Prediction of students’ academic performance using ANN with mini-batch gradient descent and Levenberg-Marquardt optimization algorithms

F R J Simanungkalit, H Hanifah, G Ardaneswari, N Hariadi and B D Handari

1Department of Mathematics, Faculty of Mathematics and Natural Sciences (FMIPA), Universitas Indonesia, Depok 16424, Indonesia

bevina@sci.ui.ac.id

Abstract. Online learning indirectly increases stress, thereby reducing social interaction among students and leading to physical and mental fatigue, which in turn reduced students’ academic performance. Therefore, the prediction of academic performance is required sooner to identify at-risk students with declining performance. In this paper, we use artificial neural networks (ANN) to predict this performance. ANNs with two optimization algorithms, mini-batch gradient descent and Levenberg-Marquardt, are implemented on students’ learning activity data in course X, which is recorded on LMS UI. Data contains 232 students and consists of two periods: the first month and second month of study. Before ANNs are implemented, both normalization and usage of ADASYN are conducted. The results of ANN implementation using two optimization algorithms within 10 trials each are compared based on the average accuracy, sensitivity, and specificity values. We then determine the best period to predict unsuccessful students correctly. The results show that both algorithms give better predictions over two months instead of one. ANN with mini-batch gradient descent has an average sensitivity of 78%; the corresponding values for ANN with Levenberg-Marquardt are 75%. Therefore, ANN with mini-batch gradient descent as its optimization algorithm is more suitable for predicting students that have potential to fail.

1. Introduction

World Health Organization (WHO) stated Coronavirus Disease 2019 (COVID-19) as a global pandemic on March 11st, 2020. Since the first case, COVID-19 positive cases around the world arise, including Indonesia. Indonesia government has been shifting learning activity from face-to-face learning to online learning to control rising COVID-19 cases in Indonesia. Online learning indirectly increased stress, thereby reducing social interaction among students and leading to physical and mental fatigue, which in turn reduced student’s academic performance [1]. Therefore, early prediction of students’ academic performance is required in identifying at-risk students regarding declining performance [2].

Artificial Neural Network (ANN) is formed through a collection of artificial neurons like biological neurons in human brain to process information with improved performance through learning data [3]. ANN can identify complex correlation, nonlinear, and unknown, between input and output data [4]. ANN has proven a success in the application of classification, forecasting, and predicting in the areas...
of healthcare [5], stock price market [6], etc. Furthermore, ANN can also be used for academic performance prediction [3] [7][8].

In general, an ANN consists of layers of neurons, where neurons in each layer are connected using a connection that contain adjustable weights during training process [3]. Weights are updated to minimize error using an optimization algorithm. There are various types of optimization algorithm. In this paper, two optimization algorithms are used: mini-batch gradient descent and Levenberg-Marquardt. The two algorithms differ in their weight updating processes, where mini-batch gradient descent utilizes gradients [9] while Levenberg-Marquardt utilizes Jacobian matrix to update weights [10].

This paper aims to identify at-risk students regarding to academic performance decline during online learning by classifying whether students passed or failed on a course through students’ learning activity data collected in Learning Management System (LMS). For an early prediction, ANNs with both optimization algorithms are implemented on data before the online learning study period ends: the first month and two months of study. The prediction in this paper is also expected to contribute as an evaluation tool to an upcoming online learning using LMS.

2. Data
The data were collected about a total of 232 students of Department Mathematics of Universitas Indonesia who enrolled in course X for 4 months. For an early prediction, the data used are the first one month and two months of study, before the midterm exam. In this paper, students’ learning activity data in course X recorded in LMS Universitas Indonesia is used. Features chosen in the data are features that affect students’ academic performance based on related research [2]. There features are divided into 5 parts, class access (Course viewed), quiz attempts (Quiz attempt started and Quiz attempt submitted), tasks upload (A submission has been submitted), study materials access (File viewed and URL viewed), and activities on discussion forum (discussion viewed and some content has been posted).

Furthermore, the end-of-term grade of students in course X is taken as an indicator of academic performance and converted into a categorical form: “Failed” if the score is less than 55 and “Passed” if the score is equal or greater than 55. In this paper, category of “Failed” is denoted as 1, while “Passed” is denoted as 0. After converting, we know that 219 students passed the course X while 13 students failed.

3. Method
Artificial Neural Network (ANN) is formed through a collection of artificial neurons like biological neurons in human brain to process information with improved performance through learning data [3]. ANN can recognize complex correlation, nonlinear, unknown, between input and output data [4]. ANN consists of a vast variety of architectures, namely, single layer perceptron and multilayer perceptron. Multilayer perceptron is a type of architecture that commonly used for doing information processing. Multilayer perceptron (MLP) has a structure of an input layer that contains input data, one or more hidden layers to process the data, and an output layer (L) that consists of neurons to produce learning outcomes. Neurons in each layer are connected using a connection that contain adjustable weights during data processing. ANN operates training and testing processes. There are two main phases in training process: the forward phase and the backward phase.

In the forward phase, input vector transmits from input layer to output layer to obtain the expected output. For layer \( l = 2, 3, ..., L \), forward phase is done for each \( p \)th example using the following equations:

\[
a^l_{p,j} = (w^{l-1})^T o^{l-1}_p + b^l_j, \tag{1}
\]

\[
o^l_{p,j} = \phi(a^l_{p,j}), \tag{2}
\]
where $a^l_{p,j}$ is an argument of activation function $\phi$, $w^{l-1}_j$ are the weights of all neurons in $(l-1)$th layer to the $j$th neuron in the $l$th layer, the bias in the $j$th neuron of the $l$th layer is denoted as $b^l_j$, and $o^l_{p,j}$ is an output of $j$th neuron in the $l$th layer for $p$th example. In the first layer, $p$th input example ($x_p$) can be written as $o^1_p$.

After the forward phase is done for all input examples, the expected output, $y_p = o^L_{p,1}$ is obtained, and the output error can be calculated using the loss function. The output error is used in the backward phase to update weights repeatedly until the stop condition is met. Weights updating is deployed with an optimization algorithm. In this paper, two optimization algorithms are used to update weights: mini-batch gradient descent and Levenberg-Marquardt.

### 3.1. Mini-batch Gradient Descent Algorithm

Gradient descent algorithm is an optimization algorithm that is commonly used to minimize loss values in the model [9]. Gradient descent algorithm uses information of error gradient with respect to weight $w$ to update weights. There are variants of gradient descent, one of them is mini-batch gradient descent.

Mini-batch gradient descent considers small number of examples over all the training examples to calculate gradient at a time to update the weights. The entire training dataset is split into some batches, where a batch is denoted as $B = \{x_p\}_{p=1}^m$, in which $m$ is a relatively small number of examples from $N$ training data. Mini-batch gradient descent reduces the variance of the weights update, which can lead to more stable convergence, and performs frequent updates that can result in faster learning [11]. In this paper, the batch size used is $32$ ($m = 32$), so there will be $32$ examples in a batch.

An update for every batch of $m$ training examples is calculated by using the following equation,

$$\left( w^{l-1} \right)’ = w^{l-1} - \frac{\alpha}{m} \sum_{p=1}^{m} \nabla w^{l-1} E(t_p, y_p),$$  \hspace{1cm} (3)

where $w^{l-1}$ and $(w^{l-1})’$ are weights before and after updating, respectively, $\alpha$ is the learning rate, and $\nabla w^{l-1} E(t_p, y_p)$ is the error gradient with respect to weight. Here, the loss function $E(t_p, y_p)$ is defined as:

$$E(t_p, y_p) = -[t_p \log y_p + (1 - t_p) \log(1 - y_p)],$$  \hspace{1cm} (4)

where $t_p$ is the $p$th target output, $y_p$ is the $p$th expected output.

An epoch in ANN consists of several iterations used to update weights. In this algorithm, an epoch consists of a number of iterations, which is equal to the number of batches. The algorithm of mini-batch gradient descent for an epoch is as follows [12]:

1. Initialize all weights randomly with small random values.
2. Input a batch $B$ of training examples.
3. For each training example $p$ in batch $B$, perform the following steps:
   a. For each layer $l = 2, ..., L$, compute forward phase using equation (1) and equation (2). In the $L$th layer, we obtain the expected output $y_p = o^L_p$.
   b. Calculate the error value $E$ of the expected output using the loss function in equation (4).
4. For each $l = L, L - 1, ..., 2$, update the weights according to the equation (3).
5. Repeat steps 2 to 4 until all batches are used to update weights.

The algorithm is repeated until the desired error or a certain number of epochs is reached.
3.2. Levenberg-Marquardt Algorithm

Levenberg-Marquardt is an optimization method invented by Kenneth Levenberg and Donald Marquardt in the neural network field [10]. The Levenberg-Marquardt method adopts gradient descent and Gauss-Newton approach. By adopting those methods, Levenberg-Marquardt becomes more effective due to faster convergence and better stabilization in comparison with other methods [13]. The following equation is used to update weights using the Levenberg-Marquardt rule:

\[
w' = w - (J^T J + \mu I)^{-1} J^T e,
\]

where \(w\) and \(w'\) are weights before and after updating, respectively, \(J\) is Jacobian matrix, \(I\) is an identity matrix and \(\mu\) is a damping factor. Damping factor values are updated in each iteration by multiplying or dividing the damping factor by a power of 10 or some other factors [10]. The \(e\) is an error vector consists of errors from each training data. With one neuron output, the Jacobian matrix for \(N\) examples in equation (5) can be written as follows:

\[
J = \begin{bmatrix}
\frac{\partial e_1}{\partial w_{1,1}} & \frac{\partial e_1}{\partial w_{1,2}} & \cdots & \frac{\partial e_1}{\partial w_{1,j}} \\
\frac{\partial e_2}{\partial w_{1,1}} & \frac{\partial e_2}{\partial w_{1,2}} & \cdots & \frac{\partial e_2}{\partial w_{1,j}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial e_p}{\partial w_{1,1}} & \frac{\partial e_p}{\partial w_{1,2}} & \cdots & \frac{\partial e_p}{\partial w_{1,j}} \\
\frac{\partial e_N}{\partial w_{1,1}} & \frac{\partial e_N}{\partial w_{1,2}} & \cdots & \frac{\partial e_N}{\partial w_{1,j}} \\
\frac{\partial e_1}{\partial w_{l,1}} & \frac{\partial e_1}{\partial w_{l,2}} & \cdots & \frac{\partial e_1}{\partial w_{l,j}} \\
\frac{\partial e_2}{\partial w_{l,1}} & \frac{\partial e_2}{\partial w_{l,2}} & \cdots & \frac{\partial e_2}{\partial w_{l,j}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial e_p}{\partial w_{l,1}} & \frac{\partial e_p}{\partial w_{l,2}} & \cdots & \frac{\partial e_p}{\partial w_{l,j}} \\
\frac{\partial e_N}{\partial w_{l,1}} & \frac{\partial e_N}{\partial w_{l,2}} & \cdots & \frac{\partial e_N}{\partial w_{l,j}} \\
\end{bmatrix}
\]

where \(e_p\) is the error of the \(p\)th example with \(p = 1, 2, ..., N\) and \(w_{l,j}^i\) is the weight of \(i\)th neuron in the \((l-1)\)th layer connected to the \(j\)th neuron in the \(l\)th layer with \(i = 1, 2, ..., n_l - 1\), \(j = 1, 2, ..., n_l\) and \(l = 2, 3, ..., L\). Each element in equation (6) is calculated using the following equation [10].

\[
\frac{\partial e_p}{\partial w_{l,j}} = -\delta_{p,l}^j x_{p,j},
\]

where \(x_{p,j}\) is the input from \(j\)th neuron for \(p\)th example and \(\delta_{p,j}^l\) is the value of the \(i\)th neuron in the \(l\)th layer for \(p\)th example. The element \(\delta_{p,l}^i\) can be calculated in the backward phase with these following equations:

\[
\delta_{p,l}^{i-1} = \sum_{j=1}^{n_l} w_{l,j}^i \delta_{p,j}^l,
\]

\[
\delta_{p,l}^l = \delta_{l-1}^{l-1} s_{p,l}^l.
\]

Equation (8) is used when layer \(l\) is backpropagated to layer \(l - 1\), where \(w_{l,j}^i\) is the weight of \(i\)th neuron connected to the \(j\)th neuron in the \(l\)th layer and \(\delta_{p,j}^l\) is a value of the \(j\)th neuron in the \(l\)th layer for \(p\)th example. Meanwhile, the equation (9) is used to backpropagate within the same layer \(l\). The element slope \(s_{p,l}^l\) in equation (9) can be obtained in forward phase with following equation:
\[ s_{p,j}^l = s_{p,j}^l = \frac{\partial o_{p,j}^l}{\partial a_{p,j}^l}, \]

where \( o_{p,j}^l \) and \( a_{p,j}^l \) can be obtained from equation (1) and (2), respectively.

The loss function is used to calculate error for Levenberg-Marquardt algorithm is the Mean Square Error (MSE), defined as:

\[ E(t_p, y_p) = \frac{1}{N} \sum_{p=1}^{N} (t_p - y_p)^2, \]

where \( t_p \) is \( p \)th target output, \( y_p \) is \( p \)th desired output, and \( N \) is the number of training examples.

In this algorithm, an epoch consists of an iteration.

The algorithm of Levenberg-Marquardt is as follows [10]:

1. For \( i \)th iteration:
   a. Input \( N \) training data and initial weights \( w(i) \).
   b. For each layer \( l = 2, \ldots, L \), compute forward phase using equation (1) and equation (2), also calculate the slope using equation (10). In \( L \)th layer, we obtain expected output \( y_p = o_{L\,p}^L \).
   c. Calculate an error value in \( i \)th iteration, denoted as \( E(i) \), using equation (11).
   d. Update the weights \( w(i) \) using equation (5) to obtain the weights \( w(i + 1) \).

2. For \( (i + 1) \)th iteration:
   a. Repeat step 1(b) using \( w(i + 1) \).
   b. Calculate \( E(i + 1) \).
   c. Compare \( E(i) \) with \( E(i + 1) \):
      (a) If the \( E(i + 1) \leq E(i) \), \( \mu \) is decreased by dividing its value by 10. Then, update \( w(i + 1) \).
      (b) If the \( E(i + 1) > E(i) \), restore \( w(i) \) as \( w(i + 1) \) and increase \( \mu \) by multiplying its value by 10. Then, update \( w(i + 1) \).

The algorithm is repeated until the desired error or a certain number of iterations is reached.

4. Proposed Methods

Before ANNs are implemented on data, the data must be pre-processed beforehand. In this paper, two pre-processing procedures are conducted, namely data normalization and applied adaptive synthetic sampling. Before normalizing data, each data is divided into training data and testing data with proportion of 70:30. It means 162 data are used as training data, while 70 data are used as testing data.

The first pre-processing procedure is data normalization, which transforms the range of values of each feature for both training and testing data, so that all features have the same range of values. Thus, the yielded result will be more accurate [14]. Min-max normalization technique is conducted in this paper’s pre-processing in which the range of each input feature is rescaled to the range 0 to 1, with following equation:

\[ x' = \frac{x - \max(x)}{ \max(x) - \min(x) }, \]

where \( x \) and \( x' \) are the value of feature before and after normalization, respectively, \( \max(x) \) is the maximum feature value, \( \min(x) \) is the minimum feature value [15].

In classification, classes with less data, known as minority classes, cause imbalanced data. Classification may be biased to majority class or the data that belongs to the minority class can be misclassified as the majority class [16]. An approach to handle imbalanced data is using Adaptive
Synthetic (ADASYN). ADASYN is a technique for data balancing by generating synthetic data of minority class based on density distribution for different minority class examples depends on their level of difficulty in learning. Examples in minority class that are harder to learn will be used to generate more synthetic data compared to the easier examples. The basis for selecting the data to form the synthetic data with ADASYN reduces the bias generated when model classify the data [17].

The algorithm of ADASYN to make both minority and majority classes have the same amount of data is as follows [17]:

1. Calculate ratio \( d \) using equation (13),

\[
d = \frac{m_n}{m_j},
\]

where \( m_n \) and \( m_j \) are the number of minority and majority examples, respectively. If \( d < 1 \), ADASYN begins to initialize the generation of synthetic data.

2. Calculate the number of generated synthetic data \( G \) with the following equation:

\[
G = m_j - m_n.
\]

3. Calculate the ratio of each example \( x_i \) in the minority class as follows,

\[
r_i = \frac{z_i}{k}, \quad i = 1, 2, \ldots, m_n,
\]

with \( k \) is the determined number of examples in \( x_i \)'s nearest neighbors retrieved from Euclidean distance, and \( z_i \) is the number of majority class examples within \( k \) neighbors.

4. The previous ratio value is normalized with the following equation:

\[
\hat{r}_i = \frac{r_i}{\sum_i r_i}.
\]

5. Synthetic data are formed from each \( x_i \). First, determine the number of generated synthetic data \( g_i \) for each \( x_i \) as follows,

\[
g_i = \hat{r}_i G.
\]

6. The \( g_i \) synthetic data are formed by using the following equation:

\[
s_i = x_i + (x_o - x_i)\lambda, \quad i = 1, 2, \ldots, g_i,
\]

where generated synthetic data from each \( x_i \) is denoted as \( s_i \), \( \lambda \) is a randomized constant, ranged from \([0,1]\). A minority data example \( x_o \) from \( k \) nearest neighbors is chosen randomly.

After preprocessing data, two ANN models, one with mini-batch gradient descent and Levenberg-Marquardt, are implemented on data with periods of one and two months of studies. In this paper, there are 10 experiments in total for each model. For each experiment, the performance of each ANN model over each period is evaluated based on accuracy, sensitivity, and specificity, which are calculated as follows:

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN'}
\]
\[ Sensitivity = \frac{TP}{TP + FN}, \]  
\[ Specificity = \frac{TN}{TN + FP}, \]  

where TP, TN, FP, and FN are shown in Table 1. Accuracy is a measurement of a model performance in classifying all data correctly. Specificity is a measurement of a model performance in classifying positive class correctly, while sensitivity is a measurement of a model performance in classifying negative class correctly. In this study, the passing students are assigned as a positive class and the failing students are assigned to a negative class.

| Actual value | Prediction | Failed | Passed |
|--------------|------------|--------|--------|
| Failed       | TP         | FN     |
| Passed       | FP         | TN     |

Based on Table 1, the terms in the confusion matrix are TP, TN, FP, and FN. TP (True Positive) is the number of students correctly classified as failed. TN (True Negative) is the number of students correctly classified as passed. FP (False Positive) is the number of students classified as failed even though they have passed. FN (False Negative) is the number of students classified as passed even though they have failed.

Based on the results of the evaluation, the period over which each model has the best performance is chosen as the best one over which to make a prediction.

5. Experimental Results

The ANN models were implemented on data that is divided into 2 periods: the first one month and two months of study.

Before implementation, each model receives hyperparameter tuning. Tuning hyperparameters is necessary to find the best combination of hyperparameters, to prevent over- and under-fitting. The hyperparameter in ANN consists of the number of hidden layers, the number of neurons in hidden layer, activation function, epochs, etc. Here, 2 hidden layers with the same number of neurons and 150 epochs are used for both ANNs: one with mini-batch gradient descent and one with Levenberg-Marquardt.

The details of hyperparameters used in ANN with mini-batch gradient descent and Levenberg-Marquardt algorithms to predict students’ academic performance in first and second period can be seen in Table 2.

| Hyperparameter              | Mini-batch gradient descent | Levenberg-Marquardt |
|-----------------------------|-----------------------------|---------------------|
|                             | 1st period | 2nd period | 1st period | 2nd period |
| Hidden neuron               | 30         | 40         | 30         | 30         |
| Activation function         | sigmoid    | sigmoid    | tanh       | tanh       |
| Learning rate               | 0.1        | 0.1        | -          | -          |
| Batch size                  | 32         | 32         | -          | -          |
| Damping factor              | -          | -          | 3          | 3          |
The implementation requires 10 experiments on the data in each period for each model. Based on these experiments, the performance of each ANN with both optimization methods on each period, specifically average accuracy, sensitivity, and specificity, can be determined. Figure 1 shows the result of students’ academic performance prediction using ANN with both optimization algorithms.

![Figure 1](image1.png)  
**Figure 1.** Model performance based on the average of students’ academic performance prediction with (a) Mini-batch gradient descent and (b) Levenberg-Marquardt

As we can see from figure 1, the performance of the ANN model with both optimization algorithms is generally improved for each metric from the first period to the second. This indicates that there is more data regarding student learning activities recorded EMAS that can improve the performance of the model. In other words, ANN with both algorithms performs better in the second period of prediction, i.e., using the data from the first two months of studies. This is supported by related research where prediction of the percentages of students passing using data collected from 50% of study time yields superior performance compared to that obtained using data collected from 25% of study time [18].

In the first period, the ANN-based prediction of academic performance using mini-batch gradient descent have an average accuracy, sensitivity, and specificity of 79%, 53%, and 81%, respectively. Meanwhile, in the second period, the average accuracy, sensitivity, and specificity are 86%, 78%, and 87%, respectively, all of which are improvements on the percentages obtained from the first period. The improvements of percentages from first period to second period also occur when the ANN is implemented using the Levenberg-Marquardt algorithm. In figure 1(b), the average accuracy, sensitivity, and specificity of the model in the second period are 87%, 75%, and 86%, respectively, all of which are superior to the percentages after the first period. From the results, ANN for both algorithm is shown better performance on the second period. Further analysis on the results of the implementation in the 2nd period are shown on table 3 below.

| Optimization algorithm      | Accuracy | Sensitivity | Specificity |
|-----------------------------|----------|-------------|-------------|
| Mini-batch gradient descent | 86%      | 78%         | 87%         |
| Levenberg-Marquardt         | 87%      | 75%         | 86%         |

Table 3. The average result of students’ academic performance prediction using ANN with mini-batch gradient descent and Levenberg-Marquardt algorithms on 2nd period
Based on table 3, the average accuracy of the ANN model with mini-batch gradient descent is 86%; therefore, in 86% of data, it will yield a correct prediction. For comparison, an ANN model with the Levenberg-Marquardt algorithm has an average accuracy 87%, which gives it slightly superior accuracy to an ANN model with mini-batch gradient descent when making accurate data predictions.

Thereafter, the performance of an ANN model with both optimization algorithms is compared using average specificity and sensitivity. Based on table 3, the average sensitivity of an ANN model with mini-batch gradient descent is 78%. This means that an ANN model using mini-batch gradient descent can successfully predict 78% of failing students correctly as failing. Meanwhile, an ANN model using Levenberg-Marquardt has an average sensitivity of 75%, which is slightly inferior to that of an ANN model with mini-batch gradient descent algorithm.

In addition of sensitivity, an ANN model with mini-batch gradient descent algorithm has an average specificity of 87%. This indicates that it can predict 87% of passing students correctly as passing. Meanwhile, the average specificity of an ANN model with Levenberg-Marquardt has an average specificity of 86%, which is slightly inferior to that obtained using the mini-batch gradient descent algorithm.

In table 3, the average result of students’ academic performance prediction using ANN with both optimization algorithms are more than sufficient. However, in order to predict students’ potential to fail, sensitivity is considered as the main consideration as evaluation of both optimization methods’ performance to choose the best model to predict. From the results, mini-batch gradient descent yields slightly better sensitivity value than Levenberg-Marquardt. Thus, mini-batch gradient descent is a superior form of optimization alongside ANN to predict students’ academic performance.

### 6. Conclusion

Early students’ academic performance prediction is required to identify at-risk students with declining performance. To do so, ANN with mini-batch gradient descent and Levenberg-Marquardt are used to predict student performance. The implementation result shows that both algorithms give better prediction over two months than over one month. Therefore, predictions of academic performance should be made using data from two months’ worth of studies. In order to predict students’ potential to fail, sensitivity should be the main consideration to determine the best optimization algorithm. The result shows that the sensitivity value for ANN with mini-batch gradient descent is slightly higher than that of an ANN with Levenberg-Marquardt. Therefore, ANN using mini-batch gradient descent as its optimization algorithm is the superior option for making such predictions.

### Acknowledgements

This research was supported by Universitas Indonesia, PUTI grant scheme No.: NKB-1983/UN2.RST/HKP.05.00/2020.

### References

[1] Mheidly N, Fares M Y and Fares J 2020 *Front. Public Heal.* **8**
[2] Akçapınar G, Altun A and Aşkar P 2019 *Int. J. Educ. Technol. High. Educ.* **16**
[3] Lau E T, Sun L and Yang Q 2019 *SN Appl. Sci.* **1** 1–10
[4] Als Salman Y S, Khamees Abu Halemah N, Alnagi E S and Salameh W 2019 *10th International Conference on Information and Communication Systems, ICICS 2019*
[5] Ma J 2020 Machine learning in predicting diabetes in the early stage *Proceedings - 2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence, MLBDBI 2020*
[6] Zhang D and Lou S 2021 *Futur. Gener. Comput. Syst.* **115**
[7] Zacharis N 2016 *Int. J. Artif. Intell. Appl.* **7**
[8] Chunqiao M, Xiaoning P and Qingyou D 2018 An artificial neural network approach to student study failure risk early warning prediction based on TensorFlow *Lecture Notes of the Institute for
Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST vol 219

[9] Ruder S 2017 An overview optimization gradients arXiv Prepr. arXiv1609.04747
[10] Yu H and Wilamowski B M 2016 Levenberg-marquardt training Intelligent Systems
[11] Saunders D J, Sigrist C, Chaney K, Kozma R and Siegelmann H T 2020 M Proc. of the Int. Jt. Conf. Neural Networks
[12] Nielsen M A 2015 Determ. Press 224
[13] Gavin H P 2019 The Levenburg-marquardt algorithm for nonlinear least squares curve-fitting problems Duke Univ.
[14] Patro S G K and Sahu K K 2015 Normalization: A preprocessing stage IARJSET
[15] Witten I H, Frank E, Hall M A and Pal C J 2016 Data Mining: Practical Machine Learning Tools and Techniques
[16] Johnson J M and Khoshgoftaar T M 2019 J. Big Data 6
[17] He H, Bai Y, Garcia E A and Li S 2008 Proc. Int. Jt. Conf. Neural Networks 1322–8
[18] Riestra-González M, Paule-Ruíz M del P and Ortin F 2021 Comput. Educ. 163