인공지능 개론

L04.1 Softmax Regression with Pytorch

국민대학교

소프트웨어융합대학원

박하명

Contents

- **Pytorch로 Softmax Regression 구현**
- 조금 더 깔끔하게 Softmax

Softmax Regression

- 학습 데이터 생성

```
import torch

x_train = torch.FloatTensor([ [1,2,1,1], [2,1,3,2], [3,1,3,4], [4,1,5,5], [1,7,5,5],
                              [1,2,5,6], [1,6,6,6], [1,7,7,7] ])
y_train = torch.FloatTensor([ [0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0],
                              [1,0,0], [1,0,0] ])
```

Softmax Regression

- W, b 초기화
- Optimizer 생성

```
W = torch.zeros(4, 3, requires_grad=True)
b = torch.zeros(1, 3, requires_grad=True)

optimizer = torch.optim.Adam([W,b], lr=0.1)
```

Softmax Regression

- 반복횟수 설정
- 가설 및 비용 설정

$$H(\mathbf{x}) = S(\mathbf{x}^{\mathsf{T}}\mathbf{w} + b)$$

```
for epoch in range(3001):
    hypothesis = torch.softmax(torch.mm(x_train, W)+b, dim=1)
    cost = -torch.mean(torch.sum(y_train * torch.log(hypothesis), dim=1))
```

```
    hypothesis = (torch.mm(x_train, W)+b).softmax(dim=1)
    cost = -(y_train * torch.log(hypothesis)).sum(dim=1).mean()
```

$$Cost(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{m} C(y_i, H(x_i)) \qquad C(y, \hat{y}) = -\sum_{j=1}^{d} y_j \log \hat{y}_j$$

Softmax Regression

- Optimizer를 이용한 경사 계산 및 W, b 업데이트

```
optimizer.zero_grad()
cost.backward()
optimizer.step()
```

Softmax Regression

- 학습이 잘 되는지 확인하기위한 내용 출력

```
if epoch % 300 == 0:
  print("epoch: {}, cost: {:.6f}".format(epoch, cost.item()))
```

Softmax Regression

- 학습 결과 확인

```
epoch: 0, cost: 1.098612
epoch: 300, cost: 0.105263
epoch: 600, cost: 0.042634
epoch: 900, cost: 0.023111
epoch: 1200, cost: 0.014479
epoch: 1500, cost: 0.009879
epoch: 1800, cost: 0.007124
epoch: 2100, cost: 0.005338
epoch: 2400, cost: 0.004113
epoch: 2700, cost: 0.003236
epoch: 3000, cost: 0.002588
```

Softmax Regression

- x 가 [1,11,10,9], [1,3,4,3], [1,1,0,1] 일 때, y값은?

```
W.requires_grad_(False)
b.requires_grad_(False)

x_test = torch.FloatTensor([[1,11,10,9], [1,3,4,3], [1,1,0,1]])
test_all = torch.softmax(torch.mm(x_test, W)+b, dim=1)
print(test_all)
print(torch.argmax(test_all, dim=1))
```

Contents

- Pytorch로 Softmax Regression 구현
- **조금 더 깔끔하게 Softmax**

조금 더 깔끔하게 Softmax

마음에 안드는 부분 1. [1,0,0], [0,1,0], [0,0,1] 대신 0, 1, 2를 쓰면 안되나?

```
y_train = torch.FloatTensor([ [0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0],
                              [1,0,0], [1,0,0] ])
```

마음에 안드는 부분 2. 이렇게 복잡한 함수를 항상 직접 구현해야하나? 어차피 softmax, cross entropy인데?

```
hypothesis = torch.softmax(torch.mm(x_train, W)+b, dim=1)
cost = -torch.mean(torch.sum(y_train * torch.log(hypothesis), dim=1))
```

pytorch가 제공하는 cross_entropy 함수를 활용하면 해결!

조금 더 깔끔하게 Softmax

torch.nn.functional을 F라는 이름으로 사용하겠다고 선언

```
import torch.nn.functional as F
```

y_train 수정

```
y_train = torch.FloatTensor([ [0,0,1], [0,0,1], [0,0,1], [0,1,0], [0,1,0], [0,1,0],
                              [1,0,0], [1,0,0] ])
```

```
y_train = torch.LongTensor([2,2,2,1,1,1,0,0])
```

조금 더 깔끔하게 Softmax

```
import torch.nn.functional as F
```

가설, 비용 수정

```
hypothesis = torch.softmax(torch.mm(x_train, W)+b, dim=1)
cost = -torch.mean(torch.sum(y_train * torch.log(hypothesis), dim=1))
```

```
z = torch.mm(x_train, W)+b
cost = F.cross_entropy(z, y_train)
```

주의! F.cross_entropy 는 softmax와 cross entropy를 합친 것.

조금 더 깔끔하게 Softmax

어차피 맨날 쓰는 W와 b. nn.Linear로 대체하자!

nn.Linear: $\mathbf{x}^T\mathbf{w}+\mathbf{b}$ 를 간단히 표현하는 방법

```
import torch.nn as nn
…
model = nn.Linear(4,3)
optimizer = torch.optim.Adam(model.parameters(),lr=1)
```

```
    z = torch.mm(x_train, W)+b
```

```
    z = model(x_train)
```

Question?