Logistic Regression

Presenter: Kyuhun Sim, Hayun Lee

Contents

1. Logistic Regression

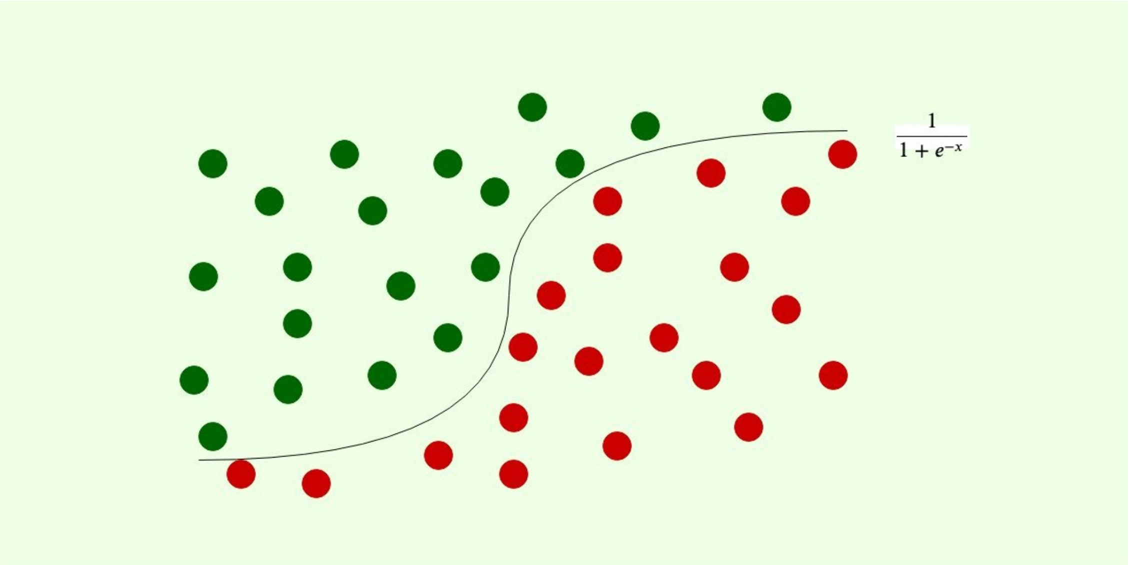
Logistic Regression

Linear Regression

- Dependent variable is normally continuous normal distribution

Logistic Regression

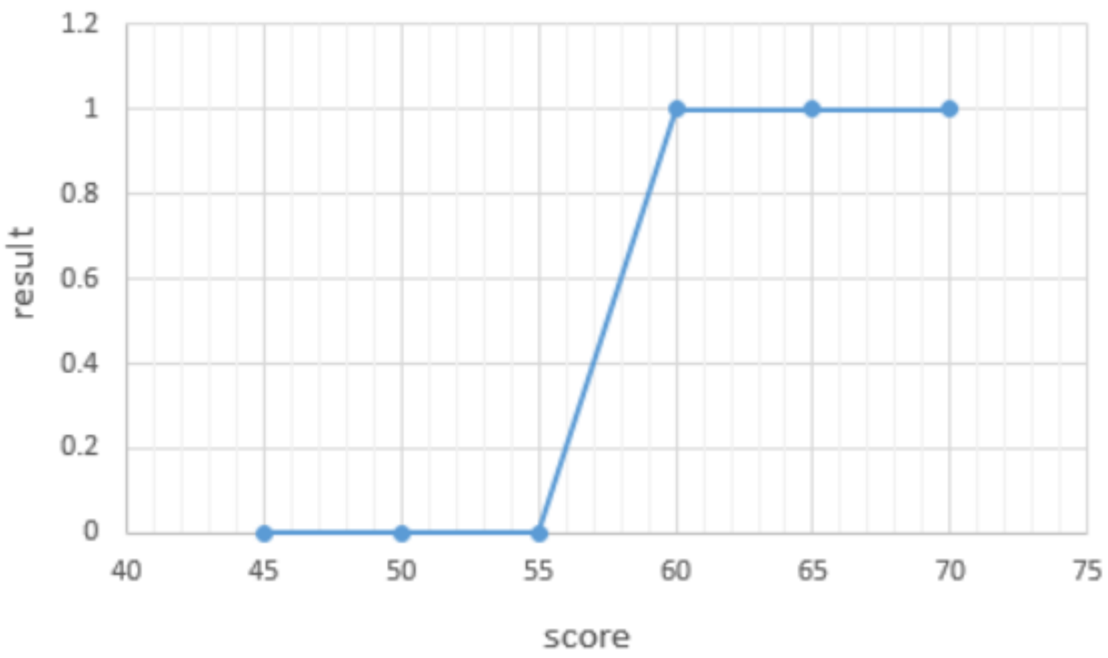
- Dependent variables are categorial, binary classification



Binary Classification

- Example) The pass or fail is divided according to the test score.

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격



https://wikidocs.net/57805

Binary Classification

- In the classification, Linear function like Wx+b is not inappropriate

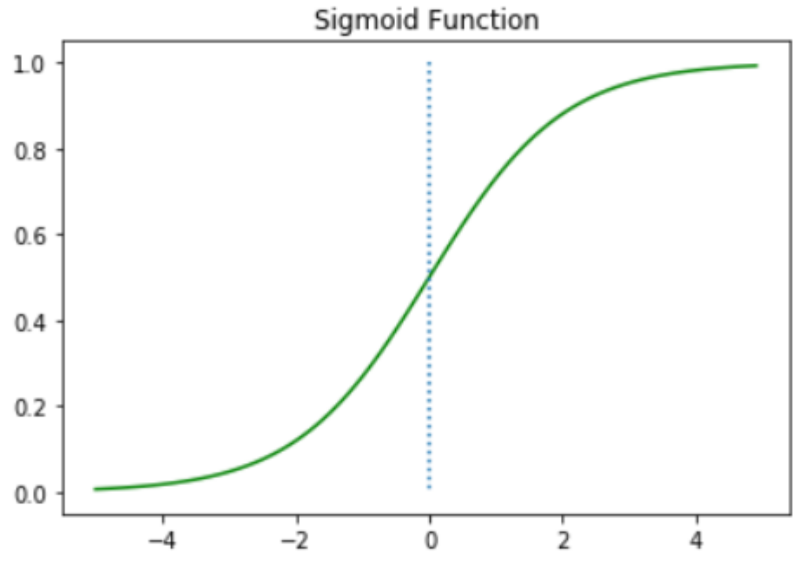
- Sigmoid function that can be expressed in S-shape

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

Why called Logistic Regression?

- sigmoid function, a logistic function in which the output value is divided between 0 and 1

- By using sigmoid, dependent variable can be expressed in categories of 0 and 1

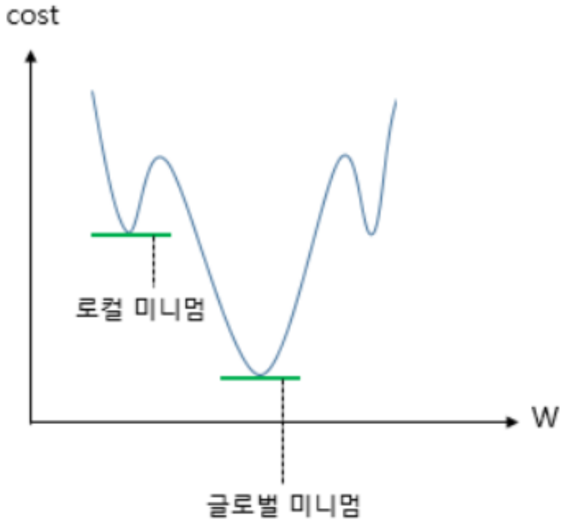


Cost Function

Linear Regression

$$cost(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)} - H(x^{(i)})\right]^2$$

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

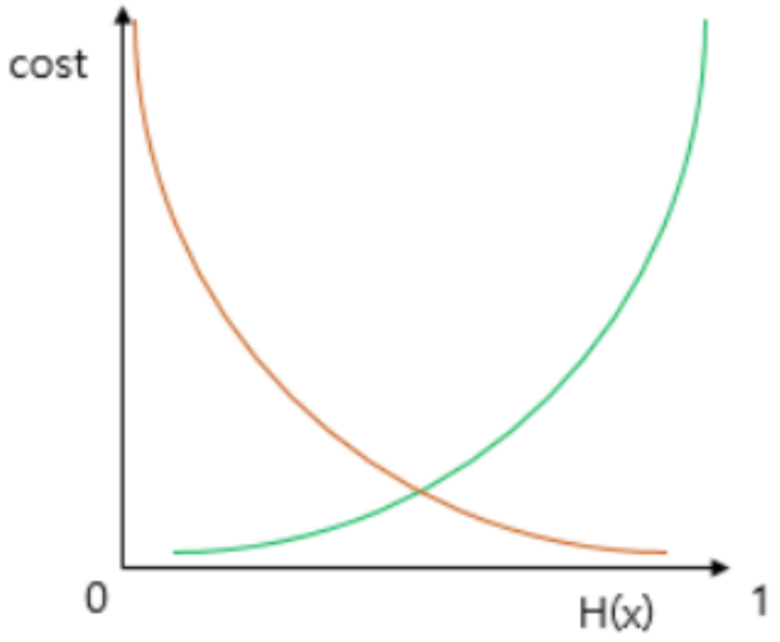


Problem

- If choose local minimum, the model performance will not increase

Cost Function

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



If the real value = 1 green graph

If the real value = 0 orange graph

1) If the real value = 1, H(x)=1, cost =0
2) If the real value = 0, H(x)=0, cost diverges

$$\text{if } y = 1 \rightarrow \text{cost}\ (H(x), y) = -\log(H(x))$$

$$\text{if } y = 0 \rightarrow \text{cost}\ (H(x), y) = -\log(1 - H(x))$$

$$\text{cost}\ (H(x), y) = -[ylogH(x) + (1-y)log(1 - H(x))]$$

Code

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

```
e^1 equals:  tensor([2.7183])
```

Setting weight ,bias, and hypothesis

```
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)))
```

```
hypothesis = torch.sigmoid(x_train.matmul(W) + b)
```

Code

- Computing the cost function

```
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>)
```

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

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

```
-(y_train[0] * torch.log(hypothesis[0]) +
  (1 - y_train[0]) * torch.log(1 - hypothesis[0]))
```

Code

- Computing the cost function with F.binary_cross_entropy

```
F.binary_cross_entropy(hypothesis, y_train)
```

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

Code

```python
# 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()
        ))
```

```python
x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]
x_train = torch.FloatTensor(x_data)
y_train = torch.FloatTensor(y_data)
```

```python
# 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 = -(y_train * torch.log(hypothesis) +
             (1 - y_train) * torch.log(1 - hypothesis)).mean()

    # 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()
        ))
```

2. Softmax Classification

Multi-Class classification

- Choose one from three or more options

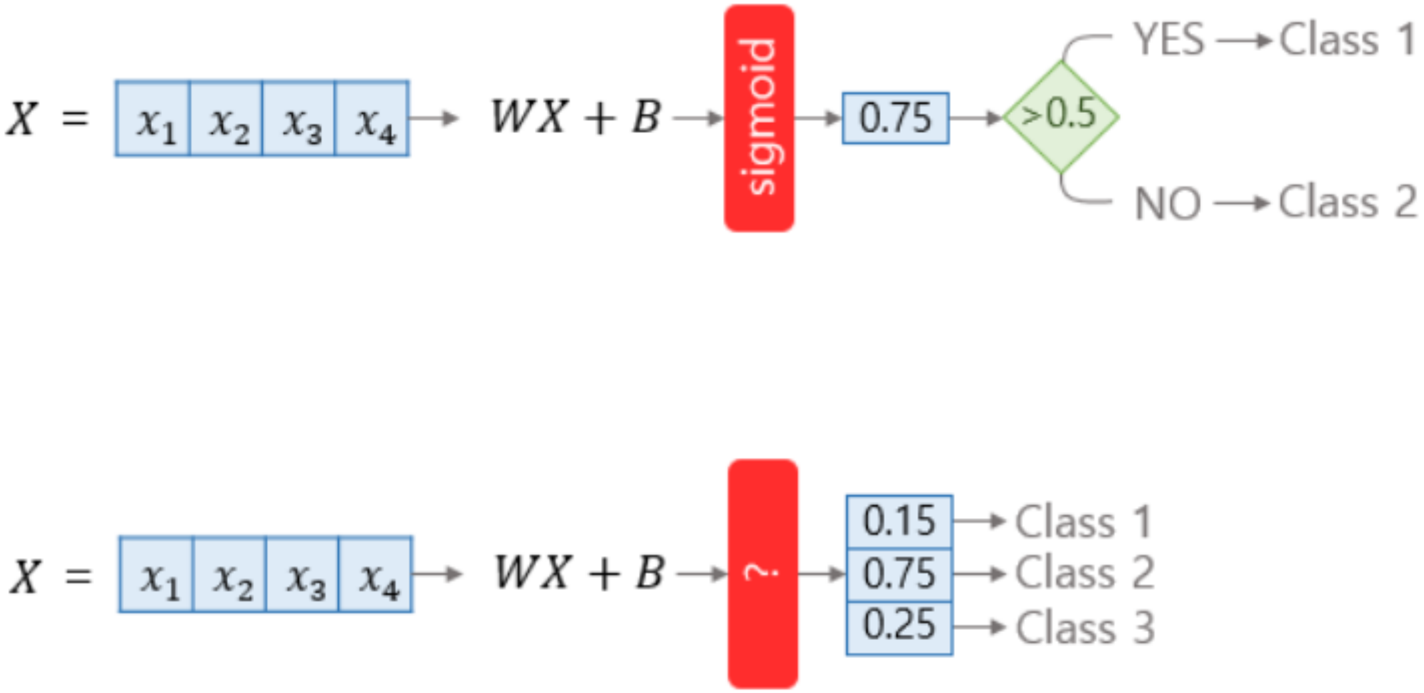
- Example: iris classification

SepalLengthCm(x_1)	SepalWidthCm(x_2)	PetalLengthCm(x_3)	PetalWidthCm(x_4)	Species(y)
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
5.8	2.6	4.0	1.2	versicolor
6.7	3.0	5.2	2.3	virginica
5.6	2.8	4.9	2.0	virginica

Predict which of three iris varieties: setosa, versicolor, and virginica from four features

Multi-Class Classification

- Difference between sigmoid and softmax



Softmax

- Let the i-th element in the k-dimensional vector be zi,
- The probability that the i-th class is the correct answer is pi

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad for\ i = 1, 2, \ldots k$$

```
z = torch.FloatTensor([1, 2, 3])
```

```
hypothesis = F.softmax(z, dim=0)
print(hypothesis)
```

```
tensor([0.0900, 0.2447, 0.6652])
```

Cross entropy loss

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

```
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=<SoftmaxBackward>)
```

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

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

```
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>)
```

https://wikidocs.net/35476

Cross-entropy Loss with torch.nn.functional

```
torch.log(F.softmax(z, dim=1))
```

```
tensor([[-1.3301, -1.8084, -1.6846, -1.3530, -2.0584],
        [-1.4147, -1.8174, -1.4602, -1.6450, -1.7758],
        [-1.5025, -1.6165, -1.4586, -1.8360, -1.6776]], grad_fn=<LogBackward>)
```

```
F.log_softmax(z, dim=1)
```

```
tensor([[-1.3301, -1.8084, -1.6846, -1.3530, -2.0584],
        [-1.4147, -1.8174, -1.4602, -1.6450, -1.7758],
        [-1.5025, -1.6165, -1.4586, -1.8360, -1.6776]],
       grad_fn=<LogSoftmaxBackward>)
```

```
# 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>)
```

https://wikidocs.net/35476

```python
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)
```

```python
# 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).mean()

    # 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()
        ))
```

```python
# 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 (2)
    z = x_train.matmul(W) + b # or .mm or @
    cost = F.cross_entropy(z, 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()
        ))
```

https://wikidocs.net/35476

3. Tips

Overfitting

- More Data
- Less features
- **Regularization**
  - Early Stooping
  - Reducing Network Size
  - Weight Decay
  - Dropout
  - Batch Normalization

Basic Approach to Train DNN

1. Make a neural network architecture.
2. Train and check that model is over-fitted.
    1. If it is not, increase the model size (deeper and wider).
    2. If it is, add regularization, such as drop-out, batch-normalization
3. Repeat from step-2

Imports

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
# For reproducibility
torch.manual_seed(1)
```

```
<torch._C.Generator at 0x7f0708f8ffb0>
```

Training and Test Dataset

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

```
x_test = torch.FloatTensor([[2, 1, 1], [3, 1, 2], [3, 3, 4]])
y_test = torch.LongTensor([2, 2, 2])
```

Model

```
class SoftmaxClassifierModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(3, 3)
    def forward(self, x):
        return self.linear(x)
```

```
model = SoftmaxClassifierModel()
```

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

Training

```python
def train(model, optimizer, x_train, y_train):
    nb_epochs = 20
    for epoch in range(nb_epochs):

        # 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()

        print('Epoch {:4d}/{} Cost: {:.6f}'.format(
            epoch, nb_epochs, cost.item()
        ))
```

Test (Validation)

```python
def test(model, optimizer, x_test, y_test):
    prediction = model(x_test)
    predicted_classes = prediction.max(1)[1]
    correct_count = (predicted_classes == y_test).sum().item()
    cost = F.cross_entropy(prediction, y_test)

    print('Accuracy: {}% Cost: {:.6f}'.format(
         correct_count / len(y_test) * 100, cost.item()
    ))
```

Run

```
train(model, optimizer, x_train, y_train)
```

```
Epoch    0/20 Cost: 2.203667
Epoch    1/20 Cost: 1.199645
Epoch    2/20 Cost: 1.142985
Epoch    3/20 Cost: 1.117769
Epoch    4/20 Cost: 1.100901
Epoch    5/20 Cost: 1.089523
Epoch    6/20 Cost: 1.079872
Epoch    7/20 Cost: 1.071320
Epoch    8/20 Cost: 1.063325
Epoch    9/20 Cost: 1.055720
Epoch   10/20 Cost: 1.048378
Epoch   11/20 Cost: 1.041245
Epoch   12/20 Cost: 1.034285
Epoch   13/20 Cost: 1.027478
Epoch   14/20 Cost: 1.020813
Epoch   15/20 Cost: 1.014279
Epoch   16/20 Cost: 1.007872
Epoch   17/20 Cost: 1.001586
Epoch   18/20 Cost: 0.995419
Epoch   19/20 Cost: 0.989365
```

```
test(model, optimizer, x_test, y_test)
```

```
Accuracy: 0.0% Cost: 1.425844
```

Learning Rate

- If the learning rate is too large, the cost increases gradually while diverging (overshooting).

```
model = SoftmaxClassifierModel()
```

```
optimizer = optim.SGD(model.parameters(), lr=1e5)
```

```
train(model, optimizer, x_train, y_train)
```

```
Epoch    0/20 Cost: 1.280268
Epoch    1/20 Cost: 976950.812500
Epoch    2/20 Cost: 1279135.125000
Epoch    3/20 Cost: 1198379.000000
Epoch    4/20 Cost: 1098825.875000
Epoch    5/20 Cost: 1968197.625000
Epoch    6/20 Cost: 284763.250000
Epoch    7/20 Cost: 1532260.125000
Epoch    8/20 Cost: 1651504.000000
Epoch    9/20 Cost: 521878.500000
Epoch   10/20 Cost: 1397263.250000
Epoch   11/20 Cost: 750986.250000
Epoch   12/20 Cost: 918691.500000
Epoch   13/20 Cost: 1487888.250000
Epoch   14/20 Cost: 1582260.125000
Epoch   15/20 Cost: 685818.062500
Epoch   16/20 Cost: 1140048.750000
Epoch   17/20 Cost: 940566.500000
Epoch   18/20 Cost: 931638.250000
Epoch   19/20 Cost: 1971322.625000
```

Learning Rate

- If the learning rate is too small, the cost hardly decreases.

```
model = SoftmaxClassifierModel()
```

```
optimizer = optim.SGD(model.parameters(), lr=1e-10)
```

```
train(model, optimizer, x_train, y_train)
```

```
Epoch    0/20 Cost: 3.187324
Epoch    1/20 Cost: 3.187324
Epoch    2/20 Cost: 3.187324
Epoch    3/20 Cost: 3.187324
Epoch    4/20 Cost: 3.187324
Epoch    5/20 Cost: 3.187324
Epoch    6/20 Cost: 3.187324
Epoch    7/20 Cost: 3.187324
Epoch    8/20 Cost: 3.187324
Epoch    9/20 Cost: 3.187324
Epoch   10/20 Cost: 3.187324
Epoch   11/20 Cost: 3.187324
Epoch   12/20 Cost: 3.187324
Epoch   13/20 Cost: 3.187324
Epoch   14/20 Cost: 3.187324
Epoch   15/20 Cost: 3.187324
Epoch   16/20 Cost: 3.187324
Epoch   17/20 Cost: 3.187324
Epoch   18/20 Cost: 3.187324
Epoch   19/20 Cost: 3.187324
```

Learning Rate

- Start with an appropriate learning rate.
  - If it diverges → adjust it small
  - If the cost does not decrease → adjust is largely

```
model = SoftmaxClassifierModel()
```

```
optimizer = optim.SGD(model.parameters(), lr=1e-1)
```

```
train(model, optimizer, x_train, y_train)
```

```
Epoch    0/20 Cost: 1.341573
Epoch    1/20 Cost: 1.198802
Epoch    2/20 Cost: 1.150877
Epoch    3/20 Cost: 1.131977
Epoch    4/20 Cost: 1.116242
Epoch    5/20 Cost: 1.102514
Epoch    6/20 Cost: 1.089676
Epoch    7/20 Cost: 1.077479
Epoch    8/20 Cost: 1.065775
Epoch    9/20 Cost: 1.054511
Epoch   10/20 Cost: 1.043655
Epoch   11/20 Cost: 1.033187
Epoch   12/20 Cost: 1.023091
Epoch   13/20 Cost: 1.013356
Epoch   14/20 Cost: 1.003968
Epoch   15/20 Cost: 0.994917
Epoch   16/20 Cost: 0.986189
Epoch   17/20 Cost: 0.977775
Epoch   18/20 Cost: 0.969660
Epoch   19/20 Cost: 0.961836
```

Data Preprocessing

```
x_train = torch.FloatTensor([[73, 80, 75],
                             [93, 88, 93],
                             [89, 91, 90],
                             [96, 98, 100],
                             [73, 66, 70]])
y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])
```

Data Preprocessing

$$x_j' = \frac{x_j - \mu_j}{\sigma_j}$$

여기서 σ 는 standard deviation, μ 는 평균값 이다.

```
mu = x_train.mean(dim=0)
```

```
sigma = x_train.std(dim=0)
```

```
norm_x_train = (x_train - mu) / sigma
```

```
print(norm_x_train)
```

```
tensor([[-1.0674, -0.3758, -0.8398],
        [ 0.7418,  0.2778,  0.5863],
        [ 0.3799,  0.5229,  0.3486],
        [ 1.0132,  1.0948,  1.1409],
        [-1.0674, -1.5197, -1.2360]])
```

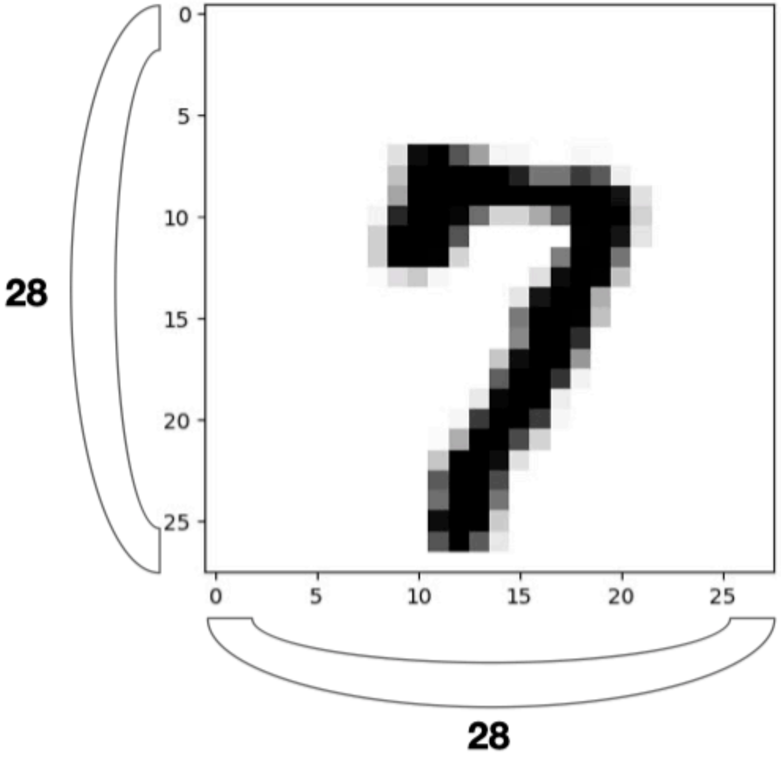
4. MNIST Introduction

What is MNIST?

- MNIST: handwritten digits dataset

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes; 60,000 samples)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
T10k-images-idx3-ubyte.gz :  test set images (1648877 bytes; 10,000 samples)
T10k-labels-idx1-ubyte.gz :  test set labels (4542 bytes)
```

Example of MNIST



- 28 x 28 image
- 1 channel gray image
- 0 ~ 9 digits

```
for X, Y in data_loader:
    # reshape input image into [batch_size by 784]
    # label is not one-hot encoded
    X = X.view(-1, 28 * 28)
```

torchvision

- The **torchvision** package consists of
  - Popular datasets
  - Model architectures
  - Common image transformations

- torchvision.datasets
  - MNIST
  - Fashion-MNIST
  - EMNIST
  - COCO
  - LSUN
  - ImageFolder
  - DatasetFolder
  - Imagenet-12
  - CIFAR
  - STL10
  - SVHN
  - PhotoTour
  - SBU
  - Flickr
  - VOC

- torchvision.models
  - Alexnet
  - VGG
  - ResNet
  - SqueezeNet
  - DenseNet
  - Inception v3
- torchvision.transforms
  - Transforms on PIL Image
  - Transforms on torch.*Tensor
  - Conversion Transforms
  - Generic Transforms
  - Functional Transforms
- torchvision.utils

Reading Data

```
import torchvision.datasets as dsets
...
mnist_train = dsets.MNIST(root="MNIST_data/", train=True, transform=transforms.ToTensor(),
download=True)

mnist_test = dsets.MNIST(root="MNIST_data/", train=False, transform=transforms.ToTensor(),
download=True)

data_loader = torch.utils.DataLoader(DataLoader=mnist_train, batch_size=batch_size,
shuffle=True, drop_last=True)
...
for epoch in range(training_epochs):
...
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
```

Epoch / Batch Size / Iteration

- In the neural network terminology:
  - One **epoch**
    - One forward pass and one backward pass of all the training examples
  - **Batch size**
    - The number of training examples in one forward/backward pass.
    - The higher the batch size, the more memory space you'll need.
  - Number of **iterations**
    - Number of passes, each pass using [batch size] num of examples.
    - To be clear, one pass = one forward pass + one backward pass.
      (we do not count the forward pass and backward pass as two different passes.)

- **Example**: If you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

Softmax

```
# MNIST data image of shape 28 * 28 = 784
linear = torch.nn.Linear(784, 10, bias=True).to(device)
# initialization
torch.nn.init.normal_(linear.weight)
# parameters
training_epochs = 15
batch_size = 100
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(data_loader)
    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
        optimier.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        avg_cost += cost / total_batch
    print("Epoch: ", "%04d" % (epoch+1), "cost =", "{:.9f}".format(avg_cost))
```

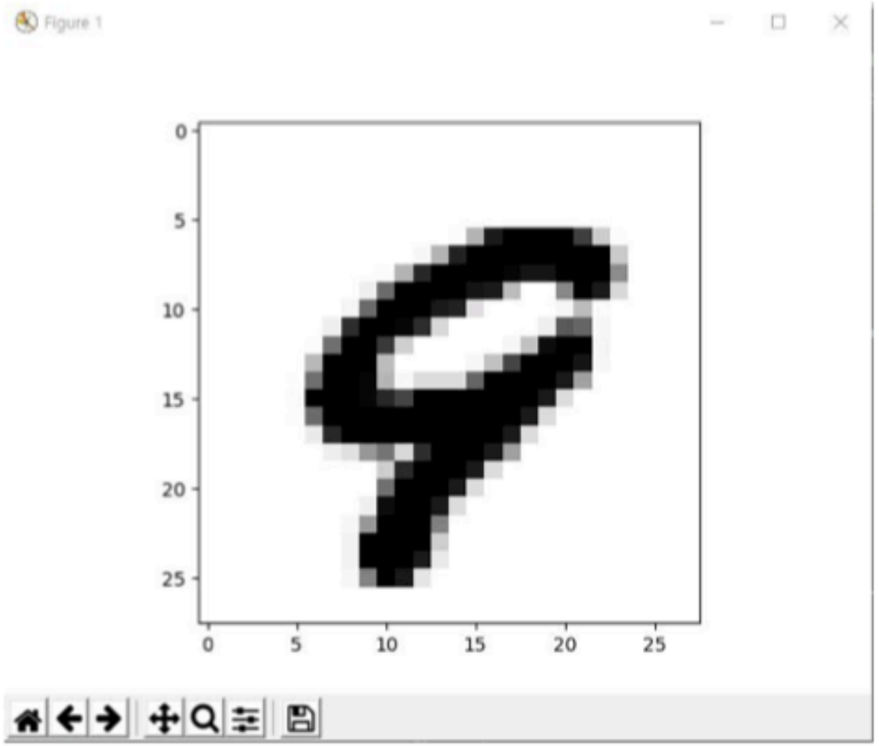
```
Epoch: 0001 cost = 2.511683702
Epoch: 0002 cost = 0.977319956
Epoch: 0003 cost = 0.797017217
Epoch: 0004 cost = 0.710427940
Epoch: 0005 cost = 0.655205429
Epoch: 0006 cost = 0.615207732
Epoch: 0007 cost = 0.584421575
Epoch: 0008 cost = 0.559486568
Epoch: 0009 cost = 0.538655698
Epoch: 0010 cost = 0.520880997
Epoch: 0011 cost = 0.505315244
Epoch: 0012 cost = 0.491431117
Epoch: 0013 cost = 0.479477882
Epoch: 0014 cost = 0.468681127
Epoch: 0015 cost = 0.458788306
Learning finished
Accuracy: 0.8718999624252319
```

Test

```python
# Test the model using test sets
With torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print("Accuracy: ", accuracy.item())
```

Visualization




```python
import matplotlib.pyplot as plt
import random
...
r = random.randint(0, len(mnist_test) - 1)
X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 *
28).float().to(device)
Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

print("Label: ", Y_single_data.item())
single_prediction = linear(X_single_data)
print("Prediction: ", torch.argmax(single_prediction,
1).item())

plt.imshow(mnist_test.test_data[r:r + 1].view(28, 28),
cmap="Greys", interpolation="nearest")
plt.show()
```

```
Label:  8
Prediction:  8
```