World Models

Reading Group 2020

James Allingham and Gregor Simm

Machine Learning Group Cambridge

18.03.2020

What are World Models about?

- Approach in **Reinforcement Learning** (RL)
- Closely related to **Model-Based RL** (sample efficiency and planning)
- **Partially Observable** Markov Decision Processes (POMDP) (hidden information)

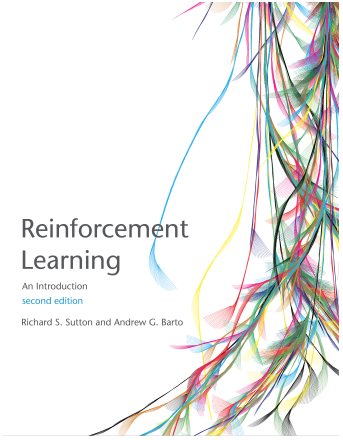
- (Some) Key Papers:
  - David Ha and Jürgen Schmidhuber. "Recurrent world models facilitate policy evolution". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2450–2462
  - Thomas Kipf, Elise van der Pol, and Max Welling. "Contrastive Learning of Structured World Models". In: *arXiv:1911.12247* (2020)
  - Alexander I. Cowen-Rivers and Jason Naradowsky. "Emergent Communication with World Models". In: *arXiv:2002.09604* (2020)
  - ...

Outline

1. Introduction to key concepts
   - Markov Decision Process (MDP)
   - Model-based vs Model-free RL
   - Partially Observable MDPs
2. **Main Paper**: David Ha and Jürgen Schmidhuber. "Recurrent world models facilitate policy evolution". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2450–2462
3. Related Work:
   - Thomas Kipf, Elise van der Pol, and Max Welling. "Contrastive Learning of Structured World Models". In: *arXiv:1911.12247* (2020)
   - Alexander I. Cowen-Rivers and Jason Naradowsky. "Emergent Communication with World Models". In: *arXiv:2002.09604* (2020)
4. Outlook and open questions

Reinforcement Learning – Useful Resources

- Book by Richard S. Sutton and Andrew G. Burto
  - Chapter 1
  - Chapter 2
  - Chapter 3
  - Chapter 8
- UCL Lecture on RL by David Silver (+ videos)
  - Lecture 1
  - Lecture 2
  - Lecture 8



Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6

David Silver. *UCL Course on Reinforcement Learning*. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html. 2015

Characteristics of Reinforcement Learning

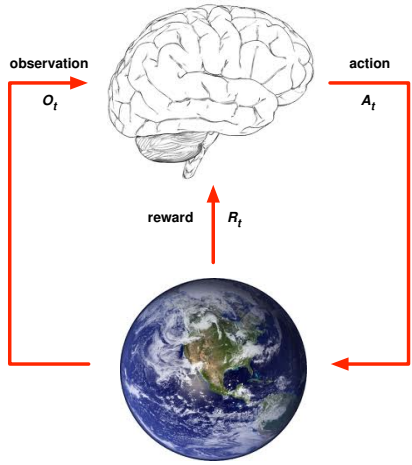
What makes RL different from other machine learning paradigms?

- No supervision – there is only a **reward** signal from an environment
- Feedback is (often) delayed
- Sequential, non i.i.d. data
- Agent's actions affect subsequent data it receives

Agent–Environment Interaction

At each step t the agent:

- gets observation O_t
  (e.g., current frame of Atari)
- sends action A_t to the environment
  (e.g., push button)
- gets scalar reward R_{t+1}
  (e.g., $+1$, 0, -1,...)
- gets next observation O_{t+1}
  (e.g., next frame)



The agent–environment interaction leads to a *trajectory* (assuming $S_t = O_t$):

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

David Silver. *UCL Course on Reinforcement Learning*. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html. 2015

Information State - Markov Property

An **Markov state** contains all "useful" information from the past.

Definition: A state S_t is Markov if and only if

$$\mathbb{P}(S_{t+1} \mid S_t) = \mathbb{P}(S_{t+1} \mid S_1, ..., S_t) \tag{1}$$

- the future is independent of the past given the present.
- the state is a sufficient statistic of the past.

Markov Process

A Markov process is a memoryless random process, i.e., a sequence of random states S_1, S_2,... with the Markov property.

Definition: A Markov Process (or Markov Chain) is a tuple $\langle \mathcal{S}, P \rangle$

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{P}$ is a state transition probability matrix, $\mathbb{P}(S_{t+1} = s' \mid S_t = s)$

Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition:
A Markov Reward Process (MRP) is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathbb{P}(S_{t+1} = s' \mid S_t = s)$
- $\mathcal{R}$ is a reward function, $S \to \mathbb{R}$
- γ is a discount factor, $\gamma \in [0, 1]$

Reward and Return

RL is based on the **Reward Hypothesis**

➔ All goals can be described by the maximisation of expected cumulative reward.

Definition: The return G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

- γ for mathematical convenience, models uncertainty about future
- γ close to 0 leads to "myopic" evaluation
- γ close to 1 leads to "far-sighted" evaluation

Markov Decision Process

A Markov decision process (MDP) is an MRP with decisions. Its a process in which all states are Markov.

Definition:
A Markov Process (or Markov Chain) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function, $S \times A \to \mathbb{R}$
- γ is a discount factor, $\gamma \in [0, 1]$

Agent – Policy

Definition: A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{3}$$

- MDP policies depend on the current state (not the history)
- policies are stationary (time-independent): $A_t \sim \pi(\cdot|S_t), \forall t > 0$
- given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ **and** a policy π:
  - the state sequence $S_1, S_2, ...$ is a Markov process
  - the state and reward sequence $S_1, R_2, S_2, ...$ is a Markov reward process

Value Function

Definition: The *state-value function* $v(s)$ of an MDP is the expected return starting from state s

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \tag{4}$$

Definition: The *action-value function* $q_\pi(s, a)$ of an MDP is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \tag{5}$$

Bellman Expectation Equation

The value function $v_\pi(s)$ can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v_\pi(S_{t+1})$

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] && (6)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] && (7)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] && (8)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] && (9)
\end{aligned}
$$

The action-value function can similarly be decomposed.

Optimal Value Function

Definition: The *optimal state-value function* $v_*(s)$ is the maximum state-value function over all policies

$$v_*(s) = \max_\pi v_\pi(s) \tag{10}$$

Definition: The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{11}$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value function.

Finding an Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \quad \text{if} \quad v_\pi(s) \geq v_{\pi'}(s), \forall s \tag{12}$$

An optimal policy can be found by maximising over $q_*(s, a)$,

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \text{argmax}_{a \in \mathcal{A}} \, q_*(s, a) \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

- If we know $q_*(s, a)$, we immediately have the optimal policy
- No closed form solution (in general) for solving the Bellman equations (for $q(s, a)$, for example)
- Many iterative solution methods exist:
  - Value Iteration
  - Policy Iteration
  - Q-learning
  - Sarsa
  - ...

Model-Free RL Algorithm – Example

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

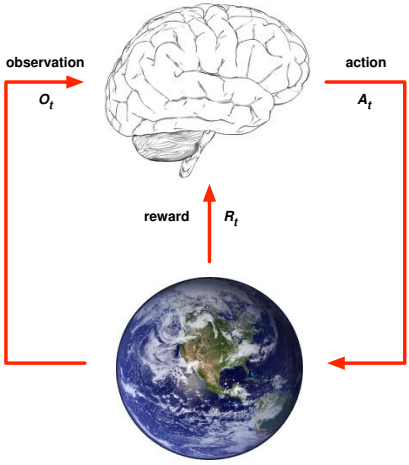
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

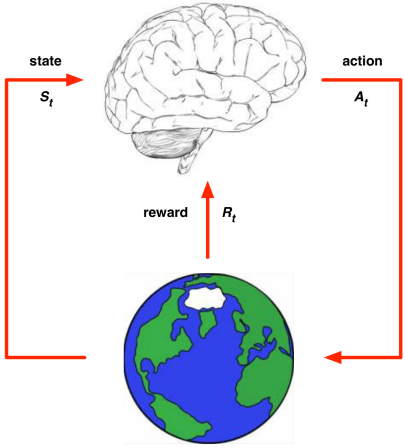
Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Fit a model from experience
  - Plan value function (and/or policy) from model

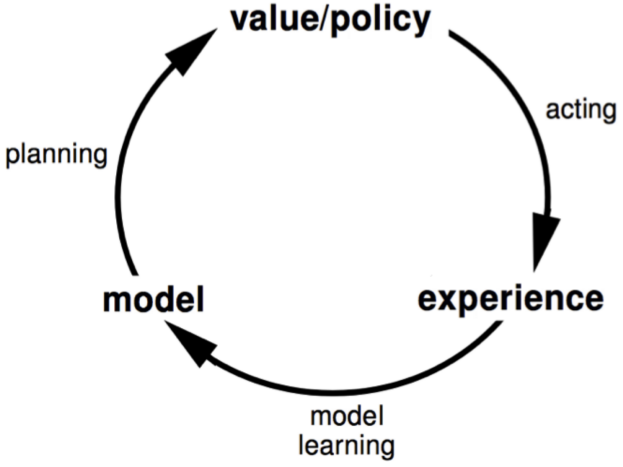
Model-Free vs Model-Based RL





Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6

Model-Based RL



Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6

What is a Model?

- A model $\mathcal{M}_\eta$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ parametrized by η
- Since $\mathcal{S}$ and $\mathcal{A}$ are known often written as $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- **Advantages**:
  - allows planning ✔
  - reason about model uncertainty ✔
  - efficiently learn model by supervised learning methods ✔
- **Disadvantages**:
  - First learn a model, then construct a value function ➔ two sources of error ✘

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$
- This is a supervised learning problem

$$
\begin{aligned}
S_1, A_1 &\to R_2, S_2 \\
S_2, A_2 &\to R_3, S_3 \\
&\dots \\
S_{T-1}, A_{T-1} &\to R_T, S_T
\end{aligned}
$$

- Learning $s, a \to r$ is a regression problem
- Learning $s, a \to s'$ is a density estimation problem
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimizes empirical loss

Application

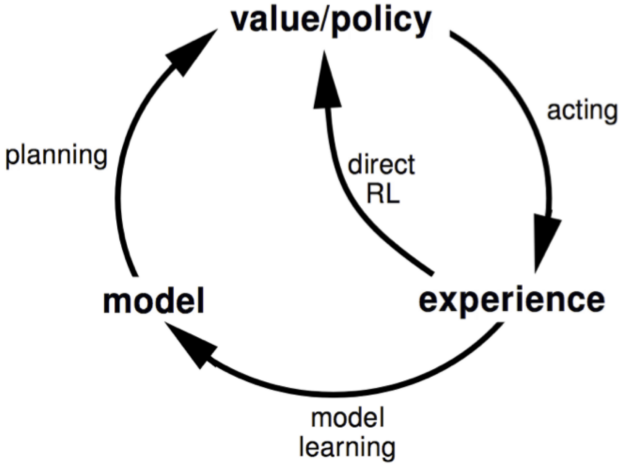
Given $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$:

1. Planning: use favorite planning algorithm (value iteration, tree search,...)
2. Sample-Based Planning:
   - Use model to generate samples, i.e., sample experience from model
   - Apply **model-free** RL to samples (e.g., Q-learning)

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Fit a model from experience
  - Plan value function (and/or policy) from model
- Dyna
  - Fit a model from real experience
  - Learn and plan value function (and/or policy) from real and simulated experience

Model-Based RL



Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6

Model-Based RL Algorithm – Dyna-Q

Tabular Dyna-Q

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

(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S,Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
(e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S,A)$
$Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6

Fully vs. Partially Observable Environments

Full observability: agent directly observes the environment state.

$$O_t = S_t^e \tag{14}$$

- The environment state S_t^e is the environment's private representation.
- Markov Decision Process (MDP) and this is often assumed.

Partial observability: agent indirectly observes environment.

$$O_t \neq S_t^e \tag{15}$$

- Example: a robot with camera vision is not told its absolute position.
- This is a Partially Observable Markov Decision Process (POMDP).

$\Rightarrow$ Agent must construct its own state representation S_t^a

Partially Oberservable Markov Decision Process

A Partially Observable Markov decision process (POMDP) is an MDP with hidden states. It is a hidden Markov model with actions.

Definition: A MOMDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{O}$ is a finite set of observations
- $\mathcal{P}$ is a state transition probability matrix, $\mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$
- $\mathcal{R}$ is a reward function, $\mathcal{R} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- $\mathcal{Z}$ is an observation function, $\mathcal{Z} = \mathbb{P}[O_{t+1} = o \mid S_t = s', A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

History and State

- The **history** is the sequence of *observations*, actions, and rewards:

$$H_t = O_0, A_0, R_1, O_1, A_1, R_2, O_2, A_2, R_3, \ldots \tag{16}$$

- Then, the **state** depends on the history

$$S_t = f(H_t) \tag{17}$$

The function f has to fulfill the Markov property:

$$f(h) = f(h') \Rightarrow \mathsf{Pr}\{O_{t+1} = o | H_t = h, A_t = a\} = \mathsf{Pr}\{O_{t+1} = o | H_t = h', A_t = a\}, \tag{18}$$

History and State (II)

- We want our states to be compact as well as Markov (not simply a concatenation).
- **Idea:** Recursive update that computes S_{t+1} from S_t, A_t and O_{t+1}:

$$S_{t+1} = u(S_t, A_t, O_{t+1}) \tag{19}$$

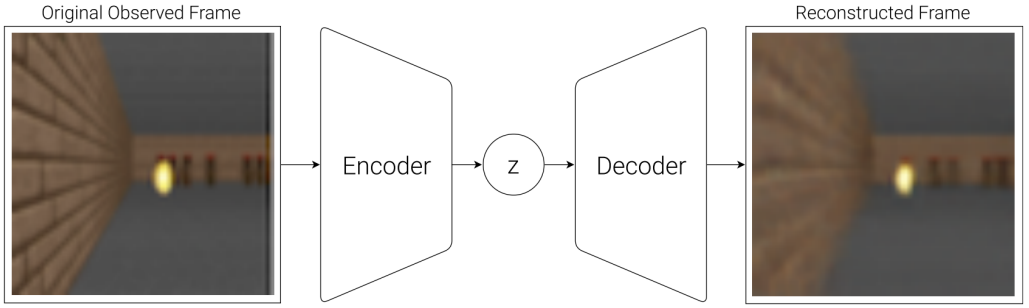
with the first state S_0 given.

World Models [4]

- Inspired by cognitive neuroscience.
- Train a generative model, the **world model**, in an unsupervised manner.
- Train a small controller using evolutionary strategies [6].
- Controller can be trained on **hallucinated** environments.

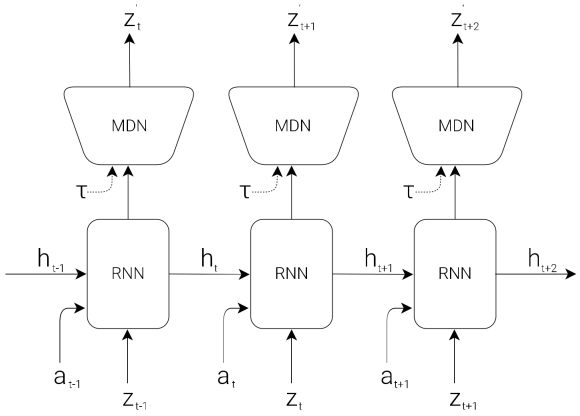
> *One way of understanding the predictive model inside our brains is that it might not simply be about predicting the future in general, but predicting future sensory data given our current motor actions.* [4]

Modeling Space - V(ision) Model



- Train a VAE [7] to reconstruct frames from video game environments.
- Compresses each observation frame into a latent representation z.

Modeling Time - M(emory) Model



- Train an MDN-RNN [3] to predict future latent states.
- Compresses past latent spaces into a hidden state h i.e. learn $P(z_{t+1}|a_t, z_t, h_t)$.
- Can adjust the temperature τ to control model uncertainty.

Controlling the Agent - C Model

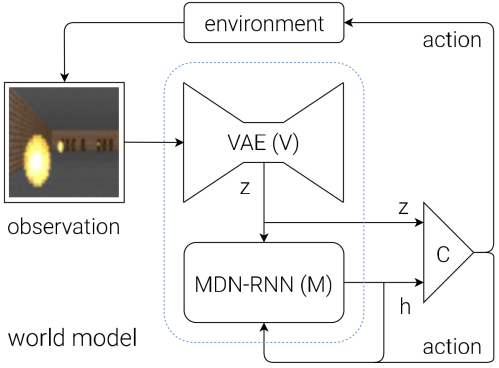
$$a_t = W_c[z_t h_t] + b_c$$

- Intentionally kept as simple as possible.
- Trained to maximise the expected cumulative reward during a rollout.

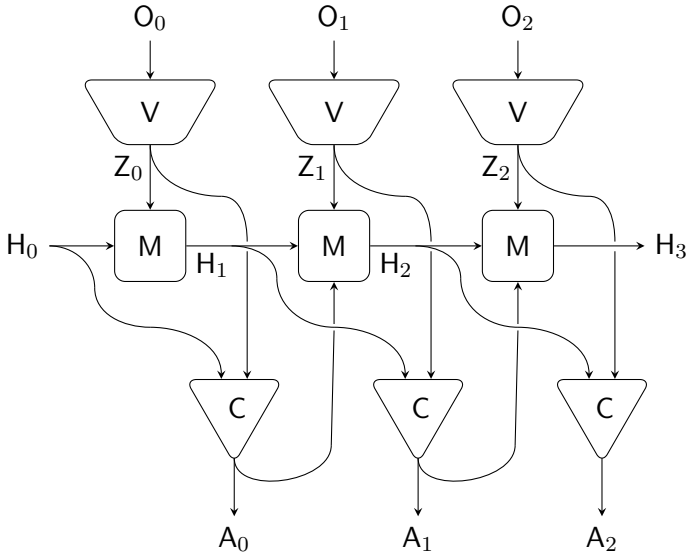
www.tasteofhome.com/recipes/cherry-chocolate-layer-cake/

Putting Everything Together

```
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
```

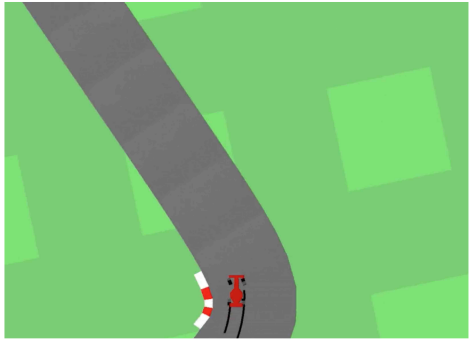


Rolling Out



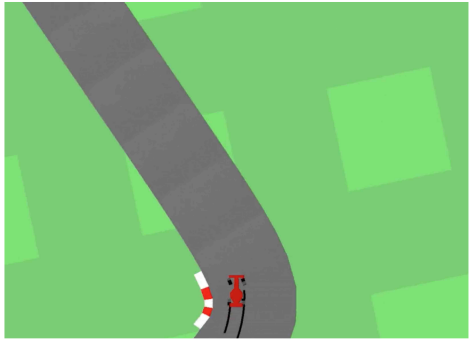
CarRacing-v0

1. Collect 10,000 rollouts from a **random** policy.
2. Train **V** to encode frames into $z \in \mathcal{R}^{32}$.
3. Train **M** to model $P(z_{t+1}|a_t, z_t, h_t)$.
4. Maximise expected reward by optimizing **C** with CMA [6].



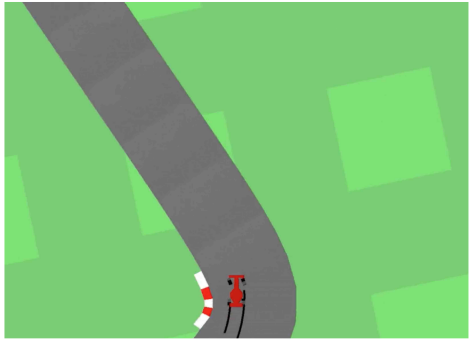
CarRacing-v0

Model	Param Count
V	4,348,547
M	422,368
C	867



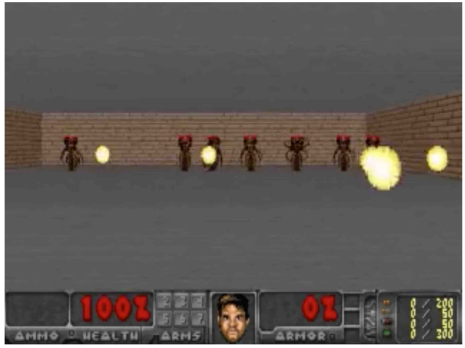
CarRacing-v0

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	$\mathbf{906 \pm 21}$



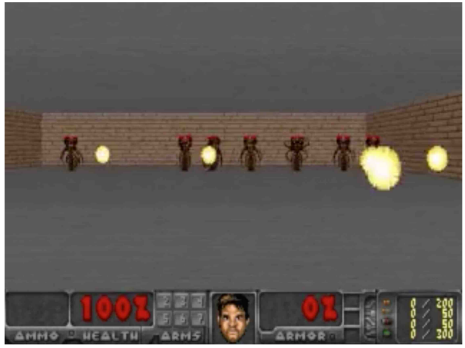
DoomTakeCover-v0

1. Collect 10,000 rollouts from a **random** policy.
2. Train **V** to encode frames into $z \in \mathcal{R}^{64}$. Encode all frames from the rollouts in step 1 into z.
3. Train **M** to model. $P(z_{t+1}, d_{t+1} | a_t, z_t, h_t)$
4. Optimize **C** to maximise expected survival time, in a **hallucinated environment**.
5. Use policy learned by **C** in the real environment.



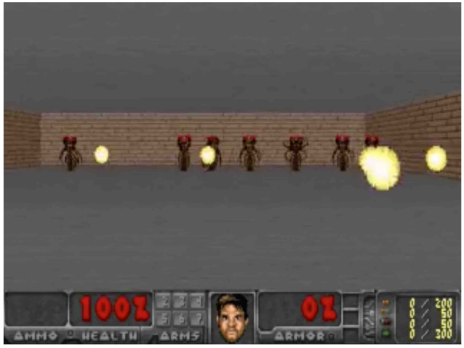
DoomTakeCover-v0

Model	Param Count
V	4,446,915
M	1,678,785
C	1,088



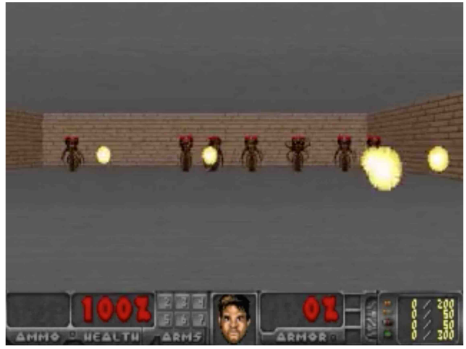
DoomTakeCover-v0

- Agent learns to survive for 900 time steps in the **hallucinated environment**.
- It also learns various rules of the environment such as:
  - How actions (e.g. `left` and `right`) cause the agent to move.
  - That the agent cannot move past walls.
  - How to keep track of projectiles.
  - That the game should end if a projectile hits the agent.



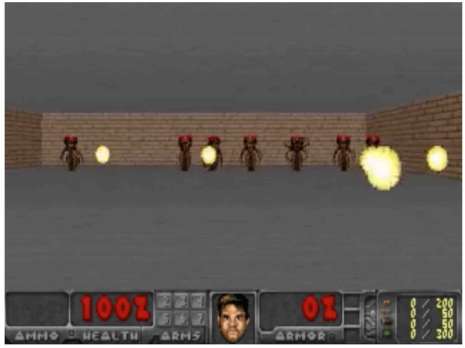
DoomTakeCover-v0

- Agent learns to survive for **1100** time steps in the **real environment**.
- The real environment is more difficult than the hallucinated environment because of it's stochasticity.
- The agent can sometimes learn to cheat in the hallucinated environment.
- The level of difficulty can be controlled with the temperature parameter of the MDN-RNN.



DoomTakeCover-v0

Temperature τ	Virtual Score	Actual Score
0.10	2086 ± 140	193 ± 58
0.50	2060 ± 277	196 ± 50
1.00	1145 ± 690	868 ± 511
1.15	918 ± 546	1092 ± 556
1.30	732 ± 269	753 ± 139
Random Policy	N/A	210 ± 108
Gym Leader	N/A	820 ± 58



CartPole

1. Initialize **M** and **C** randomly.
2. Collect N rollouts in the **real environment**.
3. Train **M** to model $P(x_{t+1}, d_{t+1}, r_{t+1}, a_{t+1}|a_t, x_t, h_t)$.
4. Optimize **C** to maximise expected reward in the **hallucinated environment**.
5. Go back to step 2 if task has not been solved.

Contrastive Learning of Structured World Models [4]

- The world model of Ha and Schmidhuber relies on the VAE framework to learn representations of the environment.
- However, this representation is not structured: it is not explicitly built up from representations of objects, relationships, and hierarchies.
- Additionally, training a generative model on pixels requires a trade-off between reconstruction loss and constraints on the latent variables.
- Contrastively-trained Structured World Models (C-SWMs) attempt to address these issues.

Contrastive Learning of Structured World Models [4]

Suppose our goal is simply to learn a representation of the world:

- Given an experience buffer $\{(s_t, a_t, s_{t+1})\}_{t=1}^{T}$,
- Learn latent representations z_t that discards any information that is not required to predict z_{t+1} given a_t.
- i.e. learn an encoder $E : \mathcal{S} \to \mathcal{Z}$ and a transition model $T : \mathcal{Z} \times \mathcal{A} \to \mathcal{Z}$.

Contrastive Learning of Structured World Models [4]

We can use an energy-based hinge loss function:

$$\mathcal{L} = d(z_t + T(z_t, a_t), z_{t+1}) + \max(0, \gamma - d(\tilde{z}_t, z_{t+1}))$$

where $d(x, y)$ is the squared euclidean distance, $z_t = E(s_t)$, $\tilde{z}_t = E(\tilde{s}_t)$ for $\tilde{s}_t$ sampled at random from the experience buffer, and γ is the margin hyper-parameter.
The hinge is placed only on the corrupted term as this was found to work better in practice.

Contrastive Learning of Structured World Models [4]

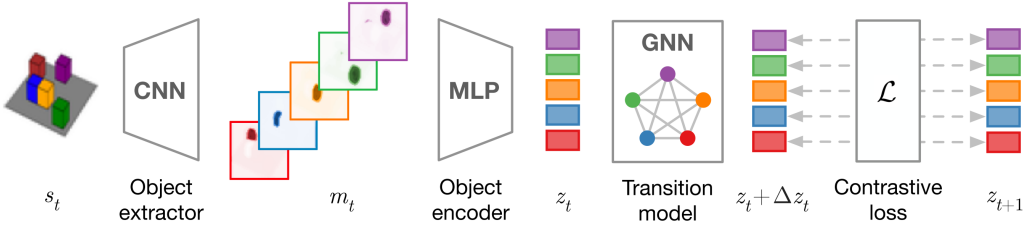
However, we want to learn a **structured** model for the world where:

- $\mathcal{Z} = \mathcal{Z}_1 \times \mathcal{Z}_2 \times \cdots \times \mathcal{Z}_K$, and
- $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_K$.

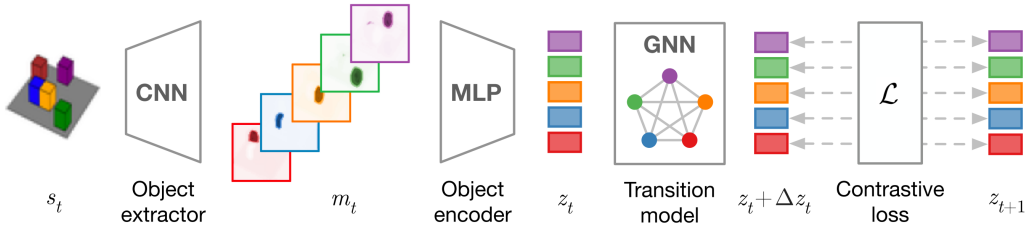
where z_i is the representation for object i in the environment, and a_i is an action applied to it.

- ensures that objects are represented independently,
- allows for parameter sharing, and
- serves as a strong inductive bias!

Contrastive Learning of Structured World Models [4]

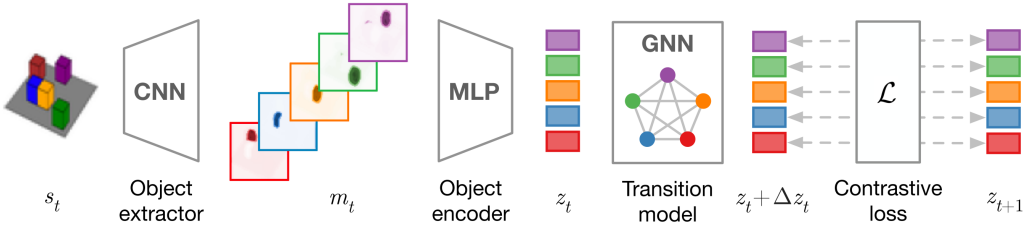


Contrastive Learning of Structured World Models [4]



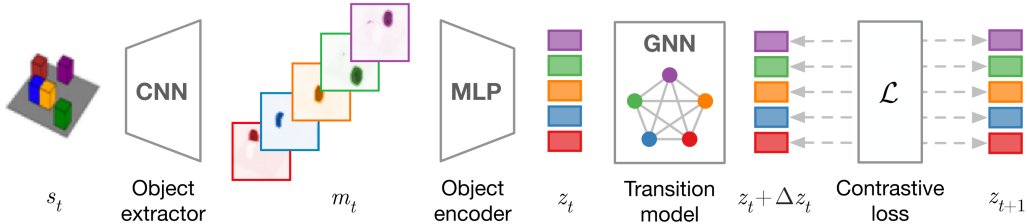
- Encoder split into two parts a CNN E_{ext} and an MLP E_{enc}:
  - $m_t^k = [E_{\text{ext}}(s_t)]_k$
  - $z_t^k = E_{\text{enc}}(m_t^k)$.
- Only one feature map per object in this work.

Contrastive Learning of Structured World Models [4]



- $\Delta z = T(z_t, a_t) = \text{GNN}(\{(z_t^k, a_t^k)\}_{k=1}^K)$.
- GNN consists of node and edge update functions implemented as MLPs:
  - $e_t^{(i,j)} = f_{\text{edge}}([z_t^i, z_t^j])$
  - $\Delta z_t^j = f_{\text{node}}([z_t^j, a_t^j, \sum_{i \neq j} e_t^{(i,j)}])$
- This work only applies 1 round of message passing.

Contrastive Learning of Structured World Models [4]



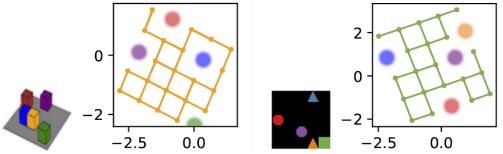
$$\mathcal{L} = H + \max(0, \gamma - \tilde{H})$$

where:

$$H = \frac{1}{K} \sum_{k=1}^{K} d(z_t^k + T^k(z_t, a_t), z_{t+1}^k) \quad \tilde{H} = \frac{1}{K} \sum_{k=1}^{K} d(\tilde{z_t^k}, z_{t+1}^k))$$

Contrastive Learning of Structured World Models [4]

(a) Discovered object masks in a scene from the 3D cubes (top) and 2D shapes (bottom) environments.



(b) Learned abstract state transition graph of the yellow cube (left) and the green square (right), while keeping all other object positions fixed at test time.

- Quantitative results for these environments and some Atari games:
  - Stronger performance than VAE-based world models.
  - Ablation study for factored states, GNN, and contrastive loss.
- Limitations:
  - Instance disambiguation.
  - Stochasticity.
  - Markov assumption.

Emergent Communication with World Models [1]

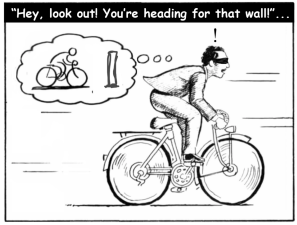


Figure 1: A *Language-Conditional* World Model, adapted from Scott McCloud's *Understanding Comics* [10] and World Models [11]. Here the cyclist has limited observability of the world around him (blindfolded), and conceptualizes danger by interpreting language within the context of his world model.

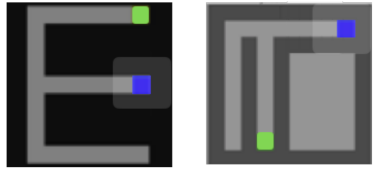


Figure 2: The partially-observable worlds that the agents interact in. The speaker (unseen) is able to view the entire map, whereas the listener (blue) only views a pixel in each direction. At the start of each game, a flag (green) is randomly placed in one of two paths. Both the speaker and listener receive a reward if the listener is able to find the flag by choosing the correct corridor.

EC Setting

- Speaker:
  - Full observation O_t
  - Sends messages m_t based on O_t to listener
- Listener:
  - Has partial observation o_t
  - Receives messages m_t from speaker
  - Acts in environment through actions a_t
- Both try to achieve the same goal

Challenges and Desiderata

- m_t should not be a command
- separate m_t from listeners decision making
- message should be grounded: should relate to what the speaker is seeing
- listener should update belief of state s_t based on m_t
- listener should "visualize" the world (dreaming)

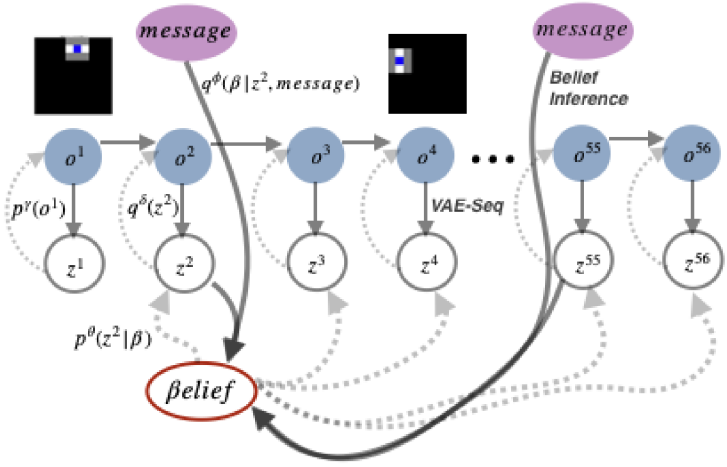
Language World Models (LWM): WM for partially observable worlds which are trained to predict future states based on messages.

Speaker – Concept Clustering

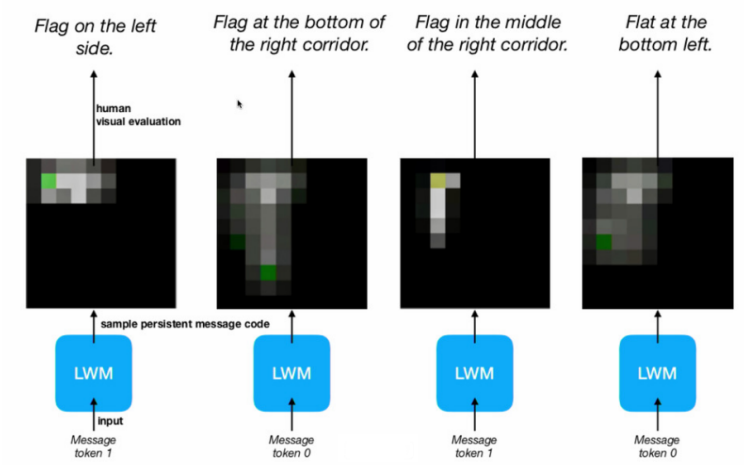
Algorithm 2 Concept Clustering

procedure CC(o) ▷ Algorithm for calculating Concept Clustering
  $\overline{p}_{soft} \leftarrow 0$ ▷ pass any decoder function $f^{o|m}$.
  $MSE \leftarrow 0$ ▷ pass the senders message network excl. softmax layer π_m
  for o in $BatchObs$ **do**
    $p_{soft} = Softmax^{\tau}(\pi_m(o) + \epsilon),\ \epsilon \sim \mathcal{G}umbel(0, 1)$
    $m_t = OneHot(argmax(p_{soft})) - StopGradient(p_{soft}) + p_{soft}$
    $\hat{o} = f^{o|m}(m_t)$ ▷ Decode message into an observation
    $\overline{p}_{soft}\ \mathrel{+}= \frac{p_{soft}}{\text{BatchSize}}$
    $MSE\ \mathrel{+}= \sum \frac{(\hat{o}-o)^2}{\text{BatchSize}*\#\text{Pixels}}$ ▷ Where the max error can be 1.0
  end for
  return $CC = \sum \overline{p}_{soft} \log(\overline{p}_{soft}) + MSE$
end procedure ▷ The entropy term aids $\mathcal{GS}$ sample exploration

Listener


message
message
$q^{\phi}(\beta \mid z^2, message)$
Belief
Inference
o^1
o^2
o^3
o^4
o^{55}
o^{56}
$p^{\gamma}(o^1)$
$q^{\delta}(z^2)$
VAE-Seq
z^1
z^2
z^3
z^4
z^{55}
z^{56}
$p^{\theta}(z^2 \mid \beta)$
$\beta elief$

Interpret Message by Inspecting Listener's Belief



Conclusions

- Key Paper: World Models
- Extensions:
  - Structured World Models: WM which learn structured representations of the environment.
  - Language World Models: WM which predict the future based on messages from a speaker.
- More reading:
  - Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels". In: *arXiv preprint arXiv:1811.04551* (2018) – shows that learning in hallucinated environments can be very sample efficient.
  - Daniel Freeman, David Ha, and Luke Metz. "Learning to Predict Without Looking Ahead: World Models Without Forward Prediction". In: *Advances in Neural Information Processing Systems*. 2019, pp. 5380–5391 – another application of World Models to partially observed environments. Very cool web version of the paper: `https://learningtopredict.github.io/`.
  - Marwin H. S. Segler. "World Programs for Model-Based Learning and Planning in Compositional State and Action Spaces". In: *arXiv:1912.13007* (2019) – World Programs: agent has to learn the allowed actions as well.

Questions?

Thank you for your attention!

References

[1] Alexander I. Cowen-Rivers and Jason Naradowsky. "Emergent Communication with World Models". In: *arXiv:2002.09604* (2020).

[2] Daniel Freeman, David Ha, and Luke Metz. "Learning to Predict Without Looking Ahead: World Models Without Forward Prediction". In: *Advances in Neural Information Processing Systems*. 2019, pp. 5380–5391.

[3] David Ha and Douglas Eck. "A neural representation of sketch drawings". In: *arXiv preprint arXiv:1704.03477* (2017).

[4] David Ha and Jürgen Schmidhuber. "Recurrent world models facilitate policy evolution". In: *Advances in Neural Information Processing Systems*. 2018, pp. 2450–2462.

[5] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels". In: *arXiv preprint arXiv:1811.04551* (2018).

[6] Nikolaus Hansen. "The CMA evolution strategy: A tutorial". In: *arXiv preprint arXiv:1604.00772* (2016).

[7] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[8] Thomas Kipf, Elise van der Pol, and Max Welling. "Contrastive Learning of Structured World Models". In: *arXiv:1911.12247* (2020).

[9] Marwin H. S. Segler. "World Programs for Model-Based Learning and Planning in Compositional State and Action Spaces". In: *arXiv:1912.13007* (2019).

[10] David Silver. *UCL Course on Reinforcement Learning*. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html. 2015.

[11] Richard S. Sutton, Andrew G. Barto, and Francis Bach. *Reinforcement Learning: An Introduction*. second edition. Cambridge, Massachusetts: MIT Press, 2018. ISBN: 978-0-262-03924-6.