Dropout is a special case of the stochastic delta rule: faster and more accurate deep learning

Noah Frazier-Logue
Rutgers University Brain Imaging Center
Rutgers University - Newark
Newark, NJ 07103
n.frazier.logue@nyu.edu

Stephen José Hanson
Rutgers University Brain Imaging Center
Rutgers University - Newark
Newark, NJ 07103
jose@rubic.rutgers.edu

Abstract

Multi-layer neural networks have lead to remarkable performance on many kinds of benchmark tasks in text, speech and image processing. Nonlinear parameter estimation in hierarchical models is known to be subject to overfitting. One approach to this overfitting and related problems (local minima, colinearity, feature discovery etc.) is called "dropout" (Srivastava, et al 2014, Baldi et al 2016). This method removes hidden units with a Bernoulli random variable with probability p over updates. In this paper we will show that Dropout is a special case of a more general model published originally in 1990 called the stochastic delta rule ("SDR", Hanson, 1990). SDR parameterizes each weight in the network as a random variable with mean $\mu_{w_{ij}}$ and standard deviation $\sigma_{w_{ij}}$. These random variables are sampled on each forward activation, consequently creating an exponential number of potential networks with shared weights. Both parameters are updated according to prediction error, thus implementing weight noise injections that reflect a local history of prediction error and efficient model averaging. SDR therefore implements a local gradient-dependent simulated annealing per weight converging to a bayes optimal network. Tests on standard benchmarks (CIFAR) using a modified version of DenseNet shows the SDR outperforms standard dropout in error by over 50% and in loss by over 50%. Furthermore, the SDR implementation converges on a solution much faster, reaching a training error of 5 in just 15 epochs with DenseNet-40 compared to standard DenseNet-40's 94 epochs.

1 Introduction

Multi-layer neural networks have lead to remarkable performance on many kinds of benchmark tasks in text, speech and image processing. Nonetheless, these deep layered neural networks also lead to high-dimensional, nonlinear parameter spaces that can prove difficult to search and lead to overfitting and poor generalization performance. Earlier neural networks using back-propagation, failed due to lack of adequate data, gradient loss recovery, and high probability of capture by poor local minima. Deep-learning (Hinton et al, 2006) introduced some innovations to reduce and control these overfitting and over-parameterization problems, including rectified linear neural units (ReLu), to reduce successive gradient loss and Dropout in order to avoid capture by local minimum and increase generalization by effective model-averaging. In this paper we will focus on the over-parameterzation of the deep-layered networks despite the tsunami of data that is now available for many kinds of classification and regression tasks. Dropout is a method that was created to mitigate the over-parameterization and therefore overfitting of deep-learning applications and incidentally to avoid poor local minima. Specifically, Dropout implements a Bernoulli random variable with probability p ("biased coin-toss") on each update to randomly remove hidden units and its connections from the network on each update producing a sparse network architecture in which the remaining weights are

Preprint. Work in progress.

updated and retained for the next Dropout step. At the end of learning the DL network is reconstituted by calculating the expected value for each weight $p_{w_{ij}}$ which approximates a model-averaging over an exponential set of networks. Dropout with DL, has been shown to reduce errors on common benchmarks by more then 50% in most cases.

In the rest of this paper we will introduce a general type of Dropout that operates at the weight level and injects gradient dependent noise on each update called the stochastic delta rule (cf. Murray & Andrews, 1991). SDR implements a random variable per weight and provide update rules for each parameter in the random variable, in this case a gaussian with adaptive parameters ($\mu_{w_{ij}}$, $\sigma_{w_{ij}}$. Although any SDR would work with any random variable (gamma, beta, binomial, etc..). We will show that Dropout is a special case with a binomial random variable with fixed parameters (np, $np(1-p)$). Finally we will test DenseNet architectures on standard benchmarks (e.g. CIFAR-10 and CIFAR-100) with Gaussian SDR and show a considerable advantage over bninomial Dropout.

2 Stochastic delta rule

It is known that actual neural transmission involves noise. If a cortically isolated neuron is cyclically stimulated with the exact same stimuli will never result in the same response (Burns, etc). Part of the motivation for SDR is based on the stochastic nature of signals transmitted through neurons in living systems. Obviously smooth neural rate functions are based on considerable averaging over many stimulation trials. This leads us to an implementation that suggests a synapse between two neurons could be modeled with a distribution with fixed parameters. The possible random variables associated with such a distribution are in the time-domain likely to be a Gamma distribution (or in binned responses; see Poisson, Burns). Here we assume a central limit theorem aggregation of independent and identically distributed random variables and adopt a Gaussian as a general form. Although, there may be an advantage to longer tail distributions in that same sense skew is required for independent component analysis (ICA).

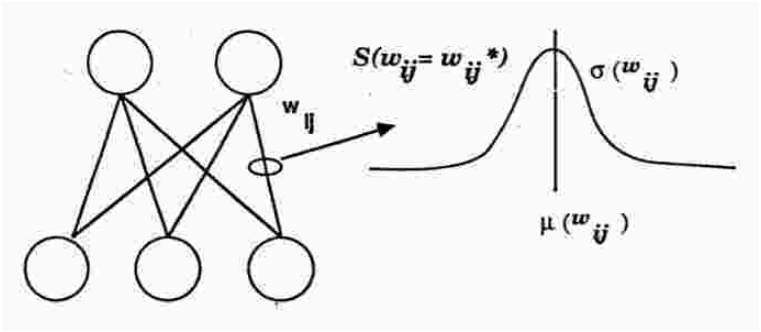

Figure 1: SDR sampling.

At present we therefore implement the SDR algorithm with a Gaussisan random variable with mean $\mu_{w_{ij}}$ and $\sigma_{w_{ij}}$ shown in figure 1. Each weight will consequently be sampled from the gaussian random variable as a forward calculation. In effect, similar to Dropout, and exponential set of networks are sampled over updates during training. The difference at this point with Dropout is that SDR adjusts the weights and the effect of the hidden unit attached to each weight to change adapatively with the error gradient on that update. The effect here again is similar to Dropout, except each hidden unit's response is factored across its weights (proportional to their effect on the credit assignment from the classification error). Consequently, each weight gradient itself is a random variable based on hidden unit prediction performance allowing for the system to (1) entertain multiple response hypotheses given the same exemplar/stimulus and (2) maintain a prediction history, unlike Dropout, that is local to the hidden unit weights, is conditional on a set or even a specific exemplar and finally (3) potentially back out of poor local minima that may result from greedy search but at the same time distant from better local minima. The consequence of the local noise injection has a global effect on the network convergence and provide the DL with greater search efficiency which we examine later on. A final advantage, suggested by G. Hinton, was that the local noise injection may

through model averaging smooth out the ravines and gullys bounded by better plauteaus in the error surface, allowing quicker and more stable convergence to better local minima.

The implementation of SDR involves 3 independent update rules for the parameters of the random variable representing each weight and the model averaging parameter causing the network search to eventually collapse to a single network effectively averaged over all sampled networks/exemplars/updates. Initially a forward pass through the network involves random sampling from each weight distribution independently, producing a w_{ij}^* per connection. This weight value is subsequently used below in the weight distribution update rules:

$$S(w_{ij} = w_{ij}^*) = \mu_{w_{ij}} + \mu_{w_{ij}}\theta(w_{ij}; 0, 1)$$

The first update rule refers to the mean of the weight distribution:

$$\mu_{w_{ij}}(n+1) = \alpha(\frac{\partial E}{\partial w_{ij}^*}) + \mu_{w_{ij}}(n)$$

and is directly dependent on the error gradient and has learning rate α. This is the usual delta rule update but conditioned on sample weights thus causing weight sharing through the updated mean value. The second update rule is for the standard deviation of the weight distribution (and for a Gaussian is known to be sufficient for identification).

$$\sigma_{w_{ij}}(n+1) = \beta|\frac{\partial E}{\partial w_{ij}^*}| + \sigma_{w_{ij}}(n)$$

Again note that the standard deviation of the weight distribution is dependent on the gradient and has a learning rate of β. And once again weight sharing is linked through the value of the standard deviation based on the w_{ij}^* sample. A further effect of the weight standard deviation rule is that as gradient error for that mean weight value (on average) increases the hidden unit those weights connect to, is getting more uncertain and more unreliable for the network prediction. Consequently we need one more rule to enforce the final weight average over updates. This is another standard deviation rule (which could be combined above—however for explication purposes we have broken it out) forces the noise to "drain" out over time assuming mean and standard deviation updates are not larger then the exponential reduction of noise on each step. This rule forces the standard deviation to converge to zero over time, causing the mean weight value to a fixed point aggregating all of the networks/updates over all samples.

$$\sigma_{w_{ij}}(n+1) = \zeta\sigma_{w_{ij}}(n+1), \zeta < 1.$$

Compared to standard backpropagation, Hanson (1990), showed in simple benchmark cases using parity tests that SDR would with high probability ($> .99$) converge to a solution while standard backpropagation (using 1 hidden layer) would converge less then 50% of the time. The scope of problems that SDR was used with often did not find a large difference if the classification was mainly linear or convex as was the case in many applications of back-propagation in the 1990s.

Next, we turn to how dropout can be shown to be a special case of SDR. The most obvious way to see this is to first conceive of the random search as a specific sampling distribution.

3 Dropout as binomial fixed parameter SDR

Dropout as described before requires that hidden units per layer (except output layer) be removed in a Bernoulli process, which essentially implements a biased coin flip at $p = 0.3$, ensuring that some hidden units per layer will survive the removal leaving behind a sparser or thin network. This process as described before also, like SDR, produces weight sharing and model averaging, reducing the effects of over-fitting. To put the Dropout algorithm in probablistic context, consider that a Bernoulli

random variable over many trials results in a Binomial distribution with mean np and standard deviation $(np(1-p))$. The random variable is the number of removals ("successes") over learning on the x axis and the probability that the hidden unit will be removed with Binomial $(np, np(1-p))$. If we compare Dropout to SDR in the same network, the difference we observe is in terms of whether the random process is affecting weights or hidden units. In Figure 3, we illustrate the convergence of Dropout as hidden unit Binomial sampling. It can readily be seen that the key difference between the two is that SDR adaptively updates the random variable parameters for subsequent sampling and Dropout samples from a Binomial random variable with fixed parameters (mean, standard deviation at p). One other critical difference is that the weight sharing in SDR is more local per hidden unit than that of Dropout, but is essentially the same coupling over sampling trials with Dropout creating an equivalence class per hidden unit and thus creating a coarser network history.

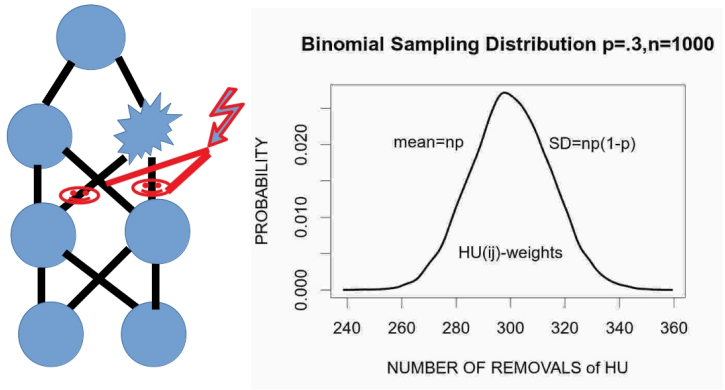

Figure 2: Dropout sampling.

Showing that SDR is a generalized form of dropout opens the door for many kinds of variations in random search, that would be potentially more directed and efficient than the fixed parameter search that Dropout represents. It is possible that longer tailed distribution for example (Gamma, Beta, LogNormal..etc) would be more similar to the distributions underlying neural spikes trains, thus allowing us to provide further connections to the stochastic nature of neural transmission. More critically in this context, does the increase in parameters that SDR represents provide more efficient and robust search that would increase performance in classification domains already well tested with many kinds of variations of deep learning with Dropout?

In what follows we implement and test a state art DL neural network, in this case DenseNet (Huang, 2017) with standard benchmark image tests (CFAR-10, CIFAR-100). Here we intend to show paired tests with Tensorflow implementations holding the learning parameters (except for the random search algorithms—SDR or dropout) constant over various conditions.

4 Tests

We used a modified DenseNet model implemented in TensorFlow, originally by I. Khlyestov (2017). The model uses a DenseNet-40, DenseNet-100, and DenseNet-BC 100 network trained on CIFAR-10 and CIFAR-100, with a growth rate of $k = 12$, batch size of 100, and 100 epochs, with all other parameters being the same as the original DenseNet implementation. Further work is being done to extend the variety in parameters and datasets (similar to the experiments done with the original DenseNet implementation). The model without SDR used a dropout rate of 0.2, i.e a 20% chance that each neuron is dropped out. The SDR implementation used parameters $\beta = 0.1, \zeta = 0.01$. α, included in the first formula of the update rule from Hanson (1990), was found to be extraneous in this implementation, likely due to the inclusion of the learning rate and momentum (not yet used at the time of Hanson (1990)). The code used for implementing and testing SDR can be found at https://github.com/noahfl/densenet-sdr/.

5 Results

Table 1: Error rates of DenseNet-SDR to DenseNet with dropout

	Dataset	
Model	CIFAR-10	CIFAR-100
DenseNet-40 (k=12)	5.160	22.60
DenseNet-100 (k=12)	3.820	11.06
DenseNet-BC 100 (k=12)	6.340	25.08
DenseNet-40 with SDR (k=12)	2.256	9.36
DenseNet-100 with SDR (k=12)	**1.360**	**5.160**
DenseNet-BC 100 with SDR (k=12)	2.520	11.12

These results show that replacing dropout with SDR in DenseNet tests yields an over 50% reduction in error on all CIFAR benchmarks, with decreases of up to 64%. Error results for the original DenseNet implementation are lower than in the original DenseNet paper as it was found that using a larger batch size resulted in higher overall accuracy.

Table 2: Number of epochs taken to reach training errors of 15, 10, and 5, respectively

	Dataset	
Model	CIFAR-10	CIFAR-100
DenseNet-40 (k=12)	8 | 16 | 94	85 | – | –
DenseNet-100 (k=12)	8 | 13 | 25	28 | 60 | –
DenseNet-BC 100 (k=12)	10| 25 | –	– | – | –
DenseNet-40 with SDR (k=12)	**5** | **8** | **15**	27 | 48 | –
DenseNet-100 with SDR (k=12)	6 | 9 | **15**	**17** | **21** | **52**
DenseNet-BC 100 with SDR (k=12)	**5** | **8** | 17	31 | 87 | –

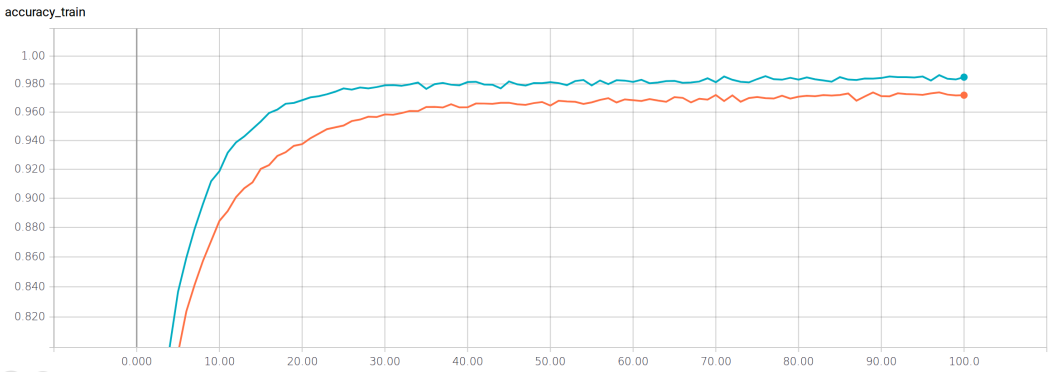

Figure 3: Training accuracy over the course of 100 epochs for DenseNet-100 with dropout (shown in orange) and DenseNet-100 with SDR (shown in blue). SDR not only increases its training accuracy faster than dropout (reaching 96% accuracy in 17 epochs to dropout's 33), it reaches 98+% accuracy within 40 epochs.

As shown in Table 2, time taken to fall below error rates of 15, 10, and 5 during training is much shorter when using SDR. DenseNet-40 takes one sixth of the number of epochs to reach an error of 5 when using SDR and DenseNet-100 requires 40% fewer epochs to reach an error of 5.

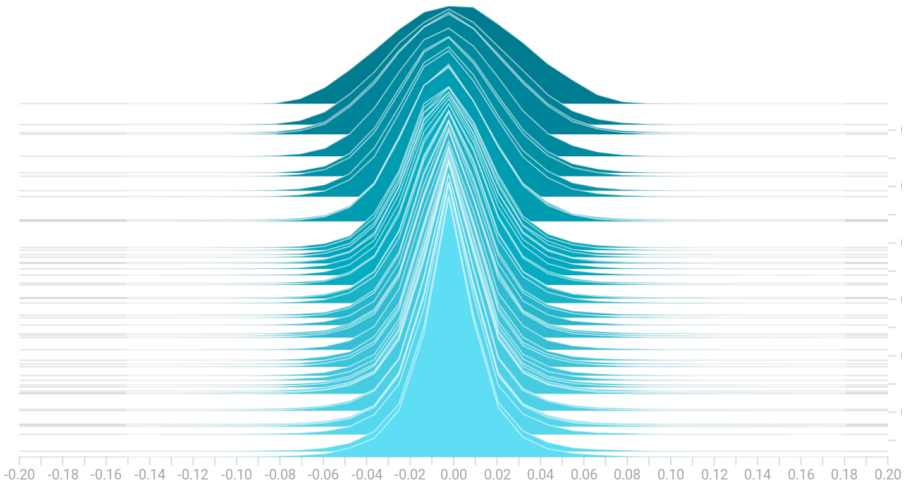

Figure 4: Histogram showing the frequency of weight values in block 1, layer 21 of DenseNet-100 with SDR, where each slice is a snapshot from one of the training epochs and the topmost slice is from the first epoch. Over the course of the 100-epoch training period, the plot narrows as the weights' standard deviations converge towards zero.

6 Discussion

In this paper we have shown how a basic deep learning algorithm (Dropout) that implements stochastic search and helps prevent over-fitting is a special case of an older algorithm, the Stochastic Delta Rule, which is based on a Gaussian random sampling on weights and adaptable random variable parameters (in this case a mean value and a standard deviation). Further, we were able to show how SDR outperforms Dropout in a state of the art DL classifier on standard benchmarks, showing improvements in loss by 50+% and improvements in accuracy of 50+%, and improvements in the rate at which low error rates are reached during the training phase. From an implementation standpoint, it is straightforward to implement and can be written in approximately 30 lines or less of code. All

that is required is access to the gradients and to the weights; SDR can be inserted into virtually any training implementation with this access. SDR opens up a novel set of directions for DL search methods which include various random variable selections that may reflect more biophysical details of neural noise, or provide more parameters to code the prediction history within the network models, thus increasing the efficiency and efficacy of the underlying search process.

References

[1] Baldi, P. and PJ. Sadowski, Understanding Dropout, In Advances in Neural Information Processing Systems 26 ppg:2814:2822, 2013

[2] Burns, B. Delisle. The Uncertain Nervous System. Edward Arnold Publ., 1968.

[3] Hanson, S. J. "A Stochastic Version of the Delta Rule", Physica D, vol. 42, pp. 265-272, 1990.

[4] Hinton, G. E. and Salakhutdinov, R. R Reducing the dimensionality of data with neural networks. Science, Vol. 313. no. 5786, pp. 504 - 507, 28 July 2006.

[5] Huang G. ,Liu, Z. Weinberger, K, & Maaten, L. (2017) Densely Connected Convolutional Networks. arXiv:1608.06993

[6] Khlyestov, I. DenseNet with TensorFlow. (2017) GitHub repository. https://github.com/ikhlestov/vision_networks

[7] Murray, A. F., "Analog Noise-Enhanced Learning in Neural Network Circuits," Electronics Letters, vol. 2, no. 17, pp. 1546-1548, 1991.

[8] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. http://arxiv.org/abs/1207.0580, 2012.