模式分析与机器智能
工业和信息化部重点实验室
MIIT Key Laboratory of
Pattern Analysis & Machine Intelligence

ParNeC 模式识别与神经计算研究组
PAttern Recognition and NEural Computing

# DOES ZERO-SHOT REINFORCEMENT LEARNING EXIST?

**Ahmed Touati, Jérémy Rapin & Yann Ollivier**
Meta AI Research, Paris, {atouati, jrapin, yol}@meta.com

**ICLR 2023**

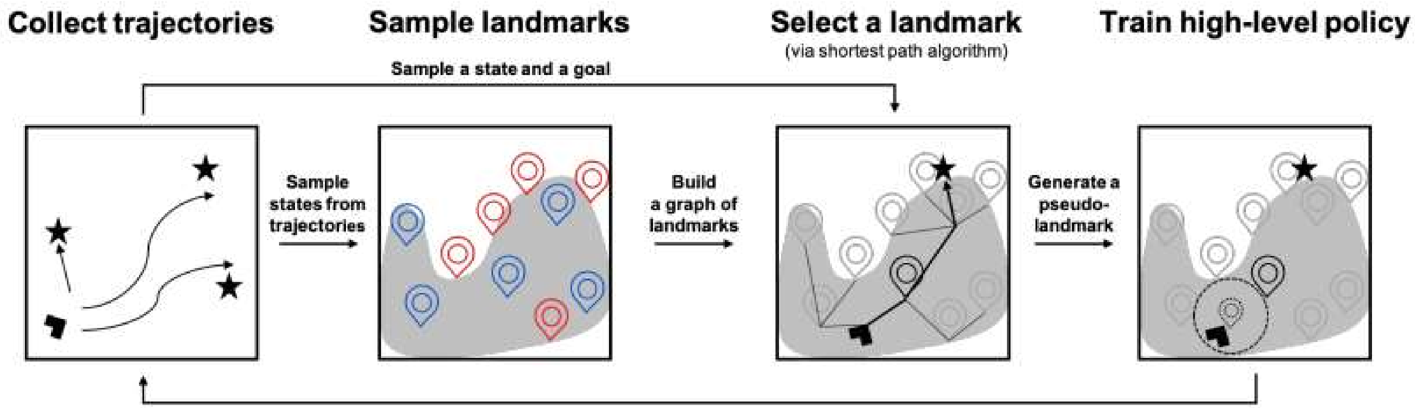
# Definition



What is Zero-Shot Reinforcement Learning?

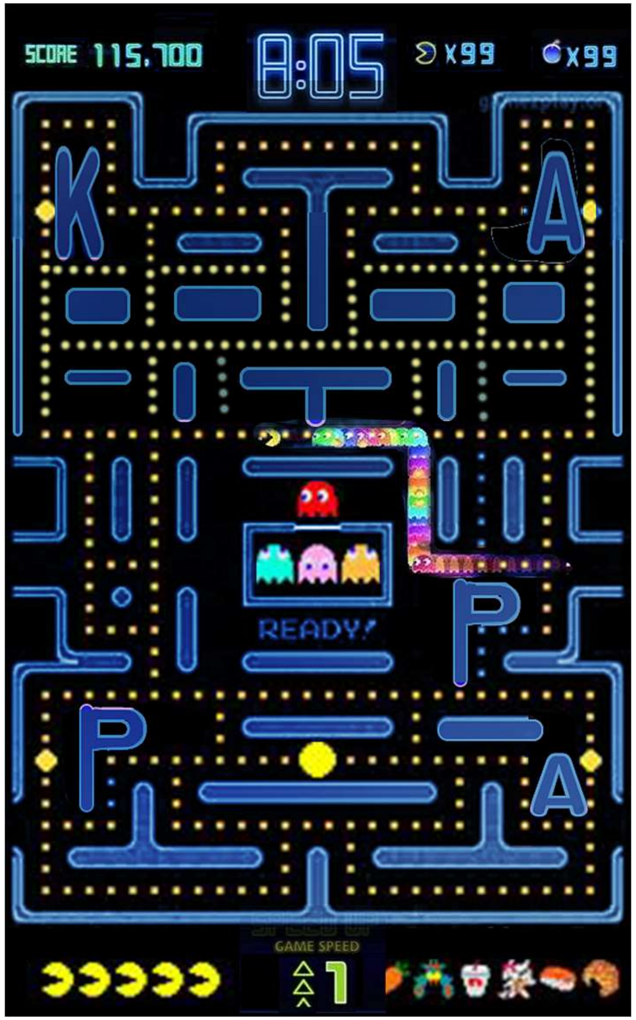
**Zero-shot RL: problem statement.** The goal of zero-shot RL is to compute a compact representation $\mathcal{E}$ of the environment by observing samples of reward-free transitions $(s_t, a_t, s_{t+1})$ in this environment. Once a reward function is specified later, the agent must use $\mathcal{E}$ to immediately produce a good policy, via only elementary computations without any further planning or learning. Ideally, for any downstream task, the performance of the returned policy should be close to the performance of a supervised RL baseline trained on the same dataset labeled with the rewards for that task.

Loose Limitations: How to define Zero-Shot RL with planning assisted?



# Motivation

Why Zero-Shot Reinforcement Learning?

## Some Methods

**Offline goal-conditioned Reinforcement Learning with ICM-like exploration?**

Advantages： Easy to Implement.

Disadvantages： goal-conditioned policy learned depends on the behavior policy.

**Diffusion Model for trajectories with rewards conditioned?**

Advantages： No need for data preprocess and subtle design.

Disadvantages： Low interpretability and solidity.

**Goal-conditioned or Skill-based Planning with landmarks?**

Advantages： Long-range planning and High interpretability.

Disadvantages： Consists of too many components and cost expensive.

**Q values and V values decoupled into successor representation and reward representation?**

Advantages： Easy to implement and Solid theoretical analysis

Disadvantages： Reward Space is infinite in continuous state space.

Perspective： View different tasks as different reward functions motivated.

# Decouple-Successor Features



这边做个统一，由于借鉴论文公式方面写的很烂，前后不对齐
我这里同意φ与s,a以及后面的reward func w为输入，Φ与s,a为输入。

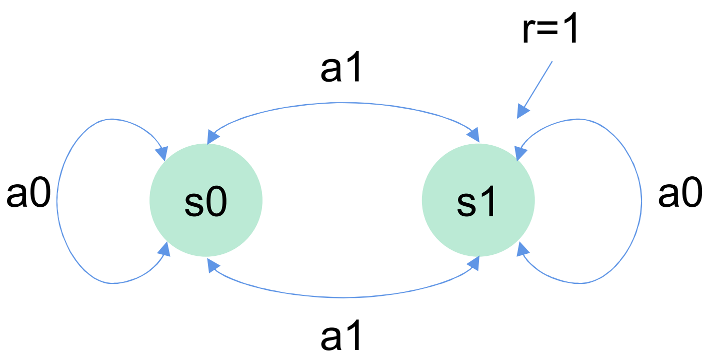
$$M \equiv (\mathcal{S}, \mathcal{A}, p, r, \gamma)$$

$$Q^{\pi}(s,a) \equiv \mathbb{E}^{\pi}\left[G_t \mid S_t = s, A_t = a\right] \quad \text{where} \quad G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i}$$

$$\pi'(s) \in \underset{a}{\operatorname{argmax}}\, Q^{\pi}(s,a);$$

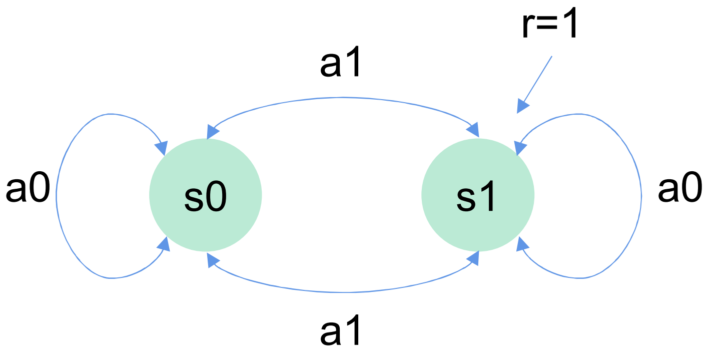
$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}\left[\sum_{i=t}^{\infty} \gamma^{i-t} \phi_{i+1} \mid S_t = s, A_t = a\right]^{\top} \mathbf{w} \equiv \psi^{\pi}(s,a)^{\top}\mathbf{w}, \quad \text{where} \quad r(s,a,s') = \phi(s,a,s')^{\top}\mathbf{w},$$

实际φ的物理意义应该是在该策略下未来s,a的折扣特征和 $\psi^{\pi}(s_0, a_0) := \mathbb{E}\left[\sum_{t \geq 0} \gamma^t \varphi(s_{t+1}) \mid s_0, a_0, \pi\right].$




$$\begin{array}{c} s_0, a_0 \\ s_0, a_1 \\ s_1, a_0 \\ s_1, a_1 \end{array} = \begin{pmatrix} 1,0,0,0 \\ 0,1,0,0 \\ 0,0,1,0 \\ 0,0,0,1 \end{pmatrix} \Rightarrow w = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

# Generalization Examples

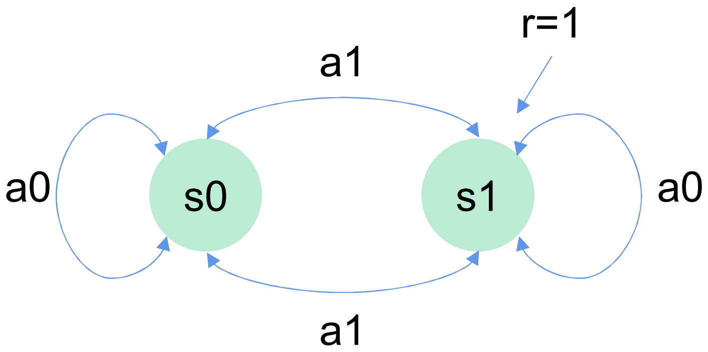



$$\begin{matrix} s_0, a_0 \\ s_0, a_1 \\ s_1, a_0 \\ s_1, a_1 \end{matrix} = \begin{pmatrix} 1,0,0,0 \\ 0,1,0,0 \\ 0,0,1,0 \\ 0,0,0,1 \end{pmatrix}$$

$w =$ W =

$$\varphi = \begin{pmatrix} 1+\gamma'+\frac{\gamma'^2}{1-\gamma} & 0 & 0 & 0 \\ 0 & 1+\frac{\gamma'^2}{1-\gamma} & 0 & 0 \\ 0 & 0 & 1+\gamma'+\frac{\gamma'^2}{1-\gamma} & 0 \\ 0 & 0 & 0 & 1+\frac{\gamma'^2}{1-\gamma} \end{pmatrix}, \gamma' = \frac{\gamma}{2}$$

## Generalization Examples




$$\begin{array}{c} s_0, a_0 \\ s_0, a_1 \\ s_1, a_0 \\ s_1, a_1 \end{array} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

$$w = W =$$

$$\varphi = \begin{pmatrix} 1+\gamma'+\frac{\gamma'^2}{1-\gamma} & \gamma'+\frac{\gamma'^2}{1-\gamma} & 0 & 0 \\ 0 & 1+\frac{\gamma'^2}{1-\gamma} & \gamma'+\frac{\gamma'^2}{1-\gamma} & 0 \\ 0 & 0 & 1+\gamma'+\frac{\gamma'^2}{1-\gamma} & \gamma'+\frac{\gamma'^2}{1-\gamma} \\ \gamma'+\frac{\gamma'^2}{1-\gamma} & 0 & \frac{\gamma'^2}{1-\gamma} & 0 \end{pmatrix}, \gamma' = \frac{\gamma}{2}$$

# How to learn Successor Features

ParNeC 模式识别与神经计算研究组 PAttern Recognition and NEural Computing

FAST TASK INFERENCE WITH VARIATIONAL INTRINSIC SUCCESSOR FEATURES

$$Q^{\pi}(s,a) = \psi(s,a,e(\pi))^{\top}\mathbf{w}.$$

$$\pi(s) = \operatorname*{argmax}_a \max_i \psi(s,a,e(\pi_i))^{\top}\mathbf{w} = \operatorname*{argmax}_a \max_i Q^{\pi_i}(s,a).$$

Maximize mutual information between w and pi_w

$$\mathcal{F}(\theta) = I(z; f(\tau_{\pi_\theta})) = H(z) - H(z|f(\tau_{\pi_\theta})).$$

$$\mathcal{F}(\theta) = -H(z|f(\tau_{\pi_\theta})).$$

$$\mathcal{F}(\theta) = \sum_{s,z} p(s,z)\log p(z|s) = \mathbb{E}_{\pi,z}[\log p(z|s)].$$

$$\begin{aligned}
-H(z|s) &= \sum_{s,z} p(s,z)\log p(z|s) \\
&= \sum_{s,z} p(s,z)\log p(z|s) + \sum_{s,z} p(s,z)\log q(z|s) - \sum_{s,z} p(s,z)\log q(z|s) \\
&= \sum_{s,z} p(s,z)\log q(z|s) + \sum_{s,z} p(s,z)\log p(z|s) - \sum_{s,z} p(s,z)\log q(z|s) \\
&= \sum_{s,z} p(s,z)\log q(z|s) + \sum_{s} p(s) KL(p(\cdot|s)||q(\cdot|s)) \\
&\geq \sum_{s,z} p(s,z)\log q(z|s) \\
&= \mathbb{E}_{\pi,z}[\log q(z|s)]
\end{aligned} \tag{12}$$

# How to learn Successor Features

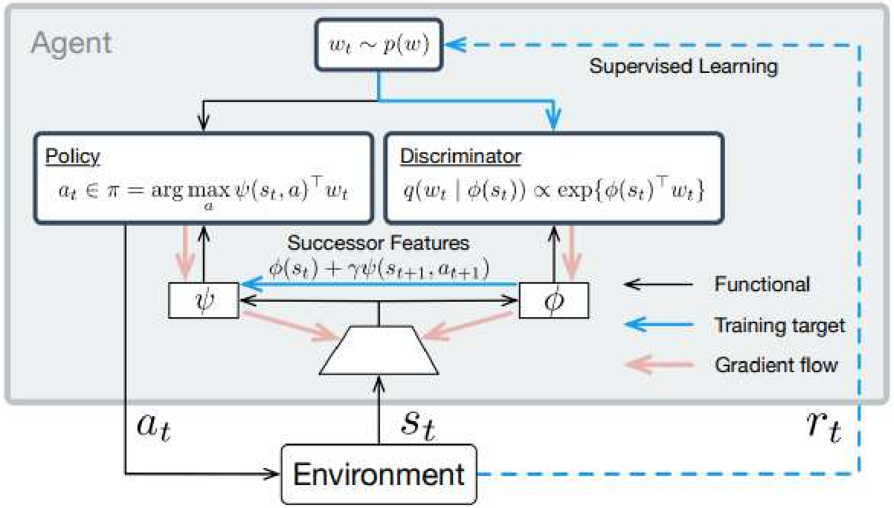



Figure 1: VISR model diagram. In practice $\mathbf{w}_t$ is also fed into $\psi$ as an input, which also allows for GPI to be used (see Algorithm 1 in Appendix). For the random feature baseline, the discriminator $q$ is frozen after initialization, but the same objective is used to train $\psi$.

$L_{\pi,q} = -\mathbb{E}_{\pi,z}[\log q(z|s)]$. Aimed at maximizing the reward collected policy under reward z (also w)

$r(s, a, s') = \phi(s, a, s')^\top \mathbf{w}$, What is predefined, View reward as a fixed point we can denote that.

$$\log q(\mathbf{w}|s) = \phi(s)^T \mathbf{w}.$$

# How to learn Successor Features



**Algorithm 1:** Training VISR

```
Randomly Initialize φ network                         // L2 normalized output layer
Randomly Initialize ψ network                         // dim(output) = #A × dim(W)
for e := 1, ∞ do
    sample w from L2 normalized N(0, I(dim(W)))       // uniform ball
    Q(·, a|w) ← ψ(·, a, w)^⊤ w, ∀a ∈ A
    for t := 1, T do
        Receive observation s_t from environment
        if using GPI then
            Q^{π_0}(·, a) ← Q(·, a|w) ∀a ∈ A
            for i := 1, 10 do
                sample w_i from VMF(μ = w, κ = 5)
                Q^{π_i}(·, a) ← ψ(·, a, w_i)^⊤ w, ∀a ∈ A
            end
            a_t ← ε-greedy policy based on max_i Q^{π_i}(s_t, ·)
        else
            a_t ← ε-greedy policy based on Q(s_t, ·|w)
        end
        Take action a_t, receive observation s_{t+1} from environment
        a' = argmax_a ψ(s_{t+1}, a, w)^⊤ w
        y = φ(s_t) + γψ(s_{t+1}, a', w)
        loss_ψ = Σ_i (ψ_i(s_t, a_t, w) − y_i)^2
        loss_φ = −φ(s_t)^⊤ w                          // minimize Von-Mises NLL
        Gradient descent step on ψ and φ               // minibatch in practice
    end
end
```

# Decouple-Successor States

ParNeC 模式识别与神经计算研究组 PAttern Recognition and NEural Computing

$$
\begin{aligned}
Q(s_0,a_0) &= r(s_0,a_0) + \gamma \sum_{s' \in S} p(s'|s_0,a_0) \sum_{a' \in A} \pi(a'|s') Q(s',a') \\
&= r(s_0,a_0) + \gamma \sum_{(s',a') \in X \times A} p(s'|s_0,a_0) \pi(a'|s') Q(s',a') \\
&= r(s_0,a_0) + \gamma \sum_{(s',a') \in X \times A} p(s',a'|s_0,a_0,\pi) Q(s',a') \\
&= r(s_0,a_0) + \gamma \sum_{(s',a') \in X \times A} p(s',a'|s_0,a_0,\pi) (r(s',a') + \gamma \sum_{(s'',a'') \in X \times A} p(s'',a''|s',a',\pi) Q(s'',a'')) \\
&= r(s_0,a_0) + \sum_{t>=1} \gamma^t \sum_{(s',a') \in X \times A} p^t(s',a'|s_0,a_0,\pi) r(s',a') \\
&= \sum_{t>=0} \sum_{(s',a') \in X \times A} \gamma^t p^t(s',a'|s_0,a_0,\pi) r(s',a')
\end{aligned}
$$

$$M^\pi(s_0,a_0,X) := \sum_{t \geq 0} \gamma^t \Pr((s_t,a_t) \in X \mid s_0,\, a_0,\, \pi)$$

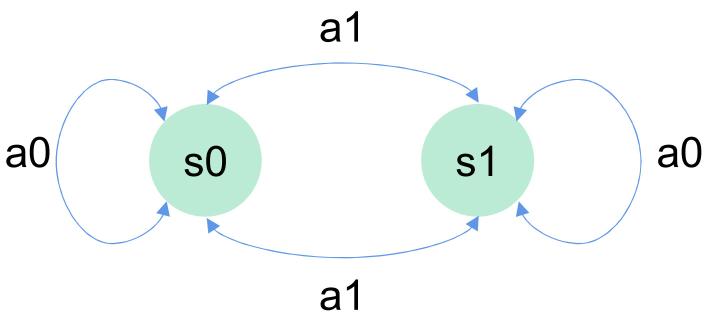
$$Q_r^\pi(s,a) = \sum_{s',a'} M^\pi(s,a,s',a') r(s',a')$$

# What M really is and the generalization



$$M^{\pi}(s_0, a_0, X) := \sum_{t \geq 0} \gamma^t \Pr\left((s_t, a_t) \in X \mid s_0, a_0, \pi\right)$$

Assume a goal-conditioned env where the goal is $s_t$ , $a_t$ from initial state $s_0$ , $a_0$. M is equal to Q($s_t$ , $a_t$).
So M is actually the discount distance between arbitrary state-action pairs under policy $\pi$.




$$\begin{array}{l} \text{s0,a0} \\ \text{s0,a1} \\ \text{s1,a0} \\ \text{s1,a1} \end{array} \quad \frac{1}{4(1-\gamma)} \begin{pmatrix} 4-2\gamma-\gamma^2 & 2\gamma-\gamma^2 & \gamma^2 & \gamma^2 \\ \gamma^2 & 4-4\gamma+\gamma^2 & 2\gamma-\gamma^2 & 2\gamma-\gamma^2 \\ \gamma^2 & \gamma^2 & 4-2\gamma-\gamma^2 & 2\gamma-\gamma^2 \\ 4-2\gamma-\gamma^2 & 4-2\gamma-\gamma^2 & \gamma^2 & 4-4\gamma+\gamma^2 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$4-4\gamma+\gamma^2 > 2\gamma-\gamma^2 \qquad 2\gamma-\gamma^2 > \gamma^2 \qquad 4-2\gamma-\gamma^2 > \gamma^2$$

If we want s1,a1 from initial s0, We can obtain from the M: Q(s0,a0)<Q(s0,a1), So we need to choose a1.

!!!Notably, the M is conditioned by the random policy, but can generalize to the goal-conditioned tasks.

# Calculate M



Like the calculation of Q value, we can use dynamic process to calculate M.

Backward: $M = \mathcal{I} + \gamma M P$

Forward: $M = \mathcal{I} + \gamma P M$

*Define a target update of $M$ via the Bellman equation, $M^{\text{tar}} := \text{Id} + \gamma P M_{\theta_t}$. Define the loss between $M$ and $M^{\text{tar}}$ via $J(\theta) := \frac{1}{2}\left\|M_\theta - M^{\text{tar}}\right\|_\rho^2$ using the norm (1). Then the gradient step on $\theta$ to reduce this loss is*

$$\widehat{\delta\theta_{\text{BTD}}} = \partial_\theta \tilde{m}_\theta(s, s) + \tilde{m}_\theta(s_1, s)\left(\gamma\, \partial_\theta \tilde{m}_\theta(s_1, s') - \partial_\theta \tilde{m}_\theta(s_1, s)\right)$$

$$\begin{aligned} -\partial_\theta J(\theta)_{|\theta=\theta_t} = \mathbb{E}_{s\sim\rho,\, s'\sim P(s,\mathrm{d}s'),\, s_2\sim\rho}\big[&\partial_\theta \tilde{m}_{\theta_t}(s, s) \\ &+ \partial_\theta \tilde{m}_{\theta_t}(s, s_2)\left(\gamma \tilde{m}_{\theta_t}(s', s_2) - \tilde{m}_{\theta_t}(s, s_2)\right)\big]. \end{aligned} \tag{21}$$

# Matrix Factorization

ParNeC 模式识别与神经计算研究组 PAttern Recognition and NEural Computing

$$\tilde{m}_\theta(s_1, s_2) = F_{\theta_F}(s_1)^\top B_{\theta_B}(s_2)$$

**Definition 1** (Forward-backward representation). *Let $Z = \mathbb{R}^d$ be a representation space, and let $\rho$ be a measure on $S \times A$. A pair of functions $F\colon S \times A \times Z \to Z$ and $B\colon S \times A \to Z$, together with a parametric family of policies $(\pi_z)_{z \in Z}$, is called a* forward-backward representation *of the MDP with respect to $\rho$, if the following conditions hold for any $z \in Z$ and $(s, a), (s_0, a_0) \in S \times A$:*

$$\pi_z(s) = \arg\max_a F(s, a, z)^\top z, \qquad M^{\pi_z}(s_0, a_0, \mathrm{d}s, \mathrm{d}a) = F(s_0, a_0, z)^\top B(s, a)\rho(\mathrm{d}s, \mathrm{d}a) \tag{2}$$

*where $M^\pi$ is the successor measure defined in* (1)*, and the last equality is between measures.*

**Theorem 2** (FB representations encode all optimal policies). *Let $(F, B, (\pi_z))$ be a forward-backward representation of a reward-free MDP with respect to some measure $\rho$.*

*Then, for any bounded reward function $r\colon S \times A \to \mathbb{R}$, the following holds. Set*

$$z_R := \int_{s,a} r(s, a)B(s, a)\,\rho(\mathrm{d}s, \mathrm{d}a). \tag{3}$$

*assuming the integral exists. Then $\pi_{z_R}$ is an optimal policy for reward $r$ in the MDP. Moreover, the optimal Q-function $Q^\star$ for reward $r$ is $Q^\star(s, a) = F(s, a, z_R)^\top z_R$.*

# Matrix Factorization



- It provides a direct representation of the value function at every state, without learning an additional model of $V$. Namely,

$$V(s) \approx F(s)^{\top} B(R), \qquad B(R) := \mathbb{E}_{s\sim\rho}[r_s\, B(s)] \tag{47}$$

  where the "reward representation" $B(R)$ can be directly estimated by an online average of $B(s)$ weighted by the reward $r_s$ at $s$. This is a direct consequence of (17). For instance, with sparse rewards, each time a reward is observed, the value function is updated everywhere. 10

  This point applies to successor states, but not to goal-dependent value functions, which cannot handle arbitrary rewards.

- It simplifies the sampling of a pair of states $(s, s_2)$ needed for forward TD. Indeed, the forward TD update (21) factorizes as an expectation over $s$, times an expectation over $s_2$ (Section 6.2 below), which can be estimated independently. The same applies to backward TD. This can potentially reduce variance a lot, and even allows for purely "trajectory-wise" online estimates using only the current transition $s \to s'$, without sampling of another independent state $s_2$. (Once more, this works for successor states but not for goal-dependent value functions, since in that case the transitions $s \to s'$ depend on $s_2$.)

- It produces two (policy-dependent) representations of states, a forward and a backward one, in a natural way from the dynamics of the MDP and the current policy. This could be useful for other purposes.

- Even in the tabular case, when the state space is discrete and unstructured, this provides a form of prior or generalization between states (based on a low-rank prior for the successor state operator). States that are linked by the MDP dynamics get representations $F$ and $B$ that are close.

- It has some of the properties of the second-order methods of Section 7, without their complexity. This is proved in Appendix F.1.

The shortcomings are as follows:

- It approximates the successor state operator by an operator of rank at most $r$. This is never an exact representation unless the representation dimension $r$ is at least the number of distinct states.

- The best rank-$r$ approximation of $(\mathrm{Id} - \gamma P)^{-1}$ erases the small singular values of $P$: thus this representation will tend to erase "high frequencies" in the reward and value function, and provide a spatially smoother approximation focusing on long-range behavior. This is fine as long as the reward is not a "fast-changing" function made up of high frequencies (such as a "checkerboard" reward).

  This can be expected: learning a reward-agnostic object such as $M$ cannot work equally well for all rewards. For these reasons, it may be useful to use a mixed model for the value function with the FB model as a baseline, such as

$$V_\varphi(s) = F(s)^{\top} B(R) + v_\varphi(s) \tag{48}$$

  where $F$ and $B$ are learned via successor states, $B(R)$ is as in (47), and $\varphi$ is learned via ordinary TD on the remainder. The $FB$ part will catch reward-independent, long-range behavior, while the $v_\varphi$ part will be needed to catch high frequencies in a particular reward function.

# Code

**Algorithm 1** FB algorithm: Unsupervised Phase

**Inputs:** replay buffer $\mathcal{D}$ , Polyak coefficient $\alpha$ , $\nu$ a probability distribution over $\mathbb{R}^d$, randomly initialized networks $F_\theta$ and $B_\omega$, learning rate $\eta$, mini-batch size $b$, number of episodes $E$, number of gradient updates $N$, temperature $\tau$ and regularization coefficient $\lambda$.

**for** $m = 1, \dots$ **do**
- `/*` `Collect` $E$ `episodes`
- **for** episode $e = 1, \dots E$ **do**
  - Sample $z \sim \nu$
  - Observe an initial state $s_0$
  - **for** $t = 1, \dots$ **do**
    - Select an action $a_t$ according to some behaviour policy (e.g the $\varepsilon$-greedy with respect to $F_\theta(s_t, a, z)^\top z$ )
    - Observe next state $s_{t+1}$
    - Store transition $(s_t, a_t, s_{t+1})$ in the replay buffer $\mathcal{D}$
  - **end for**
- **end for**
- `/*` `Perform` $N$ `stochastic gradient descent updates`
- **for** $n = 1 \dots N$ **do**
  - Sample a mini-batch of transitions $\{(s_i, a_i, s_{i+1})\}_{i \in I} \subset \mathcal{D}$ of size $|I| = b$.
  - Sample a mini-batch of target state-action pairs $\{(s'_i, a'_i)\}_{i \in I} \subset \mathcal{D}$ of size $|I| = b$.
  - Sample a mini-batch of $\{z_i\}_{i \in I} \sim \nu$ of size $|I| = b$.
  - Set $\pi_{z_i}(\cdot \mid s_{i+1}) = \texttt{softmax}(F_{\theta^-}(s_{i+1}, \cdot, z_i)^\top z_i / \tau)$
  - $\mathscr{L}(\theta, \omega) = \frac{1}{2b^2} \sum_{i,j \in I^2} \left( F_\theta(s_i, a_i, z_i)^\top B_\omega(s'_j, a'_j) - \gamma \sum_{a \in A} \pi_{z_i}(a \mid s_{i+1}) \cdot F_{\theta^-}(s_{i+1}, a, z_i)^\top B_{\omega^-}(s'_j, a'_j) \right)^2 - \frac{1}{b} \sum_{i \in I} F_\theta(s_i, a_i, z_i)^\top B_\omega(s_i, a_i)$
  - `/*` `Compute orthonormality regularization loss`
  - $\mathscr{L}_{\texttt{reg}}(\omega) = \frac{1}{b^2} \sum_{i,j \in I^2} B_\omega(s_i, a_i)^\top \texttt{stop-gradient}(B_\omega(s'_j, a'_j)) \cdot \texttt{stop-gradient}(B_\omega(s_i, a_i)^\top B_\omega(s'_j, a'_j)) - \frac{1}{b} \sum_{i \in I} B_\omega(s_i, a_i)^\top \texttt{stop-gradient}(B_\omega(s_i, a_i))$
  - Update $\theta \leftarrow \theta - \eta \nabla_\theta \mathscr{L}(\theta, \omega)$ and $\omega \leftarrow \omega - \eta \nabla_\omega (\mathscr{L}(\theta, \omega) + \lambda \cdot \mathscr{L}_{\texttt{reg}}(\omega))$
- **end for**
- `/*` `Update target network parameters`
- $\theta^- \leftarrow \alpha \theta^- + (1 - \alpha) \theta$
- $\omega^- \leftarrow \alpha \omega^- + (1 - \alpha) \omega$

**end for**

# Discussion

$$\psi^\pi(s_0, a_0) := \mathbb{E}\left[\sum_{t\geq 0}\gamma^t\varphi(s_{t+1}) \mid s_0,\, a_0,\, \pi\right].$$

$$Q^\pi(s,a) = \mathbb{E}^\pi\left[\sum_{i=t}^{\infty}\gamma^{i-t}\boldsymbol{\phi}_{i+1} \mid S_t = s, A_t = a\right]^\top \mathbf{w} \equiv \boldsymbol{\psi}^\pi(s,a)^\top \mathbf{w}, \qquad w = reward/\phi$$

$$\pi_z(s) = \arg\max_a F(s,a,z)^\top z, \qquad M^{\pi_z}(s_0, a_0, \mathrm{d}s, \mathrm{d}a) = F(s_0, a_0, z)^\top B(s,a)\rho(\mathrm{d}s, \mathrm{d}a) \qquad z_R := \mathbb{E}[r(s,a)B(s,a)]$$

If given the same representation for Φ and B in finite space, the two Q values will be totally same! Where F is just a discount matrix to calculate successor features. It means FB = φ.

But the two values differs when it is just a unsupervised feature extraction. And the procedure of inference is totally different.

# Some experiments-SFs



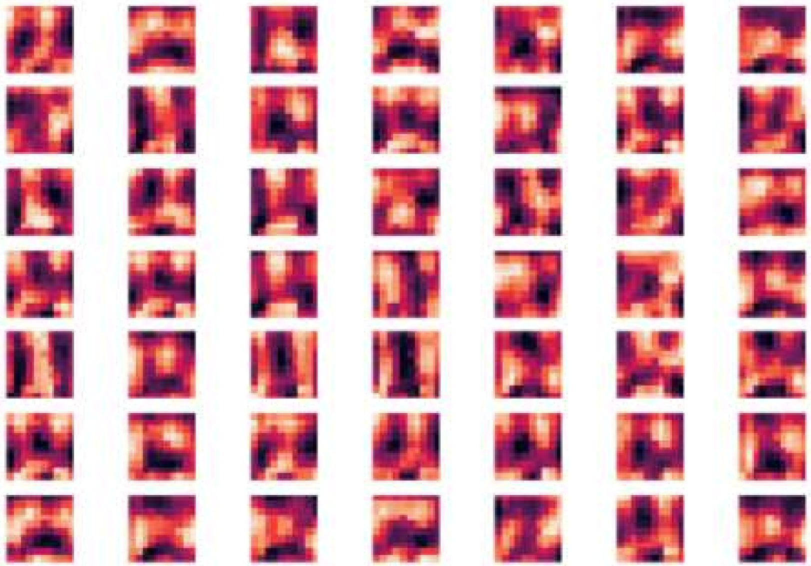


Figure 4: 49 randomly sampled reward functions learned by VISR.

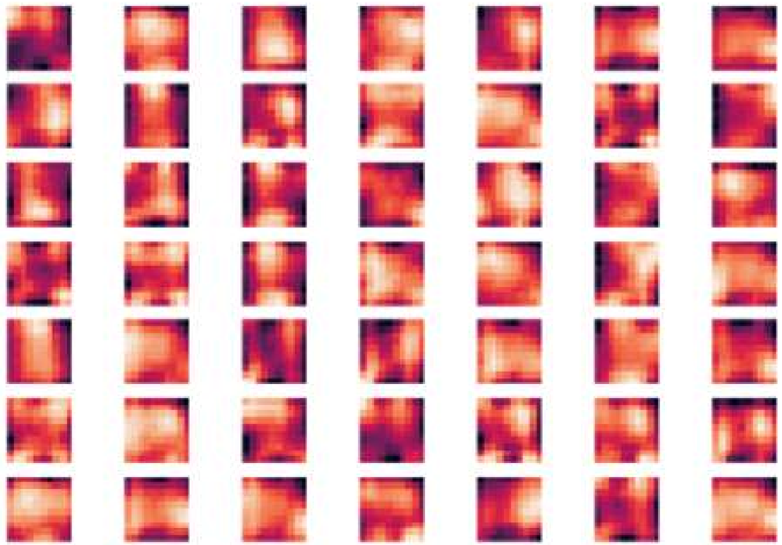


Figure 5: The approximations to the optimal value functions for the reward functions in Figure 4, computed by VISR through GPI on 10 randomly sampled policies.

# Some experiments-FB



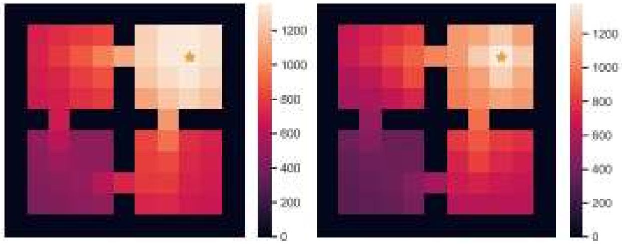



Figure 3: Heatmap of $\max_a F(s,a,z_R)^\top z_R$ for $z_R = B(☆)$ **Left**: $d = 25$. **Right**: $d = 75$.

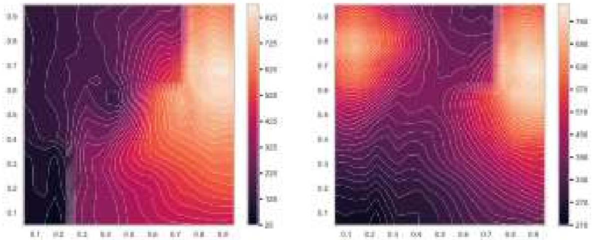


Figure 4: Contour plot of $\max_{a\in A} F(s,a,z_R)^\top z_R$ in Continuous Maze. **Left**: for the task of reaching a target while avoiding a forbidden region, **Right**: for two equally rewarding targets.

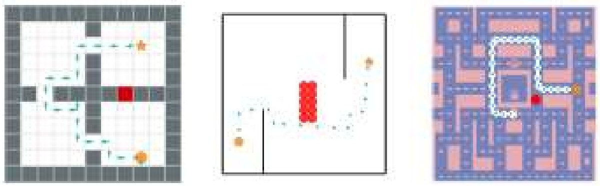


Figure 5: Trajectories generated by the $F^\top B$ policies for the task of reaching a target position (star shape ☆ while avoiding forbidden positions (red shape ●)

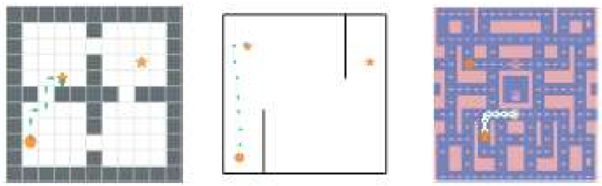


Figure 6: Trajectories generated by the $F^\top B$ policies for the task of reaching the closest among two equally rewarding positions (star shapes ☆). (Optimal $Q$-values are not linear over such mixtures.)

# Some experiments-FB



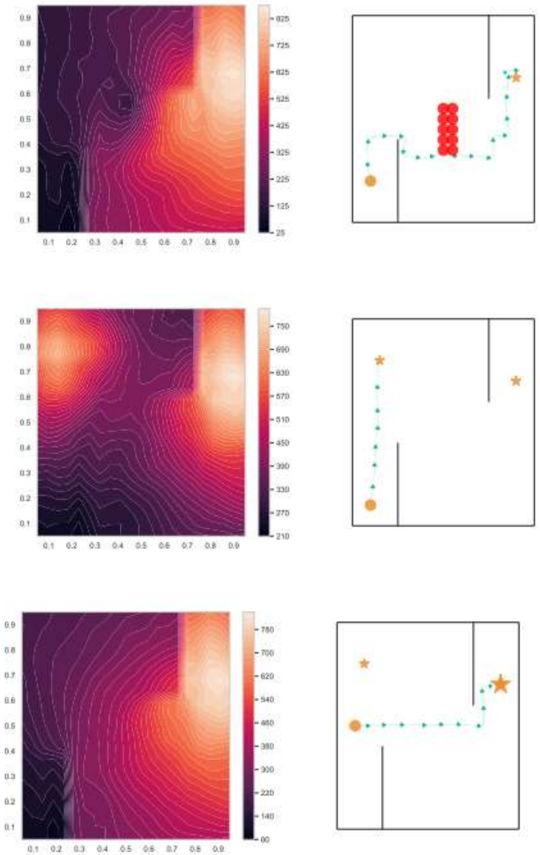


Figure 14: **Continuous Maze**: Contour plots plot of $\max_{a \in A} F(s, a, z_R)^\top z_R$ (left) and trajectories of the $\varepsilon$ greedy policy with respect to $F(s, a, z_R)^\top z_R$ with $\varepsilon = 0.1$ (right). **Left**: for the task of reaching a target while avoiding a forbidden region, **Middle**: for the task of reaching the closest goal among two equally rewarding positions, **Right**: choosing between a small, close reward and a large, distant one..

Figure 15: **Ms. Pacman**: Trajectories of the $\varepsilon$ greedy policy with respect to $F(s, a, z_R)^\top z_R$ with $\varepsilon = 0.1$ (right). **Top row**: for the task of reaching a target while avoiding a forbidden region, **Middle row**: for the task of reaching the closest goal among two equally rewarding positions, **Bottom row**: choosing between a small, close reward and a large, distant one..

Figure 18: **Continuous maze**: Visualization of FB embedding vectors after projecting them in two-dimensional space with t-SNE. **Left**: the states to be mapped. **Middle**: the $F$ embedding. **Right**: the $B$ embedding.

# Thanks