

Adaptive Moment Estimation To Minimize Square Error In Backpropagation Algorithm

R N Singarimbun¹, E B Nababan^{2}, and Opim Salim Sitompul³*

¹Graduate School of Computer Science

^{2,3}Department of Information Technology, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia

Abstract. Backpropagation Neural Network has weaknesses such as errors of gradient descent training slowly of error function, training time is too long and is easy to fall into local optimum. Backpropagation algorithm is one of the artificial neural network training algorithm that has weaknesses such as the convergence of long, over-fitting and easy to get stuck in local optima. Backpropagation is used to minimize errors in each iteration. This paper investigates and evaluates the performance of Adaptive Moment Estimation (ADAM) to minimize the squared error in backpropagation gradient descent algorithm. Adaptive Estimation moment can speed up the training and achieve the level of acceleration to get linear. ADAM can adapt to changes in the system, and can optimize many parameters with a low calculation. The results of the study indicate that the performance of adaptive moment estimation can minimize the squared error in the output of neural networks.

Keywords: Gradient Descent Backpropagation, Adaptive Moment Estimation, Minimize Square Error

Received 20 September 2019 | Revised 25 December 2019 | Accepted 29 January 2020

1. Introduction

ADAM is the Moment Adaptive Estimation, which is one method for optimizing parameters. In the study [1] using the algorithm ADAM compensated Asynchronous Delay (DC - Adam) to train Deep Neural Network (DNN). DC - ADAM can get a more accurate gradients and faster in training progress, and easy to implement with minimal memory requirements. This research [2] on online learning algorithm by using Group Method Of Data Handling Based Proportional - Integral - Derivative (GMDH - PID) for non-linear systems. With online setting method using GMDH - PID using Adaptive Estimation Moments (ADAM), which is one method of

*Corresponding author at: Department of Information Technology, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Medan, Indonesia

E-mail address: ernabr@usu.ac.id

optimization that can adapt to changes in the system and also can optimize many parameters with a low calculation.

A study about the Levenberg Marquardt - Backpropagation (LM - BP) Based Operation Quality Assessment Method For OTN (Optical Transmission Network) in the Smart Grid used in the smart grid to improve transmission speed and capacity efficient network transmission. Backpropagation Neural Network has weaknesses such as errors from gradient descent training on slowly of error function, training time is too long and is easy to fall into local optimum. While, LM algorithm convergence speed and robustness of the best on the network that are robustness resistant. The results of this study with the backpropagation algorithm Levenberg Marquardt (LM) have more optima prediction accuracy, have a better network structure and accuracy errors increased significantly from the standard back propagation neural network [3].

Artificial Neural Network (ANN) has a problem in determining the weight to the right network. A study comparing the Gradient Descent and Genetic Algorithm (GA) based training Artificial Neural Network (ANN). GA slightly better in Mean Square Error (MSE) in cancer datasets for classification errors are average, but better Gradient Descent in the dataset diabetes. From this study, it is still necessary to experiment with more datasets in ANN training [4].

In the study to improve the gradient descent in artificial neural network such as quickprop, backpropagation, Delta-Bar-Delta and Super SAB as the approximate error of function with quadratic polynomial to get the minimum squared error function. The partial derivative method until the process of update weights in the gradient descent on backpropagation can be modified to the level of learning at each weight to the neurons in the network. Improved gradient descent is better than standard gradient descent and gradient descent momentum [5].

In the study to evaluate and test the three gradient descent based on backpropagation to classify benign and malignant tumors in ultrasound imaging. In selecting the right learning rate, time complexity and network model are still important in the network system at the time of convergence in the classification process. Gradient descent (GD), the gradient descent with momentum (GDM) and adaptive gradient descent (AGD) is used for training the classification model for testing and validation. The results of this study, backpropagation based AGD is better in the process of classifying benign and malignant tumors, but AGD algorithms are very long in time complexity [6].

In the study of artificial neural networks, mean square error (MSE) is a problem that is used in learning. The training uses a theoretical backpropagation method, for correntropy conjugate gradient-based BP (CCG-BP). CCG - BP gets better results than MSP-based correntropy-based backpropagation and can minimize MSE [7]. A study of [8] to improve the convergence and global search capabilities of the network backpropagation (BP). BP has weaknesses such as long

convergence, over-fitting and easily trapped to local optima. Genetic Algorithm (GA) can improve BP by evaluating natural selection, genetic crossover and mutation gen that have advantages such as very high parallels, stochastic and global probability searches that can overcome BP shortcomings. The result is that GA can set specific targets in starting BP weight, adjusting BP training so that the epoch is smaller.

In the study using Backpropagation Modified Adaptive Approach (AMBP) in improving the performance of the Modern Artificial Intelligence algorithm to accelerate convergence by adjusting the learning rate at each layer and epoch. AMBP algorithm with a learning rate variable for the process of classifying data quickly and getting a small MSE in a short time. To improved the learning rate with momentum can be added to the network, this algorithm can be known as the backpropagation with momentum algorithm (BPM). The results of previous studies, AMBP are much better and able to provide MSE that is less than SBP and BPM [9].

Artificial Neural Networks (ANN) using levenberg - marquardt training to optimize weights on ANN. Statistical methods and ANN are methods used to predict. However, ANN does not make the basic structure of the system compared to the statistical method. ANN is also a linear regression that is complex, nonlinear and dynamic. The levenberg - marquardt algorithm is close to the speed of training, so that the performance function will always be reduced at each algorithm iteration. The Levenberg - Marquardt algorithm is the fastest method for training artificial neural networks to several hundred weights. The result is that the Levenberg-Marquardt algorithm has errors that are relatively less than 3% Mammadli [10].

In the study of Optimizing the Backpropagation by using the Nguyen – Windrow method on the input layer of the feed – forward process and adjusting the learning rate parameter in the backward process. The influence of learning with adaptive learning rate changes using a randomly selected initial weight with the Nguyen – Windrow method. Backpropagation is used to minimize error in every iteration. The result in the feed – forward phase with Nguyen – Windrow's initial weight method was able to give close value to the error value affecting the weight update to the backward phase. In the results of the backward phase adaptive learning rate parameters can be pain number of iterations (epoch) [11].

In this paper, the learning process of the backpropagation algorithm is still slow in training gradient descent from error functions and requires a very long processing time. This is because the architecture, learning rate, overfitting and the number of epochs in the training are still high so that the solution is easy to fall into the optimum local. Gradient descent backpropagation is also still not good at minimizing squared errors, so a suitable approach is needed in order to improve the gradient descent backpropagation learning process. The performance of Adaptive Estimation moment to minimize the squared error in backpropagation gradient descent algorithm.

ADAM can update the parameters of the output torque which is a torque distribution first and second moments in backpropagation gradient descent. The purpose of this study is to minimize the squared error at each iteration (epoch) at the output of the neural network. The results showed that ADAM can minimize the squared error at each iteration (epoch) at the output of neural networks.

2. Adaptive Moment Estimation And Gradient Descent Backpropagation Algorithms

2.1 Adaptive Moment Estimation

ADAM is used for optimizing a gradient descent learning algorithm to minimize the objective function (often called the loss function $E(x)$) on various parameters such as weights and biases. Error in backpropagation is the mechanism used to modify network parameters before initialization parameters to get optimized and can produce output is approaching the target output. In the error back propagation neural network used, the process of calculating the feedforward output one by one and calculate the error component obtained in the last layer. Gradient is calculated on backpropagation to get the network to be optimized. Here are the steps - steps ADAM as follows [12]:

- Initialization $m_{weight}(t)$, $m_{bias}(t)$, $v_{weight}(t)$ and $v_{bias}(t) = 0$

If the first iteration is $t = 1$, $t = 1 - 1 = 0$ (time step / early iterations on the input)

- Gradient calculation for estimating the first moment in time step $= g_t \frac{\partial E}{\partial W_{tk}^o}$
- The calculation of the estimated first moment (**mt**) weight and bias can be done after receiving the derivative calculation squared error in the output layer by the following equation:

$$\begin{aligned} m_{weight(t)} &= \beta_1 * m_{weight(t-1)} + (1 - \beta_1) * g_t \\ m_{bias(t)} &= \beta_1 * m_{bias(t-1)} + (1 - \beta_1) * g_t \end{aligned} \quad (1)$$

- The calculation of weight and bias correction estimation of the first moment \hat{m}_t by the following equation:

$$\begin{aligned} \hat{m}_{weight(t)} &= \frac{m_{weight(t)}}{1 - \beta_1^t} \\ \hat{m}_{bias(t)} &= \frac{m_{bias(t)}}{1 - \beta_1^t} \end{aligned} \quad (2)$$

- Gradient calculation for estimating the first moment in time step $= g_t \frac{\partial E}{\partial W_{ij}^h}$

- The calculation of the estimated second moment (**vt**) weight and bias can be done after receiving the derivative calculation output to the hidden layer by the following equation:

$$\begin{aligned} v_{weight(t)} &= \beta_2 v_{weight(t-1)} + (1 - \beta_2) * (g_t)^2 \\ v_{bias(t)} &= \beta_2 v_{bias(t-1)} + (1 - \beta_2) * (g_t)^2 \end{aligned} \quad (3)$$

- The calculation of weight and bias correction the estimated second moment \tilde{V}_t by the following equation:

$$\begin{aligned}\tilde{V}_{weight(t)} &= \frac{v_{weight(t)}}{1 - \beta_2^t} \\ \tilde{V}_{bias(t)} &= \frac{v_{bias(t)}}{1 - \beta_2^t}\end{aligned}\quad (4)$$

- Parameter updates weight and bias by the following equation:

$$\begin{aligned}w_{weight(t)} &= w_{weight(t-1)} - \alpha * \frac{\hat{m}_{weight(t)}}{\sqrt{\tilde{V}_{weight(t)} + \epsilon}} \\ w_{bias(t)} &= w_{bias(t-1)} - \alpha * \frac{\hat{m}_{bias(t)}}{\sqrt{\tilde{V}_{bias(t)} + \epsilon}}\end{aligned}\quad (5)$$

2.2 Gradient Descent Backpropagation

Backpropagation gradient descent algorithm to minimize Square Error (SE) for the multilayer feedforward neural network. The learning rule to change the weights and bias on the output neuron layer and hidden layer neurons. The following steps - steps in the gradient descent backpropagation ADAM [13]:

- Initialize the weights randomly on each neuron located in the input layer, hidden layer and output layer.
- Phase feed forward propagation:
 1. Calculate each neuron in the hidden layer to the equation:

$$net_{ij}^h = \sum_{i=1} X_i W_{ij}^h + w_{bias_j}^h \quad (6)$$

2. Calculate each neuron's activation function in the hidden layer with sigmoid equation:

$$f_{in_net\ n} = \frac{1}{1 + e^{-(net_{ij}^h)}} \quad (7)$$

3. Calculates the total value of the output layer to the equation:

$$net_{ik}^o = \sum_{i=1}^n (f_{in_net\ n} * w_{ik}^o) + w_{bias_k}^o \quad (8)$$

4. Calculates the sigmoid activation function in the output layer to the equation:

$$f_{out_net\ n} = \frac{1}{1 + e^{-(net_{ik}^o)}} \quad (9)$$

5. Calculates the error in the output layer based on the difference between the target and output by the equation:

$$e_{output} = Target - f_{out_net\ n} \quad (10)$$

6. Calculates the square error at the output layer to the equation:

$$\text{Square Error} = \frac{1}{2} * \sum_k (\text{Target} - f_{out_net\ n})^2 \quad (11)$$

7. Calculates the partial derivative of the weight and bias for each neuron in the output layer with the equation:

$$\frac{\partial E}{\partial W_{ik}^o} = -(\text{Target} - f_{out_net\ n}) * f'_{out_net\ n} * f_{in_net\ n} \quad (12)$$

$$\frac{\partial E}{\partial W_{bias}^o} = -(\text{Target} - f_{out_net\ n}) * f'_{out_net\ n}$$

8. Calculates the partial derivative of the weight and the bias for each neuron in the hidden layer with the equation:

$$\frac{\partial E}{\partial W_{weight\ ij}^h} = - \sum_k (T - f_{out_net\ n}) * f'_{out_net\ n} * W_{ik}^o * f'_{in_net\ n} * X_i \quad (13)$$

$$\frac{\partial E}{\partial W_{bias\ i}^h} = - \sum_k (T - f_{out_net\ n}) * f'_{out_net\ n} * W_{ik}^o * f'_{in_net\ n}$$

3. Methodology

The methodology is divided into two parts on the backpropagation network architecture, that is, giving a feed forward for weighting and part of the backward feed for error values. Starting from output, so that each neuron has a corresponding error value that roughly represents its contribution to the original output. The steps to make a research design are as follows:

- Prepare data as enter 699 data which has 9 variables and 1 target variable for class.
- The pattern of designing network architecture is the number of neurons in the input layer, the number of neurons in the hidden layer and the number of neurons in the output layer.
- Run backpropagation with adaptive moment estimation with random initial weights.
- Analysis.

3.1 Data Input

The data used in this research is data about Wisconsin Breast Cancer dataset from the University of California Irvine (UCI) Machine Learning Repository. Data has 9 attributes were rated visually with the appropriate class variables and defined for each record in the dataset. All values on 9 attributes are indexed from 1 - 10 interval ranges, while the range class value on breast cancer cells is 2 for the benign category and 4 for the malignant category. The following Table 1 descriptions of datasets WBCD:

Table 1. Descriptions of datasets WBCD

Attribute	Range interval
Clump Thickness	1-10
Uniformity of Cell Size	1-10
Uniformity of Cell Shape	1-10
marginal Adhesion	1-10
Single Epithelial Cell Size	1-10
Bare Nuclei	1-10
Bland Chromatin	1-10
normal Neucleoli	1-10
Mitoses	1-10
Class variable	Benign cells (2) and malignant cells (4)

3.2 Block Diagram

The following figure 1 is a block diagram which aims to research done in the process is not out of the specified path. Block diagram shown in Figure 1.

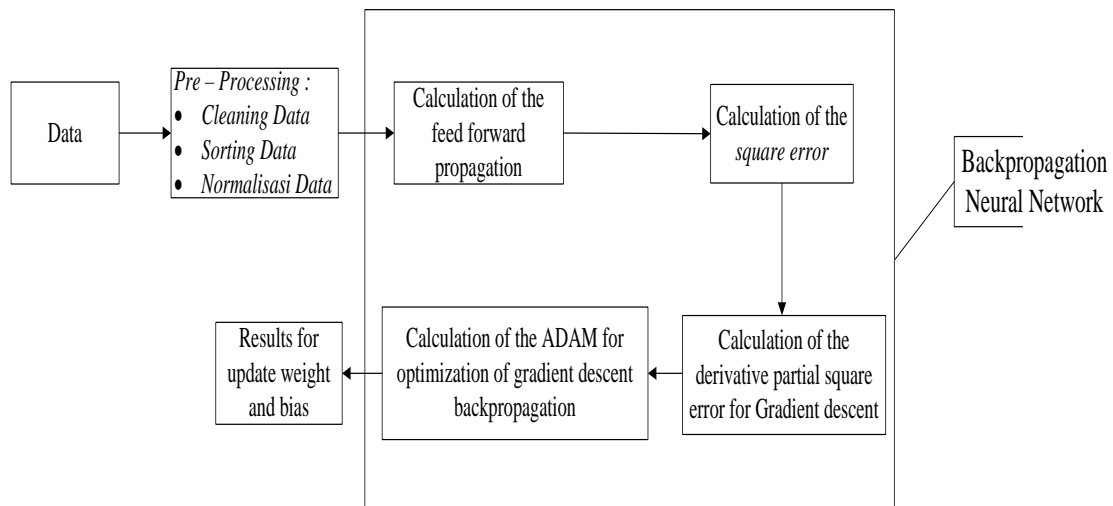
**Figure 1.** Block diagram

Figure 1 can be explained that the block diagram above is the backpropagation neural network architecture with multiple processes, namely:

- The data used is the data derived from Wisconsin Breast Cancer dataset from the University of California Irvine (UCI) Machine Learning Repository.
- Pre-processing stage has three processes, namely:
 1. Data Cleaning: used to fill in missing data values in as many as 16 data on the bare nuclei variable using the equation Paulin & Santhakumaran median method [14]:

$$\text{MEDIAN} = \text{size of } \frac{(N+1)}{2} \quad (14)$$

2. Sorting the data: is used to separate the data according to the class that is benign and malignant.
3. Normalization of data: aims to change the value on the data value range 1-10 into a value range of 0-1 by using the following equation [15]:

$$X'_i = \frac{X_i - X_{\min}}{X_{\max} - X_{\min}} \tag{15}$$

3.3 Backpropagation Neural Networks Architecture

The data used for the research is binary value after cleaning data, the dataset is 699 data and has 9 variables and 1 target variable for the class. After the dataset is carried out in the pre-processing stage, the data is then used as a dataset during testing to the input layer of the network that will be calculated for each neuron in the hidden layer and output layer. The results (output) of the error crater will be used as test data to see the minimization of the squared error. To test dataset, the following figure 2. Backpropagation neural network architecture:

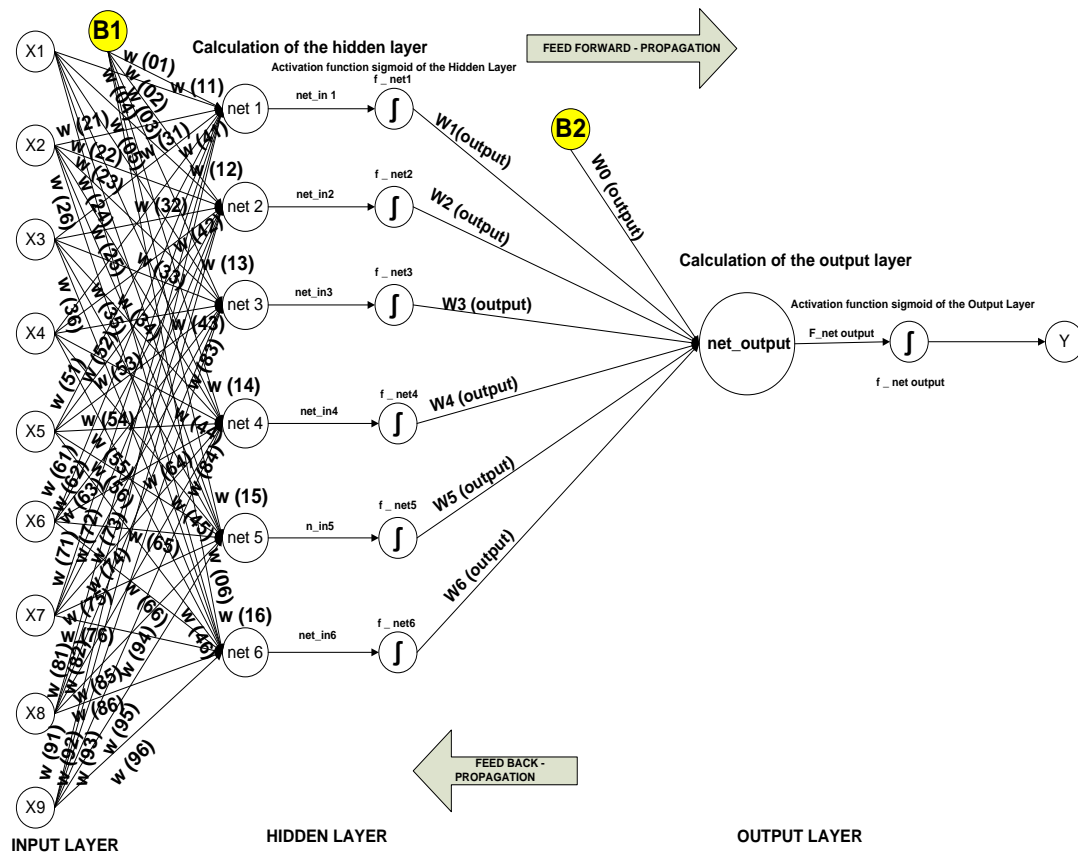


Figure 2. Backpropagation neural network architecture

Figure 2 is a back propagation neural network design. Input on this draft architecture adapted to the feature dataset from the UCI Machine Learning is 9 neurons in the input layer. In the hidden layer based on the calculation that has been calculated and determined, then the neurons in the

hidden layer of 6 neurons. And to output according to the data layer, called the output layer as many as 1 neurons.

3.4 ADAM Architecture Design In Gradient Descent Backpropagation

The image used is 24-bit color image. The calculation of MSE and PSNR aims to determine how much the image changes after message insertion. There is 1-bit storage up to 4-bit LSB to be performed, each stego-image will be calculated MSE and PSNR values to determine which image is better or how many better bits to store information or messages. The following is the formula used to calculate MSE and PSNR. Using (2.2) & (2.3)

Here is a figure 3 is the architecture of ADAM on a gradient descent backpropagation which aim to test dataset:

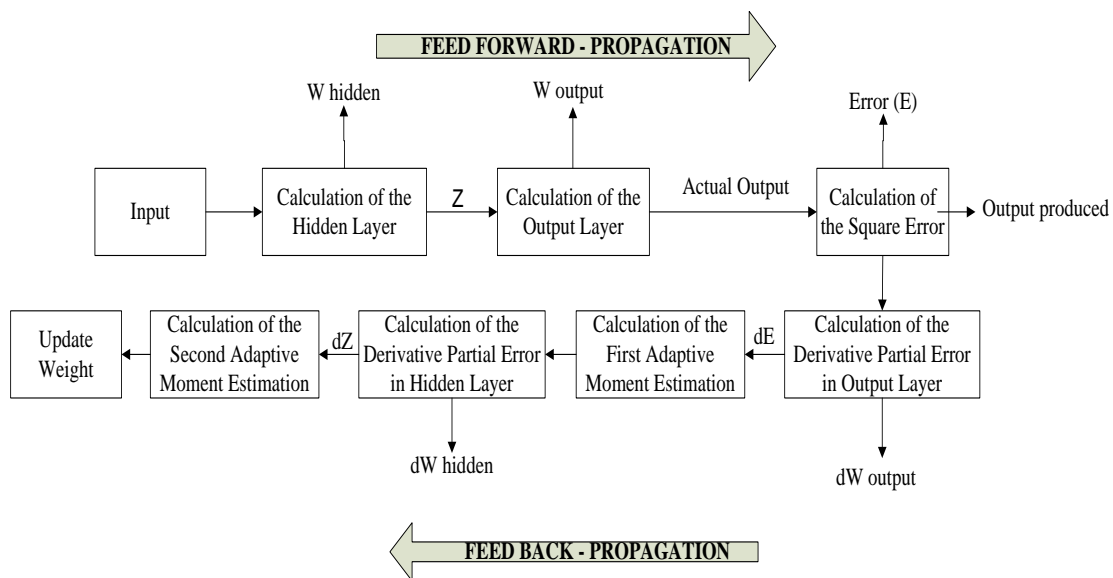


Figure 3. ADAM architecture on gradient descent backpropagation

Figure 3 is an architecture design of ADAM on a gradient descent backpropagation. In the feed forward propagation process, the input data that will be calculated on a hidden layer and output layer computations on. In the calculation results will be summed output layer to the actual value output (target value) in the data, after the reduction process will be conducted gradient descent to minimize the squared error propagation. In the propagation process takes partial derivative calculation process of error in the weight and bias in the output layer. ADAM The first will be done after getting the calculation of partial derivatives of the error in the output layer of weights and biases. After getting the results of the partial derivative calculation on the weights and biases, calculation to estimate the first moment and the moment that the first results will be corrected to gain weight and bias in the output layer. ADAM second will be carried out after obtaining the partial derivative calculation of error in hidden layer of weights and biases. After getting the results of the partial derivative calculation on the weights and biases, carried out calculations to

estimate the second moment and the second moment results will be corrected to get the weights and biases at the hidden layer, so that updates the weights and update the bias obtained.

4. Result and Discussion

The main objective of this research is to focus on the minimization of the squared error in the feed forward propagation. Initial weights randomly in feed forward propagation and update weights in at feedbackward propagation process. The process of the feed forward propagation on network performance and update of the buffer weight is affected by the provision of learning rate parameter value on network performance can be seen in Table 2.

Table 2. Parameter Value for adaptive estimation moment in gradient descent backpropagation

Parameter	information
Number Of Hidden Neurons	6
Activation Function	Binary sigmoid
maximum Epoch	5
minimum Error	0:01
Learning Rate	0001
Initialization bias and weight to network	Random
Architecture	Multilayer Network: Input: 9 Neuron Hidden Layer: 6 neurons Output: 1 neuron (second class)
Optimization In Backpropagation	Adaptive Moment Estimation $\varepsilon : 0.00000001 (10^{-8})$ The value of the Exponential for estimation of the first moment (β_1): 0.9 The value of the Exponential for the estimation of the second moment (β_2): 0.999

The data consists of 699 data, the number of variables in the input layer 9 and 1 variable to the target (have 2 classes). Later in the input data before, performed a pre processing to get the value range of 0 - 1. The result of the implementation of this program is to minimization of the squared error and the performance of the gradient descent back propagation neural network using ADAM, so I know how the system back propagation neural network to recognize a given pattern. Tests carried out with 5 testing, first on epoch 1, until the fifth test of the epoch 5. In the first test in epoch 1 consists of 699 iteration process, so that by the fifth test at the epoch 5 has the overall iterative process as much as 3494 iterations.

4.1 The First Test On EPOCH 1

The results of ADAM testing on gradient descent backpropagation for the first test on EPOCH 1 can be seen in Figure 4:

Error			Error			Error			Error		
Iteration	...	Square Error	Iteration	...	Square Error	Iteration	...	Square Error	Iteration	...	Square Error
1	...	0.2183443914320067	433	...	0.2908356936767408	577	...	0.20463806082345726	652	...	0.27508500707917555
2	...	0.21833426534162517	434	...	0.29093001507672833	578	...	0.20508514004861933	653	...	0.2761884674389882
3	...	0.21831754609634085	435	...	0.29102417519378726	579	...	0.2055672537873786	654	...	0.27729919710794615
4	...	0.2183017266126326	436	...	0.2911178986891049	580	...	0.20607395618505692	655	...	0.27840070708570175
5	...	0.21828144360287252	437	...	0.29121140685837754	581	...	0.20660439991475316	656	...	0.27949303495148825
6	...	0.2182599009705588	438	...	0.29130435590894627	582	...	0.20716852461310428	657	...	0.2805796620352388
7	...	0.2182357025427679	439	...	0.29139760803185555	583	...	0.20776477114052064	658	...	0.28166328907049043
8	...	0.21820982212860704	440	...	0.2914901013472916	584	...	0.2083913961368368	659	...	0.28275814466062965
9	...	0.21818371993013594	441	...	0.2915827121983036	585	...	0.209040315240835	660	...	0.2838584801403564
10	...	0.218157779019354	442	...	0.2916747253761926	586	...	0.20971254074474288	661	...	0.28496070607119595
11	...	0.21813291509924626	443	...	0.2917665056561608	587	...	0.2104010408243566	662	...	0.2860644311671636
12	...	0.21811060973154958	444	...	0.2918575297620595	588	...	0.21109213992087086	663	...	0.28715144137608245
13	...	0.21809117680189408	445	...	0.29194797568149294	589	...	0.21179043792140098	664	...	0.288236434263162
14	...	0.21807547275643813	446	...	0.2920385034076308	590	...	0.21251137829608513	665	...	0.2893079704538922
15	...	0.2180643898090109	447	...	0.29212832353105417	591	...	0.21324588522448748	666	...	0.29039114058067295
16	...	0.21805765830132776	448	...	0.2922174821036761	592	...	0.21399165489344618	667	...	0.29146182102240886
17	...	0.218055265341538	449	...	0.2923060206637339	593	...	0.21475975859499913	668	...	0.2925313529786253
18	...	0.21805873586664988	450	...	0.2923939766883878	594	...	0.21554055911020037	669	...	0.29359400651471657
19	...	0.2180677381706457	451	...	0.29248197971452383	595	...	0.21634059918128565	670	...	0.29464857090967395
20	...	0.21808217436254637	452	...	0.2925697817401076	596	...	0.21714488690632475	671	...	0.2957011727308291
21	...	0.21810225028688568	453	...	0.292653469769873	597	...	0.21795506858084307	672	...	0.2967481262936963
22	...	0.21812819263374175	454	...	0.2927407199430027	598	...	0.21878538809164508	673	...	0.297773926614608
23	...	0.2181601938936229	455	...	0.29282595571699144	599	...	0.2196288154288003	674	...	0.2988038148982141
24	...	0.21819844286081216	456	...	0.2929108181280084	600	...	0.22048192381407175	675	...	0.2998362416901186
25	...	0.2182430866346172	457	...	0.2929955911348958	601	...	0.22134013141240177	676	...	0.3008778234368967
26	...	0.21829370300003864	458	...	0.29308008851001044	602	...	0.22222143910369702	677	...	0.3019330824060276
27	...	0.21835212601839613	459	...	0.29302972724127624	603	...	0.22310341359329272	678	...	0.3029999755018039
28	...	0.21842510510790553	460	...	0.29287307122771	604	...	0.22401351539783182	679	...	0.3040594008962237
29	...	0.2185027968265684	461	...	0.29260394578205656	605	...	0.22494284463513914	680	...	0.30513668011910405
30	...	0.21858529723082262	462	...	0.29224969977922344	606	...	0.22589886645784346	681	...	0.30620355323868664
31	...	0.2186737745486807	463	...	0.2918021923610752	607	...	0.22685786625490298	682	...	0.3072578842628227
32	...	0.2187671866558931	464	...	0.2912710372532344	608	...	0.2278289338634118	683	...	0.3083202185525004
33	...	0.2188652985332788	465	...	0.2906650459245583	609	...	0.228815779272003	684	...	0.3093693794449982
34	...	0.21896901304775349	466	...	0.2900015104017409	610	...	0.22981743465892182	685	...	0.31041127753027803
35	...	0.21907774850203118	467	...	0.28928014850209544	611	...	0.23083136825597572	686	...	0.3114529925837359
36	...	0.21919254002789665	468	...	0.2884982692951264	612	...	0.23185514665745374	687	...	0.3124854079081913
37	...	0.21931383146701555	469	...	0.28766749546111114	613	...	0.23289220933309812	688	...	0.31356005431036652
38	...	0.21944011970893126	470	...	0.28678620767291246	614	...	0.23394211284368446	689	...	0.3145077647513129
39	...	0.21957101394059167	471	...	0.2858617550401845	615	...	0.23499168962047676	690	...	0.31552079263530025
40	...	0.21970784871846252	472	...	0.28489724102946595	616	...	0.2360518840046439	691	...	0.31653328481050713
41	...	0.21984932122790785	473	...	0.2838837626535493	617	...	0.23712689476283833	692	...	0.31753033793888913
42	...	0.2199948440373561	474	...	0.28284377623786794	618	...	0.2381986276550333	693	...	0.31852670007530987
43	...	0.22014496333359027	475	...	0.2817727269309319	619	...	0.2392974681874346	694	...	0.31951229076333454
44	...	0.2202994480642018	476	...	0.28066707968013793	620	...	0.240394899330636	695	...	0.32049352018635585
45	...	0.22045886250065064	477	...	0.27952568245051485	621	...	0.24148503721972372	696	...	0.32147163087473796
46	...	0.22062245334364802	478	...	0.27835632098039303	622	...	0.2425860255192229	697	...	0.3224377045464258
47	...	0.2207899676254977	479	...	0.2771685623143681	623	...	0.24367992340253444	698	...	0.3234075897866883
48	...	0.2209609412702453	480	...	0.27595662999501563	624	...	0.24478754422937726	699	...	0.324379559070086

Figure 4. The squared error value in the output layer at epoch 1 starts from iteration 1 to iteration 699

In the results, it can be seen in each iteration of the first test on epoch 1 from iteration 1 to iteration 466 getting an increase from the result of the squared error which is 0.2183443914320067 up to 0.2900015104017409. In the iteration of 467 until iteration 586 gets a decrease from the square of the error which is valued at 0.28928014850209544 it decreases to 0.20971254074474288. In the 587 iteration up to 699 iterations occurred an increase from the results of the error squared value which was 0.2104010408243566 up to 0.324379559070086. So that the first test on epoch 1 in each iteration starting from iteration 1 to iteration 699 does not get a decrease / minimization

In the results, it can be seen in the second test on epoch 2 in each iteration of 700 iterations up to iteration 1399 indicating a decrease / minimization of the squared error value of 700 iterations to 863 iterations which are 0.32498026050398193 decreased to 0.1899676752888724. In the iteration process 864 up to iteration 1167 get an increase from the results of the error squared value which is worth 0.1901636290061076 up to 0.25079291962345257. In the iteration process 1168 up to 1284 iterations get a decrease / minimization of the squared error which is 0.24978478229861384 decreases to 0.1794268266766979. In the iteration process 1285 up to iteration 1399 get an increase from the result of the squared error which is 0.18006584703194672 rising to 0.2807332818651873. So that the second test on epoch 2 in each iteration for the entire iteration of 700 to iteration 1399 gets a decrease / minimization of the squared error. Following figure 7 is a graph to see the results of the second test on Epoch 2:

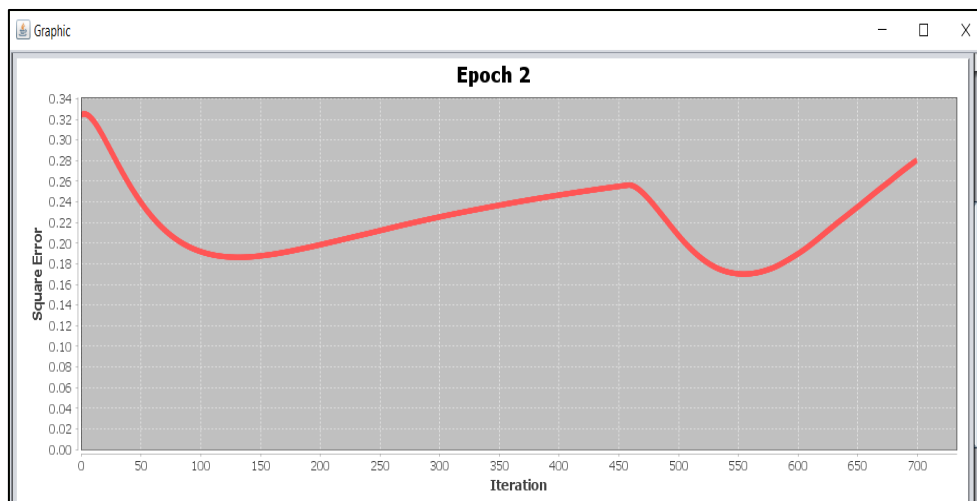


Figure 7. Graph of the second test at Epoch 2

4.3 The Third Test On EPOCH 3

The results of ADAM testing on gradient descent backpropagation for the third test on epoch 3 can be seen in Figure 8:

In the results, it can be seen in the fourth test on epoch 4 in each iteration from iteration 2100 until iteration 2799 gets a decrease / minimization of the square of error at the iteration of 2100 to iteration 2244 which is 0.22048338304646564 decreasing to 0.12194312699269359. At iteration 2245 until iteration 2566 gets an increase from the result of the squared error value of 0.12206380052970295 up to 0.1809430190725432. In the iteration 2567 up to iteration 2693, the decrease / minimization of error squares which is 0.17975220908210351 decreases to 0.10985521531074001. At iteration 2694 until iteration 2799 gets an increase from the result of the squared error value which is 0.11016314136315486 up to 0.16105322504274178. So that the fourth test on Epoch 4 in each iteration for the whole of the iteration 2100 until iteration 2799 gets a decrease / minimization of the squared error. Following figure 11 is a graph to see the results of the fourth test on Epoch 4:

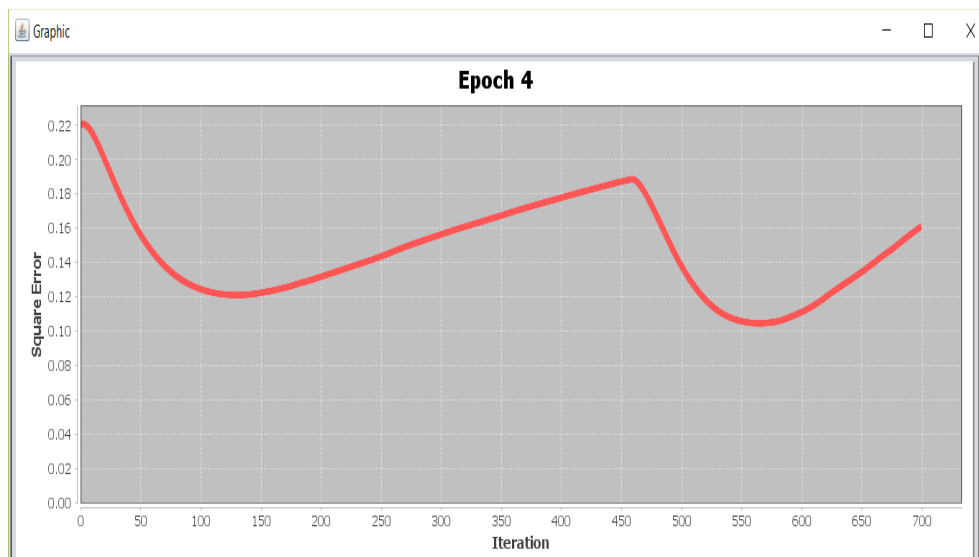


Figure 11. Graph of the fourth test at Epoch 4

4.5 The Fifth Test On EPOCH 4

The results of ADAM testing on gradient descent backpropagation for the fifth test in Epoch 5 can be seen in Figure 12:

Error		Error		Error		Error		Error	
Iteration	Square Error	Iteration	Square Error	Iteration	Square Error	Iteration	Square Error	Iteration	Square Error
2800	0.16072535211362018	2896	0.09396801937057069	3232	0.1442691379218571	3376	0.07877960662109812	3447	0.09571925191938394
2801	0.1602620085798871	2897	0.09381140827715001	3233	0.14443616760849666	3377	0.07883994539542749	3448	0.09602168654484312
2802	0.15987529423574112	2898	0.09366133403496117	3234	0.144599220094541	3378	0.07891650158907507	3449	0.09639207929026863
2803	0.15899522180610107	2899	0.09351659722504381	3235	0.14476900850359972	3379	0.07901542630350233	3450	0.09676390038092043
2804	0.15825040986247815	2900	0.09337907475156931	3236	0.14493310914757815	3380	0.07912448788259825	3451	0.0971230196741686
2805	0.15741926409966617	2901	0.09324819689019885	3237	0.14509921259738978	3381	0.07924187012076049	3452	0.09746995784773183
2806	0.1565397814638923	2902	0.093121422926573	3238	0.14526156096248094	3382	0.07936798646404492	3453	0.09780741397537489
2807	0.15561004774357476	2903	0.09300088552961404	3239	0.14542339515513047	3383	0.07949808936218164	3454	0.09813802071364758
2808	0.15463575285449335	2904	0.09288560946351113	3240	0.14558046593109145	3384	0.07962704728080029	3455	0.09847772346451927
2809	0.15361357109718324	2905	0.09277509165049559	3241	0.14573417132798644	3385	0.0797544708038734	3456	0.09883077439198053
2810	0.15254963626210233	2906	0.09267062305819765	3242	0.145890829973735018	3386	0.079895959606551748	3457	0.0991833928440633
2811	0.15143977164559866	2907	0.09257135741407164	3243	0.14604373498011474	3387	0.08003781290438983	3458	0.09953698099744508
2812	0.15030955070741842	2908	0.09247701270719444	3244	0.14619191572424508	3388	0.08018441155262936	3459	0.0998766898210254
2813	0.14917348312899429	2909	0.09239644376859887	3245	0.14633692161533407	3389	0.08033948628431249	3460	0.10021654703819755
2814	0.1480227714644482	2910	0.09232122856928325	3246	0.14647872589708962	3390	0.08049711667810719	3461	0.10054465618508272
2815	0.1468523204768586	2911	0.0922519684748931	3247	0.1466228673340229	3391	0.0806613217160222	3462	0.10088778392678649
2816	0.1456831911629975	2912	0.09218789762706146	3248	0.14676690831635889	3392	0.08082606836596734	3463	0.10122160953940146
2817	0.14451673464277373	2913	0.09212904106518643	3249	0.14690503577386194	3393	0.08098711652448799	3464	0.10155289271281259
2818	0.14335421475624924	2914	0.09207485846853616	3250	0.14704191134920488	3394	0.081156654677838	3465	0.1018765824154674
2819	0.14219777022030447	2915	0.09202549202931982	3251	0.147176015254518336	3395	0.08132673502142039	3466	0.10220002748943326
2820	0.14104684278119797	2916	0.09198069080207899	3252	0.147314042566398198	3396	0.08149919257720398	3467	0.10252083685848454
2821	0.13989791951938058	2917	0.09194014378993233	3253	0.14745105187372687	3397	0.08167578448428113	3468	0.10283647468023577
2822	0.13876179816986475	2918	0.09190372129864043	3254	0.14758645342022222	3398	0.08186509949456093	3469	0.10313985279226477
2823	0.13761358044345176	2919	0.09187122799799051	3255	0.147720175254518336	3399	0.082051544887549	3470	0.10344819123613878
2824	0.136425708783061	2920	0.09184241006413567	3256	0.1478420471066812	3400	0.08224881823629712	3471	0.10377348300398581
2825	0.1352701834552452	2921	0.091811715768264698	3257	0.147971608054493538	3401	0.08245862306540417	3472	0.1041068193203055
2826	0.13414332482341315	2922	0.09179671798305432	3258	0.14811666476656742403	3402	0.0826860743879824	3473	0.10445959344515436
2827	0.13302510270119433	2923	0.09177985687670151	3259	0.148261025012431426	3403	0.08291584780499577	3474	0.10482518194924555
2828	0.1319333267977108	2924	0.09176622712505544	3260	0.14842327712231848	3404	0.08315003768615933	3475	0.10519337243359539
2829	0.13086938893925562	2925	0.09175567808726096	3261	0.1485741509719454	3405	0.0833944908712816	3476	0.10558371264341589
2830	0.12981783479432932	2926	0.091754646361006411	3262	0.14872645923129925	3406	0.08364812882216577	3477	0.10596390336718908
2831	0.12879827023864886	2927	0.09176877835283961	3263	0.14887582731090282	3407	0.08390523377640251	3478	0.1063313357389915
2832	0.12776269228610884	2928	0.0917833304690534	3264	0.149008064625779	3408	0.08416607323943853	3479	0.10670504897441521
2833	0.12674176537409282	2929	0.0918048007589322	3265	0.1491408841034585523	3409	0.08443389016530843	3480	0.10707161045098942
2834	0.12574020121394736	2930	0.09182958920613213	3266	0.14928358177574185	3410	0.08470898603960222	3481	0.10742355694496714
2835	0.12476033849718289	2931	0.0918599153880748	3267	0.1494372606368374685	3411	0.08497901261166979	3482	0.10777912731871796
2836	0.12378495781859473	2932	0.09188847515481179	3268	0.149595757845566452	3412	0.0852723361651691	3483	0.1081352575811148
2837	0.12282833540379592	2933	0.09192182151934299	3269	0.149763846814036665	3413	0.08558012835897578	3484	0.10848015395445998
2838	0.1218965120721041	2934	0.0919597446264125	3270	0.1499317586693255748	3414	0.0858796180763784	3485	0.10881497499788674
2839	0.12098310313398788	2935	0.09200385351776505	3271	0.150099767755898221	3415	0.08622618293743355	3486	0.10915206448588316
2840	0.12008721277383727	2936	0.09205034151632839	3272	0.15028085084126138	3416	0.08656062309777683	3487	0.109450240636270171
2841	0.11920249177047691	2937	0.09209876651430773	3273	0.15047168998985945	3417	0.086888687031661813	3488	0.109842703658089
2842	0.11833543951921482	2938	0.09216898856495499	3274	0.1506722800288093	3418	0.08722034219438272	3489	0.11018296891661622
2843	0.11748773270014051	2939	0.09223994541055744	3275	0.1508827261055811	3419	0.08753951213098043	3490	0.11051006193222629
2844	0.1166627808654165	2940	0.09231185600064662	3276	0.151100632969642	3420	0.08786102226139759	3491	0.11083130189744089
2845	0.1158659577383026	2941	0.09238496833523423	3277	0.151326425264643163	3421	0.08816940529690206	3492	0.11115648262822801
2846	0.11508950541530864	2942	0.09248441990687495	3278	0.151562801917303278	3422	0.08846762710280799	3493	0.11147652721390346
2847	0.11432938259456621	2943	0.09258802792437641	3279	0.1518120875027049912	3423	0.08876238761627468	3494	0.1118082817986742

Figure 12. The squared error value in the output layer at epoch 5 starts from the iteration 2800 to iteration 3494

In the results, it can be seen in the fifth test on epoch 5 in each iteration from iteration 2800 until iteration 3494 gets a decrease / minimization of the squared error value at iterations 2800 to iteration 2934 which is 0.16072535211362018 decreases to 0.0919597446264125. In the iteration 2935 until iteration 3257 get an increase from the results of the squared error value which is 0.09200385351776505 up to 0.14711608054493538. In the iteration 3258 up to iteration 3378, the decline / minimization of the value of the results of the squared error of 0.14666476656742403 decreased to 0.07891650158907507. In the 3379 iteration up to iteration 3494 get an increase from the results of the squared error value which is 0.07901542630350233 up to 0.1118082817986742. So that the fifth test on Epoch 5 in each iteration for the overall iteration

of 2800 until iteration 3494 gets a decrease / minimization of the squared error. Following figure 13 is a graph to see the results of the fifth test on Epoch 5:

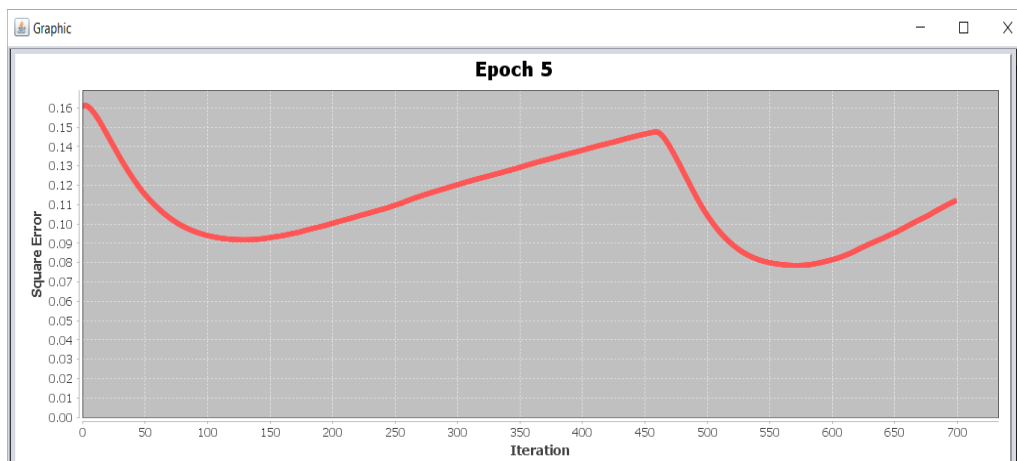


Figure 13. Graph of the fifth test at Epoch 5

From the graph of the first test on Epoch 1 to the fifth test on Epoch 5, can be seen in Figure Graph 14 which is the whole test above. The first test on EPO 1 to Fifth Test on EPO can be seen in Figure 14:

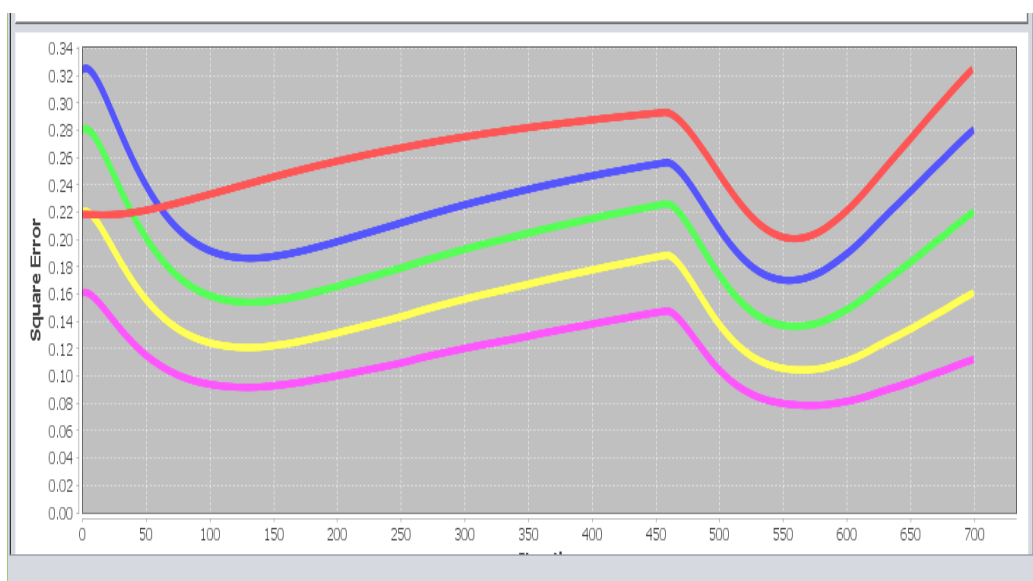


Figure 14. The first test chart until the fifth test in epoch 1 through epoch 5

5. Conclusion

The first test on Epoch 1 is an increase in the value of the squared error. In the second test on Epoch 2 to the fifth test at Epoch 5 there is minimization / decrease of squared error in each epoch test. The results of tests that have been conducted on neural network networks namely ADAM on gradient descent backpropagation can help the learning performance of neural network networks on gradient descent backpropagation to minimize / decrease squared errors. Furthermore, a new method analysis can be carried out for the learning level, so that it is expected that from the first

test on Epoch 1 to the fifth test at Epoch 5 it results in a decrease in the minimization of the squared error.

REFERENCES

- [1] Guan, N., Shan, L., Yang, C., Xu W., & Zhang, M., 2017. Delay Compensated Asynchronous Adam Algorithm for Deep Neural Networks. *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications*, pp. 852 – 859.
- [2] Wakitani, S., Yamamoto, T., & Ishimura, A., 2017. Study on an adaptive GMDH-PID controller using adaptive moment estimation. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1587 – 1591.
- [3] Wu, M., Guo, S., Chen, X., Xing, N., & Zhong, C., 2016. LM–BP based operation quality assessment method for OTN in Smart grid. *Proceedings of the IEEE Network Operations and Management Symposium*.
- [4] Ahmad, F., Isa, N., A., M., Osman M., K., & Hussain, Z., 2010. Performance comparison of gradient descent and Genetic Algorithm based Artificial Neural Networks training. *Proceedings of the IEEE International Conference on Intelligent Systems Design and Applications*, pp. 604 – 609.
- [5] Popa, C-A., 2014. Enhanced Gradient Descent Algorithms for Complex-Valued Neural Networks. *Proceedings of the IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 272 – 279.
- [6] Singh, B., K., Verma, K., & Thoke, A., S., 2015. Adaptive Gradient Descent Backpropagation for Classification of Breast Tumor in Ultrasound Imaging. *Proceedings of the Elsevier International Conference on Information and Communication Technologies* **46**(9): 1601 – 1609.
- [7] Heravi, A., R., & Hodtani, G., A., 2018. A New Correntropy-Based Conjugate Gradient Backpropagation Algorithm for Improving Training in Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* **29**(12) : 6252 – 6263.
- [8] Li, Y., Zhu, L., Zhou, L-j., & Jiang, J., 2011. Study on the BP–GA model and its application in water quality assessment. *Proceedings of the IEEE International Symposium on Water Resource and Environmental Protection*, pp. 2781 – 2784.
- [9] Achkar, R., Geagea, R., Mehio., & Kmeish, W., 2016. SmartCoach personal gym trainer: An Adaptive Modified Backpropagation approach. *Proceedings of the IEEE International Multidisciplinary Conference on Engineering Technology*, pp. 1 – 6.
- [10] Mammadli, S., 2017. Financial time series prediction using artificial neural network based on Levenberg–Marquardt algorithm. *Proceedings of the Elsevier International Conference on Theory and Application of Soft Computing, Computing with Words and Perception* **120**(6): 602 – 607.
- [11] Andayani, U., Nababan, E., B., Siregar, B., Muchtar, M., A., Nasution, T., H., & Siregar, I., 2017. Optimization backpropagation algorithm based on Nguyen–Widrow adaptive weight and adaptive learning rate. *Proceeding of the IEEE International Conference on Industrial Engineering and Application*, pp. 363 – 367.
- [12] Indolia, S., Goswami, A., K., Mishra, S., P., & Asopa, P., 2018. Conceptual Understanding of Convolutional Neural Network – A Deep Learning Approach. *Proceedings of the*

-
- Elsevier International Conference on Computational Intelligence and Data Science* **132**(10): 679 - 688.
- [13] Srinivasan, N., Ravichandran, V., Chan, K., L., Vidhya, J., R., Ramakrishnan, S., & Krishnan, S., M., 2002. Exponentiated backpropagation algorithm for multilayer feedforward neural networks. *Proceedings of the IEEE Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02*, pp. 327 – 331.
- [14] Paulin, F., & Santhakumaran, A., 2010. Back Propagation Neural Network by Comparing Hidden Neuron: Case study on Breast Cancer Diagnosis. *International Journal of Computer Application* **2**(4): 40 – 44.
- [15] Chen, C., C., Kuo, C., Kuo, S., Y., & Chou, Y., H., 2015. Dynamic Normalization BPN for Stock Price Forecasting. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2855 - 2860.