LPI: Learned Positional Invariances for Transfer of Task Structure and Zero-shot Planning

Tamas Madarasz [1]

Abstract

Real-world tasks often include interactions with the environment where our actions can drastically change the available or desirable long-term outcomes. One formulation of this in the reinforcement learning setting is in terms of non-Markovian rewards. Here the reward function, and thus the available rewards, are themselves history-dependent, and dynamically change given the agent-environment interactions. An important challenge for navigating such environments is to be able to capture the structure of this dynamic reward function, in a way that is interpretable and allows for optimal planning. This structure, in conjunction with the particular task setting at hand, then determines the optimal order in which actions should be executed, or subtasks completed. Planning methods face the challenge of combinatorial explosion if all such orderings need to be evaluated, however, learning invariances inherent in the task structure can alleviate this pressure. Here we propose a solution to this problem by allowing the planning method to recognise task segments where temporal ordering is irrelevant for predicting reward outcomes downstream. To facilitate this, our agent simultaneously learns to segment a task and predict the changing reward function resulting from its actions, while also learning about the permutation invariances in the its history that are relevant for this prediction. This dual approach can allow zero-shot or few-shot generalisation for complex, dynamic reinforcement learning tasks.

1. Introduction

When interacting with our environment in the real world, our actions often influence this environment in ways that markedly change the nature of possible future interactions. One way to conceptualize this is in terms of a non-Markovian reinforcement learning problem, where the environment's response depends on the history of agent-environment interactions. In a Non-Markovian Reward Decision Processes (Bacchus et al., 1996; Li et al., 2016; Littman et al., 2017; Camacho et al., 2017; Brafman et al., 2018; Icarte et al., 2018; Toro Icarte et al., 2019; Gaon and Brafman, 2019) for example, rewards can depend on the entire history of the agent. While this allows for the formulation of a rich variety of tasks, this setup is clearly problematic without the imposition of additional structure. As we face a combinatorial explosion of histories (and thus, when translating to a Markovian formulation, of states), generalizing to new situations from such a representation is likely to be very difficult, due to of the large number of often closely related, but unique possible experiences.

Here we explore explicitly representing additional structure in terms of invariances (LeCun and Bengio, 1998; Cohen and Welling, 2016; Mondal et al., 2020; van der Pol et al., 2021; Wang et al., 2022) in a changing reward-function. Specifically, we will consider situations when a task is composed of a sequence of subtasks that can be completed, and become available, in different order. In this non-Markovian setting rewards appear and disappear as a consequence of the agent's collection of other rewards, and the reward function is thus time-varying and autoregressive. Unlike previous work, we don't provide the agent with a sketch of subtasks to complete (Andreas et al., 2017), or policies for reaching subgoals (Sohn et al., 2020), instead we aim to flexibly learn a latent representation of different phases, or 'task states' of the overall task. This latent representation is then be combined with learning invariances in the subtask structure in a planning module that performs 'jumpy' planning over the subtasks, jumping through and optimally reordering invariant sequences of the planning process under different settings. Using these elements we hope to present an algorithmic framework that allows reasoning over an agent's interactions with a dynamic environment and the resulting

[1]MediaTek Research. Correspondence to: Tamas Madarasz <tamas.madarasz@mtkresearch.com>.

Responsible Decision Making in Dynamic Environments Workshop, 39^{th} International Conference on Machine Learning. ,

feedback loops.

2. Motivation

A core component of our approach is to augment the state space of a task specified by a non-Markov reward function using an explicit task state variable. We would like to segment the task into parts such that future rewards are independent of the history during *previous* task states, when conditioning on the *current* task state. We make the assumption that the true reward function $R_t^{T_t}(s')$, is always expressible as a function of the task identity T_t and the complete state and reward history during the episode $\mathcal{H} = (s_0, s_1, r_1, ...st, r_t)$, where we leave out the history of actions for convenience and condition rewards on the arrival state, though these assumptions are not central to our approach.

2.1. Generalizable segmentation using episodic and short-term memories

Given this assumption, we desire the decomposition of the task into task segments that are *generalizable*, in the sense that they can be transported over to new settings when e.g. the assignment of rewards to states change. This means that we want to abstract away the task structure from low-level features such as observations and states when possible. While we abstract away the task structure into the learned tasks state representations, we will rely on a memory of the current episode (short-term) and of previous episodes (episodic), to store the other relevant details of the agent's experience, similarly to some previous memory-based approaches to generalization (Santoro et al., 2016; Fortunato et al., 2019). To make predictions about currently available rewards, the agent should only be allowed to use the history of its current task state, and histories in identical task states in past episodes. Instead of this idealized and discrete setup, we encode task states as continuous latent variables, and relax this restriction by using a similarity-based key-value attention mechanism. This setup allows the retrieval of information from the external memory buffer while enabling flexible learning through gradient decent, as well as potentially the learning of higher order correlations between tasks states.

2.2. Maximal invariant task states

Further, we would like to use these task states to identify the *maximal* permutable (order-independent) sequences in a task, allowing for the most flexibility and computational savings when planning. We will incorporate this desire into our loss function when designing the algorithmic mechanism for learning about invariances by penalising the use of positional information. This means encouraging, whenever possible, the masking out of positional encodings, and establishing order-invariance.

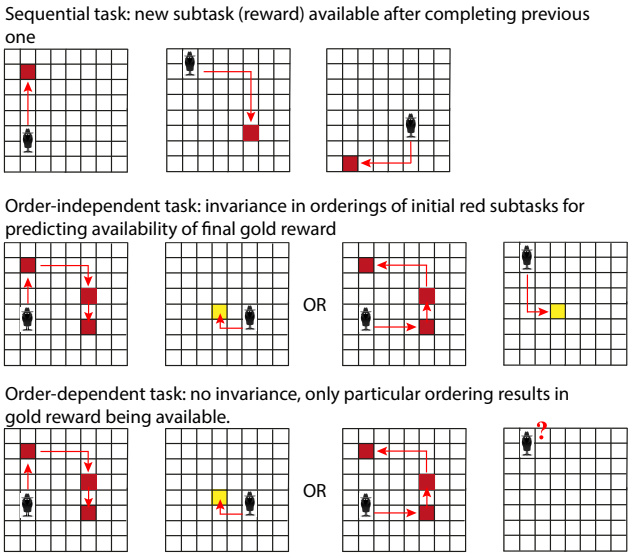

Figure 1: Examples of non-Markovian task structures, where subtasks are represented by rewards to be collected. In these examples, rewards can be collected once and new available subtasks might appear. The second row shows permutation invariance to the ordering of the red subtasks in the first phase of the task.

3. Problem setup

3.1. Order-dependent and order-independent task and subtasks

We will focus our attention on tasks where subtasks are realized as different (identifiable) rewards that can be collected during different task states, such that downstream consequences might, or might not depend on the specific ordering of collection. We illustrate the basic idea of a sequential task, as well as of an order-dependent, and order-independent task state in Fig. 1. A composite task can then be made up of a sequence of such order-dependent or independent task states.

4. Model components

Our model consists of four main components.

- An autoregressive sequence model outputting a task state variable. This Teacher network is then used in conjunction with an external memory buffer for retrieval, and a reward prediction network to predict available rewards for any state.
- A Student network that learns invariances in orderings of episodes by trying to predict the outputs of the

ordering-aware sequence model above.

- A tree search that identifies the next desirable subgoal, using the external memory, the learned invariances, and a distance metric over the state space.
- A navigation module that helps the agent navigate to the desired subgoal.

Below we give more detail for each of these.

4.1. Reward-predictive retrieval Transformer

We implement our reward-predictive model as a memory-augmented Transformer (Vaswani et al., 2017) with an encoder-decoder architecture (Fig. S1), that outputs the latent task state c_{t+1} for the next step, given a history of rewards $r_0, \cdots r_t$. This prediction happens autoregressively as the agent is acting in the environment, and in parallel for all time steps during learning using offline updates. During such updates a single pass through the transformer generates the latent task states for a complete episode, using triangular causal masks to ensure that the inference of the latent task state only uses past reward information.

We use the encoder-decoder setup to allow for a separation of two components of representation learning. The encoder first summarizes the reward history into a first estimate of the latent task state. The decoder then refines this using comparison to the previously stored task state representations in the memory buffer M, which are fed as targets to the decoder. Because the task state representations $\mathbf{c}$ proposed by the Transformer will be compared to those in the memory buffer during retrieval and reward prediction, these targets are important in ensuring that the proposed representations $\mathbf{c}$ are grounded in the memory buffer already at the point of generation. This helps faster convergence to a progressively better segmentation of the task. We do not use positional encodings for this network, as part of the challenge in this subtask-based RL setting is that the specific temporal location (both absolute and relative) of rewards changes form episode to episode, and task to task. The multi-layer architecture, together with the masking, ensures that the network has enough information about orderings, without positional encodings. It can thus capture the dependencies of task states on temporal orderings of rewards, while helping faster learning of this structure using episodes with different temporal profiles.

To predict rewards in arrival states, the task state representation is used in conjunction with the memory buffer M, to construct reward maps for the environment. To predict rewards, the model uses the current latent task state representation c and the embedding of a proposed arrival state $\phi(s)$. These act as keys for reading from a memory buffer of past transitions in the current episode M^t, and a fixed size sample of past episodes $\bar{M}$. The sampling of past episodes is prioritized to include episodes with low prediction errors on previous updates, as well as ones with diverse reward-orderings. When the memory is read, several sufficient statistics are retrieved for reward prediction, using permutation invariant retrieval mechanisms (such as sum, mean, or max over the per-transition retrieved values from the memories). These statistics in general could be predefined or learned, in this work we used three such prespecified sufficient statistics. These condition predictions by focusing on part of the agent's history that shared the same task state and same arrival state, in a soft way. We give the details for computing these statistics in Appendix A.

4.2. Student Transformer for learning positional invariances

To learn the permutation invariances in the reward function we deploy a smaller Transformer, that only has access to ordering information through positional encodings. To ensure this strict dependence when processing an episode as a sequence, we still require the use of a causal mask, as later rewards might carry information about the order in which earlier ones were collected, thus relaying ordering information without relying on positional encodings. The use of causal masks means that we also cannot feed back the output of an encoder layer for further use with self-attention, as this would break the permutation invariance of the attention mechanism with respect to the input sequence, thus again encoding order information. Therefore, to restrict ordering information to only come from the positional encodings, this network only consists of an Encoder with a single Encoder layer. Positional encodings are added to keys and values (but not to queries), but they can be masked out by a gating variable g_t, taking values in $\{0, 1\}$, thus explicitly encoding invariances. The self-attention then becomes:

$$softmax\Big(\frac{Q(K + P \odot G)^T}{\sqrt{d}}\Big)(V + P \odot G)$$

where the masking is implemented by an elementwise product with the matrix G, with $G_{i,t} = g_t$

We train the Student Transformer to predict the task state embeddings $\hat{c}_{teacher}$ produced by the Teacher Transformer in order to find the best assignment of positional masking to task states, and thus infer the relevant the permutation invariances. We assume that this invariance can be learnt as a function of the task state representation produced by the Teacher Transformer, and therefore treat g as a function of $\hat{c}_{teacher}$. The Student Transformer takes as input the sequence of rewards as before, and we train the parameters of the Transformer and evaluate on a holdout part of the memory for each masking assignment, resulting in a dual optimization problem with a discrete outer loop. We further discuss the training approach in section B of the Appendix.

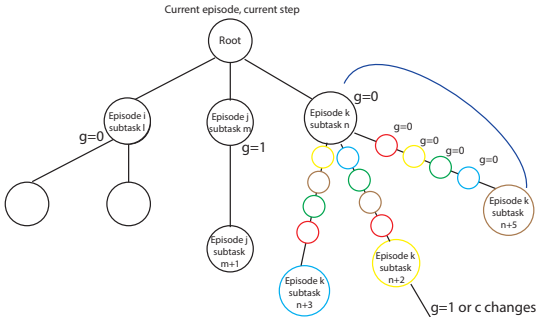

Figure 2: The tree search evaluates episodes from the memory buffer that share a position in task state space similar to the agent's current one. The search proceeds by 'translating' the episode to the current task setting (assignment of subtasks to states), and by reordering the subtasks in permutation invariant task states to find an ordering that promises the best return.

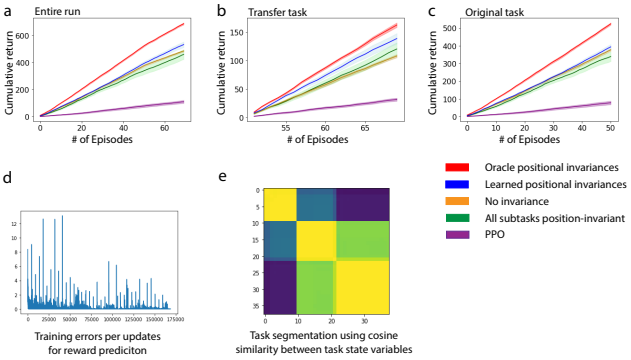

Figure 3: a, Learned positional invariances lead to better performance overall (n=10, mean± sem). b, This advantage is even more pronounced when transferring to a new task, where using the right invariances leads to better zero-shot performance. c, Learning the correct invariances also leads to better performance on the original task. d,Training errors during an example run of the reward prediction model. e, cosine similarities between the task state variables at every step of an example episode (brighter is higher).

4.3. Tree search on the memory buffer using positional invariances

The search procedure (Fig. 2) also uses a subset of episodes $\bar{M}$ from the memory buffer, allowing the agent to rearrange these previous experiences into potentially more optimal ones. The first step is to find episodes with similar histories as the current one, in order to plan ways to continue executing the current task in the best possible way. The agent tries to locate, in each past episode e_i in the sampled memory, the closest position in 'task state space' to its current position in the task in the current episode. To do this, it looks for a similar task state, and, in case of an order-dependent task state, a similar reward history within the task state, and computes match likelihoods based on these similarities.

The algorithm then samples, according to these likelihoods, a fixed number of episode/position pairs that are most likely to correspond to the current position in the current task (the same episode might be sampled with different positions, however duplicate episode/position pairs are not used). The tree search proceeds by evaluating the returns from the remaining parts of these episodes in two ways: by assigning subtasks to their locations under the current task setting, and by permuting segments in the episode that are believed to be order-invariant as indicated by the values of g.

A sampling based search procedure is necessary as we still need to evaluate different permutations of the smaller, order-independent sub-sequences. A choice of ordering can affect which subsequent trajectories are optimal in a particular task instantiation, however, this effect will only depend on the **final** subgoal of each such sub-sequence, while the intermediate subgoals can be locally optimally reordered without needing separate evaluations by a downstream search. This results in having to search through a *linear*, rather than *exponential* space of orderings. The tree search proceeds along promising branches using a procedure inspired by Monte Carlo Tree search (Kocsis and Szepesvári, 2006), with a rollout policy to leave remaining subgoals of the episode under consideration unpermuted, i.e. in the order in which they were originally experienced and recorded in the memory.

For navigating to a chosen subgoal, we used a simple setup as described in section C.

5. Experiments

We focus on a proof of concept evaluation, and test our model on a non-Markovian grid-world navigation task (Fig. S3) that combines elements from the different modalities shown in Fig. 1. We give details of the task in section D of the Appendix.

We evaluate our algorithm LPI-planning from several different perspectives. On Fig. 3a, 3b, and 3c we show the cumulative returns on the 'original' and 'transfer tasks', and compare to ablations where the positional gating is set to either always 0, or always 1, as well as to the 'oracle' value . The discount factor was always set to $gamma = 0.9$, to appropriately penalize longer trajectories. Fig. 3d shows that the predictive model learns to correctly predict upcoming rewards, while Fig. 3e illustrates an example of how the task state variables **c** segment an episode.

6. Discussion

We introduced a representational framework to learn the subtask structure of complex, non-Markovian tasks in terms of permutation invariances in their reward functions. This framework can be useful for understanding the interactions between an agent and a dynamic environment that constantly changes as a consequence of the agent's actions, and for allowing robust predictions about the behaviour of similar environments in future tasks. Important directions for future work are to demonstrate this kind of structure in real world datasets, where transfer and planning represent important challenges, as well as to relax some of the more restrictive assumptions of the current formulation.

References

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI Press, 1996. ISBN 026251091X.

Ronen Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Ltlf/ldlf non-markovian rewards. In *AAAI Conference on Artificial Intelligence*, 2018.

Alberto Camacho, Oscar Chen, Scott Sanner, and Sheila A. McIlraith. Non-markovian rewards expressed in ltl: Guiding search via reward shaping. In *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA*, 2017.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/cohenc16.html.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Comput.*, 1993. ISSN 0899-7667.

Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5c48ff18e0a47baaf81d8b8ea51eec92-Paper.pdf.

Meire Fortunato, Melissa Tan, Ryan Faulkner, Steven Hansen, Adrià Puigdomènech Badia, Gavin Buttimore, Charles Deck, Joel Z Leibo, and Charles Blundell. Generalization of reinforcement learners with working and episodic memory. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/02ed812220b0705fabb868ddbf17ea20-Paper.pdf.

Maor Gaon and Ronen Brafman. Reinforcement learning with non-markovian rewards. 12 2019.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2107–2116. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/icarte18a.html.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, page 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 354045375X. doi: 10.1007/11871842_29. URL https://doi.org/10.1007/11871842_29.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Yann LeCun and Yoshua Bengio. *Convolutional Networks for Images, Speech, and Time Series*, page 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262511029.

Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards, 2016.

Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via gltl. *arXiv preprint 1704.04341*, 2017.

Arnab Kumar Mondal, Pratheeksha Nair, and Kaleem Siddiqi. Group equivariant deep reinforcement learning, 2020.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, New York, New York, USA, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Sungryull Sohn, Hyunjae Woo, Jongwook Choi, and Honglak Lee. Meta reinforcement learning with autonomous inference of subtask dependencies. *CoRR*, abs/2001.00248, 2020. URL http://arxiv.org/abs/2001.00248.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/532435c44bec236b471a47a88d63513d-Paper.pdf.

Elise van der Pol, Daniel E. Worrall, Herke van Hoof, Frans A. Oliehoek, and Max Welling. Mdp homomorphic networks: Group symmetries in reinforcement learning, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Dian Wang, Robin Walters, and Robert Platt. SO(2)-equivariant reinforcement learning, 2022. URL https://arxiv.org/abs/2203.04439.

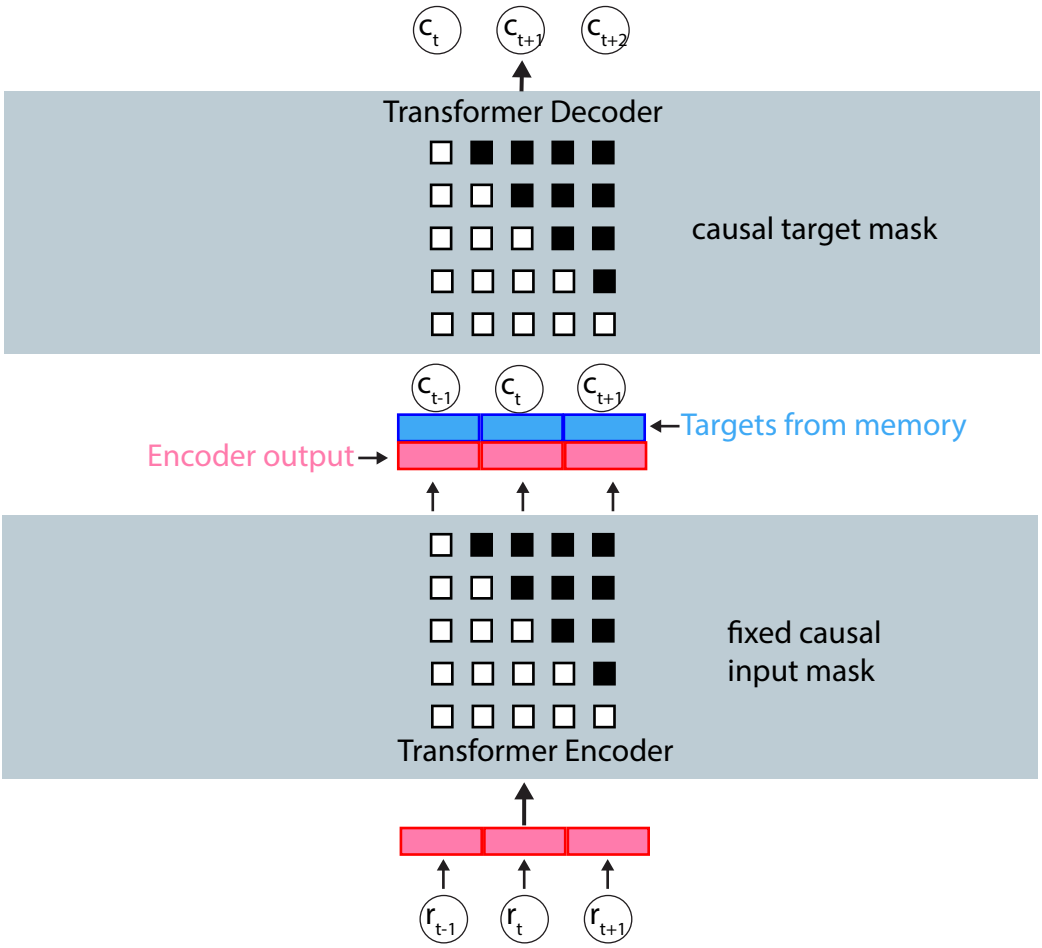

Figure S1: The Transfomer model takes as input tokenized rewards from an episode in the memory buffer, and autoregressively outputs the task state variable for the next step. As targets for the decoder layer, it uses the latent task states recorder for this episode during the last update.

A. Reward prediction using retrieved sufficient statistics

For reward prediction, we retrieve three sufficient statistics from $\bar{M}$, using c and a proposed arrival state as keys. The first statistic is the total reward collected from each episode e in $\bar{M}$ in the task state and arrival state specified by the keys c_t and $\phi(s_{t+1})$

$$\sum_k sim(c_t, c_{e,k}) \cdot sim(s_t, s_{e,k}) \cdot r_{e,k}$$

where sim denotes cosine similarity and k indexes the steps a in previous episode e. The second statistic is the maximum reward retrieved with the keys across these previous episodes

$$\max_k \left[sim(c_t, c_{e,k}) \cdot sim(s_t, s_{e,k}) \cdot r_{e,k} \right]$$

And finally the third is the sum of rewards already collected during the current episode e_t

$$\sum_k sim(c_t, c_{e_t,k}) \cdot sim(s_t, s_{e_t,k}) \cdot r_{e,k}.$$

This gives us the flexibility to predict locally non-Markovian rewards, e.g. rewards that can be 'picked up' only once, or that deplete according to some well-defined rule, or indeed are Markovian. To predict the expected reward on arriving in state s_{t+1}, these statistics are fed into a reward prediction convolutional network that predicts the value of the reward.

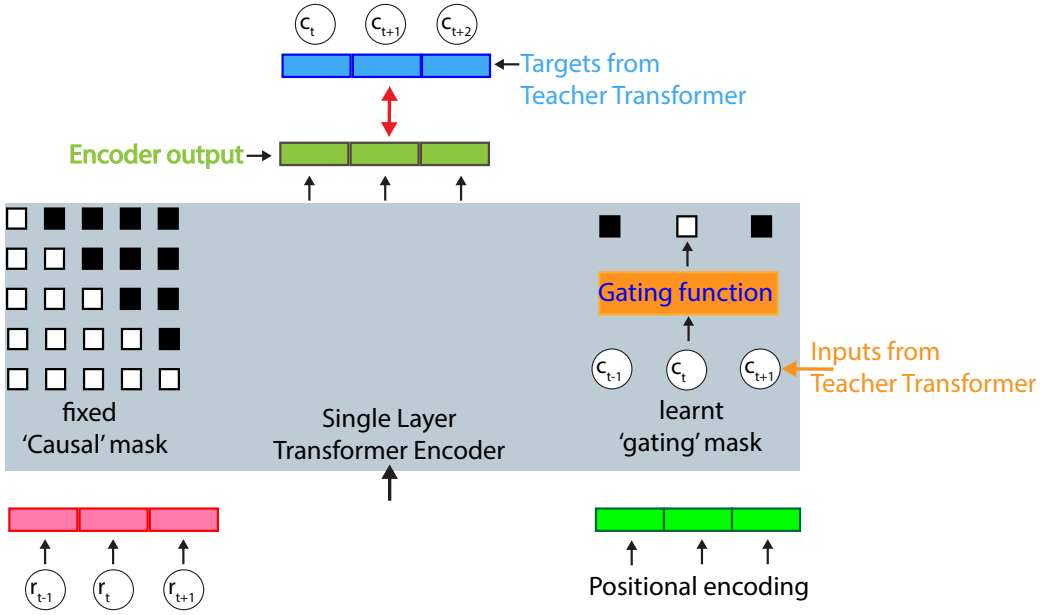

Figure S2: The Student Transformer evaluates different assignments of the gating variable that gates positional encodings of the input reward sequence. The gating function assigns a gating value of 0 (no positional encoding) or 1 (use positional encoding) to prototypes of the task state variables, and the Student network tries to predict the task state variables output by the Teacher Transformer network of Fig. S1. The Student Transformer only has access to ordering information through positional encodings, leading it to perform better when it correctly assigns gating values to order-dependent and order-independent task states.

When input into the Transformer, rewards were tokenized into a vector $\rho(r)$, using a fixed tokenization depending on the reward's numerical value. Reward prediction estimates this numerical value, which is also the one used during planning to evaluate returns. On the other hand, the weights for the network producing the state embeddings ϕs were learnt together with those of the Transformer and reward network.

B. Training procedures

B.1. Reward-predictive Teacher Transformer and CNN

The reward-predictive Transformer and CNN were trained together using offline replay of previous episodes, concurrently with the agent taking each step in the environment. For every update, a batch of 16 episodes was passed through the Transformer once, using the sequence of tokenized rewards and task state embeddings recorded in $\bar{M}$. The rewards for each step in each replayed episode were then predicted using the retrieved sufficient statistics and the reward CNN, and the mean squared error loss backpropagated through both networks, as well as the state encoder network ϕ. We used the RMSprop optimizer, with an annealed learning rate.

B.2. Student Transformer

The Student Transformer was trained every 10 episodes after an initial burn-in period in which all the positional gatings were set to 1. During this training, we first permformed k-means clustering on the tasks state variables $\mathbf{c}$ recorded in M, and then consider all the binary assignments of the gating variable to the clusters. For the training, we split the memory buffer into a training and validation part, and train the Student Transformer separately (and from scratch) for each of the binary assignments. The objective function was the mean squared error between the task state variable in the memory buffer (as output by the Teacher Transformer), together with a regularizer term that penalises the masking variable g taking the value 1:

$$\mathbb{E}\Big[(\mathbf{c}^{Teacher} - \mathbf{c}^{Student})^2 + \beta_g \cdot \mathbf{g}\Big].$$

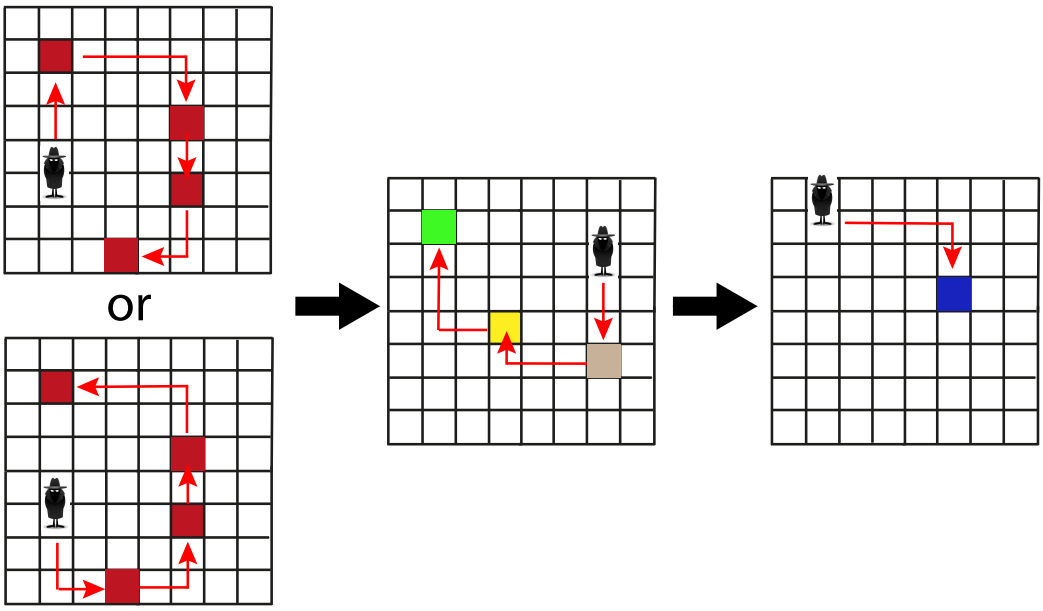

Figure S3: Non-Markovian task with an order-independent task state with four subtasks, followed by and order-dependent task state with three, followed by a final subtask available upon a particular ordering. The location of some of the relevant rewards were shared between the different task states.

C. Distance metrics and navigation to subgoals

To evaluate a proposed novel ordering of subtasks, the agent requires some distance metric on the state space between pairs of states and for lengths of (open) tours visiting a sequence of states. Previous work has used value functions (Eysenbach et al., 2019), the successor representation (Dayan, 1993) or deep learning approach to combinatorial optimization (Kool et al., 2019) to try to learn and extrapolate such a metric. To help focus on the other parts of our approach, here we used exact the exact lengths of shortest paths to compute discounted returns when planning, and leave the incorporation of similar learned approximations for future work.

Similarly, to navigate to a desired subgoal, we simply used the successor representation with a temporary, shaped reward function that incentivized the agent to reach that subgoal and pick up the environmental reward. Depending on the exact problem setting, a more involved goal-conditioned policy could be used to successfully guide the agent between subtasks.

D. Experiment details

For evaluation, we use a grid-world navigation task that combines elements from the different modalities shown in Fig. 1. For added complexity, we also assign several subtasks in different phases of the task assigned to the same state. In particular our task (Fig. S3) is composed of three distinct phases: in the first, the agent has four rewards to collect (subtasks to complete), each of which is available from the beginning of the episode, and can be collected once before it disappears. Once all four has been collected, three new appear. The agent can again collect these rewards in any order, however only one of these orderings will result in the appearance of the final reward. Whether accessing the final reward is worthwhile depends also on the particular setting and the agent's starting position.

We also focus mainly on the issue of exploitation, and give initial demonstrations to the agent by entering a limited number of trajectories (covering a limited number of orderings) into the memory buffer. After this, the agent does no explicit exploration, except through sampling when performing the tree search.

We run each agent ten times, initially for 50 episodes, at the beginning of each of which the agent is spawned at a random location in the maze, resulting in a potentially new optimal ordering of subtasks each time. After 50 episodes the task setting changes, and subtasks are assigned to new states, though the task structure remains the same. We provide this information of the new assignment to the agent, so it can appropriately re-evaluate its past experiences. Learning is disabled at this point

for the LPI-planning algorithm, but PPO (Schulman et al., 2017) is allowed to learn, or start from tabula rasa, whichever gave better results.

E. LPI Algorithm

Algorithm 1 LPI

Require: Memory buffer M , number of clusters k, hyperparameters $\{\}$
Initialise M with some preloaded trajectories
for each episode **do**
  $t \leftarrow 0$, random initial state $s \leftarrow s_0$
  while not terminal and steps taken in episode<limit **do**
    Compute task state c_t using history of rewards and M
    if $t >$burn-in **then**
      Compute gating mask g_t as $g(argmin_i||c_t - c_i^{prot}||_2)$
    else
      $g = 1$
    end if
    Subsample M into $\bar{M}$
    if received reward in last step, or steps since last planning$> n_p$ **then**
      subgoal $s_g =$ SUBTASK TREE SEARCH ($\bar{M}$)

    end if
    Take step towards s_g
    Execute a and obtain next state s' and reward $r = r(s')$
    Store $(c, g, s, s', \phi(s), \phi(s'), r)$ in M
    UPDATE TRANSFORMER($\bar{M}$)
    if every n-th episode **then**
      $(c_i^{prot})_{i=1\cdots k} =$ K-MEANS CLUSTER(M_c, k)
      Split M into training and validation sets
      for every assignment of 0 or 1 to each cluster prototype **do**
        validation errors =TRAIN STUDENT TRANSFORMER (M)
        choose winning assignment $g(c_i^{prot})$
        Every n_2 episodes, change task setting

      end for