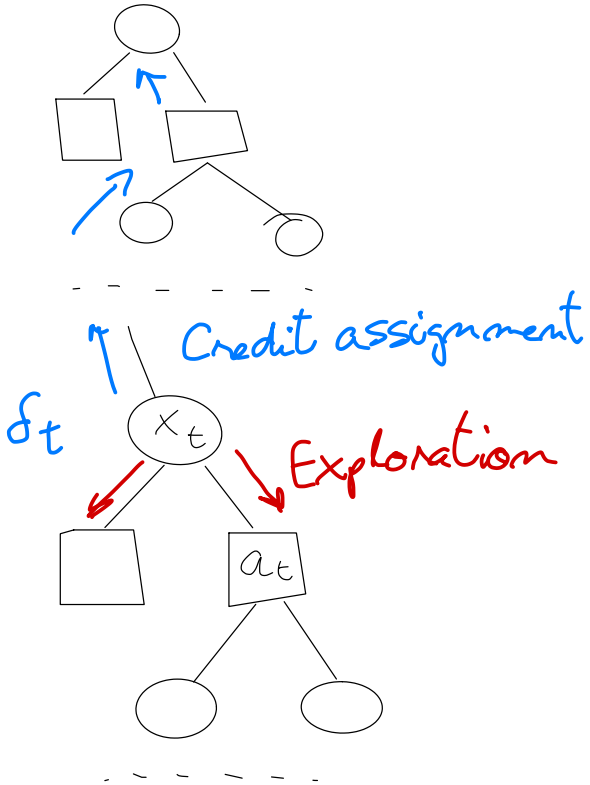
More on Continual (Never-Ending) Reinforcement Learning

COMP579, Lecture 26

Recall: Cartoon of sequential decision making

- At time t, agent receives an observation from set $\mathcal{X}$ and can choose an action from set $\mathcal{A}$ (think finite for now)
- Goal of the agent is to maximize long-term return



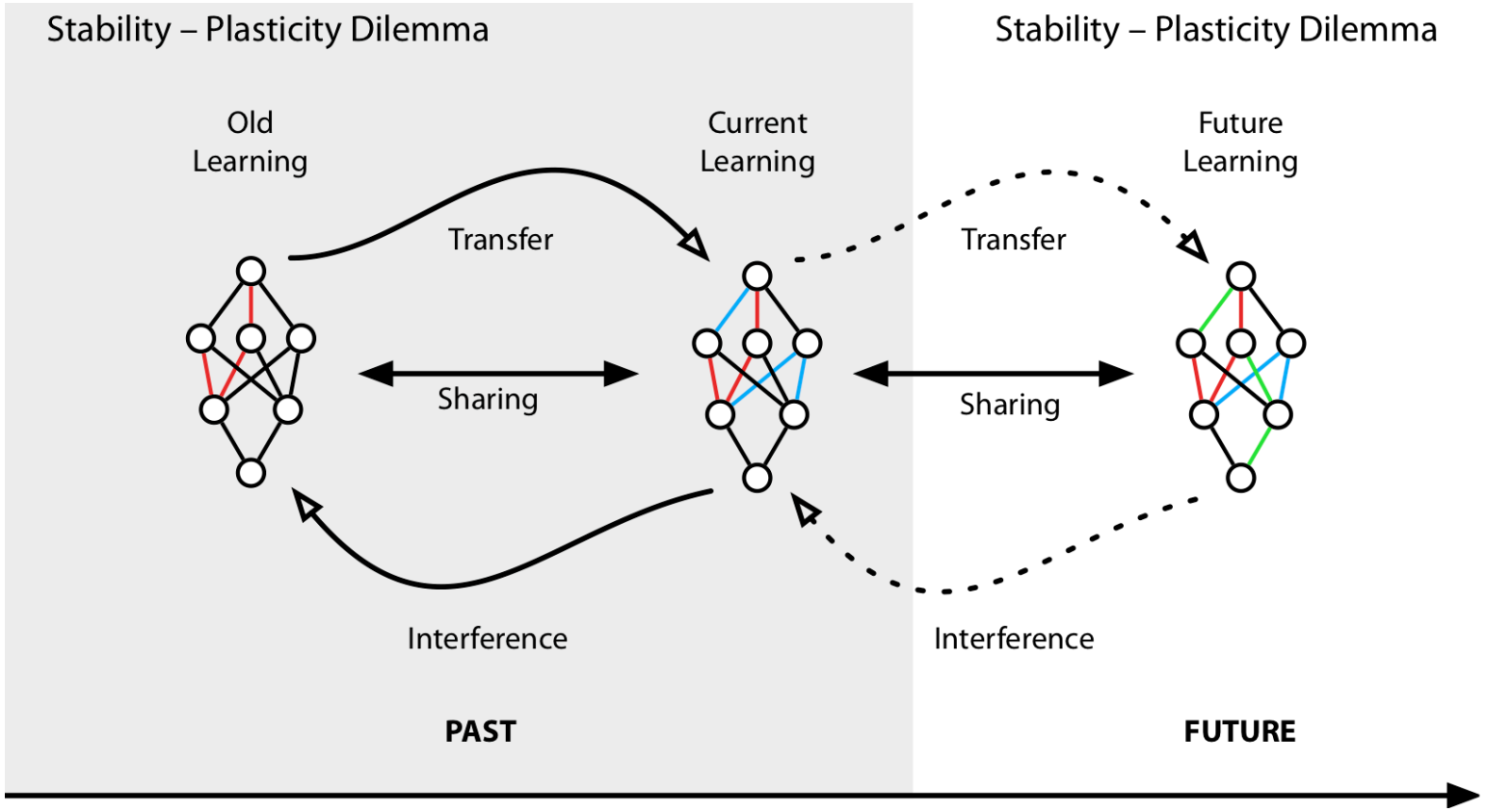
Recap from last time

- Ways of defining more general structure than MDPs
- Eg partial observability, more flexible timing of decisions
- Different notions of what is being optimized (still nebulos)

Algorithmic Approaches - Focus

- Fine-grained thinking about the agent's parameters
- Coarser approaches to split the agent's representation into fast and slow systems
- Even more high-level approaches to allow the agent control over its thinking process
- The first 2 categories are often labelled as dealing with *stability-plasticity dillema*
- The third one is often labelled as *HRL/search control*

Stability-plasticity dilemma

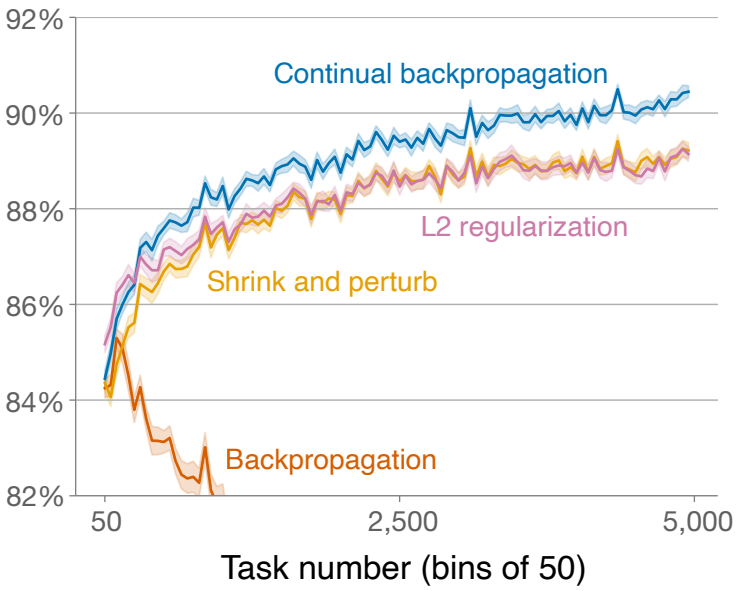


A lot of existing literature!

- DeepRL agents lose plasticity (cf Nikishin et al, 2022)
- Even deep supervised learning architectures lose plasticity (cf. Dohare et al, 2022)
- First category of solutions: weight-based

Example: Continual Backprop

%Correct
on Test Set
(at end of each task,
averaged over 30 runs)



- Dropout, Adam, other activations are even worse than vanilly backprop
- L2 regularization adds a penalty for large weights
- Shrink and Perturb is L2 reg. plus random variation of all weights
- Continual Backpropagation continually re-initializes a small fraction of units
  - otherwise its just like BackProp

Example: Continual Supervised Learning

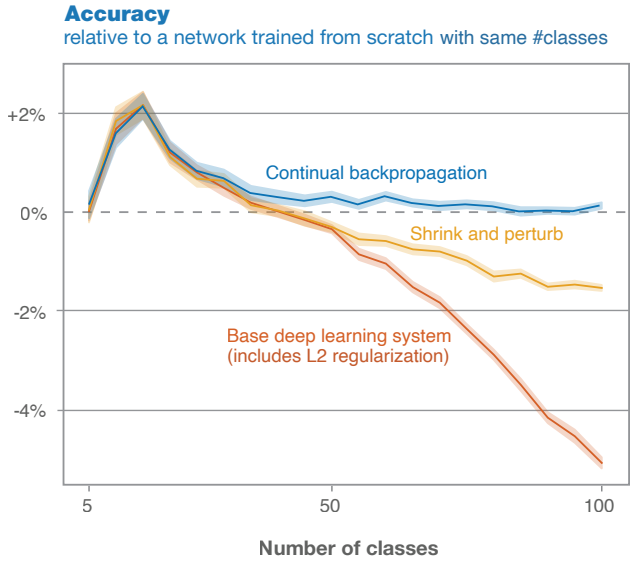
Problem

At the start, pictures from five classes have to be distinguished

five more classes are added, and pictures from all ten have to be distinguished

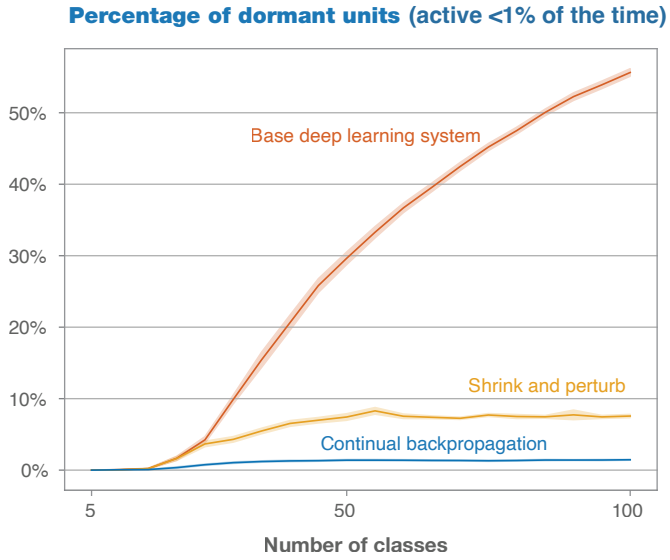
The process continues until finally all 100 classes have to be distinguished

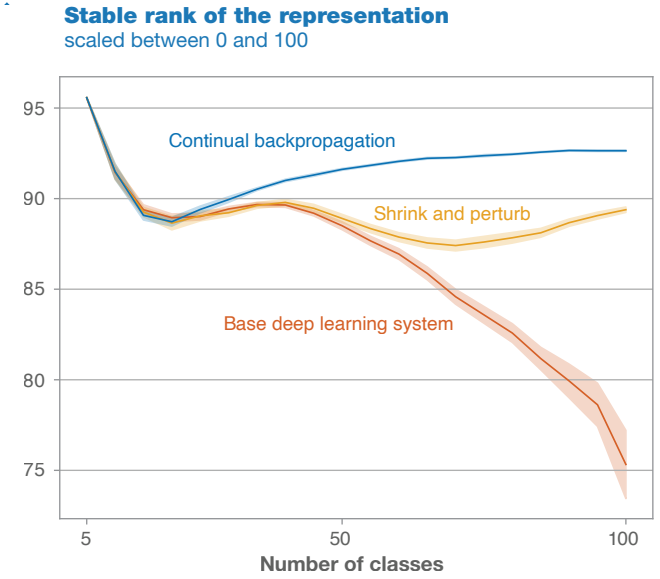
Results



Continual BackProp > Shrink & Perturb > L2 > Backprop

Example: Internal Representation

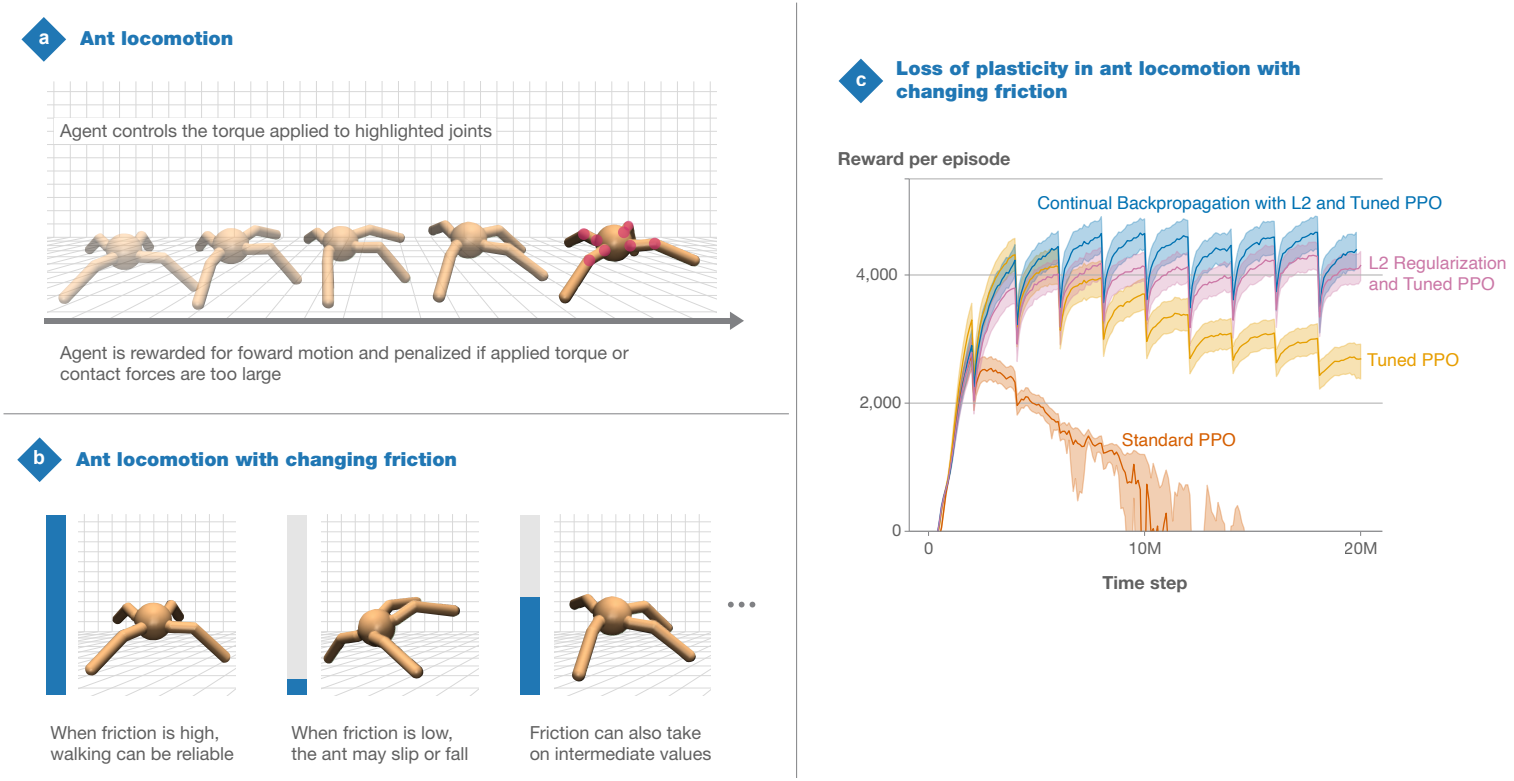




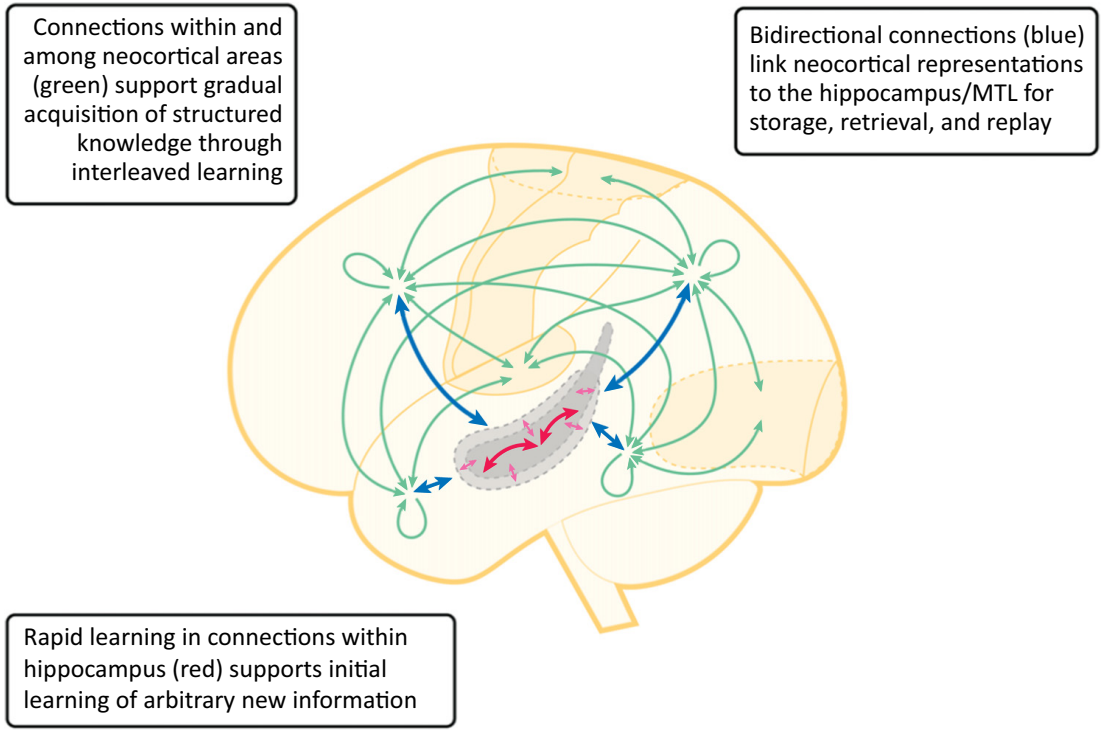
Many units go dormant unless random variation is continually injected

Causing a collapse of representational diversity

Similar Results in RL



Splitting at System Level: Complementary Learning Systems



Cf. Kumaran, Hassabis and McLelland, 2016

Simple RL Implementation

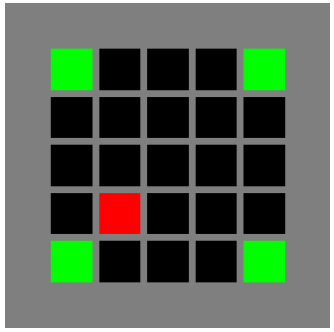
- Value function has two components:

$$V^{PT}(s) = V^P_\theta(s) + V^T_{\mathbf{w}}(s)$$

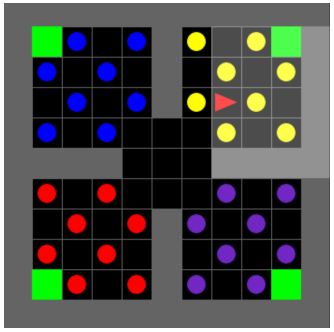
- *Permanent memory*: V^P should provide good estimates for any circumstances
- *Transient memory*: V^T should quickly compute corrections to V^P to adapt the the current distribution
- Both updated in parallel using TD-style updates

Cf. Anand and Precup, NeurIPS'2023

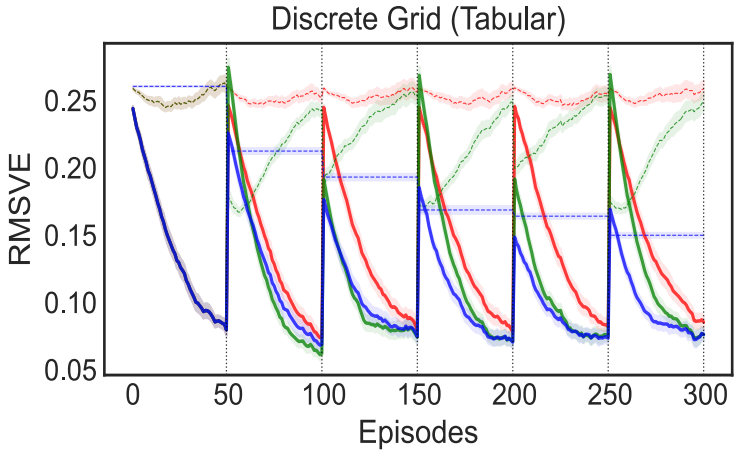
Prediction results

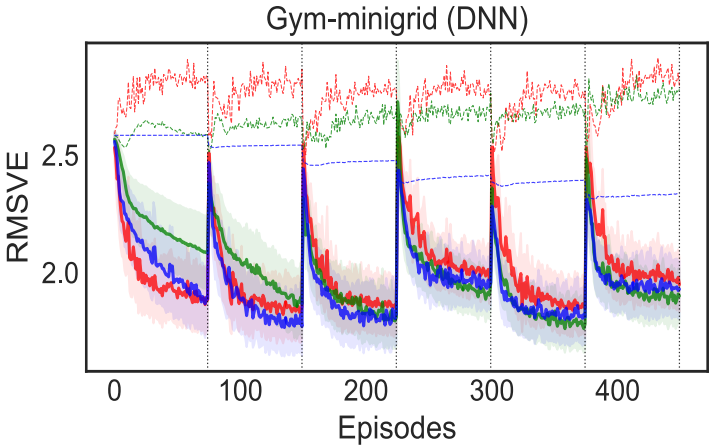


(a) Discrete grid.



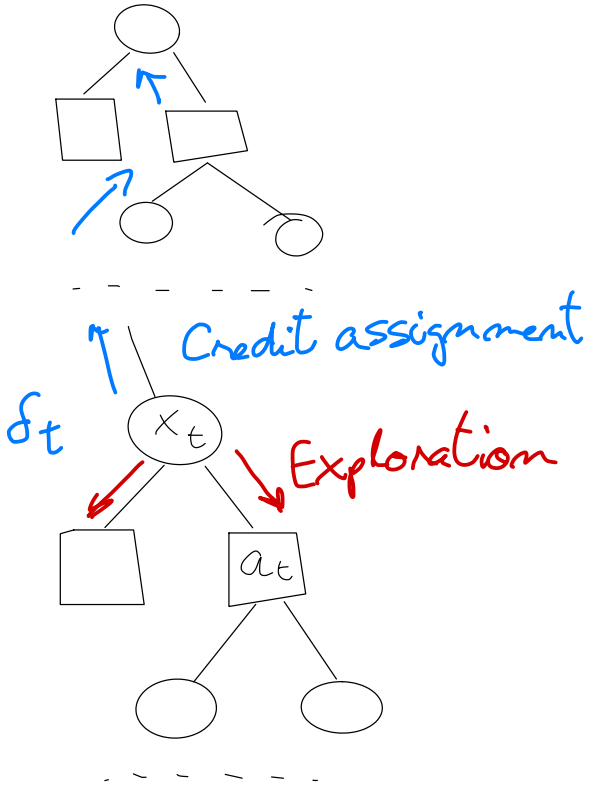
(b) Gym-minigrid.





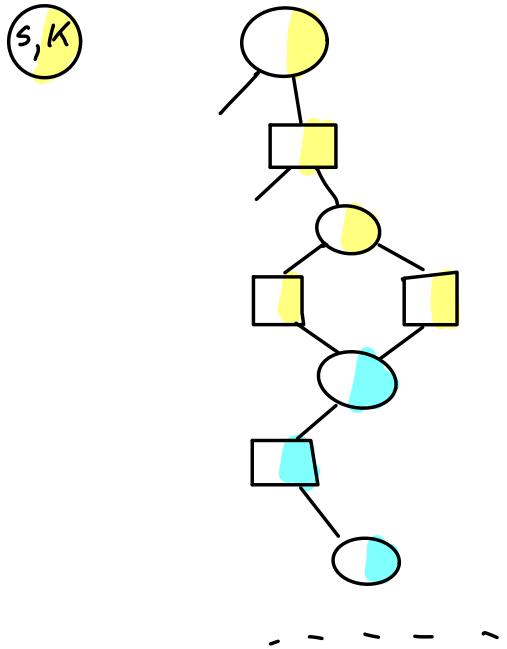
The agent retains knowledge while preserving plasticity!

Giving the agent control over its own thinking process



The agent can reshape the tree to help its decision making process

Example: Imagining multiple tasks or goals



- $x_t = \langle s_t, k_t \rangle$ where k_t is some way to specify a task at time t and s_t is the state inside the task
- *Task structure exists only in the agent's head, in order to make credit assignment and action choice easier*

Predictive knowledge: Value Function

- Given a policy π, a discount factor γ and a reward function r, the value function of the policy is given by:

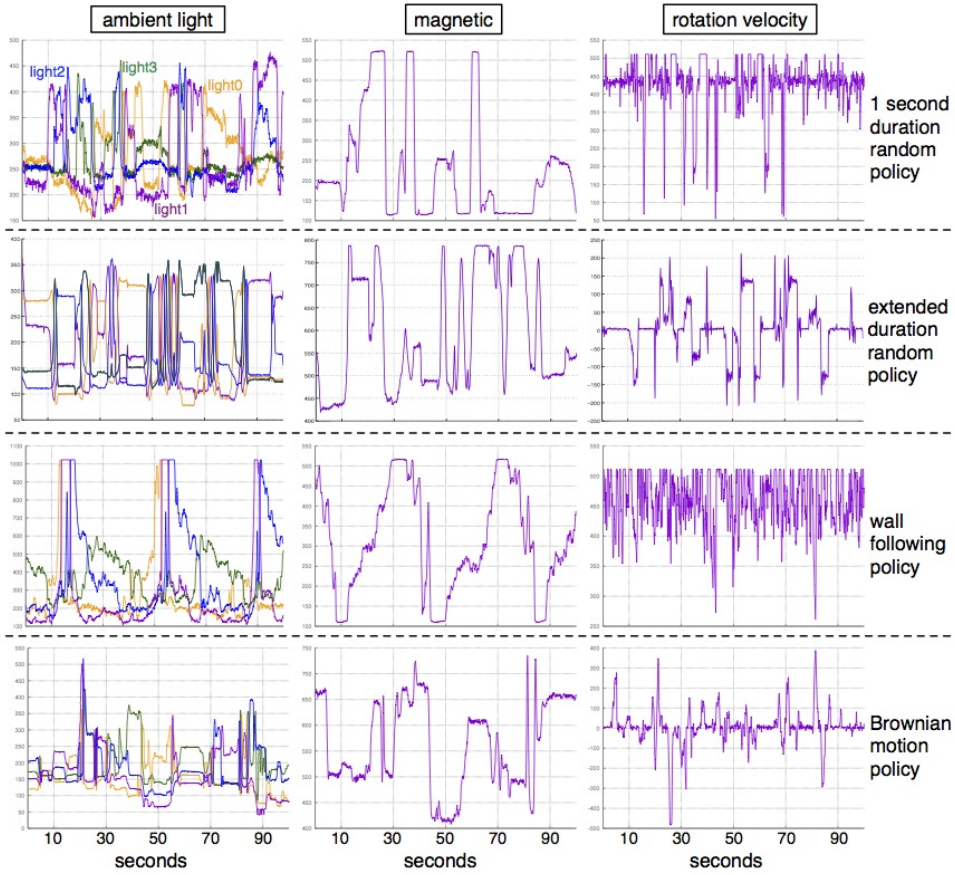
$$\begin{aligned} v_\pi(s) &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k)\gamma^{k-t} | S_t = s, A_{t:\infty} \sim \pi] \\ &= \mathbf{E}[\sum_{k=t}^{\infty} r(S_k, A_k) \prod_{i=t+1}^{k} \gamma | S_t = s, A_{t:\infty} \sim \pi] \end{aligned}$$

- r is the *signal of interest* for the prediction
- γ defines the time scale over which we want to make the prediction (in a very crude way)
- Optimal value function: given a discount factor γ and a reward function r, compute v_{π^*} and π^*, the optimal policy wrt γ, r

Focusing on value function

- Definition allows us to leverage great tools: *bootstrapping* (as in dynamic programming) and *sampling*
- We have good ideas for how to learn value functions from data using temporal-difference methods, off-policy learning...
- Usual objection: this is restricted to one reward function and usually a fixed time scale (discount)
- An agent may need to make predictions about many different things and at many different time scales

There are many things to learn! (Adam White's thesis)



Sensory stream of Critterbot robot about different sensors for different policies

Can we learn about all these signals in parallel from one stream of data?

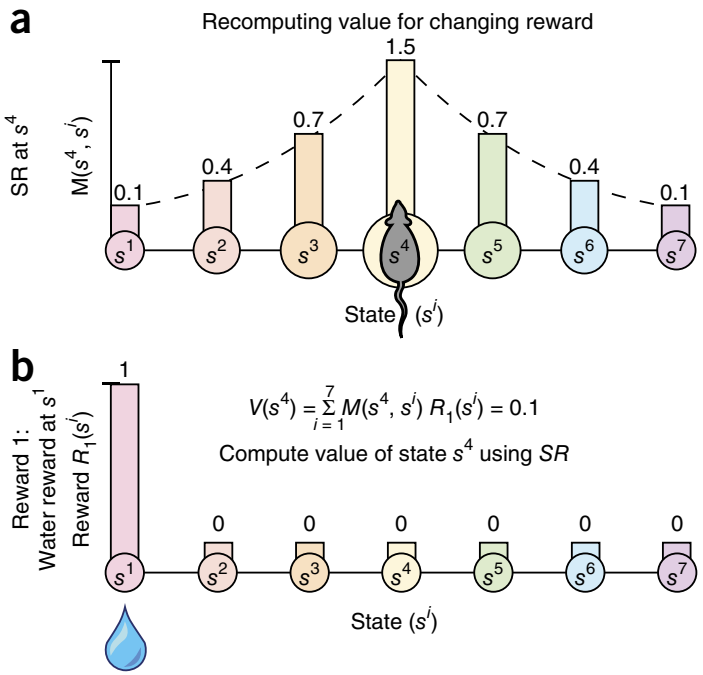
Temporally Abstract Predictions: General Value Functions (GVFs)

- Given a cumulant function c, state-dependent continuation function γ and policy π, the General Value Function $v_{\pi,\gamma,c}$ is defined as:

$$v_{\pi,c,\gamma}(s) = \mathbf{E}\left[\sum_{k=t}^{\infty} c(S_k, A_k, S_{k+1}) \prod_{i=t+1}^{k} \gamma(S_i) | S_t = s, A_{t:\infty} \sim \pi\right]$$

- *Cumulant* c can output a vector (even a matrix)
- *Continuation function* γ maps states to [0,1] (further generalizations are possible)
- Cf. Horde architecture (Sutton et al, 2011); Adam White's thesis; inspiration from Pandemonium architecture
- Special case: policy is optimal wrt c, γ, $v^*_{c,\gamma}$ - Universal Value Function approximation (UVFA) (Schaul et al, 2015)

Special case: Successor States



- Successor states (Dayan, 1992): expected occupancy of future states, for a given policy
- Allow the value function for *any reward* to be quickly computed
- Evidence linking to the hippocampus (Stachenfeld et al, 2018)

Special case: Successor Features

- *Successor features* (Barreto et al, 2017, 2018) are a natural extension of successor states
- If states are defined by a feature vector $\phi(s)$, successor features are GVFs where the *cumulant is* $c = \phi$, and there is a fixed policy and discount
- Interesting property highlighted in Barreto et al:

$$v_{\pi,\mathbf{w}^T c,\gamma}(s) = \mathbf{w}^T v_{\pi,c,\gamma}(s)$$

  which leads to one-shot computation of new GVFs

Special case: Option models

- The reward model for an option ω is defined as:

$$r_\omega(s) = \mathbb{E}_\omega[r(S_t, A_t) + \gamma(1 - \beta_\omega(S_{t+1}))r_\omega(S_{t+1})|S_t = s]$$

- This means the option reward model is a GVF:
  - policy is π_ω
  - *cumulant* is the environment reward r
  - *continuation function* is $\gamma(1 - \beta_\omega)$
- Option transition model can be similarly written as a GVF

Generalized Policy Updates

- *Generalized policy evaluation (GPE)*: compute the value of a policy π on a set of reward functions $\mathcal{R}$
- *Generalized policy improvement (GPI)*: given a set of policies Π and a reward function r, compute a new policy such that:

$$Q_r^{\pi'}(s,a) \geq \sup_{\pi \in \Pi} Q_r^{\pi}(s,a),\ \forall s \in \mathcal{S} \forall a \in \mathcal{A}$$

- If we have only one r and one π, we recover usual policy evaluation and policy improvement

Fast Generalized Policy Evaluation

- If we had a nice map from r to Q^π_r, GPE could be efficient
- Consider the class of reward functions that are linear in some feature space $\phi(s,a)$:

$$r_{\mathbf{w}}(s,a) = \mathbf{w}^T\phi(s,a) \text{ and } \mathcal{R}_\phi = \{r_{\mathbf{w}} | \mathbf{w} \in \mathbb{R}^d\}$$

  Note that ϕ can be learned and non-linear
- *Successor features*: $\psi^\pi(s,a) = \mathbf{E}_\pi[\sum_{t=1}^\infty \gamma^t \phi(s_t,a_t) | s_0 = s, a_0 = a]$
- Then the value function for a specified reward function can be easily computed as a function of the successor features:

$$Q^\pi_{\mathbf{w}}(s,a) = \mathbf{w}^T\psi^\pi(s,a)$$

- *Successor features can be pre-computed for π once and re-used thereafter (a form of model!)*
- Connections to hippocampus representations

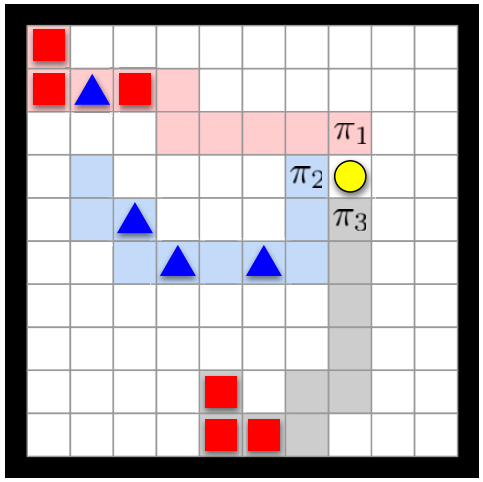
Fast Generalized Policy Improvement

- Compute the improved policy as:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \max_{\pi \in \Pi} Q_r^{\pi}(s, a)$$

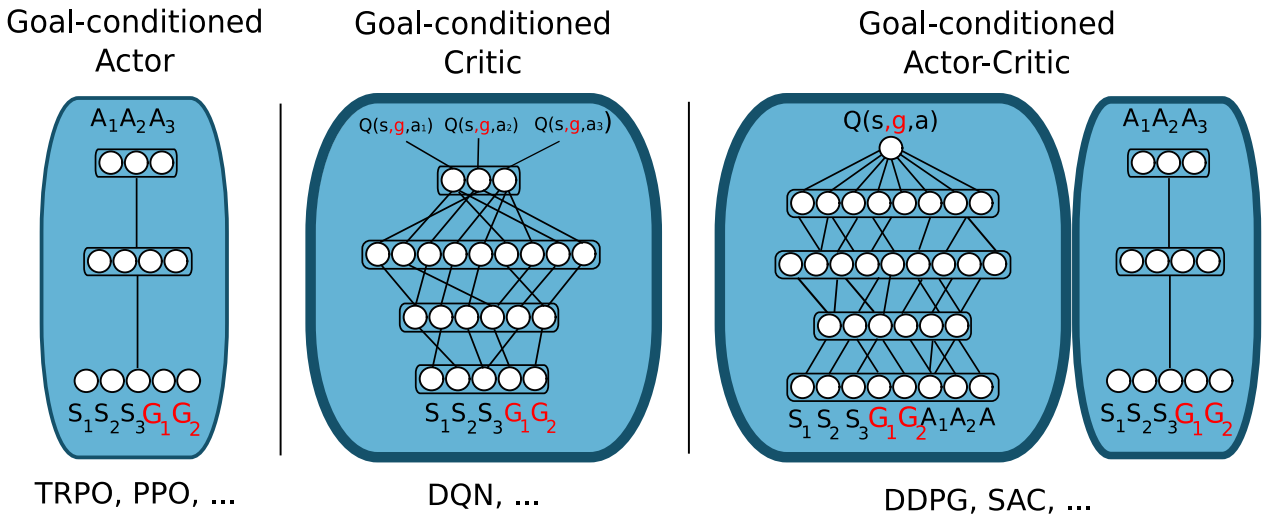
- Note that π' *could choose actions that are not chosen by any of the* π
- The process takes only *one iteration*, after which no further change to the policy π' would happen
- In contrast with iterative policy improvement...

Illustration



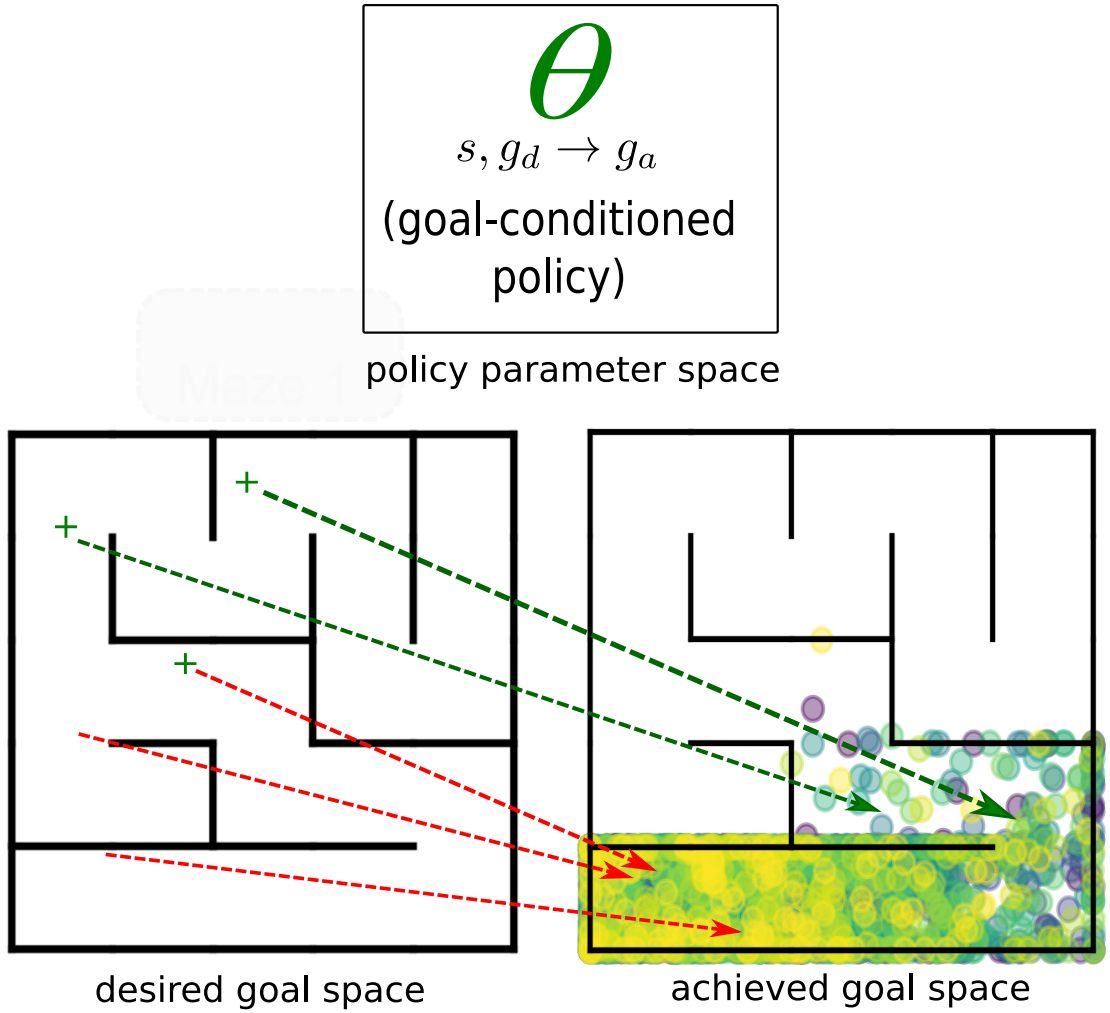
- The three policies correspond to three weight vectors: like red ($\mathbf{w}_1 = [1, 0]^T$), like blue ($\mathbf{w}_2 = [0, 1]^T$) and like red not blue ($\mathbf{w}_3 = [1, -1]^T$)
- *Note that $\mathbf{w}$ can be viewed as a preference function over features!*
- We can pre-train the policies that optimize for each preference, and train their successor features as well
- Then just do GPE/GPI!

Goal-conditioned RL



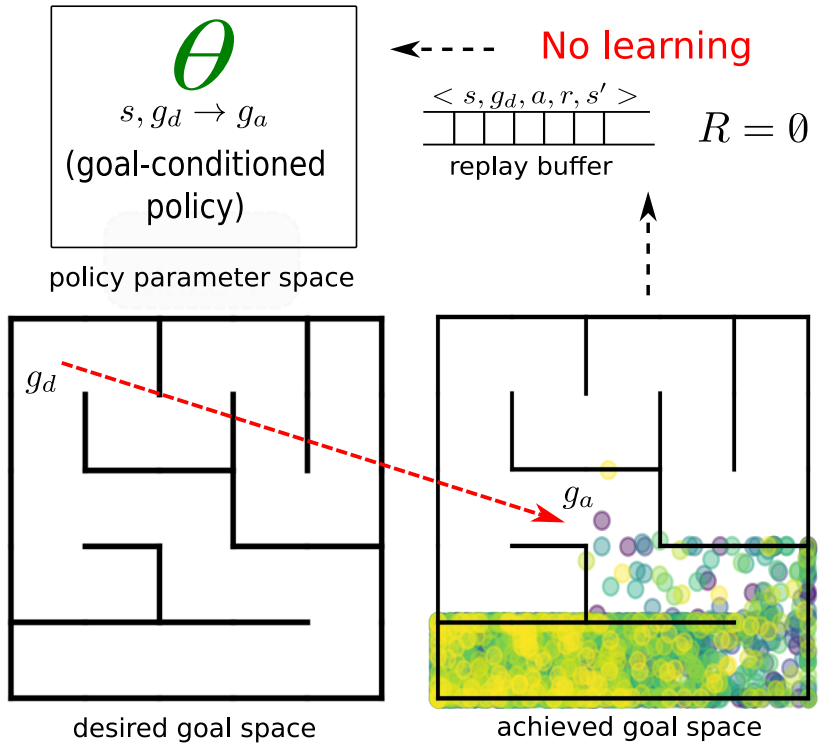
- Agent's goal becomes an input for RL
- Goal can be a state, or an embedding
- A function tells us whether goal has been achieved or not
- Can potentially learn to generalize over multiple goals!

Desired vs achieved goals



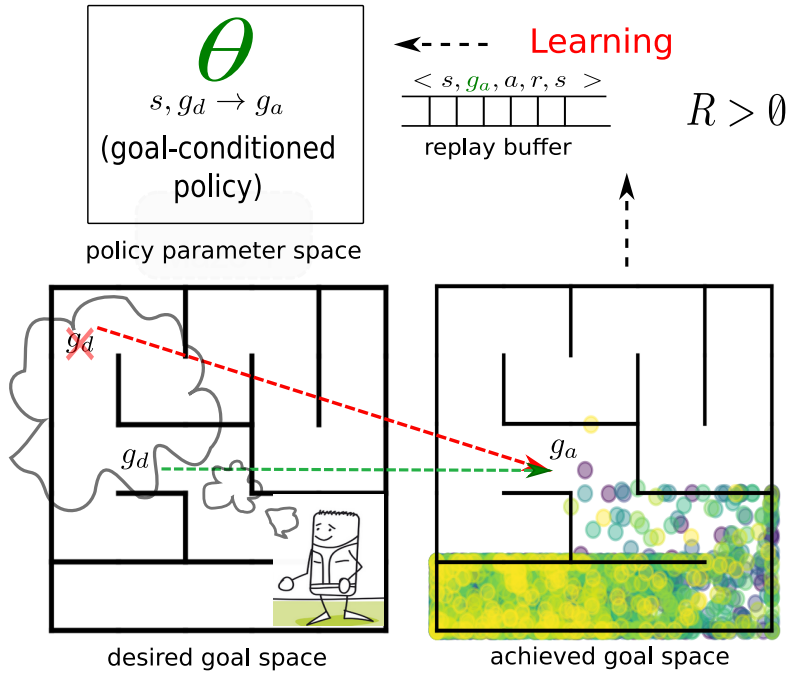
- Goal may be very hard to achieve!
- But maybe we can figure out what *was* achieved and learn from that?

Goal relabelling



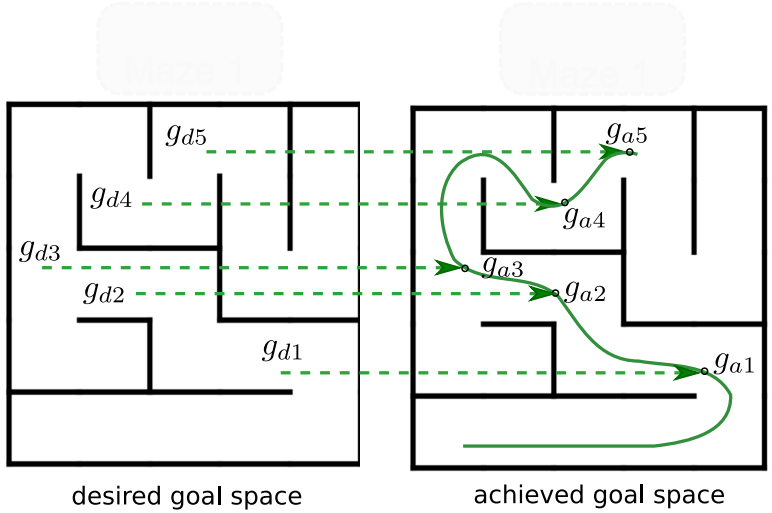
- The agent targets a desired goal g_d
- The policy π_θ produces an achieved goal g_a
- The trajectory is stored (it may produce no reward)

Hindsight experience replay (HER)



- The agent pretends it was targetting g_a
- HER relabels the stored trajectory with g_a instead of g_d
- This propagates value in the (state, action) space through generalization
- And the agent competence increases over unseen goals

When goals are states

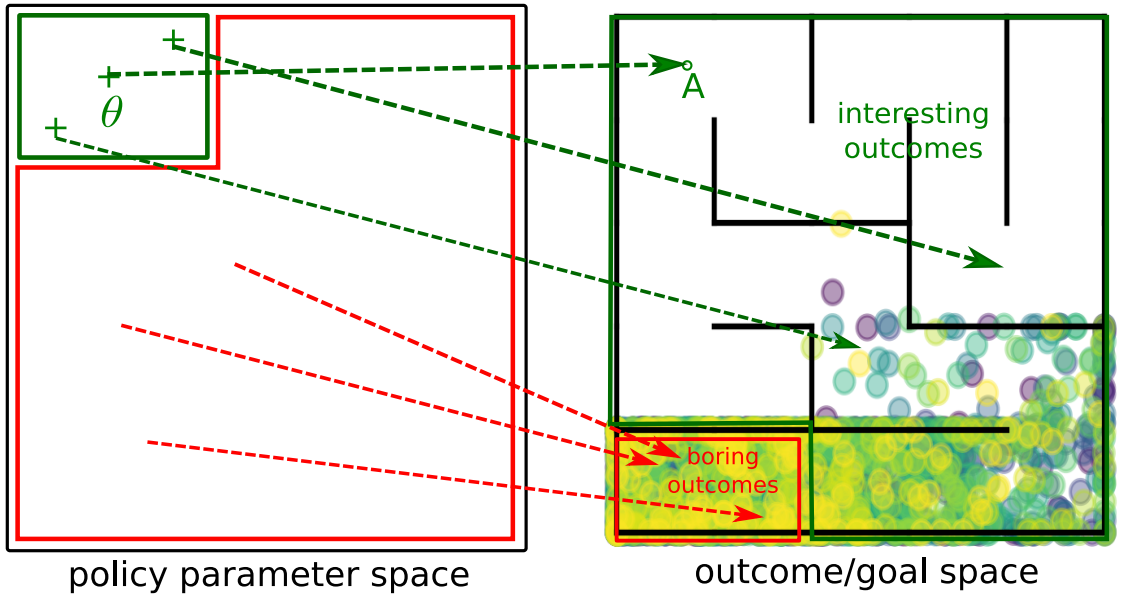


- If goal space = state space, HER may set as goal any state along the trajectory
- Trade-off between replaying more and trying more new actions (risk of over-fitting to replay)
- Variants of HER: CHER (Curriculum + HER), DHER (dynamic goals), MCHER (multi-criteria)...

From HER to curriculum learning

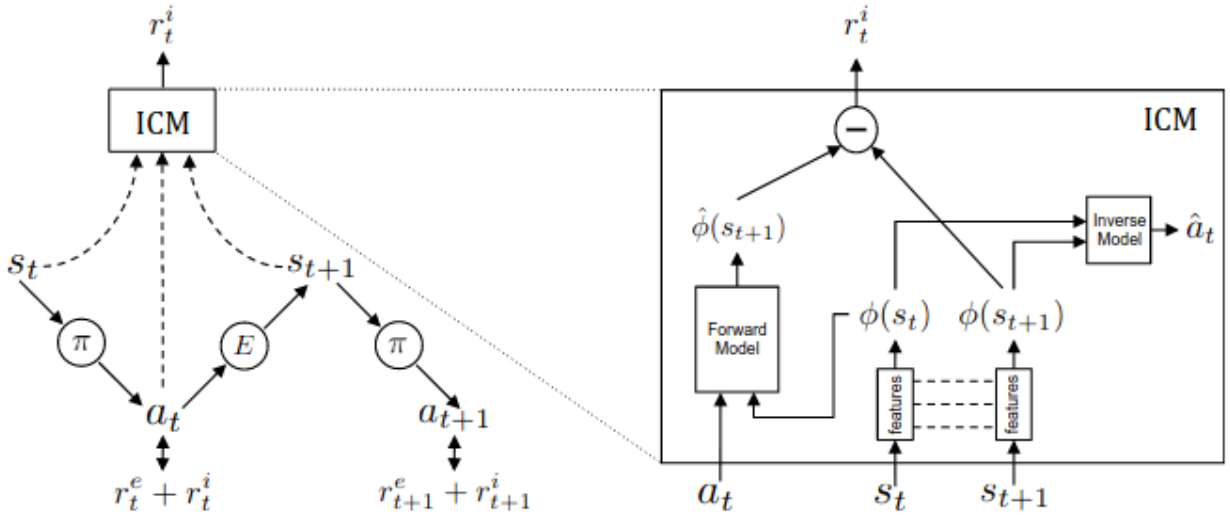
- HER relabels achieved goals as fake desired goals
- It does NOT tell you which goals the agent should strive to achieve
- *Curriculum learning* selects desired goals; can be combined with HER
- How should we select goals:
  - Coverage/Exploration: pick goals that let you get to more places
  - Performance: pick goals that you can actually accomplish

Goal exploration process



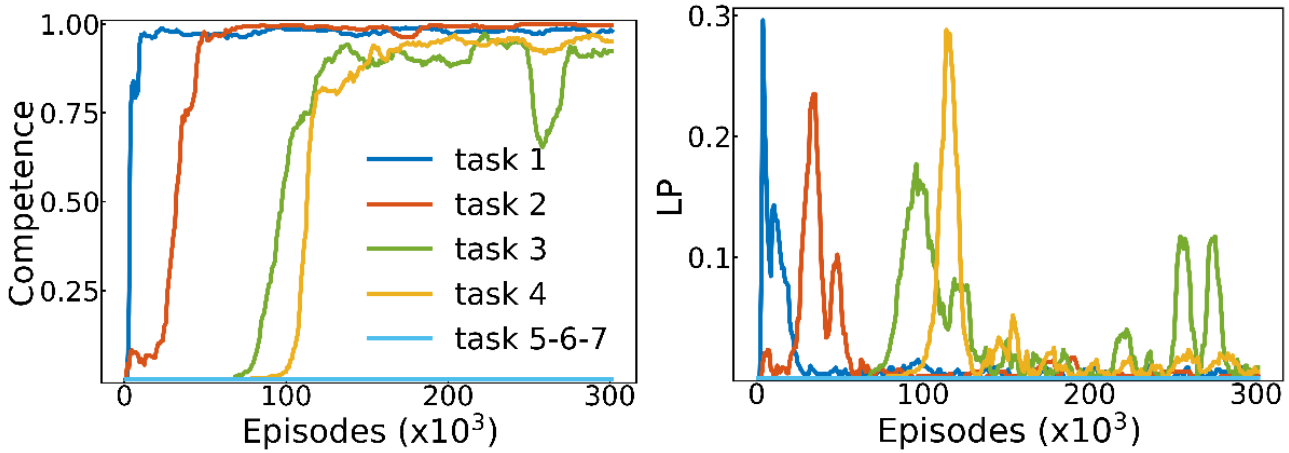
- Very often, few parameter vectors map to interesting achieved goals
- The GEP algorithm favors sampling these interesting achieved goals
  - Sample a random desired goal
  - Find the nearest achieved goal A' and select the corresponding θ
  - Perturb θ into θ' and get a new achieved goal A'
- Results in sampling "at the border" of currently achieved goals

Using surprise/novelty for curriculum



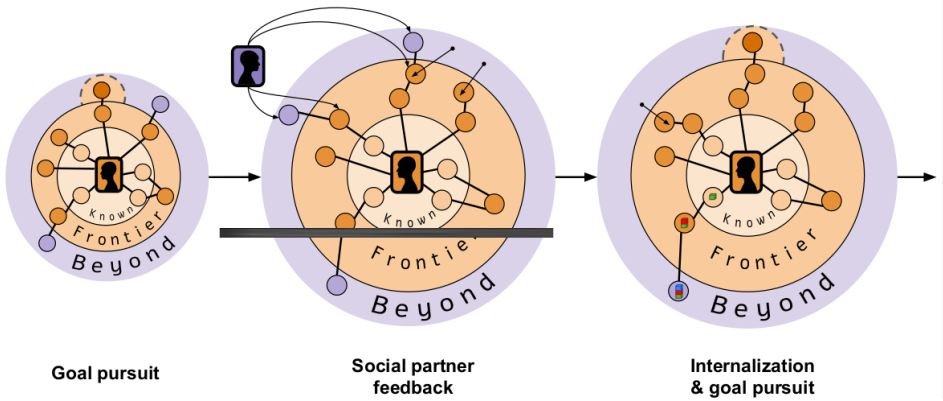
- Intrinsic motivation: reward states for which the forward model predicts poorly
- Target goals corresponding to rewarded states
- Results in visiting poorly visited states
- White noise problem: the agent may get stuck on what it cannot predict

Using learning progress

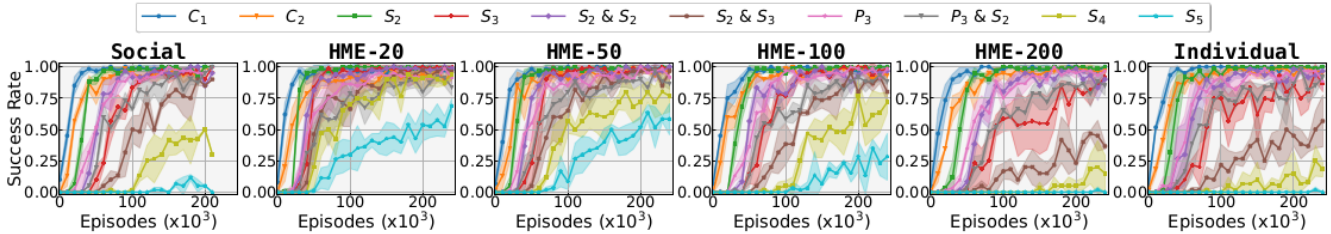


- Competence raises in order of growing difficulty
- LP generates more training in order of growing difficulty
- Catastrophic forgetting generates new training

Goal space can be learned



- Key property: the tutor has a model of the learner's knowledge
- It proposes Frontier + Beyond goals (HME)
- The learner internalizes tutor's goals, it can train on them and on its own goals

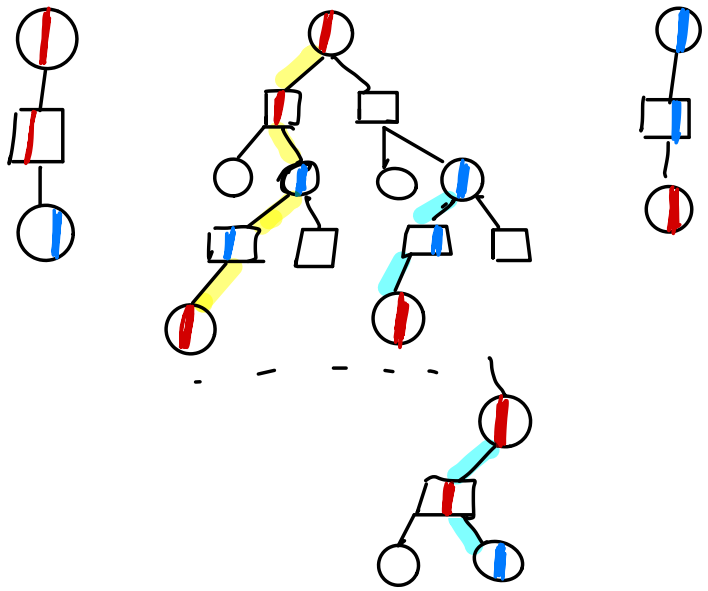


- Guided play is more efficient than learning on its own and full guidance

Summary so far

- HER allows the agent to learn from what it accomplishes
- Curriculum learning provides a good succession of goals
- Goals can help exploration and/or learning

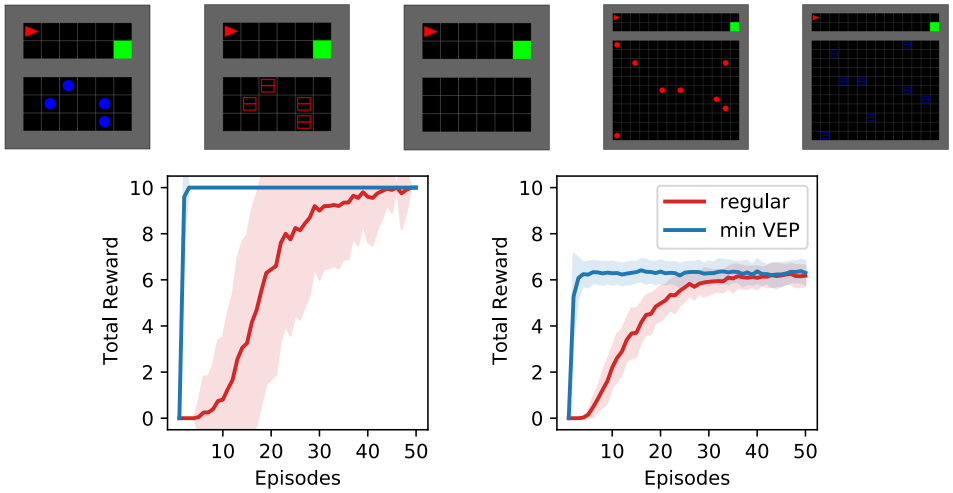
Partial Models



- Predict only specific features / cumulants
- Apply only in specific circumstances

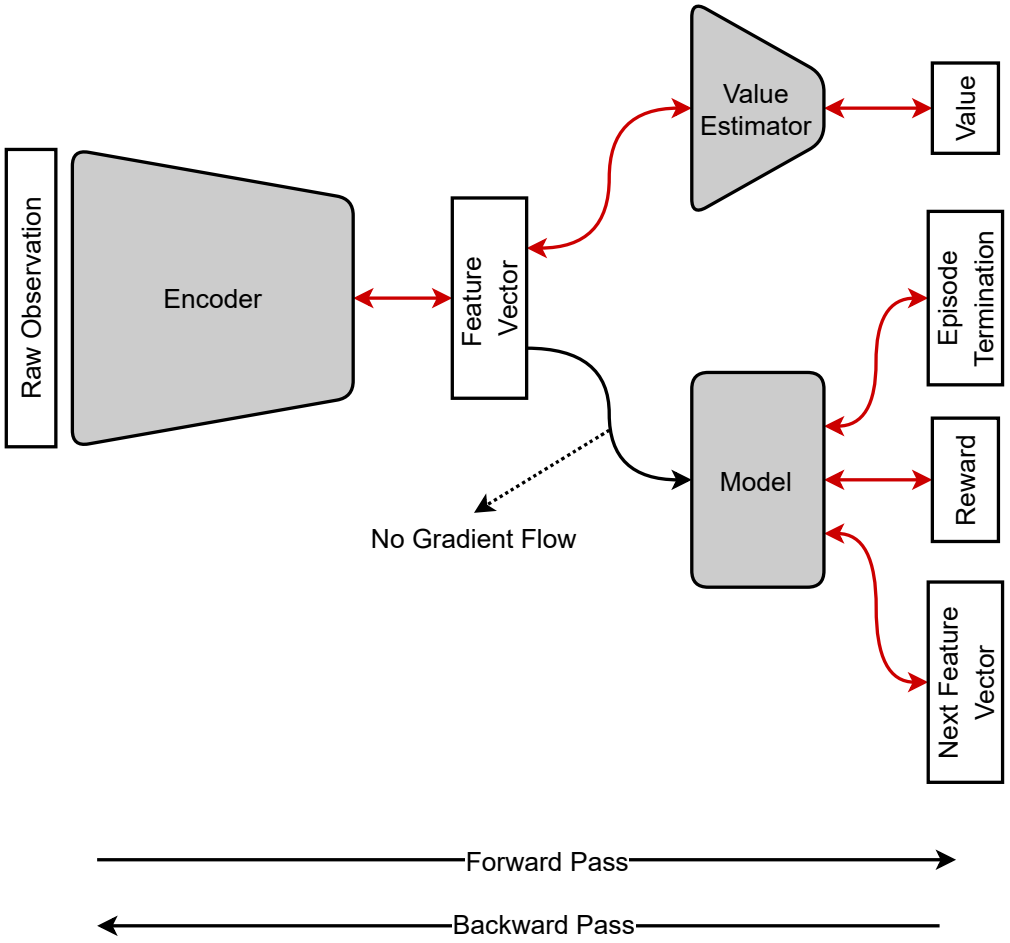
Partial Value-Equivalent Models

- Model only predicts a subset of features (not the entire observation) (cf. Talvitie & Singh, 2008)
- Goal is to obtain correct value estimates (a la MuZero), not to maximize likelihood
- Example: minigrid

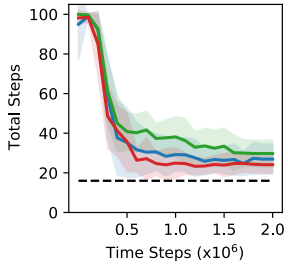


Partial models drastically improve solution speed! (cf Alver & Precup. 2023)

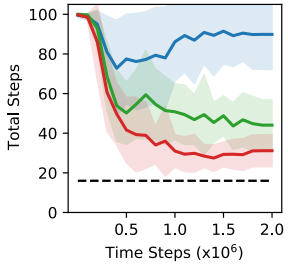
Learning Partial Value-Equivalent Models



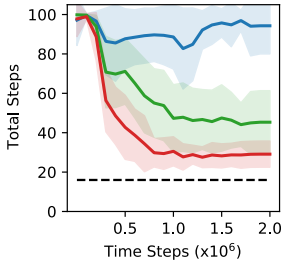
Learned partial models improve generalization



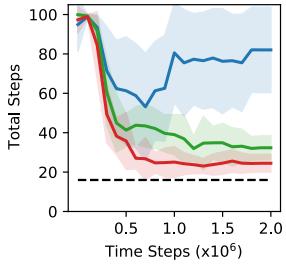
(c) 16x16 BlueBalls-R



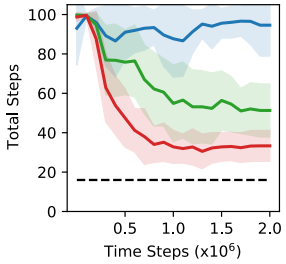
(d) 16x16 NoObstacles-R



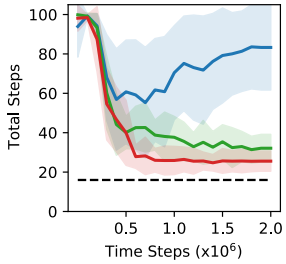
(i) 16x16 RedBalls-R



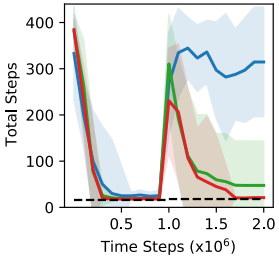
(j) 16x16 GreyBalls-R



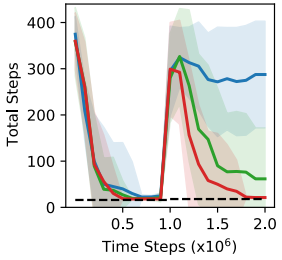
(k) 16x16 RedBoxes-R



(l) 16x16 GreyBoxes-R



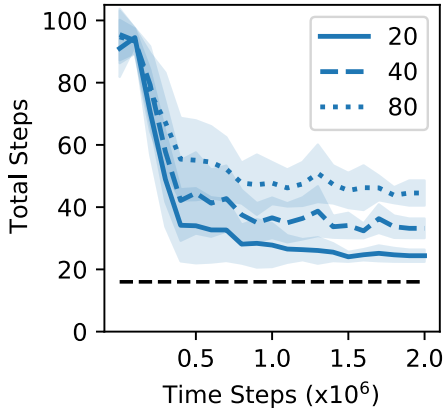
(e) 16x16 RedBalls-K



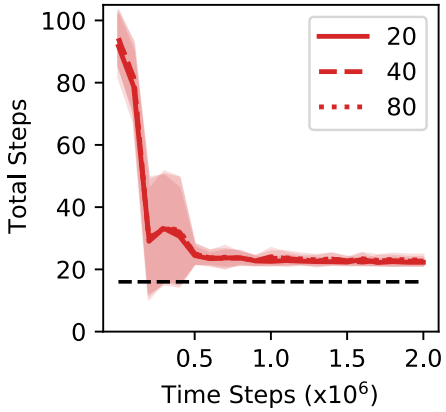
(f) 16x16 GreyBalls-K

Blue: Regular, Green: Value-Equivalent, Red: Value equivalent + models

Partial models allow deeper planning



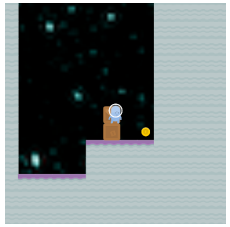
(g) The A_{REG} agent



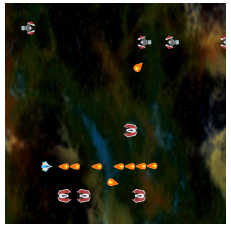
(h) The A_{VES+ME} agent

- Regular models (left) lead to worse performance when doing more planning steps, due to error propagation
- Partial models have better error propagation properties (see Alver & Precup. 2023, for details on the theory)

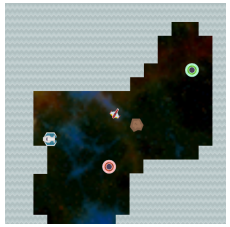
Scaling up: ProcGen



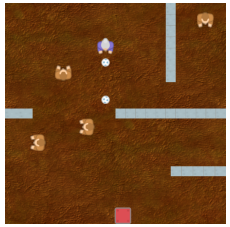
(a) CoinRun



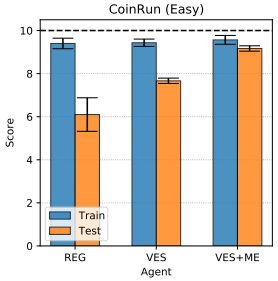
(b) StarPilot



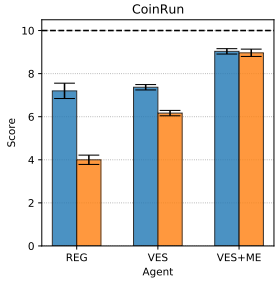
(c) CaveFlyer



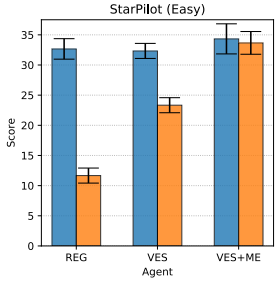
(d) DodgeBall



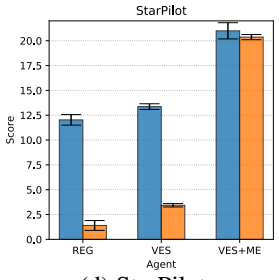
(a) CoinRun (Easy)



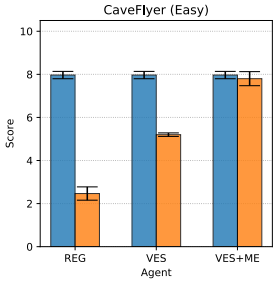
(b) CoinRun



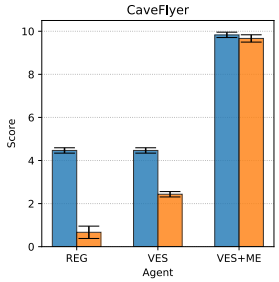
(c) StarPilot (Easy)



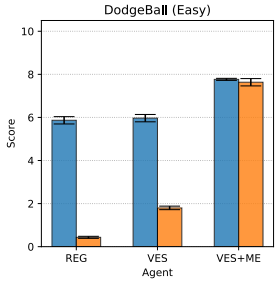
(d) StarPilot



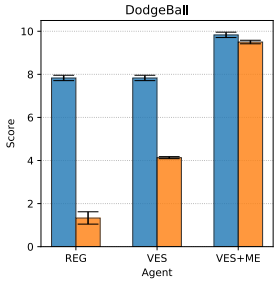
(e) CaveFlyer (Easy)



(f) CaveFlyer



(g) DodgeBall (Easy)



(h) DodgeBall

Partial models improve generalization!

Conclusion

- An agent that is much smaller than its environment will be pressured to find structure on its current trajectory: continually, online, not striving for optimality but for gradual improvement.
- The structure it builds drives two important computations: exploration decisions and credit assignment
- While agent implementations often link these two computations, they can and perhaps should be more decoupled
- Many of the ingredients needed already exist (information-directed sampling, GVFs, options, affordances, partial models)

Some challenges

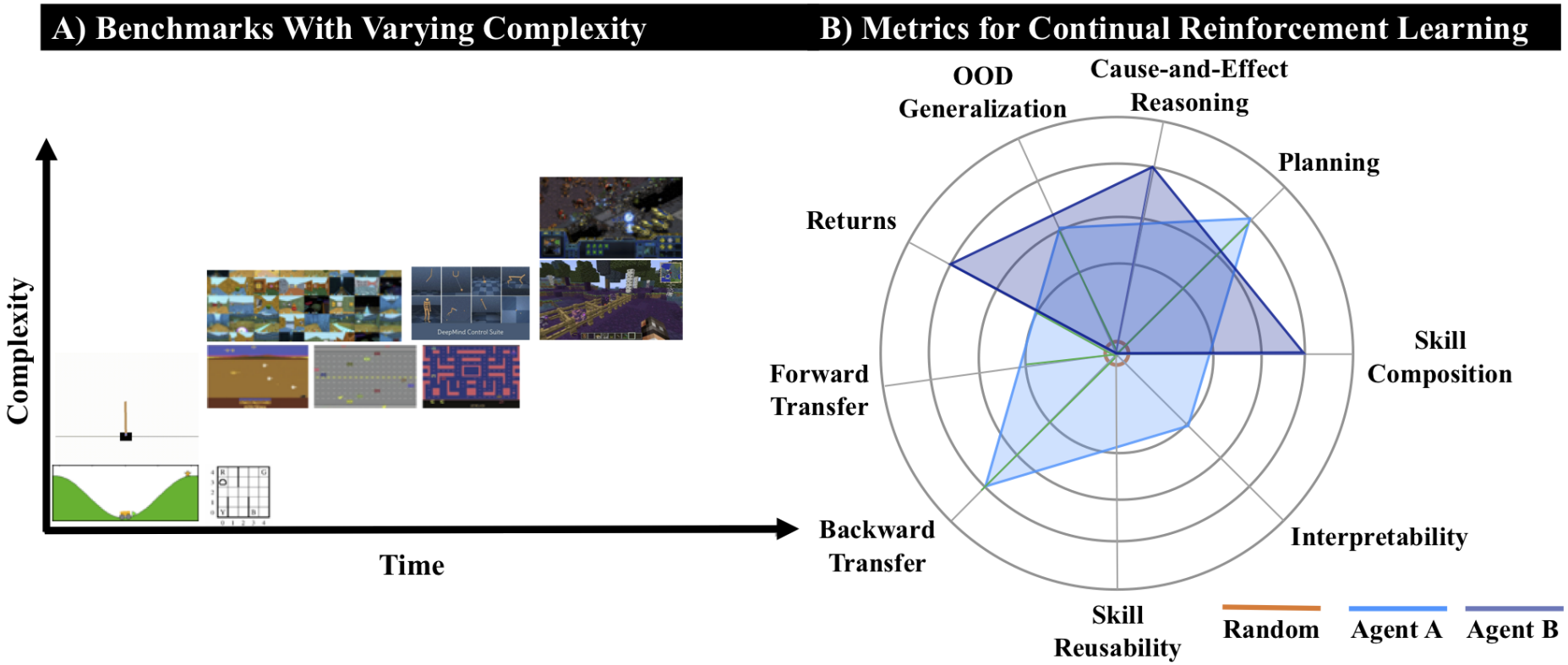
- From a theoretical point of view, we need to formalize the problem further

  Moving away from usual stationarity/recurrence assumptions to fully transient agents

- From an empirical point of view, we should think of the appropriate environments and metrics

  Reconsider reward sparsity as a mark of interesting problems?

Evaluation for continual RL



Cf. Khetarpal, Riemer, Rish and Precup, 2022