École polytechnique fédérale de Lausanne

Semester project spring 2024

Master in Computationnal Sciences and Engineering

Study the full graph (connectome) of the fly nervous system

Author:
Sam Jegou

Supervisors:
Sibo Wang
Femke Hurtak

Contents

1 Introduction **1**
- 1.1 Introduction . . . 1
- 1.2 Notations . . . 1

2 State of the art **1**

3 Methods **1**
- 3.1 Connectome-constrained neural network . . . 1
  - 3.1.1 From the connectome model... . . . 1
  - 3.1.2 ...to an artificial neural network . . . 2
  - 3.1.3 Agent and value network - the whole pipeline . . . 3
  - 3.1.4 About the initialisation . . . 3
- 3.2 Training the agent . . . 3
  - 3.2.1 Very simple task . . . 3
  - 3.2.2 Bipedal Walker model . . . 3

4 Results **4**
- 4.1 Very simple model . . . 4
- 4.2 Bipedal Walker . . . 4

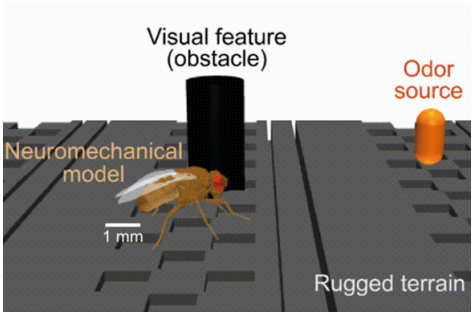
5 Discussion **7**
- 5.1 Conclusions . . . 7
- 5.2 Review of the method . . . 8
- 5.3 Future work . . . 8

A Gantt Chart **10**

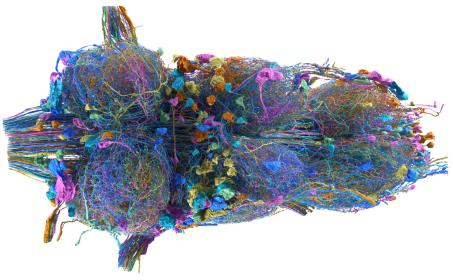
B Code **11**

Abstract

A recent study mapped full neural circuits involved in the fly behaviour, namely the Ventral Nerve Cord [7]. This graph of neuronal connections is called a connectome. Coupled with a realistic numerical model of the fly, this allows the exploration of a realistic-like neural network architecture to control the fly. This semester project aims at building a connectome constrained neural network, to reproduce the walking movement of the fly.



(a) Artificial fly



(b) Connectome of the male drosophila

Figure 1: Artificial fly and connectome

1 Introduction

1.1 Introduction

The connections between neurons in the ventral nerve cord (VNC) of the fly can be interpreted as a natural graph of connections between different neurons, that performs well to produce specific behaviour such as walking. It is a reasonable assumption to believe that this type of neuronal architecture can be implemented in a neural network that should be able to learn these behaviours.

The goal of this project is to implement such a neural network and make it control a numerical copy of the fly, to reproduce some behaviours performed by real flies.

More specifically, this semester project focuses on the walking behaviour. Hence, it aims at creating a neural network which has the same graph structure as the part neurons in the connectome responsible for the walking behaviour, and train an agent in a reinforcement learning (RL) framework to reproduce this leg movement.

1.2 Notations

The following notations will be used in this report:

- for any variable x, x_t is the instance of x at time step t.
- for any variable x, $\dot{x}$ is the time-derivative of x
- for any variable x, $\hat{x}$ represents this variable in the reference state

2 State of the art

As mentioned earlier, [7] managed to map the VNC circuits, thus creating an annotated numerical graph of the neuronal connections of the fly's brain. From this, [1] identified different neuronal circuits involved in leg movements, as well as in the coordination between all the legs. The team could also hypothesise precise groups of neurons as well as the interaction between these neurons (excitation, inhibition, etc.) responsible for different parts of the leg movement. This pre-motor circuit is referred as Standard Leg Connectome.

A team of researchers managed to build a connectome-constrained network to predict the neuronal response of the fly visual system [3]. They used the neuronal connectivity measured from connectomic measurements, together with hexagonal convolutional neural networks (to mimic the fly's vision) to accurately predict the response of the fly visual system to visual stimuli. They name the class of models they created as "connectome-constrained and task-optimised deep mechanistic networks".

The recent algorithm DMAP [2] presents as an intermediate between a completely biologically inspired artificial agent and the standard MLP approach. This algorithm has independent proprioceptive processing and a distributed policy with individual controllers for each joint. This is very relevant as [1] showed a local control of each leg, and a group of neuron for the coordination between the legs.

Finally, on the RL side, [4] released an algorithm with many techniques to help a virtual agent (human, robot, dragon, etc.) evolve and learn specific movements in a 3D environment. Their article provides a lot of insights about the difficulties of such tasks, and the strategies one can use to improve the training of a RL agent.

3 Methods

3.1 Connectome-constrained neural network

In this section is described the process to build a neural network from the connectome structure, which will be referred to as **BrainNN**.

3.1.1 From the connectome model...

The standard leg connectome [1] (Table C Figure 13) makes it easy to interpret the activation of the neurons, and therefore to assess the pertinence of the artificial neurons from a biological point of view. However, taking all the neurons in one standard leg pre-motor circuit ends in a total of 93 neurons and around 50 motorneurons, which is not enough to train a RL agent. We will refer to this model as the **Standard Leg Model**.

To improve the representative power of our model, another method was to select in the connectome all neurons targeting a specific leg. With this method, we captured 1101 neurons, with more than 37026 connections in-between them. However, it makes it harder to interpret the weights and biases in the artificial neural network from a biological point of view. This model will be referred as the **Complete Model**.

3.1.2 ...to an artificial neural network

Two main possibilities exist to build a neural network with a specific graph structure: graph neural networks and pruned and recurrent multi-layer perceptron (MLP). The latter was chosen for the familiarity with PyTorch framework, but the first one is still an interesting direction.

The first step was to build the adjacency matrix A of the weighted directed graph of neuronal connections, where A_{ij} is the number of connections from neuron i to neuron j.

Then, in order to organise all the artificial neurons in layers, one can solve a coloration problem. Indeed, in each layer of a MLP, there should not be two connected nodes. Each node is assigned to a colour, such that no node with the same colour are connected. Finding the minimal number of colours to use is a NP-hard problem, but it is possible to find a small number of colours (via `networkx` library in this report).

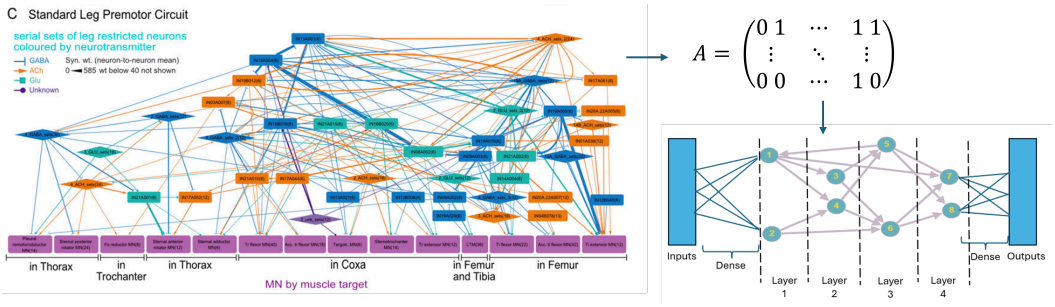


Figure 2: Schematic view of the process Connectome → NN

Then the NN corresponding to this graph has forward, skip and recurrent layers. The update rules for the layers are the following

$$\forall i = 1, ..., L, \quad \mathbf{x}^i_{t+1} = \sigma\left(\mathbf{W}^i \mathbf{x}^{i-1}_{t+1} + \mathbf{R}^i \mathbf{s}_t + \mathbf{S}^i \mathbf{s}_{t+1} + \mathbf{b}^i\right)$$
$$\forall i = 1, ..., L, \forall j = 1, ..., N_i, \quad (\mathbf{s}_{t+1})_{f(j)} = (\mathbf{x}^i_{t+1})_j$$

where

- i is the number of the layer
- j is the position of the node in its layer
- $\mathbf{x}^i_t \in \mathbb{R}^{N_i}$ is the vector of nodes in layer i at time t
- $\mathbf{s}_t \in \mathbb{R}^N$ is the hidden states vector: it contains the state of all the nodes at time t, but is updated during the propagation of the information (see below)
- $\mathbf{W}^i, \mathbf{R}^i, \mathbf{S}^i$ are the trainable weight matrices for forward, recurrent and skip layers respectively
- $\mathbf{b}^i$ is the trainable bias of layer i
- σ is the activation function

Note that the hidden states vector is updated during the propagation, hence nodes in layer $j > i$ will receive the most recent information from layer i. On the contrary, nodes in layer $i < j$ will receive (recurrent) information of layer j from the precedent propagation.

Finally, there are no input or output layers. When the BrainNN is initialised, a fraction p_{in} (resp. p_{out}) of all the nodes are chosen to receive the input information (resp. produce the output), via trainable weights and biases.

3.1.3 Agent and value network - the whole pipeline

This part aims at summarising the whole RL pipeline, more details can be found in [4].

In the different tasks, an agent interacts with an environment according to a policy, and tries to maximise the reward it obtains.

There are two main networks, the value network and the agent network. They both share the same architecture (described above), but each train its own parameters. The value network is trained to predict the average reward the agent should obtain if it starts in a state $\mathbf{s}_t$. The agent is trained to take output the mean action that should be taken given a state $\mathbf{s}_t$. Then the action the agent performs is sampled from a normal law centred on the mean action, and with a fixed variance (hyperparameter).

The rewards given to the agent are adapted from [4], and their precise values are detailed in 3.2.2.

3.1.4 About the initialisation

There are two main set of parameters to initialise in the networks: the hidden states, and the initial weights matrices and biases.

Two strategies have been tested for the hidden states initialisation, namely to set this vector to 0 or to sample it from a normal law with centred and reduced Gaussian (retained solution).

The trainable parameters are either initialised using default PyTroch initialisation (uniform sampling), or using connectome information to give higher weights to connections between two neurons sharing a high number of synapses. The second solution was used, with the following sampling

$$\sigma = 3\exp\left(2\left(\frac{N_{ij}}{N_{max}} - 1\right)\right)$$
$$w_{ij} \sim \mathcal{U}([\pm\sigma - 1, \pm\sigma + 1])$$

where the sign of $\pm\sigma$ is chosen at random.

3.2 Training the agent

3.2.1 Very simple task

To validate the behaviour of the network, a model with 8 neurons and some skip, forward and recurrent connections (see Fig.2) was trained to predict from the input $\sin(n\omega\Delta t)$ the next iterate $\sin((n+1)\omega\Delta t)$.

The neural network matched the performance of a MLP with the same number of connections on this task. The results can be found in Fig.5.

The same task was also performed receiving as input the angle and velocity ($\omega\cos(\omega\Delta t)$) and predict both the next angle and velocity.

3.2.2 Bipedal Walker model

The next step to validate the model is to train it to perform a more complex task, but simpler than the walking movement of a fly. To this purpose, the Bipedal Walker (Fig.4) environment from Gymnasium [6] has been used. The agent is composed of a big hull and two legs with two joints each (knee and hip). The agent is controlled through the torque applied on each of the joint. The observation variables are the joint angles and angular velocities, the agent velocity, the hull angle and angular velocity, and 10 "lidar readings".

In order to train the agent on this task, the PPO algorithm [5] has been used.

Instead of the default reward of the Bipedal Walker environment, the policy network receives a modified reward inspired of [4]. At each time step t, the reward is defined as $r_t = \omega^I r_t^I + \omega^G r_t^G$, where superscript G stands for "Goal" and I for "Imitation".
The imitation reward is itself the sum of pose and velocity rewards, $r_t^I = \omega^p r_t^P + \omega^v r_t^v$. These last rewards are defined as

$$r_t^p = \exp\left(-\sum_{i=1}^{4}\left(\cos(\theta_t^j) - \cos(\hat{\theta}_t^j)\right)^2 + \left(\sin(\theta_t^j) - \sin(\hat{\theta}_t^j)\right)^2\right), \tag{1}$$
$$r_t^v = \exp\left(-1.5\sum_{i=1}^{4}\left(\dot{\theta}_t^j - \hat{\dot{\theta}}_t^j\right)^2\right)$$

where θ_t^j represents the angle of joint j at time t.

The goal reward is defined as

$$r_t^G = \exp\left(-4 \max\left\{\bar{v}_x - v_{x,t}, 0\right\}\right) \tag{2}$$

where $\bar{v}_x$ is the mean horizontal velocity of the reference movement, and $v_{x,t}$ the agent's horizontal velocity at time t.

The best training were obtained for

$$\omega^I \in [0.7, 0.9] \text{ and } \omega^G \in [0.1, 0.3],$$
$$\omega^p = 0.7 \text{ and } \omega^v = 0.3$$

Another very important parameter is the standard deviation of the sampling of the action (in the policy network). During this project, it has been set to $\sigma \in \{0.1, 1\}$.

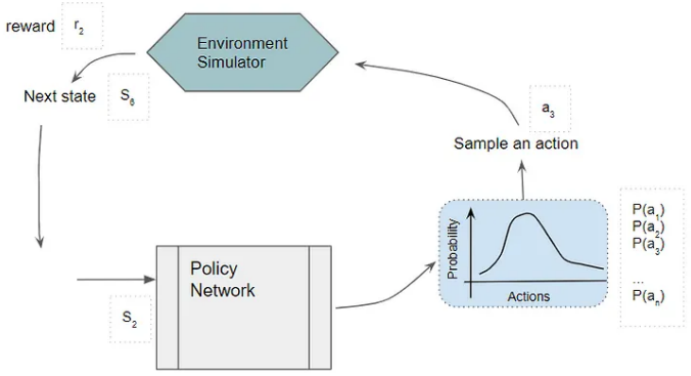


Figure 3: RL process schematic representation (from TowardsDataScience)

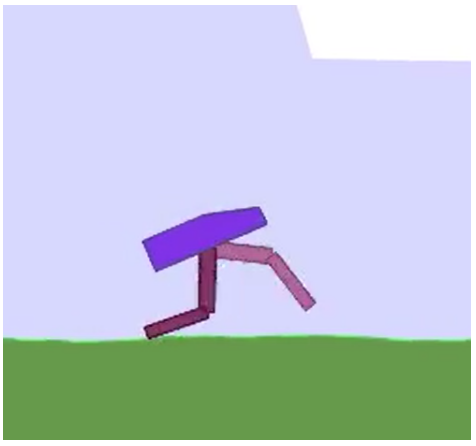


Figure 4: Bipedal walker

4 Results

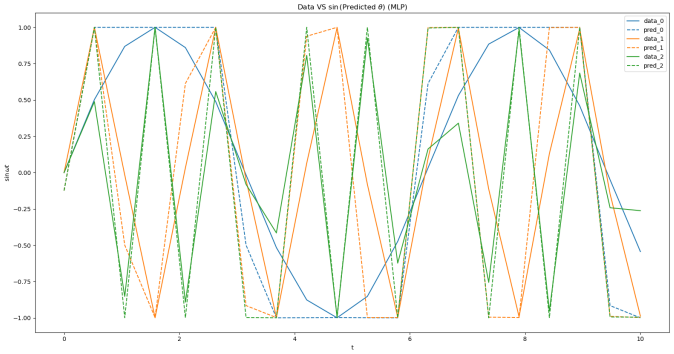
4.1 Very simple model

Even very simple NN with 8 nodes could predict the $\sin(\omega t)$, with better performances than a MLP with the same number of connections (Fig.5).

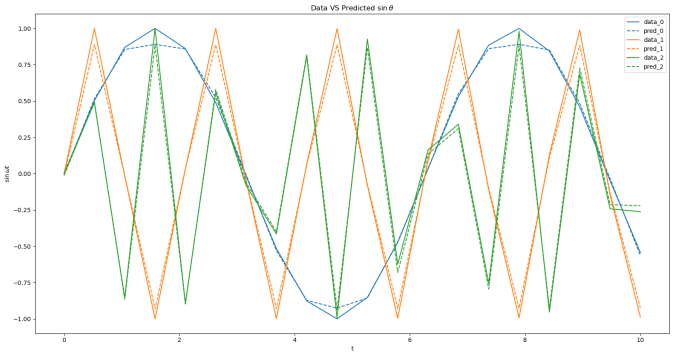
4.2 Bipedal Walker

All the training have been done on 3000 epochs, with a maximum of 500 time steps per episode.

Despite the different settings tested, the agent has not been trained successfully. The main problems and solutions implemented are listed here, as well as some insightful graphs.



(a) MLP performance



(b) Simple RNN performance

Figure 5: $\sin(\omega t)$ (plain) and $\sin(\theta_t)$ (dashed) for MLP and BrainNN

On Fig.6 one can clearly see that the agent learns to obtain higher rewards during training. However, this does not correlates with a significant decrease in the different losses. A solution investigated was to play with the discounted return factor γ, which is responsible for the rate at which the agent forgets the past.

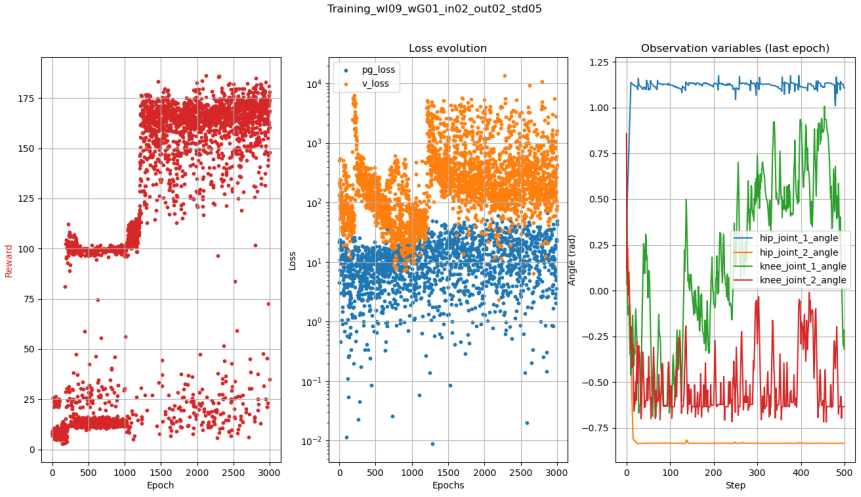


Figure 6: Reward, value and policy losses during training, joints angles (last epoch)

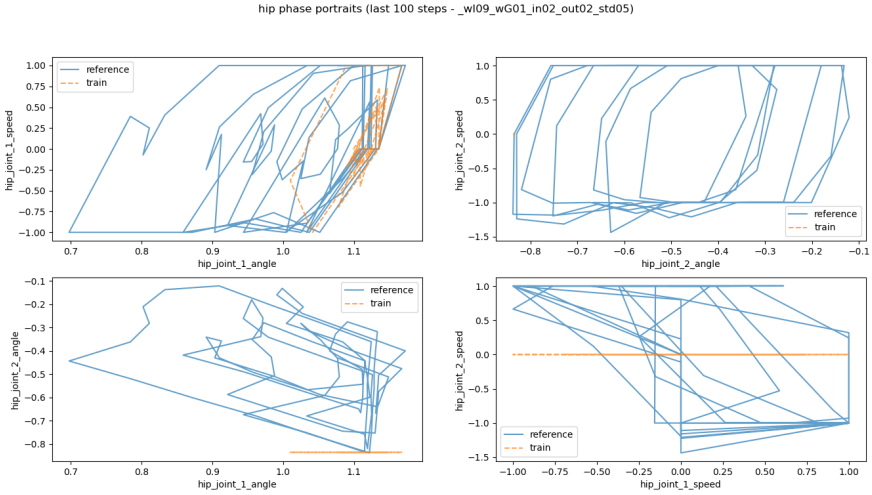
Then Fig.7 show phase portraits of the reference agent and trained agent movements, with different combinations (hip 1 angle VS hip 2 angle, hip 1 angle VS hip 1 angular velocity, etc). It is clear that the hips (Fig.7a) are not moving, which corroborates the observations in Fig.9. From the knee phase portraits, the following

observations can be made:

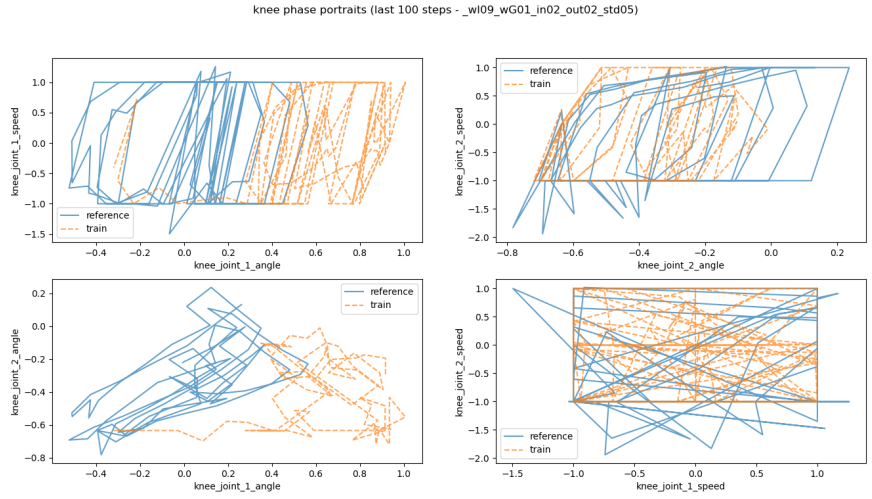
- (top-left and right): the pattern of the relation between angle and angular velocity seems to somewhat follow the one of the reference movement, even though it is shifted
- (bottom-right) there is no clear reference pattern of the knee angular speed 1 VS 2, and it is not learnt by the agent
- (bottom-left) the coordination between the two legs (knee angle 1 VS 2) is not learnt by the agent

One additional remark must be made: as the hips do not move, the agent ends in a position where some movements are not allowed by the physics of the environment, or at least very hard to reproduce given the imitation objective of the agent.

A potential solution to this problem has been to modify the weight in the imitation/goal rewards, to emphasise the imitation of the reference movement.



(a) Hip phase portraits



(b) Knee phase portraits

Figure 7: Hip and knee phase portraits of the reference (plain) and trained (dashed) agent

Then, another diagnosis can be made by looking at the actions (torque applied to each joint) taken by the agent, in Fig.8. It clearly appears that the movement of the trained agent is noisy.

However, trying to reduce the noise in the action sampling (from $\sigma = 0.5$ here to $\sigma = 0.1$) resulted in a lack of exploration, and the agent did not learn anything, nor tried to move. It was stuck in the same position from epoch 0 to the final epoch.

Another attempt has been to look at the magnitude of the action. Indeed, if the environment clips the actions to $[-1, 1]$, both the reference and the trained agents took actions with magnitude in $[-8, 8]$ approximately. A potential solution would be to directly clip the agent's actions to $[-1, 1]$ at the output of the BrainNN, but it proved inefficient (neither better nor worse).

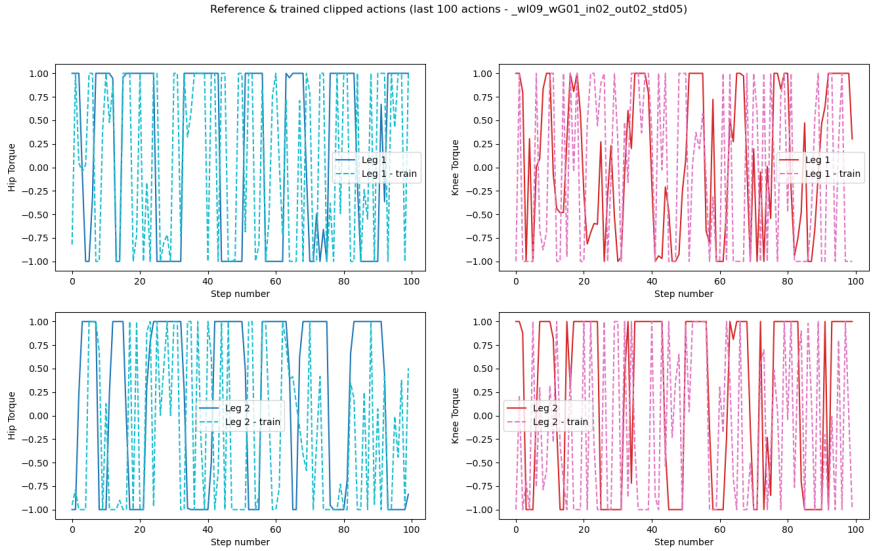


Figure 8: Action (torque) of the reference (plain) and trained (dashed) agents, leg 1 and leg 2

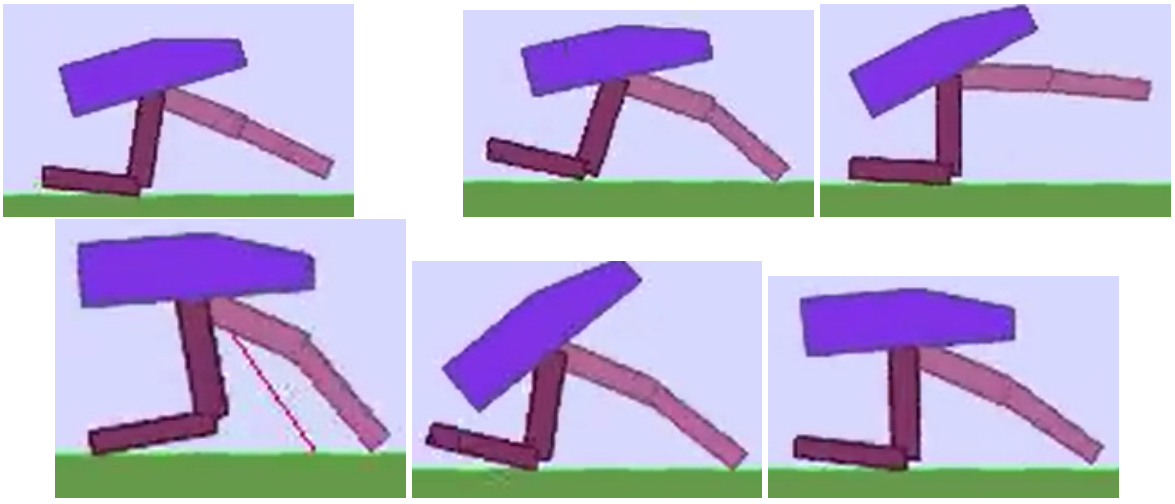


Figure 9: Agent stuck in a "low" position

5 Discussion

5.1 Conclusions

Is the BrainNN proved to perform well on a very simple task, its training in a reinforcement-learning context proved to be challenging. Despite the steps taken to simplify the problem, the complexity of the Bipedal Walker environment offered too many degrees of liberty to fix the agent's learning.

However, the difficulties met during this study contributed to a better understanding of the problems and the implementation of functionalities that will prove useful in the future.

Also, BrainNN combines a lot of existing specificities of different networks (MLP, recurrent neural network, NN with skip connections) and is itself an output of the project. Though it did not (yet) proved to be efficient on a RL task, there is still a lot to explore on this neural network.

5.2 Review of the method

If I had to do the project again from the start, I would...

- spend less time trying to fix a specific problem that can be avoided (e.g creation and use of a torchRL environment)
- be more realistic (less ambitious) in the Gannt chart, and anticipate better the problems that could occur.
- remember to often take a step back to identify the current problems, or orient the work in the appropriate direction.
- choose a simpler problem than the Bipedal Walker to test the BrainNN.

5.3 Future work

1. In order to identify precisely the problem of the current algorithm, it would be interesting to train the agent directly on the action (and not from an action-based reward). This would allow to assess the correct behaviour of the neural network, independently of the PPO algorithm.
2. Another solution that arises from Fig.7a would be to implement different rewards for the hip and the knee: it might be easier to learn to walk by focusing first on the hip joints, and then on the knee than being stuck in a low position (Fig.9) and try to activate the knee joints.
3. The last solution would be inspired from [2] and use 4 small BrainNN, one per joint. The "decentralised" training proved to be successful in the latter study.
4. If the training were a success, then the study could continue by changing of environment, to train on (one leg of) the fly numerical model.
5. Then, it would be interesting to build another neural network still based on the connectome, to control the leg coordination.
6. In parallel to the previous training, an interesting study could be to observe the characteristics of the trained neural network, either local (weights, biases) or global (node centralities, dynamic behaviour) to see if they match the ones of a biological fly brain.

References

[1] H. S. J. Cheong, K. Eichler, T. Stürner, S. K. Asinof, A. S. Champion, E. C. Marin, T. B. Oram, M. Sumathipala, L. Venkatasubramanian, S. Namiki, I. Siwanowicz, M. Costa, S. Berg, G. S. X. E. Jefferis, and G. M. Card. Transforming descending input into behavior: The organization of premotor circuits in thedrosophilamale adult nerve cord connectome. June 2023.

[2] Alberto Chiappa, Alessandro Marin Vargas, and Alexander Mathis. DMAP: a distributed morphological attention policy for learning to locomote with a changing body. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[3] Janne K. Lappalainen, Fabian D. Tschopp, Sridhama Prakhya, Mason McGill, Aljoscha Nern, Kazunori Shinomiya, Shin ya Takemura, Eyal Gruntman, Jakob H. Macke, and Srinivas C. Turaga. Connectome-constrained deep mechanistic networks predict neural responses across the fly visual system at single-neuron resolution. *bioRxiv*, 2023.

[4] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4), jul 2018.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[6] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[7] Shin ya Takemura, Kenneth J Hayworth, Gary B Huang, Michal Januszewski, Zhiyuan Lu, Elizabeth C Marin, Stephan Preibisch, C Shan Xu, John Bogovic, Andrew S Champion, Han SJ Cheong, Marta Costa, Katharina Eichler, William Katz, Christopher Knecht, Feng Li, Billy J Morris, Christopher Ordish, Patricia K Rivlin, Philipp Schlegel, Kazunori Shinomiya, Tomke Stürner, Ting Zhao, Griffin Badalamente, Dennis Bailey, Paul Brooks, Brandon S Canino, Jody Clements, Michael Cook, Octave Duclos, Christopher R Dunne, Kelli Fairbanks, Siqi Fang, Samantha Finley-May, Audrey Francis, Reed George, Marina Gkantia, Kyle Harrington, Gary Patrick Hopkins, Joseph Hsu, Philip M Hubbard, Alexandre Javier, Dagmar Kainmueller, Wyatt Korff, Julie Kovalyak, Dominik Krzemiński, Shirley A Lauchie, Alanna Lohff, Charli Maldonado, Emily A Manley, Caroline Mooney, Erika Neace, Matthew Nichols, Omotara Ogundeyi, Nneoma Okeoma, Tyler Paterson, Elliott Phillips, Emily M Phillips, Caitlin Ribeiro, Sean M Ryan, Jon Thomson Rymer, Anne K Scott, Ashley L Scott, David Shepherd, Aya Shinomiya, Claire Smith, Natalie Smith, Alia Suleiman, Satoko Takemura, Iris Talebi, Imaan FM Tamimi, Eric T Trautman, Lowell Umayam, John J Walsh, Tansy Yang, Gerald M Rubin, Louis K Scheffer, Jan Funke, Stephan Saalfeld, Harald F Hess, Stephen M Plaza, Gwyneth M Card, Gregory SXE Jefferis, and Stuart Berg. A connectome of the male drosophila ventral nerve cord. *bioRxiv*, 2023.

A Gantt Chart

The original Gantt Chart is shown in Fig.10

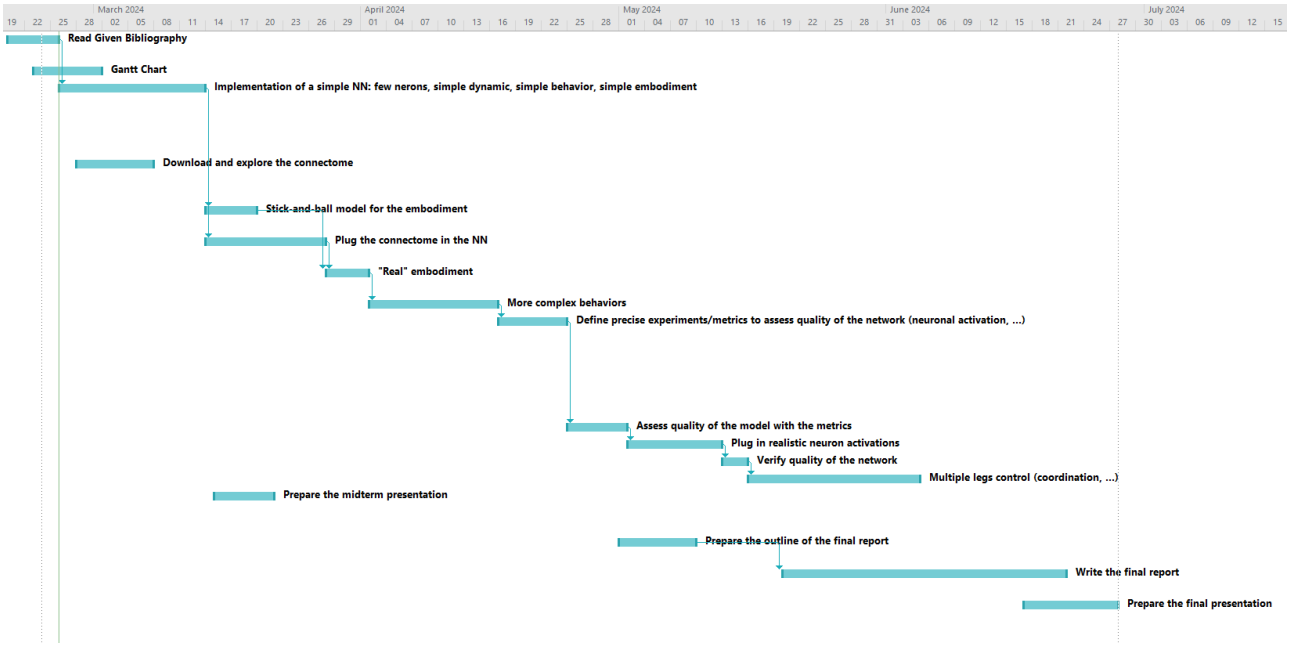


Figure 10: Original Gantt Chart

The real timeline of the project is presented in Fig.11.

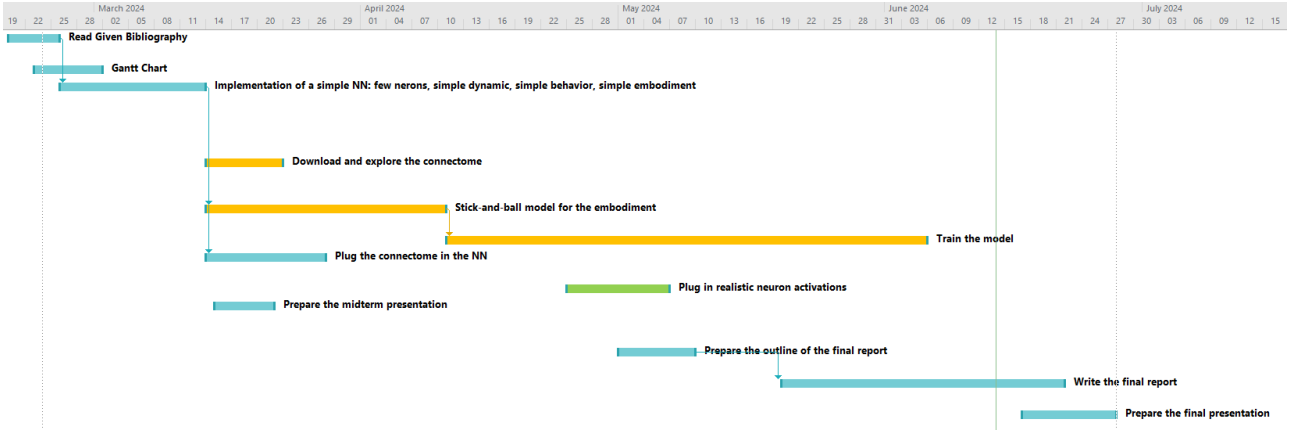


Figure 11: Real Timeline

The difference between the two timelines are the following:

- The downloading and the exploration of the connectome was delayed to after the implementation of BrainNN. This implementation as more time-consuming than anticipated as some efforts were put to make it very easy to adapt to any graph, and not just a specific one.
- The stick-and-ball embodiment has take much longer than anticipated. Indeed at first I wanted to create a TorchRL environment specific to the task, but it happened to be very difficult (as the GitHub request shows, there was a compatibility issue). So I had to choose an already-existing environment.
- The training of the model was (and still is) very difficult, as this report shows. Hence this task was delayed until the last moments.
- Finally, the only task anticipated from the original Gantt Chart was the initialisation of the weights of the network in adequacy with the number of synapses between (biological) neurons.

B Code

The main files to run the codes for this proejct are listed below

- `connectome_by_neurons.py` create an adjacency matrix from the connectome (see 3.1.2). The selected neurons are the ones present in the Standard Leg Premotor Cricuit in [1].
- `connectome_by_target.py` create an adjacency matrix from the connectome (see 3.1.2). The selected neurons are the ones with the same target (LegNp1 in this study).
- `BrainRNN.py` implements the class `BrainRNN`, which is the connectome-like neural network.
- `RL_bipedal_hand.py` is the main file to run the RL pipeline. It trains the agent as described in 3.2.
- All the `..._plots.py` are used to create the plots shown in 4.2.
- The file `settings.py` is read by BrainRNN and the RL files for the main settings (number of inputs/outputs, adjacency matrix, etc).
- The file `ref_mvt_code.py` was used to produce the reference movement (from an agent trained on Google Colab).
- The files `benchmark_MLP.py`, `study_benchmark.py` and `study_model.py` were used during the very simple task (see 4.1.