CONTRASTIVE INTROSPECTION (CONSPEC) TO RAPIDLY IDENTIFY INVARIANT STEPS FOR SUCCESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) algorithms have achieved notable success in recent years, but still struggle with fundamental issues in long-term credit assignment. It remains difficult to learn in situations where success is contingent upon multiple critical steps that are distant in time from each other and from a sparse reward; as is often the case in real life. Moreover, how RL algorithms assign credit in these difficult situations is typically not coded in a way that can rapidly generalize to new situations. Here, we present an approach using offline contrastive learning, which we call contrastive introspection (ConSpec), that can be added to any existing RL algorithm and addresses both issues. In ConSpec, a contrastive loss is used during offline replay to identify invariances among successful episodes. This takes advantage of the fact that it is easier to retrospectively identify the small set of steps that success is contingent upon than it is to prospectively predict reward at every step taken in the environment. ConSpec stores this knowledge in a collection of prototypes summarizing the intermediate states required for success. During training, arrival at any state that matches these prototypes generates an intrinsic reward that is added to any external rewards. As well, the reward shaping provided by ConSpec can be made to preserve the optimal policy of the underlying RL agent. The prototypes in ConSpec provide two key benefits for credit assignment: (1) They enable rapid identification of all the critical states. (2) They do so in a readily interpretable manner, enabling out of distribution generalization when sensory features are altered. In summary, ConSpec is a modular system that can be added to any existing RL algorithm to improve its long-term credit assignment.

1 INTRODUCTION

In real life, rewards are sparse and there are multiple steps required for success. For example, consider the steps necessary for getting a paper accepted at ICLR. One must generate an idea, conduct mathematical analyses or experiments to test the idea, create figures, write a paper, submit the paper, and finally, respond to reviewer comments in a satisfactory manner. Skipping any of these steps will lead to failure. Moreover, these steps are not themselves rewarding and they will be separated from each other and any reward by long periods of time, during which you will engage with other life tasks.

Though such situations are common in real life, they are highly nontrivial for most RL algorithms. The problem is that the traditional approach for credit assignment in RL, using the Bellman equation (Sutton & Barto; Bellman), will take an unworkable length of time to propagate value estimates back to the earlier important steps if the length of time between critical steps and any reward is sufficiently long (Raposo et al., 2021; Puigdomènech Badia et al., 2020).

Some recent work has attempted to address this problem using memory. In this approach, the system attempts to predict value or reward by using not just the current state and the Bellman equation, but also past states in the memory buffer selected by attention or other means (Arjona-Medina et al., 2018; Hung et al., 2019; Raposo et al., 2021; Chen et al., 2021; Parisotto et al., 2019; Goyal et al., 2019; 2020). The challenge with these approaches is that the agent must predict rewards at every state it encounters, which is a daunting modelling problem when there are multiple key states to go through in order to ensure success.

Here, we present an alternative approach to long-term credit assignment that can be added to any existing RL agent. Our approach, which we call contrastive introspection (ConSpec), uses offline learning from a memory buffer with a contrastive loss that identifies invariances amongst successful episodes. More precisely, ConSpec learns a state encoder and a series of prototypes that represent key intermediate states. It then delivers intrinsic rewards (Schmidhuber, 2010; Randløv & Alstrøm, 1998) to the RL agent when it arrives at states that match any of its prototypes. We show that the addition of ConSpec to an agent allows the agent to achieve success even when there are multiple steps required that are separated by long periods of time, a feat that most current RL approaches would otherwise be incapable of. We demonstrate that the reward shaping of ConSpec can be done so as to not alter the optimal policy of the agent, and in-line with this, the addition of ConSpec to an agent does not impair performance on standard RL tasks that don't require long-term credit assignment, such as Atari games (Brockman et al., 2016; Kostrikov, 2018). Furthermore, the contrastive loss and its ability to identify invariances allows ConSpec to scale up to complex 3-dimensional environments. We show that when the sensory features of critical states are altered, ConSpec's prototypes can exhibit zero-shot generalization. Thus, ConSpec is a promising approach to improving RL in a wide variety of applications and overcoming the challenge of learning when rewards are sparse and success is contingent upon multiple key steps separated by long periods of time. It moreover has the added perk that the knowledge learned is readily interpretable thanks to the use of discrete prototypes to represent key steps.

2 CONTRASTIVE INTROSPECTION FOR IDENTIFYING INVARIANCES IN KEY STEPS FOR SUCCESS

In real life, humans are very good at recognizing when their success is contingent upon multiple past events. Moreover, they are very fast at recognizing these contingencies—even children can identify contingencies in a task after only a handful of experiences (Sobel et al., 2004; Gopnik & Sobel, 2000). Intuitively, humans achieve this by engaging in introspection to examine their past and determine which steps along the way were critical for success. Such introspection often seems to involve a comparison: we seem to be good at identifying the differences between the situations that led to success and those that led to failure, and then we hone in on these differences. We reasoned that RL could be made to deal with multiple contingencies—and moreover, do it in a robust way—if it were equipped with a similar capability to take advantage of memory to sort through the key events differing between successful versus failed episodes. This paper's principal contribution is the introduction of an add-on architecture and loss for introspection of this sort: contrastive introspection (ConSpec) (Fig. 1).

At its core, ConSpec is a mechanism by which a contrastive loss can enable rapid learning of contingencies in an episodic task. It needs the following elements, which we will expand on:

- Prototypes of critical states for success that are invariant across trajectories.
- A memory system that stores successful and failed episodes.
- A contrastive loss to learn the prototypes.
- A mechanism for keeping prototypes stable once they are learned.
- A mechanism by which learned prototypes can help train a downstream RL agent.

2.1 A SET OF INVARIANT PROTOTYPES FOR SUCCESS

To recognize when a new observation corresponds to a state that success is contingent upon, we store a set of H prototypes of candidate critical states. Each prototype, h_i is a learned vector that is compared to a non-linear projection of the latent representation of currently encoded features in the environment, $g_\theta(z_t)$ (with projection parameters θ). Thus, if we have current observation O_t, encoded as a latent state $z_t = f_W(O_t)$ (with encoder parameters W), we compare $g_\theta(z_t)$ to each of the prototypes, h_i, in order to assess whether the current observation corresponds to a critical state for success.

At first glance, one might think that the number of prototypes would have to be very large, since there are a massive number of potential feature vectors. However, this is not actually the case. ConSpec is designed to discover prototypes that capture invariant features across successful episodes (see next section on the contrastive loss function for achieving this invariance). Therefore, there is a natural

grouping of observations relative to prototypes, no matter how many prototypes one uses. As such, using $H \leq 12$ was enough for all our tasks, even the ones in 3D environments (see sections 3.3-3.4) which have a lot of pixel-level variation. Of course, in real-world settings more prototypes would likely need to be used.

2.2 A MEMORY SYSTEM FOR STORING SUCCESSES AND FAILURES

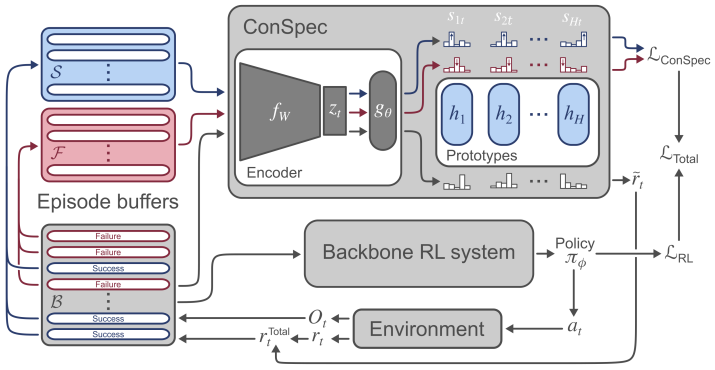

Figure 1: ConSpec is a module that can help any backbone RL system. ConSpec trains its prototypes by comparing successful v. failed episodes via a contrastive loss, and uses the match to these prototypes to output intrinsic rewards to the RL agent.

ConSpec maintains three memory buffers: one for the current mini-batch being trained on ($\mathcal{B}$; with a capacity of $M_{\mathcal{B}}$), one for successful episodes ($\mathcal{S}$; with a capacity of $M_{\mathcal{S}}$), and one for failure episodes ($\mathcal{F}$; with a capacity of $M_{\mathcal{F}}$). Each episode is of length T time steps. As we show, these memory buffers need not be very large for ConSpec to work; we find that 16 slots in memory is enough, even for 3D environments.

Each buffer stores raw observations, O_t, encountered by the agent in the environment. When a new mini-batch of observations is loaded into $\mathcal{B}$ for training, the episodes are categorized into successes and failures using the criteria described in A.1. We then fill $\mathcal{S}$ and $\mathcal{F}$ with observations from those categorized episodes in a first-in-first-out (FIFO) manner, where episodes stored from previous batches are overwritten as new trajectories come in.

2.3 A CONTRASTIVE LOSS TO LEARN PROTOTYPES

To learn our invariant prototypes and encodings we employ a contrastive loss that differentiates successes from failures. The contrastive loss uses the observations stored in $\mathcal{S}$ and $\mathcal{F}$ to learn prototypes each of which captures an invariant feature shared by a spectrum of states that success is contingent upon. The functional form of this loss is:

$$\mathcal{L}_{\text{ConSpec}} = \sum_{i=1}^{H} \left[\frac{1}{M_{\mathcal{S}}} \sum_{k \in \mathcal{S}} |1 - \max_{t \in \{1 \ldots T\}} s_{ikt}| + \frac{1}{M_{\mathcal{F}}} \sum_{k \in \mathcal{F}} | \max_{t \in \{1 \ldots T\}} s_{ikt}| \right] + \alpha \cdot \frac{1}{H} \sum_{i \neq j} \sum_{k \in \mathcal{S}} |\cos\left(\boldsymbol{s_{ik}}, \boldsymbol{s_{jk}}\right)| \tag{1}$$

where $\cos(\cdot, \cdot)$ is cosine similarity, $s_{ikt} = \cos(h_i, g_\theta(z_{kt}))$ is the similarity of prototype h_i and the latent representation for the observation from timestep t found in episode k, and $\boldsymbol{s_{ik}}$ is the vector of s_{ikt} elements concatenated along t and softmaxed over t to sparsify it.

To elaborate on this loss, for each prototype h_i and each episode's projection at time step t, $g_\theta(z_{kt})$, the cosine similarity is calculated between the prototype's vector and the latent projection vectors to yield a time-series of cosine similarity scores. The maximum similarity score across time within each episode is calculated. The maximum similarity score for each successful episode is pushed up (first term of the loss) and the maximum score for each failed episode is pushed down (second term). The 3rd term of the loss is an orthogonality constraint imposed to encourage the different prototypes to be independent from each other. More specifically, this term encourages different prototypes to attend to different time steps in an episode. The ConSpec loss is added to any RL losses during off-line training (Algorithm 1 pseudocode). Here, distinct prototypes will learn to hone in on distinct critical states that are present in success and absent in failure, as we will soon observe (Fig. 2b).

2.4 KEEPING PROTOTYPES STABLE ONCE THEY ARE LEARNED

There is a potential source of instability in learning prototypes with ConSpec. Specifically, once an agent has learned a prototype for a specific critical state and helped shape the policy so that the agent consistently achieves this state, the failure buffer will subsequently begin to have many examples where this critical state is also present because that key state may not be sufficient to achieve success by itself (if multiple key states are necessary). Thereafter the initial prototype would begin to diverge from this critical state, and this can lead to instability. To prevent this, we froze the set of memories that defined each prototype upon reaching a criterion. We did so by moving towards the use of separate success and failure buffers for each prototype, such that we had H success buffers and H failure buffers. Once prototype i's score differentiates successes from failures above $\mathcal{T}$, its memory buffers, $\mathcal{S}_i$ and $\mathcal{F}_i$, were frozen and no new memories were added to them. But, the prototype still did gradient updates on its existing memories to prevent staleness.

2.5 GENERATING INTRINSIC REWARDS BASED ON INTROSPECTION

Thus far, the described model learns a set of discrete prototypes for critical states and can identify when an observation in an episode matches a prototype. This system for learning and recognizing critical states must still be connected to the RL agent (with policy π_ϕ, parameterized by ϕ). This could proceed in a number of different ways. Here, we chose the reward shaping approach (Schmidhuber, 2010; Randløv & Alstrøm, 1998; Ng et al., 1999), wherein we add additional intrinsic rewards $\tilde{r}_{kt}$ to the rewards from the environment, r_{kt}, though other strategies are possible. One simple approach is to define the intrinsic reward according to the output similarity scores of prototypes for each time step, for instance, according to the equation:

$$\tilde{r}_{kt} = \lambda \sum_{i=1}^{H} \hat{s}_{ikt} \cdot \mathbb{1}_{\hat{s}_{ikt} = \max\{\hat{s}_{ik,t-3}, \ldots, \hat{s}_{ik,t+3}\}} \tag{2}$$

$\forall t$, and where λ is a proportionality constant and $\hat{s}_{ikt}$ is a filtered version of cosine similarity scores s_{ikt}, defined $\hat{s}_{ikt} = s_{ikt}$ if $s_{ikt} > 0.6$ and 0 otherwise. The indicator function serves to further sparsify the reward within its local temporal neighbourhood $\{\hat{s}_{ik,t-3}, \ldots, \hat{s}_{ik,t+3}\}$.

However, a concern with any reward shaping is that it can alter the underlying optimal policy (Randløv & Alstrøm, 1998; Ng et al., 1999). Therefore, we experimented with another scheme for defining intrinsic rewards:

$$\tilde{r}_{kt} = \lambda \sum_{i=1}^{H} (\gamma \cdot \hat{s}_{ik,t} - \hat{s}_{ik,t-1}) \tag{3}$$

where γ is the discount in the backbone RL algorithm. This formula for intrinsic reward can be shown to satisfy the necessary and sufficient criterion from (Ng et al., 1999) for policy invariance. Thus, if this form of the intrinsic reward is used then the ConSpec reward shaping scheme provably does not alter the optimal policy of the RL agent. In practice, we find that both forms of intrinsic reward work well. All experiments used equation 2 except for Figures 9 and 3 which investigated the use of equation 3.

2.6 INDUCTIVE BIASES IN CONSPEC THAT ENABLE RAPID CREDIT ASSIGNMENT

We aim to provide an intuition as to why the inductive biases present in ConSpec make it well suited to the problem of learning when success is contingent upon multiple critical steps. A key motivation for ConSpec is that learning a full model of the world and its transitions and values is very difficult, especially for a novice agent. ConSpec avoids learning a full model of the world and simply focuses on identifying key states for success, a more limited task that can be done rapidly. Concretely, previous approaches propagating value using a memory system or predicting rewards from intermediate states (Arjona-Medina et al., 2018; Hung et al., 2019; Raposo et al., 2021; Ren et al., 2021) can be characterized at a high level as being about learning to predict rewards given sequences of observations in memory, i.e. learning a model of the probability of success given a trajectory: $p(\text{success}|\{O_1, \ldots, O_T\})$. But predicting success as a function of previous steps can be a highly nontrivial non-linear function.

In contrast, ConSpec helps learn policies that hone in on states that are highly probable given success, i.e. it learns to identify states with high values for $p(O_t|\text{success})$. Therefore, ConSpec solves the reverse problem of predicting the critical states given success. This is a much easier problem because

it only looks at the direct dependency between the presence of a key state and success, without having to learn the joint dependency between all frames and reward. Put another way, whereas the likelihood of success is small given any specific trajectory, the likelihood of a critical state given success is high and easier to learn well.

Most crucially, this property arises ubiquitously in episodic experiences, and is therefore tailored for exploitation in RL, as ConSpec does. To see a concrete example of its application, conditioned on the successful acceptance of an ICLR paper, one can be sure that the critical states (an idea conceived, experiments run, paper written and reviewer concerns addressed) had each been achieved, and ConSpec proposes that learning these individual contingencies is a much easier problem than learning a world model that can predict whether a paper will be accepted given all of the past states. Under the above conditional independence assumption, the search space for the k critical O_t's $\propto \mathcal{O}(constant)$ rather than $\propto \mathcal{O}(k)$ or $\propto \mathcal{O}(2^k)$ which would be the case in most baseline RL algorithms, that do not assume this.

Algorithm 1 Jointly training ConSpec with an RL agent

Input: Current parameters (W, θ, ϕ, $h_{1\ldots H}$), memory buffers $\mathcal{B}$, $\mathcal{S}$, and $\mathcal{F}$, number of episodes in a mini-batch, B, and number of epochs to train for, E

for Epoch $e = 1 \ldots E$ **do**
  Collect a new mini-batch of B episodes using the current policy, π_ϕ
  Store trajectories $\{(O_1, a_1, r_1), \ldots, (O_T, a_T, r_T)\}$ in $\mathcal{B}$
  Update the $\mathcal{S}$ and $\mathcal{F}$ memory banks based on $\mathcal{B}$
  $\forall k, t$ calculate the latent representations: $z_{kt} \leftarrow f(O_{kt})$
  $\forall i, k, t$ calculate the similarity scores: $s_{ikt} \leftarrow cos(h_i, g_\theta(z_{kt}))$
  Use s_{ikt} $\forall i, k, t$ in buffer $\mathcal{B}$ to calculate intrinsic rewards, $\tilde{r}_{kt}$, according to formula 2
  Calculate total reward $\forall k, t$: $r_{kt}^{\text{Total}} = r_{kt} + \tilde{r}_{kt}$
  From total reward calculate RL loss, $\mathcal{L}_{\text{RL}}$ in-line with chosen RL backbone
  Use s_{ikt} $\forall i, k, t$ in buffers $\mathcal{S}$ and $\mathcal{F}$ to calculate $\mathcal{L}_{\text{ConSpec}}$ according to formula 1
  Update all parameters (W, θ, ϕ, $h_{1\ldots H}$) according to the loss $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{RL}} + \beta\mathcal{L}_{\text{ConSpec}}$
end for

3 Empirical results

We have tested ConSpec across a variety of RL tasks, ranging from simple grid worlds and video games to richer 3-D environments. We begin by demonstrating the unique benefits of ConSpec for tasks where success is contingent upon multiple important steps.

3.1 Testing ConSpec in grid world tasks with multiple critical states

As described above, a key difference between other solutions to the long-term credit assignment problem and ConSpec is that whereas other solutions have a search space that scales with the number of states that success is contingent upon, ConSpec has a constant search space. As such, we would expect a clear difference to emerge between ConSpec and other solutions as we increase the number of critical steps for success. Here we test that hypothesis.

We begin with some multi-key-to-door tasks in a grid world environment. In these tasks the agent must find one or more keys to open one or more doors (Fig. 2a). If the agent does not pick up all of the keys and pass through all of the doors success is impossible. To make the tasks more challenging for traditional Bellman back-up, the agent is forced to wait in a waiting room for many time steps in between the retrieval of each key. An episode is successful if and only if the agent exits the final door. When ConSpec is trained here, each prototype learned to represent a critical key-retrieval event (e.g. one prototype learned to hone in on the retrieval of the first key, another prototype on the second key, etc.). We demonstrated this by examining which state in different episodes maximally matched each prototype (Fig. 2b). Thus, ConSpec identifies critical states for success in a highly interpretable manner in the multi-key-to-door tasks.

We then tested ConSpec atop a Proximal Policy Optimization (PPO) backbone (Schulman et al., 2017; Kostrikov, 2018), i.e. we used a PPO agent that received intrinsic rewards from a ConSpec module per Algorithm 1. We compared performance to two other baseline solutions to long-term credit assignment in RL. The first baseline is a Synthetic Returns (SynthRs) system (Raposo et al., 2021),

which also attempts to identify critical states. But unlike ConSpec, it accomplishes this by attempting to predict the return based on linear regression from states in the replay buffer, and then uses these predictions to generate intrinsic rewards. The second baseline is a Temporal Value Transport (TVT) system (Hung et al., 2019). TVT uses a Reconstructive Memory Agent (RMA) (Hung et al., 2019; Wayne et al., 2018) that allows it to "transport" Bellman updates far back into the history of the agent in order to overcome long-term credit assignment challenges. Per our arguments above, one of the key differences between ConSpec and our two baselines (SynthRs and TVT) is that ConSpec is attempting to determine which states are most likely given success, whereas the two baselines are attempting to predict whether success is likely given a set of states in the memory buffer.

We see that ConSpec learns the multi-key-to-door tasks very rapidly, across a range of number of keys (Fig. 2c, *left*). Importantly, even as keys get added, ConSpec's training time remains constant when the exploration required for new keys is accounted for (Fig. 2c, *right*), consistent with the intuitions given above regarding the search space involved. In contrast, SynthRs and TVT solve the task for a single key (Fig. 7), but as keys get added, their performances collapse (Fig. 2d). These results highlight the difficulty of credit assignment with multiple contingencies faced by other algorithms, but easily accommodated by ConSpec.

Finally, we note that ConSpec learned these tasks not only atop a PPO backbone, but also when it was placed on top of an RMA backbone (Fig. 8 and also Fig. 7b), illustrating its flexibility. Taken together, these results demonstrate how ConSpec can be added to any RL system to solve challenging tasks with sparse rewards contingent upon multiple crucial steps.

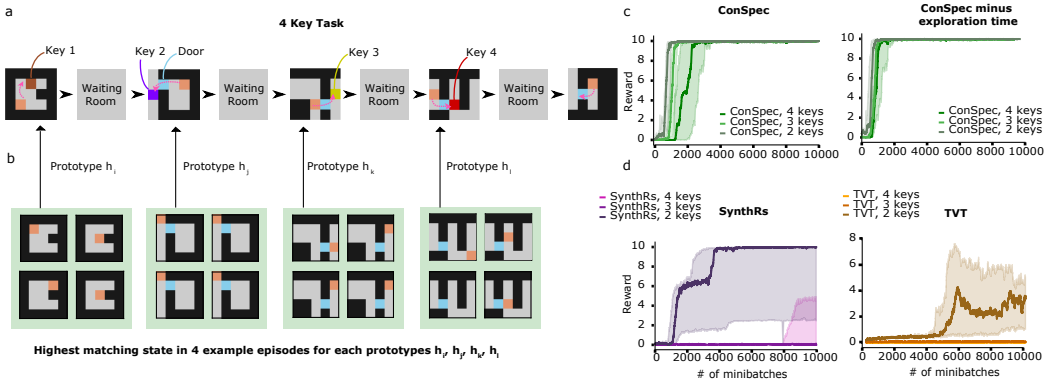

Figure 2: ConSpec rapidly learns tasks with multiple contingencies. (a) Protocol for multi-key-to-door task with 4 keys. (b) Plotted: states that maximally matched each prototype. Here, prototypes learn to hone in on states that depict retrieval of the key (where the agent is out the door and the coloured keys are gone i.e. retrieved). (c *left*) ConSpec rapidly learns the multi-key-to-door tasks, whereas (d) SynthRs and TVT performances collapse (same x and y axes). (c *right*) When exploration time is subtracted, ConSpec's training time $\propto \mathcal{O}(constant)$ even as the number of keys increases, affirming the search space predictions made in section 2.6.

3.2 Testing if ConSpec alters the optimal policy

Generating intrinsic rewards, a form of reward shaping, must be done carefully because reward shaping can alter the optimal policy since it changes the ultimate reward function that the agent experiences. This can lead to learning and performance instabilities, e.g. the construction of loops or a preference for sub-optimal trajectories. This must be avoided if the goal is to improve performance.

As noted above, one way to deal with this problem is to take advantage of a formal proof from Ng. et al (Ng et al., 1999) which provides the necessary and sufficient condition for a reward shaping function to be invariant to the optimal policy. We implemented a version of ConSpec that respected this condition (eqn. 3) and tested it on a suite of Atari games from the OpenAI Gym (Brockman et al., 2016). Notably, existing RL algorithms, such as PPO, are sufficient to solve these games. Thus, a policy invariant version of ConSpec added to PPO should perform at least as well as PPO on its own. Indeed, we found that across nine Atari games performance was unchanged by adding ConSpec to PPO (Fig. 3). As well, in the multi-key-to-door tasks, we confirmed that ConSpec applied with the policy invariant intrinsic reward was able to solve all these complex tasks with multiple contingencies (Fig. 9).

Notably, though, even when we used our original formulation of the intrinsic reward (eqn. 2), ConSpec also did not hurt performance on the Atari games, and of course, it improved performance on the multi-key-to-door tasks (Fig. 2c). We speculate that ConSpec does not create instabilities in the Atari games because it confers intrinsic rewards very sparingly and according to a relatively stringent criterion. Still, the use of the policy invariant version of the intrinsic reward provides both analytical and empirical guarantees that the underlying RL agent will not have its performance hurt by the use of ConSpec. Thus, ConSpec can be safely added to existing RL systems without breaking performance.

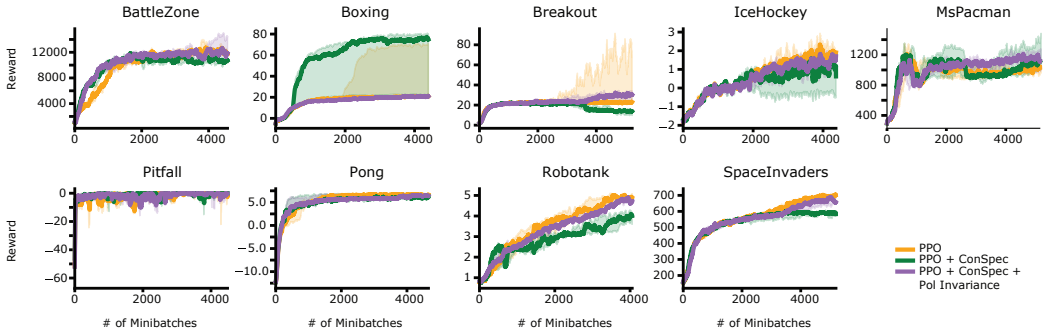

Figure 3: PPO vs PPO+ConSpec vs PPO+ConSpec+policy invariant intrinsic reward all learn to perform similarly well on Atari Gym tasks.

3.3 Testing ConSpec in 3D environments

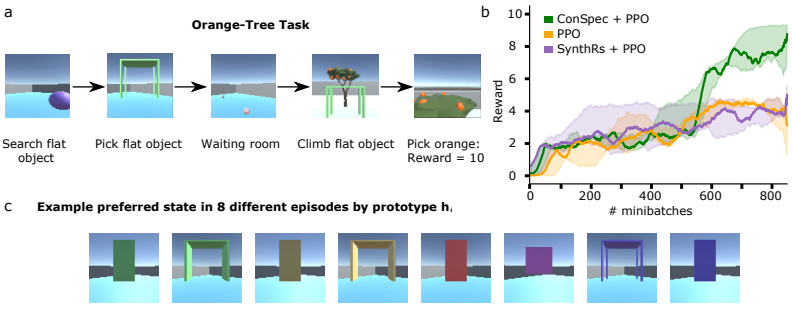

Figure 4: ConSpec learns invariant representations. (a) 3D OrangeTree task design. (b) ConSpec efficiently learns this task with substantially better average reward than the PPO and SynthRs baselines. (c) This prototype learns to robustly detect the retrieval of flat objects, invariant to variations in object type and colour.

ConSpec's contrastive system for learning prototypes confers two hypothesized benefits. First is the capacity to rapidly hone in on all the crucial states on which success is contingent upon. Second is the capacity to learn robust, invariant representations of such states. The latter hypothesis was first hinted at by inspecting the diversity of states that were mapped to particular prototypes in the multi-key-to-door task (Fig. 2b). Notably, prototypes each mapped to a specific key-retrieval event, even when the location of those events differed in different episodes. This suggests that ConSpec learns invariant representations, but it is a relatively toy example. To further investigate the invariance properties of ConSpec in non-toy settings, we used SilicoLabs'[1] Unity-based (Juliani et al., 2018) game engine to create virtual 3D environments. We generated a naturalistic variation of the key-to-door task, which we call the "OrangeTree" task, that permits us to study more interesting invariances. In stage 1 of this task, the agent sees two objects around the room, one of them round or spiky, and one of them flat (e.g. boxes or tables), and the agent can pick up one of the objects (Fig. 4a). In stage 2, the agent is again put in a waiting room. In stage 3, the agent must obtain oranges from the tree, but to do so, it must stand on a flat object in order to reach the oranges and receive reward. Thus, if the agent had picked a spiky or round object in stage 1, it would not have been able to pick oranges since only flat objects are stable enough to stand on.

[1]https://www.silicolabs.ca/

We trained ConSpec with a PPO backbone on the OrangeTree task. ConSpec could solve the OrangeTree task (Fig. 4b), but SynthRs on a PPO backbone, and vanilla PPO both found it difficult (Fig. 4b). To further dissect the reasons for ConSpec's abilities, we studied the invariances it learned. Among successful episodes, we identified the states that maximally matched the prototypes. Notably, one of the prototypes discovered an invariant feature common to all successes but not failures (Fig. 4c): the state of having picked up flat objects and not spiky or round ones. Moreover, it learned to prefer flat objects regardless of the type of object (i.e. box or table) and regardless of its colour. As such, the contrastive learning of ConSpec permitted the prototype to learn a critical feature for success, namely flatness, and to maintain invariance to irrelevant features of those objects.

3.4 Testing ConSpec in new contingencies

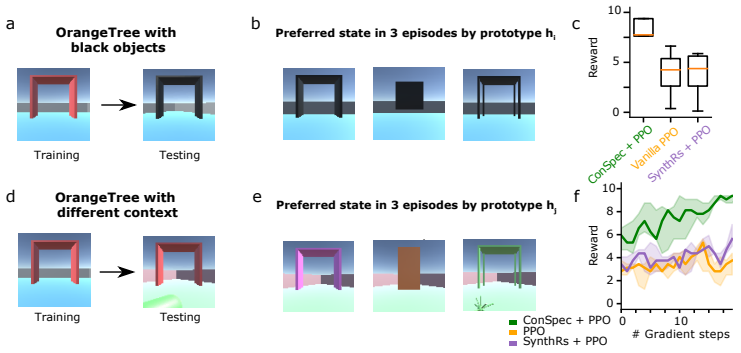

Figure 5: ConSpec's invariant representations help OoD generalization. (a, d) New OrangeTree tasks during Testing (a) with black boxes/tables, or (d) in a differently coloured room. Previously trained prototypes hone in on interpretable states even in these new contingencies (b,e) and are able to generalize (c) in zero-shot with never-before seen black objects, and (f) in few-shot in a new maze environment, approaching the maximum reward.

We asked whether ConSpec's capacity for finding invariances could aid in zero-shot learning when the sensory features of the critical states were altered. To test this, we made a variation of the OrangeTree task. During training, the agent saw flat objects that were magenta, blue, red green, yellow, or peach (Fig. 5a). During testing, a black table or box was presented, even though the agent had never seen such objects before. Intuitively, for the agent to succeed in this situation it must immediately recognize the black table or box as a flat object that is important for success despite its total lack of experience with such objects. Indeed, the agent with ConSpec successfully solved the task with the black objects with no more training, i.e. in a zero-shot manner (Fig. 5c). This shows that ConSpec was able to learn prototypes that discovered an invariant feature among successes (flatness), and was not confused by irrelevant sensory features such as colour, permitting it to generalize to contingencies involving colours never seen before.

To further test ConSpec's capacity for generalization, we made another variation of the OrangeTree task with the background environment changed, in a manner typical of out-of-distribution (OoD) generalization tasks (Arjovsky, 2021). In particular, testing took place in a different room than training, such that the new room had pink walls and a green floor (unlike the gray walls and blue floor of the training room) (Fig. 5d). In this few-shot task, after a brief period of acclimatization to the new environment (where the underlying PPO agent was trained for no more than 20 gradient descent steps while ConSpec was kept constant), the ConSpec agent was able to rapidly solve the task (Fig. 5f). For comparison, training from scratch in OrangeTree takes PPO with ConSpec hundreds of steps, and vanilla PPO by itself cannot train on the task at all within the allotted time. Moreover, as noted, the prototypes themselves were not trained further at all in the new environment. Thus, the learned prototypes generalize to the new environment in a zero-shot manner, and can be reused to rapidly retrain the underlying RL agent. Altogether, ConSpec's contrastive learning of prototypes allow it not only to very rapidly hone in on all the crucial states, but also has the added perk that the knowledge learned can help the agent rapidly generalize to new environments.

4 Related work

ConSpec is a relatively simple design, but it succeeds because it centralizes several key intuitions that it shares with other important works. ConSpec shares inductive biases with RUDDER (Arjona-Medina et al., 2018), TVT (Hung et al., 2019), Synthetic Returns (Raposo et al., 2021), and various attention-based architectures (Chen et al., 2021; Parisotto et al., 2019; Goyal et al., 2019; 2020) for

long-term credit assignment in RL, as discussed above in section 2.6. Crucially, ConSpec aims to directly focus on identifying a few critical states, a less burdensome task than modelling the value function, reward function, or actions taken for all encountered states as done in these other works.

It is also worth comparing ConSpec to another promising direction for improved credit assignment in RL, namely, Decision Transformers (Chen et al., 2021). Crucially, decision transformers require expert demonstrations, and learns poorly in the absence of near-optimal behavioural demonstrations, while ConSpec does not have such a requirement (see Fig. 11).

ConSpec was inspired, in part, by the contrastive learning literature in computer vision (Hadsell et al., 2006; Chen et al., 2020; van den Oord et al., 2018; He et al., 2019). Specifically, it was inspired by the principle that transforming a task into a categorization problem and using well-chosen positive and negative examples can be a very powerful means of learning invariant and generalizable representations. Along similar lines, (Srinivas et al., 2020; Anand et al., 2019; Sermanet et al., 2017; Finn et al., 2016; Ghosh et al., 2019; Eysenbach et al., 2022; Li et al., 2021; Agarwal et al., 2021) have begun to explore contrastive systems and binary classifiers to do imitation learning and reinforcement learning. With these works, ConSpec shares the principle that learning such an auxiliary classification is beneficial. But these works typically use classification and contrast to learn whole policies or value functions. In contrast, ConSpec uses contrastive learning for the purpose of learning prototypes in order to hone in on specific critical states. Identifying a small number of critical states was chosen as a less burdensome task than contrastively learning whole policies, thereby enabling very rapidly learning while still promoting generalizability.

5 Discussion and Conclusions

In this paper we presented ConSpec, a contrastive learning system that can be added to any RL agent to improve its long-term credit assignment and help it to identify critical states for success from sparse rewards. ConSpec works by learning prototypes of critical states with a contrastive loss that differentiates successful episodes from failed episodes. Adding ConSpec to an RL agent allows it to solve tasks it is otherwise incapable of solving, such as tasks with multiple contingencies like multi-key-to-door. Moreover, ConSpec can be designed to leave the optimal policy unchanged, allowing agents with ConSpec added to perform equally well on more standard RL tasks, such as Atari games. Importantly, we also demonstrated in 3D Unity environments that ConSpec can learn prototypes that are invariant to irrelevant features, allowing agents to engage in zero-shot generalization with the prototypes and rapidly generalize in OoD settings. Altogether, ConSpec is a potentially powerful add-on for RL systems to help resolve difficult challenges in credit assignment.

5.1 Neuroscience inspiration

Our design of ConSpec was ultimately inspired at a high level by theories from neuroscience. For example, it shares with episodic control and related algorithms (Lin, 2004; Lengyel & Dayan, 2007; Blundell et al., 2016; Pritzel et al., 2017; Moore & Atkeson, 1992; Schaul et al., 2015; Oh et al., 2018) the principle that a memory bank can be exploited for fast learning. But unlike episodic control, ConSpec exploits memory not for estimating a surrogate value function, but rather, for learning discrete prototypes of critical states that then receive intrinsic rewards. This manner of learning is loosely inspired by the neuroscience notion of episodic memories giving rise to semantic knowledge (Squire & Alvarez, 1995) which can then be used for triggering internal rewards (Miller et al., 2022). In-line with this, ConSpec takes further inspiration from the neuroscience literature on the hippocampus which suggests that the brain endeavours to discretely hone in on specific critical states in episodic experience (Sun et al., 2020; Zacks et al., 2001; Zheng et al., 358; Franklin et al., 2019). Finally, ConSpec also takes as a principle from neuroscience the idea that an agent can possess an underlying model-free RL system (e.g., the basal ganglia) with higher systems (e.g., the neocortex) (Miller et al., 2022) built (or evolved) on top of it, interacting with it, and providing learned rewards.

6 REPRODUCIBILITY

We aim to maximize the reproducibility of this study. The code and environments will be released in the camera-ready version of the manuscript. A detailed description and full pseudo-code of ConSpec is given in the method section and appendix. The ConSpec code itself has been submitted as supplementary materials in a ZIP file.

REFERENCES

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive Behavioral Similarity Embeddings for Generalization in Reinforcement Learning. *arXiv e-prints*, art. arXiv:2101.05265, January 2021.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised State Representation Learning in Atari. *arXiv e-prints*, art. arXiv:1906.08226, June 2019.

Jose A. Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. RUDDER: Return Decomposition for Delayed Rewards. *arXiv e-prints*, art. arXiv:1806.07857, June 2018.

Martin Arjovsky. Out of Distribution Generalization in Machine Learning. *arXiv e-prints*, art. arXiv:2103.02667, March 2021.

Nikhil Barhate. Minimal implementation of decision transformer. `https://github.com/nikhilbarhate99/min-decision-transformer`, 2022.

R. Bellman. Dynamic programming. *Technical report,Princeton University Press*.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-Free Episodic Control. *arXiv e-prints*, art. arXiv:1606.04460, June 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv e-prints*, art. arXiv:2106.01345, June 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv e-prints*, art. arXiv:2002.05709, February 2020.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning. *arXiv e-prints*, art. arXiv:1912.01588, December 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. *arXiv e-prints*, art. arXiv:1802.01561, February 2018.

Benjamin Eysenbach, Tianjun Zhang, Ruslan Salakhutdinov, and Sergey Levine. Contrastive Learning as Goal-Conditioned Reinforcement Learning. *arXiv e-prints*, art. arXiv:2206.07568, June 2022.

Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models. *arXiv e-prints*, art. arXiv:1611.03852, November 2016.

Nicholas T. Franklin, Kenneth A. Norman, Charan Ranganath, Jeffrey M. Zacks, and Samuel J. Gershman. Structured event memory: a neuro-symbolic model of event cognition. *bioRxiv*, 2019. doi: 10.1101/541607. URL `https://www.biorxiv.org/content/early/2019/10/10/541607`.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to Reach Goals via Iterated Supervised Learning. *arXiv e-prints*, art. arXiv:1912.06088, December 2019.

Alison Gopnik and David M. Sobel. Detecting blickets: How young children use information about novel causal powers in categorization and induction. *Child Development*, 71(5):1205–1222, 2000. ISSN 00093920, 14678624. URL http://www.jstor.org/stable/1131970.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent Independent Mechanisms. *arXiv e-prints*, art. arXiv:1909.10893, September 2019.

Anirudh Goyal, Alex Lamb, Phanideep Gampa, Philippe Beaudoin, Sergey Levine, Charles Blundell, Yoshua Bengio, and Michael Mozer. Object Files and Schemata: Factorizing Declarative and Procedural Knowledge in Dynamical Systems. *arXiv e-prints*, art. arXiv:2006.16225, June 2020.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pp. 1735–1742, 2006. doi: 10.1109/CVPR.2006.100.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. *arXiv e-prints*, art. arXiv:1911.05722, November 2019.

Chia-Chun Hung, Timothy P. Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *Nat Commun*, 10, 2019. doi: https://doi.org/10.1038/s41467-019-13073-w.

Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A General Platform for Intelligent Agents. *arXiv e-prints*, art. arXiv:1809.02627, September 2018.

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

Máté Lengyel and Peter Dayan. Hippocampal contributions to control: The third way. volume 20, 11 2007.

Bonnie Li, Vincent François-Lavet, Thang Doan, and Joelle Pineau. Domain Adversarial Reinforcement Learning. *arXiv e-prints*, art. arXiv:2102.07097, February 2021.

Longxin Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 2004.

Kevin J Miller, Matthew M Botvinick, and Carlos D Brody. Value representations in the rodent orbitofrontal cortex drive learning, not choice. *eLife*, 11:e64575, aug 2022. ISSN 2050-084X. doi: 10.7554/eLife.64575. URL https://doi.org/10.7554/eLife.64575.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1312.5602, December 2013.

Andrew Moore and Christopher Atkeson. Memory-based reinforcement learning: Efficient computation with prioritized sweeping. In S. Hanson, J. Cowan, and C. Giles (eds.), *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper/1992/file/55743cc0393b1cb4b8b37d09ae48d097-Paper.pdf.

Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 278–287. Morgan Kaufmann, 1999.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-Imitation Learning. *arXiv e-prints*, art. arXiv:1806.05635, June 2018.

Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing Transformers for Reinforcement Learning. *arXiv e-prints*, art. arXiv:1910.06764, October 2019.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural Episodic Control. *arXiv e-prints*, art. arXiv:1703.01988, March 2017.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari Human Benchmark. *arXiv e-prints*, art. arXiv:2003.13350, March 2020.

Jette Randløv and Preben Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pp. 463–471, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1558605568.

David Raposo, Sam Ritter, Adam Santoro, Greg Wayne, Theophane Weber, Matt Botvinick, Hado van Hasselt, and Francis Song. Synthetic Returns for Long-Term Credit Assignment. *arXiv e-prints*, art. arXiv:2102.12425, February 2021.

Zhizhou Ren, Ruihan Guo, Yuan Zhou, and Jian Peng. Learning Long-Term Reward Redistribution via Randomized Return Decomposition. *arXiv e-prints*, art. arXiv:2111.13485, November 2021.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. *arXiv e-prints*, art. arXiv:1511.05952, November 2015.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010. doi: 10.1109/TAMD.2010.2056368.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv e-prints*, art. arXiv:1707.06347, July 2017.

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-Contrastive Networks: Self-Supervised Learning from Video. *arXiv e-prints*, art. arXiv:1704.06888, April 2017.

David Sobel, Joshua Tenenbaum, and Alison Gopnik. Children's causal inferences from indirect evidence: Backwards blocking and bayesian reasoning in preschoolers. *Cognitive Science*, 28: 303–333, 05 2004. doi: 10.1016/j.cogsci.2003.11.001.

Larry R Squire and Pablo Alvarez. Retrograde amnesia and memory consolidation: a neurobiological perspective. *Current Opinion in Neurobiology*, 5(2):169–177, 1995. ISSN 0959-4388. doi: https://doi.org/10.1016/0959-4388(95)80023-9. URL https://www.sciencedirect.com/science/article/pii/0959438895800239.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. *arXiv e-prints*, art. arXiv:2004.04136, April 2020.

Chen Sun, Wannan Yang, Jared Martin, and Susumu Tonegawa. Hippocampal neurons represent events as transferable units of experience. *Nature Neuroscience*, art. 23, May 2020.

R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *MIT Press*.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv e-prints*, art. arXiv:1807.03748, July 2018.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z. Leibo, Adam Santoro, Mevlana Gemici, Malcolm Reynolds, Tim Harley, Josh Abramson, Shakir Mohamed, Danilo Rezende, David Saxton, Adam Cain, Chloe Hillier, David Silver, Koray Kavukcuoglu, Matt Botvinick, Demis Hassabis, and Timothy Lillicrap. Unsupervised Predictive Memory in a Goal-Directed Agent. *arXiv e-prints*, art. arXiv:1803.10760, March 2018.

JM Zacks, TS Braver, MA Sheridan, DI Donaldson, AZ Snyder, JM Ollinger, RL Buckner, and ME Raichle. Human brain activity time-locked to perceptual event boundaries. *Nature Neuroscience*, pp. 651–5, June 2001.

Jie Zheng, Andrea G. P. Schjetnan, Mar Yebra, Bernard A. Gomes, P. Mosher Clayton, Suneil K. Kalia, Taufik A. Valiante, Adam N. Mamelak, and Ueli Kreiman, Gabriel Rutishauser. Neurons detect cognitive boundaries to structure episodic memories in humans. *Nature Neuroscience*, 358.

A APPENDIX

A.1 DEFINING SUCCESS AND FAILURE

How best to define successful and unsuccessful episodes? There are three fairly obvious ways:

1. In the domain of sparse reward tasks, the definition of a "successful" episode is relatively straightforwardly defined as an episode that received one of the few rewards available, and a "failure" episode is defined as anything else.
2. In non-sparse reward settings, the notion of a "successful" episode can be defined as the top set of k episodes based on the sum of rewards.
3. In a goal-conditioned setting, a "successful" episode is simply one which achieves the goal.

In ConSpec we use the first approach, since the focus of this work is, in part, on solving tasks with sparse rewards and multiple contingencies leading to them. In one case we do take the second approach (with Atari games) (Brockman et al., 2016) to demonstrate that this can also work. We have yet to explore the third approach, which we leave for future research.

A.2 TASK IMPLEMENTATION DETAILS

In the multi-key-to-door tasks the agent must find one or more keys to open one or more doors (Fig. 2a). If the agent does not pick up all of the keys and pass through all of the doors success is impossible. In between each stage for picking up keys/passing through doors, the agent is forced to wait in a waiting room for many time steps in between the retrieval of each key. An episode is successful if and only if the agent exits the final door. Observations given to the RL agent were minimalist, at 5x5x3 pixels. Successful completion of the task resulted in a reward of 10. Tasks had alternating key retrieval and waiting periods, and the lengths of each of these stages were as follows:

Number of timesteps for each task stage:

Stage	Baseline key-to-door task	2-key task	3-key task	4-key task
Retrieve 1st key	10	10	10	10
Waiting period	85	45	45	45
Retrieve 2nd key		15	15	15
Waiting period		45	45	45
Retrieve 3rd key			15	15
Waiting period			45	45
Retrieve 4th key				15
Waiting period				45
Exit the final door	10	10	10	10
Total:	105	125	185	245

Figure 6: Protocols for multi-key-to-door experiments.

Atari tasks were implemented with frameskips of 4, and for computational expediency, and were terminated at 600 timesteps at which point rewards were tallied. Observations given to the RL agent were 84x84x3 pixels (where the 3 encodes colours RGB).

OrangeTree experiments were implemented using SilicoLabs' Unity ((Juliani et al., 2018)) based game engine to create virtual 3D environments. In stage 1 of this task, the agent sees two objects around the room, one of them round or spiky, and one of them flat (e.g. boxes or tables), and the agent can pick up one of the objects (Fig. 4). In stage 2, the agent is again put in a waiting room. In stage 3, the agent must obtain oranges from the tree, but to do so, it must stand on a flat object

in order to reach the oranges and receive reward. Successful completion of the task resulted in a reward of 10. Observations given to the RL agent were 84x84x3 pixels. The tasks were a total of 65 timesteps long, with 15 timesteps in Stage 1, 40 timesteps in Stage 2, and 10 timesteps in Stage 3.

A.3 EXPERIMENTAL AND ARCHITECTURAL DETAILS

$\mathcal{S}$ and $\mathcal{F}$ buffer sizes: All experiments used $\mathcal{S}$ and $\mathcal{F}$ buffer sizes of 16. For 3D OrangeTree experiments, although $\mathcal{S}$ and $\mathcal{F}$ buffers each held 16 episodes like all the other experiments, each gradient step was done on a random minibatch of 8 episodes sampled from this 16 due to computational resource limitations. In Atari tasks each gradient step was done on a random minibatch of 4 episodes sampled from the buffers of 16 for similar reasons.

Number of prototypes: Across all implementations of ConSpec for various tasks, the number of prototypes used was set as 8, except for the Atari tasks, which due to computational resource limitations, used only 3 prototypes and the implementation of ConSpec on an RMA backbone, which used 12 and gave rapid learning (Fig 8).

Minibatch size: all models used standard minibatch size $B = 16$, except for the Atari tasks, which due to computational resource limitations, used minibatch size $B = 8$.

Seeds in experiments: All multi-door-to-key experiments averaged over 10 seeds. All 3D OrangeTree experiments averaged over 5 seeds and the 9 Atari experiments averaged over 5 seeds each. All plots show median and quartile range.

Architectural details for models used: For all multi-key-to-door tasks, the learned encoder of the underlying RL agent was a single layer 2-D convolutional neural network (outchannels = 32, kernel = 3) with ReLU activation, followed by a final linear layer, and a 512-unit GRU , as was done in the (Hung et al., 2019) codebase for 2D key-to-door tasks. The convnet encoder for the ConSpec module was identically sized, but separately parameterized so that there would be no interference between the ConSpec module and the underlying RL agent. In the ConSpec module, the nonlinear projection g_θ in ConSpec was a 2-layer MLP with 1010 units in the intermediate layers and the final output layer, and ReLU activations between layers. Inspired by the contrastive learning literature, where the importance of having a large number of negative samples relative to positive samples has been observed (Chen et al., 2020), the success buffers were filled at a slower rate than the failure buffers: at most 2 new successful trajectories were added to $\mathcal{S}$ from each minibatch, while there was no limit to the number of new failed trajectories added to $\mathcal{F}$ from each minibatch. As discussed in section 2.4, updates to the memory buffers for each prototype were terminated upon reaching criterion that this prototype sufficiently differentiated between successes and failures above threshold $\mathcal{T}$. Across all experiments, this criterion was defined as when the average maximum cosine similarity scores among trajectories in $\mathcal{S}$- $\mathcal{F}$ differed by $> \mathcal{T}$ and that the scores in $\mathcal{S}$ were themselves $> \mathcal{T}$, all for at least 25 consecutive gradient steps. We found that $\mathcal{T} = 0.6$ worked well in practice, and this value was used throughout the experiments.

Atari tasks and 3D gridworld tasks possess larger observations, so the learned encoder used was a much larger Impala encoder (Espeholt et al., 2018; Cobbe et al., 2019) using 3 Impala layers with 16, 32, and 32 channels respectively. For these experiments, the nonlinear projection g_θ in ConSpec was a 2-layer MLP with 100 units in the intermediate layers and the final output, and ReLU activations between layers.

One of the aims of the Atari experiments was to apply ConSpec in a setting that requires a different definition of successes and failures, as elaborated in section A.1. Specifically, successful trajectories were defined as the highest cumulatively scoring ones from each minibatch, while comparatively, failed trajectories were randomly chosen from the remainder of the minibatch. To extend ConSpec to this setup, each of ConSpec's prototypes took not just the s_{ikt} with the top cosine similarity for each trajectory, but rather, averaged the s_{ikt}'s with the top-k cosine similarities for each trajectory (where $k = 25$).

For the hyperparameter λ, which scales the intrinsic reward (see equation 2), we used a different value in each set of experiments (multi-key-to-door, Atari, and OrangeTree). Specifically, we used $\lambda = 0.2$ for the multi-key-to-door, $\lambda = 1.0$ for the policy-invariant version of ConSpec in the multi-key-to-door experiment, $\lambda = 0.5$ for the Atari games, and $\lambda = 0.5$ for OrangeTree. These values

were determined via a logarithmic grid-search that selected for best reward. Other hyperparameters were found via a logarithmic grid-search, and were shared across all tasks:

- RL agent learning rate: 2×10^{-4}
- ConSpec learning rate: 2×10^{-3}
- α: 0.2
- β: 1.0

Models using a PPO backbone used the Adam optimizer with optimal PPO hyperparameters based on the repository (Kostrikov, 2018) (values below). The only major change to the hyperparameters used was the entropy coefficient used (0.02, rather than 0.01 from the repository, in order to encourage exploration necessary in the multi-key-to-door tasks):

- learning rate: 2×10^{-4}
- Adam epsilon: 1×10^{-5}
- reward discount factor: 0.99
- clipping 0.08
- value loss coefficient: 0.5
- entropy coefficient: 0.02

PPO-backbone models that were engaged with either the multi-door-to-key tasks or the 3D Orange-Tree task had reward normalized to the range $[0, 1]$. PPO-backbone models that were engaged with the 9 Atari tasks underwent reward clipping to the sign of the reward received at each timestep, as is standard practice (Mnih et al., 2013).

Models using an RMA backbone (including the implementation of TVT) were taken directly from the (Hung et al., 2019) codebase without modification of the architecture. They used the Adam optimizer with PPO hyperparameters based on the repository (Hung et al., 2019). The only major change to the hyperparameters used was the entropy coefficient used (0.1, rather than 0.05 from the repository, in order to encourage exploration necessary in the multi-key-to-door tasks):

- learning rate: 2×10^{-4}
- Adam epsilon: 1×10^{-6}
- agent discount: 0.92
- clipping 0.1
- reading strength coefficient: 50.
- image strength coefficient: 50.
- entropy coefficient: 0.1

The implementation for SynthRs used a synthetics returns MLP that was sized analogously as g_θ in ConSpec in the respective experiments (2-layer MLP with ReLU and 1010 or 100 units respectively). This module was put on top of a PPO backbone. Its convolutional neural network encoder was identical to the corresponding encoders used in ConSpec. Its learning rates were matched to the learning rates used in ConSpec to enable comparable rates of learning (2×10^{-4} for the underlying PPO agent, and 2×10^{-3} for the synthetics return module). The optimal weight of the SynthRs loss, β, as defined in Algorithm 1 was determined via further logarithmic grid-search, and $\beta = 10^{-4}$ worked well across experiments. Further hyperparameters were as follows, taken amongst the optimal values used in the SynthRs paper (Raposo et al., 2021):

- state-associate alpha: 0.01
- state-associate beta: 0.0

SynthRs was unable to solve the 3D OrangeTree task in the alotted amount of time (Figure 10), even though it did work well in the key-to-door tasks for a low number of keys. However, we found that

replacing the sigmoid with a softmax on the last layer of the gate g improved performance, so this was the version of SynthRs used in the 3D OrangeTree.

The implementation of Decision Transformers was taken from the repository (Barhate, 2022), but we added a convolutional neural network encoder that was identical to the encoder used in the ConSpec experiment. Moreover, its learning rate (2×10^{-4}) was matched to the learning rate used in ConSpec to enable comparable rates of learning. All other hyperparmeters were taken without modification from the repository (Barhate, 2022) and were:

- weight decay: 1×10^{-4}
- dropout probability: 0.1
- context length: 20
- blocks: 3

A.4 Baseline performances in vanilla single key-to-door task

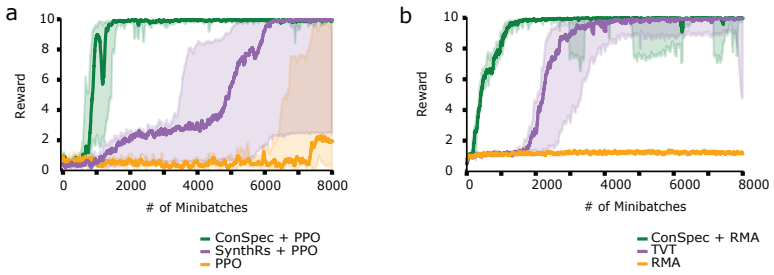

Figure 7: Baseline single key-to-door task. (a) Performance of ConSpec on a PPO backbone vs SynthRs on a PPO backbone vs PPO. (b) Performance of ConSpec on an RMA backbone vs TVT vs RMA.

A.5 ConSpec on an RMA backbone, on the multi-key-to-door tasks

Here, we show the performance of ConSpec implemented on an RMA rather than PPO backbone, training on the multi-key-to-door tasks. As shown, ConSpec was successfully able to rapidly learn and converge in each multi-key-to-door task, illustrating its flexibility in credit assignment, regardless the underlying RL agent.

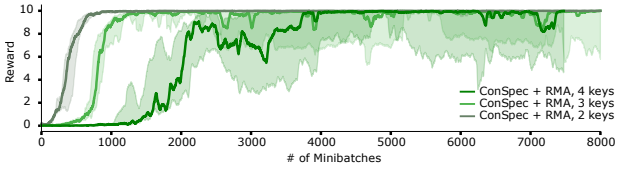

Figure 8: Performance of ConSpec on an RMA backbone, on the multi-key-to-door tasks.

A.6 ConSpec with Policy invariant intrinsic rewards, on the multi-key-to-door tasks

Here, we show the performance of ConSpec with the policy-invariant version of the intrinsic reward (equation 3) on the multi-key-to-door tasks. As shown, ConSpec was successfully able to rapidly learn each task.

A.7 SynthRs with sigmoid

A.8 ConSpec vs Decision Transformers

It is worth comparing ConSpec to another promising direction for improved credit assignment in RL, namely, Decision Transformers (Chen et al., 2021). A critical difference between ConSpec and

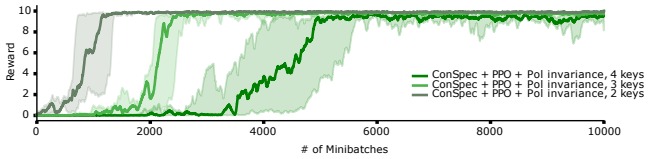

Figure 9: Performance of ConSpec with policy-invariant intrinsic reward, on the multi-key-to-door tasks.

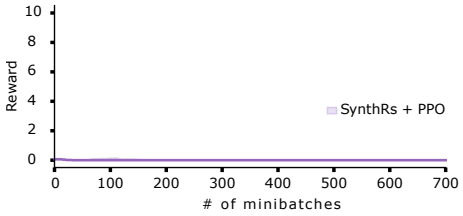

Figure 10: Performance of SynthRs with sigmoid on the OrangeTree task

Decision Transformers is that ConSpec can learn with no curation of the dataset involved. In contrast, Decision Transformers work poorly when trained on random policy-generated episodes, even when those episodes are limited to successful episodes 11. Of course, Decision Transformers learn very well when the policy is instead selected to be near-optimal for the task, affirming that Decision Transformers are useful for behavioural cloning from for expert demonstrations, but not able to learn from scratch like a standard RL system with ConSpec added 11.

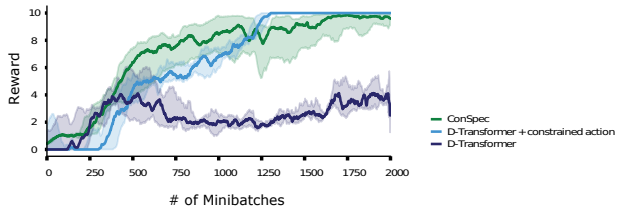

Figure 11: ConSpec is more robust than Decision Transformers on vanilla key-to-door because of its objective in learning states rather than predicting actions. Decision transformers is not able to solve the key-to-door task unlike ConSpec but is able to solve it if actions are constrained in their freedom.