

Published as a conference paper at ICLR 2018

# EIGENOPTION DISCOVERY THROUGH THE DEEP SUCCESSOR REPRESENTATION

Marlos C. Machado<sup>1\*</sup>, Clemens Rosenbaum<sup>2</sup>, Xiaoxiao Guo<sup>3</sup>  
Miao Liu<sup>3</sup>, Gerald Tesauro<sup>3</sup>, Murray Campbell<sup>3</sup>

<sup>1</sup> University of Alberta, Edmonton, AB, Canada

<sup>2</sup> University of Massachusetts, Amherst, MA, USA

<sup>3</sup> IBM Research, Yorktown Heights, NY, USA

## 【强化学习 71】Successor Representation



张楚琦

清华大学 交叉信息院博士在读

9 人赞同了该文章

### 原文传送门

Machado, Marlos C., et al. "Eigenoption discovery through the deep successor representation." arXiv preprint arXiv:1710.11089 (2017).

### 特色

之前讲了两篇学习状态表示（representation）的文章，一篇讲PVF（【强化学习 67】Proto-value Function），一篇讲DeepMDP（【强化学习 68】DeepMDP）。它们都可以看做是对于环境模型的压缩，其中前者只考虑环境本身的动力学特性（dynamics），后者不仅考虑环境的动力学特性，还考虑环境的奖励函数；因此，前者可以用于meta learning，而后者则更贴近要学习的任务，同时理论上能有更直接的关于学到策略性能的bound。

这里也是想学习一种状态表示，并且这种状态表示只反映环境的特性（即dynamics），不涉及具体的任务（即reward）。这样的表示有以下优势：

- 当环境上的任务变化的时候，该方法学到的状态表示依然有用，即适用于meta learning的设定；
- 当任务给出的奖励稀疏或者具有误导性的时候，该表示能够探索到环境本身的特性，并且利用学习到的环境性质来指导智能体探索。

本文就是想从对于MDP的随机采样（stochastic sampling）的样本中学习这样一种状态表示，进而提取出环境的性质做分层强化学习，详细的介绍见下一部分。

### 过程

#### 1. 总体思路

本文先通过在MDP上的随机采样来学习一个表示（successor representation），并且从该表示下得到一个关于环境的刻画（一个状态空间上的diffusion matrix），从而得到与环境有关的辅助奖励函数（eigenpurpose），最大化该奖励函数能够形成相应的更为抽象的行动（eigenoption），进而产生一个基于option的分层强化学习算法（参考【强化学习算法 20】Option-Critic）。

#### 2. 回顾PVF

PVF (proto-value function) 方法使用不同状态之间的转移关系构建一个状态之间的转移矩阵  $\mathbf{w}$ ，使用该转移矩阵构建随机游走的一步转移概率矩阵  $\mathbf{P}$ （或者相应的diffusion model  $\mathbf{L}$ ），对该矩阵进行特征值分解，其最大的  $k$  个特征值对应的特征向量可以作为其价值函数的基向量。注，原文用的  $\mathbf{P}$ ，为了和这篇文章统一，就写作  $\mathbf{P}$ 。

对于一个特征向量  $\mathbf{e}$ ，它的每一位表示一个状态的某种性质的数值。回顾PVF里面的两个小房间的例子：比如某特征向量数值大表示该状态位于第一个小房间，否则位于第二个小房间；再比如某特征向量数值大的表示该状态位于房间内距离goal较远的一侧。

分层强化学习里面的option相当于一个达到特定目的的一连串动作或者子策略，有一些工作里面option的目标是人为定义的，当然最好的情况是能够自动探索并且发现option。PVF里面得到的特征向量提供了一个自动探索option的方法。刚刚说到一个特征向量中各个状态的数值代表着它对于环境某方面性质的刻画，因此自然想到对于一个特征向量，定义一个option，使得该option朝着该特征向量数值较大/小的方向走。这样得到的option也叫eigenoption。eigenoption的定义可以通过定义相应的eigenpurpose (reward) 得到，最大化该奖励得到的子策略就是eigenpurpose，由此，容易想到定义它为

$$r(s, s') = \mathbf{e}(s') - \mathbf{e}(s)$$

option可以避免反复无效的探索，比如随机地一会往左、一会往右，探索很长时间还是几乎在原地，而使用option可以比如先往左走到头，然后在往右走到头，这样能够更好地探索；eigenoption更可以保证若干option之间相互正交，这样自动探索出了的option尽量各不相同。

文章考虑高维或者连续状态空间，因此实际使用的是一个状态的表示  $\phi(s)$ ，这样eigenpurpose可以定义为

$$r_i^e(s, s') = \mathbf{e}^T (\phi(s') - \phi(s)), \quad (1)$$

离散情况下状态表示  $\phi(s)$  就是one-hot向量，这样就和前面写出的定义是一样的。

### 3. Successor representation

在PVF中，观察到价值函数  $\mathbf{v}_\pi = (\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1} \mathbf{r}$ 。如果能得到  $\mathbf{P}_\pi$  并且对其做特征值分解，找出最大特征值对应的特征向量，就可以对价值函数做近似。考虑到  $\mathbf{P}_\pi$  不好获取，就用随机游走产生的一步状态转移概率矩阵来近似，即  $\mathbf{P}$ 。再进一步，使用一个diffusion model  $\mathbf{L}$  来代替。

这里更直接一点，希望直接学习  $(\mathbf{I} - \gamma \mathbf{P}_\pi)^{-1}$ ，该矩阵可以改写为 successor representation (SR)

$$\Psi_\pi(s, s') = \mathbb{E}_{\pi, P} \left[ \sum_{t=0}^{\infty} \gamma^t \mathbb{1}_{\{S_t = s'\}} \mid S_0 = s \right].$$

有如下关系

$$v_\pi(s) = \sum_{s' \in \mathcal{S}} \Psi_\pi(s, s') r(s').$$

其好处是如果用神经网络来表示它，它可以被bootstrap地更新

$$\hat{\Psi}(s, j) \leftarrow \hat{\Psi}(s, j) + \eta \left[ \mathbb{1}_{\{s=j\}} + \gamma \hat{\Psi}(s', j) - \hat{\Psi}(s, j) \right], \quad (2)$$

它与diffusion model  $\mathbf{L}$  有如下关系，如果策略为随机游走，即  $\mathbf{T} = \mathbf{D}^{-1} \mathbf{W}$ ，那么其相应的 SR  $\Psi$  和  $\mathbf{L}$

的特征值和特征向量有如下联系。

**Theorem.** *Stachenfeld et al. (2014): Let  $0 < \gamma < 1$  s.t.  $\Psi = (I - \gamma T)^{-1}$  denotes the matrix encoding the SR, and let  $\mathcal{L} = D^{-1/2}(D - W)D^{-1/2}$  denote the matrix corresponding to the normalized Laplacian, both obtained under a uniform random policy. The  $i$ -th eigenvalue ( $\lambda_{SR,i}$ ) of the SR and the  $j$ -th eigenvalue ( $\lambda_{PVF,j}$ ) of the normalized Laplacian are related as follows:*

$$\lambda_{PVF,j} = \left[ 1 - (1 - \lambda_{SR,i})\gamma^{-1} \right]$$

*The  $i$ -th eigenvector ( $\mathbf{e}_{SR,i}$ ) of the SR and the  $j$ -th eigenvector ( $\mathbf{e}_{PVF,j}$ ) of the normalized Laplacian, where  $i + j = n + 1$ , with  $n$  being the total number of rows (and columns) of matrix  $T$ , are related as follows:*

$$\mathbf{e}_{PVF,j} = (\gamma^{-1} D^{1/2}) \mathbf{e}_{SR,i}$$

知乎 @张楚珩

## 4. Eigenoption

文章完整的做法如下

- Learn representation
- Extract eigenpurpose
- Learn eigenoption
- Solve the task with learned eigenoption

第一步，学习一个好的状态表示和相应的SR，令一个使用神经网络embedding的状态表示为  $\phi(s)$ 。想要用神经网络表示SR，本来我以为是输入两个状态，输出  $\Psi(s, s')$ ，但是文章隐含地做了分解  $\Psi(s, s') = \psi(\phi(s))^T \phi(s')$ ，即另外使用一个神经网络  $\psi(\cdot)$  来完成该表示。网络结构如下

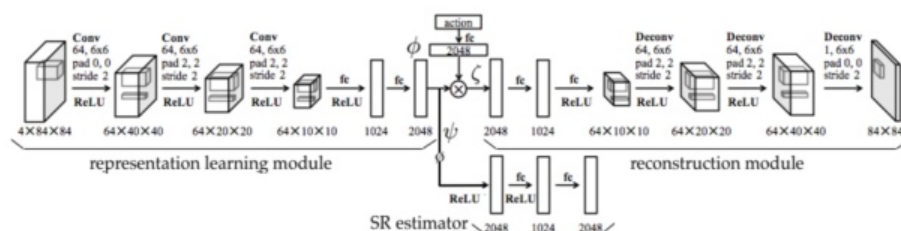


Figure 2: Neural network architecture used to learn the SR. The symbols  $\otimes$  and  $\phi$  denote element-wise multiplication and the fact that gradients are not propagated further back, respectively.

目标是学习  $\phi(\cdot)$  和  $\psi(\cdot)$ ，表示  $\phi(\cdot)$  的学习还包括了一个reconstruction module，我猜这一部分可以避免学习出来的表示没有信息量（参加DeepMDP文章中讨论的问题）。

损失函数为以下两部分的和

$$\mathcal{L}_{SR}(s, s') = \mathbb{E} \left[ \left( \phi^-(s) + \gamma \psi^-(\phi^-(s')) - \psi(\phi(s)) \right)^2 \right],$$

$$\mathcal{L}_{RE}(s, a, s') = \left( \zeta(\phi(s), a) - s' \right)^2,$$

第一个损失函数可以由前面写出来的SR更新公式得到，只需要做替换  $\Psi(s, s') = \psi(\phi(s))^\top \phi(s')$  即可。

第二步，从学习好的表示和SR中提取eigenpurpose。观察到eigenpurpose的公式

$$r_i^e(s, s') = \mathbf{e}^\top (\phi(s') - \phi(s)), \quad (1)$$

其中  $\phi(s), \phi(s')$  都很容易得到，只需要过一下学到的神经网络即可。eigenvector  $\mathbf{e}$  的需要我们得到一个矩阵  $\mathbf{x}$ ，然后对其做特征值分解，取其右特征值作为  $\mathbf{e}$ 。矩阵  $\mathbf{x}$  通过如下方式得到，提供一个均匀随机采样的策略得到  $t$  个状态，对于每个状态计算  $\psi(\phi(s))$ ，然后组成一个  $t \times \dim(\psi)$  的矩阵，其中每一行为相应的SR表示。

第三步，使用得到的若干个eigenpurpose来学习相应的eigenoption。这一步可以使用标准的RL算法来实现，不过文中使用了更为简单的方法，就是采取行动，使得即时的eigenpurpose最大（即  $\gamma=0$ ）；当任何行动都不可能产生正的eigenpurpose的时候就终止此option。

第四步，就是把学习到的若干个eigenoption作为action，来使用标准的RL算法来解决。由于这里的各个eigenoption单独的效果就会比较好（具有明确的目的性），因此，这样学起来会更容易。

## 实验

本文做了两个实验。

第一个实验室一个是四个房间的 Grid World，如下图所示，展示了该方法各个步骤中的中间结果。(a)展示了环境；(b)展示了第二步里面产生的特征向量（即 eigenpurpose）；(c)展示了学习到的相应 eigenoption；(d)展示了 eigenoption 的效果，即在 eigenoption 中随机采样平均只需要走大概 200 多步（diffusion time）就可以到达目标状态，而在原始的动作中随机采样平均需要 600 步左右才可以到达目标状态。

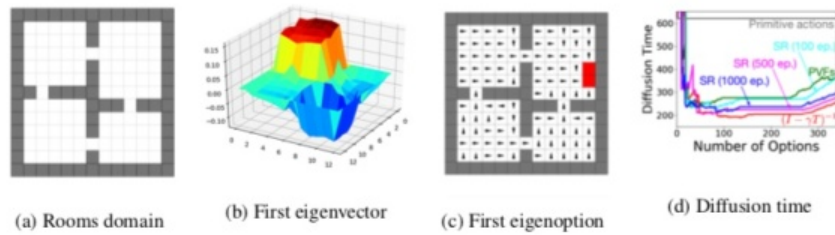


Figure 3: Results in the rooms domain. The rightmost figure depicts the diffusion time as eigenoptions are added to the agent's action set (sorted by eigenvalues corresponding to the eigenpurposes).

如下图所示，使用 eigenoption 学习，相应的 cumulative reward 指标也上升更快。不过值得指出的是，这里的横坐标并不包括学习到 eigenoption 需要的步数，因此该图不是一个公平的比较，只是为了展示学到的 eigenoption 有一定用处。

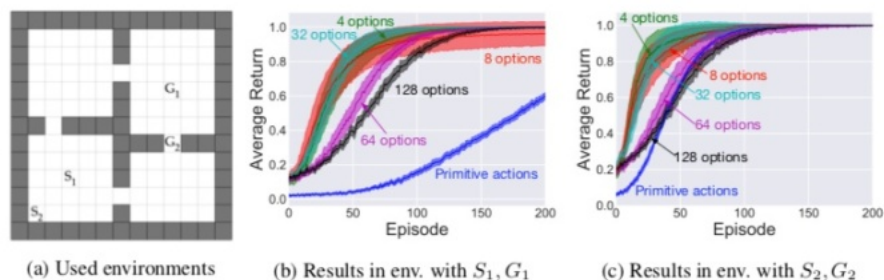


Figure 4: Different environments (varying start and goal locations) used in our evaluation (a), as well as the learning curves obtained in each one of these environments (b, c) for different numbers of options obtained from the SR when estimated after 100 episodes. See text for more details.

第二个实验是在三个在比较实际的游戏里做的。不过，本文没有使用学到的 eigenoption 来学习并且解决任务，只是展示了学习到的 eigenoption 具有一定的含义，即每一个 eigenoption 都会带领 agent 到一个比较有代表性的状态上。

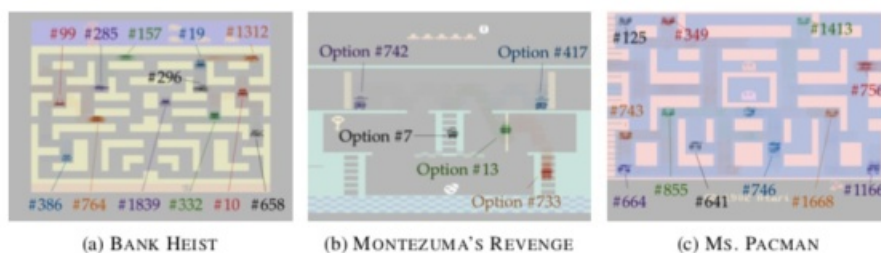



Figure 5: Plots of density of state visitation of eigenoptions discovered in three Atari 2600 games. States visited more frequently show darker images of the avatar. Note that an eigenoption's overwhelming mass of visitations corresponds to its terminal state, and that disparate options have different terminal states.

发布于 2019-06-17

强化学习 (Reinforcement Learning)

▲ 赞同 9 ▼   4 条评论   分享   喜欢   收藏   ...

文章被以下专栏收录

 强化学习前沿  
读呀读paper

进入专栏