

# BM 과정

2019

에스튜비즈  
대표컨설턴트  
이 이 백



## 목 차

- I. Big Data Architecture와 AI 시스템의 Position
- II. AI 구축 방법론
- III. Industry 4.0
- IV. AI 시스템 구축을 위한 기본 지식과 기술
- V. Deep Learning Revisit
- VI. Business Model과 AI Topic 발굴

# V

## Deep Learning Revisit

1. Perceptron
2. MLP
3. CNN
4. RNN/LSTM
5. Autoencoder

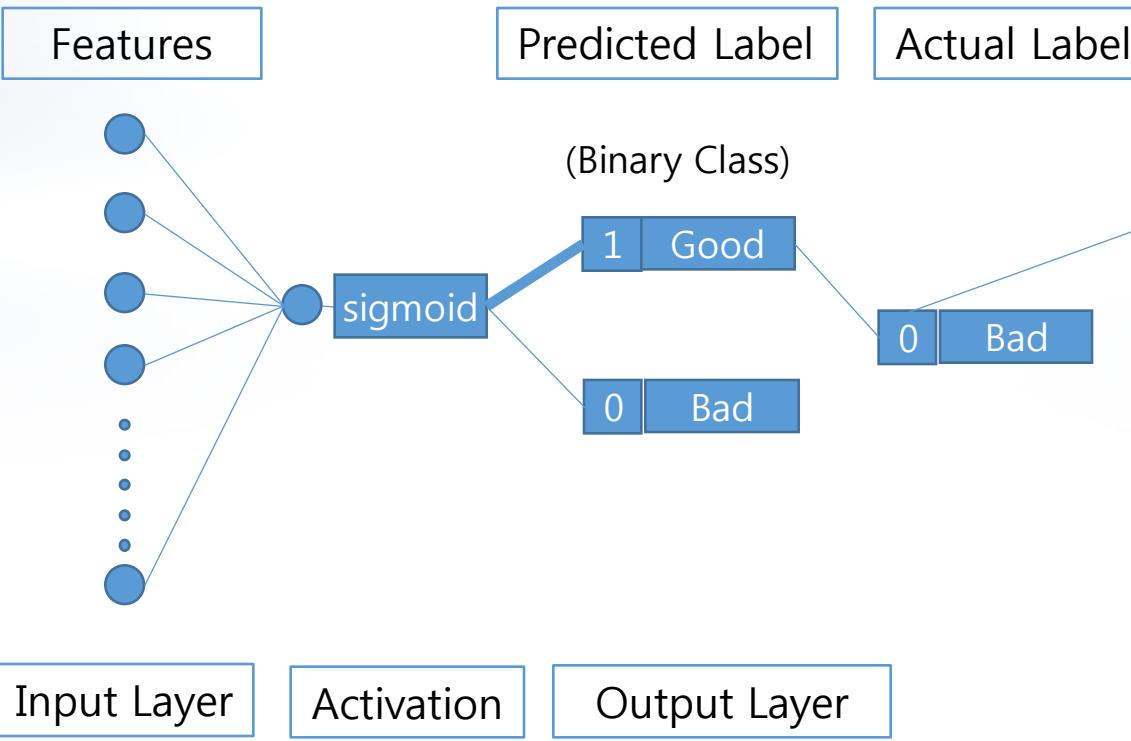


## V. Deep Learning Revisit

# 1. Perceptron

- Perceptron > Classification
    - **Case : Logistic Regression for Binary Class Label**

Observation			
...	3	2	1
	112	21	1
	57	0	3
	193	15	5
	-11	58	2



Obs	Pred	Actual
1	1	0
2	0	0
3	1	1
...		

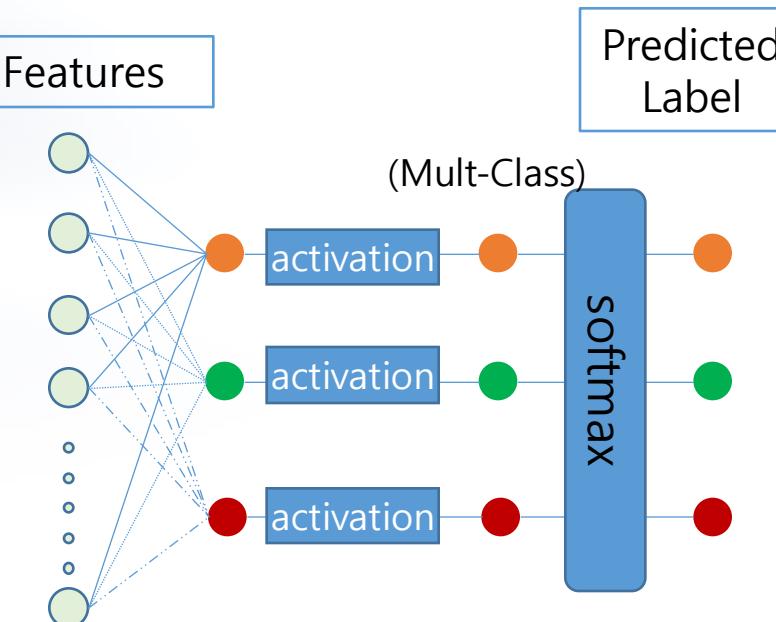
Accuracy = 66.67%

## V. Deep Learning Revisit

# 1. Perceptron

- Perceptron > Classification
    - **Case : Softmax Regression for Multi-Class Label**

Observation			
...	3	2	1
	112	21	1
	57	0	3
	193	15	5
	-11	58	2



Evaluation			
Obs	Softmax	Pred	Actual
1	[0.1,0.1,0.8]	[0,0,1]	[0,0,1]
2	[0.6,0.2,0.2]	[1,0,0]	[0,0,1]
3	[0.01,0.97,0.02]	[0,1,0]	[0,1,0]
...			

Accuracy = 66.67%

Cross-entropy = ???

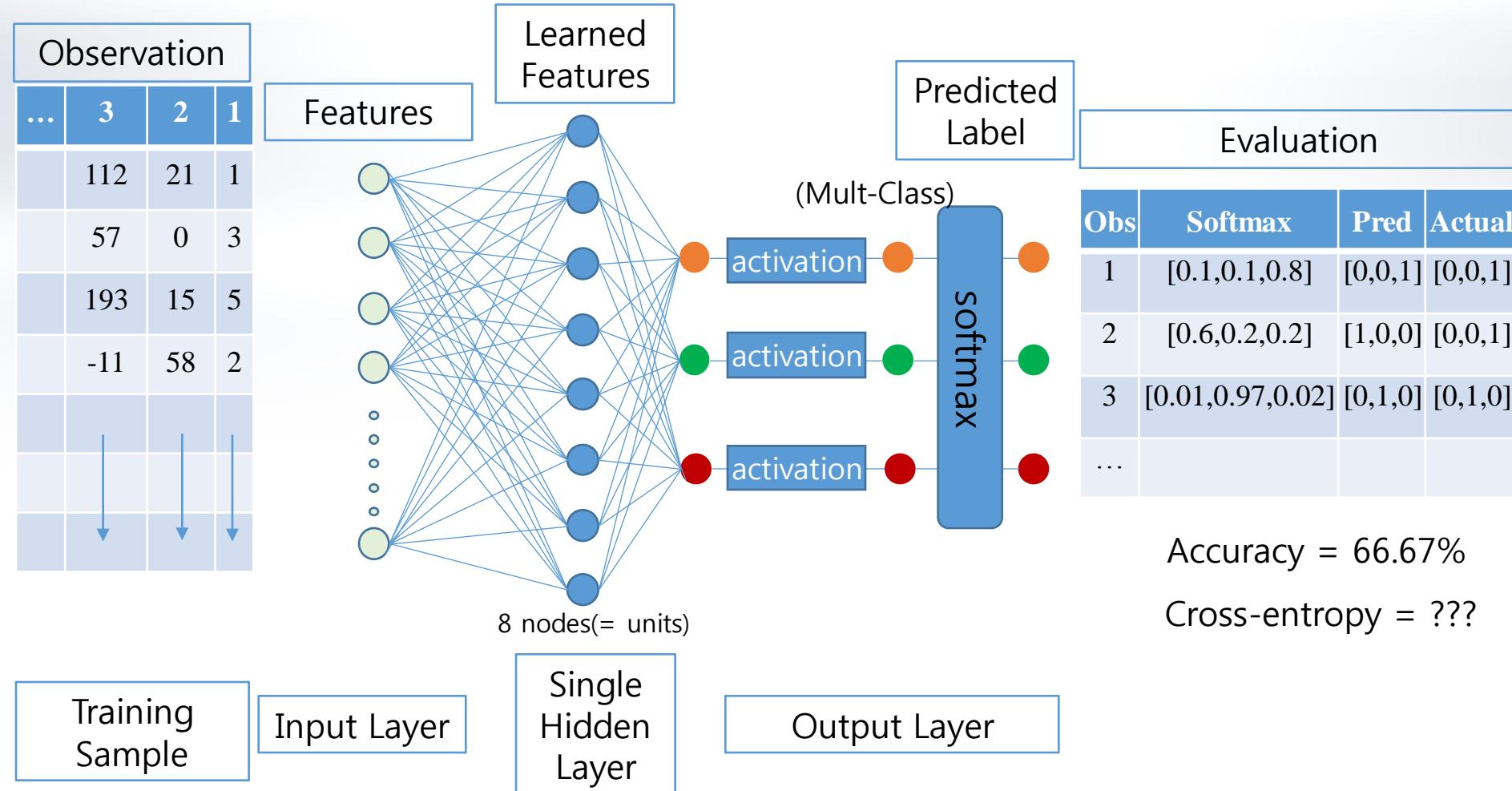
# V. Deep Learning Revisit

## 1. Perceptron

6

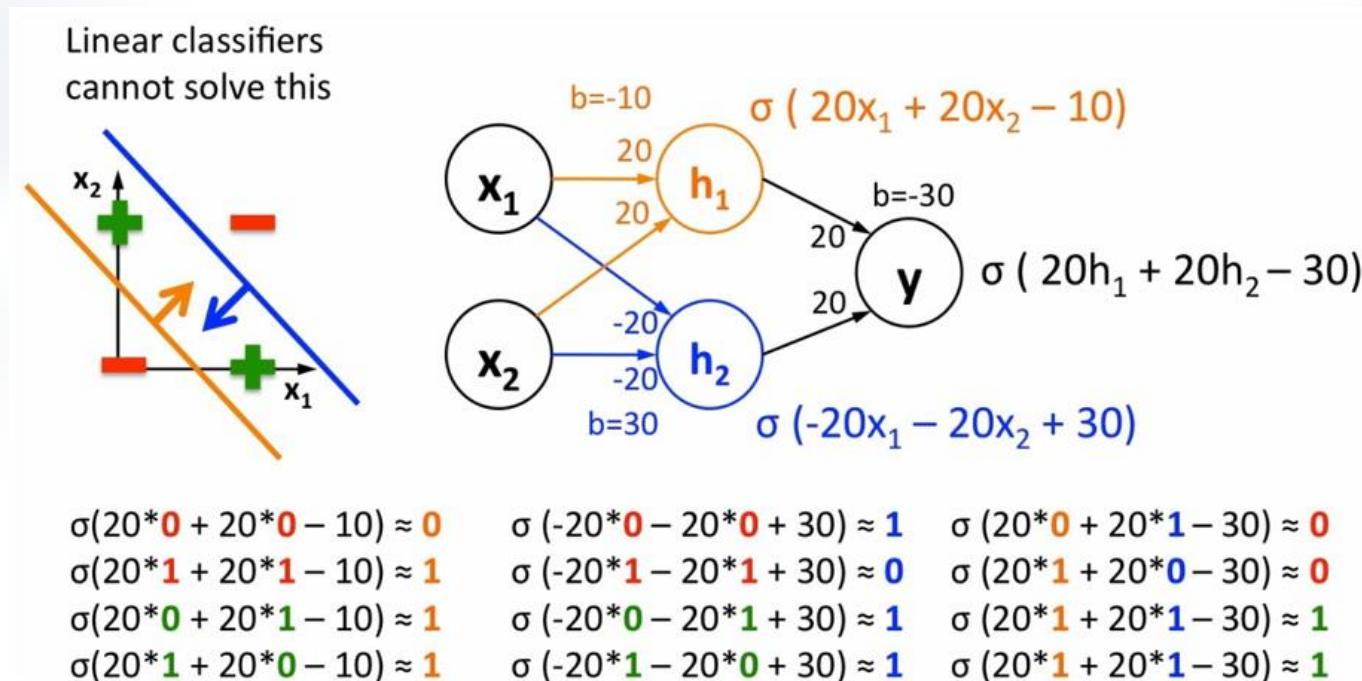
- Single Layer Perceptron > Classification

### • Case : Single Hidden Layer for Softmax Classification



## 1. Perceptron

- Single Layer Perceptron > Classification
  - **Linear Classification의 한계**

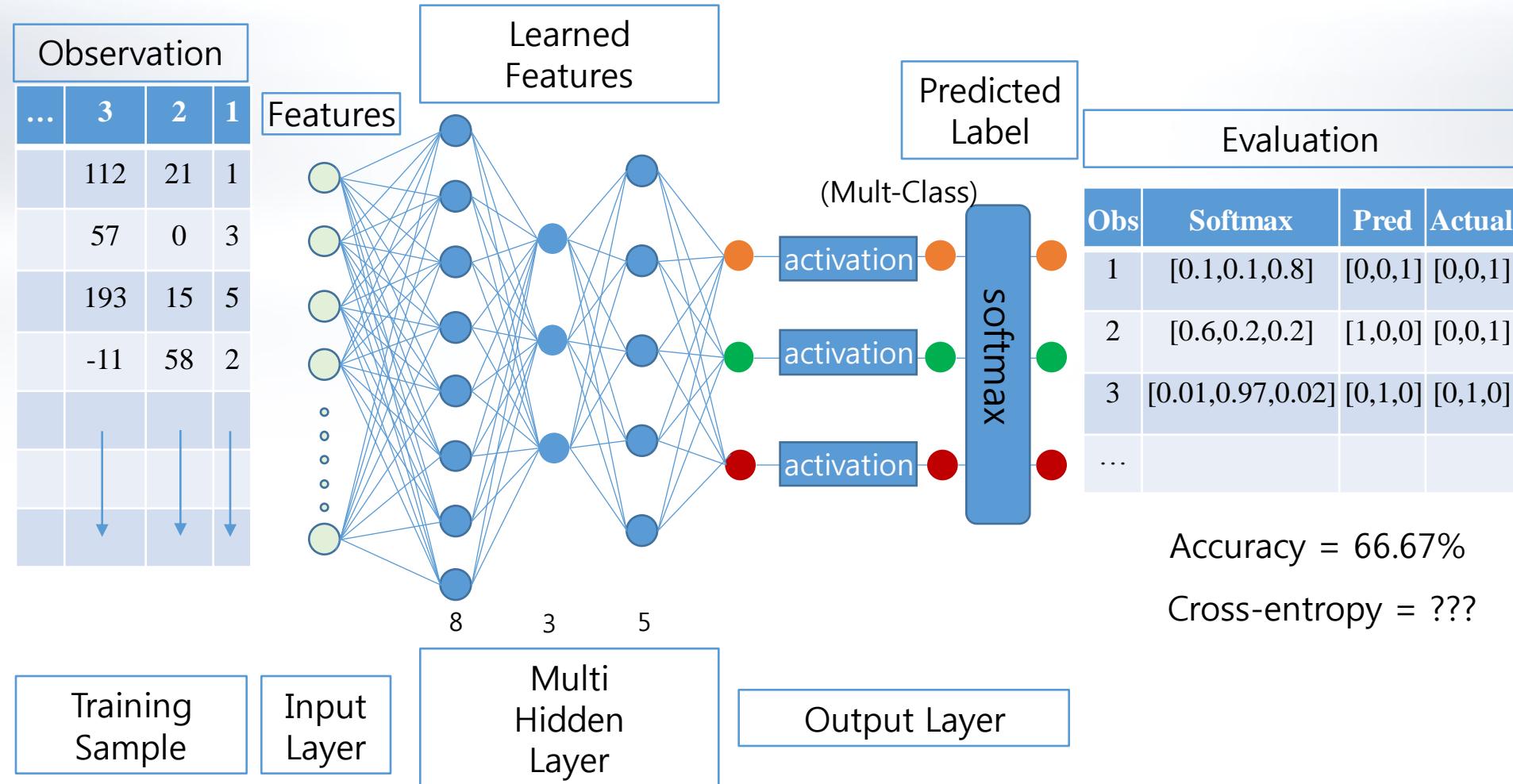


# V. Deep Learning Revisit

## 1. Perceptron

- Multi Layer Perceptron > Classification

### • Case : Multi Hidden Layer for Softmax Classification

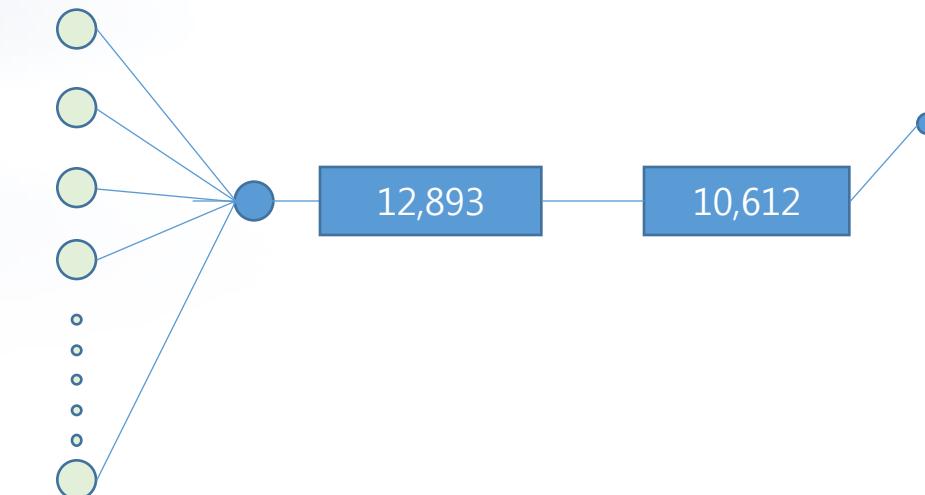


## V. Deep Learning Revisit

# 1. Perceptron

- Perceptron > Regression
    - **Case : Regression for Continuous Valued Label**

Observation			
...	3	2	1
	112	21	1
	57	0	3
	193	15	5
	-11	58	2



Evaluation			
Obs	Pred	Actual	Diff
1	12,893	10,612	2,281
2	10,002	11,322	-1,320
3	18,022	17,001	1,019
...			

MSE = ?

MAE = ?

RMSE = ?

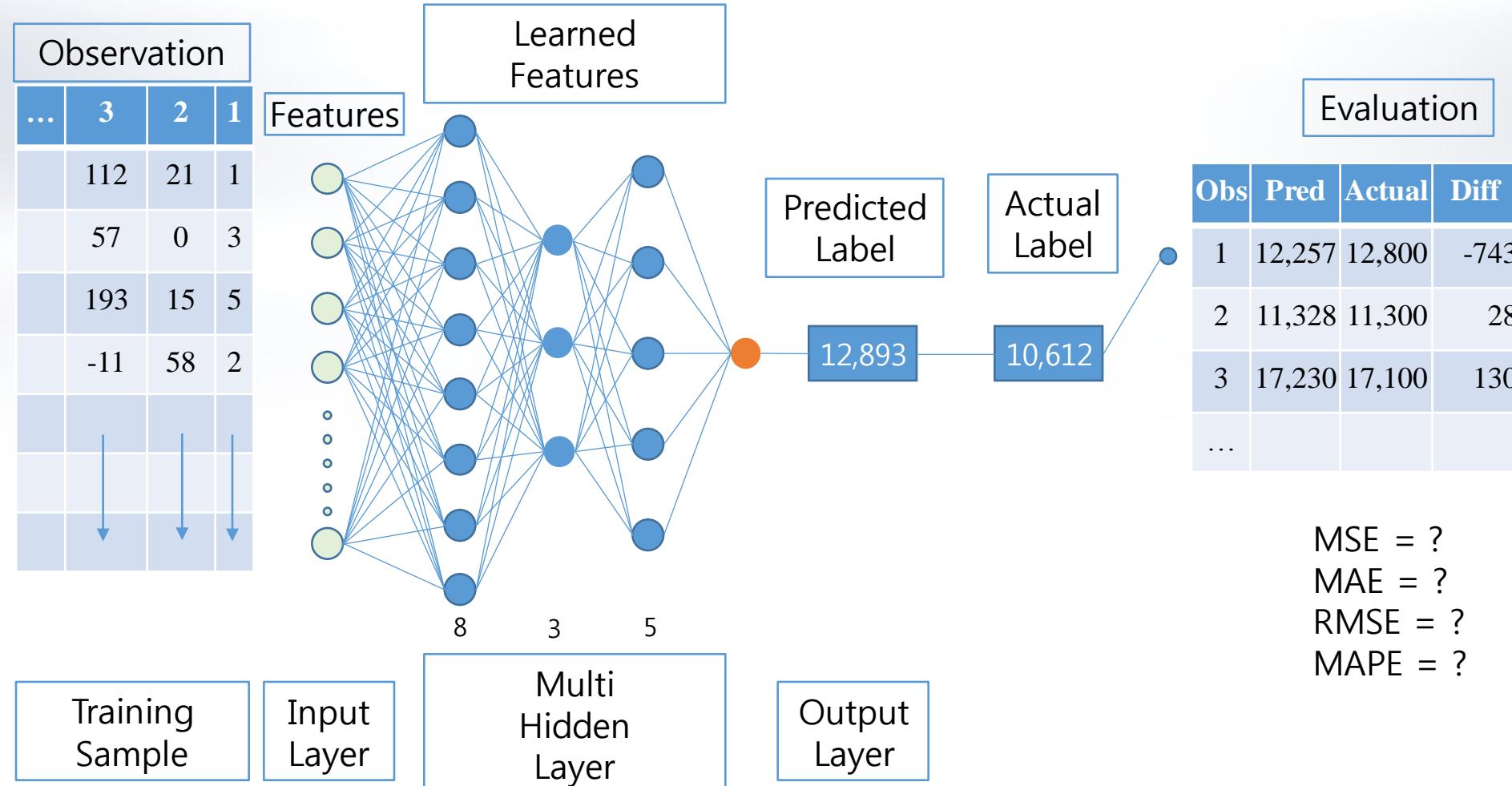
MAPE = ?

# V. Deep Learning Revisit

## 1. Perceptron

- Multi Layer Perceptron > Regression

### • Case : Multi-Layer Perceptron for Continuous Valued Label



Training Sample

Input Layer

Multi Hidden Layer

Output Layer

## 2. MLP

11

- Parameter Tuning

- 목적 : 일반화를 보증하는 오차 최소화

### 일반화

- 훈련 Sample에 의존한 예측의 문제점 → Over-Fitting
- Small Sample & Large Number of Features → Under-Fitting
- 일반화를 위한 Data 준비
  - 2 or 3 Set : Training Data, (Validation Data,) Test Data
  - Training Data → Model 수식 산출
  - Validation Data → Model Evaluation
  - Test Data → Real World Application 적용 가능성 최종 결정 및 적용할 Model 확정

### 오차 최소화 의 영향 요소

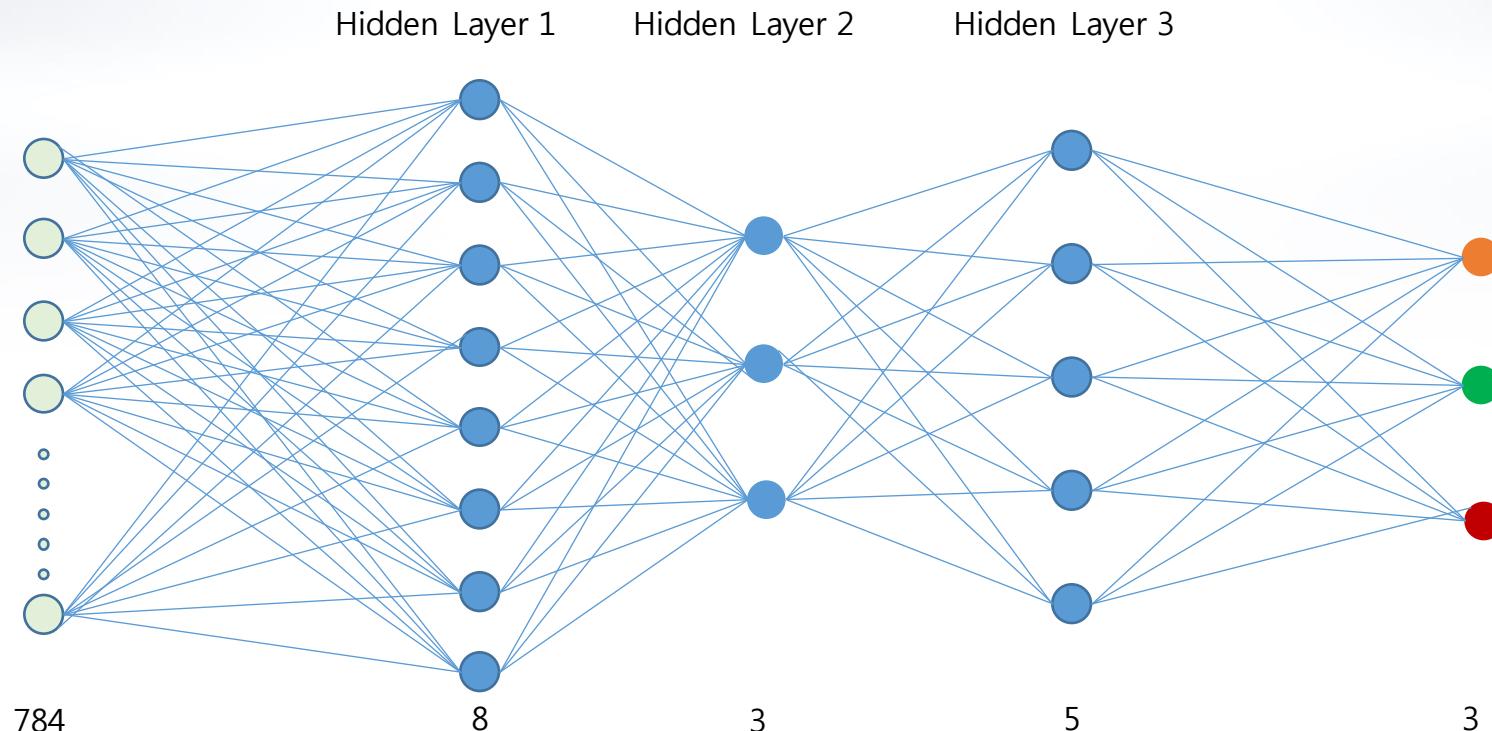
- Input Layer
  - 가중치(weight)의 변화
  - 입력변수 → 파생변수 → 특징 추출
- Hidden Layer
  - Multi-Input Variable간의 연산 → 파생변수화
  - 새로운 가중치의 연산
  - 변수들간의 관계성 또는 독립성에 대한 특징 학습
  - Activation Function 선택
- Output Layer
  - 모델의 예측값 산출
  - 현실적 기대치에 못미칠 경우 재학습 수행 – Backpropagation
  - Model Evaluation

## 2. MLP

12

- Parameters : Hidden Layers & Neurons

Parameter	내용	Case
Number of Hidden Layer	▪ 히든레이어의 수	[L1, L2, L3]
Number of Neurons per Hidden Layer	▪ 각 히든레이어의 뉴런수	[8, 3, 5]



## 2. MLP

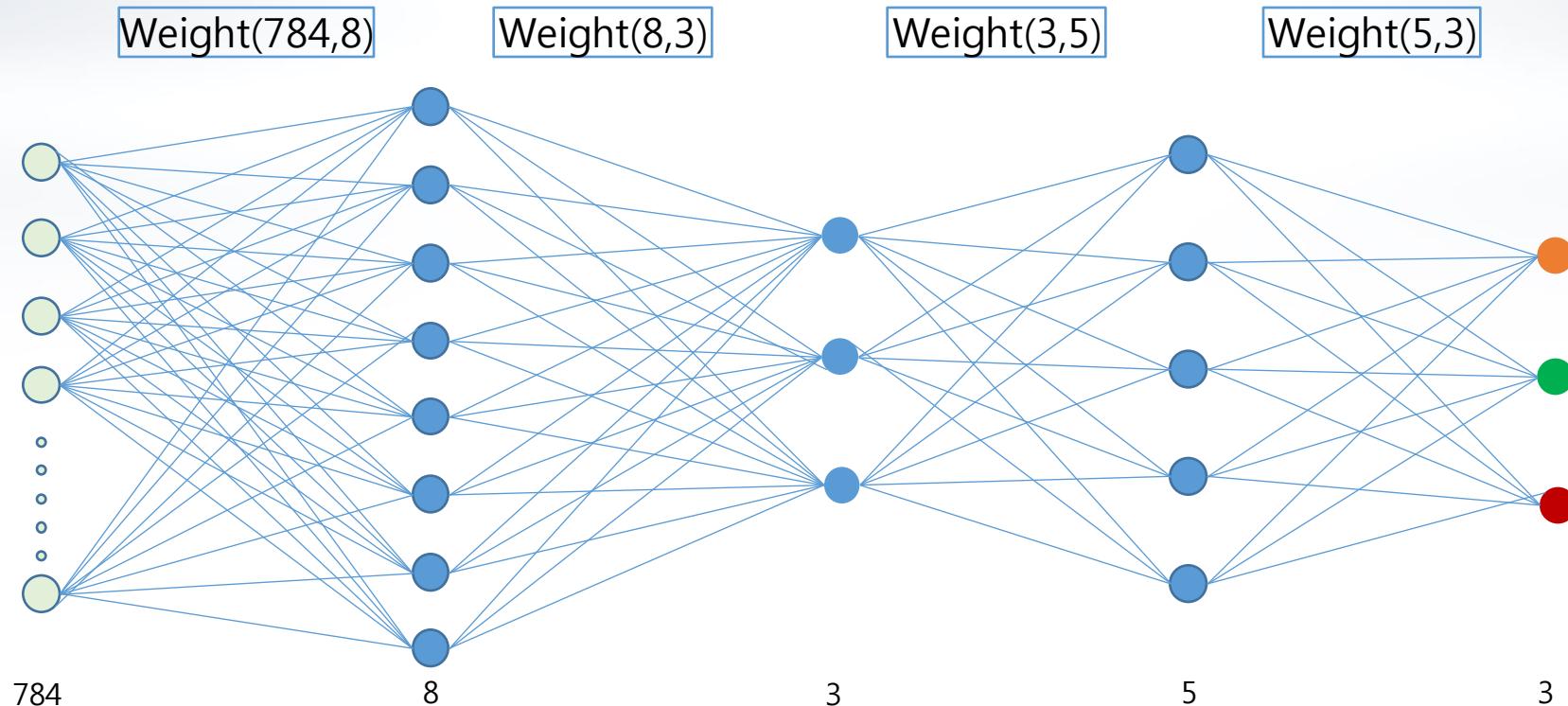
13

- Parameters : Weight

Parameter	내용
Weight Initialization	▪ 가중치 초기값 산출

### 가중치 초기화 방법

- 정규분포의 random value
- 정규분포 +/- 2시그마내 random value
- 상수값 : 예) 0, 1



## 2. MLP

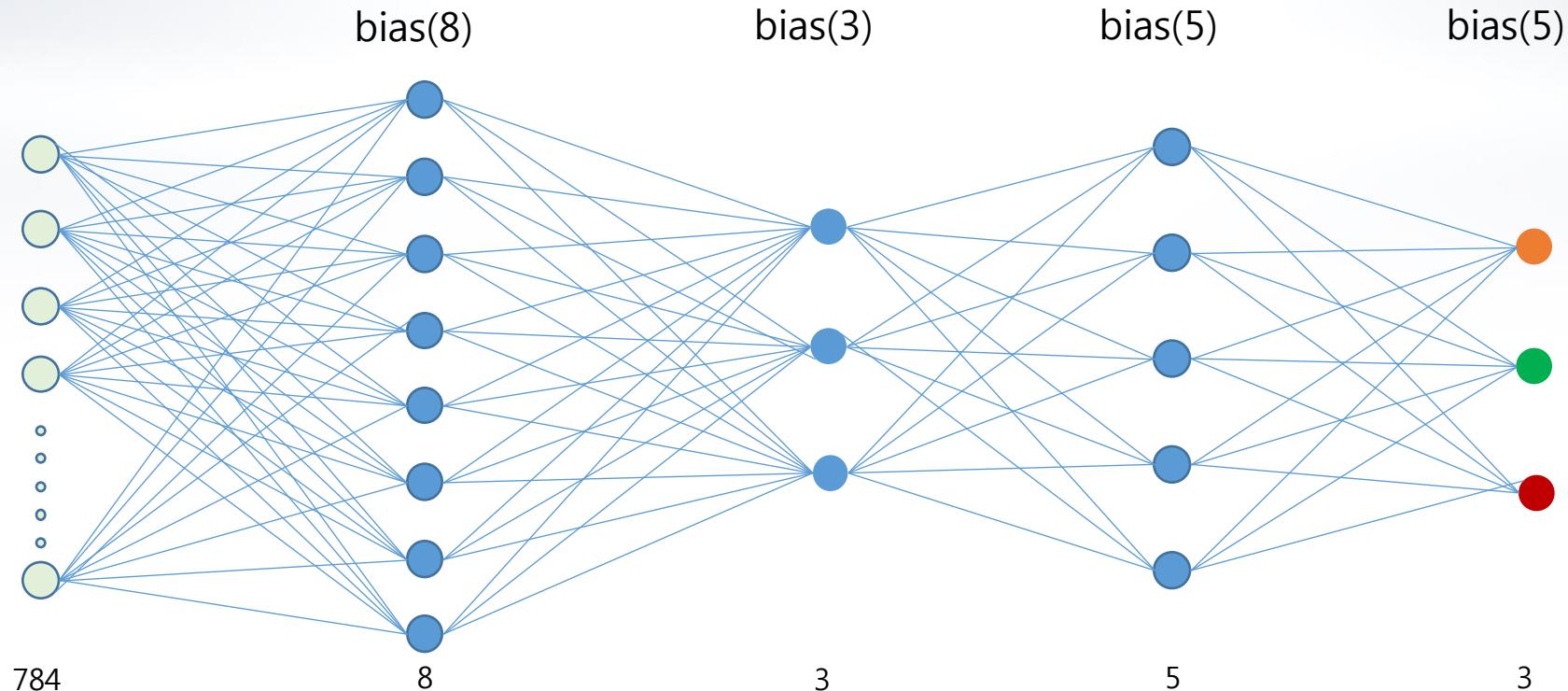
14

- Parameters : bias

Parameter	내용
bias Initialization	▪ 가중치 초기값 산출

### bias 초기화 방법

- 정규분포의 random value
- 정규분포 +/- 2시그마내 random value
- 상수값 : 예) 0, 1



# V. Deep Learning Revisit

## 2. MLP

- Parameters : Activations for Forward Propagation

Parameter	내용			
Activation function	■ 활성화함수 선택			
Name	Plot	Equation	Derivative	
Identity		$f(x) = x$	$f'(x) = 1$	
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	
Arctan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	

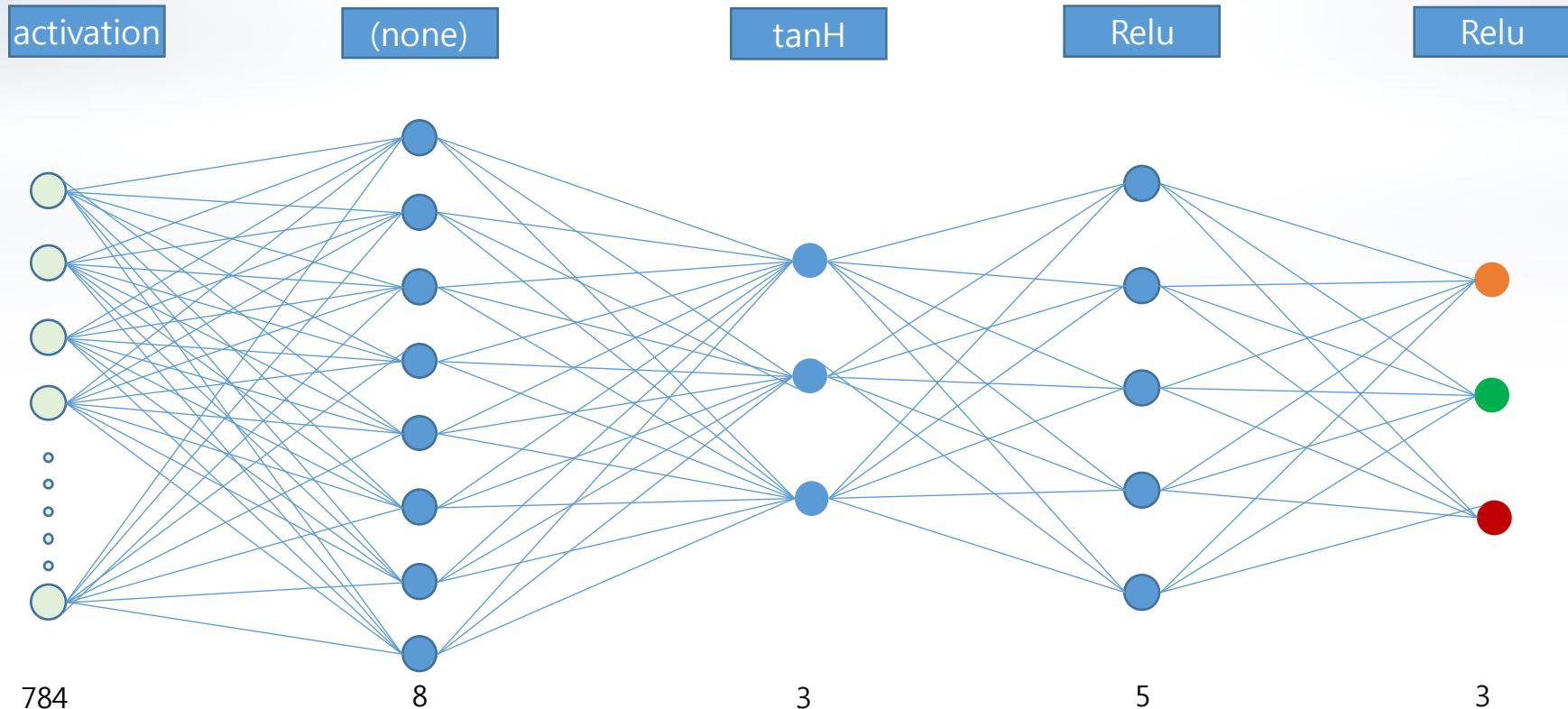
activation method

## 2. MLP

16

- Parameter for Forward Propagation

Parameter	내용
Activation function	■ 활성화함수 선택



784

8

3

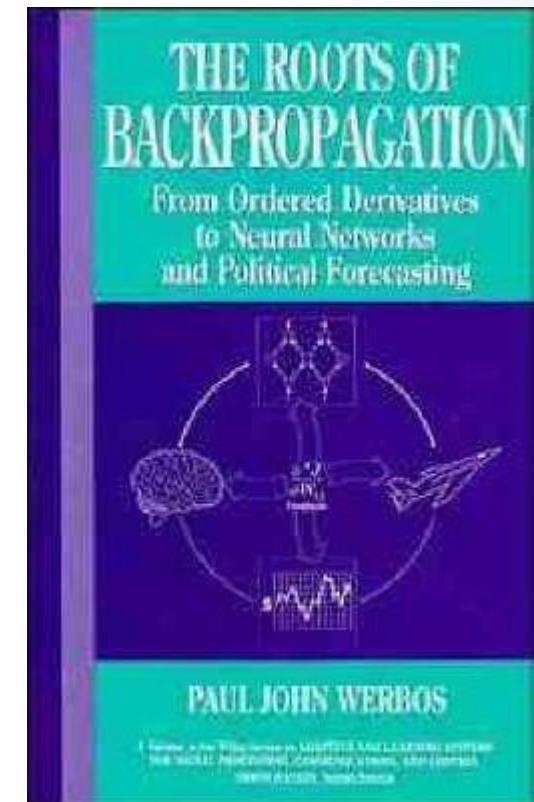
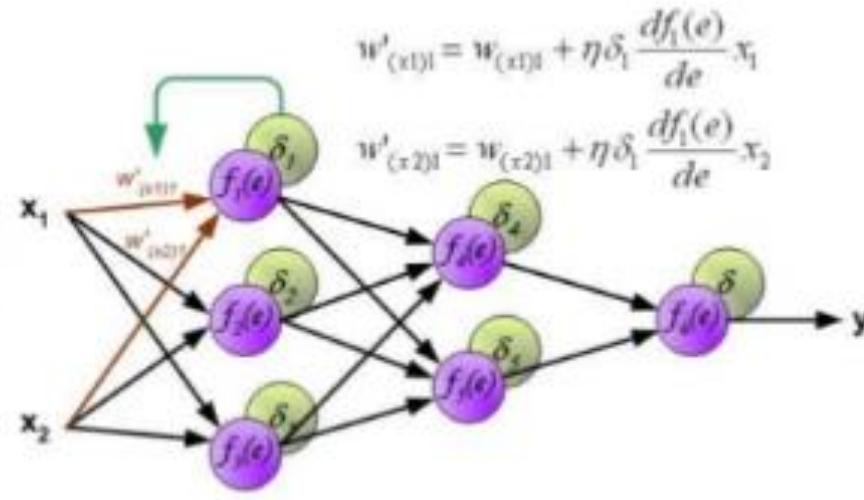
5

3

## 2. MLP

17

- Parameters : Backward Propagation Method
  - Paul Werbos(1974)의 Ph.D. thesis
  - Book : The Roots of Backpropagation



PAUL JOHN WERBOS

Author of the 1974 ground-breaking book *Neural Networks for Pattern Recognition*, and the 1994 book *Learning and Computing: Neural Network Models for Natural Language Processing, Computer Vision, and Robotics*.

# V. Deep Learning Revisit

## 2. MLP

- Parameters : Backward Propagation Method
  - Hinton(1986)
    - Paper : Learning Distributed Representations of Concepts

### Geoffrey Hinton



**Appointment**  
Advisor  
Learning in Machines & Brains

**Institution**  
University of Toronto  
Google  
Department of Computer Science

**Country**  
Canada

3

Learning distributed representations of concepts  
GEFFREY J. HINTON

This simple model's central hypothesis

This first basic intuition provides for how categorical information may be represented in neural networks. These basic ideas underlie many of today's most successful deep learning systems. They are also at the heart of many other important developments in machine learning. The two main types of distributed representations of words are called "distributed vectors" or "word embeddings". In one type of distributed vector representation, words are represented by vectors in a single space. This is similar to a learned set of shared basis functions for a single task, such as the "feature vectors" used in linear models. Feature vectors enable the same function to be learned for many different tasks. In the second type of distributed vector representation, words are represented by vectors in many different spaces. This is similar to a learned set of shared features and enables the strengths of the associations between words in one task (such as "concept" associations in a word processor) to be learned without being explicitly encoded in the shared feature space. (Fritz, Whittlesey, Beaufays, and Andrew Ng, 2009). The authors use PolyBoltzmann embeddings to measure word similarity. They find a significant number of words are encoded in a way that makes them similar to each other. These words are often used in structured meaning. They are primarily associated with the concepts humans constantly switch from one to another. This means our intuition was right: representing words in multiple spaces composed of a mixture of concepts sharing different roles within the network.

Background note:

The idea of using neural nets to implement distributed semantics of the kind above is mentioned in my introduction in Fig. 10. With hindsight I think it's remarkable how predictive we were back then!



Geoffrey Hinton

@geoffreyhinton

deep learning

가입일: 2019년 1월

트윗 12 친구 15 팔로워 33,182 마음에 들이요 2

트윗 트윗과 답글

Geoffrey Hinton @geoffreyhinton · 4월 21일  
Inman is always thought-provoking.

Walter de Back @wdeback

@geoffreyhinton 님 @NandoDF 님 와 3명에게 보내는 답글  
Great to see Inman Harvey mentioned here. I encourage all to (re)read his thought-provoking essay "Robotics: Philosophy of Mind using a Screwdriver".  
Pages 11-32 in: dspace.library.nu/bitsream/hand...

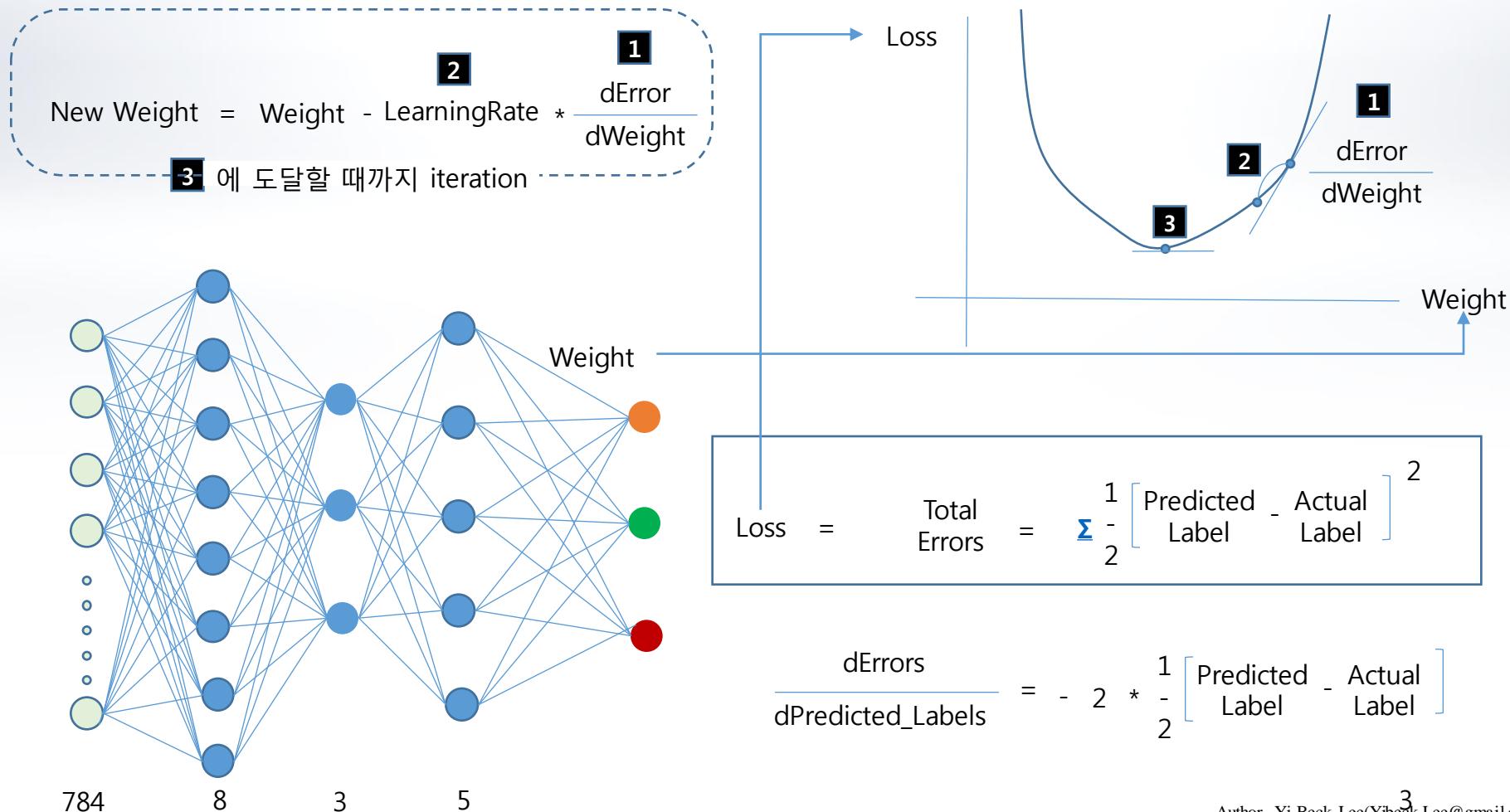
0 2 33 159

# V. Deep Learning Revisit

## 2. MLP

19

- Parameters : Backward Propagation Method
  - 전방위 학습과 반대 방향으로 거슬러 올라가기
  - Weight Update - learning rate를 조정 계수로 하여 오차가 최소화할 때 가지 가중치 갱신

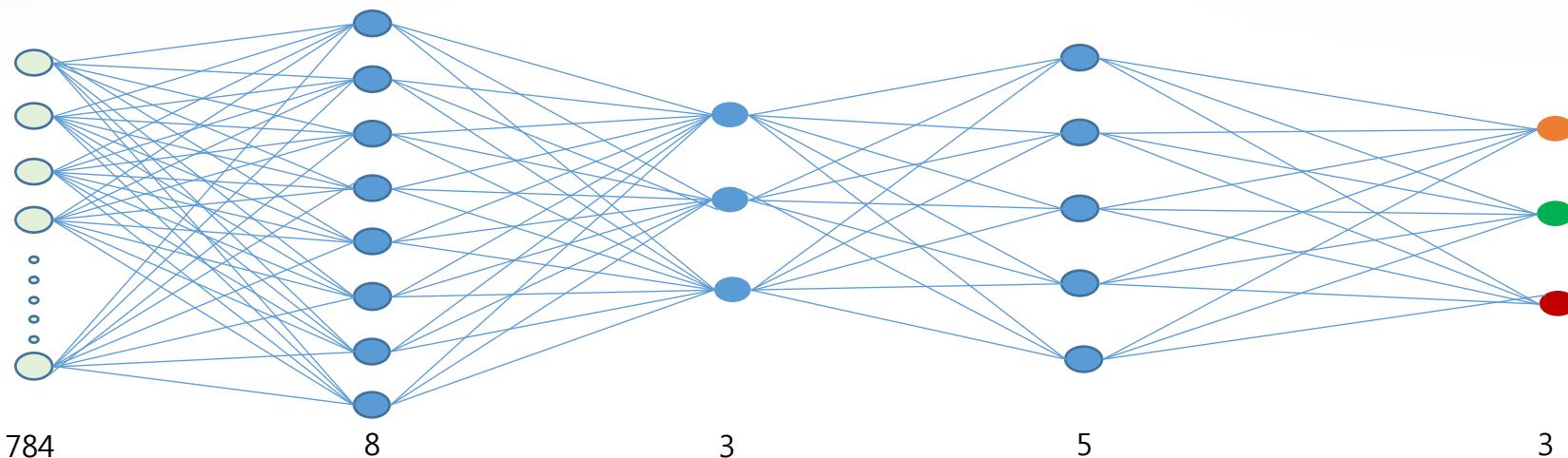
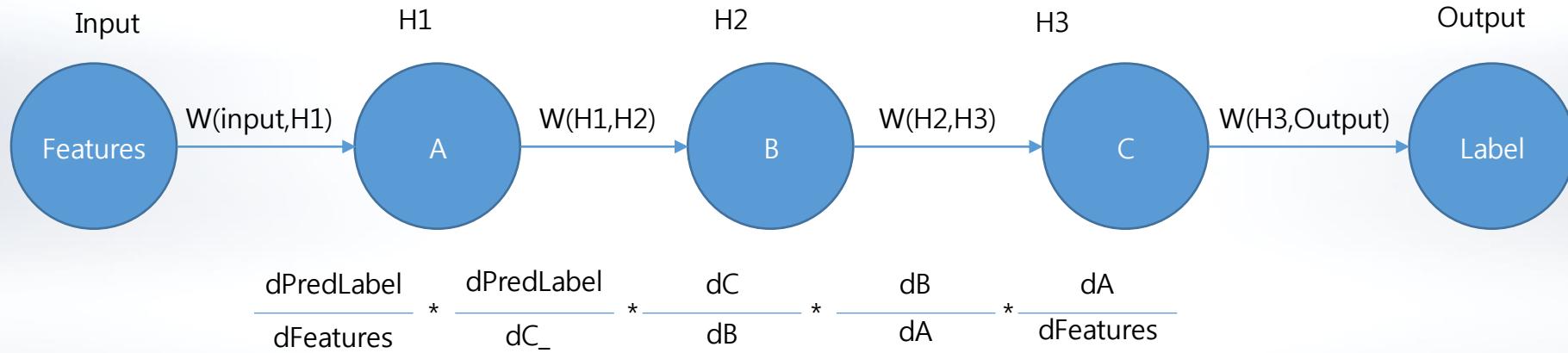


## V. Deep Learning Revisit

## 2. MLP

20

- Parameters : Backward Propagation Method
    - 전방위 학습과 반대 방향으로 거슬러 올라가기

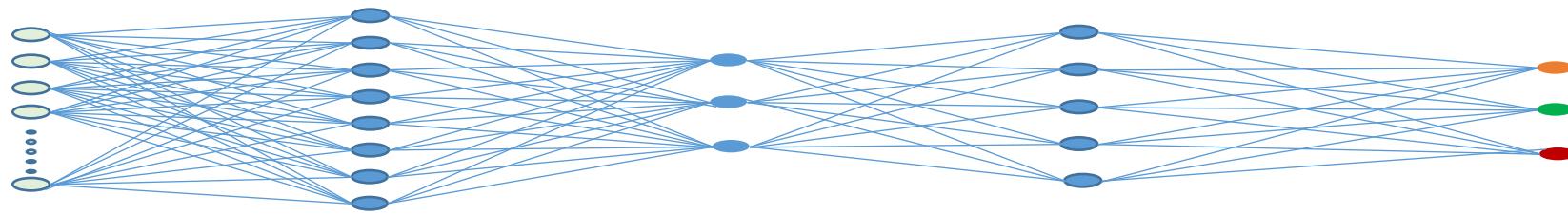
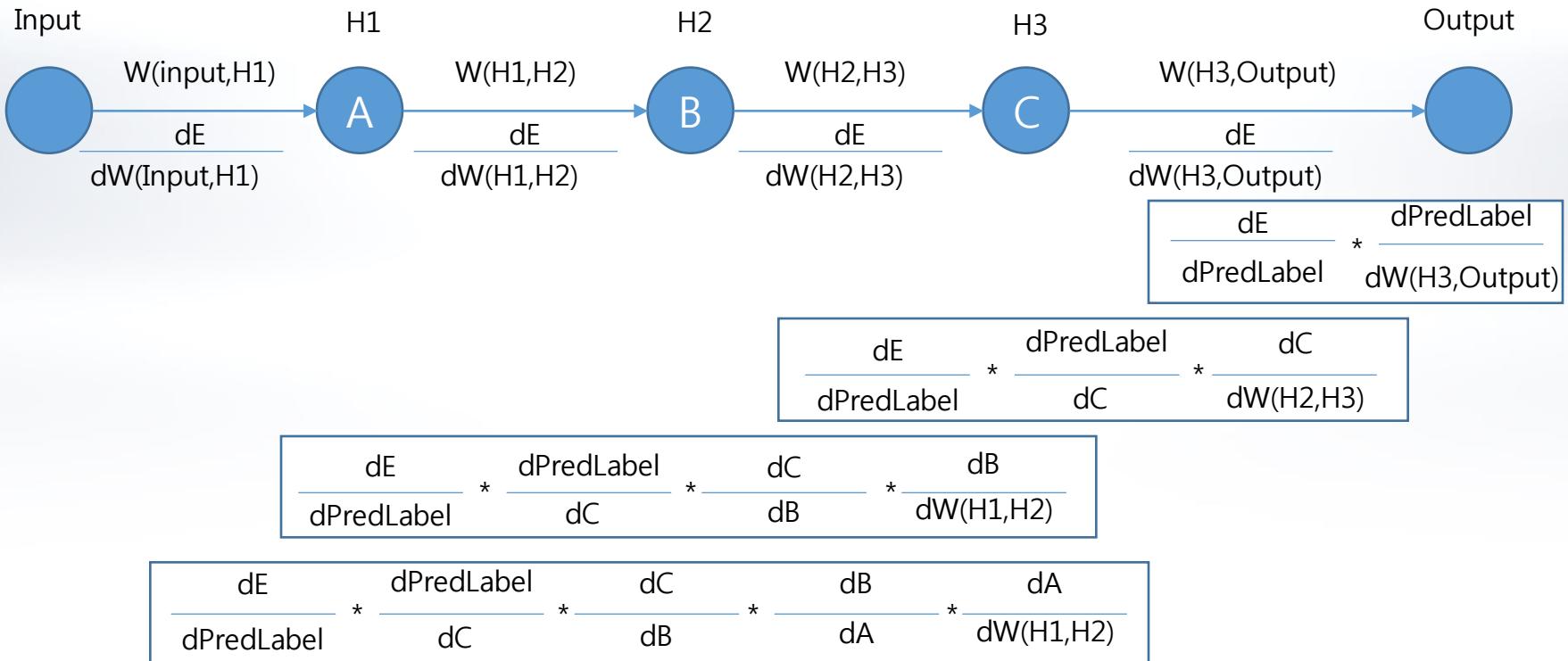


# V. Deep Learning Revisit

## 2. MLP

21

- Parameters : Backward Propagation Method
  - 전방위 학습과 반대 방향으로 거슬러 올라가기



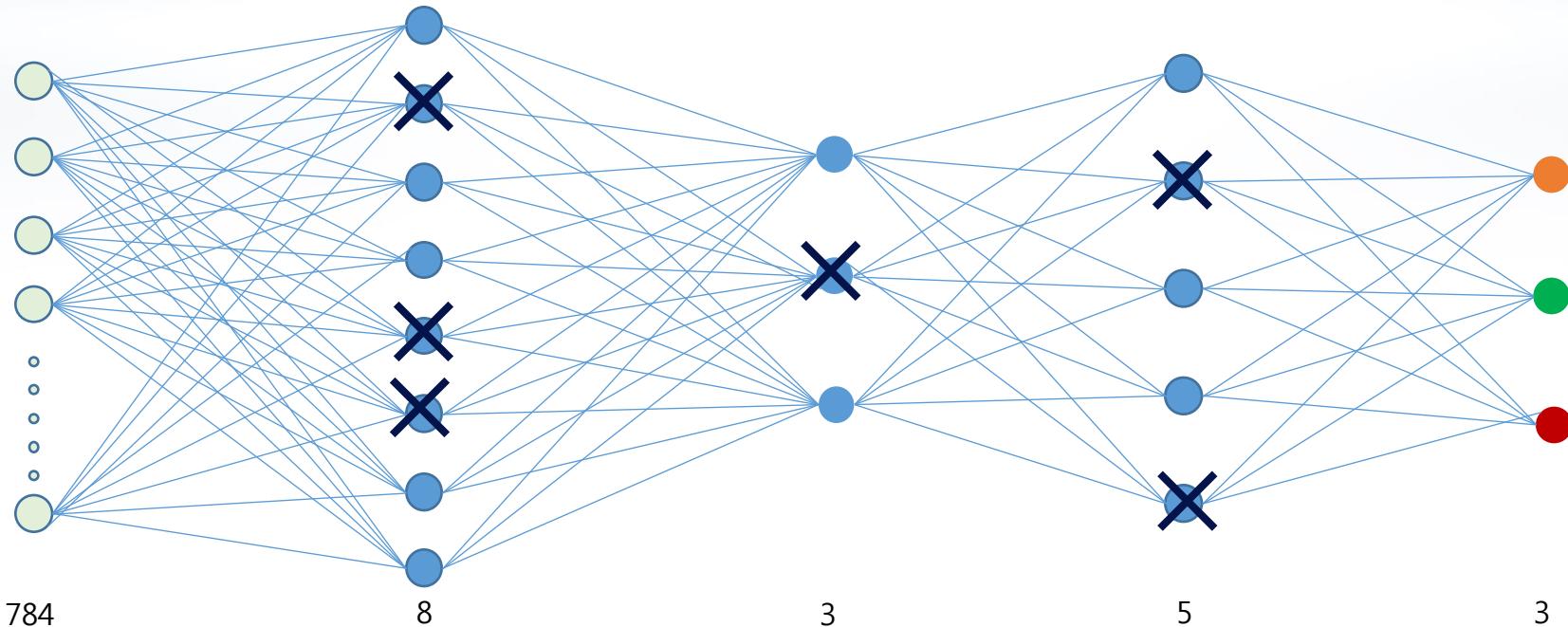
## 2. MLP

22

- Parameters : Regularization > Dropout
  - 학습을 덜 하겠다는 의미 → 과적합 방지

Parameter	내용
Dropout Rate	▪ 가중치 탈락 비율

- | Regularization Method                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>L1 Regularization → Lasso</li> <li>L2 Regularization → Ridge</li> <li>Drop-Out → Deep Learning</li> </ul> |



784

8

3

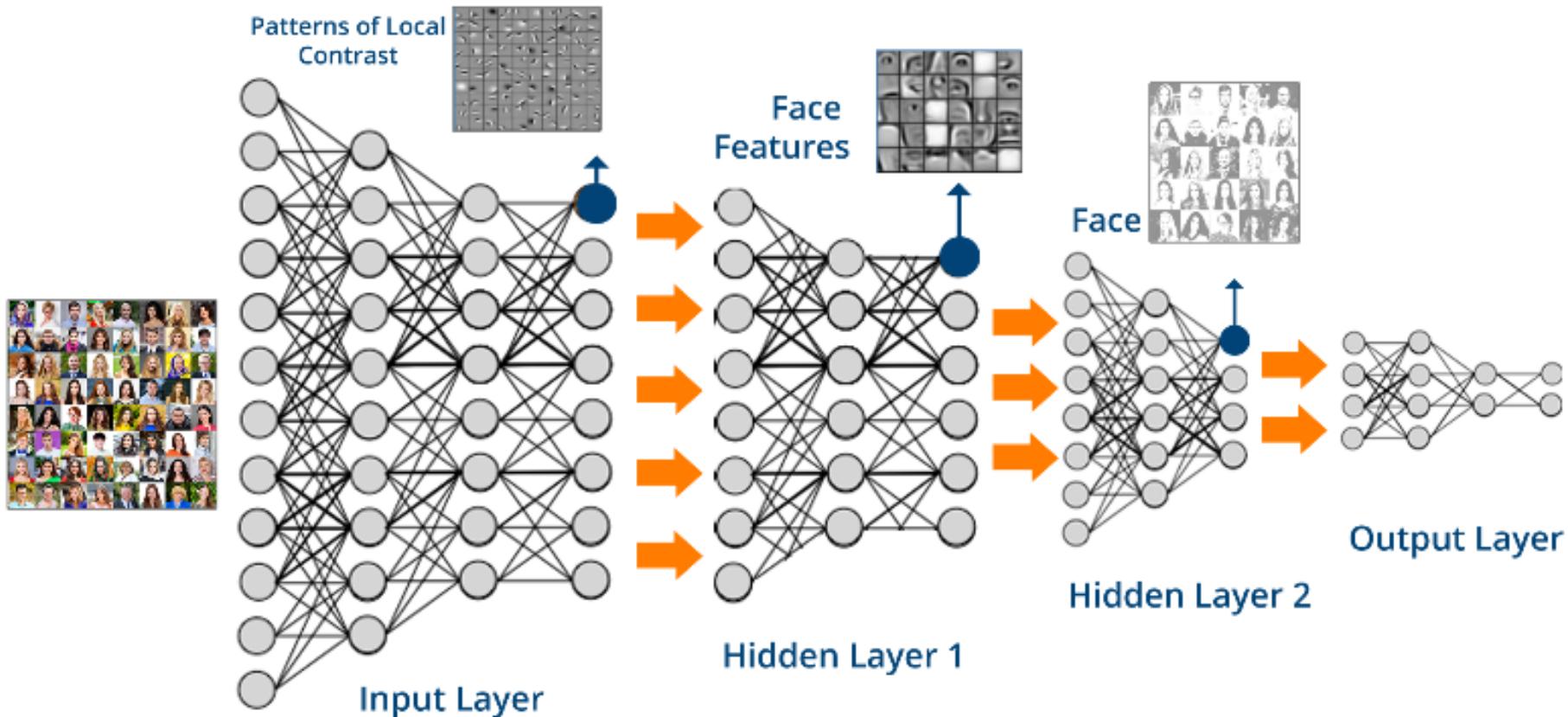
5

3

## 2. MLP

23

- Hidden Layer를 거치면서 Label에 적합하는 특징 패턴을 추출하는 과정



# V. Deep Learning Revisit

## 2. MLP

### ▪ Activation

```
1 #!/usr/bin/env /c/Apps/Anaconda3/python
2
3 # Import
4 import tensorflow as tf
5 import numpy as np
6 import pandas as pd
7 from sklearn.preprocessing import MinMaxScaler
8 import matplotlib.pyplot as plt
9
10 # Import data
11 data = pd.read_csv('./data_stocks.csv')
12
13 # Drop date variable
14 data = data.drop(['DATE'], 1)
15
16 # Dimensions of dataset
17 n = data.shape[0]
18 p = data.shape[1]
19
20 # Make data a np.array
21 data = data.values
22
23 # Training and test data
24 train_start = 0
25 train_end = int(np.floor(0.8*n))
26 test_start = train_end + 1
27 test_end = n
28 data_train = data[np.arange(train_start, train_end), :]
29 data_test = data[np.arange(test_start, test_end), :]
30
31 # Scale data
32 scaler = MinMaxScaler(feature_range=(-1, 1))
33 scaler.fit(data_train)
34 data_train = scaler.transform(data_train)
35 data_test = scaler.transform(data_test)
36
37 # Build X and y
38 X_train = data_train[:, 1:]
39 y_train = data_train[:, 0]
40 X_test = data_test[:, 1:]
```

```
41 y_test = data_test[:, 0]
42
43 # Number of stocks in training data
44 n_stocks = X_train.shape[1]
45
46 # Neurons
47 n_neurons_1 = 1024
48 n_neurons_2 = 512
49 n_neurons_3 = 256
50 n_neurons_4 = 128
51
52 # Session
53 net = tf.InteractiveSession()
54
55 # Placeholder
56 X = tf.placeholder(dtype=tf.float32, shape=[None, n_stocks])
57 Y = tf.placeholder(dtype=tf.float32, shape=[None])
58
59 # Hidden layer
60 W_hidden_1 = tf.Variable(tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform"))
61 b_hidden_1 = tf.Variable(tf.zeros_initializer())
62
63 # Output layer
64 W_out = tf.Variable(tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform"))
65 b_out = tf.Variable(tf.zeros_initializer())
66
67 # Hidden layer
68 W_hidden_2 = tf.Variable(tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform"))
69 b_hidden_2 = tf.Variable(tf.zeros_initializer())
70
71 # Hidden layer
72 W_hidden_3 = tf.Variable(tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform"))
73 b_hidden_3 = tf.Variable(tf.zeros_initializer())
74
75 # Output layer
76 W_out = tf.Variable(tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform"))
77 b_out = tf.Variable(tf.zeros_initializer())
78
79 # Hidden layer
80 hidden_1 = tf.nn.relu(tf.add(tf.matmul(X, W_hidden_1), b_hidden_1))
81 hidden_2 = tf.nn.relu(tf.add(tf.matmul(hidden_1, W_hidden_2), b_hidden_2))
```

실습 중 배포

## 3. CNN

25

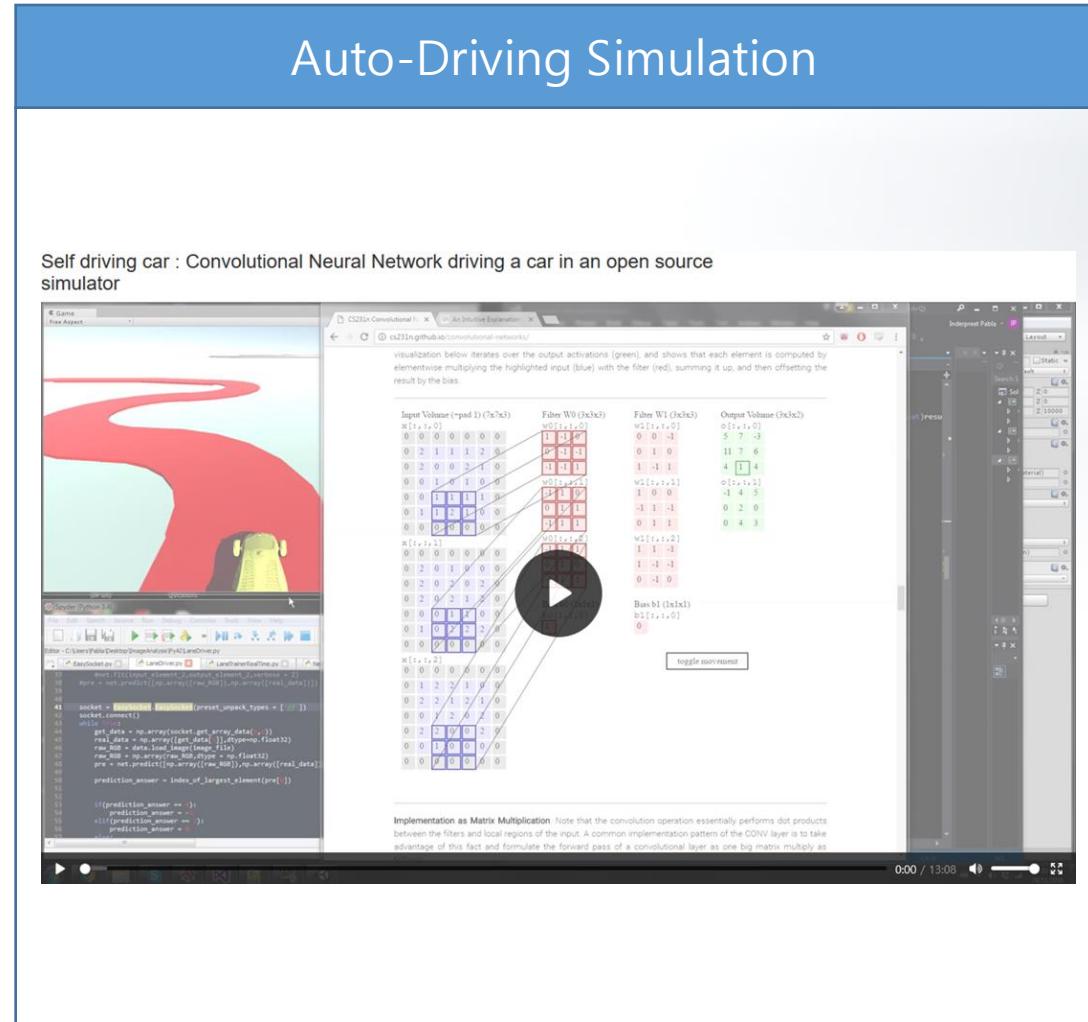
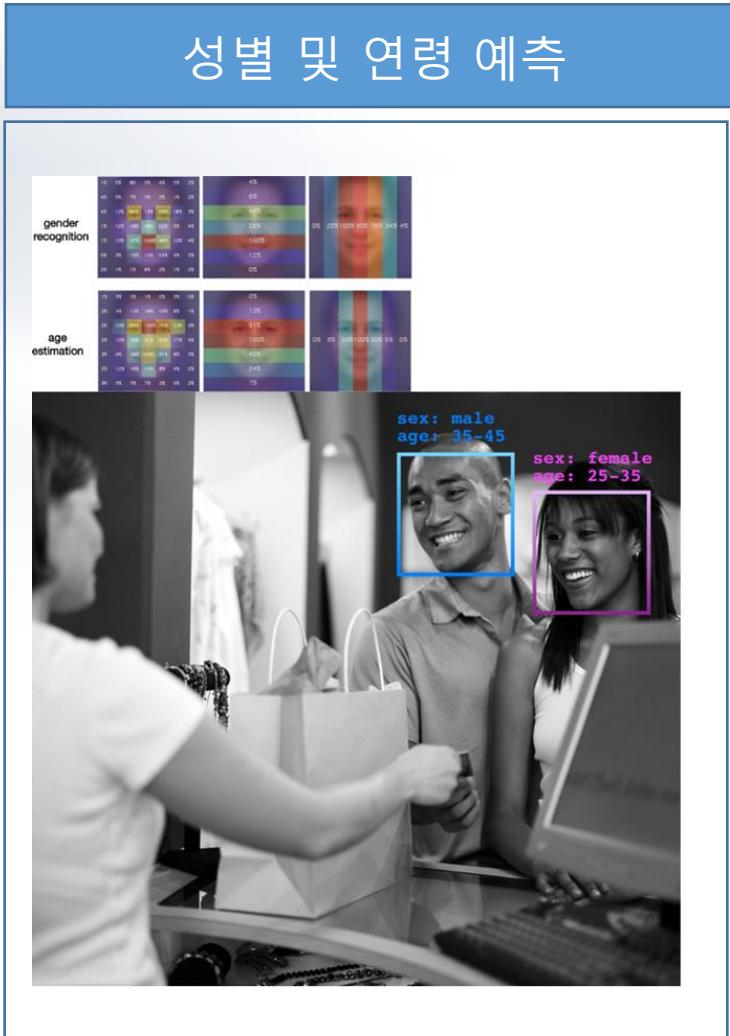
- Convolutional Neural Network(Yann LeCun, 1998)
- Convolution의 의미
  - 함수의 합성곱
- Input Layer → Convolution Layer
  - Reshape flattened input vector to 3d tensor
    - 3d tensor = [height, width, depth]
    - depth = [Red, Green, Blue] = 3 channel
    - ✓ [255, 0, 0] → Red = 1 Channel
- Convolution → ReLU → Max Pooling
  - Convolution
    - Stack of filtered images(여과지를 이동하면서 투영된 부분이미지의 적재)
      - ✓ 부분이미지 적재 → Feature Map을 만들어가는 과정
    - Kernel Filtering은 Forwarding과 동일한 효과
    - Stride → Padding → Activation(ReLU)
    - ReLU의 역할 : Normalizing
  - Pooling
    - Filtering 과정에서 적재한 특징 배열의 개별값들중 중 최대값만 취함
      - ✓ 최대값 만 선택적으로 취함 = down sampling
    - Stack을 축소(shrinking)시켜 큰 특징만 취하는 효과
      - ✓ Window, Stride, Max
- Fully Connection
  - Why? 신경망 학습 → MLP 와 동일

## V. Deep Learning Revisit

### 3. CNN

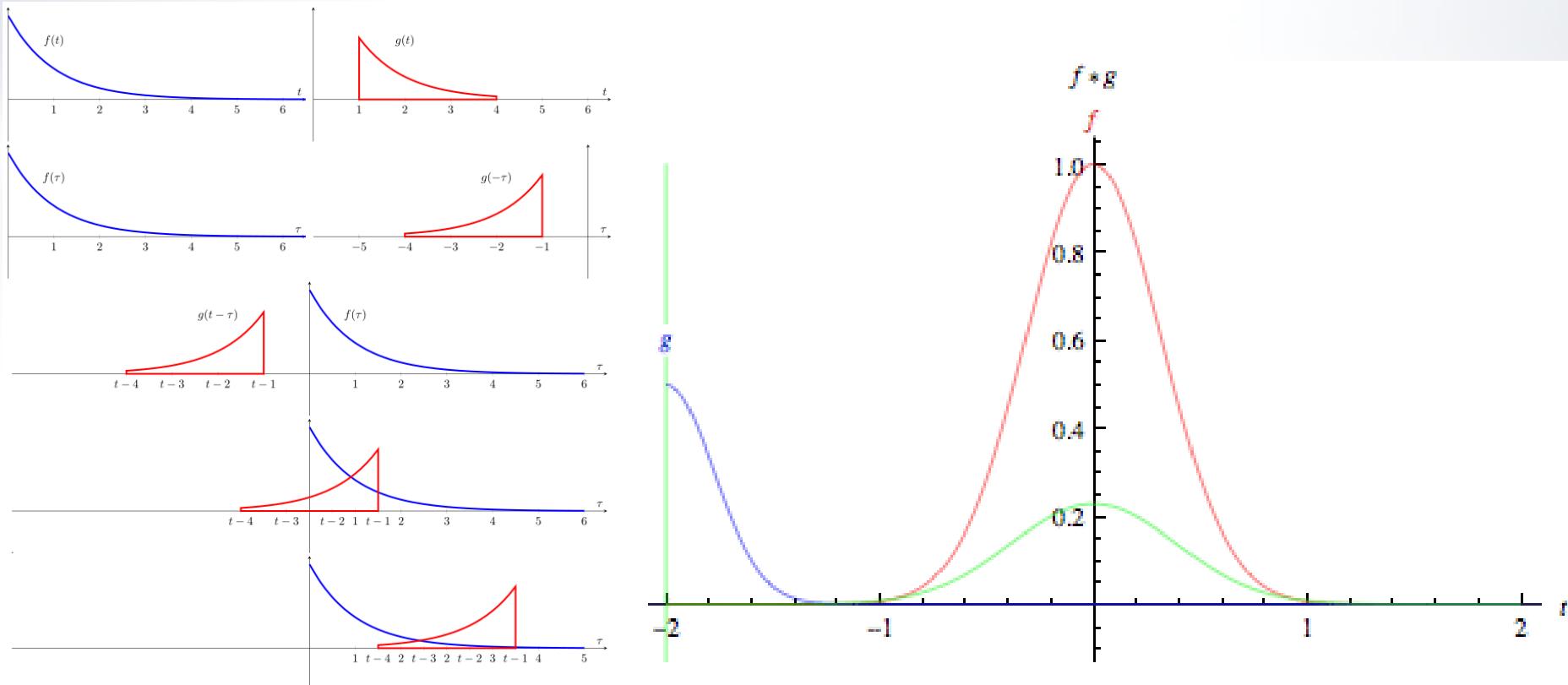
26

- #### ▪ CNN의 Output Image



## 3. CNN

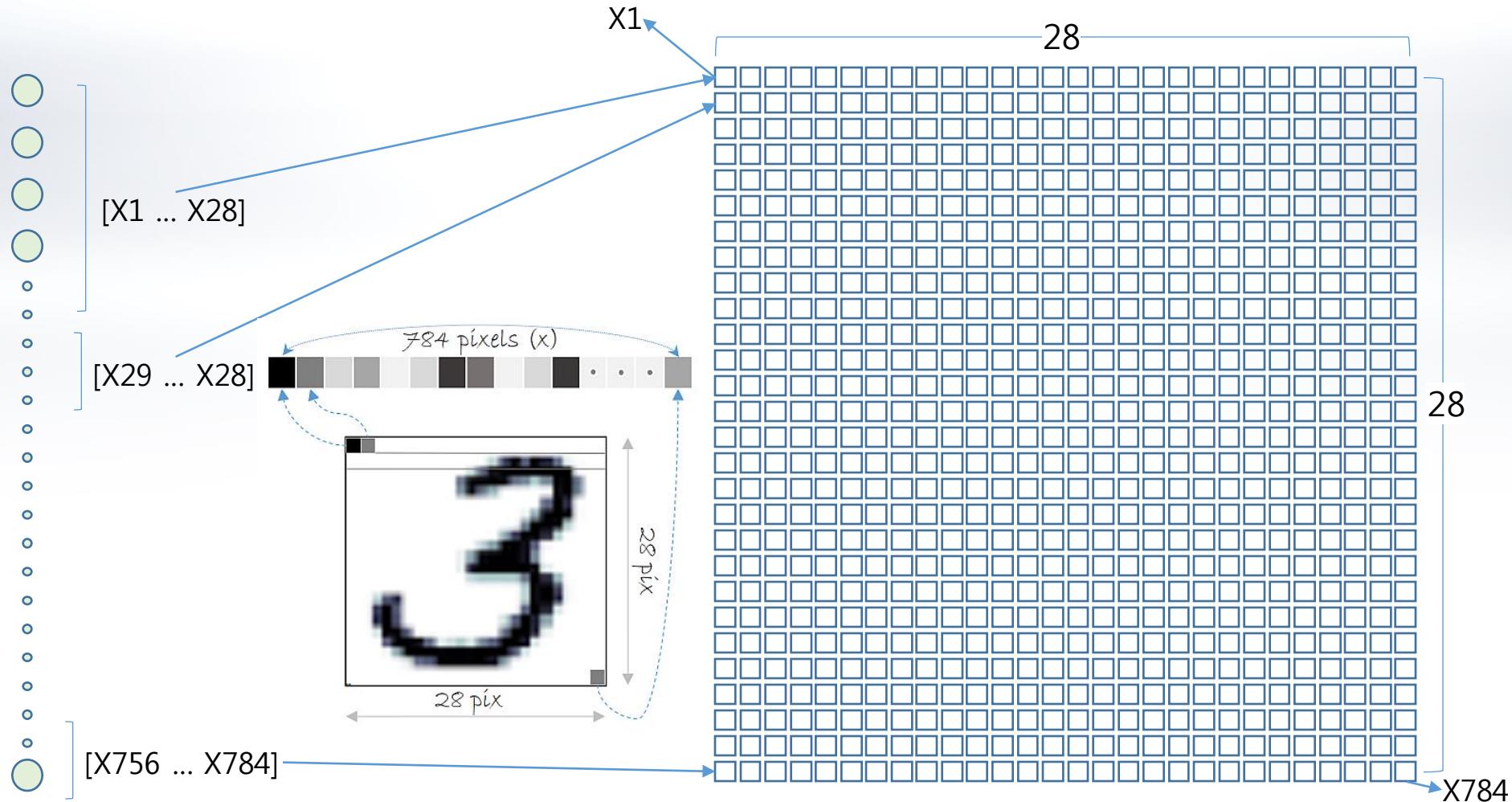
- Convolution
  - Roll Together(라틴어에서 파생)
  - Mathematics
    - Convolution은 2개의 함수를 겹쳐서 적분값을 측정(합성곱)



# V. Deep Learning Revisit

## 3. CNN

- Convolution
  - Reshape : Vector → Matrix

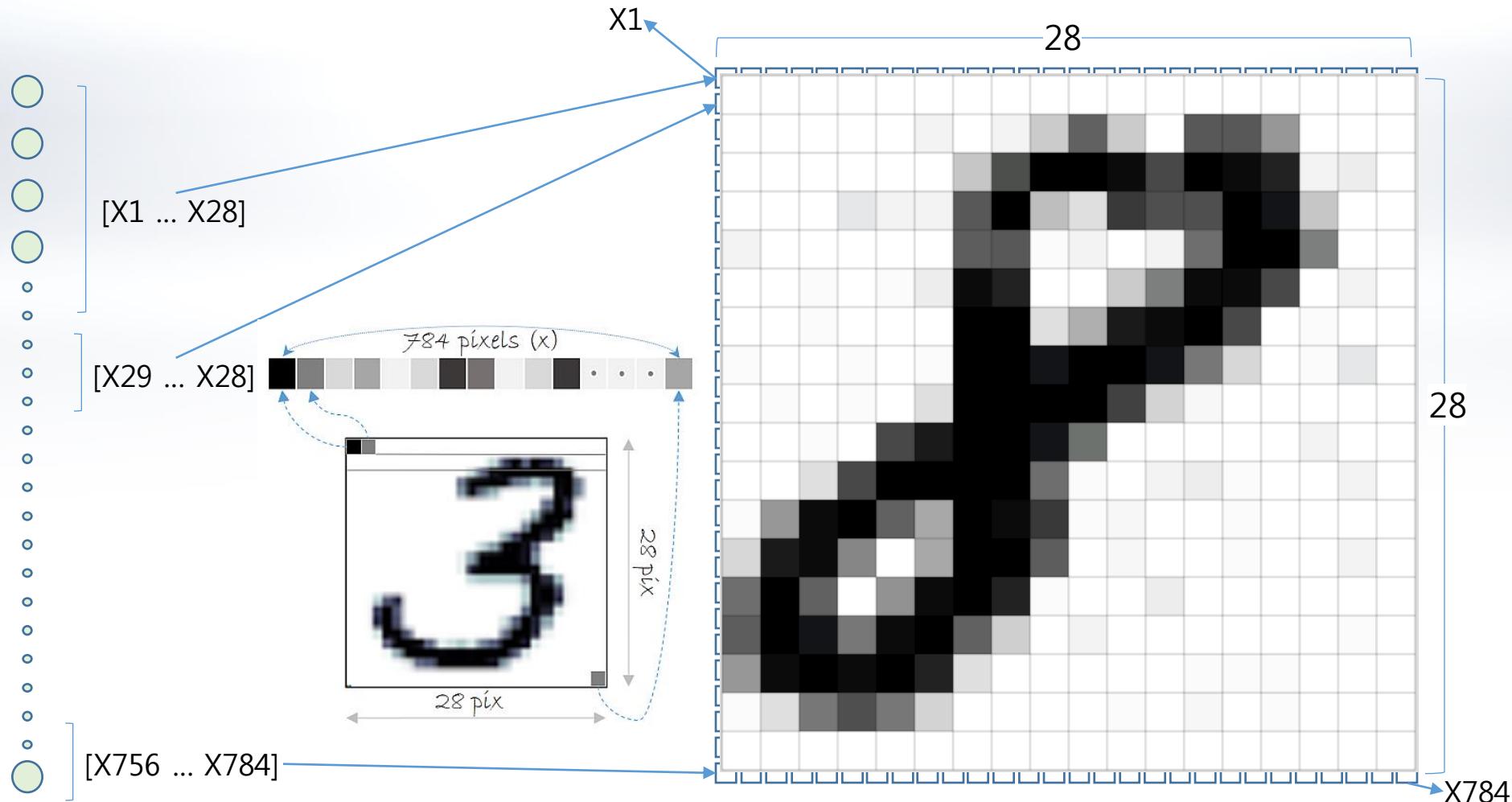


# V. Deep Learning Revisit

## 3. CNN

29

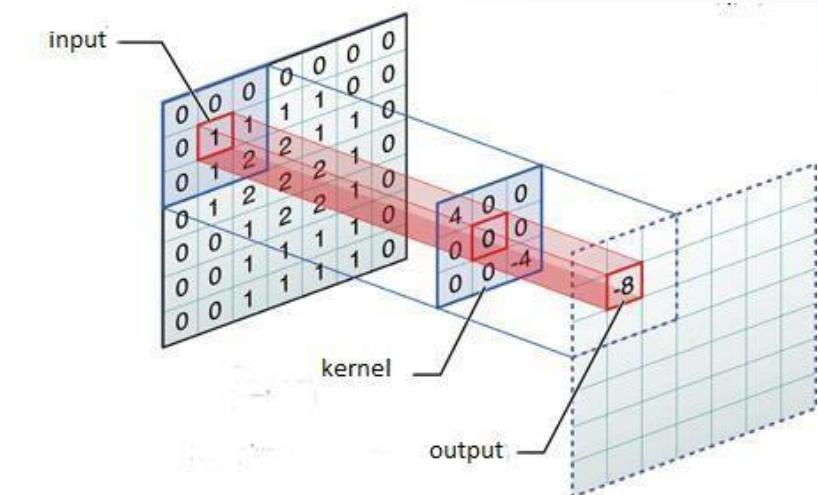
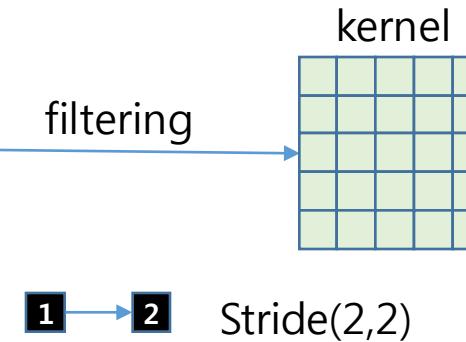
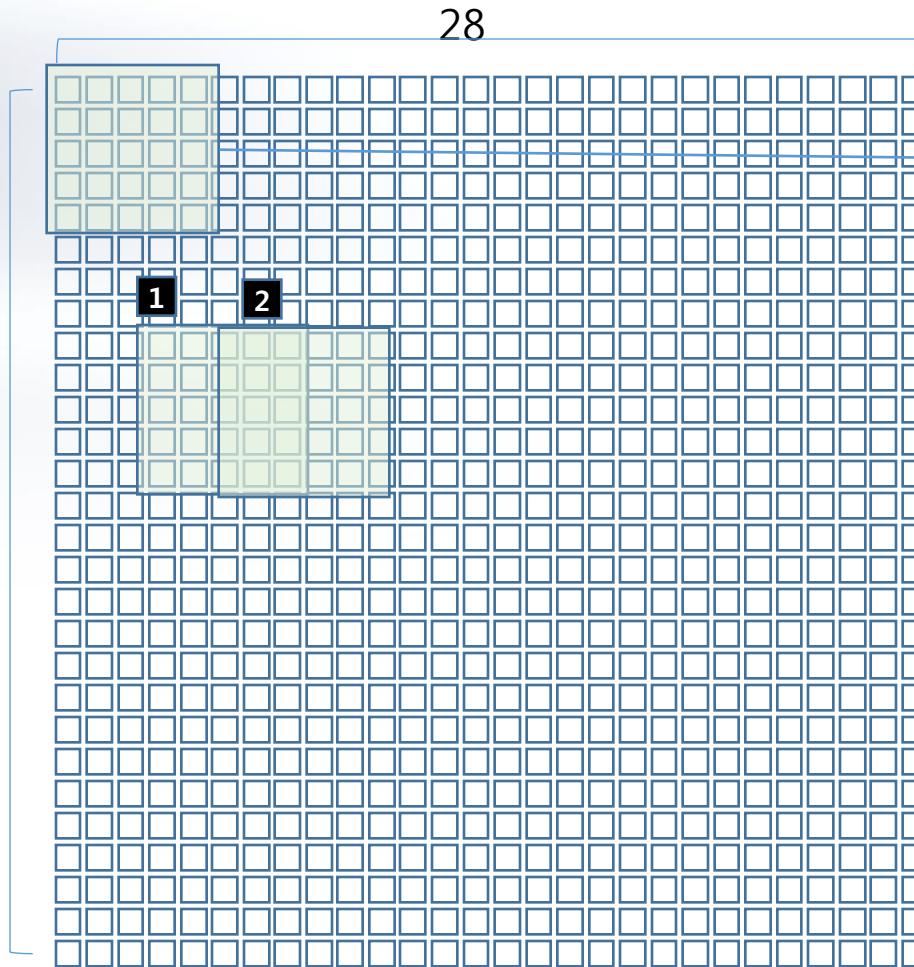
- Convolution
  - Reshape : Vector → Matrix



## 3. CNN

30

- Convolution
  - Matrix의 부분 집합 선택하여 Filtering



## 3. CNN

31

- Filter Size의 결정
  - Filter Size가 클 경우 shrunk
  - Recommended
    - $(F - 1) / 2$
    - Zero padding
    - One or Two stride

Zero Padding

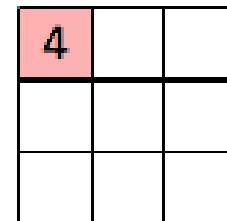
$$(A_3 - 1) / 2 = 1$$

$$(A_3 - 1) / 2 = 1$$

Without Padding  
Stride [1,1,1,1]

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image



Convolved  
Feature

Zero Padding  
Stride[1,2,2,1]

0 <sub>2</sub>	0 <sub>0</sub>	0 <sub>1</sub>	0	0	0	0
0 <sub>1</sub>	2 <sub>0</sub>	2 <sub>0</sub>	3	3	3	0
0 <sub>0</sub>	0 <sub>1</sub>	1 <sub>1</sub>	3	0	3	0
0	2	3	0	1	3	0
0	3	3	2	1	2	0
0	3	3	0	2	3	0
0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

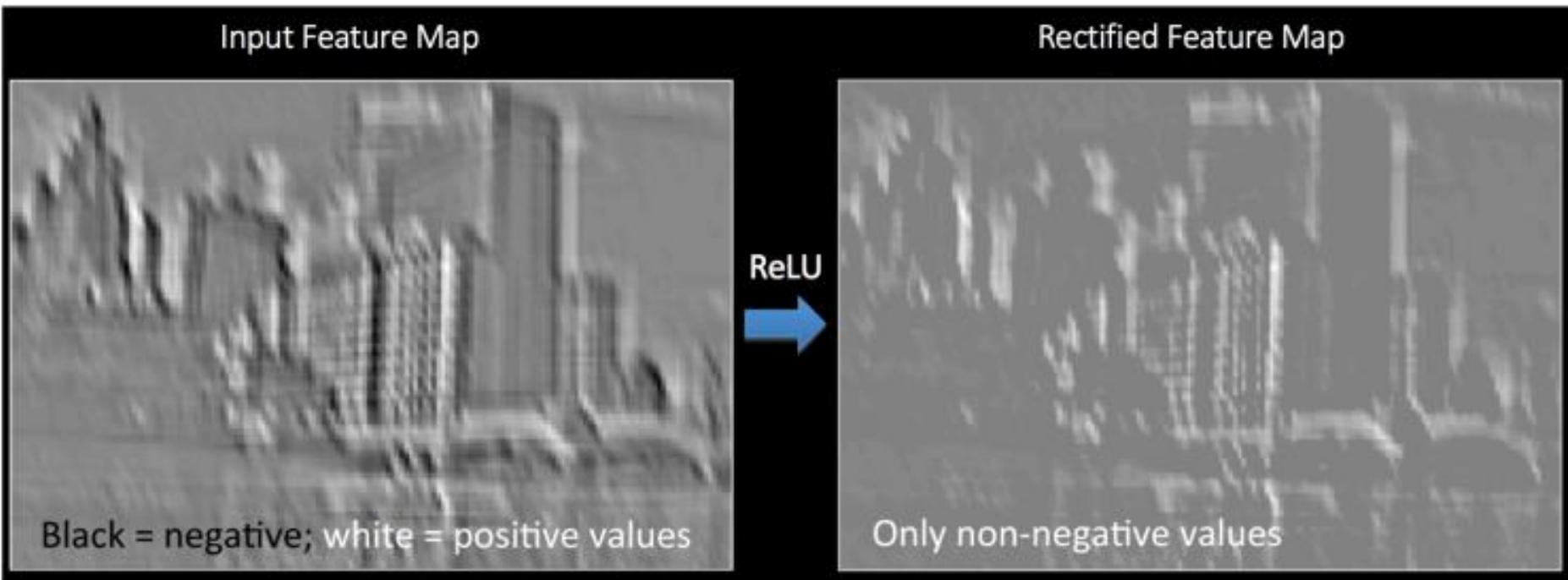
[source] <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

[source]

## 3. CNN

32

- Feature Map > ReLU effect

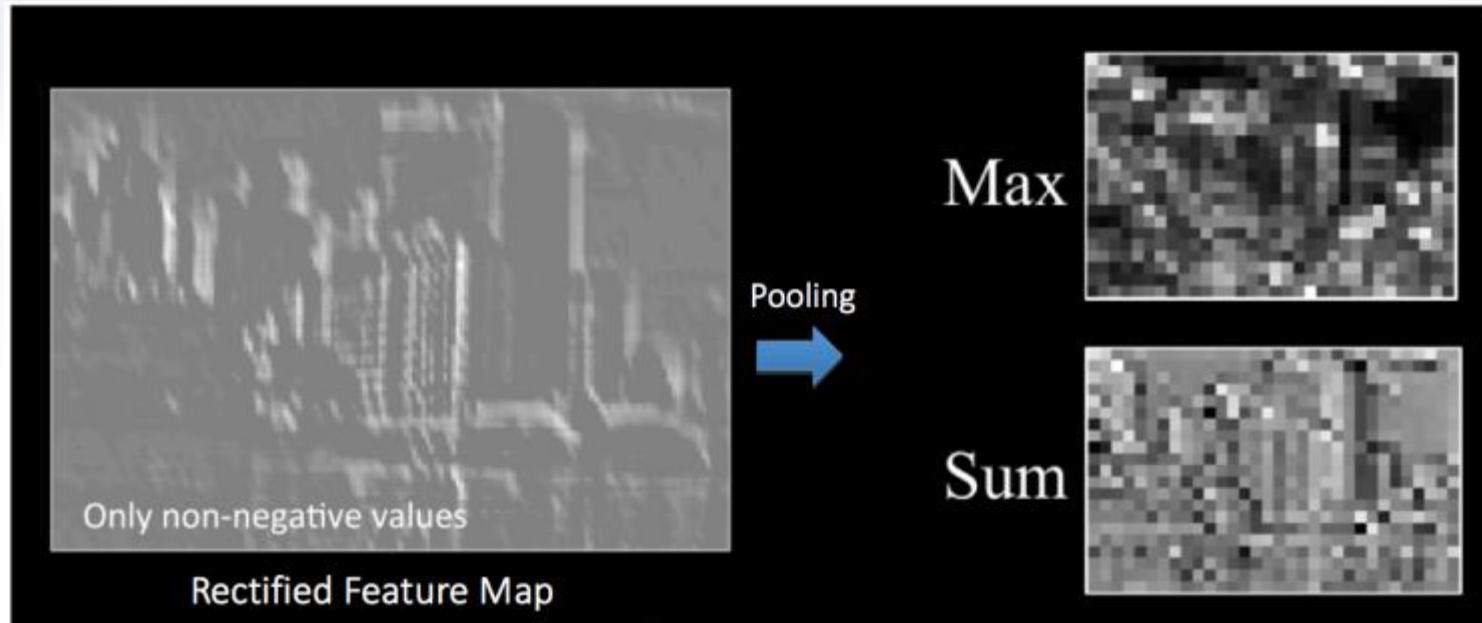


[source] <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

## 3. CNN

33

- Feature Map > ReLU effect > Max Pooling



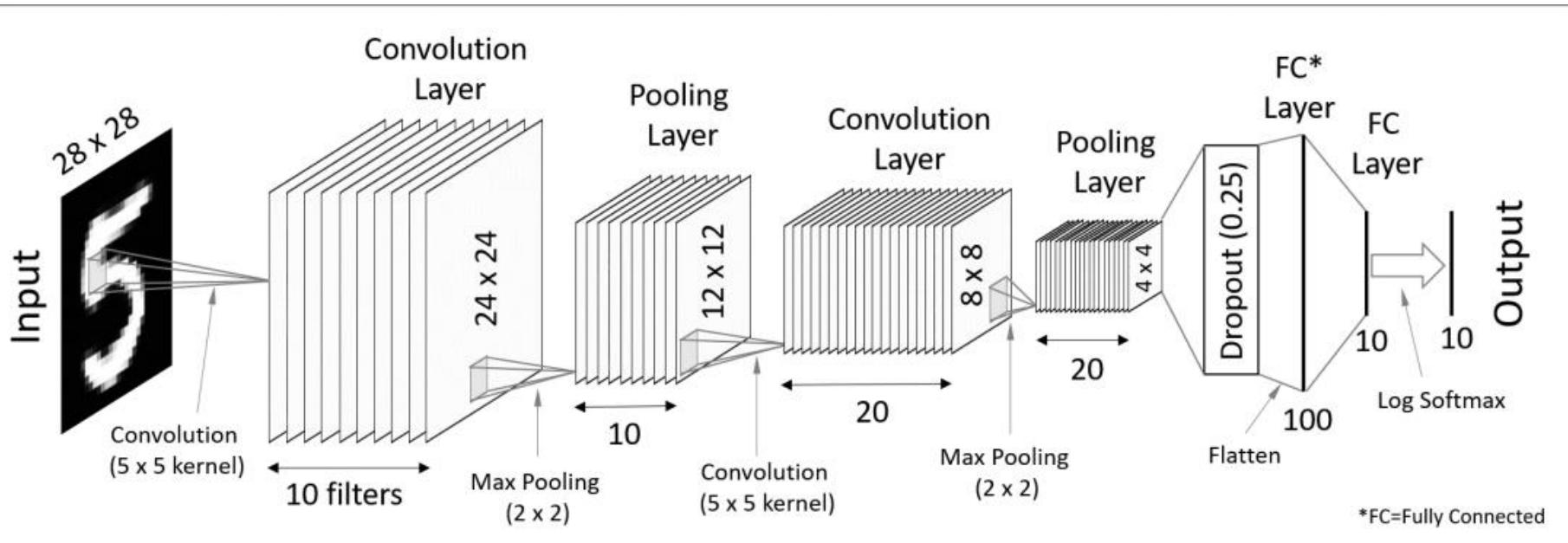
[source] <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

# V. Deep Learning Revisit

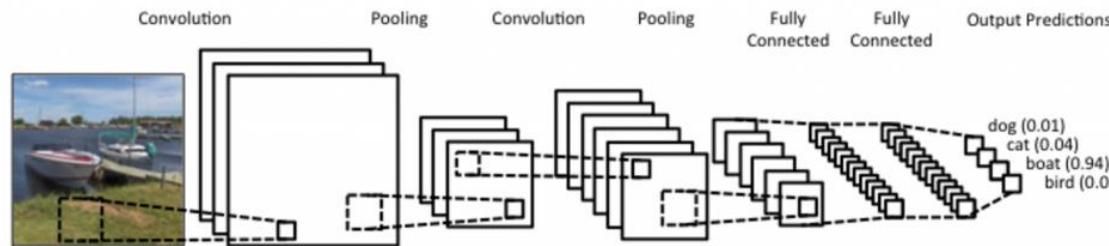
## 3. CNN

34

- Convolution



[source] <https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/>



## 3. CNN

35

- 특징 추출의 과정
- [Convolution → ReLU → Max Pooling] 의 반복 수행



[source] <https://medium.com/@udemeadofia01/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17>

# V. Deep Learning Revisit

## 3. CNN

36

- Applying Convolutional Neural Network



[source] <https://medium.com/@ismailou.sa/convolutional-neural-networks-and-their-application-in-self-driving-cars-33fa0a1625c8>

# V. Deep Learning Revisit

## 3. CNN

- Program Code

```
1 #!/usr/bin/env /c/Apps/Anaconda3/python
2
3 # Import
4 import tensorflow as tf
5 import numpy as np
6 import pandas as pd
7 from sklearn.preprocessing import MinMaxScaler
8 import matplotlib.pyplot as plt
9
10 # Import data
11 data = pd.read_csv('./data_stocks.csv')
12
13 # Drop date variable
14 data = data.drop(['DATE'], 1)
15
16 # Dimensions of dataset
17 n = data.shape[0]
18 p = data.shape[1]
19
20 # Make data a np.array
21 data = data.values
22
23 # Training and test data
24 train_start = 0
25 train_end = int(np.floor(0.8*n))
26 test_start = train_end + 1
27 test_end = n
28 data_train = data[np.arange(train_start, train_end), :]
29 data_test = data[np.arange(test_start, test_end), :]
30
31 # Scale data
32 scaler = MinMaxScaler(feature_range=(-1, 1))
33 scaler.fit(data_train)
34 data_train = scaler.transform(data_train)
35 data_test = scaler.transform(data_test)
36
37 # Build X and y
38 X_train = data_train[:, 1:]
39 y_train = data_train[:, 0]
40 X_test = data_test[:, 1:]
```

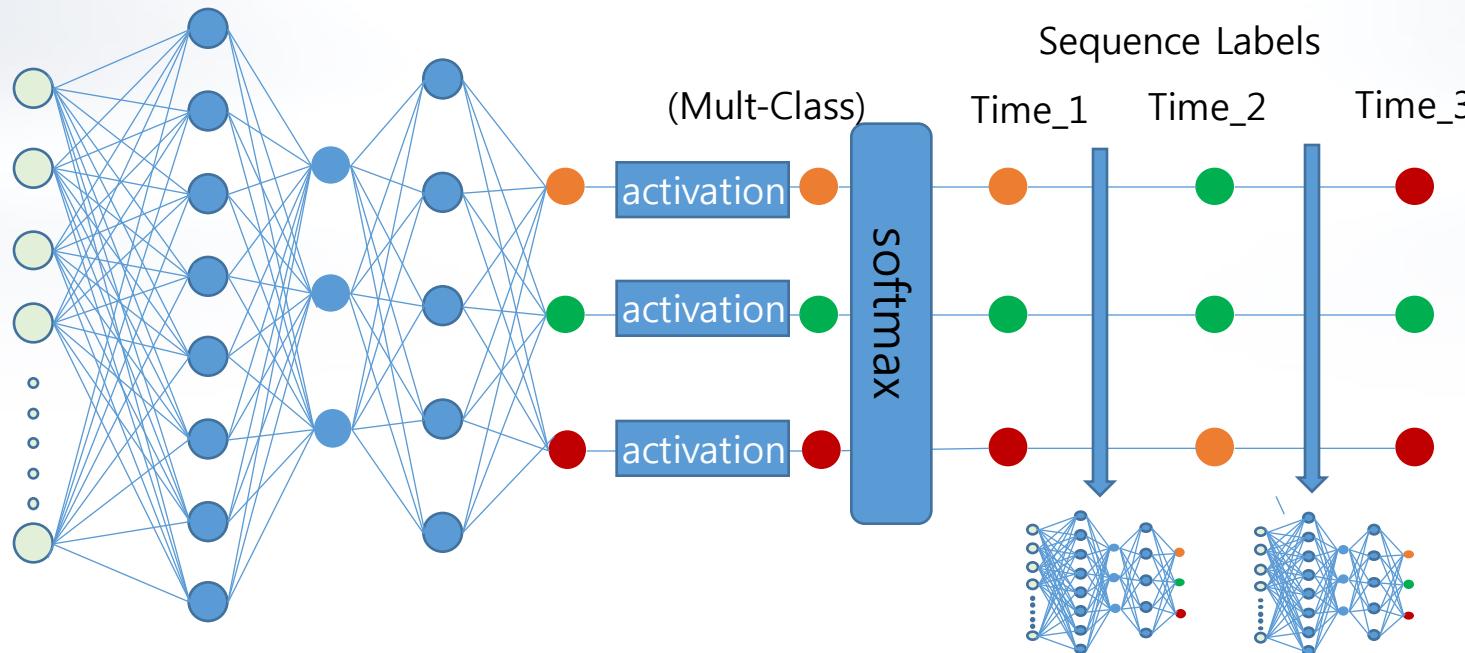
```
41 y_test = data_test[:, 0]
42
43 # Number of stocks in training data
44 n_stocks = X_train.shape[1]
45
46 # Neurons
47 n_neurons_1 = 1024
48 n_neurons_2 = 512
49 n_neurons_3 = 256
50 n_neurons_4 = 128
51
52 # Session
53 net = tf.InteractiveSession()
54
55 # Placeholder
56 X = tf.placeholder(dtype=tf.float32, shape=[None, n_stocks])
57 Y = tf.placeholder(dtype=tf.float32, shape=[None])
58
59 # Hidden layer
60 W_hidden_1 = tf.Variable(tf.random_normal([n_stocks, n_neurons_1]))
61 b_hidden_1 = tf.Variable(tf.zeros([n_neurons_1]))
62
63 # Output layer
64 W_out = tf.Variable(tf.random_normal([n_neurons_4, 1]))
65 b_out = tf.Variable(tf.zeros([1]))
66
67 # Hidden weights
68 W_hidden_2 = tf.Variable(tf.random_normal([n_neurons_1, n_neurons_2]))
69 b_hidden_2 = tf.Variable(tf.zeros([n_neurons_2]))
70
71 # Output weights
72 W_hidden_3 = tf.Variable(tf.random_normal([n_neurons_2, n_neurons_3]))
73 b_hidden_3 = tf.Variable(tf.zeros([n_neurons_3]))
74
75 # Hidden weights
76 W_hidden_4 = tf.Variable(tf.random_normal([n_neurons_3, n_neurons_4]))
77 b_hidden_4 = tf.Variable(tf.zeros([n_neurons_4]))
78
79 # Output layer
80 hidden_1 = tf.nn.relu(tf.add(tf.matmul(X, W_hidden_1), b_hidden_1))
81 hidden_2 = tf.nn.relu(tf.add(tf.matmul(hidden_1, W_hidden_2), b_hidden_2))
```

실습 중 배포

## 4. RNN/LSTM

38

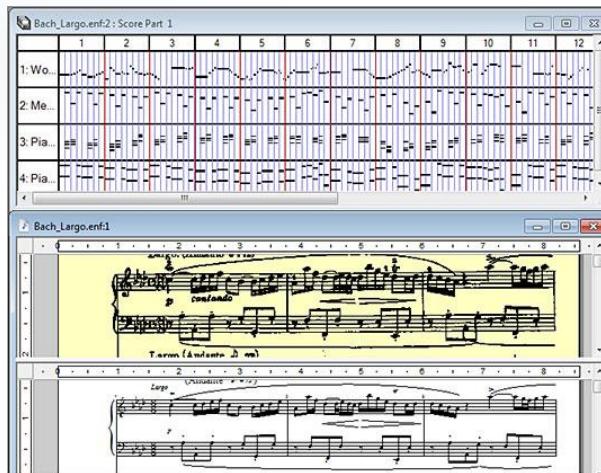
- 문제의 제기
  - Requirement : 실세계에서는 단일예측값을 확장한 연속된 예측값의 필요
  - 실세계에서의 연속 예측값 예시
    - 1주일간의 날씨 예측
    - 시간별 주가 예측



결과값을 입력에 반영하는  
재귀적 신경망 학습 필요

## 4. RNN/LSTM

- Recurrent Neural Network(David Rumelhart, 1986)
  - Learning Sequence Data
  - Sequence Data
    - 수평 또는 수직으로 연속적으로 tensor 의 확장
  - 유형
    - Time Series : Price, Temperature, RPM
    - Streaming : Music, Voice
    - Order : 문장



## 4. RNN/LSTM

40

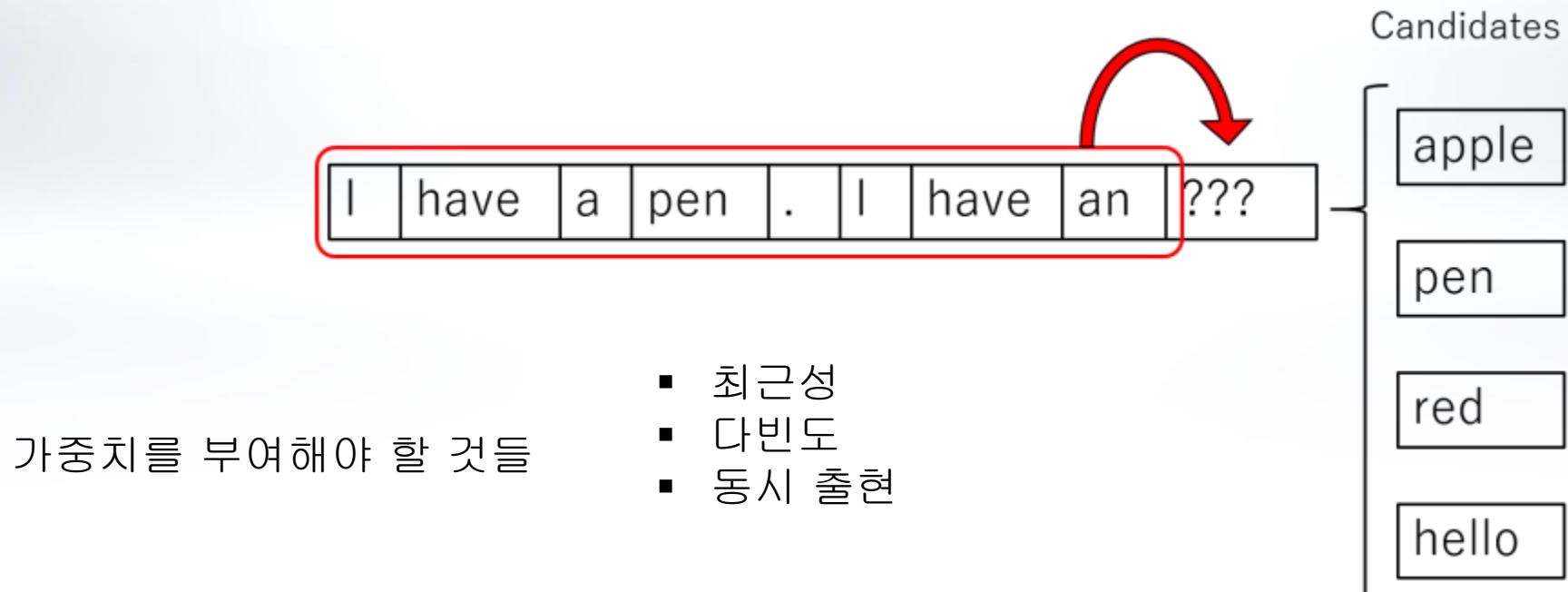
- Sequence Data에 있어서의 예측 = Next Sequence

Sequence Type	Features	Label	Label 수
Time Series	1 2 3 4 5 4 5 6 7 8 7 8 9 10 9 10 11	12	Single
Time Series	1 2 3 4 5 4 5 6 7 8 7 8 9 10 9 10 11	12 13	Multi
문장	곰 세마리가 한 집에 있어 아빠곰 엄마곰	애기곰	Single
문장	주가 하락...(중략)...경제 부양 필요	불황	Single
음악	미 레 도 레 미 미	미	Single
음악	미레 도레 미 미 미	레레레	Multi

## 4. RNN/LSTM

41

- RNN – Natural Language

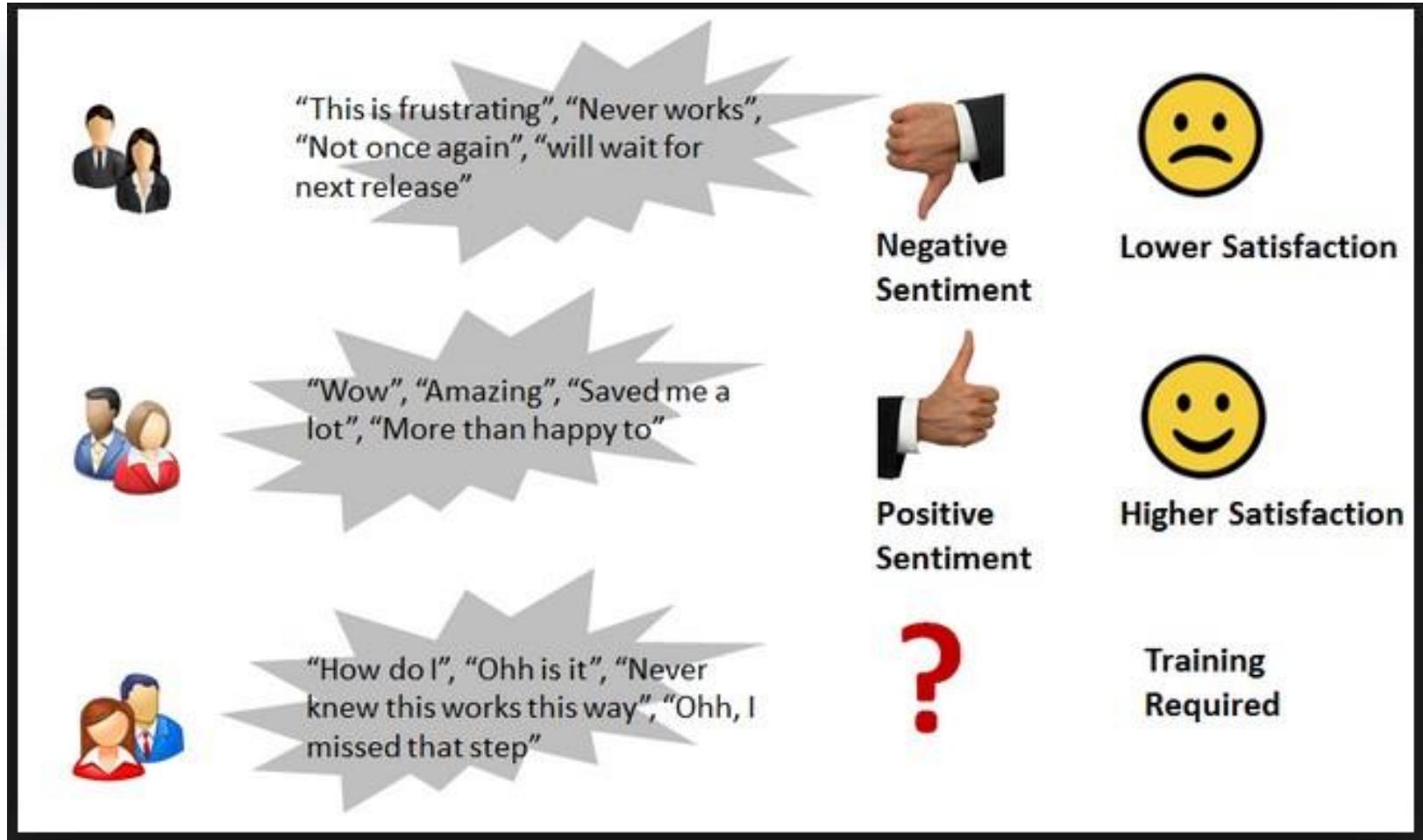


source : <http://corochann.com/recurrent-neural-network-rnn-introduction-1286.html>

## 4. RNN/LSTM

42

- RNN > 감성 식별

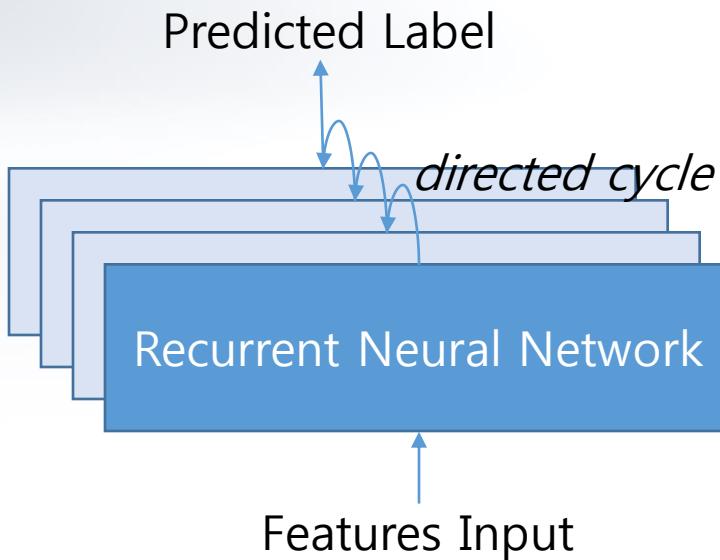


[source] <https://datasciencecmu.wordpress.com/2014/04/18/future-of-sentiment-analysis-and-problems-faced/>

## 4. RNN/LSTM

43

- Recurrent
  - Sequence Data에 있어서의 예측 = Next Sequence

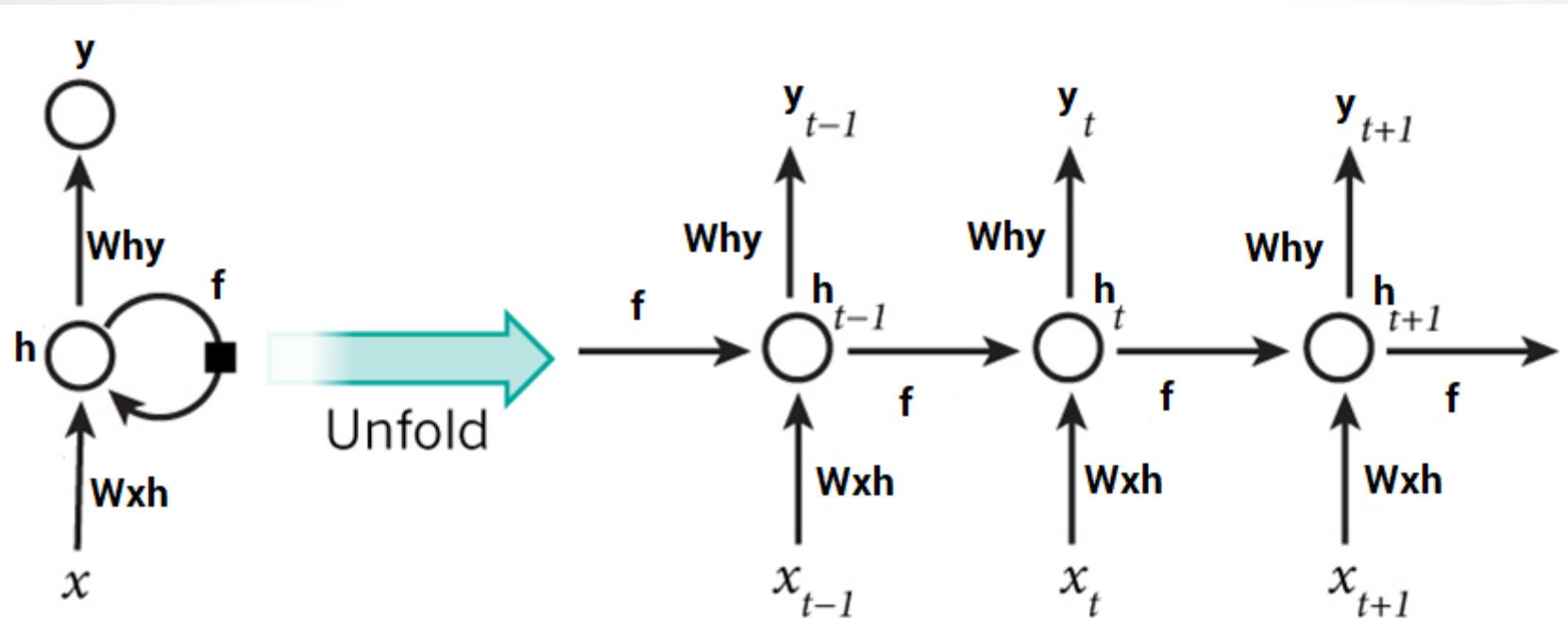


$$\begin{aligned}
 h_t &= f(h_{t-1}, Features_t) \\
 h_t &= \tanH(W_{hh}h_{t-1} + W_{Features}Features_t) \\
 Label_t &= W_{h*Label} * h_t
 \end{aligned}$$

$h_t$  :  $t$ 시점의 hidden state  
 $h_{t-1}$  :  $t - 1$  시점의 hidden state  
 $Features_{t-1}$  :  $t$ 시점의 hidden state  
 $W_{hh}$  : recurrent neurons Weight  
 $W_{Features*h}$  : Features에 대입되는 Weight

## 4. RNN/LSTM

- Recurrent
  - Forward Propagation in a Recurrent Neuron
    - source : <https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/>



[source] <https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/>

## 4. RNN/LSTM

45

- RNN의 구조

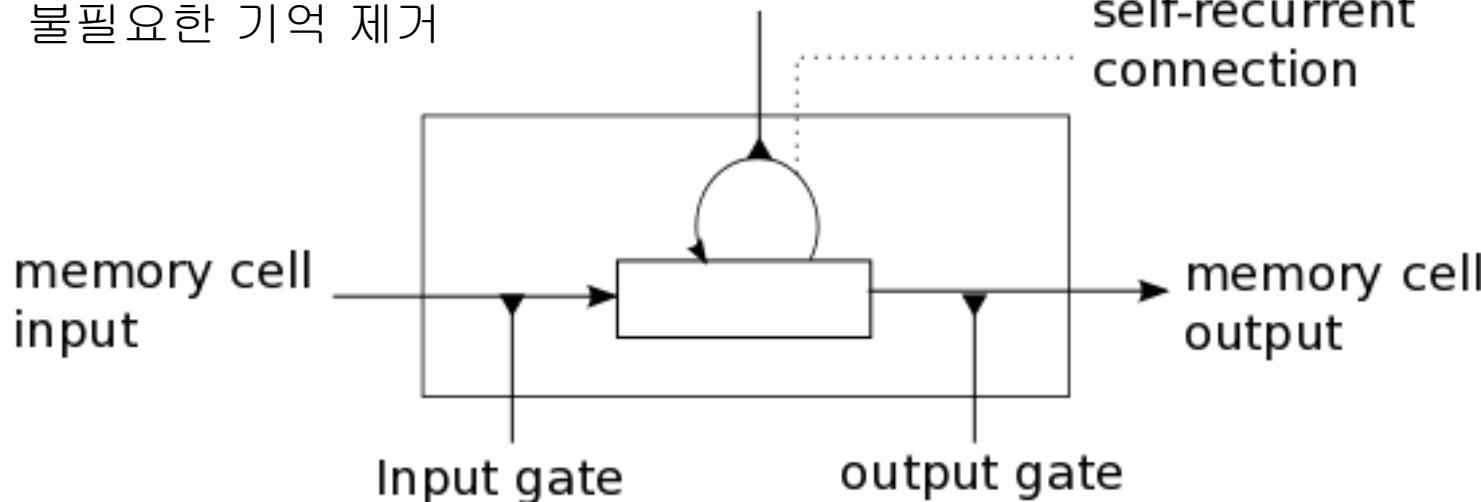
Point : 현재의 상태에 영향을 줄법한 과거 찾기

기억 세포의 양날

- 기억 세포의 활성 필요성
- 불필요한 기억 제거

forget gate

self-recurrent  
connection



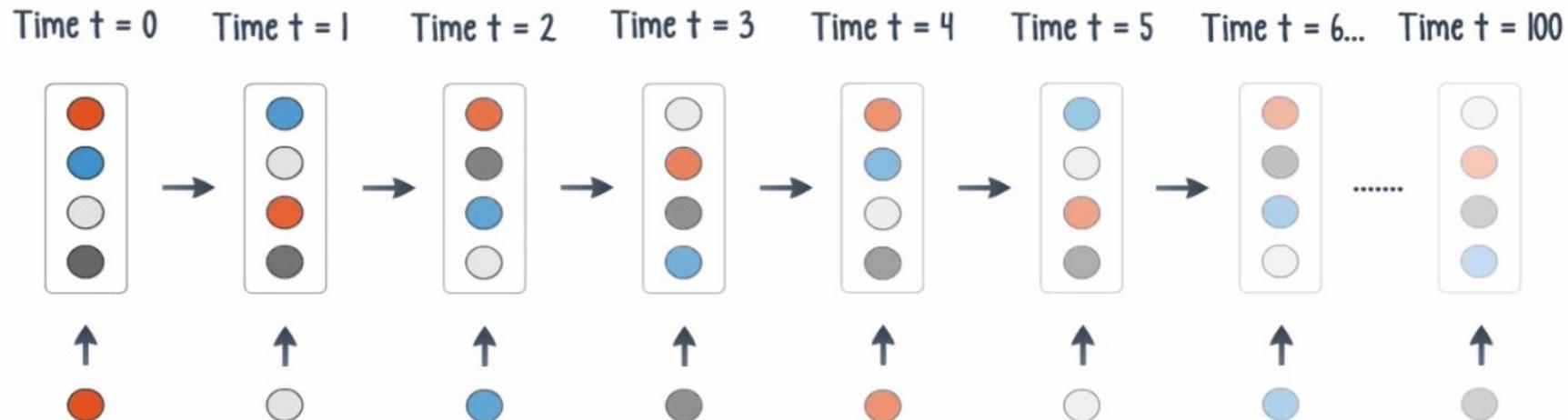
기억해야 할만한 것들...

## 4. RNN/LSTM

46

- RNN의 문제점

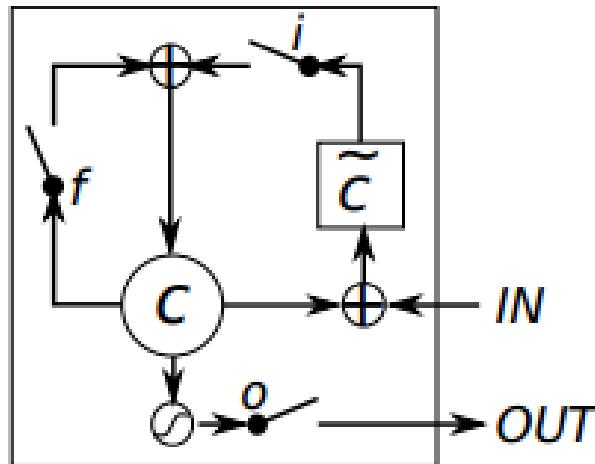
### Decay of information through time



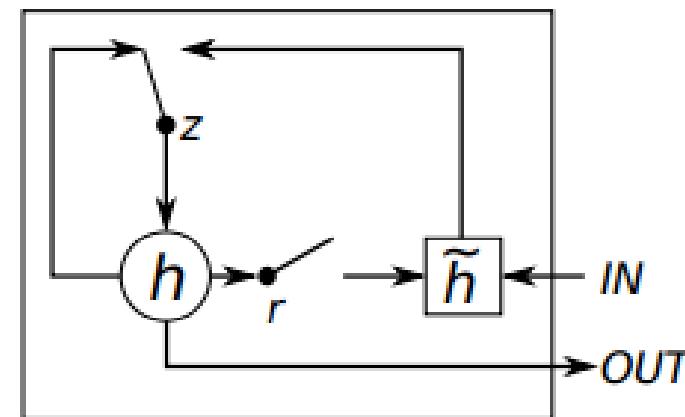
[source] : <https://towardsdatascience.com/using-rnns-for-machine-translation-11ddded78ddf>

## 4. RNN/LSTM

- Solution of RNN with Vanishing and Exploding Gradient Problem
  - LSTM : Long Short-Term Memory
  - GRU : Gated Recurrent Unit



(a) Long Short-Term Memory

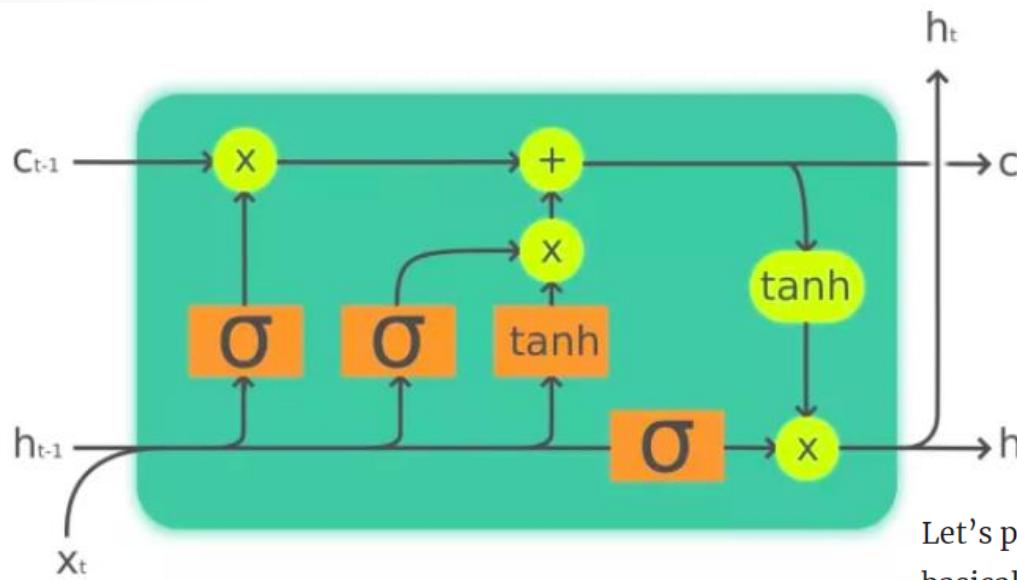


(b) Gated Recurrent Unit

## 4. RNN/LSTM

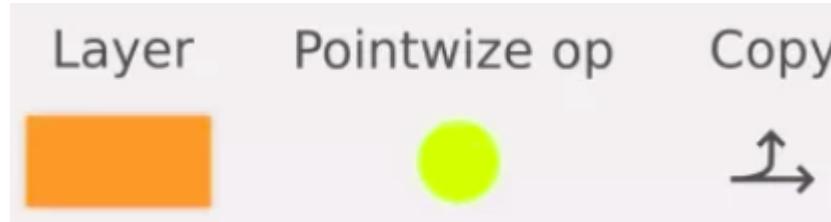
48

- Solution of RNN with Vanishing and Exploding Gradient Problem
  - LSTM : Long Short-Term Memory



$$\begin{aligned}
 i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
 f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
 g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
 o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
 c_t &= f_t * c_{(t-1)} + i_t * g_t \\
 h_t &= o_t * \tanh(c_t)
 \end{aligned}$$

Let's pick this equation apart:  $c_t$  is the new cell state, which is basically the memory of the LSTM.



$f_t$  is called the “forget gate”: it dictates how much of the previous cell state to **retain** (but is slightly confusingly named the forget gate).

$i_t$  is the “input gate” and dictates how much to update the cell state with new information.

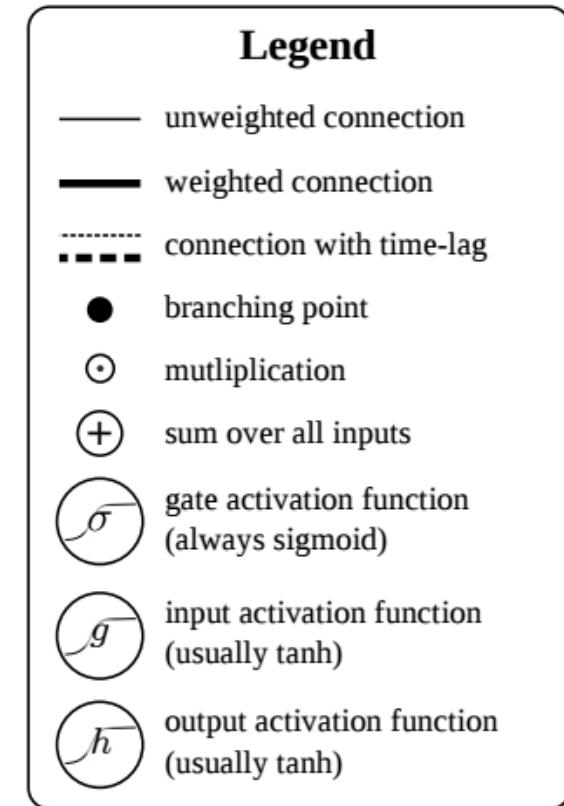
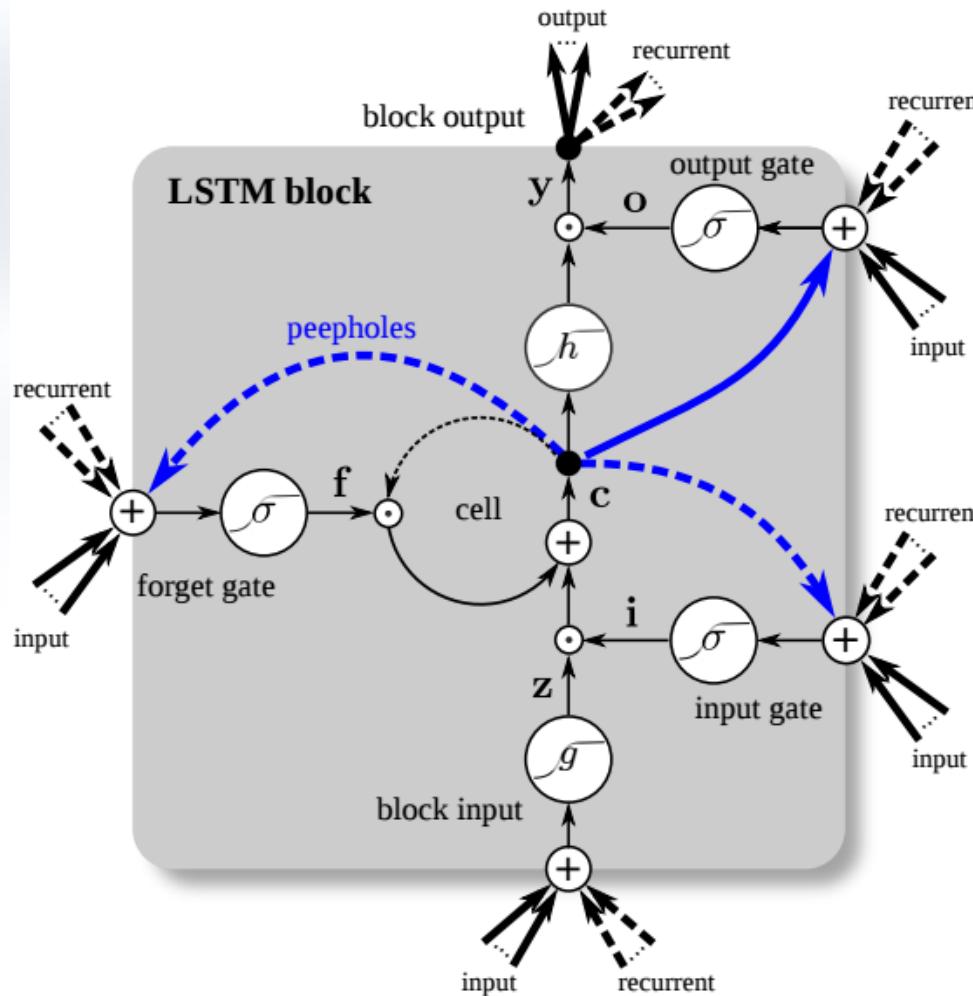
Finally,  $g_t$  is the information we use to update the cell state.

## 4. RNN/LSTM

49

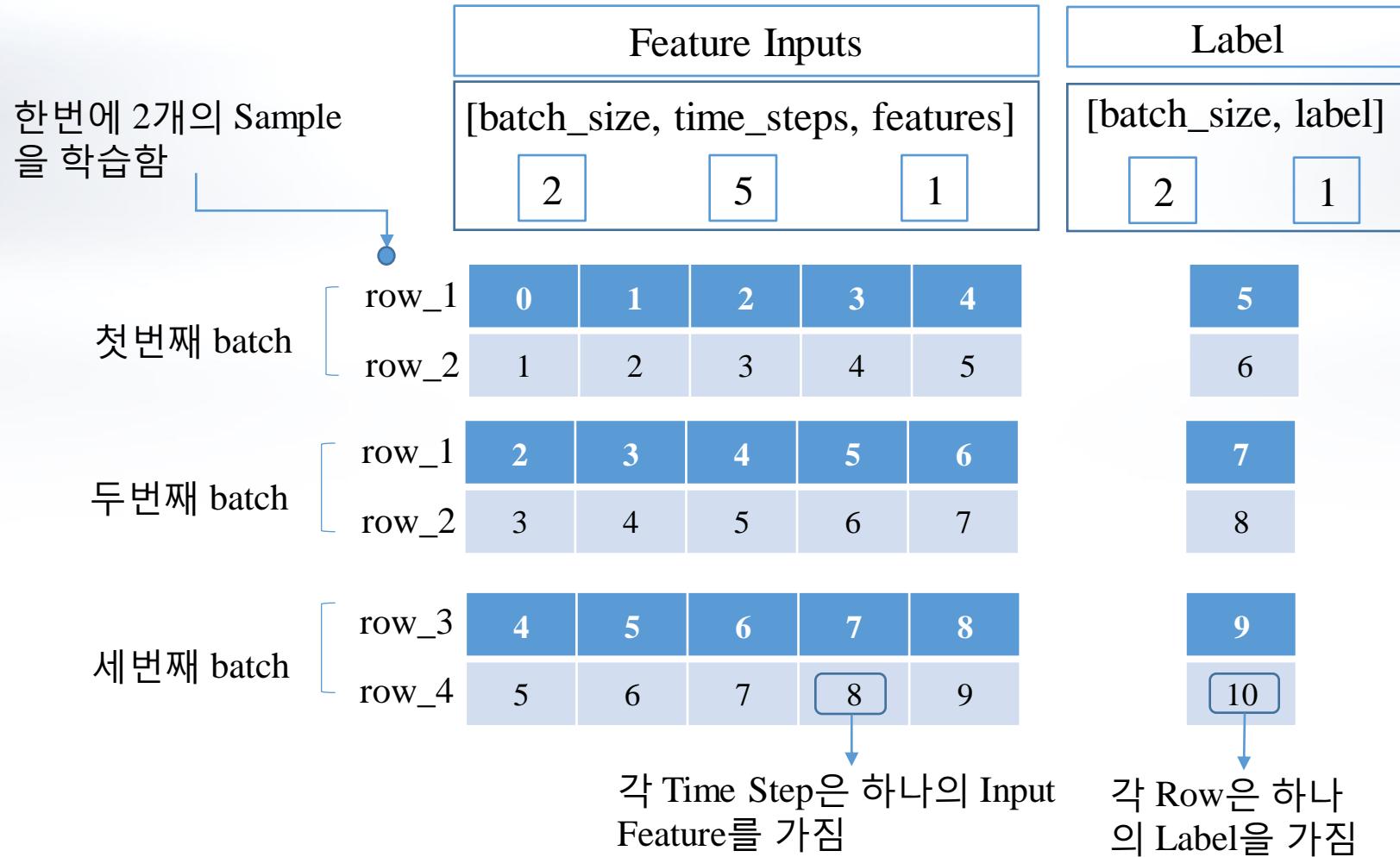
- LSTM의 구조

[source] <https://developer.nvidia.com/discover/lstm>



## 4. RNN/LSTM

- LSTM : Long Short-Term Memory



## 4. RNN/LSTM

- LSTM : Long Short-Term Memory

한번에 2개의 Sample  
을 학습함

첫 번째 batch

두 번째 batch

세 번째 batch

Feature Inputs					
	[batch_size, time_steps, features]				
	2	5	2		
row_1	[0,0]	[1,10]	[2,20]	[3,30]	[4,40]
row_2	[1,10]	[2,20]	[3,30]	[4,40]	[5,50]

Label					
	[batch_size, label]				
	2	1			
row_1	55				
row_2	66				

55
66
77
88

99
100

각 Time Step은 하나의 Input Feature를 가짐

각 Row은 하나의 Label을 가짐

# V. Deep Learning Revisit

## 4. RNN/LSTM

- LSTM : Long Short-Term Memory

한번에 2개의 Sample  
을 학습함

첫 번째 batch

두 번째 batch

세 번째 batch

Feature Inputs					
[batch_size, time_steps, features]					
2	5	2			

Label	
[batch_size, label]	
2	2

row_1	[0,0]	[1,10]	[2,20]	[3,30]	[4,40]
row_2	[1,10]	[2,20]	[3,30]	[4,40]	[5,50]

[55,66]
[66,77]

row_1	[2,20]	[3,30]	[4,40]	[5,50]	[6,60]
row_2	[3,30]	[4,40]	[5,50]	[6,60]	[7,70]

[77,88]
[88,99]

row_3	[4,40]	[5,50]	[6,60]	[7,70]	[8,80]
row_4	[5,50]	[6,60]	[7,70]	[8,80]	[9,90]

[99,110]
[110,121]

각 Time Step은 두 개의 Input Feature를 가짐

각 Row은 하나의 Label을 가짐

## 4. RNN/LSTM

- LSTM : Long Short-Term Memory
  - Text Detection

Feature Inputs	Label
[batch_size, time_steps, features]	[batch_size, label]
1      6      5	1      5
H I H E L L	I H E L L O

H	I	H	E	L	L	I	H	E	L	L	O
row_1 [1,0,0,0,0] [0,1,0,0,0] [1,0,0,0] [0,0,1,0] [0,0,0,1,0] [0,0,0,1,0] [0,1,0,0,0] [1,0,0,0] [0,0,1,0] [0,0,0,1,0] [0,0,0,1,0] [0,0,0,0,1]											

첫 번째 row\_1 [1,0,0,0,0] [0,1,0,0,0] [1,0,0,0] [0,0,1,0] [0,0,0,1,0] [0,0,0,1,0] [0,1,0,0,0] [1,0,0,0] [0,0,1,0] [0,0,0,1,0] [0,0,0,1,0] [0,0,0,0,1]  
batch

One-Hot Encoding	Encode	Decode	placeholderFeatures = tf.placeholder([None, 6, 5])	placeholderLabel = tf.placeholder([None, 6])
	[1,0,0,0,0]	H	)	)
	[0,1,0,0,0]	I	predLabel, states_ = tf.contrib.rnn.BasicLSTMCell(num_units = )	
	[0,0,1,0,0]	E	fcFeatures = tf.reshape(predLabel, [-1, hidden_size])	
	[0,0,0,1,0]	L	predLabel = tf.reshape(inputs=fcFeatures, num_outputs=5)	
	[0,0,0,0,1]	O		

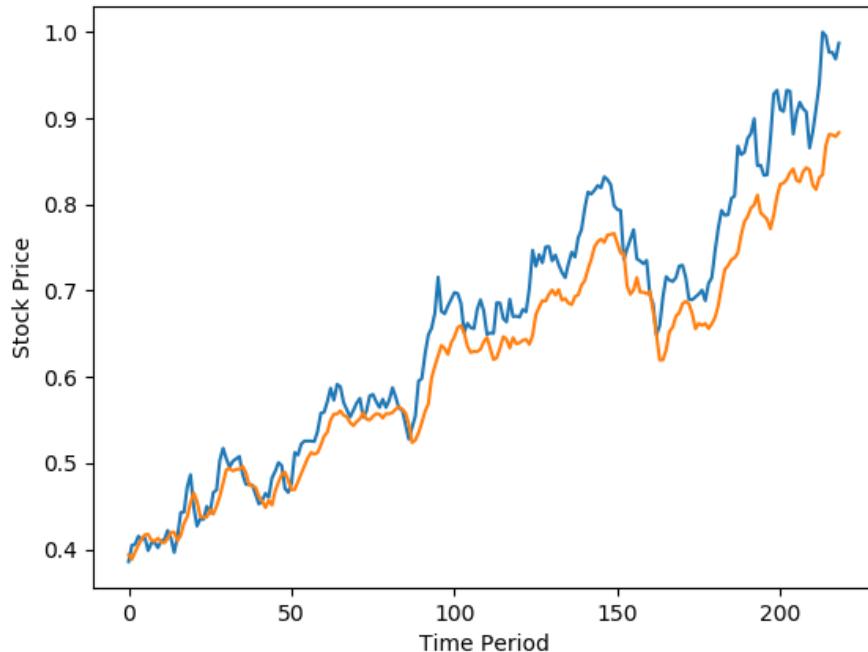
# V. Deep Learning Revisit

## 4. RNN/LSTM

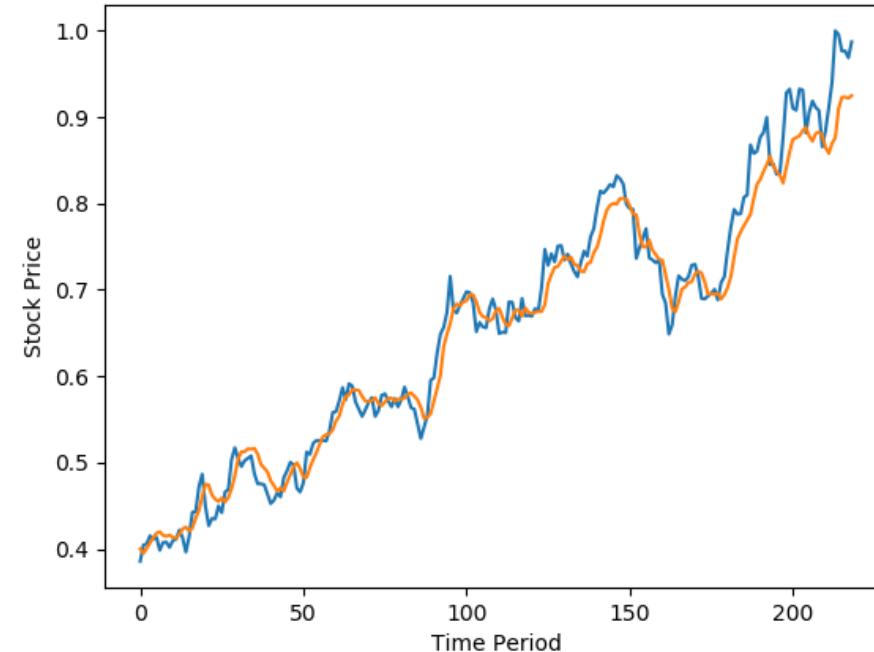
54

- LSTM > 주가 예측

주가 =  $f(\text{시가}, \text{고가}, \text{저가}, \text{종가}, \text{거래량})$



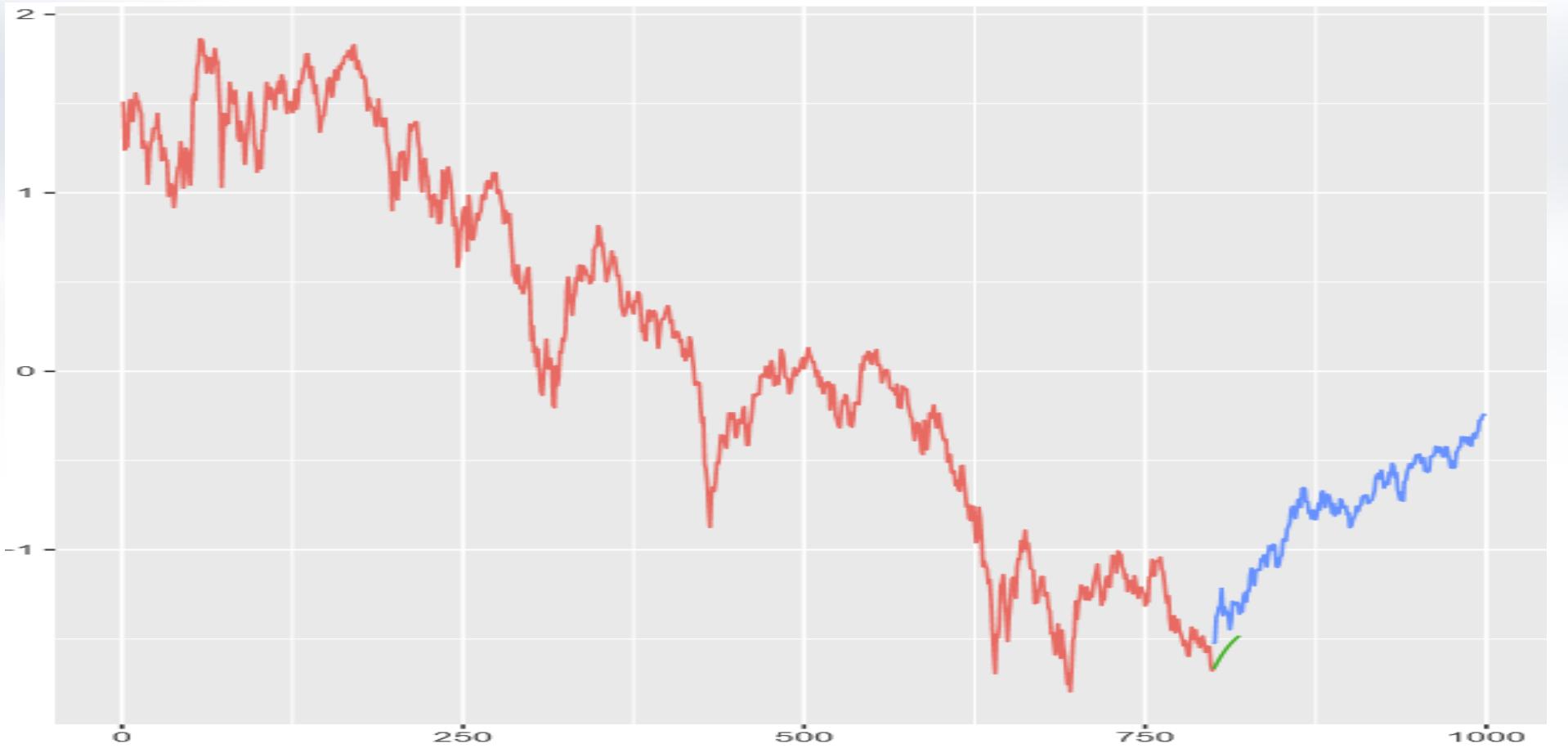
주가 =  $f(\text{시가}, \text{고가}, \text{저가}, \text{종가}, \text{거래량}, \text{환율})$



## 4. RNN/LSTM

55

- LSTM > 주가 예측



## 4. RNN/LSTM

- Program Code

```

1  #!/usr/bin/env /c/Apps/Anaconda3/python
2
3  # Import
4  import tensorflow as tf
5  import numpy as np
6  import pandas as pd
7  from sklearn.preprocessing import MinMaxScaler
8  import matplotlib.pyplot as plt
9
10 # Import data
11 data = pd.read_csv('./data_stocks.csv')
12
13 # Drop date variable
14 data = data.drop(['DATE'], 1)
15
16 # Dimensions of dataset
17 n = data.shape[0]
18 p = data.shape[1]
19
20 # Make data a np.array
21 data = data.values
22
23 # Training and test data
24 train_start = 0
25 train_end = int(np.floor(0.8*n))
26 test_start = train_end + 1
27 test_end = n
28 data_train = data[np.arange(train_start, train_end), :]
29 data_test = data[np.arange(test_start, test_end), :]
30
31 # Scale data
32 scaler = MinMaxScaler(feature_range=(-1, 1))
33 scaler.fit(data_train)
34 data_train = scaler.transform(data_train)
35 data_test = scaler.transform(data_test)
36
37 # Build X and y
38 X_train = data_train[:, 1:]
39 y_train = data_train[:, 0]
40 X_test = data_test[:, 1:]

```

```

41 y_test = data_test[:, 0]
42
43 # Number of stocks in training data
44 n_stocks = X_train.shape[1]
45
46 # Neurons
47 n_neurons_1 = 1024
48 n_neurons_2 = 512
49 n_neurons_3 = 256
50 n_neurons_4 = 128
51
52 # Session
53 net = tf.InteractiveSession()
54
55 # Placeholder
56 X = tf.placeholder(dtype=tf.float32, shape=[None, n_stocks])
57 Y = tf.placeholder(dtype=tf.float32, shape=[None])
58
59 # Hidden layer 1
60 W_hidden_1 = tf.Variable(tf.random_normal([n_stocks, n_neurons_1], mean=0.0, stddev=0.01, seed=12345))
61 bias_hidden_1 = tf.Variable(tf.zeros_initializer([n_neurons_1]))
62
63 # Hidden layer 2
64 W_hidden_2 = tf.Variable(tf.random_normal([n_neurons_1, n_neurons_2], mean=0.0, stddev=0.01, seed=12345))
65 bias_hidden_2 = tf.Variable(tf.zeros_initializer([n_neurons_2]))
66
67 # Hidden layer 3
68 W_hidden_3 = tf.Variable(tf.random_normal([n_neurons_2, n_neurons_3], mean=0.0, stddev=0.01, seed=12345))
69 bias_hidden_3 = tf.Variable(tf.zeros_initializer([n_neurons_3]))
70
71 # Output layer
72 W_out = tf.Variable(tf.random_normal([n_neurons_3, 1], mean=0.0, stddev=0.01, seed=12345))
73 bias_out = tf.Variable(tf.zeros_initializer([1]))
74
75 # Hidden layer 1
76 hidden_1 = tf.nn.relu(tf.add(tf.matmul(X, W_hidden_1), bias_hidden_1))
77
78 # Hidden layer 2
79 hidden_2 = tf.nn.relu(tf.add(tf.matmul(hidden_1, W_hidden_2), bias_hidden_2))
80
81 # Output layer
82 output = tf.add(tf.matmul(hidden_2, W_out), bias_out)
83
84 # Loss function
85 loss = tf.reduce_mean(tf.square(Y - output))
86
87 # Optimizer
88 optimizer = tf.train.AdamOptimizer().minimize(loss)
89
90 # Accuracy
91 correct_prediction = tf.equal(tf.argmax(output, 1), tf.argmax(Y, 1))
92 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
93
94 # Training loop
95 for i in range(1000):
96     net.run(optimizer)
97
98 # Test
99 accuracy_val = net.run(accuracy)
100 print('Accuracy: ', accuracy_val)

```

실습 중 배포

## 5. Autoencoder

57

- 자기 지도 학습
  - 입력 차원들의 재구성을 통한 잠재적(latent) 공통 특징 추출
  - 자기 복원(self reconstruction)
  - 비지도학습과 유사
- 자료의 재표현
  - 데이터의 재표현을 통한 유사도 측정
  - 재표현 과정은 PCA의 변수 축소과정과 동일한 맥락임
  - 재표현 기법
    - Encoding : Latent Representation
    - Decoding : Reconstruction Input Data
  - 재표현 과정에서 정보 손실 발생
    - 정보 손실량을 검토하여 이상치 식별
    - Reconstruction Error가 클수록 이상치

Formula

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

$$\phi, \psi = \arg \min_{\phi, \psi} \|X - (\psi \circ \phi)X\|^2$$

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

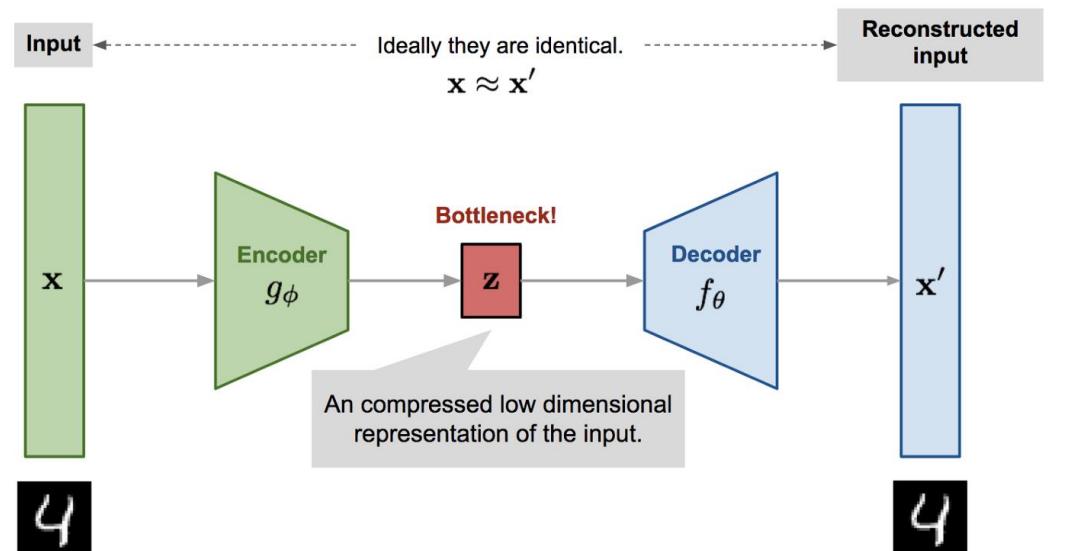
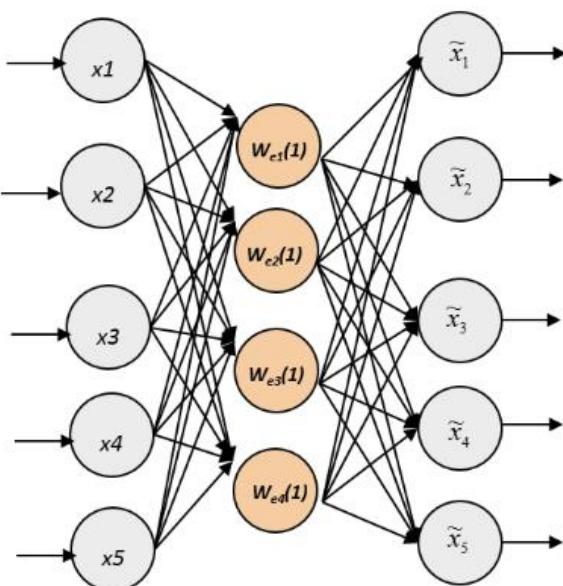
$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

## 5. Autoencoder

58

- 차원의 축소
  - Input Dimension(5) 보다 Hidden Dimension(4)의 차원이 적음



$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

## 5. Autoencoder

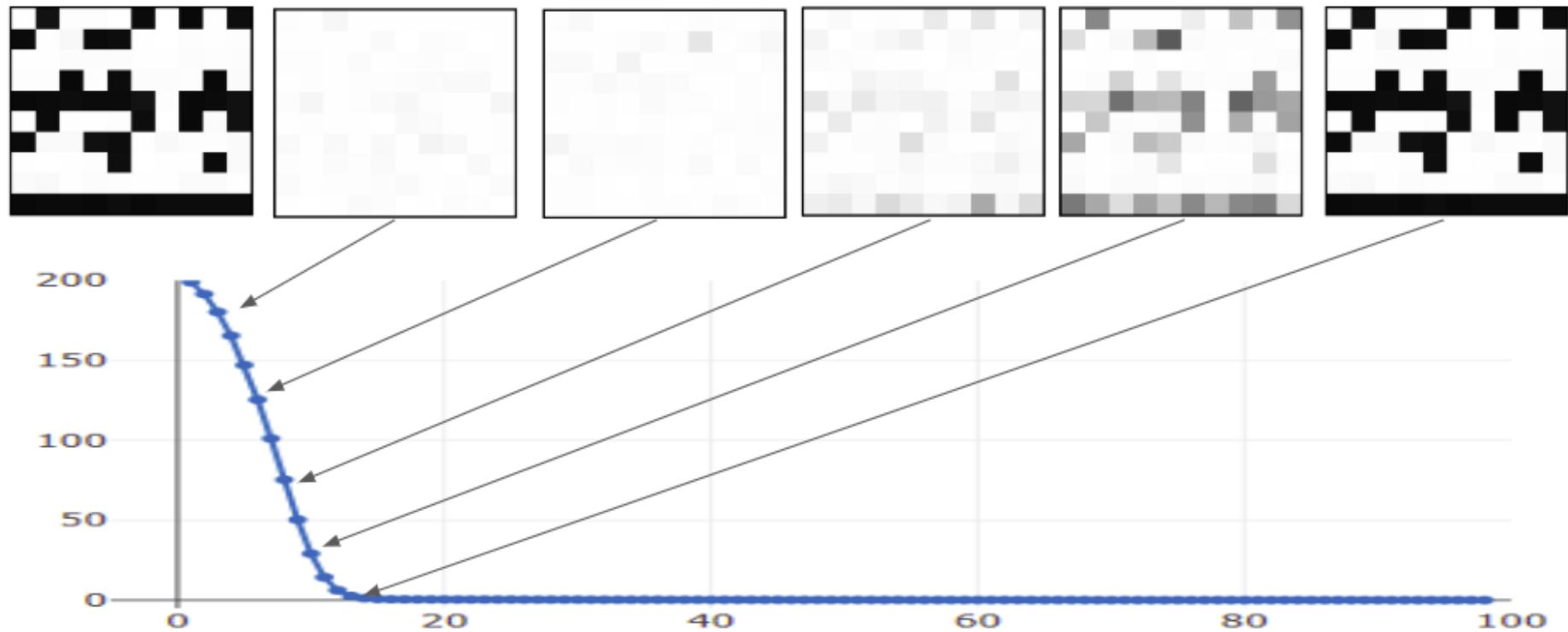
59

- 학습
  - 네트워크의 좌우 대칭 구성
  - 입력차원의 수 = 출력차원의 수
  - output = reconstructed input
  - encoder의 hidden dimension의 축소
  - encoder의 hidden dimension의 수 = decoder의 hidden dimension의 수
  - 출력차원들은 모든 Raw Data의 변환된 값을 산출
- Output
  - 차원 축소모든 Raw Data의 변환된 값을 산출(입력 차원의 수를 줄이는 것은 아님)
  - 정보 손실을 최소화
  - Reconstruction Error= Input Vector - Output Vector
  - PCA보다 높은 성능
  - Sparse한 자료에도 상대적으로 높은 성능

## 5. Autoencoder

60

- 학습 과정 데모 시연



[source] <https://github.com/max-ng/Autoencoder-in-javascript>

## 5. Autoencoder

- Program Code

```

1 #!/usr/bin/env /c/Apps/Anaconda3/python
2
3 # Import
4 import tensorflow as tf
5 import numpy as np
6 import pandas as pd
7 from sklearn.preprocessing import MinMaxScaler
8 import matplotlib.pyplot as plt
9
10 # Import data
11 data = pd.read_csv('./data_stocks.csv')
12
13 # Drop date variable
14 data = data.drop(['DATE'], 1)
15
16 # Dimensions of dataset
17 n = data.shape[0]
18 p = data.shape[1]
19
20 # Make data a np.array
21 data = data.values
22
23 # Training and test data
24 train_start = 0
25 train_end = int(np.floor(0.8*n))
26 test_start = train_end + 1
27 test_end = n
28 data_train = data[np.arange(train_start, train_end), :]
29 data_test = data[np.arange(test_start, test_end), :]
30
31 # Scale data
32 scaler = MinMaxScaler(feature_range=(-1, 1))
33 scaler.fit(data_train)
34 data_train = scaler.transform(data_train)
35 data_test = scaler.transform(data_test)
36
37 # Build X and y
38 X_train = data_train[:, 1:]
39 y_train = data_train[:, 0]
40 X_test = data_test[:, 1:]

```

```

41 y_test = data_test[:, 0]
42
43 # Number of stocks in training data
44 n_stocks = X_train.shape[1]
45
46 # Neurons
47 n_neurons_1 = 1024
48 n_neurons_2 = 512
49 n_neurons_3 = 256
50 n_neurons_4 = 128
51
52 # Session
53 net = tf.InteractiveSession()
54
55 # Placeholder
56 X = tf.placeholder(dtype=tf.float32, shape=[None, n_stocks])
57 Y = tf.placeholder(dtype=tf.float32, shape=[None])
58
59 # Hidden layer 1
60 weight_initializer = tf.variance_scaling_initializer(mode="fan_avg", distribution="uniform")
61 bias_initializer = tf.zeros_initializer()
62
63 # Hidden weights
64 W_hidden_1 = tf.Variable(weight_initializer([n_stocks, n_neurons_1]))
65 bias_hidden_1 = tf.Variable(bias_initializer([n_neurons_1]))
66
67 W_hidden_2 = tf.Variable(weight_initializer([n_neurons_1, n_neurons_2]))
68 bias_hidden_2 = tf.Variable(bias_initializer([n_neurons_2]))
69
70 W_hidden_3 = tf.Variable(weight_initializer([n_neurons_2, n_neurons_3]))
71 bias_hidden_3 = tf.Variable(bias_initializer([n_neurons_3]))
72
73 W_hidden_4 = tf.Variable(weight_initializer([n_neurons_3, n_neurons_4]))
74 bias_hidden_4 = tf.Variable(bias_initializer([n_neurons_4]))
75
76 # Output weights
77 W_out = tf.Variable(weight_initializer([n_neurons_4, 1]))
78 bias_out = tf.Variable(bias_initializer([1]))
79
80 # Hidden layer
81 hidden_1 = tf.nn.relu(tf.add(tf.matmul(X, W_hidden_1), bias_hidden_1))
82 hidden_2 = tf.nn.relu(tf.add(tf.matmul(hidden_1, W_hidden_2), bias_hidden_2))

```

실습 중 배포

