Pytorch Problem Set Introduction

Tarushii Goel

October 3, 2021

1 Introduction

Last week, you guys indicated that the pytorch problem set we assigned a few weeks ago was not adequately introduced. The purpose of this lecture is to provide the tools needed to complete the problem set and reintroduce the assignment with an extended due date.

2 Assignment

For the problem set, you will be fitting models to the CIFAR-10 dataset, which is a popular dataset in machine learning. It contains 32x32 pixel images of 10 different classes: plane, car, bird, cat, deer, dog, frog, horse, ship, truck. Here is an example of a deer image in the dataset:

3 Review

To refresh your memory, here is the code that we explained for fitting the sine curve using pytorch:

```python
import numpy as np
import torch
import matplotlib.pyplot as plt

# INITIALIZATION

# 1. Data
x = torch.linspace(-np.pi, np.pi, 2000)
Y= torch.sin(x)

# 2. Model
a=torch.rand((), requires_grad=True);
b=torch.rand((), requires_grad=True);
c=torch.rand((), requires_grad=True);
d=torch.rand((), requires_grad=True)

# 3. Optimizer and Loss Functions
lr = 1e-6

# TRAINING
for i in range(2000):
  # 4. Run Model on Batch
  y_pred = a*x**3 + b*x**2 + c*x + d
  # 5. Compute Loss
  L =  (0.5* (y_pred - Y)**2).sum()
  # 6. Reset Gradients to Zero
  with torch.no_grad():
    a.grad = None
    b.grad =  None
    c.grad =  None
    d.grad =  None
  # 7. Backpropogate (Calculate Gradients)
  L.backward()
  # 8. Update Weights According to Gradients
  with torch.no_grad():
    a -= lr*a.grad
    b -= lr*b.grad
    c -= lr*c.grad
    d -= lr*d.grad
```

How can we use are understanding of pytorch to fit the CIFAR-10 dataset instead of the sine curve? Some details in our code will have to change, but the overall structure will remain the same.

4 Assignment Shell Code

When training models for the CIFAR-10 dataset, you can use this basic shell code:

```
import torch
from torchvision import datasets
from torch import nn, optim
from torch.utils.data import DataLoader
import torchvision.transforms as transforms

# INITIALIZATION

# 1. Data
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
train_ds = datasets.CIFAR10 (root = "data/", train =True , download = True, transform=transform)
val_ds = datasets.CIFAR10 (root ="data/", train =False , download = True, transform=transform)
batch_size=64
train_loader = DataLoader(train_ds, batch_size, shuffle=True)
val_loader = DataLoader(val_ds, batch_size)

# 2. Model
model = nn.Sequential(
    # FILL: model architecture
)

# 3. Optimizer and Loss Functions
optimizer = optim.SGD(model.parameters(), lr= # FILL: learning rate
, momentum=0.9)
criterion = nn.CrossEntropyLoss()

# TRAINING
for epoch in range(15):
  running_loss = 0.0
  for i, data in enumerate(train_loader, 0):
    # 4. Run Model on Batch
    inputs, labels = data
    y_pred = model(inputs)
    # 5. Compute Loss
    L =  criterion(y_pred, labels)
    # 6. Reset Gradients to Zero
    optimizer.zero_grad()
    # 7. Backpropogate (Calculate Gradients)
```

```
41          L.backward()
42          # 8. Update Weights According to Gradients
43          optimizer.step()
44
45          # 9. (extra step) Print the Loss/Accuracy to Track Progress
46          running_loss += L.item()
47          if i % 50 == 49:
48                  print('[%d, %5d] loss: %.3f' %
49                        (epoch + 1, i + 1, running_loss / 2000))
50                  running_loss = 0.0
```

We are taking the same steps as we were with the sine curve (look at the comments in the code), but the content of those steps is different. I will explain those differences now:

4.1 Step 1: Data

4.2 Lines 10-14

Since the CIFAR-10 dataset is quite popular, torchvision (a library specific to computer vision in pytorch) has a class for the dataset. These lines use that class to download the train and validation data. The image data is stored as PIL Images in the **/data** directory. This data cannot be used for training without first converting the PIL Images into pytorch tensors, so we apply image **transforms** that convert the PIL image into a normalized tensor. If you want to query a specific image/label in the dataset, you can do that using index notation: $train\_ds[0][0]$ returns the image and $train\_ds[0][1]$ returns the label for the first image-label pair.

4.3 Lines 15-17

Pytorch has two major data classes: Datasets and Dataloaders. The previous lines created a Dataset, which has the raw information, and these lines create the Dataloader for these Datasets. Dataloaders specify how you want to load the data (e.g. in what batch size, with/without shuffling). The main reason pytorch has this distinction is because you many want to load the same dataset in different ways. Dataloaders still rely on the original Dataset object.

4.4 Steps 2, 4: Model

To build a densely connected neural network , we can stack Flatten, Linear, and Activation layers. All of these layers are implemented in the pytorch library. Here is an example model:

```
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(3072, 500),
    nn.ReLU(),
    nn.Linear(500, 100),
    nn.ReLU(),
    nn.Linear(100, 10),
    nn.Softmax()
)
```

It's important to make sure that the output dimensions of the one layer every layer matches the input dimensions of the next layer in the sequence.

4.5 Steps 3, 5: Loss

Most loss functions are readily available in the $torch.nn$ module. While in the sine curve example, we implemented our own loss function, this was not necessary; we could have just used $torch.nn.MSELoss()$. In this case, we are using cross-entropy loss, which is used frequently when you are predicting class labels. We will not go into the details of how this function works.

4.6 Steps 3, 6, 8: Optimizer

Pytorch also has functions that preform the model optimization via weight updates called **optimizers**. Instead of performing weight updates manually, like we did with the sine curve example, we can rely on the SGD optimizer (one of many different optimizers) to update the weights.

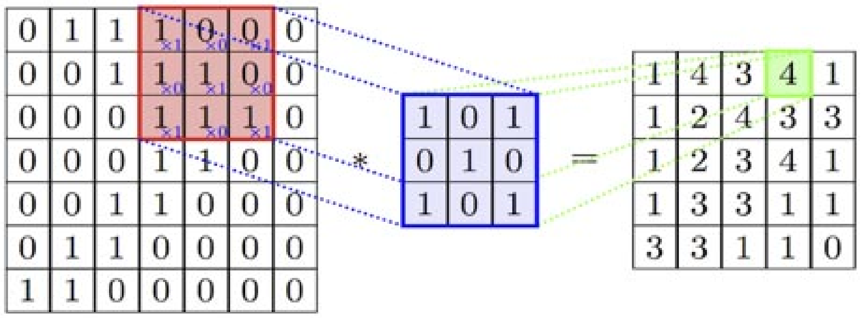
optimizer.zero_grad() resets the gradients to zero, so that the backpropogation step does not add the gradients on top of previously computed gradients.

optimizer.step() updates the weights

5 CNN Review

In the previous section, I explained how to build a regular neural network. Now, we are going to briefly review how to build a model using convolutional neural networks, since the problem set will require usage of CNNs.

5.1 The Convolution



5.2 Convolutional Layers in Pytorch

Fortunately, you don't have to actually remember or understand how the convolution works in order to use it, because the pytorch library implements convolutional layers.

The line $torch.nn.Conv2d(in\_channels, out\_channels, kernel\_size)$ creates a 2D convolutional layer.

6 Final Notes

You will need to reference the pytorch docs while completing the assignment: https://pytorch.org/docs/stable. You will also need to write your own scripts that calculate loss and accuracy, a rudimentary example was provided in step 9, but you will need to expand on it. You will also want to perform visualization of the results: print an image and print the label predicted by your model. See if it makes sense to you! If you have ever want to ask peers or officers questions, we **highly** encourage discussion in the #advanced-problems channel in our club discord (Invite Link: https://discord.gg/zjy3D4e4). Good luck!