Activation function impact on Sparse Neural Networks

Adam Dubowski
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
a.dubowski@student.utwente.nl

ABSTRACT

While the concept of a Sparse Neural Network has been researched for some time, researchers have only recently made notable progress in the matter. Techniques like Sparse Evolutionary Training allow for significantly lower computational complexity when compared to fully connected models by reducing redundant connections. That typically takes place in an iterative process of weight creation and removal during network training. Although there have been numerous approaches to optimize the redistribution of the removed weights, there seems to be little or no study on the effect of activation functions on the performance of the Sparse Networks. This research provides insights into the relationship between the activation function used and the network performance at various sparsity levels.

Keywords

Artificial Neural Network, Sparse Evolutionary Training, Activation Function, Accuracy, Sparsity, Sparsity Sweep.

1. INTRODUCTION

Artificial Neural Network (ANN) is a powerful Deep Learning method inspired by the human brain architecture [22] which has been adapted for speech recognition, computer vision, natural language processing, and many others [20].
Although constantly researched and improved, ANNs still have many flaws. While most researchers strive to improve algorithms' accuracy, it is the efficiency of deep learning solutions that is still one of its major limitations. The most common implementation of ANNs is the Multilayer Perceptron (MLP) which consists of fully connected neuron layers. That causes high redundancy because most of the connections have little or no impact on the accuracy of the model and could be removed [4]. One idea to improve the efficiency of ANNs is to limit the number of connections between layers, thus introducing sparsity in the model. Mocanu et al. suggested a training method called Sparse Evolutionary Training (SET) [18] supposed to deliver significantly greater performance by cutting up to 99.9% of connections [14].
Many other ideas on how to develop Sparse Neural Networks (SNNs) without training a fully connected model have been proposed, but most focus on the distribution of connections, while there has been little or no research on the impact of the choice of activation function used on the sparsity effect in the SNNs. This research investigates this relation to understand whether the activation functions currently used for densely connected networks [22] still behave reliably in the sparse context and which functions shall be recommended for SNN implementations. The findings will hopefully help researchers and developers make better decisions while choosing activation functions and sparsity levels for their models, thus improving the performance of their models. The research questions answered in the paper are:
RQ1 What is the impact of activation functions on the sparsity sweep and accuracy of SNNs?
RQ2 Is there a single activation function to be preferred in SNN implementations?
RQ3 Is the overfitting problem affecting SNNs with various activation functions differently?

To answer the above mentioned questions, a number of experiments have been conducted to compare accuracy and loss function scores during training. The experiments have been conducted for 5 sparse levels and the dense network, for each of the seven Activation Functions (AFs): ReLU, Sigmoid, SELU, SReLU, Tanh, Softplus, Softsign. Sparsity levels used ranged from 71.2% to 98.85%.

This paper is divided into the following sections. The Background section explains the main concepts needed to understand the methods and findings. Next, Related Work section provides information on the researchers working on the topic of Sparse Neural Networks. Then, the methodology and results are discussed separately, providing details about a suggested framework for analysis of the differences in the impact of activation functions and the results of this implementation on the SET algorithm and CIFAR10 dataset, respectively. Lastly, the Conclusions and Future Work section summarizes the findings and describes suggested future research areas.

2. BACKGROUND

2.1 Artificial Neural Networks (ANN)

Artificial Neural Networks is a family of networks built of input, hidden and output layers, inspired by the neural network of the biological brain. The most known example so far is the Multilayer Perceptron (MLP), a network consisting of densely connected layers of perceptrons [24]. While there are more network models created and used nowadays, this research will mainly focus on the MLP.

2.2 Sparse Neural Network (SNN)

In comparison to Dense Neural Networks, SNNs drop some of the connections, thus limiting the computational complexity of the algorithm. That allows them to not only train faster but also enables bigger network architectures, which typically results in an accuracy increase. Figure 1 visualizes an example of a sparsely connected architecture.

33rd Twente Student Conference on IT July. 3rd, 2020, Enschede, The Netherlands.

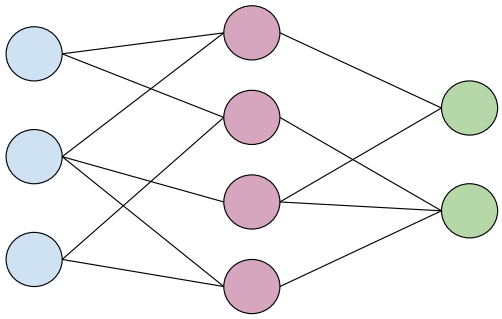


Figure 1: *Sparsely connected ANN layers.*

2.3 Sparse Evolutionary Training (SET)

Sparse Evolutionary Training is a method developed by Mocanu et al. [18] allowing to build SNNs by first starting with sparsely connected layers, iteratively growing new weights in random locations and pruning the weights closest to 0. The framework suggested in this paper uses SET implementation as a code base for training and sparsity sweep evaluation.

2.4 Activation Function (AF)

Activation Functions define the logic for each of the neurons within ANNs by modelling complex behaviour with non-linear mathematical functions [26]. Most popular functions are: Sigmoid , Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU) [22], Softsign and Softplus [15] and a number of ReLU modifications like SELU and SReLU [13]. Figure 2 visualizes the functions.

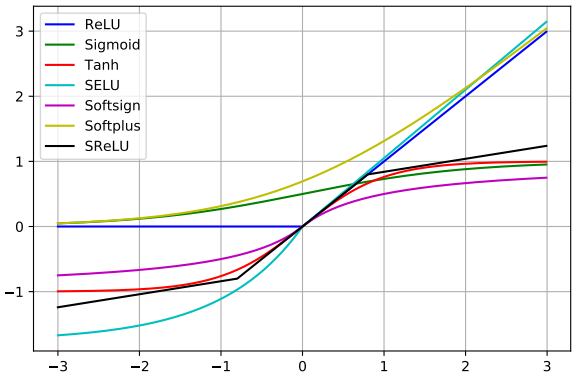


Figure 2: *Plots of activation functions used.*

2.5 SReLU

S-shaped rectified linear activation unit (SReLU) is a relatively little-known activation function suggested by Jin et al. [10] and used by Mocanu et al. [18] in their implementation of the SET algorithm. It combines 3 linear functions defined by 4 dynamic parameters. Note that the SReLU plot in Figure 2 is a sample presentation with set parameters. During training, the parameters are adjusted using back propagation [10].

2.6 Sparsity Sweep

Sparsity Sweep is the relation between the accuracy and sparsity of the model. At a first thought, with higher sparsity of the model, some information is lost which implies a drop in accuracy. However, the performance of sparse neural networks can sometimes be greater than that of their dense counterparts with the same hyperparameters when they are trained with sparse training methods e.g. SET [18].

2.7 Overfitting

Over- and underfitting are two concepts in Machine Learning describing the discrepancy between training and validation (testing) set performance. Underfitting can be noticed when there is significant potential to improve the validation set accuracy, while overfitting means the effect when the algorithm tries to memorize the training data, while reducing the test accuracy. Figure 3 explains the concepts in a visual form by showing example test and training errors during training.

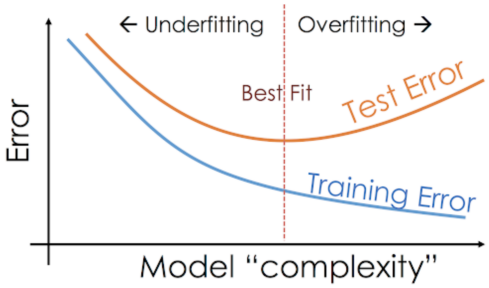


Figure 3: *Overfitting example from [25].*

3. RELATED WORK

Researchers have conducted a significant amount of research on SNN implementations. To start with, in 1990, LeCun et al. [12] suggested pruning unimportant weights from Neural Networks to achieve better results and performance. Using those insights, in 2015, Han et al. [9] suggested a training method which not only prunes unimportant weights but also retrains the model to tune the parameters. Experiments have shown [9] that this allows for a much lower computational complexity, but a limitation of that method is the maximum size of a fully connected network that can be trained. To solve the issue, researchers have been working on sparse model training methods. In 2017, Mocanu et al. [17, 18] have developed Sparse Evolutionary Training (SET) as a way to train a SNN from sparse instead of pruning a fully connected and trained ANN. The method removes weights close to 0 in a set interval and then distributes new ones randomly. Bellec et al. [1] suggested a method for supervised rewiring of the network during training keeping the total number of connections low using stochastic sampling theory. The method has been called Deep Rewiring (Deep R) due to its extensive rewiring during the Deep Learning training process. Another approach, the Neural Network Synthesis Tool (NeST) initializes a sparsely connected network and updates parameters according to gradient information and performs pruning based on the magnitude of connections and neurons [3]. SET has proven to be efficient enough to run a network with over million neurons on a normal laptop [14] and to be more accurate than Deep R [5]. Lapshyna [11] proposed an improvement to the weight update process in SET by gradually removing connections during training related to the increase in the accuracy level. It was proved that the number of connections can be vastly limited with negligible loss of accuracy. Bird et al.[2] came up with another way to improve the current algorithms by using Dendritic Normalisation to improve learning performance, based on the SET algorithm. He proved that the costs and accuracy can be vastly improved, providing reliable quality for the future research. In 2019, Mostafa et al. [19] suggested a method called Dynamic Sparse Reparametrization (DSR) presented as more efficient and accurate than the above-mentioned examples. It makes use of adaptive thresholds for the number of weights pruned and automatically reallocates parameters across layers, which avoids a fixed sparsity level. Sparse Momentum is a method redistributing weights relatively to mean momentum,

both between layers and within each of the layers. DSR and Sparse Momentum have proven slightly more accurate than SET [5]. Based on the works of their fellow researchers, a group of researchers from Google and DeepMind has come up with a Rigged Lottery (RigL) [6] method for training SNNs with fixed complexity without accuracy loss. RigL removes connections based on weight magnitudes and adds new ones according to gradient information. This method has achieved the highest accuracy level and relatively low sparsity sweep, compared to other techniques.

While all the above-mentioned researchers have been working towards improvements in the accuracy of SNNs, they all have only tried improving the previous approaches by redistributing connections and associated weights. However, while there is quite some research on the impact of dense ANNs [21],[16],[23],[7],[10], little or no research has been conducted on the impact of the activation function choice on SNNs' performance, most probably because this research direction is still new and under development. The approach suggested in this paper is mainly based on SET but can be also applied to other works, which could provide insightful overview on the differences between when many activation functions are taken into account.

4. EVALUATION FRAMEWORK

In this research, a framework for comparing performance and the sparsity effect between activation functions is proposed. Although the explained example is based on the SET algorithm implementation [18], we believe that, with some adjustments, it can also be used for other sparse training methods.
First of all, all experiments used the same hyperparameters stated in Table 1, ensuring the performance changes depend solely on the adjusted parameters - epsilon and the activation function. Although they can be adjusted, one should make sure to keep them consistent across all experiments, as they might impact the outcomes significantly.

Hyperparameter	Value
Learning rate	0.01
Optimiser	SGD
Momentum	0.9
ζ (zeta)	0.3
Batch size	100
Loss Function	Categorical Cross-entropy
Dropout rate	0.3

Table 1: *Hyperparameters used in training of MLP [8].*

Moreover, the SET algorithm implementation needs to be adjusted so that the accuracy and loss function values are saved for both the training and validation sets after each of the training epochs. Then, models are trained for all combinations of selected activation functions and epsilon values corresponding to the desired sparsity levels. Sparsity means simply the probability of a connection between neurons to be removed and thus, it is the opposite of density. Note that this definition is closely related to SET since it removes weights randomly and thus other training techniques may implement sparsity differently.

$$Sparsity = 1 - density \tag{1}$$

Since sparsity is dependent both on the epsilon value and the architecture of the algorithm , we can derive the equation for epsilon ϵ from equation (1) of SET [18] as

$$\epsilon = (1 - sparsity) \times (\frac{n_l \times n_{l+1}}{n_l + n_{l+1}}) \tag{2}$$

where n_l and n_{l+1} mean the dimensions of consecutive layers. This allows us to easily calculate epsilon for all needed sparsity levels. Table 2 shows number of incoming connections per layer, depending on the sparsity of the network. Algorithm 1 describes steps needed to optimize the evaluation procedure. Saving results in clearly declared directories allows for automatic chart generation for all activation functions and sparsity levels.

Sparsity	Epsilon	Parameters [#]		
		1st layer	2nd layer	3rd layer
98.85%	10	141312	46000	46000
97.12%	20	353895	115200	115200
94.25%	50	706560	230000	230000
88.50%	100	1413120	460000	460000
71.20%	500	3538944	1152000	1152000
dense	n.d.	12288000	4000000	4000000

Table 2: *Number of parameters between layers per sparsity*

Algorithm 1 Proposed Sparsity Sweep evaluation framework

```
set hyperparameters
adjust parameters to be saved
for each activation function (AF) do
    set activation function
    set output location for results
    for each sparsity level do
        set ϵ value using equation 2
        train model
    end for
end for
produce performance charts
produce sparsity sweep charts
produce overfitting charts
```

4.1 Overfitting function

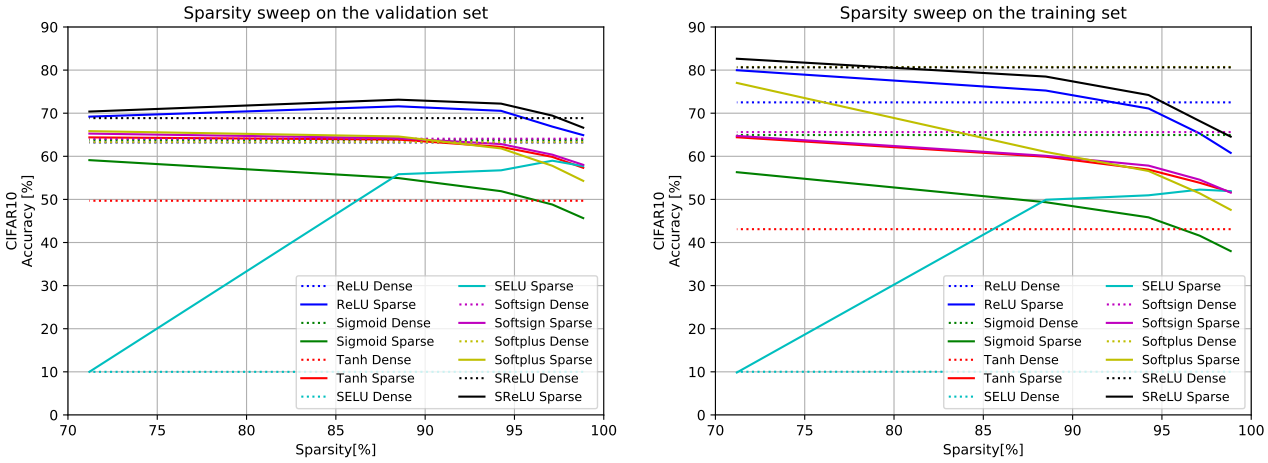
Overfitting occurs when a network is trained in too much detail, trying to fit the noise in the data into their model. Most of the time, scientists try to analyze this effect by comparing training and validation set accuracy scores over the training period. However, since several activation functions are compared in this study, a new function displaying overfitting as the difference between the scores is introduced:

$$o = acc_t - acc_v \tag{3}$$

where o, acc_t and acc_v represent overfitting, training set accuracy and validation set accuracy, respectively. Overfitting values (Eq. 3) above 0 suggest that the network is overfitting, while negative values might mean the opposite - underfitting. Please note that in the beginning of training, the network can show significantly unstable results and thus we should focus on the long-term trend of the function rather than its details.

5. RESULTS

To gain deeper insights, researchers often need to take a look at their results from many perspectives. Therefore, a number of plots have been prepared and analysed but in order to convey the most important message, only a handful is shown below. A curious reader can find other charts in the appendix. In this section, the implementation details as well as the outcomes of the study will be discussed.



(a) *Validation set*

(b) *Training set*

Figure 4: *Accuracy sparsity sweep*

5.1 Implementation and architecture

Within this research, the framework suggested above has been implemented using the adjusted SET-MLP-Keras-Weights-Mask project developed as a part of and presented in [18]. Separate scripts included in the project have been used to train the sparse and dense networks. The project uses Keras for model training and Python packages NumPy and Matplotlib to visualize the results. It is important to note that while the results observed with this implementation resemble the behavior of truly sparse networks, at the moment (June, 2020) Keras does not support SNNs and thus the SET implementation used in this research masks weights to 0 to achieve the same structure. The experiments have been conducted on the CIFAR10 dataset, which consists of 60000 images split into 10 classes, 6000 images each. As the dimensions of the images are 32x32[px] and represented in 3 channel colors, the input layer of the model has n_0 number of input parameters as follows:

$$n_0 = 32 \times 32 \times 3 = 3072 \tag{4}$$

The hidden layers have been set in the research done by Mocanu et al. [18] to 3 layers, 4000, 1000 and 4000 neurons each, respectively. Lastly, the output layer contains 10 neurons, corresponding to 10 classes in the CIFAR10 dataset. All models were trained for 500 epochs and used the same hyperparameters but the epsilon value and activation function.

5.2 Sparsity Sweep

For each activation function, a model has been trained using 6 different sparsity levels, from 98.85% (only 1.15% connections left) to the dense level. Scores for the accuracy as well as the loss function value were collected after each training epoch for both training and validation sets. Table 3 shows the final accuracy results for each of the models after 500 epochs which are visualized in the Figure 4a. It is clear to note that SReLU has performed best overall, while ReLU performed almost as well, with only a slight difference in accuracy on all sparsity levels. Networks using Softsign, Tanh and Softplus scored slightly worse and more importantly, their performance decreased with increased sparsity, meaning their optimal sparsity is much lower than for ReLU and SReLU. The case of SELU is significantly standing out because at the dense levels, the network could not predict anything at more than the random chance (10%) after 500 epochs of training, while very sparse networks trained with SELU performed better than those with Sigmoid or Softplus and just as well as Tanh and Softsign.

Activation	Accuracy [%] on the validation set at sparsity level:					
	98.85%	97.12%	94.25%	88.50%	71.20%	dense
ReLU	64.91	66.89	70.55	**71.6**	69.2	63.21
Sigmoid	45.66	48.81	51.91	54.95	59.11	**63.8**
Tanh	57.33	59.86	62.19	63.88	**64.39**	49.69
Softplus	54.3	57.8	61.83	64.6	**65.84**	63.27
Softsign	58.02	60.38	62.84	64.19	**65.27**	64.08
SELU	57.72	**58.98**	56.75	55.84	10	10
SReLU	66.64	69.44	72.22	**73.14**	70.38	68.86

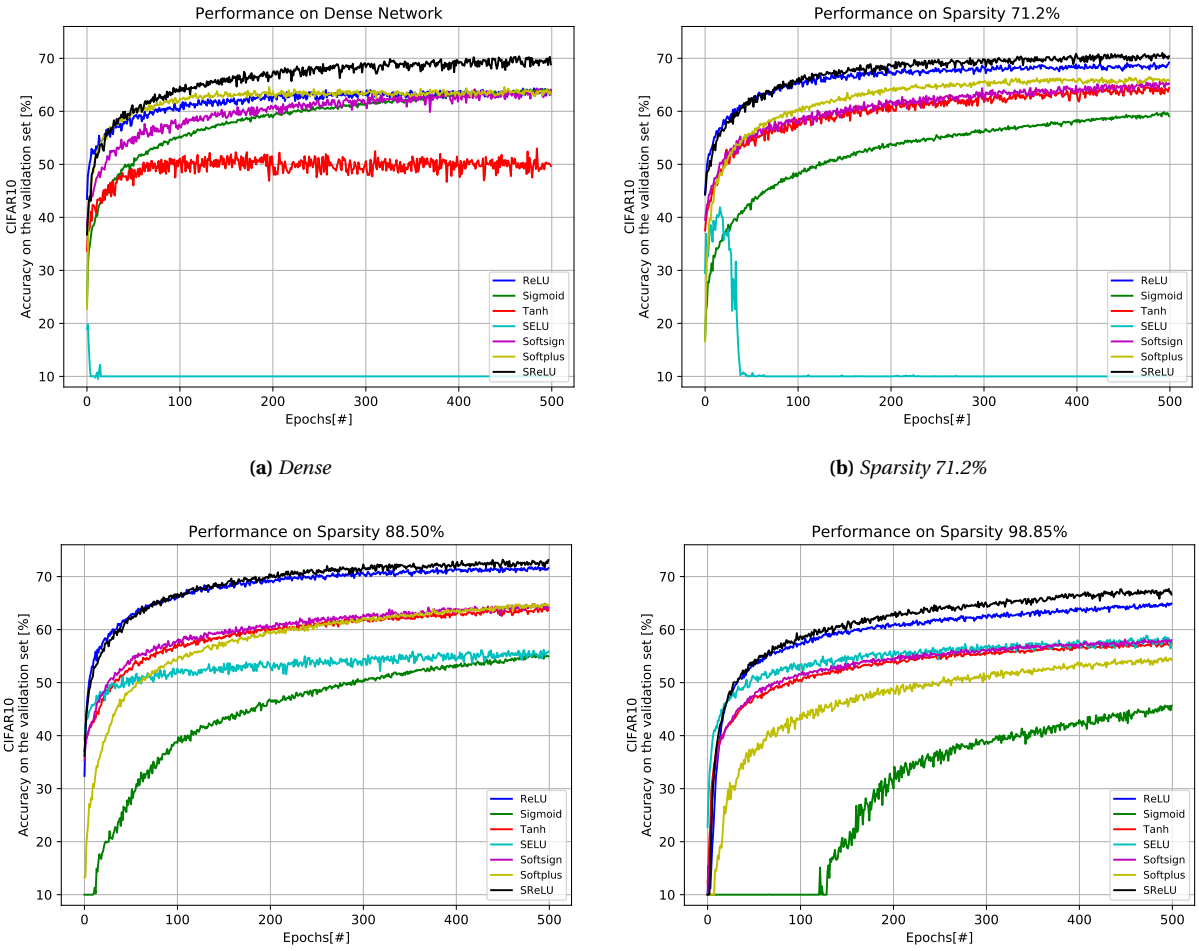
Table 3: *Accuracy [%] on the validation set after 500 epochs*

5.3 General Performance

When it comes to the training performance, it is best to look at the performance charts. Figure 5 shows four charts which point out the most notable differences between the activation functions across the four selected sparsity levels.

Clearly, SReLU is the winner when it comes to accuracy, but it does not gain much better performance at optimal sparsity levels. ReLU, on the other hand, gains a lot of accuracy when trained on a sparse network, while it performs only as well as Softplus, Softsign, SELU and Sigmoid in the dense setting. Interestingly, Sigmoid is an example of a function that performs fine on dense network, but did not work well with sparse networks - its learning progress was somewhat delayed proportionally to the sparsity level. SELU is an example of a network that shows the opposite effect - while it performs fairly well on sparse networks, the training dies out after several training epochs. The behavior will be further discussed in the discussion section. When it comes to extremely sparse networks e.g. Sparsity 98.85% at Figure 5d, we can notice that while general accuracy dropped noticeably, the difference in accuracy varies significantly, depending on the activation function. Most noticeable drop in accuracy by the sparsity affected the models using Sigmoid and Softplus, while the model using SELU activation function scored higher than its dense counterparts.

To better understand the reasons behind the differences in performance of the examined activation functions, one needs to look at the problem from many angles. Plots visualizing the data from the function-specific point of view can be found in the appendix section A.



(a) *Dense*

(b) *Sparsity 71.2%*

(c) *Sparsity 88.5%*

(d) *Sparsity 98.85%*

Figure 5: *Comparison of models' performance.*

5.3.1 SELU

Unexpected behavior of SELU is clearly noticeable on the dense levels. The performance chart of SELU in Figure 6, clearly shows that the dense- and the sparsest network (Sparsity 71.2%) experience an accuracy drop to the random level (10%) after a couple of training epochs. Our hypothesis is that the learning rate used in the training was too high. As definite conclusions cannot be gained from just the networks trained in this study, a study on many dense levels is needed to find an explanation of the issue.

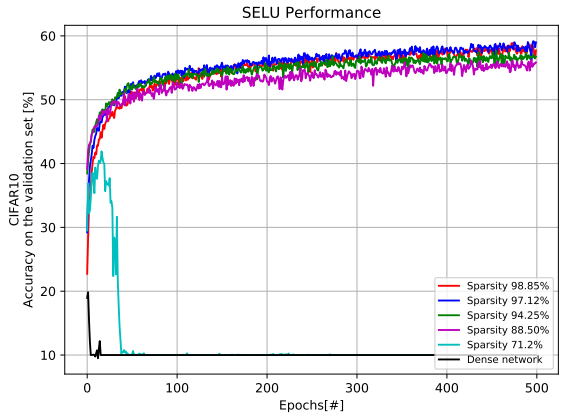
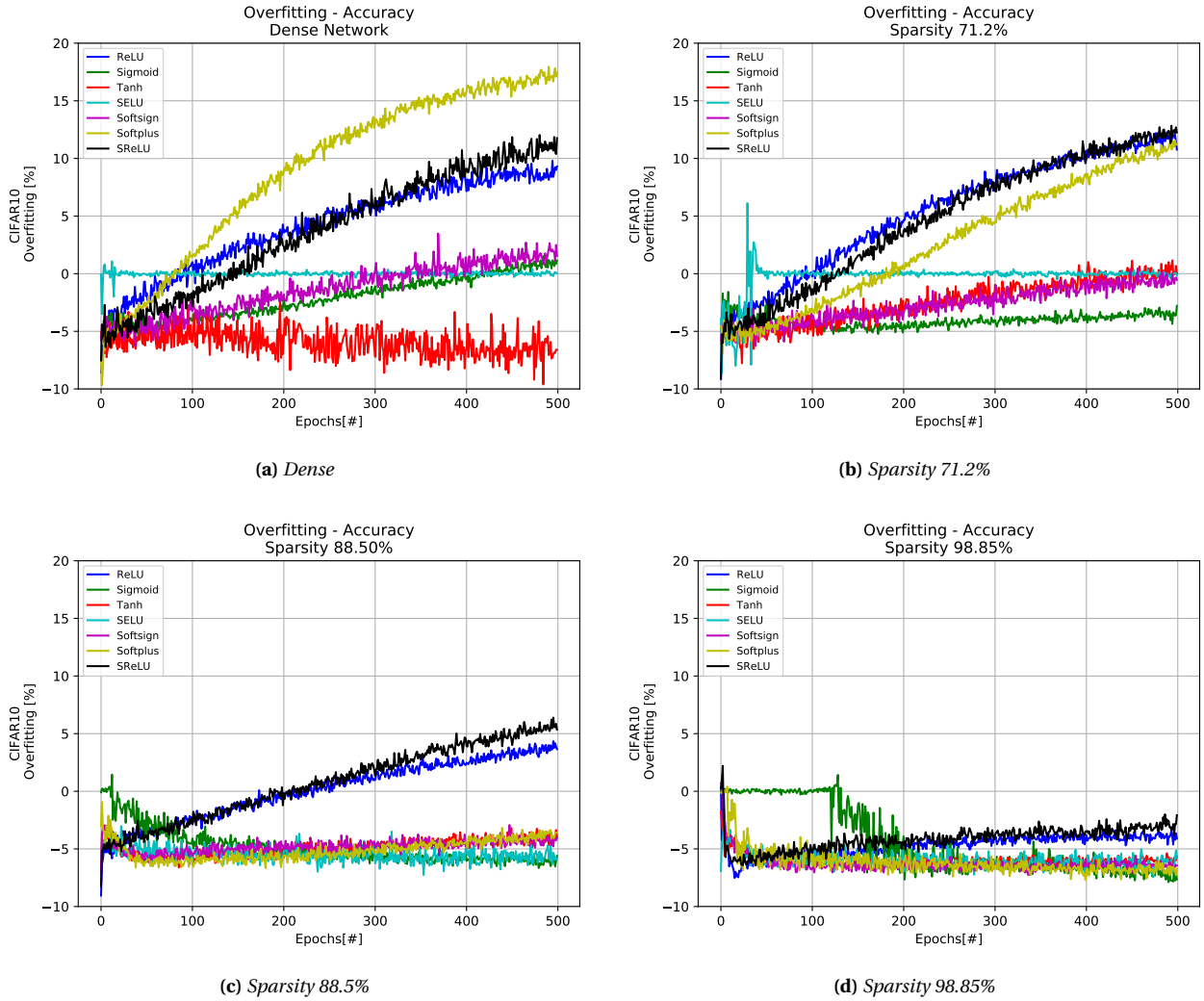


Figure 6: *SELU accuracy on the validation set*

5.4 Overfitting

Charts in Figure 7 show the above-mentioned overfitting function results for four selected sparsity levels. We can notice that for the dense level (Fig. 7a), the difference between functions is significant. Tanh is underfitting, Softplus and Sigmoid fit relatively well, while ReLU, SReLU and Softplus are significantly overfitting. The overfitting effect for SELU is not analysed on dense levels due to the network training issue mentioned before.

Since overfitting is caused by over-trained models, it is reasonable to assume that with increased sparsity the overfitting effect will decrease, which is indeed the case. However, not all functions react in the same way. At sparsity level 71.2% (Fig. 7b), most functions show smaller overfitting effect while the Hyperbolic Tangent (Tanh) seems to achieve a good fit. We can notice that Sigmoid started underfitting, while SReLU and ReLU plots seem to show barely any difference when compared to the dense plots. At sparsity 88.5% (Fig. 7c), the overfitting effect is even less noticeable - most functions are underfitting, while only ReLU and SReLU are slightly overfitting. In case of the extremely sparse networks with sparsity 98.85% (Fig. 7d), all functions were underfitting.



(a) *Dense*

(b) *Sparsity 71.2%*

(c) *Sparsity 88.5%*

(d) *Sparsity 98.85%*

Figure 7: *Overfitting effect per sparsity level.*

6. CONCLUSIONS AND FUTURE WORK

The framework suggested allows for comparisons between activation functions and their effects on the sparsity sweep of SNNs, while the experiments conducted provide insights into the impact of the selected activation functions on the accuracy of models trained with SET. Clearly, SReLU, which was used in Mocanu's implementation of SET [18], performs best at all sparsity levels, but the sparsity does not affect its achieved accuracy significantly. Other functions like ReLU, Tanh, Softplus achieve noticeably higher performance at sparsity levels 71.2% and 88.5%, higher than at the dense levels. While in general, most activation functions perform best at sparsity level 71.2%, ReLU and SReLU performed best at sparsity 88.5% and SELU achieved best performance at the extremely high sparsity level 98.85%. Sigmoid is an exception there, as its performance decreases for sparser networks. Our hypothesis is that it is caused by Sigmoid not being zero-centered. When it comes to under- and overfitting, at higher sparsity levels, most functions start underfitting but they are not affected by the sparsity equally. SReLU and ReLU seem to have highest optimal point of sparsity when they achieve the best training fit, while the overfitting effect of Softplus was most affected by the increased sparsity.

Hopefully, researchers will find this paper useful to get insight and inspirations on the impact of activation functions on the sparsity sweep of SNNs. Naturally, this research does not cover all the questions around this topic and therefore, more research in the area would help answer the questions.

In the future, researchers could work on the following issues:

1. The experiments shall be repeated on datasets other than CIFAR10 to compare the results.
2. More experiments could be done on more dense levels around 50-90%. This would provide more accurate insights for optimal sparsity levels of each of the activation functions.
3. The training issue of SELU needs to be examined with a deeper mathematical analysis on a number of dense levels.
4. More work can be done to better understand the underlying reasons for the differences between the effects of those activation functions.
5. Naturally, other AFs should still be tested to better understand the similarities between groups of AFs

7. ACKNOWLEDGEMENTS

I would like to express great appreciation to Decebal Mocanu for his guidance and useful directions which allowed me to focus on what was most important. Additionally, I would like to thank Selima Curci, who helped me overcome technical obstacles with implementation of the SReLU activation function.

8. REFERENCES

[1] G. Bellec, D. Kappel, W. Maass, and R. Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

[2] A. D. Bird and H. Cuntz. Dendritic normalisation improves learning in sparsely connected artificial neural networks. *bioRxiv*, 2020.

[3] X. Dai, H. Yin, and N. K. Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

[4] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[5] T. Dettmers and L. Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[6] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. *arXiv preprint arXiv:1911.11134*, 2019.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[9] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[10] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. Deep learning with s-shaped rectified linear activation units. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[11] V. Lapshyna. Sparse artificial neural networks: Adaptive performance-based connectivity inspired by human-brain processes. B.S. thesis, University of Twente, 2020.

[12] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[13] D. Liu. A practical guide to relu, Nov 2017.

[14] S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, and M. Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *arXiv preprint arXiv:1901.09181*, 2019.

[15] R. Maksutov. Deep study of a not very deep neural network. part 2: Activation functions, May 2018.

[16] D. Mishkin and J. Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[17] D. C. Mocanu. *Network computations in artificial intelligence*. PhD thesis, Technische Universiteit Eindhoven, 2017.

[18] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[19] H. Mostafa and X. Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. *arXiv preprint arXiv:1902.05967*, 2019.

[20] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

[21] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[22] I. Ohn and Y. Kim. Smooth function approximation by deep neural networks with general activation functions. *Entropy*, 21(7):627, 2019.

[23] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

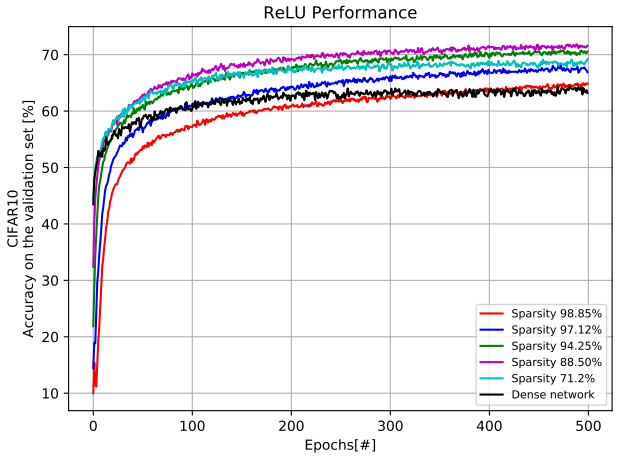
[24] H. Ramchoun, M. A. J. Idrissi, Y. Ghanou, and M. Ettaouil. Multilayer perceptron: Architecture optimization and training. *IJIMAI*, 4(1):26–30, 2016.

[25] S. Saxena. Underfitting vs. overfitting (vs. best fitting) in machine learning, Feb 2020.

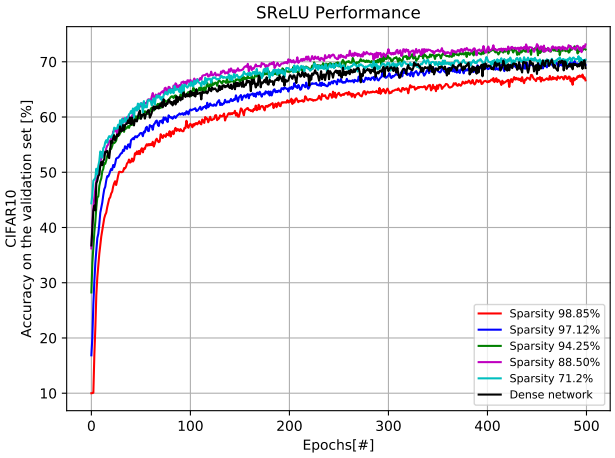
[26] A. Sharma. Understanding activation functions in neural networks. *Medium. com blog*, 30, 2017.

APPENDIX

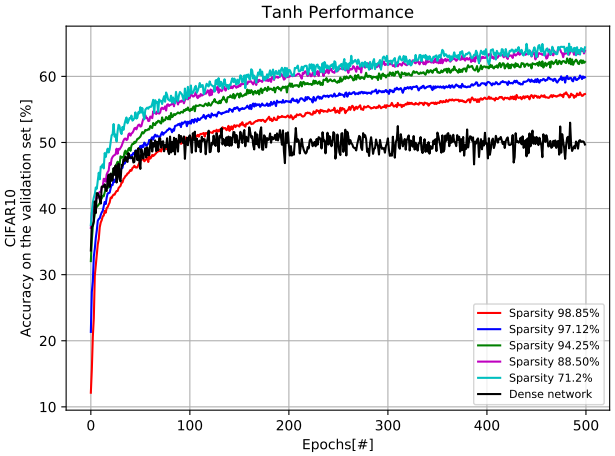
A. PERFORMANCE



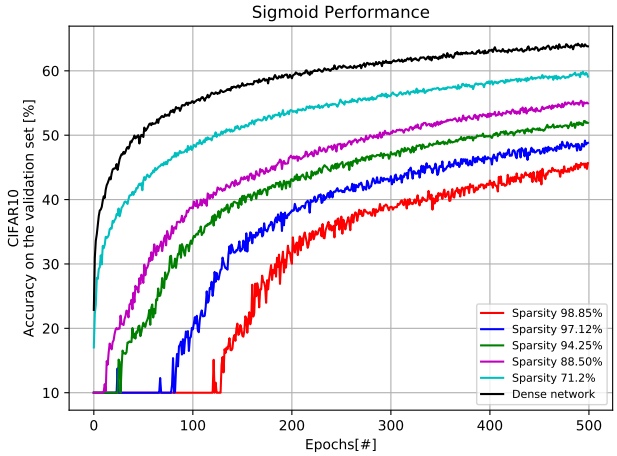
(a) *ReLU*



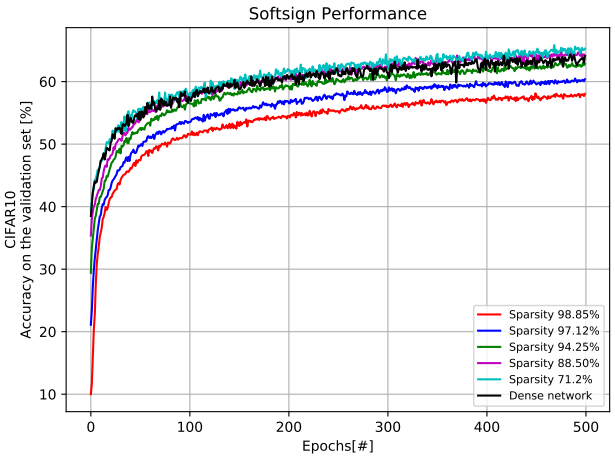
(b) *SReLU*



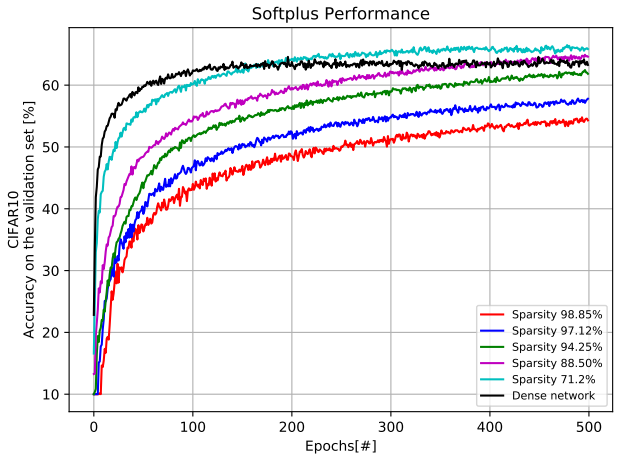
(c) *Tanh*



(d) *Sigmoid*



(e) *Softsign*



(f) *Softplus*

Figure 8: *Accuracy on the validation set per function.*