COMP 551 Assignment 4

Carl Machaalani 260895058; Winnie Yongru Pan 261001758

December 5, 2023

Abstract

In this project, we aim to reproduce a portion of the paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". During training of the neural network, at each training step, we randomly drop neurons and their connections with some probability, resulting in the parameters of only a thinned subnetwork being relied upon and updated. The key idea that we validate is that dropout is a useful technique, as it improves the accuracy of a neural network by reducing overfitting and gives major improvements over other regularization techniques.

1 Introduction

Neural networks are versatile but prone to overfitting due to learning from noise in training sets. To address this, we employ techniques like early stopping, weight penalties (L1, L2 regularization), and max-norm regularization. In our reproduction of the original paper, we recreate models as close to the one in the original paper as possible and introduce dropout, a method that efficiently simulates combining multiple network configurations by randomly omitting neurons during training, thus reducing the computational burden. The ultimate purpose of this project is to see whether the original paper's idea is replicable and applicable to most use cases.

2 Scope of Reproducibility

- Claim 1 (6.1.1): Implementing dropout in neural networks with a variety of architectures, while keeping all hyperparameters including the dropout rate (p) constant, significantly improves classification robustness. In the paper, this is evidenced by distinct test error trajectories for networks trained with versus without dropout, as demonstrated across multiple architectures. The benefit of dropout is observed without the need for architecture-specific hyperparameter tuning.
- Claim 2 (6.5): Employing dropout regularization in neural networks trained on the MNIST dataset results in a lower generalization error compared to networks without dropout. Specifically, a network architecture with 784-1024-1024-2048-10 neurons and ReLU activations achieves a generalization error of 0.95%, which further improves to 0.94% when combined with max-norm constraints and maxout units. In the original paper, this claim is supported by empirical data presented in the original paper, which directly compares the error rates of different regularization methods under the same network architecture.
- Claim 3 (7.4): Given the same model architecture, the variation of different data set size will also influence how well the dropout in term of reducing classification error. In a hidden layer dropout rate of p = 0.5 and an input layer dropout rate of p = 0.8, one should expect lower classification error with dropout compare to without when the data set size is larger then 10^3

3 Methodology

To approach the project, we follow the available instruction to reconstruct necessary models. We have chosen the MNIST data set for we are more familiar to it. Specifically, we chose this instruction from the original paper to follow: requirement document for the MNIST data-set. One of the reason why we chose to reconstruct a model ourselves is that the original code was largely based on Python 2.0 which was no longer supported in most of the IDEs and by most of the external libraries, modules, and packages. We have used a lot of the detailed described in the paper, here is the link of the paper: Paper Link This is the general link to all of the resources available from the paper: Paper Resource

3.1 Model descriptions

We constructed several models to replicate the claims presented in the paper. Our Standard Neural Network (NN) model features a architecture comprising layers with 748, 800, and 800 neurons, culminating in a 10-neuron output layer. The hidden layers are activated using the sigmoid function, while the output layer employs the softmax function. The 'Adam' optimizer was selected for its efficiency, and we adopted categorical cross-entropy as our loss function to measure the model's performance. For claim 1, we constructed two other major dropout NN models for the purpose of replication of the claims. Table 1 shows the architecture of the dropout model with logistic unit type. Table 2 shows the architecture of the dropout neural network with ReLU activation.

Layer Type	Size	Activation	Dropout
Input Layer	784	-	p = 0.2
Dense	1024	sigmoid	p = 0.5
Dense	1024	sigmoid	p = 0.5
Dense	1024	sigmoid	p = 0.5
Output Layer	10	softmax	-

Table 1: Architecture of the Dropout Neural Network with Logistic Activation

Layer Type	Size	Activation	Dropout
Input Layer	784	-	p = 0.2
Dense	1024	relu	p = 0.5
Dense	1024	relu	p = 0.5
Dense	1024	relu	p = 0.5
Output Layer	10	softmax	-

Table 2: Architecture of the Dropout Neural Network with ReLU Activation

For claim 2, we have constructed a model for testing different regularization methods. The architecture of this model is as below in table 3. This model allows us to test all of the regularization method with one python function defined.

Layer Type	Size / Units	Activation	MaxNorm	L2 Regularization
Input Layer	784	-	-	-
Dropout (Conditional)	-	-	-	-
Dense	1024	relu	3.5 (if 'maxnorm')	0.001 (if 'l2')
Dropout (Conditional)	-	-	-	-
Dense	1024	relu	3.5 (if 'maxnorm')	0.001 (if 'l2')
Dropout (Conditional)	-	-	-	-
Dense	2048	relu	5 (if 'maxnorm')	0.001 (if 'l2')
Dropout (Conditional)	-	-	-	-
Output Layer	10	softmax	-	-

Table 3: Detailed Architecture of the Neural Network with Conditional Regularizations

For Claim 3, the model architecture is modified based on the previous models to suit the needs for testing the effects of dropout on different data size. Table 4 shows the architecture of this version of model.

Layer Type	Size / Units	Activation	Dropout (Conditional)
Input Layer	784	-	-
Dropout (Conditional)	-	-	0.5 (if with_dropout)
Dense	1024	relu	-
Dropout (Conditional)	-	-	0.5 (if with_dropout)
Dense	1024	relu	-
Dropout (Conditional)	-	-	0.5 (if with_dropout)
Dense	2048	relu	-
Dropout (Conditional)	-	-	0.5 (if with_dropout)
Output Layer	10	softmax	-

Table 4: Architecture of the Neural Network with Variable Data Size

3.2 Datasets

The MNIST data set is a vision data set with a dimension of 784(28×28 gray-scale) consists of 60000 examples for the train set and 10000 examples for test set. The label distribution is as in figure 1.

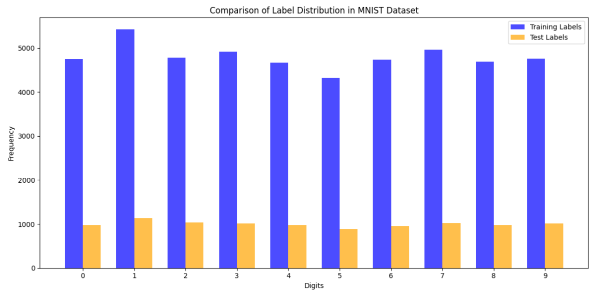


Figure 1: Label Distribution

The MNIST set only has train and test split originally. In our model, we used the sklearn train_test_split method to split the MNIST dataset again into train, test, but instead of 60000 and 10000, we let 20% as test set and 80% as train set. You can download the MNIST data set through Kaggle, with this link. For more information about the MNIST dataset, please find them here.

3.3 Hyperparameters

The number of epochs is constrained by the limitations of Google Colab, which sets an upper limit of 100 epochs for free users. Our momentum value is fixed, as the current TensorFlow module does not support dynamic momentum adjustment. Therefore, we determined the optimal momentum for our purposes through trial and error. We chose the Adam optimizer for logistic activation and SGD for ReLU activation, based on trial and error, finding that Adam and SGD were the most effective for our needs. The paper specified certain p-values, so we adhered to these p-values as outlined in the paper and replicated the necessary models accordingly. This batch size is used because after trails and errors, this batch size gives us the most optimal results for our test.

3.4 Experimental setup and code

This replication process is set up in the Google Colab Free tier version. A maximum of 12.7 GB System RAM, 15 GB GPU RAM and a 78.2 GB Storage is required to run this code. please click here to access our code on Google Collab.

3.5 Computational requirements

To run experiment on Claim 1, one would need a 5.1 GB of system RAM, 2.5 GB of GPU RAM, and 26.9 GB of storage. The average time it takes for the program 20 minutes to converge.

To run experiment on Claim 2, one would need a 4.6 GB of system RAM, 2.5 GB of GPU RAM, 26.9 GB of storage room on Google Collab, and it takes averagely 42 minutes to converge.

In order to run experiment on Claim 3, one would need a 4.4 GB of System RAM, 1.5 GB of GPU RAM, 26.9 GB of storage room and it takes 32 minutes to converge.

4 Results

The results supports the main claim of the original paper, which is that the dropout NN model overall performs better than non-dropout NN models.

4.1 Results reproducing original paper

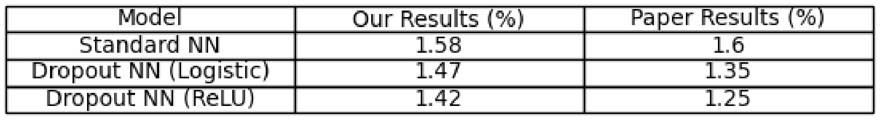
4.1.1 Result for Claim 1: Paper Section 6.1.1

Our study aimed to validate Claim 1 from Section 6.1.1 of the original Dropout paper, which asserts that Dropout improves neural network performance on the MNIST dataset. We conducted experiments to compare the test error rates of a Standard Neural Network (NN) against Dropout NNs with logistic and ReLU activations.

For the Standard NN (2 layers, 800 units), we observed a test error of 1.58%, closely aligning with the paper's reported 1.60%. Our Dropout NN with logistic units (three layers, 1024 units) resulted in a 1.47% error, slightly above the paper's 1.35%.

The Dropout NN with ReLU activations (three layers, 1024 units) was trained with a decaying learning rate to match the original paper's methodology, resulting in a 1.42% error rate compared to the reported 1.25%.

Despite slight deviations likely stemming from differences in optimizer configurations and momentum scheduling, the performance order of the 3 models is demonstrated to be similar to the paper's results, showing the effectiveness of Dropout.



Model	Our Results (%)	Paper Results (%)
Standard NN	1.58	1.6
Dropout NN (Logistic)	1.47	1.35
Dropout NN (ReLU)	1.42	1.25

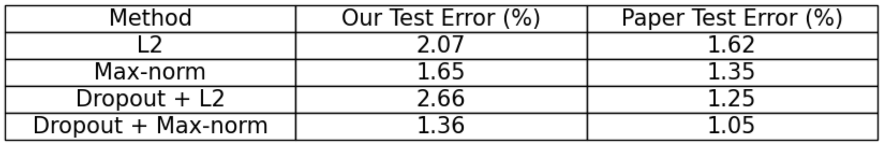
Figure 2: Error Rate Comparison Between Different Models in the Original Study and Our Replication on MNIST dataset

4.1.2 Result for Claim 2: Paper Section 6.5

In replicating the findings of section 6.5 from the original Dropout paper, we evaluated neural networks trained with various regularization techniques on the MNIST data set. More precisely, we tried replicating the results of the methods: L2, Max-norm, Dropout+L2, and Dropout+Max-norm. Our study supported Claim 2, demonstrating that Dropout with max-norm regularization yielded the lowest error, at 1.36%, closely following the paper's reported 1.05%. However, models with L2 and Max-norm regularization alone, as well as Dropout with L2, showed higher errors than those reported—2.07%, 1.65%, and 2.66%, respectively. Also, Dropout + L2 performed worse than both L2 and Max-norm alone, deviating from the paper's findings. However, the overall results of replication 2 still supports claim 2 of the original paper, which is that the Dropout + Max-norm combination gives the best performance for the model. The comparison between our data and the paper's data is given below in figure 2.

4.1.3 Result for Claim 3: Paper Section 7.4

To reproduce the conclusion of section 7.4, we evaluated the effect of data set size using the MNIST data set, an architecture of 784-1024-1024-2048-10, with the dropout p values of 0.1 for the input layers and 0.5 for all the hidden layers. We have tested the results on different data set size of 100, 500, 1k, 5k, 10k, and 50k on this setting. This setting replicated most of the paper's experimental setting, except for the input dropout rate. Many trails and errors has been tried and only with a 0.1 input dropout rate reproduce the



Method	Our Test Error (%)	Paper Test Error (%)
L2	2.07	1.62
Max-norm	1.65	1.35
Dropout + L2	2.66	1.25
Dropout + Max-norm	1.36	1.05

Figure 3: Error Rate Comparison Between Different Regularization Methods in the Original Study and Our Replication on MNIST data set

closest result from the original paper. The comparison of our result and the paper's result is as in Appendix A, figure 4 and 5. It's worth to mention that the output plot of the code differs every run time. So if the output you see in the code is different from the one presented in this report, it is normal.

From the figures we can say that our result supports the claim 3, that "for any given architecture and dropout rate, there is a sweet spot corresponding to some amount of data that is large enough to not be memorized in spite of the noise but not so large that over-fitting is not a problem anyways." From the figure, we can see that the classification error of the model without dropout performs well at the data size of around $10^{2.6}$, it then gain advantage with less classification error until it reaches 10^4. Although the "sweet spot" is slightly different from the one in the paper, it supports the claim of the paper very well.

4.2 Results beyond original paper

Often papers don't include enough information to fully specify their experiments, so some additional experimentation may be necessary. For example, it might be the case that batch size was not specified, and so different batch sizes need to be evaluated to reproduce the original results. Include the results of any additional experiments here. Note: this won't be necessary for all reproductions.

4.2.1 Additional Result 1

For the Standard NN and Dropout NN with logistic units, we opted for the Adam optimizer as the original paper did not specify an optimizer. Similarly, for the Dropout NN with ReLU, we set a fixed momentum of 0.5 rather than adjusting it dynamically as the paper suggested, since TensorFlow does not readily support momentum adjustments during training. The paper also left out batch size and epoch details, leading us to choose 100 epochs and a batch size of 32. Our epoch limit was partly due to the computational constraints of Google Colab's free tier.

4.2.2 Additional Result 2

Our replication faced challenges similar to those in Claim 1, notably the original paper's lack of batch size and epoch details, and TensorFlow's limitations on adjusting momentum. Therefore, we adopted a fixed momentum, and Google Colab's free tier limited us to 100 epochs.

4.2.3 Additional Result 3

For the claim on section 7.4 on the paper, we extended the replication a bit more on different randomization of the selection of the data sets. Since the paper did not specify how exactly did they randomly select their groups of data, We have tried different random seed on the train_test_split methods to see which random seed gives the most representative result. Most of the results aligns with the 3rd claim, some even do better compare to the non-dropout model throughout the whole time, which supported the main claim of the paper, which is that dropout model helps reduce over-fitting and yields lower classification error.

5 Discussion

For the first claim, which is the central assertion of this paper, we have successfully replicated a model that supports the paper's claim. There are slight deviations in the data, likely caused by differences in optimizer usage and momentum scheduling. The paper did not explicitly mention the optimizer used; we opted for Adam, and the limitation of 100 epochs imposed by Google Colab may have also contributed to potential deviations. The paper presented a plot illustrating numerous trials across different architectures, showing that the application of dropout consistently yields lower classification errors. Due to resource constraints, we were unable to conduct as many trials with varied architectures. Nonetheless, the successful replication of the experiment highlights the robustness of the dropout method and confirms the original claim.

As for the second claim, the discrepancies in our results may stem from the same factors mentioned for claim 1. Additionally, we had to select a custom batch size. However, the absence of powerful computing resources, which would allow us to train and adjust our model more rapidly, could also be a reason for not being able to replicate the results precisely.

Regarding the third claim, we present a very convincing plot that is similar to the one from the original paper, although it is not exactly the same. This could be due to a different randomization seed, the limitation of the epoche or methodology that we employed to select random data inputs for the data set.

5.1 What was easy

The author's code, requirement and parameter lists are clearly written. So it was easy to replicate the model by our own following the requirements and the parameter lists. The building of the standard NN model and the processing of the data-set was easy, given that we have done similar projects on neural networks and MNIST data-set before. The regularization methods we are also familiar, for we have used and learned them in the MLP layers lectures and project.

5.2 What was difficult

We was not able run the original code of the paper, even as a reference, because the paper's results are built upon the basis of Python 2. Python 2 is no longer supported by most of the necessary modules and it is no longer supported by Google Colab. We spent a long time trying to resolve the problem and try Python 2, but we ended up realizing that it might not be worth it and we chose to build the model on our own based on python 3. Some of our deviations may come from here. we also had difficulties picking a proper claim to replicate, due to the limitation of our time and computational power. For example, our original plan for claim 3 was to replicate section 7.3 of the paper, which is to see the effects of different dropout rate on the classification error. However, according to the architecture it requires, which is 784-2048-2048-2048-10, it would take about three to four hours to have the results. Adding the extra time of tuning, we decided that section 7.3 is no longer our options for replication.

6 Conclusions

In this comprehensive replication study, we set out to validate key claims from the seminal paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". Our replication align with the original findings, demonstrating the efficacy of dropout as a robust regularization technique in neural networks.

Our results affirm that neural networks employing dropout consistently outperform standard models without dropout on the MNIST dataset. This supports the central premise of the original study, highlighting dropout's role in mitigating overfitting and enhancing generalization. Particularly, our experiments replicate the performance improvements of dropout in conjunction with max-norm regularization, thereby validating its more outstanding generalization capabilities compared to other regularization methods.

Despite some deviations in exact error rates, which can be attributed to differences in optimizers, momentum scheduling, and computational constraints, the overall trend of our results mirrors the claims of the original paper. These differences also highlight the importance of detailed methodological descriptions in research for accurate replication.

Through this replication exercise, we have not only corroborated the effectiveness of dropout in preventing overfitting but also gained valuable insights into practical considerations and challenges in replicating neural network experiments. This study reaffirms the importance of dropout as a key tool in the machine learning practitioner's arsenal, especially in scenarios where over-fitting poses a significant challenge.

7 Statement of Contributions

- Carl was responsible for reproducing both claim 1 and claim 2, along with the corresponding code and result write-ups for each.
- Yongru was responsible for reproducing claim 3, along with the corresponding code and write up as well as the rest of the write up for the project.

8 Appendix

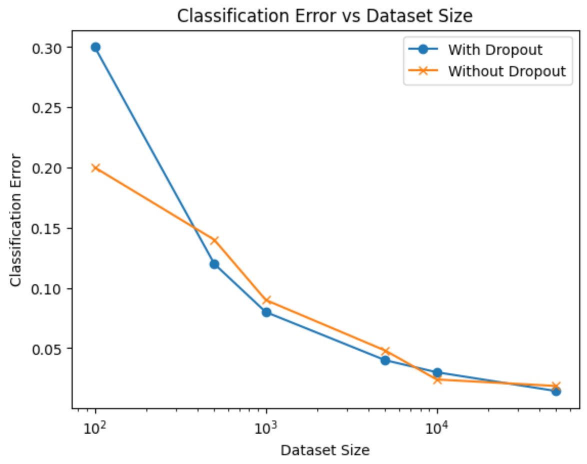


Figure 4: Our results for Section 7.4

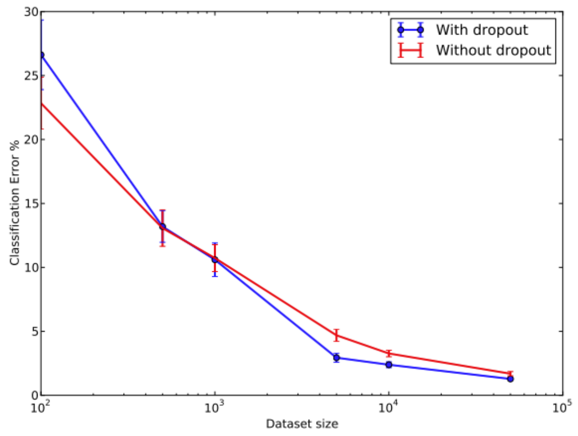


Figure 5: The Result of Section 7.4 from the Original Paper