21

January 23, 2024

[1]:
```
%pylab inline
import torch
import sys
sys.path.append('..')
sys.path.append('../..')
from data import load
device = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')
print('device = ', device)
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
device =  cuda
```

[2]:
```
class ConvNet(torch.nn.Module):
    class Block(torch.nn.Module):
        def __init__(self, n_input, n_output, stride=1):
            super().__init__()
            self.net = torch.nn.Sequential(
              torch.nn.Conv2d(n_input, n_output, kernel_size=3, padding=1,␣
 ↪stride=stride, bias=False),
              torch.nn.BatchNorm2d(n_output),
              torch.nn.ReLU(),
              torch.nn.Conv2d(n_output, n_output, kernel_size=3, padding=1,␣
 ↪bias=False),
              torch.nn.BatchNorm2d(n_output),
              torch.nn.ReLU()
            )
            self.downsample = None
            if stride != 1 or n_input != n_output:
                self.downsample = torch.nn.Sequential(torch.nn.Conv2d(n_input,␣
 ↪n_output, 1, stride=stride),
                                                      torch.nn.
 ↪BatchNorm2d(n_output))

        def forward(self, x):
            identity = x
```

```
            if self.downsample is not None:
                identity = self.downsample(x)
            return self.net(x) + identity

    def __init__(self, layers=[32,64,128], n_input_channels=3):
        super().__init__()
        L = [torch.nn.Conv2d(n_input_channels, 32, kernel_size=7, padding=3,␣
↪stride=2, bias=False),
             torch.nn.BatchNorm2d(32),
             torch.nn.ReLU(),
             torch.nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
            ]
        c = 32
        for l in layers:
            L.append(self.Block(c, l, stride=2))
            c = l
        self.network = torch.nn.Sequential(*L)
        self.classifier = torch.nn.Linear(c, 1)

    def forward(self, x):
        # Compute the features
        z = self.network(x)
        # Global average pooling
        z = z.mean(dim=[2,3])
        # Classify
        return self.classifier(z)[:,0]

model = ConvNet()
model.train()
print( model.training )
model.eval()
print( model.training )
```

```
True
False
```

[4]:
```
n_epochs = 100

train_data = load.get_dogs_and_cats(resize=(128,128), batch_size=32,␣
↪is_resnet=True)
optimizer = torch.optim.Adam(model.parameters())
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.1)
scheduler = torch.optim.lr_scheduler.MultiStepLR(optimizer, [50,75], gamma=0.1)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'max')

# Transfer the data to a GPU (optional)
model = model.to(device)
```

```python
# Construct the loss and accuracy functions
loss = torch.nn.BCEWithLogitsLoss()
accuracy = lambda o, l: ((o > 0).long() == l.long()).float()

# Train the network
for epoch in range(n_epochs):
    model.train()
    accuracies = []
    for it, (data, label) in enumerate(train_data):
        # Transfer the data to a GPU (optional)
        data, label = data.to(device), label.to(device)

        # Produce the output
        o = model(data)

        # Compute the loss and accuracy
        loss_val = loss(o, label.float())
        accuracies.extend(accuracy(o, label).detach().cpu().numpy())

        # Take a gradient step
        optimizer.zero_grad()
        loss_val.backward()
        optimizer.step()
        break
#     scheduler.step()
    scheduler.step(np.mean(accuracies))
    print( 'epoch = ', epoch, 'optimizer_lr', optimizer.param_groups[0]['lr'],␣
 ↪'accuracy', np.mean(accuracies))
```

```
epoch =  0 optimizer_lr 0.001 accuracy 0.46875
epoch =  1 optimizer_lr 0.001 accuracy 0.65625
epoch =  2 optimizer_lr 0.001 accuracy 0.96875
epoch =  3 optimizer_lr 0.001 accuracy 0.96875
epoch =  4 optimizer_lr 0.001 accuracy 1.0
epoch =  5 optimizer_lr 0.001 accuracy 1.0
epoch =  6 optimizer_lr 0.001 accuracy 1.0
epoch =  7 optimizer_lr 0.001 accuracy 1.0
epoch =  8 optimizer_lr 0.001 accuracy 1.0
epoch =  9 optimizer_lr 0.001 accuracy 1.0
epoch =  10 optimizer_lr 0.001 accuracy 1.0
epoch =  11 optimizer_lr 0.001 accuracy 1.0
epoch =  12 optimizer_lr 0.001 accuracy 1.0
epoch =  13 optimizer_lr 0.001 accuracy 1.0
epoch =  14 optimizer_lr 0.001 accuracy 1.0
epoch =  15 optimizer_lr 0.0001 accuracy 1.0
epoch =  16 optimizer_lr 0.0001 accuracy 1.0
```

```
epoch =  17 optimizer_lr 0.0001 accuracy 1.0
epoch =  18 optimizer_lr 0.0001 accuracy 1.0
epoch =  19 optimizer_lr 0.0001 accuracy 1.0
epoch =  20 optimizer_lr 0.0001 accuracy 1.0
epoch =  21 optimizer_lr 0.0001 accuracy 1.0
epoch =  22 optimizer_lr 0.0001 accuracy 1.0
epoch =  23 optimizer_lr 0.0001 accuracy 1.0
epoch =  24 optimizer_lr 0.0001 accuracy 1.0
epoch =  25 optimizer_lr 0.0001 accuracy 1.0
epoch =  26 optimizer_lr 1e-05 accuracy 1.0
epoch =  27 optimizer_lr 1e-05 accuracy 1.0
epoch =  28 optimizer_lr 1e-05 accuracy 1.0
epoch =  29 optimizer_lr 1e-05 accuracy 1.0
epoch =  30 optimizer_lr 1e-05 accuracy 1.0
epoch =  31 optimizer_lr 1e-05 accuracy 1.0
epoch =  32 optimizer_lr 1e-05 accuracy 1.0
epoch =  33 optimizer_lr 1e-05 accuracy 1.0
epoch =  34 optimizer_lr 1e-05 accuracy 1.0
epoch =  35 optimizer_lr 1e-05 accuracy 1.0
epoch =  36 optimizer_lr 1e-05 accuracy 1.0
epoch =  37 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  38 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  39 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  40 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  41 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  42 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  43 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  44 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  45 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  46 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  47 optimizer_lr 1.0000000000000002e-06 accuracy 1.0
epoch =  48 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  49 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  50 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  51 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  52 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  53 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  54 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  55 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  56 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  57 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  58 optimizer_lr 1.0000000000000002e-07 accuracy 1.0
epoch =  59 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  60 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  61 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  62 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  63 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  64 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
```

```
epoch =  65 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  66 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  67 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  68 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  69 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  70 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  71 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  72 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  73 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  74 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  75 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  76 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  77 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  78 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  79 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  80 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  81 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  82 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  83 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  84 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  85 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  86 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  87 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  88 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  89 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  90 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  91 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  92 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  93 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  94 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  95 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  96 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  97 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  98 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
epoch =  99 optimizer_lr 1.0000000000000004e-08 accuracy 1.0
```

[ ]: