Regularization of Neural Networks using DropConnect

Li Wan WANLI@CS.NYU.EDU
Matthew Zeiler ZEILER@CS.NYU.EDU
Sixin Zhang ZSX@CS.NYU.EDU
Yann LeCun YANN@CS.NYU.EDU
Rob Fergus FERGUS@CS.NYU.EDU
Dept. of Computer Science, Courant Institute of Mathematical Science, New York University

Abstract

We introduce DropConnect, a generalization of Dropout (Hinton et al., 2012), for regularizing large fully-connected layers within neural networks. When training with Dropout, a randomly selected subset of activations are set to zero within each layer. DropConnect instead sets a randomly selected subset of *weights* within the network to zero. Each unit thus receives input from a random subset of units in the previous layer. We derive a bound on the generalization performance of both Dropout and DropConnect. We then evaluate DropConnect on a range of datasets, comparing to Dropout, and show state-of-the-art results on several image recognition benchmarks by aggregating multiple DropConnect-trained models.

1. Introduction

Neural network (NN) models are well suited to domains where large labeled datasets are available, since their capacity can easily be increased by adding more layers or more units in each layer. However, big networks with millions or billions of parameters can easily overfit even the largest of datasets. Correspondingly, a wide range of techniques for regularizing NNs have been developed. Adding an ℓ_2 penalty on the network weights is one simple but effective approach. Other forms of regularization include: Bayesian methods (Mackay, 1995), weight elimination (Weigend et al., 1991) and early stopping of training. In practice, using these techniques when training big networks gives superior test performance to smaller networks trained without regularization.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

Recently, Hinton *et al.* proposed a new form of regularization called Dropout (Hinton et al., 2012). For each training example, forward propagation involves randomly deleting half the activations in each layer. The error is then backpropagated only through the remaining activations. Extensive experiments show that this significantly reduces over-fitting and improves test performance. Although a full understanding of its mechanism is elusive, the intuition is that it prevents the network weights from collaborating with one another to memorize the training examples.

In this paper, we propose DropConnect which generalizes Dropout by randomly dropping the weights rather than the activations. Like Dropout, the technique is suitable for fully connected layers only. We compare and contrast the two methods on four different image datasets.

2. Motivation

To demonstrate our method we consider a fully connected layer of a neural network with input $v = [v_1, v_2, \ldots, v_n]^T$ and weight parameters W (of size $d \times n$). The output of this layer, $r = [r_1, r_2, \ldots, r_d]^T$ is computed as a matrix multiply between the input vector and the weight matrix followed by a non-linear activation function, a, (biases are included in W with a corresponding fixed input of 1 for simplicity):

$$r = a(u) = a(Wv) \tag{1}$$

2.1. Dropout

Dropout was proposed by (Hinton et al., 2012) as a form of regularization for fully connected neural network layers. Each element of a layer's output is kept with probability p, otherwise being set to 0 with probability $(1-p)$. Extensive experiments show that Dropout improves the network's generalization ability, giving improved test performance.

When Dropout is applied to the outputs of a fully con-

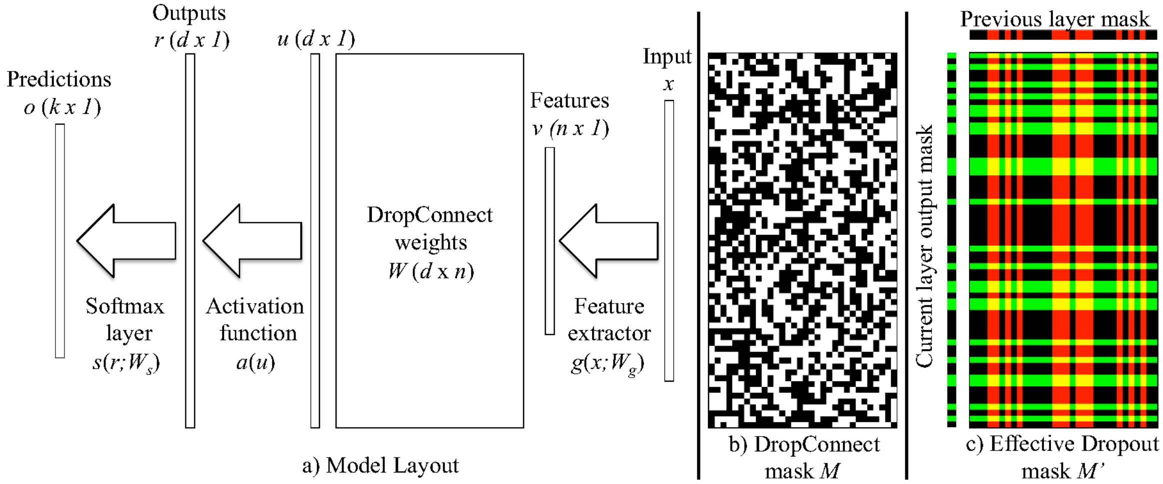

Figure 1. (a): An example model layout for a single DropConnect layer. After running feature extractor $g()$ on input x, a random instantiation of the mask M (e.g. (b)), masks out the weight matrix W. The masked weights are multiplied with this feature vector to produce u which is the input to an activation function a and a softmax layer s. For comparison, (c) shows an effective weight mask for elements that Dropout uses when applied to the previous layer's output (red columns) and this layer's output (green rows). Note the lack of structure in (b) compared to (c).

nected layer, we can write Eqn. 1 as:

$$r = m \star a(Wv) \tag{2}$$

where $\star$ denotes element wise product and m is a binary mask vector of size d with each element, j, drawn independently from $m_j \sim Bernoulli(p)$.

Many commonly used activation functions such as *tanh*, centered *sigmoid* and *relu* (Nair and Hinton, 2010), have the property that $a(0) = 0$. Thus, Eqn. 2 could be re-written as, $r = a(m \star Wv)$, where Dropout is applied at the inputs to the activation function.

2.2. DropConnect

DropConnect is the generalization of Dropout in which each connection, rather than each output unit, can be dropped with probability $1 - p$. DropConnect is similar to Dropout as it introduces dynamic sparsity within the model, but differs in that the sparsity is on the weights W, rather than the output vectors of a layer. In other words, the fully connected layer with DropConnect becomes a sparsely connected layer in which the connections are chosen at *random* during the training stage. Note that this is not equivalent to setting W to be a *fixed* sparse matrix during training.

For a DropConnect layer, the output is given as:

$$r = a\left((M \star W)\, v\right) \tag{3}$$

where M is a binary matrix encoding the connection information and $M_{ij} \sim Bernoulli(p)$. Each element of the mask M is drawn independently for each example during training, essentially instantiating a different connectivity for each example seen. Additionally, the biases are also masked out during training. From Eqn. 2 and Eqn. 3, it is evident that DropConnect is the generalization of Dropout to the full connection structure of a layer[1].

The paper structure is as follows: we outline details on training and running inference in a model using DropConnect in section 3, followed by theoretical justification for DropConnect in section 4, GPU implementation specifics in section 5, and experimental results in section 6.

3. Model Description

We consider a standard model architecture composed of four basic components (see Fig. 1a):

1. Feature Extractor: $v = g(x; W_g)$ where v are the output features, x is input data to the overall model, and W_g are parameters for the feature extractor. We choose $g()$ to be a multi-layered convolutional neural network (CNN) (LeCun et al., 1998), with W_g being the convolutional filters (and biases) of the CNN.
2. DropConnect Layer: $r = a(u) = a((M \star W)v)$ where v is the output of the feature extractor, W is a fully connected weight matrix, a is a non-linear activation function and M is the binary mask matrix.
3. Softmax Classification Layer: $o = s(r; W_s)$ takes as input r and uses parameters W_s to map this to a k dimensional output (k being the number of classes).
4. Cross Entropy Loss: $A(y, o) = -\sum_{i=1}^{k} y_i log(o_i)$ takes probabilities o and the ground truth labels y as input.

[1]This holds when $a(0) = 0$, as is the case for *tanh* and *relu* functions.

The overall model $f(x;\theta,M)$ therefore maps input data x to an output o through a sequence of operations given the parameters $\theta=\{W_g,W,W_s\}$ and randomly-drawn mask M. The correct value of o is obtained by summing out over all possible masks M:

$$o=\mathbf{E}_M\left[f(x;\theta,M)\right]=\sum_M p(M)f(x;\theta,M) \qquad (4)$$

This reveals the mixture model interpretation of DropConnect (and Dropout), where the output is a mixture of $2^{|M|}$ different networks, each with weight $p(M)$. If $p=0.5$, then these weights are equal and $o=\frac{1}{|M|}\sum_M f(x;\theta,M)=\frac{1}{|M|}\sum_M s(a((M\star W)v);W_s)$

3.1. Training

Training the model described in Section 3 begins by selecting an example x from the training set and extracting features for that example, v. These features are input to the DropConnect layer where a mask matrix M is first drawn from a $Bernoulli(p)$ distribution to mask out elements of both the weight matrix and the biases in the DropConnect layer. A key component to successfully training with DropConnect is the selection of a different mask for each training example. Selecting a single mask for a subset of training examples, such as a mini-batch of 128 examples, does not regularize the model enough in practice. Since the memory requirement for the M's now grows with the size of each mini-batch, the implementation needs to be carefully designed as described in Section 5.

Once a mask is chosen, it is applied to the weights and biases in order to compute the input to the activation function. This results in r, the input to the softmax layer which outputs class predictions from which cross entropy between the ground truth labels is computed. The parameters throughout the model θ then can be updated via stochastic gradient descent (SGD) by backpropagating gradients of the loss function with respect to the parameters, A'_θ. To update the weight matrix W in a DropConnect layer, the mask is applied to the gradient to update only those elements that were active in the forward pass. Additionally, when passing gradients down to the feature extractor, the masked weight matrix $M\star W$ is used. A summary of these steps is provided in Algorithm 1.

3.2. Inference

At inference time, we need to compute $r=1/|M|\sum_M a((M\star W)v)$, which naively requires the evaluation of $2^{|M|}$ different masks – plainly infeasible.

The Dropout work (Hinton et al., 2012) made the approximation: $\sum_M a((M\star W)v)\approx a(\sum_M(M\star W)v)$, i.e. averaging before the activation rather than after. Although this seems to work in practice, it is not justified mathematically, particularly for the $relu$ activation function.[2]

We take a different approach. Consider a single unit u_i before the activation function $a()$: $u_i=\sum_j(W_{ij}v_j)M_{ij}$. This is a weighted sum of Bernoulli variables M_{ij}, which can be approximated by a Gaussian via moment matching. The mean and variance of the units u are: $E_M[u]=pWv$ and $V_M[u]=p(1-p)(W\star W)(v\star v)$. We can then draw samples from this Gaussian and pass them through the activation function $a()$ before averaging them and presenting them to the next layer. Algorithm 2 summarizes the method. Note that the sampling can be done efficiently, since the samples for each unit and example can be drawn in parallel. This scheme is only an approximation in the case of multi-layer network, it works well in practise as shown in Experiments.

Algorithm 1 SGD Training with DropConnect

Input: example x, parameters θ_{t-1} from step $t-1$, learning rate η
Output: updated parameters θ_t
Forward Pass:
Extract features: $v\leftarrow g(x;W_g)$
Random sample M mask: $M_{ij}\sim Bernoulli(p)$
Compute activations: $r=a((M\star W)v)$
Compute output: $o=s(r;W_s)$
Backpropagate Gradients:
Differentiate loss A'_θ with respect to parameters θ:
Update softmax layer: $W_s=W_s-\eta A'_{W_s}$
Update DropConnect layer: $W=W-\eta(M\star A'_W)$
Update feature extractor: $W_g=W_g-\eta A'_{W_g}$

Algorithm 2 Inference with DropConnect

Input: example x, parameters θ, # of samples Z.
Output: prediction u
Extract features: $v\leftarrow g(x;W_g)$
Moment matching of u:
  $\mu\leftarrow E_M[u]\quad \sigma^2\leftarrow V_M[u]$
for z = 1 : Z **do** %% Draw Z samples
  for i = 1 : d **do** %% Loop over units in r
    Sample from 1D Gaussian $u_{i,z}\sim\mathcal{N}(\mu_i,\sigma_i^2)$
    $r_{i,z}\leftarrow a(u_{i,z})$
  end for
end for
Pass result $\hat{r}=\sum_{z=1}^{Z} r_z/Z$ to next layer

[2]Consider $u\sim N(0,1)$, with $a(u)=max(u,0)$. $a(E_M(u))=0$ but $E_M(a(u))=1/\sqrt{2}\pi\approx 0.4$.

Implementation	Mask Weight	Time(ms)				Speedup
		fprop	bprop acts	bprop weights	total	
CPU	float	480.2	1228.6	1692.8	3401.6	1.0 ×
CPU	bit	392.3	679.1	759.7	1831.1	1.9 ×
GPU	float(global memory)	21.6	6.2	7.2	35.0	97.2 ×
GPU	float(tex1D memory)	15.1	6.1	6.0	27.2	126.0 ×
GPU	bit(tex2D aligned memory)	2.4	2.7	3.1	**8.2**	414.8 ×
GPU(Lower Bound)	cuBlas + read mask weight	0.3	0.3	0.2	0.8	

Table 1. Performance comparison between different implementations of our DropConnect layer on NVidia GTX580 GPU relative to a 2.67Ghz Intel Xeon (compiled with `-O3` flag). Input dimension and Output dimension are 1024 and mini-batch size is 128. As reference we provide traditional matrix multiplication using the cuBlas library.

4. Model Generalization Bound

We now show a novel bound for the Rademacher complexity of the model $\hat{R}_\ell(\mathcal{F})$ on the training set (see appendix for derivation):

$$\hat{R}_\ell(\mathcal{F}) \leq p \left(2\sqrt{k}dB_s n\sqrt{d}B_h\right) \hat{R}_\ell(\mathcal{G}) \tag{5}$$

where $max|W_s| \leq B_s$, $max|W| \leq B$, k is the number of classes, $\hat{R}_\ell(\mathcal{G})$ is the Rademacher complexity of the feature extractor, n and d are the dimensionality of the input and output of the DropConnect layer respectively. The important result from Eqn. 5 is that the complexity is a linear function of the probability p of an element being kept in DropConnect or Dropout. When $p = 0$, the model complexity is zero, since the input has no influence on the output. When $p = 1$, it returns to the complexity of a standard model.

5. Implementation Details

Our system involves three components implemented on a GPU: 1) a feature extractor, 2) our DropConnect layer, and 3) a softmax classification layer. For 1 and 3 we utilize the Cuda-convnet package (Krizhevsky, 2012), a fast GPU based convolutional network library. We implement a custom GPU kernel for performing the operations within the DropConnect layer. Our code is available at `http:///cs.nyu.edu/~wanli/dropc`.

A typical fully connected layer is implemented as a matrix-matrix multiplication between the input vectors for a mini-batch of training examples and the weight matrix. The difficulty in our case is that each training example requires it's own random mask matrix applied to the weights and biases of the DropConnect layer. This leads to several complications:

1. For a weight matrix of size $d \times n$, the corresponding mask matrix is of size $d \times n \times b$ where b is the size of the mini-batch. For a 4096×4096 fully connected layer with mini-batch size of 128, the matrix would be too large to fit into GPU memory if each element is stored as a floating point number, requiring 8G of memory.
2. Once a random instantiation of the mask is created, it is non-trivial to access all the elements required during the matrix multiplications so as to maximize performance.

The first problem is not hard to address. Each element of the mask matrix is stored as a single bit to encode the connectivity information rather than as a float. The memory cost is thus reduced by 32 times, which becomes 256M for the example above. This not only reduces the memory footprint, but also reduces the bandwidth required as 32 elements can be accessed with each 4-byte read. We overcome the second problem using an efficient memory access pattern using 2D texture aligned memory. These two improvements are crucial for an efficient GPU implementation of DropConnect as shown in Table 1. Here we compare to a naive CPU implementation with floating point masks and get a 415× speedup with our efficient GPU design.

6. Experiments

We evaluate our DropConnect model for regularizing deep neural networks trained for image classification. All experiments use mini-batch SGD with momentum on batches of 128 images with the momentum parameter fixed at 0.9.

We use the following protocol for all experiments unless otherwise stated:

- Augment the dataset by: 1) randomly selecting cropped regions from the images, 2) flipping images horizontally, 3) introducing 15% scaling and rotation variations.
- Train 5 independent networks with random permutations of the training sequence.
- Manually decrease the learning rate if the network stops improving as in (Krizhevsky, 2012) according to a schedule determined on a validation set.
- Train the fully connected layer using Dropout, DropConnect, or neither (No-Drop).
- At inference time for DropConnect we draw $Z = 1000$

samples at the inputs to the activation function of the fully connected layer and average their activations.

To anneal the initial learning rate we choose a fixed multiplier for different stages of training. We report three numbers of epochs, such as 600-400-200 to define our schedule. We multiply the initial rate by 1 for the first such number of epochs. Then we use a multiplier of 0.5 for the second number of epochs followed by 0.1 again for this second number of epochs. The third number of epochs is used for multipliers of 0.05, 0.01, 0.005, and 0.001 in that order, after which point we report our results. We determine the epochs to use for our schedule using a validation set to look for plateaus in the loss function, at which point we move to the next multiplier. [3]

Once the 5 networks are trained we report two numbers: 1) the mean and standard deviation of the classification errors produced by each of the 5 independent networks, and 2) the classification error that results when averaging the output probabilities from the 5 networks before making a prediction. We find in practice this voting scheme, inspired by (Ciresan et al., 2012), provides significant performance gains, achieving state-of-the-art results in many standard benchmarks when combined with our DropConnect layer.

6.1. MNIST

The MNIST handwritten digit classification task (LeCun et al., 1998) consists of 28×28 black and white images, each containing a digit 0 to 9 (10-classes). Each digit in the $60,000$ training images and $10,000$ test images is normalized to fit in a 20×20 pixel box while preserving their aspect ratio. We scale the pixel values to the $[0, 1]$ range before inputting to our models.

For our first experiment on this dataset, we train models with two fully connected layers each with 800 output units using either *tanh*, *sigmoid* or *relu* activation functions to compare to Dropout in (Hinton et al., 2012). The first layer takes the image pixels as input, while the second layer's output is fed into a 10-class softmax classification layer. In Table 2 we show the performance of various activations functions, comparing No-Drop, Dropout and DropConnect in the fully connected layers. No data augmentation is utilized in this experiment. We use an initial learning rate of 0.1 and train for 600-400-20 epochs using our schedule.

From Table 2 we can see that both Dropout and DropConnect perform better than not using either method. DropConnect mostly performs better than Dropout in this task, with the gap widening when utilizing the voting over the 5 models.

neuron	model	error(%) 5 network	voting error(%)
relu	No-Drop	1.62 ± 0.037	1.40
	Dropout	1.28 ± 0.040	1.20
	DropConnect	1.20 ± 0.034	**1.12**
sigmoid	No-Drop	1.78 ± 0.037	1.74
	Dropout	1.38 ± 0.039	**1.36**
	DropConnect	1.55 ± 0.046	1.48
tanh	No-Drop	1.65 ± 0.026	1.49
	Dropout	1.58 ± 0.053	1.55
	DropConnect	1.36 ± 0.054	**1.35**

Table 2. MNIST classification error rate for models with two fully connected layers of 800 neurons each. No data augmentation is used in this experiment.

To further analyze the effects of DropConnect, we show three explanatory experiments in Fig. 2 using a 2-layer fully connected model on MNIST digits. Fig. 2a shows test performance as the number of hidden units in each layer varies. As the model size increases, No-Drop overfits while both Dropout and DropConnect improve performance. DropConnect consistently gives a lower error rate than Dropout. Fig. 2b shows the effect of varying the drop rate p for Dropout and DropConnect for a 400-400 unit network. Both methods give optimal performance in the vicinity of 0.5, the value used in all other experiments in the paper. Our sampling approach gives a performance gain over mean inference (as used by Hinton (Hinton et al., 2012)), but only for the DropConnect case. In Fig. 2c we plot the convergence properties of the three methods throughout training on a 400-400 network. We can see that No-Drop overfits quickly, while Dropout and DropConnect converge slowly to ultimately give superior test performance. DropConnect is even slower to converge than Dropout, but yields a lower test error in the end.

In order to improve our classification result, we choose a more powerful feature extractor network described in (Ciresan et al., 2012) (*relu* is used rather than *tanh*). This feature extractor consists of a 2 layer CNN with 32-64 feature maps in each layer respectively. The last layer's output is treated as input to the fully connected layer which has 150 *relu* units on which No-Drop, Dropout or DropConnect are applied. We report results in Table 3 from training the network on a) the original MNIST digits, b) cropped 24×24 images from random locations, and c) rotated and scaled versions of these cropped images. We use an initial

[3] In all experiments the bias learning rate is $2\times$ the learning rate for the weights. Additionally weights are initialized with $N(0, 0.1)$ random values for fully connected layers and $N(0, 0.01)$ for convolutional layers.

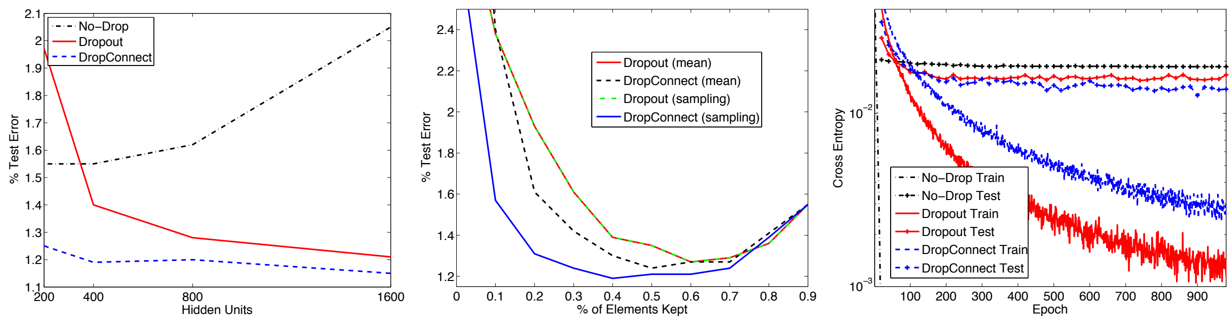

Figure 2. Using the MNIST dataset, in a) we analyze the ability of Dropout and DropConnect to prevent overfitting as the size of the 2 fully connected layers increase. b) Varying the drop-rate in a 400-400 network shows near optimal performance around the $p = 0.5$ proposed by (Hinton et al., 2012). c) we show the convergence properties of the train/test sets. See text for discussion.

learning rate of 0.01 with a 700-200-100 epoch schedule, no momentum and preprocess by subtracting the image mean.

crop	rotation scaling	model	error(%) 5 network	voting error(%)
no	no	No-Drop	0.77±0.051	0.67
		Dropout	0.59±0.039	**0.52**
		DropConnect	0.63±0.035	0.57
yes	no	No-Drop	0.50±0.098	0.38
		Dropout	0.39±0.039	0.35
		DropConnect	0.39±0.047	**0.32**
yes	yes	No-Drop	0.30±0.035	**0.21**
		Dropout	0.28±0.016	0.27
		DropConnect	0.28±0.032	**0.21**

Table 3. MNIST classification error. Previous state of the art is *0.47*% (Zeiler and Fergus, 2013) for a single model without elastic distortions and *0.23*% with elastic distortions and voting (Ciresan et al., 2012).

We note that our approach surpasses the state-of-the-art result of 0.23% (Ciresan et al., 2012), achieving a **0.21**% error rate, without the use of elastic distortions (as used by (Ciresan et al., 2012)).

6.2. CIFAR-10

CIFAR-10 is a data set of natural 32x32 RGB images (Krizhevsky, 2009) in 10-classes with 50,000 images for training and 10,000 for testing. Before inputting these images to our network, we subtract the per-pixel mean computed over the training set from each image.

The first experiment on CIFAR-10 (summarized in Table 4) uses the simple convolutional network feature extractor described in (Krizhevsky, 2012)(layers-80sec.cfg) that is designed for rapid training rather than optimal performance. On top of the 3-layer feature extractor we have a 64 unit fully connected layer which uses No-Drop, Dropout, or DropConnect. No data augmentation is utilized for this experiment.

Since this experiment is not aimed at optimal performance we report a single model's performance without voting. We train for 150-0-0 epochs with an initial learning rate of 0.001 and their default weight decay. DropConnect prevents overfitting of the fully connected layer better than Dropout in this experiment.

model	error(%)
No-Drop	23.5
Dropout	19.7
DropConnect	**18.7**

Table 4. CIFAR-10 classification error using the simple feature extractor described in (Krizhevsky, 2012)(layers-80sec.cfg) and with no data augmentation.

Table 5 shows classification results of the network using a larger feature extractor with 2 convolutional layers and 2 locally connected layers as described in (Krizhevsky, 2012)(layers-conv-local-11pct.cfg). A 128 neuron fully connected layer with *relu* activations is added between the softmax layer and feature extractor. Following (Krizhevsky, 2012), images are cropped to 24x24 with horizontal flips and no rotation or scaling is performed. We use an initial learning rate of 0.001 and train for 700-300-50 epochs with their default weight decay. Model voting significantly improves performance when using Dropout or DropConnect, the latter reaching an error rate of 9.41%. Additionally, we trained a model with 12 networks with DropConnect and achieved a state-of-the-art result of **9.32**%, indicating the power of our approach.

6.3. SVHN

The Street View House Numbers (SVHN) dataset includes 604,388 images (both training set and extra set) and 26,032 testing images (Netzer et al., 2011). Similar to MNIST, the goal is to classify the digit centered in each 32x32 RGB image. Due to the large variety of colors and brightness variations in the images, we pre-

model	error(%) 5 network	voting error(%)
No-Drop	11.18± 0.13	10.22
Dropout	11.52± 0.18	9.83
DropConnect	11.10± 0.13	**9.41**

Table 5. CIFAR-10 classification error using a larger feature extractor. Previous state-of-the-art is 9.5% (Snoek et al., 2012). Voting with 12 DropConnect networks produces an error rate of **9.32**%, significantly beating the state-of-the-art.

process the images using local contrast normalization as in (Zeiler and Fergus, 2013). The feature extractor is the same as the larger CIFAR-10 experiment, but we instead use a larger 512 unit fully connected layer with *relu* activations between the softmax layer and the feature extractor. After contrast normalizing, the training data is randomly cropped to 28×28 pixels and is rotated and scaled. We do not do horizontal flips. Table 6 shows the classification performance for 5 models trained with an initial learning rate of 0.001 for a 100-50-10 epoch schedule.

Due to the large training set size both Dropout and DropConnect achieve nearly the same performance as No-Drop. However, using our data augmentation techniques and careful annealing, the per model scores easily surpass the previous 2.80% state-of-the-art result of (Zeiler and Fergus, 2013). Furthermore, our voting scheme reduces the relative error of the previous state-of-to-art by 30% to achieve **1.94**% error.

model	error(%) 5 network	voting error(%)
No-Drop	2.26 ± 0.072	**1.94**
Dropout	2.25 ± 0.034	1.96
DropConnect	2.23 ± 0.039	**1.94**

Table 6. SVHN classification error. The previous state-of-the-art is *2.8*% (Zeiler and Fergus, 2013).

6.4. NORB

In the final experiment we evaluate our models on the 2-fold NORB (jittered-cluttered) dataset (LeCun et al., 2004), a collection of stereo images of 3D models. For each image, one of 6 classes appears on a random background. We train on 2-folds of $29,160$ images each and the test on a total of $58,320$ images. The images are downsampled from 108×108 to 48×48 as in (Ciresan et al., 2012).

We use the same feature extractor as the larger CIFAR-10 experiment. There is a 512 unit fully connected layer with *relu* activations placed between the softmax layer and feature extractor. Rotation and scaling of the training data is applied, but we do not crop or flip the images as we found that to hurt performance on this dataset. We trained with an initial learning rate of 0.001 and anneal for 100-40-10 epochs.

model	error(%) 5 network	voting error(%)
No-Drop	4.48 ± 0.78	3.36
Dropout	3.96 ± 0.16	**3.03**
DropConnect	4.14 ± 0.06	3.23

Table 7. NORM classification error for the jittered-cluttered dataset, using 2 training folds. The previous state-of-art is *3.57*% (Ciresan et al., 2012).

In this experiment we beat the previous state-of-the-art result of 3.57% using No-Drop, Dropout and DropConnect with our voting scheme. While Dropout surpasses DropConnect slightly, both methods improve over No-Drop in this benchmark as shown in Table 7.

7. Discussion

We have presented DropConnect, which generalizes Hinton *et al.* 's Dropout (Hinton et al., 2012) to the entire connectivity structure of a fully connected neural network layer. We provide both theoretical justification and empirical results to show that DropConnect helps regularize large neural network models. Results on a range of datasets show that DropConnect often outperforms Dropout. While our current implementation of DropConnect is slightly slower than No-Drop or Dropout, in large models models the feature extractor is the bottleneck, thus there is little difference in overall training time. DropConnect allows us to train large models while avoiding overfitting. This yields state-of-the-art results on a variety of standard benchmarks using our efficient GPU implementation of DropConnect.

8. Appendix

8.1. Preliminaries

Definition 1 (DropConnect Network)**.** *Given data set S with ℓ entries: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell\}$ with labels $\{y_1, y_2, \ldots, y_\ell\}$, we define the DropConnect network as a mixture model:* $\mathbf{o} = \sum_M p(M) f(\mathbf{x}; \theta, M) = \mathbf{E}_M \left[f(\mathbf{x}; \theta, M) \right]$

Each network $f(x; \theta, M)$ has weights $p(M)$ and network parameters are $\theta = \{W_s, W, W_g\}$. W_s are the softmax layer parameters, W are the DropConnect layer parameters and W_g are the feature extractor parameters. Further more, M is the DropConnect layer mask.

Now we reformulate the cross-entropy loss on top of the softmax into a single parameter function that combines the softmax output and labels, as a logistic.

Definition 2 (Logistic Loss). *The following loss function defined on k-class classification is call the logistic loss function:*

$$A_y(o) = -\sum_i y_i \ln \frac{\exp o_i}{\sum_j \exp(o_j)} = -o_i + \ln \sum_j \exp(o_j)$$

where y is binary vector with i^{th} bit set on

Lemma 1. *Logistic loss function A has the following properties: 1) $A_y(0) = \ln k$, 2) $-1 \le A'_y(o) \le 1$, and 3)$A''_y(o) \ge 0$.*

Definition 3 (Rademacher complexity). *For a sample $S = \{x_1, \dots, x_\ell\}$ generated by a distribution D on set X and a real-valued function class $\mathcal{F}$ in domain X, the empirical Rademacher complexity of $\mathcal{F}$ is the random variable:*

$$\hat{R}_\ell(\mathcal{F}) = \mathbf{E}_\sigma \left[\sup_{f \in \mathcal{F}} |\frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(x_i)| \mid x_1, \dots, x_\ell \right]$$

where sigma $= \{\sigma_1, \dots, \sigma_\ell\}$ are independent uniform $\{\pm 1\}$-valued (Rademacher) random variables. The Rademacher complexity of $\mathcal{F}$ is $R_\ell(\mathcal{F}) = \mathbf{E}_S \left[\hat{R}_\ell(\mathcal{F}) \right]$.

8.2. Bound Derivation

Lemma 2 ((Ledoux and Talagrand, 1991)). *Let $\mathcal{F}$ be class of real functions and $\mathcal{H} = [\mathcal{F}_j]_{j=1}^k$ be a k-dimensional function class. If $\mathcal{A}$: $\mathbf{R}^k \to \mathbf{R}$ is a Lipschitz function with constant L and satisfies $\mathcal{A}(0) = 0$, then $\hat{R}_\ell(\mathcal{A} \circ \mathcal{H}) \le 2kL\hat{R}_\ell(\mathcal{F})$*

Lemma 3 (Classifier Generalization Bound). *Generalization bound of a k-class classifier with logistic loss function is directly related Rademacher complexity of that classifier:*

$\mathbf{E}[A_y(o)] \le \frac{1}{\ell}\sum_{i=1}^{\ell} A_{y_i}(o_i) + 2k\hat{R}_\ell(\mathcal{F}) + 3\sqrt{\frac{ln(2/\delta)}{2\ell}}$

Lemma 4. *For all neuron activations: sigmoid, tanh and relu, we have: $\hat{R}_\ell(a \circ \mathcal{F}) \le 2\hat{R}_\ell(\mathcal{F})$*

Lemma 5 (Network Layer Bound). *Let $\mathcal{G}$ be the class of real functions $R^d \to R$ with input dimension $\mathcal{F}$, i.e. $\mathcal{G} = [\mathcal{F}_j]_{j=1}^d$ and $\mathcal{H}_B$ is a linear transform function parametrized by W with $\|W\|_2 \le B$, then $\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) \le \sqrt{d}B\hat{R}_\ell(\mathcal{F})$*

Proof. $\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) = \mathbf{E}_\sigma \left[\sup_{h \in \mathcal{H}, g \in \mathcal{G}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i h \circ g(x_i) \right| \right]$
$= \mathbf{E}_\sigma \left[\sup_{g \in \mathcal{G}, \|W\| \le B} \left| \left\langle W, \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i g(x_i) \right\rangle \right| \right]$
$\le B\mathbf{E}_\sigma \left[\sup_{f^j \in \mathcal{F}} \left\| \left[\frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i^j f^j(x_i) \right]_{j=1}^d \right\| \right]$
$= B\sqrt{d}\mathbf{E}_\sigma \left[\sup_{f \in \mathcal{F}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(x_i) \right| \right] = \sqrt{d}B\hat{R}_\ell(\mathcal{F})$ □

Remark 1. *Given a layer in our network, we denote the function of all layers before as $\mathcal{G} = [\mathcal{F}_j]_{j=1}^d$. This layer has the linear transformation function $\mathcal{H}$ and activation function a. By Lemma 4 and Lemma 5, we know the network complexity is bounded by:*

$$\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) \le c\sqrt{d}B\hat{R}_\ell(\mathcal{F})$$

where $c = 1$ for identity neuron and $c = 2$ for others.

Lemma 6. *Let $\mathcal{F}_M$ be the class of real functions that depend on M, then $\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) \le \mathbf{E}_M \left[\hat{R}_\ell(\mathcal{F}_M) \right]$*

Proof. $\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) = \hat{R}_\ell\left(\sum_M p(m)\mathcal{F}_M\right) \le \sum_M \hat{R}_\ell(p(m)\mathcal{F}_M) \le \sum_M |p(m)|\hat{R}_\ell(\mathcal{F}_M) = \mathbf{E}_M \left[\hat{R}_\ell(\mathcal{F}_M) \right]$ □

Theorem 1 (DropConnect Network Complexity). *Consider the DropConnect neural network defined in Definition 1. Let $\hat{R}_\ell(\mathcal{G})$ be the empirical Rademacher complexity of the feature extractor and $\hat{R}_\ell(\mathcal{F})$ be the empirical Rademacher complexity of the whole network. In addition, we assume:*

1. *weight parameter of DropConnect layer $|W| \le B_h$*
2. *weight parameter of s, i.e. $|W_s| \le B_s$ (L2-norm of it is bounded by $\sqrt{dk}B_s$).*

Then we have: $\hat{R}_\ell(\mathcal{F}) \le p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$

Proof.

$$\hat{R}_\ell(\mathcal{F}) = \hat{R}_\ell(\mathbf{E}_M[f(\mathbf{x};\theta,M)]) \le \mathbf{E}_M\left[\hat{R}_\ell(f(\mathbf{x};\theta,M)\right] \quad (6)$$
$$\le (\sqrt{dk}B_s)\sqrt{d}\mathbf{E}_M\left[\hat{R}_\ell(a \circ h_M \circ g)\right] \quad (7)$$
$$= 2\sqrt{k}dB_s\mathbf{E}_M\left[\hat{R}_\ell(h_M \circ g)\right] \quad (8)$$

where $h_M = (M \star W)v$. Equation (6) is based on Lemma 6, Equation (7) is based on Lemma 5 and Equation (8) follows from Lemma 4.

$$\mathbf{E}_M\left[\hat{R}_\ell(h_M \circ g)\right]$$
$$= \mathbf{E}_{M,\sigma}\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i W^T D_M g(x_i)\right|\right] \quad (9)$$
$$= \mathbf{E}_{M,\sigma}\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\left\langle D_M W, \frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g(x_i)\right\rangle\right|\right]$$
$$\le \mathbf{E}_M\left[\max_W \|D_M W\|\right]\mathbf{E}_\sigma\left[\sup_{g^j\in\mathcal{G}}\left\|\left[\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g^j(x_i)\right]_{j=1}^n\right\|\right] \quad (10)$$
$$\le B_h p\sqrt{nd}\left(\sqrt{n}\hat{R}_\ell(\mathcal{G})\right) = pn\sqrt{d}B_h\hat{R}_\ell(\mathcal{G})$$

where D_M in Equation (9) is an diagonal matrix with diagonal elements equal to m and inner product properties lead to Equation (10). Thus, we have: $\hat{R}_\ell(\mathcal{F}) \le p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$ □

References

D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3642–3649, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-1-4673-1226-4.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toront, 2009.

A. Krizhevsky. cuda-convnet. http://code.google.com/p/cuda-convnet/, 2012.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 –2324, nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society.

M. Ledoux and M. Talagrand. *Probability in Banach Spaces*. Springer, New York, 1991.

D. J. C. Mackay. Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks. In *Bayesian methods for backpropagation networks*. Springer, 1995.

V. Nair and G. E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *ICML*, 2010.

Y. Netzer, T. Wang, Coates A., A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

J. Snoek, H. Larochelle, and R. A. Adams. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In *NIPS*, 1991.

M. D. Zeiler and R. Fergus. Stochastic pooling for regualization of deep convolutional neural networks. In *ICLR*, 2013.

Regularization of Neural Networks using DropConnect Supplementary Material

1. Preliminaries

Definition 1 (DropConnect Network). *Given data set S with ℓ entries: $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell\}$ with labels $\{y_1, y_2, \ldots, y_\ell\}$, we define DropConnect network as a mixture model:*

$$\mathbf{o} = \sum_m p(M) f(\mathbf{x}; \theta, M) = \mathbf{E}_m \left[f(\mathbf{x}; \theta, M) \right] \quad (1)$$

Each network $f(x; \theta, M)$ has weights $p(M)$ and network parameters are $\theta = \{W_s, W, W_g\}$. W_s are the softmax layer parameters, w are the DropConnect layer parameters and W_g are the feature extractor parameters. Further more, m is the DropConnect layer mask.

Remark 1. *when each element of M_i has equal probability of being on and off ($p = 0.5$), the mixture model has equal weights for all sub-models $f(\mathbf{x}; \theta, M)$, otherwise the mixture model has larger weights in some sub-models than others.*

Reformulate cross-entropy loss on top of soft-max into a single parameter function that combines soft-max output and labels. Same as logistic.

Definition 2 (Logistic Loss). *The following loss function defined on k-class classification is call logistic loss function:*

$$A_y(o) = -\sum_i y_i \ln \frac{\exp o_i}{\sum_j \exp(o_j)} = -o_i + \ln \sum_j \exp(o_j)$$

where y is binary vector with i^{th} bit set on

Lemma 1. *Logistic loss function A has the following properties:*

1. $A_y(0) = \ln k$
2. $-1 \leq A'_y(o) \leq 1$
3. $A''_y(o) \geq 0$.

The first one says $A(0)$ is depend on some constant related with number of labels. The second one says A is Lipschitz function with $L = 1$. The third one says A is a convex function w.r.t x.

Definition 3 (Rademacher complexity). *For a sample $S = \{x_1, \ldots, x_\ell\}$ generated by a distribution D on set X and a real-valued function class $\mathcal{F}$ in domain X, the empirical Rademacher complexity of $\mathcal{F}$ is the random variable:*

$$\hat{R}_\ell(\mathcal{F}) = \mathbf{E}_\sigma \left[\sup_{f \in \mathcal{F}} |\frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(x_i)| \mid x_1, \ldots, x_\ell \right]$$

where sigma $= \{\sigma_1, \ldots, \sigma_\ell\}$ are independent uniform $\{\pm 1\}$-valued (Rademacher) random variables. The Rademacher complexity of $\mathcal{F}$ is $R_\ell(\mathcal{F}) = \mathbf{E}_S \left[\hat{R}_\ell(\mathcal{F}) \right]$.

Theorem 1 ((Koltchinskii and Panchenko, 2000)). *Fix $\delta \in (0,1)$ and let $\mathcal{F}$ be a class of functions mapping from M to $[0,1]$. Let $(M_i)_{i-1}^{\ell}$ be drawn independently according to a probability distribution D. Then with probability at least $1-\delta$ over random draws of samples of size ℓ, every $f \in \mathcal{F}$ satisfies:*

$$\begin{aligned} \mathbf{E}[f(M)] &\leq \hat{\mathbf{E}}[f(M)] + R_\ell(\mathcal{F}) + \sqrt{\frac{ln(2/\delta)}{2\ell}} \\ &\leq \hat{\mathbf{E}}[f(M)] + \hat{R}_\ell(\mathcal{F}) + 3\sqrt{\frac{ln(2/\delta)}{2\ell}} \end{aligned}$$

2. Bound Derivation

Theorem 2 ((Ledoux and Talagrand, 1991)). *Let $\mathcal{F}$ be class of real functions. If $\mathcal{A}$: $\mathbf{R} \to \mathbf{R}$ is Lipschitz with constant L and satisfies $\mathcal{A}(0) = 0$, then $\hat{R}_\ell(\mathcal{A} \circ F) \leq 2L\hat{R}(\mathcal{F})$*

Lemma 2. *Let $\mathcal{F}$ be class of real functions and $\mathcal{H} = [\mathcal{F}_j]_{j=1}^k$ be a k-dimensional function class. If $\mathcal{A}$: $\mathbf{R}^k \to \mathbf{R}$ is a Lipschitz function with constant L and satisfies $\mathcal{A}(0) = 0$, then $\hat{R}_\ell(\mathcal{A} \circ \mathcal{H}) \leq 2kL\hat{R}_\ell(\mathcal{F})$*

Lemma 3 (Classifier Generalization Bound). *Generalization bound of a k-class classifier with logistic loss function is directly related Rademacher complexity of that classifier*

$$\mathbf{E}[A_y(o)] \leq \frac{1}{\ell} \sum_{i=1}^{\ell} A_{y_i}(o_i) + 2k\hat{R}_\ell(\mathcal{F}) + 3\sqrt{\frac{ln(2/\delta)}{2\ell}}$$

Proof. From Lemma 1, Logistic loss function $(A - c)(x) \in \mathcal{A}$ due to $(A-c)'(x) \leq 1$ and $(A-c)(0) = 0$ with some constant c. By Lemma 2: $\hat{R}_\ell((A-c) \circ \mathcal{F}) \leq 2k\hat{R}_\ell(\mathcal{F})$ □

Lemma 4. *For all neuron activations: sigmoid, tanh and relu, we have:* $\hat{R}_\ell(a \circ \mathcal{F}) \leq 2\hat{R}_\ell(\mathcal{F})$

Lemma 5 (Network Layer Bound)**.** *Let* $\mathcal{G}$ *be the class of real functions* $R^d \to R$ *with input dimension* $\mathcal{F}$*, i.e.* $\mathcal{G} = [\mathcal{F}_j]_{j=1}^d$ *and* $\mathcal{H}_B$ *is a linear transform function parameterized by* W *with* $\|W\|_2 \leq B$*, then* $\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) \leq \sqrt{d} B \hat{R}_\ell(\mathcal{F})$

Proof.

$$
\begin{aligned}
& \hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) \\
= \; & \mathbf{E}_\sigma \left[\sup_{h \in \mathcal{H}, g \in \mathcal{G}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i h \circ g(x_i) \right| \right] \\
= \; & \mathbf{E}_\sigma \left[\sup_{g \in \mathcal{G}, \|W\| \leq B} \left| \left\langle W, \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i g(x_i) \right\rangle \right| \right] \\
\leq \; & B \mathbf{E}_\sigma \left[\sup_{f^j \in \mathcal{F}} \left\| \left[\frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i^j f^j(x_i) \right]_{j=1}^{d} \right\| \right] \\
= \; & B \sqrt{d} \mathbf{E}_\sigma \left[\sup_{f \in \mathcal{F}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(x_i) \right| \right] = \sqrt{d} B \hat{R}_\ell(\mathcal{F})
\end{aligned}
$$

□

Remark 2. *Given a layer in our network, we denote the function of all layers before as* $\mathcal{G} = [\mathcal{F}_j]_{j=1}^d$*. This layer has the linear transformation function* $\mathcal{H}$ *and activation function* a*. By Lemma 4 and Lemma 5, we know the network complexity is bounded by:*

$$\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) \leq c\sqrt{d} B \hat{R}_\ell(\mathcal{F})$$

where $c = 1$ *for identity neuron and* $c = 2$ *for others.*

Lemma 6. *Let* $\mathcal{F}_M$ *be the class of real functions that depend on* m*, then* $\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) \leq \mathbf{E}_M\left[\hat{R}_\ell(\mathcal{F}_M)\right]$

Proof.

$$
\begin{aligned}
\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) &= \hat{R}_\ell\left(\sum_M p(M)\mathcal{F}_M\right) \leq \sum_M \hat{R}_\ell(p(M)\mathcal{F}_M) \\
&\leq \sum_M |p(M)| \hat{R}_\ell(\mathcal{F}_M) = \mathbf{E}_M\left[\hat{R}_\ell(\mathcal{F}_M)\right]
\end{aligned}
$$

because of common fact: 1) $\hat{R}_\ell(c\mathcal{F}) = |c|\hat{R}_\ell(\mathcal{F})$ and 2) $\hat{R}_\ell(\sum_i \mathcal{F}_i) \leq \sum_i \hat{R}_\ell(\mathcal{F}_i)$ □

Theorem 3 (DropConnect Network Complexity)**.** *Consider the DropConnect neural network defined in Definition 1. Let* $\hat{R}_\ell(\mathcal{G})$ *be the empirical Rademacher complexity of the feature extractor and* $\hat{R}_\ell(\mathcal{F})$ *be the empirical Rademacher complexity of the whole network. In addition, we assume:*

1. *weight parameter of DropConnect layer* $|W| \leq B_h$
2. *weight parameter of* s*, i.e.* $|W_s| \leq B_s$ *(L2-norm of it is bounded by* $\sqrt{dk}B_s$*).*

Then we have:

$$\hat{R}_\ell(\mathcal{F}) \leq p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$$

Proof.

$$
\begin{aligned}
& \hat{R}_\ell(\mathcal{F}) = \hat{R}_\ell(\mathbf{E}_M[f(\mathbf{x};\theta,M)]) \\
\leq \; & \mathbf{E}_M\left[\hat{R}_\ell(f(\mathbf{x};\theta,M)\right] && (2) \\
= \; & \mathbf{E}_M\left[\hat{R}_\ell(s \circ a \circ h_m \circ g)\right] \\
\leq \; & (\sqrt{dk}B_s)\sqrt{d}\mathbf{E}_M\left[\hat{R}_\ell(a \circ h_m \circ g)\right] && (3) \\
= \; & 2\sqrt{k}dB_s\mathbf{E}_M\left[\hat{R}_\ell(h_m \circ g)\right] && (4)
\end{aligned}
$$

where $h_m = (M \circ W)v$. Equation (2) is based on Lemma 6, Equation (3) is based on Lemma 5 and Equation (4) follows from Lemma 4.

$$
\begin{aligned}
& \mathbf{E}_M\left[\hat{R}_\ell(h_m \circ g)\right] \\
= \; & \mathbf{E}_{m,\sigma}\left[\sup_{h \in \mathcal{H}, g \in \mathcal{G}} \left| \frac{2}{\ell}\sum_{i=1}^{\ell} \sigma_i w^T D_M g(x_i) \right| \right] && (5) \\
= \; & \mathbf{E}_{m,\sigma}\left[\sup_{h \in \mathcal{H}, g \in \mathcal{G}} \left| \left\langle D_M w, \frac{2}{\ell}\sum_{i=1}^{\ell} \sigma_i g(x_i) \right\rangle \right| \right] \\
\leq \; & \mathbf{E}_M\left[\max_w \|D_M w\|\right] \mathbf{E}_\sigma \left[\sup_{g^j \in \mathcal{G}} \left\| \left[\frac{2}{\ell}\sum_{i=1}^{\ell} \sigma_i g^j(x_i) \right]_{j=1}^{n} \right\| \right] && (6) \\
\leq \; & B_h p\sqrt{nd}\left(\sqrt{n}\hat{R}_\ell(\mathcal{G})\right) = pn\sqrt{d}B_h\hat{R}_\ell(\mathcal{G})
\end{aligned}
$$

where D_M in Equation (5) is an diagonal matrix with diagonal elements equal to m and inner product properties lead to Equation (6). Thus, we have

$$\hat{R}_\ell(\mathcal{F}) \leq p\left(2\sqrt{k}dB_s n\sqrt{d}B_h\right)\hat{R}_\ell(\mathcal{G})$$

□

Remark 3. *Theorem 3 implies that p is an additional regularizer we have added to network when we convert a normal neural network to a network with DropConnect layers. Consider the following extreme cases:*

1. $p = 0$*: the network generalization bound equals to 0, which is true because classifier does not depends on input any more*
2. $p = 1$*: reduce to normal network*

Symbol	Description	Related Formula
y	Data Label, can either be integer label for bit vector(depends on context)	
x	Network input data	
$g(.)$	Feature extractor function with parameter W_g	
v	Feature extractor network output	$v = g(x, W_g)$
M	DropConnect connection information parameter (weight mask)	
$h(.)$	DropConnect transformation function with parameter W, M	
u	DropConnect output	$u = h(v; W, M)$
$a(.)$	DropConnect activation function	
r	DropConnect after activation	$r = a(u)$
$s(.)$	Dimension reduction layer function with parameter W_s	
o	Dimension reduction layer output (network output)	$o = s(r; W_s)$
θ	All parameter of network expect weight mask	$\theta = \{W_s, W, W_g\}$
$f(.)$	Overall classifier(network) output	$o = f(x; \theta, M)$
λ	Weight penalty	
$A(.)$	Data Loss Function	$A(o - y)$
$L(.)$	Over all objective function	$L(x, y) = \sum_i A(o_i - y_i) + 1/2\lambda\|W\|_2^2$
n	Dimension of feature extractor output	
d	Dimension of DropConnect layer output	
k	number of class	$dim(y) = k$

Table 1. Symbol Table

References

V. Koltchinskii and D. Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, 30:2002, 2000.

M. Ledoux and M. Talagrand. *Probability in Banach Spaces*. Springer, New York, 1991.