1994 International Symposium on Speech, Image Processing and Neural Networks, 13-16 April 1994, Hong Kong

An Application of the Discrete Fourier Transformation in Simulating Large Neural Networks*

Irwin K. King
king@cu.cuhk.hk

Department of Computer Science
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong

Abstract

This paper presents an application of the discrete Fourier transform (DFT) to calculate neural activities efficiently in simulating large biologically motivated neural nets. The experimental results demonstrate the DFT technique is more superior in performing calculation of the neural activity which reduces the time complexity to a theoretical order of $O(n \log_2 n)$, n being the number of neural units at each iteration. Our study also found that although the computational speed is improved drastically, there are tradeoffs involving (1) the error generated from the transform, (2) initial setting up time, and (3) the memory storage requirement when using the DFT algorithm. More specifically, we outline criteria and conditions under which the DFT method will yield optimal results in large software neural simulations.

1 Introduction

When modeling biological neural nets, one often requires to deal with large numbers of neurons to approximate real cortical tissues. One of the most fundamental operations is to calculate the neural unit's membrane potential and its firing rate according to an update rule. For a Leaky-Integrator type neuron, this equation is defined as

$$\tau \dot{x} = -x + S + h_0 + \xi \qquad (1)$$

where τ is a RC time constant controlling the rate at which the neural activities rise and fall, S is the set of inputs to the neuron, h_0 defines the resting potential of the neuron, and ξ expresses the possible global noise term.[1] Although Eq. 1 is expressed in the differential equation form, the simulation of the continuous equation in digital computers needs to be discretized into small time intervals. One of the more popular and easily implemented methods to approximate Eq. 1 is to use the Euler method which transforms Eq. 1 into

$$\begin{aligned} x(t+\Delta t) &= -(\Delta t/\tau)x(t) + (\Delta t/\tau)S + \\ &\quad (\Delta t/\tau)h_0 + (\Delta t/\tau)\xi \end{aligned} \qquad (2)$$

where Δt controls the accuracy of the simulation; here, smaller values of Δt yield better approximations than larger ones.[2]

A straight forward approach to solve this summation problem will add up the connection by using a **For-Next** type loop with offset indices to calculate the result with little optimization. This method results in longer computation time since this type of calculation is performed over every mask matrix location for each position on the layer. The time complexity is thus $O(M \cdot N)$ where $M(= m \times m)$ is the array size of the layer and $N(= n \times n)$ is the array size of the mask. Hence, in order to simulate larger and longer neural interactions one must first seeks more efficient and effective ways to reduce the processing time of this crucial step.

When the summation of the input from the weighted mask is decoupled from Eq. 1, the sub-expression is comparable to a spatial convolution operation. Hence, to speed up the neural simulation, one of the ways is to employ parallel computation units that make concurrent calculation via hardware connections with neighboring cells. Others have used optical electronic components to implement algorithms to achieve fast parallel computations [6]. Nonetheless, the DFT method is cost-effective and easy to implement. Furthermore, its characteristics are well studies; hence, the implementers have the full control of the trade-offs. Of course, the above observations are made with the assumptions that (1) the neural network has a static connection pattern, i.e., no plasticity involved and (2) the basic operation of the calculation is the convolution operation.

*The research described in this paper was supported in part by a grant from the Center for Neural Engineering at USC, Los Angeles, CA (M. A. Arbib, principal investigator).

[1] The noise term, ξ, is typically small and can be ignored. However, it is often introduced to test the robustness of the model.

[2] There are a variety of interpolation methods available for the approximation of differential equations. Some popular techniques are the Euler's method and Runge-Kutta method [4]. Specific numerical methods for neuronal modeling are found in Mascagni [5].

2 Implementation

When the input size of the simulation array is $n = 2^k, k = 0, 1, 2, \cdots$, then there is an $O(n \log_2 n)$ optimal algorithm to evaluate the DFT and its inverse by recursively applying the divide-and-conquer strategy [1]. One of the earlier papers implementing the DFT in an efficient way is found in Cooley and Tukey [3]. The basic idea behind the DFT algorithm is to transform the DFT of length N into the sum of two DFTs, each of length $N/2$. This way, there are $\log_2 N$ recursion steps and N operations in each recursion which yields a total of order $N \log_2 N$ operations. With this discovery, the transform is made easily implemented when the size of the array to be transformed is of power of 2.

Once the input signal (neural layer) and the response function (connection mask) have been Fourier transformed, the complex multiplication between these two layers is then performed. This multiplication in the frequency domain is a point-wise operation which can be completed with N multiplication and N assignment operations. When the complex multiplication is finished, this data set is then inverse transformed to achieve the final result in the spatial domain. The inverse DFT is done with the same algorithm as in the forward DFT algorithm with minor modifications in the input since the complex conjugate is required.

3 Experimental Setup

We implemented the DFT algorithm directly from Press [7] with minimal modifications. The program is written in C on Unix workstations for portability which conforms to the minimum requirement of C without using any operating system dependent function calls.[3]

We take a 1-D array simulating a single column of neural units in a layer of cells with the size corresponding to $2^k, k = 0, 1, 2, \cdots, 14$, in our timing evaluation runs. Each neural cell in the data array stores a randomized double float value $\zeta, 0 \leq \zeta < 1$. Equally, the mask is also produced by the same set up procedure as in the case of the neural cells. For trials with array size $k \leq 7$, the result from each run is based on the average of 500 sub-trials for 1 trial to average out possible significant deviations due to the small execution time on a single trial.

4 Performance Comparison

Performance comparison of these two approaches needs to be demonstrated in the space (how much storage is needed) domain and the time (how fast is the execution speed). Moreover, we will also discuss the errors generated from the DFT approach.

4.1 Storage Requirement

The DFT method requires extra memory space to store intermediate transformed values and the imaginary part even though all input data is real. Hence, it would need at least $4MN$ array space to store both the real and the imaginary values after the transform of the neural layer and the weight mask. In the case where the array size is not a power of 2, value padding will take up more memory space, $2^{\lceil \log_2 n \rceil} - n$ to be precise. This increase is linearly scalable according to the input size and from the time analysis in the next section the speed up is still sizable.

4.2 Execution Time Comparison

Table 1 demonstrates the theoretical expected execution time, the speed up, and calculates the empirical speed up for the convolution and DFT approach. For an array size of 16384 (= 128 × 128) the speed up calculated empirically is 1735 times faster than the convolution method.

Figure 1 plots the result of Table 2 and Table 3 on a $\log_{10} - \log_2$ grid. Graphically, it also illustrates the crossover point for array size greater or equal to 8. Furthermore, it shows a fairly linear function when scaling the neural network to larger sizes using the $\log_{10} - \log_2$ grid.

The discrepancies among the theoretical and the empirical result are found in the constant overhead charged in the DFT algorithm. Hence, for a small size array the time for this overhead is more evident and noticeable than in larger arrays. From the table, the speed up we found from the experiment was consistently better for array size greater and equal to 64 (an 8 × 8 array) than the theoretical expected speed up.

4.2.1 NSL Test

We also implemented the 2-D DFT algorithm into the Neural Simulation Language (NSL) which is a language aimed to create an environment for biological network level modeling [8]. The 2-D DFT algorithm was written in addition to the direct method so the user may choose the explicit method he desires. Table 2 demonstrates the possible speed up for the DFT algorithm without the wrap-around option. The rows represent the size for a square data layer and the columns represent the size for a square mask. The results indicate that the larger the data array, the better the speed up will be for a relatively small mask size. For instance, if the data size is 256 × 256 and the mask is 32 × 32 (1/64 of the total data size) the speed up is already 2.06 times. The maximum speed up is achieved when the sizes of the data array and the mask array are equal and greater than 16. For smaller sizes, the

[3]The program was adopted with minimum amount of modifications on personal computers. The result of the execution time is scalablely similar to the result obtained from Sun workstations.

direct convolution method will be more efficient.

4.3 Error Calculations

Inevitably with most types of transforms, there will be errors generated during the mapping process. Assuming there are no errors in the input data, the algorithm still suffers from round-off errors and truncation errors.

One other possible error in the DFT of neural network layers is the *wraparound error*. This is not a calculation error but an interference error introduced with the assumption that the values in the arrays are periodic in nature with some period P extending outside of the layer. In reality, the input signal often goes forever without repetition or else contains a non-repeating of finite length. Although this assumption sometimes is valid in neural network simulations, this constraint is not universally applicable in general.

Typically, in biological simulation, a layer is a finite piece of a cortical region; hence, no wraparound is necessary. To avoid this error, Brigham [2] showed when $P \geq M + N - 1$ the individual periods of the convolution will not overlap hence will not produce interference from the wraparound. The buffer zone of zero-padded values at the end of the data array will make the interference to be zero. Since the new period is greater than the sum of either array, the length of the extended array sequence must be increased to accommodate this change.

In neural network simulation, the size of the connection mask (response function) is almost always smaller than the size of the neural layer (data set). Therefore, the zero padding routine is a must procedure to set the mask array size equal to the layer size before applying the DFT routine for convolution to avoid data corruption.

4.4 Error Defined

The total error for each trial is measured as

$$TotalError = \sum_i |x_i - x_o| \tag{3}$$

for each array entry where x_i, the desired valued, is computed via the direct convolution method, and x_o, the output, calculated from performing the Fourier transforms. The average relative cell error for each single trial is calculated as

$$AverageError = \frac{\frac{1}{N}\sum_i |x_i - x_o|}{\frac{1}{N}\sum_i |x_i|} = \frac{\sum_i |x_i - x_o|}{\sum_i |x_i|} \tag{4}$$

Depending on the type of applications, the average relative cell error is typically in the range of 1×10^{-6} which is often better than the interpolation error introduced when calculating the discretized version of the partial differential equation. For example, when using the Euler's method for approximation of the differential equation shown in Eq. 2, the error typically varies proportional to Δt, the step size. Depending on the simulation task at hand, this value usually is in the range between 1×10^{-4} to 1×10^{-2} which is much larger when compared to the error generated by the trigonometry function.

Lastly, in neural simulation, the DFT convolution is taken at every simulation step which means that the error is also accumulative temporally as well as in each discrete time step. This relative error in successive application of the DFT transformation for convolution is additive. Therefore, a simulation with many iterations will have larger errors when compared with an equal size simulation with fewer steps. Nevertheless, one basic characteristic in neural networks is its robustness to small fluctuations and noises from the environment. This resilience to minute errors should accommodate transformational errors.

5 Discussions

Often, the mask size in a typical neural network simulation is smaller than the neural layer being processed. Perhaps it seems to be a wasteful overhead to assemble an array of size equaling the neural layer with padded zeros, tests, however, have revealed that the time spent in this overhead is still far shorter than the direct convolution method with smaller array size. Furthermore, extra zero padding procedure due to the wraparound error also seems to be memory wasteful and time consuming. However, the computation time taken should not be a major concern since in all cases that we have test even when the array size is doubled for the DFT algorithm compared with the direct convolution algorithm, the DFT algorithm still comes out ahead for size ≥ 16.

6 Conclusion

The result of this study demonstrates the feasibility of using the DFT method to calculate neuronal activity in simulation software for speed up in the order of $O(N/\log_2 N)$. While there are some overhead using the DFT technique involved in setting up the initial data input for processing, e.g., the zero padding, the speed up is evident already even for small array size (> 8). From this study, the advantage for using DFT in calculating neural activities in neural networks is so overwhelming that drawbacks on extra space consumption, transformation errors, and pre-processing procedures are diminished in the face of the speed gained. Hence, if the execution speed in the neural simulation is the prime directive then the DFT method offers an efficient approach that often outweighs drawbacks from other requirements with the following rules:

- Rule 1 – Empirically, if the area size of one of the array is more than 1/64 of the other then it is advisable to use the DFT method when minor errors are unimportant in the simulation.

- Rule 2 – To optimize the calculation in the transform, the DFT algorithm requires the sum of the size from the layer and the mask minus 1 should be a number in the power of 2. Under this condition, the algorithm ensures that the wrap around error will be nullified.

With the wrap-around option activated, the optimal criteria is changed. In this case, the maximum size of either the data layer or the mask layer must be in the power of two for the algorithm to take advantage of the situation. A user should know when to use the wrap-around option for the DFT method to be effective. Cases where symmetrical masks are present or when the actual data area is less than the array size are two scenarios that the wrap around option could be used without much ill effects.

References

[1] Allfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. The design and analysis of computer algorithms. Addison-Wesley, Reading, Massachusetts, 1974.

[2] E. Oran Brigham. The fast fourier transform. In Alan V. Oppenheim, editor, *Signal Processing.* Prentice Hall, Englewood Cliffs, NJ, 1974.

[3] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[4] Germund Dahlquist and Ake Bjorck. Numerical methods. In George Forsythe, editor, *Automatic Computation.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

[5] Michael V. Mascagni. Numerical methods for neuronal modeling. In Christof Koch and Idan Segev, editors, *Methods in neuronal modeling : from synapses to networks*, pages 439–484. MIT Press, Cambridge, Mass., 1989.

[6] Mark E. Nelson, Wojtek Furmanski, and James M. Bower. Simulating neurons and networks on parallel computers. In Christof Koch and Idan Segev, editors, *Methods in neuronal modeling : from synapses to networks*, pages 397–437. MIT Press, Cambridge, Mass., 1989.

[7] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, NY, 1988.

[8] Alfredo Weitzenfeld. Nsl - neural simulation language version 2.1. Technical report, University of Southern California, 1991.

Array Size	N^2	$N \log_2 N$	Theoretical	Empirical
2	4	2	2.00	0.30
4	16	8	2.00	0.50
8	64	24	2.67	1.14
16	256	64	4.00	2.70
32	1024	160	6.40	5.92
64	4096	384	10.67	12.14
128	16384	896	18.29	23.91
256	65536	2048	32.00	41.00
512	262144	4608	56.89	76.21
1024	1048576	10240	102.40	163.75
2048	4194304	22528	186.18	289.01
4096	16777216	49152	341.33	571.05
8192	67108864	106496	630.15	948.28
16384	268435456	229376	1170.29	1735.29

Table 1: Theoretical and Empirical Result compared.

	2	4	8	16	32	64	128	256
2	0.50							
4	0.30	0.18						
8	0.11	0.05	0.20					
16	0.03	0.07	0.19	0.68				
32	0.03	0.07	0.17	0.61	2.49			
64	0.03	0.06	0.16	0.60	2.29	9.23		
128	0.02	0.05	0.15	0.56	2.18	8.69	36.12	
256	0.02	0.05	0.14	0.52	2.06	8.35	6.77	26.11

Table 2: Two dimensional Speed-Up Table (without wraparound option)

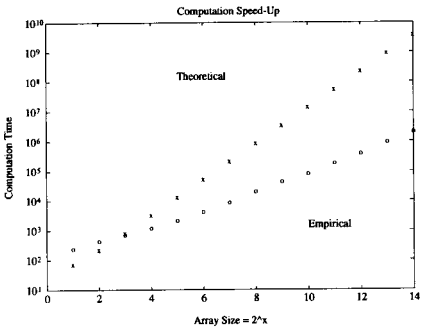

Figure 1: Computation Time Chart.

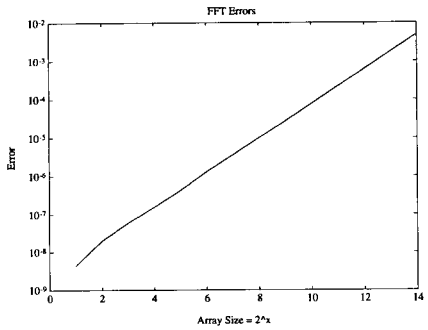

Figure 2: DFT Errors.