A study of varied orientations of rectified linear activation functions in artificial neural networks

Thomas Sharp-Riley ID: 2705696 UPI: tsha333
Sanjid Rahman ID: 327114511 UPI: srah064

Supervisor:
Dr. Patricia Riddle

Department of Computer Science
University of Auckland

Submission date: October, 2014

Abstract:

Artificial neural networks have been a dominant learning algorithm since its inception in the late 1950s. Overtime, it has evolved from being a mere implementation of perceptron learning rule to being an elegant implementation of deep belief networks. This transformation has made neural networks a state of the art technique particularly because of its ability to generate its own simpler version of features alongside the ones that are provided from observations. Consequently, neural networks are now being applied on problems as simple as regression analysis to problems as complex as driving cars using pattern recognition. Much of this progression can be attributed to the inherently simple and flexible implementation principles of neural networks. Historically, this flexibility has been demonstrated through the mere incorporation of an untried approach to neural networks. In light of this philosophy, our research addresses issues including role, gradient and properties of an inherent feature of neural networks, "activation functions" by investigating functions that are not implemented in the literature. Concretely, we have performed our experiments on open source platform (`neuralnet` package of R) and datasets (`Titanic` and MNIST) in order to demonstrate the impact of variations of the popular rectified linear activation function.

Table of contents:

List of figures:

1. <u>Introduction:</u>

Over the past few decades, artificial neural network has been a crucial learning approach in the world of machine learning. The underlying principles of neural network are motivated by the structure of the human brain and how it learns. Neural network models consist of "neurons" (which are meant to represent the cell body of neurons in the brain containing the nucleus). A neuron can take in input from training data or from other neurons via input wires of a certain value of weight that is either assigned to them or updated in the learning process (which mimics the transfer of impulse to a neuron in the brain from sensors (e.g. eyes or pain receptors) or other neuron(s) respectively via dendrites). Although a biological neuron makes use of ions but an artificial neuron has to accumulate and transfer its input(s) mathematically using a function called activation function. The output from a neuron is either passed on to other neurons or used to produce the final output (which represents a spike of electricity transferred by a neuron in the brain to other neurons or effectors (e.g. muscles) respectively via axons). Learning takes place by adjusting the weights to and fro these neurons to minimize the error between the output from the contribution of the neural network and the output that is desired (which is representative of how an infant learns to stop touching a steaming cup of coffee).

Activation functions have a very important role to play on the architecture of neural networks. A better activation function not only makes a model more accurate but also makes it more fast. Since its introduction, rectified linear functions have performed consistently better than the two classic asymptotic activation functions, namely, sigmoid and hyperbolic tangent. The reason for this is, rectified linear does not have to suffer the 'vanishing gradients' at extreme input values that its two counterparts have to (Glorot et al. , 2010). Furthermore, with enough observations, use of rectified linear also saves the trouble of pre-training. However, researchers still struggle to explain precisely how it manages to attain such success. With the motivation that there is room for improvement, we contribute by providing a comparison of a few variations of rectified linear to the asymptotic functions. Our research features the role of gradient in activation functions and keeps the practice of simply implementing a novel approach to neural networks alive with the hopes of achieving better results.

2. <u>Problem</u>

The intricate nature of neural network has proven itself to be promising in different eras of the last half a century. However, researchers have struggled to find the right methods pertaining to neural networks to tackle with three fundamental problems. Firstly, what approach should be taken to decisively minimize the error between the real/target output and the output generated by a neural network? In the world of machine learning, this problem is defined as consistently <u>producing</u> <u>global</u> <u>minima</u> during the minimization of the error function. Second problem is how to compute this minimization process faster? This is defined as making the rate of convergence faster. Thirdly, how well a hypothesis from a learning process perform on new or unseen data? This is defined as ensuring overfitting does not occur. In order to bring about improvements pertaining to these problems, the most vital transformation of the structure of neural networks has been through its activation functions. The journey from step function to sigmoid function to rectified linear drastically improved the performance of neural networks over time. More importantly, since they are an inherent property of neural networks, the transformation of activation functions have also improved the aesthetic structure of neural networks.

Despite its success, the geometric property of rectified linear functions has always been a matter of debate. Given the flatness in the negative side of the horizontal axis, why does rectified linear not act like the saturation region of a sigmoid function? There have been attempts to hypothesize its proper functionality. Researchers have hypothesised its success by attributing rectified linear networks to its inherent ability to contribute in sparse representation of models and similarities with biological neurons (Glorot et al., 2011). Nonetheless comprehensive studies should still be conducted to at least understand the nature of activation functions in neural networks if not for making improvements upon the existing ones.

3. Methodology:

i. Datasets

Our study is based on two datasets, namely “titanic” and MNIST. For the purposes of our tasks, we have pre-processed them in the following manner:

- Titanic data set has 891 observations. 500 random (without replacement) observations were chosen as training set and the rest for test set. A total of 6 features were used during training.

- MNIST is a popular dataset for evaluating machine learning approaches. It is a digit recognition dataset of 28 X 28 pixels (784 features). Originally, the dataset has 60000 observations but we have randomly chosen 1000 due to time and resource limitations. The subset is then split into 70% for training and 30% for testing. All 784 features were used during training.

ii. Tool

Despite its popularity, neural network algorithms have not been adequately implemented in open source platforms. Amongst the few that do, the `` `neuralnet` `` package in R has the capacity to implement deep neural networks (multi-layered perceptrons with 3 or more hidden layers). This package is discussed in the paper by Günther et al, (2010) which talks about how the package is implemented. It implements two approaches to minimise the cost function: back propagation and resilient backpropagation. Use of the latter makes computation faster because it utilises the sign of the derivative during gradient descent. The options of error functions in the package are sum of squared error and cross entropy. For learning in case of both datasets, the parameter `` `linear.output` `` is turned to false since these are classification problems. This parameterisation turns off activation functions in the output layer. In order to ensure just comparison of the approaches that we evaluate, the default settings of learning are retained with the exception of the following parameters:

- `` `stepmax` ``: This is the maximum number of iterations at which the algorithm stops converging. We have set the value to 2 x 10^9 which is close to the maximum value of integer that R can handle.
- `` `threshold` ``: This is the partial derivatives of the error function with respect to the weights ($\partial E/\partial w$). The default value of this stopping criteria is 0.01. However, the value of `` `threshold` `` can be thought of as the threshold that a learning algorithm must reach in order to call it successful (i.e. generate the weights for the prediction). Due to its sensitivity, this value has been changed in our experiment, the details and impact of which are described further in the results.

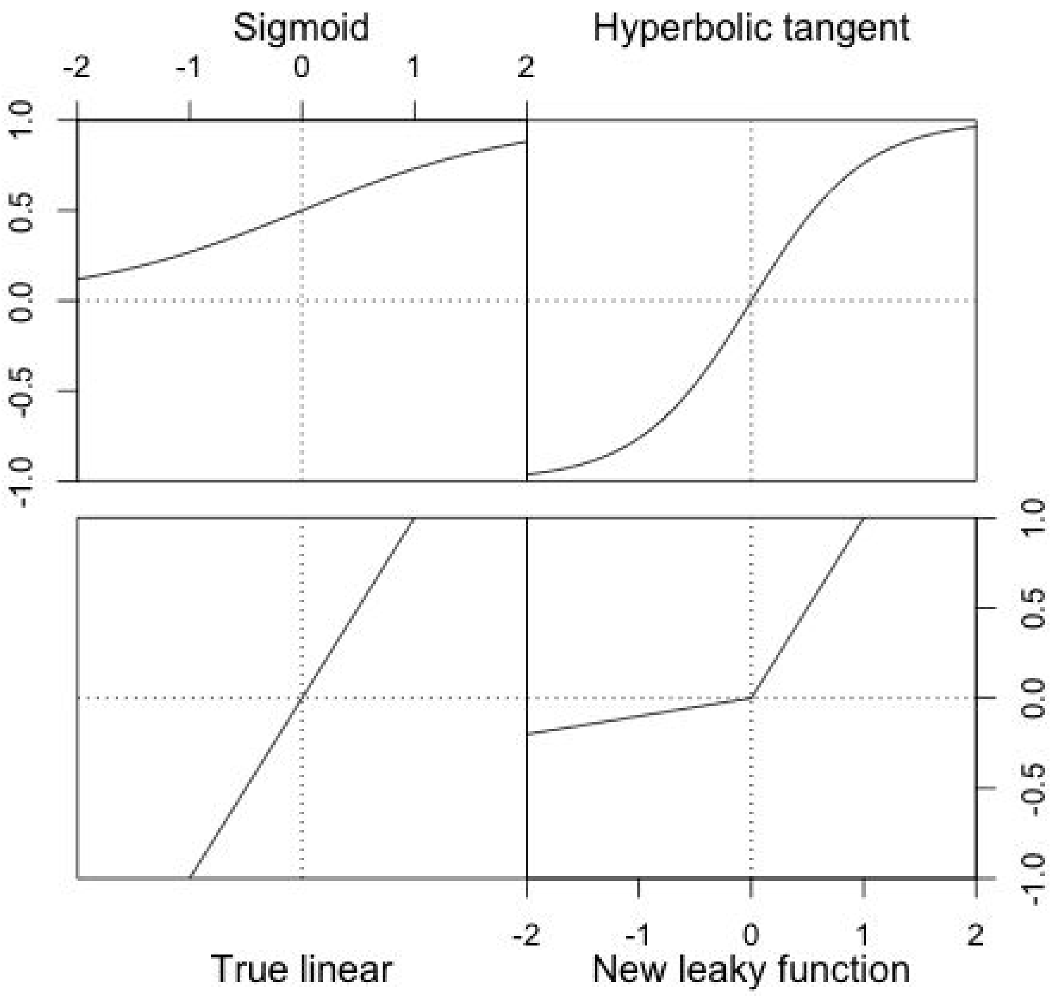
iii. Functions

The activation functions that came by default in the `` `neuralnet` `` package are `` `tanh` `` (hyperbolic tangent) and `` `logistic` `` (sigmoid unit). For these activations to be called during learning, in the source code, each of them has an expression of the function itself and an expression of its derivative. We have capitalised on these code blocks and implemented the activations functions that we are surveying in this papers. The additional function that we have finally implemented are selected from a very time consuming process of trial and error. A diagrammatic representation of all the functions that we have experimented with are given in Figure 3.1. Brief description of the new functions is given below:

- True linear: The first function is merely a linear function of gradient 1, passing through the origin. This is a particularly curious function because the values of input are mapped as themselves in the output.

- New leaky function: The second function is a piecewise function akin to rectified linear. The similarity is that it has a gradient of 1 in the positive horizontal axis. However, the difference is that the gradient in the negative horizontal axis is 0.1. The label in the diagram is `new leaky function`. This is because a similar function with a gradient of 0.01 in the negative horizontal axis has already been implemented in the literature by Maas et al. (2013). The term that they coined for their function is `leaky rectified linear`. We have discussed it in section 5.

- Finally the last function we had to adopt (not in the diagram), is also a leaky function, but with a gradient of 0.5 in the negative horizontal axis. Concretely, the new leaky function in the Titanic dataset and MNIST dataset have gradient of 0.1 and 0.5 respectively in the negative horizontal axis.



<u>FIGURE 3.1:</u> <u>Activation functions</u>

iv. <u>Setup</u>

A total of 1 to 4 hidden layers have been tried on the Titanic dataset each with 3 hidden units. Whereas, for the MNIST dataset, a total of 1 to 5 hidden layers have been tried with 2 hidden units on each. More hidden layers was used in the MNIST dataset because of its large number of features. Although, with more than 2 units in each layer, the computation becomes exceedingly expensive in the short timeframe.

There have been numerous trials and errors in the experiment. The most consistent results that could be captured by ensuring the same seed of randomness for each approach are documented in section 4. Due to variability in the parameters and the fact that the results are that of a single run, we have opted against any statistical tests for our exhaustive results.

4. Results and findings

i. Titanic

Threshold of 0.1: Challenges of convergence are faced early on by `tanh` and `sigmoid` functions in the Titanic dataset. Hence, the `threshold` value is set to 0.1 rather than the default of 0.01 in the `neuralnet` package. We note that, at a threshold of 0.1, the rate of convergence of the new functions (`linear` and `new leaky`) outperforms that of the asymptotic functions (`sigmoid` and `tanh`) especially as the number of hidden layers is increased. This is true with the exception of 1 and 3 hidden layers in case of `sigmoid function` but its worst case scenarios with 2 and 4 hidden layers indicate that they might have been fortunate anomalies. The fastest convergence happens with `linear` function however, increasing the number of layers has no impact on its accuracy. Overall accuracy of all the functions are comparable but in the `new leaky` function, it increases as the number of layers increase.

TABLE 4.1: Results from the Titanic data set

	No.of Layer	Tanh		Sigmoid		Linear		New Leaky	
Thresh->		0.1	0.5	0.1	0.5	0.1	0.5	0.1	0.5
Test accuracy	1	78.26	64.96	78.26	NA	77.49	79.8	77.49	80.31
	2	74.68	66.75	75.7	79.8	77.49	79.8	76.73	80.56
	3	75.96	74.94	74.94	80.56	77.49	79.8	79.54	80.31
	4	68.8	67.01	77.75	NA	77.49	79.8	80.31	79.8
Steps	1	1832	196	667	18	527	18	67904	26727
	2	27455	137	26965	145	320	145	1329	4215
	3	305724	815	378	444	826	444	15400	2541
	4	299318	49	391616	10	640	10	49852	16181
Training error	1	31.37	49.6927	30.21	59.474	34.49	37.47	34.30	38.52
	2	30.59	46.9214	25.88	36.1031	34.49	37.47	33.02	37.84
	3	27.43	35.2555	31.01	32.0996	34.49	37.47	33.00	38.49
	4	35.53	50.3386	21.25	59.477	34.49	37.47	34.06	38.34

<u>Threshold of 0.5:</u> To bring in a new vantage of evaluation, we have also ran our experiments with `threshold` value of 0.5. Unsurprisingly, the overall rate of convergence of all algorithms increase. However, `tanh` suffers the impact on its accuracy more than the rest. For 1 and 4 hidden layers, `sigmoid` function overfits and does not give any right prediction at all but does well with 2 and 3 hidden layers. Also, on a useful note, the `new leaky` function demonstrates how overfitting is overcome by maintaining its accuracy despite its increase in training error compared to the `threshold` of 0.1. From this experiment, we find that, with a high `threshold` value, the new functions can still give useful results. Our findings of MNIST experiment supports this finding

ii. <u>MNIST</u>

Building upon the findings of the `titanic` dataset, the `threshold` parameter becomes more useful in the MNIST dataset. For the asymptotic functions, the `threshold` value need to set very small at 0.00001 and the table below shows the result of that. It might seem like the value needs to be set even lower, but doing so brings along the vanishing gradient problem of the asymptotic functions. As a result, other than 1 and 2 hidden layers with sigmoid functions, none of the instances of the asymptotic function gives useful results. The new functions however, demonstrate a much better performance. We also note that the difference in the in-sample and out-of-sample accuracy is an exhibition of high variance for the approaches with the new functions and we believe that this difference can be reduced further by using more training data.

It is important to refer back to the note that gradient of the `new leaky` function in the negative horizontal axis for MNIST is 0.5 and not 0.1 as for Titanic. This is because, convergence with the former is slow to the extent that it does not reach the `threshold` value with `stepmax` number of epochs.

TABLE 4.2: Results from the MNIST data set

No of layers ---->		**1**	**2**	**3**	**4**	**5**
TANH:	**steps**	95	31	32	35	27
	time(s)	5.381	5.099	5.149	5.145	5.092
	in.acc	10.29	10.29	10.29	10.29	10.29
	out.acc	9	9	9	9	9
SIGMOID:	**steps**	185	199	60	65	58
	time(s)	5.815	5.834	5.243	5.267	5.291

	in.acc	17.29	20.86	10.29	10.29	10.29
	out.acc	14.67	15.33	9	9	9
LINEAR:	**steps**	101911	161204	9484	118	29005
	time(s)	398.315	658.383	45.644	5.538	139.42
	in.acc	58.29	58	56.29	33	57.86
	out.acc	11.33	10	9	16	10.67
NEW LEAKY:	**steps**	3964	994081	23813	557	8713
	time(s)	20.622	4045.489	106.207	7.488	45.604
	in.acc	48.71	59.43	60.57	44.14	57.14
	out.acc	14	9.33	15	16.67	12.67

5. Related work:

Variations of rectified linear have been tried before particularly based on the fact that the flatness of rectified linear functions in the negative side of the horizontal axis might cause it to act like the saturation region of a sigmoid or hyperbolic tangent function. Maas et al. (2013) investigate this geometric property of rectified linear units by using a small value of gradient for the horizontal line in question and calling the function "leaky rectified linear". Given their motive of proposing an activation function that is not flat, their results have been counterintuitive. The result of their experiment of speech recognition system indicate that despite a significant improvement over sigmoid unit, leaky rectified linear performs almost identically with true rectified linear. Nair et al. (2010) used a smoother versions of rectified linear in order to recognise and overcome its non-linearity at the origin of the axes. However, their implementation of a function called `softmax` did not achieve more success than rectified linear.

Amidst of geometric changes to rectified linear, changes in approach have taken places as well. The performance of rectified linear is enhanced using the technique called dropout (Hinton et al, 2012). Dropouts can afford to enjoy the benefit of starting with a large learning rate and having it decayed gradually. This is a good example where incorporation of different machine learning techniques as well as adaptations in parameter settings can improve the performance of the overall learning approach.

Machine learning can be thought of as programs that makes programs. The idea is particularly remarkable when it comes to neural network by virtue of its capacity to "invent" intrinsic features. In the more recent past, this quality of neural networks has been enhanced when Goodfellow (2013) proposed an equation for maxout neuron that can represent any convex function and Agostinelli (2015) proposed an approach where along with the weights, the parameters of the activation function of each neuron unit are also learnt during gradient descent. Both use the same standard benchmark datasets and on an approximate level, the results of the former outperforms that of dropout with rectified linear and the latter outperforms the former. Numerically and theoretically, these two approaches of activation functions produce results of the best performing neural network models to the best of our findings in the current status quo. We take inspiration from these papers and we feel that research that we conducted could be considered a segment of them.

5. Future work:

In an ideal experiment with sufficient time and computational power, we could work on the following:

I. More variations of gradient

We have identified some very important roles of gradient in activation functions in the particular experiment that we have conducted. For instance, for linear piecewise functions, a negative gradient in the negative horizontal axis and increasing the magnitude of gradient in the positive horizontal axis are both detrimental to the rate of convergence. We would like to investigate this further and study these features in other datasets and see if there can be a generalisation made. Such a study would be able to support works on adaptive activation functions like the one conducted by Agostinelli (2015).

II. Tuning parameters

Given, `threshold` value makes a big impact in our experiment, we feel that in conjunction with a validation set, there can be some interesting and useful findings about the gradient in linear and asymptotic activation functions. At the same time, this would also make the experiment more systematic

III. More hidden units in each layer and bigger dataset

With time and computation power, it would be ideal to use more hidden layers and bigger portion of data, particularly in the case of MNIST dataset. We feel that this way, we could demonstrate statistically significant results and be able to generalise our findings.

Conclusion:

Our findings suggest that the overall performance of deep neural networks increase with linear functions compared to asymptotic functions as the number of hidden layers increase. Between the asymptotic functions, hyperbolic tangent performs worse than sigmoid in both our experiments. The change that we had to make to the gradient of the 'new leaky' indicates that gradient of does have an impact on the activations generated in neural networks and that it is worthwhile to change it based on the circumstances and requirements of learning.

REFERENCE

Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, P. (2015). Learning Activation Functions to Improve Deep Neural Networks. *Iclr 2015*, (2013), 1–9. Retrieved from http://arxiv.org/abs/1412.6830

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*(7), 1527–1554. doi:10.1162/neco.2006.18.7.1527

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv: 1207.0580*, 1–18. doi:arXiv:1207.0580

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, *9*, 249–256. doi:10.1.1.207.2059

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statisitics (AISTATS) 2011*, *15*, 315–323. doi:10.1.1.208.6449

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout Networks. *arXiv Preprint*, 1319–1327. Retrieved from http://arxiv.org/abs/1302.4389

Günther, F., & Fritsch, S. (2010). neuralnet: Training of neural networks. *The R Journal*, *2*(1), 30-38.

Maas, A., Hannun, A., & Ng, A. (2013). Rectifier nonlinearities improve neural network acoustic models. *ICML Workshop on Deep Learning for …*, *28*. Retrieved from http://www.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)* (pp. 807-814).

Zeiler, M. D., Ranzato, M. A., Monga, R., Mao, M., Yang, K., Le, Q. V., ... & Hinton, G. E. (2013). On rectified linear units for speech processing. *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (pp. 3517-3521). IEEE.

`MNIST DATA:` `http://yann.lecun.com/exdb/mnist/`

`TITANIC DATA:` `https://www.kaggle.com/c/titanic/data`

`CODE:` `https://github.com/sanjid25/R`