Software Development Effort Estimation by Using Neural Networks – A Case Study

Tuğçe Uğurlu Altuntaş[1], S. Emre Alptekin[2*]
[1]Institute of Science, Industrial Engineering Program
[2]Industrial Engineering Department
*corresponding author
Galatasaray University
Ciragan Cad. No.36 34357 İstanbul
TURKEY
ugurlut@gmail.com, ealptekin@gsu.edu.tr

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 paper 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.

Key-Words: - Software development effort estimation, neural networks, back propagation algorithm

1 Introduction

A project is a temporary endeavor with a beginning and an end which creates a unique product or service [1]. An effort estimation is a prediction of how long a development activity will take to finish [2].

Since software industry and digitalization gained in importance, software effort estimation became 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 gone so bad that the existence of the company is threatened [3].

Both under and over estimation results in waste of time, resources, money and even lost prestige. According to Brode and Khalkar underestimating the costs is characterized by budget overruns, under developed functions and poor quality end-product [4]. Similarly, overestimation commits too many resources to the projects and could lead to lost contracts could mean lost jobs. Rita Mulhacy defines the term "padding", which is related with overestimating, as a sign of poor project management which can damage reputation of a project manager [5].

Since 1970s many studies and methods have been published 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 [2]. COCOMO is the first algorithmic effort estimation model 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 paper, 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 paper 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 Related Work

Software project effort estimation is a continuous activity starting with the initiation phase and continuing until closing phase [4]. There are a lot of software cost 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.

The most common used estimation approaches are expert judgement based methods in software industry [6]. 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, expert judgement estimation can be used if there is no quantified data for the project [4].

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 states, even if a person has experience, this does not mean that his/her estimates are accurate [7]. 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 [5].

Algorithmic effort estimation methods consist of mathematical models or calculations to provide effort estimation [8]. 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.

The Constructive Cost Model (COCOMO) is an algorithmic effort estimation 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, COCOMO II was developed as a successor of COCOMO. '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)' [9]. In COCOMO II, software projects are classified into three groups as organic, semi-detached and embedded projects. Organic projects are the projects, which are made of small teams or have few domains with good experience. Semi-detached projects are made of medium sized teams and have mixed experience among team members. Embedded 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; product attributes, hardware attributes, personnel attributes and project attributes. These cost drivers also referred to 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.

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' [10]. 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 grouped into simple, average and complex groups according to their step numbers. Each group has different weights. After calculating UUCW, unadjusted actor weight (UAW) is calculated. In a software project, there can be many different types of actors like client, customer, database, GUI etc. Similar to UUCW calculation, actors are grouped into three categories; simple, average and complex. Likewise, each group has different weights. Next, technical (TCF) and environmental (ECF) complexity factors are calculated. In total, there are 13 technical and 8 environmental factors. Once again, each factor has a different weight.

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 an optimal solution, so evolutionary learning based methods became popular in recent years [11]. Artificial neural network (ANN) is the most widely applied model under the umbrella terms Artificial Intelligence and Machine Learning. 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 [12]. As their structure is based on an abstraction of human brain, ANNs are able to learn and adapt to different conditions. A typical ANN as is made up from nodes in three layers; input layer, hidden layer(s) and output layer as shown in Figure 1 [13].

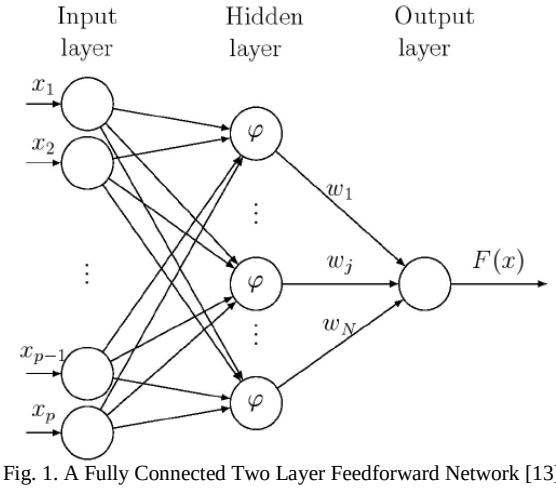

Fig. 1. A Fully Connected Two Layer Feedforward Network [13]

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.

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. Feed forward computation uses the input and the hidden layer nodes to compute output value [14]. Back-propagation is used to correct the errors made during the feed-forward phase. The algorithm iteratively adjusts weights starting from the output layer towards the input layer. When errors values reach target values, the back-propagation algorithm is ended [14]. The resulting trained network with the associated weights is ready to be used for new inputs.

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 paper, selected historical projects' data will be used to build an ANN model.

3 Proposed Methodology

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 surveys and interviews. 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.

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. Similarly, to build a Value Adjustment Factor (VAF), 14 'General System Characteristics' (GSCs) are used in Function Point Analysis (FPA) [15]. General system characteristics are also known as technical factors. GSCs has some common factors with UCP technical factors. 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 [16]. Jensen model defines 13 environmental factors. 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.

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 factor. As a result, 22 factors are identified as having a considerable effect on software project effort.

After the preselection, a survey is conducted on 19 IT experts to analyse 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 scaled from "1-Irrelevant" to "5-Highly Relevant" according to the effect on software development effort estimation by the experts.

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.

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

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 choices 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".

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 [17]. 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 [18]. Besides learning algorithm selection, also learning type and training function selection is also very important to create an ANN. According to problem and obtained data type, there are three main learning types; supervised learning, unsupervised learning and reinforcement learning [18].

In supervised learning, the desired output is provided along with the input values. When both input and output variables is provided in the neural network, and error based calculation is possible based on target output and actual output [18]. In unsupervised learning, only input variables are given and no output variable is defined. Unsupervised learning is able to find the structure or relationships between different inputs. The widely known examples for unsupervised learning are clustering, anomaly detection and blind signal separation. The third popular learning type is reinforcement learning, which is very similar to supervised learning. Reinforcement learning is defined as the problem of getting an agent to act in the world so as to maximize its rewards [18]. In this

learning type, instead of actual outputs a reward is given to neural network.

In this paper, supervised learning is selected as the learning type for the effort estimation with 77 completed project data with input and actual output variables provided to create the ANN. Learning algorithms are used to obtain weights of each neuron and relationships between neurons and layers while training the ANN. The most widely known learning algorithm for supervised learning is multi-layer perceptron with feed-forward network and back-propagation learning.

Feed forward structure defines a straightforward network that associate inputs with outputs. 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.

In Levenberg-Marquardt, all weights are updated according to Levenberg-Marquardt optimization which is also known as Damped Least-Squares method. Damped Least-Squares method is used for solving non-linear least square problems, especially in least squares curve fitting. Similarly, in 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. This process is called Bayesian regularization [19]. Bayesian Regularization obtains a well-defined statistical problem from a nonlinear regression in the manner of ridge regression [19]. 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 [20]. 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 paper, both training functions will be applied to the ANN to train the network.

4 Model Evaluation

4.1 Data Preparation

Artificial Neural Networks are inspired by human brain's nervous system. One of the most interesting character of human brain is ability to learn. Similar to human brain, ANNs learn and when they are learning they need historical data to create the complex nonlinear relationships between input and output variables.

In this study, an ANN has been created for software developı day a team member has spended. At the end of the projects, all projects' accumulated actual effort information calculated from each resources' time sheets. For the proposed ANN, actual effort is set as the target value.

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 1.

77 projects' project managers are asked to give a grade for each project's factors. 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.

Table 1: Chosen Factors and Scales

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 need to be developed with reusable code, 0 if not.	1	0			
Complex security requirements	From 1 to 5. 1 if the project doesn't need any security developments, 5 for highly complex security needs.	1	2	3	4	5

4.2 Results

As detailed in data preparation section ANN is created with 5 chosen factors. Both Bayesian

Regularization and Levenberg-Marquardt Optimization training functions are applied to ANN. 77 completed project data is used to train ANN with scaled input values and actual efforts. Project data is divided as training, validation and test data sets with the ratios %70, %15 and %15 in order.

Magnitude Relative Error (MRE) is used to compare training functions. ANN with Levenberg-Marquardt Optimization function with a % 8.471 MMRE.

$$MRE = \frac{|actual\ \ effort - estimated\ \ effort\ |}{actual\ \ effort} \quad (2)$$

Additionally, margin of error for the bank's estimation with its own estimation technique is calculated by using bank's initial effort estimation and actual effort. As a result, bank's MMRE is found as %25.921. As a sample, 30 projects' actual efforts, ANN's estimations and bank's estimations are shown in Table 2.

Table 2: Sample Project Effort Estimations

Project No	Actual Effort	ANN - Bayesian Regularization	ANN - Levenberg-Marquardt Optimization	The Bank's Initial Estimation	ANN - Bayesian Regularization MRE	ANN - Levenberg-Marquardt Optimization MRE	The Bank's Estimation MRE
1	119	104.850	200.002	120	11.891	68.069	0.840
2	138	169.174	206.616	124	22.590	49.722	10.145
3	148	190.406	201.814	110	28.652	36.361	25.676
4	155	190.960	237.662	144	23.200	53.330	7.097
5	158	159.648	168.695	167	1.043	6.769	5.696
6	170	154.387	162.881	135	9.184	4.187	20.588
7	171	165.355	189.700	60	3.301	10.936	64.912
8	183	175.585	174.637	195	4.052	4.570	6.557
9	185	130.266	141.282	191	29.586	23.631	3.243
10	189	197.820	194.419	150	4.666	2.867	20.635
11	195	210.936	271.327	143	8.172	39.142	26.667
12	202	208.980	207.350	90	3.455	2.648	55.446
13	205	221.196	278.847	244	7.900	36.023	19.024
14	208	221.196	278.847	250	6.344	34.061	20.192
15	211	221.337	248.165	290	4.899	17.614	37.441
16	219	244.172	204.414	259	11.494	6.660	18.265
17	223	222.103	223.817	200	0.402	0.366	10.314
18	226	222.103	223.817	231	1.724	0.966	2.212
19	238	217.048	224.889	250	8.803	5.509	5.042
20	238	229.158	115.457	430	3.715	51.488	80.672
21	240	221.196	278.847	290	7.835	16.186	20.833
22	243	255.113	241.758	275	4.985	0.511	13.169
23	251	244.172	204.414	270	2.720	18.560	7.570
24	266	264.370	321.441	350	0.613	20.843	31.579
25	270	255.052	278.263	149	5.536	3.060	44.815
26	275	284.055	282.679	272	3.293	2.792	1.091
27	280	246.709	254.323	300	11.889	9.171	7.143
28	283	298.423	381.417	250	5.450	34.776	11.661
29	287	316.608	347.359	382	10.316	21.031	33.101
30	292	280.311	366.780	290	4.003	25.610	0.685

MMRE values of the Levenberg-Marquardt Optimization and the Bayesian Regularization are compared to find the best working ANN model. Since the error rate difference is significant, Bayesian Regularization is chosen as the optimum learning algorithm for the software project effort estimation neural network. According to McKinsey and Oxford University's studies, on average, %66 of the large software project run over budget [3]. Considering a %66 error rate, %8.471 is a notably improved value. Similarly, comparing to the bank's own initial estimation, which is based on the numbers of the components to be developed, ANN with Bayesian Regularization is producing more accurate results.

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 paper 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. 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.

6 Acknowledgement

This research has been financially supported by Galatasaray University Research Fund, with the project number 16.402.015.

References:

[1] Project Management Institute, *A Guide to the Project Management Body of Knowledge*, PMBOK Guide, fifth edition, 2013.

[2] J. Leinonen, *Evaluating Software Development Effort Estimation Process in Agile Software Development Context*, Master's Thesis, University of Oulu, 2016.

[3] S. Chandrasekaran, S. Gudlavalleti, and S. Kaniyar, Achieving Success in Large, Complex Software Projects, *McKinsey and Company, Digital McKinsey Article*, July 2014.

[4] J.G. Borade, and V. Khalkar, Software Project Effort and Cost Estimation Techniques, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3(8), 2013, pp.730-739.

[5] R. Mulcahy, *Rita Mulcahy's PMP Exam Prep*, Rita Mulcahy, eight edition, 2013.

[6] M. Jorgensen, and M. Shepperd, (2007). A Systematic Review of Software Development Cost Estimation Studies, *IEEE Transactions on Software Engineering*, Vol. 33(1), 2007, pp.33-53.

[7] B. Boehm, C. Abts, and S. Chulani, Software Development Cost Estimation Approaches – A Survey, *Annals of Software Engineering*, Vol. 10(1), 2000, pp.177-205.

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

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

[10] A. Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2001.

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

[12] G.R. Finnie, and G.E. Wittig, A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models. *Journal of Systems and Software*, Vol. 39 (3), 1997, pp.281-289.

[13] S. Aljahdali, A.F. Sheta, and N.C. Debnath, 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, 2015, pp.1-8.

[14] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning Internal Representations by Error Propagation, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. 1, MIT Press, Cambridge, MA, 1986, pp.318-362.

[15] C.J. Lokan, An empirical analysis of function point adjustment factors, *Information and Software Technology*, Vol. 42 (9), 2000, pp.649–659.

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

[17] S. Agatonovic-Kustrin, and R. Beresford, Basic concepts of artificial neural network (ANN) modelling, *Journal of Pharmaceutical and*

Biomedical Analysis, Vol. 22, 2000, pp.717–727.

[18] D.J.C. MacKay, Bayesian Interpolation, *Neural Computation*, Vol. 4(3), 1992, pp.415-447.

[19] F. Burden, and D. Winkle, Bayesian Regularization of Neural Networks, Artificial Neural Networks Methods and Applications of the series Methods in Molecular Biology, Vol. 458, 2009, pp.23-42.

[20] K. Hirschen, and M. Schafer, Bayesian Regularization Neural Networks for Optimizing Fluid Flow Processes, *Computer Methods in Applied Mechanics and Engineering*, Vol. 195(7-8), 2006, pp.481-500.