

CS-566 Deep Reinforcement Learning

Learned Model-Based Learning



Nazar Khan
Department of Computer Science
University of the Punjab

From Model-Free to Model-Based RL

The previous chapters discussed model-*free* methods, and we saw their success in video games and simulated robotics.

In model-free methods:

- ▶ the agent updates a policy directly from the feedback that the environment provides,
- ▶ the environment performs the state transitions and calculates the reward.

A disadvantage of deep model-free methods:

- ▶ they can be slow to train,
- ▶ millions of environment samples are often needed for stable convergence or low variance.

Basic Idea of Model-Based RL

In contrast, with model-*based* methods the agent first builds its own internal transition model from the environment feedback.

Using this local transition model, the agent can:

- ▶ reason about the effect of actions on states and rewards,
- ▶ use a planning algorithm to play *what-if* games,
- ▶ generate policy updates without causing any state changes in the environment.

Generating policy updates from the internal model is called:

planning or **imagination**.

Indirect Policy Learning

Model-based methods update the policy *indirectly*:

1. first learn a local transition model from the environment,
2. then use this learned model to update the policy.

Two consequences of indirect learning:

Positive side

As soon as the agent has its own transition model, it can:

- ▶ learn the best policy for free,
- ▶ avoid further acting in the environment,
- ▶ achieve much lower sample complexity.

The Downside: Model Error

The downside is that the learned transition model may be inaccurate.

If the agent learns the policy from a **bad model**:

- ▶ the resulting policy may be of low quality,
- ▶ the policy may fail in the real environment,
- ▶ even infinite planning samples cannot fix a biased model.

Thus:

model bias and uncertainty are central challenges in model-based RL.

Historical Context and Model Types

The idea of learning an internal transition function is very old.

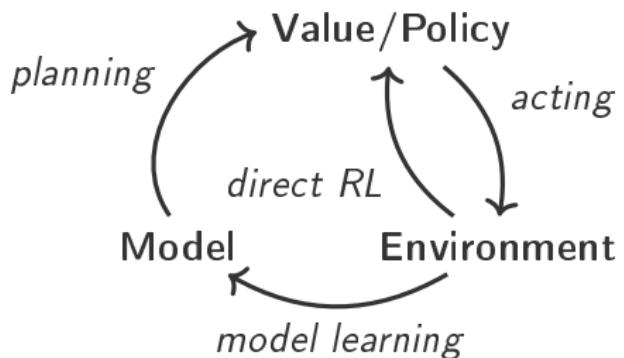
Transition models have been implemented in many ways:

- ▶ **Tabular models**
- ▶ **Deep neural network models**
- ▶ hybrids and structured models

Modern deep RL often uses:

- ▶ convolutional or recurrent predictors,
- ▶ latent dynamics models,
- ▶ multi-step world models and learned simulators.

Direct and Indirect Reinforcement Learning¹



¹sutton2018introduction.

Tabular Imagination

- ▶ **Dyna**: Classic approach² popularizing model-based RL.
- ▶ Environment samples used to:
 - ▶ train transition model,
 - ▶ plan to improve policy,
 - ▶ update policy directly.
- ▶ Hence: hybrid of model-free + model-based learning.
- ▶ “Imagination”: agent looks ahead using its own dynamics model.
- ▶ Imagined samples augment real samples at no cost.

²sutton1990integrated.

Why Hybrid?

- ▶ In strict model-based approach, policy is updated only from learned model.
- ▶ In Dyna, real samples *also* update policy directly.
- ▶ Hybrid = model-free updates + planning updates.
- ▶ Imagination \Rightarrow planning inside the agent's "mind".
- ▶ Real + imagined samples used together.

Strict Learned Dynamics Model

repeat

 Sample env E to get $D = (s, a, r', s')$

 Learn model $M = T_a(s, s'), R_a(s, s')$

for $n = 1 \dots N$ **do**

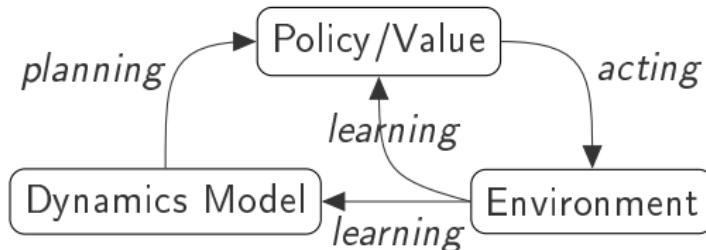
 Update policy $\pi(s, a)$ using M

 ▷ planning

end for

until π converges

Hybrid Model-Based Imagination



repeat

 Sample env E to get $D = (s, a, r', s')$

 Update policy $\pi(s, a)$ directly

 ▷ learning

 Learn model $M = T_a(s, s'), R_a(s, s')$

for $n = 1 \dots N$ **do**

 Update policy $\pi(s, a)$ using M

 ▷ planning

end for

until π converges

How Imagination Uses Feedback

- ▶ Real samples:
 - ▶ update policy,
 - ▶ update dynamics model.
- ▶ Model:
 - ▶ generates imagined transitions,
 - ▶ provides extra policy updates.
- ▶ Greatly increases number of updates without extra environment cost.

Dyna-Q Algorithm

Initialize $Q(s, a) \rightarrow \mathbb{R}$ randomly

Initialize $M(s, a) \rightarrow \mathbb{R} \times S$ randomly

► Model

repeat

Select $s \in S$ randomly

$$a \leftarrow \pi(s)$$

▷ $\pi(s)$ can be ϵ -greedy(s) based on Q

$$(s', r) \leftarrow E(s, a)$$

▷ Learn new state and reward from environment

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(s, a)]$$

$$M(s, a) \leftarrow (s', r)$$

for $n = 1, \dots, N$ do

Select \hat{s} and \hat{a} randomly

$(s', r) \leftarrow M(\hat{s}, \hat{a})$ \triangleright Plan imagined state and reward from model

$$Q(\hat{s}, \hat{a}) \leftarrow Q(\hat{s}, \hat{a}) + \alpha \cdot [r + \gamma \cdot \max_{a'} Q(s', a') - Q(\hat{s}, \hat{a})]$$

end for

until Q converges

Remarks on Dyna-Q

- ▶ Uses Q-function as behavior policy (ϵ -greedy).
- ▶ Each real sample:
 - ▶ updates Q-values (model-free),
 - ▶ updates model M .
- ▶ Each planning step:
 - ▶ samples M using random actions,
 - ▶ updates Q-values.
- ▶ N controls ratio: real vs. model updates.
- ▶ Typical large-scale settings: 1 : 1000 (env: model).

Reversible Planning vs. Irreversible Learning

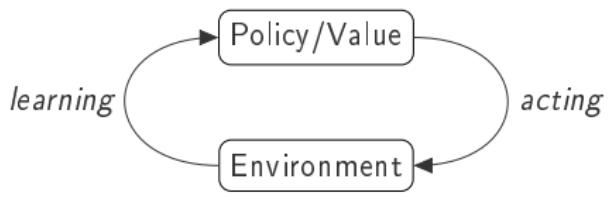
Model-Free vs. Model-Based (High-Level)

- ▶ Model-free:
 - ▶ Sample the environment directly
 - ▶ Update policy $\pi(s, a)$ in one step
 - ▶ No explicit transition model
- ▶ Model-based:
 - ▶ Learn dynamics model $\{T_a, R_a\}$ from samples
 - ▶ Update policy *indirectly* using the learned model
 - ▶ Planning replaces many real environment interactions

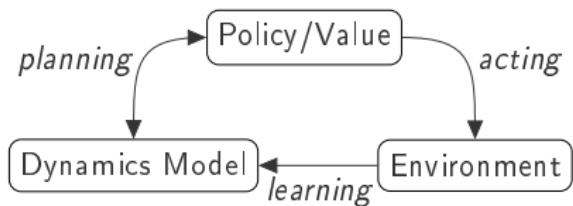
Key motivation: reduce environment samples while keeping/improving policy quality.

Model-Based vs. Model-Free

Model-Free Learning



Model-Based Learning and Planning



- ▶ Update policy directly from real experience

- ▶ Learn model $\{T_a, R_a\}$
- ▶ Plan with model to update policy

Learning vs. Planning: Big Picture

Planning:

- ▶ Uses an internal transition model
- ▶ Local state lives *inside the agent*
- ▶ Actions can be undone
- ▶ Enables search, backtracking, tree expansion
- ▶ Synonyms: imagination, simulation

Learning:

- ▶ No internal transition model
- ▶ Must act in the real environment
- ▶ Actions are irreversible
- ▶ No backtracking; only forward progression
- ▶ Synonyms: sampling, rollout

Why Planning is Reversible

- ▶ Planning uses the agent's internal model
- ▶ Local state is stored in memory
- ▶ Agent can:
 - ▶ Try an action
 - ▶ Observe predicted next state
 - ▶ Undo action and return to previous state
 - ▶ Explore alternative branches
- ▶ Enables tree search methods and “what-if” reasoning
- ▶ Similar to dreaming: we can undo actions in imagination³.

Why Learning is Irreversible

- ▶ Real actions applied to real environment
- ▶ Next state cannot be undone by the agent
- ▶ Environment transitions are one-way
- ▶ Learning must follow environment trajectory (a path)
- ▶ Policy learned by repeated real sampling
- ▶ No ability to backtrack or explore alternate realities

Comparing Planning and Learning

	Planning	Learning
Transition model in:	Agent	Environment
Undo possible?	Yes	No
State:	Reversible	Irreversible
Dynamics:	Backtracking	Forward-only
Structure:	Tree	Path
New state from:	Agent model	Env. samples
Reward from:	Agent model	Env. samples
Synonyms:	Imagination, simulation	Sampling, rollout

Similarity Between Planning and Learning

- ▶ Both collect (s, a, r, s') samples
- ▶ Both update the policy $\pi(s, a)$
- ▶ Difference is *source of samples*:
 - ▶ Learning: real environment
 - ▶ Planning: internal model
- ▶ Model-based RL learns both:
 - ▶ Policy $\pi(s, a)$
 - ▶ Dynamics $\{T_a(s, s'), R_a(s, s')\}$

Learning the Model

- ▶ In model-based RL, the transition model is learned from sampled interactions.
- ▶ Planning quality depends critically on model accuracy.
- ▶ If the model is inaccurate:
 - ▶ Planning does not improve the value or policy functions.
 - ▶ Performance may become worse than model-free RL.
- ▶ When the learning/planning ratio is large (e.g., 1/1000), even small model errors quickly degrade performance.
- ▶ Two major strategies to increase transition-model accuracy:
 1. **Uncertainty Modeling**
 2. **Latent Models**

Uncertainty Modeling: Motivation

- ▶ Transition variance can be reduced with more samples, but this is expensive.
- ▶ Goal: Explicitly estimate and propagate model uncertainty.
- ▶ Benefits:
 - ▶ More reliable long-horizon predictions.
 - ▶ Improved policy learning under model errors.

Gaussian Processes for Dynamics

- ▶ A popular approach for small or low-dimensional problems.
- ▶ Gaussian Processes learn:
 - ▶ A predictive function for the next state.
 - ▶ A covariance matrix representing uncertainty.
- ▶ Advantages:
 - ▶ Strong uncertainty estimates.
 - ▶ Very sample-efficient.
- ▶ Limitations:
 - ▶ Poor scalability to high-dimensional environments.
 - ▶ Computationally expensive for large datasets.
- ▶ Example: **PILCO** (Probabilistic Inference for Learning Control).

Example: PILCO

- ▶ Uses Gaussian Processes to model probabilistic dynamics.
- ▶ Learns policies by propagating uncertainty through predictions.
- ▶ Demonstrated strong sample efficiency on:
 - ▶ Cartpole
 - ▶ Mountain Car
- ▶ Limitation: Does not scale to high-dimensional inputs (e.g., raw pixels).

Trajectory Distribution Approaches

- ▶ Another method: sample trajectories optimized for cost.
- ▶ These trajectories are then used to train a policy.
- ▶ Uses locally linear models + stochastic trajectory optimization.
- ▶ Example: **Guided Policy Search (GPS)**
- ▶ GPS can efficiently train high-dimensional policies (hundreds–thousands of parameters).

Ensemble Methods

- ▶ Ensembles combine multiple models to reduce variance.
- ▶ Widely used in supervised ML (e.g., Random Forests).
- ▶ In model-based RL:
 - ▶ Each model provides a prediction.
 - ▶ The ensemble variance serves as an uncertainty estimate.
- ▶ Applications show strong performance on continuous-control tasks.

PETS: Probabilistic Ensembles with Trajectory Sampling

- ▶ Introduced by Chua et al. (2018).
- ▶ Uses:
 - ▶ An ensemble of probabilistic neural networks.
 - ▶ Stochastic trajectory sampling.
- ▶ Achieves excellent performance in:
 - ▶ Half-Cheetah
 - ▶ Reacher
 - ▶ Other MuJoCo tasks
- ▶ Improves sample efficiency vs. model-free baselines.

ME-TRPO: Ensembles + TRPO

- ▶ Proposed by Kurutach et al. (2018).
- ▶ Method:
 - ▶ Train an ensemble of neural network dynamics models.
 - ▶ During planning, each imaginary transition is sampled from a random ensemble member.
 - ▶ Use TRPO to optimize policy.
- ▶ Reported strong sample efficiency on:
 - ▶ Snake
 - ▶ Swimmer
 - ▶ Hopper
 - ▶ Half-Cheetah

Summary: Uncertainty Modeling

- ▶ Explicitly captures model uncertainty.
- ▶ Methods include:
 - ▶ Gaussian Processes
 - ▶ Trajectory-distribution methods (e.g., GPS)
 - ▶ Ensembles (e.g., PETs, ME-TRPO)
- ▶ Works well for:
 - ▶ Low-dimensional problems (GP methods)
 - ▶ Moderate-dimensional problems (ensembles)
- ▶ Next: A complementary approach — **Latent Models**.

Latent Models: Key Idea

- ▶ High-dimensional environments (e.g., images) contain many irrelevant details.
- ▶ Latent models compress observations into a smaller representation.
- ▶ Planning and learning occur entirely in this low-dimensional **latent space**.
- ▶ Benefits:
 - ▶ Reduced sample complexity.
 - ▶ More robust prediction.
 - ▶ Focus on task-relevant features only.

Why Latent Models?

- ▶ Raw observations often contain:
 - ▶ Background objects
 - ▶ Unchanging elements
 - ▶ Visual artifacts unrelated to reward
- ▶ Latent models:
 - ▶ Learn compact abstract states.
 - ▶ Remove irrelevant information.
 - ▶ Enable long-horizon planning in latent space.

Examples of Latent Model Approaches

- ▶ Many recent successes rely on latent-space planning:
 - ▶ World Models (Ha and Schmidhuber, 2018)
 - ▶ Dreamer / DreamerV2 / DreamerV3 (Hafner et al.)
 - ▶ Value Prediction Networks (VPN)
 - ▶ Mastering Atari with Discrete World Models (Hafner, 2020)
 - ▶ PlaNet (Hafner, 2019)
- ▶ All share the principle: **Plan and learn in a reduced latent space.**

Value Prediction Network (VPN)

- ▶ Proposed by Oh et al. (2017).
- ▶ Key idea: **Predict values and rewards without predicting observations.**
- ▶ Uses four differentiable latent-space functions:
 1. Encoding function $f_{\theta_e}^{enc}$
 2. Reward function $f_{\theta_r}^{reward}$
 3. Value function $f_{\theta_v}^{value}$
 4. Transition function $f_{\theta_t}^{trans}$
- ▶ Planning occurs on latent abstract states.

VPN: Latent-Space Functions

- ▶ **Encoding:**

$$f_{\theta_e}^{enc} : S_{actual} \rightarrow S_{latent}$$

Maps raw observations (e.g., images) to latent states.

- ▶ **Transition:** Predicts next latent state given action.
- ▶ **Reward:** Predicts expected reward in latent space.
- ▶ **Value:** Predicts the expected future return from a latent state.

VPN: Why It Works

- ▶ Latent states contain only task-relevant features.
- ▶ No need to model high-dimensional observations.
- ▶ Planning becomes:
 - ▶ Faster
 - ▶ Lower variance
 - ▶ Less computationally expensive
- ▶ Enables multi-step lookahead and imagination-based planning.

From Learning Models to Using Models

- ▶ So far we focused on improving the *accuracy* of learned internal models.
- ▶ We now shift from model *construction* to model *usage*.
- ▶ We study two planning approaches designed to tolerate model inaccuracies:
 - ▶ Trajectory rollouts with limited horizon
 - ▶ Model-predictive control (MPC)
- ▶ Goal: reduce the effect of model errors by limiting planning horizon and continuously re-planning.

Trajectory Rollouts

- ▶ At each planning step, the learned transition model

$$T_a(s) \rightarrow s'$$

predicts next states and rewards.

- ▶ Long rollouts accumulate large model errors.
- ▶ Therefore: avoid deep planning horizons.

Example: Gu et al. (2016)

- ▶ Use locally linear models.
- ▶ Rollout depth: 5–10 steps.
- ▶ Effective on MuJoCo tasks (Gripper, Reacher).

Model-Based Value Expansion (MVE)

- ▶ Feinberg et al. (2018): limit look-ahead to depth H , then combine:
 - ▶ Near future: model-based predictions
 - ▶ Far future: model-free value estimates
- ▶ Horizons tried: 1, 2, 10.
- ▶ Horizon 10 performs best on Swimmer, Walker, Half-Cheetah.
- ▶ Sample complexity improves over DDPG.

Other works:

- ▶ Janner et al. (2019), Kalweit & Boedecker (2017)
- ▶ All find that effective model horizons are much shorter than full task horizons.

Model-Predictive Control (MPC)

- ▶ Also known as *decision-time planning*.
- ▶ Standard technique in process engineering.
- ▶ Key idea:
 - ▶ Optimize a model-based plan over a short horizon.
 - ▶ Execute *only the first action*.
 - ▶ Re-learn and re-plan at every time-step.
- ▶ Why it works:
 - ▶ Many real processes are locally linear over small ranges.
 - ▶ Frequent re-planning prevents large error accumulation.

MPC in Real Systems

- ▶ Applied in automotive and aerospace domains:
 - ▶ Terrain-following
 - ▶ Obstacle avoidance
 - ▶ Complex process control
- ▶ Works well with inaccurate models due to short horizon updates.

Deep RL examples:

- ▶ Finn et al. (2017) and Ebert et al. (2018):
 - ▶ Visual foresight for robotic manipulation.
 - ▶ Model predicts future frames.
 - ▶ MPC selects lowest-cost action sequence.
- ▶ Capabilities:
 - ▶ Multi-object manipulation
 - ▶ Pushing, grasping, placing
 - ▶ Cloth folding

MPC with Ensemble Models: PETS

- ▶ PETS (Chua et al. 2018):
 - ▶ Uses probabilistic ensembles for dynamics learning.
 - ▶ Uses CEM (Cross-Entropy Method) for planning.
 - ▶ In MPC style: execute first action only; re-plan at every step.
- ▶ Common trend:
 - ▶ Ensemble dynamics models + MPC = robust planning.

Planning by a Neural Network?

- ▶ Traditionally:
 - ▶ Learn transition model with backprop.
 - ▶ Use hand-crafted algorithm for planning (e.g., limited-horizon search).
- ▶ Trend in ML:
 - ▶ Replace hand-coded algorithms with differentiable, learnable modules.
 - ▶ Train them end-to-end.
- ▶ Question:

Can we make the *planning* stage differentiable and learnable?

Why Might Planning-by-Network Work?

- ▶ Neural networks typically do:
 - ▶ Transformations
 - ▶ Filtering
 - ▶ Classification / selection
- ▶ Planning consists of:
 - ▶ Action selection
 - ▶ State unrolling
- ▶ RNNs and LSTMs *do* maintain state internally.
- ▶ Therefore: planning might be implementable using deep networks.

Value Iteration Networks (VIN)

- ▶ Tamar et al. (2016) introduced VIN:
 - ▶ A differentiable neural network that performs value iteration.
 - ▶ Designed for Grid world planning.
 - ▶ CNN layers emulate dynamic programming steps.
- ▶ Core idea:
 - ▶ Value iteration = repeated convolution + max-pooling.
 - ▶ Each CNN channel corresponds to an action's Q-value.
 - ▶ Stacking K convolution layers \approx K value-iteration updates.

Value Iteration Computation in CNN Form

Value iteration update:

$$V(s) = \max_a \sum_{s'} T_a(s, s')(R_a(s, s') + \gamma V(s'))$$

- ▶ Double loop over states and actions.
- ▶ CNN implementation:
 - ▶ Convolution implements local transition and reward propagation.
 - ▶ Max-pooling implements action maximization.
- ▶ The VIN module becomes a differentiable planner.
- ▶ Training via backprop learns:
 - ▶ Transition dynamics
 - ▶ Embedded planning behavior