DISTRIBUTED DEEP LEARNING

Sri Koundinyan
skoundinyan@nvidia.com
Mike Matus
mmatus@nvidia.com

WELCOME!

Content applicable to H100s as well as GPUs in general

Many flavors distributed deep learning. Data Parallelism is the focus today.

Bulk of today will be hands on exercise!

ENVIRONMENT SETUP

Navigate to: https://portal-aces.hprc.tamu.edu/

Sign in using ACES account credentials

Open Terminal window, ssh into appropriate node

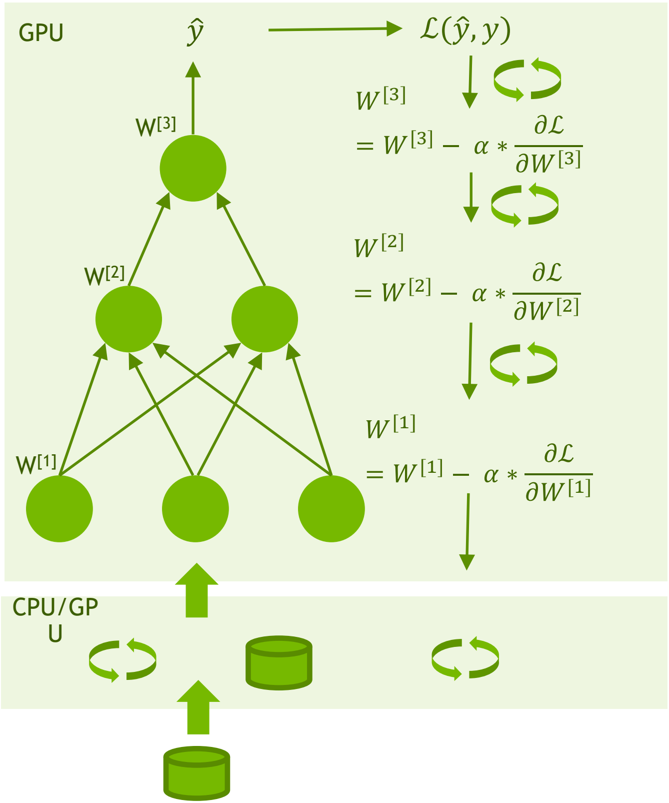
Configure environment for hands on exercise:

```
ml purge
ml WebProxy
singularity pull pytorch.sif docker://nvcr.io/nvidia/pytorch:23.06-py3
singularity shell --nv pytorch.sif
```

DATA PARALLELISM THEORY

TRAINING A NEURAL NETWORK

Single GPU



GPU

$\hat{y}$

$\mathcal{L}(\hat{y}, y)$

W[3]

W[2]

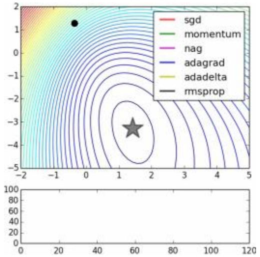
W[1]

$W^{[3]} = W^{[3]} - \alpha * \frac{\partial \mathcal{L}}{\partial W^{[3]}}$

$W^{[2]} = W^{[2]} - \alpha * \frac{\partial \mathcal{L}}{\partial W^{[2]}}$

$W^{[1]} = W^{[1]} - \alpha * \frac{\partial \mathcal{L}}{\partial W^{[1]}}$

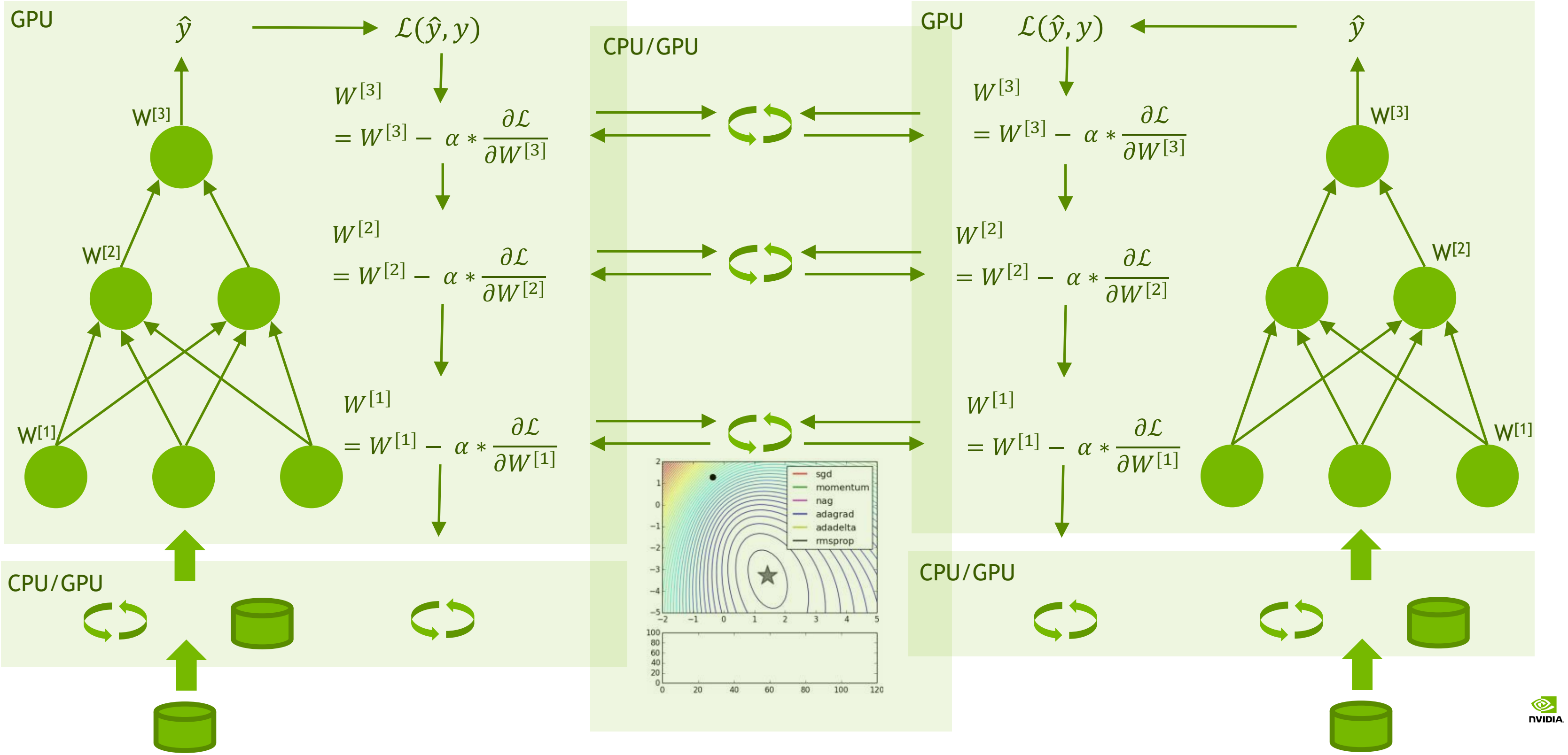
CPU/GP
U



1. Read the data
2. Transport the data
3. Pre-process the data
4. Queue the data
5. Transport the data
6. Calculate activations for layer one
7. Calculate activations for layer two
8. Calculate the output
9. Calculate the loss
10. Backpropagate through layer three
11. Backpropagate through layer two
12. Backpropagate through layer one
13. Execute optimization step
14. Update the weights
15. Return control

TRAINING A NEURAL NETWORK

Multiple GPUs



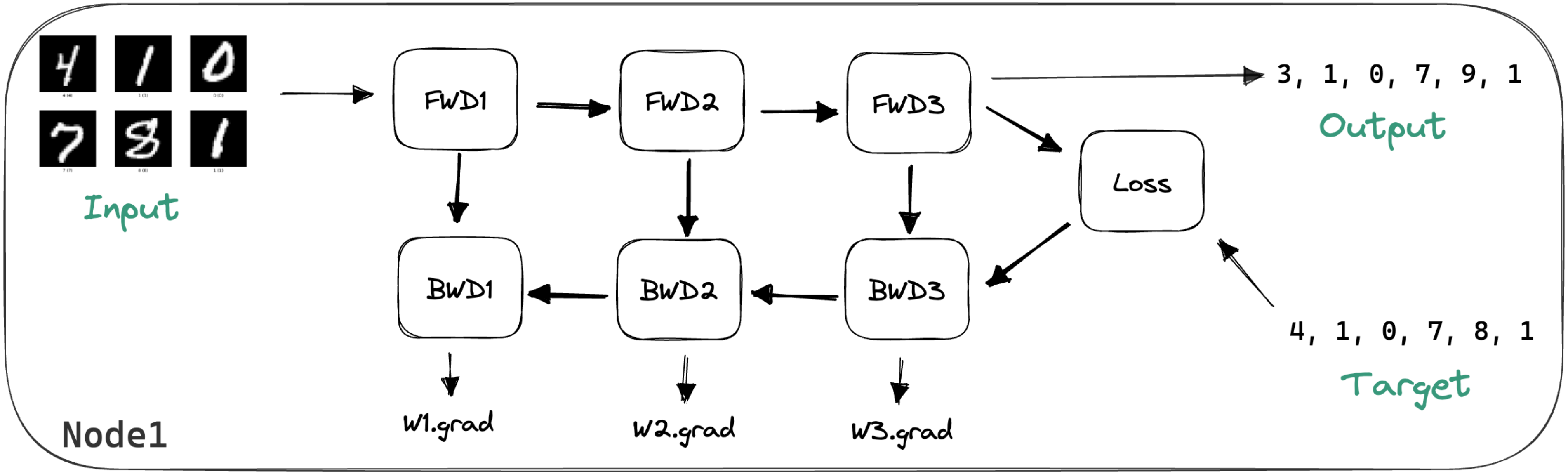
$$\mathcal{L}(\hat{y}, y)$$

$$W^{[3]} = W^{[3]} - \alpha * \frac{\partial \mathcal{L}}{\partial W^{[3]}}$$

$$W^{[2]} = W^{[2]} - \alpha * \frac{\partial \mathcal{L}}{\partial W^{[2]}}$$

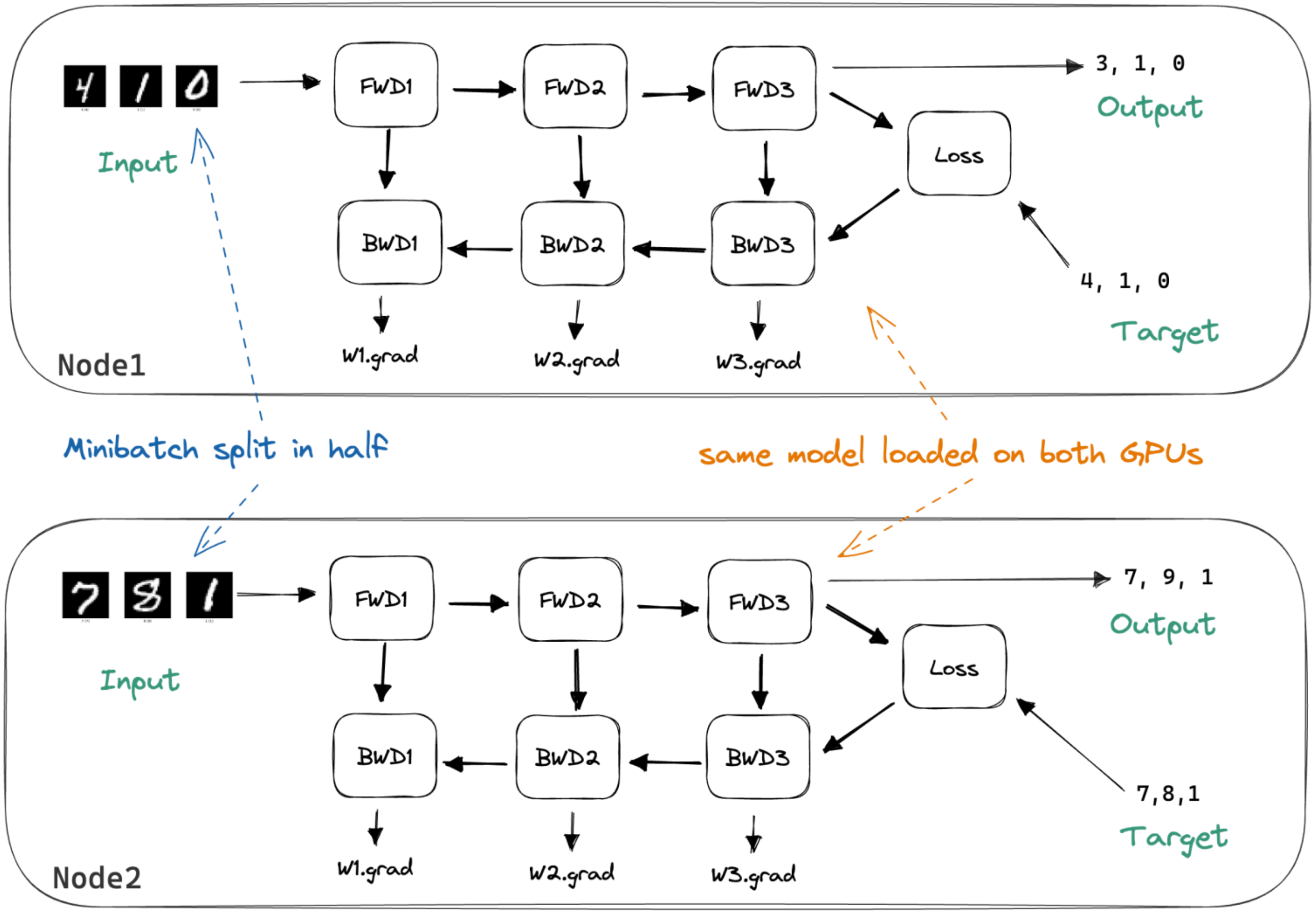
$$W^{[1]} = W^{[1]} - \alpha * \frac{\partial \mathcal{L}}{\partial W^{[1]}}$$

Sequential training



Minibatch size: 6

Data parallel training with 2 compute nodes



$$\begin{aligned}\frac{\partial\,\text{Loss}}{\partial w} &= \frac{\partial\left[\frac{1}{n}\sum_{i=1}^{n} f(x_i, y_i)\right]}{\partial w} \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial f(x_i, y_i)}{\partial w} \\ &= \frac{m_1}{n}\frac{\partial\left[\frac{1}{m_1}\sum_{i=1}^{m_1} f(x_i, y_i)\right]}{\partial w} + \frac{m_2}{n}\frac{\partial\left[\frac{1}{m_2}\sum_{i=m_1+1}^{m_1+m_2} f(x_i, y_i)\right]}{\partial w} + \cdots \\ &\quad + \frac{m_k}{n}\frac{\partial\left[\frac{1}{m_k}\sum_{i=m_{k-1}+1}^{m_{k-1}+m_k} f(x_i, y_i)\right]}{\partial w} \\ &= \frac{m_1}{n}\frac{\partial l_1}{\partial w} + \frac{m_2}{n}\frac{\partial l_2}{\partial w} + \cdots + \frac{m_k}{n}\frac{\partial l_k}{\partial w}\end{aligned}$$

Where

w is the parameters of the model,

$\frac{\partial\,\text{Loss}}{\partial w}$ is the true gradient of the big batch of size n,

$\frac{\partial l_k}{\partial w}$ is the gradient of the small batch in GPU/node k,

x_i and y_i are the features and labels of data point i,

$f(x_i, y_i)$ is the loss for data point i calculated from the forward propagation,

n is the total number of data points in the dataset,

k is the total number of GPUs/nodes,

m_k is the number of data points assigned to GPU/node k,

$m_1 + m_2 + \cdots + m_k = n$.

When $m_1 = m_2 = \cdots = m_k = \frac{n}{k}$, we could further have

$$\frac{\partial\,\text{Loss}}{\partial w} = \frac{1}{k}\left[\frac{\partial l_1}{\partial w} + \frac{\partial l_2}{\partial w} + \cdots + \frac{\partial l_k}{\partial w}\right]$$

MEET DDP

Library for distributed DL

Prepackaged into and optimized for PyTorch, an increasingly popular platform among ML engineers and researchers

PyTorch

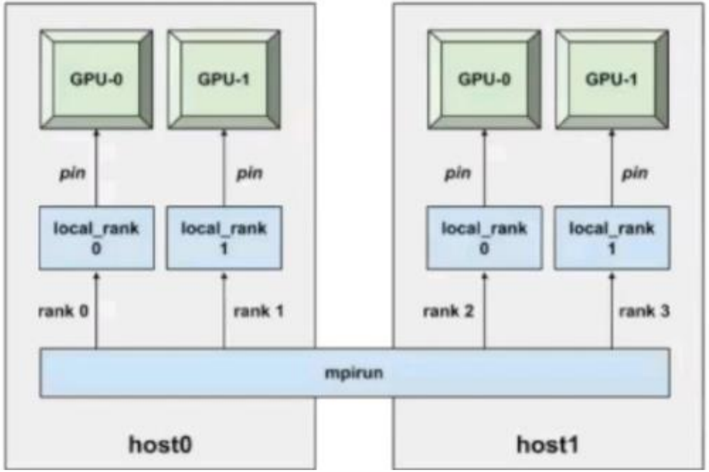
USING DDP

INITIALIZE THE PROCESS

```
def setup(global_rank, world_size):
        dist.init_process_group(backend="nccl", rank=global_rank,
        world_size=world_size)
```

PIN GPU TO BE USED

```
device = torch.device("cuda:" + str(local_rank))
model = Net().to(device)
```



ENCAPSULATE MODEL WITH DDP

```
model = nn.parallel.DistributedDataParallel(model,
device_ids=[local_rank])
```

SYNCHRONIZE INITIAL STATE

Handled internally by DDP across processes and nodes!

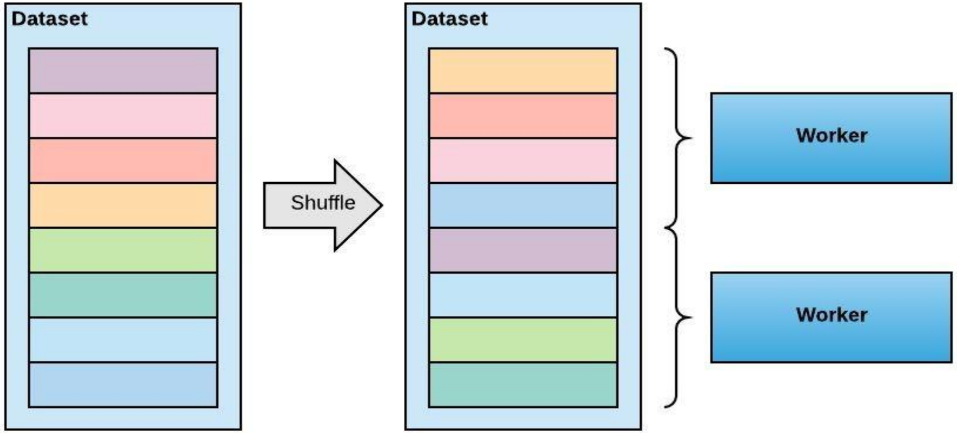
DATA PARTITIONING

Shuffle the dataset

Partition records among workers

Train by sequentially reading the partition

After epoch is done, reshuffle and partition again



DATA PARTITIONING

```
train_sampler =
torch.utils.data.distributed.DistributedSampler(train_set,
num_replicas=world_size, rank=global_rank)

train_loader =
torch.utils.data.DataLoader(train_set,
batch_size=args.batch_size, sampler=train_sampler)
```

I/O ON ONLY ON ONE WORKER

```
download = True if local_rank == 0 else False
if local_rank == 0:
       train_set = torchvision.datasets.FashionMNIST("./data",
download=download)

--------------------

if global_rank == 0:
       print("Epoch = {:2d}: Validation Loss = {:5.3f},
       Validation Accuracy = {:5.3f}".format(epoch+1, v_loss,
       val_accuracy[-1]))
```

NVIDIA