Implementing Hill Climbing with a Spiking Neural Network*

Aaron Russell Voelker[1]

Abstract— **The hill climbing algorithm, a technique for optimizing some function, is implemented using a neural network of spiking neurons. This network consists of three neural populations, encoding the position, velocity, and acceleration of an input vector. The vector is continuously evaluated by the function to generate an error signal with respect to the current velocity. This paper's main contribution is a heuristic which combines the error with oscillatory perturbations across an arbitrary basis, in order to drive the input in an exploratory manner. We show that this network converges to local optima for a number of test functions, and has interesting qualitative properties stemming from the choice of perturbations. This suggests that a similar heuristic may find use in non-neural implementations.**

I. Introduction

The hill climbing algorithm is an optimization procedure that searches for the minimum of some function, f, by evaluating candidates in the local neighbourhood of a vector $\mathbf{x}$, so as to greedily obtain incrementally better solutions [1].

Hill climbing has been used to optimize the parameters of Bayesian networks [2] and the connection weights of neural networks [3], but to the author's knowledge no work has yet attempted the reverse: implement the hill climbing algorithm using a neural network.

Within this setting, the function f represents some computation performed by a region of the brain, and the input is given by the spiking activity of a separate population of neurons. Since this input vector is a continuous time-varying signal, it is denoted $\mathbf{x}(t)$ to emphasize the importance of time. The goal is to drive $\mathbf{x}(t)$ so that $f(\mathbf{x}(t))$ is eventually minimized.

Given these constraints, we do not have the freedom to simply evaluate a discrete set of candidates before updating the current solution, as is the case for a conventional hill climbing implementation. Rather, the network makes use of continuous feedback to dynamically update the current solution based on the function's rate of change with respect to the input's velocity, $\mathbf{x}'(t)$.

This is a considerable disadvantage, since the available information is limited to a sample of the function's gradient along the chosen trajectory. However, this same constraint is faced by stochastic hill climbing, which evaluates only a single randomly chosen neighbour, and thus has the potential benefit of fewer function evaluations. Stochastic hill climbing has also been used to solve problems in reinforcement learning [4], and so it seems appropriate to adopt a similar approach when using continuous feedback.

The algorithm is carried out in the simulation of a spiking neural network built using the principles of the Neural Engineering Framework (NEF) [5]. These methods have been used to model tasks ranging from reinforcement learning to motor control, with quantitative comparisons to both behavioural and spiking data [6], [7], [8].

By constraining the model using NEF principles, this work may shed light on what strategies are feasible for the brain during cognitive tasks that can be formulated as a local search problem.

First, the model is described mathematically in terms of vectors and transformations of these vectors. Second, a brief overview is given on how the NEF compiles this description into a spiking neural network. Third, results are given for several test functions. Last, we discuss how this approach may inform a novel non-neural implementation, as well as some limitations and future directions.

II. Model

The model contains three neural populations encoding the position, velocity, and acceleration of the function's d-dimensional input vector, denoted $\mathbf{x}(t)$, $\mathbf{v}(t)$, and $\mathbf{a}(t)$, respectively. Position is recurrently connected to integrate velocity (so that $\mathbf{v}(t) = \mathbf{x}'(t)$), and similarly velocity integrates acceleration ($\mathbf{a}(t) = \mathbf{v}'(t)$).

The signal $\mathbf{x}(t)$ is continuously evaluated by some function $f : \mathbb{R}^d \to \mathbb{R}$, and crucially the implementation of this function is hidden from the rest of the system. dy tracks the function's rate of change across a 10ms interval. Similarly, the change in $\mathbf{x}(t)$, denoted $d\mathbf{x}$, is computed over a 10ms interval.

Intuitively, the quantity $(-dy)d\mathbf{x}$ represents what "force" (i.e. acceleration) should be applied to $\mathbf{x}$ to push it in a direction that will decrease error. If the current velocity is causing the error to decrease, then a force is applied in the direction of the velocity. Conversely, if the error is increasing, then a force is applied in the opposite direction, causing the input to reverse direction.

An immediate problem is that, since dy is a scalar, the error has no directional information apart from its sign. This causes $\mathbf{x}(t)$ to oscillate back and forth along the line $\mathbf{x}(0) + c\mathbf{v}(0)$, $c \in \mathbb{R}$.

A natural solution was to introduce periodic perturbations to the acceleration, in the form of a control $\mathbf{u}(t)$ term. This term emits a pulse lasting 20ms that alternates every 200ms between different vectors taken from some arbitrarily chosen basis for $\mathbb{R}^d$, denoted $\mathcal{B}$.

*Funding for this work was provided by the National Science and Engineering Research Council of Canada

[1]Aaron Russell Voelker is with the Centre for Theoretical Neuroscience, University of Waterloo, ON, N2L 3G1 arvoelke@uwaterloo.ca

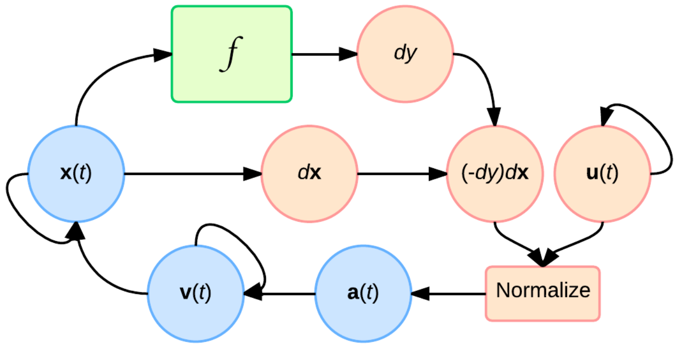

Fig. 1: (Network Architecture) $\mathbf{x}(t)$, $\mathbf{v}(t)$, $\mathbf{a}(t)$ are populations of neurons representing the input's position, velocity, and acceleration, respectively. f is the arbitrary feed-forward network being optimized. Remaining nodes are signal operations implemented in Nengo.

In effect, $\mathbf{x}(t)$ will move back and forth in one direction, while being periodically pushed in a new direction. Since $\mathcal{B}$ spans the vector space, it follows from standard results in linear algebra that this should converge to a local minimum, given appropriate scaling on $\mathbf{v}(t)$ and $\mathbf{a}(t)$[1].

The basis $\mathcal{B}$ can be understood as encoding some prior knowledge, or *intuition*, about the problem. These vectors alter the trajectory, regardless of the state of $\mathbf{x}$. If nothing is known about f, then we may let $\mathcal{B}$ be the canonical basis of $\mathbb{R}^d$, which reduces to testing rectilineарly adjacent points. However, if something is known about the directions that tend to minimize error, due to the structure of f, then it may be desirable to incorporate this knowledge into $\mathcal{B}$, in effect biasing the neighborhood of $\mathbf{x}(t)$. Furthermore, since $\mathbf{u}(t)$ merely perturbs the velocity relative to its current state, the neighbourhood's frame of reference will shift dynamically, an approach shared by dynamic hill climbing, which has the practical benefit of overcoming many common limitations of hill climbing [9].

Neurons encoding the position, velocity, and acceleration, were modeled using a Leaky-Integrate-and-Fire (LIF) circuit with an RC time constant of $\tau_{rc} = 20$ms and a refractory period of $\tau_{ref} = 2$ms.

Due to time constraints, the differentiation, computation of $(-dy)d\mathbf{x}$, and oscillatory behaviour of $\mathbf{u}(t)$, were all implemented by directly manipulating the vectors, rather than by use of neural nonlinearities. Thus, the network is not biologically plausible in its current form[2]. However, previous work has demonstrated that differentiation, multiplication, and oscillations of specific frequencies, are all possible using spiking neurons and NEF methods [10], [11].

III. Neural Engineering Framework (NEF)

The NEF is a set of biologically-motivated principles that provide methods for converting a mathematical description of a system (such as section II) into a network of spiking neurons. The following overview has been adapted from Stewart [12], which may be consulted for further details.

The first principle gives us a way to represent a vector, $\mathbf{x}$, in the spiking activity of a population of neurons. Each neuron i has an *encoding vector*, $\mathbf{e}_i$, which can be understood as the vector for which this neuron will fire most strongly. The principle states that the input current to the neural nonlinearity, G, is a linear function of $\mathbf{x}$, with coefficients given by $\mathbf{e}_i$, scaled by a gain of α_i, plus a bias current β_i. This gives a concise equation for each neuron's activity:

$$a_i = G[\alpha_i \mathbf{e}_i \cdot \mathbf{x} + \beta_i] \tag{1}$$

To decode the vector from this activity, we find a set of linear decoding weights $\mathbf{d}$ for each neuron, to approximate

$$\mathbf{x} \approx \sum_i a_i \mathbf{d}_i. \tag{2}$$

These weights can be found by optimizing a chosen error function. We typically minimize the root-mean-squared error using standard least squares solvers.

The second principle states that transformations of these vectors can be computed in the connection weights between populations of neurons. For instance, the connection weight from neuron i to j that computes the identity function is

$$\omega_{ij} = \mathbf{d}_i \cdot \mathbf{e}_j. \tag{3}$$

Interestingly, the same equation can be used to approximate any arbitrary function, by modifying the optimization problem given by equation 2 to decode a function of $\mathbf{x}$, i.e. by approximating $f(\mathbf{x}) \approx \sum_i a_i \mathbf{d}_i^f$.

The third principle provides a means for converting a control-theoretic description of a dynamical system into a recurrently connected population. For the purposes of this work, we are mainly interested in the simplest case of integrating an input vector:

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}. \tag{4}$$

By using an exponential model to filter the post-synaptic current, $h(t) = e^{\frac{-t}{\tau}}$ (decay time constant τ), it is easy to show that $\mathbf{x}$ can be represented according to equation 4 by setting the input to $\tau\mathbf{u}$, with a recurrent transformation approximating the identity function [13].

IV. Results

The model was built and simulated using Nengo [14], a software package for designing and simulating neural networks according to NEF principles. All figures and tables in this paper may be reproduced in full by running the Python code available on GitHub[3].

All simulations were performed on a conventional desktop computer[4] with a simulation time step of 1ms and 1,000 neurons per population (3,000 total).

[1]We scaled acceleration by 50 and velocity by 5

[2]Post-synaptic filtering was still modeled by convolving the feedback signal with an exponential filter with decay $\tau = 5$ms

[3]`http://github.com/arvoelke/hill-climbing`

[4]Intel® Core™ i7-4770 CPU @ 3.4GHz (64-bit)

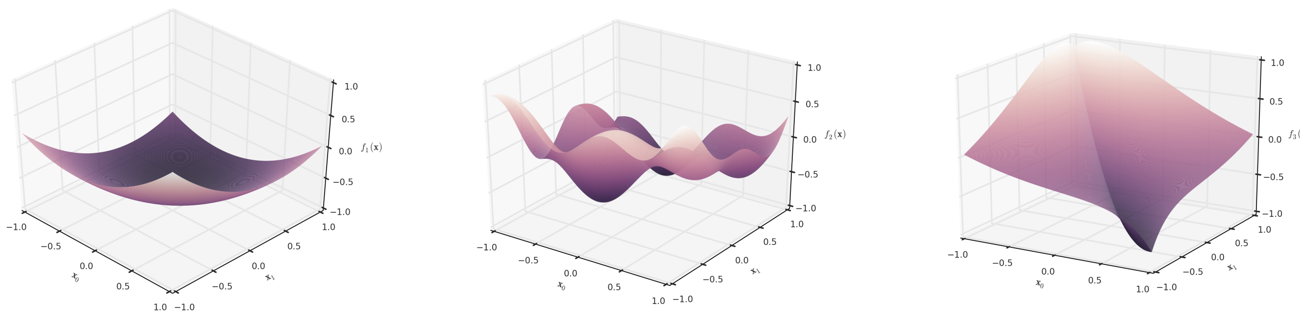

Fig. 2: Surface plots of the three test functions, corresponding to equations 5, 6, 7, from left to right, respectively.

$$f_1(x_0, x_1) = .5(x_0 + 0.3)^2 + .5(x_1 - 0.4)^2 - 1 \tag{5}$$

$$f_2(x_0, x_1) = .2\sin(6x_0) + .2\sin(6x_1) - .2\sin(2x_0)\sin(4x_1) + .4(x_0 + 0.3)^2 + .4(x_1 - 0.5)^2 - .5 \tag{6}$$

$$f_3(x_0, x_1) = .7((.5(x_0 - x_1) - .6)^2 - e^{-3|x_0+x_1|} - .3) \tag{7}$$

Three 2-dimensional functions were considered: a convex bowl (equation 5), a sinusoidal function with multiple peaks and valleys (equation 6), and a narrow *ridge function* that cannot be optimized by standard hill climbing (equation 7). For convenience, each function is defined over the interval $[-1, 1]^2$ and normalized to the range $[-1, 1]$.

Simulations were run for 3 seconds (3,000 time steps), with each trial randomizing α_i, β_i, $\mathbf{e}_i$ for all neurons (equation 1), and selecting a random set of basis vectors ($\mathcal{B}$). All model parameters, including the connection weights between neurons, were chosen independently of the particular function being optimized.

To quantitatively evaluate the algorithm, 100 trials were run on each function. Each trial recorded the best solution by decoding the neural activity (equation 2). Accuracy was calculated by taking the difference between this best solution and the actual global minimum. We also reported the simulation time by determining which time step obtained the minimum. Means are summarized in table I, with standard deviations given in parentheses.

TABLE I: Model Performance

Function	Accuracy	Simulation Time (s)
f_1	0.0021 (± 0.004)	$1.50s$ $(\pm 0.9s)$
f_2	0.0996 (± 0.030)	$1.68s$ $(\pm 0.9s)$
f_3	0.0308 (± 0.061)	$2.00s$ $(\pm 0.8s)$

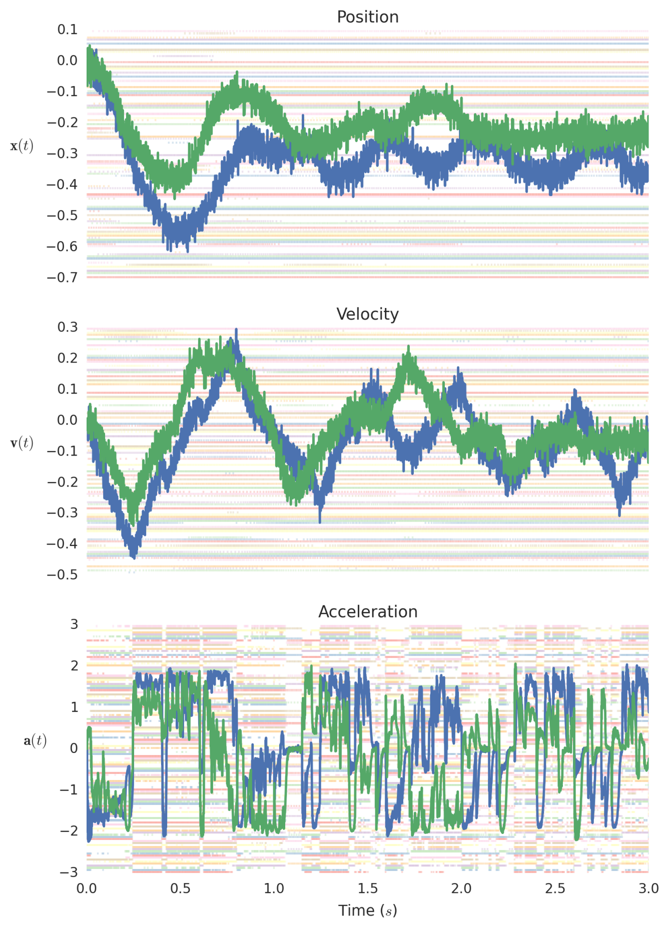

Fig. 3: Simulation data from a randomly chosen trial on f_2. The decoded and filtered activity shows that all three populations oscillate while repeatedly crossing a local minimum. Spike rasters of 120 randomly chosen neurons, taken from each population, are rendered in the backgrounds of their respective plots.

V. Conclusions

A. Discussion

The results from table I demonstrate that this method performs well on the chosen test functions. In particular, the global optimum for f_1 is located almost perfectly with very little standard deviation. The error is largest for f_2, due to the existence of a suboptimal minimum close to $(0, 0)$.

The high variability reflects a connection between $\mathbf{u}(t)$ and the algorithm's behaviour. Depending on the direction of the perturbations ($\mathcal{B}$), the solution may tend toward a different local minimum. This is also why the simulation times are highly variable – not all solutions follow the same path.

Similarly for f_3, there is a small probability that the chosen basis vectors are mostly orthogonal to the ridge, resulting in premature convergence.

Data from these simulations also provide qualitative insights into the relationship between the basis vectors and the

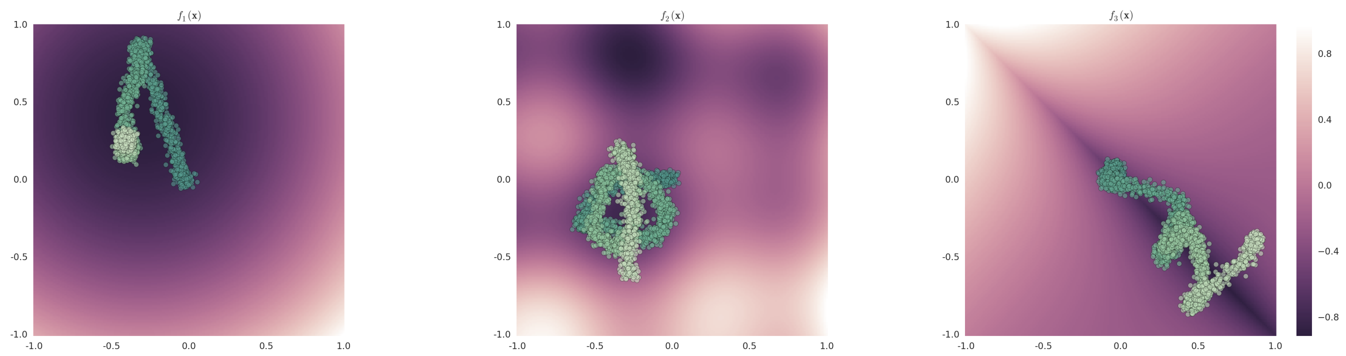

Fig. 4: Selected simulation for each test function, with the decoded estimate for $\mathbf{x}(t)$ overlaid to trace out the solution's path. (Left) The solution slowly circles around the minimum of f_1, since the region is convex with a gentle slope. (Center) The initial velocity sends $\mathbf{x}(t)$ to the nearest (suboptimal) local minimum of f_2. The basis vectors are made visible by a distinctive "X" pattern, while the solution oscillates over the valley. (Right) The initial perturbation is not orthogonal to the ridge, and so the solution saddles toward the minimum of f_3. The firing rates of the LIF neurons saturate at the radius of the unit cicle, causing $\mathbf{x}(t)$ to oscillate along the boundary.

search path, that depend on the function being optimized.

In general, the dynamic nature of the method, resulting from periodic perturbations to the velocity and noisy neural representation, allows it to overcome some common limitations of hill climbing, by continually considering new directions and occasionally making locally suboptimal decisions.

We conclude that constraining the implementation to operate using brain-like principles provided a solution with desirable characteristics for certain functions.

B. *Limitations and Future Work*

The math from section II may also be used to implement a hill climbing algorithm without directly using Nengo or the NEF. It would be interesting to compare this approach with other hill climbing extensions on a standard suite of functions. Likewise, the theoretical similarity between this approach and existing alternatives are not yet understood.

Limited analysis was performed on the effects of different basis vectors. Notably, greater performance may be achieved by allowing $\mathbf{u}(t)$ to depend on the state of $\mathbf{x}(t)$ and the magnitude of dy.

Currently, parts of the system are not biologically plausible. We believe this can be improved in future iterations of the model. More importantly, this work has not attempted to identify how the components of this system map onto neuroanatomically relevant brain regions. This undertaking would allow us to constrain the model using more detailed biological parameters and make specific predictions about what activity might be observed within these brain areas, during various cognitive tasks.

Acknowledgment

Aaron Voelker would like to acknowledge both Chris Eliasmith and the Centre for Theoretical Neuroscience for establishing the positive environment which disseminated many of the ideas necessary for this work. We would also like to sincerely thank Gladimir Baranoski for inspiring this particular thread of research while lecturing for his course, *On the Synergy between Computer Science and Biology*, at the University of Waterloo.

References

[1] Stuart Russell, Peter Norvig, and Artificial Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, 1995.

[2] Ioannis Tsamardinos, LauraE. Brown, and ConstantinF. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine Learning*, vol. 65, no. 1, pp. 31–78, 2006.

[3] Stephan Chalup and Frederic Maire, "A study on hill climbing algorithms for neural network training," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. IEEE, 1999, vol. 3.

[4] Hajime Kimura, "Reinforcement learning by stochastic hill climbing on discounted reward," in *Proceedings of the 12th International Conference on Machine Learning*, 1995, pp. 295–303.

[5] Chris Eliasmith and Charles H Anderson, *Neural engineering: Computation, representation, and dynamics in neurobiological systems*, MIT press, 2003.

[6] Daniel Rasmussen and Chris Eliasmith, "A neural model of hierarchical reinforcement learning," in *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, Paul Bello, Marcello Guarini, Marjorie McShane, and Brian Scassellati, Eds., Austin, 2014, pp. 1252–1257, Cognitive Science Society.

[7] Travis DeWolf and Chris Eliasmith, "The neural optimal control hierarchy for motor control," *The Journal of Neural Engineering*, vol. 8, pp. 21, 11/2011 2011.

[8] Chris Eliasmith, Terrence C. Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen, "A large-scale model of the functioning brain," *Science*, vol. 338, pp. 1202–1205, 2012.

[9] Deniz Yuret and Michael De La Maza, "Dynamic hill climbing: Overcoming the limitations of optimization techniques," in *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*. Citeseer, 1993, pp. 208–212.

[10] Bryan Tripp and Chris Eliasmith, "Population models of temporal differentiation," *Neural Computation*, vol. 22, pp. 621–659, 2010.

[11] Aziz Hurzook, "A mechanistic model of motion processing in the early visual system," Masters thesis, University of Waterloo, Waterloo, Ontario, 12/2012 2012.

[12] Terrence C. Stewart, "The neural engineering framework," *AISB Quarterly*, pp. 2–7, 2012.

[13] John Conklin and Chris Eliasmith, "A controlled attractor network model of path integration in the rat," *Journal of Computational Neuroscience*, vol. 18, pp. 183–203, 2005.

[14] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith, "Nengo: a python tool for building large-scale functional brain models," *Frontiers in neuroinformatics*, vol. 7, 2013.