2019 IEEE International Conference on Big Data (Big Data)

On Neural Network Activation Functions and Optimizers in Relation to Polynomial Regression

John Pomerat
University of South Alabama
Mobile, Alabama
jop1721@jagmail.southalabama.edu

Aviv Segev
University of South Alabama
Mobile, Alabama
segev@southalabama.edu

Rituparna Datta
University of South Alabama
Mobile, Alabama
rdatta@southalabama.edu

I. Introduction

Recently, research in machine learning has become more reliant on data-driven approaches. However, understanding the general theory behind optimal neural network architecture is, arguably, just as important. With the proliferation of deep learning and neural networks, finding optimal neural network architecture is vital for both accuracy and performance. Recently, extensive research on neural network architecture has been performed [3], [4]. Additionally, while there has been plenty of research on hidden layer neural network architecture [2], activation functions are often not considered. In a network, an activation function defines the output of a neuron and introduces non-linearities into the neural network, enabling it to be a universal function approximator [12]. In terms of activation functions, one significant paper is Krizhevsky's seminole work on ImageNet classification and the creation of the ReLU activation function [1]. In the paper, Krizhevsky outlines the construction of an image recognition model using the Rectified Linear Unit activation function (ReLU) for the ImageNET LSVRC-2010 competition which outperformed the state-of-the-art image recognition systems at the time [1]. Since then, ReLU has increased in popularity. In their 2018 conference paper, Bircanoğlu and Arica, with the assistance of 231 distinct training procedures, found ReLU to be the best general activation function [12]. In addition to comparisons of activation functions, Nwankpa, Ijomah, Gachagan, and Marshall conducted a meta analysis of the field of research centered around activation functions and found ReLU to be the most popular activation function choice [5]. In terms of optimizers, gradient descent has historically been the most popular loss optimization algorithm, but with Kingma and Ba's 2014 paper [8], Adam: A Method for Stochastic Optimization, Adam optimizer is slowly becoming the industry standard [11]. In their paper, Kingma and Ba cleverly combine momentum descent, RMSprop, and Adagrad optimization into one algorithm, Adam (or adaptive moment estimation) [8]. In addition to Adam, there are plenty of other optimizers to choose from, including gradient descent, RMSprop [9], Adagrad [10], and Adadelta [7]. Recently, many breakthroughs have been made in terms of neural network performance, improved GPU performance and adaptation to deep learning tasks has created massive efficiency increases for the whole field of machine learning. Furthermore, as machine learning becomes increasingly optimized, the importance of efficiency improvements will continue to rise. Thus, understanding the optimal activation function and optimizer choice for a neural network is relevant. The goal of this paper is to make comparisons between activation functions, optimizers, and, more generally, entire neural network architectures, through measured error in a training environment. In this paper, we examine the performance of a wide variety of neural network configurations on randomly generated polynomial data sets of fixed degree. To do this, we compare various neural network activation functions and optimizers while controlling for hidden layer configurations and degree of the underlying polynomial dataset. Curiously, we find that the Sigmoid activation function is more accurate than ReLU and Tanh for regression tasks on low-featured polynomial data. We also reach the same conclusion regarding Stochastic Gradient Descent (SGD) in comparison to the Adam optimization function and Root Mean Square Propagation (RMSprop). Additionally, we observe that SGD is more efficient in the short term for finding local minimums than Adam or RMSprop; however, after sufficiently many epochs, performance differences between the optimizers vanished.

II. Methodology

A. Generation of Polynomial Data

The tests in this paper were performed on artificially generated data. To begin, we used a random normal distribution to create arrays containing 1000 x-values with a mean of 0 with standard deviation varying between arrays. After the x-values had been generated, a degree, n, of a polynomial was randomly chosen

$$p(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + ... + a_nx^0 \quad (1)$$

the a_i (coefficients) in this equation were also randomly generated using a normal distribution with a mean of zero. Next, y-values were calculated by plugging the generated x-values into the polynomial (Fig. 1).

$$y = p(x) \quad (2)$$

After the y-values had been calculated, random noise was added to each y-value and the amount of noise was varied

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

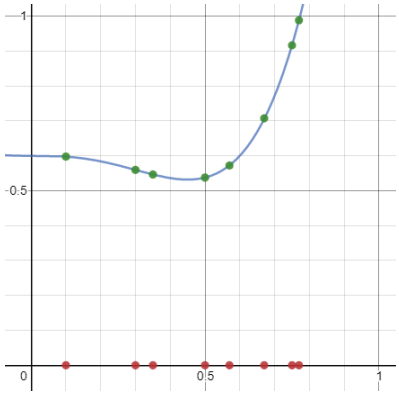

Fig. 1: Random x values (red) and calculated y values (green) on the generated polynomial function (blue).

between datasets. After the data had been generated, y-values and x-values were combined to form single featured datasets, each containing 10000 values, 8000 for training and 2000 for testing.

B. Neural Network Architecture

The neural networks we use in this paper are configured in the following way. The neural networks run regression on polynomial datasets of varying degree generated from the method described above. Additionally, each neural network normalizes input data and initializes its weights and biases to the middle of the range of the corresponding activation function in order to reduce variance. Each neural network is equipped with an optimizer, an activation function, and a hidden layer architecture that all vary from test to test. We ran each neural network for only 10 epochs to prevent overfitting, as the small size of the polynomial datasets define a small search space.

C. Activation Functions

In a neural network, if each neuron activation was calculated purely as a weighted sum of its activations, then that neuron would be a linear function. Because of this, a linear function could be built to model the output of the entire network. This is the purpose of an activation function. In machine learning, an activation function is used to introduce nonlinearities into the neural network. The three common activation functions that will be examined in this paper are the following:

Sigmoid (or Logistic curve)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

Tanh (Hyperbolic Tangent)

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{4}$$

ReLU (Rectified Linear Units)

$$f(x) = \max(0, x) \tag{5}$$

D. Loss functions and Optimizers

A loss function is a function defined on a neural network and a training set. A loss function provides a way of measuring how far off the neural networks predictions are on a training set. An optimizer is the algorithm used to minimize the loss function. The most common optimizer is Gradient Descent or Stochastic Gradient Descent (SGD) [6]. Gradient descent works by first calculating the derivative of the loss function and then changing the weights and biases in the network in the direction of the calculated gradient reducing the loss function in a series of steps. Stochastic gradient descent works the same way, but instead uses a single or a handful of training examples to calculate the gradient. Another optimizer is Root Mean Square propagation (RMSprop), which uses momentum to find minimums of the loss function computationally faster compared to traditional optimizers. Additionally, the Adam optimizer, first introduced in 2014, inherits some of the benefits of RMSprop [8]. One of the key differences between stochastic gradient descent and optimizers like Adam, is the presence of momentum. Momentum enables an optimization algorithm to use the average of the previous batches of gradient descent to take the neural network to a minimum faster. The momentum is calculated using a running average:

$$A_t = \beta A_{t-1} + (1 - \beta) X_t \tag{6}$$

Where A_t is the moving average of the loss for some training/testing example t, calculated from X_t, with β determining the number of previous values on which A_t is calculated.

III. Experiments

A. Comparing Activation Functions

The first test was performed to compare the performance of various neural network activation functions on single featured polynomial input data. First, we generated datasets in Numpy using the methodology described in II-A. We then normalized the data and fed it through neural networks suited for regression configured as described in II-B. We ran the neural networks for 10 epochs with batch size of 5, and Mean Squared Error (MSE) for error measurement. In the first test, we equipped the neural networks with the Adam optimization algorithm (with a learning rate of 0.001) which was kept constant throughout the test while the activation functions and the structure of the networks were varied. The average MSE for each activation function during the test can be found in (Fig 2).

Interestingly, even though ReLU is considered a better choice than Sigmoid or Tanh, Sigmoid performed 10.1% better than ReLU across the whole test. This is likely due to the fact that single featured input data does not suffer from the vanishing gradient problem. Curiously, the hidden layer configuration seemed to make little difference in the experiment. The only clear trend is a negative relationship between MSE and dataset polynomial degree.

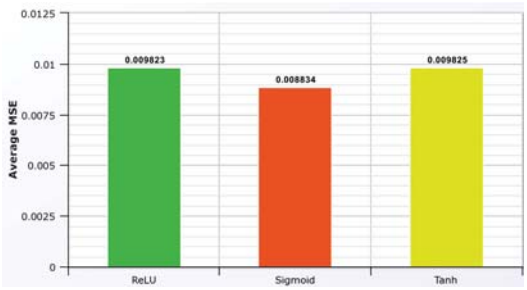

Fig. 2: Comparison of Activation Function Performance

B. Comparing Optimizers

The second test was performed similarly to the first one, comparisons were made between a collection of optimizers including the Adam optimizer (learning rate of 0.001), SGD (Stochastic Gradient Descent) and RMSprop (Root Mean Square Propagation). All of the neural networks ran on the generated polynomial data, used ReLU as an activation function, and had varied hidden layer structure. Additionally, the number of epochs was reduced to 3 to enable comparison by avoiding identical accuracy values and to prevent overfitting. We found that as degree increased, the optimizers began to perform similarly. Again, it seems that neural network hidden layer structure has virtually no impact on results. This time, we observed that SGD performed far better (86.8%) than Adam or RMSprop for low-featured polynomial input data of degrees 1-3. This is likely due to SGD not using any kind of momentum and is thus easily able to find a lesser minimum without overshooting on fewer epochs.

C. Finding Optimal Configuration

We performed a final test to make comparisons between combinations of activation functions and optimizers. To test this, we equipped a set of neural network architectures with the best performing activation function (Sigmoid) and the worst performing optimizer (Adam). We then compared this set of neural networks to the same set instead equipped with the worst performing activation function (ReLU) and the best performing optimizer (SGD). The test was run the same as in the previous two experiments with the difference being that the neural networks ran with only a single epoch. After the test, the neural networks equipped with SGD and ReLU performed better than those with the Adam optimizer and Sigmoid. To observe the change in accuracy as the number of epochs increased, this test was repeated 4 times for increasing epochs, and the MSE (rounded to the nearest ten-thousandth) across all the neural network architectures and activation function optimizer combinations was averaged and recorded. The results can be found in (Table I).

As epochs increased, the difference in accuracy began to shrink and at 5 epochs the models became overfitted, signified by an increase in MSE. Therefore, the SGD/ReLU configuration was able to find a smaller local minimum than the Adam/Sigmoid configuration for the first two epochs.

TABLE I: Adam and Sigmoid vs SGD and ReLU (MSE)

Epochs	Adam/Sigmoid	SGD/ReLU
1	0.0356	0.0112
2	0.0109	0.0099
3	0.0090	0.0091
5	0.0105	0.0106

However, after the 2nd epoch the two configurations found a strong local minimum and after the 3rd epoch, the models were overfitted. This result, combined with the results from Section III-B imply that SGD finds lower local minimums on few epochs than Adam optimization or RMSprop for single featured polynomial datasets. As epochs increase, however, accuracy differences from one configuration to another begin to vanish.

IV. Future Work

One possible direction for future work could be generalizing the results of this experiment beyond artificially generated polynomial datasets, and making more comparisons of activation functions and optimizers. Understanding activation functions and optimizers in the context of deep learning is critical in order to construct a general theory of optimal neural network architecture which is important for both computational efficiency and accuracy in industry applications of deep learning.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems, 2012.
[2] S. Xu and L. Chen. A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining, International Conference on Information Technology and Applications, 2008.
[3] D. Hunter, H. Yu, I. Pukish, S. Michael, J. Kolbusz and M. Wilamowski. Selection of proper neural network sizes and architectures - a comparative study, IEEE Transactions on Industrial Informatics, 2012.
[4] W. Liu, W. Zidong, L. Xiaohui, Z. Nianyin, L. Yurong, and A. E. Fuad. A survey of deep neural network architectures and their applications, Neurocomputing 234, 11-26, 2017.
[5] C. Nwankpa, Winifred Ijomah, A. Gachagan, and S. Marshall. Activation functions: comparison of trends in practice and research for deep learning, arXiv preprint arXiv:1811.03378, 2018.
[6] F. Rosenblatt. The perception: a probabilistic model for information storage and organization in the brain, Cornell Aeronautical Laboratory, 1954.
[7] M. D. Zeiler ADADELTA: an adaptive learning rate method, arXiv preprint arXiv:1212.5701, 2012.
[8] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[9] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning, 2012.
[10] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12, 2011.
[11] A. Karpathy. A peek at trends in machine learning, Medium.com, 2017 accessed 8/16/19.
[12] C. Bircanoğlu and N. Arica. A comparison of activation functions in artificial neural networks, Signal Processing and Communications Applications Conference, 2018.