ESE 6510 Notes: Model-Based Reinforcement Learning

Vaibhav Thakkar

Contents

Motivation: Why Learn a Model?

In model-free RL (Q-learning, policy gradients), we interact with the environment and learn either a value function or a policy directly. But what if we could also learn *how the environment works* and use that knowledge to plan more efficiently?

Recall the optimal control objective. When dynamics $f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$ are known, we can optimize actions by backpropagating gradients through the system:

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \quad \text{s.t.} \quad \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

This requires $\frac{df}{d\mathbf{x}_t}$ and $\frac{df}{d\mathbf{u}_t}$ — the *Jacobians of the dynamics*. If the dynamics are unknown, we cannot compute these.

Model-based RL addresses exactly this: learn $f(\mathbf{s}, \mathbf{a})$ from data, and then plan through it.

Core Idea

Model-based RL: Learn a dynamics model $f(\mathbf{s}, \mathbf{a}) \approx \mathbf{s}'$ from collected data, then use the learned model for planning or policy optimization — replacing the need for true environment dynamics.

The Distribution Mismatch Problem

Version 0.5: Naïve Model Learning

The simplest approach is:

1. Run a base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g. random) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. Learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. Plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

When does this work? This is essentially how *system identification* works in classical robotics. It is effective when the model class is well-specified (e.g. hand-engineered physics representations with few parameters to fit).

When does it fail? The fundamental problem is *distribution mismatch*:

$$p_{\pi_0}(\mathbf{s}_t) \neq p_{\pi_f}(\mathbf{s}_t)$$

The model is trained on states visited by π_0, but evaluated on states visited by the final policy π_f. As more expressive model classes (e.g. neural networks) are used, the model can make confidently wrong predictions in unvisited regions — leading the planner to exploit these model errors.

Version 1.0: Iterative Data Collection

To fix the distribution mismatch, we iterate:

1. Collect data with π_0: $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. Learn $f(\mathbf{s}, \mathbf{a})$ from $\mathcal{D}$
3. Plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. **Execute those actions and add the resulting data to $\mathcal{D}$**
5. Repeat from step 2

This ensures that the model is gradually trained on states that the planning policy visits.

Version 1.5: Model Predictive Control (MPC)

Even with iterative collection, open-loop planning over a long horizon can go wrong due to compounding model errors. The key insight: **replanning helps with model errors**.

Model-Based RL Version 1.5 (MPC):

1. Run base policy π_0 to collect $\mathcal{D}$
2. Learn dynamics model $f(\mathbf{s}, \mathbf{a})$
3. Plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. **Execute only the first planned action**, observe resulting state $\mathbf{s}'$
5. Append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$
6. Return to step 2 (every N steps)

Why does MPC help? The more frequently you replan, the less perfect each individual plan needs to be. With short horizons, even random sampling over action sequences can work well. Small model errors get corrected at each replanning step before they compound.

Version 2.0: Backpropagate into the Policy

Rather than re-running optimization at test time. To be effective, this requires dense rewards, or, in the case of sparse rewards, high precision value functions. In addition, the test time optimization can potentially be very computationally expensive. To amortize this cost, we can train a neural network policy π_θ to amortize the planning:

$$\mathbf{a}_t = \pi_\theta(\mathbf{s}_t), \quad \mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

Since both π_θ and f are differentiable, we can backpropagate the reward signal through time:

1. Run base policy π_0 to collect $\mathcal{D}$
2. Learn dynamics model $f(\mathbf{s}, \mathbf{a})$
3. **Backpropagate through $f(\mathbf{s}, \mathbf{a})$ into the policy** to optimize $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. Run π_θ, append visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

This is computationally cheap at runtime (just evaluate the policy), but can be numerically unstable, especially in stochastic domains where the reparameterization trick or moment matching is needed.

Summary of MBRL Versions

Version	Pro	Con
0.5	Simple, no iteration	Distribution mismatch
1.0	Fixes mismatch	Open-loop plan is fragile
1.5 (MPC)	Robust to model errors	Computationally expensive
2.0 (Policy BP)	Cheap at runtime	Numerically unstable

What Kind of Models Can We Use?

The dynamics model takes $(\mathbf{s}, \mathbf{a})$ as input and predicts $\mathbf{s}'$. Several choices exist:

Gaussian Processes (GPs). Model $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ as a GP. Very data-efficient and provides uncertainty estimates, but scales poorly (cubic in dataset size) and struggles with non-smooth dynamics.

Neural Networks. Input $(\mathbf{s}, \mathbf{a})$, output $\mathbf{s}'$. The Euclidean training loss corresponds to a Gaussian $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. More complex losses (e.g. mixture of Gaussians output heads) capture multimodal uncertainty. Very expressive and scales to large datasets, but less data-efficient.

Gaussian Mixture Models (GMMs). Train a GMM over $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ tuples. Condition on $(\mathbf{s}, \mathbf{a})$ to obtain $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$.

Domain-specific / Physics-based Models. Parameterize known physics equations and fit only a few unknown parameters (e.g. friction coefficients, masses). Excellent data efficiency when the physics structure is correct.

Latent space models. Learn a compact latent representation $\mathbf{z}$ and model dynamics in that space rather than pixel space. Used in modern methods like Dreamer and TD-MPC.

Case Studies

Neural Network Dynamics with MPC (PETS/NND-MBRL)

Using neural networks as dynamics models (instead of GPs) allows scaling to more complex tasks while retaining the MPC structure of Version 1.5. The approach (Nagabandi et al., 2018) learns a neural network $f_\theta(\mathbf{s}, \mathbf{a})$ and uses random shooting (sample many action sequences, evaluate each using the model, pick the best) for planning.

The iterative loop:

1. Collect initial data with random policy
2. Train neural network dynamics model
3. At each step: plan with random shooting through the model
4. Execute first action (MPC), observe next state, add to dataset
5. Periodically retrain the model on all data so far

This approach achieves competitive sample efficiency with model-free methods on locomotion tasks while requiring far fewer real environment interactions.

Modern World Models

Recent work scales model-based RL to high-dimensional observation spaces (images) and complex environments. The central idea: learn the model in a *latent space* rather than pixel space.

Dreamer (v1–v4)

The Dreamer family (Hafner et al., 2019–2025) learns a Recurrent State-Space Model (RSSM) consisting of:

$$\text{Sequence model:} \quad h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$$

$$\text{Encoder:} \quad z_t \sim q_\phi(z_t \mid h_t, x_t)$$

$$\text{Dynamics predictor:} \quad \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t)$$

$$\text{Reward predictor:} \quad \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t)$$

The agent trains entirely in imagination: roll out the RSSM from an encoded real state, use the actor to select actions, and backpropagate through the latent trajectories. A value function handles the limited rollout horizon (T=16 steps in practice, beyond which model drift becomes too large).

Dreamer uses a combined policy gradient + dynamics backpropagation loss. Dreamer v3 achieves remarkable breadth, surpassing tuned specialists on 150+ tasks across Atari, DMLab, ProcGen, and even Minecraft diamond mining.

Dreamer v4 (2025) goes further: train the world model entirely from offline video data (no environment interaction required), then finetune the policy inside the world model using PMPO (a variant of policy gradient using only the sign of the advantage). This is the first agent to obtain diamonds in Minecraft *purely from offline data*.

TD-MPC

TD-MPC (Hansen et al., 2022–2024) takes a different approach: rather than learning to decode future observations, the world model predicts only task-relevant quantities (rewards and Q-values) in a latent space. At each step, it runs MPPI (a sampling-based MPC algorithm) inside the latent model with a learned Q-function as terminal cost:

$$\mu^*, \sigma^* = \arg\max_{(\mu,\sigma)} \mathbb{E}_{(\mathbf{a}_t,\ldots,\mathbf{a}_{t+H})\sim\mathcal{N}(\mu,\sigma^2)} \left[\underbrace{\gamma^H Q(\mathbf{z}_{t+H}, \mathbf{a}_{t+H})}_{\text{terminal cost}} + \underbrace{\sum_{h=t}^{H-1} \gamma^h R(\mathbf{z}_h, \mathbf{a}_h)}_{\text{stage cost}} \right]$$

Key difference from Dreamer: TD-MPC does not require decoding future pixel-level observations. This focuses the model capacity on task-relevant prediction rather than accurate pixel reconstruction, making optimization significantly easier. TD-MPC2 scales to a single 317M parameter model handling 80 continuous control tasks simultaneously, substantially outperforming Dreamer and model-free baselines.

When is Model-Based RL Actually Better?

- **Sample efficiency critical:** Real robot learning, dangerous environments, slow simulators
- **Sparse rewards:** Lucky rollouts are reused via the model rather than discarded
- **Given unlimited samples:** PPO-style model-free approaches tend to match or exceed MBRL performance

Practical alternatives in simulation: Dense reward shaping and smart reset strategies often outperform MBRL for tasks where a good simulator is available.

Summary

Model-based RL introduces a rich set of trade-offs compared to model-free approaches:

1. **The distribution mismatch problem** is central: always collect data from the distribution induced by the current policy, not a fixed base policy. MPC and iterative data collection are the standard fixes.

2. **Modern latent world models** (Dreamer, TD-MPC) demonstrate that model-based learning in compact latent spaces can achieve state-of-the-art performance on complex tasks. The trend is towards larger, pre-trained world models analogous to foundation models in language and vision.

3. **Sample efficiency** is the primary advantage of MBRL; the price is increased algorithmic complexity and the challenge of model accuracy in novel states.