Section 8: Model Based Reinforcement Learning

March 16, 17

Author: Vivek Myers

1 Learning with a Simulator

How can we use an imperfect simulator of the environment to learn more effectively? This is the core problem of model-based reinforcement learning (MBRL).

A naive approach fits a model of the environment dynamics to data (Algorithm 1), and then uses it as a simulator to optimize the policy.

Algorithm 1: Naive Model-Based RL

```
1: Initialize model $f_\theta(s, a)$ and policy $\pi_\beta(a \mid s)$
2: while not converged do
3:     Collect data $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^N$ by executing $\pi_\beta$ in the environment
4:     Fit model $f_\theta$ to $\mathcal{D}$:
           $\min_\theta \mathcal{L}(\theta) = \sum_{i=1}^N \|f_\theta(s_i, a_i) - (r_i, s'_i)\|^2$
5:     Optimize policy $\pi_\beta$ using $f_\theta$ as a simulator with any model-free RL algorithm (e.g., PPO)
```

In practice, this suffers from several issues:

- Errors in the simulator compound over time, leading to poor long-horizon performance.
- RL might still be hard in the simulator.
- Learned simulator with deep neural net may be slow and inaccurate, especially with image or high-dimensional observations.
- Distribution shift between the dataset and the visitation of the policy optimized in the simulator is unbounded.

1.1 Handling Noise and Distribution Shift

As we optimize a policy π_f in the simulator f learned under π_β, we encounter distribution shift in the state occupancy p_{π_f} relative to the distribution p_{π_β} used to collect $\mathcal{D}$ and fit f. When training π_f, there may be areas of the state space where f is overly optimistic due to having low mass under p_{π_β}, which π_f can exploit by visiting.

There may also be areas of the state space where there is high variance in the dynamics and the resulting rewards of taking an action. If the reward of the mode of the distribution $p(s' \mid s, a)$ is much higher than the expected reward, the policy will be incentivized to visit these high-risk states with low expected reward.

We need two changes to Algorithm 1 to handle these issues:

1. Constrain the policy at each step to a trust region where state visitation doesn't change drastically:
$$D_{\mathrm{KL}}\left(\pi_f \| \pi_\beta\right) \le \epsilon$$
2. Use a probabilistic dynamics model $f(s' \mid s, a)$ fit to the true dynamics $p(s' \mid s, a)$.

These changes are shown in Algorithm 2.

Algorithm 2: MBRL (Improved)

```
1: Initialize model $f_\theta(s, a)$ and policy $\pi_\beta(a \mid s)$
2: while not converged do
3:     Collect data $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$ by executing $\pi_\beta$ in the environment
4:     Fit probabilistic model $f_\theta$ to $\mathcal{D}$: $\min_\theta \mathcal{L}(\theta) = \mathbb{E}_{\pi_\beta}[D_{\mathrm{KL}}(f \mid p\|)] \propto \sum_{i=1}^N -\log f_\theta(s_i' \mid s_i, a_i)$
5:     Optimize $\pi_\beta$ using $f_\theta$ as a simulator with a model-free algorithm s.t. $D_{\mathrm{KL}}(\pi_f \| \pi_\beta) \leq \epsilon$
```

1.2 Modeling Epistemic Uncertainty

We consider two kinds of uncertainty over dynamics:

Aleatoric: the stochasticity of the dynamics $p(s' \mid s, a)$ themselves

Epistemic: the uncertainty over the true dynamics p given the data $\mathcal{D}$

We would like to capture both types of uncertainty when we model dynamics, such as when we fit f in Algorithm 2.

1.2.1 Bayesian Neural Networks

If we have a neural network $p_\theta(s'|s, a)$ modeling the dynamics with aleatoric uncertainty, we can in theory use a prior distribution $p(\theta)$ over the parameters to capture the epistemic uncertainty $p(\theta \mid \mathcal{D})$. Bayesian neural network methods attempt this, usually assuming an isotropic Gaussian prior $p(\theta)$ and diagonal Gaussian variational posterior for $p(\theta \mid \mathcal{D})$.

See Blundell et al. [1], Karaletsos and Bui [2] for examples of these methods. In practice, simpler ensemble-based approaches are more common and effective for capturing epistemic uncertainty in MBRL, as we discuss in Section 1.3.

1.3 Bootstrap Ensembles

Train ensemble of models $f_{\theta_1}, f_{\theta_2} \dots f_{\theta_k}$ so

$$p(\theta \mid \mathcal{D}) \approx \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}(\theta = \theta_i).$$

Each θ_i *should* be an i.i.d sample $\theta_i \sim p(\theta \mid \mathcal{D})$. To learn θ_i, we can fit f_{θ_i} to independent sub-datasets $\mathcal{D}_i$ of $\mathcal{D}$. This means we need to sample the $\mathcal{D}_i$ with replacement from $\mathcal{D}$ (bootstrap sampling) — in practice, just training separate models is enough to capture epistemic uncertainty due to the randomness of SGD and initialization.

2 Planning

A model lets us predict the consequences of arbitrary sequences of actions. If we optimize over these sequences of actions for our objective, in theory we can select good actions without learning a policy at all. These "planning" algorithms can broadly be categorized as either closed loop (replan based on outcome of previous actions) or open loop (plan a sequence of actions and execute it without replanning). Open loop planning is easier but less robust to errors in the model, in addition to being suboptimal in stochastic environments.

2.1 Open Loop Control

In the open loop case, we optimize a sequence of actions $\mathbf{A} \triangleq A_1 \dots A_N$ to maximize the expected return $\mathcal{J}(\mathbf{A})$

$$\mathbf{A}^* \leftarrow \arg\max_{\mathbf{A}} \mathcal{J}(\mathbf{A}) = \arg\max_{\mathbf{A}} \mathbb{E}_{\{s_{t+1} \sim f(\cdot|s_t, \mathbf{A}_t)\}_{t=1}^{N-1}} \left[\sum_{t=1}^{N} r(s_t, A_t) \right] \tag{1}$$

In addition to compounding errors, open loop methods are limited in stochastic environments because they can't react to outcomes of the noise in the dynamics.

2.1.1 Random Shooting

This is the simplest method. We sample random sequences of actions $\mathbf{A}$ and select the highest return under the model f.

Algorithm 3: Random Shooting

```
1: Sample $N$ random action sequences $\{\mathbf{A}_i\}_{i=1}^N$
2: Evaluate $\mathcal{J}(\mathbf{A}_i)$ for each sequence using the model $f$ (1)
3: Select $\mathbf{A}^* = \arg\max_{\mathbf{A}_i} \mathcal{J}(\mathbf{A}_i)$
```

2.1.2 Cross-Entropy Method

The cross-entropy method (Algorithm 4) iteratively optimizes a set of N action sequences by taking the top M sequences (elites), fitting a distribution to them, and sampling the next set of N sequences from this distribution.

Algorithm 4: Cross-Entropy Method

```
1: Initialize distribution $p(\mathbf{A})$ over action sequences (e.g., Gaussian with mean 0 and large variance)
2: while not converged do
3:     Sample $N$ action sequences $\{\mathbf{A}^{(i)}\}_{i=1}^N$ from $p(\mathbf{A})$
4:     Evaluate $\mathcal{J}(\mathbf{A}^{(i)})$ for each sequence using the model $f$ (1)
5:     Select top $M$ sequences $\{\mathbf{A}^{(i_j)}\}_{j=1}^M$ with highest returns ("elite" sequences)
6:     Update distribution $p(\mathbf{A})$ to fit the selected top sequences (e.g., update mean and covariance
       if Gaussian)
```

2.2 Closed Loop Control

In the general closed loop case, we can still plan with a model by optimizing the return of the policy π_θ when running with the model f. The closed-loop return can be expressed as:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta, f} \left[\sum_{t=1}^{N} r(s_t, a_t) \right] = \int p(s_1) \prod_{t=1}^{N} \pi_\theta(a_t \mid s_t) f(s_{t+1} \mid s_t, a_t) \sum_{t=1}^{N} r(s_t, a_t) \, \mathrm{d}s_1 \, \mathrm{d}a_1 \dots \mathrm{d}s_N \, \mathrm{d}a_N. \tag{2}$$

We can compute gradients through the model to maximize $\mathcal{J}(\theta)$:

$$\nabla_\theta \mathcal{J}(\theta) = \sum_{t=1}^{H} \frac{d\mathbf{a}_t}{d\theta} \frac{d\mathbf{s}_{t+1}}{d\mathbf{a}_t} \left(\sum_{t'=t+1}^{H} \frac{dr_{t'}}{d\mathbf{s}_{t'}} \left(\prod_{t''=t+2}^{t'} \frac{d\mathbf{s}_{t''}}{d\mathbf{a}_{t''-1}} \frac{d\mathbf{a}_{t''-1}}{d\mathbf{s}_{t''-1}} + \frac{d\mathbf{s}_{t''}}{d\mathbf{s}_{t''-1}} \right) \right)$$

The product of Jacobians in the above expression can lead to vanishing or exploding gradients, which makes this approach difficult to optimize in practice.

3 Accelerating Model-free RL with a Model

Counterintuitively, the most effective use of a model is often to accelerate model-free algorithms with extra data rather than optimizing it directly. Because of compounding errors, effective use of a model requires making use of short rollouts from the model. We can then use off-policy model-free algorithms to learn from both the simulated and real rollouts.

3.1 Dyna

Dyna-like algorithms [3] make use of short rollouts from the model starting from states visited in the real environment $s \sim p^\pi$. A version of the core algorithm is shown in Algorithm 5.

Algorithm 5: Dyna

```
1: Initialize model $f_\theta(s' \mid s, a)$ and policy $\pi_\theta$
2: while training do
3:     Collect dataset $\mathcal{D}$ from the environment by executing $\pi_\theta$
4:     Fit model $f_\theta$ to $\mathcal{D}$ (with e.g., line 4 of Algorithm 2)
5:     for $K$ steps do
6:         Sample $s, a \sim \mathcal{D}$
7:         Simulate $s' \sim f_\theta(\cdot \mid s, a)$ and $r = r(s, a)$
8:         Train $\pi_\theta$ with model-free off-policy algorithm (e.g., Q-learning) on the simulated transition
           $(s, a, r, s')$
9:         Rollout $N$ more steps in the model starting from $s'$ to get a trajectory $\tau$ and train on it
```

This core recipe is adapted by more recent deep RL algorithms such as MBA [4], MVE [5], and MBPO [6].

References

[1] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. aXiv:1505.05424, 2015.

[2] Theofanis Karaletsos and Thang D Bui. Hierarchical Gaussian Process Priors for Bayesian Neural Network Weights. *Neural Information Processing Systems*, volume 33, pp. 17141–17152, 2020.

[3] Richard Sutton. Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. *Machine Learning Proceedings 1990*, pp. 216–224. Morgan Kaufmann, 1990.

[4] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning With Model-Based Acceleration. aXiv:1603.00748, 2016.

[5] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. aXiv:1803.00101, 2018.

[6] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to Trust Your Model: Model-Based Policy Optimization. *Neural Information Processing Systems*, volume 32, 2019.