Lecture 4: The Cross Entropy Method and Model-Predictive Control

Plan for Today

Learning objectives At the end of this lecture, you willbe able to

1. use a known model of an environment to select actions that maximize the sum of rewards.
2. identify limitations with model-predictive control.
3. optimize potentially-stochastic functions of low-dimensional inputs x (cross entropy method).
4. identify limitations with the cross entropy method.

1 Intro

On the first day of class, we introduced a few different mental models for thinking about the RL problem: search and optimization. Today we're going to return to the optimization perspective on RL.

The RL problem is a problem of optimizing an unknown function $R(\tau) = \sum_t \gamma^t r(s_t, a_t)$. This function is unknown and stochastic. We don't know it's derivatives. Note that this looks like a bandit problem, but where the "arm" is replaced by a trajectory. This is one way to do the reduction I alluded to at the end of last lecture.

In the first half of today's class, we'll will introduce a generic optimization method (the *cross entropy method*) for optimizing unknown functions $f(x)$. This discussion of optimization, while not directly related to RL, reveals many of the salient features of the RL problem. In fact, we'll show next week that the procedure introduced today is exactly an RL algorithm!

In the second half of class, we'll talk about how we can apply this optimization method to control problems. This will be a special case of the RL problem where the dynamics are known.

Where are we going?

1. Does time matter? [last time] If not, use a bandit method
2. Do you know the model of your system? [today] If so, try model-predictive control.
3. Do you have expert data? [next time] If so, try imitation learning.
4. If all else fails, try RL!

2 From bandits to optimization

Review of bandits. Rewards $r(a)$ are unknown, we select an action a at each round and observe the reward, and then select the actions to use in the next round. Bandits are a special case of an MDP where the action you take doesn't affect the next state you start at.

The key object was $\hat{r}(a)$, our estimate of the rewards associated with each arm. We discussed how this can be estimated by simply keeping track of the rewards each time you've taken action a and then averaging them together.

The key challenge was deciding how to balance exploration vs exploitation. If you always take the action with the highest reward seen so far (the *greedy* strategy), you might get stuck in local optima. One way to avoid this is to always select a random action with some probability (the *ϵ-greedy* strategy); this avoids local optima, but never stops exploring; in continues to take bad actions even after you've determined that they cannot be the best. The *Boltzmann/SoftMax* strategy samples actions in proportion to their observed rewards. This method avoids local optima and avoids sampling the truly bad actions, but never stops exploring. The *upper confidence bound* (UCB) strategy effectively selects whichever action has the highest upper confidence interval. The size of this confidence interval shrinks each time you take that action. After a large enough number of samples, you will converge to always sampling the best action.

One way to think about the bandit problem is as an optimization routine. You want to optimize the function $\mathbb{E}[r(a)]$, and will propose a sequence of candidate maximizers a_t. Asymptotically, you'd like to arrive at the argmax. Whereas Tuesday's lecture looked at discrete a, today we will look at continuous a. The resulting algorithm will not only tell us how to solve some interesting control problems, but is secretly one of the most important RL algorithms.

3 Zero-Order Optimization and the Cross Entropy Method

3.1 Zero-Order Optimization: A Continuous Bandit

There's a field known as "zero-order" optimization: solving

$$\max_x f(x) \tag{1}$$

when the function $f(x)$ is unknown, its gradients are also unknown, and it can be random (i.e., feeding the same input may give separate outputs). Today, we'll consider the setting where x are continuous.

Note that if x were discrete then we'd exactly recover the bandit problem. Said in other words, zero-order optimization methods typically focuses on continuous-action bandits.

We call this a "zero-order" method because it only uses information about the value of the function, not its derivatives. We call SGD a "first-order" method because it uses information about the function's value and its first derivative (gradients). We call Newton's method a "second-order" method because it makes use of the function's second derivative.

Inputs: Function $f(x)$, which you can evaluate. Evaluation is expensive so you want to minimize the number of queries. Function might be stochastic. You don't know what the math/code for $f(x)$ is; you don't know the gradients.

Output: x^*. You want $x^* \approx \arg\max_x \mathbb{E}[f(x)]$. I've included the expectation to handle the case where $f(x)$ is a stochastic function.

3.2 Examples

- Kriging – digging for gold in S. Africa.
- optimizing ratio of ingredients in chocolate chip cookies
- Protein design
- Designing airplane wings

3.3 The Cross Entropy Method

So, how are we going to optimize this function? We'll look at the **cross entropy method**.[1]

The method is going to work by maintaining a *distribution* $p(x)$ over potential candidates for what the argmax might be. We expect that this distribution will initially be close to uniform; as we get more and more samples of the function $f(x)$, it will become peaked around the argmax.

[1]There's disagreement in the field about whether to hyphenate the name.

We will use a Gaussian/Normal distribution, $p(x) = \mathcal{N}(\mu, \Sigma)$. We typically initialize $\mu = 0$ and $\Sigma = c \cdot I$ for some large c (e.g., 10, 100).

Q: What is the shape of μ and Σ

The method then proceeds as follows

The cross entropy method (CEM).
```
**for** $t = 0, 1, \cdots, T$ **do**
    Sample $x_t^{(0)}, \cdots, x_t^{(k)} \sim p(x)$
    Evaluate $f(x_t^{(i)})$ for $i = 0, \cdots, k$
    Choose samples $x_t^{(0)}$ with highest values of $f(\cdot)$
    Fit $p(x)$ to these samples: $\max_{\mu,\Sigma} \sum_{i \in \text{Best}} \log p(x_t^{(i)})$
**end for**
**return** $\mu$
```

3.4 When can you use CEM?

- Non-smooth function $f(x)$?
- Discrete x?
- x is a scalar? a vector? a matrix?
- Stochastic function $f(x)$?

3.5 Final remarks

- There's an excellent visualization of this algorithm at Ha [2].
- There are some tricks for making this more stable and sample efficient. In practice, I'd recommend using the variant CMA-ES [3]. There are a couple of good Python implementations of this algorithm.
- Why aren't we using gradients $\nabla_x f(x)$?
- CEM is a *procedure*. How might you show that it is optimizing the correct objective?
- Claim: CEM is (almost) doing gradient descent! We'll prove this next week.

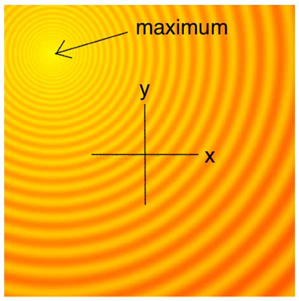

Figure 1: An example of a function to maximize/mimimize. From [2].

4 Model-predictive control

We're now going to use CEM to solve a control problem. We'll assume that you have a model/simulator of the environment. You can think of this model as a function $m(s_t, a_t) \to s_{t+1}$ or as a probability distribution $p(s_{t+1} \mid s_t, a_t)$. We assume that you know can query reward function $r(s_t, a_t)$.

4.1 The optimization problem

Let's say that the agent is currently at state s_0. The problem of figuring out which action to take next can be viewed as an optimization problem:

$$\max_{a_0} \max_{a_{1:T}} \mathbb{E}_{s_{t+1} \sim p(s_{t+1} | s_t, a_t)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right] \tag{2}$$

$$= \max_{a_{0:T}} \underbrace{\mathbb{E}_{s_{t+1} \sim p(s_{t+1} | s_t, a_t)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right]}_{\triangleq f(a_{0:T})}. \tag{3}$$

Note that there are two maximizations in the first line. This means that we want to choose the action now such that, if we act optimally in the future, we'll get a high cumulative reward. The second line notes that these two maximizations can be combined.

Q: What is the dimension of $a_{0:T}$?

4.2 Solving this optimization problem

To solve this optimization problem, we are going to use the cross entropy method:

$$a_{0:T} \leftarrow \arg\max f(a_{0:T}) \approx \text{CEM}(f). \tag{4}$$

4.3 Using these actions

We now have a long sequence of actions, telling us what to do starting at our current state. MPC says that we should take the first action a_0. At this point we have a choice. We could take the next action a_1 that we've already found. Or we could solve the optimization problem again, this time solving for $a_{1:T+1}$. MPC says that we should do the latter, *replanning* at each time step. One trick that we can employ to accelerate optimization is to use the solutions for $a_{1:T}$ from time $t = 0$ to *initialize* μ for time $t = 1$.

Note that the planning horizon is changing over time. Initially we make a plan for time steps $(0, T)$, then we make a plan for time steps $(1, T + 1)$, and so forth. For this reason, MPC is sometimes referred to as *receding horizon control*.

Q: Why is replanning after each time step a good idea?

Q: Why aren't we trying to solve this optimization problem via gradient descent?

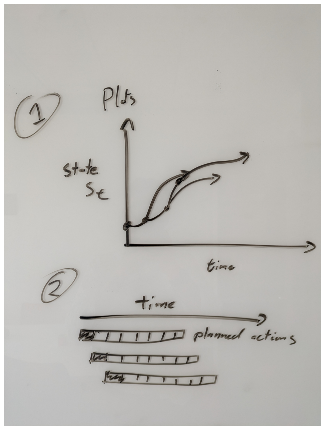

Figure 2: Visualizations of MPC

4.4 Choosing the horizon, T

The optimization problem that we've written down is just over the next T time steps. Typically T is small, perhaps 10 – 100 steps. This may often be much smaller than the total number of steps in an episode.

There's a tradeoff in choosing T. If T is large, then the optimization problem is expensive (many function evaluations) and noisy. If T is small, then we might find a myopic solution, one that maximizes the short-term rewards but ignores long-term rewards.

4.5 Advanced: Open-loop vs closed-loop control

There's a subtle issue with model-predictive control. Above, we said that we were trying to select the action now assuming that we would act optimally in the future. To clarify, we were making the assumption that we were pre-committing to the optimal sequence of actions. The keyword here is "precommit" – we are selecting the actions $a_{1:T}$ *before* knowing what the states $s_{1:T-1}$ will be. In stochastic settings, we can't know for certainty what these states will be. It may be that the optimal actions depend on the states visited.

In RL and control, we will use the term *open-loop* to refer to a strategy/policy that ignores the current observation, and blindly executes the next action. This is in contrast to a *closed-loop* (or reactive) policy, which decides the action based on the current observation. MPC is learning a closed-loop controller, but under the assumption that we will use the optimal open-loop controller in the future.

Q: Can you think of settings where MPC might result in a suboptimal policy?

4.6 What if you don't know the model?

Learn it! Given a dataset $\mathcal{D} = \{(s_t, a_t, s_{t+1})\}$, fit a model $p_\psi(s_{t+1} \mid s_t, a_t)$ using maximum likelihood:

$$\max_\psi \sum_{(s_t, a_t, s_{t+1}) \sim \mathcal{D}} \log p_\psi(s_{t+1} \mid s_t, a_t). \tag{5}$$

Where do you get the data? Start with a random model, do MPC, collect that data, fit the model, and repeat. PETS [1] is a good reference for how to get this to work well in practice.

4.7 Challenges with MPC

- The search space is high-dimensional, $|\mathcal{A}| \times T$.
- Solving long horizon tasks is challenging. When we truncate at $T \ll 1/(1-\gamma)$, we ignore the quality of the state where we end up at time s_T.
- Requires an accurate model.

References

[1] Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

[2] Ha, D. (2017). A visual guide to evolution strategies. *blog.otoro.net*.

[3] Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.