Journal of Korea Game Society 2017 Oct; 17(5): 103-112
http://dx.doi.org/10.7583/JKGS.2017.17.5.103

게임 프로그래밍

Sparse Autoencoder의 데이터 특징 추출과 ProGReGA-KF를 결합한 새로운 부하 분산 알고리즘

김차영*, 박정민**, 김혜영**
경기대학교 컴퓨터과학과*, 홍익대학교 게임학부**
kimcha0@kgu.ac.kr, bjmbam12@naver.com, hykim@hongik.ac.kr

Combing data representation by Sparse Autoencoder and the well-known load balancing algorithm, ProGReGA-KF

Chayoung Kim*, Jung-min Park**, Hye-young Kim**
Dept. of Computer Science, Kyonggi University*, School of Games, Hongik Unversity**

요 약

많은 사용자가 함께 즐기는 온라인 게임(MMOGs)에서 IoT의 확장은 서버에 엄청난 부하를 지속적으로 증가시켜, 모든 데이터들이 Big-Data화 되어가는 환경에 있다. 이에 본 논문에서는 딥러닝 기법 중에서 가장 많이 사용되는 Sparse Autoencoder와 이미 잘 알려진 부하분산 알고리즘(ProGReGA-KF)을 결합한다. 기존 알고리즘 ProGReGA-KF과 본 논문에서 제안한 알고리즘을 이동 안정성으로 비교하였고, 제안한 알고리즘이 빅-데이터 환경에서 좀 더 안정적이고 확장성이 있음 시뮬레이션을 통해 보였다.

ABSTRACT

In recent years, expansions and advances of the Internet of Things (IoTs) in a distributed MMOGs (massively multiplayer online games) architecture have resulted in massive growth of data in terms of server workloads. We propose a combing Sparse Autoencoder and one of platforms in MMOGs, ProGReGA. In the process of Sparse Autoencoder, data representation with respect to enhancing the feature is excluded from this set of data. In the process of load balance, the graceful degradation of ProGReGA can exploit the most relevant and less redundant feature of the data representation. We find out that the proposed algorithm have become more stable.

Keywords : Sparse Autoencoder, massively multiplayer online games (MMOGs), load balance, Deep Learning, data representation(딥러닝, 부하분산, Sparse Autoencoder, MMOGs, 데이터 특징 추출)

Received: Sep. 11. 2017 Revised: Oct. 18.; 2017
Accepted: Oct. 20. 2017
Corresponding Author: Hye-young Kim(Hongik University)
E-mail: hykim@hongik.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. Introduction

In the distributed architecture of massively multiplayer online games (MMOGs) in IoTs [1], there have been main existing issues such as managing thousands of participants simultaneously and providing guarantees consistency and resistance for intermediate communications on the support network. Interactions between the huge numbers of players might be generated and growing considerably compared to the number of players, which is mainly responsible for engendering a Big-Data world. There have been many proposals for considering the available information brought into the distributed servers. Obviously, the main issues of those proposals how can we exploit the historically information. That means MMOGs in the Big-Data[2] world require a new criterion for load balancing because the accumulated huge amount of information might be useful if the information can be represented into enhancing the most relevant and less redundant form. Based on the data representation by learning model[10], such as Sparse Autoencoder[6], the freer move of the players' avatars in the game world, the more possible the formation of severs' reactions to try to reduce making errors such as hotspots [7], around which the players are more concentrated than in other regions of the world, causing one of main reasons to be degraded. In the learning model, if severs get communizing each other in a manageable size of a group of avatars, then they can learn each other by interactions and occurring information between them can be huge brought into Big-Data world. We consider the ongoing issue, how we can make the demonstration of the server managing powers by the accumulated information. For that reason, the server-based software has the ability of analyzing the patterns of communications between players by preprocess and adapting the considered features to service balance. There have been the previous load balancing schemes in MMOGs for preventing the presence of hotspots from degrading the quality of the game world and considering the reduction of the inter-server communication overhead. However, there are little works on that featured information issues. Therefore, our work is beyond the qualification of the tolerable load balancing because the details of loads and the players' demands in game world are given rise to the Big-Data, making enriched meaningful features.

The purpose of our work is to learn the patterns of loads by these avatar world based on the information obtained from the past loads accumulated. In the virtual world, learning the overhead patterns in advance can benefit load balancing, prevent degradation of quality of the game and enable the severs to take prompt actions. In this paper, the past loads provided by our developed game simulation has been used as our experimental dataset. Our studied past loads have the following properties: the number of cells per region, which assigned to a different server, and how many cells it is linked with. To exploit these properties, we proposed a novel approach in terms of featured information. In our approach, we combine a Sparse Autoencoder[6] as our data reconstruction to

achieve a better representation of the inputs and a well-known greedy graph partition growing algorithm, ProGReGA[7]. The Sparsity Autoencoder[6] can makes the most relevant and less redundant of input data. So, the preprocessed feature layer by the Sparse Autoencoder[6] can further capture the most relevant features from the past loads. The proposed our algorithm is compared with the representation-free version of ProGReGA-KF [7] on the own simulation. It finds out that our proposed algorithm has become more stable and scalable than the representation-free ProGReGA-KF[7] with the less number of reducing player migrations between servers. The rest of the paper is organized as follows. The related work is presented in Section 2. Section 3 describes the details of our proposed algorithm. Section 4 gives the experimental results and comparison. Finally, we conclude our work in Section 5.

2. Related Works

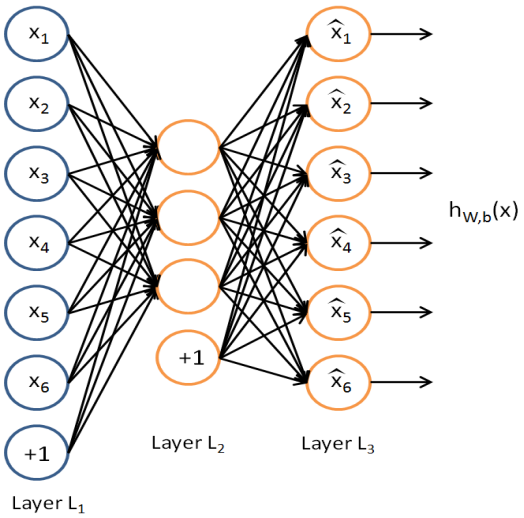
The Internet of Things (IoT)[1] is rapidly gaining ground in the field of modern wireless telecommunications. The basic foundation of the IoT is the pervasive presence around us of a variety of things or objects－such as radio-frequency identification (RFID) tags[5], sensors, actuators, mobile phones－which, through unique addressing schemes, are able to interact with each other and cooperate with their neighbors to reach common goals. In[8], the authors combine the Phase Space Reconstruction (PSR) and Group Method of Data Handling method based on Evolutionary Algorithm (EA-GMDH) for host load prediction. The prediction performance of this method is closely related to the parameters they chose, as the evolutionary algorithm is a stochastic global search method which may fall into local optima. ESN is a rather recent development in the field of RNN[8] and it lead to a fast, simple and constructive algorithm for supervised training of RNN. The basic idea of ESN is to use a large reservoir RNN as a supplier of interesting dynamics from which the desired output is combined. The philosophy adopted in Reservoir Computing is to consider the recurrent layer as a large reservoir of nonlinear transformations of the input data and decouple the learning of parameters inside and outside the reservoir.

Unsupervised feature learning[11] refers to a class of machine learning techniques[12], developed rapidly since 2006, where many stages of nonlinear information processing in hierarchical architectures are exploited for pattern classification. Recently, unsupervised feature learning technologies have been successfully used in many research areas, such as handwritten digit images recognition[3], visual object classification[3] and nature language process[3]. After initializing the deep neural network with unsupervised feature learning algorithms [e.g., autoencoder[6], matrix factorization and restricted Boltzmann machines (RBM)[4]], the weights are starting at a better location in parameter space than if they had been randomly initialized. Because the deep neural network can also be considered to perform feature learning, since they learn a representation of their input at the hidden layers which is subsequently used for

classification or regression at the output layer. One of load balancing schemes for distributed MMOG servers, taking into account the use of upload bandwidth of the server nodes[13]. The scheme[7], which is divided into three phases, proposed different algorithms for phase 2 (the balancing phase), and ProGReGA[7] presented the lowest overhead of all, while ProGReGA-KF[7] presented the second fewest migrations of players between servers, along with the second lower overhead introduced and a fair load distribution.

3. The proposed Algorithm

3.1 Sparse Autoencoder



[Fig. 1] Sparse Autoencoder

There are unlabeled training examples set $\{x^{(1)},x^{(2)},x^{(3)},...\}$,where $x^{(i)}\in R^n$. An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs .i.e., it uses $y^{(i)}=x^{(i)}$. It is trying to learn an approximation to the identityfunction, $h_{W,b}(x) \approx x$, so that the output $y^{(i)}$ is similar to $x^{(i)}$. By applying constraints on the network, such as by limiting the number of hidden units or the average activation of the hidden units, we can discover high-level features of the input data. The features learned automatically can improve the accuracy of the classification and regression tasks comparing with the features designed manually. There are arguments about the number of hidden units being small or large (perhaps even greater than the number of input pixels). In particular, there is a sparsity constraint on the hidden units, then the autoencoder will still discover interesting structure in the data, even if the number of hidden units is large. For the overall cost function, Equation 5, there are Equation 1 and Equation 2 for the sparsity parameter, ρ. In Equation 1, $a_j^{(2)}(x)$ denotes the activation of this hidden unit j in the autoencoder when the network is given a specific input x. In Equation 2, there is the constraint, where ρ is a sparsity parameter, typically a small value close to zero (say ρ = 0.05). In Equation 3, there is an added extra penalty term to the objective that penalizes $\hat{\rho}_j$ deviating significantly from ρ, where s_2 is the number of neurons in the hidden layer, and the index j is summing over the hidden units. This penalty term is based on the Kullback-Leibler(KL)[6] divergence in Equation 4. The Kullback-Leibler(KL)[6] divergence betweena Bernoulli random variable with mean ρ and a Bernoulli random variable with mean $\hat{\rho}_j$. In Equation 5, there is overall cost function,

where β controls the weight of the sparsity penalty term and $\hat{\rho}_j$ depends on W, b, and the activation of a hidden unit, j depends on the parameters W, b.

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}[a^{(2)}_j(x^{(i)})] \quad \text{---------- Equation 1}$$

$$\hat{\rho}_j = \rho \quad \text{---------- Equation 2}$$

$$\sum_{j=1}^{s2}\rho \log(\rho/\hat{\rho}_j) + (1-\rho)\log((1-\rho)/(1-\hat{\rho}_j)) \quad \text{- Equation 3}$$

$$\sum_{j=1}^{s2} KL(\rho||\hat{\rho}_j) \quad \text{---------- Equation 4}$$

$$J_{sparse}(W,b) = J(W,b) + \beta \sum_{j=1}^{s2} KL(\rho||\hat{\rho}_j) \quad \text{---- Equation 5}$$

```
Algorithm: Local regions selection
1. local_group <-{R}  #R is Region
2. local_weight <-wr(R)  #Regions' weight
3. local_capacity<-p(s(R))  #Power of Server of Region
4. average_usage<-local_weight/local_capacity
5. while average_usage > max(1, Utotal)do
                                  # Utotal s SystemUsage
6.      if there is any not selected region neighbor to one
of local_group then
7.            R<- not selected region neighbor to one of
local_group, with smallest
8.            u(s(R))          #Region's resource usage
9.      else if there is any empty region then
10.           R<- empty region with highest p(s(R))
11.     else
12.           stop when no more regions to select
13.     end if
14.     local_weight<-local_weight+wr(R)#Regions'weight
15.     local_capacity<-p(s(R))  #Power of Server of Region
16.     average_usage<-local_weight/local_capacity
17.     local_group <- local_group ∪ {R}
18. end while
19. Running the local balancing algorithm with local_group
as input
```

[Fig. 2] Local Regions Selection

3.2 ProGReGA

In the previous well-known load balance on MMOGs, there are three phases, where firstly, selecting the group of local regions, secondly, balancing these regions, assigning to each one a weight which is proportional to the power of its server, and lastly, refining the partitioning, reducing the overhead. In Fig. 2, there is a phase of selecting local regions. In Fig. 3, there is a load balancing algorithm, ProGReGA [7] based on greedy region growing method to allocate the heaviest cells to the regions managed by the most powerful servers. However, the input of the algorithm receives a list of the regions without data representation. So, it is not achievable and scalable in terms of Big-Data[2].

```
Algorithm: ProGReGA
1. weight_to_divide <-0
2. free_capacity <-0
3. for each region R in Region_List do
4.      weight_to_divide <- weight_to_divide + wr(R)
5.      free_capacity <- free_capacity + p(s(R))
6.      Free all cells from R temporarily
7. end for
8. Sort Region_List in decreasing order of p(s(R))
9. for each region R in Region_List do
10.                    weight_share <-weight_to_divide ×
p(s(R))/free_capacity
11.       while wr(R)<weight_sharedo
12.            if there is any cell from R neighboring a free
cell then
13.                 R <- R ∪ {neighbor free cell with the
highest Intc(AC)}
14.           else if there is any free cell then
15.                 R <- R ∪ {the heaviest free cell}
16.           else
17.                 Stop no more free cells
18.           end if
19.       end while
20. end for
```

[Fig. 3] ProGreGA Algorithm

3.3 ProGReGA-KF

```
Algorithm: ProGReGA-KF
1. weight_to_divide <-0
2. free_capacity <-0
3. for each region R in Region_List do
4.      weight_to_divide <- weight_to_divide + wr(R)
5.      free_capacity <- free_capacity + p(s(R))
6.      cell_list <- list of cells from R in increasing order of weight
7.      while fracr(R) > fracp(s(R)) do
8.           C<-first element from cell_list
9.           Remove C from R
10.          Remove C from cell_list
11.     end while
12. end for
13. Sort Region_List in increasing order of u(s(R))
14. for each region R in Region_List do
15.      weight_share <-weight_to_divide × p(s(R))/free_capacity
16.      while wr(R)<weight_sharedo
17.           if there is any cell from R neighboring a free cell then
18.                R <- R ∪ {neighbor free cell with the highest
Intc(AC)}
19.           else if there is any free cell then
20.                R <- R ∪ {the heaviest free cell}
21.           else
22.                Stop no more free cells
23.           end if
24.      end while
25. end for
```

[Fig. 4] ProGReGA-KF Algorithm

In this section there is another load

balancing algorithm, ProGReGA-KF[7] based on greedy region growing method like ProGReGA[7]. In this algorithm, each region will gradually release its cells in increasing order of weight, until its weight fraction is less than or equal to the power fraction of its server. Therefore the heaviest cells remain on the same server and most players do not need to migrate. Moreover, there is an advantage that the possibility of fragmenting the regions, with many isolated cells, increasing the overhead could be mitigated by the weight fraction of the region, fracr(R).

3.4 The proposed Algorithm

```
Algorithm: ProGReGA with Sparse Autoencoder
Input: Each_Cell in all regions, C = {1, 2, ... n},
Output: Region_List based on Data Reduction
1. E=Encoded(Input, activation='relu')
2. D=Decoded(E, activation='sigmoid')
3. AC=Autoencoder(Input, D)
4. weight_to_divide <-0
5. free_capacity <-0
6. for each region R in Region_List do
7.       weight_to_divide <- weight_to_divide + w_r(R)
8.       free_capacity <- free_capacity + p(s(R))
9.       Free all cells from R temporarily
10. end for
11. Sort Region_List in decreasing order of p(s(R))
12. for each region R in Region_List do
13.       weight_share <-weight_to_divide × p(s(R))/free_capacity
14.       while w_r(R)<weight_sharedo
15.             if there is any cell from R neighboring a free cell then
16.                   R <- R ∪ {neighbor free cell with the highest
Int_c(AC)}
17.             else if there is any free cell then
18.                   R <- R ∪ {the heaviest free cell}
19.             else
20.                   Stop no more free cells
21.             end if
22.       end while
23. end for
```

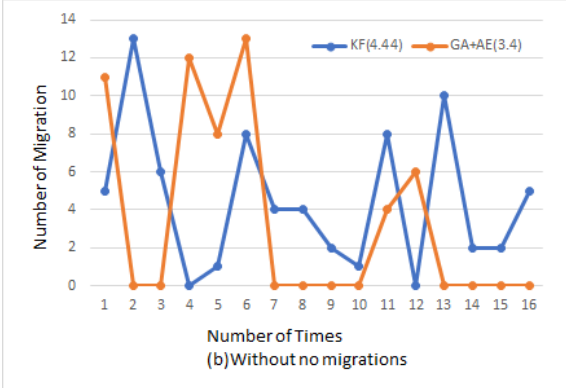
[Fig. 5] The proposed Algorithm

In this section, we explain our proposed model in details. Likewise the proposed scheme ProGReGA-KF[7], players who are interacting with each other should be connected to the same server. If two avatars of two different players might be distant from each other, both could be interacting with a third avatar between them. So, it is necessary to consider how many pairs of players and which of them will be divided into different server. Our approach is to exploit data representation by Sparse Autoencoder[6]. The overhead patterns obtained from the past loads can be a criterion of load balancing. In our approach, likewise ProGReGA-KF[7], the main aspect in our solution is focusing each server's local information that initiated the balancing process and its neighbors and accumulated information on each sever. In this regards, we attempt to map original data space to a new space which is more suitable for maximizing the relevant features as a good feature representation. The Sparse Autoencoder learning algorithm[6], which is one approach to automatically learn features from unlabeled data. After the preprocessed feature layer by the Sparse Autoencoder[6], the proposed algorithm has the simple phases like ProGReGA[7] for balancing these regions and partitioning for the overhead reduction.

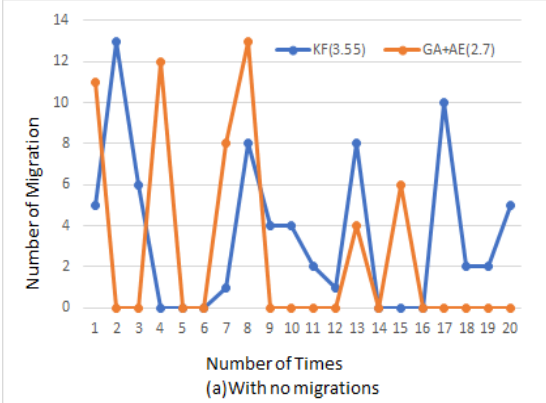
4. Performance Evaluation

In our work, our proposed algorithm, combining Sparse Autoencoder[6] and ProGReGA[7] is compared with the well-known research ProGReGA-KF[7]. For the evaluations, we consider a heterogeneous system, simulate our algorithm on a grid-like square cells, select a cell with the lowest interaction in the smallest cell cluster by the overloaded sever and transfer the loads of the selected server to the least loaded one. However, the previous one, ProGReGA-KF[7] have been forcing an uneven distribution of avatars in the virtual environment, putting to test the load balancing algorithms. The

purpose of the previous research, ProGReGA-KF[7] was how to take account of hotspots form the regions and reduce the distribution overhead. On the other way round, in a normal game world, "graceful degradation" is more realistic[7]. Therefore, in our performance evaluation, we consider the normal real game world. As our simulation based on Unity3D[9] begins, they start to move according to the random model[7]. The environment of the simulation is consisted of a two-dimensional space, divided by cells, which is belonged to some region. It can be transferred to another region for load balancing. We can take into consideration in the average weight of a region on the criterion of proportional balance and the number of migrations by the defined formulae in Chapter 3. Also, we can calculate how many time differences for preprocess between our proposed algorithm, combining Sparse Autoencoder[6] and ProGReGA[7] and the previous one ProGReGA-KF[7] in terms of Big Data.



[Fig. 6(b)] The comparison our proposed algorithm (AE+ProGreGA) with the well-known Algorithm (ProGReGA-KF)



[Fig. 6(a)] The comparison our proposed algorithm (AE+ProGreGA) with the well-known Algorithm (ProGReGA-KF)

[Fig. 6] shows the one of important results that our proposed algorithm has some advantages with respect to preprocess of Big-Data. With the reasonable size of data, load balance can be possible to handle during the game session. When it comes to Big-Data, preprocess become increasingly important. If the output of preprocess are given in a reasonable time to the load balance management, still having information meaningful, then load balance itself is considerably of serve to players during game session without letting the players know. According to the load balance depends on the specific games such as real-time games in which users constantly migrate between servers, causing delay and hinder of the interaction between players, our proposed algorithm can be considered more stable and achievable. Our proposed preprocess can handle the information from Big-Data by simplifying and managing in the proportion to the size of accumulated data.

We notice that it is considerably scalable than the previous version ProGREGA-KF[7] in terms of data representation of Sparse Autoencoder[6]. Specifically, in comparing Fig. 6(a) and Fig. 6(b), we can see that in terms of no migrations, our proposed algorithm is much more realistic. The reason for that is that our load balancing part of the proposed algorithm follows the development of ProGReGA. In the performance evaluation in [7], ProGReGA was mentioned that it has the lowest overhead as it was designed precisely to create the most realistic regions, causing the number of migrations manageable.

5. Conclusion

We have suggested the algorithm combining Sparse Autoencoder[6], one of the most used data representation, and ProGREGA[7], the most well-known load balancing scheme for distributed MMOG servers. Information obtained from the past loads of servers based on the demands of players are getting formed down into Big-Data. Our proposed algorithm tries to overcome consuming a huge amount of preprocessing time in terms of Big-Data. According to learning the loads patterns in advance can benefit load balancing, our proposed algorithm can exploit these properties by data representation of Sparse Autoencoder [6]. The proposed our algorithm is compared with the representation-free version of ProGReGA-KF[7]. Our proposed preprocess can handle the information from Big-Data by simplifying and managing in the proportion to the size of accumulated past data. Because preprocess is given in a reasonable time to the load balance management, load balance as the reaction of servers is considerably of serve to players during game session without letting the players know. Therefore, our proposed algorithm have become more scalable than the representation-free ProGReGA-KF[7] in terms of Big-Data[2]. Specifically, if it is designed precisely to create the most realistic regions, causing the number of migrations manageable, our proposed algorithm can be considered more stable and achievable.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(No. 2016RIA2B4012386)

REFERENCES

[1] D, Giusto, A. Iera, G Morabito, L. Atzori (eds) (2010) The Internet of Things. Springer, New York. ISBN 978-1-4419-1673-0

[2] IBM, Bringing Big Data to the Enterprise. http://www-01.ibm.com/software/data/bigdata/

[3] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. “A fast learning algorithm for deep belief nets.” Neural computation, Vol.18, No.7, pp.1527-1554, 2006.

[4] Fischer, Asja, and Christian Igel. “An introduction to restricted Boltzmann machines.” Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Springer Berlin Heidelberg, pp.14-36, 2012.

[5] H.-Y. Kim, “A load balancing scheme with

Loadbot in IoT networks"Journal of Supercomputing, Vol.73, pp.1-12, 2017. 7. DOI 10.1007/s11227-017-2087-6,
[6] Andrew Ng, "Sparse Autoencoder", https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf
[7] C. Eduardo, B. Bezerra, C. Fernando, and R. Geyer, "A load balancing scheme for massively multiplayer online games", Multimed. Tools Appl. (2009) Vol. 45, pp.263 - 289, DOI 10.1007/s11042-009-0302-z
[8] Q. Yang, Y. Zhou, Y. Yu, J. Yuan, X. Xing, S. Du, "Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud", J Supercomput (2015) Vol.71, pp.3037 - 3053, DOI 10.1007/s11227-015-1426-8
[9] Unity3D, https://unity3d.com/kr/
[10] C. Song, F. Liu, Y. Huang, L. Wang, and T. Tan, "Auto-encoder Based Data Clustering", CIARP 2013, LNCS 8258, pp.117 - -124, 2013.
[11] Ian Goodfellow, "Generative Adversarial Networks, NIPS 2016 Tutorial, 2016
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning", NIPS 2016
[13] Jung-min Park, Hye-Young Kim, "A Study of Smart IT convergence Framework applying a Lego-typed Sensor Module", Journal of Korea Game Society, vol.16, No.3, pp. 87-96

김 차 영 (Kim, Cha Young)

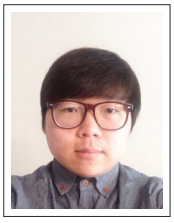
2006 고려대학교 컴퓨터학과 이학박사
2005-2008 KISTI 선임프로젝트연구원
2009- 경기대학교 컴퓨터과학과 초빙교수

관심분야: 빅데이터, 머신러닝, 딥러닝 강화학습, IoT, 클라우드 컴퓨팅, 게임개발

김 혜 영 (Kim, Hye Young)

2005 고려대학교 컴퓨터학과 이학박사
2005-2006 Wright State Uni. Post-Doc.
2014 New York State University at Albany 방문교수
2007- 홍익대학교 게임학부 부교수

관심분야: 모바일게임 온라인게임서버 게임엔진
IoT 기반의 게임 서비스 및 게임개발



박 정 민 (Park, Jung Min)

2010-2017 홍익대학교 게임학부 졸업
2017- 홍익대학교 일반대학원 게임학부(공학)재학중

관심분야: 온라인 게임 서버, 모바일 게임,게임프로그래밍