BACKPROPAGATION’S VANISHING GRADIENT PROBLEM: SIGMOID VS RELU ACTIVATION FUNCTIONS WITHIN THE MULTILAYER PERCEPTRON

Shane J. Robinson[1, 2, †]

[1] Northwestern University
633 Clark Street, Evanston, IL 60208

[2] github.com/shanergit

[†] Address to which correspondence should be addressed:
shaner.mail@icloud.com

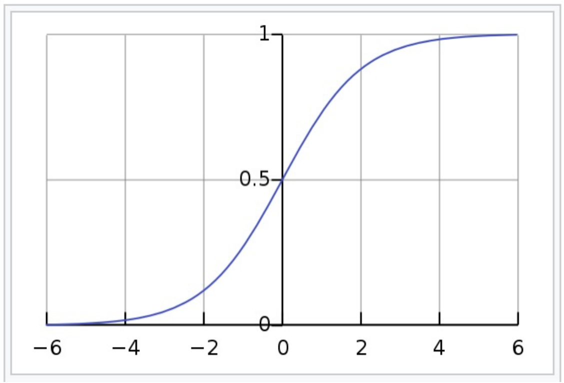
Abstract

While backpropagation has been a core algorithm for neural network training, one of the primary challenges when using backpropagation within a deep neural network (DNN) is overcoming the vanishing gradient problem. It arises when the gradients used to update the network's weights become extremely small as they propagate backward through the layers. These vanishing gradients can lead to slowed or stagnant learning, particularly within networks consisting of many layers, which makes it difficult for the network to realize complex relationships within the data. Selecting the optimal activation function for use within a backpropagating neural network can alleviate this vanishing gradient problem. Here, we will compare the learning ability of the sigmoid and Rectified Linear Unit (ReLU) activation functions using the multilayer perceptron (MLP) on MNIST data and illustrate how the ReLU outperforms the sigmoid by mitigating vanishing gradients.

Introduction

$$\frac{\partial SSE}{\partial v_{h,o}} = \frac{\partial SSE}{\partial E_o} * \frac{\partial E_o}{\partial F} * \frac{\partial F}{\partial (NdInpt_o)} * \frac{\partial (NdInpt_o)}{\partial v_{h,o}}$$
$$= E_o * (-1) * \alpha F_o[1 - F_o] * H_h$$
$$= -E_o * \alpha F_o[1 - F_o] * H_h$$

Backpropagation utilizes partial derivatives to compute the gradients of an objective function with respect to the weights in a neural network. It employs the chain rule of calculus to calculate these gradients layer by layer. What's crucial to understand is that the gradient of the objective function with respect to a module's input can be determined by tracing back from the gradient with respect to the module's output – or the input of the following module. The vanishing gradient problem caused by certain activation functions refers to a situation where the gradients used for updating the weights in a neural network become extremely small as they propagate backward through the layers. Sepp Hochreiter analyzed the vanishing gradient problem within learning in recurrent neural networks, noting the involved temporal dependencies spanning many time steps and how gradient-based learning in this method can take too much time, proposing problem solutions (Hochreiter 1998). This issue commonly occurs with activation functions that squash their inputs, such as the sigmoid (logistic) function – these functions are sigmoidal in their characterized by its "S"-shaped, or sigmoidal, curve:

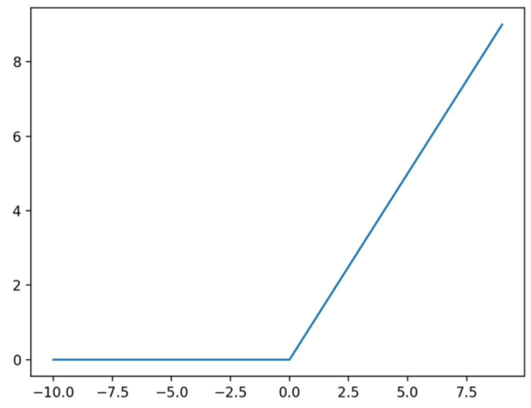


$$f(x) = \frac{1}{1 + e^{-x}}$$

ReLU is an abbreviation for rectified linear unit. Despite its name implying linearity, the ReLU possesses a derivative function, enabling backpropagation and computational efficiency. The key aspect of ReLU is that it selectively activates neurons, deactivating them only when the output of the linear transformation is negative. It returns the input directly if it is positive, otherwise, it returns zero. The graph of

ReLU is a simple piecewise function with a straight line for positive values of x and zero for negative values:

$$f(x) = \max(0, x)$$



In their paper *Deep Learning*, Yann LeCun and team note that the non-linear functions used in neural networks include ReLUs, most used in recent years, as well as sigmoids like the hyperbolic tangent (Tanh) and logistic function. They go on to illustrate the use of ReLUs inside a convolutional neural network (CNN) and how this activation function typically learns much faster within networks of many layers, due to not squashing inputs to extremely small values – vanishing gradients, which allows training of deep supervised networks without unsupervised pre-training. In the 1990's, the sigmoid activation function was used by default, followed by the Tanh up to the 2010's; for modern neural networks, however, such as the MLP and CNN, the default recommendation is the ReLU (LeCun et al 2015). The sigmoid is still commonly used within recurrent networks. We will demonstrate that the ReLU activation function indeed mitigates the vanishing gradient problem and outperforms a sigmoid function in learning ability on the MNIST handwriting dataset within the multilayer perceptron.

Related Work

A 2010 paper in *Neural Computation* detailed how large and plain MLPs, pre-ReLU, are difficult to train on MNIST as backpropagated gradients quickly vanish exponentially in the large number of layers; the authors go on to propose that the learning problems suffered by sigmoidal MLPs can be overcome by training on GPUs rather than CPUs (Cireşan et al 2010). Also in 2010, Vinod Nair and Geoffrey E. Hinton detailed how the restricted Boltzmann machine (RBM) was developed using binary stochastic hidden units and used the NORB and Labeled Faces in the Wild datasets to demonstrate how ReLUs improve RBMs by approximating the learning and influence rules within them. They posit that, compared with binary units, the ReLUs learn features that are better for object recognition on the NORB dataset and better for face verification on the Labeled Faces in the Wild dataset (Nair and Hinton 2010).

In their paper *Deep Sparse Rectifier Neural Networks*, Xavier Glorot, Antione Bordes, and Yoshua Bengio step readers through the incremental improvements to activation functions. They begin with what they posit are the most biologically plausible logistic sigmoid neurons, progress to Tanh neurons which work better for training multilayer neural networks, and ultimately arrive at rectifying neurons being an even better model of biological neurons than sigmoids while yielding equal or better learning performance than Tanh despite hard non-linearity and non-differentiability at zero. They contend that rectifying neurons create sparse representations with true zeros which are exceptionally suitable for naturally sparse data (Glorot et al 2011).

Writing from the computer science department at Stanford University, Andrew L. Maas, Awni, Y. Hannun, and Andrew Y. Ng detail how DNN acoustic models produce significant gains in large vocabulary continuous speech recognition systems, how new work with rectified linear (what they refer to as ReL) hidden units produce even further gains in final system performance versus sigmoidal nonlinearities; and they analyzed hidden layer representations to quantify differences in how ReL units encode inputs compared to sigmoidal units. They also assessed a modified version of the ReL unit that possesses a gradient more conducive to optimization to further enhance deep rectifier networks (Maas et all 2013).

What these works illustrate, whether directly stated or inferred by the reader, is that for most modern neural networks, the ReLU activation function outperforms sigmoidal functions in learning capability with like-for-like hardware system architectures.

Data

The MNIST database was chosen to illustrate how the ReLU activation function outperforms the sigmoid function, primarily by mitigating the vanishing gradient problem. It is a widely used compilation of handwritten digit images. MNIST stands for Modified National Institute of Standards and Technology, the organization that originally created the dataset. It is a benchmark dataset that is frequently used in the field of machine learning and computer vision to evaluate and develop algorithms for tasks such as image recognition and classification.

The MNIST database consists of a training set of 60,000 labeled examples and a test set of 10,000 labeled examples. Each example in the dataset is a grayscale image of size 28x28 pixels, representing a handwritten digit from 0 to 9. The digits are manually written by various contributors, ensuring diversity in the range of writing styles and variations. It serves as a standard benchmarking tool for evaluating the performance of machine learning models and algorithms.

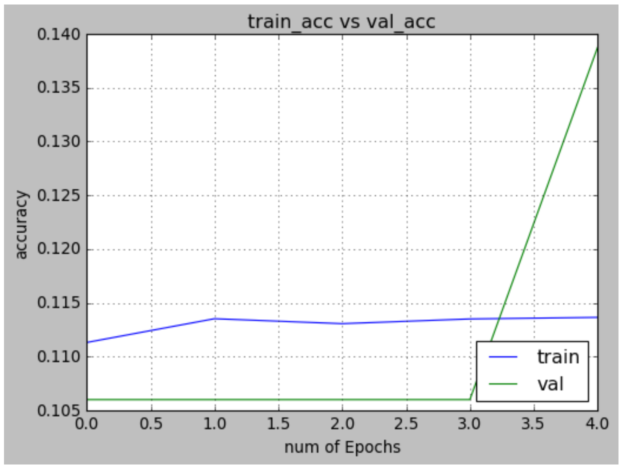
The MLP will be trained on the 60,000 training samples and then tested on the 10,000 test samples. Before training, the pixel data will be normalized by dividing by 255 so that the values will be between 0 and 1. As the original labels are values from 0-9, we convert all the data to vector format. For example, the data for the handwritten digit “1” will be represented in the vector format 0,1,0,0,0,0,0,0,0,0 and the digit “5” will be represented as 0,0,0,0,0,1,0,0,0,0. We then flatten the pixel data originally represented as 28x28 pixels so that it, too, is represented in in vector format; we do this by dividing by 784 (the total number of pixels for a handwritten image). Both the training data and the test data are converted to vector format.

Methods

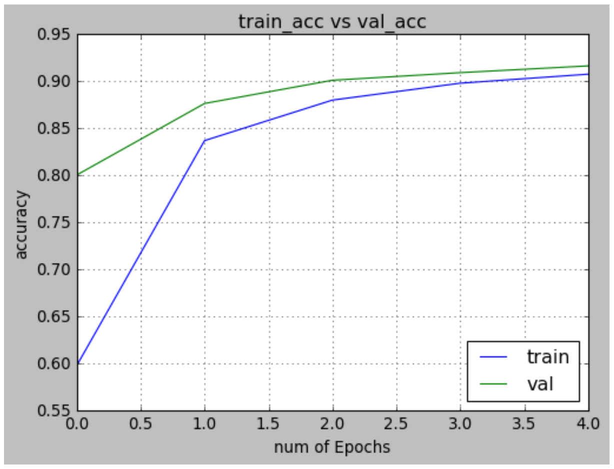
We are comparing the sigmoid and ReLU activation functions for use within a multilayer perceptron and demonstrating how the ReLU function outperforms the sigmoid in speed and learning ability. TensorFlow is used to train the model and to test the trained model on the test set. After model training and testing, comparisons will be made between the time to complete each epoch, first with five epochs, then with ten, and the accuracy value for each iteration. Within our model’s code, each of the number of hidden layers and number of output layer classes are held the same for both sigmoid and ReLU passes, to ensure a one-to-one comparison. The code for each of the sigmoid and ReLU passes is largely the same, with the primary difference being the selection of the activation function for each layer of the neural network. Each network is constructed of an input layer, three hidden layers, and an output layer of ten nodes or classes, as we're trying to identify the ten digits of 0-9. For the sigmoid network, the activation function applied to the input and hidden layers is sigmoid, with SoftMax used on the output layer; for the ReLU network, the activation function applied to the input and hidden layers is ReLU, with SoftMax used on the output layer.

Results

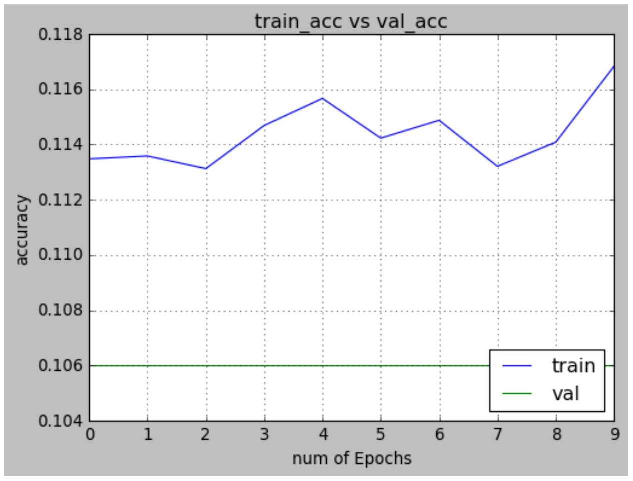
Sigmoid – 5 Epochs



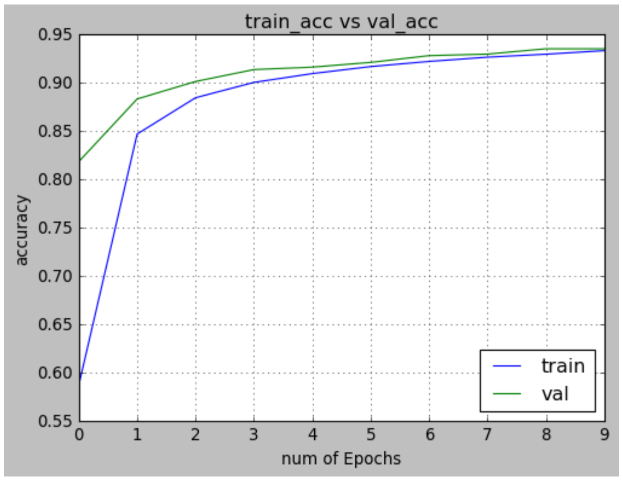
ReLU – 5 Epochs



Sigmoid – 10 Epochs



ReLU – 10 Epochs



In our five-epoch pass through the code, the ReLU showed no speed advantage as both networks ran for essentially equivalent durations. The ReLU accuracy advantage, however, was stunning, as by the fifth epoch val_accuracy reached ~91%, whereas the sigmoid stalled at 10.6% – no epochal improvement and not a good convergence. In our ten-epoch pass, the ReLU again showed no speed advantage. The ReLU increased its accuracy advantage, however, as by the tenth epoch val_accuracy reached 93.7%, whereas the sigmoid again stagnated at 10.6% with no epochal improvement.

Discussion

It was somewhat surprising that the MLPs utilizing the ReLU activation function didn't display a speed advantage over the sigmoid. We suspect the relatively smaller size of the MNIST dataset is the primary factor there, as each epoch in both networks processed in under twenty milliseconds; and a dataset of vastly larger size and complexity may indeed demonstrate the expected speed advantage. Also noteworthy is that in our test runs of code functionality, where sigmoid and ReLU were used on the output layer of each network, respectively, in addition to the input and hidden layers, the results were different, though only for the ReLU network. The sigmoid network, regardless of output layer function, stalled in its learning ability at 10-11% accuracy. Within the ReLU network, updating the output layer function from ReLU to SoftMax dramatically improved the result – from ~9% to what we see above with the 93.7% result. Indeed, in those test runs, prior to optimizing the architecture of each network, the MLP utilizing sigmoid activation functions outperformed the ReLU. Once optimized, however, the learning advantage of ReLU was clear and obvious. This serves as an illustration of other work in the field regarding the combination of activation functions within a single neural network. One such work was Abien Fred Agarap's *Deep Learning Using Rectified Linear Units (ReLU)*, in which it's noted that conventionally, within DNN's, ReLU is used as the activation function, with SoftMax used as the classification function, and the use of classification functions other than SoftMax is explored (Agarap 2018).

Conclusions

While there may remain a place for the sigmoid activation function within shallow networks with one, maybe two, hidden layers, and within recurrent networks, the vanishing gradient problem it suffers from is too great a hurdle for deeper neural networks. As demonstrated by our per-iteration and final accuracy results, the MLP utilizing the ReLU activation function displayed a 10x increase in learning ability. It is important to note that network architecture optimization plays a role in ReLU's advantage – at least, for datasets with particular characteristics, as evidenced by our code functionality testing. The AI will not do all the work, researchers still must manipulate the code to extract the best results, optimizing the number of hidden layers and epochs and deciding when-and-if to use certain activation functions. We are still of use yet.

Future Work

Further research and analysis of combining multiple activation functions within a single neural network would be beneficial, perhaps revealing more optimal combinations. Research into a mathematical proof of why the ReLU activation function learns better than the sigmoid is another area for continued study. For our own continued research, an exploration into how the ReLU can increase the performance of other deep neural networks - perhaps following Nair and Hinton's work with it improving the Restricted Boltzmann Machine - seems a natural extension.

References

Hochreiter, Sepp. "The vanishing gradient problem during learning recurrent neural nets and problem solutions." *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 6*, no. 2: 107-116. 1998.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *Nature* 521 no. 7553: 436-444. 2015.

Cireşan, Dan Claudiu, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. "Deep, big, simple neural nets for handwritten digit recognition." *Neural computation* 22 no. 12: 3207-3220. 2010.

Nair, Vinod, and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In Proceedings of the 27th International Conference on Machine Learning (ICML-10): 807-814. 2010.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics: 315–323. 2011.

Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models." In Proc. icml, vol. 30, no. 1: 3. 2013.

Agarap, Abien Fred M. "Deep Learning Using Rectified Linear Units (ReLU)." *arXiv preprint arXiv:1803.08375*. 2018.

McCulloch, W.S. and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5: 115-133. 1943.
McCulloch and Pitts invent the perceptron.

Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65, no. 6: 386. 1958.
Rosenblatt introduces the multilayer perceptron with input, hidden, and output layers.

Minsky, Marvin, and Seymour A. Papert. "Perceptrons." Cambridge, MA: MIT Press 6: 318-62. 1969.
Minsky and Papert show it's impossible for a single-layer perceptron to learn the XOR function.

Werbos, Paul. "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA. 1974.
Werbos introduces the modern iteration of backpropagation.

Linnainmaa, S. "Taylor Expansion of the Accumulated Rounding Error." BIT 16, 146-160. https://doi.org/10.1007/BF01931367. 1976.
Linnainmaa publishes a technique that is mathematically equivalent to backpropagation.