# Adasum: Adaptive Summation of Gradients for Deep Learning

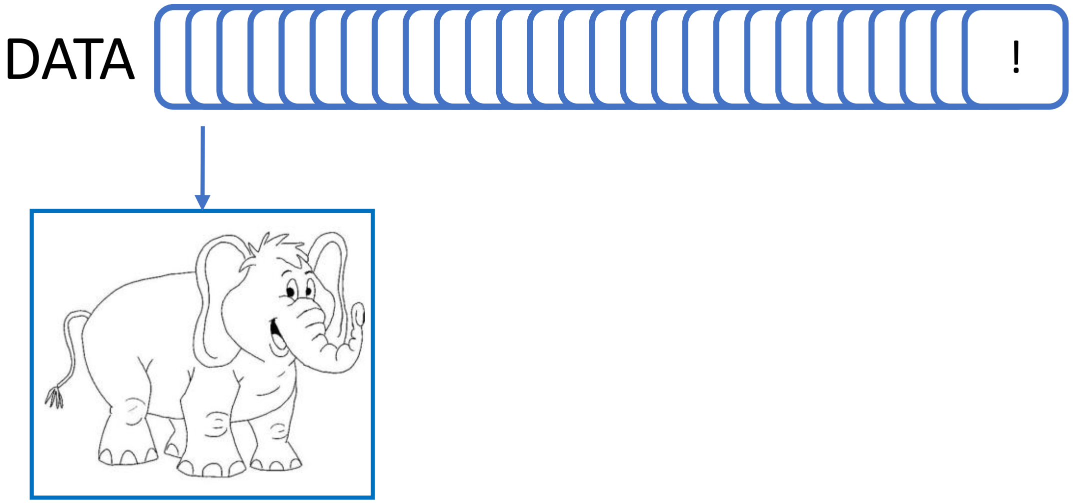
**Saeed Maleki**, Madan Musuvathi, Todd Mytkowicz, Olli Saarikivi

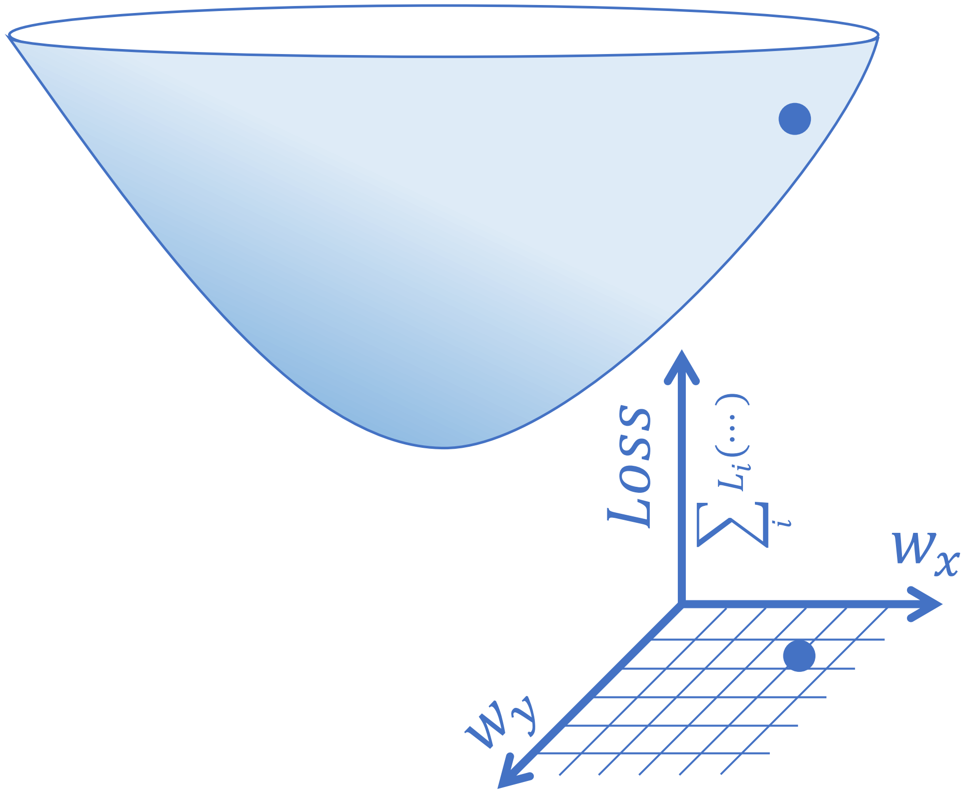
Microsoft Research

Emad Barsoum, Sergii Dymchenko, Jaliya Ekanayake, Vadim Eksarevskiy,
Maxim Lukiyanov, Tianju Xu
Microsoft

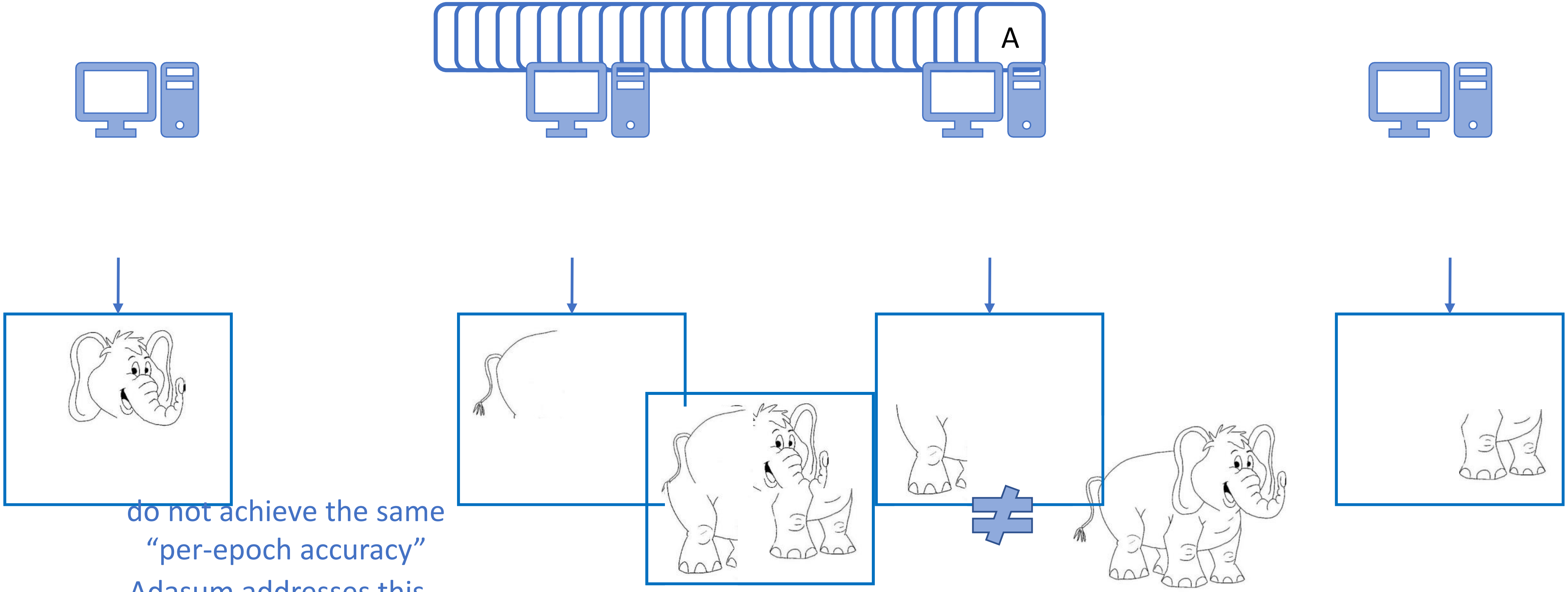
# Motivation: Neural Network Training is Inherently Sequential

- Stochastic Gradient Descent (SGD)
  - Workhorse for training a neural network
  - Inherently sequential



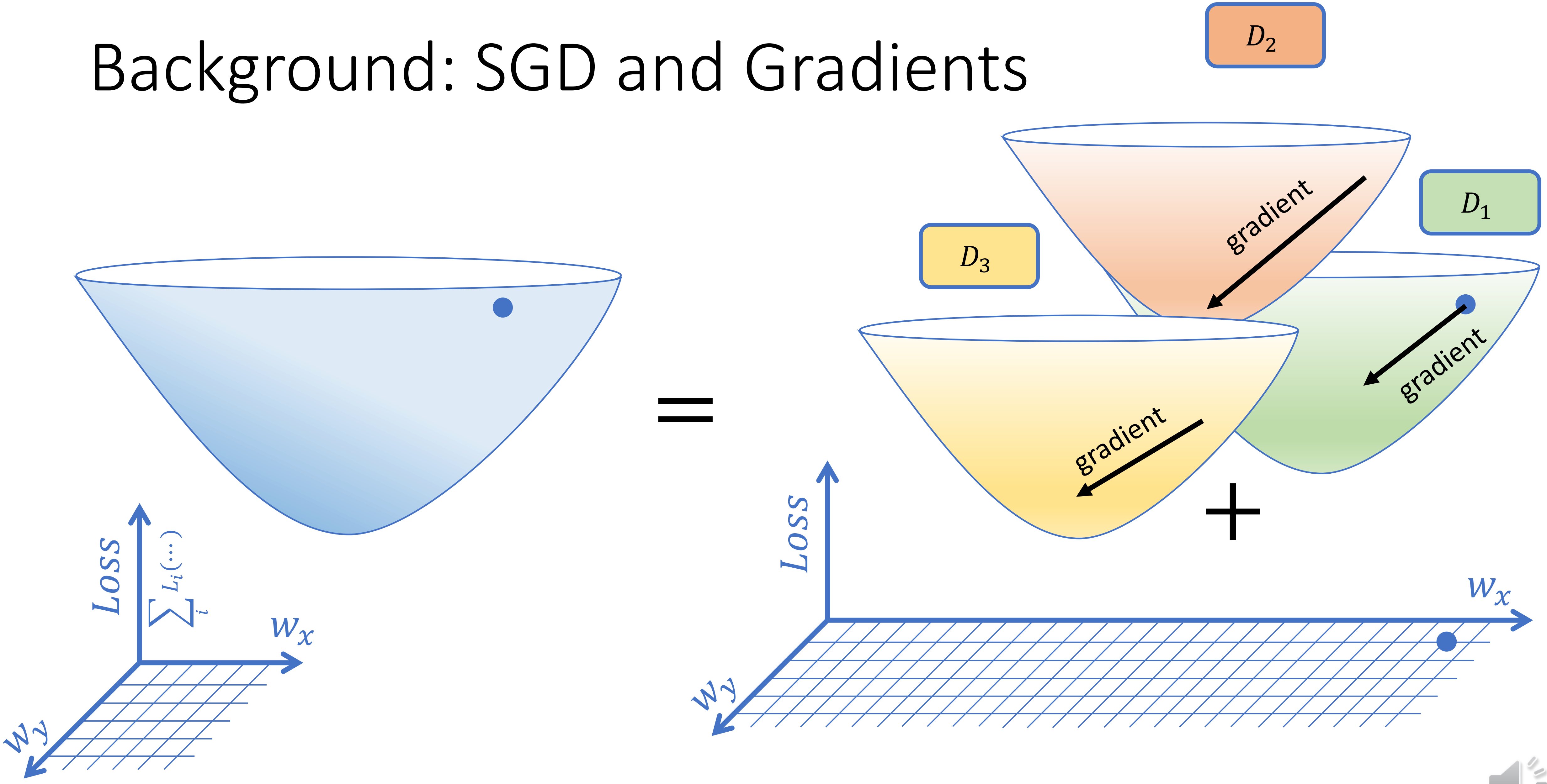



# Motivation: More parallelism = less accuracy

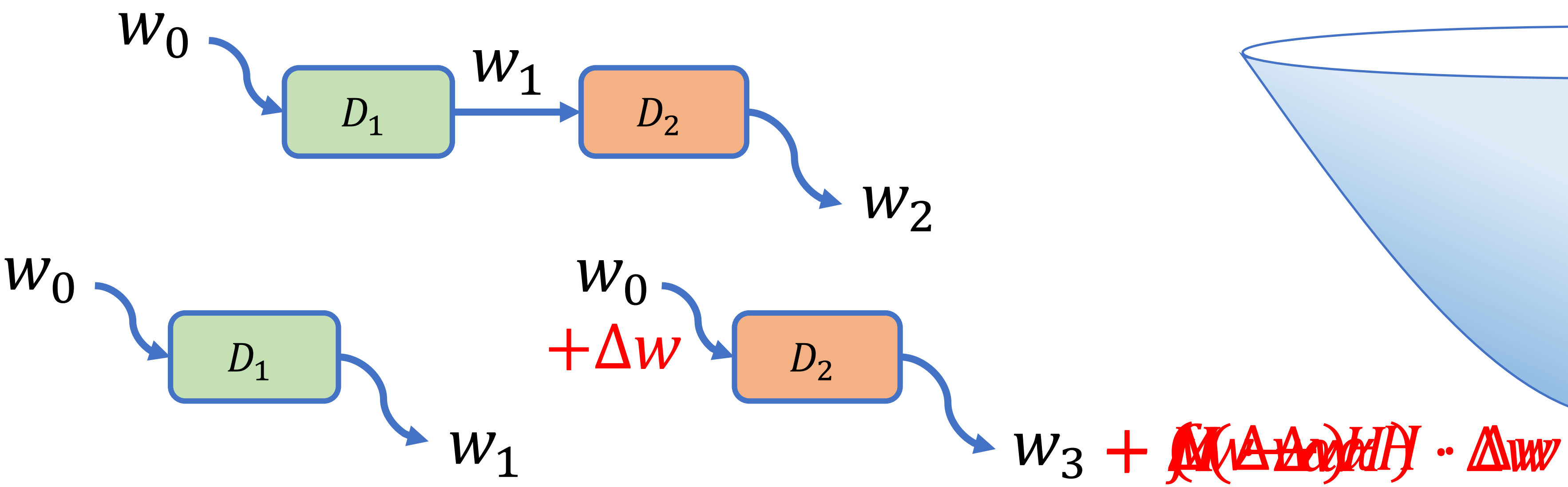


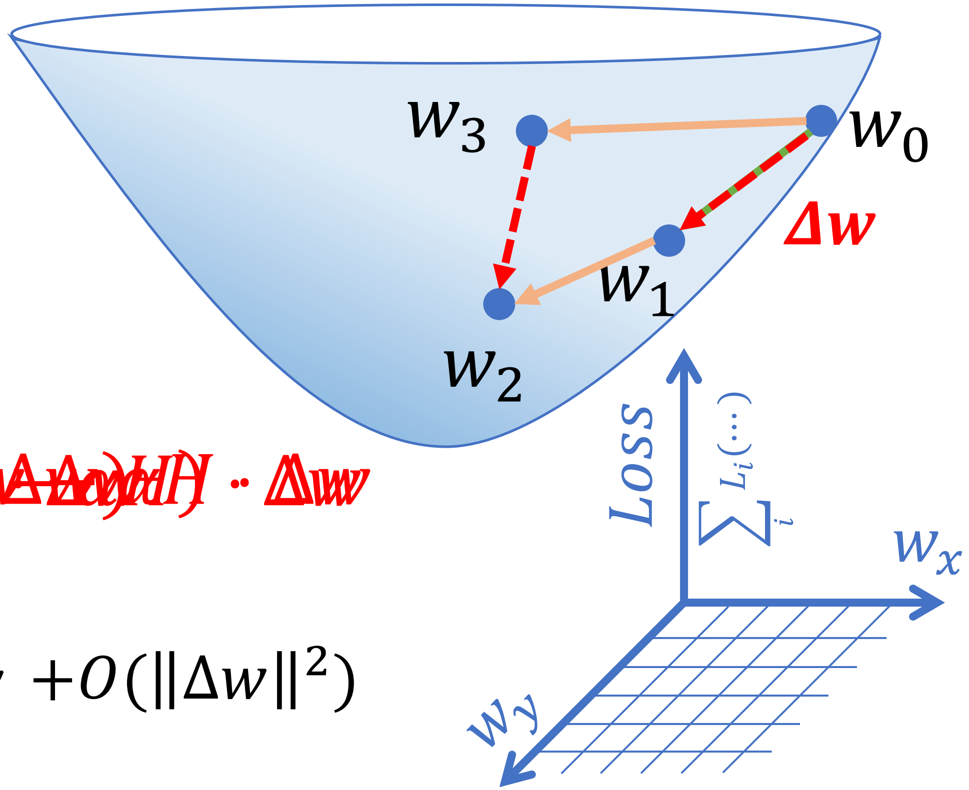

do not achieve the same
"per-epoch accuracy"
Adasum addresses this

# Background: SGD and Gradients



# Adasum: Symbolic SGD

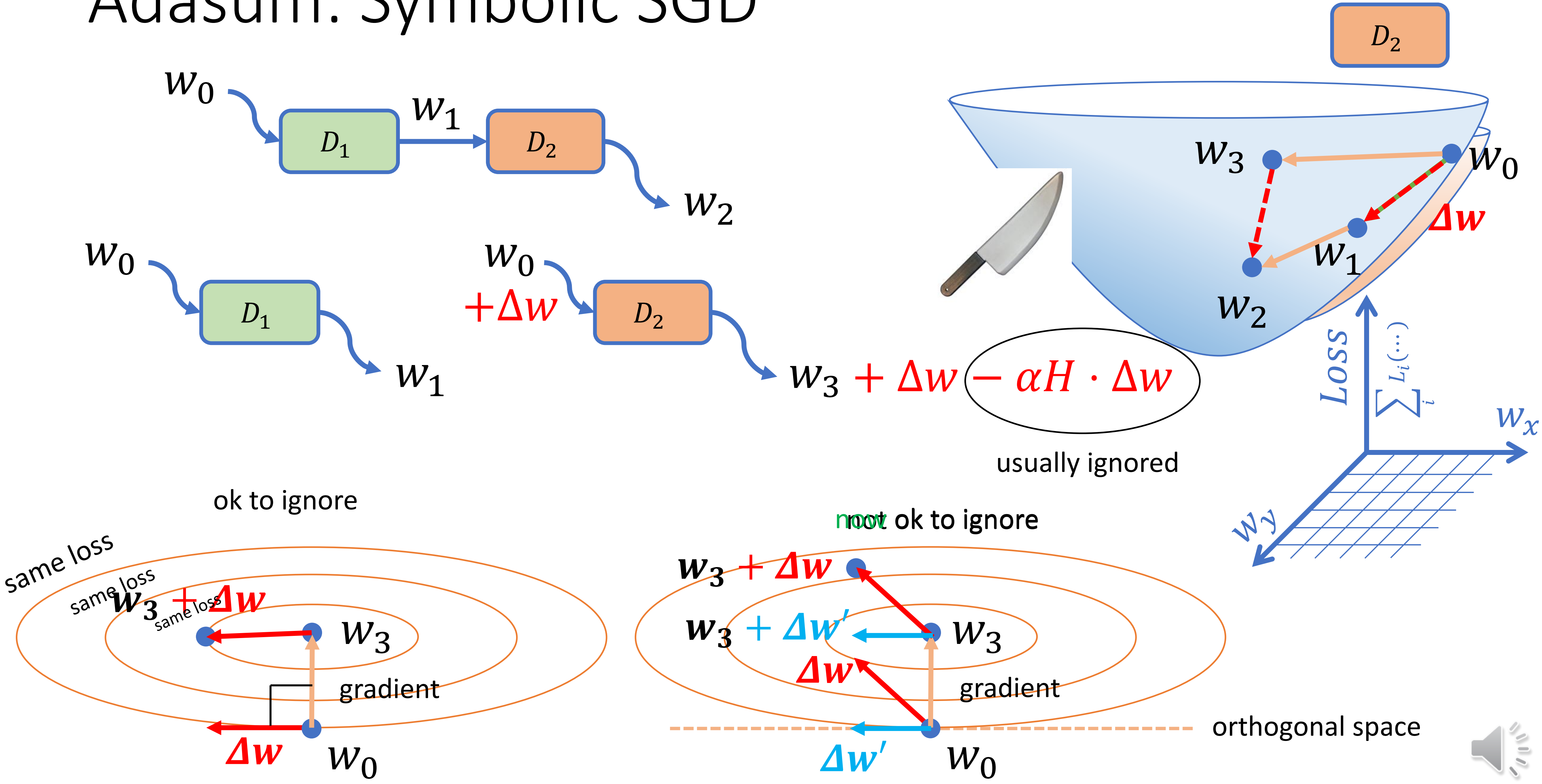



$$SGD(w_0 + \Delta w) = \overbrace{SGD(w_0)}^{\text{local model } w_3} + \overbrace{SGD'(w_0)}^{\text{a matrix } M} \cdot \Delta w + O(\|\Delta w\|^2)$$

- $M = I - \alpha H$ where $H$ is the Hessian matrix and $\alpha$ is the learning rate
- Very costly to calculate $H$

# Adasum: Symbolic SGD

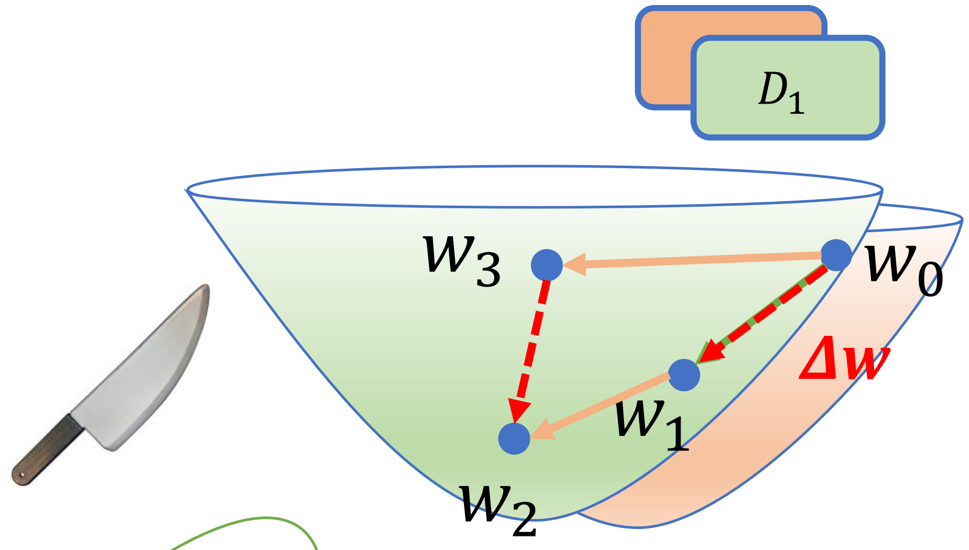


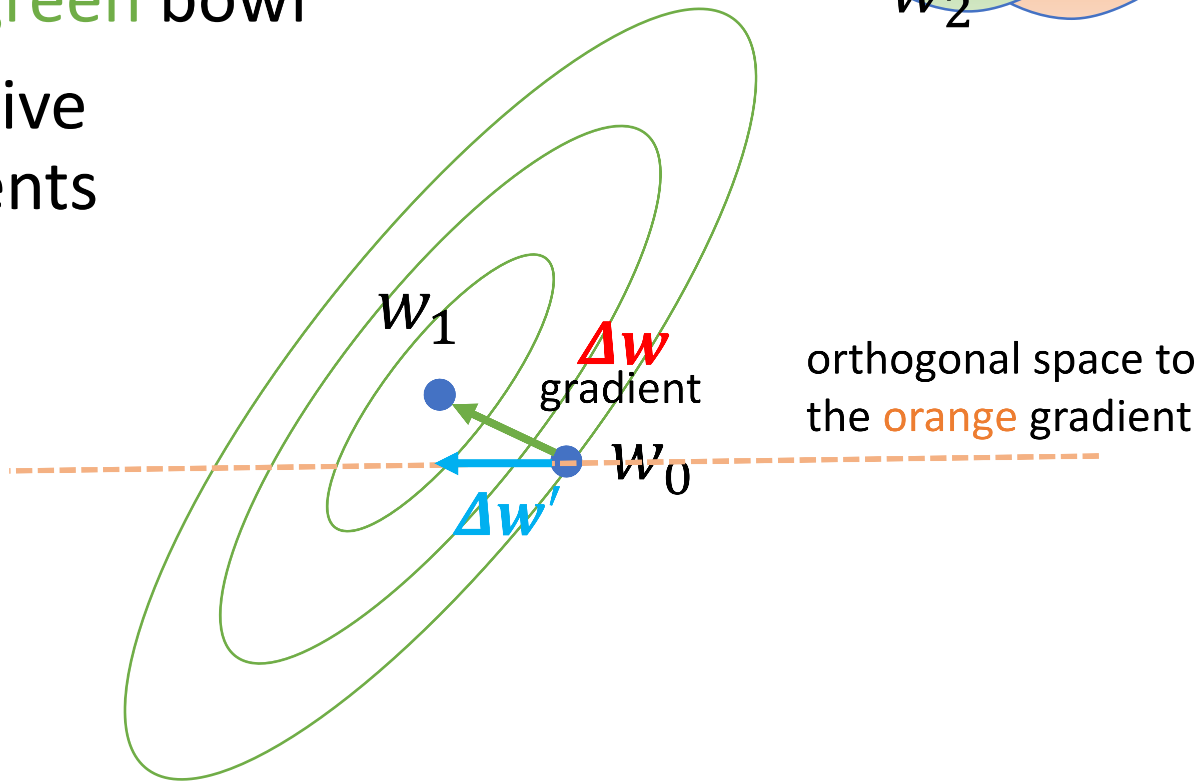
# $\Delta w'$ is a "good" direction

- Is $\Delta w'$ a "good" direction to move along?

- $\Delta w$ is the gradient w.r.t. the green bowl
- Any direction that has a positive inner product with the gradients decays the loss
  - $\Delta w'$ is a "good" direction

- Adasum operator sums $\Delta w$ with projection



# Adasum: Adaptive Sum

- Adasum combines $\Delta w$ from any number of processors

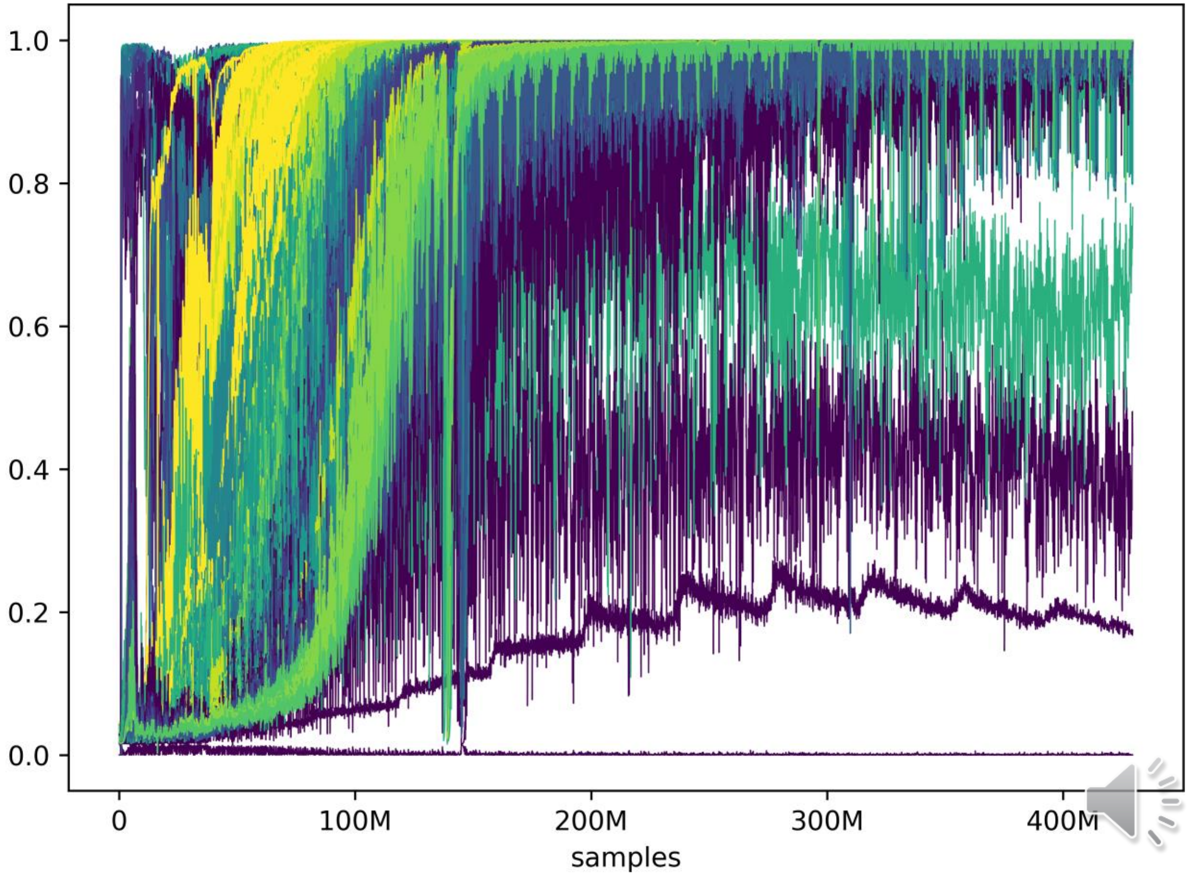
- Adasum combines $\Delta w$ from different processors by projection and summation. Effectively:
  - They are added when they are orthogonal
  - Only one is taken when they are parallel

- Traditionally, $\Delta w$ from different processors are:
  - Either summed: can be too aggressive
  - Or Averaged: can be too conservative

# Orthogonality of Gradients

- We use Pythagorean theorem to define orthogonality
- For P gradients it ranges between:
  - 1 for all orthogonal
  - 1/P for all parallel
- Gradients start out all parallel
- Later in the training they become more orthogonal
- Convergence starts out slow but speeds up later

BERT with 64 GPUs

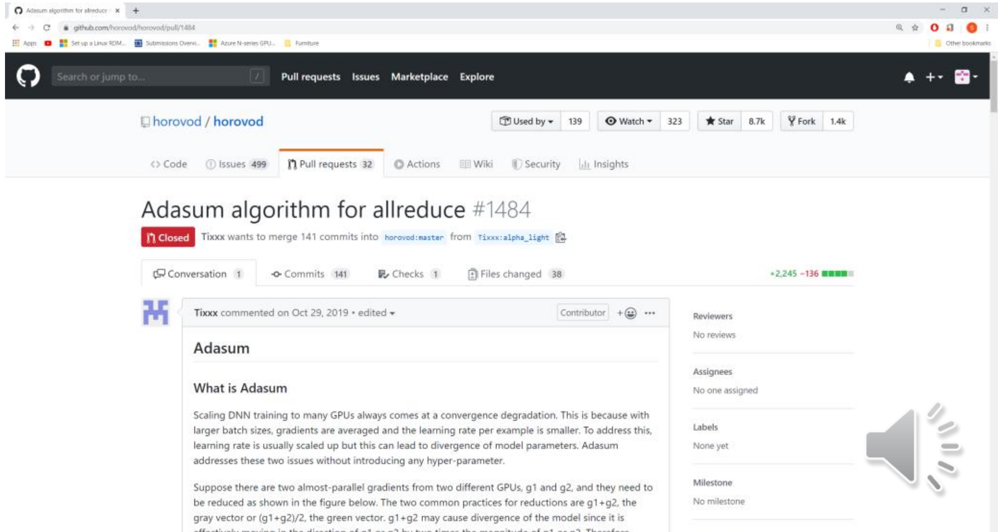


# Adasum is in Horovod

- Horovod is an open-source distributed training framework by Uber that supports both PyTorch and TensorFlow
  - Adasum is integrated in Horovod

- Adasum is easy to use:
  - `horovod.allreduce(gradients, op=hvd.adasum)`
  - No hyperparameter

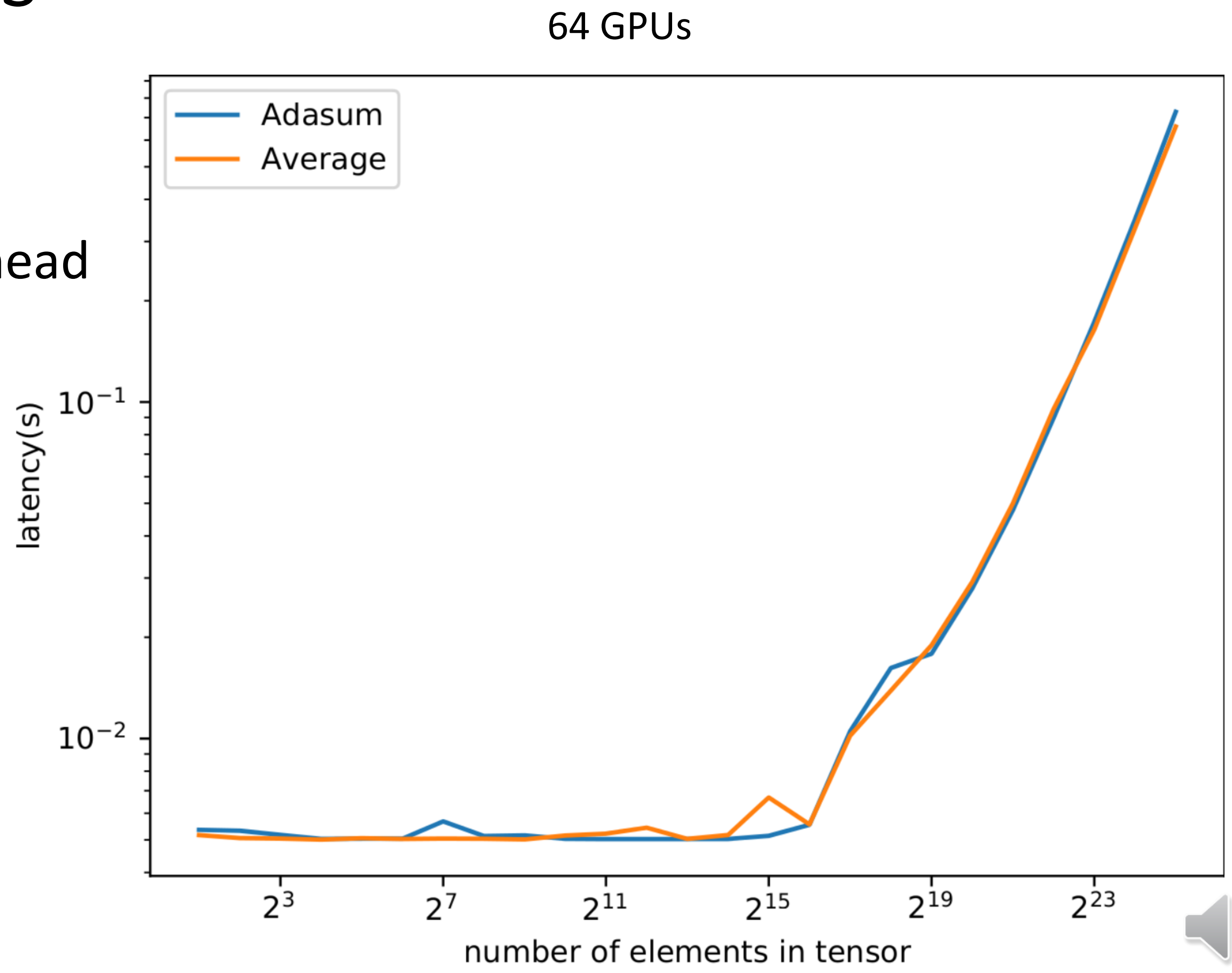
- Adasum allows scaling SGD
  - Minimizes convergence slowdown in scale



# Result – Adasum vs. Allreduce/Averaging Latency on 64 GPUs

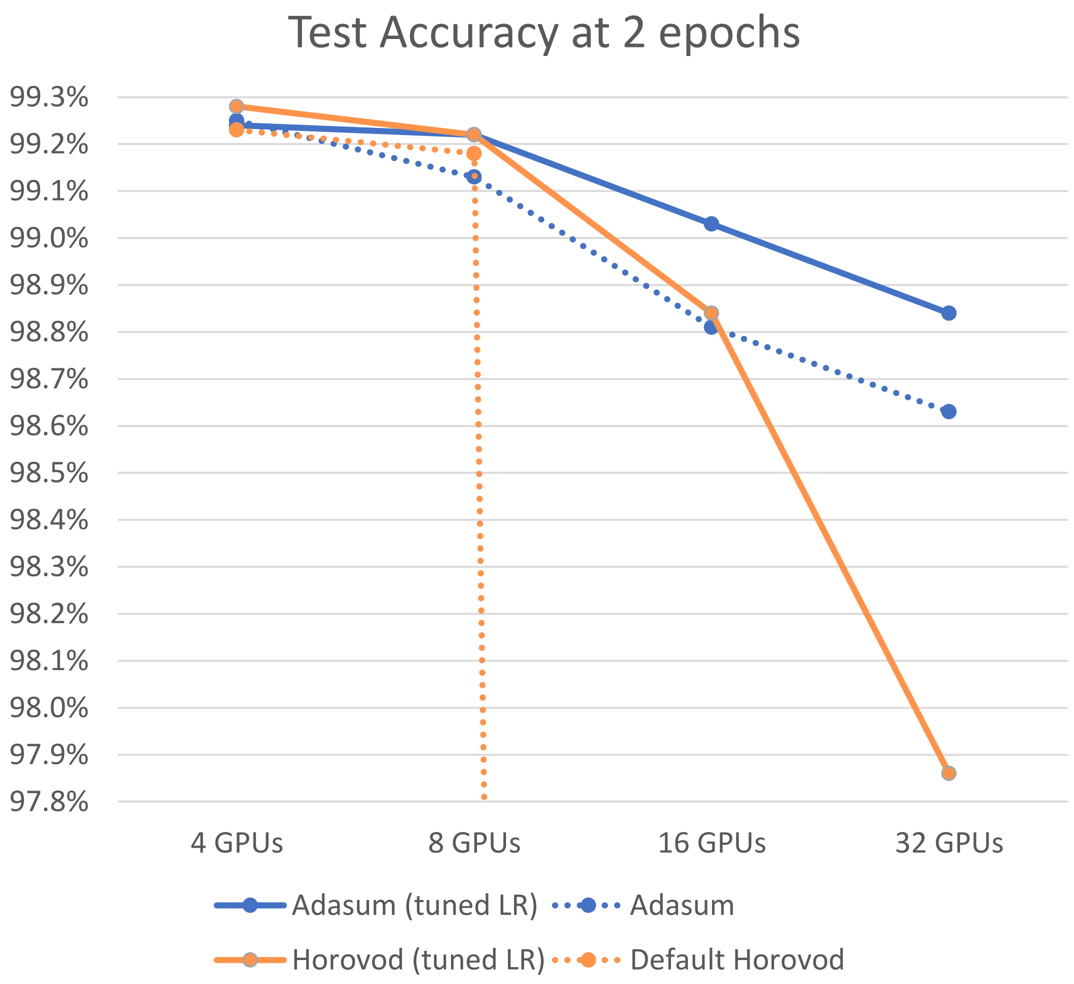
- Adasum has some computation overhead
  - But no communication overhead

- Almost negligible overhead
  - Communication latency dominates the computation latency



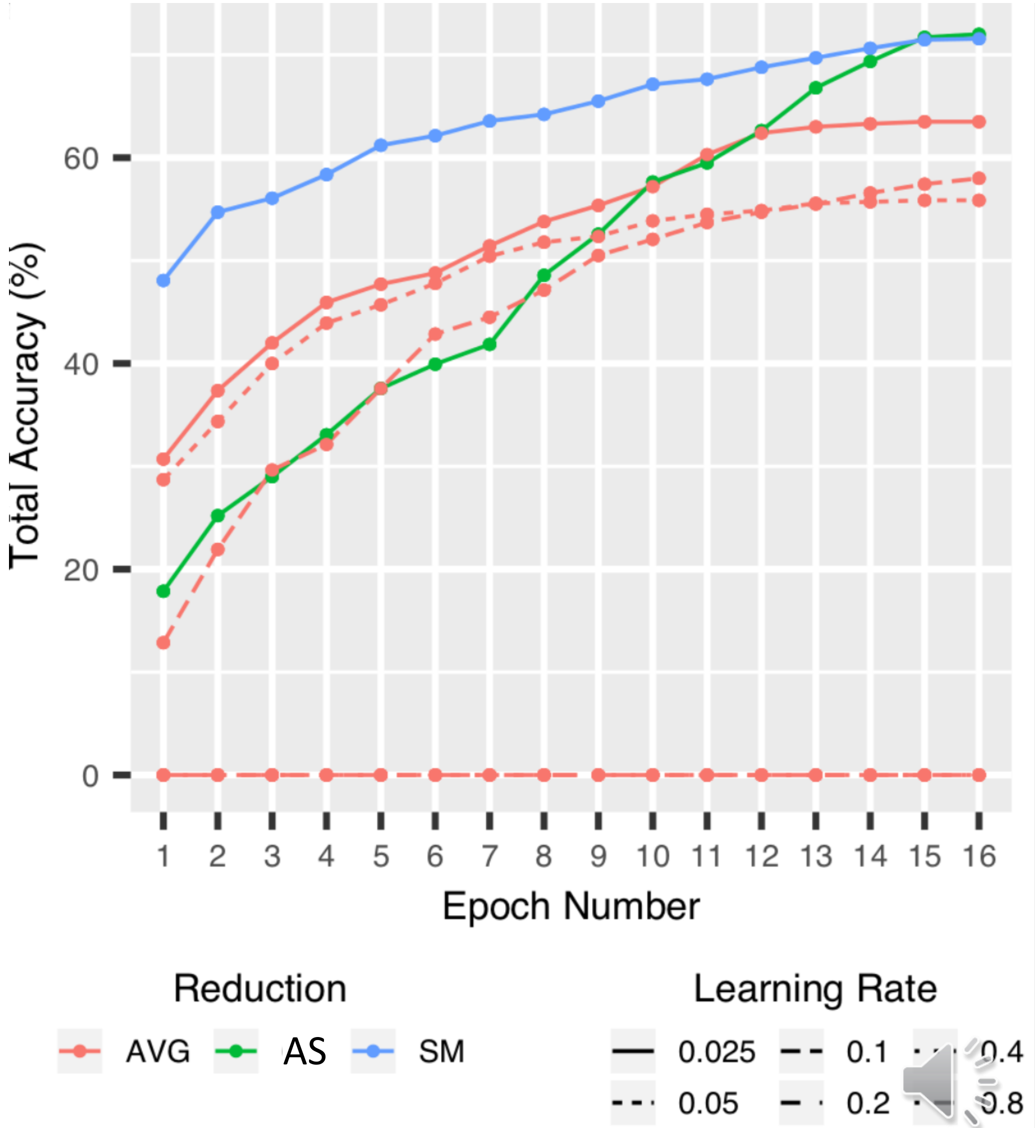
# Results – Adasum Convergence on MNIST

- Standard 2-layer CNN gets 99.3% in 2 epochs sequentially with batch size 32
- Tuned LR for averaging and Adasum
- Default Horovod fails at 16 GPUs, while Adasum still works with 32
- With tuned learning rates Adasum has much better convergence for 16 and 32 GPUs



# Results – Adasum Convergence on word2vec

- word2vec model: embedding for words
  - London to England::Paris to France
- Runs on 32 nodes
- Tuned LR for averaging
- Adasum matches sequential accuracy

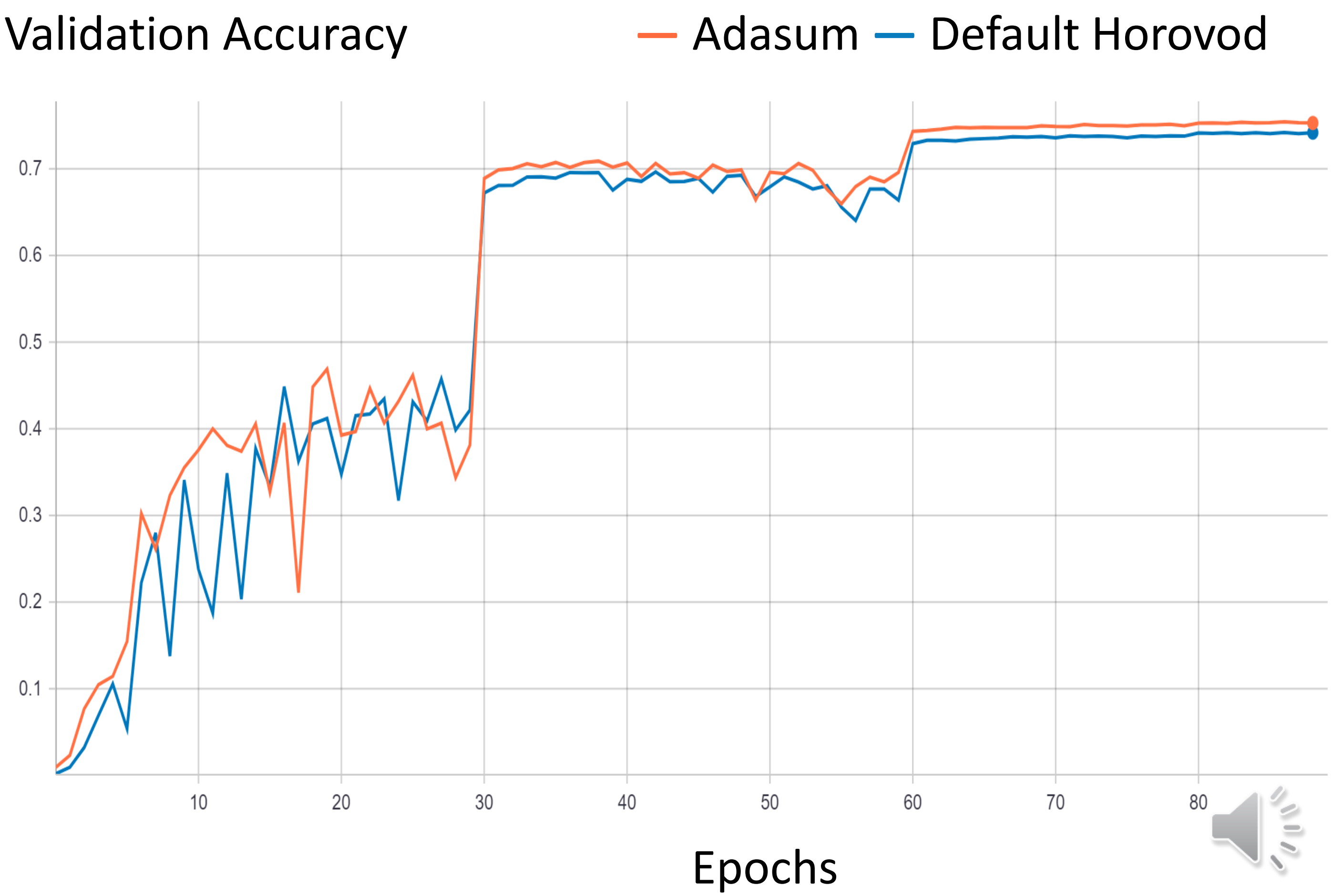


# Results – Adasum Convergence on Resnet

- The MLPerf Resnet50 on ImageNet
  - 16K batch size
  - 64 GPUs

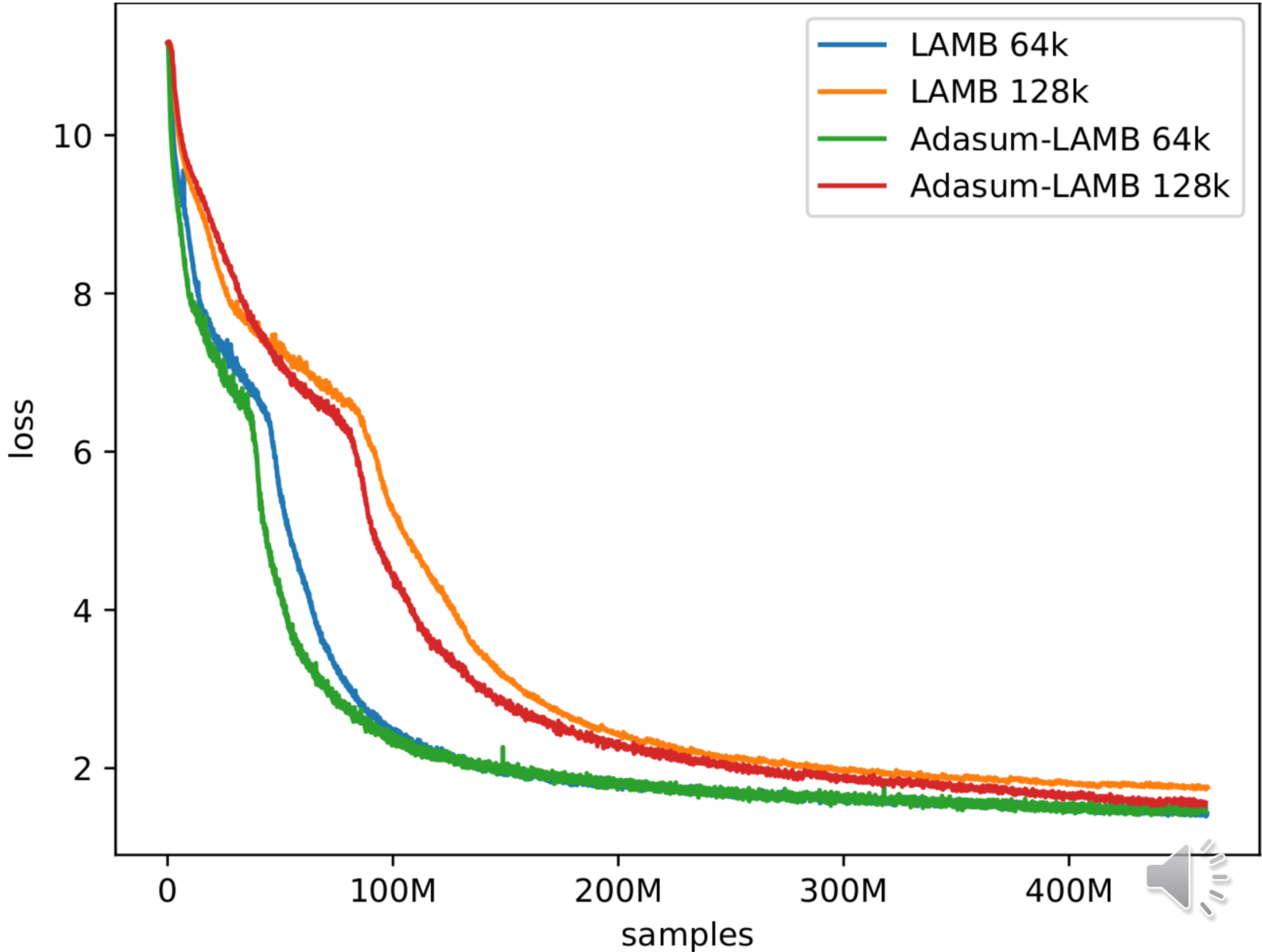
- Adasum reaches MLPerf target accuracy 74.9% in 69 epochs

- Default Horovod never reaches target accuracy in 90 epochs



# Results – Adasum Convergence on BERT

- BERT is a common model trained nowadays
- Bigger batch size = more parallelism
  - 64k and 128k
- LAMB is the optimizer used at large scales
- Adasum beats LAMB with 128k batchsize



# Please use Adasum!

- And let us know what you think!

- https://github.com/horovod/