Project 2:
Evolving Spiking-Neuron Parameters

Mikael Brevik

February 27, 2012

Contents

Chapter 1

Introduction

1.1 About the project

Purpose: Gain hands-on experience both a) evolving parameters for a single neuron, and b) quantitatively comparing neural spike trains.

I'll use my previously developed EA algorithm to evolve parameters for a single artificial neuron, based on a model by Eugene Izhikevich.

There are two aspects to this project: Implementing the Izhikevich model, and calculating the distance between two spike trains.

Chapter 2

Deliverables

2.1 Description of the system

This project is based on the previously developed EA from project 1 A and B. Except for some minor changes to the genetic functions (see section Minor changes), the code is identical.

As the system is highly object oriented, all the alterations and implementations spesific to any problem, is extended as sub classes of the main "framework" given by the EA library.

In this section, an explanation of all the different sub classes can be found, description of the genotype representation and fitness function.

2.1.1 Extensions to the original code

There are three extending classes that overwrites/extends standard behaviour; The individual, mutation and the plotting. In addition there is a new class for calculating the spike distance, with three different methods.

Indevidual: SpikingNeuron

This is where most of the magic happens. This class takes care of creating new genotype values (if not set), and converting from genotype to phenotype.

The constructor takes the gene size as an argument. This sets the number of bits each gene should have. (See Genotype representation). This is used to create a bit string genotype of the size $genesize * numberofparams$.

In addition to creating the random value the class also have a method for converting from genotype to phenotype, and a helper method for making each of the parameters fit inside of their given range (subsection 2.1.2).

The to phenotype function handles the calculations regarding the Izhikevich Spiking-Neuron Model.

Mutation: SpikeMutation

The mutation simply injects changes to the genotype by removing one random gene and replacing it with a new random generated one.

Plotter: SpikingNeuronPlotter

Added methods for plotting out the activation levels.

SDMs: Spike Train Distance Metrics

A new class for handling calculations of the spike distance. Methods are

1. Spike Time Distance Metric
2. Spike Interval Distance Metric
3. Waveform Distance Metric

These methods are used to calculate the inverted distance between two trains. The result of these methods are used as fitness. That means that the larger the distance is, the lower the result should be. Thats why the results are inverted by taking $1/distance$. This way a huge distance will become a very low fitness value, and vice versa.

In the SDM class, there's also a helper method named *compute_spike_times*(). This method takes a train as the argument, and iterates over the train in a window of size 5. The results are a generated list of the spike times in the train.

Helper functions and the main execution file

The main executon file handles all the interaction between the command line and the application. As with the previous project, all values can be changed through flags passed as arguments in the command line. Full overview of which flags are availeble, can be found bu running the main file, and attaching the $-h$ flag.

A helper function is created to read the data set. This method takes a number between 1 and 4, as argument and returns the data train set according to the argument number.

Minor changes

The crossover functions needed a fix, as they didn't take consideration for a genotype with genes larger then one bit. So the genes would be splitted up, and that destroyd the evolving.

2.1.2 Genotype representation

The genotype is represented as a bit string, as that what I first though of, and read out of the assignment diagram, also I have the genetic functions to use on binary values. I'd say it is a real-valued representation. Where there are 5 genes representing the parameters used by the model, and each of these genes are represented as binary strings.

When the model is to run, all the parameters are fitted to match the range for each of the parameters, according to the ranges defined in the project description.

1. $a \in [0.001, 0.2]$
2. $b \in [0.01, 0.3]$
3. $c \in [-80, -30]$
4. $d \in [0.1, 10]$
5. $k \in [0.01, 1.0]$

The first gene is parameter a, the second is b, and so on.

2.1.3 Fitness function

As described in section 2.1.1, the fitness is calculated by using the distance metric, and inverting it. The distance metrics, each work in different ways.

1. **Spike Time Distance Metric**: Takes in the two trains, converts both to spike times, sum up the total difference between corresponding spike times.
2. **Spike Interval Distance Metric**: Also converts both trains to spike times. With the spike times, it calculates the difference in the length of the time intervals between corresponding spike times.
3. **Waveform Distance Metric**: Does not use the spike times. Simply calculates the sum of the differences in the spikes.

2.2 Test Cases

Below the standard settings for all test cases can be seen. The system is runnning with full generational replacement as adult selection, and Tournament mechanism for parent selection.

For crossover and mutation, One-Point crossover and random value change of a gene is used.

```
std_values = {
    'output_file': 'spiking',
    'do_plot': True,
    'pop_size':  100,
    'mutation_probability': 0.25,
    'birth_probability': 1.0,
    'gene_size': 7, # The bit size for each gene (parameter)
    'generations': 200,
    'protocol': 'FullReplacement',
    'mechanism': 'Tournament',
    'reproduction': 'BinaryOnePointCrossover',
    'elitism': 0.04,
    'truncation': 0.05,
    'tau': 10.0,
    'I': 10.0,
    'timesteps': 1000,
    'spike_threshold': 35 # mV (milli Volts)
}
```

Listing 2.1: Default values for all params

2.2.1 Spike Time Distance Metric

Target Spike Trains	a	b	c	d	k	**Plots**
Spike-Train #1	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #2	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #3	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #4	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2

Table 2.1: Test cases for Spike Time Distance Metric

2.2.2 Spike Interval Distance Metric

Target Spike Trains	a	b	c	d	k	**Plots**
Spike-Train #1	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #2	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #3	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #4	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2

Table 2.2: Test cases for Spike Interval Distance Metric

2.2.3 Waveform Distance Metric

Target Spike Trains	a	b	c	d	k	**Plots**
Spike-Train #1	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #2	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #3	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2
Spike-Train #4	0.1	0.1	0.1	0.1	0.1	Figur 1, Figur 2

Table 2.3: Test cases for Waveform Distance Metric

2.3 Genotype-Phenotype Mapping

We have the genotype, which represents each of the parameters. The parameters again is used to calculate the phenotype. Thus, there is no direct interpretations from the genotype to the phenotype.

We have a sett of relativly small values as genotype, and we convert them to an array of length *timestep* (in our case 1000).

2.4 Practical Implications

This system can be used to reproduce and find the mathematical formula for simulating signals from neurons. So if a neuro-scientist uses this algorithm, sending in real data from a test, he/she could get the neccesery parameters to reproduce the result, mathematicly. This way a simulator can be built to test different implications of these signals, and so on.

2.5 Other Problem Domains

I'm not really sure what other problems this could solve. A more generic version could be used to find statistical models from graphs, or maybe be used to replicate/simulate electronic signals.

List of Figures