International Journal of Computational Intelligence Systems, Vol. 5, No. 4 (August, 2012), 679-699

SEffEst: Effort estimation in software projects using fuzzy logic and neural networks

Israel González-Carrasco [1] *, Ricardo Colomo-Palacios [2], José Luis López-Cuadrado[3], Francisco José García Peñalvo [4]

[1] Department of Computer Science, Universidad Carlos III,
Av. Universidad 30 - 28911. Leganés (Madrid). Spain. E-mail: israel.gonzalez@uc3m.es
[2] E-mail: ricardo.colomo@uc3m.es
[3] E-mail:joseluis.lopez.cuadrado@uc3m.es
[4] Departamento de Informática y Automática, Universidad de Salamanca,
Plaza de los Caídos S/N, 37008 - Salamanca. Spain. E-mail: fgarcia@usal.es

Received 31 October 2011
Accepted 18 May 2012

Abstract

Academia and practitioners confirm that software project effort prediction is crucial for an accurate software project management. However, software development effort estimation is uncertain by nature. Literature has developed methods to improve estimation correctness, using artificial intelligence techniques in many cases. Following this path, this paper presents SEffEst, a framework based on fuzzy logic and neural networks designed to increase effort estimation accuracy on software development projects. Trained using ISBSG data, SEffEst presents remarkable results in terms of prediction accuracy.

Keywords: Fuzzy Logic, Neural Networks, Software Engineering, Effort Estimation

1. Introduction

Software is now the driving force of modern business, government and military operations [79]. Given the importance of software in today's world, the development of software systems is a key activity for both industry and users. In such scenario, the requirements for project plans complying with time, effort, cost and quality have become a fundamental element for organizations producing software.

According to [42], two of the three most important causes of IT-project failure were related to poor resource estimations. Not in vain, effort estimation of software developments is an important subdiscipline in software engineering that has been the focus of much research, mostly over the last couple of decades [74]. Yet despite its importance, accurate and credible software effort estimation is a challenge for academic research and software industry [5]. Maybe that is the reason why software project effort estimation with high precision is still a largely unsolved problem. In this scenario, investigating novel methods for improving the accuracy of such estimates is essential to strengthen software companies' competitive strategy [58].

Inaccurate estimation of development efforts lowers the proficiency of the project, wastes the

*Corresponding author: israel.gonzalez@uc3m.es

company's budget, and can result in failure of the entire project [59]; not in vain, software price determination, resource allocation, schedule arrangement and process monitoring are dependent upon software estimation [34]. Underestimating the required software effort and cost can have detrimental effects on the software quality and eventually on the company's business reputation[35], apart from this, underestimation could produce under-staffing (that may result in staff burnout), under-scoping the quality assurance effort (that may increase the risk of low quality deliveries), and setting short schedule (that may result in loss of credibility as deadlines are missed). On the other hand, overestimation of the software cost can result in losing the opportunity to win the competition for the software project during price bidding [35] as well as delaying the use of the resources in the next project [19], meaning a risk for the whole project portfolio [36].

Nevertheless, accurate software estimation can provide powerful assistance when software management decisions are being made [39] in the initial steps, where budgets are approved [2]. Furthermore, any improvement in the accuracy of predicting the development effort can significantly reduce the costs from errors in this field, such as estimating inaccurately, misleading tendering bids, and disabling the monitoring progress [15].

As a result of the long tradition of software development estimation models, there are many models which can be classified into three groups:

1. Expert Judgment: Estimates using expert judgment [43], [41], can be produced easily and without the need of complicated tools or techniques. However, due to its human and subjective nature is therefore difficult to repeat [36].

2. Analogy: This method was first proposed by [69] as a valid alternative to expert judgment and algorithmic effort estimations [35]. Analogy-based software effort estimation is the process of identifying one or more historical projects that are similar to the project being developed and then deriving the estimates from them [16]. In certain respects, this method is a systematic form of expert judgment since experts often match similar project(s) from their own experiences in order to determine the appropriate effort estimates [35]. Estimation requires the comparability of structures of the project that has to be calculated and completed software developments serving as analogy groundwork [7].

3. Algorithmic models. These are the most popular models in the literature [19]. Algorithmic effort estimation involves the use of statistical data analysis methods to establish the parametric software effort estimation equation [35]. Examples of such models are COCOMO [13], [12] and SLIM [64], to cite the most relevant ones.

With the aim of extending the possibilities of algorithmic methods, along with dealing with the dynamic nature of the project ecosystems in a better way, many works are devoted to the use of artificial intelligence techniques, such as neural networks and fuzzy logic models, to bring more accurate software estimations [59]. Following this research path, this paper presents SEffEst, a framework based on fuzzy logic and neural networks designed to increase effort estimation accuracy on software development projects.

The remainder of the paper is structured as follows. Section 2 reviews the relevant literature about fuzzy logic and neural networks and their application in software development estimation. Next, in Section 3, the main contribution of the paper is presented including a description of the design, the architecture and the implementation of the tool. Section 4 presents the validation of SEffEst. Finally, the paper ends with a discussion of research findings, limitations and concluding remarks.

2. Using Fuzzy Logic and ANN for the estimation of software developments

The fuzzy sets theory provides a framework for the representation of the uncertainty of many aspects of

human knowledge. Nowadays, fuzzy rule based systems are being successfully applied to a wide range of real-world problems from different areas and in many real-world applications such as ships' stability or multi-criteria decision making domains. Although a system can be defined mathematically in a general sense, a fuzzy logic system representation is still preferred by engineers and researchers [51]. Due to the inherent uncertainity of software estimation the application of fuzzy logic to this area is an issue under investigation. This way, [45] proposes a goal-oriented programming model for optimizing the project selection process, applying fuzzy logic, so as to incorporate qualitative and quantitative aspects of software projects. These kind of studies are meaningful but not directly oriented towards software estimation. Regression models are one of the alternatives for software estimation. [57] proposes a regression model for software estimation improved by the application of a Sugeno fuzzy inference system. [34] applies fuzzy neural networks for project estimation based on the COCOMO dataset. The representation of COMOMO dataset and rules is the basis of other fuzzy approaches, such as [37], [44], [67], [55]. Bathla et al. [6] apply fuzzy logic for estimating the development time without taking into account traditional techniques which require long-term estimation processes. However, this approach only focuses on the development time. In a similar line, [46] predicts the effort based on the fuzzification of the estimation of the size. Fuzzifying the size of the project is a common practice for improving the estimation process (i.e. [55] or [46]).

Recent research works on Artificial Intelligence have proved that ANN paradigm can be succesfully applied to the Software Engineering domain and, in particular, to the field of estimation[4]. The promising results obtained show notable consistency and exactitude. Several predictions related to software estimation are related to ANN. The work of [1] shows an example of Bayesian regularization of an ANN for estimating the size of software developments. [47] estimates the cost of the software by means of a Wavelet ANN. [70] also predicts the reliability of the software by means of hybrid models, citing some of the most representative works in the field.

The use of ANN for effort estimation and prediction of software developments constitutes a prolific field of study. The combination of neural techniques with other methods has been widely employed: genetic algorithms [63,68], fuzzy logic [2,34] or the analogy method [35]. Genuine works on ANN are also available in the literature, such as [19] and [29] standing out as remarkable works. In the field of ANN validation, the studies of [59] and [22] based on function points can be considered a valid precedent for the research presented in this paper. Although previous research works show a promising way for software estimation, this research area is far from its maturity [38].

This paper is centered on the application of the SEffEst framework in the domain of the effort estimation for a software project. This framework combines a fuzzy logic component, for the processing and treatment of fuzzy values, and an ANN based methodology, to select the best neural model for the problem as well as the optimization of the ANN performance both in runtime and accuracy.

3. Proposed Solution

According to [19], effort estimation is one of the bases for the correct estimation of software projects. Therefore, having this information will facilitate project management permitting a better resource allocation and minimizing risks.

Many times, estimations are only based on the previous experience. If a project is quite similar in size and objectives to other one, the latter will probably need similar effort and similar time to be developed. Unfortunately, if the project to be estimated is different to the previous ones, then the previous experience will not be enough for an adequate estimation. The metrics exposed at the beginning of this paper have problems to be applied in certain dynamic domains which require high adaptability and flexibility [33].

Based on these assumptions, the SEffEst framework has been used in order to predict the effort required for a given software project. Therefore, this research focuses on the estimation of the final effort for a certain project taking into account several important features. To obtain a reliable description of

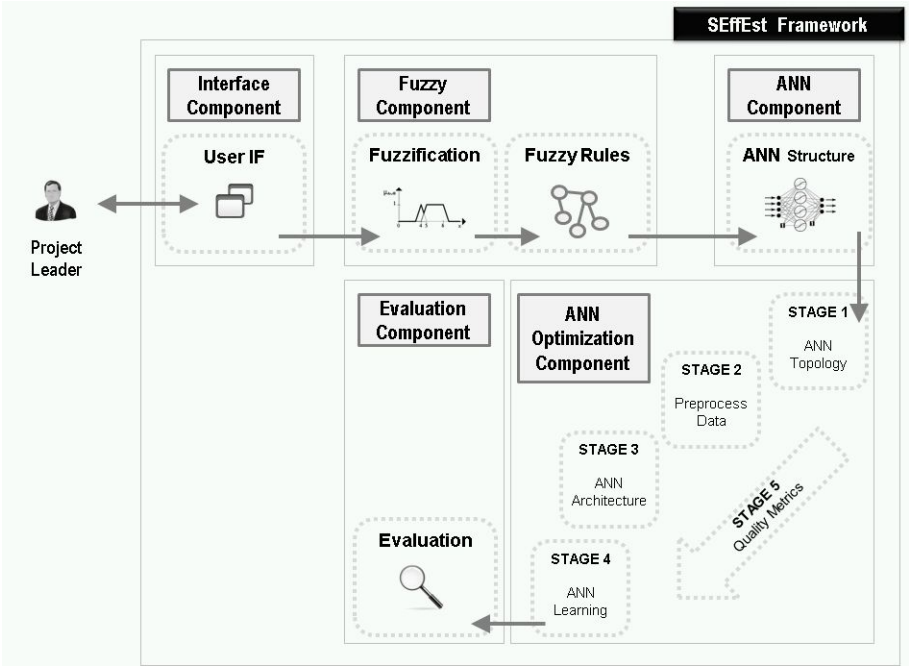

Figure 1: SEffEst framework

a software development project it has been used a subset of data, adapted from the original set of International Software Benchmarking Standards Group [†] (ISBSG).

To conduct this investigation, the original feature set has been leaked, excluding those factors related with the calculation of the function points because they need a high definition level of the project. Therefore, this research includes only those factors or features that provide essential information of the project but without a complete definition of the system.

In addition, several parameters of the original set include information that is approximate rather than fixed as well as exact or non-uniform variables. For this reason, the fuzzy logic component of SEffEst transforms these values into uniform and complete values. Thus, the fuzzy logic component is able to process incomplete data and provide approximate solutions to problems other methods find difficult to solve. These variables are specifically: project elapsed time (ET), lines of code (LC) and concurrent users (CU).

Consequently and taking into account the previous issues, the available patterns are related with data from 273 projects and they are characterized by the factors shown in the Table 1. A more detailed description of each factor can be found in [31].

The second component of the framework is an ANN based on optimization methodology. This component has been developed in order to guide the search of the best neural model for a given problem and hence improve the performance of this task both on time and accuracy. To homogenize the large number of alternatives, they have been grouped into three phases or stages, following the premise $model = pattern + architecture + algorithm$. For each of the terms of this equation, different techniques and methods will be analyzed in order to improve the final performance.

These machine learning techniques have been applied because they allow induction of knowledge. For example, they are able to generalize behaviours based on unstructured information from previous ex-

[†] http://www.isbsg.org

Table 1: List of available variables in the software estimation scenario

	Feature	***Acronym***	***Fuzzy***
Inputs (V_{inp})	Data Quality Rating	DQ	No
	Resource Level	RL	No
	Development Type	DT	No
	Development Platform	DP	No
	Language Type	LT	No
	Database Management System DBMS Used	DB	No
	Upper CASE Used	UC	No
	Project Elapsed Time	ET	Yes
	Concurrent Users	CU	Yes
	Lines of code	LC	Yes
Output (V_{out})	Summary Work Effort	SW	

amples. Furthermore, ANN can be applied to a wide range of high-complex and non-linear domains, due to their variety of design alternatives. Nevertheless, sometimes such variety of design alternatives may become a disadvantage: the lack of guidelines leads the designer to make arbitrary decisions or to use brute force techniques. Some new theoretical approaches have been proposed so as to facilitate the design process, but they have not been considered as analytical because they cannot be applied to all cases [62].

3.1. Fuzzy Logic Component

As mentioned, the information required to the estimation of software projects is not always clear. Considerations of expert estimations usually include vague expressions like "large" or "very high". For this type of variables Fuzzy Logic provides a good way of knowledge representation. Fuzzy sets are defined as:

$$A = \{(x, \mu_A(x) | x \in u\} \quad (1)$$

This fuzzy set in the universe of discourse U is characterized by a membership function $\mu_A(x)$ taking values in the interval [0.1], and can be represented as a set of ordered pairs of an element x and its membership value to the whole. After defining the fuzzy sets, the fuzzy inference rules may be used to represent the relation between these fuzzy sets. In this context, the fuzzy reasoning process is based, on the one hand, on making inferences from facts and fuzzy relations; and on the other, on a fuzzy combination of evidence which updates the accuracy of beliefs. For the fuzzy rules, the Mamdani-type fuzzy rule recommender system has been applied [53 52], due to its wide acceptance and its capability for capturing expert knowledge. These fuzzy rules, defined using a set of IF-THEN rules or Bayesian rules, are expressed as follows:

R^m: IF U_1 is ${A_1}^m$ AND U_2 IS ${A_2}^m$ AND ... U_p is ${A_p}^m$ THEN v is B^m

With $m = 1, 2...M$, where ${A_i}^m$ and B^m are fuzzy sets in $U_i \subset R$(real numbers) and $V \subset R$ respectively, $u = (u_1, u_2, ..., u_n) \in U_1 x U_2 x ... x U_n$ and $v \in V$, and $x = x_1, x_2, ..., x_n \in U$ and $y \in V$ are specific numerical values of u and v, also respectively. A rule of this type expresses a relation between the sets A and B, whose characteristic function would be $\mu_{A \to B}(x, y)$, and represents what is known as logical implication.

In order to improve the estimation process, the selected variables were analysed in order to decide the fuzzy variables and their representation (see Table 1).The aim of the fuzzy component is to provide the ANN with more refined information for the estimations. At this point, the variables selected were Project Elapsed Time (ET), Lines of Code (LC), Number of Concurrent Users (CU).

With the help of domain experts, the fuzzy sets were defined. Figure 2 depicts the fuzzy set defined

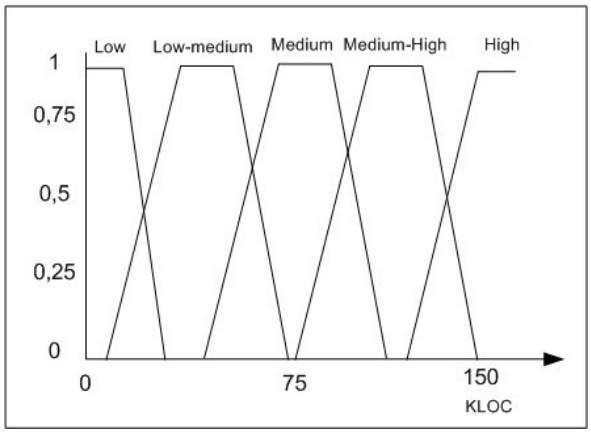

Figure 2: Size Fuzzy Set

for the size of the project in KLOC.

3.2. ANN Structure Component

The more important features of the ANN networks included in this research are detailed in the Table 2. Two complementary stopping criteria were used during the network training for all the ANN alternatives: reaching a specific number of epochs, and early stopping (ES). Thus, a part of the patterns has been assigned to the network's training (training and validation subset), whereas others have been assigned for testing the results obtained (testing subset). The input and output vector for each of the ANN are created based on the features described in Table 1.

The train set is used during the training of the ANN. The use of a validation set is a highly recommended method for stopping network training. This method monitors error on an independent set of data and stops training when this error begins to increase. This is considered to be the point of best generalization. Finally, the testing set is used to test the performance of the network. Once the network is trained the weights are frozen, the testing set is fed into the network and the network output is compared with the desired output. The other convergence criterion used (in parallel) with the early stopping is to reach a specific number of epochs. When the ANN training reaches a specific number of epochs, in this research is 81000, the training is stopped. Hence, during the training of an ANN both criteria are checked to stop the training process.

The main steps performed during the training and testing process are the following:

1. Divide the available data into training, validation and test sets.
2. Use a large number of hidden units (as much as possible).
3. Use very small random initial values (as much as possible).
4. Use a slow learning rate (as much as possible).
5. Check the convergence criteria.
   - Compute the validation error rate periodically during training.
     - Stop training if the validation error rate "starts to go up"
   - Compute the number of epochs.
     - Stop training if maximum number of epochs is reached [81000].

In the original definition of the ANN models, it has been included the extended BackPropagation algorithm. This algorithm uses the learning rate (η) and the momentum coefficient (μ) parameters for a better behaviour. There is no general rule of

Table 2: Initial features of the ANN networks

Feature	**Description**	**References**
Topology	MLP, RBF, SVM, RNN and ELN	See [27] for more details
Inputs	V_{inp}	-
Outputs	V_{out}	-
Hidden Layer and Neuron	1 Layer with 4 neurons	Based on the rules of Tarassenko in [72]
Activation Function (input-hidden-output)	f_{iden}-f_{tanh}-f_{tanh}	Based on [9,25]
Training Algorithm	Extended BackPropagation	-
Learning Parameters	μ and η for input layer (0.7 and 1) and output layer (0.3 and 0.4)	In accordance with the guidelines of [48]
Cost Function	MSE simple	-
Weight Update	*Batch*	Based on [71]
Weight Initization	Haykin Heuristic	Based on [28]
Convergence Criteria	*Epochs* [81000] and Early Stopping	-

thumb for determining the best values of μ and η. However, several researchers have developed various heuristics for their approximation. In this case, we have used a genetic algorithm to find the best values for each ANN strategy. Furthermore, the batch training mode was selected because it presents a balanced behaviour between accuracy and speed [71]), and the heuristic of Haykin has been used to initialize the weights, [28].

In regression, the network approximates the output pattern from an input pattern and by a nonlinear continuous function (F). In this case, to know the quality of a ANN model during the validation stage, the coefficient of linear correlation (r) is calculated and compared for each output variable. This coefficient reflects the relationship between the actual values of that variable and those obtained by the ANN. Moreover, this ratio gives an indication of similarity and accuracy of the response of the network after the training process:

$$r = \frac{\frac{\sum_i (x_i - \bar{x})(d_i - \bar{d})}{N}}{\sqrt{\frac{\sum_i (d_i - \bar{d})^2}{N}} \sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N}}} \quad (2)$$

where the numerator corresponds to the covariance and the denominator to the product of the standard deviations of the variables x (value obtained) and d (expected value). This statistical indicator describes the intensity of the relationship between two sets of variables x (output obtained by the network) and d (expected output), i.e., the measure of the intensity of the linear relationship between them. In particular, it shows whether the value of a variable x increments or decrements in relation to the increase or decrease of the value of another variable y. It can take values in the range $[-1, 1]$. When the value is closer to 1, a strong positive association appears. On the contrary, when the value is closer to -1, a stronger and negative association appears, i.e., an increment in the variable x produces a decrement in d. Finally if the value is zero or near zero, the variables are uncorrelated. It is said that a correlation is significant if it is between $\|0.7, 1\|$.

3.3. ANN-based Optimization Methodology

The optimization of the different neural models will be carried out by defining an optimization methodology based on a series of stages and tasks aimed to determine the best ANN model for this scenario and to improve its performance, both in time and accuracy. In order to homogenize the large number of alternatives and simplify its implementation, they have been grouped into three stages or phases: patterns, architecture and algorithm. This research describes the process of implementation and optimization of an ANN-based framework and its application

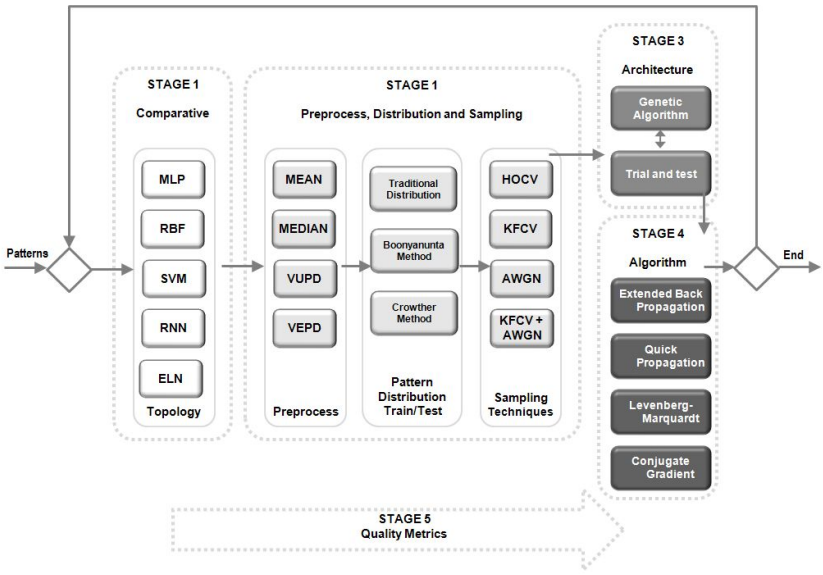

Figure 3: Optimization Methodology based in neural models

in the field of Software development effort estimation. The main aim is to improve the performance of a neural model, applied in a real scenario of software development project, looking for the best possible configuration. To achieve this goal, the optimization process is divided into five stages which are reflected in Figure 3.

The first stage aims to answer the question if the MLP is the best machine learning strategy for the domain of software effort estimation. For that reason, the performance and accuracy of various ANN strategies will be studied and compared. The second stage focuses on the different techniques and methods that allow to maximize the use of the patterns by the neural model. The third stage determines the best network architecture by implementing a Genetic Algorithm (GA) to find the best configuration of hidden neurons and layers. The fourth stage analyzes the influence of different learning algorithms, both first and second order. Finally the fifth stage is focused on ensuring the quality of the results by introducing a series of indicators based on information theory.

The procedure followed in each simulation performed in this optimization process is described below:

1. Choose the best machine learning alternative for the scenario (Stage 1).

2. Compare different preprocessing and sampling techniques and consider the best distribution for the patterns inside the train and test sets (Stage 2).

3. Locate the best configuration of hidden elements using a Genetic Algorithm (Stage 3).

4. Compare different learning algorithms, repeated 20 times with randomly initialized weights (Stage 4).

5. Calculate the quality criteria throughout the prior steps to ensure the validity of the results.

3.3.1. Stage 1. ANN topologies

This stage aims to clarify what is the best solution among different topologies based on the computational theory of the ANN. To achieve that, a comparative study will be performed to ensure the best ANN alternative for this concrete domain. The alternatives that have been included for this stage are listed in Table 3.

Table 3: Stage 1. ANN Topologies

Topology	Acronym	**Description**	**References**
MultiLayer Perceptron	MLP	This is a universal function approximator as well as easier to use and apply than other most complex ANN architectures	[33]
Radial Basis Function	RBF	This is a nonlinear hybrid networks typically containing a single hidden layer of processing elements	[60]
Support Vector Machines	SVM	These construct a hyperplane as a decision dimension which maximizes the margin of separation between the positive and negative examples in a data set	[73]
Recurrent Neural Network	RNN	This includes two types of connections, the feedforward and feedback connections, allowing information to propagate in two directions, from input neurons to output neurons and vice versa	[54]
Jordan-Elman Networks	ELN	These are dynamic neural networks which have connections from their hidden layer back to a special copy layer	[40]

3.3.2. Stage 2. Preprocess, Distribution and Sampling

This second stage focuses primarily on performing the preprocessing of the patterns to ensure the completeness of the data used. In this investigation the term ”incomplete data” refers to the unavailability of certain information in the subjects in the sample. To address these situations, several alternative methods of direct allocation of missing values have been studied [8,49]: Mean, Median, a Random Value with Uniform Probability Distribution (VUPD) and a Random Value with Estimated Probability Distribution (VEPD).

Furthermore, one of the areas that has received most attention by the research community has been the study of the relationships between the generalization capability of the network, the amount of data available and their distribution in the training and test sets [23]. To ensure the proper distribution of the patterns throughout the sets, a detailed mathematical model will be incorporated [14], which will explain the relationship between the size of the training set and predictive capacity of the network. Thanks to this study and the research carried out in [18], it has been demonstrated that the predictive power of an ANN increases until a threshold value. From this point forward, it does not compensate the use of new patterns, and therefore their production, since the reduction of the error is minimal and involves a higher cost of computation during training.

Moreover, to achieve an adequate training and validation, the division of patterns into different sets, that further reduces the amount of data, will be required. In order to address this deficiency, several statistical techniques based on multiple resampling data have been incorporated to the optimization methodology. Thus, this study included the guidelines proposed by Priddy and Keller to cope with limited amounts of data [62] as well as the rules exposed by Bishop [10] and Holmstrom and Koistinen [32] for the generation of additive noise.

The comparison made in this stage includes cross-validation techniques such as Hold-Out (HOCV) and K-fold (KFCV), the introduction of Additive White Gaussian Noise (AWGN) and the combined technique of noise and K-fold validation (KFCV + AWGN). A detailed explanation of these alternatives and an example of their use in a domain with limited information can be found in [26].

3.3.3. Stage 3. Network Architecture

To ensure an optimal generalization ability, it is essential to get a network as small as possible [50]. If the network has too many processing elements, small errors can be obtained during training, but the generalization may be poor with new data due to overtraining, i.e. it creates a very specific solution on the training data. Moreover, a network whose dimensions are too small cannot correctly resolve the problem addressed, so it will not be an optimal solution. Therefore, to achieve an optimal performance, it is essential to include in the optimization methodology heuristics to control the complexity of the ANN.

To deal with this question, a GA-based heuristic has been included trying to find the best set of hidden elements: layers and neurons. The genetic algorithms are systematic methods for solving search and optimization problems, applying the same methods of biological evolution: selection based on population, reproduction and mutation. The advantages of using evolutionary computation like genetic algorithms in conjunction with the ANN have been clearly demonstrated by the scientific community [17,21,30,66,75].

In this study the criterion used to evaluate the ability of each potential solution is the lowest cost obtained during training. The criteria and values used were chosen or fixed in accordance with the guidelines issued by Lefebvre et al. [48] and its application in practical problems. The values used are the following:

- Population: 50
- Maximum generations: 100
- Maximum time of evolution (minutes): 60
- Progression: Generational
- Termination Type: Fitness Threshold
- Selection: Roulette
- *Crossover*: One point
- *Crossover*: Probability: 0.9
- Mutation Probability: 0.01

3.3.4. Stage 4. Learning Algorithm

ANN training is based on minimizing the error function E by the variation of the weights set which define the ANN (see Equation 3). Therefore, it comes to a multi-variable optimization without restrictions, because there is no other additional requirement with respect to the function or the input variables.

$$\frac{\partial E}{\partial W} \equiv 0 \Leftrightarrow \nabla E(w*) = 0 \tag{3}$$

Therefore, it is a multivariable optimization with no restrictions because there are not additional requirements regarding the function or the input variables. The problem of optimizing a differentiable function, continuous, multivariable and without restrictions, has been studied extensively outside the field of the ANN, and the different approaches that have been followed can be applied almost directly to the problem of minimizing the error. However, there are two aspects to take into account in this specific scenario, that make a difference from the conventional optimization techniques: the high number of variables to be optimized and the large number of derivatives to be calculated in every iteration step, which makes the calculations particularly costly.

The BackPropagation algorithm calculates the Jacobian matrix, formed by the first-order partial derivatives of the error function with respect to the weights, to locate the best direction where they fit. Afterwards, the constant weights are updated in that direction. However, second-order gradient techniques use the Hessian, square matrix of second-order partial derivatives of error, with respect to the weights, so as to adapt the size of the step (step-size) in the direction of the optimal update. Expressed in mathematical form, second-order techniques try to solve Equation 4.

$$\frac{\partial^2 E}{\partial W^2} \equiv 0 \quad (4)$$

Parker proposed in 1982 [61] the original formula of the second-order method for the BackPropagation algorithm. Its main inconvenience is that it requires long computational time for finding the best solution. This is why different alternatives based on different assumptions have appeared over the years to approximate the error surface, or at least a part of it, by means of a second-order curve. One of these approaches, Quick Propagation, assumes a quadratic error surface so that the second derivative is constant while other alternatives such as Quasi-Newton and Levenberg-Marquardt use a variant of the Newton descent method.

In general, the Newton algorithms have a higher complexity, some times with limit n^2 for a size of the input vector of n ($O(n^2)$). Therefore, their application is often found in environments with architectures and input vectors with adjusted sizes.

In Table 4 the different learning algorithms included in this research are exposed.

3.3.5. Stage 5. Quality Metrics

The last phase of the framework is aimed at measuring the quality and validity of the results of the various ANN alternatives analyzed. The performance parameter calculated in each ANN, the coefficient of linear correlation (r), enables the determination of the functioning and performance of an ANN. However they do not allow for the determination of which is the better choice when various alternative models with similar results appear. To facilitate this decision, quality metrics have been included to evaluate and compare from the statistical point of view the generalization ability of the designed neural models.

Therefore, this stage includes two statistical estimates that indicate the measure of goodness of fit of a statistical model estimated. These quality indicators are the Akaike Information Criterion (AIC) developed by [3] and the principle of Minimum Description Length (MDL) proposed by [65]. Both criteria are based on the calculation of the prediction error (EP), that can be expressed as the sum of two terms [11]:

$$EP = training\ error + complexity\ term \quad (5)$$

where the complexity term represents a penalty which increases with the degrees of freedom of the neural model. Thus, it is based on two computation elements, one which decreases when the fit to the data is improved (accuracy or fitness term), and another which increases with the number of parameters (term for the penalty of the complexity of the model).

The use of these parameters in the field of ANN permits the selection of the optimal neural model for a given problem from a number of candidates. The AIC indicator has been used by researchers for different aims, e.g. to reduce the subjectivity of the choice between one architecture or another, to determine the number of hidden neurons, and even to design committees of networks [56,78]. The MDL indicator has enabled the reduction of the complexity of different architectures, minimizing the number of weights in domains with scarce data, and determining the number of neurons and hidden layers for the optimal model [77,76].

Finally, to reduce the influence of random partition of the data and the initial weights of the network, all the experiments will be repeated 20 times. This ensures that Type I and Type II (probability of accepting the null hypothesis when it is false, known as equal false) errors will be appropriate and reproducibility (the probability that two executions of the same comparison method produce the same results, known as stability) will be greater than the traditional alternative.

4. Evaluation and Methodology Validation

In this section, the evaluation performed is depicted. Firstly, the fuzzy components defuzzify the values of the fuzzy input variables in order to feed the ANN component. Lastly, the ANN-based optimization methodology locates the best ANN model for the problem of this research.

Table 4: Stage 4. List of Learning Algorithms

Learning Algorithm	Acronym	Parameters
Gradient	GR	Parameter η in hidden and output layer (with Genetic Algorithm)
Extended BackPropagation	EBP	Parameter η and μ in hidden and output layer (with Genetic Algorithm)
Quick-Propagation	QP	Parameter η and μ in hidden and output layer (with Genetic Algorithm)
Conjugate Gradient	CG	Second order method
Levenberg Marquardt	LM	Second order method

4.1. Fuzzy logic transformation

First of all, the data about the projects to be evaluated are processed by the fuzzy logic component. The data corresponding to the fuzzy variables is processed, the fuzzy rules are applied and, finally, the results are defuzzified in order to determine the input values for the neural network. In this way, the values of size, elapsed time and concurrent users are determined and incorporated to the input vector along with the non-fuzzy input values.

4.2. Stage 1. Comparison of alternatives to MLP

The basic architecture of the ANNs used in this study are shown in Table 2 and the others ANN topologies are detailed in Table 5. The initial learning rule for the ANNs is the Extended BackPropagation, with or without the parameters, except for the Support Vector Machine (SVM) topology. The optimal values for these parameters in each ANN were approximated through an adaptative process using the genetic algorithm component.

- RBF: Extended BackPropagation Learning. η=1.0 y μ=0.7 for the output layer. .
- RNN: Extended BackPropagation Learning. η=0.01 y μ=0.7 for the hidden layer. η=0.1 y μ=0.7 for the output layer.
- SVM: Learning with η=0.01 for the output layer.
- MLP: Extended BackPropagation Learning. η=0.9 y μ=0.8 for the hidden layer. η=0.1 y μ=0.7 for the output layer.
- ELN: Extended BackPropagation Learning.η=0.1 y μ=0.8 for the hidden layer. η=0.1 y μ=0.8 for the output layer.
- SOFM: Extended BackPropagation Learning. η=0.9 y μ=0.7 for the hidden layer. η=0.1 y μ=0.7 for the output layer.

Table 6 shows the average precision (correlation factor) for each of the alternatives after twenty repetitions. The MLP is the best machine learning in all cases, reaching the threshold of accuracy with 27000 epochs. In second place the RNN obtains an accuracy slightly lower than the MLP ,-0.02, with 9000 epochs but with a more unstable behaviour. The remaining proposals (RBF, ELN and SVM) fit worse to the problem with a lower level of precision. Finally, in all the classifiers analyzed, the performance decays above the 29000 epochs, indicating signs of overlearning.

In order to corroborate the former conclusions, an analysis focused on the quality measure of each learning machine was performed. To compare the goodness of fit to the data, the statistical criteria based on information theory AIC and MDL are detailed in Table 6. As shown, the neural networks based on the MLP have a better performance and adjust to the data, i.e. with lower values for these statistical indicators.

Table 5: Overview of ANN and cost function strategies included in the stage 1.

Method	Acronym	Description
Radial Basis Function	RBF	Conscience full competitive rule. Euclidean metric. Cluster Centers=70. 10-0-1 topology.
Recurrent Neural Network	RNN	Partially recurrent. 10-1-1 topology.
Support Vector Machine	SVM	Kernel Adatron algorithm. Step size 0.01. 10-0-1 topology.
MultiLayer Perceptron	MLP	10-4-1 topology
Elman Network	ELN	Inputs feed the context units. 0.8 time constant. Momentum learning rule 10-1-1 topology.
Self-Organizing Map	SOFM	Square Kohonen Full method. Rows=5, Columns=5, Starting Radius=2, Final Radius=0. 10-1-1 topology.

Table 6. Stage 1. Results with AIC and MDL criteria (20 runs)

ANN	Hit rate r	AIC	MDL	Epochs
RBF	0.6539	413.7332	454.1413	23674
SVM	0.6973	383.3616	278.4807	26937
RNN	0.7161	193.4535	180.9573	29863
ELN	0.5426	493.8069	417.4704	29541
MLP	**0.7413**	**101.2314**	**99.9212**	27850

Finally, as it has been demonstrated, the MLP is superior in the average of the results and, in addition, it presents a better fit to the data from the point of view of the indicators AIC and MDL. Therefore, in the following paragraphs, the MLP with 27850 epochs will be discussed to optimize its use within this domain.

4.3. Stage 2. Preprocessing, Distribution and Sampling

- Task 1. Missing data imputation techniques

This stage preprocesses the patterns to ensure the completeness of the data. In the simulations of the Stage 1, the arithmetic mean was used to complete the missing data and to perform the experiments, despite being the simplest technique, it usually has a correct behaviour for this kind of problems. Table 7 shows the results obtained for each alternative used to complete the missing values in patterns.

Table 7. Stage 2. Results obtained with the preprocessing techniques

Technique	Best Result
Mean	**0.7521**
Median	0.7513
VUPD	0.7302
VEPD	0.7191

In this case, the technique with best results is still using the "Mean" to complete the missing information. This technique is a simple non-random allocation of an unknown value and it consists of the assignation of the average value for a certain variable, in the cases that have numerical value. For categorical variables the corresponding distribution model has been included.

- Task 2. Techniques for Patterns Distribution

The typical distributions of 50/50 or 80/20 to train and test sets exposed in [71] do not always fit neatly into every problems. For that reason and to find the best distribution, the guidelines described in [18] will be followed and the results will be validated using the mathematical model proposed in [14].

Figure 4 displays the evolution of the MLP accuracy depending on the size of the train set. The results obtained agree with the tenets of Crowther and Fox in [18]. From the trend line associated with the results, the predictive power grows rapidly to the point A and then begins to decelerate to the point

B. However, in the last stage, from B till the end, no significant improvements were produced, arising stagnation and even deterioration in the rate of success. In this case, the maximum value of success is consistent with the assertions postulated by Boonyanunta and Zeephongsekul in [14], because the ratio for the train set is fixed to 70% of the available patterns. From this point B onwards, the classifier performance stagnates and the results are stable. The advantage of this analysis is that it facilitates to know the amount of data to reach acceptable results, because the improvement that will be achieved towards this point will be minimal.

Therefore, this situation permits initially to reduce the cost of getting additional data to increase the size of the training set and secondly to accelerate the convergence of the learning process. In this sense, it is necessary to take into account if the cost associated with the generation of more patterns will compensate the effort to earn a small percentage of accuracy, $\approx$1%. However, the problem associated with this technique is the subjectivity when determining what level of improvement is despicable. Thus, it is necessary to consider other criteria such as the cost of extraction of information, time required to develop and the train of the neural models, etc.

Table 8. Stage 2. Estimated values by the mathematical model for the distribution of the patterns

T(r)	p in T	k	P0(r)	Max.(r)	p in Max.
0.7512	501	0.002951	0.5223	0.7121	149

The mathematical model of Boonyanunta and Zeephongsekul was probed to test the evolution of the ANN with a higher number of patterns (greater than those actually available).

This model is summarized in the equation:

$$P(p) = T(1 - e^{kp}) + P(0)e^{kp} \quad (6)$$

where T is the threshold of efficiency, k is the rate of improvement in predictive power per unit increase in efficiency and $P(0)$ is the predictive power when no data are provided to the training process. Three points were chosen to approximate the model described: 50, 100 and 149 patterns. The values shown in the Table 8 were obtained, on the one hand, from the values obtained by the MLP and, on the other, with an optimization method implemented in the Solver method. The Solver method shows the maximum threshold that could reach the classifier (*T(%)*) by increasing the number of patterns (*p*). The column *Max. (%)* indicates the highest level reached by the classifier and the column *p in Max.* sets the maximum number of patterns used in that case. However, as with the investigation of reference, the model has a lag in its predictions and is not adjusted correctly for a few data. The trend line superimposed on Figure 4 shows that the threshold has not been reached. The trend lines shown are logarithmic, because it is a curved line that perfectly fits the data, being very useful when the rate of change of the data increases or decreases quickly and then stabilizes. The A and B points are estimated by calculating the average variation and the standard deviation obtained in the simulations.

Finally, based on the analysis of the results, the optimal distribution for train/test sets is 70/30.

- Task 3.Sampling Techniques

In some scenarios due to the nature of the process that recreates the data, the collection and labeling of the information becomes an intensive and complicated task. In order to improve the results without additional testing, various techniques from the field of statistics, used in estimation and classification problems, have been analyzed. This techniques allow the resampling of the original set of patterns into a bigger one. Table 9 details the different options included in this research.

The results of the simulations for this stage are shown in Table 9. Although the method HOCV, is used frequently because of its speed, it presents the problem of wasting information because only a part of the data is used to train the network. When the amount of data available is limited, as in the domain of this research, it is necessary to consider other alternatives to maximize and optimize the use of

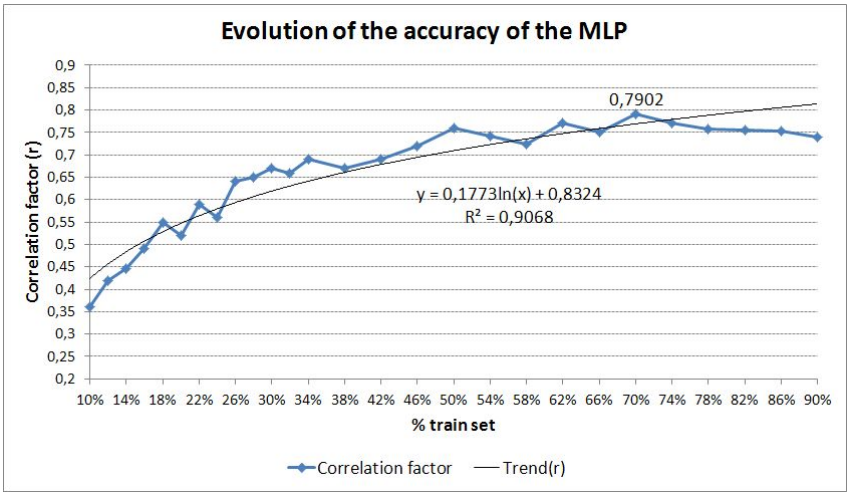

Figure 4: Stage 2. Evolution of the accuracy of the MLP related to the training set size

Table 9: Stage 2. List of experiments performed with the sampling techniques

Technique	Description	Hit rate r	AIC	MDL
HOCV	Hold-Out Cross Validation	0.7902	120.2123	156.1146
$KFCV_5$	K-fold Cross Validation with 5 sets	0.8201	108.5876	92.7751
$KFCV_{10}$	K-fold Cross Validation with 10 sets	0.8305	101.2543	111.9974
$AWGN_{50}$	Addition of 50 patterns with noise	0.8203	93.4236	97.8823
$AWGN_{100}$	Addition of 100 patterns with noise	0.8328	97.5025	101.7039
$K_{10}AWGN_{100}$	$KFCV_{10}$ and $AWGN_{100}$ combination	**0.8405**	**91.2352**	**85.1908**

the patterns. The technique $K_{10}AWGN_{100}$ obtains the best results, with superior performance from 1000 epochs. The maximum improvement obtained against the HOCV alternative is reached with 27000 epochs (the average number of epochs required is 21547), coinciding with the point of maximum accuracy of the MLP: 0.7911. Within the other alternatives, it should be highlighted the introduction of 100 noise patterns and the variant of KFCV with 10 sets, because they are the alternatives with a stable behaviour.

Thus, for the following steps, it shall be used the MLP with the configuration $K_{10}AWGN_{100}$. Thanks to this technique, it could be mitigated the influence of the random distribution of patterns of the KFCV technique. In addition, each simulation of the alternative $K_{10}AWGN_{100}$ will be repeated several times. This combination results in the $20*10cv$ strategy, also called repeated K-fold cross validation, indicating that each simulation will run 20 times with 10 data sets of similar size, using nine to train and one to test the network [20].

4.4. Stage 3. Network architecture

This stage is responsible for optimizing the architecture of the ANN by controlling the hidden elements, layers and neurons. The objective is to improve the performance and the quality of the neural model. Firstly, an adaptative method like GA will be used in order to localize within the possible ANN architectures those which the best behaviour. Lastly, a trial and error process oriented to measure the performance of the selected topologies will be used to validate the candidates.

From the point of view of the effectiveness of the

Table 10: Stage 4. List of learning algorithms studied

Algorithm	Description
EBP	Learning method with optimization parameters (hidden and output layers) η =1.0-0.4 and μ =0.7-0.4
LM	Method of second-order improvement for the gradient.
QP	Method of second-order improvement with parameters (hidden and output layers) η =1.0-0.4 and μ =0.7-0.4
GC	Method of second-order improvement for the gradient

learning, the creation of topologies with many layers is not convenient. It is always better to solve the problem with fewer layers (i.e. A MLP with one or two hidden layers) so as to achieve a faster training. In this case, the starting point is the basic MLP architecture obtained in the previous stages; so initially only one hidden layer will be used. If the error is not considered appropriate, a second layer will be added to the MLP. Since a MLP with two layers is considered an universal approximator, the use of more layers may not be necessary.

In the first step (using a GA), the starting topology includes only one hidden layer composed by four neurons: C_1 N_4. Table 11 shows the best topologies for hidden layers and neurons selected by means of the GA (marked as GA). The first one, C_1N_8, has one hidden layer with eight neurons, while the second one, $C_1N_4 - C_2N_5$, has two hidden layers with four and five neurons. In the last case (two hidden layers), as the results obtained are not better and the computational time is higher, no more complex topologies have been searched (i.e. including more hidden layers). The behaviour of the above mentioned proposals have been studied through a process of trial and error, also including a wider range of configurations. The results obtained by each architecture are shown in Table 11.

Table 11. Stage 3 results. Comparison of architectures

Architecture	Hit rate r	AIC	MDL	Epochs
10-4-1	0.8405	88.6105	41.8513	27850
9-8-1(GA)	**0.8746**	**52.7613**	**53.4398**	**19781**
10-9-1	0.8361	80.4874	38.7180	20518
10-4-3-1	0.8460	126.0079	75.84253	27796
10-4-5-1 (GA)	0.8633	90.1587	95.43103	29083
10-4-6-1	0.8324	117.8848	72.70923	32855

4.5. Stage 4. Learning algorithm

This stage studies different alternatives oriented to the numerical optimization of the learning algorithm. The aim of the learning algorithm is to locate the global minimum on the error surface in the weight space. The first or second order algorithms attempt to optimize and accelerate the search by means of different approaches as well as the way in which weights are updated, e.g using dynamic adaptation of the parameters or the second derivative of the error. Table 10 details all the alternatives proposed in this section.

The results obtained are shown in Table 12. The best proposal is the GC algorithm with a slightly better performance than the classical EBP algorithm.

Table 12. Stage 4. Results obtained for the learning algorithms

Algorithm	Hit rate r	AIC	MDL	Epochs
EBP	0.8746	52.7613	53.4398	19781
LM	0.9099	41.0923	44.6117	1694
QP	0.9070	46.5031	52.7836	18907
GC	**0.9140**	**33.7234**	**37.4981**	**16121**

4.6. Stage 5. Quality Metrics

This stage includes the AIC and MDL criteria that have been used in the previous stages of the optimization methodology. The use of these parameters in the field of ANN facilitates the selection of the optimal neural model for a given problem from a number of candidates.

4.7. Summary and Discussion

The evolution of the accuracy and the number of epochs performed in each stage are listed in Table 13. The columns "% Var." and "% Var. Total" show the percentage of variation from the previous stage and the total percentage of variation between the last stage and the initial MLP. The total average improvement is 23.38% and reducing the number of epochs reaches 45.18%. In addition, the quality metrics introduced throughout all phases of the methodology adds an additional component to ensure the validity of the conclusions reached.

Authors have tested a similar framework, using ANNs but without the fuzzy component, in similar regression problems where it was necessary to predict a value from a set of input variables. Table 14 includes a comparison between SEffEst and other similar systems proposed by the authors. The previous research work [24] carried out by the authors, obtained a hit rate of 0.8525. It implies that SEffEst improves the hit rate by 6.15%. The reason for this improvement can be rooted in the fact that SEffEst incorporates the fuzzy logic component that has allowed the inclusion of more test cases in which some information was vaguely quantified. With respect to [27], despite the domains are not comparable (software estimation vs ballistic impact), it can be observed that SEffEst hit rate is quite similar to the two scenarios proposed by [27]. Firstly, all hit rates can be considered as very high, remarking the good results achieved by SEffEst. Lastly, despite the research works are different, they are based on the evolution of the optimization methodology and its implementation into a concrete framework. Thus, it can be assumed that the optimization methodology can be applied to different domains obtaining remarkable results. It can also be concluded that the inclusion of new elements to the methodology (as fuzzy logic components in SEffEst) permits the improvement of the results.

5. Conclusions

Adequate and reliable effort estimation from a reduced data set in the early phases of a software project represents a competitive advantage and provides valuable information in order to make decisions.

There are several research lines related to software estimation. In this paper, fuzzy logic and ANN have been explored with a view to improve the estimation process. ANN technique takes into account a wide number of design factors, requiring an adequate test set that provides the accurate training of the network in order to obtain optimal results. The selection of the test set and the training process require knowledge and experience due to the inherent difficulty to these tasks. Fuzzy logic, on the other hand, permits the representation of knowledge using vague concepts in terms near to the language of the experts. This paper presents SEffEst, a framework based on an optimization methodology for ANN models oriented to effort estimation of software projects considering fuzzy input values. The main aim of this research is to improve the performance of a neural model by finding the best possible configuration for the problem to be solved and setting up the adequate fuzzy sets in order to ease the definition of the input values. The proposed framework manages to find the best neural model for the problem at hand, and thus improves the performance of the ANN both in time and precision.

To validate the proposed approach, SEffEst has been designed based on a set of parameters available at the early phases of the project. The parameters that imply a deep knowledge of the software project have been discarded in order to allow the early estimation, and the values of lines of code, elapsed time and concurrent users have been fuzzified.

Research results show that an accurate Fuzzy logic - ANN architecture for effort estimation has been obtained. The mean correlation factor obtained with SEffEst is 0.9140, improves 6.15 % the previous work of the authors. That is a promising value, because the effort estimation of a software project depends on a considerable number of variables.

Future research is focused on the application of SEffEst in stages of the project where more variables are known. This offers a promising perspective, in order to play with more fuzzy variables and more relevant and accurate information about the project.

Table 13: Stage 4. Summary of the results obtained

	Stage 1	Stage 2	% Var	Stage 3	% Var	Stage 4	% Var	% Total Var
Hit rate r	0.7413	0.8405	13.38	0.8746	4.05	0.9140	4.51	23.30
Epochs	29410	27850	-5.30	19781	-28.97	16121	-18.50	-45.18

Table 14: Comparison of SEffEst with other systems

Reference	Hit rate r	Problem type	Main Features
SEffEst	0.9140	Regression of one variable	Fuzzy Logic and ANN framework for software development domain. 5 stage optimization methodology
[24]	0.8525	Regression of one variable	ANN framework for software development domain). 5 stage optimization methodology
[27]	0.9000 and 0.9300	Scenario 1 for regression of two variables	ANN framework for impact ballistic domain. 4 stage optimization methodology
[27]	0.9200 and 0.9100	Scenario 2 for regression of two variables	ANN framework for impact ballistic domain. 4 stage optimization methodology

1. K.K. Aggarwal, Y. Singh, P. Chandra, and M. Puri. Bayesian regularization in a neural network model to estimate lines of code using function points. *Journal of Computer Science*, 1(4):505–509, 2005.
2. M. A. Ahmeda, M. O. Saliub, and J. AlGhamdia. Adaptive fuzzy logic-based framework for software development effort prediction. *Information and Software Technology*, 47(1):31–48, 2005.
3. H. Akaike. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on*, 19(6):716–723, 1974.
4. H. Al-Sakran. Software cost estimation model based on integration of multi-agent and case-based reasoning. *Journal of Computer Science*, 2(3):276–282, 2006.
5. M. Azzeh, D. Neagu, and P.I. Cowling. Fuzzy grey relational analysis for software effort estimation. *Empirical Software Engineering*, 15(1):60–90, 2010.
6. R. Bathla, S. Singh, M. Vishav, and R. Gupta. Innovative scenario in software development effort estimation based on a new fuzzy logic model. *International Journal of Information Technology and Knowledge Management*, 2(2):361–364, 2010.
7. A. Baumeister and M. Ilg. Activity driven budgeting of software projects. *International. Journal of Human Capital and Information Technology Professionals*, 1(4):14–30, 2010.
8. E.M.L. Beale and R.J.A. Little. Missing values in multivariate analysis. *Journal of the Royal Statistical Society. Series B (Methodological)*, 37(1):129–145, 1975.
9. C. Bisagni, L. Lanzi, and S. Ricci. Optimization of helicopter subfloor components under crashworthiness requirements using neural networks. *Journal of Aircraft*, 39(2):296–304, 2002.
10. C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, 1996.
11. C.M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7:108, 1995.
12. B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice-Hall, Upper Saddle River, NJ, 2000.
13. B.W. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.
14. N. Boonyanunta and P. Zeephongsekul. Predicting the relationship between the size of training sample and the predictive power of classifiers. *Knowledge-Based Intelligent Information and Engineering Systems*, 3215:529–535, 2004.
15. L. F. Capretz and V. Marza. Improving effort estimation by voting software estimation models. *Advances in Software Engineering*, online:doi:10.1155/2009/829725, 2009.
16. N.H. Chiu and S.J. Huang. The adjusted analogy-

based software effort estimation based on similarity distances. *Journal of Systems and Software*, 80(4):628–640, 2007.
17. S.-B. Cho and K. Shimohara. Evolutionary learning of modular neural networks with genetic programming. *Applied Intelligence*, 9:191–200, 1998.
18. P. Crowther and R. Cox. Accuracy of neural network classifiers as a property of the size of the data set. *Knowledge-Based Intelligent Information and Engineering Systems*, 4253:1143–1149, 2006.
19. I. F. de Barcelos Tronto, J. D. Simoes da Silva, and N. Sant'Anna. An investigation of artificial neural networks based prediction systems in software project management. *The Journal of Systems and Software*, 81(3):356–367, 2008.
20. T.G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923, 1998.
21. N. Durand, J.-M. Alliot, and F. Medioni. Neural nets trained by genetic algorithms for collision avoidance. *Applied Intelligence*, 13:205–213, 2000.
22. G. R. Finnie, G. E. Wittig, and J. M. Desharnais. A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. *Journal of Systems and Software*, 39(3):281– 289, 1997.
23. G. Foody, M.B. McCulloch, and W.B. Yates. The effect of training set size and composition on artificial neural network classification. *International Journal of Remote Sensing*, 16(9):1707–1723, 1995.
24. A. Garcia, I. Gonzalez, R. Colomo-Palacios, J. L. Lopez, and B. Ruiz. Methodology for software development estimation optimization based on neural networks. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 9(3):384 –398, 2011.
25. A. Garcia-Crespo, B. Ruiz-Mezcua, D. Fernandez, and R. Zaera. Prediction of the response under impact of steel armours using a multilayer perceptron. *Neural Computing & Applications*, 16(2):147–154, 2006.
26. I. Gonzalez-Carrasco, A. Garcia-Crespo, B. Ruiz-Mezcua, and J. Lopez-Cuadrado. Dealing with limited data in ballistic impact scenarios: an empirical comparison of different neural network approaches. *Applied Intelligence*, 35(1):89–109, 2011.
27. I. Gonzalez-Carrasco, A. Garcia-Crespo, B. Ruiz-Mezcua, and J.L. Lopez-Cuadrado. An optimization methodology for machine learning strategies and regression problems in ballistic impact scenarios. *Applied Intelligence*, 36(2):424–441, 2011.
28. S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994.
29. A. Heiat. Comparison of artificial neural network and regression models for estimating software development effort. *Information and Software Technology*, 44(15):911–922, 2002.
30. C.E. Henderson, W.D. Potter, R.W. McClendon, and G. Hoogenboom. Predicting aflatoxin contamination in peanuts: A genetic algorithm/neural network approach. *Applied Intelligence*, 12:183–192, 2000.
31. P. R. Hill. *Practical project estimation : a toolkit for estimating software development effort and duration*. International Software Benchmarking Standards Group, 2001.
32. L. Holmstrom and P. Koistinen. Using additive noise in back-propagation training. *Neural Networks, IEEE Transactions on*, 3(1):24–38, 1992.
33. K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
34. S.-J. Huang and N.-H. Chiu. Applying fuzzy neural network to estimate software development effort. *Applied Intelligence*, 30(2):73–83, 2009. 10.1007/s10489-007-0097-4.
35. S.J. Huang and N.H. Chiu. Optimization of analogy weights by genetic algorithm for software effort estimation. *Information and Software Technology*, 48(11):1034–1045, 2006.
36. S.J. Huang, N.H. Chiu, and L.W. Chen. Integration of the grey relational analysis with genetic algorithm for software effort estimation. *European Journal of Operational Research*, 188(3):898–909, 2008.
37. X. Huang, L.F. Capretz, J. Ren, and D. Ho. A neuro-fuzzy model for software cost estimation. In *Quality Software, 2003. Proceedings. Third International Conference on*, pages 126 – 133, 2003.
38. X. Huang, D. Ho, J. Ren, and F. Capretz. A soft computing framework for software effort estimation. *Soft Comput*, 10(2):170–177, 2006.
39. X. Huang, D. Ho, J. Ren, and L.F. Capretz. Improving the cocomo model using a neuro-fuzzy approach. *Applied Soft Computing*, 47(1):31–48, 2007.
40. M. Jordan. Serial order: A parallel distributed processing approach. Technical report, University of California, San Diego, 1986.
41. M. Jørgensen. A review of studies on expert estimation of software development effort. *The Journal of Systems and Software*, 1/2:37–60, 2004.
42. M. Jørgensen and T. Halkjelsvik. The effects of request formats on judgment-based effort estimation. *Journal of Systems and Software*, 83(1):29–36, 2010.
43. M. Jørgensen and D.I. Sjoberg. The impact of customer expectation on software development effort estimates. *International Journal of Project Management*, 24(4):317–325, 2004.
44. S. Kad and V. Chopra. Fuzzy logic based framework for software development effort estimation. *Research Cell: An International Journal of Engineering Sciences*, 1:330 – 342, 2011.
45. I. Kim, S. Shin, Y. Choi, N. M. Thang, E. R. Ramos, and W.-J. Hwang. Development of a project selection

method on information system using anp and fuzzy logic. *World Academy of Science, Engineering and Technology*, 53:411–416, 2009.
46. J.N.V.R.S. Kumar, T.G. Rao, Y.N. Babu, S. Chaitanya, and K. Subrahmanyam. A novel method for software effort estimation using inverse regression as firing interval in fuzzy logic. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 4, pages 177 –182, 2011.
47. K.V. Kumar, V. Ravi, M. Carr, and N. Raj-Kiran. Software development cost estimation using wavelet neural networks. *Journal of Systems and Software*, 81(11):1853–1867, 2008.
48. C. Lefebvre, C. Fancourt, J. Principe, and G. J. *Neuro Solutions Documentation*. 2007.
49. R.J.A. Little and D.B. Rubin. *Statistical analysis with missing data*. John Wiley & Sons, Inc., 1986.
50. H. Liu and R. Setiono. Incremental feature selection. *Applied Intelligence*, 9(3):217–230, 1998.
51. Jie Liu, Wilson Wang, Farid Golnaraghi, and Eric Kubica. A neural fuzzy framework for system mapping applications. *Know.-Based Syst.*, 23:572–579, August 2010.
52. E.H. Mamdani. Fuzzy sets and applications: selected papers by l.a. zadeh, r.r. yager, s. ovchinikov, r.m. tong, h.t. nguyen (eds.). *Knowledge-Based Systems*, 1(2):121, 1988.
53. E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Human-Computer Studies*, 51(2):135–147, 1999.
54. D.P. Mandic and J. Chambers. *Recurrent Neural Networks for Prediction: Learning Algorithms,Architectures and Stability*. John Wiley & Sons, Inc., 2001.
55. A. Mittal, K. Parkash, and H. Mittal. Software cost estimation using fuzzy logic. *SIGSOFT Softw. Eng. Notes*, 35:1–7, 2010.
56. N. Murata, S. Yoshizawa, and S.-i. Amari. Network information criterion — determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*, 5:865, 1994.
57. A. B. Nassif, L. F. Capretz, and D. Ho. Estimating software effort based on use case point model using sugeno fuzzy inference system. In *23rd IEEE International Conference on Tools with Artificial Intelligence*, pages 393–398, 2011.
58. A.L.I. Oliveira, P.L. Braga, R.M.F. Lima, and M.L. Cornélio. Ga-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation. *Information and Software Technology*, 52(11):1155–1166, 2010.
59. H. Park and S. Baek. An empirical validation of a neural network model for software effort estimation. *Expert Sytems with Applications*, 35(3):929–937, 2008.
60. J. Park and I.W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.
61. D. Parker. Learning logic, invention report. Technical report, Office of Technology Licensing, Stanford, 1982.
62. K. Priddy and P.E. Keller. *Artificial Neural Networks: An Introduction*. SPIE Press, 2005.
63. J.C.F. Pujol and R. Poli. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, 8:73–84, 1998.
64. L. H. Putnam. A general empirical solution to the macro sizing and estimating problem. *IEEE Transaction on Software Engineering*, 4(4):345–361, 1978.
65. J. Rissanen. Modeling by shortest data description. *Automatica*, 14:445–471, 1978.
66. N.K. Roy, W.D. Potter, and D.P. Landau. Designing polymer blends using neural networks, genetic algorithms, and markov chains. *Applied Intelligence*, 20:215–229, 2004.
67. V. Sharma and H. K. Verma. Optimized fuzzy logic based framework for effort estimation in software development. *International Journal of Computer Science Issues*, 7(2):30–38, 2010.
68. K.K. Shukla. Neuro-genetic prediction of software development effort. *Information and Software Technology*, 42(10):710–713, 2000.
69. R. Sternberg. Component processes in analogical reasoning. *Psychological Review*, 84(4):353–378, 1997.
70. Y.S. Su and C.Y. Huang. Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models. *Journal of Systems and Software*, 80(4):606–615, 2007.
71. K. Swingler. *Applying Neural Networks. A Practical Guide.* Academic Press, 1996.
72. L. Tarassenko. *A guide to neural computing applications*. Arnol / NCAF, 1998.
73. V.N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.
74. T. Wijayasiriwardhane, R. Lai, and KC Kang. Effort estimation of component-based software development–a survey. *IET software*, 5:216, 2011.
75. C. L. Xie, J. Y. Chang, X.C. Shi, and J. M. Dai. Fault diagnosis of nuclear power plant based on genetic-rbf neural network. *Int. J. Comput. Appl. Technol.*, 39(1/2/3):159–165, 2010.
76. S. Xu and L. Chen. A novel approach for determining the optimal number of hidden layer neurons for fnns and its application in data mining. In *5th International Conference on Information Technology and Applications*, pages 23–26, 2008.
77. M. Zhao, Z. Chen, and J. Hjerrild. Analysis of the behaviour of genetic algorithm applied in optimization of electrical system design for offshore wind farms. In anonymous, editor, *IEEE Industrial Electronics,*

IECON 2006 - 32nd Annual Conference on, pages 2335–2340, 2006.
78. Z. Zhao, Y. Zhang, and H. Liao. Design of ensemble neural network using the akaike information criterion. *Engineering Applications of Artificial Intelligence*, 21(8):1182– 1188, 2008.
79. Z. Zia, A. Rashid, and K.U. Zaman. Software cost estimation for componentbased fourth-generation-language software applications. *Software, IET*, 5(1):103–110, 2011.