7 Value Function Approximation Introduction

到目前为止，我们已经用一个查找表(数组???)来表示值函数，每个状态对应查找表的一条记录(或者说一个条目)也就是$V(s)$，或者每个(state-action)二元组都有一个记录，也就是$Q(s,a)$。

然而我们不想要为每个状态直接精确的存储或学习:Dynamics or reward model; Value ;State-action value; Policy。我们想要的是以更紧凑的形式来generalizes 整个 state 或者多个 state 和动作。

所以这种方法可能不能够很好地推广到具有非常大的状态和/或动作空间的问题，或者在其他情况下，相比于去求每个状态的收敛值(最优值)，我们可能更喜欢快速学习近似值。解决这个问题的一种流行方法是通过函数逼近:

$$v_\pi(s) \approx \hat{v}(s,\mathbf{w}) \quad or \quad q_\pi(s,a) \approx \hat{q}(s,a,\mathbf{w})$$

这里 w 通常被称为函数逼近器(function approximator)的参数或权重。我们列出了函数逼近器的几种常见选择:

- Linear combinations of features
- Neural networks
- Decision trees
- Nearest neighbors
- Fourier / wavelet bases
- 多项式

Generalization 的好处:①减少内存的使用②减少计算量③减少历史数据的存储或者说对数据的需求，因为能更快的找到最优 policy。

我们将进一步探讨两类流行的可微函数逼近器:线性特征表示和神经网络。下一节介绍了严格要求函数是可微的原因。

8 Linear Feature Representations

在线性函数表示中，我们使用特征向量来表示状态:

$$x(s) = [x_1(s)\ x_2(s)...x_n(s)]$$

然后，我们使用特征的线性组合来近似求解我们的值函数:

$$\widehat{v}(s,\mathbf{w}) = x(s)^T\mathbf{w} = \sum_{j=1}^{n} x_j(s)\mathbf{w}_j$$

我们把(二次)目标(也称为损失)函数定义为:

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(v_\pi(s) - \widehat{v}(s,\mathbf{w}))^2\right]$$

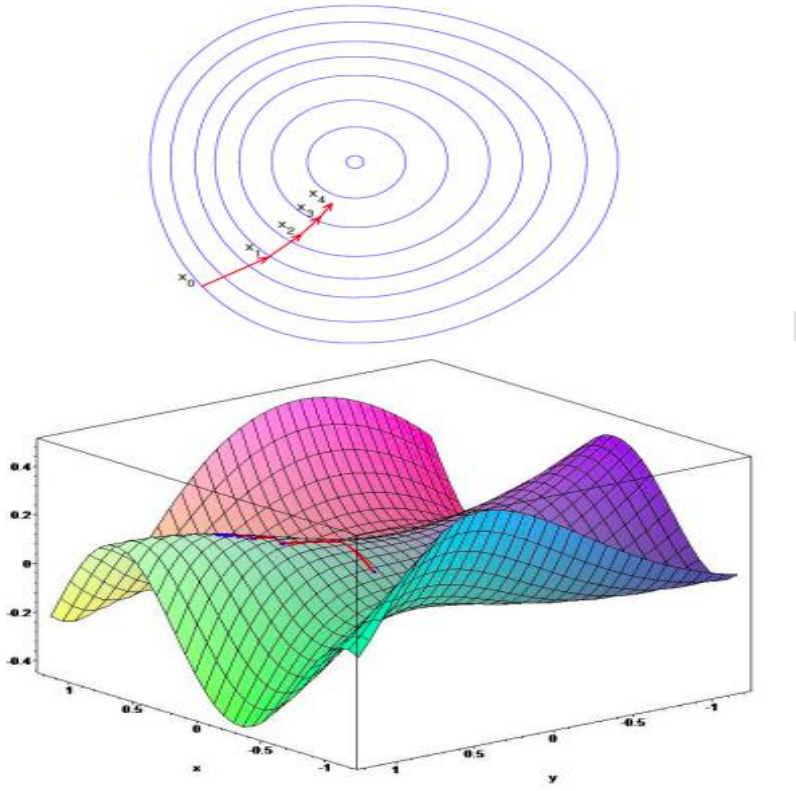

图 1:梯度下降的可视化。我们希望达到目标(损失)函数最小化的中心点。我们通过跟随红色箭头这样做，红色箭头指向我们评估的梯度的负方向

8.1 Gradient descent

最小化上述目标(损失)函数的一种常见技术被称为 ***Gradient descent***。图 1 提供了一个可视化的说明:我们从某个特定的点x_0开始，对应于我们的参数 w 的某个初始值；然后我们估计x_0处的梯度，它告诉我们目标(损失)函数中最陡(正增长)的方向；为了最小化我们的目标函数，我们沿着梯度向量的负方向走一步，到达x_1；重复这个过程，直到达到我们预先设置的收敛标准。

数学上，这可以概括为:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \left[\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_1} \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_2} \cdots \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}_n}\right] \quad \text{compute the gradient}$$

$$\Delta \mathbf{w} = -\frac{1}{2}\alpha \nabla_{\mathbf{w}} J(\mathbf{w}) \quad \text{compute an update step using gradient descent}$$

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w} \quad \text{take a step towards the local minimum}$$

8.2 Stochastic gradient descent

实际上，梯度下降并不是一个有效的样本优化器，随机梯度下降(SGD)被更频繁地使用。尽管最初的 SGD 算法是指使用单个样本来更新权重，但由于向量化(译者:其实应该是矩阵运算???)的便利性，人们通常也将基于小批次样本上的梯度下降称为 SGD。在(小批次)SGD 中，我们对过去的历史数据进行小批次采样，并用于计算我们基于小批次上的目标(损失)函数，并使用基于小批次的梯度下降更新我们的参数。现在让我们回到之前讲座中讨论的几种算法，看看如何将值函数近似法结合起来。

8.3 Monte Carlo with linear VFA

算法 1 是对 Fisrt-Visit Monte Carlo Policy Evaluation 的修改，我们用 linear VFA 代替了我们的值函数。我们还注意到，虽然我们的回报$Return(s_t)$是一个无偏估计，但它通常是非常不准(it is often very noisy)

Algorithm 1 Monte Carlo Linear Value Function Approximation for Policy Evaluation

Initialize $\mathbf{w} = 0$, $Returns(s) = 0\ \forall s$, $k = 1$
loop
  Sample k-th episode $(s_{k1}, a_{k1}, r_{k1}, s_{k2}, \ldots, s_k, L_k)$ given π
  for $t = 1, \ldots, L_k$ **do**
    if first visit to (s) in episode k **then**
      Append $\sum_{j=t}^{L_k} r_{kj}$ to $Return(s_t)$
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha(Return(s_t) - \hat{v}(s_t, \mathbf{w}))x(s_t)$
  $k = k + 1$

8.4 Temporal Difference(TD(0)) with linear VFA

回想一下，在表格设置中，我们通过自举(bootstrapping)和采样来近似$V^{\pi}(s)$，并通过以下方式更新$V^{\pi}(s)$:

$$V^{\pi}(s) = V^{\pi}(s) + \alpha(r + \gamma V^{\pi}(s') - V^{\pi}(s))$$

其中$r + \gamma V^{\pi}(s')$代表我们的目标(G,goal,回报是 return 所以这里翻译为宝

藏?)。在这里，我们使用同一个 VFA 来估计我们的目标和价值。在后面的章节中，我们将考虑使用不同 VFAs 的技术(例如 control setting)。

使用 VFAs，我们用$\widehat{V}^{\pi}$来替换V^{π},那么我们的方程就变成了:

$$\begin{aligned}\hat{V}^{\pi}(s,\mathbf{w}) &= \hat{V}^{\pi}(s,\mathbf{w}) + \alpha(r + \gamma\hat{V}^{\pi}(s',\mathbf{w}) - \hat{V}^{\pi}(s,\mathbf{w}))\nabla_{\mathbf{w}}\hat{V}^{\pi}(s,\mathbf{w}) \\ &= \hat{V}^{\pi}(s,\mathbf{w}) + \alpha(r + \gamma\hat{V}^{\pi}(s',\mathbf{w}) - \hat{V}^{\pi}(s,\mathbf{w}))\mathbf{x}(s)\end{aligned}$$

在 value function approximation(值函数逼近,值函数近似?)中，尽管我们的目标是有偏的(biased)并且是逼近V^{π}的估计值，但 linear TD(0)仍然会收敛(接近)到全局最优值。我们现在将证明这一论断。

8.5 Convergence Guarantees for Linear VFA for Policy Evaluation

我们将一个确定的 policy π的线性 VFA 的相对于真实值的均方误差定义为:

$$MSV\ E(\mathbf{w}) = \sum \mathbf{d}(s)(\mathbf{v}^{\pi}(s) - \hat{\mathbf{v}}^{\pi}(s,\mathbf{w}))^2$$

其中 d(s)是基于真实决策过程的 policy π的状态收敛后的真实分布(stationary distribution)，$\hat{\mathbf{v}}^{\pi}(s,\mathbf{w}) = \mathbf{x}(s)^T\mathbf{w}$是我们的 linear VFA。

定理 8.1. Monte Carlo policy evaluation with VFA converges to the weights $\mathbf{w}_{MC}$ with minimum mean squared error.

$$MSV\ E(\mathbf{w}_{MC}) = \min_{\mathbf{w}} \sum_{s\in\mathbf{s}} \mathbf{d}(s)(\mathbf{v}^{\pi}(s) - \hat{\mathbf{v}}^{\pi}(s,\mathbf{w}))^2$$

定理 8.2. TD(0) policy evaluation with VFA converges to the weights $\mathbf{w}_{TD}$ which is within a constant factor of the minimum mean squared error.

$$MSV\ E(\mathbf{w}_{MC}) = \frac{1}{1-\gamma}\min_{\mathbf{w}} \sum_{s\in\mathbf{s}} \mathbf{d}(s)(\mathbf{v}^{\pi}(s) - \hat{\mathbf{v}}^{\pi}(s,\mathbf{w}))^2$$

我们省略了这里的证明，并鼓励感兴趣的读者看[Tsitsiklis and Van Roy. An Analysis of Temporal-Di?erence Learning with Function Approximation]并进行更深入的讨论。

Algorithm	Tabular	Linear VFA	Nonlinear VFA
Monte-Carlo Control	Y	Y	N
SARSA	Y	Y	N
Q-Learning	Y	N	N

Table 1: Summary of convergence of Control Methods with VFA. (Yes) means the result chatters around near-optimal value function.

8.6 Control using VFA

与 VfA 相似，我们也可以对 action-value 使用函数逼近器(function approximators)。也就是说，我们让$\widehat{q}(s,a,\mathbf{w}) \approx q_\pi(s,a)$。然后，然后，我们可以通过使用$\widehat{q}(s,a,\mathbf{w})$近似进行 policy evaluation，然后通过$\in$-greedy policy improvement 来进行 policy improvement。接下来让我们用数学方法具体的写出这个。

首先，我们定义我们的目标损失函数$J(\mathbf{w})$为：

$$J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(s,a) - \widehat{q}^\pi(s,a,\mathbf{w}))^2\right]$$

与我们之前所做的类似，我们可以使用梯度下降或随机梯度下降来最小化目标损失函数。例如，对于 linear action-value function approximator(线性动作值函数逼近器)，这可以概括为：

$x(s,a) = [x_1(s,a)\ x_2(s,a)...x_n(s,a) \qquad state-action\ value\ features]$

$\widehat{q}(s,a,\mathbf{w}) = x(s,a)\mathbf{w} \qquad represent\ state-action\ value\ as\ linear\ combinations\ of\ features$

$J(\mathbf{w}) = \mathbb{E}_\pi\left[(q_\pi(s,a) - \widehat{q}^\pi(s,a,\mathbf{w}))^2\right] \qquad define\ objective\ function$

$$\begin{aligned}
-\frac{1}{2}\nabla_{\mathbf{w}} J(\mathbf{w}) &= \mathbb{E}_\pi[(q_\pi(s,a) - \hat{q}^\pi(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\hat{q}^\pi(s,a,\mathbf{w})] && \text{compute the gradient}\\
&= (q_\pi(s,a) - \hat{q}^\pi(s,a,\mathbf{w}))x(s,a)\\
\Delta\mathbf{w} &= -\frac{1}{2}\alpha\nabla_{\mathbf{w}} J(\mathbf{w}) && \text{compute an update step using gradient descent}\\
&= \alpha(q_\pi(s,a) - \hat{q}^\pi(s,a,\mathbf{w}))x(s,a)\\
\mathbf{w} &\leftarrow \mathbf{w} + \Delta\mathbf{w} && \text{take a step towards the local minimum}
\end{aligned}$$

对于蒙特卡罗方法，我们用回报G_t代替我们的目标$q_\pi(s,a)$。也就是说，我们的更新变成了：

$$\Delta\mathbf{w} = \alpha(G_T - \widehat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\widehat{q}(s,a,\mathbf{w})$$

对于 SARSA，我们使用 TD target 替换我们的目标$q_\pi(s,a)$：

$$\Delta\mathbf{w} = \alpha(r + \gamma\widehat{q}(s',a',\mathbf{w}) - \widehat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\widehat{q}(s,a,\mathbf{w})$$

对于 Q-Learning,我们用 max TD target 来替换我们的目标$q_\pi(s,a)$：

$$\Delta\mathbf{w} = \alpha(r + \gamma\max_{a'}\widehat{q}(s',a',\mathbf{w}) - \widehat{q}(s,a,\mathbf{w}))\nabla_{\mathbf{w}}\widehat{q}(s,a,\mathbf{w})$$

我们注意到，因为我们使用 value function approximations(可以是展开式)来

执行 Bellman backup，所以不能保证收敛性。我们建议用户寻找 Baird's Counterexample 来获得更具体的说明。我们在上诉表 1 中总结了收缩保证。

9 Neural Networks

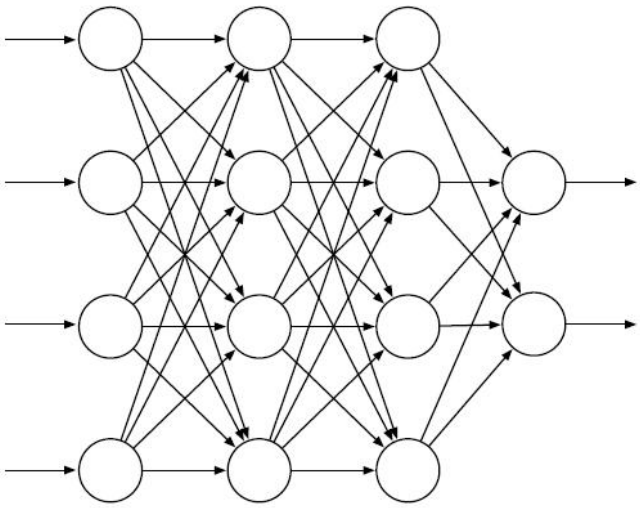

Figure 2: A generic feedforward neural network with four input units, two output units, and two hidden layers.

虽然 linear VFA 通常在给定正确的特性集和的情况下工作良好，但即时是人工制造这样的特性集和也是非常困难的。神经网络提供了一个更加丰富的函数逼近器类(function approximation class)，它能够直接从状态出发，而不需要明确的特征描述。

Figure 2 显示了一个通用前馈神经网络。图中的网络有一个由两个输出单元组成的输出层，一个有四个输入单元的输入层和两个隐藏层:隐藏层的意思是既不是输入层也不是输出层。每个环节都有一个实值权重。这些单元通常是 semi-linear，这意味着它们计算输入信号的加权和，然后对结果应用非线性函数。这通常被称为激活功能。在作业 2 中，我们研究了单隐藏层的神经网络如何具有"universal approximation"性质，经过经验和理论表明，通过构造多个隐藏层的神经网络逼近我们期望的复杂函数更为容易。