TinyTL: Reduce Memory, not Parameters for Efficient On-Device Learning

Han Cai\textsuperscript{1}  
Chuang Gan\textsuperscript{2}  
Ligeng Zhu\textsuperscript{1}  
Song Han\textsuperscript{1}

\textsuperscript{1}MIT  \textsuperscript{2}MIT-IBM Watson AI Lab

NeurIPS 2020
Adapt to Newly Collected Data on the Edge

- Customization: AI systems need to continually adapt to new data collected from the sensors.
Cloud-based Learning

- Customization: AI systems need to continually adapt to new data collected from the sensors.
On-device Learning

- Customization: AI systems need to continually adapt to new data collected from the sensors.

- Security: Data cannot leave devices because of security and regularization.
Edge devices have tight memory constraints. The training memory footprint of neural networks can easily exceed the limit.
• Edge devices have tight memory constraints. The training memory footprint of neural networks can easily exceed the limit.

• Edge devices are energy-constrained. Failing to fit the training process into the energy-efficient on-chip SRAM will significantly increase the energy cost.
Activation is the Memory Bottleneck, not Parameters

- Activation is the main bottleneck for on-device learning, not parameters.
Activation is the Memory Bottleneck, not Parameters

- Activation is the main bottleneck for on-device learning, not parameters.

- Previous methods focus on reducing the number of parameters or FLOPs, while the main bottleneck does not improve much.
Related Work: Parameter-Efficient Transfer Learning

- **Full**: Fine-tune the full network. Better accuracy but highly inefficient.
- **Last**: Only fine-tune the last classifier head. Efficient but the capacity is limited.
Related Work: Parameter-Efficient Transfer Learning

- **Full**: Fine-tune the full network. Better accuracy but highly inefficient.
- **Last**: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- **BN+Last**: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency.

**Related Work: Parameter-Efficient Transfer Learning**

- **Full**: Fine-tune the full network. Better accuracy but highly inefficient.
- **Last**: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- **BN+Last**: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. **But the memory saving is limited.**
Related Work: Parameter-Efficient Transfer Learning

- **Full**: Fine-tune the full network. Better accuracy but highly inefficient.
- **Last**: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- **BN+Last**: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. **But the memory saving is limited. The accuracy loss is still significant.**
TinyTL: Memory-Efficient Transfer Learning

- **Full**: Fine-tune the full network. Better accuracy but highly inefficient.
- **Last**: Only fine-tune the last classifier head. Efficient but the capacity is limited.
- **BN+Last**: Fine-tune the BN layers and the last classifier head. Highly effective when only considering the parameter-efficiency. But the memory saving is limited. The accuracy loss is still significant.
Updating Weights is Memory-expensive While Updating Biases is Memory-efficient

\[
\begin{align*}
\text{Forward:} & \quad a_{i+1} = a_i W_i + b_i \\
\text{Backward:} & \quad \frac{\partial L}{\partial W_i} = a_i^T \frac{\partial L}{\partial a_{i+1}}, \quad \frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial a_{i+1}} = \frac{\partial L}{\partial a_{i+2}} W_{i+1}^T
\end{align*}
\]

- Updating weights requires storing intermediate activations
- Updating biases does not
TinyTL: Fine-tune Bias Only

Freeze weights, only fine-tune biases
=> save 12x memory
TinyTL: Fine-tune Bias Only

Freeze weights, only fine-tune biases
=> save 12x memory, but also hurt the accuracy
TinyTL: Lite Residual Learning

- Add lite residual modules (only 4% memory overhead) to increase model capacity
  (1/6 channel, 1/2 resolution, 2/3 depth => ~4%)
TinyTL: Lite Residual Learning
• Using the same pre-trained model (ProxylessNAS-Mobile), TinyTL provides 4.6x memory saving without accuracy loss.

TinyTL: Same Accuracy, Up to 6.5x Memory Saving

Using the same pre-trained model (ProxylessNAS-Mobile), TinyTL provides up to 6.5x memory saving without accuracy loss.

TinyTL: Fit the Training Process Into Cache

- TinyTL supports training with batch size 1.
- It further reduces the training memory cost to 16MB (typical L3 cache size), making it easier to train on the cache, which is much more energy-efficient than training on DRAM.
TinyTL: Reduce Memory, not Parameters for Efficient On-Device Learning

User \rightarrow \text{New and Sensitive Data} \rightarrow \text{Intelligent Edge Devices}

Typical L3 Cache Size: 16MB

Project Page: http://tinyml.mit.edu