Optimize TSK Fuzzy Systems for Classification Problems: Minibatch Gradient Descent With Uniform Regularization and Batch Normalization

Yuqi Cui, Dongrui Wu, *Senior Member, IEEE*, and Jian Huang, *Senior Member, IEEE*

***Abstract*—Takagi–Sugeno–Kang (TSK) fuzzy systems are flexible and interpretable machine learning models; however, they may not be easily optimized when the data size is large, and/or the data dimensionality is high. This article proposes a minibatch gradient descent (MBGD) based algorithm to efficiently and effectively train TSK fuzzy classifiers. It integrates two novel techniques: First, uniform regularization (UR), which forces the rules to have similar average contributions to the output, and hence to increase the generalization performance of the TSK classifier; and, second, batch normalization (BN), which extends BN from deep neural networks to TSK fuzzy classifiers to expedite the convergence and improve the generalization performance. Experiments on 12 UCI datasets from various application domains, with varying size and dimensionality, demonstrated that UR and BN are effective individually, and integrating them can further improve the classification performance.**

***Index Terms*—Batch normalization (BN), minibatch gradient descent, Takagi–Sugeno–Kang (TSK) fuzzy classifier, uniform regularization (UR).**

I. Introduction

Takagi–Sugeno–Kang (TSK) fuzzy systems [1] have achieved great success in numerous applications, including both classification and regression problems. Many optimization approaches have been proposed for them.

There are generally three strategies for fine-tuning the TSK fuzzy system parameters after initialization: first, evolutionary algorithms [2]–[5]; second, gradient descent (GD) based algorithms [6]; and, three, GD plus least squares estimation (LSE), represented by the popular adaptive-network-based fuzzy inference system (ANFIS) [7]. However, these approaches may have challenges when the size and/or the dimensionality of the data increase. Evolutionary algorithms need to keep a large population of candidate solutions, and evaluate the fitness of each, which result in high computational cost and heavy memory requirement for big data. Traditional GD needs to compute the gradients from the entire dataset to iteratively update the model parameters, which may be very slow, or even impossible, when the data size is very large. The memory requirement and computational cost of LSE also increase rapidly when the data size and/or dimensionality increase. Additionally, as shown in [8], ANFIS may result in significant overfitting in regression problems.

Many efforts have been done to tackle the difficulty in optimizing the TSK fuzzy systems on big and/or high-dimensional data [9]–[11]. Dimensionality reduction and/or feature selection are usually used to reduce the number of fuzzy partitions (rules). Traditional dimensionality reduction techniques such as principal component analysis (PCA) has been used for the TSK fuzzy system optimization [12], [13]. There are also methods focusing on learning a sparse subspace of the original feature space to reduce the number of antecedents in each rule [14], [15]. Once the number of antecedents is determined, different optimization approaches can be used to tune the TSK fuzzy system on large datasets. For example, Chung *et al.* [11] utilized the equivalence between minimum enclosing ball and the Mamdani–Larsen fuzzy inference system to train the latter using the former. Gacto *et al.* [16] proposed a multiobjective evolutionary algorithm to optimize TSK fuzzy systems for high-dimensional large-scale regression problems.

Minibatch gradient descent (MBGD) [17], [18] based optimization, which is particularly popular in deep learning, can also be a solution to train TSK fuzzy systems on large and high-dimensional datasets. In each iteration, MBGD computes the gradients from a randomly selected small batch of data, instead of the entire dataset [19]. Different batch sizes can be used, according to the tradeoff among the available memory, the training speed, and the expected generalization performance. The original MBGD used a constant learning rate to update the model's parameters [19]. Later, Sutskever *et al.* [20] found that adding a momentum to MBGD can improve the final training performance. However, it still needs to manually select a learning rate, and the convergence may be very slow at the beginning. Kingma and Ba [21] proposed the well-known Adam algorithm to automatically rescale the gradients to achieve adaptive and individualized learning rate for each parameter, which leads to faster convergence. However, the generalization performance of Adam may not be as good as the momentum [22]; so Keskar and

Manuscript received August 1, 2019; revised November 30, 2019 and January 9, 2020; accepted January 13, 2020. Date of publication January 17, 2020; date of current version December 1, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61873321 and in part by the Technology Innovation Project of Hubei Province of China under Grant 2019AEA171. *(Corresponding authors: Dongrui Wu; Jian Huang.)*

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

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

Digital Object Identifier 10.1109/TFUZZ.2020.2967282

Socher [23] also tried to combine the advantages of momentum and Adam to achieve both fast convergence and good generalization. Recently, Luo *et al.* [24] also proposed AdaBound to improve Adam. AdaBound uses an adaptive bound for the learning rate of each parameter to force the optimizer to behave like Adam at the beginning and like stochastic GD at the end. Our very recent research [8] has found that TSK fuzzy systems can achieve better performance with AdaBound than Adam for regression problems.

Although MBGD-based optimization has many advantages, it may be easily trapped into a local-minimum, and may face the gradient vanishing problem. Many other techniques have been proposed to complement MBGD for better performance. In 2015, Ioffe and Szegedy [25] proposed the well-known batch normalization (BN) approach to accelerate the training of deep neural networks by reducing the internal covariate shift.[1] BN normalizes the input distribution of each layer, so it also alleviates the gradient vanishing problem. It has been used almost ubiquitously in deep learning, and many variants [27]–[30] have also been proposed.

This article, following our previous research [8] on MBGD-based optimization of TSK fuzzy systems for regression problems, considers classification problems. We use AdaBound, as in [8], to adjust the learning rates. Additionally, we propose two novel techniques for training TSK fuzzy systems for classification problems, namely, uniform regularization (UR) and BN. Our main contributions are as follows.

1) We introduce a novel UR term to the cross-entropy loss function in training TSK fuzzy classifiers, which forces all rules to have similar average firing levels on the entire dataset. Experiments show that UR can improve the generalization performance of TSK fuzzy classifiers.
2) We extend BN from the training of deep neural networks to the training of TSK fuzzy classifiers, and show that it can speed up the convergence in training and improve the generalization performance in testing.
3) We further integrate UR and BN, and show that the combined approach outperforms each individual ones.

The remainder of this article is organized as follows. Section II introduces the proposed UR and BN approaches. Section III presents the experimental results to validate the performances of UR and BN. Section IV draws conclusions and points out some future research directions.

II. UR AND BN

This section introduces the details of the TSK fuzzy classifier under consideration, our proposed UR for regularizing the loss function, and BN for more efficient and effective training of the TSK fuzzy classifier. Python implementation of our algorithm can be downloaded at https://github.com/YuqiCui/TSK_BN_UR.

[1]Recently some researchers had different opinions on why BN works. For example, Santurkar *et al.* [26] argued that BN may not reduce the internal covariate shift; instead, it helps improve the Lipschitzness of both the loss and the gradients, and also reduces the dependence on the training hyperparameters, such as the learning rate and the regularization weights.

A. TSK Fuzzy Classifier

Let the training dataset be $\mathcal{D}=\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, in which $\boldsymbol{x}_n=[x_{n,1},\ldots,x_{n,D}]^T\in\mathbb{R}^{D\times 1}$ is a D-dimensional feature vector, and $y_n\in\{1,2,\ldots,C\}$ the corresponding class label for a C-class classification problem.

Suppose the TSK fuzzy classifier has R rules in the following form:

$$\begin{aligned}\text{Rule}_r:\ &\text{IF } x_1 \text{ is } X_{r,1} \text{ and } \cdots \text{ and } x_D \text{ is } X_{r,D},\\ &\text{THEN } y_r^1(\boldsymbol{x})=b_{r,0}^1+\sum_{d=1}^D b_{r,d}^1\cdot x_d \text{ and}\cdots\\ &\text{and } y_r^C(\boldsymbol{x})=b_{r,0}^C+\sum_{d=1}^D b_{r,d}^C\cdot x_d \end{aligned} \tag{1}$$

where $X_{r,d}$ ($r=1,\ldots,R$; $d=1,\ldots,D$) is the membership function (MF) for the dth antecedent in the rth rule, and $b_{r,0}^c$ and $b_{r,d}^c$ ($c=1,\ldots,C$) are the consequent parameters for the cth class.

Different types of MFs can be used in our algorithm, as long as they are differentiable. For simplicity, Gaussian MFs are considered in this article, and the membership grade of x_d on $X_{r,d}$ is

$$\mu_{X_{r,d}}(x_d)=\exp\left(-\frac{(x_d-m_{r,d})^2}{2\sigma_{r,d}^2}\right) \tag{2}$$

where $m_{r,d}$ and $\sigma_{r,d}$ are the center and the standard deviation of the Gaussian MF, respectively.

The output of the TSK fuzzy classifier for the cth class is

$$y^c(\boldsymbol{x})=\frac{\sum_{r=1}^R f_r(\boldsymbol{x})y_r^c(\boldsymbol{x})}{\sum_{r=1}^R f_r(\boldsymbol{x})} \tag{3}$$

where

$$f_r(\boldsymbol{x})=\prod_{d=1}^D \mu_{X_{r,d}}(x_d)=\exp\left(-\sum_{d=1}^D\frac{(x_d-m_{r,d})^2}{2\sigma_{r,d}^2}\right) \tag{4}$$

is the firing level of Rule r. We can also rewrite (3) as

$$y^c(\boldsymbol{x})=\sum_{r=1}^R \overline{f}_r(\boldsymbol{x})y_r^c(\boldsymbol{x}) \tag{5}$$

where

$$\overline{f}_r(\boldsymbol{x})=\frac{f_r(\boldsymbol{x})}{\sum_{i=1}^R f_i(\boldsymbol{x})} \tag{6}$$

is the normalized firing level of Rule r.

Once the output vector $\boldsymbol{y}(\boldsymbol{x})=[y^1(\boldsymbol{x}),\ldots,y^C(\boldsymbol{x})]^T$ is obtained, the input $\boldsymbol{x}$ is assigned to the class with the largest $y^c(\boldsymbol{x})$.

To optimize the TSK fuzzy classifier, we need to fine-tune the antecedent MF parameters $m_{r,d}$ and $\sigma_{r,d}$, and the consequent parameters $b_{r,0}^c$ and $b_{r,d}^c$, where $r=1,\ldots,R$, $d=1,\ldots,D$, and $c=1,\ldots,C$.

B. Uniform Regularization

MoE [31], which is functionally equivalent to TSK fuzzy systems [32]–[34], is a popular machine learning algorithm.

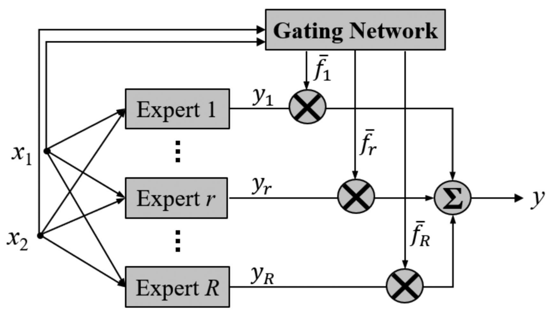

Fig. 1. Mixture of experts (MoE) [31].

Its model is shown in Fig. 1. It trains multiple local experts, each taking care of only a small local region of the input space. For a new input, the gating network determines the activations (weights) of the local experts, and the final output is a weighted average of the local expert outputs.

Although MoE has been used successfully in many applications, it may suffer from the "rich get richer" effect [35], [36]: once an expert is slightly better than others, it is always picked by the gating network, whereas other experts starve and are rarely used. This is bad for the generalization performance of the overall model.

Since MoE and TSK fuzzy systems are functionally equivalent [34], TSK fuzzy systems may also suffer from the "rich get richer" effect, i.e., only a few rules are always activated with large firing levels, whereas others have very small firing levels, and hence not adequately tuned in training. A remedy to the "rich get richer" effect in TSK fuzzy systems is to force the rules to be fired at similar degrees in the input space, so that each rule contributes equally to the output.

Next, we propose UR to achieve this goal.

UR forces the rules to have similar average firing levels, by minimizing the following loss:

$$\ell_{UR} = \sum_{r=1}^{R} \left(\frac{1}{N} \sum_{n=1}^{N} \overline{f}_r(\boldsymbol{x}_n) - \tau \right)^2 \tag{7}$$

where N is the number of training examples and τ the expected firing level of each rule, which is set to $1/C$ in this article (recall that C is the number of classes).

ℓ_{UR} can then be added to the original loss function in MBGD-based training of TSK fuzzy classifiers, i.e., for each minibatch with N training samples

$$\mathcal{L} = \ell + \alpha \ell_2 + \lambda \sum_{r=1}^{R} \left(\frac{1}{N} \sum_{n=1}^{N} \overline{f}_r(\boldsymbol{x}_n) - \frac{1}{R} \right)^2 \tag{8}$$

where ℓ is the cross-entropy loss between the estimated class probabilities (obtained by applying *softmax* to $\boldsymbol{y}(\boldsymbol{x})$) and the true class probabilities, ℓ_2 the L2 regularization of the rule consequent parameters, and α and λ the tradeoff parameters.

C. Batch Normalization

BN [25] is a very powerful technique in optimizing deep neural networks [37]–[39]. It normalizes the data distribution in each minibatch to accelerate the training. For a minibatch $\mathcal{B} = \{\boldsymbol{x}_n\}_{n=1}^N$, the output of BN is [25]

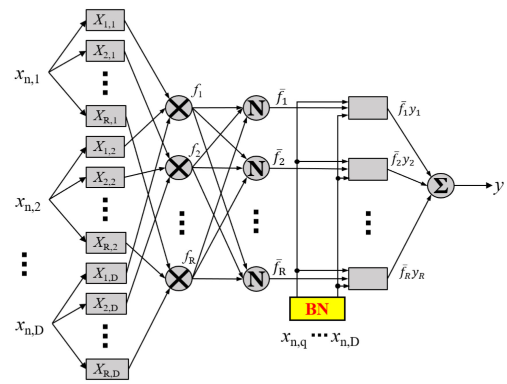

Fig. 2. BN in training a TSK fuzzy classifier. All rule consequents share the same BN layer.

$$\boldsymbol{x}'_n = BN(\boldsymbol{x}_n) = \gamma \frac{\boldsymbol{x}_n - \boldsymbol{m}_{\mathcal{B}}}{\sqrt{\boldsymbol{\sigma}_{\mathcal{B}}^2 + \epsilon}} + \beta \tag{9}$$

where $\boldsymbol{m}_{\mathcal{B}}$ and $\boldsymbol{\sigma}_{\mathcal{B}}$ are the mean and the standard deviation of the samples in the minibatch, respectively, γ and β are parameters to be learned during training, and ϵ is usually set to $1\text{e} - 8$ to avoid being divided by zero. During training, exponential weighted averages of $\boldsymbol{m}_{\mathcal{B}}$ and $\boldsymbol{\sigma}_{\mathcal{B}}$ are recorded so that they can be used in the test phase.

Since TSK fuzzy systems and neural networks share lots of similarity [34], we can extend BN to the optimization of TSK fuzzy classifiers, as shown in Fig. 2. In the training phase, we first compute the firing level of each rule using the unmodified inputs, as in traditional TSK fuzzy systems. Then, we use BN to normalize the inputs, according to their mean and standard deviation in the current minibatch. The normalized inputs are then used to compute the rule consequents. The final output is a weighted average of the rule consequents, the weights being the corresponding rule firing levels.

At the testing phase, the BN operation can be merged into the consequent layer. Assume that after training, we obtain a BN layer with learned $\boldsymbol{m} = (m_1, \ldots, m_D)^T$, $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_D)^T$, γ, and β. Then, the output y_r of the rth rule with BN is

$$y_r(BN(\boldsymbol{x}_n)) = b_{r,0} + \gamma \sum_{d=1}^{D} b_{r,d} \frac{x_{n,d} - m_d}{\sqrt{\sigma_d^2 + \epsilon}} + \beta D \tag{10}$$

which can be rewritten as

$$y_r(BN(\boldsymbol{x}_n)) = b'_{r,0} + \sum_{d=1}^{D} b'_{r,d} x_{n,d} \tag{11}$$

where

$$b'_{r,0} = b_{r,0} + \beta D - \gamma \sum_{d=1}^{D} \frac{m_d b_{r,d}}{\sqrt{\sigma_d^2 + \epsilon}} \tag{12}$$

$$b'_{r,d} = \gamma \frac{b_{r,d}}{\sqrt{\sigma_d^2 + \epsilon}}. \tag{13}$$

By doing this, the original architecture of the TSK fuzzy classifier is kept unchanged.

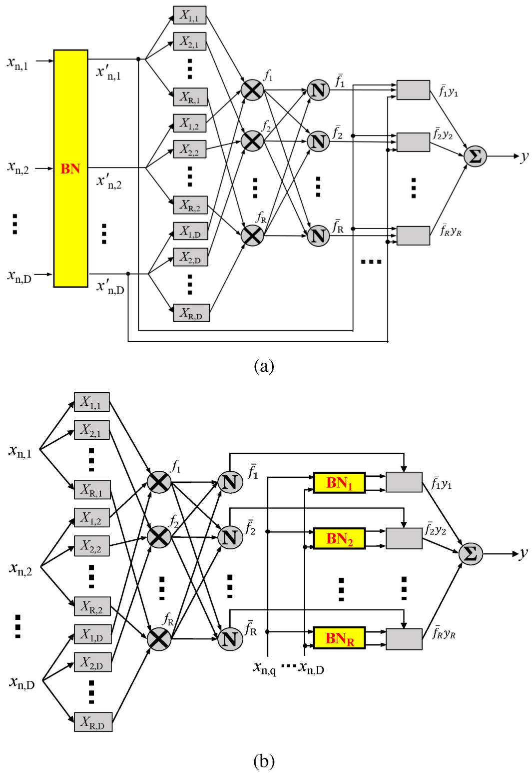

Fig. 3. (a) TSK fuzzy system with global BN (`TSK-MBGD-UR-GBN`). (b) TSK fuzzy system with rule-specific BN (`TSK-MBGD-UR-RBN`).

We also tested two variants of BN, as shown in Fig. 3. The TSK with global BN (`TSK-MBGD-UR-GBN`) approach in Fig. 3(a) uses the BN normalized inputs in both antecedents and consequents to compute the final output. In this case, the output of `TSK-MBGD-UR-GBN` for Class c is

$$y^c(\boldsymbol{x}) = \sum_{r=1}^{R} \overline{f}_r(BN(\boldsymbol{x}))y_r^c(BN(\boldsymbol{x})). \quad (14)$$

The TSK with rule-specific BN (`TSK-MBGD-UR-RBN`) approach in Fig. 3(b) uses the raw inputs to compute the antecedents, and rule-specific BN to compute each consequent individually. The output of `TSK-MBGD-UR-RBN` for Class c is

$$y^c(\boldsymbol{x}) = \sum_{r=1}^{R} \overline{f}_r(\boldsymbol{x})y_r^c(BN_r(\boldsymbol{x})) \quad (15)$$

where BN_r represents the BN operation for the rth rule.

`TSK-MBGD-UR-GBN` has the same computational cost as `TSK-MBGD-UR-BN`, but `TSK-MBGD-UR-RBN` has R times more BN parameters, and hence higher computational cost. Both of them can be reexpressed in the original TSK architecture. We also evaluate their performances in Section III-G.

III. Experiments and Results

This section validates the performances of our proposed UR and BN on multiple datasets from various application domains, with varying size and feature dimensionality.

TABLE I
Summary of the 12 Datasets

Index	Dataset	No. of Samples	No. of Features	No. of Classes
1	Vehicle[1]	846	18	4
2	Biodeg[2]	1,055	41	2
3	DRD[3]	1151	19	2
4	Yeast[4]	1,484	8	10
5	Steel[5]	1,941	27	7
6	IS[6]	2,310	19	7
7	Abalone[7]	4,177	10	3
8	Waveform21[8]	5,000	21	3
9	Page-blocks[9]	5,473	10	5
10	Satellite[10]	6,435	36	6
11	Clave[11]	10,798	16	4
12	MAGIC[12]	19,020	10	2

[1]https://archive.ics.uci.edu/ml/datasets/Statlog+%28Vehicle+Silhouettes%29
[2]https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation
[3]https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen + Data+Set
[4]https://archive.ics.uci.edu/ml/datasets/Yeast
[5]https://archive.ics.uci.edu/ml/datasets/Steel+Plates+Faults
[6]https://archive.ics.uci.edu/ml/datasets/Image+Segmentation
[7]https://archive.ics.uci.edu/ml/datasets/Abalone
[8]https://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+1)
[9]https://archive.ics.uci.edu/ml/datasets/Page+Blocks+Classification
[10]https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)
[11]https://archive.ics.uci.edu/ml/datasets/Firm-Teacher_Clave-Direction_Classification
[12]https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope

A. Datasets

We evaluated our proposed algorithms on 12 classification datasets from the UCI Machine Learning Repository.[2] Their characteristics are summarized in Table I. For each dataset, we randomly selected 70% samples as the training set and the remaining 30% as the test set for 30 times to get 30 different data splits. We ran each algorithm on these 30 data splits and report the average performance.

Some datasets contain both numerical features and categorical features. The categorical features were converted into numerical ones by one-hot coding. We z-normalized each feature using the mean and standard deviation computed from the training set.

B. Algorithms

We compared nine algorithms to validate our proposed approaches. Among them, four were tree-based approaches (`DT`, `RF`, `PART`, and `JRip`), one was a TSK fuzzy system optimized by a traditional approach (`TSK-FCM-LSE`), and the remaining four were TSK fuzzy systems optimized by MBGD based approaches (`TSK-MBGD`, `TSK-MBGD-BN`, `TSK-MBGD-UR`, `TSK-MBGD-UR-BN`).

The details of these nine algorithms are as follows.

1) `DT`: Decision tree implemented in scikit-learn[3] in Python. We used fivefold cross validation to select the maximum depth of the tree from $\{3, 4, 5, 6, 7\}$ on the training set. Other parameters were set by default.
2) `RF`: Random forest implemented in scikit-learn[4] in Python. We set the number of trees to 20 and used fivefold

[2]http://archive.ics.uci.edu/ml/index.php
[3]https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[4]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

cross validation to select the maximum depth of the trees from $\{3, 4, 5, 6, 7\}$ on the training set. Other parameters were set by default.

3) `PART` [40]: The PART (partial decision tree) classifier implemented in RWeka.[5] All parameters were set by default.
4) `JRip` [41]: The Repeated Incremental Pruning to Produce Error Reduction (RIPPER) classifier implemented in RWeka. All parameters were set by default.
5) `TSK-FCM-LSE` [42]: We used fuzzy c-means (FCM) clustering to estimate the antecedent parameters and LSE with L2 regularization to estimate the consequent parameters.
6) `TSK-MBGD`: We used MBGD and AdaBound [24] to optimize both the antecedent and the consequent parameters.
7) `TSK-MBGD-UR`: We used MBGD, AdaBound, and UR (see Section II-B) to optimize both the antecedent and the consequent parameters. The UR weight λ in (8) was selected from $\{0.1, 1, 10, 20, 50\}$ by cross validation on the training set.
8) `TSK-MBGD-BN`: We used MBGD, AdaBound, and BN (see Section II-C) to optimize both the antecedent and the consequent parameters.
9) `TSK-MBGD-UR-BN`: We used MBGD, AdaBound, BN, and UR to optimize both the antecedent and the consequent parameters. The UR weight λ in (8) was selected from $\{0.1, 1, 10, 20, 50\}$ by cross validation on the training set.

For `TSK-FCM-LSE`, `TSK-MBGD`, `TSK-MBGD-BN`, `TSK-MBGD-UR`, and `TSK-MBGD-UR-BN`, we set the L2 regularization weight $\alpha = 0.05$, and the number of rules $R = 20$. For `TSK-MBGD`, `TSK-MBGD-BN`, `TSK-MBGD-UR`, and `TSK-MBGD-UR-BN`, we set the learning rate of AdaBound to 0.01, following our previous work [8]. In order to make use of all data in the training set and to reduce overfitting simultaneously, we randomly sampled 20% data from the training set and trained the TSK model with early stopping five times. The maximum epoch number was 2000, and the patience of early stopping 40. We recorded the number of epochs at stopping in each run, and trained the final model with the average stopping epoch number on the entire training set.

k-mean clustering was used in the MBGD-based algorithms (`TSK-MBGD`, `TSK-MBGD-BN`, `TSK-MBGD-UR`, and `TSK-MBGD-UR-BN`) to initialize the antecedent parameters. We performed k-means clustering on the training set, where k equaled R, the number of rules. We then initialized the rule centers to the cluster centers, and randomly initialized the standard deviation $\sigma_{r,d}$ from a Gaussian distribution $\mathcal{N}(1, 0.2)$. For the consequent parameters, we set the initial bias of each rule to zero, and the attribute weight $b_{r,d}$ ($r = 1, \ldots, R$; $d = 1, \ldots, D$) randomly from a uniform distribution $U(-1, 1)$.

C. Performance Measures

The raw classification accuracy (RCA), which is the total number of correctly classified test samples divided by the total number of test samples, was used as our primary performance measure.

Since some datasets have significant class imbalance, in addition to the RCA, we also computed the balanced classification accuracy (BCA), which is the mean of the per-class RCAs, as our second performance measure.

D. Experimental Results

The average test RCAs and BCAs are shown in Tables II and III, respectively. The largest value (best performance) on each dataset is marked in bold. To facilitate the comparison, we also show the ranks of the RCAs and BCAs in Tables IV and V, respectively.

The following observations can be made from the above four tables:

1) *Generally, UR improved both RCA and BCA:* Comparing `TSK-MBGD` with `TSK-MBGD-UR`, and `TSK-MBGD-BN` with `TSK-MBGD-UR-BN`, we can conclude that generally UR improved the classification performance, regardless of whether BN was used or not. The average ranks in the last row of Tables IV and V demonstrate this more clearly: the average rank of `TSK-MBGD-UR` (`TSK-MBGD-UR-BN`) was smaller than that of `TSK-MBGD` (`TSK-MBGD-BN`).
2) *Generally, BN improved both RCA and BCA:* Comparing `TSK-MBGD` with `TSK-MBGD-BN`, and `TSK-MBGD-UR` with `TSK-MBGD-UR-BN`, we can conclude that generally BN improved the classification performance, regardless of whether UR was used or not. The average ranks in the last row of Tables IV and V demonstrate this more clearly: the average rank of `TSK-MBGD-BN` (`TSK-MBGD-UR-BN`) was smaller than that of `TSK-MBGD` (`TSK-MBGD-UR`).
3) *Generally, integrating BN and UR achieved further RCA and BCA improvements:* Comparing `TSK-MBGD-UR-BN` with `TSK-MBGD`, `TSK-MBGD-UR` and `TSK-MBGD-BN`, we can conclude that `TSK-MBGD-UR-BN` almost always performed the best on both RCA and BCA, as shown in Fig. 4. This indicated that BN and UR are somehow complementary, and hence integrating them may achieve better performance than using each one alone.
4) *Overall,* `TSK-MBGD-UR-BN`*achieved the best performance among the nine algorithms:* The last row of Table V shows that `TSK-MBGD-UR-BN` achieved the best average BCA performance, and the last row of Table IV shows that `TSK-MBGD-UR-BN` achieved the second best average RCA performance. Interestingly, `RF` had the best average rank on RCA, but only ranked the fifth on BCA, suggesting that `RF` may tend to overlook the minority classes. On the contrary, `TSK-MBGD-UR-BN` performed well on both RCA and BCA.

E. Statistical Analysis

To further evaluate the performance improvement of our proposed `TSK-MBGD-UR-BN` over others, we also performed nonparametric multiple comparison tests on the RCAs and BCAs using Dunn's procedure [43], with a p-value correction using the False Discovery Rate method [44]. The results are shown in

[5] https://cran.r-project.org/web/packages/RWeka/index.html

TABLE II
AVERAGE RCAS OF THE NINE ALGORITHMS ON THE 12 DATASETS

Dataset	CART	RF	JRip	PART	TSK-FCM-LSE	TSK-MBGD	TSK-MBGD-BN	TSK-MBGD-UR	TSK-MBGD-UR-BN
Vehicle	0.6907	0.7407	0.6892	0.7110	0.7411	0.6970	0.7354	0.7089	**0.7907**
Biodeg	0.8202	0.8572	0.8222	0.8362	0.8377	0.8523	0.8531	0.8539	**0.8609**
DRD	0.6283	0.6589	0.6240	0.6364	**0.6824**	0.6623	0.6618	0.6713	0.6720
Yeast	0.5564	**0.5963**	0.5731	0.5340	0.5851	0.5673	0.5770	0.5722	0.5725
Steel	0.7017	0.7328	0.7135	0.7120	0.6527	0.5864	0.7110	0.7248	**0.7350**
IS	0.9320	0.9529	0.9481	**0.9608**	0.9571	0.5762	0.7557	0.8559	0.9501
Abalone	0.7170	0.7314	0.7254	0.7104	**0.7323**	0.5821	0.7129	0.6238	0.7306
Waveform21	0.7641	0.8369	0.7908	0.7843	**0.8647**	0.6779	0.8002	0.8363	0.8234
Page-blocks	0.9651	**0.9688**	0.9681	0.9677	0.9499	0.9375	0.9419	0.9515	0.9580
Satellite	0.8524	0.8863	0.8587	0.8592	0.8864	0.4890	0.8001	0.8929	**0.8943**
Clave	0.7103	0.7600	0.7344	0.7779	0.7690	0.8223	**0.8427**	0.8187	0.8192
MAGIC	0.8427	0.8531	0.8455	0.8488	0.8319	0.7347	0.7861	0.8574	**0.8392**
Average	0.7651	0.7979	0.7744	0.7782	0.7909	0.6821	0.7648	0.7806	**0.8038**

TABLE III
AVERAGE BCAS OF THE NINE ALGORITHMS ON THE 12 DATASETS

Dataset	CART	RF	JRip	PART	TSK-FCM-LSE	TSK-MBGD	TSK-MBGD-BN	TSK-MBGD-UR	TSK-MBGD-UR-BN
Vehicle	0.6936	0.744	0.6939	0.7131	0.7443	0.7010	0.7380	0.7127	**0.7930**
Biodeg	0.7973	0.8306	0.7899	0.8122	0.8205	0.8368	0.8318	0.8390	**0.8439**
DRD	0.634	0.6624	0.6227	0.6422	**0.6845**	0.6642	0.6634	0.6717	0.6729
Yeast	0.3998	0.4867	0.5203	0.4889	0.5102	0.4951	0.5184	0.4946	**0.5332**
Steel	0.7005	0.6937	0.7129	0.7267	0.6319	0.5933	0.7258	0.7245	**0.7515**
IS	0.932	0.9529	0.9481	**0.9607**	0.9571	0.5762	0.7557	0.8559	0.9501
Abalone	0.5319	0.5362	0.5371	0.5280	0.5402	0.4567	0.5236	0.4791	**0.5402**
Waveform21	0.7637	0.8365	0.7905	0.7844	**0.8645**	0.6784	0.8003	0.8362	0.8233
Page-blocks	0.7986	0.7385	**0.8192**	0.8162	0.6003	0.5129	0.5609	0.6033	0.671
Satellite	0.8204	0.8480	0.8308	0.834	0.8558	0.4337	0.7651	0.8679	**0.8700**
Clave	0.4701	0.4878	0.4985	**0.6507**	0.4825	0.5876	0.6468	0.6374	0.6421
MAGIC	0.8058	0.8108	0.8052	0.8135	0.7886	0.6325	0.7128	**0.8225**	0.7934
Average	0.6956	0.7190	0.714	0.7309	0.7067	0.5974	0.6869	0.7120	**0.7404**

TABLE IV
RCA RANKS OF THE NINE ALGORITHMS ON THE 12 DATASETS

Dataset	CART	RF	JRip	PART	TSK-FCM-LSE	TSK-MBGD	TSK-MBGD-BN	TSK-MBGD-UR	TSK-MBGD-UR-BN
Vehicle	8	3	9	5	2	7	4	6	1
Biodeg	9	2	8	7	6	5	4	3	1
DRD	8	6	9	7	1	4	5	3	2
Yeast	8	1	4	9	2	7	3	6	5
Steel	7	2	4	5	8	9	6	3	1
IS	6	3	5	1	2	9	8	7	4
Abalone	5	2	4	7	1	9	6	8	3
Waveform21	8	2	6	7	1	9	5	3	4
Page-blocks	4	1	2	3	7	9	8	6	5
Satellite	7	4	6	5	3	9	8	2	1
Clave	9	7	8	5	6	2	1	4	3
MAGIC	5	2	4	3	7	9	8	1	6
Average	7.0	2.9	5.8	5.3	3.8	7.3	5.5	4.3	3.0

TABLE V
BCA RANKS OF THE NINE ALGORITHMS ON THE 12 DATASETS

Dataset	CART	RF	JRip	PART	TSK-FCM-LSE	TSK-MBGD	TSK-MBGD-BN	TSK-MBGD-UR	TSK-MBGD-UR-BN
Vehicle	9	3	8	5	2	7	4	6	1
Biodeg	8	5	9	7	6	3	4	2	1
DRD	8	6	9	7	1	4	5	3	2
Yeast	9	8	2	7	4	5	3	6	1
Steel	6	7	5	2	8	9	3	4	1
IS	6	3	5	1	2	9	8	7	4
Abalone	5	4	3	6	1	9	7	8	2
Waveform21	8	2	6	7	1	9	5	3	4
Page-blocks	3	4	1	2	7	9	8	6	5
Satellite	7	4	6	5	3	9	8	2	1
Clave	9	7	6	1	8	5	2	4	3
MAGIC	4	3	5	2	7	9	8	1	6
Average	6.8	4.7	5.4	4.3	4.2	7.3	5.4	4.3	2.6

TABLE VI
p-VALUES OF NONPARAMETRIC MULTIPLE COMPARISONS ON THE RCAS AND BCAS

	Metric	CART	RF	JRip	PART	TSK-FCM-LSE	TSK-MBGD	TSK-MBGD-BN	TSK-MBGD-UR
TSK-MBGD-BN	RCA	**0.0097**	0.0628	0.1368	0.3239	0.2723	**0.0000**	-	-
	BCA	0.1547	0.1627	0.4508	0.0981	0.4518	**0.0000**	-	-
TSK-MBGD-UR	RCA	**0.0001**	0.3740	**0.0090**	0.0460	0.2452	**0.0000**	0.1146	-
	BCA	**0.0036**	0.2900	0.0912	0.4420	0.0921	**0.0000**	0.0731	-
TSK-MBGD-UR-BN	RCA	**0.0000**	0.2113	**0.0002**	**0.0025**	0.0404	**0.0000**	**0.0094**	0.1409
	BCA	**0.0000**	0.0291	**0.0021**	0.0730	**0.0022**	**0.0000**	**0.0013**	0.0986

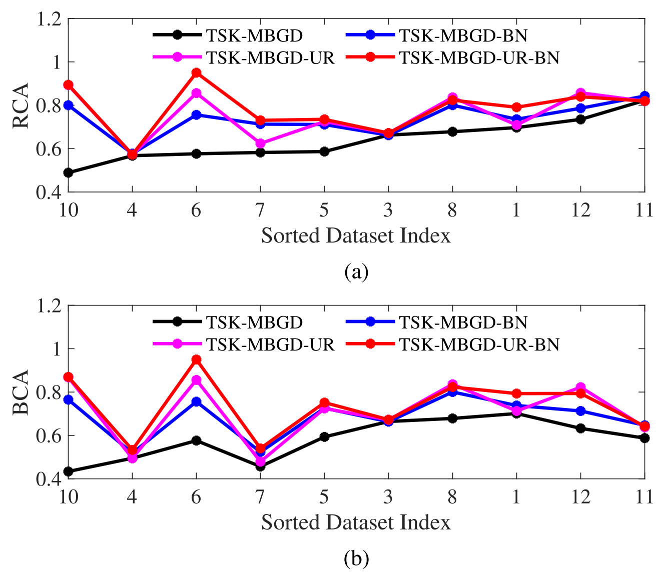

Fig. 4. (a) RCAs. (b) BCAs of the four MBGD-based TSK fuzzy classifiers on the 12 datasets. Datasets were sorted according to the RCAs of the TSK-MBGD model. The indices along the horizontal axis denote the dataset indices in Table I.

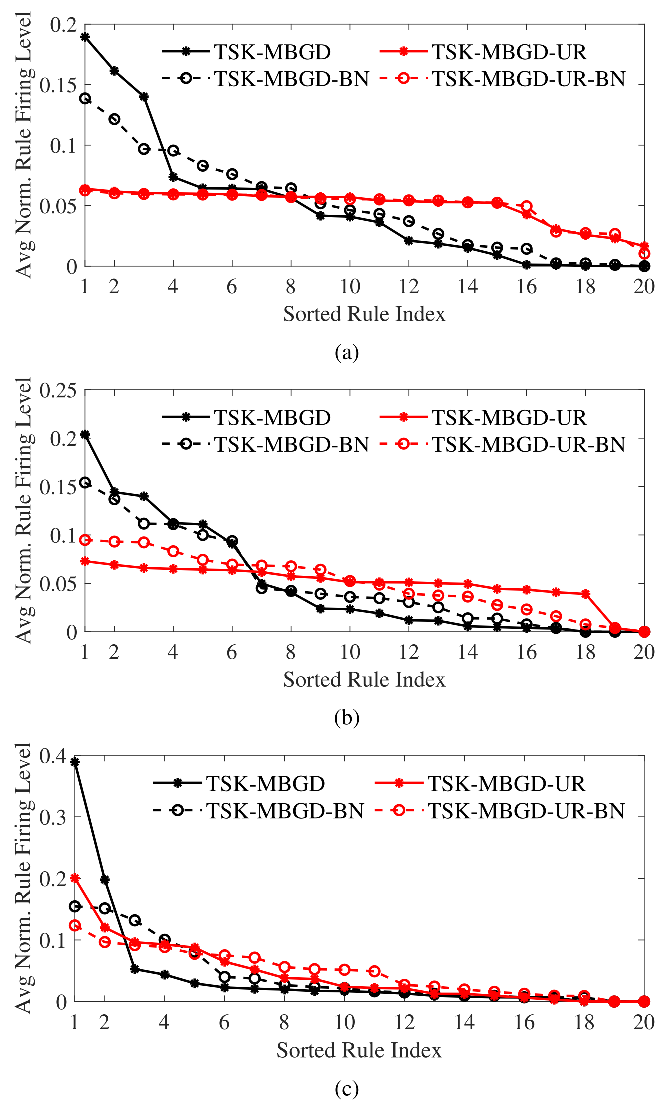

Fig. 5. Average normalized rule firing levels of TSK-MBGD, TSK-MBGD-BN, TSK-MBGD-UR, and TSK-MBGD-UR-BN. (a) Satellite. (b) Vehicle. (c) Biodeg datasets.

Table VI, where the statistically significant ones are marked in bold.

Table VI demonstrates that our proposed BN and UR can significantly improve the generalization performance of the traditional MBGD optimization for TSK fuzzy classifiers. TSK-MBGD-UR-BN statistically significantly outperformed CART, JRip, PART, TSK-MBGD, and TSK-MBGD-BN on RCA, and also statistically significantly outperformed CART, JRip, TSK-FCM-LSE, TSK-MBGD, and TSK-MBGD-BN on BCA. Although the performance improvement of TSK-MBGD-UR-BN over RF and TSK-MBGD-UR was not statistically significant, they were quite close to the threshold, especially for the BCA.

F. Effect of UR

As mentioned in Section II-B, using MBGD to optimize the TSK fuzzy system may face the "rich get richer" problem. To demonstrate this, Fig. 5 shows the average normalized firing levels of the rules on the entire dataset after the four MBGD-based TSK models were trained, on three representative datasets. For TSK-MBGD, a few "richest" rules had much larger average firing levels than others, and hence the rules contributed significantly differently to the output. BN may help alleviate this problem a little bit, as the average normalized rule firing levels in TSK-MBGD-BN were more uniform than those in TSK-MBGD, which also resulted in better classification performances, as demonstrated in the previous section. However, UR had the most direct effect on alleviating the "rich get richer" problem, as TSK-MBGD-UR (TSK-MBGD-UR-BN) had much more uniform average normalized rule firing levels than TSK-MBGD (TSK-MBGD-BN), and, hence, also better classification performance.

Note that we set $\tau = 1/C$ in (8), where $C = 6$ for Satellite, $C = 4$ for Vehicle, and $C = 2$ for Biodeg. However, the actual

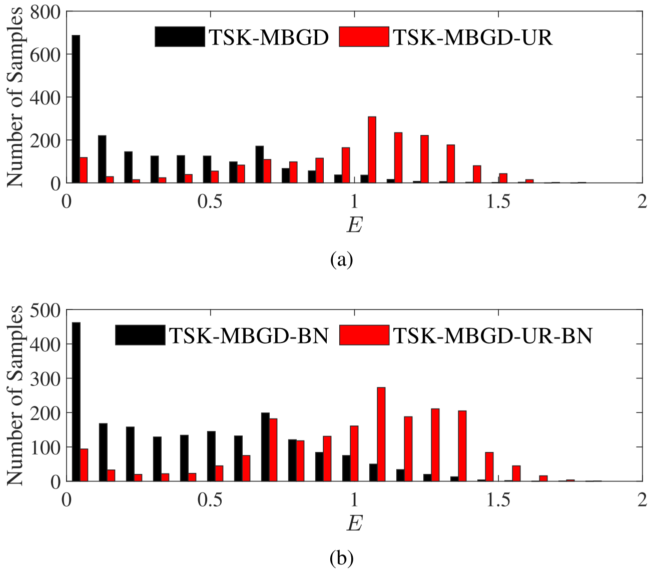

Fig. 6. Histogram of the normalized rule firing level entropy E. (a) `TSK-MBGD` and `TSK-MBGD-UR`. (b) `TSK-MBGD-BN` and `TSK-MBGD-UR-BN` on the Satellite dataset.

average normalized rule firing levels were not exactly τ on these datasets. Our experiments showed that although UR cannot guarantee the average normalized rule firing levels to be around τ, it can indeed make the rules fired more uniformly.

Why may making the rules fired more uniformly help improve the generalization performance? In [34], we pointed out that a TSK fuzzy system may be functionally equivalent to an adaptive stacking ensemble model, in which each rule can be viewed as a base learner, and the aggregation weights equal the corresponding rule firing levels. When the rule firing levels are more uniform, generally more rules are utilized in computing the output, i.e., more base learners are used in the stacking ensemble model, which may help improve the generalization performance.

To demonstrate this, we computed the entropy of the normalized rule firing levels for each input example

$$E = -\sum_{r}^{R} \overline{f}_r \, \log \overline{f}_r \tag{16}$$

where $\overline{f}_r$ is the normalized firing level of the rth rule. Generally, a larger entropy means more rules were fired.

Fig. 6 shows the histogram of the entropy distributions on the Satellite dataset. When training TSK fuzzy systems without UR, many samples had close to zero E, i.e., all except one rule had firing levels close to zero. When UR was added, the number of examples with close to zero E decreased significantly, i.e., more rules with larger firing levels were used in computing the output.

G. Effect of BN

We also used the Satellite dataset to analyze the effect of BN.

We set the UR weight $\lambda = 1$ and recorded the training loss and test BCA in the first 20 training epochs. This process was repeated ten times, and the average results are shown in Fig. 7(a) and (b), respectively. BN resulted in smaller training losses and better generalization performances in testing.

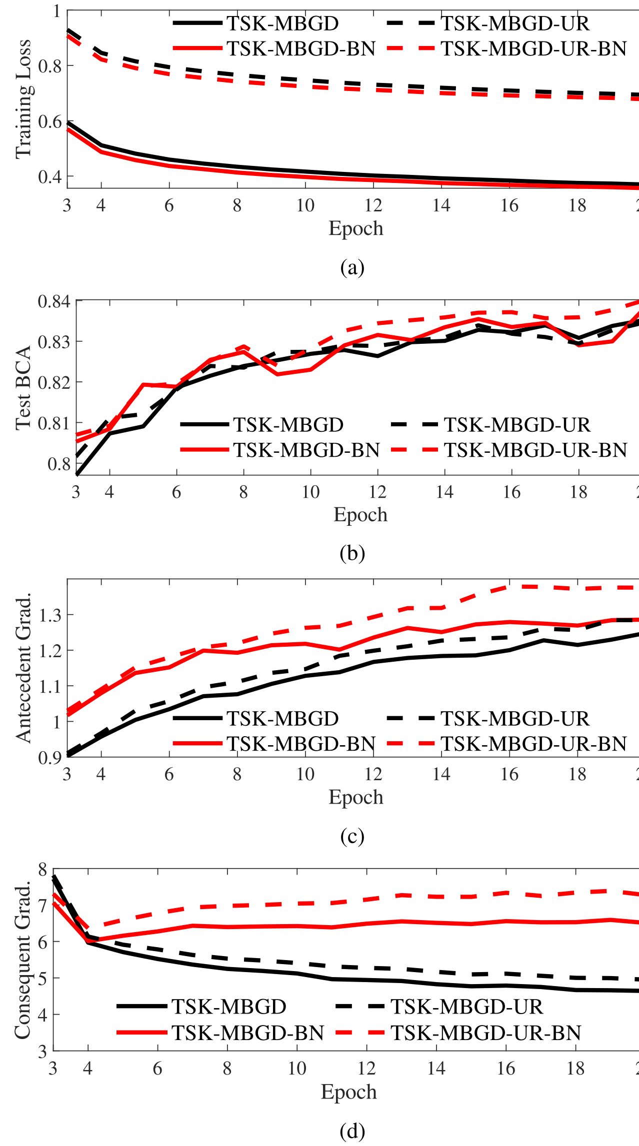

Fig. 7. (a) Training loss. (b) Test BCA. (c) L1 norm of the antecedent parameters' gradients. (d) L1 norm of the consequent parameters' gradients, in the first 20 training epochs on the Satellite dataset. The horizontal axis starts from three epochs so that the differences among the curves can be more clearly visualized.

There is still no agreement on theoretically why BN is helpful in optimizing deep neural networks [26]; thus, it is also challenging to analyze theoretically why BN can help the optimization of TSK fuzzy systems. Nevertheless, we performed an empirical study to peek into this, by recording the L1 norm of the antecedent parameters' gradients and the L1 norm of the consequent parameters' gradients in the first 20 training epochs on the Satellite dataset. The results are shown in Fig. 7(c) and (d), respectively. BN significantly increased the gradients of both antecedent and consequent parameters. With the same learning rate, this can expedite the convergence.

TABLE VII
AVERAGE BCAS OF THE THREE BN VARIANTS ON THE 12 DATASETS

Dataset	TSK-MBGD-UR	TSK-MBGD-UR-BN	TSK-MBGD-UR-GBN	TSK-MBGD-UR-RBN
Vehicle	0.7127	**0.7930**	0.7261	0.7679
Biodeg	0.8390	0.8439	0.8422	**0.8440**
DRD	0.6717	**0.6729**	0.6636	0.6650
Yeast	0.4946	0.5332	0.4352	**0.5339**
Steel	0.7245	**0.7515**	0.7332	0.7219
IS	0.8559	**0.9501**	0.9115	0.8938
Abalone	0.4791	**0.5402**	0.4924	0.5275
Waveform21	**0.8362**	0.8233	0.8232	0.8334
Page-blocks	0.6033	**0.6710**	0.5912	0.6333
Satellite	0.8679	**0.8700**	0.8679	0.8216
Clave	0.6374	0.6421	0.6090	**0.6442**
MAGIC	0.8225	0.7934	**0.8319**	0.8318
Average	0.7121	**0.7404**	0.7106	0.7265

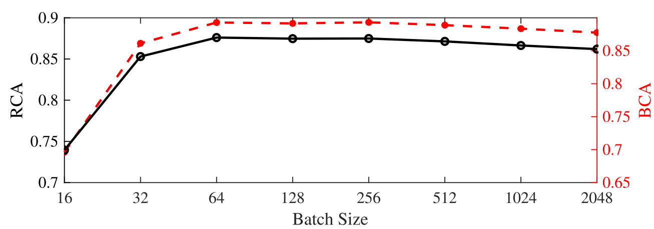

Fig. 8. Average RCAs and BCAs of TSK-MBGD-UR-BN on the Satellite dataset, using different batch sizes.

We also evaluated the performances of the two BN variants introduced in Section II-C. The BCAs of TSK-MBGD-UR, TSK-MBGD-UR-BN, TSK-MBGD-UR-GBN, and TSK-MBGD-UR-RBN are shown in Table VII. TSK-MBGD-UR-BN performed the best, and TSK-MBGD-UR-GBN the worst. Since TSK-MBGD-UR-RBN had more parameters to optimize, its training was not as stable as TSK-MBGD-UR-BN and TSK-MBGD-UR-GBN. Therefore, TSK-MBGD-UR-BN is the best choice.

H. Effect of the Batch Size

The batch size is an important hyperparameter in MBGD-based optimization. It determines the memory requirement and the convergence speed in training. A larger batch size leads to faster convergence, but also requires more memory. In [45], Keskar *et al.* analyzed the effect of the batch size on the generalization performance. Their results showed that using a larger batch size causes degradation in the model generalization performance, because it tends to converge to a shaper minimum, which makes the model sensitive to noise. A similar finding was presented in [46] that a smaller batch size leads to more stable and reliable training. However, since we used the mean, standard deviation, and mean firing level of each batch to compute the losses, too small batch size may also lead to poor performance.

We validated our model on the Satellite dataset with batch size varying from 16 to 2048. The test RCAs and BCAs averaged over 30 runs are shown in Fig. 8. The test performance decreased with too small or too large batch sizes. For TSK-MBGD-UR-BN, it seems that a batch size within [64, 256] is a good choice.

IV. CONCLUSION AND FUTURE RESEARCH

TSK fuzzy systems are powerful and frequently used machine learning models, for both regression and classification. However, they may not be easily applicable to large and/or high-dimensional datasets. Our very recent research [8] proposed an MBGD-based efficient and effective training algorithm (MBGD-RDA) for TSK fuzzy systems for regression problems. This article has proposed an MBGD-based algorithm, TSK-MBGD-UR-BN, to train TSK fuzzy systems for classification problems. It can deal with both small and big data with different dimensionalities, and may be the only algorithm that can train a TSK fuzzy classifier on big and high-dimensional datasets. TSK-MBGD-UR-BN integrates two novel techniques, which are also first proposed in this article:

1) UR, which is a regularization term in the loss function to ensure that all rules are fired similarly on average, and hence to improve the generalization performance.
2) BN, which normalizes the inputs in computing the rule consequents to speedup the convergence and to improve the generalization.

Experiments on 12 UCI datasets from various domains, with varying size and feature dimensionality, demonstrated that each of UR and BN has its own unique advantages, and integrating them can achieve the best classification performance. TSK-MBGD-UR-BN, together with MBGD-RDA proposed in [8], shall greatly promote the applications of TSK fuzzy systems in both classification and regression, especially for big data problems.

The proposed TSK-MBGD-UR-BN also has some limitations, which will be addressed in our future research. First, for very high dimensional data, fuzzy partitions of the input space become very complicated, and numeric underflow may happen when the product t-norm is used. Further research shall consider rules that automatically select the most relevant attributes as the antecedents. Second, we shall investigate how to improve the interpretability of data-driven TSK fuzzy systems. This is also partially linked to the first problem, as reducing the number of antecedents can improve the interpretability of the rules.

REFERENCES

[1] A.-T. 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.
[2] Y. Shi, R. Eberhart, and Y. Chen, "Implementation of evolutionary fuzzy systems," *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 2, pp. 109–119, Apr. 1999.
[3] D. Wu and W. W. Tan, "Genetic learning and performance evaluation of interval type-2 fuzzy logic controllers," *Eng. Appl. Artif. Intell.*, vol. 19, no. 8, pp. 829–841, 2006.
[4] O. Castillo, F. Valdez, J. Soria, L. Amador-Angulo, P. Ochoa, and C. Peraza, "Comparative study in fuzzy controller optimization using bee colony, differential evolution, and harmony search algorithms," *Algorithms*, vol. 12, no. 1, p. 9, 2019.
[5] O. Castillo and L. Amador-Angulo, "A generalized type-2 fuzzy logic approach for dynamic parameter adaptation in bee colony optimization applied to fuzzy controller design," *Inform. Sci.*, vol. 460, pp. 476–496, 2018.
[6] 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, Sep. 1992, pp. 1409–1418.

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

[8] D. Wu, Y. Yuan, J. Huang, and Y. Tan, "Optimize TSK fuzzy systems for big data regression problems: Mini-batch gradient descent with regularization, DropRule and AdaBound (MBGD-RDA)," *IEEE Trans. Fuzzy Syst.*, to be published, doi: 10.1109/TFUZZ.2019.2958559.

[9] 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.

[10] Y. Deng, Z. Ren, Y. Kong, F. Bao, and Q. Dai, "A hierarchical fused fuzzy deep neural network for data classification," *IEEE Trans. Fuzzy Syst.*, vol. 25, no. 4, pp. 1006–1012, Aug. 2017.

[11] F.-L. Chung, Z. Deng, and S. Wang, "From minimum enclosing ball to fast fuzzy inference system training on large datasets," *IEEE Trans. Fuzzy Syst.*, vol. 17, no. 1, pp. 173–184, Feb. 2009.

[12] M. Nilashi, O. Bin Ibrahim, N. Ithnin, and N. H. Sarmin, "A multi-criteria collaborative filtering recommender system for the tourism domain using expectation maximization (EM) and PCA–ANFIS," *Electron. Commerce Res. Appl.*, vol. 14, no. 6, pp. 542–562, 2015.

[13] C. K. Lau, K. Ghosh, M. A. Hussain, and C. R. C. Hassan, "Fault diagnosis of Tennessee Eastman process with multi-scale PCA and ANFIS," *Chemometrics Intell. Lab. Syst.*, vol. 120, pp. 1–14, 2013.

[14] Z. Deng, K.-S. Choi, Y. Jiang, J. Wang, and S. Wang, "A survey on soft subspace clustering," *Inform. Sci.*, vol. 348, pp. 84–106, 2016.

[15] Z. Deng, K.-S. Choi, F.-L. Chung, and S. Wang, "Enhanced soft subspace clustering integrating within-cluster and between-cluster information," *Pattern Recognit.*, vol. 43, no. 3, pp. 767–781, 2010.

[16] M. J. Gacto, M. Galende, R. Alcalá, and F. Herrera, "METSK-HDe: A multiobjective evolutionary algorithm to learn accurate TSK-fuzzy systems in high-dimensional and large-scale regression problems," *Inform. Sci.*, vol. 276, pp. 63–79, 2014.

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

[18] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[19] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc. Int. Conf. Comput. Statist.*, Paris, France, Aug. 2010, pp. 177–186.

[20] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, Atlanta, GA, USA, Jun. 2013, pp. 1139–1147.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, San Diego, CA, USA, May 2015.

[22] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 4148–4158.

[23] N. S. Keskar and R. Socher, "Improving generalization performance by switching from Adam to SGD," 2017, *arXiv:1712.07628*.

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

[25] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, Lille, France, Jul. 2015.

[26] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Proc. Adv. Neural Inf. Process. Syst.*, Montréal, QC, Canada, Dec. 2018, pp. 2483–2493.

[27] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[28] L. Fan, "Revisit fuzzy neural network: Demystifying batch normalization and ReLU with generalized hamming network," in *Proc. Adv. Neural Inform. Process. Syst.*, Long Beach, CA, USA, Dec. 2017, pp. 1923–1932.

[29] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," 2015, *arXiv:1511.07289*.

[30] Y. Wu and K. He, "Group normalization," in *Proc. Eur. Conf. Comput. Vision*, Munich, Germany, Sep. 2018, pp. 3–19.

[31] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, no. 1, pp. 79–87, 1991.

[32] H. Bersini and G. Bontempi, "Now comes the time to defuzzify neuro-fuzzy models," *Fuzzy Sets Syst.*, vol. 90, no. 2, pp. 161–169, 1997.

[33] H. Andersen, A. Lotfi, and L. Westphal, "Comments on 'functional equivalence between radial basis function networks and fuzzy inference systems' [with reply]," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1529–1532, Nov. 1998.

[34] 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.

[35] T. Shen, M. Ott, M. Auli, and M. Ranzato, "Mixture models for diverse machine translation: Tricks of the trade," 2019, *arXiv:1902.07816*.

[36] N. Shazeer *et al.*, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," 2017, *arXiv:1701.06538*.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[38] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2016, *arXiv:1605.07146*.

[39] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Honolulu, HI, USA, Jul. 2017, pp. 4700–4708.

[40] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *Proc. 15th Int. Conf. Mach. Learn.*, San Francisco, CA, USA, Jul. 1998, pp. 144–151.

[41] W. W. Cohen, "Repeated incremental pruning to produce error reduction," in *Proc. Int. Conf. Mach. Learn.*, Tahoe City, CA, USA, Jun. 1995.

[42] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, "Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence," *IEEE Trans. Autom. Control*, vol. 42, no. 10, pp. 1482–1484, Oct. 1997.

[43] O. J. Dunn, "Multiple comparisons using rank sums," *Technometrics*, vol. 6, no. 3, pp. 241–252, 1964.

[44] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *J. Roy. Statist. Soc., Ser. B*, vol. 57, no. 1, pp. 289–300, 1995.

[45] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," in *Proc. Int. Conf. Learn. Representations*, Toulon, France, Apr. 2017.

[46] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018, *arXiv:1804.07612*.

Yuqi Cui received the B.E. degree in electronic information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2017. He is currently working toward the Ph.D. degree with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan.

His research interests include machine learning, fuzzy systems, and brain computer interfaces.

Dongrui Wu (Senior Member, IEEE) 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, Ministry of Education. His research interests include affective computing, brain computer interfaces, computational intelligence, and machine learning. He has more than 130 publications, including a book entitled *Perceptual Computing* (Wiley-IEEE Press, 2010).

Dr. Wu received the IEEE Computational Intelligence Society Outstanding PhD Dissertation Award in 2012, the IEEE TRANSACTIONS ON FUZZY SYSTEMS Outstanding Paper Award in 2014, NAFIPS Early Career Award in 2014, IEEE Systems, Man and Cybernetics (SMC) Society Early Career Award in 2017, and IEEE SMC Society Best Associate Editor Award in 2018. He was also a finalist of another three Best Paper Awards. He was/is an Associate Editor of the IEEE TRANSACTIONS ON FUZZY SYSTEMS (2011-2018), IEEE TRANSACTIONS ON HUMAN-MACHINE SYSTEMS (2014), IEEE COMPUTATIONAL INTELLIGENCE MAGAZINE (2017), and IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING (2019).

Jian Huang (Senior Member, IEEE) received the B.E., M.Eng., and Ph.D. degrees in automation from the 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, Department of Mechano-Informatics and Systems, Nagoya University, Japan. He was a Research Fellow with Nagoya University supported by JSPS invitation fellowship in 2015. 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 of France. His main research interests include rehabilitation robot, robotic assembly, networked control systems, and bioinformatics.

Dr. Huang has published more than 190 papers (including 23 IEEE Transactions articles and more than 50 IEEE conference papers). He is currently an Associate Editor of the IEEE TRANSACTIONS ON FUZZY SYSTEMS, an Editor of the Springer *ROBOMECH Journal*, and the Technical Editor of PLoS ONE. He has 14 authorized patents, and received the grand prize of Science and Technology Award of China General Chamber of Commerce and Gold Medal at Geneva Inventions in 2017.