The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 24, No. 2, pp.93-98, Apr. 30, 2024. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2024.24.2.93
JIIBC 2024-2-14

딥러닝의 파일 입출력을 위한 버퍼캐시 성능 개선 연구

A Study on Improvement of Buffer Cache Performance for File I/O in Deep Learning

이정하*, 반효경**

Jeongha Lee*, Hyokyung Bahn**

요 약 인공지능과 고성능 컴퓨팅 기술이 급속히 발전하면서 다양한 분야에 딥러닝 기술이 활용되고 있다. 딥러닝은 학습 과정에서 대량의 데이터를 무작위로 읽어 학습을 진행하고, 이 과정을 반복한다. 많은 수의 파일들이 무작위로 반복 참조되는 딥러닝의 파일 입출력은 시간적 지역성을 지닌 일반적인 응용과는 다른 특징을 보인다. 이로 인한 캐싱의 어려움을 극복하기 위해 본 연구에서는 딥러닝 데이터셋 읽기의 무작위성을 줄이고 기존의 버퍼 캐시 알고리즘에 적응적으로 동작하는 새로운 데이터 읽기 방안을 제안한다. 본 논문에서는 실험을 통해 제안하는 방식이 버퍼 캐시의 미스율을 기존의 방식에 비해 평균 16%, 최대 33% 감소시키고, 수행시간을 24%까지 개선함을 보인다.

Abstract With the rapid advance in AI (artificial intelligence) and high-performance computing technologies, deep learning is being used in various fields. Deep learning proceeds training by randomly reading a large amount of data and repeats this process. A large number of files are randomly repeatedly referenced during deep learning, which shows different access characteristics from traditional workloads with temporal locality. In order to cope with the difficulty in caching caused by deep learning, we propose a new sampling method that aims at reducing the randomness of dataset reading and adaptively operating on existing buffer cache algorithms. We show that the proposed policy reduces the miss rate of the buffer cache by 16% on average and up to 33% compared to the existing method, and improves the execution time by up to 24%.

Key Words : deep learning, file I/O, dataset, file caching, sampling

Ⅰ. 서 론

최근 딥러닝과 고성능 컴퓨팅 기술의 급속한 발전으로 소프트웨어 설계에서 인공지능이 필수적인 요소가 되고 있다[1,2,3]. 특히, 모바일에서 산업용에 이르는 다양한 시스템 분야에서 인공지능 딥러닝 기술이 지속적으로 활용되고 있는 추세이다[4,5,6]. 딥러닝 시장의 규모는 급격히 증가하고 있는 추세이며, 2030년까지 연평균 36.2%의 성장률을 보일 것으로 기대되고 있다[7].

딥러닝에서 가장 중요한 것은 충분한 데이터셋의 확보이다. 딥러닝 모델이 새로운 데이터에 대해서도 높은 확률의 추론이 가능하도록 하기 위해서는 대량의 데이터를

*준회원, 이화여자대학교 컴퓨터공학과
**정회원, 이화여자대학교 컴퓨터공학과
접수일자 2024년 1월 18일, 수정완료 2024년 3월 9일
게재확정일자 2024년 4월 5일

Received: 18 January, 2024 / Revised: 9 March, 2024 / Accepted: 5 April, 2024
**Corresponding Author: bahn@ewha.ac.kr
Dept. of Computer Engineering, Ewha University, Korea

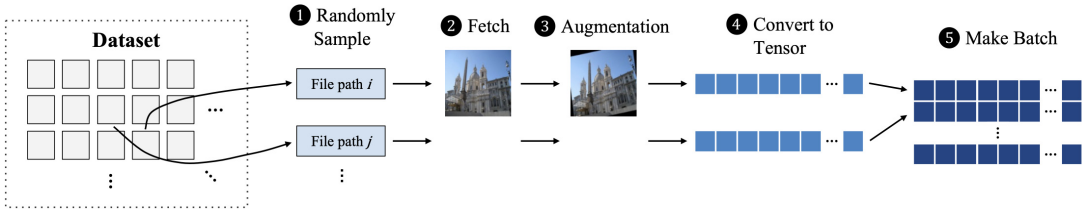


그림 1. 딥러닝 학습 중의 파일 읽기 과정
Fig. 1. File read process during deep learning training.

모델에 학습시키는 과정을 반복하는 것이 필요하다. 데이터셋 전체를 한 번에 학습하는 것은 메모리의 용량 부족 및 학습 수행시간 등의 문제가 발생하기 때문에, 데이터셋을 미니배치(mini-batch)로 나누어서 학습을 진행하는 것이 일반적이다. 이때 미니배치는 데이터셋에 대한 일반성을 높이기 위해 전체 데이터셋에서 샘플을 무작위로 추출하여 생성하는 것이 일반적이다.

그림 1은 딥러닝 학습 시 데이터셋 준비 과정을 보여주고 있다. 1) 전체 데이터셋 목록에서 무작위로 하나의 샘플을 선택하고, 2) 선택된 파일 경로에서 파일을 읽어들인 후 3) 데이터셋 증강을 위해 변형을 적용하고 4) 학습에 사용하기 위해 실수형의 텐서로 바꾼 후, 5) 여러 텐서를 결합하여 배치로 만들어 학습에 사용한다. 한 번의 학습주기(epoch) 동안 데이터셋 내의 모든 데이터 샘플에 대해 상기의 데이터 준비 및 학습을 진행하며, 여러 번의 학습주기에 걸쳐 위의 과정을 반복한다.

딥러닝의 학습 과정에서는 다수의 파일 블록에 대한 무작위적 접근을 장기간에 걸쳐 반복하는 특징을 보인다[8]. 많은 블록들이 긴 주기에 걸쳐 무작위로 재참조 되는 딥러닝 워크로드의 특성은 파일 I/O의 성능을 저하시키는 원인이 되며[9], 버퍼 캐시의 운영을 어렵게 한다. 스토리지 접근이 시스템 성능의 병목이 되는 것을 막기 위해 I/O 스케줄링 및 버퍼 캐싱에 관한 많은 연구가 있어 왔지만[10, 11], 딥러닝의 경우 기존 워크로드와 확연히 다른 참조 양상을 띠어 기존 방식의 효용을 크게 떨어뜨린다.

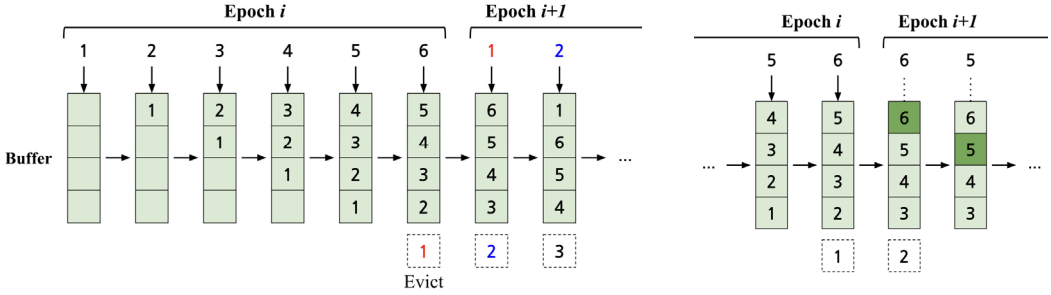
본 연구에서는 딥러닝 학습 과정에서의 파일 I/O의 특성에 대한 이해를 바탕으로, 딥러닝 파일 I/O를 효율적으로 개선하기 위한 방안을 제안한다. 딥러닝 워크로드의 데이터 블록 참조 특성을 분석하기 위해 strace[12]를 사용하여 학습 동안의 파일 참조 시스템 콜을 기록하였다. 딥러닝 라이브러리는 가장 널리 쓰이는 PyTorch 2.1.0[13]과 Torchvision 0.15.2[14]를 사용하였으며, 딥러닝 모델은 이미지 분류 문제에서 가장 대표적인 ResNet50[15]을 사용하였다. 실험에 사용한 데이터셋은 다음과 같다.

- Food101: 총 101 종류의 음식 사진으로 구성되며, 한 종류당 1000개의 사진이 존재한다[16].
- ImageNet100: 총 100 종류의 사물 사진으로 구성되며, 한 종류당 1300개의 학습용 사진과 50개의 검증용 사진이 존재한다[17]. 본 데이터셋은 ImageNet Large Scale Visual Recognition Challenge[18]의 ImageNet-1K 데이터셋 중 일부이다.

본 연구에서는 기존의 방식처럼 매 학습주기마다 데이터셋 전체를 섞는 방식이 아닌 번들(bundle)이라는 묶음 단위 안에서 무작위 읽기를 진행하여 데이터셋의 읽기 과정에서 발생하는 무작위성을 줄이고, 학습주기가 바뀔 때마다 읽기 순서를 교대로 하는 방식을 적용하여 기존 버퍼 캐시의 효율성을 높이도록 하였다. 실험을 통해, 본 논문이 제안한 방식이 딥러닝 학습 동안 파일 입출력 버퍼 캐시의 미스율을 기존 방식에 비해 최대 33%, 수행시간은 최대 24%까지 개선할 수 있음을 보인다.

II. 딥러닝 학습을 위한 파일 읽기 기법

PyTorch의 데이터 읽기에 사용되는 주요 컴포넌트에는 Dataset, Sampler, 그리고 DataLoader가 있다. Dataset 클래스는 지정 색인에 대한 데이터 샘플을 가져오는 __getitem__ 메소드를 제공한다. DataLoader는 Dataset의 샘플을 미니배치 단위로 읽어 들이는데, 이 과정에서 데이터셋 샘플의 일부를 추출하는 Sampler를 사용한다[19]. 사용자가 DataLoader의 shuffle 매개변수를 참(true)으로 설정하면, DataLoader는 내부적으로 RandomSampler를 호출하여 전체 데이터샘플의 색인을 무작위로 추출하고, 이 순서대로 데이터셋 읽기를 진행한다.



(a) 순차 읽기 (b) 번갈아 읽기

그림 2. LRU 교체 알고리즘의 동작 과정
Fig. 2. Process of the LRU replacement algorithm.

1. 묶음 단위의 무작위 읽기

본 연구에서는 기존의 RandomSampler를 변형한 BundleRandomSampler라는 클래스를 구현하여 묶음 단위의 무작위 읽기를 수행하도록 하였다. 이 클래스는 입력으로 주어진 크기로 데이터셋의 묶음을 생성하고, 해당 묶음 내의 데이터 샘플 안에서 무작위 추출을 진행한다. 이러한 동작 방식은 데이터셋의 전체 목록을 무작위로 추출하는 RandomSampler나 주어진 색인 목록에 대해서만 추출하는 기존의 SubsetRandomSampler와 차별화된다.

본 연구에서는 BundleRandomSampler의 묶음 크기를 DataLoader 호출 시 사용자가 지정할 수 있도록 DataLoader에 bundle_ratio라는 추가 매개변수를 정의하였다. bundle_ratio는 0과 1 사이의 실수값으로, bundle_ratio가 0.1이라면 한 묶음은 전체 데이터셋 크기의 10%가 되고 데이터셋은 총 10개의 묶음으로 구성된다. bundle의 크기는 셔플을 위한 개념이므로, 이는 데이터셋 샘플의 수로 나타내는 것보다 데이터셋 전체의 크기에 비례하여 결정하는 것이 자연스럽다. 즉, 사용자가 DataLoader 호출 시 bundle_ratio에 이 비율을 입력하면, DataLoader 내부에서 bundle의 크기를 계산하고, 데이터 셔플 시 BundleRandomSampler에 이 값을 전달하여 묶음 단위의 데이터셋 셔플을 진행한다.

2. 묶음 단위의 번갈아 읽기

대용량 데이터셋으로 구성된 딥러닝 학습에서는 매 학습주기마다 방대한 데이터를 참조하기 때문에 새로운 학습주기가 시작되었을 때 이전 학습주기의 데이터가 이미 버퍼 캐시에서 쫓겨나 캐싱의 효과가 크게 떨어진다. 이러한 문제점을 극복하기 위해서는 학습시 묶음 간의 참조 순서를 고려하는 것이 필요하다. 버퍼 캐시의 교체 정책으로 가장 널리 사용되고 있는 LRU(Least Recently Used) 알고리즘의 경우 가장 오래전에 참조된 블록을 캐시에서 방출하므로 학습주기마다 반복적으로 참조가 이루어지는 딥러닝의 학습에서는 비효율적이다.

이에 대해 그림을 통해 그 비효율성을 살펴보도록 하겠다. 그림 2(a)에서 전체 데이터셋은 6개의 묶음으로 구성되며, 버퍼 캐시의 크기가 데이터 묶음 4개를 동시에 수용할 수 있는 상황을 가정한 것이다. 그림에서 학습주기 i에서 묶음 5와 묶음 6을 캐싱할 때, 캐시에 있던 묶음 1과 묶음 2가 캐시에서 쫓겨난다. 그러나, 학습주기 $i+1$ 시작 시 직전에 쫓겨난 묶음 1과 묶음 2를 다시 참조하므로 캐싱의 효율성이 크게 떨어진다.

본 연구에서는 학습시 데이터 참조가 버퍼 캐시의 이러한 정책에 적응적으로 동작하도록 묶음 단위의 번갈아 읽기 방법을 제안한다. 제안하는 방식은 각 학습주기마다 데이터셋 읽기의 순서를 앞 학습주기와 반대로 하는 것이다. 상기의 예시에서 번갈아 읽기를 적용할 경우 6개의 묶음을 학습주기 1에서 1, 2, …, 5, 6의 순서로 읽었다면, 학습주기 2에서는 6, 5, …, 2, 1의 순서로 읽는 것이다. 묶음 번갈아 읽기 방식의 목적은, 하나의 학습주기가 끝나고 다음 학습주기가 시작하는 시점에서, 바로 이전 학습주기에서 마지막으로 사용한 데이터셋 묶음을 학습 대상으로 삼아 캐싱의 효과를 높이고자 함이다. 그림 2(b)에서 보는 것처럼 이미 버퍼 캐시에 저장된 묶음부터 읽기를 진행하므로 학습 시 파일 읽기에 소요되는 시간을 줄일 수 있다.

그림 3은 앞서 제안한 묶음 단위 무작위 읽기와 학습주기 간 번갈아 읽기 방식을 사용하여 데이터 읽기를 수행하는 과정의 예시를 나타낸 것이다. 학습주기 $2i-1$에서는 묶음들을 1, 2, 3, 4의 순서로, 학습주기 $2i$에서는 4, 3, 2, 1의 순서로 읽어서 학습하며, 데이터 묶음 내에서는 무작위로 읽는 방식을 사용한다.

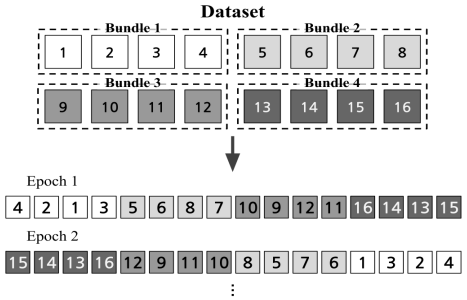


그림 3. 묶음 단위의 번갈아 읽기
Fig. 3. Alternated reading in bundles.

Ⅲ. 성능 평가

1. 기존 버퍼 캐시와의 성능 비교

제안하는 방식이 기존의 버퍼 캐시 알고리즘에도 적합하게 동작하는지 확인하기 위해, 기존의 방법과 본 연구의 제안 방법을 적용한 경우 각각에 대하여 LRU 알고리즘을 이용한 캐시 시뮬레이션을 통해 미스율의 변화를 확인하였다.

그림 4는 참조한 모든 파일 블록의 수에 비례하여 버퍼 캐시의 크기를 변화시키면서 각각의 미스율을 비교한 것이다. x축은 캐시의 크기를 나타내며, y축은 전체 블록 참조 대비 미스율을 나타내며, 기존 방식의 미스율 대비 상대적인 수치로 표현한 것이다.

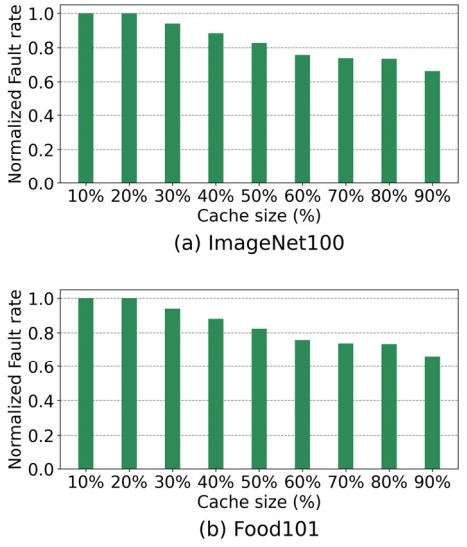


그림 4. LRU 버퍼 캐시의 미스율 비교
Fig. 4. Comparison of miss rates in LRU cache.

그림에서 볼 수 있듯이, 제안하는 방식은 대부분의 경우에서 기존 방식에 비해 개선된 미스율을 나타내었다. ImageNet100 데이터셋의 경우 최대 33.6%, 평균 16.2%, Food101 데이터 셋의 경우 최대 33.9%, 평균 16.3%의 미스율 개선을 나타내었다.

2. 학습 성능

데이터셋을 셔플링하는 목적은 한번 학습하는 데에 사용되는 단위인 미니배치의 일반성을 높여 더 효과적인 모델 학습을 하기 위함이다. 따라서 본 논문이 제안하는 번들 단위 무작위 추출 방식이 수행시간을 줄일 뿐만 아니라, 학습의 성능에 차이가 없음이 보장되어야 한다. 이를 확인하기 위해 본 절에서는 bundle_ratio를 조절하며 학습을 진행하여 그 결과를 확인하였다.

그림 5는 시간이 흐름에 따라 학습의 정확도가 기존 방식과 제안하는 방식에서 번들 크기가 바뀜에 따라 어떻게 달라지는지를 실험한 것이다. 그림에서 x축은 수행시간, y축은 학습 정확도를 나타낸 것이며, 학습주기는 30으로 하여 실험한 결과이다. 두 모델 모두 제안하는 방식이 극단적인 bundle_ratio인 0.01로 설정한 경우를 제외하고는 기존 방식과 학습 정확도 측면에서 큰 차이가 없었으며, 동일한 학습 정확도를 얻기 위한 수행시간이 일정 수준 개선되는 것을 확인할 수 있었다. 구체적으로 살펴보면 수행시간이 기존 방식 대비 ImageNet100 데이터셋의 경우 최대 16%, Food101 데이터셋의 경우 최대 24% 개선되는 것을 확인할 수 있었다.

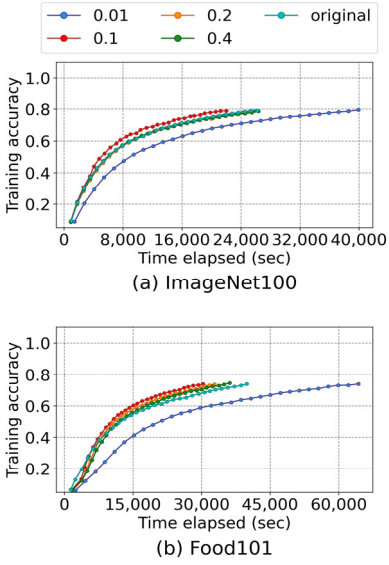


그림 5. 학습 성능 비교
Fig. 5. Comparison of training performance.

Ⅳ. 결 론

딥러닝은 모델 학습을 위해 대용량 데이터를 입력으로 활용하고 이를 반복적으로 학습시켜 성능을 높이는 것을 목표로 한다. 모델의 정확도를 위해서는 충분한 양의 데이터셋 확보가 중요하지만 데이터셋의 크기 증가는 딥러닝 학습의 새로운 병목이 되기도 한다. 본 연구에서는 일반 응용과 다른 딥러닝 학습 과정에서의 파일 입출력 특성을 분석하고, 이를 효율적으로 개선하기 위해 묶음 단위의 무작위 읽기와 학습주기 간 읽기 순서 교대 방식을 제안하였다. 본 연구에서 제안한 방식을 이용하여 딥러닝 학습에서의 파일 접근시 버퍼 캐시 미스율을 기존 방식 대비 최대 33% 개선함을 확인할 수 있었으며, 수행시간은 최대 24% 개선할 수 있음을 확인하였다.

References

[1] S. Dargan, M. Kumar, M. Ayyagari, and G. Kumar, "A survey of deep learning and its applications: a new paradigm to machine learning," Archives of Computational Methods in Engineering, vol. 27, pp. 1071-1092, 2020.
DOI: https://doi.org/10.1007/s11831-019-09344-w

[2] K. Lee, H. Jung, and S. Lee, "Anomaly detection method of user trajectories based on deep learning technologies," JKIIT, vol. 20, no. 11, pp. 101-116, 2022.
DOI: https://doi.org/10.14801/jkiit.2022.20.11.101

[3] S. Kim, W. Hur, and J. Ahn, "A progressive web application for mobile crop disease diagnostics based on transfer learning," Journal of the Korea Academia-Industrial cooperation Society (JKAIS), vol. 23, no. 2 pp. 22-29, 2022.
DOI: https://doi.org/10.5762/KAIS.2022.23.2.22

[4] S. Ki, G. Byun, K. Cho, and H. Bahn, "Co-optimizing CPU voltage, memory placement, and task offloading for energy-efficient mobile systems," IEEE Internet of Things Journal, vol. 10, no. 10, pp. 9177-9192, 2023.
DOI: https://doi.org/10.1109/JIOT.2022.3233830

[5] D. Kim, S. Lee, and H. Bahn, "An adaptive location detection scheme for energy-efficiency of smartphones," Pervasive and Mobile Computing, vol. 31, pp. 67-78, 2016.
DOI: https://doi.org/10.1016/j.pmcj.2016.04.012

[6] S. Yoo, Y. Jo, and H. Bahn, "Integrated scheduling of real-time and interactive tasks for configurable industrial systems," IEEE Transactions on Industrial Informatics, vol. 18, no. 1, pp. 631-641, 2022.
DOI: https://doi.org/10.1109/TII.2021.3067714

[7] Fortune Business Insights, Artificial Intelligence Market Size, Share & COVID-19 Impact Analysis, 2023-2030, 2023, Available online: https://www.fortunebusinessinsights.com/industry-reports/artificial-intelligence-market-100114.

[8] Z. Zhang, L. Huang, U. Manor, L. Fang, G. Merlo, C. Michoski, J. Cazes, and N. Gaffney, "FanStore: enabling efficient and scalable I/O for distributed deep learning", arXiv:1809.10799, 2018.
DOI: https://doi.org/10.48550/arXiv.1809.10799

[9] T. Nguyen, F. Trahay, J. Domke, A. Drozd, E. Vatai, J. Liao, M. Wahib, and B. Gerofi, "Why globally re-shuffle? revisiting data shuffling in large scale deep learning," Proc. IEEE Parallel and Distributed Processing Symp., pp. 1085-1096, 2022.
DOI: https://doi.org/10.1109/IPDPS53621.2022.00109

[10] T. Kim and H. Bahn, "Implementation of the storage manager for an IPTV set-top box," IEEE Trans. on Consumer Electronics, vol. 54, no. 4, pp. 1770-1775, 2008.
DOI: https://doi.org/10.1109/TCE.2008.4711233

[11] O. Kwon, H. Bahn, and K. Koh, "Popularity and prefix aware interval caching for multimedia streaming servers," Proc. IEEE Conf. on Computer and Information Technology, pp. 555-560, 2008.
DOI: https://doi.org/10.1109/CIT.2008.4594735

[12] Strace, https://strace.io/

[13] PyTorch, https://pytorch.org/

[14] Torchvision, https://pytorch.org/vision/stable

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp. 770-778, 2016.
DOI: https://doi.org/10.1109/CVPR.2016.90

[16] Food101 Dataset, https://data.vision.ee.ethz.ch/cvl/datasets_extra/food-101

[17] ImageNet100 Dataset, https://www.kaggle.com/datasets/ambityga/imagenet100

[18] ImageNet-1k Dataset, https://www.image-net. org/challenges/LSVRC/2012

[19] R. Khan, A. Yazdani, Y. Fu, A. Paul, B. Ji, X. Jian, Y. Cheng, and A. Bu, "SHADE: enable fundamental cacheability for distributed deep learning training," Proc. USENIX Conf. on File and Storage Technologies, pp. 135-152, 2023.

저 자 소 개

이 정 하(준회원)

- 2022년 2월 : 이화여자대학교 컴퓨터공학전공 학사
- 2022년 3월 ~ : 이화여자대학교 컴퓨터공학전공 대학원생

반 효 경(정회원)

- 1997년 2월 : 서울대학교 계산통계학과 학사
- 1999년 2월 : 서울대학교 전산과학과 석사
- 2002년 2월 : 서울대학교 컴퓨터공학부 박사
- 2002년 9월 ~ : 이화여자대학교 컴퓨터공학과 교수