

Document Classification on MLP

MLP LAB 임경태



Contents

01 Multi-Layer Perceptron

02 ReLU

03 Document Classification

04 NLI with MLP

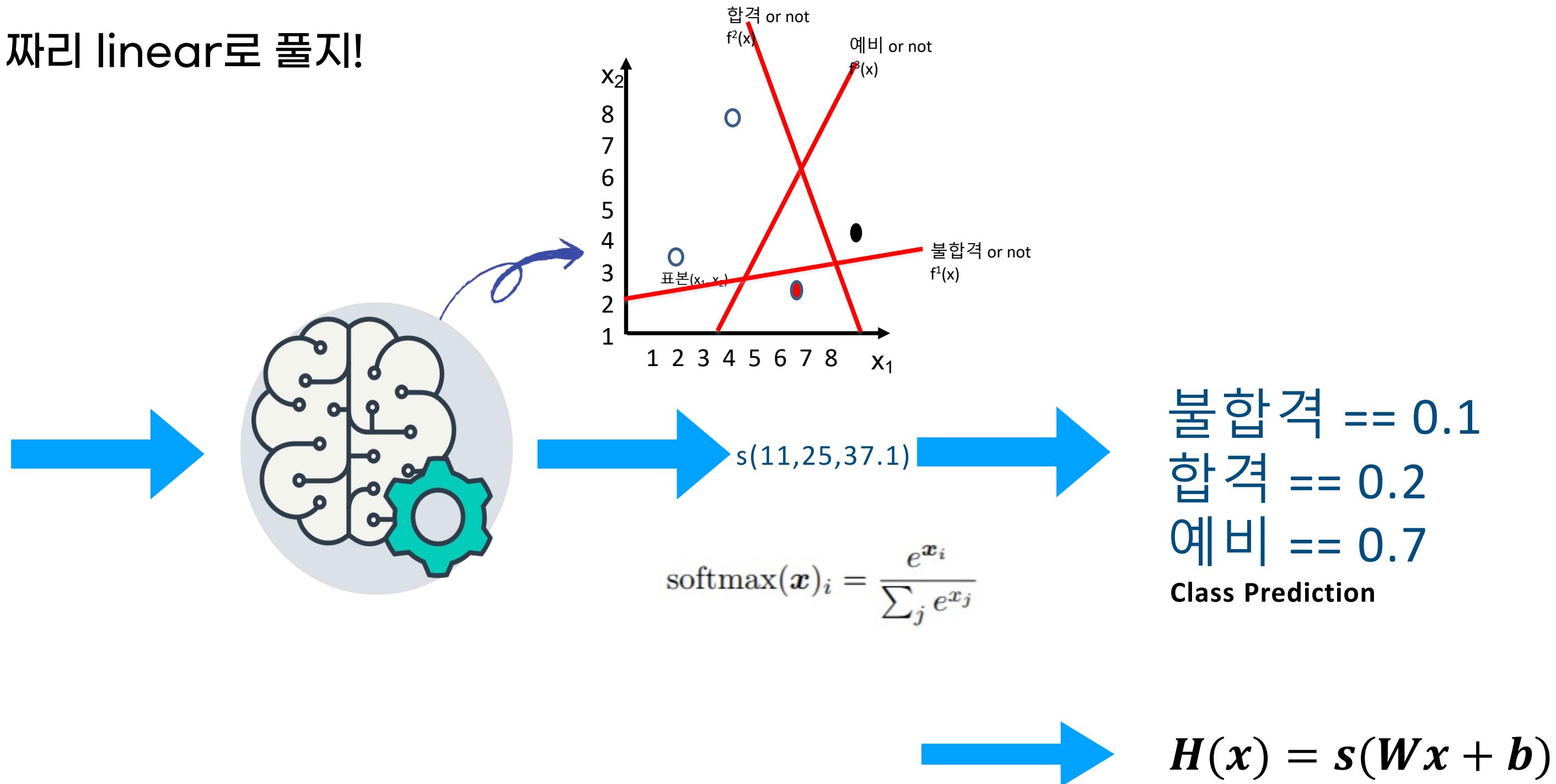


01. Multi-Layer Perceptron

Multi-class Classification의 고찰

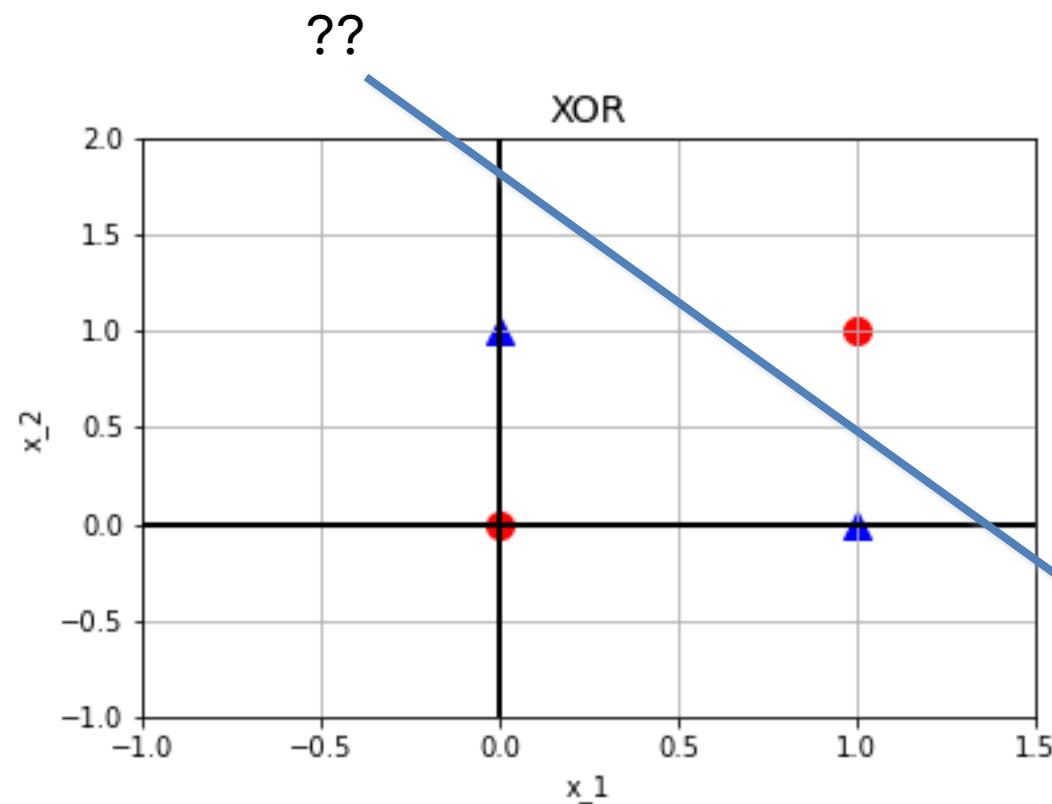
- N개의 Multi-class의 classification의 경우 어떻게 푸나?
- Linear N개 짜리 linear로 풀지!

Quiz1: 4
Quiz2: 7
Quiz Scores



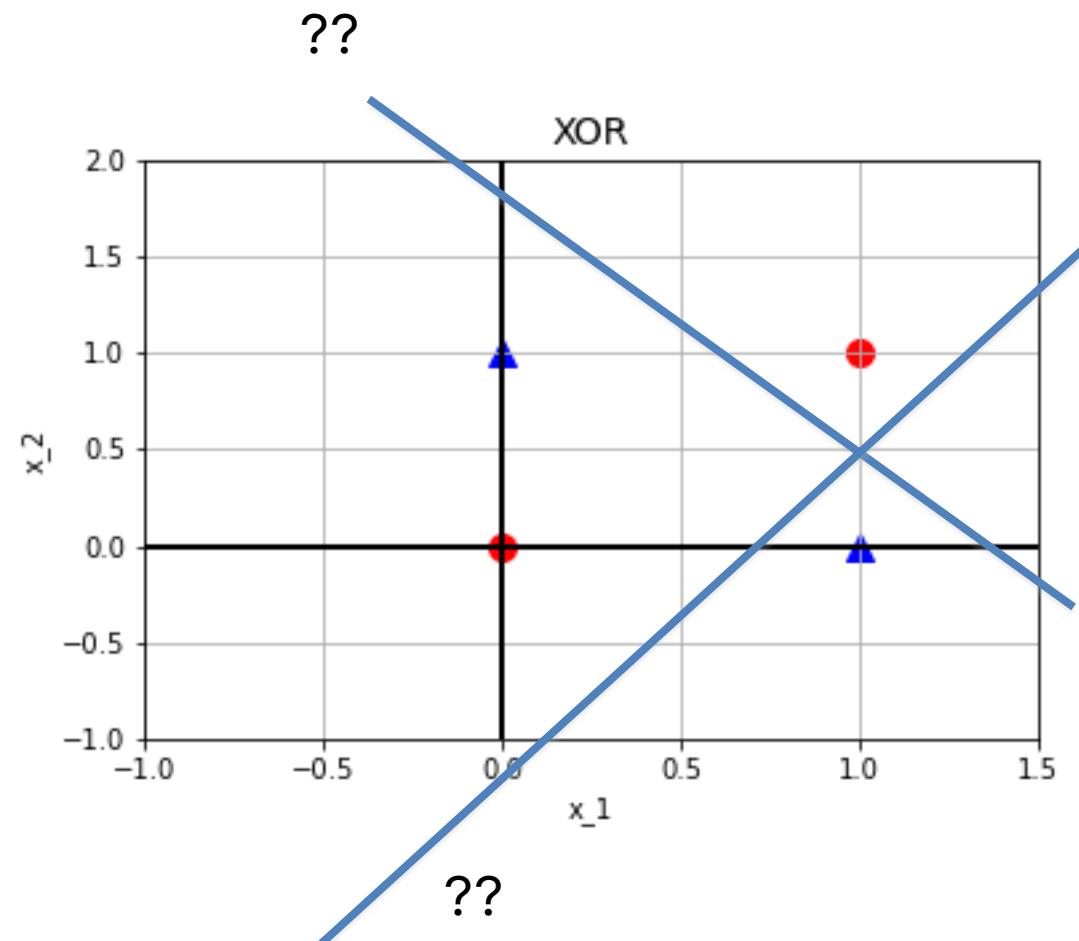
의문점

- N개의 Multi-class의 classification의 경우 어떻게 푸나?
 - Linear N개 짜리 linear로 풀지!
 - Linear로 구분하지 못하는 데이터는 어떻게 함??



의문점 (구현)

- N개의 Multi-class의 classification의 경우 어떻게 푸나?
- Linear N개 짜리 linear로 풀지!
- Linear로 구분하지 못하는 데이터는 어떻게 함??



```

X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
# nn Layers
linear = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear, sigmoid).to(device)
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(),
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())

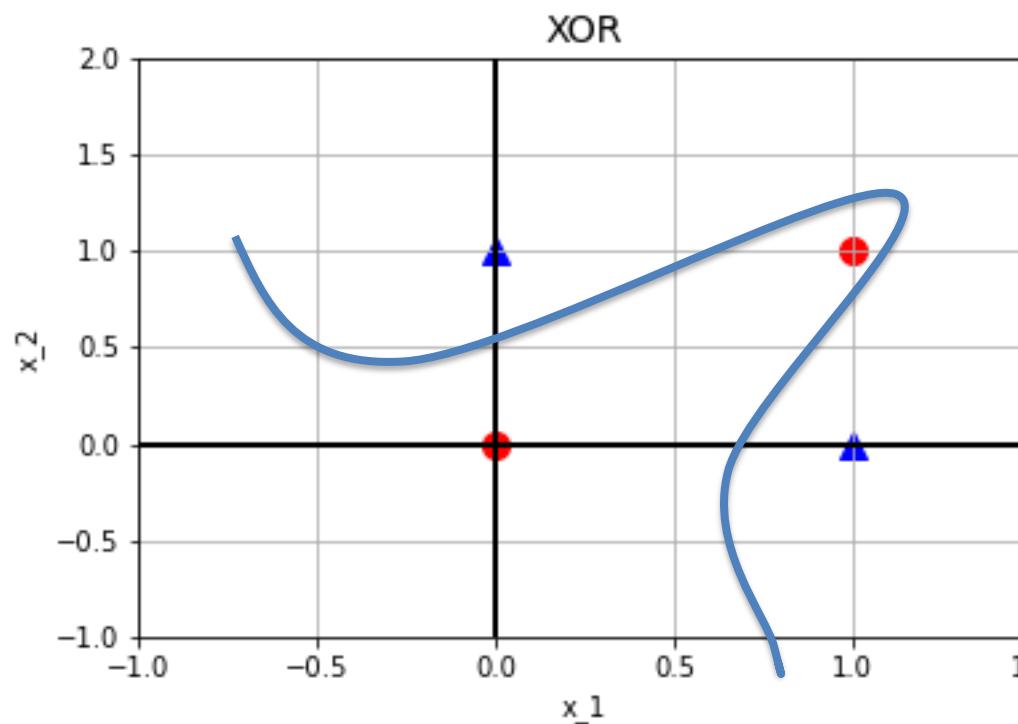
```

0 0.7273974418640137
100 0.6931475400924683
200 0.6931471824645996
300 0.6931471824645996
...
9800 0.6931471824645996
9900 0.6931471824645996
10000 0.6931471824645996

Hypothesis: [[0.5]
[0.5]
[0.5]
[0.5]]
Correct: [[0.]
[1.]
[1.]
[0.]]
Accuracy: 0.5

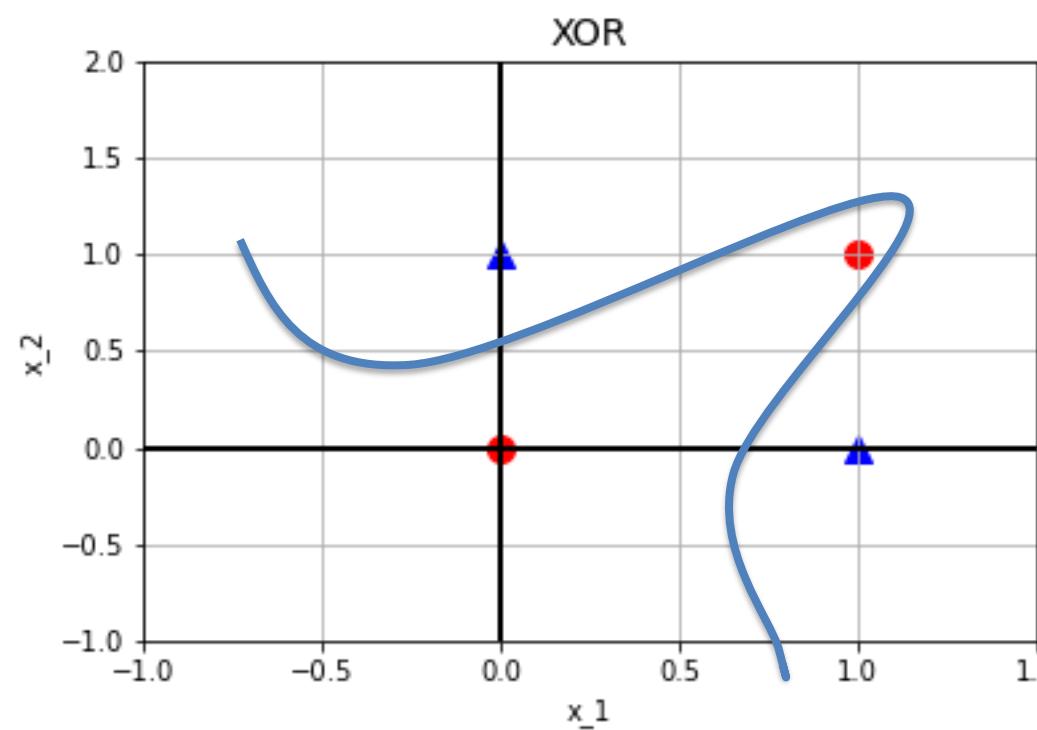
해결책

- N개의 Multi-class의 classification의 경우 어떻게 푸냐?
 - Linear N개 짜리 linear로 풀지!
- Linear로 구분하지 못하는 데이터는 어떻게 함?? → 구부려



해결책 (구현)

- N개의 Multi-class의 classification의 경우 어떻게 푸나?
 - Linear N개 짜리 linear로 풀지!
- Linear로 구분하지 못하는 데이터는 어떻게 함?? → 구부려 → 2차원 이상 함수로.. (sigmoid)



```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
# nn layers
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid).to(device)
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(),
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())

```

0 0.7434073090553284
100 0.6931650638580322
200 0.6931577920913696
300 0.6931517124176025
...
9800 0.0012681199004873633
9900 0.0012511102249845862
10000 0.0012345188297331333

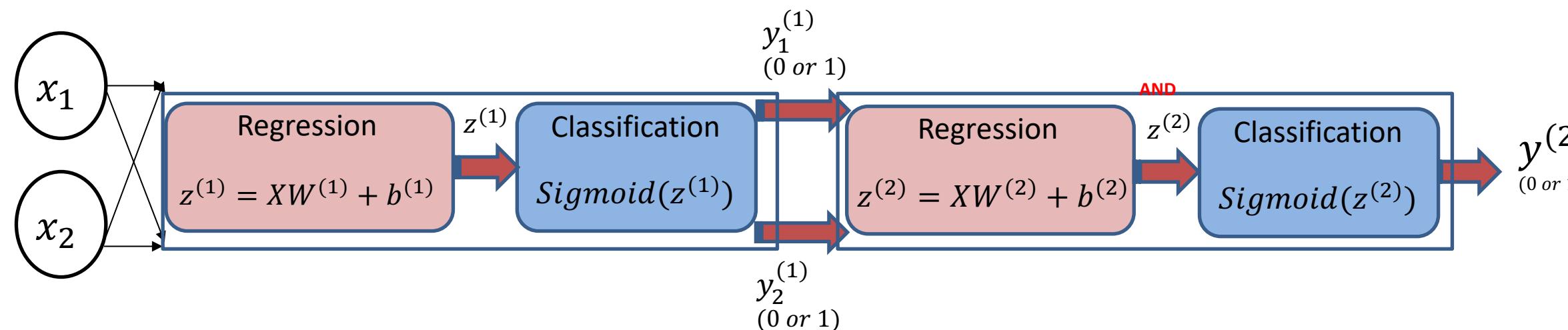
Hypothesis: [[0.00106378]
[0.9988938]
[0.9988939]
[0.00165883]]

Correct: [[0.]
[1.]
[1.]
[0.]]

Accuracy: 1.0

해결책 (구현)

- N개의 Multi-class의 classification의 경우 어떻게 푸나?
- Linear N개 짜리 linear로 풀지!
- Linear로 구분하지 못하는 데이터는 어떻게 함?? → 구부려 → 2차원 이상 함수로.. (sigmoid)



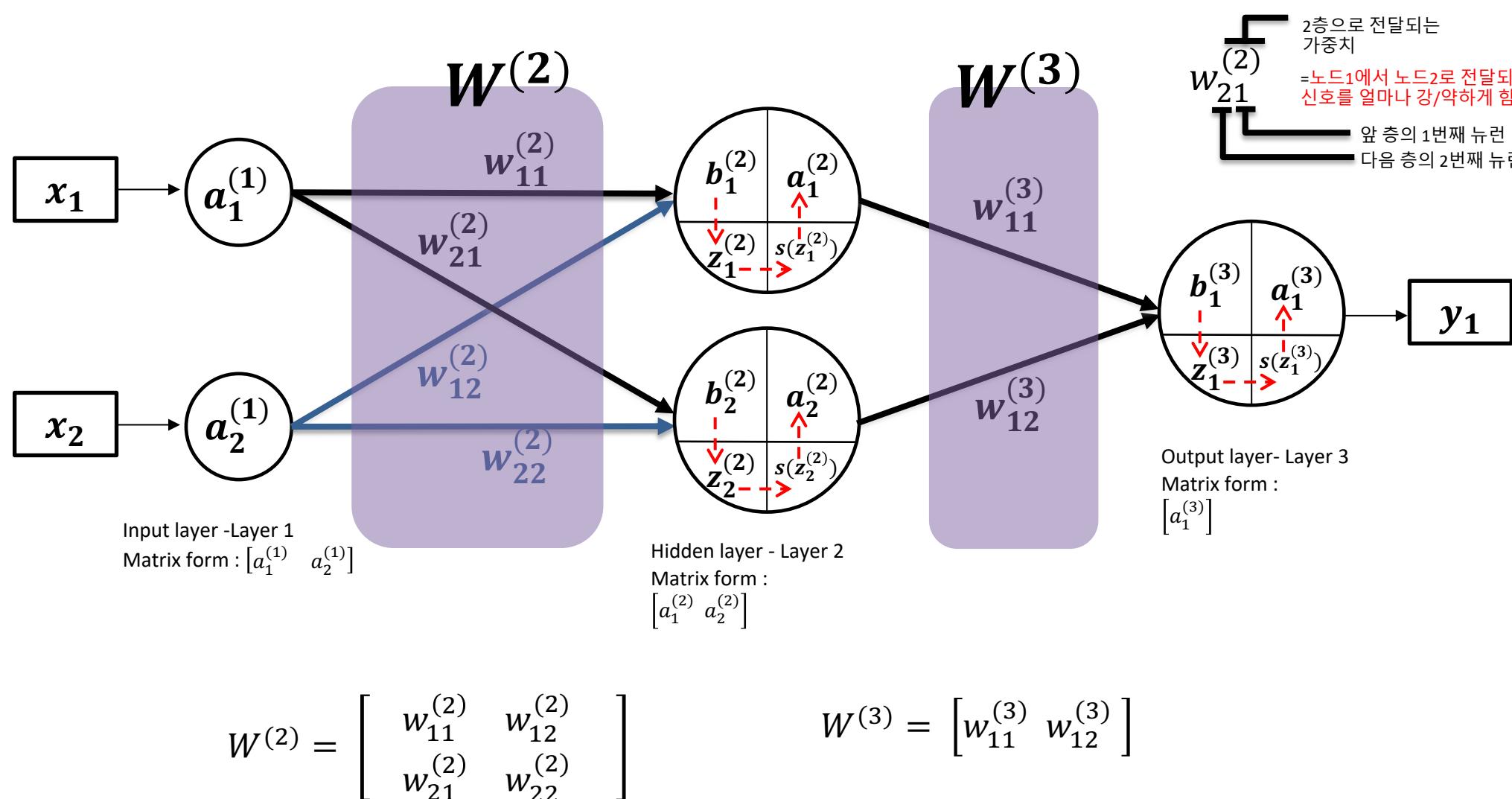
```

X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
# nn Layers
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid).to(device)
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(),
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())

```

해결책 (구현-병렬연산)

- 실제 코딩에서는 W와 X의 병렬연산을 통해 다음과 같은 그림으로 표현할 수 있다.



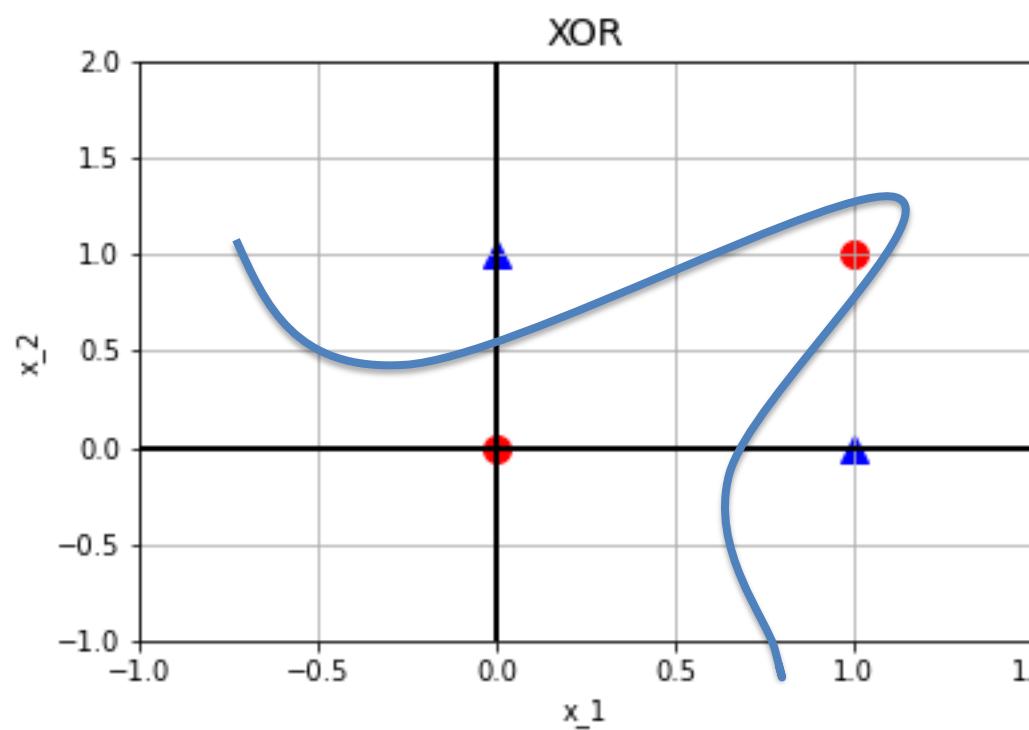
```

X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
# nn Layers
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid).to(device)
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
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())
    
```

Multi-Layer Perceptron (MLP)

결론

- Linear + activation function (sigmoid) 층을 쌓았더니 비선형 관계가 표현이 되었다.
 - 복잡한 관계가 표현이 되었다.
 - [Linear + activation function]의 중첩 → 신경망 → Multi-Layer Perceptron (MLP)



```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
# nn layers
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid).to(device)
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)

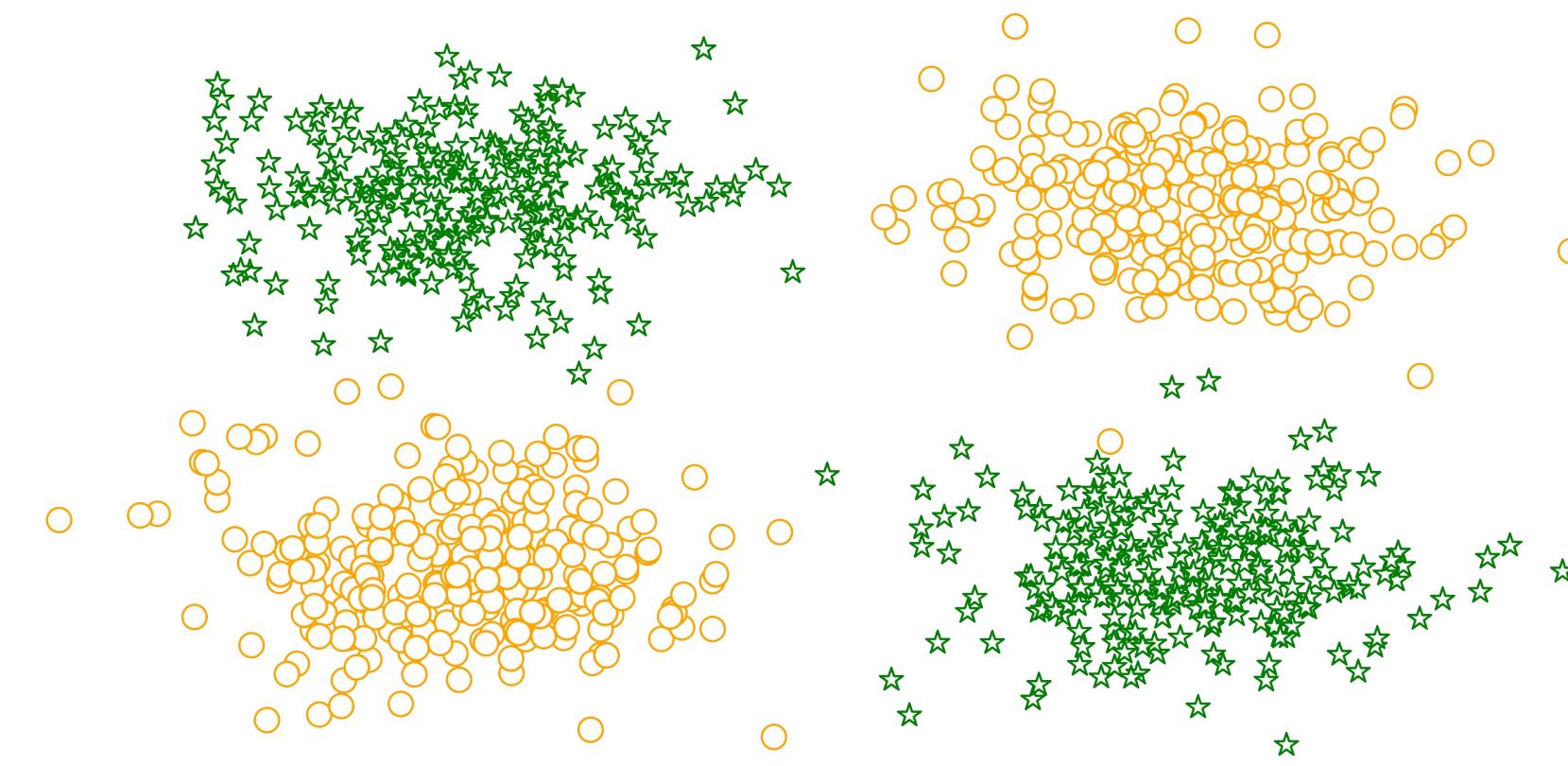
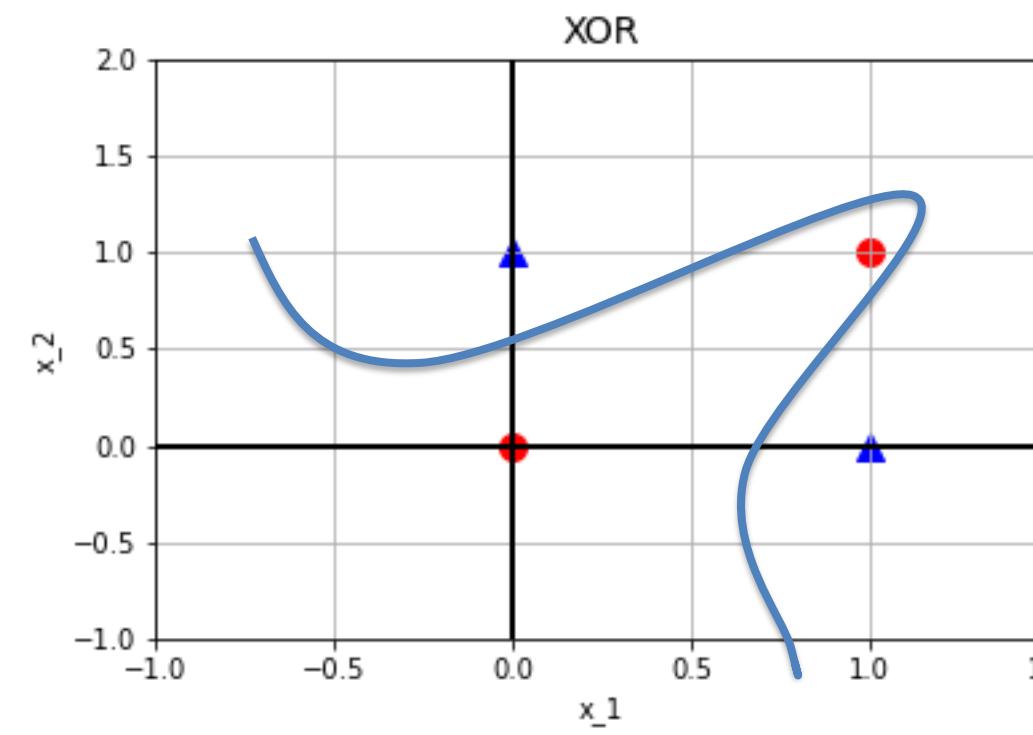
optimizer = torch.optim.SGD(model.parameters(),
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)
    # cost/loss function
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()
    if step % 100 == 0:
        print(step, cost.item())
```

0 0.7434073090553284
100 0.6931650638580322
200 0.6931577920913696
300 0.6931517124176025
...
9800 0.0012681199004873633
9900 0.0012511102249845862
10000 0.0012345188297331333

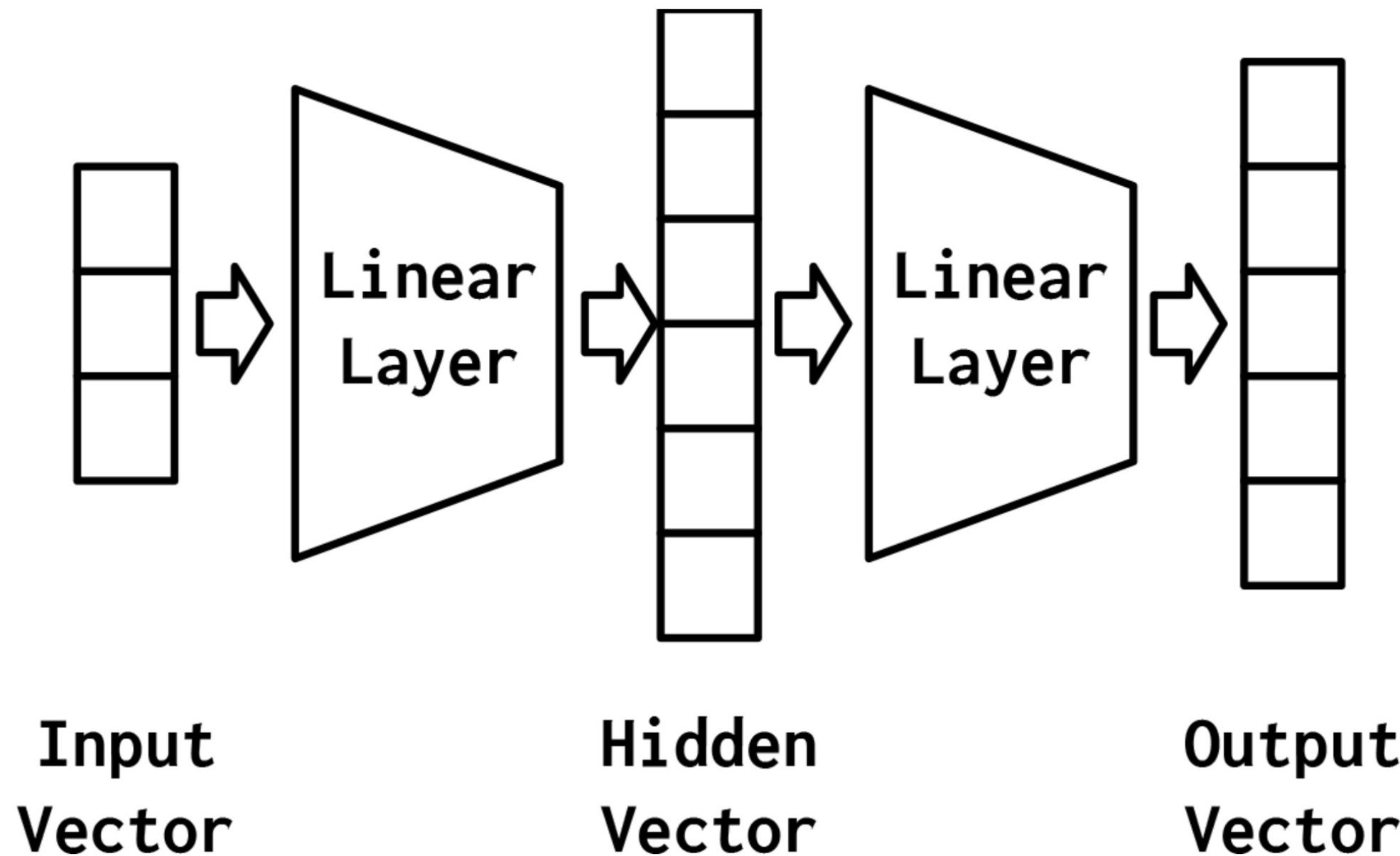
Hypothesis: [[0.00106378]
[0.9988938]
[0.9988939]
[0.00165883]]
Correct: [[0.]
[1.]
[1.]
[0.]]
Accuracy: 1.0

결론

- Linear + activation function (sigmoid) 층을 쌓았더니 비선형 관계가 표현이 되었다.
 - 복잡한 관계가 표현이 되었다.
 - [Linear + activation function]의 중첩 → 신경망 → Multi-Layer Perceptron (**MLP**)



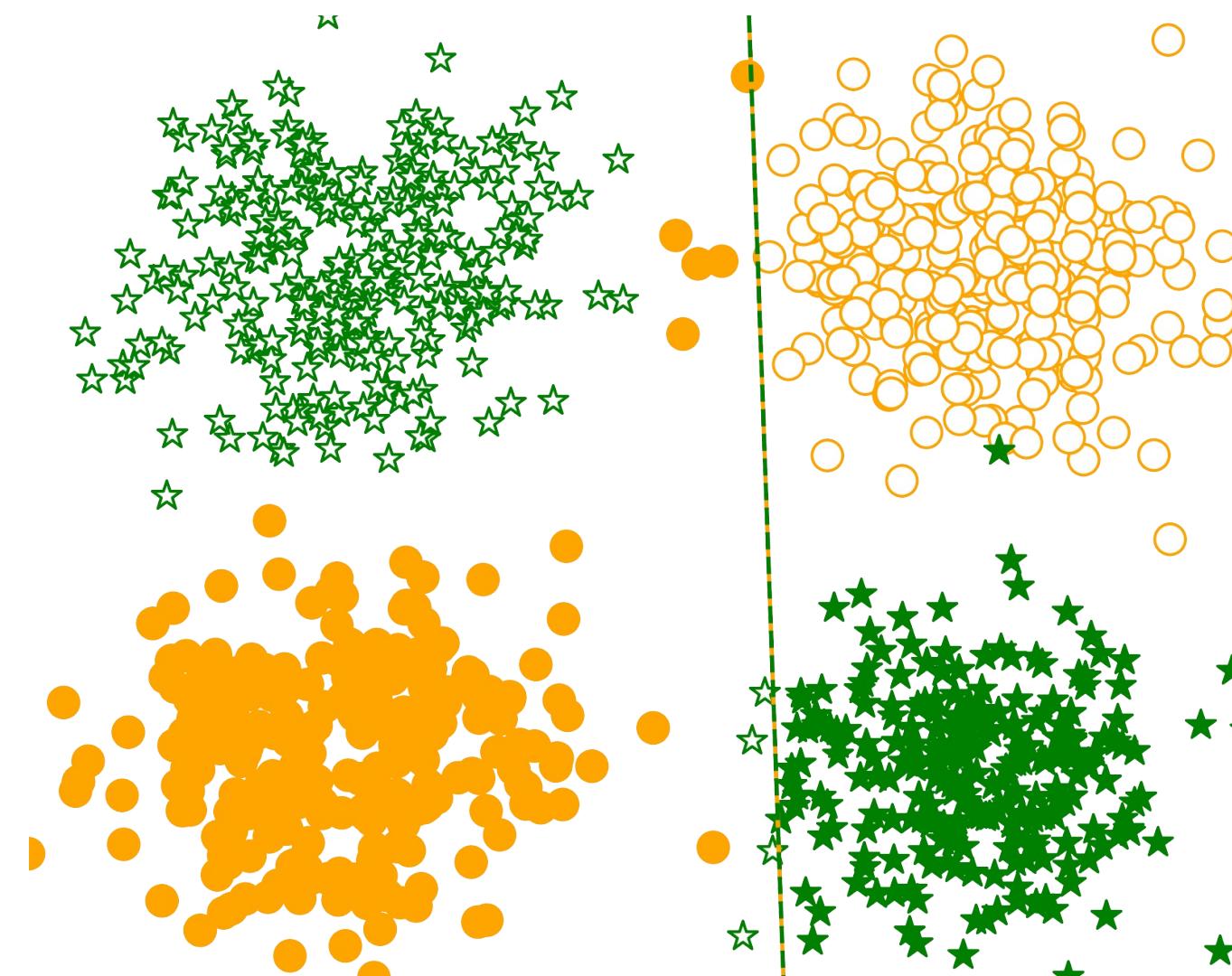
2-Layer MLP



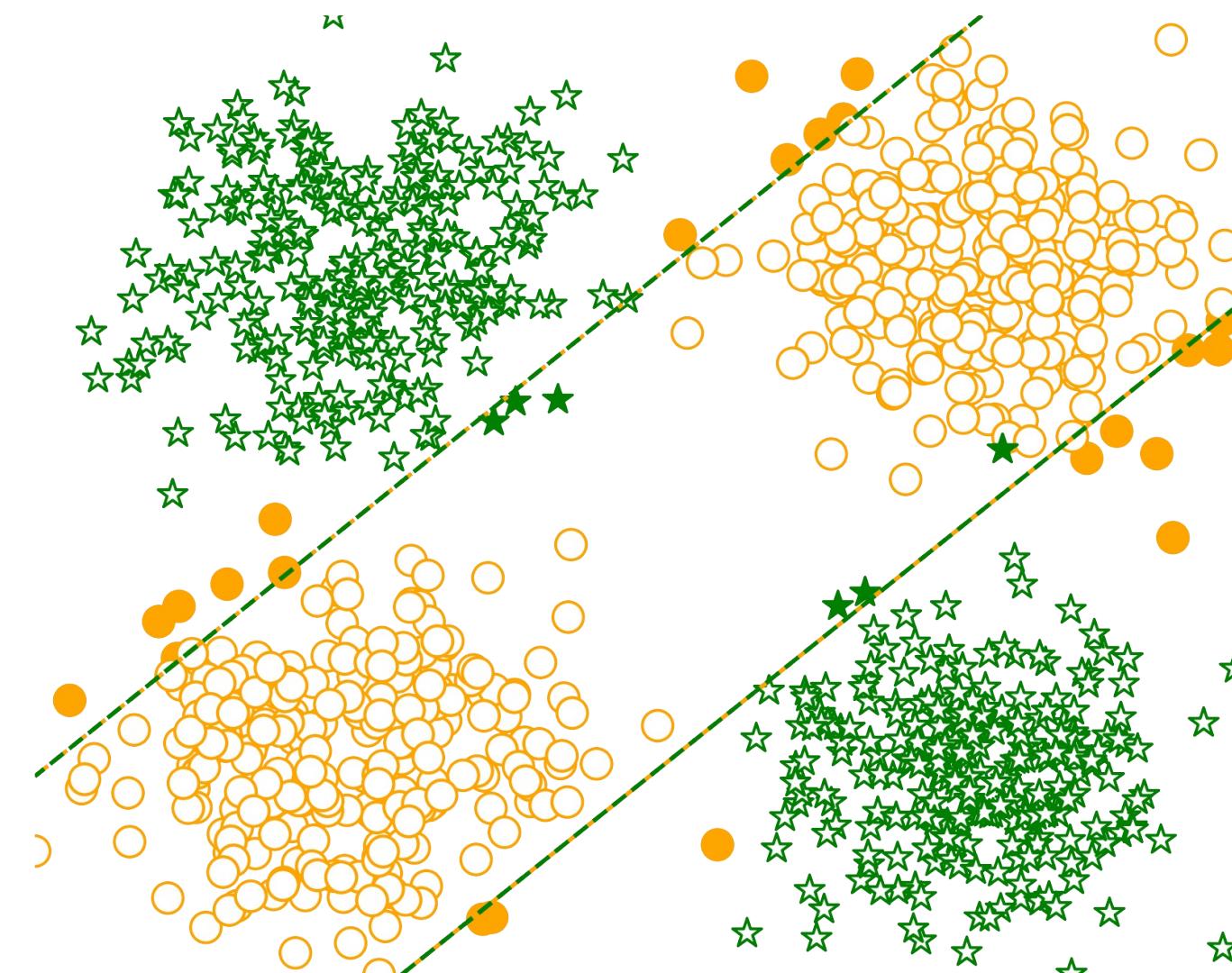
1. 입력 벡터를 모델에 전달한다.
2. 첫 번째 Linear층이 은닉 벡터를 계산한다.
3. 두 번째 Linear층이 은닉 벡터를 사용해 출력 벡터를 계산한다.

Perceptron & MLP

Linear

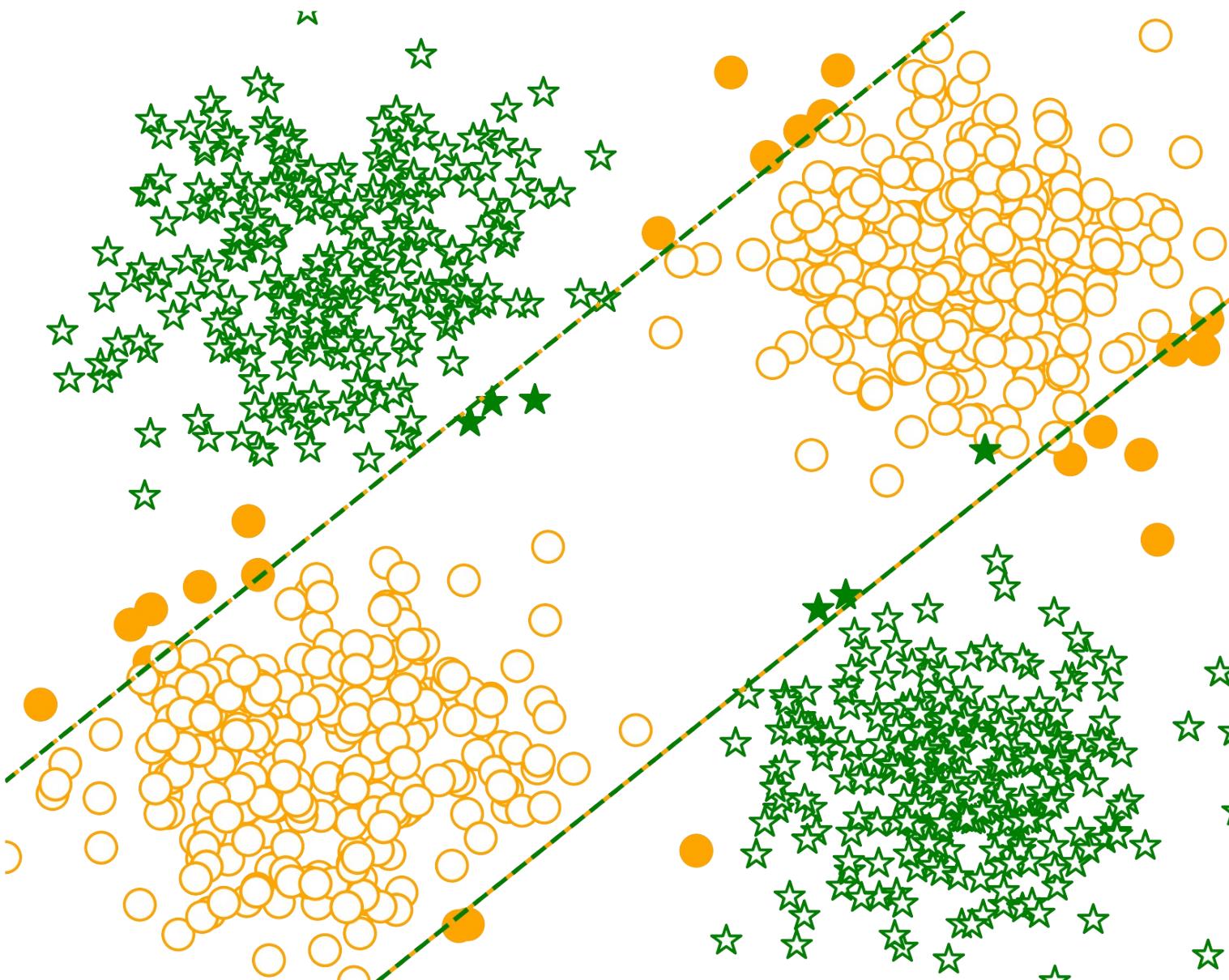


MLP



- ✓ Perceptron으로 학습한 왼쪽은 잘못 색칠된 부분이 많다.
- ✓ MLP로 학습한 오른쪽은 결정 경계를 훨씬 정확하게 학습하였다.

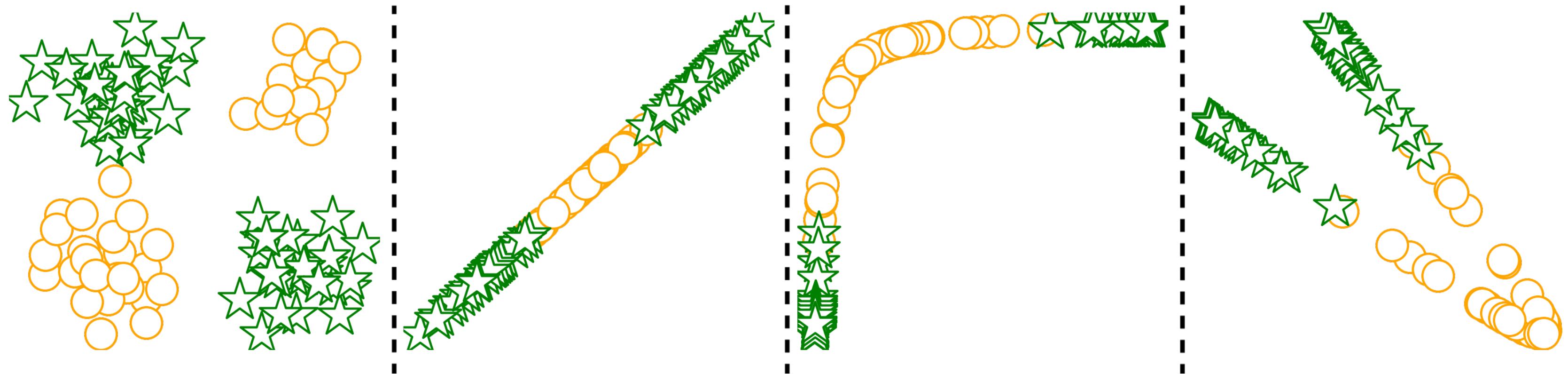
다층 퍼셉트론 (Multilayer Perceptron)



- MLP의 결정 경계가 두 개로 보이지만 사실 하나의 결정 경계임
- 중간 표현이 공간을 변형해 초평면 하나가 두 곳에 나타나도록 만들어서 결정 경계가 이렇게 보이는 것

MLP의 학습 과정

A 2-layer MLP's Input and Intermediate Representation



신경망의 입력

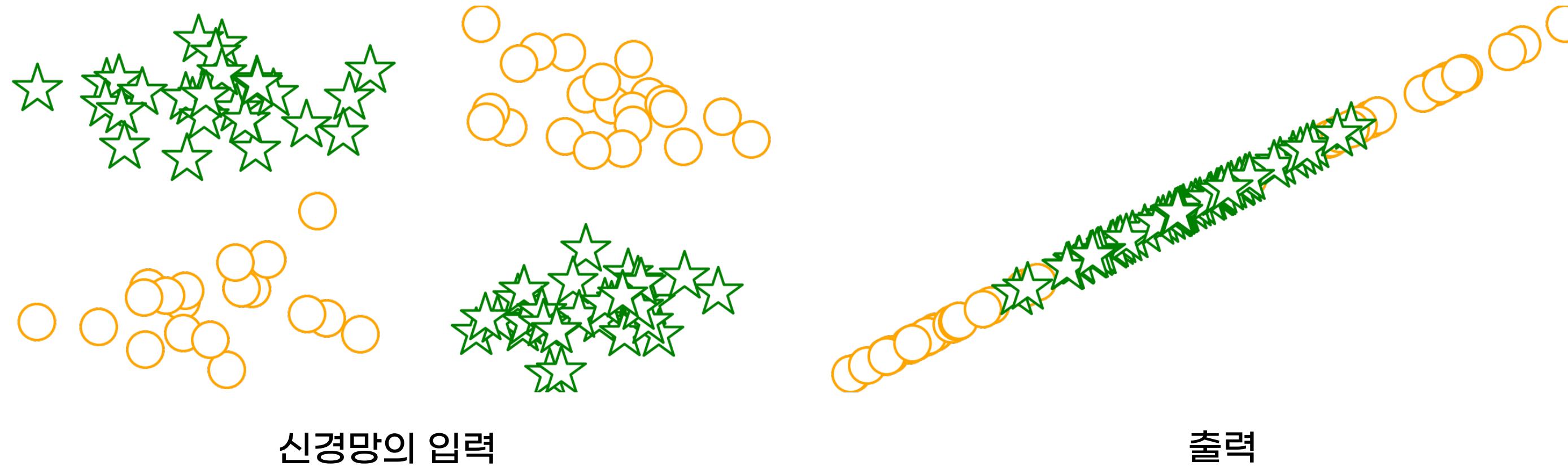
첫 번째 층의
선형 출력

첫 번째 층의
활성화 함수 출력

두 번째 층의
선형 출력

Perceptron의 학습 과정

The Perceptron's Input and Intermediate Representation

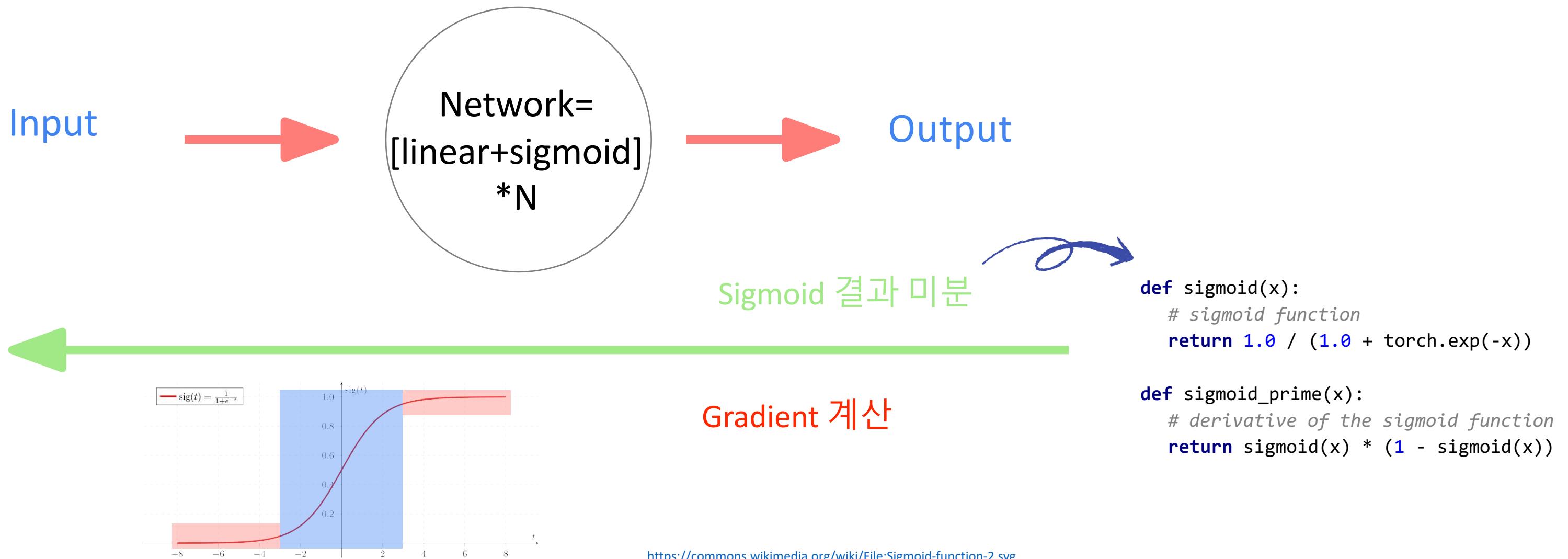


- Linear는 데이터 포인트를 모으거나 재조직하는 중간 표현이 없어 하나의 직선으로 별과 원을 구별할 수 없다.

02. ReLU

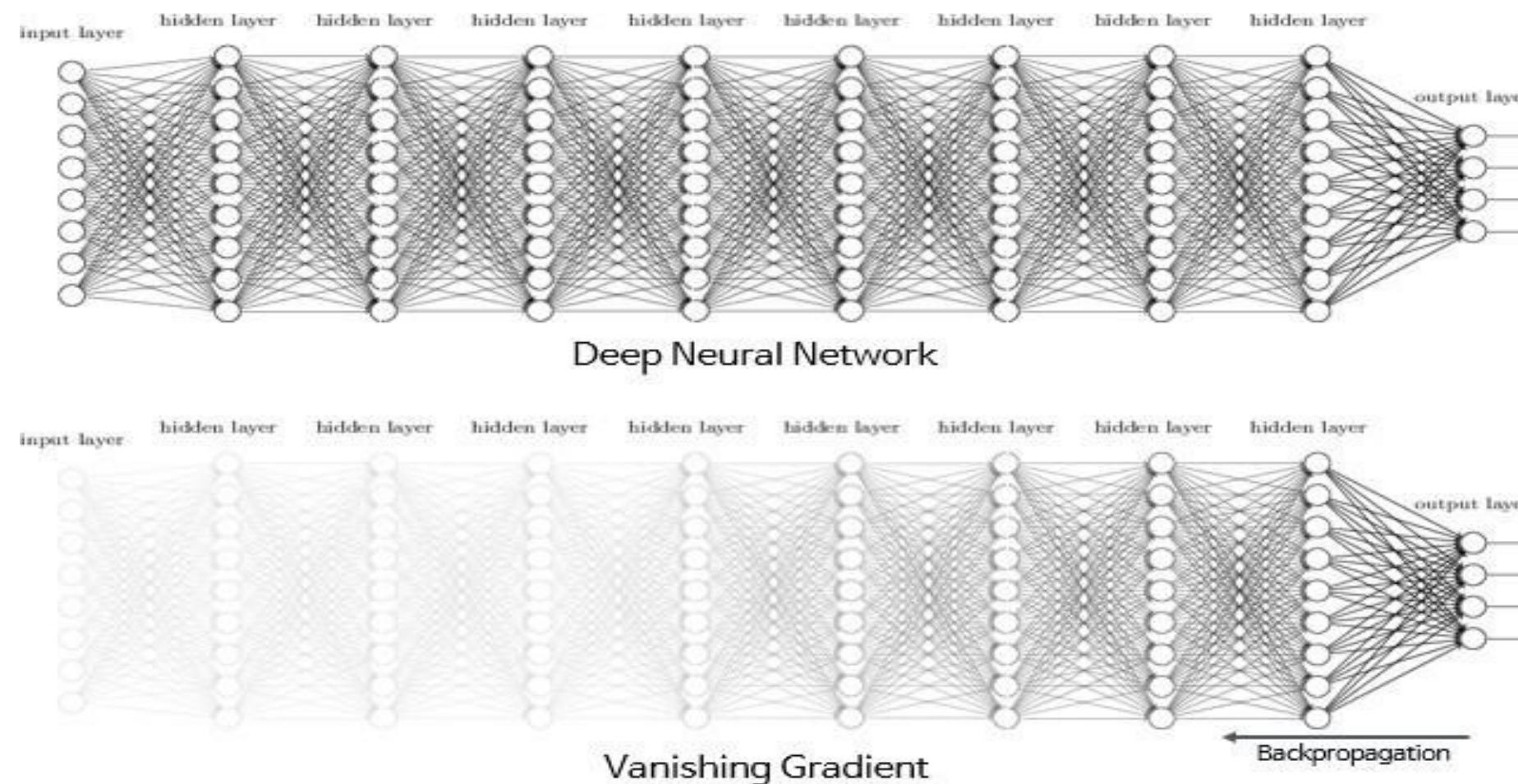
Sigmoid의 문제점

- N개의 layer로 구성된 고도화된 모델을 생각해보자
 - sigmoid 미분을 해야겠지?



Sigmoid의 문제점

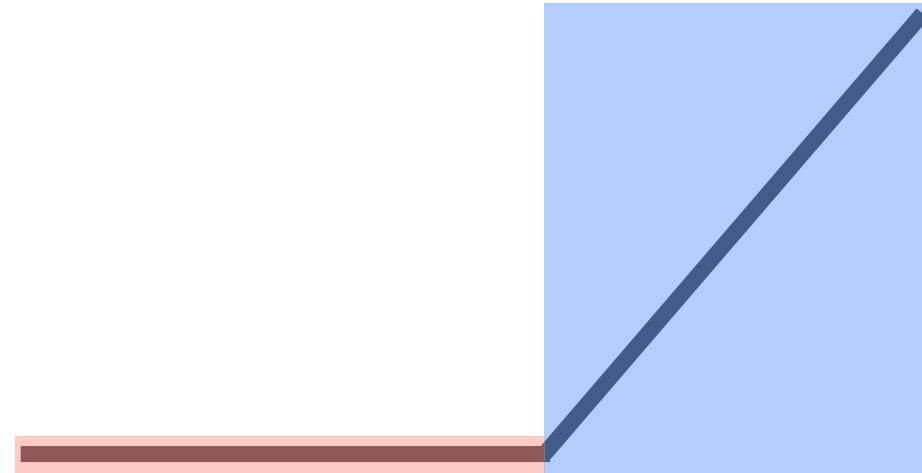
- Sigmoid 미분값이 1보다 작기 때문에 layer가 깊어질 수록 앞쪽으로 오류 전파가 잘 안되는 Vanishing Gradient 발생!



ReLU

- 그러면 미분도 잘되고 non-linear한 activation function은 없을까?
 - ReLU가 있습니다!

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



`x = torch.nn.relu(x)`

03. Document Classification

Document Classification

설명

- Document Classification: 문장의 유형을 분류하자!

- <https://dacon.io/competitions/official/236037/overview/description>

Dataset Info.

• train.csv [파일]

- ID : 샘플 문장 별 고유 ID
- 문장 : 샘플 별 한개의 문장
- 유형 : 문장의 유형 (사실형, 추론형, 대화형, 예측형)
- 극성 : 문장의 극성 (긍정, 부정, 미정)
- 시제 : 문장의 시제 (과거, 현재, 미래)

ID	문장	유형	극성	시제	확실성
1 TRAIN_00000	0.75%포인트 금리 인상은 1994년 이후 28년 만에 처음...	사실형	긍정	현재	확실
2 TRAIN_00001	이어 "앞으로 전문가들과 함께 4주 단위로 상황을 재평가...	사실형	긍정	과거	확실
3 TRAIN_00002	정부가 고유가 대응을 위해 7월부터 연말까지 유류세 인하 ...	사실형	긍정	미래	확실
4 TRAIN_00003	서울시는 올해 3월 즉시 견인 유예시간 60분을 제공하겠...	사실형	긍정	과거	확실
5 TRAIN_00004	익사한 자는 사다리에 태워 거꾸로 놓고 소금으로 코를 막...	사실형	긍정	현재	확실
6 TRAIN_00005	이같은 변화를 포함해 올해 종부세 과세 대상은 당초 21만...	사실형	긍정	현재	확실
7 TRAIN_00006	수수꽃다리과로 북한에서 주로 많이 서식한다는 수수꽃다...	사실형	긍정	현재	확실
8 TRAIN_00007	가장 최근에 있었던, OTT 예능 프로그램 출연으로 일약 스...	사실형	긍정	현재	확실
9 TRAIN_00008	이번 서비스에는 네이버가 자체 개발한 초대규모 AI 하이...	사실형	긍정	과거	확실
10 TRAIN_00009	이 같은 서울시 방침에 직격탄을 맞게 된 곳이 바로 한남근...	사실형	긍정	현재	확실

10 records

Airtable

Copy base View larger version

MLP의 구현에서 고민해볼 것들!

- Document Classification 문제를 `hidden_size = 50` 짜리 MLP로 구성해보세요
 - Layer 수를 늘려보면 어떻게 될까?
 - `hidden_size`의 수를 늘리면 성능이 어떻게 될까?
- layer size와 `hidden_size`와 같이 사람이 설정해 주는 것을 `hyperparameter`라고 한다.
- `hyperparameter`의 크기를 늘렸을 때 `trade-off`은 무엇일까?

감사합니다.