AI School 6기 2주차

파이썬 기초2

딥러닝 기초 이론2

MLP를 이용한 필기체 인식기 개발

AI School 6기 2주차

파이썬 기초2

파이썬의 자료형 – 딕셔너리

```
# dictionary 사용법
name_to_age = {"Jenny": 20, "Ella":31}
name_to_age["John"] = 26
name_to_age["Tom"] = 29
print(name_to_age["Jenny"])
print(name_to_age["John"])
print(name_to_age["Tom"])

name_to_age["Jenny"] = 21
print(name_to_age["Jenny"])
print(name_to_age.get("Jenny"))
print(name_to_age.keys())

for name in name_to_age.keys():
    print(name, name_to_age[name])

for i, name in enumerate(name_to_age.keys()):
    print(i, name, name_to_age[name])

print("Andrew" in name_to_age)
print("Ella" in name_to_age)
```

파이썬의 자료형 – 집합

```
s1 = set("Hello")
print(s1)

s2 = set([1, 1, 2, 2, 3, 4, 5])
print(s2)
```

파이썬 함수

```
# function 사용법
def sum(a, b):
    s = a + b
    return s



print(sum(3, 5))
print(sum(2, 1))



def sum_and_mul(a, b):
    return a + b, a*b

s, m = sum_and_mul(3,5)
print(s)
print(m)
print(sum_and_mul(3, 5))
print(sum_and_mul(2, 1))
```

파이썬의 제어문 – if

```
man = True

if man:
    print("남자화장실로 가세요")
else:
    print("여자 화장실로 가세요")
```

```
minimum = 165
height = 163

if height < minimum:
    print("탑승하실수 없습니다")
else:
    print("탑승하세요")
```

파이썬의 제어문 – if

```
minimum = 165
height = 163

if height <= minimum:
    print("탑승하실수 없습니다")
else:
    print("탑승하세요")
```

```
blood_type = "A"
emergency_patient = "A"

if blood_type == emergency_patient:
    print("수혈해 주세요")
else:
    print("수혈해 주실 수 없습니다")
```

파이썬의 제어문 – if

```
minimum = 165
maximum = 195
height = 200

if height < minimum or height > maximum:
    print("탑승하실 수 없습니다")
else:
    print("탑승하세요")
```

```
blood_type1 = "A"
emergency_patient_type1 = "A"
blood_type2 = "RH+"
emergency_patient_type2 = "RH+"

if blood_type1 == emergency_patient_type1 and blood_type2 ==
emergency_patient_type2:
    print("수혈해 주세요")
else:
    print("수혈해 주실 수 없습니다")
```

파이썬의 제어문 – if

```
basic = 40
intermediate = 70
advanced = 100
score = 110
if score <= basic:
    print("초급반을 수강하세요")
elif score <= intermediate:
    print("중급반을 수강하세요")
elif score <= advanced:
    print("고급반을 수강하세요")
else:
    print("점수를 확인해주세요")
```

연습문제– if

- 홀수, 짝수를 구분하는 코드를 작성하세요.

파이썬의 제어문 – for

```
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number +1
    if mark >= 60:
        print("%d번 학생은 합격입니다." % number)
    else:
        print("%d번 학생은 불합격입니다." % number)
```

```
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number +1
    if mark < 60:
        continue
    print("%d번 학생 축하합니다. 합격입니다. " % number)
```

파이썬의 제어문 – for

```
for i in range(10):
    print(i)
```

```
sum = 0
for i in range(1,11):
    sum += i
print(sum)
```

```
for i in range(2,10):
    for j in range(1, 10):
        print(i*j, end=" ")
    print('')
```

파이썬의 자료형 – 숙제 1

```
student2score = {
    "Darius": 100,
    "Dr. Mundo": 80,
    "Morgana": 60,
    "Sivir": 75,
    "Yummi": 20,
    "Viktor": 97
}

def get_special_students(student2score):
    """
    특별반 학생의 리스트를 리턴하는 함수
    특별반은 점수가 80점 이상이어야 들어갈 수 있다.
    :param student2score:
    :return special_students:
    """
    special_students = []
    return special_students
```

파이썬의 자료형 – 숙제 2

```
text = "Apple is fruit. Orange is also fruit. Tomato is fruit?"

def word_index_count(text):
    """
    텍스트 안에 단어에 id를 부여하고 각 단어의 빈도수를 세어
    각각의 텍스트에 해당하는 id를 저장하는 딕셔너리와
    각각의 단어 id에 해당하는 빈도수를 저장하는 딕셔너리 리턴

    텍스트 안에 특수 기호는 제거해야한다
    모든 단어는 소문자 형태로 관리한다
    """
    word_id = {}
    id_frequency = {}

    return word_id, id_frequency

print(word_index_count(text))
```

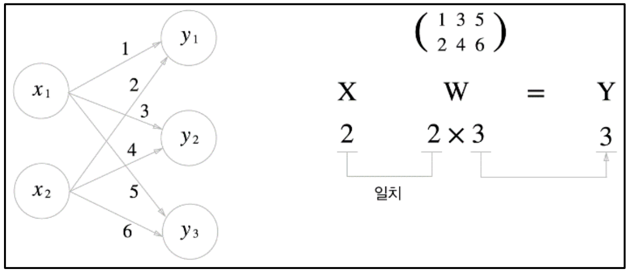
파이썬의 자료형 – 숙제 3

```
    *
   ***
  *****
 *******
*********
```

AI School 6기 2주차

딥러닝 기초 이론2

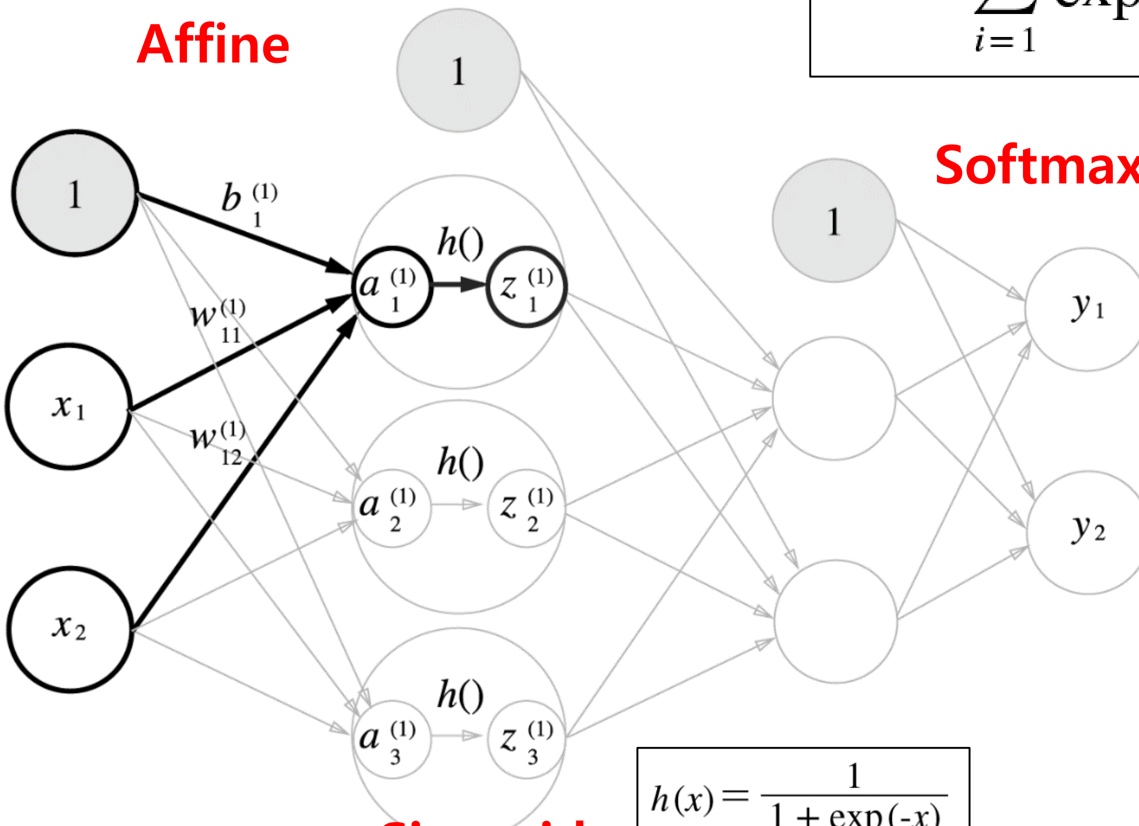
Feedforward



Affine

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

Softmax



Loss function

$$E = -\sum_k t_k \log y_k$$

$$h(x) = \frac{1}{1 + \exp(-x)}$$

Sigmoid

ReLU

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

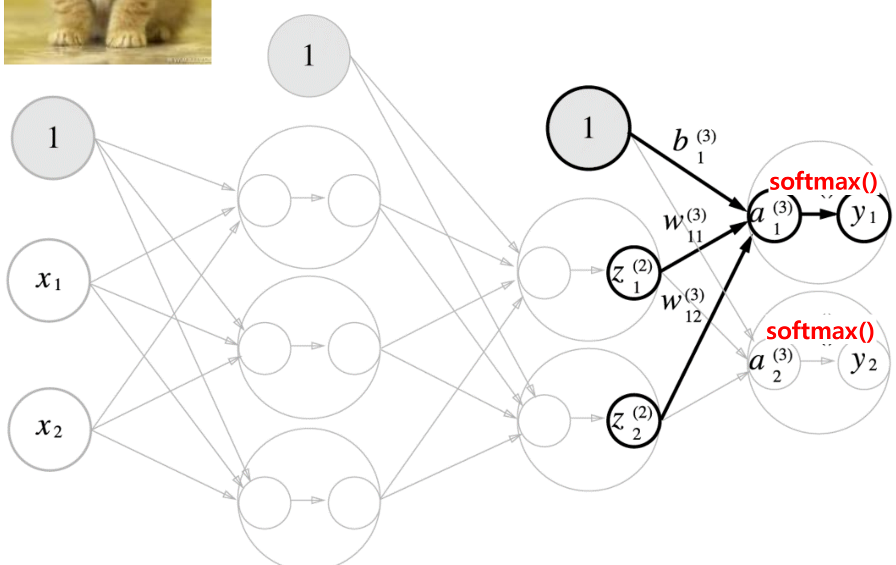
Loss Function (Error Function)

- Mean squared error (평균 제곱 오차)

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

```
def mean_squared_error(y, t):
    return 0.5 * np.sum((y-t)**2)
```

y = [**0.1**, 0.9]
t = [**1.0**, 0.0]



Cat: 0.1

Dog: 0.9

error

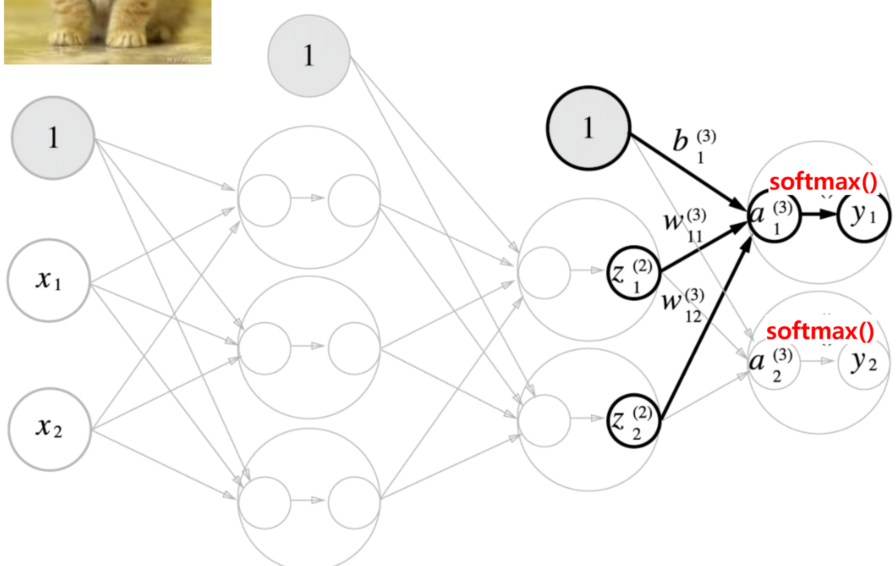
$$\boxed{\begin{array}{c}(0.1 - 1)^2 \\ \\ (0.9 - 0)^2\end{array}}$$

Loss Function (Error Function)

- Cross entropy error (교차 엔트로피 오차)
- 정답일 때의 출력이 전체 값을 결정

$$E = -\sum_k t_k \log y_k$$

y = [**0.1**, 0.9]
t = [**1.0**, 0.0]



Cat: 0.1

Dog: 0.9

error

$$\boxed{\begin{array}{l} 1 * \log 0.1 \\ 0 * \log 0.9 \end{array}}$$

Back-Propagation[1][2]

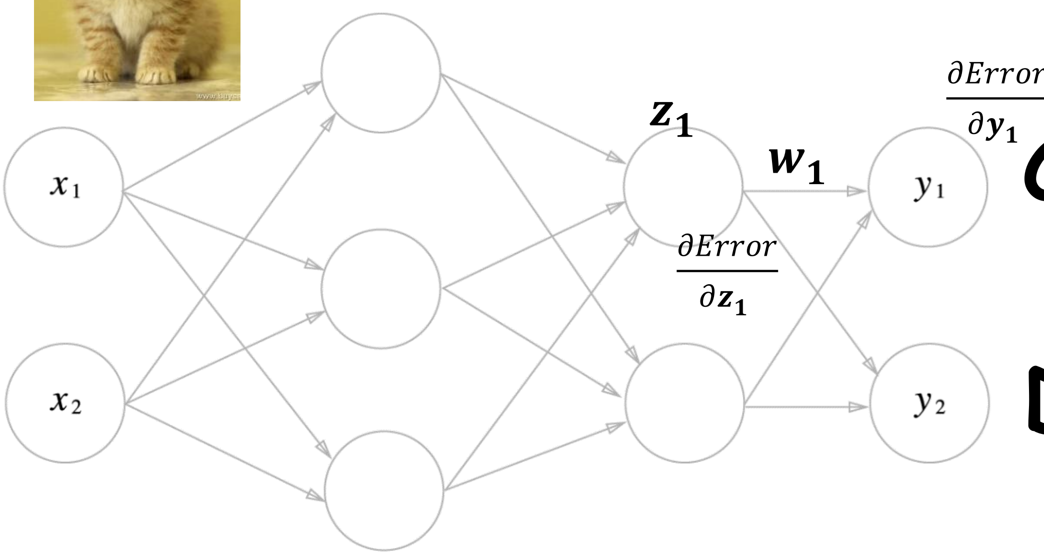
- Training : back propagation of error
  - ✓ Calculate total error at the top
  - ✓ The error propagates via chain rule

Minimize error $= \sum_{l=1}^{m} (y^{(l)} - t^{(l)})^2$

guess : $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$

Teacher's solutions: $t^{(1)}, t^{(2)}, \ldots, t^{(m)}$

problem : $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$



Cat: 0.1

Dog: 0.9

error

$(0.1 - 1)^2$

$(0.9 - 0)^2$

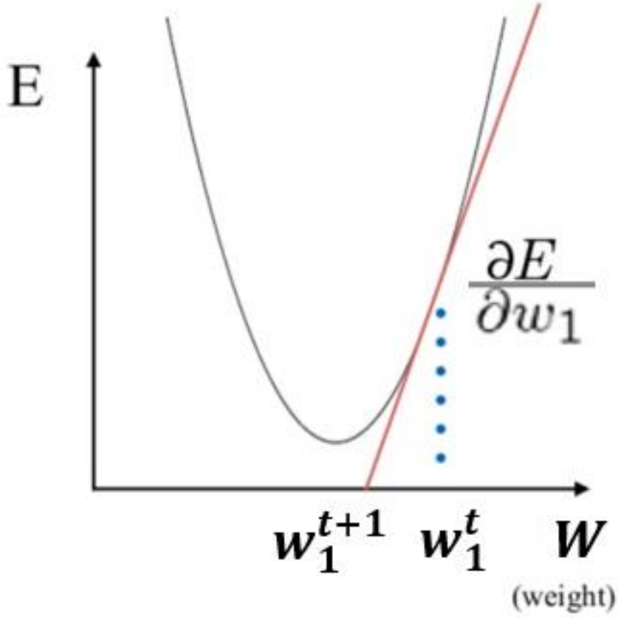
[1] Rumelhart, D., Hinton, G., and Williams, R., Learning representations by back-propagating errors, Nature 1986
[2] Rumelhart, D., McClelland, J. and PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press 1986

Gradient decent

- 임의의 한 지점으로부터 시작해, loss가 줄어드는 방향으로 parameter들을 갱신한다. (loss가 가장 적어질 때까지)
- W를 음의 기울기 방향 (-∇E)으로 조금씩 움직이는 것을 여러 번 반복

$$w_1^{t+1} = w_1^{t} - \varepsilon \triangledown E$$

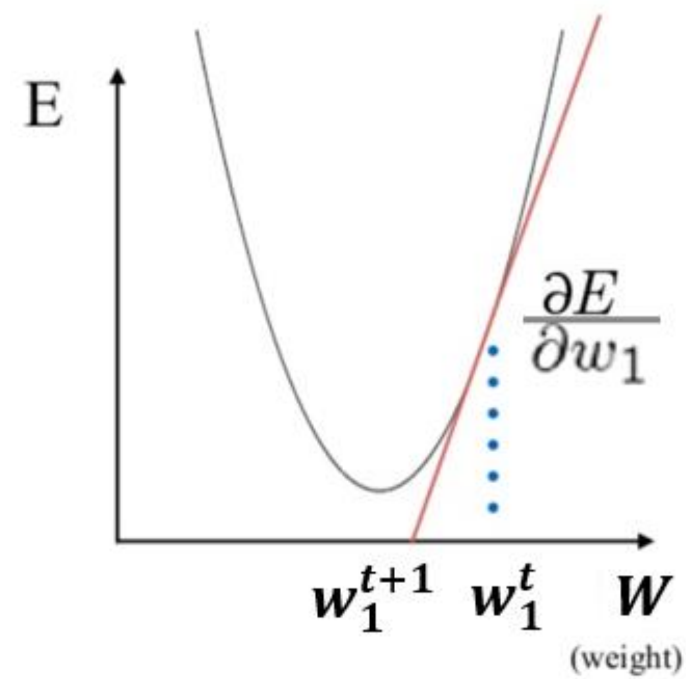
여기서 E = 모든 훈련 샘플에 대하여 계산되는 오차



Gradient decent

- Loss가 줄어드는 방향으로 얼만큼 움직여야 하는가?
  - Learning rate
- Loss function의 미분값*learning rate 만큼 parameter 수정
- **Learning rate가 작으면** 최적점을 찾는 과정이 매우 더디다
- **Learning rate이 크면**?

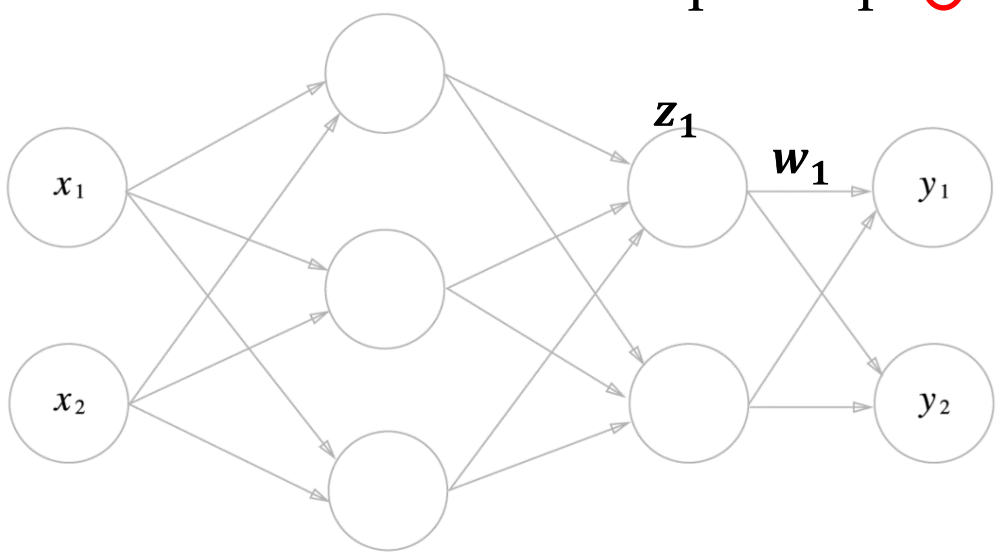
$$w_1^{t+1} = w_1^t - \varepsilon \nabla E$$

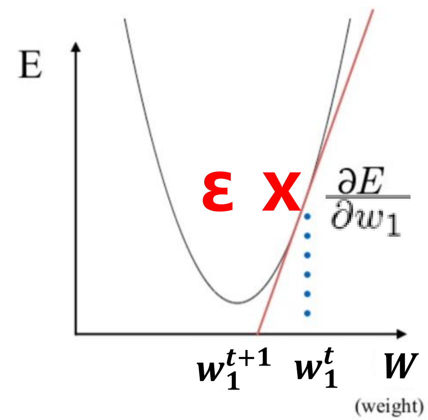


Learning Rate

Learning rate

$$w_1^{t+1} = w_1^t - \varepsilon \nabla E$$



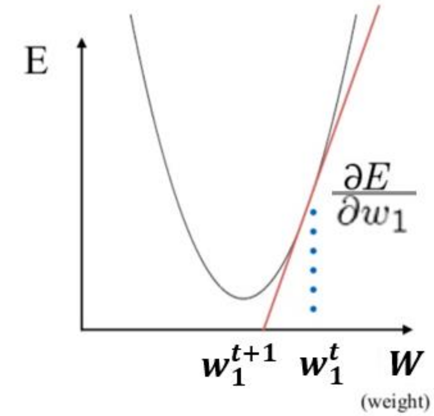


$\varepsilon = \varepsilon_0 / \alpha t$

* word2vec.c

$\varepsilon = \varepsilon_0 / (1 - t/Nt)$

Back-Propagation[1][2]

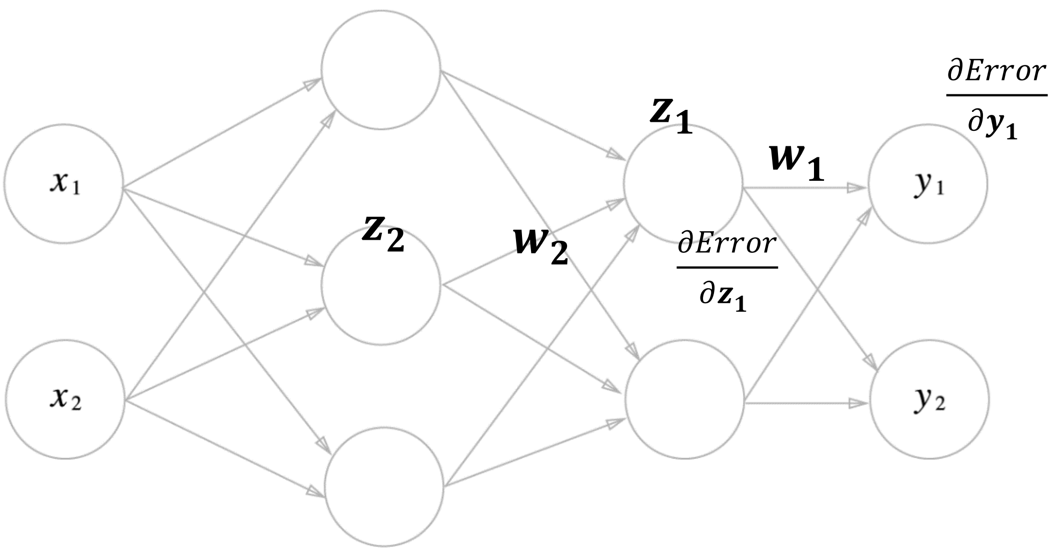


- Training : back propagation of error
  - ✓ Calculate total error at the top
  - ✓ The error propagates via chain rule

Minimize error $= \sum_{l=1}^{m} (y^{(l)} - t^{(l)})^2$

guess : $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$ Teacher's solutions: $t^{(1)}, t^{(2)}, \ldots, t^{(m)}$

problem : $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$



$$\frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial y_1}\frac{\partial y_1}{\partial w_1} = \frac{\partial Error}{\partial y_1}\,\boldsymbol{\sigma}'(\cdot)\ \boldsymbol{z_1}$$

$$* \ \frac{\partial y_1}{\partial w_1} = \boldsymbol{\sigma}'(\cdot)\ \boldsymbol{z_1}$$

$$\frac{\partial Error}{\partial w_2} = \frac{\partial Error}{\partial z_1}\frac{\partial z_1}{\partial w_2} = \frac{\partial Error}{\partial y_1}\,\boldsymbol{\sigma}'(\cdot)\ \boldsymbol{w_1}\ \boldsymbol{\sigma}'(\cdot)\ \boldsymbol{z_2}$$

$$* \ \frac{\partial z_1}{\partial w_2} = \boldsymbol{\sigma}'(\cdot)\ \boldsymbol{z_2}$$

[1] Rumelhart, D., Hinton, G., and Williams, R., Learning representations by back-propagating errors, Nature 1986
[2] Rumelhart, D., McClelland, J. and PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press 1986

Derivative

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$f(x) = 3$$

$$f(x) = x$$

$$f(x) = 2x$$

Derivative

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

$f(x) = 2x$

$f(x, y) = xy, \frac{\partial f}{\partial x}$

$f(x, y) = xy, \frac{\partial f}{\partial y}$

Derivative

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$f(x) = 3$

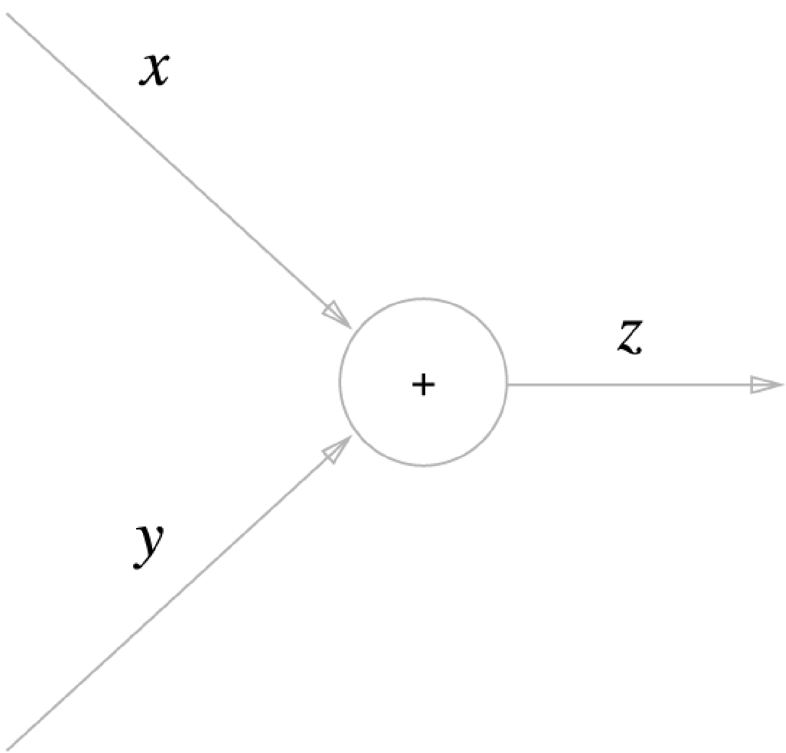
$f(x) = 2x \qquad f(x) = x + x$

$f(x) = x + 3$

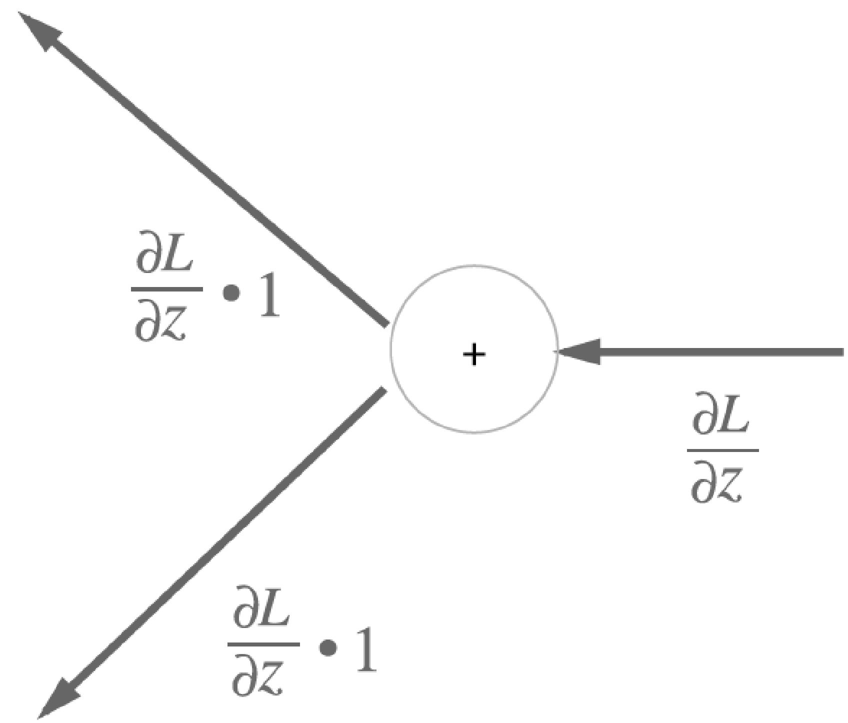
$f(x, y) = x + y, \frac{\partial f}{\partial x}$

$f(x, y) = x + y, \frac{\partial f}{\partial y}$

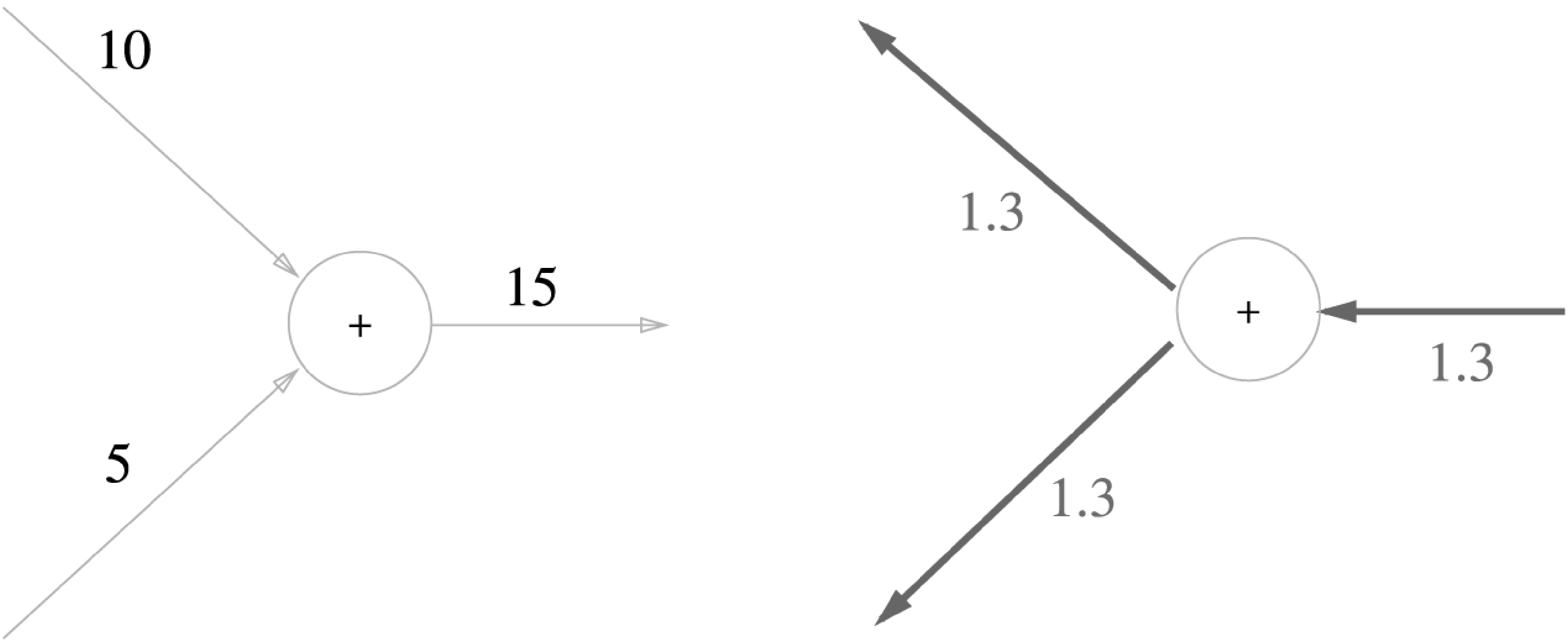
Back-Propagation



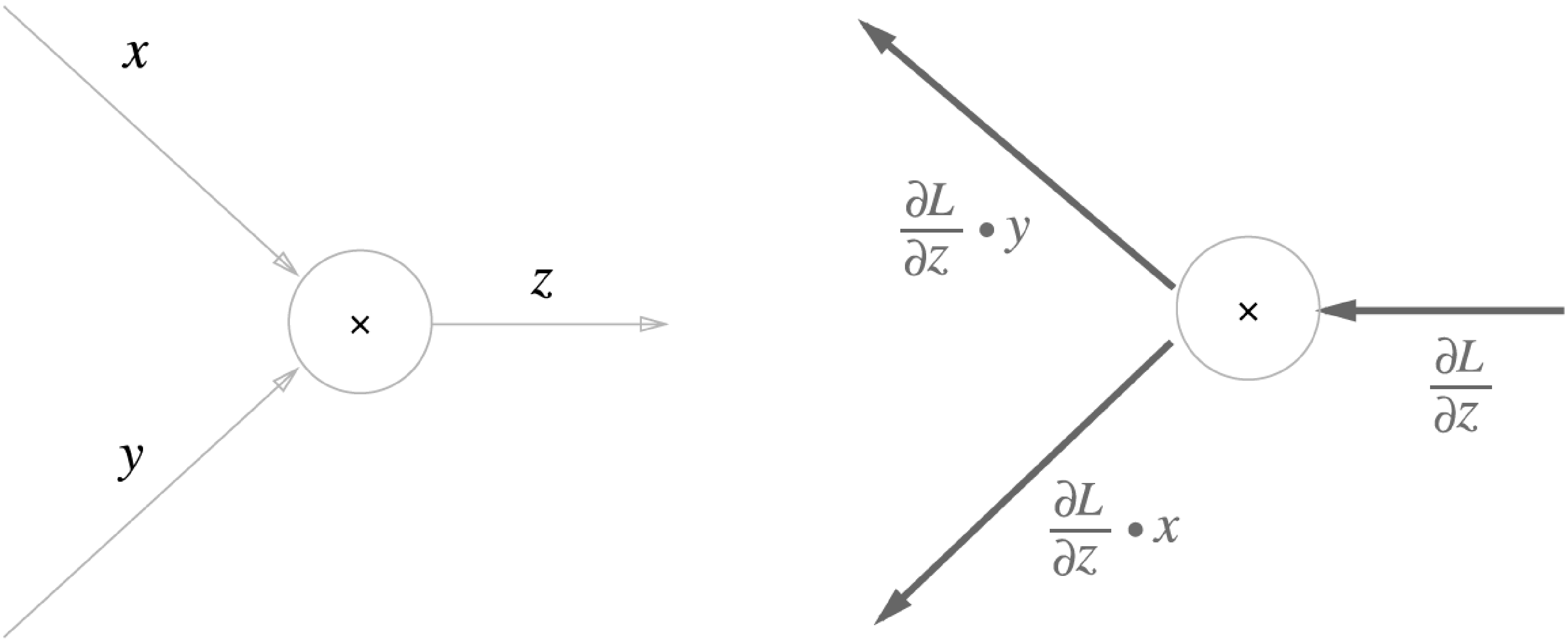
$z = x + y$



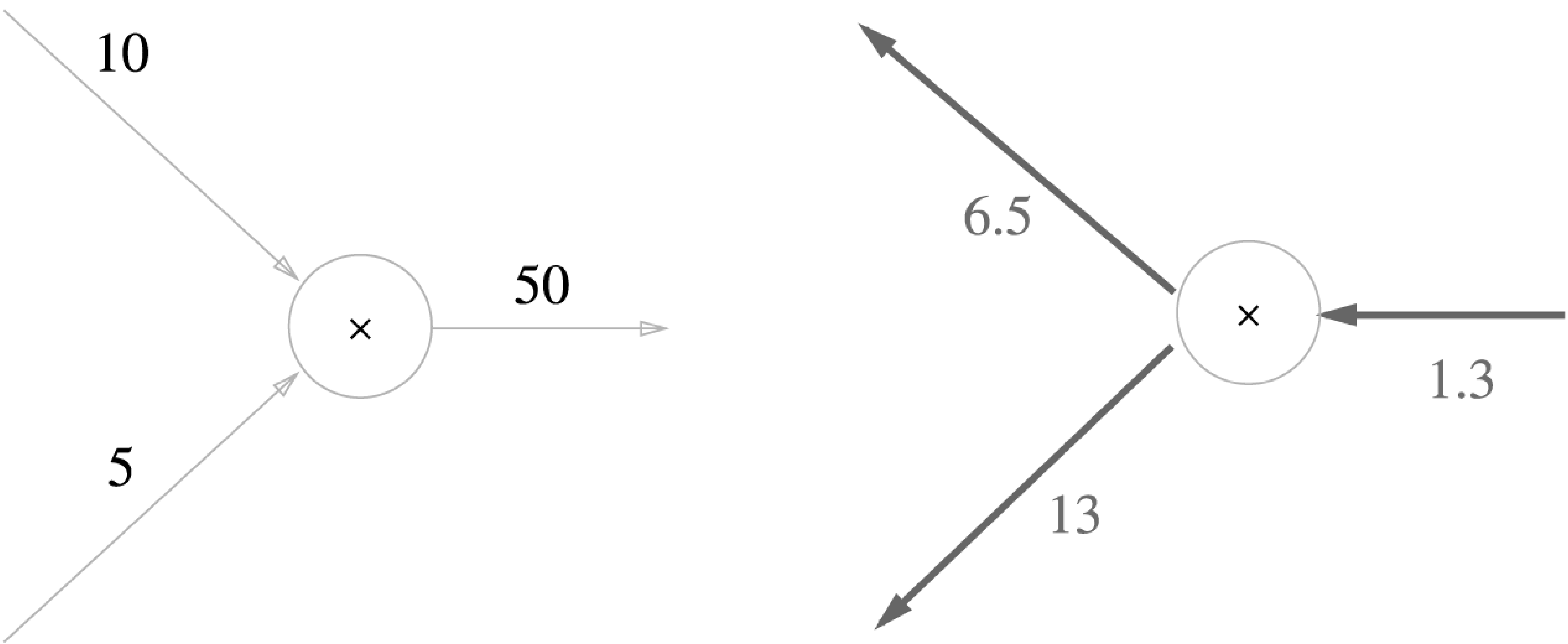
Back-Propagation



Back-Propagation



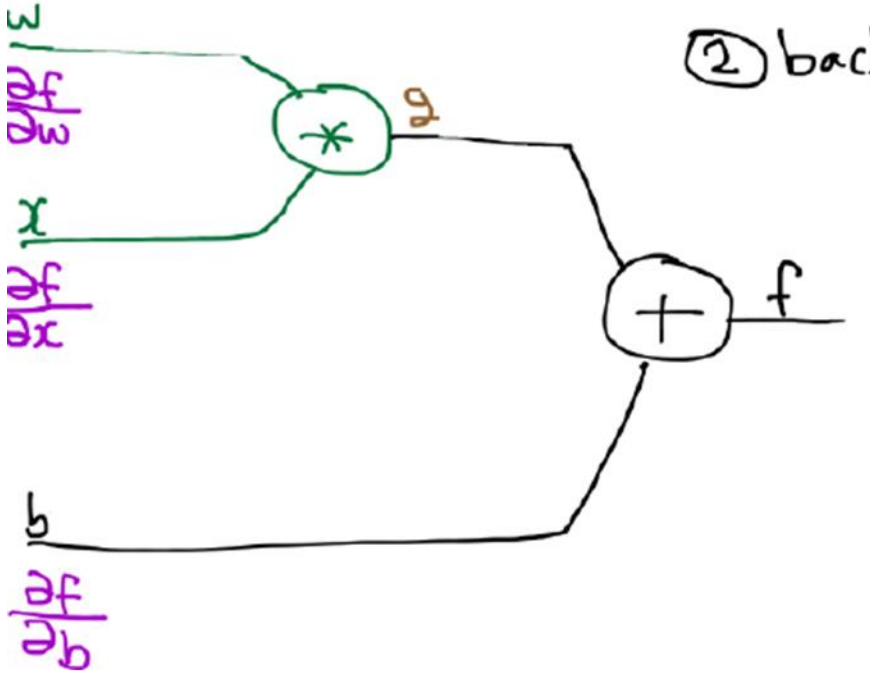
Back-Propagation



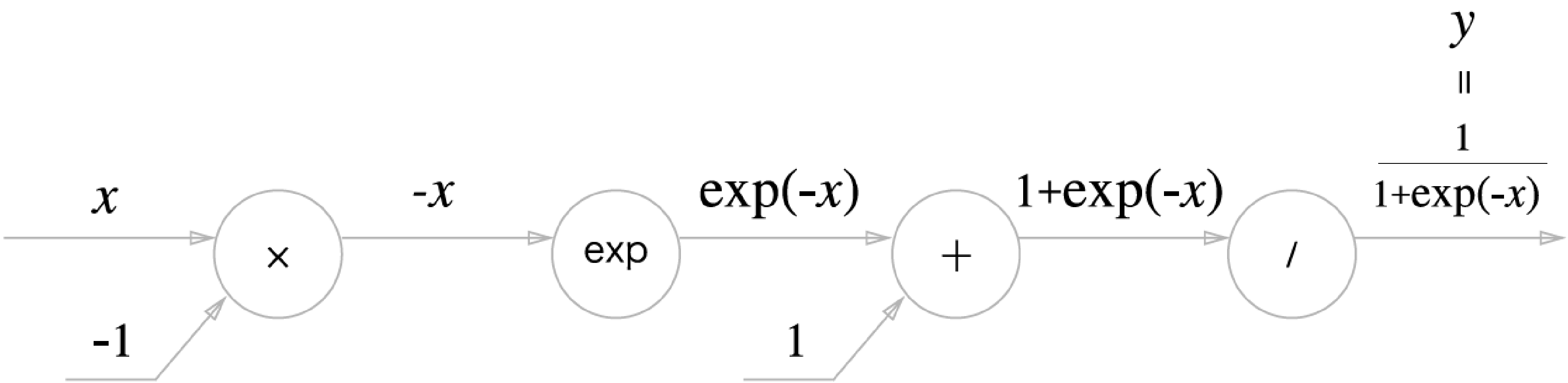
Back-Propagation

$$f = wx + b, \quad g = wx, \quad f = g + b$$

1. forward ($w = -2$, $x = 5$, $b = 3$)
2. backward



Back-Propagation (sigmoid)



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

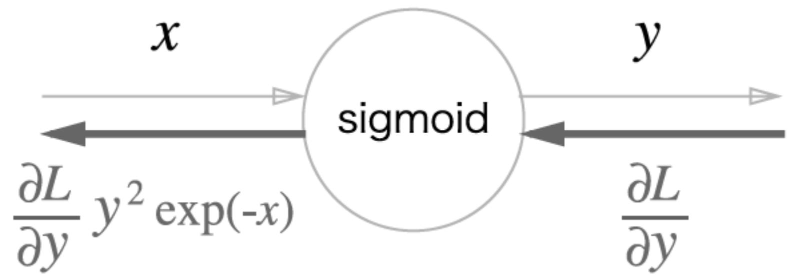
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

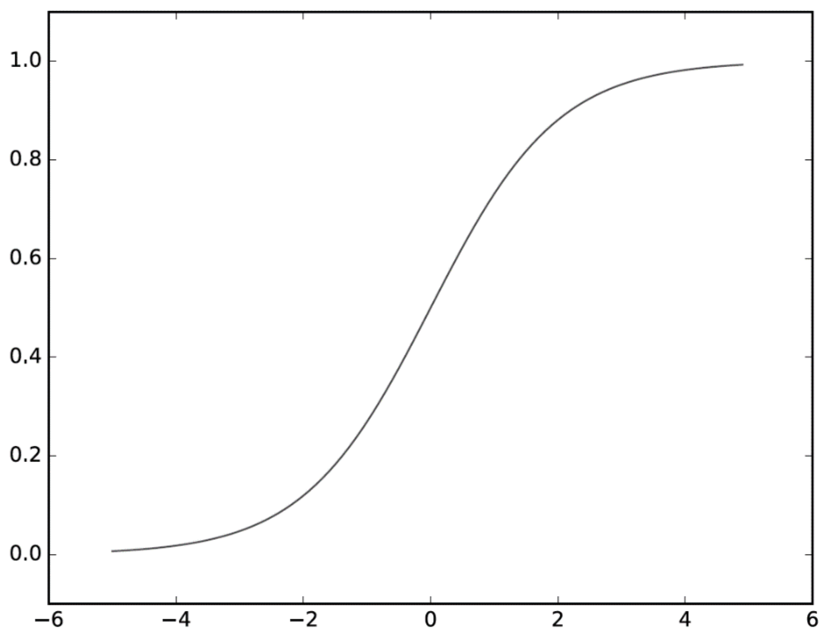
Back-Propagation (sigmoid)

$$\frac{\partial L}{\partial y} y^2 \exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1+\exp(-x))^2} \exp(-x)$$

$$= \frac{\partial L}{\partial y} \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)}$$

$$\boxed{= \frac{\partial L}{\partial y} y(1-y)}$$

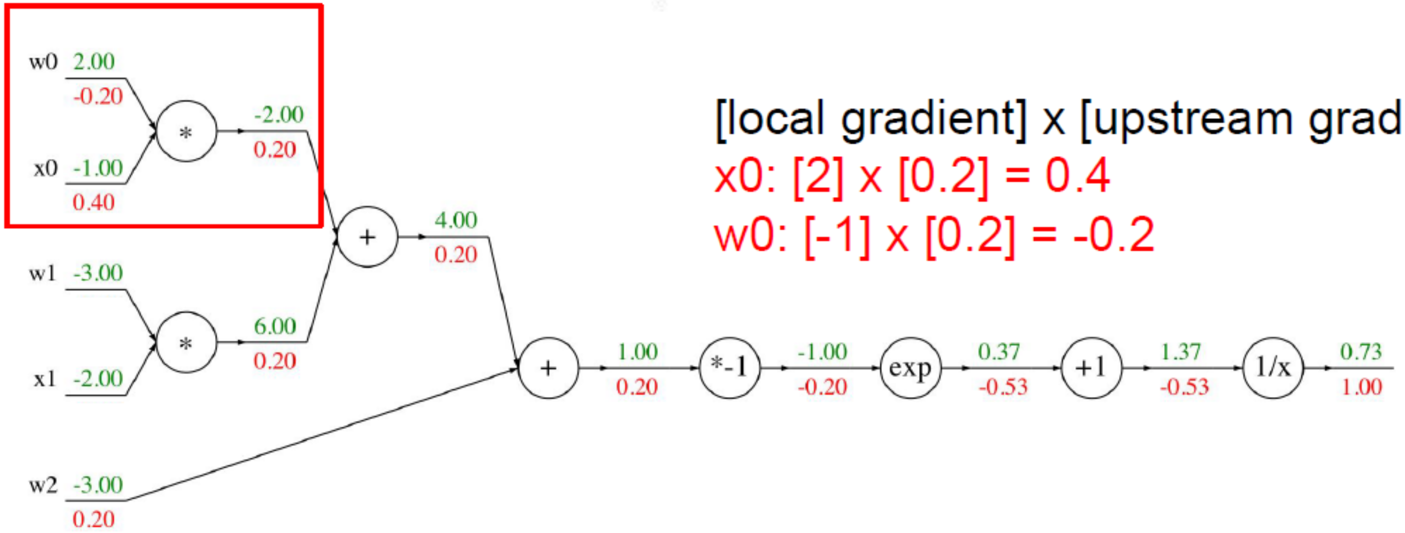





```
def __init__(self):
    self.out = None

def forward(self, x):
    out = sigmoid(x)
    self.out = out
    return out

def backward(self, dout):
    dx = dout * (1.0 - self.out) * self.out
    return dx
```

Back-Propagation

Another example: $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[local gradient] x [upstream gradient]

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2

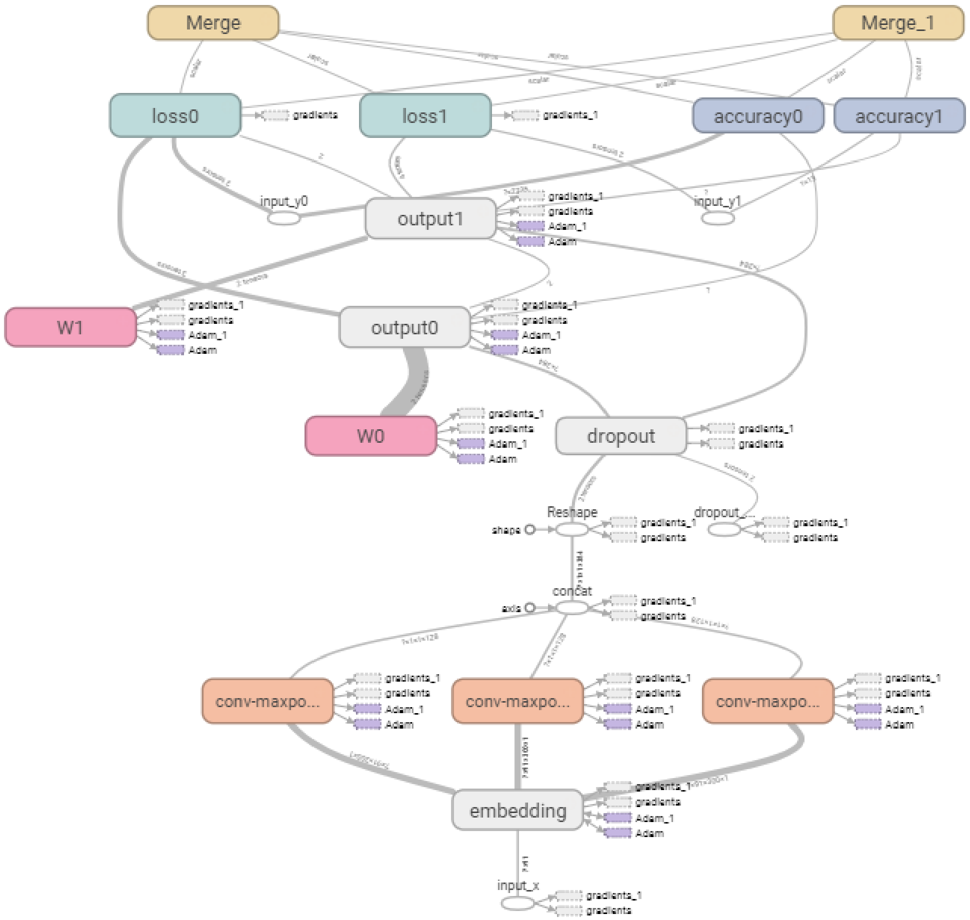
$f(x) = e^x \quad \rightarrow \quad \dfrac{df}{dx} = e^x$

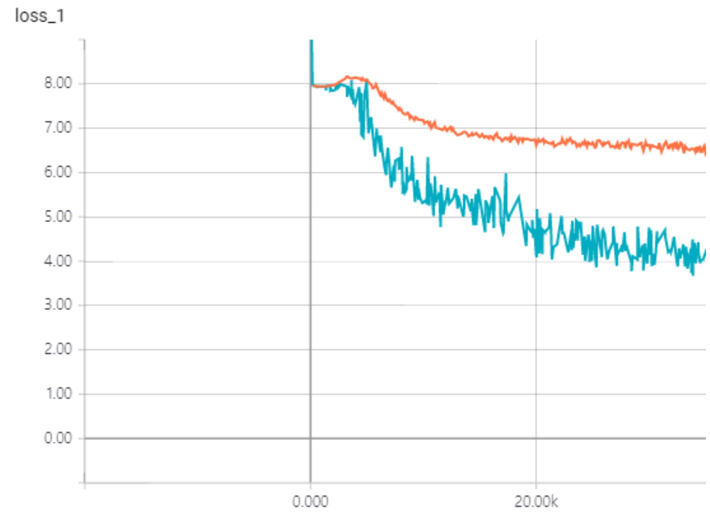
$f_a(x) = ax \quad \rightarrow \quad \dfrac{df}{dx} = a$

$f(x) = \dfrac{1}{x} \quad \rightarrow \quad \dfrac{df}{dx} = -1/x^2$

$f_c(x) = c + x \quad \rightarrow \quad \dfrac{df}{dx} = 1$

Tensor board



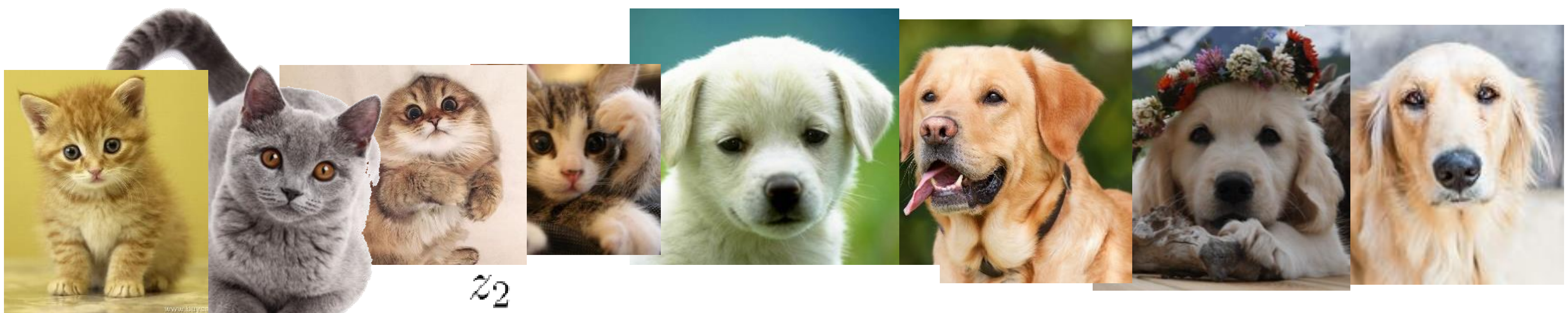


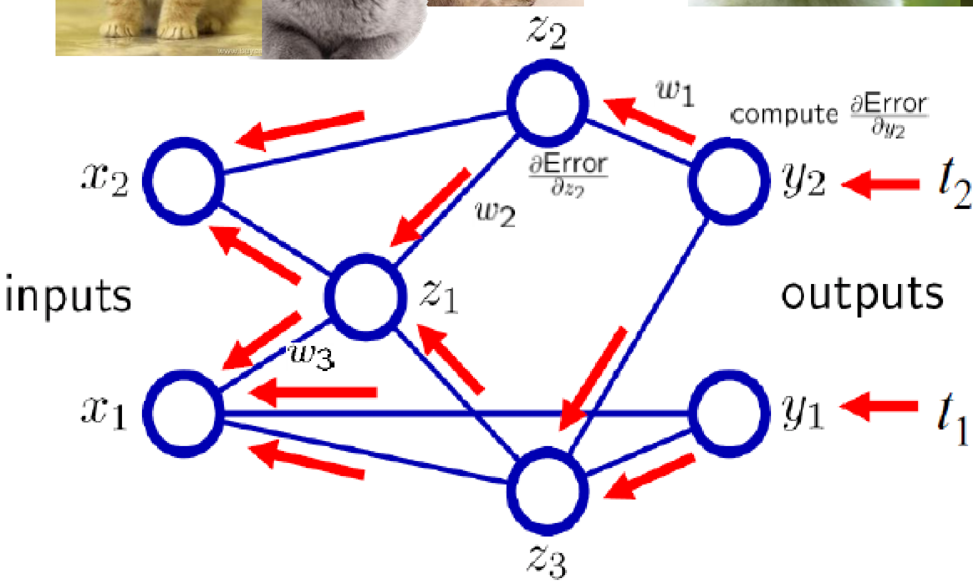
https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/g3doc/how_tos/summaries_and_tensorboard/

Batch Learning (Epoch Learning)

- **전체 훈련 데이터**를 사용
- 대규모 데이터 셋에 적용하기 힘듬
- 데이터 구성이 항상 같기 때문에 local minima에 빠질 위험이 있음

$$w_1^{t+1} = w_1^{t} - \varepsilon \nabla E \qquad E = \sum_{l=1}^{m} (y^{(l)} - t^{(l)})^2$$



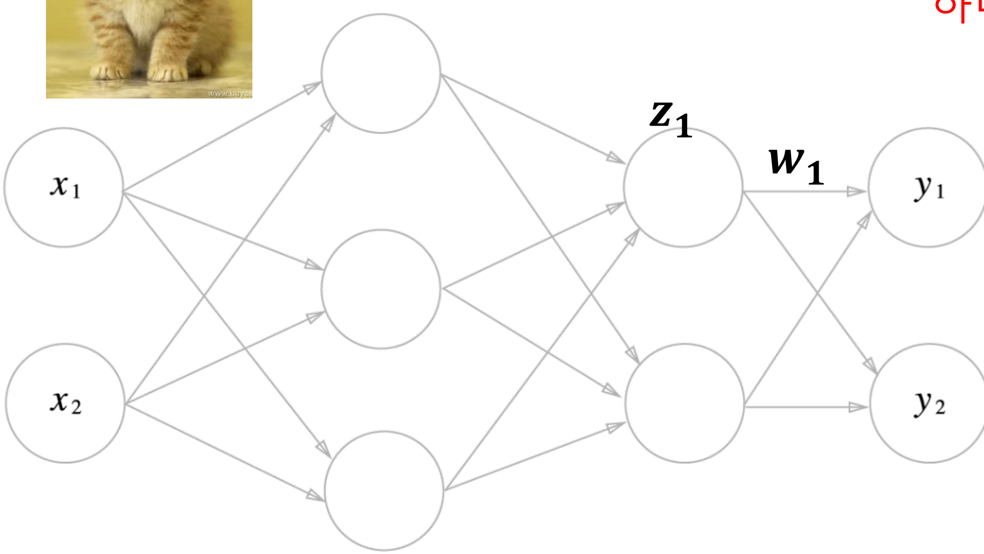


Stochastic Gradient decent (SGD)

- Batch learning과는 달리 **샘플 일부**만을 사용하여 파라미터를 업데이트하는 방법
- 빠른 학습 가능, local minima에 빠질 위험이 적음
- 노이즈 데이터로 인해 변동이 큼
- Epoch 마다 무작위로 샘플을 선택하면 그 효과를 극대화 할 수 있고 때문에 **Stochastic** Gradient decent라 부름

$$w_1^{t+1} = w_1^{t} - \varepsilon \nabla E_n$$

하나의 샘플n에 대해 계산한 오차함수의 기울기

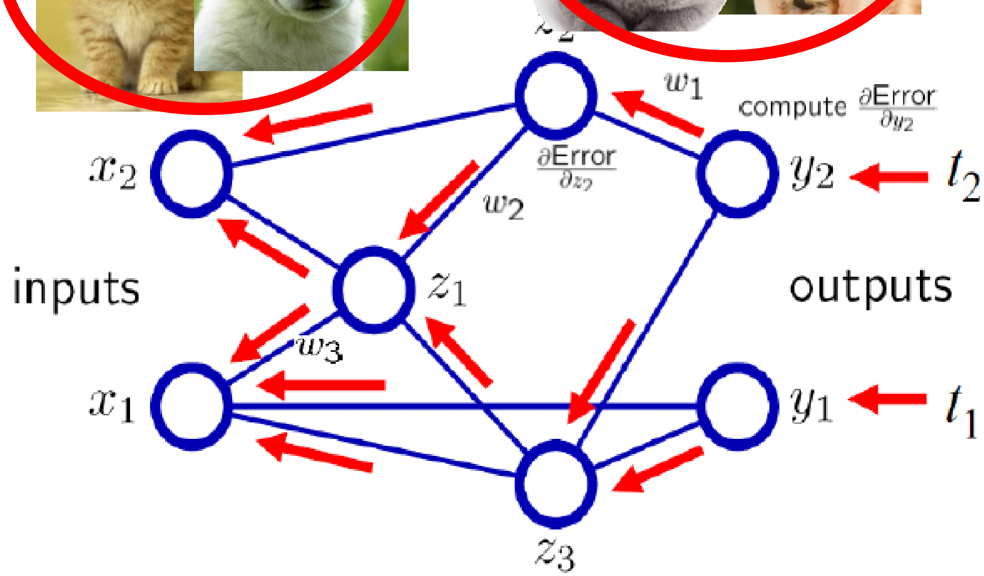


Cat: 0.1

Dog: 0.9

Mini-Batch

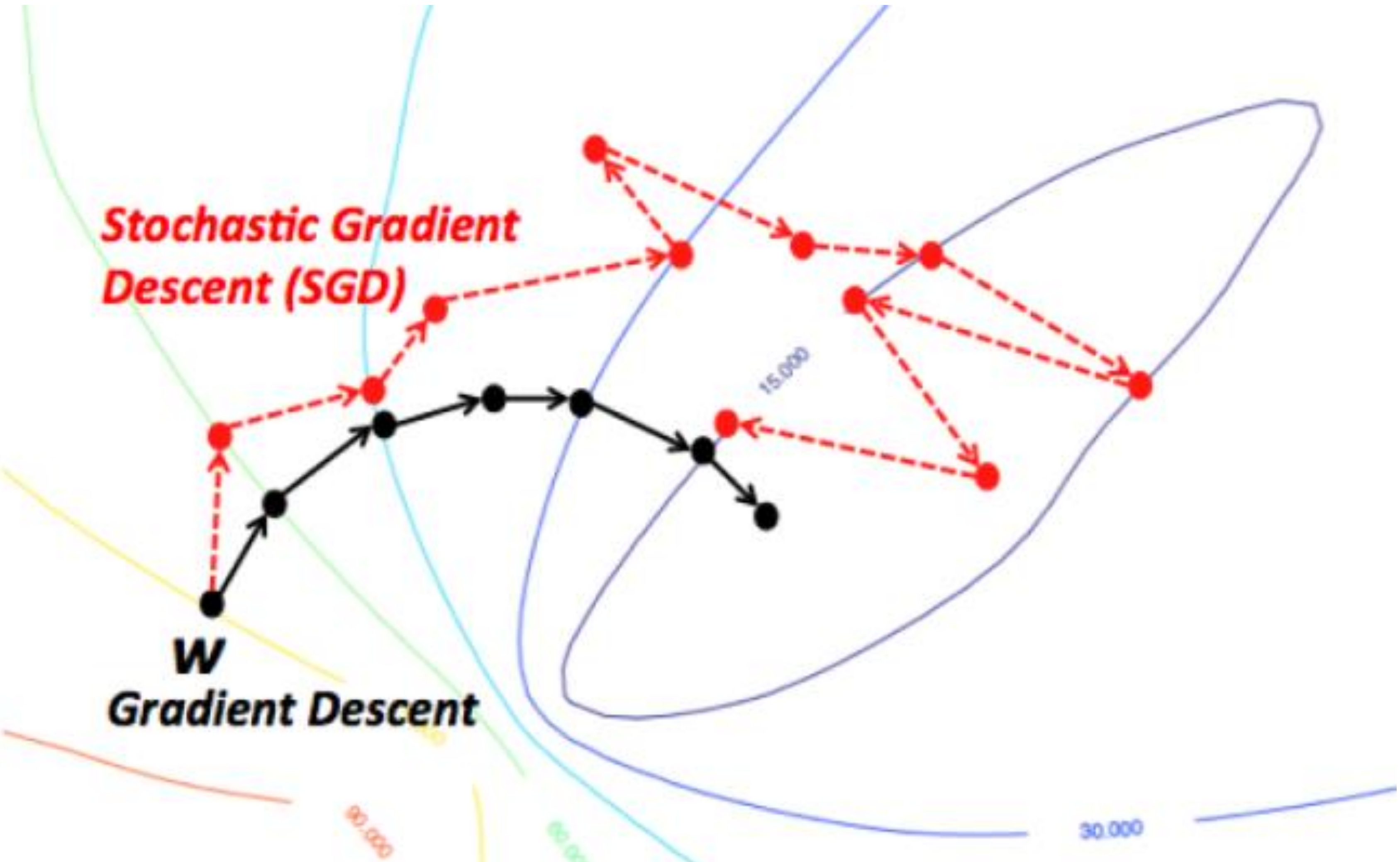
- 샘플 한 개 단위가 아니라 **몇 개의 샘플을 하나의 작은 집합으로 묶은 집합 단위로** 가중치를 업데이트
- 복수의 샘플을 묶은 작은 집합을 **미니배치**(minibatch)라고 함



$$E_t = \frac{1}{N_t}\sum_{n\in D_t} E_n$$

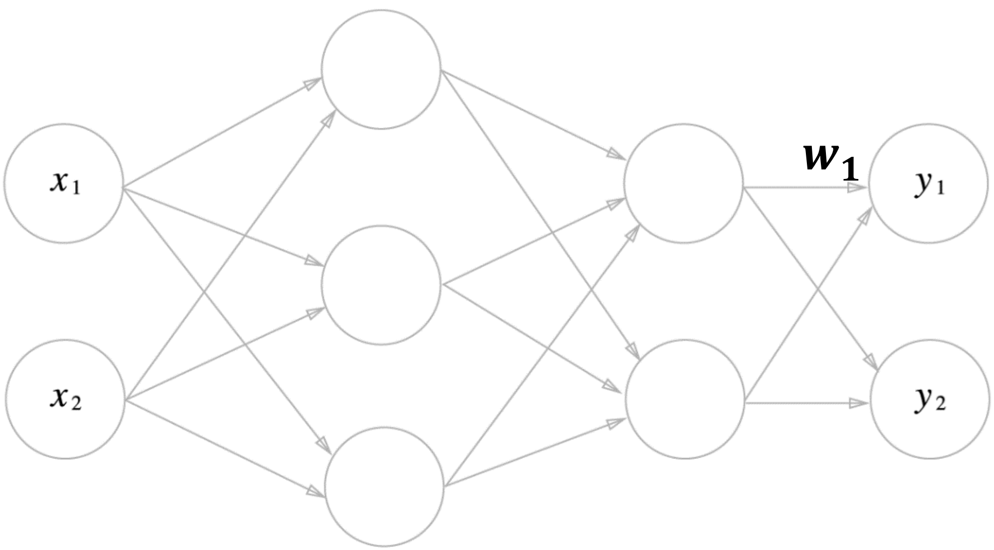
$$w_1^{t+1} = w_1^{t} - \varepsilon \nabla E_t$$

*D_t는 한 개의 미니배치를 나타내며
보통 10~100개의 샘플로 이루어짐


Stochastic Gradient
Descent (SGD)
W
Gradient Descent
15.000
30.000
90.000

AdaGrad[3]

- 개별 가중치에 적응적으로 (adaptive) 학습률을 조정하면서 학습을 진행
- 현재까지 따라서 많이 갱신된 가중치는 학습률을 낮아짐
- 즉, 학습률 감소가 개별 가중치 마다 다르게 적용



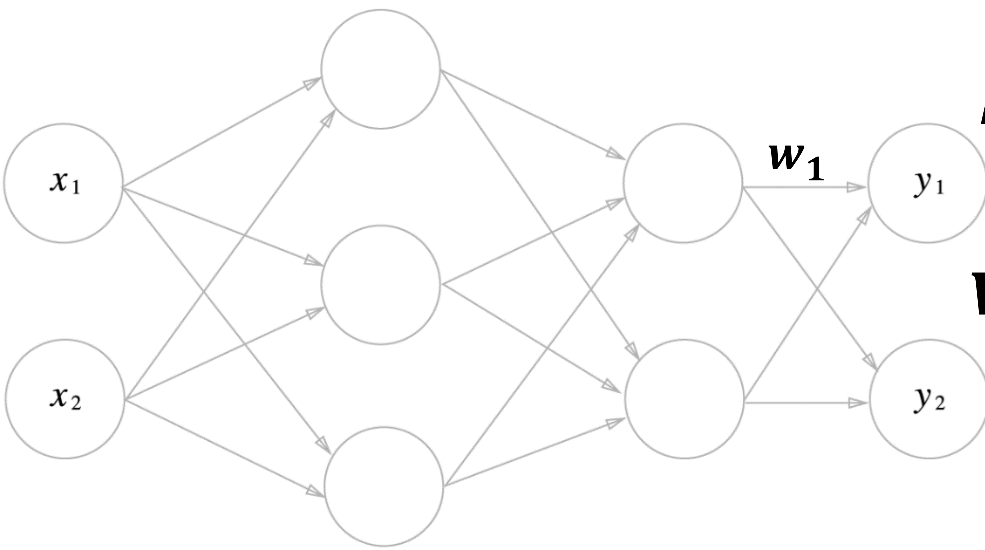
$$h = h + \left(\frac{\partial E}{\partial w_1}\right)^2$$

$$w_1^{t+1} = w_1^t - \varepsilon \frac{1}{\sqrt{h}} \left(\frac{\partial E}{\partial w_1}\right)$$

[3] John Duchi, Elad hazan, and Yoram Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, JMLR 2011

RMSProp

- AdaGrad의 단점을 해결하기 위한 방법
- AdaGrad의 식에서 gradient의 제곱값을 더하는 방식이 아니라 지수평균으로 대체
- Gradient가 무한정 커지는 것을 방지



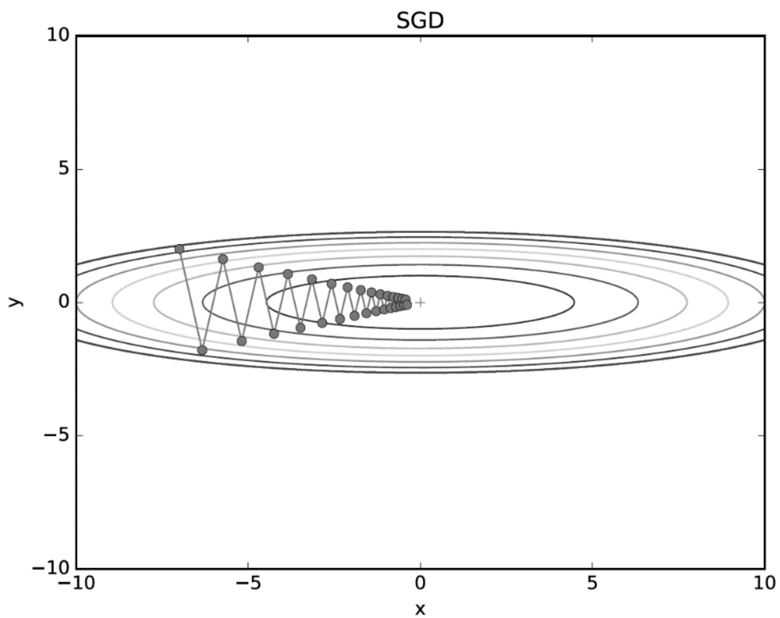
$$h = \mathrm{r}h + (1 - \mathrm{r})\left(\frac{\partial E}{\partial w_1}\right)^2$$

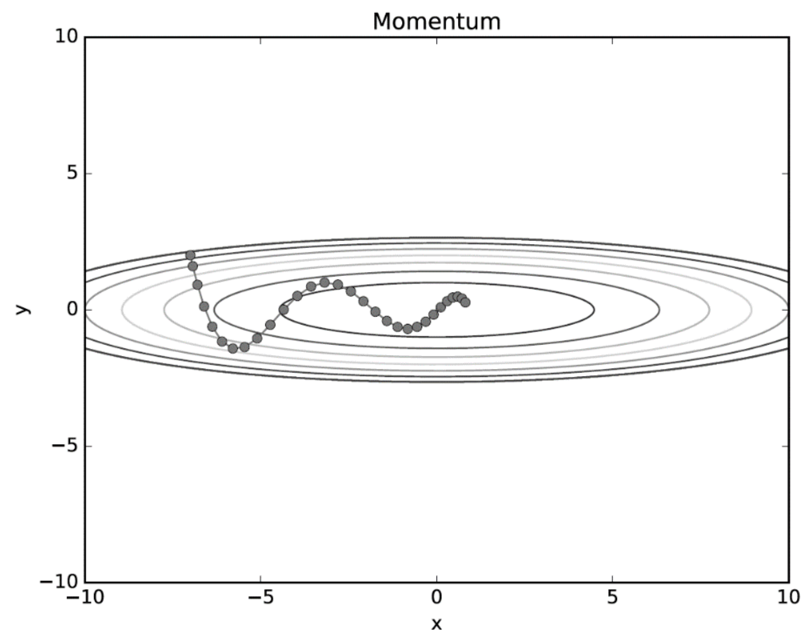
$$w_1^{t+1} = w_1^{t} - \varepsilon \frac{1}{\sqrt{h}}\left(\frac{\partial E}{\partial w_1}\right)$$

Momentum

- 가중치의 업데이트 값에 **이전 업데이트 값의 일정 비율을 더해줌**
- 즉, Gradient decent를 통해 이동하는 과정에 관성을 주는 것
- Adam[4]: AdaGrad (RMSProp) 와 Momentum을 융합한 기법

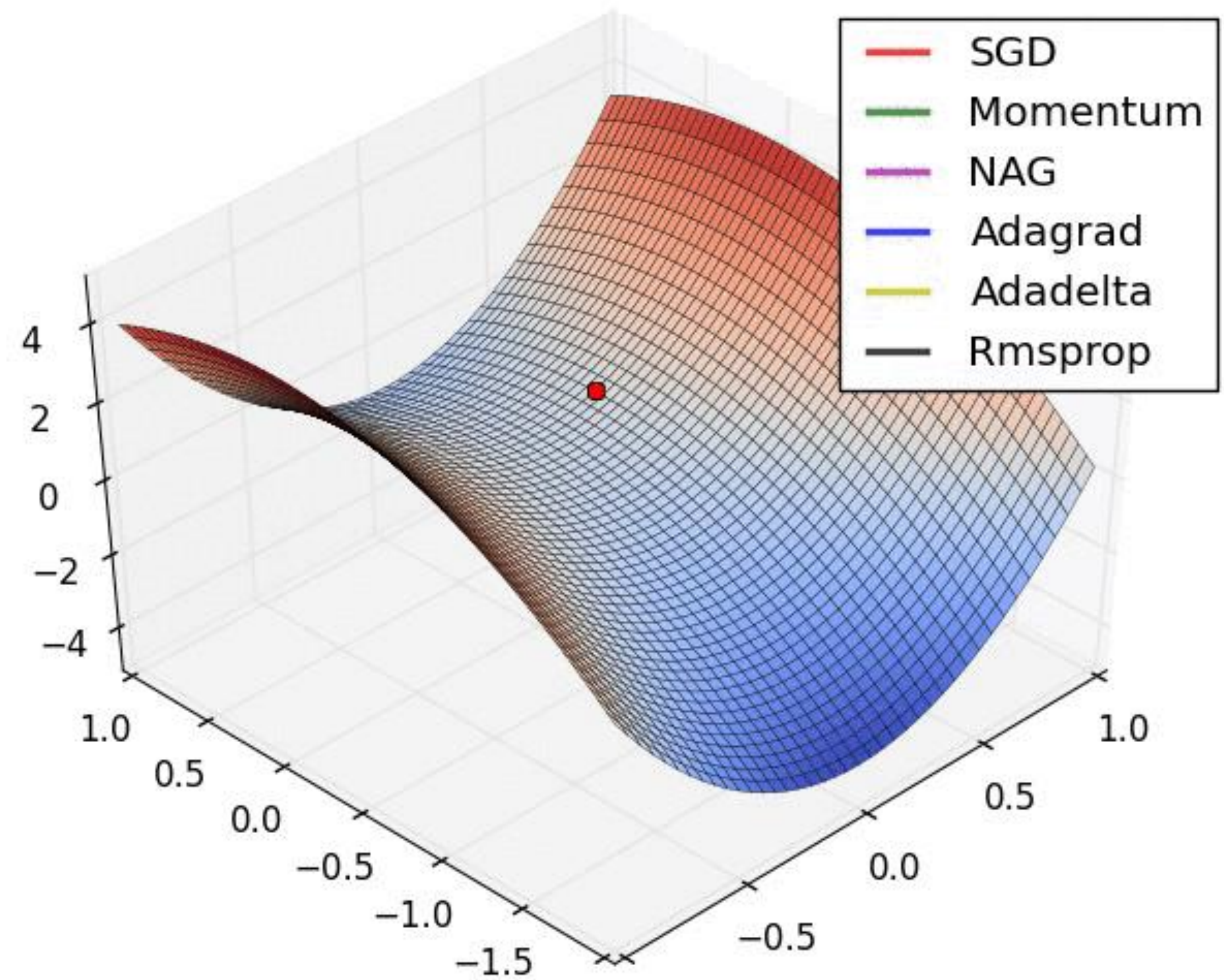
$$w^{t+1} = w^{t} - \varepsilon \nabla E_t + \mu \triangle w^{t-1}$$



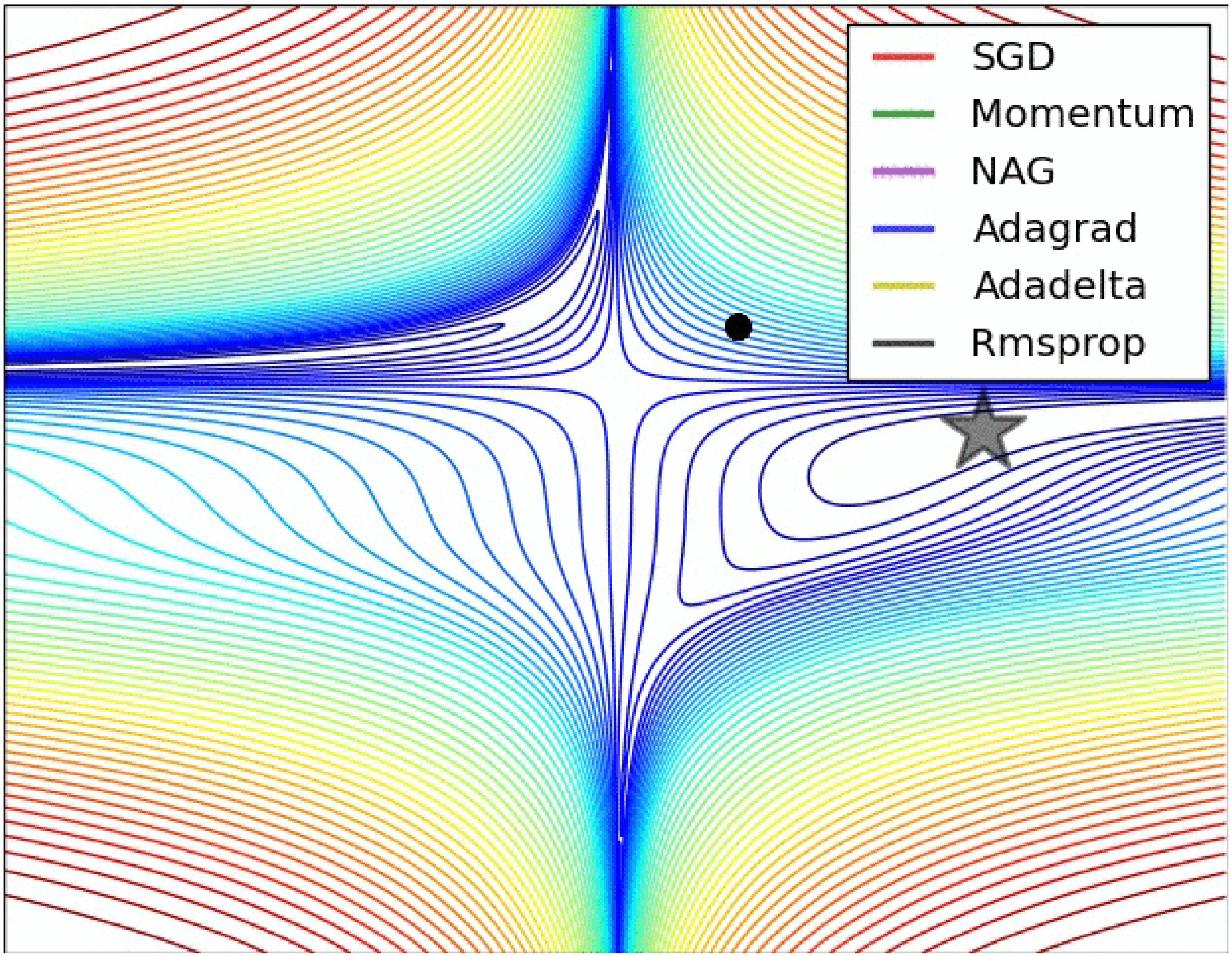


[4] Diederik Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimazation, ICLR 2015

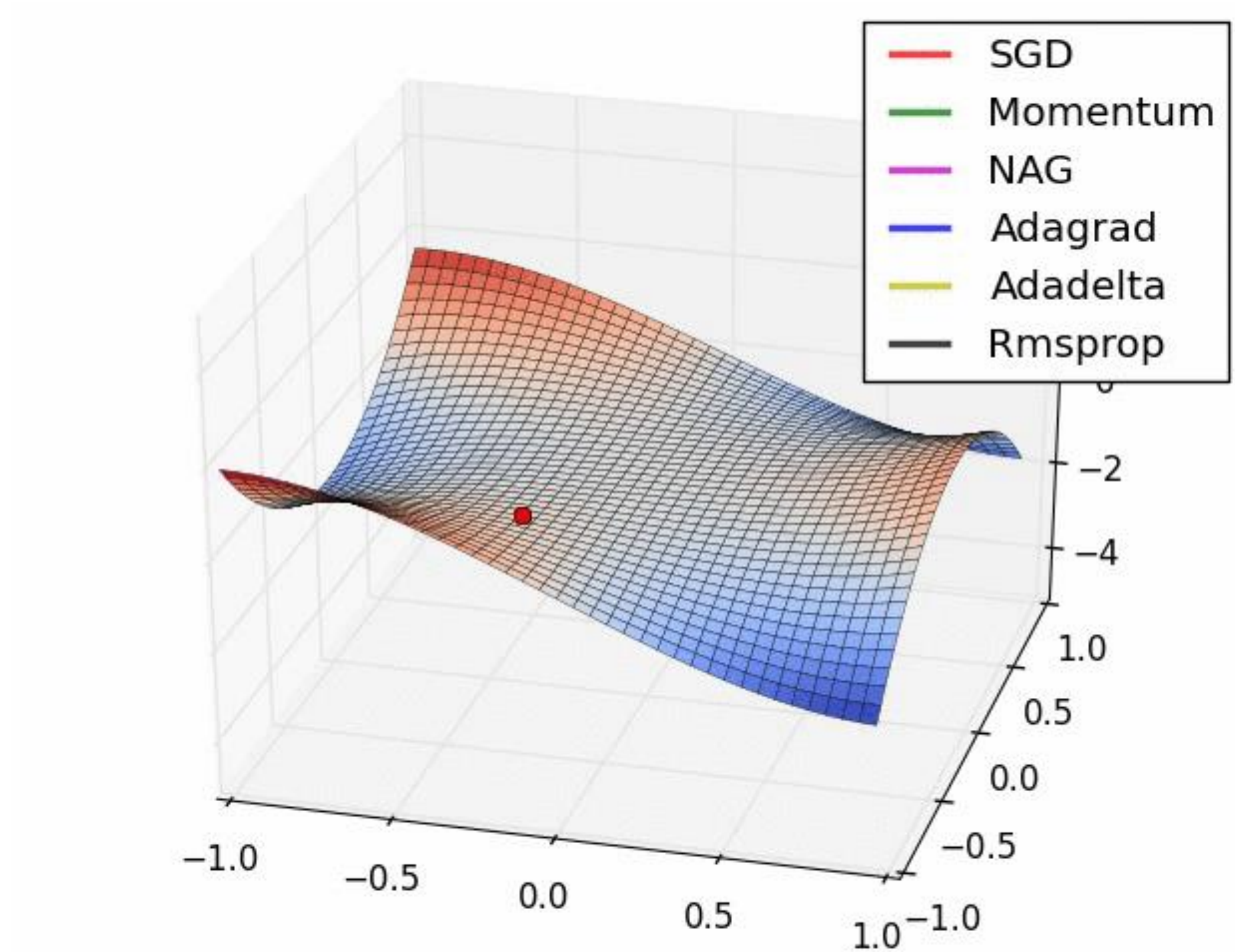
Optimization algorithms at long valley



Optimization algorithms at Beale's function



Optimization algorithms at saddle point



AI School 6기 2주차

필기체 인식기 개발

Perceptron (1958~)

```
import tensorflow as tf

x_data = [[1, 2]]

X = tf.placeholder(tf.float32, shape=[None, 2])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    prediction = sess.run(hypothesis, feed_dict={X: x_data})
    print(prediction)
```

Perceptron 숙제

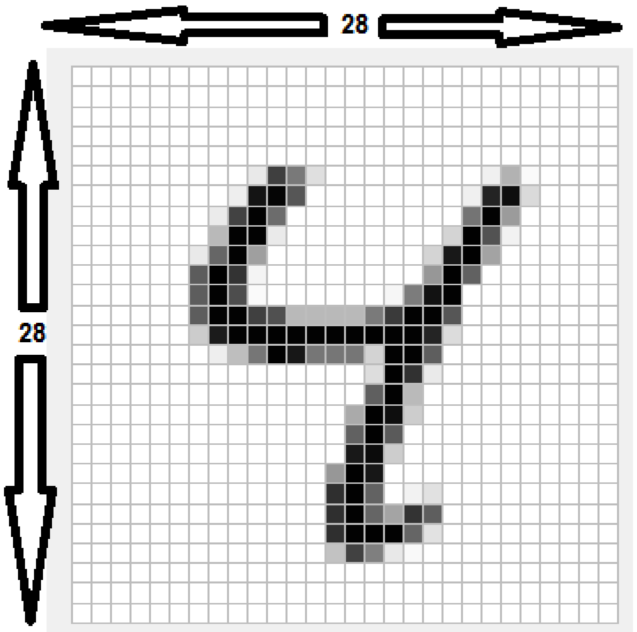
RELU 함수를 사용하여 hypothesis 구현하기!

```
import tensorflow as tf

x_data = [[1, 2]]

X = tf.placeholder(tf.float32, shape=[None, 2])

W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = 구현하기!

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    prediction = sess.run(hypothesis, feed_dict={X: x_data})
    print(prediction)
```

Perceptron training

```
import tensorflow as tf

x_data = [[1, 2], [2, 3], [3, 1], [4, 3], [5, 3], [6, 2]]
y_data = [[0], [0], [0], [1], [1], [1]]

X = tf.placeholder(tf.float32, shape=[None, 2])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

hypothesis = tf.sigmoid(tf.matmul(X, W) + b)
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(10001):
        cost_val, _ = sess.run([cost, train], feed_dict={X: x_data, Y: y_data})
        if step % 200 == 0: print(step, cost_val)
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("₩nHypothesis: ", h, "₩nCorrect (Y): ", c, "₩nAccuracy: ", a)
```

MNIST data

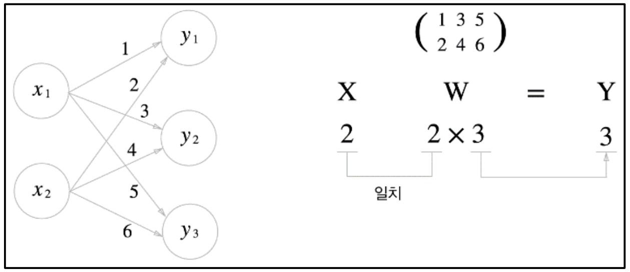



```python
# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])
```

MNIST data

```
import matplotlib.pyplot as plt
import numpy as np

from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

print(np.shape(mnist.train.images))
print(np.shape(mnist.train.labels))
print(np.shape(mnist.test.images))
print(np.shape(mnist.test.labels))

plt.imshow(
        mnist.train.images[1].reshape(28, 28),
        cmap="Greys",
        interpolation="nearest",
    )
plt.show()
```

Feedforward

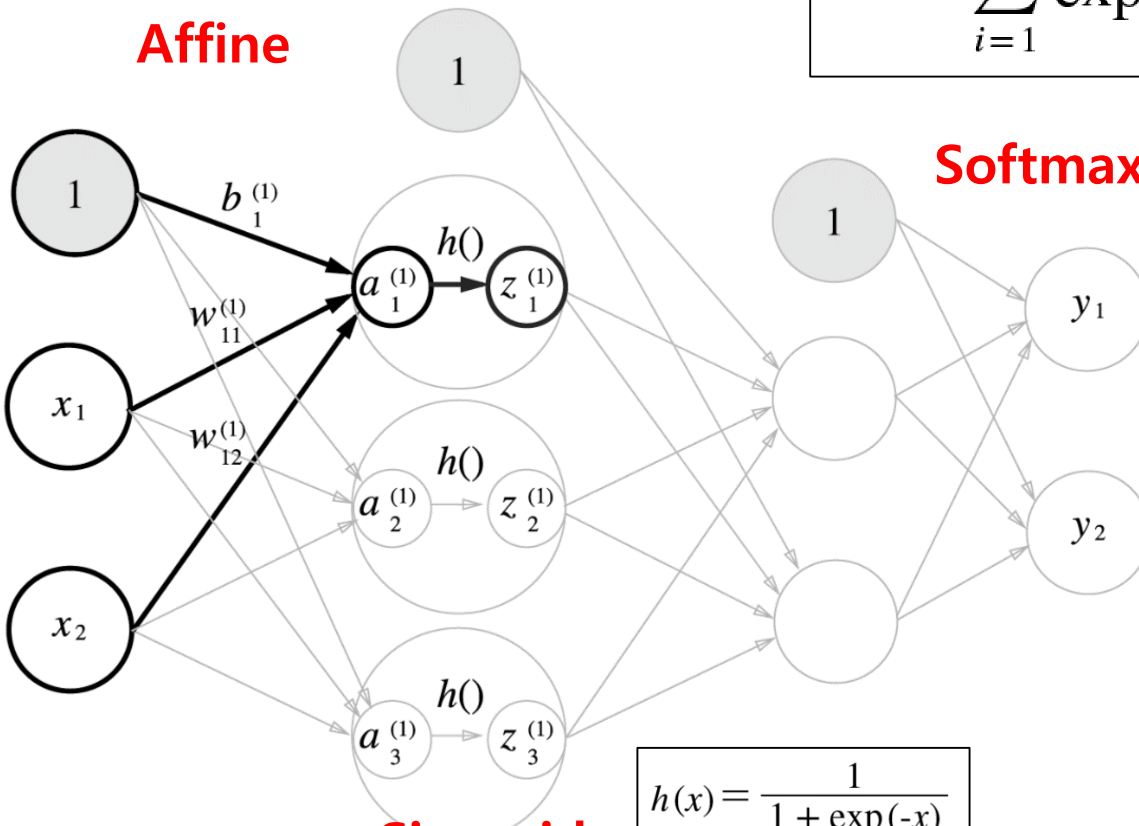


$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

Affine

Softmax

Loss function



$$E = -\sum_k t_k \log y_k$$

$$h(x) = \frac{1}{1 + \exp(-x)}$$

Sigmoid

ReLU

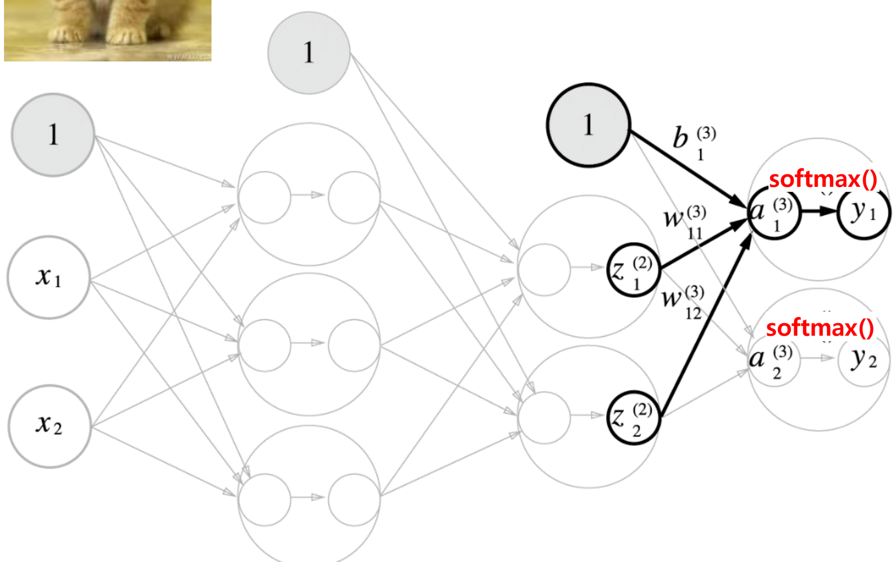
$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

Affine, Activation

```
import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3
```

Softmax

- 입력의 지수함수에 모든 입력의 지수 함수의 합으로 나누어 줌
- 출력을 확률값으로 나타냄 (출력의 총합이 1)
- 분류문제에 사용



Cat: 1.5 $\dfrac{e^{1.5}}{e^{1.5}+e^{3.7}} = 0.1$

Dog: 3.7 $\dfrac{e^{3.7}}{e^{1.5}+e^{3.7}} = 0.9$

Softmax

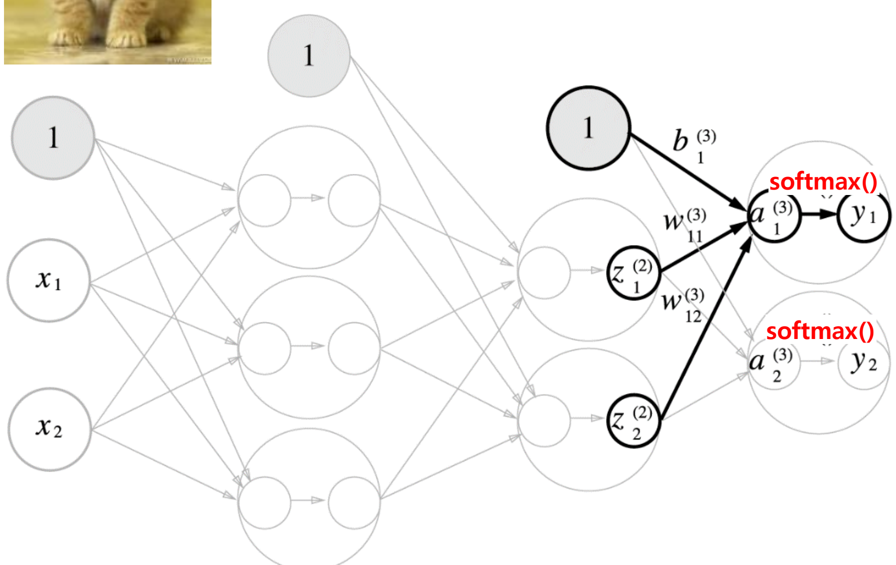
```
.
.
.
hypothesis = tf.matmul(L2, W3) + b3

softmax_result = tf.nn.softmax(hypothesis)

--------참고---------
sess = tf.Session()
sess.run(tf.global_variables_initializer())

x, y = mnist.train.next_batch(1)
feed_dict = {X: x, Y: y}
s, h = sess.run([softmax_result, hypothesis], feed_dict=feed_dict)

print(s)
print(h)
```

Loss Function (Error Function)

- Cross entropy error (교차 엔트로피 오차)
- 정답일 때의 출력이 전체 값을 결정

$$E = -\sum_k t_k \log y_k$$

y = [**0.1**, 0.9]
t = [**1.0**, 0.0]



Cat: 0.1

Dog: 0.9

error

$1 * \log 0.1$

$0 * \log 0.9$

Loss Function (Error Function)

```
hypothesis = tf.nn.softmax(hypothesis)
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

또는

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
```

Reduce mean

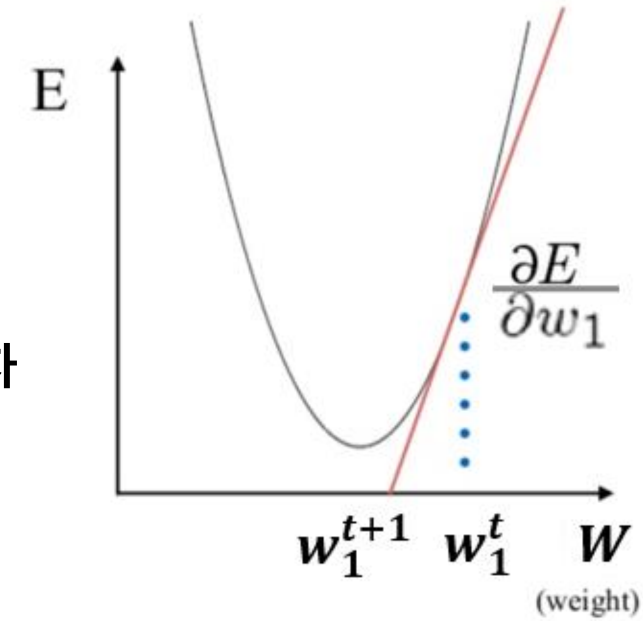
```
x = tf.placeholder(tf.float32, [None, 3])
mean = tf.reduce_mean(x, axis=1)
sess = tf.Session()

print(sess.run(mean, feed_dict={x: [[1.5, 0.5, 1.0], [1., 2., 3.]]}))
```

Gradient decent

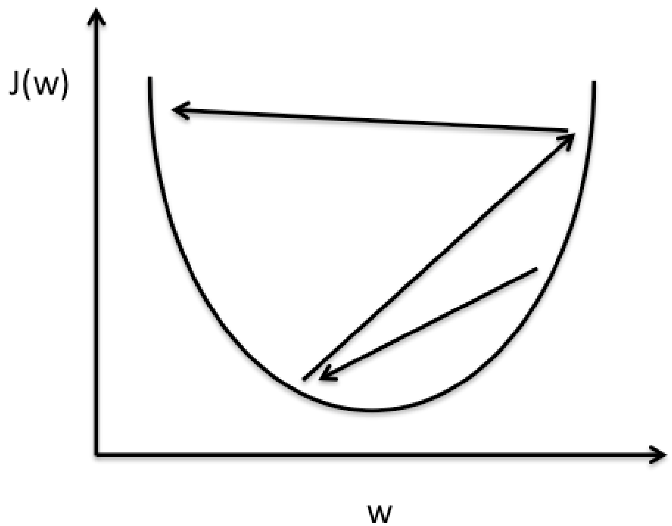
- 임의의 한 지점으로부터 시작해, loss가 줄어드는 방향으로 parameter들을 갱신한다. (loss가 가장 적어질 때까지)
- W를 음의 기울기 방향 (-∇E)으로 조금씩 움직이는 것을 여러 번 반복

$$w_1^{t+1} = w_1^{t} - \varepsilon \, \nabla E$$

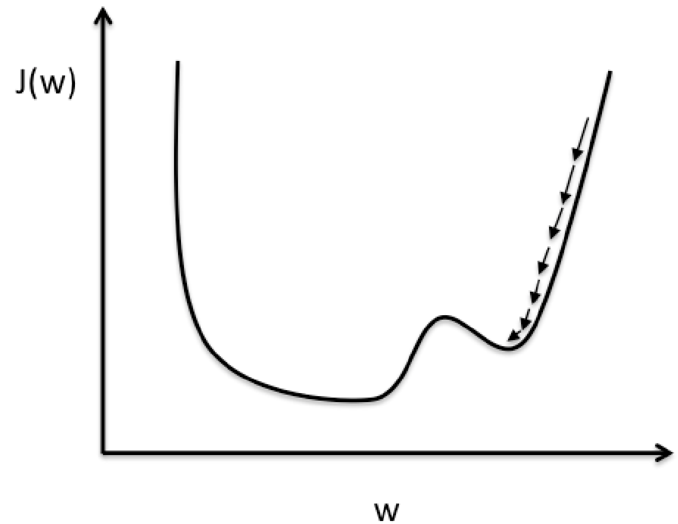
여기서 E = 모든 훈련 샘플에 대하여 계산되는 오차



Learning rate



Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

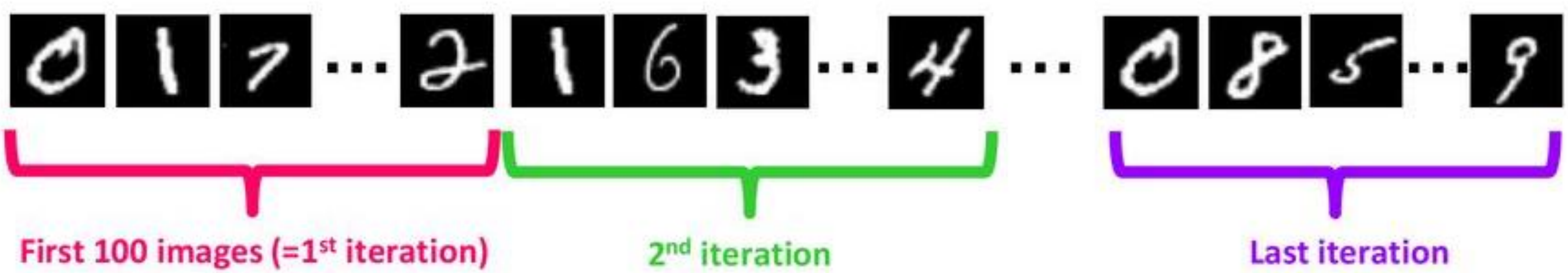
Gradient descent

```
learning_rate = 0.001
optimizer=tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

sess = tf.Session()
sess.run(tf.global_variables_initializer())

training_epochs = 15
batch_size = 100

for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
```

Epoch, Batch size, Iterations

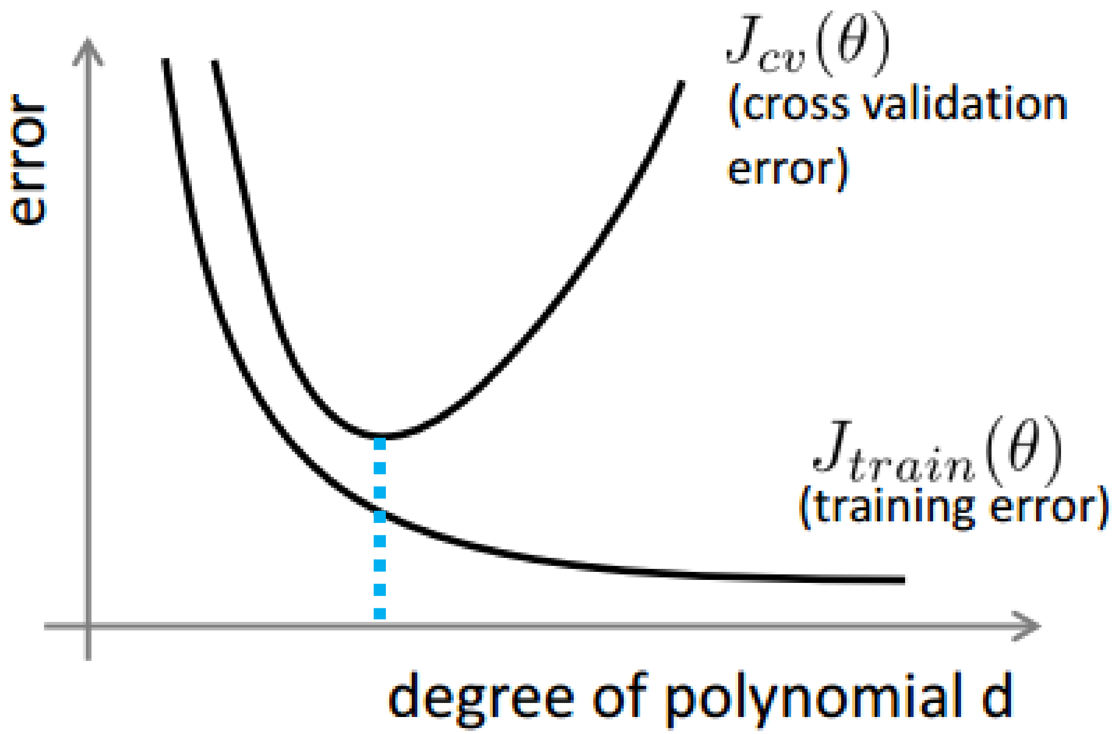
Example: MNIST data

- number of training data: **N=55,000**
- Let's take batch size of **B=100**



- How many iteration in each epoch? **55000/100 = 550** $\boxed{\textbf{1 epoch = 550 iteration}}$

Early Stopping

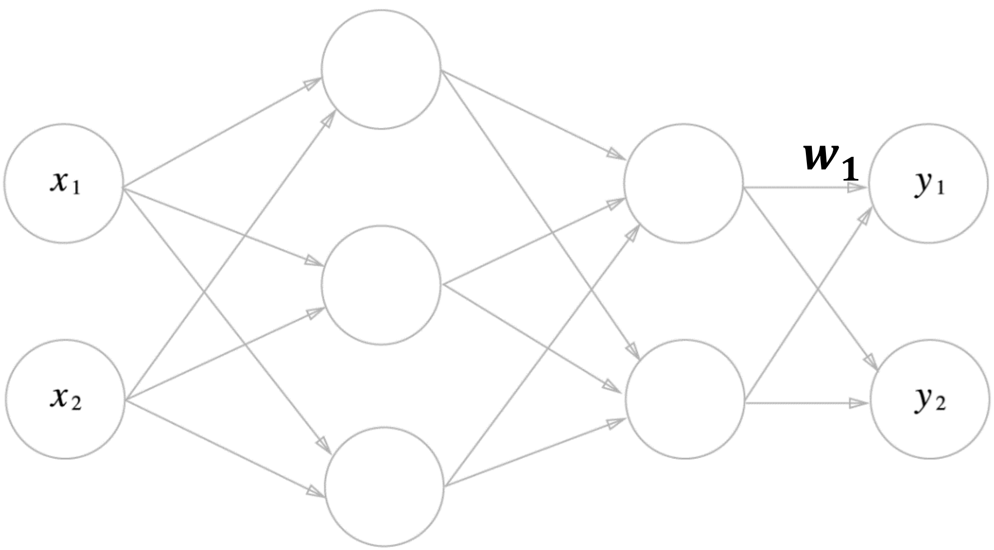


Early Stopping

```
max = 0
early_stopped_time = 0
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'training cost =', '{:.9f}'.format(avg_cost))
    correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    test_accuracy = sess.run(accuracy, feed_dict={X: mnist.test.images, Y:
mnist.test.labels})
    print('Test Accuracy:', test_accuracy)
    if test_accuracy > max:
        max = test_accuracy
        early_stopped_time = epoch + 1
print('Learning Finished!')
print('Test Max Accuracy:', max)
print('Early stopped time:', early_stopped_time)
```

AdaGrad[1]

- 개별 가중치에 적응적으로 (adaptive) 학습률을 조정하면서 학습을 진행
- 현재까지 따라서 많이 갱신된 가중치는 학습률을 낮아짐
- 즉, 학습률 감소가 개별 가중치 마다 다르게 적용



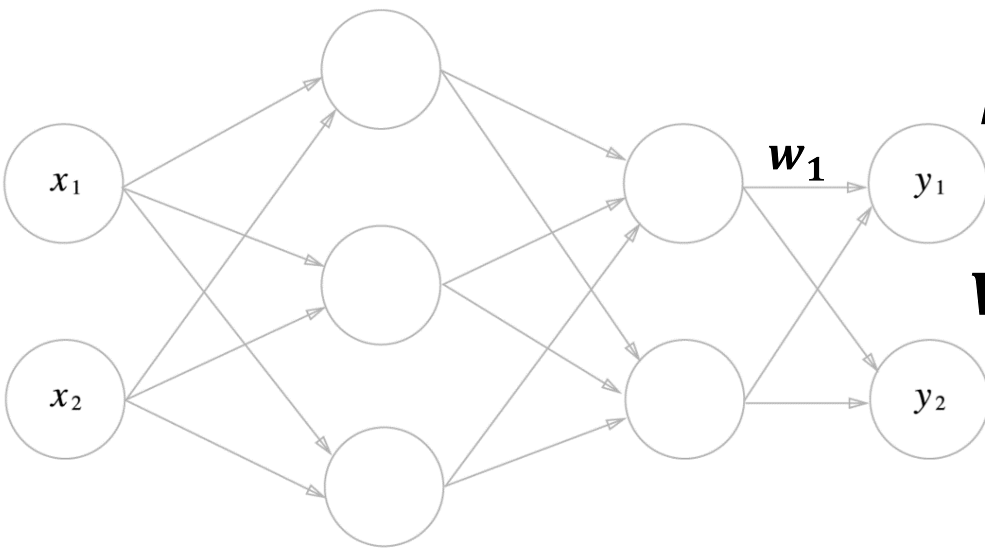
$$h = h + \left(\frac{\partial E}{\partial w_1}\right)^2$$

$$w_1^{t+1} = w_1^t - \varepsilon \frac{1}{\sqrt{h}} \left(\frac{\partial E}{\partial w_1}\right)$$

[1] John Duchi, Elad hazan, and Yoram Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, JMLR 2011

RMSProp

- AdaGrad의 단점을 해결하기 위한 방법
- AdaGrad의 식에서 gradient의 제곱값을 더하는 방식이 아니라 지수평균으로 대체
- Gradient가 무한정 커지는 것을 방지



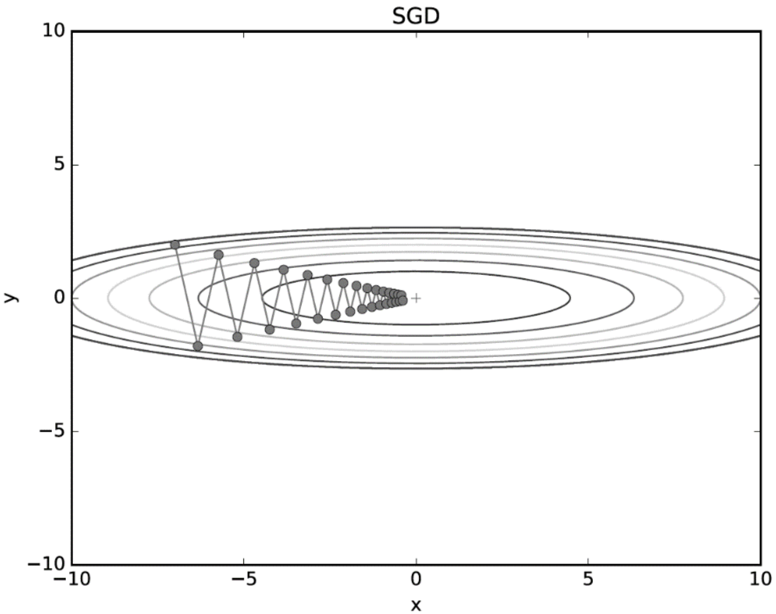
$$h = \mathrm{r}h + (1 - \mathrm{r})\left(\frac{\partial E}{\partial w_1}\right)^2$$

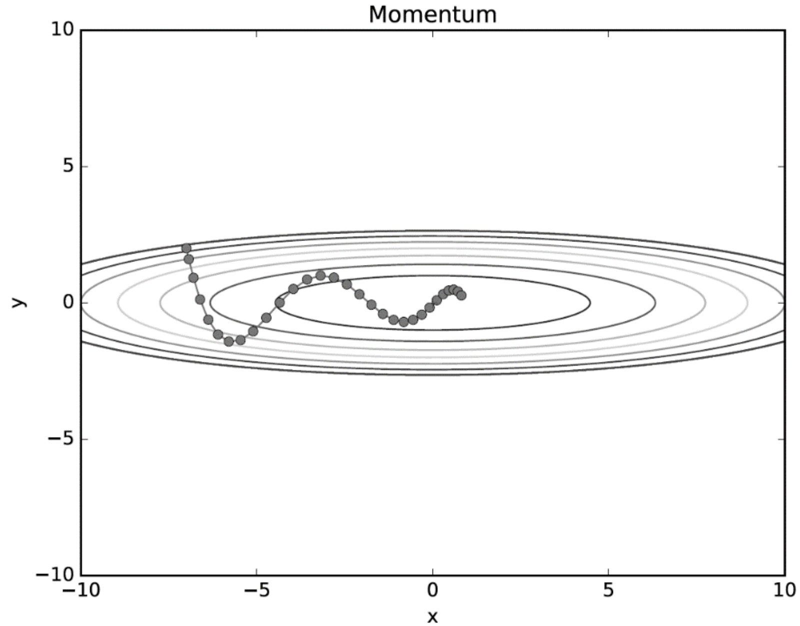
$$w_1^{t+1} = w_1^{t} - \varepsilon \frac{1}{\sqrt{h}}\left(\frac{\partial E}{\partial w_1}\right)$$

Momentum

- 가중치의 업데이트 값에 **이전 업데이트 값의 일정 비율을 더해줌**
- 즉, Gradient decent를 통해 이동하는 과정에 관성을 주는 것
- Adam[1]: AdaGrad (RMSProp) 와 Momentum을 융합한 기법

$$w^{t+1} = w^{t} - \varepsilon \nabla E_t + \mu \triangle w^{t-1}$$





[1] Diederik Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimazation, ICLR 2015

AdaGrad, RMSProp, Momentum, Adam

```
optimizer = tf.train.AdagradOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
optimizer = tf.train.RMSPropOptimizer(learning_rate=learning_rate,
momentum=0.9).minimize(cost)
```

```
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,
momentum=0.9).minimize(cost)
```

```
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

MNIST 숙제 : MLP 더 크고 깊게 쌓기!

```
.
.
.
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

W1 = tf.Variable(tf.random_normal([784, 512]))
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([512, 512]))
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([512, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3
```

Q&A

과제 제출 e-mail: dha8102@naver.com