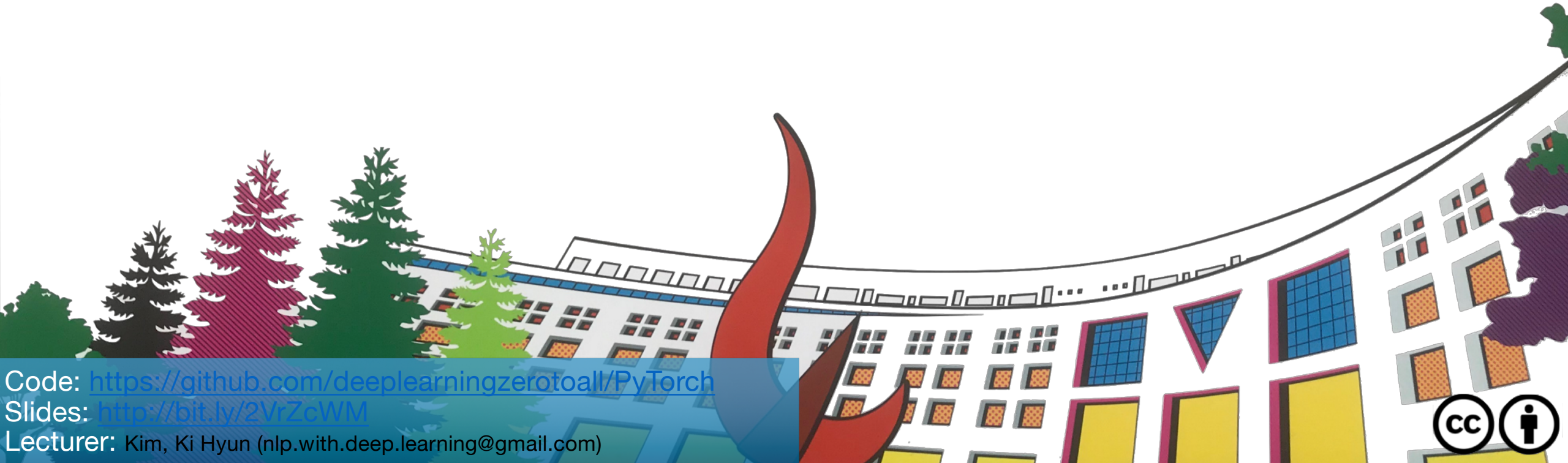
ML/DL for Everyone Season2

with PYTORCH

Tips



Code: https://github.com/deeplearningzerotoall/PyTorch
Slides: http://bit.ly/2VrZcWM
Lecturer: Kim, Ki Hyun (nlp.with.deep.learning@gmail.com)

Tips

- Reminder: Maximum Likelihood Estimation
- Reminder: Optimization via Gradient Descent
- Reminder: Overfitting and Regrularization
- Training and Test Dataset
- Learning Rate
- Data Preprocessing

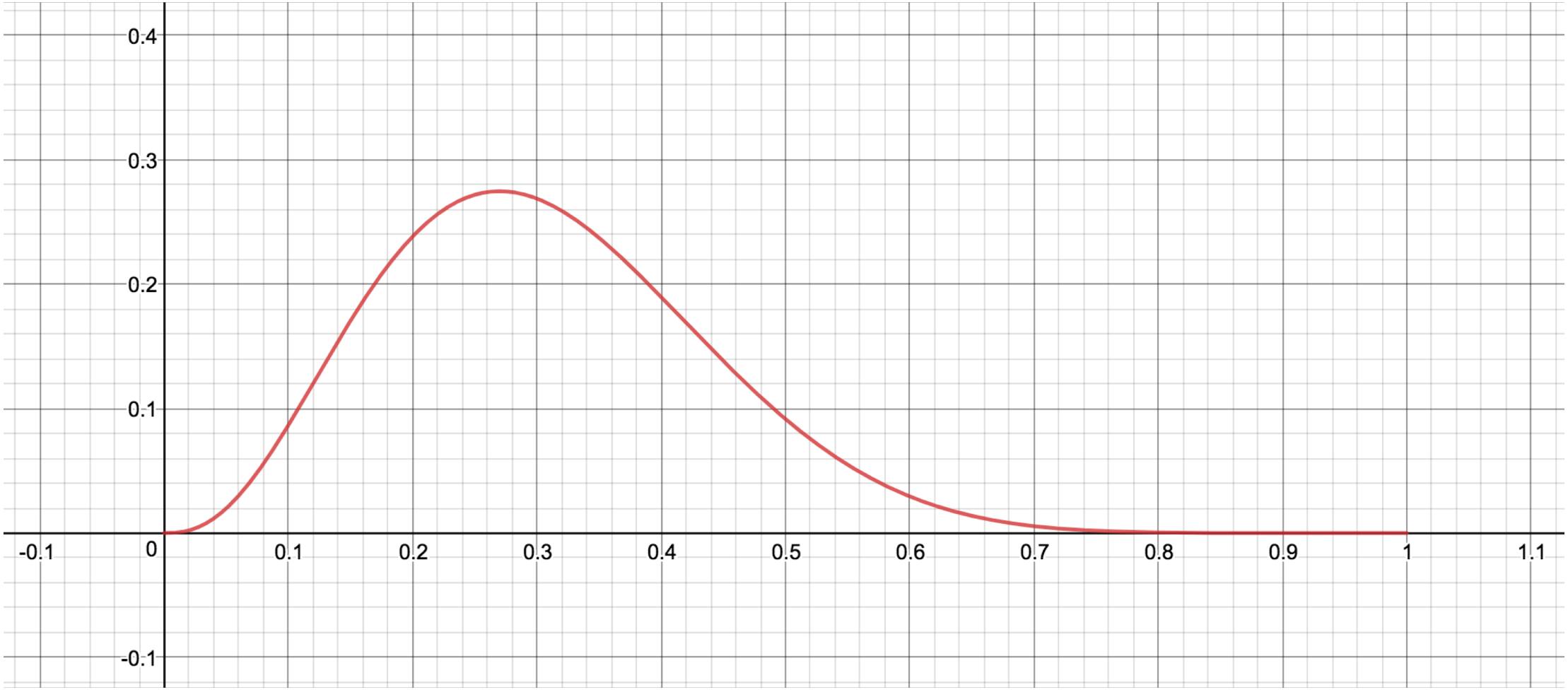
Maximum Likelihood Estimation (MLE)

Maximum Likelihood Estimation (MLE)

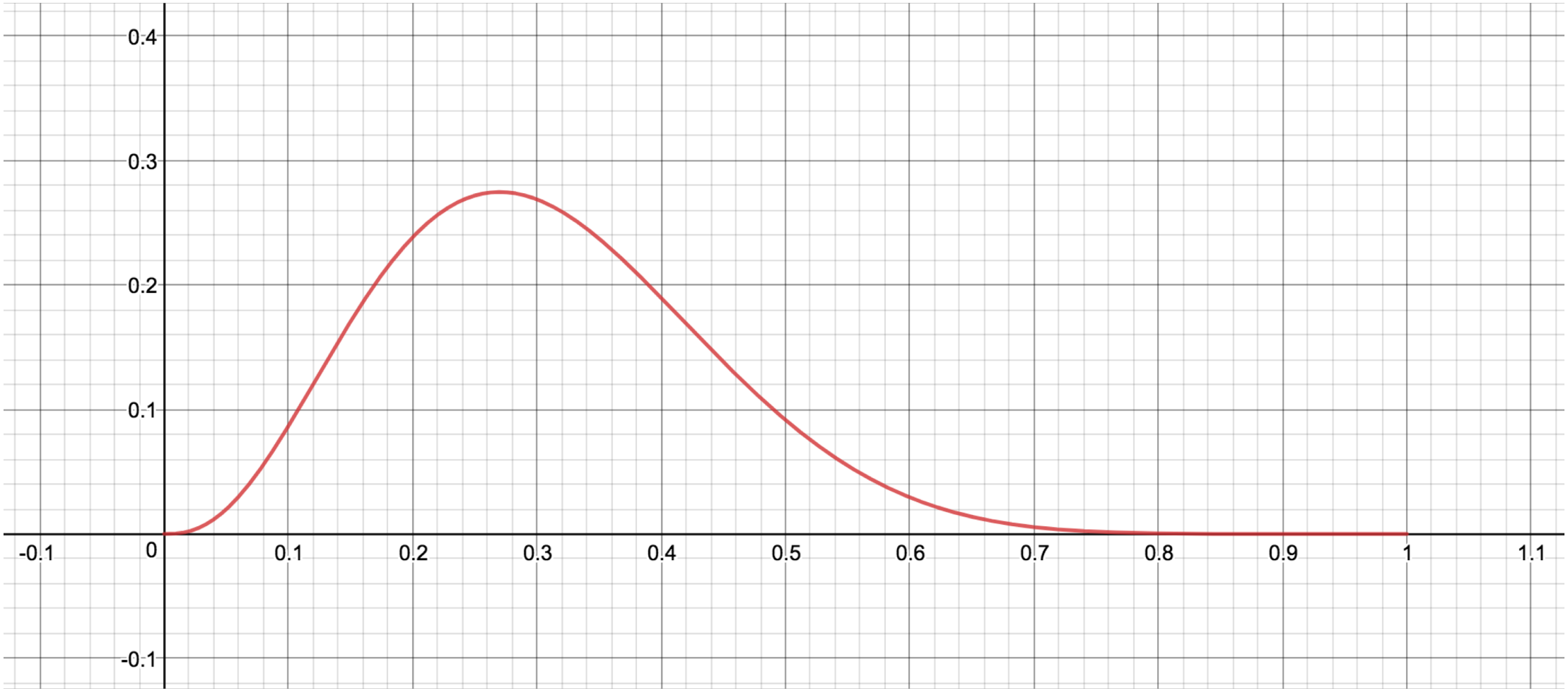
$$K \sim \mathcal{B}(n, \theta)$$

$$\begin{aligned} P(K = k) &= \binom{n}{k} \theta^k (1-\theta)^{n-k} \\ &= \frac{n!}{k!(n-k)!} \cdot \theta^k (1-\theta)^{n-k} \end{aligned}$$

Maximum Likelihood Estimation (MLE)



Optimization via Gradient Descent



Overfitting

Overfitting

Overfitting

Overfitting

- More Data
- Less features
- **Regularization**

Regularization

- Early Stopping
- 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.

   a. If it is not, increase the model size (deeper and wider).

   b. 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

```
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)

```
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

learning rate이 너무 크면 diverge 하면서 cost 가 점점 늘어난다 (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

learning rate이 너무 작으면 cost가 거의 줄어들지 않는다.

```
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

적절한 숫자로 시작해 발산하면 작게, cost가 줄어들지 않으면 크게 조정하자.

```
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]])
```

Training with Preprocessed Data

```
class MultivariateLinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(3, 1)

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

```
model = MultivariateLinearRegressionModel()
```

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

Training with Preprocessed Data

```
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.mse_loss(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()
        ))
```

Training with Preprocessed Data

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

```
Epoch    0/20 Cost: 29785.091797
Epoch    1/20 Cost: 18906.164062
Epoch    2/20 Cost: 12054.674805
Epoch    3/20 Cost: 7702.029297
Epoch    4/20 Cost: 4925.733398
Epoch    5/20 Cost: 3151.632568
Epoch    6/20 Cost: 2016.996094
Epoch    7/20 Cost: 1291.051270
Epoch    8/20 Cost: 826.505310
Epoch    9/20 Cost: 529.207336
Epoch   10/20 Cost: 338.934204
Epoch   11/20 Cost: 217.153549
Epoch   12/20 Cost: 139.206741
Epoch   13/20 Cost: 89.313782
Epoch   14/20 Cost: 57.375462
Epoch   15/20 Cost: 36.928429
Epoch   16/20 Cost: 23.835772
Epoch   17/20 Cost: 15.450428
Epoch   18/20 Cost: 10.077808
Epoch   19/20 Cost: 6.633700
```