2-15-2018

Explaining Decisions of a Deep Reinforcement Learner with a Cognitive Architecture

Sterling Somers

Constantinos Mitsupoulos

Christian Lebiere

Robert Thomson
Army Cyber Institute

Explaining Decisions of a Deep Reinforcement Learner with a Cognitive Architecture

Sterling Somers (sterling@sterlingsomers.com)
Constantinos Mitsopoulos, Christian Lebiere ([cmitsopoulos, cl]@cmu.edu)
Department of Psychology, Carnegie Mellon University, 5000 Forbes Ave
Pittsburgh, PA 15213 USA

Robert Thomson (robert.thomson@usma.edu)
Army Cyber Institute, United States Military Academy, 2101 New South Post Road
West Point, NY, 10996 USA

Abstract

The work presented is an evaluation of a method for developing a hybrid system, consisting of a Deep Reinforcement Learning (RL) agent and a cognitive model, capable of providing explanations of its action decisions. The methodology uses a symbolic/sub-symbolic cognitive architecture to introspect on the activity of the network to understand its representation. The entropy in the system's behavioral predictions could be used as a signal to affirm or deny ascribing a representation to the network.

Keywords: deep reinforcement learning; cognitive modeling; introspection

Introduction

Deep Neural Networks (DNNs) have demonstrated an increasing success in various scenarios where they are used as function approximators. Their success is based in passing low-level sensory information through their structure, and creating higher-level abstractions of that information. The output of the network can be used either to classify inputs, predict, or in the case of RL, which is a learning paradigm for decision-making processes, to make decisions. As DNNs proliferate in both academic and private sectors, there will be an increased value to developing Deep RL agents that can be introspected upon. Being able to understand the concepts abstracted by the DNNs and how those concepts are factored into action decisions will allow us to develop specialized agents trusted to achieve their purpose.

The aim of this work is to develop a methodology for providing symbolic-level explanations for the action decisions of a Deep Reinforcement Learning (RL) agent. Our approach uses a cognitive architecture to model the internal representations of an RL agent at a symbolic level. In the long-term we envision a system that can inform a human observer why the RL agent made the decisions it made. This is particularly important whenever there is a mismatch between the knowledge of the observer and the knowledge of the RL agent. A mismatch could occur: a) when the RL agent produces an optimal but atypical solution, or b) when the RL agent fails to find a solution known to a subject matter expert (SME), or c) when the feature and action space are too complex for an SME to determine efficiently why a RL agent has made certain decisions. We envision that in a) and c), our system would be able to identify the most salient features of the environment attended by the RL agent, thereby explaining what features it is responding to. In b) our system would be able to identify representations available to an SME that are not shared by the RL agent. In this work we attempt to address b) by outputting symbolic state/action combinations consistent with the terminology used by an SME.

We believe that such a system should be generative and/or adaptive. While state and action category names should come from the SME, the system should be able to learn the mapping between states and actions for some subset of the RL agent's behavior.

The explanations produced by the system should also be constrained by human cognitive capacities. A deep RL agent is not limited with respect to the number of features it can attend to. An explanation system that fails to reduce the cognitive load on the observer is not useful as an explanation because the onus of determining the most salient features remains with the human observer.

In this work we evaluate a methodology where we create an adaptive cognitive model of an RL agent trained to play a simple mission in the real-time strategy game, StarCraft 2 (SC2). Our approach involves introspecting into the RL agent by mapping the activity of neural layers, as it makes action decisions, to symbolic output that can be used to generate explanations. This approach is reminiscent of Vinokurov et al. (2011), where they combined the hybrid cognitive architecture, ACT-R, with the neural-network-based architecture, Leabra (O'Reilly & Munakata, 2000) in an image classification task. We believe the computational cognitive architecture, ACT-R, is well suited for such a task because it is constrained by cognitive capacities that, in turn, will constrain explanations; it creates a symbolic model trace that can be tractably transformed into explanations; and it operates on sub-symbolic equations that are compatible with distributed representations in DNNs.

The cognitive model is initialized with knowledge consistent with an SME and evaluated against an RL agent with a sub-optimal policy. A sub-optimal policy represents knowledge mismatch situation b), described above, where the RL agent has failed to find a solution readily available to an SME. Although it may be possible to create an RL agent that performs the task optimally, our aim is primarily focused on explanation. Our explanation system is tasked with identify-

ing and reporting to a human user when the representations provided by the SME are not appropriate for describing the internal representations used by the network and to provide instead a representation that better describes the RL agent's internal state.

Our cognitive model attempts to adapt its internal representations to better match the action decisions of the RL agent. The cognitive model provides a model trace that can be used to infer an explanation. We use this trace in this investigation to determine if the RL agent has made an inference consistent with an inference made by an SME. High entropy in the model, in categories related to the inference, indicate that the RL does not make the inference. We confirm the accuracy of the model trace using t-SNE (Maaten & Hinton, 2008) to cluster the activity of the network. The t-SNE clustering confirms that the network makes three distinct abstract representations from the input space, which correspond to three state/action categories instead of the four that would result from making the same inference as the SME.

Task and Environment

StarCraft 2 is a real-time strategy game where players control units from a third-person perspective with the aim of eliminating opponents. In the screen-shot depicted in Figure 1, a military unit is selected and two beacons are present on the map.

Figure 1: Screenshot of a StarCraft 2 beacon mission.

SC2 provides a rich and complex environment for experimenting on various types of agents. The specific domain presents multiple challenges: imperfect information, macro and micro management of resources, strategic acting and multi-agent interactions. In addition, the SC2 Editor provides complete freedom to the mission's design, although anything beyond very basic missions provides a significant learning challenge for Deep RL agents (Vinyals et al., 2017).

The mini game we present here is straight forward. The goal of the game is to get the agent to one of two beacons: either the green beacon or the orange beacon. The green beacon presents a low-value target while the orange beacon represents a high-value target.

Despite the simplicity of the mission, the network learned a sub-optimal strategy. The agent clicks the highest rewarding beacon presented in every case. This strategy is successful in every case but one: where the green beacon is in the direct path between the agent and the orange beacon (blocking scenario). Clicking directly on the highest rewarding beacon makes the agent move in a direct line between its location and the beacon. In the blocking scenario, the agent moves towards the orange beacon, hits the green beacon, receives the low-value reward, and the game is reset.

Deep Reinforcement Learning Agent

The core of the agent consists of a deep neural network that attempts to solve a Markov Decision Process (MDP) defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R})$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions, $\mathcal{T}$ is the transition probability for arriving in state s' when executing action a from state s, $\mathcal{R}$ is the reward function that defines the reward received for performing the aforementioned transition, and γ a reward discount factor. The goal of the agent is to maximize the expected return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ from each state s_t. The solution of the MDP consists of a function, named policy $\pi(.|s_t)$, that maps a state s_t to a distribution over actions that lead the agent to higher sums of rewards. The probability of performing action a_t in state s_t is denoted as $\pi(a_t|s_t)$.

The network parametrizes the policy π_θ with parameters θ. In this paper, we utilize the Advantage Actor Critic (A2C) algorithm which is the synchronous version of the A3C (Mnih et al., 2016). We adopt the same architecture (Figure 2) and implementation details as in Vinyals et al. (2017).

We consider the standard interaction between agent and environment. At each time step t the agent receives an observation s_t from the Starcraft II API and selects an action a_t according to its policy $\pi(.|s_t)$:

- **Observations:** consist of a set of image-like feature layers that represent the existence of a specific feature at a specific location on a screen. For example, in a 32×32 resolution map a feature that corresponds to enemy units type will be represented as a matrix with the same dimensions as the map. This feature matrix will have zero values apart from the elements that correspond to the pixels that are occupied by the enemy units. The value of these elements will be equal to the unit type identification number provided by the API. Generally, there are three main types of features: *map features* (entire map), *screen features* (part of map) and non-pixel information (e.g. player resources).

- **Actions:** There are two action categories: the action type and the action arguments. For example *click on* (action type), and x, y *location* (action arguments). In our examples, the action arguments will be the spatial location for performing the particular action type.

We trained the agent in the beacon task until it reached a steady performance relative to a human optimal score. We observed that the agent clicked on the highest value target in all cases. The RL agent exhibited no avoidance behavior in the blocking scenario.

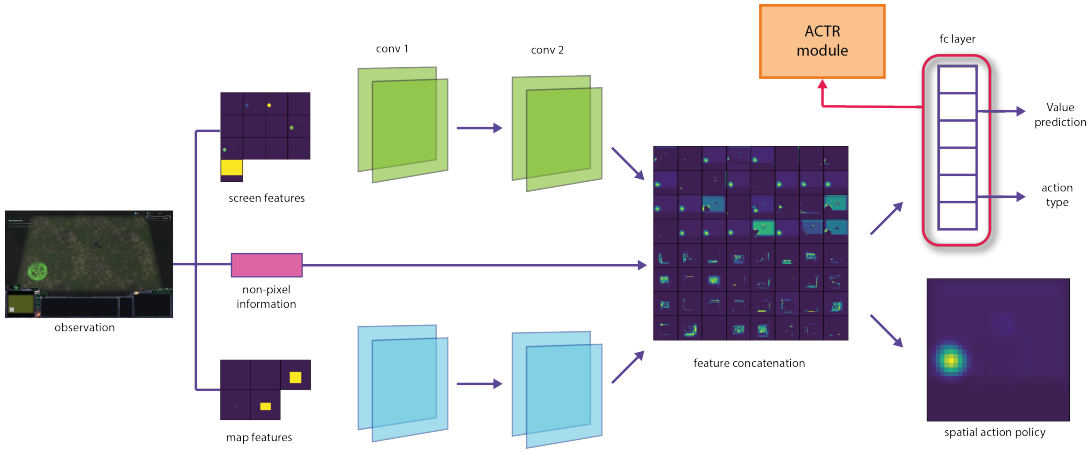

Figure 2: The Hybrid architecture: From observations, image-like features are generated for screen and map information. These are passed through two convolution layers and are concatenated with the non-spatial features. The value prediction, which represents the expected reward from the current observation, and the action type are determined by the concatenated feature representation passed through a fully connected (fc) layer with 256 units. The activities of this layer are sent to the ACT-R module for further processing. The spatial action is sampled from the probability distribution formed by a 1×1 convolved representation of the feature concatenation.

Cognitive Model

Our model is introspective in that it uses the activity of the network as the basis for ascribing representations. The RL agent and the model share a common ground: both play the game at the same time, the RL agent receiving numerical values describing the ground truth of the game state, while the model receives a symbolic version. Symbolic content comes from both interpreting signals from the game (presence of beacons) and an ontology created by SMEs. The ontology represents the kind of knowledge that a competent player should have. In the missions presented in this paper, the ontology is sparse: click on the green-beacon in green-only scenarios, click on the orange in orange-only scenarios, click on the orange in non-blocking scenarios, and go around the green beacon in blocking scenarios. Our cognitive model is developed in the ACT-R cognitive architecture.

ACT-R

ACT-R is a computational implementation of a unified theory of cognition (Anderson et al., 2004). It accounts for information processing in the mind via task-invariant mechanisms constrained by the biological limitations of the brain (see Anderson 2007 for an overview). The ACT-R architecture is organized as a set of modules, each devoted to processing a particular kind of information, which are integrated and coordinated through a centralized production system module.

The declarative memory (DM) and production system modules, respectively, store and retrieve information that corresponds to declarative knowledge and procedural knowledge. Declarative knowledge is knowledge that a person can attend to, reflect upon, and usually articulate in some way. Procedural knowledge consists of the skills we display in our behavior, generally without conscious awareness. Declarative knowledge in ACT-R is represented formally in terms of chunks. The information in the declarative memory module corresponds to personal episodic and semantic knowledge that promotes long-term coherence in behavior. In this sense, a chunk is like a data frame, integrating information available in a common context at a particular point in time in a single representational structure.

Each chunk has a base-level activation that reflects its past recency and frequency of occurrence. When a retrieval request is made, the most active matching chunk is returned from long-term declarative memory by an activation process. This process is computed as the sum of base-level activation, spreading activation, mismatch penalty and stochastic noise. Activation spreads from the current focus of attention, including goals, through associations among chunks in declarative memory. These associations are built up from experience, and they reflect how chunks co-occur in cognitive processing. The spread of activation from one cognitive structure to another is determined by weighting values on the associations among chunks.

Chunks are also compared to the desired retrieval pattern using a partial matching mechanism that subtracts from the activation of a chunk its degree of mismatch to the desired pattern, additively for each component of the pattern and corresponding chunk value. Finally, noise is added to chunk activations to make retrieval probabilistic, governed by a Boltzmann distribution.

While the most active chunk is usually retrieved, a blending process (Lebiere, 1999) can also be applied that returns a derived output reflecting the similarity between the values of the content of all chunks, weighted by their retrieval probabil-

ities reflecting their activations and partial-matching scores.

The flow of information is controlled in ACT-R by a production system, which operates on the contents of the buffers. Each production consists of if-then condition-action pairs. Conditions are typically criteria for buffer matches, while the actions are typically changes to the contents of buffers that might trigger operations in the associated modules. The production with the highest utility is selected to fire from among the eligible productions.

Mental Model

Our model is instance-based (Gonzalez et al., 2003). Whenever the model makes an action decision, it does so based on the similarity of the current situation to situations that it has stored in declarative memory. Instances are represented in the model as a chunk with the following five slots: *green*, *orange*, *blocking*, *vector*, and *action*. The slots green, orange, and blocking are binary True/False, the vector slot is populated with a chunk holding a 256-dimension vector, representing activity in the network (described below), and values for the action slot can be either: *select-green*, *select-orange*, or *select-around*.

Unlike many previous instance-based models in ACT-R, this instance-based model does not use ACT-R's blending mechanism since the output action is categorical in nature. Instead of blending a value estimate of the action category, our model simply selects the action from the stored instance that most closely matches the network activity of the RL agent.

Instances

We populated the model instances by tracing the RL agent while it plays different scenarios. In total, the RL agent played 50 games. Each game consists of a two-minute time period where a configuration of beacons is presented. The number of configurations presented varies based on the time it takes for the RL agent to reach a beacon. Once the agent reaches a beacon, a new configuration is presented, until the two minutes elapsed. Symbolic terms describing the scenario were stored in the instance (green-only, orange-only, green-and-orange). We populated the *action* slot of the model's declarative memory with the action an expert would choose: select-green in a green-only scenario, select-orange in both orange-only and non-blocking scenarios, and select-around (as an abstract action) in the blocking scenario. Importantly, given these instances, the model would never predict that the RL agent would select the orange beacon in a blocking scenario. Finally, stored for every instance in declarative memory, was a chunk containing a vector representing the network activity at that instant of game-play (introspection).

Introspection We augmented the symbolic context representation in the instance chunks with a vector that represents the network activity from the *fc* layer (top right, Figure 2). We chose this layer for two reasons. First, we wanted a layer that would be abstract enough to represent scenarios (as opposed to features in the scenarios). Second, we wanted a layer that was spatially invariant. The *fc* layer corresponds to the non-spatial action selection and chooses which action type to take but does not choose the screen location of that action to be executed. For example, it might be responsible for choosing a click-action but is not responsible for choosing where to click.

For efficiency, we follow Sanner et al. (2000), and limit the number of instances that are stored. In our case, new instances are limited to 10 items per category such that the distance of the new vector must be greater than the minimum distance between vectors across all categories. After initial testing, we limited the total number of instances per category to ten. Since each instance also stores a vector of network activity and the vectors drive the similarity measure in ACT-R, we attempted to capture a large space of vectors in each category. We chose to filter the creation of categories by maximizing the euclidean distance between vectors.

Instance-Based Learning and Instance Retrieval

Once we populated the initial set of instances in ACT-R's declarative memory, it tries to classify new game states. We use ACT-R's partial matching mechanism such that it tries to retrieve from memory instances that are most similar based upon a vector representing network activity. The similarity metric we used was euclidean distance between vectors, normalized to a scale of 0 to -1, for all vectors in the category. All other slots are not matched upon. The result of this process is that the instance with the closest vector is recalled, regardless of the symbolic content of that memory. It is possible, therefore, that symbolic content of the recalled instance does not match the symbolic content of the current game state. In fact, we are expecting that in blocking cases, the model will retrieve a non-blocking category. Once an instance is retrieved, a production that matches the action stored in the instance will fire. The production that fires is recorded for evaluation. A side effect of the production is to combine the data from the current game state and the instance retrieved from memory to create a new memory.

Memory Creation for Model Adaptation The process of creating new memories is the central feature that makes the model adaptive. As we allow for a mismatch between the symbolic content of the game state and the symbolic content of the recalled instance, the model is capable of creating new state/action categories. The new state/action categories can then be used by the model as it continues to play the game. The new memories store the vector for the current activity of the RL agent network, the current game state (green, orange, blocking), and the action from the retrieved instance, combined as a new chunk.

Evaluation

The goal of our system is to provide symbolic terms, and a model trace that can be used to infer an explanation. Our models introspects on the network and ascribes plausible representations to it.

Although it is typical to assess a model on how well it predicts behavior, our interest is not how well the model fits the network but what the performance might tell us about the appropriateness of an ascribed mental state or explanation. Since we conducted an evaluation of the network's competency, we know ahead of time that the network does not appear to represent a spatial relationship between the agent and the beacons. We inferred from this behavior that the network does not make a spatial inference. We hypothesized that by using partial matching on the vector slot only, the model would conflate blocking and non-blocking scenarios, occasionally retrieving the select-orange action (appropriate in non-blocking scenarios) and create a new instance category: green, orange, blocking, select-orange.

Analysis

Since each prediction that the model makes is categorical (an action), we treat each guess as binary (correct/incorrect).

We ran the model 100 times. Each run consisted of 10 episodes of the game, and each episode consists of roughly 25 randomly generated scenarios: green-only, orange-only, green-and-orange-non-blocking, and green-and-orange-blocking. It was difficult to estimate a number of simulation runs that would be practical and provide a reasonable estimation of the model's performance. Variance was observed to be stable at 100 runs. Importantly, since we are not overly concerned about model fit, finding a true mean was not absolutely essential.

Percent Correct The model correctly predicted the actions of the RL 70.09 % of the time across all runs. In green-only scenarios, the model predicts 99.95 % of the actions. In orange-only scenarios, the model predicts 89.39 % of the actions. In green-and-orange (Blocking and Non-Blocking) scenarios, the model predicts 44.85 % of the actions. This can be broken down further into two sub-categories: blocking and non-blocking scenarios. In blocking scenarios, the model predicts 36.04 % of the actions. In non-blocking scenarios, the model predicts 49.04 % of the actions.

New Categories As previously mentioned, the model has knowledge of what action to choose in a green-only scenario, an orange-only scenario, non-blocking scenarios, and blocking scenarios. The model attempts to make new categories based upon the RL network activity.

Table 1 summarizes what scenario/action pairs were used and created by the model. The top of the table are the state/action categories that the model was initialized with and the bottom of the table are the state/action categories created by the model, during game play. The left most column is the scenario categories. Column 2 (Action) is the associated action taken by the network in that scenario. Column 3 (Cat./Run) represents the average frequency the scenario/action category was created per model run. Column 4 is measure of how often the associated action was chosen in the given scenario. Note that the symbolic content in the table (e.g. *blocking, select orange*) are the categorical output of the cognitive model, that can be used to infer explanations.

Table 1: Category Creation and Use

Scenario	Action	Cat./Run	Chosen (%)
Green-Only	Select-Green	default	99.95
Orange-Only	Select-Orange	default	88.39
Non-Blocking	Select-Orange	default	49.04
Blocking	Select-Around	default	63.88
Blocking*	Select-Orange	9.98	36.04
Green-Only	Select-Orange	<0.01	0.03
Orange-Only	Select-Green	3.03	3.63
Non-Blocking	Select-Green	0.40	0.18
Blocking	Select-Green	0.30	0.17
Green-Only	Select-Around	0.01	0.02
Orange-Only	Select-Around	5.48	7.98
Non-Blocking*	Select-Around	10.00	50.80

* indicates expected categories.

Clustering We performed t-SNE clustering (Maaten & Hinton, 2008) on the activity of the *fc* layer of the RL network in order to investigate the network's representations. The results of the clustering are illustrated in Figure 3. As illustrated, the network has distinct categories for both the green-only and orange-only scenarios. Also illustrated in the figure, green-and-orange (non-blocking) and blocking scenarios show a complete overlap.

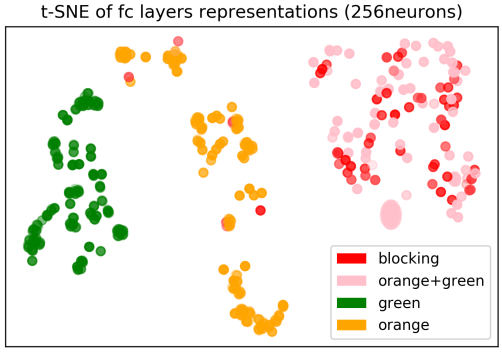

Figure 3: The RL network's representations formed at the fully connected layer are naturally clustered depending on the scenario that the agent is facing. The t-SNE process identifies 3 clusters, coloring is the result of semantic labelling.

Discussion

We know from evaluating the RL agent that it does not perform an action consistent with Select-Around. There are two ways to interpret this behavior: first, that the RL agent does not know how to go around; or second, that the RL agent does not have the concept, 'blocking', and therefore does not distinguish blocking scenarios from non-blocking scenarios.

The results of our model suggest that the latter is the case. Our model attempts to use the concept 'blocking' and creates a new state/action category: Blocking/Select-Orange, consistent with the behavior of the RL agent. Although this state/action category reflects the action of the agent, the cognitive model only approaches 50 % success in its predictions. This is a first indication in the trace that the model does not create the 'blocking' category (as opposed to an inability to go around).

Although we had not originally expected it, the cognitive model creates the state/action category Non-Blocking/Select-Around. The creation (and high usage) of the Non-Blocking/Select-Around in combination with the Blocking/Select-Orange category suggests that the RL agent cannot distinguish between green-and-orange-non-blocking and green-and-orange-blocking: that is, it suggests the RL agent does not have the category 'blocking'.

These findings are confirmed in the t-SNE clustering. In particular, the overlap between the pink and red dots in Figure 3 suggest that RL agent conflates blocking and non-blocking categories. Just as in the model, the clustering algorithm reveals only three main categories: green-only, orange-only, and green-and-orange.

Other categories created by the cognitive model (e.g. Orange-Only/Select-Around), although unexpected, are likely due to noise in either our similarity measure (not sensitive enough) or the RL's distributed representation. For the most part, those categories have both a low category creation rate and low usage percentage. It is worth pointing out that the Orange-Only/Select-Green and Orange-Only/Select-Around have a high-category creation per run (3.03, 5.48 respectively) and percent chosen (3.63 percent and 7.98 percent of the time) compared to the other unexpected categories. Overall, their usage remains quite low.

Conclusion

The work presented in this paper is an initial evaluation of a methodology for generating explanations for the behavior of Deep Reinforcement Learners. We use a computational cognitive model to introspect upon the activity of the network. Given an initial set of classifications, defined by categories present in an ontology, the model either uses existing categories to choose an action (a prediction of the network's action) or creates a new instance by combining content from the current game-state and the retrieved declarative memory instance.

We believe the output of the model can be used to generate explanations and, in particular, our system is able to detect when concepts from the ontology (the concepts humans use) are not realized in the RL agent. The model presented only begins to scratch the surface of what can be accomplished in the overall methodology.

We are currently pursuing three lines of research to expand our approach. We have developed a method for leveraging the blending mechanism in ACT-R to determine which features are most salient in an action decision. Providing the key features that were used in an action decision helps make the explanation more tractable for a naive observer. We are developing top-down interaction such that the ACT-R model can influence the training of the Deep RL agent. We do this to try and influence the agent to learn a concept that it has had trouble learning or that appears in the ontology but not the RL agent. There may be times when we want a more natural mapping between a human observer and an RL agent so that explanations are more straightforward. Finally, we are in the process of developing an ACT-R model that learns the task in a human-constrained manner.

Acknowledgments

This work was funded by a subcontract from PARC under DARPA contract FA8650-17-C-7710.

References

Anderson, J. R. (2007). *How Can The Human Mind Occur In The Physical Universe?* New York, NY: Oxford University Press.

Anderson, J. R., Bothell, D. J., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004, oct). An integrated theory of the mind. *Psychological review*, *111*(4), 1036–60. doi: 10.1037/0033-295X.111.4.1036

Gonzalez, C., Lerch, J. F., & Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science*, *27*(4), 591–635. doi: 10.1016/S0364-0213(03)00031-4

Lebiere, C. (1999). The dynamics of cognition: An ACT-R model of cognitive arithmetic. *Kognitionswissenschaft*, *8*(1), 5–19. doi: 10.1007/s001970050071

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, *9*(Nov), 2579–2605.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

O'Reilly, R. C., & Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience* (Vol. 46). MIT Press.

Sanner, S., Andrew, S., Edu, C. M. U., Anderson, J. R., Lebiere, C., Andrew, C. L., ... Lovett, M. (2000). Achieving Efficient and Cognitively Plausible Learning in Backgammon. In *Seventeenth international conference on machine learning* (pp. 823–830). Stanford, California.

Vinokurov, Y., Lebiere, C., Herd, S., & O'Reilly, R. (2011). A Metacognitive Classifier Using a Hybrid ACT-R/Leabra Architecture. In *Proceedings of the 15th aaai conference on lifelong learning* (pp. 50–55).

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... others (2017). Starcraft ii: a new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*.