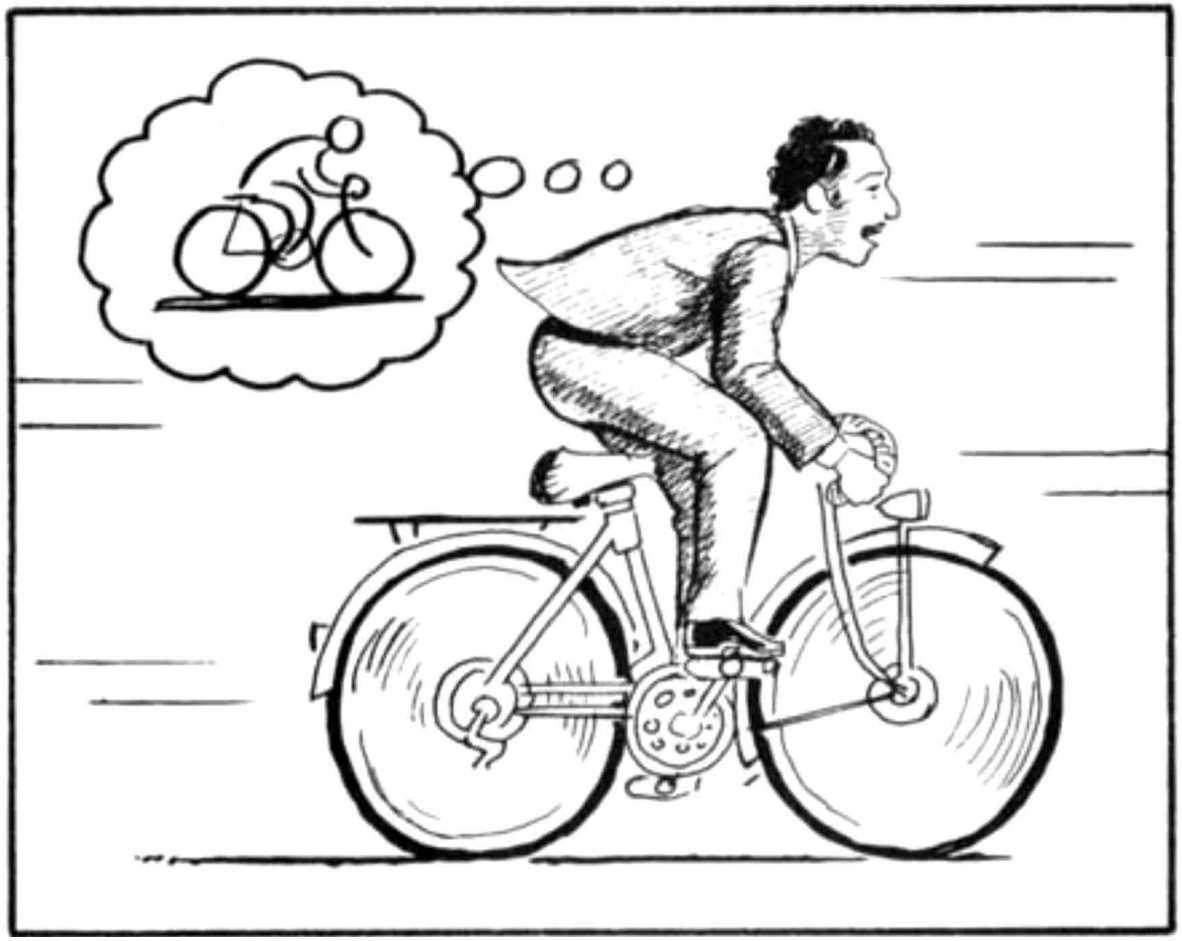
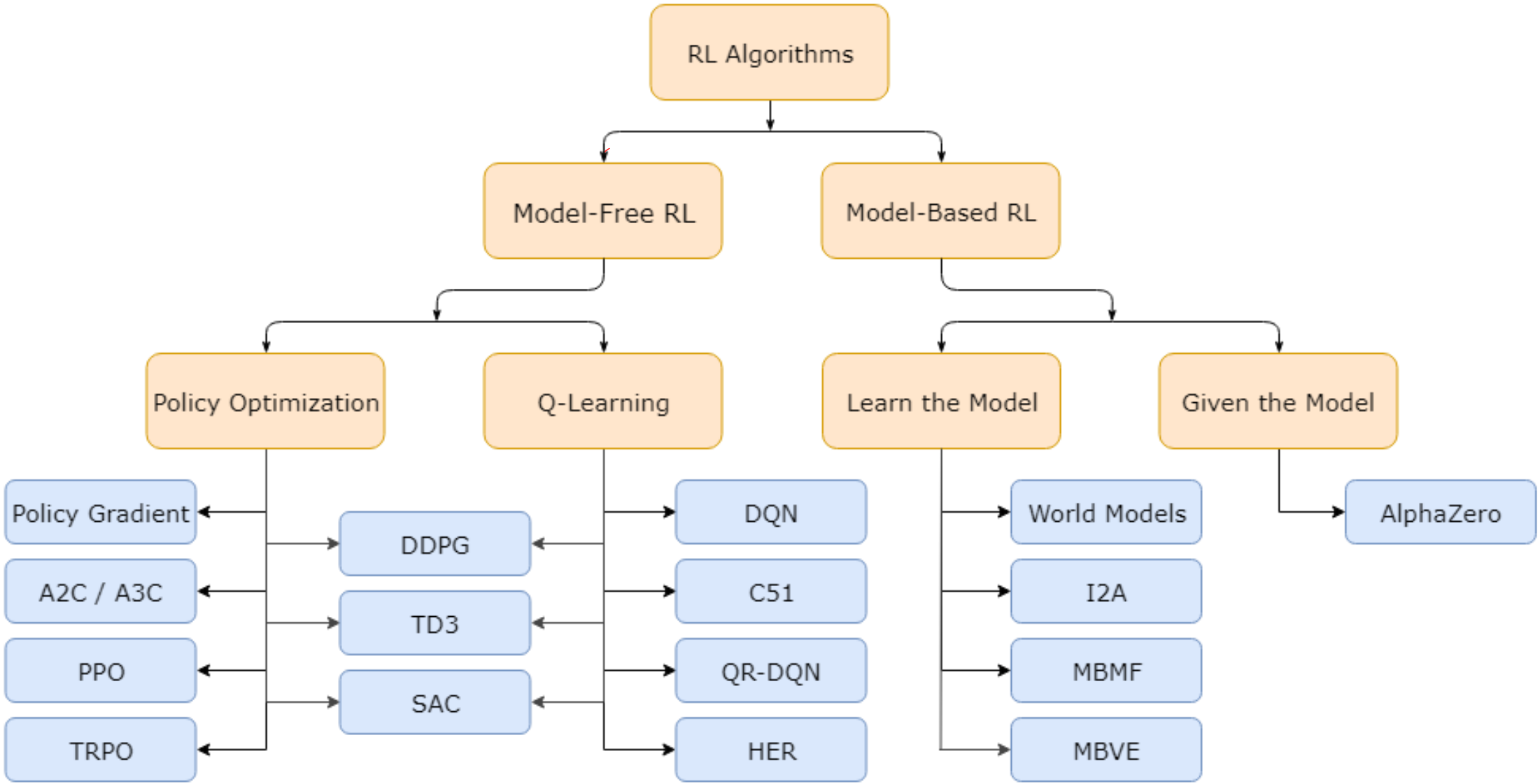
World Models and Model-Based RL

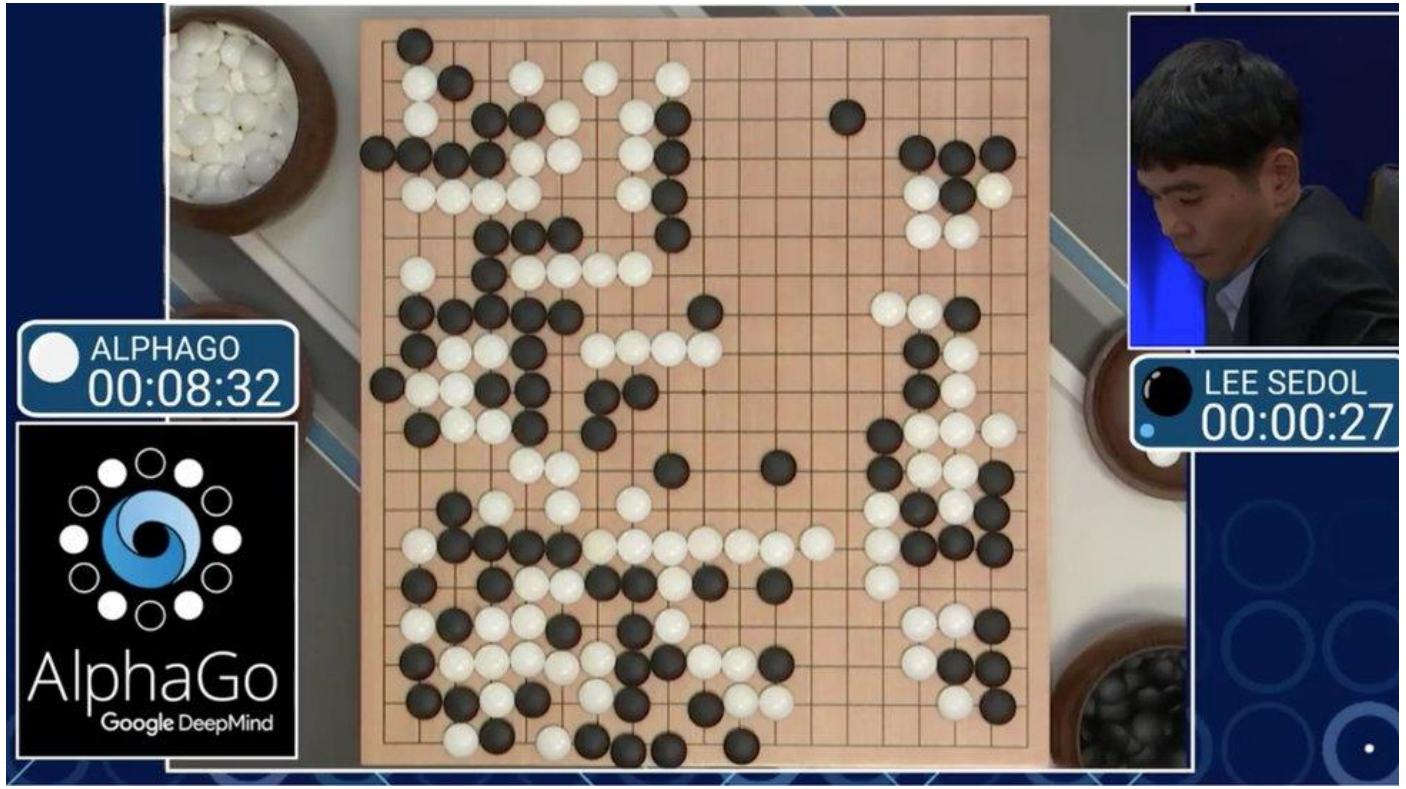


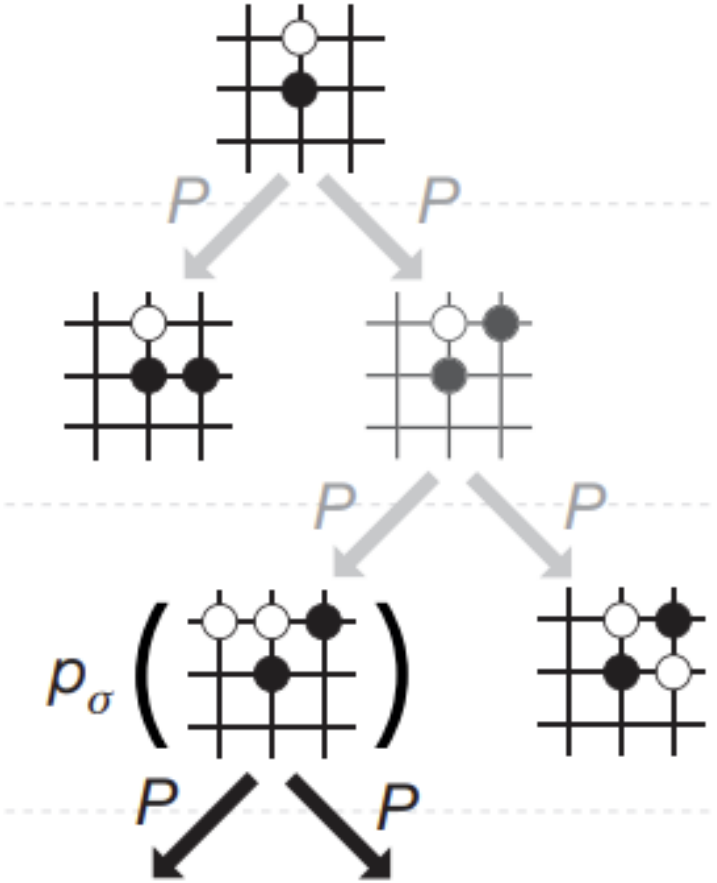
Instructor: Daniel Brown --- University of Utah

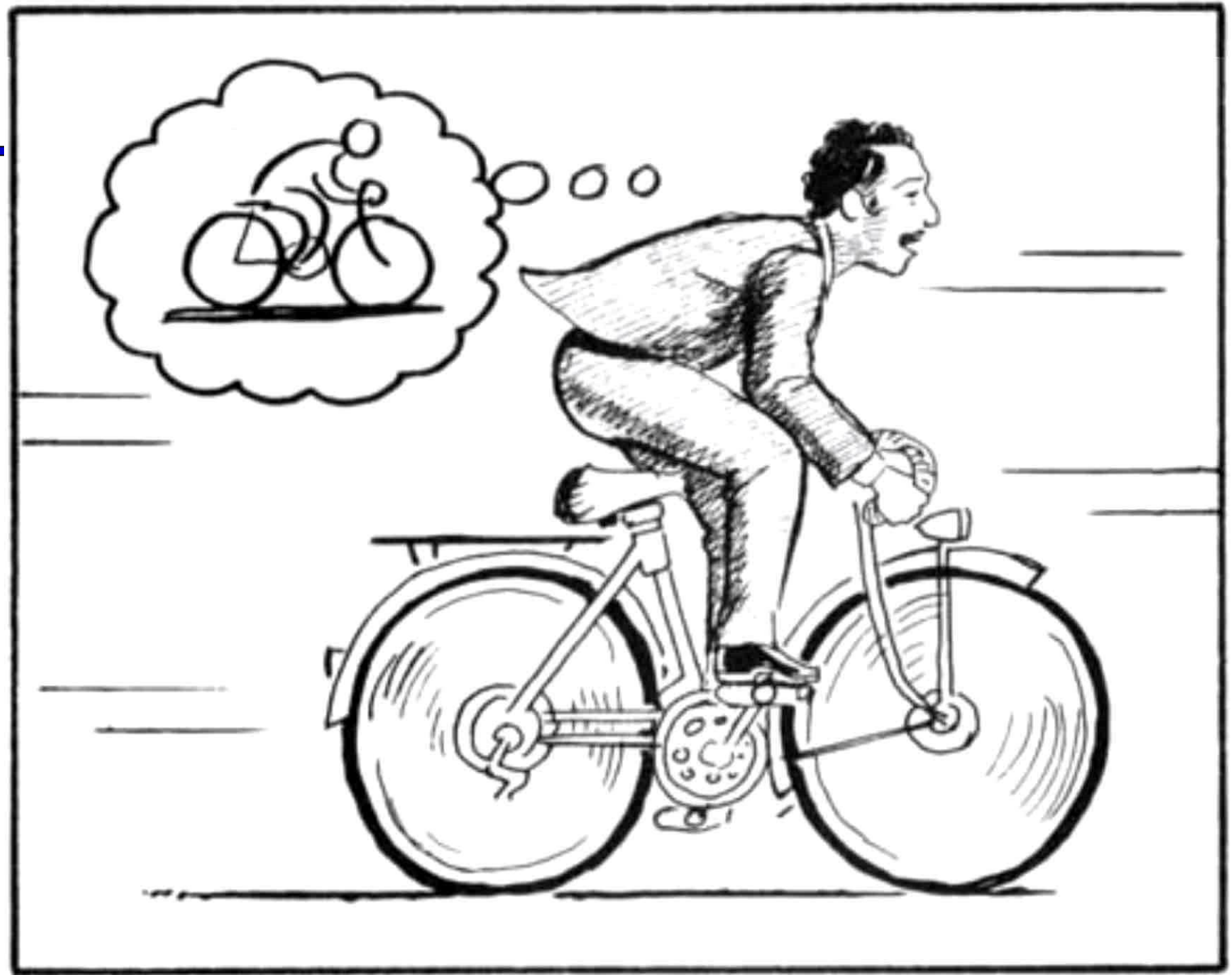
Rough Taxonomy of RL Algorithms



We've already seen model-based RL







Why use planning over model-free RL?

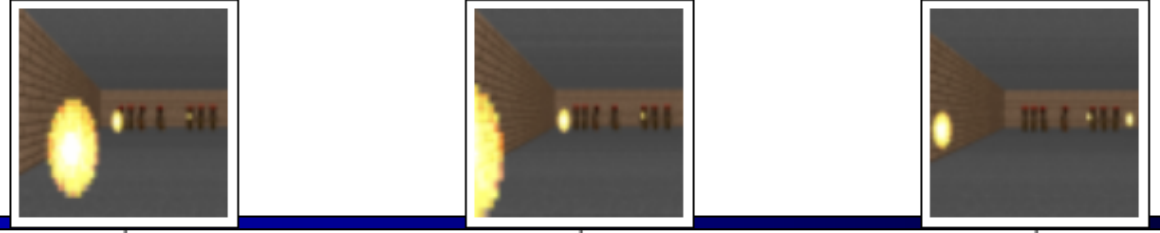
$\pi(s) \rightarrow a$

- More data efficient
- Increased performance just by increasing compute budget for search
- Learned dynamics are task independent

World Models

David Ha [1] **Jürgen Schmidhuber** [2 3]

At each time step, our agent receives an **observation** from the environment.



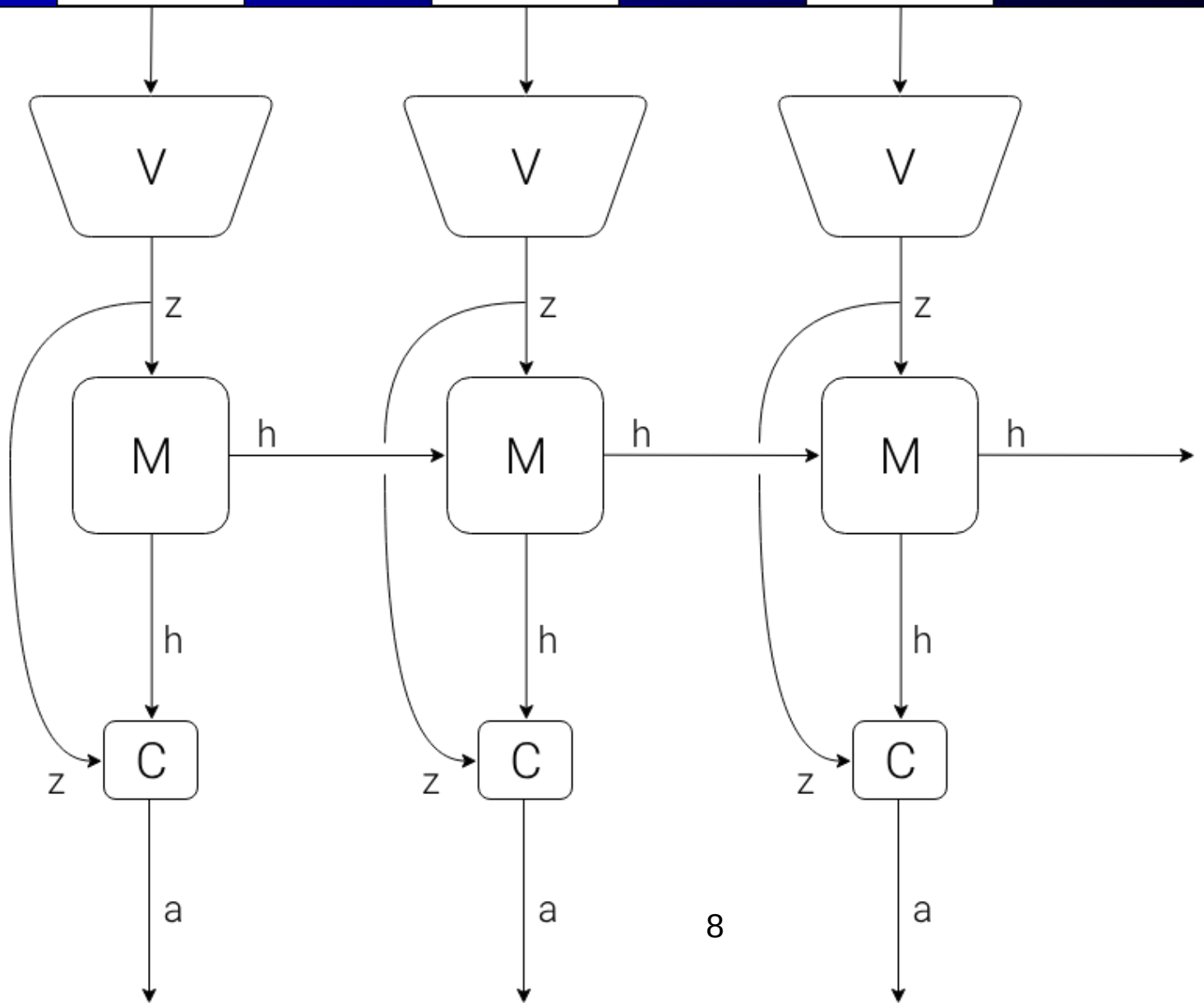
World Model

The **Vision Model (V)** encodes the high-dimensional observation into a low-dimensional latent vector.

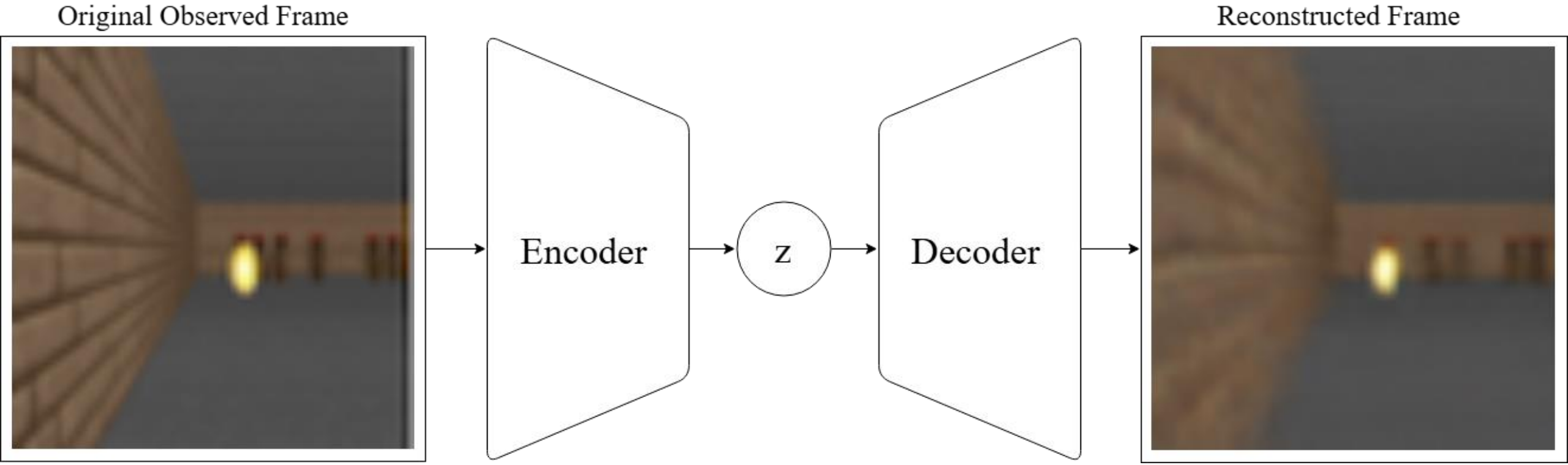
The **Memory RNN (M)** integrates the historical codes to create a representation that can predict future states.

A small **Controller (C)** uses the representations from both **V** and **M** to select good actions.

The agent performs **actions** that go back and affect the environment.



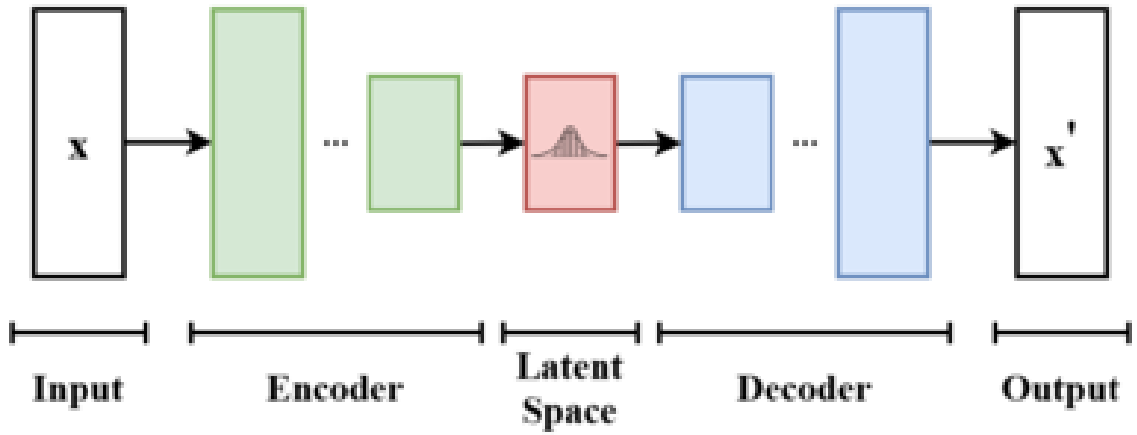
Vision Model



Demo: https://worldmodels.github.io/

Variational Autoencoders (VAEs)

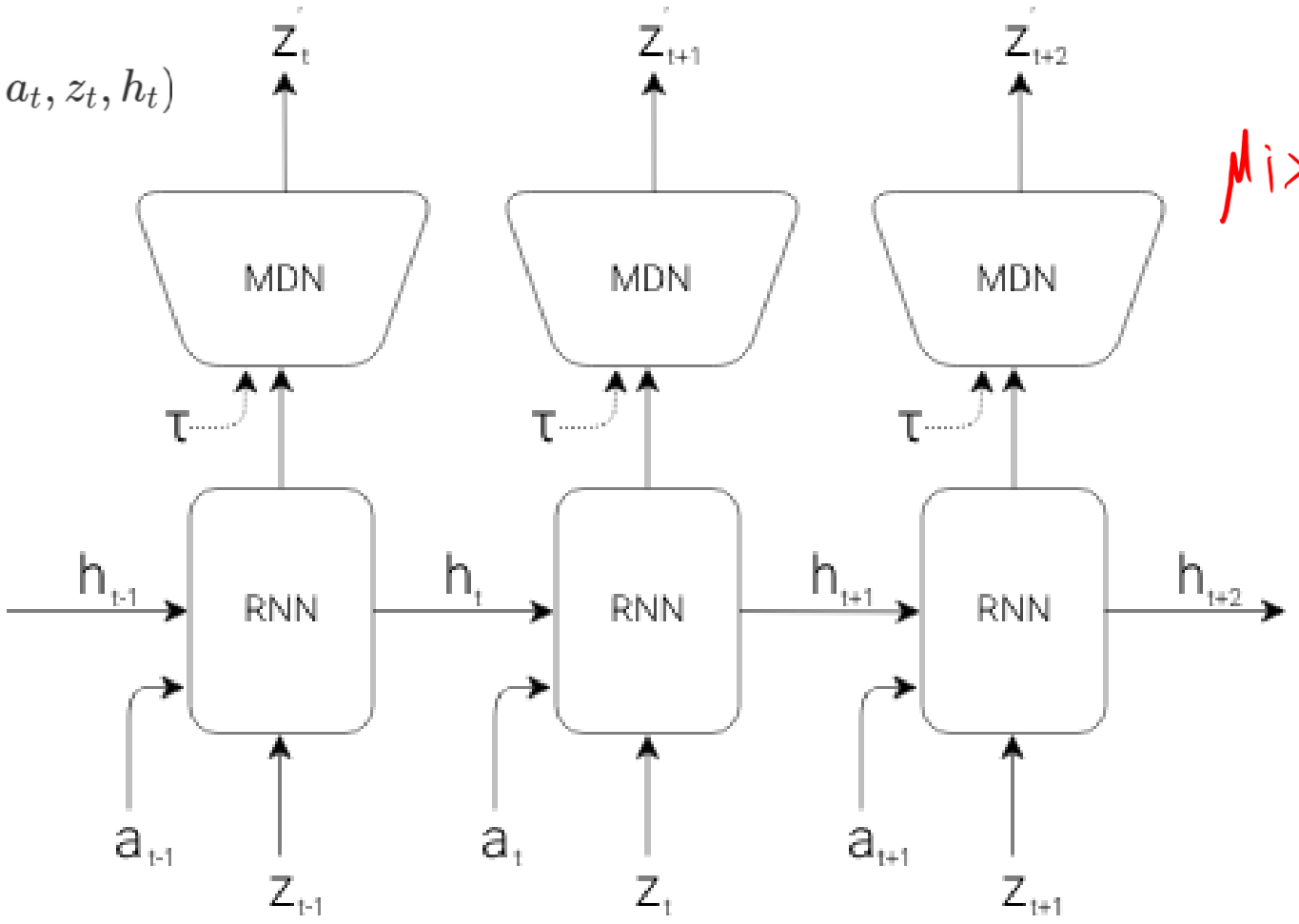
- Autoencoders learn latent representations
- VAEs map input into a distribution over latent variables z
- Loss function is reconstruction plus KL divergence



$$\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{\mathrm{KL}}(q(z|x)||p(z))$$

Memory Model

the RNN will model $P(z_{t+1} \mid a_t, z_t, h_t)$



Mixture Density Network

Simple Controller

At each time step, our agent receives an **observation** from the environment.

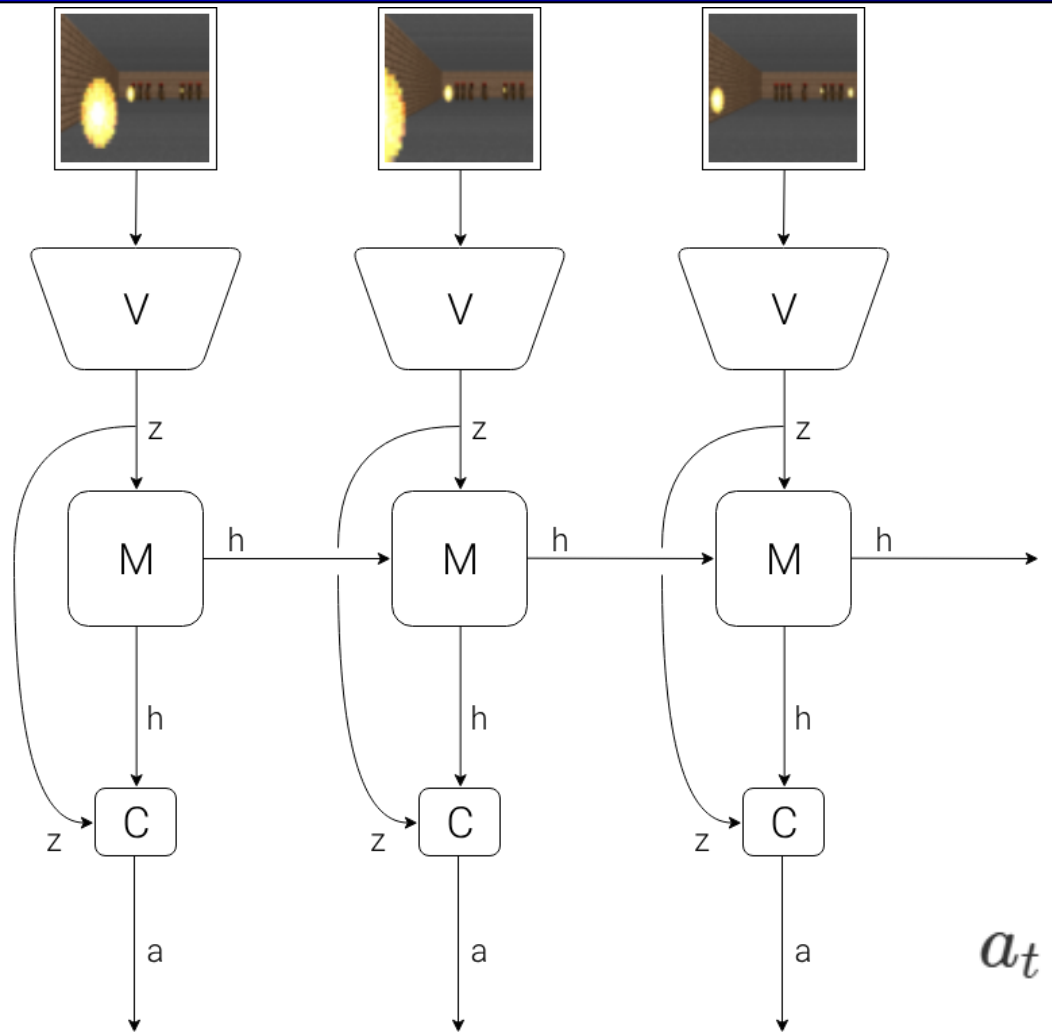
World Model

The **Vision Model (V)** encodes the high-dimensional observation into a low-dimensional latent vector.

The **Memory RNN (M)** integrates the historical codes to create a representation that can predict future states.

A small **Controller (C)** uses the representations from both **V** and **M** to select good actions.

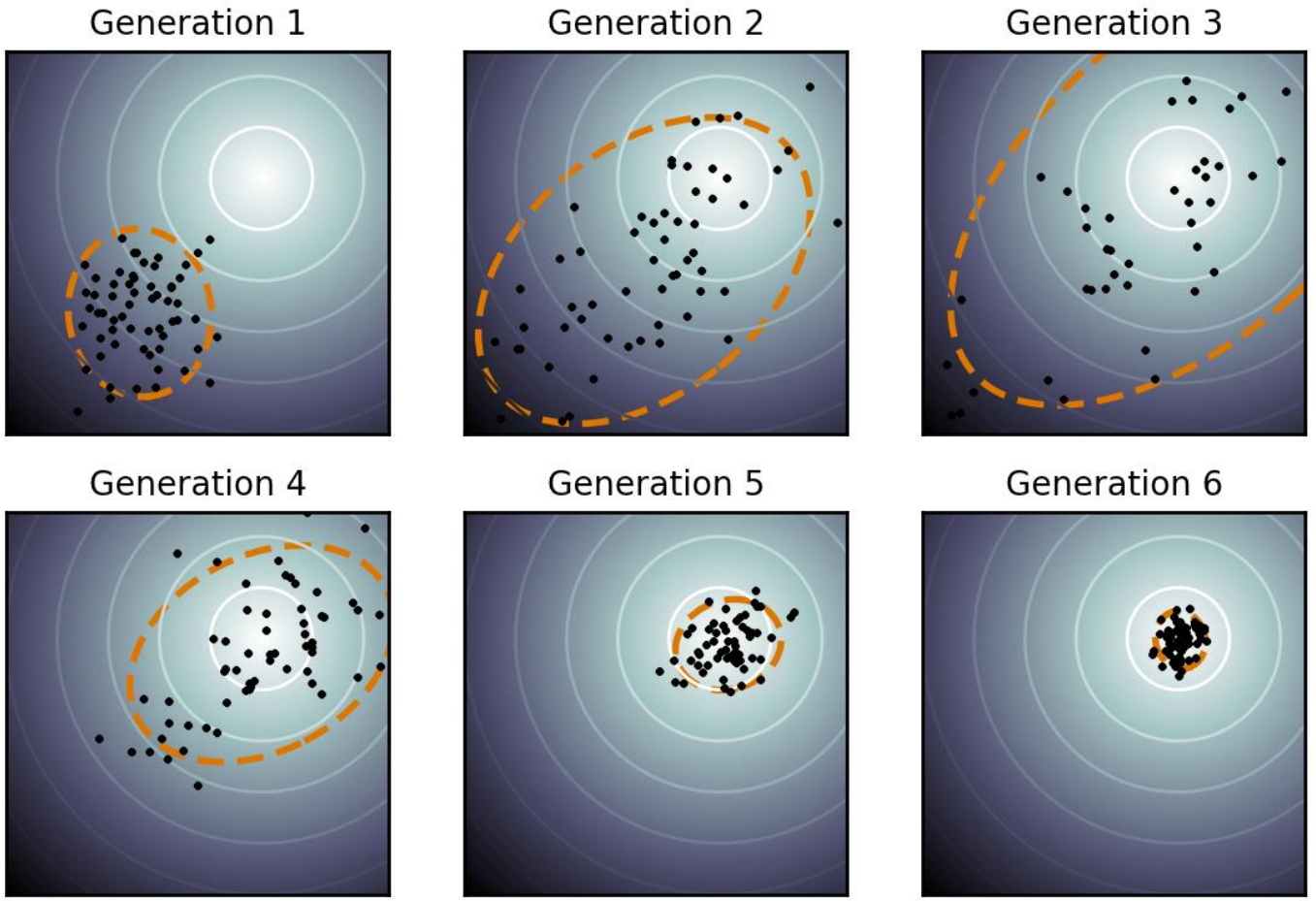
The agent performs **actions** that go back and affect the environment.



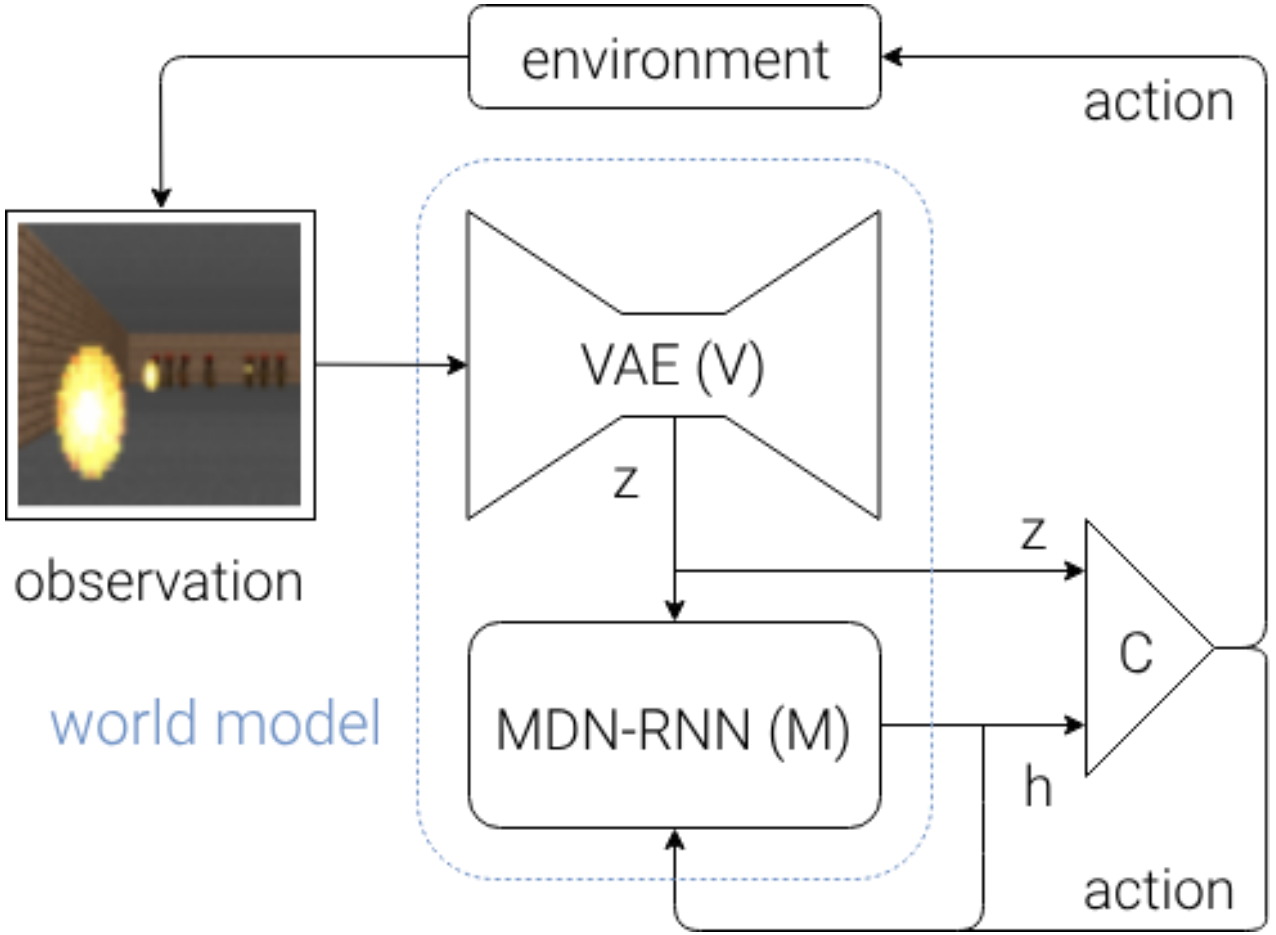
$$a_t = W_c \left[z_t \; h_t \right] + b_c$$

CMA-ES

- Covariance matrix adaptation evolution strategy



Putting everything together



Algorithm

- https://worldmodels.github.io/

Procedure

To summarize the Car Racing experiment, below are the steps taken:

1. Collect 10,000 rollouts from a random policy.
2. Train VAE (V) to encode each frame into a latent vector $z \in \mathcal{R}^{32}$.
3. Train MDN-RNN (M) to model $P(z_{t+1} \mid a_t, z_t, h_t)$.
4. Evolve Controller (C) to maximize the expected cumulative reward of a rollout.

Model	Parameter Count
VAE	4,348,547
MDN-RNN	422,368
Controller	867

METHOD	AVG. SCORE
DQN (PRIEUR, 2017)	343 ± 18
A3C (CONTINUOUS) (JANG ET AL., 2017)	591 ± 45
A3C (DISCRETE) (KHAN & ELIBOL, 2016)	652 ± 10
CEOBILLIONAIRE (GYM LEADERBOARD)	838 ± 11
V MODEL	632 ± 251
V MODEL WITH HIDDEN LAYER	788 ± 141
FULL WORLD MODEL	$\mathbf{906 \pm 21}$

Table 1. `CarRacing-v0` scores achieved using various methods.

PlaNet

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]

PlaNet

Key Improvements over World Models:

- Uncertainty modeling (probabilistic latent states).
- End-to-end differentiable training.
- Better sample efficiency and planning accuracy.

PlaNet

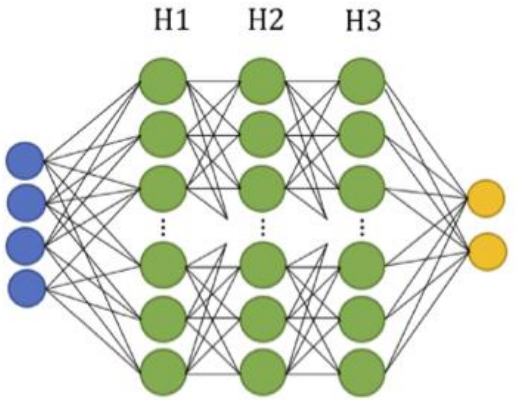
- Iteratively collects data and trains a latent dynamics model
- Models world as a Partially Observable MDP (POMDP)

Transition function:	$s_t \sim \mathrm{p}(s_t \mid s_{t-1}, a_{t-1})$
Observation function:	$o_t \sim \mathrm{p}(o_t \mid s_t)$
Reward function:	$r_t \sim \mathrm{p}(r_t \mid s_t)$
Policy:	$a_t \sim \mathrm{p}(a_t \mid o_{\leqslant t}, a_{<t}),$

- Goal: Find a policy that maximizes expected rewards
- Uses Model Predictive Control. No explicit policy network.

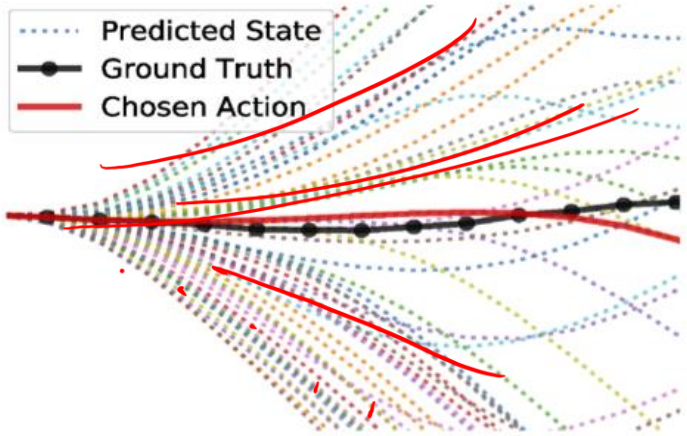
Model-Predictive Control (MPC)

- Use model to plan good looking sequence of actions.
- Take a step
- Update model of transitions
- Repeat



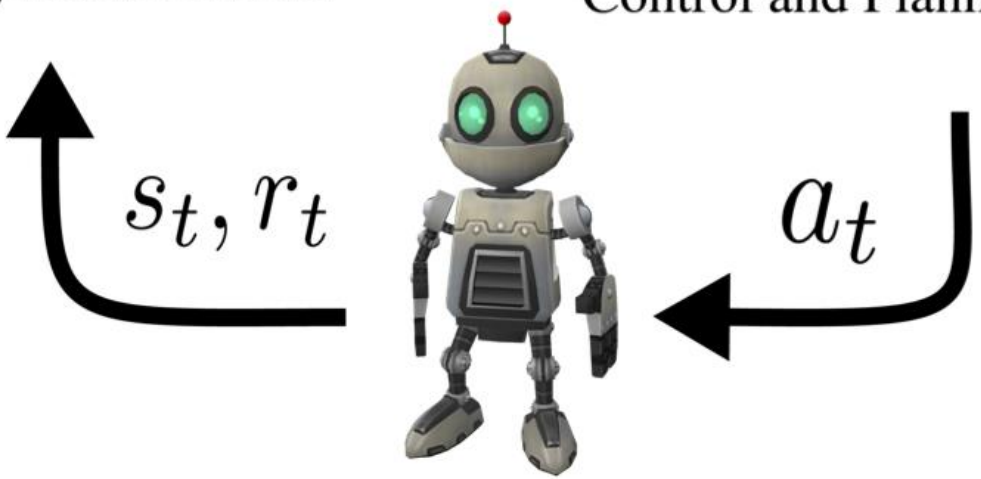
$$s_{t+1} = f_\theta(s_t, a_t)$$

Dynamics Model

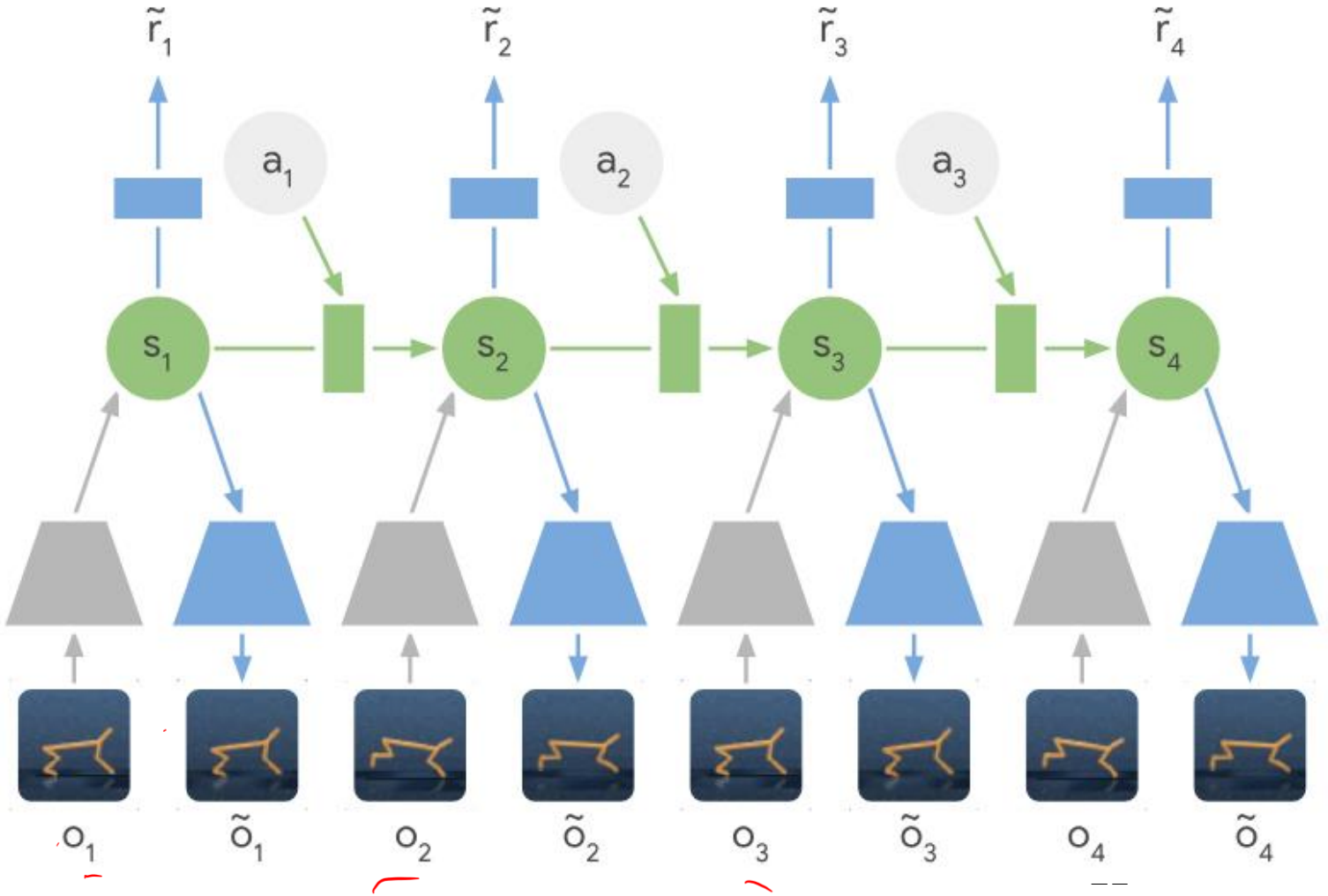


$$\arg\max_{u(t)} \sum_{t=0}^{T} r(s_t, a_t)$$

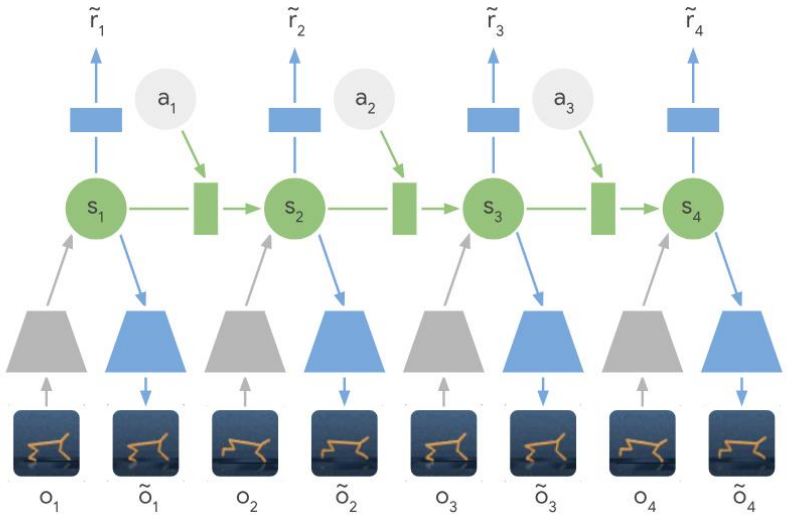
Control and Planning



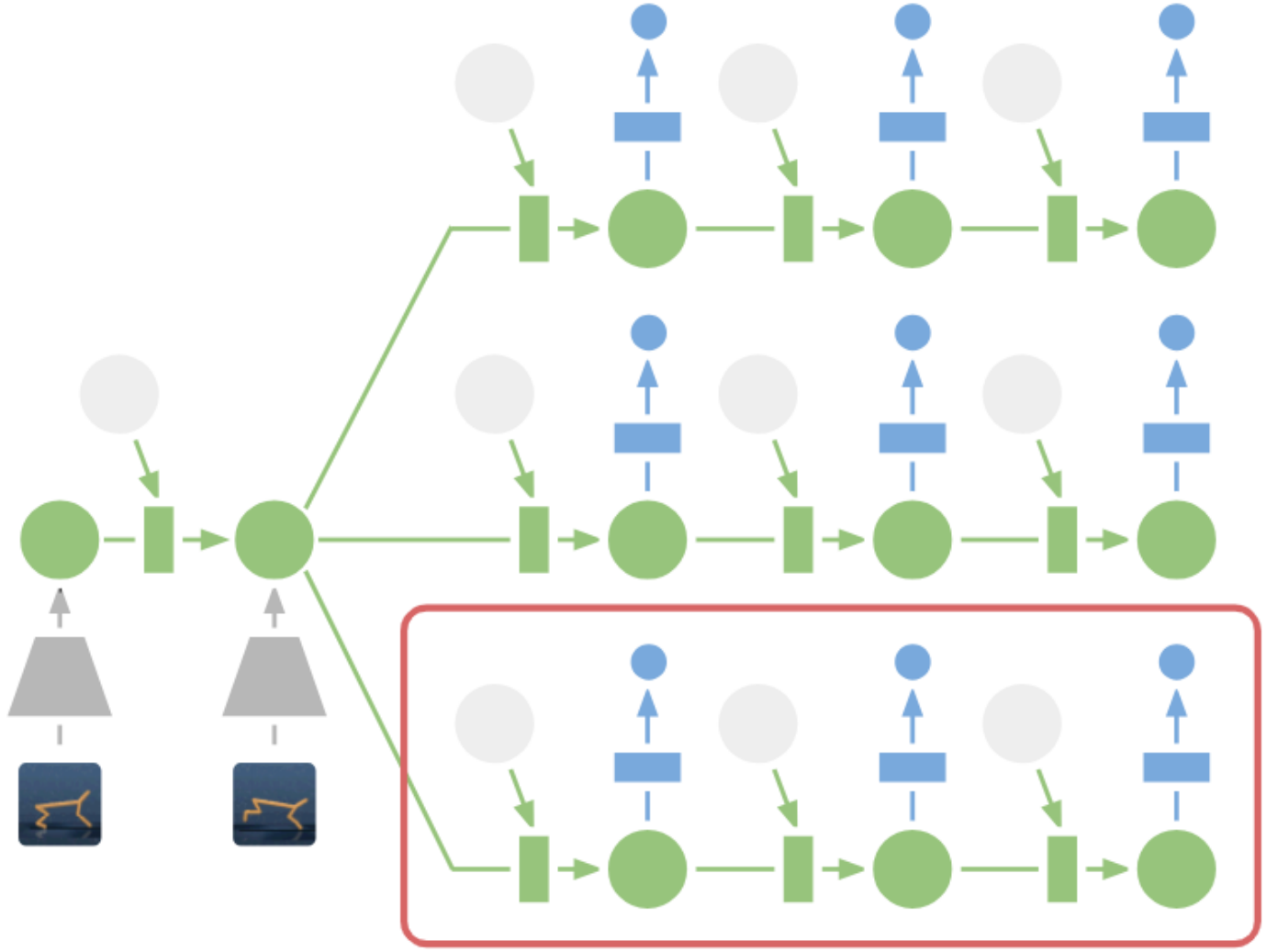
Latent Dynamics Model



Latent Dynamics Model

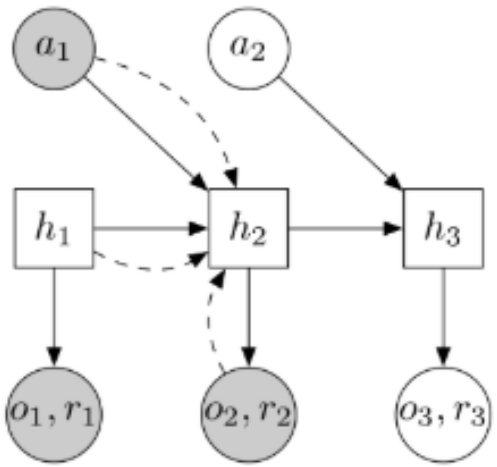


Model-Based Planning via MPC

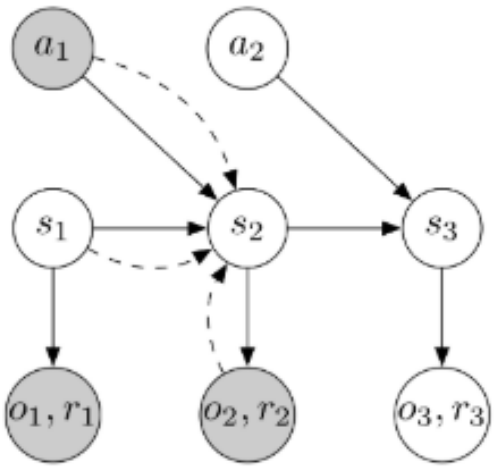


Recurrent State Space Model

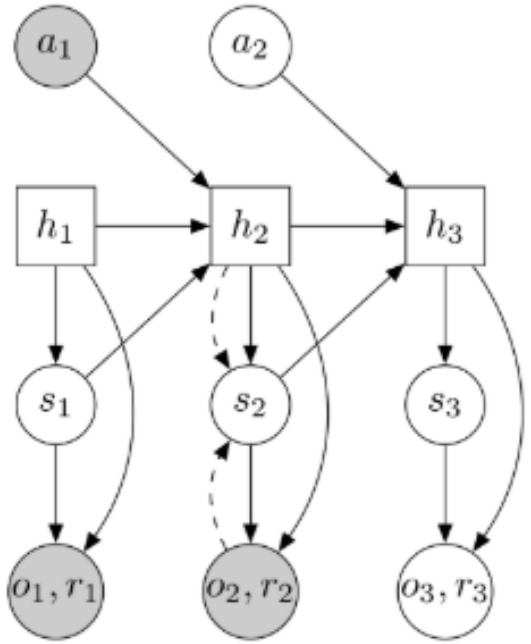
PlaNet



(a) Deterministic model (RNN)

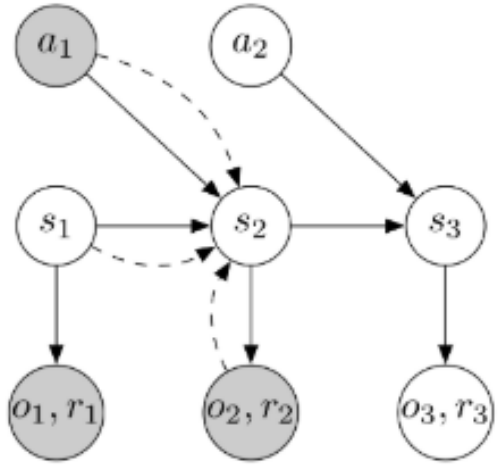


(b) Stochastic model (SSM)



(c) Recurrent state-space model (RSSM)

Stochastic State Space Model

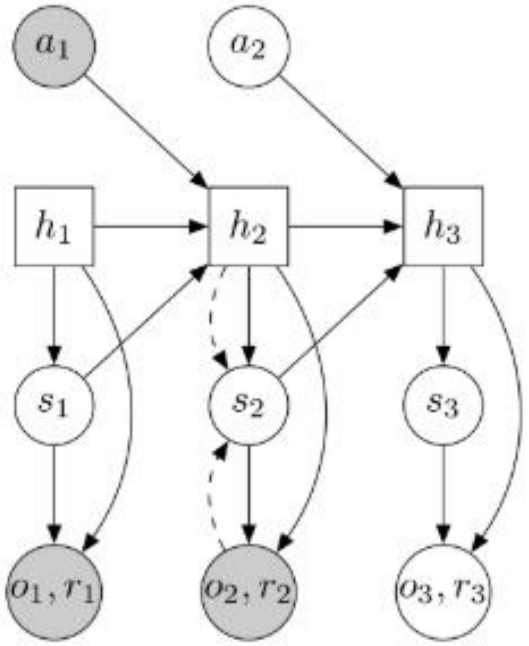


(b) Stochastic model (SSM)

- Learns an encoder $q(s_t \mid o_{<t}, a_{<t})$
- Learns a decoder $p(o_t \mid s_t)$
- Learns a latent transition model $p(s_t \mid s_{t-1}, a_{t-1})$
- Trains similar to a VAE
  - Reconstruction loss $\mathbb{E}_{q(s_t \mid o_{\leq t}, a_{<t})}[\ln p(o_t \mid s_t)]$
  - KL term

$$\mathrm{KL}(q(s_t \mid o_{\leq t}, a_{<t}) \,\|\, p(s_t \mid s_{t-1}, a_{t-1}))$$

Recurrent State Space Models

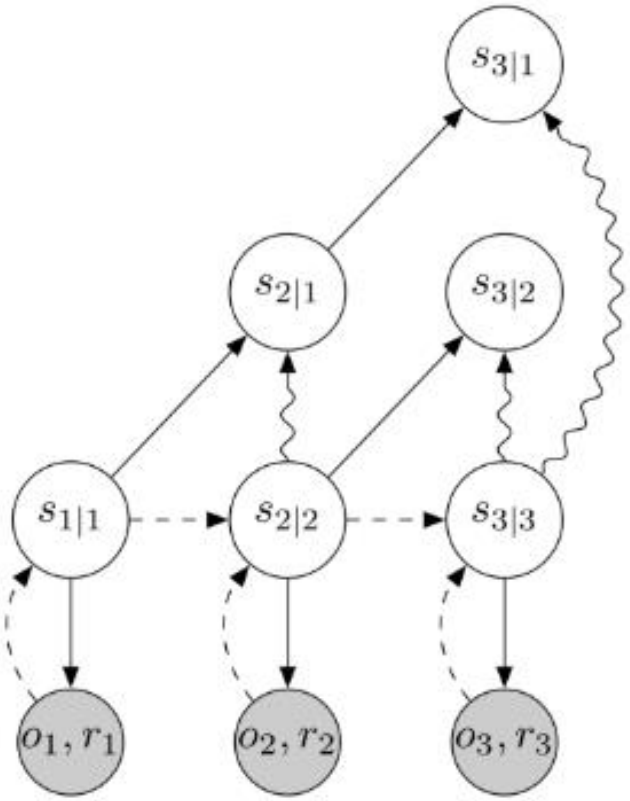


(c) Recurrent state-space model (RSSM)

- Adds a deterministic RNN with hidden state $h_t = f(h_{t-1}, s_{t-1}, a_{t-1})$
- Learns an encoder $q(s_t \mid o_t, h_t)$
- Learns a decoder $p(o_t \mid s_t, h_t)$
- Learns a latent transition model $p(s_t \mid h_t)$
- Trains similar to a VAE
- Reward model trained similarly.

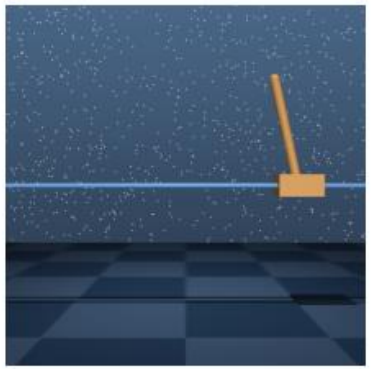
Latent Overshooting

- Errors will compound, but previous math just encourages one-step predictions.
- To fix: Predict multiple states forward in latent space to encourage consistency between one-step and multi-step predictions.



(c) Latent overshooting

Evaluation: Deep Mind Control Suite



(a) Cartpole

(b) Reacher

(c) Cheetah

(d) Finger

(e) Cup

(f) Walker

Very Sample Efficient!

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

Can also train one PlaNet model to be able to solve all tasks!

Dreamer

Published as a conference paper at ICLR 2020

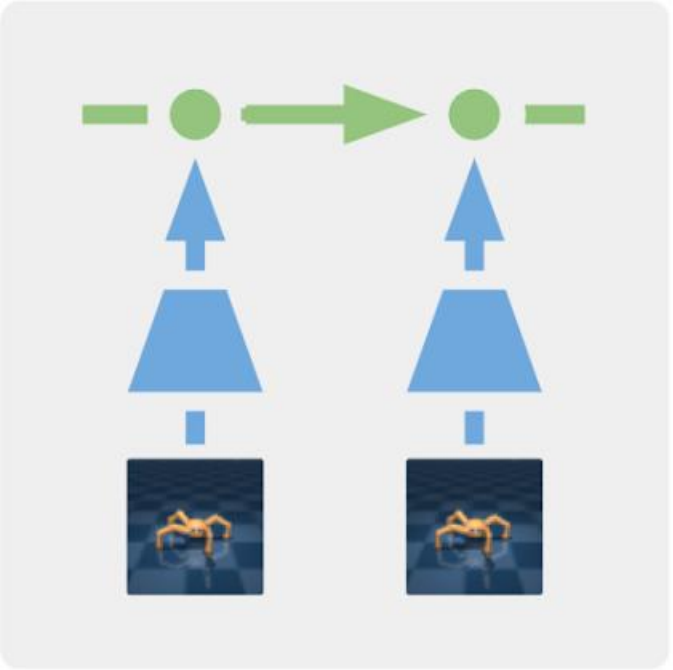
DREAM TO CONTROL: LEARNING BEHAVIORS BY LATENT IMAGINATION

Danijar Hafner *
University of Toronto
Google Brain

Timothy Lillicrap
DeepMind

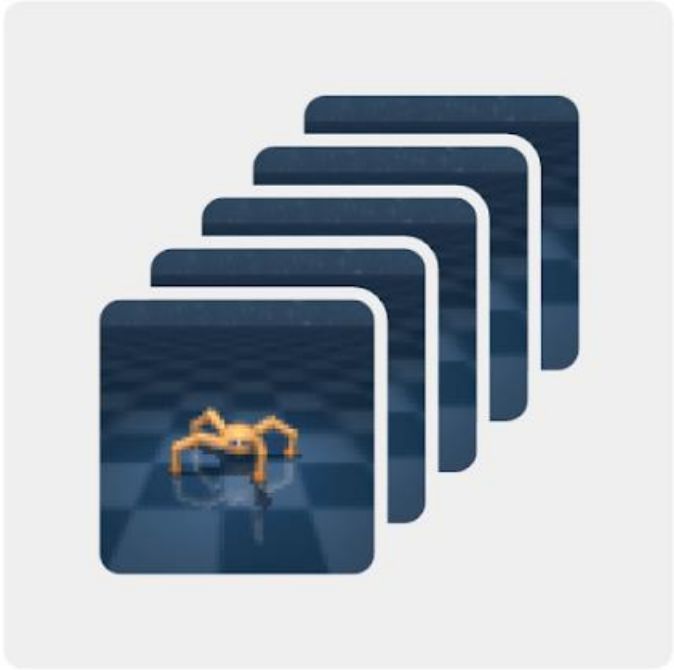
Jimmy Ba
University of Toronto

Mohammad Norouzi
Google Brain



World Model Learning

Learning Value and Actor Networks



Environment Interaction

What makes Dreamer different?

- Actor Critic Architecture
  - Learns value function to optimize Bellman error over imagined rewards
  - Actor gradients are computed through the dynamics