Introduction to Deep Learning (CS474)

Lecture 11

Outline

- Mechanics of Learning-PART VI
    - PyTorch `nn` module

Introduction

- PyTorch has a whole submodule dedicated to neural networks, called **torch.nn** .
- It contains the building blocks needed to create all sorts of neural network architectures.
- Those building blocks are called modules in PyTorch parlance (such building blocks are often referred to as layers in other frameworks).
- A PyTorch module is a Python class deriving from the nn.Module base class.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Introduction

- A module can have one or more Parameter instances as attributes, which are tensors whose values are optimized during the training process (think w and b in our linear model).

- A module can also have one or more submodules (subclasses of nn.Module) as attributes, and it will be able to track their parameters as well.

- We'll now start precisely where we left off and convert our previous code to a form that uses nn.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Example

```
import numpy as np
import torch
import torch.optim as optim
t_c = [0.5,  14.0, 15.0, 28.0, 11.0,  8.0,  3.0, -4.0,  6.0, 13.0, 21.0]
t_u = [35.7, 55.9, 58.2, 81.9, 56.3, 48.9, 33.9, 21.8, 48.4, 60.4, 68.4]
t_c = torch.tensor(t_c).unsqueeze(1)
t_u = torch.tensor(t_u).unsqueeze(1)
print(t_u.shape)

n_samples = t_u.shape[0]
n_val = int(0.2 * n_samples)

shuffled_indices = torch.randperm(n_samples)

train_indices = shuffled_indices[:-n_val]
val_indices = shuffled_indices[-n_val:]

train_indices, val_indices

t_u_train = t_u[train_indices]
t_c_train = t_c[train_indices]

t_u_val = t_u[val_indices]
t_c_val = t_c[val_indices]

t_un_train = 0.1 * t_u_train
t_un_val = 0.1 * t_u_val
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

Continuing with the earlier *notebook*!

```
import torch.nn as nn

linear_model = nn.Linear(1, 1) # <1>
linear_model(t_un_val)
```

```
tensor([[-4.9020],
        [-5.2945]], grad_fn=<AddmmBackward>)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

- All PyTorch-provided subclasses of nn.Module have their __call__ method defined.
- Calling an instance of nn.Module with a set of arguments ends up calling a method named ***forward*** with the same arguments.
- The ***forward*** method is what executes the forward computation, while __call__ does other rather important chores before and after calling ***forward***.
- The constructor to nn.Linear accepts three arguments: the number of input features, the number of output features, and whether the linear model includes a bias or not (defaulting to True , here).

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

continuing with the earlier notebook.

```
# Weight
linear_model.weight
```

```
Parameter containing:
tensor([[-0.7461]], requires_grad=True)
```

```
#bias
linear_model.bias
```

```
Parameter containing:
tensor([-0.2618], requires_grad=True)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

Assuming we need to run **`nn.Linear`** on **10 samples**, we can create an input tensor of size `B × Nin`, where **B** is the size of the *batch* and `Nin` is the number of input features, and run it once through the model.

- continuing with the earlier notebook.

```
x = torch.ones(10, 1)
linear_model(x)
```

```
tensor([[0.6130],
        [0.6130],
        [0.6130],
        [0.6130],
        [0.6130],
        [0.6130],
        [0.6130],
        [0.6130],
        [0.6130],
        [0.6130]], grad_fn=<AddmmBackward>)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Example

First, we replace our handmade model with nn.Linear(1,1) , and then we need to pass the linear model parameters to the optimizer.

- continuing with the earlier notebook.

```
linear_model = nn.Linear(1, 1)
optimizer = optim.SGD(
    linear_model.parameters(),
    lr=1e-2)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

- continuing with the earlier notebook.

```
list(linear_model.parameters())
```

```
[Parameter containing:
 tensor([[0.6873]], requires_grad=True), Parameter containing:
 tensor([-0.1283], requires_grad=True)]
```

- Earlier, it was our responsibility to create parameters and pass them as the first argument to **`optim.SGD`**.

- Now we can use the parameters method to ask any **`nn.Module`** for a list of parameters owned by it or any of its submodules.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Example

-continuing with the earlier notebook.

```
def training_loop(n_epochs, optimizer, model, loss_fn, t_u_train, t_u_val,
                  t_c_train, t_c_val):
    for epoch in range(1, n_epochs + 1):
        t_p_train = model(t_u_train)
        loss_train = loss_fn(t_p_train, t_c_train)

        t_p_val = model(t_u_val)
        loss_val = loss_fn(t_p_val, t_c_val)

        optimizer.zero_grad()
        loss_train.backward()
        optimizer.step()

        if epoch == 1 or epoch % 1000 == 0:
            print(f"Epoch {epoch}, Training loss {loss_train.item():.4f},"
                  f" Validation loss {loss_val.item():.4f}")
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

-continuing with the earlier notebook.

```
def loss_fn(t_p, t_c):
    squared_diffs = (t_p - t_c)**2
    return squared_diffs.mean()

linear_model = nn.Linear(1, 1) # <1>
optimizer = optim.SGD(linear_model.parameters(), lr=1e-2)

training_loop(
    n_epochs = 3000,
    optimizer = optimizer,
    model = linear_model,
    loss_fn = loss_fn,
    t_u_train = t_un_train,
    t_u_val = t_un_val,
    t_c_train = t_c_train,
    t_c_val = t_c_val)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Example

-continuing with the earlier notebook.

```
print()
print(linear_model.weight)
print(linear_model.bias)
```

```
Epoch 1, Training loss 329.8982, Validation loss 351.4502
Epoch 1000, Training loss 3.4184, Validation loss 2.8660
Epoch 2000, Training loss 2.8855, Validation loss 3.1971
Epoch 3000, Training loss 2.8769, Validation loss 3.2405

Parameter containing:
tensor([[5.4128]], requires_grad=True)
Parameter containing:
tensor([-17.4374], requires_grad=True)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.