Neurocomputing 328 (2019) 69–74

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Overfitting remedy by sparsifying regularization on fully-connected layers of CNNs

Qi Xu [a], Ming Zhang [a], Zonghua Gu [b,a], Gang Pan [a,*]

[a] College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China
[b] Department of EECS, University of Missouri, Columbia, MO 65211-2060, USA

ARTICLE INFO

Article history:
Received 12 November 2017
Revised 28 February 2018
Accepted 8 March 2018
Available online 18 August 2018

Keywords:
Convolutional neural networks
Fully-connected layers
Overfitting

ABSTRACT

Deep learning, especially Convolutional Neural Networks (CNNs), has been widely applied in many domains. The large number of parameters in a CNN allow it to learn complex features, however, they may tend to hinder generalization by over-fitting training data. Despite many previously proposed regularization methods, over-fitting is still a problem in training a robust CNN. Among many factors that lead to over-fitting, the numerous parameters of fully-connected layers (FCLs) of a typical CNN should be taken into account. This paper proposes the SparseConnect, a simple idea which alleviates over-fitting by sparsifying connections to FCLs. Experimental results on three benchmark datasets MNIST, CIFAR10 and ImageNet show that the SparseConnect outperforms several state-of-the-art regularization methods.

1. Introduction and related work

Convolutional Neural Networks (CNNs) have achieved great successes in many application domains, e.g., image classification and object detection [1,2]. The great performance of CNNs comes along with high model complexity. Modern CNNs usually have tens of millions of parameters. This high complexity allows CNNs to learn complex concepts from training data, but also makes it easy for CNNs to fit in training data, i.e., leading to the over-fitting problem. Over-fitting reduces CNNs' ability to generalize to unseen data. Mitigating over-fitting is a challenge in training robust CNN models.

Many regularization methods have been developed to alleviate the over-fitting problem. Data augumentation [1] enriches training data by generating additional training examples using various label-preserving transformations, such as scaling, zooming, and random cropping of images. DisturbLabel [3] mitigates over-fitting by introducing a small number of incorrectly labeled training examples into the training set. The labels are in turn used to evaluate the loss function. Weight decay reduces complexity of a neural network by imposing an upper bound on weight magnitudes [4]. Dropout [5] regularizes input and hidden layers by discarding the output of a random subset of units during each training iteration. DropConnect [6] mitigates over-fitting by disabling a random subset of connections within the network during each training iteration. Stochastic pooling [7] regularizes pooling layers so that each pooling unit randomly selects an input value from its receptive field as its output value. Liu et al. [8] presented an application of sparse dictionary learning algorithm [9] to maximize sparsity of convolutional kernels of a CNN. Table 1 summarizes the related work on tackling the over-fitting problem on different angles, including training examples, the loss function, weights, nodes, connections, and pooling layers. However, none of these papers has studied the effect of regularizing fully-connected layers (FCLs), although some work (e.g., weight decay) regularize FCLs together with other layers. Our preliminary work has imposed regularization on the FCLs [10]. Based on this basic idea, this paper further investigates what important role the sparsity of FCLs plays. We extend the previous work [10] by analysing the FCLs of CNNs and validating them with more data sets.

In a typical CNN, weights of FCLs make up most of the parameters of the network. VGG-Net [11] has 135M parameters, among which 123M are in the FCLs. For AlexNet [12], 58M out of the 60M parameters are from the FCLs. In OverFeat [13], the FCLs contribute 54.7M of the 70M parameters of the network. In a variant of LeNet-5 [14], which is a small CNN, 40.5K out of the 43K parameters are from the FCLs. The large number of parameters in FCLs may be a contributor to over-fitting and occupy the majority of the total parameters. This motivates us to propose SparseConnect as show in Fig. 1, which mitigates the over-fitting problem of CNNs by sparsifying FCLs. Because of the sparsity of FCLs, the none-zero parameters are decreased which

* Corresponding author.
E-mail addresses: xuqi123@zju.edu.cn (Q. Xu), editing@zju.edu.cn (M. Zhang), gpan@zju.edu.cn (G. Pan).

https://doi.org/10.1016/j.neucom.2018.03.080
0925-2312/

Table 1
Comparison of different regularization methods for CNNs.

Method	Data Aug.	DisturbLables	Weight decay	Dropout	DropConnect	Stochastic pooling	SparseConnect
Site	Training samples	Loss function	Weights	Nodes	Connections	Pooling layers	**FCLs**

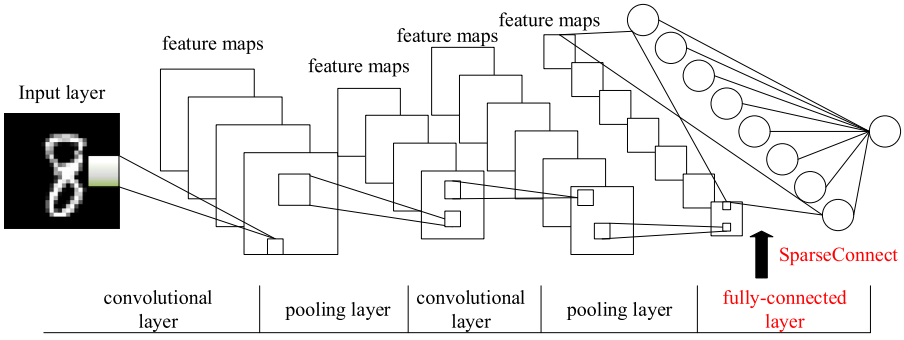

Fig. 1. SparseConnect on a CNN.

is useful for model compression. [15] removes all connections whose weights are lower than a threshold. DSD [16] discards the parameters whose magnitudes are below the threshold which is found by sensitivity-based analysis for each layer and then recovers the pruned connections to make the network dense again in re-training phase. [17] applies hard thresholding over connections and freezes the parameters whose magnitudes are below the threshold and then re-activates the frozen parameters in re-training phase. Compared with these methods, SparseConnect only aims at the FCLs and sparsifies the parameters through the specific loss function. Furthermore, SparseConnect discards the near-zero parameters directly using the threshold which is a hyperparameter not a learned parameter. SpaseConnect prunes the parameters not freezes and re-actives them in re-training phase. Experimental results show that SparseConnect is able to alleviate the over-fitting problem of CNNs and compress the CNNs.

2. SparseConnect

For a typical CNN, according to the role fully-connected layers parameters (FCLs) play, FCLs extract the global features, as for the noisy data, however, not all of the global features are useful. Meanwhile, FCLs make up the majority of the whole parameters. The numerous parameters of FCLs are likely to be a contributor to the over-fitting problem. In this section, we describe our SparseConnect method in detail. SparseConnect mitigates over-fitting by sparsifying the connections to the FCLs.

SparseConnect is used in the training phase of a CNN which is divided into two stages: SparseConnect1 and SparseConnect2. First, during SparseConnect1 we sparsify connections to the FCLs by applying ℓ_1 regularization on the weights of the FCLs. Then, SparseConnect2 follows, further enhancing the sparsity by clipping small weights to zeros.

2.1. Imposing ℓ_1 regularization on FCLs

When applied on a subset of parameters of an optimization problem, ℓ_1 regularization tends to reduce some unimportant parameters in the subset to be very small or even to be zero, and thus sparsifying the parameters in the subset [18]. In order to make connections to FCLs sparse, we apply ℓ_1 regularization on the weights $\mathbf{W_{FCL}}$ of the FCLs during the training phase of a CNN. On the other hand, we empirically find that applying ℓ_1 regularization on convolution filters of a CNN has a significant negative impact on the CNN's learning ability. Unlike previous work [19], which applies ℓ_1 regularization on all layers of a CNN to reduce the memory usage, we apply ℓ_1 regularization only on FCLs in order to avoid the significant negative impact on the CNN's learning ability.

Besides the ℓ_1 regularization on the weights of the FCLs, we also apply weight decay, i.e., ℓ_2 regularization, on all weights $\mathbf{W}$ of the CNN to constrain the weights within a certain range around zero. Both regularization terms are added to the loss function:

$$O(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(\mathbf{x}_i, \mathbf{W})) + \lambda_1 \|\mathbf{W_{FCL}}\|_1 + \lambda_2 \|\mathbf{W}\|_2^2 \quad (1)$$

where $L(y_i, f(\mathbf{x}_i, \mathbf{W}))$ is the loss between the true label y_i and the prediction of the CNN $f(\mathbf{x}_i, \mathbf{W})$ for the ith training example; $\|\cdot\|_1$ and $\|\cdot\|_2^2$ denote ℓ_1 norm and squared ℓ_2 norm, respectively; λ_1 and λ_2 are scalar hyperparameters controlling the strengths of the regularization terms.

During this stage, the CNN is trained with the new loss function in Eq. (1), using a suitable Stochastic Gradient Descent (SGD) algorithm for optimizing $L(y_i, f(\mathbf{x}_i, \mathbf{W}))$ to minimize the loss function. SparseConnect can be used with any SGD algorithm.

2.2. Discarding near-zero parameters

Furthermore, we find that the parameters in trained CNNs usually obey the distribution that is similar to the Gaussian distribution as shown in Fig. 2 and typically concentrated near zero [20], especial on the FCLs. In addition, when used together with a SGD algorithm, the ℓ_1 regularization in Eq. (1) tends to produce many weights that are very near zero, but not strictly equal to zero [19]. These observations motivate us to further enhance sparsification by removing connections very small weight values. Specifically, we continue training the CNN produced by the SparseConnect1 stage with the same loss function and hyperparameters as SparseConnect1. During every iteration of SparseConnect2, we clip a subset of weights whose magnitudes are below a certain threshold to zero. Although [15] has already pruned redundant connections, we discard near-zero parameters during 2 stages and according to a threshold. The threshold is dictated by the *Sparsification Factor (SF)*, a non-negative scalar hyperparameter controlling the degree of sparsity of connections to FCLs. For example, if the sparsification factor is 10% (SF10), 10% of weights of FCLs with the smallest magnitudes would be set to zero.

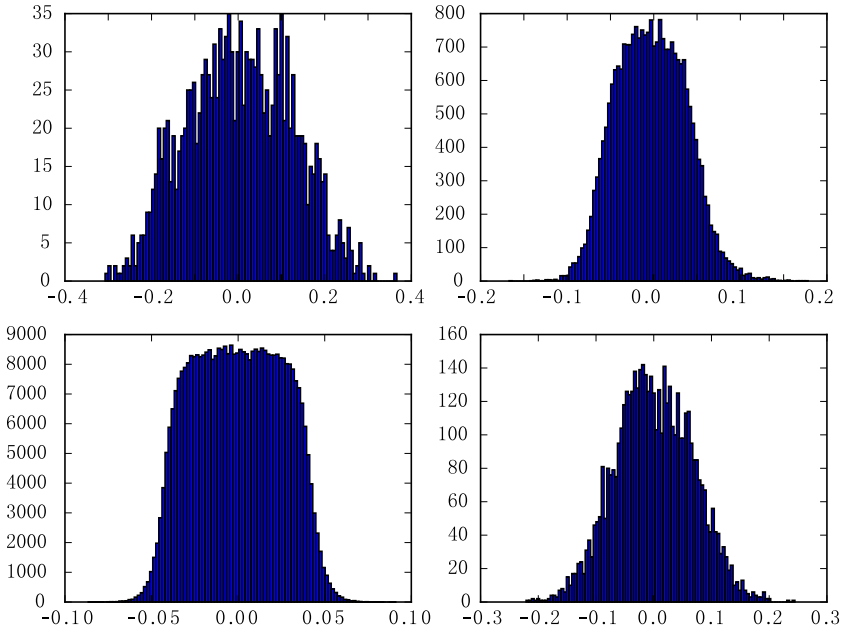

Fig. 2. Weights distributions in each layer (LeNet-5).

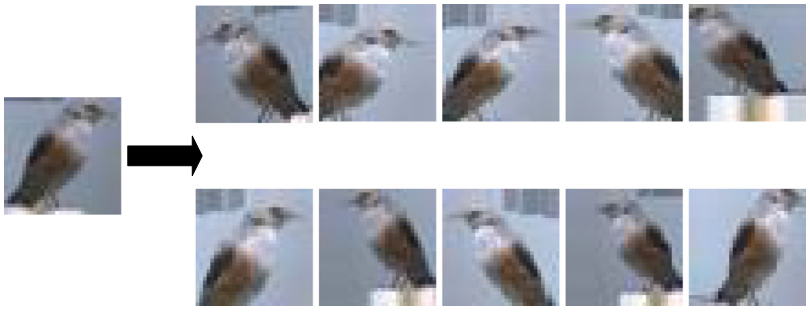

Fig. 3. A data augmentation example.

3. Experimental results

We consider three public datasets: MNIST [14], CIFAR10 [4] and ImageNet (ILSVRC2012) [21].

3.1. Experimental setting

For MNIST and CIFAR10, we train two CNN models as baseline networks. For MNIST, the baseline network is a variant of LeNet-5. The network architecture is 20C5@28x28-P2-50C5@12x12-P2-F500-F10. For CIFAR10, the network architecture of our baseline network is 32C5@32x32-P2-32C5@16x16-P2-64C5@8x8-P2-F10. We will refer to this baseline network as CIFAR10-Net. For ImageNet, we use a variant of AlexNet [12] as our baseline network.

For CIFAR10-Net, we augment the CIFAR10 training set using the image data generator of Keras [22]. For each training example, we generate 10 new images using different combinations of 5 label-preserving transformations, including rotation, translation, shearing, scaling, and horizontal flipping. Fig. 3 shows an example for data augmentation on an image from the CIFAR10 dataset.

The experiments are run on a server equipped with two NVidia GeForce GTX TITAN X GPUs, an Intel Xeon Quad-Core CPU, and 16 GB main memory. We use Tensorflow [23] for training and testing CNNs.

3.2. Performance evaluation of SparseConnect1

For each of the baseline networks LeNet-5 and CIFAR10-Net, we train six CNNs using six different loss functions, each corresponding to a combination of zero or more of the regularization terms in Eq. (1): 1) *CE:* cross-entropy only; 2) *CEAL2:* cross-entropy and ℓ_2 regularization on all layers; 3) *CEAL1:* cross-entropy and ℓ_1 regularization on all layers; 4) *CECL1:* cross-entropy and ℓ_1 regularization on convolutional layers; 5) *CEFL1:* cross-entropy and ℓ_1 regularization on fully-connected layers; 6) *SC1:* the loss function defined in Eq. (1). Each network is trained using stochastic gradient descent with a batch size of 100 for 100,000 iterations.

As shown in Figs. 4 and5, for each of the 12 CNNs, the accuracy on the training set is improved during training. For both LeNet-5 and CIFAR10-Net, the network trained without any regularization term achieves the best training accuracy. This is expected since the network trained without any regularization term has the most freedom to fit its training data. Training and test accuracies of all the 12 CNN models are shown in Table 2. On both MNIST and CIFAR10, networks trained with SC1 achieves significantly better test accuracies than the baseline networks trained without any regularization term. To see how much each of the two regularization terms in Eq. (1) contributes to the improvement, we trained network models with either CEAL2 or CEFL1. For both MNIST and CIFAR10, the network trained with CEFL1 achieves a significantly

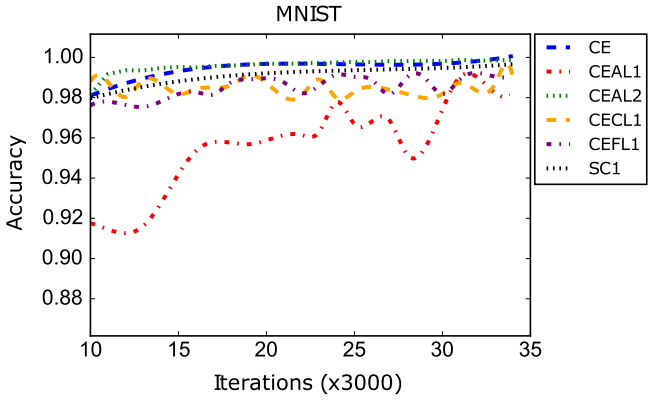

Fig. 4. Training accuracy curves of LeNet-5 with various loss functions during SparseConnect1.

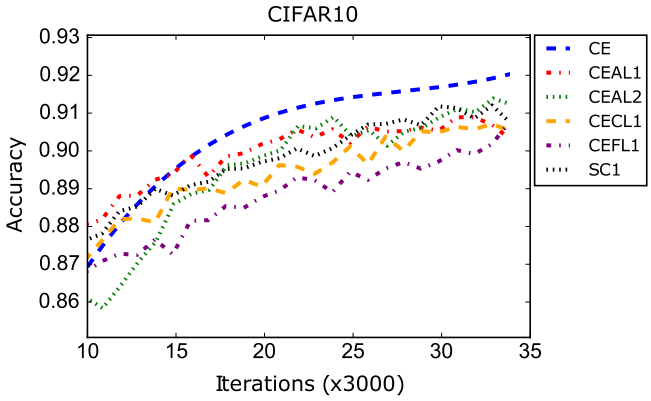

Fig. 5. Training accuracy curves of CIFAR10-Net with various loss functions during SparseConnect1.

Table 2
Training and test accuracy (%) of LeNet-5 and CIFAR10-Net with various functions with SparseConnect1.

		CE	CEAL1	CEAL2	CECL1	CEFL1	SC1
MNIST	Training	**99.53**	98.24	99.21	98.32	98.77	98.89
	Test	98.24	99.21	98.45	98.11	99.35	**99.56**
CIFAR10	Training	**91.82**	90.32	91.23	90.54	90.77	90.76
	Test	91.63	91.99	92.03	91.23	92.31	**92.35**

better test accuracy than the baseline network, but still performs worse than the network trained with SC1. This suggests that ℓ_1 regularization on weights of FCLs is a very effective regularization method to mitigate over-fitting, and confirms our conjecture that the numerous parameters of FCLs are a contributor to over-fitting. To our surprise, even if the FCL of CIFAR10-Net contributes only a small portion (around 11%) of parameters, it is still a contributor to over-fitting. Meanwhile, it can be seen that CEAL2 is also very effective against over-fitting.

By comparing CECL1 and CE, it can be seen that ℓ_1 regularization on convolution filters has a significant negative impact on both the training and test accuracies. This suggests that ℓ_1 regularization is not compatible with convolution filters. Furthermore, for both datasets, despite the negative impact of applying ℓ_1 regularization on convolution filters, the network trained with CEAL1 achieves better test accuracy than the network trained with CECL1. It seems the positive impact of applying ℓ_1 regularization on weights of FCLs has compensated for the negative impact of applying ℓ_1 regularization on the convolution filters.

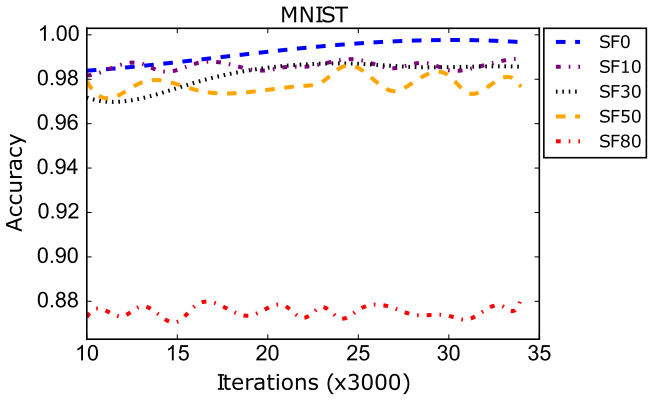

Fig. 6. Training accuracy curves of LeNet-5 during SparseConnect2.

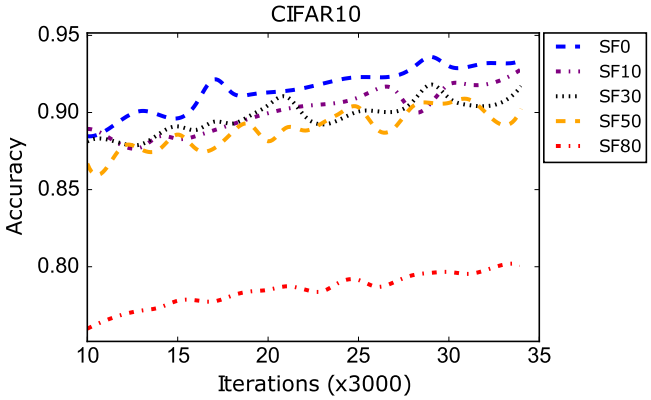

Fig. 7. Training accuracy curves of CIFAR10-Net during SparseConnect2.

Table 3
Training and test accuracy (%) of LeNet-5 and CIFAR10-Net with SparseConnect2.

		SF0	SF10	SF30	SF50	SF80
MNIST	Training	**99.75**	99.03	98.79	98.23	87.53
	Test	98.14	99.22	**99.77**	98.56	86.65
CIFAR10	Training	**92.05**	91.96	91.74	91.03	80.77
	Test	90.79	91.06	**92.53**	90.72	80.05

3.3. Performance evaluation of SparseConnect2

In order to further enhance sparsification of FCLs, we continue to train the networks produced by the SparseConnect1 stage during the SparseConnect2 stage. For each of the networks, we train five networks with five different sparsification factors: 0%, 10%, 30%, 50%, and 80%.

As shown in Figs. 6 and7, for both datasets, the network model trained with a zero sparsification factor (SF0) achieves the best accuracy in general on the training set. This is expected since the network model trained with a zero sparsification factor has the most freedom to fit the training set.

From Table 3, we observe that, for both datasets, the test accuracy increases with the increasing sparsification factor before the sparsification factor reaches a suitable value around 30%, when the best test accuracy is achieved. Then, the test accuracy decreases as the sparsification factor increases further. This observation clearly suggests that SparseConnect2 with a suitable sparsification factor further improves the network's generalization capability.

3.4. Performance evaluation on ImageNet

We evaluate SparseConnect on the ImageNet dataset, which has 1000 object categories. The dataset consists of 3 subsets: a training

Table 4
Test accuracy (%) of LeNet-5 and CIFAR10-Net with different regularization methods.

	No reg.	Dropout	DisturbLabel	SC1	SC2 (SF30)
MNIST	98.24	99.52	99.37	99.56	**99.77**
CIFAR10	91.63	92.24	92.16	92.35	**92.53**

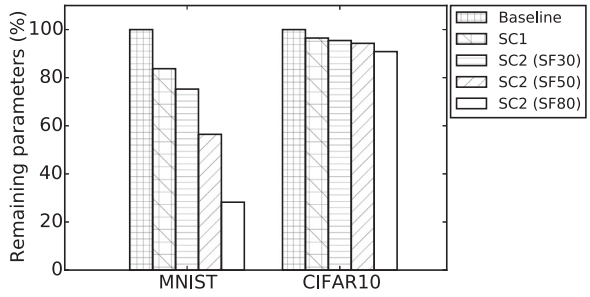

Fig. 8. Percentages of remaining parameters of LeNet-5 and CIFAR10-Net with SparseConnect.

set of 1.3M examples, a validation set of 50K examples and a test set with 150K examples [21].

For this experiment, we use BAIR AlexNet in Caffe Model Zoo [24], which is trained using Dropout with a dropout rate of 0.5, as our baseline network. For comparison, we train two variants of the AlexNet without Dropout. We train AlexNet through the SparseConnect1 stage to obtain our first variant of AlexNet. Then, the first variant is further trained through SparseConnect2 with a sparsification factor of 30% to produce the second variant.

Dropout achieves a top-1 accuracy of 56.9% and a top-5 accuracy of 81.1% on the validation set. SparseConnect1 slightly outperforms Dropout, achieving a top-1 accuracy of 57.01% and a top-5 accuracy of 81.17%. SparseConnect2 in turn slightly outperforms SparseConnect1, achieving a top-1 accuracy of 57.05% and a top-5 accuracy of 81.24%. Although the test accuracies are not significantly improved, SparseConnect with a sparsification factor of 30% reduces the amount of parameters of the baseline network from 60M to 31M.

3.5. Comparison with state-of-the-art regularization methods

We compare SparseConnect with two previously proposed state-of-the-art regularization methods: Dropout and DisturbLabel. For each of LeNet-5 and CIFAR10-Net networks, five networks are trained with different regularization methods, including: 1) *No reg.:* No regularization; 2) *Dropout*: Dropout; 3) *DisturbLabel:* DisturbLabel; 4) *SC1:* through the SparseConnect1 stage only; 5) *SC2 (SF30):* through both SparseConnect1 and SparseConnect2 with a sparsification factor of 30%.

As shown in Table 4, our SparseConnect outperforms Dropout and DisturbLabel on both datasets, achieving a test accuracy of 99.77% on MNIST and 92.53% on CIFAR10, which are significantly better than the test accuracies of Dropout and DisturbLabel.

3.6. SparseConnect as model compression

SparseConnect is also useful for model compression. Fig. 8 shows percentages of remaining parameters in the baseline networks LeNet-5 and CIFAR10-Net, the networks trained by SparseConnect1 (SC1), and the networks trained by SparseConnect2 with various sparsification factors (SC2 (SFxx)). Note that for each of the two datasets, the percentages are normalized so that the percentage of remaining parameters for the baseline network is 100%. After trained by SparseConnect1, the networks may have many tiny weights whose magnitudes are very close to zero, but not strictly equal to zero. Thus, for networks trained through SparseConnect1 only, we deem weights with a magnitude above 10^{-8} as "remaining". As for networks trained by SparseConnect2, only non-zero weights are deemed as "remaining". From Fig. 8, it can be seen that, for each of the datasets, the network trained by only SparseConnect1 has fewer parameters than the baseline network, and the network trained by SparseConnect2 has even fewer parameters. Among the networks trained by SparseConnect2, the number of remaining parameters decreases with the increasing sparsification factor. The compression rate for LeNet-5 is higher than that for CIFAR10-Net, since the FCLs of LeNet-5 contribute a larger proportion of parameters than the FCL of CIFAR10-Net.

4. Conclusions and future work

In this paper, we propose a novel method, called SparseConnect, to regularize fully-connected layers by sparsifying their incoming connections. It is based on the observation that numerous parameters of fully-connected layers of CNNs are a contributor to the over-fitting problem of CNNs. The proposed method is evaluated with experiments on three popular benchmark datasets. The experimental results show that the SparseConnect is effective against the over-fitting problem.

In future work, we will further investigate how to combine SparseConnect with other regularization algorithms. Furthermore, the basic idea of the SparseConnect may be extended to other types of neural networks, such as Deep Belief Networks [25], Auto-Encoders [26], Spiking Neural Networks [27,28] and miSFM [29], as well as modeling biological neural signals for cyborg intelligence and brain informatics [30,31].

Acknowledgment

This work was supported by grants from the National Key Research and Development Program of China (2017YFB1002503), the National Natural Science Foundation of China (No. 61772460), and Zhejiang Provincial Natural Science Foundation of China (LR15F020001).

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: International Conference on Neural Information Processing Systems, 2012, pp. 1097–1105.
[2] Y. Zhang, K. Sohn, R. Villegas, G. Pan, H. Lee, Improving object detection with deep convolutional networks via bayesian optimization and structured prediction, in: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 249–258.
[3] L. Xie, J. Wang, Z. Wei, M. Wang, Q. Tian, DisturbLabel: regularizing CNN on the loss layer, in: Computer Vision and Pattern Recognition, 2016, pp. 4753–4762.
[4] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, Department of Computer Science, University of TorontoMaster thesis.
[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[6] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, R. Fergus, Regularization of neural networks using DropConnect, in: International Conference on Machine Learning, 2013, pp. 1058–1066.
[7] M. Zeiler, R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, in: International Conference on Learning Representation, 2013.
[8] B. Liu, M. Wang, H. Foroosh, M. Tappen, M. Pensky, Sparse convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 806–814.
[9] J. Mairal, F. Bach, J. Ponce, G. Sapiro, Online dictionary learning for sparse coding, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 689–696.
[10] Q. Xu, G. Pan, SparseConnect: regularizing CNNs on fully-connected layers, Electron. Lett. 53 (18) (2017) 1246–1248.
[11] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representation, 2015.
[12] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: International Conference on Neural Information Processing Systems, 2012, pp. 1097–1105.

[13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, Overfeat: integrated recognition, localization and detection using convolutional networks, CoRR (2013). arXiv: 1312.6229.
[14] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[15] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural networks, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.
[16] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, W.J. Dally, DSD: regularizing deep neural networks with dense-sparse-dense training flow, CoRR arXiv:1607.04381.
[17] X. Jin, X. Yuan, J. Feng, S. Yan, Training skinny deep neural networks with iterative hard thresholding methods, CoRR arXiv:1607.05423.
[18] R. Tibshirani, Regression shrinkage and selection via the Lasso: a retrospective, J. R. Stat. Soc.: Ser. B (Stat. Methodol.) 73 (3) (2011) 273–282.
[19] M.D. Collins, P. Kohli, Memory bounded deep convolutional networks, CoRR (2014). arXiv: 1412.1442.
[20] D. Miyashita, E.H. Lee, B. Murmann, Convolutional neural networks using logarithmic data representation, CoRR arXiv:abs/1603.01025.
[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.
[22] F. Chollet, https://keras.io/ (2015).
[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I.J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D.G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P.A. Tucker, V. Vanhoucke, V. Vasudevan, F.B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: large-scale machine learning on heterogeneous distributed systems, CoRR arXiv:abs/1603.04467.
[24] Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Caffe: convolutional architecture for fast feature embedding, in: Proceedings of the 22nd ACM International Conference on Multimedia, 2014, pp. 675–678.
[25] G.E. Hinton, S. Osindero, Y.W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[26] P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol, Extracting and composing robust features with denoising Autoencoders, in: Proceedings of the 25th International Conference on Machine learning, 2008, pp. 1096–1103.
[27] W. Maass, Networks of spiking neurons: the third generation of neural network models, Neural Netw. 10 (9) (1997) 1659–1671.
[28] D. Ma, J. Shen, Z. Gu, M. Zhang, X. Zhu, X. Xu, Q. Xu, Y. Shen, G. Pan, Darwin: a neuromorphic hardware co-processor based on spiking neural networks, J. Syst. Archit. 77 (2017) 43–51.
[29] M. Xu, Y. Wu, P. Lv, H. Jiang, M. Luo, Y. Ye, miSFM: on combination of mutual information and social force model towards simulating crowd evacuation, Neurocomputing 168 (2015) 529–537.
[30] D. Xing, C. Qian, H. Li, S. Zhang, Q. Zhang, Y. Hao, X. Zheng, Z. Wu, Y. Wang, G. Pan, Predicting spike trains from PMd to M1 using discrete time rescaling targeted GLM, IEEE Trans. Cognit. Dev. Syst. 10 (2) (2017) 194–204.
[31] Z. Wu, Y. Zhou, Z. Shi, C. Zhang, G. Li, X. Zheng, N. Zheng, G. Pan, Cyborg intelligence: Recent progresses and future directions, IEEE Intell. Syst. 31 (6) (2016) 44–50.

Qi Xu received his Bachelor's degree in Software Engineering from Zhejiang University of Technology in 2015, and is currently a Ph.D student in CCNT Lab, Zhejiang University. His research interests include Computer Vision, Neuromorphic Computing and Cyborg Intelligence.

Ming Zhang received his Bachelor's degree in software engineering from Zhejiang University in 2010, and is currently a Ph.D. student at Zhejiang University.

Zonghua Gu received his Ph.D. degree in Computer Science and Engineering from the University of Michigan at Ann Arbor in 2004. He is currently an associate professor in the College of Computer Science, Zhejiang University. His research area is real-time and embedded systems.

Gang Pan received the B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. From 2007 to 2008, he was with the University of California, Los Angeles, CA, USA, as a Visiting Scholar. He is the author of more than 100 refereed papers. His research interests include pervasive computing, computer vision, artificial intelligence, brain-machine interfaces, and neural computation. He currently serves as a member of the Editorial Board of IEEE Systems Journal, and ACM IMWUT.