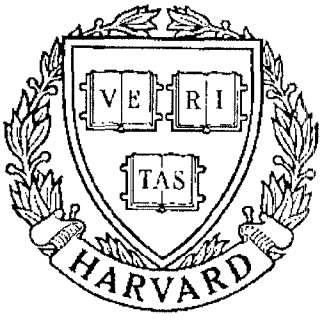
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

Fast Gravity: An n-Squared Algorithm for Identification of Synchronous Neural Assemblies

by J.E. Dayhoff

TR 92-99

Fast Gravity: An n-Squared Algorithm for Identification of Synchronous Neural Assemblies

Judith E. Dayhoff
Systems Research Center
University of Maryland
College Park, MD 20742

September 23, 1992

Abstract

The identification of synchronously active neural assemblies in simultaneous recordings of neuron activities is an important research issue and a difficult algorithmic problem. A gravitational analysis method was developed previously to detect and identify groups of neurons that tend to generate action potentials in near-synchrony from among a larger population of simultaneously recorded units. In this paper we show an improved algorithm for the gravitational clustering method. Where the original algorithm ran in n^3 time (n = the number of neurons), the new algorithm runs in n^2 time. Neurons are represented as particles in n-space that "gravitate" towards one another whenever near-synchronous electrical activity occurs. Ensembles of neurons that tend to fire together then become clustered together. The gravitational technique gives not only an identification of synchronous goroups present but also can be used for graphical display of changing activity patterns and changing synchronies among a larger population of neurons.

1 Introduction

Neurons generate action potentials - nerve impulses - that travel from the cell body to the axon terminals and signal motor, sensory, or other information within the nervous system. Experimental techniques for the recording of electrical activity of single and multiple neurons are advancing to the point where 20 to 100 or more neurons can be recorded simultaneously. Typically a multiple electrode array is implanted for the recording of neurons. Each signal channel carries an impulse train - a record of exactly when a single neuron fires an electrical impulse (an action potential) over a period of time. Sometimes signals from each electrode require pre-processing to obtain independent signal channels for each individual neuron. Analysis of these signals is required to evaluate specific hypotheses about how the nervous system works. Figure 1 illustrates simultaneously recorded nerve impulse trains.

Here we present a method that identifies groups of neurons that tend to

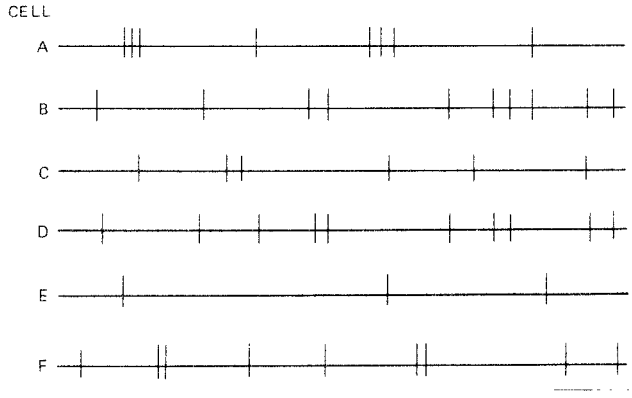


Figure 1: Simultaneously recorded nerve impulse trains. Reprinted from Dayhoff[4].

generate impulses in near-synchrony from among a larger group of recorded neurons. It is important to identify synchronous groups because groups of neurons furnish a candidate coding and representation scheme in the nervous system and in artificial neural networks. For example, a group of neurons that fire impulses in synchrony might represent a feature in a stimulus, such as a line in the visual field, a smell, or a particular sound, or might represent a movement pattern that is to be signaled to the muscles, or could be used in the internal processing of the brain.

As an example, in a recording of 20 neurons there might be a set of 5 neurons that tends to fire together in near-synchrony more often than expected at random. There may be another set of 3 neurons that also tends to fire together, but at different times from the first set. Each would represent particular pieces of information during neural processing. The task of the gravitational analysis program is to identify those subsets of neurons that tend to fire together given no a priori knowledge of which neurons would be in those subsets, or when the group would fire in synchrony.

In a classic gravitational clustering technique [3] as set of fixed vectors is to be analyzed for clusters. Each vector is represented as a particle in

n-space, with all having equal mass. The particles pull together over time due to gravitational forces. Those particles that aggregate first are identified as clusters.

In the gravitational clustering method for neurons [1], neurons are represented as particles in space that have an associated gravitational "charge". The charge varies over time and depends on the recorded electrical activity of the neuron. Whenever a neuron fires, the charge of the corresponding particle is incremented. Charges decay over time. If two particles have charge at the same time, they are pulled towards one another in proportion to the product of their charges. Particles representing neurons that tend to fire together then "gravitate" towards one another to form a cluster in n-space. The gravitational method for neurons may sometimes include both positive and negative charges [1]. In this case, pairs of like charges attract and opposite charges repel. Again, points representing neurons that tend to fire in near-synchrony gravitate towards one another.

The original algorithm for gravitational analysis of neurons, developed in 1982, ran in n^3 time, where n is the number of neurons [1]. The new algorithm presented here is n^2. The new algorithm utilizes an analogy to the center-of-gravity concept but is not the same as the real physical concept of center of gravity because of the use of a fictitious "charge" instead of gravity, the presence of two types of charges (positive and negative) and the use of a different law governing the amount of attraction as a function of distance.

2 Gravitational Clustering Method

2.1 Original Method for Neurons

First, each neuron is represented as a particle in n-space. Their positions are denoted

$$\mathbf{x_i} = (x_{i1}, x_{i2}, ..., x_{iN})$$

In the initial placement, the particles are equidistant, which requires at least n-1 dimensions. For convenience, we select n dimensions and place the particles at a distance of 1 from one another, at the following positions:

$$x_{i,j} = \begin{array}{ll} \sqrt{2}/2 & for\ i = j; \\ 0 & otherwise \end{array}$$

When a neuron fires, its corresponding particle acquires a portion of "charge" which decays over time. Additional firing that occurs later increments the charge again. The nerve impulse train can then be converted into a plot of charge versus time, as shown in Figure 2. An impulse in the data train could be implemented as any function g that eventually declines to decrease the influence of an individual impulse over time. Here we use exponential decay.

$$g(t) = ce^{-\tau t} + b \tag{1}$$

where τ is the decay constant, c is the amount of initial increment, and b is the baseline offset. Then

$$q_i(t) = \sum_i g(t - T_i)$$

where T_i is the time of the i^{th} impulse, and the sum is over all impulses i in the recorded impulse train. The value of τ is chosen to reflect the length of influence of an impulse, and thus reflects how close the impulses must be to be "considered" a near-synchrony. Longer values for τ allow longer delays between impulses in a synchronous firing.

Particles attract by a constant times the product of their charges, with an attenuation factor. At each time step,

$$\Delta \mathbf{x_i} = h\sigma A(S_{ij}) q_i \sum (q_j \mathbf{r_{ij}}) \tag{2}$$

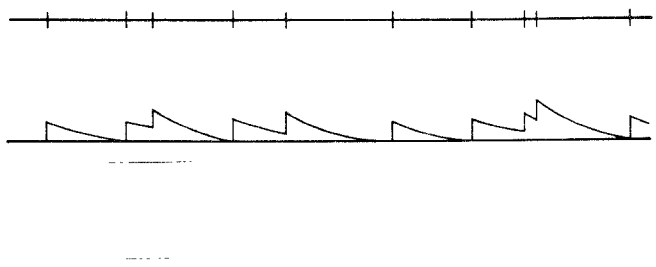


Figure 2: Conversion of a nerve impulse train to a charge versus time graph. Reprinted from Gerstein et al [1].

where h and σ are constants, S_{ij} is the distance from particle i to particle j, $A(S_{ij})$ is the attenuation function, q_i is the charge on particle i at the current time, and $\mathbf{r_{ij}}$ is the unit vector in the direction going from particle i to particle j. The value of h is proportional to the time step, and σ reflects the amount of movement per amount of charge.

Normalization is done to compensate for the fact that neurons with a higher than average frequency of firing tend to attract charges more than other neurons. With normalization, the value of c in the exponential equation above is $1/u$ where u is the average firing rate.

Negative charges are included to standardize motions so that two independently firing neurons do not move towards each other but on average stay at the same distance when the value of b in $g(t)$ is set appropriately. Thus we have both positive and negative charges on particles. Some particles may have zero charge at a particular time.

On each iteration, each pair of particles is moved towards or away from one another according to (2). Like charges attract; opposites repel.

Equation (1) has an attenuation factor, which could be set to any function of S_{ij}, the distance between units i and j. For convenience, we set

$$A(S_{ij}) = S_{ij} \tag{3}$$

Previous analysis has used $A(S_{ij}) = 1$, which has the disadvantage of causing instability when two particles are very close [1]. Setting $A(S_{ij}) = S_{ij}$ solves this "close approaches" problem so that two units that are very close together do not move past one another in the direction that pulls them together.

Note that because the unit vector

$\mathbf{r_{ij}} = (\mathbf{x_j} - \mathbf{x_i})/S_{ij}$

equation (2) becomes

$$\Delta \mathbf{x_i} = h\sigma q_i \sum_j q_j(\mathbf{x_i} - \mathbf{x_j}) \tag{4}$$

where h = the size of the time step, σ = a charge constant. Combining h and σ into one constant, we define $d = h\sigma$. Then

$$\Delta \mathbf{x_i} = dq_i \sum_j q_j(\mathbf{x_j} - \mathbf{x_i}) \tag{5}$$

On a dimension by dimension basis, (5) is equivalent to:

$$\Delta x_{ik} = dq_i \sum_j q_j(x_{jk} - x_{ik}) \tag{6}$$

3 Algorithms

3.1 Original Algorithm

The original algorithm works in n-cubed time, and is iterative, as follows:

1. For each particle i in n-space

   For each dimension $k(k = 1, 2, ..., n)$

   Compute

$$x_{ik,new} = x_{ik,old} + dq_i \sum_{j=1}^{n} q_j(x_{jk} - x_{ik}) \tag{7}$$

This algorithm is n^3 because there are n^2 iterations over the loop, which contains a sum from 1 to n.

3.2 Fast Gravity Algorithm

The basic idea of this algorithm is to calculate the center of gravity g and allow the particles to move with respect to the position of g instead of calculating pairwise movements, as specified in equation (1). In the case where all particles have positive charges, this operates similar to real gravity. Where there are both positive and negative charges, we need both a positive center of gravity g and negative center of gravity, g^c. Each positive charge moves towards g and away from g^c, and vice versa for negative charges.

CASE 1: No negative charges

1. Calculate

$$Q = \sum_{j=1}^{N} q_j$$

2. For each dimension k, compute

$$g_k = \sum_{j=1}^{n} q_j x_{jk}/Q$$

3. For each particle i, compute

$$\alpha_i = dq_i Q$$

4. For each dimension k

   For each particle i, compute

$$x_{ik,new} = x_{ik,old} + \alpha_i(g_k - x_{ik})$$

The algorithm is n^2. Step 1 is n. Step 2 is n^2. Step 3 is n. Step 4 is n^2.

Note that this algorithm is equivalent to calculating the center of gravity $\mathbf{g} = (g_1, ..., g_N)$ and moving each particle incrementally towards it. This is similar to the way that real gravity works, except that real gravity has a different attenuation factor that reflects the $1/r^2$ law. where r is the distance between particles.

CASE 2: Both negative and positive charges

1. Find the following sets:

S = the set of all neurons with positive charges.

S^c = the set of all neurons with negative charges.

Z = the set of all neurons with zero charges.

2. Compute

$$Q = \sum_{j \epsilon S} q_j$$

$$Q^c = \sum_{j \epsilon S^c} q_j$$

3. For each dimension k, compute

$$g_k = \sum_{j \epsilon S} q_j x_{jk} / Q$$

$$g_k^c = \sum_{j \epsilon S^c} q_j x_{jk} / Q^c$$

4. For each particle i, compute

$$\alpha_i = dq_i Q$$

$$\gamma_i = dq_i Q^c$$

5. For each particle i

For each dimension k, compute

$$x_{ik,new} = x_{ik,old} + \alpha_i(g_k - x_{ik}) + \gamma_i(g_k^c - x_{ik})$$

Steps 1 and 2 are each n. Step 3 is n^2. Step 4 is n. Step 5 is n^2.

4 Proof of Equivalence

In this section we show that the Fast Gravity algorithm produces the same computational results as the Original Gravity algorithm.

4.1 Case 1: No negative charges

According to Step 4 in the Case 1 algorithm above:

$$\Delta x_{ik} = \alpha_i(g_k - x_{ik}) \tag{8}$$

Plugging in for α_i, g_k, Q we get:

$$\Delta x_{ik} = dq_i(\sum_j q_j)[\sum_j (q_j x_{jk}) / \sum_j q_j - x_{ik}]$$

$$= dq_i[\sum_j q_j x_{jk} - (\sum_j q_j) x_{ik}]$$

$$= dq_i \sum_j q_j (x_{jk} - x_{ik})$$

which is the same as Step 1 in the original algorithm.

4.2 Case 2: Positive and negative charge

Begin with step 5 in Case 2 of the Fast Gravity algorithm:

$$\Delta x_{ik} = \alpha_i(g_k - x_{ik}) + \gamma_i(g_k^c - x_{ik}) \tag{9}$$

Plug in $\alpha_i, \gamma_i, g_k, g_k^c$, as follows:

$$\Delta x_{ik} = dq_i \sum_{j\epsilon S} q_j (\sum_{j\epsilon S} q_j x_{jk} / \sum_{j\epsilon S} q_j - x_{ik}) + dq_i \sum_{j\epsilon S^c} q_j (\sum_{j\epsilon S^c} q_j x_{jk} / \sum_{j\epsilon S^c} q_j - x_{ik})$$

Rearrange:

$$\Delta x_{ik} = dq_i (\sum_{j\epsilon S} q_j x_{jk} - x_{ik} \sum_{j\epsilon S} q_j) + dq_i (\sum_{j\epsilon S^c} q_j x_{jk} - x_{ik} \sum_{j\epsilon S^c} q_j)$$

$$= dq_i (\sum_{j=1}^{n} q_j x_{jk} - x_{ik} \sum_{j=1}^{n} q_j)$$

$$= dq_i [\sum_{j} q_j (x_{jk} - x_{ik})]$$

which is the same as Step 1 in the original algorithm.

5 References

1. G. L. Gerstein, D. H. Perkel and J. E. Dayhoff, 1984. Cooperative firing activity in simultaneously recorded populations of neurons: Detection and measurement. J. Neuroscience 5 (4): 881-889.

2. B. G. Lindsey, R. Shannon, and G. L. Gerstein, 1989. Gravitational representation of simultaneously recorded brainstem respiratory neuron spike trains. Brain Research 483: 373-378.

3. W. E. Wright, 1977. Gravitational clustering. Pattern Recognition 9: 151-166.

4. J. E. Dayhoff, 1990. Neural Network Architectures: An Introduction. New York: Van Nostrand Reinhold.