RL seminar #4: Model-based RL

Maksim Kretov

MIPT, Deep learning lab & iPavlov.ai

18 Nov 2017

Outline

Table of Contents

Assignments

Coding (Programming assignment #2)

Deadline: 24 Nov 2017 (Friday)

Quiz

Quiz N3 to be issued.

Questions

Lack of questions on model-based RL!

Course

Covered majority of classical algorithms, next advanced techniques. Some of not covered methods/chapters: SARSA, DPG, multi-armed bandits.

Table of Contents

Model-free RL

On-policy methods

Agent follow current policy.

- Online methods: AxC (updates during episode)
- Batch methods: Vanilla PG, TRPO (sample trajectories)

Off-policy methods

Agent may follow any behavioral policy $\rightarrow$ can re-use information about past transitions.

- Q-learning
- Deterministic Policy Gradient

No explicit model of the environment.

Premises for model-based RL

- Sometimes we have access to model of environment
- Or can (easily) learn it

Advantages

- Increase sample efficiency
- Apply tools from closely related fields (optimal control)

Table of Contents

Discrete action space

We have access to environment simulator ("checkpoints").

Monte Carlo tree search

- Find leaf s_{leaf} using TreePolicy(s)
- Evaluate the leaf using DefaultPolicy(s_{leaf})
- Update all values in tree between s and s_{leaf}

See pseudo-code in [1].

Applications

Works well even with random policy: Atari games, Go.

[1] https://arxiv.org/pdf/1207.4708.pdf

Continuous case

Let's forget for two slides about neural nets, Atari games etc.
Consider continuous state space, action space.

Task definition

$\min_{u_1,..u_T} \sum_{t=1}^{T} c(x_t, u_t)$
$x_t = f(x_{t-1}, u_{t-1})$
If we know f and it is smooth enough, we can solve minimization task by some variant of gradient descent.

Trajectory optimization

- Shooting method (optimize actions: embedded constraints into minimization equation)
- Collocation method (optimize over actions and states: optimization with constraints)

Linear Quadratic Regulator

We use "shooting" method (removed constraints).

Assumptions

- Linear dynamic
- Quadratic cost

High-level plan

- Start from the last action u_T: express it in terms of unknown last state x_T
- Go backwards: solve for u_{T-1} in terms of x_{T-1}, express cost at the step T in terms of x_{T-1} and u_{T-1} using known f.
- Backward recursion until reaching x_0.

Linear Quadratic Regulator

Backward recursion

for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t,\mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$

Forward recursion

for $t = 1$ to T:

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

[1] Source: https://goo.gl/TUjLDa (Berkeley's CS294)

Iterative Linear Quadratic Regulator

Approximate non-linear system as a linear-quadratic system.

iLQR pseudo-code

1. Until convergence:
2.   Calculate derivatives F_t, C_t, c_t
3.   Run LQR backward pass
4.   Run forward pass with real non-linear dynamic
5.   Update trajectory ($\Rightarrow$ Taylor expansion in other points)

Unknown dynamic

Idea: learn dynamic and use previous tools.

Approaches in order of increasing complexity:

1. Collect *some* data D, fit model f for D and plan. But: distribution mismatch.
2. Iteratively collect more relevant data for fitting f (DAGGER-like).
3. Add re-planning every step (Model Predictive Control).

All use global models of the environment.
ATTN: we don't train policy in this procedure $\Rightarrow$ computationally expensive. It is built implicitly during iLQR.

Unknown dynamic

Local dynamic

1. Run simulator, collect similar trajectories
2. For each time step, fit dynamics
3. Improve controller (at each time step u_t is function of x_t)

Questions

1. Slide 13-14 (L9) – same planning procedure, for example iLQR.
2. Slide 15 – BP for whole computational graph. Very easy way to introduce stochasticity is to apply reparametrization trick.
3. Slide 23 – performance of the model is limited by how good model of environment is.
4. Slide 29 – $p(u_t|x_t)$ is some behavioral policy for collecting data.

Table of Contents

Next steps

Plan for the week

- Quiz N3
- Home assignment N3 will be issued after 24 Nov.

Reading

Lectures 12-14 of CS294.
Please, post your questions about lectures in google doc:
`https://goo.gl/qN6jmJ`