Neurocomputing 536 (2023) 13–20

Contents lists available at ScienceDirect

Neurocomputing

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

Hessian regularization of deep neural networks: A novel approach based on stochastic estimators of Hessian trace

Yucong Liu [a], Shixing Yu [b], Tong Lin [c,*]

[a] Department of Statistics, University of Chicago, Chicago 60637, IL, USA
[b] Department of Electrical and Computer Engineering, University of Texas Austin, Austin 78712, TX, USA
[c] Key Lab of Machine Perception (MoE), School of Intelligence Science and Technology, Peking University, Beijing 100871, China

ARTICLE INFO

Article history:
Received 9 May 2022
Revised 27 February 2023
Accepted 12 March 2023
Available online 16 March 2023
Communicated by Zidong Wang

Keywords:
Hessian regularization
Stochastic algorithm
Dynamical system
Flat minima

ABSTRACT

In this paper, we develop a novel regularization method for deep neural networks by penalizing the trace of Hessian. This regularizer is motivated by a recent guarantee bound of the generalization error. We explain its benefits in finding flat minima and avoiding Lyapunov stability in dynamical systems. We adopt the Hutchinson method as a classical unbiased estimator for the trace of a matrix and further accelerate its calculation using a Dropout scheme. Experiments demonstrate that our method outperforms existing regularizers and data augmentation methods, such as Jacobian, Confidence Penalty, Label Smoothing, Cutout, and Mixup. The code is available at https://github.com/Dean-lyc/Hessian-Regularization.

1. Introduction

Deep neural networks (DNNs) are developing rapidly in recent years. The effectiveness of DNNs has been widely demonstrated in multiple areas including reflection removal [1], dust pollution [7], building defects detection [24], cities and urban development [22]. In literature, researchers have been exploring on different architectures design of neural networks, e.g. residual connections [9], batch normalization [16] or special design for different type of activation functions [11,33] to boost the performance. However, among various learning problems, over-fitting on training data has been consistently demonstrated to be a essential problem that greatly affected the generalization ability of deep learning models. To alleviate the overfitting problem and to achieve better generalization, regularization has long been developed to penalize on the complexity of the system.

There are two classical regression methods reducing model complexity in linear models, namely Ridge Regression [14] and Lasso Regression [29] that has profound influence on following regularization schemes. Inspecting from its core methodology, they are also called L_2 and L_1 regularization. L_2 regularization has the effect of shrinkage while L_1 regularization can be beneficial to both shrinkage and sparsity, with a special focus on the sparse structure. From a Bayesian perspective, they can be interpreted as posterior estimation given different prior, where ridge regression uses normal distribution and Lasso regression follows a Laplacian prior respectively.

As classical Lasso and Ridge Regression for linear models can be elegantly and effectively applied to the regularization of neural networks, the situation here is more complex than in linear models. So sophisticated regularization methods have been developed. An intuitive idea is starting from penalizing the magnitude of weights. The most widely used method is Weight-Decay [18], which is a technique to shrink parameters to 0 before updating by gradient. Loshchilov and Hutter [20] also showed that L_2 regularization and Weight-Decay are not identical. Dropout [27] is another method to avoid over-fitting by reducing co-adapting between units in neural networks. Dropout has inspired a large body of work studying its effects [30,10,31]. After dropout, various regularization schemes can be applied additionally, such as Label Smoothing [28] and Confidence Penalty [23]. For more literature review, see Section 2.

Most existing methods only consider about the scale of parameters and the shape of output confidence. None of them takes the local properties near the minima into consideration, such as Lyapunov stability or sharpness. Our work focus on such properties of better minima and how to find that one by gradient descent. This is the essential difference between our regularizer and existing ones.

* Corresponding author.
E-mail address: lintong@pku.edu.cn (T. Lin).

https://doi.org/10.1016/j.neucom.2023.03.017
0925-2312/

Our contributions are summarized as follows:

- We develop a novel regularization scheme by penalizing the Hessian trace. This regularization method is directly inspired by a recent generalization bound about the Jacobian norm and the Hessian trace. The Hessian trace can also be connected to flat minima with good generalization performance and stability analysis of a nonlinear dynamical system.
- We develop a stochastic Hessian trace estimation algorithm for efficient implementation in practice.
- Experimental results demonstrate that our new regularization method outperforms other existing approaches on both vision and language tasks.

2. Related Work

There are many regularization methods in previous work. Label Smoothing [28] estimates the marginalized effect of label-dropout and reduces over-fitting by preventing a network from assigning a full probability to each training example. Similarly, Confidence Penalty [23] prevents peaked confidence distributions, leading to better generalization. A network appears to be overconfident when it places all probability on a single class in the training set, which is often a symptom of over-fitting. DropBlock [8] is a structured form of dropout: instead of dropping out independent random units, contiguous regions are dropped from a feature map of each layer.

Data augmentation methods are also used in practice to improve the model's accuracy and robustness when training neural networks. Cutout [5] is a data augmentation method where parts of the input examples are zeroed out, in order to focus more on less prominent features for generalizing to masked regions. Mixup [38] extends the training distribution by incorporating the prior knowledge that linear interpolations of feature vectors should lead to linear interpolations of the associated targets.

Sokolić et al. [26] first proposed Jacobian regularization, a method focusing on the norm of Jacobian matrix with respect to input data. It was proved that generalization error can be bounded by the norm of Jacobian matrix. Besides that, Jacobian matrix shows improved stability of the model predictions against input perturbations according to Taylor expansion. Hoffman et al. [15] showed that Jacobian regularization enlarges the size of decision cells and is practically effective in improving the generalization error and robustness of the models. To simplify calculation, stochastic algorithm of Jacobian regularization was also proposed.

Motivated by Jacobian regularization, we consider the generalization error and stability of the model concerning the Hessian matrix. We propose Hessian regularization with corresponding stochastic algorithms. We compare our Hessian regularization with other methods and demonstrate promising performance in experiments. The main idea to estimate the trace of the Hessian matrix is Hutchinson Method [2] which was also discussed by Yao et al. [34]. We make an improvement by designing a new probability distribution to dropout parameters, which decreases time consumption obviously without losing generalization.

Hessian information is powerful tool used on analyzing the property of neural networks. Yao et al. [35] designed AdaHessian, a second order stochastic optimization algorithm. A Hessian-Aware Pruning method [36] was developed to find insensitive parameters in a neural network, and a Neural Implant technique was also proposed to alleviate accuracy degration. However, their methods are static in essence, whereas we focus on dynamical motion of parameters in parameter space. Sankar et al. [25] also proposed a Hessian regularization. They focused on the layerwise loss landscape via the eigenspectrum of the Hessian at each layer. We start from different perspectives of generalization error bound and dynamical system of parameters. Our experiments also shows better results than Sankar et al. [25]'s method.

The literature of Hochreiter and Schmidhuber [12], Keskar et al. [17], Dinh et al. [6] observed that the flatness of minima of the loss landscape results in good generalization. In contrast, sharp minima lead to poorer generalization. Concretely, sharp minima can be characterized by a significant number of large positive eigenvalues in the Hessian matrix which tend to generalize poorly. They found that small-batch gradient methods converge to flat minimizers, which can be characterized by having numerous small eigenvalues of the Hessian matrix. Our newly designed regularization method enables the neural network to find flat minima for achieving better performance.

3. Motivations

In this section, we start with notations and definitions in subSection 3.1. Then we introduce a generalization error bound in subSection 3.2, which motivates us to constrain the Hessian trace. After that, we explain the benefit on searching flat minima by penalizing Hessian trace in subSection 3.3. Furthermore, the connection between the Hessian trace and Nonlinear Stability Analysis is shown in subSection 3.4. Finally we define the Hessian regularization in subSection 3.5.

3.1. Notations and Definitions

Suppose there is a hypothesis function $h : \mathscr{X} \to \mathscr{Y}$ output an target $y \in \mathscr{Y}$, given an input feature vector $x \in \mathscr{X}$. Denote the joint distribution of x and y as $P(x, y)$. Sample set $\mathscr{S}$ consists of n instances $(x_1, y_1), \ldots, (x_n, y_n)$ drawn i.i.d. from $P(x, y)$. With this sample set, we want to use a DNN model $f(x; \omega)$ to approximate $h(x)$, where x is input data and ω is trainable parameters. Let ℓ be a non-negative real-valued loss function, where $\ell(f(x; \omega), y)$ measures the difference between the prediction $f(x; \omega)$ and the ground-truth y.

The empirical loss of $f(x; \omega)$ associated with the sample set is the average of loss for each sample, which is defined as

$$\ell_{emp}(f) = \hat{\mathbb{E}}[\ell(f(x; \omega), y)] = \frac{1}{n} \sum_{(x_i, y_i) \in \mathscr{S}} \ell(f(x_i; \omega), y_i), \tag{1}$$

and the expected loss of $f(x)$ is the expectation of loss under the joint distribution P, defined as

$$\ell_{exp}(f) = \mathbb{E}[\ell(f(x; \omega), y)] = \mathbb{E}_{(x,y)\sim P}[\ell(f(x; \omega), y)]. \tag{2}$$

Then the difference between $\ell_{emp}(f)$ and $\ell_{exp}(f)$ is called generalization error:

$$GE(f) = ||\ell_{exp}(f) - \ell_{emp}(f)||. \tag{3}$$

3.2. Generalization Error Bound

In this subsection, we introduce a recent generalization error bound involving Hessian trace.

Wei et al. [31] showed a generalization error bound of linear models with cross-entropy loss of M classes. Let $\boldsymbol{W}$ is the weight matrix, $\mu(\boldsymbol{W}) := \widehat{E}[||\boldsymbol{J}||_2]$ and $v(\boldsymbol{W}) := \widehat{E}[tr(\boldsymbol{H})]$, where $\boldsymbol{J}$ denotes the Jacobian matrix and $\boldsymbol{H}$ denotes the Hessian matrix. Thus, $\mu(\boldsymbol{W})$ is the average Jacobian matrix norm and $v(\boldsymbol{W})$ is the average Hessian trace over samples. Then, with probability $1 - \delta$ over the training examples, for all weight matrices $\boldsymbol{W}$ satisfying the norm bound $||\boldsymbol{W}^T||_{2,1} \leqslant A$, the following bound holds:

$$\mathbb{E}[\bar{\ell}] - 1.01\hat{\mathbb{E}}[\bar{\ell}] \lesssim \frac{(A\mu(\boldsymbol{W}))^{\frac{2}{3}}(\theta B)^{\frac{1}{3}}}{n^{\frac{1}{3}}} + \frac{A\sqrt{B\nu(\boldsymbol{W})\theta}}{\sqrt{n}} + \frac{BA^2\theta}{n\left(\log^2\left(\frac{BA^2\theta}{\nu(\boldsymbol{W})n}\right)+1\right)} + \zeta. \tag{4}$$

Here with some fixed bound $B > 0$, we define

$$\begin{aligned} &\bar{\ell} = min\{\ell, B\}, \\ &\|\boldsymbol{W}\|_{2,1} = \sum_j \sqrt{\sum_i \left(W_{ij}^2\right)}, \\ &\theta = \ln^3(nM)\max_i \|x_i\|_2^2, \\ &\zeta = \tfrac{B(\ln(1/\delta)+\ln\ln n)}{n}. \end{aligned} \tag{5}$$

So one can guarantee good generalization when both the norm of the Jacobian matrix and the trace of the Hessian matrix are small. On one hand, when learning with gradient descent, we want to find a local or global minima of the loss function. Naturally, at minima the gradient should be zero and the norm of the Jacobian matrix is small near minima. So gradient descent helps us to ensure the norm of the Jacobian matrix is small.

On the other hand, gradient descent only considers first-order derivative, it can not constrain the trace of the Hessian matrix, which is composed of the second-order derivative. From this aspect, it is necessary to add a restriction on the Hessian trace when updating parameters by gradient descent. Though the above generalization error bound holds for linear models, it is natural to generalize from linear models to DNN models: each layer of a DNN model can be viewed as a linear model except for the non-linear activation functions.

Thus, the generalization error bound indicates that restriction on Hessian trace is essential for training a DNN model.

3.3. Flat Minima

Previous work Hochreiter and Schmidhuber [12], Keskar et al. [17], Dinh et al. [6] show that flat minima have good generalization. A flat minima is a large connected region in parameter space where the error remains approximately constant. The Bayesian argument suggests that flat minima correspond to "simple" networks and prevent over-fitting.

We start by writing Taylor expansion of loss function ℓ at a minima ω_*. Since at convergence of local minima ω_*, the first-order derivative of ℓ is 0, then we have

$$\ell(\omega) = \ell(\omega_*) + (\omega - \omega_*)^T \boldsymbol{H}(\omega - \omega_*) + o\left(\|\omega - \omega_*\|^2\right),$$

where $\boldsymbol{H}$ denotes the Hessian matrix of ℓ with respect ω_*.

Each eigenvalue of $\boldsymbol{H}$ indicates the extent of how ℓ changes with minor perturbation on ω on the corresponding eigenvector direction. Then smaller eigenvalue suggests that the minima is flat in the corresponding eigenvector direction. If the Hessian $\boldsymbol{H}$ at ω_* has all eigenvalues with smaller magnitudes, then it is a relatively flatter minima. Hence, penalizing on eigenvalues of Hessian can help lead to a flat minima.

Motivated by the theoretical analysis mentioned above, it's necessary to develop a regularizer using second-order information, especially the eigenvalues of Hessian. However, Hessian is a symmetric matrix with a tremendous amount of elements, i.e. $O(n^2)$ where n is the parameter size commonly occurs to be million level in NNs. Calculating the full Hessian of size $O(n^2)$ is intractable neither in a computation perspective nor in the memory constraint of modern computation resources. Even though we have the Hessian, the computation of eigenvalues is also complicated. Thus, we need to find a scalar index that distills the information of Hessian, which is cheap to compute and has similar properties as eigenvalues. As the sum of eigenvalues, the Hessian trace is a good option. Penalizing Hessian trace implies that the magnitudes of Hessian eigenvalues can be constrained, enhancing the chance of finding flat minima.

3.4. Linear Stability Analysis

In this subsection, we focus on the optimization process of parameter ω during the training process and the local properties around the minima.

The optimization process through stochastic gradient descent can be regarded as a motion process in the parameter space, from the landscape of initialization to convergence at a local or global minima. At each discrete step, the parameter as a high-dimensional vector takes the gradient as a moving direction. Then gradient descent can be formulated as a series of discrete updates:

$$\omega_{t+1} = \omega_t - \eta g_t, \tag{6}$$

where ω_t is the parameter position at step t, η is learning rate and gradient g_t is written as:

$$g_t = g_t(\omega_t, x) = \frac{d\ell(f(x;\omega_t), y)}{d\omega_t}.$$

Considering learning rate as the discrete time interval of updating weights, i.e. $\Delta t = \eta$, and denoting Δw as the parameter update, we have:

$$\frac{\Delta\omega}{\Delta t} = -g(\omega, x). \tag{7}$$

Under the assumption that time interval is small enough, approximately we can reformulate the discrete update rule into a contiunous form:

$$\frac{d\omega}{dt} = -g(\omega, x). \tag{8}$$

Thereafter, with an initial condition, we have the complete trajectory of parameter point based on Ordinary Differential Equation (ODE) theory. The process of gradient descent is transformed to a Nonlinear Dynamical System (NDS), which allows us to leverage the basic theory of linear stability analysis for developing new methods.

According to nonlinear dynamical systems, the updating gradient $-g$ in Eq. 8 is referred to as rate function. Denoted as the equilibrium points ω_* of an NDS, the minima are such parameters where the rate function vanishes $-g(\omega_*) = 0$. If any solution starting near an equilibrium point leaves the neighborhood of ω_* as $t \to \infty$, then ω_* is called **asymptotically unstable**, while if all solutions starting within the neighborhood approach ω_* as $t \to \infty$ then the equilibrium is called **asymptotically stable**. Lyapunov [21] gave more rigorous definition and discussion, known as **Lyapunov Stability Theory**. Intuitively, during neural network training, it's beneficial to have the optimizer find unstable minima along the moving trajectory in the parameter space. In this way, it's easier for the weight vector to jump out of the local minima and to search for a better descent direction.

In an NDS, the stability of an equilibrium point is highly correlated with the Jacobian matrix denoted as $\boldsymbol{J}$, which is the Jacobian matrix of rate function $-g$ w.r.t. to the weights ω. The Theorem of Lyapunov Stability [32] states that at the equilibrium point ω_*, if all eigenvalues of the Jacobian $\boldsymbol{J}(\omega_*)$ have real parts less than zero, then ω_* is Lyapunov stable.

Back to gradient descent, the Jacobian matrix in a NDS is exactly the negative of the Hessian matrix in a neural network. Denoting the Hessian matrix of the loss function ℓ with respect to parameter ω as $\boldsymbol{H}$, we have

$$\boldsymbol{H} = \frac{\partial}{\partial \omega}\left(\frac{\partial \ell}{\partial \omega}\right) = \frac{\partial g}{\partial \omega} = -\boldsymbol{J}(\omega_*).$$

The Hessian $\boldsymbol{H}$ is a real symmetric matrix and all its eigenvalues are real numbers. Since the Hessian trace is the sum of the eigenvalues, penalizing the Hessian trace helps decrease the eigenvalues to some extent. Then it increases the eigenvalues of the Jacobian matrix in the NDS, avoiding the eigenvalues to be negative. Thus, the constraint on the Hessian trace has a inclination to weaken the Lyapunov stability of the minima.

Despite reducing stability sounds bad, it is beneficial for escaping the local minima towards finding global minima. A Lyapunov stability point is referred to as an easily-converged equilibrium point. Once the parameter ω gets close to the Lyapunov stable equilibrium point, ω has little probability of escaping from this equilibrium point. Thus, falling to a stable equilibrium point will trap the optimization from finding a better local minima. Similar to the idea of the Confidence Penalty, regularizing on the Hessian trace can help the model to rethink and to escape these 'easily converged' equilibrium points. Avoiding these Lyapunov stable points can make the gradient method find a better equilibrium point for generalization.

3.5. Hessian regularization

We define the Hessian regularization term as

$$tr(\boldsymbol{H}_{\ell,\omega}). \tag{9}$$

It's the trace of second derivative of empirical loss ℓ with respect to parameters ω. Then, we define a new loss with our Hessian regularization as

$$Loss = \ell_{emp}(f) + \lambda \cdot tr(\boldsymbol{H}_{\ell,\omega}), \tag{10}$$

where λ controls the strength of the Hessian regularization. Based on previous analysis on the generalization error bound, flat minima and Lyapunov stability, the Hessian regularization is expected to improve generalization performance.

4. Algorithms

As discussed before, there are more than millions of parameters in a typical DNN. So the calculation of the Hessian matrix is difficult. Thus, we introduce two efficient stochastic algorithms to estimate Hessian trace, SEHT-H in subSection 4.1 and SEHT-D in subSection 4.2.

4.1. Hutchinson Method

Hutchinson Method [2] is an unbiased estimator for the trace of a matrix. Let $\boldsymbol{A}$ be an $n \times n$ symmetric matrix with $tr(\boldsymbol{A}) \neq 0$. Let σ be a random vector whose entries are i.i.d Rademacher random variables ($Pr(\sigma_i = \pm 1) = \frac{1}{2}$), then $\sigma^T \boldsymbol{A}\sigma$ is an unbiased estimator of $tr(\boldsymbol{A})$, based on the following equation:

$$\begin{aligned} tr(\boldsymbol{A}) &= tr(\boldsymbol{A}\boldsymbol{I}) = tr\left(\boldsymbol{A}\mathbb{E}\left[\sigma\sigma^T\right]\right) = \mathbb{E}\left[tr\left(\boldsymbol{A}\sigma\sigma^T\right)\right] = \mathbb{E}\left[tr\left(\sigma^T\boldsymbol{A}\sigma\right)\right] \\ &= \mathbb{E}\left[\sigma^T\boldsymbol{A}\sigma\right]. \end{aligned} \tag{11}$$

In this paper, we consider the trace of Hessian matrix $\boldsymbol{H}$, which is the second derivative matrix. Since the Rademacher random vector is irrelevant to network parameters,

$$\frac{d\sigma}{d\omega} = 0.$$

Then we expand the expression of Hutchinson estimator as follow:

$$\begin{aligned} \sigma^T\boldsymbol{H}\sigma &= \sigma^T \frac{d}{d\omega}\left(\frac{dl}{d\omega}\right)\sigma = \sigma^T\left[\frac{d}{d\omega}\left(\frac{dl}{d\omega}\right)\cdot\sigma + \frac{dl}{d\omega}\cdot\frac{d\sigma}{d\omega}\right] \\ &= \sigma^T \frac{d}{d\omega}\left(\frac{dl}{d\omega}\cdot\sigma\right). \end{aligned} \tag{12}$$

Based on Eq. 12, we can estimate the Hessian trace by calculating the gradient of loss $g_\omega = \frac{dl}{d\omega}$ and the gradient of $\frac{dl}{d\omega}\cdot\sigma$. We do not need the prohibitive computation of the whole Hessian matrix. The fast second-order information estimation only includes two inner products and two gradients. We refer to the Hutchinson stochastic estimator of Hessian trace as SEHT-H, presented in Algorithm 1. In practice, we only focus on the weight parameters in each layer of DNN and ignore the bias parameters.

Algorithm 1: SEHT-H

Input: n-dimensional gradient g
Output: Estimation of $tr(\boldsymbol{H})$
for $i = 1$ **to** *maxIter* **do**
  $\sigma \sim Rademacher(n)$
  $v = g \cdot \sigma$
  $h = dv/d\omega$
  $t = \sigma^T h$
  $trace += t$
end for
Return: $trace/maxIter$

Even though SEHT-H is a stochastic algorithm which reduces considerable amount of computational overhead, it is still not efficient enough due to the great number of parameters of a neural network. Hence, we propose to modify the pipeline of SEHT-H based on the basic idea of Dropout. The Dropout method boosts its computation speed and make it more efficient for neural network training.

4.2. Dropout Method

Inspired by Dropout [27], we then propose a stochastic parametric method to accelerate SEHT-H. In Dropout, every node in a neural network has a probability p to be ignored in the training process to reduce co-adaptations. Thus, in each training iteration, only a random sub-network of the original network is used. Intuitively, in the process of Hessian trace calculation, not all the parameters are necessary to be considered. it would be much faster if we only use a small subset of the network parameters during each calculation. Thus, we ignore some parameters when constraining Hessian trace $tr(\boldsymbol{H})$. The sum of the selected subset of diagonal elements is denoted as $\tilde{tr}(\boldsymbol{H})$. Considering the layer structures of neural networks, the process of randomized parameter selection can be divided into two steps: (i) randomly select layers in neural network with probability p_1, and (ii) randomly select parameters in the selected layers with probability p_2. In other words, when carrying out Hessian regularization, we ignore layers with probability $1 - p_1$, ignore parameters in the selected layers with probability $1 - p_2$. In our experiment, we simply set $p_1 = p_2$.

Following the basic idea of Hutchinson algorithm, we want to obtain Hessian trace without the heavy calculation of Hessian matrix. To extend the algorithm from full-parameter domain to partial-parameter domain, here we define a new probability distribution $Q(p)$. If $\boldsymbol{x} \sim Q(p)$, then

$$\begin{aligned} Pr(\boldsymbol{x} = \pm 1) &= p, \\ Pr(\boldsymbol{x} = 0) &= 1 - 2p. \end{aligned}$$

Then, supposing that σ is a random vector whose entries are i.i.d. random variables following the Q distribution, we have

$$\mathbb{E}\left[\sigma\sigma^T \mid \text{fixthepositionsof 0 in } \sigma\right] = \widetilde{\boldsymbol{I}}. \tag{13}$$

Here $\widetilde{I} = \text{diag}(0, 1)$ is a diagonal matrix with diagonal elements equal to 0 or 1. Notice that, the non-diagonal entries of $\mathbb{E}[\sigma\sigma^T]$ are all zero, because for $i \neq j$,

$$\mathbb{E}[\sigma_i\sigma_j] = 1 \cdot 2p^2 + (-1) \cdot 2p^2 = 0.$$

Then similar to Eq. 11, if we fix the positions of 0 in σ, we have unbiased estimator of the partial sum of diagonal elements:

$$\tilde{tr}(\boldsymbol{A}) = tr\left(\boldsymbol{A}\widetilde{\boldsymbol{I}}\right) = tr(\boldsymbol{A}\mathbb{E}[\sigma\sigma^T]) = \mathbb{E}[tr(\boldsymbol{A}\sigma\sigma^T)] = \mathbb{E}[\sigma^T\boldsymbol{A}\sigma]. \tag{14}$$

We can expand the expression same as Eq. 12 and transform the calculation process into two inner products and two gradients. This efficient method with random selected subset of parameters for calculating Hessian trace is presented in Algorithm 2. We name it as SEHT-D.

Algorithm 2: SEHT-D

Input: probability p, parameter ω in selected layers, and corresponding n-dim gradient g
Output: Estimation of $\tilde{tr}(\boldsymbol{H})$
for $i = 1$ **to** *maxIter*bf do
  $\sigma \sim Q(p)$
  $v = g \cdot \sigma$
  $h = dv/d\omega$
  $t = \sigma^T h$
  $trace += t$
end for
Return: $trace/maxIter$

SEHT-D makes Hessian regularization tractable for neural network training. Thus, in practice, we mainly provide results of SEHT-D. Compared with other regularization methods, including Label Smoothing and Confidence Penalty, our Hessian regularization shows improved test performance.

5. Experiments

We evaluate the proposed Hessian regularization method (abbreviated as SEHT) on the tasks of image classification and language modeling respectively, which are commonly used to measure the effectiveness of regularization.

For comparison across a variety of datasets, we adopt several existing regularization methods and data augmentation methods, e.g., Confidence Penalty [23], Label Smoothing [28], Cutout [5], MixUp [38]. To implement existing methods on our backbone model for fair comparison, we have done a thorough hyperparameter search to extract the best performance of each regularization method.

5.1. Image Classification

5.1.1. CIFAR-10

The CIFAR-10 dataset consists of 60000 instances of 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. On CIFAR-10 experiment, we use ResNet-18 [9] as the backbone neural network.

For all models, we set coefficient of weight decay of 5×10^{-4}. We set learning rate 0.01, batch size 32, momentum 0.9 and all models are trained 200 epochs with Cosine Annealing [19]. For Jacobian regularization, we set number of projections $n_{proj} = 1$ and weight parameter $\lambda_{JR} = 0.01$. For DropBlock, *block_size* = 7 and *keep_prob* = 0.9. We perform a grid search over weight values {0.001, 0.005, 0.01, 0.05, 0.1} and select 0.01 for Label Smoothing, 0.001 for Confidence Penalty. Cutout size is set to 16×16 based on the validation results mentioned by DeVries and Taylor [5]. For Mixup, we set $\alpha = 1.0$ according to Zhang et al. [38]'s setting, which results in interpolations of λ uniformly distributed between 0 and 1.

For the proposed Hessian regularization SEHT-D, we perform a grid search over weight values {0.0001, 0.001, 0.01, 0.1} and finally select 0.001, testing with probability value 0.01 and 0.05. We provide results on different possibilites on the max number of iterations that updates the trace calculation, defined in Algorithm 1. It is denoted as maxIters and with a set candidate numbers in {1, 5, 10}. We also test SEHT-H (maxIter = 5), with weight value in {0.0001, 0.001, 0.01, 0.1} and select 0.001 for SEHT-H.

The top-1 accuracy is reported in Table 1 with standard errors over 5 runs.

Firstly, SEHT demonstrates its superb effectiveness in improving generalization by providing the highest test accuracy on CIFAR-10. Furthermore, both SEHT-H and SEHT-D have relatively small standard errors, which means they are stable in their performance.

Our first observation falls on comparison with classical regularization and data augmentation methods. it is surprising to find out that Jacobian regularization and Dropblock method have even worse performance than the baseline with Weight-Decay. Confidence Penalty and Label Smoothing are two popular regularization methods that penalize the prediction distribution widely adopted for training a model with better generalization ability. They indeed surpass the baseline performance but with the limited improvement of 0.4%. The best results among this category is data augmentation with MixUp [38] that provides a 95.39% accuracy, but our method still consistently provide competitive or better results under different hyperparameter settings.

Recent work such as Vanilla [3], MM + FRL [40], Lookahead [39] provide inferior accuracy to the proposed SEHT-D. In [25]'s experiment, they got test accuracy 88.13% on CIFAR-10 with ResNet-18, which is much worse than our result: 95.37% with SEHT-D (maxIter = 1, prob = 0.01) and 95.49% with SEHT-D (maxIter = 10, prob = 0.05). Moreover, their improvement is only 0.02% for full-network and 0.10% for middle-network. Our Hessian regularization method improves the model 1.37% on test accuracy with SEHT-D (maxIter = 1, prob = 0.01) and improves 1.49% on test accuracy with SEHT-D (maxIter = 10, prob = 0.05), which are much more than their improvement. These results demonstrate the effectiveness of penalizing the Hessian trace.

Table 1
Results of ResNet-18 on CIFAR-10 over 5 runs.

Model	Test Accuracy(%)
Baseline with Weight-Decay	94.00 ± 0.24
Jacobian [15]	89.23 ± 0.52
DropBlock [8]	89.23 ± 0.22
Confidence Penalty [23]	94.40 ± 0.02
Label Smoothing [28]	94.40 ± 0.07
cutout [5]	94.02 ± 0.22
mixup [38]	95.39 ± 0.13
Vanilla [3]	93.6
Middle Network Method [25]	88.13 ± 0.12
MM + FRL [40]	95.33 ± 0.12
Lookahead [39]	95.23 ± 0.19
SEHT-D(maxIter = 1, prob = 0.01)	95.37 ± 0.09
SEHT-D(maxIter = 10, prob = 0.01)	95.42 ± 0.11
SEHT-D(maxIter = 10, prob = 0.05)	95.49 ± 0.06
SEHT-H(maxIter = 5)	$\mathbf{95.59 \pm 0.06}$

Secondly, we provide different setting combinations of the maxIter parameter and the prob parameter. A trade-off can be found between computational cost and performance. SEHT-H achieves the best performance regardless of training efficiency. On the other hand, SEHT-D(maxIter = 1, prob = 0.01) only requires 1.2× training time of the baseline and SEHT-D (maxIter = 1, prob = 0.05) costs only 1.3× of the baseline, while SEHT-H is much slower. The larger maxIter and prob are, the more time it would take in the algorithm. As a result, SEHT-D achieves a balance between performance and time efficiency in this experiment.

Finally, the convergence results are is provided in Fig. 1, where we compared the convergence speed and test accuracy on SEHT-D and SEHT-H with different parameters. It's interesting to notice that although setting different maxIter and prob can lead to different convergence trajectories, they all fall into a similar range of accuracy that clearly diverges from the baseline. This also provides evidence that SEHT may find a better minima by jumping out of stable equilibrium points.

In short, our SEHT-D and SEHT-H converge faster and better than the baseline. When increasing maxIter and prob, our SEHT-D shows better test accuracy but costs more time.

5.1.2. CIFAR-100

CIFAR-100 dataset is similar to the CIFAR-10 dataset, except that the target space are separated into 100 classes.

As widely adopted on CIFAR-100, Wide Residual Networks (WRNs) are also used in our experiments as the backbone neural network. Specifically, WRN-28–10 is used with depth 28 and fixed widening factor of 10. For all models, Weight Decay is set to 5×10^{-4}, batch size to 32, momentum to 0.9, and models are trained 200 epochs. The learning rate is initially set to 0.1 and is scheduled to decrease by a factor of 5 at 60, 120, and 160 epochs. Different from CIFAR100, we set the Dropout probability to be 0.3 suggested by Zagoruyko and Komodakis [37]'s cross-validation since it's more difficult to learn 100 classes than CIFAR10. Cutout size of 8 × 8 pixels is used according to DeVries and Taylor [5]'s validation results. For mixup, we keep with $\alpha = 1.0$. A grid search over weight values {0.0001, 0.001, 0.01, 0.1} is applied for Label Smoothing, Confidence Penalty and SEHT. Then a weight value of 0.1 is used for all these three methods. We report the averaged accuracy with standard error over 5 random initialization and the results are presented in Table 2.

In this experiment, our SEHT-D(maxIter = 1, prob = 0.01) method shows better results on both top-1 accuracy and top-5 accuracy, improving 4.92% and 2.52% respectively, when comparing with the baseline. SEHT-D(maxIter = 1, prob = 0.05) improves 5.70% and 2.48% respectively, outperforming all other methods tested. When testing together with Dropout, our SEHT-D has lower accuracy, which means it may have not good combination with Dropout.

Table 2
Results of WRN-28–10 on CIFAR-100.

Model	Top-1 Acc	Top-5 Acc
Baseline	74.61 ± 0.52	92.48 ± 0.40
Confidence Penalty	77.15 ± 1.54	93.97 ± 0.66
Label Smoothing	79.38 ± 0.26	94.39 ± 0.33
cutout	76.70 ± 0.79	93.72 ± 0.40
mixup	78.38 ± 0.31	94.37 ± 0.31
SEHT-D(maxIter = 1, prob = 0.01)	$\underline{79.53 \pm 0.72}$	$\mathbf{95.00 \pm 0.24}$
SEHT-D(maxIter = 1, prob = 0.05)	$\mathbf{80.31 \pm 0.33}$	$\underline{94.96 \pm 0.05}$

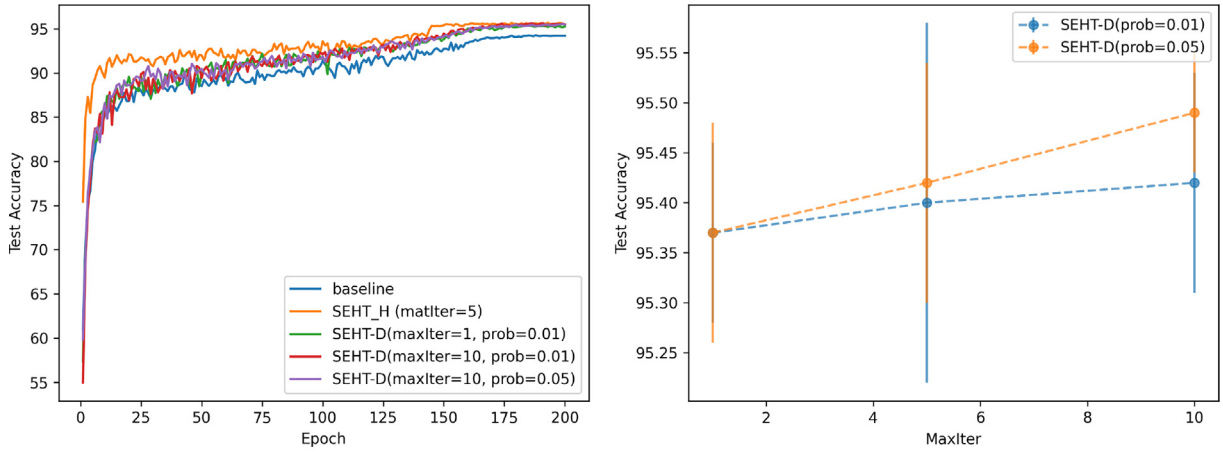
On CIFAR-100, the proposed SEHT-D method shows the best results on both top-1 accuracy and top-5 accuracy compared to other methods. Separately, SEHT-D with prob = 0.01 achieves the best top-5 accuracy of 95.00% with a 0.61% improvement than Label Smoothing while SEHT-D with prob = 0.05 outperforms Label Smoothing on top-1 accuracy drastically with around 1% improvement. The latter one has smaller standard errors on both top-1 accuracy and top-5 accuracy, indicating that SEHT-D (maxIter = 1, prob = 0.05) yields consistent good performance.

5.2. Language Modeling

5.2.1. Wiki-text2

The Wiki-Text language modeling dataset is a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia.

A 2-layer LSTM [13] is applied as the backbone model. The size of word embeddings is 512 and the number of hidden units per layer is 512. The LSTM model is trained for 40 epochs, with batch size 20, gradient clipping 0.25, and Dropout ratio 0.5. The Dropout ratio is searched from {0, 0.1, 0.2, 0.3, 0.4, 0.5} and 0.5 is chosen for best performance. The initial learning rate is tuned from {0.001, 0.01, 0.1, 0.5, 1, 10, 20, 40} and decreases by a factor of 4 when the validation error saturates and selects 20 to be the best. Parameters are initialized from a uniform distribution $[-0.1, 0.1]$. We perform the same grid search over weight values {0.001, 0.005, 0.01,



(a) Convergence on CIFAR-10 (b) Test Accuracy of SEHT-D on CIFAR-10

Fig. 1. Performance of SEHT-D on CIFAR-10.

Table 3
Results of LSTM on Wiki-Text2 over 5 runs (lower is better).

Model	Valid ppl	Test ppl
Baseline	101.82 ± 0.16	95.65 ± 0.10
Confidence Penalty	101.39 ± 0.16	95.57 ± 0.06
Label Smoothing	$\mathbf{99.58 \pm 0.06}$	95.03 ± 0.30
SEHT-D(maxIter = 1, prob = 0.05)	100.69 ± 0.27	$\mathbf{94.86 \pm 0.26}$

Table 4
Results of GRU on Wiki-Text2 over 5 runs (lower is better).

Model	Valid ppl	Test ppl
Baseline	119.04 ± 2.38	111.64 ± 1.87
Confidence Penalty	116.40 ± 0.08	109.27 ± 0.03
Label Smoothing	117.47 ± 0.24	110.46 ± 0.45
SEHT-D(maxIter = 1, prob = 0.01)	$\mathbf{116.21 \pm 0.31}$	$\mathbf{109.03 \pm 0.15}$

0.05, 0.1} for Label Smoothing, Confidence Penalty, and SEHT. The weight value of 0.01 works best for all these three methods. The probability value of 0.05 is found to outperform the value of 0.01 for Hessian regularization. We report the mean and standard error of perplexity over 5 random initialization in Table 3.

In this experiment with LSTM backbone, SEHT-D obtains the best test perplexity and Label Smoothing shows the best validation perplexity. SEHT-D improves the model by 0.79 on test perplexity. Confidence Penalty performs only slightly better than the baseline method.

For the 2-layer GRU [4] model as the backbone, the same hyperparameter search is performed. The size of word embeddings is 512 and the number of hidden units per layer is 512. We run every algorithm for 40 epochs, with batch size 20, gradient clipping 0.25, Dropout ratio 0.3, and initial learning rate 20. Parameters are initialized from a uniform distribution $[-0.1, 0.1]$. The weight value is set to be 0.05 for Label Smoothing, 0.005 for Confidence Penalty, and 0.001 for our Hessian regularization, after the grid search over {0.001, 0.005, 0.01, 0.05, 0.1}. Finally, the probability value is 0.01 in Hessian regularization. We repeat each method over 5 random initialization and results are presented in Table 4.

The Hessian regularization method has both the best validation perplexity and the best test perplexity, improving by 2.83 and 2.61 respectively compared with the baseline method. Confidence Penalty surpasses Label Smoothing on GRU model. Label Smoothing also shows better results than baseline.

Our experiments on Language Modelling demonstrate that all these three regularization methods can improve models, while our SEHT-D achieves the best performance.

6. Conclusion and Future Work

We propose a new regularization method named Stochastic Estimators of Hessian Trace (SEHT). Our method is motivated by a guarantee bound that a lower trace of the Hessian can result in a better generalization error. It can help escape Lyapunov stable points and find flat minima. To simplify computation, our method implements two versions, SEHT-H and SEHT-D. Our experiments show that SEHT-D and SEHT-H yield promising test performance. Particularly, the SEHT method achieves 95.59% accuracy on CIFAR-10 with resnet-18, outperforming classical regularization methods like Label Smoothing and Confidence Penalty.

Future works can focus on the following aspects: i) a faster estimator on Hessian trace; ii) convergence analysis by gradient descent with second order constraint; iii) generalization error bound for multi-layer neural networks.

Code Availability

Our code is available at https://github.com/Dean-lyc/Hessian-Regularization.

CRediT authorship contribution statement

Yucong Liu: Conceptualization, Methodology, Software, Writing - original draft. **Shixing Yu:** Software, Writing - review & editing. **Tong Lin:** Supervision, Writing - review & editing.

Data availability

Data will be made available on request.

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Tong Lin reports financial support was provided by National Natural Science Foundation of China. Tong Lin reports financial support was provided by National Key R&D Program of China. Tong Lin reports financial support was provided by Beijing Academy of Artificial Intelligence..

Acknowledgement

This work was supported by National Key R&D Program of China (No. 2018AAA0100300) and Beijing Academy of Artificial Intelligence (BAAI).

References

[1] A. Amanlou, A.A. Suratgar, J. Tavoosi, A. Mohammadzadeh, A. Mosavi, Single-image reflection removal using deep learning: A systematic review, IEEE Access 10 (2022) 29937–29953.
[2] H. Avron, S. Toledo, Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix, Journal of the ACM (JACM) 58 (2011) 1–34.
[3] L. Bungert, T. Roith, D. Tenbrinck, M. Burger, A Bregman learning framework for sparse neural networks, Journal of Machine Learning Research 23 (2022) 1–43.
[4] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics. pp. 1724–1734.
[5] DeVries, T., Taylor, G.W., 2017. Improved regularization of convolutional neural networks with Cutout. arXiv preprint arXiv:1708.04552.
[6] L. Dinh, R. Pascanu, S. Bengio, Y. Bengio, Sharp minima can generalize for deep nets, International Conference on Machine Learning (2017) 1019–1028.
[7] Z. Ebrahimi-Khusfi, R. Taghizadeh-Mehrjardi, F. Roustaei, M. Ebrahimi-Khusfi, A.H. Mosavi, B. Heung, M. Soleimani-Sardo, T. Scholten, Determining the contribution of environmental factors in controlling dust pollution during cold and warm months of western iran using different data mining algorithms and game theory, Ecological Indicators 132 (2021).
[8] G. Ghiasi, T.Y. Lin, Q.V. Le, Dropblock: A regularization method for convolutional networks, Advances in Neural Information Processing Systems 31 (2018) 10727–10737.
[9] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.
[10] D.P. Helmbold, P.M. Long, On the inductive bias of dropout, The Journal of Machine Learning Research 16 (2015) 3403–3454.
[11] Hendrycks, D., Gimpel, K., 2016. Gaussian error linear units (GELUS). arXiv preprint arXiv:1606.08415.
[12] S. Hochreiter, J. Schmidhuber, Flat minima, Neural computation 9 (1997) 1–42.
[13] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (1997) 1735–1780.
[14] A.E. Hoerl, R.W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, Technometrics 12 (1970) 55–67.
[15] Hoffman, J., Roberts, D.A., Yaida, S., 2019. Robust learning with Jacobian regularization. Conference on the Mathematical Theory of Deep Learning (DeepMath).

[16] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, International conference on machine learning (2015) 448–456.
[17] Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P., 2016. On large-batch training for deep learning: Generalization gap and sharp minima. arXiv preprint arXiv:1609.04836.
[18] Krogh, A., Hertz, J., 1992. A simple weight decay can improve generalization. Advances in Neural Information Processing Systems 4.
[19] Loshchilov, I., Hutter, F., 2016. SGDR: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983.
[20] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, International Conference on Learning (2019), Representations.
[21] A.M. Lyapunov, The general problem of the stability of motion, International journal of control 55 (1992) 531–534.
[22] S. Nosratabadi, A. Mosavi, R. Keivani, S. Ardabili, F. Aram, State of the art survey of deep learning and machine learning models for smart cities and urban sustainability, in: A.R. Várkonyi-Kóczy (Ed.), Engineering for Sustainable Future, Springer International Publishing, Cham, 2020, pp. 228–238.
[23] Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł, Hinton, G., 2017. Regularizing neural networks by penalizing confident output distributions. arXiv preprint arXiv:1701.06548.
[24] H. Perez, J.H.M. Tah, A. Mosavi, Deep learning for detecting building defects using convolutional neural networks, Sensors 19 (2019).
[25] Sankar, A.R., Khasbage, Y., Vigneswaran, R., Balasubramanian, V.N., 2021. A deeper look at the hessian eigenspectrum of deep neural networks and its applications to regularization, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 9481–9488.
[26] J. Sokolić, R. Giryes, G. Sapiro, M.R. Rodrigues, Robust large margin deep neural networks, IEEE Transactions on Signal Processing 65 (2017) 4265–4280.
[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The journal of machine learning research 15 (2014) 1929–1958.
[28] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826.
[29] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society: Series B (Methodological) 58 (1996) 267–288.
[30] S. Wager, S. Wang, P.S. Liang, Dropout training as adaptive regularization, Advances in neural information processing systems 26 (2013) 351–359.
[31] C. Wei, S. Kakade, T. Ma, The implicit and explicit regularization effects of dropout, International Conference on Machine Learning (2020) 10181–10192.
[32] T. Witelski, M. Bowen, Methods of mathematical modelling, Springer, 2015.
[33] Xu, B., Wang, N., Chen, T., Li, M., 2015. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.
[34] Yao, Z., Gholami, A., Keutzer, K., Mahoney, M.W., 2020. PyHessian: Neural networks through the lens of the hessian, in: 2020 IEEE International Conference on Big Data (Big Data), IEEE. pp. 581–590.
[35] Yao, Z., Gholami, A., Shen, S., Mustafa, M., Keutzer, K., Mahoney, M., 2021. AdaHessian: An adaptive second order optimizer for machine learning. Proceedings of the AAAI Conference on Artificial Intelligence 35, 10665–10673.
[36] S. Yu, Z. Yao, A. Gholami, Z. Dong, S. Kim, M.W. Mahoney, K. Keutzer, Hessian-aware pruning and optimal neural implant, in: 2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), IEEE Computer Society, 2022, pp. 3665–3676.
[37] Zagoruyko, S., Komodakis, N., 2016. Wide residual networks, in: Proceedings of the British Machine Vision Conference (BMVC), pp. 87.1–87.12.
[38] Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D., 2018. Mixup: Beyond empirical risk minimization. International Conference on Learning Representations.
[39] Zhang, M., Lucas, J., Ba, J., Hinton, G.E., 2019. Lookahead optimizer: k steps forward, 1 step back. Advances in Neural Information Processing Systems 32.
[40] Zheng, R., Yu, Z., Zhang, Y., Ding, C., Cheng, H.V., Liu, L., 2020. Learning class unique features in fine-grained visual classification. arXiv preprint arXiv:2011.10951.

Yucong Liu is a second-year master's student in the Statistics department at the University of Chicago. He received his B.S in Data Science and Big Data Technology and B.S in Mathematics and Applied Mathematics from Peking University in 2021. His research interests contain but are not limited to deep learning theory, generalization theory, optimization, and approximation theory.

Shixing Yu is a Master's student at the ECE department, The University of Texas Austin. Right now, he works closely with Prof. Atlas(Zhangyang) Wang and Prof. Diana Marculescu. He received his bachelor of science degree from the EECS department at Peking University, China, majoring in computer science. His previous research focuses on using statistical methods to compress neural networks by designing efficient architecture and algorithms. He is generally interested in energy-efficient solutions for machine learning, sparse neural network analysis, and explainable AI.

Tong Lin received the PhD degree in Applied Mathematics from Peking University in 2001. In 2002, he joined the Key Laboratory of Machine Perception at Peking University, China. Now he is an associate professor of School of Intelligence Science and Technology, Peking University. From 2004 to 2005, he was an exchange scholar at Seoul National University, Korea. From 2007 to 2008, he was an exchange scholar at UCSD Moores Cancer Center, CA, USA. His research interests are machine learning algorithms with applications in medical data analysis.