Resolving Kryptonite-n: How Model Design Matters

Javier Paez Franco [1] **Fabian Alexander Bermana** [1] **Miguel Suriol Alfonso** [1] **Duong Ha** [1]

Abstract

Recent advances in machine learning have led to debates about the limitations of these models. This work refutes the claims of *Kryptonite-n: A Simple End to Machine Learning Hype?*, which challenges the Universal Function Approximation argument. By identifying theoretical flaws and providing empirical evidence, we demonstrate that Machine Learning models can achieve high efficiency on the Kryptonite-n datasets, surpassing the benchmarks set by the original paper.

1. Introduction

Artificial Intelligence (AI) has seen remarkable advances in recent years, particularly in machine learning. The rapid progress and increasing capabilities of AI systems have led to a level of hype and anticipation not seen since the early days of the internet (Makridakis, 2017). These advances have led to intense debate about the fundamental capabilities and limitations of machine learning models. While some researchers believe we will achieve Artificial General Intelligence (AGI) sooner than later (Wong, 2024), others are highly sceptical and believe we are still far away from reaching AGI (Marcus, 2024; Voss & Jovanovic, 2023; LeCun, 2024).

Central to this AI debate is the Universal Approximation Theorem (UAT), which suggests that neural networks can approximate any continuous function (Hornik, 1991). Recently, Dr Harley Quinn and Dr Lex Luther, two AI researchers aligned with the views of AI sceptics, published "Kryptonite-n: A Simple End to Machine Learning Hype?" (Quinn & Luther, 2024). This paper claims to defeat the strongest theoretical arguments supporting AI.

In this paper, we will refute their claims and introduce several models that achieve accuracies previously considered unattainable. Consequently, in this paper, we aim to answer the research question:

Are the Kryptonite-n claims theoretically valid, and how do machine learning models compare against its accuracy goals?

This paper's contributions are:

- We provide a rigorous theoretical analysis of the Kryptonite-n paper's claims, identifying key misinterpretations of the UAT.
- We present several Machine Learning (ML) models that achieve high performance on the Kryptonite-n dataset, surpassing the accuracy targets. We offer a comparative analysis between them and with the baseline regression model.
- We provide a comprehensive environmental impact assessment of our experimental pipeline.

We begin by analysing the Kryptonite-n paper and its theoretically invalid claims in Section 2. Section 3 introduces the ML models, and Section 4 details the experimental setup. The results are presented in Section 5. Then, Section 6 discusses how our models perform and provides an environmental impact assessment. Finally, Section 7 concludes the paper by summarising the findings and discussing potential directions for future research.

2. Analysis of Kryptonite-n's original paper

The authors of the paper "Kryptonite-n: A Simple End to Machine Learning Hype?" argue that logistic regression with basis expansion performing poorly on their dataset demonstrates that ML models are not as universally capable as often claimed, challenging the Universal Approximation Theorem (UAT). We present a formal analysis of their theoretical claims, examining the fundamental flaws in their theoretical framework and interpretation.

The paper's central argument misinterprets the Universal Approximation Theorem (UAT). Let us first state the theorem formally (Cybenko, 1989; Hornik, 1991):

[1]Department of Computing, Imperial College London, London, United Kingdom. Correspondence to: Javier Paez Franco <javier.paez-franco24@imperial.ac.uk>.

Coursework. For review by instructors of Mathematics for Machine Learning (70015) . Do not distribute.

Theorem 1 (Universal Approximation Theorem)**.** *Let $\sigma(\cdot)$ be a bounded, non-constant, and continuous function. Let I_m denote the m-dimensional unit cube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, for any function $f \in C(I_m)$ and $\epsilon > 0$, there exists an integer N, real constants $v_i, b_i \in \mathbb{R}$, and real vectors $w_i \in \mathbb{R}^m$ for $i = 1, \ldots, N$, such that:*

$$F(x) = \sum_{i=1}^{N} v_i \sigma(w_i^T x + b_i) \tag{1}$$

as an approximate realisation of the function f; that is,

$$|F(x) - f(x)| < \epsilon, \quad \forall x \in I_m \tag{2}$$

The UAT is an existence theorem, not a constructive one. It guarantees the existence of a neural network with a single hidden layer capable of approximating any continuous function, given a sufficient number of hidden neurons. However, it does not provide a method for finding such a network. Formally, while it ensures $\exists N, \{v_i, w_i, b_i\}_{i=1}^N$ that achieve ϵ-approximation, it makes no claims about the learning process finding these parameters.

Furthermore, the authors acknowledge that their chosen basis functions might not fully capture the target function, suggesting that further investigation into different basis expansions could provide clearer results. This admission directly defeats their theoretical arguments.

In addition, the experimental design is flawed, particularly due to the lack of parameter tuning. Hyperparameters critically influence training speed and accuracy, hence they must be carefully configured before the training begins (Yu & Zhu, 2020). The lack of convergence or low accuracy may stem from the insufficient tuning of parameters.

To sum up, a correct interpretation of these results would be much more modest:

1. The authors have constructed a challenging dataset that certain models and training techniques struggle with.
2. This might point to interesting limitations of particular architectures or optimisation methods.
3. The dataset could serve as a useful benchmark to develop better training techniques or model designs.

Our work aims to provide a grounded analysis, examining the performance of different models on the Kryptonite dataset alongside a robust experimental design.

3. Methodology

This section outlines the methodology proposed in this study. We discuss the models we implement: neural networks, random forest, logistic regression with basis expansion, and Gaussian mixture model. These models were selected for two key reasons: (i) they were suggested in the future work section of the Kryptonite paper, and (ii) they allow for a comparison between models with fundamentally different architectures and underlying learning principles. This diverse selection includes, for instance, both supervised and unsupervised learning approaches, providing a broad evaluation spectrum.

We consider our task a supervised learning binary classification problem, with a labelled dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$, with input features $\mathbf{X} \in \mathbb{R}^{N \times n}$ and corresponding labels $\mathbf{y} \in \{0,1\}^N$. We denote a singular input by $\boldsymbol{x} \in \mathbb{R}^n$.

3.1. Neural Networks

A singular neuron is defined by the equation $\eta_i(x) = \sigma(\boldsymbol{W}_{i:}\boldsymbol{x} + b)$, where $\eta_i \in \mathbb{R}$ is the output of the neuron i, $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of inputs, σ is a non-linear activation function (see Appendix 8.1 for why this is), in our case the ReLU activation function, $\boldsymbol{W}_{i:} \in \mathbb{R}^{1 \times n}$ represents a vector of weights, and $b \in \mathbb{R}$ is a bias term. In a neural network (NN), neurons are arranged in layers, with each layer's outputs serving as inputs to the subsequent layer. Mathematically, a two-layer network is described by:

$$\boldsymbol{z}^{(1)} = \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x} + b^{(1)}) \tag{3}$$

$$\boldsymbol{z}^{(2)} = \sigma(\boldsymbol{W}^{(2)}\boldsymbol{z}^{(1)} + b^{(2)}) \tag{4}$$

$$\boldsymbol{y} = g^{-1}(\theta^\top \boldsymbol{z}^{(2)}) \tag{5}$$

Training consist of finding the parameters that minimise the loss function $\mathcal{L}(\boldsymbol{y}, \boldsymbol{z}^{(2)}) = (\boldsymbol{y} - \boldsymbol{z}^{(2)})^2$ using gradient descent. For this purpose, we employ the Adam optimiser (Kingma, 2014). Unlike SGD, Adam uses adaptive learning rates. It uses Root Mean Square Propagation (RMSProp) to adapt the learning rate based on a moving average of gradients, incorporates momentum as an estimate of the first-order moment of the gradient, and introduces bias corrections to the first and second-order moments estimates (Goodfellow et al., 2016). The equations for Adam are provided in Appendix 8.3.

3.2. Tree-based Approaches

A decision tree $\mathcal{T}$ is a directed acyclic graph having at most one path between every pair of nodes. All nodes have one incoming edge, except for the root. Non-terminal nodes are labelled with an input feature $\boldsymbol{x}$, and terminal nodes with a label $\mathbf{y}$. The arcs coming from a node labelled with an input feature are labelled with each of the possible values of the target feature, or the arc leads to a subordinate decision node on a different input feature (Izza et al., 2020). We construct the trees by recursively selecting splits that maximise quality based on the Gini impurity: $H(\mathcal{D}_m) = \sum_k p_{mk}(1 - p_{mk})$,

where $\mathcal{D}_m$ is the data at node m and p_{mk} is the proportion of class k observations in node m (Scikit-learn, 2024c).

A random forest (RF) is an ensemble of decision tree classifiers that use averages to improve the predictive accuracy and control over-fitting (Scikit-learn, 2024b). Decision trees are ideal candidates for ensemble methods since they usually have low bias and high variance, making them likely to benefit from the averaging process (Louppe, 2015). We follow the Scikit-learn implementation, where the classifiers are combined by averaging their probabilistic prediction (Scikit-learn, 2024a).

3.3. Binary Logistic Regression

Logistic regression is a powerful and established method for supervised classification (Hosmer Jr. et al., 2013). The gradient of the loss function with respect to the parameter vector $\theta \in \mathbb{R}^n$ in its vectorised form is:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbf{X}^\top (\sigma(\mathbf{X}\boldsymbol{\theta}) - y) \tag{6}$$

Where $\sigma(\mathbf{X}\boldsymbol{\theta})$ denotes the element-wise application of the sigmoid function across the predictions for each sample. Since there is no closed-form solution, by iteratively updating $\boldsymbol{\theta}$, the model converges to parameter values that improve prediction accuracy over the training data.

While logistic regression performs well on linearly separable data, the datasets present more complex structures. To address this, we adopt polynomial basis expansion, following the implementation of the Kryptonite paper.

3.4. Gaussian Mixture Model Clustering

A Gaussian Mixture Model (GMM) is a probabilistic model where input data is considered to be generated from a mixture of K Gaussian distributions, each is identified by $k \in 1, 2, ..., K$ and comprised of these parameters: mean μ_k, covariance Σ_k, and fixing probability π_k, where $\sum_{k=1}^{K} \pi_k = 1$.

The distribution of a data-point $x \in \mathbb{R}^n$ is defined as below:

$$p(x \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{n}{2}} |\boldsymbol{\Sigma}|^{\frac{1}{2}}} \exp(-\frac{1}{2}(x-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})) \tag{7}$$

Where:

$$\boldsymbol{\mu} = \sum_{k=1}^{K} \pi_k \mu_k \tag{8}$$

$$\boldsymbol{\Sigma} = \sum_{k=1}^{K} \mu_k (\Sigma_k + (\mu_k - \boldsymbol{\mu})(\mu_k - \boldsymbol{\mu})^T) \tag{9}$$

In the context of binary classification task, the training dataset is divided into two subsets based on their labels ($y = 0$ and $y = 1$), and respectively, two separate GMMs are trained on each subset: G_0 and G_1.

For a testing data-point x, each model with generate its log-likelihood as L_0 and L_1. The predicted label for x is determined by comparing the two log-likelihoods:

$$\hat{y} = \begin{cases} 1 & \text{if} \quad L_1(x) > L_0(x) \\ 0 & \text{otherwise} \end{cases}$$

4. Experimental Design

In this section, we present a comprehensive empirical evaluation of the Kryptonite-n task. First, we introduce the hypothesis we seek to test. Then, we detail the experimental framework and how we will evaluate the models' performance.

4.1. Problem Definition

The primary hypothesis for this response paper is challenging the claims made in the Kryptonite-n paper. Specifically, we hypothesize machine learning models can achieve higher accuracy than the thresholds set in the original paper. To test this hypothesis, several independent variables are used: the model architecture types (Neural Networks, Random Forest, Gaussian Mixture Models, and Logistic Regression as baseline measure) and dataset dimensionality ($n = 9$, 12, 15, and 18. Dimensions above 18 were looked at, but not achieved).

Our dependent variables consist of multiple performance metrics: accuracy, precision, recall, F1 score, and AUC-ROC values. We control several variables, including data preprocessing such as standardisation, PCA, and denoising for Neural Networks, as well as training parameters (data split ratios, fixed hyperparameters for each model type, and 4-fold cross-validation). This experimental design tests our hypothesis through multiple model implementations, statistical validation using McNemar's test, and direct comparison against the original paper's accuracy thresholds.

4.2. Data Pre-processing

An initial look at the dataset shows us that the features in all the dimensions of the Kryptonite-n dataset follow a Bernoulli distribution with some Gaussian noise. This visualization can be better seen in Figure 1.

Several pre-processing methods have been implemented. For example, random forest uses PCA to lower the dimensionality of our data, and neural Networks employ a simple de-noise method to remove the Gaussian noise, transforming the distribution into purely Bernoulli. This was simple, as there was no overlap between the feature values, which resulted in a simple rounding function. It helped greatly to the convergence of our models and increasing the accuracy. Figures of the losses and accuracies are shown in Appendix 8.6.

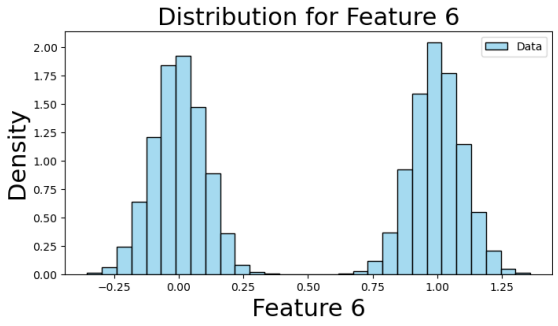

Figure 1. Distribution visualization of Feature 5

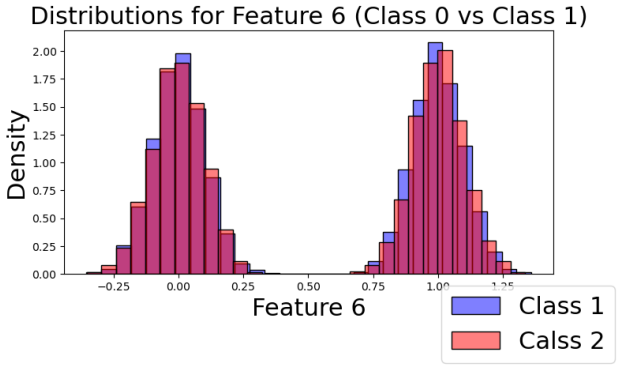

Figure 2. Distribution visualization of Feature 5 showing the different class distributions.

It is worth noting that just because the distributions follow a Bernoulli distribution with Gaussian noise, the data is not linearly separable. This is shown in Figure 2, where the two class distributions almost perfectly match.

4.3. Hyperparameter tuning

Optimising the hyperparameters of a deep learning model is crucial for training success. They control the behaviour of the algorithms and determine their performance. Instead of setting them manually, we will use Bayesian optimisation based on kernel fitting through the Tree-structured Parzen Estimator (TPE) algorithm (Watanabe, 2023), as well as a pruning strategy based on the median stopping rule. To ensure correct implementation, we employ the Optuna framework (Akiba et al., 2019).

4.4. Result validation

The validation methodology consist of two phases: parameter tuning and model evaluation. We first split the data into 80% training and 20% test sets, solely for parameter tuning. The training set is further divided into 80% training and 20% validation set. We optimise the model architecture and hyperparameters based on the validation set. Afterwards, we train the final model on the complete training set and evaluate it on the held-out test data. This methodology allows us to optimise the hyperparameters and evaluate model performance on data never utilised before. Finally, to ensure robustness, we conduct a 4-fold cross-validation on the full dataset. The final reported results represent the average over these test folds.

After choosing our best models and hyperparameters, we produce predictions on the unlabelled datasets by (Quinn & Luther, 2024), as evidence of our models' generalisation performance on unseen data. For this purpose, we re-train our best models on the whole dataset for each of the dimensions in order to utilise all the available data.

We evaluate model performance through several metrics: accuracy, recall, F1-score, and precision, to comprehensively assess the models' quality and reliability. We also use the Area Under the Receiver Operating Characteristic curve (AUC-ROC) to compare the separability capacity of the models (Sokolova & Lapalme, 2009).

To establish statistical significance in model comparisons, we employ the McNemar's test (McNemar, 1947). For two models with accuracies acc_1 and acc_2, we test the null hypothesis $H_0 : acc_1 = acc_2$, i.e. both models have the same accuracy, against the alternative hypothesis $H_1 : acc_1 \neq acc_2$. Let a be the number of examples correctly classified by both models, b the number of correctly classified examples by model 1 but not model 2, and c the number correctly classified by model 2 but not model 1. The McNemar test statistic is $\chi^2 = \frac{(b-c)^2}{(b+c)}$. If the χ^2 result is significant, there is enough evidence to reject H_0 in favour of H_1. We use a significance level of 5%.

According to (Dietterich, 1998), McNemar's test has low type I error and, as such, is better for model comparison compared to commonly used methods, such as paired-differences t-test. The test compares the best-performing model against the baseline logistic regression from (Quinn & Luther, 2024) for each dataset, enabling rigorous statistical validation of performance improvements.

To ensure a correct and bug-free implementation, we implemented the models using Scikit-learn (Pedregosa et al., 2011) and PyTorch (Paszke et al., 2019).

5. Experimental Results

In this section, we present the experimental results for all learning algorithms applied to the Kryptonite datasets. The implementation details of the ML models, including the complete list of tuned hyperparameters, are provided in Appendix 8.4 for reproducibility and reference.

Algorithm	Accuracy				Precision				Recall				F1 Score			
	9	12	15	18	9	12	15	18	9	12	15	18	9	12	15	18
NN	**0.958**	**0.965**	**0.96**	**0.929**	**0.959**	**0.966**	**0.96**	**0.903**	**0.956**	**0.965**	**0.96**	**0.963**	**0.957**	**0.966**	**0.96**	**0.932**
GMM	0.957	0.498	0.502	0.504	0.956	0.503	0.504	0.511	0.957	0.394	0.490	0.587	0.957	0.370	0.466	0.468
RF	0.952	0.866	0.595	0.510	0.952	0.866	0.595	0.510	0.952	0.866	0.596	0.510	0.952	0.866	0.595	0.509
LR	0.490	0.511	0.493	0.495	0.487	0.513	0.491	0.500	0.465	0.480	0.493	0.492	0.476	0.496	0.492	0.496

Table 1. Performance metrics comparison between NN, GMM, RF, and original LR across different dimensions. All the values have a maximum standard deviation of 0.05. Bolded values represent the best for the dataset with dimensionality n.

n	9	12	15	18
NN	0.000	0.000	0.000	0.000
RF	0.000	0.000	0.221	0.985
GMM	0.000	0.000	0.490	0.449

Table 2. McNemar Test p-values vs Baseline Binary Logistic Regression, for dimensionalities 9 through 18.

Table 2 present the McNemar Test p-values of our models vs the baseline logistic regression. We observe our Neural Network model is able to produce predictions that are statistically significantly different from the baseline logistic regression. Additionally, for the lower-dimensionality datasets ($n = 9, 12$), the other models present results significantly different to the baseline, although they are not able to achieve so on the higher-dimensionality datasets.

Table 1 presents the accuracy, precision, recall, and F1 scores for all models, obtained through hyperparameter tuning and 4-fold cross-validation. Table 3 shows the AUC values of the models, and the corresponding ROC curves are provided in Appendix subsection 8.5. The Neural Network consistently achieves the highest performance across all metrics. Random Forest and Gaussian Mixture Models show strong performance at lower dimensions, with AUC ≈ 0.96, though their effectiveness diminishes significantly as dimensionality increases. Logistic Regression consistently shows the lowest results, with a performance similar to random chance.

n	9	12	15	18
NN	0.97	0.97	0.97	0.97
RF	0.96	0.96	0.52	0.51
GMM	0.96	0.60	0.51	0.50
LR	0.50	0.51	0.48	0.50

Table 3. AUC values for Neural Networks (NN), Random Forests (RF), Gaussian Mixture Models (GMM), and Logistic Regression (LR), for dimensions 9 through 18.

6. Discussion

Our experimental results demonstrate that machine learning approaches can successfully tackle the Kryptonite-n dataset, contradicting the central claims of Quinn and Luther's original paper. In this section, we analyse these findings and discuss their broader implications.

6.1. Analysis of model performance

The experimental results demonstrate that Neural Networks (NN) not only outperform all other implemented models across all dataset dimensionalities, as shown in Table 1 and Table 3, but also surpasses the accuracy targets of the Kryptonite-n paper (up to $n = 18$). This superior performance can be attributed to the NN's ability to effectively model non-linear relationships and handle high-dimensional data. Random Forests (RF) and Gaussian Mixture Models (GMM) exhibit strong performance in lower-dimensional datasets, achieving accuracies exceeding 90% for $n = 9, 12$ and $n = 9$, respectively. However, their performance rapidly declines as dimensionality increases, indicating limitations in scalability and adaptability to complex and non-linear data distributions. On the other hand, the baseline model, Logistic Regression (LR), consistently underperforms, with accuracy levels comparable to random chance even at the lowest dimensionality ($n = 9$).

The results of our McNemar test in Table 2 confirm our models' predictions are statistically significantly different from the baseline, logistic regression. Coupled with higher accuracy values, this provides evidence that all of our models beat the baseline at least for $n = 9$ and $n = 12$, with p-values lower than 0.05, and in the case of the Neural Network for $n = 18$ too.

Despite our positive results, our study encountered several challenges. Our main limitation is that we have managed to reach the accuracy targets for up to Kryptonite-18, but none of our models could reach the specified targets for the datasets with higher dimensionality. Furthermore, only one of our models (NN) was able to significantly outperform LR of (Quinn & Luther, 2024) for Kryptonite-15 and above.

Another issue we found was model scalability. The computational requirements for the NN increased significantly with dataset dimensionality, which made it more difficult to run a hyperparameter search for the larger datasets. Future work could explore more efficient architectures or training approaches.

Finally, data preprocessing was not explored in detail, with the only significant preprocessing techniques seen in our

work being standardisation scaling and PCA. Potentially, non-linear methods such as Kernel PCA could have contributed to improving the performance of the models.

6.2. Environmental Impact Assessment

The environmental impact of machine learning has become an increasingly critical consideration in AI research and development. Model training incurs a substantial cost to the environment due to the energy required by hardware to run the process for a long period of time (Strubell et al., 2019). For instance, GPT3 generated $552,000$ kg of CO2 equivalent, the equivalent of 123 gasoline-powered passenger vehicles driven for one year (Saenko, 2024).

Conscious of these environmental concerns, we conducted a thorough assessment of our experimental pipeline's carbon footprint. Using the ML CO2 Impact calculator (Lacoste et al., 2019), we measured our model training emissions at 0.9 kg CO2 equivalent— a relatively modest impact achieved through deliberate design choices and sustainable practices. Our experimental setup utilised an Intel Xeon E5-2699 processor on Google Cloud Platform's west-europe-2 region, chosen specifically for its lower carbon intensity.

Our implementation incorporates several innovative strategies to optimise energy efficiency:

- We selected PyTorch over TensorFlow, based on comprehensive energy profiling that showed that PyTorch tends to be more energy efficient than TensorFlow at training stage (Georgiou et al., 2022).
- We developed a novel approach to hyperparameter optimisation using Optuna (Akiba et al., 2019), employing sensitivity analysis to identify and prioritise the most important parameters. This approach significantly reduced the number of experimental iterations required for model tuning.
- We implemented an ensemble of techniques to speed up convergence: Kaiming normalisation (He et al., 2015), Early Stopping, Adam optimiser (Kingma, 2014), and Dropout (Contributors, 2023).
- We used GPUs where possible, as it is faster and more energy-efficient than CPUs, thanks to its cores optimised for simultaneous computations (Rana, 2023).

The ethical implications of AI's environmental impact extend beyond immediate carbon emissions to questions of responsibility and fairness across individuals, nations, and future generations. The pursuit of higher performance through larger architectures and extensive hyperparameter experimentation creates an ethical tension between advancement and environmental impact. This challenge needs a shared responsibility among AI researchers and engineers, academic institutions, and industry partners to develop and implement sustainable practices (Tamburrini, 2022).

Looking forward, we identified several promising directions for further reducing the environmental impact of ML systems. Our analysis suggest that data-centric strategies, for instance intelligent dataset reduction (Verdecchia et al., 2022) and subsampling strategies to remove redundant data points (Dhabe et al., 2021), offer opportunities to decrease energy consumption without compromising model performance. Furthermore, we propose implementing GreenScale (Kim et al., 2023) for carbon-efficient scheduling, which could reduce carbon emissions by scheduling training during periods of low-carbon energy availability. These systems leverage real-time carbon intensity data in order to direct computations to regions with greener energy sources. Finally, we could greatly optimise GPU utilisation through better resource management and workload consolidation — current industry practices often achieve only 30-50% utilisation (NVIDIA, 2021).

7. Conclusion and Future Works

The surge of Artificial Intelligence (AI) in recent years has provoked a discussion of the true capabilities and limitations of these systems. The paper "Kryptonite-n: A Simple End to Machine Learning Hype?" proposes that ML is simply hype, and researchers should rather move to 'something else'. In this study, we critically analyse their claims by answering the research question: *Are the Kryptonite-n claims theoretically valid, and how do machine learning models compare against its accuracy goals?*

Our results demonstrate that their conclusions regarding the limitations of ML are unfounded due to theoretical misinterpretations. We show that ML models, particularly NNs, can surpass not only the LR model of their paper, but also their accuracy benchmarks, challenging their conclusions about ML limitations. Rigorous hyperparameter tuning and statistical evaluations ensured the reliability of our results. The environmental impact analysis emphasises the importance of sustainability in ML research, showing how informed practices can balance accuracy and resource efficiency.

Future work could explore more advanced ML models and techniques, such as t-SNE (Van Der Maaten & Hinton, 2008) or variational autoencoders (Kingma & Welling, 2022), investigate generalisability of the ML models in similar datasets, or assess the Kryptonite dataset's value as a benchmark (Xiao et al., 2022).

Acknowledgement

We express our appreciation to Matthew Wicker for providing this engaging coursework for the COMP70015 course.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. Optuna: A next-generation hyperparameter optimization framework, 2019. URL https://arxiv.org/abs/1907.10902.

Contributors, P. torch.nn.dropout, 2023. URL https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html. Accessed: 2024-11-19.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signals and Systems*, 2(4):303–314, December 1989. doi: 10.1007/bf02551274. URL https://doi.org/10.1007/bf02551274.

Dhabe, P., Mirani, P., Chugwani, R., and Gandewar, S. *Data Set Reduction to Improve Computing Efficiency and Energy Consumption in Healthcare Domain*, pp. 53–64. Springer International Publishing, Cham, 2021. ISBN 978-3-030-61089-0. doi: 10.1007/978-3-030-61089-0_4. URL https://doi.org/10.1007/978-3-030-61089-0_4.

Dietterich, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 10 1998. ISSN 0899-7667. doi: 10.1162/089976698300017197. URL https://doi.org/10.1162/089976698300017197.

Duarte, F. S., Rios, R. A., Hruschka, E. R., and de Mello, R. F. Decomposing time series into deterministic and stochastic influences: A survey. *Digital Signal Processing*, 95:102582, 2019. ISSN 1051-2004. doi: https://doi.org/10.1016/j.dsp.2019.102582. URL https://www.sciencedirect.com/science/article/pii/S1051200419301289.

Georgiou, S., Kechagia, M., and Sarro, F. Green ai: Do deep learning frameworks have different costs? *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp. 1082–1094, 2022. URL https://api.semanticscholar.org/CorpusID:247614182.

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

Guo, J. Ai notes: Regularizing neural networks - deeplearning.ai, 2024. URL https://www.deeplearning.ai/ai-notes/regularization/index.html.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. URL https://doi.org/10.48550/arXiv.1502.01852.

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

Hosmer Jr., D. W., Lemeshow, S., and X. Sturdivant, R. *Applied logistic regression*. Wiley, March 2013. doi: 10.1002/9781118548387. URL https://onlinelibrary.wiley.com/doi/book/10.1002/9781118548387.

IBM. What is the k-nearest neighbors algorithm? https://www.ibm.com/topics/knn, 2023. Accessed: 2023-11-19.

Izza, Y., Ignatiev, A., and Marques-Silva, J. On explaining decision trees, 2020. URL https://arxiv.org/abs/2010.11034.

Kim, Y. G., Gupta, U., McCrabb, A., Son, Y., Bertacco, V., Brooks, D., and Wu, C.-J. Greenscale: Carbon-aware systems for edge computing, 2023. URL https://doi.org/10.48550/arXiv.2304.00404.

Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes, 2022. URL https://arxiv.org/abs/1312.6114.

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

LeCun, Y., 2024. URL https://x.com/ylecun/status/1791890883425570823.

Louppe, G. Understanding random forests: From theory to practice, 2015. URL https://arxiv.org/abs/1407.7502.

Makridakis, S. The forthcoming artificial intelligence (ai) revolution: Its impact on society and firms. *Futures*, 90:46–60, 2017. ISSN 0016-3287. doi: https://doi.org/10.1016/j.futures.2017.03.006. URL https://www.sciencedirect.com/science/article/pii/S0016328717300046.

Marcus, G. Deep learning is hitting a wall - nautilus, March 2024. URL https://nautil.us/deep-learning-is-hitting-a-wall-238440/.

McNemar, Q. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

NVIDIA. Gpus for virtualization, 2021. NVIDIA Corporation.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Probst, P. and Boulesteix, A.-L. To tune or not to tune the number of trees in random forest. *Journal of Machine Learning Research*, 18(181):1–18, 2018.

Quinn, H. and Luther, L. Kryptonite-n: A simple end to machine learning hype? *Math for ML (70015) Coursework Prompt*, 2024. URL https://github.com/matthewwicker/Kryptonite-N/blob/main/PaperAndInstructions/Kryptonite_N_Coursework_Prompt.pdf.

Rana, D. Cpu vs gpu: Why gpus are more suited for deep learning?, 2023. URL https://www.analyticsvidhya.com/blog/2023/03/cpu-vs-gpu/#CPU_vs._GPU:_Which_is_Better_Suited_for_Machine_Learning_and_Why? Accessed: 2024-11-19.

Saenko, K. A computer scientist breaks down generative ai's hefty carbon footprint, February 2024. URL https://www.scientificamerican.com/article/a-computer-scientist-breaks-down-generative-ais-hefty-carbon-footprint/.

Scikit-learn. Ensembles: Gradient boosting, random forests, bagging, voting, stacking, 2024a. URL https://scikit-learn.org/stable/modules/ensemble.html.

Scikit-learn. Randomforestclassifier api reference, 2024b. URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Scikit-learn. Decision trees, 2024c. URL https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation.

Sokolova, M. and Lapalme, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2009.03.002. URL https://www.sciencedirect.com/science/article/pii/S0306457309000259.

Strubell, E., Ganesh, A., and McCallum, A. Energy and policy considerations for deep learning in NLP. In Korhonen, A., Traum, D., and Màrquez, L. (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3645–3650, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1355. URL https://aclanthology.org/P19-1355.

Tamburrini, G. The ai carbon footprint and responsibilities of ai scientists. *Philosophies*, 2022. URL https://api.semanticscholar.org/CorpusID:245870265.

Van Der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, January 2008. URL http://isplab.tudelft.nl/sites/default/files/vandermaaten08a.pdf.

Verdecchia, R., Cruz, L., Sallou, J., Lin, M., Wickenden, J., and Hotellier, E. Data-centric green ai an exploratory empirical study. In *2022 International Conference on ICT for Sustainability (ICT4S)*, pp. 35–45, 2022. doi: 10.1109/ICT4S55073.2022.00015.

Voss, P. and Jovanovic, M. Why we don't have agi yet, 2023. URL https://arxiv.org/abs/2308.03598.

Watanabe, S. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023. URL https://arxiv.org/abs/2304.11127.

Wong, M. The ai boom has an expiration date. *The Atlantic*, October 2024. URL https://www.theatlantic.com/technology/archive/2024/10/agi-predictions/680280/.

Xiao, Y., Fu, J., Ng, S.-K., and Liu, P. Are all the datasets in benchmark necessary? a pilot study of dataset evaluation for text classification, 2022. URL https://arxiv.org/abs/2205.02129.

Yu, T. and Zhu, H. Hyper-parameter optimization: A review of algorithms and applications, 2020. URL https://arxiv.org/abs/2003.05689.

8. Appendices

8.1. Appendix A: Proof of Non-linearity Needed for Activation Function

8.1.1. Linear Transformation Problem

Consider a neural network layer with input $\mathbf{x}$, weights $\mathbf{W}$, and bias $\mathbf{b}$. The pre-activation output of the layer is given by:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{10}$$

If the activation function $f(\cdot)$ is linear (e.g., $f(x) = x$), then the output of the layer becomes:

$$\mathbf{y} = f(\mathbf{z}) = \mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{11}$$

Now, consider a multi-layer neural network with n layers. The output of the entire network can be expressed as:

$$\mathbf{y} = f_n(\mathbf{W}_n \dots (f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)) \cdots + \mathbf{b}_n)) \tag{12}$$

If all activation functions $f_i(\cdot)$ are linear, the expression simplifies using the associative property of matrix multiplication:

$$\mathbf{y} = \mathbf{W}_n \cdots \mathbf{W}_1\mathbf{x} + (\mathbf{W}_n \cdots \mathbf{W}_2)\mathbf{b}_1 + \ldots + \mathbf{b}_n \tag{13}$$

This is equivalent to a single linear transformation:

$$\mathbf{y} = \mathbf{W}_{\text{effective}}\mathbf{x} + \mathbf{b}_{\text{effective}} \tag{14}$$

where:

$$\mathbf{W}_{\text{effective}} = \mathbf{W}_n\mathbf{W}_{n-1} \cdots \mathbf{W}_1, \tag{15}$$

And $\mathbf{b}_{\text{effective}}$ is a linear combination of the biases.

Thus, the entire network reduces to a single linear model, regardless of the number of layers.

8.1.2. Non-linear Activation

Let $f(\cdot)$ be a non-linear activation function (e.g., ReLU, sigmoid, tanh). The composition of a linear function $g(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$ and a non-linear function $f(\cdot)$ results in:

$$h(\mathbf{x}) = f(g(\mathbf{x})) = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{16}$$

Which is generally a non-linear function. This non-linearity allows the neural network to model complex, non-linear mappings from input to output.

8.2. K-NN

K-NN is a non-parametric, instance-based, supervised learning classifier that defines some notion of distance $d(x_1, x_2)$ to make predictions about the class of a data point (IBM, 2023). For a given query point $\mathbf{x} \in \mathbb{R}^n$, the K-NN algorithm identifies the subset $\mathcal{N}_k(\mathbf{x})$ of the dataset that contains the k closest data points to $\mathbf{x}$, using the distance function d. The classification is then made via a majority vote of the k nearest instances. Formally, the predicted class $\hat{y}$ for the query point $\mathbf{x}$ can be represented as:

$$\hat{y} = \text{mode}\left\{y_i \mid (\mathbf{x}_i, y_i) \in \mathcal{N}_k(\mathbf{x})\right\}$$

where mode returns the most frequent class label among the k nearest neighbours.

For the distance function, $d(\mathbf{x}_i, \mathbf{x}_j)$ a Euclidean definition is most commonly used. Other metrics such as Manhattan, Minkowski, or Hamming distance can also be employed, depending on the nature of the input data:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt[p]{\sum_{d=1}^{D} |x_{id} - x_{jd}|^p}$$

where D is the dimensionality of the feature space, x_{id} and x_{jd} are the d-th features of the instances $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively, and p defines the order of the norm used in the Minkowski metric (Duarte et al., 2019). For $p = 2$, it becomes Euclidean distance, and for $p = 1$ is Manhattan distance.

8.3. Adam

The Adam optimiser is defined by the following equations:

1. **Initialisation**. The first and second moment vectors, $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$, are initialised as zero vectors.
2. **Update Rules**. For each parameter $\boldsymbol{\theta}$, at iteration t, the following updates are applied:
   - Compute the gradient of the loss function with respect to parameters:
   $$\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}_t$$
   - Update the biased first and second moment estimates:
   $$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \boldsymbol{v}_t = \beta_2 v_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$$
   - Apply bias correction to the moment estimates:
   $$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t}, \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t}$$
   - Update the parameters:
   $$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon}$$
3. **Hyperparameters**:
   - β_1: Decay rate for the first moment estimates. We set it to 0.9.
   - β_2: Decay rate for the second moment estimates. We set it to 0.999.
   - η: Learning rate. We set it to 0.001.
   - ε: A small constant to prevent division by zero. We set it to 10^{-8}.

The default values for the hyperparameters are obtained from the PyTorch documentation, which are the same as the recommended values from (Goodfellow et al., 2016).

8.4. Implementation details

Firstly, certain aspects of our model architecture are fixed and not dependent on tuning:

- **Neural network**. We implement the Adam optimiser, and the activation functions are ReLU, for the hidden layers, and Sigmoid, for the output layer. In addition, we employ three regularisation techniques to prevent overfitting in the neural network: ℓ_2 regularisation adds a penalty term, based on the square of the weight coefficients, to encourage smaller weights; dropout randomly deactivates neurons during training; and early stopping halts training when validation error does not improve for some number of epochs (Guo, 2024).
- **Random forest**. In our pipeline, we first standardise each feature to have zero mean and unit variance. Then, PCA is applied on the features before the data is passed to the Random Forest model. We also fix the number of estimators to a sufficiently large number that is still computationally feasible, as recommended by (Probst & Boulesteix, 2018).
- **Logistic regression**. Our implementation follows the Kryptonite paper, with the original pipeline left unmodified. This is motivated by the aim of our paper, where we want to compare the predictive performance of our models with the main model used by (Quinn & Luther, 2024). As such, we follow the same hyperparameter settings, such as using SGD as optimiser. We use a polynomial degree of 5, which provides a good tradeoff between accuracy and complexity, based on the Kryptonite paper's results. A higher degree, such as 7, is much more prone to overfitting and is very computationally expensive.
- **Gaussian Mixture Model**. We assume that each set of data points labelled as 1 or 0 follows a multivariate Gaussian distribution. The data is split into two subsets based on their labels, and we train a separate Gaussian Mixture Model (GMM) on each subset. Hyperparameter tuning for the GMMs is performed using BayesSearchCV. After training, we compute the likelihood of each test data point under both models. The label corresponding to the model with the higher likelihood is assigned to the data point.

Table 4. Hyperparameters of the Random Forest

Parameter	n=9	n=12	n=15	n=18
n_estimators	500	500	500	500
max_depth	20	25	29	17
min_samples_split	4	2	2	4
min_samples_leaf	1	1	1	17
max_features	log(2)	sqrt	sqrt	log(2)
n_components	9	11	4	15

Table 5. Hyperparameters of the Neural Network

Parameter	n=9	n=12	n=15	n=18
example	1	1	1	1

Table 6. Hyperparameters of the Logistic Regression

Parameter	n=9	n=12	n=15	n=18
polynomial_degree	5	5	5	5

Table 7. Hyperparameters of the GMM Model (label 0)

Param	n=9	n=12	n=15	n=18
cov_type	tied	tied	tied	tied
n_comp	48	42	48	48
tol	1.62e-4	7.34e-3	1.62e-4	1.62e-4

Abbreviations: Param: Parameter, cov_type = covariance type, n_comp = number of components, tol = tolerance.

Table 8. Hyperparameters of the GMM Model (label 1)

Param	n=9	n=12	n=15	n=18
cov_type	tied	tied	tied	tied
n_comp	48	48	48	48
tol	1.62e-4	1,62e-4	1.62e-4	1.62e-4

Abbreviations: Param: Parameter, cov_type = covariance type, n_comp = number of components, tol = tolerance.

The specific hyperparameters for the Neural Network can be found in Table 5, for Random Forest in Table 4, for Logistic regression in Table 6, and for the GMM model in Tables 7,8. These parameters have been tuned using Optuna, as explained in Section 4.3.

8.5. ROC Curves per Model

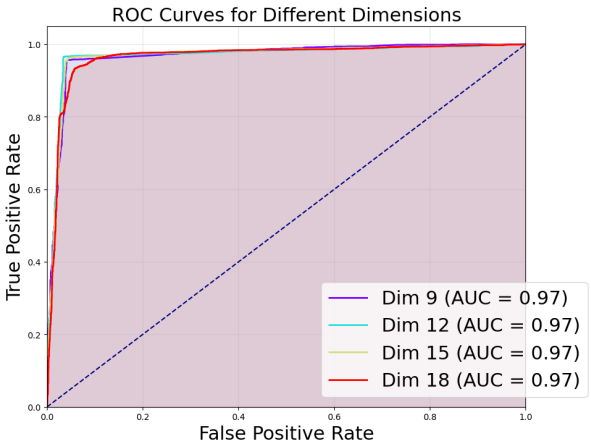

Figure 3. ROC curve for the NN for dimensions 9 through 18

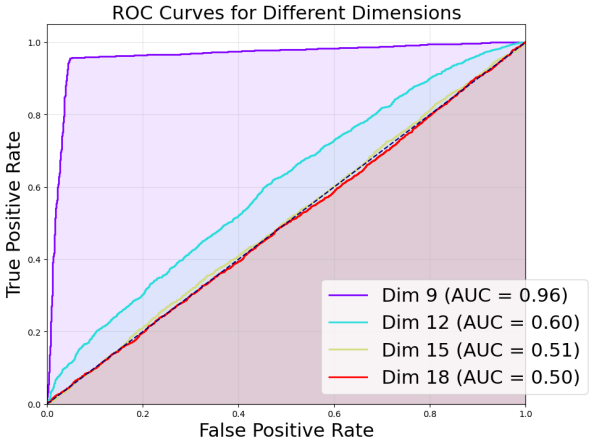

Figure 4. ROC curve for the GMM for dimensions 9 through 18

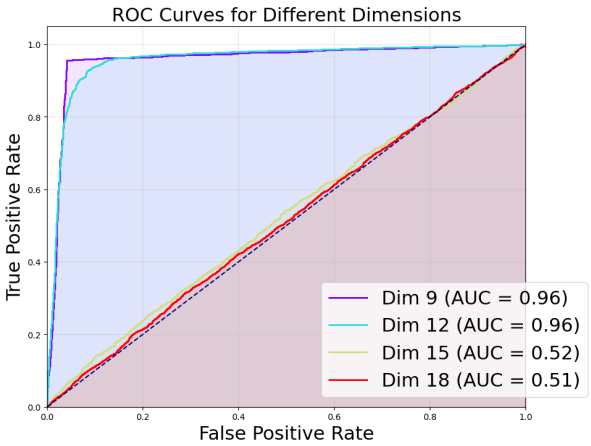

Figure 5. ROC curve for the RF model for dimensions 9 through 18

8.6. Data Pre-processing

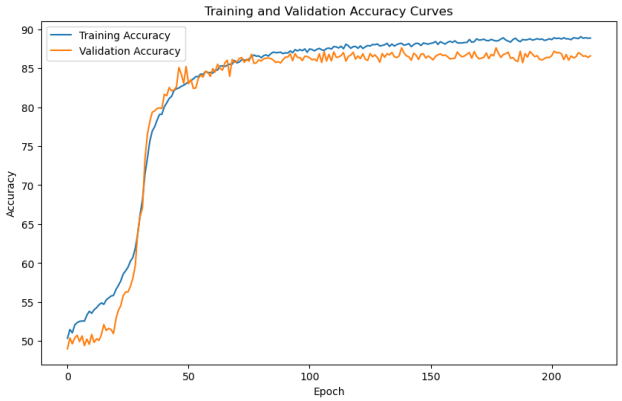

Figure 6. Training and validation accuracies of simple neural network with dimensionality of 18 without any preprocessing

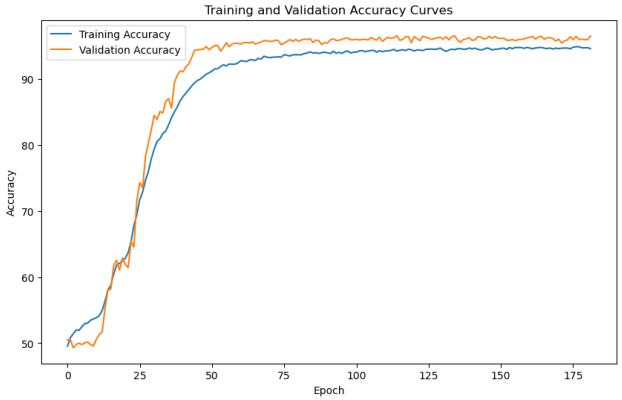

Figure 7. Training and validation accuracies of simple neural network with dimensionality of 18 with pre-processing.

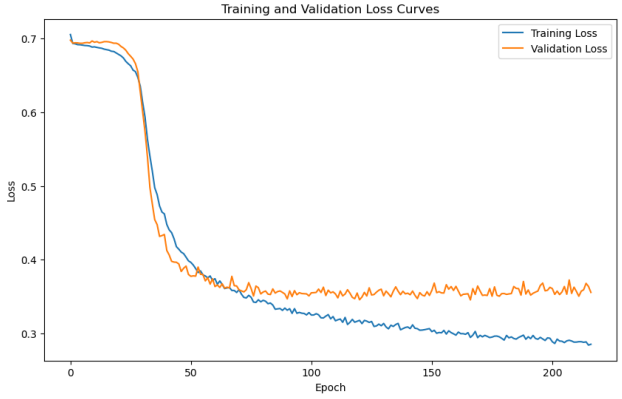

Figure 8. Training and validation losses of simple neural network with dimensionality of 18 without any preprocessing

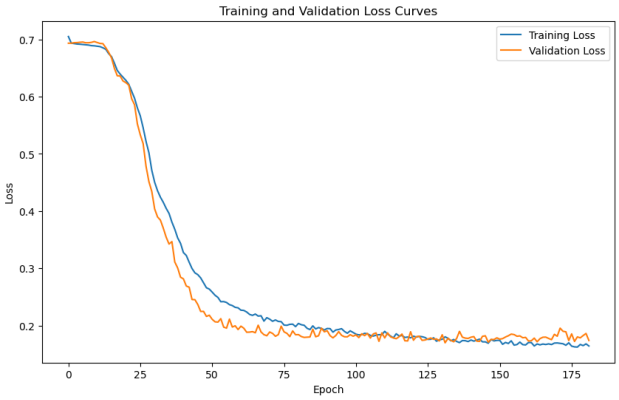

Figure 9. Training and validation losses of simple neural network with dimensionality of 18 with pre-processing.