Successor Representation, Replay and Linear Successor Feature Model

Cyril REGAN

May 26, 2021

Table of Contents

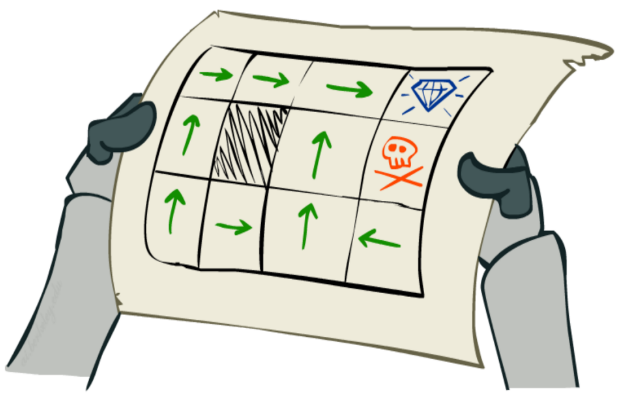
Markov Decision Process (MDP)

15

RL purpose is to solve Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$

$$
\begin{array}{lll}
\langle \quad \mathcal{S} & : & \text{State Space : } s \in \mathcal{S} \quad , \\
\mathcal{A} & : & \text{Action Space : } a \in \mathcal{A} \quad , \\
p & : & \text{Transition function } p \text{ is Markovian : } p(s'|s,a) = \\
 & & \mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a] \quad , \\
r & : & \text{Reward function } r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [\mathbb{R}, \mathbb{R}] \quad , \\
\gamma & : & \text{discount factor that captures a preference for proximal rewards} \quad \rangle
\end{array}
$$

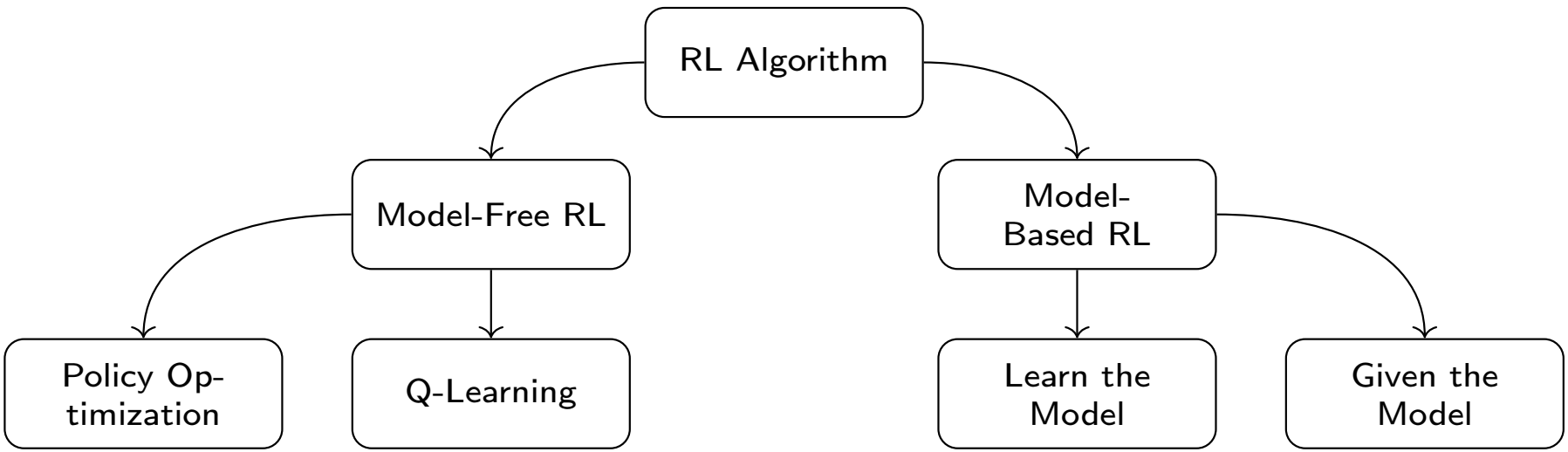
But what is solve MDP ? Find the *optimal policy* π^* to maximize the rewards. The *policy* π is a function that maps the state of an agent to an action :



$$\pi(a|s) = \mathbb{P}[a_t = a|s_t = s]$$

2 main type of Models to solve a MDP :

- Model-based : An internal model (p, r) is known or learned
- Model-free : No internal model



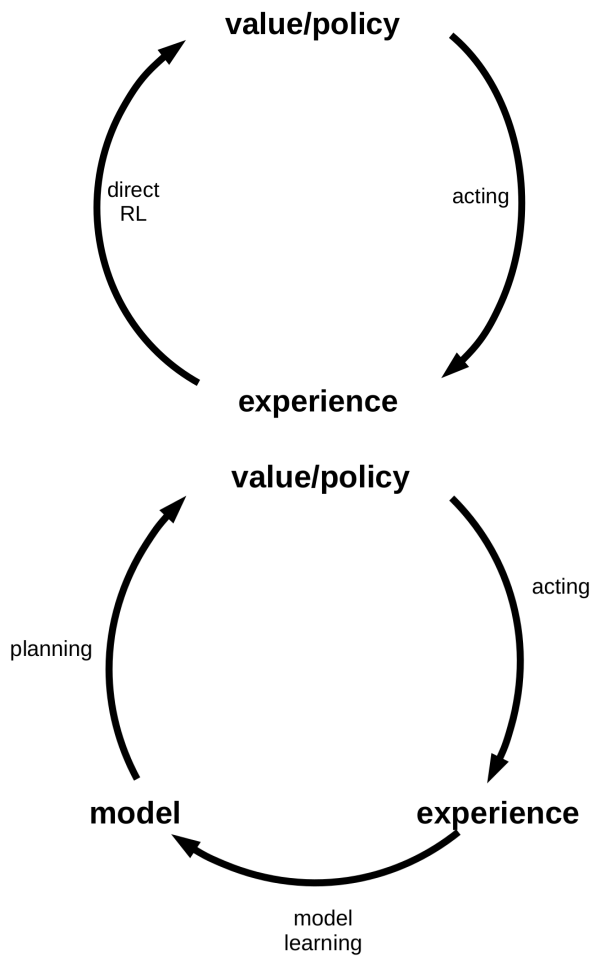
Model Free - Model Based

Model Free :

- – Need many on-line steps,
- – Inflexible (transition / reward change),
- \+ No planning phase,
- \+ Computationally cheap.

Model based :

- – Need to learn model (can be given),
- \+ few on-line steps (off-line learning),
- \+ flexible (transition / reward change),
- – planning can be complex (trajectory optimization),
- – computationally expensive.



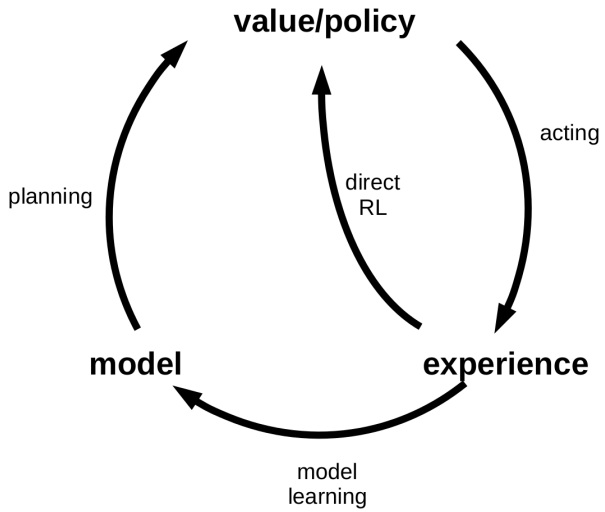
Dyna Model

Dyna Model :

- Model Free learning on-line.
- Model Based Planning off-line

$\equiv$

Simulated experiences similar to **hippocampal replay**



Dyna Model Based planning can be constructed with the :

Successor Representation (SR)

(instead to the transition function p).

Table of Contents

Successor Representation : a time based representation

SR matrix construction *(Dayan 1993; Gershman 2018)*

SR is a *"compressed time based"* representation of the **actual state** and **all the future state occupancy** (discounting by the γ factor) :

$$\boldsymbol{M}^{\pi}(s', s, a) = \mathbb{E}_{\pi}\Big[\sum \gamma^{t} \mathbb{I}\{s_t = s'\} | s_0 = s, a_0 = a\Big] \quad (1)$$

Where $\mathbb{I}\{...\} = 1$ if its argument is true (0 otherwise). In tabular state, SR matrix $\boldsymbol{M}^{\pi}$ relation with transition matrix $(\boldsymbol{\mathcal{P}}^{\pi})^{a}_{ss'} = p(s'|s, a)$ is :

$$\boldsymbol{M}^{\pi}(s', s, a) = \mathbb{I}\{s = s'\} + \gamma(\boldsymbol{\mathcal{P}}^{\pi})^{a}_{ss'} + \gamma^{2}(\boldsymbol{\mathcal{P}}^{\pi 2})^{a}_{ss'} + \gamma^{3}(\boldsymbol{\mathcal{P}}^{\pi 3})^{a}_{ss'} + ... \quad (2)$$

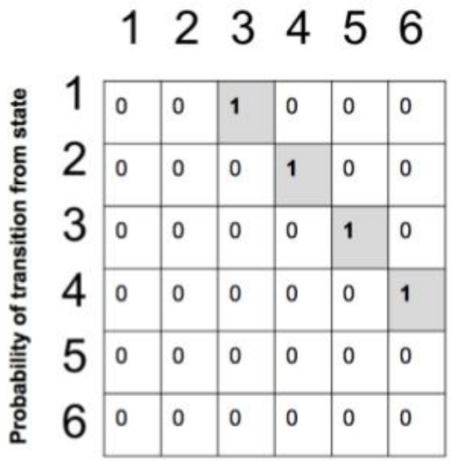
$$\text{In a matrix formulation : } \boldsymbol{M}^{\pi} = (1 - \gamma \boldsymbol{\mathcal{P}}^{\pi})^{-1} \quad (3)$$

SR relation with transition function

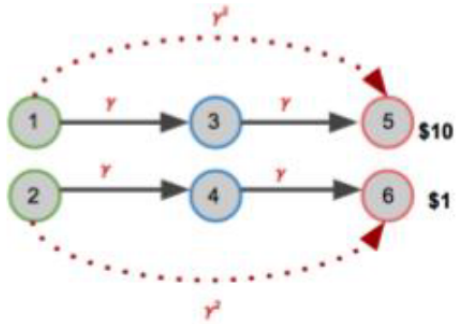
SR Matrix $\boldsymbol{M}^{\pi}$: a time based representation :

Rows : expected future occupancy of a state

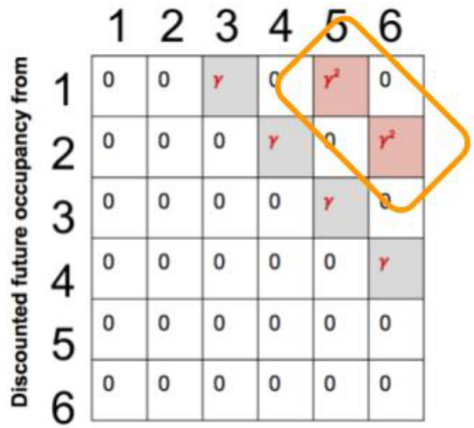
Columns : expected past occupancy of a state



Transition matrix ($\mathcal{P}^{\pi}$)



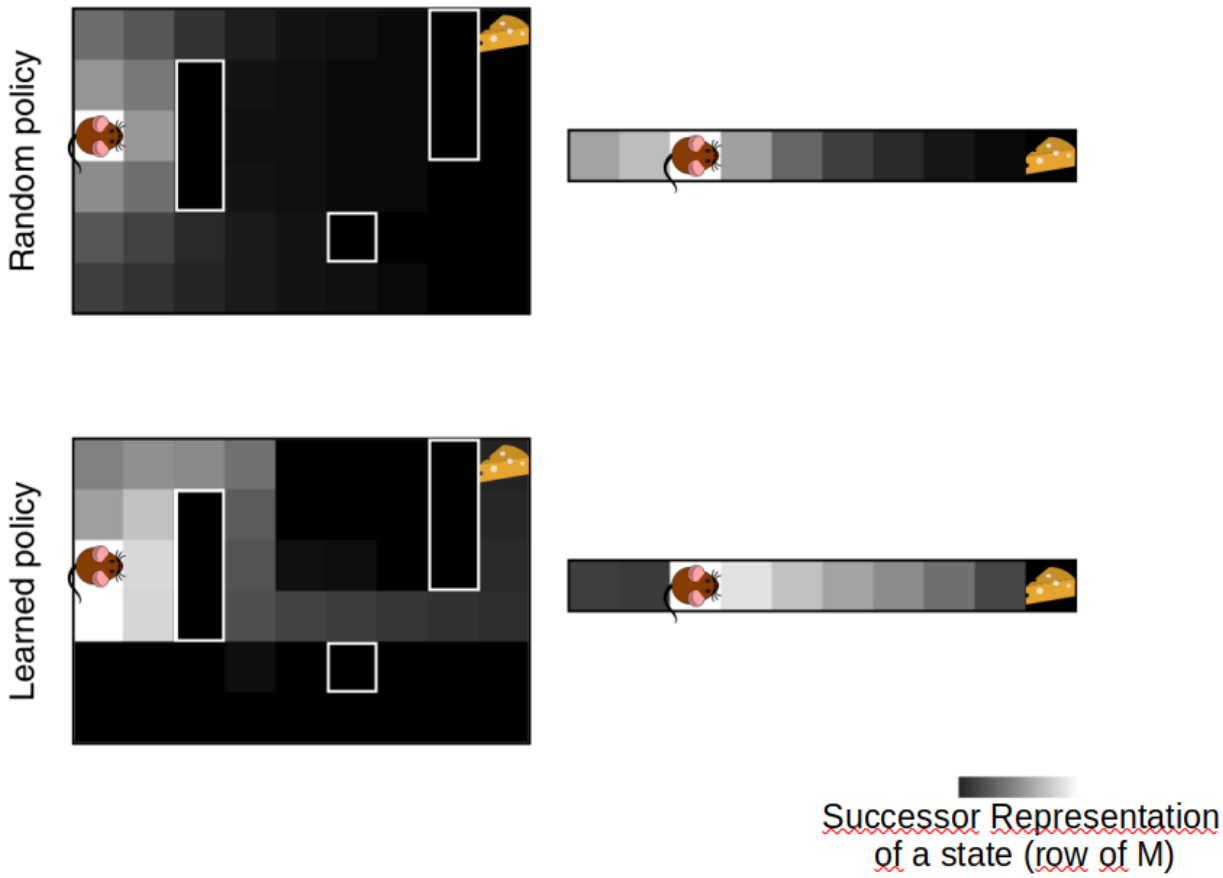
Mdp



SR matrix $\boldsymbol{M}^{\pi}$

(Momennejad and Howard 2018)

SR policy dependence



Successor Representation policy dependence (Mattar-Daw 2018)

The Successor Representation TD-Learning

SR Matrix $\boldsymbol{M}^{\pi}$ can be learn as TD-Learning (Gershman 2018):

$$\boldsymbol{M}^{\pi}(s,:) \leftarrow \boldsymbol{M}^{\pi}(s,:) + \alpha(1_s + \gamma \boldsymbol{M}^{\pi}(s',:) - \boldsymbol{M}^{\pi}(s,:)) \tag{4}$$

Nice property : Value functions (V, Q) are **linear** with SR $(\boldsymbol{M})$:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{+\infty} \gamma^k r_{t+k} | s_t = s\right] = \sum_{s'} \boldsymbol{M}^{\pi}(s,s')\boldsymbol{r}(s') \tag{5}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{+\infty} \gamma^k r_{t+k} | s_t = s, a_t = a\right] = \sum_{s'} \boldsymbol{M}^{\pi}(s,s',a)\boldsymbol{r}(s',a) \tag{6}$$

Table of Contents

Replay :

- Model-Free : experience storage + off-line learning (Lin 1992)
  - Random experience replay (Mnih 2013; Mnih 2015)
  - prioritized experience replay (Schaul 2016)
- Model-Based / SR-Based : simulated experience replay

2 $\neq$ prioritized simulated replay method :

- The Expected Value of a Bellman backup (EVB) (Mattar-Daw 2018)
- The Bidirectionnal Model-Based approach (Khamassi-Girard 2020).

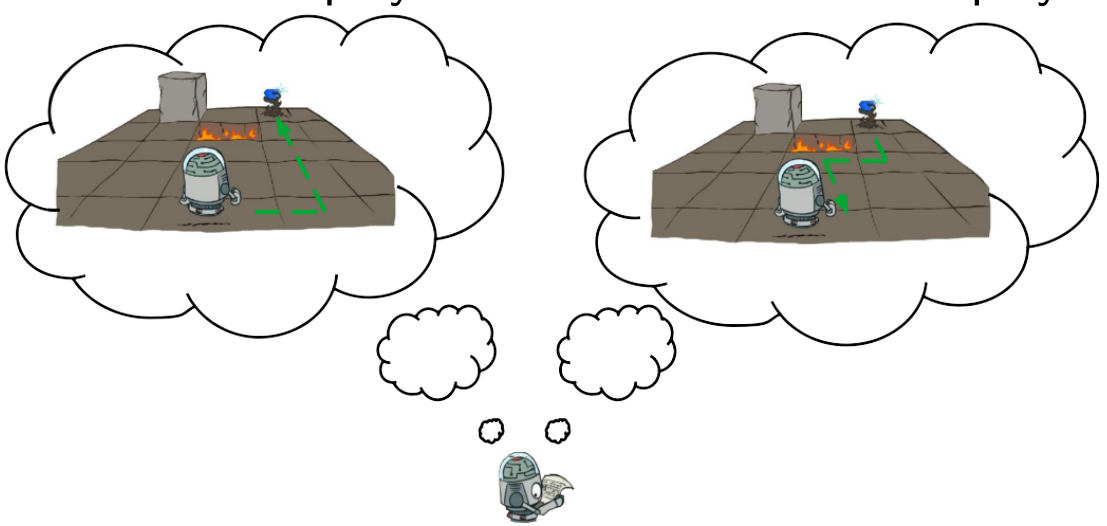
2 computational families of simulated replay :

EVB (Mattar-Daw 2018):	Need	Gain
Bidirectionnal MB (Khamassi-Girard 2020):	Trajectory sampling	Prioritized sweeping

---

Forward replay

Backward replay



Expected Value of a Bellman backup (EVB)

The Bellman backup

EVB (Mattar-Daw 2018) algorithm is based on :

- Dyna architecture,
- and prioritized Bellman backup.

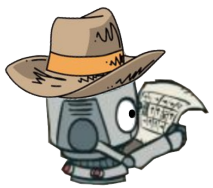
A Bellman backup is the TD-learning iteration of Q :

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha \left[r_k + \gamma \max_a Q(s'_k, a) - Q(s_k, a_k) \right] \quad (7)$$

A Bellman backups occur on-line or off-line with simulated replay.

EVB Policy :

Softmax Q function $\pi(a|s) = \frac{e^{\beta Q(s,a)}}{\sum_i e^{\beta Q(s,i)}}$:



Pure **exploration**
$\beta \rightarrow 0^+$

= Probability for all actions

Pure **exploitation**
$\beta \rightarrow \infty$

Probability = 1 for the action with max Q value (= 0 for the others)

Expected Value of a Bellman backup (EVB) :

EVB is the criterion to **order** the replay experiences for the offline update of Q (Bellman Backup).

$$EVB(s_k, a_k) = Gain(s_k, a_k) \times Need(s_k) \tag{8}$$

- **Gain** : expected return when reaching (s_k, a_k),
- **Need** : number of times the agent expects to visit s_k in the future.

Gain :

Gain on (s_k, a_k) quantifies the **improvement** in the $\overbrace{\text{expected return}}^{Q_\pi(s_k,a_k)=\mathbb{E}_\pi\left[\sum_k^\infty \gamma^i r_{t+i}\right]}$ as a result of a $\overbrace{\textbf{policy change}}^{\left(\pi_{new}(a|s_k)-\pi_{old}(a|s_k)\right)}$ after a $\overbrace{\text{Bellman Backup on } (s_k, a_k)}^{\boldsymbol{Q}^{new}(s_k,a_k)\leftarrow\boldsymbol{Q}^{old}(s_k,a_k)}$:

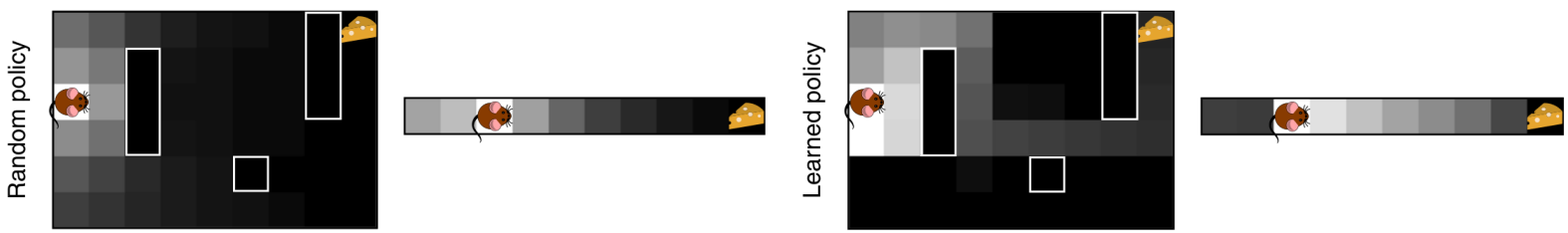
$$Gain(s_k, a_k) = \sum_a \overbrace{Q_{\pi_{new}}(s_k, a)}^{\textbf{expected return}} \overbrace{\left(\pi_{new}(a|s_k) - \pi_{old}(a|s_k)\right)}^{\textbf{policy change}} \tag{9}$$

where $\pi_{new}(a|s_k) = softmax(\boldsymbol{Q}^{new}(a, s_k))$ $_{(resp.\ \pi_{old})}$

Need :

Need on s_k quantifies the discounted number of times the agent expects to visit states in the future :

$$Need(s_k) = \sum_{i}^{\infty} \gamma^i . 1_{s_{t+i}, s_k} = \sum_{i \in \mathcal{S}} M[s_k, s_i] = \sum_{row\ s_k} M \qquad (10)$$



: SR of a state (row of M)

EVB algorithm and limitations :

Offline learning : Compute fixed number of planning steps on on the **start** and the **end** of each episode (*not flexible*).

- planning step :
  - EVB is computed **on all possible experiences** (*very high computational cost*).
  - Q update (Bellman backup) on experience with the higher EVB

Online learning = Q learning : Update (Bellman backup) on current experience

EVB Algorithm :

Loop on **all states-actions** and store all $\overbrace{(s_k, a_k, r_k, s'_k)_{k \in \mathcal{S} \times \mathcal{A}}}^{\text{experiences}}$ of the environment

Construct Transition ($\mathcal{P}$) and SR ($\boldsymbol{M} = (1 - \gamma \mathcal{P})^{-1}$) matrices

loop on n_{max} *episodes* **while** $n <= n_{max}$ **do**

  p_{max} ***offline learning*** *steps at the start of episode* **while** $p <= p_{max}$ **do**

    Compute EVB on **all states-actions** (s_k, a_k)

    $\overbrace{\text{Update}Q}^{\text{Belman Backup}}$ with the $\overbrace{\text{experience}}^{(s_k, a_k, r_r, s'_k)}$ with higher EVB

  end

  online learning

  $a_t \leftarrow argmax_a softmax_\beta(Q(s_t, a))$

  Take action a_t receive $(s_{t+1}, r_{t+1}, done)$

  Update Q (Belman Backup), $\mathcal{P}$ and $\boldsymbol{M}$ on $(s_t, a_t, s_{t+1}, r_{t+1})$

  p_{max} ***offline learning*** *steps again at the end of episode* **while** $p <= p_{max}$ **do**

  end

end

Bidirectionnal Model-Based (MB)

Bidirectionnal MB : prioritized replay with a **heuristic-based** approach (Khamassi-Girard 2020) .

- Backward replay : **Prioritized sweeping** (PS)
  - Start : replay starting from states whose Q-value changed recently,
  - and then to their predecessors
  - and the predecessors to their predecessors
  - ... and so on.
- Forward replay : **Trajectory sampling** (TS)
  - Start : generate continuous trajectories from the current position
  - ... until forward and backward search reached a connection state.

Prioritized Sweeping (PS) :

PS is backward replay. PS consists of weighing experiences based on how surprising they are, measured by the absolute value of the MB TD error signal :

$$\begin{aligned} \delta &= |\boldsymbol{Q}^{n+1}(s,a) - \boldsymbol{Q}^{n}(s,a)|, \\ \boldsymbol{Q}^{n+1}(s,a) &\leftarrow r(s,a) + \gamma \sum_{s'} \mathcal{P}^{a}_{ss'} \max_{a'} \boldsymbol{Q}^{n}(s',a') \end{aligned} \tag{11}$$

if the level of surprise δ is **large enough**, *all possible predecessors* of s are added to a priority queue *PQueue*, with a priority p :

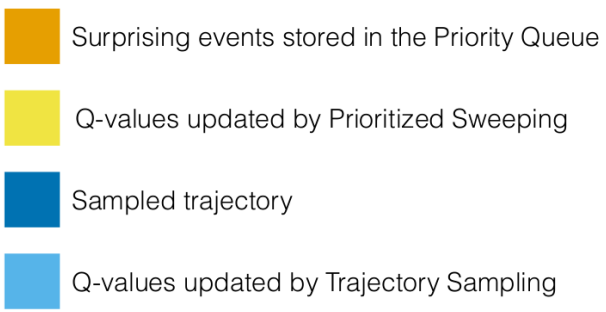
- equal to δ,
- attenuated by γ,
- and attenuated by the probability to effectively reach state s from the predecessor s' under consideration : $T(s', a', s)$

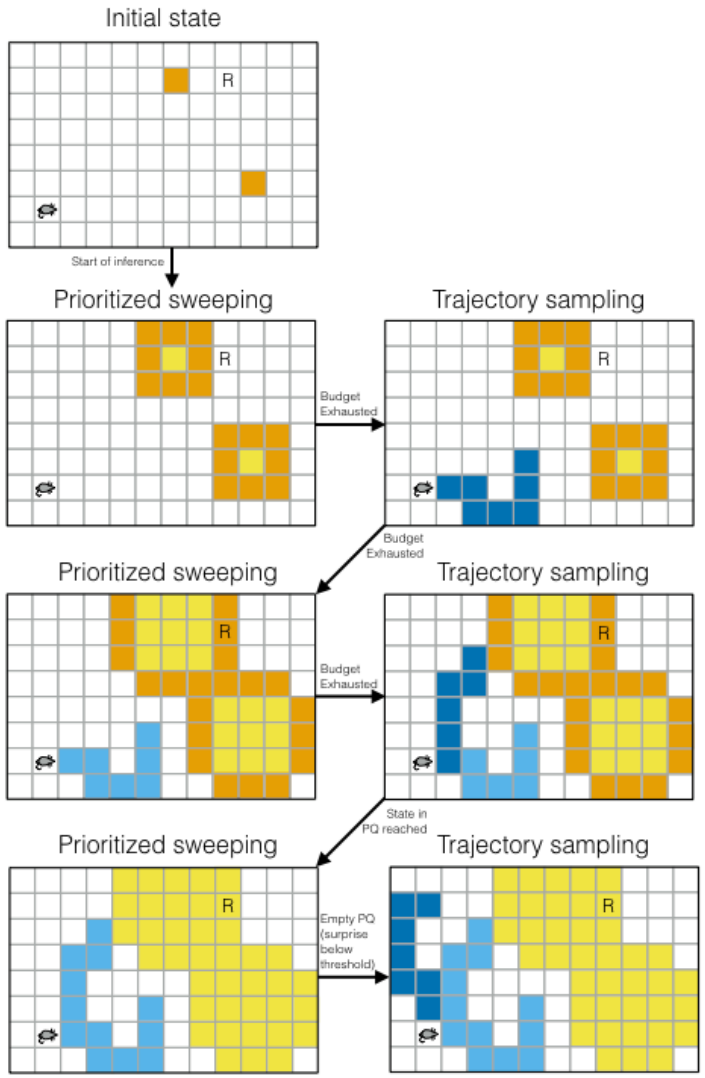
$$p \leftarrow \delta \times \gamma \times T(s', a', s) \tag{12}$$

Trajectory Sampling (TS) :

- TS drives forward replay from the *current location*.
- TS generates trajectories(with transition state function).
- TS stops when a connection state with backward search (Prioritized Sweeping) is reached.

Bidirectional Algorithm





Offline alternation between

- Prioritized Sweeping,
- Trajectory Sampling.

loop until $n_{a_{max}}$ *actions* **while** $n_a = n_{a_{max}}$ **do**

$a_t \leftarrow softmax_\beta(Q(s_t, a_t))$ | Take action a_t receive s_{t+1}, r_t,

Update $\mathcal{P}^{a_t}_{s_t s_{t+1}}$ | $R(s_t, a_t) \leftarrow r_t$ | Store $Q_{old} \leftarrow Q(s_t, a_t)$

$Q(s_t, a_t) \leftarrow R(s_t, a_t) + \gamma \sum_{s'} \mathcal{P}^{a_t}_{s_t s_{t+1}} \max_k Q(s', k)$

Compute the **level of surprise** : $\delta = |Q(s_t, a_t) - Q_{old}|$

reach predecessors (s, a) *of* s_t **for** *each* $(\boldsymbol{s, a})$ *so that* $\boldsymbol{\mathcal{P}^a_{ss_t}} \neq 0$ **do**

Compute **predecessors priority** : $p \leftarrow \delta \times \gamma \times \mathcal{P}^a_{ss_t}$

if $(s, a) \notin PQueue$ **then** : Put (s, a) in $PQueue$ with *priority* p

else : Update priority of (s, a) in $PQueue$ with p

end

$s_t \leftarrow s_{t+1}$ and $n_a = n_a + 1$

Offline MB Q-update $Q \leftarrow OfflineBidirectional(Q, PQueue, s_t)$

end

Algorithm 1: Online bidirectional algorithm

$nbLoops \leftarrow 0$
repeat
  Prioritized Sweeping (PS) **while** $PQueue[0] > thresold$ | $max\ budget$ **do**
    Select **highest prioritized tuple :** $(s, a) \leftarrow PQueue[0]$
    MB **Q Update** on $Q(s, a)$
    Update (s, a) **predecessors priority** : **for** *each* (s', a') ... **do ...**
  end
  TS start from the **current location** in the environment : $s \leftarrow s_t$
  Trajectory Sampling (TS) **while** $s \notin PQueue$ | $max\ budget$ **do**
    $a \leftarrow softmax_\beta(Q(s))$ | $s' \leftarrow$ *the* ***best*** *probability of* $T(s, a, s')$
    $Q_{old} \leftarrow Q(s, a)$ | $Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} \mathcal{P}^a_{ss'} \max_k Q(s', k)$
    Sum of ***TD error*** : $Sum\delta \leftarrow Sum\delta + |Q(s, a) - Q_{old}|$
    Jump to the ***next state*** : $s \leftarrow s'$
  end
until *TD error* ***convergence*** *or* ***budget max***;
return Q

Algorithm 2: Offline bidirectional algorithm

Table of Contents

State of the Art

SR can represent small discrete Tabular State Space
=> Unsuitable for big states space like video-games environment.

=> Need **approximate functions** to reduce the original state space
=> The resulting reduced **latent** representation can lead to **faster learning** because information can be **re-used** across different inputs

Successor Features (SFs) combine Successor Representation (SR) with arbitrary state representations ϕ. SFs are column vectors (Kulkarni 2016) :

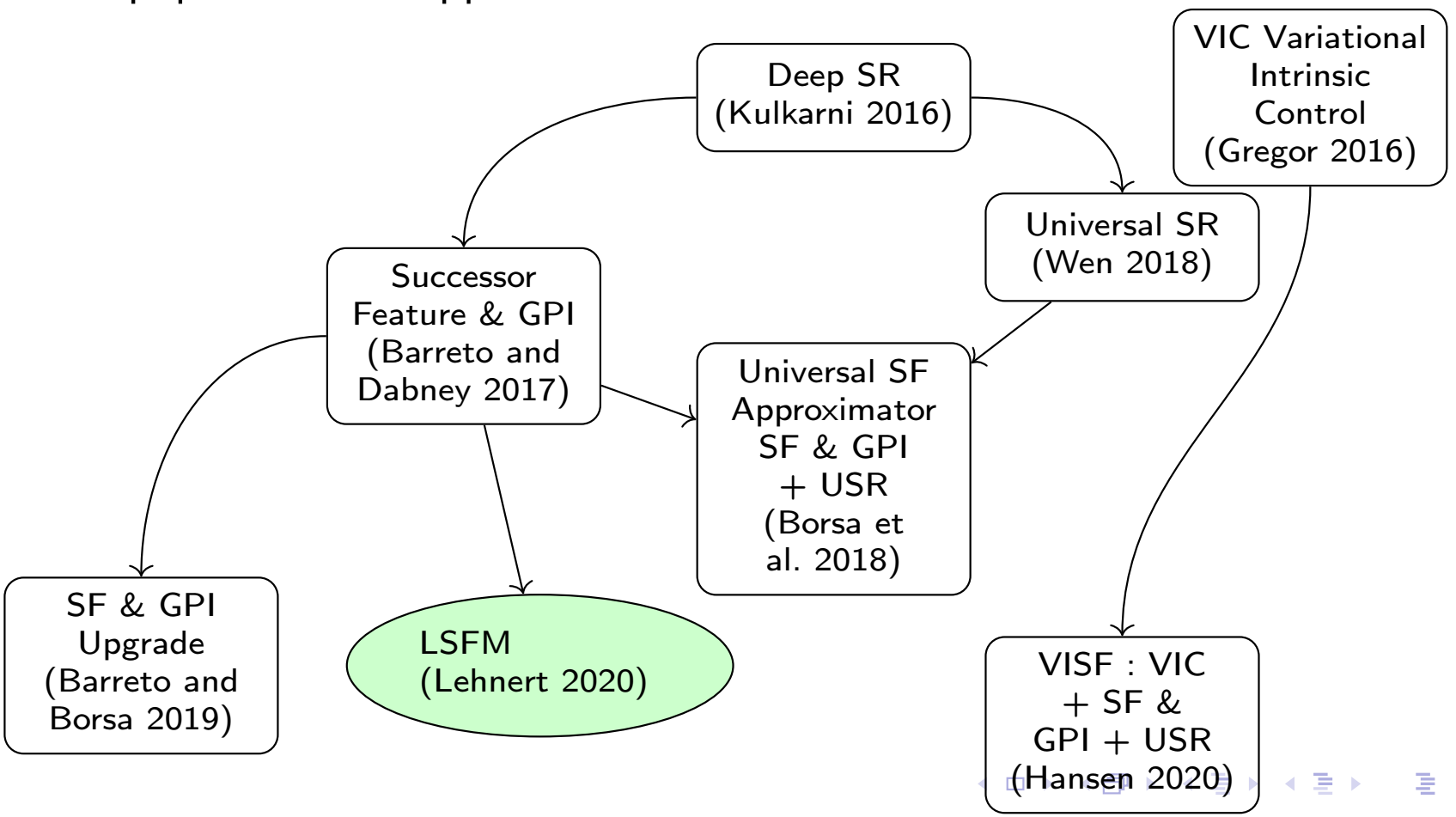
$$\boldsymbol{M}^{\pi}(s', s, a) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^{t} \mathbb{I}\{s_t = s'\} | s_0 = s, a_0 = a\Big] \tag{13}$$

$$\boldsymbol{\psi}^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\phi}_{st} \middle| s_1 = s, a_1 = a\right], \tag{14}$$

$\boldsymbol{\psi}^{\pi}$: frequency of different latent states vectors encountered following π.

Literature

Some papers on SR Approximation functions in Neural Network.



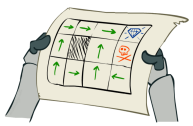
Latent space

Latent state space S_ϕ is constructed using a state representation function :

$$\phi : S \rightarrow S_\phi \tag{15}$$

S_ϕ is **value-predictive** **reward-predictive** if it retains enough information to support

- accurate $\underbrace{V^\pi \text{ or } Q^\pi \left(\mathbb{E}_\pi \left[\sum_k \gamma^k r_k\right]\right)}_{\textbf{value or Q-value}}$ predictions
- accurate $\underbrace{\textbf{future expected reward}}_{\mathbb{E}[r_1, r_2, \ldots | s, a_1, a_2, \ldots]}$ predictions

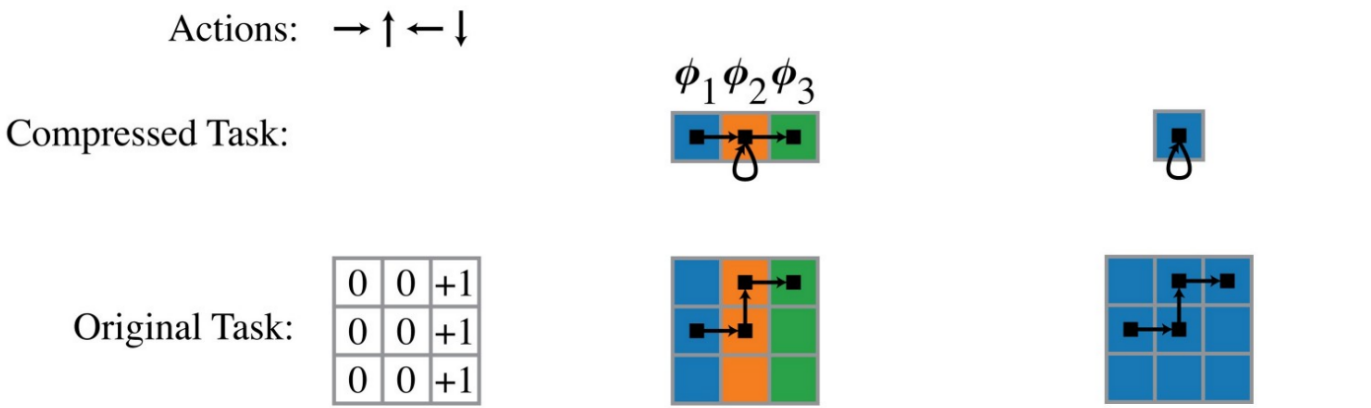


- **Value-predictive** state representation is **policy dependent**.
- **Reward-predictive** state representation **is not**.

For optimal policy π^*, **value-predictive** state representation becomes a

Reward-maximizing state representation

(s, a, r, s') experiences on $s_0 = s_{21}, \rightarrow, \uparrow, \rightarrow$ with Reward **Predictive** and **Maximizing** state representations (Lucas Lehnert n.d.)



(A) Column World Task

(B) Reward-Predictive State Representation

(C) Reward-Maximizing State Representation

Original State				Reward-Predictive				Reward-Maximizing			
$(s_{21}$,	$\rightarrow$,	0,	$s_{22})$	$(\phi_1$,	$\rightarrow$,	0,	$\phi_2)$	$(\phi$,	$\rightarrow$,	0.5,	$\phi)$
$(s_{22}$,	$\uparrow$,	0,	$s_{12})$	$(\phi_2$,	$\uparrow$,	0,	$\phi_2)$	$(\phi$,	$\uparrow$,	0,	$\phi)$
$(s_{12}$,	$\rightarrow$,	1,	$s_{13})$	$(\phi_2$,	$\rightarrow$,	1,	$\phi_3)$	$(\phi$,	$\rightarrow$,	0.5,	$\phi)$

Latent space and Successor Features (SFs)

Reward-maximizing state representations of (Barreto and Dabney 2017) are directly the SFs (ψ^π) :

$$Q^\pi(s,a) = \psi^\pi(s,a).\boldsymbol{w} \tag{16}$$

Latent space ϕ is one-step rewards : $\boldsymbol{r}(s,a,s') = \phi(s,a,s').\boldsymbol{w}$.

Reward-predictive state representations (ϕ_s) of **Linear Successor Features Model** (LSFM) are tied with SFs (ψ^π) via linear matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ (Lehnert 2020) :

$$\phi_s^\top \boldsymbol{F}_a \approx \psi^\pi(s,a), \tag{17}$$

With a LSFM $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$, a **Linear Action Model** (LAM) $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ can be constructed as

$$\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}}, \text{ where } \overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \boldsymbol{F}_a \tag{18}$$

A LAM $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ model the empirical latent transition probabilities :

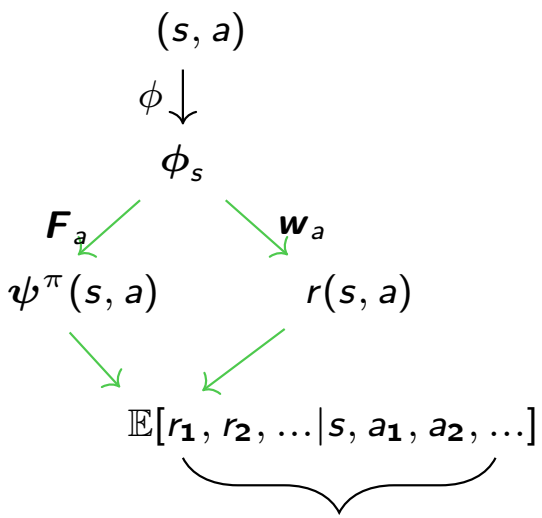
$$\phi_s^\top \boldsymbol{M}_a = \mathbb{E}[\phi'_s|s,a], \quad \text{and} \quad \phi_s^\top \boldsymbol{w}_a = \mathbb{E}[r(s,a,s')|s,a] \tag{19}$$

ϕ is (multi) **Reward-predictive** :

$$\mathbb{E}[r_t|s,a_1,...,a_t] \approx \phi_s^\top \boldsymbol{M}_{a1}...\boldsymbol{M}_{at-1}.\boldsymbol{w}_a \tag{20}$$

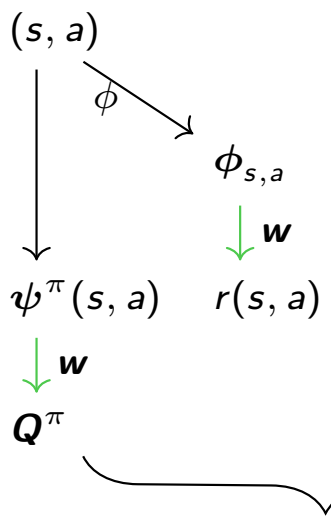
Value and **Reward** state representation models :

LSFM (Lehnert 2020)

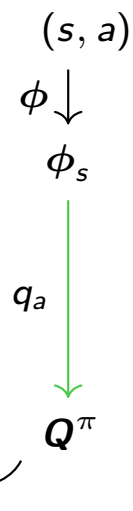


Reward-predictive Model

SF (Barreto and Dabney 2017)



QN (Riedmiller 2005)



Value-predictive Models

Learning

LSFM Learning

$$\mathcal{L}_{LSFM} = \underbrace{\sum_{i=1}^{D} \left(\boldsymbol{\phi}_{s_i}^{\top} \boldsymbol{w}_{a_i} - r_i\right)^2}_{=\mathcal{L}_r} + \alpha_\psi \underbrace{\sum_{i=1}^{D} \left\| \boldsymbol{\phi}_{s_i}^{\top} \boldsymbol{F}_a - \vec{\boldsymbol{y}}_{s_i,a_i,r_i,s_i'} \right\|_2^2}_{=\mathcal{L}_\psi} + \underbrace{\alpha_N \sum_{i=1}^{D} \left(\left\| \boldsymbol{\phi}_{s_i} \right\|_2^2 - 1 \right)^2}_{=\mathcal{L}_N} \quad (21)$$

and α_N, $\alpha_\psi > 0$ are hyper-parameters.

$$\vec{\boldsymbol{y}}_{s,a,r,s'} = \boldsymbol{\phi}_s^{\top} + \gamma \boldsymbol{\phi}_{s'}^{\top} \overline{\boldsymbol{F}} \quad (22)$$

$\overline{\boldsymbol{F}}$ the LSFM that predicts the SF for a policy **uniformly at random** :

$$\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \boldsymbol{F}_a \quad (23)$$

Q Learning

A theorem (Lehnert 2020) bounds the prediction error of finding a linear approximation of the Q-function :

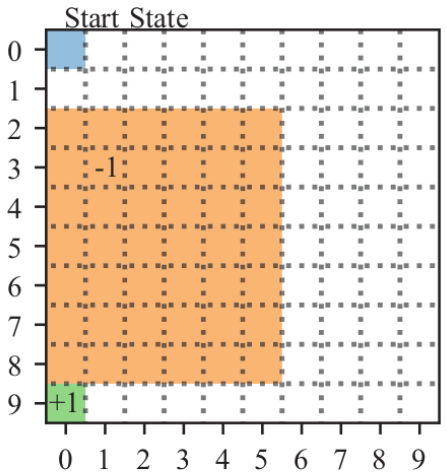
$$Q^{\pi}(s, a) \approx \phi_s^{\top} \boldsymbol{q}_a$$

using a **state representation** ϕ and a **real valued vector** $\boldsymbol{q}_a$.

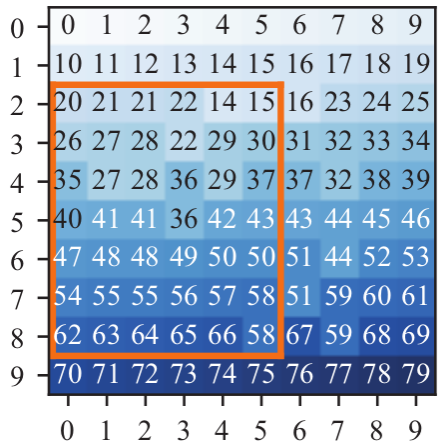
Learned Representation (Lehnert 2020)

Puddle Word /

Agglomerating clustering for LSFM learned representation



(a) Puddle-world task map

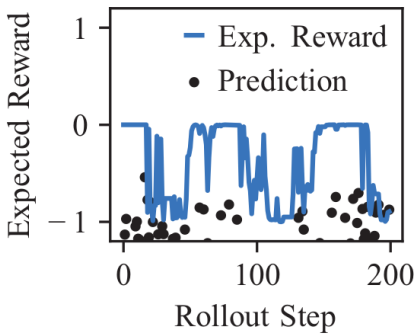


(c) LSFM generalization map

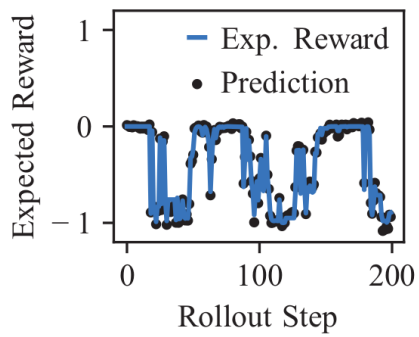
Expected reward and predictions
on random action sequence

Random representation /

LSFM representation

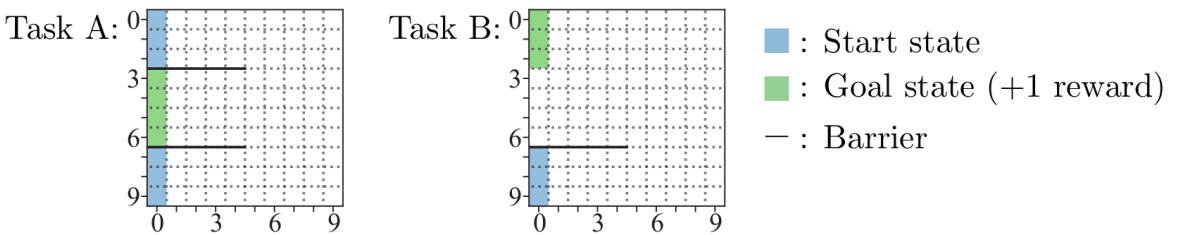


(d) Example Rollout, Random

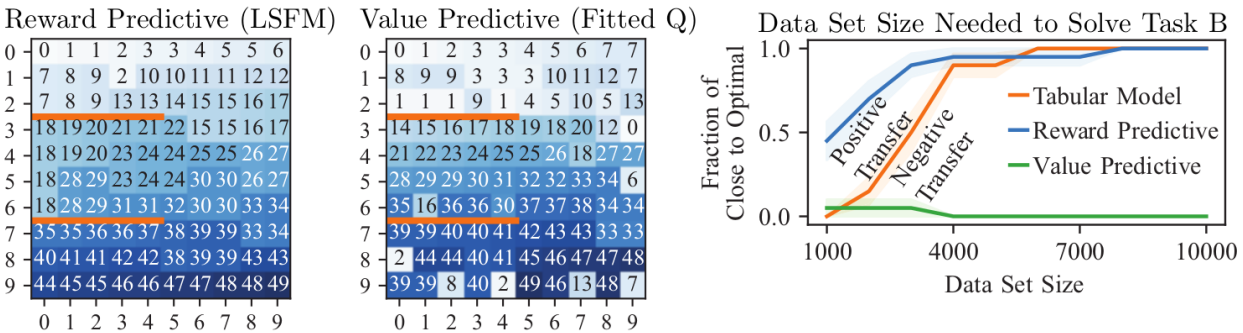


(f) Example Rollout, LSFM

Learned transfer (Lehnert 2020)



(a) Maps of Transfer Grid Worlds



(c) State partitions obtained through learning on a transition data set $\mathcal{D}$.

LSFM representation allow **faster transfer learning** than tabular model-based agent and Value Predictive representation.

value iteration construction on optimal policy

Table of Contents

Conclusion & future work

Conclusion

LSFM is **Reward-predictive**

LSFM state representations **generalize** across variations in **transition** and **reward** functions
(in contrast to (Barreto and Dabney 2017), (Kulkarni 2016), ... in reward functions only).

LSFMs must be trained on pure exploration policy (reward agnostic).

The LSFM state representation can be used to predict the **value-function** of any policy, including the **optimal** policy.

=> A working **Neural Network LSFM** is implemented in TensorFlow 2.

Futur work

***

Goal : Construction of a RL model using offline generated **Replay** with LSFM **Successor Features**

for Video games (and Neuroscience ?) applications.

***

Replay State of the Art :
(Mattar-Daw 2018) and (Khamassi-Girard 2020) proposed mathematical and heuristic models to generate Replay in DYNA algorithm on original tabular state space.

Research directions :

=> Find new or adapt existing **Replay** models on Successor Features

=> Find better (than random) **exploration policy** for LSFM learning : *Options* of (Machado 2018) ?

Any ideas and helps is welcome 

Thanks for your attention !

References I

- Barreto, Andre and Diana Borsa (2019). *Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement*.
- Barreto, Andre and Will Dabney (2017). *Successor Features for Transfer in Reinforcement Learning*. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
- Borsa, Diana et al. (2018). "universal successor feature approximator." In:
- Dayan, Peter (1993). *Improving Generalization for Temporal Difference Learning: The Successor Representation*. Computational Neurobiology Laboratory, The Salk Institute, P.0. Box 85800, Sun Diego, CA 92186-5800 USA.
- Gershman, Samuel J. (2018). *The Successor Representation: Its Computational Logic and Neural Substrates*. The Journal of Neuroscience. ISBN: 9781417642595.

References II

- Gregor (Nov. 2016). "Variational Intrinsic Control." en. In: *arXiv:1611.07507 [cs]*. arXiv: 1611.07507. URL: http://arxiv.org/abs/1611.07507 (visited on 04/29/2021).
- Hansen, Barreto (2020). "Fast Task Inference with Variational Intrinsic Successor Features." In:
- Khamassi-Girard (2020). *Modeling awake hippocampal reactivations with model-based bidirectional search*. Biological Cybernetics (Modeling), Springer Verlag, 2020, 10.1007/s00422-020- 00817-x. hal-02504897.
- Kulkarni, Gershman (2016). *Deep Successor Reinforcement Learning*. ArXiv vol abs/1606.02396.
- Lehnert, Lucas (2020). *Successor Features Combine Elements of Model-Free and Model-based Reinforcement Learning*.
- Lin (1992). *Self-improving reactive agents based on reinforcement learning, planning and teaching*. Machine learning, 8(3):293–321, 1992.

References III

- Lucas Lehnert Michael L. Littman, Michael J. Frank (n.d.). "Reward-predictive representations generalize across tasks in reinforcement learning." In: ().
- Machado (2018). *Eigenoption discovery through the deep successor representation*. conference paper at ICLR 2018.
- Mattar-Daw (2018). *Prioritized memory access explains planning and hippocampal replay*. Nature Neuroscience.
- Mnih (2013). *Playing atari with deep reinforcement learning*. ICLR 2016arXiv:1312.2013.Mnih.
- – (2015). *Human-level control through deep reinforcement learning*. Nature, 518(7540):529–533, 2015.
- Momennejad, Ida and Marc W. Howard (Oct. 22, 2018). *Predicting the Future with Multi-scale Successor Representations*. preprint. Neuroscience. DOI: 10.1101/449470. URL: http://biorxiv.org/lookup/doi/10.1101/449470 (visited on 04/21/2021).

References IV

- Riedmiller, Martin (2005). "Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method." en. In: *Machine Learning: ECML 2005*. Ed. by David Hutchison et al. Vol. 3720. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 317–328. ISBN: 978-3-540-29243-2 978-3-540-31692-3. DOI: 10.1007/11564096_32. URL: http://link.springer.com/10.1007/11564096_32 (visited on 04/30/2021).
- Schaul (2016). *Prioritized experience replay*. ICLR.
- Wen, Chen Ma & Junfeng (2018). "universal successor representations for transfer reinforcement learning." In: *Workshop track - ICLR 2018*.