A Division Based Neuron for Neural Networks

Jaelen Dixon
Dept. of EECS, Howard University
Washington, DC, United States
jaelen.dixon@bison.howard.edu

Jiang Li
Dept. of EECS, Howard University
Washington, DC, United States
jli@howard.edu

***Abstract*—We propose an alternative neuron design for artificial neural networks that replaces wx+b in a typical neuron with w1x/w2x. The design allows more complex calculations, such as division, to be performed efficiently within a neural network, and thus more efficient and specialized prediction models to be made. Along with the new design, we developed an algorithm to dynamically adjust the learning rate for model training. The algorithms were tested in the training of single-layer and single-neuron models to predict the outcome of various division-based operations. The initial test results showed that the average prediction errors are between 0.0241% and 0.898%. That is significantly more accurate than traditional neural networks with more layers.**

I. Introduction

The prevailing design of neurons in artificial neural networks typically involves a two-phase process: the neuron computes the dot product of the input and a weight vector, then an activation function is applied to this product. This configuration allows neurons to perform basic arithmetic operations like addition, subtraction, and scalar multiplication. However, it struggles to effectively model other operations like division. While large neural networks with numerous layers theoretically can represent any function within a closed range [1], modeling even simple functions could require excessive layers.

To address this limitation, a novel class of neurons is needed. These neurons would be able to handle diverse operations more efficiently, potentially reducing the overall size and enhancing the performance of neural networks. In this paper, we introduce a new neuron model that diverges from the traditional dot product approach, focusing instead on an operation that is specially tailored for division calculations.

II. Related Work

The field of neural network architecture has seen several innovative developments aimed at enhancing the capability of networks to learn and perform a diverse array of numerical operations. One notable contribution is the Neural Arithmetic Logic Unit (NALU) [2], which was designed to augment traditional neurons. Each NALU consists of multiple traditional neurons and specializes in learning numerical computations. It can perform a broad range of operations, including multiplication, division, and power functions, but it can only perform one type of operation per unit after training. In comparison, our design provides a more versatile way of combining features. It is an alternative to the traditional design, and may be used in conjunction with traditional neurons or independently.

Additionally, a type of neuron has been proposed [3] with the purpose of performing more complex operations within an individual neuron in order to simplify the structure of neural networks. By multiplying two sets of dot products, the approach creates a "second order neuron", similar to a second order quadratic function. As such, inputs are able to be multiplied by themselves and by other inputs, and the traditional neuron design becomes a special case. Our neuron design takes a similar approach, but instead of multiplying two dot products, we divide two dot products. Therefore, it can be expected that our neuron will perform better for different sets of operations and tasks such as those involving division.

III. Methodology

The new neuron in our design performs the following calculation:

$$(w_1 \cdot x + b_1)/(w_2 \cdot x + b_2) \tag{1}$$

where $w_1 \cdot x$ is the dot product between the weight vector w_1 and the input vector x, and $w_1 \cdot x$ is similar. The values b_1 and b_2 are bias parameters added to their respective dot products. It should be noted that the prevailing neuron design is a special case of our design when w_2 is the zero vector and $b_2 = 1$.

The loss function used in the model for testing is as follows, where $y^{[i]}$ is the sample value and $f(x^{[i]})$ is the predicted value.

if $y^{[i]}$ and $f(x^{[i]})$ are of different signs:
$loss = (ln(\sigma(y^{[i]}) + 1) - ln(\sigma(f(x^{[i]})) + 1))^2$
else:
$loss = (ln((y^{[i]})^2 + 10^{-32}) - ln((f(x^{[i]}))^2 + 10^{-32}))^2$

Since our neuron design involves division, it has infinitely many sets of optimal weights. This may prevent the model from converging. To circumvent this issue, we apply a combined L1/L2 regularizer to each of the weights, since using both performed slightly better than using them individually.

During training, we determined that the learning rate required for different test cases needs to begin at greatly varying values, needs to decrease as the loss value converges closer to 0, and sometimes leads to very slow convergence. In response to this, we developed an algorithm to update the learning rate dynamically during training. Specifically, the learning rate

This work was funded in part by NSF grant 1924092.

979-8-3503-5409-6/24/$31.00 ©2024 IEEE
DOI 10.1109/CAI59869.2024.00013

is decreased if the loss value fluctuates or keeps increasing while below a certain threshold, and it is increased if the loss value fluctuates above a certain threshold or is stagnant. The algorithm has a $patience$ parameter and will only update the learning rate after these conditions are met for the given number of times. The algorithm is as follows, with $wait\_0$, $wait\_1$, $reset\_patience\_0$, and $reset\_patience\_1$ initialized as 0:

if $current\_loss < best\_loss$
  best_loss = current_loss
  reset_patience_0 += 1, reset_patience_1 = 0
else if $current\_loss > best\_loss$ and $current < 0.1$
  $reset\_patience\_0 += 1, wait\_1 += 1$
  if $wait\_1 > patience$
    if $best\_loss < loss\_threshold$
      Stop training
    else
      Decrease learning rate by 50%
else
  $reset\_patience\_1 += +1, wait\_0 += 1$
  if $wait\_0 > patience$
    if $best\_loss < loss\_threshold$
      Stop training
    else
      Increase learning rate by 50%
if $reset\_patience\_0 >= patience \times 5$
  $wait\_0 = 0, reset\_patience\_0 = 0$
if $reset\_patience\_1 >= patience \times 5$
  $wait\_1 = 0, reset\_patience\_1 = 0$

This algorithm has its parameters initialized at the beginning of training and is executed once after every training epoch. The $best\_loss$ parameter is initialized to a large value such that it will be replaced by the first loss value once the algorithm runs. The $loss\_threshold$ and $patience$ parameters are adjustable for different scenarios. In our testing, these values were set to 10^{-3} and 5 respectively. Additionally, the size that the learning rate increases and decreases by is also tunable. In our testing, the learning rate is increased and decreased by 50% each time. The optimal values of these parameters can be explored in future research.

IV. Results

As an initial test of our design, we focused primarily on training a simple model consisting of one layer with one neuron. Various formulas involving divisions were used to generate training sets and test sets. The weights of the neuron are expected to converge to the values in the formula while training. Each sample of the training and test sets contained two inputs, x_1 and x_2, both in the range $[1, 2^{30}]$. Each training set has 167,000 samples, and each test set has 82,500 samples. The table below shows the loss and the mean absolute error of the model for both the training set and the test set. The percent error values show how much the predicted values deviate from the true values on average for the test set. Due to space constraints, the results for a number of other formulas were omitted and can be provided upon request.

TABLE I

Expected Formula	Training Loss	Training MAE	Test Loss	Test MAE	Percent Error
x_1/x_2	8.800E-4	0.8287	7.410E-4	0.3036	0.049%
$3x_1/\text{-}x_2$	5.804E-4	2.6569	7.120E-4	1.6912	0.053%
$x_1/3x_2$	6.229E-4	0.0251	6.971E-4	0.8422	0.0241%
$(3x_1+2x_2)/$ $(\text{-}2x_1\text{-}3x_2)$	9.375E-4	0.0089	9.424E-4	0.0092	0.898%

It should be noted that for each test, the training was set to stop once the loss value went below 10^{-3}. Additionally, in our tests, the training stopped after no more than 100 epochs. It is possible that better results could be obtained if training had continued.

For comparison, we also attempted to model the formula x_1/x_2 using a substantially larger neural network consisting only of the traditional neurons. The model consists of 5 layers, each with 16, 8, 4, 2, and 1 neurons, respectively, and was run for a similar number of epochs. The results of this test are shown below.

TABLE II

Expected Formula	Training Loss	Training MAE	Test Loss	Test MAE	Percent Error
x_1/x_2	6636.015	6.3564	17978.752	7.0408	3043.458%

It can be seen that a single one of our neurons performs substantially better than a moderately sized network of traditional neurons for the simple division operation.

V. Conclusion and Future Work

In this paper, we did an initial exploration of a novel neuron design that replaces the dot product in the prevailing neuron design with a more versatile operation involving division, with the intent of creating neurons that are capable of handling more diverse types of operations efficiently. Along with the neuron design, we developed a loss function for the models based on the neuron, as well as an algorithm to dynamically adjust the learning rate during training. Initial experiments show that the new neuron is capable of handling division-based calculations far more efficiently than the traditional ones.

While the initial results are promising, we understand that the exploration is very limited — much more study is necessary. For example, other loss functions can be explored, the neuron design can be expanded to involve more inputs, and the parameters need more careful analysis. The additional computational cost of performing division must also be explored. Ultimately, the neurons need to be tested in much larger models for real applications.

References

[1] K. Hornik, "Approximation capabilities of multilayer feedforward networks", Neural Networks, vol. 4, pp. 251-257, 1991.

[2] A. Task, et. al, "Neural Arithmetic Logic Units", Advances in Neural Information Processing Systems, August 2018, 31.

[3] F. Fan, W. Cong and Ge. Wang, "A new type of neurons for machine learning", Numerical Methods in Biomedical Engineering, vol. 34, issue 2, 27 July 2017.