Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks

Luo Chunjie [1 2] **Zhan Jianfeng** [1] **Wang Lei** [1] **Yang Qiang** [3]

Abstract

Traditionally, multi-layer neural networks use dot product between the output vector of previous layer and the incoming weight vector as the input to activation function. The result of dot product is unbounded, thus increases the risk of large variance. Large variance of neuron makes the model sensitive to the change of input distribution, thus results in poor generalization, and aggravates the internal covariate shift which slows down the training. To bound dot product and decrease the variance, we propose to use cosine similarity instead of dot product in neural networks, which we call cosine normalization. Our experiments show that cosine normalization in fully-connected neural networks notably reduces the test err with lower divergence, compared to other normalization techniques. Applied to convolutional networks, cosine normalization also significantly enhances the accuracy of classification and accelerates the training.

1. Introduction

Deep neural networks have received great success in recent years in many areas, e.g. image recognition (Krizhevsky et al., 2012), speech processing (Hinton et al., 2012), natural language processing (Mikolov et al., 2013), Go game (Silver et al., 2016). Training deep neural networks is nontrivial task. Gradient descent is commonly used to train neural networks. However, due to gradient vanishing problem (Hochreiter et al., 2001), it works badly when directly applying to deep networks.

Lots of approaches have been adopted to overcome the difficulty of training deep networks. For example, pre-training (Hinton et al., 2006) (Hinton & Salakhutdinov, 2006), special network structure (Simonyan & Zisserman, 2014) (Szegedy et al., 2015) (He et al., 2016), ReLU activation (Nair & Hinton, 2010) (Maas et al., 2013), noise injecting (Wan et al., 2013) (Srivastava et al., 2014), normalization (Ioffe & Szegedy, 2015) (Salimans & Kingma, 2016) (Ba et al., 2016).

In previous work, multi-layer neural networks use dot product (also called inner product) between the output vector of previous layer and the incoming weight vector as the input to activation function.

$$net = \vec{w} \cdot \vec{x} \tag{1}$$

where net is the input to activation function (pre-activation), $\vec{w}$ is the incoming weight vector, and $\vec{x}$ is the input vector which is also the output vector of previous layer, $(\cdot)$ indicates dot product. Equation 1 can be rewritten as Equation 2, where $\cos\theta$ is the cosine of angle between $\vec{w}$ and $\vec{x}$, $|\,|$ is the Euclidean norm of vector.

$$net = |\vec{w}|\,|\vec{x}|\cos\theta \tag{2}$$

The result of dot product is unbounded, thus increases the risk of large variance. Large variance of neuron makes the model sensitive to the change of input distribution, thus results in poor generalization. Large variance could also aggravate the internal covariate shift which slows down the training (Ioffe & Szegedy, 2015). Using small weights can alleviate this problem. Weight decay (L2-norm) (Krogh & Hertz, 1991) and max normalization (max-norm) (Srebro & Shraibman, 2005) (Srivastava et al., 2014) are methods that could decrease the weights. Batch normalization (Ioffe & Szegedy, 2015) uses statistics calculated from mini-batch training examples to normalize the result of dot product, while layer normalization (Ba et al., 2016) uses statistics from the same layer on a single training case. The variance can be constrained within certain range using batch or layer normalization. Weight normalization (Salimans & Kingma, 2016) re-parameterizes the weight vector by dividing its norm, thus partially bounds the result of dot product.

To thoroughly bound dot product, a straight-forward idea is to use cosine similarity. Similarity (or distance) based

[1]Institute of Computing Technology, Chinese Academy of Sciences [2]University of Chinese Academy of Sciences [3]Beijing Academy of Frontier Science and Technology. Correspondence to: Luo Chunjie <luochunjie@ict.ac.cn>, Zhan Jianfeng <zhanjianfeng@ict.ac.cn>, Wang Lei <wanglei_2011@ict.ac.cn>, Yang Qiang <yangqiang@mail.bafst.com>.

methods are widely used in data mining and machine learning. Particularly, cosine similarity is most commonly used in high dimensional spaces. For example, in information retrieval and text mining, cosine similarity gives a useful measure of how similar two documents are (Singhal, 2001). Also, it is widely used to measure cohesion within clusters in data mining (Tan et al., 2006)

In this paper, we combine cosine similarity with neural networks. We use cosine similarity instead of dot product when computing the pre-activation. That can be seen as a normalization procedure, which we call cosine normalization. Equation 3 shows the cosine normalization.

$$net_{norm} = \cos\theta = \frac{\vec{w} \cdot \vec{x}}{|\vec{w}|\,|\vec{x}|} \tag{3}$$

By ignoring the magnitude of $\vec{w}$ and $\vec{x}$, the input to activation function is bounded between -1 and 1. It is more robust for different input magnitude. Higher learning rate could be used for training without the risk of large variance. Moreover, network with cosine normalization can be trained by both batch gradient descent and stochastic gradient descent, since it does not depend on any statistics on batch or mini-batch examples.

We compare our cosine normalization with batch, weight and max normalization in fully-connected neural network on the MNIST and 20NEWS GROUP data sets. Additionally, convolutional network with cosine normalization is evaluated on the CIFAR-10 data set. Here is a brief summary about cosine normalization:

- It is simple and easily implemented.
- It bounds the result of dot product within a very narrow range, leading small variance of pre-activation.
- It is more stable than other normalization techniques.
- Using ReLU activation function, there is no need for additional parameters or hyper parameters.
- With stochastic gradient descent or dropout, it further improves the generalization.
- It is suitable for deep architecture. Increasing the layers improves the test accuracy.
- It accelerates neural networks training.
- It significantly enhances the test accuracy for fully-connected networks as well as convolutional networks.

2. Background and Motivation

Large variance of neuron in neural network makes the model sensitive to the change of input distribution, thus result in poor generalization. Moreover, variance could be amplified as information moves forward along layers, especially in deep network. Large variance could also aggravate the internal covariate shift, which refers the change of distribution of each layer during training, as the parameters of previous layers change (Ioffe & Szegedy, 2015). Internal covariate shift slows down the training because the layers need to continuously adapt to the new distribution. Traditionally, neural networks use dot product to compute the pre-activation of neuron. The result of dot product is unbounded. That is to say, the result could be any value in the whole real space, thus increases the risk of large variance.

Using small weights can alleviate this problem, since the pre-activation net in Equation 2 will be decreased when $|\vec{w}|$ is small. Weight decay (Krogh & Hertz, 1991) and max normalization (Srebro & Shraibman, 2005) (Srivastava et al., 2014) are methods that try to make the weights to be small. Weight decay adds an extra term to the cost function that penalizes the squared value of each weight separately. Max normalization puts a constraint on the maximum squared length of the incoming weight vector of each neuron. If update violates this constraint, max normalization scales down the vector of incoming weights to the allowed length. The objective (or direction to objective) of original optimization problem is changed when using weight decay (or max normalization). Moreover, they bring additional hyper parameters that should be carefully preset.

Batch normalization (Ioffe & Szegedy, 2015) uses statistics calculated from mini-batch training examples to normalize the pre-activation. The normalized value is re-scaled and re-shifted using additional parameters. Since batch normalization uses the statistics on mini-batch examples, its effect is dependent on the mini-batch size. To overcome this problem, normalization propagation (Arpit et al., 2016) uses a data-independent parametric estimate of mean and standard deviation, while layer normalization (Ba et al., 2016) computes the mean and standard deviation from the same layer on a single training case. Weight normalization (Salimans & Kingma, 2016) re-parameterizes the incoming weight vector by dividing its norm. It decouples the length of weight vector from its direction, thus partially bounds the result of dot product. But it does not consider the length of input vector. These methods all bring additional parameters to be learned, thus make the model more complex.

An important source of inspiration for our work is cosine similarity, which is widely used in data mining and machine learning (Singhal, 2001) (Tan et al., 2006). To thor-

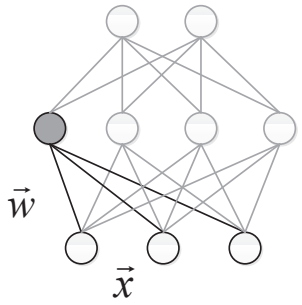


Figure 1. A simple neural network. The output of hidden unit is the nonlinear transform of dot product between input vector and incoming weight vector. That is computed by $f(\vec{w} \cdot \vec{x})$. With cosine normalization, The output of hidden unit is computed by $f(\frac{\vec{w} \cdot \vec{x}}{|\vec{w}||\vec{x}|})$

oughly bound dot product, a straight-forward idea is to use cosine similarity. We combine cosine similarity with neural network, and the details will be described in the next section.

3. Cosine Normalization

To decrease the variance of neuron, we propose a new method, called cosine normalization, which simply uses cosine similarity instead of dot product in neural network. A simple multi-layer neural network is shown in Figure 1. Using cosine normalization, the output of hidden unit is computed by Equation 4.

$$o = f(net_{norm}) = f(\cos\theta) = f(\frac{\vec{w} \cdot \vec{x}}{|\vec{w}|\,|\vec{x}|}) \tag{4}$$

where net_{norm} is the normalized pre-activation, $\vec{w}$ is the incoming weight vector and $\vec{x}$ is the input vector, $(\cdot)$ indicates dot product, f is nonlinear activation function. Cosine normalization bounds the pre-activation between -1 and 1. The result could be even smaller when the dimension is high. As a result, the variance can be controlled within a very narrow range. We found that using ReLU activation function $max(0, net_{norm})$, the result of cosine normalization needs no re-shifting and re-scaling, which are commonly used in other normalization methods like batch normalization or layer normalization. Therefore, there is no additional parameter to be learned or hyper-parameter to be preset. Meanwhile, cosine normalization performs the same computation in forward propagation at training and inference times.

One thing should be noticed is that cosine similarity can only measure the similarity between two non-zero vectors, since denominator can not be zero. Non-zero bias can be added to avoid the situation of zero vector. Let $\vec{w} = [w_1, w_2...w_i]$ and $\vec{x} = [x_1, x_2...x_i]$. After adding bias, $\vec{w} = [w_0, w_1, w_2...w_i]$ and $\vec{x} = [x_0, x_1, x_2...x_i]$, where w_0 and x_0 should be non-zero.

We can use gradient descent (back propagation) to train the neural network with cosine normalization. Cosine normalization does not depend on any statistics on batch or mini-batch examples, so the model can be trained by both batch gradient descent and stochastic gradient descent. The procedure of back propagation in neural network with cosine normalization is the same as ordinary neural network except the derivative of net_{norm} with respect to w or x.

To show the derivative conveniently, dot product can be rewritten as Equation 5, where w_i indicates the i dimension of vector $\vec{w}$, and x_i indicates the i dimension of vector $\vec{x}$.

$$net = \sum_i (w_i x_i) \tag{5}$$

Therefore, the derivative of net with respect to w_i or x_i in ordinary neural network can be calculated by Equation 6 or Equation 7.

$$\frac{\partial net}{\partial w_i} = x_i \tag{6}$$

$$\frac{\partial net}{\partial x_i} = w_i \tag{7}$$

Correspondingly, the cosine normalization can be rewritten as Equation 8.

$$net_{norm} = \cos\theta = \frac{\sum_i (w_i x_i)}{\sqrt{\sum_i (w_i^2)}\sqrt{\sum_i (x_i^2)}} \tag{8}$$

Then, the derivative of net_{norm} with respect to w_i or x_i can be calculated by Equation 9 or Equation 10.

$$\frac{\partial net_{norm}}{\partial w_i} = \frac{x_i}{\sqrt{\sum_i (w_i^2)}\sqrt{\sum_i (x_i^2)}} - \frac{w_i \sum_i (w_i x_i)}{(\sqrt{\sum_i (w_i^2)})^3 \sqrt{\sum_i (x_i^2)}} \tag{9}$$

$$\frac{\partial net_{norm}}{\partial x_i} = \frac{w_i}{\sqrt{\sum_i (w_i^2)}\sqrt{\sum_i (x_i^2)}} - \frac{x_i \sum_i (w_i x_i)}{\sqrt{\sum_i (w_i^2)}(\sqrt{\sum_i (x_i^2)})^3} \tag{10}$$

Equation 9 or Equation 10 can be briefly written as Equation 11 or Equation 12.

$$\frac{\partial net_{norm}}{\partial w_i} = \frac{x_i}{|\vec{w}|\,|\vec{x}|} - \frac{w_i(\vec{w} \cdot \vec{x})}{|\vec{w}|^3\,|\vec{x}|} \tag{11}$$

$$\frac{\partial net_{norm}}{\partial x_i} = \frac{w_i}{|\vec{w}|\,|\vec{x}|} - \frac{x_i(\vec{w} \cdot \vec{x})}{|\vec{w}|\,|\vec{x}|^3} \tag{12}$$

4. Relation to other Normalization Techniques

Under specified conditions, weight normalization (Salimans & Kingma, 2016) is equivalent to normalizing the pre-activation using cosine normalization. Weight normalization re-parameterizes the weights by using new parameters as:

$$\vec{w}_{new} = \frac{g}{|\vec{w}|}\vec{w} \tag{13}$$

Then, the output of hidden unit is computed as:

$$o = f(net_{norm}) = f(\vec{w}_{new} \cdot \vec{x}) = f(\frac{g}{|\vec{w}|}\vec{w} \cdot \vec{x}) \tag{14}$$

where g is a scalar parameter and can be learned by gradient descent. When $g = 1/|\vec{x}|$, Equation 14 is the same as Equation 4 which is used for cosine normalization. In weight normalization, g is learned during training, and is fixed after training accomplishment. While in cosine normalization, $1/|\vec{x}|$ is dynamic and depended on the example. By ignoring the magnitude of $\vec{x}$, cosine normalization bounds pre-activation within a narrower range, and makes the model more robust for different input magnitude.

Batch normalization (Ioffe & Szegedy, 2015) and layer normalization (Ba et al., 2016) use Equation 15 to normalize pre-activation, followed by re-scaling and re-shifting the normalized value (Equation 16).

$$net_{norm} = \frac{net - \mu}{\sigma} = \frac{\vec{w} \cdot \vec{x} - \mu}{\sigma} \tag{15}$$

$$o = f(\gamma\, net_{norm} + \beta) \tag{16}$$

In batch normalization, the mean μ and standard deviation σ are computed over mini-batch examples of training data, while in layer normalization, they are computed over a layer on a single training case. The γ is re-scaling parameter and β is re-shifting parameter, which are learned during training. When μ is 0 and σ is $|\vec{w}|\,|\vec{x}|$, batch or layer normalization is equivalent to cosine normalization. During inference, the μ and σ in batch normalization are calculated from the whole training data set and fixed, while they are varied in layer normalization and cosine normalization as the input changes. We notice that cosine normalization has the scaling term but no shifting term since μ is 0. A possible extension is to use Pearson correlation instead of cosine similarity:

$$net_{norm} = \frac{(\vec{w} - \mu_w) \cdot (\vec{x} - \mu_x)}{|(\vec{w} - \mu_w)|\,|\vec{x} - \mu_x|} \tag{17}$$

where μ_w is the mean of $\vec{w}$ and μ_x is the mean of $\vec{x}$. However, Pearson correlation causes more complicated gradient for training. Another way for adding shifting effect is to use mean-only batch normalization, proposed in (Salimans & Kingma, 2016) for using with weight normalization. Mean-only batch normalization subtracts out the mini-batch mean like with full batch normalization, but does not divide by the mini-batch standard deviation. However, in our experiments, cosine normalization still achieves good performance even without any shifting. Moreover, using ReLU activation function, we found that cosine normalization needs no more re-scaling and re-shifting, thus reduces the parameters of model.

5. Experiments

We compare our cosine normalization with batch, weight and max normalization in fully-connected neural network on the MNIST and 20NEWS GROUP data sets. Additionally, convolutional network with cosine normalization is evaluated on the CIFAR-10 data set.

5.1. MNIST

The MNIST (LeCun et al., 1998) data set consists of 28x28 pixel handwritten digit black and white images. The task is to classify the images into 10 digit classes. There are 60, 000 training images and 10, 000 test images in the MNIST data set. We scaled the pixel values to the [0, 1] range before inputting to our models.

A fully-connected neural network which has the structure of 784-1000-1000-10 was used. The first layer takes the image pixels as input, while the last layer is a 10-class softmax classification layer. There are two hidden layers each with 1000 units. All units of hidden layers use ReLU activation function.

Cosine normalization was compared to other normalization techniques including batch normalization, weight normalization and max normalization. Additionally, ordinary network without any normalization was used as the baseline. In our implements, each unit with batch normalization had two learned parameters for re-shifting and re-scaling. There was no additional parameter for weight normalization, that is to say, the g was set to constant of 1. The max normalization constraines the norm of incoming weight vector to be upper bounded by a fixed constant c, which was set to 2. During optimization, it projected $\vec{w}$ onto the surface of a ball of radius c, whenever $\vec{w}$ went out of it. We did not normalize the last output layer (softmax layer) except in the network with max normalization.

All weights in our experiments were randomly initialized by normal distribution with 0 mean and 0.1 variance. The bias was used just for avoiding the illegality of zero denominator, thus was set to be a very small value 0.0001. We trained the networks using mini-batch gradient descent with 200 epochs since the performances were not improved anymore (in this paper, training epoch refers a cycle that all

training data are used once for training, while training step refers the times of update for parameters). The batch size was set to 100. Different learning rate was applied, and the best result for each model is reported here. The learning rate of baseline was 0.1, cosine normalization 10, batch normalization 5, weight normalization 5, max normalization 1. No dynamic learning rate was used during training.

5.1.1. TEST ERR OF CLASSIFICATION

The results of test err of classification are shown in Figure 2. As we can see, weight normalization has little effect on this task, while the others notably decrease the test err comparing to the baseline. Comprehensively considering convergence speed, divergence and ability of generalization, cosine normalization achieves the best performance. Table 1 shows the mean and variance of test err for the last 50 epochs. Cosine normalization achieves the lowest mean 143.922 and second lowest variance 0.074 of test err. Batch normalization and max normalization cause large variance of test err as training continues. Large fluctuation of batch normalization is caused by the change of statistics on different mini-batch examples. For max normalization, the update of weight is revalued when the length is larger than a fixed constant. That is to say, the procedure of gradient descent is violated.

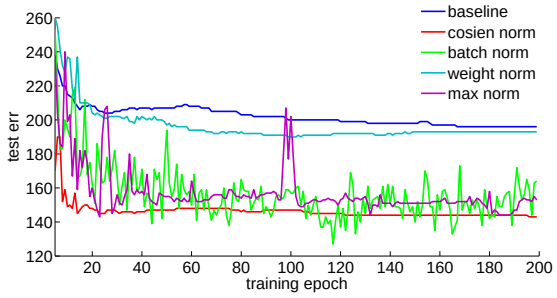


Figure 2. The MNIST test err of networks trained with different normalization techniques, vs. the number of training epoch. All normalization techniques improve the generalization compared to the baseline which does not use any normalization. Cosine normalization rapidly converges to a low test err with smaller divergence compared to other normalization techniques.

5.1.2. ANALYSIS OF PRE-ACTIVATION

To investigate the effect of normalization, we analyzed pre-activation (the input to ReLU activation function) over the course of training. Figure 3 shows how the pre-activation of one typical neuron from the last hidden layer evolves. The pre-activation in baseline network fluctuates the most seriously. In contrast, all normalization techniques reduce the variance of pre-activation. Among them, max normalization makes pre-activation stable most of the time, but there are sharp fluctuations intermittently. That is because, as mentioned above, it alters the direction of gradient descent during training. In the network with cosine normalization, the pre-activation stays the most stable not only from the time dimension, but also the dimension of example. Its 15th, 50th and 85th percentiles almost oeverlap with each other, and fluctuations were barely observed over the course of training. Actually, it does vary over the course of training, but within a very narrow range. Otherwise, the training would not work at all.

Table 1. The mean and variance of test err in last 50 epochs in MNIST experiments . Cosine normalization has the lowest mean 143.922 and second lowest variance 0.074.

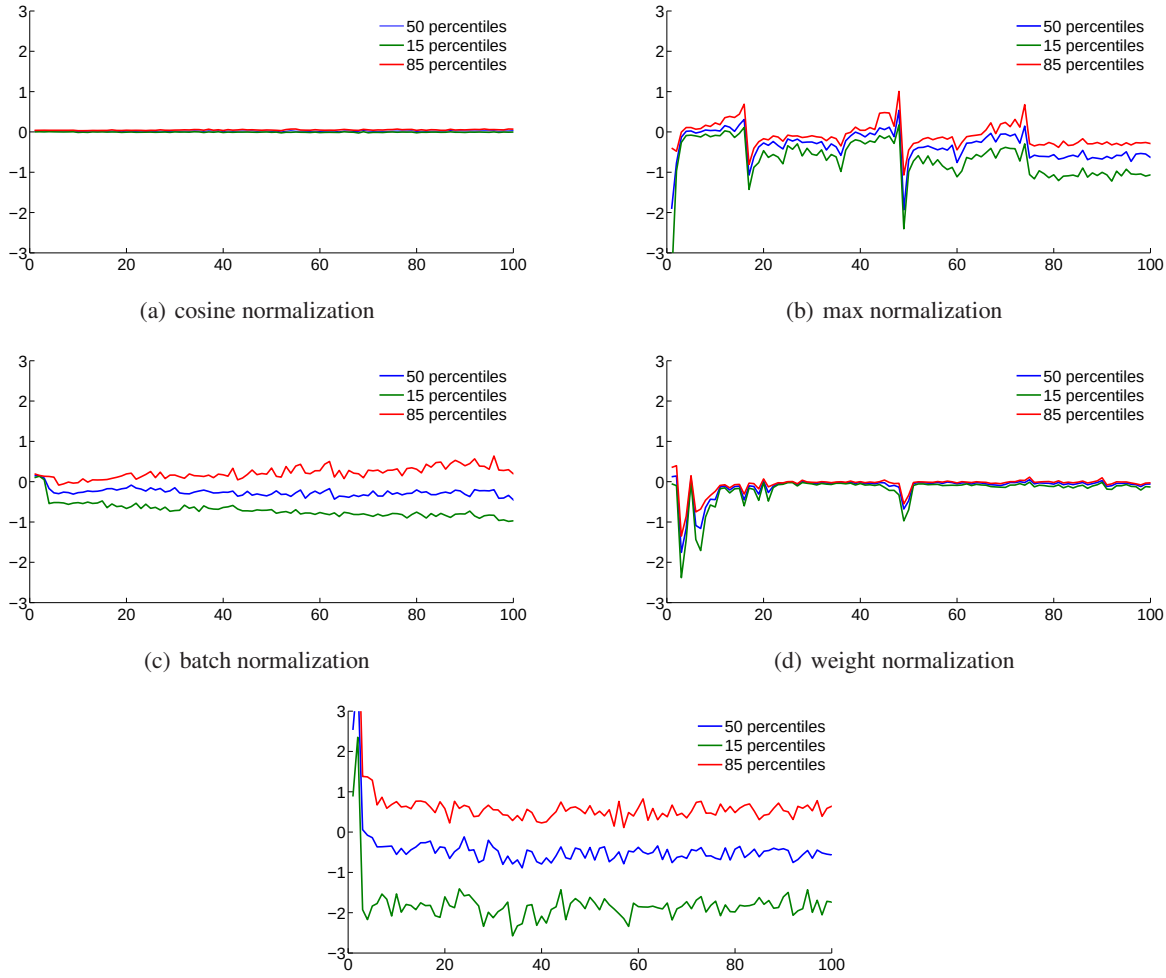
METHODS	MEAN	VARIANCE
BASELINE	196.588	0.887
COSINE NORM	143.922	0.074
BATCH NORM	151.647	85.753
WEIGHT NORM	193	0
MAX NORM	150.588	10.727

5.1.3. INJECTING NOISE

Though noise causes divergence, it prevents the model from overfitting. The influence of noise to cosine normalization was also studied. Two methods that bring noise were evaluated: stochastic gradient descent (SGD) and dropout (Srivastava et al., 2014). In stochastic gradient descent, the gradient is computed by a single example. That is to say, the batch size is 1. Stochastic gradient descent enhances the randomness of training. As shown in Figure 4 and Table 2, stochastic gradient descent decreases the mean of last 50 epoch to 130 with 0 variance. Dropout randomly drops units (along with their connections) from the neural network during training. Combined with dropout, cosine normalization further decreases the mean of last 50 epoch to 113.647, but increases the variance to 32.273. In the experiments, the drop rate is 0.8, 0.5, respectively for the input and hidden layers. The learning rate for baseline with dropout was 1, and cosine normalization with dropout was 10.

5.1.4. INCREASING THE LAYERS

Influence of the number of layers to cosine normalization was also investigated. Same experiment configurations mentioned above were used except the number of layers. The number of layers was varied from two to five, as shown in Figure 5. Using cosine normalization, two layers network reaches minimum test err 155, three layers network 143, four layers network 140, and five layers network 129. Increasing the layers makes lower test err. That implies co-



(a) cosine normalization

(b) max normalization

(c) batch normalization

(d) weight normalization

(e) baseline

Figure 3. The evolution of pre-activation of one typical neuron , over the course of training, shown as {15; 50; 85}th percentiles. Cosine normalization makes the pre-activation within a very narrow range, thus more stable not only from the time dimension, but also the dimension of example.

sine normalization is very suitable for deep learning.

5.2. 20NEWS GROUP

Different normalization techniques were also evaluated on the problem of text classification. We used 20NEWS GROUP data set [1], in which each document is classified into one topic out of 20. The original training set contains 11269 examples, and the test set contains 7505 examples. For convenience of using mini-batch gradient descent, 69 examples in training set and 5 examples in test set were randomly dropped. As a result, there were 11200 training examples and 7500 test examples in our experiments. The words whose document frequency is larger than 5 were used as the input features. There were 21567 feature dimensions finally. Then, the model of Term Frequency-Inverse Document Frequency (TF-IDF) was used to transform the text documents into vectors. After that, each feature is re-scaled to the range of [0, 1].

Similar with MNIST experiments, a fully-connected neural network which has the structure of 21567-1000-1000-20 was used. The last output layer also used softmax. All units of hidden layers used ReLU activation function. All weights were randomly initialized by normal distribution with 0 mean and 0.1 variance. The batch size was set to 100. The constant c in max normalization was set to 2. The learning rate of the baseline, cosine normalization, batch normalization, weight normalization, max normalization is 0.1, 10, 1, 1, 0.1, respectively.

[1]This data set is available here: http://qwone.com/ jason/20Newsgroups/.

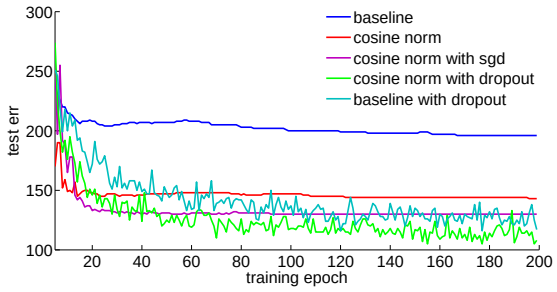


Figure 4. The MNIST test err of networks with and without noise vs. the number of training epoch. By injecting noise, cosine normalization with stochastic gradient descent or dropout further improves the generalization.

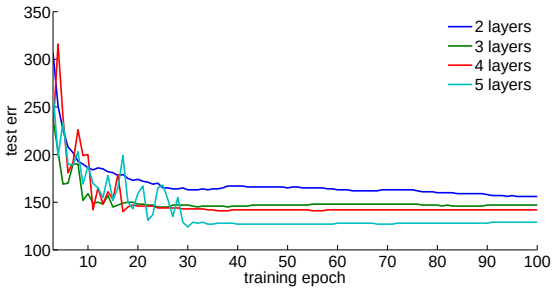


Figure 5. The MNIST test err with different layers vs. the number of training epoch. Increasing the layers, the test err decreases.

Table 2. The mean and variance of test err in last 50 epochs in MNIST experiments. With stochastic gradient descent(SGD), cosine normalization further reduces the mean to 130. Combining the dropout with cosine normalization decreases the mean to 113.647, but increases the variance to 32.273.

METHODS	MEAN	VARIANCE
BASELINE	196.588	0.887
COSINE NORM	143.922	0.074
BASELINE WITH DROPOUT	127.333	29.107
COSINE NORM WITH DROPOUT	113.647	32.273
COSINE NORM WITH SGD	130	0

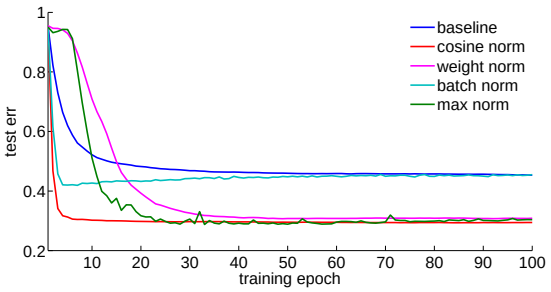


Figure 6. The 20NEWS test err of networks trained with different normalization techniques, vs. the number of training epoch. Batch normalization overfits heavily as training continues. Cosine normalization, weight normalization and max normalization reduce the test err significantly. Cosine normalization converges more quickly than weight normalization and max normalization.

The results are shown in Figure 6 and Table 3. The baseline network performs poorly in this task of high dimensional text classification. It only achieves 0.457 test err. Batch normalization decreases the test err to 0.419 using a few training epochs (7 epochs), but overfits heavily as training continues. Cosine normalization, weight normalization and max normalization achieve close performances. Through analyzing the mean and variance of test err in last 50 epochs, cosine normalization has the lowest mean 0.294 and the lowest variance 0.301×10^{-6}, a bit better than weight normalization and max normalization. Moreover, as we can see from Figure 6, cosine normalization converges more quickly than weight normalization and max normalization.

5.3. CIFAR-10

CIFAR-10 (Krizhevsky & Hinton, 2009) is a data set of natural 32x32 RGB images in 10-classes with 50, 000 images for training and 10, 000 for testing.

The images were cropped to 24 x 24 pixels, and whitened to make the model insensitive to dynamic range. Cosine normalization was evaluated by training the models both with and without data augmentation. To augment data, a series of random distortions were applied: 1) randomly flip the image from left to right. 2) randomly distort the image brightness. 3) randomly distort the image contrast.

Two convolutional layers each followed by a max pooling layer were used. Each convolutional layer has 64 filters. Each convolutional layer has a 5×5 receptive field applied with a stride of 1 pixel, and each max pooling layer pools 3×3 regions at strides of 2 pixels. The last max pooling layer is followed by two fully-connected hidden layers, respectively having 384, 192 hidden units. All units of convolutional layers and fully-connected hidden layers use ReLU activation function. The last output layer is a 10-class softmax classification layer. Cosine normalization was applied both on convolutional layers and fully-connected hidden layers. Ordinary network without normalization was used as baseline.

The exponential moving average of parameters was used during inference. The exponential moving average is calculated by $S_t = \alpha * Y_t + (1 - \alpha) * S_t$, where the α rep-

Table 3. The mean and variance of test err in last 50 epochs in 20NEWS experiments. Cosine normalization has the lowest mean 0.294 and the lowest variance 0.301×10^{-6}.

METHODS	MEAN	VARIANCE ($\times10^{-6}$)
BASELINE	0.457	2.146
COSINE NORM	0.294	0.301
BATCH NORM	0.451	4.749
WEIGHT NORM	0.308	0.312
MAX NORM	0.299	29.598

resents the degree of weighting decrease, Y_t is the value at a time period t. S_t is the value of the average at any time period t. In our experiments, the α was set to 0.9999. All weights were randomly initialized by normal distribution with 0 mean and 0.1 variance. The learning rate of the cosine normalization and the baseline was 10, and 0.01, respectively. The purpose of this experiment was to evaluate the effectiveness of cosine normalization on convolutional neural network. Therefore, the architecture of network we used was simple, and no any other sophisticated technique was used.

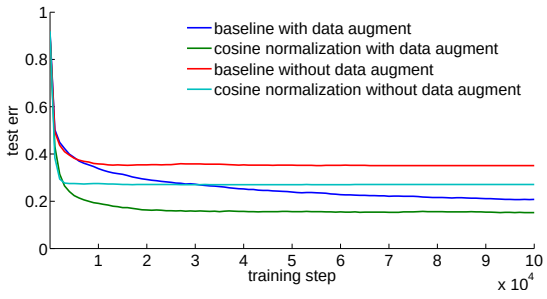


Figure 7. The CIFAR10 test err vs. the number of training step. Cosine normalization significantly reduces the test err of classification both with and without data augmentation, compared to the baseline respectively.

Table 4. For cosine normalization with data augmentation, the number of training steps required to reach the minimum test err of baseline, and the minimum test err achieved by the networks.

METHODS	STEP TO 0.190	MIN TEST ERR
BASELINE	234,000	0.190
COSINE NORM	<11,000	0.152

The results are shown in Figure 7. With fewer steps, cosine normalization significantly reduces the test err of classification both with and without data augmentation. Concretely, as shown in Table 4, the baseline network with data augmentation reaches 0.190 minimum test err, while cosine normalization reaches 0.152 minimum test err (0.038 improvement). It takes 21 times fewer steps than required by baseline network to reach 0.190. As shown in Table 5, without data augmentation, cosine normalization takes 33 times fewer steps than required by baseline network to reach its minimum test err 0.351, and reaches 0.270 minimum test err (0.081 improvement). Regarding speed and accuracy, cosine normalization achieves significantly improvements both with and without data augmentation. Particularly, cosine normalization without data augmentation achieves more improvements than with, compared to the baseline respectively. The reason is that cosine normalization gets rid of the influence of input magnitude, thus weakens the benefits that data augmentation brings. In other words, cosine normalization receives the same effect as augmenting the data by varying input magnitude, but with fewer training data and less training time.

Table 5. For cosine normalization without data augmentation, the number of training steps required to reach the minimum test err of baseline, and the minimum test err achieved by the networks.

METHODS	STEP TO 0.351	MIN TEST ERR
BASELINE	66,000	0.351
COSINE NORM	<2,000	0.270

6. Conclusions and Future Work

In this paper, we proposed a new normalization technique, called cosine normalization, which uses cosine similarity instead of dot product in neural networks. Networks with cosine normalization can be trained using back propagation. Cosine normalization significantly enhances the accuracy of classification and accelerates the training of networks. Moreover, it is more stable than other normalization techniques. Injecting noise or increasing the layers further improves the test accuracy for network with cosine normalization.

Cosine normalization does not introduce any dependencies among the examples. Those dependencies make it difficult to apply batch normalization to recurrent neural networks (RNN) or other noise-sensitive applications such as deep reinforcement learning or generative models, as shown in (Salimans & Kingma, 2016) (Ba et al., 2016). Therefore, applying cosine normalization to those applications could be well worth to try. Meanwhile, we will explore cosine normalization with other types of activation function, e.g. sigmoid. Additionally, the effect of pre-training by k-means or other similarity based methods will be studied.

References

Arpit, Devansh, Zhou, Yingbo, Kota, Bhargava U, and Govindaraju, Venu. Normalization propagation: A parametric technique for removing internal covariate shift in deep networks. *arXiv preprint arXiv:1603.01431*, 2016.

Ba, Jimmy Lei, Kiros, Jamie Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29 (6):82–97, 2012.

Hinton, Geoffrey E and Salakhutdinov, Ruslan R. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Hochreiter, Sepp, Bengio, Yoshua, Frasconi, Paolo, and Schmidhuber, Jürgen. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 448–456, 2015.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Krogh, Anders and Hertz, John A. A simple weight decay can improve generalization. In *NIPS*, volume 4, pp. 950–957, 1991.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Maas, Andrew L, Hannun, Awni Y, and Ng, Andrew Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

Salimans, Tim and Kingma, Diederik P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–901, 2016.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Singhal, Amit. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

Srebro, Nathan and Shraibman, Adi. Rank, trace-norm and max-norm. In *International Conference on Computational Learning Theory*, pp. 545–560. Springer, 2005.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Tan, Pang-Ning et al. *Introduction to data mining*. Pearson Education India, 2006.

Wan, Li, Zeiler, Matthew, Zhang, Sixin, Cun, Yann L, and Fergus, Rob. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1058–1066, 2013.