Clustering Phenomena in Dropout

David Kewei Lin
linkewei@stanford.edu

Jensen Jinhui Wang
wangjh97@stanford.edu

Phil Chen
philhc@stanford.edu

Abstract

Despite its prevalence, Dropout lacks a conclusive explanation behind its regularization effects. In this paper, we verify that applying Dropout is akin to training a stochastic neural network. We then analyze the variance across models to quantify the regularization properties of Dropout and explain the observed trends from the stochastic perspective. Subsequently, we propose that the network produces similar nodes to adapt to the stochastic training process, resulting in a clustering phenomenon that reduces model complexity. Our experiments culminate in a compression scheme that converts the trained stochastic network into a smaller, deterministic network for prediction.

1 Introduction

Dropout, as introduced in Srivastava et al. [2014], describes a form of regularization where in the training phase, nodes are randomly "dropped" with some probability p (with optimal values occurring between 0.2 and 0.5). The rationale for this was to simulate the simultaneous training of many model combinations while keeping the training process relatively efficient.

Given a fixed feed-forward network with N ordered nodes, we associate each vector $M \in \mathbb{R}^N$ a modified network with the output of each node scaled by the corresponding element in the vector. Then, Dropout randomly samples models M from a model distribution $\mathcal{M}$ that is restricted to the set of binary masks $\{0,1\}^N$, where each component is independently drawn from $\text{Bernoulli}(1-p)$ (p is the Dropout rate). Although M is sampled as a binary mask, it is often useful to consider M with elements of $\mathbb{R}$ (e.g. scaling during prediction time).

Given input x, output $o(x;\theta,M)$ (where model parameters θ will be omitted for simplicity), ground truths y, and loss $\ell(o,y)$, the Dropout training procedure approximates minimizing the loss of $\mathbb{E}_{M\sim\mathcal{M}}[\ell(o(x;M),y)]$ (Srivastava et al. [2014]). We verify this stochastic interpretation through our experiments and suggest an alternate prediction scheme that samples the network with Dropout still enabled. Subsequently, we propose that a neural network may adopt a mechanism of replicating similar nodes in response to Dropout. We verify this hypothesis in a series of experiments, which culminates in a novel compression scheme that clusters the proposed similar nodes to produce a smaller, deterministic network after training.

2 Related work

Gal and Ghahramani [2015, 2016a] reinterpret Dropout as approximate variational inference in a Bayesian neural network, where the approximate variational distribution (based on the Dropout model distribution) is made close to the true latent distribution by minimizing the evidence lower bound. In Gal and Ghahramani [2016b], their theoretical results motivate novel dropout techniques in RNNs such as dropping words at random throughout the input sentence to improve on an RNN model to achieve state-of-the-art language modelling with the Penn Treebank. Kingma et al. [2015] also apply variational dropout techniques that provide a Bayesian interpretation to Gaussian dropout. More specifically, they show that Gaussian dropout is equivalent to sampling the output of each layer from a Gaussian distribution (before applying activations) and obtain efficient estimators for the gradient of the variational lower bound via the reparameterization trick.

Hernández-Lobato and Adams [2015] tackle the challenge of scalability of Bayesian neural networks by using their interpretation of probabilistic neural network models to propose a probabilistic backpropogation algorithm, which updates the parameters of their Gaussian beliefs that minimize the KL divergence. Similarly, Tang and Salakhutdinov [2013] propose a stochastic feedforward neural network (SFNN) with hidden layers with both deterministic and stochastic variables with a generalized EM training procedure on their parameters. They reason that SFNN have attractive properties including being able to have distributed code to represent an exponential number of mixture components in output space and train with the standard backpropogation algorithm as in standard feed-forward neural networks.

These approaches all revolve around interpreting dropout through a Bayesian approach and minimizing the KL divergence between their approximated distribution and the empirical distribution. In our project, we also use sampling to acknowledge the stochasticity of the network, and we adopt the interpretation of feed-forward networks with Dropout as SFNNs.

3 Methods

We use simple feed-forward networks with 3 hidden layers on the MNIST dataset and a final output layer with 10 units (corresponding to the number of prediction classes). The output logits are then passed through a softmax layer before the cross-entropy loss with the labels is computed. Such a rudimentary model was employed to ensure that the experiments are more controlled and the results can be explained with fewer assumptions.

3.1 Test-time sampling

As pointed out in Wan et al. [2013], the fundamental approximation made by Dropout by renormalizing at prediction time is

$$\begin{aligned}
\mathbb{E}_{M\sim\mathcal{M}}[\ell(o(x;M),y)] &= \ell\left(\exp(\mathbb{E}_{M\sim\mathcal{M}}[\log o(x;M)]),y\right) \qquad \text{(for cross-entropy loss } \ell\text{, which is linear in } \log o\text{)} \\
&\approx \ell(\mathbb{E}_{M\sim\mathcal{M}}[o(x;M)],y) \qquad (1) \\
&\approx \ell(o(x;\mathbb{E}_{M\sim\mathcal{M}}[M]),y) \qquad (2)
\end{aligned}$$

(1) is valid assuming that $o(x;M)$ take similar values across M, but (2) is not guaranteed to hold as $o(x;M)$ may contain non-linearities (which are nonlinear in M).

In section 4.1, we attempt to measure the accuracy of the Dropout linearity assumption by comparing the results of the scaled model ($M_{\text{Test}} = \mathbb{E}_{M\sim\mathcal{M}}[M]$, as per standard Dropout) and the actual model combination approximated by drawing n model samples $M \sim \mathcal{M}$ and then computing the output as $o(x, M_{test}) = \exp(\frac{1}{n}\sum_{i=1}^{n}[\log o(x;M_i)])$. It is important to note that only a single model is trained though different prediction methods are used.

3.2 Metrics

The default metric used is test accuracy. However, to investigate the purported regularization effects, we also consider various measures of variance. *Model variance* is defined as the variance of the predictions across models trained on a subset of examples average across the 10 output logits. *Mean sampling standard deviation* (MSTD) is the mean of the standard deviations of the logits produced by the 10 output nodes across different samples. These are specifically investigated in subsections 4.2 and 4.3.2 respectively.

4 Experiments

4.1 Dropout minimizes the expected loss of a SFNN

4.1.1 Sampling versus Scaling

A three-layer network, with 100 units and an identical Dropout rate in each hidden layer, was trained and our sampling method was compared to the conventional scaling procedure in the prediction phase. As expected, the models perform significantly better at test time than at training time, regardless of whether it samples—indicating that a post-processing step after training is indeed necessary. Sampling at test time generally performs better than scaling (especially at higher Dropout rates), supporting the hypothesis that Dropout training actually trains a stochastic neural network that is suited for sampling. That said, the high performance of the model without sampling suggests that relatively scaling the weights at test time by $\frac{1}{1-\text{Dropout rate}}$ produces similar outputs as sampling does.

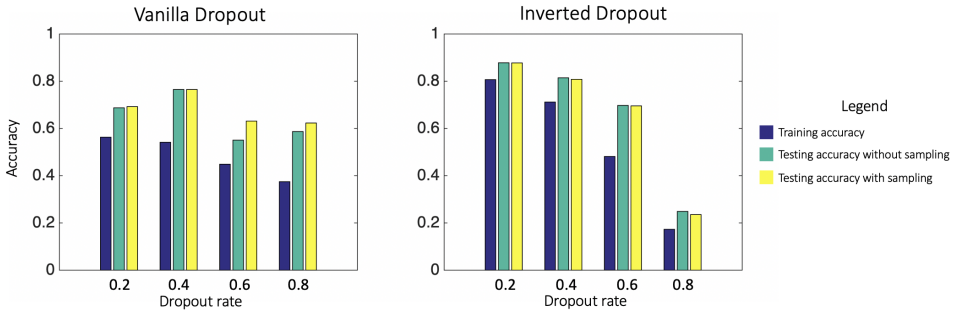

Figure 1: Effect of Dropout rate and type on accuracy during training and testing. We trained and tested models with 100 hidden units on each of the three hidden layers and the same Dropout rate on each of these hidden units.

4.1.2 Varying Batch Size

In order to minimize the expected loss over the family $\mathcal{M}$, the gradient used in a stochastic training update, given example $x^{(i)}$, should be

$$\nabla_\theta \ell^*(\theta) = \mathbb{E}_{M\sim\mathcal{M}}[\nabla_\theta \ell(o(x^{(i)}; M), y)]$$

However, Dropout only samples a single model from $\mathcal{M}$ during training time for each example $x^{(i)}$, which is surely not representative of the expectation. The actual batch gradient computed is

$$\nabla_\theta \hat{\ell}(\theta) = \sum_{i=1}^{n} \nabla_\theta \ell(o(x^{(i)}; M_i), y)$$

where $M_i \sim \mathcal{M}$. If we assume $\forall M \in \mathcal{M}$, $\nabla_\theta \ell(o(x^{(i)}, M))$ points in the same direction for all training examples $x^{(i)}$, $\nabla_\theta \hat{\ell}(\theta)$ is akin to a point estimate of $\nabla_\theta \ell^*(\theta)$. Thus, if Dropout is indeed training a stochastic network, we would expect the performance of the trained model to scale with batch size as the point estimates will be more accurate.

It can be seen from the results summarized in Table 1 that the test accuracy of identical three-layer networks, with and without Dropout, generally increases with batch size. However, the effect of increasing batch size is more significant in the case of the network with Dropout. Furthermore, the accuracy fluctuations for the network – in particular, with batch size 128 – with Dropout shows that it is more sensitive to batch size than its counterpart without Dropout, suggesting the presence of an underlying stochastic network.

Table 1: Experimental Results of Varying Batch Size in a Three-Layer Network

(a) With 0.4 Vanilla Dropout

Batch	Steps	Test Acc
16	20000	0.765
32	10000	0.787
64	5000	0.797
128	2500	0.697
256	1250	0.815
512	625	0.836

(b) Without Dropout

Batch	Steps	Test Acc
16	20000	0.891
32	10000	0.897
64	5000	0.892
128	2500	0.907
256	1250	0.906
512	625	0.918

(c) Hyperparameters

Hidden units	[100, 100, 100]
Dropout rates	[0.4, 0.4, 0.4]
Number of samples	200

4.2 Effect of Dropout on model variance

From the perspective of learning theory, Dropout as an effective form of regularization must reduce the variance of the model. To calculate model variance, we train 10 models with parameters $\theta_1, \theta_2, ..., \theta_{10}$ respectively (keeping architectures and hyperparameters constant), each on 10,000 randomly selected training examples of the MNIST dataset (out of 60,000 total). Then we compute the mean variance of the model predictions on the test set, calculated as

$$\text{Var}_\theta = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{10}\sum_{j=1}^{10}\left(\mathbb{E}_{M\sim\mathcal{M}}[o(x^{(i)}; M, \theta_j) - \overline{o(x^{(i)}; M)}]\right)^2$$

where $x^{(i)}, y^{(i)}$ are the test examples and $\overline{o(x^{(i)}; M)} = \frac{1}{10}\sum_{k=1}^{10} o(x^{(i)}; M, \theta_k)]$.

Our experiments verify that the model variance with Dropout is significantly lower than without Dropout, and vanilla Dropout results in the lowest variance (Figure 2). Interestingly, sampling the model with vanilla dropout at test time results in much lower model variances than inverting. To explain this result heuristically, we use the interpretation in Tang and Salakhutdinov [2013] of dropout models as a distribution from an exponential number of mixture components. We know dropout trains θ to minimize expected loss over the family $\mathcal{M}$. Inverting at test time is similar to selecting a single component $M_0(\theta) \in \mathcal{M}(\theta)$. Then if all $M_i(\theta) \in \mathcal{M}(\theta)$ have approximately equal variances, any sample $\frac{1}{n}\sum_{j=1}^{n} M_{i_j}(\theta)$ will satisfy $\text{Var}\left(\frac{1}{n}\sum_{j=1}^{n} M_{i_j}(\theta)\right) < \text{Var}\, M_0(\theta)$ with high probability.

4.3 Dropout clusters neurons

We propose that a neural network may produce similar nodes in response to the stochastic training imposed by Dropout (established in Section 4.1). This leads to a self-correcting mechanism where a node is able to compensate for another similar dropped node. Concretely, the network may learn to take the average output of groups of similar nodes—a plausible scheme considering how the variance of the output scales inversely with the number of similar nodes such that the network becomes more robust to dropping with more replicates.

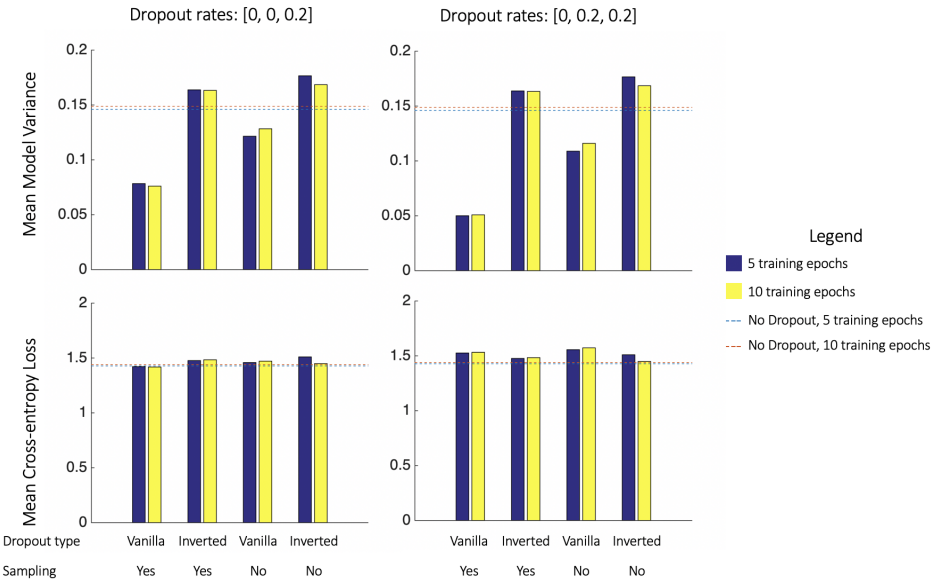

Figure 2: Comparison of model variance and cross-entropy loss at test time for models with different Dropout rates, Dropout types, training hyperparameters, and testing configurations. On the left are models with Dropout only in the third hidden layer, and on the right are models with Dropout in the second and third hidden layers.

4.3.1 Identifying Node Similarity through Clustering

Observe that a network node is represented by both an incoming weight vector and an outgoing weight vector such that node similarity naturally refers to similarity with respect to both of these vectors. As such, groups of similar nodes in a dropped layer can be identified by first individually clustering the rows of the weights of the dropped layer (incoming weights) and the columns of the weights of the following layer (outgoing weights). Subsequently, subsets of the nodes belonging to the same cluster in both clusterings can be identified as similar nodes. This method is preferable to crudely concatenating the incoming and outgoing weights together and performing a single clustering as it accommodates the possibly different dependencies on the incoming and outgoing weights.

We quantified the degree of node similarity by computing the Rand Index (Rand [1971]) between the two clusterings obtained for different Dropout rates in a fixed three-layer network. A Rand Index of 0 indicates absolute dissimilarity between two clusterings while 1 indicates complete similarity. It can be seen from Table 2 that the Rand Index generally increases with larger Dropout rates, suggesting that Dropout indeed introduces similar nodes.

Table 2: Rand Index for Different Dropout Rates

Dropout	0.0	0.2	0.4	0.6	0.8
Rand Index	0.65	0.81	0.77	0.82	0.84

4.3.2 Mean Sampling Standard Deviation with hidden units

Under the hypothesis that dropout results in clustered neurons, we propose a model to predict the relationship between the size of the hidden layers and the mean sampling standard deviation (MSTD).

Suppose $X_1, \ldots, X_h$ are the normalized outputs of a hidden layer of size h with $k \propto h^\kappa$ clusters for some $\kappa \in (0, 1)$. Assuming the variance of the output $\hat{y}$ is proportional to the variance of the mean of X_i, we have

$$\mathrm{Var}(\hat{y}) \propto \mathbb{E}\left(\frac{\sum_{i=1}^{h} X_i}{h}\right)^2 = \frac{1}{h^2}\mathbb{E}\left(\sum_{i=1}^{h} X_i^2 + 2\sum_{j\neq k} X_j X_k\right) = \frac{1}{h} + \frac{2}{h^2}\mathbb{E}\left(\sum_{j\neq k} X_j X_k\right) = \mathbb{E}_{j\neq k} X_j X_k + O\left(\frac{1}{h}\right) \tag{3}$$

Now we model the $C_1, \ldots, C_k$ clusters as follows: if $X_i, X_j \in C_p$ for some $i \neq j$, then the covariance $\mathbb{E}(X_i X_j) = \sigma$ and if they do not belong to the same cluster then the covariance $\mathbb{E}(X_i X_j) = 0$. Intuitively, this assumes that outputs of neurons in the same cluster

are correlated and outputs of neurons in different clusters are not. If we have the h neurons evenly distributed into k clusters, then the probability that two neurons are in the same cluster is $\frac{k}{h}$ and $\mathbb{E}_{j \neq k} X_j X_k = \frac{k\sigma}{h}$. Plugging this into the equation (3) yields

$$\mathrm{Var}(\hat{y}) \propto \frac{k}{h} + O\left(\frac{1}{h}\right) \propto h^{\kappa - 1} \tag{4}$$

To verify this, we used a three-layer network with a fixed Dropout rate enabled only on the first hidden layer and measured the effect of the number of units h in the first hidden layer on the MSTD (Figure 3). The fitted line very strongly suggests $\mathrm{MSTD} \propto \frac{1}{h^{0.35}}$, supporting our clustering perspective to Dropout's inner mechanism.

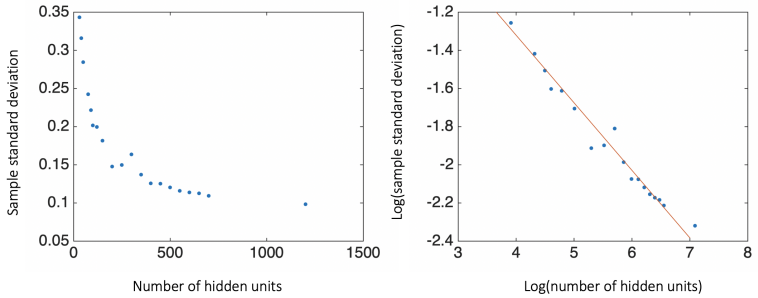

Figure 3: Comparison of sample variance of model for different number of hidden units in the first hidden layer. The left contains the normal plot and the right contains the log-log plot, with a best-fit line of $y = -0.3541x + 0.0958$ ($R^2 = 0.98$)

4.4 Model compression using clustering

The perspective of Dropout replicating nodes naturally leads to the following conversion algorithm from a stochastic to deterministic network. Firstly, the weights of a dropped layer and its following layer are each clustered into C groups using k-means clustering. From the two clusterings produced, we select the one with better empirical accuracy and compute the mean weight vector of each cluster in the chosen clustering for both weights. This produces two weight vectors for each cluster which can be associated with a compressed node in a new network—one denoting the incoming weight into a node of the dropped layer and another denoting the outgoing weight. As post-processing, both weights need to be scaled to account for the change in the number of nodes. Intuitively, the C compressed nodes are each representative of their corresponding clusters of similar nodes.

Table 3: Clustering Compression in a Three-Layer Network with Varying Units in First Layer

(a) With `0.5` Vanilla Dropout

Units	Test Acc	Compressed Units	Compressed Test Acc
400	0.869	162	0.855
500	0.879	162	0.847
600	0.884	167	0.833
700	0.877	177	0.839

(b) Without Dropout

Units	Test Acc	Compressed Units	Compressed Test Acc
400	0.882	162	0.740
500	0.894	162	0.685
600	0.892	167	0.534
700	0.891	177	0.657

When the algorithm was tested, C and the post-processing scaling factor were selected through linear search, as clustering criteria such as the Bayesian information and Calinski-Harabasz criteria produced mixed results. As seen in Table 3, our scheme only leads to a negligible decrease in evaluation accuracy, while significantly compressing the size of the network. As a control, similarly compressing an identical network trained without Dropout leads to drastic decreases in evaluation accuracy, suggesting that Dropout indeed induces clusters.

5 Conclusion

Dropout, despite being a prevalent regularization technique, lacks a conclusive explanation of its inner workings. In this paper, we conduct a series of experiments involving Dropout and suggest a possible mechanism behind Dropout from the perspective of stochastic neural networks, explaining its regularization effects. In particular, we examine Dropout's tendency to cluster nodes, introducing redundancies that reduce model complexity. For future work, we propose deriving a more comprehensive and unified theoretical model that explains the power relation between the number of hidden units and the MSTD as well as sampling's effect on model variance. The effectiveness of clustering compression could also be improved by only compressing groups of nodes that both clusterings agree on (as opposed to the current scheme of selecting one).

References

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

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, 2016a.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027, 2016b.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869, 2015.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. doi: 10.1080/01621459.1971.10482356. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356`.

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.

Yichuan Tang and Ruslan R Salakhutdinov. Learning stochastic feedforward neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 530–538. Curran Associates, Inc., 2013. URL `http://papers.nips.cc/paper/5026-learning-stochastic-feedforward-neural-networks.pdf`.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1058–III–1066. JMLR.org, 2013. URL `http://dl.acm.org/citation.cfm?id=3042817.3043055`.

6 Contributions

All authors contributed equally to this work. David Lin contributed to most of the writeup, as well as suggestions for all the experimental components. Jensen Wang created much of the experiments and writeup for clustering. Phil Chen created the experiments and writeup for model variance.