Optimize TSK Fuzzy Systems for Regression Problems: Minibatch Gradient Descent With Regularization, DropRule, and AdaBound (MBGD-RDA)

Dongrui Wu, *Senior Member, IEEE*, Ye Yuan, *Member, IEEE*, Jian Huang, *Senior Member, IEEE*, and Yihua Tan, *Member, IEEE*

***Abstract*—Takagi–Sugeno–Kang (TSK) fuzzy systems are very useful machine learning models for regression problems. However, to our knowledge, there has not existed an efficient and effective training algorithm that ensures their generalization performance and also enables them to deal with big data. Inspired by the connections between TSK fuzzy systems and neural networks, we extend three powerful neural network optimization techniques, i.e., minibatch gradient descent (MBGD), regularization, and AdaBound, to TSK fuzzy systems, and also propose three novel techniques (DropRule, DropMF, and DropMembership) specifically for training TSK fuzzy systems. Our final algorithm, MBGD with regularization, DropRule, and AdaBound, can achieve fast convergence in training TSK fuzzy systems, and also superior generalization performance in testing. It can be used for training TSK fuzzy systems on datasets of any size; however, it is particularly useful for big datasets, on which currently no other efficient training algorithms exist.**

***Index Terms*—AdaBound, Adaptive-network-based fuzzy inference system (ANFIS), DropRule, fuzzy systems, minibatch gradient descent (MBGD), regularization.**

NOMENCLATURE

Notations

N	Number of labeled training examples.
$\mathbf{x}_n$	$=(x_{n,1},\ldots,x_{n,M})^T$. M-dimensional feature vector of the nth training example, $n\in[1,N]$.
y_n	Groundtruth output corresponding to $\mathbf{x}_n$.
R	Number of rules in the TSK fuzzy system.
$X_{r,m}$	MF for the mth feature in the rth rule, $r\in[1,R]$ and $m\in[1,M]$.
$b_{r,0},\ldots,b_{r,M}$	Consequent parameters of the rth rule, $r\in[1,R]$.
$y_r(\mathbf{x}_n)$	Output of the rth rule for $\mathbf{x}_n$, $r\in[1,R]$ and $n\in[1,N]$.
$\mu_{X_{r,m}}(x_{n,m})$	Membership grade of $x_{n,m}$ on $X_{r,m}$, $r\in[1,R]$, $m\in[1,M]$, and $n\in[1,N]$.
$f_r(\mathbf{x}_n)$	Firing level of $\mathbf{x}_n$ on the rth rule, $r\in[1,R]$ and $n\in[1,N]$.
$y(\mathbf{x}_n)$	Output of the TSK fuzzy system for $\mathbf{x}_n$.
L	ℓ_2-regularized loss function for training the TSK fuzzy system.
λ	ℓ_2 regularization coefficient in ridge regression, MBGD-R, MBGD-RA, and MBGD-RDA.
M_m	Number of Gaussian MFs in each input domain.
N_{bs}	Minibatch size in MBGD-based algorithms.
K	Number of iterations in MBGD training.
α	Initial learning rate in MBGD-based algorithms.
P	DropRule rate in MBGD-D, MBGD-RD, and MBGD-RDA.
β_1, β_2	Exponential decay rates for moment estimates in AdaBound.
ϵ	Small positive number in AdaBound to avoid dividing by zero.

I. INTRODUCTION

FUZZY systems [1], particularly Takagi–Sugeno–Kang (TSK) fuzzy systems [2], have achieved great success in numerous applications. This article focuses on the applications of TSK fuzzy systems in machine learning [3], particularly supervised regression problems. In such problems, we have a training dataset with N labeled examples $\{\mathbf{x}_n, y_n\}_{n=1}^N$, where $\mathbf{x}_n\in\mathbb{R}^{M\times 1}$, and would like to train a TSK fuzzy system to model the relationship between y and $\mathbf{x}$ so that an accurate prediction can be made for any future unseen $\mathbf{x}$.

There are generally following three different strategies for optimizing a TSK fuzzy system in supervised regression.[1]

Manuscript received March 30, 2019; revised September 16, 2019 and November 28, 2019; accepted December 4, 2019. Date of publication December 9, 2019; date of current version May 4, 2020. This work was supported by the National Natural Science Foundation of China under Grant 61873321 and in part by Hubei Technology Innovation Platform under Grant 2019AEA171. *(Corresponding authors: Dongrui Wu; Yihua Tan.)*

The authors are with the Key Laboratory of Image Processing and Intelligent Control of Ministry of Education, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: drwu@hust.edu.cn; yye@hust.edu.cn; huang_jan@hust.edu.cn; yhtan@hust.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TFUZZ.2019.2958559

[1]Some novel approaches for optimizing evolving fuzzy systems have also been proposed recently [4], [5]; however, they are not the focus of this article, so their details are not included.

1) *Evolutionary algorithms* [6], in which each set of the parameters of the antecedent membership functions (MFs) and the consequents are encoded as an individual in a population, and genetic operators, such as selection, crossover, mutation, and reproduction, are used to produce the next generation. Generally, the overall fitness improves in each new generation, and a global optimum may be found given enough number of generations.
2) *Gradient descent* (GD) [7], in which the parameters are moved in the negative gradient direction of the loss function to find its local minimum. Backpropagation [8] is frequently used to calculate the gradients. These fuzzy systems are called *neurofuzzy systems* in the literature [9].
3) *GD and least squares estimation (LSE)* [10], which is used in the popular adaptive-network-based fuzzy inference system (ANFIS). The antecedent parameters are optimized by GD, and the consequent parameters by LSE. This approach usually converges much faster than using GD only.

However, all three strategies may have challenges in big data applications [11], [12]. It is well known that big data have at least three Vs[2] [13]: volume (the size of the data), velocity (the speed of the data), and variety (the types of data). Volume means that the number of training examples (N) is very large, and/or the dimensionality of the input (M) is very high. Fuzzy systems, and actually almost all machine learning models, suffer from the curse of dimensionality, i.e., the number of rules (parameters) increases exponentially with M. However, in this article, we assume that the dimensionality can be reduced effectively to just a few, e.g., using principal component analysis [14]. We mainly consider how to deal with large N.

Evolutionary algorithms are not suitable for optimizing TSK fuzzy systems when N is large, because they have very high memory and computing power requirement. They need to evaluate the fitness of each individual on the entire training dataset (which may be too large to be loaded into the memory completely), and there are usually tens or hundreds of individuals in a population and tens or hundreds of generations are needed to find a good solution. ANFIS may result in significant overfitting in regression problems, as demonstrated in Section III-E of this article. Therefore, we focus on GD.

When N is small, batch GD can be used to compute the average gradients over all N training examples, and then update the model parameters. When N is large, there may not be enough memory to load the entire training dataset, and hence batch GD may be very slow or even impossible to perform. In such cases, stochastic GD can be used to compute the gradients for each training example, and then update the model parameters. However, the stochastic gradients may have very large variance, and hence the training may be unstable. A good compromise between batch GD and stochastic GD, which has achieved great success in deep learning [15], is minibatch GD (MBGD). It randomly selects a small number (typically 32 or 64 [16]) of training examples to compute the gradients and update the model parameters. MBGD is a generic approach not specific to a particular model to be optimized, so it should also be applicable to the training of fuzzy systems. In fact, Nakasima-Lpez *et al.* [17] compared the performances of full-batch GD, MBGD, and stochastic GD on the training of Mamdani neurofuzzy systems, and showed that MBGD achieved the best performance. This article applies MBGD to the training of TSK fuzzy systems.

In MBGD, the learning rate is very important to the convergence speed and quality in training. Many different schemes, e.g., momentum [8], averaging [18], AdaGrad [19], RMSProp [20], and Adam [21], have been proposed to optimize the learning rate in neural network training. Adam may be the most popular one among them. However, to the knowledge of the authors, only a short conference paper [22] has applied Adam to the training of simple single-input rule modules' fuzzy systems [23]. Very recently, an improvement to Adam, AdaBound [24], was proposed, which demonstrated faster convergence and better generalization than by Adam. To our knowledge, no one has used AdaBound for training TSK fuzzy systems.

In addition to fast convergence, the generalization ability of a machine learning model is also crucially important. Generalization means that the model must perform well on previously unobserved inputs (not just the known training examples).

Regularization is frequently used to reduce overfitting and improve generalization. According to Goodfellow *et al.* [15], regularization is "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." It has also been used in training TSK fuzzy systems to increase their performance and interpretability [25]–[29]. For example, Johansen [25] and Lughofer and Kindermann [29] used ℓ_2 regularization (also known as weight decay, ridge regression, or Tikhonov regularization) to stabilize the matrix inversion operation in LSE. Jin [26] used regularization to merge similar MFs into a single one to reduce the size of the rulebase and, hence, to increase the interpretability of the fuzzy system. Lughofer and Kindermann [27] and Luo *et al.* [28] used sparsity regularization to identify a TSK fuzzy system with a minimal number of fuzzy rules and a minimal number of nonzero consequent parameters. All these approaches used LSE to optimize the TSK rule consequents, which may result in significant overfitting in regression problems (see Section III-E). To our knowledge, no one has integrated MBGD and regularization for TSK fuzzy system training.

Additionally, some unique approaches have also been proposed in the last few years to reduce overfitting and increase generalization of neural networks, particularly deep neural networks, e.g., DropOut [30] and DropConnect [31]. DropOut randomly discards some neurons and their connections during the training, which prevents units from coadapting too much. DropConnect randomly sets some connection weights to zero during the training. Although DropOut and DropConnect have demonstrated outstanding performance and hence widely used in deep learning, no similar techniques exist for training TSK fuzzy systems.

This article fills the gap in efficient and effective training of TSK fuzzy systems, particularly for big data regression problems. Its main contributions are as follows.

1) Inspired by the connections between TSK fuzzy systems and neural networks [32], we extend three powerful neural

[2]There may be also other Vs, e.g., veracity and value.

Algorithm 1: The MBGD-RDA Algorithm for TSK Fuzzy System Optimization. Typical Values of Some Hyper-Parameters are: $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Input: N labeled training examples $\{\mathbf{x}_n, y_n\}_{n=1}^N$, where $\mathbf{x}_n = (x_{n,1}, ..., x_{n,M})^T \in \mathbb{R}^{M\times 1}$;
$L(\boldsymbol{\theta})$, the loss function for the TSK fuzzy system parameter vector $\boldsymbol{\theta}$;
M_m, the number of Gaussian MFs in the mth input domain;
K, the maximum number of training iterations;
$N_{bs} \in [1, N]$, the mini-batch size;
$P \in (0, 1)$, the DropRule rate;
α, the initial learning rate (step size);
λ, the ℓ_2 regularization coefficient;
$\beta_1, \beta_2 \in [0, 1)$, exponential decay rates for the moment estimates in AdaBound;
ϵ, a small positive number in AdaBound;
$l(k)$ and $u(k)$, the lower and upper bound functions in AdaBound;
Output: The final $\boldsymbol{\theta}$.
for $m = 1, ..., M$ **do**
  Compute the minimum and maximum of all $\{x_{n,m}\}_{n=1}^N$;
  Initialize the center of the M_m Gaussian MFs uniformly between the minimum and the maximum;
  Initialize the standard deviation of all M_m Gaussian MFs as the standard deviation of $\{x_{n,m}\}_{n=1}^N$;
end
Initialize the consequent parameters of all R rules as 0;
$\boldsymbol{\theta}_0$ is the concatenation of all Gaussian MF centers, standard deviations, and rule consequent parameters;
$\mathbf{m}_0 = \mathbf{0}$; $\mathbf{v}_0 = \mathbf{0}$;
for $k = 1, ..., K$ **do**
  Randomly select N_{bs} training examples;
  for $n = 1, ..., N_{bs}$ **do**
    for $r = 1, ..., R$ **do**
      `// DropRule`
      $f_r(\mathbf{x}_n) = 0$;
      Generate p, a uniformly distributed random number in $[0, 1]$;
      if $p \leq P$ **then**
        Compute $f_r(\mathbf{x}_n)$, the firing level of $\mathbf{x}_n$ on Rule_r;
      end
    end
    Compute $y(\mathbf{x}_n)$, the TSK fuzzy system output for $\mathbf{x}_n$, by (4);
    `// Compute the gradients`
    for *each element* $\boldsymbol{\theta}_{k-1}(i)$ *in* $\boldsymbol{\theta}_{k-1}$ **do**

$$\mathbf{g}_k(i) = \begin{cases} \frac{\partial L}{\partial \boldsymbol{\theta}_{k-1}(i)}, & \text{if } \boldsymbol{\theta}_{k-1}(i) \text{ was used in computing } y(\mathbf{x}_n) \\ 0, & \text{otherwise} \end{cases}$$

    end
  end
  `// ` ℓ_2 `regularization`
  Identify the index set I, which consists of the elements of $\boldsymbol{\theta}$ corresponding to the rule consequent coefficients, excluding the bias terms;
  for *each index* $i \in I$ **do**
    $\mathbf{g}_k(i) = \mathbf{g}_k(i) + \lambda \cdot \boldsymbol{\theta}_{k-1}(i)$;
  end
  `// AdaBound`
  $\mathbf{m}_k = \beta_1 \boldsymbol{m}_{k-1} + (1 - \beta_1)\boldsymbol{g}_k$; $\boldsymbol{v}_k = \beta_2 \boldsymbol{v}_{k-1} + (1 - \beta_2)\boldsymbol{g}_k^2$;
  $\hat{\boldsymbol{m}}_k = \frac{\boldsymbol{m}_k}{1 - \beta_1^k}$; $\hat{\boldsymbol{v}}_k = \frac{\boldsymbol{v}_k}{1 - \beta_2^k}$;
  $\hat{\boldsymbol{\alpha}} = \max\left[l(k), \min\left(u(k), \frac{\alpha}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}\right)\right]$;
  $\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \hat{\boldsymbol{\alpha}} \odot \hat{\boldsymbol{m}}_k$;
end
Return $\boldsymbol{\theta}_K$

network optimization techniques, i.e., MBGD, regularization, and AdaBound, to TSK fuzzy systems.

2) We propose three novel techniques (DropRule, DropMF, and DropMembership) specifically for training TSK fuzzy systems.
3) Our final algorithm, MBGD with regularization, DropRule, and AdaBound (MBGD-RDA), demonstrates superior performance on ten real-world datasets from various application domains, of different sizes.

The remainder of this article is organized as follows. Section II introduces our proposed MBGD-RDA algorithm. Section III presents our experimental results. Section IV concludes this article and points out some future research directions.

II. MBGD-RDA Algorithm

This section introduces our proposed MBGD-RDA algorithm for training TSK fuzzy systems, whose pseudocode is given in Algorithm 1 and MATLAB implementation is available online.[3] Note that $\boldsymbol{\theta}_K$ returned from Algorithm 1 is not necessarily the

[3][Online]. Available: https://github.com/drwuHUST/MBGD_RDA

optimal one among $\{\boldsymbol{\theta}_k\}_{k=1}^K$, i.e., the one that gives the smallest test error. The iteration number corresponding to the optimal $\boldsymbol{\theta}^*$ can be estimated using early stopping [15]; however, this is beyond the scope of this article. Herein, we assume that the user has predetermined K.

The key notations used in this article are summarized in nomenclature. The details of MBGD-RDA are explained next.

A. TSK Fuzzy System

Assume the input $\mathbf{x} = (x_1, \ldots, x_M)^T \in \mathbb{R}^{M\times 1}$, and the TSK fuzzy system has the following R rules:

$$\text{Rule}_r : \text{IF } x_1 \text{ is } X_{r,1} \text{ and } \cdots \text{ and } x_M \text{ is } X_{r,M}$$
$$\text{THEN } y_r(\mathbf{x}) = b_{r,0} + \sum_{m=1}^{M} b_{r,m} x_m \tag{1}$$

where $X_{r,m}$ ($r = 1, \ldots, R$; $m = 1, \ldots, M$) are fuzzy sets, and $b_{r,0}$ and $b_{r,m}$ are consequent parameters. This article considers only Gaussian MFs, because their derivatives are easier to compute [33]. However, our algorithm can also be applied to other MF shapes, as long as their derivatives can be computed.

The membership grade of x_m on a Gaussian MF $X_{r,m}$ is

$$\mu_{X_{r,m}}(x_m) = \exp\left(-\frac{(x_m - c_{r,m})^2}{2\sigma_{r,m}^2}\right) \tag{2}$$

where $c_{r,m}$ is the center of the Gaussian MF, and $\sigma_{r,m}$ is the standard deviation.

The firing level of Rule_r is given as follows:

$$f_r(\mathbf{x}) = \prod_{m=1}^{M} \mu_{X_{r,m}}(x_m) \tag{3}$$

and the output of the TSK fuzzy system is

$$y(\mathbf{x}) = \frac{\sum_{r=1}^{R} f_r(\mathbf{x}) y_r(\mathbf{x})}{\sum_{r=1}^{R} f_r(\mathbf{x})} \tag{4}$$

or if we define the normalized firing levels as

$$\bar{f}_r(\mathbf{x}) = \frac{f_r(\mathbf{x})}{\sum_{k=1}^{R} f_k(\mathbf{x})}, \quad r = 1, \ldots, R \tag{5}$$

then, (4) can be rewritten as

$$y(\mathbf{x}) = \sum_{r=1}^{R} \bar{f}_r(\mathbf{x}) \cdot y_r(\mathbf{x}). \tag{6}$$

To optimize the TSK fuzzy system, we need to tune $c_{r,m}$, $\sigma_{r,m}$, $b_{r,0}$, and $b_{r,m}$, where $r = 1, \ldots, R$ and $m = 1, \ldots, M$.

B. Regularization

Assume there are N training examples $\{\mathbf{x}_n, y_n\}_{n=1}^N$, where $\mathbf{x}_n = (x_{n,1}, \ldots, x_{n,M})^T \in \mathbb{R}^{M\times 1}$.

In this article, we use the following ℓ_2-regularized loss function:

$$L = \frac{1}{2}\sum_{n=1}^{N_{bs}} [y_n - y(\mathbf{x}_n)]^2 + \frac{\lambda}{2}\sum_{r=1}^{R}\sum_{m=1}^{M} b_{r,m}^2 \tag{7}$$

where $N_{bs} \in [1, N]$, and $\lambda \geq 0$ is a regularization parameter. Note that $b_{r,0}$ ($r = 1, \ldots, R$) are not regularized in (7). As pointed out by Goodfellow *et al.* [15], for neural networks, one typically penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized. The biases typically require less data to fit accurately than the weights. The biases in neural networks are corresponding to the $b_{r,0}$ terms here, so we take this typical approach and leave $b_{r,0}$ unregularized.

C. Minibatch Gradient Descent

The gradients of the loss function (7) are given in (8)–(10), shown at bottom of this page, where $\Phi(r, m)$ is the index set of the rules that contain $X_{r,m}$, $x_{n,0} \equiv 1$, and $I(m)$ is an indicator

$$\frac{\partial L}{\partial c_{r,m}} = \frac{1}{2}\sum_{n=1}^{N_{bs}}\sum_{k=1}^{R} \frac{\partial L}{\partial y(\mathbf{x}_n)} \frac{\partial y(\mathbf{x}_n)}{\partial f_k(\mathbf{x}_n)} \frac{\partial f_k(\mathbf{x}_n)}{\partial \mu_{X_{k,m}}(x_{n,m})} \frac{\partial \mu_{X_{k,m}}(x_{n,m})}{\partial c_{r,m}}$$
$$= \sum_{n=1}^{N_{bs}} \sum_{k\in\Phi(r,m)} \left[(y(\mathbf{x}_n) - y_n) \frac{y_k(\mathbf{x}_n)\sum_{i=1}^{R} f_i(\mathbf{x}_n) - \sum_{i=1}^{R} f_i(\mathbf{x}_n) y_i(\mathbf{x}_n)}{\left[\sum_{i=1}^{R} f_i(\mathbf{x}_n)\right]^2} f_k(\mathbf{x}_n) \frac{x_{n,m} - c_{r,m}}{\sigma_{r,m}^2} \right] \tag{8}$$

$$\frac{\partial L}{\partial \sigma_{r,m}} = \frac{1}{2}\sum_{n=1}^{N_{bs}}\sum_{k=1}^{R} \frac{\partial L}{\partial y(\mathbf{x}_n)} \frac{\partial y(\mathbf{x}_n)}{\partial f_k(\mathbf{x}_n)} \frac{\partial f_k(\mathbf{x}_n)}{\partial \mu_{X_{k,m}}(x_{n,m})} \frac{\partial \mu_{X_{k,m}}(x_{n,m})}{\partial \sigma_{r,m}}$$
$$= \sum_{n=1}^{N_{bs}} \sum_{k\in\Phi(r,m)} \left[(y(\mathbf{x}_n) - y_n) \frac{y_k(\mathbf{x}_n)\sum_{i=1}^{R} f_i(\mathbf{x}_n) - \sum_{i=1}^{R} f_i(\mathbf{x}_n) y_i(\mathbf{x}_n)}{\left[\sum_{i=1}^{R} f_i(\mathbf{x}_n)\right]^2} f_k(\mathbf{x}_n) \frac{(x_{n,m} - c_{r,m})^2}{\sigma_{r,m}^3} \right] \tag{9}$$

$$\frac{\partial L}{\partial b_{r,m}} = \frac{1}{2}\sum_{n=1}^{N_{bs}} \frac{\partial L}{\partial y(\mathbf{x}_n)} \frac{\partial y(\mathbf{x}_n)}{\partial y_r(\mathbf{x}_n)} \frac{\partial y_r(\mathbf{x}_n)}{\partial b_{r,m}} + \frac{\lambda}{2}\frac{\partial L}{\partial b_{r,m}}$$
$$= \sum_{n=1}^{N_{bs}} \left[(y(\mathbf{x}_n) - y_n) \frac{f_r(\mathbf{x}_n)}{\sum_{i=1}^{R} f_i(\mathbf{x}_n)} \cdot x_{n,m} \right] + \lambda I(m) b_{r,m} \tag{10}$$

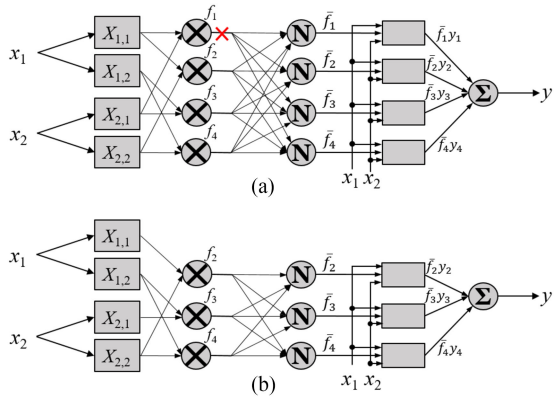

Fig. 1. DropRule, where $X_{m,i}$ is the ith MF in the mth input domain. (a) Red cross indicates that the first rule will be dropped. (b) Equivalent fuzzy system after dropping the first rule.

function given by:

$$I(m) = \begin{cases} 0, & m = 0 \\ 1, & m > 0. \end{cases}$$

$I(m)$ ensures that $b_{r,0}$ $(r = 1, \ldots, R)$ are not regularized.

In MBGD, each time we randomly sample $N_{bs} \in [1, N]$ training examples, compute the gradients from them, and then update the antecedent and consequent parameters of the TSK fuzzy system. Let $\boldsymbol{\theta}_k$ be the model parameter vector in the kth iteration, and $\partial L/\partial \boldsymbol{\theta}_k$ be the first-order gradients computed from (8)–(10). Then, the update rule is given by

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \alpha \frac{\partial L}{\partial \boldsymbol{\theta}_{k-1}} \tag{11}$$

where $\alpha > 0$ is the learning rate (step size).

When $N_{bs} = 1$, MBGD degrades to the stochastic GD. When $N_{bs} = N$, it becomes the batch GD.

D. DropRule

DropRule is a novel technique to reduce overfitting and increase generalization in training TSK fuzzy systems, inspired by the well-known DropOut [30] and DropConnect [31] techniques in deep learning. DropOut randomly discards some neurons and their connections during the training. DropConnect randomly sets some connection weights to zero during the training. DropRule randomly discards some rules during the training, but uses all rules in testing.

Let the DropRule rate be $P \in (0, 1)$. For each training example in the iteration, we set the firing level of a rule to its true firing level with probability P, and to zero with probability $1 - P$. The output of the TSK fuzzy system is again computed by a firing-level weighted average of the rule consequents. Since the firing levels of certain rules are artificially set to zero, they do not contribute anything to the fuzzy system output, i.e., they are artificially dropped for this particular training example, as shown in Fig. 1.[4] Then, GD is applied to update the model parameters in rules that are not dropped (the parameters in the dropped rules are not updated for this particular training example).

When the training is done, all rules will be used in computing the output for a new input, just as in a traditional TSK fuzzy system. This is different from DropOut and DropConnect for neural networks, which need some special operations in testing to ensure that the output does not have a bias. We do not need to pay special attention in using a TSK fuzzy system trained from DropRule, because the final step of a TSK fuzzy system is a weighted average, which removes the bias automatically.

The rationale behind DropOut is that [30] "each hidden unit in a neural network trained with dropout must learn to work with a randomly chosen sample of other units. This should make each hidden unit more robust and drive it toward creating useful features on its own without relying on other hidden units to correct its mistakes." This is also the motivation of DropRule: by randomly dropping some rules, we force each rule to work with a *randomly* chosen subset of rules, and hence each rule should maximize its own modeling capability, instead of relying too much on other rules. This may help increase the generalization of the final TSK fuzzy system. Our experiments in the next section demonstrate that DropRule alone may not always offer advantages, but it works well when integrated with an efficient learning rate adaptation algorithm such as AdaBound.

E. Adam and AdaBound

As mentioned in Section I, the learning rate is a very important hyperparameter in neural network training, which is also the case for TSK fuzzy system training. Among the various proposals for adjusting the learning rate, Adam [21] may be the most popular one. It computes an individualized adaptive learning rate for each different model parameter from the estimates of the first and second moments of the gradient. Essentially, it combines the advantages of two other approaches: AdaGrad [19], which works well with sparse gradients, and RMSProp [20], which works well in online and nonstationary settings.

Very recently, an improvement to Adam, AdaBound [24], have been proposed. It bounds the individualized adaptive learning rate from the upper and the lower, so that an extremely large or small learning rate cannot occur. Additionally, the bounds become tighter as the number of iterations increases, which forces the learning rate to approach a constant (as in the stochastic GD). AdaBound has demonstrated faster convergence and better generalization than Adam in [24], so it is adopted in this article.

The pseudocode of AdaBound can be found in Algorithm 1. By substituting L from (7) into it, we can use the bounded individualized adaptive learning rates for different elements of $\boldsymbol{\theta}$, which may result in better training and generalization performance than using a fixed learning rate. The lower and upper bound functions used in this article are similar to those in [24]

$$l(k) = 0.01 - \frac{0.01}{(1 - \beta_2)k + 1} \tag{12}$$

[4]The ANFIS representation of a TSK fuzzy system is used here. For details, please refer to Section III-E

TABLE I
SUMMARY OF THE TEN REGRESSION DATASETS

Dataset	Source	No. of examples	No. of raw features	No. of numerical features	No. of used features	No. of TSK model parameters
PM10[1]	StatLib	500	7	7	5	212
NO2[1]	StatLib	500	7	7	5	212
Housing[2]	UCI	506	13	13	5	212
Concrete[3]	UCI	1,030	8	8	5	212
Airfoil[4]	UCI	1,503	5	5	5	212
Wine-Red[5]	UCI	1,599	11	11	5	212
Abalone[6]	UCI	4,177	8	7	5	212
Wine-White[5]	UCI	4,898	11	11	5	212
PowerPlant[7]	UCI	9,568	4	4	4	96
Protein[8]	UCI	45,730	9	9	5	212

[1]http://lib.stat.cmu.edu/datasets/
[2]https://archive.ics.uci.edu/ml/machine-learning-databases/housing/
[3]https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
[4]https://archive.ics.uci.edu/ml/datasets/Airfoil+Self-Noise
[5]https://archive.ics.uci.edu/ml/datasets/Wine+Quality
[6]https://archive.ics.uci.edu/ml/datasets/Abalone
[7]https://archive.ics.uci.edu/ml/datasets/Combined+Cycle+Power+Plant
[8]https://archive.ics.uci.edu/ml/datasets/Physicochemical+Properties+of+Protein+Tertiary+Structure

$$u(k) = 0.01 + \frac{0.01}{(1-\beta_2)k}. \tag{13}$$

When $k = 0$, the bound is $[0, +\infty)$. When k approaches $+\infty$, the bound approaches [0.01, 0.01].

III. EXPERIMENTS

This section presents experimental results to demonstrate the performance of our proposed MBGD-RDA.

A. Datasets

Ten regression datasets from the CMU StatLib Datasets Archive[5] and the UCI Machine Learning Repository[6] were used in our experiments. Their summary is given in Table I. Their sizes range from small to large.

Nine of the ten datasets have only numerical features. Abalone has a categorical feature (sex: male, female, and infant), which was ignored in our experiments.[7] Each numerical feature was z-normalized to have zero mean and unit variance, and the output mean was also subtracted. Because fuzzy systems have difficulty dealing with high-dimensional data, we constrained the maximum input dimensionality to be five: if a dataset had more than five features, then principal component analysis was used to reduce them to five.

The TSK fuzzy systems had $M_m = 2$ Gaussian MFs in each input domain. For M inputs, the TSK fuzzy system has $2MM_m + (M+1)\cdot(M_m)^M$ parameters.

[5][Online]. Available: http://lib.stat.cmu.edu/datasets/
[6][Online]. Available: http://archive.ics.uci.edu/ml/index.php
[7]We also tried to convert the categorical feature into numerical ones using one-hot coding and use them together with the other seven numerical features. However, the root-mean-square errors (RMSEs) were worse than simply ignoring it.

TABLE II
PARAMETERS OF THE SEVEN ALGORITHMS USED IN THE EXPERIMENTS

Algorithm	Parameters
RR	$\lambda = 0.05$
MBGD	$M_m = 2$, $N_{bs} = 64$, $K = 500$, $\alpha = 0.01$
MBGD-R	$M_m = 2$, $N_{bs} = 64$, $K = 500$, $\alpha = 0.01$, $\lambda = 0.05$
MBGD-D	$M_m = 2$, $N_{bs} = 64$, $K = 500$, $\alpha = 0.01$, $P = 0.5$
MBGD-RD	$M_m = 2$, $N_{bs} = 64$, $K = 500$, $\alpha = 0.01$, $\lambda = 0.05$, $P = 0.5$
MBGD-A	$M_m = 2$, $N_{bs} = 64$, $K = 500$, $\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
MBGD-RDA	$M_m = 2$, $N_{bs} = 64$, $K = 500$, $\alpha = 0.01$, $\lambda = 0.05$, $P = 0.5$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

The definitions of the parameters can be found in nomenclature.

B. Algorithms

We compared the performances of the following seven algorithms.[8]

1) *Ridge regression* [35], with ridge coefficient $\lambda = 0.05$. This algorithm is denoted as RR in the sequel.
2) *MBGD*, which is a minibatch version of the batch GD algorithm introduced in [10]. The batch size N_{bs} was 64, the initial learning rate α was 0.01, and the adaptive learning rate adjustment rule in [10] was implemented: α was multiplied by 1.1 if the loss function was reduced in four successive iterations and by 0.9 if the loss function had two consecutive combinations of an increase followed by a decrease. This algorithm is denoted as MBGD in the sequel.
3) *MBGD with regularization*, which was essentially identical to MBGD, except that the loss function had an ℓ_2 regularization term on the consequent parameters [see (7)]. $\lambda = 0.05$ was used. This algorithm is denoted as MBGD-R in the sequel.
4) *MBGD with DropRule*, which was essentially identical to MBGD, except that DropRule with $P = 0.5$ was also used in the training, i.e., for each training example, we randomly set the firing level of 50% rules to zero. This algorithm is denoted as MBGD-D in the sequel.
5) *MBGD with regularization and DropRule*, which integrated MBGD-R and MBGD-D. This algorithm is denoted as MBGD-RD in the sequel.
6) *MBGD with AdaBound*, which was essentially identical to MBGD, except that the learning rates were adjusted by AdaBound. $\alpha = 0.01$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ were used. This algorithm is denoted as MBGD-A in the sequel.
7) *MBGD with regularization, DropRule, and AdaBound*, which combined MBGD-R, MBGD-D, and MBGD-A. This algorithm is denoted as MBGD-RDA in the sequel.

For clarity, the parameters of these seven algorithms are also summarized in Table II.

For each dataset, we randomly selected 70% examples for training, and the remaining 30% for test. RR was trained in

[8]We also tried to use support vector regression [34] as a baseline regression model; however, it was too time consuming to train on big datasets. Therefore, we abandoned it.

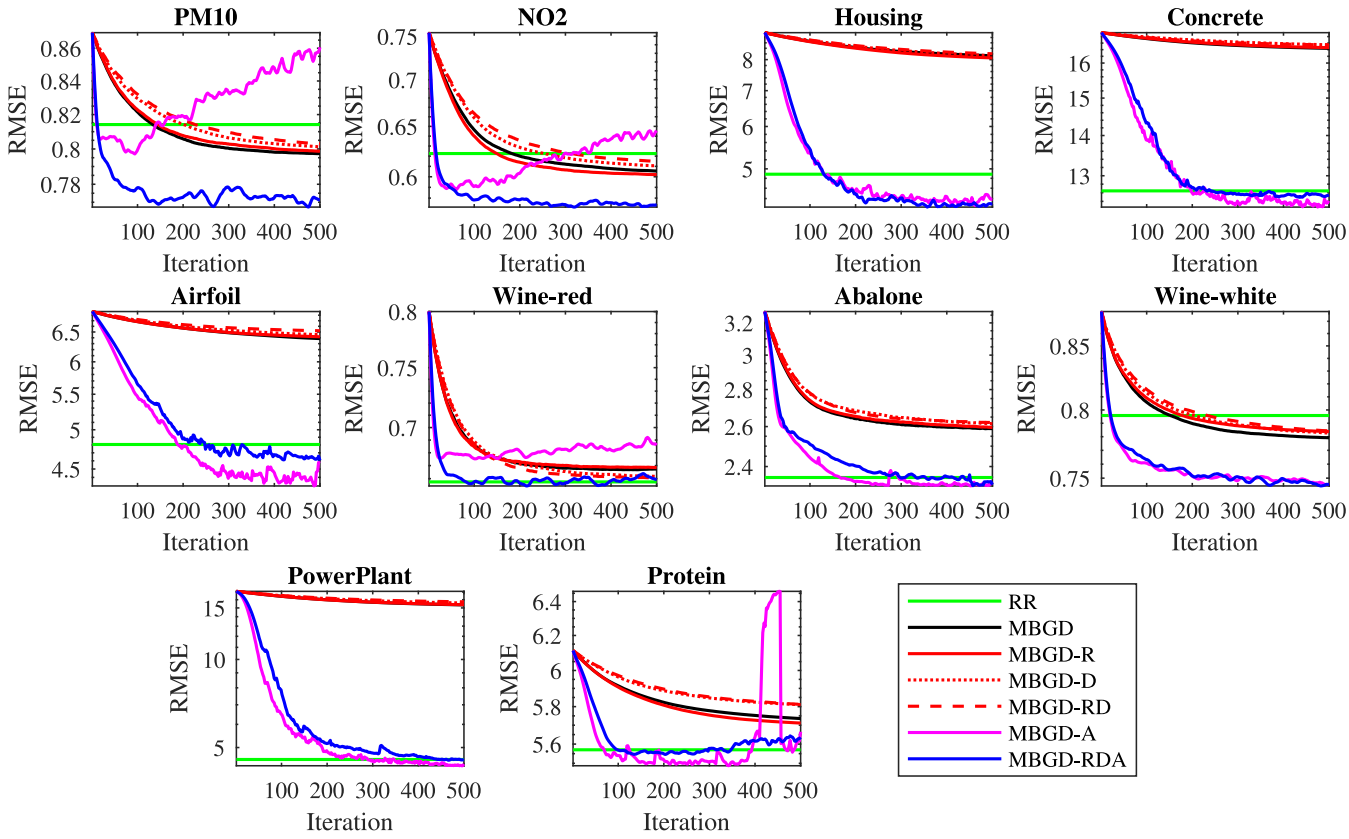

Fig. 2. Average test RMSEs of the seven algorithms on the ten datasets.

one single pass on all training examples, and then its RMSE on the test examples was computed. The other six MBGD-based algorithms were iterative. The TSK fuzzy systems had two Gaussian MFs in each input domain. Their centers were initialized at the minimum and maximum of the input domain, and their standard deviations were initialized to the standard deviation of the corresponding input. All rule consequent parameters were initialized to zero. The maximum number of iterations was 500. After each training iteration, we recorded the corresponding test RMSE of each algorithm. Because there was randomness involved (e.g., the training/test data partition and the selection of the minibatches), each algorithm was repeated ten times on each dataset, and the average test results are reported next.

C. Experimental Results

The average test RMSEs of the seven algorithms are shown in Fig. 2. We can observe the following.

1) `MBGD-R`, `MBGD-D`, and `MBGD-RD` had comparable performance with `MBGD`. All of them were worse than the simple `RR` on seven out of the ten datasets, suggesting that a model with much more parameters and nonlinearity does not necessarily outperform a simple linear regression model, if not properly trained.
2) `MBGD-RDA` and `MBGD-A` performed the best among the seven algorithms. On nine out of the ten datasets (except Wine-Red), `MBGD-A`'s best test RMSEs were smaller than by `RR`. On all ten datasets, `MBGD-RDA`'s best test RMSEs were smaller than by `RR`. `MBGD-RDA` and `MBGD-A` also converged much faster than `MBGD`, `MBGD-R`, `MBGD-D`, and `MBGD-RD`. As the final TSK fuzzy systems trained from the six MBGD-based algorithms had the same structure and the same number of parameters, these results suggest that AdaBound was indeed very effective in updating the learning rates, which in turn helped obtain better learning performances.
3) Although regularization alone (`MBGD-R`), DropRule alone (`MBGD-D`), and the combination of regularization and DropRule (`MBGD-RD`) did not result in much performance improvement (i.e., `MBGD-R`, `MBGD-D`, `MBGD-RD`, and `MBGD` had similar performances), `MBGD-RDA` outperformed `MBGD-A` on three out of the ten datasets, and they had comparable performances on many other datasets. These results suggest that using an effective learning rate updating scheme such as AdaBound can help unleash the power of regularization and DropRule, and hence achieve better learning performance.

To better visualize the performance differences among the six MBGD-based algorithms, we plot in Fig. 3 the percentage improvements of `MBGD-R`, `MBGD-D`, `MBGD-RD`, `MBGD-A`, and `MBGD-RDA` over `MBGD`: in each iteration, we treat the test RMSE of `MBGD` as one, and compute the relative percentage improvements of the test RMSEs of the other five MBGD-based algorithms over it. For example, let $\text{RMSE}_{\text{GD}}(k)$ and

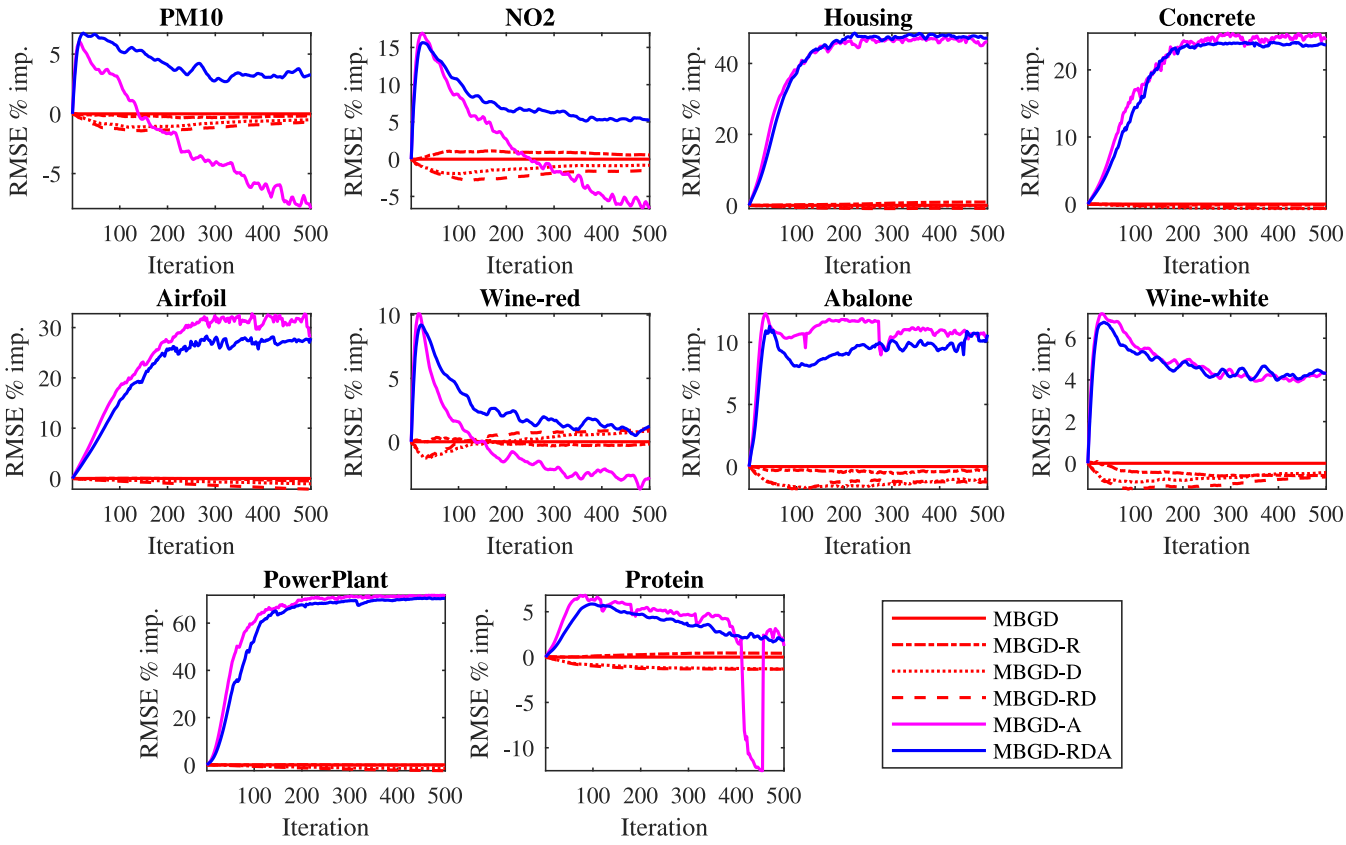

Fig. 3. Percentage improvements of the test RMSEs of MBGD−R, MBGD−D, MBGD−RD, MBGD−A, and MBGD−RDA over MBGD.

$\text{RMSE}_{\text{MBGD-RDA}}(k)$ be the test RMSEs of MBGD and MBGD−RDA at iteration k, respectively. Then, the percentage improvement of the test RMSE of MBGD−RDA over MBGD at iteration k is

$$p(k) = 100 \times \frac{\text{RMSE}_{\text{GD}}(k) - \text{RMSE}_{\text{MBGD-RDA}}(k)}{\text{RMSE}_{\text{GD}}(k)}. \quad (14)$$

Fig. 3 confirmed the observations made from Fig. 2. Particularly, MBGD−RDA and MBGD−A converged much faster and to smaller values than MBGD, MBGD−R, MBGD−D, and MBGD−RD; the best test RMSEs of MBGD−RDA and MBGD−A were also much smaller than those of MBGD, MBGD−R, MBGD−D, and MBGD−RD. Among the five enhancements to MBGD, only MBGD−RDA consistently outperformed MBGD. In other words, although MBGD−RDA may not always outperform MBGD−A, its performance was more stable than MBGD−A; so, it should be preferred over MBGD−A in practice.

The time taken to finish 500 training iterations for the MBGD-based algorithms on the ten datasets is shown in Table III. The platform was a desktop computer running MATLAB 2018a and Windows 10 Enterprise 64x, with Intel Core i7-8700 K CPU @ 3.70 GHz, 16 GB memory, and 512 GB solid state drive. The CPU has 12 cores, but each algorithm used only one core. Not surprisingly, RR was much faster than others, because it has a closed-form solution, and no iteration was needed. Among the six MBGD-based algorithms, MBGD−RDA was the fastest. One reason is that DropRule reduced the number of parameters to be adjusted in each iteration.

TABLE III
COMPUTATIONAL COST (SECONDS) OF DIFFERENT ALGORITHMS ON THE TEN REGRESSION DATASETS

Dataset	RR	MBGD	MBGD −R	MBGD −D	MBGD −RD	MBGD −A	MBGD −RDA
PM10	0.003	21.115	21.027	19.194	19.388	20.746	15.394
NO2	0.003	21.619	21.273	19.627	19.620	21.063	15.971
Housing	0.003	21.304	21.064	19.392	19.357	20.799	15.782
Concrete	0.003	29.891	30.143	27.784	27.943	30.142	25.468
Airfoil	0.005	35.813	35.928	33.800	34.532	36.608	31.027
Wine-Red	0.003	36.704	37.016	35.606	35.594	36.793	32.070
Abalone	0.003	38.780	39.046	38.406	38.861	38.909	36.921
Wine-White	0.003	74.429	75.992	73.091	74.525	74.844	70.531
PowerPlant	0.005	68.954	66.656	68.187	65.763	66.849	64.811
Protein	0.008	474.995	445.170	469.929	433.486	461.251	429.679

D. Parameter Sensitivity

It is also important to study the sensitivity of MBGD−RDA to its hyperparameters. Algorithm 1 shows that MBGD−RDA has the following hyperparameters:

1) M_m, the number of Gaussian MFs in the mth input domain;
2) K, the maximum number of training iterations;
3) $N_{bs} \in [1, N]$, the minibatch size;
4) $P \in (0, 1)$, the DropRule rate;
5) α, the initial learning rate (step size);
6) λ, the ℓ_2 regularization coefficient;
7) $\beta_1, \beta_2 \in [0, 1)$, exponential decay rates for the moment estimates in AdaBound;
8) ϵ, a small positive number in AdaBound;

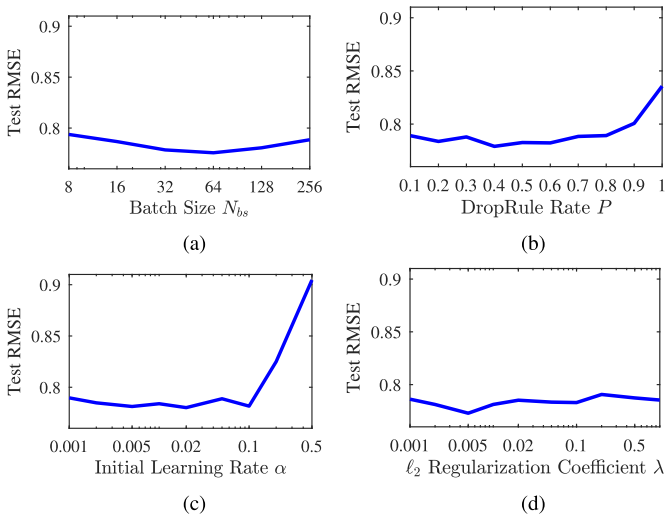

Fig. 4. Test RMSEs of MBGD-RDA w.r.t. different hyperparameters on the PM10 dataset. (a) Different batch size N_{bs}. (b) Different DropRule rate P. (c) Different initial learning rate α. (d) Different ℓ_2 regularization coefficient λ. In each subfigure, except for the hyperparameter under consideration, the values for other parameters are: $M_m = 2$, $K = 500$, $N_{bs} = 64$, $P = 0.5$, $\alpha = 0.01$, $\lambda = 0.05$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and $l(k)$ and $u(k)$ are defined in (12) and (13), respectively.

9) $l(k)$ and $u(k)$, the lower and upper bound functions in AdaBound

Among them, M_m is a parameter for all TSK fuzzy systems, not specific to MBGD-RDA; K can be determined by early stopping on a validation dataset; and β_1, β_2, ϵ, $l(k)$ and $u(k)$ are AdaBound parameters, whose default values are usually used. Therefore, we only studied the sensitivity of MBGD-RDA to N_{bs}, P, α, and λ, which are unique to MBGD-RDA.

The results, in terms of the test RMSEs, on the PM10 dataset are shown in Fig. 4, where each experiment was repeated 100 times and the average test RMSEs are shown. In each subfigure, except for the hyperparameter under consideration, the values for other parameters were: $M_m = 2$, $K = 500$, $N_{bs} = 64$, $P = 0.5, \alpha = 0.01, \lambda = 0.05, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$, and $l(k)$ and $u(k)$ are defined in (12) and (13), respectively. Clearly, MBGD-RDA is stable with respect to each of the four hyperparameters in a wide range, which is desirable.

E. Comparison With ANFIS

ANFIS [10] is an efficient algorithm for training TSK fuzzy systems on small datasets. This section compares the performance of MBGD-RDA with ANFIS on the first six smaller datasets.

The ANFIS structure of a two-input one-output TSK fuzzy system is shown in Fig. 5. It has the following five layers.

Layer 1: The membership grade of x_m on $X_{r,m}$ is computed, by (2).

Layer 2: The firing level of each rule Rule_r is computed, by (3).

Layer 3: The normalized firing levels of the rules are computed, by (5).

Layer 4: Each normalized firing level is multiplied by its corresponding rule consequent.

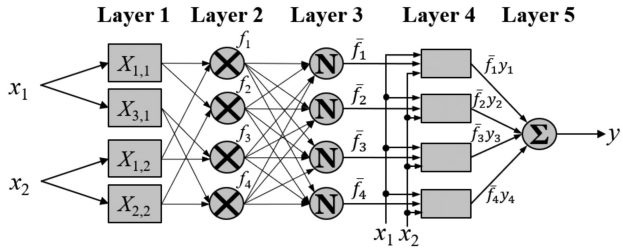

Fig. 5. TSK fuzzy system represented as a five-layer ANFIS. Note that $X_{1,1} = X_{2,1}$, $X_{3,1} = X_{4,1}$, $X_{1,2} = X_{3,2}$, and $X_{2,2} = X_{4,2}$ are used.

Layer 5: The output is computed, by (6).

All parameters of the ANFIS, i.e., the shapes of the MFs and the rule consequents, can be trained by GD [10]; or, to speed up the training, the antecedent parameters can be tuned by GD, and the consequent parameters by LSE [10].

In the experiment, we used the *anfis* function in MATLAB 2018b, which does not allow us to specify a batch size, but to use all available training examples in each iteration. For fair comparison, in MBGD-RDA, we also set the batch size to the number of training examples. *anfis* in MATLAB has two optimization options: first, batch GD for both antecedent and consequent parameters (denoted as ANFIS-GD in the sequel); and second, batch GD for antecedent parameters and LSE for consequent parameters (denoted as ANFIS-GD-LSE in the sequel). We compared MBGD-RDA with both options.

The training and test RMSEs, averaged over ten runs, are shown in Fig. 6(a). MBGD-RDA always converged much faster than ANFIS-GD, and its best test RMSE was also always smaller. Additionally, it should be emphasized that MBGD-RDA can be used also for big data, whereas ANFIS-GD cannot.

Interestingly, although ANFIS-GD-LSE always achieved the smallest training RMSE, its test RMSE was almost always the largest and had large oscillations. This suggests that ANFIS-GD-LSE had significant overfitting. If we could reduce this overfitting, e.g., through regularization, then ANFIS-GD-LSE could be a very effective TSK fuzzy system training algorithm for small datasets. This is one of our future research directions.

Fig. 6(b) shows the learning rates of ANFIS-GD, ANFIS-GD-LSE, and MBGD-RDA. For the first two ANFIS-based approaches, all model parameters shared the same learning rate. However, in MBGD-RDA, different model parameters had different learning rates, and we show the average learning rate across all model parameters on each dataset. The learning rates in ANFIS-GD and ANFIS-GD-LSE first gradually increased and then decreased. Interestingly, the learning rate of ANFIS-GD-LSE was almost always smaller than that of ANFIS-GD when the number of iterations was large. The learning rate of MBGD-RDA was always very high at the beginning, and then rapidly decreased. The initial high learning rate helped MBGD-RDA in achieving rapid convergence.

F. Comparison With DropMF and DropMembership

In addition to DropRule, there could be other DropOut approaches in training a TSK fuzzy system, e.g., as follows.

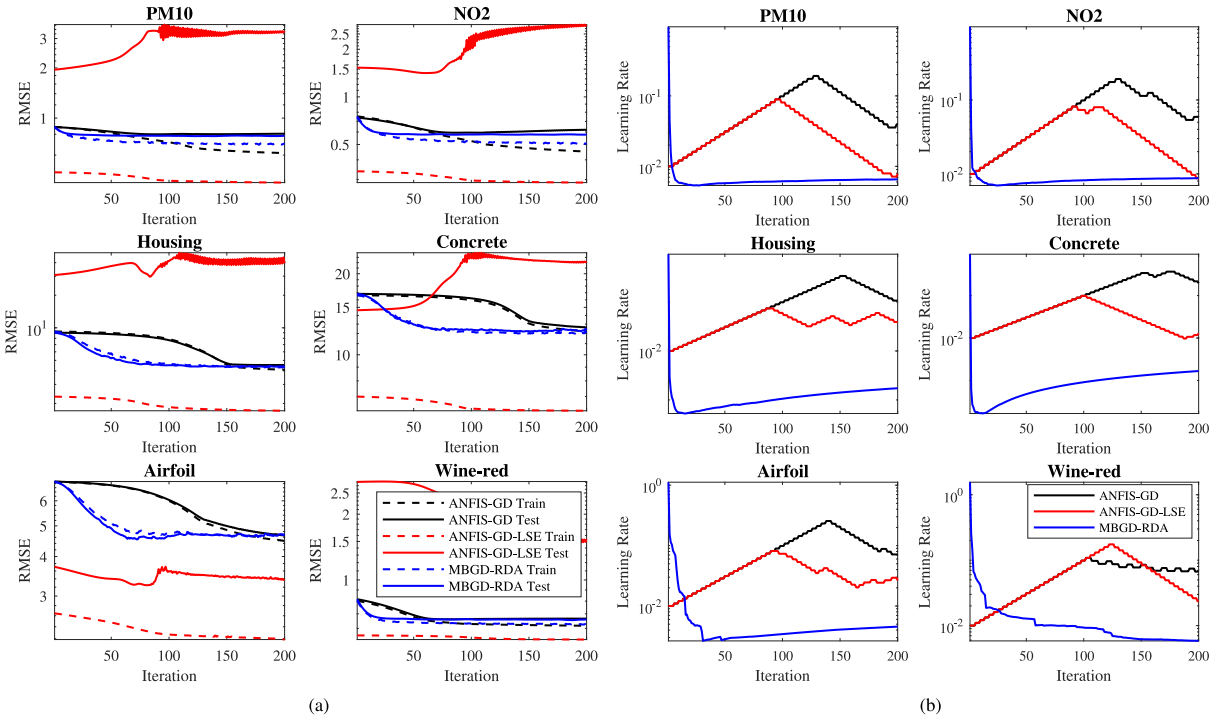

Fig. 6. (a) Performance comparison of MBGD-RDA with ANFIS-GD and ANFIS-GD-LSE in batch GD. We use logarithmic scale on the vertical axis to make the curves more distinguishable. (b) Learning rates of ANFIS-GD and ANFIS-GD-LSE, and the average learning rate (across all model parameters) of MBGD-RDA, in batch GD.

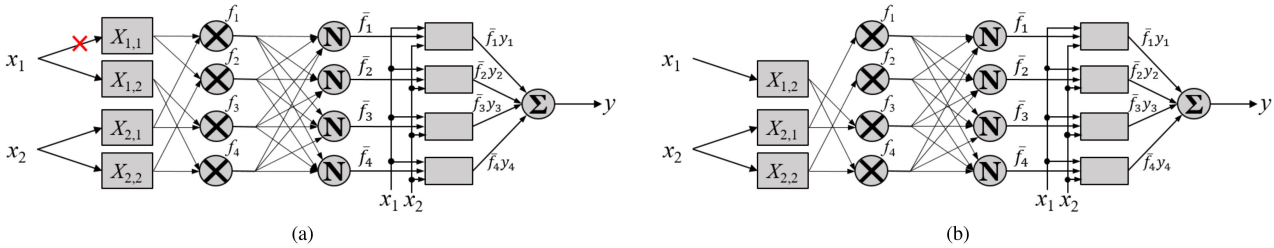

Fig. 7. DropMF, where $X_{m,i}$ is the ith MF in the mth input domain. (a) Red cross indicates that MF $X_{1,1}$ for x_1 will be dropped. (b) Equivalent fuzzy system after dropping $X_{1,1}$.

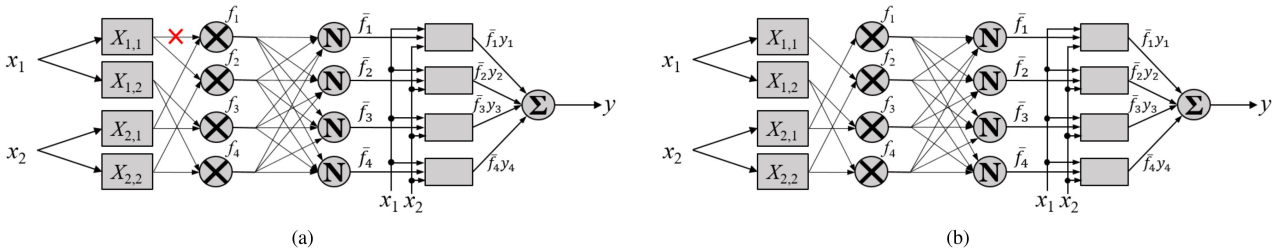

Fig. 8. DropMembership, where $X_{m,i}$ is the ith MF in the mth input domain. (a) Red cross indicates that membership $\mu_{X_{1,1}}(x_1)$ in the first rule will be dropped. (b) Equivalent fuzzy system after dropping $\mu_{X_{1,1}}(x_1)$ in the first rule.

1) *DropMF*, in which each input MF is dropped with a probability $1-P$, as illustrated in Fig. 7. Dropping an MF is equivalent to setting the firing level of that MF to 1 (instead of 0). Comparing DropMF in Fig. 7(b) and DropRule in Fig. 1(b), we can observe that each DropRule operation reduces the number of used rules by 1. On the contrary, DropMF does not reduce the number of used rules; instead, it reduces the number of antecedents in multiple rules by 1.
2) *DropMembership*, in which the membership of an input in each MF is dropped with a probability $1-P$, as illustrated in Fig. 8. Dropping a membership is equivalent to setting that membership to 1 (instead of 0). Comparing DropMembership in Fig. 8(b) and DropMF in Fig. 7(b),

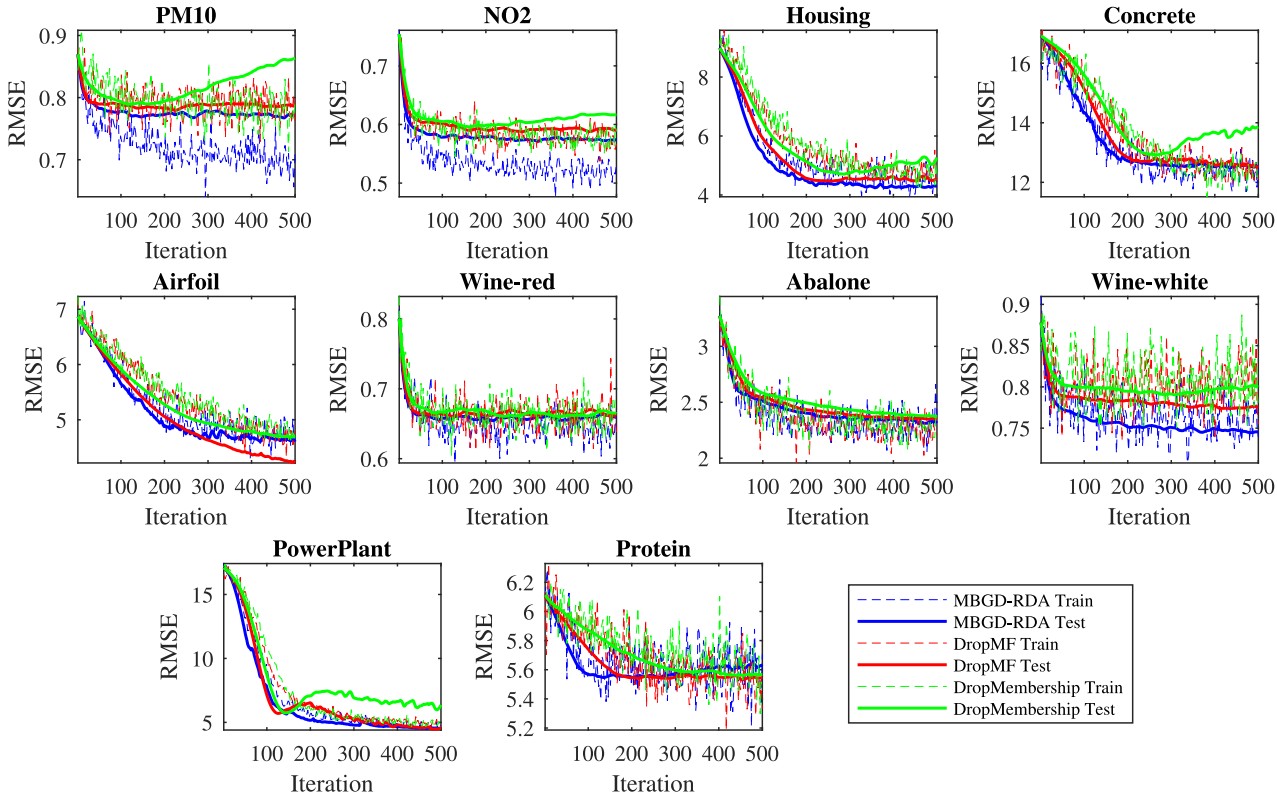

Fig. 9. Average training and test RMSEs of `MBGD-RDA`, `DropMF` (replacing DropRule in `MBGD-RDA` by DropMF), and `DropMembership` (replacing DropRule in `MBGD-RDA` by DropMembership) on the ten datasets.

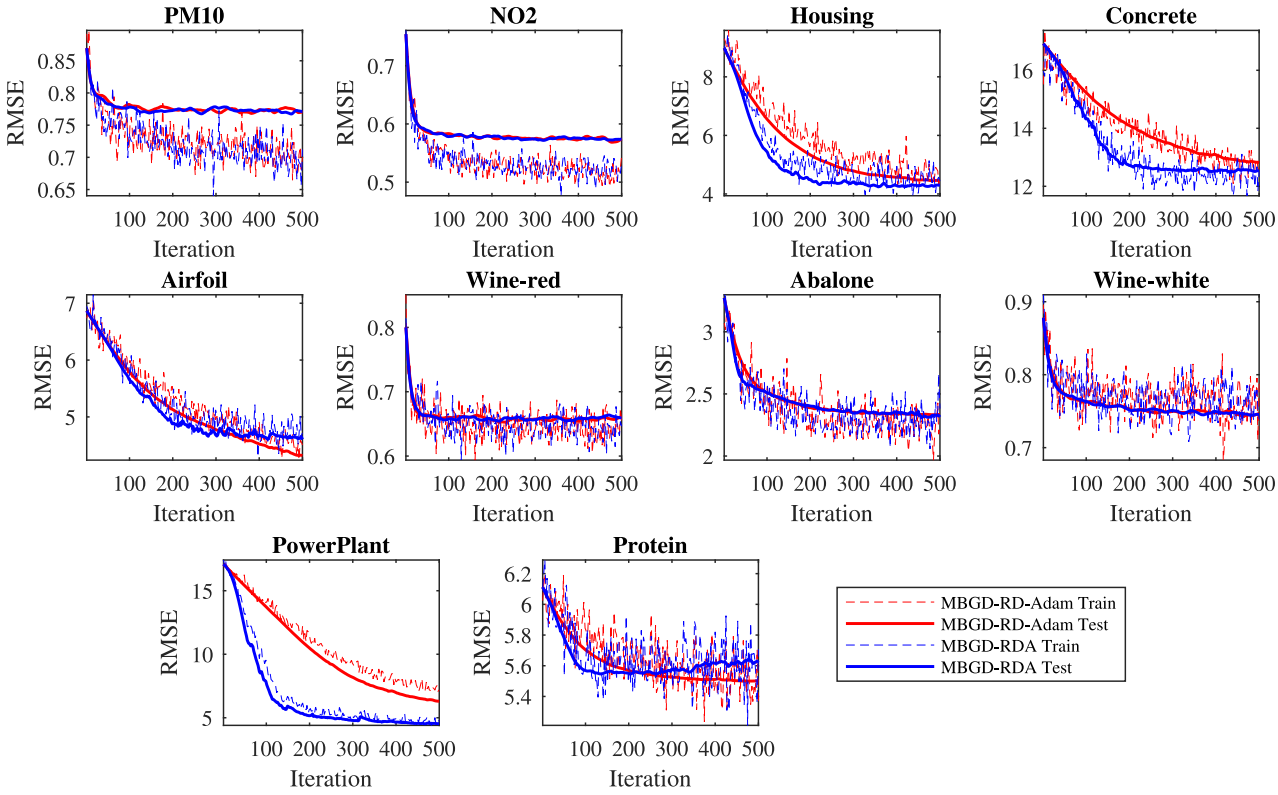

Fig. 10. Average training and test RMSEs of `MBGD-RDA` and `MBGD-RD-Adam` on the ten datasets.

we can observe that DropMembership has a smaller impact on the firing levels of the rules than DropMF. For example, in Fig. 7(b), both f_1 and f_2 are impacted by DropMF, whereas in Fig. 8(b), only f_1 is impacted by DropMembership.

Next, we compare the performances of DropMF, DropMembership with DropRule, by replacing DropRule in `MBGD-RDA` by DropMF and DropMembership. The training and test RMSEs, averaged over ten runs, are shown in Fig. 9. Generally, DropRule performed the best, and DropMembership the worst. Comparing Figs. 1(b), 7(b), and 8(b), we can observe that DropRule introduces the maximum change to the TSK fuzzy system structure, and DropMembership the smallest. This suggests that a dropout operation that introduces more changes to the TSK fuzzy system may be more beneficial to the training and test performances.

G. Comparison With Adam

We also compared the performances of AdaBound with Adam. The learning algorithm, `MBGD-RD-Adam`, was identical to `MBGD-RDA`, except that AdaBound was replaced by Adam, by setting $l(k) = 0$ and $u(k) = +\infty$ in Algorithm 1.

The training and test RMSEs, averaged over ten runs, are shown in Fig. 10. `MBGD-RDA` converged faster than, or equally fast with, `MBGD-RD-Adam`, and had smaller or comparable best test RMSEs as `MBGD-RD-Adam` on most datasets. So, it is generally safe to choose AdaBound over Adam.

IV. Conclusion and Future Research

TSK fuzzy systems are very useful machine learning models for regression problems. However, to our knowledge, there has not existed an efficient and effective training algorithm that enables them to deal with big data. Inspired by the connections between TSK fuzzy systems and neural networks, this article extended three powerful optimization techniques for neural networks, e.g., MBGD, regularization, and AdaBound, to TSK fuzzy systems, and also proposed three novel techniques (DropRule, DropMF, and DropMembership) specifically for training TSK fuzzy systems. Our final algorithm, MBGD-RDA, which integrates MBGD, regularization, AdaBound, and DropRule, can achieve fast convergence in training TSK fuzzy systems, and also superior generalization performance in testing. It can be used for training TSK fuzzy systems on datasets of any size; however, it is particularly useful for big datasets, for which currently no other efficient training algorithms exist. We expect that our algorithm will help promote the applications of TSK fuzzy systems, particularly to big data.

Finally, we need to point out that we have not considered various uncertainties in the data, e.g., missing values, wrong values, noise, and outliers, which frequently happen in real-world applications. Some techniques, e.g., rough sets [36], could be integrated with fuzzy sets to deal with them; or the type-1 TSK fuzzy systems used in this article could also be extended to interval or general type-2 fuzzy systems [1], [37] to cope with more uncertainties. These are some of our future research directions.

References

[1] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions*, 2nd ed. New York, NY, USA: Springer, 2017.

[2] A. Nguyen, T. Taniguchi, L. Eciolaza, V. Campos, R. Palhares, and M. Sugeno, "Fuzzy control systems: Past, present and future," *IEEE Comput. Intell. Mag.*, vol. 14, no. 1, pp. 56–68, Feb. 2019.

[3] I. Couso, C. Borgelt, E. Hullermeier, and R. Kruse, "Fuzzy sets in data analysis: From statistical foundations to machine learning," *IEEE Comput. Intell. Mag.*, vol. 14, no. 1, pp. 31–44, Feb. 2019.

[4] X. Gu, P. Angelov, and H.-J. Rong, "Local optimality of self-organising neuro-fuzzy inference systems," *Inf. Sci.*, vol. 503, pp. 351–380, 2019.

[5] X. Gu and P. Angelov, "Self-boosting first-order autonomous learning neuro-fuzzy systems," *Appl. Soft Comput.*, vol. 77, pp. 118–134, 2019.

[6] D. Wu and W. W. Tan, "Genetic learning and performance evaluation of type-2 fuzzy logic controllers," *Eng. Appl. Artif. Intell.*, vol. 19, no. 8, pp. 829–841, 2006.

[7] L.-X. Wang and J. M. Mendel, "Back-propagation of fuzzy systems as nonlinear dynamic system identifiers," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Diego, CA, USA, 1992, pp. 1409–1418.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[9] C.-T. Lin and C. S. G. Lee, *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[10] J. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 665–685, May/Jun. 1993.

[11] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Netw. Appl.*, vol. 19, no. 2, pp. 171–209, 2014.

[12] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[13] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, "Big data: The management revolution," *Harvard Bus. Rev.*, vol. 90, no. 10, pp. 60–68, 2012.

[14] I. Jolliffe, *Principal Component Analysis*. Wiley Online Library, 2002.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Boston, MA, USA: MIT Press, 2016.

[16] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade*. New York, NY, USA: Springer, 2012, pp. 437–478.

[17] S. Nakasima-Lpez, J. R. Castro, M. A. Sanchez, O. Mendoza, and A. Rodrguez-Daz, "An approach on the implementation of full batch, online and mini-batch learning on a Mamdani based neuro-fuzzy system with center-of-sets defuzzification: Analysis and evaluation about its functionality, performance, and behavior," *PLoS One*, vol. 14, no. 9, 2019, Art. no. e0221369.

[18] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control Optim.*, vol. 30, no. 4, pp. 838–855, 1992.

[19] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.

[20] T. Tieleman and G. Hinton, "Lecture 6.5–RMSProp: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," presented at the 3rd Int. Conf. Learn. Representations, San Diego, CA, USA, May 2015.

[22] S. Matsumura and T. Nakashima, "Incremental learning for SIRMs fuzzy systems by Adam method," in *Proc. Joint 17th World Congr. Int. Fuzzy Syst. Assoc., 9th Int. Conf. Soft Comput. Intell. Syst.*, Otsu, Japan, Jun. 2017, pp. 1–4.

[23] N. Yubazaki, J. Yi, M. Otani, and K. Hirota, "SIRMs dynamically connected fuzzy inference model and its applications," in *Proc. 7th IFSA World Congr.*, Prague, Czech Republic, Jun. 1997, pp. 410–415.

[24] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," presented at the Int. Conf. Learning Representations, New Orleans, LA, USA, May 2019.

[25] T. A. Johansen, "Robust identification of Takagi–Sugeno–Kang fuzzy models using regularization," in *Proc. 5th IEEE Int. Conf. Fuzzy Syst.*, New Orleans, LA, USA, Sep. 1996, pp. 180–193.

[26] Y. Jin, "Fuzzy modeling of high-dimensional systems: Complexity reduction and interpretability improvement," *IEEE Trans. Fuzzy Syst.*, vol. 8, no. 2, pp. 212–221, Apr. 2000.

[27] E. Lughofer and S. Kindermann, "SparseFIS: Data-driven learning of fuzzy systems with sparsity constraints," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 2, pp. 396–411, Apr. 2010.

[28] M. Luo, F. Sun, and H. Liu, "Dynamic T-S fuzzy systems identification based on sparse regularization," *Asian J. Control*, vol. 17, no. 1, pp. 274–283, 2015.
[29] E. Lughofer and S. Kindermann, "Improving the robustness of data-driven fuzzy systems with regularization," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Hong Kong, China, Jul. 2008, pp. 703–709.
[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[31] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using DropConnect," in *Proc. Int. Conf. Mach. Learn.*, Atlanta, GA, USA, Jun. 2013, pp. 1058–1066.
[32] D. Wu, C.-T. Lin, J. Huang, and Z. Zeng, "On the functional equivalence of TSK fuzzy systems to neural networks, mixture of experts, CART, and stacking ensemble regression," *IEEE Trans. Fuzzy Syst.*, to be published, doi: 10.1109/TFUZZ.2019.2941697.
[33] D. Wu, "Twelve considerations in choosing between Gaussian and trapezoidal membership functions in interval type-2 fuzzy logic controllers," in *Proc. IEEE World Congr. Comput. Intell.*, Brisbane, QLD, Australia, Jun. 2012.
[34] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statist. Comput.*, vol. 14, no. 3, pp. 199–222, 2004.
[35] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
[36] Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning About Data*, vol. 9. New York, NY, USA: Springer, 2012.
[37] D. Wu, "On the fundamental differences between interval type-2 and type-1 fuzzy logic controllers," *IEEE Trans. Fuzzy Syst.*, vol. 20, no. 5, pp. 832–848, Oct. 2012.

Dongrui Wu (S'05–M'09–SM'14) received the B.E. degree in automatic control from the University of Science and Technology of China, Hefei, China, in 2003, the M.E. degree in electrical engineering from the National University of Singapore, Singapore, in 2005, and the Ph.D. degree in electrical engineering from the University of Southern California, Los Angeles, CA, USA, in 2009.

He is currently a Professor with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China, and the Deputy Director of the Key Laboratory of Image Processing and Intelligent Control of Ministry of Education, Wuhan. He has authored or coauthored more than 130 publications, including a book entitled *Perceptual Computing* (Wiley-IEEE Press, 2010) and more than 40 IEEE Transactions papers. His research interests include affective computing, brain–computer interface, computational intelligence, and machine learning.

Dr. Wu was an Associate Editor for the IEEE TRANSACTIONS ON FUZZY SYSTEMS (2011–2018). He has been an Associate Editor for the IEEE TRANSACTIONS ON HUMAN-MACHINE SYSTEMS since 2014, IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE since 2017, and IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING since 2019. He was the recipient of the IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award in 2012, the IEEE Transactions on Fuzzy Systems Outstanding Paper Award in 2014, the NAFIPS Early Career Award in 2014, the IEEE Systems, Man and Cybernetics (SMC) Society Early Career Award in 2017, and the IEEE SMC Society Best Associate Editor Award in 2018. He was also a finalist of the USERN Prize and another three best paper awards.

Ye Yuan (M'13) received the B.Eng. degree (Valedictorian) in automation from Shanghai Jiao Tong University, Shanghai, China, in 2008, and the M.Phil. and Ph.D. degrees in engineering from the University of Cambridge, Cambridge, U.K., in 2009 and 2012, respectively.

Since 2016, he has been a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. Before that, he was a Postdoctoral Researcher with the University of California, Berkeley, and a Junior Research Fellow with Darwin College, University of Cambridge. His research interests include system identification and control with applications to cyber-physical systems.

Dr. Yuan is a recipient of the China National Recruitment Program of 1000 Talented Young Scholars, the Dorothy Hodgkin Postgraduate Awards, Microsoft Research Ph.D. Scholarship, and Best of the Best Paper Award at the IEEE PES General Meeting. He was also a best paper finalist at the IEEE International Conference on Information and Automation.

Jian Huang (M'07–SM'17) received the B.E., M.Eng., and Ph.D. degrees in automatic control from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1997, 2000, and 2005, respectively.

From 2006 to 2008, he was a Postdoctoral Researcher with the Department of Micro-Nano System Engineering and the Department of Mechano-Informatics and Systems, Nagoya University, Nagoya, Japan. In 2015, he was a Research Fellow with Nagoya University supported by JSPS invitation fellowship. He is currently a Full Professor with the School of Artificial Intelligence and Automation, HUST. He is also a Guest Professor with Nagoya University of Japan and University Paris-Est Creteil, Creteil, France. He has authored or coauthored more than 190 papers (including 23 IEEE Transactions articles and more than 50 IEEE conference papers). He has 14 authorized patents. His research interests include rehabilitation robots, robotic assembly, networked control systems, and bioinformatics.

Dr. Huang is currently an Associate Editor for the IEEE TRANSACTIONS ON FUZZY SYSTEMS, an Editor for Springer's *ROBOMECH Journal*, and a Technical Editor for *PLoS One*. He was the recipient of the Grand Prize of Science and Technology Award of China General Chamber of Commerce and Golden Medal at Geneva Inventions in 2017.

Yihua Tan (M'11) received the Ph.D. degree in pattern recognition and intelligent systems from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2004.

From 2005 to 2006, he was with the Postdoctoral staff with the Department of Electronics and Information, HUST. Since 2005, he has been with the School of Artificial Intelligence and Automation, HUST, where he is currently an Associate Professor. During 2010–2011, he was a Visiting Scholar with Purdue University, focusing his research on remote sensing image analysis. He has authored more than 70 papers in journals and conferences. His current research interests include digital image/video processing and analysis, object detection and recognition, and machine learning.