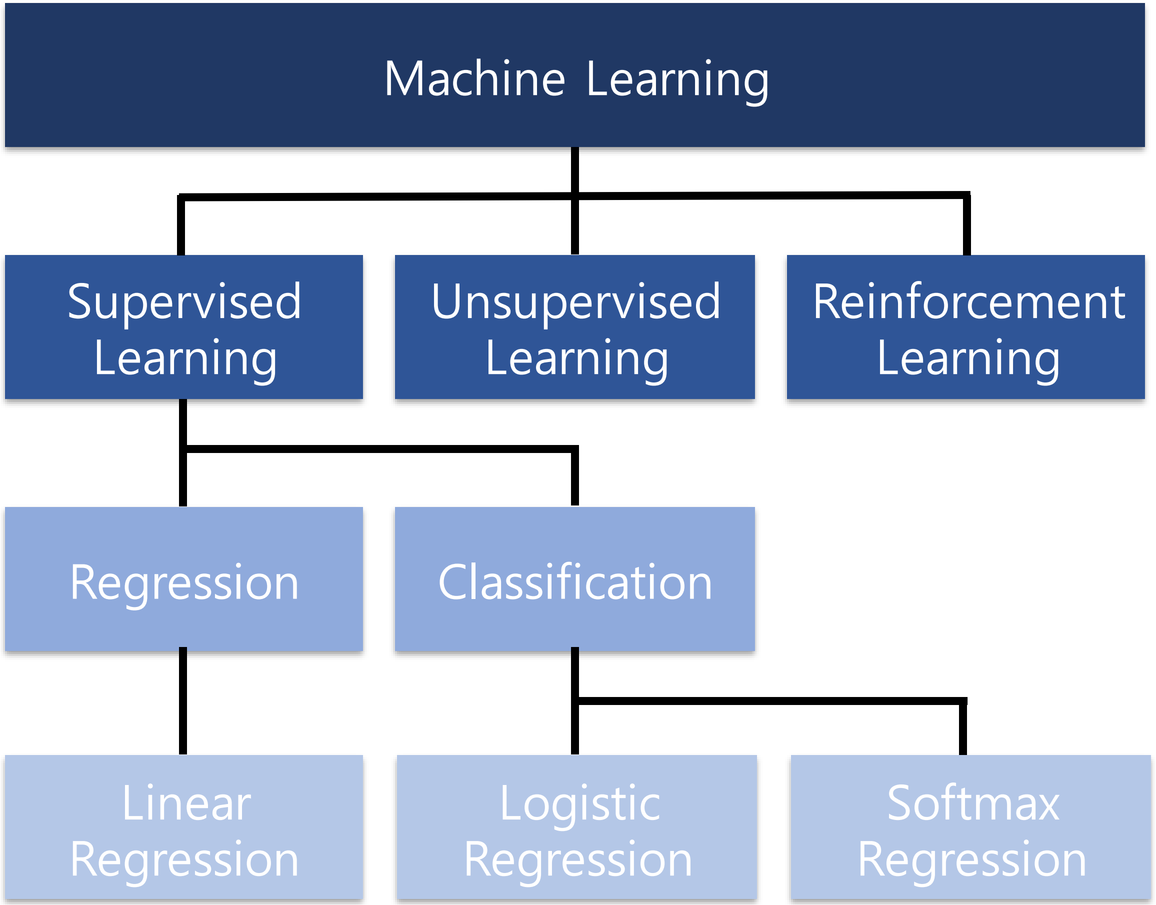
강화학습 스터디 3주차
Lec. 8-10

2020.01.29

서울대학교 조선해양공학과 생산공학 연구실
조영인

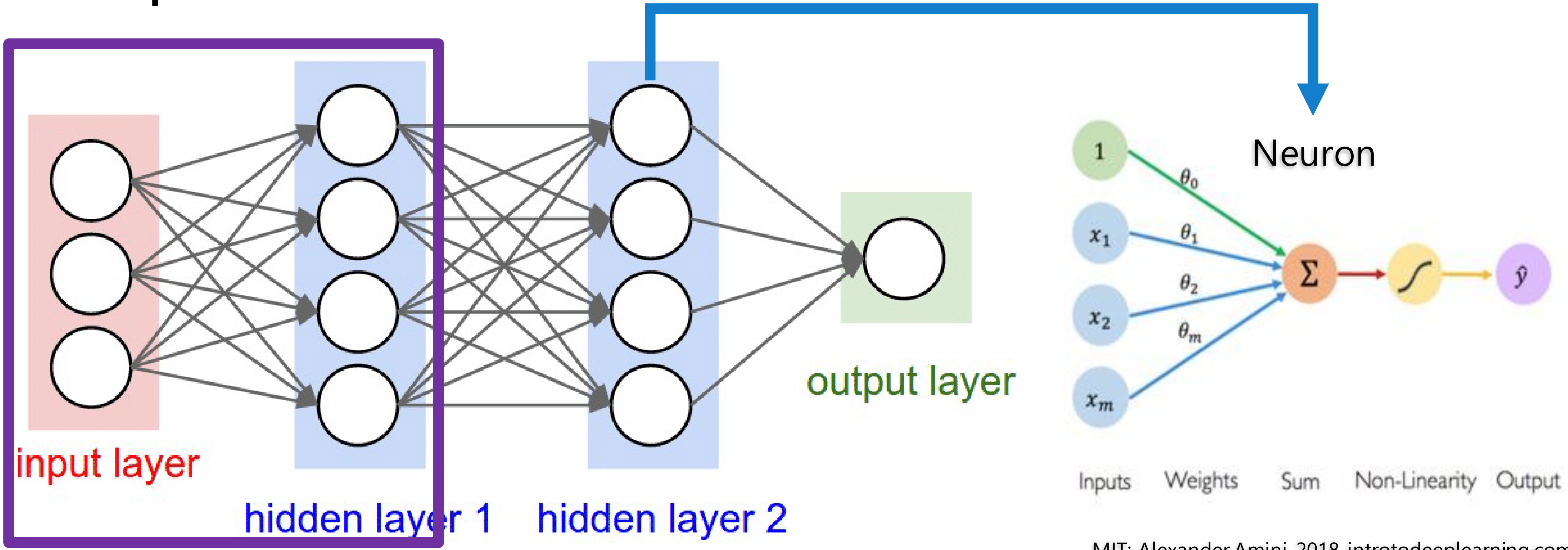
◆ 복습

- **Linear Regression**
  - **Hypothesis:** $\mathrm{H}(\mathbf{X}) = \mathbf{X} \cdot \mathbf{W} + \mathbf{b}$
  - **Cost Function: MSE**
- **Logistic Regression**
  - **Hypothesis:** $\mathrm{H}(\mathbf{X}) = \frac{1}{1+e^{-\mathbf{X}\cdot\mathbf{W}}}$
  - **Cost Function: Cross Entropy**
- **Softmax Regression**
  - **Hypothesis:** $\mathrm{H}(\mathbf{X}) = \frac{e^{-\mathbf{X}\cdot\mathbf{W}}}{\sum e^{-\mathbf{X}\cdot\mathbf{W}}}$
  - **Cost Function: Cross Entropy**



Deep Neural Network

◆ Concept of DNN



MIT: Alexander Amini, 2018 introtodeeplearning.com

Speeding up Deep Learning Computational Aspects of Machine Learning - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/A-general-model-of-a-deep-neural-network-It-consists-of-an-input-layer-some-here-two_fig1_308414212 [accessed 26 Jan, 2020]

◆ Forward Propagation

$\mathbf{X} \cdot \mathbf{W}$

$$= \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & x_{m3} \end{pmatrix} \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} & w_{41}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} & w_{42}^{(1)} \\ w_{13}^{(1)} & w_{23}^{(1)} & w_{33}^{(1)} & w_{43}^{(1)} \end{pmatrix}$$

$$= \begin{pmatrix} x_{11} w_{11}^{(1)} + x_{12} w_{12}^{(1)} + x_{13} w_{13}^{(1)} & \cdots & x_{11} w_{41}^{(1)} + x_{12} w_{42}^{(1)} + x_{13} w_{43}^{(1)} \\ x_{21} w_{11}^{(1)} + x_{22} w_{12}^{(1)} + x_{23} w_{13}^{(1)} & \cdots & x_{21} w_{11}^{(1)} + x_{22} w_{12}^{(1)} + x_{23} w_{13}^{(1)} \\ \vdots & \ddots & \vdots \\ x_{m1} w_{11}^{(1)} + x_{m2} w_{12}^{(1)} + x_{m3} w_{13}^{(1)} & \cdots & x_{m1} w_{11}^{(1)} + x_{m2} w_{12}^{(1)} + x_{m} w_{13}^{(1)} \end{pmatrix}$$

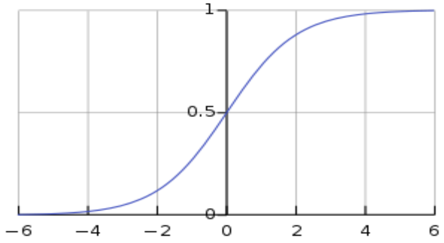
$$= \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & a_{14}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} & a_{24}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1}^{(1)} & a_{m2}^{(1)} & a_{m3}^{(1)} & a_{m4}^{(1)} \end{pmatrix}$$

$\mathbf{Z} = f(\mathbf{X} \cdot \mathbf{W})$

$$= \begin{pmatrix} f\left(a_{11}^{(1)}\right) & f\left(a_{12}^{(1)}\right) & f\left(a_{13}^{(1)}\right) & f\left(a_{14}^{(1)}\right) \\ f\left(a_{21}^{(1)}\right) & f\left(a_{22}^{(1)}\right) & f\left(a_{23}^{(1)}\right) & f\left(a_{24}^{(1)}\right) \\ \vdots & \vdots & \vdots & \vdots \\ f\left(a_{m1}^{(1)}\right) & f\left(a_{m2}^{(1)}\right) & f\left(a_{m3}^{(1)}\right) & f\left(a_{m4}^{(1)}\right) \end{pmatrix}$$

$$= \begin{pmatrix} z_{11}^{(1)} & z_{12}^{(1)} & z_{13}^{(1)} & z_{14}^{(1)} \\ z_{21}^{(1)} & z_{22}^{(1)} & z_{23}^{(1)} & z_{24}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ z_{m1}^{(1)} & z_{m2}^{(1)} & z_{m3}^{(1)} & z_{m4}^{(1)} \end{pmatrix}$$

$$f(x) = \frac{1}{1 + e^{-x}}$$



◆ Tensor

- Tensorflow의 기본 데이터 형식으로 다차원 배열을 의미

```
t = tf.constant([1,2,3,4])
tf.shape(t).eval()
```

```
array([4], dtype=int32)
```

```
t = tf.constant([[1,2],
                 [3,4]])
tf.shape(t).eval()
```

```
array([2, 2], dtype=int32)
```

```
t = tf.constant([[[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]],
                  [[13, 14, 15, 16], [17, 18, 19, 20], [21, 22, 23, 24]]]])
tf.shape(t).eval()
```

```
array([1, 2, 3, 4], dtype=int32)
```

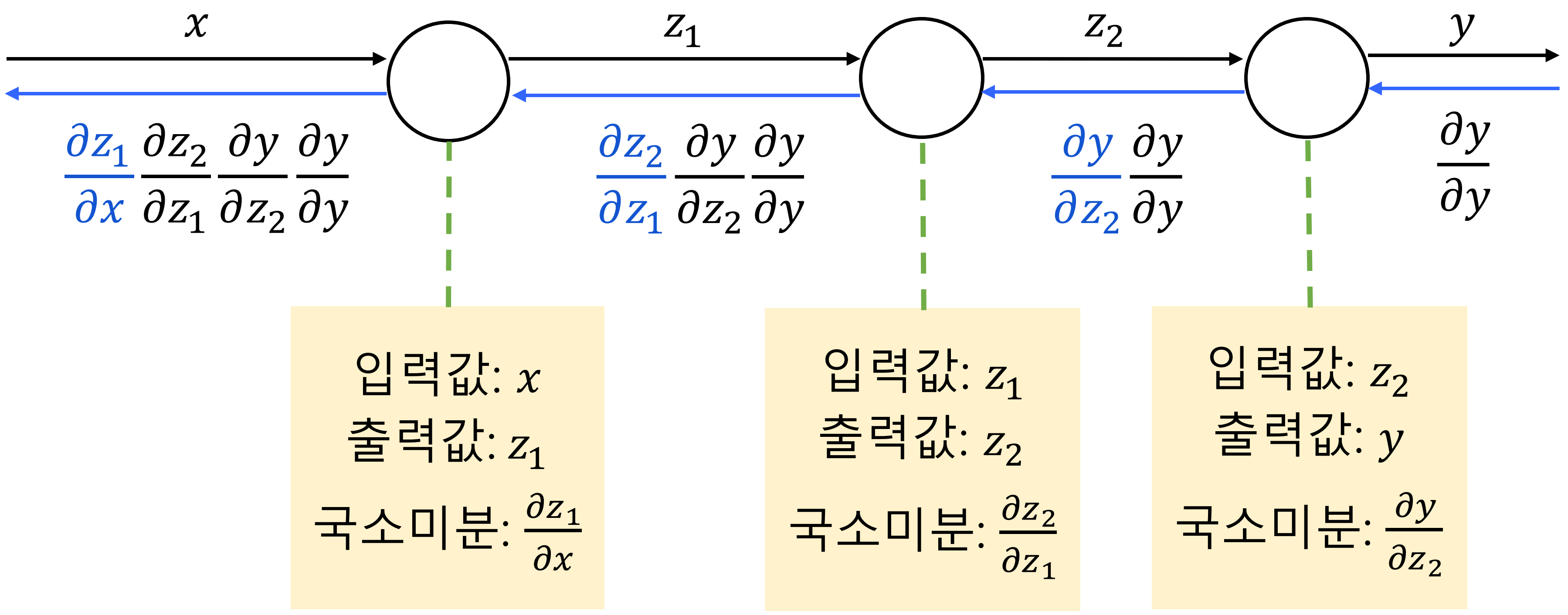
◆ Tensorflow에서 DNN 구현

- 예시: 총 4개의 층으로 이루어진 DNN

```
W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)

W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)

W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)
```

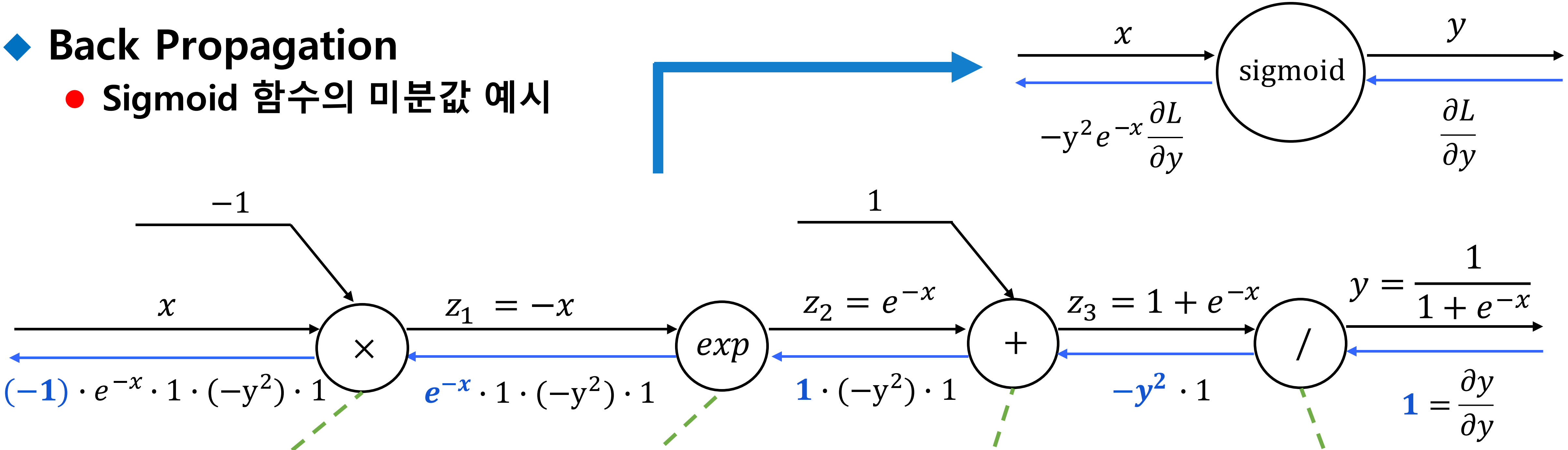
◆ Back Propagation

- 합성함수의 미분 공식인 Chain Rule을 사용하여 각 가중치의 미분값 계산



◆ Back Propagation

- Sigmoid 함수의 미분값 예시



$$\frac{\partial z_1}{\partial x} = \frac{\partial}{\partial x}(-x) = -1$$

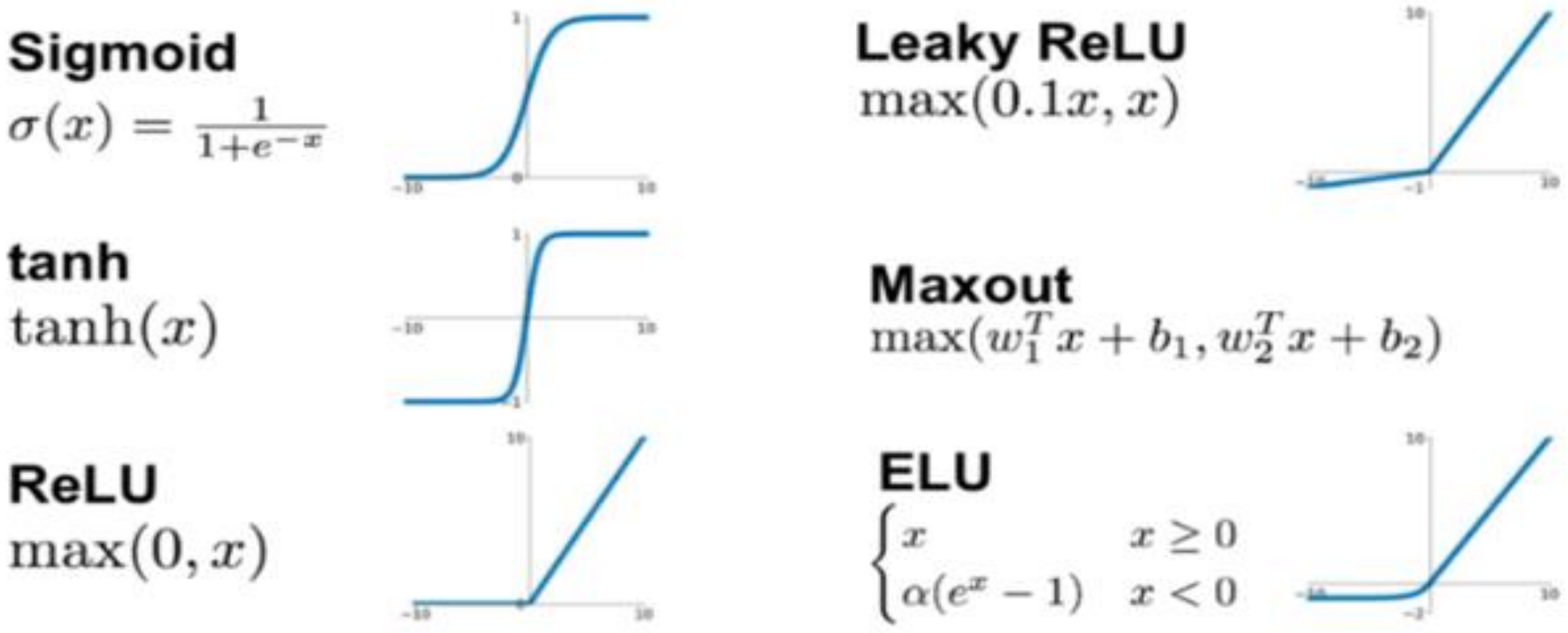
$$\frac{\partial z_2}{\partial z_1} = \frac{\partial}{\partial z_1}(e^{z_1}) = e^{z_1} = e^{-x}$$

$$\frac{\partial z_3}{\partial z_2} = \frac{\partial}{\partial z_2}(z_2 + 1) = 1$$

$$\frac{\partial y}{\partial z_3} = \frac{\partial}{\partial z_3}\left(\frac{1}{z_3}\right) = -\frac{1}{z_3^2} = -y^2$$

◆ Activation Function의 종류

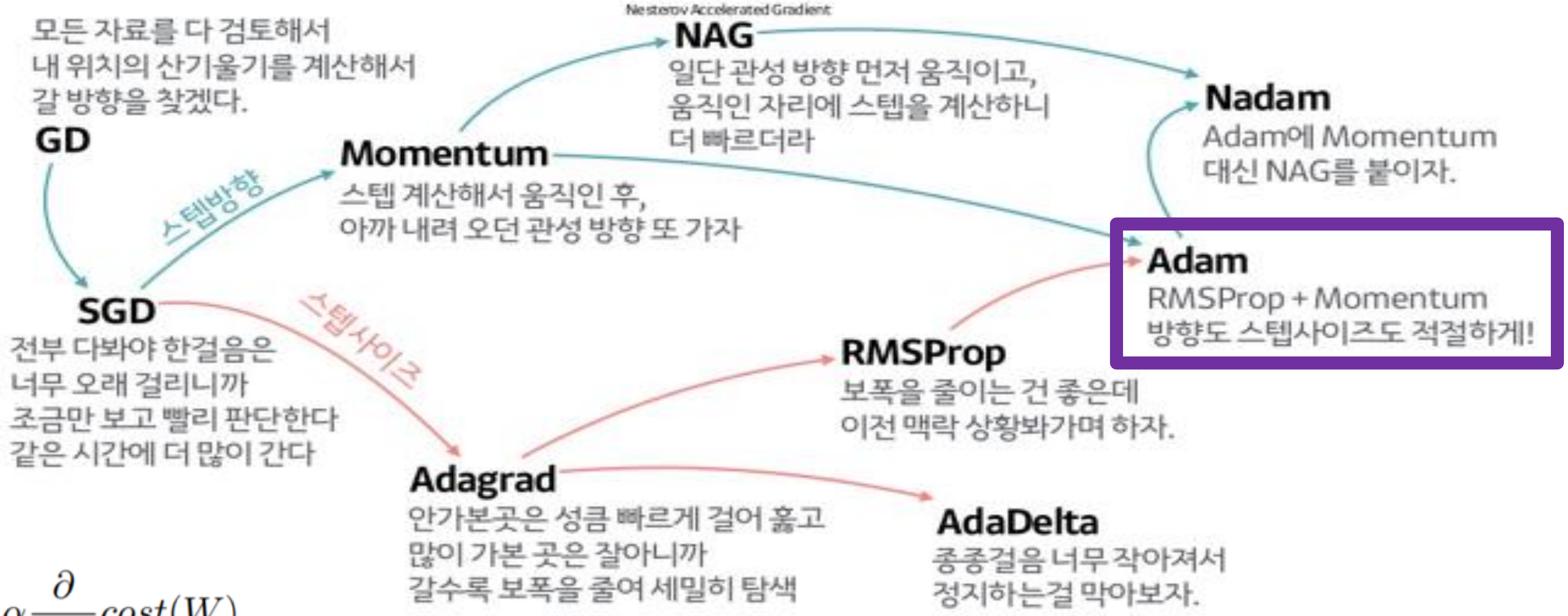
- Activation Function로 비선형 함수를 사용



https://medium.com/@kmkgabia/ml-sigmoid-%EB%8C%80%EC%8B%A0-relu-%EC%83%81%ED%99%A9%EC%97%90-%EB%A7%9E%EB%8A%94-%ED%99%9C%EC%84%B1%ED%99%94-%ED%95%A8%EC%88%98-%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0-c65f620ad6fd

◆ Optimizer

- 가중치를 업데이트함에 있어서 다양한 방법이 존재

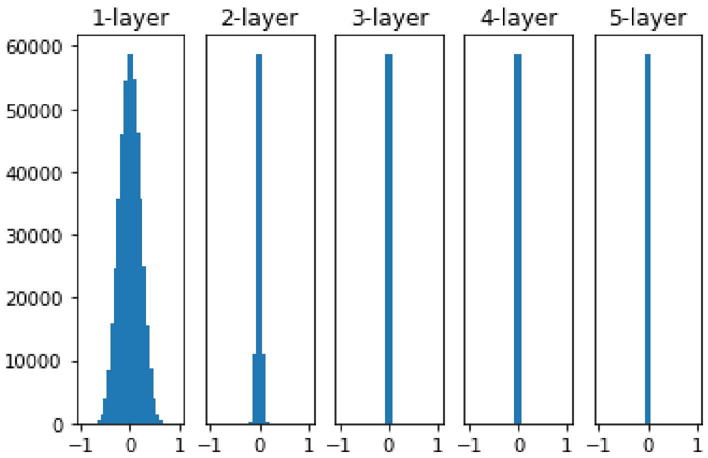


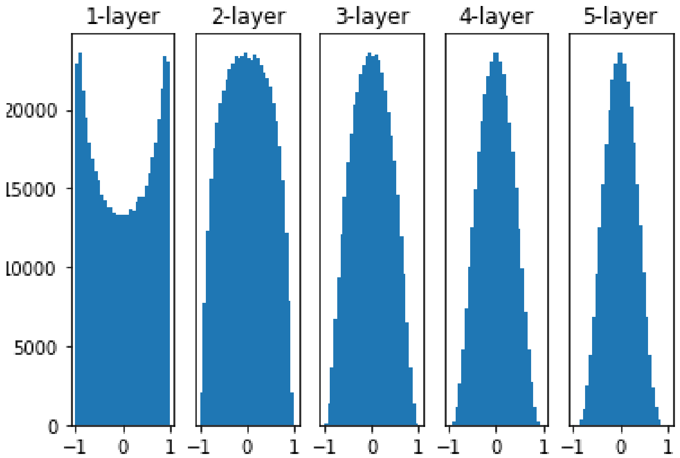
$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

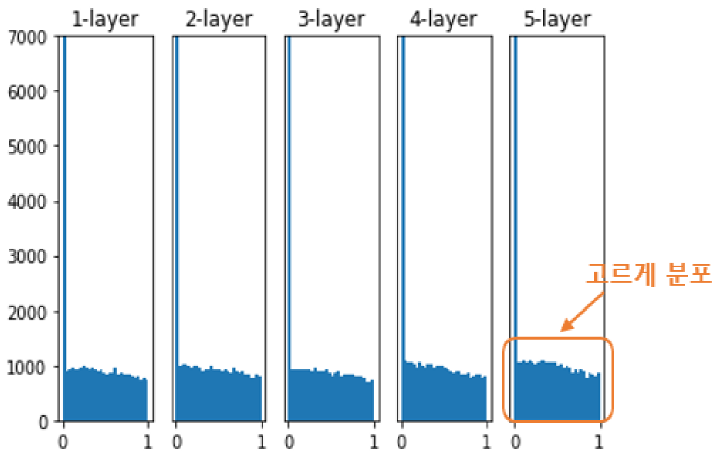
https://www.slideshare.net/yongho/ss-79607172

◆ Initialization

- 가중치의 초기화는 인공신경망의 학습에 있어서 굉장히 중요한 요인
- 평균이 0이고 표준편차가 0.01인 정규분포로 초가화한 경우
- Sigmoid를 활성화 함수로 사용하는 경우: Xavier Initialization
  - 평균이 0이고 표준편차가 $\frac{1}{\sqrt{n_{inputs}}}$인 정규분포로 초기화
- ReLU를 활성화 함수로 사용하는 경우: He Initialization
  - 평균이 0이고 표준편차가 $\sqrt{\frac{2}{n_{inputs}}}$인 정규분포로 초기화







Initialization

◆ Tensorflow의 Initialization

- 예시: Xavier Initialization

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

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581
W1 = tf.get_variable("W1", shape=[784, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[256, 256],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3
```

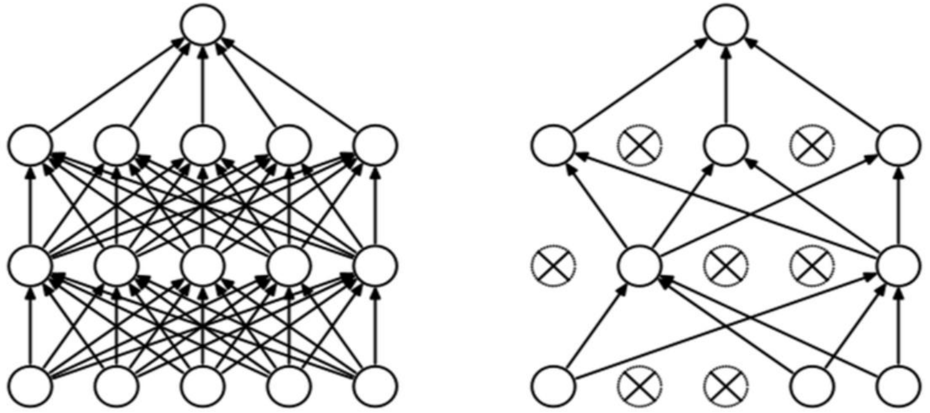
Overfitting

◆ Overfitting 문제의 해결 방안: Regularization

- Weight Decay
  - 일반적으로 overfitting은 가중치 값이 커져서 발생하는 경우가 다수
  - Cost function에 norm 항을 추가

$$cost(W) + \boxed{\lambda \sum W^2}$$

- Dropout
  - 학습 시 일부 노드를 제외하고 학습 진행
  - 여러 개의 모델을 사용하는 것과 같은 효과
  - 앙상블 학습과 비슷한 개념



- Batch Normalization
  - 활성화 함수 이전 또는 이후에서 수행
  - 미니배치 단위로 들어온 데이터를 평균이 0, 분산이 1이 되도록 정규화

Overfitting

◆ Tensorflow의 Dropout 구현

```
keep_prob = tf.placeholder(tf.float32)

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

W2 = tf.get_variable("W2", shape=[512, 512])
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)
```

```
# train my model
for epoch in range(training_epochs):
    ...
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

조영인 whduddlsi@snu.ac.kr

END OF PRESENTATION