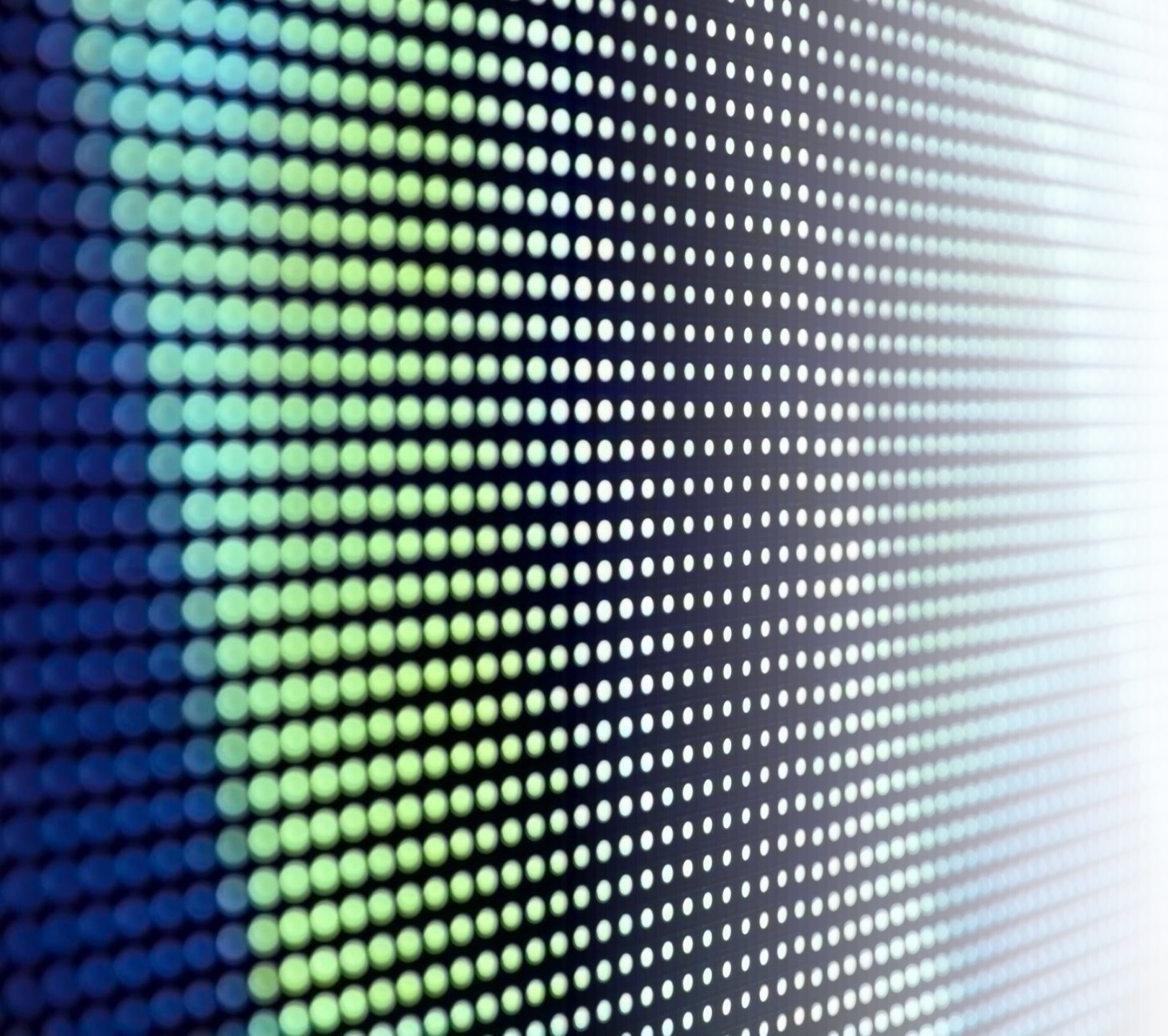
On the Robustness of Activation Functions

By Zachary Harrison

Introduction

- Hypothesis: Sigmoid, Tanh, and ELU (though less so) are activation functions that are better equipped to allow a model to ignore outliers in their inputs, thus producing a superior model.
- Experiment: Use extremely **clean** datasets that allow for models with high accuracy and modify them to create **dirty** datasets.
  - Use the same val dataset, because they do not affect weights.
  - A model that uses a robust activation function should perform well on the dirty dataset because it allows the model to ignore the incorrect datapoints.
- I am using PyTorch for training, rather than TensorFlow.



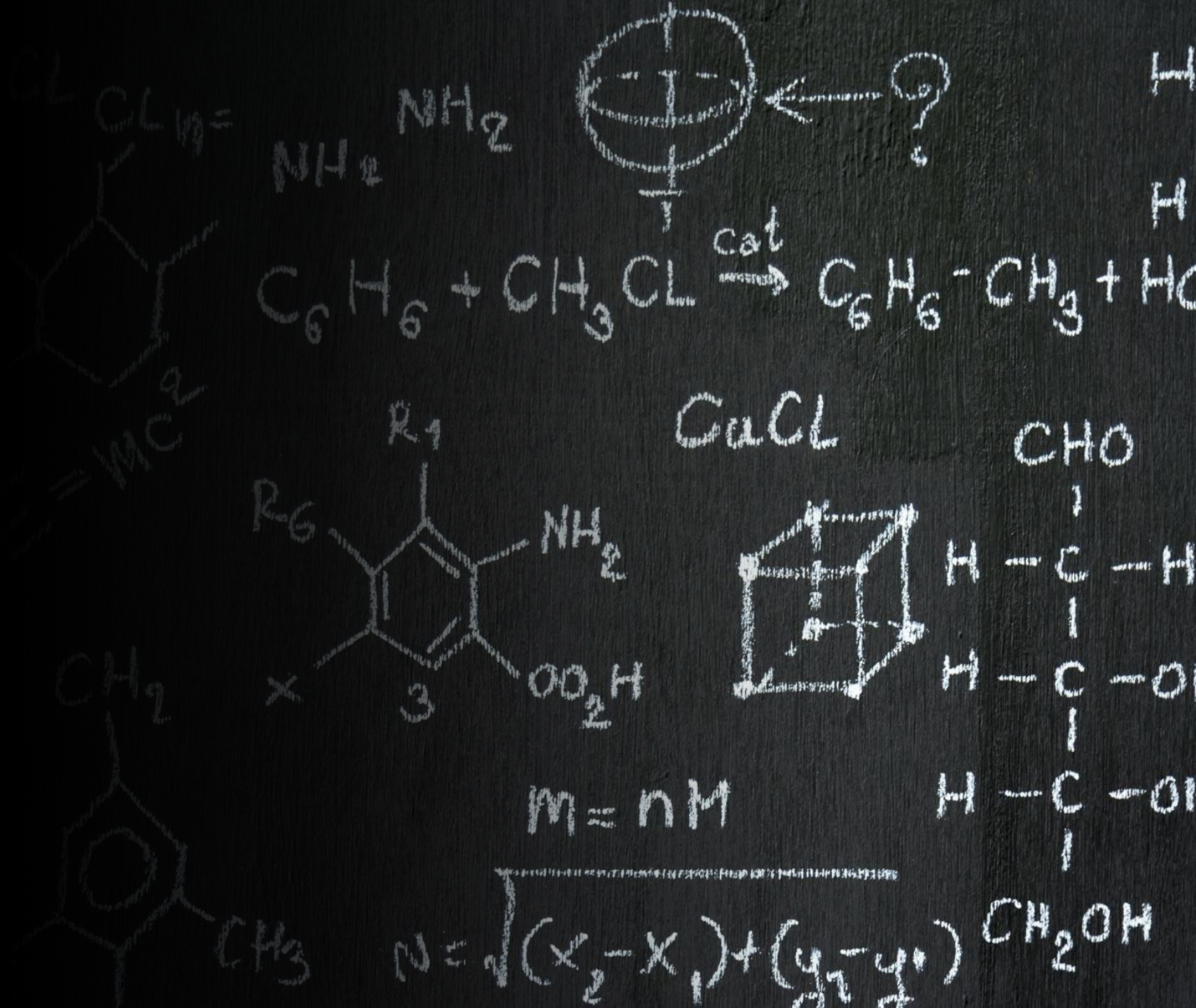
Multivariate Data

Novel Idea: Invent an Infinite Multivariate Dataset

- I got this idea from the research paper Understanding the difficulty of training deep feedforward neural networks, where they created an infinite dataset for image classification.
- Making your own data for an infinite dataset is:
  - Useful because large datasets are essential to producing accurate models.
  - Very cool because you feel like a master of reality
- Unfortunately, I couldn't find any sort of API for Shapeset-3x2 Dataset and making an image dataset sounded like a fate worse than death. So, I made an infinite dataset for multivariate data, instead!

The Axiom of Machine Learning

- All models are based on the idea that for a set of input X and output Y, there exists some function F such that F(X[i]) approximates Y[i] for all values of i.
- Basically, there is some underlying structure whereby each row of X produces its corresponding y value.



Creating the Infinite Multivariate Dataset

Step 1: Select a domain to populate each column (Ex: the integers from 1-5)

	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

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

Note: these are not the functions I used

	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

Creating the Infinite Multivariate Dataset

Step 3: Shuffle each column independently

This is different from shuffling rows.

	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

Note: This is so that the models are forced to learn each column's individual function, rather than some pattern about the domain

Step 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

Creating the Infinite Multivariate Dataset

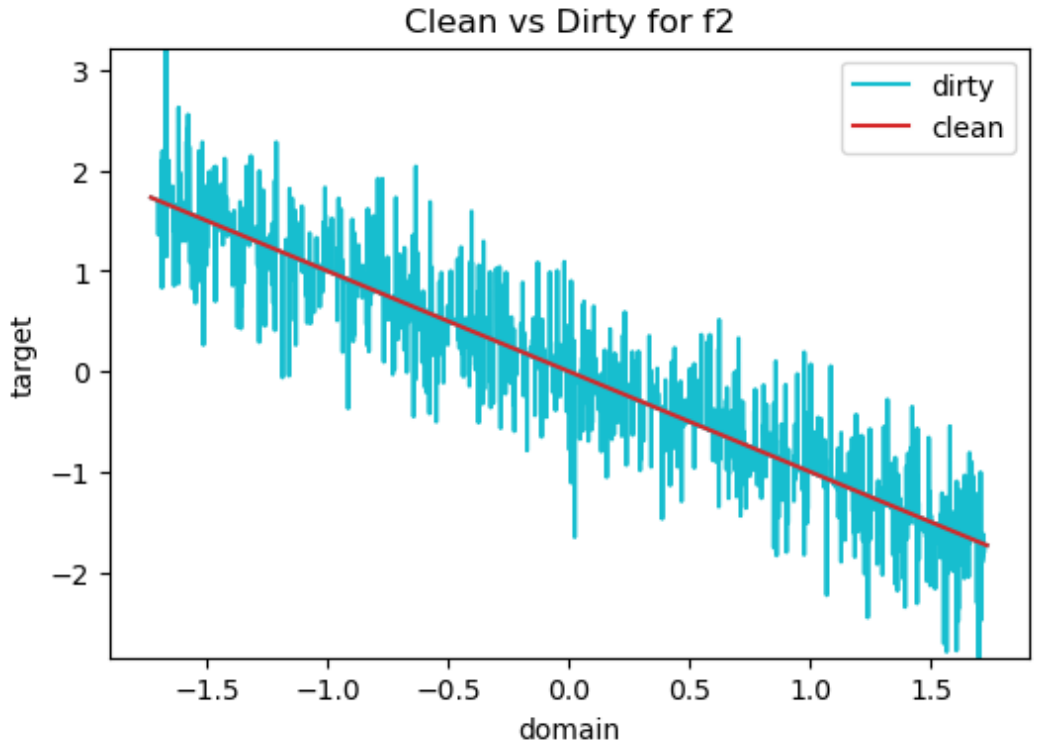
Step 5: Hide the functions

- 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.
- This dataset does not have noise... yet

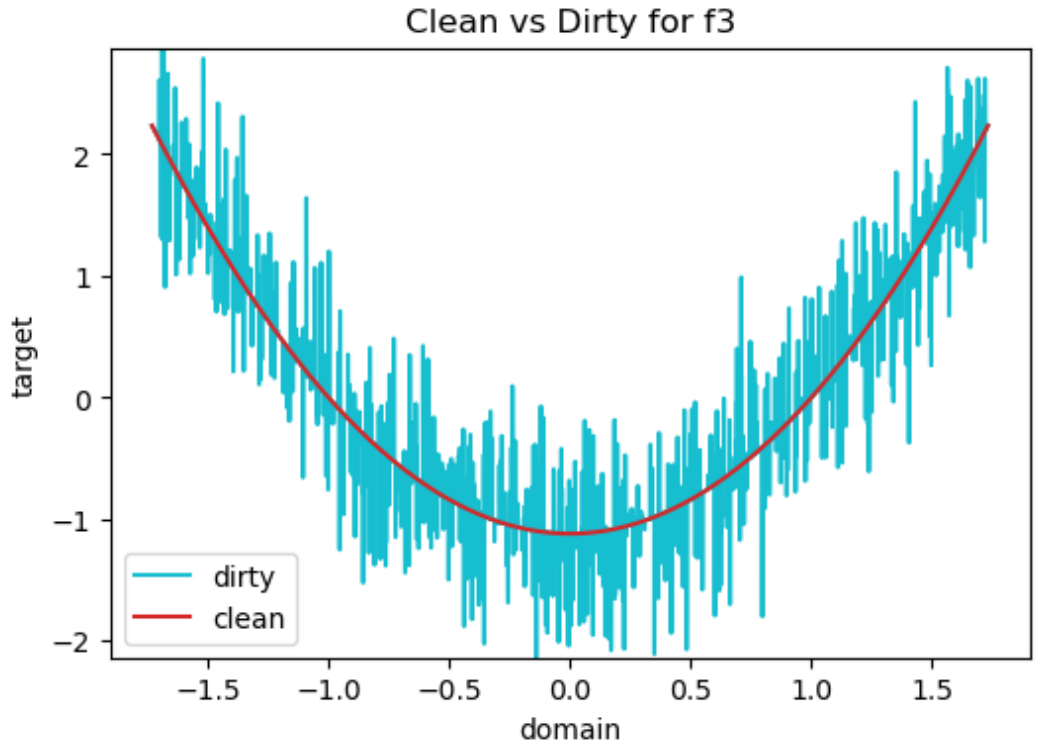
	X[0]	X[1]	X[2]	X[3]	X[4]	Y[i]
0	3.0	5.0	3.0	5.0	3.0	6.8
1	4.0	2.0	2.0	2.0	1.0	4.6
2	1.0	3.0	4.0	1.0	5.0	5.4
3	5.0	1.0	5.0	3.0	4.0	8.6
4	2.0	4.0	1.0	4.0	2.0	4.6

Visualizing the Dirty Dataset

$f_2(X)$ = RAND * x + RAND

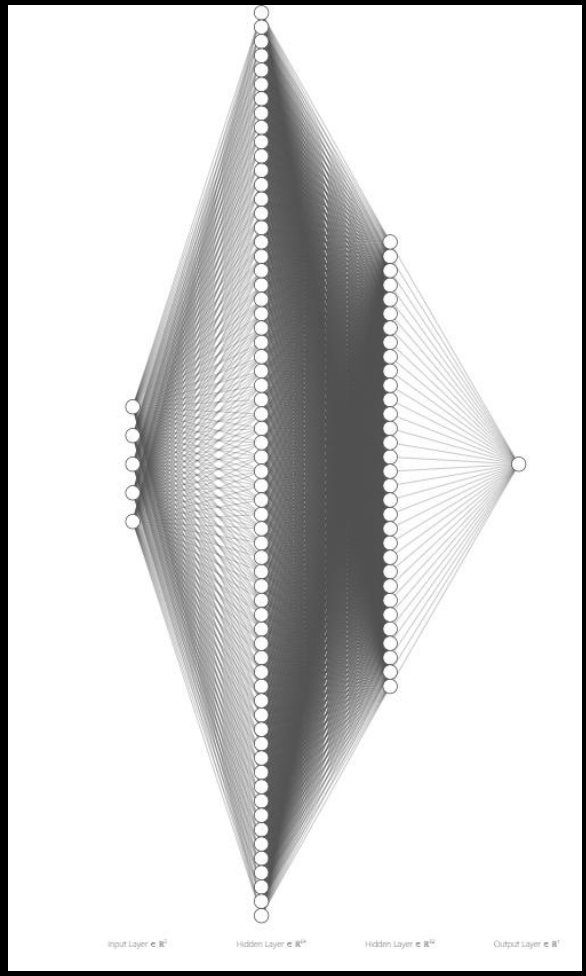


$f_3(X)$ = RAND * x^2 +RAND * x + RAND



Model Architecture

- I used the same model (shown on the right) for all of training. The only thing I would change was swapping one activation function for another
- Note that the model architecture was slightly down-scaled, where each node represents 2 nodes.
    - 10 -> 128 -> 64 -> 1



Hyperparameters

- Batch size: 64
- Epochs: 20
- Adam
  - B_1=0.9, β_2=0.999, learning rate=0.001, weight_decay=1e-10
- Train/val/test split:
  - Train: 5000 (71%)
  - val: 2000 (29%)
  - No test set

Experiments

Baseline

- There is no baseline for this dataset, because I invented it.
- However, the clean dataset **should** be result in extremely accuracy predictions.

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.

Results

- As you can see, ReLU, its copycats, 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.

Note: a "robust" activation function should result in a Dirty Loss – Clean Loss close to y=0

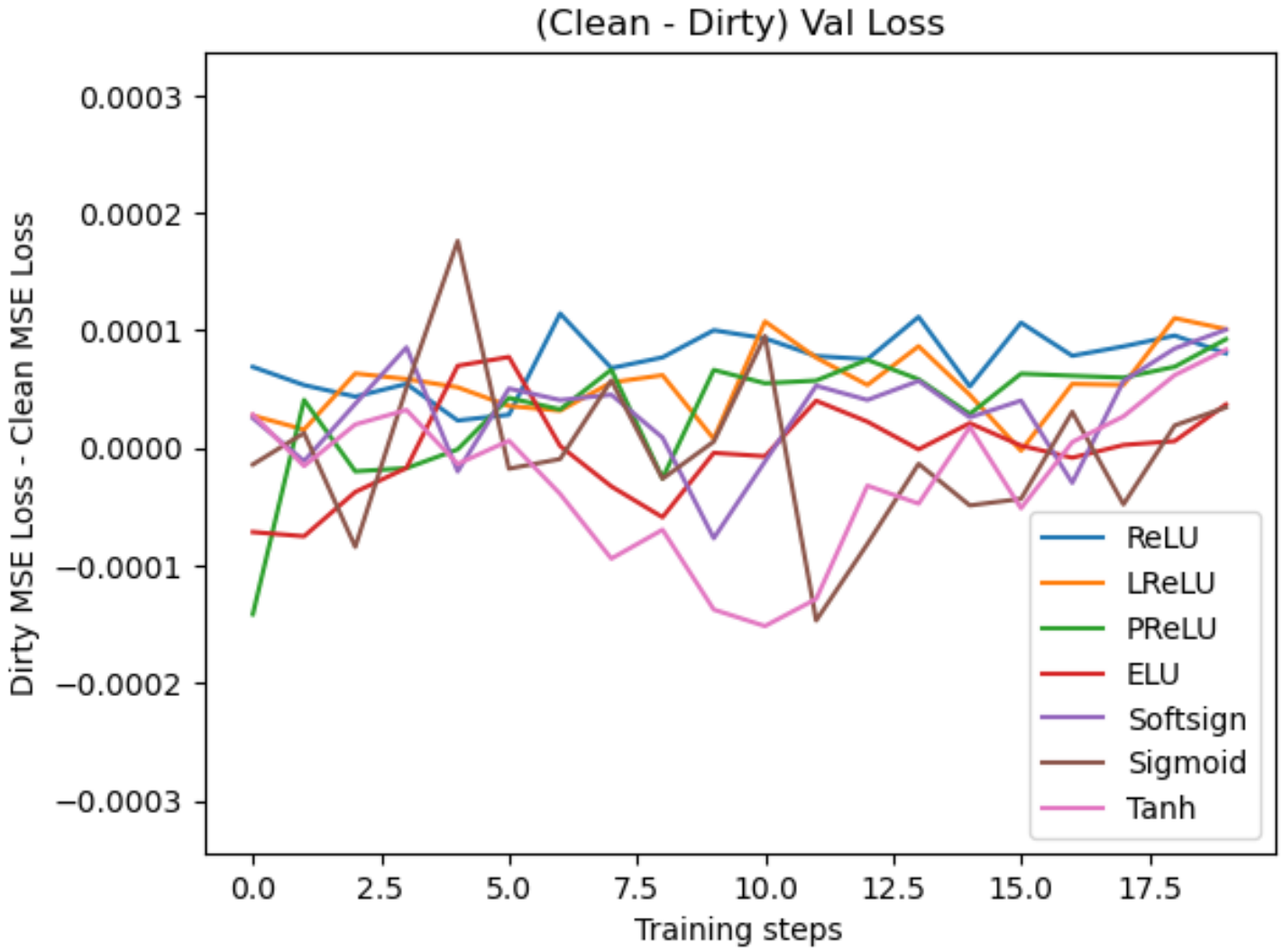
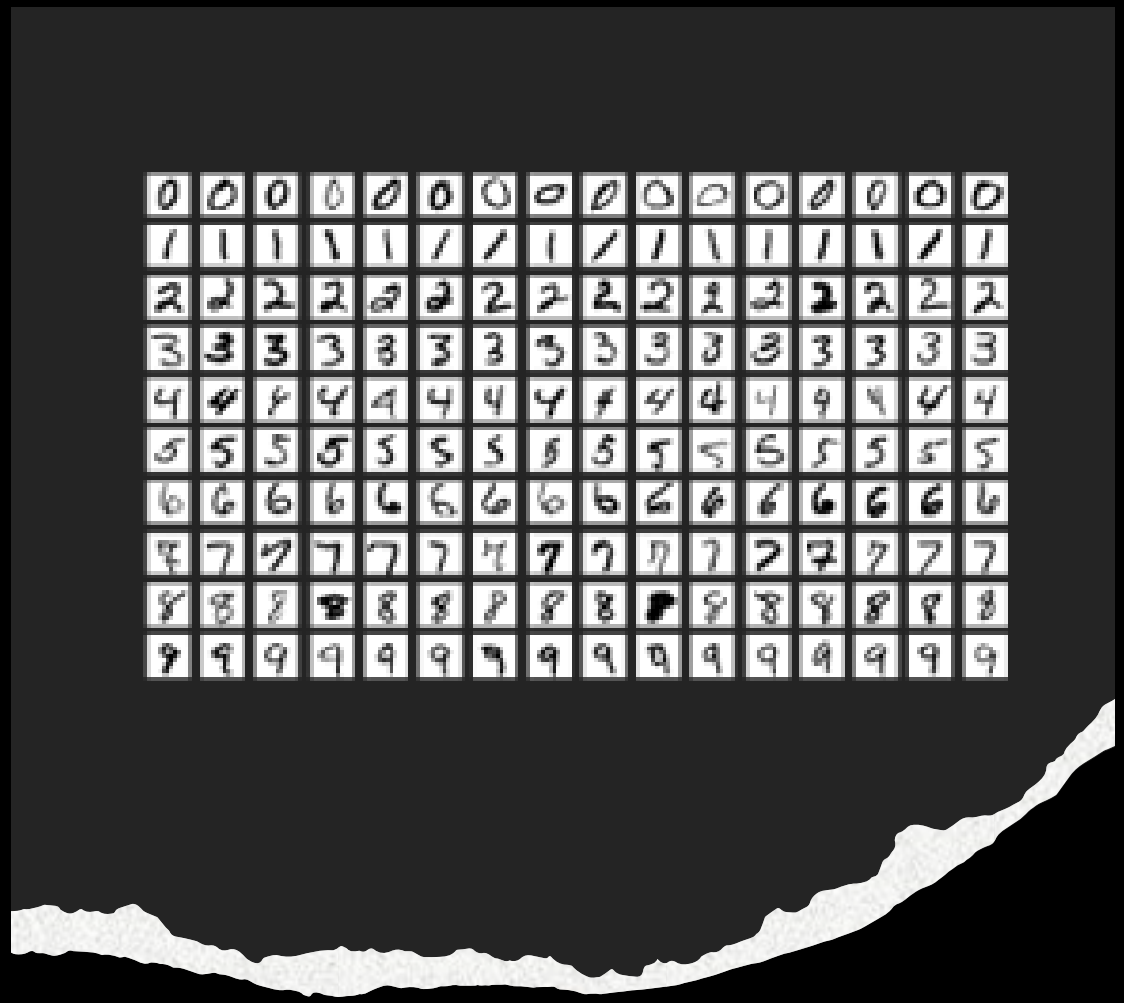


Image Classification

- For this comparison, I used the MNIST dataset because it is famously easy to train on. That is, it is very clean.
- To make this dataset dirty, I took 10% of the train y-values and changed them to incorrect values.

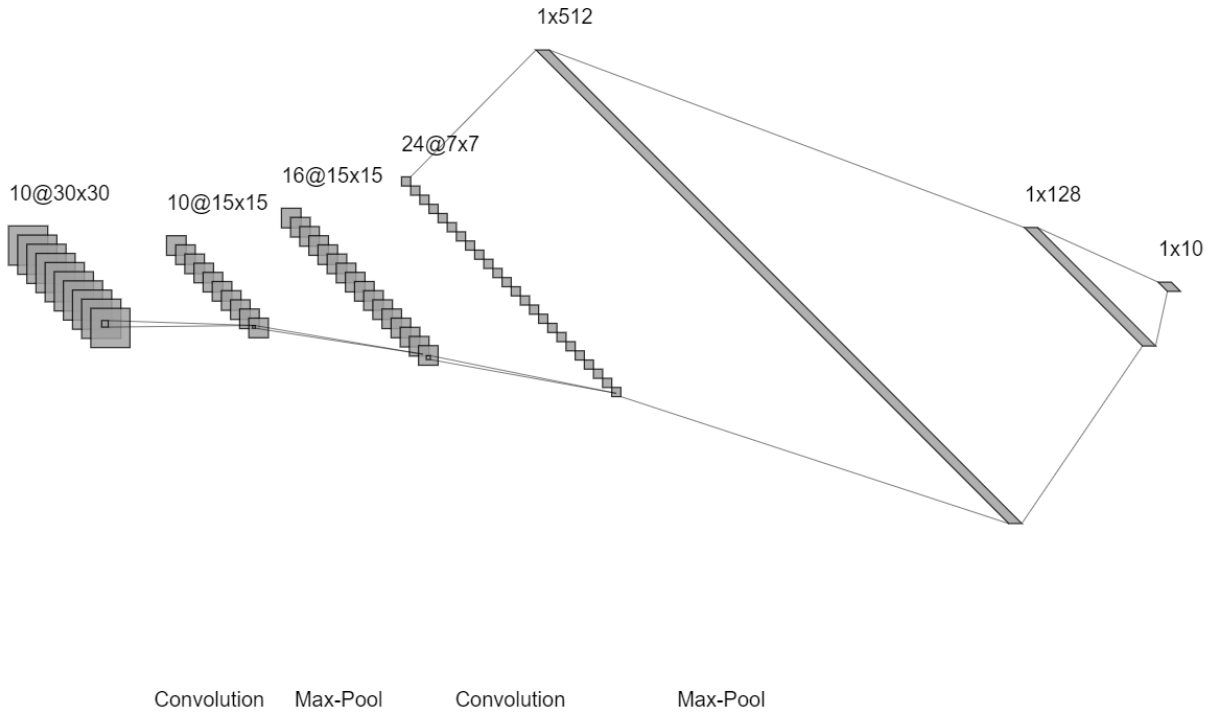


Data Preprocessing

- Resize to grayscale images from 28x28 to 64x64

Model Architecture

- I used the same model architecture, but I did one run-through where I added Dropout and weight decay to see how additionally regularization affected the robustness of the activation functions.
- It's important to note that I was severely limited in terms of time.
  - Different model for each dataset
  - Different model for each of the 7 activation functions
  - Each model takes 3-5 minutes to train
  - Fixing bugs



Hyperparameters

- Batch size: 128
- Epochs: 25
- Adam
  - B_1=0.9, β_2=0.999, learning rate=0.001, weight_decay=1e-10
- Train/val/test split
  - Train: 31,500 (45%)
  - val: 10,500 (15%)
  - test: 28,000 (40%), but didn't end up using it

Experiments

Baseline

- Achieving an accuracy of 99% or higher is to be expected for the MNIST dataset.
- Once again, we are comparing each model's accuracy when trained on the *clean* vs *dirty* datasets.
- From the Research Study
  - (N stands for normalized)

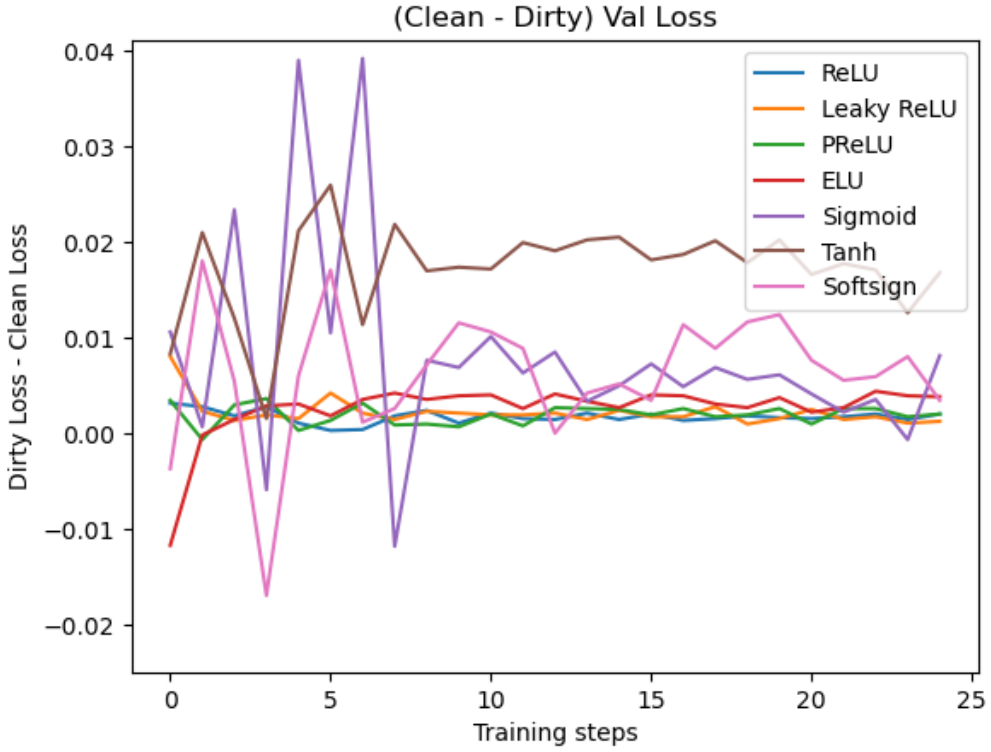
Evaluation Metrics

- 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.

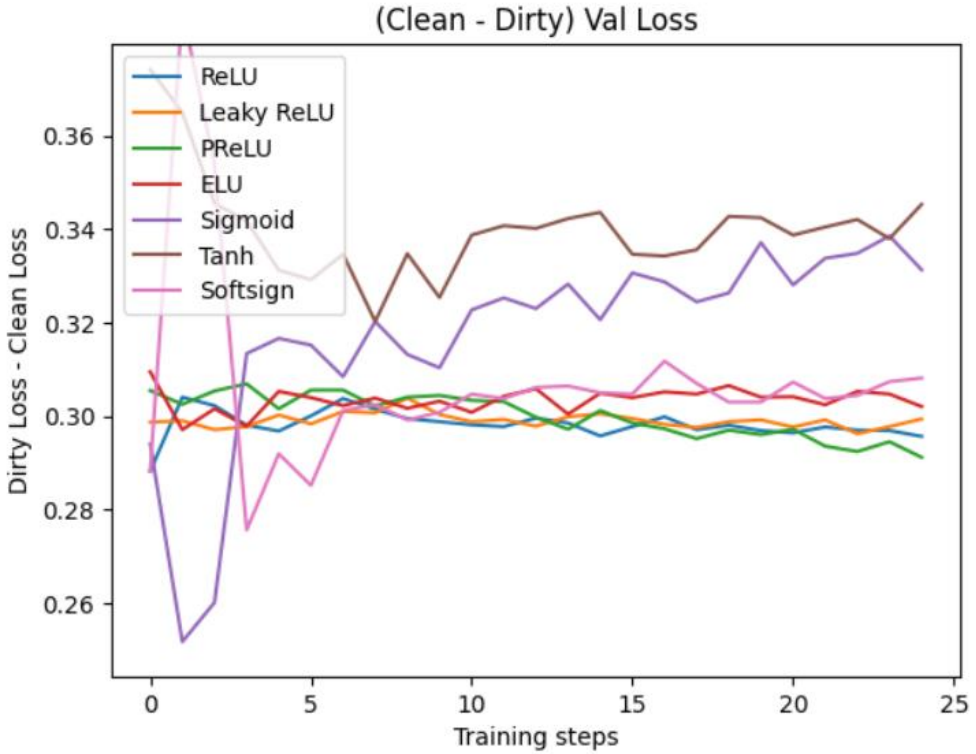
TYPE	Softsign	Softsign N	Tanh	Tanh N	Sigmoid
MNIST Acc Error	1.64	1.72	1.76	1.64	2.21

Results

Without Extra Regularization



With Dropout and Weight Decay



Note: a "robust" activation function should result in a Dirty Loss – Clean Loss close to y=0

Why does Sigmoid suck?

- When it comes to filtering out outliers, I'm still fairly certain that Sigmoid and Tanh are still better than ReLU and its copy-cats. However, most outliers are not "outliers" in the mathematical sense.
- "noise is inherent in every system" is a common phrase in data science, and I believe that this is precisely why Sigmoid and Tanh perform poorly.
  - That is, the outliers are not datapoints on the edge of a bell curve, in the wrong spot. So, Sigmoid and Tanh would not catch them.
- Also, Sigmoid creates manifolds with small slopes, leading to increased spurious local minimums and training times

Future Work

- As much as it saddens me, I think I have to give up my love for Sigmoid and admit that it is a bad activation function. As such, I will have no further work in this area.
- I would have loved to create my own activation function, but I think making one intelligently 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!

References

- Understanding the difficulty of training deep feedforward neural networks
- Activation functions and their characteristics in deep neural networks
- The Power of Approximating: a Comparison of Activation Functions
- How to Choose an Activation Function
- Small nonlinearities in activation functions create bad local minima in neural networks
- Neural Networks, Manifolds, and Topology

My Code: https://github.com/Zachary-Harrison/cs5890-ADL-project

Questions?