Imperial Coursework on Machine Learning (ImpCML 2024) Submission

Timothy Chung [1] **Yuxuan Gu** [1] **Anlan Qiu** [1] **Sebastian Tan** [1]

Abstract

This paper studies the adversarial dataset Kryptonite-N from (Quinn & Luther, 2024), claimed to be a potential counter-example to the Universal Approximation Theorem. In this research, we critically examine these claims and explore theoretical reasons for models failing to learn on Kryptonite-N as an issue with optimising the loss function. We show that Kryptonite-N can be accurately classified with methods such as neural networks and support vector machines, up to dimensions $N = 18$. Our work also provides further insight into future work for classifying Kryptonite-N in higher dimensions.

1. Introduction

The **Universal Approximation Theorem (UAT)**, states that a neural network F with a finite number of neurons can approximate any continuous function f, given appropriate activation functions and weights:

Let $\phi : \mathbb{R} \to \mathbb{R}$ be a non-constant, bounded, and continuous activation function. For any continuous function $f : [a, b]^n \to \mathbb{R}$ on a compact domain $[a, b]^n \subset \mathbb{R}^n$ and any $\varepsilon > 0$, there exists a neural network of the form:

$$F(x) = \sum_{i=1}^{M} w_i \phi\big(\langle v_i, x\rangle + b_i\big) + c,$$

where $x \in [a, b]^n$, $w_i, c \in \mathbb{R}$, $v_i \in \mathbb{R}^n$, $b_i \in \mathbb{R}$, such that:

$$\sup_{x \in [a,b]^n} \big| f(x) - F(x) \big| < \varepsilon.$$

Kryptonite-N is a collection of N-dimensional datasets with binary labels for a classification problem. (Quinn & Luther, 2024) were unable to accurately classify Kryptonite-N, positing that this behaviour was a potential counter-example to the universal function approximation theorem.

In this paper, we analyse the claims made by (Quinn & Luther, 2024) , by using neural networks, support vector machines and the XGBoost algorithm to fit the Kryptonite-N dataset. We start from $N = 9$, tune hyperparameters, and record the accuracies for each of the best performing models. If the model meets the target accuracies specified in (Quinn & Luther, 2024), the model is subsequently trained on the next largest dataset. We successfully achieve the performance threshold for dimensions up to $N = 18$.

Furthering our analysis, we analyse the mutual information between the dataset's features in X and the predicted variable y. We show Kryptonite-N exhibits strong feature interactions within its input features X, making it inherently resistant to feature selection methods. The feature interactions indicate the possibility of a combinatorial feature selection problem which is NP-hard, suggesting that the search space for optimal solutions becomes increasingly difficult to traverse as N increases, leading to a more complex loss landscape. We draw similarities between our challenges in higher dimensions and the learning difficulties faced by (Quinn & Luther, 2024), suggesting that the learning issue is not the failure of the UAT, but fundamentally an optimisation problem.

2. Methodology

2.1. Neural Networks

2.1.1. Network Architecture

The experiments carried out for this project utilised feed-forward neural networks (NN) depicted in Figure 1 with fully connected layers in-between. For N-dimensional inputs, the first layer of the NN contains N nodes; for the output of this binary classification task, the output layer contained a single output neuron with a sigmoid activation to bound the output between 0 and 1.

The hyperparameters of interest here are the number of hidden layers and the number of nodes per layer, as these directly affect the complexity and "power" of the model.

[1]Department of Electrical and Electronic Engineering, Imperial College London, London, United Kingdom. Correspondence to: Timothy Chung <timothy.chung21@imperial.ac.uk>, Yuxuan Gu <yuxuan.gu21@imperial.ac.uk>, Anlan Qiu <anlan.qiu21@imperial.ac.uk>, Sebastian Tan <sebastian.tan21@imperial.ac.uk>.

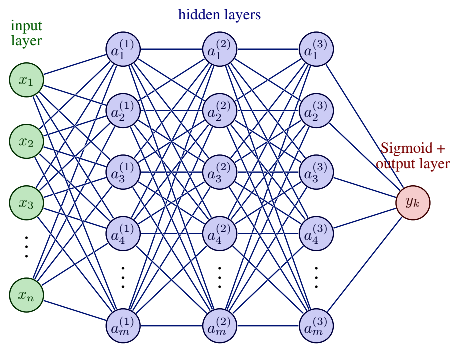

Figure 1. Neural Network for N-dimesional input

Additionally, our activation function, ReLU and GeLU, can also be tuned for optimality.

$$\text{ReLU}(x) = \max(0, x). \tag{1}$$

$$\text{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right). \tag{2}$$

$\Phi(x)$ denotes the cumulative distribution function of the standard normal distribution; erf is the Gaussian error function defined in Appendix A.

ReLU was used due to its computational simplicity, allowing for faster training and avoiding the vanishing gradient problem. Alternatively, GeLU activation worked better for the higher-dimensional datasets due to smoothness introduced by the erf() function – this behaviour is discussed further in 4.2.

Each neuron in a hidden layer computes a weighted sum of its inputs, adds a bias term, and applies an activation function to the result. Assume the hidden dimension is d for all hidden layers, for a NN with L layers, the output of the l-th layer $\mathbf{a}^{(l)}$ is computed as shown in Eq. (3) and (4).

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \tag{3}$$

$$\mathbf{a}^{(l)} = \sigma\left(\mathbf{z}^{(l)}\right), \tag{4}$$

$\mathbf{W}^{(l)} \in \mathbb{R}^{d\times d}$ is a weight matrix for layer l; $\mathbf{b}^{(l)} \in \mathbb{R}^d$ is the bias vector for layer l; $\mathbf{z}^l \in \mathbb{R}^d$ is the pre-activation value; σ is an activation function introducing non-linearity to the network; $\mathbf{a}^l \in \mathbb{R}^d$ is the input for layer l.

2.1.2. Loss Function

The loss function used in all the neural networks tested were **Binary Cross-Entropy Loss** (BCE Loss), given by the following.

$$\mathcal{L}_{\text{BCE}}(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]. \tag{5}$$

where:

- N is the total number of samples.
- $y_i \in \{0, 1\}$ is the true label for the i-th sample.
- $\hat{y}_i$ is the predicted probability that the i-th sample belongs to the positive class (class 1).

The binary cross-entropy loss (BCE) was used due to the binary nature of the ground truth being 0 or 1. The BCE measures the difference between the predicted probability distribution and the actual target probability distribution. Since the BCE function is differentiable, gradient-based optimisers can be used.

2.1.3. Backpropagation and Optimisers

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}}\frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(l)}} = \delta^{(l)}\left(\mathbf{a}^{(l-1)}\right)^T. \tag{6}$$

We utilise gradient-based optimisers to minimise the loss, updating the model weights with each iteration with **backpropagation**. The gradient with respect to weight matrix at layer l is computed in Eq. (6), where $\delta^{(l)} \in \mathbb{R}^d$ is defined in Eq. (7) for output layer and in Eq. (8) for hidden layers. $\odot$ denotes Hadamard (element-wise) product.

Output layer:

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{(L)}}\frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{z}^{(L)}} = \nabla_{\mathbf{a}^{(L)}}\mathcal{L} \odot \sigma'\left(\mathbf{z}^{(L)}\right). \tag{7}$$

Hidden layers:

$$\begin{aligned}\delta^{(l)} &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l)}} = \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}}\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(l+1)}}\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \\ &= \left(\mathbf{W}^{(l+1)}\right)^\top \delta^{(l+1)} \odot \sigma'\left(\mathbf{z}^{(l)}\right), \quad \forall l \in (0, L).\end{aligned} \tag{8}$$

The most effective optimiser in our experiments was **A**daptive **M**oment **E**stimation (**Adam**), introduced in (Kingma & Ba, 2015). It has the following parameter update rule:

$$\theta_{t+1} = \theta_t - \alpha\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \tag{9}$$

where:

- θ_t: Current parameter (weight) at time step t.
- θ_{t+1}: Updated parameter at time step $t + 1$.
- α: Learning rate, a hyperparameter that will be chosen.
- $\hat{m}_t$: Bias-corrected first moment estimate (*i.e.*, mean of gradients $\hat{m}_t = \mathbb{E}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}\right]$).
- $\hat{v}_t$: Bias-corrected second moment estimate (*i.e.*, variance of gradients $\hat{v}_t = \mathbb{V}\left[\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}\right]$).
- ϵ: Small constant to prevent division by zero.

At a high level, Adam is a robust optimiser it combines both momentum and adaptive learning rates for each parameter.

2.2. Support Vector Machines

2.2.1. Hyperplane

The second-most performant algorithm tested was the support vector machine (SVM) with a radial basis function (RBF) kernel. The SVM finds the most optimal separating plane between classes.

The SVM's hyperplane is defined as $\mathbf{w} \cdot \mathbf{x} + b = 0$, where:

- $\mathbf{w}$: the **weight vector** perpendicular to the hyperplane
- $\mathbf{x}$: the input vector
- b: the **bias** that shifts the hyperplane from the origin.

The **margin** is the distance between the hyperplane and the closest points of each class (**support vectors**).

2.2.2. Objective Function

The SVM maximises the margin between the nearest data-points of each class and the hyperplane, as a wider margin is robust and generalisable to new data. Since data is rarely perfectly separable in practice, we introduces regularisation multiplier C to penalise misclassifications with slack ξ_i (soft-margin SVM).

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \tag{10}$$

$$\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i \tag{11}$$

The margin width is given by $\frac{2}{\|\mathbf{w}\|}$. This is a solvable **quadratic programming** problem. For large datasets, the dual formulation is used:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j) \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{N}\alpha_i y_i = 0 \quad \forall i \end{aligned} \tag{12}$$

Where α_i denotes the Lagrange multipliers for each data point, with $0 < \alpha_i < C$ identifying support vectors that maximise the margin width. (Berwick, 2011) provides an intuitive explanation illustrated in Figure 2: the SVM chooses support vectors, assigning nonzero α_i, α_j for similar $\mathbf{x_i}$, $\mathbf{x_j}$ that predict different classes. Different classes imply $y_i y_j = -1$, leading to the term $\alpha_i\alpha_j y_i y_j \mathbf{x_i}\cdot\mathbf{x_j}$ being negative in Equation 12. Since the summation is multiplied by $-\frac{1}{2}$, the sum is maximised.

2.2.3. Kernel Function

We replace the dot product $(\mathbf{x}_i \cdot \mathbf{x}_j)$ in Equation 12 with a transformed distance metric in a higher dimensional space, the **Radial Basis Function** (RBF). This kernel was chosen as it satisfies the Universal Approximation Theorem (Shanthi & Sathiyapriya, 2022) and has strong performance in a range of applications compared to other kernels, as seen in (Thurnhofer-Hemsi et al., 2020).

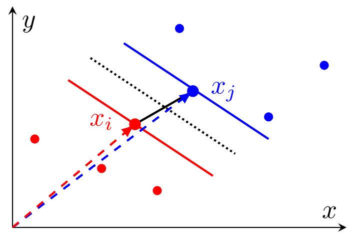

Figure 2. Graphical intuition: two very similar vectors, x_i and x_j, predicting different classes, tend to maximise the margin width.

The RBF kernel between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2\right). \tag{13}$$

where:

- $\|\mathbf{x}_i - \mathbf{x}_j\|$ is the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$
- γ controls the width of the Gaussian function. Higher γ results in a narrower Gaussian influence, leading to a more complex decision boundary around each data point.

The RBF kernel calculates a normalised similarity between data points based on their distance. For two points $\mathbf{x}_i$ and $\mathbf{x}_j$ that are far apart, the kernel value is approaches 0.

The key hyperparameters that can be tuned are C in Equation 10 and γ. C is the penalty for misclassifying points. Higher C leads to complex boundaries (overfitting risk) and lower C allows for for misclassifications (smoother boundaries with underfitting risk).

2.3. XGBoost

XGBoost (Extreme Gradient Boosting) (Chen & Guestrin, 2016) is a highly efficient gradient boosting algorithm that iteratively refines decision trees using second-order optimisation. Its scalability, regularisation, and robust performance make it well-suited for structured data and predictive modelling tasks, justifying its use in our work. The prediction of the XGBoost model is given by a summation of the prediction of K weak learners:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}, \tag{14}$$

- $\hat{y}_i$: Predicted value for the i-th data point.
- K: Number of trees.
- $f_k(x_i)$: Prediction of the k-th tree for input x_i.
- $\mathcal{F}$: Denotes the space of regression trees.

The objective function of XGBoost is:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k). \tag{15}$$

where $l(y_i, \hat{y}_i)$ is a differentiable loss function, in which we chose Binary Cross entropy (see Eq. (5)). $\Omega(f_k)$ is a regularisation term (See Appendix B).

The main hyperparameters tuned in this experiment are K- the number of trees; T- number of leaves in the tree; and the maximum depth of each tree.

3. Experimental Design

This study investigates the applicability of the **universal approximation theorem** to the Kryptonite-N dataset. To validate this, various classes of machine learning models are trained on the dataset. Achieving the target accuracy specified in Table 1 with any model will demonstrate the feasibility of approximating the dataset using machine learning models, thereby supporting our hypothesis.

The Kryptonite-N dataset comprises 7 sets of N-dimensional data, where N ranges from 9 to 45. Each dataset is divided into training (60%), validation (20%), and testing (20%) sets. Starting with the simplest 9-dimensional dataset, three types of models (NN, SVM, XGBoost) are trained on the training set, with hyperparameters optimised based on validation accuracy to prevent contamination of the testing data.

N	9	12	15	18	24	30	45
Target (%)	95	92.5	90	87.5	80	75	70

Table 1. Target accuracies for different datasets

3.1. Justifying the Data Split

According to (Wicker, 2024), the following bound is derived from Hoeffding's inequality:

$$n \geq \frac{1}{2\epsilon^2} \log \frac{2}{\delta}. \tag{16}$$

This describes the minimum quantity for n datapoints to guarantee with confidence $1 - \delta$ that the empirical mean approximates the true mean within error tolerance ϵ, ensuring the reliability of the training set. Given the high target accuracies in Table 1 we select $\epsilon = 0.1$ and $\delta = 0.01$, yielding a minimum required training size of $n = 6931$. The Kryptonite-9 dataset, with 18000 samples, provides 10800 samples (60%) for training, which exceeds this minimum requirement.

3.2. Hyperparameter Tuning Process

Hyperparameters significantly influence model performance especially for SVMs, as shown in Appendix C. To optimise each model, a consistent tuning methodology was employed: Initially, candidate hyperparameters were explored using Grid Search, which exhaustively evaluates all combinations within a predefined grid. Based on the results, the best-performing model was further refined manually, based on the Grid Search results. This iterative process increases model complexity until it overfits the data– indicated by divergence in training and validation loss curves. The model achieving the highest accuracy prior to overfitting was selected as the **best model** for its class– we record accuracy results in Table 2.

4. Experimental Results

4.1. Comparing Accuracies of Different Models

The accuracy results of the best models in each model class are recorded for different N dimension dataset variants in Table 2. The universal approximation theorem holds empirically for a subset of Kryptonite-N where $N = 9, 12, 15, 18$.

N	**9**	**12**	**15**	**18**
Target (%)	95	92.5	90	87.5
Neural Network	96.3	96.3	93.4	90.68
SVM	95.8	93.4		
XGBoost	85.0			

Table 2. Percentage accuracy of models on Kryptonite-N test data. Blank entries indicate an accuracy below 55%.

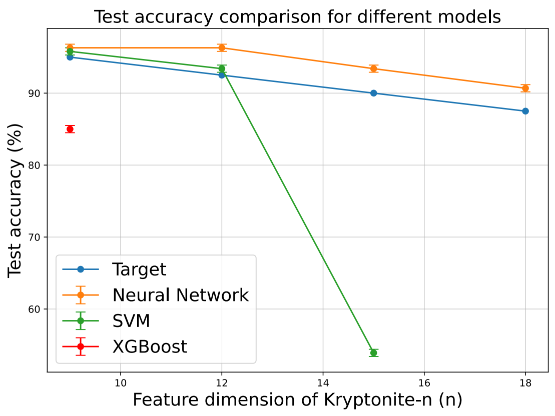

Figure 3. Test accuracy comparison of the Neural Network model and SVM model for different N dimensions.

The NN architecture was the most accurate model across all dimensions, achieving all target accuracies. On the other hand, the XGBoost failed to reach all accuracy targets, despite its best performance being relatively close to 95% accuracy for the Kryptonite-9 dataset. SVM performed better than the XGBoost but fell short of the NN, achieving accuracy targets for the $N = 9, 12$, but failing at $N = 15$.

NN vs. SVM For $N = 9, 12$, the two models show similar levels of test accuracy, both achieving above the target. However, as N increases to 15 and 18, the NN maintains above target accuracy, but the SVM performance deteriorates to 53.9%, close to random guessing. This aligns with the fact that generalisation becomes more challenging as dimensionality increases. NNs generalise better than SVMs as they excel at feature learning from raw data, not requiring manual feature engineering. SVMs overly rely on predefined kernels and hyperparameters to draw decision boundaries, which limits their flexibility in separating the data.

NN vs. XGBoost The tree-based XGBoost algorithm was unable to achieve the target accuracy even at the smallest dataset. This could be attributed to the fact that XGBoost is composed of simple **summations** of the predictions of a number of weak learners (see Eq. (14)); whereas neural networks model data using a **hierarchical composition** of weighted sums and activation functions (see Eq. (3) and (4). This hierarchical composition enables NNs to learn complex hierarchical (combinations of) feature patterns, which may have been more dominant in the Kryptonite-N dataset– we discuss this in 6.

4.2. Hyperparameters of Models

Besides the choice of model, the choice of hyperparameters of a model has a significant influence on whether the model learns from the dataset, and achieves a desirable test accuracy. This section will address two key considerations:

- Hyperparameter selection for one dataset $(N = 15)$.
- Variations in hyperparameter selection between the datasets.

Learning rate	**1e-5**	**1e-4**	**1e-3**	**1e-2**
Test loss	0.693	0.241	0.693	0.693
Test accuracy (%)	49.6	93.3	49.6	49.6

Table 3. Test losses versus accuracies of a neural network with ADAM across different learning rates on Kryptonite-15

Optimiser	**Adam**	**SGD**	**SGD with Momentum**
Test loss	0.227	0.693	0.693
Test accuracy (%)	93.4	49.8	50.6

Table 4. Comparison of test losses and accuracies of a Neural Network when trained with different optimisers on Kryptonite-15

Hyperparameter Selection for N=15 For Kryptonite-15, the **learning rate (lr)** was evaluated across $\text{lr} = 10^{-5}, 10^{-4}, 10^{-3}$, and 10^{-2} (Figure 4). At $\text{lr} = 10^{-5}$, the model failed to converge, likely due to parameters being trapped in local minima. At $\text{lr} = 10^{-2}$ and $\text{lr} = 10^{-3}$,

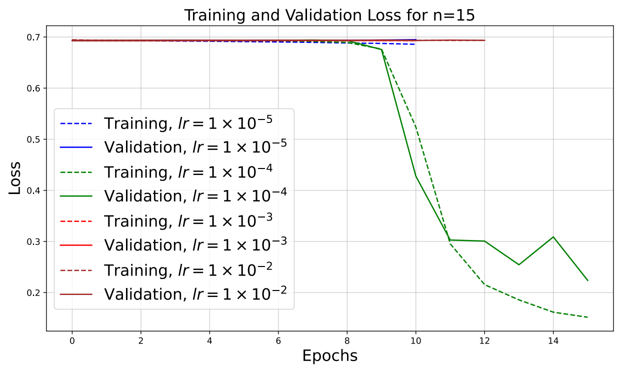

Figure 4. Loss curves of the Neural Network model using different learning rates, training on Kryptonite-15.

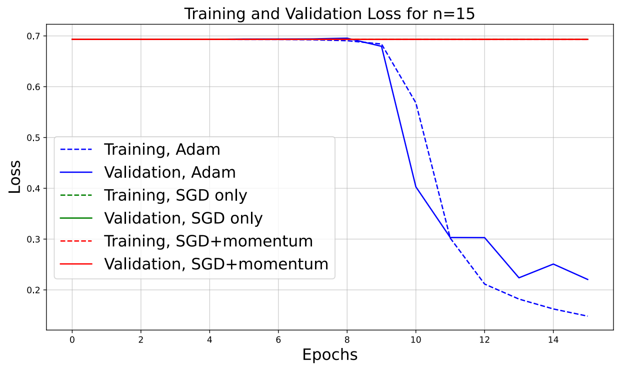

Figure 5. Comparison of loss curves of the Neural Network model when using different optimisers, training on Kryptonite-15.

convergence was hindered, potentially due to gradient vanishing caused by excessive updates. Thus, $\text{lr} = 10^{-4}$ was selected as the optimal learning rate, as it enabled convergence.

As shown in Figure 5, only Adam achieves convergence, while SGD and SGD with momentum result in stagnant losses. This is due to Adam's adaptive learning rates, which handle varying gradient scales across parameters unlike SGD-based methods.

Hyperparameter Comparison Hyperparameter selection was analysed on Kryptonite-15 $(N = 15)$. For $N = 9, 12, 15$, models used the Adam optimiser with $\text{lr} = 10^{-3}$, eight hidden layers, and ReLU activations. For $N = 18$, Adam with weight decay was employed, with $\text{lr} = 10^{-3}$, six hidden layers, and GeLU activations. We chose GeLU over ReLU due to its smooth gradient approximation, enabling easier convergence and avoiding issues like dying neurons.

Further weight perturbation analysis was performed on two identical neural networks trained on $N = 18$, one with ReLU and the other with GeLU activation functions in the hidden layer. (Appendix D) shows that GeLU had a smoother loss landscape compared to ReLU (Figures 9 and 7). This smoothness allows for more stable optimisation, making GeLU our choice for $N = 18$.

5. Sustainability

For each Kryptonite-N dataset, we evaluated approximately 10 neural network models using an NVIDIA A100 PCIe (40GB) GPU for training and testing. On average, each training session required 5 minutes, with a total of approximately 1 hour spent on hyperparameter optimisation. According to the CO_2 emission calculator proposed in (Lacoste et al., 2019), this process emitted an estimated 0.14 kg of CO_2, as calculated in Eq. (17).

$$250\,\mathrm{W} \times 1\,\mathrm{h} = 0.25\,\mathrm{kWh} \times 0.56\,\frac{\mathrm{kg\ CO_2}}{\mathrm{kWh}} = 0.14\,\mathrm{kg\ CO_2} \tag{17}$$

To mitigate the environmental impact, we ensured the use of energy-efficient hardware and conducted experiments during off-peak hours. Additionally, we minimised unnecessary training iterations by employing efficient hyperparameter optimisation techniques, such as grid search with early stopping and random search, reducing overall computational demands.

6. Discussion and Future Work

Further Mutual Information Analysis We analysed the mutual information (MI) between features of X and y for Kryptonite-18. MI measures how well a feature X_i predicts y, defined as:

$$I(X_i; y) = \sum_{x_i \in X_i} \sum_{y \in Y} p(x_i, y) \log\left(\frac{p(x_i, y)}{p(x_i)p(y)}\right),$$

where $p(x_i, y)$ is the joint probability, and $p(x_i)$, $p(y)$ are the marginals.

Figure 6 shows only 10 features of X had nonzero MI with y. Removing the 8 features with zero MI with Y caused our NN to completely fail at learning despite being able to handle all 18 features. We can conclude from this that the target y depends on combinatorial feature interactions, where combinations of features, rather than individual features, drive predictions for y. This explains why features with zero MI still contribute indirectly through interactions. We also observed similar behaviour for Kryptonite-9. Given that all Kryptonite-N datasets share the same family of the data generation function, this behaviour is expected for other dimensions as well. We attach more information for $N = 9$ in Appendix E.

Given that Kryptonite-N scales up to 45 dimensions, feature interactions create an exponentially larger search space and a more complex loss landscape, making loss function optimisation (of binary cross-entropy loss, which is already difficultly non-convex as mentioned by (Rudin, 2012)) more challenging to achieve. As such, our limitations of training time and computational requirements did not allow us to train models that could predict $N > 18$ at the target accuracy. Identifying feature combinations (in groups of more than two) is an NP-hard problem as mentioned by (Chen et al., 1997), with no polynomial-time solutions. As such, the issues with learning Kryptonite-N faced by (Quinn & Luther, 2024) is more likely an improper setup and difficulties in optimising rather than the failure of the UAT.

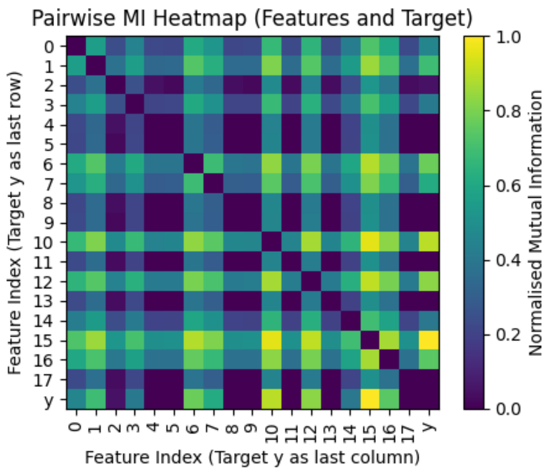

Figure 6. Mutual Information between Features and Target for Kryptonite-18. Only 10 features show nonzero MI with y.

Conclusion This project has proven that the Kryptonite-N dataset can be accurately approximated by machine learning models for N up to 18, and empirically disproves the claims of (Quinn & Luther, 2024). We also argue with evidence that the failure to learn Kryptonite-N is an issue with optimisation, not the failure of the UAT.

Future Work As the dimensionality N increased, tuning neural network hyperparameters, such as the number of hidden layers, became increasingly challenging due to the risk of overfitting. The training time and computational requirements for higher-dimensional datasets grew exponentially. With access to more powerful GPUs and extended time, further exploration of hyperparameter configurations for $N > 18$ could be conducted. Additionally, future work could investigate alternative architectures, including residual connections, convolutional layers, and transformers, given their success in other applications.

References

Berwick, R. An idiot's guide to support vector machines (svms). Technical report, Massachusetts Institute of Technology (MIT), 2011. URL https://web.mit.edu/6.034/wwwbob/svm.pdf. Village Idiot series.

Chen, B., Hong, J., and Wang, Y. The minimum feature subset selection problem. *Journal of Computer Science and Technology*, 12:145–153, 1997. doi: 10.1007/BF02951333. URL https://doi.org/10.1007/BF02951333.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. *KDD*, 2016.

Kingma, D. P. and Ba, J. L. Adam: A method for stochastic optimization. *ICLR*, 2015.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Quinn, H. and Luther, L. *Kryptonite-n: A Simple End to Machine Learning Hype?* PhD thesis, Wayne University of Metropolis, Gotham, US, 2024.

Rudin, C. Cross entropy equals logistic loss for binary classification. *Course Notes*, 2012. URL https://users.cs.duke.edu/~cynthia/CourseNotes/CrossEnt.pdf.

Shanthi, S. A. and Sathiyapriya, G. Universal approximation theorem for a radial basis function fuzzy neural network. *Materials Today: Proceedings*, 2022.

Thurnhofer-Hemsi, K., López-Rubio, E., Molina-Cabello, M. A., and Najarian, K. Radial basis function kernel optimization for support vector machine classifiers. *Preprint*, 2020.

Wicker, M. *Mathematics for Machine Learning Lecture Notes*, chapter 6. Imperial College Department of Computing, 2024.

A. Gaussian Error Function

The error function, denoted as $\text{erf}(x)$, is defined as:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt. \tag{18}$$

It represents the probability of a value falling within a certain range in a normal (Gaussian) distribution, specifically from 0 to x. Incorporating it into the GeLU activation allows introduces smoothness, that ReLU lacks, using the error function to adjust each activation gradually based on its magnitude.

B. Regularisation of XGBoost

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2. \tag{19}$$

- T: Number of leaves in the tree.
- w_j: Weight of leaf j.
- γ: Regularisation parameter for tree complexity.
- λ: L2 regularisation term on leaf weights.

C. Comparison of hyperparameters for SVMs using RBF kernels

Table 5 and 6 present the test accuracy of SVMs trained with RBF kernels using various parameter combinations of C and γ. For the Kryptonite-9 dataset, the parameter combination $C = 1$, $\gamma = 1$ achieves the highest test accuracy, whereas for Kryptonite-12 dataset, the best performance was observed with $C = 1$, $\gamma = 10$.

C	γ	Test Accuracy
0.1	0.001	0.4981
0.1	0.01	0.4981
0.1	0.1	0.4981
0.1	1	0.6998
1	0.001	0.4981
1	0.01	0.4981
1	0.1	0.5150
1	1	**0.9580**
10	0.001	0.4981
10	0.01	0.4883
10	0.1	0.5380
10	1	0.9576

Table 5. Test set accuracy for 20% randomly sampled data from Kryptonite-9 using RBF kernels with different values of C and γ. Sorted by increasing C and γ.

D. Comparing GeLU vs ReLU: 2D Plot of Weight Perturbation for NNs trained on $N = 18$

Two identical neural networks models were trained on the same splits of Kryptonite-18 for all 50 epochs. The only difference between the models was that GeLU was used for activation functions of the hidden layers in the first model (used for our submission) and ReLU was used for the other. We note that while both models achieved the target accuracies for Kryptonite-18, the loss function decreased more steadily for GeLU (Figure 8) than ReLU (Figure 10), and the training and test loss were closer in value.

C	γ	**Test Accuracy**
0.1	0.001	0.5057
0.1	0.01	0.5057
0.1	0.1	0.5057
0.1	1	0.5057
0.1	10	0.5057
1	0.001	0.5057
1	0.01	0.5057
1	0.1	0.5017
1	1	0.6312
1	10	**0.9426**
10	0.001	0.5057
10	0.01	0.5050
10	0.1	0.4988
10	1	0.8799
10	10	0.9260

Table 6. Test set accuracy for 20% randomly sampled data from Kryptonite-12 using RBF kernels with different values of C and γ.

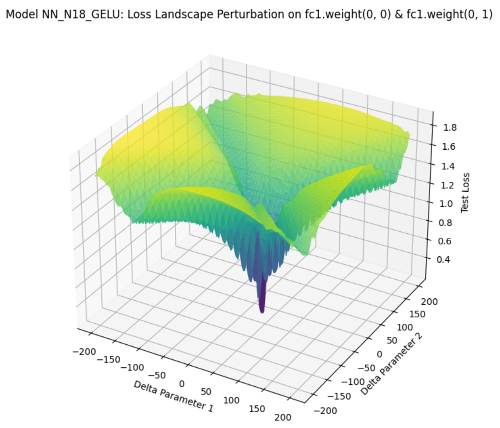

Figure 7. Loss landscape for perturbing two weights on the first layer of a neural network trained on $N = 18$. The hidden layers had GeLU activation functions.

E. Learning with Nonzero MI Features

We trained a NN with GeLU on features in X with features having zero MI with Y removed. The model is identical to the one shown in Figure 8 and can be compared directly.

Similarly to Kryptonite-18, we also performed MI analysis on Kryptonite-9, with the pairwise heatmap shown in Figure 11. It was observed that if the features in X having zero MI with Y were removed, learning also failed as seen in Figure 13. We used the identical neural network setups similar to our best model that predicted our submission for Kryptonite-9, with loss curves visualised in Figure 12, illustrating the case where our model successfully learns to predict Kryptonite-9.

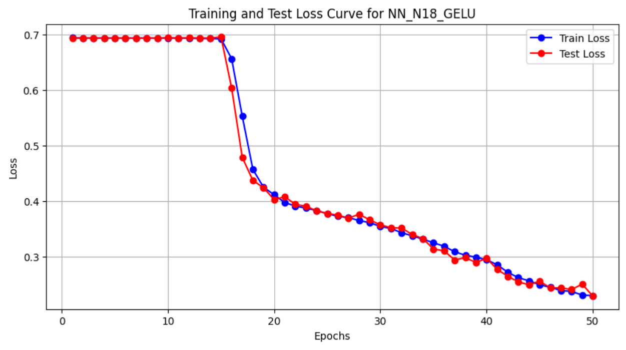

Figure 8. Training and test loss over epochs for the neural network with GeLU, predicting $N = 18$. Note how the training and validation loss are closer to each other than ReLU, and the rate of decrease appears more constant than ReLU.

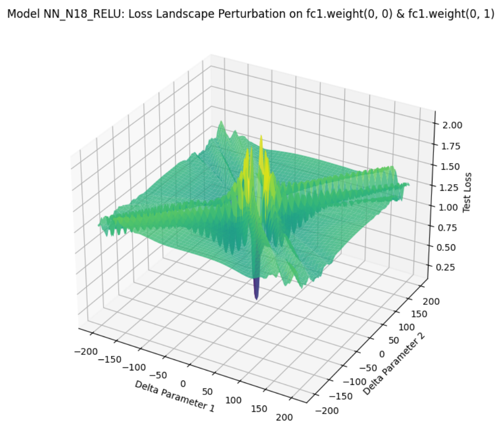

Figure 9. Loss landscape for perturbing two weights on the first layer of a neural network trained on $N = 18$. The hidden layers had ReLU activation functions. Note the increased jagged edges as compared to the GeLU loss landscape in Figure 8

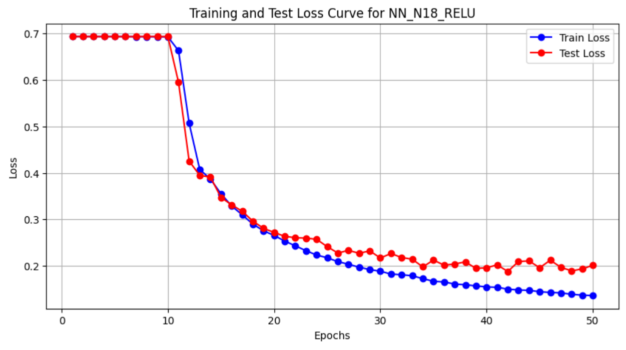

Figure 10. Training and test loss over epochs for the neural network with ReLU, predicting $N = 18$.

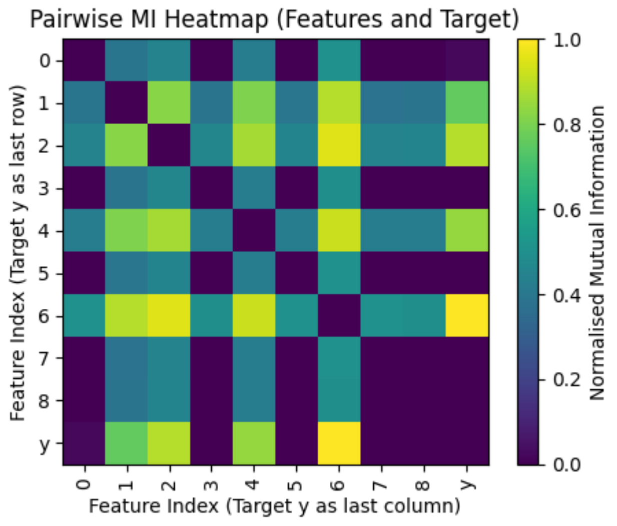

Figure 11. Mutual Information between Features and Target for Kryptonite-9. Only 4 of the 9 features show nonzero MI with y.

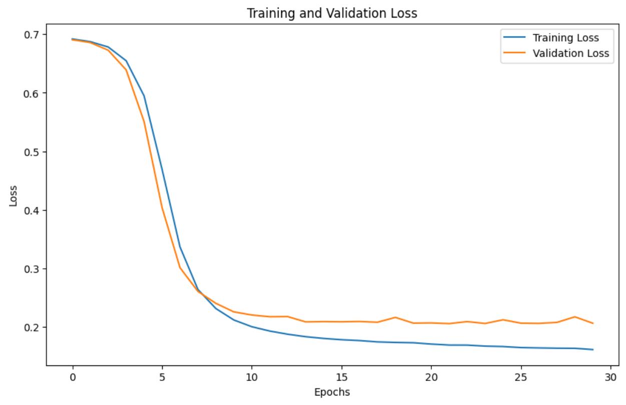

Figure 12. Loss curves for the best-performing NN for predicting Kryptonite-9

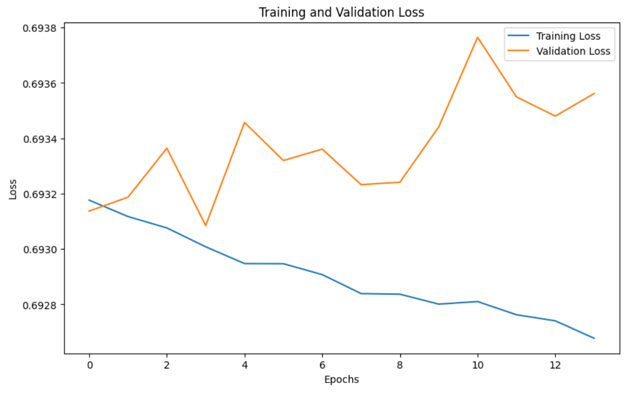

Figure 13. Loss curves for the best-performing NN for predicting Kryptonite-9, with features having zero MI with Y removed. Note how validation loss does not reduce.

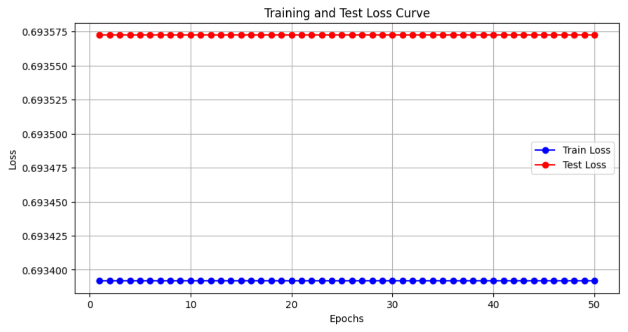

Figure 14. Training and test loss over epochs for a GeLU neural network with ReLU for $N = 18$, with Kryptonite-N features in X with features having zero MI with Y removed. To be compared directly with Figure 8. Note that learning completely fails on the data given no change in loss.