Data Parallel Training

Data Parallel Training

- we can have multiple GPUs on a single machine
- how can we take advantage of them?
- data parallel training:
  - split input batch into multiple smaller batches
  - train the same model on each GPU
  - sum up the gradients across GPUs
  - update the weights consistently across GPUs
- implemented by `torch.nn.DataParallel`
- simple modification of a model

In [1]:

```
%pylab inline
!date; hostname; whoami; pwd; curl https://ipinfo.io/hostname; nvidia-smi -L
from imp import reload
from torch import nn, optim
from torch.nn import functional as F
from torchmore import layers, flex
import torch
from torchvision import datasets, transforms
from torchvision.datasets import imagenet
import os.path
from torch.utils import data as torchdata
import helpers
```

```
Populating the interactive namespace from numpy and matplotlib
Sun Dec  8 02:56:34 UTC 2019
tmbcomp
tmb
/home/tmb/exp/bigdata19
158.153.83.34.bc.googleusercontent.com
GPU 0: Tesla V100-SXM2-16GB (UUID: GPU-ffa2b7fc-dde2-eb1c-7481-861ea0f181a3)
GPU 1: Tesla V100-SXM2-16GB (UUID: GPU-80d5ad59-f785-94da-ec20-cb43f2114bd1)
GPU 2: Tesla V100-SXM2-16GB (UUID: GPU-e1896053-8373-94a6-c0d3-febdfe9312ca)
GPU 3: Tesla V100-SXM2-16GB (UUID: GPU-ba5fd29a-104c-ac84-b13d-f04ff2f6f9bc)

/opt/conda/lib/python3.6/site-packages/torchvision/io/_video_opt.py:17: UserWarning: video reader ba
sed on ffmpeg c++ ops not available
  warnings.warn("video reader based on ffmpeg c++ ops not available")
```

Mock Loader for Benchmarking

In [2]:

```
class MockLoader(object):
    def __init__(self, shape):
        self.shape = shape
        self.data = torch.rand(shape).cuda()
        self.targets = torch.zeros((shape[0],), dtype=torch.int64)
    def __iter__(self):
        while True:
            yield self.data, self.targets

batch_size = 128
training_dl = MockLoader((batch_size, 3, 224, 224))
inputs, targets = next(iter(training_dl))
```

Serial Model

In [3]:

```
from torchvision import models
def make_model():
    return models.resnet50()
```

In [4]:

```
model = make_model()
trainer = helpers.Trainer(model.cuda())
trainer.set_lr(1e-6)
trainer.train_for(5000, training_dl, quiet=True)
serial = trainer.timers.training/batch_size
del model
```

DataParallel Model

Note: only one change.

In [5]:

```
model = make_model()
model = nn.DataParallel(make_model())
trainer = helpers.Trainer(model.cuda())
trainer.set_lr(1e-6)
trainer.train_for(5000, training_dl, quiet=True)
parallel = trainer.timers.training/batch_size
del model
```

In [6]:

```
print(serial, parallel, serial/parallel)
```

```
0.003332657925784588 0.003342693066224456 0.9969978875592062
```

Bigger Batch Size

In [8]:

```
batch_size = 512
training_dl = MockLoader((batch_size, 3, 224, 224))

model = make_model()
model = nn.DataParallel(make_model())
trainer = helpers.Trainer(model.cuda())
trainer.set_lr(1e-6)
trainer.train_for(5000, training_dl, quiet=True)
parallel512 = trainer.timers.training/batch_size
```

In [9]:

```
print(serial, parallel, parallel512, serial/parallel512)
```

```
0.003332657925784588 0.003342693066224456 0.0012320765294134618 2.704911461442352
```

Combining DataParallel with FP16

In [17]:

```
from apex import amp

class ParallelAmpTrainer(helpers.Trainer):
    def __init__(self, model):
        super().__init__(model)
    def set_lr(self, lr):
        optimizer = optim.SGD(self.model.parameters(), lr=lr, momentum=0.9)
        self.model, self.optimizer = amp.initialize(
            self.model, optimizer, opt_level="O1", loss_scale="dynamic")
        self.model = nn.DataParallel(self.model)

batch_size = 512
training_dl = MockLoader((batch_size, 3, 224, 224))

model = make_model()
trainer = ParallelAmpTrainer(model.cuda())
trainer.set_lr(1e-6)
trainer.train_for(5000, training_dl, quiet=True)
parallelamp = trainer.timers.training/batch_size
print(parallel512, parallelamp, parallel512/parallelamp)
```

```
Selected optimization level O1:  Insert automatic casts around Pytorch functions and Tensor methods.

Defaults for this optimization level are:
enabled                 : True
opt_level               : O1
cast_model_type         : None
patch_torch_functions   : True
keep_batchnorm_fp32     : None
master_weights          : None
loss_scale              : dynamic
Processing user overrides (additional kwargs that are not None)...
After processing overrides, optimization options are:
enabled                 : True
opt_level               : O1
cast_model_type         : None
patch_torch_functions   : True
keep_batchnorm_fp32     : None
master_weights          : None
loss_scale              : dynamic
0.0012320765294134618 0.0009898143354803324 1.2447551881692696
```

DataParallel

- `DataParallel` is a simple way of using multiple GPUs
- effectively you end up with much bigger batch sizes
- may create an I/O bottleneck
- consider using `DistributedDataParallel` even on a single node