On the Robustness of Activation Functions

Zachary Harrison
Computational Mathematics and Computer Science
Logan, Utah
Zachary.Mark.Harrison@gmail.com

ABSTRACT
This paper proposes a method for testing the robustness of different activation functions in neural networks in addition to a method of creating an infinite multivariate dataset for regression tasks. The method involves training static model architectures with interchangeable activation functions on clean and dirty versions of two datasets (multivariate and image), and comparing the models' validation performance on the clean and dirty datasets. The paper compares the performance of seven different activation functions: ReLU, Leaky ReLU, PReLU, ELU, Sigmoid, Tanh, and Softsign. The results show that ReLU and ReLU-like activation functions are more robust than others, performing similarly on both clean and dirty datasets. This research has important implications for improving the reliability and robustness of neural networks.

1. INTRODUCTION
Deep neural networks have been shown to be capable of anything from image recognition to natural language processing. And yet deep neural networks often have problems where they incorrectly classify data because of patterns it learned due to noisy input data. So, people responsible for these networks often clean input data to reduce the inherent noise all input has and therefore reduce incorrect classification.

The more robust a neural network is, the less it is prone to the vulnerabilities discussed above. Neural networks are only able to learn complex patterns by using activation functions to introduce nonlinearity into the network. So, increasing the robustness of an activation function or using a more robust activation function will likely produce a more robust neural network. Rather than trying to increase the robustness of activation functions, this paper proposes a method for testing the robustness of different activation functions, thereby enabling future data scientists to perform similar tests and make more informed decisions about which activation function to use for a particular type of dataset.

The rest of this paper is organized as follows: Section 2 provides a brief overview of related work in the creation of infinite datasets. Section 3 describes the proposed method in detail, including the experimental setup and evaluation metrics. Section 4 describes the creation of an infinite multivariate dataset called the FBM-Dataset, along with its experimental results. Section 5 is similar to Section 4, with the important change being the dataset. Finally, Section 6 discusses the implications of our findings and Section 7 discusses potential directions for future research.

If you want more information about how I implemented these models, take a look at my GitHub[4], which documents all my code.

2. RELATED WORK
While conducting research for a suitable dataset to use in a multivariate regression task, I came across the paper "Understanding the Difficulty of Training Deep Feedforward Neural Networks"[3], which introduced an infinite dataset called Shapeset 3×2 . Although I was unable to find an API for this dataset for the Image Classification task I discuss in section 4, I found the idea to be brilliant. The concept of Shapeset 3×2 inspired me to create my own dataset, as I was unable to find a suitably clean multivariate dataset.

3. PROPOSED METHOD
Here is the proposed method by which I am testing the robustness of different activation functions:

1. For each dataset type (multivariate and image in this case), define a static model architecture with an interchangeable activation function. Multiple instances of this model will be created throughout the training process.
2. For each dataset type, create a clean and dirty version. Note that while these datasets have different training set, they share a validation set. Since the model does not learn the validation set, it can remain clean so as to better compare performance.
3. Train each model on the **clean** training dataset, recording performance on the validation dataset.
4. Train each model on the **dirty** training dataset, recording performance on the validation dataset.
5. Compare each models' validation performance on the clean vs dirty datasets, observing that the most robust activation functions will perform roughly equally across both datasets. That is, a robust activation function will update weights less when given input that is an outlier.

Additionally, here is the list of activation functions (in the order I tested them) for which I compared performance of:

1. ReLU

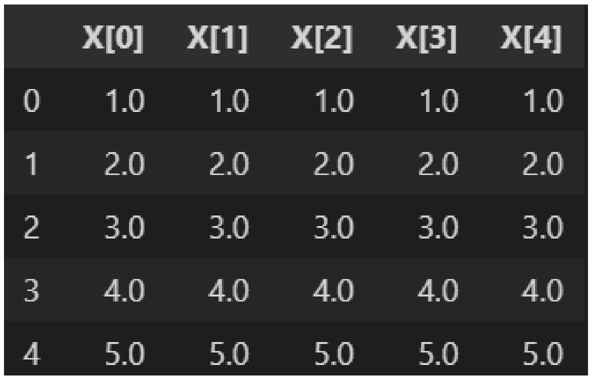
2. Leaky ReLU
3. PReLU
4. ELU
5. Sigmoid
6. Tanh
7. Softsign

4. MULTIVARIATE DATA

4.1 Novel Idea: Create the FBM-Dataset

FBM-Dataset is short for Function-Based Multivariate Dataset. I provide an outline for how to create this infinite dataset, which followed 5 steps:

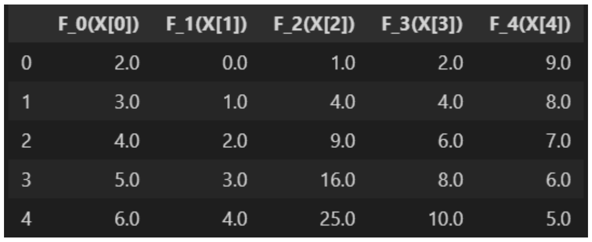
1. Select a domain to populate each column.



	X[0]	X[1]	X[2]	X[3]	X[4]
0	1.0	1.0	1.0	1.0	1.0
1	2.0	2.0	2.0	2.0	2.0
2	3.0	3.0	3.0	3.0	3.0
3	4.0	4.0	4.0	4.0	4.0
4	5.0	5.0	5.0	5.0	5.0

Figure 1. The example FBM-Dataset after step 1. As you can see, each column has the same domain, which is the list of integers from 1 to 5.

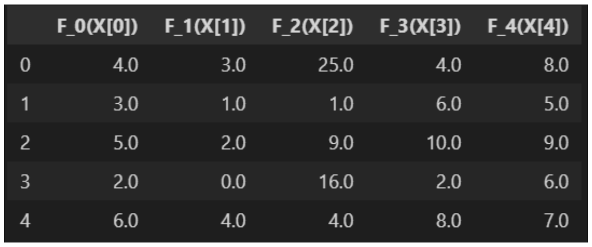
2. For each column, apply a random function to the X-values.



	F_0(X[0])	F_1(X[1])	F_2(X[2])	F_3(X[3])	F_4(X[4])
0	2.0	0.0	1.0	2.0	9.0
1	3.0	1.0	4.0	4.0	8.0
2	4.0	2.0	9.0	6.0	7.0
3	5.0	3.0	16.0	8.0	6.0
4	6.0	4.0	25.0	10.0	5.0

Figure 2. The example FBM-Dataset after step 2. Note that all of the functions have been made easily identifiable by a human, like how $f_0 = x+1$, $f_1 = x-1$. These are examples of functions to use, but they should be selected carefully so as to produce as unique features as possible.

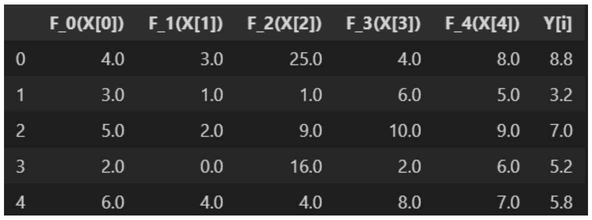
3. Shuffle each column independently.



	F_0(X[0])	F_1(X[1])	F_2(X[2])	F_3(X[3])	F_4(X[4])
0	4.0	3.0	25.0	4.0	8.0
1	3.0	1.0	1.0	6.0	5.0
2	5.0	2.0	9.0	10.0	9.0
3	2.0	0.0	16.0	2.0	6.0
4	6.0	4.0	4.0	8.0	7.0

Figure 3. The example FBM-Dataset after step 3. Note that each column was shuffled, not each row. Ex: look at row 0 = [2, 0, 1, 2, 9] in step 2 and observe how it is not present in this figure. This is to ensure that a model learns each columns' function, rather than pattern about the domain.

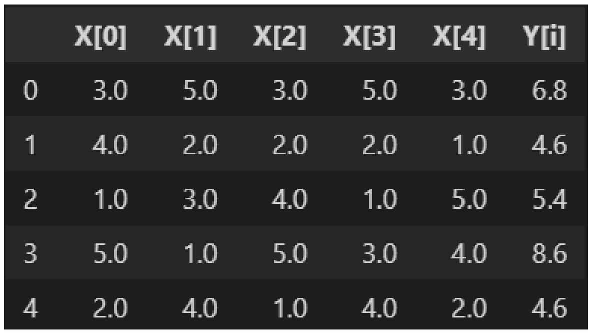
4. Let y equal the mean of each row.



	F_0(X[0])	F_1(X[1])	F_2(X[2])	F_3(X[3])	F_4(X[4])	Y[i]
0	4.0	3.0	25.0	4.0	8.0	8.8
1	3.0	1.0	1.0	6.0	5.0	3.2
2	5.0	2.0	9.0	10.0	9.0	7.0
3	2.0	0.0	16.0	2.0	6.0	5.2
4	6.0	4.0	4.0	8.0	7.0	5.8

Figure 4. The example FBM-Dataset after step 4. Ex: the mean of row 0 = $(4+3+25+4+8)/5 = 44/5 = 8.8$.

5. Hide the functions.



	X[0]	X[1]	X[2]	X[3]	X[4]	Y[i]
0	3.0	5.0	3.0	5.0	3.0	6.8
1	4.0	2.0	2.0	2.0	1.0	4.6
2	1.0	3.0	4.0	1.0	5.0	5.4
3	5.0	1.0	5.0	3.0	4.0	8.6
4	2.0	4.0	1.0	4.0	2.0	4.6

Figure 5. The example FBM-Dataset after step 5. We hide the functions from the network in order to introduce nonlinearity. At this point, there is zero noise in the data, which should enable any network to train with high performance.

This should allow for any model to exactly find the y-values because there is a definite function by which the y-value is derived from the input.

Note that steps (2) and (3) would normally be reversed in implementation so that the shuffled X-values could be stored. However, I felt that this ordering made it easier to understand the process.

4.2 Creating the Dirty FBM-Dataset

Because the FBM-dataset is as clean as can be and we need both a clean and dirty dataset, we need some method by which to add noise to this dataset to create a dirty FBM-dataset. This process is accomplished by adding gaussian noise to a certain proportion of the data. In this paper, I chose to add noise to only 10% of the data. To better visualize what was happening to the dataset and verify that I was making a valid portion of the dataset dirty, I created figures for each function. Here are some of the figures:

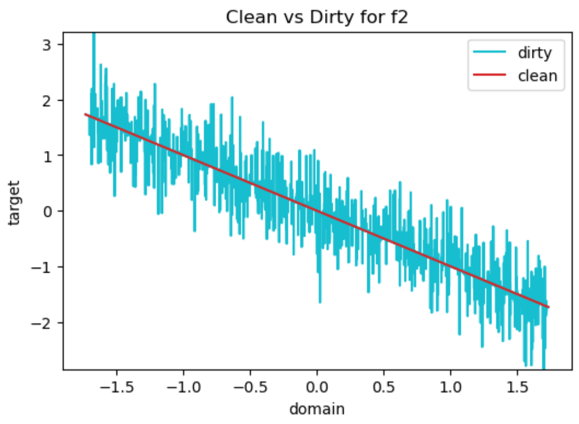


Figure 6. $f_2(X) = \textbf{RAND} \cdot x + \textbf{RAND}$**, where RAND is a random float in** $[-1, 1)$

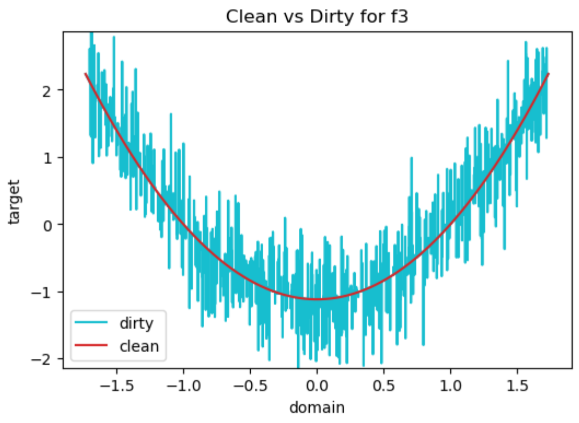


Figure 7. $f_3(X) = \textbf{RAND} \cdot x^2 + \textbf{RAND} \cdot x + \textbf{RAND}$

It is noteworthy to highlight the introduction of additional variability to the dataset by means of random function generation. Specifically, the current implementation includes a list of static functions in which each term is multiplied by a random value. While a more comprehensive approach would involve the generation of more complex and varied functions, such a solution has yet to be formulated.

4.3 Experiments

4.3.1 Model Architecture

Once again, this is a static model used repeatedly throughout the experiments, where only the activation function was changed.

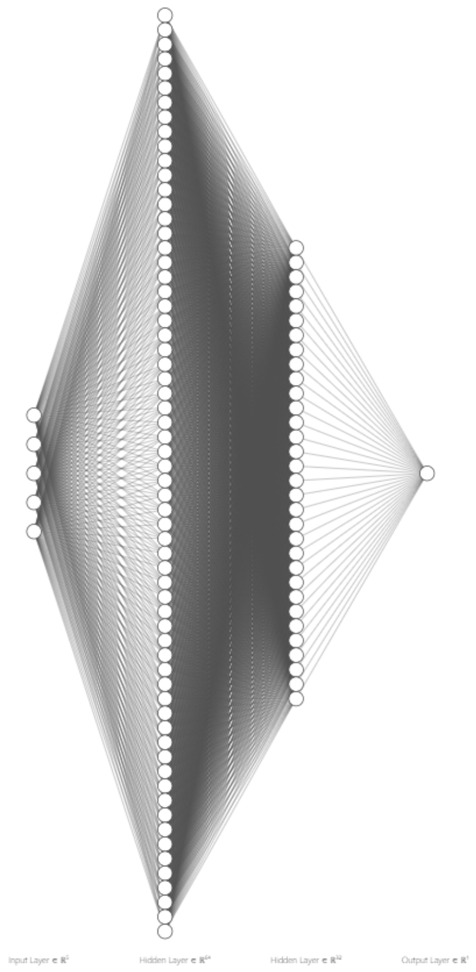


Figure 8. Due to the size of the layers, each node represents two nodes. That is, the ordered layer sizes are: [10, 128, 64, 1] with MSE as the evaluation metric

4.3.2 Hyperparameters

- Batch size: 64
- Epochs: 20
- Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate=0.001, weight decay=$1e-10$)
- Train/val/test split:
  - Train: 5000 (71%)
  - Validation: 2000 (29%)
  - Not used

Note: Since results should consistent and fair across different dataset types and I did not use a test dataset for the Image dataset type, there is no test set for this dataset.

4.3.3 Baseline

There is no Baseline for this dataset because I invented it. However, each pair of clean and dirty models are compared, using the clean model as a baseline.

4.3.4 Evaluation Metrics

The evaluation metric for each individual model is MSE, as this is a regression task.

The evaluation metric for each activation function is the difference in val MSE between the clean and dirty datasets.

4.4 Results

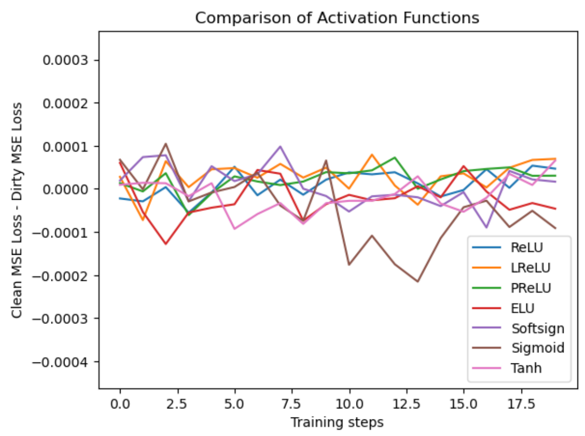


Figure 9. This figure compares the relative efficacy of a list of activation functions based on their models' relative performance on a clean vs dirty dataset.

Robust activation functions should result in a model with the same relative performance on the clean and dirty datasets. Looking at the figure, we observe that ReLU-like and Softsign perform slightly better than Sigmoid and Tanh.

While these results might indicate that the models using Sigmoid and Tanh are not far behind the rest, this is not true because the MSE for the clean dataset is lagging behind. Due to limited space, there is no figure depicting this.

5. IMAGE DATA

Since being easy to train on was my only criteria for a clean dataset, MNIST was an obvious choice.

5.1 Experiments

5.1.1 Data Preprocessing

In order to reduce extra regularization, little data preprocessing was done. Here are the two preprocessing techniques used:

- Resizing 28x28 grayscale images to 64x64 grayscale images.
- Normalizing each pixel to a mean of 0 and a standard deviation of 1.

As for creating the Dirty dataset, I took 10% of the samples and replaced their label with something else.

5.1.2 Model Architecture

Once again, this is a static model used repeatedly throughout the experiments, where only the activation function was changed.

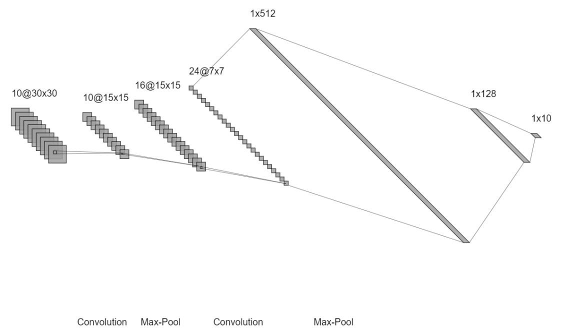


Figure 10. The static CNN model architecture for the MNIST dataset. Again, the activation function is dynamic and can be changed.

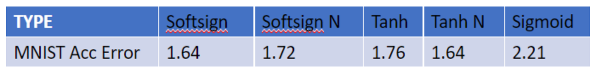
5.1.3 Hyperparameters

- Batch size: 128
- Epochs: 25
- Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate=0.001, weight decay=0 or $1e-5$)
- Dropout (p = 0 or 0.25)
- Train/val/test split:
  - Train: 31,500 (45%)
  - Validation: 10,500 (15%)
  - (not used) Test: 28,000 (40%)

I did not end up using the test set because a fair way to choose the best model to test it on could not be decided upon.

5.1.4 Baseline

The MNIST dataset is famous for being easy to train on. In fact, accuracy of over 99% is commonly observed. However, I thought that it might be reasonable to compare it to the performance in "Understanding the Difficulty of Training Deep Feedforward Neural Networks"[3] (see figure 9) because it is the paper that sparked this idea.



TYPE	Softsign	Softsign N	Tanh	Tanh N	Sigmoid
MNIST Acc Error	1.64	1.72	1.76	1.64	2.21

Figure 11. Xavier-Normalization is presented in this paper, so the N stands for Normalized.

5.1.5 Evaluation Metrics

The evaluation metric for each individual model is Cross Entropy Loss, as this is a classification task.

The evaluation metric for each activation function is the difference in val accuracy between the clean and dirty datasets.

5.2 Results

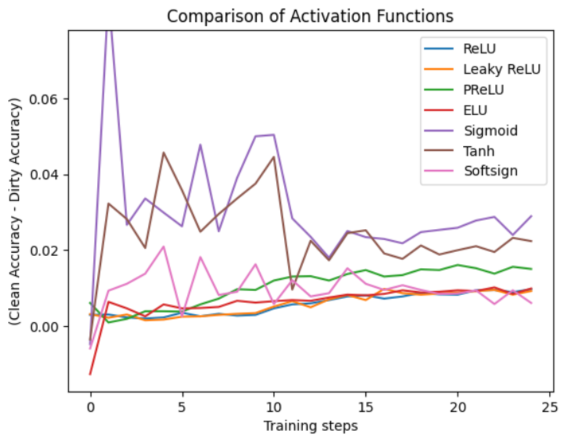


Figure 12. These are the results for the models without regularization

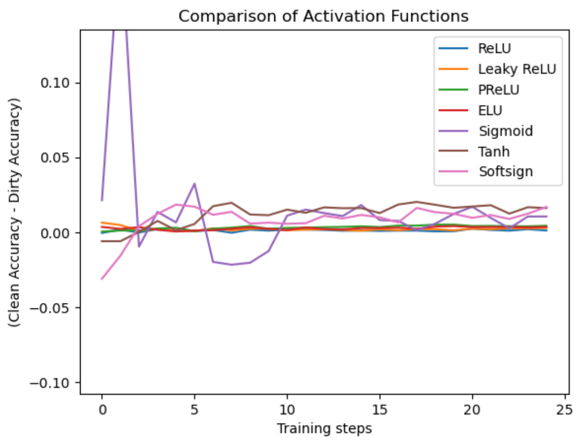


Figure 13. These are the results for the models with regularization. Specifically, there is weight decay with $1e-10$ **and two dropout layers before the linear layers with** $p = 0.25$

As you can see, ReLU-like activation functions stay much closer to $y = 0$ and do so consistently. While Softsign also performs relatively well, it is safe to say that Sigmoid and Tanh perform significantly worse than the other activation functions. As for why some models perform better on the dirty dataset (as indicated by a negative value), it is unclear at this time.

6. CONCLUSION

When I started making this paper, I believed that Sigmoid and Tanh would be good activation functions because I thought that outliers would be ignored. That is, I considered each feature of a dataset to be like a bell curve, with the edges of the bell curve being the outliers. And if were an accurate depiction of features, then perhaps Sigmoid and Tanh would perform better. However, most "outliers" are simply datapoints that have received an unnatural amount of noise and therefore have an incorrect label.

Additionally, according to "Small nonlinearities in activation functions create bad local minima in neural networks" [9]

My last idea as to why Sigmoid and similar activation functions perform poorly is that they create manifolds with shallow curves. This was something I realized after Christopher Olah's gentle introduction to manifolds[7].

Overall, I believe my work to be sufficient in concluding that Sigmoid and Tanh are inferior activation functions for the majority of datasets.

7. FUTURE WORK

It would be very interesting to try to visualize the manifolds created by different activation functions and compare the steepness of the curves. This would allow for a more intuitive comparison of activation functions in terms of how quickly they allow for training.

Additionally, I would love to create my own activation function, but I think making one using well-informed intelligence decisions is extremely difficult. Unfortunately, it's not something an undergraduate student is going to solve in a semester. But this doesn't mean I can't try to make one in the future!

8. REFERENCES

[1] Bhaskar DasGupta and Georg Schnitger. 1992. The Power of Approximating: a Comparison of Activation Functions. In *Advances in Neural Information Processing Systems*, S. Hanson, J. Cowan, and C. Giles (Eds.), Vol. 5. Morgan-Kaufmann. `https://proceedings.neurips.cc/paper_files/paper/1992/file/e555ebe0ce426f7f9b2bef0706315e0c-Paper.pdf`

[2] Bin Ding, Huimin Qian, and Jun Zhou. 2018. Activation functions and their characteristics in deep neural networks. In *2018 Chinese Control And Decision Conference (CCDC)*. 1836–1841. DOI: `http://dx.doi.org/10.1109/CCDC.2018.8407425`

[3] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Yee Whye Teh and Mike Titterington (Eds.), Vol. 9. PMLR, Chia Laguna Resort, Sardinia, Italy, 249–256. `https://proceedings.mlr.press/v9/glorot10a.html`

[4] Zachary Harrison. 2022. cs5890-ADL-Project. `https://github.com/Zachary-Harrison/cs5890-ADL-project`.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. `https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf`

[6] H. N. Mhaskar and C. A.. Micchelli. 1993. How to Choose an Activation Function. In *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro,

and J. Alspector (Eds.), Vol. 6. Morgan-Kaufmann. https://proceedings.neurips.cc/paper_files/paper/1993/file/51ef186e18dc00c2d31982567235c559-Paper.pdf

[7] Christopher Olah. Neural Networks, Manifolds, and Topology. DOI:http://dx.doi.org/posts/2014-03-NN-Manifolds-Topology/

[8] Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word Representations: A Simple and General Method for Semi-Supervised Learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL '10)*. Association for Computational Linguistics, USA, 384–394.

[9] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. 2019. Small nonlinearities in activation functions create bad local minima in neural networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rke_YiRct7