Adaptive Non-Playable Character in RPG Game Using Logarithmic Learning for Generalized Classifier Neural Network (L-GCNN)

Izza Mabruroh*[1], Darlis Herumurti[2]
[1,2]Intsitut Teknologi Sepuluh Nopember Surabaya
mabrurohizza@gmail.com*[1], darlis@its-sby.edu[2]

Abstract

Non-playable Character (NPC) is one of the important characters in the game. An autonomous and adaptive NPC can adjust actions with player actions and environmental conditions. To determine the actions of the NPC, the previous researchers used the Neural Network method but there were weaknesses, namely the action produced was not in accordance with the desired so the accuracy of action was not good. This study overcomes the problem of accuracy of action that is not good in previous studies that use Neural Network to decide on NPC actions by using the Logarithmic Learning method for Generalized Classifier Neural Network (L-GCNN) with 6 input parameters namely NPC health, distance with players, Other NPCs are involved or not, attack power, number of NPCs and NPC levels. In this study, we will discuss the accuracy of L-GCNN in determining the behavior of NPCs so that the NPC gets the optimal decision in attack compared to using other NN methods. While the output is to attack itself, attack in groups and move away. While the output is to attack itself, attack in groups and move away. For testing, this study was tested on RPG games. From the results of the experiments conducted, it shows that the L-GCNN method has better accuracy than the 3 methods compared to 7% better than NN and SVM and 8% better than RBFNN because in the L-GCNN method there is an encapsulation process that is data have the same class will. Whereas the L-GCNN training time is 30% longer than the NN method because on L-GCNN one neuron consists of one data where there are fewer NNs in the hidden layer. So that it has better action accuracy.

Keywords: *L-GCNN, NPC adaptif, aksi NPC*

1. Introduction

Since the emergence of the ideas about artificial intelligence, games are one of the items that helped advance AI research [1]. Games not only cause interesting and complex problems for AI to solve, they also provide land for creativity and expression for users [2]. Thus it can be said that the game is a rare domain where there are elements of art and interaction that make the game unique and favorite for AI studies. But not only is AI progressing through research, games have also advanced through AI research [3].

One of the forming elements of the game is Non-Playable Character (NPC) which is becoming an increasingly challenging domain for artificial intelligence techniques because of its complex nature [4]. One method commonly used to regulate NPC behavior is Finite State Machine where this method is a simple method that is easy to implement, predictable in response, flexible and has light computing. But it has the disadvantage of being a condition for a fixed state transition that is not properly used in games because of its predictable nature [5].

Yunifa et al. Applied fuzzy logic and Hierarchical Finite State Machine (HFSM) to regulate the behavior of NPCs in order to emulate human strategies in war games. The results obtained include HFSM successfully modeling the maneuvering behavior of each NPC and the application of fuzzy to manage the behavior of NPCs to successfully outperform NPCs without fuzzy up to 80% [6]. Furthermore, Supeno et al. Conducted a study on close combat games using a fuzzy coordinator where the rule base was used to regulate the fighting action of an NPC. The coordination action of this research is not enough to produce a variety of actions and sometimes the actions are not as desired [7].

Supria et al. Used L-GCNN to improve the accuracy of the introduction of SIBI sign language. In his research proposed the introduction of the Indonesian Language Signal System (SIBI) by using a combination of static features with dynamic features based on Logarithmic

Receive January 28, 2019; Revise January 29, 2019; Accepted February 02, 2019

Learning for Generalized Classifier Neural Network (L-GCNN), where static features are used for the introduction of static sign language and dynamic features used to recognize dynamic sign language. L-GCNN is used to improve the accuracy of the introduction of sign language. the results of this study are the use of a combination of static features with dynamic features and L-GCNN has an increase in accuracy of 6.67% better than the use of static features [8].

Andreas, using a neural network with 3 layers to control the behavior of AI agents in the RTS (Real Time Strategy) game, ANN was chosen because of its ability to generalize various conditions. This research produces adaptive AI agents that can react to changes in environmental conditions and unpredictable actions. The weakness of this study is that the level of accuracy that is not good seen from the action produced is sometimes not appropriate [9].

Buse Melis, proposed the development of a Neural Network using a radial basis function (RBFNN), namely the GCNN (Generalized Classifier Neural Network) to classify several data sets and proved to have better performance than GRNN and PNN, but this method requires large memory [10]. Previous researchers overcame the shortcomings of the GCNN by adding a logarithmic function (L-GCNN) for optimizing smoothing parameters so as to reduce the number of iterations, reduce training time and get good accuracy compared to previous methods [11].

Based on the explanation above, a new optimization method was tested in this study. this method uses the logaritmic learning for generalized classifier Neural Network (L-GCNN) method to regulate the adaptive behavior of each NPC in games where the L-GCNN method matches the existing literature, is the best method of some NN algorithms because this method has more performance both from the other NN methods resulting in better accuracy.

2. Research Method

This study uses the L-GCNN method to make an active state decision where the L-GCNN method is a development method of the GCNN. The structure of the second layer of the same method differs only from the cost function where GCNN uses error squares while L-GCNN uses the logarithmic function which aims to reduce the time complexity of the GCNN method [11]. The Finite State Machine method for modeling NPC behavior, while the L-GCNN method for making decisions on an active state such as Figure 1. When the game starts, if the player (in this study is considered an enemy of the NPC) has not been seen by the NPC then the NPC will continue searching until the enemy is seen After the enemy is seen, the enemy will be selected and the NPC takes action, namely a single attack, a single attack in grouping or keep distance according to the results of calculations performed by the L-GCNN method by considering the existing input.

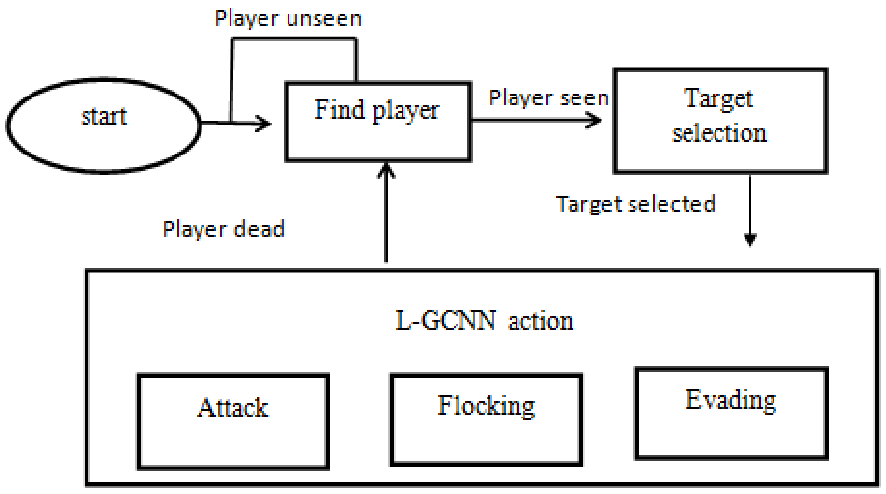

Figure 1. FSM Diagram on NPC

In this study, NPCs can do 3 actions: single attack, keep distance and single attack in grouping.

1. Single attacking: when the agent is in a position and good condition for attacking players
2. Keep distance: the agent moves away from the player and tries to avoid the player's source

3. Attack in group: when an action "attack in group", agents try to stay close to other agents in the area around them then attack in groups.

In the L-GCNN method for this research, several inputs are needed which will be used for the learning process and behavior output that will allow the NPC to defeat the player. Figure 2 is an L-GCNN design consisting of 5 layers, namely input, pattern, summation, normalization and output, where there are 6 inputs and 3 outputs with 1 node at the end of the output layer.

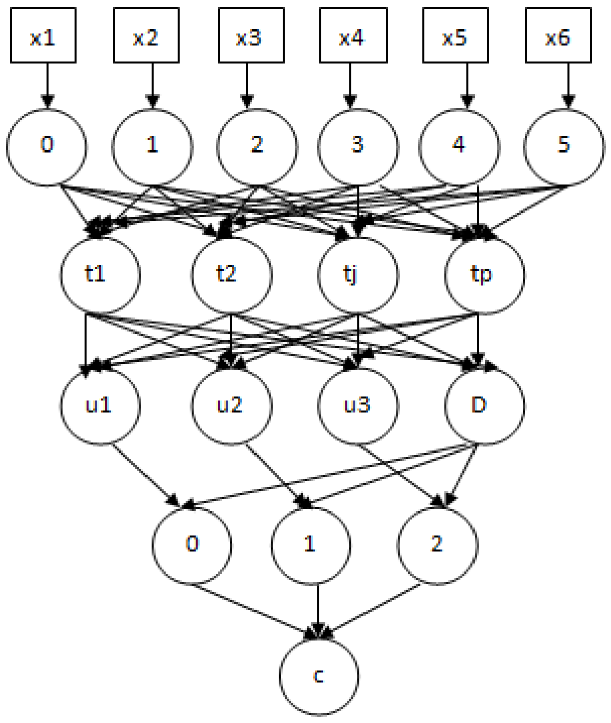

Figure 2. The L-GCNN Structure for NPC is not Different from the GCNN Difference Only in the Calculation of the Cost Function [11]

2.1 Input and Output

The input in this study consisted of 6 inputs:
- Input 0 (x1): NPC health
- Input 1 (x2): distance from the player
- Input 2 (x3): the player is involved with another NPC or not
- Input 3 (x4): NPC attack power
- Input 4 (x5): number of close NPC friends
- Input 5 (x6): NPC level

The output in this study consists of 3 outputs:
- Output 0: single attack / attack
- Output 1: Single attack in group / joint attack (more than 1 NPC)
- Output 2: keep distance

2.2 Layer Calculation

1. **Input layer**
   This layer sends the input vector x to the pattern layer. One neuron represents each training data[10][11].
2. **Pattern layer**
   After the data is received from the input layer, then calculate the Euclidian distances between vector input x and vector training data using Equation 1 where in this layer the number of neurons is equal to the amount of training data [9], [10].

$$dist\ (j) = \|x - t_j\|^2, 1\ \leq j \leq p \quad (1)$$

the output of the pattern layer is determined by using the RBF activation function, the gaussian [9], [10] using Equation 2.

$$r(j) = exp\left(-1 * \frac{dist(j)}{2\sigma^2}\right), 1 \leq j \leq p \tag{2}$$

Then determine whether the data is included in the class or not like Equation 3. The reason for choosing 0.9 and 0.1 is to prevent stuck neuron problems in the learning process. If the data is entered into the destination class neurons then it is given a value of 0.9 if not then it will be given a value of 0.1.[9][10]

$$y(j,i) = \begin{cases} 0.9 \\ 0.1 \end{cases} t_j \; show \; i \; th \; class 1 \leq i \; \leq N \\ else \; 1 \leq j \leq p \tag{3}$$

3. **Summation *layer***

   The summation layer consists of $N + 1$ neurons where N is the total number of classes and 1 neuron is a denominator neuron. At this layer, the effect term calculation is done with Equation 4. Then calculate the Numerator with Equation 5 and Denominator with Equation 6.[10], [11]

$$d\,(j,i) = \; exp^{(y(j,i)-ymax)} * y(j,i) \tag{4}$$

$$u_i \; = \sum\nolimits_{j=1}^{p} d(j,i) * r(j) \tag{5}$$

$$D = \sum\nolimits_{j=1}^{p} r(j) \tag{6}$$

4. **Normalization layer**

   The normalization layer consists of N neurons that represent each class. Each neuron divides the numerator value with the denominator obtained from the previous layer. Like Equation 7 where c_i shows the output normalized from i-th class.[10], [11]

$$c_i = \frac{u_i}{D}, 1 \; \leq i \; \leq N \tag{7}$$

5. ***Output layer***

   Finally, the winner decision mechanism is given by Equation 8, where c is the output vector in the normalization layer, showing the winning value of the neuron and the id showing the winning class.[10], [11]

$$[o, id] = \max\,(c) \tag{8}$$

6. **New smoothing parameter**

   After finding the winner, then calculating the cost function (e) using Equation 9, L-GCNN uses logarithmic. $y(z,id)$ shows the z training input data value and c_{id} shows the winner class, while the GCNN uses a squared error Then the smoothing parameter is updated with Equations 10 - 14.[10], [11]

$$e = \; (y(z,id) * \log(c_{id})) + \; (1 - y(z,id)) * log(1 - \; c_{id}) \tag{9}$$

$$\sigma_{new} \; = \; \sigma_{old} + lr * \frac{\partial e}{\partial \sigma} \tag{10}$$

$$\frac{\partial e}{\partial \sigma} = y(z,id)\left(\frac{\frac{\partial c_{id}}{\partial \sigma}}{c_{id}}\right) + (1 - y(z,id)) * \left(\frac{\frac{-\partial c_{id}}{\partial \sigma}}{c_{id}}\right) \quad (11)$$

$$\frac{\partial c_{id}}{\partial \sigma} = \frac{b(id) - l(id) * c_{id}}{D} \quad (12)$$

$$b(id) = 2 * \sum_{j=1}^{p} d(j,id) * r(j) * \frac{dist(j)}{\sigma^3} \quad (13)$$

$$l(id) = 2 * \sum_{j=1}^{p} r(j) * \frac{dist\ (j)}{\sigma^3} \quad (14)$$

3. Result and Discussion.

This research was tested on RPG type games developed using the Unity application. The parameters tested were the accuracy of the action where when the accuracy was better the NPC action was produced accordingly. The data set used is 400 data consisting of 6 parameters, namely the health of the NPC range of 0-100 values, the distance of NPCs with player ranges from 0-30, are players involved with other NPCs? With a value of 0/1, NPC attack power ranges from 0-25, the number of NPC friends ranges from 0-4 and the NPC level ranges from 1-10. 400 of these data obtained from random results using excel which is then scaled to a value between 0 and 1 to determine the action according to the predetermined rule base (see Table 1 or the sample data).

Table 1. Data Set Sample

Data	Health	Distance	Agent involved?	Attack	Number of Allies	Level	Action
	P1	P2	P3	P4	P5	P6	Y
1	0.81	0.4	0	0	0.25	0.6	0
2	0	0.533333	1	0.04	0.75	0.2	1
3	0.15	0.5	1	0.32	0	0.9	2
4	0.47	0.633333	0	0.96	1	0.4	1

The first thing to do is to test the Neural Network method with a multilayer perceptron structure using the backpropagation algorithm which is then compared with the proposed method by looking at training time and level of accuracy. Figure 3 shows an example of the results of a single attack action obtained when p1 = 1, p2 = 0.95, p3 = 1, p4 = 0.08, p5 = 0, p6 = 1 when testing. Then Figure 4 is an example of the keep the distance action produced when P1 = 0.53, P2 = 0.57, P3 = 1, P4 = 0.08, P5 = 0 and P6 = 1. Whereas Figure 5 is an example of an attack in group action produced when P1 = 0.53, P2 = 0.57, P3 = 1, P4 = 0.08, P5 = 0 and P6 = 1.

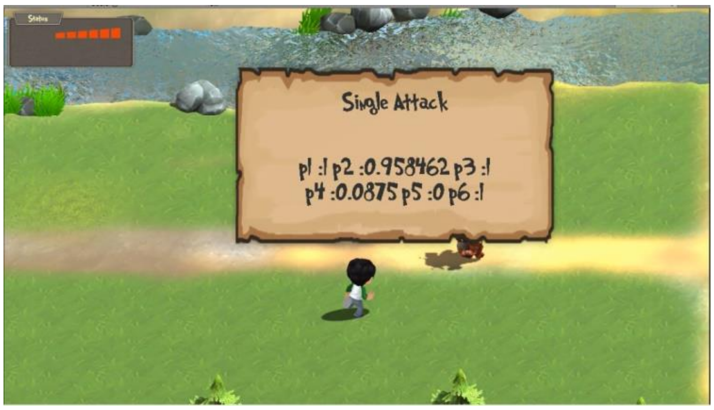

Figure 3. Single Attack Action

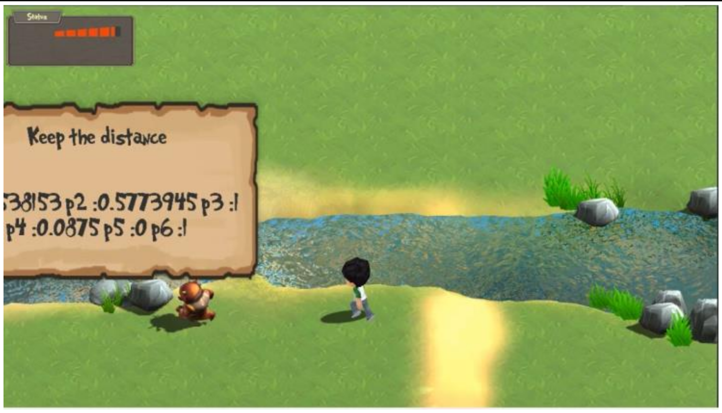

Figure 4. Keep the Distance Action

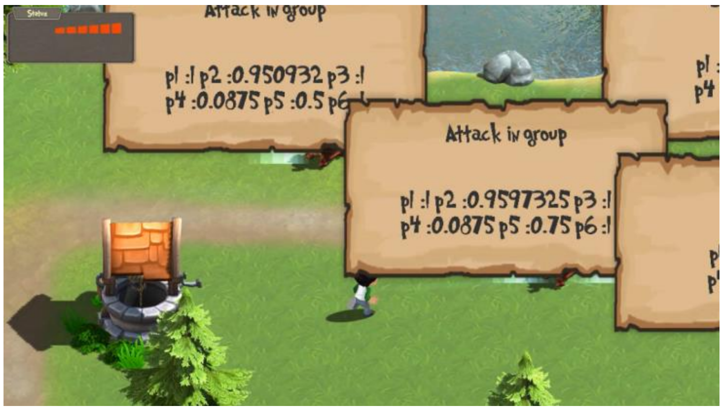

Figure 5. Attack in Group Action

3.1 Comparison of L-GCNN and NN training times

Table 2. Comparison of Training Time

Method	*Training Time*
L-GCNN	6.356 second
NN	2.36 second

The first test was conducted by comparing training time between the L-GCNN and NN methods [1]. Table 2 show the test results where NN training time is 33% faster than L-GCNN because L-GCNN has one neuron for each training data in the hidden layer where NN have fewer neurons in the hidden layer.

3.2 Comparison of accuration between L-GCNN, NN, RBFNN and SVM

The next test is comparing the L-GCNN method with NN, RBFNN and SVM by looking at the level of accuracy in each test scenario, namely ten-fold cross validation, 50% training data 50% data testing, 60% training data 40% data testing, 70% training data 30% testing data and 80% training data 20% testing data.

Figure 6 shows the results of the comparison of accuracy from the L-GCNN and NN methods where for 10-fold cross validation L-GCNN has an accuracy of 93% and NN 84%. Percentage of 50% split produces 90.5% accuracy value for L-GCNN and 85% for NN. Percentage of 60% split produces 90% accuracy value for L-GCNN and 83.75% for NN. Percentage of 70% split resulted in an accuracy value of 92.5% for L-GCNN and 84.1% for NN while the percentage of split 80% resulted in an 91% accuracy value for L-GCNN and 83.75% for NN.

Figure 7 shows the results of the accuracy comparison of the L-GCNN method and RBFNN where the results of the accuracy comparison of the L-GCNN and RRBFNN methods where for the 10-fold cross validation L-GCNN has an accuracy of 93% and RBFNN 84.2%. Percentage of 50% split results in an accuracy value of 90.5% for L-GCNN and 83.5% for RBFNN. Percentage of 60% split produces 90% accuracy value for L-GCNN and 83.75% for RBFNN. The percentage of 70% split resulted in an accuracy value of 92.5% for L-GCNN and 80% for RBFNN while the percentage of split 80% resulted in an 91% accuracy value for L-GCNN and 78.75% for RBFNN.

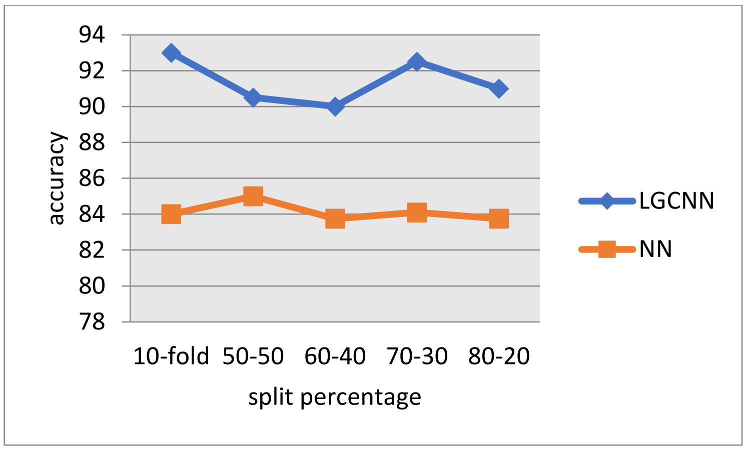

Figure 6. Comparison of Accuracy Results of L-GCNN and NN

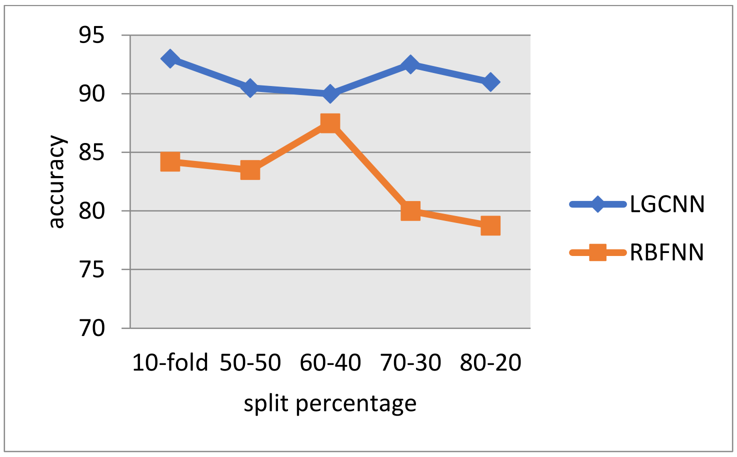

Figure 7. Comparison of Accuracy Results of L-GCNN and RBFNN

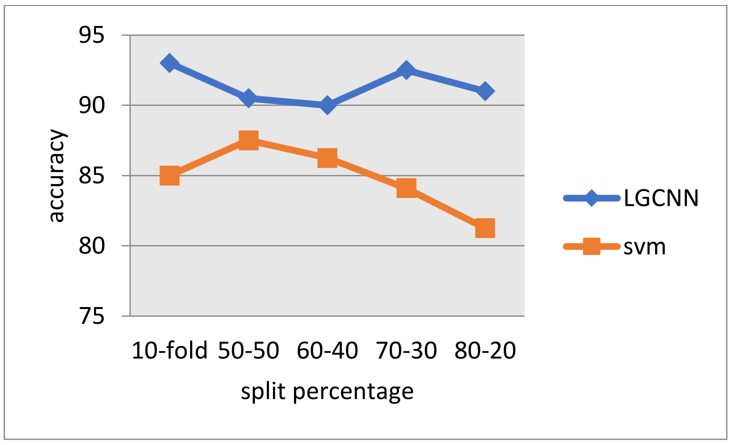

Figure 8. Comparison of Accuracy Results of L-GCNN and SVM

Figure 8 shows the results of the accuracy comparison of the L-GCNN method and SVM where the results of the accuracy comparison of the L-GCNN and SVM methods where for the 10-fold cross validation L-GCNN has an accuracy of 93% and SVM 85%. Percentage of 50% split results in an accuracy value of 90.5% for L-GCNN and 87.5% for SVM. Percentage of 60% split produces 90% accuracy value for L-GCNN and 86.25% for SVM. The percentage of 70% split resulted in an accuracy value of 92.5% for L-GCNN and 84.1% for SVM while the percentage of split 80% resulted in an 91% accuracy value for L-GCNN and 81.25% for SVM

L-GCNN has better accuracy than NN because in the L-GCNN process there is a data encapsulation process on the summation layer. If the j-data has the same class, it will be grouped according to class.

4. Conclusion

After testing, the results showed that the L-GCNN method had better accuracy than the 3 methods compared with an average accuracy rate of 7% better than NN and SVM and 8% better than RBFNN. From the data, it can be concluded that the use of smoothing parameters on L-GCNN which shows the radius with the most effective neighbors can increase the accuracy of the method. If the data is included in the same class, it can be grouped according to class so that the L-GCNN has better accuracy.

From the results of the experiments carried out, sometimes the trial data has an error in the form of an infinity cost function value or in the form of NaN, this is due to the wrong selection of initial smoothing parameters and improper learning rates. So the researcher gave advice to the next researcher to optimize the value of the initial smoothing parameters and learning rate.

Notation

$dist\ (j)$	=	Euclidian distance
x	=	vector input x
t_j	=	vector training data
p	=	*Number of training data*
σ	=	smoothing parameter
$y(j,i)$	=	*Y value* for j-th training data and i-th class
$d\ (j,i)$	=	diverge effect term
$ymax$	=	*y* maximum value
u_i	=	numerator for each class
$r(j)$	=	RBF Activation function for each *training data*
D	=	Denominator
c_i	=	*Normalization output*
$y(z,id)$	=	*input* value from z-th *training* data
c_{id}	=	Winner class
e	=	cost function
$\partial\sigma$	=	delta smoothing parameter
∂e	=	delta cost function
$d(j,id)$	=	diverge effect term value for winner class

References

[1] R. E. Leigh, T. Morelli, S. J. Louis, M. Nicolescu, and C. Miles, “Finding Attack Strategies for Predator Swarms Using Genetic Algorithms,” *2005 IEEE Congr. Evol. Comput.*, Vol. 3, Pp. 2422–2428, 2005.

[2] F. Nugroho, S. Mardi, and M. Hariadi, “Simulasi Permasalahan Multiobyektif Berbasis Agen Pada Kasus Economic dan Emission Dispatch (EED) Dengan Metode Neuro Fuzzy System di Power Plant,” Pp. 1–8, 2011.

[3] A. Nareyek, “Intelligent Agents for Computer Games,” *Comput. Games Second Int. Conf. CG 2000*, 2002.

[4] W. Jatiningsih, E. Yuniarno, and M. Hariadi, “Autonomous Agent Based NPC Swarm Attack Behaviour Using Bee Colony Algorithm,” No. Ii, Pp. 1–5, 2014.

[5] S. Asmiatun, L. Hermawan, and T. Daryatni, “Strategi Menyerang Jarak Dekat Menggunakan Klasifikasi Bayesian Pada NPC (Non Player Character),” Vol. 2013, No. November, Pp. 351–357, 2013.

[6] S. Mardi, S. Nugroho, Y. M. Arif, M. Hariadi, and M. H. Purnomo, “Perilaku Taktis Untuk Non - Player Characters Di Game Peperangan Meniru Strategi Manusia Menggunakan Fuzzy,” Vol. 6, No. 1, Pp. 55–64.

[7] P. N. Jember, “Fuzzy Coordinator Based Intelligent Agents For Team Coordination Behavior In Close,” Vol. 51, No. 2, Pp. 317–323, 2013.
[8] Supria, D. H. Murti, and W. N. Khotimah, “Pengenalan sistem isyarat bahasa indonesia menggunakan kombinasi fitur statis dan fitur dinamis lmc berbasis l-gcnn,” *J. Ilm. Teknol. Inf.*, Vol. 14, No. 2, Pp. 217–230, 2016.
[9] “Creating Adaptive Game AI in a Real Time Continuous Environment using Neural Networks,” No. March, 2009.
[10] B. Melis and M. Avci, “Generalized classifier neural network,” *Neural Networks*, Vol. 39, Pp. 18–26, 2013.
[11] B. Melis and M. Avci, “Logarithmic learning for generalized classifier neural network,” *Neural Networks*, Vol. 60, Pp. 133–140, 2014.