두산 Rokey Boot Camp

스터디 주간 활동 보고서

<table>
<tr><td>팀명</td><td>벌꿀오소리</td><td>제출자 성명</td><td>정민섭</td></tr>
<tr><td>참여 명단</td><td colspan="3">정찬원, 임소정, 서준원, 송주훈, 강인우, 정민섭</td></tr>
<tr><td>모임 일시</td><td colspan="3">2025년 2월 27일 21시 ~22시(총 1시간)</td></tr>
<tr><td>장소</td><td>Google meet 화상회의</td><td>출석 인원</td><td>6 / 6</td></tr>
<tr><td>학습목표</td><td colspan="3">교재 3 ~ 6단원 정리 및 발표</td></tr>
<tr><td>학습내용</td><td colspan="3">

- **3단원 (처음 시작하는 머신 러닝): 정찬원**

1. 합성 함수란?

두 함수 f(x)와 g(x)가 있을때, 한 함수의 출력 값을 다른 함수의 입력값으로 사용하는 방식이다. 딥러닝에서는 여러 개의 층을 쌓아, 합성함수를 구성하여 복잡한 문제를 해결한다.

</td></tr>
</table>

$$h(x) = f(g(x))$$

$$f(x) = 2x + 3, \quad g(x) = x^2$$
$$z(x) = f(g(x)) = 2x^2 + 3$$

<Fig 1. 수학적 표현 및 예제>

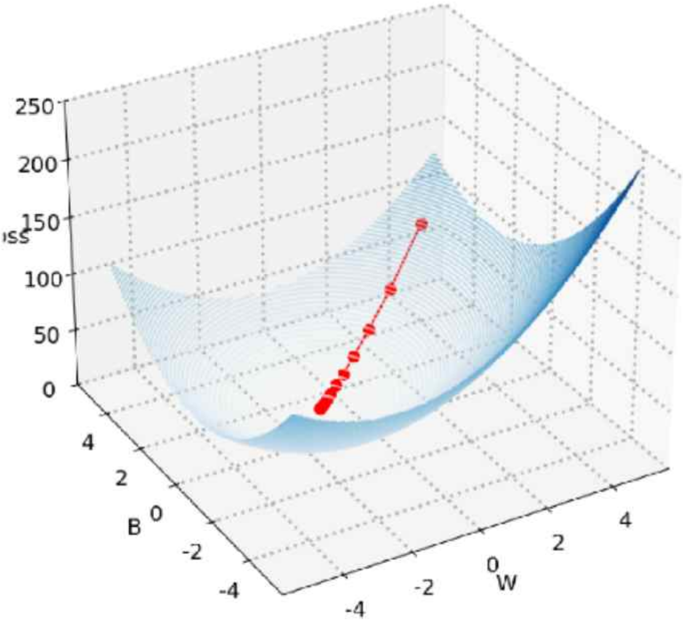
2. 경사 하강법(Gradient Desent)

경사 하강법이란 머신러닝 및 최적화 문제에서 널리 사용되는 알고리즘으로, 비용 함수를 최소화하는 방향으로 가중치를 업데이트한다. 경사 하강법의 단계는 다음과 같다.

1. **초기값 설정**: 파라미터(가중치)를 임의로 초기화
2. **기울기 계산**: 현재 위치에서 함수의 기울기(미분값)를 계산
3. **파라미터 업데이트**: 기울기의 반대 방향으로 이동하여 최적값을 찾음
4. **반복**: 수렴할 때까지 위 과정을 반복

<Fig 2. 경사 하강법의 단계>

경사 하강법을 시각화하면 다음과 같다.



<Fig 3. 경사 하강법의 3D plot>

경사 하강법의 종류로는 배치 경사 하강법, 확률적 경사 하강법, 미니배치 경사 하강법이 있다. 각 경사 하강법의 특징은 다음과 같다.

- **배치 경사 하강법 (Batch Gradient Descent)**
  - 전체 훈련 데이터를 한 번에 사용하여 기울기를 계산
  - 장점: 전체 데이터를 사용하여 안정적인 학습 가능
  - 단점: 데이터가 많을 경우 계산 비용이 큼
- **확률적 경사 하강법 (Stochastic Gradient Descent, SGD)**
  - 훈련 데이터에서 하나의 샘플을 랜덤하게 선택하여 기울기를 계산
  - 장점: 계산 속도가 빠르고, 메모리 요구량이 적음
  - 단점: 매 스텝마다 최적화 방향이 바뀌어 학습이 불안정할 수 있음
- **미니배치 경사 하강법 (Mini-batch Gradient Descent)**
  - 훈련 데이터를 작은 배치 단위로 나누어 각 배치에서 기울기를 계산
  - 장점: 배치 경사 하강법의 안정성과 SGD의 속도를 조합
  - 단점: 배치 크기 선택에 따라 성능이 달라질 수 있음

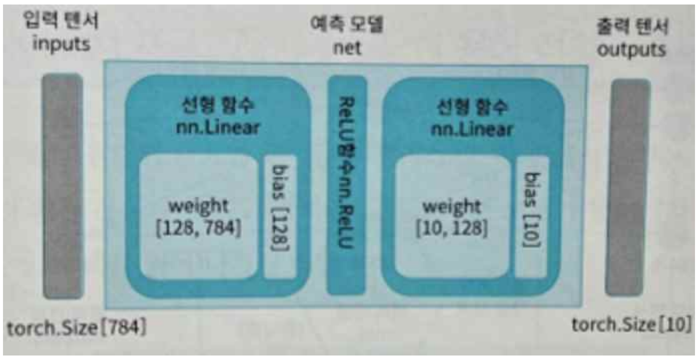
※ 이후 데이터 전처리, 예측 계산, 손실 계산, 경사 계산 등, 예시로 간단한

선형 회귀를 진행하였으나, 뒤에 단원들과 내용이 겹쳐 보고서 상에선 생략함.

- **4단원 (예측 함수 정의하기): 임소정**

1. 예측함수란?

예측함수란 레이어 함수의 조합이다. 여기서 레이어 함수란 다음과 같이 여러 빌딩블록, 레이어, 모듈 등을 쌓아놓은 함수이다.



<Fig 4. 일반적인 머신 러닝 모델>

2. 용어 정리

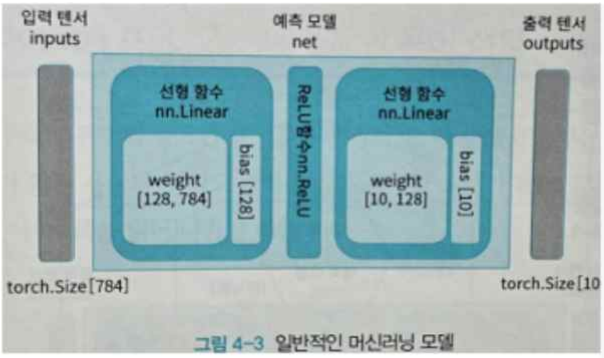
파라미터: 레이어 함수 내부에서 가지고 있는 입력 텐서 이외의 데이터 - 레이어 함수의 파라미터를 조정하는 것 = 학습

입력 텐서: 신경망 개념에서 입력층에 해당

출력 텐서: 신경망 개념에서 출력층에 해당

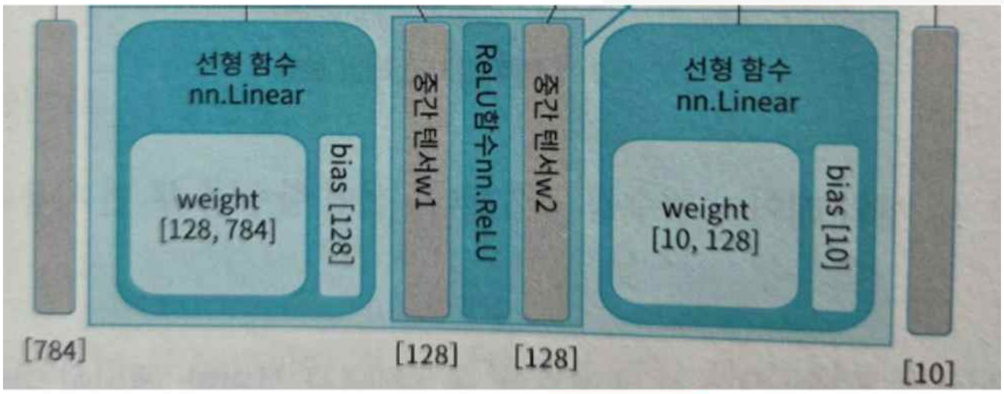
머신러닝 모델: 여러 개의 레이어 함수를 조합해 입력 텐서에 대해

바람직한 출력텐서를 출력해주는 합성함수.




```
#코드 4-1) 레이어 함수 정의

#첫번째 선형 함수
#입력수:784, 출력수:128
l1 = nn.Linear(784, 128)

#두번째 선형 함수
#입력수:128, 출력수:10
l2 = nn.Linear(128, 10)

#활성화 함수
relu = nn.ReLU(inplace=True)
```

128이 은닉층의 노드수임!




```
#코드 4-2) 입력 텐서로부터 출력 텐서를 계산

# 더미 입력 데이터 작성
inputs = torch.randn(100, 784)

# 중간 텐서 1 계산
m1 = l1(inputs)

# 중간 텐서 2 계산
m2 = relu(m1)

# 출력 텐서 계산
outputs = l2(m2)

# 입력 텐서와 출력 텐서 shape 확인
print('입력 텐서', inputs.shape)
print('출력 텐서', outputs.shape)
```

```
입력 텐서 torch.Size([100, 784])
출력 텐서 torch.Size([100, 10])
```

```
#코드 4-3) nn.Sequential을 사용해 전체를 합성 함수로 정의

net2 = nn.Sequential(
    l1,
    relu,
    l2
)

outputs2 = net2(inputs)

# 입력 텐서와 출력 텐서의 shape 확인
print('입력 텐서', inputs.shape)
print('출력 텐서', outputs2.shape)
```

```
입력 텐서 torch.Size([100, 784])
출력 텐서 torch.Size([100, 10])
```

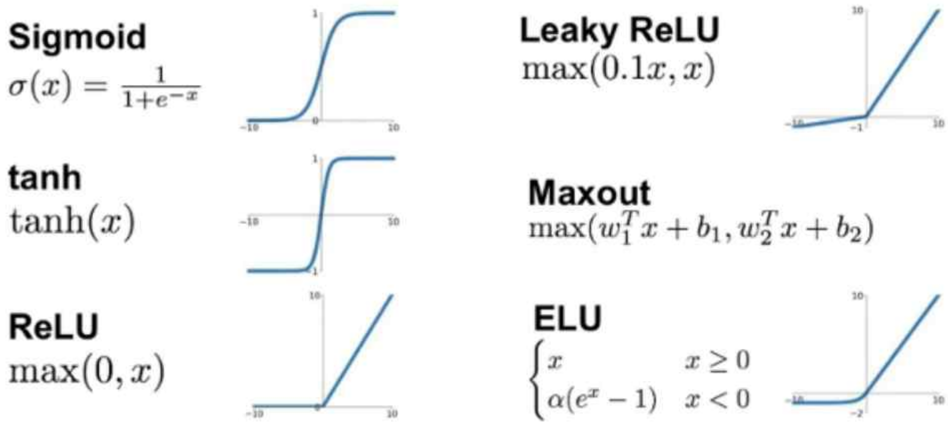
nn.Sequential을 통해 중간 텐서 생략

<Fig 5. 모델을 코드로 구현한 모습>

3. 활성화 함수의 목적

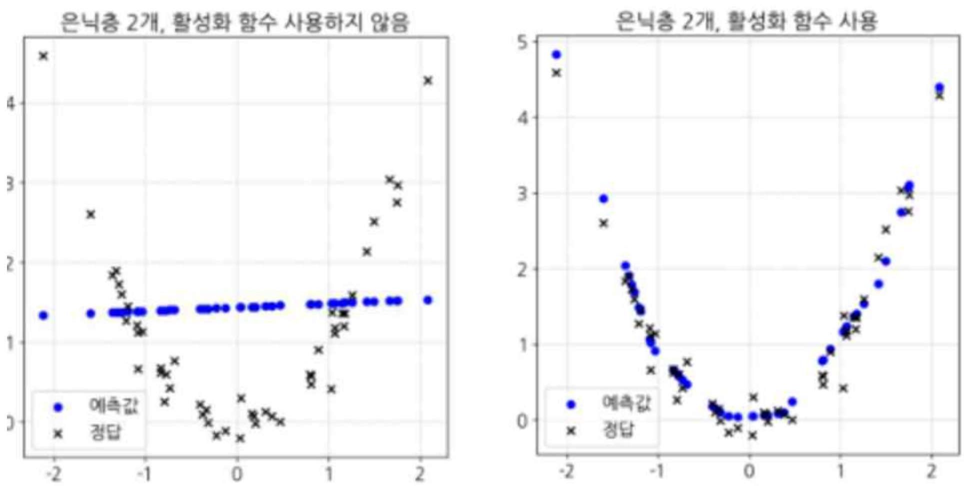
다음은 각 활성화 함수에 따른 수식과 특성을 그래프로 나타낸

것이다.



<Fig 6. 각 활성화 함수의 모양과 수식>

다음은 활성화 함수를 사용하지 않았을때와 사용하였을때의 차이를 시각화한 것이다.



<Fig 7. 활성화 함수 사용에 따른 학습의 변화>

'비선형 함수'로 불리는 활성화 함수를 선형 함수 사이에 넣어야 비로소 깊은 층을 가진 딥러닝 모델이 의미를 가지는 것을 확인할 수 있다.

- **5단원 (선형 회귀): 강인우 ,정민섭**

1. 선형회귀란?

독립 변수와 종속 변수 간의 관계를 선형 방정식으로 모델링하며, 가중치, 편향를 학습

- 지도학습
- 회귀 → 목적변수가 연속형

단순회귀: Feature가 하나

중회귀: Feature가 여러개

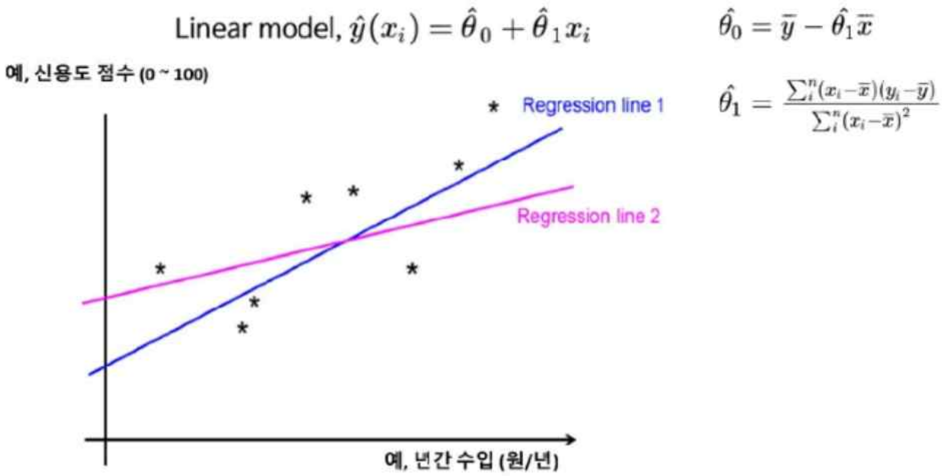
다항회귀: 2차항 이상이 포함된 경우

주요 선형회귀 기법

1. OLS (Ordinary Least Squares, 최소제곱법)
2. Ridge Regression (릿지 회귀)
3. Lasso Regression (라쏘 회귀)
4. Elastic Net (엘라스틱넷 회귀)
5. Robust Regression (강건 회귀)
6. Bayesian Regression (베이지안 회귀)

2. 최소 제곱 추정(OLS)

잔차(residual, 오차)의 제곱합을 최소화하는 방식으로 최적의 회귀선을 구하는 기법

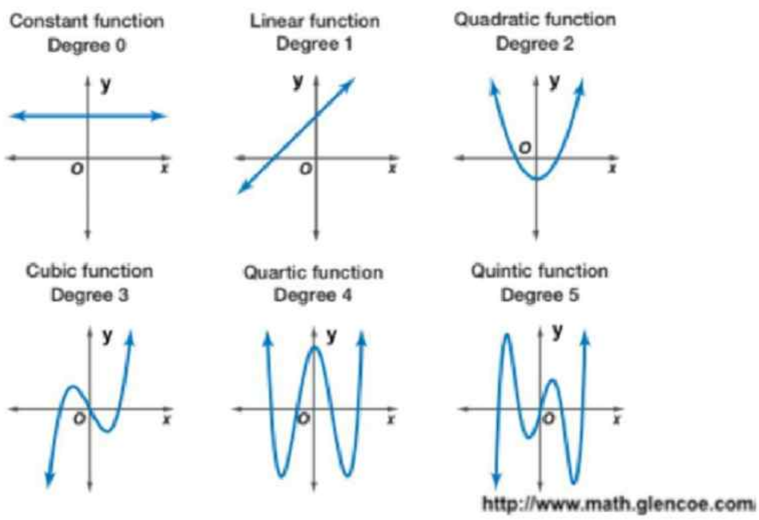


<Fig 8. OLS 추정 수식 및 시각화>

3. 다항 회귀

모델이 복잡해질 수록 차수(degree)가 증가한다.

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_p x^p + \epsilon_i$$



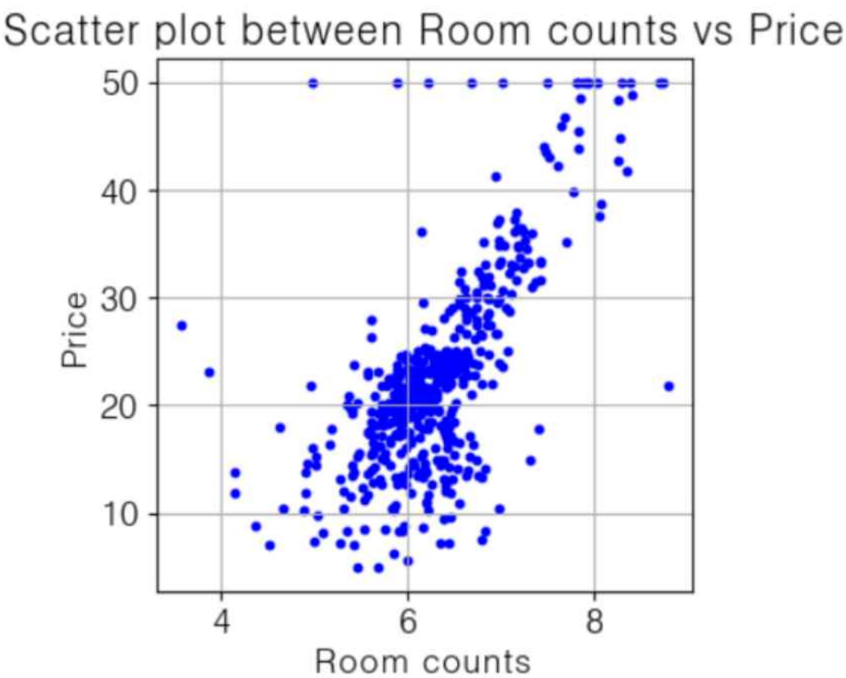
<Fig 8. 차수에 따른 그래프의 형태>

4. 주요 메서드 & 속성

메서드/속성	설명
l1.parameters() l1.named_parameters()	모델의 파라미터 값(가중치, 편향)만 반환 파라미터의 이름과 값(가중치, 편향)을 함께 반환
l1.weight l1.bias	입력 데이터에 곱해지는 학습 가능한 가중치 행렬 각 출력 뉴런에 더해지는 편향 값
tensor.shape tensor.size()	텐서의 크기(차원) 반환 `torch.Size` 객체
nn.init.constant_()	텐서를 지정한 상수 값으로 초기화
nn.MSELoss()	PyTorch의 손실 함수 MSE 생성자
nn.Linear()	선형 변환 레이어 생성

5. 보스턴 주택 가격 데이터셋 전처리

데이터셋에서 가격과 방의 크기를 추출하여 산점도로 plot하였다.

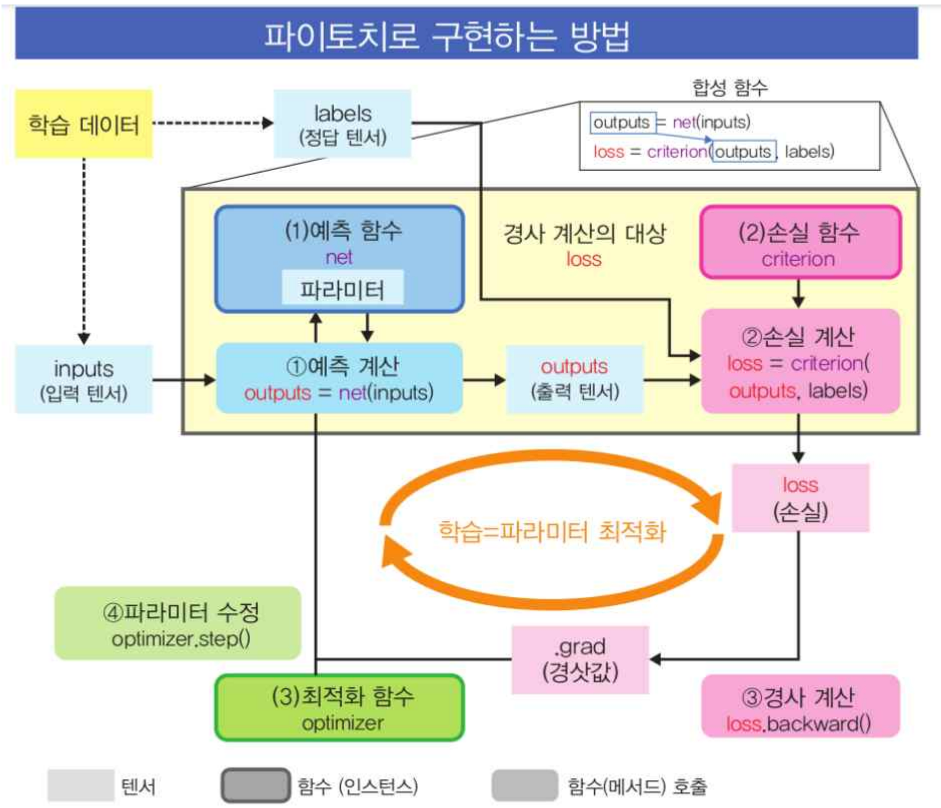


<Fig 9. 방의 갯수에 따른 가격 산점도>

6. 보스턴 주택 가격 데이터셋을 이용한 선형회귀 실습

위에서 전처리한 데이터셋을 이용하여 방의 갯수에 따른 가격을 예측하는 단일 선형 회귀 모델을 구현하였다.

모델의 workflow는 다음과 같다.

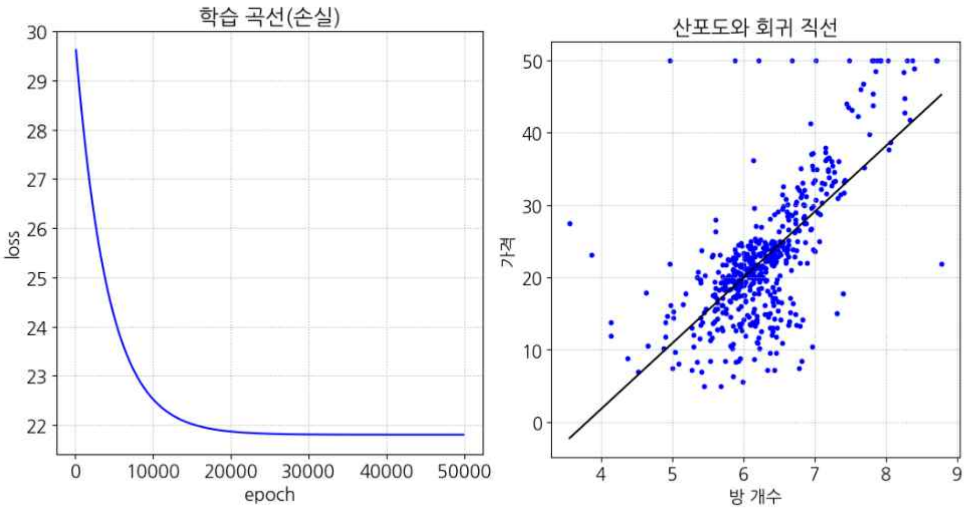


<Fig 10. 모델의 구현 과정>

다음은 모델의 구현과 학습을 코드상으로 나타낸 것이다.

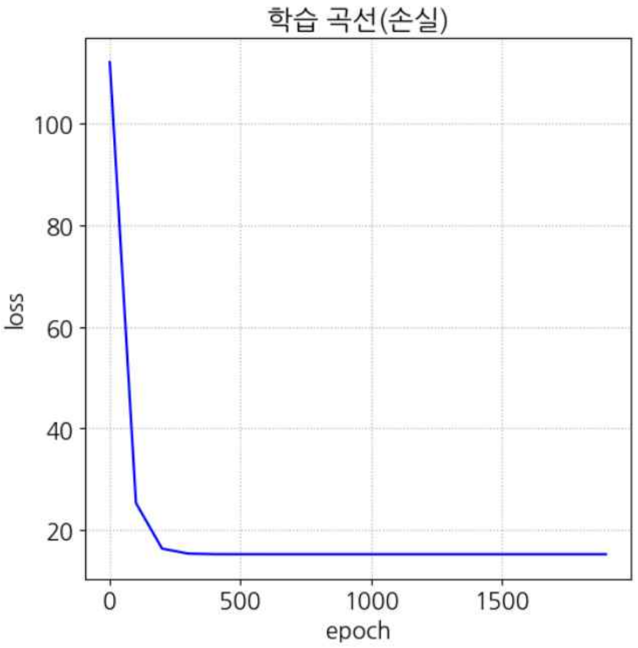
```
class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()

        self.l1 = nn.Linear(n_input, n_output)

    def forward(self, x):
        x1 = self.l1(x)
        return x1

#입력값
inputs = torch.tensor(x, dtype = torch.float32)
x = x_org[:,feature_names == 'RM']

net = Net(n_input, n_output) # 모델 정의
criterion = nn.MSELoss() # 손실 함수

lr = 0.01
num_epochs = 50000
optimizer = optim.SGD(net.parameters(), lr=lr)

history = np.zeros((0,2))


for epoch in range(num_epochs):

    optimizer.zero_grad()
    outputs = net(inputs)
    loss = criterion(outputs, labels1)
    loss.backward()
    optimizer.step()

    if epoch % 100 == 0:
        history = np.vstack((history, np.array([epoch, loss.item()])))
        print(f'Epoch {epoch} loss: {loss.item():.5f}')
```

모델의 학습곡선과 회귀 직선은 다음과 같다. 학습곡선을 보면 loss가 발산하지 않고, 점점 수렴하는 것을 확인할 수 있는데 이는 모델이 잘 학습했음을 의미한다.



<Fig 11. 단일 회귀 모델의 학습 곡선과 회귀직선>

이후 기존 데이터셋에 저소득자 비율을 추가하여 중회귀 모델로 확장하여 동일한 조건에서 학습시켰으나, loss가 발산하였다. 이를 해결하기 위해 lr를 기존 0.01에서 0.001로 변경하여 학습시켰다.



<Fig 12. 중회귀 모델의 학습 곡선과 회귀직선>

 단일 회귀 모델과 마친가지로, loss가 발산하지 않고, 점점 수렴하는 것을 확인할 수 있다. 단일 회귀의 경우는 loss값이 21.8 정도였지만 중회귀 모델의 loss값이 15.2정도이기 때문에, 이번 결과가 조금 더 나은 근사라는 것을 확인할 수 있다.

- **6단원 (이진 분류): 서준원, 송주훈**

1. 이진 분류

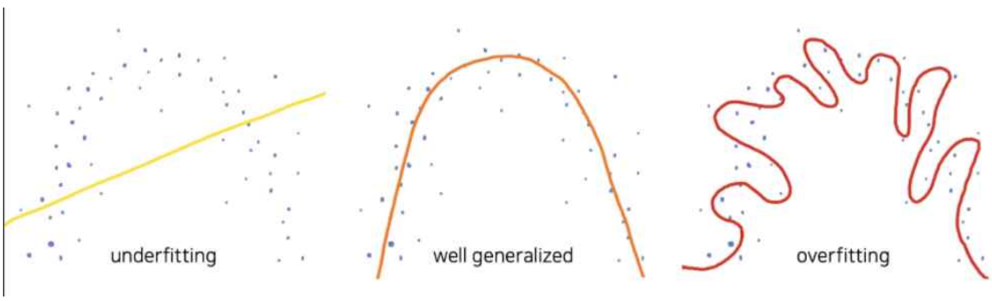
지도 학습이며, Target(반응변수)이 두개의 클래스로 분류

ex) 시험: Pass / Fail, 신용 등급: 양호/ 불량, 종양: 악성 / 양성

분류 모델은 **'정확도'**라는 지표를 사용하여 성능을 판단하며, **과학습(overfitting)**을 지양해야한다.

2. 과학습(Overfitting)

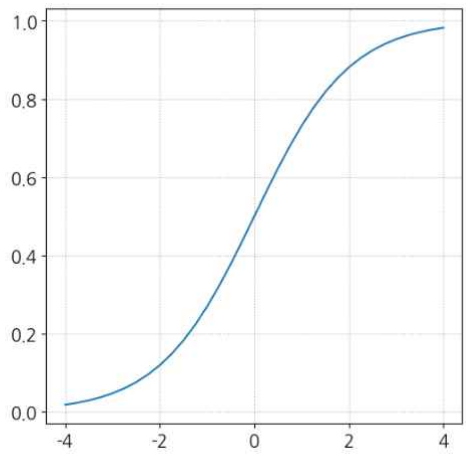
학습을 너무 많이 진행할 수록 오히려 검증 데이터의 정확도가 떨어지는 경우가 생긴다. <Fig.13>과 같이 너무 과하게 학습 데이터에 맞추기보단 적당한 선에서 모델을 학습시키는 것이 중요하다.



<Fig 13. 모델의 fitting 예시>

3. 시그모이드 함수

이진 분류에서 사용하는 예측함수로, 함수값을 확률로 해석하기 위하여 사용한다. 시그모이드함수는 다음과 같은 특징이 있다.



<Fig 14. Sigmoid함수의 형태>

- 항상 값이 증가함
- 0과 1사이의 값을 취함
- x=0 -> 0.5
- 그래프가 (0, 0.5) 기준 대칭
- $y = \dfrac{1}{1+e^{-z}}$

4. 교차 엔트로피 함수

이진 분류에서 사용하는 손실함수로, 모델이 출력하는 Ypred가 정답 레이블 y와 얼마나 가까운지를 측정한다. 수식은 다음과 같다.

L=−(ylogYpred+(1−y)log(1−Ypred))

y: 실제 정답 레이블 (0 또는 1)

Ypred: 모델이 예측한 확률값 (0 ~ 1 사이)

정답이 1일 때, −logYpred값이 최소가 되어야 함 → 즉, Ypred가 1에 가까워야 함

정답이 0일 때, −log(1−Ypred)값이 최소가 되어야 함 → 즉,

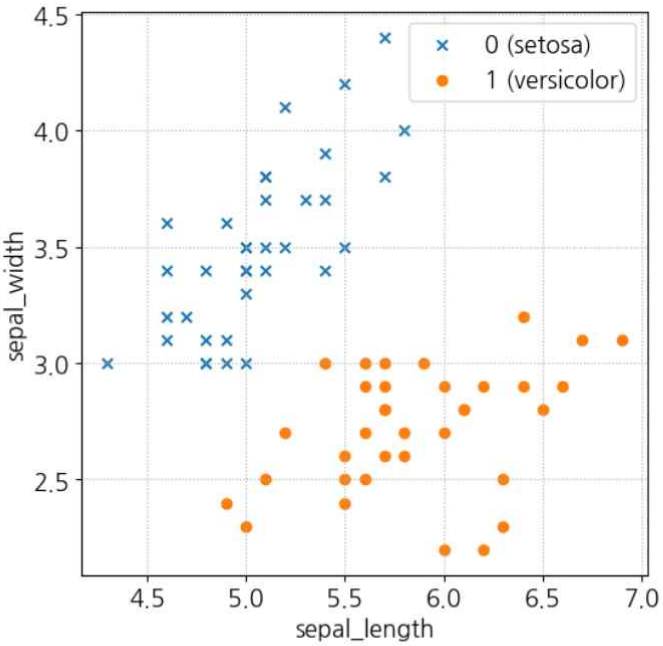
Ypred가 0에 가까워야 함

5. 붓꽃 데이터셋 전처리

붓꽃 데이터셋을 사용하여 이진 분류의 실습을 진행하였다.

데이터셋은 사이킷런에 내장되어있는 iris dataset을 사용하였다.

dataset은 serosa와 versicolor의 sepal length와 sepal width를 추출하여 훈련 데이터와 검증 데이터로 분할하였다.



<Fig 15. sepal_length VS sepal_width>

6. 이진 분류 실습

```python
# 모델 정의

# 입력 차원 수
n_input = x_train.shape[1]

# 출력 차원 수
n_output = 1

class Net(nn.Module):
    def __init__(self, n_input, n_output):
        super().__init__()
        self.l1 = nn.Linear(n_input, n_output)
        self.sigmoid = nn.Sigmoid()
        self.l1.weight.data.fill_(1.0)
        self.l1.bias.data.fill_(1.0)

    def forward(self,x):
        x1 = self.l1(x)
        x2 = self.sigmoid(x1)
        return x2

net = Net(n_input, n_output)

# 최적화 알고리즘과 손실 함수의 정의
criterion = nn.BCELoss()

# 학습률
lr = 0.01

# 최적화 함수 : 경사 하강법
optimizer = optim.SGD(net.parameters(), lr=lr)
```

```python
# 입력 데이터와 출력 데이터 텐서로 변환

inputs = torch.tensor(x_train).float()
labels = torch.tensor(y_train).float()

labels1 = labels.view((-1,1))

inputs_test = torch.tensor(x_test).float()
labels_test = torch.tensor(y_test).float()

labels1_test = labels_test.view((-1,1))
```

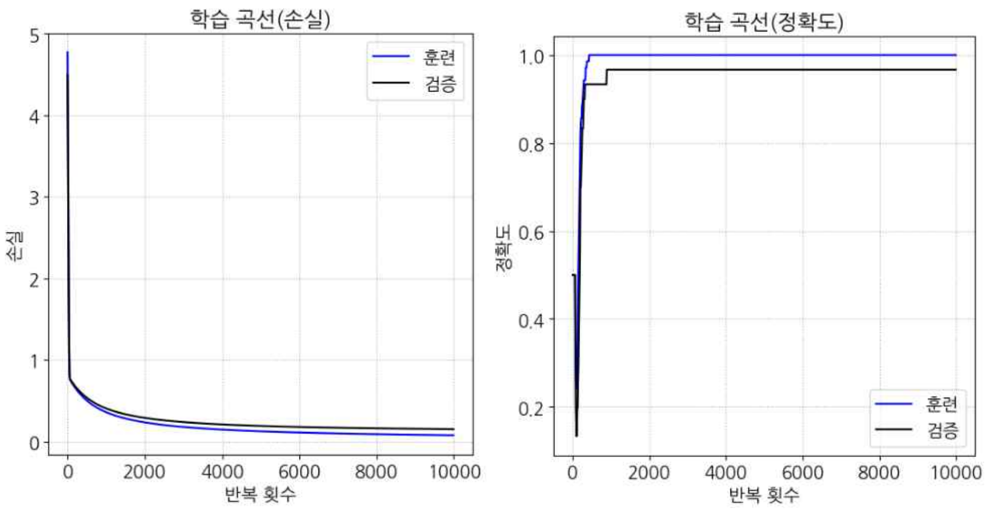
```python
# 반복 계산의 초기화 처리 정리

lr = 0.01
net = Net(n_input, n_output)
criterion = nn.BCELoss()
optimizer = optim.SGD(net.parameters(), lr=lr)
num_epochs = 10000
```

```python
history = np.zeros((0,5))

for epoch in range(num_epochs):
    # 훈련 페이즈



    # 경사값 초기화
    optimizer.zero_grad()

    # 예측 계산
    outputs = net(inputs)

    # 손실 계산
    loss = criterion(outputs, labels1)

    # 경사 계산
    loss.backward()

    # 파라미터 수정
    optimizer.step()

    # 손실 저장
    train_loss = loss.item()

    # 예측 라벨 계산
    predicted = torch.where(outputs<0.5, 0,1)

    # 정확도 계산
    train_acc = (predicted == labels1).sum() / len(y_train)
```

```
# 예측 페이즈



# 예측 계산
outputs_test = net(inputs_test)

# 손실 계산
loss_test = criterion(outputs_test, labels1_test)

# 손실 저장
val_loss = loss_test.item()

# 예측 라벨 계산
predicted_test = torch.where(outputs_test<0.5, 0,1)

# 정확도 계산
val_acc = (predicted_test == labels1_test).sum() / len(y_test)
```

7. 결과 확인

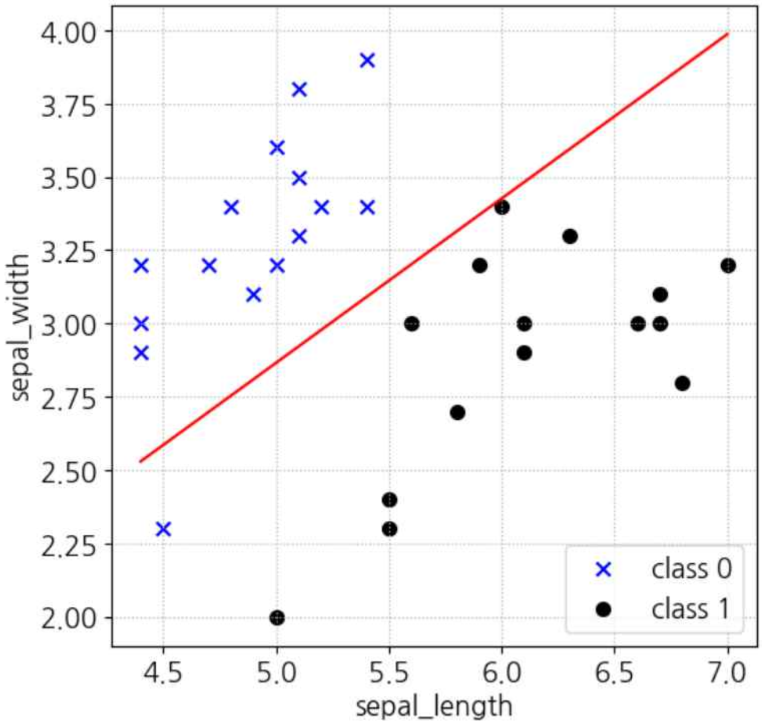
초기 상태에는 손실 : 4.49384 정확도 : 0.50000였으나, 최종 상태에는 손실 : 0.15395 정확도 : 0.96667로 높은 정확도를 보여준다. 또한 손실이 수렴하면서,정확도도 1에 가까워지는 모습을 확인할 수 있는데, 이는 모델이 안정적으로 학습했음을 보여준다.



<Fig 16. Loss와 Accuracy 곡선>

8. 결정 경계(Hyperplane)

학습한 모델의 결정 경계(Hyperplane)는 <Fig. 17>과 같다.

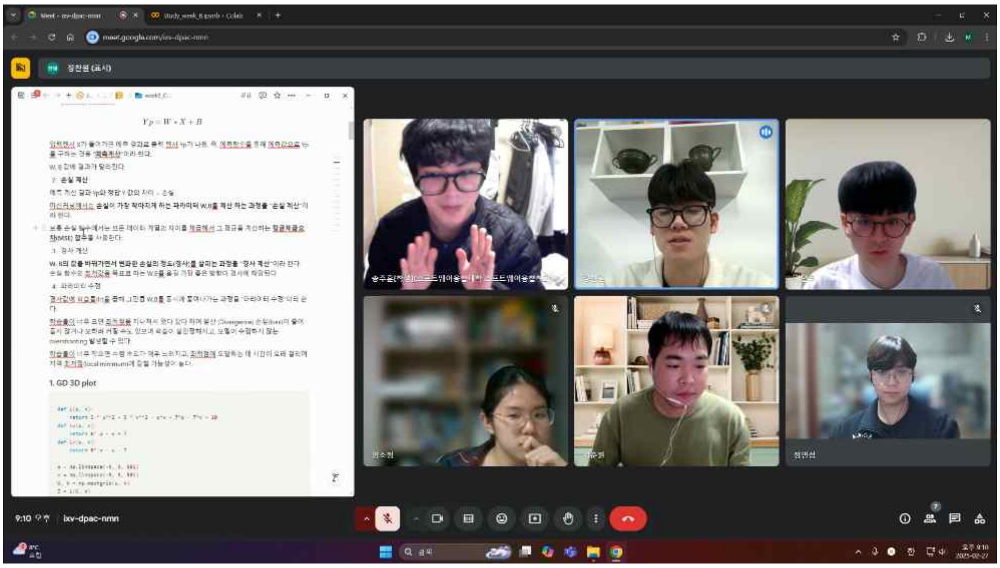


<Fig 17. 모델의 결정 경계(Hyperplane)>

9. 이진 분류 과제 풀이

스터디 당일에는 정규 수업에서 아직 이진 분류를 다루지 않아, 이진 분류 정리를 담당한 인원 중, 서준원님께서 문제 풀이를 진행하였다.

활동평가	교재의 내용을 요약 및 발표하며, 정규 수업의 내용을 복기할 수 있었다. 이를 통해 머신러닝에서 사용되는 용어와 파이토치를 이용한 머신러닝의 흐름의 이해도를 높일 수 있었다.

과제	스터디 자료 업로드 및 복기
향후 계획	교재 기준 7단원: 서준원, 강인우 8단원: 송주훈, 정민섭 9단원: 정찬원, 임소정 각자 맡은 파트 정리 후 발표
첨부 자료	

	10:10 오후 | ixv-dpac-nmn