CIFAR10_LSTM

October 25, 2019

[166]:
```
import numpy as np
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import time
```

[167]:
```
# set random seeds for reproducibility
torch.manual_seed(12)
torch.cuda.manual_seed(12)
np.random.seed(12)
```

[168]:
```
# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# If we are on a CUDA machine, then this should print a CUDA device, but we are␣
↪not, so it will run on CPU:
print(f'Working on device={device}')
```

```
Working on device=cpu
```

[169]:
```
# Hyper-parameters

# each CIFAR image is RGB 32x32, so it is an 3D array [3,32,32]
# we will flatten the image as vector dim=3*32*32

# we have 10 classes
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

num_classes = 10

num_epochs = 10
batch_size = 1024

learning_rate = 0.001
```

[170]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
```

```python
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transforms.
 ↪ToTensor())

train_loader = torch.utils.data.DataLoader(trainset, batch_size=1024,
                                           shuffle=True)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transforms.
 ↪ToTensor())
test_loader = torch.utils.data.DataLoader(testset,
                                          batch_size=1024,
                                          shuffle=False)
```

```
Files already downloaded and verified
Files already downloaded and verified
```

[171]:
```python
ies = []
image = []
label = []

for i, (images, labels) in enumerate(train_loader):
    ies.append(i)
    image.append(images)
    label.append(labels)
```

[172]:
```python
image[0][0].view(96,32).size()
```

[172]:
```
torch.Size([96, 32])
```

[173]:
```python
class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(LSTMModel, self).__init__()
        # Hidden dimensions
        self.hidden_dim = hidden_dim

        # Number of hidden layers
        self.layer_dim = layer_dim

        # Building your LSTM
        # batch_first=True causes input/output tensors to be of shape
        # (batch_dim, seq_dim, feature_dim)
        self.lstm = nn.LSTM(input_dim, hidden_dim, layer_dim, batch_first=True)

        # Readout layer
        self.fc = nn.Linear(hidden_dim, output_dim)
```

```
    def forward(self, x):
        # Initialize hidden state with zeros
        h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).
↪requires_grad_()

        # Initialize cell state
        c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).
↪requires_grad_()

        # 28 time steps
        # We need to detach as we are doing truncated backpropagation through␣
↪time (BPTT)
        # If we don't, we'll backprop all the way to the start even after going␣
↪through another batch
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))

        # Index hidden state of last time step
        # out.size() --> 100, 28, 100
        # out[:, -1, :] --> 100, 100 --> just want last time step hidden states!
↪
        out = self.fc(out[:, -1, :])
        # out.size() --> 100, 10
        return out
```

[174]:
```
input_dim = 32
hidden_dim = 128
layer_dim = 1
output_dim = 10
```

[175]:
```
model = LSTMModel(input_dim, hidden_dim, layer_dim, output_dim)
```

[176]:
```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

[177]:
```
# Number of steps to unroll
seq_dim = 96
iter = 0



for epoch in range(num_epochs):
    t0 = time.time()
    for i, (images, labels) in enumerate(train_loader):
        # Load images as a torch tensor with gradient accumulation abilities
        epoch_loss = 0
        images = images.view(-1, seq_dim, input_dim).requires_grad_()
        labels = labels.to(device)
```

```python
        # Clear gradients w.r.t. parameters
        optimizer.zero_grad()

        # Forward pass to get output/logits
        # outputs.size() --> 100, 10
        outputs = model(images)

        # Calculate Loss: softmax --> cross entropy loss
        loss = criterion(outputs, labels)
        epoch_loss += loss.item()

        # Getting gradients w.r.t. parameters
        loss.backward()

        # Updating parameters
        optimizer.step()

        iter += 1

        if iter % 49 == 0:
            # Calculate Accuracy
            correct = 0
            total = 0
            # Iterate through test dataset
            for images, labels in test_loader:
                # Resize image
                images = images.view(-1, seq_dim, input_dim)

                # Forward pass only to get logits/output
                outputs = model(images)

                # Get predictions from the maximum value
                _, predicted = torch.max(outputs.data, 1)

                # Total number of labels
                total += labels.size(0)

                # Total correct predictions
                correct += (predicted == labels).sum().item()

            accuracy = correct / total

            # Print Loss
            print(
```

```
        f'Epoch [{epoch+1}/{num_epochs}]], Iter: {iter} Loss: {loss.item():.
↪4f} Test acc: {accuracy: .4f}')
        print('{} seconds'.format(time.time() - t0))
```

```
Epoch [1/10]], Iter: 49 Loss: 2.0424 Test acc:  0.2571
90.77259087562561 seconds
Epoch [2/10]], Iter: 98 Loss: 1.8783 Test acc:  0.3171
103.57928109169006 seconds
Epoch [3/10]], Iter: 147 Loss: 1.8695 Test acc:  0.3493
101.47186303138733 seconds
Epoch [4/10]], Iter: 196 Loss: 1.7231 Test acc:  0.3665
99.46165418624878 seconds
Epoch [5/10]], Iter: 245 Loss: 1.7573 Test acc:  0.3736
100.66864204406738 seconds
Epoch [6/10]], Iter: 294 Loss: 1.6618 Test acc:  0.3871
101.9321620464325 seconds
Epoch [7/10]], Iter: 343 Loss: 1.6320 Test acc:  0.3965
99.57273006439209 seconds
Epoch [8/10]], Iter: 392 Loss: 1.6361 Test acc:  0.4086
96.93101382255554 seconds
Epoch [9/10]], Iter: 441 Loss: 1.6270 Test acc:  0.4058
100.76327204704285 seconds
Epoch [10/10]], Iter: 490 Loss: 1.6105 Test acc:  0.4018
97.01067399978638 seconds
```