LETTER Communicated by Terrence Sejnowski

The Stochastic Delta Rule: Faster and More Accurate Deep Learning Through Adaptive Weight Noise

Noah Frazier-Logue
nfrazier-logue@research.baycrest.org
Rotman Research Institute, Baycrest Health Sciences, Toronto, ON M6A 2E1, Canada

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

Multilayer neural networks have led 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 and misspecification. One approach to these estimation and related problems (e.g., saddle points, colinearity, feature discovery) is called Dropout. The Dropout algorithm removes hidden units according to a binomial random variable with probability p prior to each update, creating random "shocks" to the network that are averaged over updates (thus creating weight sharing). In this letter, we reestablish an older parameter search method and show that Dropout is a special case of this more general model, stochastic delta rule (SDR), published originally in 1990. Unlike Dropout, SDR redefines each weight in the network as a random variable with mean $\mu_{w_{ij}}$ and standard deviation $\sigma_{w_{ij}}$. Each weight random variable is sampled on each forward activation, consequently creating an exponential number of potential networks with shared weights (accumulated in the mean values). Both parameters are updated according to prediction error, thus resulting in weight noise injections that reflect a local history of prediction error and local model averaging. SDR therefore implements a more sensitive local gradient-dependent simulated annealing per weight converging in the limit to a Bayes optimal network. We run tests on standard benchmarks (CIFAR and ImageNet) using a modified version of DenseNet and show that SDR outperforms standard Dropout in top-5 validation error by approximately 13% with DenseNet-BC 121 on ImageNet and find various validation error improvements in smaller networks. We also show that SDR reaches the same accuracy that Dropout attains in 100 epochs in as few as 40 epochs, as well as improvements in training error by as much as 80%.

Neural Computation **32, 1018–1032** (2020)
https://doi.org/10.1162/neco_a_01276

1 Introduction

Deep learning has made advances in text, image, and speech analysis and processing, many times exceeding the normative benchmarks by significant margins. Nonetheless, these deep-layered neural networks also lead to high-dimensional, nonlinear parameter spaces that can prove difficult to search and lead to overfitting, model misspecification, and poor generalization performance. Earlier shallow (e.g., one hidden layer) neural networks using backpropagation were unable to solve diffcult problems due to a lack of adequate data, gradient loss recovery, and a high probability of capture by local minima or poor starting points. Deep learning (Hinton & Salakhutdinov, 2006) introduced some innovations to reduce and control these overfitting and model misspecification problems, including rectified linear units (ReLU), to reduce successive gradient loss and Dropout in order to avoid saddle points and increase generalization by effective model-averaging.

In this letter, we focus on the parameter search in the deep-layered networks. Dropout implements a binomial random variable with probability p (biased coin toss) on each update to randomly remove hidden units and their connections from the network, producing a sparse network instantiation in which the remaining weights are 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 $\mu_{w_{ij}}$ over all network shocks, which approximates model averaging over a potentially exponential set of networks. Dropout continues to be a basic regularization (Hanson & Pratt, 1989) method in the DL toolbox and has been part of all state-of-the-art DL architectures; in fact, recent benchmark improvements have all been implemented with Dropout, such as Efficient Net (Tan & Le, 2019) and GPipe (Huang et al., 2019), using AmoebaNet (Real, Aggarwal, Huang, & Le, 2019) with dropout $p = 0.5$. Also note that many improvements on benchmarks involve extra data augumentation, which is not a focus here but would be expected to have similar increases in SDR performance.

The goals for implementing SDR in a state-of-the-art DL architecture were threefold: (1) to scale SDR from a single hidden-layer architecture in the 1990s, to the typical 100 or more layers, thousands of hidden units, and millions of weights in the current DL architecture; (2) to compare SDR against Dropout to test whether more local, punctate, adaptive noise injection provides more efficient and faster search than the necessarily coarser search of Dropout; and (3) to create a more general framework for exploration of adaptive noise injection neural networks. In this context, we show that Dropout is a special case of SDR, with a binomial random variable with fixed parameters $(np, np(1-p))$, which can be implemented per layer.

2 Stochastic Delta Rule

Part of the original motivation for SDR was based on the stochastic nature of signals transmitted through neurons in living systems (Faisal, Selen, &

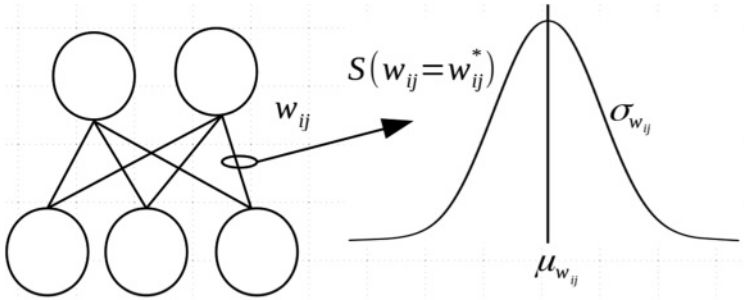

Figure 1: SDR sampling.

Wolpert, 2008). Although we want to make it clear that we are not attempting to provide a realistic model of neural firing, rather like many neural network models since the earliest neural network, SDR was inspired by biological processes that are in the spirit of the potential computation. This led to an implementation of a weight (synapse) between two nodes (neurons) modeled with a sampling distribution over weight values indexed by adaptive parameters. In the same vein, a biologically plausible random variable associated with such a distribution is in the time domain likely to be a gamma distribution (or in binned responses; see, e.g., clustered Poisson distributions Burns, 1968). Here for simplification, we assume a central limit theorem aggregation of independent and identically distributed random variables and adopt a gaussian as a general form.

Specifically we implement the SDR algorithm with a gaussian random variable with mean $\mu_{w_{ij}}$ and $\sigma_{w_{ij}}$ shown in Figure 1. Each weight will be sampled from the gaussian random variable during a forward calculation—in effect, similar to Dropout, with the set of sparse networks sampled over updates during training. A key difference between SDR and Dropout is that SDR injects noise in the weights and causes the hidden unit attached to each weight to change adaptively as a function of the error gradient at that update. The effect here again is similar to Dropout, except each hidden unit's response is dependent on its weights (proportional to their effect on the prediction error). Consequently, each weight gradient itself is a random variable based on hidden unit prediction performance allowing the system to (1) entertain multiple response hypotheses given the same exemplar or stimulus input vector; (2) maintain a prediction history, unlike Dropout, that is local to the hidden unit weights and conditional on a set of inputs; and (3) potentially back out of error extrema ("saddle points") that may result from greedy search but at the same time be distant from the global minima (Dauphin et al., 2014; Kawaguchi, 2016). The consequence of the local noise injection has a global effect on network convergence and provides the network with greater search efficiency, which we examine. A final advantage, suggested by G. Hinton (personal communication, 1990),

was that the local noise injection may, through model averaging, smooth out the ravines and gullies bounded by better plateaus in the error surface, allowing faster and more stable convergence to better minima.

The implementation of SDR involves three 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, and 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:

$$S(w_{ij} = w_{ij}^*) = \mu_{w_{ij}} + \mu_{w_{ij}}\theta(w_{ij}; 0, 1). \tag{2.1}$$

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

$$\mu_{w_{ij}}(n+1) = \alpha\left(-\frac{\partial E}{\partial w_{ij}^*}\right) + \mu_{w_{ij}}(n), \tag{2.2}$$

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 implementing 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 its identification):

$$\sigma_{w_{ij}}(n+1) = \beta\left|-\frac{\partial E}{\partial w_{ij}^*}\right| + \sigma_{w_{ij}}(n) \tag{2.3}$$

Again note that the standard deviation of the weight distribution is dependent on the gradient and has a learning rate of β. The uncertainty of the weight is linked through the value of the standard deviation based on the w_{ij}^* sample. The larger the standard deviation of the sampling distribution, the more variable the forward pass sample will be over trials. The weight standard deviation rule (due to the absolute value on the gradient error) can only increase or maintain a constant weight sampling error, thus, in general, making that weight more uncertain and more unreliable for the network prediction. Consequently, we need one more rule (a third rule) to cause the convergence of the network weight standard deviations toward zero. This is another standard deviation rule (which could be combined above; however, for explication purposes, we have broken it out) that forces the network to anneal the noise over training, assuming the standard deviation updates are decreasing with respect to the annealing schedule. This rule forces the standard deviation to converge to zero over time, causing the mean weight

value to a fixed point and, in effect, aggregating all of the networks or updates over all samples that occurred through training:

$$\sigma_{w_{ij}}(n+1) = \zeta \sigma_{w_{ij}}(n), \zeta < 1. \tag{2.4}$$

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 one 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 task was mainly linear or convex, as was the case in many applications of backpropagation 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 conceive of the random search as a specific sampling distribution.

3 Variations of SDR

Besides the algorithm we described, many variations are possible given a specific distribution parameterized with the mean and standard deviation. These rules determine the rate of convergence and the noise dynamics. Any such implementation must trade off the strict following of the mean error gradient with the noise perturbations away from the mean error gradient without resulting in chaotic trajectories. The first method, reported in Hanson (1990), was SDR-Decay, which controlled noise through weight decay due to local simulated annealing schedules (see equation 2.3). This forced the network search to eventually converge to a single average network despite periodic errors in the prediction history. The second method, first tested in 1992 (Hanson, 1992), was a search with an independent gradient for both mean and standard deviation without rule 3, which allowed a longer and more sustained search. This method was also rediscovered by A. Graves (2011). We refer to it as SDR-dynamic.

3.1 SDR-Decay. SDR-Decay is the standard implementation of SDR as described above with some added features that adapt it to the larger scale of a deep-learning architecture. In this version, SDR was adapted specifically to densenet architectures. We expand the annealing parameter ζ within each epoch linearly across the hidden layers in such a way that during the SDR update, the ζ value applied to the output layer is approximately 90% of the ζ value applied to the input layer. ζ is also restarted across epochs that reset every five epochs according to the formula

$$\zeta' = \zeta * e^{\phi * \zeta_{track}} \tag{3.1}$$

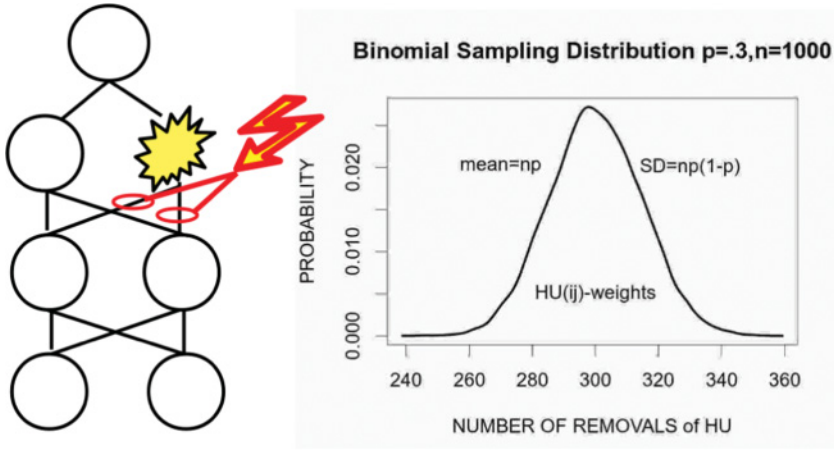

Figure 2: Dropout sampling.

where ζ_{track} is the number of epochs since the last epoch ζ was reset, ζ' is the modified ζ value, and ϕ controls the rate of decay. Within each epoch, updates to the standard deviations occur twice per epoch (halfway through the batch and at the end of the batch) for smaller data sets (CIFAR) and every 250 batches for larger data sets (ImageNet).

3.2 SDR-Dynamic. SDR-Dynamic removes ζ and the weight annealing schedules entirely. It also implements a minimum and maximum threshold on the values of the standard deviations. Once values of the standard deviations reach the minimum threshold, the value is reset in order to prevent the standard deviations from reaching zero and to maintain noise in the weight values. The maximum threshold for the standard deviations is set to 1. This particular variation was also reported by Fortunato et al. (2018) and Graves (2011). Their equations are identical to ours; however, we further attempt to control the dynamics of the noise, which can improve the learning curve stability and convergence.

4 Dropout as Binomial Fixed-Parameter SDR

Dropout requires that hidden units per layer (except the output layer) be removed in a binomial process, which essentially implements a biased coin flip at $p = 0.2$, ensuring that some hidden units per layer will survive the removal, leaving behind a sparser or thin network. This process is shown in Figure 2, and like SDR, produces weight sharing and model averaging, reducing the effects of overfitting. To put the Dropout algorithm in probablistic context, consider that a binomial 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 (see Figure 2). In Figure 2, 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 given the removal probability, p). One other critical difference is that the weight sharing in SDR is more local per hidden unit than that of Dropout, but it is essentially the same process over the sampling (whether with gaussian or binomial), with Dropout creating an equivalence class per hidden unit, thus creating a coarser network history.

Showing that Dropout is a specialized form of SDR opens the door to many kinds of variations in random search that would be potentially more directed and efficient than the fixed parameter search that Dropout represents (although p could be nonstationary). 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-of-the-art deep neural network, in this case, DenseNet (Huang, Liu, Weinberger, & Maaten, 2017) with standard benchmark image tests (CFAR-10, CIFAR-100, ImageNet). Here we intend to show paired tests with PyTorch implementations holding the learning parameters (except for the random search algorithms—SDR or Dropout) constant over various conditions.

5 Implementation

Tests were conducted on a compute server with two Intel Xeon E5-2650 CPUs, 512 GB RAM, and two NVIDIA P100 GPUs. We used a modified DenseNet model implemented in PyTorch, originally by Veit (2017). The model without SDR used a Dropout rate of 0.2, that is, a 20% chance that each neuron is dropped out. The learning rate drop ratio is maintained as well, with α/LR dropping at both 50% and 75% through the entire learning run. α/LR values vary according to the data set and model size and are fixed for both dropout and SDR. The SDR implementation used parameters that varied according to the size of the network and the number of classes, but the values were generally around $\alpha = 0.25, \beta = 0.05, \zeta = 0.7$. We hyperbolically annealed ζ for smaller networks and exponentially annealed it for larger networks so as to reduce the influence of the standard

deviations as the model converges. The standard deviations were initialized using a halved Xavier initialization (Glorot & Benjio, 2010) and were updated twice every epoch, in the middle and at the end, for DenseNet-BC 250 and DenseNet-100, and after every batch for the others. (The number of updates per epoch has an effect on the overall performance and can be treated as a hyperparameter. We noticed that larger networks needed fewer gradient updates than smaller networks.) The propagation of ζ is split between the earlier layers and the deeper layers, with the ζ value in the earlier layers being 90% of the specified value. The code used for implementing and testing SDR is publicly available.[1] The overall cost of SDR architectures is minimal, in that only two extra parameters per weight are required and updates have an added random number call per trial with subseqent two updates per parameter and one annealing schedule.

5.1 CIFAR. Experiments are conducted on the CIFAR data set (Krizhevsky, 2009), which contains 60,000 32×32 RBG images divided into 10 and 100 classes. The model uses a DenseNet-40, DenseNet-100, and DenseNet-BC 250 network trained on CIFAR-10 and CIFAR-100, with a growth rate of $k = 12$, batch size of 100, and 100 epochs, with the other parameters being the same as the original DenseNet implementation.

5.2 ImageNet. Experiments are conducted on the ImageNet ILSVRC2012 challenge data set (Deng et al., 2009), containing 1.2 million images in 1000 classes. Experiments use a batch size of 128 and are run to 100 epochs. σ updates occur every 250 batches and approximately 40 times per epoch. Images are resized and cropped to 224×224 and normalized.

6 Results

These results show that replacing Dropout with SDR in DenseNet tests yields various reductions in error in smaller networks, and 13% reduction in errors of top-5 accuracy on ImageNet (see Figure 3 and Table 1).

Tests were conducted to determine improvements in training error across benchmarks. As shown in Table 2, SDR shows an 80% or more reduction in training error across the majority of the benchmarks. The ImageNet result sees an almost three-point difference in error for a 9.3% decrease. This reduction may be applicable to other areas of deep learning, such as generative adversarial networks, where encoding of the training set is crucial to generative performance.

[1] https://github.com/noahfl/sdr-densenet-pytorch.

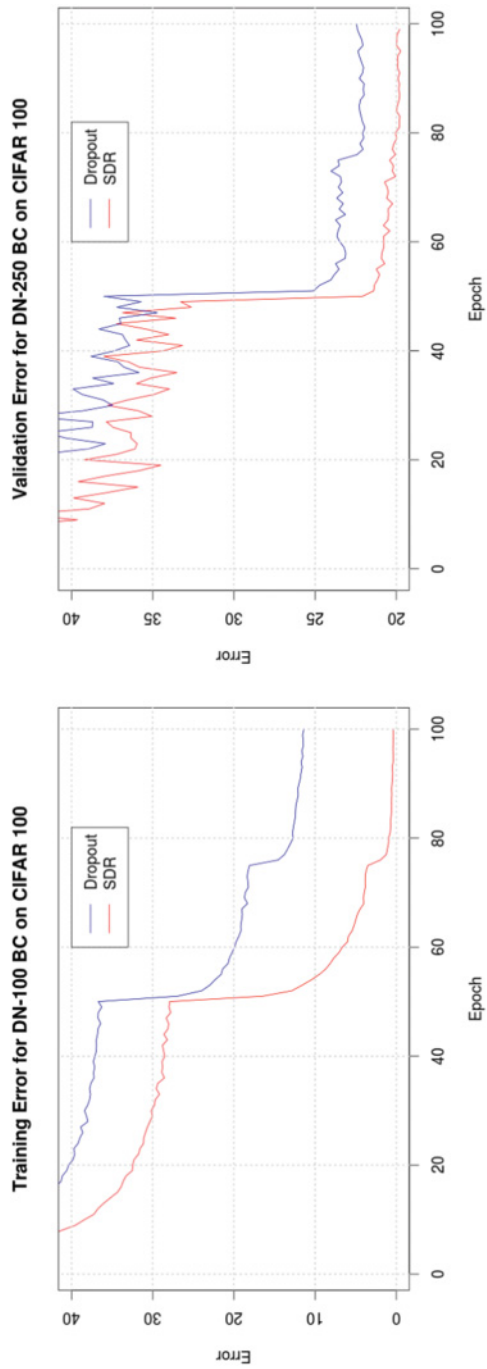

Figure 3: Error plots for DenseNet-40 training error and DenseNet-BC 250 validation error. The training error for DenseNet-40 is reduced by 83%, and the validation error for DenseNet-BC 250 is reduced by 13%.

Table 1: Top-1 Validation Error at End of Training of DenseNet-SDR Compared to DenseNet with Dropout ($p = 0.2$).

	Data Set		
	CIFAR-10	CIFAR-100	ImageNet
Dropout			
DenseNet-40	6.88	28.31	–
DenseNet-100 BC	6.02	26.18	–
DenseNet-100	5.11	23.00	–
DenseNet 250 BC	5.18	22.44	–
DenseNet 121 BC ($k = 32$)	–	–	27.26/9.01
SDR-Decay			
DenseNet-40	6.53	27.39	–
DenseNet-100 BC	**5.24 (−13.0%)**	23.45	–
DenseNet-100	4.87	**22.10 (−3.9%)**	–
DenseNet 250 BC	–	**19.79 (−11.8%)**	–
SDR-Dynamic			
DenseNet-40	**6.10 (−11.3%)**	**25.63 (−9.5%)**	–
DenseNet-100 BC	5.56	**23.19 (−11.4%)**	–
DenseNet-100	**4.82 (−5.7%)**	22.37	–
DenseNet 250 BC	**4.68 (−9.7%)**	21.45	–
DenseNet 121 BC ($k = 32$)	–	–	**25.58 (−6.2%)/7.82 (−13.2%)**

Notes: ImageNet results include both Top-1 and Top-5 error. Values in parentheses indicate percentage decrease in error compared to dropout. Dashes indicate incompatible model or data set pairs. Values in bold show SDR error improvements relative to known best benchmarks.

6.1 Speed Increases. Separate tests were conducted in order to find any increases in training speed to the validation error reached by Dropout at the end of 100 epochs. We titrated the number of epochs while maintaining the ratio of α drops (e.g., running for 40 epochs would yield α drops at epochs 20 and 30) in order to determine the minimum number of epochs required to reach Dropout's final validation error (see Figure 4). Results are shown in Table 3.

SDR allows for the same level of accuracy given by Dropout in as few as 40 training epochs to Dropout's 100. It should be noted that one SDR training epoch is approximately 3% slower than one dropout training epoch. As a result, net reduction in training time of approximately 58% is the best case.

7 Discussion

We have shown how a basic machine learning algorithm, Dropout, that implements stochastic search and helps prevent overfitting is a special case of an older algorithm, the stochastic delta rule, which is based on a gaussian

Table 2: Final Training Errors of Tested Models Compared to Dropout.

	Data Set		
	CIFAR-10	CIFAR-100	ImageNet
Dropout			
DenseNet-40	3.34	11.81	–
DenseNet-100 BC	2.24	11.36	–
DenseNet-100	0.51	2.52	–
DenseNet 250 BC	0.54	2.28	–
DenseNet 121 BC ($k = 32$)	–	–	29.61
SDR-Dynamic			
DenseNet-40	**0.22 (−93.4%)**	**2.72 (−77.0%)**	–
DenseNet-100 BC	**0.11 (−95.1%)**	**0.35 (−97.0%)**	–
DenseNet-100	**0.03 (−94.2%)**	**0.07 (−97.2%)**	–
DenseNet 250 BC	**0.04 (−92.6%)**	**0.14 (−93.9%)**	–
DenseNet 121 BC ($k = 32$)	–	–	**26.85 (−9.3%)**

Notes: Values in parentheses indicate percentage decrease in error compared to Dropout. Dashes indicate incompatible model or data set pairs. Values in bold show error improvements to known best benchmarks.

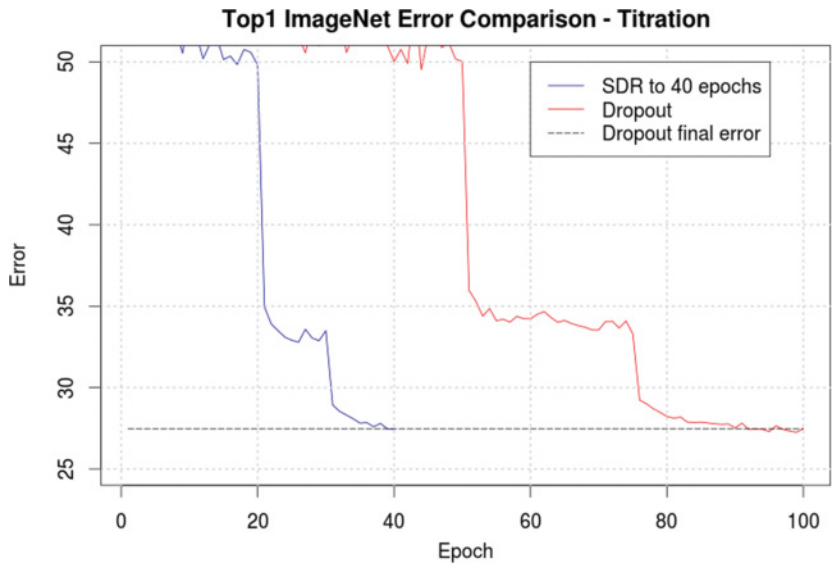

Figure 4: Error plots showing DN121-BC on ImageNet with Dropout run out to 100 epochs versus DN121-BC with SDR run to 40 epochs. The dotted line shows the final error Dropout reached.

Table 3: Number of Epochs Required for Each SDR Model to Reach the Respective Final Validation Error for Dropout.

	Data Set		
	CIFAR-10	CIFAR-100	ImageNet
Model			
DenseNet-40 with SDR	**50**	**45**	–
DenseNet-100 with SDR	–	**45**	–
DenseNet-BC 250 with SDR	–	**40**	–
DenseNet-BC 121 with SDR	–		**40**

Note: For tests on CIFAR, the default growh rate k is set to 12.

random sampling on weights and adaptable random variable parameters (in this case, a mean value, $\mu_{w_{ij}}$, and a standard deviation, $\sigma_{w_{ij}}$). We were also able to show how SDR outperforms Dropout in a state-of-the-art DL classifier on standard benchmarks, showing notable improvements in validation error by approximately. 5% to 10% or more in smaller networks and approximately 13% in larger networks and improvements in training error of 80% or more.

8 Related Work

We should note some of the algorithms to which SDR is unrelated but may seem similiar. Because weights are sampled from a distribution, each resulting final network shares weights (through the aggregate) over all forward passes of the network. This follows in that each weight mean value represents all possible forward pass values ($w_{ij}{}^*$) that will be sampled from that distribution. Consequently, weight updates encode the last forward pass in the mean and variance of the weight distribution, similiar to K-means algorithms; the next sampled forward pass is therefore a function of the new mean and standard deviation, which causes the new sampled weight vector to be biased toward the new weight distribution and its trajectory in weight space. The standard deviation updates are based on the gradient error and therefore increase or decrease as a function of prediction error. As the prediction error drops, weight distribution standard deviations begin to collapse to zero, causing the weight sampling to converge to a fixed-point, deterministic forward pass.

First, SDR should not be confused with variational Baysian estimation methods. Although variational methods model parameters as random variables, they are used for approximating various intractable integrals that result from exact forms of parameters' estimates. The random variables in this case implement a search process for network configurations. We are also not regularizing each weight as in "weight uncertainty in neural networks" (Blundell, Cornebise, Kavukcuoglu, & Wierstra, 2015), though they

focus on weight uncertainty and derive a regularizer that minimizes the variational free energy. Nor is this a Bayesian neural network in the sense of Gal and Ghahramani (2016) that represents weight uncertainty as a gaussian random variable and then uses KL divergence to approximate theoretically proper Bayes' updates to weights. Finally, we are not removing weights as in DropConnect (Wan, Zeiler, Zhang, LeCun, & Fergus, 2013); in fact, we never remove weights. In SDR, the weight is expanded to be a random variable with centrality and dispersion. Consequently, these two parameters are used to tune the annealing of the variance term toward zero, causing the network to collapse to the fixed mean network (SDR-decay). This was analyzed by a number of authors at the time of the original publication, showing, due to the annealing per weight, that SDR with sufficient samples tends to converge to a Bayes-optimal network (it finds parameters the minimize Bayes' error). Baldi and Sadowski (2014) and Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2012) in their analysis of Dropout point out that SDR is in fact a precursor to Dropout, but in contrast, it is injecting noise locally per weight in the network and then slowly removing it (annealing) based on local gradient values. Finally, Graves (2011) rediscovered SDR (-dynamic) and provided some theoretical motivation for the update rules, but pointed out in the end that although SDR-dynamic did not overfit, these networks were slower and did not work well with convolutional networks. In this case, though, we do show improvements with convolutional networks, possibly because of different initialization and annealing schedules, which Graves (2011) did not use. Also, recently, Fortunato et al. (2018) used SDR-dynamic for application of reinforcement learning on video games, reporting 48% improvement with SDR-dynamic over state-of-the-art search methods.

In addition to large improvements in the encoding of training data, SDR better generalizes to validation data. It is straightforward to implement and can be written in approximately 50 lines of code. All that is required is access to the gradients and the weights. SDR can be inserted into virtually any training implementation with this access. It thus opens up a novel set of directions for deep learning search methods that 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

Baldi, P., & Sadowski, P. J. (2014). The Dropout learning algorithm. *Artificial Intelligence*, *210*, 78–122.

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Understanding Dropout. *Advances in neural information processing systems*, *26*, 2814–2822. Red Hook, NY: Curran.

Burns, B. D. (1968). *The uncertain nervous system*. London: Edward Arnold Publications.

Dauphin, Y. N., Pascanu, R., Culcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *27*, 2933–2941. Red Hook, NY: Curran.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Piscataway, NJ: IEEE.

Faisal, A., Selen, L., & Wolpert, D. (2008). Noise in the nervous system. *Nature Reviews Neuroscience*, *9*, 292–303.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., . . . Legg, S. (2018). Noisy networks for exploration. In *Proceedings of the International Conference on Learning Representations*.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning*, *48* (pp. 1050–1059).

Glorot, X., & Benjio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 249–256).

Graves, A. (2011). Practical variational inference for neural networks. In Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *24* (pp. 2348–2356). Red Hook, NY: Curran.

Hanson, S. J. (1990). A stochastic version of the delta rule. *Physica D*, *42*, 265–272.

Hanson, S. J. (1992). SDR variations. Unpublished manuscript.

Hanson, S. J., & Pratt, L. Y. (1989). Comparing biases for minimal network construction with back-propagation. In D. Touretzky (Ed.), *Advances in neural information and processing systems*, *2* (pp. 177–185). San Mateo, CA: Morgan Kaufmann.

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313* (5786), 504–507.

Huang G., Liu, Z. Weinberger, K., & Maaten, L. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Piscataway, NJ: IEEE.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., . . . Chen, Z. (2019). GPipe: Efficient training of giant neural networks using pipeline parallelism. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Aleché-Bue, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*, *32* (pp. 103–112). Red Hook, NY: Curran.

Kawaguchi, K. (2016). Deep learning without poor local minima. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*, *29* (pp. 586–594). Red Hook, NY: Curran.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (technical report), https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*. Palo Alto, CA: AAAI.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arxiv:http://arXiv.org/abs/1207.0580.

Tan, M., & Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, *97* (pp. 6105–6114).

Veit, A. (2017). *A PyTorch implementation for densely connected convolutional networks (DenseNets)*. GitHub repository, https://github.com/andreasveit/densenet-pytorch/

Wan, L., Zeiler, M., Zhang, S., LeCun, Y., & Fergus, R. (2013). Regularization of neural networks using DropConnect. In *Proceedings of the 30th International Conference on Machine Learning*.

Received October 4, 2019; accepted January 9, 2020.