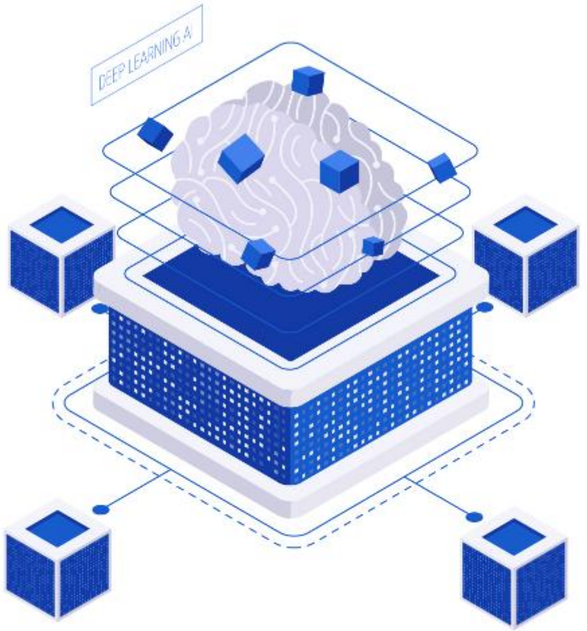
Project Proposal

A Survey on Memory Optimization Techniques for On-Device Learning in AIoT Systems



AIoT Team 06
202020708 김평주
202020820 남현원

00 AIoT Team 6

Members

Nam Hyeon-won (남현원)

Department of Software
202020820

Kim Pyung-joo(김평주)

Department of Software
202020708

01 Motivation

AIoT device growth boosts demand for on-device ML in edge computing

02 Survey

1. Melon
2. Sage
3. Dropback

03 Aspects it link to AIOT

What does efficient on-device training bring to AIoT

04 After proposal

More detail about selected papers and compare each other

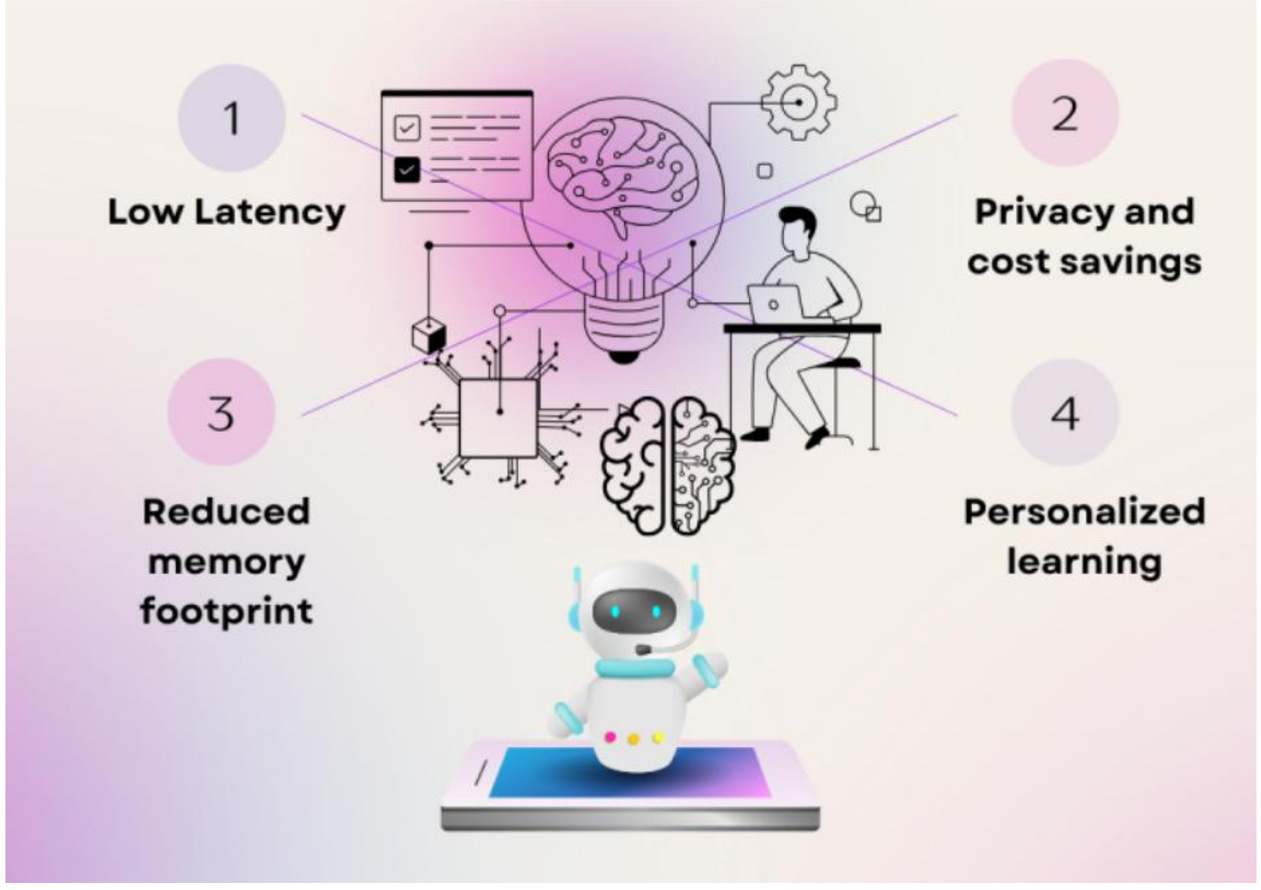
01 Motivation

AIoT device growth boosts demand for **on-device ML** in edge computing.

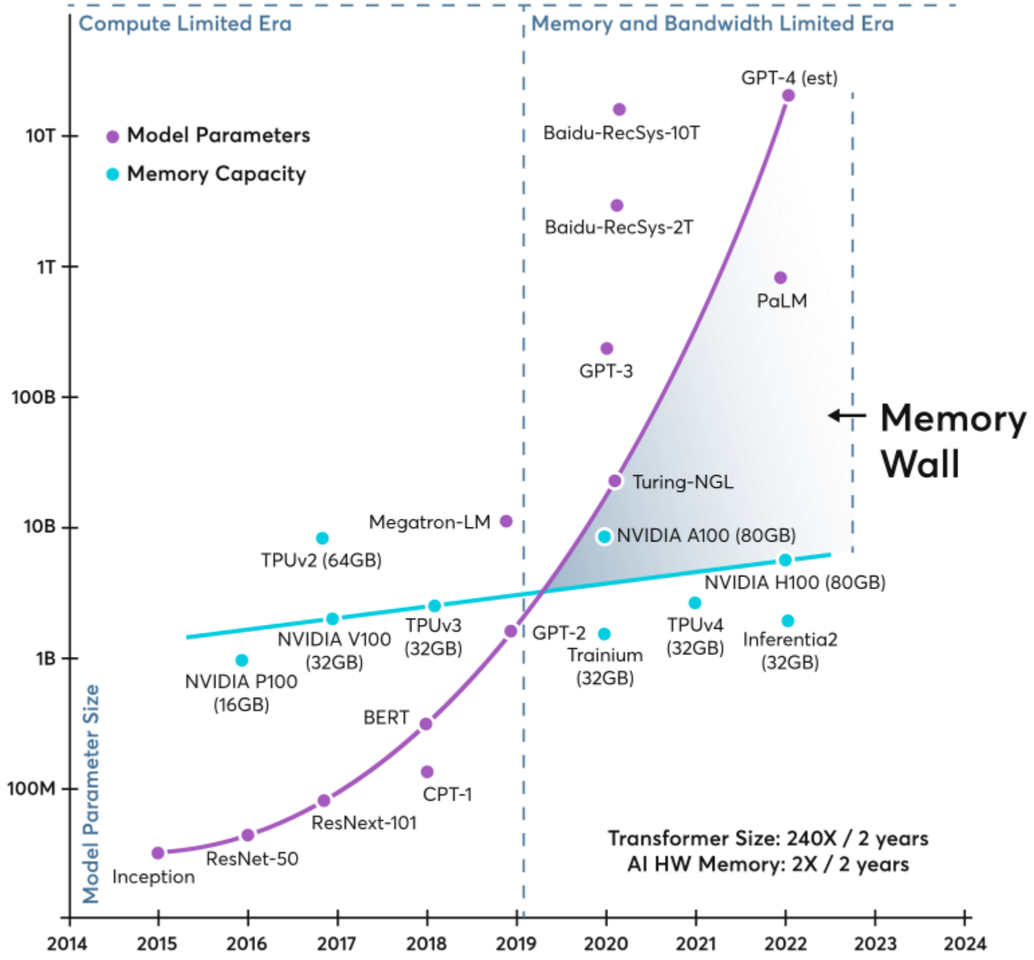
- On-device training is essential for advanced learning scenarios such as **federated learning** and **on-device transfer learning** in edge environments.
- The demand for **on-device training is growing** due to rising public concerns over **data privacy** and regulations like GDPR.

01 Motivation

4 Benefits of On-device training



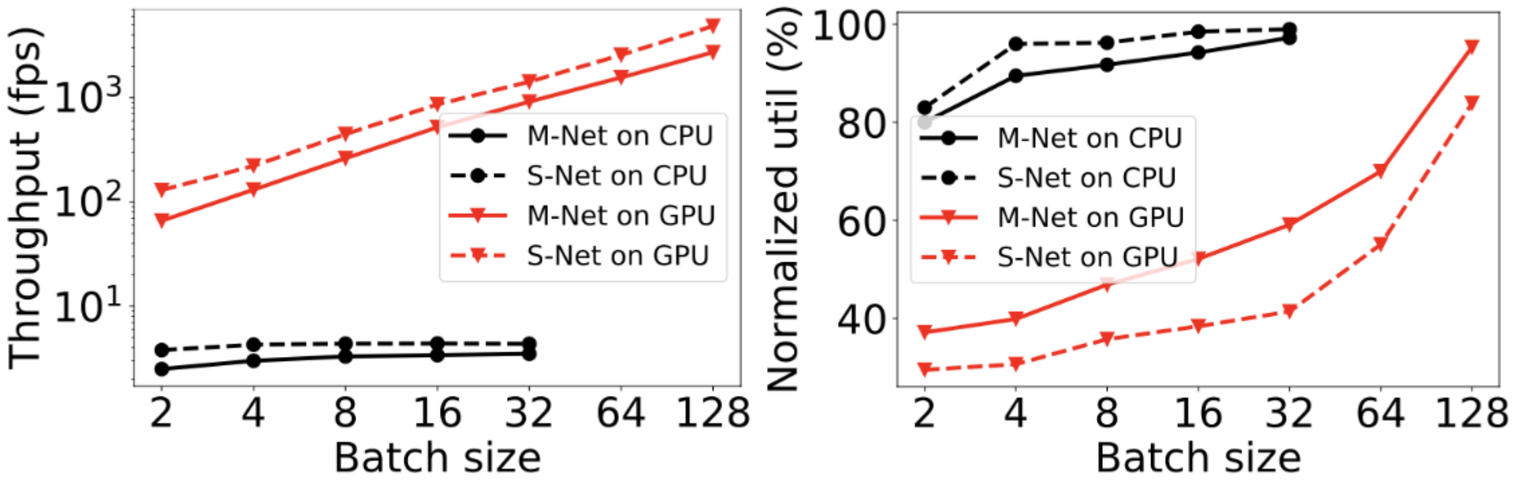
01 Motivation



- The **"Memory Wall"** arises from a widening performance gap between processor improvements and DRAM advancements.

- Processors improve at a rate of 3.0x every two years, while DRAM only advances by 1.6x, making **memory access the primary bottleneck in computing performance**.

01 Motivation



Experiments showing up to 45.73% faster convergence and 3.94% higher accuracy when using a batch size of 128 compared to 32

- **Larger batch sizes** are essential for improving both convergence speed and accuracy in DNN training.
- However, even flagship mobile devices, such as the Samsung Note 10 (8GB RAM), **cannot support large-batch training due to memory limitations**, leading to lower accuracy and slower training.

02 Survey

Various solutions have been proposed to **overcome the "Memory Wall"** in on-device training.

- **Melon**: Breaking the memory wall for resource-efficient on-device machine learning
- **Sage**: Memory-efficient DNN training on mobile devices
- **Dropback**: Full deep neural network training on a pruned weight budget

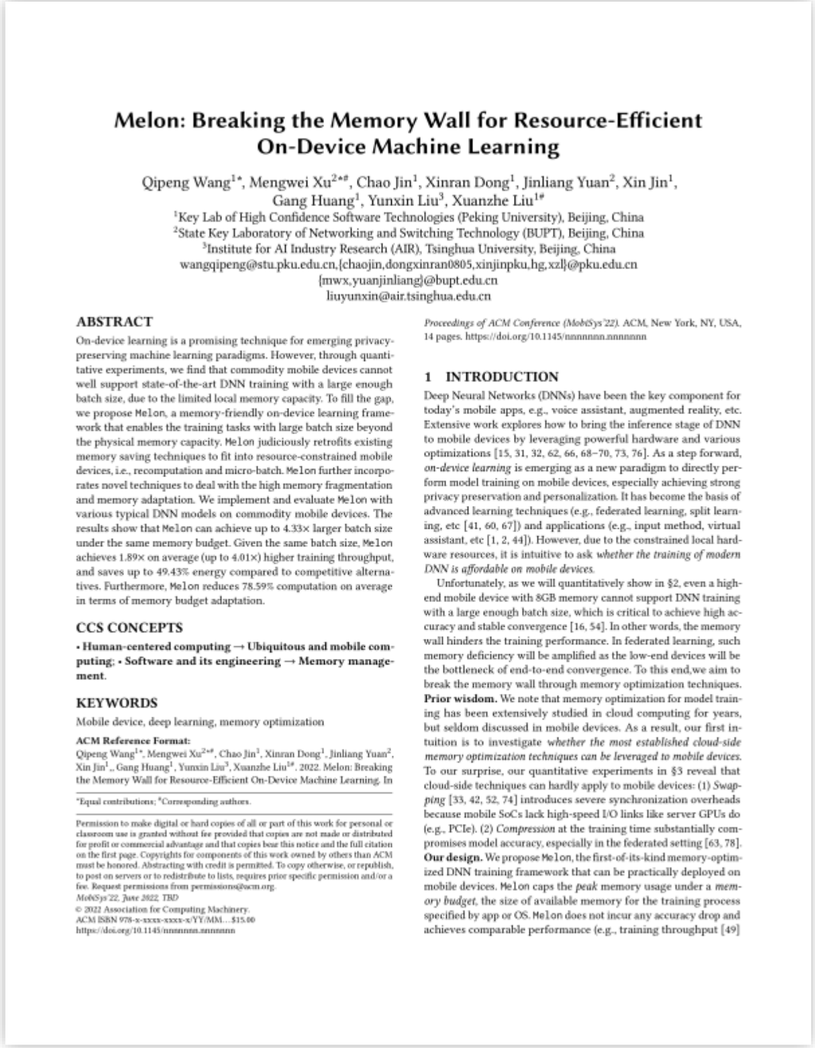
02 Survey - Melon

Melon: breaking the memory wall for resource-efficient on-device machine learning

Authors: Qipeng Wang, Mengwei Xu, Chao Jin, Xinran Dong, Jinliang Yuan, Xin Jin, Gang Huang, Yunxin Liu, Xuanzhe Liu

Published: 27 June 2022

Proposing "Melon", a groundbreaking method that enables training with **batch sizes up to 4.33× larger** in memory-constrained environments.



Melon: Breaking the Memory Wall for Resource-Efficient On-Device Machine Learning

Qipeng Wang[1*], Mengwei Xu[2*#], Chao Jin[1], Xinran Dong[1], Jinliang Yuan[2], Xin Jin[1], Gang Huang[1], Yunxin Liu[3], Xuanzhe Liu[1#]
[1]Key Lab of High Confidence Software Technologies (Peking University), Beijing, China
[2]State Key Laboratory of Networking and Switching Technology (BUPT), Beijing, China
[3]Institute for AI Industry Research (AIR), Tsinghua University, Beijing, China
wangqipeng@stu.pku.edu.cn,{chaojin,dongxinran0805,xinjinpku,hg,xzl}@pku.edu.cn
{mwx,yuanjinliang}@bupt.edu.cn
liuyunxin@air.tsinghua.edu.cn

ABSTRACT

On-device learning is a promising technique for emerging privacy-preserving machine learning paradigms. However, through quantitative experiments, we find that commodity mobile devices cannot well support state-of-the-art DNN training with a large enough batch size, due to the limited local memory capacity. To fill the gap, we propose Melon, a memory-friendly on-device learning framework that enables the training tasks with large batch size beyond the physical memory capacity. Melon judiciously retrofits existing memory saving techniques to fit into resource-constrained mobile devices, i.e., recomputation and micro-batch. Melon further incorporates novel techniques to deal with the high memory fragmentation and memory adaptation. We implement and evaluate Melon with various typical DNN models on commodity mobile devices. The results show that Melon can achieve up to 4.33× larger batch size under the same memory budget. Given the same batch size, Melon achieves 1.89× on average (up to 4.01×) higher training throughput, and saves up to 49.43% energy compared to competitive alternatives. Furthermore, Melon reduces 78.59% computation on average in terms of memory budget adaptation.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Software and its engineering** → **Memory management**.

KEYWORDS

Mobile device, deep learning, memory optimization

ACM Reference Format:

Qipeng Wang[1*], Mengwei Xu[2*#], Chao Jin[1], Xinran Dong[1], Jinliang Yuan[2], Xin Jin[1], Gang Huang[1], Yunxin Liu[3], Xuanzhe Liu[1#]. 2022. Melon: Breaking the Memory Wall for Resource-Efficient On-Device Machine Learning. In *Proceedings of ACM Conference (MobiSys'22).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Equal contributions; #Corresponding authors.

MobiSys'22, June 2022, TBD

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Deep Neural Networks (DNNs) have been the key component for today's mobile apps, e.g., voice assistant, augmented reality, etc. Extensive work explores how to bring the inference stage of DNN to mobile devices by leveraging powerful hardware and various optimizations [15, 31, 32, 62, 66, 68–70, 73, 76]. As a step forward, *on-device learning* is emerging as a new paradigm to directly perform model training on mobile devices, especially achieving strong privacy preservation and personalization. It has become the basis of advanced learning techniques (e.g., federated learning, split learning, etc [41, 60, 67]) and applications (e.g., input method, virtual assistant, etc [1, 2, 44]). However, due to the constrained local hardware resources, it is intuitive to ask *whether the training of modern DNN is affordable on mobile devices.*

Unfortunately, as we will quantitatively show in §2, even a high-end mobile device with 8GB memory cannot support DNN training with a large enough batch size, which is critical to achieve high accuracy and stable convergence [16, 54]. In other words, the memory wall hinders the training performance. In federated learning, such memory deficiency will be amplified as the low-end devices will be the bottleneck of end-to-end convergence. To this end,we aim to break the memory wall through memory optimization techniques.

Prior wisdom. We note that memory optimization for model training has been extensively studied in cloud computing for years, but seldom discussed in mobile devices. As a result, our first intuition is to investigate *whether the most established cloud-side memory optimization techniques can be leveraged to mobile devices.* To our surprise, our quantitative experiments in §3 reveal that cloud-side techniques can hardly apply to mobile devices: (1) *Swapping* [33, 42, 52, 74] introduces severe synchronization overheads because mobile SoCs lack high-speed I/O links like server GPUs do (e.g., PCIe). (2) *Compression* at the training time substantially compromises model accuracy, especially in the federated setting [63, 78].

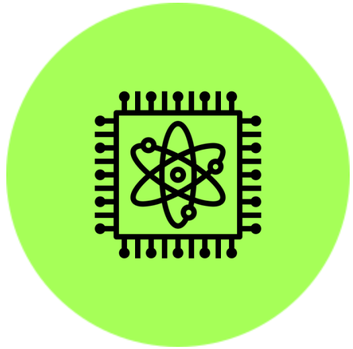
Our design. We propose Melon, the first-of-its-kind memory-optimized DNN training framework that can be practically deployed on mobile devices. Melon caps the *peak* memory usage under a *memory budget*, the size of available memory for the training process specified by app or OS. Melon does not incur any accuracy drop and achieves comparable performance (e.g., training throughput [49]

02 Survey - Melon

Key Technologies

Micro-batch

Reduces memory usage by **dividing the existing mini batch** into smaller micro batches



Recomputation

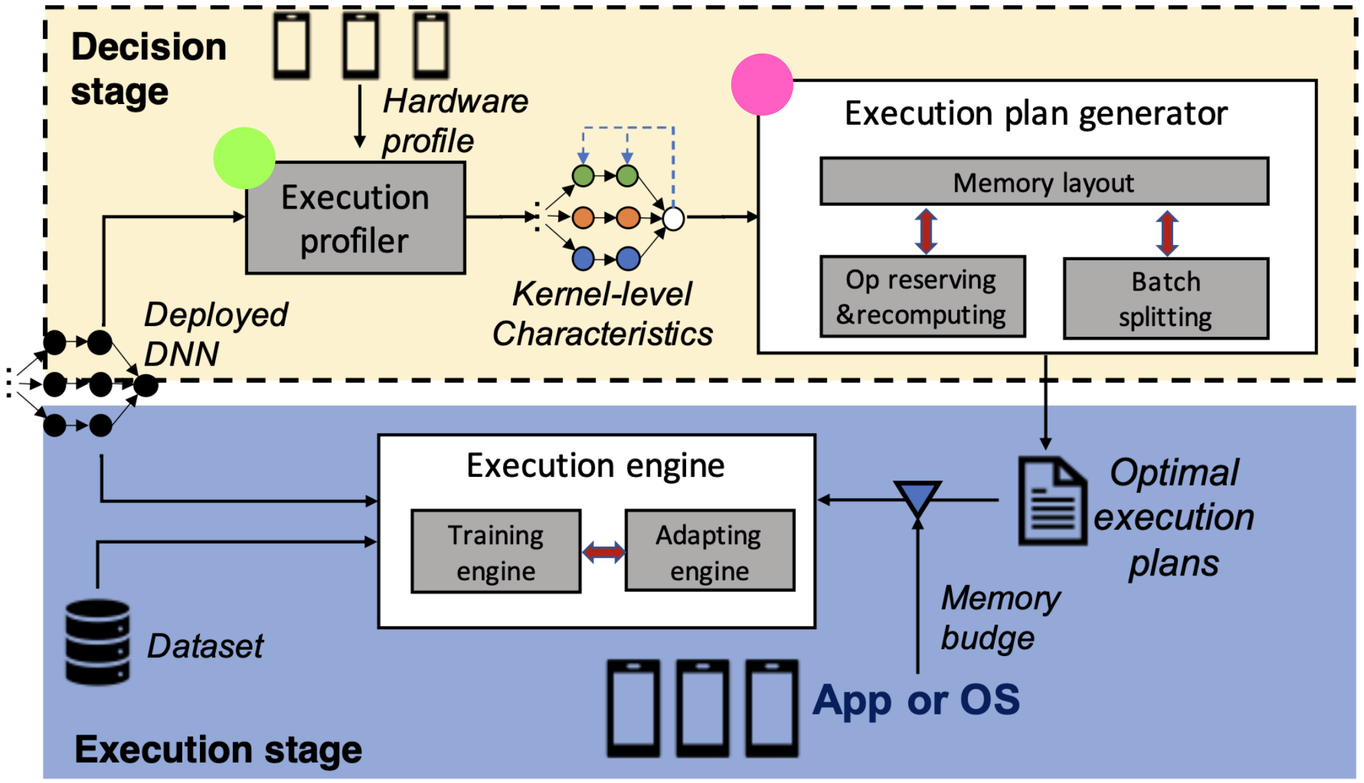
Saves memory by **recomputing intermediate tensors**

Lifetime-aware Memory Pool

Reduces fragmentation through memory allocation based on tensor lifespan

02 Survey - Melon

Decision stage

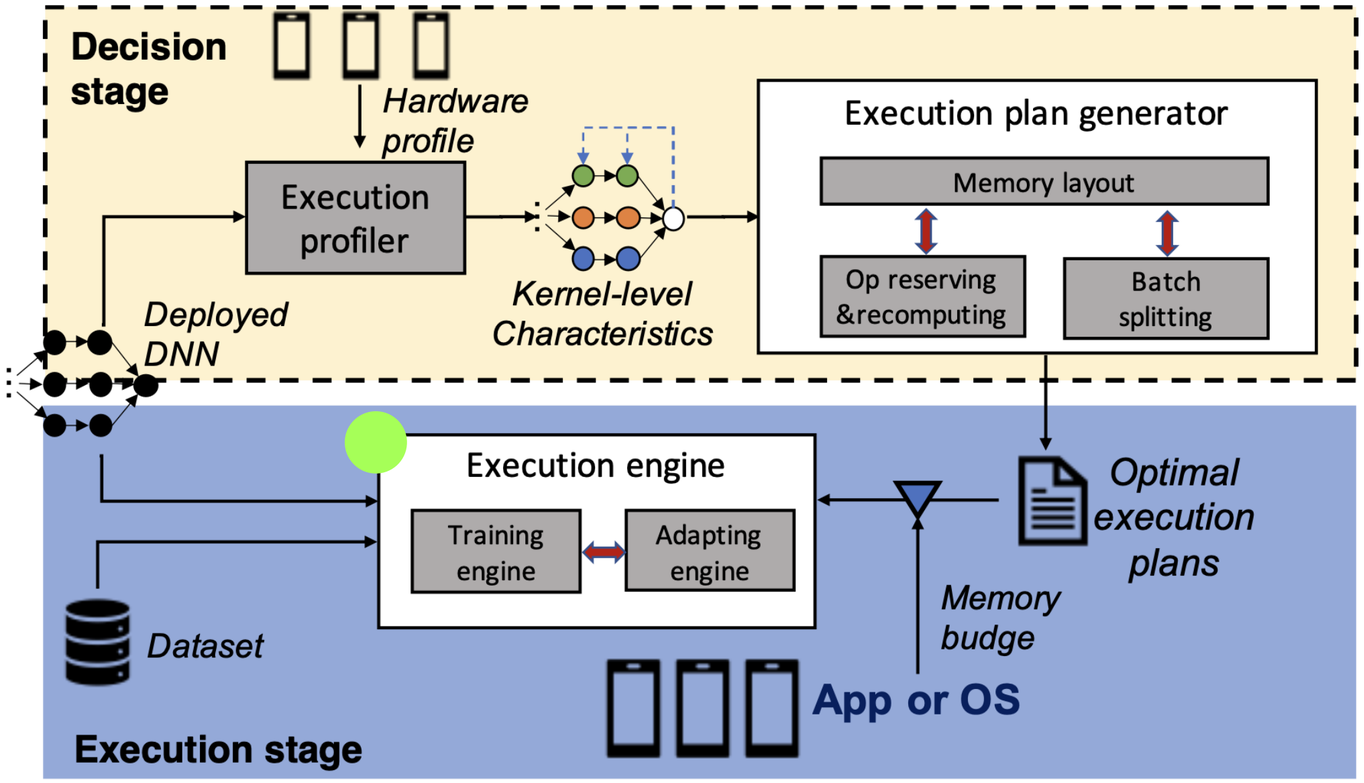


Execution profiler
Gathers hardware profile and kernel-level characteristics of the deployed DNN, including tensor sizes and computation times.

Execution plan generator
Uses this data to create optimal execution plans, managing memory layout, deciding which operations to recompute, and determining batch splitting strategies.

02 Survey - Melon

Execution stage



Execution engine

Executes training based on the generated plans, adjusting dynamically to the memory budget provided by the app or OS.

02 Survey - Melon

Key Achievements

4.33×

Memory Efficiency

4.33× larger batch size under the same memory budget

3.48×

Training Performance

Improves convergence speed by up to 3.48×

49.4%

Resource Efficiency

Reduces energy consumption by up to 49.43%

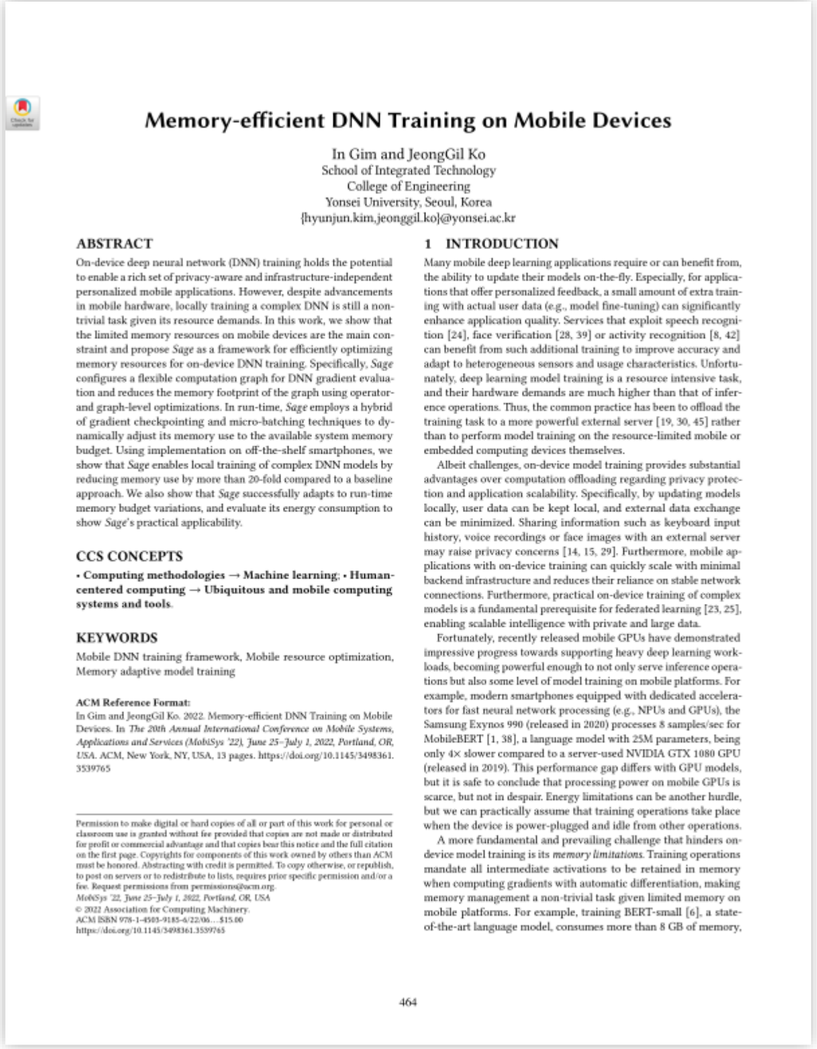
02 Survey - Sage

Memory-efficient DNN Training on Mobile Devices

Authors: In Gim and JeongGil Ko

Published: 27 June 2022

Proposing "Sage," a memory-efficient framework that enables **on-device DNN training** by reducing memory usage, while dynamically adapting to **real-time** memory constraints.



Memory-efficient DNN Training on Mobile Devices

In Gim and JeongGil Ko
School of Integrated Technology
College of Engineering
Yonsei University, Seoul, Korea
{hyunjun.kim,jeonggil.ko}@yonsei.ac.kr

ABSTRACT

On-device deep neural network (DNN) training holds the potential to enable a rich set of privacy-aware and infrastructure-independent personalized mobile applications. However, despite advancements in mobile hardware, locally training a complex DNN is still a non-trivial task given its resource demands. In this work, we show that the limited memory resources on mobile devices are the main constraint and propose *Sage* as a framework for efficiently optimizing memory resources for on-device DNN training. Specifically, *Sage* configures a flexible computation graph for DNN gradient evaluation and reduces the memory footprint of the graph using operator- and graph-level optimizations. In run-time, *Sage* employs a hybrid of gradient checkpointing and micro-batching techniques to dynamically adjust its memory use to the available system memory budget. Using implementation on off-the-shelf smartphones, we show that *Sage* enables local training of complex DNN models by reducing memory use by more than 20-fold compared to a baseline approach. We also show that *Sage* successfully adapts to run-time memory budget variations, and evaluate its energy consumption to show *Sage*'s practical applicability.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**.

KEYWORDS

Mobile DNN training framework, Mobile resource optimization, Memory adaptive model training

ACM Reference Format:

In Gim and JeongGil Ko. 2022. Memory-efficient DNN Training on Mobile Devices. In *The 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys '22), June 25–July 1, 2022, Portland, OR, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3498361.3539765

MobiSys '22, June 25–July 1, 2022, Portland, OR, USA

ACM ISBN 978-1-4503-9185-6/22/06…$15.00
https://doi.org/10.1145/3498361.3539765

1 INTRODUCTION

Many mobile deep learning applications require or can benefit from, the ability to update their models on-the-fly. Especially, for applications that offer personalized feedback, a small amount of extra training with actual user data (e.g., model fine-tuning) can significantly enhance application quality. Services that exploit speech recognition [24], face verification [28, 39] or activity recognition [8, 42] can benefit from such additional training to improve accuracy and adapt to heterogeneous sensors and usage characteristics. Unfortunately, deep learning model training is a resource intensive task, and their hardware demands are much higher than that of inference operations. Thus, the common practice has been to offload the training task to a more powerful external server [19, 30, 45] rather than to perform model training on the resource-limited mobile or embedded computing devices themselves.

Albeit challenges, on-device model training provides substantial advantages over computation offloading regarding privacy protection and application scalability. Specifically, by updating models locally, user data can be kept local, and external data exchange can be minimized. Sharing information such as keyboard input history, voice recordings or face images with an external server may raise privacy concerns [14, 15, 29]. Furthermore, mobile applications with on-device training can quickly scale with minimal backend infrastructure and reduces their reliance on stable network connections. Furthermore, practical on-device training of complex models is a fundamental prerequisite for federated learning [23, 25], enabling scalable intelligence with private and large data.

Fortunately, recently released mobile GPUs have demonstrated impressive progress towards supporting heavy deep learning workloads, becoming powerful enough to not only serve inference operations but also some level of model training on mobile platforms. For example, modern smartphones equipped with dedicated accelerators for fast neural network processing (e.g., NPUs and GPUs), the Samsung Exynos 990 (released in 2020) processes 8 samples/sec for MobileBERT [1, 38], a language model with 25M parameters, being only 4× slower compared to a server-used NVIDIA GTX 1080 GPU (released in 2019). This performance gap differs with GPU models, but it is safe to conclude that processing power on mobile GPUs is scarce, but not in despair. Energy limitations can be another hurdle, but we can practically assume that training operations take place when the device is power-plugged and idle from other operations.

A more fundamental and prevailing challenge that hinders on-device model training is its *memory limitations*. Training operations mandate all intermediate activations to be retained in memory when computing gradients with automatic differentiation, making memory management a non-trivial task given limited memory on mobile platforms. For example, training BERT-small [6], a state-of-the-art language model, consumes more than 8 GB of memory,

02 Survey - Sage

Key Technologies

Automatic Differentiation

Constructs a unified computational graph, enabling graph-level optimizations

Graph-level Optimization

Minimizes memory usage through operator fusion and subgraph reduction

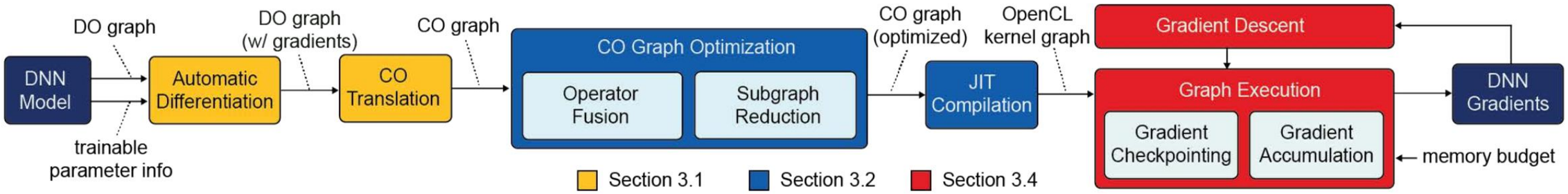
Operator-level Optimization

Manually tunes memory-intensive operations and reduces memory transfer operations

Runtime Memory Management

Employs a hybrid approach combining gradient checkpointing and micro-batching

02 Survey - Sage



Input Processing

- Takes a DNN Model with trainable parameter information
- Converts it through Automatic Differentiation into a DO graph

Optimization Stage

- Operator Fusion: Combines multiple operations to reduce memory usage
- Subgraph Reduction: Eliminates redundant computations

Execution Stage

- JIT Compilation converts optimized CO graph into OpenCL kernel graph
- Gradient Descent and Graph Execution run with Gradient Checkpointing/Accumulation

* CO: Computable Operation
* DO: Differentiable operation

02 Survey - Sage

Key Achievements

20×

Memory Efficiency

Reduces memory usage by over 20× compared to baseline approaches

Runtime Adaptability

Successfully adapts to **real-time changes** in memory availability

02 Survey - Dropback

Dropback - Full Deep Nueral Network Training on A pruned weight budget

Authors: Maximilian Golub, Guy Lemieux, Mieszko Lis

Published: 23 Nov 2019

Proposing “Dropback”, the method of storing only the weights with **high accumulated gradients** in the memory tracking set.

FULL DEEP NEURAL NETWORK TRAINING ON A PRUNED WEIGHT BUDGET

Maximilian Golub [1 2] Guy Lemieux [1] Mieszko Lis [1]

ABSTRACT

We introduce a DNN training technique that learns only a fraction of the full parameter set without incurring an accuracy penalty. To do this, our algorithm constrains the total number of weights updated during backpropagation to those with the highest total gradients. The remaining weights are not tracked, and their initial value is regenerated at every access to avoid storing them in memory. This can dramatically reduce the number of off-chip memory accesses during both training and inference, a key component of the energy needs of DNN accelerators. By ensuring that the total weight diffusion remains close to that of baseline unpruned SGD, networks pruned using our technique are able to retain state-of-the-art accuracy across network architectures — including networks previously identified as difficult to compress, such as Densenet and WRN. With ResNet18 on ImageNet, we observe an 11.7× weight reduction with no accuracy loss, and up to 24.4× with a small accuracy impact.

1 INTRODUCTION

On-device inference has become an important market, and a number of vendors offer low-power deep neural network accelerators that reduce computation costs to specifically target mobile and embedded applications (Intel, 2017; Samsung, 2018; Qualcomm, 2017; Kingsley-Hughes, 2017). To enable inference despite the comparatively smaller memories and reduced memory bandwidths in mobile devices — an order of magnitude less capacity and two orders of magnitude less bandwidth than a datacentre-class GPU — many researchers have proposed pruning, quantizing, and compressing neural networks, often trading off accuracy versus network size (LeCun et al., 1990; Hassibi et al., 1993; Han et al., 2015; 2016b; Wu et al., 2015; Choi et al., 2016; Alvarez & Salzmann, 2017; Ge et al., 2017; Zhou et al., 2017; Luo et al., 2017; Yang et al., 2017, etc.).

Comparatively little attention, however, has been paid to the problem of on-device *training*, which has so far been limited to simple models (e.g., Apple, 2017). Training takes much more energy than inference: one iteration on a single sample involves roughly thrice as many weight accesses, and typically many iterations on many samples are required. Off-chip memory accesses dominate, costing hundreds of times more energy than computation: in a 45nm process, for example, accessing a 32-bit value from DRAM costs over 700× more energy than an 32-bit floating-point compute operation (640pJ vs. 0.9pJ, Han et al., 2016a), with an even larger gap at smaller process nodes. Existing pruning, quantization, and reduction techniques are only applied *after* training, and do not ameliorate this energy bottleneck.

Most of the off-chip memory references during training come from accessing activations and weights. The ratio depends on the amount of weight reuse available in the DNN architecture being trained: for example, in an MLP weights are not reused within a single training sample and typically need more bandwidth than activations, while in a CNN each filter is reused for an entire layer and there can be an order of magnitude more activation accesses than weight accesses.

Several prior works have been able to reduce the footprint of activations by an order of magnitude, via techniques like gradient checkpointing (Chen et al., 2016), accelerator architectures that elide zero-valued or small activations (Albericio et al., 2016; Judd et al., 2017), and quantization during training (Jain et al., 2018). In any case, low-end devices with fewer compute resources may not even benefit from weight reuse across batches, as training on small batches and single images is often effective (Masters & Luschi, 2018).

In this paper, therefore, we focus on reducing the number of *weights* that must be stored during the training process. This is possible because deep networks have many more parameters than are needed to capture the intrinsic complexity of the tasks they are being asked to solve (Li et al., 2018). Importantly, training only a fraction of the weights to change naturally results in a pruned model *without* the need to refine the pruned weight set via additional training.

Our algorithm, Dropback, is a training-time pruning technique that greedily selects the most promising subset of

[1]Department of Electrical Engineering, The University of British Columbia, Vancouver, Canada [2]Mercedes-Benz Research & Development North America, Seattle, WA, USA. Correspondence to: Mieszko Lis <mieszko@ece.ubc.ca>.

Proceedings of the 2nd SysML Conference, Palo Alto, CA, USA, 2019.

02 Survey - Dropback

Key Technologies

Track the highest gradients

Reduce memory usage by tracked only the weights with **highest accumulated gradients.**

Reinitialize the non-tracked weights

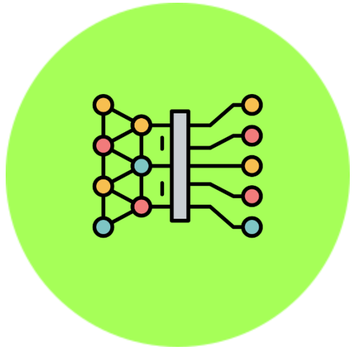
Untracked weights will not stored at memory. They will **reinitialize randomly** from scaled normal distribution.

Updated through the entire training process

Unlike original pruning, Dropback limits the set of weights that can be updated throughout the entire training process.

02 Survey - Dropback

Key Achievements



Pruning for densenetwork

Make pruning easier in **dense networks** like DenseNet, WRN, and ResNet

5x ~ 11x

Less num of weights

Able to **reduce weight counts** 5X – 11X with no accuracy loss.

02 Survey - Dropback

Algorithm 1: Dropback training. $N(0, \sigma)$ is generated from the xorshift pseudo-RNG. W_{trk} and W_{utrk} = tracked and untracked weights; T and U = tracked and untracked accumulated gradients; S = sorted accumulated gradients; k = number of gradients to track and λ = lowest tracked cumulative gradient; α = learning rate. *mask* indicates a boolean matrix with the same shape as the weights. $\overline{mask}$ indicates the logical inverse of the mask.

Initialization: $W^{(0)}$ with $W^{(0)} \sim N(0, \sigma)$
Output: $W^{(t)}$
while *not converged* **do**
  $T = \left\{ \left| \sum_{i=0}^{t-1} \frac{\alpha \partial f\left(W^{(i-1)}; x^{(i-1)}\right)}{\partial w} \right| \text{ s.t. } w \in W_{trk} \right\}$
  if not frozen:
    $U = \left\{ \left| \frac{\alpha \partial f\left(W^{(i-1)}; x^{(i-1)}\right)}{\partial w} \right| \text{ s.t. } w \in W_{utrk} \right\}$
  else:
    $U = \{\}$
  $S = \text{sort}(T \cup U)$
  $\lambda = S_k$
  $mask = \mathbb{1}(S > \lambda)$
  $W^{(t)} = mask \cdot \left(W^{(t-1)} - \alpha \nabla f\left(W^{(t-1)}; x^{(t-1)} \right) \right) + \overline{mask} \cdot W^{(0)}$
  $t = t + 1$
end

- Calculate the total accumulated gradient for each weight
- Sorted by weights with the largest total cumulative gradient.
- K th weight will be the threshold
- Only the weights greater than the threshold are updated, and the rest are converted to one of a set of random variables that follow a normal distribution with a mean of 0.

03 Aspects it link to AIoT

Recently, the size of models has increased rapidly due to the huge generative AI craze

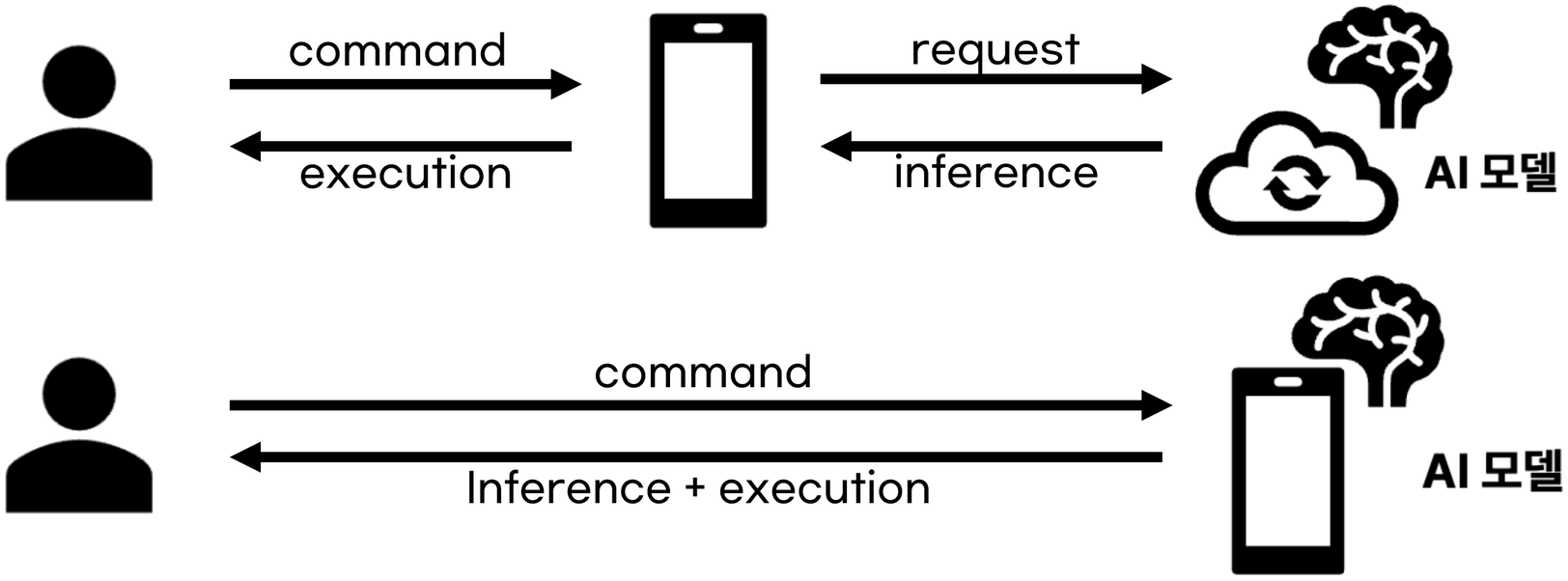
뉴스홈 | 최신기사

생성형 AI 수요 증가에…빅테크, 데이터센터 잇따라 확충

송고시간 | 2024-01-20 04:15

| 아마존, 日에 4년간 20조원 투입…구글·MS는 영국에 투자

- As a result, investments in cloud and server infrastructure are increasing. (Making data center)
  https://www.yna.co.kr/view/AKR20240120005900091



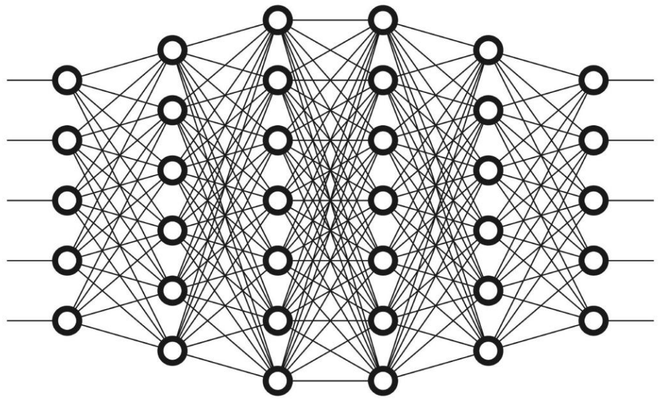
03 Aspects it link to AIoT

➢ **But only Cloud base AI is not good…**

- Excessively consumes computing resources and power
- Risk of sensitive information being leaked

Recently, many companies are tending to perform AI learning with ***existing equipment*** due to worsening investment condition

Solution ⇒ ***lightweighting deep learning models +***
efficient memory use by efficient computation

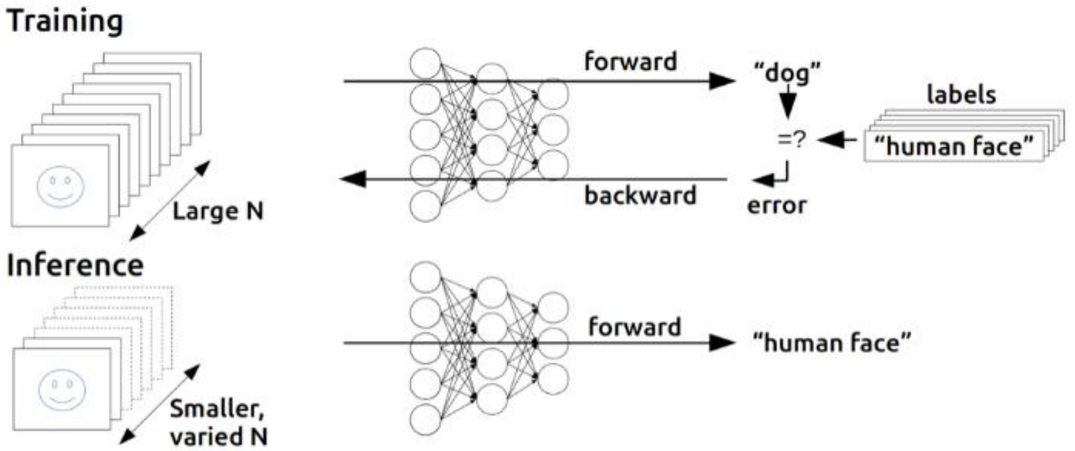


03 Aspects it link to AIOT

In On-device AI,
there is on-device inference and on device training

Comparatively little attention, however, has been paid to the problem of on-device *training*, which has so far been limited to simple models (e.g., Apple, 2017). Training takes much more energy than inference: one iteration on a single sample involves roughly thrice as many weight accesses, and typically many iterations on many samples are required. Off-chip memory accesses dominate, costing hundreds of times more energy than computation: in a 45nm process, for example, accessing a 32-bit value from DRAM costs over 700× more energy than an 32-bit floating-point compute

Full deep neural network training on a pruned weight budget



Naturally, ***more energy and time are consumed during training***.

Because while training, there is weight update by **backward**(back propagation)

In inference time, there is **only forward** (There is no weight update)

So, Memory optimization is important to on-device(IoT) Training

Directly learning from data at the point of generation enables customized predictions and recommendations.

03 Aspects it link to AIOT

AIoT Use Cases with On-Device Training

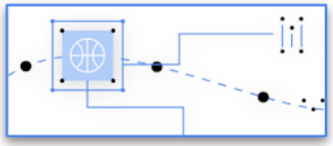
Example) Google ML Kit SDK keeps all machine learning on the device
If improve memory optimization, it will allow more complex models to be trained on device

Vision

Text recognition

Image labeling

Barcode scanning



Object detection and tracking

Face detection

Natural Language

Language Identification

Smart Reply

On-device Translation

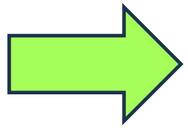
04 After proposal (plan)

1. We will do Quantitative Comparison (3 Papers and base model) and analyze in detail the reasons

Ex) Compare Base model and model with dropback applied

https://github.com/hwnam5/AIoT_Teamproject.git

```python
# 모든 가중치에 대해 상위 k개 기울기만 추적
for param in model.parameters():
    if param.grad is not None:
        # 기울기의 절댓값을 기준으로 상위 k개 기울기 추적
        grad_values = gradients[param].view(-1)
        topk_values, _ = grad_values.topk(k, largest=True)

        threshold = topk_values[-1]  # 상위 k개 기울기의 최소값 (임계값)

        # 임계값 이상이면 해당 기울기로 업데이트, 아니면 랜덤 값으로 대체
        mask = gradients[param] >= threshold
        random_update = 0 * (1 - mask.float())
        param.data -= alpha * torch.where(mask, param.grad, random_update)
```

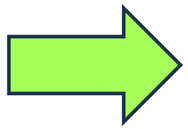


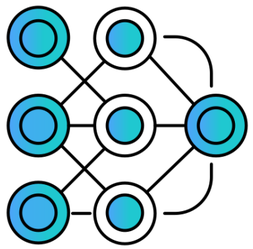
```
Epoch: 06 | Epoch Time: 1m 17s
        Train Loss: 0.626 | Train Acc: 64.48%
         Val. Loss: 0.697 |  Val. Acc: 50.03%
Epoch: 07 | Epoch Time: 1m 16s
        Train Loss: 0.639 | Train Acc: 61.68%
         Val. Loss: 0.745 |  Val. Acc: 50.03%
Epoch: 08 | Epoch Time: 1m 13s
        Train Loss: 0.576 | Train Acc: 69.55%
         Val. Loss: 0.762 |  Val. Acc: 50.03%
Epoch: 09 | Epoch Time: 1m 13s
        Train Loss: 0.575 | Train Acc: 69.12%
         Val. Loss: 0.801 |  Val. Acc: 50.03%
Epoch: 10 | Epoch Time: 1m 13s
        Train Loss: 0.555 | Train Acc: 72.67%
         Val. Loss: 0.783 |  Val. Acc: 50.03%

Vgg11 with Dropout
Total training time: 12m 2s
```

VS

```
Epoch: 06 | Epoch Time: 1m 17s
        Train Loss: 0.695 | Train Acc: 45.46%
         Val. Loss: 0.693 |  Val. Acc: 45.78%
Epoch: 07 | Epoch Time: 1m 17s
        Train Loss: 0.693 | Train Acc: 51.56%
         Val. Loss: 0.691 |  Val. Acc: 51.09%
Epoch: 08 | Epoch Time: 1m 15s
        Train Loss: 0.696 | Train Acc: 47.03%
         Val. Loss: 0.691 |  Val. Acc: 51.61%
Epoch: 09 | Epoch Time: 1m 15s
        Train Loss: 0.693 | Train Acc: 50.68%
         Val. Loss: 0.690 |  Val. Acc: 52.10%
Epoch: 10 | Epoch Time: 1m 14s
        Train Loss: 0.692 | Train Acc: 50.59%
         Val. Loss: 0.690 |  Val. Acc: 51.06%

Vgg11 with Droback
Total training time: 12m 34s
```

2. We will select best performance paper and apply the points that can be further improved in the paper.





04 After proposal (plan)

Number	Date	Detail
Week 1	**11.10 ~ 11.16**	Study more about paper(algorithm). Measure training time and accuracy of base model.
Week 2	**11.17 ~ 11.23**	Measure main idea of each papres. Upload on Github.
Week 3	**11.24 ~ 11.30**	Find areas for improvement. Search another paper or method to get idea.
Week 4	**12.01 ~ 12.07**	Implement with code. Prepare final presentation

A Survey on Memory Optimization Techniques for On-Device Learning in AIoT Systems

AIoT Team 06
202020708 김평주
202020820 남현원