SCHOOL *of* DATA SCIENCE

# Model-Based RL

Reinforcement Learning

School of Data Science

University of Virginia

Last updated: July 21, 2025

# Agenda

> Model-Free RL

> Model-Based RL

> World Models

# Model Free vs. Model-Based

So far we have looked at model-free approaches

There was no transition model $\boxed{P(s_{t+1}|s_t, a_t)}$

Instead, we *sampled* next state by running action in environment:

$$\boxed{(s_t, a_t) \xrightarrow{\text{environment}} s_{t+1}, r_{t+1}}$$

# Reminder about Model-Free RL

In some cases, we devised toy rules

In other cases, we ran a simulator

# Reminder about Model-Free RL

In some cases, we devised toy rules

In other cases, we ran a simulator

Of course, a simulator might use a model

But the RL agent doesn't know or learn the model

# Disadvantages of Model-Free RL

> **Lower sample efficiency**: without a model, agent only learns from experience

# Disadvantages of Model-Free RL

> **Lower sample efficiency**: without a model, agent only learns from experience

> **Can't plan ahead**: it is not possible to simulate rollouts and learn from them

# Disadvantages of Model-Free RL

> **Lower sample efficiency**: without a model, agent only learns from experience

> **Can’t plan ahead**: it is not possible to simulate rollouts and learn from them

> **Adaptability is challenging**: if environment / reward function changes, a lot of experience is required for learning

# Disadvantages of Model-Free RL

> **Lower sample efficiency**: without a model, agent only learns from experience

> **Can't plan ahead**: it is not possible to simulate rollouts and learn from them

> **Adaptability is challenging**: if environment / reward function changes, a lot of experience is required for learning

> **Lack of interpretability**: value functions and policies can be black boxes

# Model-Based RL – General Strategy

Two parts:

1. Learn a **dynamics function** to model observed state transitions $\boxed{P(s_{t+1}|s_t, a_t)}$

2. Use model predictions to learn what **actions** to take (e.g., learn a policy)

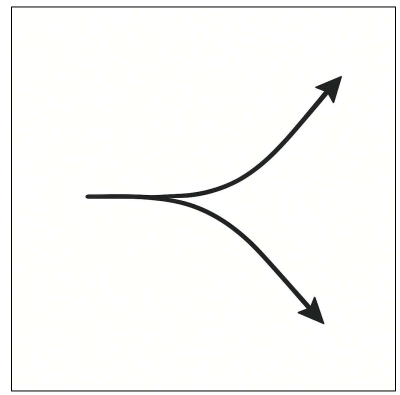
# Action Selection and Horizon

We can use model predictions to learn what **actions** to take

For example, our usual strategy is to maximize return (= expected total discounted reward)

**Q: What horizon to use?**

Infinite horizon won't work … prediction errors can compound



Instead, we use some planning horizon $H$

# Action Selection with Finite Planning Horizon

1. Set planning horizon *H*

2. Generate *K* random action sequences each with length *H, denoted*

$$\mathbf{A}^{(k)} = (a_t^{(k)}, \ldots, a_{t+H-1}^{(k)})$$

3. Use the dynamics model $f_\theta$ to predict the future states after taking each action sequence

4. Evaluate the return associated with each candidate action sequence

5. Select the best action sequence

This method is called *random shooting*

# Refinement to Action Selection: Replanning

Since our model is imperfect, we might have compounding errors as we plan into the future

We can adopt a *model predictive control* (MPC) approach:

1. At each time step, we perform random shooting or something else

2. Select the best $H$-step action sequence

3. Only take the first action from the sequence

4. Now replan at the next time step using updated state information

# Refinement to Dynamics Model: Ensemble

We are using some model (e.g., a neural network) $f_\theta$ to predict the next state

A method for **potentially improving predictions** is to use a set of models $\{f_{\theta_n}\}_{n=1}^{N}$

These can be independently initialized

**At inference time:** For each candidate action sequence, generate *N* independent rollouts

Average the rewards of the rollouts to select the best sequence

# Model-Based RL

Q: Given model-free RL limitations, why don't we always use a model?

# Model-Based RL

Q: Given model-free RL limitations, why don’t we always use a model?

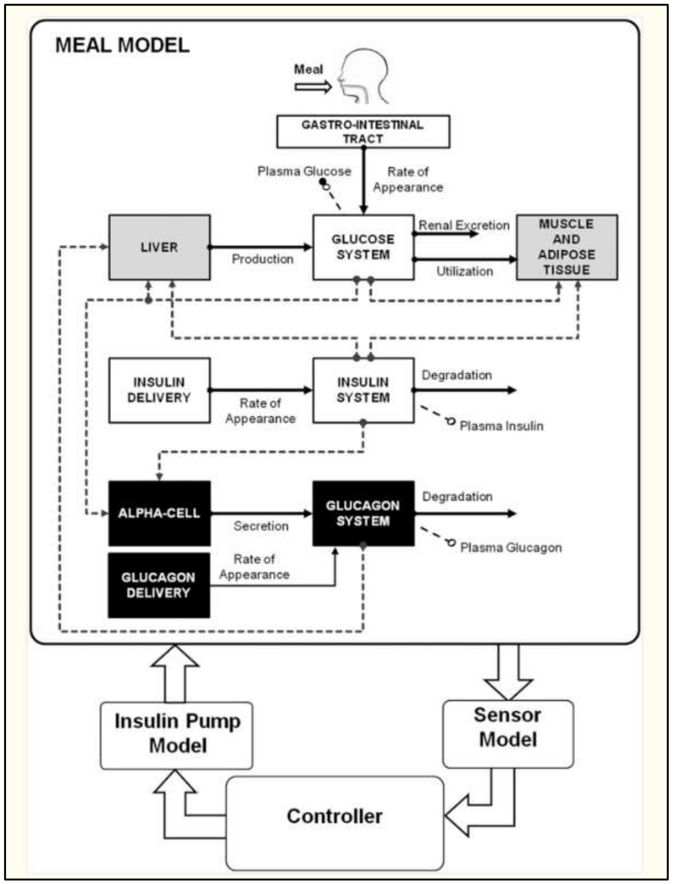
**A: Because developing an accurate model can be hard.**

$$x_1[k+1] = \theta_1 u[k-1] + \theta_2 u[k] + \theta_3 u[k+1] + \theta_4 x_1[k-1] + \theta_5 x_1[k] + \theta_6 x_2[k] + \theta_0$$

Individualization of pharmacological anemia management using reinforcement learning☆

Adam E. Gaweda[a,*], Mehmet K. Muezzinoglu[b], George R. Aronoff[a], Alfred A. Jacobs[a], Jacek M. Zurada[b], Michael E. Brier[a,c]

# Model-Based RL: Diabetes Simulator




J Diabetes Sci Technol. 2014 Jan;8(1):26–34. doi: 10.1177/1932296813514502

**The UVA/PADOVA Type 1 Diabetes Simulator**

New Features

Chiara Dalla Man [1], Francesco Micheletto [1], Dayu Lv [2], Marc Breton [2], Boris Kovatchev [2], Claudio Cobelli [1,✉]

The model of glucose kinetics is described by,

$$\begin{cases} \dot{G}_p = EGP - U_{ii} - k_1 \cdot G_p(t) + k_2 \cdot G_t(t) & G_p(0) = G_{pb} \\ \dot{G}_t = -U_{id}(t) + k_1 \cdot G_p(t) - k_2 \cdot G_t(t) & G_t(0) = G_{pb}\frac{k_1}{k_2} \end{cases}$$

# World Models

# World Models

A *world model* (David Ha, Jürgen Schmidhuber, 2018) is a learned model of the dynamics:

$$\boxed{(s_t, a_t) \rightarrow s_{t+1}, r_{t+1}}$$

Useful for:

> Predicting next state

> Planning

> Imagining possible futures without taking the actions in real environment.
**Critical when it would be dangerous or costly to try the actions in real world.**

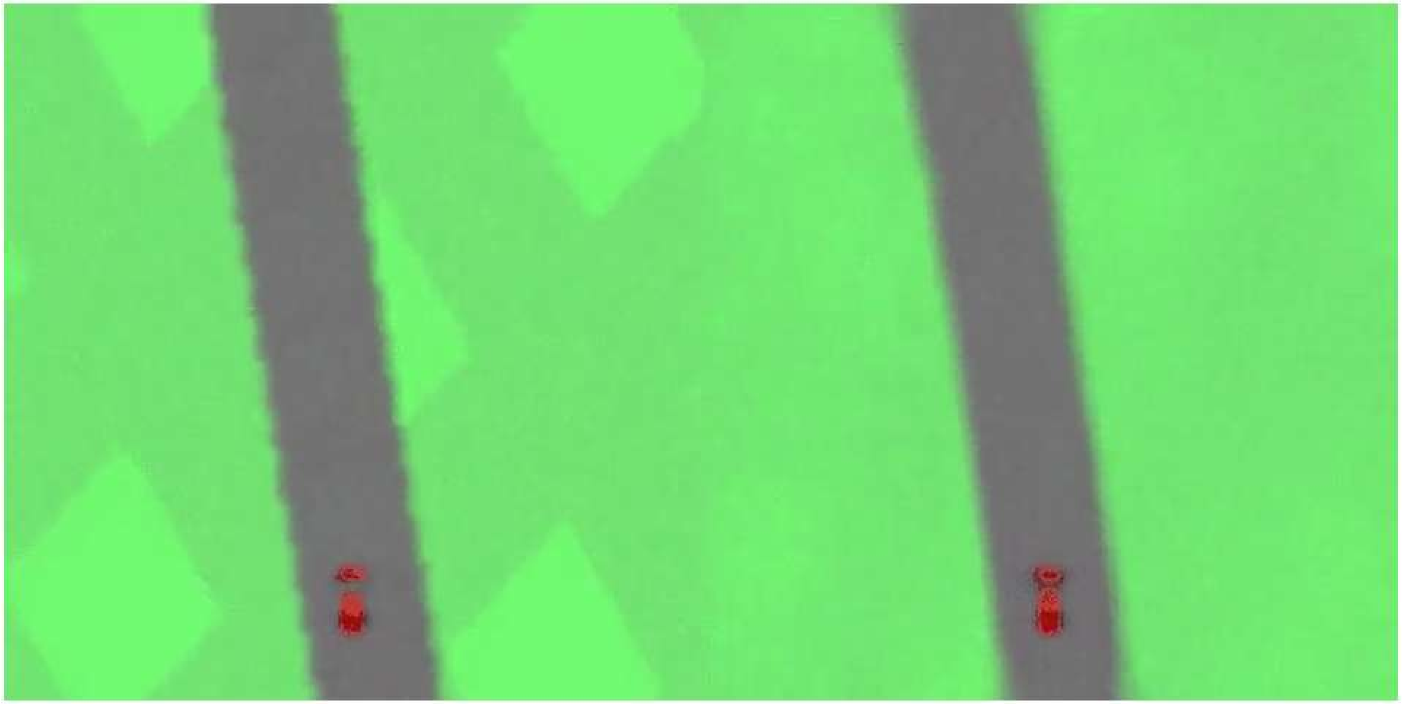
**worldmodels.github.io**

# World Models – Classic Model

From Ha & Schmidhuber paper, architecture has three parts:

| | |
|---|---|
| **VAE (encoder)** | Compress high-dim obs (e.g., images) into a low-dim latent space |
| **MDN-RNN** | Learn to predict next latent state, given current latent state, action |
| **Controller** | “Small” policy network that decides actions in latent space |

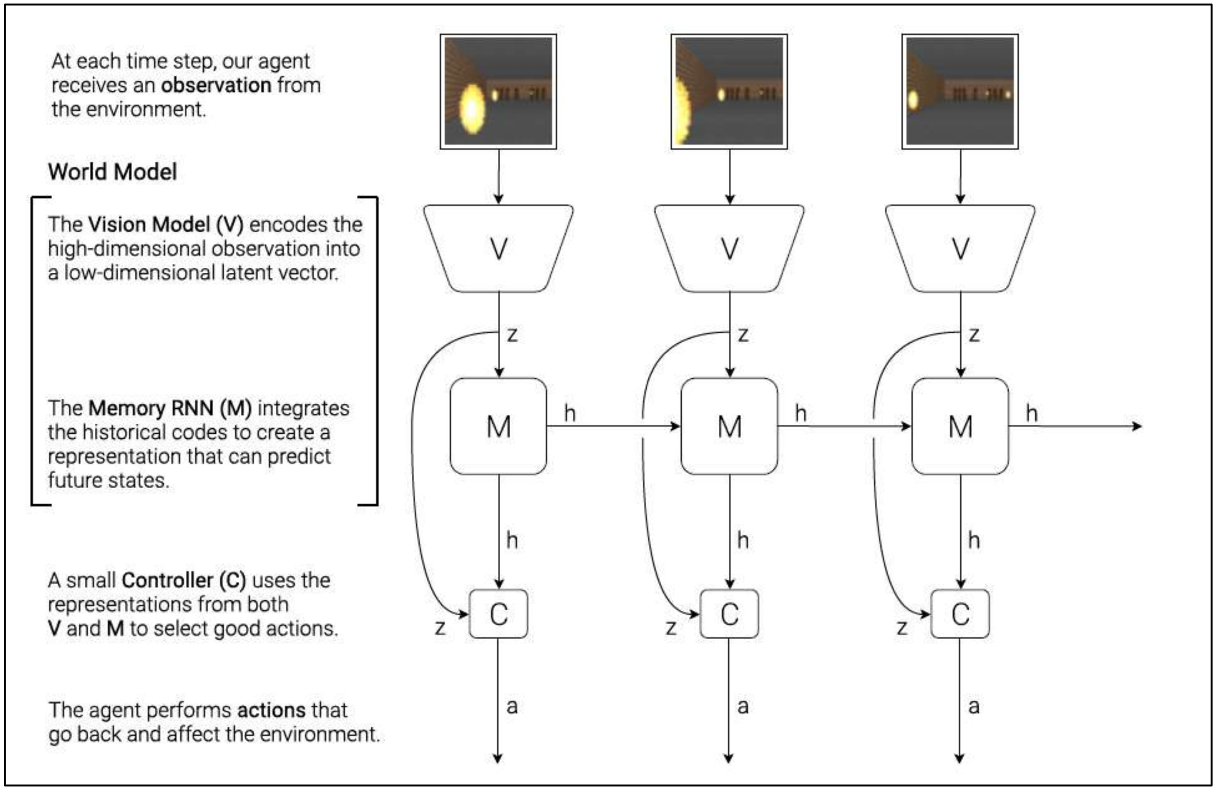
# Car Racing Experiment - Image Encoding



Actual observations from the environment. What gets encoded into $z_t$.

# World Model with Images

David Ha, Jürgen Schmidhuber, 2018



# MDN-RNN

Predicts next latent state as density function $p(z)$

Approximated as mixture of Gaussians

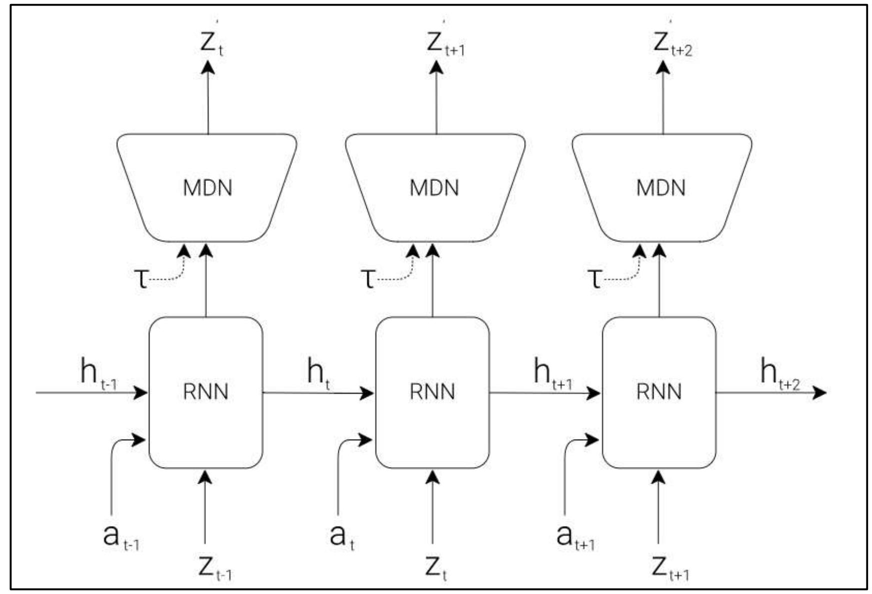
RNN used to model $P(z_{t+1} \mid a_t, z_t, h_t)$

where $a_t$ denotes the action

$h_t$ is the hidden state

$\tau$ is the *temperature* for controlling uncertainty

MDN component outputs parameters of mixture distn.



# Car Racing Experiment - Procedure

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.

# Learning in a Dream

We have seen the procedure for training a simple policy to solve tasks

Can train the agent insides its “dream” environment

Then transfer policy back to actual environment

---

Explore the paper and interactive demo:

**https://worldmodels.github.io/**