COMPLEXITY APPROXIMATION USING LEMPEL ZIV AND SENSITIVITY

Maria DaRocha

VUW, AIML431: Current Topics in Artificial Intelligence

1. INTRODUCTION

The following is my own work and supported by code available on GitHub and Google Collab.

The aim of this experiment was to demonstrate two types of complexity approximation measurements, Lempel Ziv and sensitivity, for two unique Fully Connected Neural Networks (FCNNs). In addition, different levels of regularisation were used to bias network weights, observing the resulting effects on network output complexities.

2. THEORY

Before discussing Lempel Ziv (LZ) complexity or sensitivity, it is important to first understand Kolmogorov complexity (K-complexity, henceforth) as a theoretical representation for a model's relative "simplicity," where there exists an encoding of the program applied in some universal language (e.g., universal Turing machine, UTM). [1]

Plain K-complexity asserts that, for any program which produces a measurable object as an output (e.g., a sequence of symbols), the program's complexity can be expressed as the length of the shortest possible description used to produce that object. *Prefix-free* K-complexity introduces the added constraint that an encoding within the sequence cannot first be part of another encoding.

The issue with K-complexity is that it tends to either be impractical or impossible to compute, with no way to discern which category a given program will fall into. For this reason, we instead use Lempel Ziv or sensitivity to quantify a program's relative complexity.

Lempel Ziv complexity is also known as the "K-complexity proxy" because it uses the code length of an object (sequence) *x* as a proxy for the *prefix-free* K-complexity of *x*. Computing the LZ-complexity for a binary sequence follows the method outlined in *Method 1* (*Appendix*). Crucially, LZ-complexity is computable, and the output of the compression is a prefix code which represents the shortest decodable description of the object.

Sensitivity is another approximator for complexity and though weaker, it is more common and practical to compute. Sensitivity $s(f, x)$ is a robustness measure that uses an indicator function and iterative bit shifts along the input sequence (function) to observe changes in the output. If a shift results in an output change, then the measure is incremented; if no change occurs, then robustness is retained. A more detailed explanation of the sensitivity approach can be found in *Method 2* (*Appendix*).

Alongside these measures, we must understand how probability effects function sampling. The Algorithmic Information Theorem (AIT) posits that simple outputs are more probable for random UTM programs and by extrapolation, we can prove that the same is true for functions (*Case 1, Appendix*).[1] In the case of neural networks, we expand the discussion one degree further to demonstrate their inductive bias towards simple functions (*Case 2, Appendix*).

Further, the method used for weight sampling can have a profound effect on a model's performance and the complexity of its outputs. Typically, we would expect high K-complexities to be the result of improbable functions, and low K-complexities to be the result of functions with high prior probabilities (simple functions).

As it relates to random weight sampling, it is true (by AIT) that random sampling is more likely to result in sampling simple functions than not. However, another useful practice for limiting complexity is regularisation.

Regularisation can refer to several techniques used to improve model generalisation and prevent overfitting (e.g., L2, L1, dropout, batch normalisation, etc.).[2] These techniques modify the learning process by encouraging smaller, simpler, or more sparse weights during training. Thus, regularisation, can also be understood as an approach for penalising complexity and artificially biasing a network toward simpler behaviour.

With this broader context, we can now reasonably discuss this experiment and motivations for certain aspects of its design.

[1] That is, highly probability functions must be simple (though the *inverse* isn't necessarily true).

[2] This paper defers to [2] for detailed explanation of L1, L2, and dropout as regularisation techniques.

3. EXPERIMENT

3.1. Experiment Set-Up

The experiment was set up using Python in Jupyter Lab.

Model f_1 was an FCNN mapping from $\{0,1\}^D$ to $\{0,1\}$, with two hidden layers of 64 nodes each and a single binary output. It used batch normalization, ReLU activation, dropout (20%), and sigmoid activation for the output. Complete model details can be found in *Table 1* (*Appendix*).

Model f_2 was an FCNN mapping from R^3 (normal distribution) to R^3 (unit sphere), with two hidden layers of 20 nodes each and a 3D output of real values. It used ReLU activation on hidden layers and a linear activation on the output. Complete model details can be found in *Table 2* (*Appendix*).

Both models…

- Incorporated LZ complexity and sensitivity metrics,
- Performed complexity evaluations on trained models, rather than randomly sampled weights, and
- Reported their performance and complexities at different regularisation strengths tailored to each model.

3.2. Motivations

Trained Networks vs. Randomly Sampled Networks
Trained networks were selected over randomly sampled networks for this experiment. Although randomly sampled networks would have sufficed for illustrating the interaction between function probability and complexity, they perform poorer in most practical applications.

Inducing Network Simplicity
Regularisation techniques were used during model training to bias the network towards increasingly simple structures (see 'Additional Features' and 'Regularisation' in *Table 1* and *Table 2, Appendix*). This was motivated by the decision to use trained networks over randomly sampled ones.

Complexity Measures
In real (complex) applications, sensitivity would be the preferred method for measuring complexity.[3] However, this application was simple enough to include both measures for educational purposes.

In this experiment, we can include LZ complexity because, for both models, we would expect the structure of the output sequence to be highly compressible (repeating).

- For model f_1 the dimensionality of the input is 10. Hence, the worst-case scenario (i.e., all 0 or 1) would still be a manageable 1024 bits (see *Workings 1, Appendix*).
- Additionally, a dynamic threshold was used such that classes would remain relatively balanced, further reducing the likelihood of a worst-case implementation.
- For model f_2, a 3D vector of real values was converted to a 3D binary vector using the sign of the scalars (+/-) as a discriminator. As real values are centered around zero, sampling and classifying in this way should be reasonably balanced, and so we can expect high repeatability in the output sequences of f_2, as well.
- For *these* reasons,[4] subsampling was not necessary for LZ complexity implementation, but *would* be advisable for more complex tasks, or where repeatability is less readily anticipated.

3.1. Results

The performance and complexity results for models f_1 and f_2 can be seen in *Table 3* and *Table 4* (*Appendix*), respectively.

As hypothesized, both models demonstrated decreases in average sensitivity and LZ complexity as regularisation strength increased. However, while robustness (low sensitivity) and low LZ-complexity are generally regarded as desirable traits, if the trade-off isn't well monitored, significant levels of regularisation will eventually result in severe underfitting.

Model f_2 and figures of its test data outputs (*Table 4, Appendix*) were included in this experiment to illustrate (in a more visual way than model f_1 allows) how biasing a network towards simplistic behaviour can also result in function oversimplification and as such, be equally as detrimental to a model's performance as overfitting – despite reporting increased robustness and low LZ-complexity.

4. CONCLUSION

This experiment successfully demonstrated the expected effects of regularisation on two distinct FCNNs, while also capturing two quantifiable measures for complexity, the Kolmogorov complexity proxy (i.e., Lempel Ziv complexity) and sensitivity.

Notably, while LZ-complexity is *technically* linear for one sequence (which allowed us to include it in these models), it can become *much* more expensive (both computationally and in storage) when applied to *many* or *extremely long, infrequently repeating* sequences.

In contrast, sensitivity only relies on forward passes through a network (in which operations are more straightforward, e.g., multiplication, activation, etc.). Thus, as the length and volume of sequences increases, the overhead for LZ complexity has the potential to far exceed that of sensitivity.

[3] [*Rationale discussed in conclusion.]*

[4] *[Task complexity and class balancing.]*

5. APPENDIX

5.1. Algorithms & Equations

(1) 'x' is a sequence of bits.
(2) Parse x into the shortest possible subsets without repetition.
(3) Describe the first occurrence of a subset and its index.
(4) Encode indices with prefix-free code.

Note: LZ-complexity requires two passes. The first is to acquire indices and their statistics and the second is to perform the encoding. For each sequence, LZ-complexity requires parsing, storing, and checking new substrings against previously indexed ones (i.e., dictionary search).

Algorithm 1: Lempel-Ziv Compression Algorithm

(5) $s(f,x) = \sum_{i=1}^{n} I\left(f(x) \neq f\left(x^{\oplus i}\right)\right)$
(6) $S(f) = \frac{1}{n} E_x[s(f,X)] = \frac{1}{n} * \frac{1}{2^n} \sum_{x \in \{0,1\}^n} s(f,x)$
where…

I is an indicator function and $x^{\oplus i}$ is x with the ith bit flipped. Equation (6) expresses sensitivity for the Boolean functions: $f: \{0,1\}^n \to \{0,1\}$

Algorithm 2: Sensitivity for Function Complexity

(7) (AIT): $2^{-K(x)} \leq P(x) \leq 2^{-K(x)+O(1)}$
(8) Consider nontrivial function $f^*$with finite outputs: Each output x of f^*can be encoded with length:
$$L \leq -\log(P(x)) + 1$$
(9) From the definition of complexity (AIT):
$$-\log(P(x)) + 1 \geq L \geq K(x)$$
(10) $\therefore P(x) \leq 2^{-K(x)+o(1)}$
where…

$K(x)$ represents the Kolmogorov complexity of x and (9) implies that if the probability of x is small, then its length will be large and if the probability of x is large, then the length will be small (approach zero).

Case 1: AIT for Sequences and Functions

Neural network f has parameters θ, with prior distribution $P_{par}(\theta)$. Prior probability of f is:

(11) $P(f) = \int I(f_\theta, f) P_{par}(\theta) d\theta$, *where…*
(12) $I(f_\theta, f) = \begin{cases} 1, f = f_\theta \\ 0 \; otherwise \end{cases}$

a. Sample parameters from the distribution to obtain functions
b. Measure the K-complexity of the functions
c. Measure the probability of the functions
d. Confirm each function satisfies:
$$P(f) \leq 2^{-K(f)+o(1)}$$
e. Determine if $P(f)$ is nonuniform and inductive bias exists

Case 2: AIT Extended, Inductive Bias of Neural Networks

Boolean functions $f: \{0,1\}^n \to \{0,1\}$

(13) f has 2^n possible inputs
(14) 2^{2n} possible functions
(15) $q = 2^n$ unique input symbols
(16) For q symbols there are 2^q possible symbols
(17) $\therefore K(f)$ = complexity of length 2^noutputs

Workings 1: LZ Complexity Compute

5.2. Figures & Tables

Model f_1	$\{0,1\}^D \to \{0,1\}$
Network Architecture	Fully Connected NN
Input Layer	$\{0,1\}^{10}$
Hidden Layers	2 layers, 64 nodes each
Output Layer	$\{0,1\}$
Additional Features	• Batch norm and ReLU activation on hidden layers • Dropout (20%) • Output activation sigmoid
Regularisation	[0.00, 0.05, 0.10, 0.50]
LZ Complexity	• Compute: • $2^{10} = 1024$, $O(10^3)$ [5] • Memory: • 1024×10, (~1.25 KB)
Sensitivity	• Compute: • $2^{10} \times (10+1)$ passes[6] • $O(72 \times 10^6)$ • Memory: • Linear in terms of θ (model params)

Table 1: Model f_1 Specifications

[5] See *Workings 1: LZ Complexity Compute* (*Appendix*)

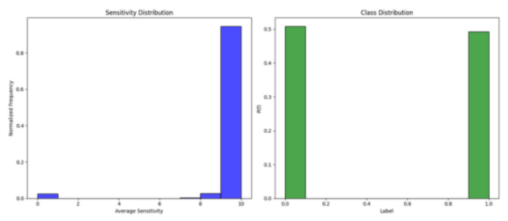
[6] 1 forward pass (fp) for original input, 10 fps for each bit flip

Model f_2	$R^3 \rightarrow R^3$
Network Architecture	Fully Connected NN
Input Layer	R^3 (Normal distribution)
Hidden Layers	2 layers, 20 nodes each
Output Layer	R^3 (Unit sphere)
Additional Features	• ReLU activation on hidden layers • Output activation is linear
Regularisation	[0.00, 0.01, 0.025, 0.05]
LZ Complexity	• Compute: • 100 samples, $O(3)$ • Memory: • 300 binary values (100 {0,1} sequences, each of length 3)
Sensitivity	• Compute: • 100 forward passes • $100 \times O(520)$[7] • Memory: • Linear in terms of θ (model params)

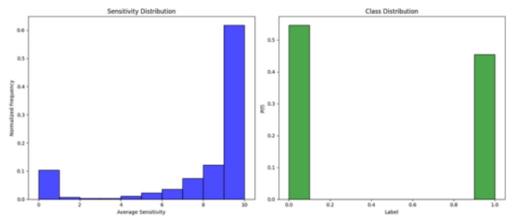
Table 2: Model f_2 Specifications

Model f_1: Results Table

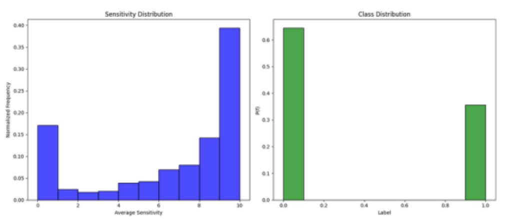
Model f_1: s(f,x) to Normalised Freq. (left), Class Distr. (right)



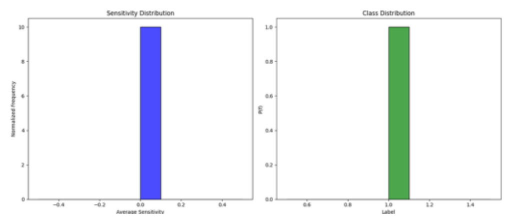
f_1 (Reg Str. = 0.0)



f_1 (Reg Str. = 0.05)

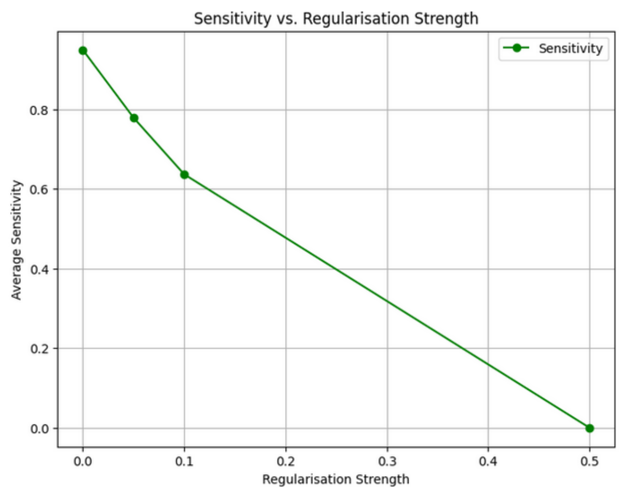


f_1 (Reg Str. = 0.1)



f_1 (Reg Str. = 0.5)

Model f_1: Sensitivity vs. Regularisation Strength

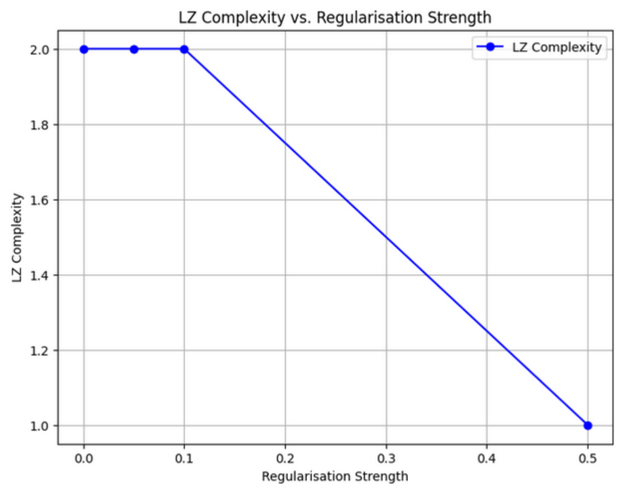


Regularisation Strength	*Average Sensitivity*
0.00	*0.949*
0.05	*0.778*
0.10	*0.637*
0.50	*0.000*

Cont'd…

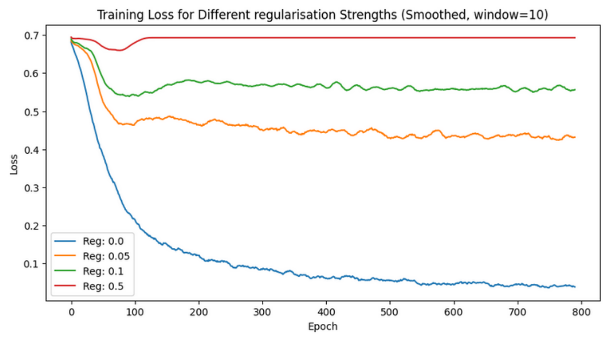
[7] *Input to 1st HL: 3D, 20 nodes* $O(3 \times 20) = \boldsymbol{O(60)}$ |
1st HL to 2nd HL: 20 nodes to 20 nodes $O(20 \times 20) = \boldsymbol{O(400)}$ |
2nd HL to Output: 20 nodes, 3 nodes $O(3 \times 20) = \boldsymbol{O(60)}$ |

Model f_1: LZ Complexity vs. Regularisation Strength



Regularisation Strength	*LZ Complexity*
0.00	*2*
0.05	*2*
0.10	*2*
0.50	*1*

Model f_1: Training Losses vs. Regularisation

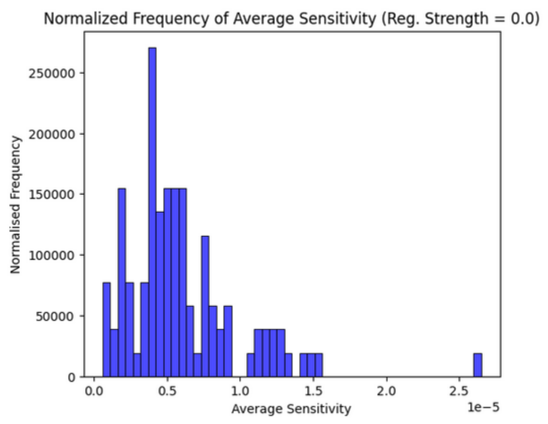


Regularisation Strength	*Final Loss*
0.00	*0.0265*
0.05	*0.4345*
0.10	*0.5574*
0.50	*0.6930*

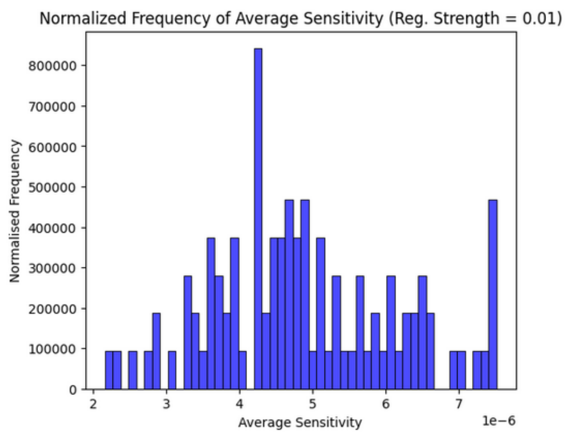
Table 3: Model f_1 Results

Model f_2: Results Table

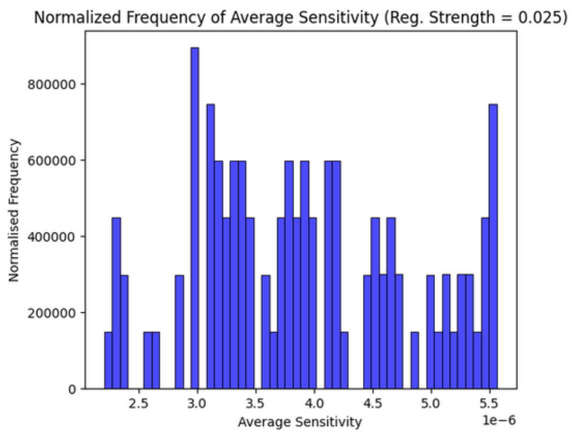
Model f_2: Average Sensitivity to Normalised Frequency



f_2 (Reg Str. = 0.0)

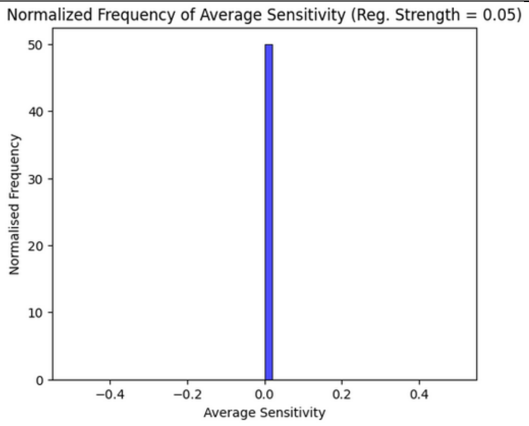


f_2 (Reg Str. = 0.01)



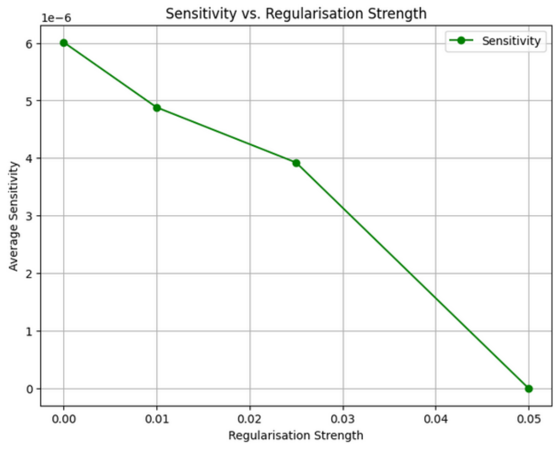
f_2 (Reg Str. = 0.025)

Cont'd…



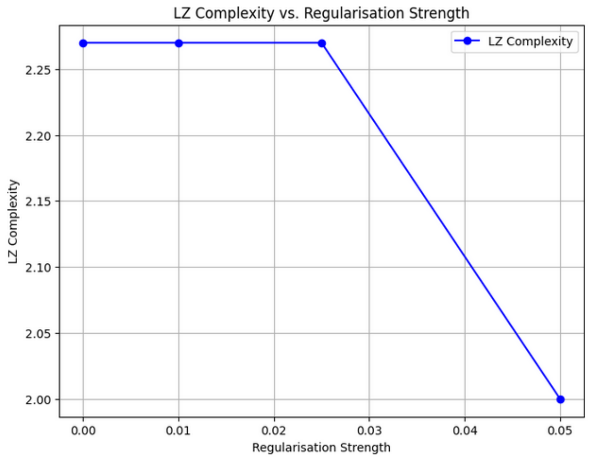
f_2 (Reg Str. = 0.05)

Model f_2: Sensitivity vs. Regularisation Strength



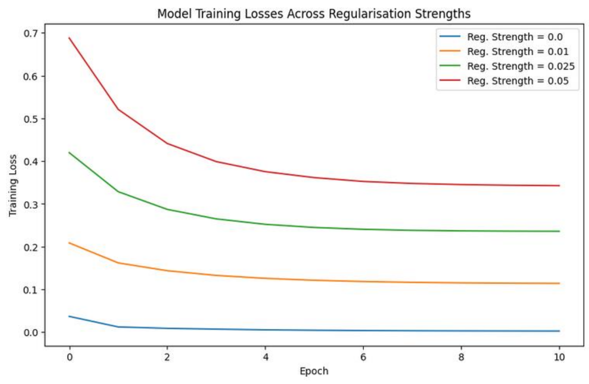
Regularisation Strength	*Average Sensitivity*
0.00	$6.013e^{-06}$
0.010	$4.881e^{-06}$
0.025	$3.922e^{-06}$
0.050	*0.000*

Model f_2: LZ Complexity vs. Regularisation Strength



Regularisation Strength	*LZ Complexity*
0.00	*2.27*
0.010	*2.27*
0.025	*2.27*
0.050	*2*

Model f_2: Training Losses vs. Regularisation



Regularisation Strength	*Final Loss*
0.00	*0.0026*
0.010	*0.1141*
0.025	*0.2359*
0.050	*0.3426*

Cont'd…

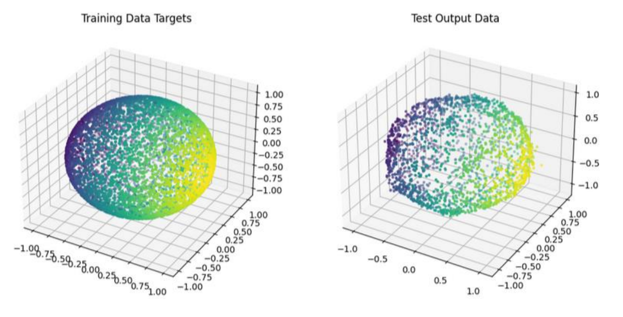
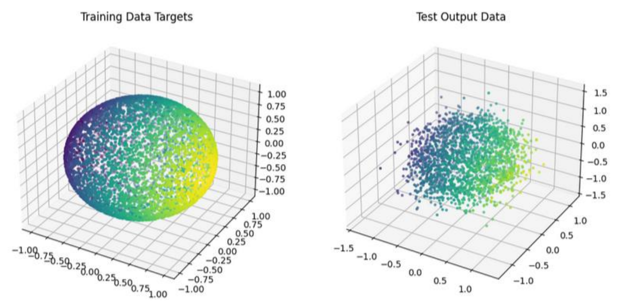
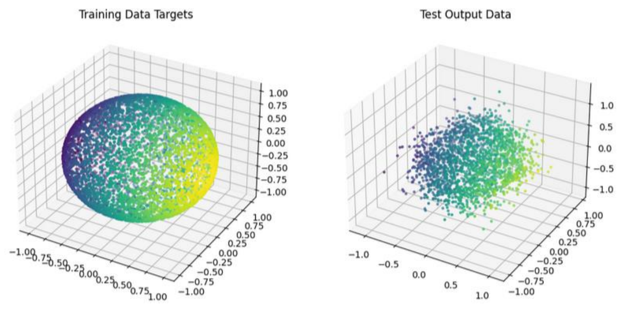
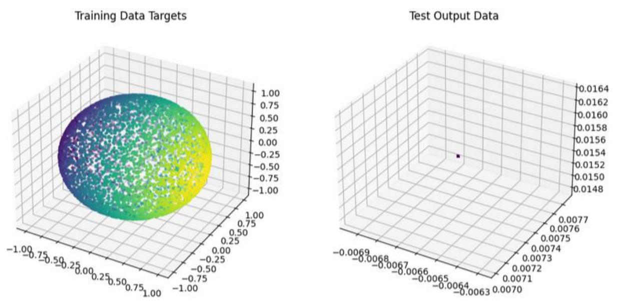
Model f_2: Test Output Data vs Regularisation Strength
 **f_2 (Reg Str. = 0.0)**
 **f_2 (Reg Str. = 0.01)**
 **f_2 (Reg Str. = 0.025)**
 **f_2 (Reg Str. = 0.025)**

Table 4: Model f_2 Results

6. REFERENCES

[1] B. Kleijn, “Why Neural Networks Work Well: AIML431 [Lecture 11 Notes],” *Master of Artificial Intelligence, Victoria University of Wellington*, Sep. 2024, Accessed: Sep. 21, 2024. [Online]. Available: https://ecs.wgtn.ac.nz/foswiki/pub/Courses/AIML431_2024T2/LectureSchedule/whyTheyWork.pdf

[2] M. DaRocha, “Demonstrating the Basic Principles of Generative Models for (FCNN) Gaussian and Uniform Mappings,” *VUW, AIML425: Neural Networks and Deep Learning*, Aug. 2024, Accessed: Sep. 10, 2024. [Online]. Available: https://github.com/Marianette/A2-AIML425