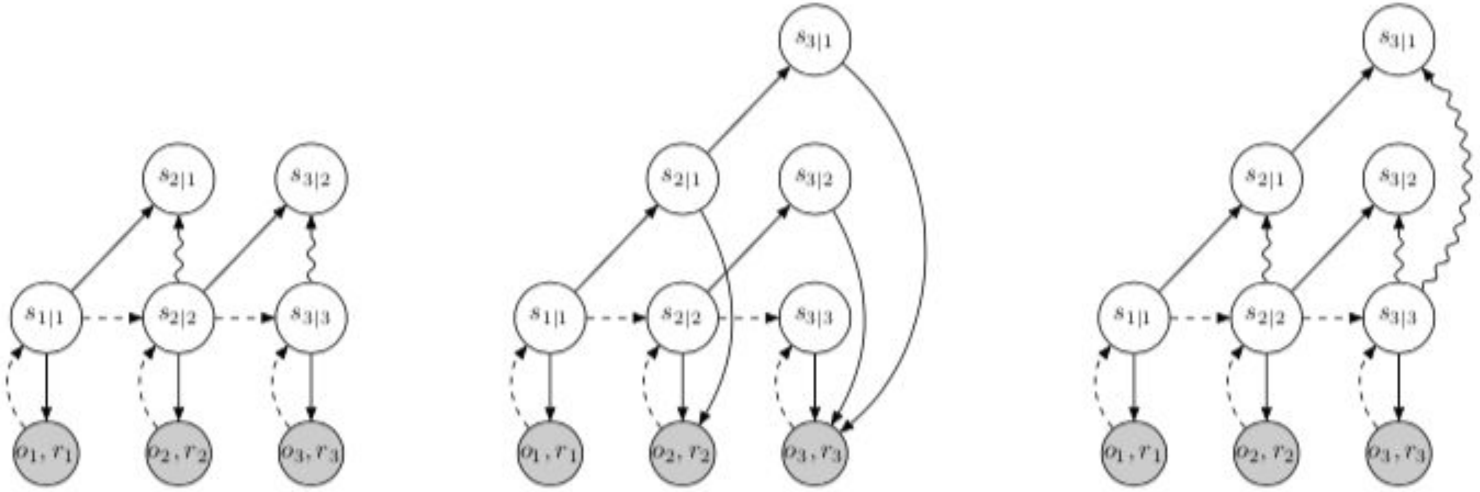
Learning Latent Dynamics for Planning from Pixels

Danijar Hafner [1 2] **Timothy Lillicrap** [3] **Ian Fischer** [4] **Ruben Villegas** [1 5]
David Ha [1] **Honglak Lee** [1] **James Davidson** [1]



(a) Deterministic model (RNN) (b) Stochastic model (SSM) (c) Recurrent state-space model (RSSM)

3. World Models



(a) Standard variational bound (b) Observation overshooting (c) Latent overshooting

4. Guided Policy Search

Explicit Planning   Local Model

Learning Contact-Rich Manipulation Skills with Guided Policy Search

Sergey Levine, Nolan Wagener, Pieter Abbeel

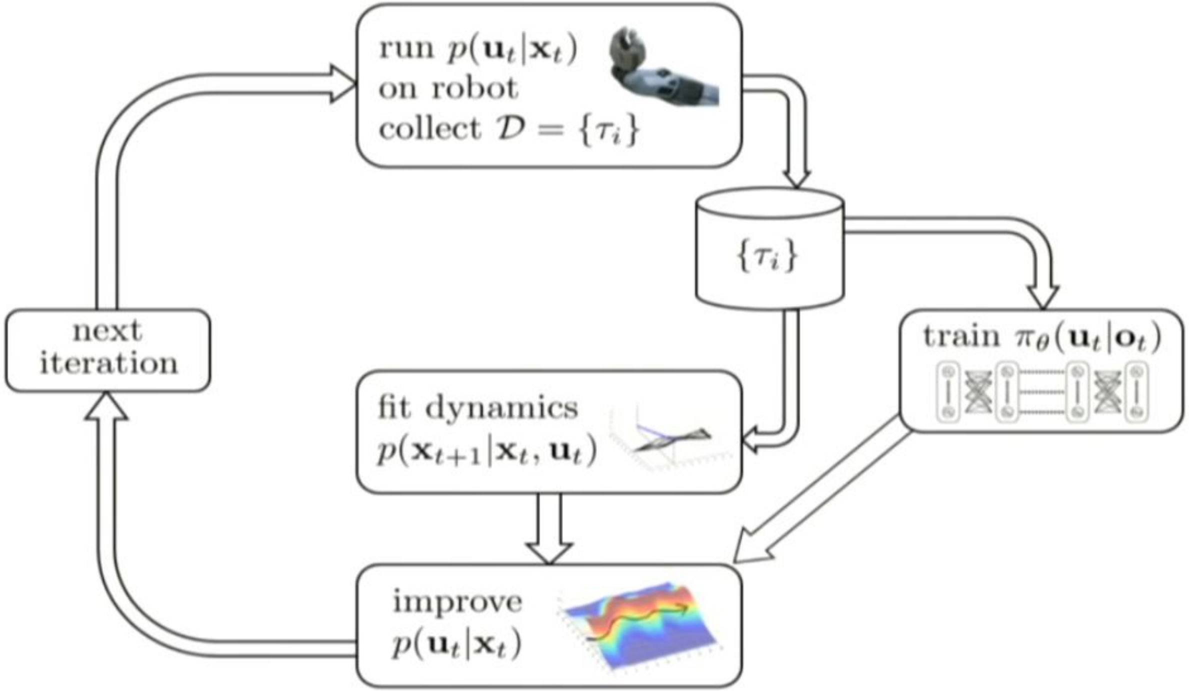
Guided Policy Search

Sergey Levine SVLEVINE@STANFORD.EDU
Vladlen Koltun VLADLEN@STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305 USA

4. Guided Policy Search

1. At each local region, learn local linear model
2. Given the local model and reward model, ***solve for the best (local) policy***
3. Train global policy to match local policy (with ***supervised learning***!)
4. Repeat to cover many region!

4. Guided Policy Search



5. Dyna-Q and NAF

Continuous Deep Q-Learning with Model-based Acceleration

Shixiang Gu[1 2 3] SG717@CAM.AC.UK
Timothy Lillicrap[4] COUNTZERO@GOOGLE.COM
Ilya Sutskever[3] ILYASU@GOOGLE.COM
Sergey Levine[3] SLEVINE@GOOGLE.COM
[1]University of Cambridge [2]Max Planck Institute for Intelligent Systems [3]Google Brain [4]Google DeepMind

Dyna-Style Planning with Linear Function Approximation and Prioritized Sweeping

Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, Michael Bowling
Reinforcement Learning and Artificial Intelligence Laboratory
Department of Computing Science, University of Alberta, Edmonton, AB Canada T6G 2E8

5. Dyna-Q and NAF

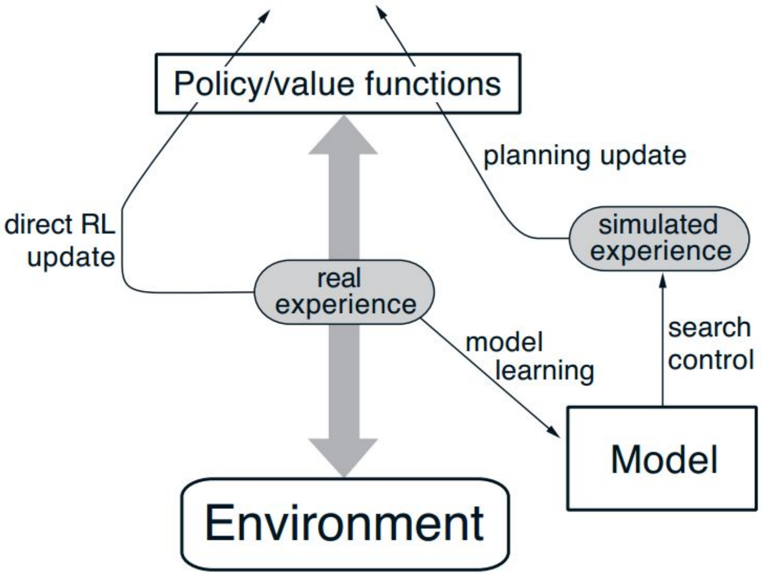


Figure 8.2: The general Dyna Architecture. Real experience, passing back and forth between the environment and the policy, affects policy and value functions in much the same way as does simulated experience generated by the model of the environment.

5. Dyna-Q and NAF

Tabular Dyna-Q

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) $A \leftarrow \epsilon$-greedy(S, Q) -> what's this ? input state, Q-learning table; output: epsilon greedy action ? probably
(c) Execute action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$ max action in S'
(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
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Random-sample one-step tabular Q-planning

Do forever:
1. Select a state, $S \in \mathcal{S}$, and an action, $A \in \mathcal{A}(s)$, at random
2. Send S, A to a sample model, and obtain
a sample next reward, R, and a sample next state, S'
3. Apply one-step tabular Q-learning to S, A, R, S':
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

5. Dyna-Q and NAF

- Model-guided exploration
- Imagination roll-out

6. Successor Representation

- Instead of learning Q-value, we find ‘the expected discounted future state occupancy.

$$M(s, s', a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}[s_t = s'] | s_0 = s, a_0 = a\right],$$

$$Q^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} M(s, s', a) R(s')$$

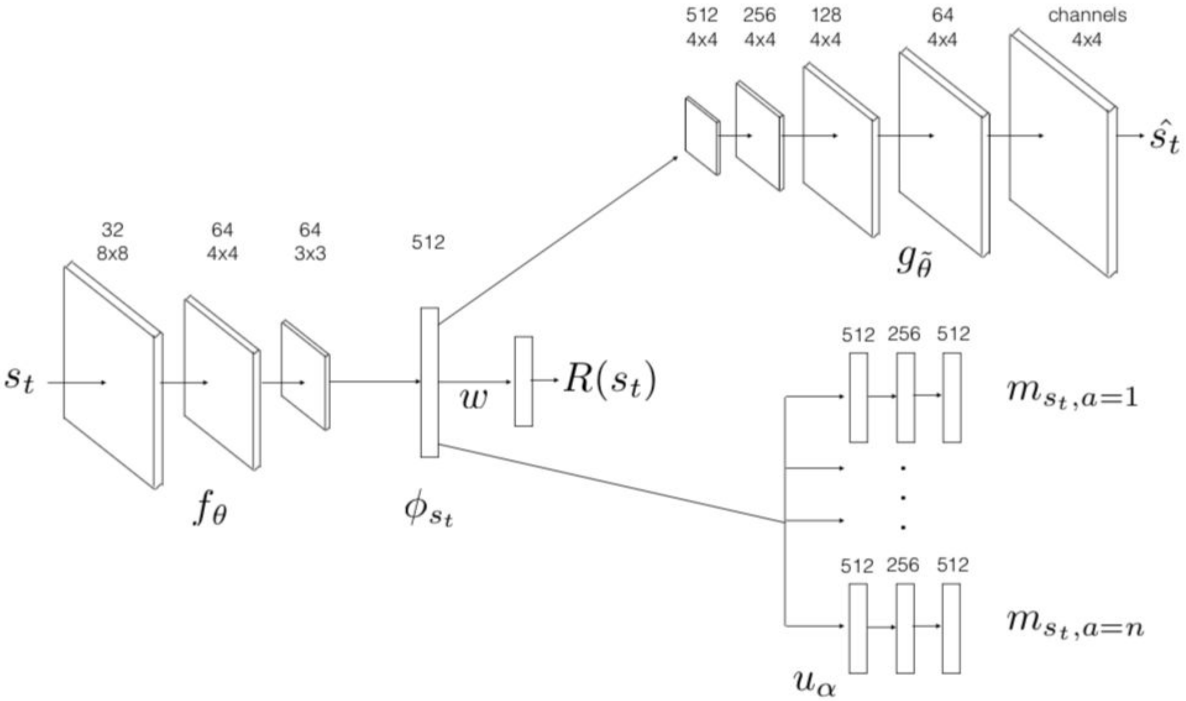
$$M(s, s', a) = \mathbb{1}[s_t = s'] + \gamma \mathbb{E}[M(s_{t+1}, s', a_{t+1})].$$

6. Successor Representation

- Successor Representation encapsulates dynamic of a policy with a specific transition function.

- Suited for transferring knowledge from different task with the same transition function!

6. Successor Representation



$$
\begin{aligned}
Q^{\pi}(s,a) &= \mathrm{E}^{\pi}\left[r_{t+1} + \gamma r_{t+2} + \dots \mid S_t = s, A_t = a\right] \\
&= \mathrm{E}^{\pi}\left[\boldsymbol{\phi}_{t+1}^{\top}\mathbf{w} + \gamma \boldsymbol{\phi}_{t+2}^{\top}\mathbf{w} + \dots \mid S_t = s, A_t = a\right] \\
&= \mathrm{E}^{\pi}\left[\textstyle\sum_{i=t}^{\infty} \gamma^{i-t} \boldsymbol{\phi}_{i+1} \mid S_t = s, A_t = a\right]^{\top} \mathbf{w} = \boldsymbol{\psi}^{\pi}(s,a)^{\top}\mathbf{w}.
\end{aligned}
$$

6. Successor Representation

Similar to the successor representation,

we could also learn the `successor feature’.

7. Value Iteration Network

Value Iteration Networks

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel

Dept. of Electrical Engineering and Computer Sciences, UC Berkeley

7. Value Iteration Network

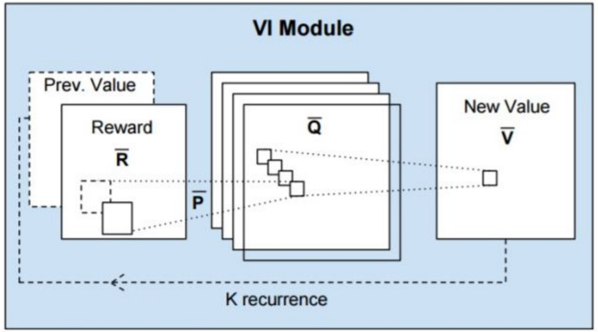
- VIN has a planning `module’ embedded in an end-to-end deep network.
- It learns to plan!

7. Value Iteration Network

Normal Value iteration

$$V_{n+1}(s) = \max_a Q_n(s,a) \quad \forall s, \quad \text{where}$$

$$Q_n(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_n(s'). \quad (1)$$



8. Predictron

The Predictron: End-To-End Learning and Planning

David Silver [*1] Hado van Hasselt [*1] Matteo Hessel [*1] Tom Schaul [*1] Arthur Guez [*1] Tim Harley [1]
Gabriel Dulac-Arnold [1] David Reichert [1] Neil Rabinowitz [1] Andre Barreto [1] Thomas Degris [1]

8. Predictron

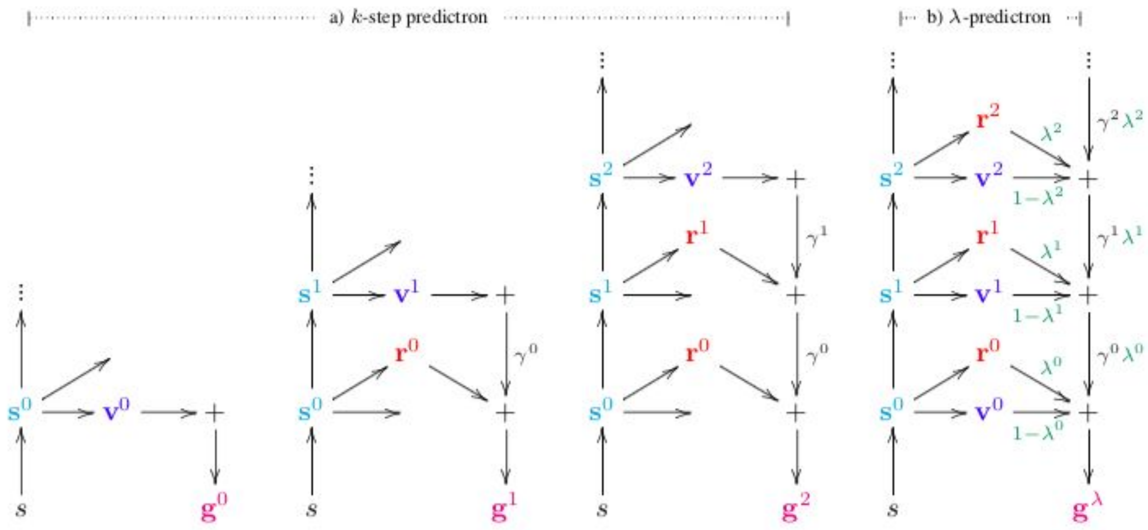


Figure 1. a) The k-step predictron architecture. The first three columns illustrate 0, 1 and 2-step pathways through the predictron. The 0-step preturn reduces to standard model-free value function approximation; other preturns "imagine" additional steps with an internal model. Each pathway outputs a k-step preturn $\mathbf{g}^k$ that accumulates discounted rewards along with a final value estimate. In practice all k-step preturns are computed in a single forward pass. b) The λ-predictron architecture. The λ-parameters gate between the different preturns. The output is a λ-preturn $\mathbf{g}^\lambda$ that is a mixture over the k-step preturns. For example, if $\boldsymbol{\lambda}^0 = \mathbf{1}, \boldsymbol{\lambda}^1 = \mathbf{1}, \boldsymbol{\lambda}^2 = \mathbf{0}$ then we recover the 2-step preturn, $\mathbf{g}^\lambda = \mathbf{g}^2$. Discount factors $\boldsymbol{\gamma}^k$ and λ-parameters $\boldsymbol{\lambda}^k$ are dependent on state $\mathbf{s}^k$; this dependence is not shown in the figure.

8. Predictron

Conclusion