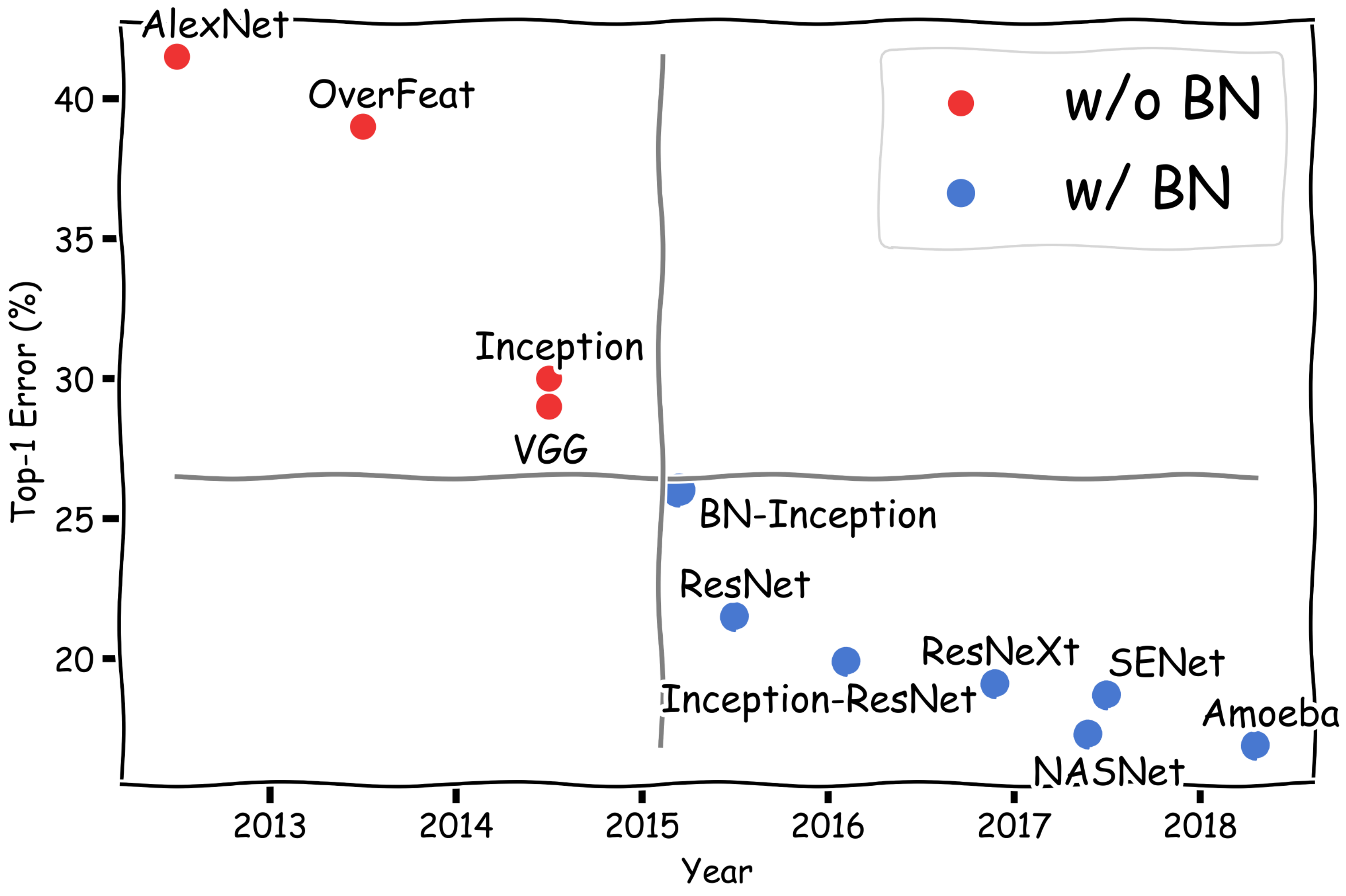
Devils in BatchNorm

Yuxin Wu

Facebook AI Research

Batch Normalization – a Milestone



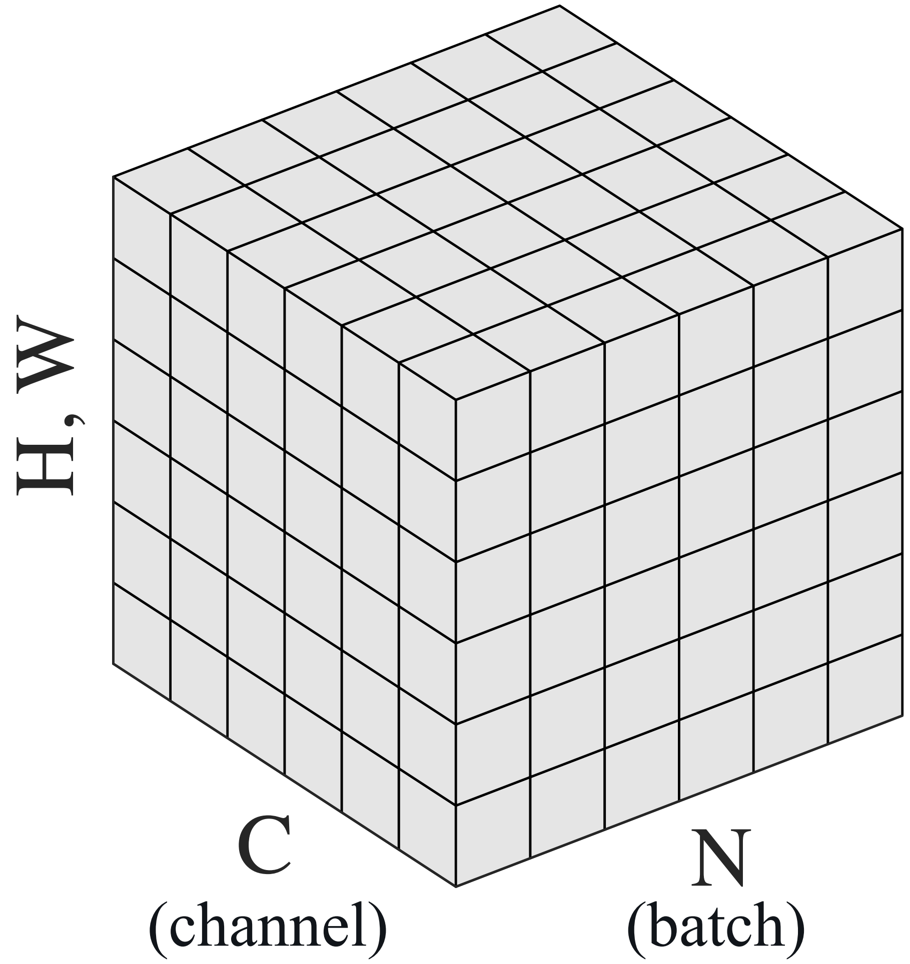
Batch Normalization – a Necessary Evil

- "Batch normalization in the mind of many people, including me, is a necessary evil. In the sense that nobody likes it, but it kind of works, so everybody uses it, but everybody is trying to replace it with something else because everybody hates it" – Yann Lecun

- "A very common source of bugs" – CS231n 2019 Lecture7

Computer Vision News 11/2018, Page 9. https://www.rsipvision.com/ComputerVisionNews-2018November/Computer%20Vision%20News.pdf
CS231n 2019 Lecture 7, Page 74. http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture07.pdf

What's Batch Norm



Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ICML* 2015.

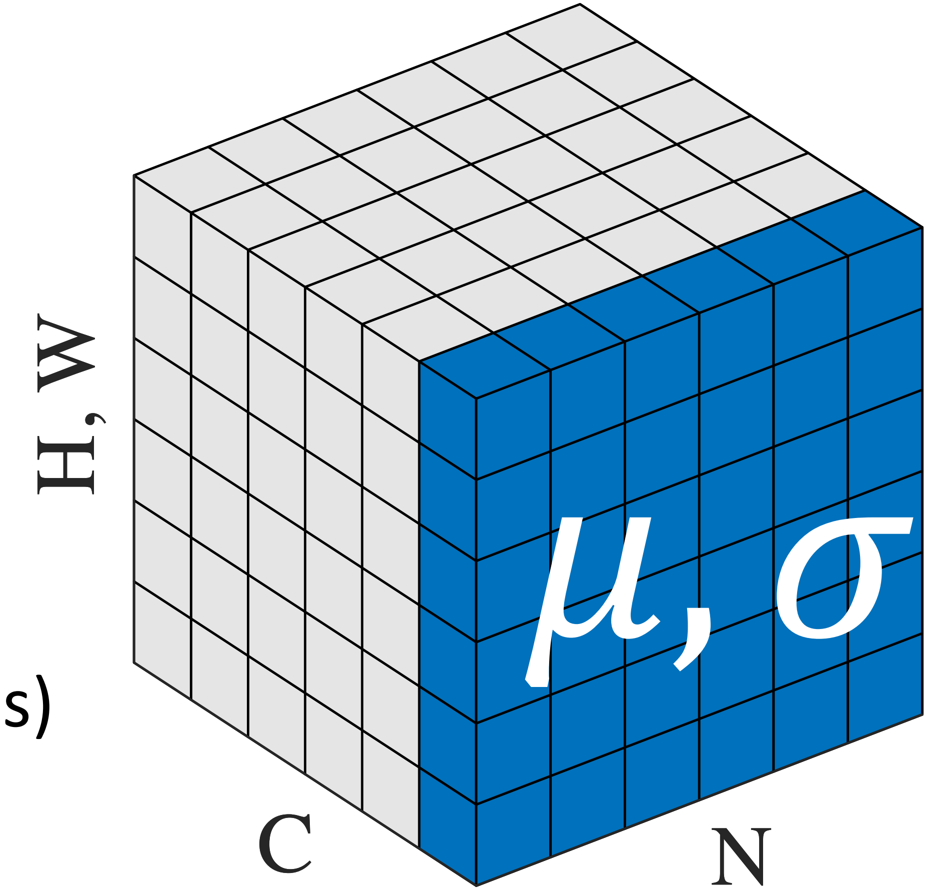
What's Batch Norm: Training Time

- Batch ...

- Normalization!

$$\hat{x} = \frac{x - \mu_B}{\sigma_B}$$

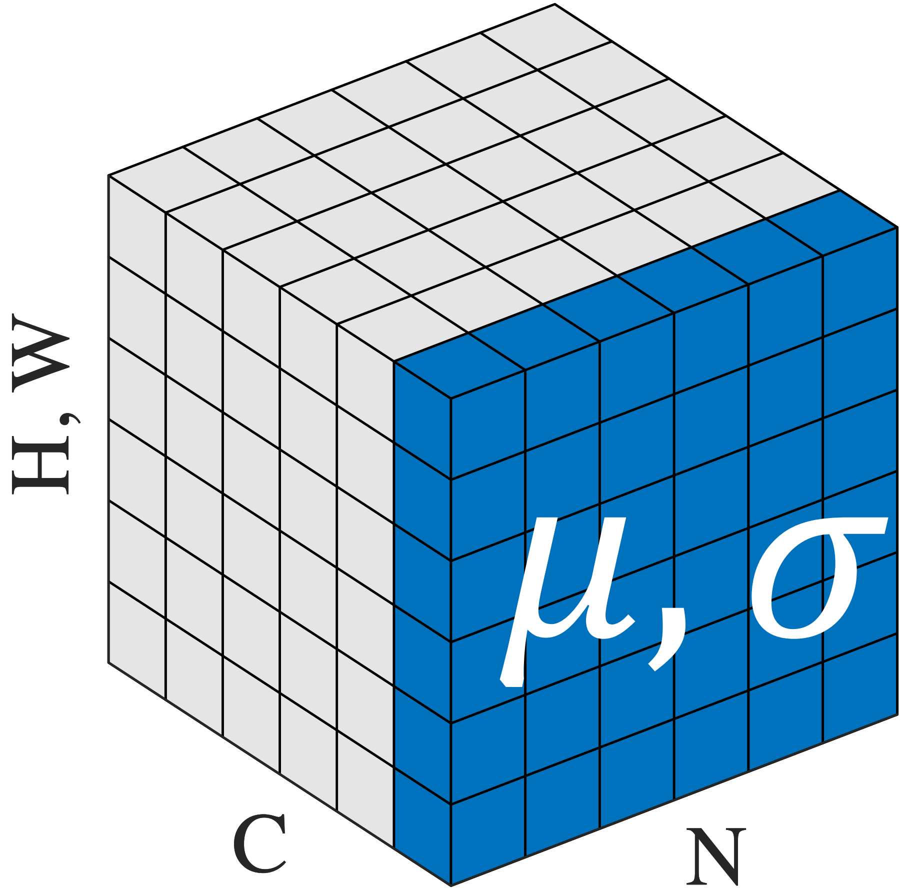
- And more …
  - Channel-wise affine (won't discuss)
  - Train/test inconsistency



Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ICML* 2015.

What's Batch Norm

- $\mu_B, \sigma_B^2 = \text{mean}, \text{var}(x, \text{axis}=[N, H, W])$
- Training time:
  - $\hat{x} = \frac{x - \mu_B}{\sigma_B}$
  - $\mu_{EMA} \leftarrow \lambda \mu_{EMA} + (1 - \lambda)\mu_B$
  - $\sigma_{EMA}^2 \leftarrow \cdots$
- Test time:
  - No concept of "batch"
  - $\hat{x} = \frac{x - \mu_{EMA}}{\sigma_{EMA}}$



Ioffe, Sergey, and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." *ICML* 2015.

BatchNorm’s Effect: Optimization

- Faster/Better Convergence
- Insensitive to initialization
- Stable Training (enable various networks to be trained)

~~Batch~~Norm's Effect: Optimization

- Faster/Better Convergence
- Insensitive to initialization
- Stable Training (enable various networks to be trained)

What's special about BatchNorm?

- Training: $\hat{x} = \frac{x - \mu_B}{\sigma_B}$ Testing: $\hat{x} = \frac{x - \mu_{EMA}}{\sigma_{EMA}}$

Inconsistency!

- Why inconsistency works?
  - Think about Dropout, or data augmentation
  - μ_{EMA}, σ_{EMA} approximate $\mathrm{E}[\mu_B], \mathrm{E}[\sigma_B]$
  - μ_B, σ_B are (slightly) noisy versions of the EMA

Why inconsistency works?

μ_{EMA}, σ_{EMA} approximate $\mathrm{E}[\mu_B], \mathrm{E}[\sigma_B]$

μ_B, σ_B are (slightly) noisy versions of the EMA

When does BatchNorm fail?

When μ_{EMA}, σ_{EMA} does not approximate μ_B, σ_B

1. When EMA are not computed properly
2. When μ_B, σ_B are not stable -- cannot be approximated well
   a) Unstable data
   b) Unstable model

Devils in testing:
EMA update

EMA update: devils in testing

- $\mu_{EMA} \leftarrow \lambda\, \mu_{EMA} + (1-\lambda)\mu_B, \sigma_{EMA}^2 \leftarrow \cdots$
- What makes EMA a bad approximation of $\mathrm{E}[\mu_B], \mathrm{E}[\sigma_B]$?
  - Small λ, EMA biased. $\text{typical } \lambda = 0.9{\sim}0.99$
  - Large λ , insufficient iterations (λ=0.99, N>1000)
  - Unstable model or data in last N iterations

- Typical error: “false overfitting” when EMA is bad

Say Goodbye to EMA

- EMA is:
  - Always biased
  - Always estimated on non-stationary data
  - Just a cheap version of “true average”
- We need True Average!

[Precise BatchNorm]

Precise BatchNorm

- Stop training, compute true $\mathrm{E}[\mu_B], \mathrm{E}[\sigma_B]$ with N iterations
- Small overhead
- Used in ResNet – but never became popular:
    - λ large enough
    - Trained long enough
    - Model is stable: converged well enough in the end
- However …

Example 1: when you need Precise BatchNorm

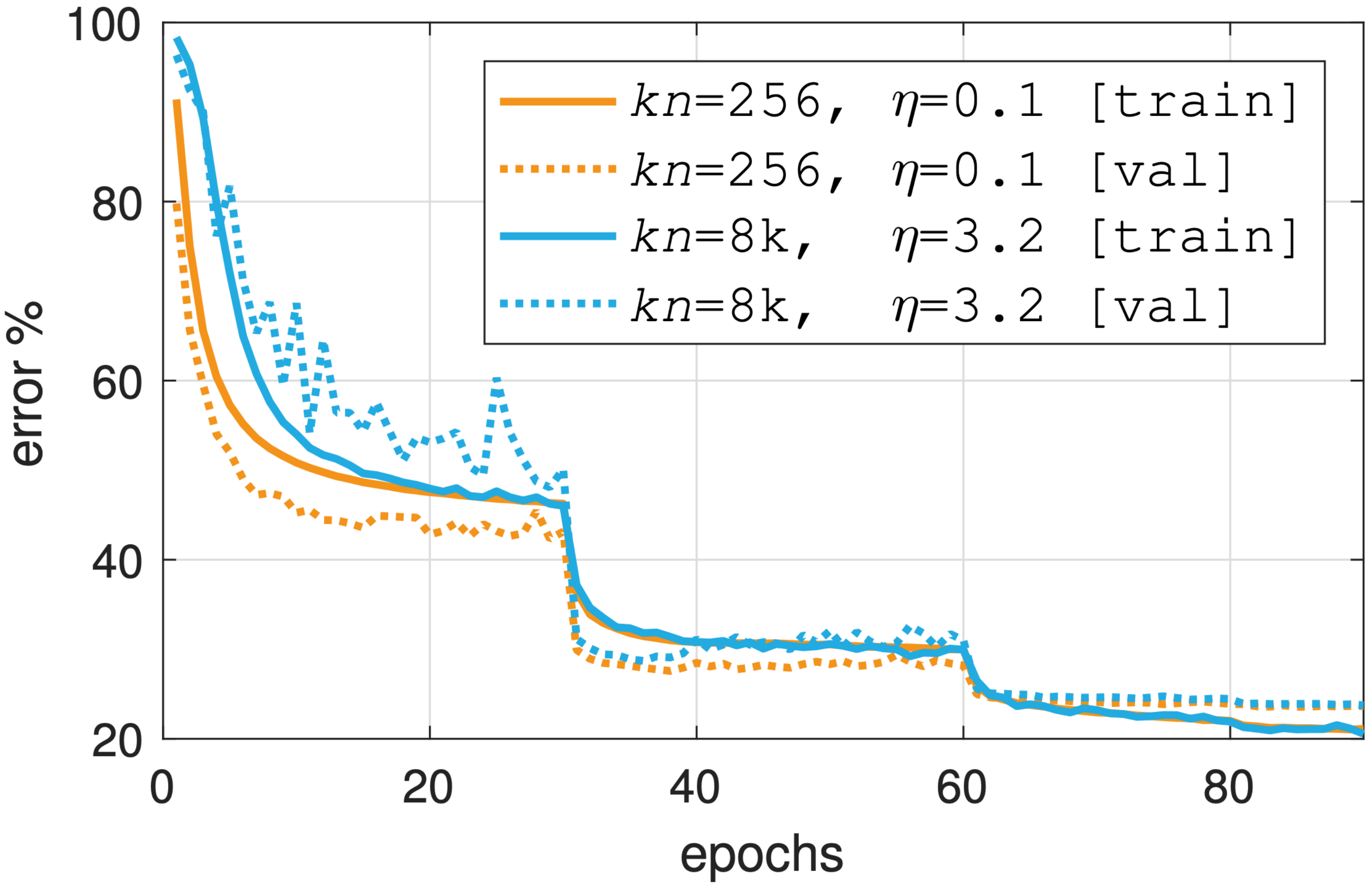


Fig 4, ImageNet in 1 hour

Priya et. al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Example 1: when you need Precise BatchNorm

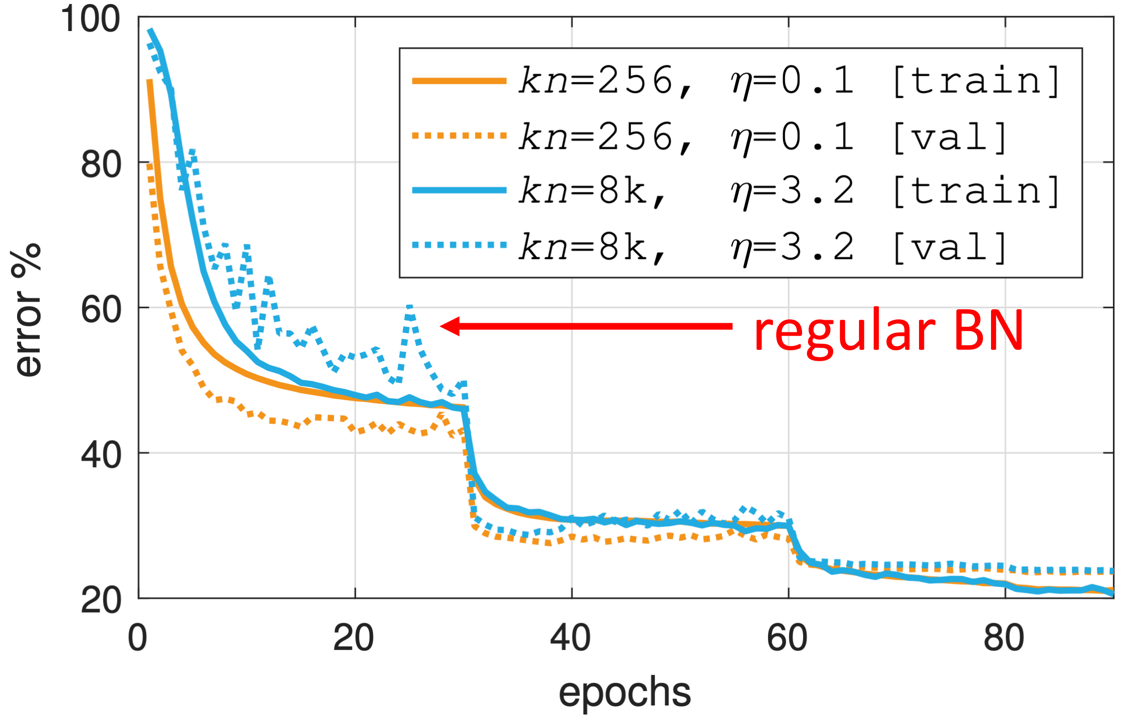


Fig 4, ImageNet in 1 hour
train & val err of large batch training

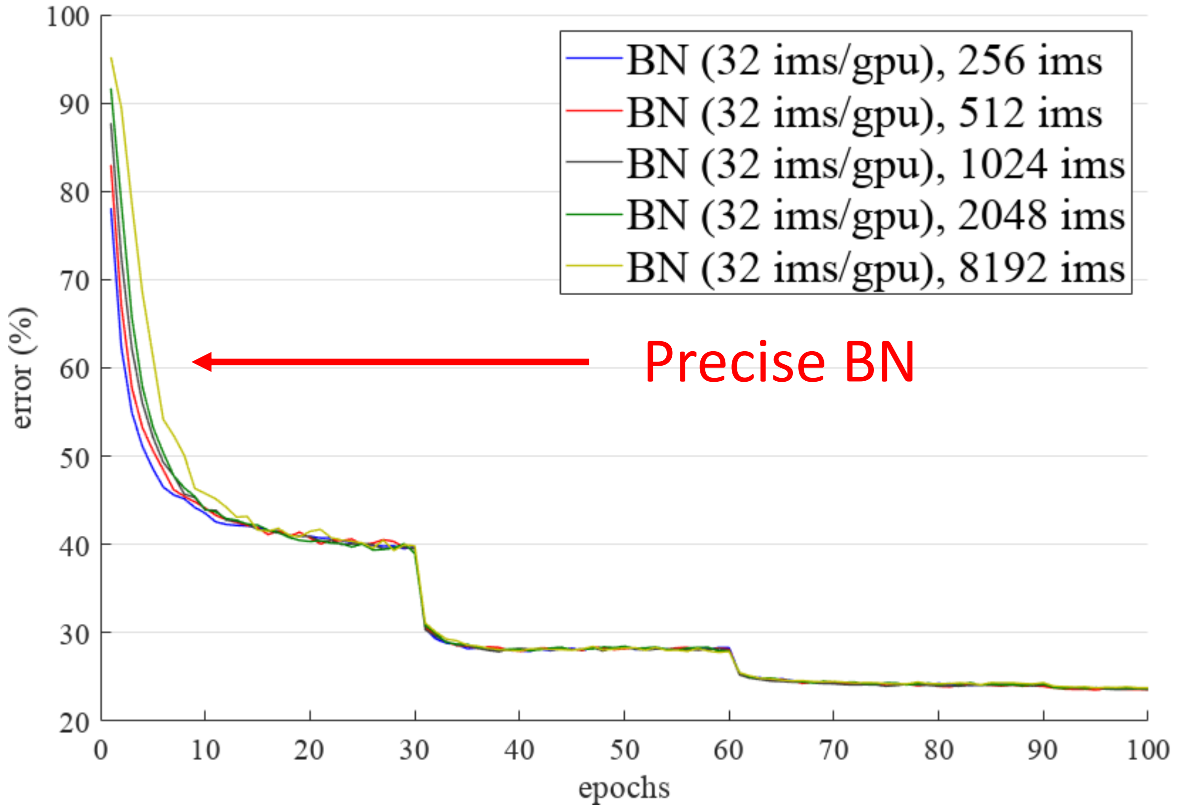
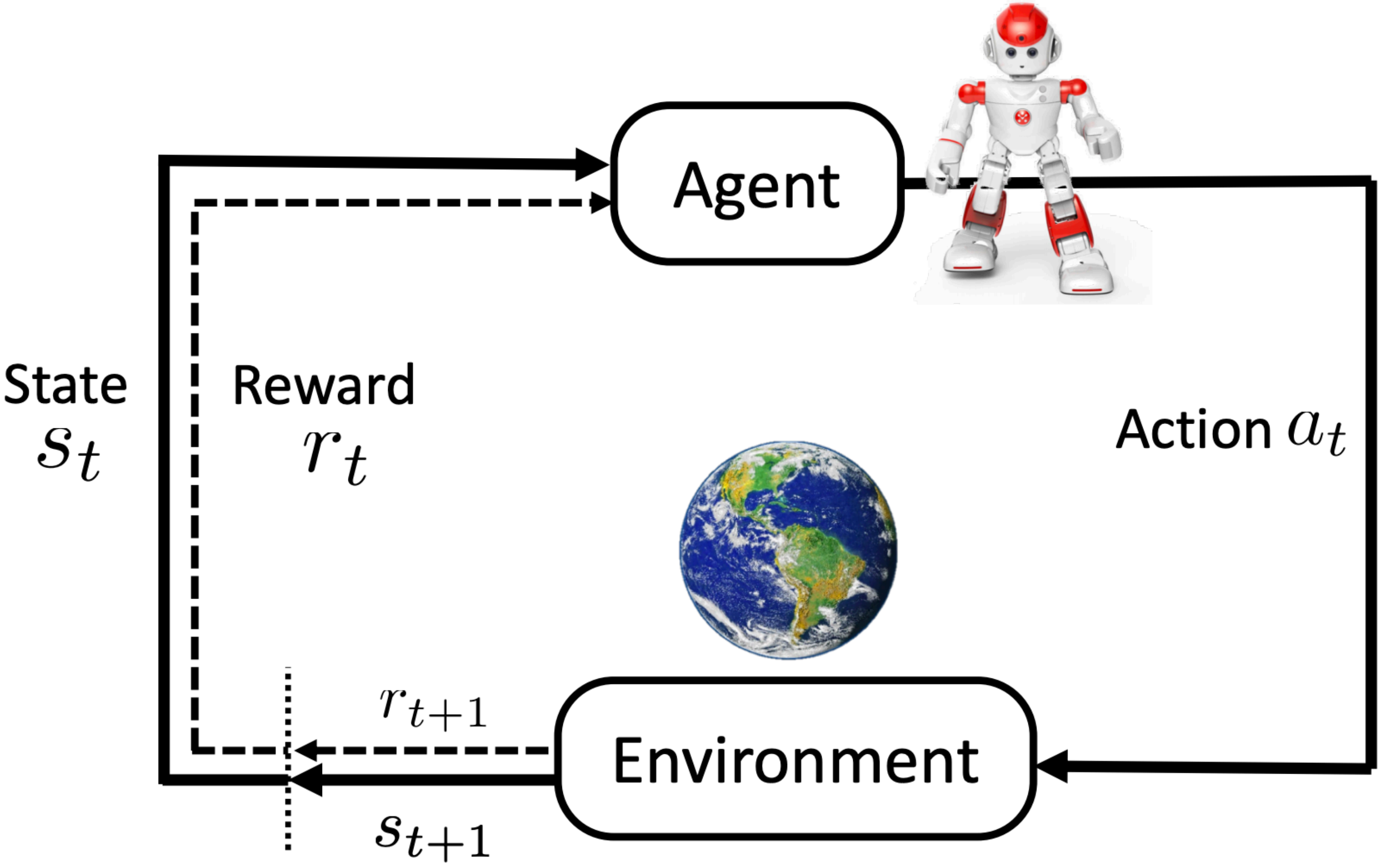


Fig 9, Group Normalization
(IJCV version)
val err of large batch training

Priya et. al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour
Yuxin Wu and Kaiming He. ”Group Normalization“ *IJCV*

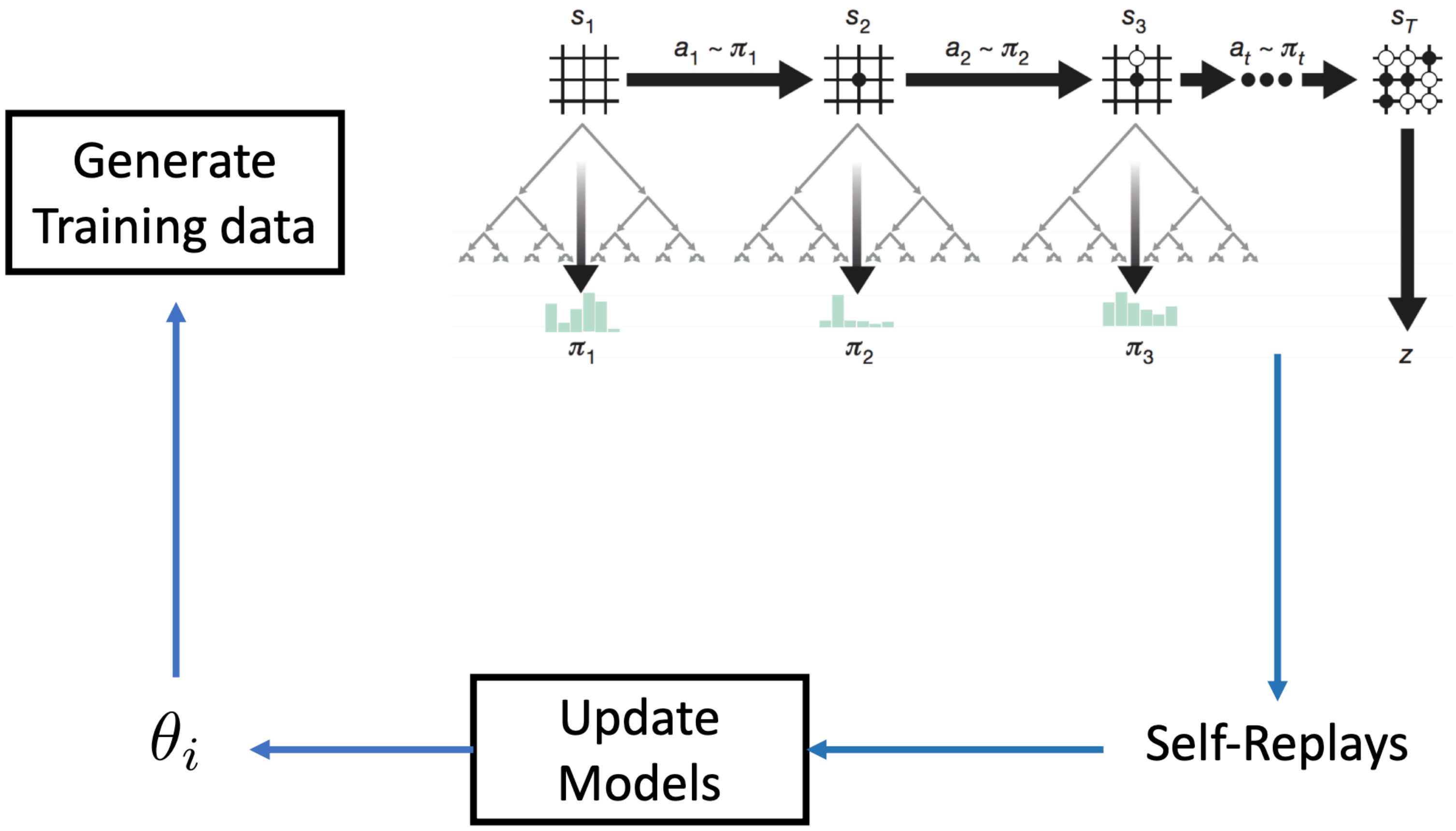
Example 2: when you need Precise BatchNorm



Reinforcement Learning

Yuandong Tian, Building Scalable Framework and Environment of Reinforcement Learning, https://yuandong-tian.com/deeplearning_summit.pdf

Example 2: when you need Precise BatchNorm



Reinforcement Learning: AlphaGo Zero

Yuandong Tian, Building Scalable Framework and Environment of Reinforcement Learning, https://yuandong-tian.com/deeplearning_summit.pdf

Example 2: when you need Precise BatchNorm

Batch normalization moment staleness Our residual network model, like that of AGZ and AZ, uses batch normalization (Ioffe & Szegedy, 2015). Most practical implementations of batch normalization use an exponentially weighted buffer, parameterized by a "momentum constant", to track the per-channel moments. We found that even with relatively low values of the momentum constant, the buffers would often be stale (biased), resulting in subpar performance.

"Moment Staleness" in ELF OpenGO

Yuandong Tian, et. al., ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero, ICML2019

When you need Precise BatchNorm?

- When you need to inference, and EMA is unstable, because:
  - Model did not stay stable for sufficient iterations
- Implementation:
  - Cheap Precise BN: just update EMA using:
    - high λ
    - a fixed model
  - Many variants are OK

Devils in training:
batch size

“Normalization batch size”

- Normalization batch != SGD batch
  - Historical implementation: per-GPU BN
    - Cannot easily tune: speed vs. memory
  - Today: Sync BN, Ghost BN, Virtual BN

- ImageNet in 1 hour setup:
  - Change “SGD batch size” & LR
  - Keep “normalization batch size” at 32

Priya et. al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

Devils in training: normalization batch size

- Training: $\hat{x} = \frac{x - \mu_B}{\sigma_B}$ Testing: $\hat{x} = \frac{x - \mu_t}{\sigma_t}$
- μ_B, σ_B: have noise from other samples in a batch
  - noise: sth you can never fit
- Small NBS -> large noise; large NBS -> small noise

Noise is Regularization!

Need Proper Regularization

Tuning the “normalization batch size”

Norm BS	2	4	8	16	32	64	128	1024
train err	30	28	26	24	22	21	20	18
val err	35	27	25	23.7	23.6	23.6	23.7	23.9

ResNet-50 on ImageNet

- NBS controls regularization strength
- Small NBS -> large noise -> poor optimization
- Large NBS -> small noise -> overfitting

(assume i.i.d. data)

Implementations: Sync BatchNorm/Cross-GPU BN

- To increase NBS, compute μ_B, σ_B of a larger batch across GPUs
- Implemented by all-reduce $2 \times C$ elements: $E[x], E[x^2]$
- Slight time/memory overhead.

- Available in Tensorpack/PyTorch/MXNet; easy to implement in TF

Chao Peng, et. al., MegDet: A Large Mini-Batch Object Detector

Implementations: Ghost BatchNorm

- To decrease NBS, just split large batch to small ones for normalization.

- Available in TF/tensorpack (`virtual_batch_size=`)
- Easy to implement in any library

Elad Hoffer, et. al., Train longer, generalize better: closing the generalization gap in large batch training of neural networks

Implementations: Virtual BatchNorm

- To increase NBS, use more images to run forward-only
- Slight memory overhead; Large time overhead
- More controllable
- Not popular

Tim Salimans, et. al., Improved Techniques for Training GANs

Implementations: Accumulate Gradients

- To keep NBS, while changing SGD batch size
- Save gradients, and update the model once a while instead

- Available in tensorpack; easy to implement in PyTorch/TF/MXNet

Related: Batch Renormalization

- Training: $\hat{x} = \frac{x - \mu_B}{\sigma_B} \times \text{stop\_gradient}(r) + \text{stop\_gradient}(d)$

  Testing: $\hat{x} = \frac{x - \mu_{EMA}}{\sigma_{EMA}}$
- r, d pushes μ_B, σ_B similar to μ_{EMA}, σ_{EMA}
- Reduce noise & inconsistency

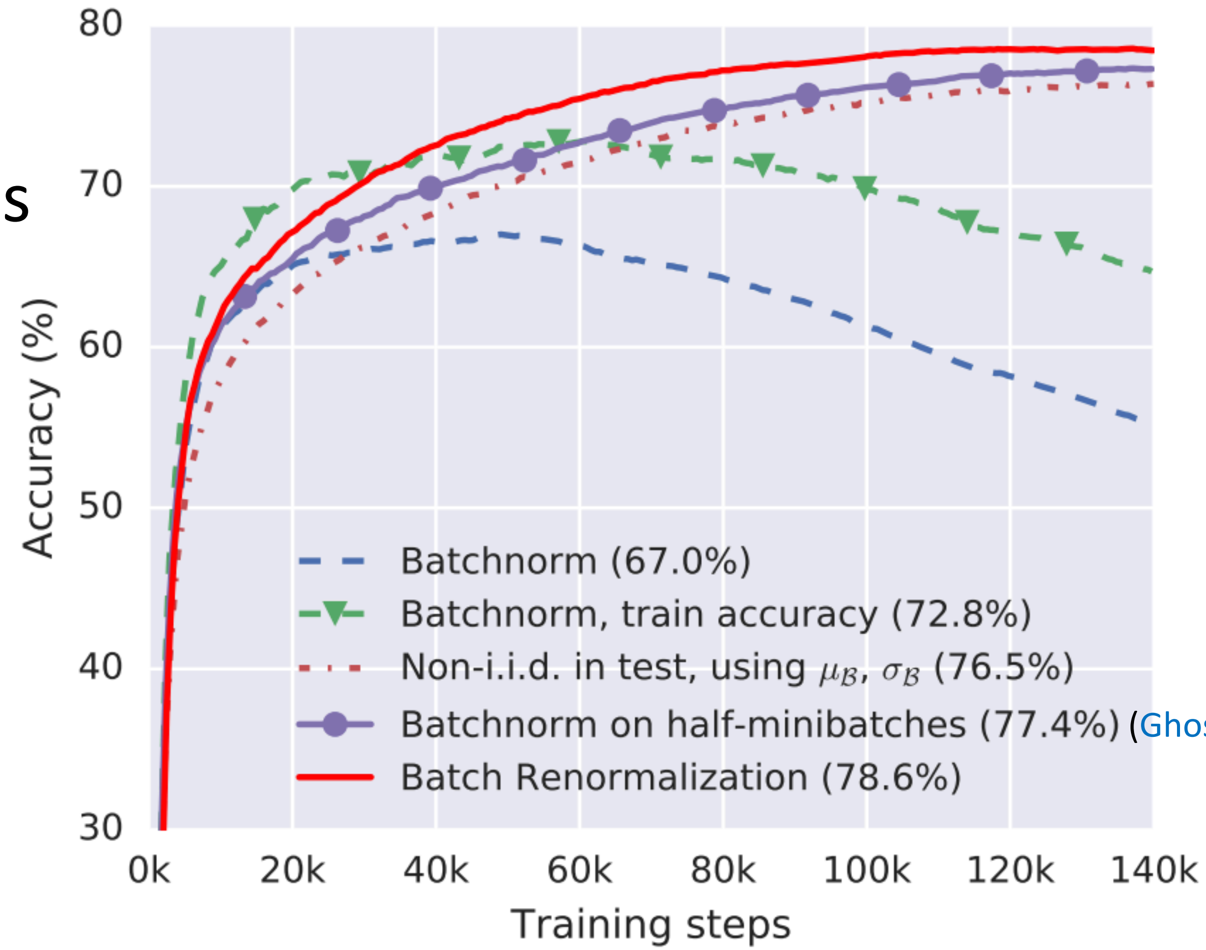
- Need to tune the limit on r, d

Sergey Ioffe, et. al., Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models

Devils in training / fine-tuning:
data distribution

Devils in training / fine-tuning: data distribution

- Non-i.i.d. data:
  - NBS = 32 = 16 labels x 2 samples
  - hurt SGD as well



(Ghost BN)

Batch Renorm, Fig 3

Sergey Ioffe, et. al., Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models

When is data distribution non-i.i.d.?

1. When data comes from different sources
    - multi-domain learning
    - adversarial defense training
    - fine-tuning

- Remediations:
    - Training: “Separate BN” statistics for each domain
    - Training/fine-tuning: Frozen BN -- 1~2 constant affine layers
    - Testing: Adaptive BN – recompute precise statistics

Cihang Xie, et. al., Intriguing properties of adversarial training
Yanghao Li, et. al., Revisiting Batch Normalization For Practical Domain Adaptation

When is data distribution non-i.i.d.?

2. GAN: real/fake distribution
    - `D(real_batch, training=True)  # D=Discriminator`
    - `D(fake_batch, training=True, update_ema=False)  # don’t update EMA`
    - `D(fake_batch, training=False)  # use EMA during training`
3. When batch is designed to come from correlated sources
    - two-stage object detector
    - video understanding
4. Data depends on environment: RL
    - target network, Precise BN

Devils in fine-tuning: fusion

Fusion affects fine-tuning

- Pre-trained model sometimes contain fusion
  - e.g. ImageNet models in Detectron
  - Doesn’t hurt if frozen, but:
- Fused models may not be fine-tuned
  - Reparameterization affects gradients
  - $f(x) = w_1 \times (w_2 x + w_3) \rightarrow f(x) = w_2' x + w_3'$

https://github.com/facebookresearch/maskrcnn-benchmark/issues/561

Devils in Implementations

PyTorch: momentum = 0.1

- 0.1 means 0.9
- γ initialized with U(0, 1)
- Legacy inherited from LuaTorch

cuDNN:

- Caffe2's `riv/running_inv_var` is actually running variance
- $\epsilon \geq 10^{-5}$; biased vs. unbiased variance -- might be relevant in conversion
- cuDNN 7: `SPATIAL_PERSISTENT` is inaccurate

Caffe:

- `running_mean` $\leftarrow$ `running_mean / scale_factor`

TensorFlow: delayed EMA update

- Motivation: EMA update does not have to happen immediately
- Reality: no speedup
- Devils:
  - Easy to forget
  - One BN used multiple times
  - BN inside `tf.cond`
  - BN defined but not always used (many GAN implementations)
- Solution: update EMA in the layer

A Good TensorFlow Implementation
(`tensorpack.models.BatchNorm`)

It supports:

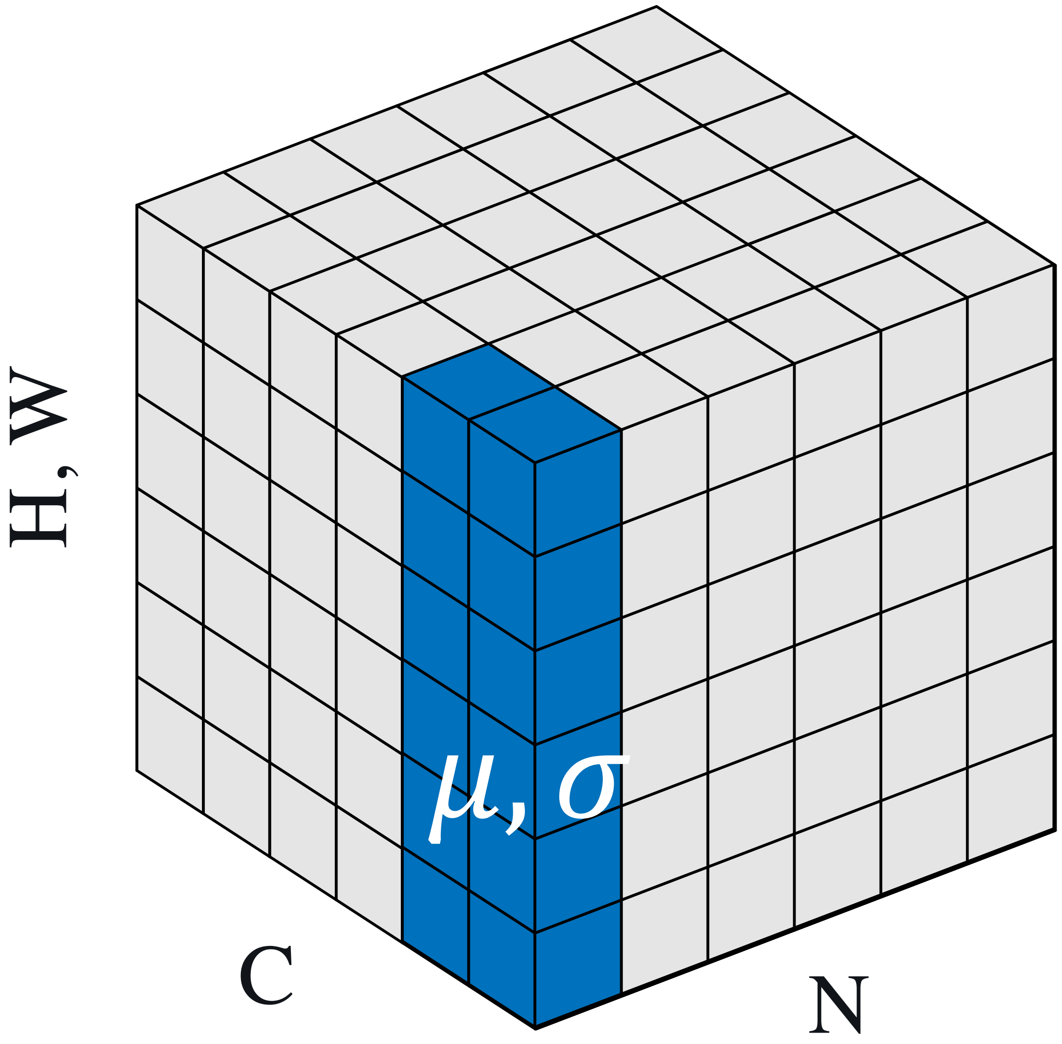
- Choose to use μ_B, σ_B or μ_{EMA}, σ_{EMA}, regardless of mode
- Choose whether to update EMA when using μ_B, σ_B
- Choose how to update EMA (in the layer or not)
- Tune “normalization batch size” with SyncBN / GhostBN

Summary: 10 ways to do BatchNorm

- Which μ, σ ?
  - $\mu_{B,}\sigma_B$; μ_{EMA}, σ_{EMA}; BRN
- How to compute μ_B, σ_B:
  - Per-GPU BN; Sync BN; Ghost BN; Virtual BN
- Whether to update μ_{EMA}, σ_{EMA} with μ_B, σ_B:
  - YES; NO; Separate BN
- What to use for testing / fine-tuning:
  - EMA; Precise BN; Adaptive BN; Frozen BN

Appendix:
Other Normalizations

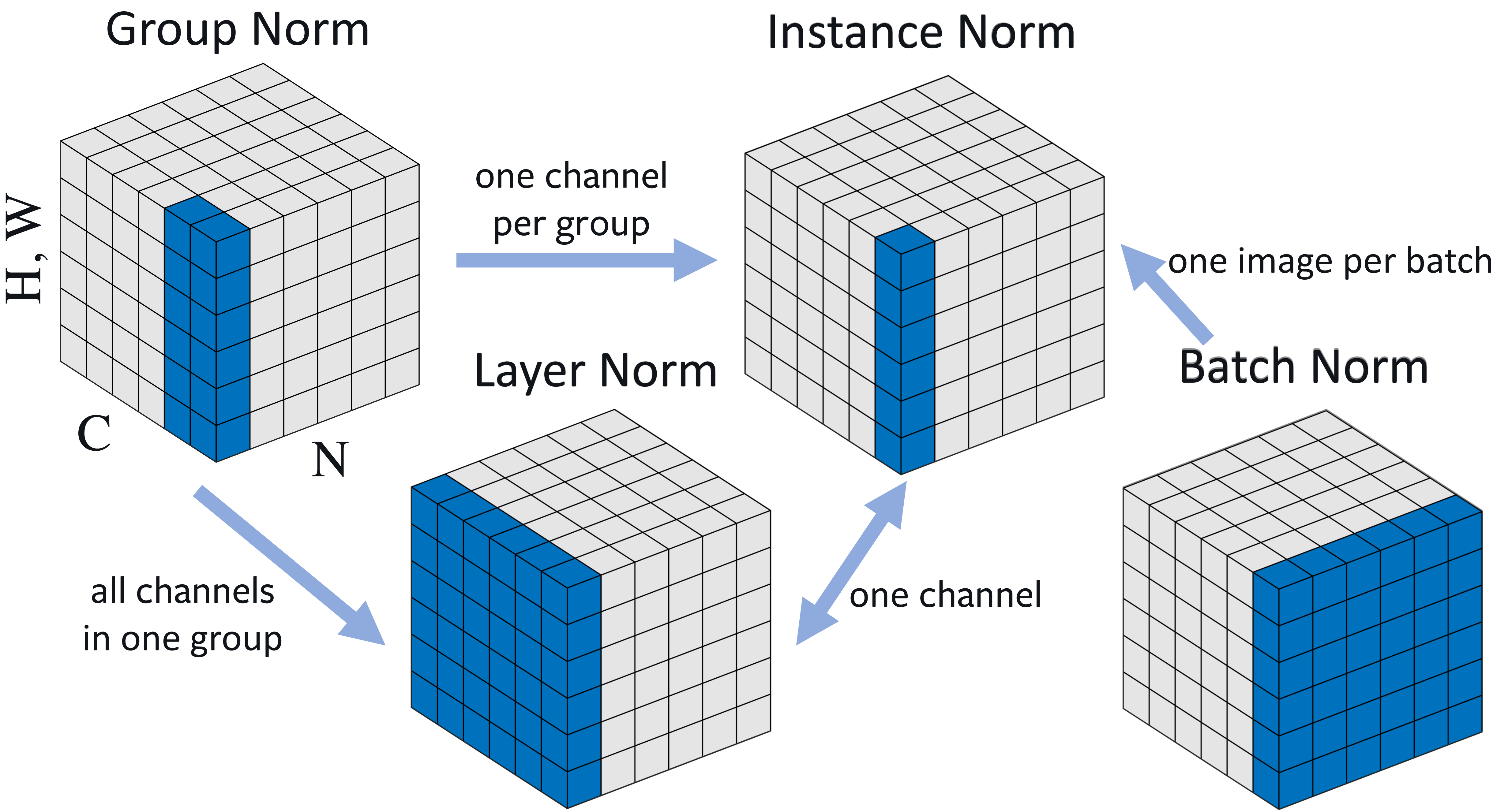
What's Group Norm



$$\hat{x} = \frac{x - \mu}{\sigma}$$

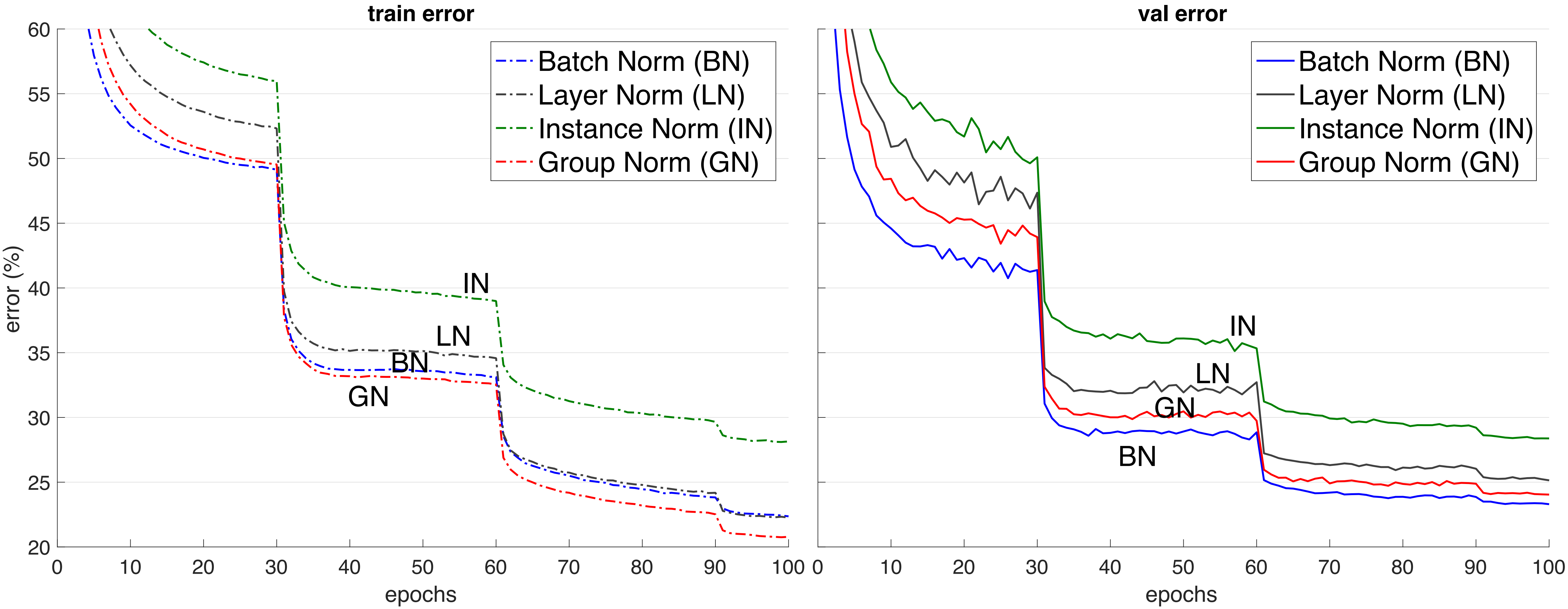
Test time:
do the same thing

Yuxin Wu and Kaiming He. "Group Normalization" *ECCV 2018*



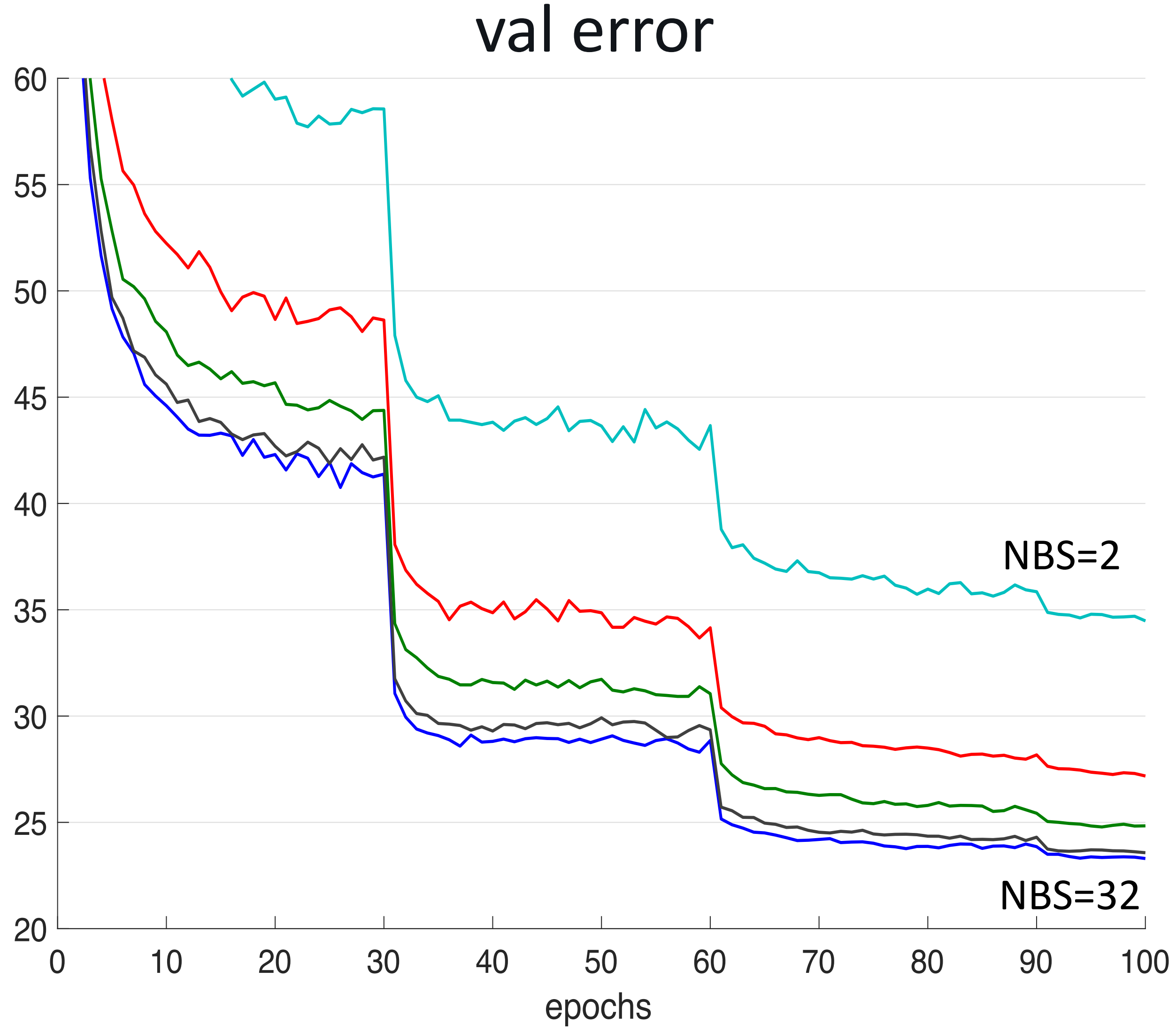
Ulyanov, Dmitry, Andrea Vedaldi, and Victor S. Lempitsky. "Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis." *CVPR* 2017.
Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." *arXiv preprint arXiv:1607.06450* (2016).

GroupNorm Fits Training Set Better
BatchNorm Has Regularization

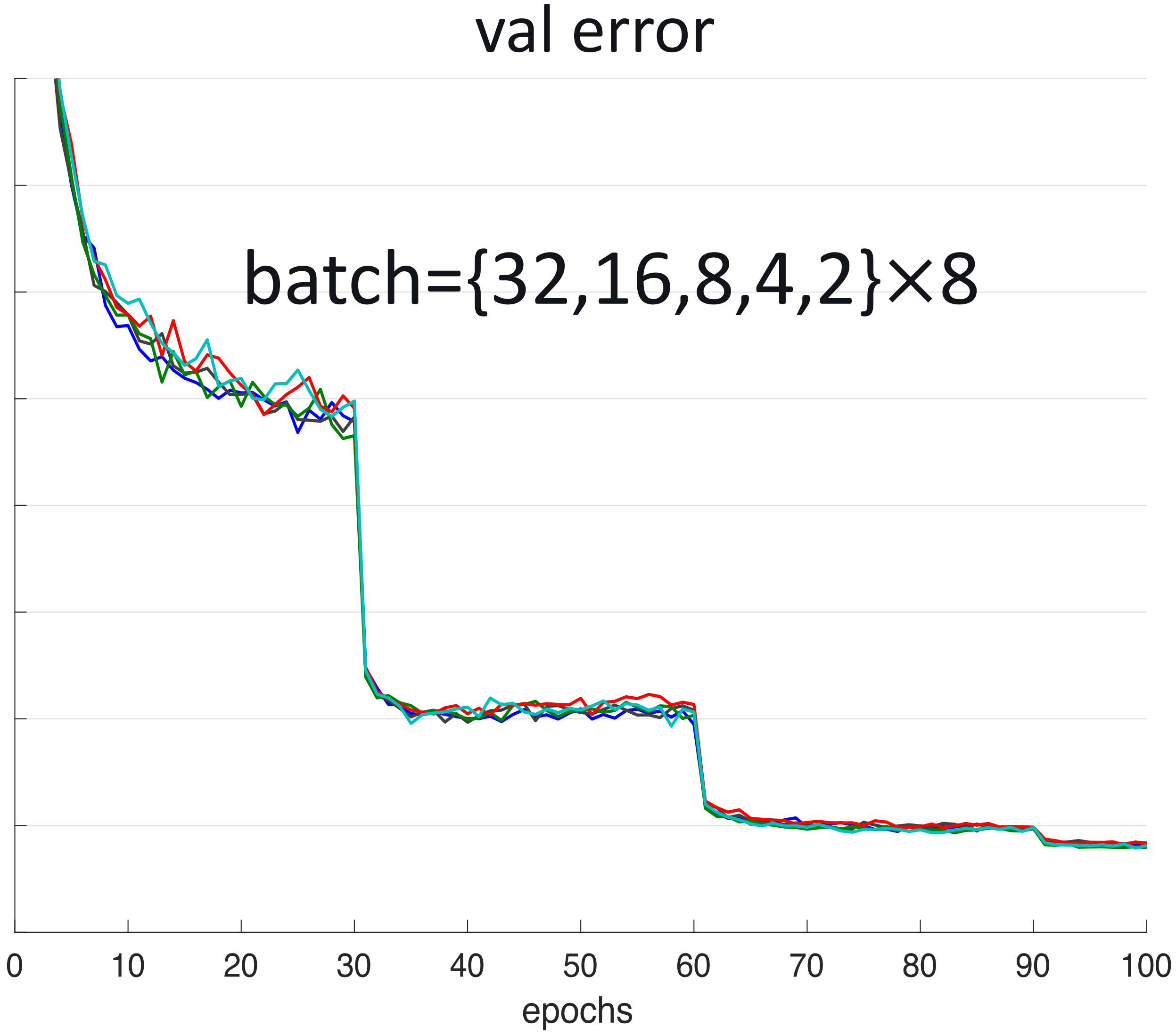


Yuxin Wu and Kaiming He. "Group Normalization" *ECCV 2018*

Small Batch Size



Batch Norm ☹

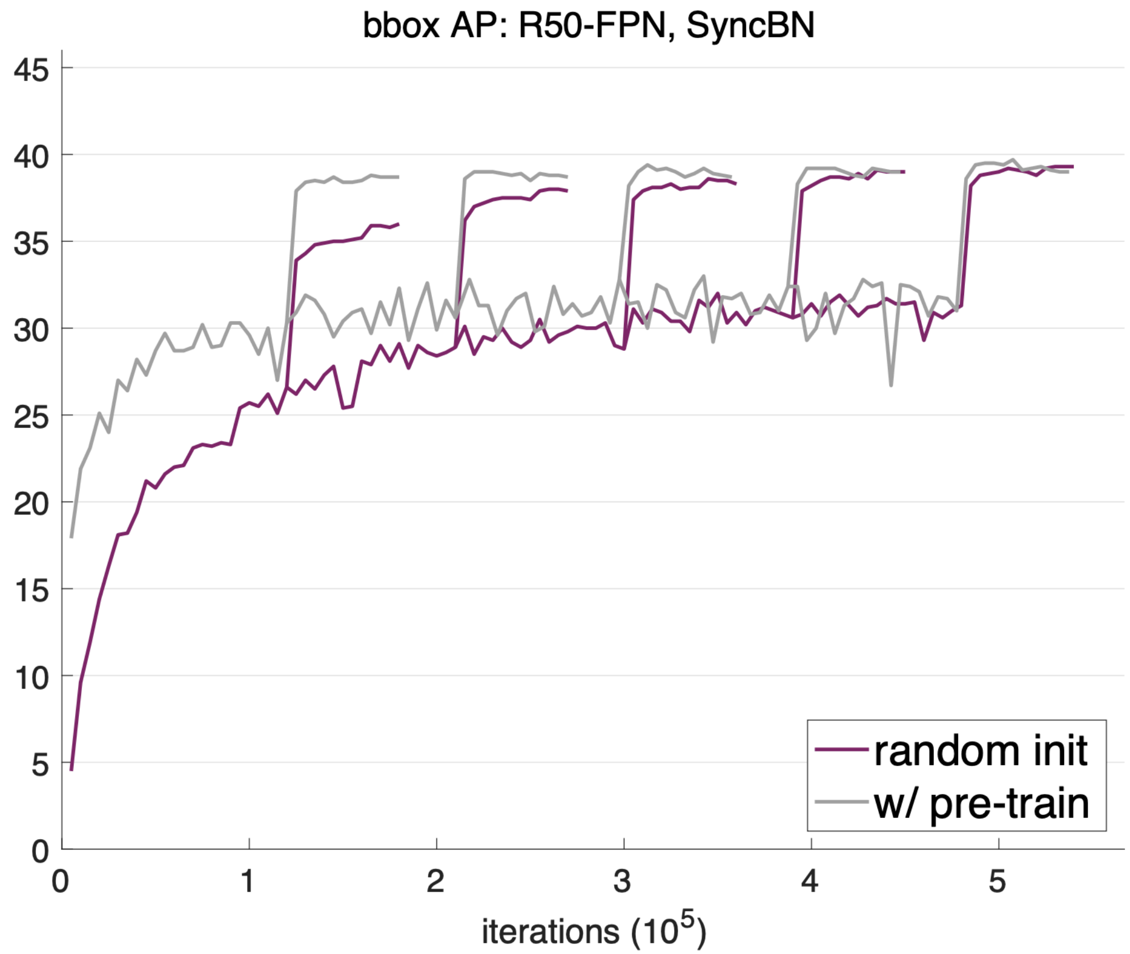


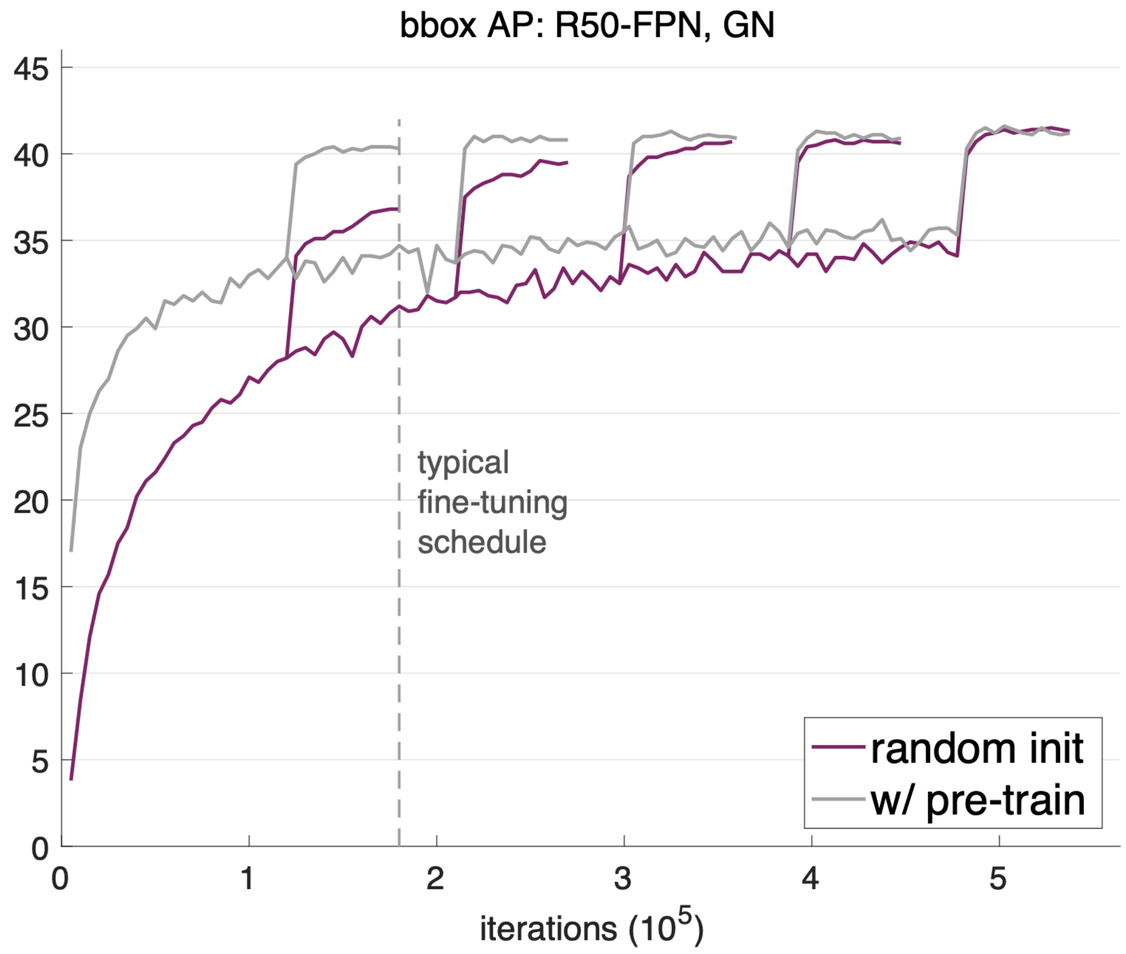
Group Norm ☺

curves match

Yuxin Wu and Kaiming He. "Group Normalization" *ECCV 2018*

R-CNN From Scratch: ~~FrozenBN,~~ SyncBN, GN





Kaiming He, et. al., "Rethinking ImageNet Pre-training" *ICCV 2019*

Other Normalizations

- L1 Normalization (L1BN, etc)
- Local Response Normalization (LRN)
- (Centered) Weight Normalization (WN, CWN, WS)
- No Normalization (Fixup)