2018 IEEE 34th International Conference on Data Engineering

Adaptive Lightweight Regularization Tool for Complex Analytics

Zhaojing Luo [1], Shaofeng Cai [1], Jinyang Gao [1], Meihui Zhang [2], Kee Yuan Ngiam [3], Gang Chen [4], Wang-Chien Lee [5]

[1] {zhaojing,shaofeng,jinyang.gao}@comp.nus.edu.sg, National University of Singapore, Singapore
[2] meihui_zhang@yeah.net, Beijing Institute of Technology, China
[3] kee_yuan_ngiam@nuhs.edu.sg, National University Health System, Singapore
[4] cg@zju.edu.cn, Zhejiang University, China
[5] wlee@cse.psu.edu, Pennsylvania State University, USA

***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 as some hyper parameters, and therefore 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 algorithm to reduce the computational cost by 4x. 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 13 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.**

I. Introduction

Recent developments in Deep Learning and Machine Learning technology [1], [2], [3] have led to a series of breakthroughs in many real world applications [4], [5], [6], [7]. Thanks to the large public datasets [8] 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 [9], [10], [11], [12].

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

Unfortunately, as the model becomes more complex, the model is more likely to fit the noise in the training data, and consequently, it does not generalize very well to the test data. This phenomenon is called overfitting [13], which is common in complex models and significantly affects the predictive ability of models. Fortunately, overfitting problem [13] can be effectively addressed by regularization [14], which typically involves adding a penalty term on the complexity of the model. Equation (1) shows the loss function that a model aims to minimize. The first term, data-misfit, is an error term that indicates how well the model fits the input data and the second term is the penalty term, which is called regularization term. It consists of a strength parameter β and a function f of model parameter $\boldsymbol{w}$.

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

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 on the type of f (e.g. L1-norm, L2-norm) and strength of the regularization, β, to achieve the best performance. In [15], 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 parameter $\boldsymbol{w}$. 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 parameter $\boldsymbol{w}$. However, the actual distribution of model parameter $\boldsymbol{w}$ for different tasks may

2375-026X/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDE.2018.00051

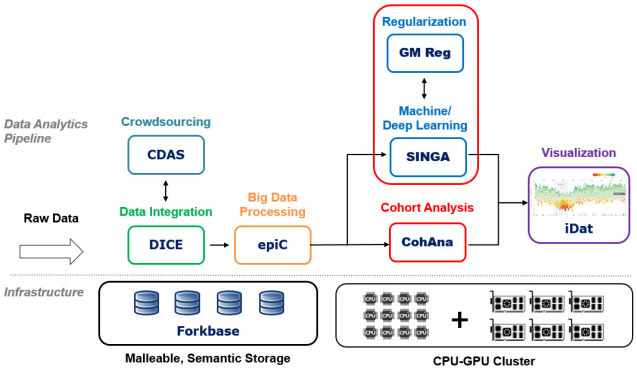


Fig. 1. GEMINI healthcare data analytics software stack.

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 parameter $\boldsymbol{w}$ 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 parameter $\boldsymbol{w}$ 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 parameter obtained during the model training process. Subsequently, this adaptive distribution function will be used to impose regularization on the model parameter.

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 parameter $\boldsymbol{w}$ better. The core idea is as following: the deep learning model inputs model parameter $\boldsymbol{w}$ 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 parameter $\boldsymbol{w}$ is challenging for several reasons. Two sets of parameters, namely GM parameters and model parameter $\boldsymbol{w}$ 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 parameter properly, we design an innovative and effective update method where GM parameters are updated via a lightweight Expectation-Maximization (EM) algorithm [16] and the model parameter can be learned under a common optimization framework such as Stochastic Gradient Descent (SGD) [17]. To reduce the computational cost, we propose a lazy update algorithm that reduces the computational time by more than 4 times.

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 [18], which has been designed to support healthcare data analytics. Figure 1 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 crowdsourcing CDAS platform to improve data quality using subject matter experts (eg. clinicians). The cleansed data can then be processed by epiC [19], which provides big data processing and analytics such as aggregation and summarization. Deep analytics is supported by Apache SINGA [20], a distributed deep learning platform. The red box illustrates the interaction of 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 [21]. Lastly, the results could be visualized via iDat. At the storage layer, the data is stored in Forkbase [22], 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 parameter instead of making ad-hoc attempts to obtain a suitable regularization setting.
- We propose an efficient update framework where GM is trained by lightweight EM algorithm [23] and model parameter is updated via SGD. We also propose an efficient lazy update algorithm to reduce the computational cost.
- We conduct extensive experiments using both real-world datasets and standard benchmark machine learning datasets. 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 [24], L2-norm regularization [14], Elastic-net regularization [25] and Huber-norm regularization [26]) 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.

The remainder of the paper is structured as follows. Section II gives some related background on Bayesian interpretation of regularization, GM and several useful priors. Section III introduces our proposed GM regularization method. Section IV introduces our regularization tool as a flexible tool, and Section V reports the experimental results. Related work is reviewed in Section VI and finally Section VII concludes this paper.

TABLE I
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 n-th sample in the training set
$y^{(n)}$	label of the n-th sample in the training set
$\boldsymbol{w} \in \mathcal{R}^{M\times 1}$	model parameter
$\mathcal{D}$	$(\boldsymbol{x}^{(n)}, y^{(n)})_{n=1}^{N}$, features/label pairs in the training set
K	number of Gaussian components
$\boldsymbol{\pi}$	$[\pi_1, \pi_2, ..., \pi_k]^T$, mixing coefficient (satisfy the constraint $\sum_{k=1}^{K} \pi_k = 1$)
$\boldsymbol{\lambda}$	$[\lambda_1, \lambda_2, ..., \lambda_k]^T$, precision (inverse of variance)
$\mathcal{N}(x\|\mu_k, \lambda_k)$	Gaussian distribution of the k-th Gaussian component
$\boldsymbol{\alpha}$	$[\alpha_1, ..., \alpha_K]^T$, parameters of Dirichlet distribution
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

II. PRELIMINARIES

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

A. Bayesian Interpretation of Regularization

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

According to Bayes' Theorem, the posterior probability of model parameter $\boldsymbol{w}$ 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 [27], [28]. The MAP problem can be written as below.

$$\begin{aligned} \boldsymbol{w}_{MAP} &= \underset{\boldsymbol{w}}{\operatorname{argmax}}\ p(\boldsymbol{w}|\mathcal{D}) \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}}\ p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w}) \\ &= \underset{\boldsymbol{w}}{\operatorname{argmax}}\ \log p(\mathcal{D}|\boldsymbol{w}) + \log p(\boldsymbol{w}) \end{aligned} \tag{3}$$

The term $\log p(\boldsymbol{w})$ is the log of model parameter prior distribution and it is the regularization term (corresponds to $f(\beta, \boldsymbol{w})$ 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 $p(\boldsymbol{w})$ 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 parameters and Laplacian distribution for large value 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.

B. Gaussian Mixture Distribution

GM is a superposition of multiple Gaussian distributions. In our GM regularization, we assume all dimensions of model parameter $\boldsymbol{w}$ 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 k-th Gaussian component.

C. Dirichlet and Gamma Prior

The intermediate model parameter during training process does not have a stationary distribution. Therefore, the actual Gaussian components cannot be learned directly from this intermediate model parameter. In order to learn the GM prior for the model parameter $\boldsymbol{w}$, 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.

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

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

As mentioned in Section II-A, 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 dimensions of model parameter are small. If GM is learned based on these dimensions of model parameter, 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 $\boldsymbol{\lambda}$.

III. PROPOSED ADAPTIVE REGULARIZATION

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

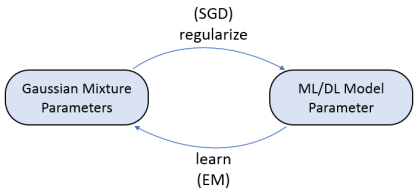


Fig. 2. Overview of SGD and EM.

A. GM Regularization Term

In this paper, GM distribution is used as model parameter prior. In particular, we assume that all the dimensions of the model parameter are independent and identically distributed (this assumption is commonly used in Gaussian prior, Laplacian prior, etc.). Also, we assume all the dimensions 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 $\boldsymbol{x}^{(n)} \in \mathcal{R}^{M\times 1}$ is used to denote the set of input/output pairs in the training set, $\boldsymbol{w} \in \mathcal{R}^{M\times 1}$ denotes model parameter, where N is the number of samples and M is the number of features. Mixing coefficients $[\pi_1, \pi_2, ..., \pi_k]^T$ are denoted as $\boldsymbol{\pi}$ and precisions $[\lambda_1, \lambda_2, ..., \lambda_k]^T$ as $\boldsymbol{\lambda}$. Considering the prior distributions introduced in Section II-C, the prior distribution of model parameter $\boldsymbol{w}$ can then be written as below.

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

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 $\boldsymbol{\lambda}$.

B. 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.

C. 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 parameter. Typically, the model parameter is learned through SGD, which is employed for updating model parameter here. For GM parameters, a lightweight EM algorithm is designed. Fig. 2 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 parameter through SGD. After the model parameter is updated via an SGD step, one step of EM is employed to update the GM based on the updated model parameter. Subsequently, a new regularization is calculated for the model parameter. This process iterates until the algorithm converges.

1) Responsibility Function: Responsibility function of the k-th Gaussian component for the m-th dimension of 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 dimension 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.

2) Gradient Descent for Model Parameter: When GM parameters are fixed, gradient descent method is used to update model parameter $\boldsymbol{w}$. The gradient for the m-th dimension of 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 m-th dimension of sample n. For simplicity, we denote the first term as the $\boldsymbol{g}_{ll}$ (all the dimensions) and the second term as the $\boldsymbol{g}_{reg}$ (all the dimensions).

The $\boldsymbol{g}_{reg}$ is a weighted sum of the product of Gaussian precision value 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 strength to different dimensions of model parameter. For areas that are near zero, the Gaussian component that has the largest precision value dominates other Gaussian components. Consequently, for dimensions of model parameter with small values, the regularization is strong because it is mainly imposed by this Gaussian component with the largest precision value. On the contrary, for dimensions of model parameter with large values, the regularization is weak.

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 k-th 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 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 Equation (11) and Equation (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 parameter to the k-th cluster.

Since $\sum_{m=1}^{M} r_k(w_m)$ corresponds to the responsibility of the k-th Gaussian component. The magnitude of $\sum_{m=1}^{M} r_k(w_m)$ is the same as the number of dimensions of model parameter (We use M to denote the number of dimensions of model parameter). 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.

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

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 Equation (15) and Equation (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 k-th 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 Equation (9), (10), (17) and (13), responsibility values can be computed and then model parameter $\boldsymbol{w}$ can be updated. After the model parameter is updated, the values for GM parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ can be updated subsequently. The procedure of updating model parameter and GM parameters are shown in Algorithm 1, in which L is the learning rate for SGD.

Algorithm 1 Update for GM Regularization with SGD and EM

Input: $\boldsymbol{w}$, $\boldsymbol{\alpha}$, a, b, $\boldsymbol{\pi}$, $\boldsymbol{\lambda}$, L
1: **initialize**: $it \leftarrow 0$
2: **while** not converged **do**
3: Compute $\boldsymbol{g}_{ll}$
/* E-step */
4: Compute responsibility function based on Equation (9)
5: Compute $\frac{\partial G}{\partial \boldsymbol{w}}$ based on Equation (10)
/* M-step */
6: Compute $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ based on Equation (17) and (13)
/* SGD step */
7: $\boldsymbol{w}^{(it+1)} \leftarrow \boldsymbol{w}^{(it)} - L\frac{\partial G}{\partial \boldsymbol{w}}$
8: $it \leftarrow it + 1$
9: **end while**

D. Lazy Update

Calculating g_{reg} 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 large number of parameters. The intuition for lazy update is explained as follows. Both $\boldsymbol{g}_{reg}$ and GM parameters $\boldsymbol{\pi}, \boldsymbol{\lambda}$ 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.

Here, 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 parameter. $\boldsymbol{g}_{reg}$ 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 2. 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.

IV. Regularization Tool in Data Analytics Pipeline

As mentioned earlier and shown in Figure 1, our proposed regularization method is designed as a general and flexible tool that can be easily integrated with the deep learning platform in the big data analytics pipeline.

In our implementation, 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 [20].

Key Functions. The key functions of GM regularization tool include:

Algorithm 2 Lazy Update for GM Regularization

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

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

With this tool, a model only needs to input intermediate model parameter $\boldsymbol{w}$. In turn, our GM regularization tool calculates and returns the gradient with respect to the regularization term, $\boldsymbol{g}_{reg}$. It is sufficiently general to support different types of models.

V. 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) [24], L2-norm regularization (L2 reg) [14], Elastic-net (Elastic-net reg) [25] and Huber-norm regularization (Huber reg) [26].

TABLE II
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

A. 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 60000 32 × 32 colour images in 10 classes, with 6000 images per class and there are 50000 training images and 10000 test images in this dataset.
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 will be readmitted into hospital within 30 days. This dataset consists of 1755 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., predictive and noisy features. The distributions of model parameter that corresponds to different kinds of features are quite different. To be specific, the distribution of model parameter that corresponds to predictive features has a larger variance while the distribution of model parameter that corresponds to noisy features has a smaller variance.
3) UCI machine learning repository datasets. These are the benchmark datasets from the UCI machine learning repository [29]. 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 assigned a separate class and missing values in the continuous features are imputed by the mean value.

Table II 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 1000 samples, but the number of features is large because the ratios of # Features to # Samples of most datasets are more than 10%.
Evaluation metric. We use accuracy, the ratio of correct predictions, to measure the classification performance of a regularization method.
Deep Learning Model Structures. Table III shows the model structures for our deep learning models. For Alex-CIFAR-10, the first convolutional layer filters the 32 × 32 × 3 input images with 32 kernels of size 5 × 5 × 3 with a stride of 1 pixel. The second convolutional layer has 32 kernels of size 5 × 5 × 32 and the third convolutional layer has 64 kernels of size 5 × 5 × 32. The last layer is the 10-way fully-connected softmax layer. pooling layer, relu layer and LRN layer [6] are inserted between convolutional layers. The number of dimensions for model parameter is 89440.

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

TABLE III
DEEP LEARNING MODEL STRUCTURES

Model	Layers							
Alex-CIFAR-10	5 × 5, 32 MaxPooling RELU LRN } × 1	5 × 5, 32 RELU AvgPooling LRN } × 1	5 × 5, 64 RELU AvgPooling } × 1	Softmax				
ResNet	3 × 3, 16 BN RELU } × 1	3 × 3, 16 BN RELU 3 × 3, 16 } × 3	3 × 3, 32	3 × 3, 32 BN RELU 3 × 3, 32 } × 3	3 × 3, 64	3 × 3, 64 BN RELU 3 × 3, 64 } × 3	Avgpooling	Softmax

The second model is the twenty-layer ResNet [9], 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 $6n$ ($n = 3$ here) layers with 3 × 3 convolutions with 16, 32, 64 filters respectively. The network ends with a 10-way softmax function. In total, there are 20 stacked weighted layers. The number of dimensions of model parameter 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. These hyper-parameters are set to be the same as [6] and [9] 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 [20]. Experiments are run on a server equipped with Intel i7-4930K CPU and three GTX Titan X GPU cards.

B. 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.

1) Easy Setting of GM Hyper-parameters: The setting of GM hyper-parameters is straightforward. It can be automatically defined by the charactersitcs 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 and each layer is enabled to adaptively learn its best GM as the regularization term.

Initial Number of Gaussian Components. The initial number of Gaussian components, K, is fixed to 4. An interesting observation is that our GM regularization learns one or two Gaussian components for different models finally. This is 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 evaluated with different initial number of Gaussian components and found 4 to be the best according to the experimental results.

Hyper-parameters a, b and α. b is set to γM, where M is the number of model parameter dimensions 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 III-C3, 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 IV
LEARNED REGULARIZATION FOR ALEX-CIFAR-10

GM Regularization		
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]
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]

2) Learned GM Regularization: Table IV and V show the learned Gaussian Mixtures for the Alex-CIFAR-10 and ResNet respectively. Since ResNet has 20 layers, only the learned $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ for the representative layers are shown. Layers with asterisk mean there are several other layers that have very similar $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$.

Compared to the baseline method (L2 Reg) where $\boldsymbol{\lambda}$ is manually set, our method can adaptively learn different GM parameters $\boldsymbol{\pi}$ and $\boldsymbol{\lambda}$ for different layers automatically while we apply the same rule of setting hyper-parameters for the GMs of different layers. 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 non-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 V-E. According to [30], the initialization distributions of model parameter between two convolutional layers are the same if the

TABLE V
REPRESENTATIVE LEARNED REGULARIZATION FOR RESNET

GM Regularization		
Layer Name	$\boldsymbol{\pi}$	$\boldsymbol{\lambda}$
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]
L2 Reg		
Layer Name	$\boldsymbol{\pi}$	$\boldsymbol{\lambda}$
All Layers	[1.000]	[50.000]

TABLE VI
ACCURACY ON DEEP LEARNING MODEL

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

two layers have the same number of filters. As shown in Table III, 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 $\boldsymbol{\pi}$.

3) Comparison on Accuracy: In this experiment, we study the effects of GM regularization on the predictive ability of deep learning models and the results are summarized in Table VI. For the Alex-CIFAR-10, the L2 Reg is carefully tuned by the human expert, where the strengths of regularization are different for different layers. L2 Reg can improve the accuracy from 0.777 (without any regularization) to 0.822, which indicates the importance of regularization in deep learning models. Our GM regularization can further improve the expert-tuned model from 0.822 to 0.83 in terms of accuracy. This result confirms the advantage of adaptive GM regularization over the manually-tuned regularization.

For the ResNet, since BN layer serves as a form of regularization, therefore the improvement of L2 Reg over no regularization is not so dramatic as that in Alex-CIFAR-10. However, our model can still further improve L2 Reg from 0.909 to 0.921. This result is nearly the same as the result of a ResNet with 1202 layers [9], which confirms the effectiveness of our proposed GM regularization.

C. 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, 5 subsamples via stratified sampling with a 80-20 train test split are obtained. The average and standard errors of accuracies under the best parameter setting, determined by cross validation, are shown in Table VII. The highest average accuracy results are indicated in bold. The result shows that GM regularization method outperforms the other four regularization methods in 9 out of 12 datasets and achieves the same best performance as other baseline methods in two datasets. GM regularization does not prevail against the baseline methods only in 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 parameter of sparse and noisy 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, thus it is likely that the useful features with large model parameter values are over-regularized. In contrast, GM regularization does not regularize these useful 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 can enable L1-norm or L2-norm regularization to dominate. In this way, Elastic-net can achieve better results than L1-norm regularization and L2-norm regularization for nearly all the datasets.

Huber-norm regularization is in the form of a piecewise function; that is, Huber-norm regularization is L2-norm regularization for small model parameter and it is L1-norm regularization for large model parameter. The two parameters μ and λ control the threshold between L1-norm and L2-nomr regularization. By tuning the threshold value, Huber-norm regularization can tradeoff between L1-norm and L2-norm regularization, which enables Huber-norm regularization to dominate L1-norm and L2-norm regularization method.

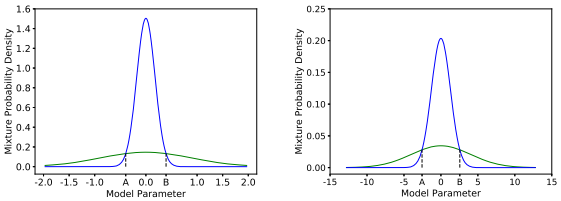
By assuming a GM prior distribution over model parameter, our adaptive GM regularization method can model the model parameter prior better thus imposing more appropriate regularization. It learns two Gaussian components as model parameter prior distribution to regularize both useful and noisy features, imposing different strengths of regularization on these two kinds of features, which results in better performance than the baselines. Details on learned Gaussian components are discussed in Section V-D.

D. 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 dataset in Fig. 3. For both datasets, two Gaussian components are learned. Points A and B labeled in the figure show where two Gaussian components have the same mixture probability density. In both figures, the small

TABLE VII
COMPARISON ON ACCURACIES AND STANDARD ERRORS

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



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

Fig. 3. The horizontal axis indicates model parameter w and the vertical axis shows the mixture probability density. 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].

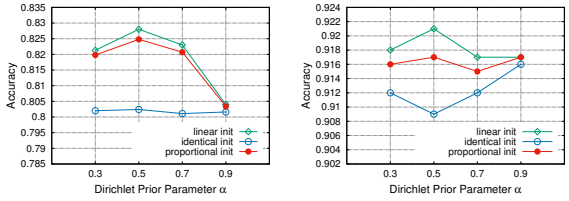
variance Gaussian component dominates in the area near zero. When model parameter is getting beyond A/B points, the large variance Gaussian component begins to dominate. This shows that our GM regularization exerts strong effects on small value model parameter which corresponds to noisy features and exerts less strong regularization on large value model parameter which corresponds to useful features. Another observation is that the Gaussian shapes of the two figures differ a lot, which illustrates 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 the model parameter corresponds to the noisy features in horse-colic dataset is much smaller and needs to be regularized more strongly.

TABLE VIII
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

E. 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 parameter. In both Deep Learning model and Logistic Regression model, the model parameter is initialized with a zero-mean Gaussian distribution. The variances of different

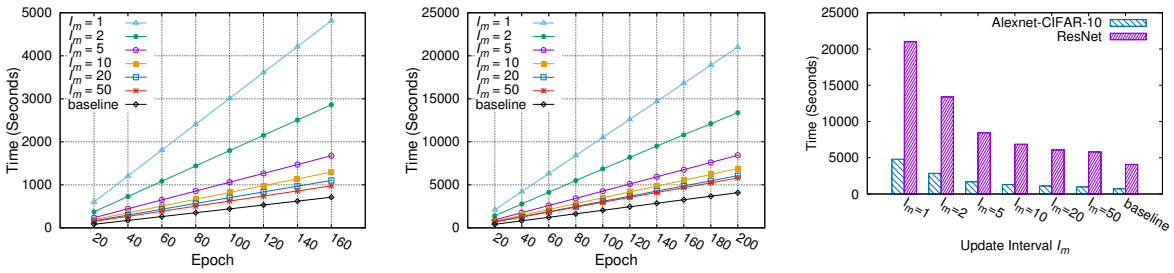


(a) Alex-CIFAR-10 (b) ResNet

Fig. 4. Accuracy for different alpha values and initialization methods.

components of the initialized GM should be larger than the variance of the initialized model parameter 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 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 parameter are different, min is set to one-tenth of the initialized model parameter precisions. For other models, since the precisions of initialized model parameter is 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 II-C, α also affects the number of Gaussian components learned. We therefore are interested in investigating the effects of GM initialization methods with respect to different α values. Fig. 4 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 VIII 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



(a) Alex-CIFAR-10 (b) ResNet (c) Convergence Time

Fig. 5. Time for different update interval values.

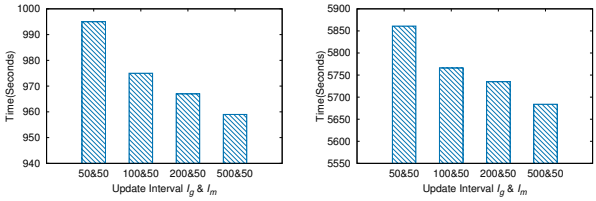
Gaussian components. If the GM components have different variances initially, they will converge to the final state faster. Another observation is that linear initialization is better than proportional initialization, because linear initialization method generates Gaussian components that are more scattered. When α is set to 0.5, we get the best performance as the GM learns multiple Gaussian components that lead to faster convergence.

F. 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.

1) Performance of Update Interval Values: Figures 5(a)(b) 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 for convergence and the algorithm with I_m = 50 takes the shortest. This is because the algorithm with larger I_m updates GM parameters and model parameter less frequently. Fig. 5(c) 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 in Figures 5(a)(b). 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, without drop in model accuracy. This again shows the effectiveness of lazy update algorithm.

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 parameter, we set I_g larger



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

Fig. 6. Time for different combinations of I_g and I_m.

than I_m. Fig. 6 (a)(b) 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. 6 shows that the convergence time can be further reduced if I_g is increased.

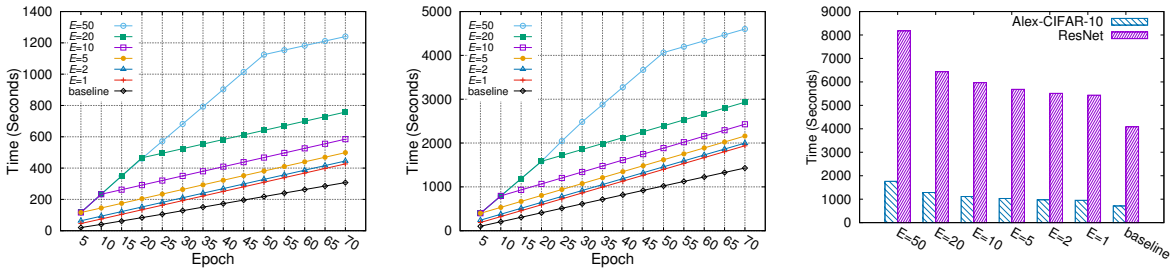
3) Performance of E Values: As mentioned in Section III-D, the number of first few epochs when the lazy update is not employed, E, is another factor that affects the time and accuracy. Fig. 7 (a) (b) show the training elapsed time with respect to epochs for different E values and baseline (L2 Reg). The results show that the lazy update algorithm takes more time for computation of each epoch before E epochs, since updating model parameter 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. 7(c) 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% the time for algorithm with E=50, without drop in model accuracy. By choosing a relatively small E value, we can obtain a high-performance model using short training time.

VI. Related Work

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

A. Bayesian Interpretation of Regularization

Many regularization strategies can be interpreted as Maximum a posteriori (MAP) Bayesian inference [27], [28]. One of the most frequently used regularization methods is



(a) Alex-CIFAR-10 (b) ResNet (c) Convergence Time

Fig. 7. Time for different E values.

L2-norm regularization [31], which is also known as weight decay [32], 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 value of model parameter. L2-norm regularization can be regarded as MAP bayesian inference [27], [28] with Gaussian prior on the model parameter. 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 [27], which is also known as Lasso [33]. L1-norm regularization is defined as adding the absolute value of the model parameter to the objective function. The L1-norm regularization forces insignificant dimensions of model parameter to be zero, which is desirable in situations where a sparse solution is preferable. L1-norm regularization corresponds to a Laplacian prior on model parameter.

There are also many other forms of regularization targeting at different scenarios [26], [25]. 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 parameter compared with L2-norm regularization. Recent experiments [26] suggest that Huber-norm regularization is more robust and can achieve higher accuracy in Logistic Regression. Elastic-net regularization[25], [34] 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 the model parameter follows 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.

B. Hyper-parameter Optimization

Deciding the strength of regularization is typically modeled as a hyper-parameter optimization problem [35], [36], [37]. Grid search [38] 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 [39], [38]. Random search improves grid search by randomly choosing trials instead of trials on a grid [38].

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(BO) [35], [37] 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. Spearmint [37] and TPE [36] are two of the popular methods under the BO framework. Although these algorithms are widely used and shown to be effective in many applications [36], [37], they can hardly scale up to handle large numbers of hyper-parameters and are 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 methods which do not exploit the information of model parameter directly, our method interacts with model parameter during the whole training process.

VII. 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 parameter is learned under SGD framework. In order to reduce computational costs, we design a lazy update algorithm to reduce the computational time by four times. Experiments show that our GM regularization method yields better performance in terms of accuracy than existing methods.

Acknowledgment

This work is supported by National Research Foundation, Prime Ministers Office, Singapore under its Competitive Research Programme (CRP Award No. NRF-CRP8-2011-08) and National Basic Research Program (973 Program, No.2015CB352400).

References

[1] C. Zhang, A. Kumar, and C. Ré, "Materialization optimizations for feature selection workloads," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2014, pp. 265–276.

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

[3] F. Yang, F. Shang, Y. Huang, J. Cheng, J. Li, Y. Zhao, and R. Zhao, "Lftf: A framework for efficient tensor analytics at scale," *Proceedings of the VLDB Endowment*, vol. 10, no. 7, pp. 745–756, 2017.

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

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

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, 2014, pp. 818–833.

[8] 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," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

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

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

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

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

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

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. Springer New York Inc., 2001.

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

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

[17] C. Tan, S. Ma, Y.-H. Dai, and Y. Qian, "Barzilai-borwein step size for stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2016, pp. 685–693.

[18] 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*. Springer, 2017, pp. 11–41.

[19] D. Jiang, G. Chen, B. C. Ooi, K.-L. Tan, and S. Wu, "epic: an extensible and scalable system for processing big data," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 541–552, 2014.

[20] 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 *Proceedings of the 23rd ACM International Conference on Multimedia*, 2015, pp. 685–688.

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

[22] S. Wang, A. Dinh, Q. Lin, Z. Xie, M. Zhang, Q. Cai, G. Chen, W. Fu, B. C. Ooi, P. Ruan *et al.*, "Forkbase: An efficient storage engine for blockchain and forkable applications," *arXiv preprint arXiv:1802.04949*, 2018.

[23] J. Xu, D. Hsu, and A. Maleki, "Global analysis of expectation maximization for mixtures of two gaussians," *arXiv preprint arXiv:1608.07630*, 2016.

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

[25] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 2, pp. 301–320, 2005.

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

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

[28] T. Kneib, S. Konrath, and L. Fahrmeir, "High dimensional structured additive regression models: Bayesian regularization, smoothing and predictive performance," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 60, no. 1, pp. 51–70, 2011.

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

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[31] Y. Wang, X. Sun, and L. Liu, "A variable step size LMS adaptive filtering algorithm based on l2 norm," in *2016 IEEE International Conference on Signal Processing, Communications and Computing*, 2016, pp. 1–6.

[32] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, 1991.

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

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

[35] M. Feurer, J. T. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning." in *AAAI conference on artificial intelligence*, 2015, pp. 1128–1135.

[36] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[37] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012, pp. 2951–2959.

[38] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[39] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, no. 1, p. 18, 2016.