Modeling a biological neural network with a weighted network model and a genetic algorithm

Ragnhild Skirdal Frøhaug[1,1]

Abstract

Purpose The aim of this project is to emulate the network connections, spiking thresholds and firing patterns of a biological neural network by implementing a weighted network model.
Method A weighted Network model was implemented to emulate the features. Neuron firing patterns was detected by simulating time with discrete time steps. The fitness of the model was then evaluated by comparing the firing rates of the individual neurons and the firing rate of the whole system across time intervals with the firing rates of a biological neural network. The features were evolved using a genetic algorithm with crossover rate, mutation and a sigma scaling roulette wheel for parent selection.
Results The best results were achieved on the Small data set, fitting the model by using the time interval firing rates as a fitness function. The best run yield a fitness value of 0.997 out of 1.

1. Introduction

Biological neurons fire when the input signal to the neuron is above a threshold (explained in a very simple manner). The signal is then propagated to other neurons via network connections. It is today possible to grow biological neural networks on multi-electrode arrays (MEAs), which makes it possible to record spiking behaviour in a biological neural network. Each electrode in the MEA records electrical signals within a small area of the biological neural network. The electrode can also stimulate the network in order to make the neurons in the biological network spike [1].

Wagenaar et al. have recorded the behaviour of sparse, small and dense biological neural networks. Sparse, small and dense refers to the amount of biological neurons and connections in the network. The aim of this project is to evolve a network model to emulate features of these sparse, small and dense biological neural networks. The behaviour of the network model is compared with the data recorded by [1].

Email address: s350110@oslomet.com (Ragnhild Skirdal Frøhaug)

For simplicity, one electrode in the data provided by Wagenaar et al. represents a neuron in the Network model. Hence, the Network model is a simplification of the biological neural network as one electrode record and affect multiple neurons and connections in the biological neural network. However, the Network model is still useful as the goal is to simulate the behaviour of the electrodes. Hence, the model might be able to emulate the behaviour of the different areas of the biological neural network. It might also be possible to detect some patterns in the weights and neuron thresholds which can provide insight on how the biological neurons are interconnected and how the neurons respond to input signals.

The Network model and the genetic algorithm implemented in this project tries to emulate the thresholds of the neurons, the typology of the network and the strength of the weight connections. Further the model aims to identify which neurons fire when through time. The model is evaluated by comparing the firing rate of the model neurons with the firing rates of the electrodes through a selected time interval. The time interval is typically between 3 and 10 seconds. Additionally, the firing rate of the whole system during a set time interval is evaluated.

The report is structured as follows: Section 2 Methods, describes the implementation of the Network model and the Genetic Algorithm. Section 3 gives an overview of the results of the runs of the genetic algorithm using the different fitness functions and data sets with different density. Next, in section 4 the results are further analyzed and discussed. Lastly, section 5 concludes the report and outlines areas to be further explored and researched.

2. Methods

This section describes both the structure and the implementation of the Network model and the structure and implementation of the genetic algorithm.

2.1. ***The Network model***

Table 1 presents an overview of the parameters of the Network model. The parameters that are evolved by the genetic algorithm are the *Weights*, the *Thresholds* and the R*Reset Firing Neurons*. The parameters *Non Spiky Neurons*, *Number of Clusters* and *Number of Zero Clusters* are used for initialization of the weights.

Further to be noticed from the network model is that one time step is simulating 1 ms, and that the refractory time of the neurons is set to be 2 ms.

The *Spiky Neurons* and the *Spiky Thresh constants* were only tested for the Sparse data set. The neurons with an above mean firing rate was identified as spiky. Then a spiky neuron

threshold constant was defined and the thresholds of the most spiky neurons was divided by this constant.

Table 1: Overview of the parameters of the network model.

Parameters	**Type**	**Description**
Number of neurons	int	The number of neurons in the network. Each neuron represents an electrode in the real data.
Weights	nxn ndarray of floats	The weights are represented as a matrix. Element i,j in the matrix represents the strength of the weight connection between neuron i and neuron j in the network.
Thresholds	nx1 ndarray of floats	The thresholds represent the value of the input signal before the neuron fire.
Reset firing neurons	nx1 ndarray of objects	If the network becomes silent, a set of neurons are popped from the list of reset firing neurons. These neurons fire, and the signal is then propagated through the network.
Non spiky neurons	nx1 ndarray of int	The non spiky neurons are the neurons that do not fire during the simulated time period. The weight connections to the non spiky neurons are set to zero.
Spiky Neurons	nx1 ndarray of int	Detecting which neurons in the data set have above mean firing rate and flagging them as spiky. Only used tested on the sparse data set.
Spiky Neurons Constants	nx1 ndarray of int	The thresholds of the spiky neurons are divided by the spiky threshold constant to lower their firing rate.
Number of Clusters	int	Used for initialization of the weights. Clusters are areas of stronger weight connections.
Number of Zero clusters	int	Used for initialization of the weights. Zero clusters are areas of the weights that are silenced.

2.1.1. Weight initialization

The weight connections are initialized drawing numbers from a uniform distribution between 0 and 1. In addition, areas with stronger connections and areas without any connections are added. The parameters *Number of Clusters* and *Number of Zero Clusters* are used to decide upon the number of Clusters and Zero Clusters in the Weight-matrix. The width and height of the clusters are between 3 and 6 (any combination), and the clusters are randomly placed across the weight matrix. Connections within a Zero Cluster is set to Zero, while connections within a Cluster is set to a number between 0.5 and 1. In addition to adding Clusters and Zero Clusters, the weight connections of the *Non Spiky Neurons* are set to zero. The weight

initialization is done for the first population the GA initializes. When the GA evolves the Network parameters, it combines the weights of the individuals. Hence, when the population is evolved through the generations, the weights of the network are passed onto the network and not initialized within the Network Model.

2.1.2. Reset firing neurons implementation

The *Reset Fire Neuron* parameter is initialized by creating an array of arrays with different length. When the network goes silent, an array with neuron indexes is popped from the "Reset Fire Neuron"-array, and used to "restart" the Network. The number of neurons that fires varies from 2 to 10 or 30.

2.1.3. Network simulation algorithm

Algorithm 1 describes the procedure of the Network Model with a simplified pseudocode. Note that for simplicity, not all elements are not included. The pseudo code should rather be viewed as an overview of how the model operates.

However, there are some comments to be made about the algorithm. First, a neuron has two values 0 and 1. When a neuron fire it is assigned the value 1. The input signal is computed by going through the list of firing neurons. Hence, the input signal to a neuron is computed as follows:

$$InputSignal_i = \sum_{n \in Fire_t} w_{ni} \tag{1}$$

Since a firing neuron has the value 1, we can sum up the weight connections going from the firing neuron n to neuron i. *InputSignal* is a list which represent all the neurons in the network. Line 15 then checks if the input signals in *InputSignal* is above the *Thresholds*. Both variables are lists, and the line checks if $InputSignal_0 > Threshold_0$, $InputSignal_1 > Threshold_1$, $InputSignal_2 > Threshold_2$ and so on.

Next, in line 9, a set of neurons is popped from *ResetFireNeurons* if the number of neurons that fire is below the *SilentThresh*, meaning that the network has gone silent. The definition of silent is $Count(Fire_t) < Round(Length(Fire_{tprev}))$. An example: 15 neurons fire, and the signals propagated through the weight connections result in 2 new neurons to fire. Hence, $SilentThres = 5$, and a new set of neurons is popped from the *ResetFiringNeurons*, due to to little "activity" in the network.

Algorithm 1 Network Model algorithm

Input Thresholds, Clusters, ZeroClusters, NumNeurons, InputWeights, ResetFireNeurons, NonSpikyNeurons,

TIMESTEPS

Output NeuronFireCount, Timeseries

procedure NETWORK MODEL PROCEDURE
  if $InputWeights == None$ **then**
    $Weights \leftarrow INITIALIZEWEIGTHS(Clusters, ZeroClusters, NonSpikyNeurons)$
  else
    $Weights \leftarrow InputWeights$
  $Fire_t \leftarrow ResetFireNeurons.pop()$
  $TotalFire_t \leftarrow Fire_t$
  for t in TIMESTEPS **do**
    if $COUNT(Fire_t) < SilentThresh$ **then**
      $Fire_t \leftarrow ResetFireNeurons.pop()$
    $InputSignal \leftarrow Zeros$
    for idx in $Fire_t$ **do**
      $InputSignal \leftarrow InputSignal + Weights[idx, :]$
    $Fire_{tprev} \leftarrow Fire_t$
    $Fire_t \leftarrow InputSignal > Thresholds$
    $Fire_t \leftarrow Fire_t * \neg Fire_{tprev}$
    $APPEND(TotalFire, Fire_t)$
  $Timeseries \leftarrow COMPUTETIMESERIES(TotalFire_t)$
  $NeuronFireCount \leftarrow COMPUTENEURON(TotalFire_t)$
  return NeuronFireCount, Timeseries

2.2. ***The Genetic Algorithm***

Algorithm 2 Genetic Algorithm

Input Population, Generations, NumAdults, NumChildren,
FitnessFunction, CrossoverRate,EarlyStopping

```
1: procedure EVOLVE GENERATIONS PROCEDURE
2:     INITPOPULATION(Population)
3:     for g in Generations do
4:         SELECTADULTS(fitnessFunction,selectionMechanism)
5:         MATING(fitnessFunction,NumAdults,Numchildren,CrossoverRate)
6:         CREATENEXTGEN()
7:
```

Algorithm 2 gives a condensed overview of the most important components of the genetic algorithm. First, an initial population is generated. Each phenotype in the initial population represents a simulation of the Network model with different input parameters.

Then, based on the fitness function a set of parents are selected from the initial population. The parents are selected using either sigma scaling or tournament selection. The number of adults are defined by the NumAdults parameter.

Next the genes of the selected parents are either copied onto the next generation or combined with the genes of another parent. The crossover rate defines the fraction of the adult population that is copied and the fraction that is combined.

There is also implemented elitism in the genetic algorithm. The elitism number is set to 5 individuals. This means that for each generation. The 5 individuals with the best fitness value is directly copied onto the next generation. Similarly, the 5 worst performing individuals in the new child population does not survive to the next generation. Table 2 gives a description of the parameters of the genetic algorithm.

2.2.1. Fitness functions

The genetic algorithm evaluates the fitness of each individual in the population based on a fitness function. The fitness functions used in this experiment measures the fitness of the firing rate of the neurons in the population and the firing rate across time in the simulation. Both fitness functions are based on the square error, comparing the firing rate from the simulation with the firing rates of the real data.

Fitness of neuron firing rate: The first fitness function tested to emulate features of the data is based on the square error of the individual neurons in the simulation. The square error is computed for each of the neurons in the simulation. This is computed as seen in equation 2;

Table 2: Overview of the parameters of the genetic algorithm.

Parameters	**Description**
Population	The number of phenotypes generated.
Generations	The number of generations of which the parameters are evolved.
Adults	The number of adults selected from the population in each generation
Children	The number of children each adult produces in each generation
Crossover rate	The number of times genes of two adults are combined (for the rest of the adults the genes are copied). Typically set to 80% crossover.
Fitness function	Either fit the timeseries firing rate, the individual neuron firing rate or a combination.
Early stopping	If the fitness value of the best performing individual have not improved over X generations. Stop the algorithm.

y_i is the real firing rate of neuron i and x_i is the simulated firing rate of neuron i.

$$SE_i = (y_i - x_i)^2 \tag{2}$$

SE is then an array consisting of all the square errors for the individual neurons. Next, the fitness of the individual is computed according to equation 3. Mean(SE) is the mean square error of the firing rates of the individual neurons. Perfect fitness of 1 is achieved when the mean square error is 0 and all the simulated firing rates equally corresponds to the firing rates of the real neurons.

$$fitness(neurons) = \frac{1}{1 - mean(SE)} \tag{3}$$

Fitness of the time interval firing rate: Another approach was also taken to evaluate the fitness of the individuals in a population. Similarly to the fitness function for neuron firing rate. This method also uses square error. However, instead of evaluating the firing rate of the individual neurons. This method evaluates the firing rate of the whole system across the whole network. As mentioned earlier, one time step in the network model equals 1 ms. The time intervals of which the firing rate was measured was set to 1 second. Hence the equation 4 represents the following; y_i is the real firing rate across time interval i and x_i is the simulated firing rate across time interval i.

$$SE_i = (y_i - x_i)^2 \tag{4}$$

SE is then an array consisting of all the square errors for the individual time intervals. Next, the fitness of the individual is computed according to equation 5. Mean(SE) is the mean square error of the firing rates of the time intervals. In addition, the square error of the time interval with the highest square error is added. This add on to the fitness function puts extra pressure

on the genetic algorithm to select the neurons that fits the worst performing time interval. Perfect fitness of 1 is achieved when the mean square error is 0 and all the simulated firing rates equally corresponds to the real firing rates of the time intervals.

$$fitness(neurons) = \frac{1}{1 - mean(SE) - max(SE)} \tag{5}$$

2.2.2. Parent Selection

Several approaches for parent selection can be taken. Two methods; sigma scaling with roulette wheel selection and tournament selection [2] have been tested in this project. However, during the experiments, sigma scaling was used the most. Both methods aims to select individuals for further evolving the population. The goal is to select individuals with features that are currently giving the highest fitness values. However, it is also important to maintain some variation amongst the individuals in the population in order to pass on characteristics which might improve the fitness at a later stage.

Tournament selection implementation: Tournament selection was implemented by selecting e phenotypes at random. Then the phenotype with the highest fitness value was added to the adult pool. Then k tournaments was performed. For this parent selection method, selection pressure is a function of k and e. K was set equal to the number of adults in the adult pool and e was set to 10.

Sigma Scaling implementation: The sigma scaling method scales the fitness value according to the mean and the variance of the fitness values in the generation as shown in equation 6 [2]. $f(g)$ is the mean fitness value of the generation, $f(i)$ is the fitness value of individual i and $\sigma(g)$ is the variance of the fitness values in the population.

$$ExpVal(i, g) = \frac{f(i) - f(g)}{2 * \sigma(g)} \tag{6}$$

Then the scaled fitness values are used to assign a selection probability to each phenotypes. In order to select phenotypes to pass onto the next generation, a "roulette wheel" is spun k times in order to select the k individual to be passed on as adults. Sigma scaling reduces selection pressure when a few individuals are much better. This means that when there are great variance in the fitness values in the population, individuals which currently have lower fitness values, gets a relative larger selection probability and a relative larger area on the roulette wheel.

Vice versa, when the population is homogeneous, the individuals that performs slightly better than the rest, gets a relative larger selection probability and a relative larger area on the roulette wheel.

2.2.3. Mating and crossover

The mating process is implemented with full generational replacement with elitism of 5. If the population is 100, then the mating process produces 100 children. Then the 95 children with the best fitness values are passed on to the next generation along with the five best performing individuals from the previous generation.

In order to produce new children, all adults in the adult pool mates with another parent in the adult pool with a crossover rate of 0.8. This means that in 20% of the cases the genes of the parent is copied onto the next generation, while in 80% of the cases, the genes of the adult is combined with the genes of another adult.

For thresholds boolean mask is used to combine the thresholds of the parents, same with the weights. For the reset firing neurons half of the reset fire neurons is from the first parent and half is from the second parent.

2.2.4. Mutation

There is a 5% chance of mutation on the thresholds and weights. In 5% of the phenotypes in a generation, one threshold in the list of neuron thresholds is replaced, and one weight in the weight-matrix is changed.

2.3. ***Parameter settings and datasets***

Table 3 shows which data sets that were used and the parameter settings used in the runs. For an individual in a population the thresholds were drawn from a uniform distribution between the number indicated in the table. For the reset firing neurons the limits represent the maximum and minimum number of reset firing neurons in one set. Lastly, the intervals for number of clusters and zero clusters are used across a population.

Regarding the spiky neurons which were tested on the sparse data set. This parameter was not further explored, as it did not seem to affect the fitness value positively in any significant manner.

Table 3: Parameter settings for the different runs.

Parameter	**Small-7-1-20**	**Sparse-7-4-35**	**Dense-2-2-20**
Thresholds	(2,40)	(2,10)	(2,100)
Reset firing neurons	(2,10)	(2,5)	(2,30)
Spiky Neurons	None	Above mean firing rate	None
Spiky Neurons Constants	None	(1,20)	None
Non spiky neurons	Yes	Yes	Yes
Number of Zero Clusters	(2,50)	(2,50)	(2,50)
Number of Clusters	(0,60)	(0,60)	(0,60)

2.4. ***Program structure and running the models***

The program consist of three main classes; the GeneticAlgorithm class, the Genotype class and the Network class. The GeneticAlgorithm class initializes one Genotype-object for each individual in the population. The Genotype-object then intializes a simulation of the Network-model and computes the fitness-value of the simulation.

In addition to these three main modules, there are also a two help classes. The DataFeatures-class fetches the data from the specified file and computes the firing rates of the neurons and the firing rates across time intervals for the real data set.

Last, Plotter-class has a set of plot functions which plots the performance of the models during and at the end of the run.

The program is controlled from the run() function. The Network parameters and the GA parameters can be adjusted at the top of the function. All adjustments to the parameters should be done here and not in the code of the models.

3. Results

This section presents the results of the runs and the performance of the models with varying the parameter settings. At least one run with a population of 500, 50 adults and 10 children was evolved over 200 generations for each of the data sets . For the small data set a run with a population of 2000 individuals (100 adults, 20 children) was evolved over 200 generations to test if a larger population improved the performance of the model for the neuron firing rate fitness values. For the dense data set, one run with both a larger population and evolved over more generations was done in order to test if this improved the neuron firing rate fitness.

An overview of the results from the different runs is shown in table 4. The abbreviations used in the table are s for the number of time steps the model was run. This is equivalent to ms in the data set recordings from the biological neural network. Next, Pop is population size, Ad is the number of adults, Ch is the number of children, es is the early stopping value. FF is the fitness function used to evolve the model; T denotes time interval firing rate fitness function, and N denotes neuron firing rate fitness function. Next, NF displays the neuron firing rate fitness value of the best performing individual of the run. TF displays the time interval firing rate of the best performing individual of the run. Last, Time denotes the total run time of each run.

Table 4: Overview of runs of the different datasets.

Dataset	Run	#s	Pop	Ad	Ch	Gens	es	FF	NF	TF	Time
Small	SmT1	70	500	50	10	200/55	40	N	0.2896	**0.9977**	21m32s
Small	SmT2	70	2000	100	20	200	NA	T	0.3406	**0.9951**	6h33m
Small	SmN1	70	500	50	10	200	40	N	0.5400	0.7835	1h53min
Small	SmN2	70	2000	100	20	200	NA	N	0.8525	0.1842	6h31m
Sparse	SpT1	40	500	50	10	200/122	60	T	0.0012	0.4376	25m15s
Sparse	SpT2	40	500	50	10	200	NA	T	0.0012	0.3865	40m5s
Sparse	SpN1	40	500	50	10	200	NA	N	0.0197	0.0411	29m28s
Sparse	SpN2	40	500	50	10	200	40	N	0.0191	0.0416	37m24s
Dense	DT1	60	500	50	10	200/66	40	T	0.0264	0.9215	30m51s
Dense	DT2	60	500	50	10	200	NA	T	0.0265	0.8835	1h21m
Dense	DN1	60	500	50	10	200	NA	N	0.0346	0.1444	1h24m
Dense	DN2	60	1000	100	10	500/445	40	N	0.0510	0.0059	5h11m

3.1. ***Analyzing the performance of each run***

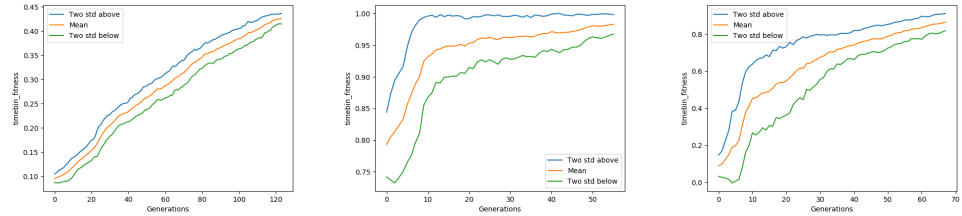
3.1.1. The mean fitness value of the population

Figure 1 shows how the mean fitness value of the population changes as the population evolves for each generation. In addition to the mean fitness value of the population, the blue and the green line represents two standard deviations above and below the mean fitness value of the population for each generation. For the regular runs (population: 500, generations: 200), the mean fitness value of the generation continues to improve as long as the population remains varied (non-homogeneous). The population stops improving when the fitness values of the individuals in the population become too similar.

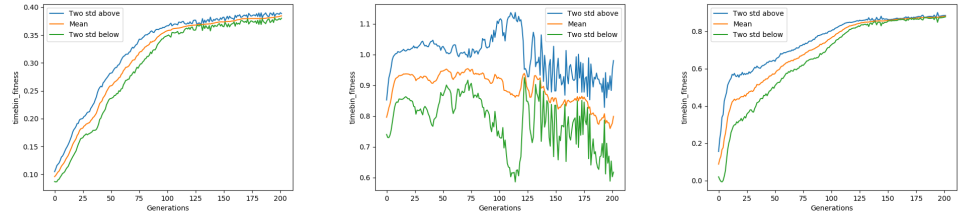
For the runs with early stopping, the evolution is stopped before the population becomes too homogeneous. However, for the runs without early stopping, we see that the population at some point becomes homogeneous almost without variation. When this happens the mean fitness value stops improving or only has slight improvements.

For the runs evolved either with a larger population or for more generations (SmT2 in figure 1e, SmN2 in figure 1k and DN2 in figure 1l), the population does not become homogeneous without variance. For SmT2 (figure 1e), the population variance remains large throughout the whole run. However, as we can see from the figure, the mean fitness of the population does not improve. Rather it declines. This indicates that even though the population is large. The GA is not able to combine the Network model parameters in such a manner that the system improves.

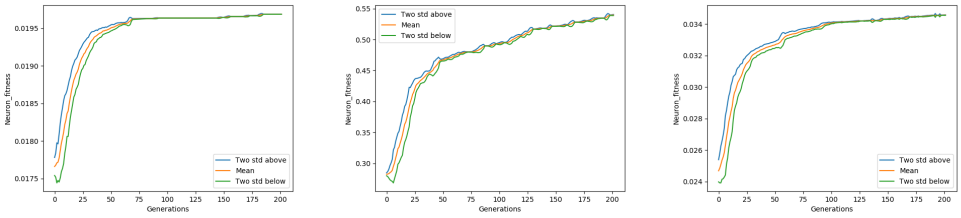
For SmN2 (figure 1k), the population is quite homogeneous with only small variances. However, the population never becomes completely homogeneous, and the mean fitness value of the population continues to improve. The graph also shows tendencies of step wise improvements.



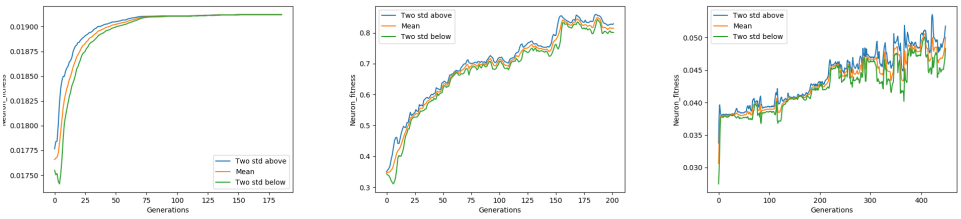
(a) Run: SpT1 (b) Run: SmT1 (c) Run: DT1



(d) Run: SpT2 (e) Run: SmT2 (f) Run: DT2



(g) Run: SpN1 (h) Run: SmN1 (i) Run: DN1



(j) Run: SpN2 (k) Run: SmN2 (l) Run: DN2

Figure 1: The mean fitness value of the population as the population evolves through the generations.

The model might have improved more if it was run for even more generations. However, this was not done due to time constraints.

DN2 (figure 1l) also has a homogeneous population with some "spikes" in the population variance. The mean fitness value improves as the population evolves. However, the fitness value improves very slowly.

3.1.2. A closer look at the neuron firing patterns and the time interval firing rates

Figure 2 shows the number of times each neuron fire during the Network model simulation in the first and the last generation, for the best performing individual in the generation, for each data set. The figure shows that for the sparse and small data set, the model is able to identify the most firing neurons. However, the neurons of the network model are too spiky

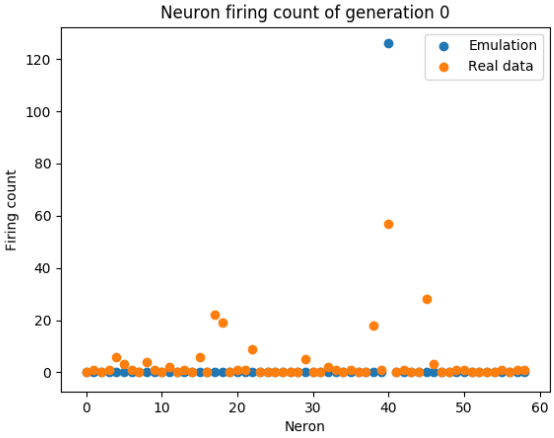
compared to the behaviour of the real data.

For the dense data set, the neurons have a sort of binary behaviour. The model is able to identify which neurons spike, and which do not. However, the neurons that spike are too spiky and they spike the same number of times.

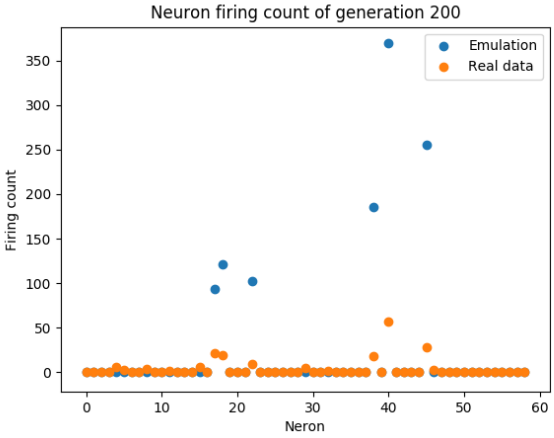
Looking at the runs with the time interval firing rate as the fitness function, the model is quite good at simulating the firing rates of the time intervals. Figure 3 shows how the firing rates of the time intervals improved from the first to the last generation of the best performing individual in the population.

As can be seen from the plots, the firing rates are somewhat random in the first generations, whereas it has the same pattern as the firing rates of the time intervals of the real data in the last generation.

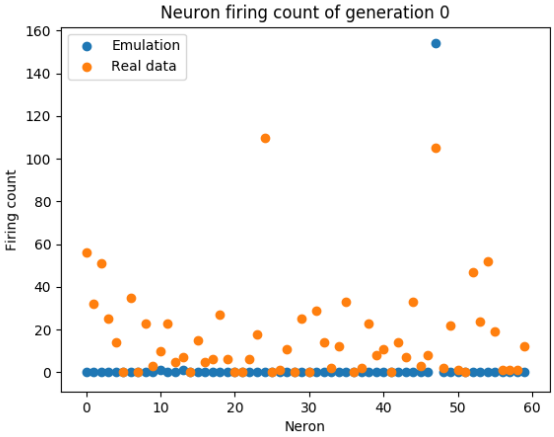
Further, when looking at figure 4 we see that the models fitting on the time interval firing rate also has individual neuron firing rates falling within the same neuron firing count interval as the recordings from the biological neuron network.



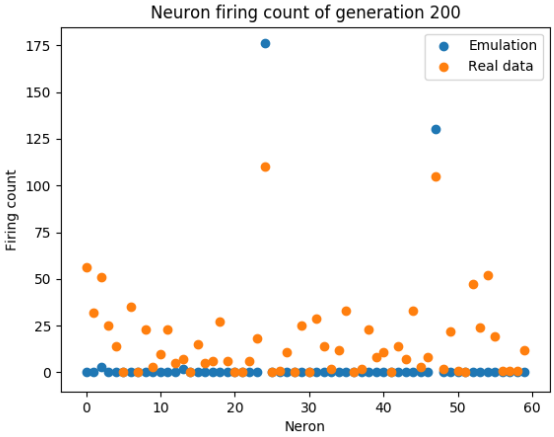
(a) Firing rate of generation 0 for run SmN2



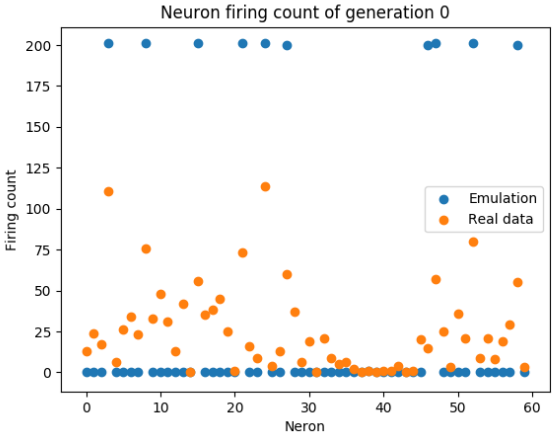
(b) Firing rate of generation 200 for run SmN2



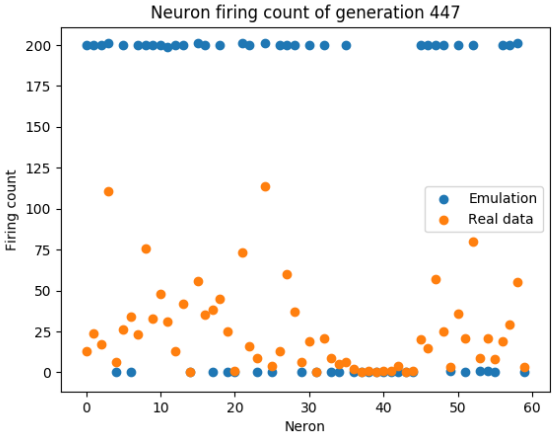
(c) Firing rate of generation 0 for run SpN1



(d) Firing rate of generation 200 for run SpN1

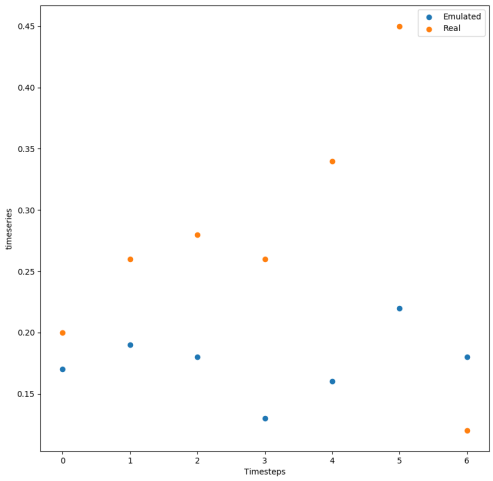


(e) Firing rate of generation 0 for run DN2

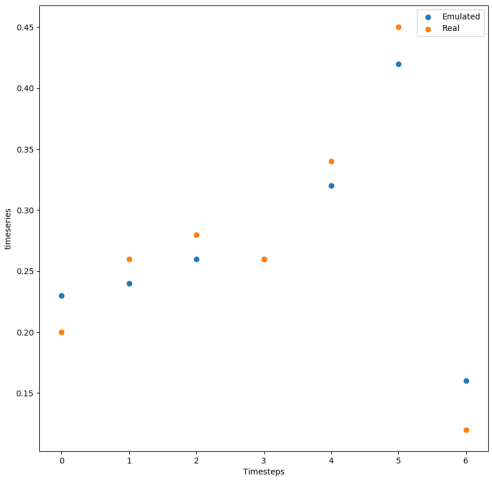


(f) Firing rate of generation 200 for run DN2

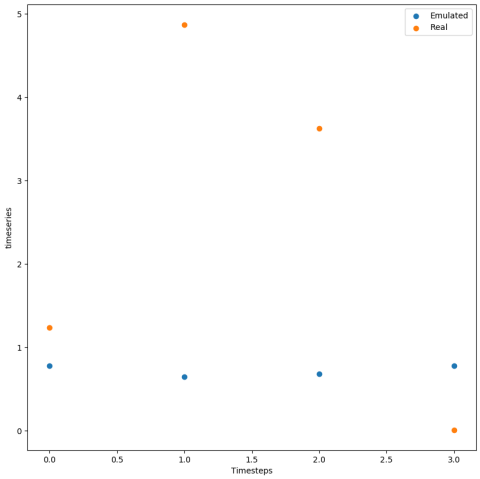
Figure 2: Neuron firing rates of the best individual of the best run on each dataset



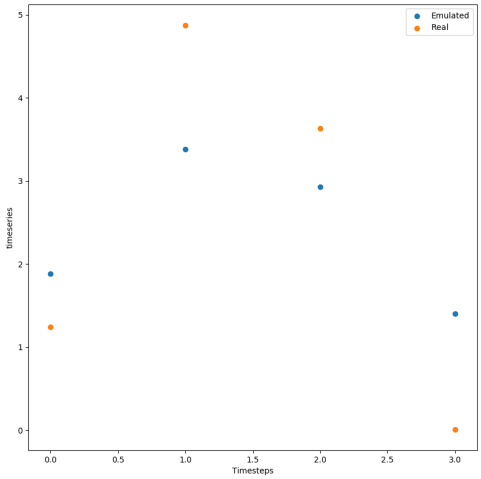
(a) Firing rate of generation 0 for run SmT1



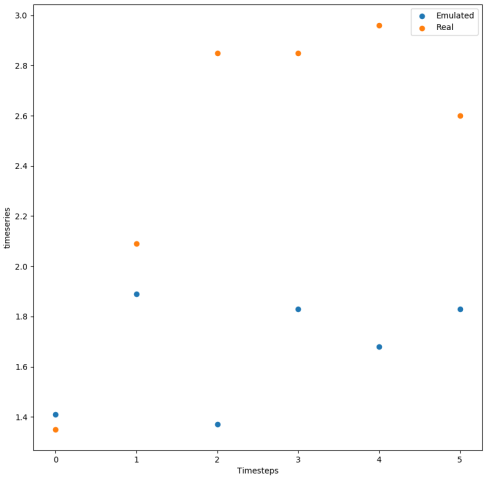
(b) Firing rate of generation 55 for run SmT1



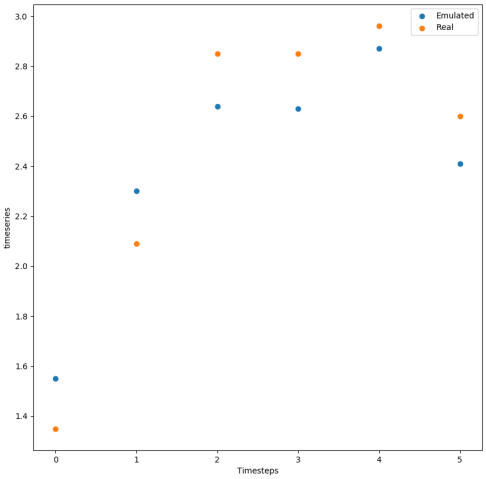
(c) Firing rate of generation 0 for run SpT1



(d) Firing rate of generation 123 for run SpT1

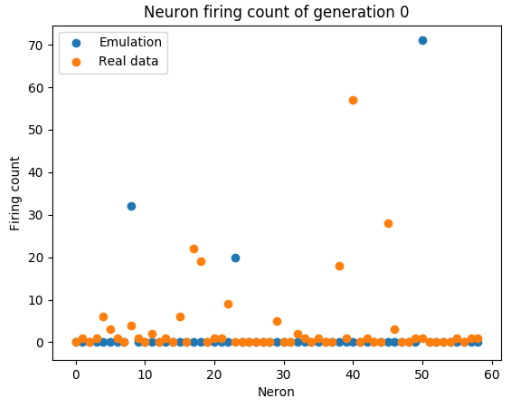


(e) Firing rate of generation 0 for run DT1

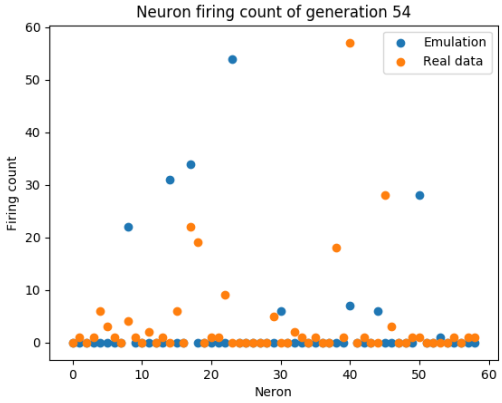


(f) Firing rate of generation 67 for run DT1

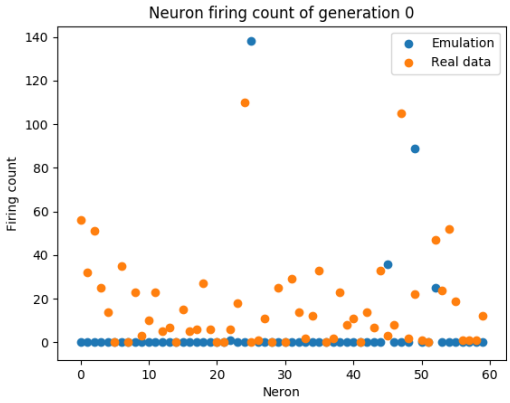
Figure 3: Time interval firing rates of the best individual on the best run of each data set.



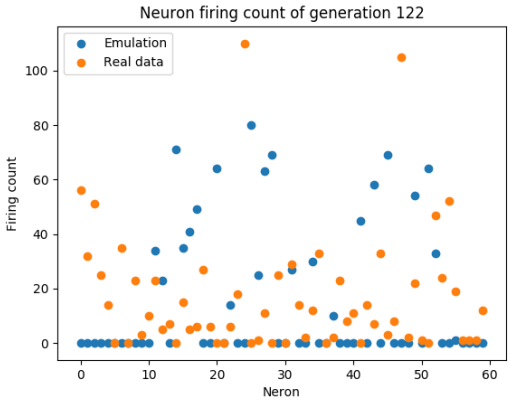
(a) Firing rate of generation 0 for run SmT1



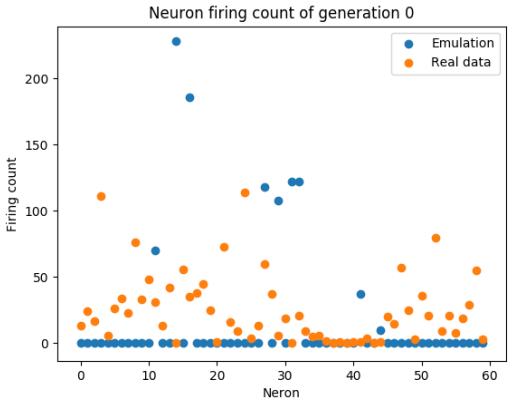
(b) Firing rate of generation 55 for run SmT1



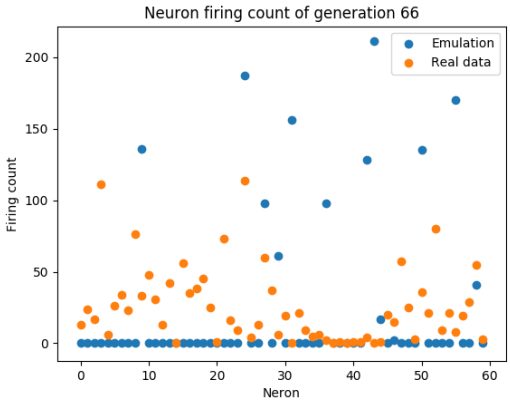
(c) Firing rate of generation 0 for run SpT1



(d) Firing rate of generation 123 for run SpT1



(e) Firing rate of generation 0 for run DT1

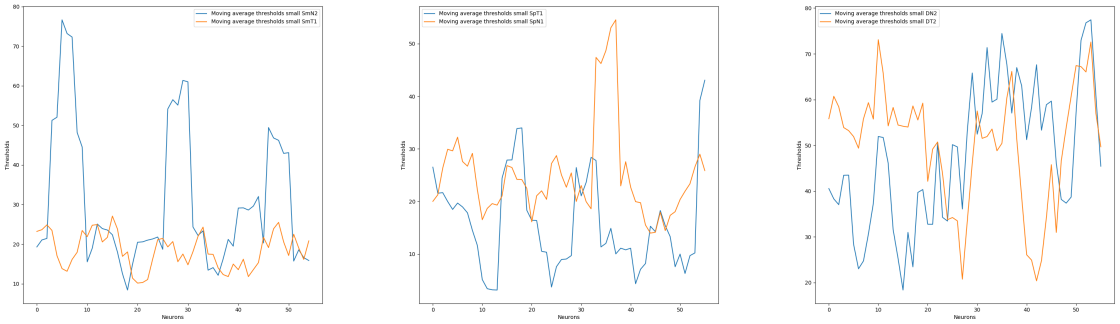


(f) Firing rate of generation 67 for run DT1

Figure 4: Individual neuron firing rates of the best individual on the best run of each data set from the run fitting on time interval firing rates.

3.1.3. A closer look at the weights and the threshold.

Figure 5 shows the moving average with a step size of 5 comparing the thresholds of the best performing individuals of each run (population: 500). For the small and the sparse data set, the run fitting on individual neuron firing rates (SmN1 and SpN1), the neuron thresholds yielded higher values than for the run fitting on the time interval firing rates.



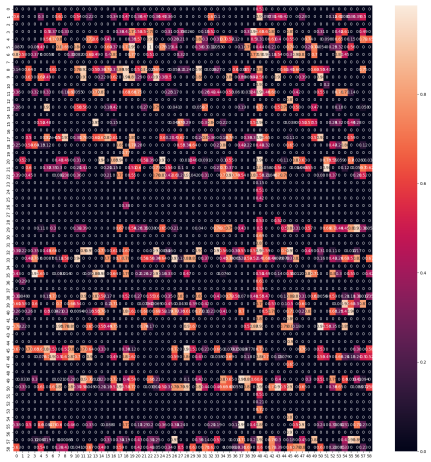
(a) Small data set thresholds. Runs: SmT1 (orange) and SmN1 (blue)

(b) Sparse data set thresholds. Runs: SpN1 (orange) and SpT1 (blue)

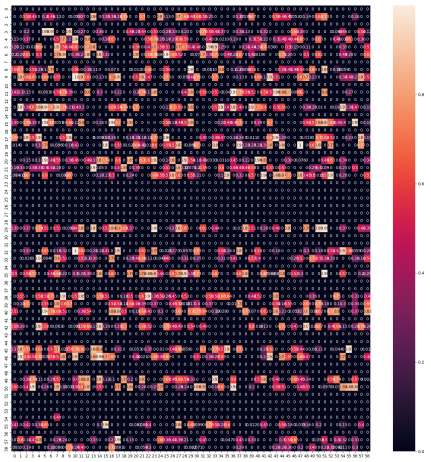
(c) Dense data set thresholds. Runs: DN2 (blue) and DT1 (orange)

Figure 5: Comparing the thresholds of the runs fitting on neuron firing rates and the runs fitting on time interval firing rates.

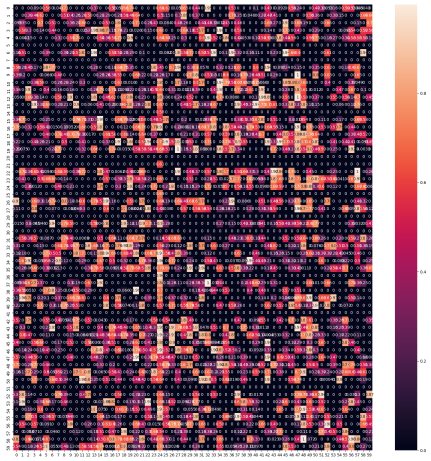
Figure 6 shows the weights of best individual for each data set for each fitness function (best run on population 500). The figure shows that the for the runs which are trained on the more dense data sets, the number of connections among the neurons increases. The horizontal black lines are the weight connections of the *Non spiky neurons*. The figure also shows a vertical line for the weight matrix from the runs trained on the individual neuron firing rate fitness function. This might explain the more spiky behaviour of single neurons.



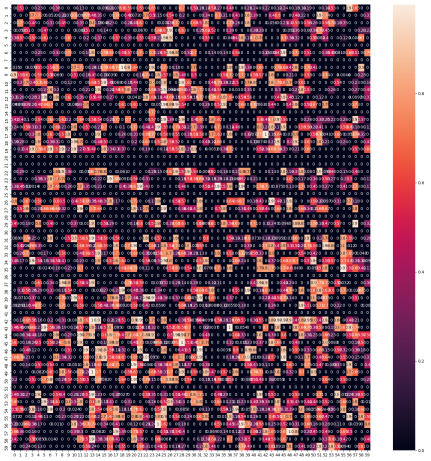
(a) Final weights of the best individual of SmN1



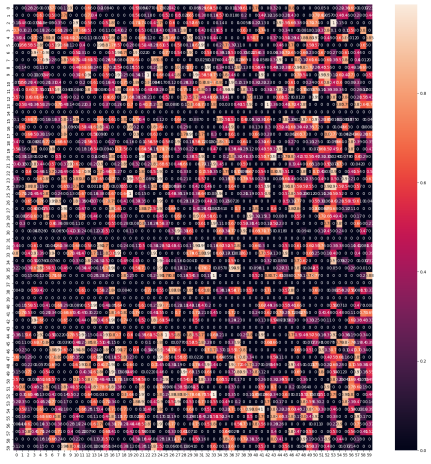
(b) Final weights of the best individual of SmT1



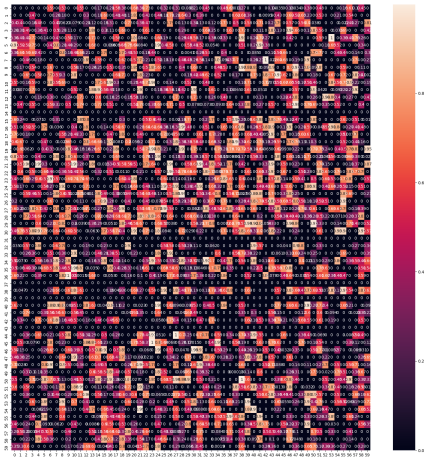
(c) Final weights of the best individual of SpN1



(d) Final weights of the best individual of SpT1



(e) Final weights of the best individual of DN2



(f) Final weights of the best individual of DT1

Figure 6: Weights of the best run for each data set for each fitness function.

4. Concluding remarks

In this section I discuss some conclusions to be drawn from this project. First, in order to evolve the population and reach better fitness values, there must be variance in the population. If the variance decreases and the population becomes too homogeneous, the model does not improve. The challenge is to tune the model parameters to maintain this variance.

In general, a larger population gives more variance, however this is more computationally expensive. Increasing the population size, might also require to evolve the model for more generations in order to take advantage of the larger variance in the population. An alternative might be to increase mutation-rates, but this must be done with caution in order to maintain a stable model.

The next point to notice is that it is easier to model the system across time, evaluating the network as a whole, rather than trying to fit the individual neuron firing rates. However, this might easier because fitting the firing rates of the time intervals means fitting the model by using fewer data points. On the other hand, the results showed that by fitting the model to the time interval firing rates, the range of the firing rates of the individual neurons were closer to the range of the biological individual neuron firing rates. Fitting the model by using the time interval firing rates was not able to detect which individual neurons fired the most. However, fitting on time interval firing rates was able to detect in which range the higher firing rates should fell.

Lastly, the challenge of evolving the thresholds and the weights to fit the spiking behaviour of the biological neuron network data is challenging. The complexity of the problem lies in the fact that the thresholds and the weights have to evolve together. Further complexity is added by the Reset Firing Neurons (intended to simulate the outer stimulus on the biological neural network). The effect of the Reset Firing Neurons on the system is a parameter I would have liked to further explore with more time.

5. Further research

There are several paths for further exploration which can be taken for this problem. First, in addition to Sigma scaling with roulette wheel selection, tournament selection was implemented in the code. However, in the early exploration, the tournament selection had poorer results than the Sigma Scaling, hence, the results were not included in this report. In addition, a fitness function which combined the individual neuron firing rates and the time interval firing rates was tested in the exploration phase. However, this fitness function proved to be unstable

and did not give any meaningful results, and was thereby not explored further. However, it might be that more patient parameter tuning could have made the combined fitness function useful.

In addition the knowledge of the model and how it simulates biological networks might be increased by implementing other types of fitness functions. Another approach for emulating features of a biological network might be to try to replicate the distribution of the firing rates. This approach is not as dependent on a one-to-one match between the firing rates of the neurons as the individual neuron firing rate fitness function. Rather it is able to emulate features that gives the same distribution of firing rates. This approach might not be able to describe the exact connections among the biological neurons (or electrodes in this particular case). However, this approach might be able to show characteristics of the typology of the network.

In addition, a Shannon entropy fitness function might give more insight into the firing patterns of the neurons throughout the simulation.

Due to computational limitations, only small fractions of time was simulated (from 5-8 seconds). Hence, the model is only able to evolve parameters that simulate the biological network for a short time period. Simulating the biological network for longer time periods might give different results.

Further, this project used 1ms as one time step. However, much might happen in a biological neuron network during 1ms. Hence, further exploration might test how the model responds to shorter time steps and if this improves the performance. This was however not done in this project as it was too computationally expensive.

References

[1] D. A. Wagenaar, J. Pine, S. M. Potter, An extremely rich repertoire of bursting patterns during the development of cortical cultures, BMC neuroscience 7 (2006) 11.

[2] K. L. Downing, Intelligence emerging: adaptivity and search in evolving neural systems, MIT Press, 2015.