Addison Kalanther (addikala@berkeley.edu)

Lecture 3: PyTorch

This is a quick overview of PyTorch, an autodiff framework used to train models.

What is a model?

A model is of the form

$$\theta^* = \text{argmin}_\theta \sum_{(x,y)\in D} \mathcal{L}(f_\theta(x), y)$$

with

- $\arg\min_\theta$ being the gradient descent,
- D being the dataset,
- $\mathcal{L}$ being the loss function, and
- f_θ being the neural network.

PyTorch does all of these and more.

Using GPU

NumPy is CPU-only and does not support autodiff. PyTorch allows for GPU use to parallelize processes and supports autodiff.

To move a PyTorch object to GPU, use `.to("cuda") / .cuda()`.

To move a PyTorch object to CPU, use `.to("cpu") / .cpu()`.

(New) To move a PyTorch object to MPS (Metal backend for Apple ARM processors), use `.to(torch.device("mps"))`.

Computing gradients

To compute the gradient of a Tensor, ensure `requires_grad=True`. Now, we can find the gradient of these variables when taking a gradient with respect to another value, most commonly loss.

Graphs for calculating gradients take up a lot of memory. To detach gradient values from Tensor, us `.detach()`.

Training loop

If there is anything to remember in PyTorch, it is these 3 lines.

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

`optimizer.zero_grad()` resets the optimizer to zero, `loss.backward()` computes the gradients with respect to loss, and `optimizer.step()` takes a step in the direction opposite of the gradients calculated by the `.backward()` call to perform gradient descent.

Example training loop

```
for epoch in range(num_epochs):
    net.train() # puts net in training mode
    for data, target in dataloader:
        data = torch.from_numpy(data).float().cuda()
        target = torch.form_numpy(target).float().cuda()

        prediction = net(data)
        loss = loss_fn(prediction, target) # loss takes (pred, truth)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    net.eval()
    # Do evaluation...
```

NumPy / PyTorch conversion

Numpy -> PyTorch: `torch.from_numpy(numpy_array).float()`

- `.float()` if trying to convert to float, most common for neural nets

PyTorch -> NumPy: `torch_tensor.detach().cpu().numpy()`

- `.detach()` only necessary if `require_grad=True`
- `.cpu()` only necessary if not on CPU