Autodiff and Optimizers

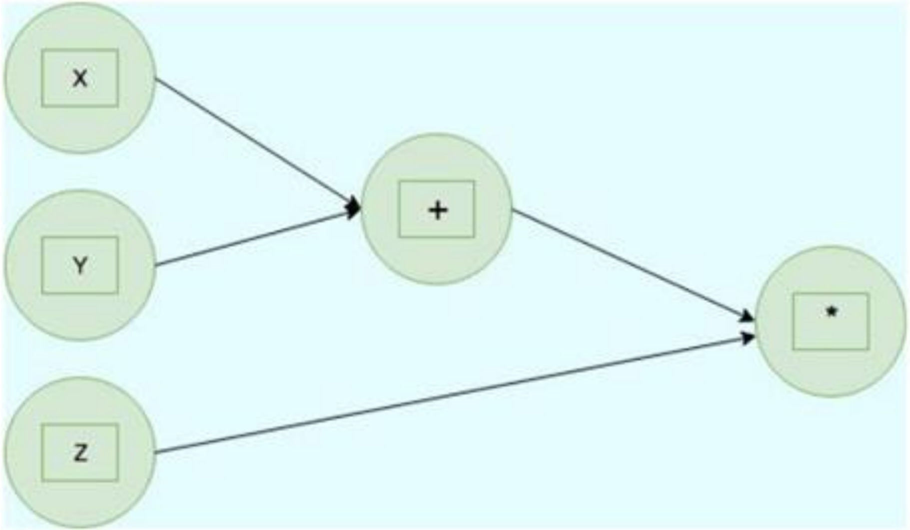
Prace2022 HPML

Autodiff

(a.k.a. Computational graphs)

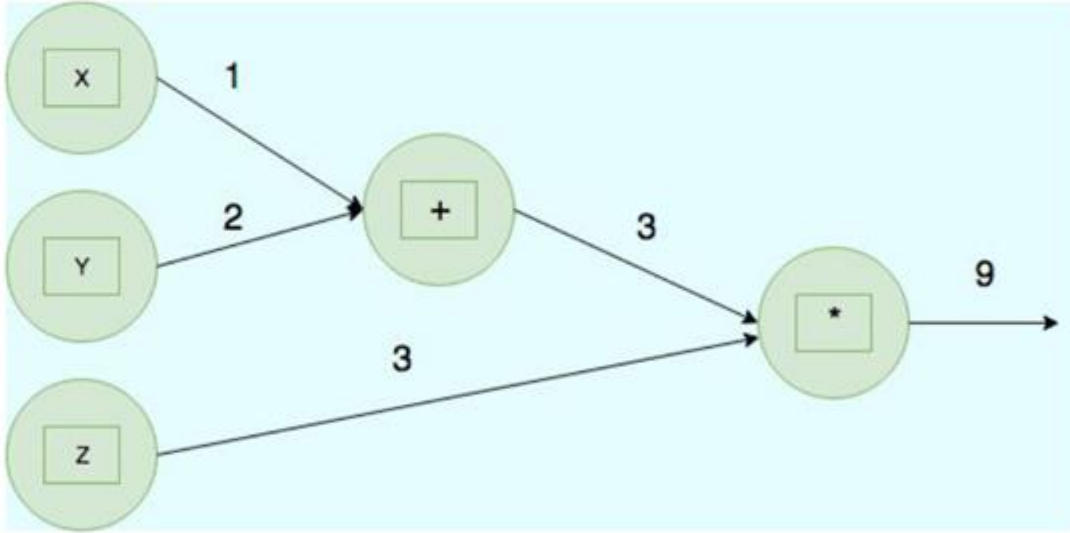
Computational Graphs

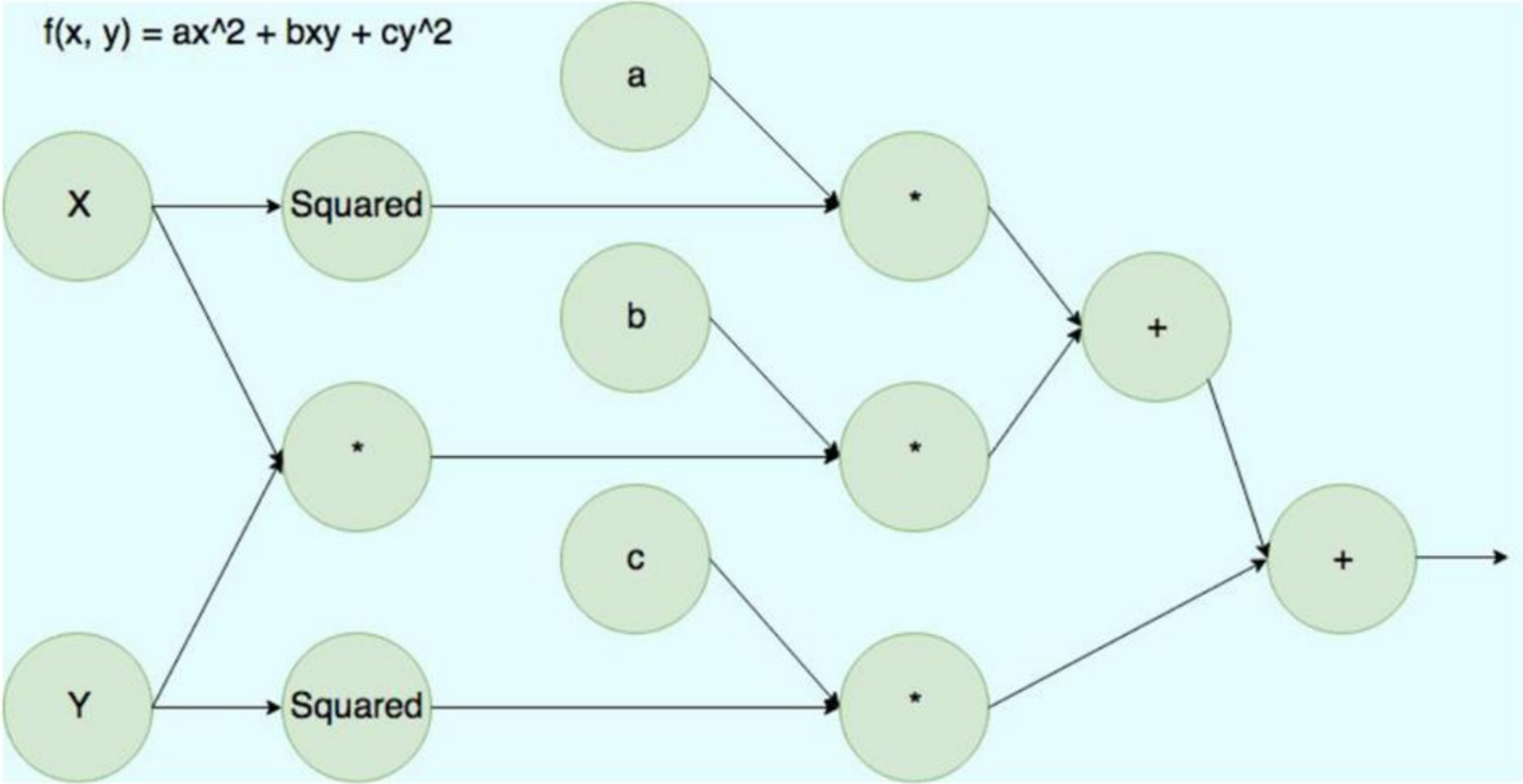
- Represents math in graph format
- Nodes and edges

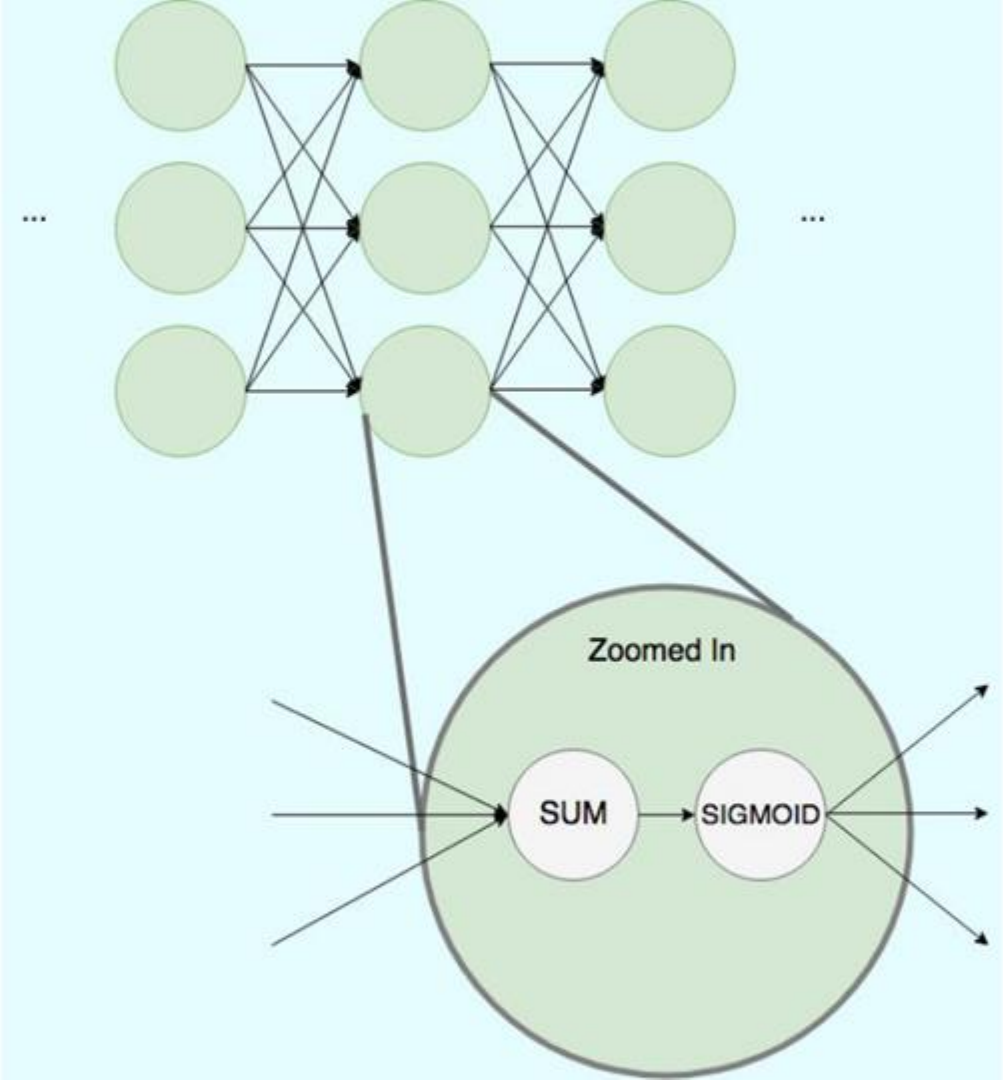


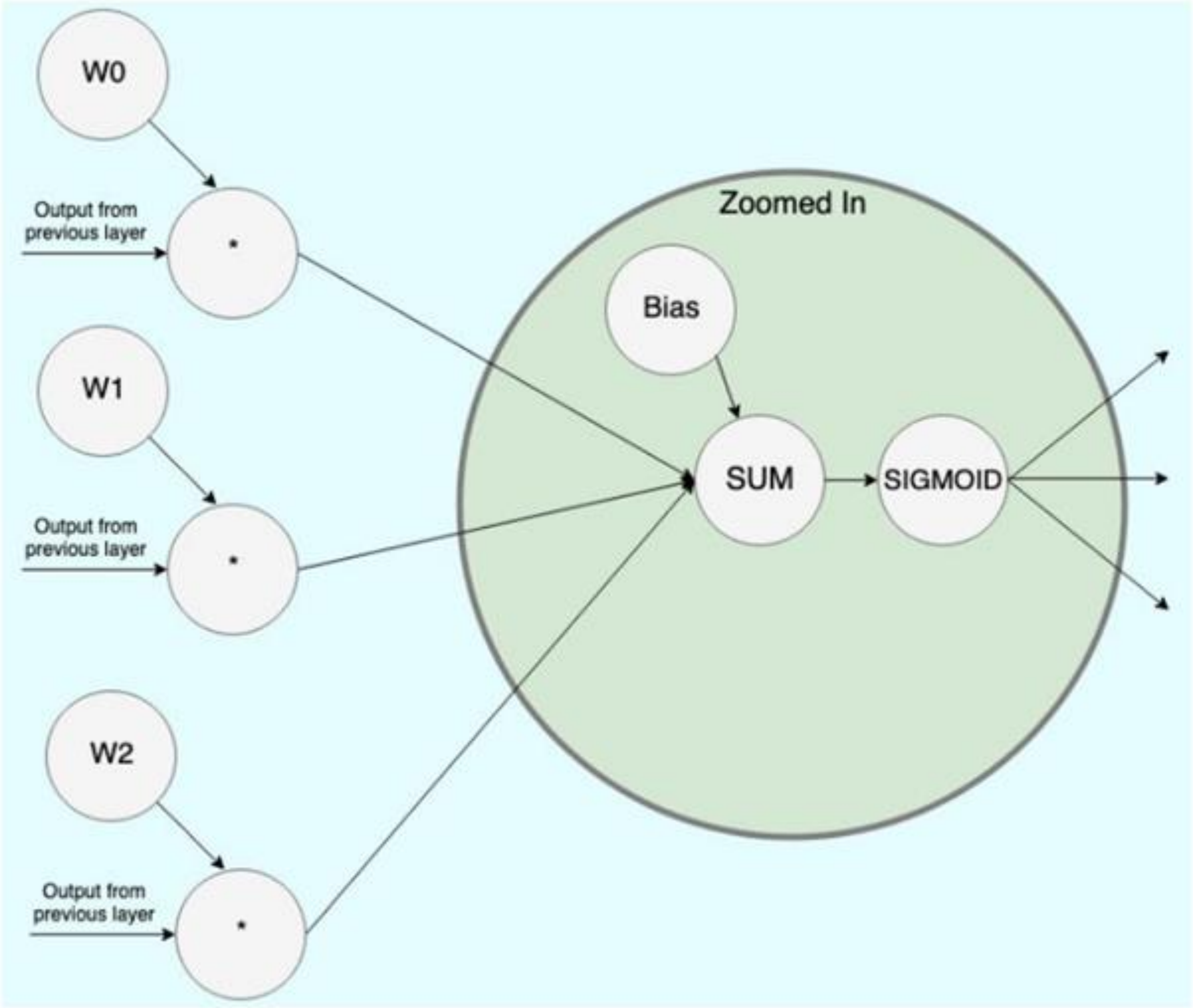
Computational Graphs

- Represents math in graph format
- Nodes and edges



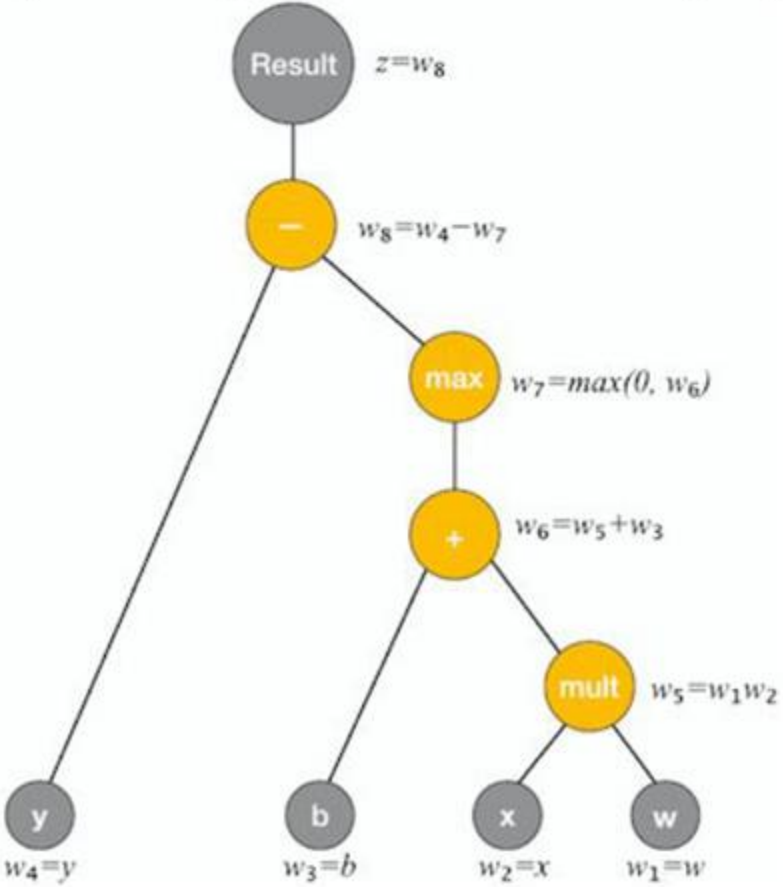

f(x, y) = ax^2 + bxy + cy^2
a
X
Squared
*
b
+
*
c
+
Y
Squared
*


...
...
Zoomed In
SUM
SIGMOID

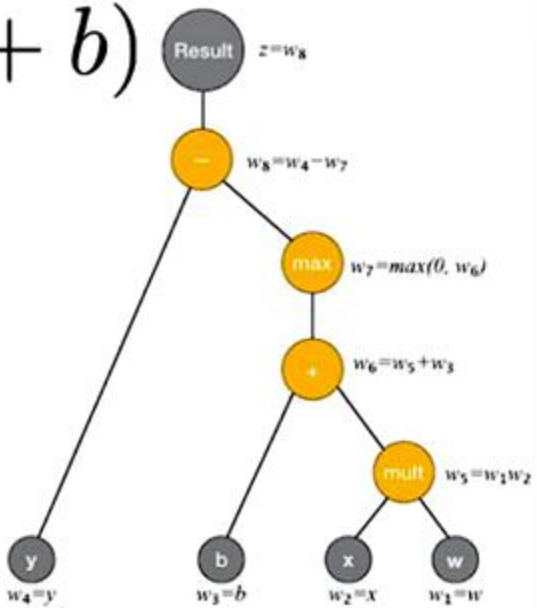

W0
Output from previous layer
*
W1
Output from previous layer
*
W2
Output from previous layer
*
Zoomed In
Bias
SUM
SIGMOID

$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$

$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$



$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$

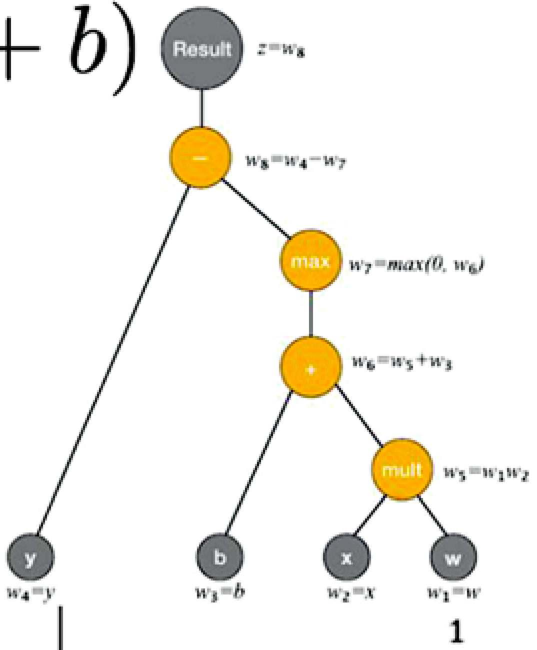


1

$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1}$$

$\frac{\delta w_5}{\delta w_1} = \frac{\delta w_1 w_2}{\delta w_1} = w_2$	$\frac{\delta w_5}{\delta w_2} = \frac{\delta w_1 w_2}{\delta w_2} = w_1$
$\frac{\delta w_6}{\delta w_5} = \frac{\delta w_5 + w_3}{\delta w_5} = 1$	$\frac{\delta w_6}{\delta w_3} = \frac{\delta w_5 + w_3}{\delta w_3} = 1$
$\frac{\delta w_7}{\delta w_6} = \frac{\delta \max(0, w_6)}{\delta w_6} = \begin{cases} 0, & w_6 < 0 \\ 1, & w_6 > 0 \end{cases}$	$\frac{\delta w_8}{\delta w_7} = \frac{\delta w_4 - w_7}{\delta w_7} = -1$
$\frac{\delta w_8}{\delta w_4} = \frac{\delta w_4 - w_7}{\delta w_4} = 1$	$\frac{\delta z}{\delta w_8} = 1$

$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$

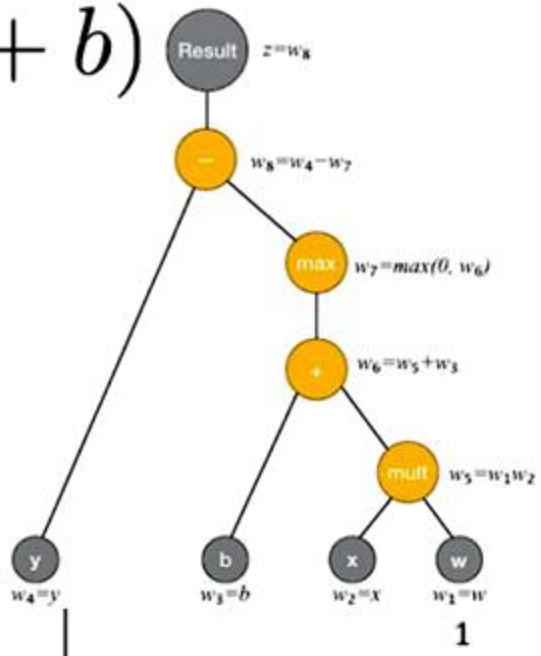


$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1}$$

$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1} = 1 \times (-1) \times \begin{cases} 0, & w_2 < 0 \\ 1, & w_2 > 0 \end{cases} \times 1 \times w_2 = \begin{cases} 0, & w_2 < 0 \\ -w_2, & w_2 > 0 \end{cases}$$

$\frac{\delta w_5}{\delta w_1} = \frac{\delta w_1 w_2}{\delta w_1} = w_2$	$\frac{\delta w_5}{\delta w_2} = \frac{\delta w_1 w_2}{\delta w_2} = w_1$
$\frac{\delta w_6}{\delta w_5} = \frac{\delta w_5 + w_3}{\delta w_5} = 1$	$\frac{\delta w_6}{\delta w_3} = \frac{\delta w_5 + w_3}{\delta w_3} = 1$
$\frac{\delta w_7}{\delta w_6} = \frac{\delta \max(0, w_6)}{\delta w_6} = \begin{cases} 0, & w_6 < 0 \\ 1, & w_6 > 0 \end{cases}$	$\frac{\delta w_8}{\delta w_7} = \frac{\delta w_4 - w_7}{\delta w_7} = -1$
$\frac{\delta w_8}{\delta w_4} = \frac{\delta w_4 - w_7}{\delta w_4} = 1$	$\frac{\delta z}{\delta w_8} = 1$

$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$

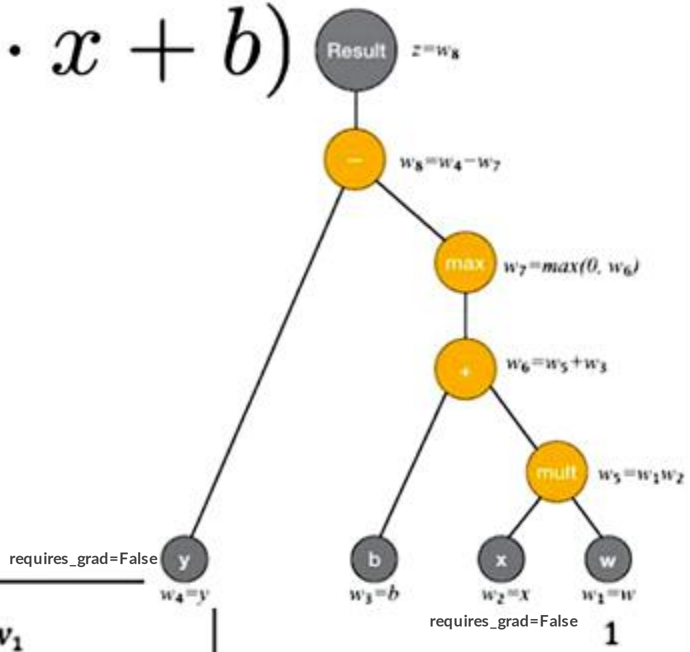


$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1}$$

What do we need to save in the forward pass?

$\frac{\delta w_5}{\delta w_1} = \frac{\delta w_1 w_2}{\delta w_1} = w_2$	$\frac{\delta w_5}{\delta w_2} = \frac{\delta w_1 w_2}{\delta w_2} = w_1$
$\frac{\delta w_6}{\delta w_5} = \frac{\delta w_5 + w_3}{\delta w_5} = 1$	$\frac{\delta w_6}{\delta w_3} = \frac{\delta w_5 + w_3}{\delta w_3} = 1$
$\frac{\delta w_7}{\delta w_6} = \frac{\delta \max(0, w_6)}{\delta w_6} = \begin{cases} 0, & w_6 < 0 \\ 1, & w_6 > 0 \end{cases}$	$\frac{\delta w_8}{\delta w_7} = \frac{\delta w_4 - w_7}{\delta w_7} = -1$
$\frac{\delta w_8}{\delta w_4} = \frac{\delta w_4 - w_7}{\delta w_4} = 1$	$\frac{\delta z}{\delta w_8} = 1$

$$C(y, w, x, b) = y - max(0, w \cdot x + b)$$



$$\frac{\delta z}{\delta w_1} = \frac{\delta z}{\delta w_8} \times \frac{\delta w_8}{\delta w_7} \times \frac{\delta w_7}{\delta w_6} \times \frac{\delta w_6}{\delta w_5} \times \frac{\delta w_5}{\delta w_1}$$

What do we need to save in the forward pass?

$\frac{\delta w_5}{\delta w_1} = \frac{\delta w_1 w_2}{\delta w_1} = w_2$	$\frac{\delta w_5}{\delta w_2} = \frac{\delta w_1 w_2}{\delta w_2} = w_1$
$\frac{\delta w_6}{\delta w_5} = \frac{\delta w_5 + w_3}{\delta w_5} = 1$	$\frac{\delta w_6}{\delta w_3} = \frac{\delta w_5 + w_3}{\delta w_3} = 1$
$\frac{\delta w_7}{\delta w_6} = \frac{\delta \max(0, w_6)}{\delta w_6} = \begin{cases} 0, & w_6 < 0 \\ 1, & w_6 > 0 \end{cases}$	$\frac{\delta w_8}{\delta w_7} = \frac{\delta w_4 - w_7}{\delta w_7} = -1$
$\frac{\delta w_8}{\delta w_4} = \frac{\delta w_4 - w_7}{\delta w_4} = 1$	$\frac{\delta z}{\delta w_8} = 1$

PyTorch workflow

1. Reset buffers (i.e. w.grad=0)
2. Create computational graph & store activations
3. Calculate actual gradients (i.e. w.grad=dL/dw)
4. Update parameters

```
optimizer.zero_grad()
output = model(data)
loss = F.nll_loss(output, target)
loss.backward()
optimizer.step()
```

Leveraging internals for our gain

```
model.zero_grad()                                   # Reset gradients tensors
for i, (inputs, labels) in enumerate(training_set):
    predictions = model(inputs)                     # Forward pass
    loss = loss_function(predictions, labels)       # Compute loss function
    loss = loss / accumulation_steps                # Normalize our loss (if averaged)
    loss.backward()                                 # Backward pass
    if (i+1) % accumulation_steps == 0:             # Wait for several backward steps
        optimizer.step()                            # Now we can do an optimizer step
        model.zero_grad()                           # Reset gradients tensors
```

Optimizers

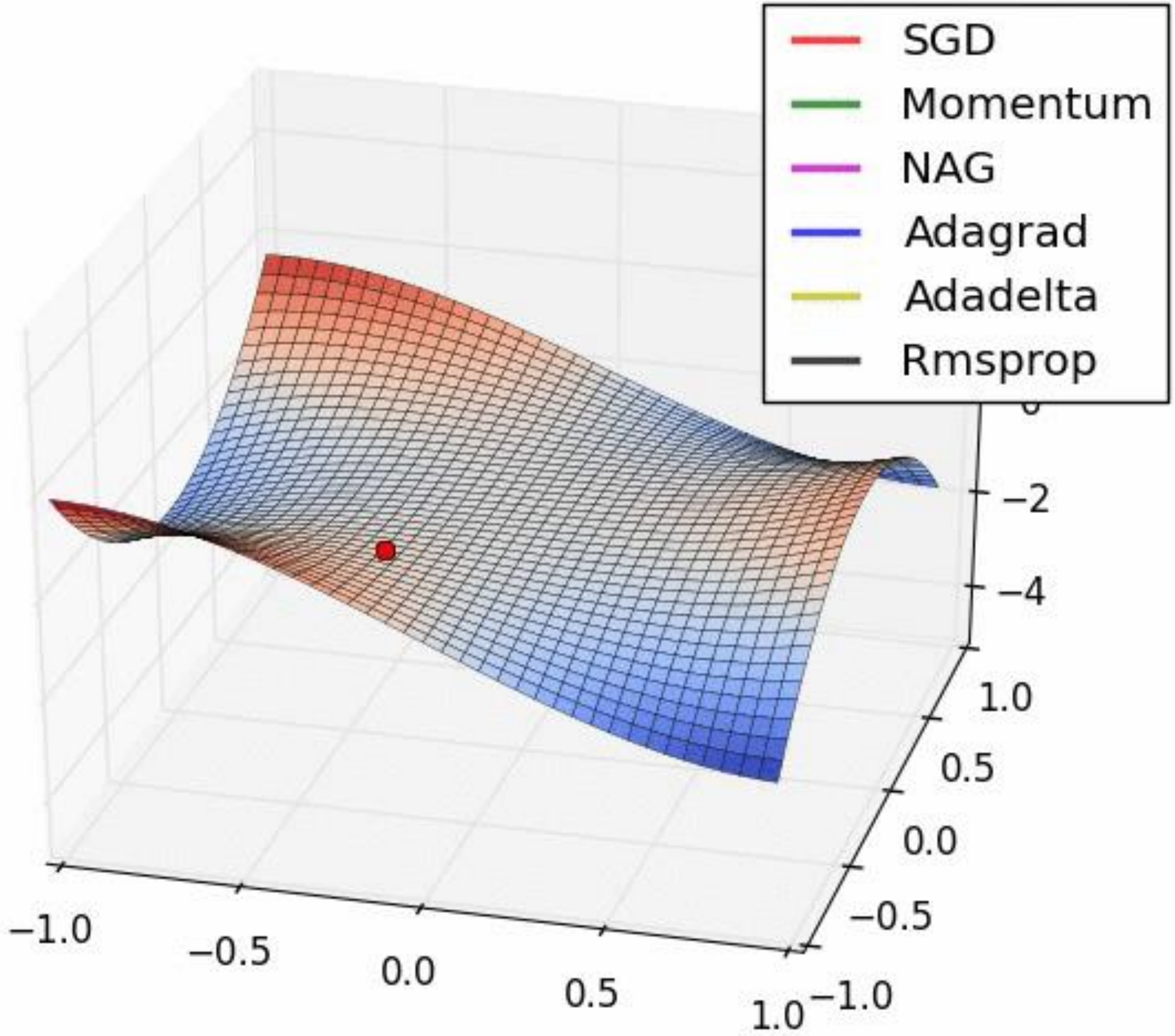
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

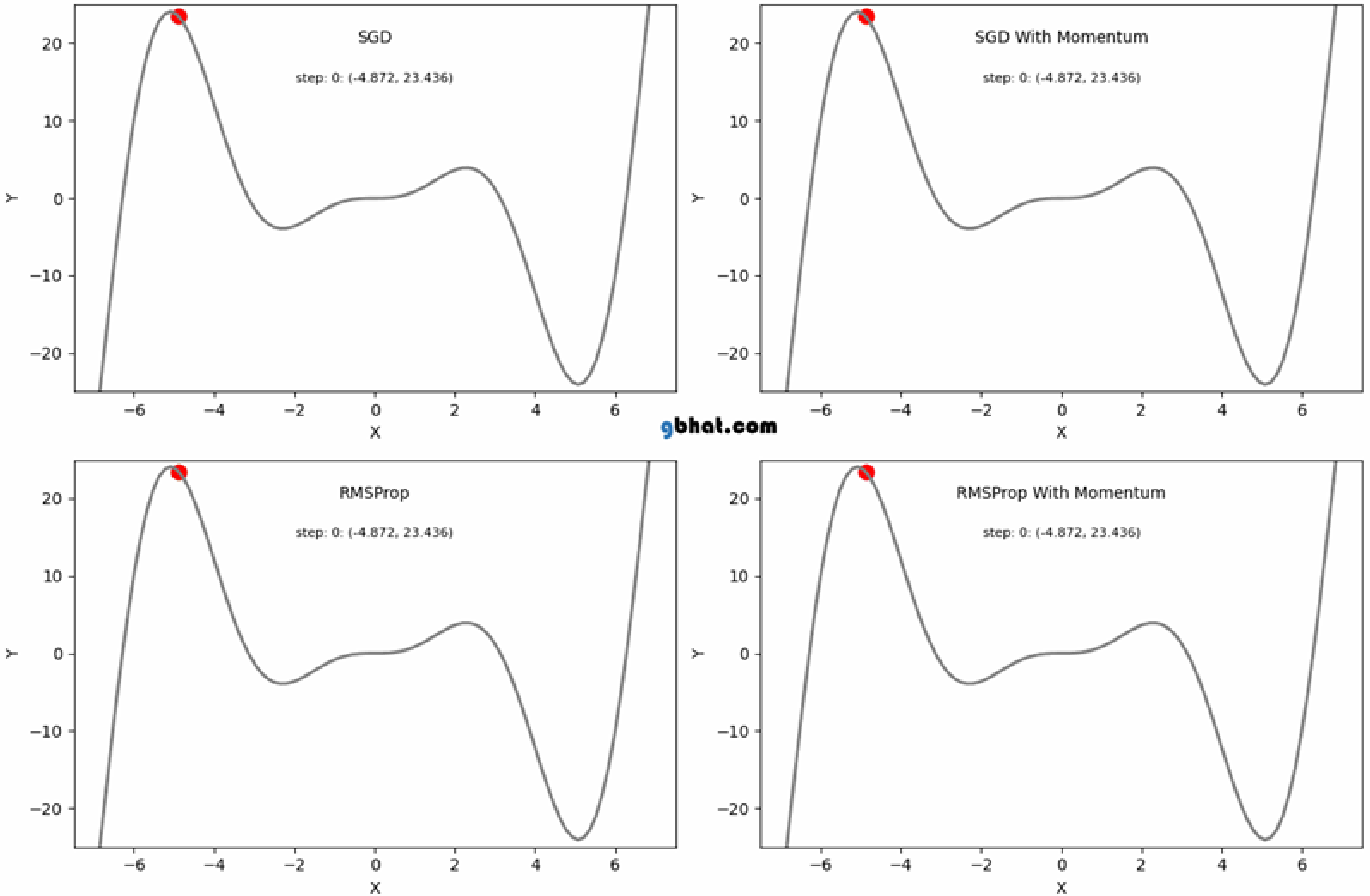
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Let’s build our own optimizer

$$w_t = w_{t-1} - \frac{\partial L}{w} \tag{1}$$

$$w_t = w_{t-1} - \alpha \frac{\partial L}{w} \tag{2}$$


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


SGD
step: 0: (-4.872, 23.436)
SGD With Momentum
step: 0: (-4.872, 23.436)
RMSProp
step: 0: (-4.872, 23.436)
RMSProp With Momentum
step: 0: (-4.872, 23.436)
Y
X
20
10
0
−10
−20
−6
−4
−2
0
2
4
6
gbhat.com

Adding momentum

$$\beta_{mean} = 0.9 \tag{3}$$

$$m_0 = 0 \tag{4}$$

$$m_t = \beta_{mean} \cdot m_{t-1} - (1 - \beta_{mean}) \cdot \frac{\partial L}{w_{t-1}} \tag{5}$$

$$w_t = w_{t-1} - \alpha \cdot m_t \tag{6}$$

Adding normalisation

$$\beta_{mean} = 0.9 \tag{7}$$

$$\beta_{var} = 0.9 \tag{8}$$

$$\epsilon = 1e - 8 \tag{9}$$

$$m_0 = 0 \tag{10}$$

$$v_0 = 0 \tag{11}$$

$$m_t = \beta_{mean} \cdot m_{t-1} - (1 - \beta_{mean}) \cdot \frac{\partial L}{w_{t-1}} \tag{12}$$

$$v_t = \beta_{var} \cdot v_{t-1} - (1 - \beta_{var}) \cdot \left(\frac{\partial L}{w_{t-1}}\right)^2 \tag{13}$$

$$w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{14}$$

Removing bias towards 0

$$\beta_{mean} = 0.9 \tag{15}$$

$$\beta_{var} = 0.9 \tag{16}$$

$$\epsilon = 1e-8 \tag{17}$$

$$m_0 = 0 \tag{18}$$

$$v_0 = 0 \tag{19}$$

$$m_t = \beta_{mean} \cdot m_{t-1} - (1-\beta_{mean}) \cdot \frac{\partial L}{w_{t-1}} \tag{20}$$

$$v_t = \beta_{var} \cdot v_{t-1} - (1-\beta_{var}) \cdot \left(\frac{\partial L}{w_{t-1}}\right)^2 \tag{21}$$

$$\hat{m}_t = \frac{m_t}{(1-\beta_{mean}^t)} \tag{22}$$

$$\hat{v}_t = \frac{v_t}{(1-\beta_{var}^t)} \tag{23}$$

$$w_t = w_{t-1} - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{24}$$

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

What optimizer should I use?

- (2014) SGD with nesterov momentum
- (2018) Adam
- (2022) AdamW + cosine annealed LR
- (2026) ???

Fully-sharded data-parallel

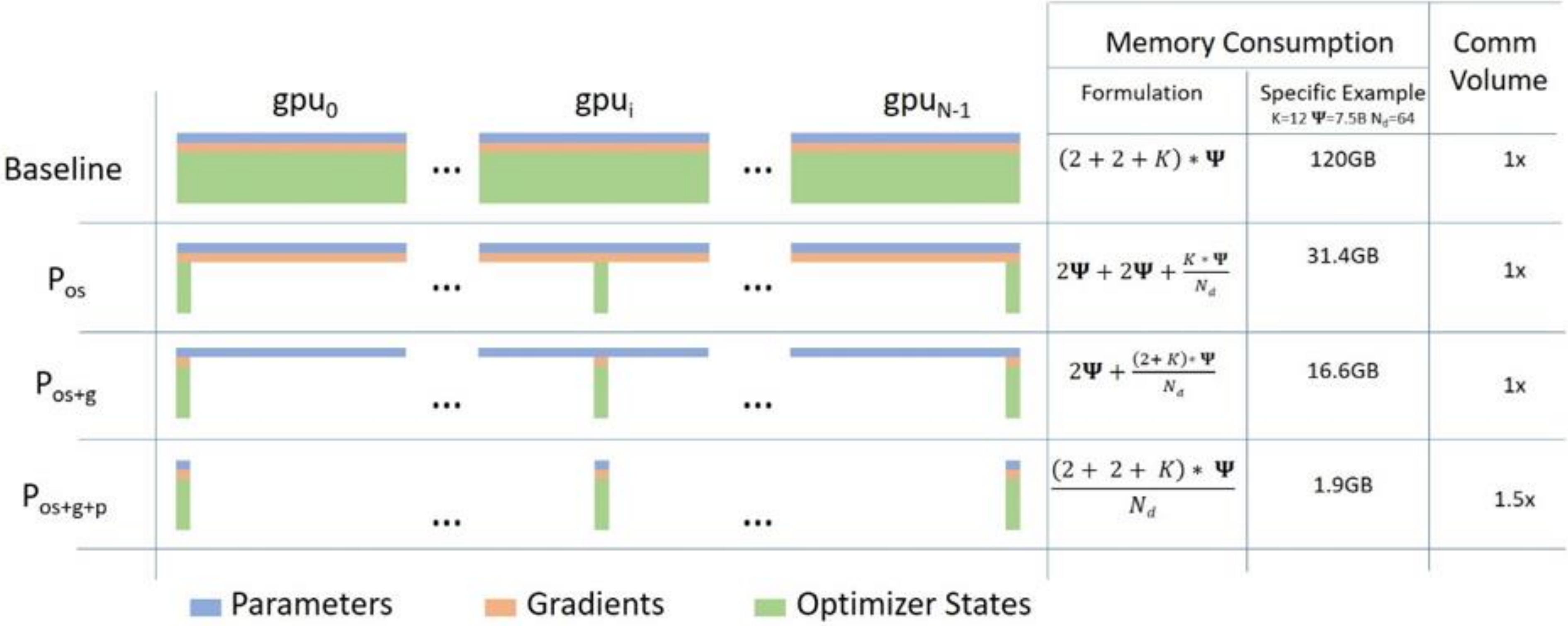
What & Why

- Models are way too big: 175B-1T parameters (325GB for only the parameters in FP16!)
- Modern GPUs have ~40GB VRAM, Google TPUv4 32GB
- Idea: Model parallelism & data parallelism at the same time
- Shard model over workers & offload grads/params/etc to CPU RAM
- Extends ‘DeepSpeed ZeRO’

3 options:

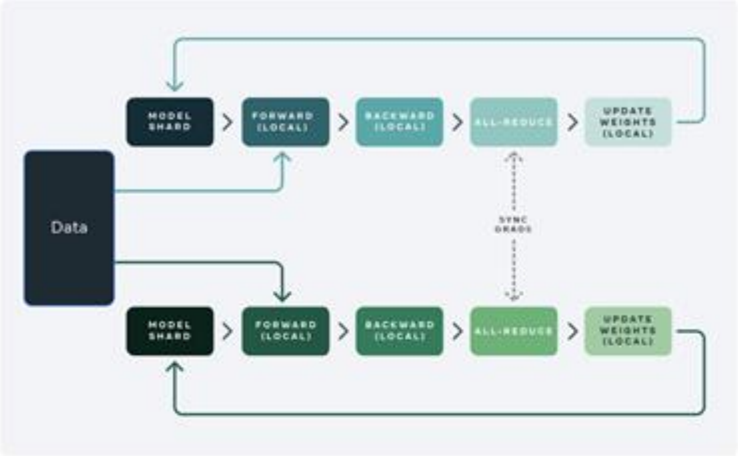
1. Shard optimizer state
2. Shard optimizer state + gradients
3. Shard optimizer state + gradients + parameters (+ activations?)

- CPU offload?
- Gradient checkpointing?

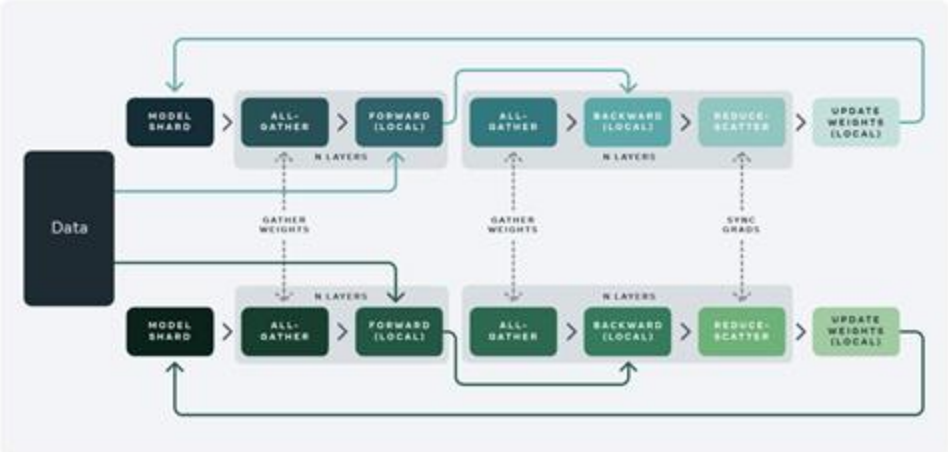


- OS = Optimizer State Partitioning
- G = Gradients
- P = Parameters

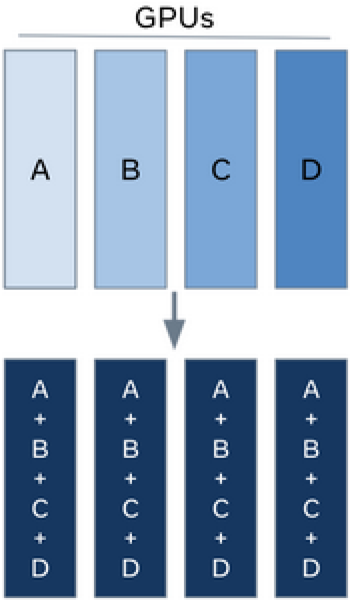
Standard data parallel training



Fully sharded data parallel training



All Reduce



FSDP Workflow

In constructor

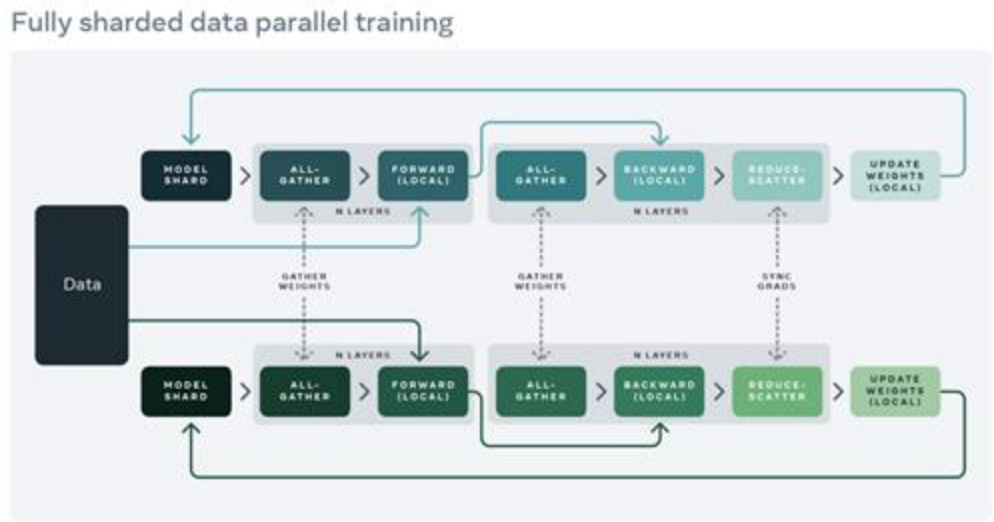
- Shard model parameters and each rank only keeps its own shard

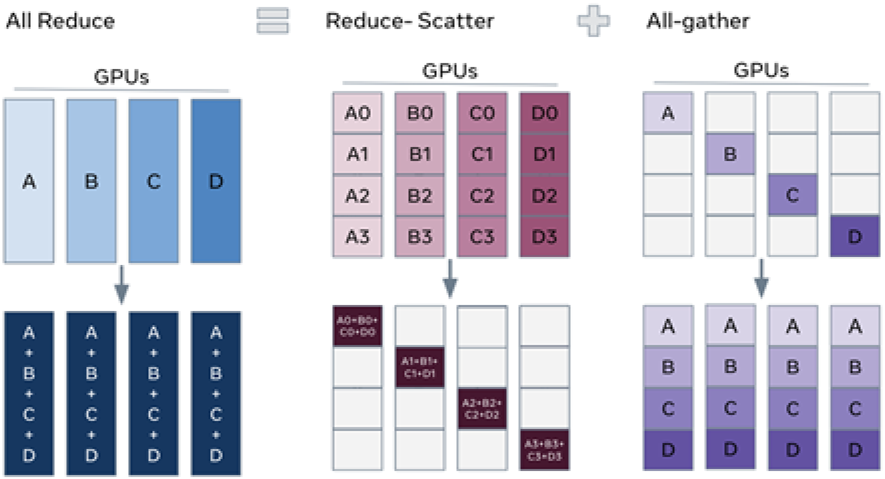
In forward path

- Run all_gather to collect all shards from all ranks to recover the full parameter in this FSDP unit
- Run forward computation
- Discard parameter shards it has just collected

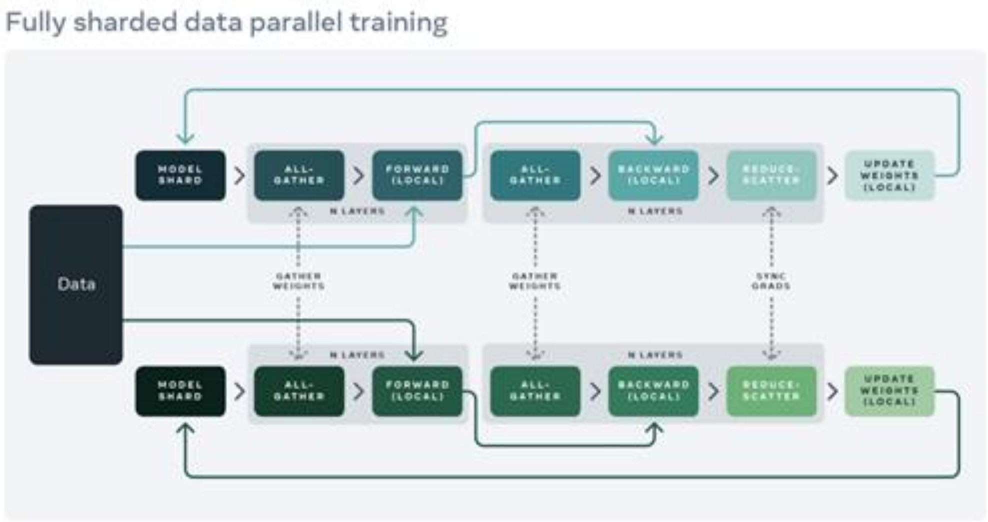
In backward path

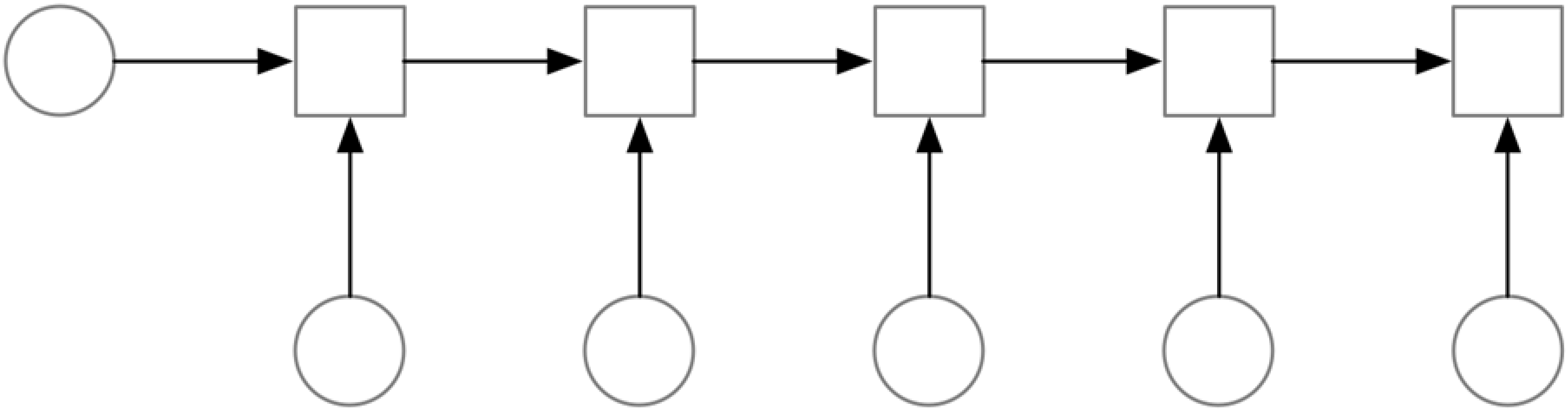
- Run all_gather to collect all shards from all ranks to recover the full parameter in this FSDP unit
- Run backward computation
- Run reduce_scatter to sync gradients
- Discard parameters.





Gradient checkpointing



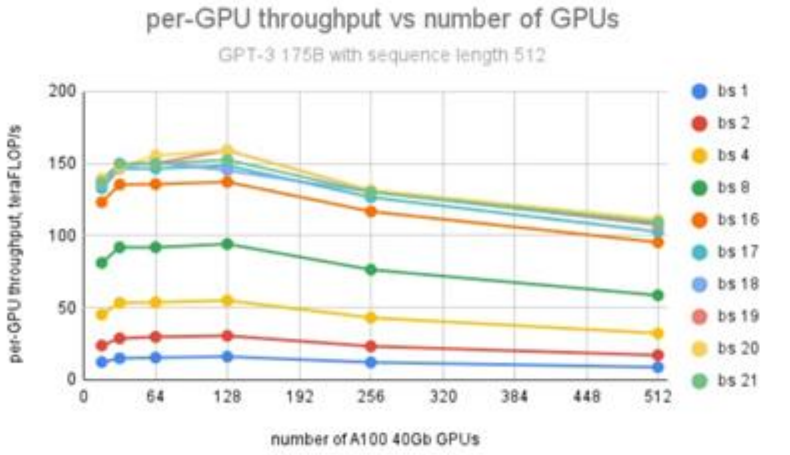


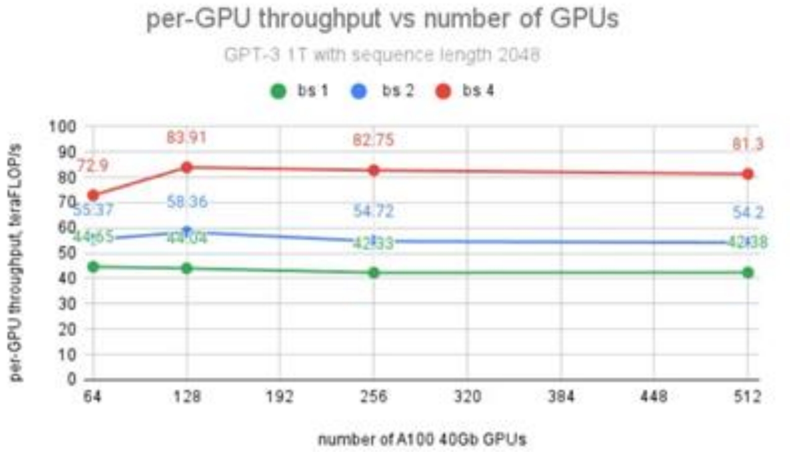
Throughput

- 175B: ~51% maximum per-gpu
- 1T: ~27% maximum per-gpu

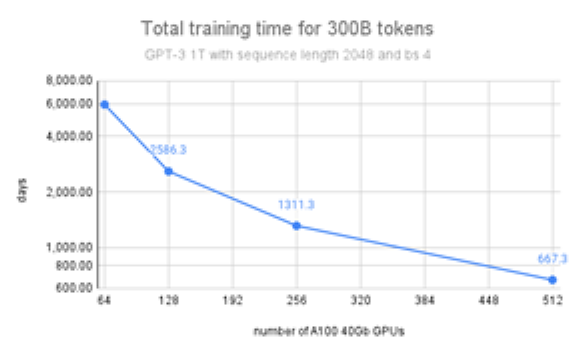
One big optimization problem:

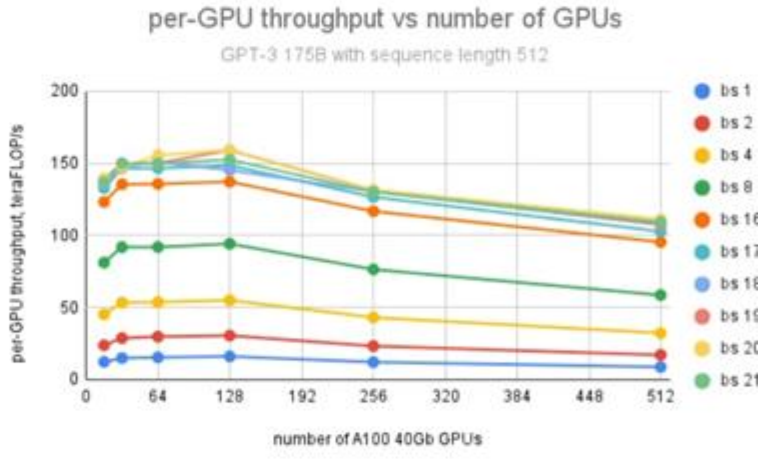
- If there are too many shards, we're wasting communication over computation
- If the shards are too small, we're inducing latency into the communication
- Need to take into account the topology of the network
- Need to take into account the memory size and theoretical throughput of our workers
- Always limited by networking

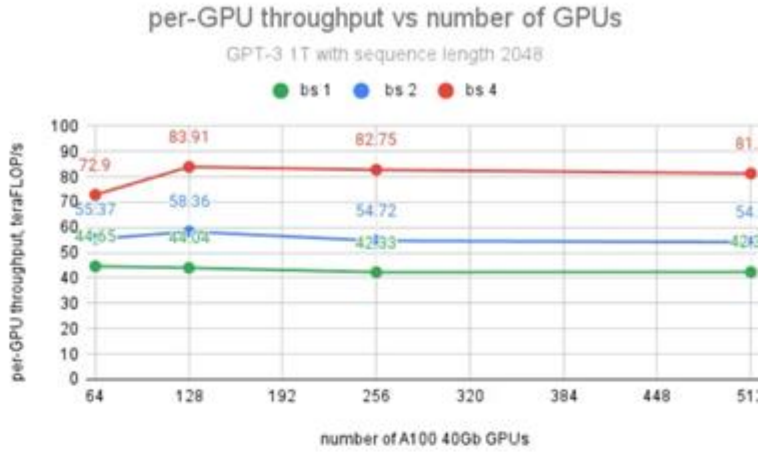




Throughput







- 175B: “Per our estimate, it would take 128 NVIDIA A100 40GB GPUs running for about 240 days to train GPT-3 175B using FSDP. According to current AWS public pricing, the strategy we would pick is to reserve 16 p4d.24xlarge instances for a duration of 1 year.”
- 1T: “Based on the total training time curve and current AWS pricing for 1 year and 3 years reservation, we suggest 2 possible strategies for training 1T GPT-like neural networks using PyTorch FSDP. Fast: 1-year training across 128 p4d.24xlarge instances, and Long: 3 years training across 43 p4d.24xlarge instances.”
- https://medium.com/pytorch/training-a-1-trillion-parameter-model-with-pytorch-fully-sharded-data-parallel-on-aws-3ac13aa96cff

In the real world:

```
model = MyModel()
trainer = Trainer(accelerator="gpu", devices=4, strategy="fsdp", precision=16)
trainer.fit(model)
```

In the real world:

```
model = MyModel()
trainer = Trainer(accelerator="gpu", devices=4, strategy="fsdp", precision=16)
trainer.fit(model)
```

https://pytorch-lightning.readthedocs.io/en/stable/advanced/model_parallel.html

https://fairscale.readthedocs.io/en/stable/api/nn/fsdp.html

Method	Batch Size Max ($BS)	Approx Train Time (minutes)
DDP (Distributed Data Parallel)	7	15
DDP + FP16	7	8
FSDP with SHARD_GRAD_OP	11	11
FSDP with min_num_params = 1M + FULL_SHARD	15	12
FSDP with min_num_params = 2K + FULL_SHARD	15	13
FSDP with min_num_params = 1M + FULL_SHARD + Offload to CPU	20	23
FSDP with min_num_params = 2K + FULL_SHARD + Offload to CPU	22	24