Lecture #8: Model-Based RL and Policy Learning

Nick, Nan and Halder

1 Introduction

Problems for backpropagating directly into policy

What’s the problem?

- Similar parameter sensitivity problems as shooting methods
- But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming
- Similar problems to training long RNNs with BPTT
  - Vanishing and exploding gradients
  - Unlike LSTM, we can’t just “choose” a simple dynamics, dynamics are chosen by nature

Constraining trajectory optimization with dual gradient descent

\[
\min_{\tau, \theta} c(\tau) \text{ s.t. } u_t = \pi_{\theta}(x_t)
\]

\[
\bar{L}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t (\pi_{\theta}(x_t) - u_t) + \sum_{t=1}^{T} \rho_t (\pi_{\theta}(x_t) - u_t)^2
\]

1. Find \(\tau \leftarrow \arg \min_{\tau} \bar{L}(\tau, \theta, \lambda)\) (e.g. via iLQR)
2. Find \(\theta \leftarrow \arg \min_{\theta} \bar{L}(\tau, \theta, \lambda)\) (e.g. via SGD)
3. \(\lambda \leftarrow \lambda + \alpha \frac{d\theta}{d\lambda}\)
Deterministic case

\[
\min_{\tau, \theta} c(\tau) \quad \text{s.t.} \quad u_t = \pi_{\theta}(x_t)
\]

\[
\hat{L}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda t (\pi_{\theta}(x_t) - u_t) + \sum_{t=1}^{T} \rho t (\pi_{\theta}(x_t) - u_t)^2
\]

1. Optimize \( \tau \) with respect to surrogate \( \hat{c}(\tau) \)
2. Optimize \( \theta \) with respect to supervised objective
3. Increment or modify dual variables \( \lambda \)

GPS

Stochastic (Gaussian) GPS with local models

DAgger

Imitating optimal control with DAgger

1. from current state \( s_t \), run MCTS to get \( a_t, a_{t+1}, \ldots \)
2. add \( (s_t, a_t) \) to dataset \( D \)
3. execute action \( a_t \sim \pi(\cdot | s_t) \) (not MCTS action)
4. update the policy by training on \( D \)

PLATO
Dagger does not care about how the actions are generated, it needs to make sure that actions are optimal with respect to the real reward function.

Imitating MPC: PLATO algorithm

1. train $\pi_t(o_t|x_t)$ from data $D = \{o_1, u_1, \ldots, o_N, u_N\}$
2. run $\hat{f}(o_t, a_t)$ to get dataset $D_t = \{a_1, \ldots, a_M\}$
3. Ask computer to label $D_t$ with actions $u_t$
4. Aggregate: $D \leftarrow D \cup D_t$

Simple stochastic policy: $\hat{\pi}(u_t|x_t) = N(K_t x_t + b_t, \Sigma_t)$

$\hat{\pi}(u_t|x_t) = \arg\min_{\pi} \mathbb{E}_{x_t \sim D_t} \left[ \mathbb{E}[r(x_t, u_t) + \lambda D(x_t, \hat{f})(u_t)] \right]$

DAgger vs GPS

- DAgger does not require an adaptive expert
  - Any expert will do, so long as states from learned policy can be labeled
  - Assumes it is possible to match expert’s behavior up to bounded loss
    - Not always possible (e.g. partially observed domains)
- GPS adapts the “expert” behavior
  - Does not require bounded loss on initial expert (expert will change)

Why imitate?

- It combines supervised learning and control and planning, which are stable and reliable to use
- Input is $o_t$ instead of $x_t$ for handling real observation
- get rid of numerical instability
Why imitate?

- Relatively stable and easy to use
  - Supervised learning works very well
  - Control/planning (usually) works very well
  - The combination of the two (usually) works very well
- Input remapping trick: can exploit availability of additional information at training time to learn policy from raw observations
- Overcomes optimization challenges of backpropagating into policy directly
- Usually sample-efficient and viable for real physical systems

Dyna Algorithm

Dyna is an online Q-learning algorithm that performs model-free RL with a model.

1. given state $s$, pick action $a$ using exploration policy
2. observe $s'$ and $r$, to get transition $(s, a, s', r)$
3. update model $p(s'|s, a)$ and $\hat{r}(s, a)$ using $(s, a, s')$
4. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. repeat $K$ times:
   - sample $(s, a) \sim B$ from buffer of past states and actions
6. Q-update: $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s',r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

Comparison: Model-Based RL VS Integrated Architecture (Dyna)

Figures are taken from Richard Sutton’s book: Reinforcement Learning: An Introduction
General “Dyna-style” model-based RL recipe

1. collect some data, consisting of transitions \((s, a, s', r)\)
2. learn model \(\hat{p}(s'|s, a)\) (and optionally, \(\hat{r}(s, a)\))
3. repeat \(K\) times:
   4. sample \(s \sim B\) from buffer
   5. choose action \(a\) (from \(B\), from \(\pi\), or random)
   6. simulate \(s' \sim \hat{p}(s'|s, a)\) (and \(r = \hat{r}(s, a)\))
   7. train on \((s, a, s', r)\) with model-free RL
   8. (optional) take \(N\) more model-based steps

References

- https://dl.acm.org/citation.cfm?id=122377
- https://medium.com/@ranko.mosic/online-planning-agent-dyna-q-algorithm-and-dyna-maze-example-sutton-and-barto-2016-7ad84a6dc52b

2 Summary

Model-based RL algorithms summary

- Learn model and plan (without policy)
  - Iteratively collect more data to overcome distribution mismatch
  - Replan every time step (MPC) to mitigate small model errors
- Learn policy
  - Backpropagate into policy (e.g., PILCO) – simple but potentially unstable
  - Imitate optimal control in a constrained optimization framework (e.g., GPS)
  - Imitate optimal control via DAgger-like process (e.g., PLATO)
  - Use model-free algorithm with a model (Dyna, etc.)

THIS WILL BE ON HW4!
Limitations of model-based RL

- Need some kind of model
  - Not always available
  - Sometimes harder to learn than the policy
- Learning the model takes time & data
  - Sometimes expressive model classes (neural nets) are not fast
  - Sometimes fast model classes (linear models) are not expressive
- Some kind of additional assumptions
  - Linearizability/continuity
  - Ability to reset the system (for local linear models)
  - Smoothness (for GP-style global models)
  - Etc.

Model-Free RL

- No model
- Learn value function (and/or policy) from real experience

Model-Based RL (using Sample-Based Planning)

- Learn a model from real experience
- Plan value function (and/or policy) from simulated experience

Dyna

- Learn a model from real experience
- Learn and plan value function (and/or policy) from real and simulated experience

3 Questions

1. Why quadratic loss in the second term

Deterministic case

\[
\min_{\tau, \theta} c(\tau) \text{ s.t. } u_t = \pi_\theta(x_t)
\]

\[
\tilde{L}(\tau, \theta, \lambda) = c(\tau) + \sum_{t=1}^{T} \lambda_t (\pi_\theta(x_t) - u_t) + \sum_{t=1}^{T} \rho_t (\pi_\theta(x_t) - u_t)^2
\]

2. Is iLQR a shooting method or a collocation method