[Re] How Attention Can Create Synaptic Tags for the Learning of Working Memories in Sequential Tasks

Erwan Le Masson, Frédéric Alexandre

ReScience

[Re] How Attention Can Create Synaptic Tags for the Learning of Working Memories in Sequential Tasks

Erwan Le Masson[1, 2, 3] **and Frédéric Alexandre**[2, 1, 3]

1 LaBRI, Université de Bordeaux, Bordeaux INP, CNRS, UMR 5800, Talence, France **2** INRIA Bordeaux Sud-Ouest, 200 Avenue de la Vieille Tour, 33405 Talence, France **3** IMN, Université de Bordeaux, CNRS, UMR 5293, Bordeaux, France

frederic.alexandre@inria.fr

Editor
Olivia Guest

Reviewers
Julien Vitay
Etienne Roesch

Received Aug, 9, 2016
Accepted Dec, 1, 2016
Published Dec, 9, 2016

Competing Interests:
The authors have declared that no competing interests exist.

Article repository

Code repository

Data repository

Jupyter notebook

A reference implementation of

→ How Attention Can Create Synaptic Tags for the Learning of Working Memories in Sequential Tasks, J. Rombouts, M. Bohte and P. Roelfsema (2015), PLoS Computational Biology 11.3, e1004060. DOI: 10.1371/journal.pcbi.1004060

Introduction

The reference paper [2] introduces a new reinforcement learning model called Attention-Gated MEmory Tagging (AuGMEnT). The results presented suggest new approaches in understanding the acquisition of tasks requiring working memory and attentional feedback, as well as biologically plausible learning mechanisms. The model also improves on previous reinforcement learning schemes by allowing tasks to be expressed more naturally as a sequence of inputs and outputs.

A Python implementation of the model is available on the author's GitHub page [1] which helped to verify the correctness of the computations. The script written for this replication also uses Python along with NumPy.

Methods

Model

The model is composed of three layers: sensory, association and motor or Q-value. The association layer has specialized memory units keeping trace of the sensory signal variation to build an internal state of the environment. By implementing the $SARSA(\lambda)$ algorithm, the model is capable of predicting the reward as a function of all its possible actions. Attentional feedback is used during learning, meaning only synapses participating in the decision are updated. All computations are done locally at the unit's level, which is a strong argument to present AuGMEnT as biologically plausible.

The initial intention was to implement the model using an artificial neural network simulator. The simulation tool ANNarchy [5] was considered for its ability to simulate rate-coded networks. Unfortunately, there were several incompatibilities with AuGMEnT. The fixed order of evaluation between entities, i.e. connections then populations, and the unspecified order of evaluation between different populations make it difficult to implement cascading evaluations. The use of ANNarchy was abandoned and it was instead decided to write a custom script to simulate the network.

The paper's description of the model details update functions for all populations and connections and is relatively straight forward to implement. Some informations are however missing for equation 17: the initial value for $q_a(t-1)$ is not provided and the article omits that $q_{a'}(t)$ needs to be set to 0 when receiving the end-trial signal, only mentioning *a δ that reflects the transition to the terminal state*. This information can actually be found in the 2012 conference paper about AuGMEnT [4]. Also, when it is said that *Q-values are set to zero* at the end of a trial, $q_a(t-1)$ is the only value which needs to be reset. It might also be useful to clarify the nature of the feedback weights w' in equations 14 and 16: once an action is selected, only the feedback synapses leaving the corresponding selected Q-value unit are activated to update tags, more precisely: $w'_{ij} = w_{ij} \times z_i$. The model could also have dedicated feedback connections but the simpler method is to use the feedforward synapses' weights.

To offer some discussion about the model and its limits, the first point to bring forward would be its artificial time management. The extreme discretization of time and explicit signals such as trial begin and end make it difficult to consider real-time simulation or even realistic environments implementations. These constraints became apparent when trying to use ANNarchy as it is not designed to work with large time steps. In fact, the authors have published a "continuous" version of AuGMEnT [7], as well as a "learning to reset" version [3] to address theses issues. As a whole, we have found these mechanisms for artificial time management rather misleading when reproducing the model, and providing ad hoc solutions, not robust nor generic enough. We would consequently recommend to use them with parsimony and only with a strong justification.

Another possible weakness worth noting is the ambiguity of some memory traces. Because the traces in memory units are defined as the sums of changes in input, there exist sequences the model would be incapable of distinguishing. For example, the sequences `((0, 0), (1, 0), (1, 1))` and `((0, 0), (0, 1), (1, 1))` have the same memory traces `(1, 1)`.

Tasks

The descriptions of the tasks used to test the network are somewhat minimal and it was necessary to refer to other resources for more informations. In this section, some details of implementation are exposed.

For the fixation tasks , i.e. saccade/anti-saccade and probabilistic decision making, the sequence of phases is as listed:

1. *Begin*: blank screen for one step.
2. *Fixation*: fixation point on screen for a maximum of 8 steps. Once the network has fixated the point, it has to maintain fixation for an additional step before moving on to the next phase with a potential reward. [1]
3. *Cues*: all visual cues are displayed (over several steps when there are multiple shapes).
4. *Delay*: only the fixation points is on screen for two steps.
5. *Go*: the screen appears blank for a maximum of 10 steps. The network has to choose a direction to look at, if it chooses the intended target, it is rewarded.
6. *End*: extra step to give the final reward and signal the end of the trial with a blank display.

Additional informations were found in the author's implementation of the saccade/anti-saccade task: once the network has fixated the point in the *Fixation* phase, it has to

[1] In the code, this phase is split in a *Wait* phase to wait for the network to fixate once and *Fixate* phase to ensure it maintains fixation.

maintain fixating until the *Go* signal when the screen turns off, otherwise the experiment is failed. Moreover, during the *Go* phase, the gaze can only be chosen once, if it is not the target, the trial fails.

The provided code did not implement the probabilistic decision making task but, fortunately, the original experiment's article [6] provided a more thorough methodology description. The shaping strategy for the probabilistic decision making task consists in gradually increasing the difficulty of the task. Table 3 in the article describes all 8 levels of difficulty. The column *# Input Symbols* is the size of the subset of shapes. The network is not presented all shapes immediately: first, the two shapes with infinite weights are used, then shapes with the smallest absolute weights are added as the difficulty increases. The column *Sequence Length* is the number of shapes shown during a trial. The more shapes there are on screen, the more difficult it is to determine which target should be chosen. A number of settings is randomized, such as which shapes should appear, their order of apparition, but also their locations around the fixation point. The first shape can appear in any of the 4 locations, the second in any of the remaining 3 locations, etc. If the total weight of the input symbols is infinite the corresponding target is guaranteed to give the reward. If the total weight is 0, meaning both targets are equally likely to be rewarded, the network can look in either direction for the trial to be successful, but only one random target gives a reward. Finally, the triangle and heptagon, shapes with infinite weights, cancel each other in the computation of the total weight.

Results

Implementations Comparison

As both codes use NumPy and identical data structures, they function in very similar way. The main difference is their structures since the replication extensively uses object oriented paradigms for readability. The computations in [1] variate in two points: the initial value of $q_a(t-1)$ is set to $q_{a'}(t)$ whereas the replication script simply uses 0 and the Q-values are rounded to 5 decimals. From our tests these modifications do not yield better results. The replication offers a 40% speedup, mostly from the way it handles bias weights: they are created alongside the units' activities instead of being concatenated before every computation. A profiling of the author's code indicates most of its execution time is spent inside the function *hstack*.

Replicated Data

Only the saccade/anti-saccade task and the probabilistic decision making task were implemented. For the probabilistic decision making task, the results are very similar. However, the saccade/anti-saccade task results are slightly worse than announced in the original article. The results presented in table 1 were obtained using the same parameters as in tables 1 and 2 of the reference article. *Success* designates the ratio of networks which successfully learned the task and *Convergence* the median number of trials necessary to learn it over 10,000 networks for saccade/anti-saccade and 100 networks for probabilistic decision making. Since the results are fairly sensible to the task's protocol, it is possible the differences for the saccade tasks come from undocumented changes in the experiments. Qualitative results such as the use of shaping strategy to obtain better performances are confirmed by this replication. See also figures 1 and 2 for the replicated activity traces of figures 2D and 4C in the reference article.

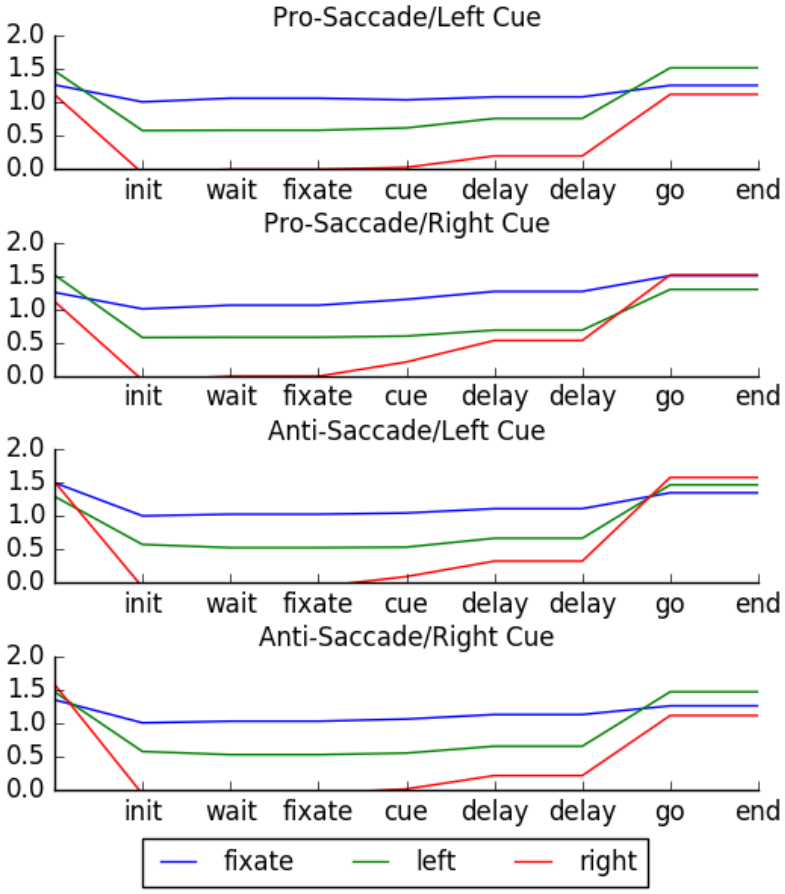

Figure 1: Q-value units' Activity for the Saccade/Anti-saccade Task (reproduction of figure 2D). The "Fixate" action dominates until the "Go" phase where the model correctly chooses the direction to look at. As in the reference article, there is a noticeable reaction after the "Cue" phase.

Table 1: Results

Task	Success in [2]	Success	Convergence in [2]	Convergence
Saccade with shaping	99.45%	90.55%	4100 trials	3970 trials
Saccade without shaping	76.41%	59.10%	-	4785 trials
Probabilistic decision	99.0%	100.0%	55234 trials	55988 trials

Conclusion

The results obtained are comparable to those announced in the article. Ambiguities in the experiments' descriptions could be the cause for worse performances, but do not contradict the article's overall conclusion.

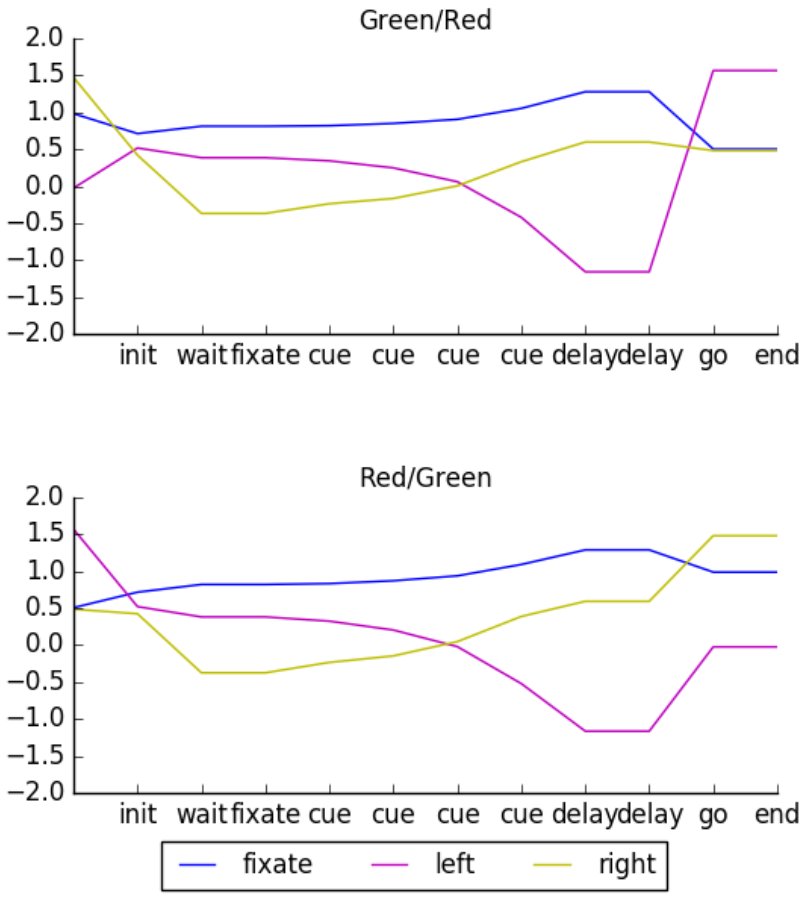

Figure 2: Q-value units' Activity for the Probabilistic Decision Making Task (reproduction of figure 4C). For both trials, the green target in the best choice. Once again, the model maintains fixation until the "Go" phase where it makes the correct decision.

References

[1] J. Rombouts. *Implementation of simple AuGMEnT network example*. URL: https://github.com/JRombouts/augment.

[2] J. Rombouts, S. Bohte, and P. Roelfsema. "How Attention Can Create Synaptic Tags for the Learning of Working Memories in Sequential Tasks". In: *PLoS Computational Biology* 11.3 (2015), e1004060. DOI: 10.1371/journal.pcbi.1004060.

[3] J. Rombouts, P. Roelfsema, and S. Bohte. "Learning Resets of Neural Working Memory". In: *22nd European Symposium on Artificial Neural Networks, Computational Intelligence And Machine Learning*. 2014. URL: http://www.i6doc.com/en/livre/?GCOI=28001100432440.

[4] J. Rombouts, P. Roelfsema, and S. Bohte. "Neurally Plausible Reinforcement Learning of Working Memory Tasks". In: *Advances in Neural Information Processing Systems 25*. 2012. URL: http://papers.nips.cc/paper/4813-neurally-plausible-reinforcement-learning-of-working-memory-tasks.

[5] J. Vitay, H. Dinklebach, and F. Hamker. "ANNarchy: a code generation approach to neural simulations on parallel hardware". In: *Frontiers Neuroinformatics* 9.19 (2015). DOI: 10.3389/fninf.2015.00019.

[6] T. Yang and M. Shalden. "Probabilistic reasoning by neurons". In: *Nature* 447.7148 (2007), pp. 1075–80. DOI: 10.1038/nature05852.

[7] D. Zambrano, P. Roelfsema, and S. Bohte. "Continuous-time on-policy neural Reinforcement Learning of working memory tasks". In: *International Joint Conference on Neural Networks* (2015). DOI: 10.1109/IJCNN.2015.7280636.