Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOFTWARE EFFORT ESTIMATION USING HIERARCHICAL ATTENTION NEURAL NETWORK

HAITHEM KASSEM[1], KHALED MAHAR[2], AMANI SAAD[3]

[1]Multimedia Center, AASTMT, Egypt

[2]College of Computing and Information Technology, AASTMT, Egypt

[3]College of Engineering and Technology, AASTMT, Egypt

E-mail: [1] haithem_k@aast.edu, [2] khmahar@aast.edu, [3]amani.saad@aast.edu

ABSTRACT

In every software project, a software effort estimation process is not only vital but also extremely critical. Project success or failure depends massively on, the concise knowledge of effort and schedule estimates. The development of agile techniques in the field of software development has presented researchers and practitioners with many opportunities and challenges. An estimated effort for agile software development is one of the main challenges. Although traditional estimates of effort are used to estimate effort for agile software projects, most of them lead to inaccurate estimates. This paper focuses on the development of the agile effort estimation model. A machine learning classifier that uses information contained within an issue report is proposed to classify the difficulty or the weight of a given task according to a range of story point scales. The model has two levels of attention mechanisms implemented at the word and sentence levels, allowing it to pay distinguished attention to more and less relevant semantic features when constructing the document representation. The proposed model has achieved 87% classification accuracy. An empirical evaluation demonstrates that our approach has a greater or at least equivalent F score, Precision, and Recall when compared to classical classifiers.

Keywords: *Software Effort Estimation, Story Points, Deep Learning, GloVe, Hierarchical Attention Networks, Agile*

1. INTRODUCTION

The main goal for all software project managers is to complete the project on time and within the given budget. Many companies have chosen to use agile approaches to guide software development since the release of the agile manifesto [1]. Estimating effort is important for successful project management in the agile sense. Indeed, accurate estimates are needed to avoid inefficient resource allocation [2], [3].

The story points [4], [5] are a common way to estimate task effort. Story points are usually allocated in the context of agile development through organized group meetings known as Planning Poker sessions [6]. These meetings rely heavily on human judgment: the more the developers understand the job, the more reliable their estimation will be. Human judgment, on the other hand, can be limited by a number of factors. Humans are positive by nature, and this bias is amplified in group interactions [7], [8], [9]. Furthermore, the presence of a project manager, other senior developers, or dominant personalities in the meeting has been shown to affect developer estimation [10].

In order to solve these issues, we propose a machine learning classifier that uses information contained within an issue report to classify the difficulty or the weight for a given task according to a range of story point scale. Issue reports were considered because they are frequently used by developers to describe development tasks and are also heavily used in Planning Poker sessions [11].

There are three advantages of using machine learning classifier. To begin with, the classifier has comprehensive knowledge of the project dating back to its inception, and it bases its predictions on all previous issues in the issue tracking system. Second, since the classifier's estimates can be traced back to the features used for classification, it is not affected or coerced by other people. Third, the estimation is repeatable and predictable: the system never gets tired and consistently produces the same results.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

We propose a prediction model that assists teams by recommending a story point estimate for a given user story. To predict the size of new issues, the model learns from the team's previous story point estimates. This prediction system will be used in conjunction with (rather than in place of) the team's existing estimation techniques. It could also be used as a decision support system to assist in the estimation process. This is analogous to the concept of combination-based effort estimation, in which estimates are derived from various sources, such as a combination of expert and formal model-based estimates [38, 64].

The proposed model learns semantic features that represent the meaning of user stories or issue reports automatically, freeing users from manually designing and extracting features. Feature engineering typically relies on domain experts who use their specific knowledge of the data to create features on which machine learning techniques can be applied.

The key novelty of our approach resides in employing hierarchical attention networks (HAN) for categorizing user stories depending on a number of story points.

We consider the classifier as a member of the team, sharing its predictions with other developers. In this way, the classifier acts as an extra creator, generating a real-time calculation based on objective analysis.

An empirical evaluation has been carried out to answer the following research questions:

RQ1: What is the effect of adding attention layer during the training phase?

RQ2: Can attention mechanism be leverage to improve software effort estimation?

RQ3: Does the use of Hierarchical Attention Networks provide more accurate story point estimates than using traditional classification technique?

The remainder of the paper is organized as follows: Section 2 provides context for story points and Deep Learning. Section 3 presents related work, while Section 4 focuses on the design of the proposed model. Section 5 discusses experimental results and analysis, Section 6 shows future work and finally Section 7 presents the conclusion.

2. BACKGROUND

2.1 Story Points

In agile development, user stories or issues commonly describe what has to be built in the software project. Story points are the most common unit of measure used for estimating the effort involved in completing a user story or resolving an issue. The story point is a metric that agile teams use to estimate the amount of effort required to complete a development task [4], [5].

Instead of quantifying the amount of work required to complete a given task, the number of story points is an estimate of how difficult a given task is for the development team. The team usually agrees on the number of story points that a baseline task deserves as a first step. From then on, effort estimation is based on a comparison to that baseline. Story points are frequently assigned in a manner that deviates slightly from the Fibonacci sequence (i.e. 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞) [12]. This sequence reflects the uncertainty that comes with estimating complex tasks in real-world software.

In a project, story points will be evaluated by the entire team. The widespread planning poker method [13], for example, indicates that every team Member gives an estimate and after a few rounds of discussion and re-estimation a consensus estimate is reached. This practice is different in several respects from traditional approaches (e.g. points of function). Both story and function points reflect an effort to solve a problem. However, a standard set of rules (e.g., input count, outgoing information or inquiry count) which could be applied by any trained practitioner can be used to determine function points.

2.2 Deep Learning

In recent years, deep learning technology (DL) [14] has shown impressive results in a variety of fields, including machine vision [15], speech recognition [16], and text classification [17]. Most deep learning studies on text classification can be divided into two parts: (1) learning word vector representations through neural language models [14] and (2) performing classification composition over the learnt word vectors.

Convolutional neural networks (CNNs) [18] and recurrent neural networks (RNNs) [19] are two types of deep learning models used in text classification. Many text classification methods based on CNNs or RNNs have been proposed in recent years [20], [24].

CNNs are capable of learning the local response from temporal or spatial data, but not sequential correlations. RNNs, unlike CNNs, are designed for sequential modeling but are unable to extract features in a parallel manner. Text classification, in particular, can be thought of as a sequential modeling activity. RNNs are used more commonly in text classification because of their characteristics.

2.2.1 Long Short-Term Memory (LSTM)

RNNs are a form of feed forward neural network with a recurrent hidden state that is triggered at a specific time by the previous states. As a consequence, RNNs can dynamically model contextual information and manage variable-length sequences [25]. LSTM is a form of RNN architecture that has recently become the standard structure for RNNs. The LSTM solves the vanishing gradient problem by replacing self-connected hidden units with memory blocks. The memory block stores data in purpose-built memory cells and is better at detecting and leveraging long-range sequence.

The standard LSTM network can only use the historical background. However, the absence of a future contexts may cause the meaning of the problem to be incomplete. Through the combination of the forward hidden layer and a backward hidden layer, as shown in Figure 1, bidirectional Gated recurrent (BiLSTM) is therefore proposed to reach both the preceding and the future context. In a similar fashion to regular network, both forward and backward passes are done over time through the unfolded network; only BiLSTM must unfold the forward hidden state and the back-hidden state at all times [25].

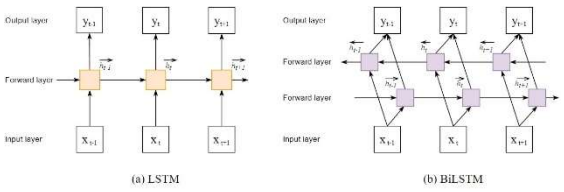


Figure 1. (a) LSTM model and (b) BiLSTM model.

2.2.2 Gated Recurrent Unit (GRU)

Gated Recurrent Unit is another form of RNN that overcomes the short-term memory problem [61]. Its internal gates, allow it to monitor the flow of information in a similar manner as the LSTM. Its composition is simpler than LSTM, with just the reset and upgrade gates and no memory cells. In deciding which data to keep or throw away, the update gate works similarly to the input and forget gates of an LSTM. The reset gate, on the other hand, determines which data from the past should be discarded.

GRU requires fewer operations to be estimated than LSTM due to its simplicity. As a result, GRU-based networks can achieve comparable performance to LSTM-based networks for the same task, while training is much faster.

The GRU tracks the state of sequences using a gating system rather than independent memory cells. The reset gate r_t and the update gate z_t are the two forms of gates. They are in charge of how knowledge is modified to the current state. The GRU computes the new state at time t as a vector which holds information for the current unit and passes it down to the network.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (1)$$

This is a linear interpolation using new sequence knowledge between the previous state h_{t-1} and the present new state $\tilde{h}_t$. The gate z_t determines how much old data is retained and how much new data is added. z_t has been changed to:

$$z_t = \sigma(W_z x_t + u_z h_{t-1} + b_z) \quad (2)$$

where x_t denotes the sequence vector at t. The candidate state $\tilde{h}_t$ is calculated in the same way as a typical recurrent neural network (RNN) is:

$$\tilde{h}_t = \tanh(W_h x_t + r_t \odot (u_z h_{t-1}) + b_h) \quad (3)$$

The reset gate r_t determines how much the previous state contributes to the candidate state. If r_t is empty, the previous state is forgotten. The following is an update to the reset gate:

$$r_t = \sigma(w_r x_t + u_r h_{t-1} + b_r) \quad (4)$$

Where b_z, b_h and b_r are the biases applied to the update, reset gates and hidden neurons.

2.3 Attention

This concept was first introduced in the context of computer vision. We can learn to amass information about a shape and identify the image by glancing at different areas of the image (glimpses), according to Larochelle and Hinton [65]. Later, the same idea was used to sequences. We can look at all of the words at once and learn to "pay attention" to the ones that are correct for the task at hand. This is

what we now refer to as attention, which is just a concept of memory gained by paying attention to a variety of inputs throughout time.

2.4 Hierarchical Attention Networks

Hierarchical attention networks are made up of a word sequence encoder, a word-level attention layer, a sentence encoder, and a sentence-level attention layer [55]. We'll go over the details of each component later in this paper.

2.5 Word Embedding

Words in the text must be converted into vectors in order to deal with various natural language problems and application. Word embedding are distributional vector representations of words that have been introduced to represent their syntax and semantics. The Word2vec [71] and GloVe [57] word embedding models have recently been used in a variety of natural language processing applications. Global Vectors for Word Representation GloVe was used for obtaining vector representation for each word. GloVe is an unsupervised learning algorithm for obtaining vector representations for words [72]. Every embedding approach has its own set of advantages and disadvantages. The GloVe embedding has the advantage of using the entire corpus statistics during word feature extraction, whereas Word2Vec simply looks at local window-based data. The GloVe takes into account word frequency both locally and globally, as well as word-to-word co-occurrences [79].

2.6 Transfer Learning

Transfer learning [73] is a fascinating machine learning method for transferring knowledge learnt from one problem to another different but related problem. Feature-based Transfer Learning is a set of deep network-based algorithms that allow feature extraction knowledge learnt from one problem to be reused in the solution of another [74], [75]. This eliminates the requirement for the network to learn how to extract features, resulting in less overall learning time.

3. RELATED WORKS

Software effort estimation methods can generally be categorized into three major groups: expert-based, model-based, and hybrid approaches. Expert-based methods rely upon human understanding to make estimations, and are the most common techniques in practice [26], [27]. Expert-based estimation requires the provision of experts on every occasion the estimation needs to be made. Model-based approaches use data from past projects but they are also varied in terms of building customized models. The well-known construction cost (COCOMO) model [28] is an example of a fixed model in which factors and variables are fixed. Their relationships are already established. Data from a variety of previous projects was used to build such estimation models. As a consequence, they are typically only sufficient for a particular type of projects that was used to create the model.

The customized model building approach requires context-specific data and employs a variety of methods, including regression (e.g. [29], [30]), neural networks (e.g. [31], [32]), fuzzy logic (e.g. [33]), Bayesian belief networks (e.g. [34]), analogy-based (e.g. [35], [36]), and multi-objective evolutionary approaches (e.g. [37]). However, no single method is likely to be the best performer for all project types [38], [39], [40]. As a result, some recent work proposes combining estimates from multiple estimators [41]. Hybrid approaches combine expert judgments with available data, [42], [43], which is similar to the ideas in this paper.

While most existing work is focused on estimating an entire project, little is done in building models for agile projects in particular. Agile, dynamic and incremental projects today require different planning and estimation approaches [44]. Recent approaches use machine learning techniques to help in estimating effort for agile projects. Recently, in [45] the work has suggested an approach that extracts TF-IDF features from the problem description in order to develop a model for story point estimations. The uniform selection technique is then applied to the features extracted and fed into classifiers such as SVM.

Furthermore, the research in [46] used Cosmic Function Points (CFP) [47] to estimate the effort needed to complete an agile project. In [48], the work developed an effort prediction model for the development of iterative software by regression models and neural networks. This model is built after each iteration (instead of at the end of the design phase) to estimate the effort for the next iteration, unlike traditional effort estimate models.

The authors in [49] created a Bayesian network model for predicting effort in software projects that use the agile Extreme Programming method. Their model, on the other hand, is based on a variety of criteria (such as process effectiveness and

improvement) that involve extensive learning and fine tuning. In order to identify problems in Scrum-based software development projects, Bayesian networks are often used, as in [50], to model dependencies between different factors (e.g., sprint progress and sprint planning quality affect product quality).

The work in [66] is a first step in understanding how to create a user story-based Agile effort prediction model. The datasets used in this project are small, and the developers created user stories in both Italian and English.

Moharreri et. Al [64] created an automated Agile story card estimating system that effectively combines existing machine learning algorithms to historical estimation data gathered from people. They aimed to find supervised learning models that might outperform manual Planning Poker. None of these techniques were able to outperform manual Planning Poker by a significant margin.

The work in [67] employs distributed word embedding to create a system capable of estimating effort based on only basic project management metrics and, most importantly, textual descriptions of tasks. To automate the effort estimation task, an artificial neural network was used.

Choetkiertikul in [56] focuses on estimating issues with story points by using deep learning techniques to automatically learn semantic features that reflect the true meaning of issue descriptions, which is a significant improvement over previous work. Previous research (as in [51], [52], [53] and [54]) has been done in forecasting the elapsed time for fixing a bug or the risk of resolving an issue with a pause.

The research of this paper is distinct from previous work in that it focuses specifically on estimating issues according to three story points' classes providing the majority distribution for the story point. The selection of the three classes is based on the application and is done experimentally. A story point estimate reflects the relative amount of effort involved in resolving or completing the user story. We are providing the team with a machine learning classifier to work with them in the estimation process. This classifier should give the team a guide line. The classifier should reduce the number of iterations done by the agile team to reach an agreement on the estimated issues. Hierarchical attention networks were used to learn semantic features that convey the true meaning of the issue descriptions automatically.

4. THE PROPOSED MODEL

The proposed model introduces the use of hierarchical attention networks (HAN) for classifying user stories according to a range of story points. These ranges are clustered into three classes. The model constructs a document vector incrementally by gathering important words into sentence vectors and then combining important sentence vectors to document vectors. The document vectors are then fed into a shallow neural network for classification.

The main contributions of this paper are:

1. Proposing a new balanced data set that can be used for classification with sufficient data for training deep learning network.
2. Turning the estimation problem from a regression into a classification problem, where each class has a range of values. Since effort estimation process is based on uncertainty, having a range of values for effort estimation will provide the software team with a more flexible way than previous work where estimation was done according to a fixed value.
3. Reducing the training time by using pre-trained embedding vectors by exploiting transfer learning using GloVe, instead of building a new embedding vector.
4. Implementing hierarchical attention network [55] to capture two fundamental insights about document structure. First, since documents are hierarchical (words form sentences, sentences form a text), we create a document representation by first constructing sentence representations and then aggregating them into a document representation. Second, various terms and sentences in a text are found to have different levels of knowledge.

The proposed model, is made up of five layers, as shown in Figure 2, and is explained briefly as follows:

1. **Input layer**: takes a document, consisting of sentences, where each sentence consisting of a sequence of word IDs that represent user stories or issues describing what has to be built in the software project. Assume that a document includes L sentences s_i and each sentence contains T_i words. w_{it} with $t \in [1, T]$ represents the words in the i^{th} sentence.

Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2. **Embedding layer**: embeds each word in each sentence separately, to produce Sequences of word vectors, one for each sentence. That is, it converts incoming text into dense word vectors that encode its meaning as well as its context. Global Vectors for Word Representation GloVe was used for obtaining vector representation for each word.

3. **Encoding layer**: from preceding layer we have a sequence of word vectors, the aim of this layer is to compute a sentence matrix from which we can build document matrix. Sentence matrix is composed of rows each row represents the meaning of each Model Configuration in the sentence. This layer is implemented using Bidirectional RNN. Each token's vector is split into two sections, one computed using a forward pass and the other using a backward pass. We simply combine the two to get the complete vector. This layer contains two encoders which are:
   - Sentence Encoder: converts sequence of word vectors to sentence matrix.
   - Document Encoder: converts sequence of sentence vectors to document matrix.

Given a sentence with words w_{it}, $t \in [0, T]$, the Sentence Encoder embeds the words to vectors through an embedding matrix w_e, such that $x_{ij} = w_e w_{ij}$.

Bidirectional GRU is used to get words annotations by summarizing words information from both ways, so it adds contextual information into the annotation. The bidirectional GRU contains the forward GRU $\vec{f}$ which reads the sentence s_i from w_{i1} to w_{iT} and a backward GRU $\overleftarrow{f}$ which reads from w_{iT} to w_{i1}. Eq. (5) to Eq. (7) describe the encoding process.

$$X_{it} = W_e w_{it}, t \in [1, T]. \quad (5)$$

$$\overrightarrow{h_{it}} = \overrightarrow{GRU}(X_{it}), t \in [1, T]. \quad (6)$$

$$\overleftarrow{h_{it}} = \overleftarrow{GRU}(X_{it}), t \in [T, 1]. \quad (7)$$

Document Encoder works in a similar manner, given the sentence vectors s_i, it produces a document vector. To encode the sentences, we use a bidirectional GRU with:

$$\overrightarrow{h_i} = \overrightarrow{GRU}(s_i), i \in [1, L]. \quad (8)$$

$$\overleftarrow{h_i} = \overleftarrow{GRU}(s_i), i \in [L, 1]. \quad (9)$$

4. **Attention layer**: in this layer our objective is to reduce the Sentence matrix obtained from pervious layer to a single vector that can be used by feed-forward network for prediction. The role of this layer is to find words that are most important semantically for a user story.

5. **Output layer**: predict the class of story point that match the given user stories. This layer was implemented using shallow neural network with softmax activation [80] for classification.

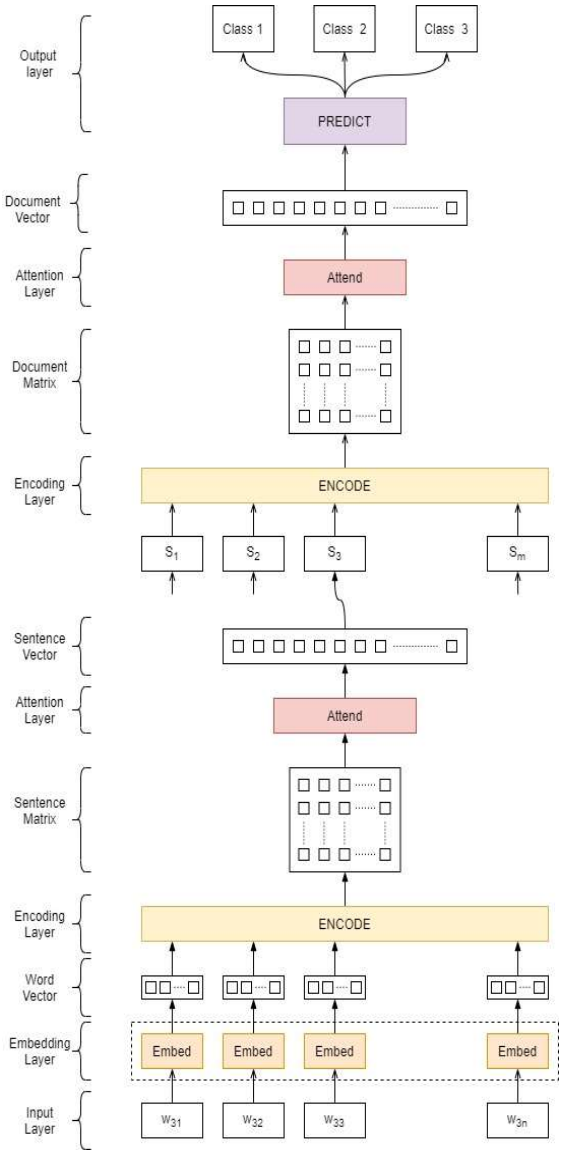


Figure 2. Proposed model

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Dataset

Our data set consists of 23,313 issues, with the story points of 16 various projects [56]: Apache Mesos (ME), Apache Usergrid (UG), Appcelerator Studio (AS), Aptana Studio (AP), Titanum SDK/CLI (TI), DuraCloud (DC), Bamboo (BB), Clover (CV), JIRA Software (JI), Moodle (MD), Data Management (DM), Mule (MU), Mule Studio (MS), Spring XD (XD), Talend Data Quality (TD), and Talend ESB (TE).

The story points used in planning poker are usually arranged in a Fibonacci sequence, such as 1, 2, 3, 5, 8, 13, 21, and so on [24]. Only seven of the projects we examined (Usergrid, Talend ESB, Talend Data Quality, Mule Studio, Mule, Appcelerator Studio, and Aptana Studio) followed the Fibonacci scale, while the other nine projects did not use any scale. We have selected these seven projects to make a subset from the dataset, we were able to merge the seven projects to form a new dataset with 7,459 issues, and our intention is to test the proposed model on this dataset.

After analyzing the new dataset by assuming that every story point represents a class we faced the imbalanced data problem, i.e., several classes are under-represented (minority classes) in comparison to others (majority classes). To overcome this problem, the new dataset was clustered into 3 classes.

- First class (Low) ranging from 1-8 story point.
- Second class (Medium) ranging from 8-15 story point.
- Third class (High) ranging from 15 -21 story point.

For further enhancement the three new classes were balanced. There are several techniques to obtain balanced data, but oversampling [63] was used in this paper to duplicate the data samples in the minority classes.

After applying the oversampling process, the data set are used such that 70% of the data set for training, 15% for validation, and the remaining 15% for testing.

5.2 Model configuration

We used NLTK [30] to break documents into sentences and tokenize each sentence. Before constructing the vocabulary, several pre-processing steps has to be applied. These steps involve removing punctuations, prepositions, stop words, tags, html, removing numbers, extra spaces and URLs, lower casing and stemming.

A dense vector that represents a word is referred to as an embedding. During the training process, the embedding vectors are randomly initialized, then progressively improved with the gradient descent algorithm at each back-propagation stage, so that similar words or words in the same lexical field end up near in terms of distance in the new vector space.

Pre-trained word embedding is a Transfer Learning example. The key idea is to use public embedding, which has already been trained on large datasets. In particular, we set these previously trained integrations as the initialization weights, instead of initializing our neural network weights at random. This technique helps to speed up training and improve NLP model performance.

GloVe [57] is the most popular methods for inducing word embedding from the text corpora. It provides embedding off the shelf trained on huge text corpora. The GloVe package provides embedding with varying sizes. The three hundred embedding size was used for this experiment.

Overfitting occurs when your model fits well on training data but does not generalize well on new, unseen data. In other words, the model learned patterns that are unique to the training data and are irrelevant to other data. Validation metrics such as loss and accuracy can be used to detect overfitting. After a certain number of epochs, the validation metric typically stops improving and starts to decline. Since the model aims to find the best match for the training data, the training metric continues to improve.

There are several methods for reducing overfitting in deep learning models. Obtaining more training data is the best option. Unfortunately, this is not always possible in real-world situations.

We have chosen Dropout proposed by Srivastava [62] to handle the proposed model. Dropout is a training technique in which randomly selected neurons are ignored. They are dropped out at random. This means that on the forward pass, their contribution to the activation of downstream neurons is eliminated temporally, and on the backward pass, any weight changes are not added to the neuron. As a consequence, the network's sensitivity to individual

neuron weights decreases. As a result, the network will be able to generalize better and will be less likely to overfit the training data.

The implementation was done on Intel (i7-7700, 16 GB RAM) windows 10. Keras [29], a Python library, is used to implement the proposed model.

Our experiments were done by testing three scenarios. The first was by removing the attention layer then two variations for the attention was tested. Tables 1,2,3, and 4 show the achieved results. The second scenario is using the first type of attention with eq. (10) to (12).

We employ the attention mechanism to reward sentences that provide clues to correctly classify a document, and we create a sentence level context vector e_t, which we use to measure the relevance of the sentences and get a normalized importance weight α_t through a softmax function. This yields:

$$e_t = tanh(Wh_t + b). \quad (10)$$

$$\alpha_t = softmax(e_t). \quad (11)$$

$$o = \sum \alpha_t h_t. \quad (12)$$

where o is the document vector that summarizes all the information of sentences in a document. W and b are Learnable weights, and h_t the output from GRU.

The third scenario is using Second type with the following equations [59].

$$e_t = tanh(Uc + Wh_t + b). \quad (13)$$

$$\alpha_t = softmax(e_t). \quad (14)$$

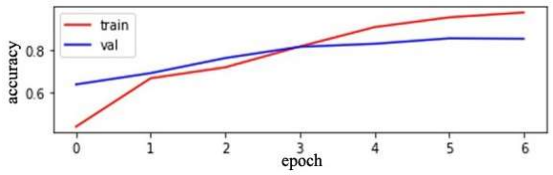
$$o = \sum \alpha_t h_t. \quad (15)$$

Where c is the vector obtained by applying max poling to the matrix obtained from GRU, and U is a new weight vector differs from pervious type context learned vector c.

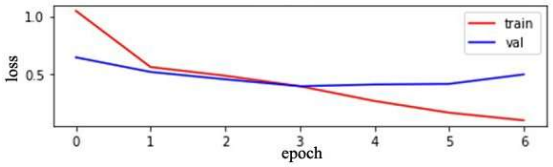
5.3 Analysis

In the first phase the objective is training the model to be generalized well and to avoid overfitting. The number of training epochs to use is a concern when it comes to neural network training. Too many epochs may cause overfitting, whereas too few may cause underfitting. To handle this problem, the Keras [29] API was used for adding early stopping. Early stopping is a technique that allows you to specify an arbitrary large number of training epochs and then stop training when the model performance on a holdout validation dataset stops improving. The first sign of no further improvement may not be the best time to stop training. The model may get better or worse after this point. This problem was handled by adding a delay to the trigger equal to the number of epochs we should wait to see no improvement. This can be accomplished by using the "patience" provided by Keras.

The learning curves in Figure 3 can be summarized as X-axis shows the training time, the Y-axis shows the accuracy and the loss. The red line presents the training set and the blue the validation set. The training curve continues to get better till it reaches the best value. The behavior of the validation is not the same; the validation keeps on improving until it reaches a point of intersection with the training curve after this point overfitting will occur and the curve is getting worse. The gap between training and validation accuracy is a clear indication of over fitting. The larger the gap, the higher the overfitting. At this point, an early stopping should occur additionally patience option present in keras gives the model additional number of epochs to make sure no improvement may take place.



(a) Model accuracy



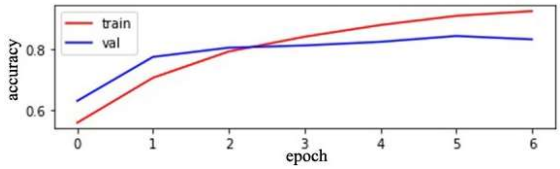
(b) Model loss

Figure 3. Learning curve for model with no attention (a) model accuracy, (b) model loss

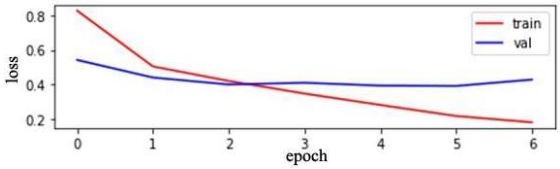
The model behavior during training phase when adding attention layer using first type of attention is shown in Figure 4 that the point of intersection between training and validation curve is reached earlier than the previous cases.

Journal of Theoretical and Applied Information Technology
30th September 2022. Vol.100. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195



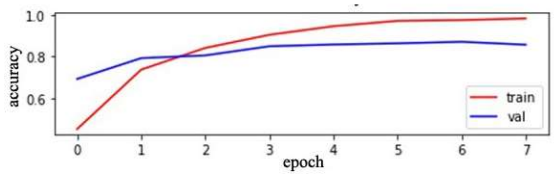
(a) Model accuracy



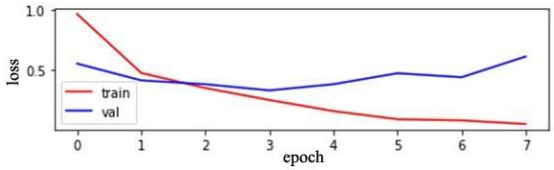
(b) Model loss

Figure 4. learning curve for model with first type attention (a) model accuracy, (b) model loss

The model behavior during training phase when adding attention layer using second type of attention is shown in Figure 5. The figure shows that the point of intersection between training and validation curves is reached earlier than the previous cases this may answer RQ1 thus reducing the training time.



(a) Model accuracy



(b) Model loss

Figure 5. learning curve for model with second type attention (a) model accuracy, (b) model loss

At the second phase we will be evaluating the proposed model which is done by comparing the overall accuracy F score, Precision and recall.

Accuracy is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

Both the preceding and subsequent contextual information can be accessed by BiGRU. As a result, BiGRU can better understand the context of each word in the text. The purpose of the attention mechanism is to determine the impact of each word on the phrase. It can capture the main semantic components of a sentence by assigning attention weights to each word. The combination of these methods improves the accuracy of sentence semantic understanding and the classification ability of the proposed model. Thus, leading to better results in the overall accuracy. Experimental results presented in Table 1, Table 2, and Table 3 show that the attention mechanism has a significant impact on the proposed model's performance. Adding the context vector has added extra information that leads to better performance.

As a result, we may say that the attention mechanism may be used to precisely figure out the differences in relevance in a document, which results in upgrading the classification accuracy by acquiring more effective information. The accuracy has improved when adding the attention layer and when modifying the attention layer by adding the context vector has added extra information and leads to better performance and this part addresses RQ2.

Scores for the third class shown in Table 2, and Table 3, are relatively high because this class is the minority class and it contains some redundant data caused by oversampling the first and the second classes (majority classes).

The F-score is a method of combining the model's precision and recall, and it is defined as the harmonic mean of the precision and recall of the model. To calculate the F Score, you need to know the Precision and Recall scores and input them into the following formula:

$$F\ Score=2*((Precision*Recall)/(Precision+Recall). \quad (16)$$

So, we can focus on f score.

Table 1. Classification Accuracy while using different attention variation

Model with no attention	Model with first type attention	Model with second type attention
0.81	0.83	0.87

Table 2. F score while using different attention variation.

Classes	**F score**		
	Model with no attention	Model with first type attention	Model with second type attention
Class 1	0.69	0.73	**0.79**
Class 2	0.76	0.77	**0.82**
Class 3	0.99	0.99	**0.99**

Table 3. Precision while using different attention variation.

Classes	Precision		
	Model with no attention	Model with first type attention	Model with second type attention
Class 1	0.82	0.79	**0.86**
Class 2	0.68	0.72	**0.77**
Class 3	0.99	0.99	**0.98**

Table 4. Recall while using different attention variation.

Classes	Recall		
	Model with no attention	Model with first type attention	Model with second type attention
Class 1	0.60	0.68	**0.72**
Class 2	0.87	0.82	**0.88**
Class 3	0.99	0.99	**1.0**

To the best of our knowledge, [64] is the only approach testing the possibility of enhancing planning poker estimation by finding supervised learning models that might outperform manual Planning Poker. Due to the unavailability of the data used in [64], and in general, the scarcity of publicly available data to study and train on, an experiment was conducted to test the models used in [64] using our dataset. The experiment uses TF-IDF (Term Frequency – Inverse Document Frequency) of a word as a metric to quantify the significance of a word which takes into account word frequency and the inverse of the counts of documents containing that word.

Our result cannot be compared to other previous works as they have worked on a regression problem and they have their evaluation matrix.

Tables 5, 6 and 7, show comparison between the proposed model with 2nd type attention against classical classifiers.

*Table 5. Comparison between the proposed Model with 2nd type attention against classical classifiers (**F score**).*

Classes	F score				
	Naive Bayes	Support Vector Machine	Decision Tree	Logistic Regression	Model with 2nd type attention
Class 1	0.66	0.75	0.77	0.76	**0.79**
Class 2	0.71	0.76	0.81	0.76	**0.82**
Class 3	0.92	0.99	0.99	0.99	**0.99**

*Table 6. Comparison between the proposed Model with 2nd type attention against classical classifiers (**Precision**).*

Classes	Precision				
	Naive Bayes	Support Vector Machine	Decision Tree	Logistic Regression	Model with 2nd type attention
Class 1	0.80	0.78	0.86	0.79	**0.86**
Class 2	0.66	0.74	0.75	0.74	**0.77**
Class 3	0.86	0.98	0.97	0.97	**0.98**

*Table 7. Comparison between the proposed Model with 2nd type attention against classical classifiers (**Recall**).*

Classes	Recall				
	Naive Bayes	Support Vector Machine	Decision Tree	Logistic Regression	Model with 2nd type attention
Class 1	0.56	**0.72**	0.66	0.71	**0.72**
Class 2	0.76	0.78	0.88	0.77	**0.88**
Class 3	0.99	1.0	1.0	1.0	**1.0**

The second type of attention has better or at least the same F score, Precision and Recall when compared to classical classifiers like Naive Bayes, Support Vector Machine, Decision Tree, and Logistic Regression as shown in Tables 5, 6, and 7, and this would answer the last research question QR3. Obviously produced the highest results in terms of classification accuracy, F score, precision, and memory.

5.4 Threats to Validity

By using real-world data from issues reported in large open source projects, we attempted to minimize challenges to build validity. The title and description provided with these issue reports, as well as the actual story points assigned to them, were gathered. We are aware that those story points were calculated by human teams, and as a result, they may contain biases and, in some situations, may be inaccurate. We should acknowledge, however, that our dataset may not reflect all types of software initiatives, particularly in commercial environments (although open source projects and commercial projects are similar in many aspects). The nature of contributors, developers, and project stakeholders is one of the fundamental variations between open source and commercial projects that may affect story point estimation. For commercial agile projects, more research is required. Datasets of various sizes were used in this study. In order to reduce conclusion instability [83], we carefully followed recent best practices in evaluating effort estimation models [81], [82].

5.5 Implications

The fast emergence of agile development demand more research on estimation at the issue or user story level. To our knowledge, work on effort estimation mainly focus on estimating the whole project with a small number of data points (see the datasets in the PROMISE repository [76]). The China dataset, for example, comprises only 499 data points, whereas Desharnais has 77 and Finish has 38. All of these data sets are extensively utilized in existing effort estimation studies [77], [78]. On the other hand, in this work, we're dealing with 7,459 data points actually more the data, the more robust the network will be.

On other side, this paper uses hierarchical attention networks (HAN) to automatically learn a suitable representation of a problem issues and use them to estimate the effort required to resolve it. The evaluation findings show that (HAN) approach has significantly improved predictive performance. This is an important result since it encourages software developers to abandon the manual feature engineering method. Feature engineering is typically done by domain specialists who use their in-depth understanding of the data to build features that machine learners can use. Features are automatically learned from a textual description of an issue in our method.

The goal isn't for the machine learner to take the place of existing agile estimate methodologies. Instead, the goal is to use the machine learner to support existing methods by acting as a decision support system. Teams would still meet, discuss user stories, and develop estimates as usual, but with the extra benefit of having access to the machine learner's insights. As with any decision support system, teams would be free to reject the machine learner's recommendations. The actual estimates created in every such estimating exercise are recorded as data to be supplied to the machine learner, regardless of whether these estimates are based on the machine learner's suggestions or not. This estimation approach assists the team in not just understanding enough facts about what it will take to address such challenges, but also in aligning with previous estimates.

6. FUTURE WORK

The future work will involve comparing the results of our model to other Pre-trained language model like BERT [69], GPT [70] and XLNet [68]. These models have been shown to attain the state of the art in a range of tasks such as question answering, named entity recognition, and natural language inference. Also, it is planned to test the proposed model on other agile data sets.

7. CONCLUSION

The proposed model implemented the hierarchical attention networks (HAN) for classifying user stories according to a range of story points. These ranges should provide a guide line for the agile team in their estimation process. Three scenarios were tested as part of the experiments. The first was to remove the attention layer, after which two attention variations were tested. The second type of attention has achieved the best result for the Classification Accuracy, F score, Precision & and recall. The second type of attention has obviously produced the highest results in terms of classification accuracy, F score, precision, and memory. When compared to classical classifiers such as Naive Bayes, Support Vector Machine, Decision Tree, and Logistic Regression, the second type of attention has a superior or at least equal F score, Precision, and Recall.

The novelty of our work is providing a range for estimation instead of a unique value, make use of transfer learning to reduce training time and introducing a new balanced data set that can be used for classification and contains enough data to train a deep learning network. Our experiment has shown that attention layer has improved the overall accuracy and F score and the second type of attention has achieved the best result.

REFERENCES:

[1] K. Beck, M. Beedle, A. Va Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Je_ries, et al. Manifesto for agile software devel-opment, 2001.

[2] L. C. Briand. On the many ways software engineering can bene_t from knowledge engineering. In Proceedings of the 2002 international conference on Software engi-neering and knowledge engineering, pages 3{6. ACM, 2002.

[3] J. W. Paulson, G. Succi, and A. Eberlein. An empirical study of open-source and closed-source software prod-ucts. IEEE Transactions on Software Engineering, 30(4):246{256, 2004.

[4] What is a story point? https://goo.gl/Vfb9v1, 2007. Accessed: 2016-06-02.

[5] M. Cohn. It's e_ort, not complexity https://goo.gl/nNYlL1, 2010. Accessed: 2016-06-02.

[6] J. Grenning. Planning poker or how to avoid analysis paralysis while release planning. Hawthorn Woods: Re-naissance Software Consulting, 3, 2002.

[7] R. Brown. Group Processes: Dynamics Within and Be-tween Groups. Wiley-Blackwell, 2000

[8] E. Aronson, T. D. Wilson, and R. M. Akert. Social Psy-chology (3rd Edition). Pearson, 1999.

[9] R. L. Atkinson, R. C. Atkinson, E. E. Smith, D. J. Bem, and S. Nolen-Hoeksema. Hilgard's Introduction to Psychology (12th Edition). Orlando: Harcourt

[10] J. Aranda and S. Easterbrook. Anchoring and adjust-ment in software estimation. In Proceedings ofthe 2005 European Software Engineering Conference Held Jointly with the 2005 International Symposium on Foundations of Software Engineering, pages 346{355. ACM, 2005.

[11] Planning poker 3.0. https://goo.gl/Tl5mws. Accessed: 2016-06-16.

[12] Why do the cards deviate slightly from the _bonacci sequence? http://goo.gl/xgs7PF. Accessed:2016-07-13.

[13] M. Usman, E. Mendes, F. Weidt, and R. Britto. E_ort estimation in agile software development: Asystematic literature review. In Proceedings of the 2014 Internation-al Conference on Predictive Models in Software Engi-neering, pages 82{91. ACM, 2014.

[14] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (January 01, 2015) (2015) 85–117.

[15] V. Campos, B. Jou, X. Giro-i Nieto, From pixels to sentiment: Fine-tuning cnns for visual sentiment predic-tion, Image Vis. Comput. 65 (September 2017) (2017) 15–22.

[16] L. Brocki, K. Marasek, Deep belief neural networks and bidirectional long-short term memory hybrid for speech recognition, Arch. Acoust. 40 (2) (2015) 191–195.

[17] K.S. Tai, R. Socher, C.D. Manning, Improved seman-tic representations from tree-structured long short-term memory networks, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguis-tics and the 7th International Joint Conference on Natu-ral Language Processing of the Asian Federation of Natu-ral Language Processing ACL-IJCNLP 2015, Association for Computational Linguistics (ACL), Beijing, China, 2015, pp. 1556–1566.

[18] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep con-volutional neural networks, in: Proceedings of the Advances in Neural Infor- mation Processing Systems 25: 26th Annual Conference on Neu-ral Informa- tion Processing Systems 2012 (NIPS'2012), Lake Tahoe, NV, United states, 2012, pp. 1097–1105.

[19] K.-i. Funahashi, Y. Nakamura, Approximation of dynamical systems by contin- uous time recurrent neural networks, Neural Netw. 6 (6) (1993) 801–806.

[20] Y. Kim, Convolutional neural networks for sentence classification, in: Pro- ceedings of the Conference on Empirical Methods in Natural Language Pro- cessing, As-sociation for Computational Linguistics (ACL), Doha, Qatar, 2014, pp. 1746–1751.

[21] S. Liao, J. Wang, R. Yu, K. Sato, Z. Cheng, CNN for situations understanding based on sentiment analysis of twitter data, in: Proceedings of the 8th In- ternational Conference on Advances in Information Technology, Elsevier B.V., Macau, China, 2016, pp. 376–381.

[22] W. Cao, A. Song, J. Hu, Stacked residual recurrent neural network with word weight for text classification, IAENG Int. J. Comput. Sci. 44 (3) (2017) 277–284.

[23] Y. Zhang, M.J. Er, R. Venkatesan, N. Wang, M. Pratama, Sentiment classification using comprehensive attention recurrent models, in: Proceedings of the International Joint Conference on Neural Networks, IEEE, Vancouver, BC, Canada, 2016, pp. 1562–1569.

[24] L. Wang, Z. Wang, S. Liu, An effective multivariate time series classification ap- proach using echo state network and adaptive differential evolution algorithm, Expert Syst. Appl. 43 (January 1, 2016) (2016) 237–249.

[25] Liu G, Guo J. Bidirectional LSTM with attention mechanism and convolutional layer for text classifica-tion. Neurocomputing. 2019 Apr 14;337:325-38. new

[26] M. Jorgensen, "A review of studies on expert estima-tion of software development effort," J. Syst. Softw., vol. 70, no. 1/2, pp. 37–60, 2004.

[27] M. Jorgensen and T. M. Gruschke, "The impact of lessons-learned sessions on effort estimation and uncer-tainty assessments," IEEE Trans. Softw. Eng., vol. 35, no. 3, pp. 368–383, May/Jun. 2009.

[28] B. W. Boehm, R. Madachy, and B. Steece, Software Cost Estimation with Cocomo II. Englewood Cliffs, NJ, USA: Prentice Hall, 2000.

[29] P. Sentas, L. Angelis, and I. Stamelos, "Multinomial logistic regression applied on software productivity pre-diction," in Proc. 9th Panhellenic Conf. Informat., 2003, pp. 1–12.

[30] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Soft-ware productivity and effort prediction with ordinal re-gression," Inf. Softw. Technol., vol. 47, no. 1, pp. 17–29, 2005.

[31] S. Kanmani, J. Kathiravan, S. S. Kumar, M. Shanmu-gam, and P. E. College, "Neural network based effort es-timation using class points for OO systems," in Proc. Int. Conf. Comput.: Theory Appl. (ICCTA), 2007, pp. 261–266.

[32] A. Panda, S. M. Satapathy, and S. K. Rath, "Empirical validation of neural network models for agile software effort estimation based on story points," Procedia Com-put. Sci., vol. 57, pp. 772–781, 2015.

[33] S. Kanmani, J. Kathiravan, S. S. Kumar, and M. Shanmugam, "Class point based effort estimation of OO systems using fuzzy subtractive clustering and artificial neural networks," in Proc. 1st India Softw. Eng. Conf., 2008, pp. 141–142.

[34] S. Bibi, I. Stamelos, and L. Angelis, "Software cost prediction with predefined interval estimates," in Proc. 1st Softw. Meas. Eur. Forum, 2004, pp. 237–246.

[35] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," IEEE Trans. Softw. Eng., vol. 23, no. 11, pp. 736–743, Nov. 1997.

[36] L. Angelis and I. Stamelos, "A simulation tool for efficient analogy based cost estimation," Empirical Softw. Eng., vol. 5, no. 1, pp. 35–68, 2000.

[37] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in Proc. 38th Int. Conf. Softw. Eng., 2016, pp. 619–630.

[38] M. Jørgensen and M. Shepperd, "A systematic review of software development cost estimation studies," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 33–53, Jan. 2007.

[39] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 1403–1416, Nov./Dec. 2012.

[40] F. Collopy, "Difficulty and complexity as factors in software effort estimation," Int. J. Forecasting, vol. 23, no. 3, pp. 469–471, 2007.

[41] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," IEEE Trans. Softw. Eng., vol. 38, no. 6, pp. 1403–1416, Nov./Dec. 2012.

[42] R. Valerdi, "Convergence of expert opinion via the wideband delphi method: An application in cost estima-tion models," INCOSE Int. Symp., vol. 21, no. 1, pp. 1246–1259, 2011.

[43] S. Chulani, B. Boehm, and B. Steece, "Bayesian anal-ysis of empirical software engineering cost models," IEEE Trans. Softw. Eng., vol. 25, no. 4, pp. 573–583, Jul./Aug. 1999.

[44] M. Cohn, Agile Estimating and Planning. London, U.K.: Pearson Education, 2005.

[45] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli, "Estimating story points from issue reports," Proc. 12th Int. Conf. Predictive Models Data Anal. Softw. Eng., 2016, Art. no. 2.

[46] R. Djouab, C. Commeyne, and A. Abran, "Effort estimation with story points and COSMIC function points - an industry case study," pp. 25–36, 2008. [Online]. Available: http://cosmicsizing.org/wp-content/uploads/2016/03/Estimation-model-v-Print-Format-adapter.pdf

[47] ISO/IEC JTC 1/SC 7, INTERNATIONAL STANDARD ISO/IEC Software Engineering COSMIC: A Functional Size Measurement Method, vol. 2011, 2011. [Online]. Available: https://www.iso. org/standard/54849.html

[48] P. Abrahamsson, R. Moser, W. Pedrycz, A. Sillitti, and G. Succi, "Effort prediction in iterative software devel-opment processes—incremental versus global prediction models," in Proc. 1st Int.Symp. Empirical Softw. Eng. Meas., 2007, pp. 344–353.

[49] P. Hearty, N. Fenton, D. Marquez, and M. Neil, "Pre-dicting project velocity in XP using a learning dynamic Bayesian network model," IEEE Trans. Softw. Eng., vol. 35, no. 1, pp. 124–137, Jan./Feb. 2009.

[50] M. Perkusich, H. De Almeida, and A. Perkusich, "A model to detect problems on scrum-based software de-velopment projects," in Proc. ACM Symp. Appl. Com-put., 2013, pp. 1037–1042.

[51] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in Proc. 2nd Int. Workshop Recommenda-tion Syst. Softw. Eng., 2010, pp. 52–56.

[52] L. D. Panjer, "Predicting eclipse bug lifetimes," in Proc. 4th Int. Workshop Mining Softw. Repositories, 2007, pp. 29–32.

[53] P. Bhattacharya and I. Neamtiu, "Bug-fix time predic-tion models: Can we do better?" in Proc. 8th Working Conf. Mining Softw. Repositories, 2011, pp. 207–210.

[54] P. Hooimeijer and W. Weimer, "Modeling bug report quality," in Proc. 22 IEEE/ACM Int. Conf. Automated Softw. Eng., Nov. 2007, pp. 34–44.

[55] Yang Z, Yang D, Dyer C, He X, Smola A, Hovy E. Hierarchical attention networks for document classifica-tion. InProceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies 2016 Jun (pp. 1480-1489).

[56] Choetkiertikul M, Dam HK, Tran T, Pham T, Ghose A, Menzies T. A deep learning model for estimating story points. IEEE Transactions on Software Engineering. 2018 Jan 12;45(7):637-56.

[57] Pennington J, Socher R, Manning CD. Glove: Global vectors for word representation. InProceedings of the 2014 conference on empirical methods in natural lan-guage processing (EMNLP) 2014 Oct (pp. 1532-1543).

[58] Raffel C, Ellis DP. Feed-forward networks with atten-tion can solve some long-term memory problems. arXiv preprint arXiv:1512.08756. 2015 Dec 29.

[59] Cho K, Courville A, Bengio Y. Describing multimedia content using attention-based encoder-decoder networks. IEEE Transactions on Multimedia. 2015 Sep 4;17(11):1875-86.

[60] Bahdanau D, Cho K, Bengio Y. Neural machine trans-lation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473. 2014 Sep 1.

[61] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase rep-resentations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078. 2014 Jun 3.

[62] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neu-ral networks from overfitting. The journal of machine learning research. 2014 Jan 1;15(1):1929-58.

[63] Brownlee, Jason. Imbalanced classification with Py-thon: better metrics, balance skewed classes, cost-sensitive learning. Machine Learning Mastery, 2020.

[64] Moharreri K, Sapre AV, Ramanathan J, Ramnath R. Cost-effective supervised learning models for software effort estimation in agile environments. In2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC) 2016 Jun 10 (Vol. 2, pp. 135-140). IEEE.

[65] Larochelle H., Hinton G, (2010), Learning to combine foveal glimpses with a third-order Boltzmann machine

[66] Abrahamsson P, Fronza I, Moser R, Vlasenko J, Pedrycz W. Predicting development effort from user stories. In2011 International Symposium on Empirical Software Engineering and Measurement 2011 Sep 22 (pp. 400-403). IEEE.

[67] Ionescu VS. An approach to software development effort estimation using machine learning. In2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP) 2017 Sep 7 (pp. 197-203). IEEE.

[68] Yang Z, Dai Z, Yang Y, Carbonell J, Salakhutdinov R, Le QV X. Generalized autoregressive pretraining for lan-guage understanding. 2019.

[69] Devlin J, Chang MW, Lee K, Toutanova K. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805. 2018 Oct 11.

[70] Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[71] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Effi-cient Estimation of Word Representations in Vector Space," arXiv:1301.3781 [cs], Jan. 2013.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[72] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1532–1543

[73] S. J. Pan and Q. Yang, "A survey on transfer learning," IEEE Transactions on knowledge and data engineering, vol. 22, no. 10, pp. 1345–1359, 2010.

[74] B. Chikhaoui, F. Gouineau, and M. Sotir, "A cnn based transfer learning model for automatic activity recognition from accelerometer sensors," in International Conference on Machine Learning and Data Mining in Pattern Recogni-tion. Springer,2018, pp. 302–315.

[75] M. A. A. H. Khan and N. Roy, "Untran: Recognizing unseen activities with unlabeled data using transfer learn-ing," in 2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI). IEEE, 2018, pp. 37–47.

[76] T. Menzies, et al., "The PROMISE Repository of empirical software engineering data," North Carolina State University, Department of Computer Science, 2015, [Online]. A

[77] P. L. Braga, A. L. I. Oliveira, and S. R. L. Meira, "Software effort estimation using machine learning techniques with robust confidence intervals," in Proc. 7th Int. Conf. Hybrid Intell. Syst., 2007, pp. 352–357. vailable: http://openscience.us/repo

[78] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," in Proc. 38th Int. Conf. Softw. Eng., 2016, pp. 619–630.

[79] Hossain, Md Rajib, Mohammed Moshiul Hoque, Nazmul Siddique, and Iqbal H. Sarker. "Bengali text document categorization based on very deep convo-lution neural network." Expert Systems with Applica-tions 184 (2021): 115394.

[80] Dunne, R.A. and Campbell, N.A., 1997, June. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In Proc. 8th Aust. Conf. on the Neural Networks, Melbourne (Vol. 181, p. 185). Citeseer

[81] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," Inf. Softw. Technol., vol. 54, no. 8, pp. 820–827, 2012. [Online]. Available: http://dx.doi.org/ 10.1016/j.infsof.2011.12.008

[82] P. A. Whigham, C. A. Owen, and S. G. Macdonell, "A baseline model for software effort estimation," ACM Trans. Softw. Eng. Methodology, vol. 24, no. 3, 2015, Art. no. 20.

[83] T. Menzies and M. Shepperd, "Special issue on repeatable results in software engineering prediction," Empirical Softw. Eng., vol. 17, no. 1/2, pp. 1–17, 2012.