Fast dropout training

Sida I. Wang SIDAW@CS.STANFORD.EDU
Christopher D. Manning MANNING@STANFORD.EDU
Department of Computer Science, Stanford University, Stanford, CA 94305

Abstract

Preventing feature co-adaptation by encouraging independent contributions from different features often improves classification and regression performance. Dropout training (Hinton et al., 2012) does this by randomly dropping out (zeroing) hidden units and input features during training of neural networks. However, repeatedly sampling a random subset of input features makes training much slower. Based on an examination of the implied objective function of dropout training, we show how to do fast dropout training by sampling from or integrating a Gaussian approximation, instead of doing Monte Carlo optimization of this objective. This approximation, justified by the central limit theorem and empirical evidence, gives an order of magnitude speedup and more stability. We show how to do fast dropout training for classification, regression, and multilayer neural networks. Beyond dropout, our technique is extended to integrate out other types of noise and small image transformations.

1. Introduction

Recent work (Hinton et al., 2012) has shown that preventing feature co-adaptation by dropout training is a promising method for regularization. Applied to neural network training, the idea is to dropout (zero) randomly sampled hidden units and input features during each iteration of optimization. Dropout played an important role in the systems that won recent learning competitions such as ImageNet classification (Krizhevsky et al., 2012) and the Merck molecular activity challenge at `www.kaggle.com`, and improves performance on various tasks. Dropout can be considered another approach to regularization in addition to the widely used parameter shrinkage methods and model averaging. This process lowers the trust in a feature that is only helpful when other specific features are present, since any particular feature may be dropped out and cannot be depended on. Alternatively, the procedure can be seen as averaging over many neural networks with shared weights.

Other observations of harmful co-adaptation and ways to address them exist in the literature. Naive Bayes, by completely ignoring co-adaptation, performs better than discriminative methods when there is little data (Ng & Jordan, 2002), and continues to perform better on certain relatively large datasets (Wang & Manning, 2012). In (Sutton et al., 2006), it is observed that training involves trade-offs among weights, where the presence of highly indicative features can cause other useful but weaker features to undertrain. They propose feature bagging: training different models on subsets of features that are later combined, an idea further pursued under the name logarithmic opinion pools by (Smith et al., 2005). Improved performance on Named Entity Recognition and Part-of-Speech Tagging was demonstrated.

While the effectiveness of these methods in preventing feature co-adaptation has been demonstrated, actually sampling, or training multiple models, make training slower. Moreover, with a dropout rate of p, the proportion of data still not seen after n passes is p^n (e.g., 5 passes of the data are required to see 95% of it at $p = 0.5$). If the data is not highly redundant, and if we make the relevant data only partially observable at random, then the task becomes even harder, and training efficiency may reduce further.

In this paper, we look at how to achieve the benefit of dropout training without actually sampling, thereby using all the data efficiently. The approach uses a Gaussian approximation that is justified by the central limit theorem and empirical evidence. We show the validity of this approximation and how it can provide an order of magnitude speed-up at training time,

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

while also giving more stability. Fast dropout fits into the general framework of integrating out noise added to the training data (Matsuoka, 1992; Bishop, 1995). See (van der Maaten et al., 2013) for an alternative approach to integrating out noise and a survey of related work from that angle. Their approach is exact for loss functions decomposable by the moment generating function of the independent noise such as the exponential loss and squared error loss. Our approach does not require independence: it can integrate out small transformations that an image classifier should be invariant to. We begin with logistic regression for simplicity, then extend the idea to other loss functions, other noise, and neural networks. Code is provided at the author's website.

2. Fast approximations to dropout

2.1. The implied objective function

We illustrate the idea with logistic regression (LR) given training vector x, and label $y \in \{0,1\}$. To train LR with dropout on data with dimension m, first sample $z_i \sim \text{Bernoulli}(p_i)$ for $i = 1...m$. Here p_i is the probability of not dropping out input x_i. After sampling $z = \{z_i\}_{i=1...m}$ we can compute the stochastic gradient descent (sgd) update as follows:

$$\Delta w = (y - \sigma(w^T D_z x)) D_z x$$

where $D_z = \text{diag}(z) \in \mathbb{R}^{m \times m}$, and $\sigma(x) = 1/(1 + e^{-x})$ is the logistic function.

This update rule, applied over the training data for multiple passes, can be seen as a Monte Carlo approximation to the following gradient:

$$\Delta \bar{w} = E_{z;z_i \sim \text{Bernoulli}(p_i)}[(y - \sigma(w^T D_z x)) D_z x] \quad (1)$$

The objective function with the above gradient is the expected conditional log-likelihood of the label given the data with dropped out dimensions indicated by z, for $y \sim \text{Bernoulli}(\sigma(w^T D_z x)))$. This is the implied objective function for dropout training:

$$\begin{aligned} L(w) &= E_z[\log(p(y|D_z x; w)] \quad (2) \\ &= E_z[y \log(\sigma(w^T D_z x)) + (1-y)\log(1 - \sigma(w^T D_z x))] \end{aligned}$$

Since we are just taking an expectation, we still have a convex optimization problem provided that the negative log-likelihood is convex.

Evaluating the expectation in (1) naively by summing over all possible z has complexity $O(2^m m)$. Rather than directly computing the expectation with respect to z, we propose a variable transformation that allows us to approximately compute the expectation with respect to a simple random variable $Y \in \mathbb{R}$, instead of $z \in \{0,1\}^m$. In the next subsection, we describe an efficient $O(m)$ approximation that is accurate for machine learning applications where $w_i x_i$ usually come from a unimodal or bounded distribution.

2.2. The Gaussian approximation

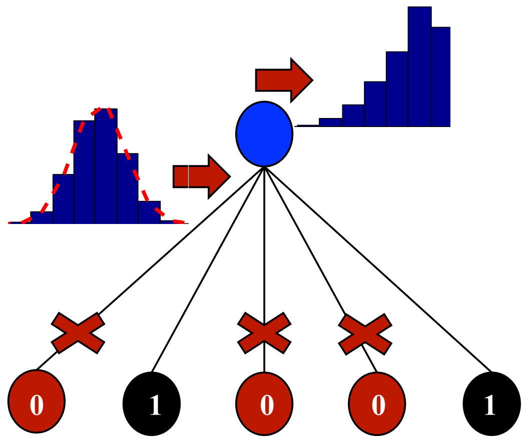

Figure 1. Illustration of the fast dropout idea: The numbers at the bottom are the dropout indicator variables $z_1...z_5$. As z is repeatedly sampled, the resulting inputs to the top unit are close to being normally distributed.

We make the observation that evaluating the objective function $L(w)$ involves taking the expectation with respect to the variable $Y(z) = w^T D_z x = \sum_i^m w_i x_i z_i$, a weighted sum of Bernoulli random variables. For most machine learning problems, $\{w_i\}$ typically forms a unimodal distribution centered at 0, $\{x_i\}$ is either unimodal or in a fixed interval. In this case, Y can be well approximated by a normal distribution even for relatively low dimensional data with $m = 10$. More technically, the Lyapunov condition is generally satisfied for a weighted sum of Bernoulli random variables of the form Y that are weighted by real data (Lehmann, 1998). Then, Lyapunov's central limit theorem states that $Y(z)$ tends to a normal distribution as $m \to \infty$ (see figure 1). We empirically verify that the approximation is good for typical datasets of moderate dimensions, except when a couple of dimensions dominate all others (see figure 3). Finally, let S be the approximating Gaussian ($Y \xrightarrow{d} S$)

$$S = E_z[Y(z)] + \sqrt{\text{Var}[Y(z)]}\epsilon = \mu_S + \sigma_S \epsilon \quad (3)$$

where $\epsilon \sim \mathcal{N}(0,1)$, $E_z[Y(z)] = \sum_{i=1}^m p_i w_i x_i$, and $\text{Var}[Y(z)] = \sum_{i=1}^m p_i(1-p_i)(w_i x_i)^2$.

In the following subsections, based on the Gaussian assumption above, we present several approximations at different tradeoff points between speed and accuracy. In the end, we present experimental results showing that there is little to no performance loss when using the faster, less accurate approximations.

2.3. Gradient computation by sampling from the Gaussian

Given good convergence, we note that drawing samples of the approximating Gaussian S of $Y(z)$, a constant time operation, is much cheaper than drawing samples of $Y(z)$ directly, which takes $O(m)$. This effect is very significant for high dimensional datasets. So without doing much, we can already approximate the objective function (2) m times faster by sampling from S instead of $Y(z)$. Empirically, this approximation is within the variance of the direct MC approximation of (2) by taking 200 samples of z.

Approximating the gradient introduces a complication when using samples from the Gaussian. The gradient (1) involves not only $Y(z) \xrightarrow{d} S$, but also $D_z x$ directly:

$$\nabla L(w) = E_z[(y - \sigma(Y(z)))D_z x] \tag{4}$$

Let $f(Y(z)) = y - \sigma(Y(z))$ and let $g(z) = D_z x$. Naively approximating $E_z[f(Y(z))g(z)]$ by either $E_S[f(S)]E_z[g(z)]$, or worse, by $f(E_s[S])E_z[g(z)]$ works poorly in terms of both approximation error and final performance. Note that $g(z)$ is a linear function and therefore $E_z[g(z)] = g(E_z[z]) = \text{diag}(p)x$. A good way to approximate (4) is by analytically taking the expectation with respect to z_i and then using a linear approximation to the conditional expectation. More precisely, consider dimension i of the gradient:

$$\begin{aligned}
\frac{\partial L(w)}{\partial w_i} &= E_z[f(Y(z))x_i z_i] \\
&= \sum_{z_i \in \{0,1\}} p(z_i) z_i x_i E_{z_{-i}|z_i}[f(Y(z))] \\
&= p(z_i = 1) x_i E_{z_{-i}|z_i=1}[f(Y(z))] \qquad (5)\\
&\approx p_i x_i \Bigg(E_{S\sim\mathcal{N}(\mu_S,\sigma_S^2)}[f(S)] \\
&\qquad + \Delta\mu_i \frac{\partial E_{S\sim\mathcal{N}(\mu,\sigma_S^2)}[f(S)]}{\partial \mu}\bigg|_{\mu=\mu_S} \\
&\qquad + \Delta\sigma_i^2 \frac{\partial E_{S\sim\mathcal{N}(\mu_S,\sigma^2)}[f(S)]}{\partial \sigma^2}\bigg|_{\sigma^2=\sigma_S^2} \Bigg) \\
&= p_i x_i(\alpha(\mu_S, \sigma_S^2) + \Delta\mu_i \beta(\mu_S, \sigma_S^2) + \Delta\sigma_i^2 \gamma(\mu_S, \sigma_S^2))
\end{aligned}$$

where z_{-i} is the collection of all other zs except z_i, μ_S, σ_S is defined in (3), $\Delta\mu_i = (1 - p_i)x_i w_i$, $\Delta\sigma_i^2 = -p_i(1-p_i)x_i^2 w_i^2$ are the changes in μ_S, σ_S^2 due to conditioning on z_i. Note that the partial derivatives as well as $E_{S\sim\mathcal{N}(\mu_S,\sigma_S^2)}[f(S)]$ only need to be computed once per training case, since they are independent of i.

α, β, γ can be computed by drawing K samples from S, taking time $O(K)$ (whereas K samples of $Y(z)$ take time $O(mK)$). Concretely,

$$\alpha(\mu, \sigma^2) = y - E_{S\sim\mathcal{N}(0,1)}\left[\frac{1}{1 + e^{-\mu - \sigma_S S}}\right]$$

$\beta(\mu, \sigma^2) = \frac{\partial \alpha(\mu,\sigma^2)}{\partial \mu}$, and $\gamma(\mu, \sigma^2) = \frac{\partial \alpha(\mu,\sigma^2)}{\partial \sigma^2}$ can be computed by differentiating inside the expectation.

One can combine (5) and what we do in (7) below to obtain a more accurate yet relatively cheap approximation to the derivative. However, in practice, using only β approximates the derivative to within the variance of successive MC computations of the objective L (see figure 4). Empirically, this is 2–30 times faster compared to MC dropout (see figure 5 and table 1).

At a slightly higher loss in accuracy, we can get rid of z completely by re-parameterizing the problem in μ_s and σ_s and taking derivatives with respect to them instead of approximating the derivative directly. So the objective function (2) becomes

$$L(w) \approx E_{S\sim\mathcal{N}(\mu_S,\sigma_S)}[y \log(\sigma(S)) + (1-y)\log(1-\sigma(S))] \tag{6}$$

2.4. A closed-form approximation

In the binary classification case, we can avoid sampling by tabulating α, β, γ, and their partial derivatives (they are just functions of 2 arguments). Interestingly, an accurate closed-from approximation is also possible by using the Gaussian cumulative distribution function $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-t^2/2}dt$ to approximate the logistic function. It can be shown by parameter differentiation with respect to μ and then integrating with respect to μ that

$$\int_{-\infty}^{\infty} \Phi(\lambda x)\mathcal{N}(x|\mu, s)dx = \Phi\left(\frac{\mu}{\sqrt{\lambda^{-2} + s^2}}\right)$$

Substituting in $\sigma(x) \approx \Phi(\sqrt{\pi/8}x)$, we get

$$\int_{-\infty}^{\infty} \sigma(x)\mathcal{N}(x|\mu, s^2)dx \approx \sigma\left(\frac{\mu}{\sqrt{1 + \pi s^2/8}}\right) \tag{7}$$

This is an approximation that is used for Bayesian prediction when the posterior is approximated by a Gaussian (MacKay, 1992). As we now have a closed-form approximation of α, one can also obtain expressions for β and γ by differentiating.

Furthermore, by substituting $x = \mu + st$, differentiating with respect to μ, and (7), we can even approximate the objective function (6) in a closed-form:

$$E_{X\sim\mathcal{N}(\mu,s^2)}[\log(\sigma(X))] = \int_{-\infty}^{\infty} \log(\sigma(x))\mathcal{N}(x|\mu,s^2)dx$$
$$\approx \sqrt{1+\pi s^2/8}\log\sigma\Big(\frac{\mu}{\sqrt{1+\pi s^2/8}}\Big) \quad (8)$$

The actual objective as defined in (2) can be obtained from the above by observing that $1-\sigma(x)=\sigma(-x)$. The gradient and Hessian with respect to w can be found by analytically differentiating.

3. Generalizations

3.1. Least squares regression

In contrast to all the approximations so far, dropout training of regression with squared error loss can be computed exactly. Let y be the true label and $\hat{Y}=\sum_i w_i x_i z_i$ be the predicted label with $\mu = E\hat{Y} = p\sum_{i=1}^m w_i x_i$ and $s^2 = \operatorname{Var}\hat{Y} = p(1-p)\sum_{i=1}^m w_i^2 x_i^2$

By the bias-variance decomposition, the expected squared error loss is

$$E_{\hat{Y}\sim\mathcal{N}(\mu,s^2)}[(\hat{Y}-y)^2] = \int_{-\infty}^{\infty}(\hat{y}-y)^2\mathcal{N}(\hat{y}|\mu,s^2)d\hat{y}$$
$$= (\mu-y)^2+s^2 \quad (9)$$

Since (9) is completely determined by the mean and variance of $\hat{Y}$, it does not matter which distribution $\hat{Y}$ comes from as long as μ and s^2 are matched. As a result, (9) is also the exact loss function of the original dropout objective if we summed over z_i instead. So over the whole dataset of size n, dropout regression has the following equivalent objective:

$$L(w) = \frac{1}{n}\sum_{j=0}^{n}(\hat{y}(w,x^{(j)})-y)^2 + \lambda\sum_{i=1}^{m}c_i w_i^2$$

This is a form of L_2 regularization depending on $c_i = 1/n\sum_{j=1}^n x_i^{(j)2}$ so that weights of larger features are regularized more strongly.

Alternatively, let $X\in\mathbb{R}^{n\times m}$ be the design matrix, then the normal equations for dropout training and ridge regression are, respectively,

$$w = (X^TX+\lambda\operatorname{diag}(X^TX))^{-1}X^Ty$$
$$w = (X^TX+\lambda I)^{-1}X^Ty \quad (10)$$

where $\operatorname{diag}(A)$ represents the diagonal matrix with the same diagonal as A. The diagonal of X^TX is stronger by a multiplicative factor $1+\lambda$ for dropout instead of the additive λI for L_2. The equivalent value for λ determined by dropout is $(1-p)/p$.

3.2. Hinge loss and the Maxout unit

Our apporach can be applied to the classical hinge loss and the recently proposed maxout network (Goodfellow et al., 2013). The structured SVM loss is

$$L(w) = \max_{\hat{y}\in\mathcal{Y}}\{\ell(y,\hat{y}) + (w_{\hat{y}}^Tx) - (w_y^Tx)\}.$$

where $\mathcal{Y}$ is the set of possible predictions and $\ell(y,y')$ is the loss incurred by predicting $\hat{y}$ when the true label is y. The maxout unit computes

$$h(x) = \max_j w_j^T x$$

Under the fast dropout approach, both of these reduce to the problem of computing the maximum of Gaussians $\max_i X_i$ for $X_i\sim\mathcal{N}(\mu(x,w_i),\sigma^2(x,w_i))$ not necessarily indepedent. Several approaches to this problem is presented in (Ross, 2010).

3.3. Softmax and general loss

Unfortunately, the best way to compute the cross-entropy loss for softmax seems to be sampling from the input Gaussian directly with $S\in\mathbb{R}^{|\mathcal{Y}|}$ where $\mathcal{Y}$ is the set of possible predictions.

$$L = E_{S\sim\mathcal{N}(\mu,\Sigma)}[\sum_{i=1}^{|\mathcal{Y}|} t_i\log(\operatorname{softmax}(S)_i)]$$
$$= E_{S'\sim\mathcal{N}(0,I)}[\sum_{i=1}^{|\mathcal{Y}|} t_i\log(\operatorname{softmax}(\mu+US')_i)]$$

where $\operatorname{softmax}(s)_i = e^{s_i}/\sum_{j=1}^{|\mathcal{Y}|}e^{s_j}$ and $\Sigma = UU^T$. The required partial derivatives can again be computed by differentiating inside the expectation. This is also the general way to do fast dropout training on output units that may be vector-valued functions of vectors.

3.4. Transformation invariance as noise

More image data can be generated by applying transformations like small translations, rotations, shearing etc. to the original training data. A transformation of magnitude ϵ can be approximated locally by its Lie derivative as $T_\alpha(x) = x+\epsilon L_{T,x}$ (Simard et al., 1996). For translation, rotation, shearing, we can generate more data by randomly sampling $\epsilon_i\sim\mathcal{N}(0,\sigma_i^2)$ and computing $X = x+\sum_i\epsilon_i L_i$. Notice that w^TX is again normally distributed and the techniques presented in this paper can be used to integrate out these transformations without actually generating the transformed data. Here we do not need the central limit theorem and the noise is not independent.

3.5. Other noise

Like the exact approach in (van der Maaten et al., 2013), the Gaussian approximation can be applied to other noise models (Poisson, Gaussian, etc). We just need to characterize the noise in terms of its mean and variance and rely on the central limit theorem.

4. Fast dropout for neural networks

Dropout training, as originally proposed, was intended for neural networks where hidden units are dropped out, instead of the data. Fast dropout is directly applicable to dropping out the final hidden layer of neural networks. In this section, we approximately extend our technique to deep neural networks and show how they apply to several popular types of hidden units. For the last layer of a neural network, any output unit outlined in section 3 can be used.

4.1. The hidden layers

Under dropout training, each hidden unit takes a random variable as input, and produces a random variable as output. When the number of hidden units is more than 10 or so, we may again approximate their inputs as Gaussians and characterize their outputs by the output means and variances. A complication is that the inputs to hidden units have a covariance as shown in figure 2.

Consider any hidden unit in dropout training. We may approximate its input as a Gaussian variable $X \sim \mathcal{N}(x|\mu, s^2)$, and let its output mean and variance be ν and τ^2. E.g., for the commonly used sigmoid unit

$$\nu = \int_{-\infty}^{\infty} \sigma(x)\mathcal{N}(x|\mu, s^2)dx \approx \sigma\left(\frac{\mu}{\sqrt{1+\pi s^2/8}}\right)$$

This integral can be evaluated exactly for the rectified linear unit $f(x) = \max(0, x)$. Let $r = \mu/s$, then

$$\nu = \int_{-\infty}^{\infty} f(x)\mathcal{N}(x|\mu, s^2)dx = \Phi(r)\mu + s\mathcal{N}(r|0, 1)$$

The rectified linear unit is a special case of the maxout unit, for which techniques in (Ross, 2010) can be used to compute its mean and variance.

With dropout training, each hidden unit also has an output variance. Sigmoid squared can be approximated by a translatedscaled version of the sigmoid:

$$\begin{aligned}\tau^2 &= \underset{X\sim\mathcal{N}(\mu,s^2)}{\mathrm{Var}}[\sigma(X)] = E[\sigma(X)^2] - E[\sigma(X)]^2 \\ &\approx E[\sigma(a(X-b))] - E[\sigma(X)]^2\end{aligned}$$

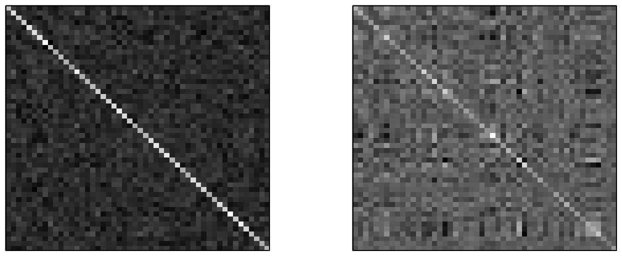
Figure 2. *MC dropout covariance matrices of the inputs of 50 random hidden units*: left: at random initialization; right: trained to convergence. The covariance is not completely diagonal once trained to convergence.

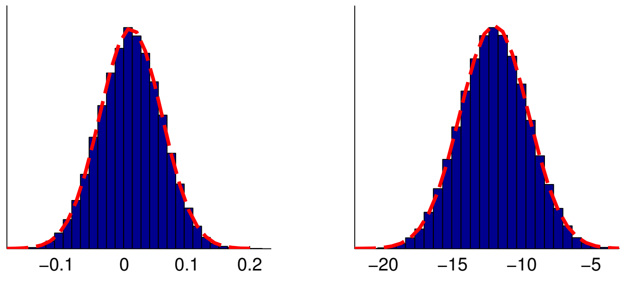

Figure 3. *Empirical input distribution of the input of a hidden unit*: left: random initialization; right: trained to convergence. We lose almost nothing here.

a,b can be found by matching the values and derivatives ($a = 4 - 2\sqrt{2}$ and $b = -\log(\sqrt{2} - 1)$).

4.2. Training with backpropagation

The resulting neural network can be trained by backpropagation with two sets of partial derivatives. In normal backpropagation, one only needs to keep $\frac{\partial L}{\partial \mu_i}$ for each hidden unit i with input μ_i. For fast dropout training, we need $\frac{\partial L}{\partial s_i^2}$ as well for input variance s_i^2. Where $\mu_i = p\sum_j w_{ij}\nu'_j$ and $s_i^2 = \sum_j p(1-p)\nu'^2_j w_{ij}^2 + p\tau'^2_j w_{ij}^2$ and ν'_j and τ'_j are the output mean and variance of the previous layer. In practice, the method still works well if we ignore the output variance τ, so the input variance to the next layer is generated by dropout alone.

5. Relation to Bayesian model selection

Once we make the Gaussian approximation, there is an alternative interpretation of where the variance comes from. In the dropout framework, the variance comes from the dropout variable z. Under the alternative interpretation where w is a random variable, we can view dropout training as maximizing a lower bound on the Bayesian marginal likelihood among a class of models M_μ indexed by $\mu \in \mathbb{R}^m$. Concretely, let $\mu_i = pw_i$, then the dropout objective

$$\begin{aligned} L(w) &= E_{z;z_i \sim \text{Bernoulli}(p_i)}[\log p(y|w^T D_z x)] \\ &\approx E_{Y \sim \mathcal{N}(E[w^T D_z x], \text{Var}[w^T D_z x])}[\log p(y|Y)] \\ &= E_{v:v_i \sim \mathcal{N}(\mu_i, \alpha\mu_i^2)}[\log p(y|v^T x)] \\ &\leq \log E_{v:v_i \sim \mathcal{N}(\mu_i, \alpha\mu_i^2)}[p(y|v^T x)] \\ &= \log(M_\mu) \end{aligned}$$

where $M_\mu = \int p(\mathcal{D}|v)p(v|\mu)dv$ is the Bayesian evidence. $p(v_i|\mu_i) = \mathcal{N}(v_i|\mu_i, \alpha\mu_i^2)$ and $p(y|v^T x) = \sigma(v^T x)^y (1 - \sigma(v^T x))^{1-y}$ is the logistic model. For dropout training, $\mu = w/p$ and $\alpha = (1-p)/p$.

Here the variance of v is tied to its magnitude, so a larger weight is only beneficial when it is robust to noise. While α can be determined by the dropout process, we are also free to choose α and we find empirically that using a slightly larger α than that determined by dropout often performs slightly better.

6. Experiments

6.1. Evaluating the assumptions and speed

For logistic regression (LR), figure 4 shows that the quality of the gradient approximation using Gaussian samples is comparable to the difference between different MC dropout runs with 200 samples. Figure 5 shows that, under identical settings, the Gaussian approximation is much faster than MC dropout, and has a very similar validation error profile. Both Gaussian dropout training and real dropout training reduce validation error rate by about 30% over plain LR when trained to convergence, without ever overfitting.

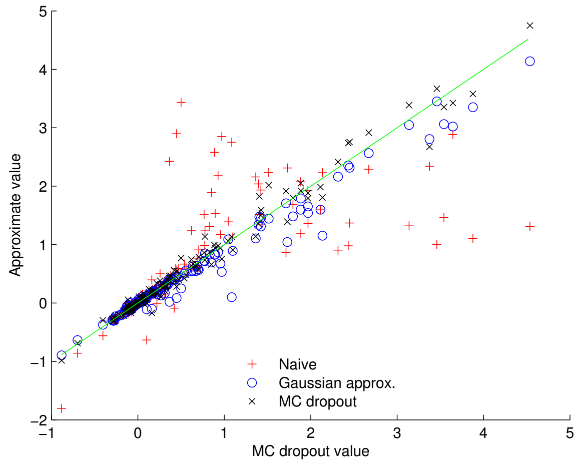

Figure 4. Scatterplot of various approximations (y-axis) vs. direct MC dropout for LR: Each point is a random dimension of the gradient, with its x-value computed from MC dropout with 200 samples of z, and its y-value computed by the method in the legend. MC dropout and Gaussian approximation used 200 samples. Naive is the approximation defined after (4), by assuming that $f(z)$ and $g(z)$ are independent. The green line is the reference $y = x$.

6.2. Experiments on document classification

We show the performance of fast dropout LR on several sentiment and topic document classification tasks, both accuracy and time taken, in the top half of table 1. Sampling from the Gaussian is generally around 10 times faster than MC dropout and performs comparably to NBSVM in (Wang & Manning, 2012), which is a method specifically engineered for document classification. Further speedup is achieved by directly optimizing the objective in (8) and that is only 30% slower than plain logistic regression. While each iteration of fast dropout is still slower than LR, fast dropout sometimes reaches a better validation performance in less time as seen in figure 5. Note that for the MPQA dataset where the average number of non-zero dimensions is $m \approx 4$, the Gaussian assumption is unjustifiable, but the derived method works empirically anyways. We compare to other papers in the bottom half of the table 1, using either a test/train split or N-fold cross validation, depending on what is the most standard for the dataset. With the right regularization parameters and bigram features, our plain LR baseline is itself quite strong relative to previous work.

6.3. Experiments on MNIST

Experimental results on MNIST using 2-hidden-layer neural networks are shown in table 2 and the validation error curves with a slight smaller net are shown in figure 6. Here is a case where the data is fairly redundant so that dropping out input features does not make the problem much harder and MC dropout on minibatches converges fairly quickly. We replicate the original experiment using the exact settings described in (Hinton et al., 2012) with a 20% dropout of the inputs, an exponentially decaying learning rate, a momentum schedule, and minibatch stochastic gradient descent. Under the learning schedule in the original experiment, no improvement resulted from doing fast dropout in the minibatch setting. In fact, each minibatch of fast dropout takes 1.5 times as much time as real dropout with 1 sample. However, the fast dropout objective is suitable for standard optimization technology, and we were able to train faster using L-BFGS where it converged in less than 100 epochs as opposed to over 500 epochs (see figure 6). 160 errors is the previous best result without pre-training or weight-sharing or enhancement of the training data.

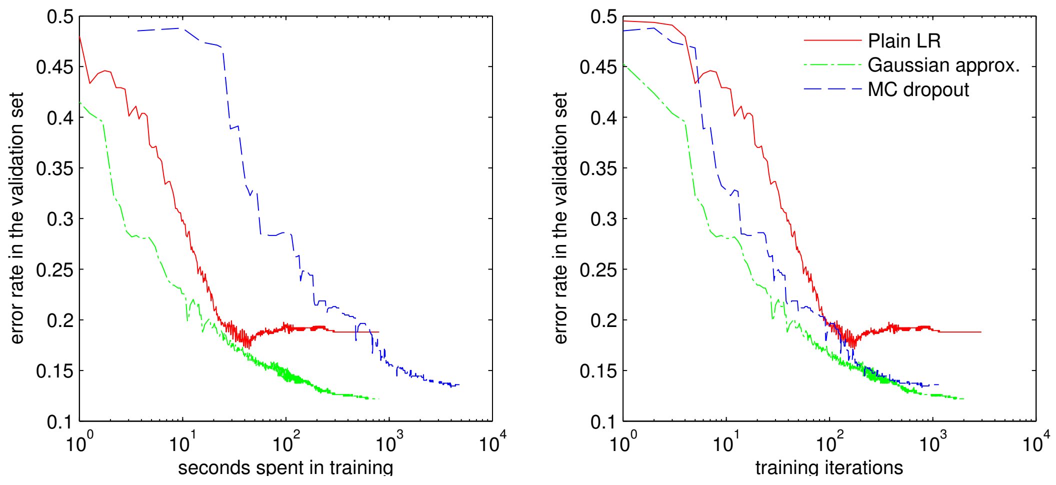

Figure 5. Validation errors vs. time spent in training (left), and number of iterations (right): trained using batch gradient descent on the 20-newsgroup subtask alt.atheism vs. religion.misc. 100 samples are used for both MC and Gaussian dropout. For MC dropout, z_i is sampled only for non-zero x_i.

Methods\ Datasets	MR-2k	IMDB	RTs	Subj	AthR	CR	MPQA	Average
Real (MC) dropout	89.8	91.2	79.2	93.3	86.7	82.0	86.0	86.88
training time	*6400*	*6800*	*2300*	*2000*	*130*	*580*	*420*	*2700*
Gaussian dropout	89.7	91.2	79.0	93.4	87.4	82.1	86.1	86.99
training time	*240*	*1070*	*360*	*320*	*6*	*90*	*180*	*320*
Fast (closed-form) dropout	89.5	91.1	79.1	93.6	86.5	81.9	86.3	86.87
training time	*120*	*420*	*130*	*130*	*3*	*28*	*35*	*120*
plain LR	88.2	89.5	77.2	91.3	83.6	80.4	84.6	84.97
training time	*140*	*310*	*81*	*68*	*3*	*17*	*22*	*92*
Previous results								
TreeCRF(Nakagawa et al., 2010)	-	-	77.3	-	-	81.4	86.1	-
Vect. Sent.(Maas et al., 2011)	88.9	88.9	-	88.1	-	-	-	-
RNN(Socher et al., 2011)	-	-	77.7	-	-	-	86.4	-
NBSVM(Wang & Manning, 2012)	89.4	91.2	79.4	93.2	87.9	81.8	86.3	87.03
$\lvert\{i : x_i > 0\}\rvert$	788	232	22	25	346	21	4	

Table 1. Results on document classification: 100 samples are used for both MC dropout and the Gaussian dropout. The last row shows the average number of non-sparse dimensions in the dataset. See Wang & Manning (2012) for more details on the datasets and experimental procedures.

Method	2NN	FD	+Var	+Tr	Real
Errors	170	124	110	85	105-120

Table 2. Test errors of neural networks on MNIST: 2NN: 2-hidden-layer neural net with 784-1200-1200-10. FD: fast dropout. +Var: more artificial variance by increasing α in FD. +Tr: integrating out translation, rotation and scaling described in 3.4. MC: real dropout.

6.4. The test time utility of fast dropout

For the case of real dropout, at test time, Hinton et al. (2012) propose using all features, with weights scaled by p. This weight scaling heuristic does not exactly match the training objective being optimized, but greatly speeds run time performance. If we are not concerned about run time, we can still apply dropout at test time.

In contrast, the test time procedure for fast dropout is exactly the same as the training time procedure. One shortcoming of fast dropout is that the implementation of training does become more complicated, mainly in the backpropagation stage, while the forward computation of the network function is still straightforward.

One compromise here is to use fast dropout at test time, even if we want to train with real dropout for

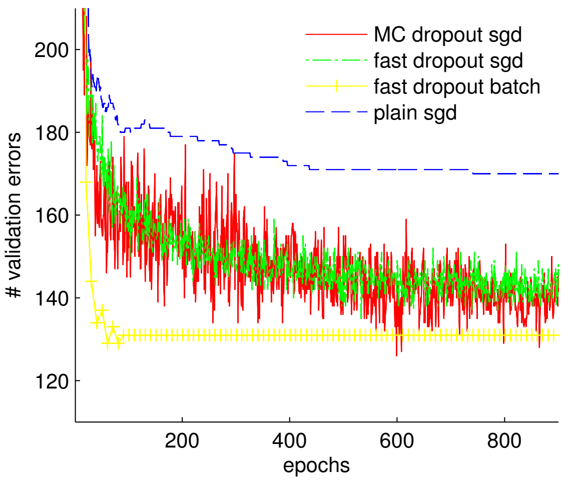

Figure 6. Validation errors vs. epochs: we used the exact SGD training schedule described in (Hinton et al., 2012) and a 784-800-800-10 2-hiden-layer neural network. This training schedule is presumably tuned for real dropout. fast dropout performs similarly to real dropout but with less variance. fast dropout batch: use batch L-BFGS and fast dropout, with validation error evaluated every 10 epochs.

simplicity. Table 3 compares several test time methods on neural networks trained for MNIST and CIFAR using real dropout. Multiple real dropout samples and fast dropout provide a small but noticeable improvement over weight scaling.

6.5. Other experiments

The normal equations (10) show the contrast between additive and multiplicative L_2 regularization. For linear regression, L_2 regularization outperformed dropout on 10 datasets from UCI that we tried.[1] Results on 5 of them are shown in table 4.

Classification results using neural networks on small UCI datasets are shown in table 5 where fast dropout does better than plain neural networks in most cases.

7. Conclusions

We presented a way of getting the benefits of dropout training without actually sampling, thereby speeding up the process by an order of magnitude. For high dimensional datasets (over a few hundred), each iteration of fast dropout is less than 2 times slower than normal training. We provided a deterministic and easy-to-compute objective function approximately equivalent to that of real dropout training. One can optimize this objective using standard optimization methods, whereas standard methods are of limited use in real dropout because we only have a noisy measurement of the gradient. Furthermore, since fast dropout is not losing any information in individual training cases from sampling, it is capable of doing more work in each iteration, often reaching the same validation set performance in a shorter time and in less iterations.

[1] http://archive.ics.uci.edu/ml/

	Full	Scale	D1	D10	D100	FD
MNIST number of errors						
Test	129	108	199	118	105	103
Train	4	1	36	1	1	1
Time(s)	10	10	13	110	1.1K	16
CIFAR-10 percent error						
Test	53	47	51	45	43	44
Train	42	35	42	33	32	33
Time(s)	17	23	25	230	2.2K	29

Table 3. Different test time methods on networks trained with real dropout: A 784-800-800-10 neural network is trained with real dropout on MNIST (3072-1000-1000-10 for CIFAR-10) and tested using: Full: use all weights without scaling; Scale: $w \leftarrow pw$; D(n): take n real dropout samples; FD: fast dropout.

Dataset	L_2 train	L_2 test	FD_{train}	FD_{test}
Autos	0.25	0.51	0.41	0.57
Cardio	109.24	117.87	140.93	188.91
House	23.57	21.02	65.44	56.26
Liver	9.01	9.94	9.69	9.88
Webgrph	0.17	0.20	0.19	0.21

Table 4. Linear regression using eq. (10). Dropout performs worse than L_2 regularization (recall that fast dropout is exact). While a digit is still easily recognizable when half of its dimensions are dropped out, dropout noise is excessive for the low dimensional regressors.

Classification accuracy				
Dataset	L_2 train	L_2 test	FD_{train}	FD_{test}
SmallM	100	87	99	**90**
USPS	100	95	98	**96**
Isolet	100	91	95	**93**
Hepatitis	100	94	99	**95**
Soybean	100	**91**	94	89

Table 5. Classification results on various datasets: we used a M-200-100-K neural network, and cross validated the parameters.

Acknowledgments

We thank Andrej Karpathy, Charlie Tang and Percy Liang for helpful discussions. Sida Wang was partially funded by a NSERC PGS-M scholarship and a Stanford School of Engineering fellowship.

References

Bishop, Christopher M. Training with noise is equivalent to Tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

Goodfellow, Ian J., Warde-Farley, David, Mirza, Mehdi, Courville, Aaron C., and Bengio, Yoshua. Maxout networks. *CoRR*, abs/1302.4389, 2013.

Hinton, Geoffrey E., Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. ImageNet classification with deep convolutional neural networks. In *Proceedings of NIPS*, 2012.

Lehmann, Erich L. *Elements of Large-Sample Theory.* Springer, 1998. ISBN 03873985956.

Maas, Andrew L., Daly, Raymond E., Pham, Peter T., Huang, Dan, Ng, Andrew Y., and Potts, Christopher. Learning word vectors for sentiment analysis. In *Proceedings of the ACL*, 2011.

MacKay, David J.C. The evidence framework applied to classification networks. *Neural Computation*, 5 (4):720–736, 1992.

Matsuoka, Kiyotoshi. Noise injection into inputs in back-propagation learning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):436–440, 1992.

Nakagawa, Tetsuji, Inui, Kentaro, and Kurohashi, Sadao. Dependency tree-based sentiment classification using CRFs with hidden variables. In *Proceedings of ACL:HLT*, 2010.

Ng, Andrew Y. and Jordan, Michael I. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proceedings of NIPS*, volume 2, pp. 841–848, 2002.

Ross, Andrew. Computing bounds on the expected maximum of correlated normal variables. *Methodology and Computing in Applied Probability*, 12:111–138, 2010. ISSN 1387-5841.

Simard, Patrice, LeCun, Yann, Denker, John, and Victorri, Bernard. Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*, pp. 239–27, 1996.

Smith, Andrew, Cohn, Trevor, and Osborne, Miles. Logarithmic opinion pools for conditional random fields. In *Proceedings of the ACL*, pp. 18–25, 2005.

Socher, Richard, Pennington, Jeffrey, Huang, Eric H., Ng, Andrew Y., and Manning, Christopher D. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of EMNLP*, 2011.

Sutton, Charles, Sindelar, Michael, and McCallum, Andrew. Reducing weight undertraining in structured discriminative learning. In *Proceedings of HLT-NAACL)*, 2006.

van der Maaten, Laurens, Chen, Minmin, Tyree, Stephen, and Weinberger, Kilian. Learning with marginalized corrupted features. In *Proceedings of ICML*, 2013.

Wang, Sida and Manning, Christopher. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the ACL*, pp. 90–94, 2012.