Real-Time Recurrent Reinforcement Learning Appendix

Derivation of update equations

Consider a CT-RNN that has N hidden states and I inputs, activation φ and a combined weight matrix $W \in \mathbb{R}^{N \times X}$ where $Z = I + N + 1$. Each neuron has a time-constant $\tau \in \mathbb{R}^N$ and the next state $h_{t+1} \in \mathbb{R}^N$ is computed as follows:

$$h_{t+1} = h_t + \frac{1}{\tau}\left(-h_t + \varphi(W\xi_t)\right) \qquad \xi_t = \begin{bmatrix} x_t \\ h_t \\ 1 \end{bmatrix} \in \mathbb{R}^Z$$

where x_t is the input at time t, and 1 concatenated to ξ_t accounts for the bias. The output $\hat{y}_t \in \mathbb{R}^O$ is given by a linear mapping $\hat{y}_t = W_{out} h_t$. The latent space follows the ODE $\tau \dot{h} = -h_t + \varphi(W\xi_t)$.

RFLO leverages the state-update expression in order to simplify the RTRL update. For this we expand the gradient of f in equation 6. Note that previous work on RFLO kept the time constant τ fixed and trained the recurrent weights W only, hence the restricted $\hat{J}^W$ in the equation:

$$\begin{aligned} \hat{J}^W_{t+1} &= \frac{d}{dW} h_t + \frac{d}{dW} f(x_t, h_t) \\ &= \frac{d}{dW} h_t + \frac{d}{dW} \frac{1}{\tau}\left(-h_t + \varphi(W\xi_t)\right) \\ &= \frac{d}{dW} h_t (1 - \frac{1}{\tau}) + \frac{d}{dW} \frac{1}{\tau} \varphi(W\xi_t) \\ &= (1 - \frac{1}{\tau})\, \hat{J}^W_t + \frac{1}{\tau} \nabla_W \varphi(W\xi_t) \\ &\quad + \frac{1}{\tau} \hat{J}^W_t \nabla_{h_t} \varphi(W\xi_t) \end{aligned} \tag{10}$$

In order to achieve biological plausibility, RFLO boldly drops the last summand, since it requires horizontal gradient communication. The partial derivative of the activation is simply $\nabla_W \varphi(W\xi_t) = \varphi'(W\xi_t)^\top \xi_t$ where φ' is the pointwise derivative of the activation function φ:

$$\hat{J}^W_{t+1} \approx (1 - \frac{1}{\tau})\, \hat{J}^W_t + \frac{1}{\tau} \varphi'(W\xi_t)^\top \xi_t \tag{11}$$

We analogously derive the RFLO update for τ. Again we drop the communicated gradients $\hat{J}^\tau_t \nabla_{h_t} \varphi(W\xi_t)$ and arrive at the expression for $\hat{J}^\tau$.

$$\begin{aligned} \hat{J}^\tau_{t+1} &= \hat{J}^\tau_t - \frac{d}{d\tau} \frac{1}{\tau} h_t + \frac{1}{\tau} \varphi(W\xi_t) \\ &= \hat{J}^\tau_t - \nabla_\tau \frac{1}{\tau} h_t - \hat{J}^\tau_t \nabla_{h_t} \frac{1}{\tau} h_t + \frac{d}{d\tau} \frac{1}{\tau} \varphi(W\xi_t) \\ &= \hat{J}^\tau_t (1 - \frac{1}{\tau}) + \frac{1}{\tau^2} h_t + \nabla_\tau \frac{1}{\tau} \varphi(W\xi_t) \\ &\quad + \hat{J}^\tau_t \nabla_{h_t} \frac{1}{\tau} \varphi(W\xi_t) \\ &= \hat{J}^\tau_t (1 - \frac{1}{\tau}) + \frac{1}{\tau^2} h_t - \frac{1}{\tau^2} \varphi(W\xi_t) \\ &\quad + \hat{J}^\tau_t \nabla_{h_t} \frac{1}{\tau} \varphi(W\xi_t) \end{aligned} \tag{12}$$

Description	Symbol	Value
number of neurons	n	32
discount factor	γ	0.99
Actor learning rate	α_A	1e-3
Critic learning rate	α_C	1e-3
RNN learning rate	α_R	1e-3
actor RNN trace scale	η_A	1.0
entropy rate	η_H	1e-5
Actor eligibility decay	λ_A	0.99
Critic eligibility decay	λ_C	0.99
RNN eligibility decay	λ_R	0.99
CT-RNN ODE timestep	dt	1.0
patience in epochs		20
maximum environment steps		50 mil.
optimizer		adam
batch size		1
learning rate decay		0
action epsilon		0
update period		1
gradient norm clip		1.0
normalize observations		False

Table 1: Hyperparameters of RTRRL

Implementation Details

Upon acceptance, we will publish our code on GitHub. Logging of experiments is implemented for Aim [1] or Weights & Biases [2] as backend. Our implementation is highly configurable and allows for many tweaks to the base algorithm. Available options include gradient clipping, learning rate decay, epsilon greedy policy, delayed RNN parameter updates and many more. Please refer to table 1 and the `Readme.md` in the code folder for a list of configurables. Table 1 summarizes the hyperparameters of RTRRL. We kept them at the listed default values for all our experiments.

Algorithm 2 repeats RTRRL with if-cases for RTRL without Feedback Alignment for demonstrative purposes. The algorithm can be divided into 4 distinct steps that are depicted in figure 7. When developing the algorithm, we had to figure out the proper order of operations. Figure 8 is a flowchart that was created to help understand at what point the eligibility traces are combined with the TD-error and approximate Jacobian, to form the parameter updates.

Since not using batched experiences, our algorithm unsurprisingly suffers from large variance and in some cases catastrophic forgetting ensues. Using an exponentially decaying learning rate for the RNN can help ins such cases but for simplicity we chose to make our experiments without this fix. Since a biologically plausible agent should retain its capability of reacting to shifts in the environment, the adaptability of our algorithm would be tainted as the learning rate of the RNN approaches 0. Nonetheless, RTRRL most of the time converges to an optimal solution without the use of decaying learning rates.

[1] https://aimstack.readthedocs.io/

[2] httpbs://wandb.ai/

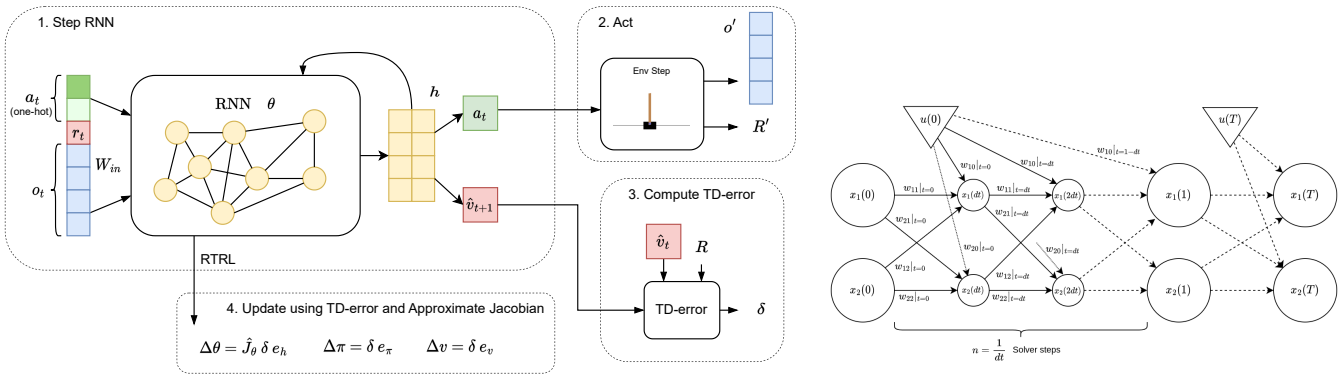

Figure 7: Left: RTRRL can be devided into 4 parts that are repeated throughout training. Right: CT-RNNs are solving an Ordinary Differential Equation. In general, any solver may be used. When using forward Euler, $dt = k^{-1}$ is a hyperparameter that determines the number of solver steps and subsequently the accuracy of the solution.

Neuron Model. The simplified CT-RNN outlined in the paper is taken from Murray (2019). Our code allows for increasing the number of steps $k = dt^{-1}$ when solving the underlying Ordinary Differential Equation with the forward Euler method. More steps lead to a more expressive model meaning you can get away with fewer neurons, but also to increased computational complexity. In our experiments we kept $k = 1$ for simplicity.

More Experiments

In Table 3 we summarize all our results for the experiments explained in the remainder of this section. The column "RFLO CT-RNN" corresponds to biologically plausible RTRRL as defined above, but we have also included results for RTRRL where RTRL is used in place of RFLO. The values presented in the table are the median and standard-deviation of the best evaluation episodes, throughout the training of 5 runs per environment and algorithm. As one can see, RTRRL performs best on average, followed by RTRRL with RTRL, as the second best algorithm.

In the remainder of this section we discuss some additional experiments we conducted that did not make it into the main manuscript of the paper.

Deep Exploration. Exploration versus exploitation is a trade-off, central to all agents learning a task online. The `DeepSea` environment included in `bsuite` (Osband et al. 2020) is tailored to benchmark the ability of RL algorithms to explore in unfavourable environments. The agent is required to explore to reach the goal position albeit receiving negative rewards when moving towards it. We give results for exponentially increasing task length.

The classic control environment `Acrobot`, too, requires extensive exploration. A double pendulum starting in hanging position is set into motion by controlling the middle joint. The outer segment has to be elevated above a certain height, located above the anchor of the inner segment. Figure 10 shows the median rewards of 5 runs over the number of environment steps. We find that RTRRL with RFLO finds a solution significantly faster than the other methods hinting at superior exploration of our RTRRL approach.

Reservoir Computing. The term reservoir computing refers to using a fixed RNN with trainable linear readout weights, sparing the hassle of exploding or vanishing gradients. Yamashita and Hamagami (2022) employed this technique for solving POMDPs. We conducted experiments to test whether optimizing the RNN in RTRRL is actually beneficial. By keeping the RNN fixed after initialization, training the linear actor and critic only, we create our own echo state network. Figure 9 right shows results for the `MemoryChain` environment. We conclude that training the RNN does improve performance.

Biological implausibility of BPTT

The three main objections for the plausibility of BPTT in biological neural networks are: First, the reliance on shared weights between the forward and the backward synapses; Second, reciprocal error-transport, which means propagating back the errors, without interfering with the neural activity (Bartunov et al. 2018); Third, the concern that BPTT requires storing long sequences of the exact activation for each cell (Lillicrap and Santoro 2019). We rephrase the premises of Bartunov et al. (2018) and Lillicrap and Santoro (2019) as follows. A biologically plausible learning algorithm has to be: (1) *Local*, up to some low-dimensional reward signal, (2) *Online*, computing parameter updates and the network outputs in parallel, and (3) *Without weight transport*, in its error computation, meaning that synapses propagating back error signals must not mirror the strength of forward synapses. We further elaborate on each of these requirements for obtaining biological plausibility in the following.

Fully online. The computation of updates should not depend on alternating forward and backward phases. Conceivably, such phases could be implemented using pacemaker neurons. However, there must not be any freezing of values, which occurs in BPTT. Furthermore, parallel streams of experience, such as in batched environments of modern DRLs, violate this constraint, as a biological agent can only interact with the singular environment in which it is situated.

No weight transport. Synapses that propagate back error signals cannot have their strength mirroring the strength of

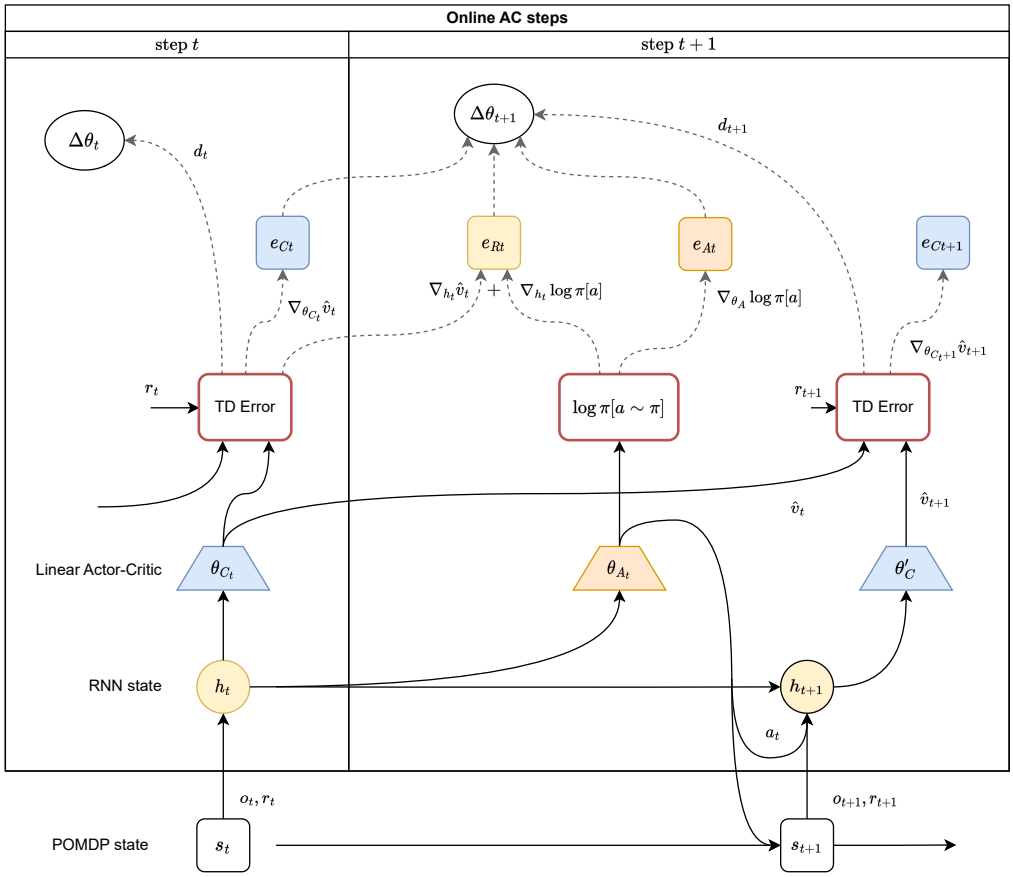

Figure 8: Flowchart depicting the computations done throughout one RTRRL step.

Model Environment	Linear TD(λ)	RTRL CTRNN	RFLO CTRNN	PPO CTRNN
CartPole-vel	454.55±34.47	500.00±0.00	500.00±0.00	112.11±40.38
CartPole-pos	57.47±1.38	178.57±93.04	500.00±163.17	70.76±11.42
MetaMaze	28.00±43.33	34.00±3.28	40.00±7.30	10.80±1.34
DiscountingChain	1.10±0.04	1.05±0.04	1.10±0.05	1.10±0.00
UmbrellaChain	0.79±0.05	0.99±0.15	1.29±0.07	1.04±0.12
BernoulliBandit	404.40±20.51	373.80±18.20	371.70±5.17	89.14±18.54
GaussianBandit	0.01±5.83	-126.17±70.60	24.05±21.43	0.00±0.00
Reacher	106.66±6.04	114.45±9.40	111.06±26.76	16.42±0.95
Swimmer	91.26±7.77	97.57±4.80	442.54±225.73	31.06±2.35
MountCarCont	-30.00±8.94	-40.00±5.48	-0.03±0.67	-1104.72±446.12
MountCarCont-vel	-41.88±16.54	-1.31±16.14	-20.00±40.80	-1079.70±204.52
MountCarCont-pos	-46.91±15.50	0.04±8.43	-59.54±43.45	-1071.67±40.27
MemoryChain-4	0.07±0.01	1.00±0.11	1.00±0.00	1.00±0.00
MemoryChain-8	0.07±0.02	0.90±0.25	1.00±0.00	1.00±0.00
MemoryChain-16	0.12±0.04	0.61±0.27	1.00±0.00	1.00±0.00
MemoryChain-32	0.17±0.03	0.56±0.13	0.77±0.18	1.00±0.00
MemoryChain-64	0.26±0.07	0.61±0.13	0.32±0.08	1.00±0.47
DeepSea-4	0.99±0.00	0.99±0.44	0.99±0.00	0.99±0.00
DeepSea-8	0.99±0.54	0.99±0.44	0.99±0.54	0.99±0.00
DeepSea-16	-0.00±0.54	0.99±0.54	0.99±0.44	0.00±0.00

Table 2: Summary of RTRRL experiments on Gymnax for networks with 32 neurons. Numbers reported are the median and standard deviation of the best validation reward achieved throughout training for 5 runs (larger is better). "RTRRL RFLO" denotes our biologically plausible version of RTRRL.

Model Environment	Linear TD(λ)	RTRRL RFLO	RTRRL LRU	PPO CTRNN
MetaMaze	55.93±43.33	61.60±9.24	15.80±5.02	38.79±27.03
DiscountingChain	1.09±0.02	1.10±0.00	1.10±0.00	1.32±0.00
UmbrellaChain	1.19±0.05	1.72±0.13	1.20±0.04	1.04±0.09
Catch	0.27±0.08	1.00±0.00	0.39±0.83	1.00±0.00
BernoulliBandit	393.38±20.51	893.96±11.35	807.96±100.19	883.44±18.72
GaussianBandit	nan±nan	61.10±32.98	37.78±67.01	0.00±0.00
PointRobot	1.34±0.21	5.30±4.30	0.98±0.49	1.04±0.57
Reacher	120.79±8.69	216.28±16.90	199.77±9.26	194.98±4.84
Swimmer	95.87±5.94	465.08±17.47	102.17±70.34	51.78±7.13
MemoryChain	0.08±0.01	1.00±0.00	nan±nan	1.00±0.00
DeepSea	0.99±0.00	0.99±0.00	0.99±0.00	0.99±0.00
StatelessCartPole	0.72±0.02	1.00±0.00	1.00±0.00	0.99±0.02
NoisyStatelessCartPole	0.50±0.01	0.99±0.01	0.93±0.04	0.59±0.03
Minesweeper	0.14±0.02	0.02±0.05	0.12±0.01	-0.14±0.05
MultiArmedBandit	0.25±0.05	0.39±0.03	0.38±0.08	0.19±0.10
RepeatFirst	0.15±0.12	-0.22±0.13	0.05±0.33	0.03±0.11
RepeatPrevious	-0.35±0.01	0.54±0.23	1.00±0.00	0.93±0.01

Table 3: Summary of RTRRL experiments including LRU experiments for networks with 32 neurons. Numbers reported are the median and standard deviation of the best validation reward achieved throughout training for 5 runs (larger is better). "RTRRL RFLO" denotes our biologically plausible version of RTRRL.

forward synapses. This is heavily violated by backpropagation since backward pathways need access to, and utilize, parameters that were used in the forward pass.

No horizontal gradient communication. Individual neurons x_k are very likely unable to communicate exact activation gradients $\nabla_{w_{ij}} x_k(t)$, with respect to the synaptic parameters w_{ij}, to other neurons that are not in their vicinity. In other words, intermediate (source) neurons should not be able to tell other (target) neurons how each of the synaptic weights of the source influence the dynamics of the target, nor vice-versa.

Biological Interpretation of RTRRL

Real-Time Recurrent Reinforcement Learning is a model of goal-directed behaviour learning and training of reflexes in animals, located in the human basal ganglia (Rusu and Pennartz 2020). It comprises a scalar-valued global reward signal that can be interpreted as dopaminergic synapses, a value estimator (critic) and motor output (actor).

Dopamine and Learning

Dopamine is a neurotransmitter found in the central nervous system of mammals. It is widely agreed upon that dopamine plays an important role in rewards and reinforcement. The molecule is released by specialized neurons located primarily in two areas of the brain: the substantia nigra zona compacta (SNc) and the ventral tegmental area (VTA). (Wise 2004) These neurons have large branching axonal arbors making synapses with many other neurons - mostly in the striatum and the pre-frontal cortex. Those synapses are usually located at the dendritic stems of glutamate synapses and can therefore effectively influence synaptic plasticity. (Sutton and Barto 2018)

Whenever receiving an unexpected reward, dopamine is released which subsequently reinforces the behavior that led to the reward. However, if the reward is preceded by a conditioned stimulus, it is released as soon as the stimulus occurs instead of when receiving the expected reward. If then the expected reward is absent, levels will drop below baseline representing a negative reinforcement signal. (Wise 2004) Various experiments have shown that dopamine is crucial for stamping in response-reward and stimulus-reward associations which in turn is needed for motivation when confronted with the same task in the future. Particularly, moderate doses of dopamine antagonists (neuroleptics) given to a live animal will reduce motivation to act. Habitual responses decline progressively in animals that are treated with neuroleptics. (Wise 2004)

Ternary Synapses

Hetero-synaptic plasticity is a type of synaptic plasticity that involves at least three different neurons. Usually, a sensory neuron is forming a synapse with a motor neuron. A third so called *facilitating* neuron also forms a synapse at the same spot and so is able to influence the signal transmission between the sensory and the motor neuron. A historic example is found in the gill-withdrawal reflex circuitry of Aplysia (Kandel 2013). The RPE signalling, facilitating inter-neuron strengthens the motor response (withdrawing the gill) when a negative reward (shock) is experienced or expected. RTRRL could be implemented in biological neural networks with ternary synapses where the facilitating synapses are projecting back from the TD-error computing inter-neuron.

Brain Structures

There is ample evidence that certain brain structures are implementing *actor-critic* methods. The striatum is involved

[3] https://commons.wikimedia.org/w/index.php?curid=85845448

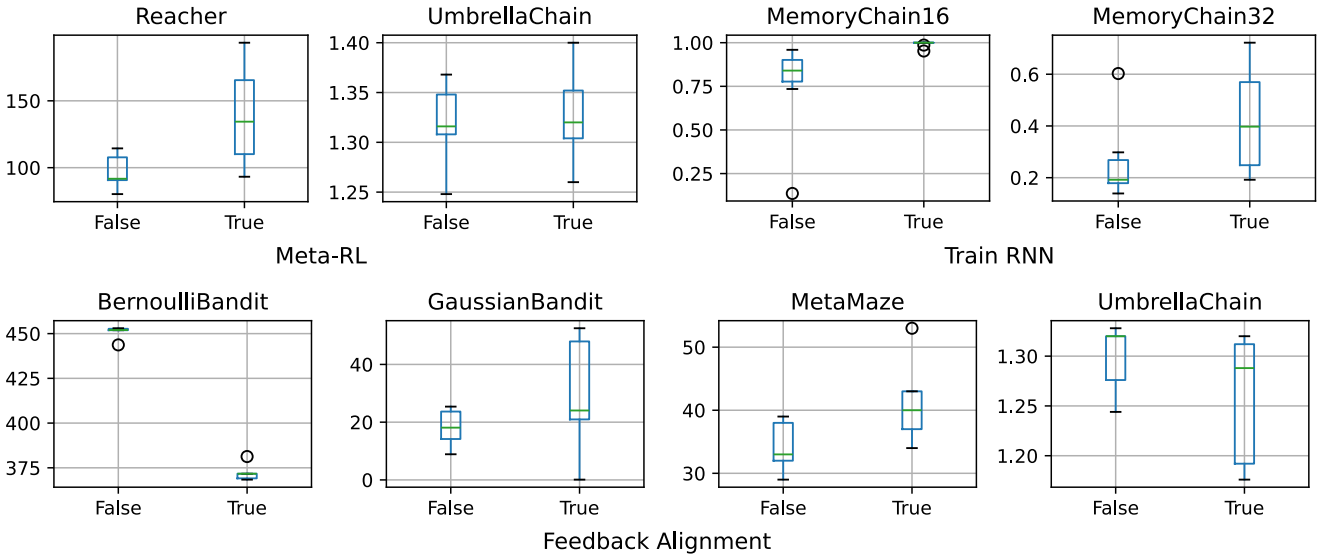

Figure 9: Left: While in most cases it does not make a difference, not using the Meta-RL architecture hampers performance in some cases. Right: Keeping the RNN frozen as in echo state networks leads to significantly worse results. Bottom: Comparing RTRRL with and without biologically plausible propagation of gradients using FA. Some environments such as `GaussianBandit` and `MetaMaze`, seem to benefit from it, while others do better with regular backpropagation.

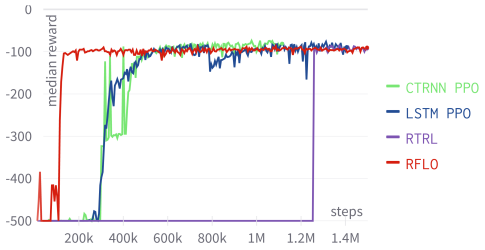

Figure 10: Median reward of 5 runs on `Acrobot`. The key to solving this environment is exploration. RNNs trained with RFLO solve it quicker than when trained with BPTT.

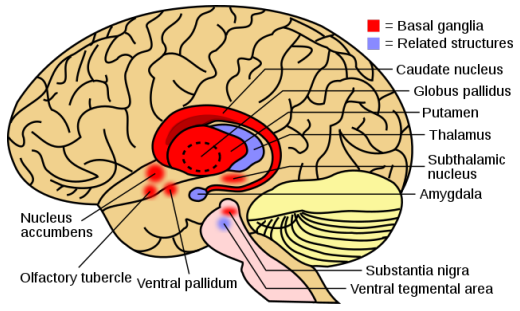

Figure 11: The basal ganglia are located at the base of the forebrain and play a major role in motivation and behavioural learning. Source: Wikimedia, Leevanjackson[3]

in motor and action planning, decision-making, motivation, reinforcement and reward perception. It is also heavily innervated by dopamine axons coming from the VTA and the SNc. It is speculated that dopamine release in the ventral striatum "energizes the next response" while it acts by stamping in the procedural memory trace in the dorsal striatum "establishing and maintaining procedural habit structures". (Wise 2004) Subsequently, the ventral striatum would correspond to the *critic* and the dorsal striatum to the *actor* of an RL algorithm (Sutton and Barto 2018). Dopamine would then correspond to the *TD-error* which is used to update both the *actor* and the *critic*. As dopamine neuron axons target both the ventral and dorsal striatum, and dopamine appears to be critical for synaptic plasticity, the similarities are evident. Furthermore, the *TD-error* and dopamine levels are both encoding the RPE: they are high whenever an unexpected reward is received and they are low (or negative in case of the *TD-error*) when an expected reward does not occur. (Sutton and Barto 2018) These similarities could be beneficial for RL as well as for Neuroscience as advances in either field could lead to new insights that are beneficial to the other.

Timing comparison

In order to compare wall-clock time, we did a quick performance test for RTRRL with CT-RNN and PPO with LSTM. In both cases we trained for 5000 steps, with 32 units and a batch size of 1, on a machine with a single GeForce RTX 2070 GPU. For both cases, we repeated the test 3 times and calculated the average time per step of the algorithm. Our results were 7,58 ms / step for PPO-LSTM and 7,49 ms / step for RTRRL-RFLO. Please note that these results may

Algorithm 2: Real-Time Recurrent Reinforcement Learning with precise backpropagation and entropy regularization

Require: Linear policy: $\pi(a|h, \theta_A)$
Require: Linear value-function: $\hat{v}(h, \theta_C)$
Require: CT-RNN body: RNN$([o, a, r], h, \hat{J}, \theta_R)$
$\theta_A, \theta_C, \theta_R \leftarrow$ Randomly initialize parameters
if RFLO **then**
  $B_A, B_C \leftarrow$ Randomly initialize feedback matrices
end if
$h, e_A, e_C, e_R \leftarrow \mathbf{0}$
$o \leftarrow$ Reset Environment
$h, \hat{J} \leftarrow$RNN$([o, \mathbf{0}, 0], h, \theta_R)$
$v \leftarrow \hat{v}(h, \theta_C)$
while not done **do**
  logits $\leftarrow \pi(h, \theta_A)$
  $a \leftarrow$ Sample(logits)
  $o, R \leftarrow$ Take action a in Environment
  $h', \hat{J}' \leftarrow$RNN$([o, a, r], h, \hat{J}, \theta_R)$
  $v' \leftarrow \hat{v}(h', \theta_C)$
  $\delta \leftarrow R + \gamma v' - v$
  $e_C \leftarrow \gamma \lambda_C e_C + \nabla_{\theta_C} \hat{v}$
  $e_A \leftarrow \gamma \lambda_A e_A + \nabla_{\theta_A} \ln \pi[a]$
  if RTRL **then**
    $e_R \leftarrow \gamma \lambda_R e_R + \nabla_{\theta_R} [\hat{v} + \eta_\pi \ln \pi[a]]$
    $g_{H,R} \leftarrow \eta_H \nabla_{\theta_R} H(\pi)$
  else if RFLO **then**
    $g_C \leftarrow B_C \mathbf{1}$
    $g_A \leftarrow B_A \nabla_h \ln \pi[a]$
    $e_R \leftarrow \gamma \lambda_R e_R + \hat{J}(g_C + \eta_A g_A)$
    $g_{H,R} \leftarrow \eta_H \hat{J} \nabla_h H(\pi)$
  end if
  $\theta_C \leftarrow \theta_C + \alpha_C \delta e_C$
  $\theta_A \leftarrow \theta_A + \alpha_A(\delta e_A + \eta_H \nabla_{\theta_A} H(\pi))$
  $\theta_R \leftarrow \theta_R + \alpha_R(\delta e_R + g_{H,R})$
  $v \leftarrow v', \quad h \leftarrow h', \quad \hat{J} \leftarrow \hat{J}'$
end while

not carry over to larger model or batch sizes.

Reproducibility Checklist

This paper:

Includes a conceptual outline and/or pseudocode description of AI methods introduced (**yes**) Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (**yes**) Provides well marked pedagogical references for less-familiare readers to gain background necessary to replicate the paper (**yes**)

Does this paper make theoretical contributions? (**no**)

Does this paper rely on one or more datasets? (**no**)

Does this paper include computational experiments? (**yes**)

If yes, please complete the list below.

Any code required for pre-processing data is included in the appendix. (**partial**). All source code required for conducting and analyzing the experiments is included in a code appendix. (**partial**) All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. (**yes**) All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (**partial**) If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results. (**yes**) This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks. (**partial**) This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. (**yes**) This paper states the number of algorithm runs used to compute each reported result. (**yes**) Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information. (**yes**) The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank). (**no**) This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments. (**yes**) This paper states the number and range of values tried per (hyper-) parameter during development of the paper, along with the criterion used for selecting the final parameter setting. (**partial**)