A spiking neural network model of reinforcement learning in the basal ganglia

Guilherme Miotto
Student of the MSc. in Computer Science at the University of Freiburg

ABSTRACT

It is widely accepted that synaptic plasticity is the underlying mechanism that makes learning possible. However, it is not clear how traditional Hebbian plasticity rules, like spike-timing-dependent plasticity (STDP), could allow learning by trial-and-error. The issue is a temporal one: while rules like STDP operate in the time scale of dozens of miliseconds, learning by trial-and-error requires making associations between action and action-outcome, and these two elements may occur seconds apart from each other. This problem is known as credit assignment or distal reward problem, and can be solved using three-factor plasticity rules. These rules employ the concepts of eligibility traces and chemical neuromodulators to determine which synapses should be reinforced or weakened at the moment of the action-outcome exposure.

In this work, we build and simulate a network of spiking neurons based on the striatum, the main input nucleus of the vertebrate basal ganglia. The striatum receives sensory excitatory inputs from the cortex and a dopaminergic modulatory signal from the Ventral Tegmental Area (VTA). This network is able to consistently learn cue-action associations in an instrumental conditioning task. It is also capable of adapting to action-reward contingency inversions.

This report is a description of the activities performed as a requirement for the 16 ECTS course **Master project in the research field neurorobotics**, during the summer semester of 2019.

KEYWORDS

Reinforcement learning
Dopamine
Plasticity
Spiking neurons
Simulation

INTRODUCTION

In the field of Computer Science, Reinforcement Learning (RL) is a set of Artificial Intelligence (AI) algorithms that defines an agent capable of taking actions in an environment to maximize rewards. The agent must be (at least partially) aware of the current state of the environment in order to evaluate the possible actions. If the agent is not exploring, it takes the action with the highest value and experiences the action-outcome. Based on the action-outcome, the agent can update its prediction mechanisms and repeat the process. As this loop is iterated over and over again, the actions of the agent become less random and more effective in maximizing reward. This kind of trial-and-error learning is also observed in biological systems.

In animals, the state representation can be divided into two parts. The animal's own internal state, for example, memories, hunger level, arousal level, etc., and the external state which is acquired via sensory stimuli. Based on the state representation, the animal then takes an action that is expected to maximize rewards (e.g. food, mating opportunities) or minimize punishments (e.g. pain, encounters with predators). The ability to learn by trial-and-error seems to be fundamental for animals to thrive in their natural habitats.

But also in more controlled settings, biological rein-

Manuscript compiled: Tuesday 24th September, 2019

[1]Corresponding author: guilherme.miotto@gmail.com

forcement learning can be observed. For instance, in instrumental conditioning experiments, animals can learn to associate certain cues (e.g. an odor or a light) to certain actions (e.g. pull a lever) if this specific cue-action pattern is paired with reward delivery (e.g. a food pellet) or punishment avoidance (e.g. turning off an electric shock). Such similarities between the "artificial" and the biological reinforcement learning beg the question of whether they operate by similar computations. And if so, where in the brain would those computations take place. Scientific investigation, especially during the last three decades, has begun to clarify this matter.

There is a good amount of evidence that the basal ganglia is highly involved in reinforcement learning at least for simple behaviors (Tai *et al.* (2012); Kravitz *et al.* (2012); Cox and Witten (2019); Kwak and Jung (2019); Neftci and Averbeck (2019)). The basal ganglia is a set of sub-cortical nuclei present in the brain of vertebrates that plays an important role in motor control and movement initiation (Graybiel *et al.* (1994); Mink (1996); Turner and Desmurget (2010)). The input nucleus of the basal ganglia is the striatum. The striatum receives massively convergent afferents from the cortex (Haber (2016)), therefore it is "aware" of the state of the environment. The output of the basal ganglia is mainly directed to the thalamus that is then forwarded back to the cortex. This cortico-basal-thalamo-cortico loop, that receives sensory information as inputs and outputs movement initiation signals, gives the basal ganglia a suitable structure to perform reinforcement learning.

Another key element of this system is the dopaminergic projection coming from the midbrain. Axons from the Ventral Tegmental Area (VTA) and Substantia Nigra pars Compacta (SNc) densely innervate the striatum and there they branch profusely. The main neurotransmitter released by those axons is dopamine (DA) which acts as a neuromodulator in the striatum. Since the late 90's, it is known that midbrain dopaminergic neurons encode reward prediction errors (RPE) (Schultz *et al.* (1997)); meaning that they fire above the baseline firing rate when the animal experiences an unexpected reward and, likewise, they fire bellow their baseline when an unpredicted aversive event happens. If something rewarding or aversive happens that was already predicted by the animal, DA neurons keep firing at their baseline. The RPE encoded by DA neurons is staggering similar to the time-difference error (TD-error) used in AI's RL. Therefore, this signal could be used to modulate plasticity of the cortico-striatal synapses to perform reinforcement learning.

However, traditional models of Hebbian plasticity, like spike-timing-dependent plasticity (STDP), does not account for a neuromodulator third-factor. In the traditional form of STDP, synaptic weights are adjusted based solely on the timing of the spikes of the pre and post-synaptic neurons. If the pre-synaptic neuron fires before the post-synaptic one, the synapse is potentiated. Conversely, the synapse is depressed if the post-synaptic neuron fires before the pre-synaptic one. But if the difference between the spike time of the neurons is greater than approximately 100ms, the change in the synaptic strength is negligible ((Bi and Poo 1998)). This limited time-window poses a challenge for RL: how to determine which synapses of the decision-making circuitry should be potentiated or depressed if the action-outcome can be experienced seconds after the decision was made? One possible solution is a three-factor plasticity rule with eligibility traces.

In the three-factor plasticity rule discussed in this study, synaptic weights are adjusted based not only on spike-timings but also on the extracellular concentration of a neuromodulator, more specifically, dopamine. According to this rule, the pairing pre and post-synaptic spikes will not be enough to drive synaptic weight changes, but it will raise the amount of eligibility trace of that synapse. The eligibility trace has a natural decay so that if it is not increased by spike pairing events, it will tend to zero. If the extracellular concentration of dopamine deviates from baseline, all synapses with a non-zero eligibility trace will have their weights adjusted. For the decision-making process, this translates as follows: First, the animal gets to a state where an action has to be taken. Then, it integrates sensory information and the internal state, both arriving from the cortex and landing on the striatum. The decision-making is conducted in the striatum and the corticostriatal synapses that were active in the process have their eligibility traces increased. During the next seconds, if the dopamine concentration in the striatum doesn't deviate from the baseline, no synaptic weight is changed. However, if the animal receives an unexpected reward, dopamine concentration increases and synapses with non-zero eligibility trace will be strengthened. Conversely, if the action-outcome was unexpected aversive, dopamine concentration will fall and the eligible synapses will be depressed. This system ensures that actions that resulted in unexpected rewards become more likely to be taken again in the future, while actions that resulted in punishments become less likely to be repeated.

While this theory of decision making in the striatum via STDP modulated by dopamine is appealing due to its simplicity and powerful implications, it was not clear until recently whether it was biologically plausible. However, an ever-growing body of evidence seems to support it. First of all, activation of dopamine neurons has been shown to lead to, for instance, operant self-stimulation (Rossi *et al.* (2013)), habit acquisition (Yin and Knowlton (2006); Graybiel and Grafton (2015)) and addiction (Kauer (2004)). Therefore, it seems clear that animals will learn to repeat behaviors that previously increased activity of dopamine neurons. Moreover, training animals on instrumental conditioning tasks was shown to selective increase synaptic weights of certain populations of cortico-striatal synapses (Xiong *et al.* (2015)). Also, stimulating distinct groups of cortico-striatal neurons was sufficient to bias the decision-making of rodents towards specific actions (Tai *et al.* (2012); Znamenskiy and Zador (2013)). Finally, plasticity of corticostriatal neurons in rodents was shown to be indeed modulated by dopamine (Pignatelli and Bonci (2015); Brzosko *et al.* (2019)), and this modulation seems to be based on an eligibility trace that allows maximum weight change for rewards arriving no later than two seconds after the spike pairing (Shen *et al.* (2008); Yagishita *et al.* (2014); Fisher *et al.* (2017); Shindou *et al.* (2019) but see Pawlak *et al.* (2010) and Gerstner *et al.* (2018) for reviews). Biological evidence of eligibility traces is a remarkable example of theoretical predictions guiding experimentalists to relevant findings.

In the present study, we simulate a network of spiking neurons that is capable of learning by trial-and-error in a biological plausible way. The task at hand is an instrumental conditioning protocol, involving two different sensory cues and two possible actions. The network is capable of learning the correct cue-action associations and to adapt when the action-reward contingency change.

METHODS

The network, being a generic theoretical model, was built to be agnostic towards the sensory modality and nature of the actions. However, to make the results presented here easier to relate, the task shown in Figure 1A was used to illustrate the kind of problem the network has to solve. In this task, a rat is trained in a two-choice forced action problem, known in the AI field as a two-arm bandit problem. The training process consists of a sequence of many trials. A trial starts with the presentation of an auditory cue; it could either be a high or a low pitch sound. Cue selection is done at random with 50-50 probability. After that, the rat has to communicate its decision by poking a snout-port either at the left-hand side or at the right-hand side. If the sound presented was a low tone and the rat chooses the left-hand side, then the animal is rewarded with a drop of water; it is also rewarded if the sound was high and the choice was the right-hand side. The other two possible combinations (low-right and high-left) are not rewarded. After some seconds a new trial starts.

The basic structure of the network of spiking neurons, here used as an abstraction for the rat's brain, is shown in Figure 1B. The cortical network is made of 1250 neurons, which are randomly connected to each other with a fixed in-degree of 125. From those neurons, 1000 are excitatory and 250 are inhibitory. Inhibitory synapses are 8 times stronger than the excitatory ones. The striatal network is made 200 hundred inhibitory neurons that connect to each other with a fixed in-degree of 20. The synaptic strength in the striatum is the same as the inhibitory neurons in the cortex. All synapses within the cortex and striatum are static. All neurons in the cortex and striatum are current-based leaky-integrate-and-fire neurons, with alpha-function shaped synaptic currents. Both cortex and striatum receive excitatory drive from Poissonian spike trains in order to bring the average firing rate of those networks to around 1 Hz, as observed *in-vivo* (Swadlow (1990); Hikosaka *et al.* (1989)). The excitatory population of the cortex connects to the striatum with a fixed in-degree of 100. These are the only plastic connections of the model. Initially, their synaptic weight is the same as other cortical excitatory synapses. Cortico-striatal synaptic plasticity is controlled by a three-factor rule modulated by dopamine concentration. The dopamine concentration is a consequence of the activity in the VTA, which, for simplicity, is modeled as a single neuron that fires once every 0.1 ms. Every-time the VTA neuron spikes, dopamine concentration increases by a fixed value, but this concentration also decays exponentially, therefore, the concentration tends to a quasi-constant baseline value. As long as the dopamine concentration stays at baseline levels, the weights of the network remain fixed. The only way that the dopamine concentration can deviate from the baseline is if the VTA changes its firing rate.

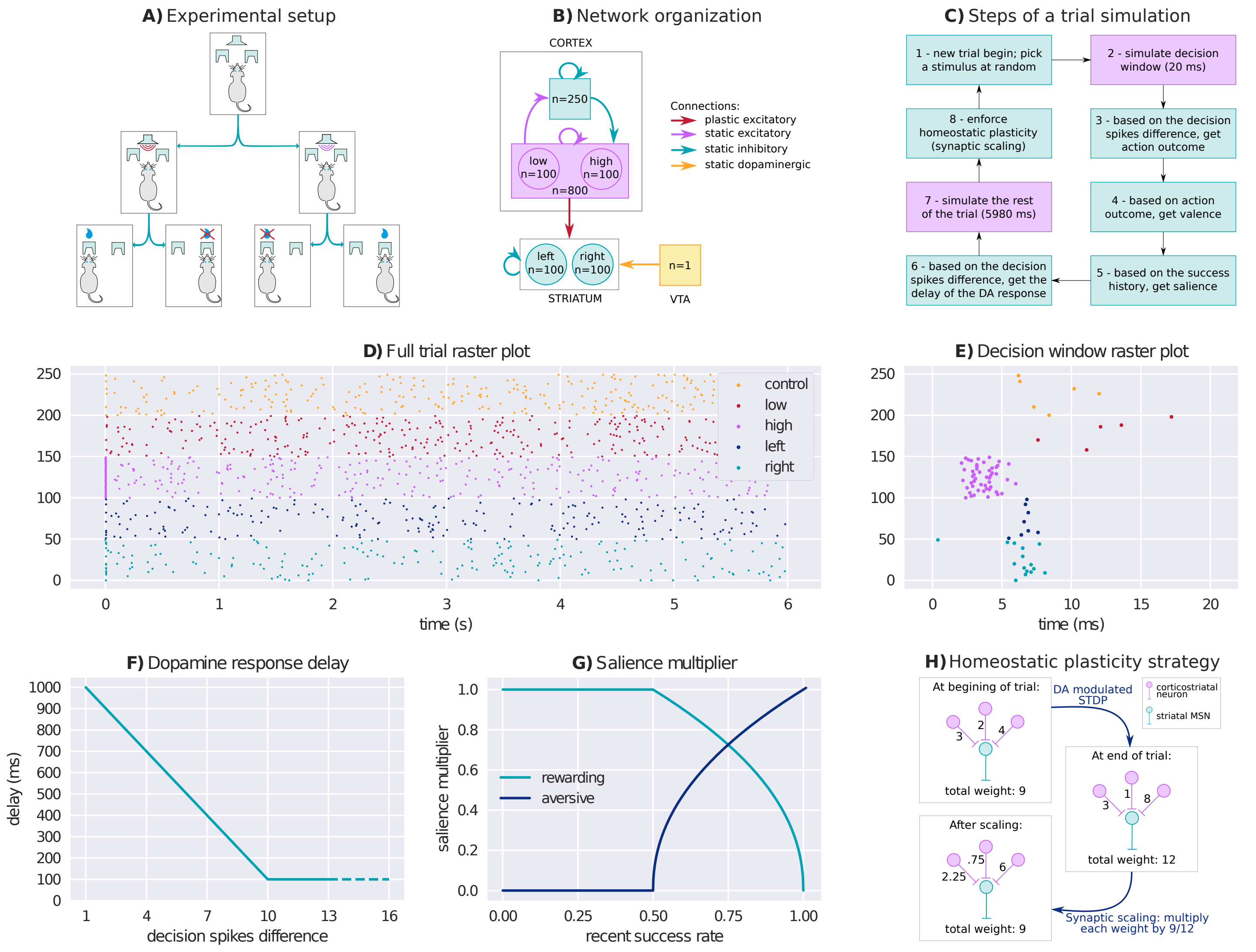
A) Experimental setup
B) Network organization
CORTEX
n=250
low n=100
high n=100
n=800
left n=100
right n=100
STRIATUM
n=1
VTA
Connections:
plastic excitatory
static excitatory
static inhibitory
static dopaminergic
C) Steps of a trial simulation
1 - new trial begin; pick a stimulus at random
2 - simulate decision window (20 ms)
3 - based on the decision spikes difference, get action outcome
4 - based on action outcome, get valence
5 - based on the success history, get salience
6 - based on the decision spikes difference, get the delay of the DA response
7 - simulate the rest of the trial (5980 ms)
8 - enforce homeostatic plasticity (synaptic scaling)
D) Full trial raster plot
control
low
high
left
right
time (s)
E) Decision window raster plot
time (ms)
F) Dopamine response delay
delay (ms)
decision spikes difference
G) Salience multiplier
salience multiplier
rewarding
aversive
recent success rate
H) Homeostatic plasticity strategy
At begining of trial:
total weight: 9
DA modulated STDP
corticostriatal neuron
striatal MSN
At end of trial:
total weight: 12
After scaling:
total weight: 9
Synaptic scaling: multiply each weight by 9/12

Figure 1 Methods. **A)** The task modeled: A rat is placed in a cage with two snout-poke ports. Next, an auditory cue is presented; it could be either a low tone or a high tone. Then, the rat must decide which port to poke. To get a reward, the rat has to poke the left port when the low tone is presented, or the right port when the high tone is presented. Other combinations do not provide any reward. **B)** Basic network structure: The network can be divided into three subnetworks: cortex, striatum and VTA. The cortex has 1250 neurons; 250 inhibitory and 1000 excitatory. From the 1000 excitatory neurons, 100 of them are sensitive to low tones and 100 sensitive to high tones. The striatum has 200 inhibitory neurons, 100 encoding the action go-left and 100 go-right. All neurons from cortex and striatum connect to each other with fixed indegree of 10% the size of the pre-synaptic population. Cortico-striatal connections are plastic, all the others are static. The VTA is modeled as a single neuron, whose firing rate regulates dopamine concentration. Dopamine concentration is broadcasted equally among all cortico-striatal synapses. **C)** Steps involved in the simulation of one trial. **D)** Raster plot of randomly selected neurons in a randomly selected trial. Samples of 50 neurons from 5 subpopulations are shown. Orange: cortex neurons that are not sensitive to any cue; Red: cortex neurons that are sensitive to low tones; Pink: cortex neurons that are sensitive to high tones; Blue: Striatum neurons encoding the action go-left; Red: Striatum neurons encoding the action go-right; In this example trial, the presented cue was a high tone and the action taken was go-right. **E)** Same raster plot as in D but zoomed into the decision window (first 20ms of the trial) **F)** Delay between end of the decision window and the disturbance of the VTA firing rate as a function of the difference between the number of spikes of the go-left and go-right populations. **G)** Salience of the dopaminergic response as a function of the success rate in case of taking rewarded actions (rewarding) and unrewarded actions (aversive). **H)** An example of how homeostatic plasticity is performed. At the end of every trial, each neuron in the striatum scale all its input synaptic weights by the same constant. This constant is the one that brings the sum of these weights back to the value it had at the beginning of the trial.

For this network to be able to interact with the environment it needs to be sensitive to sensory inputs and be able to take actions. Sensory stimuli are inputted at the cortex. From the 1000 excitatory cortical neurons 200 of them, picked at random, are sensitive to auditory stimuli: 100 to low-pitch and 100 to high-pitch sounds. Each trial starts with the auditory cue presentation. When the cue is presented, the corresponding population receives a direct current stimulation of 300 pA during 10 ms. This is sufficient to reliably make a great share of that population fire.

Actions are read from the striatum. From the 200 neurons in the striatum, half of them encode the action go-left and the other half the action go-right. The selected action is determined by the population that fires the most during the decision making time-window, which spans the first 20 ms of each trial. This is the same decision-making strategy used by Izhikevich (2007). Figure 1D shows a raster plot of an example trial and Figure 1E is a zoom of the first 20ms of this trial.

Depending on whether the action taken was rewarded or not, VTA firing rate momentarily increases or decreases. The change in firing rate starts just after some delay. This delay is variable and it represents the time to fulfill the action movements and experience the action outcome. This delay is proportional to the difference in the number of spikes of the striatal populations. If this difference is high, the delay is short. If the difference is small, the delay is long. This is to reflect the biological observation that the difference in activity of competing action-enconding populations is proportional to the action vigor (Gold and Shadlen (2007); Ding and Gold (2013)). In other words, if the competition between candidate actions is won by a large margin, the winner action is executed faster and the animal experiences the action-outcome sooner. The delay can vary between 1000ms, if the spike-count difference is just 1, and 100 ms, if the spike difference is equal or greater than 10, as shown in Figure 1F. After deciding when the action-outcome will be revealed, it is necessary to establish how salient the dopamine response will be, i.e. how strong the change in VTA's firing rate will be.

As mentioned before, the activity of the dopaminergic neurons of the VTA can encode RPEs. The RPE is the difference between the expected reward and the reward that was actually obtained. In the experiment modeled in this study (Figure 1A), RPEs tend to be large and positive during the first trials. This is because the animal has no reward expectations at this stage and all obtained rewards will be perceived as large positive surprises. However, reward expectations begin to increase as the animal gets trained on the task and the success rate gets larger than random chance (50%). Since the obtained rewards are not such surprises anymore, RPE decreases. Eventually, the animal will be certain that the reward will be delivered and the RPE for obtained rewards will be zero. Nevertheless, at this point, if the animal chooses the wrong action, the RPE will be very large but negative. This is because in this situation the expected reward is large, but the received reward was zero, therefore the RPE is large and negative. Positive RPEs are rewarding by nature, and animals tend to act to maximize them, negative RPEs are aversive and animals tend to avoid them.

It is out of the scope of this study to model the network of neurons that compute the RPE, therefore the activity of VTA is modeled here directly by equations. Equations 1 and 2 define the reward multiplier (r_m) and the aversion multiplier (a_m), receptively. As expected, both equations are direct functions of the success rate (s_r). They have a fixed behavior for success rates bellow 50% (naive rat; no expectations; doing just as well as random chance) and progressively change their values as the success rate gets higher than 50%. The success rate is calculated using the history of the last 31 trials. Both the length of the relevant history (31) and the exponent of the equations (0.4625) were determined by optimization, i.e. those are the values that resulted in the best performing networks. Finally, (r_m) and (a_m) are multiplied by 20 to arrive to how many extra/missing spikes should be added/subtracted to/from the VTA's regular activity. Equations 1 and 2 are shown graphically in Figure 1G.

$$r_m = \begin{cases} 1 & s_r \leq 0.5 \\ (2-2s_r)^{0.4625} & s_r > 0.5 \end{cases} \tag{1}$$

$$a_m = \begin{cases} 0 & s_r \leq 0.5 \\ (2s_r-1)^{0.4625} & s_r > 0.5 \end{cases} \tag{2}$$

Simulations were conducted using NEST v2.16 (Gewaltig and Diesmann (2007); Linssen *et al.* (2018)). Mathematical details about the dopamine modulated plasticity model and how it was implemented on NEST can be found in Izhikevich (2007) and Potjans *et al.* (2010).

RESULTS

Figure 2 summarizes the main results of this study. Figure 2G shows that the network is capable of learning the task even in the face of an action-reward contingency inversion. To get to about 95% of success rate, the network takes about 25 trials when untrained and about 50 trials after the reversal.

The underlying mechanism of learning in this network is the selective reinforcement of certain cortico-striatal synapses. Figure 2A shows the mean weight of the synapses between cortical neurons that are responsive to low-pitch sounds and the striatum. As training progresses, neurons from group this get biased towards increasing its connectivity to neurons from the go-left population. This makes the action go-left more likely after a low-tone cue is presented. Nevertheless, this connectivity preference is changed after contingency reversal. Figure 2B shows the same type of plot, but for cortical neurons that are responsive to high-tone sounds. The results are similar, the only difference is that the connectivity preference is inverted.

Not only the neurons of the low and high populations are involved in the decision making. In fact, all synapses between cortex and striatum are plastic and influence the action choice. But, because the remaining 800 cortical neurons are not directly sensitive to the cues, all the increased activity they get when a cue is presented comes from in-cortex recurrent connections with the low-tone and high-tone populations. Since a random network was used to model the cortex (no topology or feature-specific connectivity), these neurons receive on average the same amount of inputs from the low-pitch and high-pitch populations. Therefore, the overall effect is that the weights of these synapses remain roughly fixed, even though they are constantly being potentiated and depressed throughout the training, as shown in Figure 2C. However, it is possible to see in this Figure that the weights do not remain exactly fixed, but they decrease slowly. This is a consequence of the synaptic scaling mechanism, that depresses these synapses to "make room" for other synapses to be potentiated. This decay is not caused by the synaptic plasticity rule itself.

Figure 2E shows the absolute difference in the number of spikes of the striatal populations happening inside the decision window:

$$|spikes(left) - spikes(right)|_{dw}$$

It can be observed that this number increases as training progresses. This phenomena can be understood as the animal becoming surer of which action to take, and performing it faster (more vigor). Immediately after the reversal, this number drops considerably, because of the sudden increase in uncertainty.

Figure 2F shows the magnitude of RPE (i.e. the salience of the action-outcome) at each trial in case of success (rewarding) or failure (aversive). As explained before, in the beginning of the training, positive outcomes are very rewarding, but as the success rate increases, their become less rewarding. The opposite is observed for negative outcomes. At first, they are not aversive, but as the success rate increases, they become more aversive. When the performance is close to perfect, the rewarding salience is zero and the aversive salience is maximum. However, the network makes almost no mistakes anymore at this point, therefore, no RPE (rewarding or aversive) is experienced, and the dopamine concentration stays at its baseline. During this period, there is no change in synaptic weights as shown in Figures 2A and B.

DISCUSSION

In this study, a network of spiking neurons based on the interface between cortex and striatum was built. To overcome the distal reward problem, a three-factor plasticity rule was used. Even though all cortico-striatal synapses were plastic, selective potentiation of subsets of synapses emerged as a consequence of training. This allowed the network to progressively increase its probability of getting more rewards. The network was able to learn an instrumental conditioning task commonly used in experiments with rodents. Reversal learning was also tested with satisfactory results.

The instrumental conditioning task used in this study was a simple one. However, it involves fundamental building blocks of decision making, that in turn are essential for more complex cognitive processes. Moreover, this task can be the testbed for many learning abilities, like cue-action association, reversal learning, handling probabilistic outcomes, handling ambiguous sensory information (e.g. cloud of tones) and balancing exploration-exploitation. This task also allows easy performance comparisons with traditional RL algorithms like TD-learning and q-learning. Therefore, future studies could easily be performed using the already built model.

The plasticity rule that was used in this study was the one implemented on NEST (Izhikevich (2007); Pot-

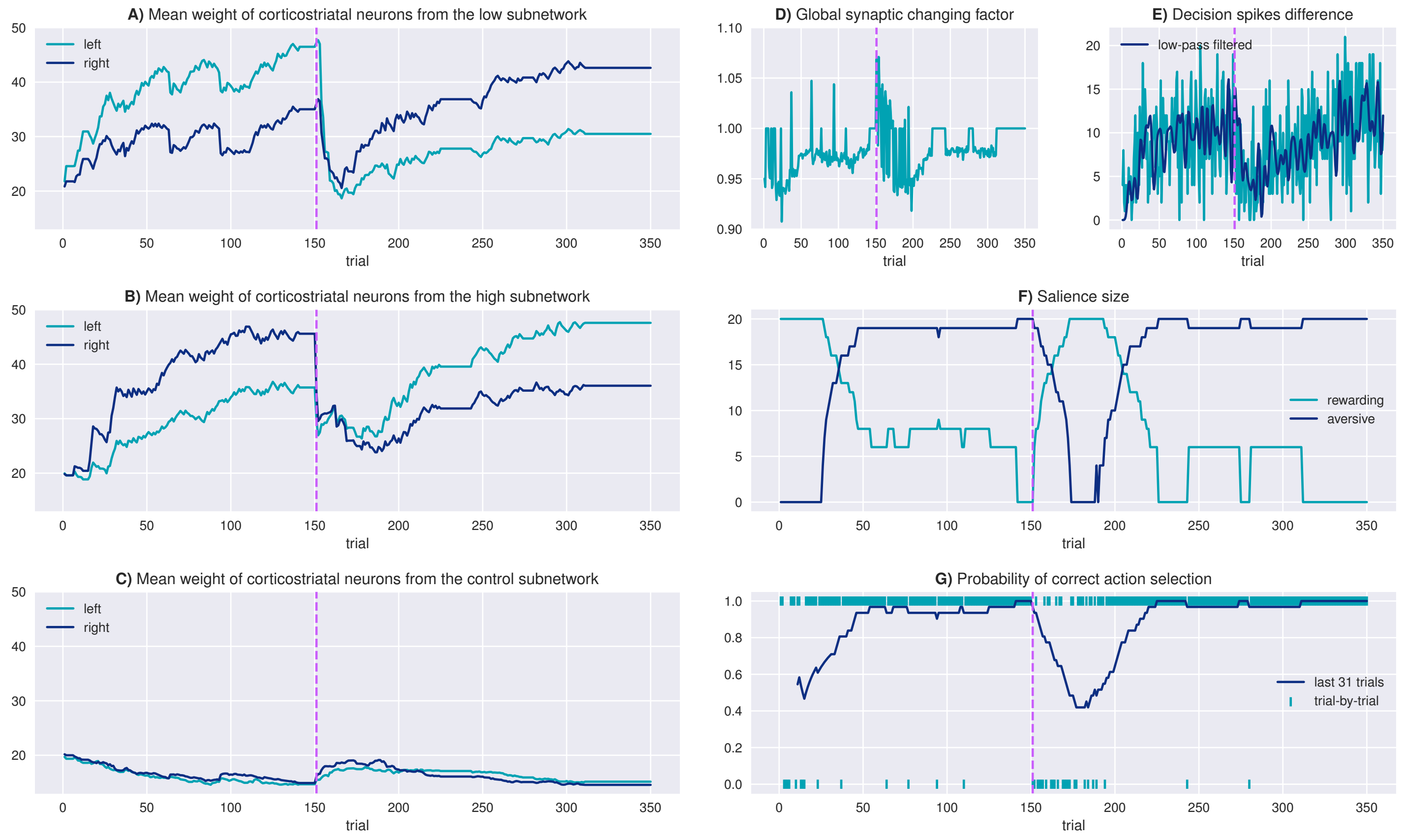

Figure 2 Results. The pink vertical line indicates the moment of action-reward contingency inversion **A)** Mean synaptic weight between cortical neurons that are sensitive to low-tone sounds and the striatum. **B)** Mean synaptic weight between cortical neurons that are sensitive to high-tone sounds and the striatum. **C)** Mean synaptic weight between cortical neurons that are not sensitive to sounds and the striatum. **D)** Ratio between the sum of all cortico-striatal weights before and after a trial. It gives an idea of how large are the changes in synaptic weight in every trial. **E)** Absolute difference between the number of spikes of the go-left and go-right striatal populations that occurred inside the decision window. **F)** Number of VTA spikes to be added to the baseline (rewarded trials – rewarding) or subtracted from the baseline (non-rewarded trials – aversive). **G)** Success history. The green vertical bars represents the result at each trial: 0 means failure and 1 means success. The dark blue line represents the success rate over the previous 31 trials.

jans *et al.* (2010)). This rule operates on the gating-principle, which means that weight changes just happen if dopamine concentration deviates from baseline; when the concentration is above baseline, normal STDP takes place (pre-before-post results in LTP and post-before-pre results in LTD), but when the concentration is below baseline, a reversed STDP happens (pre-before-post results in LTD and post-before-pre results in LTP). Although the gating principle is backed by some experiments (Yagishita *et al.* (2014); Fisher *et al.* (2017)), it is contradicted by other studies. For instance, a recent study (Shindou *et al.* (2019)) has shown that D1 neurons of the striatum undergo LTD at baseline dopamine concentrations when stimulated by pre-before-post protocol. But this LTD could be converted into LTP if a phasic increase of the dopamine concentration happened 2 seconds after the stimulation, raising the possibility that LTD itself could be an eligibility trace for D1 striatal neurons. Nevertheless, regardless of the specific role of dopamine in cortico-striatal plasticity, be it gating plasticity or converting LTD into LTP, from a computational neuroscience point-of-view what matters is that the dopamine concentration is capable of influencing the outcome of the synaptic weight change. Given that this feature is present, the influence of dopamine could be modeled in many different ways and the final computational results would be similar. In other words, the exact way that the dopamine influence is exerted, though important to be represented in future models, is not an essential requisite to demonstrate the computational capabilities of this kind of plasticity.

The network built in this study operates in a fundamentally different way from an AI RL algorithm (e.g. TD-learning), but it is still possible to compare some characteristics of those two. For instance, in AI RL, exploration is generally handled as an add-on rule and not by the algorithm itself. For example, the epsilon-greedy method will ocasionally overwrite the RL algorithm's desired action with a random one. On the other hand, exploration arises naturally in the presented network as a consequence of the random neural activity. Indeed, the learning process selectively potentiates certain groups of cortico-striatal synapses, but this is not enough to make action selection deterministic, it only increases the probability of choosing certain actions.

The network is capable of basic reversal learning, but animals use a more elaborated reversal mechanism that the network is not capable to reproduce. In the deterministic outcome scenario, as the one explored in this study, rats tend to decrease the amount of trials they need to learn the reversed action-reward contingency. Which means that after the first time the contingency is reversed, it can take a couple of trials for the animal to get back to its previous success rate. However, as training progresses, the number of "relearning trials" reduce and, eventually, a single non-rewarded trial is enough to make the animal flip its decision-making rule. This is a strong indication that the animal is relying on something more than altering the synaptic weights between cortical sensory inputs and striatal decision making populations. A hypothesis is that the animal expands its state representation with an extra feature: a latent variable representing the contingency currently being rewarded. Latent variables are not obtained directly via sensory input, instead, they are inferred by the animal. However, the network presented here can not learn any latent variable. As a consequence, every time a contingency reversal occurs, the network takes roughly the same time to relearn the new association[3]. This is a great deficiency compared to what is observed in animals.

How animals learn latent variables and compose their state representations is an open and active research question (Gershman *et al.* (2015); Tervo *et al.* (2016)), but some theories begin to emerge (Costa *et al.* (2015); Wilson *et al.* (2014)). Expanding the model of this study, so that it can dynamically handle state representations with latent variables, could be an interesting direction for further developments of this project.

ACKNOWLEDGMENTS

I am thankful to my advisors Prof. Dr. Stefan Rotter and Prof. Dr. Joschka Boedecker for the guidance and fruitful discussions. Also thanks to my colleagues Jijo George and Janine Laturner for helping me with IT issues and giving me ideas for code troubleshooting. This work would not be possible without the computer cluster NEMO, which is supported by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant INST 39/963-1 FUGG (bwForCluster NEMO).

LITERATURE CITED

Bi, G.-q. and M.-m. Poo, 1998 Synaptic modifications in cultured hippocampal neurons: dependence on

[3] This result is not explicitly shown. In Figure 2, just one reversal was performed.

spike timing, synaptic strength, and postsynaptic cell type. Journal of neuroscience **18**: 10464–10472.

Brzosko, Z., S. B. Mierau, and O. Paulsen, 2019 Neuromodulation of spike-timing-dependent plasticity: Past, present, and future. Neuron **103**: 563 – 581.

Costa, V. D., V. L. Tran, J. Turchi, and B. B. Averbeck, 2015 Reversal learning and dopamine: a bayesian perspective. Journal of Neuroscience **35**: 2407–2416.

Cox, J. and I. B. Witten, 2019 Striatal circuits for reward learning and decision-making. Nature Reviews Neuroscience p. 1.

Ding, L. and J. I. Gold, 2013 The basal ganglia's contributions to perceptual decision making. Neuron **79**: 640–649.

Fisher, S. D., P. B. Robertson, M. J. Black, P. Redgrave, M. A. Sagar, *et al.*, 2017 Reinforcement determines the timing dependence of corticostriatal synaptic plasticity in vivo. Nature communications **8**: 334.

Gershman, S. J., K. A. Norman, and Y. Niv, 2015 Discovering latent causes in reinforcement learning. Current Opinion in Behavioral Sciences **5**: 43–50.

Gerstner, W., M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, 2018 Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. Frontiers in neural circuits **12**.

Gewaltig, M.-O. and M. Diesmann, 2007 Nest (neural simulation tool). Scholarpedia **2**: 1430.

Gold, J. I. and M. N. Shadlen, 2007 The neural basis of decision making. Annual review of neuroscience **30**.

Graybiel, A. M., T. Aosaki, A. W. Flaherty, and M. Kimura, 1994 The basal ganglia and adaptive motor control. Science **265**: 1826–1831.

Graybiel, A. M. and S. T. Grafton, 2015 The striatum: where skills and habits meet. Cold Spring Harbor perspectives in biology **7**: a021691.

Haber, S. N., 2016 Corticostriatal circuitry. Neuroscience in the 21st Century pp. 1–21.

Hikosaka, O., M. Sakamoto, and S. Usui, 1989 Functional properties of monkey caudate neurons. i. activities related to saccadic eye movements. Journal of neurophysiology **61**: 780–798.

Izhikevich, E. M., 2007 Solving the distal reward problem through linkage of stdp and dopamine signaling. Cerebral cortex **17**: 2443–2452.

Kauer, J. A., 2004 Learning mechanisms in addiction: synaptic plasticity in the ventral tegmental area as a result of exposure to drugs of abuse. Annu. Rev. Physiol. **66**: 447–475.

Kravitz, A. V., L. D. Tye, and A. C. Kreitzer, 2012 Distinct roles for direct and indirect pathway striatal neurons in reinforcement. Nature neuroscience **15**: 816.

Kwak, S. and M. W. Jung, 2019 Distinct roles of striatal direct and indirect pathways in value-based decision making. eLife **8**.

Linssen, C., M. E. Lepperød, J. Mitchell, J. Pronold, J. M. Eppler, *et al.*, 2018 Nest 2.16.0.

Mink, J. W., 1996 The basal ganglia: focused selection and inhibition of competing motor programs. Progress in neurobiology **50**: 381–425.

Neftci, E. O. and B. B. Averbeck, 2019 Reinforcement learning in artificial and biological systems. Machine Intelligence p. 3.

Pawlak, V., J. R. Wickens, A. Kirkwood, and J. N. Kerr, 2010 Timing is not everything: neuromodulation opens the stdp gate. Frontiers in synaptic neuroscience **2**: 146.

Pignatelli, M. and A. Bonci, 2015 Role of dopamine neurons in reward and aversion: a synaptic plasticity perspective. Neuron **86**: 1145–1157.

Potjans, W., A. Morrison, and M. Diesmann, 2010 Enabling functional neural circuit simulations with distributed computing of neuromodulated plasticity. Frontiers in computational neuroscience **4**: 141.

Rossi, M. A., T. Sukharnikova, V. Y. Hayrapetyan, L. Yang, and H. H. Yin, 2013 Operant self-stimulation of dopamine neurons in the substantia nigra. PLoS One **8**: e65799.

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

Shen, W., M. Flajolet, P. Greengard, and D. J. Surmeier, 2008 Dichotomous dopaminergic control of striatal synaptic plasticity. Science **321**: 848–851.

Shindou, T., M. Shindou, S. Watanabe, and J. Wickens, 2019 A silent eligibility trace enables dopamine-dependent synaptic plasticity for reinforcement learning in the mouse striatum. European Journal of Neuroscience **49**: 726–736.

Swadlow, H. A., 1990 Efferent neurons and suspected interneurons in s-1 forelimb representation of the awake rabbit: receptive fields and axonal properties. Journal of Neurophysiology **63**: 1477–1498.

Tai, L.-H., A. M. Lee, N. Benavidez, A. Bonci, and L. Wilbrecht, 2012 Transient stimulation of distinct subpopulations of striatal neurons mimics changes in action value. Nature neuroscience **15**: 1281.

Tervo, D. G. R., J. B. Tenenbaum, and S. J. Gershman, 2016 Toward the neural implementation of structure learning. Current opinion in neurobiology **37**: 99–105.

Turner, R. S. and M. Desmurget, 2010 Basal ganglia contributions to motor control: a vigorous tutor. Current opinion in neurobiology **20**: 704–716.

Wilson, R. C., Y. K. Takahashi, G. Schoenbaum, and Y. Niv, 2014 Orbitofrontal cortex as a cognitive map of task space. Neuron **81**: 267–279.

Xiong, Q., P. Znamenskiy, and A. M. Zador, 2015 Selective corticostriatal plasticity during acquisition of an auditory discrimination task. Nature **521**: 348.

Yagishita, S., A. Hayashi-Takagi, G. C. Ellis-Davies, H. Urakubo, S. Ishii, *et al.*, 2014 A critical time window for dopamine actions on the structural plasticity of dendritic spines. Science **345**: 1616–1620.

Yin, H. H. and B. J. Knowlton, 2006 The role of the basal ganglia in habit formation. Nature Reviews Neuroscience **7**: 464.

Znamenskiy, P. and A. M. Zador, 2013 Corticostriatal neurons in auditory cortex drive decisions during auditory discrimination. Nature **497**: 482.

APPENDIX: SOURCE CODE

Following is the main part of the source code of the model presented here. The complete source code, including the plotting functions, can be found online at: https://github.com/gui-miotto/spiking-BG-RL. The most recent version of this report is available at this URL.

Listing 1 main.py

```
import sys, os
import numpy as np
from SpikingBGRL import Experiment

if __name__ == '__main__':
    # Build experiment
    exp = Experiment()

    # Make any tweaks here. For example:
    #exp.brain.vta.DA_pars['A_plus'] = .15 * exp.brain.vta.DA_pars['weight']

    # Run normal conditioning
    success_history = exp.train_brain(n_trials=150, save_dir='run1')
    result = np.sum(success_history[-100:])
    # Run reversal learning
    success_history = exp.train_brain(n_trials=150, rev_learn=True, save_dir='run1')
    result += np.sum(success_history[-100:])

    print('Successful␣trials:' result)
```

Listing 2 Experiment.py

```
import nest
import numpy as np
from mpi4py import MPI
from time import time
from datetime import timedelta
from .BrainStructures import Brain
from .DataIO import ExperimentResults, ExperimentMethods, NetworkSnapshot

class Experiment():
    """Class representing the instrumental conditioning of a brain. A experiment is sequence of
    trials. At each trial, a cue is presented to the brain and an action is taken by the brain.
    Class members whose names are followed by a trailing _ (e.g. self.success_) are updated at every
    trial, the others are constant throughout the whole experiment.
    """
    def __init__(self, seed=42, debug_mode=False):
        """Constructor

        Parameters
        ----------
        seed : int, optional
            Master seed for EVERYTHING. Runs with the same seed and number of virtual processes
            should yeld the same results. By default 42
        """
        self.debug = debug_mode

        # Experiment parameters
        self.trial_duration = 1100. if self.debug else 6000.  # Trial duration
        self.eval_time_window = 20. # Time window to check response via spike count
```

```python
        self.tail_of_trial = self.trial_duration - self.eval_time_window
        self.min_DA_wait_time = 100. # Minimum waiting time to reward
        self.max_DA_wait_time = 1000. # Maximum waiting time to reward
        self.warmup_magnitude = 1. if self.debug else 25. # The duration of the warmup period is
                                                           # given by warmup_magnitude * vta.tau_n

        # A random number generator (used to determine the sequence of cues)
        self.rng = np.random.RandomState(seed)

        # The brain to be trained
        scale = .2 if self.debug else 1.
        self.brain = Brain(master_seed=seed, scale=scale)
        self.brain_initiated = False

        #MPI rank (here used basically just to avoid multiple printing)
        self.mpi_rank = MPI.COMM_WORLD.Get_rank()
        self.rank0 = self.mpi_rank == 0


    def train_brain(self, n_trials=400, syn_scaling=True, aversion=True,
        rev_learn=False, baseline_only=False, full_io=True, save_dir='/tmp/learner'):
        """ Creates a brain and trains it for a specific number of trials.

        Parameters
        ----------
        n_trials : int, optional
            Number of trials to perform, by default 400
        syn_scaling : bool, optional
            If True, a homeostatic plasticity rule (synaptic scaling like) will be applied at the
            end of every trial, by default True
        aversion : bool, optional
            If True, taking wrong actions makes dopamine sink bellow the baseline. If False, taking
            wrong actions will keep dopamine concentrarion at baseline levels. By default True.
        rev_learn : bool, optional
            If True the stimuli/action association that results in reward is reversed, by default
            False
        baseline_only : bool, optional
            If True dopamine is kept at baseline levels regardless of the action taken, by default
            False
        full_io : bool, optional
            If False, there are no IOs to files and not essential MPI messages are not sent. Setting
            this variable to False is useful for tests and automated optimization processes that
            depend only on the success rate By default True.
        save_dir : str, optional
            Directory where the outputs will be saved (if full_io=True). Existing files will be
            overwritten. By default '/tmp/learner'

        Returns
        -------
        list[bool]
            A list with the success history
        """
        # Some handy variables
        color = {'red' : '\033[91m', 'green' : '\033[92m', 'none' : '\033[0m'}

        # Create brain and simulate a warmup
        if not self.brain_initiated:
            self._initiate_brain(full_io, save_dir)

        # Simulate trials
        trials_wall_clock_time = list()
        for trial in range(1, n_trials +1):
            self.trial_begin_ = nest.GetKernelStatus('time')
            if self.rank0:
                print(f'Simulating␣trial␣{trial}␣of␣{n_trials}:')

            # Adjust the amplitude of the dopamine bursts/dips
            self.brain.vta.adjust_salience_size(self.success_)

            # Simulate one trial and measure time taken to do it
            trial_start = time()
            self._simulate_one_trial(
                aversion=aversion, rev_learn=rev_learn, baseline_only=baseline_only)
            wall_clock_time = time() - trial_start
            trials_wall_clock_time.append(wall_clock_time)

            # Synaptic scaling
            if syn_scaling:
                self.brain.homeostatic_scaling(log_syn_change_factor=full_io)

            # Store experiment results on file(s):
            if full_io:
                self.brain.read_spike_detectors()
                self.brain.read_synaptic_weights()
                ExperimentResults(self).write(save_dir)
            self.brain.reset_spike_detectors()

            # Print some useful monitoring information
            n_suc = np.sum(self.success_)
            if self.rank0:
                print(f'Trial␣simulation␣concluded␣in␣{wall_clock_time:.1f}␣seconds')
                print(f'End-of-trial␣weight␣change:␣{self.brain.syn_change_factor_:.5f}')
                if self.success_[-1]:
                    print(f'{color["green"]}Correct␣action{color["none"]}')
                else:
                    print(f'{color["red"]}Wrong␣action{color["none"]}')
                print(f'{n_suc}␣correct␣actions␣so␣far␣({n_suc␣*␣100.␣/␣len(self.success_):.2f}%)')
                mean_wct = np.mean(trials_wall_clock_time)
                print(f'Average␣elapsed␣time␣per␣trial:␣{mean_wct:.1f}␣seconds')
                remaining_wct = round(mean_wct * (n_trials - trial))
                print(f'Expected␣remaining␣time:␣{timedelta(seconds=remaining_wct)}\n')

        if full_io:
            self.brain.store_network_snapshot()
            NetworkSnapshot(self).write(save_dir)

        return self.success_

    def _initiate_brain(self, full_io, save_dir):
        # Create the whole neural network
        if self.rank0:
            print('\nBuilding␣network')
        build_start = time()
        n_nodes = self.brain.build_local_network()
```

```python
        build_elapsed_time = time() - build_start

        # Write to file the experiment properties which are trial-independent
        if full_io:
            ExperimentMethods(self).write(save_dir)

        # Print build information
        warmup_duration = self.warmup_magnitude * self.brain.vta.tau_n
        if self.rank0:
            print(f'Building completed in {build_elapsed_time:.1f} seconds')
            print('Number of nodes:', n_nodes)
            print(f'Initial total plastic weight: {self.brain.initial_total_weight:,}')
            print(f'Simulating warmup for {warmup_duration} ms')

        # Simulate warmup
        warmup_start = time()
        syn_change = self.simulate_rest_state(
            duration=warmup_duration, reset_weights=True, return_change_factor=full_io)
        warmup_elapsed_time = time() - warmup_start

        # Print warmup statistics
        if self.rank0:
            print(f'Warmup simulated in {warmup_elapsed_time:.1f} seconds')
            print(f'Synaptic change during warmup: {syn_change:.5f}\n')

        # Some variable initiation
        self.success_ = list()
        self.brain_initiated = True


    def _simulate_one_trial(self, aversion, rev_learn, baseline_only):
        # Decide randomly what will be the next cue and do the corresponding stimulation
        self.cue_ = ['low', 'high'][self.rng.randint(2)]
        self.brain.cortex.stimulate_subpopulation(spop=self.cue_, delay=self.brain.dt)

        # Simulate evaluation window and count the resulting decision spikes
        self.brain.vta.set_drive(length=self.eval_time_window, drive_type='baseline')
        nest.Simulate(self.eval_time_window)
        decision_spikes = self.brain.striatum.count_decision_spikes()

        # Check if the action the correct one
        self.lminusr_ = decision_spikes['left'] - decision_spikes['right']
        if self.lminusr_ == 0:
            success = False
        else:
            success = (self.cue_ == 'low' and self.lminusr_ > 0) or \
                      (self.cue_ == 'high' and self.lminusr_ < 0)
            success = not success if rev_learn else success
        self.success_.append(success)

        # According to the action outcome, deliver the appropriate DA response
        if self.lminusr_ == 0 or baseline_only:  # just keep the baseline
            self.brain.vta.set_drive(length=self.tail_of_trial, drive_type='baseline')
        else:
            wait_time = self.max_DA_wait_time - (abs(self.lminusr_) - 1) * 100.
            wait_time = round(np.clip(wait_time, self.min_DA_wait_time, self.max_DA_wait_time))
            drive_type = 'rewarding' if success else 'aversive' if aversion else 'baseline'
            self.brain.vta.set_drive(
                length=self.tail_of_trial, drive_type=drive_type, delay=wait_time)

        # Simulate the rest of the trial
        nest.Simulate(self.tail_of_trial)


    def simulate_rest_state(self, duration=100., reset_weights=True, return_change_factor=True):
        """Simulates the network in its resting state, i.e.: no stimulus and under dopamine baseline
        levels. This function is used to simulate the warmup period and is a great debuging tool.

        Parameters
        ----------
        duration : float, optional
            Simulation duration, by default 100.
        reset_weights : bool, optional
            If true corticostriatal synapses will be set to it initial value after the simulation,
            by default True
        return_change_factor : bool, optional
            If True returns the synaptic change factor that happened during the simulation.
            by default True

        Returns
        -------
        [type]
            Synaptic change factor (i.e. the original total plastic weight divide by the total
            weight after simulation). Ideally should be as close to 1. as possible.
        """
        self.brain.vta.set_drive(length=duration, drive_type='baseline')
        nest.Simulate(duration)
        syn_change_factor = self.brain.get_total_weight_change() if return_change_factor else -1.
        self.brain.reset_spike_detectors()
        if reset_weights:
            self.brain.reset_corticostriatal_synapses()

        return syn_change_factor
```

Listing 3 BaseBrainStructure.py

```
import nest
import numpy as np

class BaseBrainStructure(object):
    # static numpy random number generators
    _py_rngs = None
    @property
    def py_rngs(self):
        return type(self)._py_rngs


    def __init__(self, scale=1):
        self.scale = scale  #TODO: make it static?
        self.N = dict()  # Number of neurons in each subpopulation
        self.neurons = dict()  # Neuron handles for each subpopulation
        self.spkdets = dict()  # Spike detectors
        self.events_ = dict()  # Events registered by the spike detectors
        self.grouped_synapses = list()  # A list of lists of plastic synapses grouped by target
```

```python
        self.plastic_weight_setpoint = None # Total plastic weight per target neuron - will be used
                                            # as homeostatic setpoint for each neuron


    def build_local_network(self):
        raise NotImplementedError('All brain scructures must implement build_local_network()')


    def initiate_membrane_potentials_randomly(self, v_min=None, v_max=None, pops=['ALL']):
        if v_min == None and v_max == None:
            neu_pars = nest.GetDefaults('default_neuron')
            v_min, v_max = neu_pars['V_reset'], neu_pars['V_th']

        for pop in pops:
            node_info = nest.GetStatus(self.neurons[pop])
            local_nodes = [(ni['global_id'], ni['vp']) for ni in node_info if ni['local']]
            for gid, proc in local_nodes:
                nest.SetStatus([gid], {'V_m': self.py_rngs[proc].uniform(v_min, v_max)})


    def read_spike_detectors(self):
        for pop, spkdet in self.spkdets.items():
            self.events_[pop] = nest.GetStatus(spkdet, 'events')[0]


    def reset_spike_detectors(self):
        for spkdet in self.spkdets.values():
            nest.SetStatus(spkdet, {'n_events' : 0 })


    def group_synapses_per_target(self, sources, targets, syn_model):
        local_gids = [ni['global_id'] for ni in nest.GetStatus(targets) if ni['local']]
        self.grouped_synapses = [nest.GetConnections(sources, [gid], syn_model) for gid in local_gids


    def homeostatic_scaling(self):
        # TODO: this loop is naturally parallelized if using mpi. Maybe it could be interesting to
        # paralelize this loop also for multithreading
        for syns in self.grouped_synapses:
            current_weights = np.array(nest.GetStatus(syns, 'weight'))
            scaling_factor = self.plastic_weight_setpoint / np.sum(current_weights)
            new_weights = scaling_factor * current_weights
            nest.SetStatus(syns, params='weight', val=new_weights)
```

Listing 4 Brain.py

```python
import nest, multiprocessing
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from itertools import product
from mpi4py import MPI
from copy import deepcopy

from .BaseBrainStructure import BaseBrainStructure
from .Cortex import Cortex
from .Striatum import Striatum
from .VTA import VTA



class Brain(BaseBrainStructure):
    """Abstraction of a trainable brain. A brain is made of a cortex, a striatum and a VTA. Synapses
    between those areas are handled by this class. Synapses between the cortex and striataum are
    excitatory, plastic and modulated by the dopamine of the VTA.
    """
    def __init__(self, master_seed, **args):
        super().__init__(**args)

        # Default neuron parameters
        tauSyn = .5  # synaptic time constant in ms
        self.neuron_params = {
                "C_m": 250.,  # capacitance of membrane in in pF
                "tau_m": 20.,  # time constant of membrane potential in ms
                "tau_syn_ex": tauSyn,
                "tau_syn_in": tauSyn,
                "E_L": 0.0,
                "V_reset": 0.0,
                "V_m": 0.0,
                "V_th": 20.,  # membrane threshold potential in mV
                }

        # Default synapse parameters
        self.J = 20.  # amplitude of excitatory postsynaptic current
        self.syn_delay = 1.5  # synaptic delay in ms
        self.syn_change_factor_ = -1.

        # MPI communication
        self.mpi_comm = MPI.COMM_WORLD
        self.mpi_rank = self.mpi_comm.Get_rank()
        self.mpi_procs = self.mpi_comm.Get_size()

        # Kernel parameters
        self.dt = .1
        self.verbosity = 20
        self.kernel_pars = {
            'print_time' : False,
            'resolution' : self.dt,
            'local_num_threads' : 1 if self.mpi_procs > 1 else multiprocessing.cpu_count(),
            'grng_seed' : master_seed,
            }

        # Define structures in the Brain
        self.cortex = Cortex(neu_params=self.neuron_params, J_E=self.J, scale=self.scale)
        self.striatum = Striatum(C_E=self.cortex.C['E'], J_I=self.cortex.J['I'], scale=self.scale)
        self.vta = VTA(dt=self.dt, J_E=self.J, syn_delay=self.syn_delay, scale=self.scale)
        self.structures = [self.cortex, self.striatum, self.vta]


    def _configure_kernel(self):
        # Internal random number generator (RNG) for NEST (i.e. used by the kernel)
        v_procs = self.mpi_procs * self.kernel_pars['local_num_threads']
        mid_seed = self.kernel_pars['grng_seed'] + 1 + v_procs
        self.kernel_pars['rng_seeds'] = range(self.kernel_pars['grng_seed'] + 1, mid_seed)

        # RNGs for the user (i.e used by these scripts)
        BaseBrainStructure.py_rngs = \
            [np.random.RandomState(seed) for seed in range(mid_seed, mid_seed + v_procs)]

        # Configure kernel
        nest.set_verbosity(self.verbosity)
        nest.ResetKernel()
        nest.SetKernelStatus(self.kernel_pars)


    def build_local_network(self):
        # Configure kernel and threads
        self._configure_kernel()

        # Create default neuron and synapse (will be used by the structures bellow)
        nest.CopyModel('iaf_psc_alpha', 'default_neuron', self.neuron_params)
        nest.CopyModel('static_synapse', 'default_synapse', {
            'delay' : self.syn_delay,
            'weight' : self.J,
            })

        # Create neurons from structures of the brain
        for struct in self.structures:
            struct.build_local_network()
            self.spkdets.update(struct.spkdets)
            #TODO: there shouldnt be repeated keys here. assure that

        # Connect cortex to striatum in a balanced way (We wouldnt need to be so careful if the
        # network was larger, because the chance of having big percentual differences in
        # connectivity between suppopulations would be smaller
        self.plastic_weight_setpoint = self.cortex.C['E'] * self.J
        self.initial_total_weight = self.cortex.C['E'] * self.J * self.striatum.N['ALL']
        for source, target in product(['low', 'high', 'E_no_S'], ['left', 'right']):
            nest.Connect(
                self.cortex.neurons[source],
                self.striatum.neurons[target],
                {'rule': 'fixed_indegree', 'indegree': self.cortex.C[source]},
                'corticostriatal_synapse'
                )
        self.group_synapses_per_target(  # For later use on synaptic scaling
            self.cortex.neurons['E'], self.striatum.neurons['ALL'], 'corticostriatal_synapse')

        # Get connections for later weight monitoring
        self.w_ind = ['low', 'high', 'E_rec', 'E']
        self.w_col = ['left', 'right', 'ALL']
        self.synapses = pd.DataFrame(index=self.w_ind, columns=self.w_col)
        self.weights_count = deepcopy(self.synapses)  # number of synapses
        self.weights_mean_ = deepcopy(self.synapses)  # average weight
        self.weights_hist_ = deepcopy(self.synapses)  # weight histogram
        for source, target in product(self.w_ind, self.w_col):
            cnns = nest.GetConnections(self.cortex.neurons[source], self.striatum.neurons[target])
            self.synapses.loc[source, target] = cnns
            self.weights_count.loc[source, target] = len(cnns)

        # Return total number of nodes in the network
        return nest.GetKernelStatus('network_size')


    def read_synaptic_weights(self):
        for source, target in product(self.w_ind, self.w_col):
            weights = nest.GetStatus(self.synapses.loc[source, target], 'weight')
            self.weights_mean_.loc[source, target] = np.mean(weights)
            self.weights_hist_.loc[source, target] = np.histogram(
                weights, bins=21, range=(0., self.vta.DA_pars['Wmax']))[0]


    def reset_corticostriatal_synapses(self):
        nest.SetStatus(self.synapses.loc['E', 'ALL'], params='weight', val=self.J)


    def homeostatic_scaling(self, log_syn_change_factor):
        if log_syn_change_factor:
            self.syn_change_factor_ = self.get_total_weight_change()
        super().homeostatic_scaling()


    def get_total_weight_change(self):
        """Calculates how much the total sum of plastic weights has increased/decreased in relation
        to the initial (before trials) values. This helps monitoring network explosions/implosions.
        (This function uses MPI communication.)

        Returns
        -------
        [float]
            initial_total_weight / current_total_weight ratio
        """
        weights = nest.GetStatus(self.synapses.loc['E', 'ALL'], 'weight')
        total_weight = np.sum(weights, dtype='f')
        recvbuf = np.empty(self.mpi_procs, dtype='f') if self.mpi_rank == 0 else None
        self.mpi_comm.Gather(total_weight, recvbuf, root=0)
        if self.mpi_rank == 0:
            total_weight = np.sum(recvbuf)
            change_factor = self.initial_total_weight / total_weight
        else:
            change_factor = None
        change_factor = self.mpi_comm.bcast(change_factor, root=0)
        return change_factor


    def store_network_snapshot(self):
        """ Reads and stores all synaptic weights individually
        """
        all_neurons = self.cortex.neurons['ALL'] + self.striatum.neurons['ALL']
        local_neurons = [ni['global_id'] for ni in nest.GetStatus(all_neurons) if ni['local']]
        self.snapshot_ = list()
        for cnn in nest.GetConnections(all_neurons, local_neurons):  # synapse info is stored in the
            cnn_info = nest.GetStatus([cnn])[0]                       # post synaptic side
            self.snapshot_.append({
                'source' : cnn_info['source'],
                'target' : cnn_info['target'],
                'weight' : cnn_info['weight'],
                'delay' : cnn_info['delay'],
            })
```

Listing 5 Cortex.py

```
import math, nest
from .BaseBrainStructure import BaseBrainStructure


class Cortex(BaseBrainStructure):
    def __init__(self, neu_params, J_E, **args):
        super().__init__(**args)

        # Number of neurons
        self.N['I'] = int(250 * self.scale)  # number of inhibitory neurons
        self.N['E'] = 4 * self.N['I']  # number of excitatory neurons
        self.N['E_rec'] = self.N['I_rec'] = int(100 * self.scale)  # num of neurons to record from
        self.N['low'] = self.N['high'] = int(100 * self.scale)  # subpops associated to stimuli
        self.N['E_no_S'] = self.N['E'] - self.N['low'] - self.N['high']
        self.N['ALL'] = self.N['I'] + self.N['E']

        # Connectivity
        epsilon = .1  # connection probability
        self.C = {pop : int(epsilon * n) for pop, n in self.N.items()} # num synapses per neuron

        # Synapse parameters
        g = 8.  # ratio inhibitory weight/excitatory weight
        self.J = {'E' : J_E} # amplitude of excitatory postsynaptic current
        self.J['I'] = -g * self.J['E']  # amplitude of inhibitory postsynaptic current

        # Background firing rate
        eta = .88  # external rate relative to threshold rate
        nu_th = neu_params['V_th'] * neu_params['C_m']
        nu_th /= self.J['E'] * self.C['E'] * math.e * neu_params['tau_m'] * neu_params['tau_syn_ex']
        nu_ex = eta * nu_th
        self.bg_rate = 1000.0 * nu_ex * self.C['E']

        # Stimulation protocol
        self.stim_duration = 10. #3.  #ms
        self.stim_intensity = 300.  #pA


    def build_local_network(self):
        # Create neurons
        for pop in ['E', 'I']:
            self.neurons[pop] = nest.Create('default_neuron', self.N[pop])

        # Sample subpopulations
        cut = 0
        for pop in ['low', 'high', 'E_rec']:
            self.neurons[pop] = self.neurons['E'][cut:cut+self.N[pop]]
            cut += self.N[pop]
        self.neurons['I_rec'] = self.neurons['I'][:self.N['I_rec']]
        self.neurons['E_no_S'] = tuple(
            set(self.neurons['E']) - set(self.neurons['low']) - set(self.neurons['high']))
        self.neurons['ALL'] = self.neurons['E'] + self.neurons['I']

        # Connect subpopulations to spike detectors
        for pop in ['low', 'high', 'E_rec', 'I_rec']:
            self.spkdets[pop] = nest.Create('spike_detector')
            nest.Connect(self.neurons[pop], self.spkdets[pop])

        # Connect neurons with each other
        for pop in ['E', 'I']:
            syn_model_name = f'cortex_{pop}_synapse'
            nest.CopyModel('default_synapse', syn_model_name, {"weight": self.J[pop]})
            conn_params = {'rule': 'fixed_indegree', 'indegree': self.C[pop]}
            nest.Connect(self.neurons[pop], self.neurons['ALL'], conn_params, syn_model_name)

        # Create and connect background activity
        background_activity = nest.Create('poisson_generator', params={"rate": self.bg_rate})
        nest.Connect(background_activity, self.neurons['ALL'], syn_spec='cortex_E_synapse')

        # initiate membrane potentials
        self.initiate_membrane_potentials_randomly()

        # Create and connect sensory stimulus
        self.stimulus = dict()
        for pop in ['low', 'high']:
            self.stimulus[pop] = nest.Create('step_current_generator')
            nest.Connect(self.stimulus[pop], self.neurons[pop])

    def stimulate_subpopulation(self, spop, delay=0.):
        stim_onset = nest.GetKernelStatus()['time'] + delay
        nest.SetStatus(self.stimulus[spop], params={
            'amplitude_times' : [stim_onset, stim_onset + self.stim_duration],
            'amplitude_values' : [self.stim_intensity, 0.],
        })
```

Listing 6 Striatum.py

```
import nest
import numpy as np
from itertools import product
from mpi4py import MPI
from .BaseBrainStructure import BaseBrainStructure

class Striatum(BaseBrainStructure):
    """ Abstraction of a striatum. Contains just inhibitiony neurons mutually connected randomly
    with constant indegree. Can be divided into two subpopulations. Connections within a
    subpopulation can have greate (i.e. less negative) weights than those across
    subpopulations. Class members whose names are followed by a trailing _
    (e.g. self.firing_rates_) are updated at every trial, the others are constant throughout
    the whole experiment.
    """
    def __init__(self, C_E, J_I, **args):
        super().__init__(**args)

        # Number of neurons
        #n = int(1.25 * C_E)  # neurons per subpopulation
        n = int(100 * self.scale)
        self.N['left'] = self.N['right'] = n
        self.N['ALL'] = self.N['left'] + self.N['right']

        # Connectivity
        epsilon = .1  # connection probability
        self.conn_params = {'rule': 'fixed_indegree', 'indegree': int(epsilon * n)}

        # synapse parameters
        self.w = 0. # deviation between strength of inter and intra−subpopulation synapses
        self.J_inter = J_I * (1. + self.w)  # weight between neurons of distinct sub populations
        self.J_intra = J_I * (1. − self.w)  # weight between neurons of the same sub populations;

        # Background activity
        self.bg_rate = 7950.

        # MPI communication
        self.mpi_comm = MPI.COMM_WORLD
        self.mpi_rank = self.mpi_comm.Get_rank()
        self.mpi_procs = self.mpi_comm.Get_size()

    def build_local_network(self):
        # Create neurons and connect them to spike detectors
        for pop in ['left', 'right']:
            self.neurons[pop] = nest.Create('default_neuron', self.N[pop])
            self.spkdets[pop] = nest.Create('spike_detector')
            nest.Connect(self.neurons[pop], self.spkdets[pop])
        self.neurons['ALL'] = self.neurons['left'] + self.neurons['right']

        # Connect neurons to each other
        nest.CopyModel('default_synapse', 'striatum_intra_syn', {"weight": self.J_intra})
        nest.CopyModel('default_synapse', 'striatum_inter_syn', {"weight": self.J_inter})
        for origin, target in product(['left', 'right'], ['left', 'right']):
            syn_model = 'striatum_intra_syn' if origin == target else 'striatum_inter_syn'
            nest.Connect(self.neurons[origin], self.neurons[target], self.conn_params, syn_model)

        # Create and connect background activity
        background_activity = nest.Create('poisson_generator', params={"rate": self.bg_rate})
        nest.Connect(background_activity, self.neurons['ALL'], syn_spec='cortex_E_synapse')

        # initiate membrane potentials
        self.initiate_membrane_potentials_randomly()

    def count_decision_spikes(self):
        dec_spk = [nest.GetStatus(self.spkdets[pop], 'n_events')[0] for pop in ['left', 'right']]
        dec_spk = np.array(dec_spk, dtype='i')
        recvbuf = np.empty([self.mpi_procs, 2], dtype='i') if self.mpi_rank == 0 else None
        self.mpi_comm.Gather(dec_spk, recvbuf, root=0)
        if self.mpi_rank == 0:
            recvbuf = np.sum(recvbuf, axis=0)
            decision_spikes = {pop : recvbuf[it] for it, pop in enumerate(['left', 'right'])}
        else:
            decision_spikes = dict()
        decision_spikes = self.mpi_comm.bcast(decision_spikes, root=0)
        return decision_spikes
```

Listing 7 VTA.py

```
import nest
import numpy as np
from .BaseBrainStructure import BaseBrainStructure


class VTA(BaseBrainStructure):
    def __init__(self, dt, J_E, syn_delay, **args):
        super().__init__(**args)

        self.dt = dt  # simulation timestep
        self.N['ALL'] = 1  # Number of neurons. No need to scale here

        # Dopamine modulation parameters
        self.tau_n = 200.
        self.DA_pars = {
            'weight' : J_E,  # Default 1.
            'delay': syn_delay, # Default 1.; Synaptic delay
            'tau_n' : self.tau_n, # Default 200.; Time constant of dopaminergic trace in ms
            'b' : 1. / self.dt,  # Default 0.; Dopaminergic baseline concentration
            'n' : 1. / self.dt + 2.5 / self.tau_n, # Default 0.; Initial dopamine concentration
            'A_plus' : .1993088006 * J_E,  # Default 1.; Amplitude of weight change for facilitation
            'A_minus' : .102861686 * J_E,  # Default 1.5; Amplitude of weight change for depression
            'Wmax' : 3.8622647000698906 * J_E, # Maximal synaptic weight
            #'tau_c' : 1000., # Default 1000.,  # Time constant of eligibility trace in ms
            #'tau_plus' : 20.0, #  Default 20.; STDP time constant for facilitation in ms
            #'Wmin' : 0., # Default 0. # Minimal synaptic weight
            #'vt' : volt_DA[0], # Volume transmitter will be assigned later on
            }
        self.max_salience = 20  # integer greater than 0. Number of spikes added or subtracted to
                               # the baseline in the face of rewarding or aversive events
                               # (respectively)
        self.reward_size_ = self.max_salience
        self.aversion_size_ = 0
        self.memory = 31  # How many trials is taken into account to calculate the salience size
        self.degree = .4625462336213272


    def adjust_salience_size(self, success_history):
        recent_successes = np.sum(success_history[-self.memory:])
        failure_rate = (self.memory - recent_successes) / self.memory
# TODO: make it dependent on
                                                                      # the success rate, as in the
                                                                      # report

        # If failure rate is no better than chance, use max_salience and zero
        if failure_rate >= .5:
            self.reward_size_ = self.max_salience
            self.aversion_size_ = 0
        # Otherwise adjust salience gradually proportianally to the failure rate
        else:
            reward_mult = (2. * failure_rate) ** self.degree
            aversi_mult = (1. - 2. * failure_rate) ** self.degree
            self.reward_size_ = round(reward_mult * self.max_salience)
            self.aversion_size_ = round(aversi_mult * self.max_salience)
            # round() returns an integer if ndigits is omitted


    def build_local_network(self):
        # Create nodes
        self.drive = nest.Create('spike_generator')  # Spike generator to drive VTA activity
        self.neurons['ALL'] = nest.Create('parrot_neuron', self.N['ALL']) #A middleman parrot neuron
        self.vt = nest.Create('volume_transmitter')  # volume transmitter
        # Connect nodes in a chain
        # We can't connect the spike generator directly to the volume transmitter due to a NEST bug)
        nest.Connect(self.drive, self.neurons['ALL'], syn_spec={'delay' : self.dt})
        nest.Connect(self.neurons['ALL'], self.vt, syn_spec={'delay' : self.dt})
        self.DA_pars['vt'] = self.vt[0]

        # Create synapse that will be used by cortico-striatal neurons
        nest.CopyModel('stdp_dopamine_synapse', 'corticostriatal_synapse', self.DA_pars)


    def set_drive(self, length, drive_type='baseline', delay=None):
        drive_types = ['baseline', 'rewarding', 'aversive']
        if drive_type not in drive_types:
            raise ValueError('drive_type␣must␣one␣of␣those:', drive_types)

        begin = nest.GetKernelStatus()['time'] + self.dt
        end = begin + length - .5 * self.dt  # subtract .5 dt for numerical stability

        if drive_type == 'baseline':
            spike_times = np.arange(begin, end, self.dt)
        else:
            if delay is None:
                raise ValueError('It␣is␣necessary␣to␣specify␣the␣delay␣for␣reward␣or␣aversion')
            delivery = begin + delay
            if drive_type == 'rewarding':  # i.e the baseline with some extra spikes
                spike_times = np.sort(np.concatenate((
                    np.arange(begin, end, self.dt),
                    np.arange(delivery, delivery + (self.reward_size_ - .5) * self.dt, self.dt)
                )))
            elif drive_type == 'aversive':  # i.e. the baseline with some missing spikes
                spike_times = np.concatenate((
                    np.arange(begin, delivery - .5 * self.dt, self.dt),
                    np.arange(delivery + (self.dt * self.aversion_size_), end, self.dt)
                ))

        spike_times = np.round(spike_times, decimals=1)
        nest.SetStatus(self.drive, params={'spike_times' : spike_times})
```