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.

$x_n = (x_{n,1},...,x_{n,M})^T$ M-dimensional feature vector of the $n$th training example, $n \in [1,N]$.

$y_n$ Groundtruth output corresponding to $x_n$.

$R$ Number of rules in the TSK fuzzy system.

$X_{r,m}$ MF for the $m$th feature in the $r$th rule, $r \in [1,R]$ and $m \in [1,M]$.

$b_{r,0},...,b_{r,M}$ Consequent parameters of the $r$th rule, $r \in [1,R]$.

$y_r(x_n)$ Output of the $r$th rule for $x_n$, $r \in [1,R]$ 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 $n \in [1,N]$.

$f_r(x_n)$ Firing level of $x_n$ on the $r$th rule, $r \in [1,R]$ and $n \in [1,N]$.

$y(x_n)$ Output of the TSK fuzzy system for $x_n$.

$L$ $\ell_2$-regularized loss function for training the TSK fuzzy system.

$\lambda$ $\ell_2$ 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.

$\alpha$ Initial learning rate in MBGD-based algorithms.

$P$ DropRule rate in MBGD-D, MBGD-RA, and MBGD-RDA.

$\beta_1, \beta_2$ Exponential decay rates for moment estimates in AdaBound.

$\epsilon$ 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 $\{(x_n, y_n)\}_{n=1}^N$, where $x_n \in \mathbb{R}^{M \times 1}$, and would like to train a TSK fuzzy system to model the relationship between $y$ and $x$ so that an accurate prediction can be made for any future unseen $x$.

There are generally following three different strategies for optimizing a TSK fuzzy system in supervised regression.\footnote{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.}

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; ye@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) **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 antecedents 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-Lopez 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 Luhhoefer and Kindermann [29] used \(\ell_2\) 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. Luhhoefer 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
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.

Note that \( \theta_K \) returned from Algorithm 1 is not necessarily the...
optimal one among \( \{ \theta_k \}_{k=1}^K \), i.e., the one that gives the smallest test error. The iteration number corresponding to the optimal \( \theta^* \) 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 \( x = (x_1, \ldots, x_M)^T \in \mathbb{R}^{M \times 1} \), and the TSK fuzzy system has the following \( R \) rules:

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

(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

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

(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:

\[
f_r(x) = \prod_{m=1}^M \mu_{X_{r,m}}(x_m)
\]  

(3)

and the output of the TSK fuzzy system is

\[
y(x) = \frac{\sum_{r=1}^R f_r(x) y_r(x)}{\sum_{r=1}^R f_r(x)}
\]  

(4)

or if we define the normalized firing levels as

\[
\bar{f}_r(x) = \frac{f_r(x)}{\sum_{k=1}^R f_k(x)}, \quad r = 1, \ldots, R
\]  

(5)

then, (4) can be rewritten as

\[
y(x) = \sum_{r=1}^R \bar{f}_r(x) \cdot y_r(x).
\]  

(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 \( \{ x_n, y_n \}_{n=1}^N \), where \( x_n = (x_{n,1}, \ldots, x_{n,M})^T \in \mathbb{R}^{M \times 1} \).

In this article, we use the following \( \ell_2 \)-regularized loss function:

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

(7)

where \( N_{bs} \in [1, N] \), and \( \lambda \geq 0 \) is a regularization parameter. Note that \( b_{r,0} \) is 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.
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 $\theta_k$ be the model parameter vector in the $k$th iteration, and $\partial L / \partial \theta_k$ be the first-order gradients computed from (8)–(10). Then, the update rule is given by

$$\theta_k = \theta_{k-1} - \alpha \frac{\partial L}{\partial \theta_{k-1}}$$  \hspace{1cm} \text{(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.\textsuperscript{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 $\theta$, 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]

$$l(k) = 0.01 - \frac{0.01}{(1 - \beta_2)k + 1} \hspace{1cm} \text{(12)}$$
Ridge regression was trained in MBGD-RD, which was essentially identical to MBGD in the sequel. was 0.01, and the adaptive MBGD-RDA in the MBGD with regularization and DropRule in the \( \beta = 0 \). MBGD with DropRule was also MBGD with AdaBound. \( \alpha \) MBGD with regularization, DropRule, and AdaBound \( \lambda \). This algorithm is denoted as MBGD-RDA. 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 \( N_{bs} = 64 \), the initial learning rate \( \alpha = 0.01 \), and the adaptive learning rate adjustment rule in [10] was implemented: \( \alpha \) 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 \( \ell_2 \) regularization term on the consequent parameters \( \{ \} \). \( \lambda = 0.05 \) 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. \( \alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8} \) 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

### 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 \( 2M M_m + (M + 1) \cdot (M_m)^M \) parameters.

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

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

### Table I

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Source</th>
<th>No. of examples</th>
<th>No. of raw features</th>
<th>No. of numerical features</th>
<th>No. of used features</th>
<th>No. of TSK model parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM1</td>
<td>StatLib</td>
<td>500</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>NO2</td>
<td>StatLib</td>
<td>500</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>Housing</td>
<td>UCI</td>
<td>506</td>
<td>13</td>
<td>13</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>Concrete</td>
<td>UCI</td>
<td>1,030</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>Airfoil</td>
<td>UCI</td>
<td>1,503</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>Wine-Red</td>
<td>UCI</td>
<td>1,599</td>
<td>11</td>
<td>11</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>Abalone</td>
<td>UCI</td>
<td>4,177</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>Wine-White</td>
<td>UCI</td>
<td>4,898</td>
<td>11</td>
<td>11</td>
<td>5</td>
<td>212</td>
</tr>
<tr>
<td>PowerPlant</td>
<td>UCI</td>
<td>9,568</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>Protein</td>
<td>UCI</td>
<td>45,730</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>212</td>
</tr>
</tbody>
</table>


\(^7\) 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{RR} )</td>
<td>( \lambda = 0.05 )</td>
</tr>
<tr>
<td>MBGD</td>
<td>( M_m = 2, N_{bs} = 64, K = 500, \alpha = 0.01 )</td>
</tr>
<tr>
<td>MBGD-R</td>
<td>( M_m = 2, N_{bs} = 64, K = 500, \alpha = 0.01, \lambda = 0.05 )</td>
</tr>
<tr>
<td>MBGD-D</td>
<td>( M_m = 2, N_{bs} = 64, K = 500, \alpha = 0.01, P = 0.5 )</td>
</tr>
<tr>
<td>MBGD-RD</td>
<td>( M_m = 2, N_{bs} = 64, K = 500, \alpha = 0.01, \lambda = 0.05, P = 0.5 )</td>
</tr>
<tr>
<td>MBGD-A</td>
<td>( M_m = 2, N_{bs} = 64, K = 500, \alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8} )</td>
</tr>
<tr>
<td>MBGD-RDA</td>
<td>( M_m = 2, N_{bs} = 64, K = 500, \alpha = 0.01, \lambda = 0.05, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8} )</td>
</tr>
</tbody>
</table>

The definitions of the parameters can be found in nomenclature.

---

\(^8\) 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 $\text{RMSE}_{\text{MBGD}}(k)$ and...
Fig. 3. Percentage improvements of the test RMSEs of MBGD–R, MBGD–D, MBGD–RD, MBGD–A, and MBGD–RDA over MBGD.

\[ p(k) = 100 \times \frac{\text{RMSE}_{\text{GD}}(k) - \text{RMSE}_{\text{MBGD-RDA}}(k)}{\text{RMSE}_{\text{GD}}(k)}. \]  

(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>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PM10</td>
<td>0.003</td>
<td>21.115</td>
<td>21.027</td>
<td>19.194</td>
<td>19.388</td>
<td>20.746</td>
<td>15.394</td>
</tr>
<tr>
<td>Housing</td>
<td>0.003</td>
<td>21.304</td>
<td>21.064</td>
<td>19.392</td>
<td>19.357</td>
<td>20.799</td>
<td>15.782</td>
</tr>
<tr>
<td>Concrete</td>
<td>0.003</td>
<td>29.891</td>
<td>30.143</td>
<td>27.784</td>
<td>27.943</td>
<td>30.142</td>
<td>25.468</td>
</tr>
<tr>
<td>Airfoil</td>
<td>0.005</td>
<td>35.813</td>
<td>35.928</td>
<td>35.800</td>
<td>34.532</td>
<td>36.608</td>
<td>31.027</td>
</tr>
<tr>
<td>Wine-Red</td>
<td>0.003</td>
<td>36.704</td>
<td>37.016</td>
<td>35.606</td>
<td>35.594</td>
<td>36.793</td>
<td>32.070</td>
</tr>
<tr>
<td>Abalone</td>
<td>0.003</td>
<td>38.780</td>
<td>39.046</td>
<td>38.406</td>
<td>38.861</td>
<td>38.909</td>
<td>36.921</td>
</tr>
<tr>
<td>Wine-White</td>
<td>0.003</td>
<td>74.425</td>
<td>75.992</td>
<td>73.091</td>
<td>74.525</td>
<td>74.844</td>
<td>70.531</td>
</tr>
<tr>
<td>PowerPlant</td>
<td>0.005</td>
<td>68.954</td>
<td>66.656</td>
<td>68.187</td>
<td>65.763</td>
<td>66.849</td>
<td>64.811</td>
</tr>
<tr>
<td>Protein</td>
<td>0.008</td>
<td>474.995</td>
<td>445.170</td>
<td>469.929</td>
<td>433.486</td>
<td>461.251</td>
<td>429.679</td>
</tr>
</tbody>
</table>

### 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 \( m \)th input domain;
2. \( K \), the maximum number of training iterations;
3. \( N_{\text{bs}} \in [1, N] \), the minibatch size;
4. \( P \in (0, 1) \), the DropRule rate;
5. \( \alpha \), the initial learning rate (step size);
6. \( \lambda \), the \( \ell_2 \) regularization coefficient;
7. \( \beta_1, \beta_2 \in [0, 1] \), exponential decay rates for the moment estimates in AdaBound;
8. \( \epsilon \), a small positive number in AdaBound;
The firing level of each rule in the $l$ with both options. $X$, ANFIS-GD always achieved the MBGD-RDA. The normalized firing levels of the rules are computed in MATLAB ANFIS-GD-LSE. $X$ in achieving rapid convergence. was always very high at the beginning, and $\epsilon$ can be determined by early. $0.99, 0.01, P$ ANFIS-GD MBGD-RDA $X$ is always converged much faster ANFIS-GD. $u$ $X$ is a parameter for all TSK fuzzy systems, $l$ with ANFIS on the first six smaller $\epsilon$ 500, $u$ $\epsilon$ ANFIS-GD-LSE $=6$ 4 $\lambda$ =0 ANFIS- $ANFIS-GD$ ANFIS-GD $X$ had significant overfitting. If we could reduce this $x$ is computed, to MBGD-RDA ANFIS- $X$ is almost always smaller than that of anfis $=0$ in MATLAB has two $−=0$ $x$ $w.r.t.$ different hyperparameters on the $0.05, \alpha$ MBGD-RDA are defined in $\beta$ are used. $\lambda$ are defined in (12) and (13), respectively. Rule $u$ MBGD-RDA, $\alpha$ on $K$ and $\beta$ cannot. $X$ MBGD-RDA is computed, by (5). $X$ is stable with respect to each of the four $\beta$ and $\alpha$. clearly, MBGD-RDA is stable with respect to each of the four hyperparameters in a wide range, which is desirable. 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 $\beta_1, \beta_2, \epsilon, l(k)$ and $u(k)$ are AdaBound parameters, whose default values are usually used. Therefore, we only studied the sensitivity of MBGD-RDA to $N_{bs}$, $P$, $\alpha$, and $\lambda$, 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 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. 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 $Rule_i$ 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.  

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 $i$th MF in the $m$th 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 $i$th MF in the $m$th 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

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, 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 Micronano System Engineering and the Department of Mechatronics and Informational Systems, Nagoya University, Nagoya, Japan. In 2015, he was a Research Fellow with Nagoya University supported by JSPS invitation fellowship. He is currently an Associate 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.

Authorized licensed use limited to: Huazhong University of Science and Technology. Downloaded on July 06,2020 at 15:13:04 UTC from IEEE Xplore. Restrictions apply.