Performance and Interpretability of RNN Architectures: A Case Study on psMNIST

Luis Irisarri[*]
Institute for Cross-Disciplinary Physics and Complex Systems (IFISC), CSIC-UIB, Palma de Mallorca, Spain

(Dated: May 18, 2024)

This study evaluates the performance of various Recurrent Neural Network (RNN) architectures, including simple RNN (sRNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Legendre Memory Unit (LMU) hybrids, on the permuted sequential MNIST (psMNIST) dataset. We analyzed training and validation losses, accuracies, confusion matrices, and weight distributions. Our findings reveal that LMU-enhanced models consistently outperform non-LMU models, achieving lower losses, higher accuracies, and more efficient training times. The confusion matrix highlighted model strengths and weaknesses in digit classification, while weight distribution analysis indicated successful learning of data patterns. This study underscores the superior performance of LMU models and suggests future research directions, including activation function analysis and complex systems approaches, to enhance neural network interpretability and robustness.

I. INTRODUCTION

Artificial Neural Networks (ANNs) have revolutionized the field of machine learning and artificial intelligence, achieving state-of-the-art performance in a variety of applications such as image recognition, natural language processing and autonomous driving. The success of ANNs can be attributed to their ability to model complex, non-linear relationships between inputs and outputs by learning from large datasets. Despite these advances, traditional ANNs face limitations in handling sequential data, which is inherently present in numerous real-world tasks including time series prediction, speech recognition, and language modeling [1].

Recurrent Neural Networks (RNNs) address this challenge by incorporating temporal dynamics into their architecture. Unlike feedforward networks, RNNs maintain a hidden state that captures information from previous time steps, effectively giving them a form of memory. This makes RNNs particularly well-suited for tasks where context and sequential dependencies are critical. This capability of RNNs to process and generate sequences of data underscores their importance in advancing the capabilities of neural networks in handling sequential data [1].

In this work, the main objective is to assess the performance of different RNN architectures on large randomized sequential data. In this manner, RNNs are particularly challenged since the information is randomly spread across the entire sequence, making it difficult to capture long-range dependencies [2]. The study aims to compare the performance of different RNN architectures on this problem and identify the most suitable architecture for this task. Additionally, we will deepen the interpretability of our results. This includes analyzing the training and validation losses and accuracies to understand generalizability. We will also evaluate testing capabilities using the confusion matrix. Finally, we will examine the weights distributions of the models to uncover the underlying network structure that represents the learned process.

The remainder of this paper is organized as follows. In Section II, we describe the materials and methods used in the study. This includes data preparation, the RNN models considered, hyperparameter selection, and computational specifics. In Section III, we present the results obtained, discuss their implications, and enhance interpretability. Finally, in Section IV, we summarize the main conclusions of the study and suggest directions for future research.

II. MATERIALS & METHODS

In this section, we first define the dataset considered, and outline the data preparation procedure in §II A. Next, in §II B, we introduce the Recurrent Neural Network (RNN) models used in this study. Subsequently, in §II C, we detail the selection of hyperparameters employed for model training. Finally, in §II D, we describe the computational environment and resources utilized for conducting the experiments.

A. Data Preparation

For our study, we have utilized the Modified National Institute of Standards and Technology (MNIST) dataset, a well-known benchmark in image classification. This dataset comprises a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, labeled with a digit from 0 to 9. To facilitate the assessment of the learning process, we further split the training set into 50,000 training examples and 10,000 validation examples.

In particular, we consider the permuted sequential MNIST (psMNIST) dataset, which is a transformation of the original MNIST. To generate psMNIST, we flattened the 28x28 images into a 784-dimensional vector and then

[*] luis.irisarri1@estudiant.uib.es

randomly permuted the pixels of the original MNIST images. This permutation is fixed across all images in the dataset. Additionally, we normalized the pixel values to the range [0, 1] by dividing by 255, as it is well known that this improves neural network training [3, 4]. The goal is to predict the original label of the image after the permutation has been applied.

The psMNIST dataset presents a significant challenge for RNNs due to its long sequences and the random distribution of information across these sequences. This makes it an excellent benchmark for evaluating the capability of RNNs to capture long-range dependencies [2].

B. RNN Models

In this study, we have considered several RNNs with different architectures to ensure completeness and enable comprehensive comparison. Notably, state-of-the-art results have been achieved with RNNs incorporating Legendre Memory Units (LMU), as demonstrated in the literature [2, 5, 6]. Specifically, we have evaluated the following RNN architectures: simple RNN (sRNN), long short-term memory (LSTM), gated recurrent unit (GRU), LMU + sRNN, LMU + LSTM, and LMU + GRU. For all models, we selected similar hyperparameters to ensure a fair comparison.

In general, an RNN is a type of neural network designed to handle sequential data. An RNN processes sequences by iterating through the elements and maintaining a hidden state that depends on previous elements in the sequence. The hidden state is updated at each time step and is used to make predictions. The main difference between various types of RNNs lies in how the hidden state is updated.

For the sRNN, at each time step t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$a^{<t>} = g_1\left(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a\right) \tag{1}$$

$$y^{<t>} = g_2\left(W_{ya}a^{<t>} + b_y\right) \tag{2}$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

The advantages of the sRNN are its simplicity and ease of implementation. However, sRNNs are computationally slow and cannot consider any future input for the current state. More importantly, they struggle to capture long-range dependencies due to the vanishing and exploding gradient problems, which can hinder training. These issues arise because capturing long-term dependencies involves multiplicative gradients that can exponentially decrease or increase with the number of layers.

To address the vanishing gradient problem, specific gates are used in some types of RNNs. These gates, typically denoted Γ, have a well-defined purpose and are expressed as:

$$\Gamma = \sigma\left(Wx^{<t>} + Ua^{<t-1>} + b\right) \tag{3}$$

where W, U, b are coefficients specific to the gate, and σ is the sigmoid function. The main models that implement the gated mechanism are LSTM and GRU. The gates help mitigate the vanishing gradient effect, though they do not completely eliminate it. On the other hand, the LMU constructs memory cells that can retain information over long periods, enabling the model to identify long and complex temporal patterns in the sequential input.

For a more detailed explanation of the different RNN architectures, we refer the reader to [1, 7–9].

C. Hyperparameter Selection

Selecting the hyperparameters for RNNs is a nontrivial task due to the numerous parameters that can be tuned. However, we have chosen a set of hyperparameters based on previous literature that has proven effective for this type of problem [2, 5].

Regarding network structure, we used an input layer with $28 \times 28 = 784$ neurons, a single hidden layer with 256 units, and an output layer with 10 neurons. For non-LMU RNNs, we included a batch normalization layer after the hidden layer, given its known benefits [10–12]. The tanh activation function was employed for the hidden layer, while the sigmoid activation function was used for the recurrent activations. The output layer utilized the softmax activation function, which is standard in multiclass classification problems [4]. To enhance generalization, we applied a dropout of 0.2 and a recurrent dropout of 0.5 to the hidden layer, except for the sRNN, which had a dropout of 0.3 and no recurrent dropout [13].

For the training procedure, we used the Adam optimizer, known for its general-purpose performance [14]. We set the batch size to 1000 and the learning rate to 0.001. Non-LMU models were trained for 100 epochs, while LMU models were trained for 60 epochs due to their faster convergence. The loss function used was categorical cross-entropy, the standard for multi-class classification problems [4].

As we shall see in the next section (§III), the selected hyperparameters were appropriate for this problem, as the models achieved good performance on the psMNIST dataset. Nonetheless, further hyperparameter tuning could potentially improve the models' performance, and we leave this for future work.

D. Computational Specifics

The modeling and training of the RNNs were conducted using the `Python` programming language, with the `Keras` library and the `TensorFlow` backend. To implement the LMU, we utilized the `keras-lmu` library [15], which provides a robust implementation of the LMU cell for `Keras` models. More details about the coding may be found in `https://github.com/liris8`.

Given the high computational demands of training these models, we employed the High Performance Computing cluster *Nuredduna*. This cluster features CPU-intensive nodes equipped with AMD Epyc2 7402 and Intel Xeon E5-2630 processors, managed as a single queue with a total of 1568 cores (992 Epyc2 and 576 Xeon). Additionally, it includes GPU nodes with NVIDIA RTX 3090 GPUs, which significantly accelerate the training process. Leveraging the GPU nodes allowed us to efficiently train the models over a reasonable number of epochs, ensuring the reliability and robustness of our results.

III. RESULTS & DISCUSSION

In this section, we present the main results of the study, including the performance of different RNN architectures on the psMNIST dataset. We analyze the training and validation losses and accuracies to assess the learning process, the confusion matrix to evaluate the model's performance on each class, and the weight distributions to understand the learned representations.

First, we address the issues of underfitting and overfitting. Underfitting occurs when the model fails to capture the underlying structure of the data, leading to high training and validation losses. Conversely, overfitting happens when the model learns the noise in the training data, resulting in low training losses but high validation losses. The ideal scenario is achieving low training and validation losses, indicating that the model has effectively learned the underlying structure of the data without overfitting [1]. Our observations show that all models achieved low training and validation losses, indicating successful learning of the psMNIST dataset's structure. However, the LMU models exhibited the lowest training and validation losses, suggesting superior capability in capturing long-range dependencies compared to non-LMU models.

For the non-LMU models (see **??**), the sRNN exhibited the highest training and validation losses, indicating difficulty in capturing long-range dependencies in the psMNIST dataset. The LSTM and GRU models showed lower training and validation losses, with the LSTM model achieving the most balanced training and validation losses. As expected, the sRNN's simplicity resulted in higher training efficiency, while the LSTM and GRU models required more computational resources and time due to their gated structures.

For the LMU models (see **??**), all three models demonstrated outstanding performance with steep convergence (note the difference in epochs). Interestingly, except for the LMU + sRNN model, the LMU + LSTM and LMU + GRU models were more efficient in training time than their non-LMU counterparts.

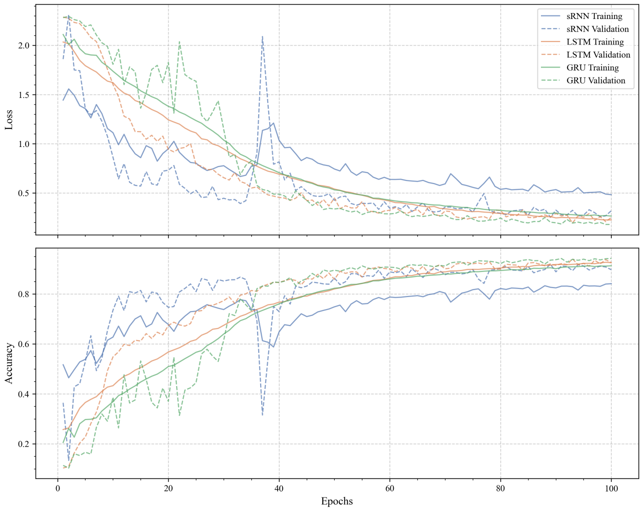


Figure 1. **Training and Validation Losses and Accuracies of Basic RNNs (No LMU)**. This figure shows the evolution of training and validation losses and accuracies for the sRNN, LSTM, and GRU models on the psMNIST dataset. The hyperparameters used are detailed in §II C.

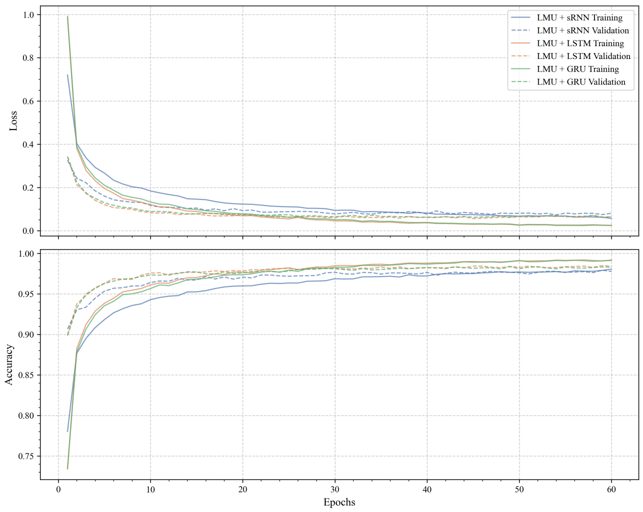


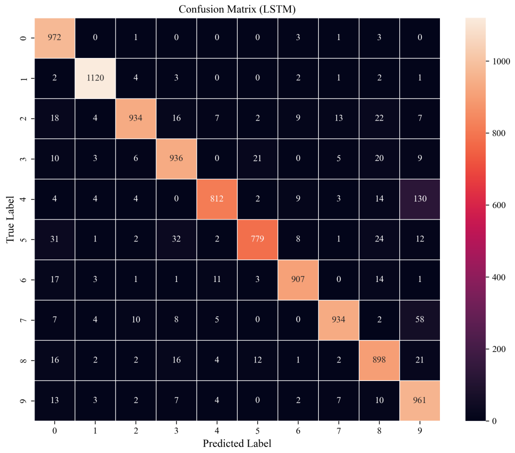
Figure 2. **Training and Validation Losses and Accuracies of LMU Architectures**. This figure shows the evolution of training and validation losses and accuracies for the LMU + sRNN, LMU + LSTM, and LMU + GRU models on the psMNIST dataset. The hyperparameters used are detailed in §II C.

The summary performance results of the different RNN architectures are presented in tab. I. The table details the number of parameters for each model, the final training, validation, and testing losses, the final training, validation, and testing accuracies, as well as the average epoch time. Our findings are consistent with those reported in the existing literature [2, 6].

To further deepen our understanding of the models' predictive power, we analyze the confusion matrix (CM). The CM entry C_{ij} represents the number of observations known to be in class i and predicted to be in class j. The CM allows us to evaluate the model's performance on

Model RNN Architecture	Parameters	Final Losses (±0.0001)			Final Accuracies (±0.01%)			Epoch time
		Training	Validation	Testing	Training	Validation	Testing	(±1 s)
sRNN	69642	0.4825	0.3200	0.3406	84.15	89.74	89.20	11
LSTM	267786	0.2263	0.2507	0.2637	92.56	92.54	92.53	47
GRU	202506	0.2652	0.1800	0.1969	91.33	94.26	93.66	41
LMU + sRNN	134155	0.0568	0.0809	0.0649	98.06	97.77	97.32	22
LMU + LSTM	528907	0.0241	0.0637	0.0643	99.18	98.42	98.32	32
LMU + GRU	398091	0.0250	0.0646	0.0691	99.14	98.30	98.36	31

Table I. **Performance of Different RNN Architectures on the psMNIST Dataset.** This table summarizes the key performance metrics of various RNN models evaluated in the study, including the number of parameters, final training, validation, and testing losses and accuracies, as well as the average epoch time. The hyperparameters used are detailed in §II C.



	0	1	2	3	4	5	6	7	8	9
0	972	0	1	0	0	0	3	1	3	0
1	2	1120	4	3	0	0	2	1	2	1
2	18	4	934	16	7	2	9	13	22	7
3	10	3	6	936	0	21	0	5	20	9
4	4	4	4	0	812	2	9	3	14	130
5	31	1	2	32	2	779	8	1	24	12
6	17	3	1	1	11	3	907	0	14	1
7	7	4	10	8	5	0	0	934	2	58
8	16	2	2	16	4	12	1	2	898	21
9	13	3	2	7	4	0	2	7	10	961

Figure 3. **Confusion Matrix of LSTM**. In this figure, we plot the confusion matrix of the LSTM model for the psMNIST dataset. The hyperparameters used are detailed in §II C. Although this CM is for the LSTM model, the CMs of the other models exhibit almost identical patterns.

each class, providing insights into the model's strengths and weaknesses. In fig. 3, we present the CM for the LSTM model for brevity, noting that the CMs for the other models exhibit nearly identical patterns. As expected, the digit 1, often written as a straight line, is the best-predicted digit by all models. Conversely, the digit 5 is the worst-predicted, likely due to its similarity to digits 3, 6, and 8. Additionally, digits 4 and 9, which are visually similar, are the most frequently confused by all models.

We also analyzed the weight distributions of the models to understand the learned representations. Again for brevity, we only present the weight distributions of the LSTM model in fig. 4, noting that the other models show similar patterns. The trained weight distributions are more bell-shaped compared to the uniformly distributed non-trained weights, suggesting that training has regularized and fine-tuned these parameters for better model performance and stability. In contrast, the uniformly distributed non-trained weights reflect random initialization, indicating that the model has not yet learned the underlying structure of the data.

Finally, we consider the actual values of the model weights. In fig. 5, we present the weights of the LSTM model. It is evident that the trained weights exhibit an underlying structure, whereas the non-trained weights are randomly distributed, indicating that the model has learned the data's underlying structure.

IV. CONCLUSIONS

In this study, we have systematically assessed the performance of various Recurrent Neural Network (RNN) architectures on the permuted sequential MNIST (psMNIST) dataset. The architectures considered include the simple RNN (sRNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Legendre Memory Unit (LMU) hybrids: LMU + sRNN, LMU + LSTM, and LMU + GRU. Our analysis focused on several key performance metrics: training and validation losses and accuracies, confusion matrix analysis, and weight distribution evaluation.

The results demonstrate that the LMU-enhanced models consistently outperformed their non-LMU counterparts across all metrics. Specifically, LMU models achieved lower training and validation losses, higher accuracies, and more efficient training times. This superior performance is attributed to the LMU's ability to capture long-range dependencies more effectively. The confusion matrix analysis further highlighted that while all models performed well on certain classes (e.g., digit 1), they struggled with others (e.g., digit 5), which can be attributed to the visual similarities among certain digits. The analysis of the weight distributions and values indicated that the trained weights of the models exhibited more bell-shaped distributions and underlying structure, suggesting that the models successfully learned the intrinsic patterns of the data.

While our study provides significant insights into the performance and learning characteristics of different RNN architectures on the psMNIST dataset, it also opens avenues for further research. Previous works have explored the interpretability of neural networks using various techniques, ranging from network theoretic ap-

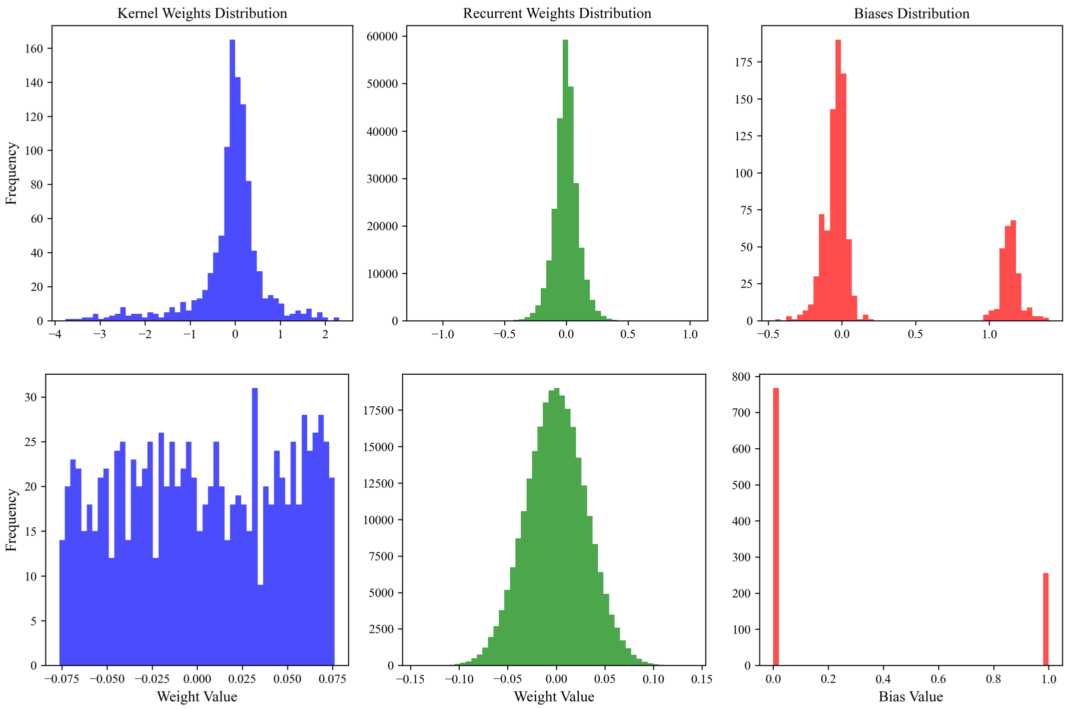


Figure 4. **Weights Distributions of LSTM**. In this figure, we plot on top the trained weights distributions of the LSTM model for the psMNIST dataset. On the bottom, we plot the non-trained weights distributions of the LSTM model. The hyperparameters are the same in both cases, see §II C for further details.

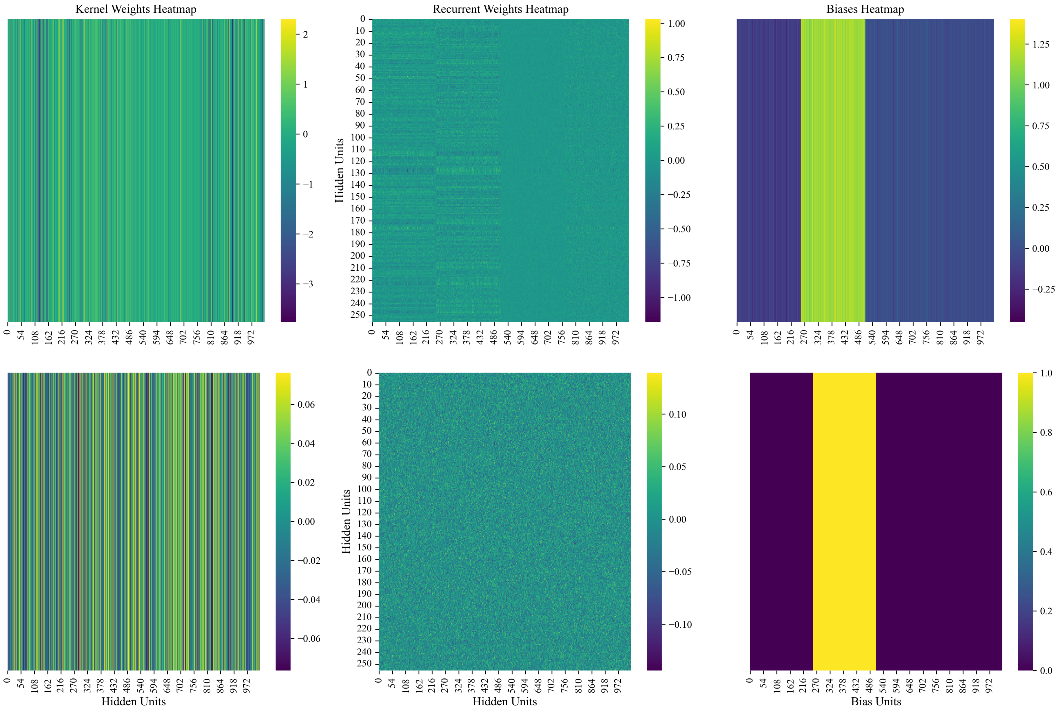


Figure 5. **Weights Values of LSTM**. In this figure, we plot on top the trained weights of the LSTM model for the psMNIST dataset. On the bottom, we plot the non-trained weights of the LSTM model. The hyperparameters are the same in both cases, see §II C for further details.

proaches [16–19], to information-theoretic methods [20–23], and dynamical systems approaches [24], as well as other methods [25, 26]. For instance, additional analyses could focus on the activation functions used by the models when predicting specific classes, which might offer deeper insights into the decision-making process of the networks. Furthermore, applying this ideas from complex systems, such as information theory, dynamical systems, or network theory, could enhance our understanding of the neural networks' interpretability and robustness.

In conclusion, our comprehensive evaluation of various RNN architectures highlights the superior performance of LMU-enhanced models on the psMNIST dataset. These findings contribute to the growing body of research on the efficacy of advanced RNN architectures and underscore the importance of exploring new methodologies for enhancing neural network performance and interpretability. Future research should continue to explore these directions, leveraging advanced analytical techniques to further unravel the complexities of neural network learning and generalization.

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[2] Solving the permuted sequential MNIST (psMNIST) task — KerasLMU 0.3.0 docs, https://www.nengo.ai/keras-lmu/v0.3.0/examples/psMNIST.html.

[3] R. Pramoditha, Acquire, Understand and Prepare the MNIST Dataset (2022).

[4] Marmikpatani, MNIST classification using different activation functions and optimizers with implementation— . . . (2020).

[5] A. Voelker, I. Kajić, and C. Eliasmith, Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks, in *Advances in Neural Information Processing Systems*, Vol. 32 (Curran Associates, Inc., 2019).

[6] Papers with Code - Sequential MNIST Benchmark (Sequential Image Classification), https://paperswithcode.com/sota/sequential-image-classification-on-sequential.

[7] S. Hochreiter and J. Schmidhuber, Long Short-Term Memory, Neural Computation **9**, 1735 (1997).

[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (2014), arxiv:1406.1078 [cs, stat].

[9] S. Amidi and A. Amidi, CS 230 - Recurrent Neural Networks Cheatsheet, https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[10] J. Brownlee, A Gentle Introduction to Batch Normalization for Deep Neural Networks (2019).

[11] K. Doshi, Batch Norm Explained Visually — How it works, and why neural networks need it, https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739 (2021).

[12] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, How Does Batch Normalization Help Optimization? (2019), arxiv:1805.11604 [cs, stat].

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The journal of machine learning research **15**, 1929 (2014).

[14] S. Ruder, An overview of gradient descent optimization algorithms (2017), arxiv:1609.04747 [cs].

[15] Nengo/keras-lmu, Nengo (2024).

[16] E. La Malfa, G. La Malfa, G. Nicosia, and V. Latora, Characterizing Learning Dynamics of Deep Neural Networks via Complex Networks, in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)* (2021) pp. 344–351.

[17] E. La Malfa, G. La Malfa, C. Caprioli, G. Nicosia, and V. Latora, Deep Neural Networks as Complex Networks (2022), arxiv:2209.05488 [cs].

[18] E. La Malfa, G. La Malfa, G. Nicosia, and V. Latora, Deep Neural Networks via Complex Network Theory: A Perspective (2024), arxiv:2404.11172 [cs].

[19] V. A. C. Horta, I. Tiddi, S. Little, and A. Mileo, Extracting knowledge from Deep Neural Networks through graph analysis, Future Generation Computer Systems **120**, 109 (2021).

[20] E. R. Balda, A. Behboodi, and R. Mathar, An Information Theoretic View on Learning of Artificial Neural Networks, in *2018 12th International Conference on Signal Processing and Communication Systems (ICSPCS)* (2018) pp. 1–8.

[21] X. Xu, S.-L. Huang, L. Zheng, and G. W. Wornell, An Information Theoretic Interpretation to Deep Neural Networks, Entropy **24**, 135 (2022).

[22] N. Tishby and N. Zaslavsky, Deep learning and the information bottleneck principle, in *2015 IEEE Information Theory Workshop (ITW)* (2015) pp. 1–5.

[23] R. Shwartz-Ziv, Information Flow in Deep Neural Networks (2022), arxiv:2202.06749 [cs].

[24] K. Danovski, M. C. Soriano, and L. Lacasa, Dynamical stability and chaos in artificial neural network trajectories along training (2024), arxiv:2404.05782 [cond-mat, physics:nlin, physics:physics].

[25] J. D. Olden and D. A. Jackson, Illuminating the "black box": A randomization approach for understanding variable contributions in artificial neural networks, Ecological Modelling **154**, 135 (2002).

[26] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling, Visualizing Deep Neural Network Decisions: Prediction Difference Analysis (2017), arxiv:1702.04595 [cs].