Learning the Group Structure of Deep Neural Networks with an Expectation Maximization Method

Subin Yi*
T-brain
SK
Seoul, Republic of Korea
yisubin@sktbrain.com

Jaesik Choi
School of Electrical and Computer Engineering
Ulsan National Institute of Science and Technology
Ulsan, Republic of Korea
jaesik@unist.ac.kr

***Abstract*—Many recent deep learning research work use very deep neural networks exploiting huge amount of parameters. It results in the strong expressive power, however, it also brings issues such as overfitting to training data, increasing memory burden and requiring excessive computations. In this paper, we propose an expectation maximization method to learn the group structure of deep neural networks with a group regularization principle to resolve those issues. Our method clusters the neurons in a layer based on how they are connected to the neurons in the next layer using a mixture model and the neurons in the next layer based on which group in the current layer they are most strongly connected to. Our expectation maximization method uses the Gaussian mixture model to keep the most salient connections and remove others to acquire a grouped weight matrix in a block diagonal matrix form. We refine our method further to cluster the kernels of convolutional neural networks (CNNs). We define the representative value of each kernel and build a representative matrix. The matrix is then grouped and the kernels are pruned out based on the group structure of the representative matrix. In experiments, we applied our method to fully-connected networks, 1-dimensional CNNs, and 2-dimensional CNNs and compared with baseline deep neural networks in MNIST, CIFAR-10, and United States groundwater datasets with respect to the number of parameters and classification and regression accuracy. We show that our method can reduce the number of parameters significantly without loss of accuracy and outperform the baseline models.**

***Keywords*-Machine Learning, Deep Learning, Clustering**

I. Introduction

Deep learning has been remarkably successful in many fields of applications such as visual processing, speech recognition, game playing, and others, even outperforming human experts at some problems. The expressive power of the deep neural networks originates from the multiple nonlinear transformations of the network layers. The recent interest in deep learning research has been led to build very deep, therefore large neural networks [1]–[3] to escalate the expressive power of networks. Building deeper neural networks has been shown to be successful in existing work [4], [5]

However, there are drawbacks with the huge number of parameters. Large number of parameters increase not only the expressive power, but also the memory burden and computational cost of both training and testing. It becomes problematic when the model should run in a small chip or time-critical applications, therefore the real world application can be significantly limited. Moreover, they can fit the data better than the small number of parameters at the training time, however, generalize poorly outside the training set. This overfitting problem is one of the common problems of deep neural networks and has been studied in many researches [6]–[8].

A simple remedy for overfitting is to use the reduced number of parameters that can fit the data. In practice, it is shown that there are many redundant and duplicate connections in deep neural networks [9] and the same performance can be achieved using only smaller number of parameters. One way to obtain the smaller network without loss of performance is to build a small network and train it to approximate the larger deep neural network [10], [11]. Another way is to define an importance measurement and remove unnecessary or redundant connections [12]. Also, the number of parameters can be reduced by acquiring exclusive sparsity using ℓ_1-regularization, which achieves the weight sparsity by selectively adopt the weights to optimize the objective function. However, ℓ_1-regularization methods drop parameters only when each parameter is penalized enough to be zero. Also, the pruning methods require additional calculations and searches for measuring the importance and finding the parameters to be eliminated.

In this paper, we propose a method to find group structure of deep neural networks (DNNs) in an end-to-end learning. Our method clusters the input nodes and output nodes of the network layers and removes connections between input and output groups that are weakly connected, leaving only the strongest group-wise connections. Then, the weight matrix is simplified as a block diagonal matrix as shown in Fig. 1.

To group the input nodes, we utilize the expected value of negative log likelihood of Gaussian mixture model (GMM)

*This work was done at the Ulsan National Institute of Science and Technology.

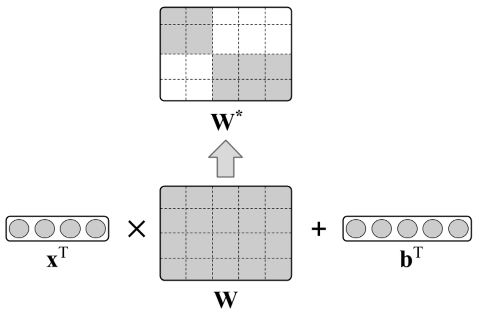


Fig. 1. Group regularization of a network layer. The input and output nodes are grouped into two clusters and weights between the clusters that are the most strongly connected are maintained.

while optimizing DNNs. We devise a gradient descent algorithm to update parameters of GMM so that the mixture model parameters can be trained by the gradient descent updates together with the updates of network parameters in an end-to-end way. We also prove that the devised gradient descent algorithm converges to an optimal point in Section V.

We applied our method to a fully-connected network, 2-D CNNs, and a 3-D CNN and validated using MNIST dataset, groundwater dataset, and CIFAR-10 dataset. We compared our model with base models in terms of the classification and regression accuracy and the number of parameters. In the experimental section, we evaluate our method comparing with other baseline methods and show that our method can significantly reduce the number of parameters without loss of accuracy and even outperform the baseline models.

II. Related Work

Finding an efficient structure with a proper number of parameters is an important issue when building a DNN. One intuitive way is to prune unnecessary redundant parameters. Such pruning algorithms can be categorized into two categories [12]. One is to estimate the sensitivity of the error function and the other is to impose regularization terms to the objective function that rewards the efficient connections. The sensitivity-based methods define the sensitivity of the error function which measures the effect of the parameters on the model error and remove those have least effect. The penalty-based method penalizes unnecessary weights which they are close to zeros during the training. In a more recent research of [13], weak weights were pruned out by simple thresholding, and the network was retrained to fine tune the remaining parameters. Also, [14], [15] took advantage of the inherent redundancy of convolutional layers and fully-connected layers and compressed the neural networks using hash functions. Some works focused on compressing CNNs in particular by pruning out convolution filters [16], [17]. ℓ_1-norm can be used to select unimportant filters [16] or the filters can be pruned out based on their influence on the accuracy [17].

There have been efforts to achieve better pruning results. Reference [18] introduced weight quantization and Huffman coding steps after the pruning procedure and [19] proposed to both prune and splice the network for compressing DNNs to avoid possible inaccurate pruning. Reference [20] used additional compressor networks instead of measuring the importance of parameters and pruning them. They combined the compressor with the original network and built a compressor-critic framework.

Regularization approaches were also used to restrain the networks from growing unnecessarily large. Inspired by the Lasso algorithm [21] the Lasso regularization performs both the regularization and variable selection by bounding the ℓ_1 norm of the weights. Moreover, some of the recent researched introduced the idea of grouping to Lasso regularization. Group Lasso method was proposed in [22], which considers the cases where the variables are structured into K groups and the group memberships are given. The sparse group Lasso [23]–[25] extended group Lasso by adding group sparsity to the group Lasso. Reference [26] also used group structure for a machine learning task efficiently.

Unfortunately, these methods have some drawbacks. ℓ_1-regularization methods or penalty-based methods remove the parameters from the network only when the parameters are reduced frequently enough to become zeros. Also, sensitivity analysis-based pruning algorithms demand additional steps for calculating the sensitivity and finding the parameters to be eliminated.

Our method, similarly with the group Lasso, exploits the group structure of DNNs. However, it can discover the underlying structure of the networks by learning the group structure of the network using a mixture model and select groups to perform compression with an end-to-end algorithm. Moreover, it does not require further learning steps as the group structure learning occurs simultaneously with the parameter learning. Also, parameters are instantly pruned away based on the groups they belong to, which makes the justification for the reason the parameters are chosen more explainable.

III. Motivation

The main idea of this paper is to consider a DNN as a mixture of smaller subgroups connected to each other. For a certain group of neurons to be activated, only some groups of the input layer play important roles while other groups do not contribute much. If we remove the less significant inter-group connections, the original weight matrix can be compressed to a block matrix form as shown in Fig. 1. The goal is to learn the group structure of the layer and restrict the connections between layers based on the groups so that the weight matrix can be transformed to a block matrix form.

To achieve the goal, we group the rows and columns of the weight matrix as in Fig. 2. We first group the the rows of the weight matrix W using GMM assuming that the vectors in the same group are generated from the same Gaussian distribution.

Then we compare the mean vector of the Gaussian distributions of input groups. If the i-th elements of the mean vectors have the greatest value at the j-th group, we put the column i into the cluster j. Each row of the matrix represents the weights from the i-th node to the next layer $H \in \mathbb{R}^M$ where

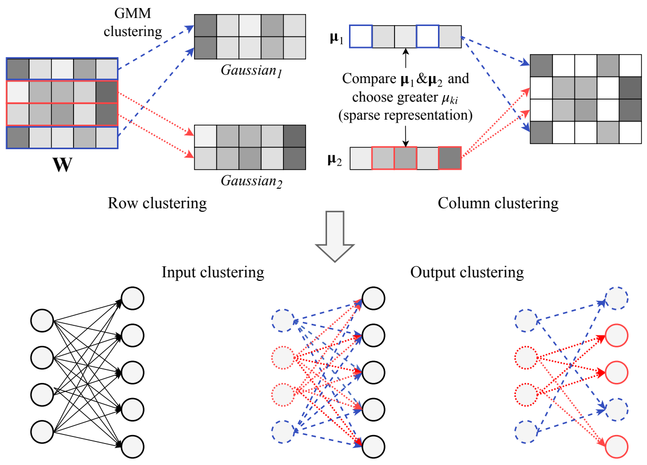


Fig. 2. Group Regularization using EM method.

M is the number of nodes in H, and each column represents the weights from the input layer to each output node. Therefore grouping rows and columns is equivalent to grouping the nodes of the input layer and the next layer based on the distributions of the connections to the next layer or from the current layer.

Fig. 2 describes the process using an example weight matrix $\mathbf{W}$. Comparing the rows of $\mathbf{W}$, the connections to the nodes in the next layer, the first and the fourth nodes are grouped into the $Gaussian_1$ and the second and the third nodes are grouped into the $Gaussian_2$. Then the mean vectors of two Gaussian distributions are compared. The first and the fourth elements of the $Gaussian_1$ are greater than the $Gaussian_2$ and the other elements are smaller than the $Gaussian_2$. So we cluster the first and the fourth nodes in the next layer and remove the connections from the nodes in the $Gaussian_2$ to the first and the fourth nodes of the next layer and those from the nodes in the $Gaussian_1$ to the other output nodes.

This process compares mean vectors and eliminates smaller elements, thereby removes the weaker inter-group connections and acquires a compact sparse representation of the weight matrix as ℓ_1 regularization does. The resulting sparse matrix can be reordered into a block diagonal matrix as in Fig. 1.

IV. Expectation Maximization Algorithm

A. Mixture Model

A mixture model represents the probability of the observations in the overall population by the sum of individual distributions. Using K probabilistic models, a mixture model describes the probability distribution of an observation $\mathbf{x}$ as below:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x}|\boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{\theta}$ is the parameters of the base distributions. The variables π_k's are the mixing weights or the mixing coefficients of this mixture model. It can be formulated with respect to a latent variable $\mathbf{z} = [z_1 ... z_K]^{\mathrm{T}}$. $\mathbf{z}$ is the vector of K binary random variables having a 1-of-K representation so that $z_k \in \{0,1\}, \forall k \in \{1,...,K\}$, and $\sum_k z_k = 1$. Then the mixing coefficients π_k's can be formally defined as $\pi_k = p(z_k = 1)$ satisfying $\sum_{k_1}^{K} \pi_k = 1$ together with $0 \leq \pi_k \leq 1$.

Another way to view a mixture model is to consider each component distribution as one cluster and to separate the data points according to the clusters. From this perspective of view, the variable $\mathbf{z}$ represents the cluster label. To accomplish the clustering, the posterior distribution $p(z_k = 1|\mathbf{x}_i)$ is used, which is the likelihood of $\mathbf{x}_i$ belonging to k-th cluster given the data point. This is called the *responsibility* of k-th cluster for the i-th data point and can be inferred using Bayes' Rule as below:

$$\gamma_{ik} = p(z_k = 1|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_k = 1|\boldsymbol{\theta}) p(\mathbf{x}_i|z_k = 1, \boldsymbol{\theta})}{\sum_{j=1}^{K} p(z_j = 1|\boldsymbol{\theta}) p(\mathbf{x}_i|z_j = 1, \boldsymbol{\theta})} \tag{2}$$

B. Expectation Maximization Algorithm

EM algorithm is an iterative algorithm [27] for computing the maximum likelihood (ML) and maximum a posteriori (MAP) estimates given data with latent variables. For a complete data D, let $\mathbf{x}_1, ..., \mathbf{x}_N$ be the observed variables and $\mathbf{z}_1, ..., \mathbf{z}_K$ be other latent variables where $N >> K$. EM algorithm maximizes the log likelihood of the observed data defined as below where $\boldsymbol{\theta}$ represents the parameters to estimate.

$$\ell(\boldsymbol{\theta}) = \log p(\mathcal{D}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \log \left[\sum_{\mathbf{z}_i} p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) \right] \tag{3}$$

Unfortunately, this is hard to compute due to the inner summation. Instead, EM defines the complete data log likelihood, $\ell_c(\boldsymbol{\theta})$, as $\sum_{i=1}^{N} \log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta})$. However, the complete data log likelihood is not applicable due to the latent variables so EM algorithm maximizes the expected log likelihood under the posterior distribution of the latent variables, $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$, defined as $\mathbb{E}[\ell_c(\boldsymbol{\theta})|\mathcal{D}, \boldsymbol{\theta}^{t-1}]$ where t is the current iteration index.

EM algorithm is composed of two steps: **E-step** and **M-step**. **E-step** is to compute $Q(\boldsymbol{\theta}', \boldsymbol{\theta})$ and **M-step** is to update the parameters $\boldsymbol{\theta}$ by solving the optimization problem $\mathrm{argmin}_{\boldsymbol{\theta}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$.

C. EM for Gaussian Mixture Models

Gaussian mixture model (GMM) is a mixture model which adopts Gaussian distributions as its base distributions. It can be written as a superposition of K Gaussian distributions with the mean vector and covariance matrix of the k-th component $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. The probability distribution (1) becomes:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4}$$

Here, $\boldsymbol{\theta} = \{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, ..., \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\}$ is a set of the base distributions' parameters where $\mathbf{x}, \boldsymbol{\mu}$ are L-dimensional vectors and $\boldsymbol{\Sigma}$ is $L \times L$ matrix.

1) Expected Log Likelihood Function: The EM algorithm for GMM maximizes $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ given as:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1}) = \sum_{i=1}^{N}\sum_{k=1}^{K} \gamma_{ik} \log \pi_k + \sum_{i=1}^{N}\sum_{k=1}^{K} \gamma_{ik} \log p(\mathbf{x}_i|\boldsymbol{\theta}_k) \tag{5}$$

where $\mathbf{z}_i$ is same as the vector $\mathbf{z}$ introduced above corresponding to the variable $\mathbf{x}_i$ and z_{ik} is the k-th element of $\mathbf{z}_i$. The variable $\gamma_{ik} = p(z_{ik} = 1|\mathbf{x}_i, \boldsymbol{\theta}^{t-1})$ is the *responsibility* that the k-th Gaussian distribution takes for the observed data $\mathbf{x}_i$.

2) E step: In the **E step** of the EM algorithm, γ_{ik} value is updated following the equation below:

$$\gamma_{ik} = p(z_{ik} = 1|\mathbf{x}_i, \boldsymbol{\theta}^{t-1}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k^{t-1}, \boldsymbol{\Sigma}_k^{t-1})}{\sum_{j=1}^{k} \pi_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j^{t-1}, \boldsymbol{\Sigma}_j^{t-1})} \tag{6}$$

The variable $\gamma(z_k) = p(z_k = 1|\mathbf{x}, \boldsymbol{\theta}^{t-1})$ can be considered as the posterior probability corresponding to π_k once $\mathbf{x}$ is observed.

3) M step: In the **M step**, the parameters are updated toward maximizing the Q function w.r.t. $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$. To maximize the expected log likelihood function, set the derivative of (4) with respect to the means $\boldsymbol{\mu}_k$ to be 0. Then we get

$$-\sum_{i=1}^{N} \frac{\pi_k p(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\boldsymbol{\Sigma}_j \pi_j p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0. \tag{7}$$

Multiplying by $\boldsymbol{\Sigma}_k^{-1}$ and rearranging, we get $\mu_k = \frac{1}{\gamma_k}\sum_{i=1}^{N} \gamma_{ik}\mathbf{x}_i$ where $\gamma_k = \sum_{i=1}^{N} \gamma_{ik}$. Similarly, if we take derivative with respect to Σ_k and π_k respectively and follow the same reasoning, we get $\Sigma_k = \frac{1}{\gamma_k}\sum_{i=1}^{N} \gamma_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T$ and $\pi_k = \frac{\gamma_k}{N}$. After computing the new estimates, the parameters are updated for $k = 1, .., K$ and the updates are iterated until the log likelihood estimate converges.

V. Regularizing DNNs Using EM Method

A. The Structure Learning Pipeline

First, we cluster the input nodes based on the weights to the next layer so that the nodes in the same group have similar distributions in their connections to the next layer. Then, we let each output node pick one of the groups which contributes to their activation the most and ignore the inputs from other groups by disconnecting the weights. Therefore the output nodes are clustered in a way that the output nodes in the same group are connected to the same input cluster. With the weights disconnected, the weight matrix would look sparse but is same as the block diagonal matrix in the Fig. 1 when the rows and columns are rearranged.

To cluster the input nodes, we use a GMM and train the mixture model parameters by using an expectation maximization (EM) method. We add the negative of the expected log likelihood function to the objective function so that the mixture model parameters can be trained by the backpropagation together with the network parameters and we can learn the group structure by end-to-end learning.

Once one of the layer forms a group structure, we can push forward the groups to the following layers by reusing an output clustering result of one layer as an input clustering result of the next layer so that we do not have to train additional GMMs. For CNNs, we build a representative value matrix whose elements are the representative values of convolution kernels and apply the same method.

B. Regularizing DNNs

Consider a layer of a neural network which receives input variables, $\mathbf{x} = [x_1 \cdots x_N]^T$ and produces the output $\mathbf{h} = \{h_1, ..., h_M\}$. Let $W \in \mathbb{R}^{N \times M}$ be its weight matrix and $\mathbf{b} = [b_1 \cdots b_M]^T$ be the bias. Then the output layer $\mathbf{h}$ can be formally defined as below:

$$\mathbf{h} = \sigma(\mathbf{x}^{\mathrm{T}} W + b^{\mathrm{T}}) \tag{8}$$

where $\sigma(\cdot)$ is a nonlinear function. Then each element h_i in $\mathbf{h}$ can be written as :

$$h_i = \sigma(\sum_{j=1}^{N} x_j w_{j,i} + b_i) \tag{9}$$

where $w_{j,i}$ is the j-th row, i-th column element in W.

Formally, we substitute $\mathbf{w}_i = [w_{i,1} \cdots w_{i,M}]^{\mathrm{T}}$ for the $\mathbf{x}_i$ in the Section IV-C and calculate the expected likelihood function $Q(\theta, \theta^{t-1})$ in (5). Then $\mathbf{w}_i$ are clustered into K groups in a way that $\mathbf{w}_i$ falls into the k-th group where the responsibility γ_{ik} has the greatest value. Then, the mean vectors $\boldsymbol{\mu}_k$ of the Gaussian distributions are compared with each other to find out in which group k $\mu_{k,j}$, the j-th element in $\boldsymbol{\mu}_k$, has the greatest value. For the j-th column, the row i that falls into the k-th group where $\mu_{k,j}$ has the largest value remains and other rows are erased to 0 (Figure 1) so that only the strongest connections remain.

This row-column clustering of a weight matrix can be interpreted as input and output clustering of the network layer (Fig. 2). First, we cluster the input nodes based on the weights toward output nodes so that the weights from the nodes in the same group to the next layer are from the same distribution. Then, we let the output nodes choose one of the groups that are most strongly connected to them on average and ignore other groups by disconnecting the weights. Therefore, the output nodes are clustered according to the input cluster they chose.

For the implementation, only the diagonal elements of the covariance matrix $\boldsymbol{\Sigma}_k$ were used with small noise added to make sure that its inverse matrix exists and the normal distribution can be calculated. Also, when more than one layer were grouped in a way described above, the column clustering was used as a row clustering at the next layer.

C. Regularizing CNNs

Consider a convolutional layer of a CNN which receives an input X and produces an output H. Given that the input X is composed of N channels and the output H is composed of M channels, we can describe a convolutional layer same as the fully connected layer as below:

$$H = \sigma(X * W + \mathbf{b}) \tag{10}$$

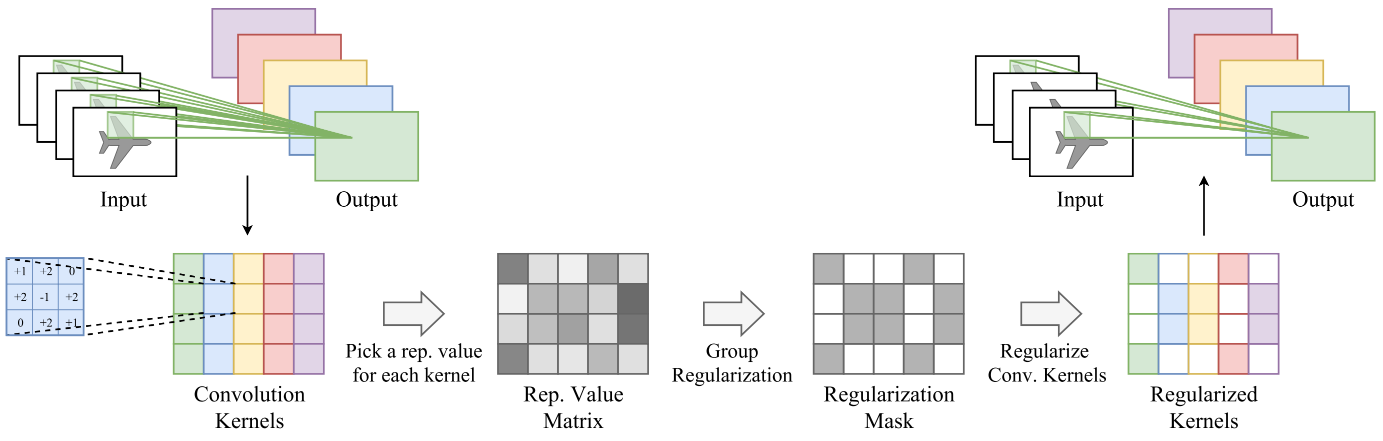


Fig. 3. Regularization of a convolutional layer using EM method.

Here, W is also a $N \times M$ matrix but its elements are the convolution kernels and $*$ is a convolution operation. The i-th row, j-th column element in W, $\mathbf{w}_{i,j}$, represents the i-th channel of the j-th kernel.

To regularize a convolutional layer, we define a representative value for each kernel to build a representative matrix $\hat{W} \in \mathbb{R}^{N \times M}$ and we apply the similar process (Fig. 3) with the Section V-B. Specifically, we use the difference of the maximum and the minimum of the kernel parameters. Then we accomplish the channel-level clustering and learn the group structure of the input and output channels.

D. Optimization

In our model, there exist two sets of parameters to optimize. One is the network parameters which are the weight matrices and bias vectors, and the other is the Q parameters for the GMM. Let's denote them as $\boldsymbol{\theta}_{NN} = \{\mathbf{W}, \mathbf{b}\}$ and $\boldsymbol{\theta}_Q = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}\}$. To optimize these two sets of parameters at the same time, we introduce the following two functions.

First functions is an error function ERR_{NN}, which is the RMSE of the network output $\mathbf{y}$ and the target output $\mathbf{t}$.

$$ERR_{NN} = \sum_{i=1}^{p} (y_i - t_i)^2 \tag{11}$$

The parameters $\boldsymbol{\theta}_{NN}$ are trained to minimized the ERR_{NN} function.

Also, consider the equation (5). Let's denote the negative of the $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ function as $NLL_Q = -Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{t-1})$ which represents the negative log likelihood function. The parameters $\boldsymbol{\theta}_Q$ are trained to minimize the NLL_Q function.

To achieve both goals, we defined the cost function as $ERR_{NN} + NLL_Q$ and solve the optimization problem

$$\min_{\boldsymbol{\theta}_{NN}, \boldsymbol{\theta}_Q} Err_{NN} + \rho NLL_Q \tag{12}$$

Here, ρ, which is called the Q *learning rate* is the hyperparameter that controls the learning speed of $\boldsymbol{\theta}_Q$ separately from $\boldsymbol{\theta}_{NN}$.

TABLE I
LEARNING OF GMM BY GRADIENT DESCENT

Derivatives	Gradient Descent Updates
$\frac{\partial}{\partial \mu_k} NLL_Q$	$-\frac{\gamma_{ik}}{2}(\boldsymbol{\Sigma}_k^{-1} + \boldsymbol{\Sigma}_k^{-T})(\mathbf{x}_i - \boldsymbol{\mu}_k)$
$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} NLL_Q$	$-\frac{\gamma_{ik}}{2}\boldsymbol{\Sigma}_k^{-T}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-T} + \frac{\gamma_{ik}}{2}\frac{\mathrm{tr}(\mathrm{adj}(\boldsymbol{\Sigma}_k))}{(2\pi)^L \lvert\boldsymbol{\Sigma}_k\rvert}$
$\frac{\partial}{\partial \pi_k} NLL_Q$	$-\frac{\partial \gamma_{ik}}{\partial \boldsymbol{\pi}_k}(\log \boldsymbol{\pi}_k + \log p(\mathbf{x}_i \vert \boldsymbol{\theta}_k)) - \frac{\gamma_{ik}}{\pi_k}$
$\frac{\partial \gamma_{ik}}{\partial \pi_k}$	$\frac{p(\mathbf{x}_i \vert \boldsymbol{\theta}^{t-1}) \sum_k' \pi_{k'} p(\mathbf{x}_i \vert \boldsymbol{\theta}^{t-1})}{\{\sum_{k'} \pi_{k'} p(\mathbf{x} \vert \boldsymbol{\theta}^{t-1})\}^2}$

TABLE II
MODEL ARCHITECTURE OF TEST MODELS ON MNIST

Model	Input	Layer1	Layer2	Output
MLP	400	200	200	10
MLP-res	400	200	**200**	10
MLP-ℓ_1	400	200	**200**	10
MLP-DNS	400	200	**200**	10
MLP-EM (this work)	400	200	**200**	10

Both parameters are updated by gradient descent.After learning the $\boldsymbol{\theta}_Q$ parameters for some steps, gradient descent optimization of the NLL_Q term are stopped to ensure that the network parameters of the lower layers are not affected much by the structural changes caused in NLL_Q term. During the training when the NLL_Q value starts to increase the gradients from the NLL term are blocked.

The derivative with respect to each parameter is monotonically non-decreasing (Table I) assuring that the functions are concave. Therefore, the parameters converge during the gradient update.

VI. EXPERIMENTS

In this section, we evaluate our method on three types of deep neural networks: fully-connected network, or a multi-layer perceptron; 1-dimensional CNN which receives sequential data; and 2-dimensional CNN which receives image data. We use three public datasets for each DNN; MNIST dataset

TABLE III
MODEL ARCHITECTURE OF TEST MODELS ON GROUNDWATER

Model	Input	Conv1	Conv2	Conv3	Reshape[1]	Fully1[2]	Output
		[5]@200 conv, stride 1 [3] max pool, stride 3					
CNN	27x87	27x200	9x200	3x200	600	100	1
ConvLSTM	27x87	27x200	9x200	3x200	**3x200**	**100**	1
CNN-res	27x87	27x200	**9x200**	**3x200**	600	100	1
CNN-ℓ_1	27x87	27x200	**9x200**	**3x200**	600	100	1
CNN-DNS	27x87	27x200	**9x200**	**3x200**	600	100	1
CNN-EM	27x87	27x200	**9x200**	**3x200**	600	100	1

[1]A reshape layer flattens the previous layer in CNNs and does nothing in the ConvLSTM.
[2]This layer is replaced with a many-to-one LSTM layer in the ConvLSTM. The LSTM layer receives 200 dimensional vectors and computes activations for 3 time steps.

TABLE IV
MODEL ARCHITECTURE OF TEST MODELS ON CIFAR-10

Model	Input	Conv1	Conv2	Conv3	Conv4	Reshape	Fully1	Output
		[3x3]@128 conv, stride 1 [2x2] max pool, stride 2						
CNN	32x32x3	16x16x128	8x8x128	4x4x128	2x2x128	512	128	10
CNN-res	32x32x3	16x16x128	**8x8x128**	**4x4x128**	**2x2x128**	512	128	10
CNN-ℓ_1	32x32x3	16x16x128	**8x8x128**	**4x4x128**	**2x2x128**	512	128	10
CNN-DNS	32x32x3	16x16x128	**8x8x128**	**4x4x128**	**2x2x128**	512	128	10
CNN-EM	32x32x3	16x16x128	**8x8x128**	**4x4x128**	**2x2x128**	512	128	10

for MLPs, U.S. groundwater dataset for 1-d CNNs, and CIFAR-10 dataset for 2-d CNNs. We compare our method with three baseline methods which are residual model [1], ℓ_1 regularization, Dynamic Network Surgery (DNS) [19], and additional convLSTM [28], [29] for the groundwater dataset in terms of the performance and the number of parameters.

We build conventional DNNs first, then select some layers and modify them to build the baseline models. To build residual model, we replace selected layers with residual layers [30] which implement $ReLU(\mathbf{x}^T W_1)^T W_2 + \mathbf{x}$. Also, those selected layers are regularized using regularization to build ℓ_1 regularized model or pruned using DNS method [19], [31]. To build convLSTM, we replace the fully connected layers on top of the convolutional layers with LSTM layers.

The number of clusters used in the experiments is determined experimentally and the pruning rate of DNS models is adjusted so that they keep similar number of parameters as our grouped models. Models for the experiments are designed for the purpose of comparative observations, not the state-of-the-art classification or regression performance.

A. MNIST

MNIST dataset is a handwritten digits dataset [32] which is composed of 28×28 pixel grayscale handwritten digit images. We resize the images into 20×20 pixels as done in [7] to hasten the experiment and feed them to the test models. We build a standard MLP with two fully connected layers first, then modify the second layer to build other baseline models; a residual model (MLP-res), ℓ_1 regularized model (MLP-ℓ_1), compressed model (MLP-DNS), and our grouped model (MLP-EM). Model architecture are described in the Table II. The number of clusters of MLP-EM is determined to be 3.All models were trained for 500 epochs and the learning rate was set to 0.01. The Q learning rate, ρ in (12), was set to 0.00001 for MLP-EM.

TABLE V
NUMBER OF PARAMETERS AND TEST ACCURACY IN MNIST

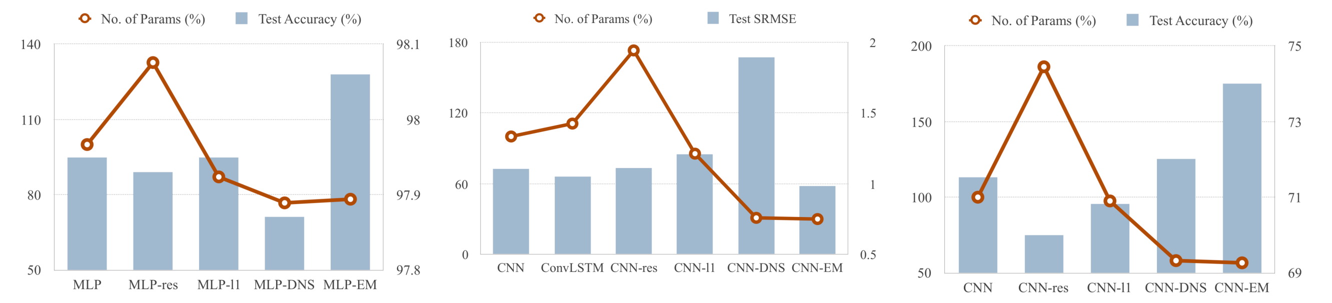
Model	No. of Params (%)	Test Accuracy (%)
MLP	100	97.95
MLP-res	132.69	97.93
MLP-ℓ_1	87.09	97.95
MLP-DNS	76.73	97.87
MLP-EM (this work)	78.19	**98.06**

Table V shows the results. MLP-EM is the only model that outperforms the basic MLP with the test accuracy of 98.06% using 78% parameters of the basic MLP model. MLP-res uses the most parameters as the residual layer is composed of two feature mappings, however, does not show better performance than MLP. MLP-l1 and MLP-DNS could reduce the number of parameters, however, only with loss the accuracy. Fig. **??** shows the grouped parameters when the number of clusters is set to be 2 and 3. The true grouped matrices are sparse but can be described as a block diagonal matrix as described in the Fig. 1.

B. Groundwater

The groundwater dataset is provided by United States Geological Survey (USGS)[1]. It is composed of parameters related to the groundwater status collected from various sites in the United States territory and we use daily depth to water level records from the regions excluding Hawaii and Alaska. Regions which have records of 28 years (1987-2015) or more were selected and those that have unrecorded periods longer than two months were excluded. Empty records shorter than

[1]https://waterdata.usgs.gov



(a) MNIST (b) Groundwater (c) CIFAR-10

Fig. 4. Experiment Results

two months were filled using linear interpolation. Final dataset contains records from 88 sites of $10,228$ days. The models are trained to reconstruct one randomly selected site using other 87 sites.

To train groundwater data, we use 1 dimensional CNNs. The basic CNN and CNN variant baseline models receive small patches of the time series data that are sampled by sliding window method as inputs and process them using 1-dimensional convolutions. The most basic CNN that we use comprises three convolutional layers followed by two fully connected layers. We replaced Conv2 and Conv3 layers with residual layer (CNN-res), layer regularized by ℓ_1 norm (CNN-ℓ_1), layer pruned by DNS (CNN-DNS), and grouped layer (CNN-EM).

Both CNN and LSTM are applicable to sequential data and it is hard to say which one outperforms the other [33]. However, combined models have shown to be better performing than both CNNs and LSTMs [28], [29]. Therefore, we compare our CNN-EM model additionally with convLSTM for the groundwater data. To build a convLSTM, we replace the first fully-connected layer with a many-to-one LSTM layer.

To compare the performance, we use the standardize root mean squared error ($SRMSE$) because the groundwater level of different sites are largely different. SRMSE is defined as following:

$$SRMSE = \frac{rmse(\mathbf{y},\mathbf{t})}{rmse(\mathbf{t},\bar{\mathbf{t}})} \tag{13}$$

$$rmse(\mathbf{y},\mathbf{t}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2} \tag{14}$$

where $\mathbf{y}$ is the network output, $\mathbf{t}$ is the target output, and $\bar{\mathbf{t}}$ is the mean value of $\mathbf{t}$.

We repeated random selection and reconstruction for three times and took average of SRMSEs and the number of parameters. The chosen sites are 582, 2769, and 3866. The models are trained for 300-100 epochs with learning rates set to 10^{-6} and weight decaying factor 0.1. The number of groups for CNN-EM is 5 and the Q learning rate is set to 0.01.

Table VI shows the experiment results. Eventhough CNN-res exploits more paramters, it cannot improve the performance, however, ConvLSTM show lower error than CNN. Also, both CNN-DNS and CNN-ℓ_1 loss the accuracy by reducing the number of parameters. On the other hand, CNN-EM shows lower error than CNN using only 30% parameters of CNN model. Moreover, it shows lower average SRMSE than that of convLSTM.

TABLE VI
AVERAGE NUMBER OF PARAMETERS AND SRMSE IN GROUNDWATER

Model	No. of Params (%)	Test SRMSE (%)
CNN	100	1.1014
ConvLSTM	110.95	1.0516
CNN-res	173.09	1.1125
CNN-ℓ_1	85.41	1.2071
CNN-DNS	30.91	1.8934
CNN-EM (this work)	29.88	**0.9800**

TABLE VII
NUMBER OF PARAMETERS AND TEST ACCURACY IN CIFAR-10

Model	No. of Params (%)	Test Accuracy (%)
CNN	100	71.53
CNN-res	186.26	70.00
CNN-ℓ_1	97.54	70.82
CNN-DNS	58.07	72.01
CNN-EM (this work)	56.60	**74.00**

C. *CIFAR-10*

CIFAR-10 dataset is composed of 32×32 pixel images of three channels that belong to 10 object classes. We use a CNN with four convolutional layers and 2 fully-connected layers as the base model and modify Conv1, Conv2, and Conv3 layers to build other baseline models, CNN-res, CNN-ℓ_1, and CNN-DNS. Also, we group the same layers and build CNN-EM. The models are trained for 300-100 epochs with learning rates 0.0001 and weight decaying factor 0.1. The number of clusters is 2 and the Q learning rate, ρ, is set to 10^{-6}.

As shown in Table VII, CNN-res uses 86% more parameters than the basic CNN model but its test accuracy was lower by 1.53%. Test accuracy of CNN-ℓ_1 was also lower than CNN. On the other hand, CNN-DNS improves the accuracy by 0.48% using only 58% parameters than CNN. However, CNN-EM's test accuracy is 2.47% higher than the CNN model and 1.99% higher than the CNN-DNS.

VII. Conclusion

We proposed a novel method to eliminate redundant parameters of a deep neural network by learning the group structure and removing unnecessary group connections. We also proposed a end-to-end expectation maximization method to train the GMM parameters which were used to group the networks. The GMM parameters are updated by the devised gradient descent updates together with the deep neural network weight parameters during the back propagation. We proved that GMM parameters converge to an optimal point during the updates as shown in the Table I.

In the experiments with three different types of neural networks on three public datasets, we showed that our method can reduce the number of parameters without loss of accuracy. As shown in the Fig. 4, our method improved the performance resulting in the best accuracy in every experiments.

Acknowledgment

The authors thank Janghoon Ju for his insightful comments. This work was supported by the National Research Foundation of Korea (NRF) grant (No. 2014M2A8A2074096) and Institute for Information and communications Technology Promotion (IITP) grant (No. 2017-0-01779, A machine learning and statistical inference framework for explainable artificial intelligence) funded by the Korea government (Ministry of Science and ICT, MSIT).

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[2] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[3] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2015, pp. 2377–2385.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ArXiv preprint 1409.1556*, 2014.

[5] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1646–1654.

[6] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[7] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the International Conference on Machine Learning*, 2013, pp. 1058–1066.

[8] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2017, pp. 3859–3869.

[9] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. D. Freitas, "Predicting parameters in deep learning," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2013, pp. 2148–2156.

[10] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2014, pp. 2654–2662.

[11] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proceedings of the Deep Learning Workshop, colocated with the Annual Conference on Neural Information Processing Systems*, 2015.

[12] R. Reed, "Pruning algorithms-a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[13] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2015, pp. 1135–1143.

[14] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proceedings of the International Conference on Machine Learning*, 2015, pp. 2285–2294.

[15] W. Chen, J. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing convolutional neural networks in the frequency domain," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[16] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," 2017, pp. 5068–5076.

[17] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proceedings of the International Conference on Learning Representations*, 2017, pp. 1–13.

[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations*, 2016, pp. 1–14.

[19] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Proceedings of the Annual Conference on Neural Information Processing Systems*, 2016, pp. 1379–1387.

[20] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher, "Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework," in *ACM Conference on Embedded Network Sensor Systems*, 2017, pp. 1–14.

[21] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[22] M. Yuan and Y. Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[23] J. Friedman, T. Hastie, and R. Tibshirani, "A note on the group lasso and a sparse group lasso," *arXiv preprint arXiv:1001.0736*, 2010.

[24] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," *Journal of Computational and Graphical Statistics*, vol. 22, no. 2, pp. 231–245, 2013.

[25] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, 2017.

[26] C. Kemp, J. B. T. T. L. Griffiths, T. Yamada, and N. Ueda, "Learning systems of concepts with an infinite relational model," in *Proceedings of the National Conference on Artificial Intelligence*, 2006, pp. 381–388.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (methodological)*, pp. 1–38, 1977.

[28] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015, pp. 4580–4584.

[29] F. J. Ordóñez and D. Roggen, "Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition," *Sensors*, vol. 16, no. 1, p. 115, 2016.

[30] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the International Conference on Machine Learning*, 2010, pp. 807–814.

[31] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *ArXiv preprint 1408.5093*, 2014.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[33] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of cnn and rnn for natural language processing," *ArXiv preprint 1702.01923*, 2017.