classwork

October 7, 2024

[ ]:
```python
import torch
import torch.nn as nn

from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torch.utils.data import TensorDataset
```

[ ]:
```python
torch.__version__
```

[ ]:
```
'2.4.0+cu118'
```

[ ]:
```python
torch.manual_seed(42)
```

[ ]:
```
<torch._C.Generator at 0x281abee4710>
```

[ ]:
```python
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

[ ]:
```
device(type='cuda', index=0)
```

[ ]:
```python
map_courses = {0: 'DSAI', 1: 'IM', 2: 'WEM', 3: 'GIS', 4: 'RGLS', 5: 'CS'}
```

[ ]:
```python
import numpy as np

x_train = np.array([[400, 0, 28, 10],
                    [100, 1, 26, 8],
                    [200, 0, 29, 2],
                    [700, 2, 23, 5],
                    [900, 3, 27, 9],
                    [800, 4, 40, 5],
                    [500, 5, 29, 7],
                    [600, 0, 35, 10]], dtype='float32')

y_train = np.array([3.9, 3.7, 2.4, 2.9, 3.9, 3.0, 3.0, 3.1], dtype='float32')

inputs = torch.from_numpy(x_train)
targets = torch.from_numpy(y_train)
print(inputs.size())
```

```python
print(targets.size())
```

```
torch.Size([8, 4])
torch.Size([8])
```

[ ]:
```python
train_ds = TensorDataset(inputs, targets)
train_ds[0:3]
```

[ ]:
```
(tensor([[400.,   0.,  28.,  10.],
         [100.,   1.,  26.,   8.],
         [200.,   0.,  29.,   2.]]),
 tensor([3.9000, 3.7000, 2.4000]))
```

[ ]:
```python
batch_size = 3
train_dl = DataLoader(train_ds, batch_size, shuffle=True)
```

[ ]:
```python
for xb, yb in train_dl:
    print(xb)
    print(yb)
    break
```

```
tensor([[400.,   0.,  28.,  10.],
        [800.,   4.,  40.,   5.],
        [100.,   1.,  26.,   8.]])
tensor([3.9000, 3.0000, 3.7000])
```

[ ]:
```python
class Custom(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Linear(4, 1)

    def forward(self, x):
        x = self.fc(x)
        return x
```

[ ]:
```python
model = Custom()
print(model.fc.weight)
print(model.fc.weight.size()) # (out_features, in_features)
print(model.fc.bias)
print(model.fc.bias.size())
```

```
Parameter containing:
tensor([[ 0.0472, -0.4938,  0.4516, -0.4247]], requires_grad=True)
torch.Size([1, 4])
Parameter containing:
tensor([0.3860], requires_grad=True)
torch.Size([1])
```

[ ]:
```python
list(model.parameters())
```

[ ]: [Parameter containing:
 tensor([[0.0472, -0.4938, 0.4516, -0.4247]], requires_grad=True),
 Parameter containing:
 tensor([0.3860], requires_grad=True)]

[ ]:
```
print(sum(p.numel() for p in model.parameters() if p.requires_grad))
```

5

[ ]:
```
preds = model(inputs)
preds
```

[ ]:
```
tensor([[27.6588],
        [12.9539],
        [22.0699],
        [40.6945],
        [49.7463],
        [52.1024],
        [31.6345],
        [40.2580]], grad_fn=<AddmmBackward0>)
```

[ ]:
```
criterion_mse = nn.MSELoss()
criterion_softmax_cross_entropy_loss = nn.CrossEntropyLoss()
```

[ ]:
```
mse = criterion_mse(preds, targets)
print(mse)
print(mse.item())
```

```
tensor(1147.2864, grad_fn=<MseLossBackward0>)
1147.286376953125
```

```
d:\DataScience\Anaconda3\envs\dl4cv\lib\site-
packages\torch\nn\modules\loss.py:538: UserWarning: Using a target size
(torch.Size([8])) that is different to the input size (torch.Size([8, 1])). This
will likely lead to incorrect results due to broadcasting. Please ensure they
have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

[ ]:
```
opt = torch.optim.Adam(model.parameters(), lr=0.001)
```

[ ]:
```
import sys

# Utility function to train the model
def fit(num_epochs, model, loss_fn, opt, train_dl):

    # Repeat for given number of epochs
    for epoch in range(num_epochs):

        # Train with batches of data
```

```
        for xb,yb in train_dl:

            xb.to(device) #move them to gpu if possible, if not, it will be cpu
            yb.to(device)

            # 1. Predict
            pred = model(xb)

            # 2. Calculate loss
            loss = loss_fn(pred, yb)

            # 3. Calculate gradient
            opt.zero_grad()  #if not, the gradients will accumulate
            loss.backward()

            # Print out the gradients.
            #print ('dL/dw: ', model.weight.grad)
            #print ('dL/db: ', model.bias.grad)

            # 4. Update parameters using gradients
            opt.step()

        # Print the progress
        if (epoch+1) % 10 == 0:
            sys.stdout.write("\rEpoch [{}/{}], Loss: {:.4f}".format(epoch+1,␣
↪num_epochs, loss.item()))
```

[ ]:
```
#train for 100 epochs
fit(100, model, criterion_mse, opt, train_dl)
```

```
Epoch [100/100], Loss: 0.0628
```

[ ]:
```
preds = model(inputs)
loss = criterion_mse(preds, targets)
print(loss.item())
```

```
0.35663479566574097
```

[ ]:
```
preds = model(inputs)
preds
```

[ ]:
```
tensor([[3.3230],
        [3.1469],
        [2.6553],
        [2.6472],
        [3.1514],
        [3.3107],
        [3.1696],
```

```
         [3.5440]], grad_fn=<AddmmBackward0>)
```

[ ]:
```python
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(x_train, y_train)

preds2 = model.predict(x_train)
preds2
```

[ ]:
```
array([3.696323 , 3.4681077, 2.4174857, 3.0752082, 3.63602  , 2.733765 ,
       3.347672 , 3.5254207], dtype=float32)
```

[ ]:
```python
from sklearn.metrics import mean_squared_error, r2_score

print('MSE: ', mean_squared_error(preds.detach().numpy(), preds2))
print('R2_score: ', r2_score(preds.detach().numpy(), preds2))
```

```
MSE:  0.1352608
R2_score:  -0.5435817138557375
```

We are getting Linear Regression model better because of number of training samples. Less training sample - classical ml is better. Deep learning benefits from a lot of data. Our training sample is just 8 examples, thats why Linear regression results are better

MSE: 0.135 R2_score is really bad, it is worser than mean (r2_score(mean) = 0)