On the Importance of Stochasticity in DropOut

Surya Teja Chavali
Department of Computer Science
Roll no. CS13B1028
IIT Hyderabad
cs13b1028@iith.ac.in

Abstract

DropOut has been a highly successful and powerful trick in training Deep Neural Networks. It can be viewed as a means of model averaging - as a pseudo-ensemble method - as well as a stochastic regularizer. In this work, we seek to empirically study how important randomness is to DropOut over a series of experiments in which we deterministically drop some weights out in the network. We also further investigate the effect of sampling the weights to drop from a non-uniform distribution, namely the Gaussian.

1 Introduction and Motivation

DropOut[1], a common regularization mechanism in neural nets, can be viewed as a boosting strategy - training multiple models on small subsets of the data, and combining them by taking the (geometric)mean of their predictions. It has been studied under the framework of Model Averaging, as well as Weight Sharing.

We decided to investigate the role of stochasticity in DropOut as well as the character of the distribution in which we drop weights so as to glean further insights into why it works. [2] asks and answers many questions about DropOut, but the fundamental question of whether we can do better by dropping weights in a manner guided by intuition and maybe theoretical backing, remained. We specifically ask the following questions, predominantly viewing DropOut from the regularization perspective:

Firstly, it seems like DropOut samples several sparse models at random, and averages them. Can we re-weight a (possibly smaller) number of sparse models and average them? It also seems that weights which have abnormally small or large magnitudes are probably over-fitting, because they seemed to capture a correlation which was idiosyncratic to the training data - impinging on generalization ability. We tried to answer this by dropping weights in the band of $[\mu-p\sigma, \mu+p\sigma]$ while training the net, where μ is the mean weight, σ is the standard deviation of the weights, and p ($0<p\leq1$) is a user-specified parameter. Then, we multiply all weights by 0.5 during test time. We also try a variant where a weight is divided by the (number of times it was dropped + 1).Further, we checked if using the *absolute values* of the weights, rather than the weights themselves, mattered. We term this dropping of weights as **rethink**.

We also tried to *complement* the DropOut - that is, if we drop out weights that were included, and keep weights that were dropped out during training, will that help? In other words, are these so-called ‘high correlations’ the ‘essence’ of the model - the ones that really matter, and need to be fine-tuned?

We found uniformly in all these experiments, that stochasticity seems to be among the most important reasons as to why DropOut works.

Secondly, we asked the question of whether the uniform distribution is the best distribution to sample the weights to be dropped. More specifically, we tried to see if sampling numbers from the Gaussian Distribution with mean μ and standard deviation σ, and dropping weights within a small neighbourhood of these numbers, helped. It did not seem so as well.

2 Related Work

DropOut has been studied extensively in many works, by many researchers in the past. [2] studies DropOut by means of a several interesting experiments, and concludes that DropOut's approximation to the GM of the models it averages is pretty good. Further, the authors find that using the GM instead of the AM in the model doesn't change the result much. They also conclude that weight sharing is the key to DropOut's success.

[3] and [4] study DropOut in Linear Models. [5] studies DropOut in Linear and Sigmoidal Models and characterizes its approximation to the Geometric Mean. [6] shows that the difference between AM and GM is bounded.

[7] shows that, in MLPs with no non-linearity applied to each unit, with SoftMax output, DropOut is equivalent to GM-based model averaging.

[8],[9] investigated the fidelity of the weight scaling approximation in the context of rectifier networks and maxout networks, respectively, through the use of a Monte Carlo approximation to the true model average.

3 Experimental Setup

In this work, we considered the MNIST[15] digit Classification problem as the 'stock' dataset; however, we also generated some artificial, non-linearly separable data for the sake of rigour and completeness.

The MNIST dataset, with 32x32 images was used, after vectorizing it to a 1024 dimensional vector, and normalizing all pixel values by dividing them by 255.

The basic architecture of the Neural Net we used was one input layer, two hidden layers, and an output layer. The value of p in the input to the first hidden layer is half the value of p in other layers - in all experiments. The reason for this is that we, in keeping with Hinton et. al., use a smaller DropOut rate for the input features.

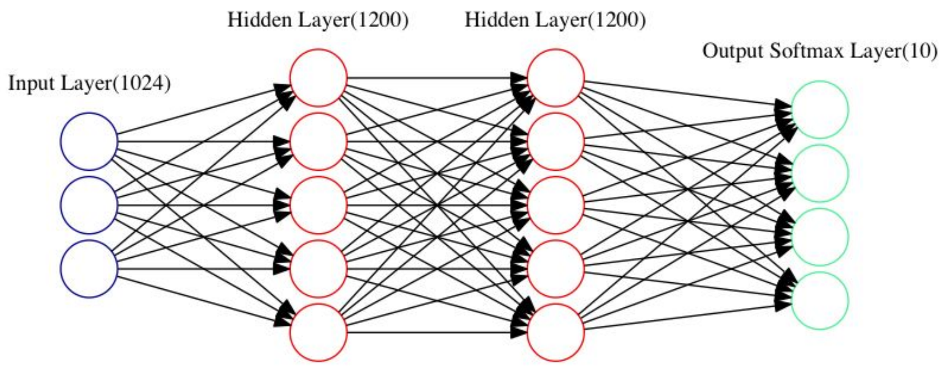

Figure 1: The architecture of the net for the MNIST dataset.

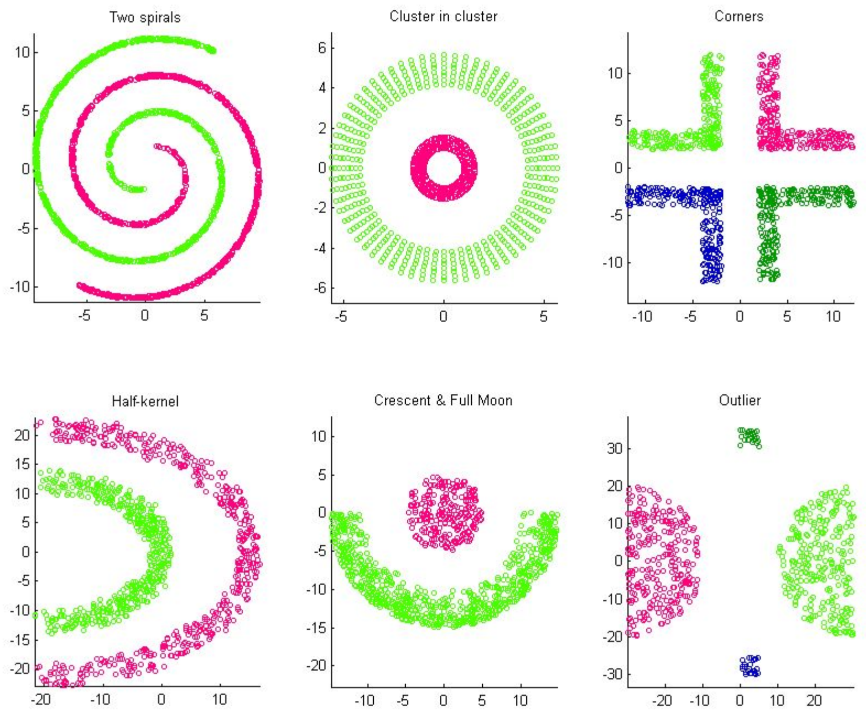

Figure 2: The six artificial datasets generated. We used a net with two inputs, 10 neurons in each hidden layer, and a softmax output layer with 2 or 4 outputs as is the number of classes.

Six artificial datasets were generated, and tested with a similar architecture, as shown in Fig.2.

In each experiment, we considered three activation functions: ReLU, tanh and sigmoid. The value of p was increased from 0.05 to 1 in steps of 0.05. The 'rethink' phase of the net was applied epochally. Loss and error were plotted.

4 The Heuristics

We basically tested seven ideas in this work:

- Drop weights in the band $[\mu - p\sigma, \mu + p\sigma]$ during training. Halve the weights while testing.
- Drop weights outside the band $[\mu - p\sigma, \mu + p\sigma]$ during training. Halve the weights while testing.
- Drop weights in the band $[\mu - p\sigma, \mu + p\sigma]$ during training. Divide weights by (#of times dropped) + 1 while testing.
- Drop weights outside the band $[\mu - p\sigma, \mu + p\sigma]$ during training. Divide weights by (#of times dropped) + 1 while testing.
- Drop weights whose absolute values are in the band $[\mu - p\sigma, \mu + p\sigma]$ during training.
- Drop weights whose absolute values are outside the band $[\mu - p\sigma, \mu + p\sigma]$ during training.
- Use a Gaussian to sample px(#of weights) numbers, and drop the weights in a small band of each sampled number during training.

5 Results

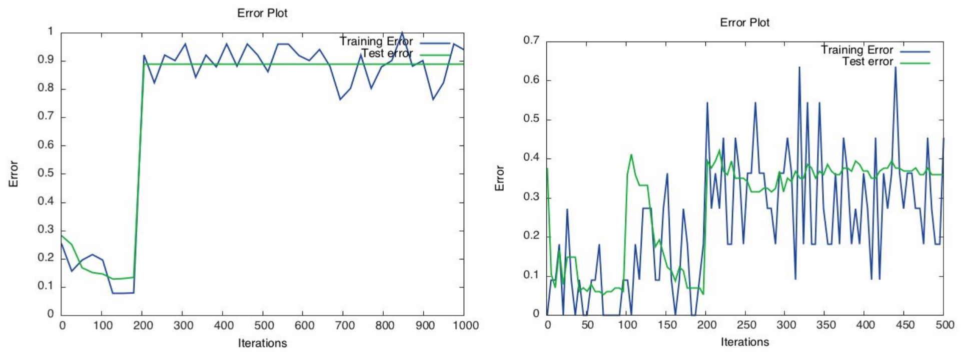

Figure 3: Two representative error plots for **all deterministic dropping strategies** discussed above. The error is low for the first epoch, where there is no dropping, and it then shoots up after that, never to come down. We have MNIST on the left, and cluster-in-cluster on the right.

The result of this study has been **resoundingly in favour of randomness** in DropOut. In all of the deterministic dropping strategies, error shoots up after the first epoch, never to come down.The same result has been observed with ReLUs as well as tanh units, with different intensity. However, it has been more pronounced in MNIST than in the artificial datasets. Sigmoid activations have shown a more or less constant error throughout training, however, in all experiments. This has been the case even after running the training for a large number of epochs.

However, in the case of sampling weights from a Gaussian, we observe a peculiar phenomenon. After the first epoch, the error sharply increases or decreases, and settles down. In other words, we 'converge' after the first epoch to more or less the same error rate. Otherwise, the error remains stable barring a few fluctuations right from the beginnning.

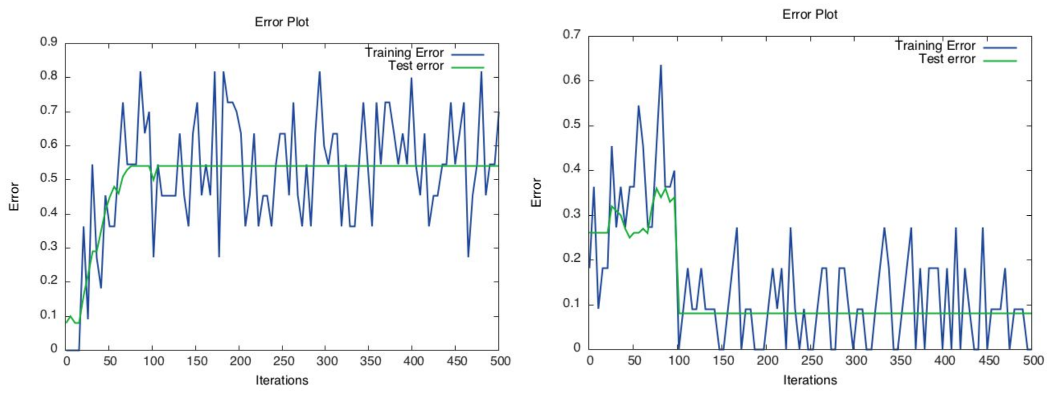

Figure 4: The curious case of the Gaussian sampling. We increase or decrease after the first epoch, and then remain stable. These are plots of what happens with the 'outlier' dataset.

6 Conclusions and Emerging Directions

It seems the randomness of DropOut/DropConnect allows them to theoretically average the entire space of models, while our determinism restricts the space of models to a small(and particularly bad?) subset of the model space, causing us to under-fit miserably. This hypothesis is strengthened by the fact that, dropping 'in band' weights as well as 'out of band' weights has given rise to similar results: no weight is less important in a 'global' sense. United in update, the weights stand; divided, they fall.

However, there seem to be a few other directions to pursue in this work before we conclude that sampling weights to drop from the uniform distribution is the best.

On the other hand, we feel there is a large scope for interpretation of DropOut in the sense of Pseudo-Ensembles, as well as in the regularization sense.

Acknowledgments

I'd like to acknowledge the extreme co-operation of my professor, Dr.Vineeth during the course of this project for taking time out to meet me - indeed, we spoke many times rather late at night! I would also like to thank the VIGIL Lab and Adepu Ravi Sankar for giving me an account in the GPU Server. I am very thankful to Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet for Torch7[10], the Machine Learning framework in which I implemented all this. Further, I'd like to acknowledge the work of Jeroen Kools for the MATLAB® code to generate the artificial datasets[11], Keith Brady for MATLAB® code to read CSV files with column headers[12], and John-Alexander M. Assael for his Torch7 implementation of DropConnect[13]. I'd also like to thank Nando de Freitas for his code on his repository for the Oxford Neural Nets course[14].

References

[1] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R.(2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* **15**(2014):1929-1958.

[2] Warde-Farley,D., Goodfellow,I. J., Courville, A., & Bengio, Y.(2014) An empirical analysis of dropout in piecewise linear networks. In *ICLR*, 2014.

[3] Baldi, P. and Sadowski, P. J. (2013). Understanding dropout. In *Advances in Neural Information Processing Systems* 26, pages 2814–2822.

[4] Wager, S., Wang, S., and Liang, P. (2013). Dropout training as adaptive regularization. In Advances in Neural Information Processing Systems 26, pages 351–359.

[5] Baldi, P. and Sadowski, P. J. (2013). Understanding dropout. In Advances in Neural Information Processing Systems 26, pages 2814–2822.

[6] Cartwright, D. I. and Field, M. J. (1978). A refinement of the arithmetic mean-geometric mean inequality. Proceedings of the American Mathematical Society, 71(1), pp. 36–38.

[7] Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013a). Maxout networks. In ICML'2013.

[8] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinv, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

[9] Srivastava, N. (2013). Improving Neural Networks With Dropout. Master's thesis, U. Toronto

[10] Collobert, R., Kavukcuoglu, K., & Farabet, C. Torch7: A Matlab-like Environment for Machine Learning. *NIPS* 2011.

[11]6 functions for generating artificial datasets, by Jeroen Kools, 23 Apr 2013, retrieved from http://www.mathworks.com/matlabcentral/fileexchange/41459-6-functions-for-generating-artificial-datasets

[12]CSV with column headers, by Keith Brady, 06 Jan 2011, retrieved from http://in.mathworks.com/matlabcentral/fileexchange/29933-csv-with-column-headers

[13]Torch7 implementation of DropConnect by John-Alexander M. Assael, retrieved from https://github.com/iassael/torch-dropconnect

[14] MNIST dataset download code, Nando de Frietas, retrieved from https://github.com/oxford-cs-ml-2015/practical3

[15] LeCun, Y., Cortes, C., Burges, C.J.C., The MNIST Database of handwritten digits.