Advances in Intelligent Systems and Computing 1138

Atulya K. Nagar
Kusum Deep
Jagdish Chand Bansal
Kedar Nath Das *Editors*

Soft Computing for Problem Solving 2019

Proceedings of SocProS 2019, Volume 1

Springer

Advances in Intelligent Systems and Computing

Volume 1138

The series "Advances in Intelligent Systems and Computing" contains publications on theory, applications, and design methods of Intelligent Systems and Intelligent Computing. Virtually all disciplines such as engineering, natural sciences, computer and information science, ICT, economics, business, e-commerce, environment, healthcare, life science are covered. The list of topics spans all the areas of modern intelligent systems and computing such as: computational intelligence, soft computing including neural networks, fuzzy systems, evolutionary computing and the fusion of these paradigms, social intelligence, ambient intelligence, computational neuroscience, artificial life, virtual worlds and society, cognitive science and systems, Perception and Vision, DNA and immune based systems, self-organizing and adaptive systems, e-Learning and teaching, human-centered and human-centric computing, recommender systems, intelligent control, robotics and mechatronics including human-machine teaming, knowledge-based paradigms, learning paradigms, machine ethics, intelligent data analysis, knowledge management, intelligent agents, intelligent decision making and support, intelligent network security, trust management, interactive entertainment, Web intelligence and multimedia.

The publications within "Advances in Intelligent Systems and Computing" are primarily proceedings of important conferences, symposia and congresses. They cover significant recent developments in the field, both of a foundational and applicable character. An important characteristic feature of the series is the short publication time and world-wide distribution. This permits a rapid and broad dissemination of research results.

**** Indexing: The books of this series are submitted to ISI Proceedings, EI-Compendex, DBLP, SCOPUS, Google Scholar and Springerlink ****

More information about this series at http://www.springer.com/series/11156

Atulya K. Nagar · Kusum Deep ·
Jagdish Chand Bansal · Kedar Nath Das
Editors

Soft Computing for Problem Solving 2019

Proceedings of SocProS 2019, Volume 1

Springer

Editors
Atulya K. Nagar
School of Mathematics, Computer Science and Engineering
Liverpool Hope University
Liverpool, UK

Jagdish Chand Bansal
Department of Mathematics
South Asian University
New Delhi, Delhi, India

Kusum Deep
Department of Mathematics
Indian Institute of Technology Roorkee
Roorkee, India

Kedar Nath Das
Department of Mathematics
National Institute of Technology Silchar
Silchar, India

ISSN 2194-5357 ISSN 2194-5365 (electronic)
Advances in Intelligent Systems and Computing
ISBN 978-981-15-3289-4 ISBN 978-981-15-3290-0 (eBook)
https://doi.org/10.1007/978-981-15-3290-0

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

We are delighted that the 9th International Conference on Soft Computing for Problem Solving, SocProS 2019, took place at Liverpool Hope University, UK, during 02–04 September 2019. This was a particularly important event as it coincided with the 175th anniversary of the foundation of Liverpool Hope University (LHU). The SocProS conference series has a glorious history; the earlier editions of the conference have been organised in various prestigious institutions of India. For the very first time, this event in the series of the conference was hosted outside of India at Liverpool in the UK. This is significant because along with IIT Roorkee and South Asian University in India, LHU has been one of the key institutions that initiated this prestigious meeting. Continuing the trend, once again the 9th edition of this conference touched many milestones in terms of quality research papers and fruitful discussions. The theme of SocProS 2019 was "*Unlocking the Power and Impact of Artificial Intelligence*".

This proceedings as an outcome of the 9th meeting of the SocProS community includes a collection of selected high-quality articles on various topics related to soft computing and artificial intelligence and their applications. The book is being prepared in two volumes to cover the recent advances and challenges in the themes of machine learning, neural networks, scientific computing, and intelligent systems and includes several chapters addressing the problems arising in real-life applications comprising that of image classification, deep learning, fuzzy systems, flow shop scheduling, support vector machines, mobile robot path planning, P-systems, machine learning, and spiking neural networks, to name a few contributions. We have also tried to capture the impact aspects of research in this area, particularly impact beyond the academic world. We have made further efforts in this direction to embed impact as part of our conference series, and going forward we very much hope that, as was agreed at the conference, we will continue to mainstream impact in our work and intensify our efforts to reach out to non-academic beneficiaries and users to realise the impact from our research.

Highlighting theoretical perspectives and empirical research, it is hoped that this edited volume will prove to be a comprehensive reference source for researchers, practitioners, students, and professionals interested in the current advancements and

efficient use of soft computing as well as in making the impact happen. We express our heartfelt gratitude to all the authors, reviewers, and Springer personnel for their motivation and patience.

Liverpool, UK
September 2019

Atulya K. Nagar
Kusum Deep
Jagdish Chand Bansal
Kedar Nath Das

Contents

About the Editors

Atulya K. Nagar holds the Foundation Chair as Professor of Mathematical Sciences, and is the Pro-Vice-Chancellor for Research and Dean of the Faculty of Science at Liverpool Hope University, United Kingdom. He is also the Head of the School of Mathematics, Computer Science and Engineering, which he established at the University. He is an internationally respected scholar working at the cutting edge of theoretical computer science, applied mathematical analysis, operations research, and systems engineering. He received a prestigious Commonwealth Fellowship to pursue his doctorate (DPhil) in Applied Nonlinear Mathematics, which he earned from the University of York (UK) in 1996. He holds a BSc (Hons), an MSc and MPhil (with distinction) in Mathematical Physics from the MDS University of Ajmer, India. His research expertise spans both applied mathematics and computational methods for nonlinear, complex, and intractable problems arising in science, engineering and industry.

Prof. Kusum Deep is a Professor at the Department of Mathematics, Indian Institute of Technology Roorkee. Her research interests include numerical optimization, nature inspired optimization, computational intelligence, genetic algorithms, parallel genetic algorithms, and parallel particle swarm optimization.

Dr. Jagdish Chand Bansal is an Assistant Professor at the South Asian University, New Delhi, India and visiting research fellow at Liverpool Hope University, UK. He has an excellent academic record and is a leading researcher in the field of swarm intelligence. He has published numerous research papers in respected international and national journals.

Dr. Kedar Nath Das is an Assistant Professor at the Department of Mathematics, National Institute of Technology, Silchar, Assam, India. Over the past 10 years, he has made substantial contributions to research on soft computing, and has published several papers in prominent national and international journals. His chief area of interest is evolutionary and bio-inspired algorithms for optimization.

Self-taught Learning: Image Classification Using Stacked Autoencoders

Upendra Pratap Singh, Swapnil Chavan, Sahil Hindwani and Krishna Pratap Singh

Abstract Availability of a large amount of unlabeled data and practical challenges associated with annotating datasets for different domains has led to the development of models that can obtain knowledge from one domain (called source domain) and use this knowledge in a similar domain (called the target domain). This forms the core of transfer learning. Self-taught learning is one of the popular paradigms of transfer learning frequently used for classification tasks. In a typical self-taught learning setting, we have a source domain having a large amount of unlabeled data instances and a target domain having limited labeled data instances. With this setting, self-taught learning proceeds as follows: Given ample unlabeled data instances in the source domain, we try to obtain their optimal representation. Basically, we are learning the transformation that maps unlabeled data instances to their optimal representation.The transformation learnt in the source domain is then used to transform target domain instances; the transformed target domain instances along with their corresponding labels are then used in supervised classification tasks. In our work, we have applied self-taught learning in image classification task. For this, we have used stacked autoencoders (for grayscale images) and convolutional autoencoders (for colored images) to obtain an optimal representation of the images present in the source domain. The transformation function learnt in the source domain is then used to transform target domain images. The transformed target domain instances along with their labels are then used for building the supervised classifier (in our case, SVM). Rigorous experiments on MNIST, CIFAR10 and CIFAR100 dataset

U. P. Singh (✉) · S. Chavan · S. Hindwani · K. P. Singh
Indian Institute of Information Technology Allahabad, Allahabad, India
e-mail: rsi2017001@iiita.ac.in

S. Chavan
e-mail: ise2016008@iiita.ac.in

S. Hindwani
e-mail: iit2014088@iiita.ac.in

K. P. Singh
e-mail: kpsingh@iiita.ac.in

A. K. Nagar et al. (eds.), *Soft Computing for Problem Solving 2019*,
Advances in Intelligent Systems and Computing 1138,
https://doi.org/10.1007/978-981-15-3290-0_1

show that our self-taught learning approach is doing well against the baseline model (where no transfer learning has been used) even when there are limited number of labeled data instances in the target domain.

Keywords Self-taught learning · Stacked autoencoders · Convolutional autoencoders · Representation learning · Unsupervised feature learning · Transfer learning

1 Introduction

Acquiring large quantities of labeled data is often difficult and expensive task [1]. To add to this, there are more number of classes (as well as fine-grained subclasses) that are being created dynamically. The above factors make it extremely challenging for efficient supervised classification tasks across multiple application domains. A number of methods for supervised classification have been proposed in the recent literature like [2–7] etc. These works dealt with different fields like image classification, object localization, segmentation, time series forecasting, etc. Though the researchers were able to obtain state-of-the-art results with their proposed algorithms and techniques, yet all these supervised classification tasks were limited by the availability of the labeled data in the domain they were working on.

The above limitations paved the way for transfer learning where given data in a related domain called the source domain, and it was possible to extract meaningful information and use it for carrying out the target domain task. A number of works have received attention in this direction [8–12]. In addition to this breakthrough, representational learning [13], otherwise known as *feature learning* is a fairly new concept that seeks to learn useful representations of the data as well as its *essence* for building effective classifiers. Recently, a number of efforts have been made toward representation learning using deep learning architectures [13–17].

Self-taught learning [1] is one of the most interesting paradigms of transfer learning based on representation learning. In this paradigm, given ample unlabeled instances in the source domain, we learn the optimal representations for these data instances and then use the learnt transformation to extract the features of scarce labeled instances present in the target domain. Once we get the useful representations for the labeled instances, we can use them for supervised classification tasks in the target domain. This optimal representation compensates for the limited availability of the labeled instances in the target domain. A number of works have been done on self-taught learning. For instance, Ref. [18] performed a comparative analysis of non-negative matrix factorization, principal component analysis and sparse coding methods to get the optimal representations for classification of music genres. Authors in [19] modeled two self-taught learning frameworks (independent component analysis based and convolutional autoencoder based) for classification of HSI images. In a slightly different direction, Ref. [20] attempts to solve weakly supervised object localization problem using self-taught learning. Recently, Ref. [21] used Isomap to

learn the representations in the source domain and then used the learnt transformations in the target domain to perform self-taught learning. In the field of medical diagnosis, Ref. [22] utilized the gas samples of formaldehyde, acetone, benzene, etc., to obtain the basis vectors and then used these basis vectors to construct new representations for the wound samples. To the best of our knowledge, this work is first to evaluate the performance of self-taught learning model against target domain dataset size. Limited work has been done using deep learning architectures for learning the representations. This motivated us to exploit the stacked architectures for self-taught learning. Our work differs from [22] in a way that we have used target domain training dataset size to evaluate the self-taught learning-based supervised classification task (labeled instances present in the target domain are scarce).

The rest of the paper is structured as follows: Sect. 2 provides the mathematical background for self-taught learning, autoencoders and related concepts. Section 3 describes the experimental details and setup. Section 4 presents or results along with its analysis. The work is concluded in Sect. 5 with Sect. 6 giving the future prospects of our work.

2 Mathematical Background

2.1 An Overview of Self-taught Learning

In self-taught learning [1], we have a target domain labeled dataset containing m examples $\{(\boldsymbol{x}_l^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}_l^{(m)}, y^{(m)})\}$ that are drawn under i.i.d assumption from some data distribution $\mathscr{D}$ where the subscript l means 'labeled' instances; $y^{(i)} \in \{1, 2, \ldots, C\}$ is the corresponding class label for each $\boldsymbol{x}_l^{(i)} \in \Re^n$. In addition to this, we are given a source domain having k unlabeled instances $\{\boldsymbol{x}_u^{(1)}, \boldsymbol{x}_u^{(2)}, \ldots, \boldsymbol{x}_u^{(k)}\} \in \Re^n$ that are not necessarily drawn from the same distribution $\mathscr{D}$ as that of the instances in the target domain, but which in some sense are 'related' to the target domain instances.

Given these labeled and unlabeled datasets, self-taught learning seeks to learn a hypothesis $h : \Re^n \rightarrow 1, 2, \ldots, C$ that mimics the input-label relationship contained in the labeled dataset; this hypothesis h is tested under the same distribution $\mathscr{D}$ that generated the labeled data.

2.2 Unsupervised Feature Learning Using Stacked Autoencoders

An autoencoder is a class of neural networks that seeks to copy essential aspects of its input to its output so as to learn useful representations for the input [1]. The network consists of an encoding function $f_e(\boldsymbol{x}_u)$ that gives the *code* (or the representation

$\boldsymbol{h}_u$) for the unlabeled input instance $\boldsymbol{x}_u$ and a decoding function $f_d(\boldsymbol{h}_u)$ that tries to reconstruct the input at the output layer given its *code* $\boldsymbol{h}_u$.

Mathematically, the *code* $\boldsymbol{h}$ and the reconstructed input for the unlabeled instance $\boldsymbol{x}_u$ can be expressed as

$$\boldsymbol{h}(\boldsymbol{x}_u) = f_e(\boldsymbol{W}_{eu}^T \boldsymbol{x}_u + \boldsymbol{b}_{eu}) \tag{1}$$

$$\hat{\boldsymbol{x}_u} = f_d(\boldsymbol{W}_{du}^T \boldsymbol{h} + \boldsymbol{b}_{du}) \tag{2}$$

where $\boldsymbol{W}_{eu}$ and $\boldsymbol{W}_{du}$ are the weight matrices connecting the input layer to the hidden layer and hidden layer to the output layer, respectively. $\boldsymbol{b}_{eu}$ and $\boldsymbol{b}_{du}$ are the corresponding bias vectors. The subscripts e and d denote that the matrices (and the bias vectors) are involved in encoding and decoding (reconstruction) of the input, respectively, while u in the subscript means that corresponding matrix/bias vector is obtained using unlabeled data. The encoding and decoding functions $f_e(.)$ and $f_d(.)$ are the activation functions. In our case, ReLU is used as an encoding function while sigmoid is used as the decoding activation function.

Given m unlabeled input instances, the objective function of the autoencoder can be put as follows:

$$\underset{\boldsymbol{W}_{eu}, \boldsymbol{W}_{du}, \boldsymbol{b}_{eu}, \boldsymbol{b}_{du}}{\operatorname{argmin}} \sum_{i=1}^{m} ||\boldsymbol{x}_u^i - \hat{\boldsymbol{x}}_u^i||^2 \tag{3}$$

Thus, the objective function penalizes the network parameters when the input $\boldsymbol{x}_u^i$ is different from its reconstructed version $\hat{\boldsymbol{x}}_u^i$. Once the learning is complete, we get the optimized values for weight matrices and bias vectors. Algorithm 1 summarizes the steps involved in unsupervised feature learning.

The above modeling holds for an autoencoder having only one hidden layer between input and output (called shallow autoencoders). For stacked autoencoders having multiple hidden layers, the *code* as well as the reconstructed input at any of the hidden layer can be obtained by using the corresponding weight matrices $\boldsymbol{W}_{eu}^{(K)}$, $\boldsymbol{W}_{du}^{(K)}$, bias vectors $\boldsymbol{b}_{eu}^{(K)}$, $\boldsymbol{b}_{du}^{(K)}$ and the inputs for the Kth hidden layer. Figure 1 shows the architecture of a shallow autoencoder with one hidden layer. Stacked autoencoder can be obtained from this architecture by plugging in multiple hidden layers between input and the output layer.

2.3 Feature Extraction from Labeled Data

We now seek to get the optimal representation (through process called feature extraction) for the labeled instances $\boldsymbol{x}_l^i$ present in the target domain. Remember that our target domain data consists of instance-class pairs $(\boldsymbol{x}_l^i, y^i)$. The optimal representation for our labeled instances is given by

$$\boldsymbol{h}(\boldsymbol{x}_l^i) = f_e(\boldsymbol{W}_{eu}^T \boldsymbol{x}_l^i + \boldsymbol{b}_{eu}) \tag{4}$$

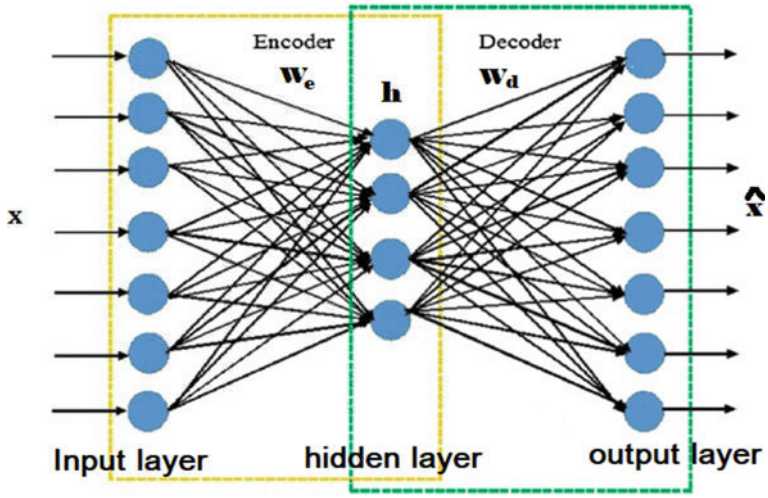

Fig. 1 Architecture of a shallow autoencoder having one hidden layer between input and the output layer. Note that this autoencoder is under complete [23] configuration as the number of neurons in the hidden layer is less than that of input layer

where $\boldsymbol{W}_{eu}$ and $\boldsymbol{b}_{eu}$ are the same weight matrix and bias vector that were obtained from unlabeled data (see the subscript u). In summary, we are using the parameters $(\boldsymbol{W}_{eu}, \boldsymbol{b}_{eu})$ obtained from the source domain (containing unlabeled data $\boldsymbol{x}_u$) and using them in our target domain to get the representation for the labeled data $\boldsymbol{x}_l$.

Thus, after feature extraction, our target domain data is modified from $(\boldsymbol{x}_l^i, y^i)$ to $(\boldsymbol{h}(\boldsymbol{x}_l^i), y^i)$. In our experiments with autoencoders, we have used different representations for the labeled instances (by changing the architecture of the autoencoder network). Each of these representations has different dimensionality (we have experimented with 32-, 64-, 128- and 256-dimensional representations). We then use the extracted representations for the target domain data in further supervised classification tasks.

2.4 Support Vector Machines

SVMs [24] are supervised classification models that learn a separating hyperplane such that the positive and negative class samples are as far away from the hyperplane as possible (maximum margin principle). Given the labeled data points $\{(\boldsymbol{x}_n, y_n)\}$, where $\boldsymbol{x}_n$ are the instances and y_n the corresponding labels, SVM assumes that there exists a hyperplane such that

$$\mathbf{w}^T \boldsymbol{x}_n + b \geq 1 \quad \forall\, y_n = +1 \tag{5a}$$

$$\mathbf{w}^T \boldsymbol{x}_n + b \leq -1 \quad \forall\, y_n = -1 \tag{5b}$$

Equivalently, the margin condition for all the samples becomes

$$y_n(\mathbf{w}^T \boldsymbol{x}_n + b) \geq 1 \quad \forall\, n \tag{6}$$

The margin on each side is given by

$$\gamma = \min_{1 \leq n \leq N} \frac{|\mathbf{w}^T x_n + b|}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||} \tag{7}$$

Given the training instances and their associated labels, SVM seeks to get the hyperplane $(\mathbf{w}, b)$ with largest possible margin. Since maximizing the margin $\gamma \propto \frac{1}{||\mathbf{w}||}$ is equivalent to minimizing $||\mathbf{w}||^2$, the objective function of hard-margin SVM is modeled as

$$\boxed{\min_{\mathbf{w},b} \; f(\mathbf{w}, b) = \frac{||\mathbf{w}||^2}{2} \text{ subject to } y_n(\mathbf{w}^T \boldsymbol{x}_n + b) \geq 1} \tag{8}$$

2.5 Supervised Classification Using SVM

The transformed training instances in the target domain $(\boldsymbol{h}(\boldsymbol{x}_l^i), y^i)$ are fed to the SVM for training and subsequent classification. The test data to assess the generalization capability of the SVM comes from the same distribution $\mathscr{D}$ that generated the labeled instances in the target domain [1]. A simple classifier like SVM is purposely chosen so that the classification results obtained could be attributed to the richness of the extracted features and not the classifier itself. Algorithm 2 summarizes the feature extraction as well as the SVM classification in the target domain.

2.6 Convolutional Autoencoders

Fully connected AEs (as well as DAEs) tend to ignore the spatial information present in an image. This is challenging in computer vision tasks as it introduces redundancy in the parameters, thereby forcing the features to get global. However, the recent practices in computer vision tasks try to discover recurring localized features in the input. Convolutional autoencoders [25] differ from conventional AEs as their weights are shared among all spatial locations in the input. This ensures that spatial locality in the input remains preserved. The reconstruction thus takes place as a result of the linear combination of basic image patches. Figure 2 shows the architecture of convolutional autoencoder with different layers shown in different colors and dimensions for easy interpretation.

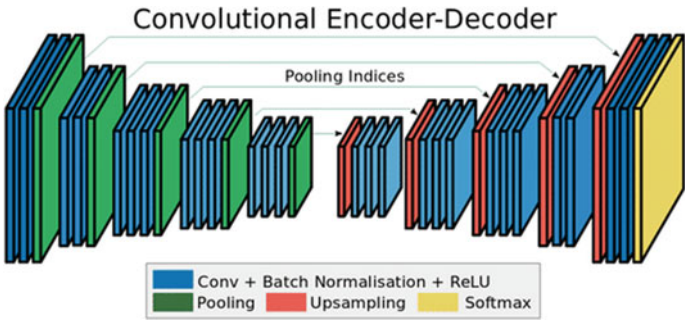

Fig. 2 Architecture of convolutional autoencoders. Different layers have been shown in different colors for interpretability of the architecture

Input : Source domain containing a set of unlabeled data instances $\boldsymbol{x}_u^i$
Output : Optimized values for the weight matrices $\boldsymbol{W}_{eu}^{(K)}$ and the bias vectors $\boldsymbol{b}_{eu}^{(K)}$
1. Randomly initialize weight matrices $\boldsymbol{W}_{eu}^{(K)}$, $\boldsymbol{W}_{du}^{(K)}$ and bias vectors $\boldsymbol{b}_{eu}^{(K)}$, $\boldsymbol{b}_{du}^{(K)}$
2. Train the autoencoder using the unlabeled input instances to get the optimized values for $\boldsymbol{W}_{eu}^{(K)}$ and $\boldsymbol{b}_{eu}^{(K)}$
3. Report the values obtained in the above step

Algorithm 1: Unsupervised Feature Learning Using Autoencoders

Figure 3 describes our proposed methodology. Stacked autoencoder is being used to learn the representations for the input which are then used as input for supervised classification task using SVMs.

Input : Set of labeled data $(\boldsymbol{x}_l^i, y^i)$ in the target domain and optimized values for weight matrices $\boldsymbol{W}_{eu}^{(K)}$ and bias vectors $\boldsymbol{b}_{eu}^{(K)}$ obtained from unsupervised feature learning
Output : Class membership c of the test input $\boldsymbol{x}_t$ in the target domain
1. Use the optimized values of $\boldsymbol{W}_{eu}^{(K)}$ and $\boldsymbol{b}_{eu}^{(K)}$ to get the *code* for the input instances $\boldsymbol{x}_l^i$. Mark these *codes* as $\boldsymbol{h}(\boldsymbol{x}_l^i)$
2. Transform the target domain data to these *code*-label pairs $(\boldsymbol{h}(\boldsymbol{x}_l^i), y^i)$.
3. Use the transformed target domain data for training the SVM
4. Obtain the *code* for the test instance $\boldsymbol{x}_t$ using the same weight matrices and bias vectors $\boldsymbol{W}_{eu}^{(K)}$, $\boldsymbol{b}_{eu}^{(K)}$. Mark this *code* as $\boldsymbol{h}(\boldsymbol{x_t})$
5. Use the trained SVM to classify $\boldsymbol{h}(\boldsymbol{x_t})$ into one of the C classes

Algorithm 2: SVM training and subsequent classification using the transformed input features in the target domain

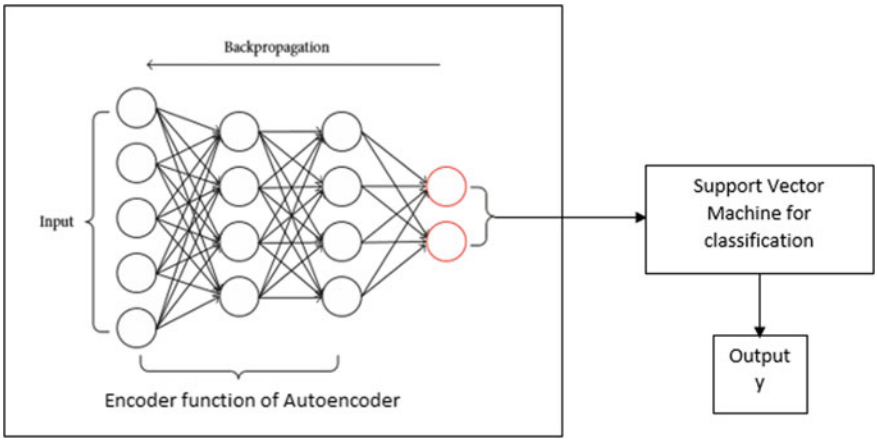

Fig. 3 Block diagram showing our proposed methodology. The features extracted from the autoencoder (using weight matrices and bias vector) along with their corresponding labels are fed to the SVM for supervised classification. The output y represents the class to which the target domain input instance belongs to

3 Experiments

3.1 Experimental Setup

The experiments have been performed on Intel(R) Core(TM) $i7-6700$ CPU operating at 2×3.40 GHz with 16GB of memory space on Windows 10 Pro operating system. All implementations are done in Python using appropriate machine/deep learning libraries as and when required.

3.2 Dataset Description

MNIST [26] is a highly preferred dataset among the researchers for use in various pattern recognition tasks. This dataset is a subset of larger dataset available from NIST [27] and consists of 60,000 training and 10,000 testing image samples of size-normalized handwritten digits. The dimension of each image is 28×28 pixels, i.e., a total of $28 \times 28 = 784$ pixel values per image.

The CIFAR-10 dataset [28] consists of 60,000 colored images each of dimension 32×32 spanning ten classes. The training dataset is divided into five training batches containing 10,000 images each (i.e., we have 50,000 training samples). The test dataset contains 10,000 images samples. CIFAR100, on the other hand, has 60,000 image samples spanning 100 classes. Training dataset contains 50,000 images samples, while remaining 10,000 samples form the test dataset. In addi-

tion, the 100 classes are grouped into 20 superclasses. Each image comes with a 'fine' label (class-membership) as well as a 'coarse' label (superclass-membership). Although the datasets above are annotated in their original form, for the purpose of unsupervised feature learning, we have discarded the label information.

3.3 Neural Network Configuration for Autoencoder

Input data is first normalized in the range [0, 1] so that cross-entropy loss function can be used for training the autoencoder [29]. RMSProp algorithm has been used to optimize the loss function. The encoder uses *ReLU* activation, while the decoder uses *Sigmoid* activation. Autoencoder is configured to run for 50 epochs with a batch size equal to 256. The deep autoencoder used in our experiments has six hidden layers between input and the output layer (again ReLU is used for encoding layers while Sigmoid is used in the decode layers).

3.4 Using Conventional SVM as a Baseline Model

In order to assess how well our self-taught learning approach is working on different representations for the input, we have compared the performance of these self-taught learning models with a baseline SVM trained on original 784 features (size of MNIST image is 28×28). The performance of this baseline model (baseline, since it does not exploit transfer learning) can be seen in Figs. 4 and 5. The penalty parameter C for this baseline SVM is set to 1.0, while the tolerance value for its RBF kernel is 0.001.

3.5 Convolutional Autoencoder Configuration

The input is first normalized in the range [0, 1] before it is fed to the convolutional autoencoder. CAE minimizes the cross-entropy loss function using RMRProp algorithm. We use three convolutional layers followed by maxpooling layer as the encoder, and after getting the latent representation (the code), we use three deconvolutional layers followed by upscaling to get the reconstructed image. The number of filters in each convolutional layer is 100 each of size 3×3. These values for the hyperparameters have been chosen empirically taking SVM classification accuracy as the metrics. The representation obtained at the encoder is of 1600 dimensions. ReLU activation function is used for the convolutional layers, while the last layer of the decoder uses a sigmoid function. Our network is trained to run for 10 epochs with a batch size of 128.

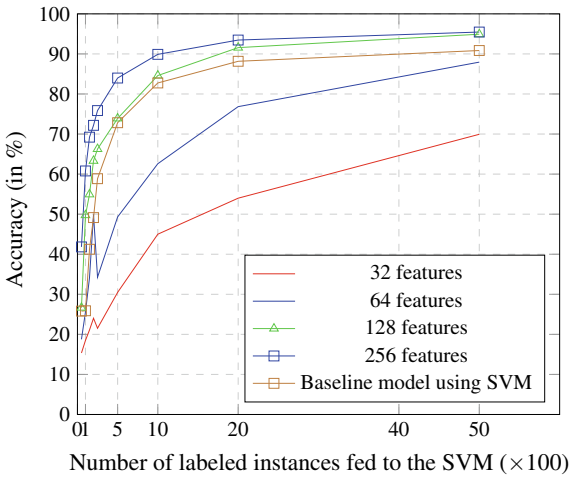

Fig. 4 Variation of SVM classification accuracy with respect to the size of labeled training dataset (on different feature representations obtained from *shallow* autoencoders) for MNIST dataset

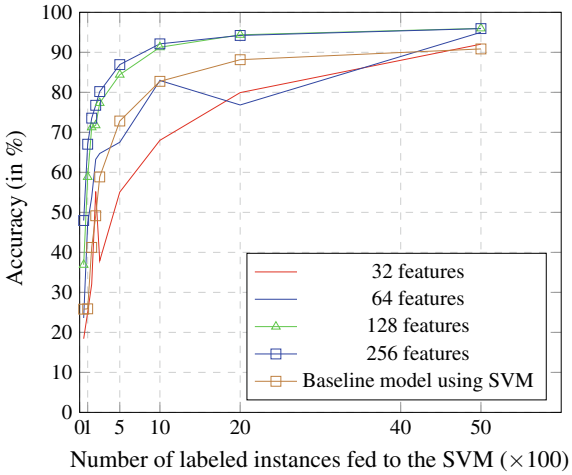

Fig. 5 Variation of SVM classification accuracy with respect to the size of labeled training dataset (on different feature representations obtained from *deep* autoencoders) for MNIST dataset

To fully exploit the self-taught learning paradigm, we proceeded as follows: We first trained our model on CIFAR10 dataset (source domain dataset) to get the representations, and then we applied the obtained transformation on the CIFAR100 dataset (target domain dataset). As a result of this transformation, CIFAR100 instances got transformed. We used these transformed CIFAR100 instances and corresponding labels for SVM training and testing. Next, we went in the opposite direction, i.e., we learnt the representations using CIFAR100 dataset and used the learnt transformation on CIFAR10 so that the transformed instances and labels of the latter could be fed to the SVM for training and classification.

4 Results and Discussion

Figure 4 shows the variation of SVM classification accuracy with respect to the size of training dataset used for its training. The different plots are for different representations learnt from a shallow autoencoder. Along with this, we have also provided the performance of the baseline SVM model that used the original 784-dimensional features (instead of learnt features). As can be seen from the plots, the accuracy of the SVM increases with an increase in the size of its training set. This is well expected as with increasing training set, the model is able to avoid underfit and in the process performs better on the test dataset. One more observation from this graph is that the lower-dimensional representations of the data (32- and 64-dimensional) could not keep up with the baseline. This is justified by the fact that when 784 dimensions are being squashed into 32 or 64 dimensions, there is a substantial information loss leading to a fairly underfit model. On the other hand, representations of lengths 128 and above (in our case 256) were able to capture the essence of the data very well. As such, these models showed good classification performance on the test dataset. Similarly, Fig. 5 shows the performance of SVM classification accuracy when the representations have been learnt using deep autoencoders. Similar trends were observed with the exception that the classification performance was better compared to what we had in the shallow autoencoders for all the representations except the baseline (baseline has been trained on the original 784 dimensions).

One notable point from the above two plots is that with increased training dataset size, the classification performance starts to *saturate*. This could be explained by the fact that up to a certain training size, the training data is helping the model to uncover useful patterns. Beyond this, the increased size seems to have little effect on the performance since by then the model is already aware of the *essence* of the data.

Figure 6 shows the performance of SVM classification when the features were learnt using CIFAR100 dataset (source domain data) and the transformed instances of CIFAR10 dataset (target domain data) were used for SVM Training And Subsequent classification. The classification accuracy of the SVM (on the test data) has been plotted against the size of training dataset fed to the SVM. The training instances were chosen randomly from all the training classes. We observe a similar trend as in Figs. 4 and 5. Initially, we find a sharp increase in classification accuracy, and then this sharpness decreases in value and the model begins to saturate. Likewise, Fig. 7 shows the performance in the reverse direction (source and target domain are interchanged). The learnt transformations on CIFAR10 dataset (now, the source domain) were used to transform CIFAR100 data (target domain data) instances, and then the latter was used for SVM experiments. The performance that we got from our SVM model was on the lower side. This could be attributed to the fact that CIFAR100 is a fine-grained dataset as against CIFAR10 (which is coarse-grained dataset). As a result, finer details required to separate two CIFAR100 classes may not have been acquired when we learnt the features using CIFAR10 dataset. Nevertheless, the results hold great prospect given a sophisticated model and sufficient computational resources.

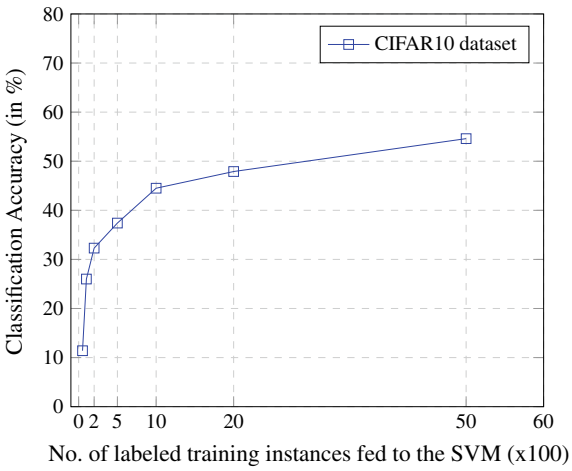

Fig. 6 Variation of SVM classification accuracy with respect to the size of labeled training dataset (for CIFAR10 dataset) using convolutional autoencoder

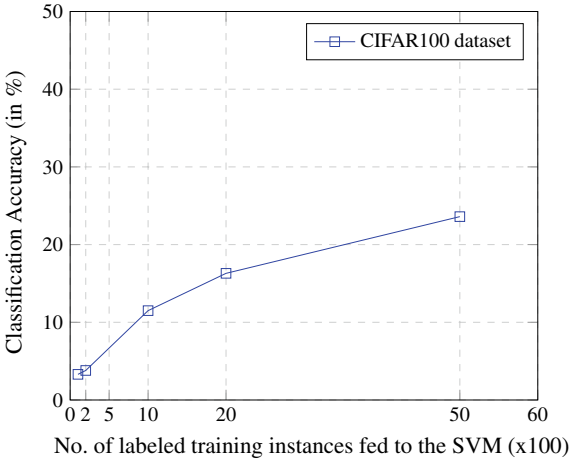

Fig. 7 Variation of SVM classification accuracy with respect to the size of labeled training dataset (for CIFAR100 dataset) using convolutional autoencoder

The experiments based on convolutional autoencoder have used 1600-dimensional representations in both the case.

In our experiments, we have used the entire test dataset (as provided by the original datasets) for evaluating the SVM accuracy.

5 Conclusion

In this work, we implemented self-taught learning for classification of grayscale handwritten digits as well as colored images. For classification of handwritten digits, we have trained a stacked autoencoder neural network to learn the latent

representation of source domain unlabeled instances. Once the representations are learnt, we then use the same network parameters to extract the features for labeled instances present in the target domain. The transformed labeled instances along with their corresponding labels are then fed to the SVM for training and subsequent classification. We follow a similar procedure in case of colored images with the features getting learnt using a convolutional autoencoder network instead of stacked autoencoders. Results obtained on benchmark datasets show promising prospects.

6 Future Scope

In our experiments, the network parameters (i.e., the transformation) that were learnt using unlabeled data in the source domain were used directly for feature extraction in the target domain. However, there is a need to adapt these parameters when they are being transferred from source to target domain. This adaptation remains a part of our future work. The adapted self-taught learning models will then be compared with a series of baseline models like the ones based on variational stacked autoencoders, de-noising, contractive autoencoders, autoencoder SVMs, etc.

References

1. R. Rajat, B. Alexis, L. Honglak, P. Benjamin, Y.N. Andrew, Self-taught learning: transfer learning from unlabeled data, in *Proceedings of the 24th International Conference on Machine Learning*, vol. 8 (2007), pp. 759–766
2. M. Lei, L. Manchun, M. Xiaoxue, C. Liang, D. Peijun, L. Yongxue, A review of supervised object-based land-cover image classification. ISPRS J. Photogram. Remote Sens. **130**, 277–293 (2017)
3. D. Thibaut, M. Taylor, T. Nicolas, C. Matthieu, Wildcat: weakly supervised learning of deep convnets for image classification, pointwise localization and segmentation, in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2 (2017)
4. G. Jie, W. Hongyu, F. Jianchao, M. Xiaorui, Deep supervised and contractive neural network for SAR image classification. IEEE Trans. Geosci. Remote Sens. **55**(4), 2442–2459 (2017)
5. B.J. Kyle, M.O. Hakeem, Generation of a supervised classification algorithm for time-series variable stars with an application to the LINEAR dataset. New Astron. **52**, 35–47 (2017)
6. L. Peng, R.C. Kim-Kwang, W. Lizhe, H. Fang, SVM or deep learning? A comparative study on remote sensing image classification. Soft Comput. **21**(23), 7053–7065 (2017)
7. E. Tuba, N. Bacanin, An algorithm for handwritten digit recognition using projection histograms and SVM classifier (2015)
8. T. Chuanqi, S. Fuchun, K. Tao, Z. Wenchang, Y. Chao, L. Chunfang, A survey on deep transfer learning (2018)
9. Y. Zhilin, S. Ruslan, W.C. William, Transfer learning for sequence tagging with hierarchical recurrent networks (2017)
10. W. Karl, M.K. Taghi, D.D. Wang, A survey of transfer learning. J. Big Data **3**(1), 9 (2016)
11. L. Shao, F. Zhu, X. Li, Transfer learning for visual categorization: a survey. IEEE Trans. Neural Netw. Learn. Syst. **26**(5), 1019–1034 (2015)
12. L. Jie, B. Vahid, H. Peng, Z. Hua, X. Shan, Z. Guangquan, Transfer learning using computational intelligence: a survey. Knowl.-Based Syst. **80**, 14–23 (2015)

13. B. Yoshua, C. Aaron, V. Pascal, Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**(8), 1798–1828 (2013)
14. K.L.L. Chamara, Z. Hongming, H. Guang-Bin, V. Chi Man, Representational learning with extreme learning machine for big data. IEEE Intell. Syst. **28**(6), 31–34 (2013)
15. S. Dinggang, W. Guorong, S. Heung-Il, Deep learning in medical image analysis. Annu. Rev. Biomed. Eng. **19**, 221–248 (2017)
16. C. Diego, T. Isabel, A.F. Manuel, B. Søren, M.G.I. José, Deep feature learning for virus detection using a Convolutional Neural Network (2017)
17. J. Luyang, Z. Ming, L. Pin, X. Xiaoqiang, A convolutional neural network based feature learning and fault diagnosis method for the condition monitoring of gearbox. Measurement **111**, 1–10 (2017)
18. M. Konstantin, M. Tomoko, High level feature extraction for the self-taught learning algorithm. EURASIP J. Audio Speech Music Process. **1**, 6 (2013). https://doi.org/10.1186/1687-4722-2013-6
19. K. Ronald, K. Christopher, Self-taught feature learning for hyperspectral image classification. IEEE Trans. Geosci. Remote Sens. **55**(5), 2693–2705 (2017)
20. J. Zequn, W. Yunchao, J. Xiaojie, F. Jiashi, L. Wei, Deep self-taught learning for weakly supervised object localization (2017)
21. Y. Sethi, V. Jain, K.P. Singh, M. Ojha, Isomap based self-taught transfer learning for image classification
22. H. Peilin, J. Pengfei, Q. Siqi, D. Shukai, Self-taught learning based on sparse autoencoder for E-nose in wound infection detection. Sensors **17**(10), 2279 (2017)
23. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (The MIT Press, Cambridge, 2016)
24. J.S. Alex, S. Bernhard, A tutorial on support vector regression. Stat. Comput. **14**(3), 199–222 (2004)
25. X. Guo, X. Liu, E. Zhu, J. Yin, Deep clustering with convolutional autoencoders, in *International Conference on Neural Information Processing* (2017), pp. 373–382
26. F. Mazdak, MNIST handwritten digits (2014)
27. D. Li, The MNIST database of handwritten digit images for machine learning research [best of the web]. IEEE Sig. Process. Mag. **29**(6), 141–142 (2012)
28. K. Alex, H. Geoffrey, Learning multiple layers of features from tiny images (2009)
29. Q. Yu, W. Yueming, Z. Xiaoxiang, W. Zhaohui, Robust feature learning by stacked autoencoder with maximum correntropy criterion, in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2014), pp. 6716-6720

Performance Analysis of Whale Optimization Algorithm Based on Strategy Parameter

Amarjeet Singh and Kusum Deep

Abstract The performance of heuristic algorithms highly depends on the parameter values of the algorithms. Whale optimization algorithm (WOA) is a newly developed heuristic algorithm which has strategy parameter a that decreases linearly from 2 to 0 as iteration increases. In this paper, two algorithms, modified whale optimization algorithm-1 (MWOA-1) and modified whale optimization algorithm-2 (MWOA-2), have been proposed based on the variation of the strategy parameter a. The experiments are performed on a set of 23 benchmark problems. Results are compared with original WOA, gravitational search algorithm and grasshopper optimization algorithm. Based on the analysis of results, it is concluded that the overall performance of MWOA-1 and MWOA-2 is better than others on scalable unimodal function with dim = 30, scalable multimodal functions with dim = 30 and low-dimensional multimodal functions.

Keywords Whale optimization algorithm · Heuristic algorithm · Numerical optimization · Grasshopper optimization algorithm

1 Introduction

Optimization is a process to find the best values of the decision variables for a particular problem in which objective function is to be minimized or maximized. The vast majority of the scientific, engineering and real-life problems can be formulated as nonlinear continuous optimization problems. From the most recent couple of decades,

A. Singh (✉)
Department of Mathematics, Janki Devi Memorial College, Sir Ganga Ram Hospital Marg, New Delhi, Delhi 110060, India
e-mail: amarjeetiitr@gmail.com

K. Deep
Department of Mathematics, Indian Institute of Technology Roorkee, Roorkee 247667, Uttarakhand, India
e-mail: kusumfma@iitr.ac.in

A. K. Nagar et al. (eds.), *Soft Computing for Problem Solving 2019*,
Advances in Intelligent Systems and Computing 1138,
https://doi.org/10.1007/978-981-15-3290-0_2

nature-inspired algorithms (NIAs) are becoming more and more popular in engineering applications because they are based on simple idea and easy to implement. The beauty of these algorithms is that they bypass local optima. Nature-inspired optimization algorithms mimic some biological or physical phenomena. Genetic algorithm (GA) [1], biogeography-based optimization [2], gravitational search algorithms [3–5], particle swarm optimization [6], ant colony optimization [7], etc., are the popular algorithms.

The literature to solve optimization problems is very rich, but there is no single method which can solve all the problems [8]. Therefore, new algorithms and existing algorithms are being developed with the hope that they have some advantage over existing ones. Whale optimization algorithm (WOA) [9] is a newly developed algorithm. It has been tested on various optimization problems with different difficulty levels but like other heuristic algorithm WOA suffers with weak local search ability in complex problems. Kaveh and Ghazaan [10] proposed enhanced whale optimization algorithm to minimize the weight of skeletal structures. Based on whale optimization algorithm, Cherukuri and Rayapudi [11] proposed a novel maximum power point tracking to analyze analytic modeling of PV system considering both series and shunt resistances for maximum power point tracking under partial shaded condition. Aljarah et al. [12] used whale optimization algorithm to train neural networks. Bhesdadiya et al. [13] employed whale optimization algorithm to solve the optimal power flow problem. Yan et al. [14] proposed ameliorative whale optimization algorithm to solve multi-objective water resource allocation optimization models. Nasiri and Khiyabani [15] proposed whale clustering optimization algorithm for clustering. Kaur and Arora [16] proposed chaotic whale optimization algorithm. Jadhav and Gomathi [17] proposed a technique for data clustering using whale optimization algorithm. Elaziz and Oliva [18] used opposition-based learning to enhance the exploration phase of whale optimization algorithm and used it to estimate the parameters of solar cells using three different diode models. The more literature may be studied in [19, 20]. In this paper, two modified whale optimization algorithms have been proposed based on the variation of its strategy parameter.

The remaining paper is composed as follows: In Sect. 2, the whale optimization algorithm is explained. In Sect. 3, modified whale optimization algorithm is proposed. In Sect. 4, the numerical results are analyzed. Finally in Sect. 5, the conclusions are drawn.

2 Whale Optimization Algorithm

Whale optimization algorithm (WOA) is proposed by Mirjalili and Lewis in 2016 [9]. It is inspired from the foraging behavior of humpback whales. The foraging behavior of humpback whales includes encircling prey, bubble-net attacking method and searching of prey.

In WOA, first a random population of size N is initialized. The parameter a is decreased linearly from 2 to 0 in order to provide exploration and exploitation,

respectively. Then, $\vec{A}$ is evaluated which depends on a and random vector $\vec{r}$.

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a}, \ \ \vec{C} = 2\vec{r} \tag{1}$$

The position of the search agents is updated with respect to a randomly chosen search agent when $\text{Abs}(\vec{A}) \geq 1$ or the best solution obtained so far when $\text{Abs}(\vec{A}) < 1$. Search agents follow either a spiral or circular movement with the probability $p = 0.50$.

$$\vec{X}(t+1) = \begin{cases} \text{if } p < 0.5, & \begin{cases} \text{if } \text{Abs}(\vec{A}) < 1, & \overrightarrow{X}_{\text{best}}(t) - \vec{A} \cdot \text{Abs}(\vec{C} \cdot \overrightarrow{X}_{\text{best}}(t) - \vec{X}(t)) \\ \text{if } \text{Abs}(\vec{A}) \geq 1, & \overrightarrow{X}_{\text{rnd}}(t) - \vec{A} \cdot \text{Abs}(\vec{C} \cdot \overrightarrow{X}_{\text{rnd}}(t) - \overrightarrow{X}(t)) \end{cases} \\ \text{if } p \geq 0.5, & \text{Abs}(\overrightarrow{X}_{\text{best}}(t) - \vec{X}(t)) \cdot e^{\text{bl}} \cdot \cos(2\pi l) + \overrightarrow{X}_{\text{best}}(t) \end{cases} \tag{2}$$

where t represents the current iteration, $\overrightarrow{X}_{\text{best}}(t)$ is global best solution, $\vec{X}$ is position vector, $\vec{X}_{\text{rnd}}(t)$ is a random position vector chosen from the current population, p is a random number in $[0, 1]$, $\vec{r}$ is a random vector in $[0, 1]$, l is a randomly distributed number in $[-1, 1]$ and b is a constant which controls the shape of the logarithmic spiral movement of humpback whales in a spiral equation. $\vec{A}$ and $\vec{C}$ are coefficient vectors, and $\vec{a}$ decreases linearly from 2 to 0 over the course of iterations.

WOA terminates when it satisfies termination criterion. The pseudo-code of the WOA is given in Fig. 1.

3 Proposed Algorithm

The performance of the heuristic algorithms highly depends on their parameter value. The behavior of the algorithm could be modified by changing the value of one or more of its parameters. The algorithm may perform best at a specific value of parameters and may perform relatively worse at some other values of parameters. The issue of setting the values of various parameters of an algorithm is crucial for good performance. Selecting such parameter value that yields good performance is a long-standing grand challenge of the field and is a subject of research.

Setting of parameter values can be studied into two major forms [21]: parameter tuning and parameter control. Parameter tuning is a common practice approach that amounts to search best values for the parameters before the run of the algorithm. The algorithm utilizes these values at the running time, which remain fixed during the run. Parameter control forms an alternative, as it amounts to starting a run with initial parameter values that are changed during the run. Parameter tuning is a typical approach to design an algorithm. Such tuning is done by experimenting with different values and selecting the ones that give the best results on the test problems at hand. However, it is a very time-consuming activity.

Set number of particles = N
Set dimension of the problem = m
Initialize the whale population $X_i\left(i=1,2,...,N\right)$
Calculate the fitness of each search agent
$\vec{X}_{best}(t)$ =the best search agent
while (t < maximum number of iterations)
  for each search agent
  Update a, A, C, l and p
    if1 ($p<0.5$)
      if2 ($|A|<1$)
        Update the position of current search agent by
        $\vec{X}(t+1)=\vec{X}_{best}(t)-\vec{A}\cdot Abs(\vec{C}\cdot\vec{X}_{best}(t)-\vec{X}(t))$
      elseif2 ($|A|\geq 1$)
        Select a random search agent ($\vec{X}_{rnd}(t)$)
        Update the position of current search agent by
        $\vec{X}(t+1)=\vec{X}_{rnd}(t)-\vec{A}\cdot Abs(\vec{C}\cdot\vec{X}_{rnd}(t)-\vec{X}(t))$
      endif2
    elseif1 ($p\geq 0.5$)
      Update the position of current search agent by
      $\vec{X}(t+1)=Abs(\vec{X}_{best}(t)-\vec{X}(t))\cdot e^{bl}\cdot\cos(2\pi l)+\vec{X}_{best}(t)$
    endif1
  endfor
Check if any search agent goes beyond search space and amend it
Calculate the fitness of each search agent
Update $\vec{X}_{best}$ if there is a better solution
t=t+1
endwhile

Fig. 1 Pseudocode of WOA

Based on the experimentations, the following are the technical drawbacks of parameter tuning:

- It is impossible to try all different combinations of parameters practically.
- The process of parameter tuning is a time-consuming process.
- The chosen parameters for a given problem may not be necessarily optimal for other problem.

Methods based on variation in parameter (i.e., parameter control) values can be classified into three categories.

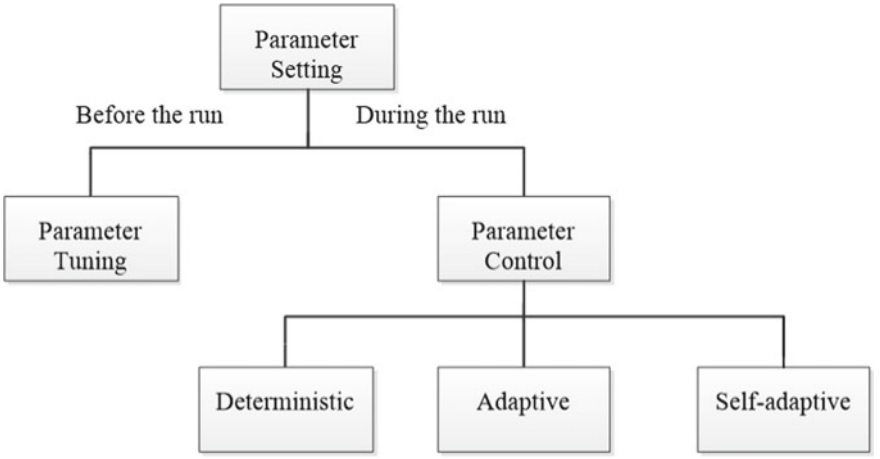

Fig. 2 Classification of parameter setting

Deterministic parameter control: In this method, the value of a strategy parameter is changed by some deterministic law. It does not use any feedback from search, and the strategy parameter is modified in a fixed, predetermined way.

Adaptive parameter control: It takes some feedback from the search as an input that is used to determine the direction or magnitude of the change to the strategy parameter.

Self-adaptive parameter control: This method uses the idea of the evolution to implement the self-adaption of parameters.

The classification of the parameter setting can be depicted from Fig. 2. The basic difference between parameter tuning and parameter control is that in parameter tuning, first best suited parameter value(s) is(are) decided by conducting several experiments and then it keeps fixed in final experiment but in parameter controlling, the parameter value may vary in each iteration.

Whale optimization algorithm is a nature-inspired algorithm. The working procedure of WOA has been described in Sect. 2. WOA has strategy parameter a. In WOA, strategy parameter a decreases linearly from 2 to zero by the equation

$$a = 2 - \frac{2}{\text{max\_iter}} \times t \tag{3}$$

where t is current iteration and max_iter is maximum iteration to run the algorithm. Clearly, the parameter a is decided by deterministic parameter control method.

In the present study, the behavior of WOA has been studied, when a decreases exponentially by the equation.

$$a = a_1 \times \exp\left(-a_2 \times \frac{t}{\text{max\_iter}}\right) \tag{4}$$