Sentiment Analysis on Classification

MLP LAB 임경태, 임현석

Contents

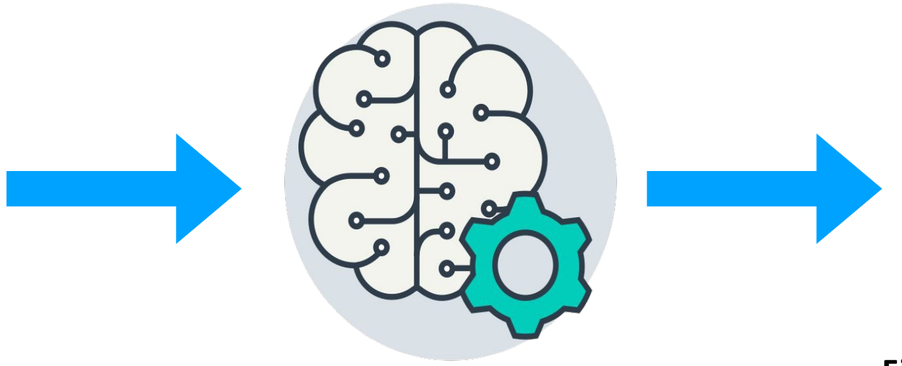
01. Logistic Regression

Recap

- Multi-variable Linear Regression

Quiz1: 73
Quiz2: 80
Quiz3: 75

Quiz Scores



152

Final Score Prediction

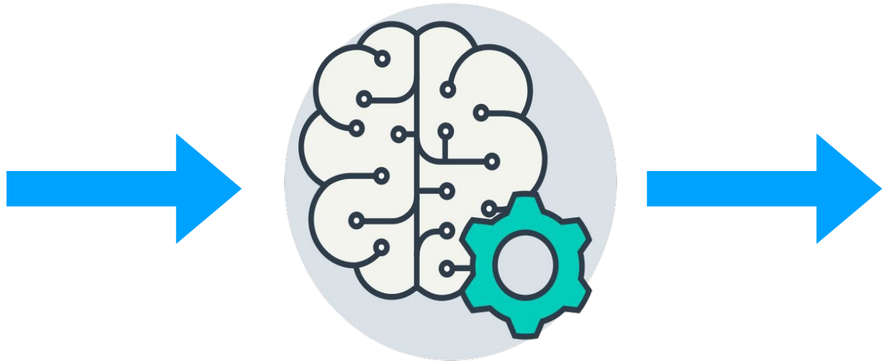
$$H(x) = Wx + b$$

Questions

- How do I solve these cases?

Quiz1: 73
Quiz2: 80
Quiz3: 75

Quiz Scores



Fail
Pass

Class Prediction

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

Solution

- Logistic Regression (LR) Model
  - Let's switch from regression to a classification problem!

    Simply output the model's output as 1 for passing and 0 for failing!

$$H(x) = Wx + b \quad \rightarrow \quad \boldsymbol{H(x) = s(Wx + b)}$$

Logistic Regression (LR)

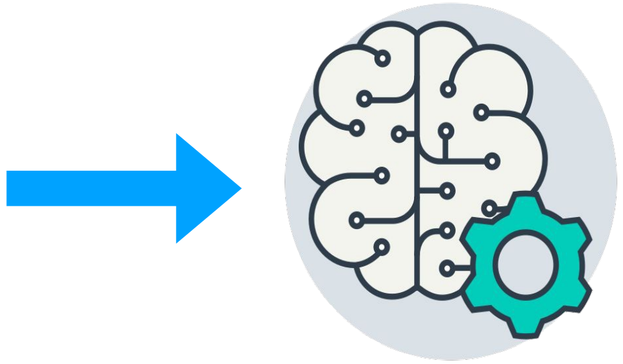
- Sigmoid function

  - A Function to convert from regression to classification problems

  Change the output of a model to a value between 0 and 1

Quiz1: 73
Quiz2: 80
Quiz3: 75

Quiz Scores

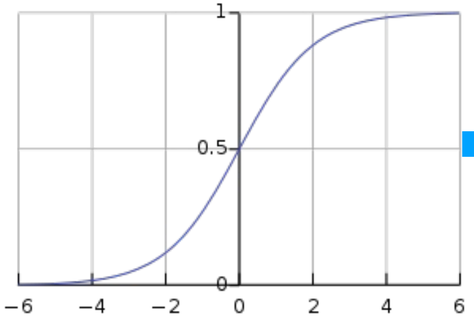


s(152)

$$s(x) = \frac{1}{1 + e^{-x}}$$

Fail == 0
Pass == 1

Class Prediction



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

Logistic Regression (LR)

- Loss of LR?: Negative Log Likely-hood cost function

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

$$\mathbb{P}(y=1|z) = \sigma(z) = \frac{1}{1+e^{-z}}$$
$$\mathbb{P}(y=0|z) = 1-\sigma(z) = \frac{1}{1+e^{z}}$$

$$\mathbb{P}(y|z) = \sigma(z)^y(1-\sigma(z))^{1-y}$$

Logistic Regression (LR)

- Gradient Descent Function

Weight Update via Gradient Descent

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

- α: Learning rate

02. Logistic Regression codes

Logistic Regression (LR) with Pytorch

- Data preparation

```
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

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

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

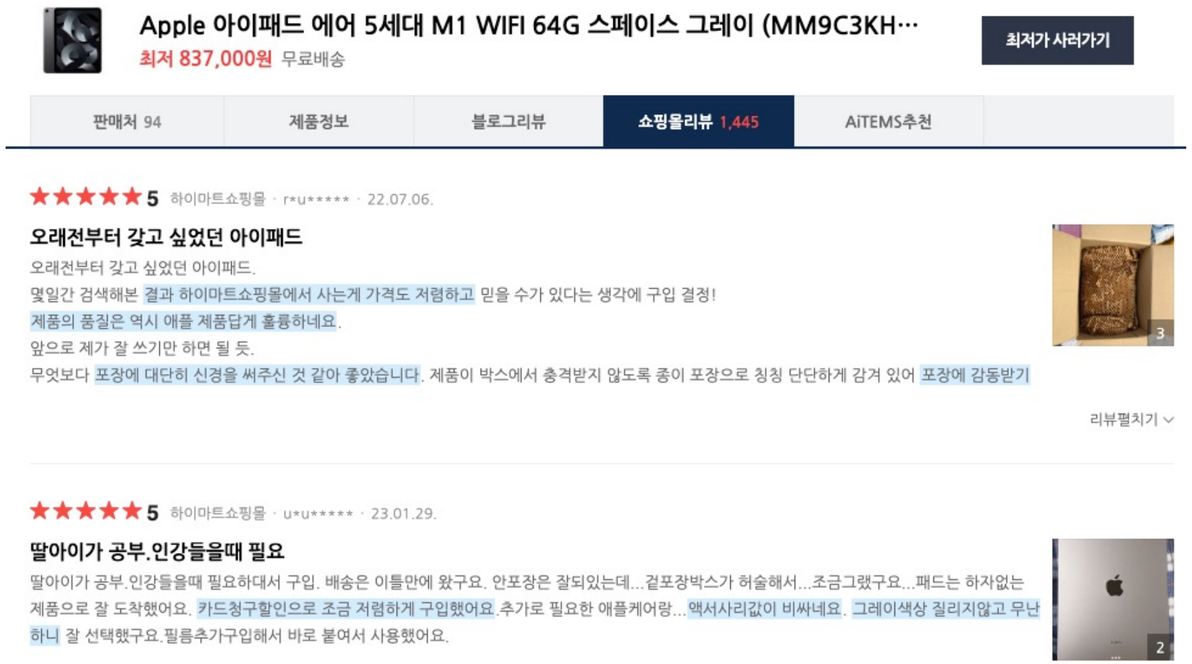
```
# optimizer 설정
optimizer = optim.SGD(model.parameters(), lr=1)

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 {:2.2f}%'.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

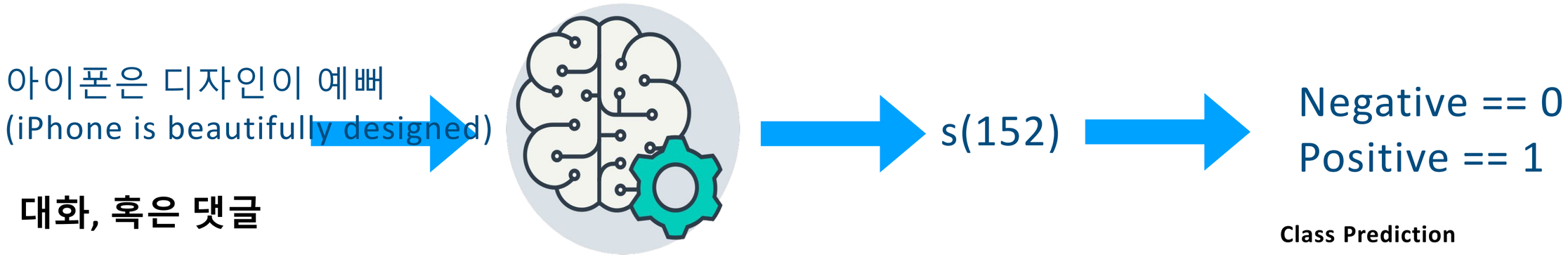
Sentiment Analysis

- Sentiment analysis: A technique for categorizing the polarity of sentences and documents, such as positive/negative/neutral.



A simple Sentiment Analysis on LR

- Simple Modeling of a Binary Classification Problem



A simple Sentiment Analysis on LR

- Simple Modeling of a Binary Classification Problem
  - An example of “Bag of Words Meets Bags of Popcorn” at Kaggle
  - Can we refactor AWE Linear Regression code?

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. Multinominal Classification

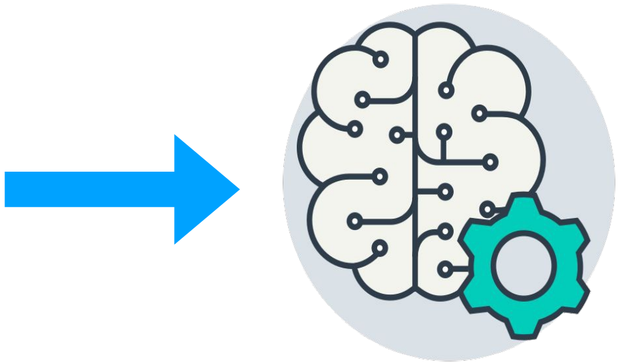
Questions

- How do I solve multi-class prediction?

  - This is known as the Multinominal Classification problem.

Quiz1: 73
Quiz2: 80
Quiz3: 75

Quiz Scores



Failed
Accepted
Preliminary

Class Prediction

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

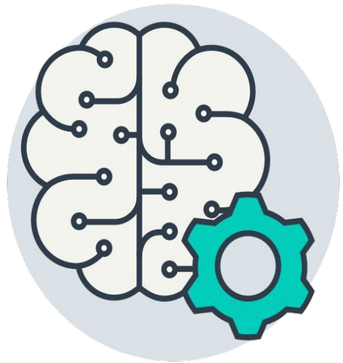
Solution

- Probability Distribution Output with Softmax
  - Let's switch from regression to a classification problem!

    Simply make the output of the model a probability distribution over the number of classes.

Quiz1: 73
Quiz2: 80
Quiz3: 75

Quiz Scores



s(11,25,37.1)

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

Failed == 0.1
Accepted == 0.2
Preliminary == 0.7

Class Prediction

$$\boldsymbol{H(x) = s(Wx + b)}$$

Multinominal Classification

- Loss of Multinominal Classification: Cross Entropy

  - A method to calculate the distance between two probability distributions

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

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

```
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
```

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

```
[89] # 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)
```

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

```
[93] # 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)
```

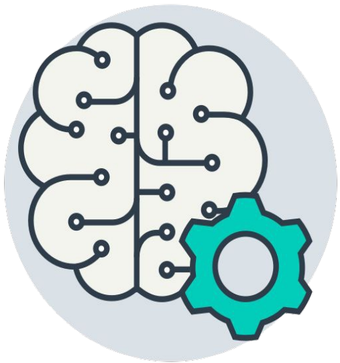
```
tensor([[   nan,    nan,    nan,    nan, 0.0000, 0.0101]])
```

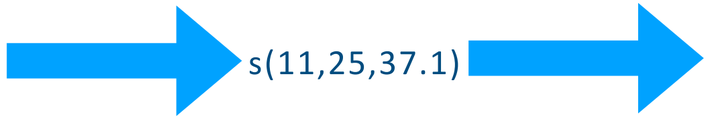
Multinominal Classification

- Loss of Multinominal Classification: Cross Entropy

  - A method to calculate the distance between two probability distributions

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





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

Failed == 0.1
Accepted == 0.2
Preliminary == 0.7
Class Prediction

CE loss

Failed == 0
Accepted==1
Preliminary==0
Target (실제정답)

Multinominal Classification

- Implementing Cross Entropy

Preparing the Data

- Let's create random dummy data and save it as z, assuming it results from a regression.

- To change the regression result to Multinominal Classification, we need Softmax!

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=<SoftmaxBackwa
rd>)
```

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

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

Multinominal Classification

- Implementing Cross Entropy

  - Change the actual answer data to a probability distribution, one-hot encoding

- Calculate Loss: Calculate the CE Loss for the two probability distributions of the true answer and the predicted value

```
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>)
```

Multinominal Classification

- Cross Entropy (low, high-level)

```
# 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>)
```

Multinominal Classification

- Gradient Descent Fuction

Weight Update via Gradient Descent

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

- α: Learning rate

Thank you