A dendritic transfer function in a novel fully-connected layer

Undergraduate Honor's Thesis

1st Mark Musil
Electrical and Computer Engineering Department
Portland State University
mmusil@pdx.edu

***Abstract*—**

***Index Terms*—Dendrite, transfer function, dendritic, neural network, layer**

I. Introduction

Modern artificial neural networks are only loosely based on the structure of the human brain[] but have shown great success in image recognition, functional estimation, and other machine learning tasks. The model that is widely used - although greatly customized - is the feed-forward network with the familiar equation $y = \sigma(\vec{x}W + \vec{b})$. This model uses a very simple model of the neuron as can be seen in figure 1. (Many key neural structures such as...)

Despite this success, machine learning researchers are hesitant to return to the human brain for more insights into artificial intelligence algorithms[]. This hesitation from the scientific community leaves a large space for investigation into which operations performed by the human brain may be useful for machine learning.

The neuron contains many signal processing abilities that have not been fully explored. This work asks the following question to further the understanding of the processes within the neuron: can a non-linear dendritic branch function[] be combined with a two-layer neuron model[] to create a computational unit that, when used to create a fully connected layer, improves performance of a CNN (Convolutional Neural Network) using this layer on the MNIST data-set?

II. Background

It has long been known that stimuli input to the spine-like synapses along the dendritic branch are not conducted directly to the soma in all cases [1]. Several factors impact how much a voltage spike at a synapse increases the soma's potential [2] but for the sake of neural networks many of these are not useful. Stimuli at distal synapses are subject to two sources of attenuation: spatial and temporal. Spatial attenuation is caused by leakage current along the dendritic branch and means that soma potential change is proportional to distance from the soma. Temporal attenuation can be thought of as a saturation effect; when many stimuli occur at distant locations along a branch but at roughly the same time, the soma to cell-exterior voltage difference will not be the sum of the voltage differences at the synapses. The same 'muting' effect takes place when many stimuli take place at relatively close locations but not necessarily at the same time. The aforementioned effects do not take into account inter-branch interactions or how different charge carriers impact signal transfer.

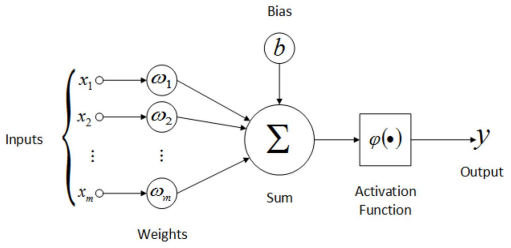

Figure 1. The classic artificial neuron, which assumes that all synapses are connected to the same lossless dendritic branch and are only acted upon after reaching the soma. Source scielo.br

The function $T(Y)$ mimics the signal transmission behavior of a dendrite (figure **??**) and acts as an 'activation function' for the inputs on the

dendritic branch.

This work will focus on using an existing model of the neuron that incorporates the dendritic transfer function [3], and then using that model to create a layer for an artificial neural network in order to test the performance of the dendritic neuron model.

III. Literature Review

The behavior of dendrites has long been studied and recognized as computationally significant. In [4] the role of the dendrite in memory is discussed and in [1] it is acknowledged that dendrites work as separate thresholding units pre-filtering the input to the soma. [2] further establishes the importance and nature of dendritic computation. [2] even presents certain organisms who possess certain single neuron systems capable of (thanks to dendritic computation) solving much higher order problems. For example, the locust has a single neuron in its visual system that, using dendrites only dendritic computations, searches for potential items on a collision course with the locust's head [2]. Such single-neuron tasks show that much processing takes place in the dendrites.

These examples establish the clear potential for dendrites in computational neuroscience. However, it is no small task to translate an observed natural phenomenon into something that can be used in artificial applications. The authors of [3] took the first step by establishing an artificial transfer function (equations 2 and 3) which simplifies the process of creating a useful computational unit in software. [3] also proposes a two-layer neuron model which uses the dendritic transfer function in the first layer. This is based on the model proposed in [5] but advances it by adding the dendritic transfer function to the first layer. Despite this respectable body of knowledge, there is no apparent evidence of an experiment which attempts to place the two-layer dendrite-based neuron model into a neural network. It is crucial that this space is explored in order to test the hypothesis that dendrites have significance for computational problems beyond those observed in the human brain.

IV. Methods

My work is largely inspired by previous work described in [3] where the dendritic transfer function used in my thesis is derived. The graphical representation for the dendritically modeled neuron that was described in [3] can be seen in figure 3. The addition of the dendritic transfer function $T(\vec{Y})$ is the most important piece to observe in that diagram. This function mimics the electrical qualities of a dendritic branch by handling saturation conditions (too much stimuli on the branch) before they reach the final summation and activation (analogous to the soma).

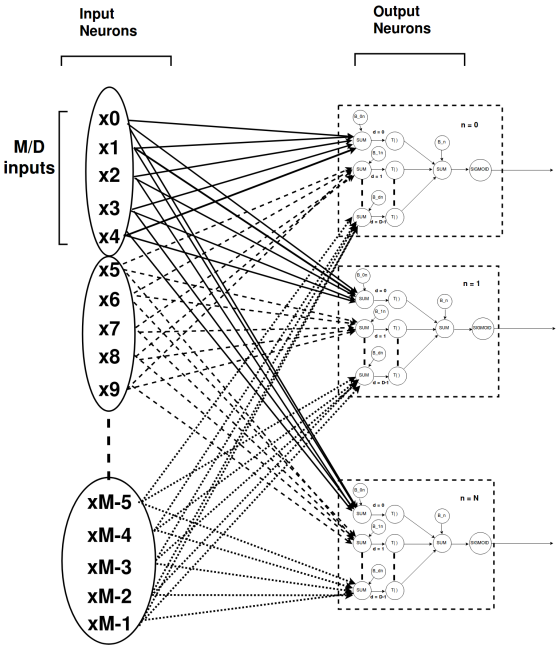

Figure 2. A limited display of the proposed fully connected layer. Note that the same group of M/D inputs connect to the same d^{th} dendrite of each neuron.

In this thesis, This neuron model will be used to create a fully connected layer that can be incorporated into existing neural networks. For these experiments the neural network into which the new layer will be placed is the CNN (convolutional neural network). This network has been chosen because vision in neuroscience is the main focus of the group conducting the research. There is no special reason besides that and the layer is designed to work in any feed-forward network trained using backpropagation.

Figure 2 shows the proposed structure for the novel layer and figure 3 shows a closer view of the dendritic neuron. It is important to notice

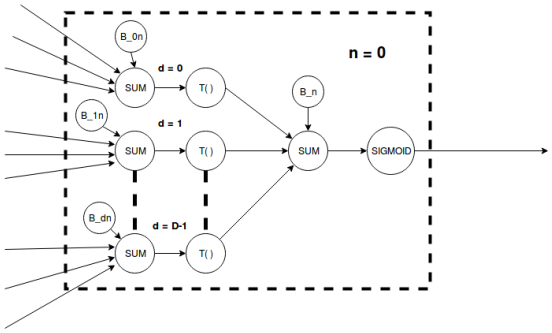

Figure 3. The schematic for the neuron model that includes the dendritic activation mechanism. $T(\vec{X})$ is the novel function that is being implemented. Two sets of summation and a final sigmoid output are also parts of the model.

that each dendritic branch (what the arrows are pointing to) does not see all of the input space but only the D^{th} fraction of the input space where D is the number of dendrites per neuron. This is because each dendrite should only gather information about a small portion of the input space. Once each dendrite has seen its window of the input space the soma of the neuron 'decides' whether the information that was presented to the soma (by the dendrites) matches the patterns that that neuron has been trained to recognize.

The dendritic neurons will be assembled in layers of N neurons where each neuron will be fully connected to the input space but no single dendrite will see the entire input space.

The dendritic transfer function is actually split into two functions, the activation function $T(\vec{Y})$ and the boundary function $g(y)$ [3].. Although when the dendritic transfer function is discussed $T(\vec{Y})$ is the intended meaning. In addition, the multivariate logistic-sigmoid σ_{multi} is also defined in equation 1 and is used in $T(\vec{Y})$.

$$\sigma_{multi}(\vec{X}) : R^n \rightarrow R = [1 + exp(-\Sigma X_i)]^{-1} \tag{1}$$

$$g(y) = ln(\frac{(1+exp(\alpha_L(y-b_L)))^{\alpha_L^{-1}}}{(1+exp(\alpha_U(y-b_U)))^{\alpha_U^{-1}}}) + b_L \tag{2}$$

$$T(\vec{Y_{dn}}) = g(c_d \sigma_{multi}(a_d[\vec{Y_{dn}} - b_d]) + \Sigma Y_{dn,i}) \tag{3}$$

	α_L	α_U	b_U	b_L
Value	0.5	0.5	0	1
	a_d	c_d	b_d	
Value	1	Trained	”	

Figure 4. Table of values for $T(\vec{Y})$ and $g(y)$.

Note that b_d is a single real number and not a vector. Table 4 enumerates all of the miscellaneous variables in these equations.

Note that $\vec{Y}$ and y are used as notation for the inputs to $g(y)$ and $T(\vec{Y})$ as opposed to the conventional x because the input to the dendritic transfer function is the output of the classic layer weight function $\vec{Y_{dn}} = \vec{X}W_d + B_{dn}$. $\vec{X}$ is the input activation of the previous layer and B_{dn} is the bias term.

The algorithm for the layer implementation is described in algorithm 1. The algorithm assumes that **N** neurons and **D** dendrites per neuron have been chosen for the layer in which the computation is taking place. The subscripts d and n denote individual dendrites (d) or neurons (n). B_{dn} (two subscripts) is the bias for a single dendrite while B_n (one subscript) is the bias for the soma of the neuron. ϕ is the activation function and O_n is the output of the n[th] neuron in the output layer.

Algorithm 1: The algorithm that defines the functionality of the novel layer.

Data: Set of input activations from the previous layer $\vec{X}$
Result: Set of output activations of the current layer $\vec{O}$

for $n \leftarrow 0$ **to** *N-1* **do**
  for $d \leftarrow 0$ **to** *D-1* **do**
    Compute $\vec{Y_{dn}} = \vec{X}W_d + B_{dn}$;
    Compute $T(\vec{Y_{dn}})$;
  end
  Compute $O_n = \phi([\sum_{j=0}^{j=D-1} T(\vec{Y_{jn}})] + B_n)$;
end

The algorithm will be implemented as a network layer following the structure displayed in figure 2. This will allow the robust and approachable Keras interface to be leveraged during the

project. The layer will be able to seamlessly integrate with other Keras layers and this approach allows my layer to be published and used by other Keras users easily. The completed source code is on GitHub [6] where it is accessible by anyone interested in using it.

V. Results

After many attempts, it was found that algorithm 1 could not be implemented using a standard deep learning APi (PyTorch) because it was not defined in a backward-differentiable manner. That is to say that any errors created by algorithm 1 and passed forward to the output layer can never be traced back to the individual weight or bias that caused the error. Plans for future work can be found in the discussion section.

VI. Discussion and Future Work

Although the experiment was not completed, the reason that it failed has important implications for the future of deep learning. The main issue of algorithm 1 is the presence of for loops. For loops are a perfectly mathematically valid way of expressing many types of layer operations in neural networks. However, this structure does not allow the error at the output layer to be traced back through this layer via the chain rule.

This hurdle is quite common and the movement to create frameworks that can translate functional programming into differentiable layers in a deep network is called **differentiable programming**. Differentiable programming is in turn based on the emerging field of automatic differentiation [7]. Simple methods existing to implement automatic differentiation such as that described in [8]: " The essence of Forward AD [automatic differentation] is to attach perturbations to each number, and propagate these through the computation.... This is typically accomplished by creating a unique tag for each derivative calculation, tagging the perturbations, and overloading the arithmetic operators." There have been attempts to create such system which allow for the programmer to simply create a forward defined algorithm with little else required. One such API is Diffsharp (diffsharp.github.io) which will most likely be the tool used to rework algorithm 1.

Now that the issue with the algorithm is identified work can beginning on converting it into a differentiable representation of the same process. The first step will be to ensure that the connections from the previous layer to the dendritic layer shown in figure 2 will allow for a differentiable tensor representations. Next, the algorithm should be redefined as part of a computational graph where tensor wide operations are made all at once. This is as opposed to an index-by-index approach taken in algorithm 1.

References

[1] S. Sardi, R. Vardi, A. Goldental, and I. Kanter, "New types of experiments reveal that a neuron functions as multiple independent threshold units," *Neuron*, vol. 7, no. 18036, 2017.

[2] M. London and M. Hausser, "Dendritic computation," *Annual Review of Neuroscience*, vol. 28, 2005.

[3] M. F. Singh and D. H. Zald, "A simple transfer function for non-linear dendritic integration," *Frontiers in Computational Neuroscience*, vol. 9, no. 10, 2015.

[4] P. Poirazi and B. W. Mel, "Impact of active dendrites and structural plasticity on the memory capacity of neural tissue," *Neuron*, vol. 29, no. 5451, p. 779–796, 2001.

[5] P. Poirazi, B. T., and B. W. Mel, "Pyramidal neuron as two-layer neural network," *Neuron*, vol. 37, no. 6, pp. 989–99, 2003.

[6] M. Musil. (2018) Github repository for this honor's thesis. [Online]. Available: https://github.com/mmusil25/CorGraphProject/tree/master/DendriticLayer

[7] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic differentiation in machine learning: a survey," *CoRR*, vol. abs/1502.05767, 2015. [Online]. Available: http://arxiv.org/abs/1502.05767

[8] O. Manzyuk, B. A. Pearlmutter, A. A. Radul, D. R. Rush, and J. M. Siskind, "Confusion of tagged perturbations in forward automatic differentiation of higher-order functions," *CoRR*, vol. abs/1211.4892, 2012. [Online]. Available: http://arxiv.org/abs/1211.4892