XOR problem

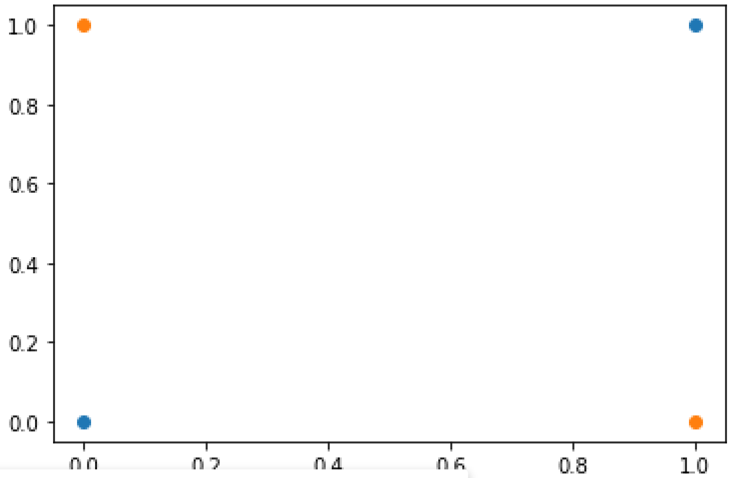
```
import torch
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.optim as optim
```

Create XOR data

```
if torch.cuda.is_available():
    device = 'cuda'
else:
    device = 'cpu'
```

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

```
for label in [0, 1]:
    sub_d = X[(y==label)[:,0], :]
    plt.plot(sub_d[:, 0], sub_d[:, 1], 'o')
```



저장이 완료되었습니다.

Create logistic regression model and training

tip) torch.nn.Sequential(*args) 을 활용하면 여러 layer를 linear하게 연결할 수 있습니다

```
class linear_model(nn.Module):
    def __init__(self):
        super().__init__()
#           self.linear = nn.Linear(2, 1, bias=True)
#         self.sigmod = torch.nn.Sigmod()
        self.layer = nn.Sequential(
                                nn.Linear(2, 1, bias=True),
                                nn.Sigmoid())

    def forward(self, x):
#         out = self.linear(x)
#         out = self.sigmod(input_layer)
        out = self.layer(x)

        return out
```

```
model = linear_model()
```

```
epochs = 10000
learning_rate = 1
```

```
criterion = nn.BCELoss().to(device)
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

for step in range(epochs+1):
    optimizer.zero_grad()
    hypothesis = model(X)

    cost = criterion(hypothesis, y)
    cost.backward()
    optimizer.step()

    if step % 1000 == 0:
        print(step, cost.item())

```

```
0 0.7110074162483215
1000 0.6931471824645996
2000 0.6931471824645996
3000 0.6931471824645996
4000 0.6931471824645996
5000 0.6931471824645996
6000 0.6931471824645996
```

저장이 완료되었습니다. ×

```
10000 0.6931471824645996
```

결과 확인

```
model(X)
```

```
tensor([[0.5000],
        [0.5000],
        [0.5000],
        [0.5000]], grad_fn=<SigmoidBackward>)
```

```
with torch.no_grad():
    hypothesis = model(X)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == y).float().mean()
    print('₩nHypothesis: ', hypothesis.detach().cpu().numpy(), '₩nCorrect: ',
```

```
Hypothesis:  [[0.5]
 [0.5]
 [0.5]
 [0.5]]
Correct:  [[0.]
 [0.]
 [0.]
 [0.]]
Accuracy:  0.5
```

Create multilayer perceptron and training

```
class multi_layer_model(nn.Module):
    def __init__(self):
        super().__init__()

        self.layer = nn.Sequential(
                                nn.Linear(2, 2, bias=True),
                                nn.Sigmoid(),
                                nn.Linear(2, 1, bias=True),
                                nn.Sigmoid())

    def forward(self, x):
        out = self.layer(x)

        return out
```

```
model = multi_layer_model()
```

저장이 완료되었습니다. ✕

```
criterion = nn.BCELoss().to(device)
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

for step in range(epochs+1):
    optimizer.zero_grad()
    hypothesis = model(X)

    cost = criterion(hypothesis, y)
    cost.backward()
    optimizer.step()

    if step % 1000 == 0:
        print(step, cost.item())

```

```
0 0.6997400522232056
1000 0.03586730733513832
2000 0.007624122779816389
3000 0.004210012499243021
4000 0.0028997245244681835
5000 0.0022089118137955666
6000 0.0017828410491347313
7000 0.0014940585242584348
8000 0.0012854698579758406
9000 0.001127827214077115
10000 0.0010044733062386513
```

```
model(X)
```

```
tensor([[9.6269e-04],
        [9.9911e-01],
        [9.9868e-01],
        [8.3550e-04]], grad_fn=<SigmoidBackward>)
```

```
with torch.no_grad():
    hypothesis = model(X)
    predicted = (hypothesis > 0.5).float()
    accuracy = (predicted == y).float().mean()
    print('₩nHypothesis: ', hypothesis.detach().cpu().numpy(), '₩nCorrect: ',
```

```
Hypothesis:  [[9.6268614e-04]
 [9.9910718e-01]
 [9.9867564e-01]
 [8.3550456e-04]]
Correct:  [[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:  1.0
```

저장이 완료되었습니다. ×

1분 5초 오전 11:39에 완료됨

저장이 완료되었습니다.