

前面关于模型提升的思想也只是一些直观的想法，并没有严格的理论证明，所以随着学习不断推进，模型效果下降的情况是可能发生的。为了解决这个问题，AlphaZero 中还包含了第三个模块，这个模块的存在使模型的单调提升更有保证。

13.1.5 模型的对决

为了让模型能够拥有接近单调提升的性质，AlphaZero 中包含了模型之间对决的环节。俗话说“是骡子是马，拉出来遛遛”，对于如此复杂的问题，最好的检验方法就是在真实对弈中看效果。于是每完成一轮迭代，新的模型都要和旧的模型进行一定轮棋局的对弈。新的模型只有获得更多的胜利（超过 55% 的胜率）才被认为超过了旧模型，才能替换旧的模型，否则就抛弃这个新模型，重新进行新的训练。

虽然这个思想并不复杂，但其中包含了大量的计算，而且算法中的模型比较大，在对决时需要将其部署到多台机器上同时进行多局对弈，因此其实现需要工程上的考量。相对而言，这个方法在思想上比较简单，而 AlphaZero 中的自我博弈已经具备一定的使策略提升的功能，因此这里的设计可以简单一些。

以上就是 AlphaZero 的核心流程。它的三个核心模块：自我对弈、深层模型和新旧模型对决组成了完整的模型，为棋类游戏构建了一个新的框架范本。很多与围棋形式接近的棋类也可以应用这个框架实现对应的策略。在论文中作者也介绍了 AlphaZero 在国际象棋和日本将棋上的应用，很多人也尝试在其他棋类上应用，并且收到了很好的效果，这也证明了通用学习框架对人工智能发展起到的作用。更多关于模型训练的细节请读者参阅相关论文，这里不再赘述。

13.2 iLQR

iLQR 算法的全称为 iterative Linear Quadratic Regulator，是一种 Model-based 的规划方法，它使用确定的策略函数和状态转移函数建立起序列的全过程。由于这部分知识最早与控制论相关的内容有关，所以它的变量表示法和前面章节中强化学习的表示方法有所不同，它的变量的对应方式如下所示。

观测状态： $s \rightarrow x$ 。

行动： $a \rightarrow u$ 。

回报（损失）函数： $r(s, a) \rightarrow c(x, u)$ 。

状态转移函数： $s_{t+1} \sim p(s_{t+1}|s_t, a_t) \rightarrow x_{t+1} = f(x_t, u_t)$ 。

在控制论中，损失函数 $c(\mathbf{x}, \mathbf{u})$ 表示执行行动后产生的代价。因此强化学习的目标——“最大化长期回报”变成了“最小化长期损失”。它要解决的问题形式如下所示：

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(\mathbf{x}_2, \mathbf{u}_2) + \dots + c(\mathbf{x}_T, \mathbf{u}_T)$$

因为状态转移函数已知，所以目标函数也可以写作

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)), \mathbf{u}_T)$$

第 6 章介绍了状态转移概率已知的问题。那么这个问题和第 6 章介绍的问题有什么区别呢？第 6 章介绍的问题在状态和行动空间上都是离散的，而本章介绍的问题在状态和行动空间上都是连续的，而且状态转移模型是一个确定的模型。在现实中我们也会经常遇到这样的问题，例如操纵一个机器人在一个环境中实现某个任务，由于任务的全部物理过程是已知的，所以状态转移函数是确定的；同时，我们也可以构建确定的策略函数，这样完整的交互序列可以由公式表示。可以看出这个问题和第 6 章提到的蛇棋问题的形式不同，当然二者的算法也不一样。

13.2.1 线性模型的求解法

我们先从一个比较简单的问题入手，介绍在计算过程上相对简单一些的算法 LQR。假设损失函数是一个二次函数，而状态转移函数是一个一次函数，那么两个公式可以写作

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}$$

这样定义后，目标函数中唯一未知的变量就是行动 \mathbf{u}_t ，我们可以通过求解公式的极值找到最优的行动。为了让问题看上去不那么复杂，我们先不考虑较长序列的问题，而是把行动轨迹限定在两个时间段内，那么整个序列为 $(\mathbf{x}_1, \mathbf{u}_1, \mathbf{x}_2, \mathbf{u}_2)$ ，我们要推导出当 \mathbf{x}_1 已知时使损失函数最小化的行动，即求出公式的最优解

$$\min_{\mathbf{u}_1, \mathbf{u}_2} Q = c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2)$$

如果可以从这个问题中找到规律，就可以把求解公式推广到更长的行动序列上。将公式完全展开，可以得到

$$\begin{aligned} Q &= \frac{1}{2} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^T \mathbf{C} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix}^T \mathbf{C} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix}^T \mathbf{c} \\ &= \frac{1}{2} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^T \mathbf{C} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^T \mathbf{c} + \frac{1}{2} \begin{bmatrix} f(\mathbf{x}_1, \mathbf{u}_1) \\ \mathbf{u}_2 \end{bmatrix}^T \mathbf{C} \begin{bmatrix} f(\mathbf{x}_1, \mathbf{u}_1) \\ \mathbf{u}_2 \end{bmatrix} + \begin{bmatrix} f(\mathbf{x}_1, \mathbf{u}_1) \\ \mathbf{u}_2 \end{bmatrix}^T \mathbf{c} \end{aligned}$$

公式中的 \mathbf{C} 和 \mathbf{c} 可以展开为

$$\begin{aligned} \mathbf{C} &= \begin{bmatrix} C_{\mathbf{x}_T, \mathbf{x}_T}, C_{\mathbf{x}_T, \mathbf{u}_T} \\ C_{\mathbf{u}_T, \mathbf{x}_T}, C_{\mathbf{u}_T, \mathbf{u}_T} \end{bmatrix} \\ \mathbf{c} &= \begin{bmatrix} c_{\mathbf{x}_T} \\ c_{\mathbf{u}_T} \end{bmatrix} \end{aligned}$$

其中矩阵 \mathbf{C} 是一个对称矩阵，也就是说 $C_{\mathbf{x}_T, \mathbf{u}_T} = C_{\mathbf{u}_T, \mathbf{x}_T}$ 。 \mathbf{F} 也可以展开为

$$\mathbf{F} = [F_{\mathbf{x}_T}, F_{\mathbf{u}_T}]$$

公式待求的是 $\mathbf{u}_1, \mathbf{u}_2$ 的值，对其分别进行求导，并使导数为 0，就可以得到函数取得极值时两个变量的解，首先对 \mathbf{u}_2 求导可以得到

$$\nabla_{\mathbf{u}_2} Q = C_{\mathbf{x}_T, \mathbf{u}_T} f(\mathbf{x}_1, \mathbf{u}_1) + C_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{u}_2 + c_{\mathbf{u}_T}^T$$

再对 \mathbf{u}_1 进行求导，可以得到

$$\nabla_{\mathbf{u}_1} Q = C_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{x}_1 + C_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{u}_1 + c_{\mathbf{u}_T}^T + F_{\mathbf{u}_T}^T (C_{\mathbf{x}_T, \mathbf{x}_T} f(\mathbf{x}_1, \mathbf{u}_1) + C_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{u}_2 + c_{\mathbf{x}_T}^T)$$

虽然我们得到了这两个变量的导数，但是它们的形式十分烦琐。两个行动的求解计算已经如此复杂，两个以上行动的计算应该会更复杂，而且每一时刻行动的导数计算公式都不相同，因此我们不能使用这样的方法对公式直接展开求解，而是需要使用一种更简洁且易操作的方法实现。这就是 LQR 方法要解决的问题。

LQR 方法包含反向计算部分和前向计算部分，其中反向计算用于确定计算每一个时刻行动的参数值，而前向计算则根据初始状态和反向计算的参数得到所有时刻的行动。反向计算比较复杂，它通过迭代的形式完成计算，每一个迭代中，我们要完成一个

时刻行动参数的计算，当这些参数计算完成后，我们就可以通过前向计算得到具体的行动值。迭代内的推导过程分为如下几个步骤。

(1) 根据 T 时刻的目标公式对 \mathbf{u} 求导，计算使函数最优的 \mathbf{u} 值。

(2) 用 \mathbf{x} 替换 \mathbf{u} ，得到关于 \mathbf{x} 的优化公式。

(3) 将公式转换成 $t - 1$ 时刻变量 \mathbf{x} 、 \mathbf{u} 的形式。

下面就来介绍具体过程。第一步，在 $T = 2$ 时刻通过最小化 $C(\mathbf{x}_2, \mathbf{u}_2)$ 求出 \mathbf{u}_2 ，根据公式，可以得到

$$Q(\mathbf{x}_2, \mathbf{u}_2) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix}^T \mathbf{C} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{u}_2 \end{bmatrix}^T \mathbf{c}$$

对其进行求导，可以得到

$$\nabla_{\mathbf{u}_2} Q(\mathbf{x}_2, \mathbf{u}_2) = \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{x}_2 + \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{u}_2 + \mathbf{c}_{\mathbf{u}_T}^T$$

令函数的导数为 0，可以得到关于 \mathbf{u}_2 的公式：

$$\mathbf{u}_2 = -\mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T}^{-1} (\mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T}^T \mathbf{x}_2 + \mathbf{c}_{\mathbf{u}_T}^T)$$

为了简化表达式，用变量 \mathbf{K}_2 和 \mathbf{k}_2 代替公式中的表达式，可以得到

$$\begin{aligned} \mathbf{K}_2 &= -\mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T}^{-1} \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T}^T \\ \mathbf{k}_2 &= -\mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T}^{-1} \mathbf{c}_{\mathbf{u}_T}^T \end{aligned}$$

于是 \mathbf{u}_2 的值可以表示为关于 \mathbf{x}_2 的公式：

$$\mathbf{u}_2 = \mathbf{K}_2 \mathbf{x}_2 + \mathbf{k}_2$$

这样第一步就完成了。第二步的目标是得到关于 \mathbf{x}_2 的优化公式，将上面关于 \mathbf{u}_2 的公式代入损失函数中，可以得到

$$V(\mathbf{x}_2) = \min Q(\mathbf{x}_2, \mathbf{u}_2) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{K}_2 \mathbf{x}_2 + \mathbf{k}_2 \end{bmatrix}^T \mathbf{C} \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{K}_2 \mathbf{x}_2 + \mathbf{k}_2 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_2 \\ \mathbf{K}_2 \mathbf{x}_2 + \mathbf{k}_2 \end{bmatrix}^T \mathbf{c}$$

将其展开可以得到

$$\begin{aligned}
 &= \frac{1}{2} \mathbf{x}_2^\top \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} \mathbf{x}_2 + \frac{1}{2} \mathbf{x}_2^\top \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{K}_2 \mathbf{x}_2 + \frac{1}{2} \mathbf{x}_2^\top \mathbf{K}_2^\top \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_2 + \frac{1}{2} \mathbf{x}_2^\top \mathbf{K}_2^\top \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{K}_2 \mathbf{x}_2 \\
 &+ \mathbf{x}_2^\top \mathbf{K}_T^\top \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{k}_T + \frac{1}{2} \mathbf{x}_2^\top \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{k}_T + \mathbf{x}_2^\top \mathbf{c}_{\mathbf{x}_T} + \mathbf{x}_2^\top \mathbf{K}_2^\top \mathbf{c}_{\mathbf{u}_T} + \text{const}
 \end{aligned}$$

这个公式展开后也非常复杂，我们同样需要对公式中的变量进行替换，使用变量 \mathbf{V}_2 和 \mathbf{v}_2 ，可以得到

$$\begin{aligned}
 \mathbf{V}_2 &= \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{K}_2 + \mathbf{K}_2^\top \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} + \mathbf{K}_2^\top \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{K}_2 \\
 \mathbf{v}_2 &= \mathbf{c}_{\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{k}_2 + \mathbf{K}_2^\top \mathbf{C}_{\mathbf{u}_T} + \mathbf{K}_2^\top \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{k}_2
 \end{aligned}$$

公式又重新变成了十分简洁的形式：

$$V(\mathbf{x}_2) = \text{const} + \frac{1}{2} \mathbf{x}_2^\top \mathbf{V}_2 \mathbf{x}_2 + \mathbf{x}_2^\top \mathbf{v}_2$$

这样第二步就完成了。接下来是第三步，即将目标函数转换成以 $\mathbf{x}_1, \mathbf{u}_1$ 表示的形式，将原公式中第二时刻的损失函数替换成前面计算的结果，就可以得到下面的形式：

$$Q(\mathbf{x}_1, \mathbf{u}_1) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^\top \mathbf{C} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^\top \mathbf{c} + V(f(\mathbf{x}_1, \mathbf{u}_1))$$

可以看出公式等号右边包含三个项目，第一项和第二项是 $t = 1$ 时刻的行动损失，第三项是前面计算得到的 $V(\mathbf{x}_2)$ ，将上面公式的第二项完全展开，可以得到

$$\begin{aligned}
 V(f(\mathbf{x}_1, \mathbf{u}_1)) &= V(\mathbf{F} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} + \mathbf{f}) \frac{1}{2} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^\top \mathbf{F}^\top \mathbf{V}_2 \mathbf{F} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} \\
 &+ \frac{1}{2} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^\top \mathbf{F}^\top \mathbf{V}_2 \mathbf{f} + \frac{1}{2} \mathbf{f}^\top \mathbf{V}_2 \mathbf{F} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^\top \mathbf{F}^\top \mathbf{v}_2 + \text{const}
 \end{aligned}$$

展开后的第二项和原公式中的前两项存在可合并的同类项，这样就可以做进一步的变量替换和整理，得到新的公式

$$\mathbf{Q}_1 = \mathbf{C} + \mathbf{F}^\top \mathbf{V}_2 \mathbf{F}$$

$$\mathbf{q}_1 = \mathbf{c} + \mathbf{F}^\top \mathbf{V}_2 \mathbf{f} + \mathbf{F}^\top \mathbf{v}_2$$

$$Q(\mathbf{x}_1, \mathbf{u}_1) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^T Q_1 \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix} + \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_1 \end{bmatrix}^T \mathbf{q}_1$$

实际上，到这里我们已经完成了一轮迭代的推演。我们发现上面的公式变成了只包含 \mathbf{x}_1 和 \mathbf{u}_1 的公式，可以使用第一步的计算方法求出使公式最优的 \mathbf{u}_1 值：

$$\nabla_{\mathbf{u}_1} Q(\mathbf{x}_1, \mathbf{u}_1) = Q_{\mathbf{u}_T, \mathbf{u}_T}^T \mathbf{x}_1 + Q_{\mathbf{u}_T, \mathbf{u}_T}^T \mathbf{u}_1 + \mathbf{q}_{\mathbf{u}_T}^T = 0$$

$$\mathbf{u}_1 = \mathbf{K}_1 \mathbf{x}_1 + \mathbf{k}_1$$

$$\mathbf{K}_1 = -Q_{\mathbf{u}_T, \mathbf{u}_T}^{-1} Q_{\mathbf{u}_T, \mathbf{x}_T}$$

$$\mathbf{k}_1 = -Q_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{q}_{\mathbf{u}_T}$$

而此时 \mathbf{x}_1 已知，我们就可以直接得到 \mathbf{u}_1 的值，然后根据状态转移函数得到 \mathbf{x}_2 ，再利用参数 K_2 和 k_2 进行计算得到行动 \mathbf{u}_2 。这其实也就是前向计算的过程。理解了长度为 2 的序列，再去推导长度为 N 的序列也变得不那么复杂，完整的算法过程如下。

算法 LQR 反向计算算法

$$V_{T+1} = 0$$

$$v_{t+1} = 0$$

For $t = T$ to 1;

$$Q_t = C + F^T V_{t+1} F$$

$$q_t = c + F^T V_{t+1} f + F^T v_{t+1}$$

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

$$K_t = -Q_{tu_T, u_T}^{-1} Q_{tu_T, x_T}$$

$$k_t = -Q_{tu_T, u_T}^{-1} q_{tu_T}$$

$$u_t = \operatorname{argmin}_{u_t} Q(\mathbf{x}_t, \mathbf{u}_t) = K_t \mathbf{x}_t + k_t$$

$$V_t = Q_{tx_T, x_T} + Q_{tx_T, u_T} K_t + K_t^T Q_{tu_T, x_T} + K_t^T Q_{tu_T, u_T} K_t$$

$$v_t = q_{tx_T} + Q_{tx_T, u_T} k_t + K_t^T Q_{tu_T} + K_t^T Q_{tu_T, u_T} k_t$$

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

算法 LQR 前向计算算法

for $t = 1$ to T :

$$u_t = K_t \mathbf{x}_t + k_t$$

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

13.2.2 非线性模型的解法

本节我们介绍 iLQR 算法。13.2.1 节介绍的方法存在一个假设：损失函数为二次项函数，状态转移函数为一次函数。然而在很多真实场景下，我们得到的损失函数和状态转移函数并不一定满足上面的条件，例如它们并不是二次函数或者一次函数，而且问题的形式往往也会有一些不同的设定，例如最后一个状态并不需要给出行动，而是采用另一个损失函数直接计算最终状态的花费。由于 LQR 的推导相对复杂，在此我们可以借助前面的经验，对问题进行重新求解。

我们会对最终状态单独设定一个花费函数 $c_N(\mathbf{x})$ ，用于计算其花费，而其他时刻的花费仍然使用 $c(\mathbf{x}, \mathbf{u})$ 计算，这样 N 个时刻总体的花费就可以计算成

$$J(\mathbf{x}_0, \mathbf{U}) = \sum_{t=0}^{N-1} c(\mathbf{x}_t, \mathbf{u}_t) + c_N(\mathbf{x}_N)$$

其中 \mathbf{U} 代表序列中执行的全体行动 $\{\mathbf{u}_t\}_{t=1}^N$ 。为了使用 13.2.1 节的方法进行求解，我们需要将问题转化为类似的形式，这就要用到泰勒展开这个工具。对于任意的损失函数和状态转移函数，我们可以将其近似地展开，首先是损失函数：

$$\begin{aligned} c(\mathbf{x}_t, \mathbf{u}_t) &= c(\hat{\mathbf{x}}_t + \delta\mathbf{x}_t, \hat{\mathbf{u}}_t + \delta\mathbf{u}_t) \\ &\simeq c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^\top \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \\ &= c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \mathbf{c} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^\top \mathbf{C} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \end{aligned}$$

假设 $\mathbf{x}_t, \mathbf{u}_t$ 和 $\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t$ 足够接近，那么两者的梯度也可以近似，我们因此可以得到

$$\begin{aligned} \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\mathbf{x}_t, \mathbf{u}_t) &\simeq \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \\ \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\mathbf{x}_t, \mathbf{u}_t) &\simeq \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \end{aligned}$$

同理，我们可以对状态转移函数进行展开：

$$\begin{aligned} f(\mathbf{x}_t, \mathbf{u}_t) &= f(\hat{\mathbf{x}}_t + \delta\mathbf{x}_t, \hat{\mathbf{u}}_t + \delta\mathbf{u}_t) \\ &\simeq f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix}^\top \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta\mathbf{x}_t \\ \delta\mathbf{u}_t \end{bmatrix} \end{aligned}$$

$$= f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \mathbf{F} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}^\top \mathbf{F} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}$$

同损失函数类似，我们可以得到近似梯度的公式：

$$\nabla_{\delta \mathbf{x}_t, \delta \mathbf{u}_t} f(\mathbf{x}_t, \mathbf{u}_t) \simeq \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\nabla_{\delta \mathbf{x}_t, \delta \mathbf{u}_t}^2 f(\mathbf{x}_t, \mathbf{u}_t) \simeq \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

经过这样的转换，我们发现当函数的表示形式发生变化时，我们从直接求解 $\mathbf{x}_t, \mathbf{u}_t$ ，变成了已知序列 $\{\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t\}_{t=1}^N$ 的情况下求解 $\delta \mathbf{x}$ 和 $\delta \mathbf{u}$ 。转换后的求解并不困难，我们可以初始化一个行动序列 $\{\hat{\mathbf{u}}_t\}_{t=1}^N$ ，然后从初始状态 s_1 出发通过状态转移函数得到这样的序列。为了展示公式的推导过程，我们还是假设一个相对简单的序列 $\{\hat{\mathbf{x}}_1, \hat{\mathbf{u}}_1, \hat{\mathbf{x}}_2, \hat{\mathbf{u}}_2, \hat{\mathbf{x}}_3\}$ ，从这个序列上得到的公式很容易应用到解决复杂的问题上。

下面就来使用和 13.3.1 节类似的求解方法进行计算。首先对 \mathbf{x}_3 进行优化，可以直接得到

$$V(\mathbf{x}_3) = c_N(\mathbf{x}_3)$$

$$\nabla_{\mathbf{x}} V(\mathbf{x}_3) = \nabla_{\mathbf{x}} c_N(\mathbf{x}_3)$$

$$\nabla_{\mathbf{x}\mathbf{x}} V(\mathbf{x}_3) = \nabla_{\mathbf{x}\mathbf{x}} c_N(\mathbf{x}_3)$$

由于我们直接得到了关于 \mathbf{x}_3 的目标函数，所以直接进行 13.2.1 节中反向计算的第 3 步，将目标函数表示为 $\mathbf{x}_2, \mathbf{u}_2$ 的形式，此时的目标函数为

$$Q(\mathbf{x}_2, \mathbf{u}_2) = c(\hat{\mathbf{x}}_2 + \delta \mathbf{x}_2, \hat{\mathbf{u}}_2 + \delta \mathbf{u}_2) + V(f(\hat{\mathbf{x}}_2 + \delta \mathbf{x}_2, \hat{\mathbf{u}}_2 + \delta \mathbf{u}_2))$$

对其求导，将公式右边的第一项简写为 c ，第二项简写为 f ，可以得到上面函数的导数：

$$\nabla_{\delta \mathbf{x}_2} Q = \nabla_{\hat{\mathbf{x}}_2} c + \nabla_{\hat{\mathbf{x}}_3} V \nabla_{\delta \mathbf{x}_2} f$$

$$\nabla_{\delta \mathbf{u}_2} Q = \nabla_{\hat{\mathbf{u}}_2} c + \nabla_{\hat{\mathbf{x}}_3} V \nabla_{\delta \mathbf{u}_2} f$$

继续求二阶导，可以得到

$$\nabla_{\delta \mathbf{x}_2 \delta \mathbf{x}_2} Q = \nabla_{\delta \mathbf{x}_2} (\nabla_{\hat{\mathbf{x}}_2} c + \nabla_{\hat{\mathbf{x}}_3} V \nabla_{\delta \mathbf{x}_2} f)$$

$$\begin{aligned}
&= \nabla_{\hat{x}_2 \hat{x}_2} c + \nabla_{\delta x_2} f^\top \nabla_{\hat{x}_3 \hat{x}_3} V \nabla_{\delta x_2} f + \nabla_{\hat{x}_3} V \nabla_{\delta x_2 \delta x_2} f \\
\nabla_{\delta u_2 \delta x_2} Q &= \nabla_{\delta u_2} (\nabla_{\hat{x}_2} c + \nabla_{\hat{x}_3} V \nabla_{\delta x_2} f) \\
&= \nabla_{\hat{u}_2 \hat{x}_2} c + \nabla_{\delta u} f^\top \nabla_{\hat{x}_3 \hat{x}_3} V \nabla_{\delta x} f + \nabla_{\hat{x}_3} V \nabla_{\delta u_2 \delta x_2} f \\
\nabla_{\delta u_2 \delta u_2} Q &= \nabla_{\delta u_2} (\nabla_{\hat{u}_2} c + \nabla_{\hat{x}_3} V \nabla_{\delta u_2} f) \\
&= \nabla_{\hat{u}_2 \hat{u}_2} c + \nabla_{\delta u_2} f^\top \nabla_{\hat{x}_3 \hat{x}_3} V \nabla_{\delta u_2} f + \nabla_{\hat{x}_3} V \nabla_{\delta u_2 \delta u_2} f
\end{aligned}$$

知道了这些导数，就可以进行 13.2.1 节中反向计算的第 1 步，求出当前时刻的最优值 u ，这里我们实际求出的是最优的 δu ，令 $\nabla_{\delta u_2} Q = 0$ ，可以得到

$$\delta u_2 = -\nabla_{x_T, u_T} Q^{-1} (\nabla_{x_T, u_T} Q^\top \delta x_2 + \nabla_{u_T} Q^\top)$$

将公式进行整理，可以得到

$$\begin{aligned}
K_2 &= -\nabla_{x_T, u_T} Q^{-1} \nabla_{x_T, u_T} Q^\top \\
k_2 &= -\nabla_{x_T, u_T} Q^{-1} \nabla_{u_T} Q^\top \\
u_2 &= \hat{u}_2 + \delta u_2 = \hat{u}_2 + K_2 \delta x_2 + k_2
\end{aligned}$$

接下来就是 13.2.1 节反向计算的第 2 步，整理公式得到关于 x_2 的目标函数：

$$\begin{aligned}
V(x_2) &= Q(\hat{x}_2 + \delta x_2, \hat{u}_2 + K_2 \delta x_2 + k_2) \\
&= c(\hat{x}_2 + \delta x_2, \hat{u}_2 + K_2 \delta x_2 + k_2) + V(f(\hat{x}_2 + \delta x_2, \hat{u}_2 + K_2 \delta x_2 + k_2))
\end{aligned}$$

同样，我们可以求出公式的一阶导和二阶导供后续使用：

$$\begin{aligned}
\nabla_x V &= \nabla_x c + K_2^\top \nabla_u c + \nabla_x V + K_2^\top \nabla_u V \\
&= \nabla_x c + \nabla_x V + K_2^\top \nabla_u c + K_2^\top \nabla_u V \\
&= \nabla_x Q + K_2^\top \nabla_u Q \\
\nabla_{xx} V &= \nabla_x (\nabla_x Q + K_2^\top \nabla_u Q) \\
&= \nabla_{xx} Q + K_2^\top \nabla_{ux} Q
\end{aligned}$$

有些文献也喜欢把这两个公式进行一定的变换，将其变为

$$\begin{aligned}
\nabla_x V &= \nabla_x Q - K_2^\top \nabla_{uu} Q k_2 \\
\nabla_{xx} V &= \nabla_{xx} Q - K_2^\top \nabla_{uu} Q K_2
\end{aligned}$$

感兴趣的读者可以证明前后两组公式的等价性。这样，一个完整循环的计算内容就结束了。对于更长的循环，我们也可以使用这样的方式进行计算。需要注意的是，这一次我们得到的结果不是行动本身，而是最优行动对于当前行动的偏差。由于偏差采用梯度的形式表示，因此在更新时我们可以进行多轮迭代更新，这样算法就变成了 iterative LQR。

13.2.3 iLQR 的实现

下面我们介绍 iLQR 的实现，这里参考 <https://github.com/neka-nat/ilqr-gym> 中的代码实现。这个项目主要实现了两部分功能。

1. 基于连续行动的平衡车环境

Gym 中实现了经典的控制任务 CartPole，环境中有一个小车，这个车只能左右移动，车的中心有一根钉子，这个钉子钉住了一个棍子的一头，对应的界面如图 13-7 所示。我们的目标是控制小车使棍子竖立在空中并保持一定时间，如果可以做到这一点，那么 Agent 就可以获得分数；如果棍子失去平衡掉落，那么 Agent 将无法获得分数。整个环境由一些物理规则实现，每一时刻 Agent 将获得 4 个状态观测值：小车的位置、小车的速度、棍子对应的角度，以及棍子的角速度。Agent 根据状态值给出对小车施加的力的大小。

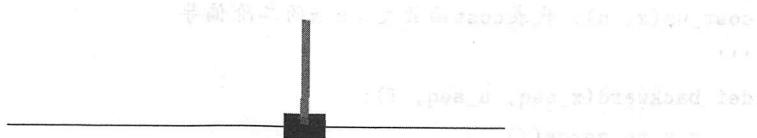


图 13-7 CartPole 界面图

2. iLQR 算法

这里我们直接将源代码展现出来，代码中同时包含对变量的介绍。首先是反向计算部分，这一部分比较复杂。

...

代码变量介绍

$v[t]$: 代表 $V(x[t])$

$v_x[t]$: 代表 $V(x[t])$ 对于 x 的导数

$v_{xx}[t]$: 代表 $V(x[t])$ 对于 x 的二阶导

f_x_t : 当前时刻 f 函数关于 x 的一阶偏导值

f_u_t : 当前时刻 f 函数关于 u 的一阶偏导值

q_x : 当前时刻 q 函数关于 x 的一阶偏导值

q_u : 当前时刻 q 函数关于 u 的一阶偏导值

q_{xx} : 当前时刻 q 函数关于 x 的二阶偏导值

q_{uu} : 当前时刻 q 函数关于 u 的二阶偏导值

q_{ux} : 当前时刻 q 函数关于 u, x 的二阶偏导值

kk, k : 代表计算最优 u 时的两个替代变量

代码函数介绍

$\text{costn}(x)$: 代表最终状态的花费

$\text{costn}_x(x)$: 代表最终状态花费对于 x 的导数

$\text{costn}_{xx}(x)$: 代表最终状态花费对于 x 的二阶导

$f_x(x, u)$: 代表 f 函数关于 x 的一阶偏导

$f_u(x, u)$: 代表 f 函数关于 u 的一阶偏导

$f_{xx}(x, u)$: 代表 f 函数关于 x 的二阶偏导

$f_{uu}(x, u)$: 代表 f 函数关于 u 的二阶偏导

$f_{ux}(x, u)$: 代表 f 函数关于 u, x 的二阶偏导

$\text{cost}_x(x, u)$: 代表 cost 函数关于 x 的一阶偏导

$\text{cost}_u(x, u)$: 代表 cost 函数关于 x 的一阶偏导

$\text{cost}_{xx}(x, u)$: 代表 cost 函数关于 x 的二阶偏导

$\text{cost}_{uu}(x, u)$: 代表 cost 函数关于 x 的二阶偏导

$\text{cost}_{ux}(x, u)$: 代表 cost 函数关于 u, x 的二阶偏导

...

```
def backward(x_seq, u_seq, T):
    v = np.zeros(T)
    v_x = np.zeros_like(v)
    v_xx = np.zeros([T, T])
    # 初始化最终状态的V函数
    v[-1] = costn(x_seq[-1])
    v_x[-1] = costn_x(x_seq[-1])
    v_xx[-1] = costn_xx(x_seq[-1])
    k_seq = []
    kk_seq = []
    for t in range(self.pred_time - 1, -1, -1):
        # step 1: 计算Q函数的对应部分
        f_x_t = self.f_x(x_seq[t], u_seq[t])
```

```

f_u_t = self.f_u(x_seq[t], u_seq[t]) q_u += f_u_t
q_x = self.cost_x(x_seq[t], u_seq[t]) + np.matmul(f_x_t.T,
    self.v_x[t + 1])
q_u = self.cost_u(x_seq[t], u_seq[t]) + np.matmul(f_u_t.T,
    self.v_x[t + 1])
q_xx = self.cost_xx(x_seq[t], u_seq[t]) + \
    np.matmul(np.matmul(f_x_t.T, self.v_xx[t + 1]), f_x_t) + \
    np.dot(self.v_x[t + 1], np.squeeze(self.f_xx(x_seq[t],
        u_seq[t])))
tmp = np.matmul(f_u_t.T, self.v_xx[t + 1])
q_uu = self.cost_uu(x_seq[t], u_seq[t]) + np.matmul(tmp, f_u_t) + \
    np.dot(self.v_x[t + 1], np.squeeze(self.f_uu(x_seq[t],
        u_seq[t])))
q_ux = self.cost_ux(x_seq[t], u_seq[t]) + np.matmul(tmp, f_x_t) + \
    np.dot(self.v_x[t + 1], np.squeeze(self.f_ux(x_seq[t],
        u_seq[t])))
# step 2: 计算最优u值
inv_q_uu = np.linalg.inv(q_uu)
k = -np.matmul(inv_q_uu, q_u)
kk = -np.matmul(inv_q_uu, q_ux)
# step 3: 计算V函数对应部分
dv = 0.5 * np.matmul(q_u, k)
self.v[t] += dv
self.v_x[t] = q_x - np.matmul(np.matmul(q_u, inv_q_uu), q_ux)
self.v_xx[t] = q_xx + np.matmul(q_ux.T, kk)
k_seq.append(k)
kk_seq.append(kk)
k_seq.reverse()
kk_seq.reverse()
return k_seq, kk_seq

```

其次是前向计算。我们可以将反向计算求解出的参数带入前向计算中，这样就得到了更新后的行动：

```

def forward(self, x_seq, u_seq, k_seq, kk_seq):
    x_seq_hat = np.array(x_seq)
    u_seq_hat = np.array(u_seq)
    for t in range(len(u_seq)):

```

```

control = k_seq[t] + np.matmul(kk_seq[t], (x_seq_hat[t] -
    x_seq[t]))
u_seq_hat[t] = np.clip(u_seq[t] + control, -self.umax, self.umax)
x_seq_hat[t + 1] = self.f(x_seq_hat[t], u_seq_hat[t])
return x_seq_hat, u_seq_hat

```

经过一段时间的训练，模型就可以控制小车将棍子稳定地竖立在空中。感兴趣的读者可以亲自尝试。

LQR 属于微分动态规划（Differential Dynamic Programming，简称 DDP）中的一种算法，这样我们就能把第 6 章中的动态规划法和本节介绍的方法统一起来，两者是针对不同空间的动态规划方法。关于 DDP 的内容还有很多，其中还会涉及很多很复杂的内容，由于篇幅的原因这里不做深入介绍。

13.3 总结

本章我们主要介绍了模型已知问题的求解方法，总结如下。

- (1) AlphaZero 采用了基于 MCTS 的自我学习方法进行策略学习。
- (2) iLQR 通过依时间迭代计算的方式对控制类问题进行规划求解。

13.4 参考资料

- [1] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587):484-489.
- [2] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550(7676):354-359.
- [3] Silver D, Hubert T, Schrittwieser J, et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm[J]. 2017.
- [4] Browne C B, Powley E, Whitehouse D, et al. A Survey of Monte Carlo Tree Search Methods[J]. IEEE Transactions on Computational Intelligence & Ai in Games, 2012, 4:1(1):1-43.
- [5] Tassa Y, Erez T, Todorov E. Synthesis and stabilization of complex behaviors through online trajectory optimization[C]// Ieee/rsj International Conference on Intelligent Robots and Systems. IEEE, 2012:4906-4913.