PyTorch Lightning

Amin Saied

In the beginning…

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```python
# download CIFAR 10 data
trainset = torchvision.datasets.CIFAR10(
    root="../data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor(),
)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=4, shuffle=True, num_workers=2
)
```

```python
# define convolutional network
net = Net()

# set up pytorch loss /  optimizer
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

# train the network
for epoch in range(2):

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # unpack the data
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:
            loss = running_loss / 2000
            print(f"epoch={epoch + 1}, batch={i + 1:5}: loss {loss:.2f}")
            running_loss = 0.0
```

- Iterate through data
- Forward pass
- Backward pass
- Handle gradients
- Log metrics
- Validation loop

What's the problem?

Fundamental issue is that this single loop combines disparate notions:

- Model code
- Dataloaders
- Training
  - Including logging
- Validation
  - Applying torch.no_grad()
- Optimization
- Infra
  - E.g., Distributed training, mixed-precision,
  - deepspeed, ORT, …

Example: Distributed training with DDP

- Distributed dataloaders
- Init process group
- Wrap model in DDP
- Distributed checkpointing
- Check LR schedulers handle global step
- …

```
import torch.distributed as dist
from torch.nn.parallel import DistributedDataParallel as DDP

# init the process group
dist.init_process_group("nccl", rank=rank, world_size=world_size)

# create model and move it to GPU with id rank
model = AmazingModel().to(rank)
ddp_model = DDP(model, device_ids=[rank])

CHECKPOINT_PATH = tempfile.gettempdir() + "/model.checkpoint"
if rank == 0:
    # All processes should see same parameters as they all start from same
    # random parameters and gradients are synchronized in backward passes.
    # Therefore, saving it in one process is sufficient.
    torch.save(ddp_model.state_dict(), CHECKPOINT_PATH)

# Use a barrier() to make sure that process 1 loads the model after process
# 0 saves it.
dist.barrier()
# configure map_location properly
map_location = {'cuda:%d' % 0: 'cuda:%d' % rank}
ddp_model.load_state_dict(
    torch.load(CHECKPOINT_PATH, map_location=map_location))
```

Hugginface transformer.Trainer()

- Convenient out-of-the-box trainer with flags to control basic use cases

Drawbacks

- Tightly coupled to transformers library
- Over time these trainer's become unwieldy black boxes of tech debt
- For example, the Huggingface trainer is **>2500 loc** with **~500 if/else conditions**.

```
# grab data
train_dataset = WhatsppPromptsDataset(data_files["train"])
eval_dataset = WhatsppPromptsDataset(data_files["eval"])

# set up model
tokenizer = GPT2TokenizerFast.from_pretrained(args.model_checkpoint)
model = AutoModelForCausalLM.from_pretrained(args.model_checkpoint)

# configure Trainer
training_args = TrainingArguments(
    output_dir=args.output_dir,
    overwrite_output_dir=True,
    num_train_epochs=3,
    learning_rate=1e-5,
    weight_decay=0.01,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    evaluation_strategy="steps",
    logging_dir='./logs',
    logging_steps=100,
    local_rank=args.local_rank,
    deepspeed=args.deepspeed,
)

# define trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
)

# train!
trainer.train()
```

https://github.com/huggingface/transformers/blob/master/src/transformers/trainer.py

Demo

Debugging

- Fast Dev Run

```
trainer = Trainer(fast_dev_run=True)
```

  - Runs a “unit test” by running 1 training batch and 1 validation batch.
  - Detect bugs in the training/validation loop without having to wait for a full epoch
- Inspect Grad Norms

```
trainer = Trainer(track_grad_norm=2)
```

  - Track the L2-norm of each weight matrix

Debugging

- Performance Profiling

```
trainer = Trainer(..., profiler=True)
```

```
Profiler Report

Action                  |  Mean duration (s)    |  Total time (s)
-----------------------------------------------------------------
on_epoch_start          |  5.993e-06            |  5.993e-06
get_train_batch         |  0.0087412            |  16.398
on_batch_start          |  5.0865e-06           |  0.0095372
model_forward           |  0.0017818            |  3.3408
model_backward          |  0.0018283            |  3.4282
on_after_backward       |  4.2862e-06           |  0.0080366
optimizer_step          |  0.0011072            |  2.0759
on_batch_end            |  4.5202e-06           |  0.0084753
on_epoch_end            |  3.919e-06            |  3.919e-06
on_train_end            |  5.449e-06            |  5.449e-06
```

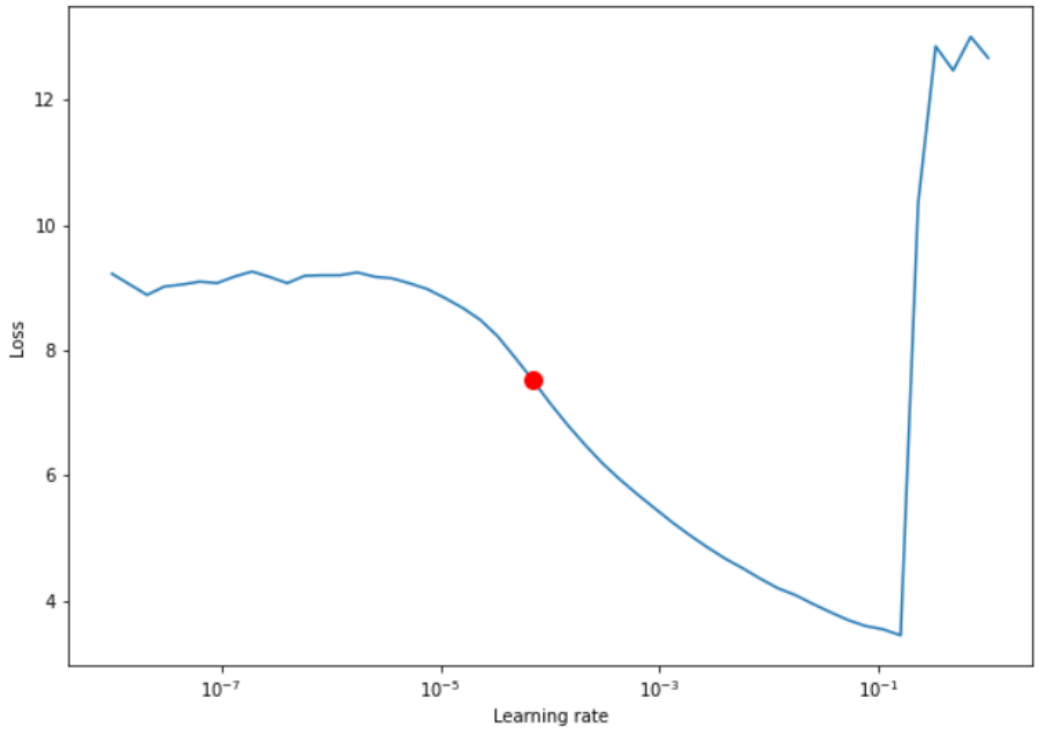
```
trainer = Trainer(..., profiler="pytorch")
```

```
Profile stats for: training_step_and_backward rank: 0
--------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                                  Name    Self CPU %      Self CPU  CPU total %     CPU total  CPU time avg    # of Calls
--------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
                       EmbeddingBackward        0.01%      54.114ms        63.36%      393.103s      33.006ms         11910
                  aten::embedding_backward        0.01%      34.842ms        63.35%      393.048s      33.002ms         11910
             aten::embedding_dense_backward       63.15%     391.795s        63.34%      393.014s      32.999ms         11910
                           AddmmBackward          0.45%       2.808s         4.31%       26.757s      93.607us        285840
                             aten::addmm          3.11%      19.308s         3.60%       22.334s      78.136us        285840
                             aten::matmul          0.33%       2.057s         3.54%       21.989s     147.700us        148875
                               aten::mm            2.99%      18.580s         3.29%       20.397s      34.597us        589545
                             aten::empty           2.46%      15.293s         2.46%       15.293s       2.979us       5133210
           torch::autograd::AccumulateGrad         1.08%       6.679s         2.43%       15.066s      17.095us        881340
                             aten::mul             2.02%      12.521s         2.42%       14.985s      17.354us        863475
                             aten::bmm             1.97%      12.211s         2.30%       14.287s      33.322us        428760
                             aten::view            2.12%      13.123s         2.12%       13.123s       4.407us       2977500
    Optimizer.zero_grad#AdamW.zero_grad            0.79%       4.894s         1.98%       12.268s       2.060ms          5955
                           ViewBackward            0.40%       2.504s         1.91%       11.823s      10.181us       1161225
                       aten::empty_like            0.53%       3.276s         1.79%       11.104s       7.400us       1500660
                           BmmBackward0            0.19%       1.205s         1.76%       10.889s      76.192us        142920
                             aten::reshape         0.37%       2.286s         1.63%       10.122s       7.657us       1322010
                              aten::copy_          1.58%       9.826s         1.58%       9.826s       17.553us        559770
                             aten::dropout         0.13%     832.536ms        1.55%       9.602s       43.579us        220335
                        aten::_fused_dropout       0.94%       5.838s         1.41%       8.769s       39.800us        220335
--------------------------------------  ------------  ------------  ------------  ------------  ------------  ------------
Self CPU time total: 620.466s
```

Auto LR Finder

- Lightning implements an automated learning rate finder based on [1]

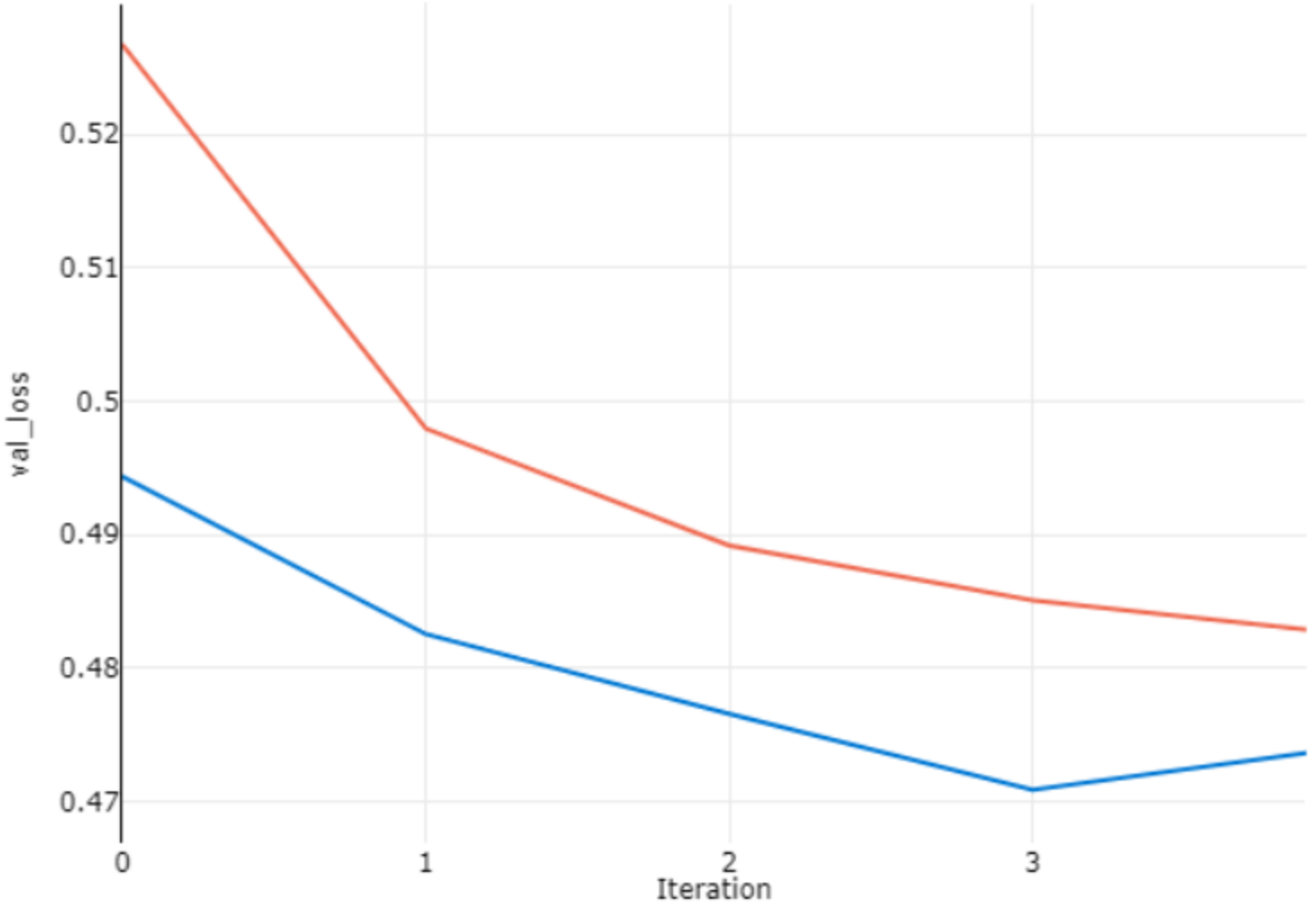
```
import pytorch_lightning as pl
from pytorch_lightning.tuner.tuning import Tuner

autome_clm_args = AutomeCLMArguments()

autome_clm = AutomeCLM(
    train_dataset=train_dataset,
    val_dataset=val_dataset,
    hparams=autome_clm_args,
)

trainer = pl.Trainer()
tuner = Tuner(trainer)

lr_find_output = tuner.lr_find(autome_clm, num_training=50)

fig = lr_find_output.plot(suggest=True)
fig.show()

lr_find_output.suggestion()
# 6.918309709189363e-05
```



[1] - *Cyclical Learning Rates for Training Neural Networks* by Leslie N. Smith

Auto LR Finder

- Comparison with Huggingface default of 1e-5
- Red=1e-5
- Blue=6.9e-5



Accelerators

- Mixed precision
- Native DeepSpeed support:

```
from pytorch_lightning.plugins import DeepSpeedPlugin
deepspeed_plugin = DeepSpeedPlugin(
    config="config.json"  # or pass a dict
)
trainer = Trainer(..., plugins=[deepspeed_plugin], precision=16)
```

[Medium blog announcement]

```
def __init__(
    self,
    zero_optimization: bool = True,
    stage: int = 2,
    cpu_offload: bool = False,
    cpu_offload_params: bool = False,
    cpu_offload_use_pin_memory: bool = False,
    contiguous_gradients: bool = True,
    overlap_comm: bool = True,
    allgather_partitions: bool = True,
    reduce_scatter: bool = True,
    allgather_bucket_size: int = 2e8,
    reduce_bucket_size: int = 2e8,
    zero_allow_untested_optimizer: bool = True,
    config: Optional[Union[Path, str, dict]] = None,
    logging_level: int = logging.WARN,
    num_nodes: int = 1,
    parallel_devices: Optional[List[torch.device]] = None,
    cluster_environment: Optional[ClusterEnvironment] = None,
    loss_scale: float = 0,
    initial_scale_power: int = 16,
    loss_scale_window: int = 1000,
    hysteresis: int = 2,
    min_loss_scale: int = 1,
    partition_activations: bool = False,
    cpu_checkpointing: bool = False,
    contiguous_memory_optimization: bool = False,
    synchronize_checkpoint_boundary: bool = False,
    save_full_weights: bool = True,
)
```