Int J Syst Assur Eng Manag (August 2022) 13(4):1637–1651
https://doi.org/10.1007/s13198-021-01519-8

ORIGINAL ARTICLE

Incorporating whale optimization algorithm with deep belief network for software development effort estimation

Anupama Kaushik[1] · **Niyati Singal**[1] · **Malvika Prasad**[1]

Received: 21 August 2019 / Revised: 3 November 2021 / Accepted: 8 November 2021 / Published online: 4 January 2022

Abstract The software industry is highly competitive, and hence, it is imperative to have an accurate method to estimate the effort needed in the key phases of software development. Accurate estimates ensure efficient allocation of human and machine resources for the project. This paper proposes a technique for software development effort estimation using deep belief network (DBN). For fine-tuning of DBN, Whale Optimization Algorithm (WOA) is used which mimics the social behaviour of humpback whales. The proposed technique DBN-WOA has been experimentally evaluated on four promise datasets—COCOMO81, NASA93, MAXWELL and CHINA. The results from DBN-WOA are compared with the results from fine-tuning of DBN with backpropagation (DBN-BP) and it is observed that the proposed technique outscores DBN-BP. The proposed approach is also empirically validated through a statistical framework.

Keywords Software development effort estimation · Deep belief network · Backpropagation · Whale optimization algorithm · Metaheuristics · Neural networks

1 Introduction

The software typically developed has a "compilation phase" without which the software is not deemed to be complete. It may not be efficient, reliable, maintainable or scalable in absence of resources. In the juggernaut of IT revolutions in this world, Software Development Effort Estimation (SDEE) is a research area that is undergoing intense study as it assists in the production of an error-free software product which adheres to the requirements of the customers (Gupta et al. 2014). A software failure, in the present world of IT, is extremely perilous and can sometimes prove to be fatal as well. In the past, many software projects have failed such as Y2K (2000); Mariner Bugs Out (1962); Hartford Coliseum Collapse (1978); (http://www.devtopics.com/20-famous-software-disasters/). Some of the prominent causes discovered were inaccurate estimates of needed resources, badly defined system requirements, unmanaged risks, poor project management etc. Hence, it seems apparent to be more systematic and organised while producing software or on the whole to devote the accurate time, effort and cost required for producing software.

SDEE helps in realizing these goals of producing quintessential software. It involves estimating the effort (or manpower) required for producing software, prior to its development phase. Since effort is the main driver for producing software, its estimation will ultimately lead to the estimations of the cost, time and staffing levels required to complete the project. Hence, the accurate estimates of software development effort are imperative as they are responsible for the entire framework and success of the software development process.

SDEE is usually done when most of the attributes of the software details are not divulged. Therefore, it has always been a very challenging and daunting task. Since the 1980s, many SDEE models have been proposed in literature (Jørgensen and Sheppard 2007; Elish 2009; Benala et al. 2012). But no model is perfect or has proved to be 100% accurate.

✉ Anupama Kaushik
anupama@msit.in

[1] Department of IT, Maharaja Surajmal Institute of Technology, New Delhi, India

Traditionally, expert judgment was used for SDEE. It involved consulting an expert, who had a lot of experience in developing software (Srivastava et al. 2020). This failed when no experts were available to consult. Moreover, it was completely intuitive. Later, algorithmic models were developed such as Constructive Cost Model (COCOMO) (Boehm 1984; Fenton and Pfleeger, 1997; Pressman 1997), Putnam Resource Allocation (Putnam 1978) etc. These belonged to linear-least-squares regression. The equations in these models were constructed after studying the datasets of previous software projects. Their input was usually software size and/or function points. But these models could not deal with non-linear relations across the characteristics of project and effort (Gray 1999).

Recently, various authors have been estimating software development effort using non-algorithmic models.

These non-algorithmic models do not rely upon one fixed formula, it allows experts intervention to adjust the model used in software cost prediction. They have the capability to work even on imprecise, noisy and vague data and still produce suitable results.

Non-algorithmic models work well in SDEE as at the early stages of software development, the attributes of the software are not known or are imprecise, vague and noisy. Two widely known techniques that can be classified as non-algorithmic techniques are machine learning techniques and metaheuristic techniques. The machine learning techniques that have been used by some authors in the domain of SDEE, include Artificial Neural Network (Kumar et al. 2008), Genetic Algorithms (Burgess and Lefley 2001), Multiple Additive Regression trees (Elish 2009), linear regression models (Jiang et al. 2007; Xia et al. 2008) etc. Many researchers also incline towards using metaheuristic techniques in various domains for solving complex problems. These can also be used in affiliation with the machine learning techniques for solving problems. Some of the metaheuristic techniques that have been used by some authors in the domain of SDEE include Firefly Algorithm (Kaushik et al. 2016), COA-Cuckoo optimization (Kaushik et al. 2017) and ALO-Ant Lion Optimization (Kaushik et al. 2020). The other domains of software engineering where these metaheuristic techniques are used are software reliability (Sharma et al. 2011; Sharma and Pant 2014) and Software Project Scheduling (SPS) Problem (Sharma 2016).

Deep learning is another domain that is gaining importance these days. It outperforms previously existing techniques in the multiple domains, such as image processing (Ciregan et al. 2012), optical character recognition (Breuel et al. 2013), text-to-speech synthesis (Fan et al. 2014) etc. The contribution of this research is to predict software effort using integration of DBN, which is a model of deep learning, and WOA, which is a metaheuristic technique. This paper attempts to evaluate the success of deep learning for SDEE. The paper further also attempts to judge the effectiveness of integrating metaheuristic technique, WOA with DBN as compared to integrating backpropagation with DBN. This research attempts to answer the following questions:

1. Can deep learning accurately predict the software development effort?
2. Can the fine tuning of DBN using Whale Optimization Algorithm (WOA) perform better than the fine tuning of the DBN using backpropagation?

The remainder of the paper is organised as follows: Sect. 2 summarises the related work; Sect. 3 discusses the overview of techniques employed in the paper; Sect. 4 describes the proposed methodology of SDEE; Sect. 5 describes the experimental evaluation and results; Sect. 6 discusses the statistical validation; Sect. 7 lists the limitations and future scope and Sect. 8 concludes the paper and answers the research questions.

2 Related research

Realizing the importance of Software Development Effort Estimation, researchers over the past few decades have developed a number of techniques, which have been discussed here.

Muzaffar and Ahmed (2010) presented a study which showed that accuracy of effort prediction of a fuzzy logic-based system (FLS) is largely dependent on the architecture of the system, the relative parameters, and the training algorithms. They concluded that the steepest descent algorithm is a better training algorithm than heuristic based algorithm on FLS based effort prediction.

Qin and Fang (2011) listed out the three types of software cost estimation methods- the top-down method, the bottom-up method and the analogy method. They discussed the COCOMO model, the COCOMO 2 model and also the inability of the COCOMO model to handle Commercially available Off-The-Shelf (COTS) product integration costs.

Wen et al. (2012) analysed 84 studies through a thorough literature review of empirical studies on Machine learning (ML) models published between 1991 and 2010. They discovered that eight machine learning techniques have been employed in SDEE models, and that the estimation accuracy of these techniques was not only close to acceptable estimation levels but better than non-machine learning models. They also concluded that different ML models were suitable to different estimation contexts, based on the strengths and weaknesses of the model.

Nassif et al. (2013) compared Multilayer Perceptron (MLP) and novel log linear regression model which was

based on Use Case Point (UCP) model for SDEE. They developed a linear regression model for prediction of the values of the productivity factor in their novel regression model. They also developed a Multi-Layer Perceptron artificial neural network model which uses the size of the software and the productivity of the team which is represented by eight factors as inputs to give the software effort. They claimed that MLP model surpasses the regression model for small projects (effort $<$ 3000 person-hours) and that the log-linear regression model gives improved results when larger projects (effort $>$ 3000 person-hours) are considered.

Dave and Dutta (2014) provided a review covering various artificial neural network-based models for SDEE as proposed by various researchers. They reviewed twenty-one articles and emphasized on the abilities of neural network-based models in SDEE.

Rao and Kumar (2015) proposed a Generalized Regression Neural Network (GRNN) to incorporate enhanced software effort estimation for COCOMO dataset. They used the Mean Magnitude Relative Error (MMRE) and Median Magnitude Relative Error (MdMRE) as the evaluation criteria. They compared the proposed GRNN with various techniques such as M5, Linear regression and Radial Basis Function (RBF) kernel.

Panda et al. (2015) used the Story Point Approach (SPA) to enhance the accuracy of prediction of agile SDEE. They used different neural networks—GRNN, Probabilistic Neural Network (PNN), Group Method of Data Handling (GMDH), Polynomial Neural Network and Cascade-Correlation Neural Network. They concluded that cascade networks outscore other networks.

Kaushik et al. (2016) integrated firefly algorithm and artificial neural network (ANN) for accurate cost predictions. The novel method was compared with particle swarm optimization and it was concluded that the ANN model gives more accurate estimations than particle swarm optimization.

Rijwani and Jain (2016) used Multi Layered Feed Forward Neural Network for software effort estimation and experimentally evaluated their model on COCOMO dataset.

Miandoab and Gharehchopogh (2016) proposed a COA-Cuckoo optimization and K-Nearest Neighbours (KNN) algorithm for SDEE. They evaluated the proposed technique on eight evaluation criteria: Mean Magnitude of Error Relative to the estimate (MMER), Mean Magnitude of Relative Error (MMRE), Median Mean Magnitude of Relative Error (MDMRE), Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Prediction(N) (PRED (N)) and Mean Square Error (MSE), and six datasets: KEMERER, MAXWELL, MIYAZAKI 1, NASA93 60, NASA93 63 and NASA93 and concluded that the proposed technique outscore other techniques, such as COCOMO, KNN and Cuckoo optimization algorithm, in KEMERER, MIYAZAKI1, NASA93 60 and NASA93 datasets.

Kaushik et al. (2017) proposed a novel method CUCKOO-FIS, which integrates two techniques: Cuckoo optimizations, a meta-heuristic search algorithm and Fuzzy Inference System. The collaborated technique is used for effort estimation and is successfully evaluated on software effort estimation datasets.

Rajpurohit et al. (2017) provided a comprehensive list of metaheuristic algorithms developed so far which is very insightful for solving various engineering problems including software cost estimation. The work provided a base for the researchers to work with metaheuristic algorithms.

Sharma and Pant (2017) proposed an Intermediate Artificial Bee Colony Greedy (I-ABC Greedy) algorithm. This algorithm overcomes the limitations of ABC algorithm. The efficiency of the proposed technique was verified by applying it on three real world problems consisting of parameter estimation in software reliability growth models, software effort estimation and redundancy optimization in modular software system models.

Kaushik et al. (2020) used Deep Belief Network along with Ant Lion Optimization (DBN-ALO) for effort estimation in both agile and non-agile datasets. Their approach worked best for both agile and non-agile development approaches.

All the above techniques were unique in their own way and every year as the new techniques are coming there remains always a scope of improvement. So, this work is to explore that improvement by integrating Whale Optimization Algorithm (WOA) and Deep Belief Network (DBN) for Software Development Effort Estimation (SDEE).

3 Overview of techniques employed

3.1 Deep learning

Deep learning is a part of a family of machine learning methods based on learning data representations. It uses a given training set to extrapolate new features from a limited set of features. Hence, it owes an advantage over previous neural networks and other machine-learning algorithms. It has the ability to search for and discover other features that correlate to those that are already known. It may discover new ways of separating the noise from the signal to better hear the signal. Two models of deep learning that have been used in this research are described in Sects. 3.1.1 and 3.1.2.

3.1.1 Restricted Boltzmann machine (RBM)

RBMs (Yang and Papa 2016; Fischer and Igel 2012) are energy-based stochastic neural networks that have two layers—the visible layer (to represent observable data) and the hidden layer (to capture the dependencies observed between data). RBMs are connected in a complete bipartite graph manner as shown in Fig. 1, where there are m visible units and n hidden units. Each connection between a visible unit and a hidden unit in an RBM is associated with a weight. The matrix of all the weights of an RBM is represented by a weight matrix W.

The energy function of an RBM is given in Eq. (1)

$$E(v,h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i h_j w_{ij} \quad (1)$$

where a and b are the biases of visible and hidden layer respectively, v_i is the value of observable data at ith visible unit, h_j is the state of jth hidden unit and w_{ij} is the weight associated with the connection between ith visible unit and jth hidden unit.

In a DBN, each RBM is trained in an unsupervised manner, independently of other RBMs.

3.1.2 Deep belief network

A Deep belief network (DBN) (Yang and Papa 2016; Fischer and Igel 2012) is formed when more than one Restricted Boltzmann Machines (RBMs) are stacked upon each other. In a DBN, the hidden layer of the ith RBM acts as input layer to $(i + 1)$th RBM. Each RBM in a DBN is trained in an unsupervised manner independently of other RBMs. A final output layer is also added to the top of the DBN. The DBN can be fine-tuned with any algorithm (e.g. gradient descent algorithm, metaheuristic algorithm etc.) to minimize some error measure with the output obtained at the final layer. A typical structure of DBN is shown in Fig. 2. It represents L RBM layers, with W^i being the weight matrix of the ith RBM.

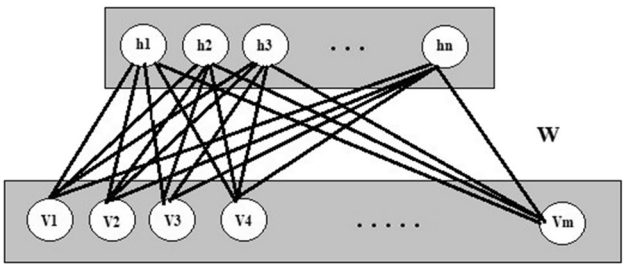

Fig. 1 A typical structure of RBM

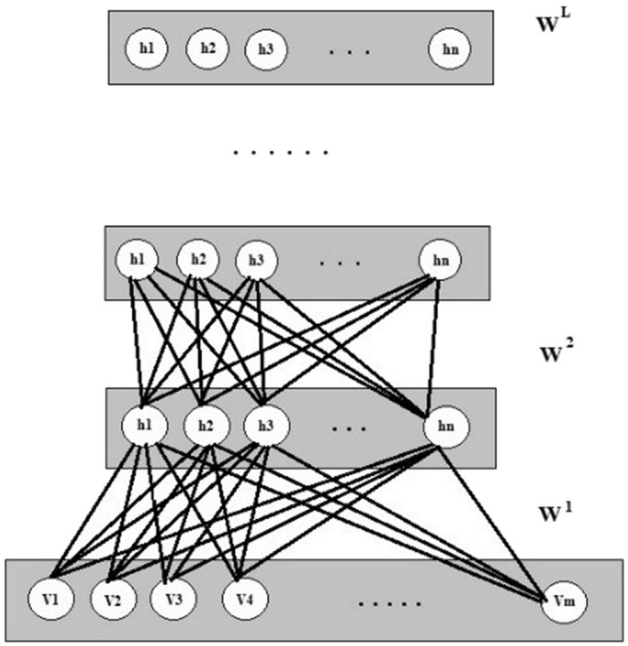

Fig. 2 A typical structure of DBN

3.2 Whale optimization algorithm

Whale Optimization Algorithm (WOA) (Mirjalili and Lewis 2016) draws its inspiration from the hunting behaviour of humpback whales called the bubble-net feeding method. Generally, they hunt schools of krill and small fish close to the surface. They do so by creating distinctive bubbles along a circle or '9'-shaped path, which comprises upward-spirals' and 'double- loops. The humpback whales encircle the prey with fins which are flashing, that keeps the prey contained and does not escape. The mathematical model of encircling prey, spiral bubble-net foraging manoeuvrer and prey search is described in the following section:

3.2.1 Encircling prey

Humpback whales encircle the prey and keep updating their positions towards the best search agent as the iterations increase i.e. from start to a maximum number of iterations. This behaviour is mathematically formulated as:

$$\vec{E} = \left| \vec{D} \cdot \overrightarrow{Y^*}(t) - \vec{Y}(t) \right| \quad (2)$$

$$\vec{Y}(t+1) = \overrightarrow{Y^*}(t) - \vec{B} \cdot \vec{E} \quad (3)$$

where $\vec{B}$ and $\vec{D}$ are the coefficient vectors, t indicates the current iteration, Y * is the position vector of the best solution obtained so far, $\vec{Y}$ is the position vector, | | is the absolute value and.is an element-by-element multiplication. The vectors $\vec{B}$ and $\vec{D}$ are calculated as follows:

$$\vec{B} = 2\vec{b} \cdot \vec{s} \cdot \vec{b} \quad (4)$$

$$\vec{D} = 2 \cdot \vec{s} \quad (5)$$

where $\vec{b}$ is decreased linearly from 2 to 0 over the course of iterations (in both exploration and exploitation phases) and $\vec{s}$ is a random vector in [0, 1].

3.2.2 Bubble-net attacking method

The following two approaches are developed to mathematically model the bubble-net behaviour of the humpback whales:

1. *Shrinking encircling mechanism* This behaviour is achieved by decreasing the value of b from 2 to 0 in Eq. (4) over the course of iterations. The new position of a search agent can be defined anywhere in between the original position of the agent and the position of the current best agent by setting random values for $\vec{B}$ in [− 1, 1].
2. *Spiral updating position* The spiral equation between the position of the prey and the whale to imitate the helix-shaped movement of the humpback whales is as follows:

$$\vec{Y}(t+1) = \vec{E'} \cdot e^{cl} \cdot cos(2\pi l) + \vec{Y^*}(t) \quad (6)$$

The humpback whales move around the prey within a shrinking circle as well as a spiral-shaped path simultaneously. Subsequently, in order to perfect this behavior, probability of 50% is there to choose between either the shrinking encircling method or the spiral model, in order to update the position of whales. The mathematical model is described as follows:

$$\vec{Y}(t+1) = \left\{ \vec{Y^*}(t) - \vec{B}.\vec{E} \right\} \text{ if } p \leq 0.5 \quad (7)$$

$$\vec{E'} \cdot e^{cl} \cdot cos(2\pi l) + \vec{Y^*}(t) \text{ if } p \geq 0.5 \quad (8)$$

where $\vec{E'} = \left| \vec{Y^*}(t) - \vec{Y}(t) \right|$ and is the distance of the ith whale to the prey (the best solution obtained so far), c is constant for defining the shape of the logarithmic spiral, l is a random number in [− 1, 1] and p represents a random number in [0, 1].

3.2.3 Search for prey

The variation of $\vec{B}$ vector can be utilized to search for prey, i.e., exploration phase. The mathematical model for this phase is as follows:

$$\vec{E} = \left| \vec{D}.\vec{Y_{rand}} - \vec{Y} \right| \quad (9)$$

$$\vec{Y}(t+1) = \vec{Y_{rand}} - \vec{B} \cdot \vec{E} \quad (10)$$

where $\vec{Y_{rand}}$ is a random position vector (a random whale) chosen from the current population.

4 Proposed methodology

This research uses the Deep Belief Network to predict software development effort. The DBN is fine-tuned using Whale Optimization Algorithm. The block diagram for the proposed methodology is given in Fig. 3. The graphical Representation of the proposed DBN for SDEE is given in Fig. 4.

The following key steps were taken for implementing the proposed methodology for SDEE:

a. Construction of RBM
b. Training of RBM
c. Construction of DBN
d. Fine-tuning of DBN with WOA

These steps are explained below:

4.1 Construction of RBM

The Restricted Boltzmann Machine (RBM) is constructed by taking two layers—input layer (visible layer) and hidden layer. The inputs (i.e. effort multipliers) v_i from datasets are fed to the input layer. For example, from COCOMO81 dataset, 15 effort multipliers (EMs), whose values for project ID 1, are 0.88, 1.16, 0.7, 1, 1.06, 1.15, 1.07, 1.19, 1.13, 1.17, 1.1, 1, 1.24, 1.1 and 1.04 are fed to the input layer of RBM 1. Subsequently, the values of hidden units of hidden layer are calculated using Eq. (11)

$$h_j = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + b_j\right) \quad (11)$$

where m is the number of visible units, b is the bias of the hidden layer, v_i is the value of observable data at ith visible unit, h_j is the state of jth hidden unit, w_{ij} is the weight associated with the connection between ith visible unit and

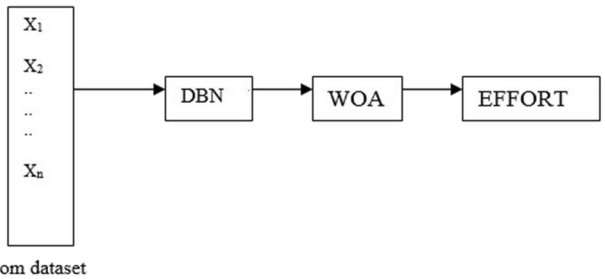

Fig. 3 Block diagram representing the proposed methodology

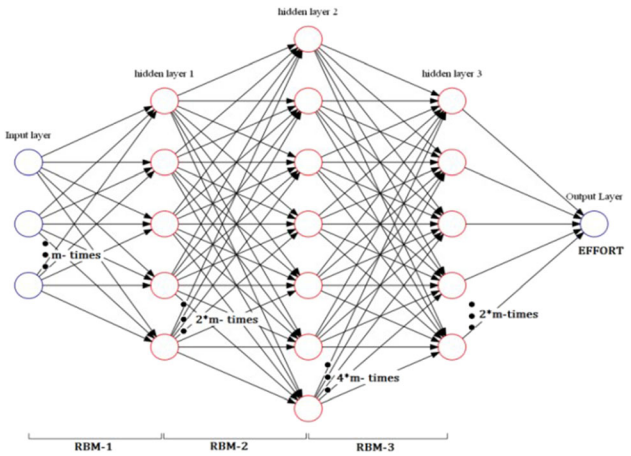

Fig. 4 Graphical Representation of the proposed DBN for SDEE

*j*th hidden unit and $\sigma(.)$ is the sigmoid activation function given in Eq. (12).

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{12}$$

Weights w_{ij} and bias b are initialized to random values.

4.2 Training of RBM

For training of RBM (Yang et al. 2016; Fischer et al. 2012), three parameters are updated, the weight matrix W (the set of weights between visible layer and hidden layer), the bias of visible layer a and the bias of hidden layer b in each iteration. In this research, the number of iterations for training of RBM is taken as 100. Firstly, probability of configuration *(v, h)* for all visible-hidden unit pair is computed using Eq. (13)

$$P(v,h) = \frac{e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \tag{13}$$

where E (v, h) is computed using Eq. (14)

$$E(v,h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i h_j w_{ij} \tag{14}$$

here, a and b are the biases of visible layer and hidden layer respectively (initialized randomly), m is the number of visible units, n is the number of hidden units, v_i is the value of observable data at *i*th visible unit, h_j is state of *j*th hidden unit and w_{ij} is the weight associated with the connection between *i*th visible unit and *j*th hidden unit. Weights w_{ij} and biases a and b are initialized to random values. Then, given a visible unit (and its observable data), its probability is computed over all the hidden vectors as given in Eq. (15)

$$P(v) = \frac{\sum_h e^{-E(v,h)}}{\sum_{v,h} e^{-E(v,h)}} \tag{15}$$

The data-driven probability, $E[hv]^{data}$ is directly used for updating the weight matrix W and biases a and b which is calculated using Eq. (16) as

$$E[hv]^{data} = P(h|v)v^T \tag{16}$$

where $P(h|v)$ is the conditional probability for obtaining h given v and this is calculated using Eq. (17) as

$$P(v) = \frac{P(v,h)}{P(v)} \tag{17}$$

The reconstructed data-driven probabilities $[hv]^{model}$, is also needed in updating the weight matrix W and biases a and b. It is calculated by a method given by (Hinton and Geoffrey 2002) which is based on contrastive divergence. Firstly, the states of hidden units are computed from visible units (which are initialized with a training sample) using Eq. (18) and the states of visible units are reconstructed using Eq. (19).

$$P(h_j = 1|v) = \sigma\left(\sum_{i=1}^{m} w_{ij} v_t + b_j\right) \tag{18}$$

$$P(h_j = 1|h) = \sigma\left(\sum_{i=1}^{m} w_{ij} h_i + b_j\right) \tag{19}$$

where $\sigma(.)$ is the sigmoid activation function given in Eq. (12).

Formula for $E[hv]^{model}$ is given in Eq. (20)

$$E[hv]^{model} = P(\tilde{h}|\tilde{v})\tilde{v}^T \tag{20}$$

Finally, Eq. (21), Eq. (22) and Eq. (23) are used to update the weight matrix W and biases a and b respectively.

$$\begin{aligned} W^{t+1} &= W^t + \eta\left(E\left[hv\right]^{data} - E\left[hv\right]^{model}\right) \\ &= W^t + \eta\left(P\left[h|v\right]v^T - P\left[\tilde{h}|\tilde{v}\right]\tilde{v}^T\right) \end{aligned} \tag{21}$$

$$a^{t+1} = a^t + \eta\left(v - E[v]^{model}\right) = a^t + \eta(v - \tilde{v}) \tag{22}$$

$$\begin{aligned} b^{t+1} &= b^t + \eta\left(E\left[h\right]^{data} - E\left[h\right]^{model}\right) \\ &= b^t + \eta\left(P\left[h|v\right] - P\left[\tilde{h}|\tilde{v}\right]\right) \end{aligned} \tag{23}$$

The training of RBM 1 is done for the number of iterations in an unsupervised manner. After stacking multiple RBMs on each other, a DBN is constructed. Its process of construction is given in subsequent section.

4.3 Construction of DBN

The DBN is constructed when multiple RBMs are stacked upon each other. In this research, we have stacked three RBMs on each other to construct the DBN. Each RBM is trained independently in an unsupervised manner for a specific number of epochs.

The values of the hidden units of RBM 1 after training are fed to the visible units of RBM 2. Subsequently, the values of the hidden units of RBM 2, after training, are fed to the visible units of RBM 3 (final RBM). Finally, after the training of RBM 3, an output layer is added to RBM 3 consisting of only one output unit as shown in Fig. 4. The values of the hidden units of the final hidden layer (RBM 3's hidden layer) are multiplied with the weights between the final hidden layer and the output unit and summed up to give the calculated effort.

For example, after the *m* effort multipliers of the project ID 1 of COCOMO81 dataset is fed to the input layer of RBM 1, RBM 1 was trained using 100 epochs. The value of *m* is 15 in this case. The values of its hidden layer nodes turned out to be 0.842, 0.989, 0.823, 0.759, 0.751, 0.788, 0.969, 0.764, 0.855, 0.839, 0.805, 0.778, 0.789, 0.765, 0.732, 0.932, 0.864, 0.761, 0.760, 0.960, 0.805, 0.893, 0.958, 0.735, 0.933, 0.726, 0.890, 0.909, 0.848, and 0.940. After the training of RBM 1, RBM 2 was trained in a similar way. The values of its hidden layer nodes turned out to be 0.818, 0.812, 0.819, 0.799, 0.811, 0.777, 0.812, 0.813, 0.803, 0.804, 0.810, 0.808, 0.813, 0.819, 0.825, 0.796, 0.819, 0.802, 0.812, 0.822, 0.814, 0.785, 0.808, 0.820, 0.780, 0.807, 0.807, 0.808, 0.795, 0.802, 0.796, 0.800, 0.789, 0.809, 0.798, 0.802, 0.810, 0.803, 0.810, 0.802, 0.811, 0.810, 0.803, 0.797, 0.779, 0.804, 0.807, 0.817, 0.795, 0.805, 0.779, 0.824, 0.802, 0.815, 0.826, 0.822, 0.796, 0.805, 0.789 and 0.797. Finally, after the training of RBM 2, RBM 3 was trained in a similar way. The values of its hidden layer nodes turned out to be 0.902, 0.907, 0.888, 0.887, 0.908, 0.906, 0.903, 0.903, 0.902, 0.891, 0.884, 0.906, 0.902, 0.902, 0.899, 0.905, 0.898, 0.900, 0.893, 0.902, 0.888, 0.904, 0.887, 0.898, 0.903, 0.892, 0.908, 0.895, 0.912 and 0.899.

The DBN is then fine-tuned using the whale optimization algorithm explained in next Sect. 4d.

4.4 Fine tuning of DBN with WOA

For fine-tuning of DBN, the whale optimization algorithm is run and weights between the hidden layer of the final RBM and the single output unit are calculated from the parameters returned by the whale algorithm in the first for loop. The matrices used; weights' calculations and dimensions of matrices used are given in Table 1. The matrices used in the algorithm are initialized to some random values. Table 2 provides 13 objective functions (F1–F13) (Mirjalili and Lewis 2016) which are used for finding the fitness values in WOA. The results are obtained from all the functions, F1–F13, for the proposed technique on all the four datasets but only the functions which provided the best results are shown in experimental evaluation under Sect. 5. Table 3 shows the values for the initialization of parameters used in the proposed approach.

The hidden units' values of RBM 3, obtained after training, are multiplied with their corresponding weight values obtained from WOA at the output unit and added as given in Eq. (24).

$$X = \left(\sum_{i=1}^{n} w_i h_i \right) \tag{24}$$

where X is the final sum obtained at the output layer of DBN, n is the number of hidden units in final RBM's hidden layer, h_i is the value of these hidden units and w_i are the values of weights between ith hidden unit of the final RBM and the single output unit obtained from WOA. For example, in our case of COCOMO81 dataset for project ID 1, when considering the WOA algorithm, the value of X came out to be 19.521. The sum X obtained from Eq. (24) is multiplied with the size of the project taken from the dataset to obtain the final calculated Effort as given in Eq. (25).

$$Effort = X * size \tag{25}$$

In our case, the value of size is 113 and thus the value of calculated *Effort* comes out to be 2205.793 where the actual value of effort is 2040.The calculated Effort is compared with the actual Effort from the dataset using various evaluation criteria as discussed under Sect. 5. The process of fine-tuning is terminated, if the magnitude of relative error (MRE) is within the acceptable range (taken as 25%) (Di Martino et al. 2011; Foss et al. 2003) or if the maximum number of epochs is reached which is taken as 500. Else, the procedure is repeated.

The metaheuristic algorithm continues to explore and exploit the search space until the end criteria is not satisfied. The actual Effort from the dataset is just used to verify whether MRE is within the acceptable range or not. The error measure or the actual value of Effort is not used for updating any parameters. The proposed technique has also been compared with DBN-BP. The equation used for updating the weights between RBM 3 and final output layer in DBN-BP is given as Eq. (26)

$$w_{new} = w_{old} + (\delta * alpha) \tag{26}$$

where δ is equal to MRE and alpha is the learning rate whose value lies in the range [0, 1].

Table 1 Particulars used in WOA for fine-tuning of the DBN

Matrices used	Dimensions of matrices	Weight calculation of DBN
Position matrix of whales Matrix of coefficient vectors	For position matrix: 1. Number of rows (Number of whales) = Number of hidden units at RBM 3 2. Number of columns (Number of dimensions of positions) = 1 For coefficient vector matrix*: 1. Number of rows (Number of whales) = Number of hidden units at RBM 3 2. Number of columns = 1 (indicating the fitness value) *Coefficient vector matrix is calculated by passing the positions of position matrix to an objective function	WOA as given in Sect. 3.2 is run and the whales' positions and the best whale's fitness value or the leader score obtained, after first for loop, are used in the following equation for weights' calculation: $w_i = \lvert position\ of\ whale_i \rvert \times leader\_score$ Where, $position\_of\_whale_i$ is the position of ith whale, w_i is the weight between ith hidden unit and the output unit, and $leader\_score$ is the fitness value/score of the best whale/leader

4.5 Complexity

There are four steps in the algorithm. The construction of RBM includes creating input layer and hidden layer. This takes constant time i.e., O (1). Let's say, time complexity for the training of RBM is $O(R)$. The construction of DBN involves training of stacked RBMs. In our proposed algorithm, 3 RBMs are trained, so time complexity is $O(3R)$. In the last step, for each iteration during the fine tuning of DBN, optimized weights from WOA are received, so time complexity is $O(N*W)$ where N is the maximum number of iterations and $O(W)$ is the time complexity of WOA. Hence, total time complexity for DBN-WOA is $O(1 + 3R + N*W) = O(R + N*W)$.

5 Experimental evaluation

Testing the proposed methodology for SDEE, is imperative after training the DBN, as it gives clear estimates of how well the technique can perform when it is used to predict effort with unseen projects (not seen during training period). There are various methodologies to test and validate the techniques. Some of them include—Holdout method, Leave-one-out cross validation, tenfold Cross Validation, threefold Cross Validation etc. In this research, the proposed technique is experimentally validated using threefold cross validation which has been widely used by various authors in SDEE to validate their proposed methodologies such as Burgess and Lefley (2001), Kumar et al. (2008). It divides the dataset into three parts in a certain ratio. Any of the two parts are taken for training and the left-out part is taken for testing. This is repeated three times so that all the parts get to be in a training set exactly twice and, in a testing set exactly once. In this research, we have divided the datasets into three equal parts. The advantage of threefold Cross Validation is that it does not matter how the dataset gets divided. Every data point is in a test set exactly once and is in a training set exactly twice.

The datasets used are COCOMO81, NASA93, MAXWELL and CHINA for experimentally and statistically evaluating the proposed techniques. These datasets have been obtained from PROMISE Software Engineering Repository (http://promise.site.uottawa.ca/SERepository/). These datasets have been used widely by various authors in the domain of SDEE (Benala et al. 2012; Elish 2009; Kaushik et al. 2016). Evaluation criteria play an important role in estimating and commenting upon the success of a technique. In this paper, we have used three evaluation criteria—Mean of Magnitude of Relative Error (MMRE), Prediction (l) (Pred (l)) and Median of MRE (MdMRE).

Mean of magnitude of relative error (MMRE) is calculated as the average of magnitude of relative errors (MREs) of all projects in the dataset. The formula for calculating MMRE is given in Eq. (27).

$$\text{MMRE} = \frac{1}{N}\sum_{i=1}^{N} MRE_i \tag{27}$$

where N is number of projects in dataset.

Magnitude of relative error (MRE) finds the relative error between actual effort and estimated effort for each project in the dataset. Formula for calculating MRE is given in Eq. (28).

$$\text{MRE} = \frac{\lvert \text{Actual Effort} - \text{Estimated Effort} \rvert}{\text{Actual Effort}} \tag{28}$$

Pred (l) represents the percentage of MREs less than or equal to the value l among all projects. It is defined in Eq. (29).

$$\text{Pred}(l) = \frac{k}{n} \tag{29}$$

Table 2 Description of unimodal and multimodal benchmark function (Mirjalili and Lewis 2016)

Function	V_no	Range	f_{min}
$F1(x) = \sum_{i=1}^{n} x_i^2$	30	[− 100, 100]	0
$F2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	[− 10, 10]	0
$F3(x) = \sum_{i=1}^{n} (\sum_{j-1}^{i} x_j^2)^2$	30	[− 100, 100]	0
$F4(x) = \boldsymbol{max}_i\{\lvert x_i \rvert, 1 \le i \le n\}$	30	[− 100, 100]	0
$F5(x) = \sum_{i=1}^{n} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	30	[− 30, 30]	0
$F6(x) = \sum_{i=1}^{n} (\lvert x_i + 0.5 \rvert)^2$	30	[− 100, 100]	0
$F7(x) = \sum_{i=1}^{n} i x_i^4 + \boldsymbol{random}[0, 1)$	30	[− 1.28, 1.28]	0
$F8(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	30	[− 500, 500]	− 4.18.9829 × 5
$F9(x) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	30	[− 5.12, 5.12]	0
$F10(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{\pi}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	30	[− 32, 32]	0
$F11(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	[− 600, 600]	0
$F12(x) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2\left[1 + 10sin^2(\pi y_{i+1})\right] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{x_i+1}{4}$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m x_i > a \\ 0 - a < x_i < a \\ k(-x_i - a)^m x_i < -a \end{cases}$	30	[− 50, 50]	0
$F13(x) = 0.1\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{n} (x_i - 1)^2\left[1 + \sin^2(3\pi x_i + 1)\right] + (x_n - 1)^2\left[1 + \sin^2(2\pi x_n)\right]\right\} \sum_{i=1}^{n} u(x_i, 5, 100, 4)$	30	[− 50, 50]	0

where n is the total number of observations and k is the number of observations whose MRE is less than or equal to l. MdMRE is the median of MREs of all projects of a dataset. MdMRE is less susceptible to extreme values. It behaves similar to MMRE but is more likely to select a true model if under-estimation is served (Kumar et al, 2008). The software development effort estimation models are said to be acceptably accurate if the value of MMRE and MdMRE is less than or equal to 0.25 and Pred (0.25) is greater than or equal to 0.75 (Di Martino et al. 2011; Foss et al. 2003).

The results of training of the proposed technique on all the four datasets are given in Table 4. The proposed DBN-WOA was evaluated with all the 13 evaluation functions and for all the three folds of the dataset. The objective function which gave the best result is tabulated for the respective datasets as Table 4. It is found that the objective function F11 is best for COCOMO81 and NASA93 datasets; the objective function F4 is best for MAXWELL dataset and the objective function F1 is best for CHINA dataset. The proposed technique has been tested using the best objective functions obtained from Table 4. After the optimum weights of DBN are successfully explored and

Table 3 Initialization of parameters for implementing the proposed methodology

Parameter used	Initialization
Number of input nodes (m) in datasets COCOMO81, NASA93	15
Number of input nodes (m) in dataset MAXWELL	24
Number of input nodes (m) in dataset CHINA	10
Number of hidden nodes in hidden layer 1	2*m
Number of hidden nodes in hidden layer 2	4*m
Number of hidden nodes in hidden layer 3	2*m
Weights between input—hidden layer 1	Random values
Weights between hidden layer 1—hidden layer 2	Random values
Weights between hidden layer 2—hidden layer 3	Random values
Weights between hidden layer 3—output layer	Random values
Bias *a*	Random value
Bias *b*	Random value
Number of epochs for the training of each RBM	100
Max_limit on the number of epochs for fine-tuning of DBN	500
Position matrix of Whales	Random values
Acceptable MRE in the end criteria for fine-tuning of DBN	0.25

searched using WOA in the training phase, the resultant DBN is subject to testing. The results from the testing phase on the four datasets are presented in Table 5. Since both the training and the testing errors are low, it can be inferred that the model is neither overfitting nor underfitting.

The snapshot of the convergence graph for a particular stage of the proposed methodology where the calculated effort is trying to get closer to the actual effort for the project ID 32 of COCOMO81 dataset is given as Fig. 5. The convergence is helping the DBN get optimum weights.

The proposed technique is compared against DBN fine-tuned with backpropagation and the results from its training and testing phases are presented in Tables 6 and 7 respectively. The results show that the integration of WOA with the DBN greatly enhances the efficiency for estimation of software development effort as compared to the integration of backpropagation with DBN. The rationale behind this is that searching for weights in a DBN can be modelled as an optimization problem where the goal is to achieve optimum weights. Metaheuristic techniques tend to surpass other optimization techniques as they tend to avoid local optima.

Figure 6 shows the bar graph representation of DBN-WOA and DBN-BP for COCOMO81 training dataset.

The results of the proposed DBN-WOA technique is also compared with the work given by the authors (Rijwani and Jain 2016; Kaushik et al. 2020). Table 8 demonstrates the result on few of the COCOMO dataset as the same project Ids were used by the authors (Rijwani and Jain 2016).

Table 4 Results of training of the proposed technique, DBN-WOA

Datasets	Objective function	Fold	Evaluation criteria		
			MMRE (%)	Pred (25) (%)	MdMRE (%)
COCOMO 81	F11	Fold 1	7.228	86.316	5.472
		Fold 2	10.013	89.474	5.025
		Fold 3	7.191	86.190	6.879
NASA93	F11	Fold 1	11.366	96	2.028
		Fold 2	5.853	92.258	4.901
		Fold 3	9.829	87.097	0.897
MAXWELL	F4	Fold 1	9.9182	89.428	6.832
		Fold 2	1.327	96	5.035
		Fold 3	5.926	90.588	7.725
CHINA	F1	Fold 1	13.543	86.939	10.949
		Fold 2	13.744	80.918	8.388
		Fold 3	17.903	85.918	9.886

Table 5 Results of testing of the proposed technique, DBN-WOA

Dataset	Objective function	Fold	Evaluation criteria		
			MMRE (%)	Pred (25) (%)	MdMRE (%)
COCOMO81	F11	Fold 1	8.012	88.571	9.558
		Fold 2	7.884	90.909	9.207
		Fold 3	7.786	88.095	6.357
NASA93	F11	Fold 1	9.012	85.065	5.666
		Fold 2	6.388	94.065	7.650
		Fold3	8.126	80.968	9.578
MAXWELL	F4	Fold 1	9.012	84.065	5.666
		Fold 2	8.388	88.065	7.650
		Fold3	7.126	90.968	9.578
CHINA	F1	Fold 1	9.022	87.818	5.733
		Fold 2	6.368	93.860	5.506
		Fold3	7.500	94.957	4.696

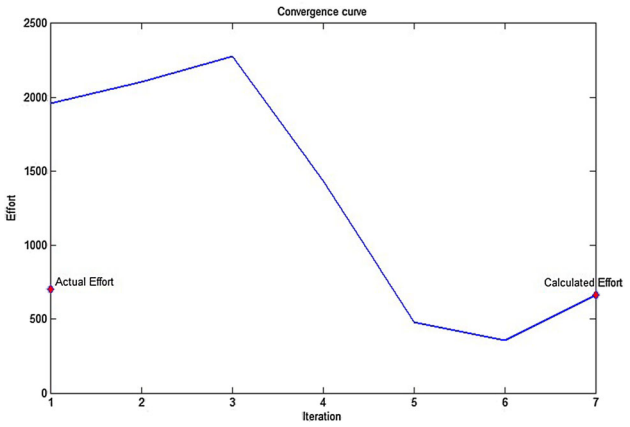

Fig. 5 Epochs vs Effort curve for DBN-WOA for COCOMO81 dataset for project ID 32

Table 6 Results of training of DBN-BP for the datasets used

Dataset	Fold	Evaluation criteria		
		MMRE (%)	Pred (25) (%)	MdMRE (%)
COCOMO81	Fold 1	37.168	62.857	39.678
	Fold 2	34.224	65.714	38.823
	Fold 3	28.983	50.588	24.348
NASA93	Fold 1	37.831	63.207	34.553
	Fold 2	34.780	57.647	38.701
	Fold 3	35.941	43.076	31.697
MAXWELL	Fold 1	40.470	51.1282	36.545
	Fold 2	33.383	52.882	73.055
	Fold 3	32.628	54.714	41.353
CHINA	Fold 1	30.911	63.26	41.773
	Fold 2	40.903	57.40	38.880
	Fold 3	38.156	68.55	43.997

Table 7 Results of testing of DBN-BP for the datasets used

Dataset	Fold	Evaluation criteria		
		MMRE (%)	Pred (25) (%)	MdMRE (%)
COCOMO81	Fold 1	31.925	64.761	27.499
	Fold 2	35.588	69.047	38.556
	Fold 3	32.226	70	39.969
NASA93	Fold 1	41.786	66.451	37.815
	Fold 2	38.129	69.677	38.155
	Fold 3	41.580	72.580	32.816
MAXWELL	Fold 1	32.034	69.047	38.538
	Fold 2	33.191	63.809	38.120
	Fold 3	37.728	70.56	32.584
CHINA	Fold 1	34.663	72.25	34.185
	Fold 2	39.666	72.244	30.418
	Fold 3	37.310	74.285	34.538

Table 8 shows that the estimated effort using DBN-WOA is better for most of the projects.

The authors (Kaushik et al. 2020) used Ant Lion Optimization (ALO) and proposed DBN-ALO. The estimated effort given by them for COCOMO dataset with P.Id 1is 2378 and for P.Id 3 it is 230 whereas the estimated effort using DBN-WOA is 2205.793 for P.Id1 and it is 240.68 for P.Id 3. Thus, DBN-WOA is providing better results than the techniques used by the earlier researches.

6 Statistical validation

Since, experimental evaluation alone cannot be used to accurately determine the success of the proposed technique for SDEE, the analysis of the technique using statistical

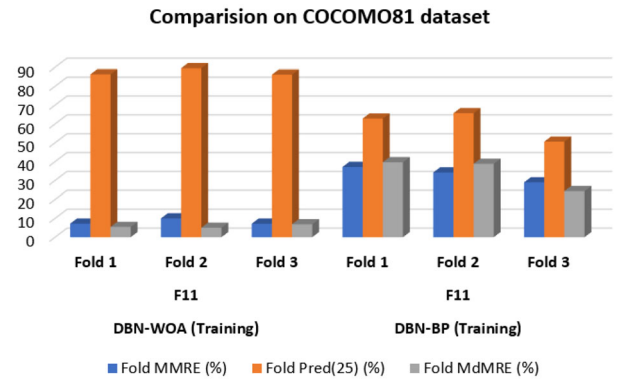

Fig. 6 Comparison of DBN-WOA and DBN-BP on COCOMO81 training dataset

tests is imperative (Prasad Reddy et al. 2010; Kitchenham and Mendes 2009).

Statistical inferential tests are of three types- parametric, non-parametric and semi-parametric. The parametric tests assume that the data comes from a type of probability distribution and make deductions about the distribution parameters. Non-parametric tests, on the other hand, do not rely on data that belongs to any specific distribution, and includes techniques that assume that the model structure is not fixed. They are mainly used for ordinal data. The semi-parametric tests make use of the merits of both parametric as well non-parametric techniques in a sophisticated manner. Mittas et al. (2015) has given a new set of techniques called semi-parametric models (SPM) to handle software estimations. In software cost estimations, we assume that the project data does not follow any distribution. Hence, to validate the work statistically, non-parametric tests are used. Here, Friedman test is used which the non-parametric equivalent of the well is known analysis of variance. The Friedman test tests the change between several related samples. The null hypothesis of the Friedman test mentions that all techniques perform equivalent. This test is carried

Table 8 Effort Comparison on COCOMO dataset

P.Id	Effort (Dataset)	Effort (COCOMO)	Effort MLFFN (Rijwani & Jain)	Effort DBN-WOA
1	2040	1616.38	2031.84	2205.793
3	243	233.88	239.039	240.68
11	218	189.93	193.802	200.52
18	11.400	8552.88	11.229	11.304
20	6400	3603.34	5811.2	5910.23
26	387	279.93	371.52	377.57
27	88	59.002	74.976	79.92
50	176	132.162	172.515	173.46
51	122	114	119.926	123.85
54	20	6.24	12	10.67
55	18	7.5	16.902	16.96
56	958	537	953.21	954.28
60	57	23.91	48.2562	50.68

Table 9 Results of Friedman Test depicting mean ranks for the datasets

Objective Functions	COCOMO81	NASA93	CHINA	MAXWELL
F1	6.36	6.32	4.55	4.42
F2	6.10	6.48	5.24	–
F3	6.00	6.33	4.84	4.53
F4	5.34	4.95	4.67	3.92
F5	–	–	–	–
F6	5.39	5.28	5.02	4.40
F7	9.33	8.58	–	8.42
F8	–	–	–	–
F9	6.16	6.21	4.69	4.77
F10	5.11	5.69	5.87	5.02
F11	5.10	4.06	–	–
F12	5.80	5.88	4.58	4.66
F13	5.31	6.23	5.55	4.87

out using IBM SPSS (Statistical Package for the Social Sciences) tool. It provided the mean ranks which are depicted in Table 9. Using this test, it was confirmed that the objective function which was giving the best experimental results provided the same here also. So, for each of the datasets the MRE values calculated using different objective functions were used as an input to the tool. The objective functions which were giving the best values in experimental evaluations were giving best values here also. For example, the function F11 was giving best results for COCOMO 81 and NASA 93, here also this function has the lowest mean in both the datasets. The value is 5.10 for COCOMO81 and 4.06 for NASA93 dataset. Similarly, the function F1 was performing best for CHINA dataset and here also it has the minimum rank with the value 4.55 and for MAXWELL dataset, the function F4 has the lowest mean rank of 3.92. This is depicted in Table 9.

The objective functions F5 and F8 for COCOMO81 and NASA93 dataset; the objective functions F5, F7, F8, F11 for CHINA and the objective functions F2, F5, F8, F11 for MAXWELL were not used in statistical validation as they provided NAN (not a number) value in MATLAB during experimental validation.

7 Limitations and future scope

The limitations of the proposed technique can be summarized as follows.

- The effort multipliers used in datasets employed may not be extremely accurate because developers might be optimistic in answering questions related to their capabilities. Therefore, it will inculcate errors in the calculated effort. Also, the effect of environmental factors cannot be determined with the available data.
- In this paper, traditional datasets are used for estimating the performance of the proposed technique. Nowadays, software is being developed using agile techniques. The proposed technique should also be evaluated and validated using these latest software projects' datasets.
- Although initialization of the parameters in the employed technique is done through extensive study and evaluation, more optimum initializations of these parameters might be possible.
- In this paper, k-fold cross validation is employed where the value of k is taken as 3, but it is very difficult to decide the appropriate value of k. Techniques employed can, therefore, be tested with more validation techniques such as tenfold cross validation, Leave-one out cross validation etc.
- Though some objective functions gave remarkable results for SDEE, functions such as F5 and F8 failed to provide appropriate efforts for all the proposed techniques.
- In this paper, 3 RBM's have been stacked upon each other to form a DBN. But it is difficult to know the appropriate value of the number of RBMs to be stacked to form a DBN. Using a different number of stacked RBMs might give different results.

All the above limitations can be overcome by further work in this domain. This research attempts to evaluate the effectiveness of integrating deep learning and metaheuristics for SDEE. Traditional datasets are used for conducting the experimental evaluation. Nowadays, software is majorly developed using agile techniques. Hence, the research can also be verified using the agile datasets (Panda et al. 2015). Integration of metaheuristic techniques with other deep learning models such as Recurrent Neural Networks and Convolutional Neural Networks for SDEE can also be evaluated to confirm the effectiveness of deep learning in effort estimates. For more refined fine-tuning of the parameters in DBN, techniques proposed by Calvet et al. (2016) can be employed. Also, the technique proposed can be tested by using different validation techniques.

8 Conclusion

In this research, Deep Belief Network (DBN) is used to predict software development effort and its parameters are finetuned using a metaheuristic technique, Whale Optimization Algorithm (WOA). The technique DBN-WOA, is evaluated on four effort estimation datasets and is compared with another variant of DBN which uses Back Propagation algorithm i.e., DBN-BP. The experimental and statistical results show that the technique DBN-WOA is at par than DBN-BP. In the future, we will work on the limitations listed in Sect. 7, to improve the existing method.

8.1 Answers to research questions

1. *Can deep learning accurately predict the software development effort?*
   Yes, deep learning can predict the software development effort within 25% MRE as can be observed from Tables 4 and 5. But the results of the technique can vary depending upon the DBN architecture.
2. *Can the fine tuning of DBN using Whale Optimization Algorithm (WOA) perform better than the fine tuning of the DBN using backpropagation?*

Yes, the fine-tuning of DBN with metaheuristic technique, like WOA outscores the fine-tuning of DBN with

backpropagation as can be inferred from the tabular results in Sect. 5. Here also, depending upon the initialization of parameters the performance of WOA may alter.

Funding The author(s) received no financial support for the research, authorship, and/or publication of this article.

Declarations

Conflict of interest All the authors declare that there are no conflicts of interest in publishing this paper.

Ethical approval This article does not contain any studies with human participants and/or animals performed by any of the authors.

References

Benala TR, Dehuri S, Mall R (2012) Functional link artificial neural networks for software cost estimation. Int J Appl Evol Comput 3(2):62–82. https://doi.org/10.4018/jaec.2012040104

Boehm BW (1984) Software engineering economics. IEEE Trans Softw Eng 10(1):4–21. https://doi.org/10.1109/TSE.1984.5010193

Breuel TM, Ul-Hasan A, Al-Azawi MA, Shafait F (2013) High-performance OCR for printed english and fraktur using LSTM networks. In: 12th International conference on document analysis and recognition, Washington, DC, USA, pp. 683–687, https://doi.org/10.1109/ICDAR.2013.140

Burgess CJ, Lefley M (2001) Can genetic programming improve software effort estimation? A comparative evaluation. Inf Softw Technol 43(14):863–873

Calvet L, Juan AA, Serrat C, Ries J (2016) A statistical learning-based approach for parameter fine-tuning of metaheuristics. Stat Oper Res Trans 40(1):201–224

Ciregan D, Ueli M, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: IEEE conference on computer vision and pattern recognition, Providence, RI, USA, 2012, pp. 3642–3649, https://doi.org/10.1109/CVPR.2012.6248110

Dave VS, Dutta K (2014) Neural network based models for software effort estimation: a review. Artif Intell Rev 42(2):295–307

Di Martino S, Ferrucci F, Gravino C, Sarro F (2011) Using web objects for development effort estimation of web applications: a replicated study. Lect Notes Comput Sci 6759:186–201

Elish MO (2009) Improved estimation of software project effort using multiple additive regression trees. Expert Syst Appl 36(7):10774–10778. https://doi.org/10.1016/j.eswa.2009.02.013

Fan Y, Qian Y, Xie FL, Soong FK (2014) TTS synthesis with bidirectional LSTM based recurrent neural networks. In: 15th annual conference of the international speech communication association, Singapore, pp. 1964–1968

Fenton NE, Pfleeger SL (1997) Software metrics: a rigorous and practical approach. Boston, MA, USA: Second Ed. PWS

Fischer A, Igel C (2012) An introduction to restricted Boltzmann machines. Lect Notes Comput Sci 7441:14–36

Foss T, Stensrud E, Kitchenham B, Myrtveit I (2003) A simulation study of the model evaluation criterion MMRE. IEEE Trans Softw Eng 29:985–995

Gray AR (1999) A simulation-based comparison of empirical modelling techniques for software metric models of development effort. In: 6th international conference on neural information processing. Proceedings (Cat. No. 99EX378) 2. Perth, WA, Australia, pp. 526–531 https://doi.org/10.1109/ICONIP.1999.845649

Gupta P, Gupta S, Gandhi OP (2014) Annual maintenance budget estimation for a plant system: digraph model and matrix approach. J Qual Maint Eng 20(2):193–210

Hinton GE (2002) Training products of experts by minimizing contrastive divergence. Neural Comput 14(8):1771–1800

Jiang Z, Naudé P, Jiang B (2007) The effects of software size on development effort and software quality. Int J Comput Inf Sci Eng 1(10):2939–2943

Jorgensen M, Shepperd M (2007) A systematic review of software development cost estimation studies. IEEE Trans Softw Eng 33(1):33–53. https://doi.org/10.1109/TSE.2007.3

Kaushik A, Tayal DK, Yadav K, Kaur A (2016) Integrating firefly algorithm in artificial neural network models for accurate software cost predictions. J Softw: Evol Process 28(8):665–688

Kaushik A, Verma S, Singh HJ, Chhabra G (2017) Software cost optimization integrating fuzzy system and COA-Cuckoo optimization algorithm. Int J Syst Assur Eng Manag 8(2):1461–1471

Kaushik A, Tayal DK, Yadav K (2020) A comparative analysis on effort estimation for agile and non-agile software projects using DBN-ALO. Arab J Sci Eng 45:2605–2618

Kitchenham B, Mendes E (2009) Why comparative effort prediction studies may be invalid. In: 5th International conference on predictor models in software engineering, Vancouver British Columbia, Canada, pp. 1–5, https://doi.org/10.1145/1540438.1540444.

Kumar KV, Ravi V, Carr M, Kiran NR (2008) Software development cost estimation using wavelet neural networks. J Syst Softw 81(11):1853–1867. https://doi.org/10.1016/j.jss.2007.12.793

Miandoab EE, Gharehchopogh FS (2016) A novel hybrid algorithm for software cost estimation based on cuckoo optimization and k-nearest neighbors algorithms. Eng Technol Appl Sci Res 6(3):1018–1022

Mittas N, Papatheocharous E, Angelis L, Andreou AS (2015) Integrating non-parametric models with linear components for producing software cost estimations. J Syst Softw 99:120–134

Muzaffar Z, Ahmed MA (2010) Software development effort prediction: a study on the factors impacting the accuracy of fuzzy logic systems. Inf Softw Technol 52(1):92–109

Nassif AB, Ho D, Capretz LF (2013) Towards an early software estimation using log-linear regression and a multilayer perceptron model. J Syst Softw 86(1):144–160

Online document, 20 Famous Software Disasters. http://www.devtopics.com/20-famous-software-disasters/. Accessed 6 March 2019

Online document, Promise Software Engineering Repository. http://promise.site.uottawa.ca/SERepository/datasets-page.html. Accessed 615 March 2019

Panda A, Satapathy SM, Rath SK (2015) Empirical validation of neural network models for agile software effort estimation based on story points. Procedia Comput Sci 57:772–781

Prasad Reddy PVGD, Sudha KR, Rama Sree P, Ramesh SNSVSC (2010) Software effort estimation using radial basis and generalized regression neural networks. J Comput 2(5):87–92

Pressman RS (1997) Software engineering: a practitioner's approach, 4th edn. McGraw-Hill, New York

Putnam LH (1978) A general empirical solution to the macro software sizing and estimating problem. IEEE Trans Softw Eng 4(4):345–361

Qin X, Fang M (2011) Summarization of software cost estimation. Procedia Eng 15:3027–3031

Rajpurohit J, Sharma TK, Abraham A, Vaishali A (2017) Glossary of metaheuristic algorithms. Int J Comp Inf Syst Ind Manage Appl 9:181–205

Rao PS, Kumar RK (2015) Software effort estimation through a generalized regression neural network. Adv Intell Syst Comput 337(1):19–30

Rijwani P, Jain S (2016) Enhanced software effort estimation using multi layered feed forward artificial neural network technique. Procedia Comput Sci 89:307–312

Sharma TK (2016). Application of shuffled frog leaping algorithm in software project scheduling. In Pratiksha S, Dipti S, & Millie P (Eds.), Problem solving and uncertainty modeling through optimization and soft computing applications (pp. 225–238). IGI Global

Sharma TK, Pant M (2014) Redundancy level optimization in modular software system models using ABC. Int J Intell Syst Appl 04:40–48

Sharma TK & Pant M (2017). Swarm intelligence in software engineering design problems. In: Ashish M. Gujarathi, Babu BV (Eds.), Evolutionary computation: techniques and applications (pp.163–194). Apple Academic Press Inc

Sharma TK, Pant M & Abraham A (2011). Dichotomous search in ABC and its application in parameter estimation of software reliability growth models. In: 3rd world congress on nature and biologically inspired computing salamanca, Spain, pp. 207–212, https://doi.org/10.1109/NaBIC.2011.6089460.

Seyedali M, Lewis A (2016) The whale optimization algorithm. Adv Eng Softw 95(C):51–67

Srivastava AK, Kumar G, Gupta P (2020) Estimating maintenance budget using monte carlo simulation. Life Cycle Reliab Saf Eng 9(1):77–89

Wen J, Li S, Lin Z, Hu Y, Huang C (2012) Systematic literature review of machine learning based software development effort estimation models. Inf Softw Technol 54(1):41–59

Xia W, Capretz LF, Ho D, Ahmed F (2008) A new calibration for function point complexity weights. Inf Softw Technol 50(7–8):670–683

Yang XS & Papa J (2016) *Bio-inspired computation and applications in image processing*. Academic Press