Improving Data Analytics with Fast and Adaptive Regularization

Zhaojing Luo, Shaofeng Cai, Gang Chen, *Member, IEEE*, Jinyang Gao, Wang-Chien Lee, *Member, IEEE*, Kee Yuan Ngiam, and Meihui Zhang, *Member, IEEE*

Abstract—Deep Learning and Machine Learning models have recently been shown to be effective in many real world applications. While these models achieve increasingly better predictive performance, their structures have also become much more complex. A common and difficult problem for complex models is overfitting. Regularization is used to penalize the complexity of the model in order to avoid overfitting. However, in most learning frameworks, regularization function is usually set with some hyper-parameters where the best setting is difficult to find. In this paper, we propose an adaptive regularization method, as part of a large end-to-end healthcare data analytics software stack, which effectively addresses the above difficulty. First, we propose a general adaptive regularization method based on Gaussian Mixture (GM) to learn the best regularization function according to the observed parameters. Second, we develop an effective update algorithm which integrates Expectation Maximization (EM) with Stochastic Gradient Descent (SGD). Third, we design a lazy update and sparse update algorithm to reduce the computational cost by 4x and 20x, respectively. The overall regularization framework is fast, adaptive, and easy-to-use. We validate the effectiveness of our regularization method through an extensive experimental study over 14 standard benchmark datasets and three kinds of deep learning/machine learning models. The results illustrate that our proposed adaptive regularization method achieves significant improvement over state-of-the-art regularization methods.

Index Terms—Adaptive regularization, data analytics, complex analytics, data science, knowledge discovery and data mining

1 Introduction

Recent developments in Deep Learning and Machine Learning technology [1], [2], [3], [4] have led to a series of breakthroughs in many real world applications [5], [6], [7], [8]. Thanks to the large public datasets [9] and high-performance computing systems, e.g., large-scale distributed clusters of computers equipped with GPUs, we are now able to build more complex and better-performing predictive models [10], [11], [12], [13].

In the past few years, many advanced deep learning models have been developed. An important trend in these recent developments is that the number of layers in these models (and thus their complexity) has increased rapidly. Take ILSVRC, the famous visual recognition challenge, as an example. The number of layers of the annual winning model has increased from 8 layers in 2012 to 152 layers in 2015.

Unfortunately, as the models become more complex, the overfitting problem is becoming more severe as well. Let's use a standard benchmark image dataset, CIFAR-10, as an example to illustrate the overfitting problem. Using the CIFAR-10 dataset, the task is to train a classifier that classifies a given image into one of the 10 classes (airplane, automobile, truck, etc.) The accuracy of Alex-CIFAR-10 model on the CIFAR-10 image classification problem is shown in Fig. 1. Let us first focus on the plain Alex-CIFAR-10 model. As we can observe from the figure, the training accuracy of plain Alex-CIFAR-10 is very high (approaching 100 percent) but the test accuracy is quite low (only about 76 percent) when the model converges. This is because the model is likely to fit the noise in the training data so that it does not generalize very well to the test data. This phenomenon of overfitting [14] is common in complex models and significantly affects the predictive ability of models.

Fortunately, overfitting problem [14] can be effectively addressed by regularization [15], which typically involves adding a penalty term on the complexity of the model. Equation (1) shows the overall loss function that a model aims to minimize. It consists of two terms, the first term, data-misfit, also called the training loss, is an error term that indicates how well the model fits the training data and the second term is the penalty term, which is called regularization term. It consists of a strength parameter β and a

- Z. Luo, S. Cai, and J. Gao are with the Department of Computer Science, Schoool of Computing, National University of Singapore, 117417, Singapore. E-mail: {zhaojing, shaofeng, jinyang.gao}@comp.nus.edu.sg.
- G. Chen is with the College of Computer Science and Technology, Zhejiang University, and also with the Key Laboratory of Intelligent Computing Based Big Data of Zhejiang Province, Hangzhou, Zhejiang 310027, China. E-mail: cg@zju.edu.cn.
- W.-C. Lee is with the Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802 USA. E-mail: wlee@cse.psu.edu.
- K.Y. Ngiam is with National University Health System, 119228, Singapore. E-mail: kee_yuan_ngiam@nuhs.edu.sg.
- M. Zhang is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: meihui_zhang@bit.edu.cn.

Manuscript received 23 Sept. 2018; revised 6 Apr. 2019; accepted 28 Apr. 2019. Date of publication 13 May 2019; date of current version 11 Jan. 2021. (Corresponding author: Zhaojing Luo.) Recommended for acceptance by P. K. Chrysanthis, B. C. Ooi, and J. Dittrich. Digital Object Identifier no. 10.1109/TKDE.2019.2916683

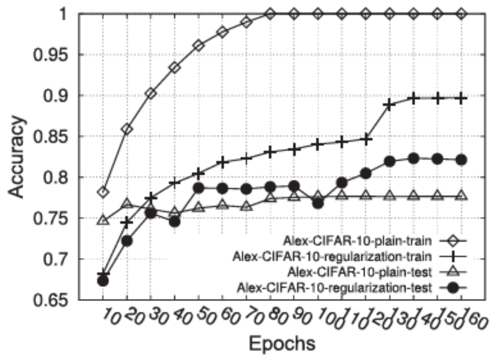


Fig. 1. Training and test loss with and without regularization.

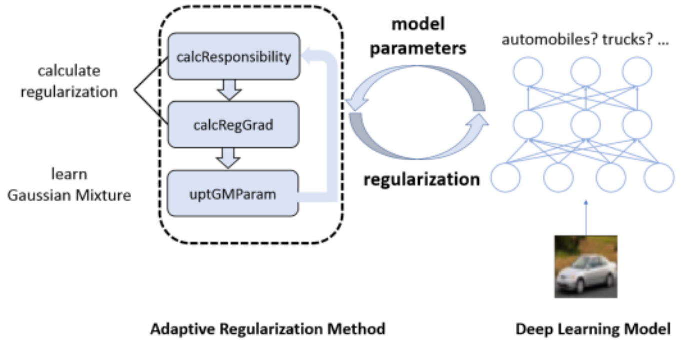


Fig. 2. Overview of adaptive regularization tool.

function f of the model parameters $\boldsymbol{w}$.

$$\text{Loss(w)} = \text{data-misfit} + f(\beta, \boldsymbol{w}). \tag{1}$$

Typically, the function f is related to L1-norm ($f(\beta, \boldsymbol{w}) = \beta||\boldsymbol{w}||_1$) or L2-norm ($f(\beta, \boldsymbol{w}) = \beta||\boldsymbol{w}||_2^2$) of the parameter $\boldsymbol{w}$. We refer back to Fig. 1. For Alexnet-CIFAR-10, human experts [16], [17] have set $f(\beta, w) = \beta||w||_2^2$, and different β values for different layers. We can observe that when the regularization is applied, the training accuracy decreases while the test accuracy increases significantly.

Many regularization methods have been proposed in the literature. However, the use of these methods is typically ad hoc and experimental in nature. Given a specific application, data scientists typically need to make many painstaking attempts to choose the optimal type of f (e.g., L1-norm, L2-norm) and strength of the regularization, β, on a held-out validation dataset, to decide the best f and β which will be applied on the test dataset.

In [17], models are carefully tuned by the human experts and it has been shown that using different values of β at different layers of a deep learning model may lead to better performance in the test data. Nevertheless, the sharp rise in the number of deep learning layers poses a great challenge for manual setting of effective regularization for each layer.

Driven by the difficulty of setting the best regularization, we ask the following question: Is it possible to adaptively learn the best regularization term? The answer is YES. From the Bayesian view point, regularization term corresponds to a prior distribution for the model parameters $\boldsymbol{w}$. For instance, L1-norm regularization corresponds to a Laplacian prior and the L2-norm regularization corresponds to a Gaussian prior. The best regularization should be the one that equals to the actual distribution of model parameters w. However, the actual distribution of model parameters $\boldsymbol{w}$ for different tasks may vary quite significantly, and as a result, there is no single regularization setting (strength β and type of f) that can fit every problem well.

Fortunately, the intermediate model parameters $\boldsymbol{w}$ learned during training process can be very informative to approximate the actual distribution. Based on this insight, we propose an adaptive regularization tool designed to learn the best regularization term during the model training process. Instead of employing a fixed prior distribution (i.e., by fixing f and β) for the model parameters w as the regularization term, our key idea is to capture the model parameter distribution with an adaptive distribution function. In other words, we propose to fit an adaptive distribution function using the intermediate model parameters obtained during the model training process. Subsequently, this adaptive distribution function will be used to impose regularization on the model parameters.

Based on the above principle, we propose a practical adaptive regularization tool based on the Gaussian Mixture (GM) framework. We choose GM because it provides a richer class of density models, thus can model the prior of model parameters $\boldsymbol{w}$ better. Fig. 2 illustrates the core idea: the deep learning model inputs model parameters $\boldsymbol{w}$ to the adaptive regularization tool; the tool will then iteratively learn the Gaussian Mixture as actual prior over $\boldsymbol{w}$ and adaptively calculate the regularization term for the model.

However, adaptively learning such GM from the intermediate model parameters w is challenging for several reasons. Two sets of parameters, namely GM parameters and model parameters $\boldsymbol{w}$ need to be updated. These two sets of parameters are closely related to each other. An appropriate algorithm should be designed to update these two sets of parameters properly. Also, learning GM is an iterative and time-consuming process. We need to design an appropriate algorithm to reduce the computational cost.

In order to update GM parameters and model parameters properly, we design an innovative and effective update method where GM parameters are updated via a lightweight Expectation-Maximization (EM) algorithm [18] and the model parameters can be learned under a common optimization framework such as Stochastic Gradient Descent (SGD) [19]. To reduce the computational cost, we propose lazy update and sparse update algorithms that reduce the computational time by more than 4 times and 20 times respectively.

Our proposed regularization method is a general and flexible tool designed to support different machine learning and deep learning models. The regularization tool has been integrated into our GEMINI software stack [20] designed to support healthcare data analytics. Fig. 3 shows GEMINI, where various subsystems form an end-to-end big data analytics pipeline, from data cleansing to visualization. When the raw data is first fed into GEMINI, the data cleaning and integration system, DICE, cleans the data. It works with a crowdsourcing platform, CDAS, to improve data quality using subject matter experts (e.g., clinicians). The cleansed data can then be processed by a distributed and scalable data processing system, epiC [21], which provides big data processing and analytics such as aggregation and summarization. Deep analytics is supported by Apache SINGA [22], a distributed deep learning platform. The red box illustrates the interaction of

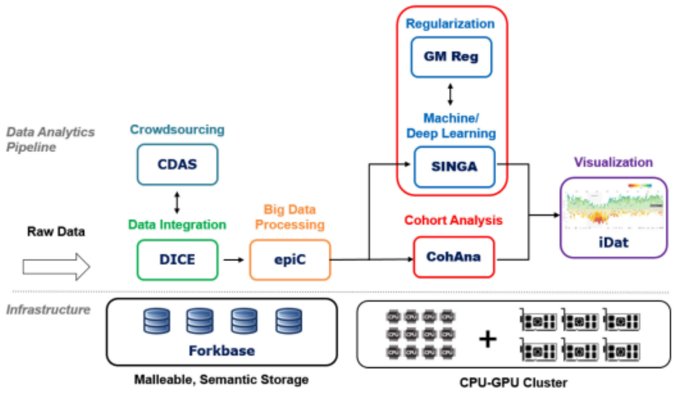


Fig. 3. GEMINI healthcare data analytics software stack.

the proposed GM regularization tool (GM Reg) with Apache SINGA to provide adaptive regularization for deep learning models. In the meantime, the data can be fed into CohAna for cohort analysis [23], [24]. Lastly, the results could be visualized via iDat. In order to support such a large data analytics software stack, advanced database supports are needed for the storage layer [25]. In our case, the data is stored in Forkbase [26], a universal immutable storage system. GEMINI runs either on a single machine or in a CPU-GPU cluster.

Contributions. The contributions of this paper include:

- We propose a general adaptive regularization method based on GM to learn the best regularization according to the intermediate model parameters instead of making ad-hoc attempts to obtain a suitable regularization setting.
- We propose an efficient update framework where GM is trained by a lightweight EM algorithm [27] and the model parameters are updated via SGD. We also propose efficient lazy update and sparse update algorithms to reduce the computational cost by 4 times and 20 times respectively. Apart from experimental results, theoretical analysis is also given for these update algorithms.
- We conduct extensive experiments using both real world datasets and standard benchmark machine learning datasets. We also propose a supervised way to evaluate the effectiveness of the GM regularization. Results on all the datasets demonstrate that our regularization can achieve better (or equally good) performance than all the baseline regularization methods (L1-norm regularization [28], L2-norm regularization [15], Elastic-net regularization [29] and Huber-norm regularization [30]) under their best settings.
- We design our regularization method to be an easy-to-use and general tool. We also provide guidance on setting the appropriate hyper-parameters for different kinds of models.

This article is an extension of our conference paper [31]. In this work, we extend the GM regularization to sparse datasets and also provide a supervised way to show the effectiveness of our GM regularization method. Additionally, we provide a theoretical analysis on the effectiveness of our proposed lazy update algorithm and design easy-to-use APIs for the GM regularization tool. The remainder of the paper is structured as follows. Section 2 gives some related background on Bayesian interpretation of regularization, GM and several useful priors. Section 3 introduces our proposed GM regularization method. Section 4 introduces the theoretical analysis for lazy update algorithm and Section 5 introduces our designed APIs for our regularization tool. Section 6 reports the experimental results. Related work is reviewed in Section 7 and finally Section 8 concludes this paper.

TABLE 1
Table of Symbols

Symbol	Definition
M	number of features
N	number of samples
$\boldsymbol{x}^{(n)} \in \mathcal{R}^{M\times 1}$	features of the nth sample in the training set
$y^{(n)}$	label of the nth sample in the training set
$\boldsymbol{w} \in \mathcal{R}^{M\times 1}$	model parameters
$\mathcal{D}$	$(\boldsymbol{x}^{(n)}, y^{(n)})_{n=1}^{N}$, features/label pairs in the training set
K	number of Gaussian components
$\boldsymbol{\pi}$	$[\pi_1, \pi_2, \ldots, \pi_k]^T$, mixing coefficient (satisfy the constraint $\sum_{k=1}^{K} \pi_k = 1$)
$\boldsymbol{\lambda}$	$[\lambda_1, \lambda_2, \ldots, \lambda_k]^T$, precision (inverse of variance)
$\mathcal{N}(x\|\mu_k, \lambda_k)$	Gaussian distribution of the kth Gaussian component
$\boldsymbol{\alpha}$	$[\alpha_1, \ldots, \alpha_K]^T$, parameters of Dirichlet distribution
$\boldsymbol{g}_{ll}$	the gradients of model parameters with respect to the negative log likelihood function
$\boldsymbol{g}_{reg}$	the gradients of model parameters with respect to the regularization term
L	learning rate
B	the number of mini-batches in the training set
I_g	GM parameter update interval
I_m	model parameter update interval
E	number of first few epochs when lazy update is not employed
t_{ll}	the time for calculating $\boldsymbol{g}_{ll}$ for each mini-batch
RL	the ratio of time for E-step, M-step and SGD step when lazy update is employed to t_{ll}
RNL	the ratio of time for E-step, M-step and SGD step when lazy update is not employed to t_{ll}

2 Preliminaries

In this section, we discuss the required prior knowledge of regularization. Table 1 lists the definitions of symbols used in this paper.

2.1 Bayesian Interpretation of Regularization

From the Bayesian perspective, regularization corresponds to a prior distribution over the model parameters $\boldsymbol{w}$. Let $\mathcal{D}$ denote the observed data and $\boldsymbol{w}$ denote the model parameters.

According to Bayes' Theorem, the posterior probability of model parameters $\boldsymbol{w}$ is given by

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})}, \tag{2}$$

where $p(\mathcal{D}|\boldsymbol{w})$ is the likelihood function and $p(\mathcal{D})$ is a constant.

For $\boldsymbol{w}$, the probability is usually estimated using maximum a posteior (MAP) estimation [32], [33]. The MAP problem can be written as below.

$$\begin{aligned} \boldsymbol{w}_{MAP} &= \arg\max_{\boldsymbol{w}} \; p(\boldsymbol{w}|\mathcal{D}) \\ &= \arg\max_{\boldsymbol{w}} \; p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) \\ &= \arg\max_{\boldsymbol{w}} \; \log p(\mathcal{D}|\boldsymbol{w}) + \log p(\boldsymbol{w}). \end{aligned} \tag{3}$$

The term $\log p(\boldsymbol{w})$ is the log of model parameter prior distribution and it is the regularization term (corresponds to $f(\beta, w)$ in Equation (1). Specifically, if $p(\boldsymbol{w})$ is a Laplacian distribution or Gaussian distribution, this term corresponds to L1-norm and L2-norm regularization respectively. For Elastic-net regularization, the prior distribution $p(w)$ corresponds to a compromise between the Laplacian and Gaussian distributions. For Huber-norm regularization, the corresponding prior distribution is piecewise: Gaussian distribution for small value model parameters and Laplacian distribution for large value model parameters. In our work, we assume $p(\boldsymbol{w})$ follows a GM distribution where each Gaussian component is centered at zero but may have different variances.

2.2 Gaussian Mixture Distribution

GM is a superposition of multiple Gaussian distributions. In our GM regularization, we assume all model parameters $\boldsymbol{w}$ are sampled from the same one-dimensional GM distribution. The one-dimensional GM distribution can be expressed in the form below.

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \lambda_k), \tag{4}$$

where K is the number of Gaussian components and π_k is the mixing coefficients which satisfy the constraint $\sum_{k=1}^{K} \pi_k = 1$, and $\mathcal{N}(x|\mu_k, \lambda_k)$ is Gaussian distribution. μ_k is the mean and λ_k is the precision (inverse of Gaussian variance) of the kth Gaussian component.

2.3 Dirichlet and Gamma Prior

The intermediate model parameters during the training process do not have a stationary distribution. Therefore, the actual Gaussian components cannot be learned directly from these intermediate model parameters. In order to learn the GM prior for the model parameters $\boldsymbol{w}$, two prior distributions are introduced for mixing coefficients π_k and Gaussian precisions λ_k respectively.

Dirichlet distribution, which is used as the prior distribution for mixing coefficients π_k, is defined as follows.

$$\mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\ldots\Gamma(\alpha_K)} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}, \tag{5}$$

where $\alpha_1, \ldots, \alpha_K$ are the parameters for the distribution, $\alpha_0 = \sum_{k=1}^{K} \alpha_k$, and $\boldsymbol{\alpha}$ denotes $[\alpha_1, \ldots, \alpha_K]^T$. $\Gamma(x)$ is the Gamma function. This prior distribution is introduced in order to learn more Gaussian components.

As mentioned in Section 2.1, the means of all the Gaussian components of the GM distribution are set to zero. When the mean of a Gaussian distribution is fixed, the Gamma distribution is the conjugate prior for the Gaussian precision. Gamma distribution is defined as follows.

$$\mathrm{Gam}(\lambda|a, b) = \frac{1}{\Gamma(a)} b^a \lambda^{a-1} \exp(-b\lambda), \tag{6}$$

where a and b are the two parameters for the distribution. They control the shape and the decaying rate of the Gamma distribution.

In the process of GM learning, a and b are used for controlling the scale of $\boldsymbol{\lambda}$. This is due to the fact that the values of most model parameters are small. If GM is learned based on these model parameters, large $\boldsymbol{\lambda}$ will be learned which will impose very strong regularization and that is harmful to the model. a and b can help to smoothen the learning of $\boldsymbol{\lambda}$.

3 Proposed Adaptive Regularization

In this section, we introduce our proposed adaptive regularization method based on GM.

3.1 GM Regularization Term

In this paper, GM distribution is used as model parameter prior. In particular, we assume that all the model parameters are independent and identically distributed (this assumption is commonly used in Gaussian prior, Laplacian prior, etc.). Also, we assume all the model parameters follow a GM distribution where each Gaussian component is centered at zero but may have different variances.

$\mathcal{D} = (\boldsymbol{x}^{(n)}, y^{(n)})_{n=1}^{N}$, with $\boldsymbol{x}^{(n)} \in \mathcal{R}^{M\times 1}$ and $y^{(n)} \in \mathcal{R}$, is used to denote the set of input/output pairs in the training set, $\boldsymbol{w} \in \mathcal{R}^{M\times 1}$ denotes model parameters, where N is the number of samples and M is the number of features. Mixing coefficients $[\pi_1, \pi_2, \ldots, \pi_k]^T$ are denoted as $\boldsymbol{\pi}$ and precisions $[\lambda_1, \lambda_2, \ldots, \lambda_k]^T$ as $\boldsymbol{\lambda}$. Considering the prior distributions introduced in Section 2.3, the prior distribution of model parameters $\boldsymbol{w}$ can then be written as below.

$$\begin{aligned} &p(\boldsymbol{w}, \boldsymbol{\pi}, \lambda|\boldsymbol{\alpha}, a, b) = p(\boldsymbol{w}|\boldsymbol{\pi}, \lambda)p(\boldsymbol{\pi}|\boldsymbol{\alpha})p(\lambda|a, b) \\ &=(\prod_{m=1}^{M}(\sum_{k=1}^{K}\pi_k\mathcal{N}(w_m|0, \lambda_k)))\mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha})\prod_{k=1}^{K}\mathrm{Gam}(\lambda_k|a, b), \end{aligned} \tag{7}$$

where K is the number of Gaussian components, $p(\boldsymbol{\pi}|\boldsymbol{\alpha})$ and $p(\boldsymbol{\lambda}|a, b)$ are the prior distributions for GM parameters $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$.

3.2 Loss Function–Optimization Function

In order to solve the MAP problem formulated in Equation (3), given the GM regularization term, the loss function (optimization function) is defined as below.

$$G = -\log p(\mathcal{D}|\boldsymbol{w}) - \log p(\boldsymbol{w}, \boldsymbol{\pi}, \boldsymbol{\lambda}|\boldsymbol{\alpha}, a, b), \tag{8}$$

where the first term is the negative log likelihood function and the second term is the regularization term. This is the function that is optimized for the prediction and classification tasks.

3.3 Optimization

As mentioned in the introduction, for our adaptive regularization method, two sets of related parameters need to be updated, namely, GM parameters and model parameters. Typically, the model parameters are learned through SGD, which is employed for updating model parameters here.

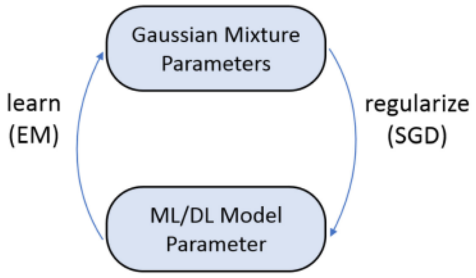


Fig. 4. Overview of SGD and EM.

For GM parameters, a lightweight EM algorithm is designed. Fig. 4 shows how SGD interacts with EM in our update method. After both kinds of parameters are initialized, regularization based on GM is calculated. The calculated regularization then affects model parameters through SGD. After the model parameters are updated via an SGD step, one step of EM is employed to update the GM based on the updated model parameters. Subsequently, a new regularization is calculated for the model parameters. This process iterates until the algorithm converges.

3.3.1 *Responsibility Function*

Responsibility function of the kth Gaussian component for the mth model parameter, w_m, is defined as follows.

$$r_k(w_m) = \frac{\pi_k p_k(w_m)}{\sum_{j=1}^{K} \pi_j p_j(w_m)}, \tag{9}$$

where $p_k(w_m)$ is the probability density of w_m under Gaussian component k. This function calculates the conditional probability that a particular model parameter of $\boldsymbol{w}$, w_m, is generated by a particular Gaussian component k. This responsibility function is used for calculating model parameter gradients and GM parameter update formulas introduced below.

3.3.2 *Gradient Descent for Model Parameters*

When GM parameters are fixed, gradient descent method is used to update model parameters $\boldsymbol{w}$. The gradient for the mth model parameter $\boldsymbol{w}$ with respect to G is given by

$$\frac{\partial G}{\partial w_m} = \frac{-\log p(\mathcal{D}|\boldsymbol{w})}{\partial w_m} + \sum_{k=1}^{K} (r_k(w_m)(\lambda_k w_m)), \tag{10}$$

where $\boldsymbol{x}_m^{(n)}$ is the mth dimension of sample n. For simplicity, we denote the first term as the $\boldsymbol{g}_{ll}$ (all model parameters) and the second term as the $\boldsymbol{g}_{reg}$ (all model parameters).

The $\boldsymbol{g}_{reg}$ is a weighted sum of the product of Gaussian precision and the model parameter value. The responsibility function $r_k(w_m)$ determines the weighting value. This equation shows the regularization effect is a collective effect of different Gaussian components.

Since the probability density function $p_k(w_m)$ is in the form of exponential function, it affects the responsibility function significantly. This provides an opportunity for giving different regularization strengths to different model parameters. For areas that are near zero, the Gaussian component that has the largest precision dominates other Gaussian components. Consequently, for model parameters with small values, the regularization is strong because it is mainly imposed by this Gaussian component with the largest precision. On the contrary, for model parameters with large values, the regularization is weak.

3.3.3 *Update for GM Parameters*

In this section, the update formulas for GM parameters $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ are introduced.

The derivatives for the Gaussian precisions and mixing coefficients are as follows.

$$\frac{\partial G}{\partial \lambda_k} = -(\frac{a-1}{\lambda_k} - b) - \sum_{m=1}^{M} r_k(w_m)(\frac{1}{2\lambda_k} - \frac{w_m^2}{2}) \tag{11}$$

$$\frac{\partial G}{\partial \pi_k} = -\frac{\alpha_k - 1}{\pi_k} - \sum_{m=1}^{M} \frac{r_k(w_m)}{\pi_k}, \tag{12}$$

where λ_k and π_k are the kth dimension of $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$.

The second term of $\frac{\partial G}{\partial \lambda_k}$ is a weighted sum of the difference between the squared model parameter and the variance of GM distribution. The first term is controlled by the GM hyper-parameters a and b.

Given fixed responsibility values, the minimizer for λ_k and π_k can be obtained from Equations (11) and (12). Deriving the update formula for λ_k is straightforward. Setting Equation (11) to zero gives the following equation.

$$\lambda_k = \frac{2(a-1) + \sum_{m=1}^{M} r_k(w_m)}{2b + \sum_{m=1}^{M} r_k(w_m) w_m^2}. \tag{13}$$

Here $2(a-1)$ and $2b$ work as smoothing terms which correspond to adding "pseudo" model parameters to the kth cluster.

Since $\sum_{m=1}^{M} r_k(w_m)$ corresponds to the responsibility of the kth Gaussian component. The magnitude of $\sum_{m=1}^{M} r_k(w_m)$ is the same as the number of model parameters (We use M to denote the number of model parameters). Inspired by this observation, b is set as a proportional function to M. a plays a similar role as b and fine-tunes the value of learned $\boldsymbol{\lambda}$. For a, the proportion with b affects the magnitude of the learned $\boldsymbol{\lambda}$. So a is set as a proportional function to b. But since $\sum_{m=1}^{M} r_k(w_m)$ plays the major role in the numerator of Equation (13), the setting of a is not so significant.

For mixing coefficient $\boldsymbol{\pi}$, deriving the update formula is a bit complex because the constraint $\sum_{k=1}^{K} \pi_k = 1$ must be satisfied. Thus a Lagrange multiplier has to be used here. The Lagrangian of the loss function is defined as follows.

$$L = G + \lambda_{lag}\left(\sum_{k=1}^{K} \pi_k - 1\right), \tag{14}$$

where λ_{lag} is the Lagrange multiplier. Set the derivatives of L with respect to π_k and λ_{lag} to zero

$$\frac{\partial L}{\partial \pi_k} = -\frac{\alpha_k - 1}{\pi_k} - \sum_{m=1}^{M} \frac{r_k(w_m)}{\pi_k} + \lambda_{lag} = 0 \tag{15}$$

$$\frac{\partial L}{\partial \lambda_{lag}} = \sum_{k=1}^{K} \pi_k - 1 = 0. \tag{16}$$

Solving Equations (15) and (16), the update formula for π_k can be given as follows.

$$\pi_k = \frac{\sum_{m=1}^{M} r_k(w_m) + (\alpha_k - 1)}{M + \sum_{j=1}^{K}(\alpha_j - 1)}, \tag{17}$$

where π_k is the kth dimension of $\boldsymbol{\pi}$.

This equation shows that α greatly affects the number of Gaussian components learned. If α is large, then most probably one Gaussian will be learned because all dimensions of $\boldsymbol{\pi}$ will be the same. Typically, α is set to the power of M (e.g., $M^{0.5}$).

Given Equations (9), (10), (17) and (13), responsibility values can be computed and then model parameters $\boldsymbol{w}$ can be updated. After the model parameters are updated, the values for GM parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ can be updated subsequently.

3.4 Lazy Update

Calculating $\boldsymbol{g}_{reg}$ (i.e., the second term of Equation (10)) and updating GM parameters $\boldsymbol{\pi}, \boldsymbol{\lambda}$ are time-consuming because the Gaussian Probability Density Function needs to be computed. This is the bottleneck of the algorithm.

In order to reduce the computational time, a lazy update method is employed for models with a large number of parameters. The intuition for the lazy update is explained as follows. Both $\boldsymbol{g}_{reg}$ and GM parameters $\boldsymbol{\pi}, \lambda$ do not change too much after the first few epochs. Consequently, they do not need to be updated in every iteration after the first few epochs.

Algorithm 1. Lazy Update for GM Regularization

```
   Input: $\boldsymbol{w}, \boldsymbol{\alpha}, a, b, \boldsymbol{\pi}, \boldsymbol{\lambda}, L, I_g, I_m, E, B$
1: initialize: $it \leftarrow 0, epoch\_it \leftarrow 0$
2: while not converged do
3:   Compute $\boldsymbol{g}_{ll}$
     /* E step */
4:   if $epoch\_it < E$ or $it\ mod\ I_m = 0$ then
5:     Compute responsibility function based on Equation (9)
6:     Compute $\boldsymbol{g}_{reg}$
7:   end if
8:   Compute $\frac{\partial G}{\partial w_m}$ based on Equation (10)
     /* M step */
9:   if $epoch\_it < E$ or $it\ mod\ I_g = 0$ then
10:    Compute $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ according to Equations (17) and (13)
11:  end if
     /* SGD step */
12:  $\boldsymbol{w}^{(it+1)} \leftarrow \boldsymbol{w}_j^{(it)}$ - $L\frac{\partial G}{\partial \boldsymbol{w}}$
13:  $it \leftarrow it + 1$
14:  if $epoch\_it\ mod\ B = 0$ then
15:    $epoch\_it \leftarrow epoch\_it + 1$
16:  end if
17: end while
```

Algorithm 1 shows the lazy update algorithm. Here, L is the learning rate for SGD and E is denoted as the number of the first few epochs when lazy update is not employed, I_g and I_m as the update interval for GM parameters and model parameters. $\boldsymbol{g}_{reg}$ is updated every iteration when the epoch number is less than E. Thereafter, $\boldsymbol{g}_{reg}$ is updated every I_m steps as shown in line 6 of Algorithm 1. Here, B is the number of mini-batches in the training set. When $\frac{\partial G}{\partial w_m}$ is calculated, the $\boldsymbol{g}_{reg}$ calculated in E-step is used . Since E-step is not carried out in every iteration, thus $\boldsymbol{g}_{reg}$ is not updated in every iteration. For GM parameters $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$, after E epochs, they are updated every I_g steps.

Algorithm 2. Sparse Update for GM Regularization

```
   Input: $\boldsymbol{w}, \boldsymbol{\alpha}, a, b, \boldsymbol{\pi}, \boldsymbol{\lambda}, L, I_g, E, B, N_B$
1: initialize: $it \leftarrow 0, epoch\_it \leftarrow 0$
2: initialize:
3: it = 0
4: while not converged do
5:   Compute $\boldsymbol{g}_{ll}$
6:   for $j \leftarrow 0$ to $N_B - 1$ do
7:     for $m$ such that $\boldsymbol{x}_{j,m} \neq 0$ do
8:       Compute responsibility function for $\boldsymbol{w}_m$ based on Equation (9)
9:       Compute $\boldsymbol{g}_{reg,m}$ for $\boldsymbol{w}_m$ based on Equation (18)
10:      $\boldsymbol{u}_m \leftarrow it$
11:    end for
12:  end for
     /* E step and M step */
13:  if $epoch\_it < E$ or $it\ mod\ I_g = 0$ then
14:    Compute responsibility function based on Equation (9)
15:    Compute $\boldsymbol{\pi}$ and $\lambda$ according to Equations (17) and (13)
16:  end if
     /* SGD step */
17:  $\boldsymbol{w}^{(it+1)} \leftarrow \boldsymbol{w}_j^{(it)}$ - $L\frac{\partial G}{\partial \boldsymbol{w}}$
18:  $it \leftarrow it + 1$
19:  if $epoch\_it\ mod\ B = 0$ then
20:    $epoch\_it \leftarrow epoch\_it + 1$
21:  end if
22: end while
```

3.5 Sparse Update

For sparse datasets, the first term of Equation (10) is sparse because it only touches dimensions where the input vector is non-zero. However, the second term is not sparse and it takes much time to calculate the responsibility values. Consequently, we do not calculate Equation (10) in every iteration. Instead, we perform a "sparse update" in order to reduce the computational time. For the mth model parameter w_m, we batch the dense "regularization update" up and then perform update on w_m only when the corresponding input dimension x_m is non-zero. We introduce a vector $\boldsymbol{u}$, in which $\boldsymbol{u}_m$ stores the last iteration when the mth model parameter was regularized. Consequently, if x_m is non-zero, the gradient for the mth model parameter with respect to G is derived as follows.

$$\frac{\partial G}{\partial w_m} = \frac{-\log p(\mathcal{D}|\mathbf{w})}{\partial w_m} + (it - \boldsymbol{u}_m)\sum_{k=1}^{K}(r_k(w_m)(\lambda_k w_m)), \tag{18}$$

where it is the current iteration, the second term of the equation is denoted as $\boldsymbol{g}_{reg,m}$. We multiply $(it - \boldsymbol{u}_m)$ by the original $\boldsymbol{g}_{reg}$ term in Equation (10) because we need to batch the regularization updates up.

Algorithm 2 shows the detailed update algorithm for sparse update. In this algorithm, N_B is the number of samples in a mini-batch and $\boldsymbol{x}_{j,m}$ is the mth dimension of the jth sample in the mini-batch. In line 7 and line 8, responsibility and $\boldsymbol{g}_{reg}$ are only calculated for the dimensions where the input vector is non-zero, which reduces a lot of computational time because the input vectors are sparse.

```
class GMRegularization(input_dim, a=None,
b=None, alpha=None, gm_num=None,
sparse=None, uptint=None):

def calcResponsibility(input, model_param)
def calRegGrad(input, model_param)
def uptGMParam(input, model_param)
```

Fig. 5. The GM regularization API.

For the update of GM parameters $\boldsymbol{\pi}, \lambda$, since we need to calculate the responsibility values for every model parameter, we choose to update them in every iteration for the first E epochs, and after that, we update the parameters every I_g iterations, which is the same as the lazy update algorithm described in Section 3.4.

4 Theoretical Time Analysis for Lazy Update

In this section, We give a mathematical formular for the calculation of total computational time and analyze the effect of GM parameter update interval I_g, model parameter update interval I_m and number of the first few epochs when the lazy update is not employed, E. Here we assume $I_g = I_m$.

From Algorithm 1, we can see there are four steps that consume computational time, namely computing $\boldsymbol{g}_{ll}$, E step, M step and SGD step. We use t_{ll} to indicate the time for computing $\boldsymbol{g}_{ll}$ for each iteration. For the other three steps, since we set $I_g = I_m$, responsibility function, $\boldsymbol{g}_{reg}$, $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ are updated in the same iteration. we use RNL and RL to indicate the ratio of total time for E step, M step and SGD step when lazy update is and is not employed to t_{ll}, respectively.

Assume P is the total number of epochs for convergence, B is the total number of mini-batches every epoch. For the same dataset and model, t_{ll}, B, E and P are fixed for different I_m. The total time T for different I_m and E is calculated as follows:

$$\begin{aligned} T = {} & t_{ll} \times B \times E \times (1 + RNL) \\ & + t_{ll} \times B \times (P - E) \\ & \times (1 + RL + (RNL - RL)/I_m). \end{aligned} \tag{19}$$

In Equation (19), the first term is the time for the first few epochs when the lazy update is not employed and the second term is the time for the remaining epochs. Equation (19) can be reformulated as follows:

$$\begin{aligned} T = t_{ll} \times B \times \Bigg(& P + P \times RL + (RNL - RL) \\ & \times \left(E + \frac{(P - E)}{I_m} \right) \Bigg). \end{aligned} \tag{20}$$

From Equation (20), we can see that E has a positive correlation with the total time T and I_m has a negative correlation with the total time T. If E is set small, I_m plays a significant role in reducing the total time T. On the contrary, if E is set large, the impact of I_m is greatly reduced. Practically, we set E to a relatively small value and set I_m to a relatively large value.

5 Regularization Tool with General APIs

As mentioned earlier and shown in Fig. 3, our proposed regularization method is designed as a general and flexible tool that can be easily integrated with a deep learning platform in the big data analytics pipeline.

In this section, we describe how we design the APIs for our regularization tool so that it is easy-to-use and general enough for different datasets, models and platforms.

Fig. 5 shows the APIs we design for the regularization tool. The parameters are explained as follows:

- input_dim: dimension of input
- a (optional): GM parameter a (default: 1+0.1*b)
- b (optional): GM parameter b (default: related to input_num, see Section 6.2.1)
- alpha (optional): GM parameter α (default: $input\_num^{0.5}$)
- gm_num (optional): initial number of Gaussian components (default: 4)
- sparse (optional): specify whether sparse update is employed (default: False)
- uptint (optinal): a tuple including GM and model parameter update intervals for lazy update. If either of these two values is larger than 1, lazy update is employed (default: (1,1))

Our GM Regularization tool is easy-to-use: for the parameters, only input_dim is required. Others are optional and GM regularization provides default values for them.

In the meantime, our GM Regularization tool provides different kinds of options for users, such as lazy update for complex models, sparse update for sparse datasets, tunable update intervals for users to trade off between model accuracy and computational time.

Key Functions. GM regularization tool provides three functions:

- calResponsibility(input, model_param): calculating responsibility values
- calcRegGrad(input, model_param): calculate $\boldsymbol{g}_{reg}$
- uptGMParam(input, model_param): updating GM parameters using EM algorithm

As shown in Fig. 2, with this tool, a model only needs to input model parameters and input vectors to GM regularization tool. In turn, our GM regularization tool returns $\boldsymbol{g}_{reg}$ after calculating responsibility values, $\boldsymbol{g}_{reg}$ and GM parameters. It is sufficiently general to support different kinds of models.

In order to demonstrate that our regularization is general for different platforms, we implemented our regularization tool with machine learning libraries in python as well as integrated our regularization tool with deep learning models on an open-source deep learning platform Apache Singa [22].

6 Experimental Results

In this section, we evaluate the performance of our proposed GM regularization method through experiments using standard benchmark datasets. The baseline regularization methods include L1-norm regularization (L1 reg) [28], L2-norm regularization (L2 reg) [15], Elastic-net (Elastic-net reg) [29], Huber-norm regularization (Huber reg) [30]. For all these baseline methods, both Grid Search (GS) [34] and Bayesian Optimization (BO) [35], [36], [37], [38], [39] are employed to find their best regularization strengths.

TABLE 2
UCI DataSet Characteristics

Dataset	# Samples	# Features	Feature Type
breast-canc	699	81	categorical
breast-canc-dia	569	30	continuous
breast-canc-pro	198	33	continuous
climate-model	540	18	continuous
congress-voting	435	32	categorical
conn-sonar	208	60	continuous
credit-approval	690	42	combined
cylindar-bands	541	93	combined
hepatitis	155	34	combined
horse-colic	368	58	combined
ionosphere	351	33	combined

6.1 Experiment Settings

Datasets. Experiments on both real datasets and machine learning/deep learning benchmark datasets are conducted.

1) CIFAR-10.[1] This is a benchmark dataset for image classification, which consists of 60,000 32×32 colour images in 10 classes, with 6,000 images per class. The training and test datasets contain 50,000 and 10,000 images respectively.

2) Hospital Frequent Admitter (Hosp-FA) Dataset. This is a real dataset from a hospital, which consists of inpatient visits (i.e., cases) of patients, including various types of medical features (i.e., diagnosis, demographics, etc.) On this dataset, we predict whether a patient is readmitted into the hospital within 30 days. This dataset consists of 1,755 patient samples each of which has 375 features. Dealing with medical features which have varying numbers of observations is challenging. In this dataset, there are different kinds of features, e.g., informative and less informative features. The distributions of model parameters that correspond to different kinds of features are quite different. To be specific, the distribution of model parameters that correspond to informative features usually has a larger variance while the distribution of model parameters that correspond to less informative features has a smaller variance.

3) UCI Machine Learning Repository Datasets. These are the benchmark datasets from the UCI machine learning repository [40]. These datasets are also referred to as UCI datasets. To avoid selection bias, we choose the first 11 (in alphabetical order) binary classification datasets (by skipping those datasets where all regularization methods achieve perfect performance). One-hot encoding method is used to transform the categorical features to binary features and we preprocess the continuous features to have zero mean and unit variance. Missing values in the categorical features are a separate class and missing values in the continuous features are imputed by the mean value.

Table 2 shows the characteristics of UCI datasets, where # Features is the number of features after one-hot encoding. The features of this dataset can be either, categorical, continuous or combined (the dataset has both categorical and continuous features). Most of these datasets have less than 1,000 samples, but the number of features is large because the ratios of # Features to # Samples of most datasets are more than 10 percent.

1. https://www.cs.toronto.edu/~kriz/cifar.html

4) Synthetic Dataset. The synthetic dataset is obtained by generating features from a multivariate Gaussian distribution with identity matrix and generating model parameters that sample from a Gaussian Mixture model with mixing coefficients [0.75, 0.25] and precision [200, 10]. The numbers of features and model parameters are 1,000 and the numbers of generated training samples and generated test samples are 10,000 and 40,000 respectively. In this dataset, each model parameter is connected with a feature, if the value of a model parameter is small, its corresponding feature is regarded as less informative because its contribution is small. From the GM mixing coefficients and precision, we can see that 75 percent features are less informative whereas 25 percent features are informative.

5) URL Reputation DataSet [41]. This is an extremely sparse dataset. It has around 2.4 million samples and 3.2 million features, less than 1 percent of the features are non-zero. The features of this dataset are anonymized and correspond to lexical and host-based features gathered for each URL. The task is to identify malicious URLs. We use 80 percent of samples for training and 20 percent samples for testing via stratified sampling.

Evaluation Metric. We use *accuracy*, i.e., the ratio of correct predictions, on the test dataset to measure the classification performance of regularization methods under examination.

Deep Learning Model Structures. Table 3 shows the model structures for our deep learning models. For Alex-CIFAR-10, the first convolutional layer filters the $32 \times 32 \times 3$ input images with 32 kernels of size $5 \times 5 \times 3$ with a stride of 1 pixel. The second convolutional layer has 32 kernels of size $5 \times 5 \times 32$ and the third convolutional layer has 64 kernels of size $5 \times 5 \times 32$. The last layer is the 10-way fully-connected softmax layer. Pooling layer, RELU layer and LRN layer [7] are inserted between convolutional layers. The number of model parameters is 89440.

The second model is the twenty-layer ResNet [10], which is a representative model with a large number of stacked layers. We experiment on this model to study the behavior of our GM regularization in complicated and deep neural networks. The inputs of the network are 32×32 images, with the per-pixel mean subtracted. The first layer is 3×3 convolutions, followed by a stack of $6\,n$ ($n = 3$ here) layers with 3×3 convolutions with 16, 32, 64 filters respectively. The subsampling is performed by convolutions with stride 2 and the network ends with a 10-way softmax function. In total, there are 20 stacked weighted layers. The number of model parameters is 270896.

For these two models, the momentum is set to 0.9, the learning rate is 0.001 for Alex-CIFAR-10 and 0.1 for ResNet. For other hyper-parameters such as stopping criteria and weight decay, they are set to be the same as [7] and [10] respectively.

Data augmentation is performed for ResNet but not for Alex-CIFAR-10. By doing this, we aim to show the effectiveness of our proposed regularization both on a simple structure neural network without any data augmentation and on a complex neural network with data augmentation.

Environment. Logistic Regression (LR) is implemented using python and Convolutional Neural Network (CNN) is implemented on Apache Singa [22]. Experiments are run on a server equipped with Intel i7-4930K CPU and three GTX Titan X GPU cards.

TABLE 3
Deep Learning Model Structures

Model	Layers						
Alex-CIFAR-10	$\left.\begin{matrix}5\times5,32\\ \text{MaxPooling}\\ \text{RELU}\\ \text{LRN}\end{matrix}\right\}\times 1$	$\left.\begin{matrix}5\times5,32\\ \text{RELU}\\ \text{AvgPooling}\\ \text{LRN}\end{matrix}\right\}\times 1$	$\left.\begin{matrix}5\times5,64\\ \text{RELU}\\ \text{AvgPooling}\end{matrix}\right\}\times 1$	Softmax			
ResNet	$\left.\begin{matrix}3\times3,16\\ \text{BN}\\ \text{RELU}\end{matrix}\right\}\times 1$	$\left.\begin{matrix}3\times3,16\\ \text{BN}\\ \text{RELU}\\ 3\times3,16\end{matrix}\right\}\times 3$	$3\times3,32$	$\left.\begin{matrix}3\times3,32\\ \text{BN}\\ \text{RELU}\\ 3\times3,32\end{matrix}\right\}\times 3$	$3\times3,64$	$\left.\begin{matrix}3\times3,64\\ \text{BN}\\ \text{RELU}\\ 3\times3,64\end{matrix}\right\}\times 3$	Avgpooling Softmax

6.2 Case Study: Adaptive Regularization Tool on Deep Learning Models

In this section, we show a case study on how our adaptive regularization tool can be used to learn Gaussian Mixtures in deep learning models and how appropriate regularization is generated by our adaptive regularization tool. Alex-CIFAR-10 and ResNet are used in this section.

6.2.1 *Easy Setting of GM Hyper-Parameters*

The setting of GM hyper-parameters is straightforward. It can be automatically defined by the characterisitcs of the dataset. The same setting of GM prior hyperparameters a, b, α and K (initial number of Gaussian components) is used for different layers to adaptively learn their best GMs for regularization.

Initial Number of Gaussian Components. The initial number of Gaussian components, K, is fixed to 4. An interesting observation is that our GM regularization tends to learn one or two Gaussian components for different models eventually, because some of the Gaussian components are gradually merged to one during the GM learning process. The final number of Gaussian components (whether one or two) is learned automatically. We experiment with different initial number of Gaussian components and find four to be the best according to the experimental results shown in Section 6.7.

Hyper-Parameters a, b and α. b is set to γM, where M is the number of model parameters for each layer, and the parameter grid for γ is [0.0002, 0.0005, 0.001, 0.002, 0.005, 0.01, 0.02, 0.05]. As mentioned in Section 3.3.3, a is not a significant parameter, it is set to $1 + 10^{-2}\ b$ or $1 + 10^{-1}\ b$. For α, it is set to $M^{0.5}$.

TABLE 4
Learned Regularization for Alex-CIFAR-10

GM Reg		
Layer Name	$\boldsymbol{\pi}$	$\boldsymbol{\lambda}$
conv1/weight	[0.216, 0.784]	[10.727, 835.959]
conv2/weight	[0.019, 0.981]	[0.640, 1904.024]
conv3/weight	[0.013, 0.987]	[0.095, 2017.931]
dense/weight	[0.036, 0.964]	[3.939, 1277.578]
GS L2 Reg (expert-tuned)		
Layer Name	$\boldsymbol{\pi}$	$\boldsymbol{\lambda}$
conv1/weight	[1.000]	[200.000]
conv2/weight	[1.000]	[200.000]
conv3/weight	[1.000]	[200.000]
dense/weight	[1.000]	[50000.000]
BO L2 Reg		
Layer Name	$\boldsymbol{\pi}$	$\boldsymbol{\lambda}$
conv1/weight	[1.000]	[429.200]
conv2/weight	[1.000]	[63.950]
conv3/weight	[1.000]	[500.000]
dense/weight	[1.000]	[12500.000]

6.2.2 *Learned GM Regularization*

This section shows the learned GMs for Alex-CIFAR-10 and ResNet respectively. In terms of baseline methods, for L2 Reg using GS (GS L2 Reg), $\boldsymbol{\lambda}$ is manually set by experts using grid search [10], [35], while for L2 Reg using BO (BO L2 Reg), the search space for regularization strength is from 0.00001 to 1.0 and from 0.0000005 to 0.05 for Alex-CIFAR-10 and ResNet respectively.

Tables 4 and 5 show the learned GMs under the best parameter setting. Since ResNet has 20 layers, only the learned $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ for the representative layers are shown here. Layers with asterisk indicate that there are several other layers that have very similar $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$. Compared with both GS L2 Reg and BO L2 Reg, GM Reg learns more than one Gaussian for each layer by adaptively learning different GM parameters $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ for different layers automatically. This result confirms that our GM regularization is adaptive to the model parameter distributions of different layers so that the best regularization for each layer can be learned. The two Gaussian components learned for each layer correspond to informative and less informative features respectively. The learned $\boldsymbol{\lambda}$ for Alex-CIFAR-10 is larger than that of ResNet, which indicates that Alex-CIFAR-10 needs stronger regularization; this is due to lack of Batch-normalization (BN) layers in the Alex-CIFAR-10. For ResNet, many layers have similar $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ because of the GM initialization method which is discussed in detail in Section 6.8. According to [42], the initialization distributions of model parameters between two convolutional layers are the same if the two layers have the same number of filters. As shown in Table 3, there are three stacks of filters, 16, 32 and 64 respectively. The layers in each stack have the same initialized Gaussian variance, leading to similar learned GM parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$.

6.2.3 *Comparison on Accuracy*

In this experiment, we study the effects of GM regularization on the predictive ability of deep learning models. In terms of baseline methods, the settings are the same as those

TABLE 5
Representative Learned Regularization for ResNet

GM Reg		
Layer Name	π	λ
conv1/weight	[0.377, 0.623]	[0.301, 8.106]
2a-br1-conv1/weight	[0.066, 0.934]	[0.149, 22.620]
2a-br1-conv2/weight*	[0.062, 0.938]	[0.145, 23.016]
3a-br2-conv/weight	[0.152, 0.848]	[0.195, 22.010]
3a-br1-conv1/weight	[0.047, 0.953]	[0.141, 22.824]
3a-br1-conv2/weight*	[0.032, 0.968]	[0.121, 23.617]
4a-br2-conv/weight	[0.068, 0.932]	[0.157, 22.733]
4a-br1-conv1/weight	[0.023, 0.977]	[0.114, 23.868]
4a-br1-conv2/weight*	[0.016, 0.984]	[0.109, 24.396]
ip5/weight	[0.230, 0.770]	[0.865, 6.979]
GS L2 Reg		
Layer Name	π	λ
All Layers	[1.000]	[50.000]
BO L2 Reg		
Layer Name	π	λ
conv1/weight	[1.000]	[15.950]
2a-br1-conv1/weight	[1.000]	[24.850]
2a-br1-conv2/weight*	[1.000]	[25.000]
3a-br2-conv/weight	[1.000]	[25.000]
3a-br1-conv1/weight	[1.000]	[25.000]
3a-br1-conv2/weight*	[1.000]	[21.250]
4a-br2-conv/weight	[1.000]	[25.000]
4a-br1-conv1/weight	[1.000]	[7.050]
4a-br1-conv2/weight*	[1.000]	[25.000]
ip5/weight	[1.000]	[25.000]

TABLE 6
Accuracy on Deep Learning Model

	Alex-CIFAR-10	ResNet
no regularization	0.777	0.901
GS L2 Reg	0.822(expert-tuned)	0.909
BO L2 Reg	0.824	0.912
GM regularization	**0.830**	**0.921**

in Section 6.2.2. The results under the best parameter setting on the test dataset are summarized in Table 6.

For the Alex-CIFAR-10, the GS L2 Reg is carefully tuned by the human expert, where the strengths of regularization are different for different layers. From Table 6, we can see GS L2 Reg improves the accuracy from 0.777 (without any regularization) to 0.822, which indicates the importance of regularization in deep learning models. For BO L2 Reg, it further improves GS L2 Reg which is consistent with the observation in [35] for the following two reasons. First, BO is able to obtain different regularization strengths for different layers as shown in Table 4. Second, BO finds the best regularization strengths in a more principled way by only evaluating the most promising regularization strengths. Our GM regularization further improves the BO Reg from 0.824 to 0.83 in terms of accuracy. This result confirms the advantage of adaptive GM regularization over both GS L2 Reg and BO L2 Reg.

For the ResNet, since the BN layer serves as a form of regularization, the improvement of L2 Reg over no regularization is not so dramatic as that in Alex-CIFAR-10. Similar to Alex-CIFAR-10, BO L2 Reg is able to improve GS L2 Reg. In the meantime, our GM Reg further improves BO L2 Reg from 0.912 to 0.920. This result is nearly the same as the result of a ResNet with 1202 layers [10], which confirms the effectiveness of our proposed GM regularization.

6.3 Comparison on Small Dataset

In this section, we evaluate our GM regularization with Logistic Regression model on one real hospital readmission dataset and 11 machine learning benchmark datasets. Since the number of samples is small, for each dataset, we run a 5-fold cross-validation. For all the baseline methods, we use both GS and BO to find the best regularization strength. For GS, the grid of regularization strength is $[10^{-4}, \ldots, 10^{3}]$, which is consistent with [30]. For BO, the search space for regularization strength is from 10^{-4} to 10^{3} accordingly.

The average and standard errors of accuracies under the best parameter setting on the test dataset are shown in Table 7. The highest average accuracy results are indicated in bold. The result shows that the GM regularization method outperforms the other four regularization methods optimized by both GS and BO in 9 out of 12 datasets and achieves the same best performance as other baseline methods in two of the other three datasets. GM regularization does not prevail against the baseline methods only in the breast-canc-dia dataset. For this dataset, our adaptive GM regularization is comparable with the baselines and our standard errors are smaller than Huber Reg. These results again provide clear evidence for the performance benefits of our adaptive GM regularization.

Compared with L1-norm regularization method, GM regularization achieves better results in all the datasets. This is because L1-norm regularization tends to reduce the model parameters of less informative features to zero, which totally removes the effect of these features. On the contrary, GM regularization method learns a small variance Gaussian component for these features so that the effects of these features are retained instead of being removed.

L2-norm regularization imposes the same regularization strength for all the features, so it is likely that the informative features with large model parameter values are over-regularized. In contrast, GM regularization does not regularize these informative features strongly since a large variance Gaussian component is learned to exert weak regularization.

Both Elastic-net and Huber-norm regularization tradeoff between L1-norm and L2-norm regularization. For Elastic-net, it uses a parameter *l1_ratio* to control the proportion of L1-norm regularization and L2-norm regularization. By tuning this parameter, Elastic-net enables L1-norm or L2-norm regularization to dominate. As such, Elastic-net achieves better results than L1-norm regularization and L2-norm regularization for nearly all the datasets in terms of both GS and BO.

Huber-norm regularization is in the form of a piecewise function; that is, Huber-norm regularization is L2-norm regularization for small model parameters and it is L1-norm regularization for large model parameters. The two parameters μ and λ control the threshold between L1-norm and L2-norm regularization. By tuning the threshold value, Huber-norm regularization can tradeoff between L1-norm and L2-norm regularization to dominate these two regularization methods in terms of both GS and BO.

Comparing BO with GS, we can find BO achieves better results in most datasets, due to the fact that BO is able to

TABLE 7
Comparison on Accuracies and Standard Errors

Dataset	Method	L1 Reg	L2 Reg	Elastic-net Reg	Huber Reg	GM Reg
Hosp-FA	GS	0.844 ± 0.023	0.842 ± 0.021	0.847 ± 0.022	0.845 ± 0.022	**0.848 ± 0.021**
	BO	0.846 ± 0.023	0.842 ± 0.022	0.847 ± 0.020	0.848 ± 0.022	
breast-canc	GS	0.963 ± 0.012	0.969 ± 0.012	**0.970 ± 0.011**	**0.970 ± 0.011**	**0.970 ± 0.011**
	BO	0.964 ± 0.011	**0.970 ± 0.011**	**0.970 ± 0.011**	0.970 ± 0.014	
breast-canc-dia	GS	0.972 ± 0.012	0.979 ± 0.008	0.981 ± 0.007	**0.982 ± 0.011**	0.981 ± 0.007
	BO	0.972 ± 0.008	0.981 ± 0.007	**0.982 ± 0.011**	0.981 ± 0.007	
breast-canc-pro	GS	0.818 ± 0.044	0.834 ± 0.050	0.839 ± 0.040	0.834 ± 0.051	**0.859 ± 0.036**
	BO	0.819 ± 0.046	0.834 ± 0.051	0.844 ± 0.040	0.839 ± 0.041	
climate-model	GS	0.965 ± 0.010	0.963 ± 0.013	0.965 ± 0.010	0.967 ± 0.011	**0.969 ± 0.011**
	BO	0.965 ± 0.010	0.967 ± 0.011	0.967 ± 0.011	0.967 ± 0.011	
congress-voting	GS	0.968 ± 0.015	0.970 ± 0.015	0.972 ± 0.017	0.972 ± 0.013	**0.977 ± 0.018**
	BO	0.972 ± 0.013	0.970 ± 0.015	0.975 ± 0.012	0.975 ± 0.017	
conn-sonar	GS	0.803 ± 0.034	0.832 ± 0.042	0.837 ± 0.050	0.830 ± 0.052	**0.847 ± 0.057**
	BO	0.813 ± 0.037	0.837 ± 0.055	0.832 ± 0.055	0.832 ± 0.063	
credit-approval	GS	0.867 ± 0.032	0.868 ± 0.022	0.875 ± 0.032	0.874 ± 0.028	**0.878 ± 0.033**
	BO	0.868 ± 0.028	0.868 ± 0.018	0.877 ± 0.037	0.877 ± 0.035	
cylindar-bands	GS	0.782 ± 0.038	0.791 ± 0.017	0.795 ± 0.020	0.791 ± 0.023	**0.798 ± 0.016**
	BO	0.784 ± 0.023	0.789 ± 0.020	0.797 ± 0.020	0.795 ± 0.018	
hepatitis	GS	0.866 ± 0.067	0.898 ± 0.040	**0.904 ± 0.038**	0.898 ± 0.040	**0.904 ± 0.038**
	BO	0.865 ± 0.045	0.898 ± 0.040	**0.904 ± 0.038**	0.891 ± 0.048	
horse-colic	GS	0.835 ± 0.064	0.842 ± 0.040	0.864 ± 0.040	0.859 ± 0.060	**0.870 ± 0.047**
	BO	0.835 ± 0.046	0.845 ± 0.028	0.867 ± 0.037	0.862 ± 0.049	
ionosphere	GS	0.906 ± 0.029	0.903 ± 0.028	0.909 ± 0.027	0.909 ± 0.037	**0.920 ± 0.024**
	BO	0.909 ± 0.033	0.898 ± 0.027	0.903 ± 0.025	0.912 ± 0.030	

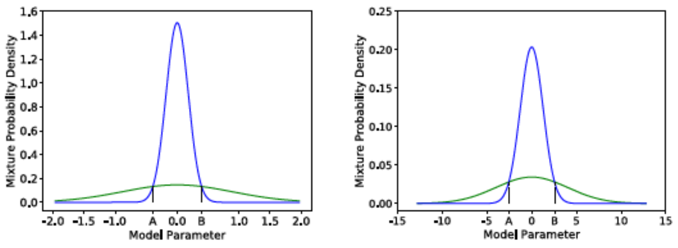
explore different regularization strengths in a more principled and automatic way. Only the most promising regularization strengths will be evaluated, thus higher-quality regularization strengths can be obtained.

By assuming a GM prior distribution over model parameters, our adaptive GM regularization method models the model parameter prior better, thus imposing more appropriate regularization. It learns two Gaussian components as the model parameter prior distribution to regularize both informative and less informative features, imposing different strengths of regularization on these two kinds of features, which results in better performance than the baselines optimized by both GS and BO. Details on the learned Gaussian components are discussed in Section 6.4.

6.4 Learned Gaussian Components for Small Datasets

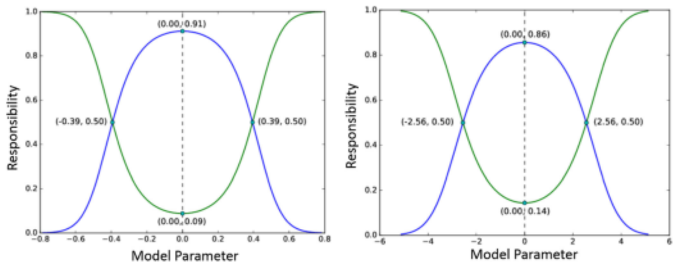
In this section, we evaluate the adaptively learned Gaussian components for two representative small datasets, horse-colic and conn-sonar datasets. Fig. 6 shows the mixture probability density for different values of model parameters for two Gaussian components in both datasets. For both of these two datasets, two Gaussian components are learned. The horizontal axis indicates the model parameter value w_m and the vertical axis shows the mixture probability density, $\pi_k p_k(w_m)$, the numerator in Equation (9). Points A and B labeled in the figure show where two Gaussian components have the same mixture probability density. In both figures, the small variance Gaussian component dominates in the area near zero. When model parameters are getting beyond A/B points, the large variance Gaussian component begins to dominate. This shows that our GM regularization exerts strong effects on small value model parameters which correspond to less informative features and exerts less strong regularization on large value model parameters which correspond to informative features. Another observation is that the Gaussian shapes of the two figures differ a lot, which illustrates that our adaptive GM regularization method can learn different GM distributions for different datasets. The variance of the small variance Gaussian in horse-colic dataset is much smaller than that in conn-sonar dataset, this shows that the model parameters corresponding to the less informative features in the horse-colic dataset are much smaller and need to be regularized more strongly.

Fig. 7 shows the change of responsibility values along with the change of model parameter values for the two learned Gaussian components in both datasets. The blue and green lines show the large precision Gaussian component and small precision Gaussian component respectively. The



(a) Horse-colic dataset (b) Conn-sonar dataset

Fig. 6. The horizontal axis indicates the model parameter and the vertical axis shows the mixture probability density. The GM parameters for horse-colic dataset are π = [0.326 0.674], λ = [1.270 31.295], and for conn-sonar dataset are π = [0.345, 0.655], λ = [0.062, 0.607].



(a) Horse-colic dataset (b) Conn-sonar dataset

Fig. 7. The horizontal axis indicates the model parameter and the vertical axis shows the responsibility. The GM parameters for horse-colic dataset are $\boldsymbol{\pi} = [0.326\ 0.674]$, $\boldsymbol{\lambda} = [1.270\ 31.295]$, and for conn-sonar dataset are $\boldsymbol{\pi} = [0.345, 0.655]$, $\boldsymbol{\lambda} = [0.062, 0.607]$.

responsibility values of these two Gaussian components sum up to 1. The two points in the vertical dotted line that passes through the origin show the responsibility values of these two Gaussian components for zero value model parameters. From the figures, we see that the responsibility value of the large precision Gaussian component is much larger than that of the small precision Gaussian component, which shows that the large precision Gaussian component dominates in the area near zero. Then the responsibility value of the large precision Gaussian component decays fast. Two curves cross when the absolute values of the model parameters are ± 0.39 and ± 2.56 for Horse-colic and Conn-sonar datasets respectively. When the absolute values of model parameters are beyond 0.39 or 2.56, the small precision Gaussian component begins to dominate and exert weaker regularization on the model parameters with large values. This shows that our GM regularization exerts strong effects on small value model parameters and exerts weaker regularization on large value model parameters, which is consistent with the observations made in Fig. 6.

6.5 Comparison on Synthetic Dataset

This section demonstrates the effectiveness of GM regularization using a supervised method. The generation of the synthetic dataset is explained in Section 6.1. As introduced in Section 1, our GM regularization learns a GM distribution using the intermediate model parameters during training to approximate the actual distribution of the model parameters while imposes appropriate regularization on the model parameters. In this section, we show two observations: (1) The GM learned by our GM regularization is a good approximation of the actual distribution of the model parameters. (2) The distribution of the final learned model parameters regularized by GM regularization resembles the actual distribution of the model parameters more than other baseline regularization methods. For this synthetic dataset, we add some noise when generating the labels. As the accuracy may not be convincing because of the noise, we also present the cross entropy loss that compares the predicted probability with the label (label loss) and true probability (probability loss).

The same as Section 6.3, both GS and BO are employed to find the best regularization strengths for baseline methods. In this experiment, for grid search, the grid of regularization strength is $[10^{-4}, \ldots, 10^{3}]$, while for BO Reg, the search space for regularization strength is from 10^{-4} to 10^{3} accordingly. The results under the best parameter setting on the test dataset are shown in Table 8. From the table, we can see that our GM regularization achieves the best results in terms of all

TABLE 8
Comparison on Synthetic Dataset

Regularization	Method	Accuracy	Label Loss	Probability Loss
L1 reg	GS	0.8363	14077.36	14117.00
	BO	0.8365	14046.24	14081.12
L2 reg	GS	0.8365	14091.44	14136.12
	BO	0.8365	14088.04	14133.96
Elastic-net reg	GS	0.8363	14079.72	14119.60
	BO	0.8363	14015.56	14054.36
Huber reg	GS	0.8368	14064.68	14104.20
	BO	0.8373	14007.16	14044.20
GM reg		**0.8377**	**13995.04**	**14026.16**

TABLE 9
Learned Gaussian Components on Synthetic Dataset

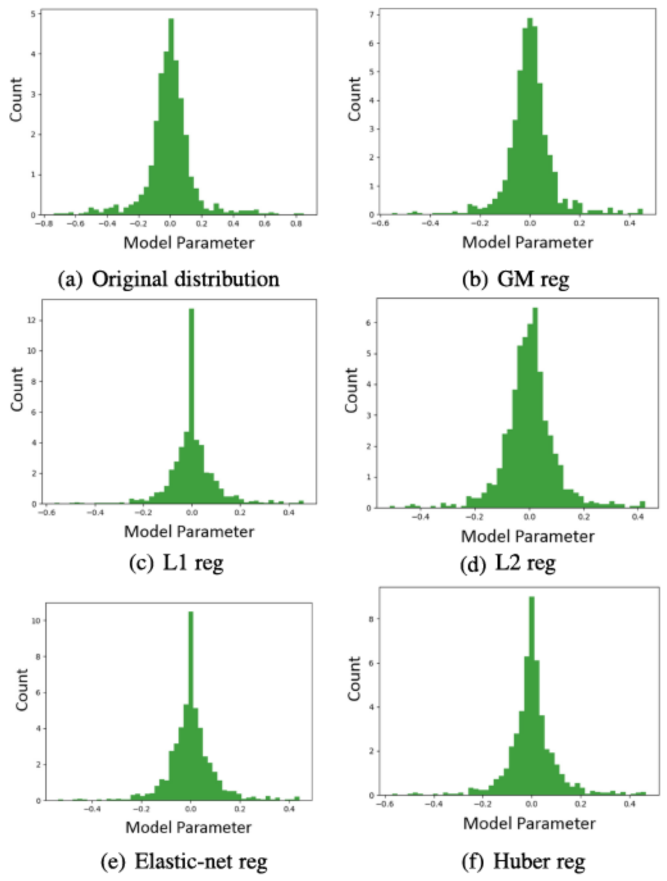
original $\boldsymbol{\pi}$	original $\boldsymbol{\lambda}$
[0.75, 0.25]	[200, 10]
learned $\boldsymbol{\pi}$	learned $\boldsymbol{\lambda}$
[0.76, 0.24]	[277.65, 27.23]

metrics. L2 Reg performs the worst in label loss and probability loss than other baseline methods in terms of both GS and BO because the generated dataset has a large proportion of less informative features, i.e., 75 percent features are less informative. Note that L2 Reg does not perform as good as L1 reg, which regularizes the less informative features strongly, in terms of both GS and BO. Huber Reg achieves better results than other methods in terms of both GS and BO except for GM regularization because of its ability to trade off between L1 and L2 as mentioned in Section 6.3.

Comparing BO with GS, we can find BO achieves better results than GS, which is consistent with the observations in Section 6.3.

Table 9 shows the mixing coefficient $\boldsymbol{\pi}$ and precision $\boldsymbol{\lambda}$ of the learned GM. The table shows that our GM regularization learns two Gaussian components with $\boldsymbol{\pi} = [0.24, 0.76]$, $\boldsymbol{\lambda} = [27.23, 277.65]$. The learned mixing coefficient $\boldsymbol{\pi}$ is nearly the same as the original $\boldsymbol{\pi}$. Both of the learned Gaussian components have higher precisions than the original GM because the regularization tends to "pull" the model parameters closer to zero, leading to the learned GM having smaller variances and larger precisions.

In order to show the distribution of the model parameters regularized by our GM regularization better resembles the actual model parameter distribution than other baseline regularization methods, Fig. 8 shows the distribution of the original model parameters and the distributions of the model parameters regularized by different regularization methods optimized by BO. We choose BO here because BO achieves better results than GS as shown in Table 8. From the figure, we can see that more than 8 model parameters regularized by L1 reg, Elastic-net Reg and Huber Reg are nearly zero, more than those of original distribution, GM Reg and L2 Reg. This is attributed to the less informative features. L1 reg, Elastic-net Reg and Huber Reg impose strong regularization on less informative features, reducing the values of their corresponding model parameters to zero or nearly zero in order to restrict the effects of these less informative features. In this dataset,



(a) Original distribution (b) GM reg (c) L1 reg (d) L2 reg (e) Elastic-net reg (f) Huber reg

Fig. 8. Histogram of model parameters for original distribution and different regularization methods.

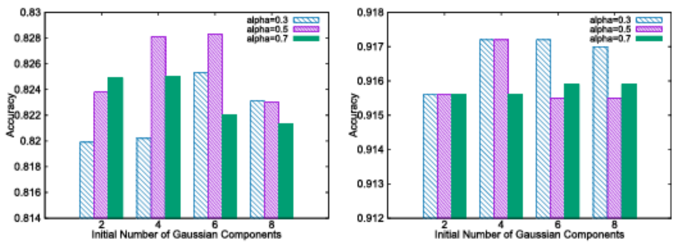
because most features, i.e., 75 percent features, are less informative, strongly regularizing these less informative features removes the beneficial effects of these features, which is harmful to the model. Comparing L2 Reg with GM Reg, we can observe that for L2 Reg, less model parameters are beyond ± 0.4 but there are more model parameters around zero due to the regularization strength. For L2 reg, it is a special case of the GM Reg with number of Gaussian components equaling to one. Consequently, the strength of regularization imposed by L2 Reg is the same for all model parameters. In this dataset, the optimal learned precision for L2 Reg is 91.56, which results in overly weak regularization for small value model parameters around zero and overly strong regularization for large value model parameters. On the contrary, GM Reg learns two precisions, namely 277.65 and 27.23, regularizing small value model parameters and large value model parameters respectively. This result confirms that our GM regularization is able to impose appropriate regularization strengths to different kinds of features.

6.6 Comparison on URL Reputation Dataset

In this section, we show the effectiveness of sparse update method proposed for extremely high-dimensional and sparse datasets. In this dataset, our sparse GM regularization achieves the same best accuracy as the baseline, 0.9870, which is consistent with [41]. Table 10 shows the computational time (in seconds) for GM regularization and sparse GM regularization in each iteration. From the table, we can see that the sparse update method decreases the computational time by about 20 times, which confirms the effectiveness of our proposed sparse update algorithm.

TABLE 10
Computational Time on URL Reputation DataSet

Method	GM Reg	sparse GM Reg
Time (Sec)	4.000	0.206



(a) Alex-CIFAR-10 (b) ResNet

Fig. 9. Accuracy for different initial numbers of Gaussian components.

6.7 Experiment on Initial Number of Gaussian Components

In this section, we show the performance of the GM regularization method by varying the initial number of Gaussian components. Since α also affects the final number of Gaussian components learned, we are therefore interested in investigating the effects of initial number of Gaussian components with respect to different α values. Fig. 9 shows the accuracy of different combinations of initial number of Gaussian components and α. From the figure, we can see that when the initial number of Gaussian components is two, the results are worse than those of four or six; when the initial number of Gaussian components is more than six, either the results are becoming worse or there is no significant improvement. The reason is explained as follows. When the initial number of Gaussian components is two, all the model parameters will be assigned to the two Gaussian components in the beginning. Since in the end, at most two Gaussian components are learned, the initial assignment of the model parameters largely determines the final assignment of model parameters when the model converges, which is not appropriate. On the contrary, if we set the initial number of Gaussian components to be four or six in the beginning, these initial Gaussian components can be gradually merged into two Gaussian components. However, if the initial number of Gaussian components is set to more than six, the accuracy decreases because the model is more difficult to converge to two Gaussian components if we start with too many Gaussian components in the beginning. Comparing four with six, we choose four because having more Gaussian components incurs more computation cost and it is more difficult for the model to converge.

6.8 Effectiveness of Proposed Initialization Methods for GM

It is well known that fitting GM can be very sensitive to poor initial conditions. In this section, several proposed initialization methods for GM are compared. For these proposed methods, the initialization of GM is related to the initialization of model parameters. In both Deep Learning model and Logistic Regression model, the model parameters are initialized with a

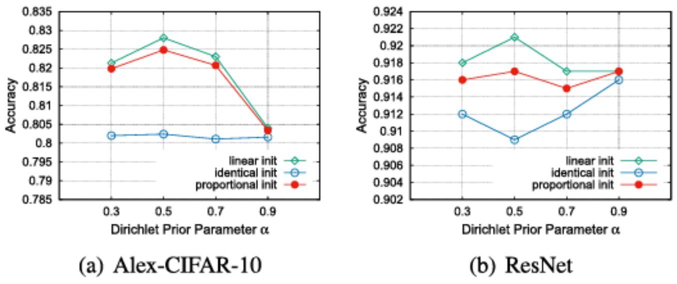


Fig. 10. Accuracy for different alpha values and initialization methods.

TABLE 11
Average Accuracy for Different Initialization Methods

Method	Alex-CIFAR-10	ResNet
linear	**0.819**	**0.918**
identical	0.802	0.912
proportional	0.817	0.916

zero-mean Gaussian distribution. The variances of different components of the initialized GM should be larger than the variance of the initialized model parameters so that the initial regularization is not too strong. We shall consider three initialization methods, namely identical, linear and proportional. For ease of explanation, in the remaining of this paragraph, we work with precision, the inverse of the variance. In the identical method, the precisions of different GM components are set identically to min. For ResNet, since the precisions of each layer's initialized model parameters are different, min is set to one-tenth of the initialized model parameter precisions. For other models, since the precisions of initialized model parameters are 100, all the min values are set to 10. In the second method, linear initialization, the precisions of the K initial GM components are linearly spaced between $[min, K \times min]$. The third initialization method is proportional initialization. In this method, the precision of the latter GM component is set to be two times the precision of the former component. The precision of the first GM component is min. Both linear and proportional methods cause the initial responsibility of different GM components to be different.

As mentioned in Section 2.3, α also affects the number of Gaussian components learned. We therefore investigate the effects of GM initialization methods with respect to different α values. Fig. 10 shows the accuracy of different combinations of GM initialization methods and α. The results show, for both Alex-CIFAR-10 and ResNet, linear and proportional initialization methods perform far better than identical initialization method. Table 11 shows the average accuracy of different GM initialization methods over different α values. The average accuracy of linear and proportional initialization methods are also far better than identical initialization method, mainly due to the final state. For Alex-CIFAR-10 and ResNet, the final state of the learned GM is two Gaussian components. If the GM components have different variances initially, they will converge to the final state faster. Another observation is that linear initialization is better than proportional initialization, because linear initialization method generates Gaussian components that are more scattered. When α is set to 0.5, we get the best performance as the GM learns multiple Gaussian components that lead to faster convergence.

6.9 Effectiveness of Lazy Update

In this experiment, we investigate the effect of GM parameter update interval I_g, model parameter update interval I_m and number of the first few epochs when the lazy update is not employed, E.

6.9.1 *Performance of Update Interval Values*

Figs. 11a and 11b show the training elapsed time with respect to the number of epochs for different I_m values and the baseline (L2 Reg). In this experiment, we set $I_g = I_m$ and E to two so that after two epochs, the increase of time is mainly due to lazy update. The results show that all six settings grow linearly in time as the number of epochs increases, which confirm the effectiveness of our proposed lazy update algorithm. We can observe that the algorithm with $I_m = 1$, where no lazy update is employed, takes the longest time to converge and the algorithm with $I_m = 50$ takes the shortest. This is because the algorithm with a larger I_m updates GM parameters and model parameters less frequently. Fig. 11c shows the convergence time for different I_m values and the baseline (L2 Reg). The number of epochs for convergence is 160 for Alex-CIFAR-10 and 200 for ResNet. In this experiment, we set $I_m = I_g$. Among the six settings, the algorithm with I_m=1 takes the longest time and the algorithm with I_m=50 takes the shortest. This is consistent with the observations made in Figs. 11a and 11b. For both Alex-CIFAR-10 and ResNet, algorithm with I_m=50 takes almost one fourth the time of the algorithm with I_m=1, where no lazy update is employed. Fig. 12 shows the accuracy of different I_m values for both models. From the figure, we can see that even if GM parameters and model parameters are updated less frequently, there is no significant

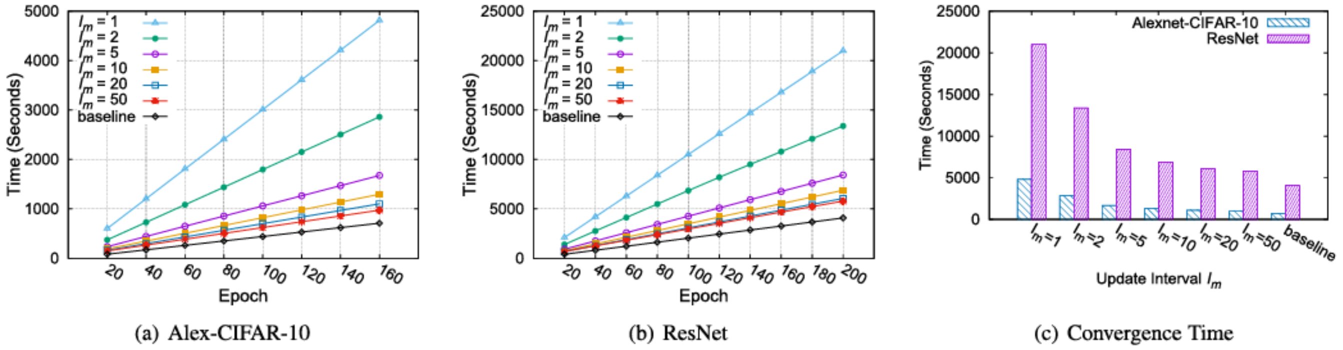


Fig. 11. Time for different update interval values.

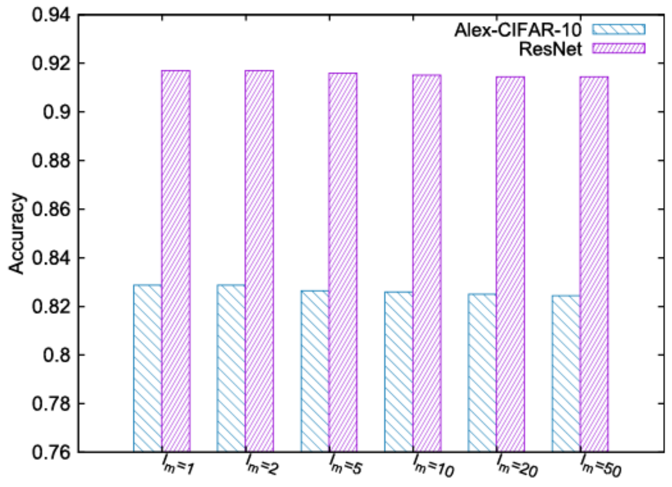


Fig. 12. Accuracy for different update interval values for Alexnet and Resnet.

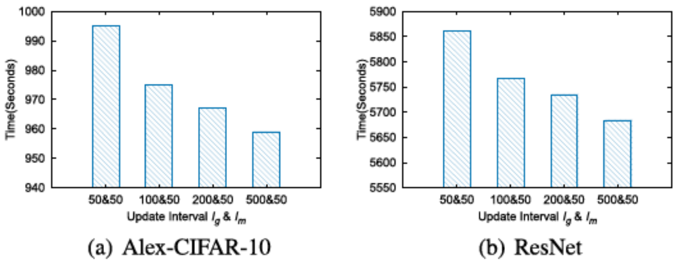


Fig. 13. Time for different combinations of I_g and I_m.

drop in accuracy. This again shows the effectiveness of lazy update algorithm.

6.9.2 Performance of GM Parameter Update Interval Values

Another factor that can further reduce time is I_g because the update of GM parameters includes calculating the responsibility value as well as calculating new λ and π using the high-dimensional model parameter vector, which is quite time-consuming. Considering the fact that the GM parameters converge faster than the model parameters, we set I_g larger than I_m. Figs. 13a and 13b show the convergence time for different combinations of I_g and I_m, where I_m is fixed to 50 and I_g is increased from 50 to 500. Fig. 13 shows that the convergence time can be further reduced if I_g is increased.

6.9.3 Performance of E Values

As mentioned in Section 3.4, the number of the first few epochs when the lazy update is not employed, E, is another factor that affects the time and accuracy. Figs. 14a and 14b show the training elapsed time with respect to epochs for different E values and baseline (L2 Reg). The results show that the lazy update algorithm takes more time for computation of each epoch before E epochs, since updating model parameters and GM parameters each step consumes more time than not updating them. After 70 epochs, we can observe the algorithm with E=50 takes the most time for computation while the algorithm with E=1 takes the least. This is because the algorithm with larger E takes more time in the first E epochs when lazy update is not employed. Fig. 14c shows the convergence time for different E values and baseline. The result shows that the decrease of time is proportional to the decrease of E. When E is decreased to 1, the convergence time consumed is only about 70 percent the time for the algorithm with E = 50. Fig. 15 shows the accuracy of different E values for both models. From the figure, we can see that even smaller E values do not degrade the final accuracy too much. By choosing a relatively small E value, we can obtain a high-performance model within a short training time.

7 Related Work

Our work extends two veins of research: Bayesian interpretation of regularization and hyper-parameter optimization.

7.1 Bayesian Interpretation of Regularization

Many regularization strategies can be interpreted as Maximum a posteriori Bayesian inference [32], [33]. One of the most frequently used regularization methods is L2-norm regularization [43], which is also known as weight decay [44], ridge regression or Tikhonov regularization. L2-norm regularization adds a quadratic term to the objective function. The addition of this weight decay term shrinks the values of model parameters. L2-norm regularization can be regarded as MAP Bayesian inference [32], [33] with Gaussian prior on the model parameters. It is a special case of GM regularization when the number of Gaussian components is restricted to one.

While L2-norm regularization is the most common form of regularization, another common method to regularize the model is L1-norm regularization [32], which is also known as Lasso [45]. L1-norm regularization is defined as adding the absolute values of the model parameters to the objective function. The L1-norm regularization forces insignificant model

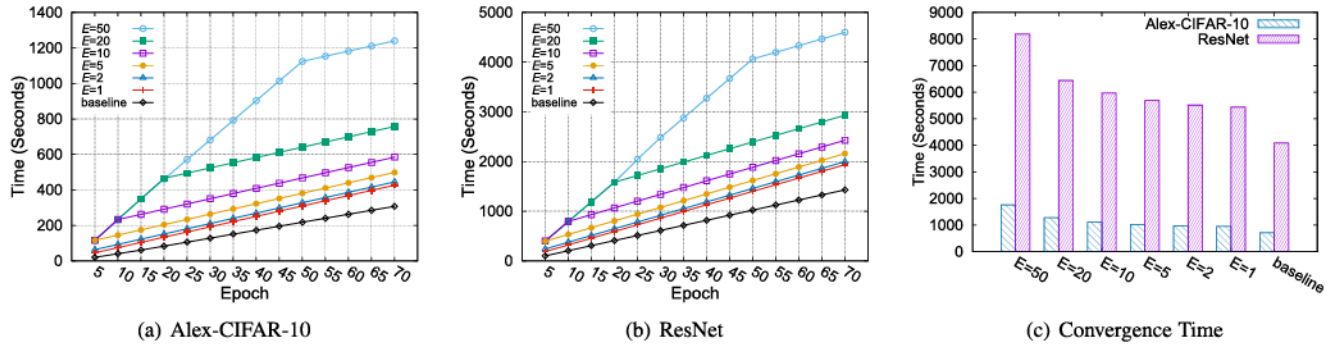


Fig. 14. Time for different E values.

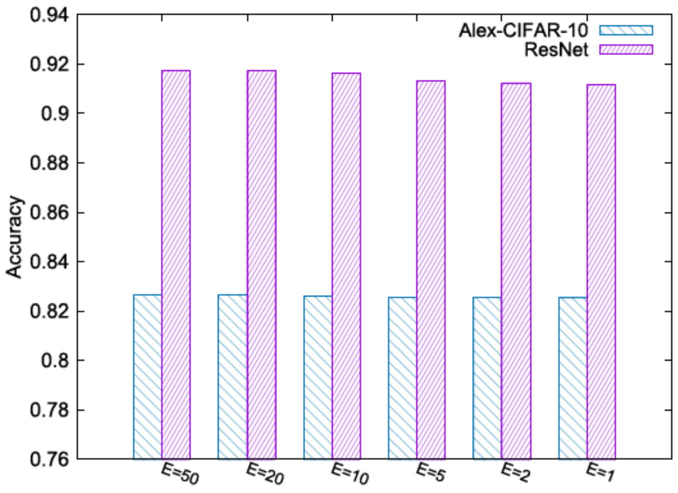


Fig. 15. Accuracy for different E values for Alexnet and Resnet.

parameters to be zero, which is desirable in situations where a sparse solution is preferable. L1-norm regularization corresponds to a Laplacian prior on model parameters.

There are also many other forms of regularization targeting at different scenarios [29], [30]. For example, Huber-norm regularization interpolates between L2-norm and L1-norm regularization by using a piecewise function. Unlike L1-norm regularization, Huber function is differentiable. Huber-norm regularization also imposes less penalty on large model parameters compared with L2-norm regularization. Recent experiments [30] suggest that Huber-norm regularization is more robust and can achieve higher accuracy in Logistic Regression. Elastic-net regularization [29], [46] is another norm regularization method combining L1-norm and L2-norm regularization. The Elastic-net regularization encourages a grouping effect, where strongly correlated predictors tend to be in or out of the model together. In circumstances where the number of features is much larger than observations, Elastic-net regularization outperforms L1-norm regularization significantly.

Compared with Laplacian distribution and Gaussian distribution, GM provides a richer class of density models, modelling the parameter prior better and thus imposing a more appropriate regularization. Also, different from previous works which define a specific regularization function, we aim to develop an adaptive regularization method that can learn the best regularization function. We assume that the model parameters follow a GM prior distribution which provides a richer class of density models. This GM is learned adaptively via a lightweight EM algorithm so that no painstaking ad-hoc attempts need to be made in order to obtain the optimal regularization function.

7.2 Hyper-Parameter Optimization

Deciding the strength of regularization is typically modeled as a hyper-parameter optimization problem [35], [47]. Grid search [34] has long been a conventional method for obtaining the regularization strength. This method, although simple and easy to implement, is shown to be not efficient [34], [48]. Random search improves grid search by randomly choosing trials instead of trials on a grid [34].

Recently, it has been shown that methods which optimize hyper-parameters in a more principled and automatic way can obtain higher-quality hyper-parameters. Bayesian optimization [35], [36], [37], [38], [39] is one of these methods. The key idea of BO is to view the hyper-parameter optimization as the optimization of an unknown black box function, and builds a probabilistic model for the black box function by using multiple pairs of hyper-parameters and their corresponding validation loss. One advantage of BO is that hyper-parameters that need to be evaluated can be automatically determined. In this manner, BO is able to find high-quality hyper-parameters. The BO framework for hyper-parameter optimization has several degrees of freedom to be instantiated, such as initialization, the acquisition function and the probabilistic model. Although BO is widely used and shown to be effective in many applications [35], [36], [37], [38], [39], it can hardly scale up to handle large numbers of hyper-parameters and is not efficient for big datasets.

The key idea of our proposed adaptive GM regularization is to learn the strength of regularization adaptively. Our method is easy to scale up because of the efficient update method that incorporates SGD and EM. Also, different from BO which does not exploit the information of model parameters directly, our method interacts with model parameters during the whole training process.

8 Conclusions

In this paper, we propose an adaptive regularization method based on GM to impose appropriate regularization on different kinds of features. Dirichlet and Gamma prior distributions are introduced for the GM parameters to control the learning of mixing coefficients and the shapes of different Gaussian components. We design a lightweight EM algorithm to update GM parameters and the model parameters are learned under the SGD framework. In order to reduce computational costs, we design lazy update and sparse update algorithms to reduce the computational time by four times and twenty times respectively. Experiments show that our GM regularization method yields better performance in terms of accuracy than existing methods.

Acknowledgments

This work is supported by the Singapore Ministry of Education Academic Research Fund Tier 3 under MOE's official grant number (MOE2017-T3-1-007) and National Key R&D Program of China (2017YFB1201001). Wang-Chien Lee's work is supported in part by the US National Science Foundation under Grant No. IIS-1717084.

References

[1] W. Wang, J. Gao, M. Zhang, S. Wang, G. Chen, T. K. Ng, B. C. Ooi, J. Shao, and M. Reyad, "Rafiki: Machine learning as an analytics service system," *Proc. VLDB Endowment*, vol. 12, no. 2, pp. 128–140, 2018.
[2] C. Zhang, A. Kumar, and C. Ré, "Materialization optimizations for feature selection workloads," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 265–276.
[3] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K.-L. Tan, "Database meets deep learning: Challenges and opportunities," *SIGMOD Rec.*, vol. 45, no. 2, pp. 17–22, 2016.
[4] F. Yang, F. Shang, Y. Huang, J. Cheng, J. Li, Y. Zhao, and R. Zhao, "LFTF: A framework for efficient tensor analytics at scale," *Proc. VLDB Endowment*, vol. 10, no. 7, pp. 745–756, 2017.

[5] J. Shin, S. Wu, F. Wang, C. De Sa, C. Zhang, and C. Ré, "Incremental knowledge base construction using DeepDive," *Proc. VLDB Endowment*, vol. 8, no. 11, pp. 1310–1321, 2015.

[6] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia, "NoScope: Optimizing neural network queries over video at scale," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1586–1597, 2017.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[8] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 818–833.

[9] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Int. Conf. on Learning Representations*, 2014.

[12] W. Wang, X. Yang, B. C. Ooi, D. Zhang, and Y. Zhuang, "Effective deep learning-based multi-modal retrieval," *The VLDB J.*, vol. 25, no. 1, pp. 79–101, 2016.

[13] W. Wang, B. C. Ooi, X. Yang, D. Zhang, and Y. Zhuang, "Effective multi-modal retrieval based on stacked auto-encoders," *Proc. VLDB Endowment*, vol. 7, no. 8, pp. 649–660, 2014.

[14] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM Comput. Surv.*, vol. 27, no. 3, pp. 326–327, 1995.

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2001.

[16] D. J. C. MacKay, "Bayesian modeling and neural networks," Computation and Neural Systems, California Institute of Technology, Pasadena, CA, 1991.

[17] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, 2009.

[18] S. Balakrishnan, M. J. Wainwright, B. Yu et al., "Statistical guarantees for the em algorithm: From population to sample-based analysis," *The Ann. Statist.*, vol. 45, no. 1, pp. 77–120, 2017.

[19] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, "Barzilai-borwein step size for stochastic gradient descent," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 685–693.

[20] C. Lee, Z. Luo, K. Y. Ngiam, M. Zhang, K. Zheng, G. Chen, B. C. Ooi, and W. L. J. Yip, "Big healthcare data analytics: Challenges and applications," in *Handbook of Large-Scale Distributed Computing in Smart Healthcare*. Berlin, Germany: Springer, 2017, pp. 11–41.

[21] D. Jiang, G. Chen, B. C. Ooi, K.-L. Tan, and S. Wu, "epiC: An extensible and scalable system for processing big data," *Proc. VLDB Endowment*, vol. 7, no. 7, pp. 541–552, 2014.

[22] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang, Z. Xie, M. Zhang, and K. Zheng, "SINGA: A distributed deep learning platform," in *Proc. 23rd ACM Int. Conf. Multimedia*, 2015, pp. 685–688.

[23] Q. Cai, Z. Xie, M. Zhang, G. Chen, H. Jagadish, and B. C. Ooi, "Effective temporal dependence discovery in time series data," *Proc. VLDB Endowment*, vol. 11, no. 8, pp. 893–905, 2018.

[24] D. Jiang, Q. Cai, G. Chen, H. Jagadish, B. C. Ooi, K.-L. Tan, and A. K. Tung, "Cohort query processing," *Proc. VLDB Endowment*, vol. 10, no. 1, pp. 1–12, 2016.

[25] W. Wang, M. Zhang, G. Chen, H. Jagadish, B. C. Ooi, and K.-L. Tan, "Database meets deep learning: Challenges and opportunities," *ACM SIGMOD Rec.*, vol. 45, no. 2, pp. 17–22, 2016.

[26] S. Wang, A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, B. C. Ooi, and P. Ruan, "ForkBase: An efficient storage engine for blockchain and forkable applications," *Proc. VLDB Endowment*, vol. 11, no. 10, pp. 1137–1150, 2018.

[27] J. Xu, D. Hsu, and A. Maleki, "Global analysis of expectation maximization for mixtures of two Gaussians," *Advances in Neural Information Processing Systems*, pp. 2676–2684, 2016.

[28] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statistical Soc.: Series B (Methodological)*, vol. 58, pp. 267–288, 1996.

[29] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *J. Roy. Statistical Soc.: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

[30] O. Zadorozhnyi, G. Benecke, S. Mandt, T. Scheffer, and M. Kloft, *Huber-Norm Regularization for Linear Prediction Models*. Berlin, Germany: Springer International Publishing, 2016, pp. 714–730.

[31] Z. Luo, S. Cai, J. Gao, M. Zhang, K. Y. Ngiam, G. Chen, and W.-C. Lee, "Adaptive lightweight regularization tool for complex analytics," in *Proc. 34th IEEE Int. Conf. Data Eng.*, 2018, pp. 485–496.

[32] P. M. Williams, "Bayesian regularization and pruning using a laplace prior," *Neural Comput.*, vol. 7, no. 1, pp. 117–143, 1995.

[33] T. Kneib, S. Konrath, and L. Fahrmeir, "High dimensional structured additive regression models: Bayesian regularization, smoothing and predictive performance," *J. Roy. Statistical Soc.: Series C (Appl. Statist.)*, vol. 60, no. 1, pp. 51–70, 2011.

[34] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. Feb, pp. 281–305, 2012.

[35] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 2951–2959.

[36] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task Bayesian optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 2004–2012.

[37] J. Snoek, K. Swersky, R. Zemel, and R. Adams, "Input warping for Bayesian optimization of non-stationary functions," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1674–1682.

[38] J. R. Snoek, "Bayesian optimization and semiparametric models with applications to assistive technology," Ph.D. dissertation, Univ. Toronto, Toronto, Canada, 2013.

[39] M. A. Gelbart, J. Snoek, and R. P. Adams, "Bayesian optimization with unknown constraints," *Proc. 30th Conf. Uncertainty in Artificial Intelligence*, pp. 250–259, 2014.

[40] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[41] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious URLs: An application of large-scale online learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 681–688.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[43] Y. Wang, X. Sun, and L. Liu, "A variable step size LMS adaptive filtering algorithm based on L2 norm," in *Proc. IEEE Int. Conf. Signal Process. Commun. Comput.*, 2016, pp. 1–6.

[44] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1991, pp. 950–957.

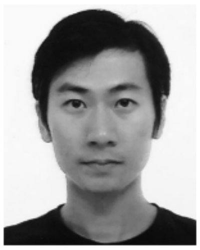
[45] N. Meinshausen and P. Bhlmann, "High-dimensional graphs and variable selection with the lasso," *The Ann. Statist.*, vol. 34, no. 3, pp. 1436–1462, 2006.

[46] C. De Mol, E. De Vito, and L. Rosasco, "Elastic-net regularization in learning theory," *J. Complexity*, vol. 25, no. 2, pp. 201–230, 2009.

[47] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 2546–2554.

[48] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Netw. Model. Anal. Health Inform. Bioinf.*, vol. 5, no. 1, 2016, Art. no. 18.

Zhaojing Luo received the BEng degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2014. He is currently working toward the PhD degree in the School of Computing, National University of Singapore. His research interests include the areas of deep learning, machine learning, and healthcare analytics.



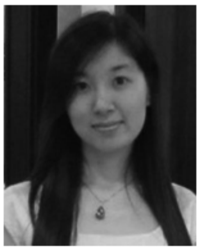
Shaofeng Cai received the BSc degree from the School of Electronics Engineering and Computer Science, Peking University, China, in 2016. He is currently working toward the PhD degree in the School of Computing, National University of Singapore. His research interests include the areas of deep learning and machine learning.

Gang Chen received the BSc, MSc, and the PhD degrees in computer science and engineering from Zhejiang University, in 1993, 1995, and 1998, respectively. He is currently a professor with the College of Computer Science, Zhejiang University. His research interests include databases, information retrieval, information security, and computer-supported cooperative work. He is also the executive director of the Zhejiang University Netease Joint Lab on Internet Technology. He is a member of the IEEE.

Jinyang Gao received the BSc degree from the School of Electronics Engineering and Computer Science, Peking University, China, and the PhD degree in computer science from the National University of Singapore, in 2012 and 2016, respectively. He is currently a research fellow of the School of Computing, National University of Singapore. His research interests include the areas of database, deep learning, and machine learning.

Wang-Chien Lee received the PhD degree in computer and information science from the Ohio State University. Currently, he is an associate professor with the Department of Computer Science and Engineering, the Pennsylvania State University, leading the Intelligent Pervasive Data Access (iPDA) research group to pursue cross-area research in data management, pervasive/mobile computing, and networking. He has published more than 280 research papers in these areas. He serves as an associate editor on the editorial boards of the *IEEE Transactions on Service Computing* (*TSC*) and the *ACM Transactions on Intelligent Systems and Technology* (*TIST*). He is a member of the IEEE.

Kee Yuan Ngiam received the MBBS degree from University College London, United Kingdom, in 2003, and the master's degree in surgery from the National University of Singapore, in 2007 and was inducted as fellow of the Royal College of Physicians and Surgeons of Edinburgh, in 2012. He is currently an assistant professor with the National University of Singapore, School of Medicine and concurrently the group chief technology officer of the National University Health System, Singapore. His research interests include clinical applications of artificial intelligence, big data analytics, and surgical technology research, especially in Thyroid and endocrine surgery. He is an associate editor of the *ACM Transactions in Datascience*.



Meihui Zhang received the BEng degree in computer science from the Harbin Institute of Technology, China, and the PhD degree in computer science from the National University of Singapore, in 2008 and 2013, respectively. She is currently a professor with the Beijing Institute of Technology, and was an assistant professor with the Singapore University of Technology and Design (SUTD) from 2014 to 2017. Her research interest includes crowdsourcing-powered data analytics, massive data integration, spatiotemporal databases, blockchain, and AI. She is a member of the IEEE.