Embedding Compressive Layers in Deep Neural Networks

Chiho Choi Sangpil Kim Joon Hee Choi Karthik Ramani
Purdue University
West Lafayette, IN 47907, USA
{chihochoi, kim2030, choi240, ramani}@purdue.edu

Abstract

We propose a compressive technique for deep neural networks to learn more informative representations of data. Our approach is flexible to add more layers and go deeper in the deep learning architecture while keeping the number of parameters the same.

1. Introduction

There has been a great effort to solve the memory and computational efficiency problem of a deep neural network in the literature. The main stream of this category is focused on quantizing the network parameters into bins. In [2], the weights are first converted to the frequency domain using a discrete cosine transform and then quantized into hash buckets to group the frequency parameters. By sharing a single value for the parameters in the same bucket, the size of the model can be reduced. However, the compressed model may significantly lose accuracy mainly because of hashing and training procedure [6]. Also, the weights are compressed using the vector quantization techniques in [3], but this method may result in the reduction of predictive performance.

In contrast to these works, [7] compresses the network parameters using random matrix projection without dropping accuracy. In the training process, they try to learn the weight matrix which is split into a set of matrices based on the sparsity of the Johnson-Lindenstrauss (JL) transform. Our compressive framework is also focuses on the JL transform. However, we aim to embed more layers and design a deeper network while preserving the original network structure by compressing the embedded layers. We do not explicitly reduce the network parameters but do compress the network layers, and therefore we are able to keep the same amount of parameters. The embedded layers only appear at training time to learn more informative representations by backpropagation based on the fixed JL transform, instead of updating its entries [7]. In this way, we effectively train a network model using deeper layers with the reduced computational cost at test time.

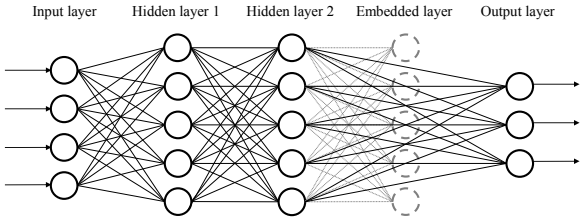

Figure 1. The proposed compressive technique. At training time, the network parameters learned from the embedded layers are compressed using ramdom projection to preserve the number of parameters the same.

2. Compressive Neural Network

In random projection, the size or dimension of the random variables can be effectively reduced from a size m to k where $k \ll m$ by mapping high dimensional data into a lower dimensional space using a random matrix $\mathbf{J} \in \mathbb{R}^{k \times m}$. Let $\mathbf{E} \in \mathbb{R}^{m \times n}$ be a set of n m-dimensional data. Then, the projection to a lower dimensional vector space can be $\mathbf{L} = \mathbf{JE}$, where $\mathbf{L} \in \mathbb{R}^{k \times n}$ is a lower dimensional matrix.

2.1. Johnson-Lindenstrauss lemma for dimensionality reduction

The JL lemma and its variants have shown a linear mapping $f : \mathbb{R}^m \rightarrow \mathbb{R}^k$ while preserving the pairwise distances of the data in the Euclidean space. Our compressive technique is highly inspired by the fact that the distance of a pair $\mathbf{e}_i, \mathbf{e}_j \in \mathbf{E}$ can be preserved with high probability and small distortion ϵ after projecting on a lower dimensional space,

$$\sqrt{\frac{k}{m}}\|\mathbf{e}_i - \mathbf{e}_j\|_2(1-\epsilon) \leq \|f(\mathbf{e}_i) - f(\mathbf{e}_j)\|_2 \leq \sqrt{\frac{k}{m}}\|\mathbf{e}_i - \mathbf{e}_j\|_2(1+\epsilon). \quad (1)$$

We extend this insight to embed a compressive layer that ouputs higher dimensional neurons and then compress the embedded layer to preserve the identical network architecture to the original network. Assume the original network reduces k neurons to n neurons from the i-th hidden layer

with a weight matrix $\mathrm{W}_i \in \mathbb{R}^{k\times n}$. Also assume that we are able to achieve better performance by embedding the hidden layers with two weight matrices $\mathrm{W}_i^{\mathrm{e}_1} \in \mathbb{R}^{k\times l}$ and $\mathrm{W}_i^{\mathrm{e}_2} \in \mathbb{R}^{l\times n}$, where $l > n$. Then, the computational cost increases from $\mathcal{O}(kn)$ to $\mathcal{O}(kl+ln)$ at test time. In practice, however, we might not be able to design such network from the embedded systems and platforms as they run with limited computational power and memory. Thus, we compress the embedded layer with $\mathrm{W}_i^{\mathrm{e}_1}$ into a matrix $\mathrm{W}_i^{\mathrm{c}} \in \mathbb{R}^{k\times n}$ (the same size as W_i) using the JL lemma to reduce the computational cost to $\mathcal{O}(kn)$ while achieving performance higher than the original network with W_i. The detailed process is as follows: (i) reshape $\mathrm{W}_i^{\mathrm{e}_1}$ into a matrix $\mathbf{E} \in \mathbb{R}^{m\times n}$ where m is an integer equal to $\frac{l}{n} \times k$; and (ii) calculate $\mathrm{W}_i^{\mathrm{c}}$ = **JE** where **J** is the fixed transform matrix of size $k \times m$ where the pairwise distances of **E** are preserved in $\mathrm{W}_i^{\mathrm{c}}$ with high probability based on the JL lemma (Eqn. 1).

Although the storage and computational costs of this process during training are both $\mathcal{O}(km)$ with the JL transform, they can be further reduced, in particular, to $\mathcal{O}(m \log m + \epsilon^{-3} \log^2 n)$ for computational time by the fast Johnson-Lindenstrauss transform (FJLT) [1]. The FJLT picks three real-valued matrices **PHD** of the form **J** = **PHD**, where $\mathbf{P} \in \mathbb{R}^{k\times m}$ is a sparse matrix with Gaussian random numbers, $\mathbf{H} \in \mathbb{R}^{m\times m}$ is a Hadamard matrix, and **D** $\in \mathbb{R}^{m\times m}$ is a diagonal matrix with random ± 1 entries.

2.2. Backpropagation of compressive networks

Our compressive method learns the weight matrix $\mathbf{W}_i^{\mathrm{e}_1}$ and a bias term $\mathbf{b}_i \in \mathbb{R}^{1\times n}$ during backpropagation. Let $C(\mathbf{W}, \mathbf{b})$ be a cost function (*i.e.* average sum of squared error) for the i-th layer. The bias is first updated using the gradient of C with respect to $\mathbf{b}_i$,

$$\mathbf{b}_i \leftarrow \mathbf{b}_i - \alpha \nabla_{\mathbf{b}_i} C(\mathbf{W}, \mathbf{b}), \tag{2}$$

where α is the learning rate. In contrast, we cannot directly update $\mathbf{W}_i^{\mathrm{e}_1}$ from the gradient of C because of its dimensionality. Instead, we find the low dimensional gradient of C with respect to $\mathbf{W}_i^{c}$ and then update the matrix $\mathbf{W}_i^{\mathrm{e}_1}$ as:

$$\mathbf{W}_i^{\mathrm{e}_1} \leftarrow \mathbf{W}_i^{\mathrm{e}_1} - \alpha \mathbf{J}^T \nabla_{\mathbf{W}_i^{\mathrm{c}}} C(\mathbf{W}, \mathbf{b}). \tag{3}$$

We note that our technique guarantees the computational cost $\mathcal{O}(kn)$ with $\mathrm{W}_i^{\mathrm{c}}$ at test time, despite the embedded layers are added into the architecture. The time complexity can be further decreased by incorporating the parameter compression methods such as [5, 7] into our approach.

3. Experiment on MNIST dataset

We evaluate the proposed technique using an MNIST dataset for proof of concept. We train a reference model (*Ref*) from scratch using the LeNet configuration implemented on the Caffe framework [4]. The network is comprised of two convolutional layers (Conv) with a max pooling layer and two subsequent fully connected layers (FC). The error rate of the reference model is 0.87 %. For comparison, we train two sets of network models using different configurations; the models with a label *JL* where we embed additional layers and reduce dimensionality by compression, and the prefix *D* denotes whether we use the dropout technique to prevent overfitting. The experimental results are shown in Table 1. Note that all of network models preserve the same computational complexity with the *Ref* model at test time. The best performance is achieved from the model *JL-FC1* (the number of output neurons is 6000) with an error rate of 0.75 % without using dropout. It demonstrates that classification with the compressed parameters effects on extracting more expressive representations of data, and thus the proposed compressive network is able to achieve a fundamental goal of designing the deeper network architecture. Interestingly, we observe that the higher accuracy may not be accomplished by simply increasing the size of the embedded feature map or the number of layers, as the network architecture experiences overfitting (see *JL-C2-FC1*). Thus, we use the dropout layers to regularize the network in the following experiments. With an aid of the dropout layers, the model *D-JL-C2* (the number of kernels is 250) shows a minimum error rate of 0.51 %. It validates the rationale of our compressive technique to further improve the performance while preserving the same computational power.

Model	Comp. layer	Output size	Error
Ref	-	-	0.87 %
JL-C2	$\mathrm{Conv}_2^{e_1}$	250	0.85 %
JL-FC1	$\mathrm{FC}_1^{e_1}$	6000	**0.75 %**
JL-C2-FC1	$\mathrm{Conv}_2^{e_1}$ & $\mathrm{FC}_1^{e_1}$	250 & 6000	1.02 %
D-Ref	-	-	0.70 %
D-JL-C2	$\mathrm{Conv}_2^{e_1}$	250	**0.51 %**
D-JL-FC1	$\mathrm{FC}_1^{e_1}$	6000	0.64 %
D-JL-C2-FC1	$\mathrm{Conv}_2^{e_1}$ & $\mathrm{FC}_1^{e_1}$	250 & 6000	0.57 %

Table 1. The evaluations using the MNIST dataset. *Ref* is a model trained using the LeNet configuration of Caffe and *JL* is where we apply the proposed compressive (Comp) technique. The prefix *D* shows if we use the dropout technique to avoid overfitting. The output size denotes either the number of kernels (Conv) or the number of outputs (FC) embedded in between layers.

4. Conclusion

We propose a new technique to design a deeper network by compressing the embedded layers. Our main insight is that the high dimensional data can be mapped into the lower dimensional space while preserving the pairwise distances of the elements using the JL transform. The evaluation results validate the efficacy of the proposed approach as we improve the classification accuracy.

References

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 557–563. ACM, 2006. 2

[2] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks. *arXiv preprint arXiv:1506.04449*, 2015. 1

[3] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 1

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014. 2

[5] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015. 2

[6] L. Shi, S. Feng, et al. Functional hashing for compressing neural networks. *arXiv preprint arXiv:1605.06560*, 2016. 1

[7] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483, 2015. 1, 2