Fast Learning with Noise in Deep Neural Nets

Zhiyun Lu*
U. of Southern California
Los Angeles, CA 90089
zhiyunlu@usc.edu

Zi Wang*
Massachusetts Institute of Technology
Cambridge, MA 02139
ziwang.thu@gmail.com

Fei Sha
U. of Southern California
Los Angeles, CA 90089
feisha@usc.edu

Abstract

Dropout has been raised as an effective and simple trick [1] to combat overfitting in deep neural nets. The idea is to randomly mask out input and internal units during training. Despite its usefulness, there has been very little and scattered understanding on injecting noise to deep learning architectures' *internal* units. In this paper, we study the effect of dropout on both input and hidden layers in deep neural nets via explicit formulation of an equivalent marginalization regularizer. We show that training with regularizer from marginalized noise in deep neural nets doesn't loose much performance compared to dropout, yet in significantly shorter amount of training time and noticeably less sensitivity to hyperparameter tuning, which are main practical concerns of dropout.

1 Introduction and Related Work

In the last few years, deep neural nets (DNN) has been very successfully applied to problems in a wide range of application areas. Dropout, one of its recent tricks, which masks the input and internal (hidden) units randomly during training, has drawn considerable amount of interest from researchers [1]. The dropout technique has reliably and significantly improved many already competitive deep learning systems.

The benefits of artificially injecting noise, where dropout is just a special form, to learning has long been noted. Many research works [2, 3] have converged to the belief that properly introducing artificial noise benefits learning due to the regularization effect. Bishop elegantly demonstrated that the effect of noise is equivalent to adding a Tikhonov regularizer to the original loss function [2]. His analysis and most later-on work were largely based on marginalizing noise and use the second-order Taylor expansion around the unperturbed model to approximate the expected loss function. Following this thread, people have tried to analyze and understand why and how dropout works effectively and efficiently [4, 5, 6, 7]. For example, [6] applied to log-linear models and identified the resulting regularizer being adaptive to the input statistics. [7] applied to denoising autoencoders, whose analysis led to a regularizer explicitly penalizing the magnitude of the network output's Jacobian. The analysis in this paper follows similar steps.

Despite these progresses, however, there has been very little and scattered understanding on injecting noise to deep learning architectures' *internal* units. There are many open questions. For example, will such perturbation also introduce a form of regularization? If so, how will that regularization interplay with the regularization effect induced by the perturbation to the input units?

This paper has set out to take a few necessary steps in addressing those questions. With the standard perturbation analysis up to the second-order, we derive novel regularizers corresponding to applying

*equal contribution

noise to either the inputs or the hidden layers of deep networks. Empirically, we show that training deep networks with such regularizers does not loose much performance as training with dropout, yet in significantly shorter amount of training time and noticeably less sensitivity to hyperparameter tuning. Both are important to the practical utility of applying deep learning and dropout.

2 Approach

We start by describing the learning problem setting and introducing necessary notations. We then describe our approach of understanding the effect of incorporating noise into learning. We will show that adding noise is equivalent to adding a regularizer to the loss function. While our approach is generally applicable to many different models with various noise injection settings, we mainly discuss in detail the deep learning models, of sigmoid transfer function, with the dropout noise [1].

Our approach is based on marginalizing the noise, with appropriate approximation when adding noise on lower layers to reduce the computation complexity.

2.1 Problem setting

We focus on a (deep) neural network for K-way classification, with input $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$, and output $\boldsymbol{o} \in [0,1]^{\mathsf{K}}$. The output layer is softmax transformation with its input denoted as $\boldsymbol{a} \in \mathbb{R}^{\mathsf{K}}$, namely $o_k \propto \exp\{a_k\}$, such that o_k is interpreted as the posterior probability of $\boldsymbol{x}$ assigned to class k.

Training dataset is given by $\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{\mathsf{N}}$, where $\boldsymbol{y}_n \in [0,1]^{\mathsf{K}}$ and $\sum_k y_{nk} = 1$, our goal is to learn a nonlinear mapping $f : \boldsymbol{x} \to \boldsymbol{o}$. We use loss function $\ell(\boldsymbol{o}_n, \boldsymbol{y}_n)$ where $\boldsymbol{o}_n = f(\boldsymbol{x}_n)$ to measure the difference between the target and the actual outputs, with the assumption that both ℓ and f are smooth. As a concrete example, for multi-way classification tasks, we use the cross-entropy loss function $\ell(\boldsymbol{o}_n, \boldsymbol{y}_n) = -\sum_{k=1}^{\mathsf{K}} y_{nk} \log o_{nk}$. The mapping f, parameterized by the network parameters, is selected to minimize the empirical risk $\ell_{\mathsf{N}} = \frac{1}{\mathsf{N}} \sum_n \ell(\boldsymbol{o}_n, \boldsymbol{y}_n)$.

Injecting noise We are interested in understanding how the learning can be positively affected by adding noise. Noise can be added to the inputs, where we consider the following noise model

$$\tilde{\boldsymbol{x}} \sim p(\tilde{\boldsymbol{x}}|\boldsymbol{x}) \qquad \text{with} \quad \mathbb{E}\left[\tilde{\boldsymbol{x}}\right] = \bar{\boldsymbol{x}}, \text{and} \quad \mathbb{VAR}\left[\tilde{\boldsymbol{x}}\right] = \boldsymbol{\Sigma}_{\boldsymbol{x}} \tag{1}$$

Likewise, for a hidden layer, let $\boldsymbol{z} \in \mathbb{R}^{\mathsf{H}}$ denote its output. The noise on hidden layer is given by

$$\tilde{\boldsymbol{z}} \sim p(\tilde{\boldsymbol{z}}|\boldsymbol{z}), \qquad \text{with} \quad \mathbb{E}\left[\tilde{\boldsymbol{z}}\right] = \bar{\boldsymbol{z}}, \text{and} \quad \mathbb{VAR}\left[\tilde{\boldsymbol{z}}\right] = \boldsymbol{\Sigma}_{\boldsymbol{z}} \tag{2}$$

We use superscripts such as $\boldsymbol{z}^{(1)}, \cdots, \boldsymbol{z}^{(L)}$ to differentiate different layers if needed. While injecting noise on input augments the original data with noisy copies of $(\tilde{\boldsymbol{x}}, y)$, hidden layer noise model corresponds to the recent technique of *dropout* [1, 8].

The main idea of our analysis is to use low order perturbations to study how the empirical risk ℓ_{N} fluctuates, due to the randomness now introduced to either the inputs or the hidden units. To avoid notation cluttering, we consider only one sample (thus, dropping the subscript n).

2.2 Noise-induced Regularization

In what follows, we analyze the perturbed loss function due to noise in inputs and noise in the hidden units separately. However, we will show that our analysis can unify these two cases and also extend naturally when we add noise to both inputs and hidden units.

Adding noise to inputs The perturbed loss function is given by $\tilde{\ell} = \ell(\tilde{\boldsymbol{o}}, \boldsymbol{y})$ with $\tilde{\boldsymbol{o}} = f(\tilde{\boldsymbol{x}})$. Since both the mapping and the loss function are smooth, we expand the perturbed loss function at $\boldsymbol{x}$, and retain the terms up to the 2nd order of the (random) change $\delta_{\boldsymbol{x}} = \tilde{\boldsymbol{x}} - \boldsymbol{x}$. It gives $\tilde{\ell} = \ell + \boldsymbol{J}^{\mathsf{T}}\delta_{\boldsymbol{x}} + \frac{1}{2}\operatorname{tr}\left\{\boldsymbol{H}\delta_{\boldsymbol{x}}\delta_{\boldsymbol{x}}^{\mathsf{T}}\right\} + o(\|\delta_{\boldsymbol{x}}\|_2^2)$, where $\boldsymbol{J}$ and $\boldsymbol{H}$ are the Jacobian and Hessian of ℓ with respect to $\boldsymbol{x}$ respectively. Taking expectation with the conditional distribution $p(\tilde{\boldsymbol{x}}|\boldsymbol{x})$, assuming the perturbation is unbiased (i.e., $\bar{\boldsymbol{x}} = \boldsymbol{x}$), we obtained the *marginalized* loss function,

$$\hat{\ell} \approx \ell + \frac{1}{2}\operatorname{tr}\left\{\boldsymbol{H}\boldsymbol{\Sigma}_{\boldsymbol{x}}\right\} \tag{3}$$

While the right-hand-side is reminiscent of a regularized loss function, there is no guaranteed that the extra term is bounded below. We study this term in more details in the following.

For the cross-entropy loss function eq. (2.1), we have,

$$\boldsymbol{H} = \sum_k o_k \left(\frac{\partial a_k}{\partial \boldsymbol{x}}\right)\left(\frac{\partial a_k}{\partial \boldsymbol{x}}\right)^{\mathrm{T}} - \left(\sum_k o_k \frac{\partial a_k}{\partial \boldsymbol{x}}\right)\left(\sum_k o_k \frac{\partial a_k}{\partial \boldsymbol{x}}\right)^{\mathrm{T}} + \sum_k (o_k - y_k)\frac{\partial^2 a_k}{\partial \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}} \tag{4}$$

The last term is indefinite, thus is not possible to be bounded in the most general case. For the time being, we discard this term altogether — a similar strategy has been used in [2].

The first two terms are analogous to the variation for Jacobian $\boldsymbol{J}_k = \partial a_k/\partial \boldsymbol{x}$ with respect to o_k, a normalized probability. Thus, we define a regularized loss function,

$$\hat{\ell} = \ell + \frac{1}{2}\operatorname{tr}\left\{\mathbb{VAR}\left[\boldsymbol{J}_k\right]\boldsymbol{\Sigma}_{\boldsymbol{x}}\right\} \tag{5}$$

which marginalizes over noisy injected into the learning. Note that the regularizer is always non-negative as it is the inner product of two positive semidefinite matrices.

Adding noise to hidden units The above procedure generalizes naturally to applying noise to intermediate hidden layers of deep neural networks. For the time being, we do not perturb the inputs and we perturb only a single hidden layer $\boldsymbol{z}$ according to the noise model eq. (2). Conceptually, this is equivalent to adding noise to the inputs of a smaller neural network whose inputs are $\boldsymbol{z}$.

Thus, analogous to eq. (5), the marginalized loss function is given by

$$\hat{\ell} = \ell + \frac{1}{2}\operatorname{tr}\left\{\mathbb{VAR}\left[\boldsymbol{J}_k\right]\boldsymbol{\Sigma}_{\boldsymbol{z}}\right\} \tag{6}$$

where $\boldsymbol{J}_k$ in this case is defined as $\partial a_k/\partial \boldsymbol{z}$.

Adding noise to both inputs and hidden layers The similarity between eq. (5) and eq. (6) also enables us to generalize to adding noise to a deep networks at all the layers. To illustrate this, consider a network with one hidden layer and we perturb both that layer and the input layer.

First by perturbing the hidden layer, we get marginalized loss function $\hat{\ell}_{\boldsymbol{z}}$ from eq. (6). Now treating $\hat{\ell}_{\boldsymbol{z}}$ as a new loss function, we perturb the input by eq. (3), where the Hessian $\boldsymbol{H}$ is the Hessian of $\hat{\ell}_{\boldsymbol{z}}$ with respect to input $\boldsymbol{x}$. With further ignorance on high-order derivatives reflecting the contribution of variance at the inputs to the hidden layer, we have

$$\hat{\ell} = \ell + \frac{1}{2}\operatorname{tr}\left\{\mathbb{VAR}\left[\frac{\partial a_k}{\partial \boldsymbol{z}}\right]\boldsymbol{\Sigma}_{\boldsymbol{z}}\right\} + \frac{1}{2}\operatorname{tr}\left\{\mathbb{VAR}\left[\frac{\partial a_k}{\partial \boldsymbol{x}}\right]\boldsymbol{\Sigma}_{\boldsymbol{x}}\right\} \tag{7}$$

This can be extended to multiple layers

$$\hat{\ell} = \ell + \sum_{l=0}^{\mathsf{L}} \frac{1}{2}\operatorname{tr}\left\{\mathbb{VAR}\left[\frac{\partial a_k}{\partial \boldsymbol{z}^{(l)}}\right]\boldsymbol{\Sigma}_{\boldsymbol{z}^{(l)}}\right\} \tag{8}$$

where we have used $\boldsymbol{z}^{(0)}$ to represent $\boldsymbol{x}$ and $\boldsymbol{\Sigma}_{\boldsymbol{z}^{(l)}}$ is the covariance matrix for the noise model for the l-th layer.

Further approximation To reduce computational complexity and enable efficient back propagation algorithm for the regularizer, we further approximate the regularizer in Eqn. 5 to

$$\frac{1}{2}\sum_{k,j}\left[o_k(\frac{\partial a_k}{\partial z_j})^2 - (o_k\frac{\partial a_k}{\partial z_j})^2\right]\operatorname{diag}(\boldsymbol{\Sigma}_{\boldsymbol{x}}) \odot \left(\frac{\partial z_j}{\partial \boldsymbol{x}}\right)^2 \tag{9}$$

where $\odot$ is the dot product operator. This approximation holds for any shallower (not the last) hidden layer's regularizer.

3 Experiments

We validate our theoretical analysis with empirical studies on MNIST (*mnist*). We are interested in comparing dropout (in a sampling fashion) with our derived regularizers, in terms of classification performance and training efficiency. Sampling denotes the training algorithm with dropout noise, while Exact and Approx represents training with regularizer in Eq. 8 and Eq. 9 respectively. For regularizer of adding noise on the last hidden layer only, there is no difference between Exact and Approx, where we use Regularizer instead.

3.1 Setup

mnist has 60,000 images in the original training set which we randomly sampled 10,000 as validation and the remaining 50,000 as training. We use the original 10,000 test images as test set. The input and output size are 784 and 10 respectively. We compare various methods by their classification error rates, and training efficiency. Classification error rates are reported on test dataset with the best performance model on validation. Average is taken if multiple models perform equally on validation.

Hyper-parameters are tuned as follows. Because the strength of our derived regularizer λ can be related to dropout retain rate q, the probability that a unit is *not* dropped, with $\lambda = \frac{1-q}{2q}$, we tune retain rate q altogether to range in $[0.05, 0.955]$ in both settings. Other optimization parameters for stochastic gradient descent are learning rate in $[0.1, 2]$, learning rate decay in $[0, 0.001]$, momentum in $[0, 0.95]$. Learning rate decay is defined as $learning\ rate = \frac{learning\ rate}{1+(decay\ rate*i)}$ for epoch i. Batch size is fixed as 100.

3.2 Results

3.2.1 Comparison on Classification Performance

Table 1 and 2 display *mnist* classification performance under various settings of one-hidden-layer network with 1024 hidden units, and 2-hidden layer DNN of size 784-500-1000-10 respectively. Both networks are pretrained as Restricted Boltzmann Machine using contrastive divergence [9]. As expected, Sampling improves accuracy compared to training without any noise (with or without ℓ_2 regularizer). Our regularizer, both Exact and Approx, performs about the same with Sampling for 1-hidden layer network, and doesn't loose much in 2-hidden layer setting. This confirmed that the approximation we did in Section 2 is reasonable and works well in practice.

method	Without noise		Add noise to input			Add noise to hidden		Add noise to both		
	w/o reg	ℓ_2 reg	Sampling	Exact	Approx	Sampling	Regularizer	Sampling	Exact	Approx
Error rate (%)	1.40	1.37	1.28	1.26	1.28	1.31	1.26	1.28	1.34	1.25

Table 1: Classification error rates (%) on *mnist* with 1-hidden-layer network

method	Without noise	Add noise to both	
	ℓ_2 reg	Sampling	Approx
Error rate (%)	1.31	1.06	1.15

Table 2: Classification error rates (%) on *mnist* with 2-hidden layer deep belief nets

3.3 Comparison on Efficiency

Here we focus on convergence time and model tuning efforts to compare efficiency. We consider the practical scenario of how a neural network are trained and tuned when adding noise to input. In particular, we run our learning algorithms under different sets of configurations and choose the model with best performance on validation. To be more specific, we selected the top 175 models out of 200 (due to numerical issue), which are trained under 4 different hyper-parameter configurations (learning rate = $\{0.25, 0.5, 0.75, 1\}$, momentum = $\{0, 0.6, 0.7, 0.8, 0.9\}$, retain rate = $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ and whether pretrain or not).

Fig. 1 illustrates that the number of epochs (an epoch is defined as one pass through the whole training set) required (x-axis) to attain the optimal validation error rates (y-axis) for the 175 models using Sampling, Exact and Approx method. We can observe that using regularized objective functions tends to attain optimal error rates in a much smaller number of epochs. Moreover, the number of epochs required by sampling is not only larger but also more varying.

Table 3 provides more specific statistics of the scatter plot, with mean and standard deviation of convergence epochs and validation error rate. Besides, $T_{\text{CPU}}/\text{epoch}$ is the per epoch running time while T_{CPU} is the total convergence time on CPU. $T_{\text{GPU}}/\text{epoch}$ and T_{GPU} are results on GPU.[1] Mean and standard deviation of error rate quantitively assessed the sensitivity of each algorithms to hyper-parameters. Note that regularizer Approx is especially appealing: the optimal error rates are smaller and more concentrated. We can conclude from the experiment that compared to dropout, our regularized neural network converges faster and requires less tuning efforts to get a good result.

[1] the CPU is 64 AMD Opteron(tm) Processor 6380, with total memory of 512 GB and the GPU is a Nvidia Tesla K20m. We use Matlab's native support for GPU, under version Matlab 2014a.

	Sampling	Exact Reg	Approx Reg
#Epoch	665.91± 274.08	141.42±202.74	229.02±262.91
Error rate (%)	1.81±1.39	2.12±1.62	1.45±0.26
T_{CPU}/epoch (s)	28.64	386.98	53.93
T_{GPU}/epoch (s)	1.91	31.41	2.61
T_{CPU} ($\times 10^3$s)	19.1	54.7	12.4
T_{GPU} ($\times 10^3$s)	1.3	4.4	0.6

Table 3: Model Convergence Time

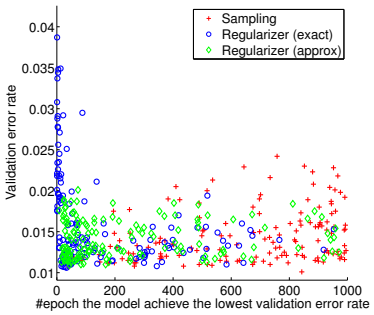

Figure 1: Convergence Epoch for Different Hyper-parameters

References

[1] 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.

[2] Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

[3] Salah Rifai, Xavier Glorot, Yoshua Bengio, and Pascal Vincent. Adding noise to the input of a model trained with a regularized objective. *arXiv preprint arXiv:1104.3250*, 2011.

[4] Pierre Baldi and Peter J Sadowski. Understanding dropout. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. 2013.

[5] Laurens Maaten, Minmin Chen, Stephen Tyree, and Kilian Q. Weinberger. Learning with marginalized corrupted features. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 410–418. JMLR Workshop and Conference Proceedings, 2013.

[6] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, pages 351–359, 2013.

[7] Minmin Chen, Kilian Weinberger, Fei Sha, and Yoshuo Bengio. Marginalized denoising autoencoders for nonlinear representation. In *International Conference on Machine Learning*, 2014.

[8] Nitish Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, 2013.

[9] Miguel A Carreira-Perpinan and Geoffrey E Hinton. On contrastive divergence learning. In *Proceedings of the tenth international workshop on artificial intelligence and statistics*, pages 33–40. Citeseer, 2005.