Randomized Sparse Connectivity in Deep Neural Networks: A Literature Review and Methodological Framework

Adrian Seeley

February 15, 2025

Abstract

We review the literature on randomized and sparsely connected neural networks and introduce a novel framework in which each neuron in the hidden layers receives a fixed number of random inputs from any preceding layer while the output layer remains fully connected. The proposed methodology is described in detail, including the construction of a directed acyclic graph (DAG) to represent the network, the training procedure on the MNIST dataset, and extensive parameter exploration. Our preliminary results indicate that increasing the number of inputs per neuron yields a higher return on parameter investment compared to increasing layer or neuron counts. Future work will investigate these findings further. The completelete source code CSV resultsults are included as supplementary material.

1 Introduction

Deep neural networks (DNNs) have advanced considerably in recent years, driven by innovations such as skip connections, sparse connectivity, and random initialization. In this work, we provide a review of the literature of related approaches and introduce a new framework where each neuron in the hidden layers is connected to a fixed number of randomly selected inputs from any preceding layer, followed by a fully connected output layer. This architecture is motivated by the need to balance network capacity with parameter efficiency and to explore the potential benefits of randomized connectivity patterns. We aim to offer a clear and reproducible methodology that may inspire further research in this area.

2 Related Work

Several lines of research have explored random connections and sparsity in neural networks. In the following subsections, we summarize key contributions that have influenced our work.

2.1 Randomly Wired Neural Networks

Xie et al. [15] introduced the concept of random wired neural networks utilizing classical random graph models such as Erds-Rényi, Barabási-Albert and Watts-Strogatz. Their results indicate that unconventional connectivity patterns can be competitive with hand-designed architectures. Mendes [9] further validated these findings through extensive empirical studies.

2.2 Gradient Descent and Skip Connections

Skip connections have been shown to improve training dynamics in deep networks. Weinan et al. [14] demonstrated that such connections facilitate convergence during gradient descent. Additional studies by Oyedotun et al. [10] and Shan et al. [11] have examined their impact on optimization and generalization, even incorporating them within neural architecture search frameworks.

2.3 Random Vector Functional Link Networks

Random Vector Functional Link (RVFL) networks offer an alternative to conventional training methods by using randomization in hidden layers. Katuwala et al. [7] proposed a deep RVFL (dRVFL) model with randomly initialized and fixed hidden layers. Hu et al. [3] further developed an ensemble approach that incorporates random skip connections. Huang et al. [4] provided a comprehensive survey of extreme learning machines (ELMs), emphasizing their efficiency and robust approximation capabilities through randomization.

2.4 Randomization-Based Neural Architectures

Randomization has been central to several neural network design strategies. Suganthan and Katuwal [13] discussed non-iterative training and stochastic configuration networks, while Berry and Quoy [1] examined random recurrent networks and the emergence of structured connectivity through Hebbian learning.

2.5 Sparse Connectivity and Evolutionary Training

Sparse connectivity is considered an efficient alternative to dense architectures. Mocanu et al. [7] introduced Sparse Evolutionary Training (SET), which iteratively prunes and regrows connections during training. Liu et al. [13] further demonstrated that networks can be trained with a fixed parameter budget while maintaining competitive performance.

2.6 Optimization Techniques and Network Architectures

The success of deep learning has been underpinned by effective optimization methods such as Adam optimizer [8] and batch normalization [6]. In parallel, architectural innovations such as highway networks [12], deep residual networks (ResNets) [2], wide residual networks (WRNs) [16], and densely connected networks (DenseNets) [5] have significantly advanced network performance. The lottery ticket hypothesis [12] further suggests that sparse subnetworks can be effectively trained in isolation.

3 Methodology

In this work, we propose a deep neural network framework based on a modular computational graph. This section describes the network wiring, graph construction, training procedure, and the experimental parameter exploration.

3.1 Network Wiring

Traditional feedforward networks employ dense, layer-to-layer connections. Our approach deviates from this norm in two ways:

1. **Random Sparse Connections:** Each neuron in the hidden layers randomly selects exactly I inputs from any preceding layer (including the raw input layer). If fewer than I inputs are available, all available inputs are used. The selection is performed without replacement.

2. **Fully Connected Output Layer:** The final layer is fully connected to every neuron in the network, ensuring that all features contribute to the final decision.

Figure 1 illustrates a comparison between a standard feedforward network and the proposed architecture with random sparse internal connections.

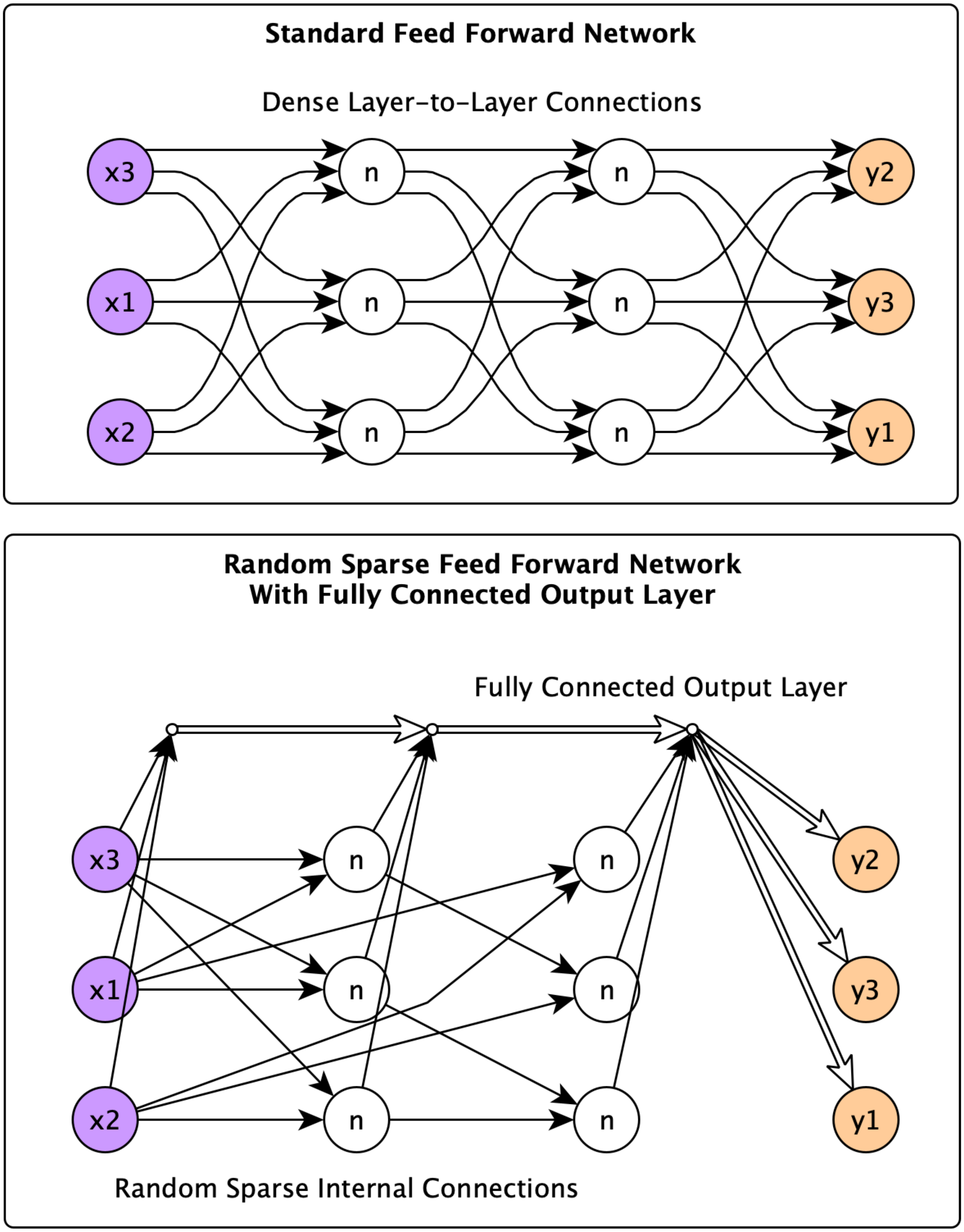
Standard Feed Forward Network
Dense Layer-to-Layer Connections
x3
x1
x2
n
n
n
n
n
n
y2
y3
y1
Random Sparse Feed Forward Network
With Fully Connected Output Layer
Fully Connected Output Layer
x3
x1
x2
n
n
n
n
n
n
y2
y3
y1
Random Sparse Internal Connections

Figure 1: Comparison of network wiring strategies. (Top) A standard feedforward network with dense sequential connections. (Bottom) The proposed architecture with random sparse internal connections and a fully connected output layer.

3.2 Graph Construction and Compilation

The computational graph is assembled as follows:

1. **Component Definition:** Inputs, outputs, constants, learned variables, and operations are defined.
2. **Connection Specification:** Relationships among these components are established to form the network.

Once constructed, the graph is compiled via dependency resolution, which organizes operations into a parallel forward execution schedule. This facilitates:

- Insertion of inputs,
- Execution of a forward pass to compute outputs, and
- Backpropagation to compute gradients with respect to learned variables.

An integrated automatic differentiation module then performs gradient descent using a squared error loss function.

3.3 Training Procedure

Training is performed on mini-batches of 256 samples randomly drawn with replacement from the 60,000-sample MNIST training set. Approximately 235 epochs are required to expose the network to the entire dataset. The training procedure includes:

1. **Initialization:** All learned variables are initialized with small random values drawn uniformly from $(-0.0001, 0.0001)$.
2. **Gradient Computation:** For each mini-batch, gradients are computed for each sample and then averaged.
3. **Optimization:** The Adam optimizer [8] is applied with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$) and a learning rate of 0.001.
4. **Regularization:** A small weight decay factor of 0.9999 is applied to every non-constant learned variable at the end of each epoch.

No additional regularization techniques (such as dropout, L1, or L2 regularization) were used. Training continued until approximately 100,000 gradient updates had been processed, corresponding to roughly 1.6 exposures per training sample.

3.4 Network Architecture

The network architecture is defined as follows:

Input Layer: Normalized MNIST images (with pixel values scaled to $[0, 1]$) serve as input. No further operations are applied at this stage.

Randomly Connected Neuron Block: This block consists of L layers, each containing N neurons. For each neuron:

- Up to I inputs are randomly selected from any preceding layer (including the input layer).
- If fewer than I inputs are available, all available inputs are used.

Output Layer: The final layer (layer $L + 1$) includes 10 neurons corresponding to the MNIST digit classes (using one-hot encoding). This layer is fully connected to all preceding layers, ensuring complete feature integration.

Activation Function: All neurons (in both hidden and output layers) use the LeakyReLU activation function with a leakiness coefficient of 0.01 to mitigate the issue of inactive neurons.

3.5 Parameter Exploration

We trained nearly 1000 neural networks with varying configurations to assess the impact of key parameters:

- **Number of layers** (L): 1 to 100,
- **Neurons per layer** (N): 1 to 100,
- **Number of random inputs per neuron** (I): 1 to 100.

This exploration aims to identify the minimal requirements for effective learning and to evaluate trade-offs between network capacity and performance.

3.6 Data Preprocessing and Evaluation

Preprocessing: The MNIST dataset is normalized on a per-component basis so that all input features fall within the range $[0, 1]$. Output labels are converted to one-hot encoded vectors, and all samples are retained in both the training and test sets.

Evaluation: After training, each test sample is processed via a forward pass. The predicted class is determined by applying the argmax function to the output vector, and performance is measured as the percentage of misclassified samples (argmax error). Evaluation on the training set is also performed to assess generalization, ensuring that no test data is inadvertently used during training.

4 Results

Our experiments indicate that uniformly increasing all parameters (neuron count, layer count, and neuron input count) consistently improves accuracy until an asymptotic limit is reached. Notably, when the parameter budget is constrained, increasing the number of inputs per neuron appears to yield a higher return on performance than simply increasing the number of layers or neurons per layer. This suggests that providing neurons with access to a broader set of features may be more beneficial than merely increasing network depth or width, particularly given that the MNIST dataset comprises 784 features.

Figure 2 shows that increasing the layer count alone, without a corresponding increase in neuron count or input connections, results in suboptimal performance. Similar trends are observed in

Figure 3 for the neuron count. In contrast, Figure 4 demonstrates that networks with higher numbers of inputs per neuron consistently achieve lower test argmax errors. The 3D scatter plots (Figures 5 and 6) further illustrate the combined effects of these parameters on network performance.

It is noteworthy that even a network with only a fully connected output layer achieved a test argmax error of approximately 0.08, while the addition of multiple layers of sparse, randomly connected neurons reduced the test argmax error to 0.0231 (i.e., 231 misclassifications in 10,000 test samples). These preliminary results suggest that random sparse connectivity in deep networks can be both stable and effective, provided that the overall capacity is sufficient. All training was completed using 8 CPU cores (4.0 GHz) over 9 days, without GPU acceleration, and no data leakage was observed.

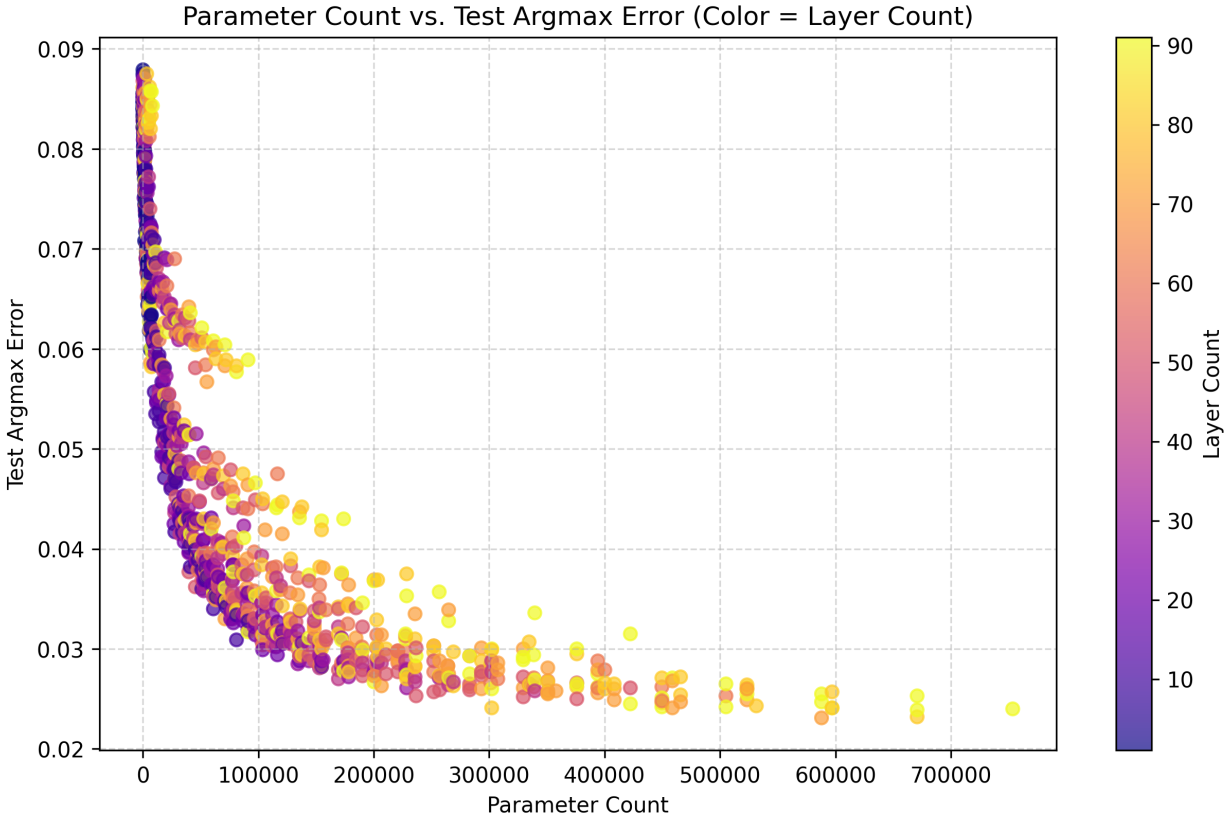

Figure 2: Effect of increasing the layer count on test argmax error and total parameter count. Increasing layer count without increasing neuron count or input count results in suboptimal performance, as indicated by the horizontal error branches.

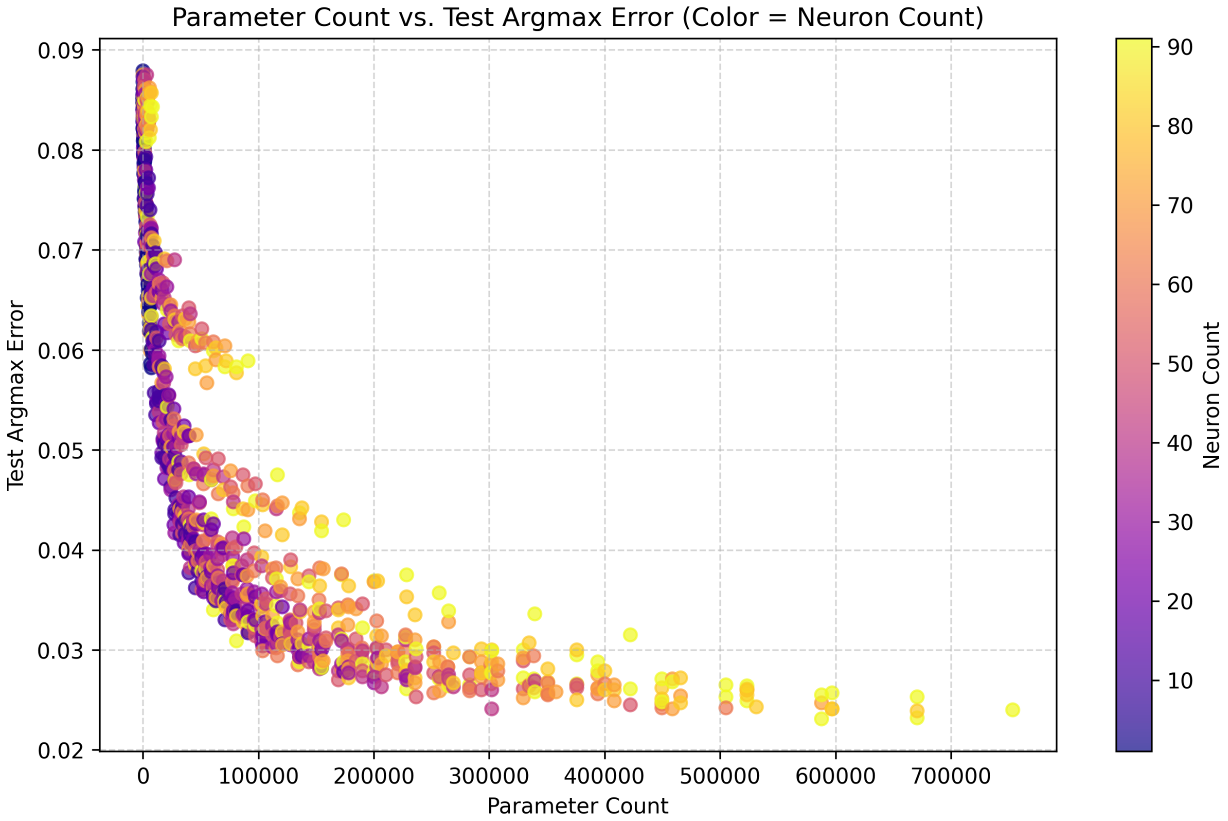

Figure 3: Effect of increasing the neuron count on test argmax error and total parameter count. Similar to the layer count, increasing neuron count alone does not yield significant improvements in accuracy.

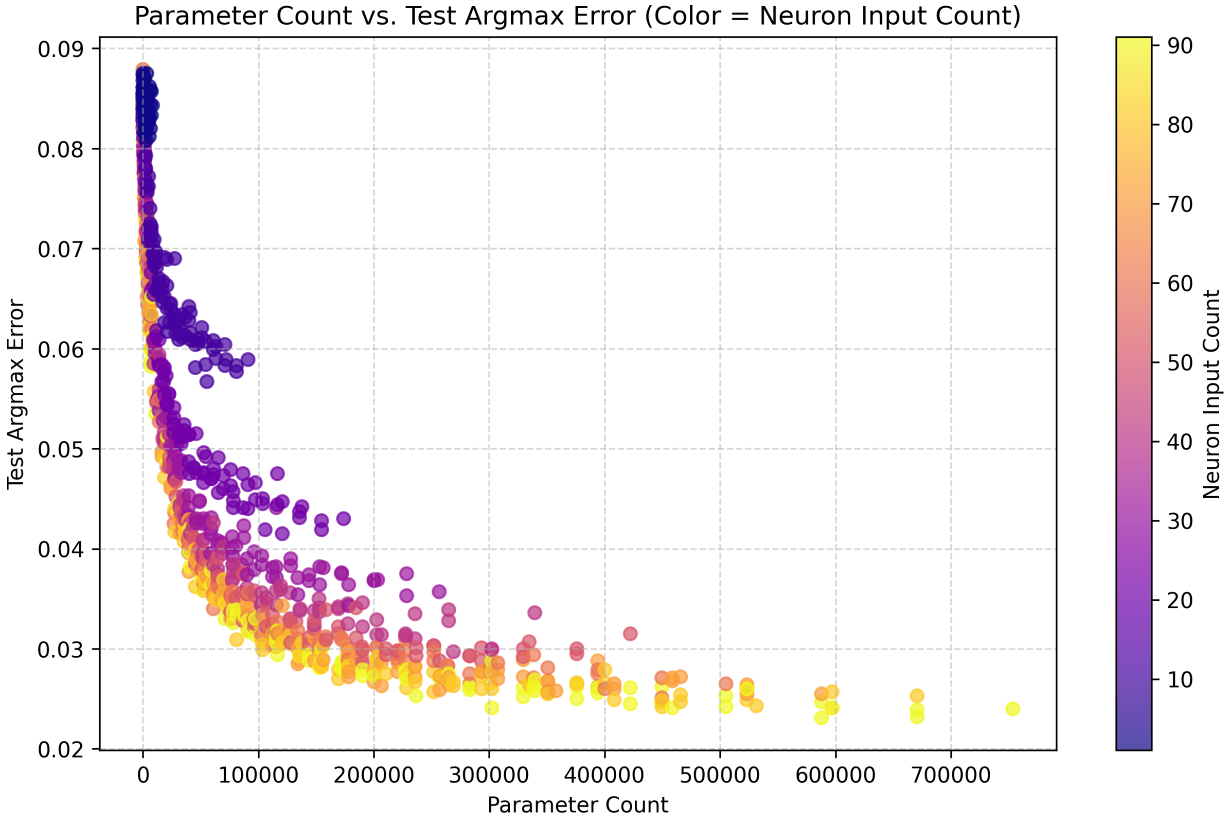

Figure 4: Effect of increasing the neuron input count on test argmax error and total parameter count. Networks with a higher number of inputs per neuron consistently achieve lower test argmax errors.

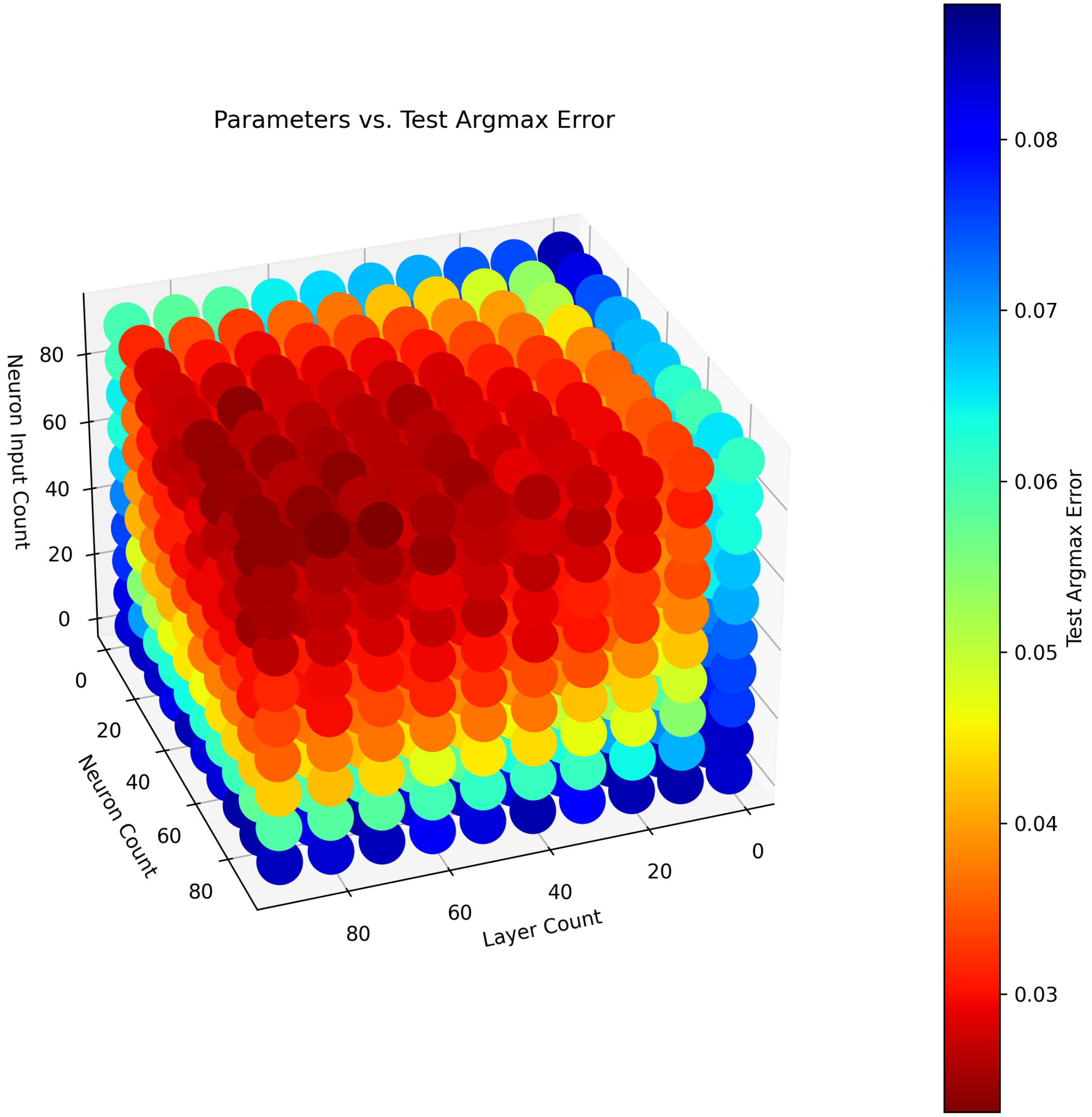

Figure 5: 3D scatter plot (front view) showing the relationship between layer count, neuron count, neuron input count, and test argmax error.

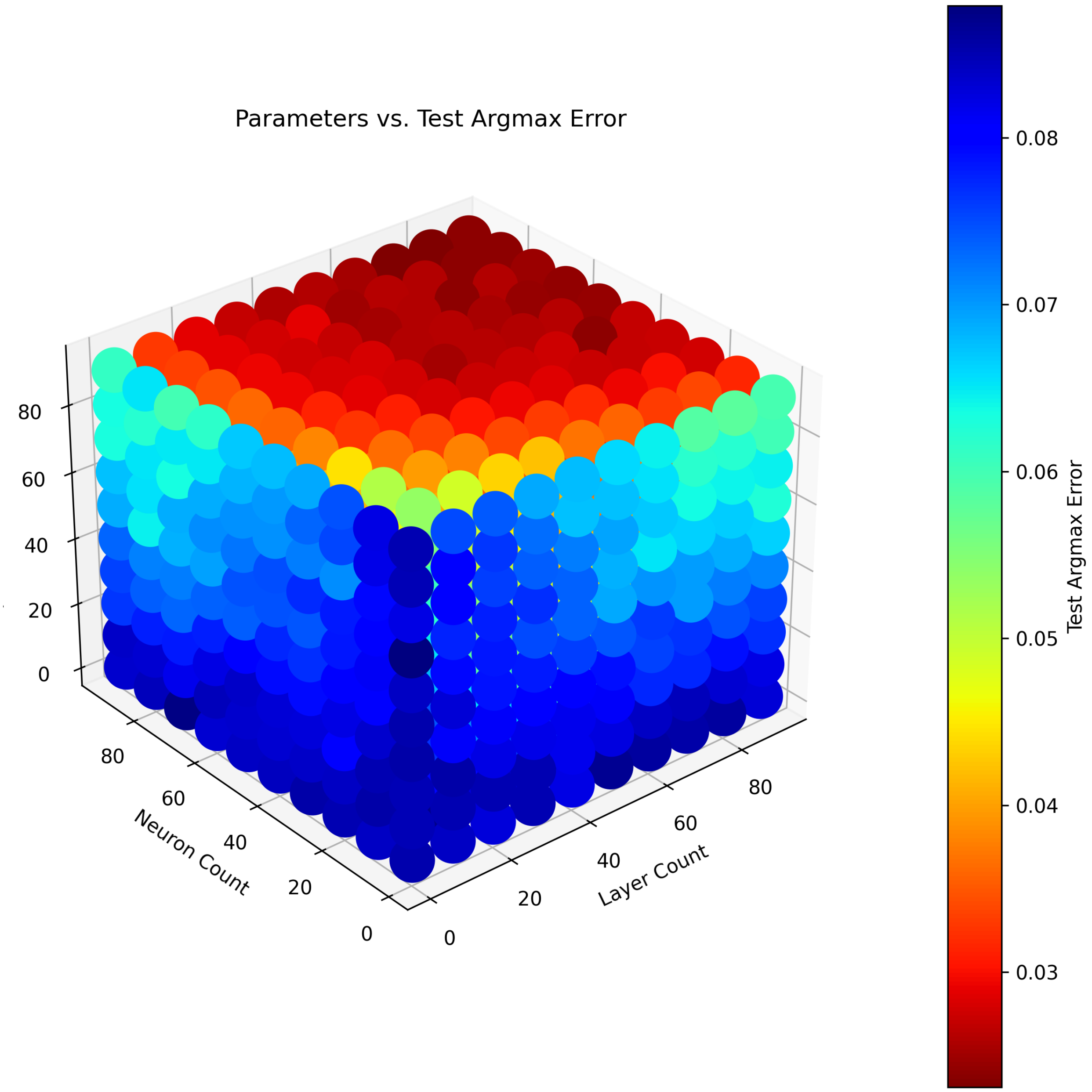

Figure 6: 3D scatter plot (back view) illustrating the combined effects of the key parameters on test argmax error.

5 Conclusion

We have presented a literature review and a novel framework for deep neural networks that employs random sparse connectivity in the hidden layers combined with a fully connected output layer. Our methodology, based on constructing a modular computational graph and an efficient training procedure, demonstrates that increasing the number of random inputs per neuron can be more beneficial than simply increasing network depth or width. These initial results on the MNIST dataset suggest that randomized sparse connectivity is a promising direction for future research, particularly in settings where parameter efficiency is critical. Future work will involve more detailed experiments and further exploration of extremely deep and wide networks.

References

[1] Hugues Berry and M. Quoy. Structure and dynamics of random recurrent neural networks. *Adaptive Behavior*, 14:129 – 137, 2006.

[2] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, null:770–778, 2015.

[3] Minghui Hu, Jet Herng Chion, P. Suganthan, and Rakesh Katuwal. Ensemble deep random vector functional link neural network for regression. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53:2604–2615, 2023.

[4] G. Huang, Dianhui Wang, and Yuan Lan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2:107–122, 2011.

[5] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, null:2261–2269, 2016.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167:null, 2015.

[7] Rakesh Katuwal, P. Suganthan, and M. Tanveer. Random vector functional link neural network based ensemble deep learning. *ArXiv*, abs/1907.00350:null, 2019.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980:null, 2014.

[9] João Mendes. Empirical evaluation of random wired neural networks. 2020.

[10] O. Oyedotun, Kassem Al Ismaeil, and D. Aouada. Why is everyone training very deep neural network with skip connections? *IEEE Transactions on Neural Networks and Learning Systems*, 34:5961–5975, 2022.

[11] Dongjing Shan, Xiongwei Zhang, Wenhua Shi, and Li Li. Neural architecture search for a highly efficient network with random skip connections. *Applied Sciences*, null:null, 2020.

[12] R. Srivastava, Klaus Greff, and J. Schmidhuber. Training very deep networks. 2015.

[13] P. Suganthan and Rakesh Katuwal. On the origins of randomization-based feedforward neural networks. *Appl. Soft Comput.*, 105:107239, 2021.

[14] E. Weinan, Chao Ma, Qingcan Wang, and Lei Wu. Analysis of the gradient descent algorithm for a deep neural network model with skip-connections. *ArXiv*, abs/1904.05263:null, 2019.

[15] Saining Xie, Alexander Kirillov, Ross B. Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, null:1284–1293, 2019.

[16] Sergey Zagoruyko and N. Komodakis. Wide residual networks. *ArXiv*, abs/1605.07146:null, 2016.