MODEL-BASED POLICY OPTIMIZATION WITH SAC FOR EFFICIENT CONVERSATIONAL CHATBOTS

Dang Tran Tan Luc

August 5, 2025

Abstract

In this paper, we propose a model-based reinforcement learning method that can improve the uncertainty for policy in multitask reinforcement learning without sampling from the model. Most of model-free reinforcement learning agent use only the observation of the environment to make decisions. This cause uncertainty when they encounter a new environment or the environment change constantly. We introduce a method that use a latent variable as a prediction of the environment, and this variable is able to support an agent that can make certain decisions in the next state of the environment. In practice, it show that the policy depend on our latent to give action. This increases the stability of agent but still hold efficiency in maximizing return in fast changing environment..

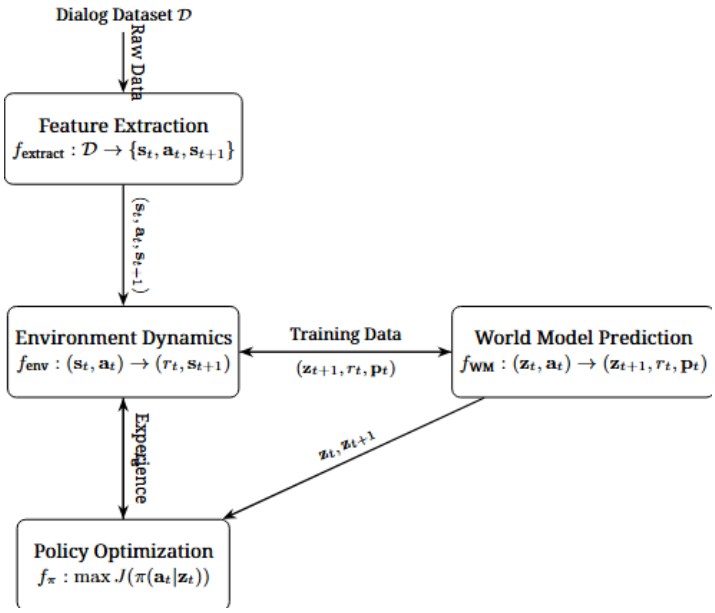

Figure 1: Architecture of model

1 Introduction

At present, reinforcement learning (RL)[**?**] stands as one of the most vibrant and rapidly advancing subfields within the realm of machine learning. Reinforcement learning is distinct from other machine learning fields like supervised learning[**?**] and unsupervised learning[**?**]. It focuses on maximizing rewards by taking appropriate actions in specific situations. Reinforcement learning is applied in various domains to determine optimal behavior or paths. Unlike supervised learning, where training data includes correct answers, reinforcement learning learns from trial and error without explicit answers.

It relies on accumulating data through interactions with the environment. The practical applications of reinforcement learning span across various domains, including autonomous driving cars [**?**][**?**], chatbot[**?**][**?**], robot[**?**][**?**][**?**], video-games[**?**], dialogue systems[**?**] and other industrial applications.

In RL, the agent interacts directly with the environment, it is learned to give the best action for a certain state to maximize the return. Suppose that the agent has been previously trained and when encountering a new state of the environment that it has never encountered before or when facing a new task. The agent will act with uncertainty, leading to model bias[**?**] and potentially causing collisions in the real world. This problem is called uncertainty in reinforcement learning[**?**]. For example, the agent is an autonomous car, while training, it meets a state that never met, suppose a hole on the road. In this case, uncertainty will be not allowed and a mistake will cause big damage. Another example is when we use reinforcement learning in chatbot applications, the agent sometimes will be confused by strange customer questions and give irrelevant replies to them.

The agent needs to be able to imagine what it will be if it gives certain actions before it steps into a real environment. So, we have to model the world. That is where model-based reinforcement learning comes from. The idea of model-based reinforcement learning is to model the environment to predict the future states of the agent. The model is like a memory that stores all information in history and then predicts future information. It uses background knowledge about the world to exhibit a simulation of the environment. The agent can try to act in that simulation before stepping into the real environment. This allows the agent to minimize risks in the real world. In each step, the model will predict some new states of the environment, then the agent tries to sample from this model to envision what going on in a few next steps. However, the environment is very large, the model is also very complex. In recent model-based reinforcement learning such as PETS [**?**], SimPLe [**?**], SOLAR [**?**], PlaNet [**?**], Dreamer [**?**],..., they use giant models to represent the environment.

In this paper, we study how to most effectively use a predictive model for policy optimization. We first formulate and analyze a class of model-based reinforcement learning algorithms with improvement guarantees. Although there has been recent interest in the monotonic improvement of used model-based reinforcement learning algorithms, most commonly used model-based approaches, lack the improvement guarantees that underpin many model-free methods. While it is possible to apply analogous techniques to the study of model-based methods to achieve similar guarantees, it is more difficult to use such analysis to justify model usage in the first place due to pessimistic bounds on the model error. However, we show that more realistic model error rates derived empirically allow us to modify this analysis to provide a more reasonable tradeoff on model usage.

Our main contribution is a practical algorithm built on these insights, which we call model-based policy optimization (MPBO), that makes limited use of a predictive model to achieve pronounced improvements in performance compared to other model-based approaches. More specifically, we disentangle the task horizon and model horizon by querying the model only for short rollouts. We empirically demonstrate that a large amount of these short model-generated rollouts can allow a policy optimization algorithm to learn substantially faster than recent model-based alternatives while retaining the asymptotic performance of the most competitive model-free algorithms. We also show that MBPO does not suffer from the same pitfalls as prior model-based approaches, avoiding model exploitation and failure on long-horizon tasks. Finally, we empirically investigate different strategies for model usage, supporting the conclusion that careful use of short model-based rollouts provides the most benefit to a reinforcement learning algorithm.

2 Background knowledge

2.1 Reinforcement Learning

Reinforcement Learning (RL). The standard framework of RL is the infinite-horizon Markov decision process (MDP) $M = \langle S, A, P, r, \mu_0, \gamma \rangle$ where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ the (possibly stochastic) transition dynamics, $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ the reward function, μ_0 the initial state distribution, and $\gamma \in [0, 1]$ the discount factor. The goal of RL is to find, for each state $s \in \mathcal{S}$, a distribution π (s) over the action space A, called the policy, that maximizes the expected sum of discounted rewards $\eta(\pi) := \mathbb{E}_{(s_0)\sim\mu_0, a_t\sim\pi, s_t>0\sim P} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$. Under a policy π, we define the state value function at s $\mathcal{S}$ as the expected sum of discounted rewards, starting from the state s, and

following the policy π afterwards until termination:$V\pi(s) = \mathbb{E}_{a_t \sim \pi, s_t > 0 \sim P}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \;\middle|\; s_0 = s\right]$.

2.2 Model-based learning

Model-based RL (MBRL). MBRL algorithms address the supervised learning problem of estimating the dynamics of the environment $\hat{P}$ (and sometimes also the reward function $\hat{r}$) from data collected when interacting with the real system. The model's loss function is typically the log-likelihood $\mathcal{L}(D; \hat{P}) = \frac{1}{N}\sum_{i=1}^{N} \log \hat{P}(s_{t+1}^{(i)} \mid s_t^{(i)}, a_t^{(i)})$ or Mean Squared Error (MSE) for deterministic models. The learned model can subsequently be used for policy search under the MDP $\hat{\mathcal{M}} = \langle S, A, \hat{P}, r, \mu_0, \gamma \rangle$. This MDP shares the state and action spaces S, A, reward function r, with the true environment M, but learns the transition probability $\hat{P}$ from the dataset D.

2.3 Large Language Model

Large Language Models (LLMs). Within the field of Natural Language Processing, Large Language Models (LLMs) have emerged as a powerful tool for understanding and generating human-like text. An LLM is typically defined as a neural network model, often based on the transformer architecture (Vaswani et al., 2017), that is trained on a vast corpus of sequences, $U = U_1, U_2, ..., U_i, ..., U_N$, where each sequence $Ui = (u_1, u_2, ..., u_j, ..., u_{n_i})$ consists of tokens u_j from a vocabulary V. Decoder-only LLMs (Radford et al., 2019; Dubey the Llama 3 team,2024) typically encode an autoregressive distribution, where the probability of each token is conditioned only on the previous tokens in the sequence, expressed ass $\mathrm{p}_\theta(U_i) = \prod_{j=1}^{n_i} p_\theta(u_j \mid u_{0:j-1}) Theparameters\theta$ are learned by maximizing the probability of the entire dataset, $p_\theta(U) = \prod_{i=1}^{N} p_\theta(U_i)$. Every LLM has an associated tokenizer, which breaks an input string into a sequence of tokens, each belonging to V.

3 Approach

3.1 Problem setup

Give me a dataset with limited texts or images of length $\mathcal{T}$, with $_t \in \mathcal{S}$, we want to predict the next states, rewards (s_T, r_T). If we use model-free, we can predict the next states and rewards, but with a large dataset of texts used for generating text (example: a chatbot assistant) or used for auto cars. Then, the Q-table can not save a large tuple. And it needs many datasets for a trained model. With this use case specific, I will use a model-based approach for that problem, which integrates planning and learning, and can process the above problem. It is just used to a little dataset for real experience, and using planning and learning to create a simulated experience to update the data. But in there, we are integrating LLM to compute the reward when customer replies dialog of seller (chabtbot).

3.2 Use case in reinforcement learning

As explored in the preceding sections. LLMs can be used as accurate dynamic learners for proprioceptive control through in-context learning. We now state our contributions to MBRL. First, we generalize the return bound of Model-Based Policy Optimization (MBPO) to the more general case of multiple branches and use it to analyze our method. Next, we leverage the LLM to augment the replay buffer of an off-policy RL algorithm, leading to a more sample-efficient algorithm. In a second application, we apply our method to predict the reward, resulting in a hybrid model-based policy evaluation technique. Finally, we show that the LLM provides

3.3 Algorithm

1. **Feature Extraction:** We perform feature extraction on the "Kinda Pizza" e-commerce dialog dataset, which contains multiple multi-turn conversations between a **customer** and a **seller**. Each dialog is processed to extract meaningful (state, action, and next state) tuples for training a dialog agent.

- **State (s):** Represents the conversation's history prior to the seller's response. This includes a sequence of alternating utterances from both the customer and the seller, up to a maximum number of turns defined by a hyperparameter **max_seq_len**. The conversation is initialized with a fixed greeting utterance: `"I'am KinPizza chatbot, what can I help you with"`
- **Action (a):** The seller's current response to the customer query based on the given state. This is the target response that the agent should learn to generate
- **Next State (s')**: The customer's next utterance following the seller's reply, representing the subsequent context in the dialog.

To convert these textual sequences into numerical representations, we use a pre-trained BERT model as a sentence encoder. Each utterance is passed through the BERT tokenizer to obtain token IDs, attention masks, and segment embeddings. The tokenized inputs are then fed into the BERT encoder, from which we extract the embedding corresponding to the `[CLS]` token of the final hidden layer. This embedding serves as a fixed-length vector representation (768 dimensions) for the utterance.

For each dialog history (state), we encode up to `max_seq_len` utterances. If the history is shorter than the maximum length, we pad the sequence with zero vectors. The final state representation is thus a tensor of shape $[L, 768]$, where L is the number of utterances in the state (at most `max_seq_len`).

This structured approach enables us to transform raw dialog data into meaningful state-action-next state transitions suitable for training reinforcement learning or supervised learning agents in conversational AI tasks.

2. **Model-Based Policy Optimization (MBPO)** Model-Based Policy Optimization (MBPO) is a reinforcement learning algorithm that combines a learned world model with deep policy optimization. It improves data efficiency and generalization, particularly in complex environments such as natural language dialog systems where both the state and action spaces are high-dimensional and dynamic.

   - **Latent State Representation:** In dialog tasks, the environment observation is a sequence of utterances with variable length. To make the state space tractable, we encode the conversation into a fixed-dimensional latent space.
     Each utterance x_i is first embedded using a pretrained BERT model:

     $$h_i = \text{BERT}(x_i)[:, 0], \quad h_i \in \mathbb{R}^{768}$$

     The sequence of utterances up to time t is:

     $$X_t = \{h_1, h_2, \ldots, h_t\}$$

     This sequence is passed through a Transformer encoder:

     $$H_t = \text{Transformer}(X_t)$$

     The latent state $z_t \in \mathbb{R}^{d_z}$ is extracted by projecting the final hidden state:

     $$z_t = W \cdot H_t[-1] + b, \quad W \in \mathbb{R}^{d_z \times 768}$$

   - **World Model $\hat{M}$:**The world model $\hat{M}$ learns the environment dynamics by predicting the next latent state and the expected reward. This model is used to generate synthetic transitions for training the policy.
     Given a latent state z_t and action a_t, the model predicts:

     $$(\hat{z}_{t+1}, \hat{r}_t) = \hat{M}(z_t, a_t)$$

     The world model is typically implemented as a feedforward neural network or recurrent model and trained by minimizing the following loss:

     $$L_{\text{model}} = \|\hat{z}_{t+1} - z_{t+1}\|_2 + \|\hat{r}_t - r_t\|_2$$

- **Policy Learning via Soft Actor-Critic (SAC):**
  MBPO adopts the Soft Actor-Critic (SAC) algorithm to optimize the policy in the latent space. The goal is to maximize the expected reward while encouraging exploration through entropy regularization:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_t \gamma^t \left(r_t + \alpha \mathcal{H}(\pi(\cdot \mid z_t)) \right) \right]$$

  Where:
  - $\mathcal{H}(\pi)$ is the entropy of the policy,
  - α is a temperature parameter that balances exploration and exploitation.

  The critic and actor networks are optimized via:
  Critic loss:

$$L_Q = \mathbb{E}_{(z_t, a_t, r_t, z_{t+1})} \left[\left(Q(z_t, a_t) - \left(r_t + \gamma \mathbb{E}_{a' \sim \pi(\cdot \mid z_{t+1})} \left[Q(z_{t+1}, a') - \alpha \log \pi(a' \mid z_{t+1}) \right] \right) \right)^2 \right]$$

  Actor loss:

$$L_\pi = \mathbb{E}_{z_t} \left[\mathbb{E}_{a_t \sim \pi(\cdot \mid z_t)} \left[\alpha \log \pi(a_t \mid z_t) - Q(z_t, a_t) \right] \right]$$

- **Real and Virtual Experience Replay** MBPO utilizes both real and synthetic (imagined) data for policy training:
  Real transitions (z_t, a_t, r_t, z_{t+1}) are collected from actual environment interactions.
  The world model $\hat{M}$ is used to generate synthetic rollouts:

$$(z_t, a_t) \to \hat{M}(\hat{z}_{t+1}, \hat{r}_t)$$

  Both real and synthetic transitions are stored in the replay buffer and used to train the SAC policy.
  This hybrid training approach improves sample efficiency and allows for faster policy convergence.

- **Overall MBPO Training Procedure**
  (a) Collect real interactions from the dialog environment.
  (b) Encode the raw observation into latent state z_t.
  (c) Train the world model using the real dataset.
  (d) Use the world model to generate synthetic rollouts for k steps.
  (e) Train the actor and critic networks on both real and synthetic data.
  (f) Periodically update target networks using soft updates.

3. **Attention Mechanism:** The attention mechanism is the core innovation of the Transformer, enabling it to weigh the importance of different parts of the input data when making predictions. In the context of the code, attention is used within the BERT model for text processing and likely in the multimodal Transformer for cross-modal fusion.

   Attention allows the model to focus on relevant parts of the input sequence when processing each element, rather than treating all parts equally. It computes a weighted sum of input features, where weights reflect the relevance of each input to the current context. The primary attention mechanism used in Transformers is scaled dot-product attention. Here's how it works:

   - **Inputs:**
     - **Query (Q)**: Represents the current token or feature being processed.
     - **Key (K)**: Represents all tokens/features in the sequence to compare against.
     - **Value (V)**: Contains the actual information to be weighted.
   - **Linear Transformations:** These are derived from the input embeddings via linear transformations:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

     where X is the input sequence and W_Q, W_K, W_V are learnable weight matrices.

- **Attention scores:**
  - Compute the similarity between the query and each key using a dot product: Score = QK^T.
  - Scale the scores by the square root of the key dimension ($\sqrt{d_k}$) to prevent large values: Scaled Score = $\frac{QK^T}{\sqrt{d_k}}$.
  - Apply a softmax to obtain attention weights: Attention Weights = softmax $\left(\frac{QK^T}{\sqrt{d_k}}\right)$.

- **Multi-Head Attention:**
  - Instead of computing attention once, multi-head attention splits the input into multiple subspaces (heads), applies scaled dot-product attention to each, and concatenates the results.
  - **Formula:** MultiHead(Q, K, V) = Concat$(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W_O$, where head_i = Attention$(QW_{Q_i}, KW_{K_i}, VW_{V_i})$, and W_O is a projection matrix.

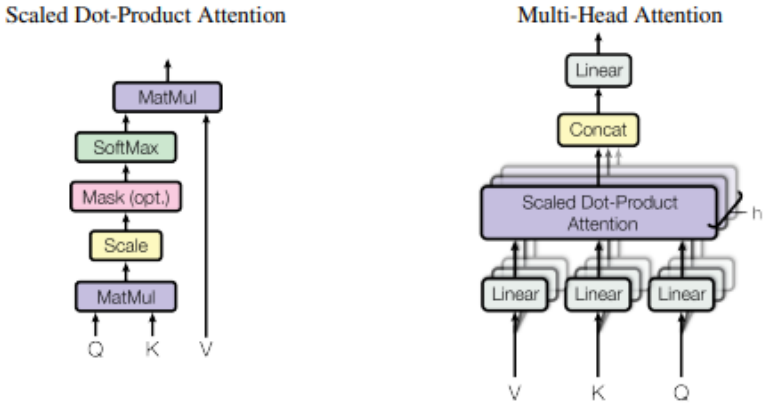

Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

The attention mechanism is the core innovation of the Transformer, enabling it to weigh the importance of different parts of the input data when making predictions. In the context of the code, attention is used within the BERT model for text processing and likely in the multimodal Transformer for cross-modal fusion.

Attention allows the model to focus on relevant parts of the input sequence when processing each element, rather than treating all parts equally. It computes a weighted sum of input features, where weights reflect the relevance of each input to the current context. The primary attention mechanism used in Transformers is scaled dot-product attention. Here's how it works:

- **Inputs:**
  - **Query (Q)**: Represents the current token or feature being processed.
  - **Key (K)**: Represents all tokens/features in the sequence to compare against.
  - **Value (V)**: Contains the actual information to be weighted.
- **Linear Transformations:** These are derived from the input embeddings via linear transformations:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

  where X is the input sequence and W_Q, W_K, W_V are learnable weight matrices.

- **Attention scores:**
  - Compute the similarity between the query and each key using a dot product: Score = QK^T.

- Scale the scores by the square root of the key dimension ($\sqrt{d_k}$) to prevent large values: Scaled Score = $\frac{QK^T}{\sqrt{d_k}}$.
- Apply a softmax to obtain attention weights: Attention Weights = softmax $\left(\frac{QK^T}{\sqrt{d_k}}\right)$.

- **Multi-Head Attention:**
  - Instead of computing attention once, multi-head attention splits the input into multiple subspaces (heads), applies scaled dot-product attention to each, and concatenates the results.
  - **Formula:** $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W_O$, where $\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i})$, and W_O is a projection matrix.

4 Experience

5 Discusstion

6 Good luck!

We hope you find Overleaf useful, and do take a look at our help library for more tutorials and user guides! Please also let us know if you have any feedback using the Contact Us link at the bottom of the Overleaf menu — or use the contact form at https://www.overleaf.com/contact.

References