Inducing Functions through Reinforcement Learning without Task Specification

Anonymous Author(s)
Affiliation
Address
email

Abstract

We report a bio-inspired approach for training a neural network through reinforcement learning to induce high level functions within the network. Based on the interpretation that animals have gained their cognitive functions such as object recognition — without ever being specifically trained for — as a result of maximizing their fitness to the environment, we place our agent in a custom environment where developing certain functions may facilitate decision making; the custom environment is designed as a partially observable Markov decision process in which an input image and the initial value of hidden variables are given to the agent at each time step. We show that our agent, which consists of a convolutional neural network, a recurrent neural network, and a multilayer perceptron, learns to classify the input image and to predict the hidden variables. The experimental results show that high level functions, such as image classification and hidden variable estimation, can be naturally and simultaneously induced without any pre-training or specifying them.

1 Introduction

Advances in reinforcement learning has not only facilitated the development of artificial intelligence for solving challenging problems Ha & Schmidhuber (2018); Badia et al. (2020a); Kapturowski et al. (2019); Badia et al. (2020b); Mnih et al. (2015); Vinyals et al. (2019), but it has also bridged the gap between learning in biological systems and learning in artificial systems Neftci & Averbeck (2019); Dabney et al. (2020). For example, temporal difference learning Sutton (1991), —which was established in the machine learning field, — led to the reward prediction error theory of dopamine, which suggests that the phasic activity of dopaminergic neurons encodes the difference between the predicted rewards and the experienced rewards Schultz et al. (1997). More recently, distributional reinforcement learning Bellemare et al. (2017) led to another interesting hypothesis that suggests that a brain simultaneously predicts future rewards as a probability distribution over possible outcomes; that hypothesis was later supported by animal experiments Dabney et al. (2020). The researchers in Merel et al. (2020) implemented a virtual rodent that had access to vision and proprioceptive information, and they used neuroethological modeling to characterize motor activity of the rodent. Such research provides a unique opportunity to access the data that is not available from biological systems, which often leads to valid hypotheses on biological intelligence Schultz et al. (1997) which in turn contributes to artificial intelligence Dabney et al. (2020); Banino et al. (2018).

From the point of view of evolution theory, biological intelligence is just one of the many features that animals have gained through evolution as it provides advantages to the species' survival and the spread of their genes. In other words, animals and their biological intelligence optimize themselves for survival. For example, animals have attained vision and object recognition capability through evolution as such capabilities have provided a huge survival advantage, but not because they were

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

explicitly trained for such tasks. Similarly, animals can understand that objects continue to exist even with the lack of current sensory clues Miller et al. (2009), and they can predict hidden variables such as the trajectory of an invisible moving target Barborica & Ferrera (2003) — both of which can facilitate their decision making.

Based on this observation, we conjecture that, through reinforcement learning in an environment where image classification and hidden variable estimation are helpful, such functions will be naturally induced within the network without them ever being specified in the training procedure. We show that this is indeed possible by training a neural network in a custom environment.

Specifically, we designed a custom environment that models an animal's survival task in nature as a partially observable Markov decision process. In the environment, the agent encounters discrete events (e.g., a predator or a prey) and receives a corresponding vision input (i.e., an image). In addition, there are variables in the environment that the agent does not have access to. We find that, through end-to-end reinforcement learning of an agent with convolutional and recurrent sub-networks, image classification and hidden variable estimation capabilities can be induced within each sub-network. The two capabilities are defined as follows:

- **Image classification**: to map the input images to the feature vectors that are linearly separable by their classes.
- **Hidden variable estimation**: to yield output values that are linearly proportional to the hidden variables.

This suggests that high level functions can be induced in an artificial intelligence system through reinforcement learning without task specification, which opens up the possibility of implementing networks with various cognitive functions.

2 Related work

2.1 Inducing functions within networks

Training a neural network can be considered as a procedure of learning a composite function $f(x) = f_n \circ f_{n-1} \circ \cdots \circ f_1(x)$ to solve a task that is specified through a loss function. The function learned by a part of the network (i.e., $f_k(x)$) is determined jointly by the loss function, constraints on the function space of each $f_j, j \in \{1, 2, \cdots n\}$ and constraints on the vector space that each f_j projects onto. In Krizhevsky et al. (2012), the weights of the first convolution layer converged to certain shapes which corresponded to image processing functions (e.g., low-pass or high pass filtering). In Kingma et al. (2014); Chen et al. (2016), the authors demonstrated that certain parts of the network can learn to disentangle different types of information (e.g., style and content) in the input with guidance through the loss function. In Makhzani et al. (2016), an adversarial autoencoder was trained in a semi-supervised manner in which MNIST Deng (2012) images were projected to a latent space where the images were clustered by their corresponding digits. However, inducing high level functions without task specification in a reinforcement learning setting has not been demonstrated.

2.2 Bio-inspired reinforcement learning

Owing to the deep-rooted connection between the reinforcement learning in artificial and biological agents, there has been a number of previous works on bio-inspired reinforcement learning Neftci & Averbeck (2019). In Singh et al. (2005, 2010), an intrinsic reward for reinforcement learning was proposed to accelerate learning by setting the agent to receive internal rewards from a critic which was part of the agent. Niekum et al. (2010, 2011) also employed the notion of intrinsic reward and used a genetic algorithm to search for alternate reward functions. Thereafter, neuro-inspired intrinsic reward construction in reinforcement learning have been extensively studied Gershman (2019); Lin et al. (2019); Rasmussen et al. (2017); Lehnert et al. (2020). Bellemare et al. (2017) designed an agent that represented the future rewards as a probability distribution over multiple possible outcomes. In Pontes-Filho & Nichele (2019), evolution was modeled as the inheritance and change of network topology of spiking neural networks. Abrantes et al. (2020) proposed a learning method that mimics evolution in which the reward function was slowly aligned with the fitness function. The idea that penetrates these works is that an advanced form of artificial intelligence may be developed by mimicking certain aspects of biological intelligence. Our work shares that spirit,

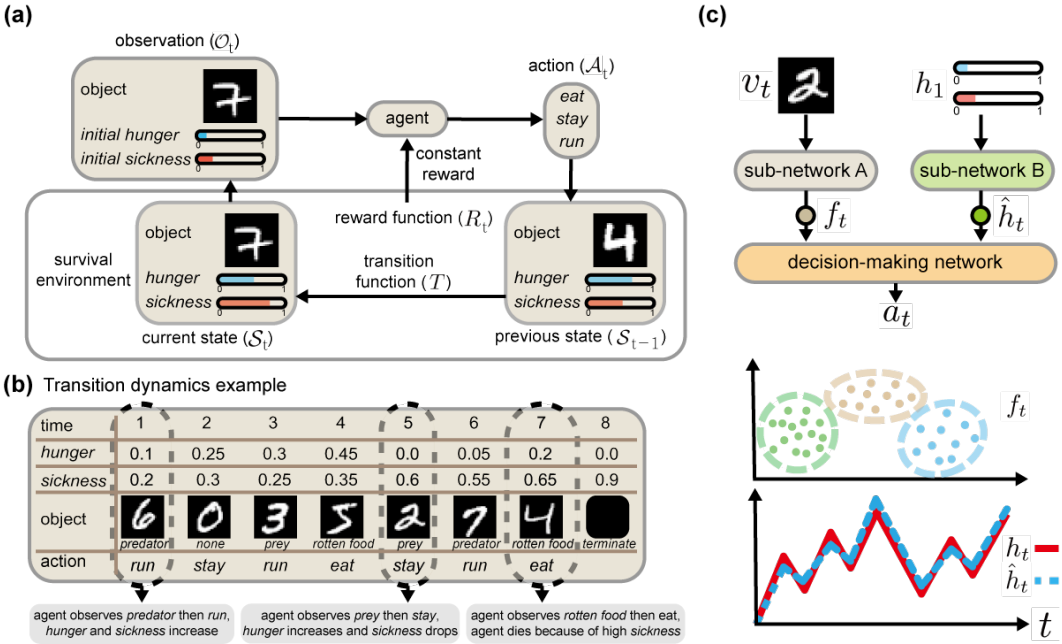

Figure 1: Overview of the survival environment. (a) An animal's survival task in nature is modeled as a partially observable Markov decision process. At each time point, the agent encounters an object (*none*, *predator*, *prey* or *rotten food*) represented as an image. In addition, there are two hidden variables (*hunger* and *sickness*) that are not observed by the agent. The agent should choose an action (*run*, *stay* or *eat*) to maximize its life span. The image and the initial value of the hidden variables are given to the agent. (b) A transition dynamics example. Each action has an opportunity cost and hence, it is not possible to minimize the risks from all sources at the same time. (c) The agent receives the image and the initial value of the hidden variables (left). The goal is to induce image classification and hidden variable estimation capabilities within the network without any pre-training or specification (right).

but it differs from the previous works in that our goal is to induce high level functions by placing an agent in an environment where certain high level functions can help decision making rather than directly mimicking the evolutionary procedure for training networks.

2.3 Reinforcement learning with recurrent networks

Deep recurrent Q-learning Hausknecht & Stone (2015) and a deep recurrent policy gradient Heess et al. (2015), which employed an agent with convolutional layers and an LSTM, demonstrated their capability to integrate the information across multiple frames. Since those studies, recurrent units have been widely employed for reinforcement learning. For instance, Kapturowski et al. (2019) further improved this method for memory-critical problems by devising two strategies for initialization of the state of recurrent units, rather than naive zero initialization Hausknecht & Stone (2015). Ha & Schmidhuber (2018) proposed a model-based reinforcement learning method, called the world model, that used a recurrent unit which was motivated by the fact that a human uses a mental model of the world to predict the future. The recurrent unit was separately trained in a supervised manner to predict the next output from the convolution layers which was then fed to the following fully connected layer for choosing the action. Simulated policy learning Kaiser et al. (2020) demonstrated excellent performance in a low data regime by alternatively updating the world model and the policy. In these methods, the role of the recurrent units was to integrate the information from the vision input across time as opposed to predicting the information outside the observation.

3 Survival environment

3.1 Formal description of the environment

We built a custom environment in which an agent makes a sequence of decisions to maximize the length of its life span. The environment is modeled as a partially observable Markov decision process with a tuple $(\mathcal{S}, \mathcal{A}, T, R, \Omega, \mathcal{O})$ (Fig. 1(a)), where the state space $\mathcal{S}$ is constructed as a tuple of a vision input, denoted by v_t, (i.e., image at time t) and additional state variables, denoted by h_t, i.e., the state at time t is $(v_t, h_t) := s_t \in \mathcal{S}$. The image corresponds to the object class that the agent encounters, which is one of *none*, *predator*, *prey* and *rotten food*. For each class, we allocated two random digits between 0 and 9 (Appendix A) and randomly picked 1,000 images for each number from MNIST dataset. The state variables are *hunger* H_t and *sickness* S_t (i.e., $h_t = (H_t, S_t) \in [0, 1]^2$), which depend on their previous values, vision input and action (i.e., $h_t = g(h_{t-1}, v_{t-1}, a_{t-1})$, where g is the function that determines state variable transition). The initial values of the state variables are randomly set at the beginning of each episode. At each time point, the agent chooses one of the following actions: *stay*, *run*, and *eat*, i.e., $\mathcal{A} = \{\textit{stay}, \textit{run}, \textit{eat}\}$. A constant reward $r_t = 1$ is given at each time step t to set the goal of the agent as simply making the episodes as long as possible.

Following the observation function $\mathcal{O}$, only partial information is given to the agent; the vision input is observable by the agent and the state variables $h_t = (H_t, S_t)$ are accessible to the agent only on step 1, after which they are hidden. For this reason, h_t will be called the hidden variables throughout the paper. We note that this is different from most previous reinforcement learning settings with convolutional and recurrent units Ha & Schmidhuber (2018); Hausknecht & Stone (2015); Kapturowski et al. (2019); Badia et al. (2020b) in which the recurrent units are used to predict the future vision inputs or to integrate the information across time. The complete information on the environment including the transition function $\mathcal{T}$ is given in Appendix B.

3.2 Biological interpretation of the environment

Our custom environment can be interpreted as follows. The goal of the agent — which models an animal — is to survive as long as possible, where the possible causes of death are being eaten by a *predator* and a high *hunger* or *sickness* level. Each action is designed to have an opportunity cost (e.g., *running* decreases the chance to be eaten by a *predator* but increases *sickness*, *eating rotten food* decreases *hunger* but increases *sickness*) as it would in nature (Fig. 1(b)), and therefore it is not possible to minimize the risk from all sources at the same time. Consequently, the optimal action at each state depends not only on the object it encounters, but also on the hidden variables (e.g., for a *predator* encounter, choosing *stay* over *run* may be better when *sickness* is high). Importantly, the object class information is indirectly given to the agent as an image and the hidden variables are not accessible to the agent. This is similar to the situation where an animal in nature (a) has to process sensory input (e.g., visual, auditory, and olfactory signals) to determine what it is encountering and (b) has to predict unavailable information to make the decisions using its own memory. Essentially, we are placing an agent in an environment where certain high level functions (image classification and hidden variable estimation) can help the decision making to induce the functions (Fig. 1(c)).

4 Survival agent

4.1 Agent model

Our agent has three components as shown in Fig. 2(a). The first is a convolutional neural network (CNN) that encodes the input image into a 4-dimensional vector. The second is a recurrent neural network (RNN) that makes predictions on hidden variables based on historical information. Lastly, the agent has a multilayer perceptron (MLP) that calculates the Q-value of each action based on the outputs from the CNN and the RNN.

The network operates as follows. The CNN takes the image v_t from the environment and generate the output $f_t = \text{CNN}(v_t)$. When $t = 1$, the RNN is not utilized: h_1 bypasses the RNN and directly goes to the MLP. For $t > 1$, the RNN takes its previous output, previous feature vector and previous action tuple, $(\hat{h}_{t-1}, \text{CNN}(v_{t-1}), a_{t-1})$, and generates the output $\hat{h}_t$ whose dimension is the same as the number of hidden variables ($i.e.$, $\hat{h}_t = \text{RNN}(\hat{h}_{t-1}, \text{CNN}(v_{t-1}), a_{t-1}) \in \mathbb{R}^2$). The estimated

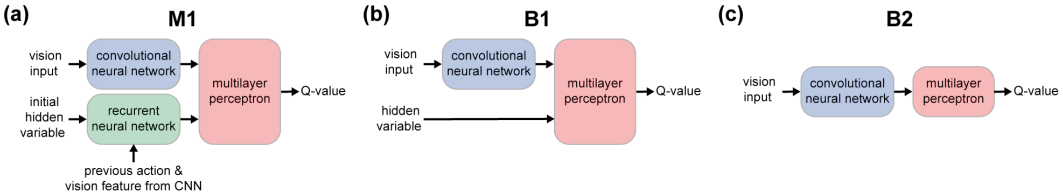

Figure 2: (a) The network architecture of our agent M1. A convolutional neural network (CNN) receives vision inputs from the environment and a recurrent neural network receives (RNN) the initial value of hidden variables, the previous action and the previous feature vector from the CNN. The outputs from the CNN and RNN are fed to a multilayer perceptron (MLP) that generates the Q-value of each action. (b) The network architecture of a baseline model B1. True hidden variables are directly fed to the MLP. Otherwise the same as M1. (c) The network architecture of a baseline model B2. The MLP takes the output only from the CNN. Otherwise the same as M1.

hidden variable $\hat{h}_t$ and the CNN output f_t are then fed to the MLP to calculate the Q-value (i.e., $Q(v_t, h_1, a) = \text{MLP}(\text{CNN}(v_t), \hat{h}_t, a)$). The details on the network design are in Appendix C.

4.2 Baseline models

In addition to our agent described in Section 4.1, which we denote by M1, we implemented two baseline models for comparison. The first baseline model B1 differs from M1 in that the true *hunger* and *sickness* variables are directly fed to the MLP (Fig. 2(b)), and therefore B1 does not have to perform hidden variable prediction. In the second baseline model B2, only the output from the CNN is fed to the MLP without any direct or indirect exploitation of the *hunger* and *sickness* variables (Fig. 2(c)). The architectures of CNNs and the MLPs in M1, B1 and B2 are the same. Therefore, the achievable performance gap between B1 and B2 represents the importance of the *hunger* and *sickness* variables for choosing the action. Since our agent M1 has an RNN which takes the initial values of the hidden variables, its achievable performance is upper-bounded by that of B1 which has direct access to the variables and lower-bounded by that of B2 which does not have any access to the variables.

4.3 Update method

We first tested the *Random Update* method Kapturowski et al. (2019) for training the agent with recurrent components, where the estimated hidden state $\hat{h}_t$ from RNN is stored in the replay buffer when the agent interacts with the environment. Then, randomly sampled consecutive transactions $\tau = ((v_t, \hat{h}_t, a_t, r_t), \ldots, (v_{t+L_{ran}-1}, a_{t+L_{ran}-1}, r_{t+L_{ran}-1}))$, where L_{ran} is fixed sequence length, are used for the parameter update. We note that, in this setting, inaccurate hidden states stored in the replay buffer may hinder learning, since hidden variables depend heavily on their previous values and are randomly initialized in every episode.

To alleviate this issue, we developed a *Sequential Update* method that is built upon Hausknecht & Stone (2015) and Kapturowski et al. (2019) to train a network with recurrent units with hidden states. We first split each episode $e = ((v_1, h_1, a_1, r_1), (v_2, a_2, r_2), \ldots, (v_T, a_T, r_T))$ into sub-episodes with a fixed length L_{seq} to construct mini-batchs, where T is the terminal time. For the sub-episodes that begin at $t \neq 1$, the initial value of the hidden variables was set as the RNN output $\hat{h}_{t-1}$, because the true "initial" values of the hidden states are unknown. Then, the parameter update is performed by sequentially going through all transactions in the randomly sampled sub-episodes (Appendix D).

5 Results

In this section, we present our experimental results, which confirm that high level functions can be induced without task specification, and we compare the results from different settings. The baseline models B1 and B2 were trained with deep Q-learning Mnih et al. (2015), and our agent M1 was trained with a variant of deep recurrent Q-learning Hausknecht & Stone (2015); Kapturowski et al. (2019), which was described in Section. 4.3. The networks were trained using an Adam optimizer with

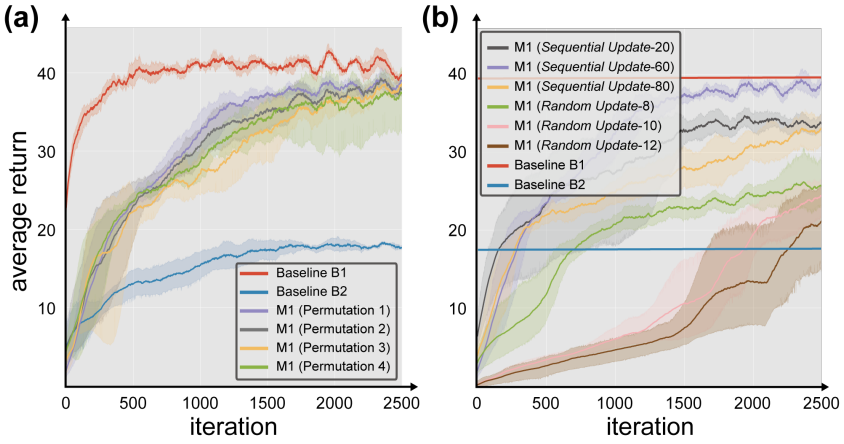

Figure 3: (a) Average learning curves of the agents M1, B1 and B2 from five episodes for each model. B1 with direct access to the hidden variables shows the highest performance (red). The average return of M1, which takes the initial value of the hidden variables as the input to its RNN, approaches that of B1 regardless of the digit assignment to each class (purple, grey, yellow, green). The average return of B2, which does not receive any information on the hidden variables, shows the lowest performance (blue). One iteration corresponds to 500 gradient updates. (b) Average learning curves of the agents M1 obtained with multiple learning methods and sub-episode lengths. sequential update (grey, purple, yellow) achieved higher performance than random update (green, pink, brown) proposed in Kapturowski et al. (2019). The performance of B1 (red) and B2 (blue) after training is overlaid for comparison.

the learning rate of 3×10^{-4} for 1.2 million gradient updates. The target network was updated with Polyak averaging. For training B1 and B2, we used replay buffers with a size of 10^6. For M1, 10^5 sub-episodes that contain roughly 6×10^6 transactions were saved in the replay buffer. The networks were implemented using Pytorch and trained on a workstation with two Intel Xeon Scalable Silver 4214R CPUs, four NVIDIA GeForce RTX 2080 Ti GPUs, and 128 GB of RAM. The full implementation of the network and the environment is available at `https://github.com/anonymous`[1].

5.1 Learning curve

First we verified the achievable performance of the agent M1 and the baseline models B1 and B2. Fig. 3(a) shows the average learning curves from five episodes for each model. B1 achieved an average return of around 40, whereas B2 had an average return of less than 20. This performance gap between B1 and B2 represents the importance of taking the hidden variables into account for choosing actions. Only for M1, we used four different permutations for the digit assignment to each class (Appendix A). By the end of the training, the average return of M1 was close to that of B1 regardless of the digit assignment which implies that the images were classified by their class and the RNN was successfully trained and played an important role in the decision making. In Fig. 3(b), the average learning curves of M1 obtained with different learning methods are compared. We evaluated our sequential update method with multiple sub-episode lengths $L_{seq} = 20, 60, 80$ and the update method proposed in Kapturowski et al. (2019) with multiple sequence lengths $L_{ran} = 8, 10, 12$. Sequential update methods with a sub-episode length of 60 achieved the highest performance.

5.2 Learning to classify images

To verify whether the network maps the input images to the feature vectors that are linearly separable by their corresponding classes (i.e., *none*, *predator*, *prey* and *rotten food*), we extracted the CNNs from M1, B1 and B2, denoted by CNN-M1, CNN-B1 and CNN-B2, respectively, after the training was completed. Then, we fed MNIST images — both that were used and not used for the reinforcement learning — to CNN-M1, CNN-B1 and CNN-B2 and obtained 4-dimensional output vectors. Using the output vectors from each network $f_t = \text{CNN}(v_t) \in \mathbb{R}^4$ as the input, we trained a linear support

[1] anonymized for review

Table 1: Classification accuracy of linear support vector machine classifiers. Each classifier is trained to classify the corresponding object class of the feature vectors from each CNN. The classification accuracy embodies the capability of each CNN to map the images to the linearly separable feature vectors. CNN-M1 shows consistent accuracy regardless of the digit assignment to each class. The average accuracy and standard deviation were obtained from five episodes with random seeds.

Network	Training accuracy	Test accuracy
CNN-B1	99.28 ± 0.167	95.26 ± 0.678
CNN-B2	80.06 ± 3.018	75.60 ± 3.350
CNN-M1 (permutation 1)	98.01 ± 0.495	93.86 ± 0.503
CNN-M1 (permutation 2)	97.90 ± 0.591	93.03 ± 0.714
CNN-M1 (permutation 3)	98.20 ± 0.385	93.01 ± 0.766
CNN-M1 (permutation 4)	98.67 ± 0.377	94.37 ± 0.450

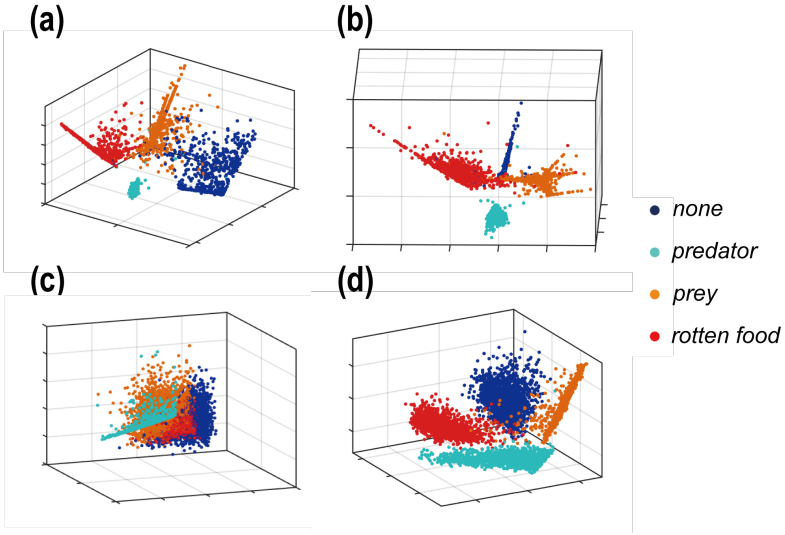

Figure 4: Principal component analysis is applied to the feature vectors from the CNN to reduce their dimension from 4 to 3 for visualization. The vectors are linearly separable by their corresponding classes. (a) Result from CNN-M1 (permutation 1). (b) Result from CNN-M1 (permutation 3). (c) Result from CNN-B2. (d) Result from CNN-B1.

vector machine classifier and measured the classification accuracy. A total of 8,000 and 32,000 images were used for measuring training accuracy and test accuracy, respectively.

The convolutional network from our agent CNN-M1 achieved an average training and test accuracy of 98.20% and 93.57%, respectively, which shows that the CNN is capable of classifying not only the images used for training, but also the unseen images. The classification accuracy was consistent across different permutations. The accuracy from CNN-B1 was slightly higher than that from CNN-M1, whereas the accuracy from CNN-B2 was significantly lower. The low performance of CNN-B2 is likely because B2 was not able to comprehend the environment. The results from all the networks are summarized in Table 1.

For visualization of the capability of CNN-M1, CNN-B1, and CNN-B2, we applied principal component analysis to the output vectors and reduced their dimension from 4 to 3 (Fig. 4). The result shows that the mapped feature vectors are linearly separable by their classes except for CNN-B2, which lacks the capability to incorporate the hidden variables for making the decisions.

It should be noted that the images with two randomly picked digits from the MNIST dataset were assigned to each class, which indicates that the network learned to cluster images based on their contextual meanings as opposed to simply clustering visually similar images. In addition, the image class and the optimal action are decoupled in our setting, which also confirms that the CNNs learned to classify images (See Section 5.4).

5.3 Learning to predict hidden variables

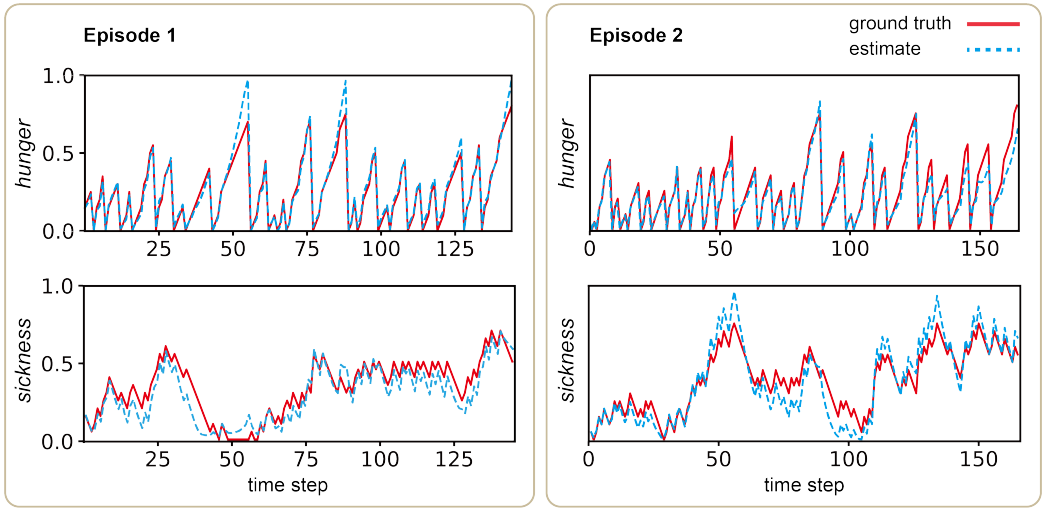

Figure 5: Ground truth values of the hidden variables (solid red) and the RNN output (dotted blue). The time courses of the variables from two episodes are shown. The average Pearson correlation coefficients between the ground true hidden variables and the RNN output from 200 episodes were 0.9772 and 0.9450, for *hunger* and *sickness*, respectively.

Next, we verified whether the network could yield output values that were linearly proportional to the hidden variables using M1 (permutation 1). After the training was completed, we collected the test episodes from 200 episodes. For each episode, the output from the RNN was compared with the true hidden variables as shown in Fig. 5. It is clear that the RNN is capable of predicting the hidden variables despite the fact that it was never specified to do so and it did not have access to the ground truth values of the hidden variables: We simply let the MLP take the output from RNN as its input. The average Pearson correlation coefficients between the ground true hidden variables and the RNN output were 0.9772 and 0.9450, for *hunger* and *sickness*, respectively.

5.4 Vision-Action dependencies

Fig. 6 shows that the agent chooses different actions for the same object class depending on the *hunger* and *sickness* level (i.e., vision input and the agent's action are decoupled). The *hunger* level was categorized as high if above 0.8 and as low if below 0.4. The *sickness* level was categorized as high if above 0.7 and as low if below 0.65. In M1 and B1, the probability to choose each action depends largely on the *sickness* and *hunger* values, whereas it remains nearly unchanged in B2. This shows that the vision input and the optimal action are decoupled in the environment, which indicates that learning to choose the optimal action is a different task than learning to classify the images.

6 Discussion

The main idea of this work is that even high level functions such as image classification can be naturally induced within the network via reinforcement learning. This is not surprising considering that biological intelligence has developed various cognitive functions due to evolutionary pressure. Our results indicate that various functions can be induced within artificial networks via reinforcement learning if the environment provides pressure to do so and the network architecture can support the functions. While our demonstration was limited to the classification of MNIST images and the prediction of scalar variables, we believe this framework could be applied and extended to induce more complicated functions by placing the agent in a complex environment such as Minecraft, Grand Theft Auto V Gao et al. (2019), and Crafter Hafner (2021).

While this may provide a new framework to develop an artificial intelligence with various cognitive functions, there still are some limitations that need to be overcome. First, an environment that is

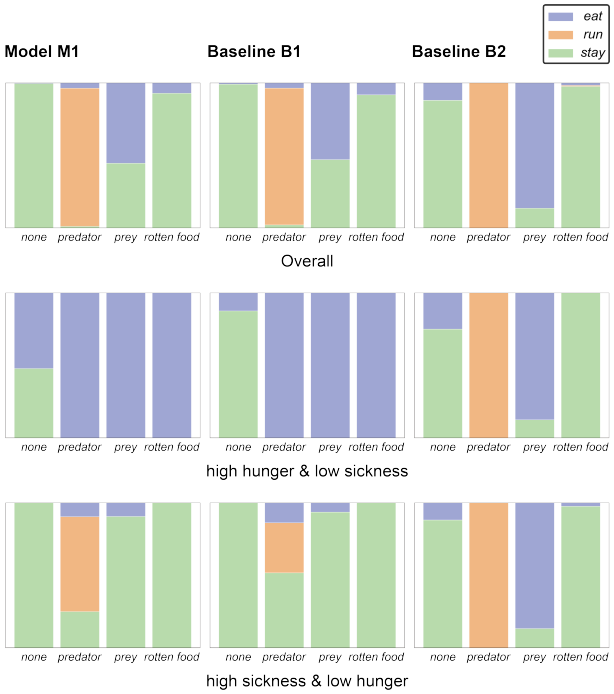

Figure 6: vision-action dependencies of M1 (permutation 1), B1 and B2 after training.

as complex as the real world would be necessary to develop an agent that truly mimics biological intelligence. Considering the infeasibility of developing such environment, it would be necessary to devise a way to let the agent directly interact with the real world. Second, without knowing what functions the agent may develop beforehand, it is difficult to assess the suitability of a network architecture. Potentially, a solution may arise from the field of connectomics Li et al. (2020) which studies the neural network architectures of biological brains.

7 Conclusion

In this paper, we proposed a bio-inspired framework for training a neural network via reinforcement learning to induce high level functions within the network without task specification. The network learned to classify images and estimate hidden variables simply by placing the agent in an environment where such functions were helpful for making the decisions. We argue that this closely resembles why and how biological intelligence has developed various cognitive functions and hence, our strategy can be employed and extended to develop artificial intelligence that is truly similar to biological intelligence.

References

Abrantes, J. P., Abrantes, A. J., and Oliehoek, F. A. Mimicking evolution with reinforcement learning. *arXiv preprint arXiv:2004.00048*, 2020.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the Atari human benchmark. In *Proceedings of the International Conference on Machine Learning*, 2020a.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., and Blundell, C. Never give up: Learning directed exploration strategies. In *Proceedings of the International Conference on Learning Representations*, 2020b.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., Wayne, G., Soyer, H., Viola, F., Zhang, B., Goroshin, R., Rabinowitz, N.,

Pascanu, R., Beattie, C., Petersen, S., Sadik, A., Gaffney, S., and King, H. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557:429–433, 2018.

Barborica, A. and Ferrera, V. P. Estimating invisible target speed from neuronal activity in monkey rontal eye field. *Nature Neuroscience*, 6:66–74, 2003.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2017.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. Infogan: Interpretable representation learning. In *Advances in Neural Information Processing Systems*, 2016.

Dabney, W., Kurth-Nelson, Z., Uchida, N., Starkweather, C. K., Hassabis, D., Munos, R., and Botvinick, M. A distributional code for value in dopamine-based reinforcement learning. *Nature*, 577:671–675, 2020.

Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29:141–142, 2012.

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., and Darrell, T. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:802.05313*, 2019.

Gershman, S. J. Uncertainty and exploration. *Decision*, 6:277–286, 2019.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, 2018.

Hafner, D. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Hausknecht, M. and Stone, P. Deep recurrent q-learning for partially observable mdps. In *Proceedings of the AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.

Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

Kaiser, L., Babaeizadeh, M., Miłos, P., Osiński, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. Model based reinforcement learning for atari. In *Proceedings of the International Conference on Learning Representations*, 2020.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 2019.

Kingma, D. P., Jimenez Rezende, D., Mohamed, S., and Welling, M. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Lehnert, L., Littman, M. L., and Frank, M. J. Reward-predictive representations generalize across tasks in reinforcement learning. *PLOS Computational Biology*, 16:1–27, 2020.

Li, P. H., Lindsey, L. F., Januszewski, M., Zheng, Z., Bates, A. S., Tyka, M., Nichols, M., Li, F., Perlman, E., Maitin-shepard, J., Blakely, T., Leavitt, L., Jefferis, G. S., Bock, D., and Jain, V. Automated reconstruction of a serial-section em drosophila brain with flood-filling networks and local realignment. *bioRxiv*, 2020. doi: https://doi.org/10.1101/605634.

Lin, B., Cecchi, G., Bouneffouf, D., Reinen, J., and Rish, I. A story of two streams: Reinforcement learning models from human behavior and neuropsychiatry. *arXiv preprint arXiv:1906.11286*, 2019.

Makhzani, A., Shlens, J., Jaitly, N., and Goodfellow, I. Adversarial Autoencoders. In *Proceedings of the International Conference on Learning Representations Workshop*, 2016.

Merel, J., Aldarondo, D., Marshall, J., Tassa, Y., Wayne, G., and Ölveczky, B. Deep neuroethology of a virtual rodent. In *Proceedings of the International Conference on Learning Representations*, 2020.

Miller, H. C., Gipson, C. D., Vaughan, A., Rayburn-Reeves, R., and Zentall, T. R. Object permanence in dogs: Invisible displacement in a rotation task. *Psychonomic Bulletin and Review*, 16:150–155, 2009.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Neftci, E. O. and Averbeck, B. B. Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, 1:133–143, 2019.

Niekum, S., Barto, A. G., and Spector, L. Genetic programming for reward function search. *IEEE Transactions on Autonomous Mental Development*, 2:83–90, 2010.

Niekum, S., Spector, L., and Barto, A. Evolution of reward functions for reinforcement learning. In *Genetic and Evolutionary Computation Conference*, 2011.

Pontes-Filho, S. and Nichele, S. A conceptual bio-inspired framework for the evolution of artificial general intelligence. *arXiv preprint arXiv:1903.10410*, 2019.

Rasmussen, D., Voelker, A., and Eliasmith, C. A neural model of hierarchical reinforcement learning. *PLOS ONE*, 12:1–39, 2017.

Schultz, W., Dayan, P., and Montague, P. R. A neural substrate of prediction and reward. *Science*, 275:1593–1599, 1997.

Singh, S., Barto, A. G., and Chentanez, N. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, 2005.

Singh, S., Lewis, R. L., and Barto, A. G. Where do rewards come from? In *Proceedings of the International Symposium on AI-Inspired Biology*, 2010.

Sutton, R. S. Learning to predict by the methods of temporal differences. In *Machine Learning Proceedings*, 1991.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in starcraft 2 using multi-agent reinforcement learning. *Nature*, 575:350–354, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

(b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Please check our supplemental material.

   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.

   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Figure 3.

   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 2 for MNIST data citation, and please check our submitted code for code citation.

   (b) Did you mention the license of the assets? [Yes]

   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]