Machine Learning Practice

9주차 실습

-chap.4실습 - 1

전자공학과
2022144007
김의진

#과제 1

1) One-Hot Encoding 구현

Classification

구분	분류 방식	특징
0.5 기준	값 ≥ 0.5 → class 1 값 < 0.5 → class 0	이진 분류(2-class)
구간 분할	[0.0 ~ 0.33]: Class 1 [0.33 ~ 0.66]: Class 2 [0.66 ~ 1.0]: Class 3	여러 클래스 대상, 구간 나눔

→ Class가 많아지면 출력구간 촘촘해짐

→ 미세한 출력변화에 class 바뀌기 쉬움

→ One-Hot Encoding 적용

#과제 1

1) One-Hot Encoding 구현

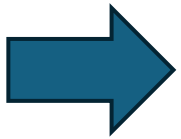
- One-Hot Encoding

각 클래스 독립된 이진 벡터로 표현

Ex) class 1 → 0, 0, 1

class 2 → 0, 1, 0

class 3 → 1, 0, 0

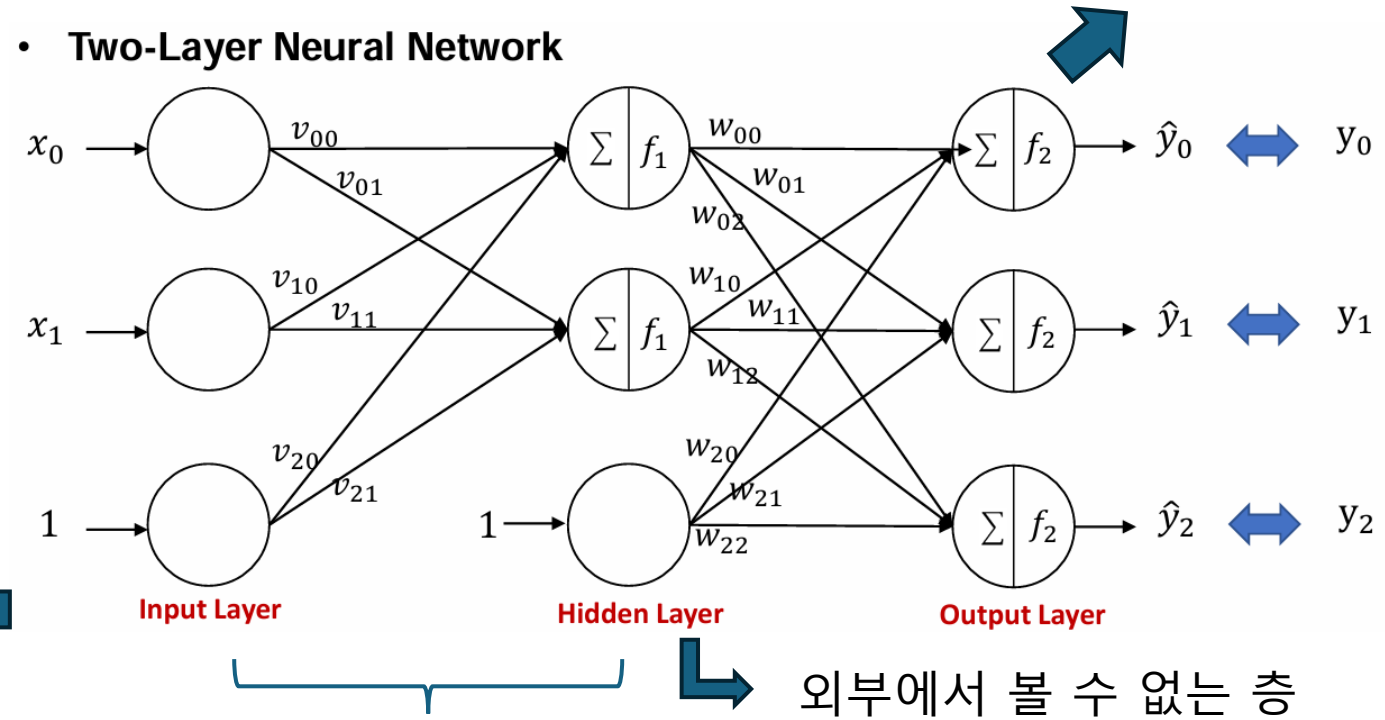


Neural Network에서 이용할 수 있음

#과제 1

1) One-Hot Encoding 구현

데이터의 output class 수



데이터
input 속성수

Perceptron
: 입력과 가중치의 선형 조합
→ Logistis regression과 구조상 비슷함

외부에서 볼 수 없는 층
사용자는 hidden layer의 node만 조작 가능

#과제 1

1) One-Hot Encoding 구현

NN_data.csv파일 출력 종류: 1, 2, 3, 4, 5, 6 → class 개수 6개

One-Hot Encoding

y_1	1	0	0	0	0	0
y_2	0	1	0	0	0	0
y_3	0	0	1	0	0	0
y_4	0	0	0	1	0	0
y_5	0	0	0	0	1	0
y_6	0	0	0	0	0	1

#과제 1

2) Two-Layer Neural Network 구현

```
def Two_Layer_Neural_Network(x_input, y_data, L):

    M = ch_count(x_input)
    Q = ch_count(y_data)


    x_input = add_dummy(x_input)
    v_matrix = np.random.rand(L, M + 1) -0.5
    alpha = np.dot(v_matrix, x_input)
    b_matrix = sigmoid_function(alpha)

    b_matrix = add_dummy(b_matrix)
    w_matrix = np.random.rand(Q, L + 1) - 0.5
    beta = np.dot(w_matrix, b_matrix)
    y_hat = sigmoid_function(beta)

    return y_hat
```

사용자 지정 Hidden Layer node 수

Input 속성 수, output class 수 check

Hidden Layer

Sigmoid함수의 기울기는 0에 가까울수록 값이 커짐

→ Weight초기값이 0 근처여야 sigmoid 출력이 0.5 근처에 골고루 분포

0.5기준으로 분류 했을 때를 고려함

#과제 1

2) Two-Layer Neural Network 구현

```
v_matrix = np.random.rand(L, M + 1) - 0.5
w_matrix = np.random.rand(Q, L + 1) - 0.5
```

이와 같이 weight초기값 설정할 때 범위: -0.5 ~ 0.5

Weight 범위

-1 ~ 0

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0

0 ~ 1

	0	1	2	3	4	5	6
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1

-0.5 ~ 0.5

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1
2	1	1	1	1	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0

Sigmoid 함수의 특성으로 인해 0.5기준으로 분류 하면 왼쪽과 같은 data를 얻게 됨

#과제1

3) Accuracy 함수 구현

```
1) print("0.5 기준 분류 정확도:", accuracy1)
2) print("Two Layer Neural Network 정확도:", accuracy2)
3) print("Rule-based 분류 정확도:", accuracy3)
```

1) One_hot Encoding(0.5를 기준)을 이용한 classification
2) One_hot Encoding(최대값만 1)을 이용한 classification
3) 1을 class 수만큼 나누어 차례대로 class를 분류한 classification

----------weight update없는 forward propagation에서----------

1) 의 예측 정확도 → class 6개 각각이 독립됐다 가정해 $1 / 2^6$승이라 가정
2) 의 예측 정확도 → class 6개중 하나만 1 일 확률은 1 / 6이라 가정
3) 의 예측 정확도 → class 6개라 1을 6구간으로 나눈 것 중 하나에 위치할 확률 1 / 6이라 가정

#과제1

3) Accuracy 함수 구현

실제 예측 데이터에 대한 정확도

```
class 수: 6
0.5 기준 분류 정확도: 0.0005555555555555556
Two Layer Neural Network 정확도: 0.1661111111111111
Rule-based 분류 정확도: 0.15444444444444444
```

1) $1 / 2^6 \approx 0.015$ 지만 sigmoid 함수 분포 특성 때문에 0또는 더 작은 값들이 나옴

2) 가정한 정확도(1 / 6 ≈ 1.66666~)와 거의 같은 값 0.1661이 나옴

3) 가정한 정확도(1 / 6 ≈ 1.66666~)와 거의 같은 값 0.1544~이 나옴

결론

1) 0.5 기준으로 한 classification: 한 데이터에 대한 성분들 중 1이 하나만 있는 것이 아니고 모든 class에 대해 1이 나올 확률 높아 정확도가 떨어짐 → 향후 역전파이용 weight update시에도 정확도 높을지 불분명함

2) 최대값을 이용한 classification: 최대값만을 1로 만들어서 1)과 달리 좋은 예측값 만드는데 기대가 됨→ 역전파 이용한 weigh update시 좋은 성능을 낼 수 있을 듯.

→Neural Network에 적합

3) 0~1사이 class 개수로 구간 나눈 classification: weight update없는 순전파에서 2)와 같이 예측값 같음 → 하지만 weight update 포함한 역전파에서 세부 파라미터들에 예민해 좋은 예측값 찾기 번거로울 것 같음