J Inf Process Syst, Vol.16, No.5, pp.1074~1082, October 2020
https://doi.org/10.3745/JIPS.02.0141

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Comparison of Reinforcement Learning Activation Functions to Improve the Performance of the Racing Game Learning Agent

Dongcheul Lee*

Abstract
Recently, research has been actively conducted to create artificial intelligence agents that learn games through reinforcement learning. There are several factors that determine performance when the agent learns a game, but using any of the activation functions is also an important factor. This paper compares and evaluates which activation function gets the best results if the agent learns the game through reinforcement learning in the 2D racing game environment. We built the agent using a reinforcement learning algorithm and a neural network. We evaluated the activation functions in the network by switching them together. We measured the reward, the output of the advantage function, and the output of the loss function while training and testing. As a result of performance evaluation, we found out the best activation function for the agent to learn the game. The difference between the best and the worst was 35.4%.

Keywords
Activation Function, Racing Game, Reinforcement Learning

1. Introduction

Reinforcement learning (RL) is one kind of machine learning paradigm concerned with how software agents should take actions in an environment to maximize reward [1]. The environment is typically stated in the form of a Markov decision process (MDP) but it does not assume knowledge of an exact mathematical model of the MDP [2]. The main objective of RL is finding a policy that decides the agent's action selection, which results in a maximum reward. Several RL algorithms have been proposed such as fixed Q-targets, deep Q-learning (DQN), prioritized experience replay (PER), asynchronous advantage actor-critic (A3C), actor-critic with experience replay (ACER), and proximal policy optimization (PPO). Among these algorithms, ACER achieved the best result when it comes to learning a two-dimensional (2D) racing game [3]. Generally, but not always, ACER is good for discrete actions in the multi-processed environment. However, since it has a more complex architecture, it is harder to implement. To learn video games, researches have to decide the features of the game by themselves. Another way is using pixels of the game screen into the input of a neural network. To process the pixels in the neural network, convolutional neural network (CNN) is generally used [4]. In some video games, gamers cannot get a

Manuscript received March 25, 2020; first revision May 12, 2020; second revision July 20, 2020; accepted July 27, 2020.

Corresponding Author: Dongcheul Lee (jackdclee@hnu.kr)
* Dept. of Multimedia Engineering, Hannam University, Daejeon, Korea (jackdclee@hnu.kr)

reward right after they took action. They can only get a reward after a certain period of time. For example, in case of a flight shooting game, shooted missiles can reach a target after a certain time and get a reward. To learn this kind of game, the agent should remember previous states and use them later when it decides which action to choose to get the best reward. Generally, the neural network uses recurrent neural network (RNN) or long short-term memory (LSTM) to remember previous states [5].

When there are many layers in the neural network, we use activation functions to combine the layers. Researches have been developed many kinds of activation functions until now. The performance of the agent depends on several factors, but using any of the activation functions is also an important factor. However, there has been no activation function which generally shows the best result in every circumstance. In this paper, we build an RL agent to learn a 2D racing game using the ACER algorithm. The network composed of CNN and LSTM. Also, we compare the performance of the various activation functions which were used in the agent.

This paper consists of the following: the next section introduces the gaming environment and activation functions that were used in the agent. In Section 3, we define the architecture of the agent to compare the performance of the activation functions. In Section 4, we evaluate the gaming results of the agent and analyze the performance of each activation function. The paper concludes with a short discussion.

2. Related Works

2.1 OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing RL algorithms [6]. It makes no assumptions about the architecture of the agent and provides a collection of Atari games—environments—that we can use to work out the agent. The environment returns an observation which includes pixel data from a screen for each frame. It also returns a reward which is achieved by the previous actions. Generally, the goal of the agent is always to increase its total reward in a short time. In each timestep, the agent chooses an action, and the environment returns an observation and a reward for the action. With OpenAI Gym, RL researchers can have various and easy-to-setup RL environments that are needed for better benchmarks and standardization of environments used in their publications.

2.2 Activation Function

Generally, we use a neural network for RL. The neural network consists of layers of neurons. The neuron is a mathematical function that combines input with a set of weights (w) that either amplify or dampen the input so that it can assign significance to inputs with regard to the task the agent is trying to learn. The output of the neuron is the weighted sum of its inputs plus a bias (b) that allows us to make affine transformations to the input. Then the output is passed through the activation function to determine whether and to what extent that input should progress further through the network for classification. If the input passes through, the neuron has been activated. The activation function is defined as follows:

$$f(x) = Activation(\sum_{i=1}^{n} x_i w_i + b) \quad (1)$$

Table 1 and Fig. 1 show the activation functions used in the agent. Early-stage of an activation function was a sigmoid function. It has been used because it exists between (0 to 1) so that is can predict the probability as an output. It is differentiable so that we can use backpropagation. However, for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This results in the network being too slow to learn or stop to learn further.

Table 1. Activation functions used in the RL agent

Name	Equation
ReLU	$f(x) = \begin{cases} 0, \text{for } x < 0 \\ x, \text{for } x \geq 0 \end{cases}$
ReLU6	$f(x) = \begin{cases} 0 \quad , \text{for } x < 0 \\ x, \text{for } 0 \leq x < 6 \\ 6 \quad , \text{for } x \geq 6 \end{cases}$
Leaky ReLU	$f(x, \alpha) = \begin{cases} \alpha x, \text{for } x < 0 \\ x \quad , \text{for } x \geq 0 \end{cases}$
ELU	$f(x, \alpha) = \begin{cases} \alpha(e^x - 1), \text{for } x < 0 \\ x \qquad\qquad , \text{for } x \geq 0 \end{cases}$
SELU	$f(x, \alpha, \beta) = \begin{cases} \alpha\beta(e^x - 1), \text{for } x < 0 \\ \beta x \qquad\qquad , \text{for } x \geq 0 \end{cases}$
Swish	$f(x) = \dfrac{x}{1 + e^{-x}}$
CReLU	$f(x) = (\max(0, x), \max(0, -x))$

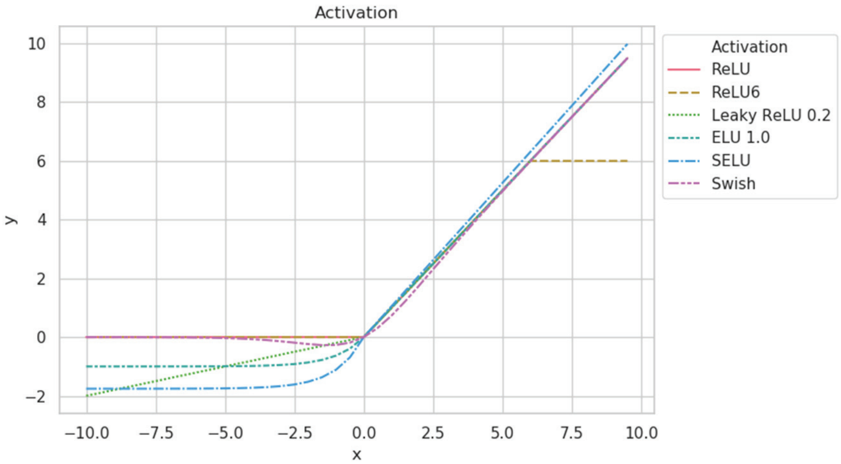


Fig. 1. The plot of activation functions used in the RL agent.

Rectified linear unit (ReLU) can solve this problem and it is the most used activation function right now. It is half rectified from the bottom. It is computationally efficient and non-linear allowing the network to converge quickly and allowing for backpropagation. When the unit of ReLU is capped at 6, it is called ReLU6. This encourages the network to learn sparse input earlier. This is equivalent to imagining that each ReLU unit consists of only 6 replicated bias-shifted Bernoulli units, rather than an infinite amount [7]. However, both ReLU and ReLU 6 cause the dying ReLU problem: when inputs approach zero or are negative, the gradient of the function becomes zero so that the network cannot perform backpropagation and cannot learn.

Leaky ReLU has a small positive slope in the negative area, so it enables backpropagation even for negative input [8]. However, it does not provide consistent predictions for negative input. We used 0.2 for the value of α. Exponential linear unit (ELU) also has a small curved slope in the negative area to solve the dying ReLU problem. It saturates to a negative value with smaller inputs and thereby decrease the forward propagated variation and information [9]. We used 1.0 for the value of α. Scaled ELU (SELU) extends ELU inducing self-normalizing properties. It adds one more parameter to control not only the gradient of the negative area but also the positive area [10]. We used 1.76 for the value of α and 1.05 for β.

Swish is a new, self-gated activation function discovered by researchers at Google [11]. According to their paper, it performs better than ReLU with a similar level of computational efficiency. In experiments on ImageNet with identical models running ReLU and Swish, the new function achieved top-1 classification accuracy 0.6%–0.9% higher [11].

Concatenated ReLU (CReLU) Concatenates a ReLU which selects only the positive part of the activation with a ReLU which selects only the negative part of the activation. As a result, this non-linearity doubles the depth of the activations [12].

3. Analysis Scheme

We build an RL agent that can learn a 2D racing game in the OpenAI Gym to analyze the effect of the activation functions which were discussed in Section 2.2. The agent utilizes the ACER algorithm to learn the game using CNN and LSTM. Fig. 2 shows the composition of the network. We have the following implementation principle to build this agent: Before the agent provides input to the network, it converts a color image to a grayscale image and crops the border to reduce the complexity of the input. Then CNN processes the pixels from the game screen to decide which action to take. LSTM remembers the previous state to help the decision. The network includes 3 CNN, 1 fully-connected network, and 1 LSTM. Each CNN uses 32, 64, and 64 filters, respectively. The size of each filter is 8×8, 4×4, and 3×3 with a stride 4, 2, and 1, respectively. LSTM uses 512 cells. The activation functions are used in each hidden layer.

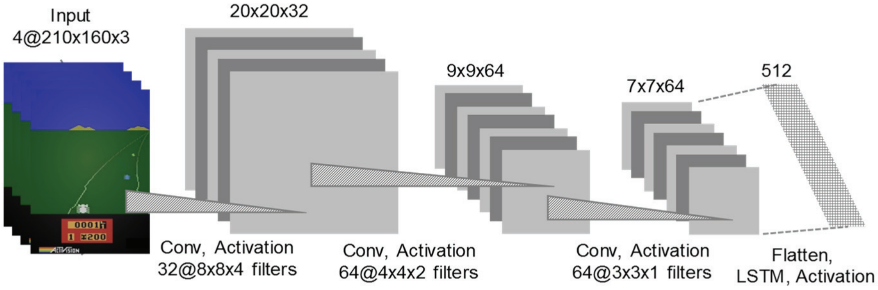


Fig. 2. Illustration of the neural network used by the RL agent to learn how to play a 2D racing game.

We used Enduro v4 in OpenAI Gym for playing a 2D racing game. The game consists of maneuvering a race car in a long-distance endurance race. The object of the game is to pass 200 cars each day. The driver should avoid other cars, otherwise, the driver's car stops. As the driver passes another car, the reward increases. However as other cars pass the driver, the reward decreases.

The agent observes pixel data of the game s_t at time step *t*, chooses an action a_t according to a policy $\pi(a_t|s_t)$, and observes a reward $r_t{}^t$ produced by the game. The goal of the agent is to maximize the discounted return $R_t = \sum_{i\geq 0} \gamma^i r_{t+i}$ at time step *t*. Discount factor $\gamma \in [0,1)$ trades-off the importance of rewards. The output layer of the network produce $\pi(a_t|s_t)$ and a value function $V^\pi(s_t)$. The state-action and state only value function is defined as:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}:\infty, a_{t+1}:\infty}[R_t|s_t, a_t], \text{ and} \tag{2}$$

$$V^\pi(s_t) = \mathbb{E}_{a_t}[Q^\pi(s_t, a_t)|s_t]. \tag{3}$$

The advantage function provides a relative measure of the value of each action, which is defined as:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{4}$$

We defined a loss function to minimize the total error of the training data. The function allows us to increase the weight for actions that yielded a positive reward, and decrease them for actions that yielded a negative reward. It is defined as:

$$L^\pi(s_t) = -\log \pi(a_t|s_t) A^\pi(s_t, a_t). \tag{5}$$

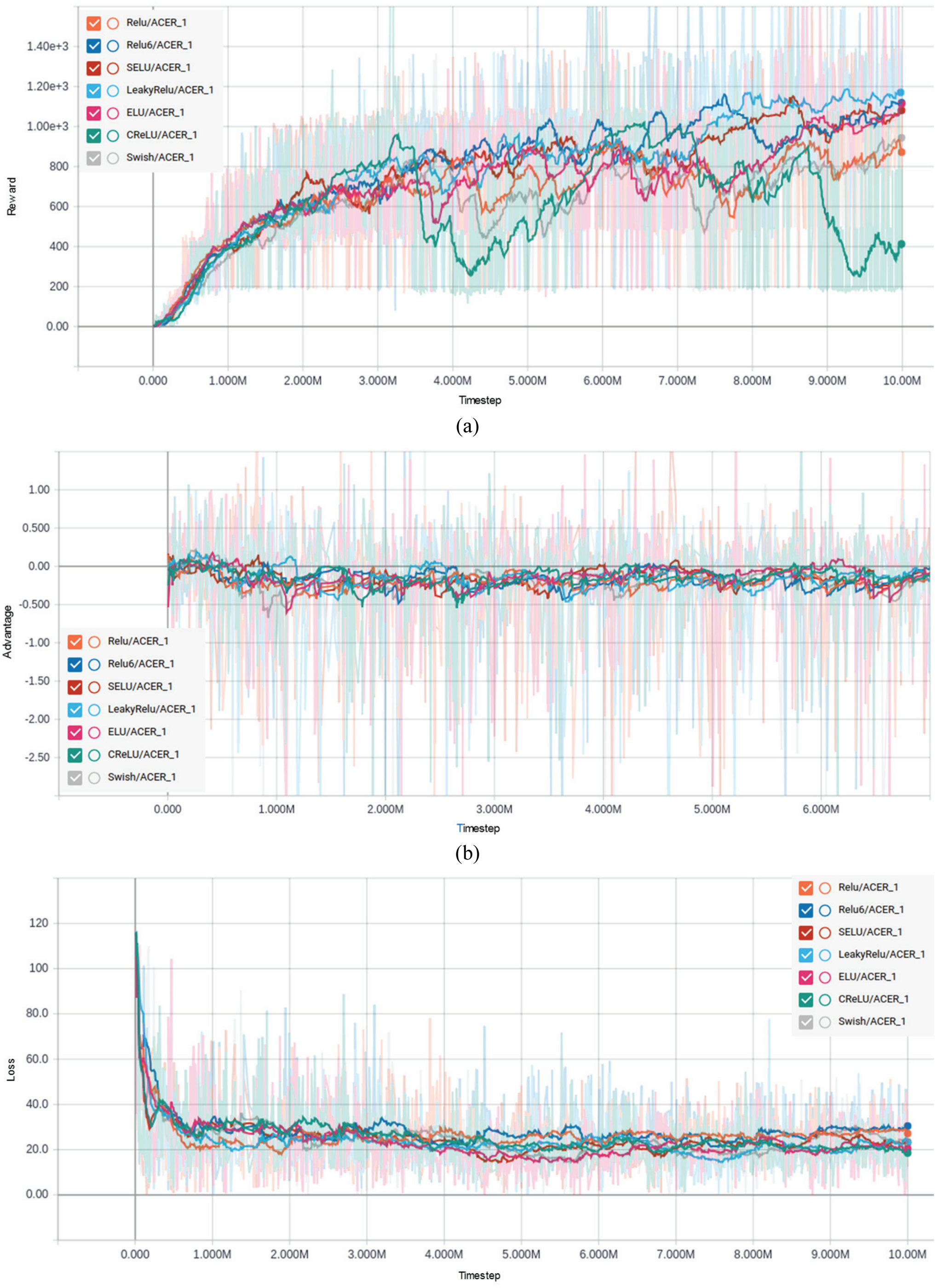
The agent runs on Ubuntu 18.04 machine with Intel Xeon w-2102 CPU, Nvidia GeForce GTX 1080ti GPU, and 32G RAM. We build a python agent using TensorFlow 1.10, Keras 2.2, OpenAI Gym 0.13, and Python 3.6. The agent trains itself during 1×10^7 timesteps for each activation function. Hyperparameters in the agent are listed in Table 2.

Table 2. Hyperparameters in the agent

Name	Value
Discount factor	0.99
N_steps	20
Q value coefficient	0.5
Entropy coefficient	0.01
Learning rate	0.0007
RMSProp decay parameter	0.99
RMSProp epsilon	0.00001
Buffer size	5000
Replay ratio	4

4. Performance Analysis

To evaluate the training performance of each activation function, we compared the reward, the value of $A^\pi(s_t, a_t)$, and the value of $L^\pi(s_t)$ during the training. Fig. 3 shows performance metrics for each activation function along with timesteps during training. The plot was smoothed with a smoothing factor 0.8. Original data is plotted with a dim color. Except for CReLU, every activation function gets increased reward as timestep increases. Leaky ReLU got the highest reward whereas CReLU got the smallest at the end of the training. In the case of CReLU, it seems the agent was stuck in local optima at 4.2M and 9.4M timestep. Reducing the learning rate or incorporating a method of maintaining stochasticity will resolve this. There is not much difference in $A^{\pi}(s_t, a_t)$ and $L^{\pi}(s_t)$.



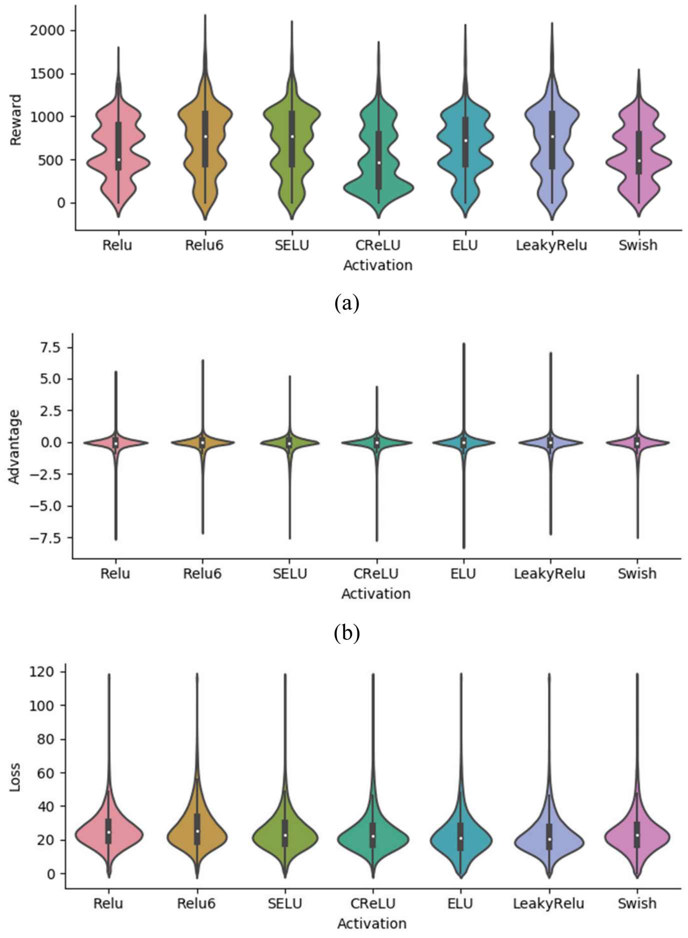
(a)

(b)

(c)

Fig. 3. Performance metrics for each activation function along with timesteps during training. (a) The reward. (b) The output of the advantage function $A^{\pi}(s_t, a_t)$. (c) The output of the loss function $L^{\pi}(s_t)$.

Fig. 4 shows the violin plot of performance metrics for each activation function during training. Leaky ReLU has high probability density on the high reward whereas CReLU has a high density on the low reward. Most values of $A^{\pi}(s_t, a_t)$ are densely distributed around the negative area of zero, which means most of the actions taken had been stably distributed around the optimal actions. The values of $L^{\pi}(s_t)$ are distributed around 20 after 1.0M timestep meaning our model needs to be improved to reduce the loss. The plot has a greater variance than the plot of $A^{\pi}(s_t, a_t)$.



(a)

(b)

(c)

Fig. 4. The violin plot of performance metrics for each activation function during training: (a) the reward, (b) the output of the advantage function $A^{\pi}(s_t, a_t)$, and (c) the output of the loss function $L^{\pi}(s_t)$.

After 1×10^7 timesteps' training, 100 episodes of the game were played during the testing. Table 3 shows the mean and maximum value of the reward for each activation function. ReLU6 achieved the highest mean reward while CReLU was the smallest. The ReLU6's reward is 35.4% higher than the CReLU's. ReLU6 also achieved the highest maximum reward while Swish was the smallest. The ReLU6's reward is 42.8% higher than the Swish's. Even though Leaky ReLU got the highest reward in training, ReLU6, which got the second-highest reward in training, got the highest in testing due to the stochastic nature of the algorithm.

Table 3. Mean and maximum reward for each activation function during the testing

Activation function	Mean	Max
ReLU	617.350515	1643
ReLU6	711.957198	1984
Leaky ReLU	693.454545	1888
ELU	651.442935	1897
SELU	693.479087	1918
Swish	575.337388	1389
CReLU	525.686466	1692

5. Conclusion

In this paper, we have compared the performance of the activation functions in the RL agent to learn a 2D racing game. We have built the agent using the ACER algorithm and CNN+LSTM model. We have tested the activation functions in the hidden layer of CNN by switching them together. We have measured the reward, the output of the advantage function, and the output of the loss function while training and testing. As a result of performance evaluation, we have found ReLU6 performs better than other activation functions whereas CReLU performs the worst in terms of the total reward. The difference between them was 35.4%. This result shows that choosing an activation function is important since we can get a worse reward even though we used the same RL scheme in the same environment. Most RL papers have used ReLU not mentioning why they use it. Through our benchmark, we can see that the result would vary depending on which activation function they've used.

As future work, we will evaluate the performance of other factors than the activation function for the RL agent. Through this work, we can figure out suitable factors to improve the performance depending on the type of a game or the learning algorithm. We hope we can utilize this result in real-world applications such as a self-driving car.

References

[1] A. Jeerige, D. Bein, and A. Verma, "Comparison of deep reinforcement learning approaches for intelligent game playing," in *Proceedings of 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, 2019, pp. 366-371.

[2] M. N. Moghadasi, A. T. Haghighat, and S. S. Ghidary, "Evaluating Markov decision process as a model for decision making under uncertainty environment," in *Proceedings of 2007 International Conference on Machine Learning and Cybernetics*, Hong Kong, China, 2007, pp. 2446-2450.

[3] D. Lee and B. Park, "Comparison of deep learning activation functions for performance improvement of a 2D shooting game learning agent," *The Journal of the Institute of Internet, Broadcasting and Communication*, vol. 19, no. 2, pp. 135-141, 2019.

[4] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611-629, 2018.

[5] D. W. Lu, "Agent inspired trading using recurrent reinforcement learning and LSTM neural networks," 2017 [Online]. Available: https://arxiv.org/abs/1707.07338.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI gym,” 2016 [Online]. Available: https://arxiv.org/abs/1606.01540.

[7] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, “Dying ReLU and initialization: theory and numerical examples,” 2019 [Online]. Available: https://arxiv.org/abs/1903.06733.

[8] X. Zhang, Y. Zou, and W. Shi, “Dilated convolution neural network with LeakyReLU for environmental sound classification,” in *Proceedings of 2017 22nd International Conference on Digital Signal Processing (DSP)*, London, UK, 2017, pp. 1-5.

[9] A. Shah, E. Kadam, H. Shah, S. Shinde, and S. Shingade, “Deep residual networks with exponential linear unit,” in *Proceedings of the 3rd International Symposium on Computer Vision and the Internet*, Jaipur, India, 2016, pp. 59-65.

[10] Z. Huang, T. Ng, L. Liu, H. Mason, X. Zhuang, and D. Liu, “SNDCNN: self-normalizing deep CNNs with scaled exponential linear units for speech recognition,” 2019 [Online]. Available: https://arxiv.org/abs/1910.01992.

[11] G. C. Tripathi, M. Rawat, and K. Rawat, “Swish activation based deep neural network predistorter for RF-PA,” in *Proceedings of 2019 IEEE Region 10 Conference (TENCON)*, Kochi, India, 2019, pp. 1239-1242.

[12] Z. Wang and X. Xu, “Efficient deep convolutional neural networks using CReLU for ATR with limited SAR images,” *The Journal of Engineering*, vol. 2019, no. 21, pp. 7615-7618, 2019.

Dongcheul Lee **https://orcid.org/0000-0002-1621-6557**

He has been an associate professor in the Department of Multimedia at Hannam University, Korea, since 2012. He received the B.S. and M.S. degrees in Computer Science and Engineering from POSTECH, Korea in 2002 and 2004, respectively, and Ph.D. degrees in Electronics and Computer Engineering from Hanyang University, Korea in 2012. He had been a senior researcher at the KT Mobile Network Laboratory, Korea from 2004 to 2012. His main topics of interest have been communication in mobile applications, rich communication service, a software framework for mobile games, and machine learning in games.