SELF-ATTENTIONAL CREDIT ASSIGNMENT FOR TRANSFER IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The ability to transfer knowledge to novel environments and tasks is a sensible desiderata for general learning agents. Despite the apparent promises, transfer in RL is still an open and little exploited research area. In this paper, we take a brand-new perspective about transfer: we suggest that the ability to assign credit unveils structural invariants in the tasks that can be transferred to make RL more sample efficient. Our main contribution is SECRET, a novel approach to transfer learning for RL that uses a backward-view credit assignment mechanism based on a self-attentive architecture. Two aspects are key to its generality: it learns to assign credit as a separate offline supervised process and exclusively modifies the reward function. Consequently, it can be supplemented by transfer methods that do not modify the reward function and it can be plugged on top of any RL algorithm.

1 INTRODUCTION

To some, intelligence is measured as the capability of transferring knowledge to unprecedented situations. While the notion of intellect itself is hard to define, the ability to reuse learned information is a desirable trait for learning agents. The coffee test (Goertzel et al., 2012), presented as a way to assess general intelligence, suggests the task of making coffee in a completely unfamiliar kitchen. It requires a combination of advanced features (planning, control and exploration) that would make the task very difficult if not out of scope for the current state-of-the-art Reinforcement Learning (RL) agents to learn. On the other hand, it is solved trivially by humans, who exploit the universally invariant structure of coffee-making: one needs to fetch a mug, find coffee, power the coffee machine, add water and launch the brewing process by pushing the adequate buttons. Thus, to solve the coffee test, transfer learning appears necessary. Were we to possess a random kitchen simulator and a lot of compute, current transfer methods would still fall short of consistently reusing structural information about the task, hence also falling short of efficient adaptation.

Credit assignment, which in RL refers to measuring the individual contribution of actions to future rewards, is by definition about understanding the structure of the task. By structure, we mean the relations between elements of the states, actions and environment rewards. In this work, we investigate what credit assignment can bring to transfer. Encouraged by recent successes in transfer based on supervised methods, we propose to learn to assign credit through a separate supervised problem and transfer credit assignment capabilities to new environments. By doing so, we aim at recycling structural information about the underlying task.

To this end, we introduce SECRET (SElf-attentional CREdit assignment for Transfer), a transferable credit assignment mechanism consisting of a self-attentive sequence-to-sequence model whose role is to reconstruct the sequence of rewards from a trajectory of agent-environment interactions. It assigns credit for future reward proportionally to the magnitude of attention paid to past state-action pairs. SECRET can be used to incorporate structural knowledge in the reward function without modifying optimal behavior, as we show in various generalization and transfer scenarios that preserve the structure of the task.

Existing backward-view credit assignment methods (Arjona-Medina et al., 2019; Hung et al., 2018) require to add auxiliary terms to the loss function used to train agents, which can have detrimental effects to the learning process (de Bruin et al., 2018), and rely on an external memory, which hinder the generality of their approach. SECRET does neither. Also, as we show in Sec. 3.1, the architecture we consider for SECRET has interesting properties for credit assignment. We elaborate about our

novelty with respect to prior work in Sec. 4. We insist on the fact that the focus of our work is on transfer and that it is not our point to compete on credit assignment capabilities.

We would like to emphasize several aspects about the generality of SECRET: 1) our method does not require any modification to the RL algorithm used to solve the tasks considered, 2) it does not require any modification to the agent architecture either and 3) it does not alter the set of optimal policies we wish to attain. Moreover, our method for credit assignment is offline, and as a result, it can use interaction data collected by any mean (expert demonstrations, replay memories (Lin, 1992), backup agent trajectories...). We believe that this feature is of importance for real-world use cases where a high number of online interactions is unrealistic but datasets of interactions exist as a byproduct of experimentation.

Background We place ourselves in the classical Markov Decision Process (MDP) formalism (Puterman, 1994). An MDP is a tuple (S, A, γ, R, P) where S is a state space, A is an action space, γ is a discount factor ($\gamma \in [0, 1)$), $R : S \times A \times S \to \mathbb{R}$ is a bounded reward function that maps state-action pairs to the expected reward for taking such an action in such a state. Note that we choose a form that includes the resulting state in the definition of the reward function over the typical $R : S \times A \to \mathbb{R}$. This is for consistency with objects defined later on. Finally, $P : S \times A \to \Delta_S$ is a Markovian transition kernel that maps state-action pairs to a probability distribution over resulting states, Δ_S denoting the simplex over S.

An RL agent interacts with an MDP at a given timestep t by choosing an action $a_t \in A$ and receiving a resulting state $s_{t+1} \sim P(\cdot|s_t, a_t)$ and a reward $r_t = R(s_t, a_t, s_{t+1})$ from the environment. A trajectory $\tau = (s_i, a_i, r_i)_{i=1,...,T}$ is a set of state-action pairs and resulting rewards accumulated in an episode. The performance of an agent is evaluated by its expected discounted cumulative reward $\mathbb{E}\big[\sum_{t=0}^{\infty} \gamma^t r_t\big]$. In a partially observable MDP (POMDP), the agent receives at each timestep t an observation $o_t \sim \mathcal{O}(\cdot|s_t, a_t)$ that contains partial information about the underlying state of the environment.

2 SECRET: SELF-ATTENTIONAL CREDIT ASSIGNMENT FOR TRANSFER

SECRET uses previously collected trajectories from environments in a source distribution. A self-attentive sequence model is trained to predict the final reward in subtrajectories from the sequence of observation-action pairs. The distribution of attention weights from correctly predicted nonzero rewards is viewed as credit assignment. In target environments, the model gets applied to a small set of trajectories. We use the credit assigned to build a denser and more informative reward function that reflects the structure of the (PO)MDP. The case where the target distribution is identical to the source distribution (in which we use held-out environments to assess transfer) will be referred to as generalization or *in-domain transfer*, as opposed to *out-of-domain transfer* where the source and the target distributions differ.

2.1 SELF-ATTENTIONAL CREDIT ASSIGNMENT

Credit assignment as offline reward prediction We learn to assign credit through an offline reward prediction task, based on saved trajectories of agent-environment interactions. We create a sequence-to-sequence (seq2seq) model (Sutskever et al., 2014) that takes as input the sequence of observation-action pairs and has to reconstruct the corresponding sequence of environment rewards. Being offline, the reward prediction task is learned separately from the RL task, and the reward prediction model does not share representations with the agent. This way, the representations learned for credit assignment do not affect or get mixed with the representations learned for control. Operating offline brings several advantages: one can directly interact with the replay memory of agents and even use expert demonstrations or arbitrary saved transitions as a source of supervision, which could be useful in settings where on-policy interactions are costly, such as robotics. We equip our seq2seq model with an attention mechanism (Bahdanau et al., 2015) and view the attention weights of the reward reconstruction task as our primary source of assigned credit. The motivation to do so is that the seq2seq model looks into the past to find predictive signal in order to reconstruct the reward, so observation-action pairs it attends to should be those which reduce its uncertainty about the future, in other words those that explain future reward and should be credited.

On the use of observations In MDPs, environment states follow the Markov property: they summarize the history of previous interactions and are sufficient to predict the future. As such, predictive models are highly biased towards focusing on the current sequence element, which hinders credit assignment. Under that consideration, when dealing with MDPs, we turn states into observations by applying transformations that hide a certain amount of information from states and break the Markov assumption. For instance, in gridworlds with visual states, we crop the image and get a player centered image with a given window size. Doing so encourages the model to look into the past to find predictive signal, and allow us to track the relative importance given to each element to reconstruct the credit assigned. Note that in POMDPs, this is unnecessary since the observations agents get from the environment are incomplete by construction. In the following, we use o to refer to all kinds of observations and $h(s)$ to specify observations created from states.

Self-attention for credit assignment Unlike other seq2seq architectures, self-attentive models like Transformers (Vaswani et al., 2017) have direct computational paths between pairs of sequence elements, due to their representations that depend on projections of all sequence elements. This feature is key to long-term credit assignment. As an illustration, consider an RL task where the terminal reward depends only on the first observation, which is drawn randomly. Predicting the reward correctly requires to remember the first observation, which would be very challenging for a recurrent architecture whose memory goes through $O(n)$ transformations, n being the size of the sequence. On the other hand, a self-attentive model directly accesses the value of the initial observation, which makes credit assignment easier.

Reward prediction architecture We use a Transformer decoder with a single self-attention layer (Lin et al., 2017) and a single attention head. The model input is a sequence of observation-action couples $(o_t, a_t)_{t=0,\dots,T}$. Each observation goes through a series of convolutional layers (for visual inputs) followed by a series of feed-forward layers. Each action representation, a one-hot vector in the discrete action case, is concatenated to the learned observation embedding. Those representations are combined with positional encoding and fed to a self-attention layer and then to a position-wise feedforward layer that outputs logits for reward prediction classes.

Self-attention is an attention mechanism with parameterization (W_k, W_q, W_v), each matrix belonging to $\mathbb{R}^{d_i \times d_k}$, that puts sequence elements in relation by computing non-linear similarity scores for all pairs of elements in the sequence. To do so, each sequence element is mapped to a query vector that is matched against keys and values obtained from the previous elements. To be consistent with the goal of assigning credit, the model should not be able to peek into the future. Thus, we restrict the computational window of each sequence element to the information stored in representations of the previous elements in the sequence and its own by multiplying a causal mask M to the result of the pairwise similarity computations.

Let $X = (x_t)_{t=0,\dots,T} \in \mathbb{R}^{T \times d_i}$ denote the input sequence in a matrix form, x_t being the result of internal computations of the model on its tth input. In the same fashion, we note $Z = (z_t)_{t=0,\dots,T} \in \mathbb{R}^{T \times d_k}$ the sequence resulting from the application of self-attention. We then have

$$Z = \text{softmax}\left(\frac{M \odot (QK^T)}{\sqrt{d_k}}\right) V,$$

where $Q = XW_q \in \mathbb{R}^{T \times d_k}$ stores queries, $K = XW_k \in \mathbb{R}^{T \times d_k}$ keys, and $V = XW_v \in \mathbb{R}^{T \times d_k}$ values as linear projections of the input; d_k stands for the dimension of the key vectors, $M \in \{0, 1\}^{T \times T}$ is a binary matrix that acts as a causal mask (a lower triangular matrix), and $\odot$ is the Hadamard product.

Notably, the resulting observation-action representation can be viewed as a linear combination of the values of previous elements: $z_t = \sum_{i=0}^{t} \alpha_{i \leftarrow t} v_i$ where $\alpha_{\cdot \leftarrow t} = (\alpha_{i \leftarrow t})_{i=1,\dots,t} \propto \exp(\langle q_t, k_i \rangle / \sqrt{d_k})$. The vector α_t contains the normalized attention weights for the prediction at timestep t and sums to 1. Since observations contain only a portion of their initial information, the fact that the model succeeds in the prediction task indicates that it reconstructed the missing information from its past. Therefore, attention weights themselves can be viewed as a form of credit assignment, and will be used as such in what follows.

While performing regression on the rewards could also be an option, our experiments found that regression tends to converge to poor local optima. Consequently, we predict the sign of the experi-

enced rewards: $q(r) = \text{sign}(r)$ with $\text{sign}(0) = 0$. We chose the sign as the classification target for its invariance to the scale of the rewards. We use a weighted sequential cross-entropy as the loss function over the class-wise model predictions $f_{\theta,c}$, writing $\tau(o, a)$ the subtrajectory of τ ending with the observation-action couple (o, a) to translate the effect of the binary mask:

$$\mathcal{L}_\theta(\tau) = -\sum_{c\in\{-1,0,1\}} \frac{w(c)}{|\tau|} \sum_{(o,a,r)\in\tau} \mathbb{1}\{q(r) = c\} \log\big(f_{\theta,c}(\tau(o, a))\big).$$

We have found class weighting to be very important to keep the variance of prediction metrics across a variety of datasets of sampled trajectories low.

Generating trajectories To train SECRET, we generate a dataset of trajectories which contains a certain proportion of successful trajectories. If source environments are simple enough so that the task has sufficient chance to be solved by acting randomly, we use a random policy to generate trajectories. For more complex distributions of environments, we use a RL agent (either trained or in the learning phase) to generate trajectories. We think purely exploratory methods like Ecoffet et al. (2019) could have advantages over using an RL agent and leave the study of their use for future work.

2.2 LEVERAGING CREDIT VIA REWARD SHAPING

In this subsection, we explain how we use credit assignment to densify the reward function and make learning more sample-efficient.

Reward shaping In RL, agents often deal with sparse rewards that make the learning process slow. Reward shaping Ng et al. (1999) is a technique that aims at densifying the reward so as to improve sample efficiency. It defines a class of reward functions that can be added to the original environment rewards without modifying the set of optimal policies. For a given MDP $M = (S, A, \gamma, R, P)$, we define a new MDP $M' = (S, A, \gamma, R', P)$ where $R' = R + F$ is the shaped reward and F the shaping. The reward shaping theorem states that if there exists a function ϕ such that $F : (s, a, s') \to \gamma\phi(s') - \phi(s)$, then M and M' admit the same set of optimal policies. ϕ is called a potential function. With domain knowledge, one can use reward shaping to design more informative reward functions without encouraging unwanted behavior. Nevertheless, shaping rewards requires good priors for the task to solve and the potential function must often be engineered manually.

Since SECRET weighs the contribution of observation-action pairs to future reward, we use it to derive a shaped reward that corresponds to the sum of future reward reachable from the underlying state, weighted by the attention calculated by the model. We explain the process in the following.

Computing the potential function We define the redistributed return $R^{\leftarrow}_\tau$ of a trajectory τ as:

$$R^{\leftarrow}_\tau(s, a) = \sum_{t=1}^{T} \mathbb{I}\{s_t = s, a_t = a\} \sum_{i=t}^{T} \alpha_{t\leftarrow i} r(s_i, a_i),$$

where $\alpha_{i\leftarrow j}$ is the attention weight on (o_i, a_i) when predicting the reward r_j and s_i are environment states. Indeed, SECRET uses observations but we keep the states they are constructed from to compute the potential. In POMDPs, we recover an approximate state from the observation, see Appendix A for details.

To compute the potential function, we generate a set D of trajectories like described in Sec. 2.1. Since we operate on trajectories, the same state-action pair can appear twice in a sequence and benefit from a different amount of attention, which is why we must include the first summation. In the reward shaping formalism, the potential function ϕ depends only on the state. To stay within its bounds, we define ϕ as the forwarded redistributed return. It is computed as the following estimate:

$$\hat{\phi}(s) = \frac{1}{|D|} \sum_{\tau\in D} \sum_{t=1}^{T} \mathbb{I}\{s^{(\tau)}_t = s\} R^{\leftarrow}_\tau(s^{(\tau)}_{t-1}, a^{(\tau)}_{t-1}).$$

Some states are generally missing from the data distribution induced by the set of trajectories used. For those states, we set to potential to 0, which results in a $-\hat{\phi}(s)$ additional reward when transitioning to those from the state s. As a result, it gives agents incentive to stay on the support of the data distribution unless they encounter high-reward states.

Because it relies on reward shaping, SECRET conserves optimal policies. We empirically find that agents learn faster with the resulting augmented reward function. A way to look at it is that we densify the learning signal and bias the agent towards behaviors that encourage future rewards.

2.3 TRANSFERRING CREDIT ASSIGNMENT

We start by conveying intuition as to why SECRET should transfer to new environments. In other fields than RL, seq2seq models similar to that of SECRET have shown outstanding transfer capabilities (Devlin et al., 2018; Peters et al., 2018; Howard and Ruder, 2018), even in low-resource settings (Zoph et al., 2016). In transfer scenarios that preserve the structure of the MDP, the optimal finegrained control sequence can vary drastically from one environment to another. This is why credit assignment is an interesting alternative to the transfer of weights: given an underlying environment state and a specific action, their contribution to future rewards is not fundamentally altered. Such scenarios include specific changes in the state (or observation) distribution and changes to the reward function that preserve the rank of the rewards. These also include changes in the dynamics of the environment, and though it affects credit assignment, we show later on that SECRET adapts surprisingly well to such scenarios. Another point that motivates the use of our method for transfer is the fact that we keep the representations learned for credit assignment separate from the control representations learned by agents. Indeed, de Bruin et al. (2018) showed that RL representations were not optimal for transfer.

Transfer setting We argue that transfer should be considered effective when agents learn to solve target tasks efficiently because efficiency gains in the target domain compound while the cost of training in the source is fixed. Hence, we use the Total Target Time Scenario metric (Taylor and Stone, 2009) to assess transfer. Nevertheless, collecting trajectories in the source domain can be costly. We report the number of trajectories used to train SECRET in each scenario.

As before, SECRET is trained on episodes of interaction sampled from the source distribution. In each target environment, we sample multiple trajectories (see the following section for details about the policies used to generate the trajectories). We then compute the attentional potential function by calculating an estimate of the expected redistributed reward, as described in Sec. 2.2.

3 EXPERIMENTS

In this section, we aim to answer the following questions: can SECRET improve the sample efficiency of learning for RL agents? Does it generalize and/or transfer? How does it compare to transfer baselines? Is the credit assigned by SECRET interpretable?

The Triggers environment We introduce Triggers, an interpretable and customizable environment that we use to assess the quality of the credit inferred with our method. In Triggers, the agent is located in a two-dimensional bounded grid. Its actions consist solely of moving of one cell in one of the cardinal directions. Any action that would lead the agent outside the boundaries of the environment (as indicated by the walls in the figure) is ignored but still counted as an action taken by the agent. The goal of the agent (represented as a yellow square) is to activate all the switches (red squares) and then collect all the prizes (pink squares). Prizes are the only source of reward and give a -1 penalty unless all switches are activated, in which case they give a $+1$ bonus. Both prizes and switches disappear once collected. The main feature of Triggers is that every positive reward is conditional to the presence of a known subset of states in

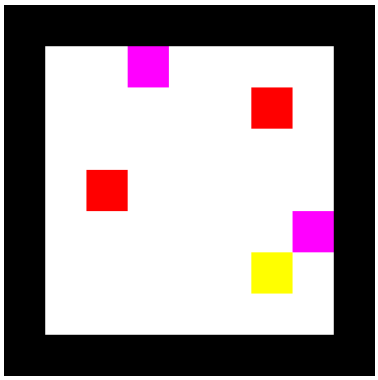

Figure 1: Example of a Triggers environment.

the agent history, and thus credit assignment can be assessed in a *rigorous* way. Some instances of Triggers can prove challenging to solve optimally for traditional RL methods since agents have to activate every Triggers before experiencing rewards. Triggers environments being MDPs, we turn their states into observations by cropping the view around the agent. We use 3x3 windows in all our experiments. Trajectories are generated with random policies.

DMLab keys doors We use the `keys_doors_puzzle` 3D environment from DMLab (Beattie et al. (2016)) in which the agent must locate keys whose colors indicate the doors they open. It can only possess one key, therefore picking the wrong key prevents it from reaching further rewards. The agent receives as input what would correspond to a first person view of what is in its line of sight. It can move forward, backward and rotate. Each key picked up grants a $+1$ bonus, equally to each door opened. Independently, a cake rewards the agent by a $+50$ increase in score when collected. We do not apply any transformation to the agent inputs. In that setup, agents benefit from understanding the link between keys and doors. We hypothesized that our credit assignment mechanism might identify this relation and reward the agent for picking up keys. To assert this, we modified the setting so that picking up keys does not provide rewards. Additionally, the visual input is richer than the one from Triggers environments and the average number of steps per episode is extended. Finally, agents move and rotate across the room. Since picking up a key does not require to actually see the key, it can be hard to know whether a key was taken and predict further door opening rewards. Trajectories are generated with a trained agent.

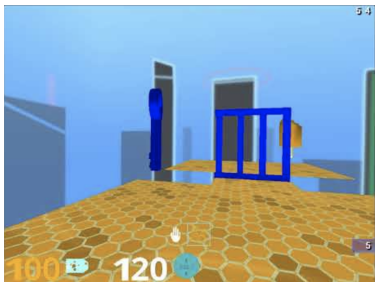

Figure 2: Observations from DMLab are first person views.

Agents used We use tabular Q-learning (Watkins and Dayan, 1992) for in-domain and out-of-domain experiments in Triggers except for the transfer to environments with modified dynamics where we use Deep Q-Networks (DQN) (Mnih et al., 2015). We use Proximal Policy Optimization (PPO) Schulman et al. (2017) agents for in-domain experiments in DMLab.

3.1 CREDIT ASSIGNMENT

We provide an analysis of the credit inferred by SECRET. The analysis is qualitative and quantitative, since we rely on both visual assessment and binary detection metrics.

The process of evaluating credit assignment in Triggers goes as follows: we first generate trajectories and train the model. We then compare the credit assigned by SECRET on trajectories sampled from held-out environments to a ground truth credit assignment. We build that ground truth by exploiting the exact knowledge of where triggers are. It is a vector that is 0 almost everywhere and 1 on state-action couples that precede the activation of a Triggers. By doing so, we explicitly target the state-action couples whose resulting state is causally linked to the reward experienced later.

We find the redistribution to be near optimal in simple instances of Triggers (see Fig. 3-left): attention concentrates quasi exclusively on state-action pairs that enable the collection of future reward. This is confirmed by precision-recall analysis: over the distribution of scenarios considered, a simple binarization heuristic over attention values yields an average precision of 0.96 for an average recall of 0.94. More information on the heuristic is in Appendix A.

In `keys_doors_puzzle`, we adopt the same set of experiments. Since the agent can move backward and spin, in some scenarios it takes a key that is not in is line of sight. In addition, the granularity of the state space is such that off-by-one prediction errors are common but do not hinder the credit mechanism: attributing credit to the state-action couple preceding the collection of a key or the previous one leads to imperceptible changes in the resulting shaped rewards. Fig. 3-right shows similar results as for Triggers. Appendix A also provides a heatmap for this task that shows that attention concentrates around the keys.

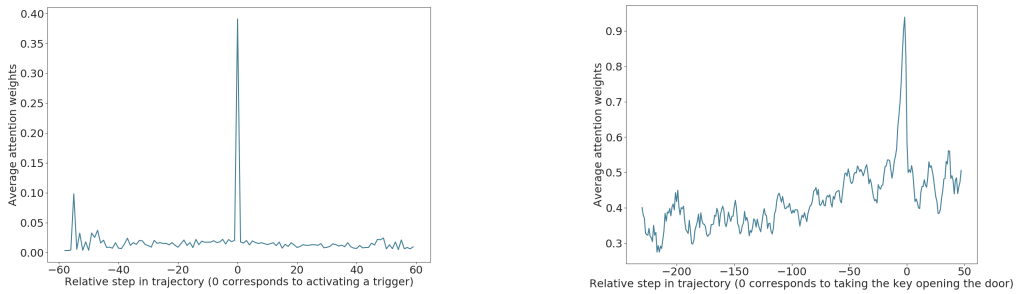

Figure 3: On the **left**, the distribution of attention weights around triggers for correct positive reward predictions in a 8x8 Triggers maze with 3 triggers and one reward. The x-axis denotes the signed number of steps between the state-action couple receiving attention and the actual moment the agent took the key on. On the **right**, the distribution of attention weights around keys for correct reward predictions for door traversals in DMLab.

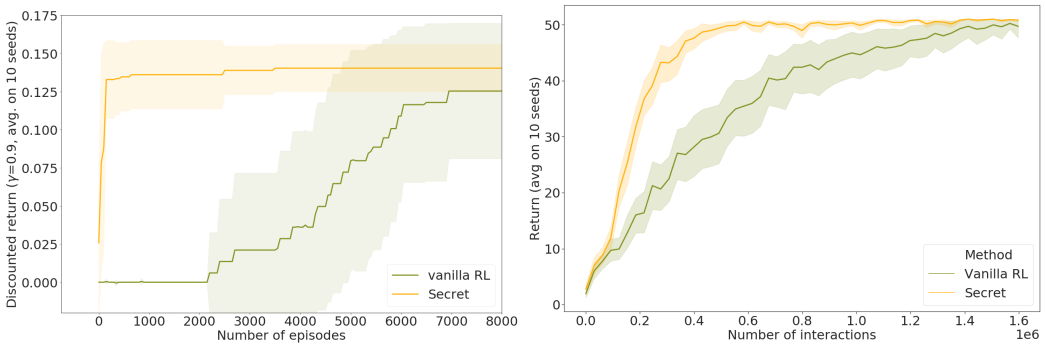

Figure 4: On the **left**, in-domain transfer results on a 8x8 Triggers with 3 triggers and 1 reward. On the **right**, results in DMLab.

3.2 TRANSFER

We then study how we can leverage the inferred credit and transfer representations that are helpful in new scenarios. We show that agents train faster when using shaped rewards from SECRET. As before, the reward model is trained on episodes of interaction in environments sampled from the source distribution. In transfer environments, we sample multiple trajectories, each using the same maze configuration. We then compute the attentional potential function by calculating an estimate of the expected redistributed reward, as described in Sec. 2.2. To evaluate its effect, we compare agents trained from environment rewards to agents that use the resulting shaped reward.

In-domain transfer For in-domain transfer, we transfer the representations for credit assignment to held-out instances of the same distribution over MDPs. For the Triggers environment, the RL agents are tabular Q-learners. For the DMLab environment, we use PPO agents (Schulman et al., 2017) and modify the original task: we do not reward the agent for collecting keys but only to open doors so that the attention can focus on the key positions. Note that this makes the task harder.

As we display in Fig. 4 agents learn visibly faster to solve tasks when benefitting from SECRET in both environments.

Out-of-domain transfer For out-of-domain transfer we use the Triggers environment and consider two scenarios that are hard for standard agents: transfer to bigger environments (see Fig. 5) and transfer to environments with inverted dynamics (see Fig. 6). In the bigger setting, direct weight transfer cannot be used since the visual input has bigger spatial dimensions. On the other hand, SECRET can be used since the transformation we apply to turn states into observations conserves the visual input dimensions. In the inverted dynamics setting, the effect of the agent's actions are inverted, which makes the task hard for transfer methods. In that setting, we compare the transferability

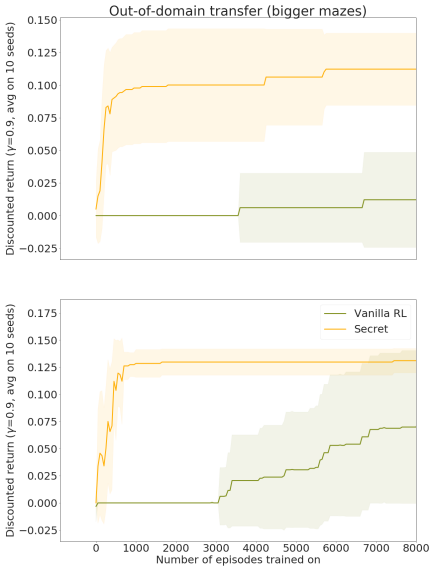

Figure 5: Bigger environments we consider are bigger mazes where the structure of the original task is conserved (number of triggers, number of prizes). Environments drawn are 12x12 grids with 1 trigger and 1 prize for the top figure versus 2 prizes for the bottom one. Environments from the training distribution are 8x8 grids.

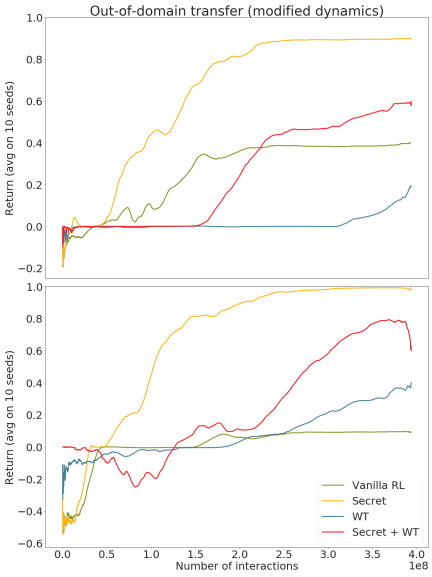

Figure 6: The controls of the out-of-domain distribution are inverted (up becomes down, right becomes left). Environments are 8x8 grids with 1 trigger and 1 prize (top figure) or 2 prizes (bottom figure). The effect of the shaping is exclusively beneficial, while transferring weights from the source task can be detrimental to the learning process.

of our mechanism to that of the representations learned by an agent equipped with deep function approximation. To this end we use DQN agents and either train them from scratch in the target environments or start from the set of weights learned in the source environments.

In both settings, shaping the rewards assists the agent in learning to solve the task. We display some results in Fig. 5 and Fig. 6. When transferring to bigger environments, the agent benefits very early on from the shaped reward, while also reaching better asymptotical performance.

4 Related work

Transfer in RL While a lot of approaches exist in the transfer literature, to the best of our knowledge none explicitly transfer credit assignment capabilities. Previous works aimed at making the training of an agent in the same task more sample-efficient by using a pretrained model as a teacher (Rusu et al., 2016a; Schmitt et al., 2018). We learn to assign credit as a parallel task that does not modify the representations of the RL agent. Others learn auxiliary reward functions in the hope that they will enable transfer by imposing consistency in the reward (Houthooft et al., 2018; Hessel et al., 2018; Agarwal et al., 2019). Although we also learn additional reward signal, it is based on a redistribution of rewards from the environment, which ensures consistency with the original reward function. Transfer is also viewed as learning tasks in a sequential way (Rusu et al., 2016b; Kirkpatrick et al., 2017) and this suggests to introduce inductive bias to the neural architectures of agents to reduce catastrophic forgetting. Our method does not require to alter the agent's architecture. Other explicitly address the problem of transfer through the lens of multitask learning (Parisotto et al., 2016; Teh et al., 2017) while we stick to learning from an initial distribution of environments. Meta-learning approaches aim to train agents on a distribution of tasks or environments so that their learned skills and representations work across the underlying continuum, and allow for fast adaptation of the agents (Duan et al., 2016; Wang et al., 2016; Finn et al., 2017; Mishra et al., 2018; Co-Reyes et al., 2019; Zou et al., 2019). In contrast to meta-learning methods, we do not modify the RL algorithm used to train the agent and Secret is compatible with any core algorithm for RL.

Credit assignment Previous works investigated the role of attention mechanisms for credit assignment. Ke et al. (2018) propose SAB, a sparse attention mechanism used to derive a modified backpropagation algorithm. We draw inspiration from SAB but operate in the RL context without sparsity assumptions about the attention weights. RUDDER (Arjona-Medina et al., 2019) proposes to equip RL agents with an online method for credit assignment based on return decomposition. Our method operates offline and decomposes individual rewards. RUDDER requires a specific exploration scheme, an additional episodic replay buffer, a compute-heavy contribution analysis method and the addition of several auxiliary losses to the objective the RL agent optimizes. In comparison, SECRET is a lightweight method that does not deal with exploration. Crucially, the focus of Arjona-Medina et al. (2019) is on online credit assignment while ours is on transfer. Hung et al. (2018) provide an agent with an external memory and the unsupervised task of reconstructing its inputs (both states and rewards). The agent uses memory reads as a way to identify related elements in sequences, and uses those to transfer the value of states providing delayed rewards to the bootstrapping target of contributing elements. In contrast, SECRET makes use of a non-autoregressive architecture, does not reconstruct states, makes use of reward shaping instead of modifying the update function and most importantly does not rely on an external memory. Recall Traces (Goyal et al., 2018) use a generative model that goes backward from high-reward states and samples state-action pairs that could have led to that state. SECRET also works backward from high-reward states but creates links to previous states from existing trajectories instead of sampling them.

5 CONCLUSION

In this work, we investigated the role credit assignment could play in transfer learning and came up with SECRET, a novel transfer learning method that takes advantage of the relational properties of self-attention and transfers credit assignment instead of policy weights. We showed that SECRET led to improved sample efficiency in generalization and transfer scenarios in non-trivial gridworlds and a more complex 3D navigational task. To the best of our knowledge, this is the first line of work in the exciting direction of credit assignment for transfer. We think it would be worth exploring how SECRET could be incorporated into online reinforcement learning methods and leave this for future work.

REFERENCES

Agarwal, R., Liang, C., Schuurmans, D., and Norouzi, M. (2019). Learning to generalize from sparse and underspecified rewards. *Proceedings of the International Conference on Machine Learning (ICML 2019)*.

Arjona-Medina, J. A., Gillhofer, M., Widrich, M., Unterthiner, T., and Hochreiter, S. (2019). Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems (2019)*.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR 2015)*.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Co-Reyes, J. D., Gupta, A., Sanjeev, S., Altieri, N., DeNero, J., Abbeel, P., and Levine, S. (2019). Meta-learning language-guided policy learning. In *Proceedings of the International Conference on Learning Representations (ICLR 2019)*.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2018). Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters (2018)*, 3(3):1394–1401.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the Annual Meeting of North American chapter of ACL (NAACL 2019)*.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). Rl^2: Fast reinforcement learning via slow reinforcement learning. *Proceedings of the International Conference on Learning Representations (ICLR 2017)*, abs/1611.02779.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*.

Goertzel, B., Iklé, M., and Wigmore, J. (2012). The architecture of human-like general intelligence. In *Theoretical Foundations of Artificial General Intelligence*, pages 123–144. Springer.

Goyal, A., Brakel, P., Fedus, W., Lillicrap, T., Levine, S., Larochelle, H., and Bengio, Y. (2018). Recall traces: Backtracking models for efficient reinforcement learning. *Proceedings of the International Conference on Learning Representations (ICLR 2019)*.

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. (2018). Multi-task deep reinforcement learning with popart. *arXiv preprint arXiv:1809.04474*.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Ho, O. J., and Abbeel, P. (2018). Evolved policy gradients. In *Advances in Neural Information Processing Systems (2019)*, pages 5400–5409.

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification. In *ACL 2018*. Association for Computational Linguistics.

Hung, C., Lillicrap, T. P., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. (2018). Optimizing agent behavior over long time scales by transporting value. *CoRR*, abs/1810.06721.

Ke, N. R., Goyal, A., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. (2018). Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in Neural Information Processing Systems (2018)*, pages 7650–7661.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (2017)*, 114(13):3521–3526.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Lin, Z., Feng, M., dos Santos, C. N., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. In *Proceedings of the International Conference on Learning Representations (ICLR 2017)*.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. In *Proceedings of the International Conference on Learning Representations (ICLR 2018)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the International Conference on Machine Learning (ICML 1999)*, volume 99, pages 278–287.

Parisotto, E., Ba, J., and Salakhutdinov, R. (2016). Actor-mimic: Deep multitask and transfer reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR 2016)*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the Annual Meeting of North American chapter of ACL (NAACL 2018)*.

Puterman, M. L. (1994). *Markov Decision Processes*. Wiley.

Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2016a). Policy distillation. In *Proceedings of the International Conference on Learning Representations (ICLR 2016)*.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016b). Progressive neural networks. *CoRR*, abs/1606.04671.

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. (2018). Episodic curiosity through reachability. *Proceedings of the International Conference on Learning Representations (ICLR 2019)*.

Schmitt, S., Hudson, J. J., Zídek, A., Osindero, S., Doersch, C., Czarnecki, W. M., Leibo, J. Z., Küttler, H., Zisserman, A., Simonyan, K., and Eslami, S. M. A. (2018). Kickstarting deep reinforcement learning. *CoRR*, abs/1803.03835.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR 2016)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems (2014)*, pages 3104–3112.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems (2017)*, pages 4496–4506.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (2017)*, pages 5998–6008.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *CoRR*, abs/1611.05763.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Zoph, B., Yuret, D., May, J., and Knight, K. (2016). Transfer learning for low-resource neural machine translation. *ACL 2016*.

Zou, H., Ren, T., Yan, D., Su, H., and Zhu, J. (2019). Reward shaping via meta-learning. *arXiv preprint arXiv:1901.09330*.

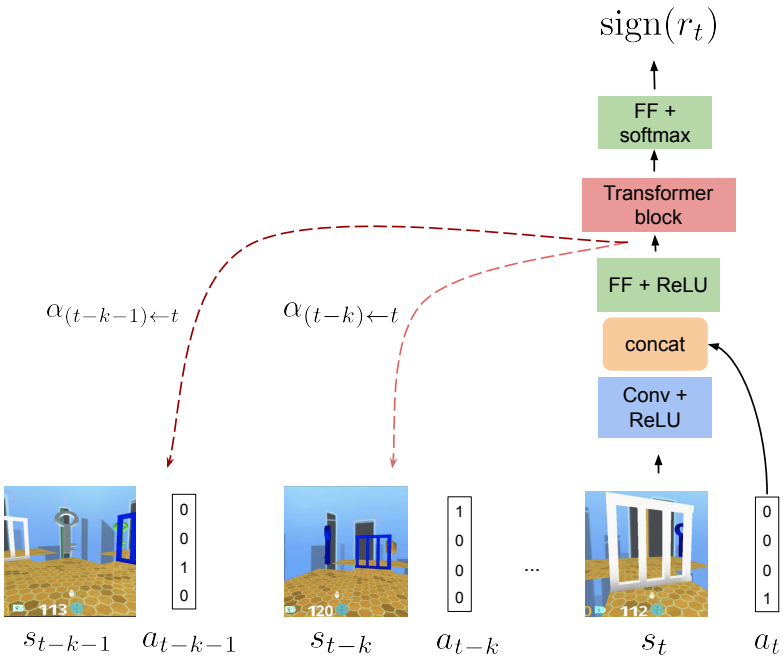

Figure 7: The architecture used for SECRET. $\alpha_{\cdot\leftarrow t}$ is the vector containing the attention weights of the model for its prediction at step t.

A ADDITIONAL EXPERIMENT DETAILS

In this section we provide additional details about our experimental setup and the hyperparameters we use.

A.1 REWARD PREDICTION MODEL

We use the same set of hyperparameters in all our experiments with few variation. In Triggers experiments, we use 128 units per dense layer, 32 convolutional filters and a single convolutional layer to process partial states and actions. We apply dropout (Srivastava et al., 2014) at several places in the model as a regularizer. We use a dropout rate of 0.1 after dense layers, a dropout rate of 0.2 in the self-attention mechanism, and a dropout rate of 0.2 in the normalization blocks of the Transformer architecture. We use the same positional encoding scheme as in Vaswani et al. (2017). In DMLab experiments, we use two convolutional layers, 16 filters for each, and otherwise identical hyperparameters. Typically, we set the class weights in the loss function to $w(1) = w(-1) = 0.499$, $w(0) = 0.02$. Fig. 7 gives an overview of the whole architecture.

A.2 HEURISTIC FOR PRECISION-RECALL ANALYSIS

In Sec. 3.1, we compare the attention vectors we get as outputs from the reward prediction model to ideal credit assignment with binary metrics. The ground truth we use is a binary vector of the size of the attention vector. Its values are 0 everywhere and 1 for timesteps that correspond to the activation of a Triggers. To do so, we introduce a simple heuristic to binarize the attention scalars: we consider all values above a threshold α to correspond to events to be credited. Then, we can measure precision and recall as in a binary classification paradigm. The precision and recall reported are the average precision and recall over 4 scenarios : Triggers with a 8x8 grid, 1 trigger and 1 reward; Triggers with a 8x8 grid, 1 trigger and 2 rewards; Triggers with a 8x8 grid, 2 triggers and 2 rewards; and Triggers with a 8x8 grid, 3 triggers and 1 reward. In each scenario, we train the model over a set of 40000 trajectories, each of which is drawn from a randomly sampled maze. Then, we apply it

on 5000 trajectories from held-out environments and collect the attention weights corresponding to predictions on timesteps where the agent experiences positive reward. We use a fixed α of 0.2.

A.3 TRANSFER IN TRIGGERS

We collect 40000 trajectories sampled from random policies to train the prediction reward model in all distributions of environments.

Experiments involving tabular Q-learning we use online Q-learning with a learning rate of 0.1 and a constant greediness factor ϵ also equal to 0.1.

For the out-of-domain transfer experiment with modified dynamics, we use a smaller version of the DQN architecture in Mnih et al. (2015). The first convolutional layer has 8 filters, a 3x3 kernel size and a stride of 2. The second and the third convolutional layers have both 16 filters, a 3x3 kernel size and a stride of 1. Those are completed by a feed-forward layer with 64 units followed by another feedforward layer with as many units as the number of available actions. The greediness factor ϵ is decayed linearly from 1 to 0.01 over 250000 steps in the environment at train time and has a constant value of 0.001 at test time. We use RMSProp (Tieleman and Hinton, 2012) as an optimizer with a base learning rate of 0.00025. We update the target network every 2000 steps and initially fill the replay buffer with 5000 transitions sampled following a random policy. The replay buffer has a maximum size of 1000000.

A.4 IN-DOMAIN TRANSFER IN DMLAB

We provide additional details about this setup: we train SECRET using 10000 trajectories sampled from a distribution of mazes that are generated randomly. These trajectories are sampled using an agent trained over the same distribution. We do so to increase the proportion of trajectories where rewards are experienced. Indeed, we found that using random policies yielded very few of these. Once the model is trained, we use it to compute the attentional potential function over a fixed maze. 1000 trajectories are sampled on the fixed maze using the same policy as the one that generated the trajectories used to train the reward prediction model. Since consecutive frames can be very similar, we consider a positive reward prediction to be correct (and thus use the corresponding attention weights when estimating the potential) if it happens within 5 frames of a reward actually experienced in the environment. We then compare the performance of agents trained with the original reward function to those trained with the shaped reward.

An important point is that we use the knowledge of the position and the keys the agent possesses to compute and then exploit the potential function. This information is not given to the agent. We acknowledge that relying on the knowledge of the state the agent is in limits the generality of our approach, but we are confident that this limitation can be addressed in future work by using an estimate of the true state.

All agents mentioned in that section are PPO learners with a learning rate of 0.00019, an entropy coefficient of 0.0011, 12 actors, a discount factor $\gamma = 0.99$. They use generalized advantage estimation (Schulman et al., 2016) with $\lambda = 0.95$. All those hyperparameter values are taken as is from the experiment section in Savinov et al. (2018), whose code is open-source.

A.5 ATTENTION HEATMAP IN DMLAB

In Fig. 8, we display the per-position average attention weight along trajectories starting in the middle-left room that led to opening the blue doors. Attention (shown in shades of red) concentrates around the key position in the lower-left corner.

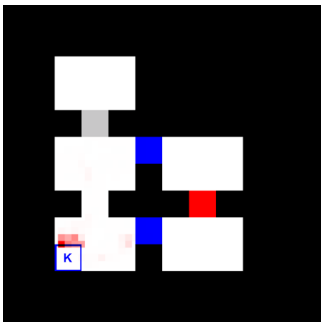

Figure 8: Attention heatmap in DMLab. Attention concentrates around the key which is necessary to unlock reward.