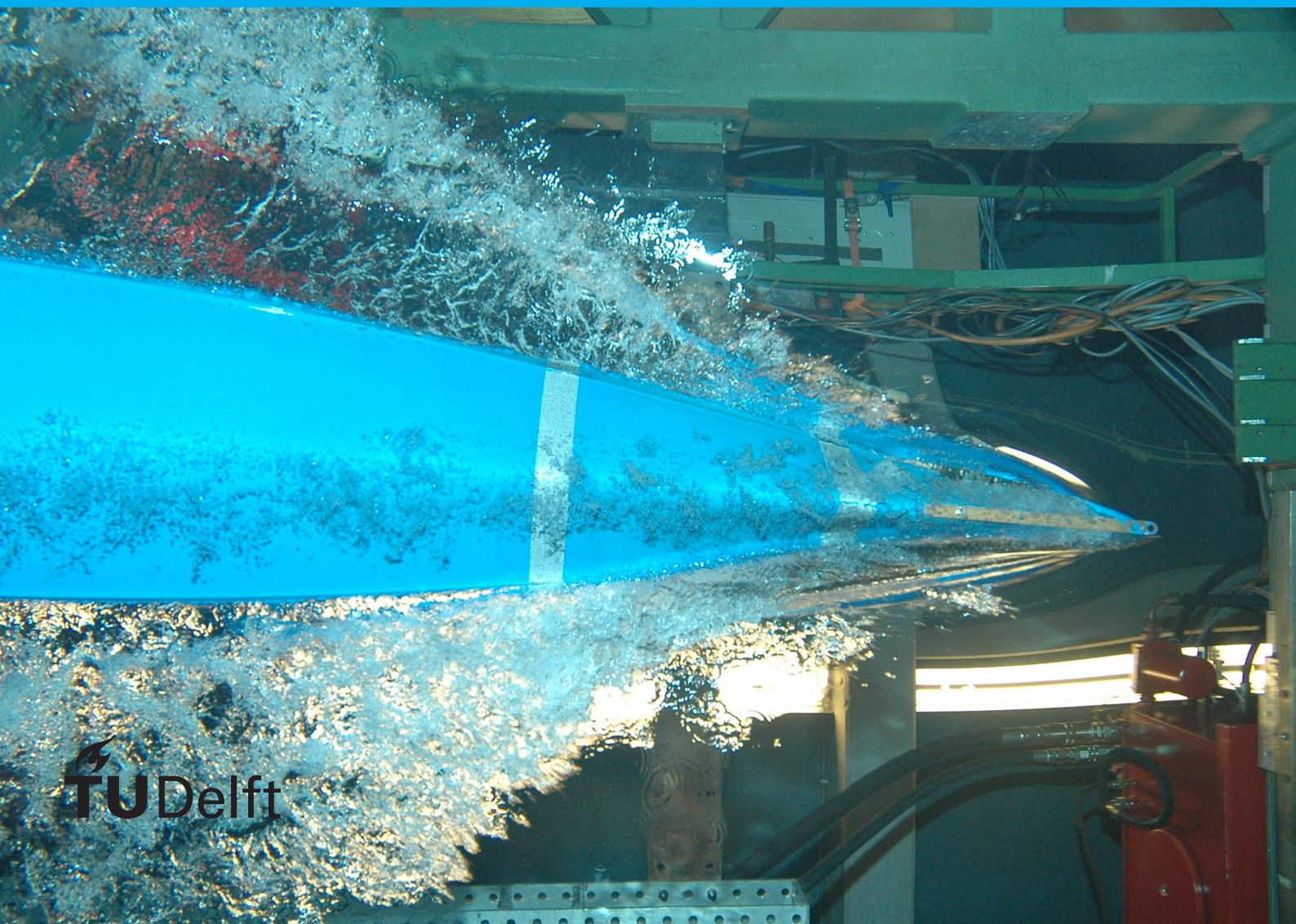
Regularization Effect of Dropout

Xunyi Zhao



Regularization Effect of Dropout

by

Xunyi Zhao

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday May 27, 2021 at 9:00 AM.

Student number:	5139686	
Project duration:	September 1, 2020 – May 27, 2021	
Thesis committee:	Dr. D.M.J. Tax,	TU Delft, supervisor
	Prof. M.J.T. Reinders,	TU Delft
	Dr. F.H. van der Meulen,	TU Delft

An electronic version of this thesis is available at
http://repository.tudelft.nl/.

TU Delft

Preface

Dropout is one of the most popular regularization methods used in deep learning. The general form of dropout is to add random noise to the training process, limiting the complexity of the models and preventing overfitting. Evidence has shown that dropout can effectively reduce overfitting. This thesis project will show some results where dropout regularizes the deep neural networks only under certain circumstances. Potential explanations would be discussed. Our major contributions are 1. summarizing the regularization behaviors of dropout, including how different hyper-parameters could affect dropout's performance; 2. proposing possible explanations to the dropout's regularization behaviors.

This project was conducted within the Pattern Recognition Lab at TU Delft, under the supervision of Dr. D.M.J Tax. I want to thank David for giving me many helpful suggestions. His support was the key to the success of this project. I would also like to thank Prof. M.J.T. Reinders and Dr. F.H. van der Meulen for their interest in my thesis and evaluation of my work.

Xunyi Zhao
Delft, May 2021

Contents

1 Introduction

Dropout is one of the most popular regularization methods used in deep learning. The general form of dropout is to add random noise to the training process, limiting the complexity of the models and preventing overfitting. Evidence has shown that dropout can effectively reduce overfitting. However, dropout is also found not to improve the model's performance when the training set is too small. This surprising finding in a regularizer inspired us to look into how dropout works.

1.1. Dropout and its regularization effect

Srivastava et al. [23] proposed dropout as a simple regularizer in deep learning. They test the performance of dropout in several experiments, studying how hyper-parameters settings could contribute. Specifically, they compare the performance of the networks trained with dropout to those trained without it. They find dropout makes the classification error even higher when the dataset size is small, where the regularization effect is supposed to occur.

To confirm that such behaviors of dropout can be reproduced, we apply the same experiment as Srivastava et al. [23]. Note that many hyper-parameters settings are not mentioned in the paper, including learning rate, momentum, training epochs, and activation functions. Thus some details of our reproduction are different from the original results. Figure 1.1 show the test error vs. the training dataset size from the original results and our reproduction. In both figures of 1.1, there is a range of dataset size where dropout can significantly improve the performance compared to the original network. When the dataset is too small or too large, dropout does not work very well. It is common that a regularizer does not work well with big datasets, because overfitting is usually not a problem when the training set is big enough to reflect the true distribution. But it is hard to explain why dropout does not work with small datasets.

As one of the most important regularization methods in deep learning, dropout has never been fully understood[14]. The result that dropout does not work on the small datasets motivates us to study the regularization behavior of dropout. In this thesis, we try to understand how dropout regularizes deep learning models, especially what makes dropout not work with small datasets. Our focus lies in the classification task with fully connected deep neural networks, where standard dropout is applied in the training stage.

1.2. Complexity and co-adaptation

As a property usually related to overfitting, complexity will be investigated in this report. We will study the relationship between models' complexity and the performance of dropout on those models. We will use the complexity measures depending on not only the number of trainable parameters but also their values. Those measures are found to be useful in predicting the generalization gap[18], which is defined as the difference between training and testing performance. Specifically, three complexity measures will be studied in this

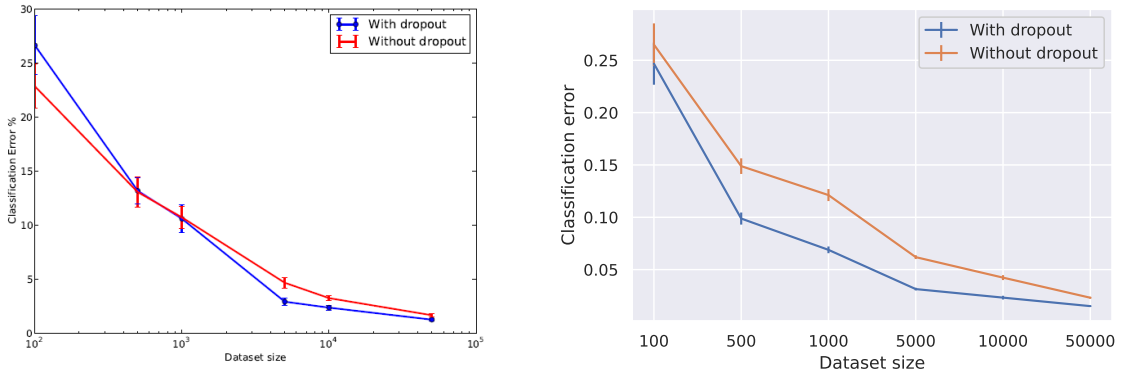


Figure 1.1: Test error vs. training dataset size. Left: original results from Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). *Dropout: a simple way to prevent neural networks from overfitting.* The journal of machine learning research, 15(1), 1929-1958.[23] Right: Our reproducing results. Error bars represent the standard deviation of 10 samples.

report including norms of weights [17], sharpness [24] and sensitivity [21].

Co-adaptation, defined by Hinton et al. [8] as a situation where "a feature detector is only helpful in the context of several other specific feature detectors", has been proposed as an explanation why dropout works. We will develop a way to measure the co-adaptation of a neural network and find its relationship with dropout's performance. The relationships we found between dropout, complexity, and co-adaptation would provide us an intuitive explanation of how dropout works in the way shown in Figure 1.1.

1.3. Preliminaries

We denote matrices, vectors, scalar variables and sets by capital letters, Roman small letters, small letters and script letters, respectively (e.g. X, x, x, and $\mathscr{X}$). We consider a set of deep feedforward neural networks $f_\Theta : \mathscr{X} \to \mathscr{Y}$, where $\mathscr{X} \subseteq \mathbb{R}^{d_0}$ and $\mathscr{Y} \subseteq \mathbb{R}$ denote the input and output space. The function of f is realized as $f_\Theta(X) = \sigma^{(L)}\left(\ldots\sigma^{(2)}\left(\sigma^{(1)}\left(X\theta^{(1)}\right)\theta^{(2)}\right)\ldots\theta^{(L)}\right)$, where L is the number of layers in the network f_Θ. $\theta^{(l)} \in \mathbb{R}^{d_{l-1}\times d_l}$ and $\sigma^{(l)}$ are the weight matrix and the activation function at the l-th layer, where d_l is the number of neurons in the l-th layer, except d_0 is the number of input features. $X \in \mathbb{R}^{m\times d_0}$ is the input batch with m as the batch size. The total number of neurons is $D = \sum_{i=1}^{L} d_i$. A typical way to find the target function is to fit f_Θ on m samples X_{train}, Y_{train} by minimizing a pre-defined loss function $\mathscr{L}(f_\Theta(X), Y)$. The performance of the trained model is defined with the true distribution: $\mathscr{L}(f_\Theta(X_{true}), Y_{true})$ where X_{true}, Y_{true} represent the infinite data set sampled from the true distribution $\mathscr{X}$ and $\mathscr{Y}$. In practice, we approximate the performance based on a finite size samples set: $\mathscr{L}(f_\Theta(X_{test}), Y_{test})$, where X_{tesr}, Y_{test} represent the data pairs not used for the training.

Dropout is defined in the following way: the outputs of neurons in the l-th layer are multiplied by a random diagonal matrix $\xi^{(l)} \in \mathbb{R}^{d_l\times d_l}$, where $\xi^{(l)}_{i,i} = \frac{1}{1-p^{(l)}} b_i, b_i \overset{\text{i.i.d.}}{\sim} Bern(1-p^{(l)})$. $Bern(q)$ is a Bernoulli distribution which gives 1 with probability q and 0 otherwise. The random variables are re-scaled by a normalization factor $\frac{1}{1-p^{(l)}}$ to make $E[\xi^{(l)}_{i,i}] = 1$ for all the is and ls. $p^{(l)}$ is a pre-defined probability of dropping a neuron in the l-th layer. In most cases, the dropout probability $p^{(0)}$ and $p^{(L)}$ are set as 0 to avoid information loss in the predictions.

If not specified, we only apply dropout in the training phase. The dropout network can be written as: $f_{\Theta,\Xi}(X) = \sigma^{(L)}\left(\ldots\sigma^{(2)}\left(\sigma^{(1)}\left(X\xi^{(0)}\theta^{(1)}\xi^{(1)}\right)\theta^{(2)}\xi^{(2)}\right)\ldots\theta^{(L)}\right)\xi^{(L)}$. When dropout is turned off (e.g. in the evaluation phase), the network would be written as $f_\Theta(X) = \sigma^{(L)}\left(\ldots\sigma^{(2)}\left(\sigma^{(1)}\left(X\theta^{(1)}\right)\theta^{(2)}\right)\ldots\theta^{(L)}\right)$, where Θ and Ξ represent the set of all the θ's and ξ's.

Usually, a trained model performs better on the training data than the unseen data, which is called *overfitting*. As a measure of overfitting, the generalization gap is defined as follows:

Given a loss function $\mathscr{L}(f_\Theta(X), Y)$, where f_Θ is the network on dataset (X, Y), the generalization gap is defined

as the difference between the training error and the true error: $\mathscr{L}(f_\Theta(X_{true}), Y_{true}) - \mathscr{L}(f_\Theta(X_{train}), Y_{train})$. In practice, the generalization gap is approximated by the difference between the training error and the test error: $\mathscr{L}(f_\Theta(X_{test}), Y_{test}) - \mathscr{L}(f_\Theta(X_{train}), Y_{train})$

Note here the training loss $\mathscr{L}(f_\Theta(X_{train}), Y_{train})$ is measured with the complete model f_Θ instead of dropout model $f_{\Theta,\Xi}$. The dropout loss used during training $\mathscr{L}(f_{\Theta,\Xi}(X_{train}), Y_{train})$ is usually bigger than $\mathscr{L}(f_\Theta(X_{train}), Y_{train})$ due to information loss. If not specified, we will always use $\mathscr{L}(f_\Theta(X), Y)$ for evaluation, but Θ could be obtained by minimizing $\mathscr{L}(f_{\Theta,\Xi}(X_{train}), Y_{train})$ (trained with dropout) or $\mathscr{L}(f_\Theta(X_{train}), Y_{train})$ (trained without dropout). We define the regularization effect as follows: if a model trained with a method has a smaller generalization gap than the model trained without it, the method has a regularization effect on the model with the specific experimental settings.

1.4. Outline

This thesis is structured as follows:

- Chap. 2: Related works are discussed, including deep neural networks, dropout, and the complexity measures in deep learning.
- Chap. 3: Some regularization behaviors of dropout are summarised with empirical results.
- Chap. 4: The average effect of dropout can be written as a data-dependent regularization term in the loss function.
- Chap. 5: Model's complexity is found to have the same trends as dropout's performance. The potential reason how dropout improves complex models is discussed.
- Chap. 6: Further discussion on why dropout does not work with the small datasets. Intuitive explanations are provided.
- Chap. 7: Dropout's effect on the robustness of the autoencoders.
- Chap. 8: Conclude all the findings and discussion of the future work.

2

Related works

Some previous works have laid the foundation. In this chapter, we will shortly discuss those relevant topics from the literature.

2.1. Deep neural networks

Deep learning has been empirically found great success in many practical fields, including image classification, speech recognition, and bioinformatics. LeCun et al. [15] had summarized the architecture of a deep neural network as "a multilayer stack of simple modules". The modules, often referred to as neurons, only realize simple functions. But with multiple layers of non-linear modules, a network can implement extremely intricate functions.

Although powerful in many application fields, deep neural networks often suffer from the overfitting problem. It is very common that a deep learning model has more trainable parameters than the number of training samples. Zhang et al. [28] showed that deep convolutional neural networks could even fit a training set with all the labels randomly shuffled. Overfitting would make the models have very different performance on the training data and the unseen data, and more importantly, models usually need to learn some complicated pattern from the noise to achieve a good training performance. Many regularization methods have been proposed to prevent overfitting.

2.2. Dropout

Dropout was first proposed by Hinton et al. [8], Srivastava et al. [23] as a simple way to prevent overfitting. The idea is to randomly remove some neurons from the networks at each training step. The activation of those "dropped" neurons would be set to 0; thus the parameters connected to such neurons would not contribute to the network. The activations of all the other neurons would be re-scaled. When the training is finished, all the neurons and their connections will be included in the model to make predictions on test data. This method is also referred to as the standard dropout.

Srivastava et al. [23] proposed a theory inspired by sexual reproduction to explain the superiority of dropout. They argue that a hidden unit in a neural network trained with dropout has to learn to work independently since all the other units could be randomly dropped. Co-adaptation, defined as "a feature detector is only helpful in the context of several other specific feature detectors" by Hinton et al. [8], is therefore reduced. Hinton et al. [8] also suggested that the neural network trained with dropout approximates the expectation of a large number of potential realizations of dropout neural networks. This method is similar to bagging, where many networks are trained separately, and the average of their outputs is used for prediction.

These explanations did not completely satisfy the curiosity of the deep learning community, thus many researchers have been trying to understand the mechanism of dropout further. Labach et al. [14] summarized

some of the interpretations of dropout in their survey. The early studies of dropout focus on its average effect. Baldi and Sadowski [2] tried to study the average effect of dropout on the outputs of neurons. This approach inspired many researchers to study dropout from the average effect perspective. Wager et al. [26] formalized the average effect of dropout as a penalty term in the loss function, which is equal to a l_2-norm regularizer when the input data is normalized. Helmbold and Long [7] further discovered some behaviors of dropout as a regularizer. They find the dropout penalty term is not necessarily monotonic as individual weights increase from 0. Also, they found the penalty term is not always convex, which is different from some conventional regularizers, like weight decay.

Another interpretation links dropout to Bayes neural networks. Bayes models assume a prior distribution over the parameters, and the posterior distribution of the parameters is computed or approximated with a training set. These methods are usually expensive. Gal and Ghahramani [6] argued that dropout is a simple way to approximate a deep Gaussian process. A deep Gaussian process would produce a probability distribution of the prediction, which could be estimated by applying dropout at the test stage. This method is called Monte Carlo dropout, where the uncertainty of the model's prediction can be estimated by the variance of the distribution. Their work triggered the studies relating dropout to Bayesian neural networks.

Srivastava [22] showed how standard dropout help to regularize deep neural networks. Different variants of dropout have been proposed since then. Wan et al. [27] propose *DropConnect*, which randomly drop *weights* instead of *activations* of units like standard dropout. Ba [1] proposed *standout*, where dropping probability of each layer can be optimized like weights. Also, dropout methods have expanded to a variety of applications beyond regularization. Gal and Ghahramani [6] proposed that applying dropout at the test stage could be an easy way to realize a Bayesian neural network. This work inspired other works like *variational dropout* proposed by Kingma et al. [12], where the dropout probability is learned in the Gaussian dropout to make a better model. In this thesis, we focus on understanding the effect of standard dropout.

2.3. Complexity

In the classical complexity theory, the *hypothesis-space complexity* defined by Kawaguchi et al. [10] considers the worst-case gap for the function that could found in f. An upper bound is usually provided for the generalization gap of f_Θ for $\forall\Theta$. A widely-used one is the Rademacher complexity proposed by Shalev-Shwartz and Ben-David [20], which guarantees the performance of a network when the parameters are not determined.

In deep learning, empirical results suggest that two specified models from the same network structure could have completely different performances. Zhang et al. [28] found a typical deep Residual network could make 100% training accuracy while all training samples are randomly labeled. This result suggests the incompetency of the classical complexity measures, which do not depend on the values of trainable parameters. Kawaguchi et al. [10] argued that complexity measures in deep learning should measure the complexity of a specific model with trainable parameters determined. Neyshabur et al. [17] define the *complexity measure* as a mapping that assigns a non-negative number to every specified model f_Θ. For clarification, we use the word *complexity* in the further sections to refer to this definition in this thesis report. Several complexity measures have been proposed in recent years. Most of them can be categorized into three classes: norms, sharpness, and sensitivity.

The norm of weights is long being used to reflect overfitting[13]. They tried to regularize models by punishing large weights based on the assumption that large values of parameters could result in high generalization error. Neyshabur et al. [16] proved that the norm-based regularization method could efficiently control the model's capacity by bounding the Rademacher complexity. They show that l_p-norm regularization can only control the capacity independent of network size when $p \leq 2$. Neyshabur et al. [17] further proposed to use a normalized l_1 and l_2-norm to measure the complexity of the specified model.

Complexity is not only related to the networks but also the task. Dataset can also be included in the complexity measure. One of the most popular data-dependent measures is the flatness in the parameter space. The idea of flatness was first proposed by Hochreiter and Schmidhuber [9]. They defined *flat minima* in the parameter space where the weight vectors within the region have similar small errors. Keskar et al. [11] argued that sharp minima are more likely to be bad minima. They also proved that stochastic gradient descent is effective in avoiding sharp minima. Dinh et al. [5] proposed a scaling method to manipulate the sharpness

of the model, which proved that a sharp minimum could also generalize. Tsuzuku et al. [24] proposed a new sharpness measure of *normalized sharpness* derived from the PAC-bound.

Dimopoulos et al. [4] proposed that the first and second-order derivatives on input could reflect the complexity of the model. Recently, the Jacobian norm has been used as a regularizer in many researches. Sokolić et al. [21] proposed to add the Frobenius norm of the Jacobian norm matrix in the loss function as a regularization term. Even not usually applied as a complexity measure, the Jacobian norm is found related to the generalization gap by Novak et al. [19].

3

Regularization Behaviors of Dropout

In this chapter, some empirical results of dropout will be discussed. Specifically, we will focus on how dropout affects the generalization gap and the test accuracy. Some hyper-parameters are found related to the performance of dropout: dataset size, network width, and data dimension. Possible explanations of how they affect dropout would be discussed in the next chapters.

Our experiments are based on three artificial datasets, marked as 2-d Gaussian, 10-d Gaussian, and 10-d noise. All three datasets are binary classification tasks where $\mathscr{Y} = \{0, 1\}$. In each dataset, the input features $\mathscr{X}$ are generated from two Gaussian distributions. Distributions in 2-d Gaussian center at $\mu_0 = [1, 1]$ and $\mu_1 = [-1, -1]$. Distributions in 10-d Gaussian center at $\mu_0 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$ and $\mu_1 = [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1]$. Distributions in 10-d noise center at $\mu_0 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$ and $\mu_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, -1]$. All the Gaussian distributions have $\sigma = 4I$, where I is the identity matrix in the corresponding dimension.

3.1. Dropout and dataset size

Our first experiment is to confirm if the pattern found in Figure 1.1 appears in other tasks. Different sizes of subsets of the 2-d Gaussian dataset and the 10-d Gaussian dataset are used to train the neural networks in this experiment. Figure 3.1 shows the performance of models trained with or without dropout.

In the right figures of 3.1, there is a clear trend that dropout does not improve the performance when the dataset is either too small or too large. Such trend is not so significant in the left figures of 3.1, where the 2-d Gaussian dataset is used. Also, the range of dataset size where dropout performs the best changes from task to task. For 2-d Gaussian data, dropout stops working when 500 samples are used for training, but much more is needed for the 10-d Gaussian data task before dropout stops working. It suggests dropout may not work well on the 2-d Gaussian set when the training samples are less than 10. We also notice that the generalization gap does not always agree with the test performance, which means reducing overfitting does not necessarily make it a better model (for example, in the left figures in 3.1, dropout makes both test accuracy and generalization gap lower when the dataset size is 10).

3.2. Overfitting of dropout models

In the right figures of Figure 3.1, both generalization gap and test accuracy are not improved by dropout when the dataset is too small. In this section, we will study overfitting from a different angle. We use dropout in a very noisy, low sample-size problem to investigate how it behaves in an overfitting scenario. In the 10-d noise dataset, only the last feature is informative to create a linear boundary, while the first nine features are pure noise. This dataset provides a new measure of overfitting: how much the model relies on the noisy features.

Figure 3.2 shows the performance of the models trained with and without dropout. The trends of their performance are similar to the ones shown in Figure 3.1. The test accuracy is relatively lower than the one on the

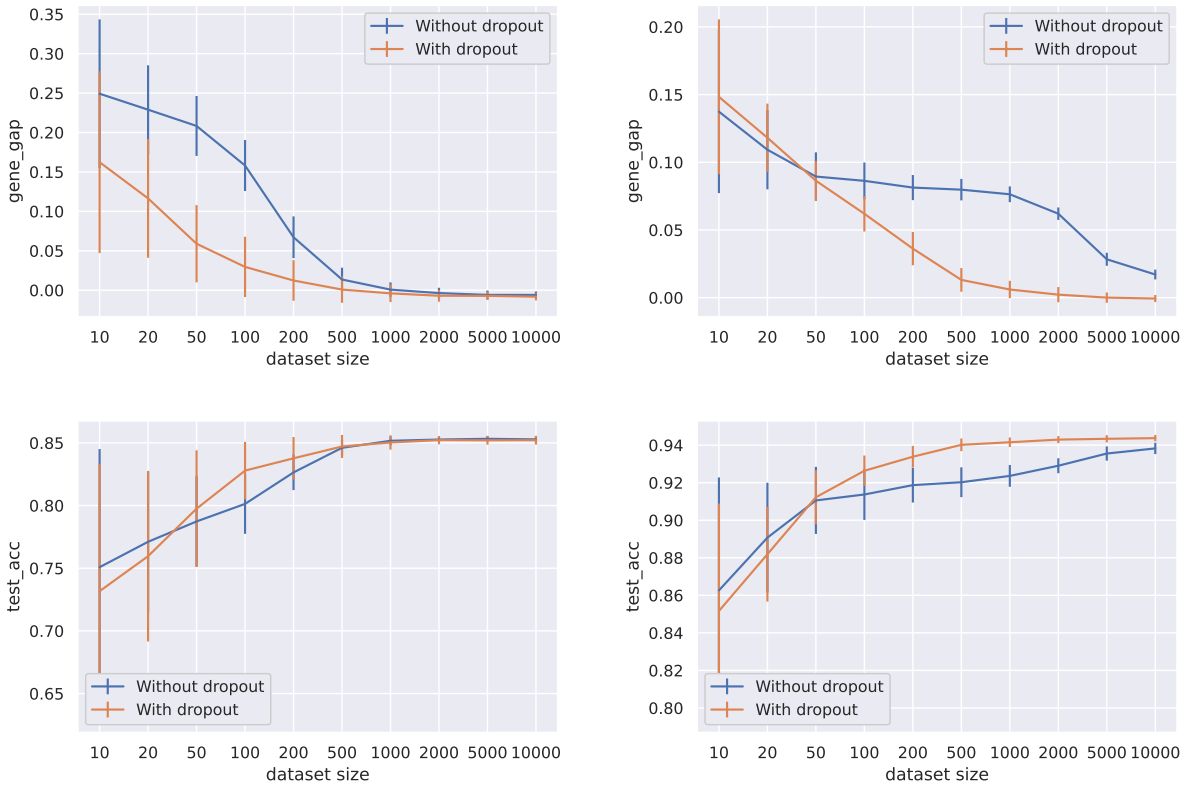


Figure 3.1: Top figures: generalization gap vs. training set size. the number of samples used to train the model. Bottom figures: test accuracy vs. training set size. The generalization gap is defined as the difference between the training and the testing accuracy. Left models are trained on the 2-d Gaussian dataset, where $[d_0, d_1, d_2, d_3] = [2, 100, 100, 2]$. Right models are trained on the 10-d Gaussian dataset, where $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. For all models, dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. Error bars represent the standard deviations of 100 models. The batch size is fixed as 10.

10-d Gaussian dataset, due to the high overlapping samples in the 10-d noise dataset.

An interesting study of the 10-d noise dataset is to see how much the models depend on the noisy features, which we know is useless for the classification. To measure the dependence on the noisy features, some sub-networks are taken from the trained models, where the informative features are ignored in the first layer of the networks. The learning curves of the sub-networks are shown in Figure 3.3, where "with dropout" and "without dropout" indicate if the sub-networks are taken from the complete networks trained with dropout. It shows that models could rely on noisy input features when the dataset size is small and reduce the dependence on those features as the dataset size grows. The effect of dropout can be summarized as follows: when the dataset is large enough, dropout can significantly reduce the influence of the noise features. However, dropout appears to makes the model rely more on the noise features when the dataset is small. It suggests that under certain circumstances, dropout could even increase overfitting.

3.3. Dropout and network size

At each training step, dropout deactivates some neurons in each layer of the networks. As a result, the number of active neurons is reduced at each training step. Naturally, we want to see how the number of neurons in the networks influences the performance of dropout. In this experiment, networks with different widths would be trained with dropout, where width is defined as the number of neurons in each layer.

Figure 3.4 shows the results of the networks trained on 10-d noise dataset. The network width is chosen in $\{10, 100, 1000\}$. In all three networks, dropout only improves the performance within a certain range of dataset sizes. This range changes with the network width: when the network has 10 neurons in each hidden layer, about 100 training samples would be enough for dropout to make an improvement; when the network

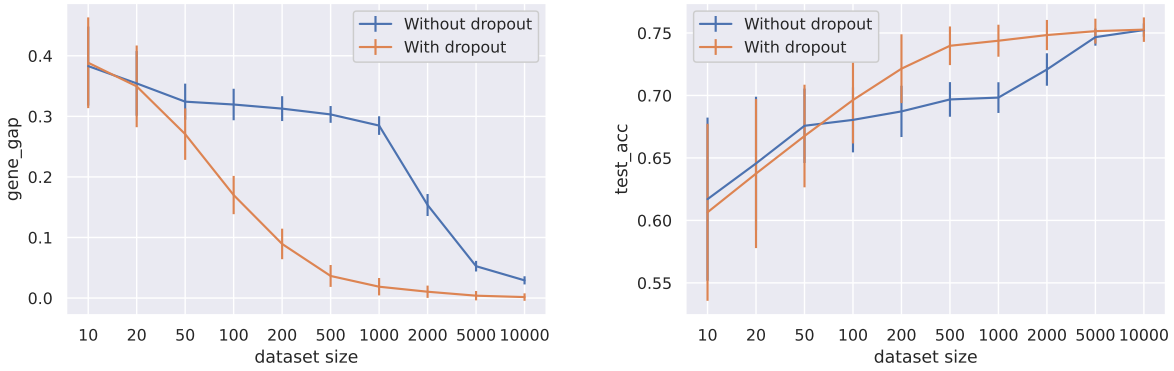


Figure 3.2: Left figures: generalization gap vs. training set size. the number of samples used to train the model. Right figures: test accuracy vs. training set size. The generalization gap is defined as the difference between the training and the testing accuracy. The models are trained on the 10-d noise dataset, where $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. Dropout rate is set as $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. Error bars represent the standard deviations of 100 models. The batch size is fixed as 10.

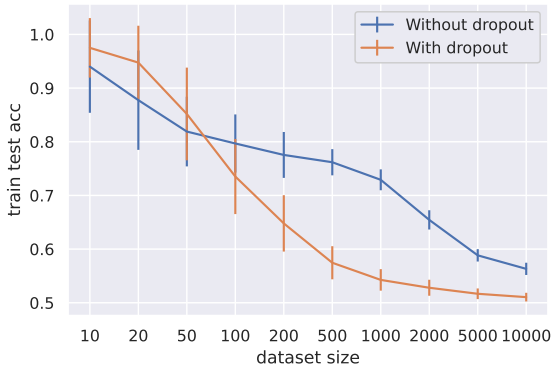


Figure 3.3: Training accuracy of the sub-network vs. training set size. The models were trained on 10-d noise dataset, when all 10 features of the input data are used for training. A sub-network is then taken from each trained model so that the last informative feature will be ignored. The sub-networks have $[d_0, d_1, d_2, d_3] = [9, 100, 100, 2]$. The error bars represent the standard deviations of 100 models. The dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$.

has 1000 neurons in each hidden layer, it needs at least 1000 samples before dropout starts to be a good regularizer. This trend is consistent with the performance of the models trained without dropout. For those models trained on 1000 samples, the generalization gap is below 0.1 when the network's width is 10 and about 0.3 when the network's width is 1000. Another way to interpret such trends is: if it takes more training samples to solve the overfitting of the original model, it also requires more training samples to make dropout works.

3.4. Summary

We summarize the regularization behaviors of dropout as follows:

1. In most cases, dropout improves performance over the basic model within a range of dataset sizes. Too many or too few training samples would make the improvement less significant. Sometimes the model trained with dropout is even worse than the original one.
2. When the input features include noise, dropout does not necessarily suppress the deteriorating effect of such noise features. When the dataset is small, dropout could even make the models overfit more on the noise.
3. The range of dataset sizes where dropout works varies from task to task. When the network size is too big, dropout only regularizes the model when many training samples are used.

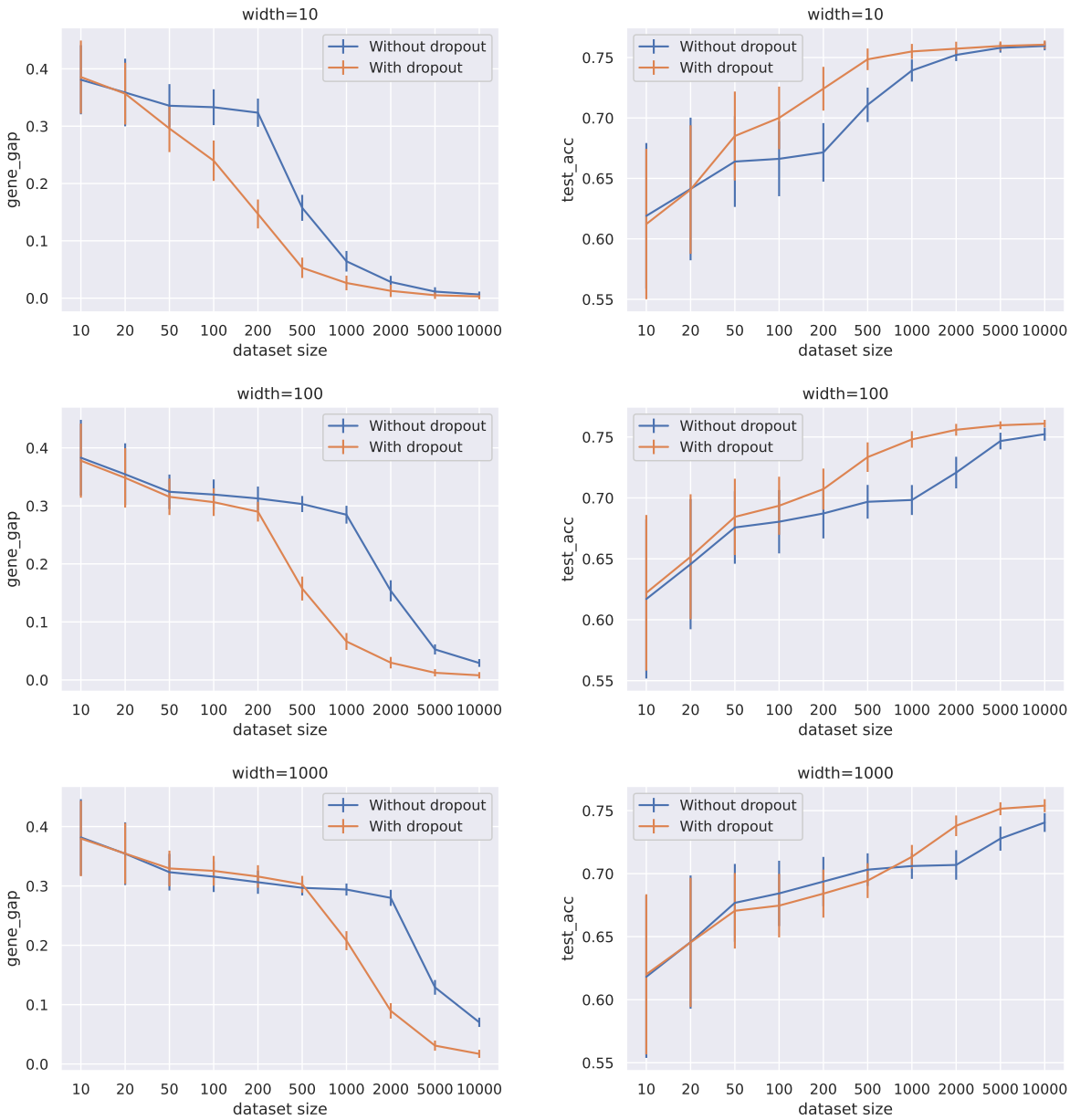


Figure 3.4: Left figures: generalization gap vs. training set size. The generalization gap is the difference between the training and the testing accuracy. Right figures: test accuracy vs. training set size. All models are trained on the 10-d noise dataset. Top models have $[d_0, d_1, d_2, d_3] = [10, 10, 10, 2]$. Middle models have $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. Bottom models have $[d_0, d_1, d_2, d_3] = [10, 1000, 1000, 2]$. For all models, dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. The batch size is fixed as 10. The left figures show the generalization gap, which is the difference between the training and the testing accuracy. Error bars represent the standard deviations of 100 models.

In the following chapters, we would try to find the reasons behinds those behaviors.

4

Dropout: A Data-dependent Regularizer

A key finding from the last chapter is that dropout does not always work when there is overfitting; a reasonably large dataset has to be used for training so that dropout could help. In this chapter, we will study dropout's dependence on the training set.

4.1. Penalty term

Helmbold and Long [7], Wager et al. [26] proposed to interpret the effect of dropout by analyzing the difference between the expected loss with dropout and the one without it. The difference turns out to be a data-dependent penalty term. We formally define the penalty term as the difference between the *original loss* and the *dropout loss* on a predefined dataset (X, Y): $Reg_{(\Xi,\Theta)}(X, Y) = E_{\Xi}[\mathscr{L}(f_{\Theta,\Xi}(X), Y)] - \mathscr{L}(f_{\Theta}(X), Y)$. Assume in a linear regression task on (X, Y) of size n, the *original loss* is defined as: $\mathscr{L}(f_{\Theta}(X), Y) = \frac{1}{n}\sum_i^n(\sum_j^{d_0}\theta_j X_{ij} + b - Y_i)^2$, where X_{ij} is the j-th feature of the i-th data sample of $X \in \mathbb{R}^{n\times d_0}$, and θ_j, b represent the weight and bias for the model. If a dropout layer is applied by replacing X_{ij} by $\xi_j X_{ij}$ for every sample i, the *dropout loss* would be $\mathscr{L}(f_{\Theta,\Xi}(X), Y) = \frac{1}{n}\sum_i^n(\sum_j^d \theta_j\xi_j X_{ij} + b - Y_i)^2$. The effect can then be derived from definition:

$$
\begin{aligned}
Reg_{(\Xi,\Theta)}(X, Y) &= E_{\Xi}[\frac{1}{n}\sum_i^n(\sum_j^d \theta_j\xi_j X_{ij})^2] - \frac{1}{n}\sum_i^n(\sum_j^d \theta_j X_{ij})^2 \\
&= \frac{1}{n}\sum_i^n\sum_j^d((1-p)\times(\frac{1}{1-p})^2 - 1)(\theta_j X_{ij})^2 \\
&= \frac{1}{n}\sum_i^n \frac{p}{1-p}\sum_j^d(\theta_j X_{ij})^2
\end{aligned}
\tag{4.1}
$$

4.2. Empirical results

To check if the effect of dropout can be captured by the penalty term, we perform the experiments on a regression task with 8-dimensional features. California housing prices dataset is obtained from []pace1997sparse. The input data is 20640 samples, each with 8 features.

The purpose of the experiments is to confirm that $Reg_{(\Xi,\Theta)}(X, Y)$ derived in equation 4.1 has the same effect on the results as dropout with the corresponding probability. We use the simplest network with $[d_0, d_1] = [8, 1]$. The prediction can be wirtten as $f_{\Theta}(X) = \left(X\xi^{(0)}\theta^{(1)}\right)$, where θ is a 8×1 vector. No activation is used in this network. The dropout probability $p^{(0)}$ is chosen from $\{0.25, 0.5\}$. To compare with the dropout networks, another set of networks are trained with the penalty term in the loss function:

$$\mathscr{L}(f_{\Theta,\lambda}(X),Y)=\frac{1}{n}\sum_{i}^{n}(\sum_{j}^{d_0}\theta_j X_{ij}+b-Y_i)^2+\frac{1}{n}\sum_{i}^{n}\lambda\sum_{j}^{d}(\theta_j X_{ij})^2 \tag{4.2}$$

Where λ is the weight for the penalty term. From equation 4.1 we know the penalty term should have the same effect as dropout if $\lambda=\frac{p}{1-p}$. Hence, the penalty term weight λ is chosen from $\{0.2,0.5\}$.

The results are presented in Figure 4.1, which shows that a penalty term computed in equation 4.1 has an almost identical effect as the corresponding dropout when the dataset size is big enough. It suggests that the effect of such a penalty term could represent the effect of dropout on a simple model.

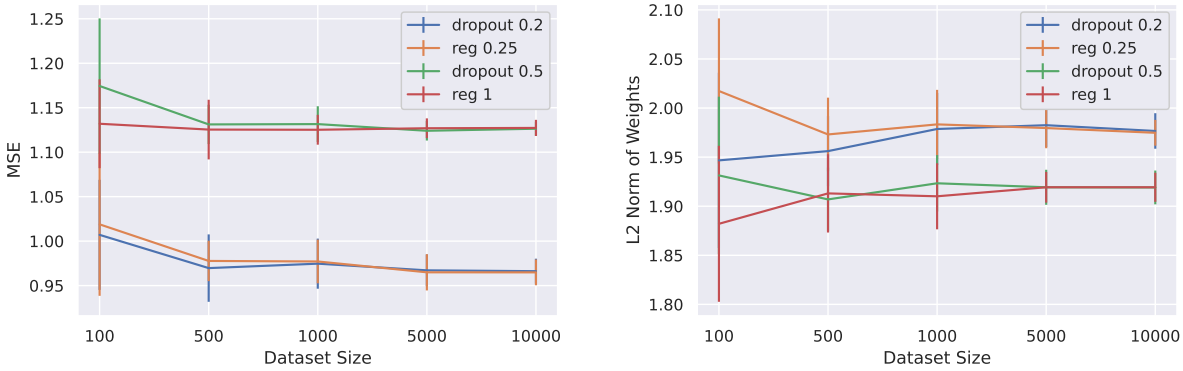


Figure 4.1: Left figure: test mean squared error vs. training dataset size. Right figure: norms of weights in the networks vs. training dataset size. Experiments on California housing price dataset. Blue and green curves represent the results from models trained with dropout. Yellow and red curves represent the results from models trained with the penalty term defined in equation 4.2.

4.3. Complex model

Since Ξ is the set of all random variable ξ's, the expectation over Ξ should consider every possible realization of Ξ. Each $\xi^{(l)}$ can be chosen from $0,\frac{1}{1-p^{(l)}}$, so the number of total possibilities is $2^{|\Xi|}$. When there are many neurons, the penalty term would be very hard to analyze. In the last section, we only consider a linear model and mean squared error loss, which allow the penalty term to be written in a simple form as in equation 4.1. However, the penalty term could be very complicated with a more general setting. Consider a logistic regression model on 2-dimensional inputs: $f_{\theta_1,\theta_2}(x)=\frac{1}{1+e^{-(\theta_1x_1+\theta_2x_2)}}$, the dropout model would be $f_{\theta_1,\theta_2,\xi_1,\xi_2}(x)=\frac{1}{1+e^{-(\theta_1x_1\xi_1+\theta_2x_2\xi_2)}}$. The difference would be:

$$\begin{aligned}
&E_{\xi_1,\xi_2}[\frac{1}{1+e^{-(\theta_1x_1\xi_1+\theta_2x_2\xi_2)}}-\frac{1}{1+e^{-(\theta_1x_1+\theta_2x_2)}}]=\\
&-\frac{p\left(p\left(-e^{\theta_1x_1+\theta_2x_2}\right)+pe^{\frac{\theta_1x_1}{1-p}+\theta_1x_1+\theta_2x_2}+2e^{\theta_1x_1+\theta_2x_2}+pe^{\frac{\theta_1x_1}{1-p}}-2e^{\frac{\theta_1x_1}{1-p}}-p\right)}{2\left(e^{\theta_1x_1+\theta_2x_2}+1\right)\left(e^{\frac{\theta_1x_1}{1-p}}+1\right)}\\
&-\frac{(1-p)\left(p\left(-e^{\theta_1x_1+\frac{\theta_2x_2}{1-p}+\theta_2x_2}\right)+pe^{\frac{\theta_1x_1}{1-p}+\frac{\theta_2x_2}{1-p}}+pe^{\frac{\theta_1x_1}{1-p}+\theta_1x_1+\frac{\theta_2x_2}{1-p}+\theta_2x_2}+e^{\theta_1x_1+\frac{\theta_2x_2}{1-p}+\theta_2x_2}\right)}{\left(e^{\theta_1x_1+\theta_2x_2}+1\right)\left(e^{\frac{\theta_2x_2}{1-p}}+1\right)\left(e^{\frac{\theta_1x_1}{1-p}+\frac{\theta_2x_2}{1-p}}+1\right)}\\
&-\frac{(1-p)\left(-e^{\frac{\theta_1x_1}{1-p}+\frac{\theta_2x_2}{1-p}}-e^{\frac{\theta_1x_1}{1-p}+\frac{2\theta_2x_2}{1-p}}+e^{\theta_1x_1+\theta_2x_2}-pe^{\frac{\theta_2x_2}{1-p}}\right)}{\left(e^{\theta_1x_1+\theta_2x_2}+1\right)\left(e^{\frac{\theta_2x_2}{1-p}}+1\right)\left(e^{\frac{\theta_1x_1}{1-p}+\frac{\theta_2x_2}{1-p}}+1\right)}
\end{aligned} \tag{4.3}$$

It gets simpler when we take the logarithm, which is the difference of the logistic loss:

$$
\begin{aligned}
&E_{\xi_1,\xi_2}[\log(\frac{1}{1+e^{-(\theta_1 x_1 \xi_1+\theta_2 x_2 \xi_2)}})-\log(\frac{1}{1+e^{-(\theta_1 x_1+\theta_2 x_2)}})]= \\
&(1-2p+p^2)\log\left(\frac{1}{e^{\frac{\theta_1 x_1+\theta_2 x_2}{p-1}}+1}\right)-\log\left(\frac{e^{\theta_1 x_1+\theta_2 x_2}}{e^{\theta_1 x_1+\theta_2 x_2}+1}\right) \\
&+(p-p^2)\left(\log\frac{1}{e^{\frac{\theta_2 x_2}{p-1}}+1}+\log\frac{1}{e^{\frac{\theta_1 x_1}{p-1}}+1}\right)-p^2\log(2)
\end{aligned}
\tag{4.4}
$$

We conclude that with any non-linearity in the network or the loss function, the number of terms in $Reg_{(\Xi,\Theta)}(X,Y)$ would be exponential in $|\Xi|$. It will be very hard to give the analytic expression of the dropout penalty term for a modern deep neural network, where there could be thousands of neurons in a simple model.

4.4. Effect of the dropout regularizer

Wager et al. [26] showed that if the input data is normalized, 4.1 can be reduced to the L_2-regularization. If the data features are not normalized, the dropout regularizer will punish those weights connected to large features more than those connected to the small features. This property could be useful when the informative features are significantly smaller than the noisy features.

In this chapter, we found the average effect of dropout is a data-dependent penalty term, which is different from the classical regularizers like the *LASSO*. The form of the penalty term can be very complicated for large networks, which makes it difficult to understand its exact effect on regularization. What we can know from such penalty terms is that those terms can be overfitted as well. When there are few training samples, parameters in a large network can be carefully arranged to make the penalty term small. When the training set is larger, it would be harder to do so. However, data-dependence of dropout does not explain its regularization behaviors found in Figure 1.1 and 3.1. More evidence is required to understand why dropout does not work when the dataset size is small.

5

Dropout and Complexity

In this chapter, we will study the relationship between dropout and the model's complexity. We will propose an interpretation of the average effect of dropout and explain the empirical results of dropout affecting the model's complexity. All complexity measures we use are defined for specific models.

5.1. Complexity

Overfitting, i.e., when the training performance is much better than the test performance, usually happens when the model's complexity is too high. There are different approaches to define the complexity of a model. One of the most famous method is the *Rademacher complexity* [3]. It can be bound by other parameters of the network. For example, Neyshabur et al. [16] propose a bound on the Rademacher complexity, which includes exponential dependence on the depth of the network.

In recent years, complexity has been studied in different ways. Zhang et al. [28] found Residual Networks could achieve 100% training accuracy on CIFAR-10 even with all the labels shuffled. Since random labels are completely independent of the input images, the network must memorize every training sample. The generalization gaps of such models are apparently large. However, this generalization gap cannot be reflected by the conventional complexity measures; in fact, the same Residual Network trained with true labels could have a very small generalization gap. Researchers are inspired to find the measures to distinguish a network trained on a random labeled dataset and a network trained on the true one [18]. Neyshabur et al. [17] define the *complexity measure* as a mapping that assigns a non-negative number to every specified model f in a model class $\mathscr{F}$ with a specified training set S - $\mathscr{M} : \{\mathscr{F}, S\} \rightarrow \mathbb{R}^+$. In this thesis report, we use the word *complexity* to refer to this definition.

Three criteria are used in this chapter to measure the complexity of a model:

1. **Norms**: norms is defined as $\sum_l \sqrt{\left\|\theta^{(l)}\right\|_{\mathrm{F}}^2}$, where $\|\cdot\|_{\mathrm{F}}^2$ is the Frobenius norm. Krogh and Hertz [13] showed that reducing the norms of weight during training could effectively reduce overfitting, which is usually referred as *weight decay*. In this thesis we assume the norms of weight could reflect the complexity of a specific neural network.

2. **Sharpness**: the definition of sharpness was proposed by Tsuzuku et al. [24], which is the combination of norms and the second-order derivatives of the parameters: $\sum_l \sqrt{\left\|\theta^{(l)}\right\|_{\mathrm{F}}^2 H^{(l)}}$, where $H^{(l)} := \sum_{i,j} \frac{\partial^2 \mathscr{L}(f_\Theta(X),Y)}{\partial \theta_{i,j}^{(l)} \partial \theta_{i,j}^{(l)}}$.

3. **Sensitivity**: the methods proposed by Novak et al. [19] measure the sensitivity: $E_X[\|\mathbf{J}(X)\|_{\mathrm{F}}]$, where $\mathbf{J}(X) = \partial f_\Theta(X)/\partial X^{\mathbf{T}}$. They confirmed that the Jacobian norm can be predictive of the generalization gaps in many tasks.

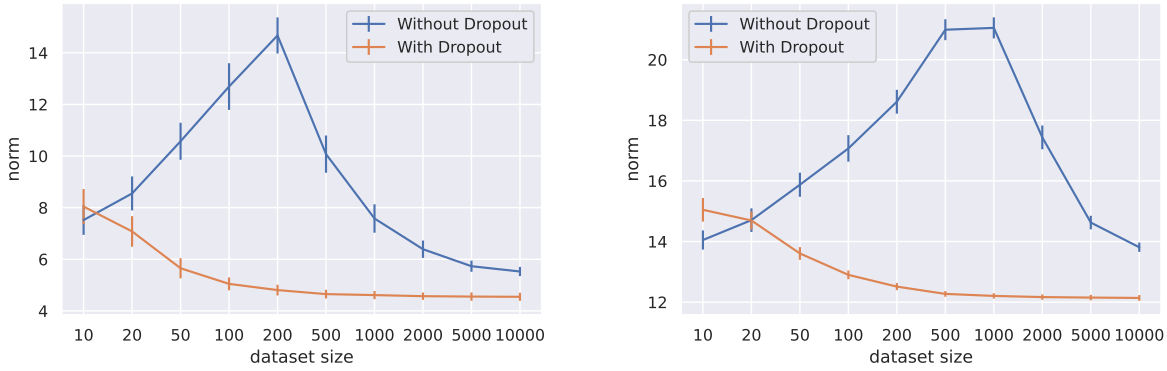


Figure 5.1: Norms of weights in the trained networks vs. training dataset size. Left figure shows the results from the network with width 10, where $[d_0, d_1, d_2, d_3] = [10, 10, 10, 2]$. Right figure shows the results from the network with width 100, where $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. All the networks are trained on 10-d noise dataset. For all models, dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. Error bars represent the standard deviations of 100 models. The batch size is fixed as 10.

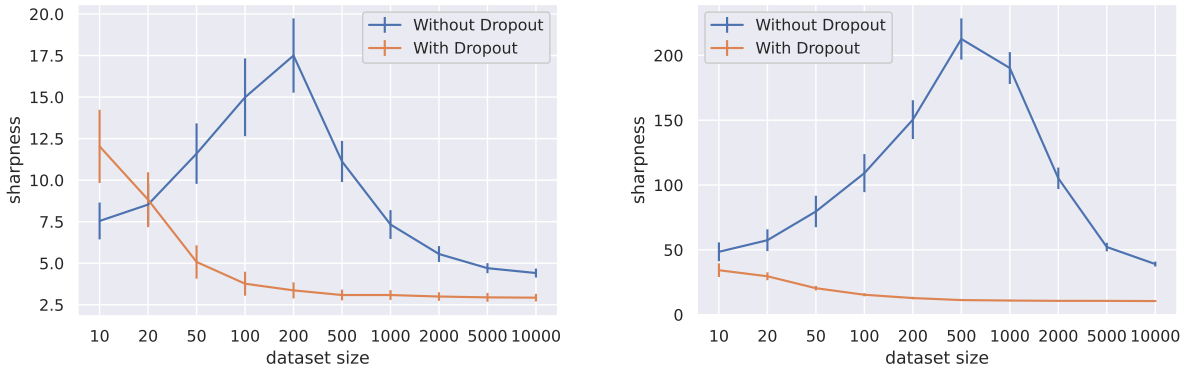


Figure 5.2: Sharpness of the trained networks vs. training dataset size. Left figure shows the results from the network with width 10, where $[d_0, d_1, d_2, d_3] = [10, 10, 10, 2]$. Right figure shows the results from the network with width 100, where $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. All the networks are trained on 10-d noise dataset. For all models, dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. Error bars represent the standard deviations of 100 models. The batch size is fixed as 10.

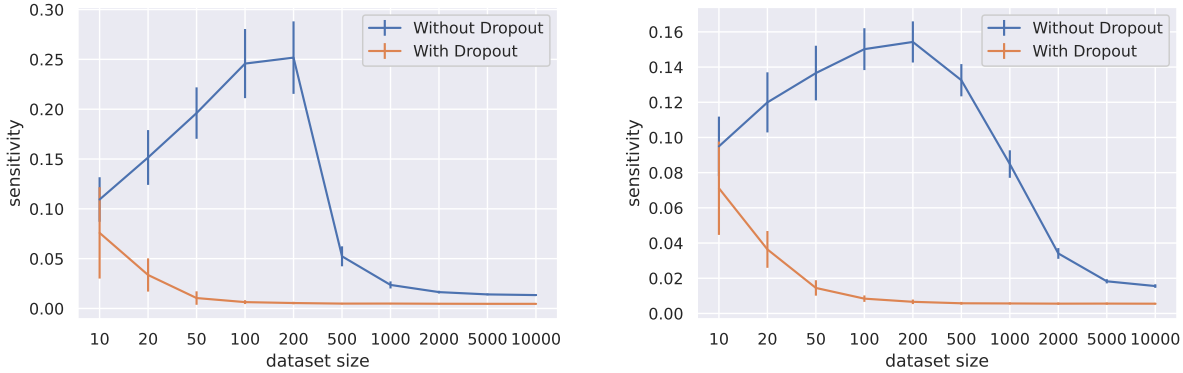


Figure 5.3: Sensitivity of the trained networks vs. training dataset size. Left figure shows the results from the network with width 10, where $[d_0, d_1, d_2, d_3] = [10, 10, 10, 2]$. Right figure shows the results from the network with width 100, where $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. All the networks are trained on 10-d noise dataset. For all models, dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. Error bars represent the standard deviations of 100 models. The batch size is fixed as 10.

Norms, sharpness, and sensitivity vs. the dataset sizes are shown in Figure 5.1, 5.2, and 5.3. All three complex-

ity measures show the same pattern with the dataset size for the models trained without dropout. When the dataset is small, the complexity grows with it; after hitting some peak, it starts to decrease rapidly. The high complexity occurs in a certain range of dataset size, where dropout usually works very well in Figure 3.4. For the models trained with dropout, it seems that complexity always decreases with the dataset size; when the original models are very complex, dropout makes a big difference in the complexity. Otherwise, the model trained with or without dropout has similar complexity, so do the test accuracy and the generalization gap.

Another interesting finding in Figure 5.1, 5.2, and 5.3 is that the shape and position of the peak could be affected by the network size. We found that when the network width is 100, the peak is usually found in a larger dataset than the network with the width of 10. This pattern is also consistent with Figure 3.4 where a large network needs more training samples to make dropout work. Based on the pattern we found in Figure 5.1, 5.2, 5.3 and 3.4, we propose that dropout may only work when the complexity is too high.

5.2. Dropout loss landscape

In the last section, we found some evidence that dropout only works when the model's complexity is high. In this section, some reasons would be discussed. Equation 4.1 interpret the expected effect of dropout as a data-dependent penalty term in the loss function. Same method can be used by studying the expected loss landscape of dropout loss $E[\mathscr{L}(f_{\Theta,\Xi}(X_{train}), Y_{train})]$. Instead of finding an explicit form of the penalty term, this interpretation will rely on the geometry of the dropout loss landscape. The function could therefore be used even for a very complicated network.

The expectation of the dropout loss landscape can be written as:

$$E[\mathscr{L}(f_{\Theta,\Xi}(X_{train}), Y_{train})] = \sum_{\Xi\in\{0,1\}^D} \prod_{\xi^{(l)}} p^{(l)d_l - \sum\xi^{(l)}}(1-p^{(l)})^{\sum\xi^{(l)}} \mathscr{L}(f_{\Theta,\{\xi^{(1)},\ldots,\xi^{(l)}\}}(X_{train}), Y_{train}) \tag{5.1}$$

The right terms in the summation are the scaled loss on the loss landscape of the model trained without dropout. Another interpretation is: a point on the dropout loss landscape is the weighted average of many points on the original loss landscape. A visualization would be helpful to have a clear picture of this.

The loss landscape of the networks defined in preliminaries is too complex to visualize. A neural network with a few layers usually has thousands to millions of weights, making it hard to visualize the full picture of the loss landscape. To see how dropout works, a simple loss function can be defined without a neural network. Given a function with "dropout" applied on two parameters $f_{\theta_1,\theta_2,\xi_1,\xi_2}(x)$, for which the loss function can be denoted as $\mathscr{L} = g(\theta_1\xi_1, \theta_2\xi_2)$, where g is a function we defined to create a loss landscape that we want.

With $\xi_1, \xi_2 \in \{0, \frac{1}{1-p}\}$, we define the loss landscape with dropout as in Figure 5.4. The top left figure shows the original loss landscape with four minima $g(\theta_1, \theta_2)$, while the other figures show the expected dropout loss defined as:

$$g_p(\theta_1,\theta_2) = (1-p)^2 g(\frac{\theta_1}{1-p}, \frac{\theta_2}{1-p}) + p(1-p)g(\frac{\theta_1}{1-p}, 0) + p(1-p)g(0, \frac{\theta_2}{1-p}) + p^2 g(0,0) \tag{5.2}$$

Figure 5.4 shows the dropout landscape of function $g(\theta_1\xi_1, \theta_2\xi_2)$, and Figure 5.5 shows the corresponding loss function on single parameters. In both figures, the whole picture "shrinks" towards the origin. We conclude the effect of dropout on the loss landscape as follows: 1. the loss landscape would "shrink" towards the origin by the factor $\frac{1}{1-p}$. 2. the dropout loss of (θ_1, θ_2) is the weighted average over 2^D projections of the scaled point $(\frac{\theta_1}{1-p}, \frac{\theta_2}{1-p})$ on different coordinate planes and axes of the original landscape. Since dropout is applied on the neurons, the weights from the same column of $\theta^{(l)}$ are dropped together. Thus not every projection in the parameter space contributes to the expected dropout loss.

Visualizing the loss landscape of a real-world neural network is too hard, but it is possible to draw the training loss with respect to a single weight, which could partly reflect the sharpness of the landscape. In figure 5.6 we present the dropout training loss versus a single weight from the trained network. The dropout training losses are approximated by the training losses of 100 realizations of dropout neural networks. All the weights

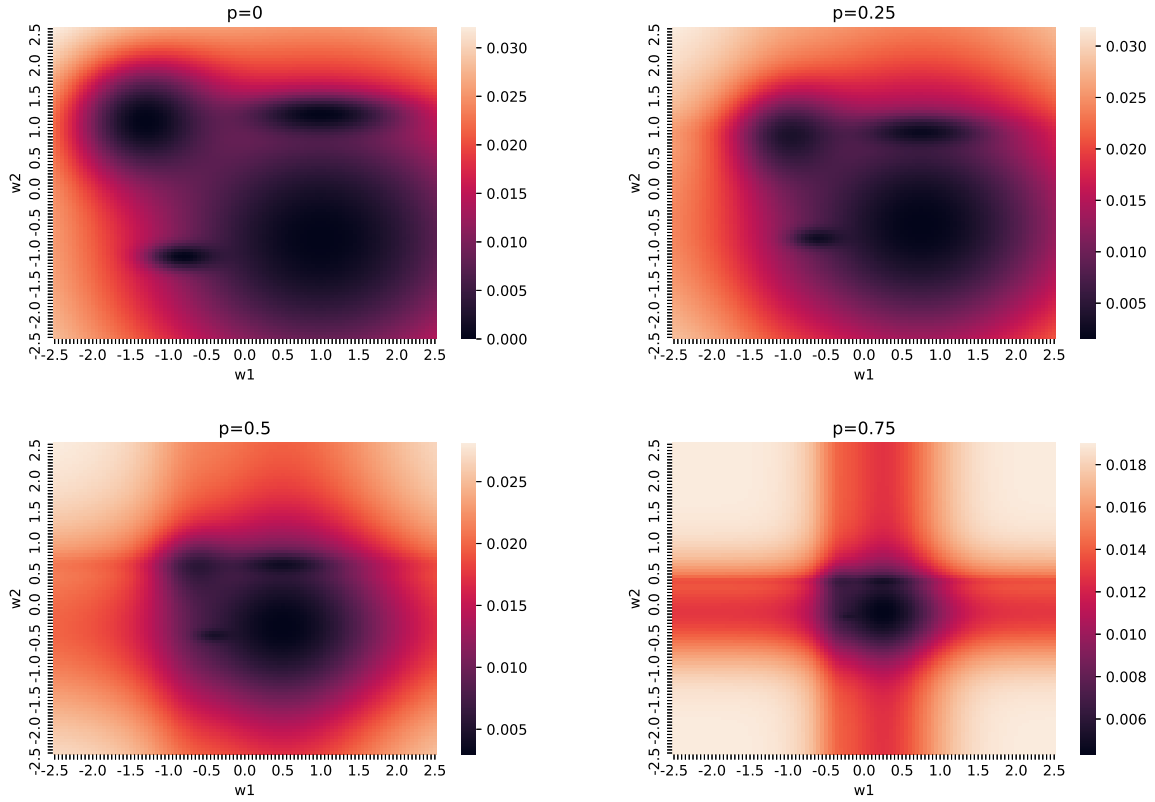


Figure 5.4: Dropout loss landscape $g_p(\theta_1,\theta_2)$ with different drop probabilities p. Top left: p=0, top right: p=0.25. Bottom left: p=0.5, bottom right: p=0.75.

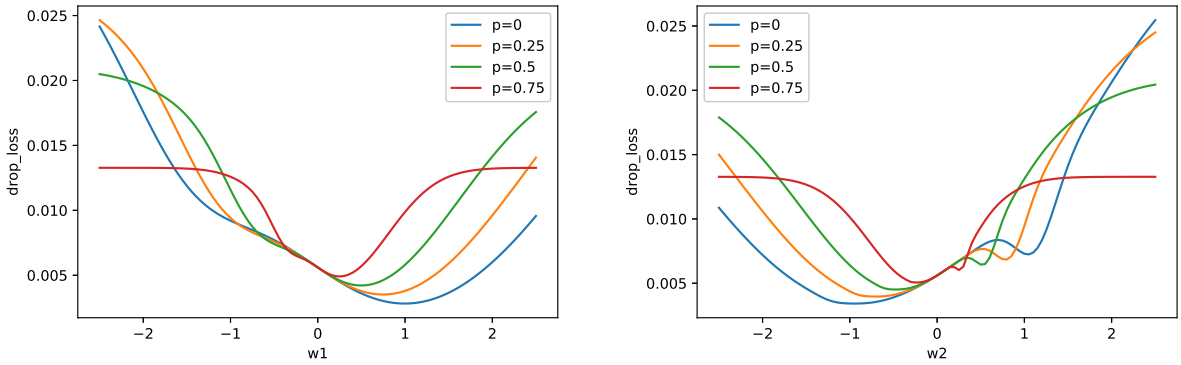


Figure 5.5: Dropout loss $g_p(\theta_1,\theta_2)$ vs. a single weight, when the drop probabilities $p = 0.5$. Left: θ_2 is set as 0. Right: θ_1 is set as 0.

are fixed except one. The weight is chosen from the layer where dropout is applied, which significantly affects the original loss. When the dropout mask Ξ neural network is fixed, this chosen weight varies within [-1.5,1.5] to see how the training loss changes. It appears that when the dropout probability increases, the loss values increased rapidly. Also, the dropout training loss is smoother when the drop probability is high: a minimum in the top left figure of 5.6 is much lower than the neighbors, but in the bottom right figure, this minimum is only a little bit better than the others.

Now we can try to explain the complexity behaviors found in Figure 5.2 by the dropout loss landscape: dropout would bring other projections onto the sharp minima, which would not have a significantly lower loss compared to the neighbors. Therefore, models are less likely to converge to those sharp minima. On the other hand, when the sharpness of the original model is low, minima would not be significantly lower than

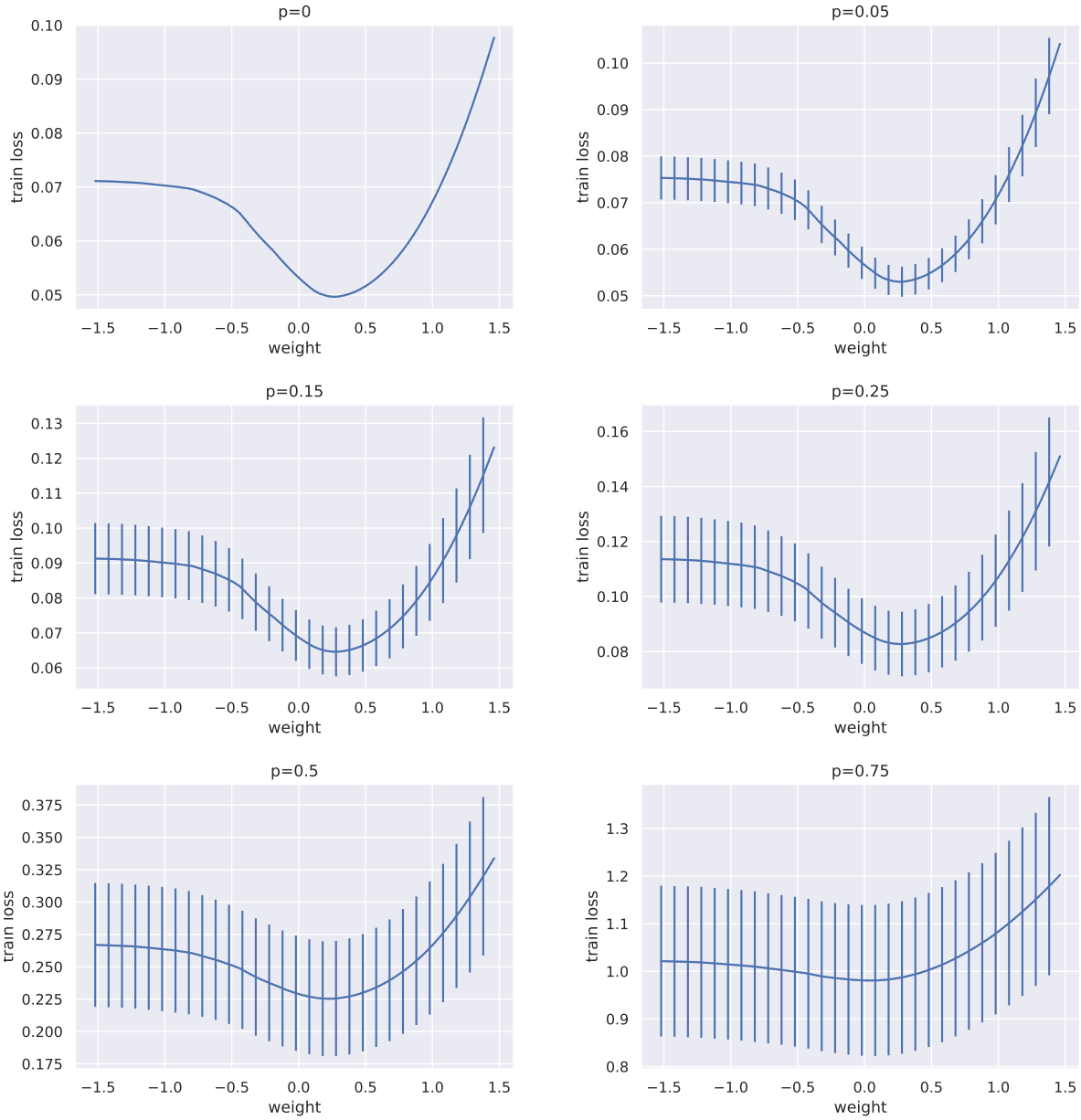


Figure 5.6: Dropout training loss curves vs. a single weight, where the error bars represent the standard deviation created by 100 realizations of the dropout neural network. The network is full-batch trained on a subset of the FashionMNIST without dropout, where $[d_0, d_1, d_2, d_3, d_4] = [784, 64, 32, 32, 10]$. To plot the figures, we drop the second hidden layer with probability p. The weight to vary is chosen from $\theta^{(2)}$, which is the 64×32 weight matrix.

the neighbors. Hence dropout cannot make much difference to the model's performance.

6

Dataset size and Co-adaptation

In this chapter, we study how dataset size affects complexity and co-adaptation. We will also see examples of how dropout affects the classifier and explain the results in the previous chapters.

6.1. Co-adaptation

The complexity pattern found in Section 5.1 is very inspiring. In this section, we will try further to study the relationship between the co-adaptation and dataset size. Hinton et al. [8] suggest that dropout regularizes models by reducing co-adaptation, which is the collaboration between neurons. Evidence found by Srivastava et al. [23] showed that dropout breaks up the co-adaptation by making the presence of any particular neuron unreliable. In our experiment, a simple method is used to measure the co-adaptation: when a model is tested with dropout, the reduction in the performance compared to the testing without dropout.

In Figure 6.1 we present the performance results tested with dropout. The models are trained without dropout, but a certain fraction of the neurons are dropped during the testing. The results show that models trained on small datasets are robust to the dropout testing, while the models trained with big datasets have much worse performance when dropout is used in the testing. More importantly, we can see the co-adaptation starts to decrease when the dataset achieves a certain size (in Figure 6.1, when dataset size> 2000). This behavior is also seen with the generalization gap.

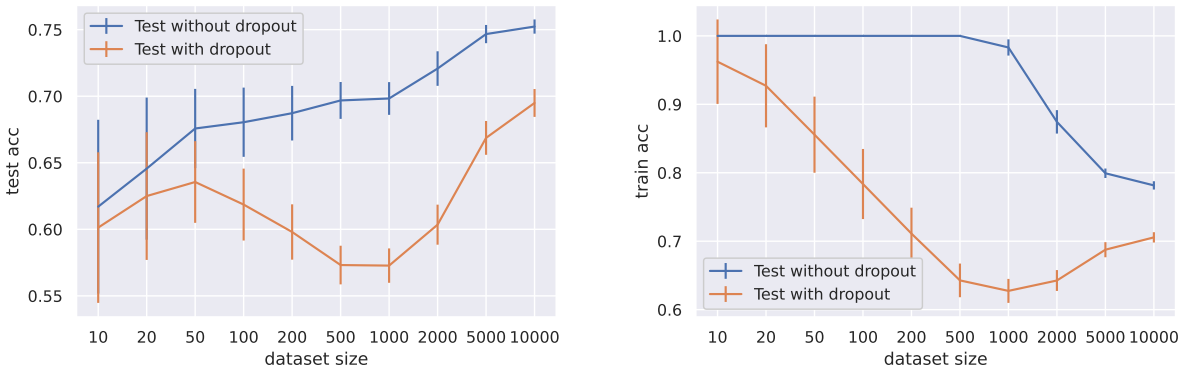


Figure 6.1: Test accuracy vs. training dataset size. In the left figure, the blue curve represent the test accuracy without dropout: $\mathscr{L}(f_\Theta(X_{test}), Y_{test})$, the orange curve represent the test accuracy with dropout: $\mathscr{L}(f_{\Theta,\Xi}(X_{test}), Y_{test})$. In the right figure, the blue curve represent the training accuracy without dropout: $\mathscr{L}(f_\Theta(X_{train}), Y_{train})$, the orange curve represent the test accuracy with dropout: $\mathscr{L}(f_{\Theta,\Xi}(X_{train}), Y_{train})$. All the networks have $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$ and dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. All the networks are trained on the 10-d noise dataset. The batch size used for training is fixed as 10.

In addition to the average reduction in the performance, we present the distribution of the reductions in Figure 6.2. We have trained 100 models with the same settings. Each model is tested with dropout 100 times. We can see that models trained on big datasets are much more sensitive to neuron loss. Like the non-monotonic behavior of co-adaptation, the distribution of the model trained on large datasets will be closer to 0 after some points.

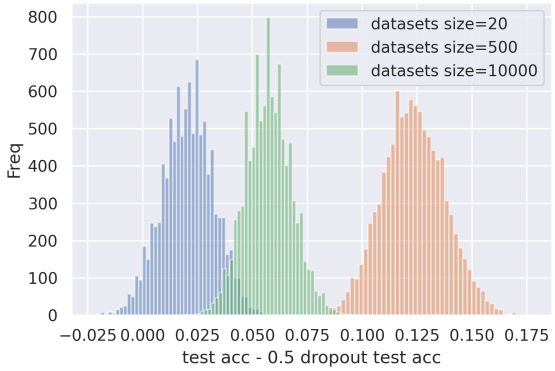


Figure 6.2: Distribution of the test performance reduction because of dropout in testing. The networks are trained on the 10-d noise dataset without dropout, $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. The horizontal axis represents the difference between the original test performance and the one with dropout. The dropout test accuracy is computed with $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0, 0.5, 0.5, 0]$.

Overall, our results show that models trained on small datasets have less functioning neurons, thus, are less sensitive to neuron loss. Figure 6.2 shows a trend that the co-adaptation will decrease even without dropout when the dataset is too big. This pattern is also seen with the complexity in Figure 5.1, 5.2, and 5.3. It gives us a good reason why dropout does not work well with small datasets: the models are not complex enough and thus not very sensitive to neuron loss.

6.2. Boundary visualization

Finally, we want to study how small datasets become the reason that models are not complex. The reason can be easily explained with some examples of decision boundaries. In Figure 6.3 we present the classification surface of models trained on a varying number of samples.

The first thing we learn from Figure 6.3 is that dataset size does affect complexity. We will discuss the left figures from top to bottom: in the top left of Figure 6.3, the classifier trained with only 10 samples is simple but not very accurate; when there are 100 training samples (middle left figure), the boundary becomes more complicated: we can see some "islands" that are mispredicted. Those "islands" are used to adapt to those samples that a Bayes classifier could misclassify. We can imagine those complicated classification boundaries require more neurons to work in different ways. In the end, when the training samples are so abundant that overfitting is no longer an issue, the model makes a boundary very close to the Bayes classifier.

To see the effect of dropout, we will discuss the right figures from bottom to top: the bottom right figure has roughly the same boundary as the bottom left one. The only change dropout brings is making the prediction probability smaller. It is because the left boundary is already very simple. In the middle right figure, we can see a clear picture of how dropout makes it better: The "islands" in the left figure almost disappeared, and the prediction surface seems very smooth and simple. This shows a good example of regularization. In the top right figure, we see that dropout does not make the classifier much better. This is probably because the classifier in the top left figure is simple enough. It appears that dropout can make a complicated model simpler but cannot do much if the model is simple already.

Based on the results above, we make the conclusion as follows:

1. Overfitting small datasets does not require complex models, making the complexity of models trained on small datasets lower than those trained on big datasets; such simple models require only a small fraction of the neurons to work, so it is not very sensitive to neurons loss.

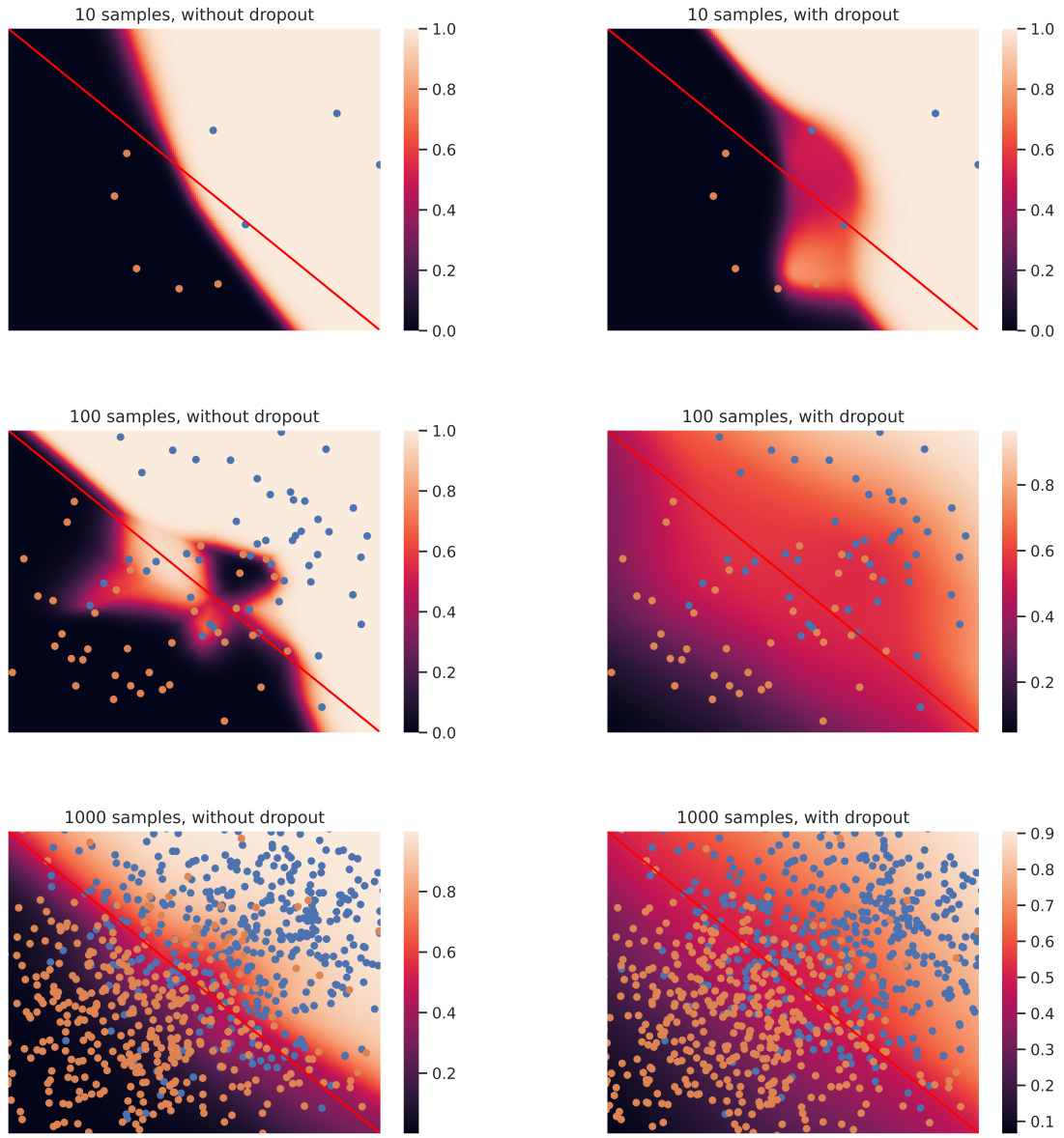


Figure 6.3: Prediction probability surface of the networks trained on the 2-d Gaussian dataset. Heatmap color represents the sigmoid predicting score over the input space. Each axis represents one input feature range from [-3,3]. All the dots indicate the training samples used to train the model, with colors mark their labels. All networks have $[d_0, d_1, d_2, d_3] = [2, 100, 100, 2]$ and dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. The red straight line represents the Bayes classifier. Only one network is presented in each figure, but the pattern is found in most networks trained with the same settings.

2. Dropout only helps when the neurons work with others. If the model's complexity is low, the neurons typically work independently. Therefore, dropout does not help when the dataset is small.

7

Beyond Regularization

Although first proposed as a regularizer, dropout is not only useful in preventing overfitting. In this chapter, dropout's influence on autoencoder networks will be discussed. Some properties can be applied under certain scenarios beyond overfitting.

7.1. Robustness and sensitivity

First proposed as a regularizer, dropout is usually studied in an overfitting situation. On the other hand, some ideas within dropout can also being inspiring in the cases where overfitting is not an issue: adding noise has been long applied in the autoencoder[25], where noise is added in the input and targets to improve the robustness to noise. This application is also discussed as an interpretation of dropout by Srivastava [22]. Naturally, dropout is an easy way to add such noise.

In this section, some examples will be discussed to show that dropout can be applied in the autoencoders to improve the robustness. We trained a simple autoencoder on MNIST, where the network structure is $[d_0, d_1, d_2, d_3, d_4, d_5, d_6] = [784, 256, 64, 16, 64, 256, 784]$. The loss fucntion is defined as the Jacobian norm of the difference between the input and output. Three models are trained in this experiment: one is trained without dropout; one is trained with dropout in the encoder: $[p_0, p_1, p_2, p_3, p_4, p_5, p_6] = [0, 0.5, 0.5, 0, 0, 0, 0]$ and one is trained with dropout in the decoder: $[p_0, p_1, p_2, p_3, p_4, p_5, p_6] = [0, 0, 0, 0, 0.5, 0.5, 0]$. All models are trained on '0' and '1's. The predictions of those three models are shown in Figure 7.1 along with the input sample in the left column. The model trained with dropout in the encoder seems to predict something similar to '0' or '1' all the time, as shown in the second column from the right. The other two models make very noisy predictions on those samples, since those digits are not used in training.

The same experiment on the FashionMNIST is presented in Figure 7.2. The models are trained with 'pullover's and 'coat's of FashionMNIST. Compare to the results in Figure 7.1, dropout is not making a big difference on FashionMNIST predictions. A possible explanation is that FashionMNIST is noisier compared to MNIST. Samples in the same class of cloth could be much different on the pixel level than the digits. As a result, the autoencoder will need to learn to adjust to the noisy input. Another interpretation is that the distribution of 'pullover's and 'coat's are much wider than '0's and '1's in the input space. Therefore, the autoencoders trained on the FashionMNIST dataset will be more active in predicting unseen data.

Our experiments show an interesting function of dropout: it helps the model to adapt to the input noise. The behaviors shown in Figure 7.1 could be applied in some specific tasks, where the model is encouraged to make bold predictions on the unseen data. It requires more experiments to understand this property of dropout and apply it in applications.

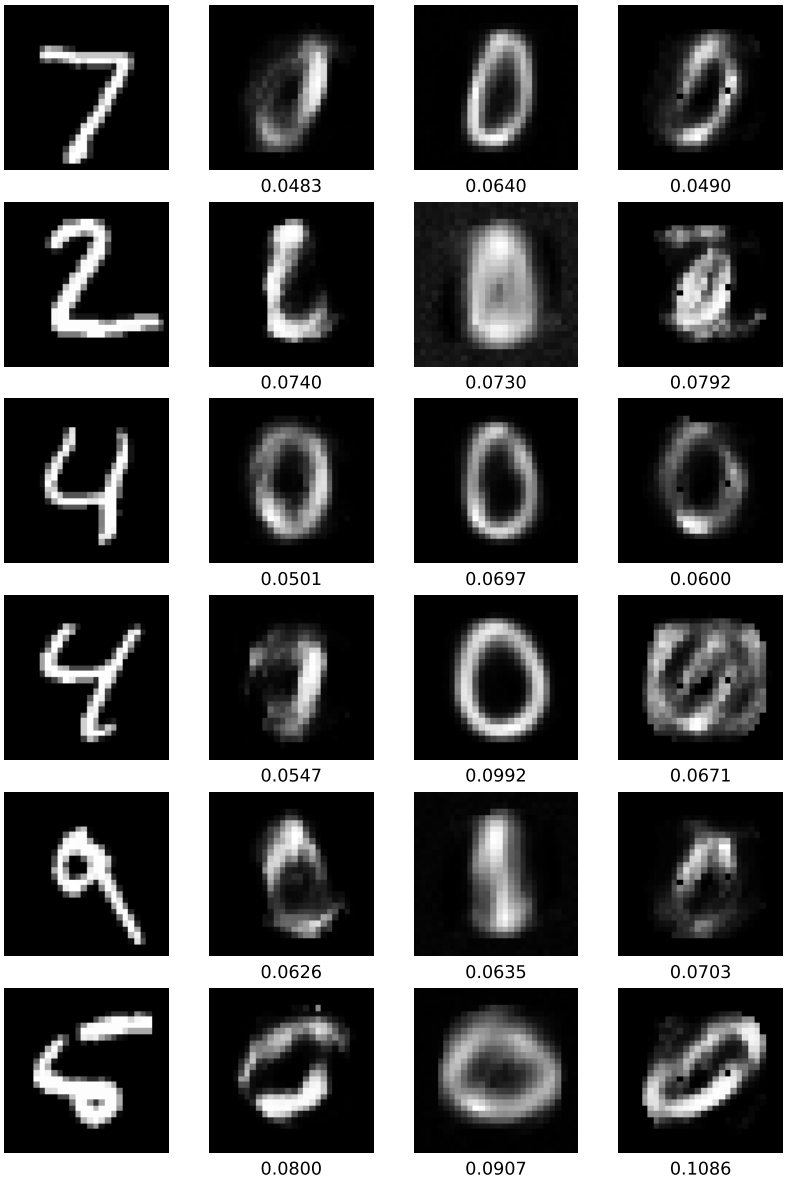


Figure 7.1: The models are trained on the '0's and '1's of the MNIST dataset. The first column is the input, the second column is the prediction from the model trained without dropout, the third column is trained with dropout in the encoder, the fourth column is trained with dropout in the decoder. The number beneath is the reconstruction loss.

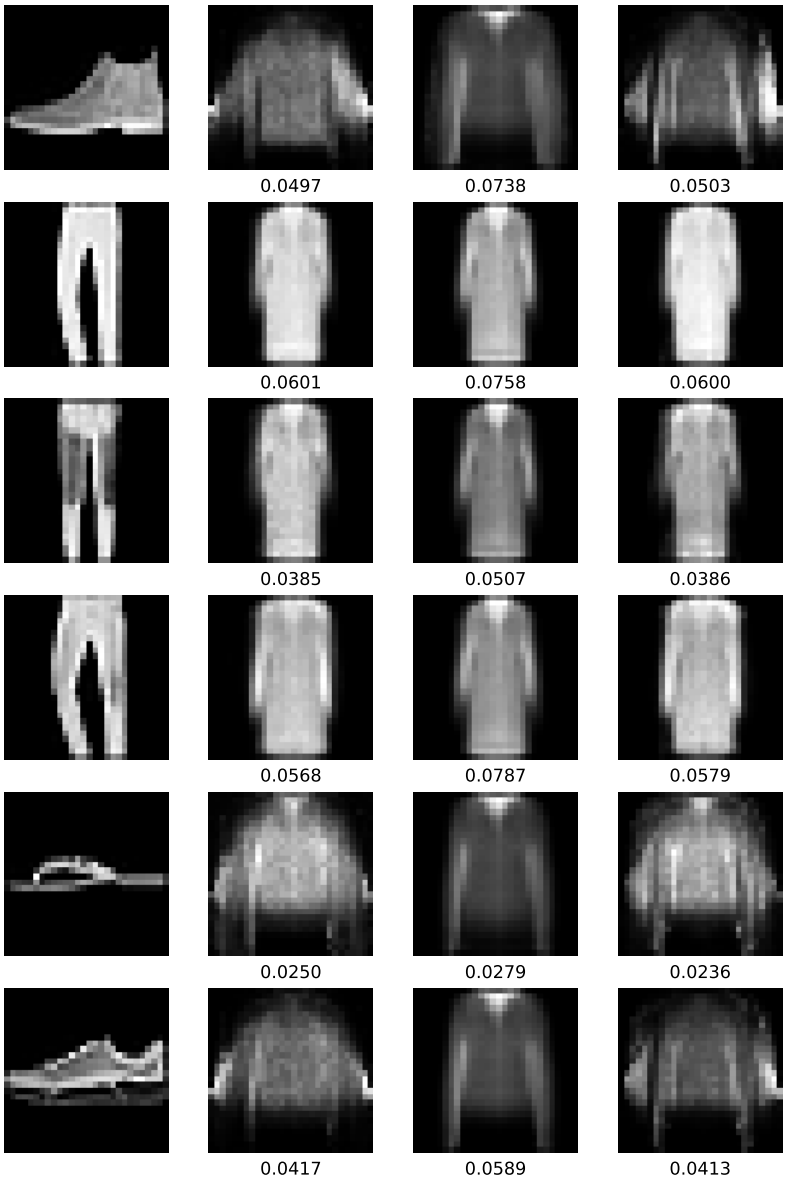


Figure 7.2: The models are trained with 'pullover's and 'coat's of the FashionMNIST dataset. The first column is the input, the second column is the prediction from the model trained without dropout, the third column is trained with dropout in the encoder, the fourth column is trained with dropout in the decoder. The number beneath is the reconstruction loss.

8

Conclusion and Discussion

8.1. Conclusion

In this thesis project, we study how dropout affects the generalization gap, the difference between training and true accuracy, and the test accuracy of deep neural networks. We conclude the regularization behaviors of dropout as follows:

1. Dropout only works with reasonably training set sizes. When the dataset size is too large or too small, dropout will not significantly improve the generalization gap nor the test accuracy. Sometimes, due to the loss of information, dropout can even make it worse than the original models.

2. When the input features include noise, dropout does not necessarily prevent the dependence on such noise features. When the dataset is small, dropout could even make the model rely more on the noise.

3. The network size affects the range of dataset sizes where dropout could improve the performance. If the network is big, it needs more training samples to make dropout work.

4. The average effect of dropout can be interpreted as a data-dependent regularization term in the loss function. This term is usually larger when the input features are larger.

5. On the average dropout loss landscape, sharp minima are smoother than those on the original loss landscape. A potential consequence is that dropout would reduce the model's complexity by making it converge to a flatter minimum.

We are particularly interested in the behavior that dropout does not work well on the small datasets. It is strange because a regularizer should work when the model is capable of overfitting all training samples. We confirm that overfitting is very high for small datasets, where models can mostly depend on the noisy features. To understand dropout's preference on the dataset size, we study how the dataset size affects the model's complexity and co-adaptation. Three criteria are used to measure the model's complexity: norms, sharpness, and sensitivity. Co-adaptation is measured as the performance loss when neurons are dropped in the test stage. We observe some behaviors of the complexity and co-adaptation vs. dataset size:

- Complexity and co-adaptation have the same pattern over dataset size: when the dataset is small, complexity/co-adaptation grows with the dataset size; after hitting a peak, complexity/co-adaptation starts to decrease.

- A model with a smaller network size could find the peak on a smaller dataset size, where complexity/co-adaptation could still grow with dataset size for the large networks.

Based on the similar pattern found with complexity, co-adaptation, and dropout's performance, we propose the following explanation to the phenomenon that dropout does not work well on small datasets. When the dataset is small, it is easier to overfit all the training samples, which means the model has no reason to converge to a very sharp minimum. At a simple minimum, many hidden neurons are working in the same way, thus dropping neurons would not make a difference to the model. When the dataset size grows, there are more samples to fit, so the network needs to make full use of all the hidden neurons to make a complex boundary. As a result, the model is very sensitive to neuron loss, which is why dropout could significantly reduce complexity/co-adaptation on those models. When the dataset size is big enough, the model cannot overfit all the training samples; it is forced to abandon the slight noise and to converge to a simple classifier close to the Bayes classifier. The simple model does not need so many neurons to work, which again makes dropping neurons insignificant. The difference to the small dataset case is that now the training samples are more representative of the true distribution so that the simple classifier would be more accurate.

8.2. Limitations

In this project, we did hundreds of experiments to find how different hyper-parameters affect the performance of dropout. To make sure our results are significant, we repeat each experimental setting 100 times and report the average results in the figures. The cost is that we cannot study all the parameters which might be relevant. For instance, the datasets used in the experiments are not diverse enough: only the Gaussian distributions for binary classification tasks are used to study the complexity trends. Although we believe these trends are universal, more evidence is required to make such a statement. We always use ReLU as the activation function, but a different choice may affect the complexity. We only consider the fully connected deep neural networks, while dropout is also applied in different deep learning models like convolutional neural networks. Overall, some efforts are needed to extend our conclusions of dropout to other scenarios in deep learning.

8.3. Future work

There are several topics that could be interesting for further research:

- Our work has shown the relationship between dropout's performance and complexity/co-adaptation, along with some intuitive explanations. But we haven't strictly proved whether high complexity/co-adaptation is the necessary condition of exploiting dropout. Proving this relationship could make our conclusion more convincing.
- We mostly focus on how dataset size could affect dropout, but other properties of the dataset may contribute. For instance, when the Bayes classifier is very complex, dropout is more likely to make things worse. Studying the performance in more scenarios could help us to see the full picture of dropout.

One particularly interesting finding in our project is that complexity is small when the dataset size is small. It suggests that other regularizers may not work very well under such circumstances. Figure 8.1 shows the performance comparison of models trained with dropout and those trained with weight decay. It appears that weight decay has a very similar behavior as dropout: it only works when the training set is within a reasonable range. Therefore, we believe some of the conclusions in this paper could hold for other regularizers. A possible future work is to study the regularization behaviors of other regularizers on the training dataset size.

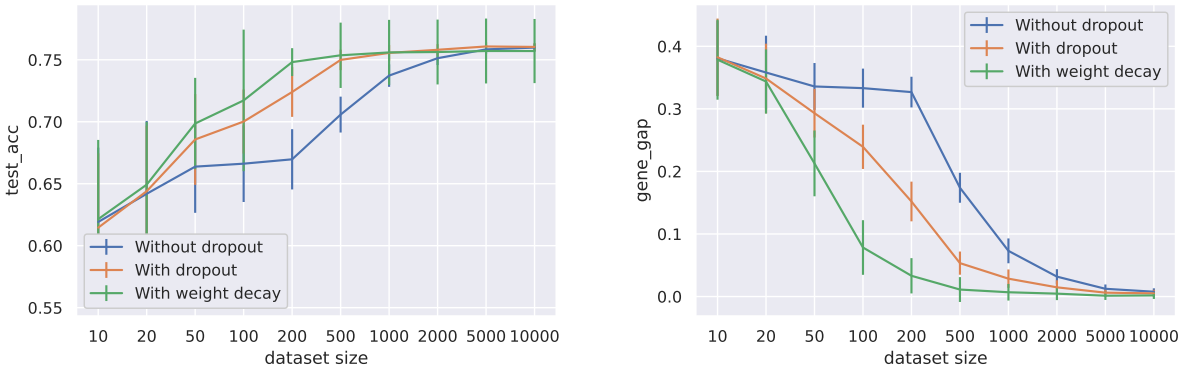


Figure 8.1: Left: test accuracy vs. training set size. Right: generalization gap vs. training set size. The generalization gap is defined as the difference between the training and the testing accuracy. Models are trained on the 10-d Gaussian dataset, where $[d_0, d_1, d_2, d_3] = [10, 100, 100, 2]$. For models trained with dropout, the dropout rate $[p^{(0)}, p^{(1)}, p^{(2)}, p^{(3)}] = [0.5, 0.5, 0.5, 0]$. For the models trained with weight decay, the weight is set as 0.1. Error bars represent the standard deviations of 100 models. The batch size is fixed as 10.

Bibliography

[1] Lei Ba. *Adaptive dropout for training deep neural networks*. PhD thesis, 2013.

[2] Pierre Baldi and Peter J Sadowski. Understanding dropout. *Advances in neural information processing systems*, 26:2814–2822, 2013.

[3] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[4] Yannis Dimopoulos, Paul Bourret, and Sovan Lek. Use of some sensitivity criteria for choosing networks with good generalization ability. *Neural Processing Letters*, 2(6):1–4, 1995. URL `https://link.springer.com/content/pdf/10.1007/BF02309007.pdf`.

[5] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[7] David P Helmbold and Philip M Long. On the inductive bias of dropout. *The Journal of Machine Learning Research*, 16(1):3403–3454, 2015.

[8] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Simplifying neural nets by discovering flat minima. In *Advances in neural information processing systems*, pages 529–536, 1995. URL `http://papers.nips.cc/paper/899-simplifying-neural-nets-by-discovering-flat-minima.pdf`.

[10] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017. URL `https://arxiv.org/pdf/1710.05468.pdf`.

[11] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[12] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *arXiv preprint arXiv:1506.02557*, 2015.

[13] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[14] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. *arXiv preprint arXiv:1904.13310*, 2019.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[16] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Conference on Learning Theory*, pages 1376–1401, 2015. URL `http://proceedings.mlr.press/v40/Neyshabur15.pdf`.

[17] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017. URL `http://papers.nips.cc/paper/7176-exploring-generalization-in-deep-learning.pdf`.

[18] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. Towards understanding the role of over-parametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018. URL https://arxiv.org/pdf/1805.12076.pdf.

[19] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*, 2018.

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. URL https://www.cse.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf.

[21] Jure Sokolić, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017. URL https://ieeexplore.ieee.org/iel7/78/4359509/07934087.pdf.

[22] Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182(566):7, 2013.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15 (1):1929–1958, 2014.

[24] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Normalized flat minima: Exploring scale invariant definition of flat minima for neural networks using pac-bayesian analysis. In *International Conference on Machine Learning*, pages 9636–9647. PMLR, 2020.

[25] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[26] Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359, 2013.

[27] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[28] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016. URL https://arxiv.org/pdf/1611.03530.pdf.

Appendix

Experiment Results

Dataset	Dataset size	Dropout	Width	Test acc	Gene gap
10-d Gaussian	1000	[0.0, 0.0, 0.0, 0.0]	100	0.924 ± 0.006	0.076 ± 0.006
10-d Gaussian	1000	[0.0, 0.0, 0.0, 0.0]	1000	0.925 ± 0.005	0.075 ± 0.005
10-d Gaussian	500	[0.0, 0.5, 0.5, 0.0]	10	0.934 ± 0.005	0.031 ± 0.009
10-d Gaussian	200	[0.0, 0.5, 0.5, 0.0]	10	0.925 ± 0.008	0.065 ± 0.010
10-d Gaussian	100	[0.0, 0.5, 0.5, 0.0]	10	0.920 ± 0.011	0.076 ± 0.013
10-d Gaussian	50	[0.0, 0.5, 0.5, 0.0]	10	0.917 ± 0.017	0.082 ± 0.017
10-d Gaussian	20	[0.0, 0.5, 0.5, 0.0]	10	0.906 ± 0.023	0.094 ± 0.023
10-d Gaussian	10	[0.0, 0.5, 0.5, 0.0]	10	0.878 ± 0.046	0.122 ± 0.046
10-d Gaussian	500	[0.5, 0.5, 0.5, 0.0]	10	0.940 ± 0.005	0.003 ± 0.009
10-d Gaussian	200	[0.5, 0.5, 0.5, 0.0]	10	0.937 ± 0.006	0.015 ± 0.017
10-d Gaussian	100	[0.5, 0.5, 0.5, 0.0]	10	0.931 ± 0.008	0.031 ± 0.018
10-d Gaussian	2000	[0.0, 0.5, 0.5, 0.0]	1000	0.929 ± 0.004	0.053 ± 0.005
10-d Gaussian	50	[0.5, 0.5, 0.5, 0.0]	10	0.920 ± 0.012	0.055 ± 0.021
10-d Gaussian	2000	[0.0, 0.0, 0.0, 0.0]	1000	0.928 ± 0.004	0.072 ± 0.004
10-d Gaussian	20	[0.5, 0.5, 0.5, 0.0]	10	0.888 ± 0.026	0.107 ± 0.026
10-d Gaussian	10	[0.5, 0.5, 0.5, 0.0]	10	0.850 ± 0.049	0.148 ± 0.051
10-d Gaussian	500	[0.0, 0.0, 0.0, 0.0]	10	0.915 ± 0.009	0.081 ± 0.010
10-d Gaussian	200	[0.0, 0.0, 0.0, 0.0]	10	0.914 ± 0.012	0.086 ± 0.012
10-d Gaussian	100	[0.0, 0.0, 0.0, 0.0]	10	0.911 ± 0.015	0.089 ± 0.015
10-d Gaussian	50	[0.0, 0.0, 0.0, 0.0]	10	0.907 ± 0.020	0.093 ± 0.020
10-d Gaussian	1000	[0.0, 0.5, 0.5, 0.0]	1000	0.922 ± 0.006	0.078 ± 0.005
10-d Gaussian	20	[0.0, 0.0, 0.0, 0.0]	10	0.890 ± 0.029	0.110 ± 0.029
10-d Gaussian	500	[0.0, 0.0, 0.0, 0.0]	1000	0.922 ± 0.007	0.078 ± 0.007
10-d Gaussian	200	[0.0, 0.0, 0.0, 0.0]	1000	0.920 ± 0.008	0.080 ± 0.008
10-d Gaussian	1000	[0.0, 0.5, 0.5, 0.0]	10	0.940 ± 0.003	0.015 ± 0.007
10-d Gaussian	2000	[0.5, 0.5, 0.5, 0.0]	10	0.942 ± 0.003	-0.000 ± 0.006
10-d Gaussian	2000	[0.0, 0.5, 0.5, 0.0]	100	0.941 ± 0.002	0.016 ± 0.005
10-d Gaussian	1000	[0.5, 0.5, 0.5, 0.0]	10	0.942 ± 0.004	0.001 ± 0.007
10-d Gaussian	1000	[0.0, 0.5, 0.5, 0.0]	100	0.936 ± 0.004	0.034 ± 0.007
10-d Gaussian	2000	[0.0, 0.0, 0.0, 0.0]	10	0.935 ± 0.004	0.021 ± 0.006
10-d Gaussian	2000	[0.5, 0.5, 0.5, 0.0]	100	0.943 ± 0.002	0.002 ± 0.006
10-d Gaussian	1000	[0.5, 0.5, 0.5, 0.0]	100	0.942 ± 0.003	0.006 ± 0.006
10-d Gaussian	1000	[0.0, 0.0, 0.0, 0.0]	10	0.928 ± 0.006	0.044 ± 0.009
10-d Gaussian	2000	[0.0, 0.0, 0.0, 0.0]	100	0.929 ± 0.004	0.062 ± 0.005
10-d Gaussian	10	[0.0, 0.5, 0.5, 0.0]	1000	0.872 ± 0.056	0.128 ± 0.056
10-d Gaussian	10	[0.0, 0.0, 0.0, 0.0]	1000	0.866 ± 0.064	0.134 ± 0.064
10-d Gaussian	20	[0.0, 0.5, 0.5, 0.0]	1000	0.897 ± 0.028	0.103 ± 0.028
10-d Gaussian	20	[0.0, 0.0, 0.0, 0.0]	1000	0.896 ± 0.029	0.104 ± 0.029
10-d Gaussian	50	[0.0, 0.5, 0.5, 0.0]	1000	0.911 ± 0.017	0.089 ± 0.017
10-d Gaussian	50	[0.0, 0.0, 0.0, 0.0]	1000	0.912 ± 0.017	0.088 ± 0.017
10-d Gaussian	100	[0.0, 0.5, 0.5, 0.0]	1000	0.916 ± 0.012	0.084 ± 0.012
10-d Gaussian	100	[0.0, 0.0, 0.0, 0.0]	1000	0.916 ± 0.012	0.084 ± 0.012
10-d Gaussian	200	[0.0, 0.5, 0.5, 0.0]	1000	0.917 ± 0.009	0.083 ± 0.009
10-d Gaussian	500	[0.0, 0.5, 0.5, 0.0]	1000	0.919 ± 0.007	0.081 ± 0.007

Continued on next page

Dataset	Dataset size	Dropout	Width	Test acc	Gene gap
10-d Gaussian	2000	[0.0, 0.5, 0.5, 0.0]	10	0.942 ± 0.003	0.007 ± 0.005
10-d Gaussian	10	[0.0, 0.0, 0.0, 0.0]	10	0.855 ± 0.063	0.145 ± 0.063
10-d Gaussian	20	[0.0, 0.5, 0.5, 0.0]	100	0.901 ± 0.024	0.099 ± 0.024
10-d Gaussian	10000	[0.5, 0.5, 0.5, 0.0]	100	0.944 ± 0.002	-0.001 ± 0.003
10-d Gaussian	5000	[0.5, 0.5, 0.5, 0.0]	100	0.943 ± 0.002	0.000 ± 0.004
10-d Gaussian	10000	[0.0, 0.0, 0.0, 0.0]	100	0.938 ± 0.003	0.017 ± 0.004
10-d Gaussian	5000	[0.0, 0.0, 0.0, 0.0]	100	0.936 ± 0.004	0.028 ± 0.005
10-d Gaussian	10	[0.0, 0.0, 0.0, 0.0]	100	0.863 ± 0.060	0.137 ± 0.060
10-d Gaussian	500	[0.0, 0.5, 0.5, 0.0]	100	0.931 ± 0.005	0.061 ± 0.007
10-d Gaussian	200	[0.0, 0.5, 0.5, 0.0]	100	0.924 ± 0.008	0.076 ± 0.008
10-d Gaussian	100	[0.0, 0.5, 0.5, 0.0]	100	0.920 ± 0.011	0.080 ± 0.011
10-d Gaussian	50	[0.0, 0.5, 0.5, 0.0]	100	0.917 ± 0.015	0.083 ± 0.015
10-d Gaussian	5000	[0.0, 0.5, 0.5, 0.0]	100	0.942 ± 0.002	0.007 ± 0.004
10-d Gaussian	500	[0.5, 0.5, 0.5, 0.0]	100	0.940 ± 0.003	0.013 ± 0.009
10-d Gaussian	5000	[0.0, 0.0, 0.0, 0.0]	10	0.940 ± 0.003	0.009 ± 0.004
10-d Gaussian	100	[0.5, 0.5, 0.5, 0.0]	100	0.926 ± 0.008	0.062 ± 0.013
10-d Gaussian	10	[0.0, 0.5, 0.5, 0.0]	100	0.878 ± 0.047	0.122 ± 0.047
10-d Gaussian	50	[0.5, 0.5, 0.5, 0.0]	100	0.912 ± 0.014	0.086 ± 0.015
10-d Gaussian	500	[0.0, 0.0, 0.0, 0.0]	100	0.920 ± 0.008	0.080 ± 0.008
10-d Gaussian	20	[0.5, 0.5, 0.5, 0.0]	100	0.882 ± 0.025	0.118 ± 0.025
10-d Gaussian	10	[0.5, 0.5, 0.5, 0.0]	100	0.852 ± 0.057	0.148 ± 0.057
10-d Gaussian	200	[0.0, 0.0, 0.0, 0.0]	100	0.919 ± 0.009	0.081 ± 0.009
10-d Gaussian	100	[0.0, 0.0, 0.0, 0.0]	100	0.914 ± 0.014	0.086 ± 0.014
10-d Gaussian	200	[0.5, 0.5, 0.5, 0.0]	100	0.934 ± 0.006	0.036 ± 0.012
10-d Gaussian	50	[0.0, 0.0, 0.0, 0.0]	100	0.911 ± 0.018	0.089 ± 0.018
10-d Gaussian	10000	[0.0, 0.5, 0.5, 0.0]	10	0.943 ± 0.002	0.001 ± 0.003
10-d Gaussian	5000	[0.0, 0.5, 0.5, 0.0]	10	0.942 ± 0.002	0.002 ± 0.004
10-d Gaussian	10000	[0.0, 0.5, 0.5, 0.0]	100	0.943 ± 0.002	0.004 ± 0.003
10-d Gaussian	10000	[0.0, 0.0, 0.0, 0.0]	10	0.941 ± 0.002	0.005 ± 0.003
10-d Gaussian	5000	[0.5, 0.5, 0.5, 0.0]	10	0.942 ± 0.003	-0.001 ± 0.004
10-d Gaussian	20	[0.0, 0.0, 0.0, 0.0]	100	0.891 ± 0.029	0.109 ± 0.029
10-d Gaussian	10000	[0.5, 0.5, 0.5, 0.0]	10	0.942 ± 0.003	-0.001 ± 0.002
10-d noise	1000	[0.5, 0.5, 0.5, 0.0]	10	0.751 ± 0.011	0.010 ± 0.014
10-d noise	5000	[0.0, 0.5, 0.5, 0.0]	1000	0.752 ± 0.005	0.031 ± 0.009
10-d noise	10000	[0.0, 0.0, 0.0, 0.0]	1000	0.741 ± 0.007	0.070 ± 0.008
10-d noise	5000	[0.0, 0.0, 0.0, 0.0]	1000	0.728 ± 0.010	0.129 ± 0.012
10-d noise	10000	[0.0, 0.5, 0.5, 0.0]	100	0.761 ± 0.003	0.008 ± 0.006
10-d noise	5000	[0.0, 0.5, 0.5, 0.0]	100	0.760 ± 0.003	0.012 ± 0.006
10-d noise	10000	[0.5, 0.5, 0.5, 0.0]	100	0.753 ± 0.010	0.002 ± 0.006
10-d noise	5000	[0.5, 0.5, 0.5, 0.0]	100	0.751 ± 0.010	0.004 ± 0.008
10-d noise	10000	[0.0, 0.0, 0.0, 0.0]	100	0.752 ± 0.005	0.029 ± 0.007
10-d noise	5000	[0.0, 0.0, 0.0, 0.0]	100	0.747 ± 0.007	0.053 ± 0.009
10-d noise	1000	[0.0, 0.5, 0.5, 0.0]	10	0.755 ± 0.006	0.026 ± 0.013
10-d noise	2000	[0.5, 0.5, 0.5, 0.0]	100	0.748 ± 0.012	0.010 ± 0.010
10-d noise	2000	[0.0, 0.5, 0.5, 0.0]	10	0.757 ± 0.006	0.013 ± 0.011
10-d noise	2000	[0.0, 0.5, 0.5, 0.0]	100	0.756 ± 0.005	0.030 ± 0.010
10-d noise	1000	[0.0, 0.0, 0.0, 0.0]	10	0.739 ± 0.009	0.064 ± 0.018
10-d noise	2000	[0.0, 0.0, 0.0, 0.0]	10	0.752 ± 0.005	0.028 ± 0.011
10-d noise	5000	[0.0, 0.5, 0.5, 0.0]	10	0.760 ± 0.004	0.005 ± 0.006
10-d noise	10000	[0.0, 0.5, 0.5, 0.0]	10	0.761 ± 0.003	0.003 ± 0.005
10-d noise	2000	[0.5, 0.5, 0.5, 0.0]	10	0.751 ± 0.011	0.006 ± 0.011
10-d noise	1000	[0.0, 0.5, 0.5, 0.0]	100	0.748 ± 0.007	0.066 ± 0.015
10-d noise	2000	[0.0, 0.5, 0.5, 0.0]	1000	0.738 ± 0.008	0.090 ± 0.013
10-d noise	5000	[0.5, 0.5, 0.5, 0.0]	10	0.753 ± 0.010	0.001 ± 0.007

Continued on next page

Dataset	Dataset size	Dropout	Width	Test acc	Gene gap
10-d noise	1000	[0.0, 0.5, 0.5, 0.0]	1000	0.713 ± 0.009	0.208 ± 0.016
10-d noise	500	[0.0, 0.5, 0.5, 0.0]	1000	0.694 ± 0.014	0.303 ± 0.014
10-d noise	200	[0.0, 0.5, 0.5, 0.0]	1000	0.684 ± 0.019	0.316 ± 0.019
10-d noise	100	[0.0, 0.5, 0.5, 0.0]	1000	0.675 ± 0.025	0.325 ± 0.025
10-d noise	50	[0.0, 0.5, 0.5, 0.0]	1000	0.671 ± 0.030	0.329 ± 0.030
10-d noise	20	[0.0, 0.5, 0.5, 0.0]	1000	0.645 ± 0.051	0.355 ± 0.051
10-d noise	10	[0.0, 0.5, 0.5, 0.0]	1000	0.620 ± 0.064	0.380 ± 0.064
10-d noise	2000	[0.5, 0.5, 0.5, 0.0]	1000	0.740 ± 0.014	0.016 ± 0.012
10-d noise	1000	[0.5, 0.5, 0.5, 0.0]	1000	0.734 ± 0.018	0.032 ± 0.014
10-d noise	500	[0.5, 0.5, 0.5, 0.0]	1000	0.728 ± 0.021	0.063 ± 0.018
10-d noise	200	[0.5, 0.5, 0.5, 0.0]	1000	0.705 ± 0.026	0.163 ± 0.028
10-d noise	100	[0.5, 0.5, 0.5, 0.0]	1000	0.675 ± 0.034	0.266 ± 0.032
10-d noise	50	[0.5, 0.5, 0.5, 0.0]	1000	0.648 ± 0.039	0.342 ± 0.038
10-d noise	20	[0.5, 0.5, 0.5, 0.0]	1000	0.620 ± 0.056	0.377 ± 0.060
10-d noise	10	[0.5, 0.5, 0.5, 0.0]	1000	0.594 ± 0.062	0.405 ± 0.062
10-d noise	2000	[0.0, 0.0, 0.0, 0.0]	1000	0.707 ± 0.012	0.280 ± 0.014
10-d noise	1000	[0.0, 0.0, 0.0, 0.0]	1000	0.706 ± 0.010	0.294 ± 0.010
10-d noise	500	[0.0, 0.0, 0.0, 0.0]	1000	0.703 ± 0.013	0.297 ± 0.013
10-d noise	200	[0.0, 0.0, 0.0, 0.0]	1000	0.694 ± 0.020	0.306 ± 0.020
10-d noise	100	[0.0, 0.0, 0.0, 0.0]	1000	0.684 ± 0.026	0.316 ± 0.026
10-d noise	5000	[0.0, 0.0, 0.0, 0.0]	10	0.758 ± 0.004	0.011 ± 0.008
10-d noise	50	[0.0, 0.0, 0.0, 0.0]	1000	0.677 ± 0.031	0.323 ± 0.031
10-d noise	10000	[0.0, 0.0, 0.0, 0.0]	10	0.760 ± 0.004	0.006 ± 0.005
10-d noise	20	[0.0, 0.0, 0.0, 0.0]	1000	0.646 ± 0.053	0.354 ± 0.053
10-d noise	10	[0.0, 0.0, 0.0, 0.0]	1000	0.618 ± 0.064	0.382 ± 0.064
10-d noise	10000	[0.5, 0.5, 0.5, 0.0]	10	0.755 ± 0.008	-0.000 ± 0.006
10-d noise	1000	[0.5, 0.5, 0.5, 0.0]	100	0.744 ± 0.013	0.019 ± 0.014
10-d noise	10	[0.0, 0.0, 0.0, 0.0]	100	0.617 ± 0.065	0.383 ± 0.065
10-d noise	1000	[0.0, 0.0, 0.0, 0.0]	100	0.698 ± 0.012	0.285 ± 0.015
10-d noise	100	[0.0, 0.5, 0.5, 0.0]	10	0.700 ± 0.026	0.240 ± 0.035
10-d noise	50	[0.0, 0.5, 0.5, 0.0]	10	0.685 ± 0.037	0.296 ± 0.041
10-d noise	20	[0.0, 0.5, 0.5, 0.0]	10	0.641 ± 0.053	0.357 ± 0.054
10-d noise	500	[0.0, 0.0, 0.0, 0.0]	10	0.711 ± 0.014	0.158 ± 0.023
10-d noise	200	[0.0, 0.0, 0.0, 0.0]	10	0.672 ± 0.024	0.324 ± 0.025
10-d noise	100	[0.0, 0.0, 0.0, 0.0]	10	0.666 ± 0.031	0.333 ± 0.031
10-d noise	50	[0.0, 0.0, 0.0, 0.0]	10	0.664 ± 0.038	0.336 ± 0.038
10-d noise	20	[0.0, 0.0, 0.0, 0.0]	10	0.641 ± 0.059	0.359 ± 0.059
10-d noise	10	[0.0, 0.0, 0.0, 0.0]	10	0.619 ± 0.060	0.381 ± 0.060
10-d noise	200	[0.0, 0.5, 0.5, 0.0]	10	0.724 ± 0.018	0.147 ± 0.025
10-d noise	50	[0.5, 0.5, 0.5, 0.0]	100	0.668 ± 0.041	0.271 ± 0.042
10-d noise	200	[0.5, 0.5, 0.5, 0.0]	100	0.721 ± 0.028	0.089 ± 0.025
10-d noise	500	[0.5, 0.5, 0.5, 0.0]	100	0.740 ± 0.015	0.036 ± 0.018
10-d noise	10	[0.0, 0.5, 0.5, 0.0]	100	0.622 ± 0.064	0.378 ± 0.064
10-d noise	20	[0.0, 0.5, 0.5, 0.0]	100	0.652 ± 0.051	0.348 ± 0.051
10-d noise	50	[0.0, 0.5, 0.5, 0.0]	100	0.684 ± 0.031	0.316 ± 0.031
10-d noise	100	[0.0, 0.5, 0.5, 0.0]	100	0.694 ± 0.024	0.306 ± 0.024
10-d noise	200	[0.0, 0.5, 0.5, 0.0]	100	0.707 ± 0.017	0.290 ± 0.017
10-d noise	500	[0.0, 0.5, 0.5, 0.0]	100	0.733 ± 0.012	0.158 ± 0.021
10-d noise	2000	[0.0, 0.0, 0.0, 0.0]	100	0.721 ± 0.013	0.154 ± 0.018
10-d noise	100	[0.5, 0.5, 0.5, 0.0]	100	0.696 ± 0.035	0.170 ± 0.032
10-d noise	500	[0.0, 0.5, 0.5, 0.0]	10	0.749 ± 0.009	0.053 ± 0.018
10-d noise	10	[0.0, 0.5, 0.5, 0.0]	10	0.612 ± 0.062	0.386 ± 0.064
10-d noise	20	[0.5, 0.5, 0.5, 0.0]	10	0.635 ± 0.068	0.270 ± 0.063
10-d noise	10000	[0.0, 0.5, 0.5, 0.0]	1000	0.754 ± 0.005	0.017 ± 0.007

Continued on next page

Dataset	Dataset size	Dropout	Width	Test acc	Gene gap
10-d noise	20	[0.0, 0.0, 0.0, 0.0]	100	0.646 ± 0.053	0.354 ± 0.053
10-d noise	50	[0.0, 0.0, 0.0, 0.0]	100	0.676 ± 0.030	0.324 ± 0.030
10-d noise	100	[0.0, 0.0, 0.0, 0.0]	100	0.680 ± 0.026	0.320 ± 0.026
10-d noise	10	[0.5, 0.5, 0.5, 0.0]	10	0.619 ± 0.078	0.341 ± 0.094
10-d noise	200	[0.5, 0.5, 0.5, 0.0]	10	0.725 ± 0.033	0.050 ± 0.025
10-d noise	100	[0.5, 0.5, 0.5, 0.0]	10	0.699 ± 0.041	0.090 ± 0.038
10-d noise	50	[0.5, 0.5, 0.5, 0.0]	10	0.671 ± 0.054	0.168 ± 0.044
10-d noise	500	[0.5, 0.5, 0.5, 0.0]	10	0.745 ± 0.016	0.019 ± 0.018
10-d noise	20	[0.5, 0.5, 0.5, 0.0]	100	0.637 ± 0.060	0.350 ± 0.067
10-d noise	200	[0.0, 0.0, 0.0, 0.0]	100	0.687 ± 0.021	0.313 ± 0.021
10-d noise	500	[0.0, 0.0, 0.0, 0.0]	100	0.697 ± 0.014	0.303 ± 0.014
10-d noise	10	[0.5, 0.5, 0.5, 0.0]	100	0.607 ± 0.071	0.388 ± 0.075
2-d Gaussian	5000	[0.0, 0.5, 0.5, 0.0]	10	0.854 ± 0.002	-0.007 ± 0.005
2-d Gaussian	100	[0.0, 0.0, 0.0, 0.0]	100	0.801 ± 0.024	0.158 ± 0.032
2-d Gaussian	50	[0.0, 0.0, 0.0, 0.0]	100	0.787 ± 0.036	0.208 ± 0.038
2-d Gaussian	20	[0.0, 0.0, 0.0, 0.0]	100	0.771 ± 0.056	0.229 ± 0.056
2-d Gaussian	10	[0.0, 0.0, 0.0, 0.0]	100	0.751 ± 0.094	0.249 ± 0.094
2-d Gaussian	500	[0.0, 0.0, 0.0, 0.0]	10	0.850 ± 0.005	0.005 ± 0.015
2-d Gaussian	200	[0.0, 0.0, 0.0, 0.0]	10	0.842 ± 0.009	0.026 ± 0.022
2-d Gaussian	100	[0.0, 0.0, 0.0, 0.0]	10	0.823 ± 0.021	0.079 ± 0.037
2-d Gaussian	50	[0.0, 0.0, 0.0, 0.0]	10	0.799 ± 0.033	0.158 ± 0.046
2-d Gaussian	10	[0.0, 0.0, 0.0, 0.0]	10	0.754 ± 0.094	0.246 ± 0.094
2-d Gaussian	10000	[0.5, 0.5, 0.5, 0.0]	10	0.852 ± 0.004	-0.009 ± 0.005
2-d Gaussian	5000	[0.5, 0.5, 0.5, 0.0]	100	0.852 ± 0.003	-0.007 ± 0.005
2-d Gaussian	200	[0.0, 0.0, 0.0, 0.0]	100	0.826 ± 0.014	0.067 ± 0.026
2-d Gaussian	5000	[0.0, 0.0, 0.0, 0.0]	10	0.854 ± 0.002	-0.006 ± 0.005
2-d Gaussian	10000	[0.0, 0.0, 0.0, 0.0]	10	0.854 ± 0.002	-0.007 ± 0.004
2-d Gaussian	5000	[0.5, 0.5, 0.5, 0.0]	10	0.852 ± 0.005	-0.008 ± 0.006
2-d Gaussian	10000	[0.0, 0.0, 0.0, 0.0]	100	0.853 ± 0.002	-0.006 ± 0.004
2-d Gaussian	5000	[0.0, 0.0, 0.0, 0.0]	100	0.853 ± 0.002	-0.006 ± 0.006
2-d Gaussian	10000	[0.0, 0.5, 0.5, 0.0]	10	0.853 ± 0.002	-0.007 ± 0.004
2-d Gaussian	20	[0.0, 0.0, 0.0, 0.0]	10	0.775 ± 0.056	0.221 ± 0.059
2-d Gaussian	500	[0.0, 0.0, 0.0, 0.0]	100	0.846 ± 0.006	0.014 ± 0.015
2-d Gaussian	200	[0.5, 0.5, 0.5, 0.0]	100	0.838 ± 0.017	0.012 ± 0.026
2-d Gaussian	20	[0.5, 0.5, 0.5, 0.0]	10	0.774 ± 0.068	0.096 ± 0.075
2-d Gaussian	2000	[0.0, 0.5, 0.5, 0.0]	10	0.853 ± 0.003	-0.006 ± 0.007
2-d Gaussian	1000	[0.0, 0.5, 0.5, 0.0]	10	0.853 ± 0.002	-0.003 ± 0.010
2-d Gaussian	2000	[0.5, 0.5, 0.5, 0.0]	10	0.851 ± 0.005	-0.007 ± 0.007
2-d Gaussian	1000	[0.5, 0.5, 0.5, 0.0]	10	0.851 ± 0.008	-0.005 ± 0.011
2-d Gaussian	2000	[0.0, 0.0, 0.0, 0.0]	10	0.853 ± 0.002	-0.004 ± 0.007
2-d Gaussian	1000	[0.0, 0.0, 0.0, 0.0]	10	0.853 ± 0.002	-0.002 ± 0.010
2-d Gaussian	2000	[0.0, 0.5, 0.5, 0.0]	100	0.853 ± 0.002	-0.005 ± 0.007
2-d Gaussian	500	[0.0, 0.5, 0.5, 0.0]	100	0.849 ± 0.006	0.005 ± 0.015
2-d Gaussian	1000	[0.0, 0.5, 0.5, 0.0]	100	0.852 ± 0.003	-0.003 ± 0.010
2-d Gaussian	2000	[0.5, 0.5, 0.5, 0.0]	100	0.852 ± 0.003	-0.007 ± 0.008
2-d Gaussian	1000	[0.5, 0.5, 0.5, 0.0]	100	0.850 ± 0.006	-0.004 ± 0.011
2-d Gaussian	2000	[0.0, 0.0, 0.0, 0.0]	100	0.853 ± 0.003	-0.004 ± 0.007
2-d Gaussian	10	[0.0, 0.0, 0.0, 0.0]	100	0.770 ± 0.062	0.230 ± 0.062
2-d Gaussian	200	[0.0, 0.0, 0.0, 0.0]	100	0.779 ± 0.017	0.215 ± 0.016
2-d Gaussian	10	[0.0, 0.5, 0.5, 0.0]	100	0.795 ± 0.044	0.195 ± 0.062
2-d Gaussian	200	[0.0, 0.5, 0.5, 0.0]	100	0.803 ± 0.011	0.122 ± 0.020
2-d Gaussian	1000	[0.0, 0.0, 0.0, 0.0]	100	0.852 ± 0.003	0.001 ± 0.009
2-d Gaussian	200	[0.0, 0.5, 0.5, 0.0]	100	0.841 ± 0.011	0.025 ± 0.023
2-d Gaussian	100	[0.0, 0.5, 0.5, 0.0]	100	0.828 ± 0.019	0.067 ± 0.033

Continued on next page

Dataset	Dataset size	Dropout	Width	Test acc	Gene gap
2-d Gaussian	50	[0.0, 0.5, 0.5, 0.0]	100	0.807 ± 0.030	0.136 ± 0.042
2-d Gaussian	50	[0.5, 0.5, 0.5, 0.0]	10	0.810 ± 0.043	0.045 ± 0.053
2-d Gaussian	100	[0.5, 0.5, 0.5, 0.0]	10	0.834 ± 0.024	0.022 ± 0.034
2-d Gaussian	200	[0.5, 0.5, 0.5, 0.0]	10	0.842 ± 0.016	0.007 ± 0.025
2-d Gaussian	500	[0.5, 0.5, 0.5, 0.0]	10	0.848 ± 0.009	-0.002 ± 0.016
2-d Gaussian	10	[0.5, 0.5, 0.5, 0.0]	100	0.732 ± 0.101	0.162 ± 0.115
2-d Gaussian	20	[0.5, 0.5, 0.5, 0.0]	100	0.760 ± 0.068	0.116 ± 0.075
2-d Gaussian	50	[0.5, 0.5, 0.5, 0.0]	100	0.797 ± 0.047	0.059 ± 0.049
2-d Gaussian	100	[0.5, 0.5, 0.5, 0.0]	100	0.828 ± 0.023	0.030 ± 0.038
2-d Gaussian	500	[0.5, 0.5, 0.5, 0.0]	100	0.847 ± 0.009	0.001 ± 0.017
2-d Gaussian	10	[0.0, 0.5, 0.5, 0.0]	10	0.786 ± 0.073	0.200 ± 0.083
2-d Gaussian	20	[0.0, 0.5, 0.5, 0.0]	10	0.809 ± 0.041	0.138 ± 0.054
2-d Gaussian	50	[0.0, 0.5, 0.5, 0.0]	10	0.827 ± 0.026	0.070 ± 0.049
2-d Gaussian	100	[0.0, 0.5, 0.5, 0.0]	10	0.841 ± 0.013	0.033 ± 0.036
2-d Gaussian	200	[0.0, 0.5, 0.5, 0.0]	10	0.845 ± 0.010	0.016 ± 0.024
2-d Gaussian	500	[0.0, 0.5, 0.5, 0.0]	10	0.851 ± 0.005	0.002 ± 0.017
2-d Gaussian	10	[0.0, 0.5, 0.5, 0.0]	100	0.774 ± 0.077	0.225 ± 0.079
2-d Gaussian	20	[0.0, 0.5, 0.5, 0.0]	100	0.795 ± 0.048	0.201 ± 0.050
2-d Gaussian	10	[0.5, 0.5, 0.5, 0.0]	10	0.747 ± 0.090	0.140 ± 0.092
2-d Gaussian	10000	[0.5, 0.5, 0.5, 0.0]	100	0.852 ± 0.004	-0.008 ± 0.005

Continued on next page