# Spiking Neural Networks as a Model for the Nervous System of *Hydra*

## 1 Introduction

An artificial neural network is a simplification of a biological neural network, generally based on a directed weighted graph. The vertices of the graph represent neurons, and the edges represent connections between them. Typically, these artificial neurons are only vaguely inspired by biological neurons, and therefore are unsuitable for use in modeling biological nerve networks. We hope to work past some of the biological limitations of traditional artificial neural networks using what are known as spiking neural networks, and aim to apply them in creating a model of the very primitive nervous system of the metazoans *Hydra*.

## 2 The nervous system of *Hydra*

*Hydra* is a genus of small, freshwater cnidarians which have long been studied in laboratories, due to both their ease of culturing and small size (typically 10 mm in length). They have one of the simplest nervous systems known among metazoans, yet they are still capable of nontrivial behaviors, unlike more basal clades such as *Porifera* (sponges). Due to this simplicity, as well as the large knowledge base surrounding *Hydra*, we aim to develop a basic model of the *Hydra* nervous system.

Like other cnidarians, *Hydra* do not have a central nervous system, but rather a nerve net, which is characterized by neurons distributed throughout the body with little centralization (i.e. no brain). That said, the neurons are still not distributed homogeneously across the body of *Hydra*, but assuming as much is a decent approximation. Extending our model to consider more complex neuronal distributions would be an interesting topic for further exploration.

Being such primitive organisms, *Hydra* have a very limited set of behaviors. *Hydra* are capable of movement, but they are primary sessile, with their primary behaviors being elongation and contraction. Interestingly, Han et al., 2018, examined freely behaving *Hydra* and found that the specific set of behaviors observed is independent of environmental conditions. Dupre and Yuste, 2017, completed the first ever full nervous system activity map using *Hydra*, and found that the primary behaviors of *Hydra* are controlled by disjoint circuits in the nerve net. That is, each neuron belongs to a single connected group of neurons (a circuit), and each circuit controls a specific behavior. However, at least two of the circuits have an antagonistic relationship, so they are not completely independent of each other.

We plan to use a spiking neural network as the basis for our model. Unfortunately, we do not have any numerical data to use, so we will have to train the network on estimated data points. The exact values

should not matter as long as orders of magnitude are consistent across the data set – all that should matter is correlation. For example, as mechanical pressure is applied to the tentacles, the *Hydra* should contract longitudinally (as observed experimentally), and this negative correlation should be reflected in the data set.

# 3 Spiking Neural Networks

Traditional neural networks used today in deep learning have their neurons fire only during what is known as a propagation cycle. That is, if the neural network is to act as an agent in some environment, then the timesteps must be discrete. On one hand, making the simplification that neurons fire only during a propagation cycle makes the simulation of networks much easier, and is perfectly fine for applications such as image recognition. However, it is far less realistic when trying to simulate an actual biological nervous systems which runs on a continuous time scale.

Spiking neural networks aim to solve this problem in part. Firstly, this type of neural network includes continuous time as a factor in the model. This leads artificial neurons to behave more like biological neurons, only firing once a predetermined 'membrane potential' has been reached. If the input is not strong enough, the current potential will degrade over time. If the neuron fires, the signal travels to other neurons but not instantaneously as in traditional networks. Finally, a refractory period can be added so that the neuron does not fire immediately after having fired, giving it a sort of cool down period as is observed in biological neurons. All these properties together make spiking neural networks a better choice for our model than any other traditional neural network design.

# 4 Appendix: Optimization through graph tensor products

We assume that the graph structure of a biological neural network at some time $t$ can be modeled by a time dependent function $\mathbb{M}$ that transforms the edge and vertex sets of an initial graph $G$ in some fashion. Further, we will model networks under the assumption that the initial structure of $G$ before any edge weights are modified can be approximated by the tensor product of other graphs:

$$G = (H_1 \otimes H_2 \otimes \cdots \otimes H_L)$$

The reason for this assumption is as follows. For the tensor product $C = A \otimes B$ of two graphs $A, B$ with $|V_A| = h, |V_B| = k$, the size of the vertex set of $C$ is $h \cdot k$. However, we can simulate a random walk on $C$ with only $(h + k)$ memory for the vertices, despite the vastly increased complexity of the graph. This is a fundamental property of graph tensor products. For our applications, of course, we are not trying to simulate a random walk, but there are computational savings nonetheless. See section 4.1 for more details.

An important distinction is that a neural networks changes with time as it learns, and this is where our function $\mathbb{M} : (G, t) \mapsto G_t$ comes into play. In most artificial feed-forward neural networks, the function would simply modify the weights of the edges of $G$ in accordance to some fitness function. However, for our purposes, modification of the underlying product components $H_1, H_2, \ldots, H_L$ may be advantageous, particularly in the early stages of learning. This is because modifications to the underlying product components create large changes in the structure of the network at little computational expense.

## 4.1 Code notes

- When storing the graph $G$, we store the tensor product components $H_1, H_2, \ldots, H_L$ where $L$ is the number of layers. To specify a vertex $v$ in $G$, we store in an $L$-element array the vertices $v_i \in H_i$ to which $v$ corresponds to.

- To store an edge $e$ that is an addition to $G$ or the change of an existing edge weight in $G$s we store the vertices $v_1, v_2$ which it connects, and store the edge in a data structure sorted by the input vertex $v_1$. Given a vertex $v_1$, this lets us easily access the edges leading away from $v_1$.

- To avoid having the repetitive weights from the original graph $G$, the weights of $M(G, 0)$ are determined by hash function until the vast majority of edges are changed, and then we begin to store the edges directly.