Computational Modeling of Neural Systems

Martin Torres

California State University Stanislaus

One University Circle

Turlock, CA 95382

mtorres27@csustan.edu

March 2015

Abstract

As computational power grows, new methods of modeling complex systems are developed that allow scientists to understand more about biological systems. This in turns leads to the development of novel algorithms and computational techniques, which creates a cycle of discovery and innovation.

In this paper we will focus on artificial neural networks, which are biologically inspired statistical learning algorithms. Specifically, we will discuss the architecture of these networks, as well as stochastic gradient descent and backpropagation, which are two core algorithms in neural networks.

We will also discuss some computational models that are being used to advance our understanding of the brain. The tools we will discuss are NEURON and EEGLAB. We will discuss how NEURON, a simulation engine, is used to model neurons and networks of neurons. We will then discuss how EEGLAB analyzes electroencephalographic (EEG) recordings.

1 Introduction

When asked about the relationship between the brain and computers, often the brain-computer analogy is the first thing to come to mind. However, this analogy falls short in many ways, and is often highly criticized [1]. Even though the brain and the computer do not work in the same way, scientists continue to model algorithms after biological processes, and try to model the brain through the use of computer software. Because the brain and computer are better at different tasks, we can use one to learn about the other.

Motivating Questions. By modeling the way that neurons are connected in the brain, artificial neural networks have been used to solve problems that involve machine learning and pattern recognition. These networks have made advances that would otherwise be difficult with logic programming. If our algorithms get more complex with the understanding of the human brain, will we eventually be able to teach a computer to do difficult tasks such as creative writing?

Will we ever be able to completely model a human brain? Scientists have developed software that is useful for modeling specific biological processes, but much progress still needs to be made for software that realistically models the entire brain. Is a complete understanding of the human brain necessary in order to create a full-scale model? Just because we cannot fully model the brain, does not mean that simulations have not contributed to understanding how the brain works. Simulations allow researchers to gain a better understanding of specific processes in

the brain, such as what happens when an ion channel releases stored calcium in a neuron [2].

Outline. Neural Networks are discussed in Section 2. This section is divided into four parts, which cover the perception and sigmoid neurons (2.1), neural network architecture (2.2), stochastic gradient descent (2.3), and backpropagation (2.4). Section 3 will cover computational models and tools that are used to research the brain. The models and tools that will be discussed are NEURON (3.1), and EEGLAB (3.2). Finally Section 4 will discuss insights and conclude the paper.

2 Artificial Neural Networks

Threshold logic, created by Warren McCulloch and Walter Pitts in 1943, was one of the first approaches to computationally model neural networks [3]. This opened up research for decades to come, but then slowed due to the lack of processing power. Artificial neural networks (ANNs) have gained recent popularity, in part due to increased computational power and new techniques in deep learning.

2.1 Perceptrons and Sigmoids

In order to understand ANNs, we need to briefly discuss the fundamental components that make up the networks. Today, most neural networks are implemented with the sigmoid neuron, but in order to fully appreciate and understand the sigmoid neuron, we must first step back to the perceptron, which was created by Franklin Rosenblatt [4]. The perceptron is based on a simple idea, a

unit takes several binary inputs, and then produces a single binary output. In order to compute the output, weights are used to model the importance for each of the inputs. The output, 0 or 1, is evaluated based on whether or not the weighted sums are less than or greater than a threshold value.

Perceptron Notation 1

$$output = \begin{cases} 0 & if \sum_j w_j x_j \leq threshold \\ 1 & if \sum_j w_j x_j > threshold \end{cases}$$

We can further simplify this equation by writing the summation of the weights as a dot product, where *w* and *x* are vectors that contain weights and inputs. We can also move the threshold to the other side of the equation, which is known as the perceptron's bias *b* = -threshold [5].

Perceptron Notation 2

$$output = \begin{cases} 0 & if\ w \cdot x + b \leq 0 \\ 1 & if\ w \cdot x + b > 0 \end{cases}$$

Based on this equation, the higher the bias is, the easier it is for the perceptron to output a 1. This way, it is easier to see how the bias is changing the output of the perceptron. Perceptrons however, have a troubling problem. Suppose you want to change the behavior of the perceptron to output a slight difference by

changing the weight. Because the perceptron is binary, a small change in the weight can cause the output to completely change. This in turn, will change the entire network of neurons, making it difficult to make small adjustments. The sigmoid neutron helps to overcome this problem.

Sigmoid neurons, like perceptrons, receive inputs and produce a single output. However, instead of binary values, inputs can be anywhere between 0 and 1. Sigmoid neurons also have weights for each input. The sigmoid neuron's output is defined as follows:

Sigmoid Output 1

$$\sigma(w \cdot x + b)$$

Where σ is called the sigmoid function [6]:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Which leads to the full equation:

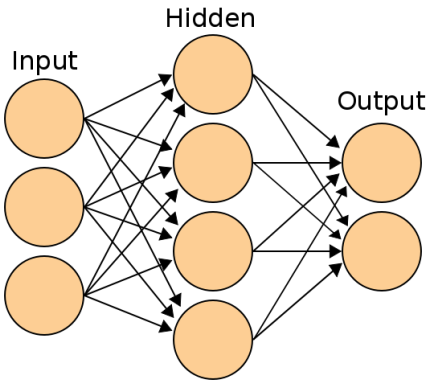
$$output = \frac{1}{1 + e^{(w \cdot x + b)}}$$

Using this equation, we can see that σ also produces a number between 0 and 1. This is useful because now, a small change in the weight or biases will cause a small change in the output. Using sigmoid neurons, a neural network could adjust its weights and bias to achieve a desired small change in the output of a neuron, which is much more useful in learning networks.

2.2 Architecture

The three layers in an ANN are the input layer, hidden layer, and output layer [6]. Input layers are defined as neurons that are feeding information into the system, and output layers are neurons or a neuron that gives final output from the network. Hidden layers are the neurons in between the input and output layers. The amount of hidden layers varies throughout each network, and researchers are always trying to find the most efficient ways to design neural networks for each learning task.

Example Artificial Neural Network



[11]

Feedforward networks are ANNs that do not have loops within the network structure, they move straight through the network. There are also recurrent neural networks, which contain loops and allow neurons to fire for a certain amount of time.

The architecture of ANNs are relatively straightforward, but extremely powerful and versatile. The study of neural networks with a large amount of hidden layers has become increasingly popular with the increase of computational power, which adds even more flexibility to neural networks [8].

2.3 Stochastic Gradient Descent

We know that changing the weights and biases of a sigmoid neuron will change the output as well, but how can we use this in order to make a whole network learn? ANNs use training sets, where they try to adjust the weights and biases to match the desired output of the network. In order to automatically adjust the weights and biases to produce the desired output, we use a cost function (sometimes referred to as the objective or loss function). The objective is to find weights and biases that minimize the cost function [9]. The reason we use the cost function to assess the performance of the ANN instead of the actual classification accuracy is because it is much easier to measure how small changes affect the cost function rather than the overall classification accuracy.

Neural networks often use gradient descent to minimize the cost function [10]. In order to explain gradient descent, we will think of trying to minimize a cost function $C(v_1, v_2)$ where v_1 and v_2 are two parameters. If we wanted to find out how a change in v_1 (Δv_1) or a change in v_2 (Δv_2) affects C, we could use the following formula:

Gradient Vector

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

Where ∂ is the partial derivative. We can define the gradient of C as the vector of partial derivatives:

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$$

Where T is the transpose operation. Defining Δv as $(\Delta v_1, \Delta v_2)^T$ gives us

$$\Delta C \approx \nabla C \cdot \Delta v$$

Now remember, the purpose is to find values that would cause C to minimize. So we would like to choose Δv in order to make ΔC negative. If we use

$$\Delta v = -\eta \nabla C,$$

where η is a small, positive parameter, then…

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$$

which means that ΔC will be minimized as long as we use the above formula. If we repeat this using the following update formula, we will be constantly moving towards a minimum. [5]

$$v \rightarrow v' = v - \eta \nabla C$$

The above was an example with only two variables, but in reality, there are many more variables in a neural network. Computing the partial derivatives for these variables would quickly become computationally expensive. Stochastic gradient descent speeds up the process of finding the gradient by using mini-batches (small samples of training data). By averaging the small sample sizes, we can get a relatively good estimate of what the gradient will be [12].

2.4 Backpropagation

Previously, we discussed stochastic gradient descent, and how we can use the gradient to update weights and biases in our network to minimize the cost function. However, we did not actually compute the gradient. Backpropagation aims to calculate the gradient, which means finding the partial derivatives of the weights and biases. Not only does backpropagation compute the gradient, but it also takes into account error δ^L (where L signifies the layer) in the network. There are four main equations that compose the backpropagation algorithm [13]. We will briefly introduce each of them, and the show how they work together to create the backpropagation algorithm.

Four equations

Error in the output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right)$$

Where $a_j^L = \sigma(w^l a^{l-1} + b^l)$. Weight matrix is defined as w^l and bias vector by b^l. This equation keeps track of two relations, the rate at which the cost is changing at the j^{th} output, and the rate that the activation function σ (the likelihood that a neuron will fire) is changing at z_j^L(the weighted input at the layer L). We can also write this in a matrix-based form,

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

where $\odot$ is the Hadamard product.

Error in terms of the error in the next layer:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L)$$

The only new term in this equation is $(w^{l+1})^T$, which is the transpose weight matrix of the next layer. Combining the first two equations allows us to measure the error in any layer of the network.

Rate of change of cost in terms of any bias in the network:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

This equation shows that the rate of change of bias is the error, which can be easily found in any layer using the equations above.

Rate of change of cost in terms of any weight in the network:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

We now know how to compute both the error and the activation in order to obtain the weight in the network. [5]

Now that we have the four equations that are necessary for the algorithm, we will outline the proper steps of backpropagation.

Backpropagation algorithm

---

1. Input *x*: First we initialize the input layer and set up the corresponding activation a^1

2. Propagate activity forward: For each layer *l*, we compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$
3. Calculate the error in the output layer: Compute $\delta^L = \nabla_a C \odot \sigma'^{(z^L)}$
4. Backpropagate the error: For each layer, compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
5. Output: The gradient of the cost function $\frac{\partial C}{\partial b_j^l} = \delta_j^l$ and $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Backpropagation has allowed the much faster computation times of computing gradients, which accelerated the research and usefulness of neural networks [14]. Although the algorithm is difficult to grasp, it is important to take time to understand this core algorithm that is used in ANNs.

3 Computational Models of the Brain

The brain, like computer science, has many different areas of study. Software packages have helped researchers focus on specific areas of the brain, and allow researchers to simulate processes in the brain that would be otherwise difficult to study in real time. We will briefly introduce some of these software packages, and give examples of how the technology can be put to use.

3.1 NEURON

NEURON is a free, open source software package that aims to simulate neurons and networks of neurons. It was developed at Yale and Duke by Michael

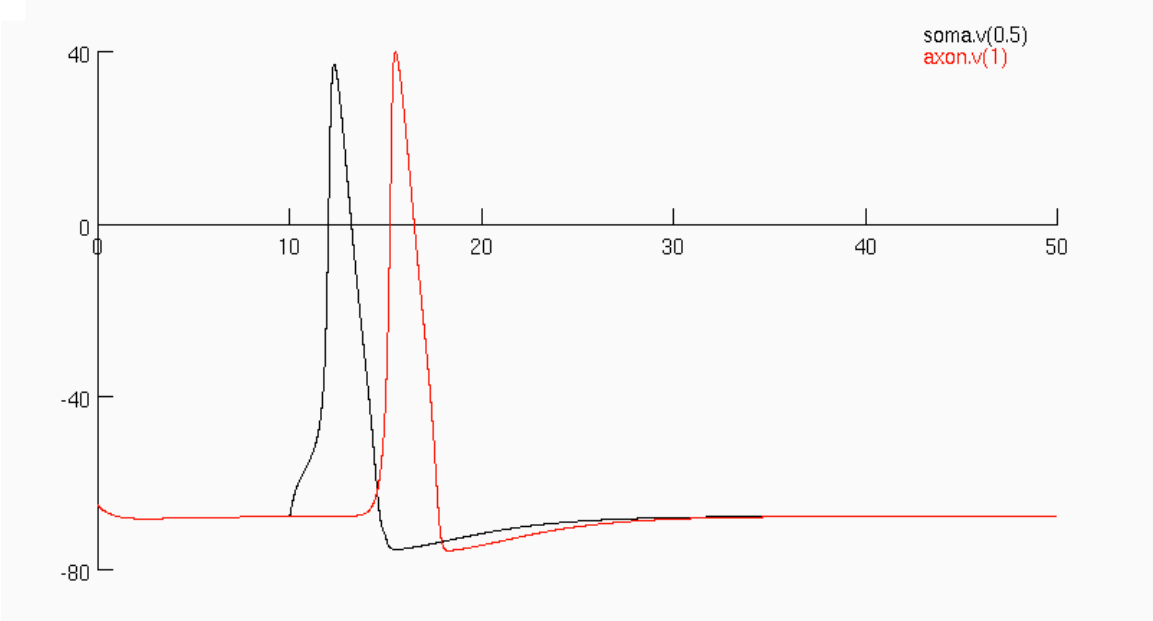
Hines, John W. Moore, and Ted Carnevale. NEURON has been used with more than 1450 scientific articles and journals, and has a well-established community [15]. NEURON also allows Python to be run as the interpreter, which allows for the use of scientific packages such as Numpy, Scipy, and Matplotlib [16].

Creating a simulation and plotting the results is fairly straightforward. We will show some pseudo code of how to set up a neuron simulation.

Creating a Neuron

```
create  soma, axon
soma {
        L = 100         //Length
        diam= 100       //Diameter
        insert hh       //Hodgkin-Huxley channels
        insert pas      //Passive Membrane property
       }
axon {
        L = 5000
        diam = 10
        insert hh
        insert pas
        nseg = 10       //The axon will have 10 compartments
       }
connect axon(0), soma(1)
objref stim
soma stim = new IClamp(0.5)        //Insert stimulus clamp in neuron
stim.del = 10                      //delay
stim.dur = 5                       //duration
stim.amp = 10                      //amplitude
run()
```

When we plot this to a graph, we can see how the voltage affects the soma and the axon.



[17] [18]

3.2 EEGLAB

EEGLAB runs under MATLAB, and is meant for processing EEG data that is collected during research. It has a large array of functions, which makes it easy to preprocess, visualize, and perform analysis on EEG data from any number of channels [18]. EEGs record the electrical impulses in the brain by using electrodes attached to the scalp. Because neurons are constantly firing, large amounts of data begin to accumulate quickly. Factors such as the amount of electrical impulses, the timing, and the patterns of neurons firing all need to be recorded. EEGLAB helps to analyze this type of data.

Example: Twenty seconds of EEG recordings on multiple channels.

[19]

EEGLAB heavily focuses on data processing and Independent Component Analysis (ICA). ICA is meant to remove noise of other electrical signals, such as from the eyes and other muscles [20]. This is a great tool in helping speed up preprocessing, which has made EEG data much easier to work with.

4 Conclusions

One can argue that we have made much progress in the areas of computational neuroscience and advancing algorithms. However, one can also argue that computational neuroscience is still in its infancy, and that we still have much to learn and discover. We have made more progress in the past century than we have in thousands of years. Is this attributed to the brain or is it attributed to computers? After examining algorithms like backpropagation and stochastic gradient descent, it is fascinating to see how humans have developed these methods in order to gain

better understanding of our world and solve difficult problems. History has shown us that humans have done things that we once thought would be impossible, therefore I have no reason to doubt that our scientific progress will continue, and we will eventually understand the brain and develop highly advanced and intelligent algorithms.

References

[1] Carello, C., Turvey, M., Kugler, P., & Shaw, R. (1984). Inadequacies of the Computer Metaphor. In Handbook of Cognitive Neuroscience (pp. 229-248). Springer US.

[2] Weiliang Chen, Erik De Schutter. Python-based geometry preparation and simulation visualization toolkits for STEPS. Frontiers in Neuroinformatics, 2014; 8 DOI: 10.3389/fninf.2014.00037

[3] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics 5 (4): 115–133. doi:10.1007/BF02478259.

[4] Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". Psychological Review 65 (6): 386–408.doi:10.1037/h0042519. PMID 13602029.

[5] Nielson, M. (n.d.). CHAPTER 1. Retrieved March 25, 2015, from http://neuralnetworksanddeeplearning.com/chap1.html

[6] Martin Anthony (January 2001). Discrete Mathematics of Neural Networks: Selected Topics. SIAM. pp. 3–. ISBN 978-0-89871-480-7.

[7] Judith E. Dayhoff, Neural network architectures: an introduction, Van Nostrand Reinhold Co., New York, NY, 1990

[8] Ebru Arisoy , Tara N. Sainath , Brian Kingsbury , Bhuvana Ramabhadran, Deep neural network language models, Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT, p.20-28, June 08-08, 2012, Montreal, Canada

[9] Nikulin, M.S. (2001), "Loss function", in Hazewinkel, Michiel, *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4

[10] Akarachai Atakulreka , Daricha Sutivong, Avoiding local minima in feedforward neural networks by simultaneous learning, Proceedings of the 20th Australian joint conference on Advances in artificial intelligence, December 02-06, 2007, Gold Coast, Australia

[11] "Artificial neural network" by en:User:Cburnett - Own workThis vector image was created with Inkscape.. Licensed under CC BY-SA 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg#/media/File:Artificial_neural_network.svg

[12] Rainer Gemulla , Erik Nijkamp , Peter J. Haas , Yannis Sismanis, Large-scale matrix factorization with distributed stochastic gradient descent, Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, August 21-24, 2011, San Diego, California, USA

[doi>10.1145/2020408.2020426]

[13] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". *Nature* **323** (6088): 533–536.doi:10.1038/323533a0.

[14] Werbos, P.J. (1975). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*.

[15] (n.d.). Retrieved March 26, 2015, from http://www.neuron.yale.edu/neuron/what_is_neuron

[16] Hines, M. L., Davison, A. P., and Muller, E. (2009). NEURON and Python. Front. Neuroinform, 2009

[17] Carnevale, N.T. and Hines, M.L. *The NEURON Book.* Cambridge, UK: Cambridge University Press, 2006.

[18] Neuron (software). (n.d.). Retrieved March 26, 2015, from http://en.wikipedia.org/wiki/Neuron_(software)

[19] BMC Infectious Diseases 2006, 6:169doi:10.1186/1471-2334-6-169 http://commons.wikimedia.org/wiki/File:Eeg_CJD.jpg

[20] Delorme, A., & Makeig, S. (n.d.). EEGLAB: An open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods,* 9-21.