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 or centered cosine similarity (Pearson Correlation Coefficient) instead of dot product in neural networks, which we call cosine normalization. We compare cosine normalization with batch, weight and layer normalization in fully-connected neural networks as well as convolutional networks on the data sets of MNIST, 20NEWS GROUP, CIFAR-10/100 and SVHN. Experiments show that cosine normalization achieves better performance than other normalization techniques.

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; Arpit et al., 2016; Ren et al., 2016).

[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>.

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 methods are widely used in data mining and machine learning (Tan et al., 2006). 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).

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.

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

To extend, we can use the centered cosine similarity, Pearson Correlation Coefficient (PCC), instead of dot product. By ignoring the magnitude of $\vec{w}$ and $\vec{x}$, the input to activation function is bounded between -1 and 1. 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 layer normalization in fully-connected neural networks on the MNIST and 20NEWS GROUP data sets. Additionally, convolutional networks with different normalization techniques are evaluated on the CIFAR-10/100 and SVHN data sets. Here is a brief summary:

- Cosine normalization achieves lower test error than batch, weight and layer normalization
- Centered cosine normalization (Pearson Correlation Coefficient) further reduces the test error.
- Cosine normalization is more stable than other normalization techniques, specially batch normalization.
- Cosine normalization can accelerate neural networks training as well as other normalization.

1. Background and Motivation

Large variance of neuron in neural network makes the model sensitive to the change of input distribution, thus results 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 thoroughly 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.

2. 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.

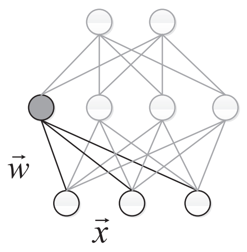

Figure 1. A simple neural network. The output of hidden unit is the nonlinear transform of dot product between input vector and incoming weight vector. That is computed by $f(\vec{w}\cdot\vec{x})$. With cosine normalization, The output of hidden unit is computed by $f(\frac{\vec{w}\cdot\vec{x}}{|\vec{w}||\vec{x}|})$

Empirically, we find that using ReLU activation function $max(0, net_{norm})$, the result of normalization needs no re-scaling and re-shifting. Therefore, there is no additional parameter to be learned or hyper-parameter to be preset. However, when using other activation functions , like sigmoid, tanh, or softmax, the result of normalization should be re-valued to fully utilize the non-linear regime of the functions.

When implementing of cosine normalization in fully-connected nets, we just need divide the norm of incoming weight vector, as well as the norm of input vector. The input vector is the output vector of previous layer. That is to say, the hidden units in the same layer have the same norm of input vector. While in the convolutional nets, the input vector is constrained in a receptive field. Different receptive fields have different norms.

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. Comparing to batch 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. Meanwhile, cosine normalization performs the same computation in forward propagation at training and inference times. 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.

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

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

As pointed in (LeCun et al., 2012), centering the inputs of units can help the training of neural networks. Batch or layer normalization centers the data by subtracting the mean of batch or layer, while mean-only batch normalization can enhance the performance of weight normalization (Salimans & Kingma, 2016). We can use Pearson Correlation Coefficient (PCC), which is centered cosine similarity, to extend cosine normalization:

$$net_{norm} = \frac{(\vec{w} - \mu_w) \cdot (\vec{x} - \mu_x)}{|\vec{w} - \mu_w|\,|\vec{x} - \mu_x|} \tag{13}$$

where μ_w is the mean of $\vec{w}$ and μ_x is the mean of $\vec{x}$.

3. Discussions

3.1. Comparing to weight normalization

Weight normalization (Salimans & Kingma, 2016) re-parameterizes the weights by using new parameters as:

$$\vec{w}_{new} = \frac{g}{|\vec{w}|}\vec{w} \tag{14}$$

Then, the output of hidden unit is computed as:

$$o = f(net_{norm}) = f(\vec{w}_{new} \cdot \vec{x}) = f\left(\frac{g}{|\vec{w}|}\vec{w} \cdot \vec{x}\right) \tag{15}$$

where g is a re-scaling parameter and can be learned by gradient descent. Ignoring the re-scaling parameter g, weight normalization could be seen as partial cosine normalization which only constrains the weights. By additionally dividing the magnitude of $\vec{x}$, cosine normalization bounds pre-activation within a narrower range, thus makes lower variance of neurons.

Moreover, cosine normalization makes the model more robust for different input magnitude. For example, in the forward procedure of the fully-connected network, we have $\vec{x_{l+1}} = f(\vec{w} \cdot \vec{x_l})$. If we scale the $\vec{x_l}$ by a factor λ, then $\vec{x_{l+1}} = f(\vec{w} \cdot (\lambda\vec{x_l}))$. When the activation function f is ReLU, we have $\vec{x_{l+1}} = \lambda f(\vec{w} \cdot \vec{x_l})$. So the λ is linearly transmitted to the last layer. When the last layer is softmax, $\exp(\vec{x})/\sum\exp(\vec{x})$, the output distribution becomes more steep due to the nonlinearity of softmax. For example, if the input vector to softmax is [1, 2], then the output distribution is [0.2689, 0.7311]. When the $\lambda = 10$, after the linearly transmitting, the input vector to softmax becomes [10, 20], and the output distribution becomes [0, 1]. Supposing we want to recognize a handwritten digit, scaling the whole digit by a factor does not bring any valid information. In other words, the output distribution should not be changed. By using cosine normalization, the output distribution can be stable when the input magnitude varies, and it depends only on the angle between the input and the weight.

In the backward procedure of weight normalization, the derivative of net_{norm} with respect to w_i or x_i can be calculated by Equation 16 or Equation 17.

$$\frac{\partial net_{norm}}{\partial w_i} = \frac{x_i}{|\vec{w}|} - \frac{w_i(\vec{w} \cdot \vec{x})}{|\vec{w}|^3} \tag{16}$$

$$\frac{\partial net_{norm}}{\partial x_i} = \frac{w_i}{|\vec{w}|} \tag{17}$$

After scaling the input by λ, the derivative of net_{norm} with respect to w_i becomes Equation 18. Comparing Equation 16 with Equation 18, we can see that the scaling of input also makes the gradient scaling in weight normalization. While in cosine normalization, as shown in Equation 11, the scaling factor λ can be offset by the $|\lambda\vec{x}|$ in the denominator.

$$\begin{aligned}\frac{\partial net_{norm}}{\partial w_i} &= \frac{\lambda x_i}{|\vec{w}|} - \frac{w_i(\vec{w} \cdot \lambda\vec{x})}{|\vec{w}|^3} \\ &= \lambda\left(\frac{x_i}{|\vec{w}|} - \frac{w_i(\vec{w} \cdot \vec{x})}{|\vec{w}|^3}\right)\end{aligned} \tag{18}$$

3.2. Comparing to layer normalization

Layer normalization (Ba et al., 2016) use Equation 19 to normalize pre-activation, followed by re-scaling and re-shifting the normalized value (Equation 20).

$$net_{norm} == \frac{net - \mu}{\sigma} = \frac{\vec{w} \cdot \vec{x} - \mu}{\sigma} \tag{19}$$

$$o = f(\gamma\, net_{norm} + \beta) \tag{20}$$

The mean μ and standard deviation σ 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.

Because $|\vec{x} - \mu_x| = \sqrt{\sum_i (x_i - \mu_x)^2}$, and $\sigma_x = \sqrt{\frac{1}{n}\sum_i (x_i - \mu_x)^2}$, where n is a constant referring to the dimension of $\vec{x}$, The centered cosine normalization (Pearson Correlation Coefficient) can be re-write as:

$$net_{norm} = \frac{(\vec{w} - \mu_w) \cdot (\vec{x} - \mu_x)}{n\sigma_w\sigma_x} \tag{21}$$

Ignoring the constraining of weights, layer normalization is similar with Pearson Correlation Coefficient by constraining the $\vec{x}$ in fully-connected networks.

However, there are three differences between Pearson Correlation Coefficient and layer normalization: 1) Pearson Correlation Coefficient constrains $\vec{w}$ as well as $\vec{x}$, while layer normalization constrains only $\vec{x}$. Thus Pearson Correlation Coefficient is robust to the scaling or shifting of both weight and input. 2) Layer normalization computes the mean and standard deviation before activation and after dot product, while Pearson Correlation Coefficient computes the mean and standard deviation before dot product and after activation. 3) In convolutional networks, Pearson Correlation Coefficient calculates the mean and standard deviation from the receptive fields, while layer normalization calculates the mean and standard deviation from the whole layer. That is to say, different receptive fields have different mean and standard deviation using Pearson Correlation Coefficient, while the same layer has the same mean and standard deviation using layer normalization. As pointed in (Ba et al., 2016), layer normalization works well when all the hidden units in a layer make similar contributions, while the assumption of similar contributions is no longer true for convolutional networks. Pearson Correlation Coefficient just needs the assumption of similar contributions in the receptive fields rather than the whole layer. That is more reasonable for the convolutional networks.

3.3. Similarity metric in neural networks

In machine learning and data mining, there are lots of metrics to measure the similarity or distance between different samples. Among them, cosine similar or the centered cosine (Pearson Correlation Coefficient), is heavily used in many fields, e.g. K-nearest neighbors for classification, K-means for clustering, information retrieval, item or user based recommendation. There are also some neural networks using similarity metrics as the output of neurons, e.g. Radial Basis Function networks (RBF) (Moody & Darken, 1989), Self-Organizing Map (SOM) (Kohonen, 1982). The training of these networks is not using back propagation, and it is hard to build end-to-end deep networks using RBF or SOM. The paper (Lin et al., 2013) argues that the level of abstraction is low with dot product (generalized linear model), thus uses multi-layer perceptron (network in network) to learn convolution filter in convolutional networks. Since dot product is not a decent metric, we may directly try other metrics. As far as we know, it is the first time to use cosine similarity or Pearson Correlation Coefficient as the basic metric to build end-to-end deep network trained by back propagation.

4. Experiments

We compare our cosine normalization and centered cosine normalization (PCC) with batch, weight and layer normalization in fully-connected neural networks on the MNIST and 20NEWS GROUP data sets. Additionally, convolutional networks with different normalization are evaluated on the CIFAR-10, CIFAR-100 and SVHN data sets. We also test the networks without any normalization both for fully-connected and convolutional. The results are much worse than with normalization, thus we focus only on comparison of different normalization techniques.

4.1. Date sets

4.1.1. 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 scale the pixel values to the [0, 1] range before inputting to our models.

4.1.2. 20NEWS GROUP

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

4.1.3. CIFAR-10/100

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. CIFAR-100 is similar with CIFAR-10 but with 100 classes. To augment data, the images are cropped to 24 x 24 pixels, centrally for evaluation or randomly for training. Then, a series of random distortions are applied: 1) randomly flip the image from left to right. 2) randomly distort the image brightness. 3) randomly distort the image contrast. The procedure of augmentation is the same as CIFAR-10 example in Tensorflow (Abadi et al., 2016).

4.1.4. SVHN

The Street View House Numbers (SVHN) (Netzer et al., 2011) dataset includes 604, 388 images (both training set and extra set) and 26, 032 testing images. Similar to MNIST, the goal is to classify the digit centered in each 32x32 RGB image. We augment the data using the same procedure as CIFAR-10/100 mentioned above.

4.2. Protocols

A fully-connected neural network which has two hidden layers is used in experiments of MNIST and 20NEWS GROUP. Each hidden layer has 1000 units. The last layer is the softmax classification layer with 10-class for MNIST, and 20-class for 20NEWS GROUP. To evaluated the convolutional networks, as shown in Table 1, a VGG-like architecture with 12 weighted layers is evaluated in experiments of CIFAR-10/100 and SVHN. Each convolutional layer has 3×3 receptive field with a stride of 1, and each max pool layer has 2×2 regions with a stride of 1.

Table 1. VGG-like architecture.

conv-512
conv-512
conv-512
maxpool
conv-512
conv-512
conv-512
maxpool
conv-512
conv-512
conv-512
maxpool
fully-connected-1000
fully-connected-1000
fully-connected-10/100
soft-max

ReLU activation function is used in the hidden layers. All weights are randomly initialized by truncated normal distribution with 0 mean and 0.1 variance. Mini-batch gradient descent is used to train the networks. The batch size is 100 in experiments of fully-connected nets, and 128 in convolutional nets. In our experiments, we use no re-scaling and re-shifting after normalization for hidden layers. However, for the last layer, we re-scale the normalized values before inputting to softmax. We tried different learning rate for all normalization techniques, and found that cosine normalization can use larger learning rate than other normalization techniques. The learning rate of the cosine normalization, centered cosine normalization (PCC), batch normalization, weight normalization, layer normalization is 10, 10, 1, 1, 1, respectively in our experiments. The exponential moving average of parameters with 0.9999 is used during inference in convolutional networks. No any regularization, dropout, or dynamic learning rate is used. We train the fully-connected nets with 200 epochs and convolutional nets 10^5 step since the performances are 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).

4.3. Results

4.3.1. MNIST

The results of test error for MNIST are shown in Figure 2. As we can see, the converging speeds for different normalization techniques are close. That observation is also true for other data sets we will present next. That is to say, cosine normalization can accelerate the training of networks as well as other normalization. We can also observe that centered cosine normalization (Pearson Correlation Coefficient) and cosine normalization achieve similar test errors, and which are slightly better than layer normalization. Table 2 shows the mean and variance of test error for the last 50 epochs. Centered cosine normalization achieves the lowest mean of test error 1.39%, while cosine and layer normalization achieve 1.40%, 1.43% respectively. Weight normalization has the highest test error comparing to other normalization. Although batch normalization gets lowest test error at some point, it causes large variance of test error as training continues. Large fluctuation of batch normalization is caused by the change of statistics on different mini-batch examples.

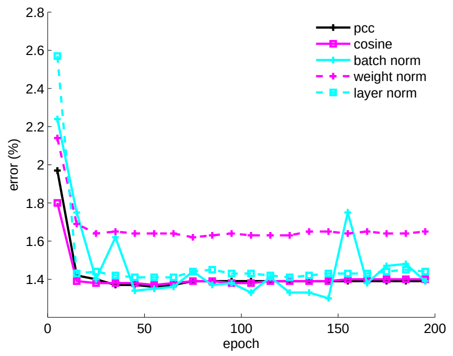

Figure 2. The MNIST test error of different normalization techniques, vs. the number of training epoch.

Table 2. The mean and variance of test error in last 50 epochs in MNIST experiments.

methods	mean %	variance (10^{-3})
centered cosine (PCC)	1.39	0
cosine norm	1.40	0.009
batch norm	1.45	6.740
weight norm	1.65	0.054
layer norm	1.43	0.108

4.3.2. 20NEWS GROUP

The results for 20NEWS GROUP are shown in Figure 3 and Table 3. Centered cosine normalization achieves the lowest test error 29.37%, and cosine normalization achieves the second lowest test error 31.73%. The batch normalization performs poorly in this task of high dimensional text classification. It only achieves 43.94% test error. Weight normalization (33.55%) and layer normalization (33.29%) achieve close performances. Both batch and weight normalization have larger variance of test error than other normalization.

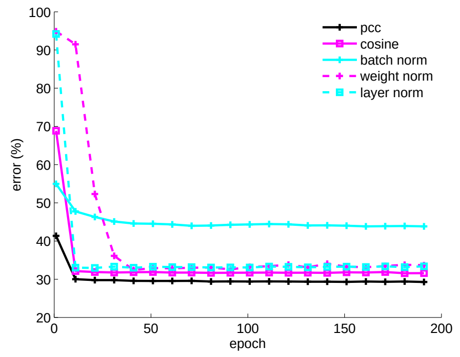

Figure 3. The 20NEWS GROUP test error of different normalization techniques, vs. the number of training epoch.

Table 3. The mean and variance of test error in last 50 epochs in 20NEWS experiments.

methods	mean %	variance (10^{-2})
centered cosine (PCC)	29.37	0.201
cosine norm	31.73	0.633
batch norm	43.94	1.231
weight norm	33.55	4.775
layer norm	33.29	0.556

4.3.3. CIFAR-10

The results for CIFAR-10 are shown in Figure 4 and Table 4. Centered cosine normalization achieves the lowest test error 6.39%, and cosine normalization achieves the second lowest test error 7.33%. The layer normalization also achieves good performance, better than batch normalization, in this experiment. It achieves 7.42% test error. Batch normalization achieves test error 8.08%, and still has larger variance of test error than other normalization. Weight normalization achieves the highest test error 8.55%.

4.3.4. CIFAR-100

The results for CIFAR-100 are shown in Figure 5 and Table 5. Centered cosine normalization achieves the lowest test error 27.49%. Cosine normalization and batch normalization achieve very close performance, 31.02% and 31.01% respectively. But batch normalization have larger variance of test error. Weight normalization achieves the highest test error 37.87%.

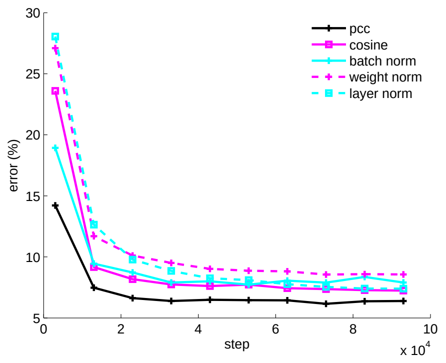

Figure 4. The CIFAR-10 test error of different normalization techniques, vs. the number of training step.

Table 4. The mean and variance of test error in last 10000 step in CIFAR-10 experiments.

methods	mean %	variance (10^{-3})
centered cosine (PCC)	6.39	0.076
cosine norm	7.33	0.036
batch norm	8.08	1.052
weight norm	8.55	0.010
layer norm	7.42	0.008

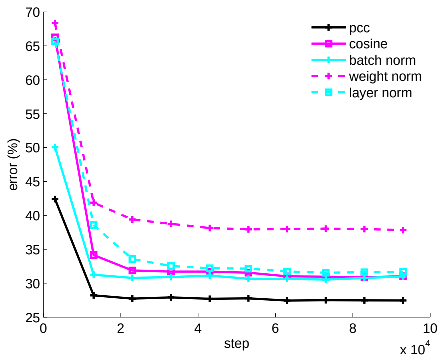

Figure 5. The CIFAR-100 test error of different normalization techniques, vs. the number of training step.

Table 5. The mean and variance of test error in last 10000 step in CIFAR-100 experiments.

methods	mean %	variance (10^{-4})
centered cosine (PCC)	27.49	1.03
cosine norm	31.02	0.43
batch norm	31.01	3.23
weight norm	37.87	1.36
layer norm	31.66	0.22

4.3.5. SVHN

The results for SVHN are shown in Figure 6 and Table 6. Centered cosine normalization achieves the lowest test error 2.22%, and cosine normalization achieves the second lowest test error 2.34%. Batch and layer normalization achieve test error 2.49%, 2.58% respectively. Weight normalization has the highest test error 2.63%.

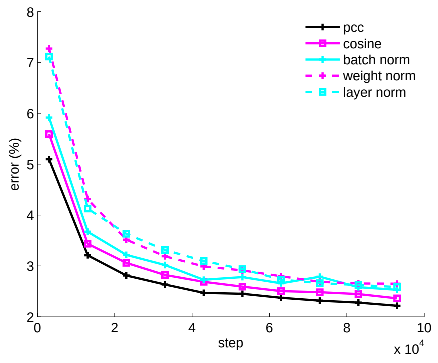

Figure 6. The SVHN test error of different normalization techniques, vs. the number of training step.

Table 6. The mean and variance of test error in last 10000 step in SVHN experiments.

methods	mean %	variance (10^{-4})
centered cosine (PCC)	2.22	0.01
cosine norm	2.34	0.11
batch norm	2.49	0.14
weight norm	2.63	0.03
layer norm	2.58	0.01

5. Conclusions

In this paper, we propose a new normalization technique, called cosine normalization, which uses cosine similarity or centered cosine similarity, Pearson correlation coefficient, instead of dot product in neural networks. Cosine normalization bounds the pre-activation of neuron within a narrower range, thus makes lower variance of neurons. Moreover, cosine normalization makes the model more robust for different input magnitude. Networks with cosine normalization can be trained using back propagation. It does not depend on any statistics on batch or mini-batch examples, and performs the same computation in forward propagation at training and inference times. In convolutional networks, it normalizes the neurons from the receptive fields rather than the same layer or batch size. Cosine normalization is evaluated on different types of network (fully-connected network and convolutional network) and on different data sets (MNIST, 20NEWS GROUP, CIFAR-10/100, SVHN). Experiments show that cosine normalization and centered cosine normalization significantly reduce the test error of classification comparing to batch, weight and layer normalization.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

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.

Kohonen, Teuvo. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43 (1):59–69, 1982.

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.

LeCun, Yann A, Bottou, Léon, Orr, Genevieve B, and Müller, Klaus-Robert. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

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.

Moody, John and Darken, Christian J. Fast learning in networks of locally-tuned processing units. *Neural computation*, 1(2):281–294, 1989.

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.

Netzer, Yuval, Wang, Tao, Coates, Adam, Bissacco, Alessandro, Wu, Bo, and Ng, Andrew Y. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Ren, Mengye, Liao, Renjie, Urtasun, Raquel, Sinz, Fabian H, and Zemel, Richard S. Normalizing the normalizers: Comparing and extending network normalization schemes. *arXiv preprint arXiv:1611.04520*, 2016.

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.