

Revenue growth divisions.

TYU division

FRT division

Projected sales of main products in 2013

# Sentiment Analysis on Classification

## MLP LAB 임경태, 임현석

	TYU division			FRT division		
GHT	254	550	254	274	154	415
RDW	650	320	754	273	825	154
TRG	241	450	144	364	954	174
RTG	254	650	874	657	125	274
WCF	794	145	124	752	741	759
HBT	452	794	954	341	741	345



# Contents

- 
- [\*\*01 Logistic Regression\*\*](#)
  - [\*\*02 Logistic Regression 구현\*\*](#)
  - [\*\*03 Sentiment Analysis\*\*](#)
  - [\*\*04 Logistic Classification\*\*](#)
  - [\*\*05 Logistic Classification 구현\*\*](#)
  - [\*\*06 레스토랑 리뷰 감성 분석하기\*\*](#)

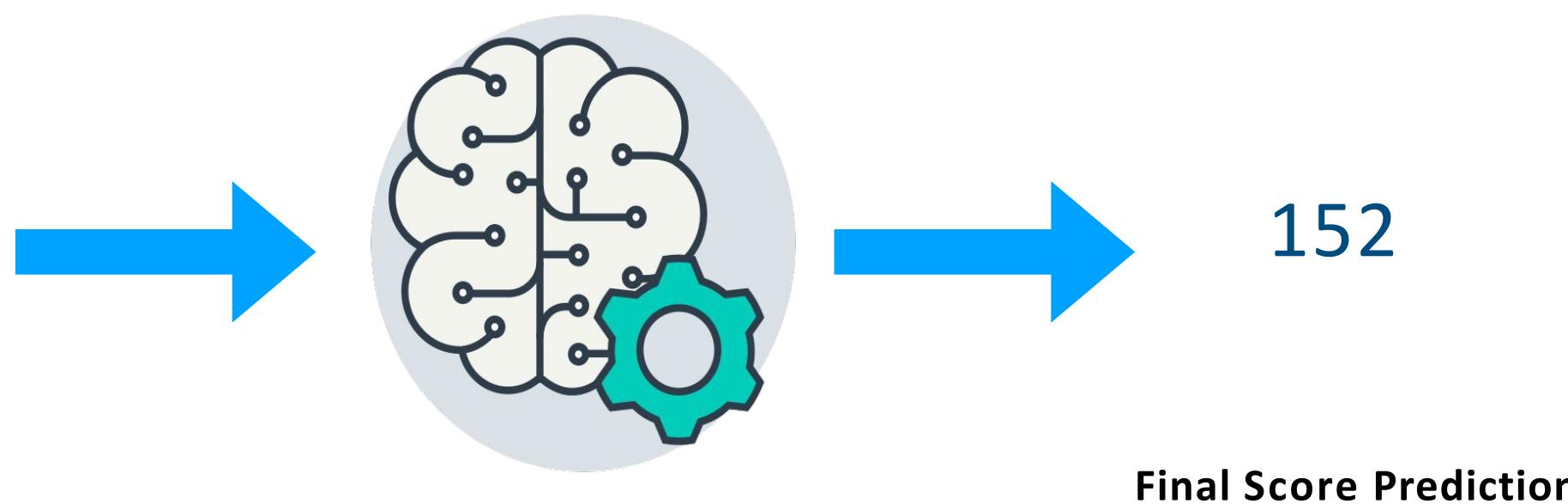


# 01. Logistic Regression

# Recap

- Multi-variable Linear Regression

Quiz1: 73  
Quiz2: 80  
Quiz3: 75  
**Quiz Scores**



$$H(x) = Wx + b$$

# 의문점

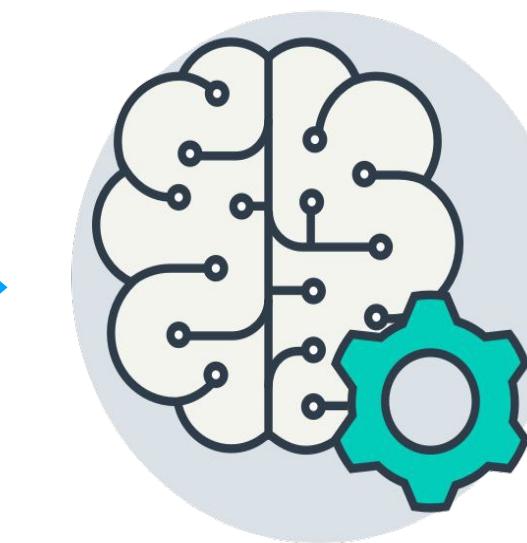
- 이런 경우는 어떻게 푸나?

Quiz1: 73

Quiz2: 80

Quiz3: 75

**Quiz Scores**



불합격  
합격

**Class Prediction**

$$H(x) = Wx + b$$

## 해결

## • Logistic Regression (LR) 모델

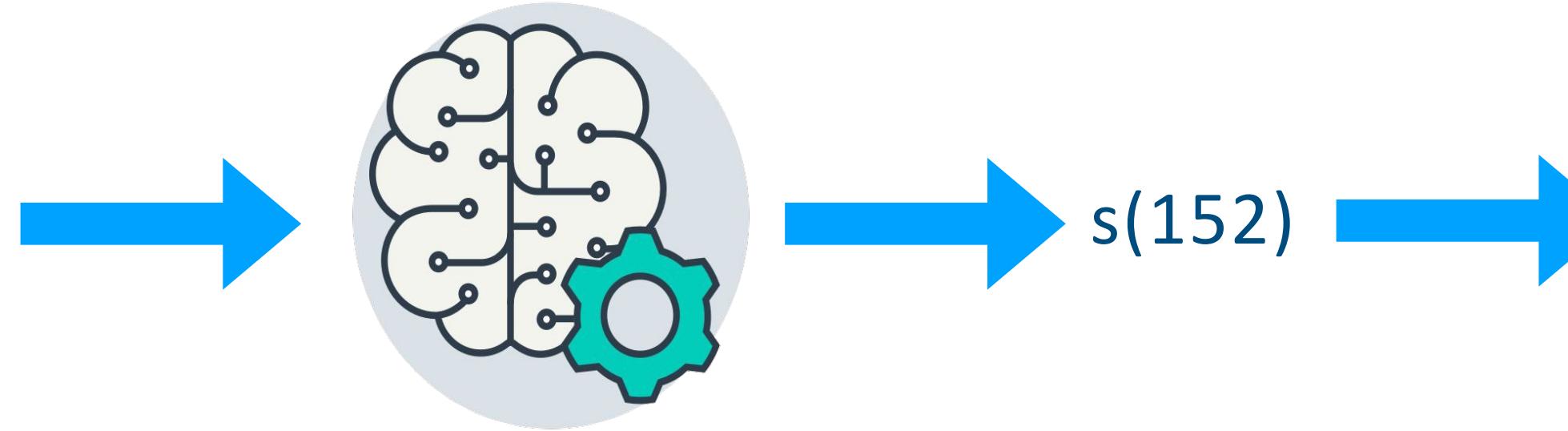
- 회귀에서 분류 문제로 바꾸자! 간단하게 모델의 출력을 합격이면 1에 가깝게 불합격은 0에 가깝게 출력!

Quiz1: 73

Quiz2: 80

Quiz3: 75

Quiz Scores



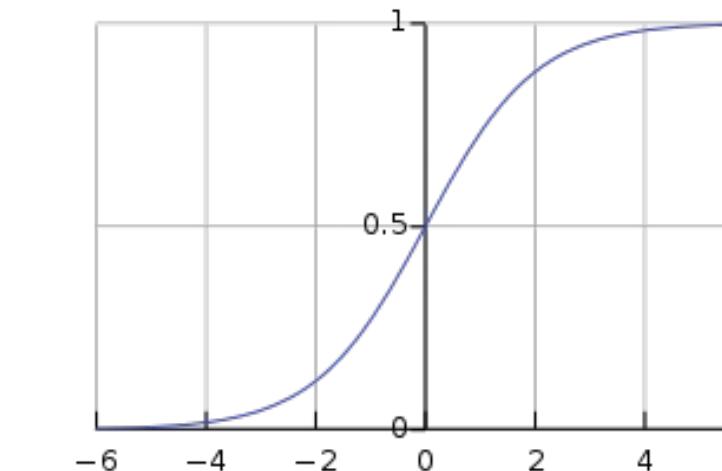
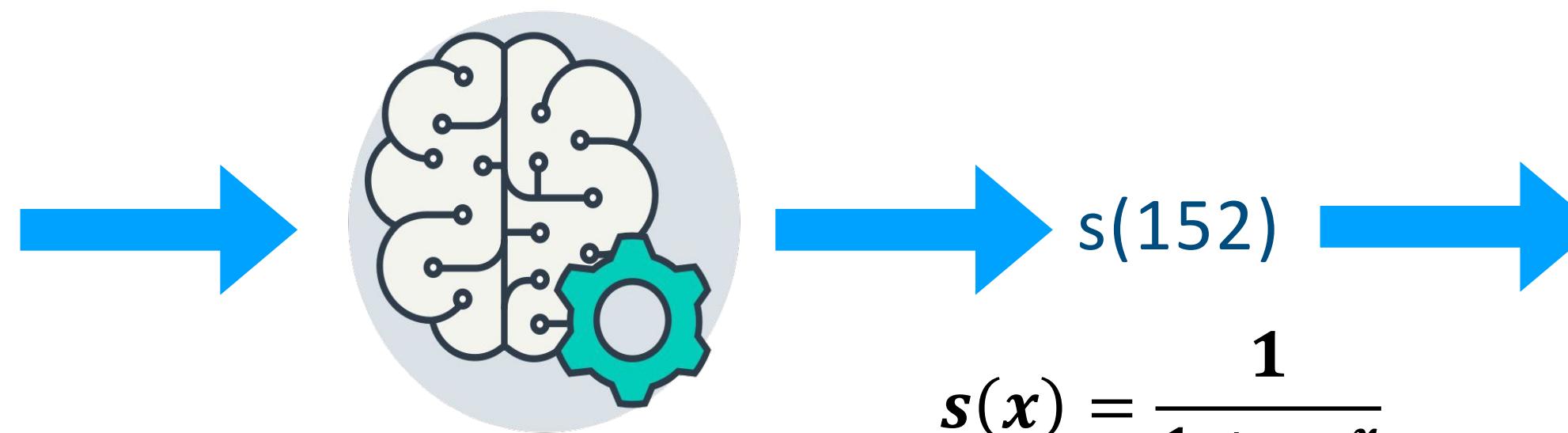
$$H(x) = Wx + b$$

$$H(x) = s(Wx + b)$$

# Logistic Regression (LR)

- Sigmoid function
  - 회귀에서 분류 문제로 변환해주는 함수 → 모델의 출력을 0~1사이 값으로 변경해줌

Quiz1: 73  
 Quiz2: 80  
 Quiz3: 75  
**Quiz Scores**



$$H(x) = \frac{1}{1 + e^{-(Wx+b)}}$$

# Logistic Regression (LR)

- Negative Log Likely-hood cost 함수

- $Cost(W, b) = -\frac{1}{m} \sum_{i=1}^m y \log(H_\theta(x)) + (1 - y) \log(1 - H_\theta(x))$

$$\left. \begin{aligned} \mathbb{P}(y = 1|z) &= \sigma(z) = \frac{1}{1 + e^{-z}} \\ \mathbb{P}(y = 0|z) &= 1 - \sigma(z) = \frac{1}{1 + e^z} \end{aligned} \right\} \mathbb{P}(y|z) = \sigma(z)^y (1 - \sigma(z))^{1-y}$$

# Logistic Regression (LR)

- Gradient Descent 함수

---

## Weight Update via Gradient Descent

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

- 
- $\alpha$ : Learning rate

# 02. Logistic Regression

구현

# Logistic Regression (LR) with Pytorch

- 데이터 준비

```
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]  
y_data = [[0], [0], [0], [1], [1], [1]]
```

Consider the following classification problem: given the number of hours each student spent watching the lecture and working in the code lab, predict whether the student passed or failed a course. For example, the first (index 0) student watched the lecture for 1 hour and spent 2 hours in the lab session ([1, 2]), and ended up failing the course ([0]).

```
x_train = torch.FloatTensor(x_data)  
y_train = torch.FloatTensor(y_data)
```

As always, we need these data to be in `torch.Tensor` format, so we convert them.

```
print(x_train.shape)  
print(y_train.shape)
```

```
torch.Size([6, 2])  
torch.Size([6, 1])
```

# Logistic Regression (LR) with Pytorch

- Computing the Hypothesis

$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

PyTorch has a `torch.exp()` function that resembles the exponential function.

```
print('e^1 equals: ', torch.exp(torch.FloatTensor([1])))  
e^1 equals:  tensor([2.7183])
```

We can use it to compute the hypothesis function conveniently.

```
w = torch.zeros((2, 1), requires_grad=True)  
b = torch.zeros(1, requires_grad=True)  
  
hypothesis = 1 / (1 + torch.exp(-(x_train.matmul(w) + b)))  
  
print(hypothesis)  
print(hypothesis.shape)  
  
tensor([[0.5000],  
        [0.5000],  
        [0.5000],  
        [0.5000],  
        [0.5000],  
        [0.5000]], grad_fn=<MulBackward0>)  
torch.Size([6, 1])
```

# Logistic Regression (LR) with Pytorch

- Computing the Cost Function

$$cost(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) \log(1 - H(x))$$

We want to measure the difference between `hypothesis` and `y_train`.

```
print(hypothesis)
print(y_train)

tensor([[0.5000],
        [0.5000],
        [0.5000],
        [0.5000],
        [0.5000],
        [0.5000]], grad_fn=<SigmoidBackward>)
tensor([[0.],
        [0.],
        [0.],
        [1.],
        [1.],
        [1.]])
```

# Logistic Regression (LR) with Pytorch

- Computing the Cost Function

For one element, the loss can be computed as follows:

```
- (y_train[0] * torch.log(hypothesis[0]) +  
  (1 - y_train[0]) * torch.log(1 - hypothesis[0]))  
  
tensor([0.6931], grad_fn=<NegBackward>)
```

# Logistic Regression (LR) with Pytorch

- Computing the Cost Function from Scratch

To compute the losses for the entire batch, we can simply input the entire vector.

```
losses = -(y_train * torch.log(hypothesis) +
           (1 - y_train) * torch.log(1 - hypothesis))
print(losses)

tensor([0.6931],
      [0.6931],
      [0.6931],
      [0.6931],
      [0.6931],
      [0.6931]), grad_fn=<NegBackward>)
```

Then, we just `.mean()` to take the mean of these individual losses.

```
cost = losses.mean()
print(cost)

tensor(0.6931, grad_fn=<MeanBackward1>)
```

# Logistic Regression (LR) with Pytorch

- Computing the Cost Function using the Pytorch Functional package

```
F.binary_cross_entropy(hypothesis, y_train)  
tensor(0.6931, grad_fn=<BinaryCrossEntropyBackward>)
```

# Logistic Regression (LR) with Pytorch

- Whole Training Procedure (w/o Class)

```
# 모델 초기화
W = torch.zeros((2, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)
# optimizer 설정
optimizer = optim.SGD([W, b], lr=1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # Cost 계산
    hypothesis = torch.sigmoid(x_train.matmul(W) + b) # or .mm or @
    cost = F.binary_cross_entropy(hypothesis, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 100번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{}, Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()))
    ))
```

```
Epoch    0/1000 Cost: 0.693147
Epoch   100/1000 Cost: 0.134722
Epoch   200/1000 Cost: 0.080643
Epoch   300/1000 Cost: 0.057900
Epoch   400/1000 Cost: 0.045300
Epoch   500/1000 Cost: 0.037261
Epoch   600/1000 Cost: 0.031672
Epoch   700/1000 Cost: 0.027556
Epoch   800/1000 Cost: 0.024394
Epoch   900/1000 Cost: 0.021888
Epoch  1000/1000 Cost: 0.019852
```

# Logistic Regression (LR) with Pytorch

- Evaluation

After we finish training the model, we want to check how well our model fits the training set.

```
hypothesis = torch.sigmoid(x_train.matmul(w) + b)
print(hypothesis[:5])

tensor([[0.4103],
       [0.9242],
       [0.2300],
       [0.9411],
       [0.1772]], grad_fn=<SliceBackward>)
```

# Logistic Regression (LR) with Pytorch

- Evaluation

We can change **hypothesis** (real number from 0 to 1) to **binary predictions** (either 0 or 1) by comparing them to 0.5.

```
prediction = hypothesis >= torch.FloatTensor([0.5])
print(prediction[:5])

tensor([[0],
       [1],
       [0],
       [1],
       [0]], dtype=torch.uint8)
```

# Logistic Regression (LR) with Pytorch

- Evaluation

Then, we compare it with the correct labels `y_train`.

```
print(prediction[:5])
print(y_train[:5])

tensor([[0],
       [1],
       [0],
       [1],
       [0]], dtype=torch.uint8)
tensor([[0.],
       [1.],
       [0.],
       [1.],
       [0.]])
```

# Logistic Regression (LR) with Pytorch

- Evaluation

```
correct_prediction = prediction.float() == y_train
print(correct_prediction[:5])

tensor([[1],
       [1],
       [1],
       [1],
       [1]], dtype=torch.uint8)
```

# Logistic Regression (LR) with Pytorch

- Higher Implementation with Class

```
class BinaryClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(8, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        return self.sigmoid(self.linear(x))

model = BinaryClassifier()
```

# Logistic Regression (LR) with Pytorch

- Higher Implementation with Class

```
# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=1)

nb_epochs = 100
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = model(x_train)

    # cost 계산
    cost = F.binary_cross_entropy(hypothesis, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 20번마다 로그 출력
    if epoch % 10 == 0:
        prediction = hypothesis >= torch.FloatTensor([0.5])
        correct_prediction = prediction.float() == y_train
        accuracy = correct_prediction.sum().item() / len(correct_prediction)
        print('Epoch {:4d}/{} Cost: {:.6f} Accuracy {:.2f}%'.format(
            epoch, nb_epochs, cost.item(), accuracy * 100,
        ))
```

```
Epoch  0/100 Cost: 0.704829 Accuracy 45.72%
Epoch 10/100 Cost: 0.572391 Accuracy 67.59%
Epoch 20/100 Cost: 0.539563 Accuracy 73.25%
Epoch 30/100 Cost: 0.520042 Accuracy 75.89%
Epoch 40/100 Cost: 0.507561 Accuracy 76.15%
Epoch 50/100 Cost: 0.499125 Accuracy 76.42%
Epoch 60/100 Cost: 0.493177 Accuracy 77.21%
Epoch 70/100 Cost: 0.488846 Accuracy 76.81%
Epoch 80/100 Cost: 0.485612 Accuracy 76.28%
Epoch 90/100 Cost: 0.483146 Accuracy 76.55%
```

# 03. Sentiment Analysis

# Sentiment Analysis

- 감성분석: 문장 및 문서에 대해 긍정/부정/중립 등의 극성을 분류하는 기술

 Apple 아이패드 에어 5세대 M1 WIFI 64G 스페이스 그레이 (MM9C3KH...  
최저 837,000원 무료배송 최저가 사려가기

판매처 94 제품정보 블로그리뷰 **쇼핑몰리뷰 1,445** AiITEMS추천

**★★★★★ 5** 하이마트쇼핑몰 · r\*u\*\*\*\*\* · 22.07.06.

**오래전부터 갖고 싶었던 아이패드**

오래전부터 갖고 싶었던 아이패드.  
몇일간 검색해본 결과 하이마트쇼핑몰에서 사는게 가격도 저렴하고 믿을 수가 있다는 생각에 구입 결정!  
제품의 품질은 역시 애플 제품답게 훌륭하네요.  
앞으로 제가 잘 쓰기만 하면 될 듯.  
무엇보다 포장에 대단히 신경을 써주신 것 같아 좋았습니다. 제품이 박스에서 충격받지 않도록 종이 포장으로 칭칭 단단하게 감겨 있어 포장에 감동받기



리뷰펼치기 ▼

---

**★★★★★ 5** 하이마트쇼핑몰 · u\*u\*\*\*\*\* · 23.01.29.

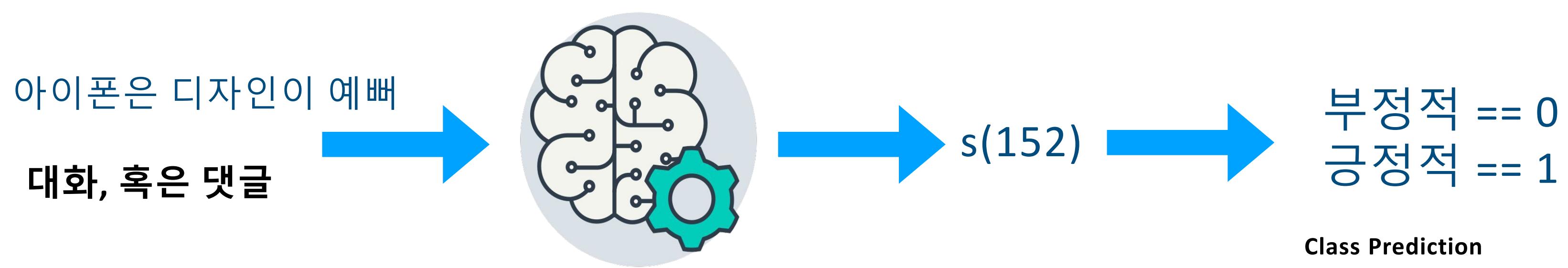
**딸아이가 공부.인강들을때 필요**

딸아이가 공부.인강들을때 필요하대서 구입. 배송은 이틀만에 왔구요. 안포장은 잘되있는데...겉포장박스가 허술해서...조금그랬구요...패드는 하자없는  
제품으로 잘 도착했어요. 카드청구할인으로 조금 저렴하게 구입했어요. 추가로 필요한 애플케어랑...액서사리값이 비싸네요. 그레이색상 질리지않고 무난  
하니 잘 선택했구요.필름추가구입해서 바로 붙여서 사용했어요.



# A simple Sentiment Analysis on LR

- Binary Classification 문제로 간단한 모델링 해보자



# A simple Sentiment Analysis on LR

- Binary Classification 문제로 간단한모델링 해보자
  - Kaggle 의 Bag of Words Meets Bags of Popcorn 예제 진행
  - AWE Linear Regression 코드를 재활용하면 쉽지 않을까?

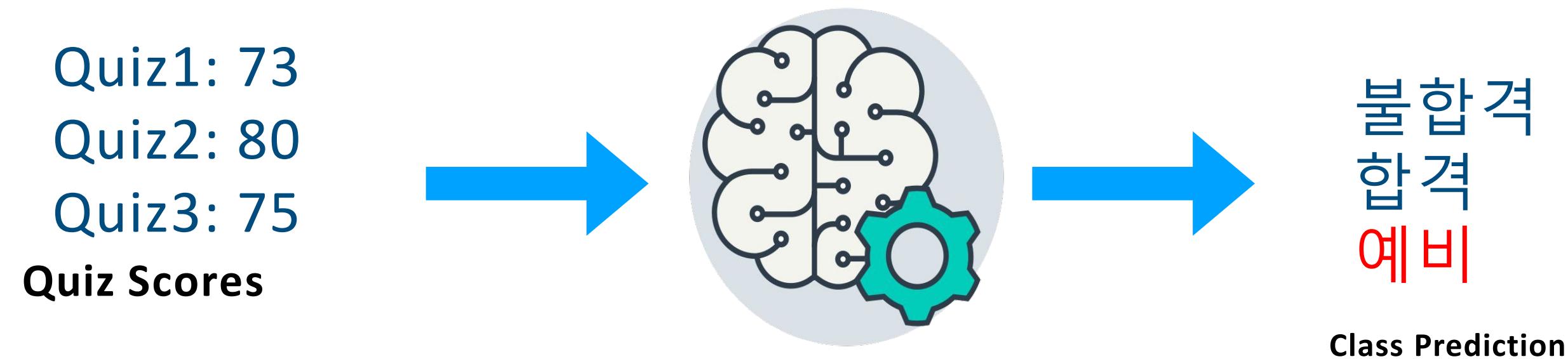
<https://www.kaggle.com/competitions/word2vec-nlp-tutorial/overview>

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

# 04. Multinomial Classification

### 의문점

- 예측이 Multi-class의 경우는 어떻게 푸냐?
  - 이런 경우를 Multinomial Classification 문제라고 한다.

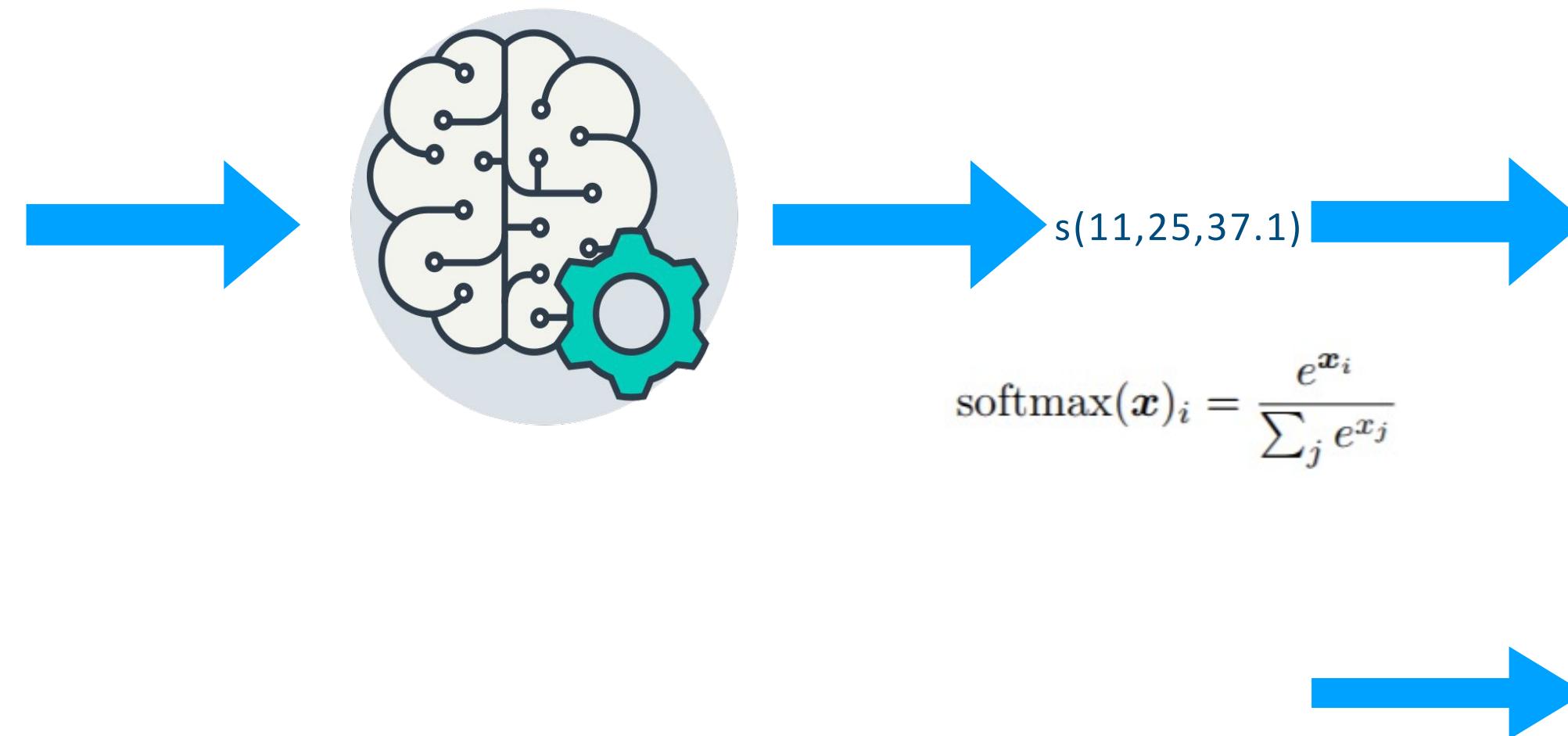


$$H(x) = Wx + b$$

### 해결

- Softmax를 이용한 확률분포 출력
  - 회귀에서 분류 문제로 바꾸자! 간단하게 모델의 출력을 Class의 개수만큼의 확률분포로 만들자!

Quiz1: 73  
Quiz2: 80  
Quiz3: 75  
**Quiz Scores**



# Multinomial Classification

- 손실함수: Cross Entropy

- Cross Entropy 두 확률분포 사이의 거리를 계산하는 방법

$$H(P, Q) = -\mathbb{E}_{x \sim P(x)}[\log Q(x)] = -\sum_{x \in \mathcal{X}} P(x) \log Q(x)$$

- $CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{N_c} y_i \log(\hat{y}_i)$

```
import torch
def cross_ent_loss(x,y):
    return (y * -torch.log(x))

a = torch.FloatTensor([[0.1, 0.2, 0.7]])
b = torch.FloatTensor([[0, 0, 1]])
cross_ent_loss(a,b)
```

▼ 1. Basic usage of the CE loss

```
[62] import torch
      import matplotlib.pyplot as plt
      import numpy as np

[63] def cross_ent_loss(x,y):
      return (y * -torch.log(x))

[64] # if we have two different prob distributions below:
      a = torch.FloatTensor([[0.1, 0.2, 0.7]])
      b = torch.FloatTensor([[0, 0, 1]])
      cross_ent_loss(a,b)

[65] tensor([[0.0000, 0.0000, 0.3567]])

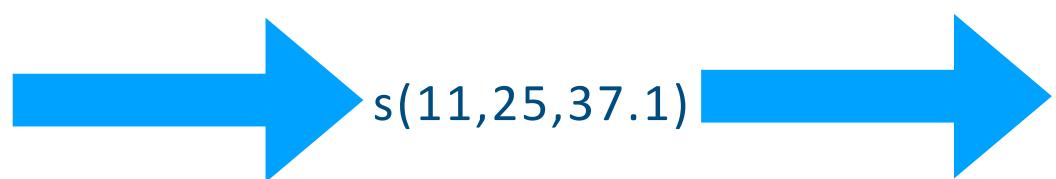
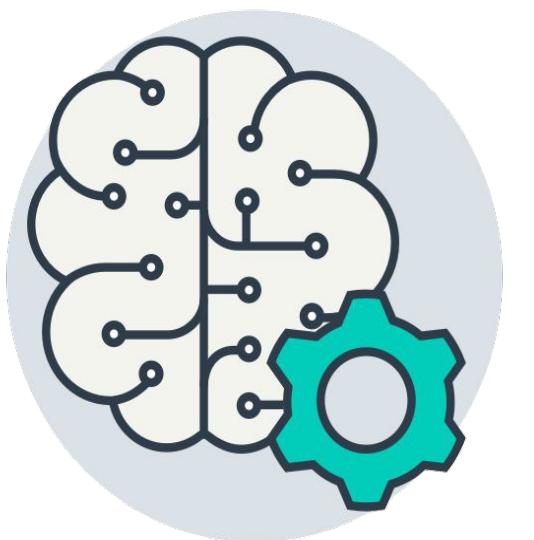
[66] # Even two distributions are almost identical, it returns marginal loss values
      a = torch.FloatTensor([[0, 0, 0, 0, 0.01, 0.99]])
      b = torch.FloatTensor([[0, 0, 0, 0, 0, 1]])
      cross_ent_loss(a,b)

[67] tensor([[    nan,     nan,     nan,     nan, 0.0000, 0.0101]])
```

# Multinomial Classification

- 손실함수: Cross Entropy
  - Cross Entropy 두 확률분포 사이의 거리를 계산하는 방법

$$CE(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{i=1}^{N_c} y_i \log(\hat{y}_i)$$



$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

불합격 == 0.1  
합격 == 0.2  
예비 == 0.7  
**Class Prediction**

CE로  
거리비교  
= loss

불합격 == 0  
합격 == 1  
예비 == 0  
**Target (실제정답)**

# Multinomial Classification

- Cross Entropy 구현

Data 준비하기

- 랜덤으로 Dummy data를 만들고 이를 Regression 결과로 가정해 z로 저장하자
- Regression 결과를 Multinomial Classification으로 변경하려면 Softmax가 필요하지!

For multi-class classification, we use the cross entropy loss.

$$L = \frac{1}{N} \sum -y \log(\hat{y})$$

where  $\hat{y}$  is the predicted probability and  $y$  is the correct probability (0 or 1).

```
z = torch.rand(3, 5, requires_grad=True)
hypothesis = F.softmax(z, dim=1)
print(hypothesis)

tensor([[0.2645, 0.1639, 0.1855, 0.2585, 0.1277],
        [0.2430, 0.1624, 0.2322, 0.1930, 0.1694],
        [0.2226, 0.1986, 0.2326, 0.1594, 0.1868]], grad_fn=<SoftmaxBackward0>)

y = torch.randint(5, (3,)).long()
print(y)

tensor([0, 2, 1])
```

# Multinomial Classification

- Cross Entropy 구현
  - 실제 정답 데이터를 확률분포인 one-hot encoding으로 변경하자
- Loss 계산하기: 실제정답, 예측값의 두 확률분포를 CE Loss로 계산하자

```
y_one_hot = torch.zeros_like(hypothesis)
y_one_hot.scatter_(1, y.unsqueeze(1), 1)
```

```
tensor([[1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 1., 0., 0., 0.]])
```

```
cost = (y_one_hot * -torch.log(hypothesis)).sum(dim=1).mean()
print(cost)
```

```
tensor(1.4689, grad_fn=<MeanBackward1>)
```

# Multinomial Classification

- Cross Entropy 구현 (low, high 버전)

```
# Low level
(y_one_hot * -torch.log(F.softmax(z, dim=1))).sum(dim=1).mean()

tensor(1.4689, grad_fn=<MeanBackward1>)
```

```
# High level
F.nll_loss(F.log_softmax(z, dim=1), y)

tensor(1.4689, grad_fn=<NllLossBackward>)
```

PyTorch also has `F.cross_entropy` that combines `F.log_softmax()` and `F.nll_loss()`.

```
F.cross_entropy(z, y)

tensor(1.4689, grad_fn=<NllLossBackward>)
```

# Multinomial Classification

- Gradient Descent 함수

---

## Weight Update via Gradient Descent

$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

- 
- $\alpha$ : Learning rate

# 05. Multinomial Classification

구현

# Cross Entropy Loss (Low-level)

```
y_one_hot = torch.zeros_like(hypothesis)
y_one_hot.scatter_(1, y.unsqueeze(1), 1)
```

```
tensor([[1., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0.],
        [0., 1., 0., 0., 0.]])
```

```
cost = (y_one_hot * -torch.log(hypothesis)).sum(dim=1).mean()
print(cost)
```

```
tensor(1.4689, grad_fn=<MeanBackward1>)
```

# Multinomial Classification

- 데이터 준비

```
x_train = [[1, 2, 1, 1],  
           [2, 1, 3, 2],  
           [3, 1, 3, 4],  
           [4, 1, 5, 5],  
           [1, 7, 5, 5],  
           [1, 2, 5, 6],  
           [1, 6, 6, 6],  
           [1, 7, 7, 7]]  
y_train = [2, 2, 2, 1, 1, 1, 0, 0]  
x_train = torch.FloatTensor(x_train)  
y_train = torch.LongTensor(y_train)
```

# Multinomial Classification

- 전체 학습 (low-level)

```
# 모델 초기화
W = torch.zeros((4, 3), requires_grad=True)
b = torch.zeros(1, requires_grad=True)
# optimizer 설정
optimizer = optim.SGD([W, b], lr=0.1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # Cost 계산 (1)
    hypothesis = F.softmax(x_train.matmul(W) + b, dim=1) # or .mm or @
    y_one_hot = torch.zeros_like(hypothesis)
    y_one_hot.scatter_(1, y_train.unsqueeze(1), 1)
    cost = (y_one_hot * -torch.log(F.softmax(hypothesis, dim=1))).sum(dim=1

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 100번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()
        ))
```

Epoch 0/1000 Cost: 1.098612  
Epoch 100/1000 Cost: 0.901535  
Epoch 200/1000 Cost: 0.839114  
Epoch 300/1000 Cost: 0.807826  
Epoch 400/1000 Cost: 0.788472  
Epoch 500/1000 Cost: 0.774822  
Epoch 600/1000 Cost: 0.764449  
Epoch 700/1000 Cost: 0.756191  
Epoch 800/1000 Cost: 0.749398  
Epoch 900/1000 Cost: 0.743671  
Epoch 1000/1000 Cost: 0.738749

# Multinomial Classification

- 전체 학습 (high-level)

```

class SoftmaxClassifierModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(4, 3) # Output 0/ 3!

    def forward(self, x):
        return self.linear(x)

model = SoftmaxClassifierModel()

```

```

# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=0.1)

nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    prediction = model(x_train)

    # cost 계산
    cost = F.cross_entropy(prediction, y_train)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    # 20번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{}, Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()))

```

```

Epoch 0/1000 Cost: 1.849513
Epoch 100/1000 Cost: 0.689894
Epoch 200/1000 Cost: 0.609259
Epoch 300/1000 Cost: 0.551218
Epoch 400/1000 Cost: 0.500141
Epoch 500/1000 Cost: 0.451947
Epoch 600/1000 Cost: 0.405051
Epoch 700/1000 Cost: 0.358733
Epoch 800/1000 Cost: 0.312912
Epoch 900/1000 Cost: 0.269521
Epoch 1000/1000 Cost: 0.241922

```

# 06. 예제 : 레스토랑 리뷰 감성 분석하기

## 01 옐프 리뷰 데이터셋

- 레스토랑 등급 중 별 1개, 2개는 음성 클래스 / 별 3개, 4개는 양성 클래스
- 라이트 버전 사용 : 데이터셋 전체 훈련 샘플의 10%만 사용
  - 훈련 - 테스트 반복이 빠름 → 실험 속도 향상
  - 전체 데이터를 사용할 때보다 모델이 낮은 정확도 달성

## 06 레스토랑 리뷰 감성 분석하기

```
# 훈련, 검증, 테스트를 만들기 위해 별점을 기준으로 나눕니다
by_rating = collections.defaultdict(list)
for _, row in train_reviews.iterrows():
    by_rating[row.rating].append(row.to_dict())

# 분할 데이터를 만듭니다.
final_list = []
np.random.seed(args.seed)

for _, item_list in sorted(by_rating.items()):
    np.random.shuffle(item_list)

    n_total = len(item_list)
    n_train = int(args.train_proportion * n_total)
    n_val = int(args.val_proportion * n_total)

    # 데이터 포인터에 분할 속성을 추가합니다
    for item in item_list[:n_train]:
        item['split'] = 'train'

    for item in item_list[n_train:n_train+n_val]:
        item['split'] = 'val'

    # 최종 리스트에 추가합니다
    final_list.extend(item_list)

# 리뷰를 전처리합니다
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"([.,!?])", r" \1 ", text)
    text = re.sub(r"^[^a-zA-Z.,!?]+", " ", text)
    return text

final_reviews.review = final_reviews.review.apply(preprocess_text)
```

- 모델 파라미터를 추정하는 데 훈련 세트를 사용
- 하이퍼파라미터를 선택하는 데 검증 세트를 사용
- 마지막 평가와 보고에 테스트 세트 사용

## 06 레스토랑 리뷰 감성 분석하기

---

	rating	review	split
0	negative	terrible place to work for i just heard a stor...	train
1	negative	hours , minutes total time for an extremely s...	train
2	negative	my less than stellar review is for service . w...	train
3	negative	i m granting one star because there s no way t...	train
4	negative	the food here is mediocre at best . i went aft...	train
...	...	...	...
55995	positive	great food . wonderful , friendly service . i ...	test
55996	positive	charlotte should be the new standard for moder...	test
55997	positive	get the encore sandwich ! ! make sure to get i...	test
55998	positive	i m a pretty big ice cream gelato fan . pretty...	test
55999	positive	where else can you find all the parts and piec...	test

[56000 rows x 3 columns]

<데이터 셋의 형태>

## 02 파이토치 데이터셋 이해하기

- 해당 데이터셋은 리뷰를 공백을 기준으로 나눠서 토큰 리스트를 얻을 수 있음.
- 데코레이터는 클래스 인스턴스를 만들지 않고 호출할 수 있는 정적 메서드 생성

```
class ReviewDataset(Dataset):
    def __init__(self, review_df, vectorizer):
        """
        매개변수:
        review_df (pandas.DataFrame): 데이터셋
        vectorizer (ReviewVectorizer): ReviewVectorizer 객체
        """
        self.review_df = review_df
        self._vectorizer = vectorizer

        self.train_df = self.review_df[self.review_df.split=='train']
        self.train_size = len(self.train_df)

        self.val_df = self.review_df[self.review_df.split=='val']
        self.validation_size = len(self.val_df)

        self.test_df = self.review_df[self.review_df.split=='test']
        self.test_size = len(self.test_df)

        self._lookup_dict = {'train': (self.train_df, self.train_size),
                            'val': (self.val_df, self.validation_size),
                            'test': (self.test_df, self.test_size)}

        self.set_split('train')

    def set_split(self, split):
        if split == 'train':
            self.train_df = self.review_df[self.review_df.split=='train']
            self.validation_size = len(self.val_df)
            self.test_size = len(self.test_df)
            self._lookup_dict = {'train': (self.train_df, self.train_size),
                                'val': (self.val_df, self.validation_size),
                                'test': (self.test_df, self.test_size)}
        elif split == 'val':
            self.train_df = self.review_df[self.review_df.split=='train']
            self.validation_size = len(self.val_df)
            self.test_size = len(self.test_df)
            self._lookup_dict = {'train': (self.train_df, self.train_size),
                                'val': (self.val_df, self.validation_size),
                                'test': (self.test_df, self.test_size)}
        elif split == 'test':
            self.train_df = self.review_df[self.review_df.split=='train']
            self.validation_size = len(self.val_df)
            self.test_size = len(self.test_df)
            self._lookup_dict = {'train': (self.train_df, self.train_size),
                                'val': (self.val_df, self.validation_size),
                                'test': (self.test_df, self.test_size)}```
```

생성자

```
@classmethod
def load_dataset_and_make_vectorizer(cls, review_csv):
    """
    데이터셋을 로드하고 새로운 ReviewVectorizer 객체를 만듭니다.
    """

    매개변수:
    review_csv (str): 데이터셋의 위치
    반환값:
    ReviewDataset의 인스턴스
    """

    review_df = pd.read_csv(review_csv)
    train_review_df = review_df[review_df.split=='train']
    return cls(review_df, ReviewVectorizer.from_dataframe(train_review_df))

@classmethod
def load_dataset_and_load_vectorizer(cls, review_csv, vectorizer_filepath):
    """
    데이터셋을 로드하고 새로운 ReviewVectorizer 객체를 만듭니다.
    캐시된 ReviewVectorizer 객체를 재사용할 때 사용합니다.
    """

    매개변수:
    review_csv (str): 데이터셋의 위치
    vectorizer_filepath (str): ReviewVectorizer 객체의 저장 위치
    반환값:
    ReviewDataset의 인스턴스
    """

    review_df = pd.read_csv(review_csv)
    vectorizer = cls.load_vectorizer_only(vectorizer_filepath)
    return cls(review_df, vectorizer)
```

데이터셋 로드 후 Vectorizer 객체 생성

## 06 레스토랑 리뷰 감성 분석하기

```
def set_split(self, split="train"):
    """ 데이터프레임에 있는 열을 사용해 분할 세트를 선택합니다

    매개변수:
        split (str): "train", "val", "test" 중 하나
    """
    self._target_split = split
    self._target_df, self._target_size = self._lookup_dict[split]
```

데이터 분할

```
def __getitem__(self, index):
    """ 파이토치 데이터셋의 주요 진입 메서드

    매개변수:
        index (int): 데이터 포인트의 인덱스
    반환값:
        데이터 포인트의 특성(x_data)과 레이블(y_target)로 이루어진 딕셔너리
    """
    row = self._target_df.iloc[index]

    review_vector = \
        self._vectorizer.vectorize(row.review)
    rating_index = \
        self._vectorizer.rating_vocab.lookup_token(row.rating)

    return {'x_data': review_vector,
            'y_target': rating_index}
```

데이터셋 진입 메서드

```
def get_num_batches(self, batch_size):
    """ 배치 크기가 주어지면 데이터셋으로 만들 수 있는 배치 개수를 반환합니다

    매개변수:
        batch_size (int)
    반환값:
        배치 개수
    """
    return len(self) // batch_size
```

배치 개수 반환

## 03 Vocabulary, Vectorizer, DataLoader 클래스

1. 각 토큰을 정수에 맵핑
2. 이 맵핑을 각 데이터 포인터에 적용하여 벡터 형태로 변환
3. 벡터로 변환된 데이터 포인트를 모델을 위해 미니배치로 모음

### 1. 각 토큰을 정수에 맵핑 (Vocabulary)

```
class Vocabulary(object):
    """ 매핑을 위해 텍스트를 처리하고 어휘 사전을 만드는 클래스 """

    def __init__(self, token_to_idx=None, add_unk=True, unk_token=<UNK>):
        """
        매개변수:
            token_to_idx (dict): 기존 토큰-인덱스 매핑 딕셔너리
            add_unk (bool): UNK 토큰을 추가할지 지정하는 플래그
            unk_token (str): Vocabulary에 추가할 UNK 토큰
        """

        if token_to_idx is None:
            token_to_idx = {}
        self._token_to_idx = token_to_idx

        self._idx_to_token = {idx: token
                             for token, idx in self._token_to_idx.items()}

        self._add_unk = add_unk
        self._unk_token = unk_token

        self.unk_index = -1
        if add_unk:
            self.unk_index = self.add_token(unk_token)
```

- 해당 클래스는 딕셔너리에 사용자가 토큰을 추가하면 자동으로 인덱스를 증가시키고 ‘UNK’라는 토큰도 관리.
- 이후 나오는 Vectorizer에서는 Vocabulary에서 자주 등장하지 않은 토큰을 제한

### 1. 각 토큰을 정수에 맵핑 (Vocabulary)

```
def lookup_index(self, index):
    """ 인덱스에 해당하는 토큰을 반환합니다.

    매개변수:
        index (int): 찾을 인덱스
    반환값:
        token (str): 인덱스에 해당하는 토큰
    예외:
        KeyError: 인덱스가 Vocabulary에 없을 때 발생합니다.
    """
    if index not in self._idx_to_token:
        raise KeyError("Vocabulary에 인덱스(%d)가 없습니다." % index)
    return self._idx_to_token[index]
```

```
def add_many(self, tokens):
    """ 토큰 리스트를 Vocabulary에 추가합니다.-

    매개변수:
        tokens (list): 문자열 토큰 리스트
    반환값:
        indices (list): 토큰 리스트에 상응되는 인덱스 리스트
    """
    return [self.add_token(token) for token in tokens]
```

```
def lookup_token(self, token):
    """ 토큰에 대응하는 인덱스를 추출합니다.
    토큰이 없으면 UNK 인덱스를 반환합니다.

    매개변수:
        token (str): 찾을 토큰
    반환값:
        index (int): 토큰에 해당하는 인덱스
    노트:
        UNK 토큰을 사용하려면 (Vocabulary에 추가하기 위해)
        `unk_index`가 0보다 커야 합니다.
    """
    if self.unk_index >= 0:
        return self._token_to_idx.get(token, self.unk_index)
    else:
        return self._token_to_idx[token]
```

- 새로운 토큰 추가를 위해 `add_token()`
- 토큰에 해당하는 인덱스를 추출할 때 `lookup_token()`
- 특정 인덱스에 해당하는 토큰을 추출할 때는 `lookup_index()`

### 2. 매핑된 정수를 각 데이터 포인트에 적용하여 벡터 형태로 변환 (Vectorizer)

```
class ReviewVectorizer(object):
    """ 어휘 사전을 생성하고 관리합니다 """
    def __init__(self, review_vocab, rating_vocab):
        """
        매개변수:
            review_vocab (Vocabulary): 단어를 정수에 매핑하는 Vocabulary
            rating_vocab (Vocabulary): 클래스 레이블을 정수에 매핑하는 Vocabulary
        """
        self.review_vocab = review_vocab
        self.rating_vocab = rating_vocab

    def vectorize(self, review):
        """ 리뷰에 대한 원-핫 벡터를 만듭니다

        매개변수:
            review (str): 리뷰
        반환값:
            one_hot (np.ndarray): 원-핫 벡터
        """
        one_hot = np.zeros(len(self.review_vocab), dtype=np.float32)

        for token in review.split(" "):
            if token not in string.punctuation:
                one_hot[self.review_vocab.lookup_token(token)] = 1

        return one_hot
```

- 다른 데이터 포인트에서 만든 벡터와 합쳐지므로 항상 길이가 같아야 함.

### 2. 매핑된 정수를 각 데이터 포인트에 적용하여 벡터 형태로 변환 (Vectorizer)

```
@classmethod
def from_dataframe(cls, review_df, cutoff=25):
    """ 데이터셋 데이터프레임에서 Vectorizer 객체를 만듭니다

    매개변수:
        review_df (pandas.DataFrame): 리뷰 데이터셋
        cutoff (int): 빈도 기반 필터링 설정값
    반환값:
        ReviewVectorizer 객체
    """
    review_vocab = Vocabulary(add_unk=True)
    rating_vocab = Vocabulary(add_unk=False)

    # 점수를 추가합니다
    for rating in sorted(set(review_df.rating)):
        rating_vocab.add_token(rating)

    # count > cutoff인 단어를 추가합니다
    word_counts = Counter()
    for review in review_df.review:
        for word in review.split(" "):
            if word not in string.punctuation:
                word_counts[word] += 1

    for word, count in word_counts.items():
        if count > cutoff:
            review_vocab.add_token(word)

    return cls(review_vocab, rating_vocab)
```

- **from\_dataframe()**

- 데이터셋에 있는 모든 토큰의 빈도수를 카운트
- 키워드 매개변수 cutoff에 지정한 수보다 빈도가 높은 토큰만 사용하는

### Vocabulary 객체 생성

### 2. 매핑된 정수를 각 데이터 포인트에 적용하여 벡터 형태로 변환 (Vectorizer)

- `vectorize()`
  - 매개 변수로 리뷰 문자열을 받고 이 리뷰의 벡터 표현을 반환
  - 원-핫 벡터 사용
    - Vocabulary의 크기와 길이가 같고 0과 1로 이루어진 벡터
    - 해당 벡터에서 리뷰의 단어에 해당하는 위치가 1
      - 희소한 배열이다 : 한 리뷰의 고유 단어 수는 항상 Vocabulary의 전체 단어 수보다 훨씬 작음
      - 리뷰에 등장하는 단어 순서를 무시(BoW 방식 : 각 단어가 등장한 횟수를 수치화하는 텍스트 표현 방법)

### 3. 벡터로 변환한 데이터 포인트 모으기 (DataLoader)

```
def generate_batches(dataset, batch_size, shuffle=True,
                     drop_last=True, device="cuda"):
    """
    파이토치 DataLoader를 감싸고 있는 제너레이터 함수.
    각 텐서를 지정된 장치로 이동합니다.
    """
    dataloader = DataLoader(dataset=dataset, batch_size=batch_size,
                           shuffle=shuffle, drop_last=drop_last)

    for data_dict in dataloader:
        out_data_dict = {}
        for name, tensor in data_dict.items():
            out_data_dict[name] = data_dict[name].to(device)
        yield out_data_dict
```

- 신경망 훈련에 필수인 미니배치로 모으는 작업을 편하게 해줌
- cpu와 gpu 간에 데이터를 간편하게 전환하는 파이썬 제너레이터
- `yield`를 사용하면 제너레이터를 반환한다.
  - 제너레이터 : 여러 개의 데이터를 미리 만들어 놓지 않고 필요할 때마다 즉석해서 하나씩 만들어낼 수 있는 객체

# 04 퍼셉트론 분류기

```

class ReviewClassifier(nn.Module):
    """ 간단한 퍼셉트론 기반 분류기 """
    def __init__(self, num_features):
        """

        매개변수:
            num_features (int): 입력 특성 벡터의 크기
        """

        super(ReviewClassifier, self).__init__()
        self.fc1 = nn.Linear(in_features=num_features,
                            out_features=1)

    def forward(self, x_in, apply_sigmoid=False):
        """ 분류기의 정방향 계산

        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
                x_in.shape는 (batch, num_features)입니다.
            apply_sigmoid (bool): 시그모이드 활성화 함수를 위한 플래그
                크로스-엔트로피 손실을 사용하려면 False로 지정합니다
        반환값:
            결과 텐서. tensor.shape은 (batch,)입니다.
        """

        y_out = self.fc1(x_in).squeeze()
        if apply_sigmoid:
            y_out = torch.sigmoid(y_out)
        return y_out

```

- 음성 혹은 양성인지를 분류하는 이진 분류 문제이기에 출력은 하나이다.
- 비선형 활성화 함수로는 시그모이드함수 사용
  - softmax는 다중 클래스 분류 문제를 위해 설계되었으며 선형 계층 출력을

다중 클래스에 걸친 확률 분포에 매핑함

- 이진 분류 문제이기에 BCELoss 사용
- 시그모이드와 손실 함수를 같이 사용하게 되면 수치 안정성 이슈가 있음
  - 파이토치에서는 시그모이드 함수와 BCELoss 함수를 연결하여 수치적으로 안정된 계산을 할 수 있도록 도와주는 BCEWithLogitsLoss()라는 제공

## 05 모델 훈련

```
args = Namespace(  
    # 날짜와 경로 정보  
    frequency_cutoff=25,  
    model_state_file='model.pth',  
    review_csv='data/yelp/reviews_with_splits_lite.csv',  
    # review_csv='data/yelp/reviews_with_splits_full.csv',  
    save_dir='model_storage/ch3/yelp/',  
    vectorizer_file='vectorizer.json',  
    # 모델 하이퍼파라미터 없음  
    # 훈련 하이퍼파라미터  
    batch_size=128,  
    early_stopping_criteria=5,  
    learning_rate=0.001,  
    num_epochs=100,  
    seed=1337,  
    # 실행 옵션  
    catch_keyboard_interrupt=True,  
    cuda=True,  
    expand_filepaths_to_save_dir=True,  
    reload_from_files=False,  
)
```

- 훈련 과정의 핵심
  - 모델 생성, 데이터셋 순회, 입력 데이터에서 모델의 출력 계산, 손실 계산, 손실에 비례하여 모델 수정
- 모든 결정 요소를 관리하는 args 객체 사용

## 06 레스토랑 리뷰 감성 분석하기

```
def make_train_state(args):
    return {'stop_early': False,
            'early_stopping_step': 0,
            'early_stopping_best_val': 1e8,
            'learning_rate': args.learning_rate,
            'epoch_index': 0,
            'train_loss': [],
            'train_acc': [],
            'val_loss': [],
            'val_acc': [],
            'test_loss': -1,
            'test_acc': -1,
            'model_filename': args.model_state_file}

# CUDA 체크
if not torch.cuda.is_available():
    args.cuda = False

print("CUDA 사용여부: {}".format(args.cuda))

args.device = torch.device("cuda" if args.cuda else "cpu")

dataset = ReviewDataset.load_dataset_and_make_vectorizer(args.review_csv)
vectorizer = dataset.get_vectorizer()

classifier = ReviewClassifier(num_features=len(vectorizer.review_vocab))
classifier = classifier.to(args.device)

loss_func = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(classifier.parameters(), lr=args.learning_rate)
```

- 훈련 상태가 복잡한 정보를 다룰 수 있도록 args 객체를 매개변수로 받음
- 데이터셋과 Vectorizer 객체 생성
- 퍼셉트론 기반 분류기 객체 생성
- BCEWithLogitsLoss 와 Adam 사용

## 06 레스토랑 리뷰 감성 분석하기

```
try:  
    for epoch_index in range(args.num_epochs):  
        train_state['epoch_index'] = epoch_index  
  
        # 훈련 세트에 대한 순회  
  
        # 훈련 세트와 배치 제너레이터 준비, 손실과 정확도를 0으로 설정  
        dataset.set_split('train')  
        batch_generator = generate_batches(dataset,  
                                           batch_size=args.batch_size,  
                                           device=args.device)  
        running_loss = 0.0  
        running_acc = 0.0  
        classifier.train()  
  
        for batch_index, batch_dict in enumerate(batch_generator):  
            # 훈련 과정은 5단계로 이루어집니다  
  
            # -----  
            # 단계 1. 그레이디언트를 0으로 초기화합니다  
            optimizer.zero_grad()  
  
            # 단계 2. 출력을 계산합니다  
            y_pred = classifier(x_in=batch_dict['x_data'].float())  
  
            # 단계 3. 손실을 계산합니다  
            loss = loss_func(y_pred, batch_dict['y_target'].float())  
            loss_t = loss.item()  
            running_loss += (loss_t - running_loss) / (batch_index + 1)  
  
            # 단계 4. 손실을 사용해 그레이디언트를 계산합니다  
            loss.backward()  
  
            # 단계 5. 옵티마이저로 가중치를 업데이트합니다  
            optimizer.step()  
            # -----  
  
            # 정확도를 계산합니다  
            acc_t = compute_accuracy(y_pred, batch_dict['y_target'])  
            running_acc += (acc_t - running_acc) / (batch_index + 1)  
  
            # 진행 바 업데이트  
            train_bar.set_postfix(loss=running_loss,  
                                  acc=running_acc,  
                                  epoch=epoch_index)  
            train_bar.update()  
  
            train_state['train_loss'].append(running_loss)  
            train_state['train_acc'].append(running_acc)
```

- 앞에서 초기화한 객체를 사용하여 모델의 성능이 높아지도록 모델 파라미터를 업데이트
- 두 개의 반복문으로 구성
  - 내부 반복문 : 데이터셋의 미니배치에 대해서 반복을 수행
  - → 미니배치마다 손실을 계산하고 옵티마이저가 모델 파라미터를 업데이트
- 외부 반복문 : 내부 반복문을 여러 번 반복

## 06 레스토랑 리뷰 감성 분석하기

```
try:  
    for epoch_index in range(args.num_epochs):  
        train_state['epoch_index'] = epoch_index  
  
        # 훈련 세트에 대한 순회  
  
        # 훈련 세트와 배치 제너레이터 준비, 손실과 정확도를 0으로 설정  
        dataset.set_split('train')  
        batch_generator = generate_batches(dataset,  
                                           batch_size=args.batch_size,  
                                           device=args.device)  
        running_loss = 0.0  
        running_acc = 0.0  
        classifier.train()  
  
        for batch_index, batch_dict in enumerate(batch_generator):  
            # 훈련 과정은 5단계로 이루어집니다  
  
            # -----  
            # 단계 1. 그레이디언트를 0으로 초기화합니다  
            optimizer.zero_grad()  
  
            # 단계 2. 출력을 계산합니다  
            y_pred = classifier(x_in=batch_dict['x_data'].float())  
  
            # 단계 3. 손실을 계산합니다  
            loss = loss_func(y_pred, batch_dict['y_target'].float())  
            loss_t = loss.item()  
            running_loss += (loss_t - running_loss) / (batch_index + 1)  
  
            # 단계 4. 손실을 사용해 그레이디언트를 계산합니다  
            loss.backward()  
  
            # 단계 5. 옵티마이저로 가중치를 업데이트합니다  
            optimizer.step()  
            # -----  
  
            # 정확도를 계산합니다  
            acc_t = compute_accuracy(y_pred, batch_dict['y_target'])  
            running_acc += (acc_t - running_acc) / (batch_index + 1)  
  
            # 진행 바 업데이트  
            train_bar.set_postfix(loss=running_loss,  
                                  acc=running_acc,  
                                  epoch=epoch_index)  
            train_bar.update()  
  
            train_state['train_loss'].append(running_loss)  
            train_state['train_acc'].append(running_acc)
```

- 에폭 횟수만큼 for문 반복문 실행
- 어떤 데이터셋을 사용할지 지정 → 데이터로더 클래스인 `generate_batches()`를 호출하기 전 필요
- 배치 간의 손실과 정확도 기록
- `train()` 메서드 호출하여 훈련 모드에 돌입, 모델 파라미터 수정 가능 지정
- `batch_generator`에서 추출한 훈련 배치를 순회하며 모델 파라미터를 업데이트하는 핵심 연산 수행
  1. `optimizer.zero_grad()` → 옵티마이저의 그레이디언트 초기화
  2. 출력 계산
  3. 출력과 타깃 사이의 손실 계산
  4. 손실을 사용하여 `backward()` 메서드를 호출하여 파라미터에 그레이디언트 전파
  5. `optimizer.step()` → 파라미터 업데이트
  6. → SGD의 핵심
- 최종 손실과 정확도 값으로 업데이트

## 06 레스토랑 리뷰 감성 분석하기

- eval() 메서드는 모델 파라미터를 수정하지 못하게 하고 드롭아웃을 비활성화, 손실을 계산하지 않고 그레이디언트를 파라미터로 전파 X
  - 단순 측정

# 06 평가, 추론, 분석

## <test 데이터로 평가>

# 06 평가, 추론, 분석

```

def predict_rating(review, classifier, vectorizer, decision_threshold=0.5):
    """ 리뷰 점수 예측하기

    매개변수:
        review (str): 리뷰 텍스트
        classifier (ReviewClassifier): 훈련된 모델
        vectorizer (ReviewVectorizer): Vectorizer 객체
        decision_threshold (float): 클래스를 나눌 결정 경계
    """
    review = preprocess_text(review)

    vectorized_review = torch.tensor(vectorizer.vectorize(review))
    result = classifier(vectorized_review.view(1, -1))

    probability_value = torch.sigmoid(result).item()
    index = 1
    if probability_value < decision_threshold:
        index = 0

    return vectorizer.rating_vocab.lookup_index(index)

test_review = "this is a pretty awesome book"

classifier = classifier.cpu()
prediction = predict_rating(test_review, classifier, vectorizer, decision_threshold=0.5)
print("{} -> {}".format(test_review, prediction))
✓ 0.5s

```

this is a pretty awesome book -> positive

<새로운 데이터 포인트 추론하여 분류하여 평가>

# 06 평가, 추론, 분석

```

# 가중치 정렬
fc1_weights = classifier.fc1.weight.detach()[0]
_, indices = torch.sort(fc1_weights, dim=0, descending=True)
indices = indices.numpy().tolist()

# 긍정적인 상위 20개 단어
print("긍정 리뷰에 영향을 미치는 단어:")
print("-----")
for i in range(20):
    print(vectorizer.review_vocab.lookup_index(indices[i]))

print("====\n\n\n")

# 부정적인 상위 20개 단어
print("부정 리뷰에 영향을 미치는 단어:")
print("-----")
indices.reverse()
for i in range(20):
    print(vectorizer.review_vocab.lookup_index(indices[i]))

```

긍정 리뷰에 영향을 미치는 단어:

-----  
delicious  
fantastic  
pleasantly  
amazing  
great  
vegas  
excellent  
yum  
perfect  
awesome  
ngreat  
yummy  
love  
bomb  
solid  
pleased  
wonderful  
chinatown  
notch  
deliciousness  
=====

부정 리뷰에 영향을 미치는 단어:

-----  
worst  
mediocre  
bland  
horrible  
meh  
awful  
rude  
terrible  
tasteless  
overpriced  
disgusting  
unacceptable  
poorly  
slowest  
unfriendly  
nmaybe  
disappointing  
disappointment  
downhill  
underwhelmed

&lt;모델 가중치 분석&gt;

**감사합니다.**