Project 2
IT3708 — Subsymbolic methods in AI

Didrik Jonassen, Imre Kerr

March 5, 2013

1 Description of Code

To begin with, we decided to do a major rewrite of the base EA system, based on things we had learned in the previous assignment. The system design is now much more modular and consistent. For instance, every functional unit used in the loop is now a function that accepts a population and returns a population, and they are all used in the same way (by passing them as parameters to the main loop).

1.1 Overview

The entry point of the program is the problem-specific script `izzy.py`. It generates all the functions needed by the main EA loop by calling *generator functions*, which prompt the user for parameters such as selection mechanisms, mutation rates etc. Some of the functions can be used as is, without decoration. It then gets a few more parameters (population size, max generations, fitness goal), generates the initial population, and passes all of this to the main EA loop (in `ea/main.py`). This loop runs until max generations or the fitness goal has been reached, and returns a list of the population for every generation. This list is passed to the `visualize` function, which plots the best spiketrain and the fitness graph using matplotlib. The parameter values and fitness of the best solution are printed to standard output.

1.2 Genotype Representation

Rather than using a bit-vector for the genotype, we opted to use a list of float values directly, one for each parameter value. This allowed us to use fancier mutation and crossover functions:

- **Gaussian difference mutation** — Adds a gaussian random value to each float with a given probability. The standard deviations of these values is given as a fraction of the ranges of the variables.
- **Uniform value mutation** — Sets each value to a uniformly distributed new value with a given probability.
- **Random choice crossover** — Crosses two genomes by randomly choosing values from one or the other.
- **Randomly weighted average crossover** — For each parameter p with parent values p_a and p_b, generates a random value $x \in [0, 1\rangle$, and sets $p = xp_a + (1 - x)p_b$.

1.3 Fitness and Development

Since we already have distance metrics, our fitness function is simply $\frac{1}{1+dist}$. This has the nice property of maximum fitness always being 1.0. The distance and development functions are basically given in the assignment text, so no further description of these is necessary. However, we did find that these were quite computationally intensive, and there was a lot of speedup to be had by adding multiprocessing support for these functions.

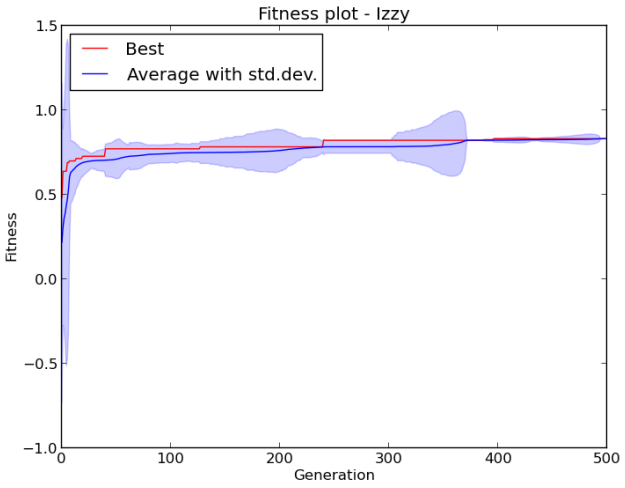
2 Test Cases

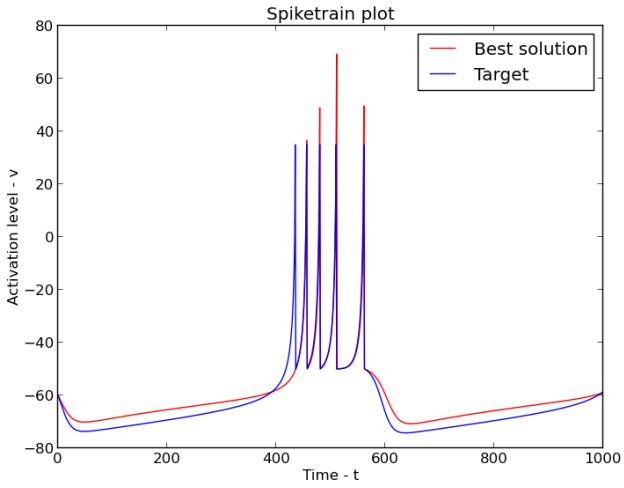
For all of these, the EA parameters are as follows unless otherwise noted:

Parameter	Value
Population size	100
Adult selection	Generational mixing
Litter size	100
Parent selection	Sigma scaling
Mutation type	Gaussian
Mutation rate	0.4
Mutation std.dev.	0.2
Crossover type	Random choice
Max generations	500
Fitness goal	1.0

2.1 Training Data Set 1

2.1.1 Waveform Distance Metric

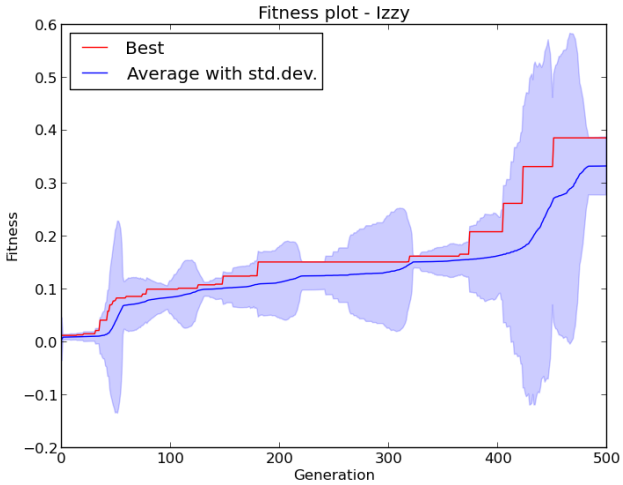


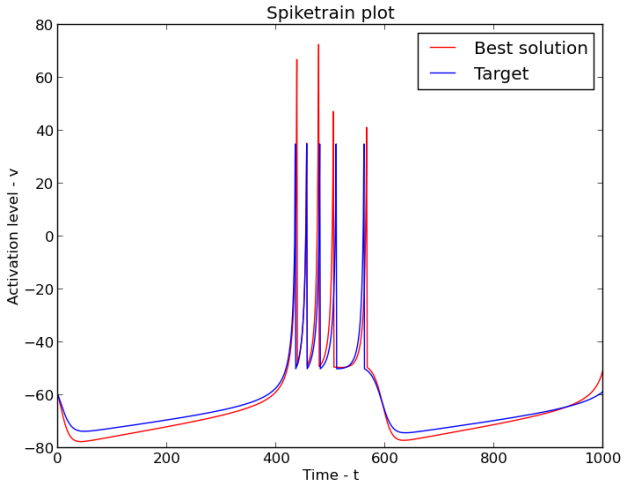


Best solution: a: 0.07191 b: 0.08117 c: -49.85 d: 2.162 k: 0.04047
Fitness: 0.8317

The waveform distance metric gives a pretty good result, but looking closely we see that the first spike is actually missing. This highlights a problem with the waveform metric. Since spikes are so narrow, a spike or two can be missing without too much penalty.

2.1.2 Spike Time Distance Metric

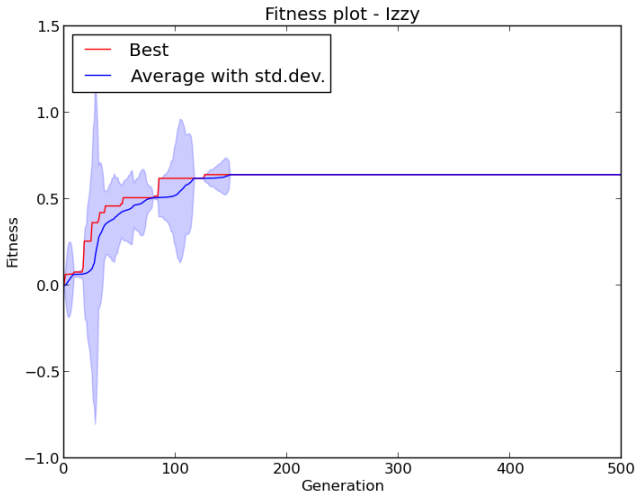


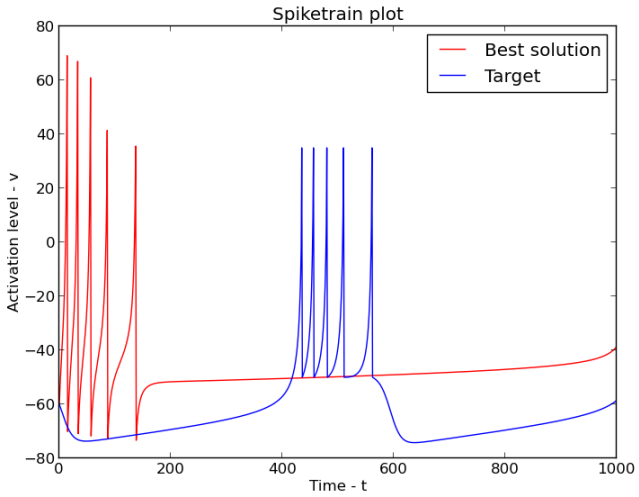


Best solution: a: 0.02418 b: 0.2438 c: -49.34 d: 2.506 k: 0.0393
Fitness: 0.3865

The spike time metric yields a very good result in this case. We had to up the mutation standard deviation to 0.5, otherwise we'd end up with some very odd spiketrains as the algorithm would stagnate at some poor local maximum. We think this is because spiketrains similar to the target occupy a very small part of the solution space, so you have to explore a bit to get something that's close enough to work from. Also note the fact that the fitness increases over the entire run, so there's probably some room for improvement if we let it run longer.

2.1.3 Spike Interval Distance Metric



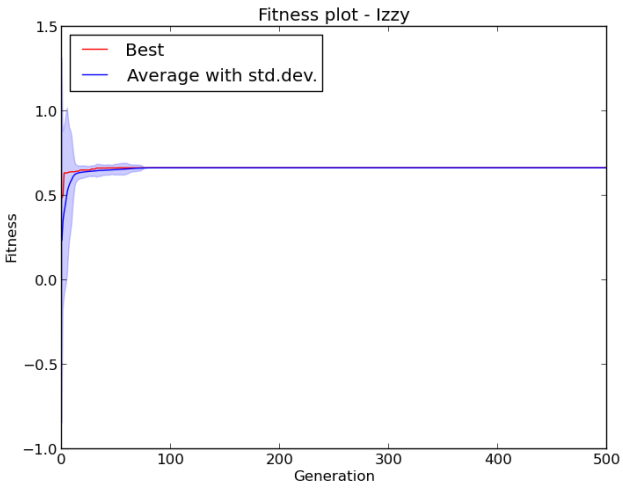


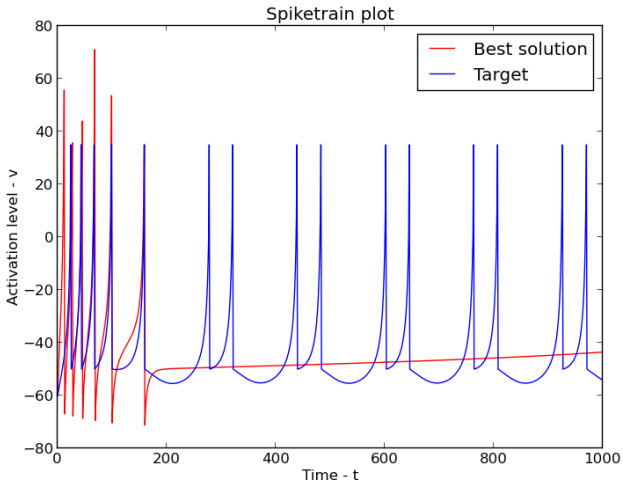
Best solution: a: 0.001 b: 0.01 c: -80 d: 8.2448 k: 0.0559
Fitness: 0.6414

Here we see one weakness of the spike interval method. It pays no attention to the position of the spikes, only the intervals between them. In this case, we therefore end up with a radically different spiketrain.

2.2 Training Data Set 2

2.2.1 Waveform Distance Metric

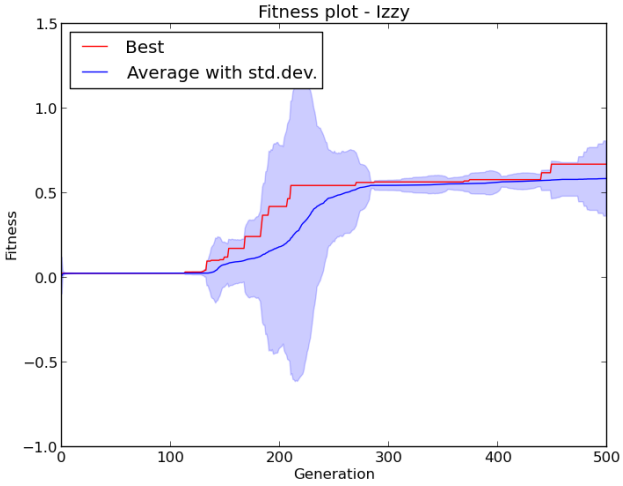


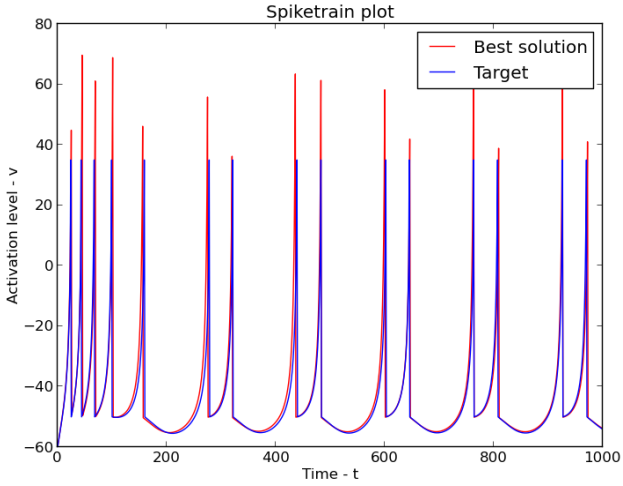


Best solution: a: 0.001 b: 0.1267 c: -80 d: 8.831 k: 0.0609
Fitness: 0.6656

This one starts out good, and then completely flatlines after about t=180. Looking at the bottoms of the spikes we see that these have a different shape from the ones in the target. This suggests that we may be stuck in the wrong part of the solution space.

2.2.2 Spike Time Distance Metric

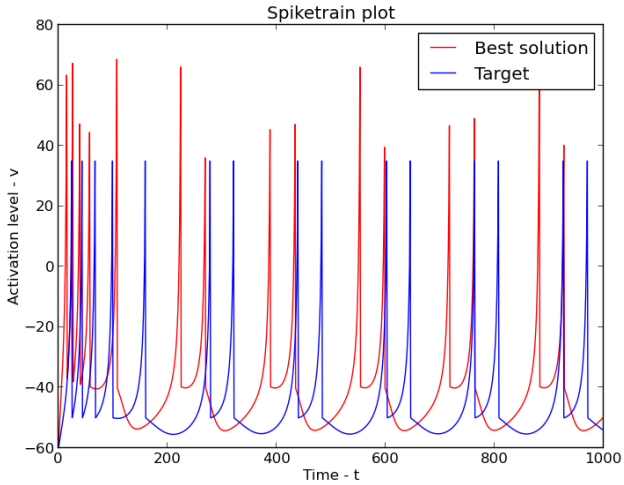




Best solution: a: 0.02456 b: 0.1832 c: -50.14 d: 3.909 k: 0.04510
Fitness: 0.6712

Very good solution, and still improving near the end of the run.

2.2.3 Spike Interval Distance Metric

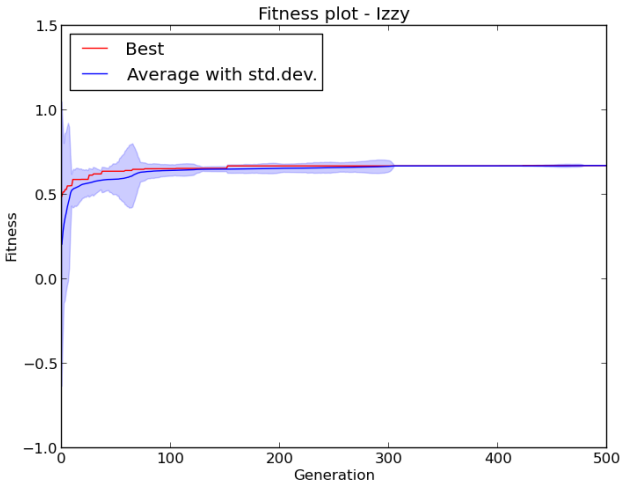


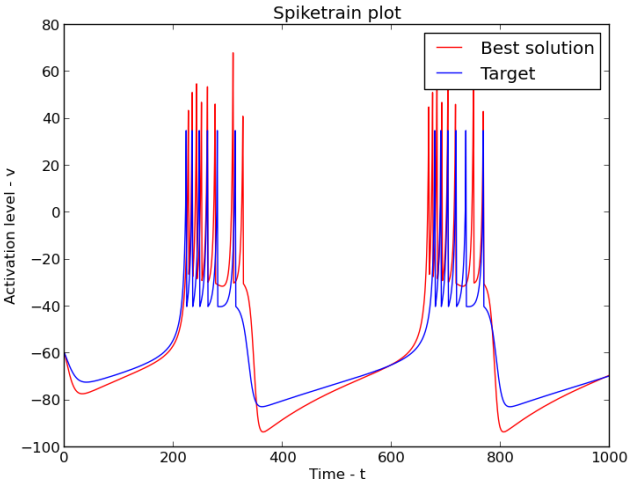
Best solution: a: 0.03495 b: 0.02956 c: -39.65 d: 10 k: 0.05349
Fitness: 0.3841

Once again the waveform is moved to one side, but (according to the eyeball test) it seems that this is a much smaller problem here.

2.3 Training Data Set 3

2.3.1 Waveform Distance Metric

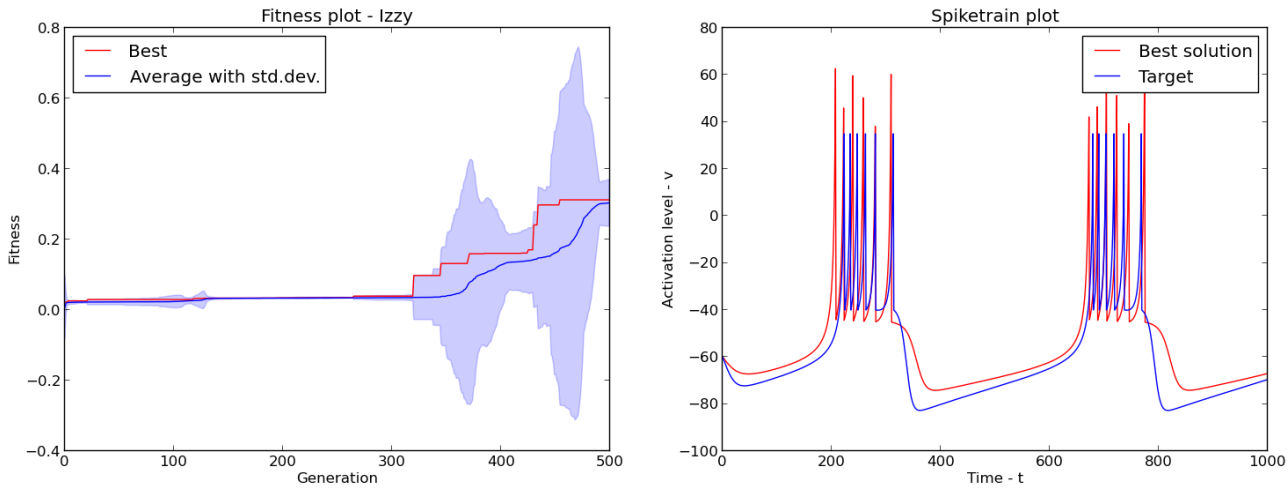




Best solution: a: 0.09712 b: 0.2148 c: -30 d: 9.589 k: 0.03891
Fitness: 0.6728

The number of spikes is of course all wrong, but the general shape of the solution is definitely there. The non-spiking portion of the target has a very characteristic shape here, which may be part of the reason why waveform works so well here.

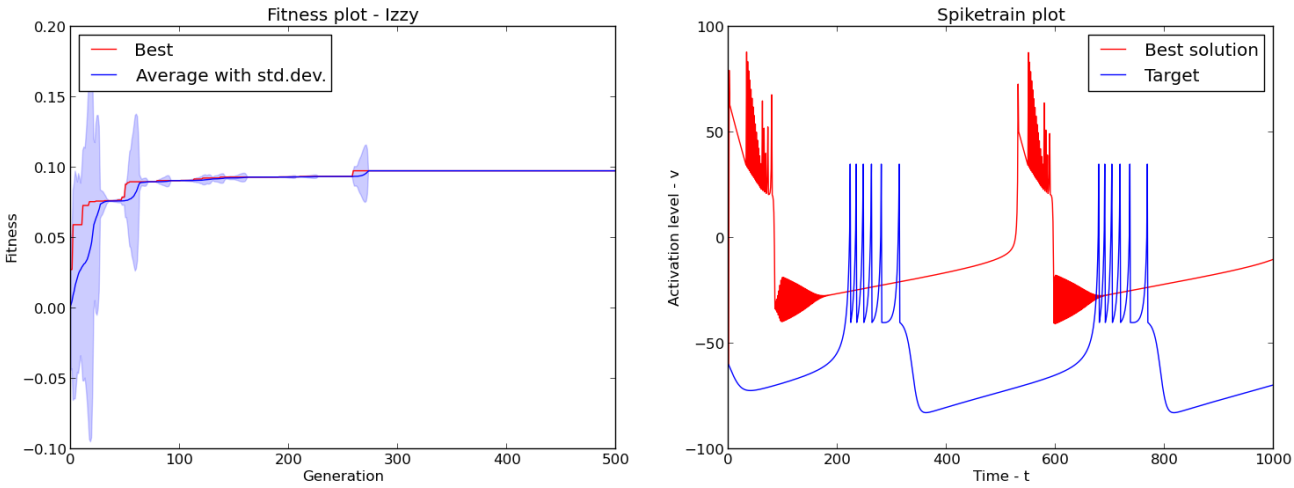
2.3.2 Spike Time Distance Metric



Best solution: a: 0.0334 b: 0.1712 c: -45.11 d: 2.6261 k: 0.04083
Fitness: 0.3125

We have the right number of spikes, and the general shape down. The fitness plot shows us we're nowhere near done, so this one has potential.

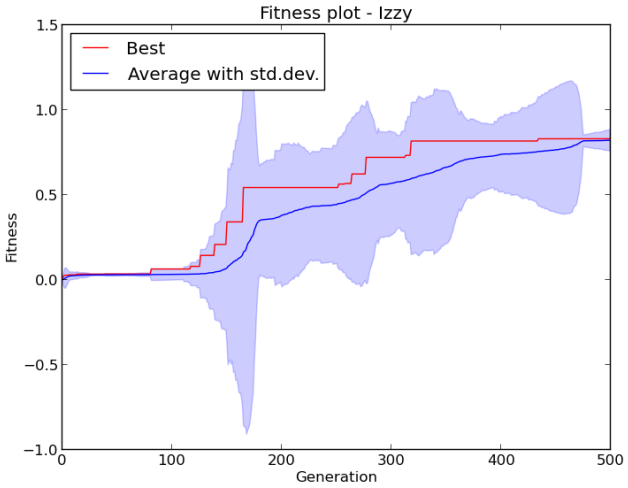
2.3.3 Spike Interval Distance Metric

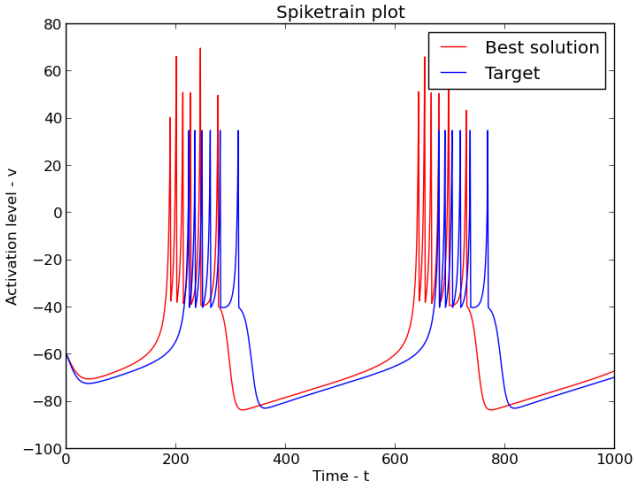


"Best" solution: a: 0.02664 b: 0.2838 c: -55.58 d: 9.989 k: 0.4288
Fitness: 0.09774

With the given SDMs, we were not able to get a good solution here. Notice the blocks of solid color, which we can only assume are periods of one spike every other tick. The triangular shapes near the top will probably be interpreted as a single spike, due to the window size being 5. Verdict: Not even close. If we were evolving movies, this one would be *Plan 9 from Solution Space*.

This all changed when we changed the SDM a little. Rather than using a window size of 5, we used 3, and suddenly it worked much better. In retrospect, we can't really see a good reason to use 5. Anyway, these were the results:

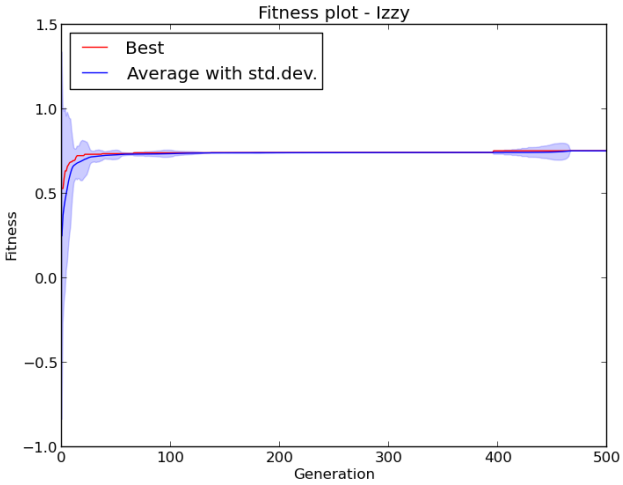


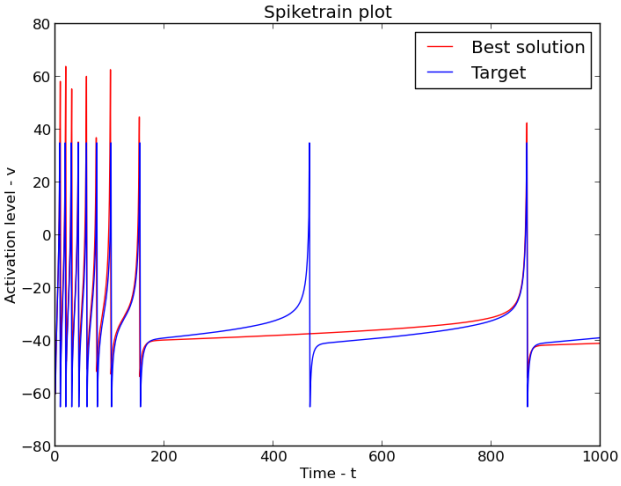


Solution: a: 0.03810 b: 0.2496 c: -39.14 d: 5.247 k: 0.04029
Fitness: 0.8311

2.4 Training Data Set 4

2.4.1 Waveform Distance Metric

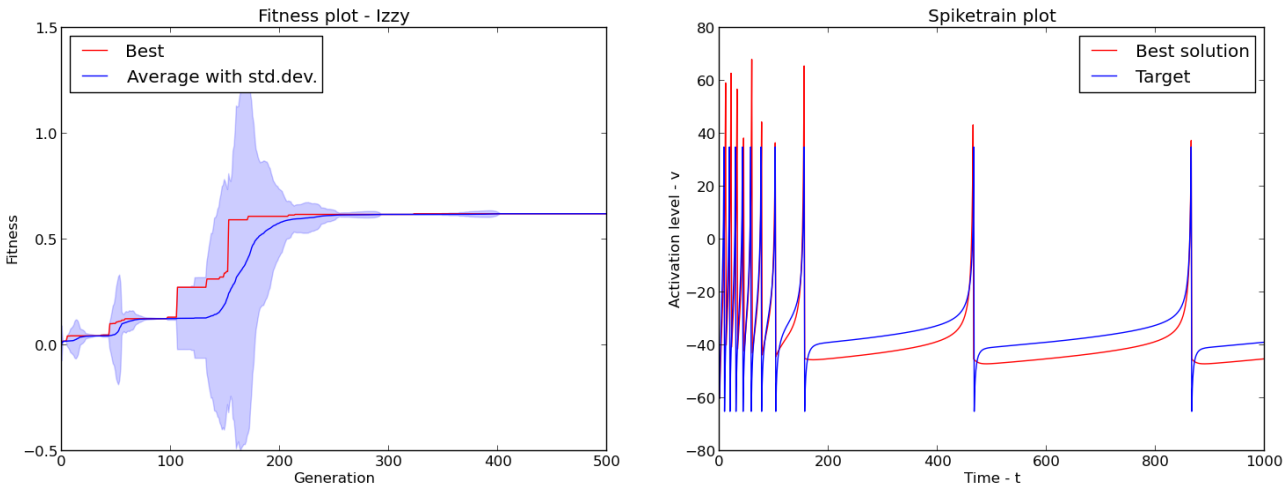




Best solution: a: 0.00132 b: 0.02804 c: -59.11 d: 9.810 k: 0.07971
Fitness: 0.7544

Not all that bad, but again the middle spike is missing.

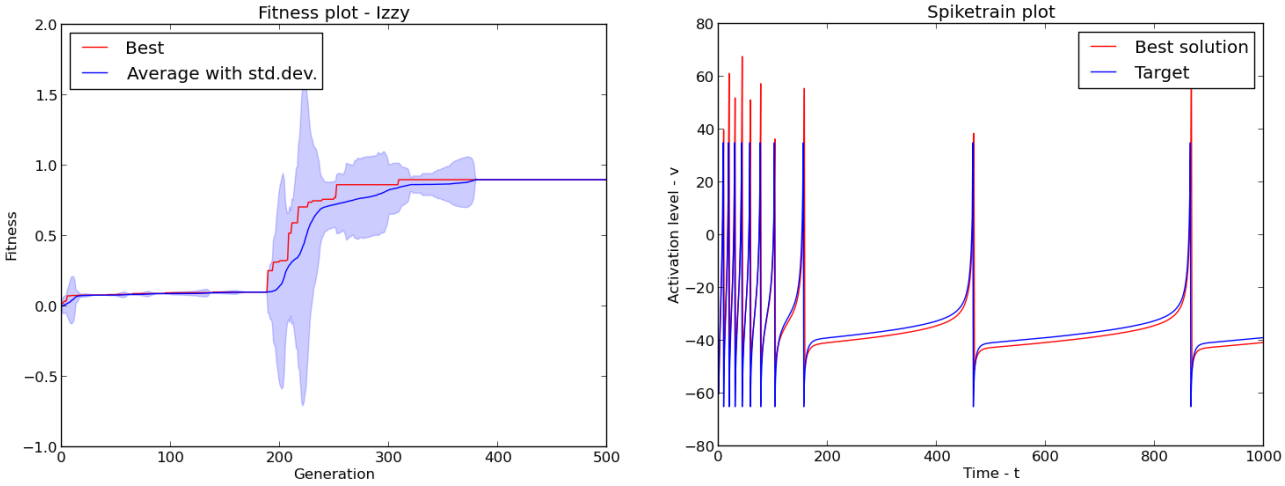
2.4.2 Spike Time Distance Metric



Best solution: a: 0.0031 b: 0.1058 c: -44.92 d: 7.577 k: 0.06569
Fitness: 0.6218

The spikes line up almost perfectly, but the bottoms have the wrong shape. We are unsure if this is relevant for further use in artificial neural networks.

2.4.3 Spike Interval Distance Metric



Best solution: a: 0.00304 b: 0.1951 c: -53.23 d: 9.326 k: 0.07541
Fitness: 0.9

Almost perfect. Once, while testing the code, this combination of SDM and test data attained a fitness of 1.0 in just 14 generations, so we can conclude that this SDM is very well suited for evolving this type of spiketrain.

3 Discussion

3.1 Genotype–phenotype Mapping

The mapping is data-oriented, since the values in the genotype are used as input data for a fixed algorithm. The developmental effort is quite high, since computing a spiketrain from the parameter

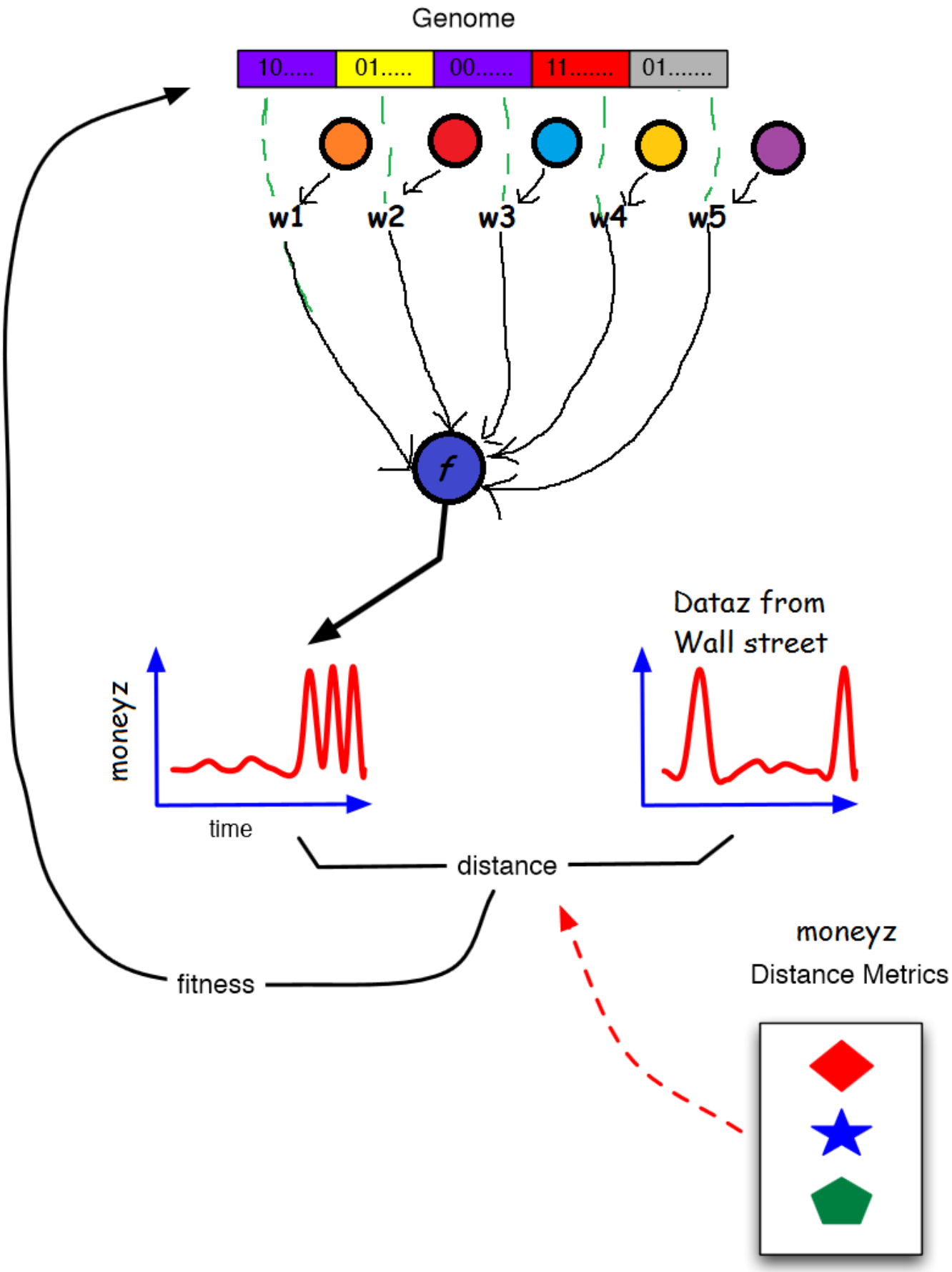
values requires doing a lengthy numerical computation. There is, however, high correlation between the genotype and phenotype, since a small change in the genotype doesn't change the phenotype much. This would be very different if we'd used e.g. a bit vector, where a single bit flipping can cause a huge change in numerical value, and hence the phenotype.

3.2 Practical Implications

Spiking is a phenomenon that is observed in certain naturally occuring neurons. Since computational neuroscience tries to mimic natural neural networks, it is useful to have a computational model for this type of behavior. If one knows that a naturally occuring neuron spikes according to a specific pattern, one could use this tool to evolve the parameters to accurately model it, and use these in an ANN simulation.

3.3 Other Problem Domains

Any system where you have a time-varying response based on some external output is a potential candidate here. The stock market is one example. One could create a model of the stock market as a function of some collection of external factors, with unknown weights for each factor, and then use this evolutionary algorithm to find these weights. One would then (possibly) have a good model that could be used to predict future stock market trends. The challenge would of course be coming up with the function. Obviously, spike-based distance metrics would not be applicable here.


Genome
10.....
01.....
00.....
11.....
01.....
w1
w2
w3
w4
w5
f
Dataz from
Wall street
moneyz
time
distance
fitness
moneyz
Distance Metrics