ASEAN Engineering Journal

Full Paper

IMPROVING A DEEP NEURAL NETWORK GENERATIVE-BASED CHATBOT MODEL

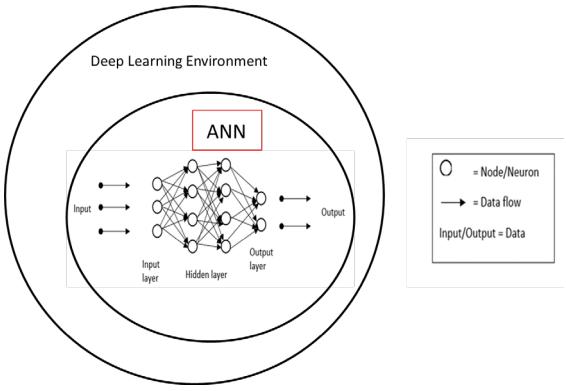
Wan Solehah Wan Ahmad*, Mohamad Nazim Jambli

Faculty of Computer Science and Information Technology, Universiti Malaysia Sarawak, 14300, Kota Samarahan, Sarawak, Malaysia

Article history
Received
01 July 2023
Received in revised form
25 September2023
Accepted
14 November 2023
Published online
31 May 2024

*Corresponding author
19020167@siswa.unimas.my

Graphical abstract



Abstract

A chatbot is an application that is developed in the field of machine learning, which has become a hot topic of research in recent years. The majority of today's chatbots integrate the Artificial Neural Network (ANN) approach with a Deep Learning environment, which results in a new generation chatbot known as a Generative-Based Chatbot. The current chatbot application mostly fails to recognize the optimum capacity of the network environment due to its complex nature resulting in low accuracy and loss rate. In this paper, we aim to conduct an experiment in evaluating the performance of chatbot model when manipulating the selected hyperparameters that can greatly contribute to the well-performed model without modifying any major structures and algorithms in the model. The experiment involves training two models, which are the Attentive Sequence-to-Sequence model (baseline model), and Attentive Seq2Sequence with Hyperparametric Optimization. The result was observed by training two models on Cornell Movie-Dialogue Corpus, run by using 10 epochs. The comparison shows that after optimization, the model's accuracy and loss rate were 87% and 0.51%, respectively, compared to the results before optimizing the network (79% accuracy and 1.05% loss).

Keywords: Deep learning, Artificial Neural Network, Generative-based chatbot, hyperparameter optimization, Attentive Sequence-to-Sequence

1.0 INTRODUCTION

In 2016, 44% of consumers claimed they would rather interact with a chatbot than a human customer service agent [1]; and the percentages are projected to continue to rise in the future years. Every industry must design a solution that uses a third-party software to automate practically everything. Chatbot application naturally fit in with the characteristic of the industry demand. A chatbot is an application that is developed in the field of machine learning, which has become a hot topic of research in recent years. It's a Human-Computer Interaction (HCI) model and an artificial intelligence application that simulates a human-computer conversation [2]. Users' intent, input-output processing, and a response generating technique that obtains information from the knowledge base are all part of the application. Figure 1 depict the general architecture of the chatbot application from upper layer view.

The majority of today's chatbots integrate the Artificial Neural Network (ANN) approach with a Deep Learning environment. The combination of ANN and Deep Learning areas, as well as Natural Language Processing (NLP) techniques, results in a new generation chatbot known as a Generative-Based Chatbot. Generative-based methods leverage natural language generation (NLG) techniques to respond to a message [3][4]. However, to build a Deep Neural Network (DNN) Generative-Based chatbot requires a complex adaption of training network that works in tandem with optimal hyperparameter. As the environment can vary, how the hyperparameter reacts with the environment will also set a different optimum value. The current chatbot application mostly fails to recognize the optimum capacity of the network

environment due to its complex nature resulting a low accuracy performance.

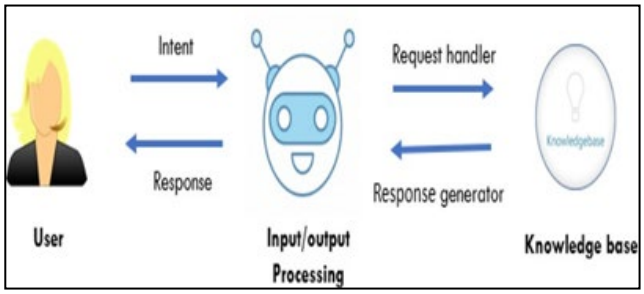


Figure 1 Chatbot's General Architecture

This is because chatbot continues to rely significantly on the algorithm's predefined rules [5]; while forcing the learning from a bad environment.

Having an optimal deep learning environment is essential in constructing a neural network chatbot because it provides a link between the neural response and stimuli in the neural networks. Hyperparameters plays a major role in the environment by influence the structure and training of networks [6]. Before training, each hyperparameter must be set, with the variables' optimal values being the most critical.

Thus, building an intelligent chatbot that can operate efficiently in response to user demands, is fairly difficult as it not only requires contextual understanding, text entailment and language understanding [7]; but also that can operate in an optimal training's environment.

Put aside the enhancement that can only be done with extensive modification and research on the model, this paper aim to conduct an experiment in evaluating the performance of model when manipulating the selected hyperparameters that modifying any major structures and algorithms in the model. Precisely, the paper will focus on a specific neural network chatbot model known as Attentive Sequence-to-Sequence (Seq2Seq) and for training, a dataset from the Cornell Movie-Dialog Corpus will be deployed, which contains information extracted from movie and television program screenplays. The evaluation will be done by measuring the loss and the accuracy value whether there is an improvement or not compared to the based model.

This paper is structured as follows: The next section discusses related work. Section 3 give an overview of the experimental setup. Section 4 lay out the results and analysis. The last section concludes the paper.

2.0 RELATED WORK

In this section, a detail of literature works based on two major components involved in the experiment of studies is explained.

2.1 DNN Generative-Based Chatbot

Before going further, a component that involves in Generative-Based chatbot namely incorporated as a Deep Neural Network Chatbot will be explained. The first crucial component is the involvement of Artificial Neural Network (ANN).

ANNs are Machine Learning models that seek to emulate the operation of the human brain [8]; which is made up of a vast number of interconnected neurons. The input layer, hidden layer, and output layer are the three layers that make up an ANN. Meanwhile, there are numerous nodes or neurons in each layer. The outputs from the preceding levels are used as inputs in each layer. In other words, the nodes transport the input data along the interconnecting layer.

The next component is Deep Learning environment configuration. Deep Learning is linked to feature modification and extraction [9]; in order to create a relationship between stimuli and the neural responses of the network. To put it another way, deep learning serves as an environment that aids neural networks in completing their job which is portrayed as Figure 2. Deep learning indicates that a large number of layers will be used to extract higher-level characteristics from the data provided to the neural network in order to improve the neural models' performance [10].

In chatbot applications, a Recurrent Neural Network (RNN) is adopted to execute a sequential job such as natural language [11]. The current input data in sequential tasks typically depend on the previously applied inputs. As a result, RNN is established, which is tasked to evaluate the relationship between the current input and earlier inputs via a Hidden Layer. A hidden state is a feature within the hidden layer that is used to remember information about a sequence. RNNs may handle any length of information sequence, offering equal weight to phrase processing and chatbot applications. However, RNN has a long-term dependency problem and a vanishing gradient issue [12][13].

A vanishing gradient might be argued is one of the most critical challenges in neural networks, limiting RNN capabilities. To tackle this, a novel network dubbed as Long-Short Term Memory (LSTM) is proposed, which is purposely meant to store information for extended periods of time and overcomes the vanishing gradient problem in RNN networks. The purpose of LSTM is to retain a cell in a condition that permits information to flow via regulated structures known as "gates" [14].

By this, the development of LSTM has prompted the construction of the Sequence-to-Sequence (Seq2Seq) model. A Seq2Seq model is a problem setting in which the input and output are both sequences. It facilitates the translation of an input sentence (in sequence form) into a new sequence (target), both of which can be of any length [15]. Both the encoder and the decoder are effectively two neural networks merged into an RNN adopting LSTM modules.

Despite of this, Seq2Seq model that is made up from fixed-length context vector become a significant drawback for the model especially in dealing with the long sentences. To address this problem, an attention mechanism was introduced. Attention mechanism can be described as the relevance weights of attention processes that can be represented as a vector (attention). The significance weights are used to anticipate the word or element that will appear in a phrase or picture.

There are multiple crucial processes in implementing Attention, but the Context Vector calculation is the most important. When the alignment scores between the previous decoder hidden state and encoder hidden state of each input sequence are generated in a Seq2Seq model, the scores are aggregated and turned into a single vector, which is then executed in Softmax function.

The overview formula of attention mechanism is calculated as shown as:

$$score(s_t, h_i) = v^T tanh(W_a[s_t; h_i])$$

where,

s_t = decoder hidden state
h_i = hidden state
v_a and W_a = alignment weight matrices
T = position of output word
tanh = non-linear activation function

The alignment scores and the encoder hidden state are multiplied to form the context vector in this procedure [19]. The preceding decoder output is then concatenated with the context vector to decode the output. Meanwhile, the output is fed into the Decoder RNN according to the time step, along with the previous decoder hidden states, to produce the new output. The procedure of earlier steps repeats itself for each time step until the output is produced at a predetermined maximum length.

Although attention mechanism can improve the overall performance of generative-based chatbot model, there is still room for improvement where enhancement can be made within the algorithm itself or by enhancing the environment that closely working in tandem with the algorithm.

In this paper, the enhancement will be done with the environment of the model which is involving the hyperparameter in the neural network.

2.2 Hyperparameter Selection

It is critical to understand which hyperparameters to employ during model training. The hyperparameters are the parameters defined before the start of training. The hyperparameters employed determine the length of time required to train and test a model. Thus, it is critical to set the hyperparameters to ensure that the training environment for the neural network model has the maximum capacity [16]. This is because even little changes in the values of hyperparameters can have a significant effect on the model's performance.

For the scope of the experiment, this paper will address the Optimizer, Learning Rate, and Dropout parameters as the parameters that selected under Loss Function hyperparameter.

The Loss Function is a critical hyperparameter in a neural network since it has a significant impact on the model's performance. Adjusting it to the appropriate environment will result in an increase in the efficiency with which the network performs its job [16]. To summarize, hyperparameter optimization seeks to find the ideal tuple that minimizes a predetermined loss function on a given collection of data.

Similarly, in order to accomplish the primary research aim, hyperparameter tuning is viewed as a critical step in optimizing the chatbot model's training environment. Hence, picking high-impact parameters would result in a major outcome for the training.

To clarify, the loss function is a technique for measuring the algorithm's success in modelling the dataset by computing the gradients associated with the neural network's error prediction. In terms of error prediction, the loss function calculates the error for a single training sample by comparing the predicted output to the intended output [17]. While this is occurring, gradients are leveraged to adjust the weights to suit the training data. Thus, it is acceptable to state that the loss function's sole purpose is to quantify the model's performance on a single training set.

In the case of optimizers, the implementation is required since the optimizer and loss function operate in tandem for the algorithm's fit to the data optimally. One of the cases considers the Mean Squared Error (MSE) in line as the most fundamental loss function. The MSE is determined across all inputs for each set of weights attempted in the models by squaring the predicted and actual output [18]; and then averaging it across the whole dataset. The model then optimizes the MSE functions, reducing them to the minimum value achievable with the use of an optimizer.

There are numerous optimizers that can be used in neural network training, including Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent (Mini-batch Gradient Descent), Root Mean Square Propagation (RMSProp), Adaptive Gradient Algorithm (AdaGrad), Adaptive Learning Rate Method (AdaDelta), and Adaptive Moment Estimation (Adam). Only three optimizers will be addressed in this case study for the sake of specificity that includes AdaGrad, RMSProp, and Adam.

AdaGrad optimizer employs an adaptive gradient algorithm that has a unique learning rate for each parameter in the model.

The strength of AdaGrad is that it will prioritise the parameters based on the frequency with which they are changed. The most often encountered will be educated at a slow
pace of learning. Meanwhile, parameters that are changed seldom are taught at a high learning rate [19]. However, the issue arose when the model's most frequently updated parameters began to skew it. The downside of AdaGrad is that after a few batches, the learning rate decreases significantly.

RMSProp was developed to address the AdaGrad problem specifically to exponentially decrease the learning rate. RMSProp updates by utilising the prior learning rate [20]. The concept of momentum may be illustrated by the movement of a ball rolling down a hill (the state of the neural network) (the graph of the loss function), where the longer the ball travels in a certain direction, the quicker it travels due to its high momentum. The momentum is necessary to ensure that the network can cross through local minima without being caught in them.

Adam is practically identical to RMSProp, the method for updating using prior learning rates, except that it additionally uses prior gradients to accelerate learning. Adam essentially moves with enormous force through the network to ensure that it does not come to a halt unexpectedly [21].

Therefore, it is not odd to claim that the optimizer is the engine that drives neural network training. It will assist the machine in acquiring knowledge about it. That is why the optimizer is included in the list of critical neural network hyperparameters. Optimizers function by applying the gradient to a neural network, and the majority of them would automatically determine the learning rate.

However, for the sake of optimum optimization, the learning rate will be examined separately to determine whether it can be optimized independently of the optimizer, hence indirectly enhancing model accuracy. It is because learning rate has the ability to determine how fast or slowly a neural network model

learns a problem [22]; which has become a critical component in why learning rate has been picked as a high impact parameter.

With regards to the dropout, it is a regularization approach used to prevent complicated co-adaptations of training data in deep learning neural networks [23]. A common technique for reducing overfitting is to construct neural networks with a variety of model configurations. However, this strategy is costly because of the training and maintenance of many models. As a result, training a single model is far more optimal.

While a single model may be trained using a variety of different network designs, it can rapidly overfit a training dataset. To mitigate this, the dropout strategy will be deployed to eliminate nodes randomly during training.

Based on the fundamental of all the chosen hyperparameter, experiments will be done by testing different values that works well with each hyperparameter according to their initial configurations to improve the model performance.

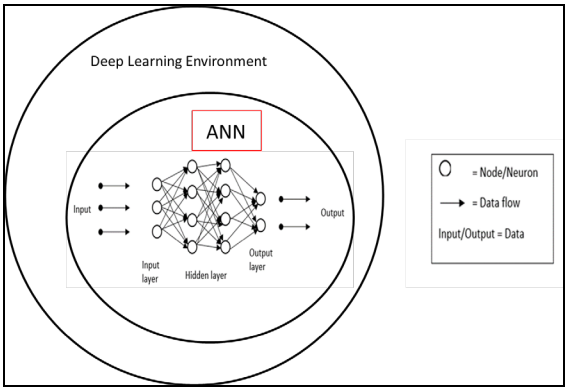


Figure 2 Set of DNN

3.0 EXPERIMENTAL SETUP

This section describes the pre-setup of the configurations in the model that is going to be trained with and without hyperparameter optimization. Every hyperparameter has its unique configurations.

3.1 Optimizer Experimental Setup

The experimental method for assessing the best optimizer for chatbot models begins with the identification of three common optimizers that have been demonstrated to perform optimally in their unique configurations. Three optimization algorithms will be evaluated: AdaGrad, RMSProp, and Adam. Each optimizer is preconfigured with the default value for each hyperparameter that is shown in Table 1.

By using the optimizers as the manipulative variables, the experiment is then done based on the model configuration described in Table 2.

Table 1 Default's Optimizer Configuration

Optimizer	Configuration Parameters		
ADAGRAD	Learning Rate = 0.001	Initial Accumulator = 0.1	EPSILON = 1E-7
RMSPROP	Learning Rate = 0.001	Rho = 0.1	EPSILON = 1E-7
ADAM	LEARNING RATE = 0.001	BETA = 0.9 & 0.999	EPSILON = 1E-7

Table 2 Optimizers Experimental Configuration

Model Configuration	*Value*
Batch Size	*128*
Percentage of validation split	*0.1%*
Dropout	*0.05*
Epoch	*10*
Encoder Type	*Bi-directional*

3.2 Learning Rate Experimental Setup

The preceding section explored the various types of optimizers and demonstrated that Adam outperforms the others. As seen in the experiment, each optimizer has its unique configuration and, interestingly, each of them has a learning rate parameter. The optimizers have already set the learning rate to the default setting, which is regarded to be fairly effective in terms of model performance.

The learning rate will be changed to maximize the optimizer in this experiment to see if improving the optimizer's performance also improves the model's accuracy. To assess the learning rate's performance variation, numbers will be put on a logarithmic scale ranging from 0.0001 for the lower bound to 1.0 for the upper bound. Except for the learning rate, the model setup is fixed at the values shown in Table 3.

Table 3 Learning Rate Experimental Configuration

Model Configuration	Value
Learning Rate	0.0001, 0.001, 0.01, 0.1, 1.0
Optimizer	ADAM
Percentage Of Validation Split	0.1%
Batch Size	128
Dropout	0.05
Epoch	10
Encoder Type	Bi-Directional

3.3 Dropout Experimental Setup

In a neural network, dropout is implemented per layer. The dropout is used in this work in recurrent layers, which include hidden layers as well as the visible or input layer. Because of the dropout, the network's weights will be higher than usual. As a result, before completing the network, the weight will be scaled by the dropout rate first, lowering the error rate. In a recurrent neural network (RNN), however, a dropout is only applied to feed-forward connections.

In a neural network, a value of 1.0 indicates no dropout while a value of 0.0 indicates no output from the layer. The dropout will be set on a scale of 0.05 for the lower bound to 0.1 for the upper bound for the experiment. Except for dropout, the model setup is fixed into the values stated in Table 4.

Table 4 Dropout Experimental Configuration

Model Configuration	Value
Dropout	0.05, 0.06, 0.07, 0.08, 0.09, 0.1
Learning Rate	0.01
Optimizer	ADAM
Percentage Of Validation Split	0.1%
Batch Size	128
Epoch	10
Encoder Type	Bi-Directional

4.0 RESULT AND ANALYSIS

4.1 Fine Tuning Optimizers

Figure 3 depicts the comparison result for accuracy performance of optimizer types as a line graph with the y-axis representing the accuracy rate and the x-axis representing the number of epochs.

Based on results, Adam optimizer achieved the highest training accuracy. Due to the small difference in accuracy performance between Adam and RMSprop optimizers, the loss rate value is also measured to choose the optimal optimizer. To reiterate, the lower the value of the loss rate, the better the performance.

According to Figure 4 of the bar chart comparing the accuracy and loss rates of the three optimizers, Adam optimizer has the lowest loss rate value when compared to the other optimizers trained throughout the experiment.

A good neural network model should optimize loss minimization, which means that it should be as low as possible. Thus, an optimizer should be capable of causing model learning to converge enough in terms of loss reduction when the model is capable of handling sparse gradients on noisy tasks. Additionally, in order to determine the optimal optimizer for the model, the optimizer must meet the following criteria: it must be capable of providing a high accuracy value and a low loss rate value.

Thus, by considering both the accuracy and loss rate values, the Adam optimizer meets both requirements and therefore performs better for the model than RMSprop.

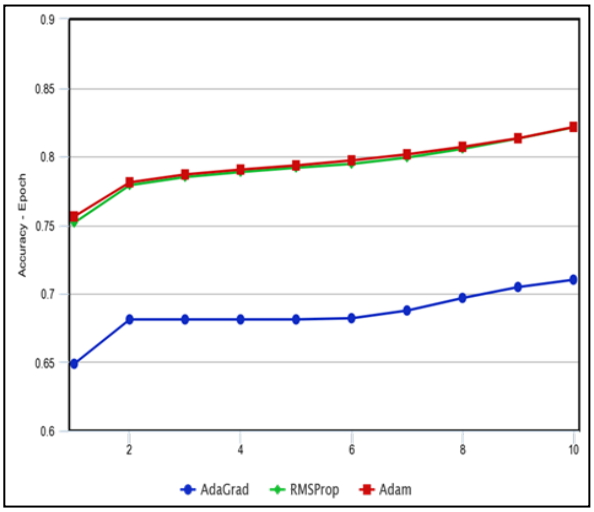


Figure 3 Accuracy Performance of Optimizer Types

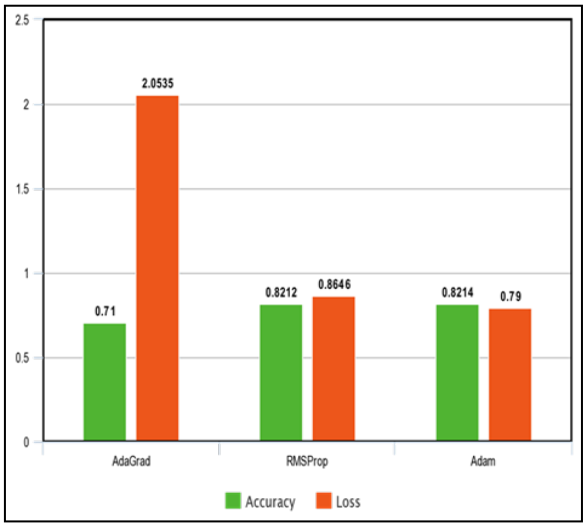


Figure 4 Model Performance According to Optimizer Types

4.2 Fine Tuning Learning Rate

The results will be five graphs indicating the model's accuracy for various values of the learning rate. Figure 5 depicts the comparison result as a line graph with the y-axis representing the accuracy rate and the x-axis representing the number of epochs. number of epochs.

The figure depicts the learning rate's behaviors as it relates to the model's learning. The highest value (1.0) indicates an unpredictable behavior that is rapidly convergent, rendering the model incapable of learning and resulting in the lowest accuracy rate. Meanwhile, a learning rate of (0.1) rendered the model incapable of learning effectively. The straight line on the line graph of values (0.1 1.0) indicates that the model has been stuck in the learning process.

To emphasize the point, when the learning rate approaches convergence rapidly (gradient was updated too often), model distortion occurs, impairing the model's capacity to overcome

local minima. The continuous line shows that the gradient is altered seldom or not at all.

After removing the worst value, the learning rates of 0.0001, 0.001, and 0.01 provide stable graphs that demonstrate typical learning activity. However, a learning rate of 0.01 can be found to function well with the model, as it can effectively understand the issue, producing a final accuracy of 85 %. Thus, the value of the moderate learning rate is set at 0.01 as the optimal value for use with the Adam optimizer in the final experiment of optimizing the chatbot model.

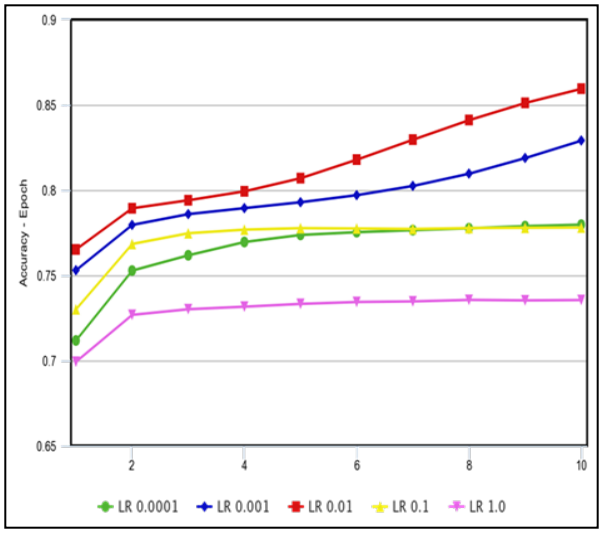


Figure 5 Accuracy Performance of the Value Learning Rate Used

4.3 Fine Tuning Dropout

The results are visualized as below in Figure 6 with the y-axis indicating the accuracy rate and the x-axis showing the number of epochs while Figure 7 illustrated the comparison of the loss rate value after applying these dropout values as y-axis that indicates the loss rate, and the x-axis showing the number of epochs.

The graph in Figure 7 depicts the loss curve (error rate) as a function of the dropout rate. The accuracy rate was observed at 86% and 87%, respectively, after the dropout successfully reduced the error rate by extending from overfitting that led to maximum regularization.

Beyond the ideal dropout threshold, the upper bound value of 0.1 is too large for the model, resulting in a greater error rate. Meanwhile, the lower bound's value of 0.05 slows convergence and prevents the model from fitting properly. Finally, the dropout value of 0.07 is the most appropriate for use in the final experiment.

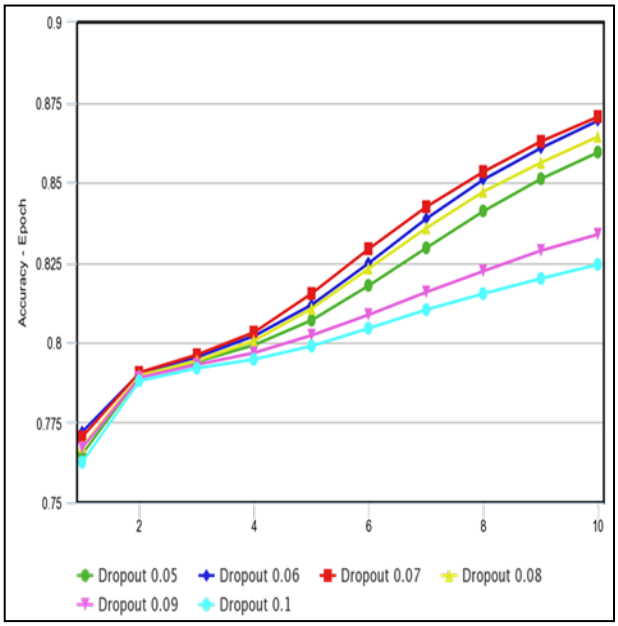


Figure 6 Accuracy Performance of Dropout Value Used

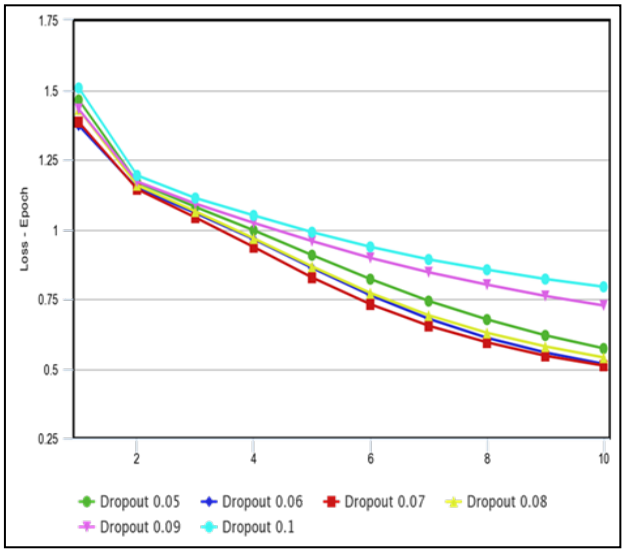


Figure 7 Loss Rate According to Dropout Values Used

5.0 CONCLUSION

To conclude the findings, a final experiment is conducted to prove the improvement that has been made. All of the final findings recorded from the previous experiment will be used from here on. The objective of the experiment is to finalize the theory and actual findings into one conclusion through data representation that will be illustrated from the result of the model's training.

Table 5 Final Chatbot Performance Experimental Configuration

Hyper- parameter	Baseline	Range Value		Optimal Value
		Lower bound	Upper bound	
Optimizer	Adam	-		Adam
Learning rate	0.001	0.0001	1.0	0.01
Dropout	0.05	0.05	0.1	0.07
Encoder Type	Uni-directional	-		Bi-directional

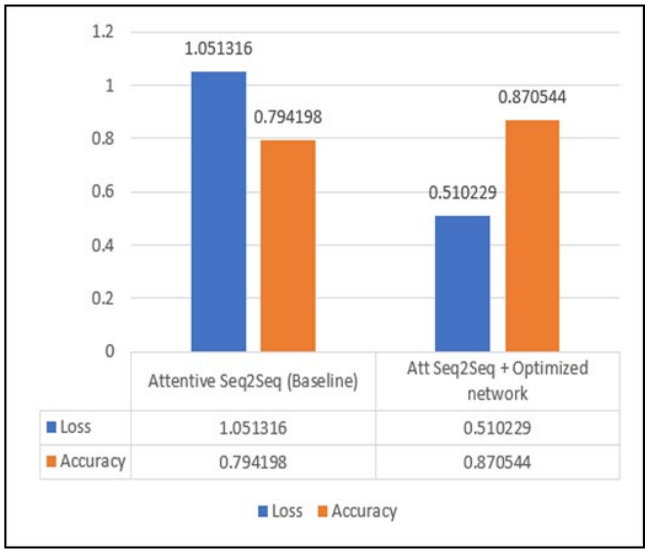


Figure 8 Data and Visualization of Overall Model Performance

The experiment involves training two models, which are the Attentive Seq2Seq model (baseline model), and Attentive Seq2Seq with hyperparameter optimization. These models will be trained based on their specific setting that would define how the model works.

Table 5 shows the summary of the findings for the previous experiment. It also serves as the baseline configuration and the final configuration after the optimal value and type selection for this experiment.

The models were then trained on Cornell Movie-Dialogue Corpus, run by using 10 epochs to compare the model performance according to the accuracy and loss function. The results are tabulated and illustrated as in Figure 8. The graph indicates the comparison of accuracy value and loss rate between the three models.

The graph shows that after optimization, the model's accuracy and loss rate were 87% and 0.51%, respectively, compared to the results before optimizing the network (79% accuracy and 1.05% loss).

The optimized model clearly outperforms the baseline model, as evidenced by this result.

As a conclusion, the following paper contributions were made via network training environment optimization whereas we proposed the optimization of the network training environment hyperparameters as a general technique to improve deep neural network (DNN) generative-based chatbot. The loss function is chosen as a high impact hyperparameter while fine-tuning the Optimizer, Learning Rate, and Dropout parameters.

The future works by considering the cost and time, we suggest that the optimization will be done on a whole network that considering all of potential high impact hyperparameters by using automated techniques such as Grid Search, Random Search and Bayesian Optimization.

Acknowledgement

The authors fully acknowledged Universiti Malaysia (UNIMAS) for their financial support under Zamalah Graduate Scholarship in which contributing the support in this research process.

References

[1] Mathew, S. 2018. The Value Of Chatbots For Today's Consumers. Forbes,https://www.forbes.com/sites/forbescommunicationscouncil/2018/02/13/the-value-of-chatbots-for-todays-consumers/?sh=4b3f408b2918 Retrieve on 23 September 2023

[2] Bansal, H., and R. Khan. 2018. A Review Paper on Human Computer Interaction. *International Journals of Advanced Research in Computer Science and Software Engineering*. ISSN: 2277-128X. 8(4): 53–56.

[3] Shang, L., Z. Lu, and H. Li. 2015, March. Neural responding machine for short- text conversation. P*roceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (Volume 1: Long Papers)

[4] Vinyals, O., and Q. V. Le. 2015, July. A neural conversational model. *Proceedings of the 31st International Conference on Machine Learning, Lille, France*. 37

[5] Nuruzzaman, M., and O. K. Hussain. 2018. A Survey on Chatbot Implementation in Customer Service Industry through Deep Neural Networks. *IEEE 15th International Conference on e-Business Engineering (ICEBE*). 54–61.

[6] Elgeldawi, E., A. Sayed, A. R. Galal, and A. M. Zaki. 2021, November. Hyperparameter Tuning for Machine Learning Algorithms. *Informatics 2021*. 8(79)

[7] Wu, Y., Z. Li, W. Wu, and M. Zhou. 2018. Response selection with topic clues for retrieval-based chatbots. *Neurocomputing*. 316: 251–261.

[8] Grossi, E., and M. Buscema. 2008. Introduction to artificial neural networks. *European Journal of Gastroenterology Hepatology*. 1046–1054.

[9] Deng, L., and D. Yu. 2014. Deep Learning: Methods and Applications. *Foundations and Trends in Signal Processing*. 7 (3–4): 197–387

[10] Vargas, R., A. Mosavi, and R. Ruiz. 2017. Deep Learning: A Review. *Advances in Intelligent Systems and Computing*. 5.

[11] Sherstinsky, A. 2020, March. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *Physica D: Nonlinear Phenomena: Special Issue on Machine Learning and Dynamical Systems*. 404.

[12] Hochreiter, S. 1998, April. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*. 6: 107–116. DOI: http://dx.doi.org/10.1142/S0218488598000094

[13] Pascanu, R., T. Mikolov, and Y. Bengio. 2012, November. On the difficulty of training Recurrent Neural Networks. *30th International Conference on Machine Learning, ICML 2013.*

[14] Yuhuang, H., H. E. G. Adrian, A. Jithendar, and L. Shih-Chii. 2018. Overcoming the vanishing gradient problem in plain recurrent networks. *ArXiv*, *vol. abs/1801.06105*.

[15] Agnihotri, S. 2019. Hyperparameter Optimization on Neural Machine Translation.

[16] Zheng, A. 2015. *Evaluating Machine Learning Models*. O'Reilly.

[17] Hajiabadi, M., M. Farhadi, V. Babaiyan, and A. Estebsari. 2020, August. Deep Learning with Loss Ensembles for Solar Power

Prediction in Smart Cities. *Smart Cities*. 3: 842–852. DOI: http://dx.doi.org/10.3390/smartcities3030043

[18] Jun, Q., D. Jun, S. Marco, M. Xiaoli, and L. Chin-Hui. 2020, August. On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression.

[19] Sebastian, R. 2018, September. *An overview of gradient descent optimization algorithms.*

[20] Mukkamala, M. C., and M. Hein. 2017. Variants of RMSProp and Adagrad with logarithmic regret bounds. *In Proceedings of the 34th International Conference on Machine Learning.* 70 (ICML'17): 2545–2553.

[21] Kingma, D. P., and J. Ba. 2019. Adam: A Method for Stochastic Optimization. *CoRR, vol. abs/1412.6980*.

[22] Senior, A., G. Heigold, M. Ranzato, and K. Yang. 2015. An empirical study of learning rates in deep neural networks for speech recognition. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 6724-6728. DOI: http://dx.doi.org/10.1109/ICASSP.2013.6638963

[23] Srivasta, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014, June. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 15: 1929 - 1958