XOR과 활성화함수

-모두를 위한 딥러닝 시즌2

한신대학교 AINC Lab
노진산

2024.11.27(수)

목 차

- XOR
- XOR_nn
- XOR_WIDE_DEEP
- 소프트맥스 함수
- ReLU 함수

XOR

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

In [1]:
```
# Lab 9 XOR
import torch
```

In [2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

In [3]:
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

In [4]:
```
# nn layers
linear = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
```

1개의 선형계층

In [5]:
```
# model
model = torch.nn.Sequential(linear, sigmoid).to(device)
```

In [6]:
```
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)
```

In [7]:
```
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)

    # cost/loss function
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0:
        print(step, cost.item())
```


https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-08_1_xor.ipynb

XOR

```
0 0.7273974418640137
100 0.6931475400924683
200 0.6931471824645996
300 0.6931471824645996
400 0.6931471824645996
500 0.6931471824645996
600 0.6931471824645996
700 0.6931471824645996
800 0.6931471824645996
900 0.6931471824645996
1000 0.6931471824645996
1100 0.6931471824645996
1200 0.6931471824645996
1300 0.6931471824645996
1400 0.6931471824645996
1500 0.6931471824645996
1600 0.6931471824645996
1700 0.6931471824645996
1800 0.6931471824645996
1900 0.6931471824645996
2000 0.6931471824645996
2100 0.6931471824645996
2200 0.6931471824645996
2300 0.6931471824645996
2400 0.6931471824645996
2500 0.6931471824645996
2600 0.6931471824645996
2700 0.6931471824645996
2800 0.6931471824645996
2900 0.6931471824645996
3000 0.6931471824645996
```

https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-08_1_xor.ipynb

0.693147, 이것은 -log(0.5)에 해당
즉, 모델이 학습하지 못하고 있으며, 출력값이 변하지 않는 상태

XOR_nn

In [1]:
```
# Lab 9 XOR
import torch
```

In [2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

In [3]:
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

In [4]:
```
# nn layers
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
```

두개의 선형계층으로 바뀐 모습

In [5]:
```
# model
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid).to(device)
```

In [6]:
```
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)  # modified learning rate from 0.1 to 1
```

In [7]:
```
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)

    # cost/loss function
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0:
        print(step, cost.item())
```

https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-08_2_xor_nn.ipynb

XOR_nn

```
0 0.7434073090553284
100 0.6931650638580322
200 0.6931577920913696
300 0.6931517124176025
400 0.6931463479995728
500 0.6931411027908325
600 0.693135678768158
700 0.6931295394897461
800 0.693122148513794
900 0.6931126713752747
1000 0.6930999755859375
1100 0.693082332611084
1200 0.6930568814277649
1300 0.6930190920829773
1400 0.6929606199264526
1500 0.6928659677505493
1600 0.6927032470703125
1700 0.6923960447311401
1800 0.6917301416397095
1900 0.6899654865264893
2000 0.6838318109512329
2100 0.6561676263809204
2200 0.4311096668243408
2300 0.1348954439163208
2400 0.0663050040602684
2500 0.04216844588518143
2600 0.03045402094721794
2700 0.02366602048277855
2800 0.019277796149253845
2900 0.01622406765818596
3000 0.013983823359012604
3100 0.012273991480469704
3200 0.010928178206086159
3300 0.009842487052083015
3400 0.008949032984673977
3500 0.008201336488127708
3600 0.007566767744719982
3700 0.007021686062216759
3800 0.006548595614731312
3900 0.006134253926575184
4000 0.005768374539911747
```

```
7600 0.0018075400730594993
7700 0.0017733527347445488
7800 0.0017404207028448582
7900 0.0017087138257920742
8000 0.00167809752747416 5
8100 0.0016485570231452584
8200 0.00162000255659 2226
8300 0.0015924491453915834
8400 0.0015657917829230428
8500 0.0015400308184325695
8600 0.0015150615945458412
8700 0.0014909136807546 02
8800 0.0014674977865070105
8900 0.0014444813678972423
9000 0.0014228166546672583
9100 0.0014014765620231628
9200 0.0013806892093271017
9300 0.0013606036081910133
9400 0.0013410557294264436
9500 0.0013220303226262 33
9600 0.0013035575393587 35
9700 0.0012856370303779 84
9800 0.0012681199004873633
9900 0.0012511102249845862
10000 0.0012345188297331333
```

https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-08_2_xor_nn.ipynb

XOR_WIDE_DEEP

In [1]:
```
# Lab 9 XOR
import torch
```

In [2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

In [3]:
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

In [4]:
```
# nn layers
linear1 = torch.nn.Linear(2, 10, bias=True)
linear2 = torch.nn.Linear(10, 10, bias=True)
linear3 = torch.nn.Linear(10, 10, bias=True)
linear4 = torch.nn.Linear(10, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
```

4개의 선형 계층

In [5]:
```
# model
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid, linear3, sigmoid, linear4, sigmoid).to(device)
```

In [6]:
```
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)  # modified learning rate from 0.1 to 1
```

In [7]:
```
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)

    # cost/loss function
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0:
        print(step, cost.item())
```

https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-08_3_xor_nn_wide_deep.ipynb

XOR_WIDE_DEEP

```
0 0.6948983669281006
100 0.6931558847427368
200 0.6931535005569458
300 0.6931513547897339
400 0.6931493282318115
500 0.6931473016738892
600 0.6931453943252563
700 0.6931434869766235
800 0.6931416988372803
900 0.6931397914886475
1000 0.6931380033493042
1100 0.6931362152099609
1200 0.6931343078613281
1300 0.6931324005126953
1400 0.6931304931640625
1500 0.6931284666061401
1600 0.6931264400482178
1700 0.6931242942810059
1800 0.6931220293045044
1900 0.6931196451187134
2000 0.6931171417236328
2100 0.6931145191192627
2200 0.6931115984916687
2300 0.6931085586547852
2400 0.693105161190033
2500 0.6931014657020569
2600 0.6930974721908569
2700 0.6930930018424988
2800 0.6930880546569824
2900 0.6930825710296631
3000 0.6930763125419617
```

```
7000 0.0004836336011067033
7100 0.0004537721397355199
7200 0.0004272061923984438
7300 0.00040348825859837234
7400 0.00038214115193113685
7500 0.00036286652903072536
7600 0.00034532143035903573
7700 0.00032935672788880765
7800 0.00031471892725676 3
7900 0.00030131853418424726
8000 0.0002889616880565882
8100 0.0002774993481580168
8200 0.0002669314562808722
8300 0.0002570493088569492
8400 0.0002478678361512720 6
8500 0.0002393123868387192 5
8600 0.0002312936267117038 4
8700 0.000223766764975152 9
8800 0.0002167019993066787 7
8900 0.0002100546262226998 8
9000 0.000203779898583889
9100 0.0001978629152290523
9200 0.0001922291266964748 5
9300 0.0001869381812866777 2
9400 0.0001819155295379459 9
9500 0.0001771611860021948 8
9600 0.0001726155169308185 6
9700 0.0001682934234850108 6
9800 0.0001641501876292750 2
9900 0.0001602156116859987 4
10000 0.0001565046259202063
```

1개, 2개의 선형 계층일 때보다 xor문제를 정확하게 해결하는 모습

https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-08_3_xor_nn_wide_deep.ipynb

소프트맥스 함수

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

출력값의 범위는 0~1 사이

$$\frac{A}{A+B+C}$$

출력값의 합은 항상 1.

```
hypothesis = F.softmax(z, dim=0)
print(hypothesis)
```

```
tensor([0.0900, 0.2447, 0.6652])
```

https://github.com/deeplearningzerotoall/PyTorch/blob/master/lab-06_1_softmax_classification.ipynb

첫번째 클래스 약 9퍼센트,
두번째 클래스 약 24퍼센트,
세번째 클래스 약 66퍼센트.

ReLU 함수

$$f(x) = \max(0, x)$$

```python
import torch
import torch.nn as nn

relu = nn.ReLU()
x = torch.tensor([-2.0, -1.0, 0.0, 1.0, 2.0])
output = relu(x)
print(output)  # Output: tensor([0., 0., 0., 1., 2.])
```

코드 복사

죽은 ReLU 문제
음수의 값 학습X.

Leaky ReLU 함수, PReLU와 같은 변형 함수 등장.