ActNN: Reducing Training Memory Footprint via 2-Bit Activation Compressed Training

Jianfei Chen*, Lianmin Zheng*, Zhewei Yao, Dequan Wang
Ion Stoica, Michael Mahoney, and Joseph Gonzalez
UC Berkeley

Presenter: Jianfei Chen

ICML 2021

AI and Memory Wall

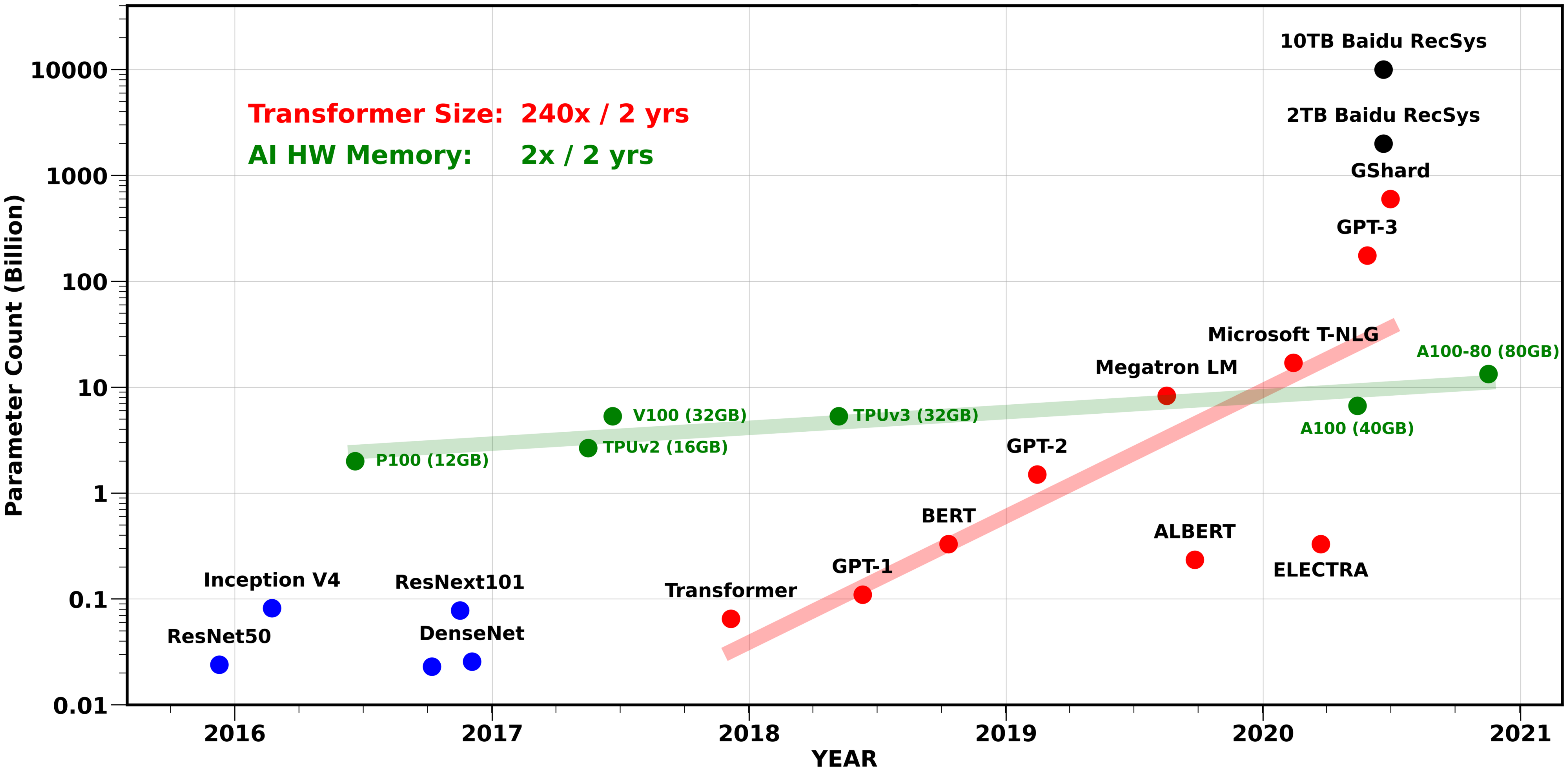
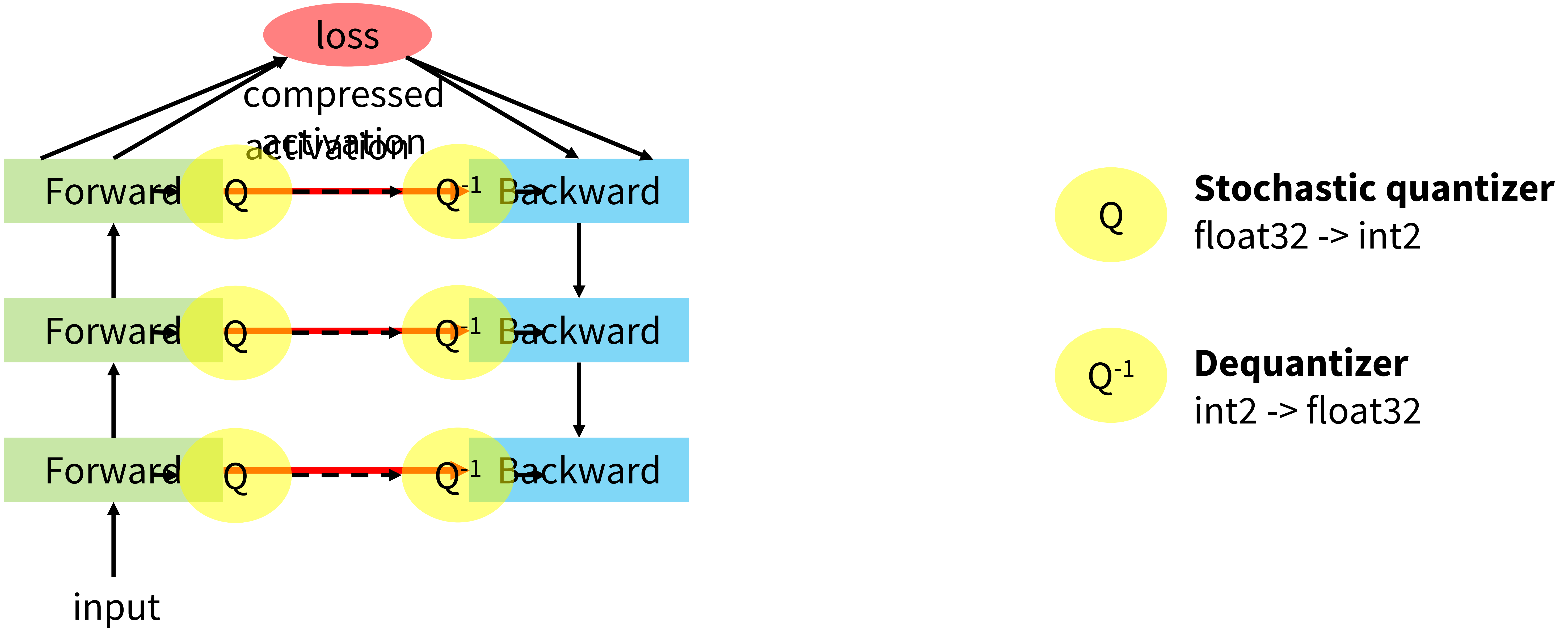


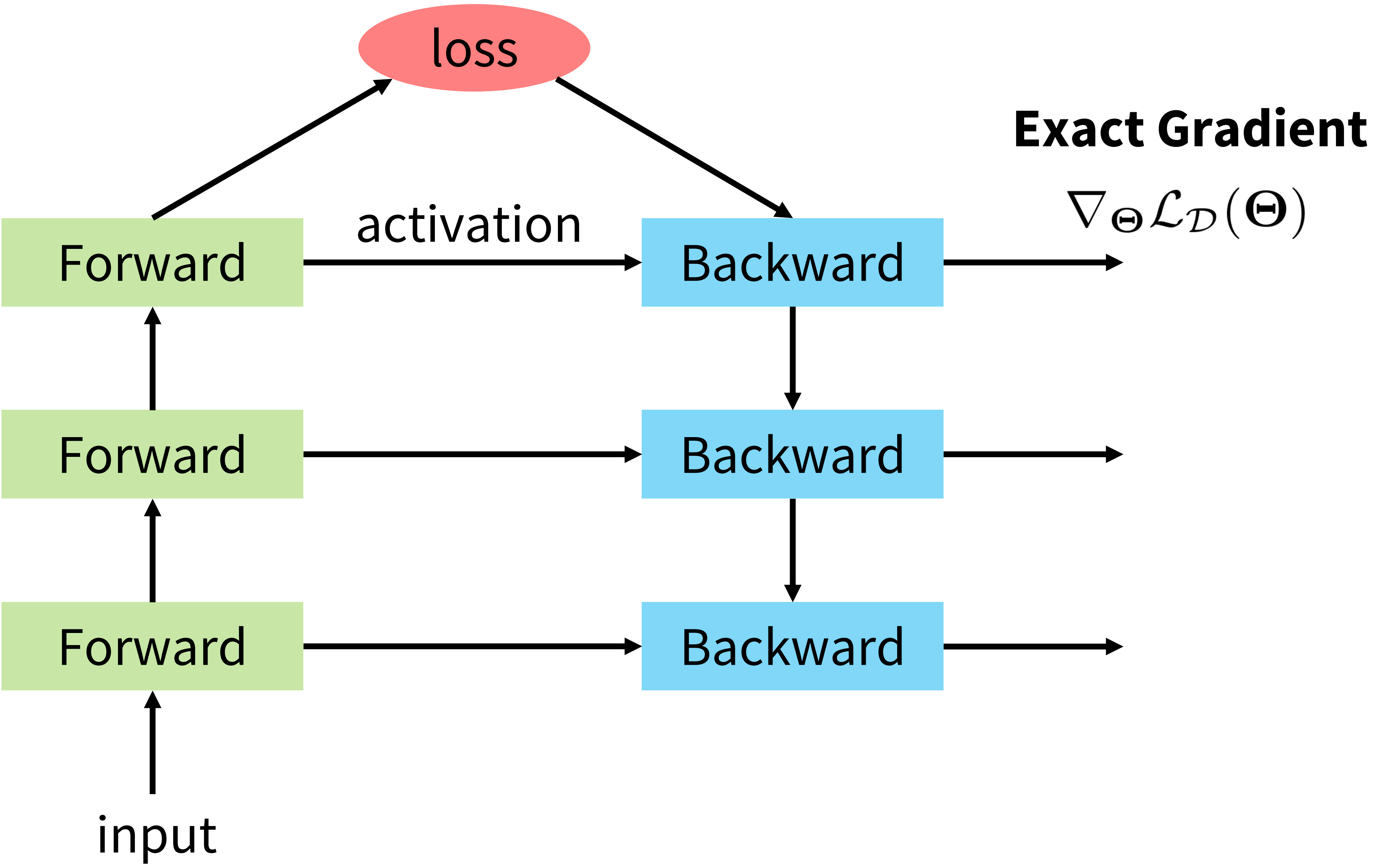
Figure credit: Gholami A, Yao Z, Kim S, Mahoney MW, Keutzer K. AI and Memory Wall. RiseLab Medium Blog Post, University of California Berkeley, 2021, March 29.

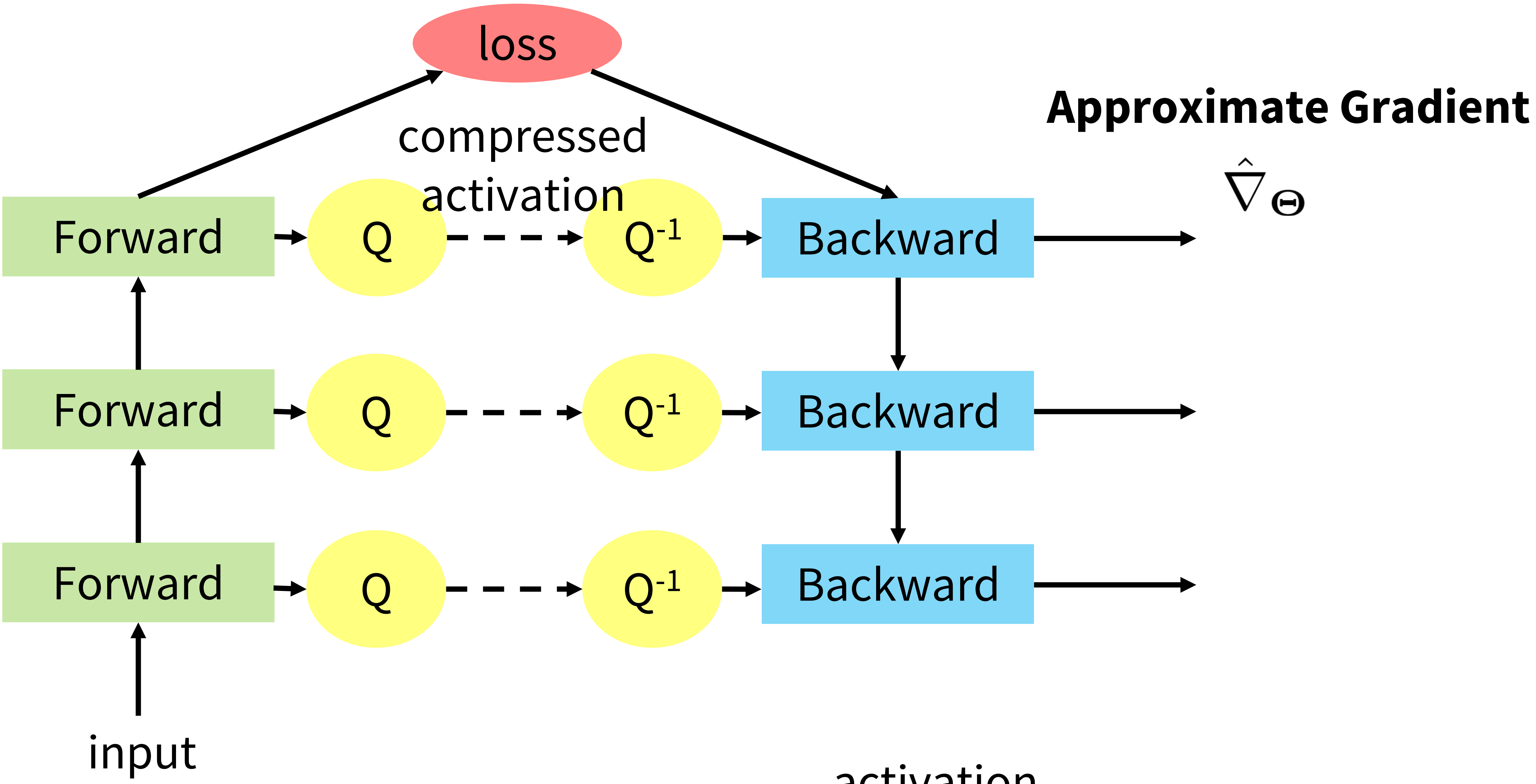
ActNN: Activation Compressed Training of Neural Networks

- In many applications, memory is mainly consumed by the **activations**
- We reduce the training memory footprint by compressing the activations



Unbiased Gradient





activation

Theorem 1. *(Unbiased Gradient) There exists random quantization strategies for* $\hat{\mathbf{C}}$*, such that*

$$\mathbb{E}\left[\hat{\nabla}_{\Theta}\right] = \nabla_{\Theta}\mathcal{L}_{\mathcal{D}}(\mathbf{\Theta}).$$

average over stochastic quantization noise

Convergence

- Stochastic Gradient Descent with unbiased gradient

$$\mathbf{\Theta}_{t+1} \leftarrow \mathbf{\Theta}_t - \alpha \hat{\nabla}_{\mathbf{\Theta}_t}$$

assuming…

A1. The loss $\mathcal{L}_{\mathcal{D}}(\mathbf{\Theta})$ is continuous differentiable and $\nabla \mathcal{L}_{\mathcal{D}}(\mathbf{\Theta})$ is β-Lipschitz continuous.
A2. $\mathcal{L}_{\mathcal{D}}(\mathbf{\Theta})$ is bounded below by $\mathcal{L}_{inf}$.
A3. There exists $\sigma^2 > 0$, such that $\forall \mathbf{\Theta}$, $\mathrm{Var}\left[\hat{\nabla}_{\mathbf{\Theta}}\right] \le \sigma^2$, where for any vector $\mathbf{x}$, $\mathrm{Var}[\mathbf{x}] := \mathbb{E}\|\mathbf{x}\|^2 - \|\mathbb{E}[\mathbf{x}]\|^2$.

Theorem 2. *(Convergence) If A1-A3 holds, and $0 < \alpha \le \frac{1}{\beta}$, take the number of iterations t uniformly from $\{1, \ldots, T\}$, where T is a maximum number of iterations. Then*

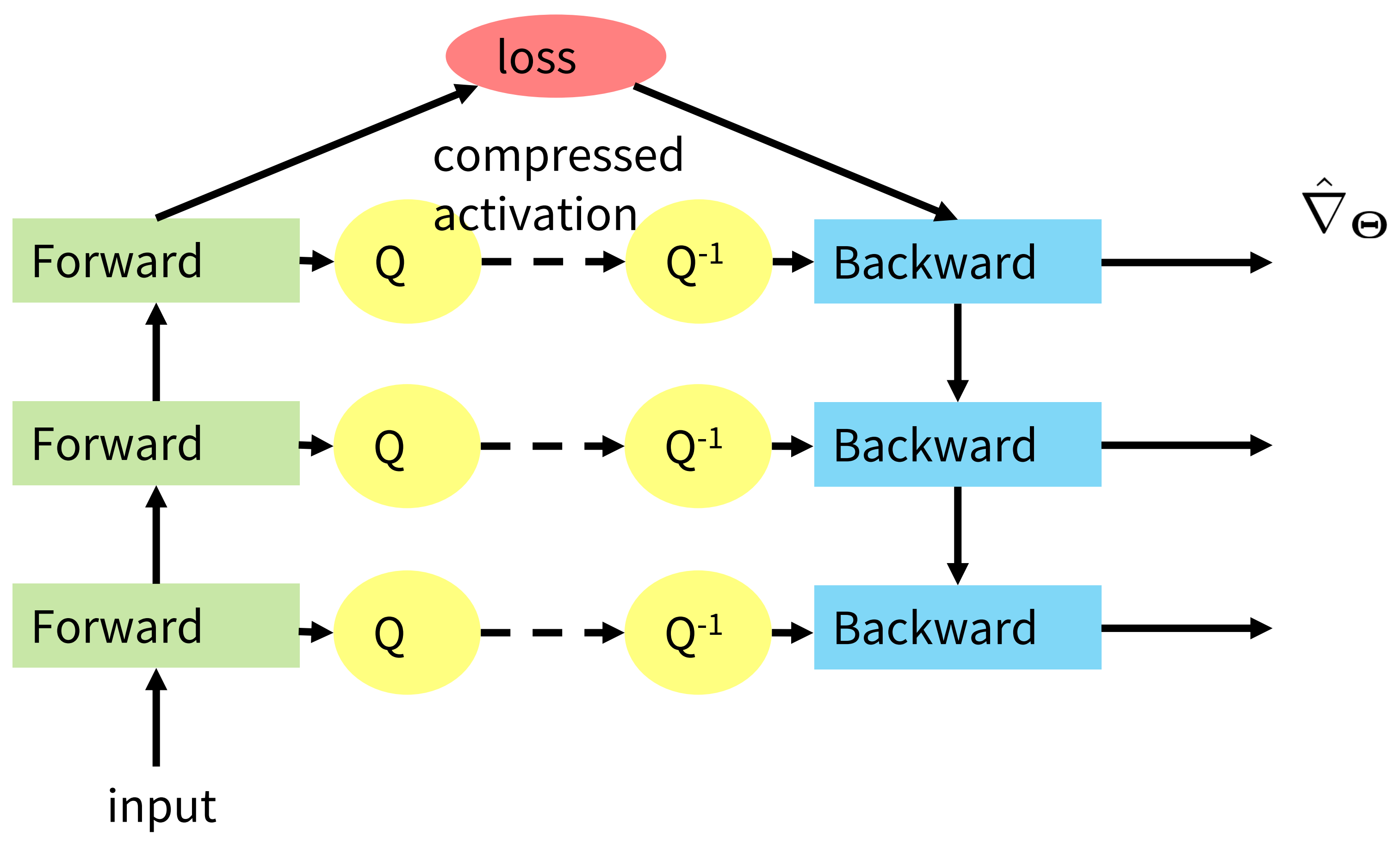
$$\mathbb{E}\left\|\nabla \mathcal{L}_{\mathcal{D}}(\mathbf{\Theta}_t)\right\|^2 \le \frac{2(\mathcal{L}(\Theta_1) - \mathcal{L}_{inf})}{\alpha T} + \alpha\beta\sigma^2. \tag{6}$$

learning rate

smoothness

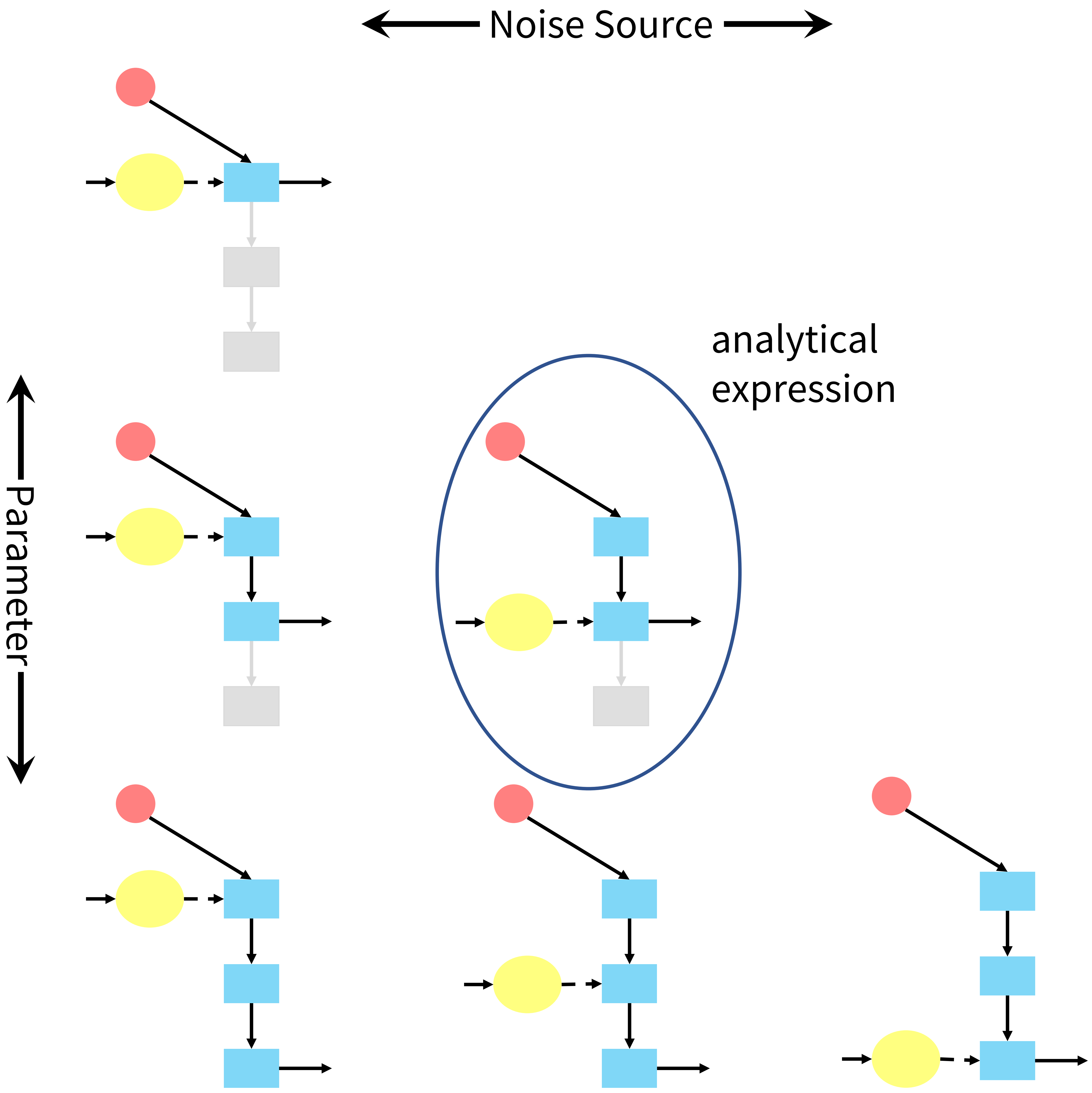
Gradient variance

Gradient Variance



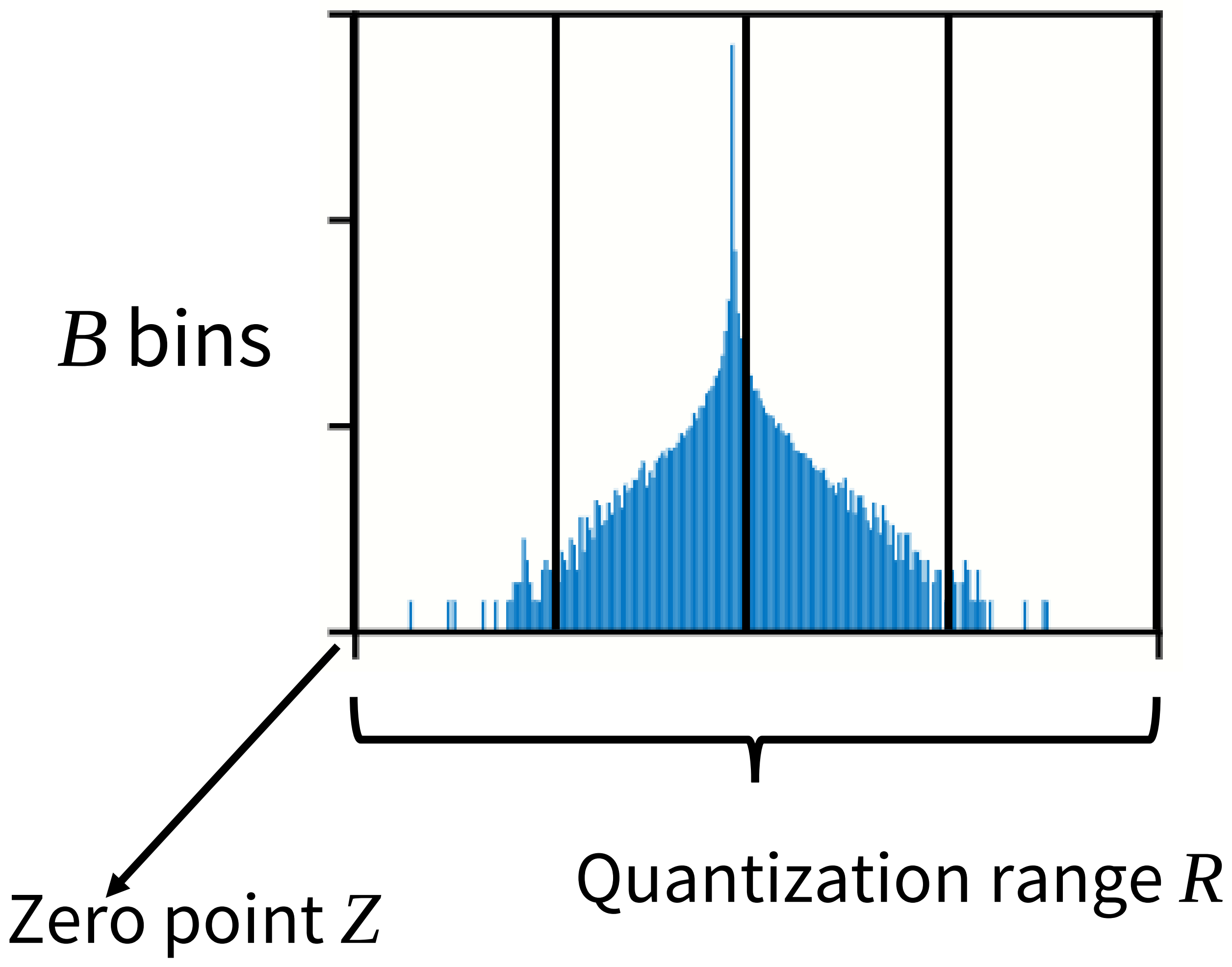
Theorem 3. *(Gradient Variance)*

$$\mathrm{Var}\left[\hat{\nabla}_{\Theta^{(l)}}\right] = \mathrm{Var}\left[\nabla_{\Theta^{(l)}}\right] + \sum_{m=l}^{L} \mathbb{E}\left[\mathrm{Var}\left[\mathbf{G}_{\Theta}^{(l\sim m)}\left(\hat{\nabla}_{\mathbf{H}^{(m)}}, \hat{\mathbf{C}}^{(m)}\right) \mid \hat{\nabla}_{\mathbf{H}^{(m)}}\right]\right].$$



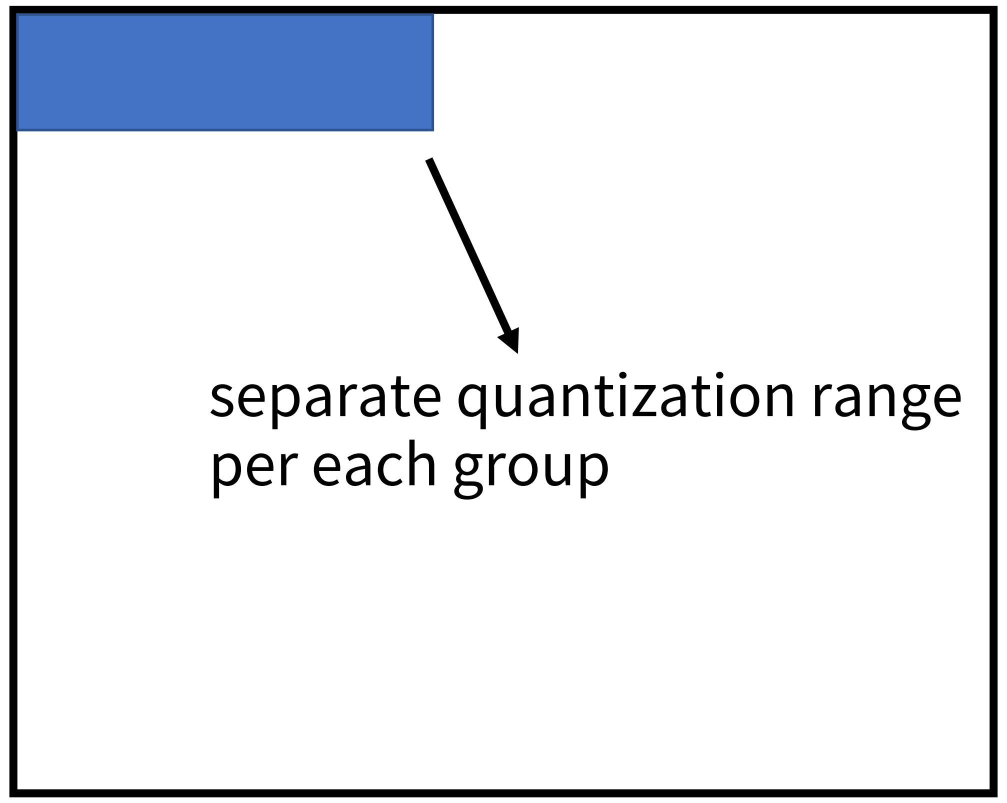
(Per-group) quantization

Good quantizer → fewer bits to achieve convergence → better compression ratio



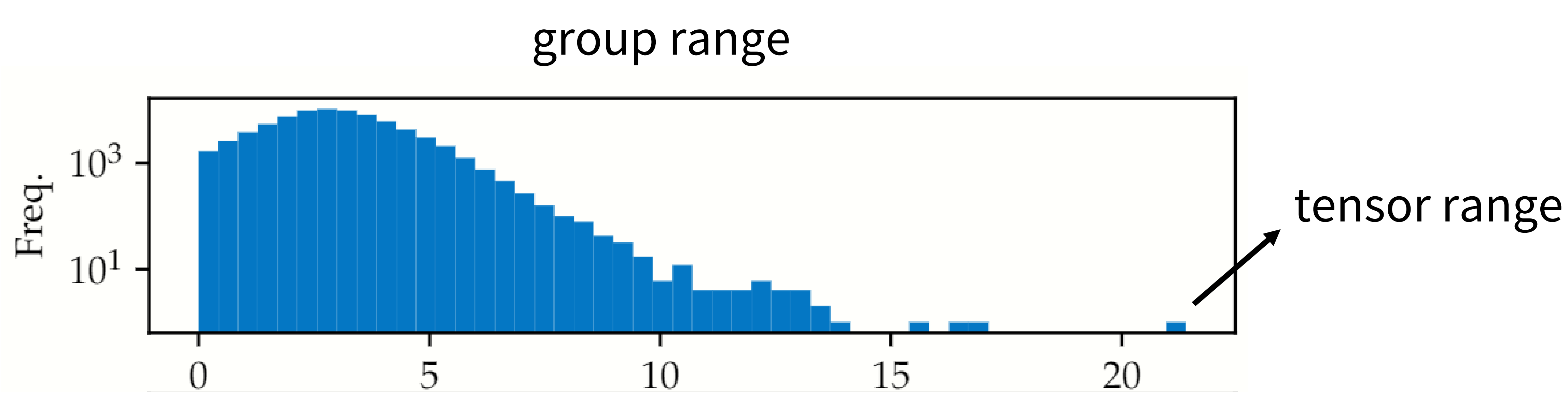
per-tensor quantization

per-group quantization



$$Q(x) = \mathrm{StoRound}(B(x - Z)/R)$$

$$0.7 \rightarrow \begin{cases} 1 & \text{w.p. } 70\% \\ 0 & \text{w.p. } 30\% \end{cases}$$



Fine-Grained Mixed Precision

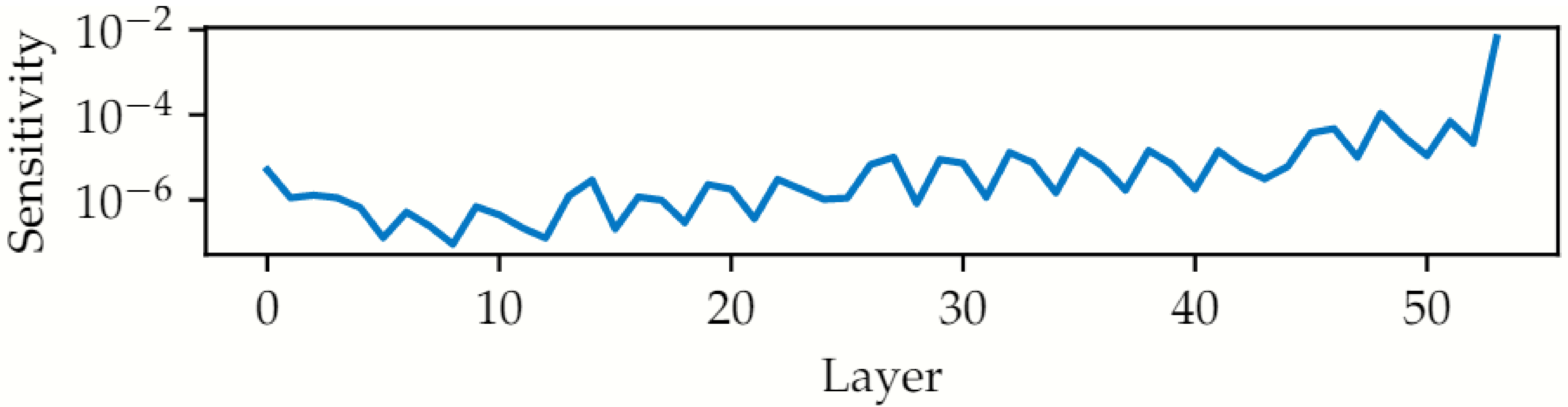
- Each sample / layer has different sensitivity to quantization noise
- The sensitivity can be (approximately) computed **analytically**
- **Minimize the variance** within a given total bits budget

$$\min_{b_n^{(l)}} \sum_{l=1}^{L}\sum_{n=1}^{N} w_n^{(l)}/{B_n^{(l)}}^2 \quad \text{s.t.} \sum_{l=1}^{L} D^{(l)} \sum_{n=1}^{N} b_n^{(l)} \le b_{total}$$

layer sample sensitivity bins

- Allocate the bits dynamically during training

Dense layer: $w_n^{(l)} = \frac{G}{6}\|\hat{\nabla}_{\mathbf{h}_n^{(l)}}\|^2\|\mathbf{R}_n^{(l)}\|^2$



System Implementation

- `actnn`: a collection of activation compressed layers in PyTorch

```
class RegularLayer:
  def forward(context, input):
    context.save_for_backward(input)
    return compute_output(input)

  def backward(context, grad_output):
    input = context.saved_tensors
    return compute_gradient(grad_output, input)

class ActivationCompressedLayer:
  def forward(context, input):
    context.save_for_backward(compress(input))
    return compute_output(input)

  def backward(context, grad_output):
    input = decompress(context.saved_tensors))
    return compute_gradient(grad_output, input)
```

√ Support arbitrary computational graph

√ Dynamic execution

√ No ahead-of-training overhead

√ Standalone package

√ Combine with other memory-saving techniques

Supported Layers

- Conv / ConvTranspose / Linear
- BatchNorm, SyncBatchNorm
- ReLU, MaxPool

1-line conversion

from original PyTorch layers to ActNN layers

```
import actnn
model = actnn.QModule(model)
```

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(3, 16, 3, stride=2)
    self.relu = nn.ReLU()
    self.bn1 = nn.BatchNorm2d(16)

    self.conv2 = nn.Conv2d(16, 32, 3, stride=2)
    self.bn2 = nn.BatchNorm2d(32)

    self.fc = nn.Linear(32, 10)

model = Net()
```

PyTorch layers

```
class Net(nn.Module):
  def __init__(self):
    super(Net, self).__init__()
    self.conv1 = actnn.QConv2d(3, 16, 3, stride=2)
    self.relu = actnn.QReLU()
    self.bn1 = actnn.QBatchNorm2d(16)

    self.conv2 = actnn.QConv2d(16, 32, 3, stride=2)
    self.bn2 = actnn.QBatchNorm2d(32)

    self.fc = actnn.QLinear(32, 10)
```

Memory-efficient ActNN layers

12x activation memory compression

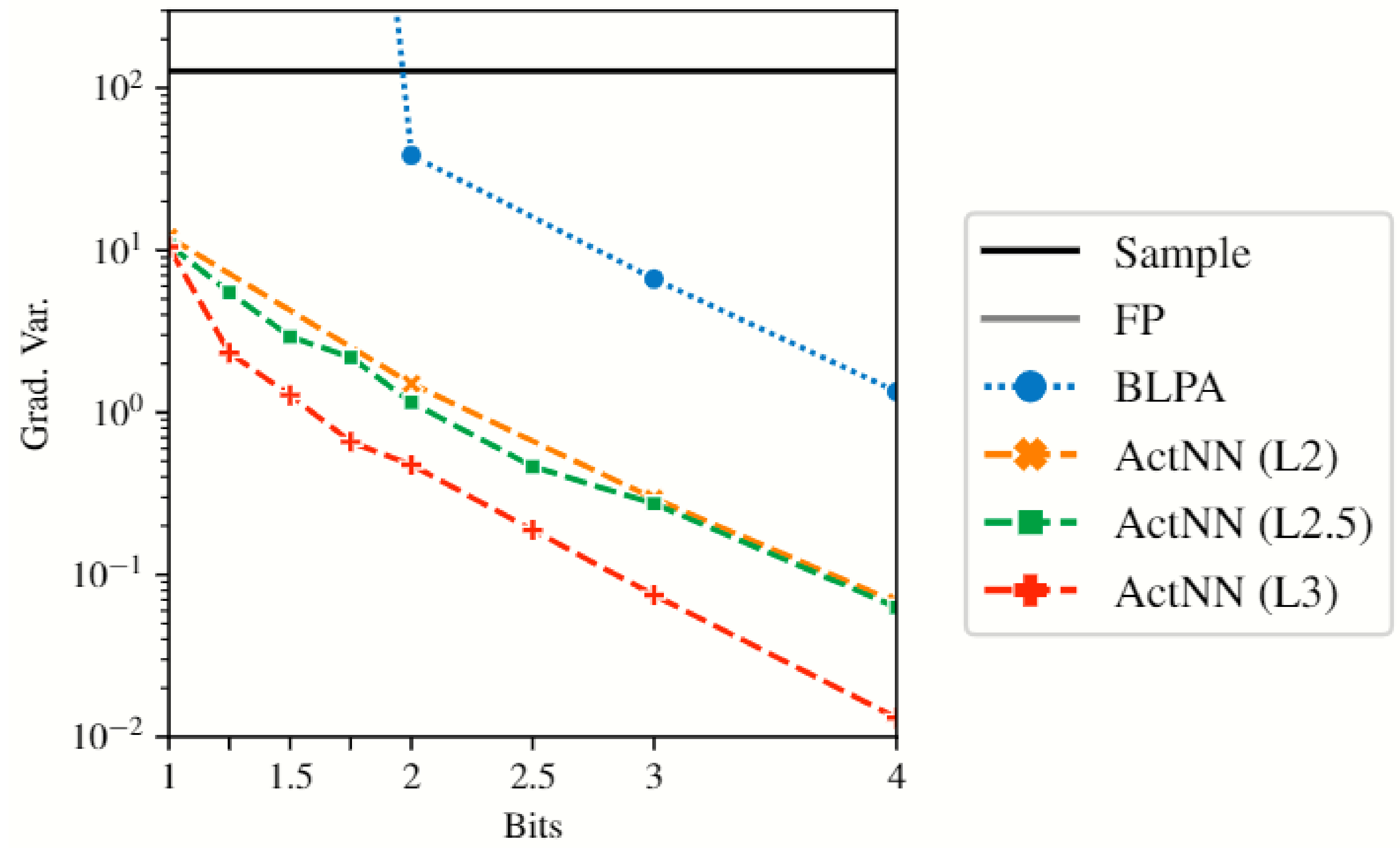
Empirical Convergence

ResNet50 on ImageNet

BLPA: Chakrabarti, Ayan, and Benjamin Moseley. "Backprop with approximate activations for memory-efficient network training." *NeurIPS'19*

	Bits	32	4	3	2	1.5	1.25
	FP	**77.1**	N/A	N/A	N/A	N/A	N/A
	BLPA	N/A	**76.6**	Div.	Div.	N/A	N/A
pergroup	ActNN (L2)	N/A	-	**77.4**	0.1	N/A	N/A
pergroup + persample MP	ActNN (L2.5)	N/A	-	-	**77.1**	75.9	75.1
pergroup + persample/layer MP	ActNN (L3)	N/A	-	-	**76.9**	76.4	75.9

N/A: not available
Div.: diverge
"-": skipped since lower precision achieves lossless results



Near-lossless results (<0.5%) on all our benchmarks

- Segmentation: HRNet, Dilation8, FPN
- Detection: RetinaNet
- Self-supervised learning: MoCov2, BYOL

Activation Memory Reduction

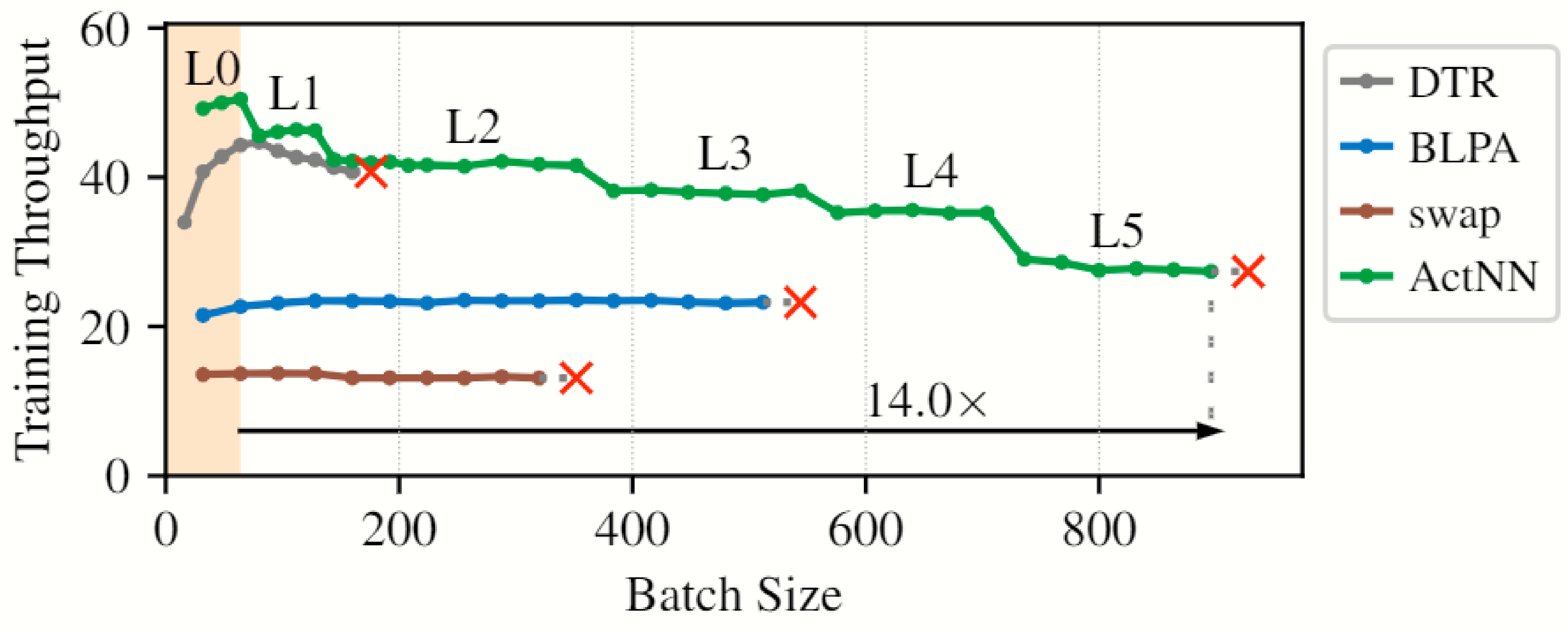
- 2-bit quantization reduces activation memory by 12x

Network	Batch	Total Mem. (GB)			Act. Mem. (GB)		
		FP	ActNN (L3)	R	FP	ActNN (L3)	R
ResNet-152	32	6.01	1.18	5×	5.28	0.44	12×
	64	11.32	1.64	7×	10.57	0.88	12×
	96	OOM	2.11	/	OOM	1.32	/
	512	OOM	8.27	/	OOM	7.01	/
FCN-HR-48	2	5.76	1.39	4×	4.76	0.39	12×
	4	10.52	1.79	6×	9.52	0.79	12×
	6	OOM	2.17	/	OOM	1.18	/
	20	OOM	4.91	/	OOM	3.91	/

Large Batch Size Training

Maximum batch size for ResNet-152

with a Nvidia T4 (16GB)



optimization levels

Level	Compression Strategy	Bits
L0	Do not compress	32
L1	per-group quantization for conv. layers	4, 32
L2	per-group quantization	4
L3	L2 + fine-grained mixed-precision	2
L4	L3 + swapping	2
L5	L4 + defragmentation	2

√ ActNN can be **combined** with other memory-efficient training techniques (e.g. swapping)

√ and other quantized training techniques (e.g., AMP)

DTR: Kirisame, Marisa, et al. "Dynamic tensor rematerialization." *ICLR'21*
BLPA: Chakrabarti, Ayan, and Benjamin Moseley. "Backprop with approximate activations for memory-efficient network training." *NeurIPS'19*

Larger Model

- ActNN enables training larger models without additional resources
- Example: scaling up ResNet-152

Comparison of the largest models ActNN can train before out-of-memory with the same batch size(64) with a Nvidia Tesla T4 (16GB)

	Dim.	Maximum Value			Training Throughput (TFLOPS)		
		FP	ActNN (L3)	ActNN (L4)	FP	ActNN (L3)	ActNN (L4)
Depth	D	160	660	1016	0.59	0.46	0.38
Width	W	92	332	340	0.70	1.07	1.09
Resolution	R	240	636	740	0.59	0.46	0.42

Summary

- Reducing Memory Footprint by Quantizing the activation to 2-bits
- Convergence Guarantee with SGD
- Adaptive Quantization Techniques
- A Plug-and-Play PyTorch library

```python
import actnn
model = actnn.QModule(model)
```

github.com/ucbrise/actnn

Supported Layers

- Conv / ConvTranspose / Linear
- BatchNorm, SyncBatchNorm
- ReLU, MaxPool

Tested Models

- Classification: ResNet / DenseNet
- Segmentation: HRNet / Dilation8 / FPN
- Detection: FPN
- Self-supervised learning: MoCov2, BYOL

Thanks!

github.com/ucbrise/actnn