

Softmax Classification

Softmax Classification

- Softmax
- Cross Entropy
- Low-level Implementation
- High-level Implementation
- Training Example

Import

```
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 0x7f8740076fb0>
```

Softmax

Convert numbers to probabilities with softmax.

$$P(\text{class} = i) = \frac{e^i}{\sum e^i}$$

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

PyTorch has a `softmax` function.

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

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

$$P(z=1) = \frac{e^1}{e^1 + e^2 + e^3}$$

$$P(z=2) = \frac{e^2}{e^1 + e^2 + e^3}$$

$$P(z=3) = \frac{e^3}{e^1 + e^2 + e^3}$$

Since they are probabilities, they should add up to 1. Let's do a sanity check.

```
hypothesis.sum()
```

합은 1!

```
tensor(1.)
```

Cross Entropy

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=<SoftmaxBackward>)
```

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

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







Cross Entropy

```
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)
```

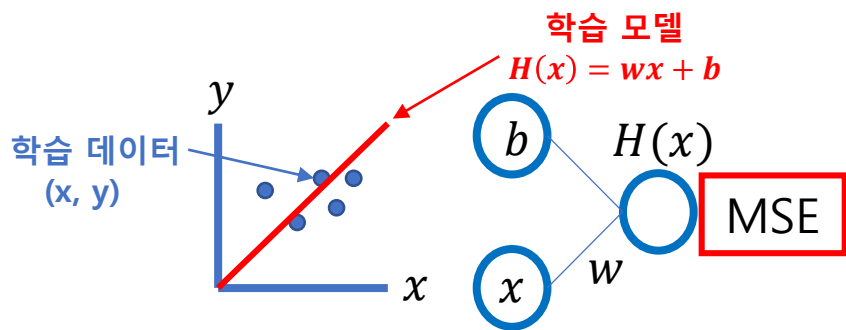
```
tensor(1.4689, grad_fn=<MeanBackward1>)
```

		1	2	3
		1	0	0
		0	1	0
		0	0	1

값 추정이 아닌 분류(Classification)일 경우 정답 값(라벨)을 one-hot 형태로 바꿔서 나타낸다.

Linear Regression => 값 추정!!

Simple Linear Regression



No sigmoid!

MSE (Mean Squared Error)

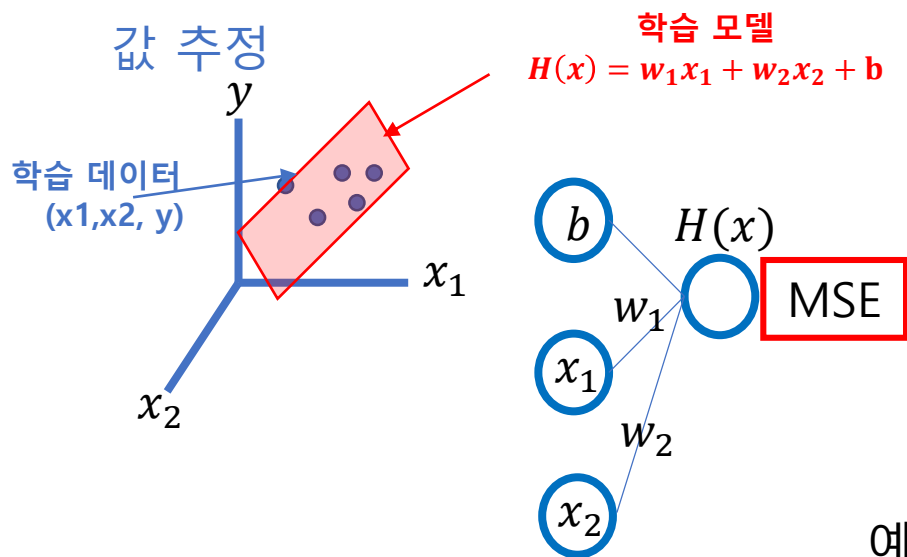
$$\frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y)^2$$



예시) 입력 : 배달거리, 출력 : 배달 시간

1. 값 추정이므로 범위제한 x (no sigmoid!)
2. 값 추정이므로 MSE로 임의의 실수 값을 추정해야 한다. (MSE)

Multivariate Linear Regression



No sigmoid!

MSE (Mean Squared Error)

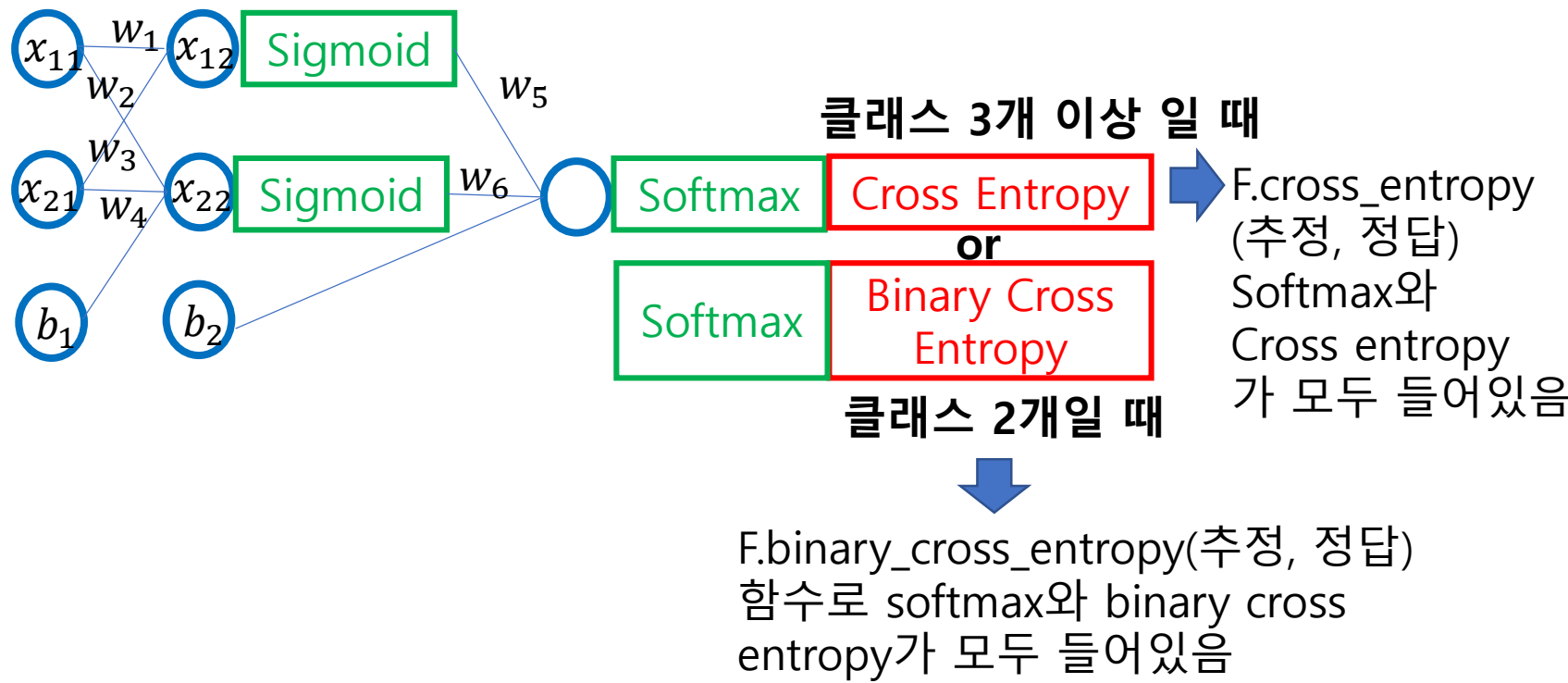
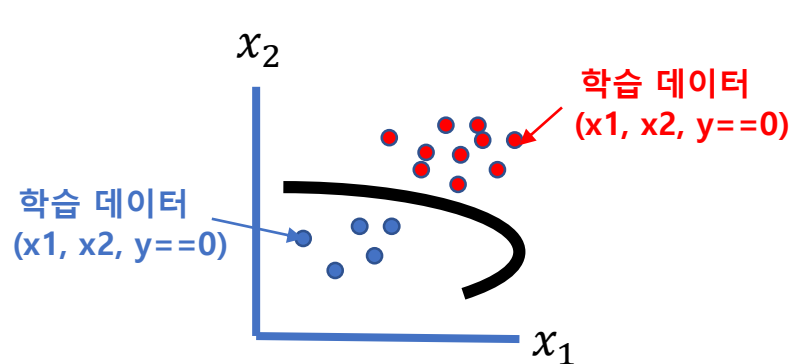
$$\frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y)^2$$



예시) 입력 : Quiz1, Quiz2, Quiz3, 출력 : Final(합산 점수)

Logistic Regression=> 값 추정, 구분(Classification)!!

=> 값 추정은 여기서 다루지 않는다. 구분(Classification)위주로 다룸



중간은 Sigmoid!=> 다양한 모양

끝에는 Softmax=> 확률 형태

Cross_entropy => 구분(Classification)

$$L = \frac{1}{N} \sum -y \log(H(x)) \quad \leftarrow \quad L = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y)^2$$

1. Sigmoid를 통해 다양한 모양을 표현 할 수 있음
2. Softmax의 경우 sigmoid와 달리 모든 합이 1이며, 확률형태로 나타낼 수 있는 장점이 있음
3. MSE를 쓰지 않는 이유는 원래 그래프를 제공하면 그래프가 복잡 해져서 Local Minima가 많이 생긴다. Cross Entropy의 log는 오히려 그래프를 단순하게 만든다.
4. 또한 0또는 1 형태의 정답인 classificatio의 경우 cross entropy가 학습이 더 잘된다.

Tips

Tips

- Overfitting and Regularization
- Training and Test Dataset
- Learning Rate
- Data Preprocessing

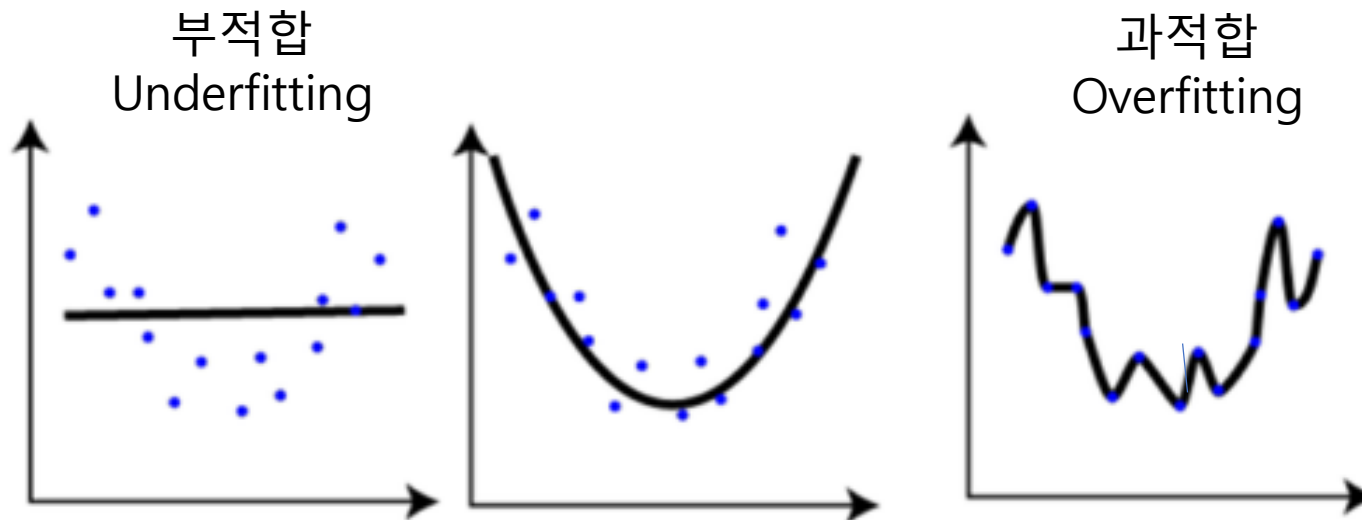
Tips

- Overfitting and Regularization
- Training and Test Dataset
- Learning Rate
- Data Preprocessing

01 Deep Learning Remind

Overfitting

- 통계에서 좋은 결과를 얻기 위해서는 전체를 대표할 수 있는 샘플
- 전체를 잘 설명할 수 있는 모델을 만들 어야함
- 부적합 : 데이터를 잘 나타내고 있지 못함
- 과적합 : 모든 데이터에 맞춰져 새로운 데이터가 들어오면 취약함
- 이상적인 해결 방법 : 데이터의 개수를 늘리는 것=>대부분 데이터를 늘리기는 어렵다.
- 오컴의 면도날 : 사고 절약의 원리 => 두가지 주장이 있다면 그나마 간단한 쪽을 택하자



01 Deep Learning Remind

지도 학습 (Supervised Learning)

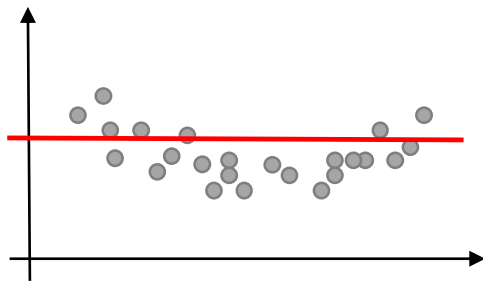
회귀의 과적합(overfitting)과 부적합(underfitting)

과적합

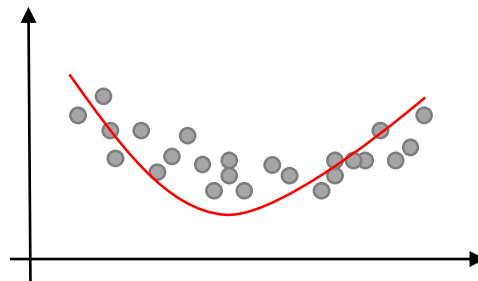
지나치게 복잡한 모델(함수) 사용

부적합

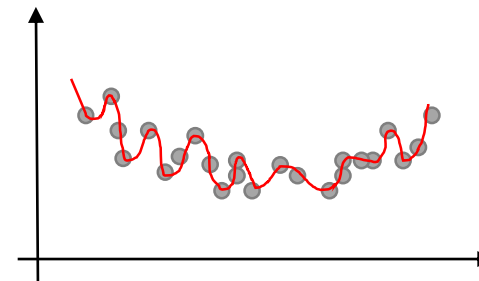
지나치게 단순한 모델(함수) 사용



부적합(underfitting)



정적합(good fitting)



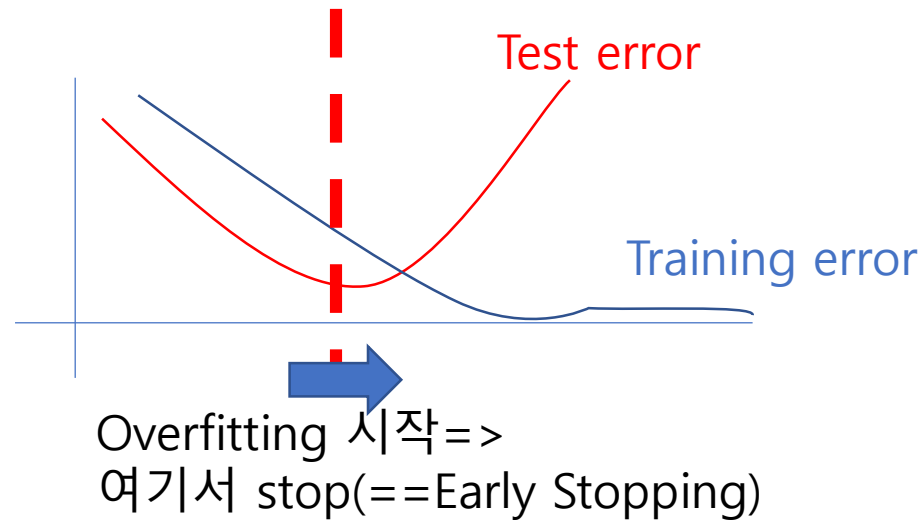
과적합(overfitting)

Regularization

- Early Stopping
- Reducing Network Size
- Weight Decay
- Dropout

Regularization

- Early Stopping



```
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])
```

Regularization

- Early Stopping

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)
```


Regularization

- Early Stopping

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

Regularization

- Early Stopping

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()  
    ))
```

Regularization

- Early Stopping

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
```

Regularization

- 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

Regularization

- Learning Rate

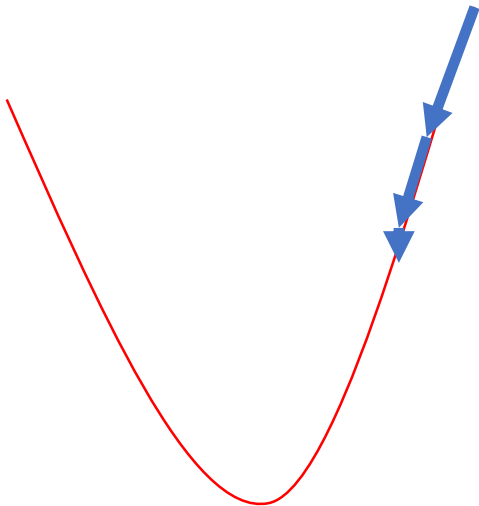
learning rate이 너무 작으면 cost가 거의 줄어들지 않는다.

```
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



Regularization

- 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
```

Regularization

- 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
```

Regularization

- Data Processing

```
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]])
```


Regularization

- Data Processing

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

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]])
```

Regularization

- Data Processing

```
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)
```

Regularization

- Data Processing

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

Regularization

- Data Processing

```
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
```

01 Deep Learning Remind

- Weight Decay

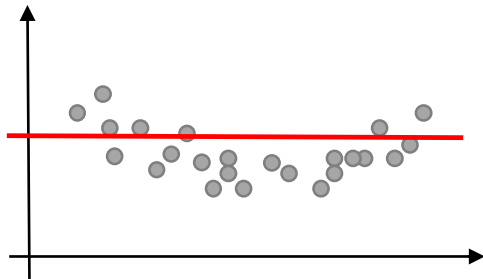
회귀의 과적합(overfitting) 대응 방법

모델의 복잡도(model complexity)를 성능 평가에 반영

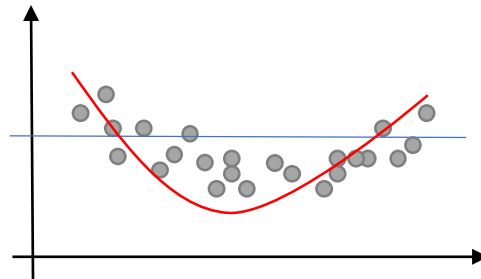
=> 모델의 복잡도가 낮아지는 방향으로 최적화가 됨

목적함수 = 오차의 합 + (가중치)*(모델 복잡도)

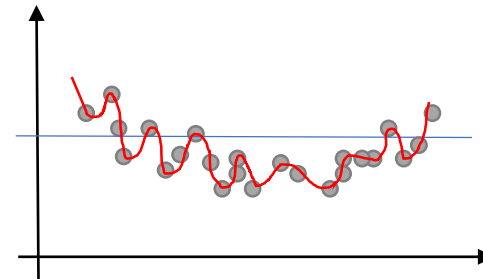
↑
벌점(penalty) 항



부적합(underfitting)



정적합(good fitting)



과적합(overfitting)

Training Level with Low Level Cross Entropy

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

Training Level with F.Cross_Entropy

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

위의 코드에서 그리는 모델 네트워크 그림 그려 보기!! (활성화 함수 및 Loss 함수 포함해서)

간편해짐

High Level Implementation with nn.module

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

```
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)
```

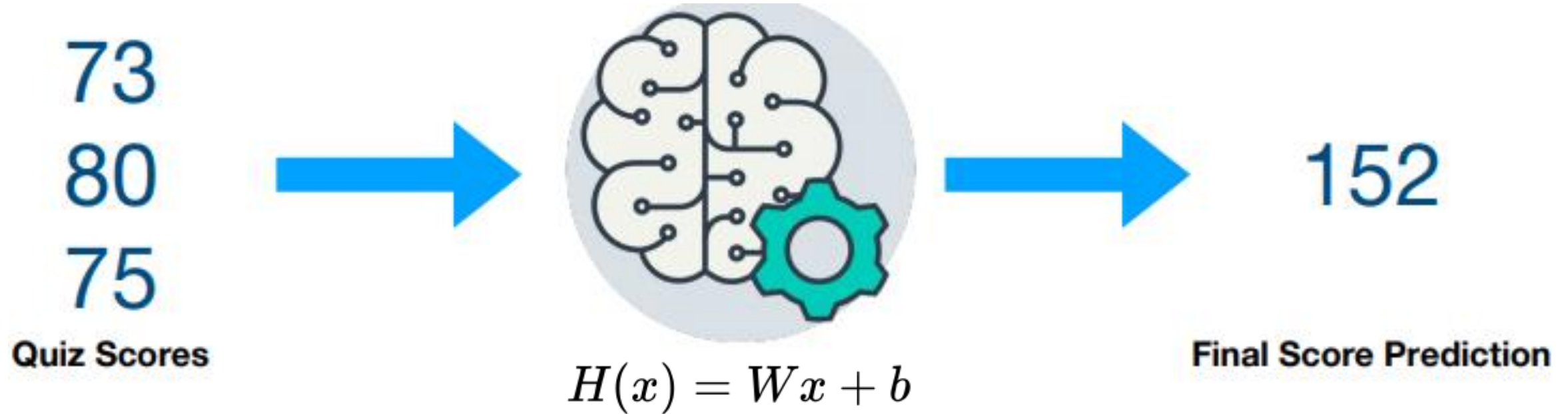
```
model = SoftmaxClassifierModel()
```

model=torch.nn.Sequential(linear, sigmoid).to(device)로도
해보기

Loading Data

Multivariate Linear Regression

입력이 여러개



Quiz 1 (x1)	Quiz 2 (x2)	Quiz 3 (x3)	Final (y)
73	80	75	152
93	88	93	185
89	91	80	180
96	98	100	196
73	66	70	142

Multivariate Linear Regression

Epoch	0/20 hypothesis: tensor([0., 0., 0., 0., 0.]) Cost: 29661.800781
Epoch	1/20 hypothesis: tensor([67.2578, 80.8397, 79.6523, 86.7394, 61.6605]) Cost: 9298.520508
Epoch	2/20 hypothesis: tensor([104.9128, 126.0990, 124.2466, 135.3015, 96.1821]) Cost: 2915.713131
Epoch	3/20 hypothesis: tensor([125.9942, 151.4381, 149.2133, 162.4896, 115.5097]) Cost: 915.040527
Epoch	4/20 hypothesis: tensor([137.7968, 165.6247, 163.1911, 177.7112, 126.3307]) Cost: 287.936005
Epoch	5/20 hypothesis: tensor([144.4044, 173.5674, 171.0168, 186.2332, 132.3891]) Cost: 91.371017
Epoch	6/20 hypothesis: tensor([148.1035, 178.0144, 175.3980, 191.0042, 135.7812]) Cost: 29.758139
Epoch	7/20 hypothesis: tensor([150.1744, 180.5042, 177.8508, 193.6753, 137.6805]) Cost: 10.445305
Epoch	8/20 hypothesis: tensor([151.3336, 181.8983, 179.2240, 195.1707, 138.7440]) Cost: 4.391228
Epoch	9/20 hypothesis: tensor([151.9824, 182.6789, 179.9928, 196.0079, 139.3396]) Cost: 2.493135
Epoch	10/20 hypothesis: tensor([152.3454, 183.1161, 180.4231, 196.4765, 139.6732]) Cost: 1.897688
Epoch	11/20 hypothesis: tensor([152.5485, 183.3610, 180.6640, 196.7389, 139.8602]) Cost: 1.710541
Epoch	12/20 hypothesis: tensor([152.6620, 183.4982, 180.7988, 196.8857, 139.9651]) Cost: 1.651413
Epoch	13/20 hypothesis: tensor([152.7253, 183.5752, 180.8742, 196.9678, 140.0240]) Cost: 1.632387
Epoch	14/20 hypothesis: tensor([152.7606, 183.6184, 180.9164, 197.0138, 140.0571]) Cost: 1.625923
Epoch	15/20 hypothesis: tensor([152.7802, 183.6427, 180.9399, 197.0395, 140.0759]) Cost: 1.623412
Epoch	16/20 hypothesis: tensor([152.7909, 183.6565, 180.9530, 197.0538, 140.0865]) Cost: 1.622141
Epoch	17/20 hypothesis: tensor([152.7968, 183.6643, 180.9603, 197.0618, 140.0927]) Cost: 1.621253
Epoch	18/20 hypothesis: tensor([152.7999, 183.6688, 180.9644, 197.0662, 140.0963]) Cost: 1.620500
Epoch	19/20 hypothesis: tensor([152.8014, 183.6715, 180.9666, 197.0686, 140.0985]) Cost: 1.619770
Epoch	20/20 hypothesis: tensor([152.8020, 183.6731, 180.9677, 197.0699, 140.1000]) Cost: 1.619033

Final(y)

152

185

180

196

142

Data in the Real World

- 복잡한 머신러닝 모델을 학습하기 위해서는 엄청난 양의 데이터가 필요
- 대부분 데이터셋은 적어도 수십만개의 데이터를 제공함



자그마치 14,197,122 개의 이미지

Multivariate Linear Regression

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

Data in the Real World: Problem

- 엄청난 양의 데이터를 한번에 학습시킬 수 없음
 - 너무 느리다.
 - 하드웨어적으로 불가능하다.(수십만개의 네트워크를 동시에 돌리는 셈)
- 일부분의 데이터로만 학습하면 어떨지?

Multivariate Linear Regression

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

Gradient Descent Optimization Algorithm

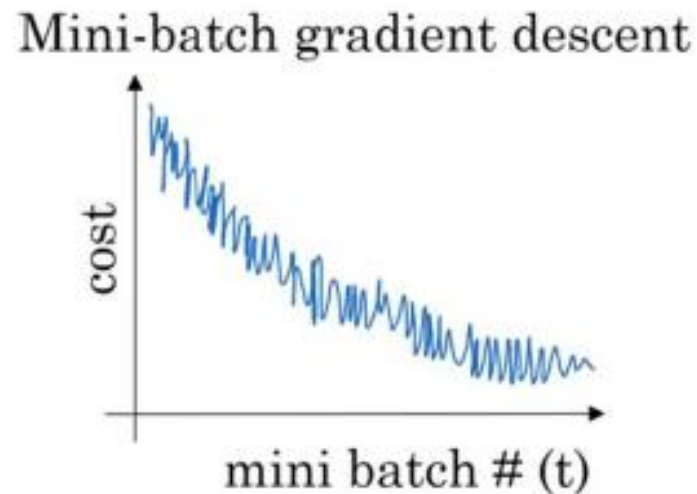
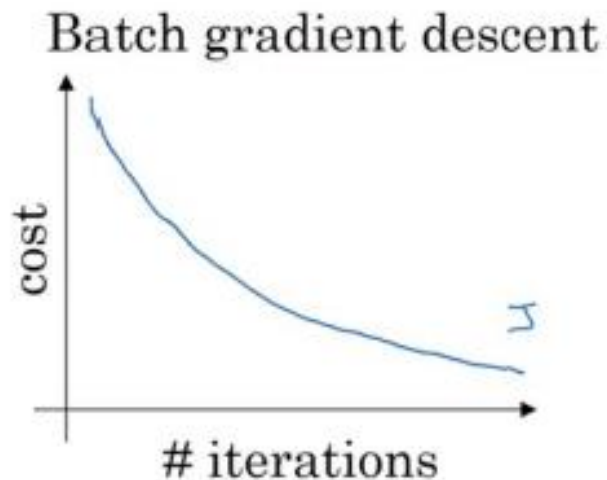
- Batch Gradient Descent (BGD)
 - 한번의 step (epoch)에 전체 training dataset을 사용하는 방법
 - 너무 느리다.
 - 하드웨어적으로 불가능하다.(수십만개의 네트워크를 동시에 돌리는 셈)
- Stochastic Gradient Descent (SGD)
 - 한번의 step (epoch)에 한 개의 training dataset만을 사용하는 방법
 - 너무 느리다.
 - Global Minima에 수렴할 가능성이 낮다.

Multivariate Linear Regression

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

Gradient Descent Optimization Algorithm

- MiniBatch Gradient Descent (MGD)
 - 한번의 step (epoch)에 일정 batch단위의 training dataset만큼 돌리는 것
 - 업데이트를 좀 더 빠르게 할 수 있다.
 - 전체 데이터를 쓰지 않아서 잘못된 방향으로 업데이트 할 수도 있다.



Multivariate Linear Regression

<http://shuuki4.github.io/deep%20learning/2016/05/20/Gradient-Descent-Algorithm-Overview.html>

PyTorch Dataset

```
from torch.utils.data import Dataset
```

```
class CustomDataset(Dataset):
```

```
    def __init__(self):
```

```
        self.x_data = [[73, 80, 75],  
                        [93, 88, 93],  
                        [89, 91, 90],  
                        [96, 98, 100],  
                        [73, 66, 70]]
```

```
        self.y_data = [[152], [185], [180], [196], [142]]
```

```
    def __len__(self):
```

```
        return len(self.x_data)
```

```
    def __getitem__(self, idx):
```

```
        x = torch.FloatTensor(self.x_data[idx])
```

```
        y = torch.FloatTensor(self.y_data[idx])
```

```
        return x, y
```

```
dataset = CustomDataset()
```

이미지의 경우

1. 이미지 데이터를 불러온다.
2. 리스트 형태로 배열한다.
(total_batch, height, width)

- torch.utils.data.Dataset 상속
- __len__()
 - 이 데이터셋의 총 개수
- __getitem__()
 - 어떠한 인덱스 idx를 받았을 때 그에 상응하는 입출력 데이터를 반환

PyTorch Dataset

```
from torch.utils.data import DataLoader
```

```
dataloader = DataLoader(  
    dataset,  
    batch_size=2,  
    shuffle=True,  
)
```

- torch.utils.data.DataLoader 사용
- Batch_size=2
 - 각 minibatch의 크기
 - 통상적으로 2의 제곱수로 설정 (16, 32, 64, 128...)
- Shuffle=True
 - Epoch 마다 데이터셋을 섞어서, 데이터가 학습되는 순서를 바꾼다,

MNIST

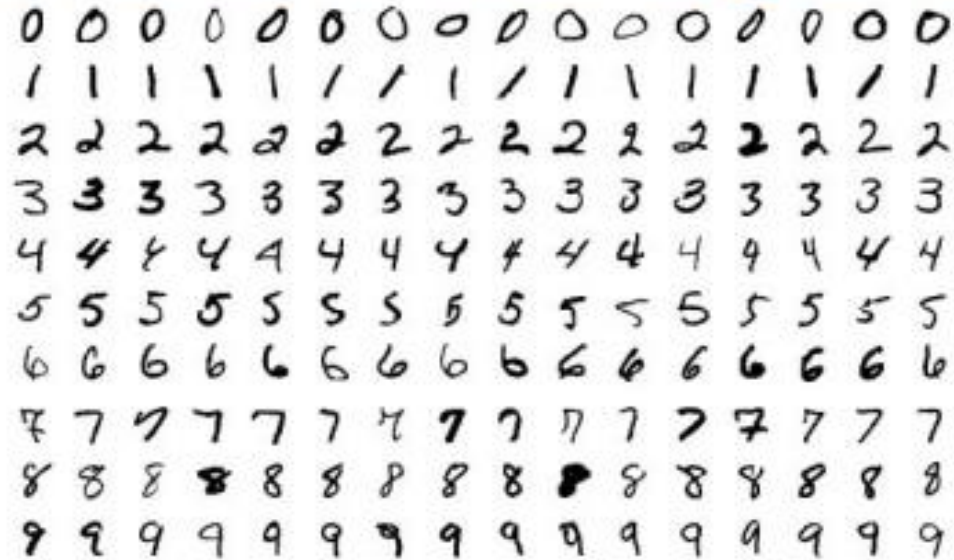
Introduction

MNIST Introduction

- What is MNIST?
- Code MNIST Introduction

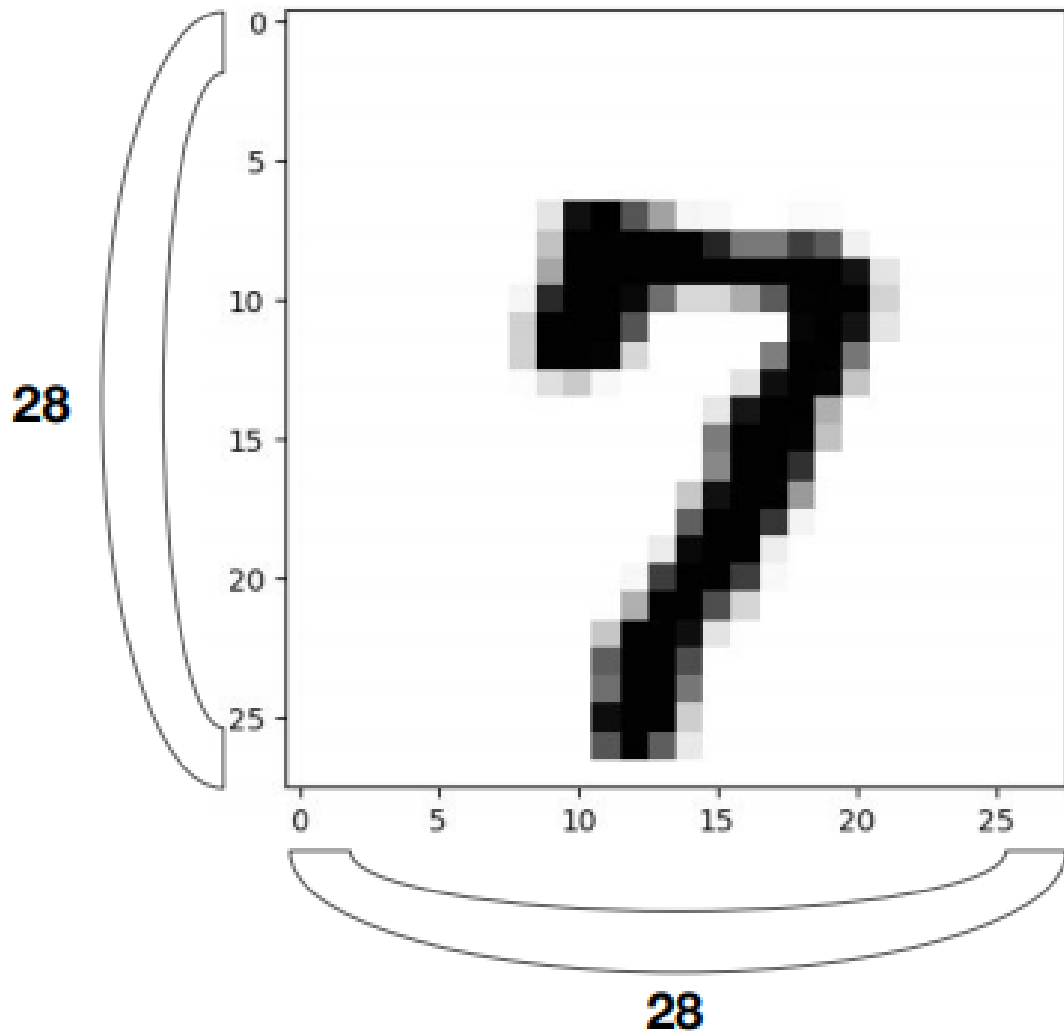
What is MNIST?

MNIST: handwritten digits dataset



- training set images (9912422 bytes; 60,000 samples)
- <http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz>
- training set labels (28881 bytes)
- <http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz>
- test set images (1648877 bytes; 10,000 samples)
- <http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz>
- test set labels (4542 bytes)
- <http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz>

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, and common image transformations for computer vision.

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

Torchvision 설치법

Pytorch버전부터 확인

```
>>> import torch
>>> print(torch.__version__)
1.1.0
```

버전이 1.1.0보다 낮으면 torchvision설치가 안되므로 pytorch 버전을 올려야 한다.

⇒ pytorch.org에 들어가서 아래의 그림을 조건에 맞게 선택하면 그 밑에 입력해야 할 명령어가 나온다.

⇒ 명령어를 치면 pytorch가 1.10 이상의 버전으로 업데이트 된다.

OS	Linux	MacOS	Windows	
Package Manager	conda	pip	Source	
Python	2.7	3.5	3.6	3.7
CUDA	8	9.0	9.2	None

Run this command:

```
conda install pytorch-cpu -c pytorch
pip3 install torchvision
```

Torchvision 설치법

Window의 CPU용 torchvision을 설치하기 위해서는 다음과 같은 명령어를 입력

To install this package with conda run:

```
conda install -c pytorch torchvision-cpu
```

다음과 같은 명령어를 입력해서 에러가 안 뜬다면 설치 완료

```
>>> import torchvision
>>>
```


Reading data

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

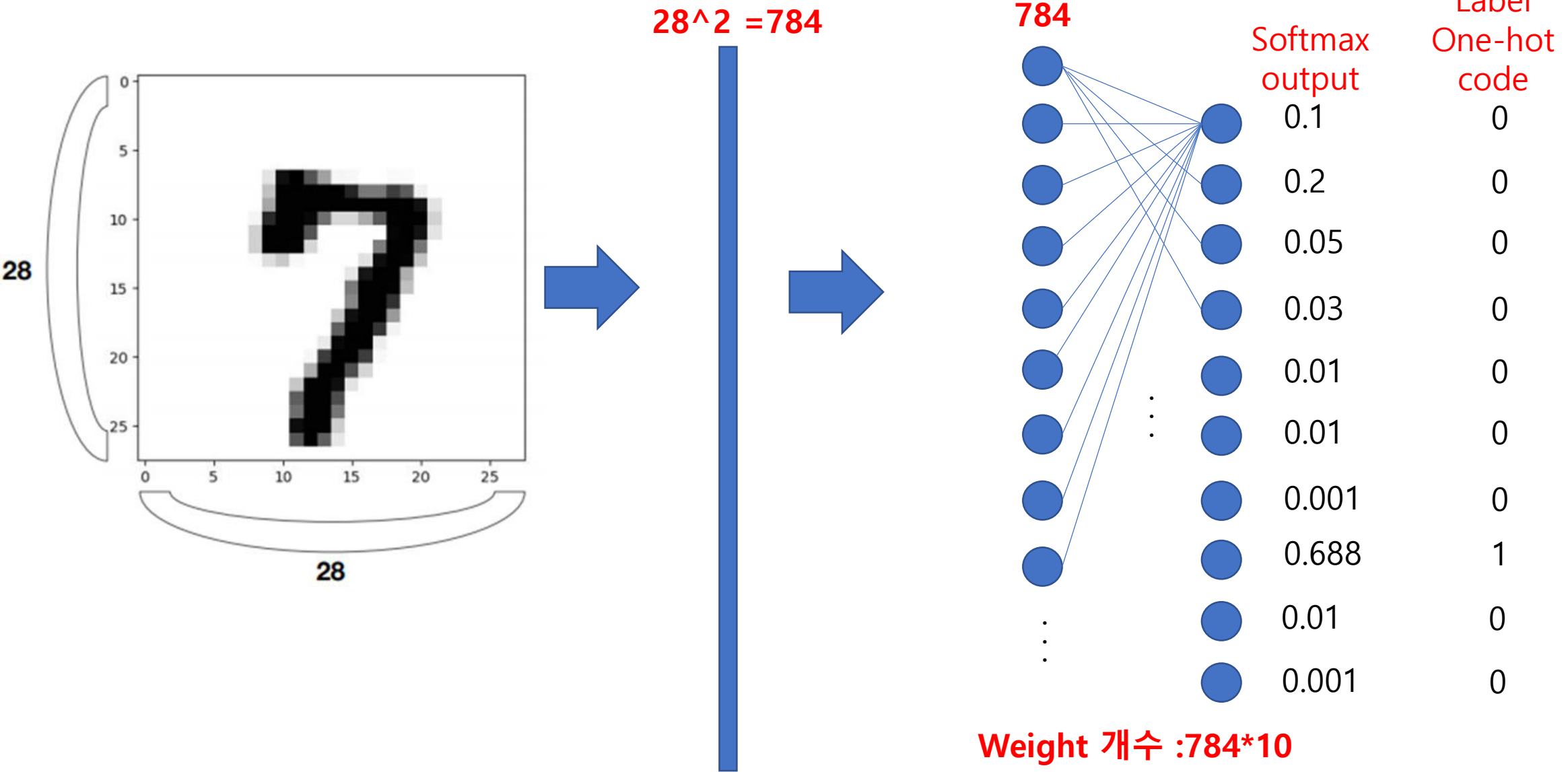
mnist_test = datasets.MNIST(root="MNIST_data/", train=False, transform=transforms.ToTensor(),
                             download=True)

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

- One epoch => 모든 학습 데이터(Training data)가 한번 학습 된 것을 One epoch으로 기준
- Batch size => forward와 backward할 때 한번에 들어가는 데이터 개수(메모리 허용량 까지만 가능)
- Iterations => 하나의 batch 단위의 학습데이터가 학습 된 것을 one iterations으로 기준

Softmax, Onehot-coding



Softmax, Onehot-coding

```
# 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)
        optimizer.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))
```

Test

```
# 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())
```

결과 보기!

Visualization

```
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()
```

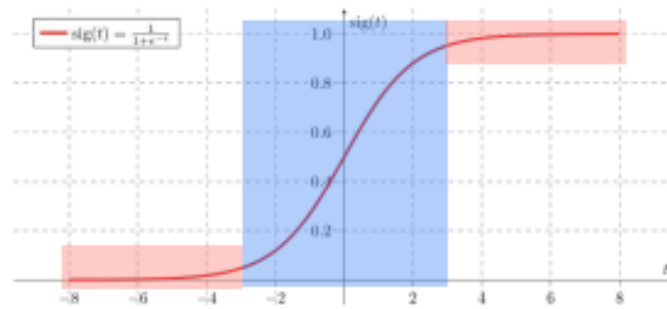
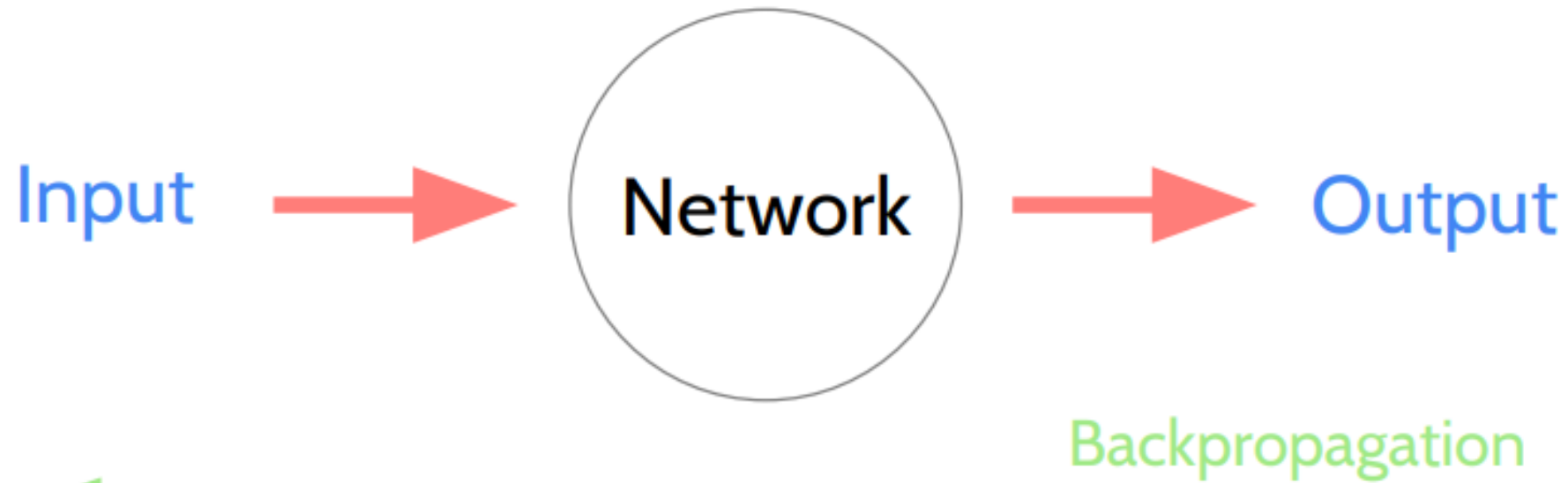
결과 보기!

ReLU
(Rectified Linear Unit)

ReLU

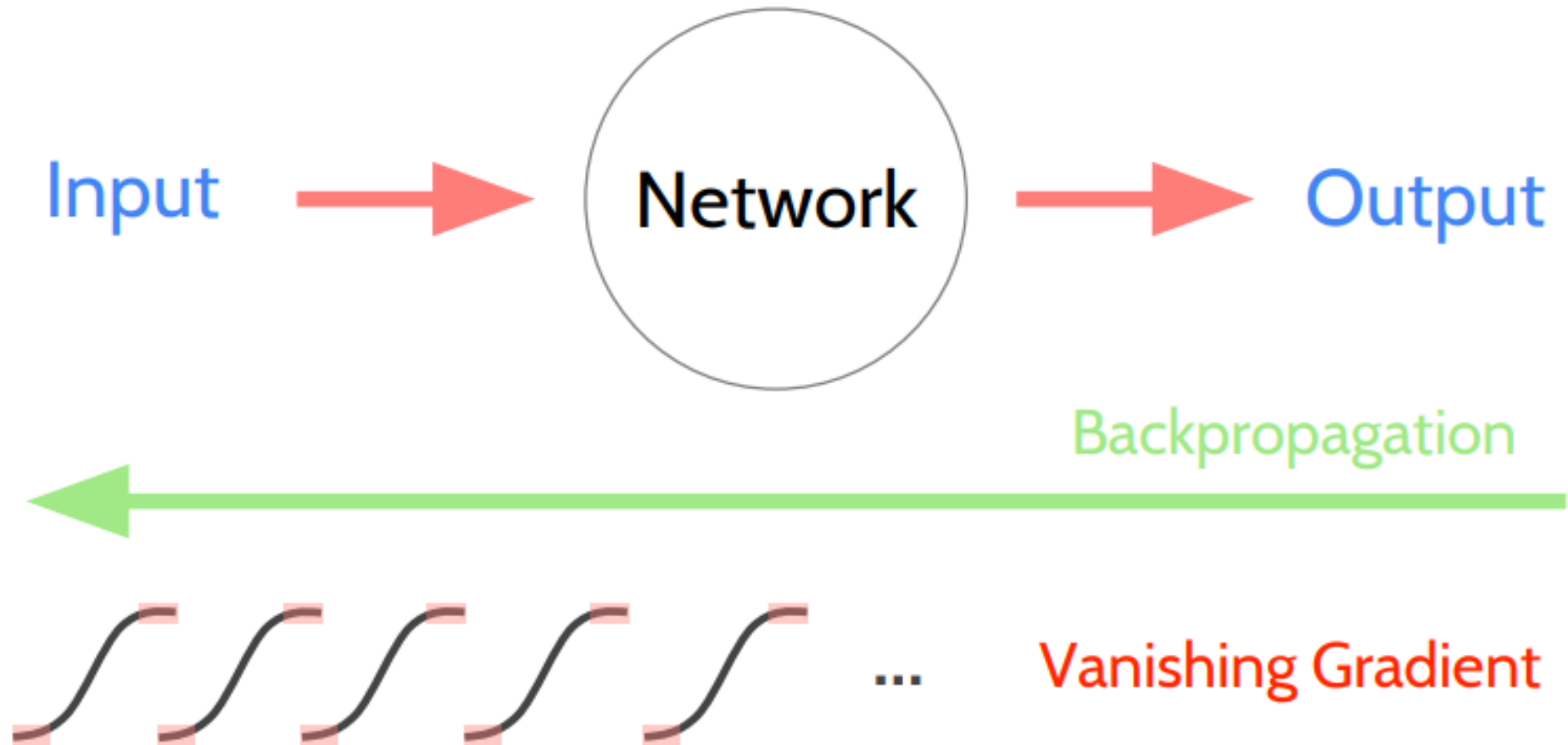
- Problem of Sigmoid
- ReLU
- Optimizer in PyTorch
- Review: MNIST
- Code: mnist_softmax
- Code: mnist_nn

Problem of Sigmoid

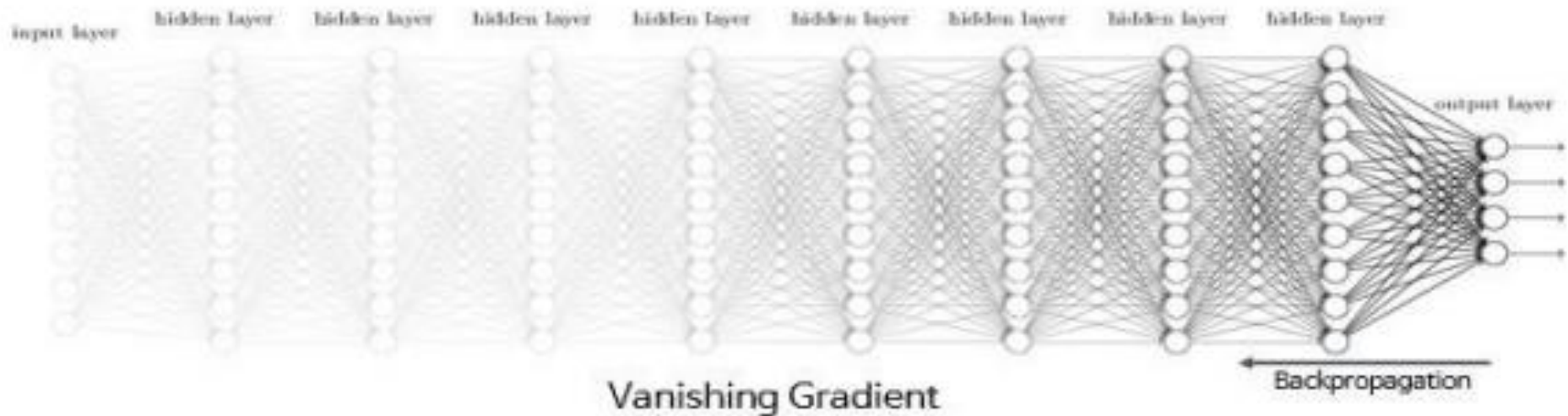
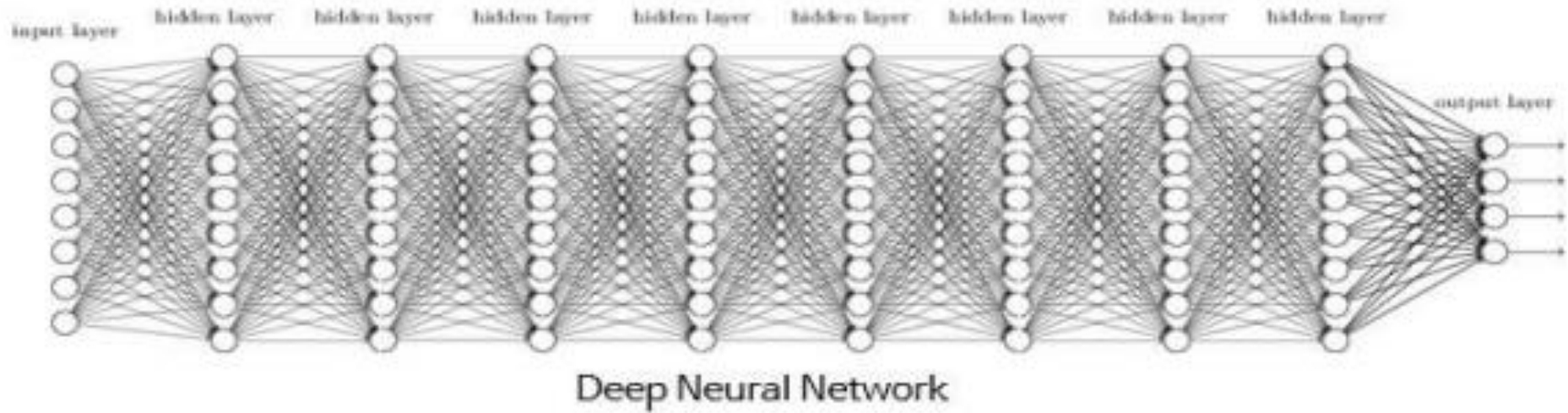


Vanishing Gradient

Problem of Sigmoid

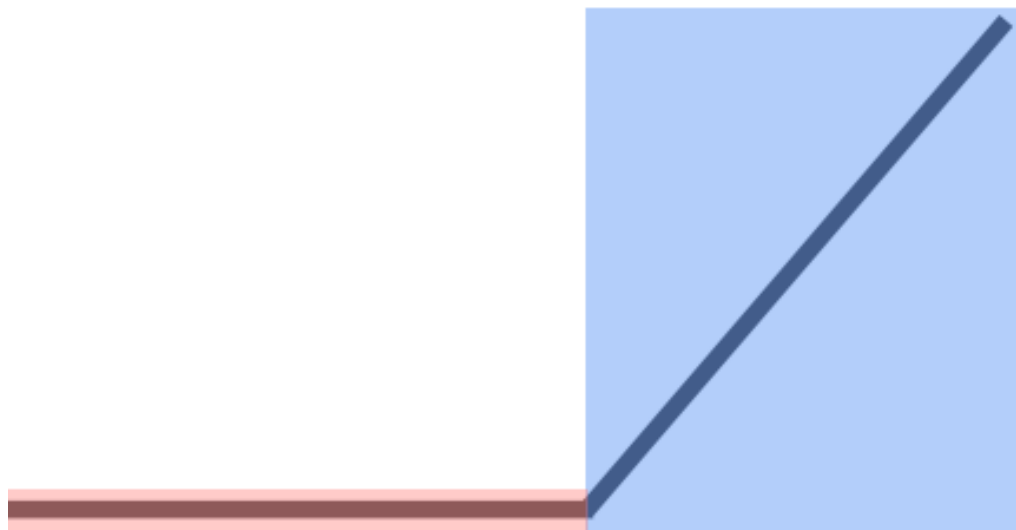


Problem of Sigmoid



ReLU

$$f(x) = \max(0, x)$$



```
x = torch.nn.sigmoid(x)
```

```
x = torch.nn.relu(x)
```

```
torch.nn.sigmoid(x)
```

```
torch.nn.tanh(x)
```

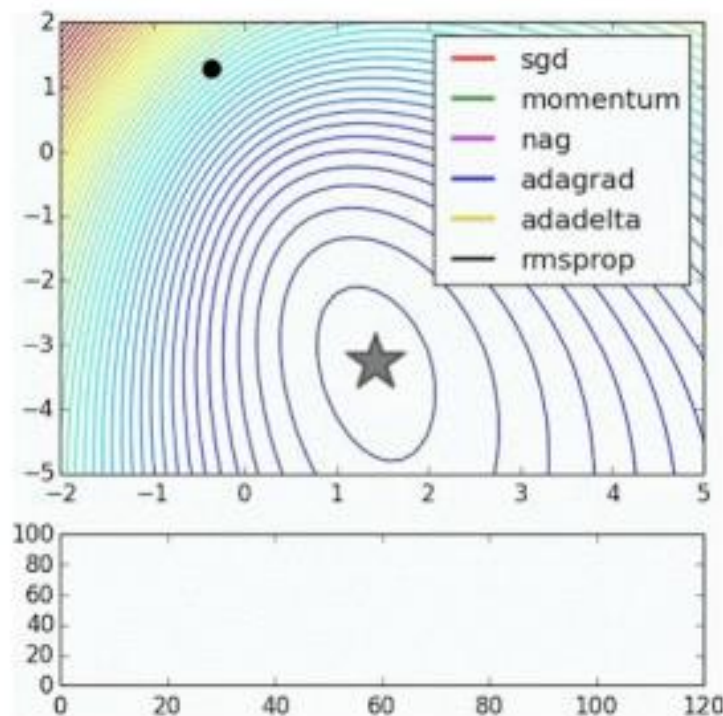
```
torch.nn.relu(x)
```

```
torch.nn.leaky_relu(x, 0.01)
```

ReLU

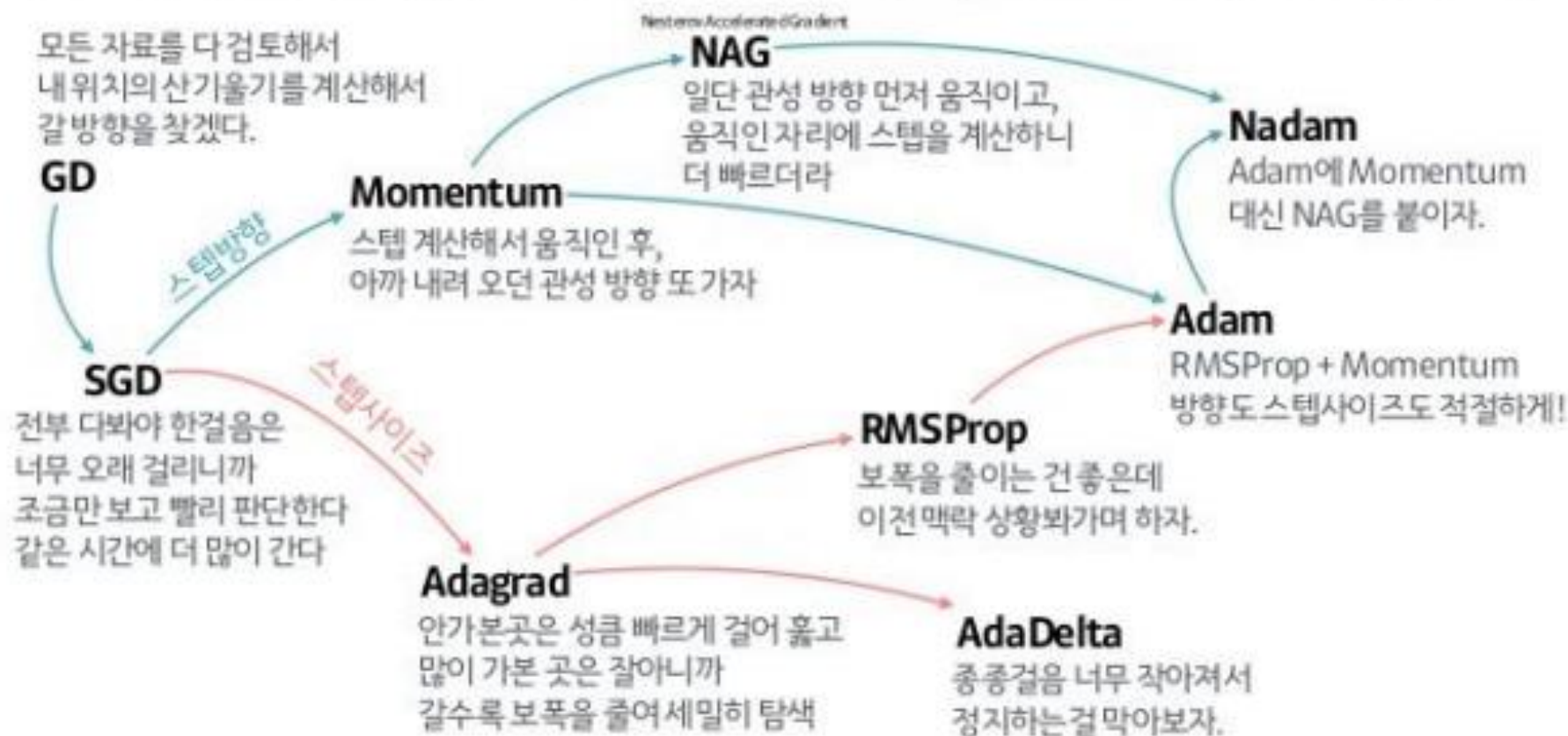
[`torch.optim`](#) is a package implementing various optimization algorithms. Most commonly used methods are already supported, and the interface is general enough, so that more sophisticated ones can be also easily integrated in the future.

- `torch.optim.SGD`
- `torch.optim.Adadelta`
- `torch.optim.Adagrad`
- `torch.optim.Adam`
- `torch.optim.SparseAdam`
- `torch.optim.Adamax`
- `torch.optim.ASGD`
- `torch.optim.LBFGS`
- `torch.optim.RMSprop`
- `torch.optim.Rprop`



ReLU

산 내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보



Review: 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)
```

Review: mnist_softmax

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
...

# 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)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(linear.parameters(), lr=learning_rate)
```


Review: mnist_softmax_Training

```
total_batch = len(data_loader)
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))
```

결과 보기!

Review: mnist_nn

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
...

# MNIST data image of shape 28 * 28 = 784
linear1 = torch.nn.Linear(784, 256, bias=True).to(device)
linear2 = torch.nn.Linear(256, 256, bias=True).to(device)
linear3 = torch.nn.Linear(256, 10, bias=True).to(device)
relu = torch.nn.ReLU()

# Initialization
torch.nn.init.normal_(linear1.weight)
torch.nn.init.normal_(linear2.weight)
torch.nn.init.normal_(linear3.weight)

# model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3).to(device)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device) # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

남은 부분
코드 짜서
결과 보기!

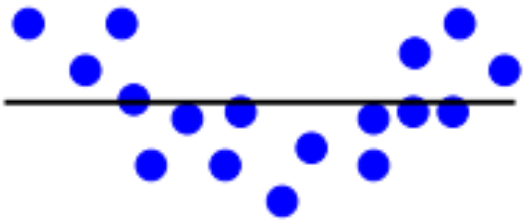
Dropout

Dropout

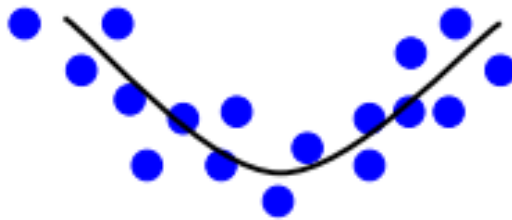
- Overfitting
- Dropout
- Code: `mnist_nn_dropout`

Overfitting

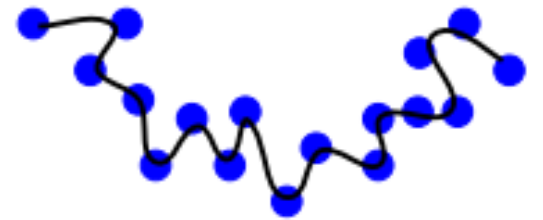
Underfitting



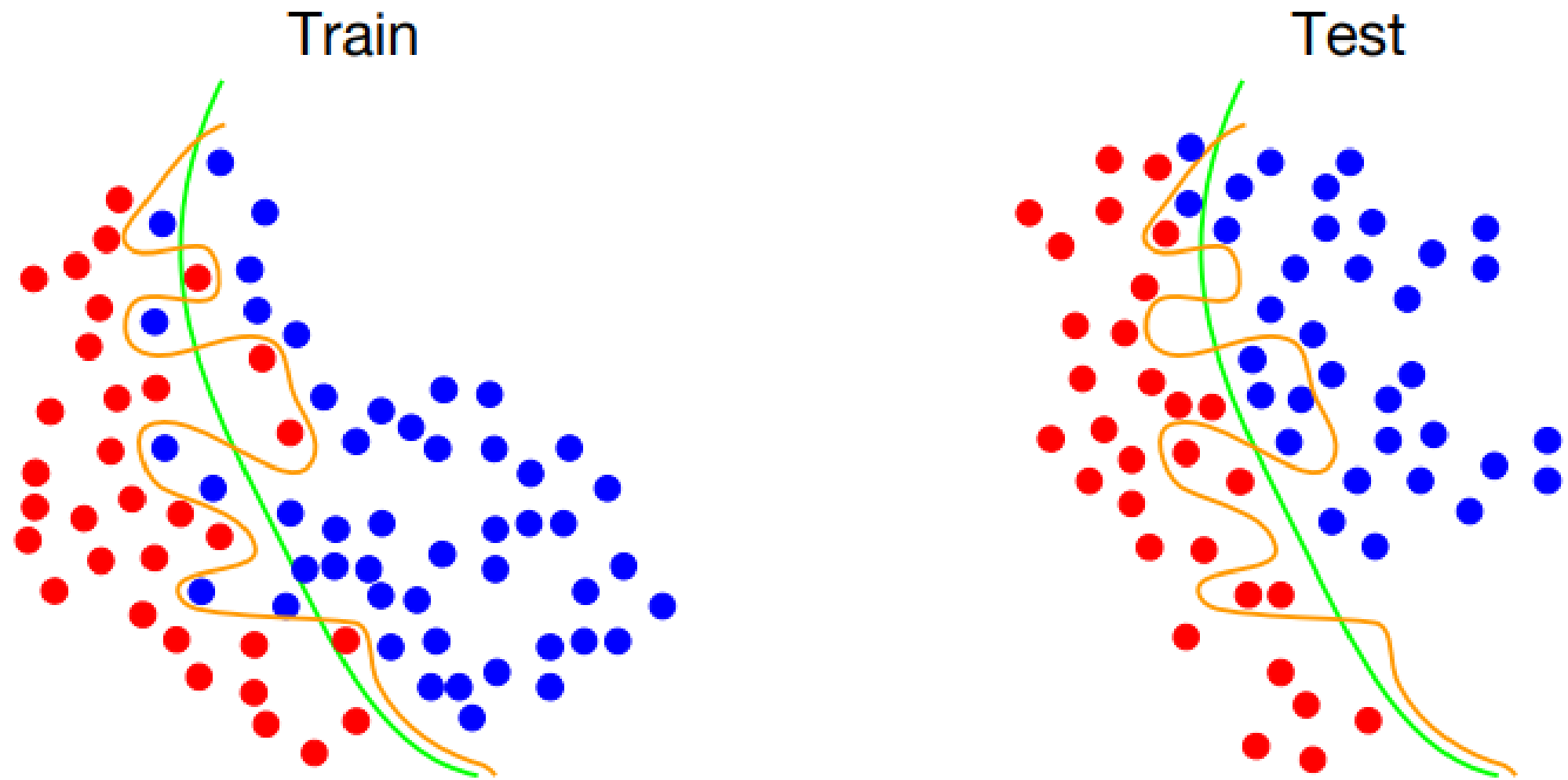
Good



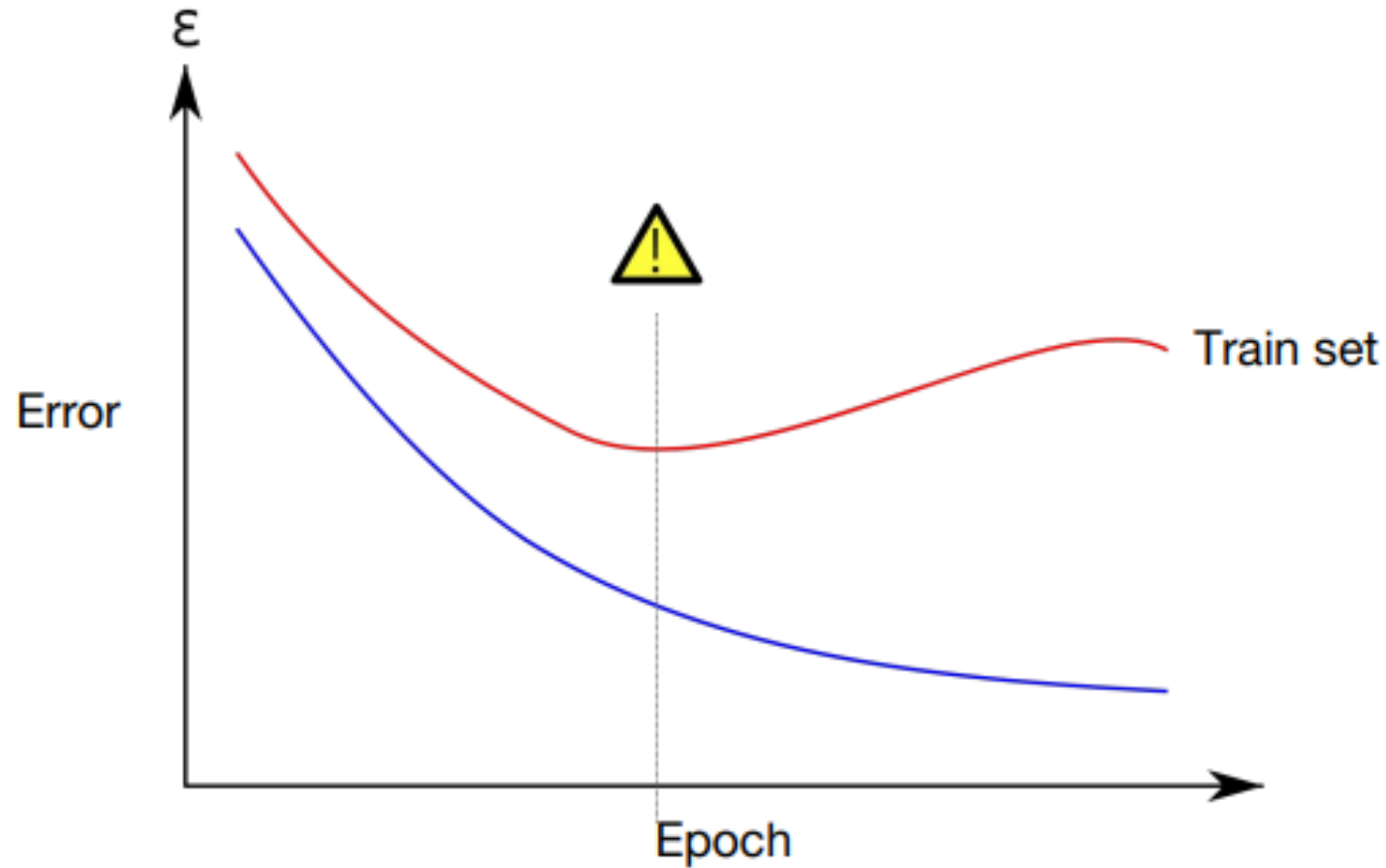
Overfitting



Overfitting



Overfitting

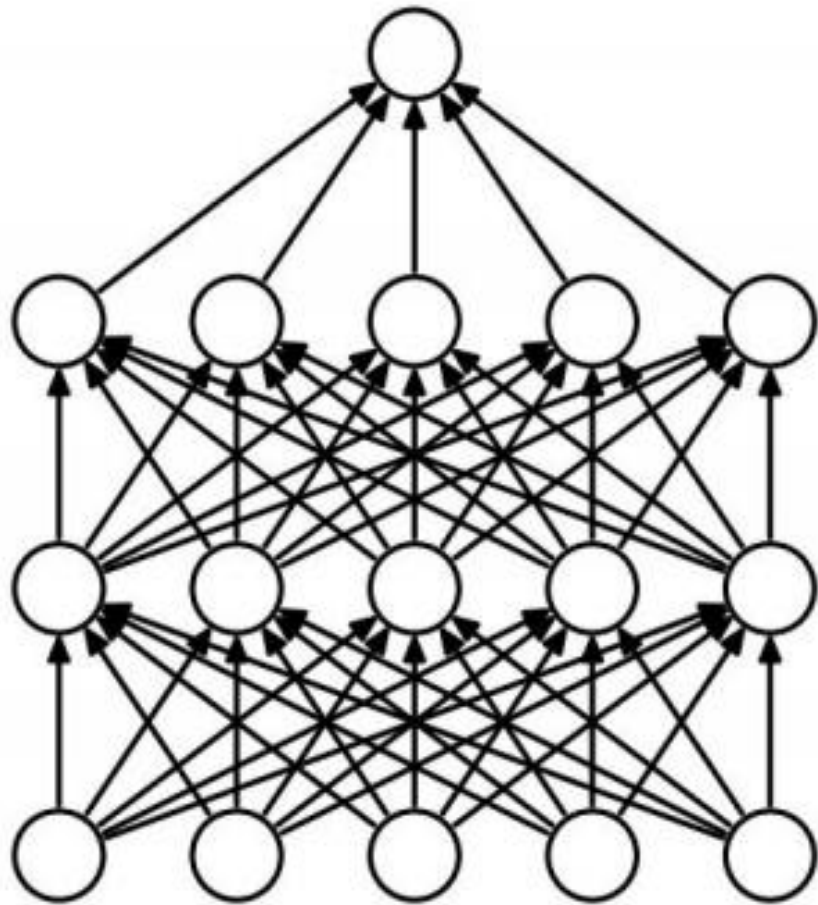


- Very high accuracy on the training dataset (e.g., 0.99)
- Poor accuracy on the test dataset (e.g., 0.85)

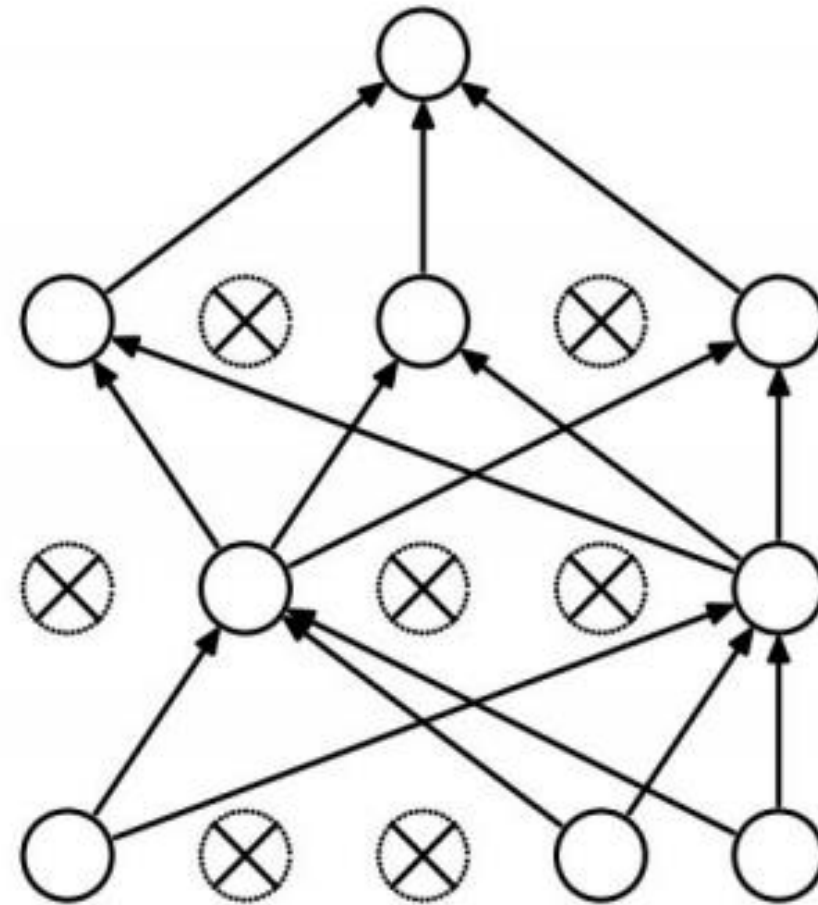
Solutions for overfitting

- More training data
- Reduce the number of features
- Regularization
- **Dropout!**

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Code: mnist_nn_dropout

...

nn layers

```
linear1 = torch.nn.Linear(784, 512, bias=True)
```

```
linear2 = torch.nn.Linear(512, 512, bias=True)
```

```
linear3 = torch.nn.Linear(512, 512, bias=True)
```

```
linear4 = torch.nn.Linear(512, 512, bias=True)
```

```
linear5 = torch.nn.Linear(512, 10, bias=True)
```

```
relu = torch.nn.ReLU()
```

```
dropout = torch.nn.Dropout(p=drop_prob)
```

model

```
model = torch.nn.Sequential(linear1, relu, dropout,  
                             linear2, relu, dropout,  
                             linear3, relu, dropout,  
                             linear4, relu, dropout,  
                             linear5).to(device)
```

남은 부분
코드 짜서
결과 보기!

From Next Class..

- CNN(Convolutional Neural Network) 개념
- CNN을 이용한 MNIST(sigmoid, relu, dropout, data processing...)
- Dataloader를 이용하여 직접 만들어서 MNIST TEST 뽑아 보기
- CNN을 이용한 CIFAR-10
- 나만의 데이터베이스를 만들어서 CNN으로 구분해보기