SOFTWARE DEVELOPMENT EFFORT ESTIMATION BY USING ARTIFICIAL NEURAL NETWORKS

(YAPAY SİNİR AĞLARI İLE YAZILIM PROJELERİNİN EFORUNUN TAHMİNLENMESİ)

by

Tuğçe UĞURLU, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

INDUSTRIAL ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

June 2017

This is to certify that the thesis entitled

SOFTWARE DEVELOPMENT EFFORT ESTIMATION BY USING ARTIFICIAL NEURAL NETWORKS

prepared by **Tuğçe UĞURLU** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering** at the **Galatasaray University** is approved by the

Examining Committee:

Assoc. Prof. Dr. S. Emre ALPTEKİN (Supervisor)
Department of Industrial Engineering
Galatasaray University -------------------------

Assoc. Prof. Dr. Müjde GENEVOIS
Department of Industrial Engineering
Galatasaray University -------------------------

Assoc. Prof. Dr. Seda YANIK
Department of Industrial Engineering
İstanbul Technical University -------------------------

Date: -------------------------

ACKNOWLEDGEMENTS

I have learned a lot and really enjoyed while working on this thesis. I would like to sincerely thank to all those who helped me with their valuable support during the entire process of this thesis.

I am deeply grateful to my superviser, Associate Professor S. Emre Alptekin for his helpful guidance, support and contribution. The completion of this thesis could not have been possible without his endless support.

I want to thank Emrah Yayıcı for his encouragement and helpful advices. I also want to thank Yapı Kredi Bank Project Management Departmant for their generous support during data gathering process.

Finally, I would like to offer my special thanks to my husband Onur Altuntaş. He has been extraordinarily tolerant and supportive during my studies.

June 2017
Tuğçe UĞURLU

TABLE OF CONTENTS

LIST OF SYMBOLS

ANN	**:** Artificial Neural Network
BP	**:** Back Propagation
COCOMO	**:** The Constructive Cost Model
ECF	**:** Environmental Complexity Factors
FP	**:** Function Point
FPA	**:** Function Point Analysis
GSC	**:** General System Characteristics
SLOC	**:** Source Lines of Code
KLOC	**:** Lines of codes in thousands
LM	**:** Levenberg - Marquardt
m/d	**:** Man day
MRE	**:** Magnitude of Relative Error
MMRE	**:** Mean Magnitude of Relative Error
TCF	**:** Technical Complexity Factors
UAW	**:** Unadjusted Actor Weight
UCP	**:** Use Case Point
UUCW	**:** Unadjusted Use Case Weight

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The software industry is growing rapidly and gaining importance all over the world. Nearly all companies and institutions from various industries have software projects to develop new applications and platforms. As required with every project, accurate effort estimation has become a crucial problem for the companies, especially for project managers.

Since 1970s different methods and models have been developed for estimating software projects' efforts. The first milestone model was COCOMO, which is a constructive method proposed in the late 1970s. Many different models followed, the most popular and usable models being Function Point and Use Case Point. After 2000s, due to advances in technology, Artificial Neural Networks has gained in importance especially among the problem domains that benefit from data analysis and self-learning. Software development effort estimation also share similar characteristics as there is typically old projects' data on hand that should help foresee new projects' efforts.

Therefore, in this study we build a software estimation model by using neural network methodology. The features for the network were chosen as a result of an extensive survey. The applicability of the methodology is demonstrated via real-life software project data provided by one of the largest banks in Turkey.

ÖZET

Yazılım endüstrisi gün geçtikçe hızla büyümekte ve tüm dünyada önem kazanmaktadır. Hemen hemen tüm sektörlerden şirketler ve kurumlar yeni uygulama ve platform geliştirmek için yazılım geliştirme projeleri yapmaktadır. Bununla beraber yazılım projelerinin eforunun doğru tahminlenmesi şirketler için önemli bir sorun haline gelmektedir.

1970'lerden bu yana yazılım projelerinin eforunun doğru tahminlenmesi için çeşitli çalışmalar yapılmaktadır. Bu çalışmalara öncü olan ilk model COCOMO olarak bilinir. COCOMO modelini Kullanım Senaryosu bazlı model UCP ve Fonksiyon bazlı model FPA takip etmiştir. 2000'lerden sonra ise, teknolojinin gelişimi ile beraber, Yapay Sinir Ağları önem kazandı ve data analizlerinde sıklıkla kullanılmaya başlandı. Yazılım projelerinin eforunun tahminlenmesi de tamamlanmış proje datalarının kullanılabilecek olması nedeniyle Yapay Sinir Ağları'nı kullanmaya uygun karakteristik özelliklere sahiptir.

Bu çalışmada yazılım projelerinin eforunun tahmin edilebileceği bir yapay sinir ağı oluşturulmuştur. Çalışma kapsamında kullanılan datalar Türkiye'nin en büyük bankalarından birinden elde edilmiştir.

1. INTRODUCTION

A project is a temporary endeavor with a beginning and an end which creates a unique product or service (Mulcahy, 2013). Effort estimation is a prediction of how long a development activity will take to finish (Leinoen, 2016).

Since software industry and digitalization gained in importance, software effort estimation is the most important problem for IT companies. McKinsey and Oxford University’s study showed that 66 percent of the large software project is over budget and 33 percent is over schedule, also 17 percent of the IT projects go so bad so the existence of the company is threatened (Chandrasekaran et al., 2014).

Both under estimation and over estimation causes the waste of time, resource, money and even prestige lost. According to Borade and Khalkar (2013) underestimating the costs is characterized by budget overruns, under developed functions and poor quality end-product. Overestimation commits too many resources to the projects and could lead to lost contracts could mean lost jobs. Mulhacy (2013) defines the term “padding”, which is related with overestimating, as a sign of poor project management which can damage reputation of a project manager.

Since 1970s many studies and methods have been publised to overcome software project effort estimation problems. All the methods aim to estimate efforts accurately. Here, estimation accuracy simply defines the comparison of the estimate to the actual effort that is known after the task has been finished (Leinonen, 2016). COCOMO is the one of the first algortihmic effort estimation models studied in late 1970s. After COCOMO, Use Case Point and Function Point methods have become the de facto standard for accurate software efforts estimation.

Since 2000s, artificial intelligence and especially neural networks are noticed by the software industry for their ability to handle complex relationships between inputs (factors/features) and outputs (estimated effort). Neural networks in this context define a supervised learning model which uses historical data to explain the relationship between inputs and outputs with the help of so called training algorithms and produce outputs for the new scenarios without subjective manual calculations and adjustments. The model potentially improves itself by each new data added to retrain the network.

In this thesis, a feed forward neural network model will be proposed to estimate software projects' efforts accurately for the software project department at one of the largest banks in Turkey. Two different learning algorithms will be applied to obtain the best output with the minimum error. The findings will be compared with the current approaches applied by the organization.

The remainder of the thesis is organized as follows: in Section 2, related work is summarized. Section 3 presents the methodologies that form the proposed model. The data gathering process and obtained results as part of model evaluation are given in Section 4. Section 5 concludes the study discussing the findings and further study possibilities.

2. LITERATURE REVIEW

Since 1950, project management and software development have become an important issue due to complex requirements of the companies and gaining acceleration in technology industry. Over than 30 years, there is a significant challenge for effective resource prediction (Santani et al., 2014).

In the beginning of the studies, researchers had focused on algorithmic models and quantative based techniques for effort estimation process. In 1979, Allan Albrecht published a parametric based model, Function Point Analysis (FPA). At about the same time, in late 1970s, The Constructive Cost Model (COCOMO) had been released by Barry W. Boehm and improved version of the model had been developed in 1997. Another parametric effort estimation model, Use Case Point (UCP) has been developed by Gustav Karner.

In 1990s, clustering, case-based reasoning and ANN became effective for predicting software effort estimation. ESTOR, a cased based approach, was developed by Vicinanza et al. and it has been claimed that ESTOR performs better than FPA and COCOMO on restricted samples.

In 1994, Witting and Finnie applied back propagation algorithm on a multilayer perceptron by using ANN. Similarly, in 1997, they used ANN to produce more accurate resource estimation for software projects. The compared ANN with cased-based reasoning and FPA models. As a result, ANN was slightly better than cased-based reasoning model and much better than FPA (Finnie & Wittig, 1997).

Also, in 1992, Karuannitthi used ANN to predict software reliability and Samson et al. (1997) used an Albus multiplayer perceptron in order to predict software effort on the

Boehm’s COCOMO dataset and compared linear regression with a neural networks approach (Santani et al., 2014).

Khoshgoftaar et al. (2000) presented a case study considering real time software to predict the testability of each module from source code static measures. They consider ANNs as promising techniques to build predictive models, because they are capable of modeling nonlinear relationships (Santani et al., 2014).

Apart from algorithmic methods, expert judgement based methods are used and preffered since they are easy to apply. In 1950, Delphi Method is conceived by Olaf Helmer and Norman Dalkey. This method attempts to capture expert opinion through a group of experts (Rush & Roy, 2001).

As some of them detailed above, there are many effort estimation methods. Although different groupings are found in the literature, three categories are usually used to classify estimation methodologies: Expert judgement, algorithmic estimation and learning based estimation.

In the following chapters these three categories will be detailed and different methods will be discussed including Neural Network Model.

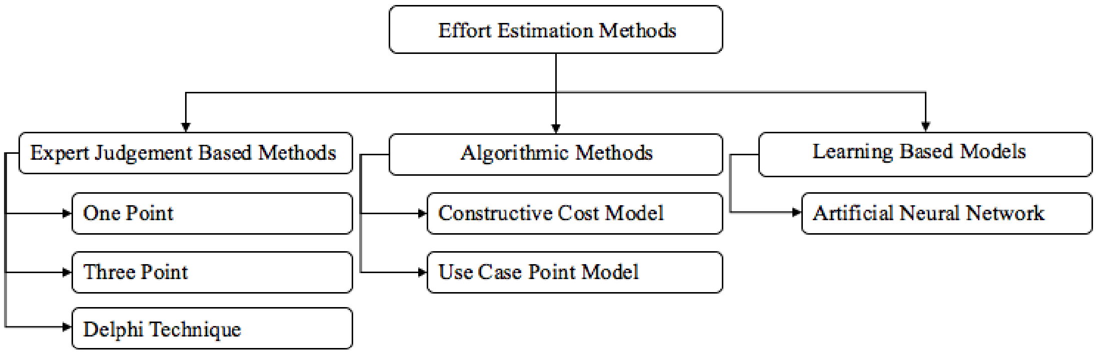

Figure 1: Effort Estimaton Methods

2. 1 Expert Judgement Based Methods

The most common used estimation approaches are expert judgement based methods in software industry (Jorgensen & Shepperd, 2007). Since, at the beginning of the projects, project team does not have a proper data to estimate the cost, expertise based methods are preferred by companies. Expert judgement based methods generate cost estimations based on experts' or project team's opinions. According to Leinonen (2016), expert judgement estimation can be used if there is no quantified data for the project.

Also lack of time is another reason to choose expert judgement based approaches. Thus, taking less time and without gathering detailed data are the main advantages of expert judgement methods. The main disadvantage is, as Boehm et al. (2000) states, even if a person has experience, this does not mean that his/her estimates are accurate. Furthermore, in real life scenarios, there are many unknowns about project team members, who are estimators, make the assumption and double it. This is usually considered as a sign of padding which indicates poor project management (Mulcahy, 2013).

In the next subchapters three mainly used expert based methods will be detailed which are One Point Estimation, Three Point Estimation and Delphi methods.

2.1.1 One Point Estimation

In One Point Estimation, the estimator submits one estimate per activity (Mulcahy, 2013). For example; the estimator says for one activity the cost will be 5 days. By summing up each activities' costs, the final number will be the project's cost.

Rita Mulhacy stated that One Point estimation can be problematic because it can force the estimator into padding, also important points like risk and uncertainties can be hidden in this method (Mulcahy, 2013).

2.1.2 Three Point Estimation

In Three Point Estimation estimators give an optimistis (O), pesimistic (P) and most likely (M) estimate for each activity (Mulcahy, 2013). Three Point estimation can be calculated in two different ways according to risk factors of projects.

In Triangular Distribution, a simple avagare formula is applied to estimates. The formula is;

$$Cost = \frac{(P+O+M)}{3} \tag{1}$$

Optimistic, pesimistic and most likely estimates have equal weight on triangular distribution.

In Beta Distribution, a weighted avarage formula is applied to estimations which give stronger consideration to the most likely estimate (Mulcahy, 2013). The formula is,

$$Cost = \frac{(P+4O+M)}{6} \tag{2}$$

According to Rita Mulhacy, when a good risk management process is followed, generally the most likely estimation occurs. So that the Beta Distrubiton is advantegous in such a case.

2.1.3 Delphi Technique

Delphi Technique is an estimation approach which allows estimators share their estimations with others and calibrate their estimations by exchanging the necessary information (Kumari & Pushkar, 2013). Delphi technique steps are as follows;

1. Coordinator provides an estimation form with spesification of project to each estimator.
2. Estimators fill out the forms by themselves.
3. Coordinator sets up a group meeting which estimators can share and discuss their estimations.
4. Coordinator prepares and distributes an iteration form which is a summary of estimations.
5. Step 2 and 3 are applied again until a consensual estimation is obtained.

Although Delphi Technique is good because of interactive aspects, there are some drawbacks. As Sadhu (2014) stated, the tendency in a group is to aggree with the majority eventhough individual feels that majority is wrong.

2.2 Algortihmic Models

Algorithmic effort estimation methods consist of mathematical models or calculations to provide effort estimation (Usharani et al., 2016). Most of the algorithmic estimation models use project size, environmental and/or technical factors to calculate projects' costs. Depending on the model, calculation procedure varies. In some models, source of line codes (SLOC) is used, whereas others use function or use case points. Also, factors and cost drivers are not common among different methods. COCOMO and Use Case Point are the most acknowledged methods in algorithmic effort estimation models.

COCOMO and Use Case Point are the most known methods in algorithms effort estimation models. In the following subchapters these models will be detailed.

2.2.1 The Constructive Cost Model

The Constructive Cost Model (COCOMO) is an algorithmic effort estimaton model developed by Barry W. Boehm in the late 1970s. The model is based on project size in SLOC and factors which are obtained from 63 projects data. In 1997 COOMO II was developed as a successor of COCOMO and 161 project data are used to obtain factors. ‘COCOMO II is a parametric cost estimation model that requires size, product and personnel attributes as inputs and outputs the estimated effort in Person-Months (PM)’ (Boehm et al, 2016).

In COCOMO II, software projects are classified into three groups as organic, semi-detached and embeded projects. Organic projects are the projects which have small teams or few domains with good experience. Semi-detached projects are made of medium teams and mixed experience team members. Embeded projects are the projects which have strict constraints, many domains and hardware, software and operational needs. Each project type has different coefficients for effort estimation.

Moreover, in COCOMO II there are four types of cost drivers with different multipliers; product attributes, platform attributes, personnel attributes and project attributes, see Table 1.

Table 1: COCOMO II Effort Multipliers

Effort Multipliers		Rating Levels and Multipliers					
		Very Low	Low	Nominal	High	Very High	Extra High
Product Factor	Software Reliability	0.82	0.92	1.00	1.10	1.26	n/a
	Data Base Size	n/a	0.90	1.00	1.14	1.28	n/a
	Product Complexity	0.73	0.87	1.00	1.17	1.34	1.74
	Developed for Reusability	n/a	0.95	1.00	1.07	1.15	1.24
	Documentation Needs	0.81	0.91	1.00	1.11	1.23	n/a
Platform Factor	Execution Time Constraint	n/a	n/a	1.00	1.11	1.29	1.63
	Main Storage Constraint	n/a	n/a	1.00	1.05	1.17	1.46
	Platform Volatility	n/a	0.87	1.00	1.15	1.30	n/a
Personnel Factor	Analyst Capability	1.42	1.19	1.00	0.85	0.71	n/a
	Programmer Capability	1.34	1.15	1.00	0.88	0.76	n/a
	Personnel Continuity	1.29	1.12	1.00	0.90	0.81	n/a
	Applications Experience	1.22	1.10	1.00	0.88	0.81	n/a
	Platform Experience	1.19	1.09	1.00	0.91	0.85	n/a
	Language and Tool Experience	1.20	1.09	1.00	0.91	0.84	n/a
Project Factors	Use of software Tools	1.17	1.09	1.00	0.90	0.78	n/a
	Multisite Development	1.22	1.09	1.00	0.93	0.86	0.80
	Required Development Schedule	1.43	1.14	1.00	1.00	1.00	n/a

These cost drivers also called as effort multipliers have scale factors from very low to very high. According to scaling, each attribute has a unique coefficient just like project types. Finally, to calculate a software project effort, the given formula is applied;

$$Effort\ (PM) = A * Size^{E} * \prod_{i=1}^{17} Effort\ Multiplier_i \tag{3}$$

The constant A is initially set when the model is calibrated to the project database reflecting a global productivity average. The COCOMO model should be calibrated to local data which then reflects the local productivity and improves the model's accuracy (Abts et al., 2000). Size is the lines of codes in thousands (KLOC). Effort Multipliers are the coefficients of attributes which is obtained from Table 1. The exponent E is a collection of five scale factors which is shown in Table 2.

Table 2: COCOMO II Scale Factors

Scale Factors	Values					
	Very Low	Low	Nominal	High	Veryh High	Extra High
Precedentedness	6.20	4.96	3.72	2.48	1.24	0.00
Development Flexibity	5.07	4.05	3.04	2.03	1.01	0.00
Architecture Risk Resolution	7.07	5.65	4.24	2.83	1.41	0.00
Team Cohesion	5.48	4.38	3.29	2.19	1.10	0.00
Process Maturity	7.80	6.24	4.68	3.12	1.56	0.00

Equation 4 defines the exponent E. In the equation, B is equal to 0.91 which is obtained from historical data of COCOMO II. As mentioned before, COCOMO parameters including B, should be calibrated to the local organization for better results. SF is the values of each scale factor.

$$E = B + 0.01 * \sum_{J=1}^{5} SF_J \tag{4}$$

Even though COCOMO is one of the oldest software project estimation models and has many versions, it is not used in real life. Since code lines are not available in early life cycle and estimation is based KLOC, COCOMO model has an important disadvantage. Also as Ren and Yun (2013) indicated that estimation results vary greatly due to different languages and algortihms.

As a result, use of COCOMO in software industry remains ‘marginal’ (Trendowicz et al., 2011).

2.2.2 Use Case Point Method

Use Case Point (UCP) method is an effort estimation model based on use cases, actors, technical and environmental factors. ‘A use case captures a contract between the stakeholders of a system about its behaviour. The use case describes the system’s behaviour under various conditions as the system responds to a request from one of the stakeholders, called primary actor’ (Cockburn, 2011).

The main input of UCP method is use cases. Generally, in medium and large size projects there are many use cases and each use case has different number of steps. In UCP method, to calculate Unadjusted Use Case Weight (UUCW), the use cases of the projects are groupped into simple, avarage and complex groups according to their step numbers. If transaction number of a use case is smaller than 4 than use case is classified as Simple, if transaction number is between 4 and 7 than use case is classified as Avarage, if transaction number is bigger than 7 than use is complex. Each complexity group has different weights as shown in Table 3.

Table 3: UCP Use Case Complexity Weights

Use Case Complexity	**Weight**
Simple	5
Avarage	10
Complex	15

After classifying use cases of the project, UUCW can be calculated as in Equation 5. In the equation weight is the use case complexity weight in the Table. Cardinality is the number of use cases assigned to complexity class C, as simple, avarage or complex.

$$UUCW = \sum_{i \in C} Weight\ (C) * Cardinality\ (C) \qquad (5)$$

After calculating UUCW, Unadjusted Actor Weight (UAW) is calculated. In a software project, there can be many different type of actors like client, customer, database, GUI etc. Similar to UUCW calculation, actors are groupped into three categories; simple, avarage and complex. Each group has different weights as shown in Table 4. Simple actor is a system actor which communicates with other system by API. Avarage actor is a system actor communicates via a protocol like HTTP or a person who interacts with a system via a terminal console. Complex actor is a person actor uses User Interface to interact with system.

Table 4: UCP Actor Complexity Weights

Actor Complexity	**Weight**
Simple	1
Avarage	2
Complex	3

After classifying actors, UAW can be calculated as in Equation 6. In the equation weight is the actor complexity weight in the Table. Cardinality is the number of actors assigned to complexity class C, as simple, avarage or complex.

$$UAW = \sum_{i \in C} Weight\ (C) * Cardinality\ (C) \qquad (6)$$

As the last two steps of UCP calculation, technical (TCF) and environmental (EF) complexity factors are calculated. There are 13 technical and 8 environmental factors. Each factor has a different weight as shown in the Table 5.

Table 5: UCP Technical and Environmental Factors

Factor Type	No	Factor Name	Weight
Technical	**TF1**	Distributed system	2
	TF2	Response time/performance objectives	1
	TF3	End-user efficiency	1
	TF4	Internal processing complexity	1
	TF5	Code reusability	1
	TF6	Easy to install	0.5
	TF7	Easy to use	0.5
	TF8	Portability to other platforms	2
	TF9	Easy to change	1
	TF10	Concurrent/parallel processing	1
	TF11	Security features	1
	TF12	Access/Dependence for third parties	1
	TF13	End user training	1
Environmental	**EF1**	Familiarity with development process used	1.5
	EF2	Application experience	0.5
	EF3	Object-oriented experience of team	1
	EF4	Lead analyst capability	0.5
	EF5	Motivation of the team	1.5
	EF6	Stability of requirements	2
	EF7	Part-time staff	-1
	EF8	Difficult programming language	-1

To calculate TCF and EF Equation 7 and 8 are used. *TFWeight* and *EFWeight* refer to the factor weights in the table above. *Value* is the predicted degree of influence for each factor which can be between 0 and 5. If value is 0 that means this factor has no effect or relationship with the project. On the contrary, if the value is 5 that mean this factor has a strong effect or relation ship with the project.

$$TCF = 0.6 + (0.01 * \sum_{i=1}^{13} TFWeight_i * Value_i) \quad (7)$$

$$EF = 1.4 + (-0.03 * \sum_{i=1}^{8} EFWeight_i * Value_i) \quad (8)$$

Finally, UCP is calculated as follows;

$$UCP = (UUCW + UAW) * TCF * EF \quad (9)$$

To calculate project's effort in man hours, UCP is multiplied by 20 hours as Karner proposed (Banerjee, 2001).

$$Effort = UCP * 20 \quad (10)$$

As Kirsten Ribu stated, per UCP hours can range from 15 hours to 30 hours as field experience shows (Ribu, 2001). Eventhough adjusting hours per use case point according to companys' history can be an advantage, UCP method has important drawbacks. The effort estimation can not be arrived before all use cases are written (Cohn, 2005). And writing all use cases means analysis phase is completed for a software project, which is quite late for effort estimation. Additionally, counting use case steps can be a problem, especially for large size projects.

As a conclusion, UCP is an easy to calculate method as a mathematical formula but it has important disadvantages to apply in real life.

2.3 Learning Based Models

Learning based effort estimation models use current knowledge and historical data of the projects. As Gabrani and Saini stated, learning based methods are trying to imitate natural evolution and they are refining until finding and optimal solution, so evolutionary learning based methods became popular in last years (Gabrani & Saini, 2016).

Machine learning can be defined as a set of mechanisms which enable computers learn from experiences (Negnevitsky, 2002). Artificial neural network (ANN) is the most widely applied model under the umbrella terms Artificial Intelligence and Machine Learning. In the next subchapter artificial neural network model will be detailed.

2.3.1 Artificial Neural Networks

ANN is a "reasoning based human brain model" which uses interconnected neurons to learn and execute transactions or functions just like human brain does with 10 billion neurons and 60 trillion connections (Negnevitsky, 2002). ANNs are preferred as they enable to model even complex non-linear relationships and are pretty much capable of approximating any measurable function without an explicit model of the system (Finnie & Wittig, 1997).

A typical ANN as is made up from nodes in three layers; input layer, hidden layer(s) and output layer as shown in Figure 2.

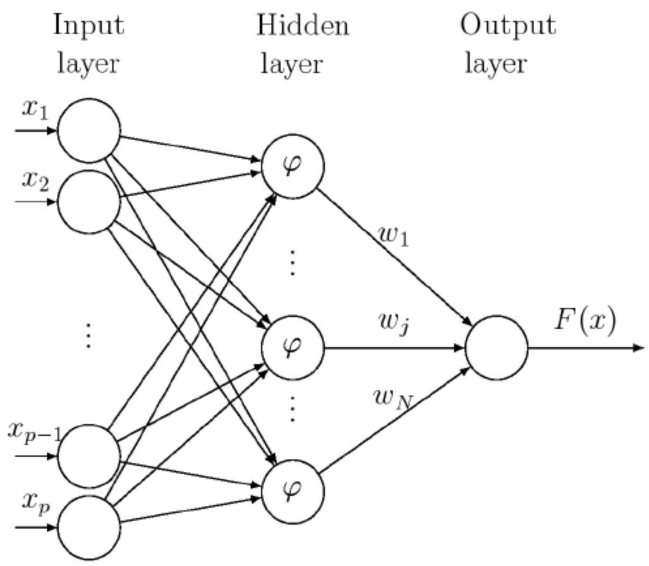

Figure 2: Multilayer Perceptron ANN with one hidden layer

Each input layer node is connected to the next hidden layer nodes and each hidden layer node is connected to the next one ending with the output layer node. Nodes in the input layer, hidden layers and output layer and hidden layer numbers may change depending on the problem. Each connection between nodes represents a weight. Input layer represents the input data for learning algorithm.

Hidden layer and output layer use the data from previous layer and combine them with the corresponding weights to trigger a so called activation function. The output layer combines all the outputs generated by the activation functions and outputs a value once again using an activation function. There are various activation functions used in the literature, linear, sigmoid, Gaussian, etc. As Michael Negnevitsky stated, "weights are the basic means of long-term memory in ANNs. They express the strength, or in other words importance, of each neuron input. A neural network 'learns' through repeated adjustments of these weights." (Negnevitsky, 2002).

If there ise a linear relationship between input and output layer, then it means there is no need for a hidden layer. This kind of ANN is called as a *perceptron*. In contrast, if there is a non linear relationship between input and output layer, one or more hidden layers are needed to solve the problem. In these cases, ANN is called as *Multi Layer Perceptron*.

There are two main types of ANN architecture called as Feed Forward and Feed Back networks. Feed forward network progresses only one way from input neurons to output. Feed-forward networks tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition.

Feed Back networks have feedback connections from the output layer to the input layer or from the hidden layer to the input layer. In other words, a feedback architecture distinguishes itself from a feedforward architecture, in that it has at least one feedback link (Chiang & Li-Chiu Chang, 2004).

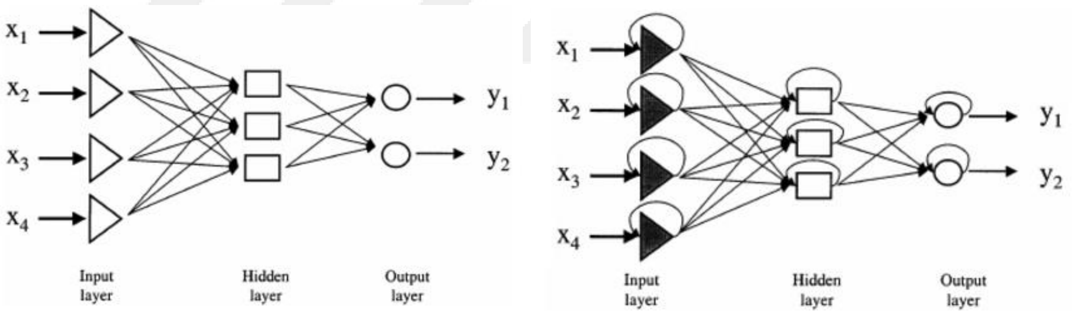

Figure 3: Feed Forward and Feed Back Networks
(Agatonovic-Kustrin & Beresford, 2000)

There are different types of learning algorithms for ANNs. One of the most popular types is multi-layer perceptron with the combination of feed-forward and back-propagation algorithms. In feed forward backpropagation algorithm there are two phases to reach the results. The first phase is called as Forward Phase. In this phase, input signals go through the network from input nodes to outputs until an error signal is computed. Error signal is difference between desired and the actual output. And the second phase ise backward phase. In this phase, error signal moves in the backward direction for the adjustments until minimizing error and obtaining an acceptable value.

The aim of the feed forward back propagation is to minimize cost function to find the best result with minimum error margin. As cost function, generally a quadratic function is used to figure out how to make small changes in weights so as to get an improvement the quadratic cost (Nielsen, 2017). To solve quadratic cost function, gradient descent technique is used since the problem that ANN is trying to solve is a minimization problem.

The ideal solution is to find a global minimum for the problem but in real life, ANN may have to solve problems with millions of inputs and outputs. In such cases, finding a global minimum could not be possible. So with gradient descent, ANN chooses a starting point randomly, with random weights and tries to find a local minimum. While trying to find a local minimum, a learning rate is used. Learning rate should be small enough to address the inputs to correct outputs and to minimize cost function. But also it should not be very small. Because with a very small learning rates, gradient descent works very slowly and also correct input-output match up problems may occur.

The process of feed forward back propagation is summarized iteratively below;

- Cost function is defined as; $E = \frac{1}{2}\sum_i e_i^2\ (k)$
- In this function; $e_i(k)$ is the error for the i^{th} neuron for the k^{th} iteration.
- $e_i(k)$ is defined as; $e_i = d_i - y_i(k)$. Here, d_i is the desired output for i^{th} neuron and $y_i(k)$ is the actual output.
- So cost function can be also written like; $E = \frac{1}{2}\sum_i(d_i - y_i(k))^2$
- Cost function depends on the weights, since ANN learns by adjusting weights of the neurons. At the beginning, weights are assigned randomly. Changes of the each neuron is found by Gradient Descent Algorithm, which can be represented as $\Delta w_{ij} = -\mu \frac{\partial E(\underset{w}{\rightarrow})}{\partial w_{ij}}$. Here μ is the learning coefficient.
- In forward phase, output values are obtained according to the weights which are applied during the forward process. In backward phase, weights are reassigned according to the errors on outputs.

- Each change on the weights are calculated as; $\Delta w_{ij} = \mu * \delta_j * y_i$.
- δ_j is defined as $e_j(k) * f_j'$ for output layer neurons and $f_j' \sum_m \delta_m w_{mj}$ for hidden layers.
- f_j is the activation function of j^{th} neuron.

In this context, ANNs are used to calculate estimated software project efforts. Since it is a learning based model, with enough previous project data and feature set, the model can predict accurately project efforts.

Compared to other effort estimation models, ANNs have an important advantage, as they are trained using a company's own data, they can estimate project cost more accurately for a specific company then a generic model with a standard set rules. Moreover, ANNs do not need an implicit or complete programming as required by regression based methods. In this work, selected historical projects' data will be used to build an ANN model.

3. MATERIALS AND METHODS

The aim of this study is to build an ANN and use the network to estimate software projects' efforts. As detailed in previous sections, an ANN depend on input variables to make the estimation. In order to build an ANN, five input variables are identified through preliminary data analysis using expert interviews, focus group and surveys.

This initial step is required as ANNs actually mimic the decision making process of experts by replacing the expert opinion with a black-box approach. Therefore, software project managers of one of the largest bank in Turkey are consulted in order to define the basic information that is needed for software effort estimation. The relationship between these inputs and the corresponding effort estimation is handled by the trained ANN. For training purposes, 77 IT projects' data is obtained from the bank's Project Management department.

In the next subchapters input variable selection, data collection, generating ANN and obtaining valuable estimation topics will be detailed.

3.1 Input Variable Selection

'The choice of input variables is a fundamental, and yet crucial consideration in identifying the optimal functional form of statistical models.' (May et al., 2011). Similarly input variable selection has a crucial importance to create ANN on a sound basis.

As detailed in Literature Review, there are many different methods for effort estimation. Each method has different cost drivers and input parameters. In this study, existing effort estimation models' inputs and the bank's IT experts' opinions are considered to obtain the most effective input variables on effort estimation.

3.1.1 Input Variable Alternatives

Input variables (parameter) selection is one of the most important tasks to estimate software projects' efforts accurately. In literature, for algorithmic models, different factor groups and variables are used. Generally, they are grouped into two categories as 'Technical Factors' and 'Environmental Factors'. In this study, Use Case Point, Function Point Analysis and Jensen Model's factors are considered to be used as input to our proposed ANN model.

In UCP method, there are two types of factors categorized as technical and environmental. Technical factors define 13 parameters and environmental factors consists of 8 parameters.

Table 6: UCP Technical Factor Descriptions

No	Technical Factor Name	Description
TF1	Distributed system	Refers to a single and integrated coherent network requirement to share different resources and capabilities to provide users.
TF2	Response time/performance objectives	Refers to response time requirements for the desired system. Some system transactions are needed to have very short response time as money exchange transactions.
TF3	End-user efficiency	Refers to system needs for end users. End user of the desired system can be an external client or internal client. End-user efficiency weight changes depend on client type and requirements.
TF4	Internal processing complexity	Refers to system's dependency to each other and multiple system integration needs.
TF5	Code reusability	Refers to reusable, parametric code requirements.
TF6	Easy to install	Refers to accessibility and installability of desired system or application.
TF7	Easy to use	Refers to usability requirements including system-human interaction.

TF8	Portability to other platforms	Refers to usability of the same software in different environments.
TF9	Easy to change	Refers to easy code changing requirements.
TF10	Concurrent/parallel processing	Refers to simultaneous access requirements.
TF11	Security features	Refers to special security requirements. As an example, developing a login system for an application or system may need very complex security requirements.
TF12	Access/Dependence for third parties	Refers to third party access needs for desired systems. For example the application that will be developed may need to access government database to take client's identity information.
TF13	End user training	Refers to end user training needs. As an example, for call center system/transaction changes, call center staff may need to be trained to use new system.

Table 7: UCP Environmental Factor Descriptions

No	Environmental Factor Name	Description
EF1	Familiarity with development process used	Refers to familiarity with life cycle model used for the project team. Agile, Waterfall or any other development life cycles may be used for the development projects.
EF2	Application experience	Refers to application experience of project team.
EF3	Object-oriented experience of team	Refers to object oriented experience of project team, especially for the developers.
EF4	Lead analyst capability	Refers to lead analyst capability to identify, understand requirements accurately.
EF5	Motivation of the team	Refers to project team's motivation.
EF6	Stability of requirements	Refers to stability level of requirements. Requirements of a project would not be clear at the beginning of the project.
EF7	Part-time staff	Refers to part time staff of the project team.
EF8	Difficult programming language	Refers to programming language's use of difficulty.

Similar to UCP method, to build an effort estimation model, 14 'General System Characteristics' (GSCs) are used in Function Point Analysis (FPA) (Lokan, 2000).

General system characteristics are also known as technical factors. GSCs have some common factors with UCP technical factors.

Table 8: FPA General System Characteristics Descriptions

No	General System Characteristics Name	Description
GSC1	Data Communications	Refers to data transfer needs by using communication technologies.
GSC2	Distrubuted Data Processing	Refers to a single and integrated coherent network requirement to share different resources and capabilities to provide users.
GSC3	Performance	Refers to system performance needs including response times.
GSC4	Heavily Used Configuration	Refers to degree of computer resource restrictions which effects the development of the application
GSC5	Transaction Rate	Refers to the rate of business transactions needs which influences the development of the application
GSC6	Online Data Entry	Refers to online data entry requirements through interactive transactions.
GSC7	End User Efficiency	Refers to human-application interaction and usability needs.
GSC8	Online Update	Refers to internal logical files' online update requirements.
GSC9	Complex Processing	Refers to complex processing logic requirements which effects development of the application.
GSC10	Reusability	Refers to reusable, parametric code requirements.
GSC11	Installation Ease	Refers to accessibility and installability of desired system or application.
GSC12	Operational Ease	Refers to easy operational usage needs of the system in such processes like recovery, back up or start up.
GSC13	Multiple Sites	Refers to multiple different hardware and software environmental needs for the application.
GSC14	Facilitate Change	Refers to easy modification of processing logic or data structure requirements.

Jensen model is a software development schedule/effort estimation model which incorporates the effects of any of the environmental factors impacting the software development cost and schedule (Baik, 2000). This model is an empirical model and related to the effective size of the system and the technology to the implementation of the system. Since Jensen model's environmental factors are considere as candidate factors since they seem suitable for the bank's organization. Jensen model defines 13 environmental factors.

Table 9: Jensen Model Environmental Factor Descriptions

No	Environmental Factors for Jensen Model	Description
JEF1	Special Display Requirements	Refers to human-application interaction and usability needs including front end designs.
JEF2	Operational Requirement Detail and Stability	Refers to operational usage needs of the system in such processes like recovery, back up or start up.
JEF3	Real Time Operation	Refers to time-constrained operation requirements.
JEF4	CPU Time Constraint	Refers to system performance needs including response times.
JEF5	Memory Constraint	Refers to memory, storage requirements.
JEF6	Virtual Machine Experience	Refers to developers virtual machine experience.
JEF7	Concurrent ADP Development	Refers to concurrent Automatic Data Processing (ADP) development requirements like complex logical processes.
JEF8	Developer Using Remote Computer	Refers to developer's remote computer experience.
JEF9	Development at Operational Site	Refers to development needs at the operational site which means according to operation systems.
JEF10	Development Computer Different than Target Computer	Refers to rehosting needs which means protecting business logic and data trapped in proprietary hardware and software, while opening paths to future modernization by moving to a more extensible architecture.
JEF11	Development at Multiple Sites	Refers to multiple different hardware and software environmental needs for the application's development process.
JEF12	Programming Language Experience	Refers to developer's programming language experience which will be used in the project.
JEF13	System Reliability	Refers to system's ability requirements to perform as it is intented to design.

In our case, besides UCP, FPA and Jensen Model parameters, 5 additional parameters are considered to have an effect on project effort estimation as they are already used by the experts of the selected bank's IT department. These parameters are shown in Table 10.

Table 10: The Bank's Expert Opinion Input Descriptions

No	Expert Opinion Inputs	Description
EO1	Domain Number	Refers to number of the IT domains which will involve to the project as a stakeholder.
EO2	Software Development Project Methodology	Refers to methodogy type which project will get on. Methodology can be Agile or Waterfall or any other.
EO3	Team Characteristics	Refers to project team characteristics as experience, taking part in the same project before etc.
EO4	Key Turn Project	Refers to a project which will done by outsource project team from beginning to the end.
EO5	Business Unit Efficiency	Refers to businnes units efficiency on the project.

In total, 53 factors from UCP, FPA, Jensen Model and expert opinions are considered as candidate inputs to the ANN model. As this list was too comprehensive and it would require a lot of project data to train the ANN, we consulted 6 expert project managers to evaluate the importance of these factors. As a result, 22 factors are identified as having a considerable effect on software project effort.

Table 11: Choosen Factors by Focus Group

No	Factor Name
F1	Well-defined and stable requirements
F2	Access/Dependence on 3rd party company's code
F3	Multiple domain integration
F4	Reusable code
F5	Complex security requirements
F6	Developer's application experience

F7	Easy to change (parametric design)
F8	Team's familiarity with the project
F9	System reliability
F10	Performance requirements
F11	Real time operation needs
F12	Business unit/client attendance
F13	Team characteristics
F14	Lead analyst's capability
F15	Outsource analysts/developers
F16	Operational ease
F17	Object-Oriented experience
F18	Team motivation
F19	Installation (deployment) ease
F20	Methodology (Waterfall, Agile etc.)
F21	Part time staff
F22	Programming language

3.1.2 Conducting Survey and Analayzing Survey Results

After preselection, a survey is conducted on 19 IT experts to analyze the effect of the parameters according to expert opinions and to select the most relevant factors as input to ANN model. 22 preselected factors are asked to scale from “1-Irrelevant” to “5-Relevant” according to the effect on software development effort estimation. Scaling range is listed in the Table 12.

Table 12: Factor Scaling Ranges

No	**Scaling Range Description**
1	Irrelevant
2	Slightly Relevant
3	Relevant
4	Fairly Relevant
5	Highly Relevant

An ‘effect level’ is calculated based on the ratings of factors by IT experts. Weights are assigned to each scale range and by multiplying scale weight and experts’ choices, effect level is obtained.

$$Effect\ Level = \sum_{i=1}^{5} w_i * c_i \qquad (11)$$

Where *i* is the number of scale range from irrelevant to highly relevant, *w* is the weight of scale range and *c* is the number of choice for the factor. According to the effect level calculation, top 5 factors with the highest effect level are selected as the input factors to ANN model which are; “well defined and stable requirements”, “dependence in 3rd party company’s code”, “multiple domain integration”, “reusable code” and “complex security requirements”. Survey’s effect level results are shown in table 13 where I is Irrelevant, SR is Slightly Relevant, R is Relevant, FR is Fairly Relevant and HR is Highly Relevant. Also w is the weight of each scale.

Table 13: Survey Results for Choosen Factors

Factor	**I (w=1)**	**SR (w=2)**	**R (w=3)**	**FR (w=4)**	**HR (w=5)**	**Effect Level**
Well-defined and stable requirements	0	0	0	6	13	89
Dependence on 3rd party company's code	0	1	3	5	10	81
Multiple domain integration	0	0	3	8	8	81
Reusable code	0	0	2	12	5	79
Complex security requirements	0	2	3	8	6	75
Developer's application experience	0	2	6	8	4	74
Easy to change	0	1	3	13	2	73
Team's familiarity with the project	0	2	4	8	5	73
System reliability	1	0	5	8	5	73
Performance requirements	0	1	7	6	5	72
Real time operation needs	0	2	6	6	5	71
Business unit/client attendance	1	1	5	9	3	69
Team characteristics	1	2	5	7	4	68

Lead analyst's capability	0	2	8	7	2	66
Outsource analysts/developers	0	4	7	6	2	63
Operational ease	0	5	7	5	2	61
Object-Oriented experience	1	3	8	6	1	60
Team motivation	1	3	8	6	1	60
Installation (deployment) ease	0	6	6	5	2	60
Methodology (Waterfall, Agile etc.)	1	7	4	4	3	58
Part time staff	0	6	8	4	1	57
Programming language	0	5	9	5	0	57

3.2 Data Collection

Similar to human brain ANN learns and when it is learning it needs the historical data to create the complex non linear relationships between input variables. In this study, ANN has been created for software development projects' effort estimations. To create ANN, 77 completed software project data is handled from one of the Turkey's biggest bank's Project and Program Management Office.

During the project period, each project team member is required to fill timesheets to show how many man day a team member has spended. At the end of the projects, all projects' accumulated actual effort information calculated from each resource's time sheets. For the proposed ANN, actual effort is set as the target value. For these 77 projects actual efforts are shown in Table 14.

Table 14: 77 Projects' Actual Efforts

Project No	**Actual Effort (m/d)**	**Project No**	**Actual Effort (m/d)**	**Project No**	**Actual Effort (m/d)**
Project 1	359	Project 27	673	Project 53	1690
Project 2	344	Project 28	579	Project 54	655
Project 3	292	Project 29	280	Project 55	366
Project 4	205	Project 30	270	Project 56	1429
Project 5	202	Project 31	183	Project 57	2996
Project 6	171	Project 32	75	Project 58	1651

Project 7	170	Project 33	68	Project 59	1899
Project 8	148	Project 34	68	Project 60	1925
Project 9	84	Project 35	62	Project 61	2477
Project 10	45	Project 36	251	Project 62	211
Project 11	429	Project 37	238	Project 63	692
Project 12	363	Project 38	745	Project 64	119
Project 13	348	Project 39	541	Project 65	189
Project 14	208	Project 40	287	Project 66	100
Project 15	80	Project 41	266	Project 67	223
Project 16	503	Project 42	407	Project 68	449
Project 17	488	Project 43	1594	Project 69	243
Project 18	238	Project 44	1085	Project 70	823
Project 19	185	Project 45	636	Project 71	158
Project 20	114	Project 46	453	Project 72	195
Project 21	67	Project 47	275	Project 73	226
Project 22	65	Project 48	2463	Project 74	138
Project 23	429	Project 49	598	Project 75	155
Project 24	219	Project 50	283	Project 76	240
Project 25	115	Project 51	630	Project 77	110
Project 26	854	Project 52	1902		

Well defined and stable requirements", "dependence in 3rd party company's code", "multiple domain integration", "reusable code and complex security requirements" are the chosen input factors for the ANN model as mentioned before. Each factor is scaled to obtain input parameter values for the projects as shown in Table 15.

Table 15: Factor Scale Definitions and Ranges

Factor Name	Scale Definition	Range of Values				
Well-defined and stable requirements	From 1 to 5. 1 for weak defining/no stability, 5 for well-defined and stable requirements	1	2	3	4	5
Dependence on 3rd party company's code	1 if there is a dependence on 3rd party code, 0 if not.	1	0			
Multiple domain integration	Domain number. From 1 to n.	1	.	.	.	n
Reusable code	1 if projects needs to be developed with reusable code, 0 if not.	1	0			
Complex security requirements	From 1 to 5. 1 if the project doesnt need any security developments, 5 for highly complex security needs.	1	2	3	4	5

77 projects' project manager is asked to give a grade for each project's factors. For having a consistent grading, sample case projects' gradings are shown to be based on. As a result, each historical project data has been graded for the 5 selected input variables and historical project data with actual effort is obtained. shown in Table 16.

Table 16: 77 Projects' Input Data Set

Project No	Well-defined and stable requirements	Dependence on 3rd party company's code	Multiple domain integration	Reusable code	Complex security requirements	Actual Effort (m/d)
Project 1	4	0	2	1	4	359
Project 2	3	1	4	1	3	344
Project 3	3	0	2	0	3	292
Project 4	3	1	2	1	3	205
Project 5	4	0	2	0	3	202
Project 6	3	0	2	1	3	171
Project 7	4	1	2	1	3	170
Project 8	5	1	2	0	3	148
Project 9	3	0	2	0	2	84
Project 10	4	1	4	1	2	45
Project 11	2	1	4	1	3	429
Project 12	5	0	10	1	2	363

Project 13	3	0	6	1	3	348
Project 14	3	1	2	1	3	208
Project 15	2	1	6	1	1	80
Project 16	2	1	4	0	3	503
Project 17	1	1	4	0	2	488
Project 18	5	1	4	1	3	238
Project 19	4	0	4	0	2	185
Project 20	5	1	4	0	2	114
Project 21	2	0	6	1	1	67
Project 22	3	0	4	1	2	65
Project 23	5	1	6	0	3	429
Project 24	2	1	6	0	1	219
Project 25	4	0	4	0	2	115
Project 26	2	0	6	1	5	854
Project 27	2	0	8	1	3	673
Project 28	2	0	4	1	4	579
Project 29	4	0	6	0	2	280
Project 30	3	1	6	1	2	270
Project 31	5	0	6	0	2	183
Project 32	4	0	6	0	1	75
Project 33	5	0	4	0	2	68
Project 34	4	1	4	1	2	68
Project 35	5	0	4	0	2	62
Project 36	2	1	6	0	1	251
Project 37	3	1	7	0	1	238
Project 38	4	0	8	1	4	745
Project 39	1	1	8	1	2	541
Project 40	4	1	8	1	2	287
Project 41	2	0	6	1	2	266
Project 42	5	1	7	1	3	407
Project 43	1	1	16	0	4	1594
Project 44	4	0	10	1	5	1085
Project 45	4	1	10	0	2	636
Project 46	5	0	12	0	2	453
Project 47	5	0	13	1	1	275
Project 48	1	1	26	1	5	2463
Project 49	4	0	14	1	2	598
Project 50	2	0	10	1	1	283
Project 51	4	1	16	1	1	630
Project 52	3	0	22	1	5	1902

Project 53	4	0	18	0	5	1690
Project 54	4	0	18	1	1	655
Project 55	5	0	12	0	1	366
Project 56	1	1	22	1	2	1429
Project 57	1	1	34	1	5	2996
Project 58	3	0	30	1	2	1651
Project 59	3	1	24	1	4	1899
Project 60	1	1	18	1	5	1925
Project 61	4	1	32	1	5	2477
Project 62	4	1	10	1	1	211
Project 63	5	0	20	1	1	692
Project 64	2	1	6	1	1	119
Project 65	3	0	6	1	2	189
Project 66	5	1	6	1	2	100
Project 67	5	0	12	1	1	223
Project 68	5	0	16	1	1	449
Project 69	4	0	8	1	2	243
Project 70	5	0	20	0	1	823
Project 71	5	0	4	1	3	158
Project 72	1	0	4	1	2	195
Project 73	5	0	12	1	1	226
Project 74	4	0	8	0	1	138
Project 75	5	0	1	1	4	155
Project 76	3	1	2	1	3	240
Project 77	5	1	4	0	2	110

3.3 Creating Artificial Neural Network

Effort estimation using ANNs defines parameters in order to find the optimal solution based on the input parameters as part of the training process. Complex relationships can be reproduced by ANNs based using appropriate weight calculation techniques (Aljahdali et al., 2015). The learning process within artificial neural networks is a result of changes in the network's weights. The objective is to find a set of weights, which should map any input to a correct output (Jacobson, 2014).

To create a proper ANN; learning type, learning algortihm, hidden layer and neuron number selection tasks are very important. In the next subchapters these tasks will be detailed and explained which one is choosen for which reason.

3.3.1 Learning Type Selection

Artificial Neural Network aims to find the optimal solution (output) according to input variables and values. To find the optimal solution weight assignment to each neuron is very important. Before finding the best weight assignment way, the problem which ANN will deal with must be determined carefully. In some kind of problems, both input and ideal or actual output values can be obtained to find the best solution and train ANN. On the contrary, in some different cases only the input variable values can be obtained and an ideal solution is trying to be predicted by obtaining the relationships of the data sets. According to problem and obtained data type, there are three main learning types; Supervised Learning, Unsupervised Learning and Reinforcement Learning.

Supervised learning is a form of regression that relies on example pairs of data: inputs and outputs of the training set. One or more target values are predicted from input variable(s) (Agatonovic & Beresford, 2000). When both input and output variables are provided in the neural network, and error based calculation is possible based on target output and actual output (Jacobson, 2014). In supervised learning, the input layer neurons receive data from a data file and the output neurons provide ANN's response to the input data. Hidden neurons communicate only with other neurons. Supervised network with the back propagation learning algorithm is a frequently used ANN which is excellent at prediction and classification tasks (Agatonovic & Beresford, 2000).

In Unsupervised Learning, there is only a given set input variables and no desirable output variable. Unsupervised learning is able to find the structure or relationships between complex input data sets. To group input variables, the system itself must decide the features which will be used. This is often referred to as self-organization or adaption. (Agatonovic & Beresford, 2000). The widely known examples for unsupervised learning are clustering, anomaly detection and blind signal seperation.

The third popular learning type is Reinforcement Learning which is very similar to Supervised Learning. ‘Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards.’ (Murphy, 1998). In this learning type, instead of actual outputs a reward is given to neural network.

Each learning type is suitable for some specific problems. Supervised learning is generally used for curve fitting problems. Unsupervised learning is suitable for clustering cases. Reinforcement learning can be used in different problems like blind signal separation. For our study, supervised learning is suitable as the learning type since 77 completed project data with input and actual output variables are provided. Also our aim is to predict output efforts on completion for the software projects.

3.3.2 Learning Algortihm Selection

Learning algorithms are used to obtain weights of each neuron and relationships between neurons and layers while traning the ANN. The most widely known learning algorithm for supervised learning is multi-layer perceptron with feed-forward network and back-propagation learning as mentioned in section 2.3.1.

When feed forward network and back propagation is combined, ANN can progress in both directions from input to output and/or from output to input. Also, feed forward back propagation can have relationships between the neurons in the same layer. So that neurons in the same layer can have linear or non linear relationships. The goal of this algorithm is to decrease global error (Chiang & Chang, 2004). Since Feed Forward Back Propagation provides complex, non linear relationships between neurons to reach the goal and to find the optimal solution, in our thesis study, ANN will be trained by Feed Forward Back Propagation learning algorithm.

There are many different types of Back Propagation functions which can be used for supervised learnings. Bayesian Regularization Back Propagation and Levenberg-Marquardt Back Propagation are the mostly adapted functions for back propagation algorithms.

3.3.2.1 Levenberg-Marquardt

The Levenberg–Marquardt algorithm blends the steepest descent method and the Gauss–Newton algorithm. Fortunately, it inherits the speed advantage of the Gauss–Newton algorithm and the stability of the steepest descent method (Yu & Wilamowski, 2010). The update rule of Levenberg-Marquardt (LM) algortihm is as in the Equation 12 (Yu & Wilamowski, 2010).

$$\Delta w = (J^T J + \mu \mathrm{I})^{-1} J^T e \tag{12}$$

In the equation, $\boldsymbol{w}$ is the weight factor, **I** is the identity matrix. $\boldsymbol{\mu}$ is the combination coefficient which is always positive, generally starts as a small value like 0.1. $\boldsymbol{J}$ is the Jacobian Matrix (P X M) X M. $J^T J$ is also known as Hessian Matrix. $\boldsymbol{e}$ is the error vector (P X M) X 1. $\boldsymbol{J}$ and $\boldsymbol{e}$ are defined as;

$$J = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \cdots & \frac{\partial e_{11}}{\partial w_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial e_{PM}}{\partial w_1} & \cdots & \frac{\partial e_{PM}}{\partial w_N} \end{bmatrix} \qquad e = \begin{bmatrix} e_{11} \\ e_{12} \\ \cdots \\ e_{1M} \\ \cdots \\ e_{P1} \\ \cdots \\ e_{PM} \end{bmatrix} \tag{13}$$

where P is the number of training patterns, M is the number of outputs, and N is the number of weights. Elements in error vector e are calculated by;

$$e_{PM} = d_{PM} - o_{PM} \tag{14}$$

where d_{PM} is the desired output and o_{PM} is the actual output, respectively, at network output M when training pattern P.

The training process using Levenberg–Marquardt algorithm is designed as follows (Yu & Wilamowski, 2010);

1. The total error (SSE) is evaluated with the initial weights which are randomly generated.
2. Updates in the LM algorithm are done to adjust weights. As the combination of the steepest descent algorithm and the Gauss–Newton algorithm, the LM algorithm switches between the two algorithms during the training process. When the combination coefficient μ is very small (nearly zero), LM algorithm approaches to Gauss–Newton algorithm where $H = J^T J$. When combination coefficient μ is very large, LM algorithm approaches to the steepest descent method where $H = J^T J + \mu I$.
3. The total error is evaluated with the new weights.
4. If the current total error is increased as a result of the update, then the step is retracted (such as reset the weight vector to the precious value) and combination coefficient $\boldsymbol{\mu}$ is increased by a factor of 10 or by some other factors. Then step 2 is applied and another update is tried again.
5. If the current total error is decreased as a result of the update, then the step is accepted (such as keep the new weight vector as the current one) and the combination coefficient μ is decresed by a factor of 10 or by the same factor as step 4.
6. Step 2 is applied with the new weights until the current total error is smaller than the required value.

3.3.2.2 Bayesian Regularization

Bayesian regularization is implemented in the Levenberg - Marquardt algorithm to minimize a liner combination of squared errors and weights. This implementation is one of the approaches to stop over-fitting a problem. It also reduces the need to test a different number of hidden neurons for a problem (Pandya et al., 2017)

Like Levenberg- Marquardt algorithm, Bayesian Regularization, training function obtains all the weights of neurons by using Levenberg-Marquardt optimization. In addition to Levenberg-Marquardt optimization, squared errors and weights are minimized by Bayesian Regularization function and then function determines the correct combination to provide an ANN which generalizes well. The process is called Bayesian regularization. Bayesian Regularization obtains a well-defined statistical problem from a nonlinear regression in the manner of ridge regression (Burden & Winkler, 2008). The benefit of Bayesian Regularization is that all available data can be used as training data, which means no test or validation set is needed (Hirschen & Schafer, 2005). Also it can be a solution for the ‘over fitting’ problems. Bayesian cost function is as follows;

$$C(w) = \beta * E_d + \alpha * E_w \tag{15}$$

In the equation 15, *C(w)* is the cost function. α and β are the hyperparameters of Bayesian Regularization that shows which direction must be seek by learning process. Directions can be minimum error or minimum weight. E_d is the sum of squared erros and E_w is the sum of squared weights. A third variable, *gamma* γ, indicates the number of effective weights being used by the network, thus giving an indication on how complex the network should be (Souza, 2009). Bayesian Regularization works as follows;

1. Jacobian *J* is computed.
2. Error gradient $g = J^T e$ is computed.
3. Hessian matrix is computed.
4. Cost function *C(w)* is calculated.
5. $(J^t J + \lambda \mathrm{I})\sigma$ equation is solved to find σ.
6. Using σ network weights are updated.
7. Cost function *C(w)* is recalculated using the updated weights.
   7.1 If the cost has not decreased new weights are discarded, λ is increased. After that algorithm begins again from step number 5.
   7.2 If the sum squared errors has decreased, λ is decreased.
8. Bayesian hyperparameters are updated by using MacKay'S or Poland's formulae.
   8.1 $\gamma = w - (\alpha * tr(H^{-1}))$
   8.2 $\beta = \frac{N-\gamma}{2*E_d}$
   8.3 $\alpha = w/(2 * E_w + tr(H^{-1}))$ *[modified Poland's update]*, **or** $\alpha = \gamma/(2 * E_w)$ *[original MacKay's update]*, **where**:
      8.3.1 *w* is the number of network parameters (number of weights and biases)
      8.3.2 *N* is the number of entries in the training set
      8.3.3 $tr(H^{-1})$ is the trace of the inverse Hessian matrix

Another simple flow for Bayesian Regularization Back Propagation is (Yue et al., 2011);

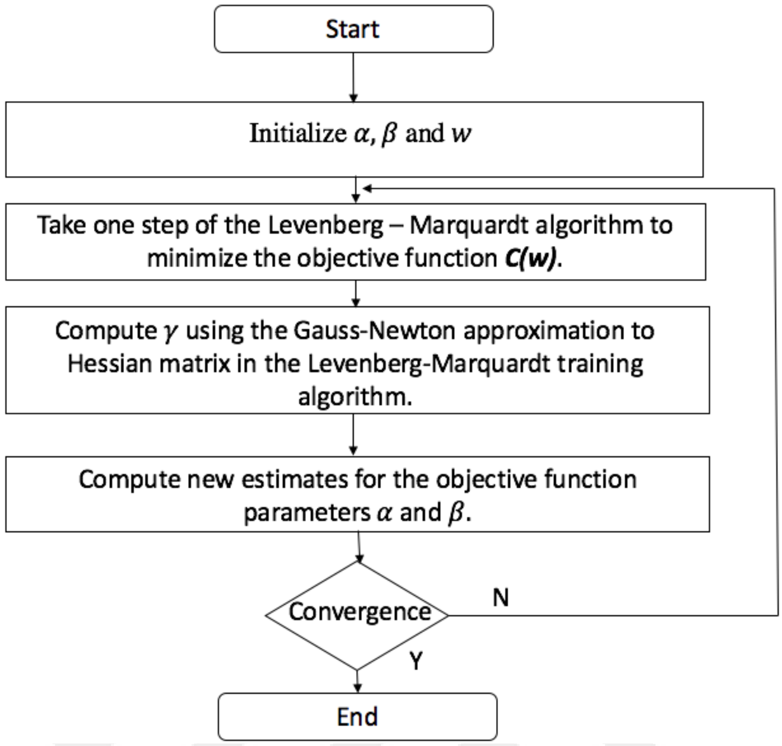

Figure 4: Bayesian Regularization Back Propagation process

Since ANN algorithm and nonlinear relationships are produced as a 'black box', it is not possible before hand to correctly identify which method will be superior, choose Bayesian Regularization or Levenberg-Marquardt Optimization. In this work, both training functions will be applied to the ANN to train the network.

3.3.3 Hidden Layer and Neuron Number Selection

In addition to learning type, learning algortihm and training algortihm selection; number of hidden layers and neurons is another parameter for the ANN model. As Karsoliya (2012) mentioned; 'The hidden layer is the collection of neurons which has activation function applied on it as well as provide an intermediate layer between the input layer and the output layer'. If the relationship between input data and results is linear then there is no need for a non-linear complex relationship and so there ise no need for a hidden layer. In contrast, if the relationship is complex or unknown; then at least one hidden layer is needed to solve the problem.

There is no certain formula for the number of hidden layers. Generally, a single layer is adequate with optimum number of neurons for creating an ANN for many problems (Bugmann et al., 2001). In contrast for deep neural networks with many inputs and outputs, like face recognition, generally two or even much more hidden layers is needed. In our study, ANNs will be created by using one hidden layer. Since we have 5 inputs and 1 output, there is no need 2 or more hidden layers.

Similarly, there is no way to choose hidden layer neuron number. There are some rule of thumb methods which forge a bond between input layer neuron number and output layer neuron number, as an example; the number of hidden layer neurons should be less than twice of the number of neurons in input layer. But these kind of methods can not be generalized since the ideal neuron number changes depend of the problem.

Additionally, hidden layer neuron number is very important parameter for the ANN model because it can cause over fitting or under fitting. ‘If the number of neurons are less as compared to the complexity of the problem data then “Underfitting” may occur. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set. If unnecessary more neurons are present in the network, then “Overfitting” may occur.’ (Karsoliya, 2012). In our study, different number of neurons will be applied to the model from 1 to 100 to see which neuron number gives the better result.

4. RESULTS

As detailed in chapter 3.3, learning type, learning algortihm, algorithm type, hidden layer and neuron number selection are the crucial tasks to create a proper ANN. In our study, learning type is chosen as 'supervised learning' since 77 completed software project data and these projects' input variable values are obtained. As learning algorithm, feed forward back propogation will be applied since it can provide complex non-linear relationships between input variables to achieve the optimum results. Also, one hidden layer will be used for ANN.

In the next subchapters Bayesian Regularization Back Propagation and Levenberg-Marquardt Back Propagation will be applied to the ANN model with different neuron numbers by using Matlab. These two algorithms' estimation results will be compared by their Mean Magnitude Relative Error (MMRE) which measures avarage estimation accuracy. The Magnitude Relative Error (MRE) of each project's estimate is defined as;

$$MRE = \frac{|actual\ effort - estimated\ effort|}{actual\ effort} \tag{16}$$

Additionally, besides Bayesian Regularization Back Propagation and Levenberg-Marquardt Back Propagation, the bank's first estimations and actual efforts will be compared to compare ANN with the real life scenarios.

4.1 Levenberg-Marquardt Back Propagation

In this study, neural network architecture is created on Matlab program by using data analysis features, especially Neural Network toolbox. Also, a spesific code is used to create ANN to find the ideal neuron number for the hidden layer. Code can be found in the appendix. Additionally, all project data is normalized by Neural Network Toolbox automatically.

Trainlm function is used as Levenberg-Marquardt Back Propagation training algortihm. %70 of the completed project data is used as training data, %15 of the completed project data is used as validation data and similarly %15 of the completed project data is used as test data set. Finally, as noticed before MRE is used find the error ratio of each project estimation and the best result is obtained with 10 neurons as shown in the Table 17.

Table 17: Levenberg-Marquardt Back Propagation ANN Results

Project No	Actual Effort on Completion (m/d)	ANN Results with Trainlm (m/d)	MRE
1	45	56,53797307	25,63994015
2	62	70,80007183	14,19366425
3	65	65,2196452	0,337915699
4	67	67,08557295	0,127720827
5	68	70,80007183	4,117752695
6	68	56,53797307	16,85592196
7	75	73,23965852	2,347121978
8	80	80,6576506	0,822063248
9	84	86,4456566	2,911495957
10	100	101,4535833	1,453583282
11	110	114,0596731	3,69061191
12	114	114,0596731	0,052344826
13	115	144,5462057	25,69235278
14	119	80,6576506	32,22046168
15	138	144,2760976	4,547896817
16	148	380,9015512	157,365913

17	155	169,8075498	9,553257913
18	158	151,8566416	3,888201519
19	170	181,9308698	7,018158727
20	171	170,1692297	0,485830568
21	183	183,0279477	0,015271953
22	185	144,5462057	21,86691584
23	189	78,42173829	58,50701678
24	195	194,7031287	0,15224169
25	202	203,6528837	0,818259276
26	205	209,3596201	2,126643944
27	208	209,3596201	0,653663503
28	211	211,0948401	0,044947908
29	219	235,2453029	7,417946515
30	223	221,3515537	0,739213601
31	226	221,3515537	2,056834659
32	238	238,0749363	0,031485826
33	238	237,6811236	0,133981682
34	240	209,3596201	12,76682496
35	243	251,2371012	3,389753582
36	251	235,2453029	6,276771766
37	266	268,5301725	0,95119267
38	270	271,0017357	0,371013234
39	275	286,3460033	4,125819375
40	280	288,746181	3,12363607
41	283	281,9557697	0,368985959
42	287	284,7407249	0,78720388
43	292	287,6292505	1,496832018
44	344	343,2784847	0,209742833
45	348	348,0150786	0,00433294
46	359	290,740555	19,01377297
47	363	364,9186949	0,528566078
48	366	359,8501545	1,680285648
49	407	405,8337655	0,286544099
50	429	429,5014932	0,116898193
51	429	452,9295502	5,577983729
52	449	450,1336305	0,252478959
53	453	673,2346017	48,61690986
54	488	487,6417129	0,073419496
55	503	645,9804616	28,42553909
56	541	539,9577917	0,192644794

57	579	578,9934435	0,001132386
58	598	597,0298297	0,162235839
59	630	629,7167462	0,044960924
60	636	636,0804048	0,012642258
61	655	653,8721487	0,172191033
62	673	472,8498001	29,73999999
63	692	762,4122108	10,17517498
64	745	744,8464449	0,020611418
65	823	823,9177239	0,111509583
66	854	2686,90309	214,6256546
67	1085	1273,146488	17,34069017
68	1429	1429,095656	0,006693943
69	1594	1594,006492	0,000407281
70	1651	1550,61249	6,080406429
71	1690	1690,269499	0,015946716
72	1899	1900,377159	0,072520211
73	1902	5521,816293	190,316314
74	1925	1923,733761	0,065778646
75	2463	2463,410339	0,016660125
76	2477	3380,769629	36,4864606
77	2996	2996,0405	0,001351817

As a result of ANN with trainlm training function, MMRE is calculated as 13,66224842 by using formula X where n is project number.

$$MMRE = \frac{\sum_{i=1}^{n} MRE_i}{n} \tag{17}$$

Additionaly, 'Neural Network Training Regression' diagrams are used to analyze the relationships between output and target results. For our study, target result is the completed projects' actual efforts and output is the result from ANN. If the R is equals to 1, this indicates that there is a perfect linear relationship between outputs and targets. On the contrary, when R is close to 0, then there is no linear relationship.

In the diagram, it is obvious that there is a nearly perfect linear relationship between output and target data for the training set since R is 0,99992. Automatic normalization has also an effect on R value. Similarly, R is 0,98764 for validation set, which is also very close to linear relationship. But in the test set R is 0,86946, which is getting close to non-linear relationship. These values can be a foreshow for a possible over fitting problem. For training set, there is nearly a perfect fit between target and output results, but when new data is added like test data set, fitting is getting poor.

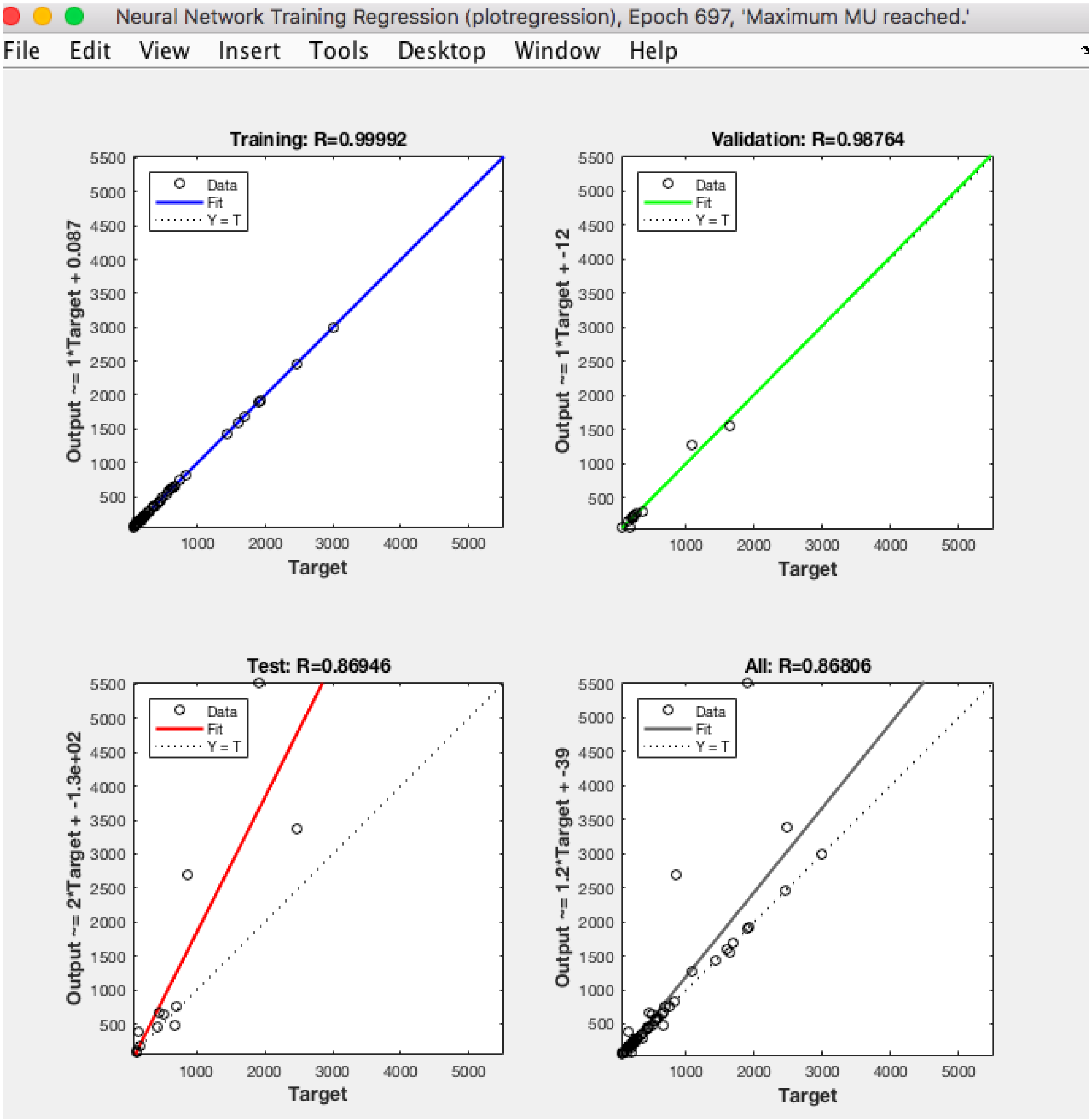

Figure 5: Neural Network Training Regression for Levenberg-Marquardt Algortihm

4.2 Bayesian Regularization Back Propagation

As Bayesian Regularization Back Propagation training algortihm *Trainbr* function is used. %70 of the completed project data is used as training data, %15 of the completed project data is used as validation data and similarly %15 of the completed project data is used as test data set. To compare Levenberg-Marquardt Back Propagation, MRE and MMRE is used to find error ratio.

As noticed different neuron numbers from 1 to 100 have been tried to find the optimum results and the best result is obtained with 86 neurons as shown in the Table 18. MMRE is calculated from MRE results, and found as 8,661127888.

Table 18: Bayesian Regularization Back Propagation ANN Results

Project No	**Actual Effort on Completion (m/d)**	**ANN Results with Trainbr (m/d)**	**MRE**
1	45	57,62967209	28,06593799
2	62	62,02423948	0,03909593
3	65	77,72991904	19,58449083
4	67	62,60849647	6,55448288
5	68	62,02423948	8,787883123
6	68	57,62967209	15,25048222
7	75	58,70231467	21,73024711
8	80	99,56125324	24,45156655
9	84	111,0246125	32,17215778
10	100	122,3250729	22,32507294
11	110	97,17949555	11,65500404
12	114	97,17949555	14,75482846
13	115	135,0863715	17,46641004
14	119	99,56125324	16,33508131
15	138	161,5858246	17,09117723
16	148	186,7728613	26,19787926
17	155	193,6078396	24,90828359
18	158	163,159314	3,265388602

19	170	152,3094605	10,40619968
20	171	162,5340565	4,950844138
21	183	169,3965617	7,433572832
22	185	135,0863715	26,9803397
23	189	191,300907	1,217411112
24	195	213,9985554	9,742848923
25	202	214,3018872	6,090043191
26	205	216,2538358	5,489676019
27	208	216,2538358	3,968190307
28	211	212,0965322	0,519683504
29	219	249,8165354	14,07147736
30	223	223,7057422	0,316476309
31	226	223,7057422	1,015158332
32	238	219,8520901	7,625172231
33	238	221,7092836	6,84483882
34	240	216,2538358	9,894235067
35	243	249,1312768	2,523159163
36	251	249,8165354	0,471499835
37	266	256,1740635	3,693961091
38	270	245,7152779	8,994341508
39	275	285,0131794	3,641156145
40	280	241,2768318	13,82970293
41	283	284,6708258	0,590397822
42	287	309,9167991	7,984947418
43	292	290,7336411	0,433684568
44	344	342,9041087	0,318573058
45	348	400,208444	15,00242644
46	359	310,6945223	13,45556481
47	363	316,2238911	12,88598042
48	366	309,1568471	15,5309161
49	407	416,1026972	2,236534943
50	429	409,7623531	4,484299978
51	429	432,7967803	0,885030375
52	449	473,4376027	5,442673214
53	453	515,8255205	13,86876833
54	488	419,399666	14,05744549
55	503	536,8730937	6,734213464
56	541	509,3516666	5,849969208
57	579	556,8579155	3,824194216
58	598	625,2146772	4,550949368

59	630	604,6991779	4,016003505
60	636	533,5649493	16,10614005
61	655	656,643127	0,250859081
62	673	586,8694426	12,79800258
63	692	732,795657	5,895326156
64	745	692,8144676	7,004769447
65	823	796,3994026	3,232150355
66	854	913,8148994	7,004086579
67	1085	1060,890765	2,222049342
68	1429	1456,281499	1,909132207
69	1594	1621,620095	1,732753757
70	1651	1753,037881	6,180368319
71	1690	1682,039759	0,471020183
72	1899	1917,32962	0,96522487
73	1902	1922,528584	1,079315654
74	1925	1909,865408	0,786212595
75	2463	2417,081211	1,864343836
76	2477	2490,676357	0,552133927
77	2996	2865,825967	4,344927657

'Neural Network Training Regression' results are shown below. Comparing to Levenberg-Marquardt Back Propagation R values are for all data sets are very close 1, which indicates a nearly linear relationship between target and output data sets. Automatic normalization has also an effect on R value. Also, it can be said that, there is no spesific overfitting or underfitting problems, since R values for test and validation data sets are nearly similar with the R value of training data set.

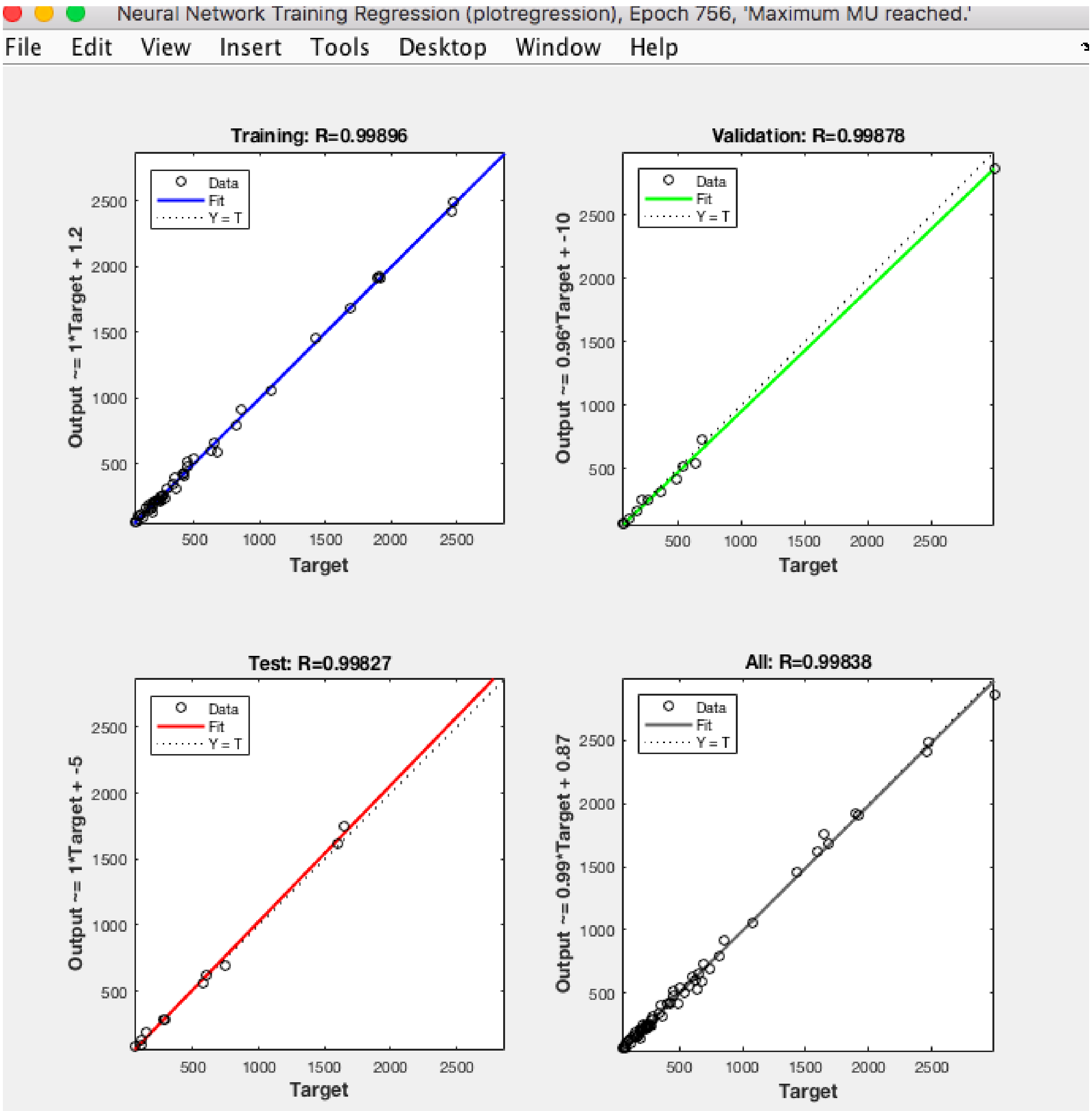

Figure 6: Neural Network Training Regression for Bayesian Regularization

4.2.1 Bayesian Regularization Back Propagation Results According to Project Size

As detailed in previous section, the best result with minimum MMRE is found with Bayesian Regularization Back Propagation algorithm. Since project data set includes 77 projects with different sizes (m/d), MMRE is calculated for different project sizes, to see if Bayesian Regularization estimates more accurately on a specific project size.

77 projects have differents efforts on completion which ranges between 45 m/d to 2996 m/d. 45 m/d to 100 m/d projects are grouped as “small project”, 101 m/d to 1000 m/d projects are grouped as “medium project” and 1001 m/d to 3000 m/d projects are groped as “large project”.

Based on Table 18, for each group MMRE is calculated again. As a result; as given in Table 19, for small projects, MMRE is 17,8961417. For medium projects, MMRE is 8,31853478. And for the large projects, MMRE is 2,00977112. It is obvious that, the existing ANN with Bayesian Regularization Back Propagation gives better estimation with large projects.

According to the results; it can be said that ANN performance shows a change depend on project size. As a future work, different ANNs can be created according to the project size. Also domain number is another important indicator of project size which is used in real-life scenerios. In this sense, project size and domain number relationship can be studied and new ANNs can be created within this context.

Table 19: MMRE Results According to Project Size

Type	Definition	MMRE
Small Project	Actual effort at completion is between 45 m/d and 100 m/d.	17,8961417
Medium Project	Actual effort at completion is between 101 m/d and 1000 m/d.	8,31853478
Large Project	Actual effort at completion is between 1001 m/d and 3000 m/d.	2,00977112

4.2.2 Sensitivity Analysis

5 input variables are selected according to expert opinion survey results as valuable to estimate software project effort accurately. As detailed in previous sections, ANN with Bayesian Regularization Back Propagation gives the best result with minimum MMRE for estimations.

In sensitivity analysis, an ANN is created with the top 4 input variables to see if ANN can estimate more accurately or same with less variables. These variables are “well-defined and stable requirements”, “dependency on 3rd party company’s code”, “multiple domain integration” and “reusable code”.

ANN is created with Bayesian Regularization Back Propagation algorithm and 1 hidden layer. ANN gave the best results with 27 neurons. In Table 20, the estimations and MRE values are shown. As a result, ANN made estimation with 0,70274273 MMRE which is significantly higher error margin than the main ANN with 5 input variables.

Table 20: MRE values for the ANN with 4 input variables

Project No	Actual Effort on Completion (m/d)	ANN Results with 4 Input Variables (m/d)	MRE
1	359	242,0867316	0,3256637
2	344	180,0120917	0,476709036
3	292	141,0432109	0,516975305
4	205	109,9355507	0,463729021
5	202	84,51858176	0,581591179
6	171	233,5423411	0,365744685
7	170	146,6761717	0,13719899
8	148	98,1159214	0,337054585
9	84	141,0432109	0,679085844
10	45	210,7937027	3,684304505
11	429	224,8168113	0,475951489
12	363	403,9540546	0,112821087
13	348	399,1101361	0,146868207
14	208	109,9355507	0,471463698
15	80	312,0710374	2,900887968
16	503	239,1974985	0,524458254
17	488	458,9901451	0,059446424
18	238	261,2929683	0,097869615
19	185	182,4884854	0,013575755
20	114	175,038386	0,535424439
21	67	406,2790251	5,063866046

22	65	311,3036219	3,789286492
23	429	261,4345923	0,390595356
24	219	381,6294953	0,742600435
25	115	182,4884854	0,586856395
26	854	406,2790251	0,524263437
27	673	527,6115666	0,216030362
28	579	300,9720439	0,480186453
29	280	294,8299631	0,052964154
30	270	259,4549882	0,039055599
31	183	173,5946856	0,051395161
32	75	294,8299631	2,931066174
33	68	89,18871215	0,311598708
34	68	210,7937027	2,099907393
35	62	89,18871215	0,438527615
36	251	381,6294953	0,520436236
37	238	332,2622961	0,396060068
38	745	447,9706864	0,398697065
39	541	580,7662086	0,073505007
40	287	359,6054559	0,252980683
41	266	406,2790251	0,527364756
42	407	349,7033741	0,140777951
43	1594	1647,6724	0,033671518
44	1085	534,9429876	0,506964988
45	636	503,4993453	0,208334363
46	453	513,6365584	0,133855537
47	275	517,1021182	0,880371339
48	2463	2324,647024	0,056172544
49	598	748,6052585	0,251848258
50	283	666,951028	1,356717413
51	630	784,5553473	0,245325948
52	1902	1662,303988	0,12602314
53	1690	1337,193803	0,208761063
54	655	1028,68506	0,570511542
55	366	513,6365584	0,403378575
56	1429	1896,830519	0,327383148
57	2996	2988,870715	0,002379601
58	1651	2543,337445	0,540483007
59	1899	1663,262365	0,124137775
60	1925	1454,302278	0,244518297
61	2477	2352,036889	0,050449379

62	211	447,1557281	1,11922146
63	692	905,2989529	0,308235481
64	119	312,0710374	1,622445693
65	189	399,1101361	1,111693842
66	100	319,0140474	2,190140474
67	223	476,5650577	1,137063039
68	449	658,974137	0,467648412
69	243	447,9706864	0,843500767
70	823	1215,50113	0,476915103
71	158	236,3516264	0,49589637
72	195	334,6148778	0,715973732
73	226	476,5650577	1,108694946
74	138	423,6250019	2,069746391
75	155	171,2266961	0,104688362
76	240	109,9355507	0,541935205
77	110	175,038386	0,591258055

‘Neural Network Training Regression’ results are shown below. R value is 0,93443 and lower than Bayesian Back Back Propagation with 5 input variables. For training set, R is 0,95095 which is close to 1 that means a nearly linear relationship between target and output data sets. In test set R value is 0,92305 and in validation set R value is 0,91845 which indicates a possible fitting problem comparing to training set.

As a result, for software effort estimation, ANN with 5 input variables gives much better results with less error margin comparing to ANN with 4 variables. For future work, ANN variable number may be increased to see if ANN would estimate better comparing to ANN with less variables.

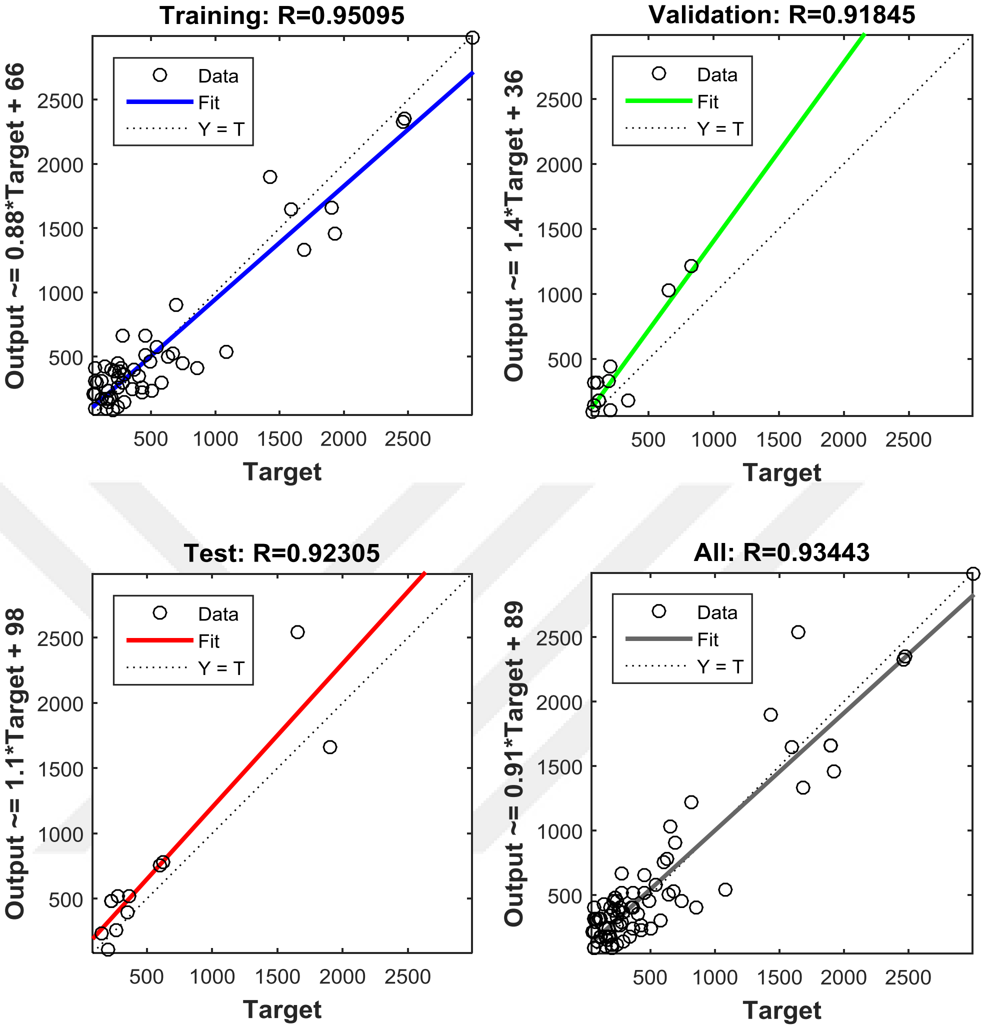

Figure 7: Neural Network Training Regression for Sensitivity Analysis

4.3 The Bank's Estimations

The Bank which provides the completed projects' data uses a custom estimation method, based on the number of the components that will be developed in the projects. These components can be interfaces, services, batches or reports and each component has a specific coefficient for the estimation according to their complexity group as simple, average or complex. At the beginning of the project, these components are estimated by the domain managers and experts and then project estimation is obtained.

In Table 21 baseline estimations for the completed projects and MRE values are shown. As a result, MMRE is found as 25,92084985 which is very high comparing to ANN results both for Levenberg-Marquardt Back Propagation and Bayesian Regularization Back Propagation.

The bank which we gathered data, spent 160.000 m/d (actual effort) for IT projects in 2016. By using the bank's existing effort estimation model, the estimations at beginning of the projects were like ∓ %25, which means approximately 120.000 m/d or 200.000 m/d. By using ANN model which is created, the estimation could be 172.800 m/d or 147.200 m/d based on ∓ %8,6 MMRE of Bayesian Regularization. That means 27.200 m/d resources saving for the bank annually. As we assume that 1 person (resource) works 250 m/d in a year, 27.200 m/d means 109 resources will be saved which is very critical for the annual budget.

Table 21: MRE values for the bank's estimation

Project No	Actual Effort on Completion (m/d)	Baseline Estimation (m/d)	MRE
1	45	69	53,33333333
2	62	80	29,03225806
3	65	65	0
4	67	50	25,37313433
5	68	77	13,23529412
6	68	44	35,29411765
7	75	47	37,33333333
8	80	90	12,5
9	84	87	3,571428571
10	100	100	0
11	110	110	0
12	114	100	12,28070175
13	115	117	1,739130435
14	119	120	0,840336134
15	138	124	10,14492754
16	148	110	25,67567568
17	155	144	7,096774194
18	158	167	5,696202532

19	170	135	20,58823529
20	171	60	64,9122807
21	183	195	6,557377049
22	185	191	3,243243243
23	189	150	20,63492063
24	195	143	26,66666667
25	202	90	55,44554455
26	205	244	19,02439024
27	208	250	20,19230769
28	211	290	37,44075829
29	219	259	18,26484018
30	223	200	10,31390135
31	226	231	2,212389381
32	238	250	5,042016807
33	238	430	80,67226891
34	240	290	20,83333333
35	243	275	13,16872428
36	251	270	7,569721116
37	266	350	31,57894737
38	270	149	44,81481481
39	275	272	1,090909091
40	280	300	7,142857143
41	283	250	11,66077739
42	287	382	33,1010453
43	292	290	0,684931507
44	344	415	20,63953488
45	348	570	63,79310345
46	359	300	16,43454039
47	363	368	1,377410468
48	366	442	20,76502732
49	407	373	8,353808354
50	429	444	3,496503497
51	429	780	81,81818182
52	449	560	24,72160356
53	453	807	78,14569536
54	488	380	22,13114754
55	503	275	45,32803181
56	541	462	14,6025878
57	579	100	82,72884283
58	598	738	23,41137124

59	630	710	12,6984127
60	636	829	30,34591195
61	655	700	6,870229008
62	673	802	19,1679049
63	692	600	13,29479769
64	745	661	11,27516779
65	823	1384	68,16524909
66	854	600	29,74238876
67	1085	2000	84,33179724
68	1429	1898	32,82015395
69	1594	805	49,49811794
70	1651	2000	21,13870382
71	1690	2277	34,73372781
72	1899	1040	45,23433386
73	1902	1178	38,06519453
74	1925	2740	42,33766234
75	2463	1970	20,01624036
76	2477	2010	18,85345176
77	2996	5200	73,564753

5. CONCLUSION

Software projects are essential tools of a typical organization to develop new applications and platforms. However, mostly due to inherent complexities of these projects combined with limited resources and time constraints, projects tend to overshoot initial resource estimations. Moreover, as software projects continually are added to the list of current tasks or changed to respond to changing customer needs and/or competitors' offerings, accurate effort estimations are needed to manage resources efficiently/effectively. In literature, different methods and models have been proposed to calculate software projects' efforts. Though, these approaches tend to fail in real life scenarios due to the fact that own organization based tailored solutions are usually required to correctly estimate teams' efforts.

Artificial neural networks with the ability to handle complex relationships and to adapt to changing conditions seems to attract a lot of attention recently. Software development effort estimation is one the areas that will benefit from adaptable and learning frameworks. Therefore, in this thesis we build a software estimation model by using neural network methodology. The features for the network were chosen as a result of a survey realized at one of the largest banks in Turkey. The findings suggest that current approaches used at the bank mostly lack accuracy and ANN based methodology is handling the uncertainties and complexities pretty effectively and therefore is a superior approach than the classical algorithmic estimation models at least for the current scenario.

As future work, historical project data set could be extended to handle possible overfitting issues of the neural network model. In addition, different ANNs can be created for different size of projects for effort estimation by grouping projects based on domain number. Also, input variable set could be augmented by using other preselected factors. Similarly, to generalize effort estimation model, input variable selection surveys can be realized with IT experts from different sectors like telecom or insurance.

REFERENCES

Abts, C., Horowitz, E., Brown, A.W., Madachy, R., Chulani, S., Reifer, D., Clark B., Steece B. (2000). COCOMO II Model Definition Manual. URL: *http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf*

Agatonovic-Kustrin, S., Beresford, R. (2000). Basic concepts of artificial neural network (ANN) modeling, *Journal of Pharmaceutical and Biomedical Analysis:* 22 (5) : 717–727.

Aljahdali, S., Sheta, A.F., Debnath, N.C. (2015). *Estimating Software Effort and Function Point Using Regression, Support Vector Machine and Artificial Neural Networks Models, 12th International Conference of Computer Systems and Applications*, Marrakech, Morocco, pp. 1-8.

Baik, J. (2000). *The Effects of Case Tools on Software Development Effort*, Doctoral dissertation, University of Southern California.

Banerjee, G. (2001). Use Case Points - An Estimated Approach. URL: *http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/reporty/use_case_points.pdf*

Borade, J.G., Khalkar, V. (2013). Software Project Effort and Cost Estimation Techniques. IJARCSSE [online]. 3 (8). URL: http://www.ijarcsse.com/docs/papers/Volume_3/8_August2013/V3I7-0468.pdf

Burden, F., Winkler, D. (2008). Bayesian regularization of neural networks. URL: *https://www.ncbi.nlm.nih.gov/pubmed/19065804*

Chandrasekaran, S., Gudlavalleti, S., Kaniyar S. (2014). Achieving success in large, complex software projects. URL: *http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/achieving-success-in-large-complex-software-projects*

Chiang, Y.-M., Li-Chiu Chang, F.-J. C. (2004). Comparison of static-feedforward and dynamic-feedback, *Journal of Hydrology*: 290 (3-4) : 297–311.

Cockburn A. (2011). Writing Effective Use Cases, 23rd edn, Addison-Wesley US.

Cohn, M. (2015). Estimating with Use Case Points. URL: *https://www.mountaingoatsoftware.com/articles/estimating-with-use-case-points*

Finnie, G.R., Wittig, G.E. (1997). A comparison of software effort estimation techniques: Using function points with neural networks, case-based reasoning and regression models. Journal of Systems and Software [online]. 39 (3), pp. 281-289. URL: *http://www.sciencedirect.com/science/article/pii/S0164121297000551* [accessed 24 March 2017].

Gabrani, G. and Saini, N. (2016). Effort Estimation Models Using Evolutionary Learning Algorithms for Software Development, *Symposium on Colossal Data Analysis and Networking*, CDAN'16, Indore, India, pp. 1-6.

Hira, A. , Sharma, S. Boehm, B. (2016). Calibrating COCOMO® II for Projects with High Personnel
Turnover, *Proceedings of the International Conference on Software and Systems Process*, ICSSP '16 , ACM New York, NY, USA, pp. 51-55.

Hirschen, K. and Schafer, M. (2005). Bayesian regularization neural networks for optimizing
fluid flow processes. Comput. Methods Appl. Mech. Engrg. [online]. 195 (7-8), pp. 481-500. URL: *http://www.sciencedirect.com/science/article/pii/S0045782505000617* [accessed March 26, 2017].

Jacobson, L. (2014). Introduction to Artificial Neural Networks Part 2 - Learning. URL: *http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-2-learning/8*

Jorgensen M., Shepperd M., (2007). A Systematic Review Of Software Development Cost Estimation Studies, IEEE Transactions on Software Engineering [online]. 33 (1), pp. 33-53. URL: *http://ieeexplore.ieee.org/document/4027147/* [accessed 1 May 2017].

Karsoliya, S. (2012). Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture, International Journal of Engineering Trends and Technology [online]. 3 (6), pp. 717-717. URL: http://ijettjournal.org/volume-3/issue-6/IJETT-V3I6P206.pdf [accessed 1 May 2017].

Kumari, W. and Pushkar, S. (2013). Performance Analysis of the Software Cost Estimation Methods: A Review. IJARCSSE [online]. 3 (7). URL: *http://www.ijarcsse.com/docs/papers/Volume_3/7_July2013/V3I7-0247.pdf*

Leinonen, J. (2016). *Evaluating Software Development Effort Estimation Process in Agile Software Development Context*, Master’s Thesis, University of Oulu.

Lokan, C.J. (2000). An empirical analysis of function point adjustment factors. Information and Software Technology [online]. 42 (9), pp. 649–659. URL: http://www.sciencedirect.com/science/article/pii/S0950584900001087 [accessed March 26, 2017].

Mathworks (2016): trainbr. URL: *https://www.mathworks.com/help/nnet/ref/trainbr.html*

May, R., Dandy, G., Maier, H. (2011). Review of Input Variable Selection Methods for Artificial Neural Networks, *Artificial Neural Networks - Methodological Advances and Biomedical Application,* InTech, Chapter 3.

Mulcahy R. (2015). Rita Mulcahy's PMP Exam Prep, eight edn, RMC Publications US.

Murphy, K. (1998). A brief introduction to reinforcement learning. URL: *http://www.cs.ubc.ca/~murphyk/Bayes/pomdp.html*

Negnevitsky M. (2002). Artificial Intelligence A Guide to Intelligent Systems, second edn, Pearson England

Nielsen M. (2017): Using Neural Nets to Recognize Handwritten Digits. URL: http://neuralnetworksanddeeplearning.com/chap1.html

Pandya, D.A., Dennis, B.H., Russell, R.D. (2017). A computational fluid dynamics based artificial neural network model to predict solid particle erosion, WEAR [online]. Vol 378-379, pp. 198-210. URL: http://dx.doi.org/10.1016/j.wear.2017.02.028 [accessed 1 May 2017]

Rafiq, M.Y., Bugmann, G., Easterbrook, D.J. (2001). Neural network design for engineering applications, Computers and Structures [online]. 79 (17), pp. 1541-1552. URL: http://www.sciencedirect.com/science/article/pii/S0045794901000396 [accessed 1 May 2017].

Ren, A. and Yun C. (2013). Research of Software Size Estimation Method, *International Conference on Cloud and Service Computing*, CSC 2013, Beijing, China, pp. 154-155.

Ribu, K. (2001). *Estimating Object-Oriented Software Projects with Use Cases,* Master of Science Thesis, University of Oslo.

Rush, C., Roy, R. (2001). Expert judgement in cost estimating: Modelling the reasoning process, Concurrent Engineering [online]. 9 (4), pp. 271-284. URL: https://doi.org/10.1177/1063293X0100900404 [accessed 1 May 2017].

Sadhu, A.K. (2014). DELPHI TECHNIQUE FOR TECHNOLOGY FORECASTING. URL: *http://managementversity.com/delphi-technique/*

Santani, D., Bundele, M., Rijwani, P. (2014). Artificial Neural Networks for Software Effort Estimation: A Review, IJAEST [online]. V3/N3, pp. 193-200. URL: http://www.ijaestonline.com/admin/post_image/1419591344_Artificial_Neural_Networks_for_Software_Effort_Estimation_A_Review.pdf [accessed 1 May 2017].

Souza, C. (2009). Neural Network Learning by the Levenberg-Marquardt Algorithm with Bayesian Regularization (part 2). URL: *http://crsouza.com/2009/11/18/neural-network-learning-by-the-levenberg-marquardt-algorithm-with-bayesian-regularization-part-2/*

Trendowicz, A., Münch, J., Jeffery, R. (2011). State of the Practice in Software Effort Estimation: A Survey and Literature, *Central and East European Conference on Software Engineering Techniques*, CEE-SET 2008, Brno, Czech Republic, pp. 232-245.

Usharani, K., Vignaraj Ananth, V., Velmurugan, D. (2016). A Survey on Software Effort Estimation,
International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016, Chennai, India pp. 505-509

Wilamowski, B.M., Yu, H. (2010). Improved Computation for Levenberg–Marquardt Training, IEEE [online]. 21 (6), pp. 930-937. URL: http://ieeexplore.ieee.org/document/5451114/ [accessed 1 May 2017].

Wilamowski, B.M., Yu, H. (2011). Levenberg–Marquardt Training, *Industrial Electronics Handbook, vol. 5 – Intelligent Systems*, CRC Press US, pp. 12-1 to 12-15.

Yue, Z., Songzheng, Z., Tianshi, L. (2011). Bayesian Regularization BP Neural Network Model for Predicting Oil-gas Drilling Cost, 2011 *International Conference on Business Management and Electronic Information*, Guangzhou, pp. 483-487.

APPENDICES

Appendix A

```
% Solve an Input-Output Fitting problem with a Neural Network
% Script generated by Neural Fitting app
% Created 29-Dec-2016 15:39:43
%
% This script assumes these variables are defined:
%
%   Input - input data.
%   Output - target data.

x = Input;
t = Output;
performance_history =  [];
trainPerformance_history = [];
testPerformance_history = [];
network_performance_history = [];
error_percentage_history = [];
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainbr';  % Bayesian Regularization backpropagation.

% Create a Fitting Network

hiddenLayerSize = [86];
net = feedforwardnet(hiddenLayerSize,trainFcn);
net.layers{1}.transferFcn = 'tansig';
% net.layers{2}.transferFcn = 'purelin';
% net.layers{3}.transferFcn = 'purelin';
% net.layers{4}.transferFcn = 'logsig';
% % Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};
% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
net.trainParam.max_fail=1000;
net.trainParam.epochs=10000;
net.trainParam.lr=0.05;
net.trainParam.mc=0.9;
% net.trainParam.mu_max = 1e20;
% Choose a Performance Function
```

```matlab
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse';  % Mean Squared Error

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
error_percentage = sum(abs(e./t))/length(t);
performance = perform(net,t,y);

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

network_performance_history = [network_performance_history; hiddenLayerSize performance];
trainPerformance_history = [trainPerformance_history trainPerformance];
testPerformance_history = [testPerformance_history testPerformance];
error_percentage_history = [error_percentage_history; hiddenLayerSize error_percentage];
% View the Network
% view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','MatrixOnly','yes');
    y = myNeuralNetworkFunction(x);
end
```

```
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end

trainPerformance_avg = mean(trainPerformance_history);
testPerformance_avg = mean(testPerformance_history);
performance_history = [performance_history; hiddenLayerSize trainPerformance_avg
testPerformance_avg];
```

BIOGRAPHICAL SKETCH

Tuğçe Uğurlu was born in 1988. She attended Gaziantep Anatolian High School in 2002. She studied Mathematics Engineering in Istanbul Technical University between 2005 – 2009. After bachelor, she entered Galatasaray University Industrial Engineering Master Program in 2009.

In 2010, Uğurlu started working at Yapı Kredi Bank Information Technologies Department as Business Analyst in Digital Banking Channels division. Currently she is working as Project Manager at Project and Program Management Office.