前言

当我们引入$v_\pi(s)$或者$q_\pi(s,a)$这样的符号时候，我们考虑的是离散的表格化数据。实际生活中，s可能数目多到难以列举，一种极端就是s是连续的。所以如何基于之前的现有理论，将其拓展，以处理大规模强化学习问题，是本lecture要讨论的问题。解决方法标题给出了：Value Function Approximation。

介绍

Large-Scale Reinforcement Learning Problems，e.g.

- Game of Go：10^{170} states
- Backgammon: 10^{20} states
- Automated Driving: continuous state space

我们如何扩展上两个lecture中的model free prediction和control？旧的方法要存储太多数据到内存，而且由于数据规模太大学习会非常缓慢。基本思想就是用连续函数来近似估计这些离散数据。

Value function approximation
$\hat{v}(s,\vec{w}) \approx v_\pi(s)$
$\hat{q}(s,a,\vec{w}) \approx q_\pi(s,a)$
表达近似函数，我们有很多方法，主要考虑常用的线性模型和神经网络。

增量方法（Incremental Methods)

Gradient Descent

- 我们定义目标函数$J(\vec{w})$来代表当前参数产生的误差之和
- 梯度(gradient):

$$\nabla_w J(\vec{w}) = \begin{pmatrix} \frac{\partial J(\vec{w})}{\partial w_1} \\ \dots \\ \frac{\partial J(\vec{w})}{\partial w_n} \end{pmatrix}$$

- 我们基于梯度更新$\vec{w}$:

$$\vec{w} \leftarrow \vec{w} - \frac{1}{2}\alpha \nabla_w J(\vec{w})$$

Stochastic Gradient Descent

$$J(\vec{w}) = E_\mu[(v_\pi(s) - \hat{v}(s,\vec{w}))^2]$$

我们在SGD做的事情是不是计算上式与所有状态相关的期望梯度，而是针对每一个状态来计算一个采样梯度：

$$\Delta\vec{w} = \alpha(v_\pi(s) - \hat{v}(s,\vec{w}))\nabla_w \hat{v}(s,\vec{w})$$

SGD一般来说比标准GD会获得更好的解，尤其是在训练数据庞大的时候。

Linear Function Approxiamation

- Feature Vector：一般来说，状态的表示本身就是一个特征向量
- Value Function：$\hat{v}(s,\vec{w}) = \sum_{i=1}^{n} x_i(s)w_i$
- Stochastic Gradient Descent: $\Delta\vec{w} = \alpha(v_\pi(s) - \hat{v}(s,\vec{w}))\overrightarrow{x(s)}$
- SGD converges on global optimum

Incremental Prediction Algorithms
善于思考的人肯定发现了一个问题，$v_\pi(s)$我们在求解时是假设为已知量的，可是我们并不像监督学习一样有这些“标记”，我们有的只是可以和环境交互而得到的reward。所以我们仿照前两个lecture的prediction方法，通过采样来得到一个近似替代：

- MC: We substitute G_t for $v_\pi(s)$
- TD(0): We substitute $R_{t+1} + \gamma\hat{v}(s_{t+1},\vec{w})$ for $v_\pi(s)$
- ...

有关上述替代的对于收敛性的影响以及两种$TD(\lambda)$，有如下结论：

- Monte-Carlo evaluation converges on a local optimum（Even when using non-linear function）
- Linear TD(0) converges (close) to global optimum
- Foward View and backward view linear $TD(\lambda)$ are equivalent
  注：这里等价体现在对于同一条轨迹，两种方法产生的$\sum_{i=1}^{T}\Delta\vec{w}$是相同的

Incremental Control Algorithms

- Policy evaluation：Approximate policy evaluation，$\hat{q}(.,.,\vec{w}) \approx q_\pi$
- Policy improvement: ϵ-Greedy policy improvement
  注：这里的近似策略评估仍然指在采到有限的数据后，更新值函数之后就立刻提升策略，例如MC中，仍然可以是一条轨迹一次提升的迭代周期。

批次方法（Batch Methods)

个人理解Batch Method是为了保证训练的收敛性和效率提出的一种思路，他需要维护一个数据库来存储智能体的历史经验，这和Incremental Methods中得到一条经验更新参数之后就丢掉是一种对立的做法，他的优势就在于可以重复利用环境的信息。

Least Squares Prediction

$$LS(\vec{w}) = E_D[(v_\pi(s) - \hat{v}(s,\vec{w}))^2]$$

我们想要最小化$LS(\vec{w})$,其中
$D = \left\{ < s_1, v_1^\pi(s_1) >, \quad < s_2, v_2^\pi(s_2) >, \quad \ldots, \quad < s_2, v_1^\pi(s_2) > \right\}$，$\pi$是当前策略

Stochastic Gradient Descent with Experience Replay
Repeat：

1. Sample state,value from $\boldsymbol{D}$
2. Apply stochastic gradient descent update

Converge to:

$$w^{\pi} = \arg\min_{w} LS(\vec{w})$$

DQN+Experience Replay+Fixed Q-targets
这是最常见的Double DQN，我们需要维护两个Q-Networks，其中一个代表old fixed network，以$\vec{\theta}_t^{'}$表示，另外一个则是online network，用$\vec{\theta}_t$表示。即：

$$Q-Target = R_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a} Q(s_{t+1}, a; \vec{\theta}_t); \vec{\theta}_t^{'})$$

具体细节参考原论文。
DQN的系列算法还有很多种，如：

- Averaged DQN
- Distributional DQN
- Noisy DQN
- ...

感兴趣都可以参阅有关paper。