AI Programming 복습 3

한국항공대학교
KOREA AEROSPACE UNIVERSITY

1

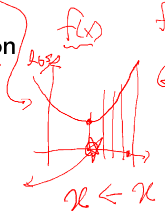
지난목차

- Linear Algrebra — Scalar Vector matrix tensor
- Derivative — $\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$ 출력차이 / 입력 차이
- Gradient Descent
- Back-propagation
- Gradient Descent of Loss Function

$f(x) \dashrightarrow f(\vec{x})$

Gradient

$\nabla f(x) = [\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n}]^T$



$x \leftarrow x - \gamma \nabla f(x)$

$\nabla_w loss$

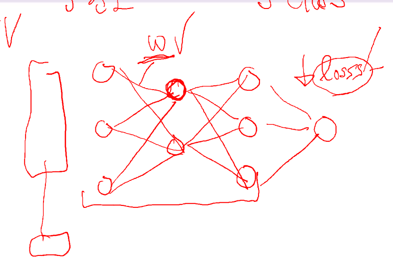
2

목차

- Vectorization I
- Vectorization II
- Pytorch의 소개
- Class: 'torch.Tensor' 연산
- AUTOGRAD
- torch.nn
- PyTorch.nn의 응용 – model
- PyTorch.nn의 응용 – training

3

3원소 3 class softmax



Vectorization I

4

인공신경망

ReLU

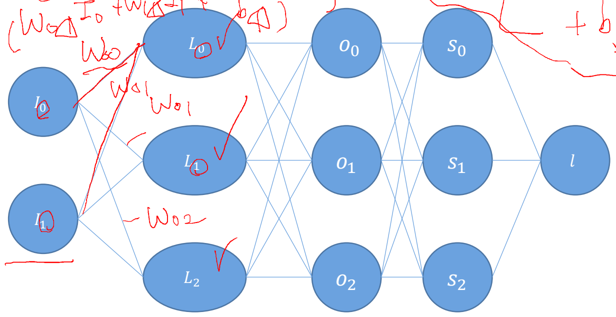
$\rightarrow L_0 = (W_{00} I_0 + W_{10} I_1 + b_0)$

$\rightarrow L_1 = ReLU(W_{01} I_0 + W_{11} I_1 + b_1)$

2 ReLU

$L_j = [(W_{0j} I_0 + W_{1j} I_1) + b_j]$

vectorization



5

$$(A \cdot B)_j = \sum_{i}^{2} a_i b_{ij} \qquad W^{(0)} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} \Leftarrow (I \cdot W^{(0)})_j = \sum_{i=0}^{2} I_i \cdot w_{ij}^{(0)}$$

$$I = [I_0 \quad I_1 \quad I_2]$$

$$L_0 = ReLU(\sum_{i=0}^{2} w_{i0}^{(0)} \cdot I_i + B_0) \qquad L_0 = \sum_{i=0}^{2} I_i \cdot w_{i0}^{(0)}$$

$$L_1 = ReLU(\sum_{i=0}^{2} w_{i1}^{(0)} \cdot I_i + B_2) \Rightarrow L_1 = \sum_{i=0}^{2} I_i \cdot w_{i1}^{(0)} \Rightarrow L_j = \sum_{i=0}^{2} I_i \cdot w_{ij}^{(0)}$$

$$L_2 = ReLU(\sum_{i=0}^{2} w_{i2}^{(0)} \cdot I_i + B_2) \qquad L_2 = \sum_{i=0}^{2} I_i \cdot w_{i2}^{(0)}$$

6

$$W^{(0)} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$

$$L = [L_0 \quad L_1 \quad L_2]$$

$$B = [B_0 \quad B_1 \quad B_2]$$

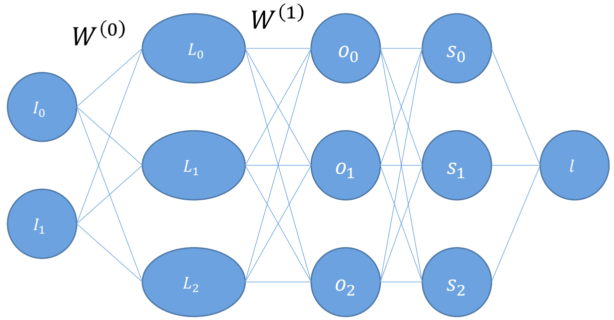
$$L_0 = ReLU(\sum_{i=0}^{2} w_{i0}^{(0)} \cdot I_i + B_0)$$

$$L_1 = ReLU(\sum_{i=0}^{2} w_{i1}^{(0)} \cdot I_i + B_2) \Rightarrow L = ReLU(I \cdot W^{(0)} + B)$$

$$L_2 = ReLU(\sum_{i=0}^{2} w_{i2}^{(0)} \cdot I_i + B_2)$$

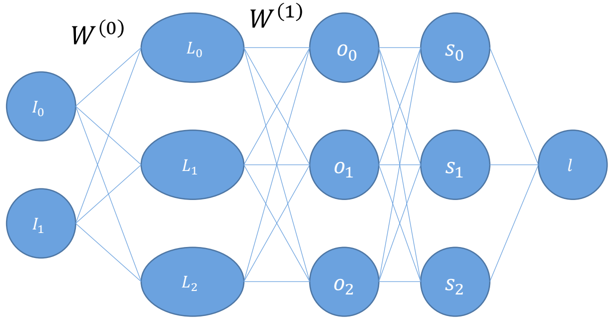
7

$$L = ReLU\left(I \cdot W^{(0)} + B\right) \qquad O = L \cdot W^{(1)}$$



8

$$L = ReLU\left(I \cdot W^{(0)} + B\right) \qquad S = softmax(O)$$
$$O = L \cdot W^{(1)} \qquad l = cross\_entropy(S)$$



9

입력 1개

입력 n개

Vectorization II

10

$$W^{(0)} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$
$$I = [I_0 \quad I_1 \quad I_2]$$
$$B = [B_0 \quad B_1 \quad B_2]$$

$$\downarrow \qquad L = ReLU\left(I \cdot W^{(0)} + B\right)$$

11

$$W^{(0)} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$
$$I = [I_0 \quad I_1 \quad I_2]$$
$$B = [B_0 \quad B_1 \quad B_2]$$

$$\downarrow \qquad L = ReLU\left(I \cdot W^{(0)} + B\right)$$

$$L = [L_0 \quad L_1 \quad L_2]$$

12

$$W^{(0)} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$

$$I = [I_0 \quad I_1 \quad I_2] \Rightarrow I = \begin{bmatrix} I_0 & I_1 & I_2 \\ I'_0 & I'_1 & I'_2 \end{bmatrix}$$

$$B = [B_0 \quad B_1 \quad B_2]$$

$$L = ReLU(I \cdot W^{(0)} + B)$$

13

$$W^{(0)} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}$$

$$I = [I_0 \quad I_1 \quad I_2] \Rightarrow I = \begin{bmatrix} I_0 & I_1 & I_2 \\ I'_0 & I'_1 & I'_2 \end{bmatrix}$$

$$B = [B_0 \quad B_1 \quad B_2]$$

$$L = ReLU(I \cdot W^{(0)} + B)$$

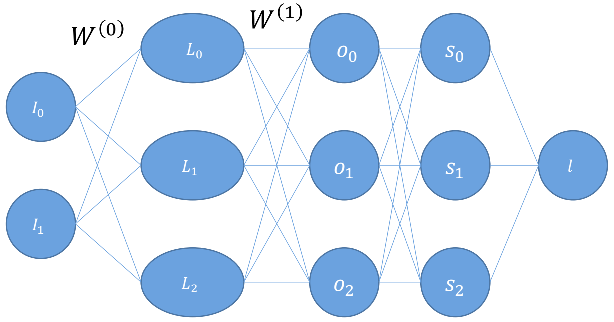
$$\Downarrow$$

$$L = \begin{bmatrix} L_0 & L_1 & L_2 \\ L'_0 & L'_1 & L'_2 \end{bmatrix}$$

14

$$L = ReLU(I \cdot W^{(0)} + B) \qquad S = softmax(O)$$

$$O = L \cdot W^{(1)} \qquad l = cross\_entropy(S)$$



15

$$L = ReLU(I \cdot W^{(0)} + B) \qquad S = softmax(O)$$

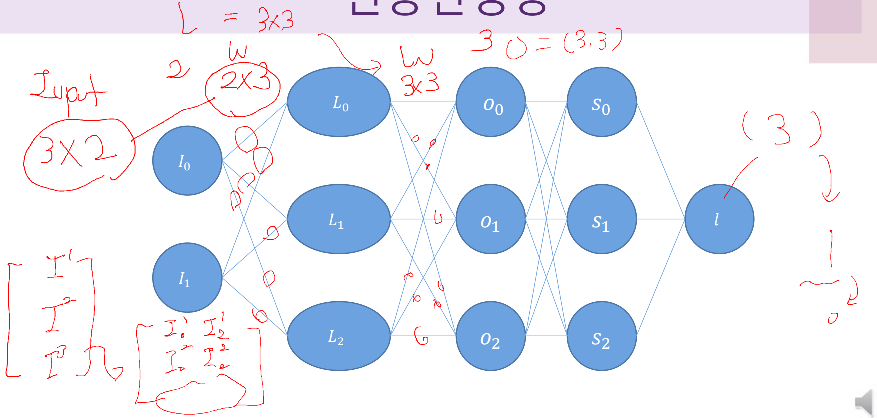
$$O = L \cdot W^{(1)} \qquad l = cross\_entropy(S)$$

$$l = [l, l']$$

$$l_f = \frac{l + l'}{2}$$

16

인공신경망



17

PyTorch의 소개

18

Pytorch

PyTorch

- 프로그램 라이브러리
- Deep Learning 의 여러 계산을 편리하게 위한 라이브러리
- Feed-forward, Back propagation, etc.

```
import torch
```

19

Pytorch의 구성요소

- torch: Tensor를 생성하는 라이브러리
- torch.autograd: 자동 미분 기능를 제공
- torch.nn: 신경망 생성
- torch.multiprocessing: 병렬처리



20

torch.Tensor

```python
import torch

x = torch.empty(3)
print(x)
```

```
tensor([0.0000e+00, 4.6566e-10, 0.0000e+00])
```

- class → Tensor
- 객체
- Attribute 과 Methods

21

tensor: method() → size()

```
>>> x
tensor([[0.0944, 0.2820, 0.5201, 0.0791],
        [0.1943, 0.8991, 0.6799, 0.3343]])
>>> x.size()
torch.Size([2, 4])
```

22

tensor: view()

```python
x = torch.randn(4, 5)
y = x.view(20)
z = x.view(5, -1)

print(x.size())
print(y.size())
print(z.size())
```

```
torch.Size([4, 5])
torch.Size([20])
torch.Size([5, 4])
```

23

tensor: indexing

```
>>> x
tensor([[0.0944, 0.2820, 0.5201, 0.0791],
        [0.1943, 0.8991, 0.6799, 0.3343]])
>>> x[0,0]
tensor(0.0944)
>>> x[1,1]
tensor(0.8991)
>>> x[0,0].item()
0.09443610906600952
```

24

tensor: indexing

```
>>> x[:,1]
tensor([0.2820, 0.8991])
>>> x[1,:]
tensor([0.1943, 0.8991, 0.6799, 0.3343])
>>> x[0:2]
tensor([[0.0944, 0.2820, 0.5201, 0.0791],
        [0.1943, 0.8991, 0.6799, 0.3343]])
>>> x[:,0:2]
tensor([[0.0944, 0.2820],
        [0.1943, 0.8991]])
```

25

Class: ‘torch.Tensor’ 연산

26

tensor + 숫자

```
>>> x
tensor([[0.0944, 0.2820, 0.5201, 0.0791],
        [0.1943, 0.8991, 0.6799, 0.3343]])
>>> x+1
tensor([[1.0944, 1.2820, 1.5201, 1.0791],
        [1.1943, 1.8991, 1.6799, 1.3343]])
>>> x*2
tensor([[0.1889, 0.5640, 1.0403, 0.1582],
        [0.3887, 1.7983, 1.3598, 0.6687]])
>>> x/2
tensor([[0.0472, 0.1410, 0.2601, 0.0396],
        [0.0972, 0.4496, 0.3400, 0.1672]])
```

27

tensor + tensor

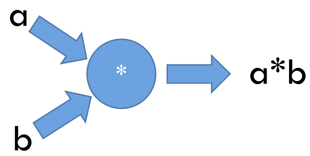
```
>>> x
tensor([[0.0944, 0.2820, 0.5201, 0.0791],
        [0.1943, 0.8991, 0.6799, 0.3343]])
>>> y
tensor([[ 0.8229, -0.4968, -0.6924, -0.2391],
        [ 0.5471,  0.7664, -0.1118, -0.8951]])
>>> x+y
tensor([[ 0.9174, -0.2148, -0.1722, -0.1600],
        [ 0.7414,  1.6655,  0.5681, -0.5607]])
>>> x*y
tensor([[ 0.0777, -0.1401, -0.3601, -0.0189],
        [ 0.1063,  0.6891, -0.0760, -0.2992]])
```

28

AUTOGRAD

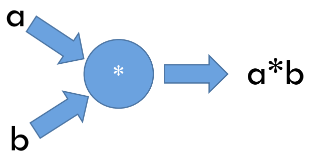
29

Why?



30

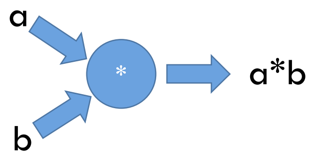
Why?



f(a, b) = a*b
a*b를 a 에 대해서 미분하면?

31

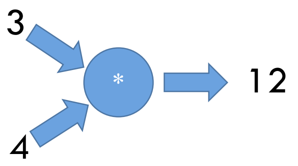
Why?



a*b를 a 에 대해서 미분하면?
b

32

Why?

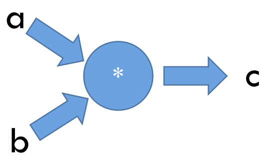


12를 4 에 대해서 미분하면?
3

autograd

```
>>> a = torch.tensor(3, dtype=torch.float, requires_grad=True)
>>> a
tensor(3., requires_grad=True)
>>> b = torch.tensor(4, dtype=torch.float, requires_grad=True)
>>> c = a*b
>>> c.backward()
>>> a.grad
tensor(4.)
>>> b.grad
tensor(3.)
```


Why?



$\frac{\partial c}{\partial a}$ → a.grad $\frac{\partial c}{\partial b}$ → b.grad


```
>>> a = torch.ones(3, requires_grad = True)
>>> b = a + 1
>>> c = b*b
>>> a.grad
>>> a.grad_fn
>>> b.grad
<input>:1: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor
>>> b.grad_fn
<AddBackward0 object at 0x000001EE4C814BC8>
>>> c.grad
<input>:1: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor
>>> c.grad_fn
<MulBackward0 object at 0x000001EE4C803F08>
>>> d = c.sum()
>>> d.backward()
>>> a.grad
tensor([4., 4., 4.])
>>> b.grad
<input>:1: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor
>>> c.grad
<input>:1: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor
>>> d.grad
<input>:1: UserWarning: The .grad attribute of a Tensor that is not a leaf Tensor
```

(1, 1, 1)
(2, 2, 2)
(4, 4, 4)
loss
a → b → c → d
a
d = 12 → scalar

torch.nn

37

$$L = ReLU(I \cdot W^{(0)} + B)$$
$$O = L \cdot W^{(1)}$$
$$S = softmax(O)$$
$$l = cross\_entropy(S, label)$$

```
w_ih = torch.rand(...)
w_ho = torch.rand(...)

L = torch.mm(input, w_ih)
L = ReLU(L)
O = torch.mm(L, w_ho)

S = softmax(O)
Loss = cross_entropy(s, label)
```

38

model

$$L = ReLU(I \cdot W^{(0)} + B)$$
$$O = L \cdot W^{(1)}$$

training

$$S = softmax(O)$$
$$l = cross\_entropy(S, label)$$

model

```
w_ih = torch.rand(...)
w_ho = torch.rand(...)

L = torch.mm(input, w_ih)
L = ReLU(L)
O = torch.mm(L, w_ho)
```

training

```
S = softmax(O)
Loss = cross_entropy(s, label)
```

39

Pytorch의 구성요소

- torch: Tensor를 생성하는 라이브러리
- torch.autograd: 자동 미분 기능를 제공
- torch.nn: 신경망 생성
- torch.multiprocessing: 병렬처리

40

Pytorch의 구성요소

- torch: Tensor를 생성하는 라이브러리
- torch.autograd: 자동 미분 기능를 제공
- **torch.nn: 신경망 생성**
- torch.multiprocessing: 병렬처리

41

Pytorch

torch.nn

- CLASS 로 인공신경망의 모델 구현을 위한 라이브러리
- FC Layer, CNN, RNN, Transformers 등의 구조가 미리 입력

42

PyTorch.nn의 응용 – model

43

model

$$L = ReLU(I \cdot W^{(0)} + B)$$
$$O = L \cdot W^{(1)}$$

model

MODULE

CLASS torch.nn.Module [SOURCE]

Base class for all neural network modules.

Your models should also subclass this class.

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()



    def forward(self,x):



        return x
```

44

model

MODULE

CLASS `torch.nn.Module` [SOURCE]

Base class for all neural network modules.

Your models should also subclass this class.

`train(mode=True)` [SOURCE]

Sets the module in training mode.

`eval()` [SOURCE]

Sets the module in evaluation mode.

`cpu()` [SOURCE]

Moves all model parameters and buffers to the CPU.

`cuda(device=None)` [SOURCE]

Moves all model parameters and buffers to the ~~GPU~~.

45

model

w_ih = torch.rand(784,100)
w_ho = torch.rand(100,10)

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(784, 100)
        self.fc2 = nn.Linear(100, 10)
        elf.relu = nn.ReLU()

    orward(self,x):

    eturn x
```

model

LINEAR

CLASS `torch.nn.Linear(in_features, out_features, bias=True, device=None, dtype=None)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

46

model

w_ih = torch.rand(784,100)
w_ho = torch.rand(100,10)

model

LINEAR

CLASS `torch.nn.Linear(in_features, out_features, bias=True,` `dtype=None)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T +$

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(784, 100)
        self.fc2 = nn.Linear(100, 10)
        self.relu = nn.ReLU()

    def forward(self,x):


        return x
```

47

model

L = torch.mm(input, w_ih)
L = ReLU(L)
O = torch.mm(L, w_ho)

model

RELU

CLASS `torch.nn.ReLU(inplace=False)` [SOURCE]

Applies the rectified linear unit function element-wise:

$$\mathrm{ReLU}(x) = (x)^+ = \max(0, x)$$

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.fc1 = nn.Linear(784, 100)
        self.fc2 = nn.Linear(100, 10)
        self.relu = nn.ReLU()

    def forward(self,x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x
```

48

PyTorch.nn의 응용 – training

49

training

S = softmax(O)
Loss = cross_entropy(s, label)

```
cls_loss = nn.CrossEntropyLoss()
loss = cls_loss(outputs, label)
```

training

CROSSENTROPYLOSS

CLASS `torch.nn.CrossEntropyLoss(weight=None, size_average=None, ignore_index=- 100, reduce=None, reduction='mean', label_smoothing=0.0)` [SOURCE]

This criterion computes the cross entropy loss between input and target.

!! 주의 !!
softmax + cross_entropy
+ one_hot

50

training

Loss.backward()
W = W – lr * W.grad

training

CLASS `torch.optim.Optimizer(params, defaults)` [SOURCE]

Base class for all optimizers.

```
learning_rate = 0.01
optimizer = optim.SGD(network.parameters(), lr=learning_rate)

optimizer.zero_grad()
loss.backward()
optimizer.step()
```

51

① 인공뉴런 → 망 → O—O → 학습 loss
backpropagation
GD
vector [ ]
→ SGD
→ initialized W
→
vector [ ]
② pytorch → Model { nn.module
__init__
forward(_x) } → loss
loss.backward() optimizer.step()

52

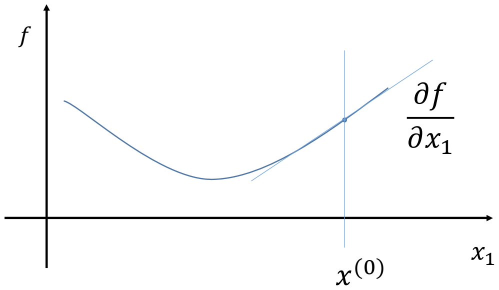
목차

53

Gradient Descent & Vectorization II

54

Gradient Descent



55

Gradient Descent

$$f(X)$$

$f(X)$의 값을 줄이기 위한 X의 방향은?

$$X \leftarrow X \ - \eta \, \nabla_X f$$

$$X = [x_1, x_2, x_3, x_4]$$

$$\nabla X = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial x_4}\right]$$

$$x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1}$$

56

Gradient Descent

$$loss(x; W)$$

$loss$의 값을 줄이기 위한 W의 방향은?

$$W \leftarrow W - \eta \nabla_W loss$$

$$W = \left[w_{11}^{(0)}, w_{12}^{(0)}, \dots, w_{nm}^{(l)}\right]$$

$$w_{11}^{(0)} \leftarrow w_{11}^{(0)} - \eta \frac{\partial f}{\partial w_{11}^{(0)}}$$

$$x?$$

57

Gradient Descent

$$loss(x; W)$$

$loss$의 값을 줄이기 위한 W의 방향은?

$$W \leftarrow W - \eta \nabla_W loss$$

$$W = \left[w_{11}^{(0)}, w_{12}^{(0)}, \dots, w_{nm}^{(l)}\right]$$

$$w_{11}^{(0)} \leftarrow w_{11}^{(0)} - \eta \frac{\partial f}{\partial w_{11}^{(0)}}$$

$$x \leftarrow x_1, x_2, \dots, x_m$$

58

Stochastic Gradient Descent

59

Gradient Descent

$$loss(x; W)$$

$loss$의 값을 줄이기 위한 W의 방향은?

$$W \leftarrow W - \eta \nabla_W W$$

$$W = \left[w_{11}^{(0)}, w_{12}^{(0)}, \dots, w_{nm}^{(l)}\right]$$

$$w_{11}^{(0)} \leftarrow w_{11}^{(0)} - \eta \frac{\partial f}{\partial w_{11}^{(0)}}$$

$$x \leftarrow x_1, x_2, \dots, x_m$$

60

Gradient Descent

$$loss(x; W)$$

$$x \leftarrow x_1, x_2, \ldots, x_{100}$$

$loss$ 계산 후

$$w_{11}^{(0)} \leftarrow w_{11}^{(0)} - \eta \frac{\partial f}{\partial w_{11}^{(0)}}$$

$$x \leftarrow x_{101}, x_{102}, \ldots, x_{200}$$

61

Gradient Descent

$$loss(x; W)$$

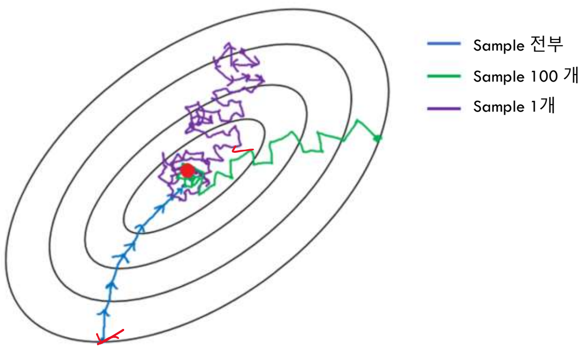
$$x \leftarrow x_{101}, x_{102}, \ldots, x_{200}$$

$loss$ 계산 후

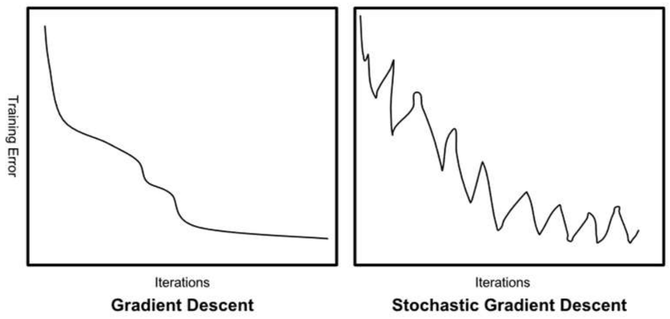
$$w_{11}^{(0)} \leftarrow w_{11}^{(0)} - \eta \frac{\partial f}{\partial w_{11}^{(0)}}$$

$$x \leftarrow x_{201}, x_{202}, \ldots, x_{300}$$

62



63



64