
# Lecture 07:
# Efficient DNN Training, Parameter efficient Finetuning, Federated Learning

# Some Notes: Project Proposal

- Due on Mar 19th (1 page)
- You should begin forming teams of 2-3 students and start brainstorming project ideas now.
- You should propose something feasible to do.
- Please discuss with me during office hours.

# Some Notes: Midterm

- April 2nd.
- Will cover materials up to lecture 8.
- Will release the detailed coverage.
- In-class exam.

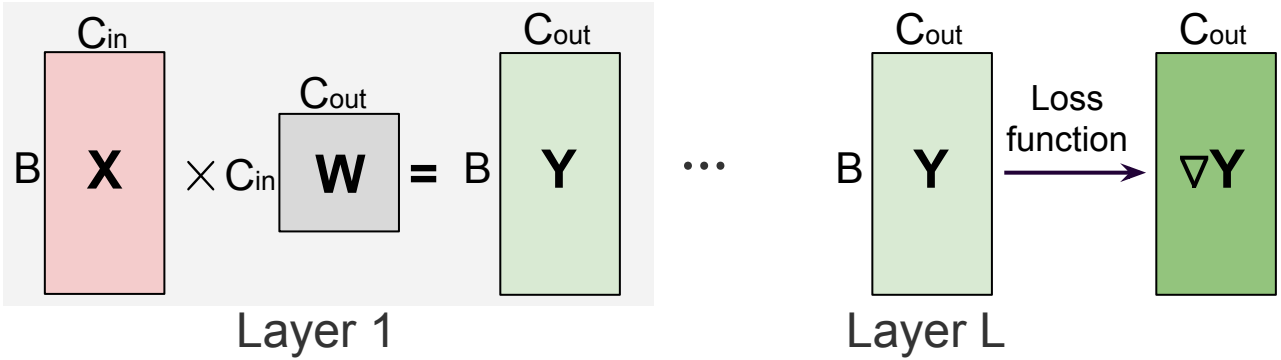
# Recap

- Distillation
- Low-rank factorization
- Neural Network Search (NAS)

# Topics

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Federated Learning

# Forward Pass for Linear Layer



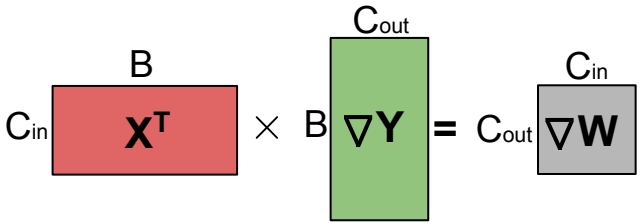

B: batch size $C_{in}$: input channels $C_{out}$: output channels

**X**: input maps **W**: weight filters **Y**: output maps

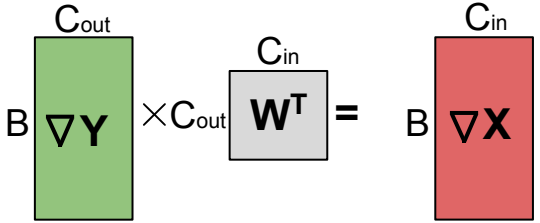
- The fully-connected layer during the forward propagation can be converted into matrix multiplications

# Backward Pass for Linear Layer

**Weight Gradient Computation**




**Data Gradient Computation**




**X**: input maps **W**: weight filters **Y**: output maps

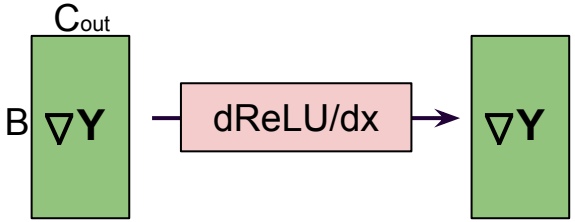
**∇X**: input gradient **∇W**: weight gradient **∇Y**: output gradient

- DNN backward propagation involves two matrix multiplications

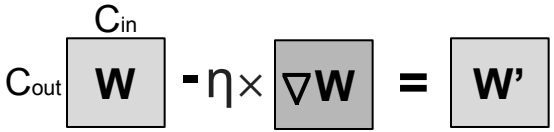
NYU SAI LAB

# Backward Pass for Linear Layer

**Data Gradient Computations**




**Weight Gradient Updates**

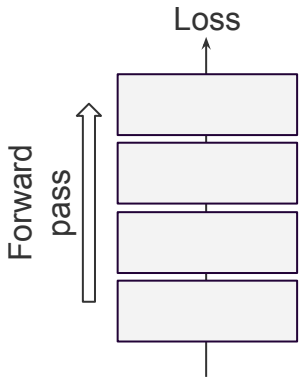



**X**: input maps **W**: weight filters **Y**: output maps

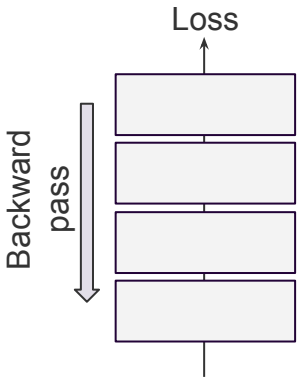
**∇X**: input gradient **∇W**: weight gradient **∇Y**: output gradient

- DNN backward propagation involves two matrix multiplications

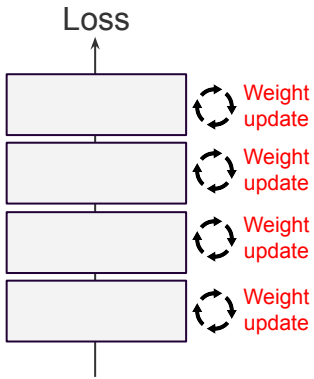
# Training Process




```
def forward(self, x):
    ...
return
```




```
loss.backward()
```



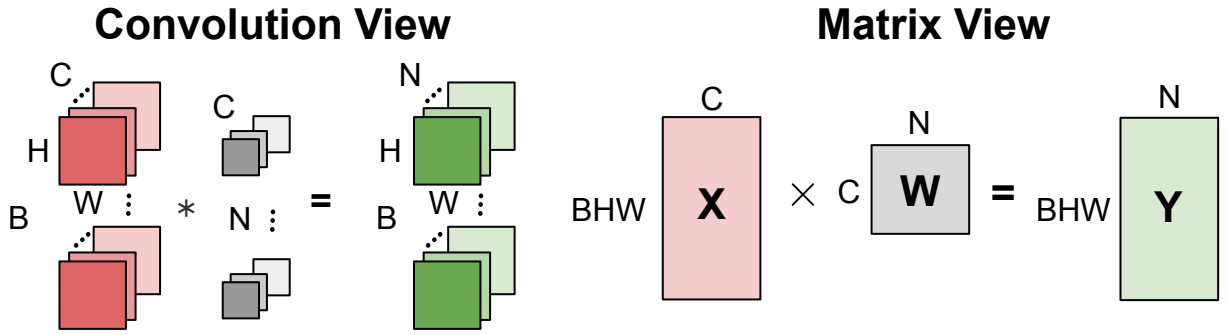

```
optimizer.step()
```

NYU SAI LAB

# Forward Pass for Convolutional Layer

Forward Pass

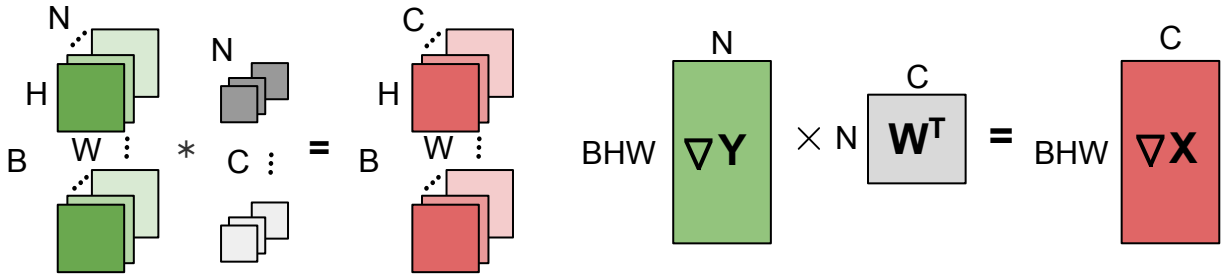
Compute output **Y**




- Assume a weight kernel size of 1✖1.

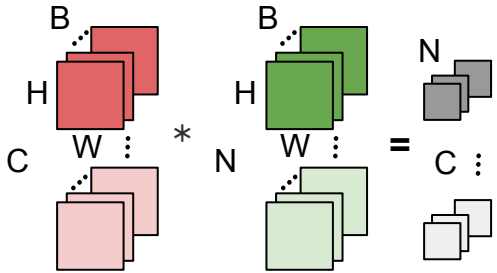
# Backward Pass for Convolutional Layer

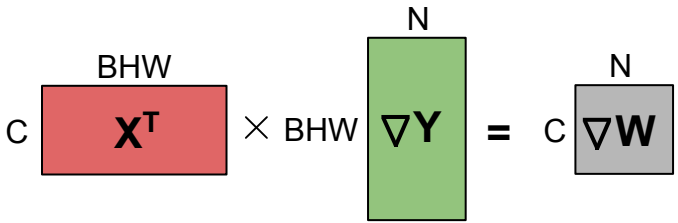
Backward Pass
Compute Activation gradients $\nabla X$



# Backward Pass for Convolutional Layer

Backward Pass
Compute weight gradients $\nabla \mathbf{W}$

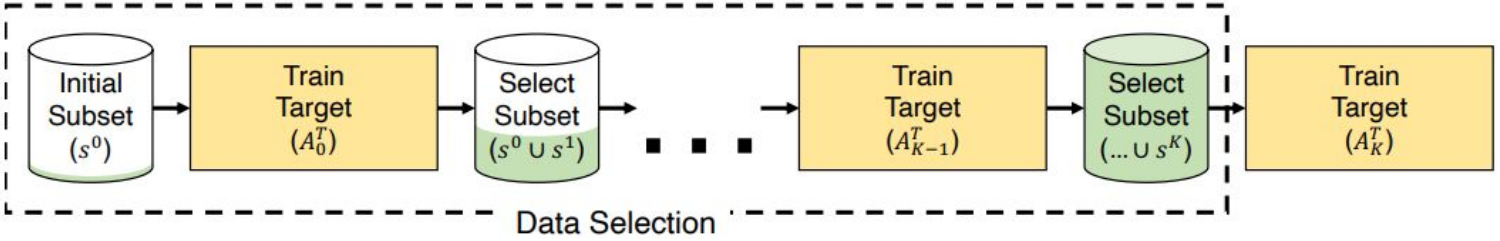


# Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
  - Training data sampling
  - Parameter sampling
  - Pruning during training
  - Quantization during training

# Training Data Sampling for Efficiency

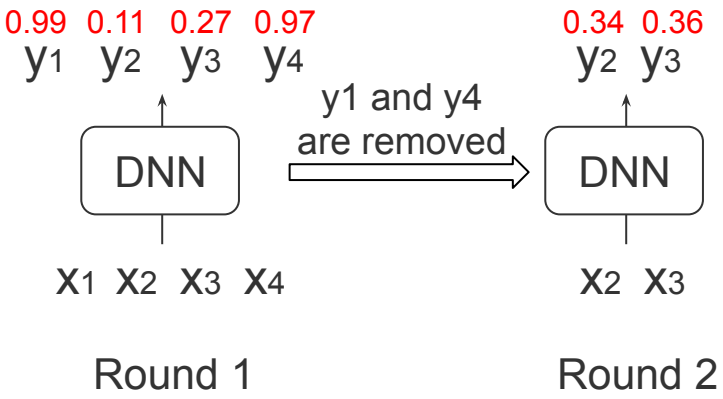



- Assume a total b samples are targeted to be picked. We consider a batch setting with K rounds where we select b/K points in every round.
- Training the target model with b/K samples, then evaluate the rest of the sample over the model. Find the batch with the least confidence score. Append it to the training dataset.

NYU SAI LAB

Cody Coleman, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Select via proxy: Efficient data selection for training deep networks, 2019.

# Training Data Sampling for Efficiency




- Assume a total b samples are targeted to be picked. We consider a batch setting with K rounds where we select b/K points in every round.
- Training the target model with b/K samples, then evaluate the rest of the sample over the model. Find the batch with the least confidence score. Append it to the training dataset.

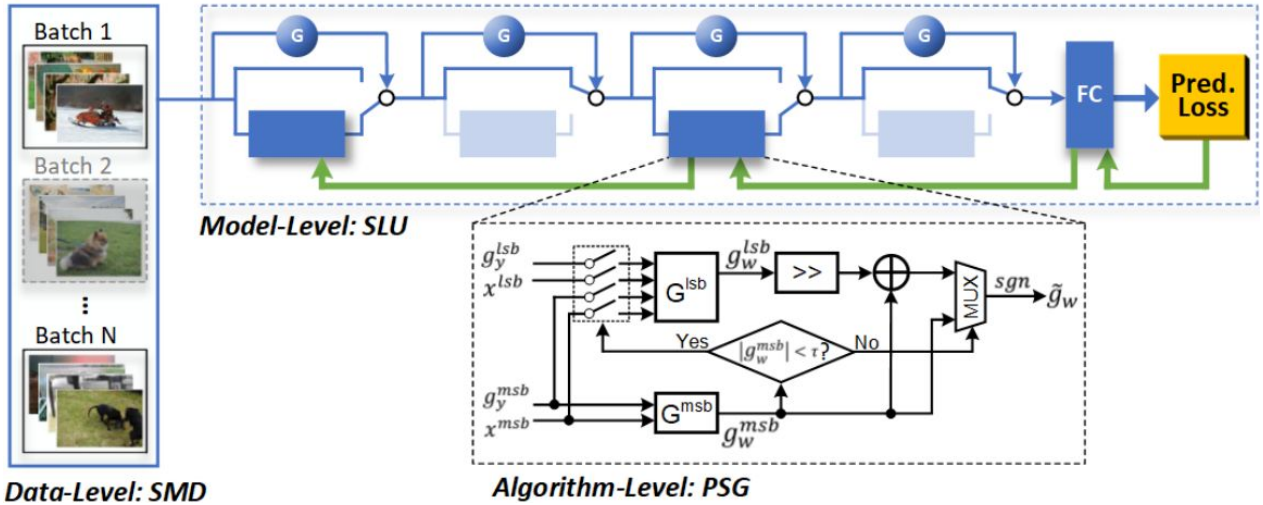
NYU SAI LAB

Cody Coleman, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Select via proxy: Efficient data selection for training deep networks, 2019.

# Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
  - Training data sampling
  - Parameter sampling
  - Pruning during training
  - Quantization during training

# E2-Train

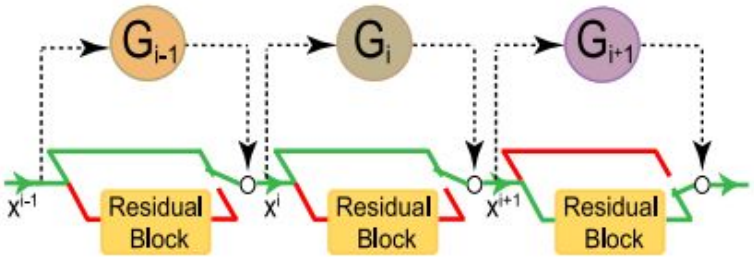



- A stochastic mini-batch dropping strategy is proposed.
- Stochastic minibatch dropping simply skips every mini-batch with a default probability of 0.5.
- For some easy dataset, this will generate negligible impact on performance.



Wang, Yue, et al. "E2-train: Training state-of-the-art cnns with over 80% energy savings." *Advances in Neural Information Processing Systems* 32 (2019).

# Dynamically Layer Skipping




- $G_i(x_i) \in \{0, 1\}$ is the gating function for layer i.
- It determines whether to skip to current residual block or not.
- During the training, G and residual blocks are trained together.
- Loss = acc_loss + computation_loss
- We will skip different layers adaptively based on inputs.

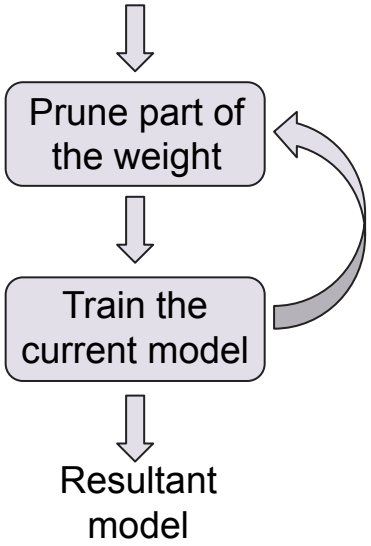

Wang, Xin, et al. "Skipnet: Learning dynamic routing in convolutional networks." *Proceedings of the European conference on computer vision (ECCV)*. 2018.

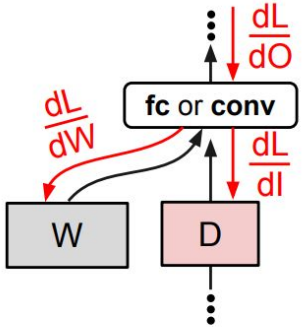
Wang, Yue, et al. "E2-train: Training state-of-the-art cnns with over 80% energy savings." *Advances in Neural Information Processing Systems* 32 (2019).

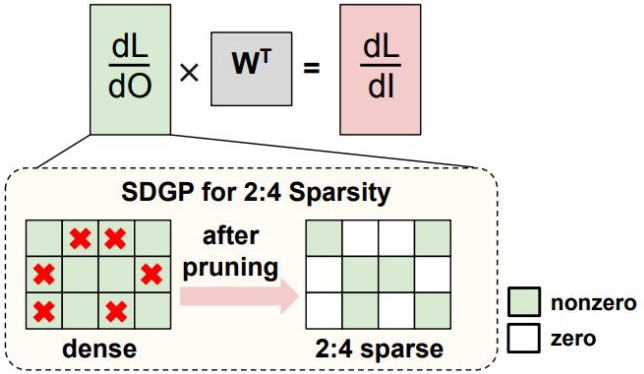
# Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
  - Training data sampling
  - Parameter sampling
  - Pruning during training
  - Quantization during training

# Pruning during Training




- We can remove the unnecessary weight during the DNN training process.



McDanel, Bradley, Helia Dinh, and John Magallanes. "Accelerating dnn training with structured data gradient pruning." *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE, 2022.

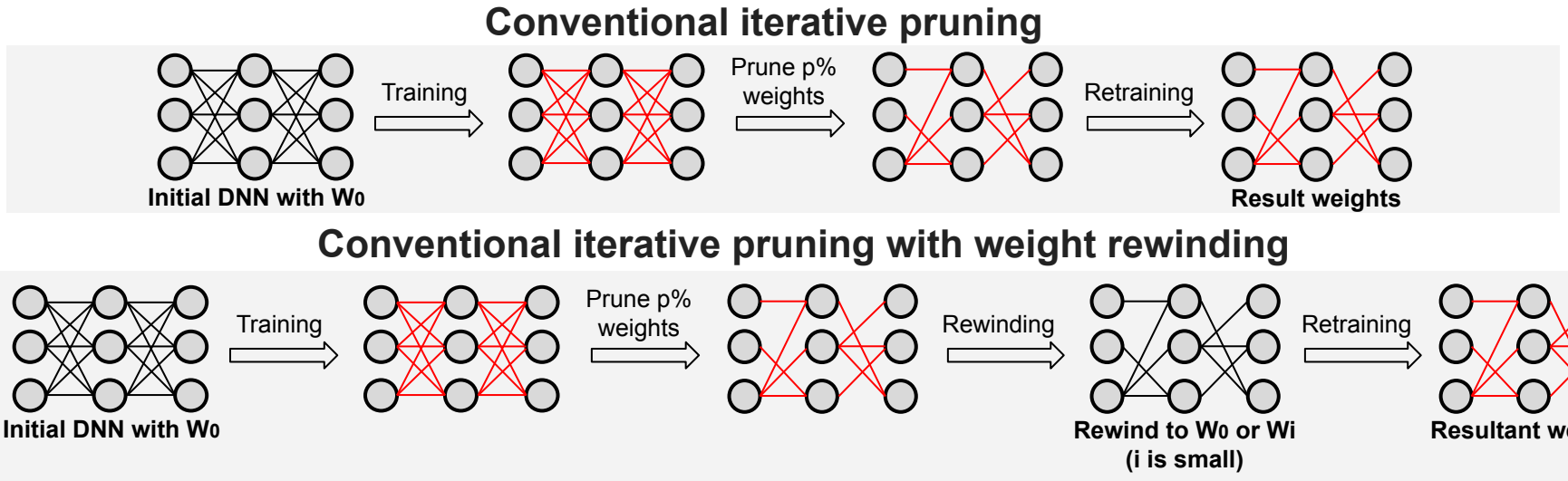
# How to Find the Winning Tickets?

- **Iterative Magnitude Pruning (IMP):**
  - Initialized DNN with random weights $w_0$.
  - While the sparsity level has not reached:
    - Train the DNN with k epochs until convergence
    - prune p% of the nonzero weights.
    - Reinitialize the remaining weights using the values in $w_0$, finetune the remaining weights for k epochs (**Rewind**).
  - Return the weights.
- Later work has shown that rewind to $w_i$ (i is small) works better for larger networks.



Frankle, Jonathan, et al. "Stabilizing the lottery ticket hypothesis." *arXiv preprint arXiv:1903.01611* (2019).

# Weight Rewinding

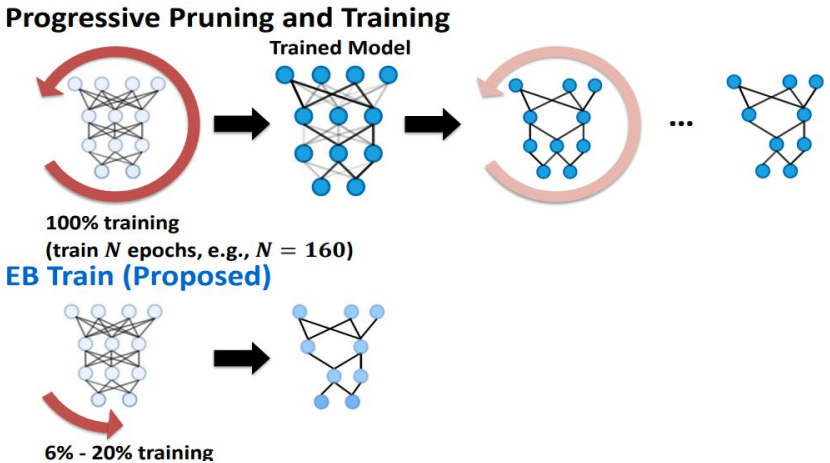



- The pruned architecture itself, rather than a set of inherited “important” weights, is more crucial to the accuracy in the final model, which suggests that in some cases pruning can be useful as an architecture search paradigm.

NYU SAI LAB

Liu, Zhuang, et al. "Rethinking the value of network pruning." *arXiv preprint arXiv:1810.05270* (2018).

# Early-bird Ticket




- LTH shows that there exist winning tickets (small but critical subnetworks) for dense, randomly initialized networks, that can be trained alone to achieve a comparable accuracy to the latter in a similar number of iterations.
- The winning tickets can be drawn very early in training and with aggressively low-cost training algorithms.
- Early-bird tickets can be founded via low-cost training schemes (e.g., early stopping and low-precision training) at large learning rates

NYU SAI LAB

You, Haoran, et al. "Drawing early-bird tickets: Towards more efficient training of deep networks." *arXiv preprint arXiv:1909.11957* (2019).

# Early-bird Ticket

**Algorithm 1:** The Algorithm for Searching EB Tickets

Initialize the weights $\boldsymbol{W}$, scaling factor $r$, pruning ratio $p$, and the FIFO queue $Q$ with length $l$;
**while** $t$ (epoch) $< t_{max}$ **do**
  Update $\boldsymbol{W}$ and $r$ using SGD training;
  Perform structured pruning based on $r_t$ towards the target ratio $p$;
  Calculate the **mask distance** between the current and last subnetworks and add to $Q$.
  $t = t + 1$
  **if Max**$(Q) < \epsilon$ **then**
    $t^* = t$
    **Return** $f(x; m_{t^*} \odot \boldsymbol{W})$ (EB ticket);
  **end if**
**end while**

The mask pattern is stable

- To search for the lottery ticket, we can early stop the DNN training.



You, Haoran, et al. "Drawing early-bird tickets: Towards more efficient training of deep networks." *arXiv preprint arXiv:1909.11957* (2019).

# Efficient Computing during Training

- To reduce the training cost of DNN, we can proceed from the following dimensions:
  - Training data sampling
  - Parameter sampling
  - Pruning during training
  - Quantization during training

# DoReFaNet

Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$.
**for** $k = L$ **to** 1 **do**
  Back-propagate $g_{a_k}$ through activation function $h$
  $g^b_{a_k} \leftarrow f^G_\gamma(g_{a_k})$
  $g_{a_{k-1}} \leftarrow \texttt{backward\_input}(g^b_{a_k}, W^b_k)$
  $g_{W^b_k} \leftarrow \texttt{backward\_weight}(g^b_{a_k}, a^b_{k-1})$
  Back-propagate gradients through pooling layer if there is one
**end for**
{2. Accumulating the parameters gradients:}
**for** $k = 1$ **to** $L$ **do**
  $g_{W_k} = g_{W^b_k} \frac{\partial W^b_k}{\partial W_k}$
  $W^{t+1}_k \leftarrow Update(W_k, g_{W_k}, \eta)$
**end for**

- Linear quantize the weights and activations
- Apply stochastic quantization for the gradients.



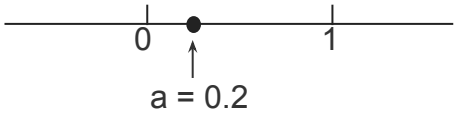
Zhou, Shuchang, et al. "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients." *arXiv preprint arXiv:1606.06160* (2016).

# DoReFaNet

- Usually gradients requires far more bitwidth than weight and activation.
- Usually gradient requires **stochastic quantization**.

| W | A | G | Training Complexity | Inference Complexity | Storage Relative Size | AlexNet Accuracy |
|---|---|---|---|---|---|---|
| 1 | 1 | 6 | 7 | 1 | 1 | 0.395 |
| 1 | 1 | 8 | 9 | 1 | 1 | 0.395 |
| 1 | 1 | 32 | - | 1 | 1 | 0.279 (BNN) |
| 1 | 1 | 32 | - | 1 | 1 | 0.442 (XNOR-Net) |
| 1 | 1 | 32 | - | 1 | 1 | 0.401 |
| 1 | 1 | 32 | - | 1 | 1 | 0.436 (initialized) |
| 1 | 2 | 6 | 8 | 2 | 1 | 0.461 |
| 1 | 2 | 8 | 10 | 2 | 1 | 0.463 |
| 1 | 2 | 32 | - | 2 | 1 | 0.477 |
| 1 | 2 | 32 | - | 2 | 1 | 0.498 (initialized) |
| 1 | 3 | 6 | 9 | 3 | 1 | 0.471 |
| 1 | 3 | 32 | - | 3 | 1 | 0.484 |
| 1 | 4 | 6 | - | 4 | 1 | 0.482 |
| 1 | 4 | 32 | - | 4 | 1 | 0.503 |
| 1 | 4 | 32 | - | 4 | 1 | 0.530 (initialized) |
| 8 | 8 | 8 | - | - | 8 | 0.530 |
| 32 | 32 | 32 | - | - | 32 | 0.559 |

# Deterministic and Stochastic Quantization



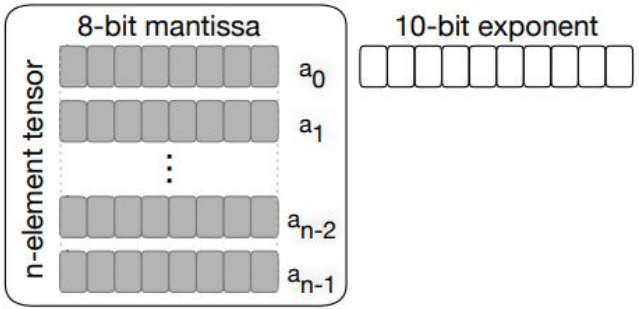

- To quantize a, conventional linear quantization will make q(a) = 0. However, this will cause a bias.
- With stochastic quantization:

$$q(a) = \begin{cases} 1 & \text{for } p = 0.2 \\ 0 & \text{for } p = 0.8 \end{cases}$$

- Stochastic quantization is extremely useful when applying quantization to accelerate DNN training.

# Training DNNs with Hybrid BFP


(a) FP repr. with an exponent per tensor element.




(b) BFP repr. with an exponent per tensor.

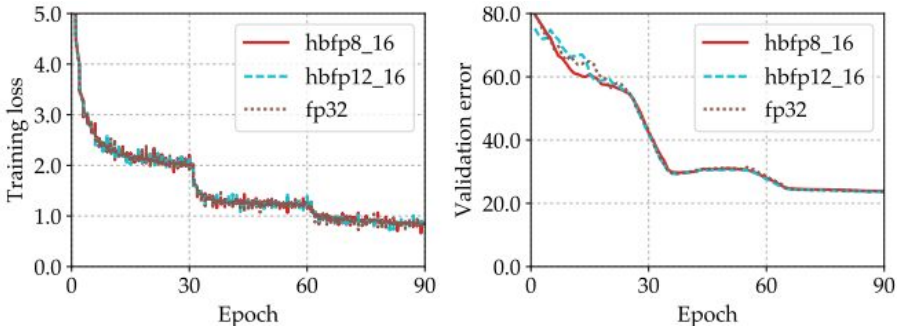
- Block floating point format achieves a better hardware efficiency and comparable representation capability than FP.

NYU SAI LAB

Drumond, Mario, et al. "Training dnns with hybrid block floating point." *Advances in Neural Information Processing Systems* 31 (2018).

# Training DNNs with Hybrid BFP

- Use BFP in all dot-product-based operations present in DNNs (i.e., convolutions, matrix multiplications, and outer products), and floating-point representations for all other operations (i.e., activations, regularizations, etc).
- To minimize data loss in long-lasting training state, the weights are stored with wider mantissas.



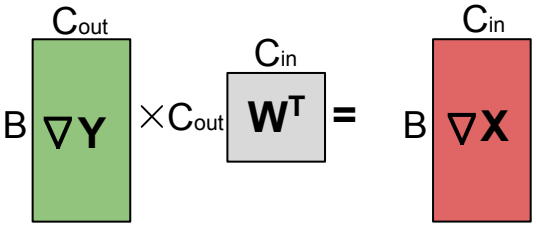

- ResNet-50 trained on ImageNet for 90 epochs.
- 8 bit mantissa, 16 bits weight seems to achieve comparable performance as FP32. A mantissa bitwidth of 12 achieves an even better performance.
- A tile size of 24

NYU SAI LAB

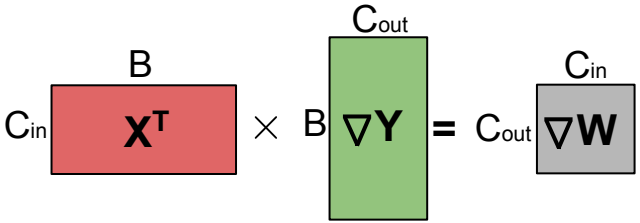
Drumond, Mario, et al. "Training dnns with hybrid block floating point." *Advances in Neural Information Processing Systems* 31 (2018).

# Two Copies of Weights

**Input data gradient Computation**




**Weight Gradient Computation**

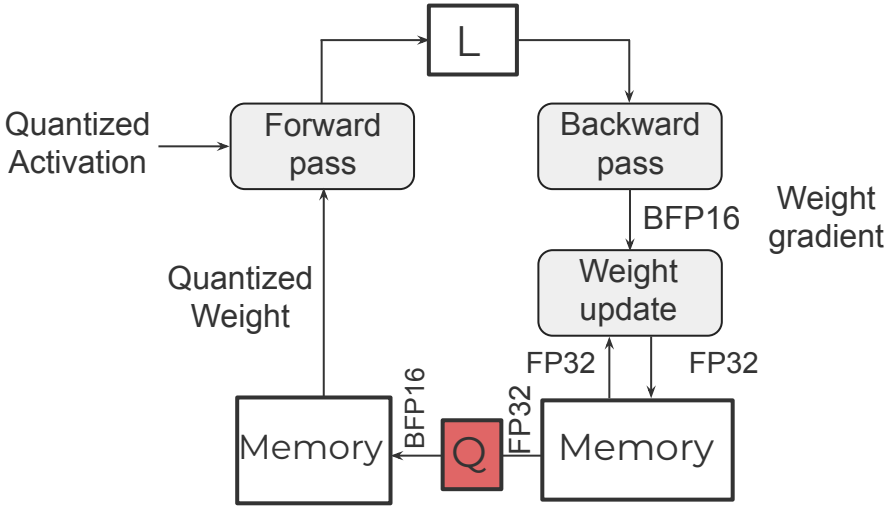



**Weight Gradient Updates**


- Gradient and forward propagation are performed using BFP.
- Weights are updated using FP.
- Two copies of weights are used.

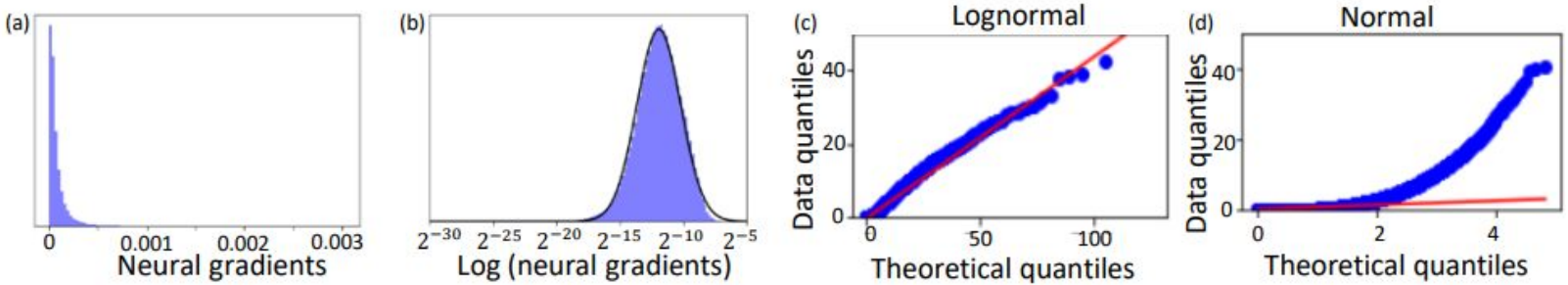
NYU SAI LAB

# Two Copies of Weights




- Two pieces of copies are needed to be kept in the memory.
- The weight updates are usually performed with higher precision (e.g., FP16).

# Neural Gradients are Near-Lognormal: Improved Quantized and Sparse Training




- The distribution of neural gradients is approximately lognormal.
- We can use lognormal regression to determine the optimal quantization setting (e.g., bitwidth, quantization interval).

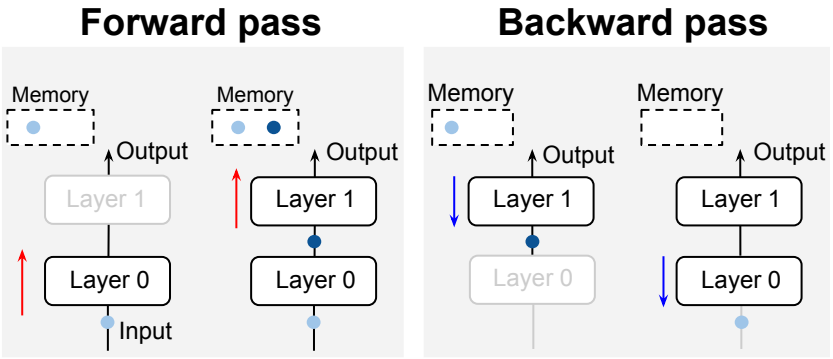
NYU SAI LAB

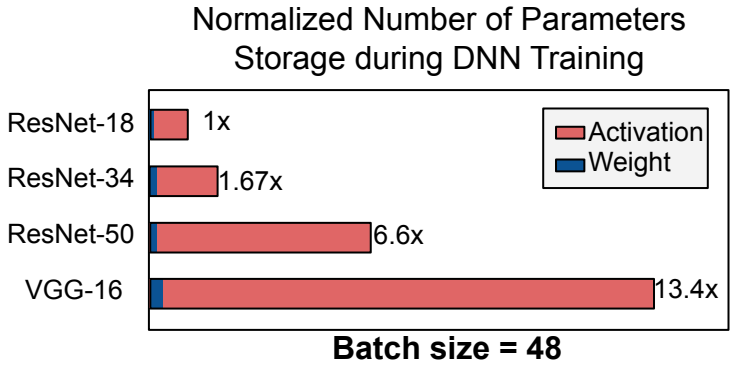
Chmiel, Brian, et al. "Neural gradients are near-lognormal: improved quantized and sparse training." *arXiv preprint arXiv:2006.08173* (2020).

# Topics

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Federated Learning

# Memory Consumption During Training



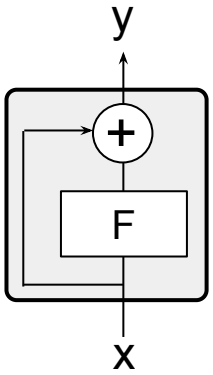




- The memory footprint grows proportional with the layer depth. The activation in the early DNN layers need to be stored for a long time.
- Activations consume most of the memory space, approximately 13 times larger than the weights on average.

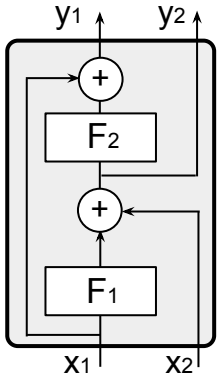
NYU SAI LAB

# Memory Efficiency During Training

**Residual Architecture**




**Reversible Architecture**




**Forward pass:**

$y_2 = F_1(x_1) + x_2$

$y_1 = F_2(y_2) + x_1$

**Backward Pass:**

$x_1 = y_1 - F_2(y_2)$

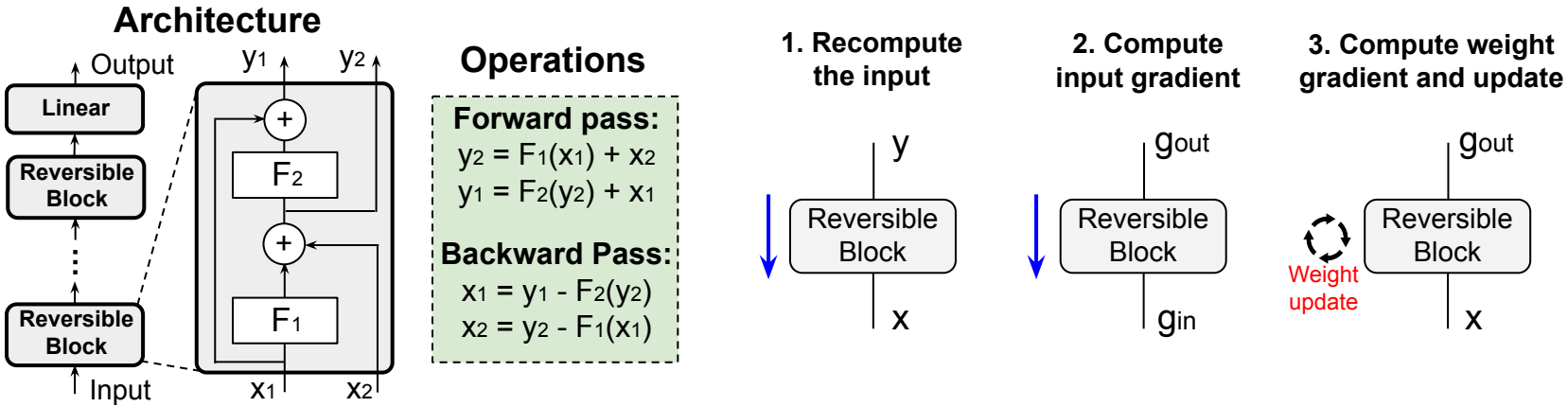
$x_2 = y_2 - F_1(x_1)$

- A reversible residual network (RevNet) is a variant of the canonical residual neural network (ResNet).



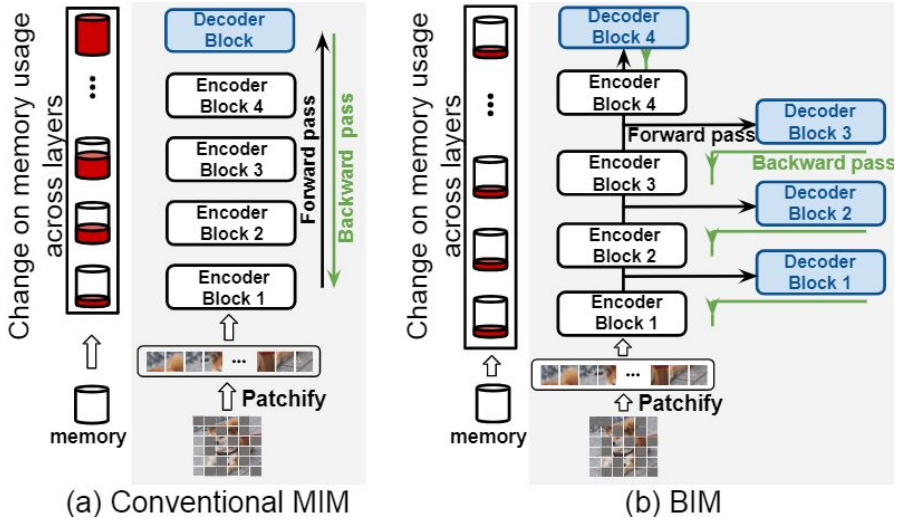
Gomez, Aidan N., et al. "The reversible residual network: Backpropagation without storing activations." *Advances in neural information processing systems* 30 (2017).

# Memory Efficiency During Training




- The backward pass computations can be performed without storing the input activations.
- Given the output y, the input activations are first recomputed. Afterwards, the input and weight gradients are computed with standard backward pass operations.

# BIM: Block-Wise Local Learning with Masked Image Modeling

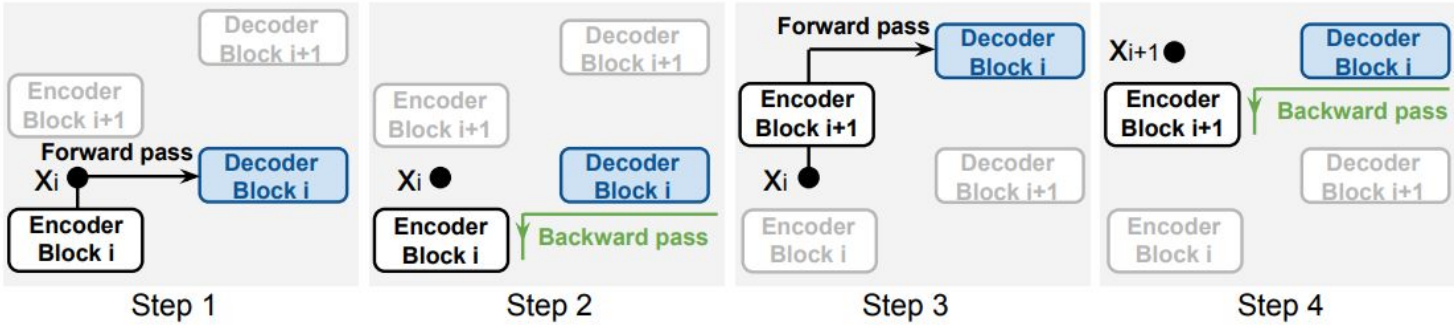



- Local exist is introduced during the training process.
- The intermediate results can be discarded once the training process for the current layer is complete.



Luo, Yixuan, Mengye Ren, and Sai Qian Zhang. "BIM: Block-Wise Self-Supervised Learning with Masked Image Modeling." *arXiv preprint arXiv:2311.17218* (2023).

# BIM: Block-Wise Local Learning with Masked Image Modeling




- Once the parameter updates in encoder block i and decoder block i are finished, all intermediate features stored in the buffer, except for xi , can be cleared from memory, preserving them for future use.

NYU SAI LAB

Luo, Yixuan, Mengye Ren, and Sai Qian Zhang. "BIM: Block-Wise Self-Supervised Learning with Masked Image Modeling." *arXiv preprint arXiv:2311.17218* (2023).

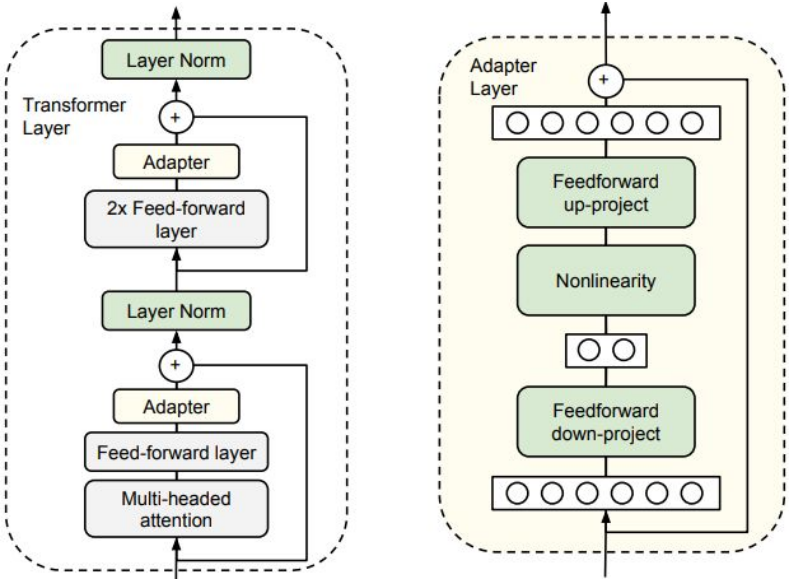
# Topics

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Federated Learning

# Parameter-efficient Finetuning (PEFT)

- Large models (LMs), often consisting of billions of parameters, require vast amounts of computational resources for execution.
- The expansive scale and computational demands pose considerable challenges when customizing them for particular downstream tasks.
- To better adapt the LMs over the downstream tasks, we can finetune a small portion of the LM parameters. This will make LMs achieve great performance over the downstream tasks while minimizing the training cost.
- Some of the popular PEFT Algorithms:
  - LoRA
  - Adapter
  - BitFit

# Parameter-Efficient Transfer Learning for NLP




- We add the adapter module twice to each Transformer layer.
- The adapter consists of a bottleneck which contains few parameters relative to the attention and feedforward layers in the original model. The adapter also contains a skip-connection.
- The learnable parameters contributes to around 0.5 − 8% of the parameters of the original model.



Houlsby, Neil, et al. "Parameter-efficient transfer learning for NLP." *International conference on machine learning*. PMLR, 2019.

# BitFit

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

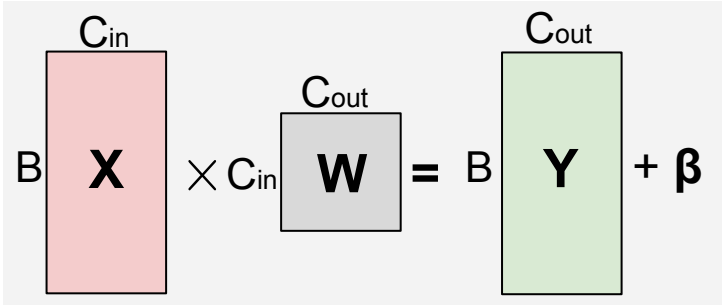
$$\mathbf{h}_2^\ell = \text{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$
$$\mathbf{h}_4^\ell = \text{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right)$$
$$\mathbf{h}_5^\ell = \text{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right)$$
$$\text{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

- BitFiT is a sparse-finetuning method where only the bias-terms of the model are being modified.
- Applying BitFit on pre-trained BERT models is competitive with (and sometimes better than) fine-tuning the entire model.
- Bias parameters make up 0.09% of the total number of parameters in BER.



Zaken, Elad Ben, Shauli Ravfogel, and Yoav Goldberg. "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models." *arXiv preprint arXiv:2106.10199* (2021).

# Finetune Bias is Cheap



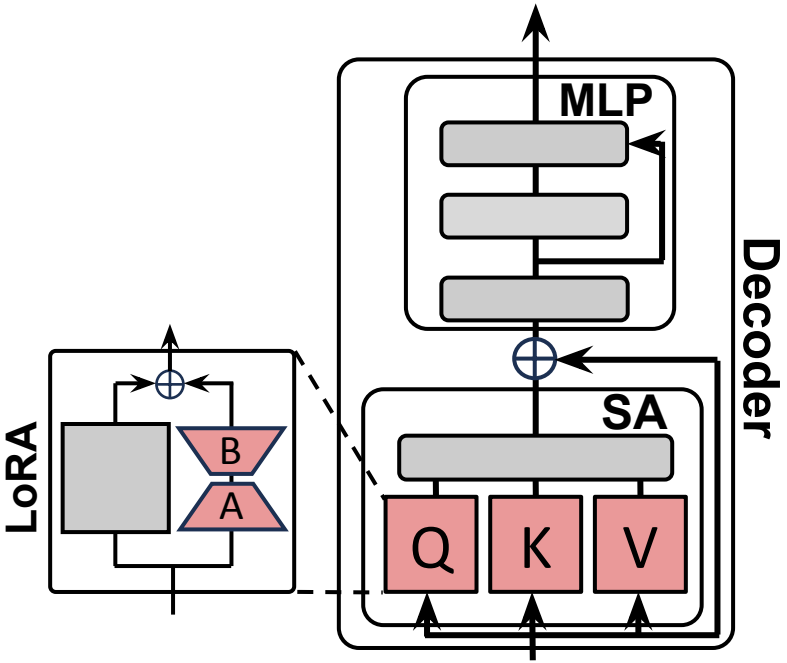

$$\frac{dL}{d\beta} = \sum_{y \in Y} \frac{dL}{dy}$$

- Updating the bias does not require buffering any intermediate results during the forward pass of DNN training.



Cai, Han, et al. "Tinytl: Reduce activations, not trainable parameters for efficient on-device learning." *arXiv preprint arXiv:2007.11622* (2020).

# Low-rank Adaptation (LoRA)




$$h = W_0 x + \Delta W x = W_0 x + BAx$$

- Only the weights within the red blocks are updated.
- Assume the weight matrix has a dimension of k×k, A and B have a size of k×r and r×k, where r << k (e.g., r=4).
- BA can be merged with the original weight $W_0$, leading to no additional computational and storage cost.



Hu, Edward J., et al. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).

# Low-rank Adaptation (LoRA)

| Model & Method | # Trainable Parameters | MNLI | SST-2 | MRPC | CoLA | QNLI | QQP | RTE | STS-B | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| $\text{RoB}_{\text{base}}$ (FT)* | 125.0M | **87.6** | 94.8 | 90.2 | **63.6** | 92.8 | **91.9** | 78.7 | 91.2 | 86.4 |
| $\text{RoB}_{\text{base}}$ (BitFit)* | 0.1M | 84.7 | 93.7 | **92.7** | 62.0 | 91.8 | 84.0 | 81.5 | 90.8 | 85.2 |
| $\text{RoB}_{\text{base}}$ ($\text{Adpt}^{\text{D}}$)* | 0.3M | $87.1_{\pm.0}$ | $94.2_{\pm.1}$ | $88.5_{\pm1.1}$ | $60.8_{\pm.4}$ | $93.1_{\pm.1}$ | $90.2_{\pm.0}$ | $71.5_{\pm2.7}$ | $89.7_{\pm.3}$ | 84.4 |
| $\text{RoB}_{\text{base}}$ ($\text{Adpt}^{\text{D}}$)* | 0.9M | $87.3_{\pm.1}$ | $94.7_{\pm.3}$ | $88.4_{\pm.1}$ | $62.6_{\pm.9}$ | $93.0_{\pm.2}$ | $90.6_{\pm.0}$ | $75.9_{\pm2.2}$ | $90.3_{\pm.1}$ | 85.4 |
| $\text{RoB}_{\text{base}}$ (LoRA) | 0.3M | $87.5_{\pm.3}$ | $\mathbf{95.1}_{\pm.2}$ | $89.7_{\pm.7}$ | $63.4_{\pm1.2}$ | $\mathbf{93.3}_{\pm.3}$ | $90.8_{\pm.1}$ | $\mathbf{86.6}_{\pm.7}$ | $\mathbf{91.5}_{\pm.2}$ | **87.2** |
| $\text{RoB}_{\text{large}}$ (FT)* | 355.0M | 90.2 | **96.4** | **90.9** | 68.0 | 94.7 | **92.2** | 86.6 | 92.4 | 88.9 |
| $\text{RoB}_{\text{large}}$ (LoRA) | 0.8M | $\mathbf{90.6}_{\pm.2}$ | $96.2_{\pm.5}$ | $\mathbf{90.9}_{\pm1.2}$ | $\mathbf{68.2}_{\pm1.9}$ | $\mathbf{94.9}_{\pm.3}$ | $91.6_{\pm.1}$ | $\mathbf{87.4}_{\pm2.5}$ | $\mathbf{92.6}_{\pm.2}$ | **89.0** |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{P}}$)† | 3.0M | $90.2_{\pm.3}$ | $96.1_{\pm.3}$ | $90.2_{\pm.7}$ | $\mathbf{68.3}_{\pm1.0}$ | $\mathbf{94.8}_{\pm.2}$ | $\mathbf{91.9}_{\pm.1}$ | $83.8_{\pm2.9}$ | $92.1_{\pm.7}$ | 88.4 |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{P}}$)† | 0.8M | $\mathbf{90.5}_{\pm.3}$ | $\mathbf{96.6}_{\pm.2}$ | $89.7_{\pm1.2}$ | $67.8_{\pm2.5}$ | $\mathbf{94.8}_{\pm.3}$ | $91.7_{\pm.2}$ | $80.1_{\pm2.9}$ | $91.9_{\pm.4}$ | 87.9 |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{H}}$)† | 6.0M | $89.9_{\pm.5}$ | $96.2_{\pm.3}$ | $88.7_{\pm2.9}$ | $66.5_{\pm4.4}$ | $94.7_{\pm.2}$ | $92.1_{\pm.1}$ | $83.4_{\pm1.1}$ | $91.0_{\pm1.7}$ | 87.8 |
| $\text{RoB}_{\text{large}}$ ($\text{Adpt}^{\text{H}}$)† | 0.8M | $90.3_{\pm.3}$ | $96.3_{\pm.5}$ | $87.7_{\pm1.7}$ | $66.3_{\pm2.0}$ | $94.7_{\pm.2}$ | $91.5_{\pm.1}$ | $72.9_{\pm2.9}$ | $91.5_{\pm.5}$ | 86.4 |
| $\text{RoB}_{\text{large}}$ (LoRA)† | 0.8M | $\mathbf{90.6}_{\pm.2}$ | $96.2_{\pm.5}$ | $\mathbf{90.2}_{\pm1.0}$ | $68.2_{\pm1.9}$ | $\mathbf{94.8}_{\pm.3}$ | $91.6_{\pm.2}$ | $\mathbf{85.2}_{\pm1.1}$ | $\mathbf{92.3}_{\pm.5}$ | **88.6** |
| $\text{DeB}_{\text{XXL}}$ (FT)* | 1500.0M | 91.8 | **97.2** | 92.0 | 72.0 | **96.0** | 92.7 | 93.9 | 92.9 | 91.1 |
| $\text{DeB}_{\text{XXL}}$ (LoRA) | 4.7M | $\mathbf{91.9}_{\pm.2}$ | $96.9_{\pm.2}$ | $\mathbf{92.6}_{\pm.6}$ | $\mathbf{72.4}_{\pm1.1}$ | $\mathbf{96.0}_{\pm.1}$ | $\mathbf{92.9}_{\pm.1}$ | $\mathbf{94.9}_{\pm.4}$ | $\mathbf{93.0}_{\pm.2}$ | **91.3** |

NYU SAI LAB

- LoRA achieves better results than Adapter and BitFit.

# Topics

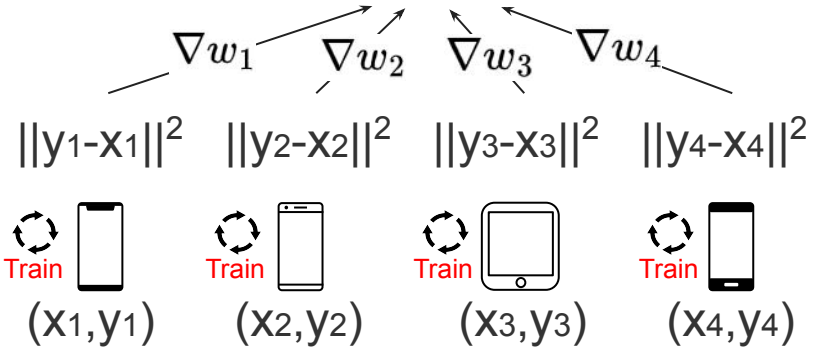
- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Federated Learning

# Federated Learning

- Training data: $(x_1,y_1)$, $(x_2,y_2)$, $(x_3,y_3)$, $(x_4,y_4)$

$$\sum_{i=1}^{4} ||y_i - F(x_i)||^2$$

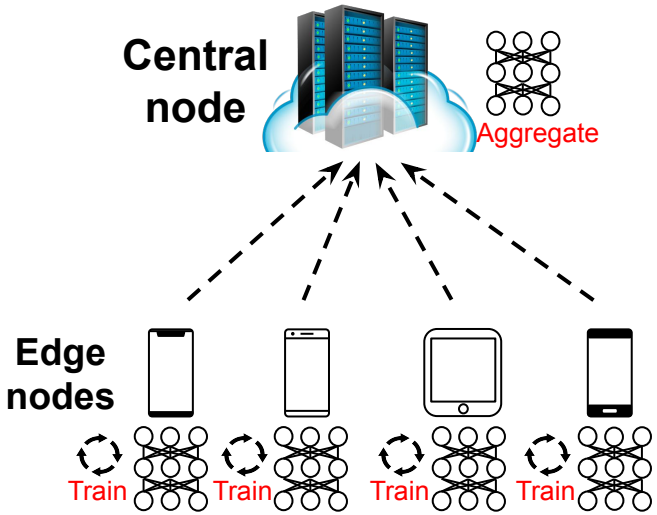
$$\nabla w = \frac{\nabla w_1 + \nabla w_2 + \nabla w_3 + \nabla w_4}{4}$$




- Non-iid training data distribution
- Heterogeneity among the edge devices
- Communication error

NYU SAI LAB

# Federated Learning

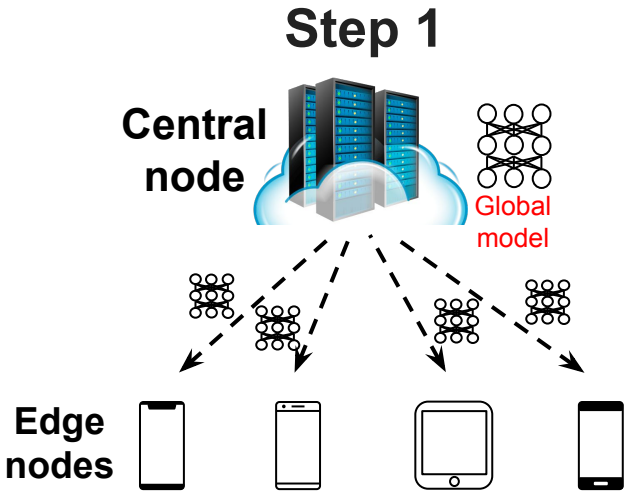



- Federated learning is a machine learning technique that allows the training of models across multiple decentralized nodes holding local data samples, **without exchanging their data**.
- This approach enhances privacy, user can train the powerful DNN model **without sharing the dataset**.



McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

# FedAvg



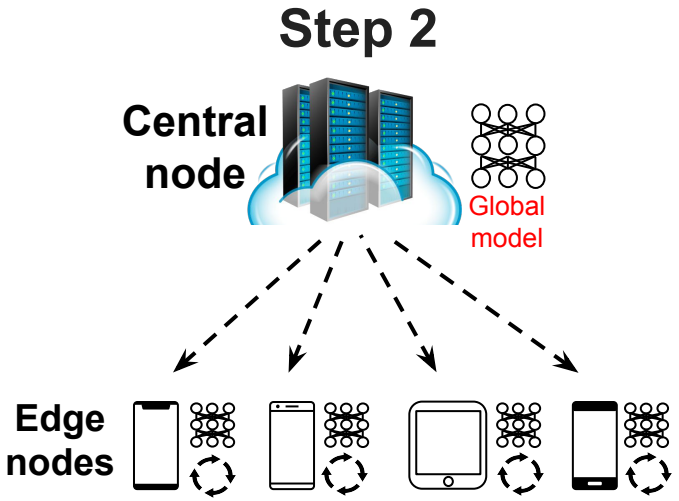

- A global model is initialized on the central node and sent to all participating nodes .

$$w_i = w_{global} \quad \text{For each i}$$



McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

# FedAvg



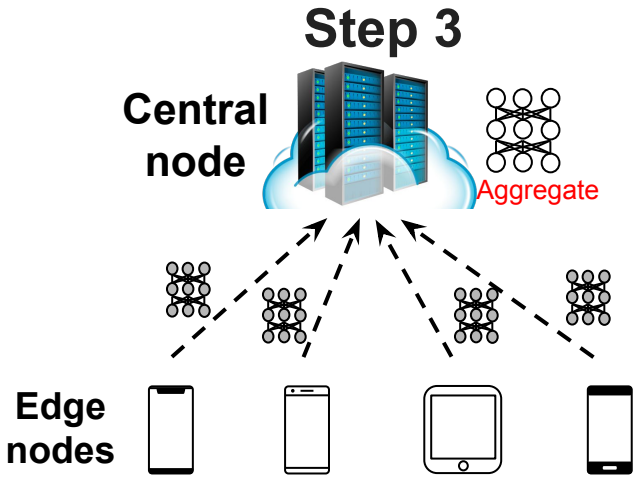

$$w_i' = \underset{w_i}{min} L(F(D_i), Y_i)$$

- Each node i trains the global model locally using its own data for a few epochs.
- The length of local training process may vary.



McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

# FedAvg



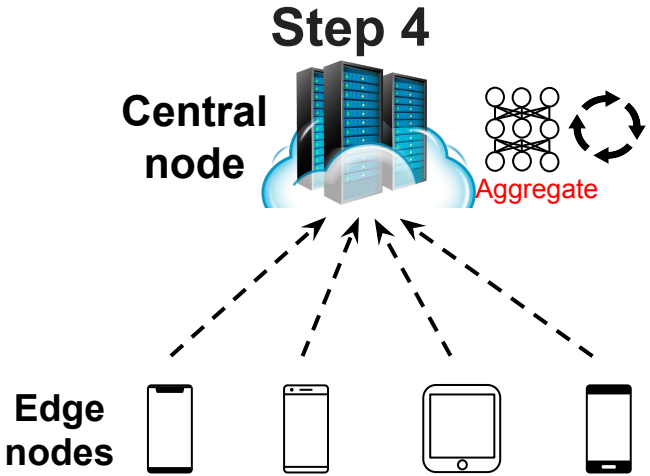

$$\Delta w_i = w - w_i$$

- Local updates are sent from each node to the central node.



McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

# FedAvg




- The central node aggregates the local updates to update the global model.

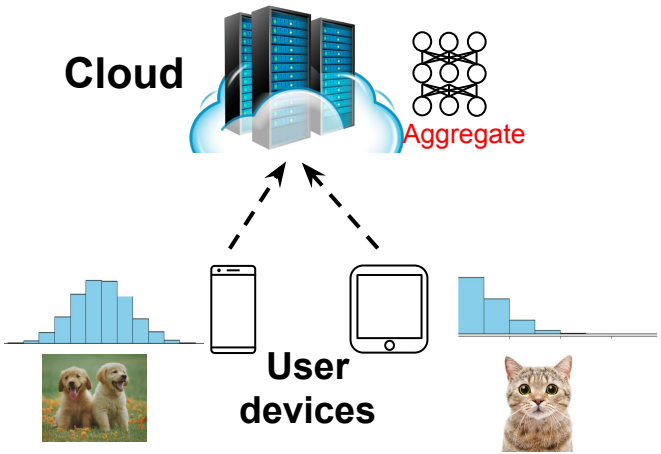
$$w + \frac{1}{N} \sum_i \Delta w_i$$



McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

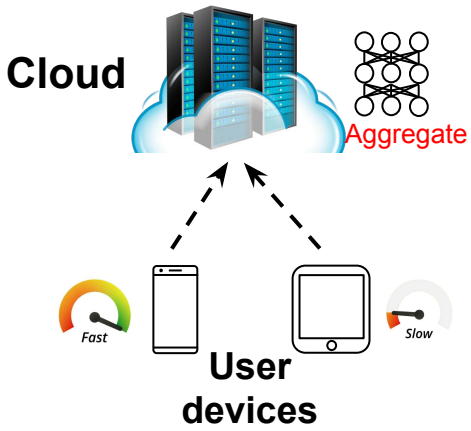
# Federated Learning Problems: Non-IID

- However, in FL, the data distributed across different devices or clients is not drawn from the same statistical distribution.
- Unlike the scenario distributed training, where the training data are randomly distributed. For FL, the data stored in each device is highly biased.
- This may lead to significant accuracy degradation for the global model.



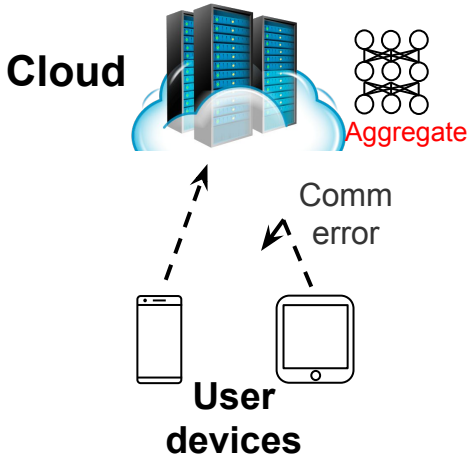

NYU SAI LAB

# Federated Learning Problems: Heterogeneity



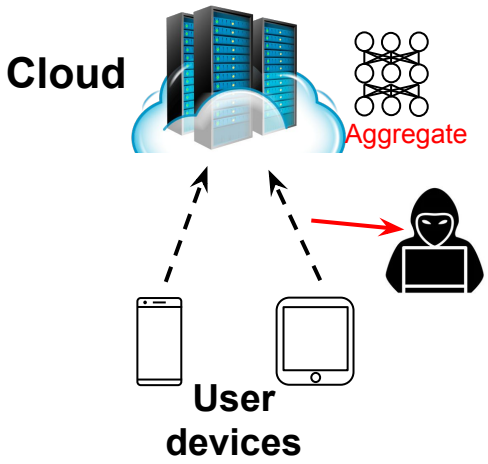

- Different edge device may have different processing speed.
- This will cause the total latency of each training round bottlenecked by the straggler, leading to a slow convergence of the training process.

# Federated Learning Problems: Communication



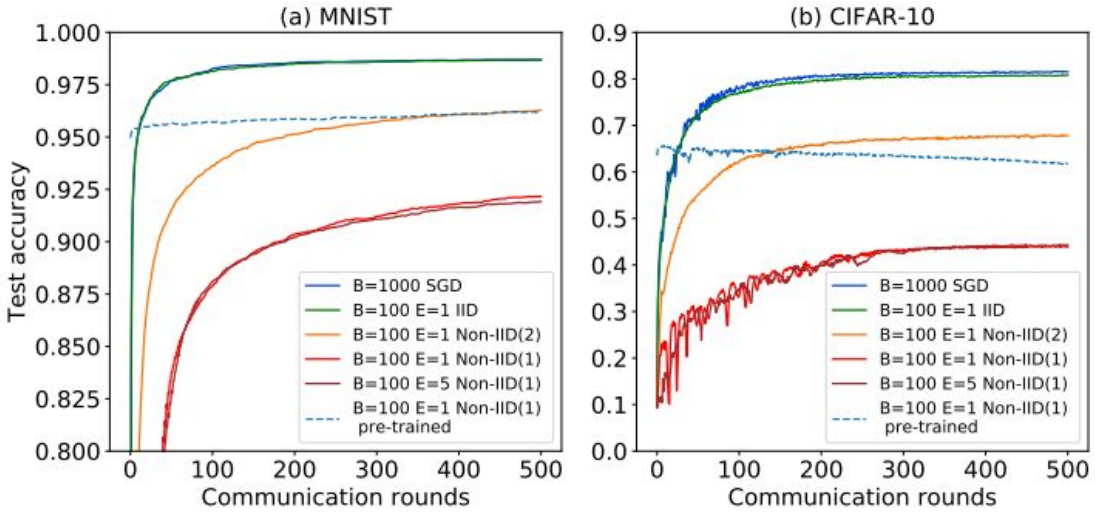

- The communication between edge devices and central cloud may incur transmission loss or error.
- This will impact the training latency and accuracy.

# Federated Learning Problems: Privacy




- The attacker can leverage the transmitted gradient to reconstruct the original input training data.
- This will lead to privacy leakage.

# Federated Learning with Non-iid Data



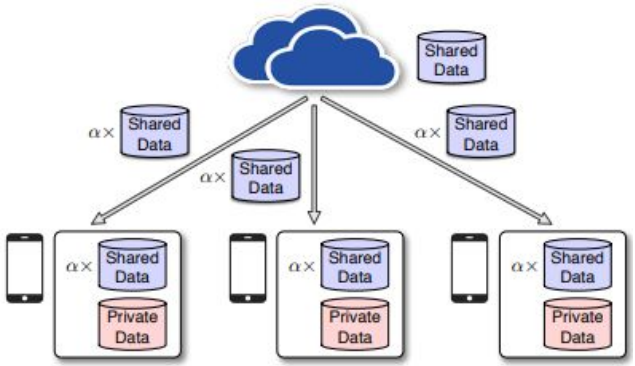

- The training sets are evenly partitioned into 10 clients.
- For IID setting, each client is randomly assigned a uniform distribution over 10 classes.
- For non-IID setting, the data is sorted by class and divided to create two extreme cases: (a) 1-class non-IID, where each client receives data partition from only a single class, and (b) 2-class non-IID, where the sorted data is divided into 20 partitions and each client is randomly assigned 2 partitions from 2 classes.



Zhao, Yue, et al. "Federated learning with non-iid data." *arXiv preprint arXiv:1806.00582* (2018).

# Federated Learning with Non-iid Data

- We propose a data-sharing strategy to improve FedAvg with non-IID data by creating a small subset of data which is globally shared between all the edge devices.
- Experiments show that test accuracy can be increased by ~30% on CIFAR-10 dataset with only 5% globally shared data.




NYU SAI LAB

Zhao, Yue, et al. "Federated learning with non-iid data." *arXiv preprint arXiv:1806.00582* (2018).

# FedProx

**Algorithm 2** `FedProx` (Proposed Framework)

**Input:** $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
**for** $t = 0, \cdots, T-1$ **do**
  Server selects a subset $S_t$ of $K$ devices at random (each device $k$ is chosen with probability $p_k$)
  Server sends $w^t$ to all chosen devices
  Each chosen device $k \in S_t$ finds a $w_k^{t+1}$ which is a $\gamma_k^t$-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w \; h_k(w;\; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in S_t$ sends $w_k^{t+1}$ back to the server
  Server aggregates the $w$'s as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
**end for**

$$\min_w h_k(w;\; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

- We add an extra term to minimize the $l_2$ distance between the initial weight $w_t$ and the learned weight w.
- This loss ensures that the learnt w is not too different from the original w.



Li, Tian, et al. "Federated optimization in heterogeneous networks." *Proceedings of Machine learning and systems* 2 (2020): 429-450.

# Presentation

- FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search (Monish)
- Neural gradients are near-lognormal: improved quantized and sparse training (Rahil)
- Lora: Low-rank adaptation of large language models (Minghui)
- COAT: Compressing Optimizer states and Activation for Memory-Efficient FP8 Training (Yash)
- Federated optimization in heterogeneous networks (Rujuta)

# Project Proposal

- Due on Mar 19th (1 page)
  - Project summary (1 paragraph)
  - Project plan (1 paragraph, gantt chart (optional))
  - Individual responsibilities (1 paragraph)
- You should begin forming teams of 2-3 students and start brainstorming project ideas now.
- Please discuss with me during office hours.
- You should propose something feasible to do.
  - Difficulty, Resource,…

# Some Example

- Study the pruning/quantization behavior for CNN/Diffusion model/Large Models.
- Distillation using Large Model.
- Low precision Parameter efficient finetuning.
- Federated learning with Large Model.
- Hardware simulation on Large Model.