General policy mapping: online continual reinforcement learning inspired on the insect brain

Anonymous Author(s)
Affiliation
Address
email

Abstract

We have developed a model for online continual reinforcement learning (RL) inspired on the insect brain. Our model leverages the offline training of a feature extraction and a common general policy layer to enable the convergence of RL algorithms in online settings. Sharing a common policy layer across tasks leads to positive backward transfer, where the agent continuously improved in older tasks sharing the same underlying general policy. Biologically inspired restrictions to the agent's network are key for the convergence of RL algorithms. This provides a pathway towards efficient online RL in resource-constrained scenarios.

1 Introduction

Insects exhibit a remarkable ability to learn new tasks. In addition to their well-documented abilities in the wild, researchers have taught insects such as honeybees and bumblebees a wide range of tasks, from learning to manipulate objects to access food, to navigating a maze using visual cues, to "playing soccer".(1; 6) These capabilities are particularly relevant in the context of online reinforcement learning (RL) and, in particular, in situations where a single agent must learn from the interaction with its environment in as few examples as possible. Convergence to optimal policies under these assumptions is extremely challenging within the framework of Markov decision processes.(8) Therefore, the question is: How can insects accomplish this?

In this work, we have formulated a model, which we refer to as *general policy mapping*, that is consistent with the current understanding of insect neuroscience. At the core of this approach is an effective partitioning of the problem into offline and online phases that greatly simplifies the online learning of new tasks. In particular, the policy network is partitioned into three different components, each of which has a well-defined counterpart in the insect brain: a feature extraction layer, an online policy mapping layer, and a general policy layer shared across tasks.

We have evaluated this algorithm against a series of 1 in 5 target discrimination tasks implemented in a custom open world first person point of view environment. In these tasks, the agent is shown 5 objects and needs to learn to reach the correct target. Through sampling over multiple seeds, we have demonstrated the convergence of a simple cross entropy algorithm when the agent is only allowed to interact with a single environment under sparse reward conditions. The same algorithm fail to converge when presented with the general task in the absence of offline contributions. Sharing a common general policy across tasks also lead to positive backward transfer in continual RL scenarios.

2 Model

In hymenopterans, such as bees and ants, visual inputs are processed in a way that is reminiscent of the visual processing system in mammals. This internal representation, which is encoded in

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

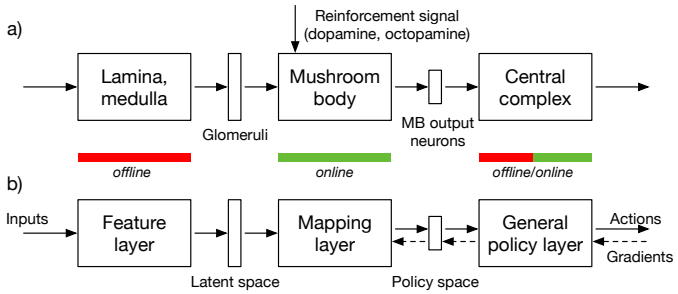

Figure 1: a) High level diagram of online learning pathways in hymenopterans (i.e. bees and ants) b) General policy mapping model explored in this work

the optical glomeruli, is then passed into the mushroom body, which is the learning center of the insect brain (Fig. 1). The mushroom body (MB) plays a critical role in online learning, with ablation experiments showing that the absence of a functional MB prevents insects from learning new tasks while preserving their ability to carry out innate tasks.(2; 5) The MB aggressively reduces the amount of data being processed, transforming large-dimensional inputs into a low-dimensional space. Moreover, despite its central importance, the number of connections coming out of the MB is relatively small. The consensus is that the MB does not encode policies, but instead encodes saliency and valence in a task-dependent fashion. This information is then passed to other centers in the insect brain, where it is used to drive insect behavior.(3)

From an RL perspective, this structure is consistent with a model in which the input space $\mathbf{s}$ is subsequently transformed into a latent space $\mathbf{s}_l$ and a policy space $\mathbf{s}_p$, which dictates which actions the insect is taking based on a set of policy drivers:

$$\mathbf{s} \rightarrow \mathbf{s}_l \rightarrow \mathbf{s}_p \rightarrow a \tag{1}$$

The general policy mapping approach introduced in this work implements this type of structure, splitting the policy network into three different components (Fig. 1): a *feature layer*, a *mapping layer*, and a *general policy layer*. Both the feature and general policy layers are pretrained offline, with the mapping layer being responsible for mapping online tasks to the policy layer.

In order to strengthen the roles of these three components, we have introduced two key restrictions that are present in the insect brain: 1) the mapping layer behaves as a saliency layer, with the output being a linear combination of multiple channels in which both inputs and weights are restricted to be positive. This is consistent with the preservation of the type of synapse (in this case excitatory) during the online learning process. As we will show later, this restriction turns out to be critical for the convergence of single environment RL algorithms. 2) The policy layer is restricted so that, in absence of inputs, the output reverts to a default policy, which in our case we chose it to be a random policy. Again, this ensures that the outputs from the mapping layer act as drivers that shift the policy towards specific goals and encourages exploration when facing a new task.

Finally, while the mapping layer is task-specific, the general policy layer is shared among all tasks.[Fig. 2(a)] Enabling plasticity in the general policy layer during the online phase lets the agent refine its general policy as it faces new online tasks. This is markedly different from the multiple head approach used in task-incremental continual learning, where only features are shared across tasks. This feature is critical to achieve positive backward transfer in continual RL scenarios.

3 Experiments

The core hypotheses we want to explore are whether the policy mapping approach enables simple RL algorithms to reliably converge to optimal policies under online RL restrictions, and whether it promotes continual/lifelong learning through the sharing of a common general policy across tasks. To this end we designed a series of 1 in 5 object identification tasks where the agent has to reach a task-specific target, and implemented in RLBug, a custom first person point of view, open world environment developed specifically for this work. Snapshots of five different tasks are shown in Figure 2(b). Inputs are 3 channel 84x84 images, and the agent's action space comprises 5 possible

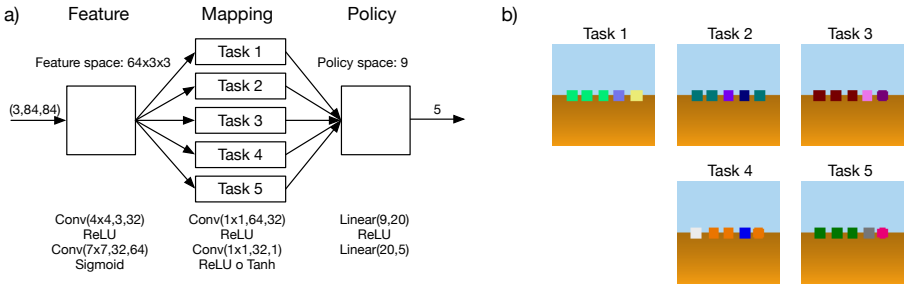

Figure 2: a) Policy matching network structure b) Snapshots of 1 in 5 object identification task implemented in our custom environment RLBug

actions (moving forward, stationary turns, and turning while moving). In order to mimic the type of scenarios that insects would find in the wild, a constant positive reward is offered only if the agent reaches the right object before the end of the episode. The positions and ordering of the different objects are randomly selected for each episode to ensure that the agent learns to identify the right target object rather than going to a specific location. Episodes only terminate prematurely if the agent reaches any of the objects. Otherwise, the agent is allowed to wander freely until reaching a maximum number of steps. This further adds to the difficulty of the task. Details of the environment can be found in the Appendix.

The experiments comprise an offline and an online training phase: the offline phase involves the training of both the feature and the general policy layers. The feature layer is trained using an object classification task comprising 27 different objects of random sizes placed in random positions in front of the agent (see the Appendix for details). The resulting feature extraction layer is transferred directly to the RL agent. The general policy layer is trained using a baseline task where the agent needs to reach a single object, using a 84x84 binary saliency map as input. The head of the trained network is then used as the general policy layer for the online learning phase.

The online phase of the experiment involves training the agent in 5 1 in 5 object identification tasks. We have trained the agent using the cross-entropy algorithm(4) with an additional entropy term.(7; 9) This algorithm is known to struggle when tasks are too complex, but it requires the least amount of prior information on the tasks, which is consistent with online scenarios where tasks are not known in advance. Consequently, the convergence probability is one of the key metrics that we have explored in this work, using 20 randomly selected seeds per task to generate a distribution of learning curves. For the continual learning setting, we trained the agent consecutively on the five tasks in a task incremental setting.

4 Results

4.1 Online reinforcement learning

The results obtained show that the general policy matching model enables a simple RL algorithm such as cross entropy to converge to the optimal policy with high probability. (Table 1). The values shown in the Table have been obtained after averaging a total of 100 runs (20 per each of the tasks). Ablation studies show that the offline training of the feature layer and the clamping of the weights in the mapping layer are key to achieve high levels of convergence.

Offline RL training to a general policy using a surrogate task significantly decreases the number of episodes required for learning with respect to the case in which the policy layer is not pretrained, reducing the average number of episodes required at least by a factor of three. As a reference, a deep Q learning algorithm trained on a similar network requires an average of 1200 episodes.

Finally, we also examined the case with no offline training in ether the feature or the general policy component. In this scenario, the convergence achieved with the cross entropy algorithm is really poor.

Table 1: Convergence of cross entropy algorithm for the full general policy matching model as well as selected ablated configurations for the 1 in 5 object selection task. Each data point is the average of 100 independent runs across five different tasks shown in Figure 2(b)

	Cross entropy algorithm	
Configuration	Convergence	Episodes, av(std)
Fixed general policy	100%	530 (240)
Adaptive general policy	**99%**	**280 (80)**
No offline policy	94%	820 (320)
No offline feature	30%	740 (340)
No offline components	10%	1200 (420)
Unclamped weights	70%	230(130)

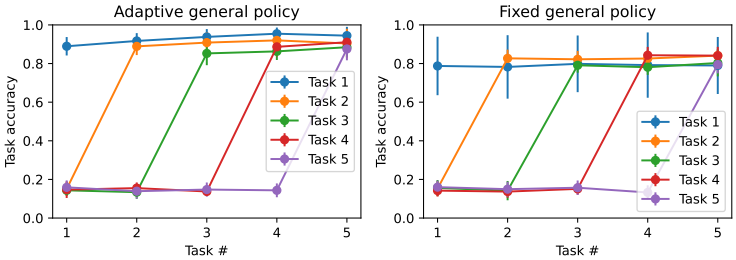

Figure 3: Task accuracy in a task-incremental continual RL test involving Tasks 1 to 5. Results are shown for two agents: policy matching agent with an adaptive general policy and with a fixed (offline) general policy. Results shown are the average of 20 separate runs

4.2 Continual learning

We have then taken the two top performing configurations, the policy matching model with and without online adaptive policy, and applied them to a task-incremental continual learning task where a single agent is exposed to the five 1-in-5 tasks consecutively. In between each task we evaluated the agent's performance on each of the tasks to track the impact that learning newer tasks had on the agent's ability to learn older ones. We used the same cross-entropy algorithm to train the agent in each of the tasks without any additional strategies involving either synaptic consolidation or replay to preserve information of prior tasks.

The results, shown in Figure 3, show that even when we share the general policy layers among the tasks, the agent is capable of sequentially learning multiple tasks without any evidence of catastrophic forgetting. In fact, the performance on earlier tasks seem to increase as it is trained on newer tasks [Figure 3(a)], which is an evidence of positive backwards transfer. In absence of online learning in the general policy layer, the agent's performance in older tasks remains stationary as it is trained on newer tasks [Figure 3(b)]. The performance using a baseline deep-Q learning algorithm show a significant amount of catastrophic forgetting, as shown in the Appendix. In this case, a buffer was restricted to each individual task to eliminate the agent's ability to train on data from older tasks.

5 Conclusions

In this work we we explored a model for online continual RL inspired on the insect brain. The results obtained show that this type of architecture promotes convergence of simpler algorithms that are well suited for online RL scenarios where no prior information of the specific tasks is available. The best performing configurations also excel in continual learning tasks, in the adaptive general policy actually improving its performance on older tasks as the agent refines its underlying general policy. Here we have focused on a set of tasks that resemble the type of learning that insects would be required to do as it explores an open world environment. In addition to shedding some light on how biological constraints can promote online RL, this model can be valuable to explore novel architectures for online, on chip learning under resource constrained scenarios.

References

[1] Sylvain Alem, Clint J. Perry, Xingfu Zhu, Olli J. Loukola, Thomas Ingraham, Eirik Søvik, and Lars Chittka. Associative mechanisms allow for social learning and cultural transmission of string pulling in an insect. *PLOS Biology*, 14(10):1–28, 10 2016.

[2] Yoshinori Aso, Daisuke Hattori, Yang Yu, Rebecca M Johnston, Nirmala A Iyer, Teri-TB Ngo, Heather Dionne, LF Abbott, Richard Axel, Hiromu Tanimoto, and Gerald M Rubin. The neuronal architecture of the mushroom body provides a logic for associative learning. *eLife*, 3:e04577, dec 2014.

[3] Matthew Collett and Thomas S. Collett. How does the insect central complex use mushroom body output for steering? *Current Biology*, 28(13):R733–R734, 2018.

[4] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.

[5] Jean-Marc Devaud, Thomas Papouin, Julie Carcaud, Jean-Christophe Sandoz, Bernd Grünewald, and Martin Giurfa. Neural substrate for higher-order learning in an insect: Mushroom bodies are necessary for configural discriminations. *Proceedings of the National Academy of Sciences*, 112(43):E5854–E5862, 2015.

[6] Olli J. Loukola, Cwyn Solvi, Louie Coscos, and Lars Chittka. Bumblebees show cognitive flexibility by improving on an observed complex behavior. *Science*, 355(6327):833–836, 2017.

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[9] Ronald J. Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

A Appendix

A.1 RLBug environment

RLBug is a RL environment that implements the interaction of an agent from a first person point of view perspective in an open-world environment. RLBug is implemented as a pure Python environment and it is extremely lightweight, capable to achieve more than 500 frames per second in a single core for an scenario comprising 20 objects. Our goal was to find a balance between the need to develop a visually rich set of tasks while avoiding the slower frame rates of near real time environments based on sophisticated engines, which are harder to implement in high performance computing settings.

Tasks are generated by passing a configuration of objects in which each type of objects is defined by color, height, width, and the reward of the object. The color of the ground, sky, objects, and other effects such as depth-fading, are all customizable, as is the type of output (either 1 channel or 3 channel images). For this work, we considered a state comprising an RGB 3-channel 84x84 pixel image encoded as a (3,84,84) array, and an action space with five actions: moving forward, turning left and right, and turning while moving. RLBug will be made publicly available by the time of the workshop.

In this work we used RLBug to implement a 1 in 5 object identification task. In this task the agent is presented five objects, one target object, one decoy object, and three instances of background objects. The agent must learn to move and reach the target object within a maximum number of steps. In order to prevent the agent from learning a deterministic policy, the objects are placed at distance from the agent that is randomly determined for each episode around a mean value. The ordering of the objects as seen by the agent are also randomized for each episode. A positive reward is provided only when the agent reaches the target within the predetermined number of steps.

A key difficulty of this task is that the agent is free to roam in the 2D environment until the maximum number of steps is reached. This means that, if the agent loses track of the objects, the agent is seeing an empty landscape as input. Baseline studies using deep Q learning and various policy gradient methods show that this reduces the algorithms ability to converge with respect to the case where the episode ends once none of the 5 objects is within the agent's visual field. However, we feel that this type of open world interaction reproduces the type of interactions that insect have with their environment and is also representative of many potential applications where an agent has to learn to carry out tasks in the world without any external cues.

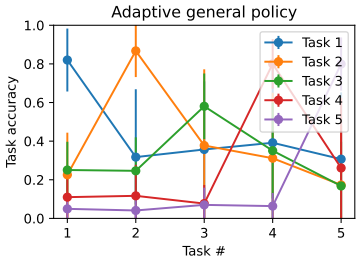

Figure 4: Task accuracy during the continual learning of Tasks 1-5 using deep Q learning. The results are consistent with a significant amount of catastrophic forgetting.

A.2 Offline training of the feature layer

Insects do not learn everything from scratch, but instead they come with a well-defined architecture for processing visual inputs. In insect this is carried out primarily in the lamina and medulla, which have evolved to maximize the insect's ability to learn tasks that are not known beforehand across multiple domains.

The problem of how to identify an optimal set of features that maximizes an agent's ability to quickly learn new RL tasks after deployment is currently one of our active areas of research. For this work, though, we focused on a single architecture that is inspired on some of the design principles in the lamina and medulla of insects that we have trained on a single downstream task. A key characteristic is that the output layer comprises several non-overlapping large field areas with many channels each.

For this work, we considered a feature extraction network comprising two convolutions, one 4x4 layer producing 32 channels, and one 7x7 layer outputting 64 channels. A rectified linear unit and a sigmoid function were used for the first and second convolutional layers (Figure 2). In order to train the feature extraction layer, we added an all-to-all linear layer as output layer. We trained this network in an object classification task comprising 27 different types of objects. We generated a dataset of images in which we placed an object of a random size and width on a random location within the agent's visual field. We then partitioned the RGB space into equally sized regions and generated objects using randomly selected colors within each of these regions. The resulting dataset comprises 500 images per category with 100 additional images reserved for testing. We trained the network against the training dataset, achieving 98% accuracy in the testing dataset after 20 epochs. The feature extraction component was then used in the RL tasks.

A.3 Deep Q learning baseline results

As a baseline we used a deep Q learning algorithm to sequentially train the agent on Tasks 1 to 5. In order to prevent cross-talks between the tasks we started each Task with an empty buffer. For this baseline case we used the pretrained feature layer and we maintained the same matching and general policy layers used for the cross-entropy method. The key difference is that now the network is used to learn the state-action values $Q(s, a)$, and we applied an ε-greedy approach to compute our policy. The results, shown in Figure 4, are consistent with the presence of catastrophic forgetting, with the agent increasingly forgetting the learned tasks. A cut-off number of 2000 episodes is used for each tasks, with the learning process ending earlier if the agent's performance exceeds 95% in the last 25 episodes under a value of $\varepsilon = 0.02$. The higher error bars are due to the variance in the learning process.