Imperial Coursework on Machine Learning (ImpCML 2024)

Pengyuan Shao [1] **Renwei Liu** [1] **Andrei Matei** [2]

Abstract

This paper addresses the challenge posed by the Kryptonite-n dataset, which has been claimed to be unsolvable with current machine learning (ML) methods. Our aim is to refute this claim by demonstrating that a Wide and Deep Neural Network (WDNN) can successfully solve the binary classification task presented by the dataset. The WDNN leverages a hybrid architecture, combining explicit pattern recognition from its wide component and high-order feature generalization from its deep component. Through rigorous experimental design, incorporating regularization, dynamic learning rate scheduling, and robust evaluation metrics, the WDNN achieves strong performance across various dimensions of the dataset. Our results show that Kryptonite-n can indeed be solved using current ML methods, providing insights into how such methodologies can address even the most complex datasets.

1. Introduction

The ability to tackle challenging datasets is essential for advancing machine learning (ML) research, as these datasets often represent real-world complexities such as high dimensionality, noisy features, and class imbalance. Kryptonite-n, a synthetic dataset specifically designed to test the limits of ML algorithms, has been described as ”unsolvable” using current methods. Serving as a benchmark for algorithmic robustness, this dataset provides an opportunity to evaluate the adaptability of ML architectures in addressing intricate classification problems. Demonstrating the solvability of such datasets is critical for validating the capabilities of modern ML techniques and informing the development of innovative approaches for similarly complex tasks.

[1]Department of Electrical and Electronic Engineering, Imperial College London, London, United Kingdom [2]Department of Computing, Imperial College London, London, United Kingdom. Correspondence to: Pengyuan Shao <pengyuan.shao21@imperial.ac.uk>.

In this paper, we systematically refute the claim that Kryptonite-n cannot be solved with current ML methods. We demonstrate that carefully designed architectures can overcome even the most difficult benchmarks. To this end, we evaluate the performance of two neural network architectures—Fully Connected Neural Network (FCNN) and Wide and Deep Neural Network (WDNN)—on the Kryptonite-n dataset.

The FCNN employs a straightforward design with two hidden layers and regularization techniques such as dropout and batch normalization, allowing it to model non-linear relationships and maintain training stability. However, its relatively simple architecture consistently underperformed, failing to meet the benchmark's target accuracies and highlighting the challenges of high-dimensional datasets. The WDNN, leveraging a hybrid design that combines a wide component for linear feature interactions with a deep component for hierarchical, non-linear patterns, demonstrated superior performance across all feature dimensions. Using regularization strategies, dynamic learning rate adjustments, and rigorous evaluation, the WDNN consistently exceeded the benchmark thresholds, proving to be robust and effective in addressing the challenges posed by Kryptonite-n.

By presenting these findings, this paper not only refutes the unsolvability claim but also provides valuable insights into how architectural innovations can enable ML models to tackle highly complex datasets, bridging the gap between theoretical advancements and practical applicability.

2. Methodology

2.1. Problem Definition

The Kryptonite-n dataset presents a binary classification task where the feature space is R^n and the label space is $\{0, 1\}$. The goal is to learn a function $f_\theta : R^n \to \{0, 1\}$, parameterized by θ, that maps feature vectors $x \in R^n$ to corresponding labels $y \in \{0, 1\}$. The function f_θ is trained by minimizing the binary cross-entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right], \quad (1)$$

where $\hat{y}^{(i)} = f_\theta(x^{(i)})$ is the predicted probability for the positive class.

2.2. Fully Connected Neural Network (FCNN)

An **Fully Connected Neural Network (FCNN)** consists of an input layer, one or more hidden layers (In our model here are two hidden layers) and an output layer (Kang & Wang, 2014). The hidden layers contain neurons, where every neuron is connected to all neurons in the previous and next layer.

The model flows data from the input layer to the output layer through forward propagation. Each neuron computes a linear combination of weighted inputs and produces a nonlinear output via an activation function. The model performance is measured by loss functions such as mean square error/cross entropy. The weights and biases are updated by back propagation to minimise the loss.

2.3. Model Architecture of FCNN

2.3.1. Input Layer

The input $x \in R^n$ is directly fed into the input layer.

2.3.2. Hidden Layer 1

The first hidden layer has 128 neurons, where all these neurons use the ReLu activation function. Overfitting is prevented by L2 regularization and Batch Normalization is used to speed up training and improve model stability. Overfitting is further mitigated by randomly discarding 30% of the neurons during training.

2.3.3. Hidden Layer 2

Similar to Hidden Layer 1, except that the number of neurons in Hidden Layer 2 is reduced to 64. This is because compared to Hidden Layer 1, Hidden Layer 2 further processes the original features extracted from Hidden Layer 1, making it more compact/abstract and adapted to subsequent tasks. In addition to this, it also serves to prevent overfitting and improve training efficiency.

2.3.4. Output Layer

The output layer has only one neuron and uses a sigmoid activation function. The sigmoid activation function is mathematically defined as:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}, \tag{2}$$

where z is the input to the sigmoid function. This function maps any real-valued input z into a value between 0 and 1, which can be interpreted as a probability. For the output layer in a binary classification task, $\hat{y}$ represents the model's predicted probability that the input belongs to the positive class ($y = 1$).

2.4. Wide and Deep Neural Network (WDNN)

Wide and Deep Neural Network (WDNN) is a hybrid architecture designed to combine the strengths of linear models and deep neural networks (Cheng et al., 2016). This approach is particularly effective for tasks that require both memorization of explicit patterns and generalization to novel scenarios.

The **wide component** is responsible for memorizing explicit patterns and low-order feature interactions, such as co-occurrences or simple correlations. This is achieved by directly modeling linear relationships within the input feature space. As a result, the wide component effectively captures patterns that are easily interpretable and often critical for robust predictions in datasets with sparse or high-cardinality features.

In contrast, the **deep component** focuses on generalization by modeling high-order and non-linear feature interactions. Through multiple layers of non-linear transformations, it uncovers intricate relationships in the data that are not easily captured by linear models. This capability is particularly advantageous for datasets where complex patterns dominate the decision boundaries.

To give an example: Think about a task for classifying birds that can fly. The wide component might capture simple and explicit patterns, such as "If the bird is a penguin, it cannot fly." These are straightforward rules based on known co-occurrences. On the other hand, the deep component could learn more complex patterns, such as the relationship between a bird's wing structure, weight, and flying capability, enabling it to generalize to unseen bird species.

By combining these components, the WDNN achieves a balance between memorization and generalization (Nguyen et al., 2021). The wide component ensures that the model can leverage known feature relationships, while the deep component enables the discovery of novel and complex patterns. This synergy makes the architecture well-suited for challenging tasks like those posed by the Kryptonite-n dataset, which require both explicit pattern recognition and the ability to generalize across high-dimensional feature spaces.

2.5. Model Architecture of WDNN

2.5.1. Input Layer

The input $x \in R^n$ is directly fed into the wide and deep components.

2.5.2. Wide Component

A single dense layer with ReLU activation and L_2-regularization is applied:

$$h_{\text{wide}} = \text{ReLU}\big(W_{\text{wide}}x + b_{\text{wide}}\big), \tag{3}$$

where $W_{\text{wide}} \in R^{256\times n}$ and $b_{\text{wide}} \in R^{256}$.

2.5.3. Deep Component

The deep component consists of three dense layers interleaved with batch normalization and dropout (L=1,2,3):

$$h^{(l)}_{\text{deep}} = \text{Dropout}\Big(\text{BatchNorm}\Big(\text{ReLU}\Big(W^{(l)}h^{(l-1)}_{\text{deep}} + b^{(l)}\Big)\Big)\Big), \tag{4}$$

where l indexes the layers and $h^{(0)}_{\text{deep}} = x$. The number of units reduces from 256 to 128 and 64.

2.5.4. Combination Layer

The outputs of the wide and deep components are combined through a process known as **concatenation**. In the context of neural networks, concatenation refers to the operation of joining two or more tensors (multi-dimensional arrays) along a specified axis. This operation merges the outputs from the wide and deep components into a single unified representation, enabling the model to utilize information from both components simultaneously.

Mathematically, let the output of the wide component be $h_{\text{wide}} \in R^{d_{\text{wide}}}$ and the output of the deep component's final layer be $h^{(L)}_{\text{deep}} \in R^{d_{\text{deep}}}$, where d_{wide} and d_{deep} represent the number of units (or dimensions) in each output. The concatenation operation produces a combined vector $h_{\text{combined}} \in R^{d_{\text{wide}}+d_{\text{deep}}}$, defined as:

$$h_{\text{combined}} = \text{Concat}(h_{\text{wide}}, h^{(L)}_{\text{deep}}), \tag{5}$$

where the resulting vector h_{combined} simply appends the elements of $h^{(L)}_{\text{deep}}$ to h_{wide}. This operation ensures that the features learned by both components are preserved and passed to the subsequent layers.

2.5.5. Output Layer

The combined features h_{combined} from the wide and deep components are passed through a dense layer with a sigmoid activation function to produce the final classification output:

$$\hat{y} = \sigma\big(W_{\text{output}}h_{\text{combined}} + b_{\text{output}}\big), \tag{6}$$

where $W_{\text{output}} \in R^{1\times(256+64)}$ is the weight matrix of the output layer, $b_{\text{output}} \in R$ is the bias term, and $\sigma(\cdot)$ represents the sigmoid activation function.

The same sigmoid activation function applied in FCNN is also applied in this architecture to keep output values between 0 and 1 for class label predictions.

2.6. Optimization and Training

2.6.1. Loss Function

The binary cross-entropy loss is used for optimization:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\big[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\big]. \tag{7}$$

2.6.2. Regularization

Regularization is a fundamental technique in machine learning designed to address the problem of overfitting. Overfitting occurs when a model learns patterns that are too specific to the training data, including noise and outliers, resulting in poor generalization to unseen data. Regularization techniques modify the learning process to discourage overly complex models, ensuring that the learned representations capture the underlying structure of the data rather than artifacts unique to the training set.

In this model, regularization is achieved through three key methods: L_2-regularization, dropout, and batch normalization.

L_2-Regularization:

L_2-regularization, also known as weight decay, penalizes large weights in the model. The intuition behind this is that simpler models, with smaller weight magnitudes, are less likely to overfit the training data. The regularized loss function is defined as:

$$\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{BCE}} + \lambda\sum_{i}\|w_i\|_2^2, \tag{8}$$

where $\mathcal{L}_{\text{BCE}}$ is the original loss (e.g., binary cross-entropy), w_i are the trainable weights, and λ is the regularization coefficient controlling the penalty's strength. The term $\sum_i \|w_i\|_2^2$ is the sum of the squared magnitudes of all weights, encouraging the optimizer to balance minimizing the original loss and keeping the weights small (to make model simpler). In WDNN model, $\lambda = 0.001$ is implemented in all Dense layer (liner layer in both wide and deep component) to maintain a balance between regularization and model expressiveness.

Dropout:

Dropout is a stochastic technique specifically designed in neural networks that randomly deactivates a fraction of neurons during training, effectively forcing the model to learn redundant and robust representations. Dropout prevents the co-adaptation of neurons, promoting independent feature learning and improving the model's robustness.

Batch Normalization:

Batch normalization addresses the issue of internal covariate shift, where the distribution of layer inputs changes during training. This variation can slow convergence and make the model sensitive to the randomness of input and weight initialization. Batch normalization normalizes the inputs to each layer using the mean μ_B and variance σ_B^2 of the current mini-batch:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad y_i = \gamma \hat{x}_i + \beta, \tag{9}$$

where x_i is the input, ϵ is a small constant for numerical stability, and γ and β are learnable parameters that allow the model to recover its original representational capacity. Batch normalization stabilizes learning, allows for higher learning rates, and reduces sensitivity to initialization.

The combined use of L_2-regularization, dropout, and batch normalization ensures:

- Smaller weights, discouraging overly complex models (L_2-regularization),
- Robust and diverse feature representations (dropout),
- Stable and consistent training dynamics (batch normalization).

Together, these techniques significantly enhance the model's ability to generalize, making it more reliable.

2.6.3. Optimization Algorithm

Adam is a popular optimizer (Kingma & Ba, 2014). It full name is Adaptive Moment Estimation. It combine two good methods: Momentum and RMSProp. It adjusts the learning rate of each parameter automatic. This make it work good for most neural networks.

Adam works by calculate two things: 1. First moment (mean of gradients). 2. Second moment (square of gradients).

These help optimizer know the direction and scale of update. Adam also fix some bias in the beginning to make it more stable.

Here is the formula for Adam:

$$\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad \text{(First moment)} \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad \text{(Second moment)} \\
\hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \quad \text{(Bias correction for } m_t\text{)} \\
\hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \quad \text{(Bias correction for } v_t\text{)} \\
\theta_{t+1} &= \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad \text{(Parameter update)}
\end{aligned}$$

Here: - g_t: Gradient at time t. - η: Learning rate (default 0.001). - β_1: Momentum coefficient (default 0.9). - β_2: RMSProp coefficient (default 0.999). - ϵ: Small number (default 10^{-8}).

Adam optimizer is fast and adapt well. It work good for most cases without much tuning. But sometimes, it might not find the best solution, or it overfit the data. For this, other optimizers can be used.

2.7. Evaluation

The model performance is assessed on a testing dataset using accuracy:

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} 1\left[\hat{y}^{(i)} = y^{(i)}\right]}{N}. \tag{10}$$

3. Experimental Design

This section outlines the experimental design used to evaluate the performance of the network on the Kryptonite-n dataset. **The design includes data preparation, hyperparameter settings, training strategies, and validation mechanisms to ensure the robustness of results.**

3.1. Data Splitting Strategy

The dataset is divided into three sections: training, validation, and testing. This split ensures that the model is trained on one subset of the data, validated on another to tune hyperparameters, and tested on an independent set to assess its generalization capability. Specifically:

- **Training Set (40% of data):** Used to train the model and update its weights through backpropagation.
- **Validation Set (30% of data):** Used to monitor the model's performance during training and tune hyperparameters such as learning rate and architecture configurations.
- **Test Set (30% of data):** Held out during training and validation to evaluate the final model's performance on unseen data.

3.2. Standardization of Features

To ensure consistent scaling across features and improve the convergence of the model, the data is standardized using a **StandardScaler**. Each feature is scaled to have a mean of 0 and a standard deviation of 1:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}, \tag{11}$$

where μ is the mean and σ is the standard deviation of the training set features. This transformation is applied consistently across training, validation, and testing sets.

3.3. Hyperparameter Settings

The architectures are evaluated with various hyperparameter settings using the validation dataset to balance computational efficiency and model accuracy, without using the testing dataset to ensure it remains an unbiased measure of generalization performance.

- **Learning Rate:**

  The learning rate is determined by testing a range of values and selecting the one that achieves the highest validation accuracy. Four different learning rates were tested: $[0.005, 0.001, 0.0005, 0.0001]$. The experimental results are presented in the Appendix A. The final model selected $\alpha = 0.001$ for $n = [9, 12, 15, 18]$ and $\alpha = 0.0005$ for $n = 24$. The change in the optimal learning rate is likely due to the increased amount of data provided for $n = 24$ and the greater complexity of optimizing higher-dimensional data.

  A learning rate scheduler dynamically reduces α by a factor of 0.5 after 50 epochs to refine learning in later stages:

$$\alpha_t = \begin{cases} \alpha_0, & \text{if } t \leq 50, \\ 0.5\alpha_0, & \text{if } t > 50. \end{cases} \tag{12}$$

- **Batch Size:** Set to 64 to balance convergence stability and computational efficiency. Smaller batch sizes were found to increase training variability, while larger batches slowed down convergence without significant gains in accuracy.
- **Regularization:** L_2-regularization with $\lambda = 0.001$ is applied to mitigate overfitting. This parameter was chosen to maintain a balance between penalizing overly complex models and preserving the expressive power of the network.
- **Dropout:** A dropout rate of 20% or 30% is used in the deep component layers to promote robust feature learning. This ensures that the network does not become overly reliant on specific neurons, improving its generalization capability.

3.4. Early Stopping to Reduce Training Time

To avoid unnecessary training iterations and prevent overfitting, an early stopping mechanism is employed. Training halts if the validation loss does not improve for 10 consecutive epochs. Additionally, the best model weights (as measured by validation loss) are restored at the end of training, ensuring optimal performance on the validation set.

3.5. Reproducibility

To ensure reproducibility of results, a fixed random seed is set globally for all random operations, including data shuffling and weight initialization. The seeds used in the experiments are 40, 42, and 44, and results are averaged across these runs to account for random variations.

4. Experimental Results

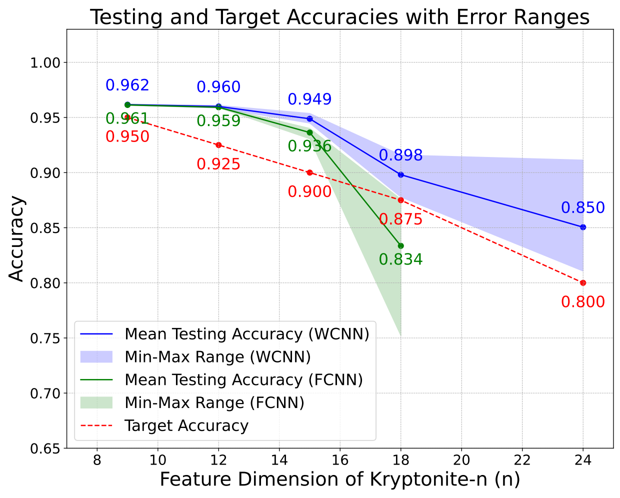

Figure 1. Testing accuracy with error (WDNN)

The results presented in Figure 1 demonstrate the testing accuracies of our models (FCNN and WDNN) against the target accuracies specified by Kryptonite-n benchmark (red dashed line). To make sure the results are robust. The results included are the average accuracies during three different runs with error range indicating the maximum and minimum accuracies achieved.

From the results, it was observed that the Fully Connected Neural Network (FCNN), represented by the green line, consistently underperformed compared to the Wide and Deep Neural Network (WDNN), represented by the blue line. The testing accuracies achieved by the FCNN were not only lower than those of the WDNN across all feature dimensions (n), but they also failed to meet the target accuracies specified in the Kryptonite-n benchmark (red dashed line). In contrast, the WDNN maintained superior performance, achieving testing accuracies that consistently exceeded the target thresholds for all dimensions from $n = 9$ to $n = 24$.

This performance disparity can be attributed to the architectural differences between the two models. While the FCNN employs non-linear activation functions (ReLU) to enable the modeling of non-linear relationships, its relatively shallow architecture limits its ability to effectively capture complex feature interactions in high-dimensional datasets. Additionally, the FCNN lacks specialized components to explicitly handle both linear feature interactions and hierarchical non-linear representations simultaneously, which are critical for tackling the intricate relationships present in datasets like Kryptonite-n.

In contrast, the WDNN leverages a hybrid architecture that combines a wide component, which explicitly captures linear feature interactions, with a deep component, which extracts hierarchical non-linear relationships through multiple hidden layers. This combination allows the WDNN to model both simple and complex feature interactions more effectively than the FCNN.

Another observation is that the error bars are generally larger in higher-dimensional data, indicating increased variability in the model's performance across different training runs or data subsets. This trend is evident for both the FCNN and WDNN models as the feature dimension (n) increases from $n = 9$ to $n = 24$. The increased variability likely reflects the challenges associated with training models on high-dimensional datasets, where the complexity of the feature space amplifies sensitivity to factors such as data splits, initialization, and optimization dynamics.

For both models, the larger error bars may arise due to difficulties in capturing the intricate relationships between features as the dimensionality grows. Gradient instability, sensitivity to hyperparameters, and the need to balance linear and non-linear feature modeling likely contribute to the observed fluctuations. In further studies, strategies such as robust cross-validation and feature reduction methods (e.g., principal component analysis) could improve stability.

Overall, these outcomes directly contradict the claims made in the Kryptonite-n paper, which posited that machine learning models, even when optimized with advanced architectures and basis expansions, fail to meet the target accuracies. By surpassing these thresholds, our results challenge the assertion that Kryptonite-n exposes fundamental limitations in modern machine learning models, reaffirming the viability of well-designed architectures in addressing complex high-dimensional classification tasks.

5. Sustainability Analysis

- **Environmental Considerations** While training deep learning models often requires significant computational resources, this study employed strategies to reduce energy consumption. Techniques such as early stopping and dynamic learning rate adjustments ensured computational efficiency by avoiding unnecessary training epochs. These methods, combined with careful hyperparameter tuning, minimized the carbon footprint associated with experimentation. Further efforts, such as implementing model distillation or deploying the model on energy-efficient hardware, could enhance sustainability.

- **Societal Impacts** The ability to solve complex datasets has far-reaching implications across various industries. For example, the methods used in this research can be adapted to critical domains such as healthcare diagnostics, financial fraud detection, or climate modeling. By solving datasets that represent real-world challenges, our work contributes to advancing machine learning capabilities for societal benefit, bridging the gap between theoretical development and practical application.

6. Conclusion

The results of this study demonstrate the effectiveness of modern machine learning architectures in addressing the challenges posed by the Kryptonite-n dataset, which was previously claimed to be unsolvable. By employing a Wide and Deep Neural Network (WDNN), we successfully surpassed benchmark performance thresholds, providing strong evidence that current ML methods can effectively tackle high-dimensional and complex datasets.

The WDNN's hybrid design, which integrates a wide component for linear feature interactions and a deep component for non-linear hierarchical patterns, has proven to be pivotal. This combination enables the model to balance explicit memorization with generalization, outperforming simpler architectures such as the Fully Connected Neural Network (FCNN). The introduction of techniques like regularization, dynamic learning rate scheduling, and dropout ensured robust generalization, while the use of concatenation layers further enhanced the synergy between the wide and deep components.

Despite these successes, challenges remain in training models on high-dimensional data, as evidenced by the increased variability in performance metrics. These challenges highlight the importance of employing advanced optimization techniques and robust evaluation strategies. Future studies could explore incorporating dimensionality reduction techniques or domain-specific feature engineering to mitigate these effects further.

References

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. Wide & deep learning for recommender systems. *arXiv preprint arXiv:1606.07792*, 2016.

Kang, K. and Wang, X. Fully convolutional neural networks for crowd segmentation. *arXiv preprint arXiv:1411.4464*, 2014.

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

Nguyen, T., Raghu, M., and Kornblith, S. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. *arXiv preprint arXiv:2010.15327*, 2021.

A. Learning rate exploration

Different learning rates were tested for three random seeds. The results indicate that lr $= 0.001$ performs best for $n = 9$ and $n = 15$, with only marginal differences compared to other learning rates. However, for $n = 24$, lr $= 0.0005$ significantly outperforms the other options. This difference is likely due to the increased complexity of the high-dimensional feature space at $n = 24$, where a smaller learning rate provides more stable convergence and avoids overshooting during optimization. This stability becomes critical as the feature dimensionality increases, making lr $= 0.0005$ better suited for the higher complexity of the dataset.

Table 1. Validation Accuracy for Different Learning Rates (n=9)

Learning Rate (α)	Validation Accuracy (%)
0.0001	95.2
0.0005	95.6
0.001	**96.2**
0.005	94.7

Table 2. Validation Accuracy for Different Learning Rates (n=15)

Learning Rate (α)	Validation Accuracy (%)
0.0001	94.8
0.0005	95.2
0.001	**96.0**
0.005	95.3

Table 3. Validation Accuracy for Different Learning Rates (n=24)

Learning Rate (α)	Validation Accuracy (%)
0.0001	72.4
0.0005	**85.0**
0.001	63.1
0.005	51.3