Additional Techniques

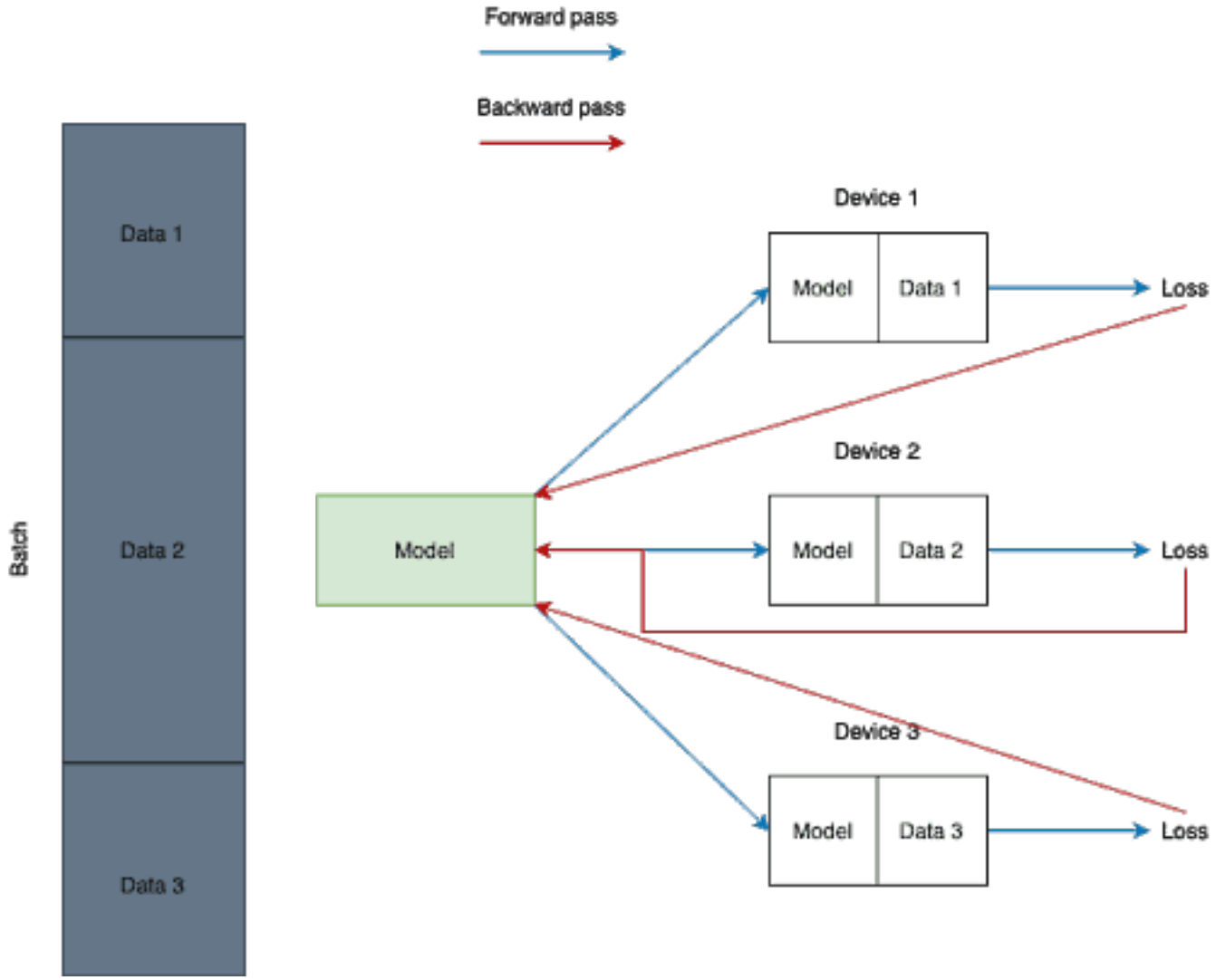
Tony T. Luo

May 2024

Parallelizing the training process

Use torch.nn.DataParallel

Idea: Split the input across CUDA devices by dividing the batch into several parts.



In the **forward** pass, the model is replicated on each device, and each replica handles a portion of the input.

During the **backward** pass, gradients from each replica are summed into the original model.

```
gpu_list = [‘0’, ‘1’, ‘2’]
os.environ['CUDA_VISIBLE_DEVICES'] = gpu_list

model = DataParallel(model.cuda(), device_ids = gpu_list)
```

```
GPU = 0, 1
gpu_list = ''
multi_gpus = False
if isinstance(GPU, int):
    gpu_list = str(GPU)
else:
    multi_gpus = True
    for i, gpu_id in enumerate(GPU):
        gpu_list += str(gpu_id)
        if i != len(GPU) - 1:
            gpu_list += ','
os.environ['CUDA_VISIBLE_DEVICES'] = gpu_list

net = net.cuda()
if multi_gpus:
    net = DataParallel(net, device_ids = gpu_list)
```

Save and Load Models / Checkpoints

- Useful to save the time-consuming training process after it is done
- Also useful to save a checkpoint and resume later

Three Methods

- **torch.save(model, PATH)** # can be model, tensor, or dictionary
- **torch.load(PATH)**
- **torch.load_state_dict(model)**

```python
# 1) save whole model
torch.save(model, PATH)

model = torch.load(PATH)
model.eval()

# 2) save only the state_dict (recommended)
torch.save(model.state_dict(), PATH)

# model class must be defined somewhere, model must be created again
model = Model(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

Coding Demo