

Artificial Neural Network

With tensorflow and keras

경희대학교 컴퓨터공학과
2019102191 신주영

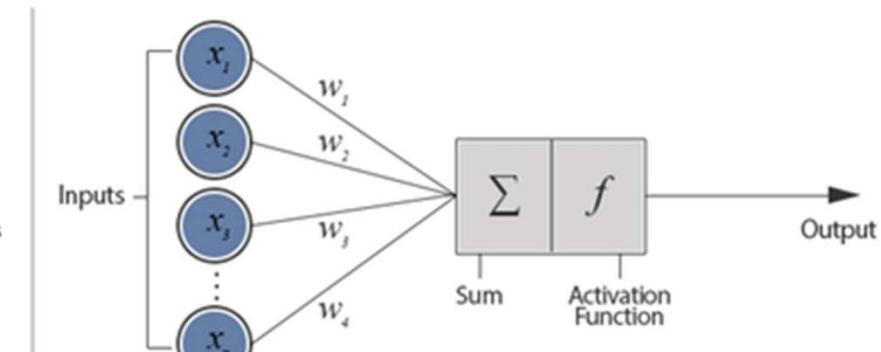
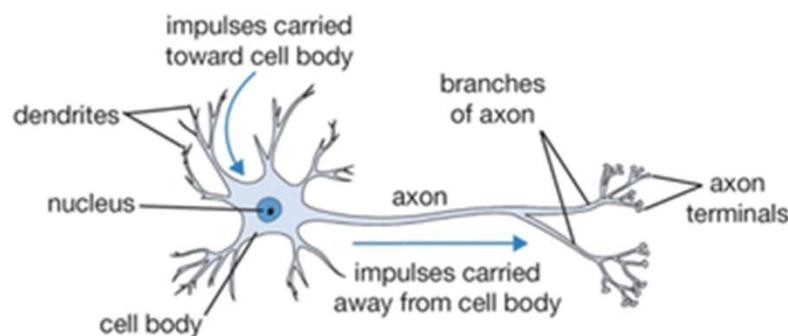


인공신경망 (Artificial Neural Network)

- 인간 두뇌에 대한 계산적 모델을 통해 인공지능을 구현하려는 분야
- 인간의 뇌 구조를 모방: 뉴런과 뉴런 사이에는 전기신호를 통해 정보를 전달

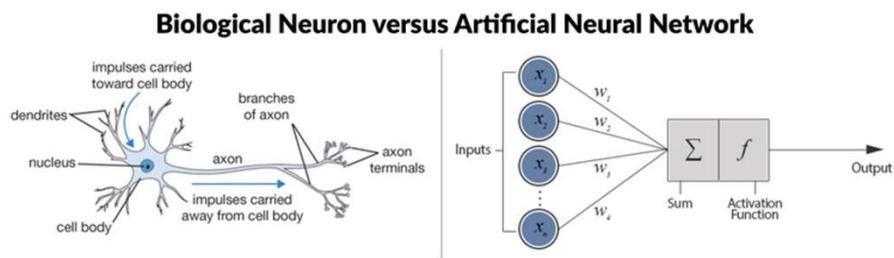
생물학적 신경세포와 인공신경망 비교

Biological Neuron versus Artificial Neural Network



경희대학교
KYUNG HEE UNIVERSITY

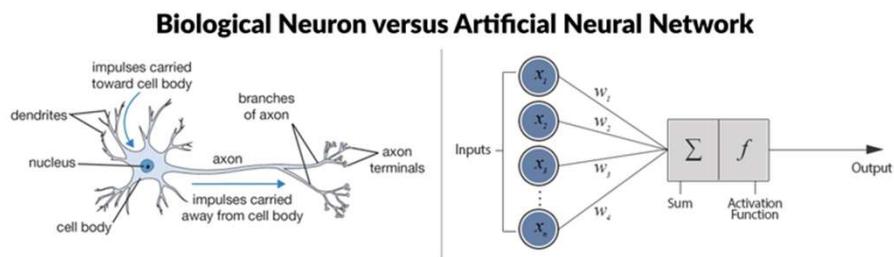
생물학적 신경세포와 인공신경망 비교



- **신경세포(Neuron)**

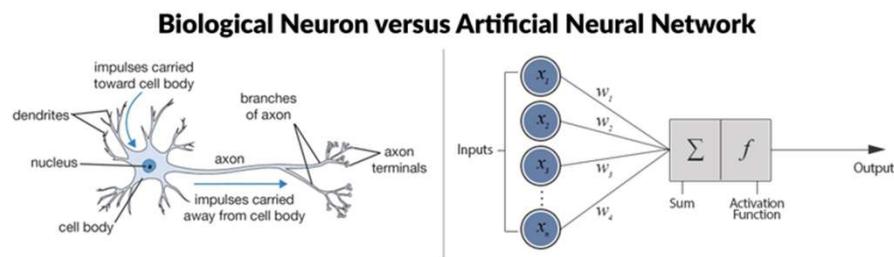
- 수상돌기(Dendrite) : 다른 신경 세포의 축색돌기와 연결되어 전 기화학적 신호를 받아들이는 부위
- 축색돌기(Axon) : 수신한 전기화 학적 신호의 합성결과 값이 특정 임계 값이 이상이면 신호를 내보는 부위
- 신경연접(Synapse) : 수상돌기와 축색돌기 연결 부위, 전달되는 신호의 증폭 또는 감쇄

생물학적 신경세포와 인공신경망 비교



- 인공 뉴런(Artificial Neuron)
 - 신경세포 구조를 단순화하여 모델링한 구조
 - 노드(Node)와 엣지(Edge)로 표현
 - 하나의 노드 안에서 입력(Inputs)과 가중치(Weights)를 곱하고 더하는 선형구조(linear)
 - 활성화 함수(activation function)를 통한 비선형 구조(non-linear) 표현 가능

생물학적 신경세포와 인공신경망 비교



- 인공 신경망(Artificial Neural Network)
 - 여러 개의 인공뉴런들이 모여 연결된 형태
 - 뉴런들이 모인 하나의 단위를 층(layer)이라고 하고, 여러 층(multi layer)으로 이루어질 수 있음
 - ex) 입력층(input layer), 은닉층(hidden layer), 출력층(output layer)

Deep Learning Framework

Tensorflow



TensorFlow

- 가장 널리 쓰이는 딥러닝 프레임워크
- 구글이 주도적으로 개발하는 플랫폼
- 파이썬, C++ API를 기본적으로 제공하고, 자바스크립트(JavaScript), 자바(Java), 고(Go), 스위프트(Swift) 등 다양한 프로그래밍 언어를 지원
- tf.keras를 중심으로 고수준 API 통합(2.x 버전)
- TPU(Tensor Processing Unit) 지원
 - TPU는 GPU보다 전력을 적게 소모, 경제적
 - 일반적으로 32비트(float32)로 수행되는 루프 연산을 16비트(float16)로 낮춤



경희대학교
KYUNG HEE UNIVERSITY

Deep Learning Framework

Keras



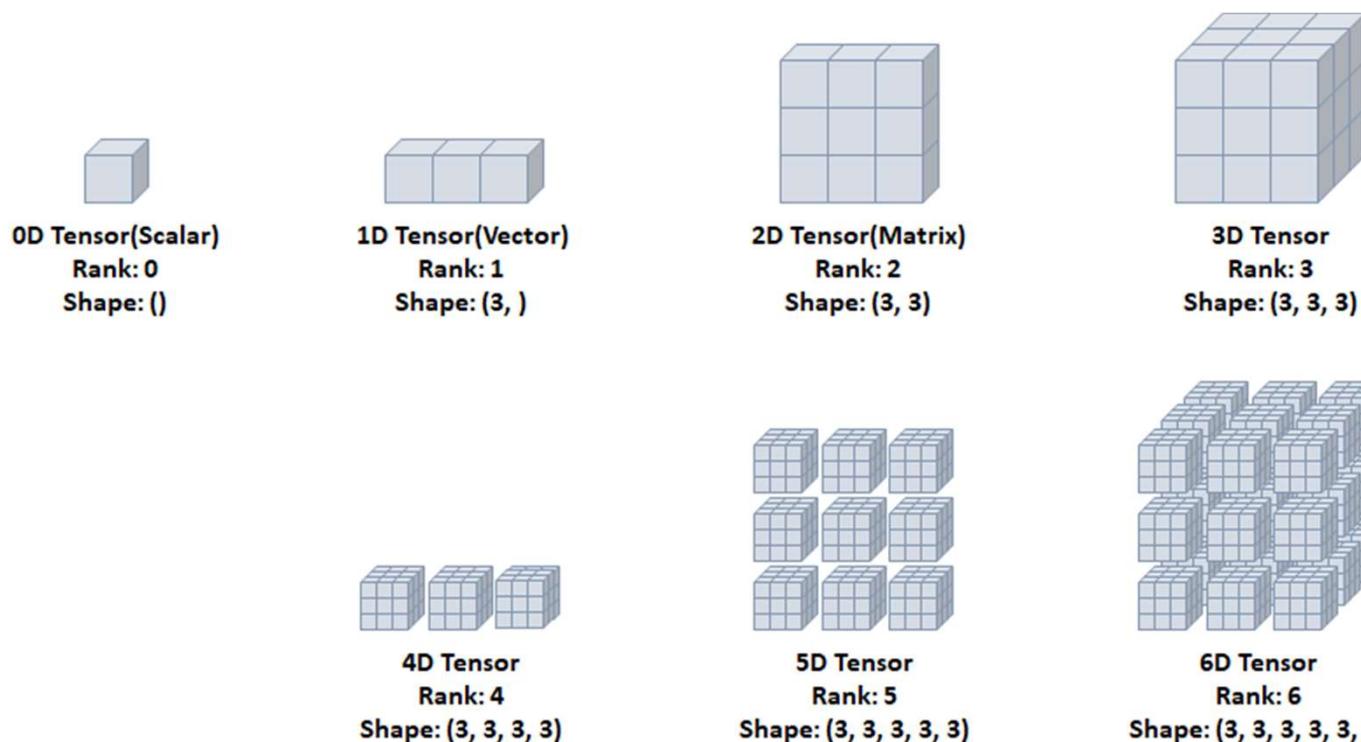
Keras

- 파이썬으로 작성된 고수준 신경망 API로 TensorFlow, CNTK, 혹은 Theano와 함께 사용 가능
- 사용자 친화성, 모듈성, 확장성을 통해 빠르고 간편한 프로토타이핑 가능
- 컨볼루션 신경망, 순환 신경망, 그리고 둘의 조합까지 모두 지원
- CPU와 GPU에서 매끄럽게 실행

딥러닝 데이터 표현과 연산

- 데이터 표현을 위한 기본 구조로 텐서(tensor)를 사용
- 텐서는 데이터를 담기 위한 컨테이너(container)로서 일반적으로 수치형 데이터를 저장

딥러닝 데이터 표현과 연산



텐서 (Tensor)

- Rank: 축의 개수
- Shape: 형상(각 축에 따른 차원 개수)
- Type: 데이터 타입



0D Tensor(Scalar)

- 하나의 숫자를 담고 있는 텐서(tensor)
- 축과 형상이 없음

```
In [2]: t0 = tf.constant(1)
print(t0)
print(tf.rank(t0))
```

```
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(0, shape=(), dtype=int32)
```

1D Tensor(Vector)

- 값들을 저장한 리스트와 유사한 텐서
- 하나의 축이 존재

```
In [3]: t1 = tf.constant([1, 2, 3])
print(t1)
print(tf.rank(t1))
```

```
tf.Tensor([1 2 3], shape=(3,), dtype=int32)
tf.Tensor(1, shape=(), dtype=int32)
```

2D Tensor(Matrix)

- 행렬과 같은 모양으로 두개의 축이 존재
- 일반적인 수치, 통계 데이터셋이 해당
- 주로 샘플(samples)과 특성(features)을 가진 구조로 사용

```
In [4]: t2 = tf.constant([[1, 2, 3],  
                      [4, 5, 6],  
                      [7, 8, 9]])  
  
print(t2)  
print(tf.rank(t2))  
  
tf.Tensor(  
[[1 2 3]  
 [4 5 6]  
 [7 8 9]], shape=(3, 3), dtype=int32)  
tf.Tensor(2, shape=(), dtype=int32)
```

3D Tensor

- 큐브(cube)와 같은 모양으로 세개의 축이 존재
- 데이터가 연속된 시퀀스 데이터나 시계열 데이터에 해당
- 샘플(samples), 타임스텝(timesteps), 특성(features) 등

```
In [5]: t3 = tf.constant([[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]],
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]],
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]])
print(t3)
print(tf.rank(3))
```

```
tf.Tensor(
[[[1 2 3]
[4 5 6]
[7 8 9]]

[[1 2 3]
[4 5 6]
[7 8 9]]

[[1 2 3]
[4 5 6]
[7 8 9]]], shape=(3, 3, 3), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
```



4D Tensor

- 4개의 축
- 컬러 이미지 데이터가 대표적인 사례 (흑백 이미지 데이터는 3D Tensor로 가능)
- 주로 샘플(samples), 높이(height), 너비(width), 컬러 채널(channel)을 가진 구조로 사용

5D Tensor

- 5개의 축
- 비디오 데이터가 대표적인 사례
- 주로 샘플(samples), 프레임(frames), 높이(height), 너비(width), 컬러 채널(channel)을 가진 구조로 사용

Tensor Data Type

- 텐서의 기본 dtype
 - 정수형 텐서: int32
 - 실수형 텐서: float32
 - 문자열 텐서: string
- int32, float32, string 타입 외에도 float16, int8 타입 등이 존재
- 연산 시 텐서의 타입 일치 필요
- 타입변환에는 tf.cast() 사용

Tensor Calculation

```
In [13]: print(tf.constant(2) + tf.constant(2))
print(tf.constant(2) - tf.constant(2))
print(tf.add(tf.constant(2), tf.constant(2)))
print(tf.subtract(tf.constant(2), tf.constant(2)))
```

```
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(0, shape=(), dtype=int32)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(0, shape=(), dtype=int32)
```

```
In [14]: print(tf.constant(2) * tf.constant(2))
print(tf.constant(2) / tf.constant(2))
print(tf.multiply(tf.constant(2), tf.constant(2)))
print(tf.divide(tf.constant(2), tf.constant(2)))
```

```
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(1.0, shape=(), dtype=float64)
tf.Tensor(4, shape=(), dtype=int32)
tf.Tensor(1.0, shape=(), dtype=float64)
```

```
In [15]: print(tf.cast(tf.constant(2), tf.float32) + tf.constant(2.2))
```

```
tf.Tensor(4.2, shape=(), dtype=float32)
```



딥러닝 구조 및 학습

- 딥러닝 구조와 학습에 필요한 요소
 - 모델(네트워크)를 구성하는 레이어(layer)
 - 입력 데이터와 그에 대한 목적(결과)
 - 학습시에 사용할 피드백을 정의하는 손실 함수(loss function)
 - 학습 진행 방식을 결정하는 옵티마이저(optimizer)

레이어(Layer)

- 신경망의 핵심 데이터 구조
- 하나 이상의 텐서를 입력받아 하나 이상의 텐서를 출력하는 데 이터 처리 모듈
- 상태가 없는 레이어도 있지만, 대부분 **가중치(weight)**라는 레이어 상태를 가짐
- 가중치는 확률적 경사 하강법에 의해 학습되는 하나 이상의 텐서

Keras에서 사용되는 주요 레이어

- Dense
- Activation
- Flatten
- Input

```
In [16]: from tensorflow.keras.layers import Dense, Activation, Flatten, Input
```

Dense

- 완전연결계층(Fully-Connected Layer)
- 노드 수, 활성화 함수 등을 지정
- name을 통한 레이어 간 구분 가능
- 가중치 초기화(**kernel_initializer**)
 - 신경망의 성능에 큰 영향을 주는 요소
 - 보통 가중치의 초기값으로 0에 가까운 무작위 값 사용
 - 특정 구조의 신경망을 동일한 학습 데이터로 학습시키더라도, 가중치의 초기값에 따라 학습된 신경망의 성능 차이가 날 수 있음
 - 오차역전파 알고리즘은 기본적으로 경사하강법을 사용하기 때문에 최적해가 아닌 지역해에 빠질 가능성이 있음
 - Keras에서는 기본적으로 Glorot uniform 가중치(Xavier 분포 초기화), zero bias로 초기화
 - **Kernel_initializer** 인자를 통해 다른 가중치 초기화 지정 가능
 - Keras에서 제공하는 가중치 초기화 종류: <https://keras.io/api/layers/initializers/>

Dense

```
In [17]: Dense(10, activation = 'softmax')
```

```
Out[17]: <keras.layers.core.dense.Dense at 0x26af63003a0>
```

```
In [18]: Dense(10, activation = 'relu', name = 'Dense Layer')
```

```
Out[18]: <keras.layers.core.dense.Dense at 0x26af6300c10>
```

```
In [19]: Dense(10, kernel_initializer = 'he_normal', name = 'Dense Layer')
```

```
Out[19]: <keras.layers.core.dense.Dense at 0x26af63bb580>
```

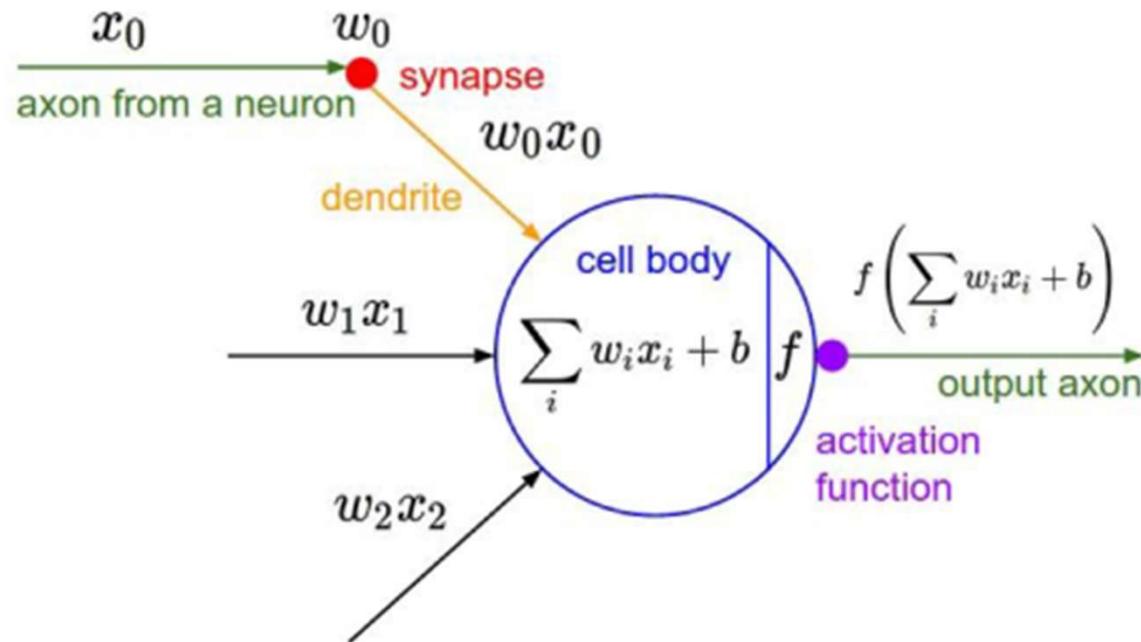


경희대학교
KYUNG HEE UNIVERSITY

Activation Function

- 입력 신호의 총합을 출력신호로 변환하는 함수를 일반적으로 Activation Function이라고 한다.
- 활성화(Activate)라는 이름에서 알 수 있듯이 활성화 함수란 입력 신호의 총합이 활성화를 일으키는지 정하는 역할을 한다.
- Dense layer에서 미리 활성화 함수를 지정할 수도 있지만 필요에 따라 별도 레이어를 만들어줄 수 있다.
- input과 weight의 내적 값에 bias를 더한 값을 얼마나 출력시킬지 정하기 위해 Activation Function을 사용한다.

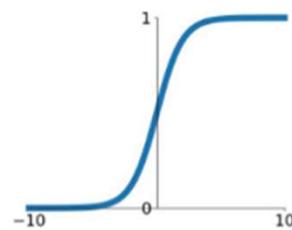
Activation Function



Activation Functions

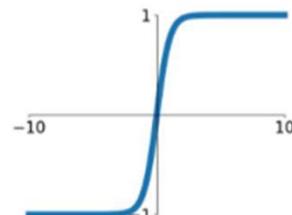
Sigmoid

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



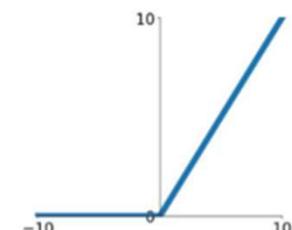
tanh

$$\tanh(x)$$



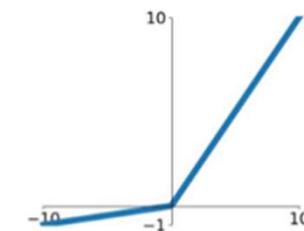
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

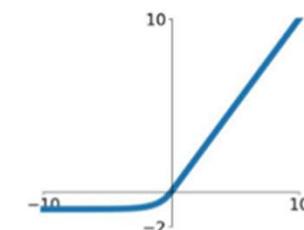


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



경희대학교
KYUNG HEE UNIVERSITY

Activation Function

- 경사하강법: 모델이 데이터를 잘 표현할 수 있도록 변화율을 사용하여 모델을 조금씩 조정하는 최적화 알고리즘이다. 즉, 어떤 손실함수가 정의되었을 때 손실함수 값이 최소가 되는 지점을 찾아가는 방법이다.
 - 직접 변화율을 계산(기울기 값)하는 방법
 - 손실함수를 미분(기울기 값)하는 방법
- Vanishing Gradient Problem: 기울기 값이 사라지는 문제 (loss function의 최솟값을 얻기 위해서는 loss function의 미분 값이 0에 가까워 지는 지점을 찾아야 하는데, 그 지점을 찾는데 cost가 많이 들거나 찾지 못하는 경우가 발생)
- Back Propagation: Cost를 최소화하기 위해 w 를 갱신해야 하며 이를 위해 선 Cost의 미분 값이 필요합니다. 이것이 ' $\partial f / \partial x$ ' 이고 이것을 구하는데 이용되는 것이 연쇄법칙인 것이다.

Activation Function

Sigmoid

- Sigmoid 함수는 음수 값을 0에 가깝게 표현하기 때문에 입력 값이 최종 레이어에서 미치는 영향이 적어지는 Vanishing Gradient Problem이 발생한다.
- Sigmoid 도함수 그래프에서 미분 계수를 보면 최대값이 0.25이다. 딥러닝에서 학습을 위해 Back-propagation을 계산하는 과정에서 활성화 함수의 미분 값을 곱하는 과정이 포함되는데, Sigmoid 함수의 경우 은닉층의 깊이가 깊으면 오차율을 계산하기 어렵다는 문제가 발생하기 때문에, Vanishing Gradient Problem이 발생한다.
- 즉, x 의 절대값이 커질수록 Gradient Backpropagation 시 미분 값이 소실될 가능성이 큰 단점이 있다.
- 이러한 문제 때문에 딥러닝 실무에서는 잘 사용되지 않지만, 미분 결과가 간결하고 사용하기 쉽다.
- 모든 실수 값을 0보다 크고 1보다 작은 미분 가능한 수로 변환하는 특징을 같이 때문에, Logistic Classification과 같은 분류 문제의 가설과 비용 함수(Cost Function)에 많이 사용된다.
- sigmoid()의 리턴 값이 확률 값이기 때문에 결과를 확률로 해석할 때 유용하다.

Activation Function

Tanh

- tanh 함수는 함수의 중심점을 0으로 옮겨 sigmoid가 갖고 있던 최적화 과정에서 느려지는 문제를 해결했다.
- 하지만 미분함수에 대해 일정 값 이상에서 미분 값이 소실되는 Vanishing Gradient Problem은 여전히 남아있다.

Activation Function

ReLU

- ReLU(Rectified Linear Unit, 경사함수)는 가장 많이 사용되는 활성화 함수 중 하나이다.
- Sigmoid와 tanh가 갖는 Gradient Vanishing 문제를 해결하기 위한 함수이다.
- x 가 0보다 크면 기울기가 1인 직선, 0보다 작으면 함수 값이 0이 된다. 이는 0보다 작은 값들에서 뉴런이 죽을 수 있는 단점을 야기한다.
- sigmoid, tanh 함수보다 학습이 빠르고, 연산 비용이 적고, 구현이 매우 간단하다는 특징이 있다.

Activation Function

Leaky ReLU

- Leaky ReLU는 ReLU가 갖는 Dying ReLU(뉴런이 죽는 현상) 을 해결하기 위해 나온 함수이다.
- 0.01이 아니라 매우 작은 값이라면 무엇이든 사용 가능하다.
- Leaky ReLU는 x 가 음수인 영역의 값에 대해 미분값이 0이 되지 않는다는 점을 제외하면 ReLU의 특성을 동일하게 갖는다.

Activation Function

PReLU

- Leaky ReLU와 거의 유사하지만 새로운 파라미터 α 를 추가해 x 가 음수인 영역에서도 기울기를 학습한다.

Activation Function

ELU

- Exponential Linear Unit은 ReLU의 모든 장점을 포함하며 Dying ReLU 문제를 해결했다.
- 출력 값이 거의 zero-centered에 가까우며, 일반적인 ReLU와 다르게 \exp 함수를 계산하는 비용이 발생한다.

Activation Function

Maxout

- ReLU의 장점을 모두 갖고, Dying ReLU 문제 또한 해결한다. 하지만 계산해야 하는 양이 많고 복잡하다는 단점이 있다.

Activation Function

softmax

- 소프트맥스 함수(softmax function)는 로지스틱 함수의 다차원 일 반화이다.
- 인공신경망에서 확률분포를 얻기 위한 마지막 활성함수로 많이 사용된다.
- softmax function formula: $p_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}, j = 1, 2, \dots, K$
 - K: 클래스 수(열 수)
 - z_j : 소프트맥스 함수의 입력값

Activation Function

Example

```
In [20]: dense = Dense(10, activation = 'relu', name = 'Dense layer')  
Activation(dense)
```

```
Out [20]: <keras.layers.core.activation.Activation at 0x19d04510d90>
```

Flatten

- 배치 크기(또는 데이터 크기)를 제외하고 데이터를 1차원으로 쭉 펼치는 작업
- 예시: $(128, 3, 2, 2) \rightarrow (128, 12)$

```
In [21]: Flatten(input_shape = (128, 3, 2, 2))
```

```
Out [21]: <keras.layers.core.flatten.Flatten at 0x19d045ac130>
```

Input

- 모델의 입력을 정의
- shape, dtype을 포함
- 하나의 모델은 여러 개의 입력을 가질 수 있음
- summary() 메소드를 통해서는 보이지 않음

```
In [22]: Input(shape = (28, 28), dtype = tf.float32)
```

```
Out [22]: <KerasTensor: shape=(None, 28, 28) dtype=float32 (created by layer 'input_1')>
```

```
In [23]: Input(shape = (8, ), dtype = tf.int32)
```

```
Out [23]: <KerasTensor: shape=(None, 8) dtype=int32 (created by layer 'input_2')>
```

Model

- 딥러닝 모델은 레이어로 만들어진 비순환 유향 그래프(Directed Acyclic Graph, DAG) 구조
- Sequential()
- 서브클래싱(Subclassing)
- 함수형 API

Sequential()

- 모델이 순차적인 구조로 진행할 때 사용
- 간단한 방법
 - Sequential 객체 생성 후, `add()`를 이용한 방법
 - Sequential 인자에 한번에 추가하는 방법
- 다중 입력 및 출력이 존재하는 등의 복잡한 모델을 구성할 수 없음

Sequential()

- 모델이 순차적인 구조로 진행할 때 사용
- 간단한 방법
 - Sequential 객체 생성 후, `add()`를 이용한 방법
 - Sequential 인자에 한번에 추가하는 방법
- 다중 입력 및 출력이 존재하는 등의 복잡한 모델을 구성할 수 없음

Sequential()

```
In [24]: from tensorflow.keras.layers import Dense, Input, Flatten  
from tensorflow.keras.models import Sequential, Model  
from tensorflow.keras.utils import plot_model
```

```
In [25]: model = Sequential()  
model.add(Input(shape = (28, 28)))  
model.add(Dense(300, activation = 'relu'))  
model.add(Dense(100, activation = 'relu'))  
model.add(Dense(10, activation = 'relu'))  
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 28, 300)	8700
dense_2 (Dense)	(None, 28, 100)	30100
dense_3 (Dense)	(None, 28, 10)	1010
<hr/>		
Total params: 39,810		
Trainable params: 39,810		
Non-trainable params: 0		

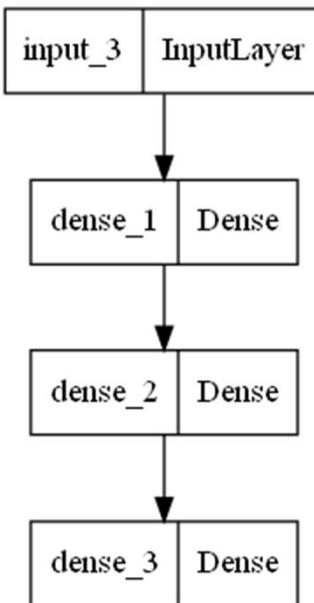


경희대학교
KYUNG HEE UNIVERSITY

Sequential()

```
In [26]: plot_model(model)
```

```
Out [26]:
```



함수형 API

- 가장 권장되는 방법
- 모델을 복잡하고, 유연하게 구성 가능
- 다중 입출력을 다룰 수 있음

함수형 API (Sequential)

```
In [29]: inputs = Input(shape=(28, 28, 1))
x = Flatten(input_shape = (28, 28, 1))(inputs)
x = Dense(300, activation = 'relu')(x)
x = Dense(100, activation = 'relu')(x)
x = Dense(10, activation = 'softmax')(x)

model = Model(inputs = inputs, outputs = x)
model.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
<hr/>		
input_5 (InputLayer)	[(None, 28, 28, 1)]	0
flatten_1 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 300)	235500
dense_5 (Dense)	(None, 100)	30100
dense_6 (Dense)	(None, 10)	1010
<hr/>		

```
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
```

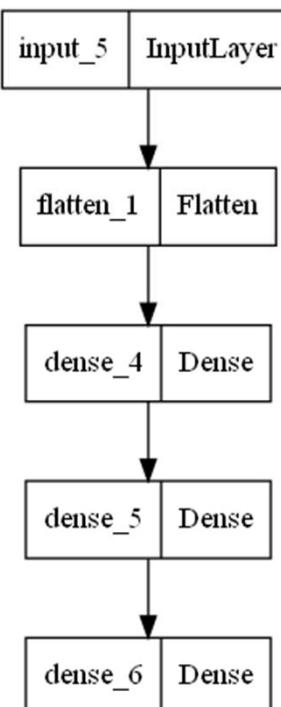


경희대학교
KYUNG HEE UNIVERSITY

함수형 API (Sequential)

```
In [30]: plot_model(model)
```

```
Out [30]:
```



함수형 API (Concatenate)

```
In [31]: from tensorflow.keras.layers import Concatenate

input_layer = Input(shape = (28, 28))
hidden1 = Dense(100, activation = 'relu')(input_layer)
hidden2 = Dense(30, activation = 'relu')(hidden1)
concat = Concatenate()([input_layer, hidden2])
output = Dense(1)(concat)

model = Model(inputs = [input_layer], outputs = [output])
model.summary()

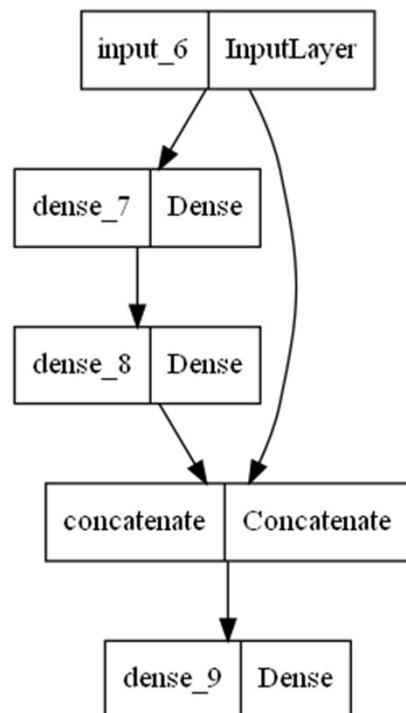
Model: "model_1"
-----  
Layer (type)          Output Shape         Param #  Connected to  
-----  
input_6 (InputLayer)   [(None, 28, 28)]      0          []  
dense_7 (Dense)        (None, 28, 100)       2900       ['input_6[0] [0]']  
dense_8 (Dense)        (None, 28, 30)        3030       ['dense_7[0] [0]']  
concatenate (Concatenate) (None, 28, 58)      0          ['input_6[0] [0]',  
                           'dense_8[0] [0]']  
dense_9 (Dense)        (None, 28, 1)        59         ['concatenate[0] [0]']  
-----  
Total params: 5,989  
Trainable params: 5,989  
Non-trainable params: 0
```



함수형 API (Concatenate)

```
In [32]: plot_model(model)
```

```
Out [32]:
```



함수형 API (Two outputs)

```
In [35]: input_ = Input(shape = (10, 10), name = 'input_')
hidden1 = Dense(100, activation = 'relu')(input_)
hidden2 = Dense(10, activation = 'relu')(hidden1)
output = Dense(1, activation = 'sigmoid', name = 'main_output')(hidden2)
sub_out = Dense(1, name = 'sum_output')(hidden2)

model = Model(inputs = [input_], outputs = [output, sub_out])
model.summary()
```

Model: "model_3"

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
input_ (InputLayer)	[(None, 10, 10)]	0	[]
dense_12 (Dense)	(None, 10, 100)	1100	['input_[0][0]']
dense_13 (Dense)	(None, 10, 10)	1010	['dense_12[0][0]']
main_output (Dense)	(None, 10, 1)	11	['dense_13[0][0]']
sum_output (Dense)	(None, 10, 1)	11	['dense_13[0][0]']
<hr/>			

Total params: 2,132

Trainable params: 2,132

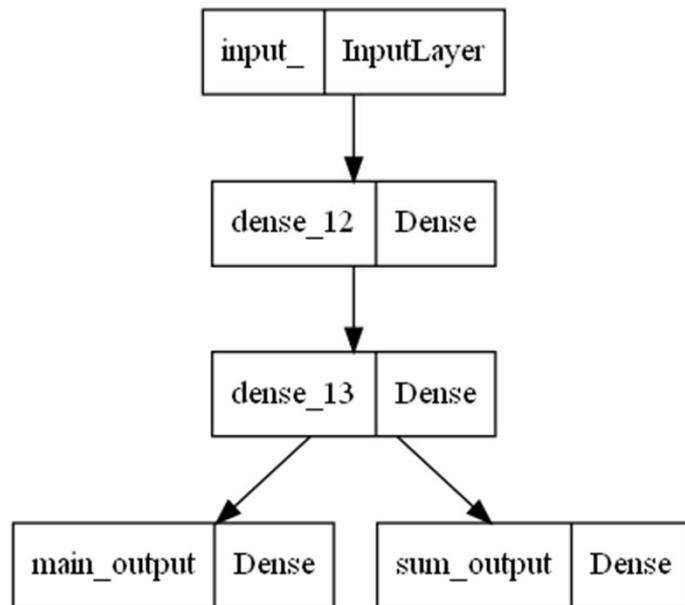
Non-trainable params: 0



함수형 API (Two outputs)

```
In [36]: plot_model(model)
```

```
Out [36]:
```



서브클래싱 (Subclassing)

- 커스터마이징에 최적화된 방법
- Model 클래스를 상속받아 Model이 포함하는 기능을 사용할 수 있음
 - fit(), evaluate(), predict()
 - save(), load()
- 주로 call() 메소드 안에서 원하는 계산 가능
 - For, if, 저수준 연산 등
- 권장되는 방법은 아니지만 어떤 모델의 구현 코드를 참고할 때, 해석할 수 있어야함.

서브클래싱 (Subclassing)

```
In [39]: class MyModel(Model):
    def __init__(self, units = 30, activation = 'relu', **kwargs):
        super(MyModel, self).__init__(**kwargs)
        self.dense_layer1 = Dense(300, activation = activation)
        self.dense_layer2 = Dense(100, activation = activation)
        self.dense_layer3 = Dense(units, activation = activation)
        self.output_layer = Dense(10, activation = 'softmax')

    def call(self, inputs):
        x = self.dense_layer1(inputs)
        x = self.dense_layer2(x)
        x = self.dense_layer3(x)
        x = self.output_layer(x)
        return x
```

모델 가중치 (Model Weight) 확인

```
In [40]: inputs = Input(shape = (28, 28, 1))
x = Flatten(input_shape = (28, 28, 1))(inputs)
x = Dense(300, activation = 'relu')(x)
x = Dense(100, activation = 'relu')(x)
x = Dense(10, activation = 'softmax')(x)

model = Model(inputs = inputs, outputs = x)
model.summary()

Model: "model_5"
-----  
Layer (type)          Output Shape         Param #
-----  
input_7 (InputLayer)   [(None, 28, 28, 1)]   0  
flatten_2 (Flatten)    (None, 784)           0  
dense_16 (Dense)      (None, 300)           235500  
dense_17 (Dense)      (None, 100)           30100  
dense_18 (Dense)      (None, 10)            1010  
-----  
Total params: 266,610  
Trainable params: 266,610  
Non-trainable params: 0
```

모델 가중치 (Model Weight) 확인

```
In [44]: weights, biases = hidden_2.get_weights()  
print(weights.shape)  
print(biases.shape)
```

```
(784, 300)  
(300,)
```

```
In [45]: print(weights)
```

```
[[-0.03597279  0.03609728  0.01270981 ...  0.00054117  0.02663758  
  0.06468211]  
 [-0.00830096 -0.06001972  0.03141721 ...  0.00659844 -0.0641862  
  0.02630989]  
 [ 0.06358416 -0.0388729   0.03559221 ...  0.06427467  0.02337152  
  0.03443405]  
 ...  
 [-0.04329812  0.04722911 -0.0687301  ... -0.01310849 -0.05222657  
  0.0294061 ]]  
 [-0.06089339  0.07370585 -0.06408777 ... -0.03420869  0.01072968  
 -0.03729089]  
 [ 0.03960556 -0.05056622 -0.06666388 ... -0.03884588 -0.06816068  
  0.0078054 ]]
```

```
In [46]: print(biases)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.  
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```



모델 컴파일 (compile)

- 모델을 구성한 후, 사용할 손실 함수(loss function), 옵티마이저(optimizer)를 지정

```
In [47]: model.compile(loss = 'sparse_categorical_crossentropy',
                     optimizer = 'sgd',
                     metrics = ['accuracy'])
```

손실 함수(Loss Function)

- 학습이 진행되면서 해당 과정이 얼마나 잘 되고 있는지 나타내는 지표
- 모델이 훈련되는 동안 최소화될 값으로 주어진 문제에 대한 성공 지표
- 손실 함수에 따른 결과를 통해 학습 파라미터를 조정
- 최적화 이론에서 최소화 하고자 하는 함수
- 미분 가능한 함수 사용
- Keras에서 주요 손실 함수 제공
 - `sparse_categorical_crossentropy`: 클래스가 배타적 방식으로 구분, 즉 $(0, 1, 2, \dots, 9)$ 와 같은 방식으로 구분되어 있을 때 사용
 - `categorical_crossentropy`: 클래스가 one-hot encoding 방식으로 되어 있을 때 사용
 - `binary_crossentropy`: 이진 분류를 수행할 때 사용

손실 함수(Loss Function)

평균절대오차(Mean Absolute Error, MAE)

- 오차가 커져도 손실함수가 일정하게 증가
- 이상치(outlier)에 강건함(Robust)
 - 데이터에서 [입력-정답]관계가 적절하지 않은 경우에, 좋은 추정을 하더라도 오차가 발생하는 경우가 있음.
 - 해당 이상치에 해당하는 지점에서 손실 함수의 최소값으로 가는 정도의 영향력이 크지 않음.
- 회귀 (Regression)에 자주 사용
- 평균절대오차 식: $Error = \frac{1}{n} \sum_{i=1}^n |y_i - \tilde{y}_i|$
 - y_i : 학습 데이터의 i 번째 정답
 - \tilde{y}_i : 학습 데이터의 입력으로 추정한 i 번째 출력

손실 함수(Loss Function)

평균제곱오차(Mean Squared Error, MSE)

- 가장 많이 쓰이는 손실 함수 중 하나
- 오차가 커질수록 손실함수가 빠르게 증가
 - 정답과 예측한 값의 차이가 클수록 더 많은 페널티를 부여
- 회귀 (Regression)에 쓰임
- 평균제곱오차 식: $Error = \frac{1}{n} \sum_{i=1}^n (y_i - \tilde{y}_i)^2$
 - y_i : 학습 데이터의 i 번째 정답
 - \tilde{y}_i : 학습 데이터의 입력으로 추정한 i 번째 출력

one-hot encoding

- 범주형 변수를 표현할 때 사용
- 가변수(Dummy Variable)이라고도 함
- 정답인 레이블을 제외하고 0으로 처리

Label Encoding			One Hot Encoding			
Food Name	Categorical #	Calories	Apple	Chicken	Broccoli	Calories
Apple	1	95	1	0	0	95
Chicken	2	231	0	1	0	231
Broccoli	3	50	0	0	1	50

Cross Entropy Error, CEE

- 설명 link
- 이진 분류 (Binary Classification), 다중 클래스 분류 (Multi Class Classification)
- 소프트맥스 (softmax)와 원-핫 인코딩 (ont-hot encoding) 사이의 출력 간 거리를 비교
- 정답인 클래스에 대해서만 오차를 계산
- 정답을 맞추면 오차가 0, 틀리면 그 차이가 클수록 오차가 무한히 커짐

Cross Entropy Error, CEE

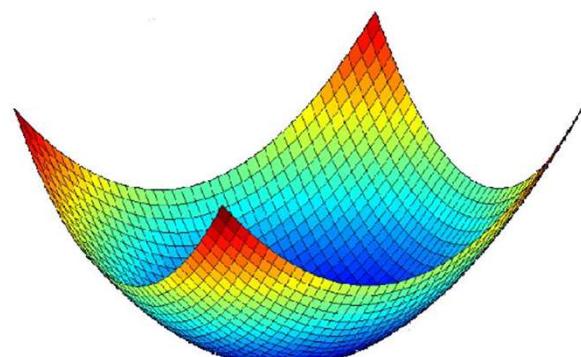
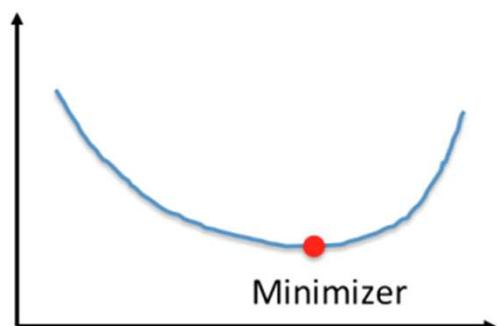
- CEE Formula: $Error = -\frac{1}{N} \sum_n \sum_i y_i \log \tilde{y}_i$
- y_i : 학습 데이터의 i 번째 정답
- \tilde{y}_i : 학습 데이터의 입력으로 추정한 i 번째 출력
- N : 전체 데이터의 개수
- i : 데이터 하나당 클래스 개수
- 정답 레이블(y_i)는 one-hot encoding으로 정답인 인덱스에만 1이고, 나머지는 모두 0이어서 다음과 같이 나타낼 수 있음
 - $E = -\log \tilde{y}_i$
- 정답에 가까워질수록 오차 값은 작아진다.
- 학습 시, one-hot encoding에 의해 정답 인덱스만 살아남아 비교하지만, 정답이 아닌 인덱스들도 학습에 영향을 미침. 왜냐하면 다중 클래스 분류는 소프트맥스 (softmax)함수를 통해 전체 항들을 모두 다루기 때문

Optimizer

- 손실 함수를 기반으로 모델이 어떻게 업데이트되어야 하는지 결정 (특정 종류의 확률적 경사 하강법 구현)
- Keras에서 여러 옵티마이저 제공
- keras.optimizer.SGD(): 기본적인 확률적 경사하강법
- keras.optimizer.Adam(): 자주 사용되는 옵티마이저
- Keras에서 사용되는 옵티마이저
- 보통 옵티마이저의 튜닝을 위해 따로 객체를 생성하여 컴파일

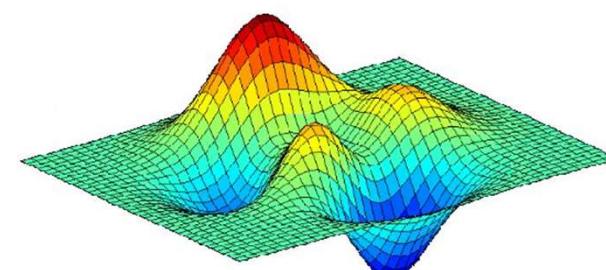
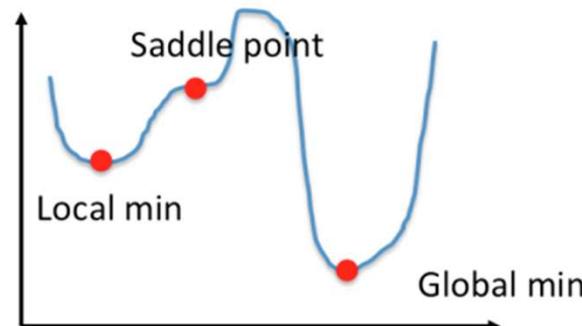
Convex Function & Non-Convex Function

Convex



convex function

Non-Convex



non-convex function



경희대학교
KYUNG HEE UNIVERSITY

Convex Function & Non-Convex Function

- 볼록함수(Convex Function)

- 어떤 지점에서 시작하더라도 최적값(손실함수가 최소로하는 점)에 도달할 수 있음

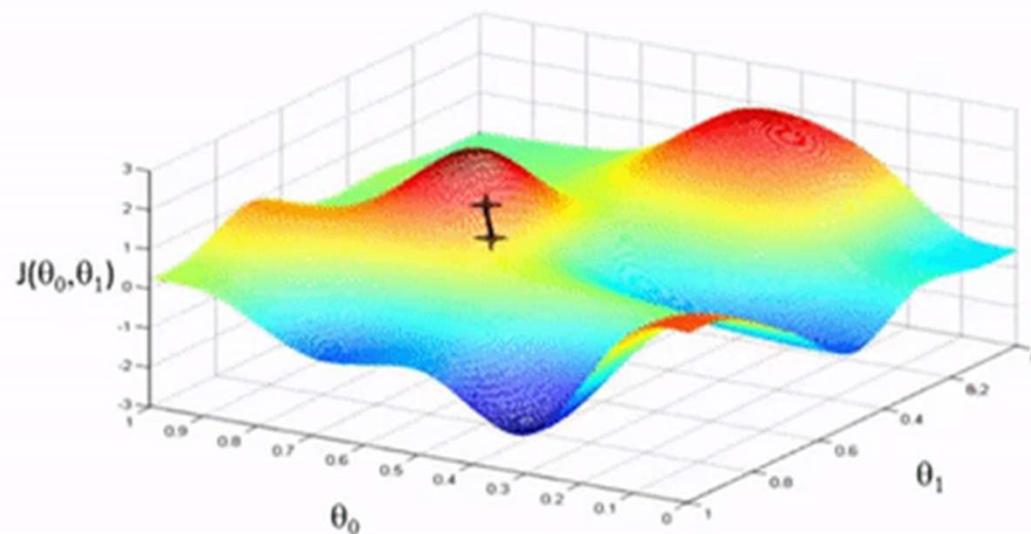
- 비볼록함수(Non-Convex Function)

- 비볼록 함수는 시작점 위치에 따라 다른 최적값에 도달할 수 있음

경사하강법 (Gradient Decent)

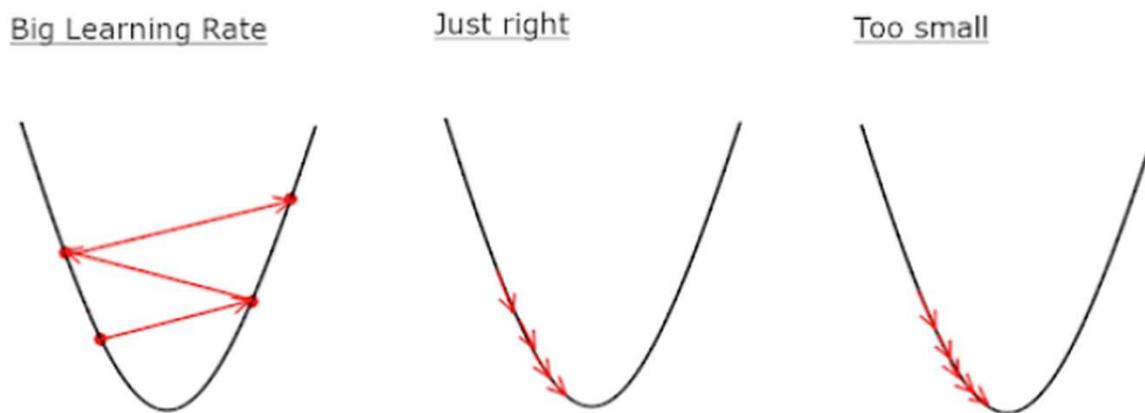
- 경사하강법의 과정
- 경사하강법은 한 스텝마다의 미분값에 따라 이동하는 방향을 결정
- $f(x)$ 의 값이 변하지 않을 때까지 반복
- $$x_n = x_{n-1} - \eta \frac{\delta f(x)}{\delta x}$$
- 즉, 미분 값이 0인 지점을 찾는 것

경사하강법 (Gradient Decent)



학습률 (learning rate)

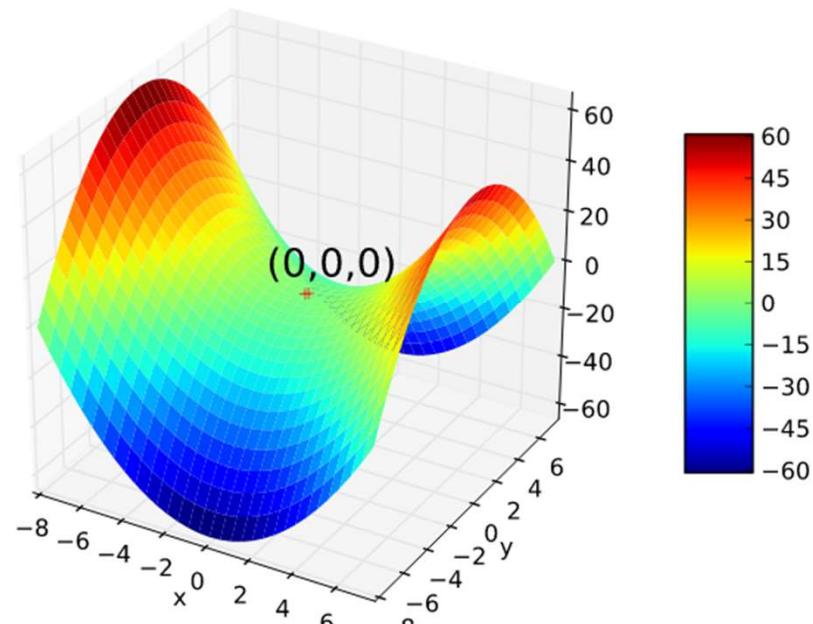
- 적절한 학습률을 지정해야 최저점에 잘 도달할 수 있음
- 학습률이 너무 크면 발산하고, 너무 작으면 학습이 오래 걸리거나 최저점에 도달하지 않음



경희대학교
KYUNG HEE UNIVERSITY

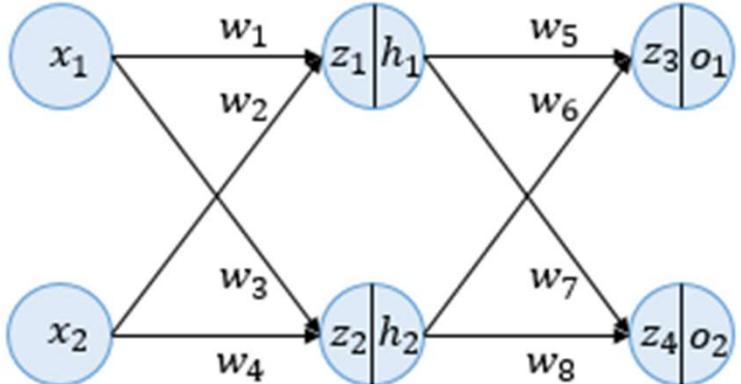
안장점 (Saddle Point)

- 기울기가 0이지만 극값이 되지 않음
- 경사하강법은 안장점에서 벗어나지 못함



Propagation

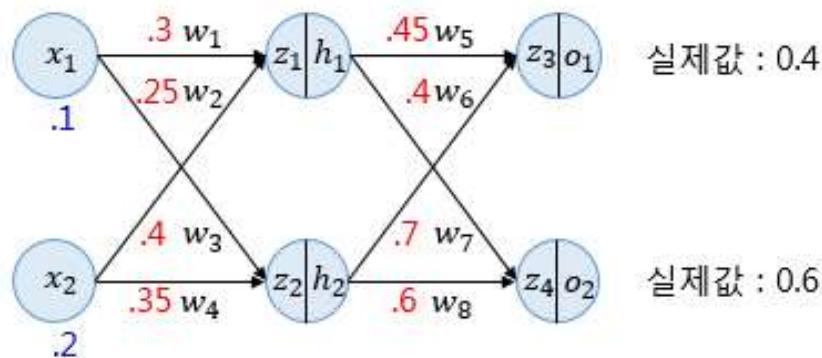
Neural Network Setting



- x_i : 입력 값
- z_i : 이전 층의 모든 입력이 각각의 가중치와 곱해진 값들이 모두 더해진 가중합
- h_i : 활성화 함수를 거친 이후의 값
- o_i : 출력 값
- w_i : 가중치
- 활성화 함수: *sigmoid*

Propagation

Forward Propagation (step 1, input layer to hidden layer)

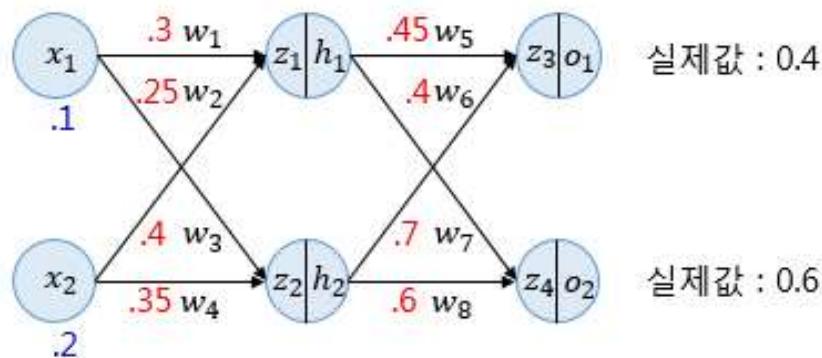


- 파란색 숫자: 입력 값
- 빨간색 숫자: 각 가중치 값
- 각 입력은 입력층에서 은닉층으로 향하면서 각 입력에 해당하는 가중치와 곱해지고 가중합으로 계산되어 은닉층 뉴런의 *sigmoid* 함수 입력 값이 된다.
- $z_1 = w_1x_1 + w_2x_2 = 0.08$
- $z_2 = w_3x_1 + w_4x_2 = 0.11$



Propagation

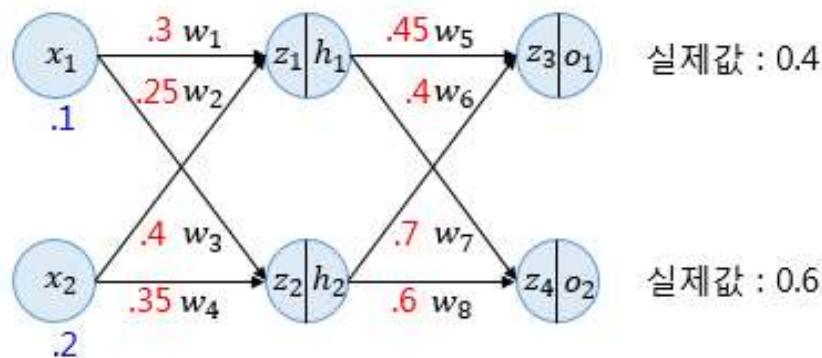
Forward Propagation (step 2, activation function at hidden layer's node)



- z_1 과 z_2 는 각각의 은닉층 뉴런에서 *sigmoid* 함수를 거쳐 h_1 , h_2 를 반환한다.
- $h_1 = \text{sigmoid}(z_1) = 0.519989$
- $h_2 = \text{sigmoid}(z_2) = 0.527472$

Propagation

Forward Propagation (step 3, 4, hidden layer to output layer)



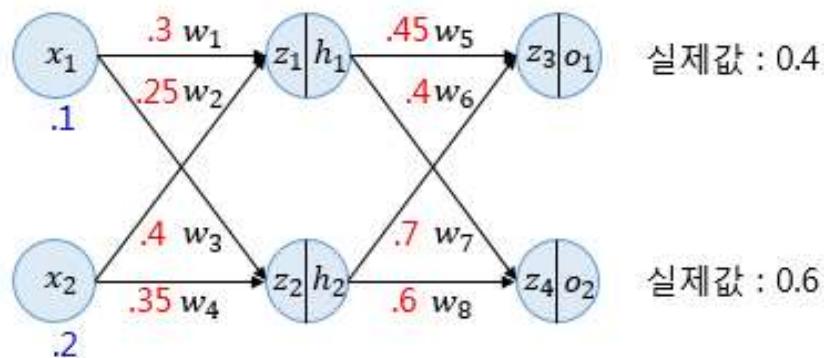
- h_1 과 h_2 는 다시 출력층의 뉴런으로 향해 각각의 가중치와 곱해지고 다시 가중합되어 sigmoid함수의 입력값이 된다.

- $$z_3 = w_5 h_1 + w_6 h_2 = 0.444984$$
- $$z_4 = w_7 h_1 + w_8 h_2 = 0.680476$$

- 출력층의 sigmoid함수값은 예측값이 된다.
- $$o_1 = \text{sigmoid}(z_3) = 0.609446$$
- $$o_2 = \text{sigmoid}(z_4) = 0.663845$$

Propagation

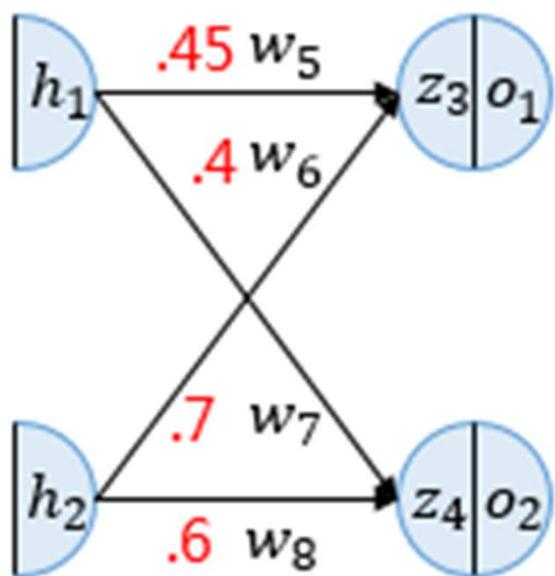
Forward Propagation (step 5, Calculate Error using MSE)



- $E_{O_1} = MSE(o_1) = 0.021934$
- $E_{O_2} = MSE(o_2) = 0.002038$
- $E_{total} = E_{O_1} + E_{O_2} = 0.023972$

Propagation

Back Propagation (step 1, output layer to hidden layer)



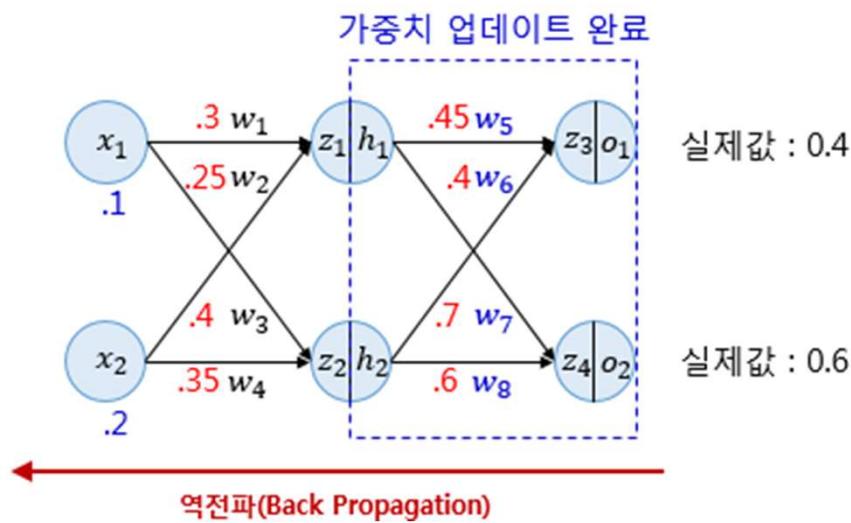
- 경사하강법(SGD)을 수행하려면 w_5 를 업데이트 하기 위해서 $\frac{\delta E_{total}}{\delta w_5}$ 을 계산해야 한다.
- $\frac{\delta E_{total}}{\delta w_5}$ 을 계산하기 위해 연쇄 법칙을 이용한다.
- $$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta o_1} \times \frac{\delta o_1}{\delta z_3} \times \frac{\delta z_3}{\delta w_5}$$
- $w_5^+ = w_5 - \alpha \frac{\delta E_{total}}{\delta w_5}$, α = learning rate
- 마찬가지 원리로 w_6^+, w_7^+, w_8^+ 를 계산할 수 있다.



경희대학교
KYUNG HEE UNIVERSITY

Propagation

Back Propagation (step 2, hidden layer to input layer)



- 경사하강법(SGD)을 수행하려면 w_1 를 업데이트 하기 위해서 $\frac{\delta E_{total}}{\delta w_1}$ 을 계산 해야 한다.
- $\frac{\delta E_{total}}{\delta w_1}$ 을 계산하기 위해 연쇄 법칙을 이용한다.
- $$\frac{\delta E_{total}}{\delta w_1} = \frac{\delta E_{total}}{\delta h_1} \times \frac{\delta h_1}{\delta z_1} \times \frac{\delta z_1}{\delta w_1}$$
- $$w_1^+ = w_1 - \alpha \frac{\delta E_{total}}{\delta w_1}, \alpha = \text{learning rate}$$
- 마찬가지 원리로 w_2^+, w_3^+, w_4^+ 를 계산 할 수 있다.

Model training, predict and evaluate

- **fit()**
 - X: 학습 데이터
 - y: 학습 데이터 정답 레이블
 - epochs: 학습 회수
 - batch_size: 단일 배치에 있는 학습 데이터의 크기
 - iteration: 1회의 epoch을 batch_size로 나누어 실행하는 횟수(자동으로 계산)
 - validation_data: 검증을 위한 데이터
- **evaluate()**
 - 테스트 데이터를 이용한 평가
- **predict()**
 - 임의의 데이터를 이용해 예측

MNIST 딥러닝 모델 예제

- 손으로 쓴 숫자들로 이루어진 이미지 데이터셋
- 기계 학습 분야의 트레이닝 및 테스트에 널리 사용되는 데이터
- keras.datasets에 기본으로 포함되어 있는 데이터셋

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9



MNIST 딥러닝 모델 예제

모듈 임포트

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras import models
from tensorflow.keras.layers import Dense, Input, Flatten
from tensorflow.keras.utils import to_categorical, plot_model

from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
```



MNIST 딥러닝 모델 예제

데이터 로드 및 전처리

```
In [49]: tf.random.set_seed(111)

(X_train_full, y_train_full), (X_test, y_test) = load_data(path = 'mnist.npz')

X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full,
                                                test_size = 0.3,
                                                random_state = 111)
```

```
In [50]: num_x_train = (X_train.shape[0])
num_x_val = (X_val.shape[0])
num_x_test = (X_test.shape[0])

print('학습 데이터: {} 레이블: {}'.format(X_train_full.shape, y_train_full.shape))
print('학습 데이터: {} 레이블: {}'.format(X_train.shape, y_train.shape))
print('검증 데이터: {} 레이블: {}'.format(X_val.shape, y_val.shape))
print('테스트 데이터: {} 레이블: {}'.format(X_test.shape, y_test.shape))
```

```
학습 데이터: (60000, 28, 28)    레이블: (60000,)
학습 데이터: (42000, 28, 28)    레이블: (42000,)
검증 데이터: (18000, 28, 28)    레이블: (18000,)
테스트 데이터: (10000, 28, 28)  레이블: (10000,)
```



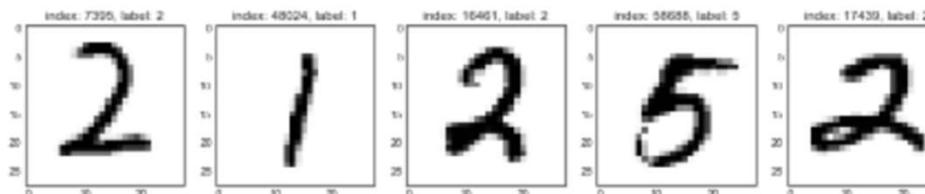
MNIST 딥러닝 모델 예제

데이터 로드 및 전처리

```
In [51]: num_sample = 5
random_indices = np.random.randint(60000, size = num_sample)

plt.figure(figsize = (15, 3))
for i, idx in enumerate(random_indices):
    img = X_train_full[idx, :, :]
    label = y_train_full[idx]

    plt.subplot(1, num_sample, i+1)
    plt.imshow(img)
    plt.title('index: {}, label: {}'.format(idx, label))
```



```
In [52]: # regularization
X_train = X_train / 255.
X_val = X_val / 255.
X_test = X_test / 255.

# one-hot encoding
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)
```



MNIST 딥러닝 모델 예제

모델 구성 (Sequential)

```
In [53]: model = Sequential([Input(shape = (28, 28), name = 'input'),
                           Flatten(input_shape = (28, 28), name = 'flatten'),
                           Dense(128, activation = 'relu', name = 'dense1'),
                           Dense(64, activation = 'relu', name = 'dense2'),
                           Dense(32, activation = 'relu', name = 'dense3'),
                           Dense(10, activation = 'softmax', name = 'output')])
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense1 (Dense)	(None, 128)	100480
dense2 (Dense)	(None, 64)	8256
dense3 (Dense)	(None, 32)	2080
output (Dense)	(None, 10)	330

=====
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0



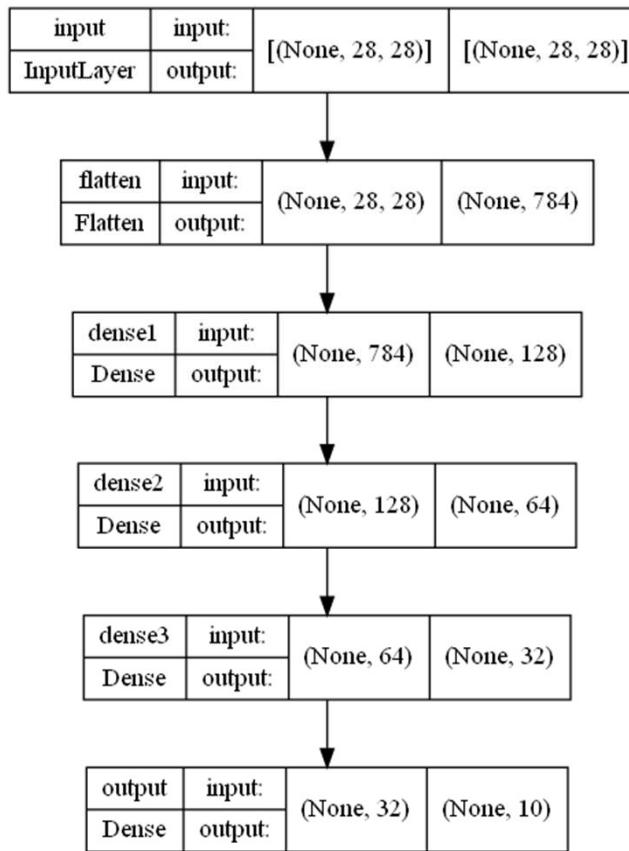
경희대학교
KYUNG HEE UNIVERSITY

MNIST 딥러닝 모델 예제

모델 구성 (Sequential)

```
In [55]: plot_model(model, show_shapes = True)
```

```
Out [55]:
```



경희대학교
KYUNG HEE UNIVERSITY

MNIST 딥러닝 모델 예제

모델 컴파일 및 학습

모델 컴파일 및 학습

```
In [56]: model.compile(loss = 'categorical_crossentropy',
                     optimizer = 'sgd',
                     metrics = ['accuracy'])

In [57]: print(y_train.shape, y_val.shape)
(42000, 10) (18000, 10)

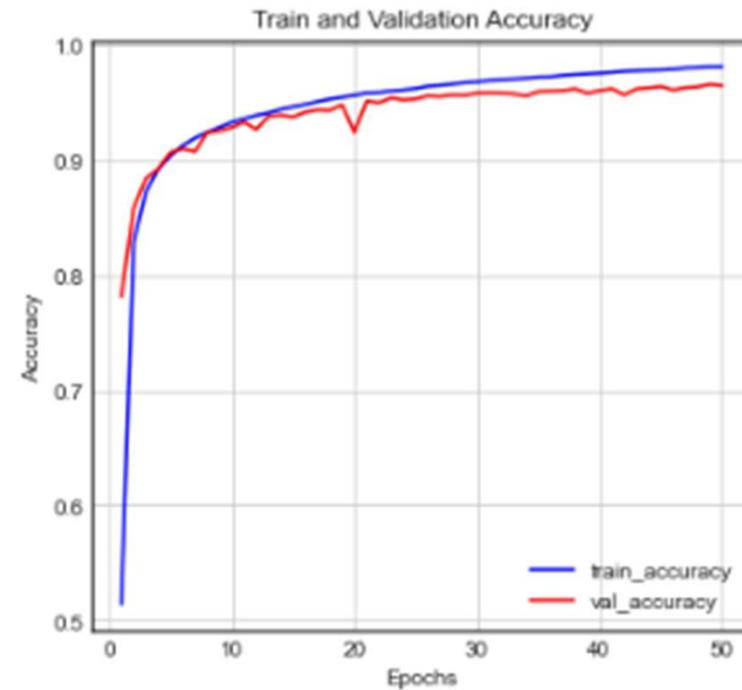
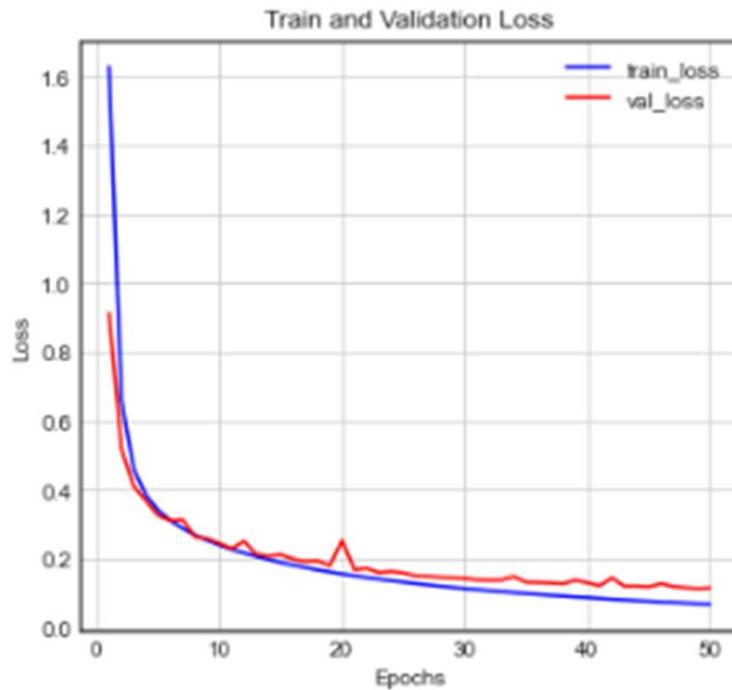
In [58]: history = model.fit(X_train, y_train,
                         epochs = 50,
                         batch_size = 128,
                         validation_data = (X_val, y_val))

0.9782 - val_loss: 0.1208 - val_accuracy: 0.9629
Epoch 45/50
329/329 [=====] - 1s 3ms/step - loss: 0.0768 - accuracy: 0.9785 - val_loss: 0.1188 - val_accuracy: 0.9641
Epoch 46/50
329/329 [=====] - 1s 2ms/step - loss: 0.0747 - accuracy: 0.9791 - val_loss: 0.1277 - val_accuracy: 0.9611
Epoch 47/50
329/329 [=====] - 1s 2ms/step - loss: 0.0733 - accuracy: 0.9802 - val_loss: 0.1197 - val_accuracy: 0.9631
Epoch 48/50
329/329 [=====] - 1s 2ms/step - loss: 0.0711 - accuracy: 0.9806 - val_loss: 0.1162 - val_accuracy: 0.9638
Epoch 49/50
329/329 [=====] - 1s 2ms/step - loss: 0.0696 - accuracy: 0.9814 - val_loss: 0.1135 - val_accuracy: 0.9659
Epoch 50/50
329/329 [=====] - 1s 2ms/step - loss: 0.0679 - accuracy: 0.9814 - val_loss: 0.1156 - val_accuracy: 0.9648
```



MNIST 딥러닝 모델 예제

validation loss and accuracy according to epoch



경희대학교
KYUNG HEE UNIVERSITY

MNIST 딥러닝 모델 예제

모델 평가 및 예측

```
In [61]: model.evaluate(X_test, y_test)
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.1145 - accuracy: 0.9668
```

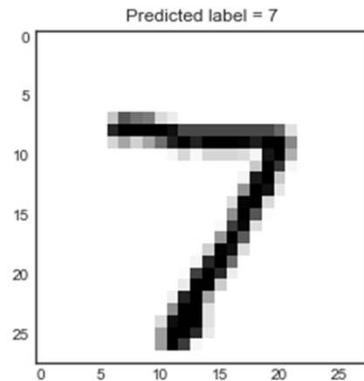
```
Out[61]: [0.11454666405916214, 0.9667999744415283]
```

```
In [62]: pred_ys = model.predict(X_test)
print(pred_ys.shape)

np.set_printoptions(precision = 7) #소수점 반올림
print(pred_ys[0])
```

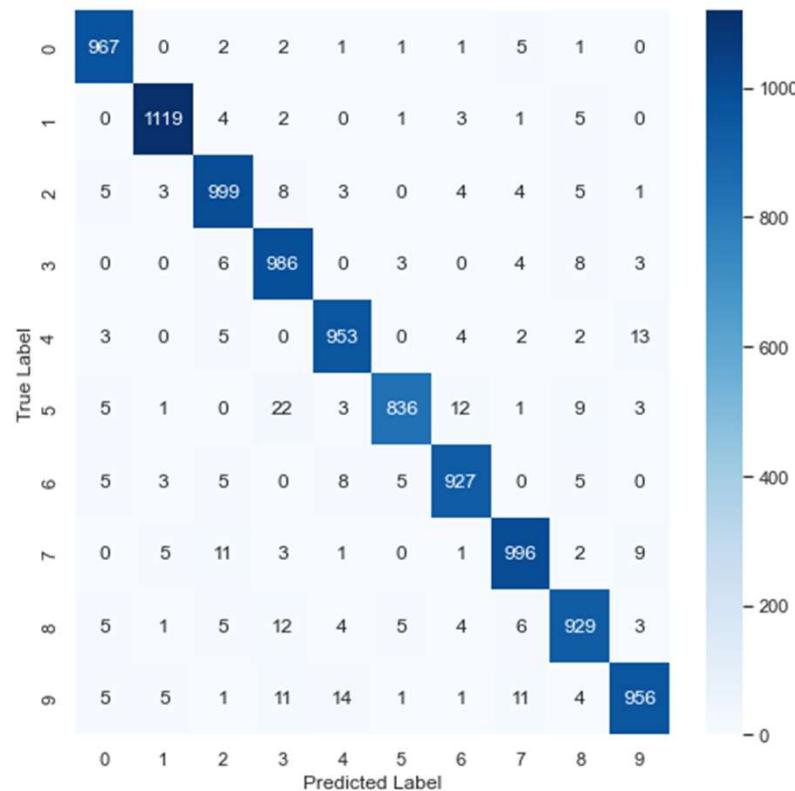
```
(10000, 10)
[1.6704307e-07 4.5911904e-07 4.8955421e-06 9.8328944e-04 2.3565173e-08
 5.7932522e-07 3.3495626e-11 9.9900454e-01 3.1884059e-07 5.6777749e-06]
```

```
In [63]: arg_pred_y = np.argmax(pred_ys, axis = 1) #np.argmax: 주어진 배열의 최댓값 인덱스 반환
plt.imshow(X_test[0])
plt.title('Predicted label = {}'.format(arg_pred_y[0]))
plt.show()
```



MNIST 딥러닝 모델 예제

모델 평가 및 예측



MNIST 딥러닝 모델 예제

모델 평가 및 예측

	precision	recall	f1-score	support
0	0.97	0.99	0.98	980
1	0.98	0.99	0.99	1135
2	0.96	0.97	0.97	1032
3	0.94	0.98	0.96	1010
4	0.97	0.97	0.97	982
5	0.98	0.94	0.96	892
6	0.97	0.97	0.97	958
7	0.97	0.97	0.97	1028
8	0.96	0.95	0.96	974
9	0.97	0.95	0.96	1009
accuracy			0.97	10000
macro avg	0.97	0.97	0.97	10000
weighted avg	0.97	0.97	0.97	10000

MNIST 딥러닝 모델 예제

모델 저장과 복원

- `save()`
- `load_model()`
- Sequential API, 함수형 API에서는 모델의 저장 및 로드가 가능하지만 서브클래싱 방식으로는 할 수 없음
- 서브클래싱 방식은 `save_weights()`와 `load_weights()`를 이용해 모델의 파라미터만 저장 및 로드
- JSON 형식
 - `model.to_json()`
 - `tf.keras.models.model_from_json(file_path)`
- YAML로 직렬화
 - `Model.to_yaml()`
 - `tf.keras.models.model_from_yaml(file_path)`

MNIST 딥러닝 모델 예제

모델 저장과 복원

```
In [66]: model.save('mnist_model.h5')
```

```
In [67]: loaded_model = models.load_model('mnist_model.h5')
loaded_model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
<hr/>		
flatten (Flatten)	(None, 784)	0
dense1 (Dense)	(None, 128)	100480
dense2 (Dense)	(None, 64)	8256
dense3 (Dense)	(None, 32)	2080
output (Dense)	(None, 10)	330
<hr/>		

Total params: 111,146

Trainable params: 111,146

Non-trainable params: 0



경희대학교
KYUNG HEE UNIVERSITY

MNIST 딥러닝 모델 예제

모델 저장과 복원

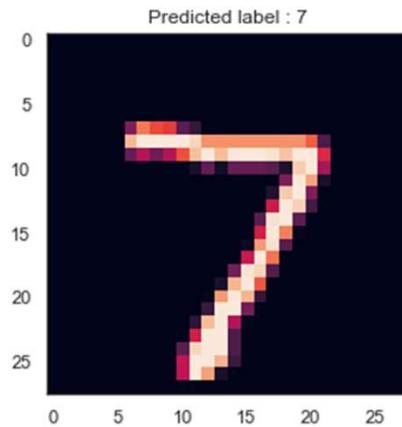
```
In [68]: pred_ys2 = loaded_model.predict(X_test)
print(pred_ys2.shape)

np.set_printoptions(precision=7)
print(pred_ys[0])

(10000, 10)
[1.6704307e-07 4.5911904e-07 4.8955421e-06 9.8328944e-04 2.3565173e-08
 5.7932522e-07 3.3495626e-11 9.9900454e-01 3.1884059e-07 5.6777749e-06]
```

```
In [69]: arg_pred_y2 = np.argmax(pred_ys2, axis = 1)

plt.imshow(X_test[0])
plt.title('Predicted label : {}'.format(arg_pred_y2[0]))
plt.show()
```



MNIST 딥러닝 모델 예제

Callbacks

```
In [70]: from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, TensorBoard
```

- fit() 함수의 callbacks 매개변수를 사용하여 케라스가 훈련의 시작이나 끝에 호출할 객체 리스트를 지정할 수 있음
- 여러 개 사용 가능
- ModelCheckpoint
 - `tf.keras.callbacks.ModelCheckpoint`
 - 정기적으로 모델의 체크포인트를 저장하고, 문제가 발생할 때 복구하는데 사용
- EarlyStopping
 - `tf.keras.callbacks.EarlyStopping`
 - 검증 성능이 한동안 개선되지 않을 경우 학습을 중단할 때 사용
- LearningRateScheduler
 - `tf.keras.callbacks.LearningRateScheduler`
 - 최적화를 하는 동안 learning rate를 동적으로 변경할 때 사용
- TensorBoard
 - `tf.keras.callbacks.TensorBoard`
 - 모델의 경과를 모니터링할 때 사용



MNIST 딥러닝 모델 예제

Callbacks, ModelCheckpoint

```
In [71]: check_point_cb = ModelCheckpoint('keras_mnist_model.h5')
history = model.fit(X_train, y_train, epochs = 10,
                     callbacks = check_point_cb)
```

```
Epoch 1/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0766 - accuracy: 0.9781
Epoch 2/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0709 - accuracy: 0.9790
Epoch 3/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0645 - accuracy: 0.9810
Epoch 4/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0597 - accuracy: 0.9823
Epoch 5/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0546 - accuracy: 0.9844
Epoch 6/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0508 - accuracy: 0.9852
Epoch 7/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0470 - accuracy: 0.9868
Epoch 8/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0435 - accuracy: 0.9874
Epoch 9/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0399 - accuracy: 0.9887
Epoch 10/10
1313/1313 [=====] - 1s 1ms/step - loss: 0.0368 - accuracy: 0.9900
```



MNIST 딥러닝 모델 예제

Callbacks, EarlyStopping

```
In [73]: check_point_cb = ModelCheckpoint('keras_mnist_model.h5', save_best_only = True)
early_stopping_cb = EarlyStopping(patience = 3, monitor = 'val_loss',
                                  restore_best_weights = True)
history = model.fit(X_train, y_train, epochs = 10,
                     validation_data = (X_val, y_val),
                     callbacks = (check_point_cb, early_stopping_cb))

Epoch 1/10
1313/1313 [=====] - 2s 2ms/step - loss: 0.0142 - accuracy: 0.9974 - val_loss: 0.1120 - val_accuracy: 0.9704
Epoch 2/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0127 - accuracy: 0.9979 - val_loss: 0.1120 - val_accuracy: 0.9710
Epoch 3/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0115 - accuracy: 0.9983 - val_loss: 0.1077 - val_accuracy: 0.9719
Epoch 4/10
1313/1313 [=====] - 2s 2ms/step - loss: 0.0104 - accuracy: 0.9988 - val_loss: 0.1126 - val_accuracy: 0.9716
Epoch 5/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0094 - accuracy: 0.9990 - val_loss: 0.1096 - val_accuracy: 0.9732
Epoch 6/10
1313/1313 [=====] - 2s 1ms/step - loss: 0.0089 - accuracy: 0.9989 - val_loss: 0.1124 - val_accuracy: 0.9720
```

MNIST 딥러닝 모델 예제

Callbacks, LearningRateScheduler

```
In [74]: def scheduler(epoch, learning_rate):
    if epoch < 10:
        return learning_rate
    else:
        return learning_rate * tf.math.exp(-0.1)
```

```
In [75]: round(model.optimizer.lr.numpy(), 5)
```

```
Out[75]: 0.01
```

```
In [76]: lr_scheduler_cb = LearningRateScheduler(scheduler)

history = model.fit(X_train, y_train, epochs = 15,
                     callbacks = [lr_scheduler_cb], verbose = 0)

round(model.optimizer.lr.numpy(), 5)
```

```
Out[76]: 0.00607
```

딥러닝 학습 기술

Using IMDB Dataset

- 영화 사이트 IMDB의 리뷰 데이터
- 텍스트 분류, 감성 분류를 위해 자주 사용하는 데이터
- 리뷰 텍스트와 리뷰가 긍정인 경우 1을 부정인 경우 0으로 표시한 레이블
- 케라스에서는 IMDB 영화 리뷰 데이터를 `imdb.load_data()` 함수를 통해 다운로드 가능

딥러닝 학습 기술

Using IMDB Dataset, Data Preprocessing

```
In [80]: from tensorflow.keras.datasets import imdb
import numpy as np

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 10000)

def vectorize_seq(seqs, dim = 10000):
    results = np.zeros((len(seqs), dim))
    for i, seq in enumerate(seqs):
        results[i, seq] = 1.

    return results

X_train = vectorize_seq(train_data)
X_test = vectorize_seq(test_data)

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

딥러닝 학습 기술

Using IMDB Dataset, Model Configuration

```
In [81]: import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([Dense(16, activation = 'relu', input_shape = (10000, ), name = 'input'),
                    Dense(16, activation = 'relu', name = 'hidden'),
                    Dense(1, activation = 'sigmoid', name = 'output')])

model.compile(optimizer = 'rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['acc'])

model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
input (Dense)	(None, 16)	160016
hidden (Dense)	(None, 16)	272
output (Dense)	(None, 1)	17

=====

Total params: 160,305

Trainable params: 160,305

Non-trainable params: 0

=====



딥러닝 학습 기술

Using IMDB Dataset, Model Training

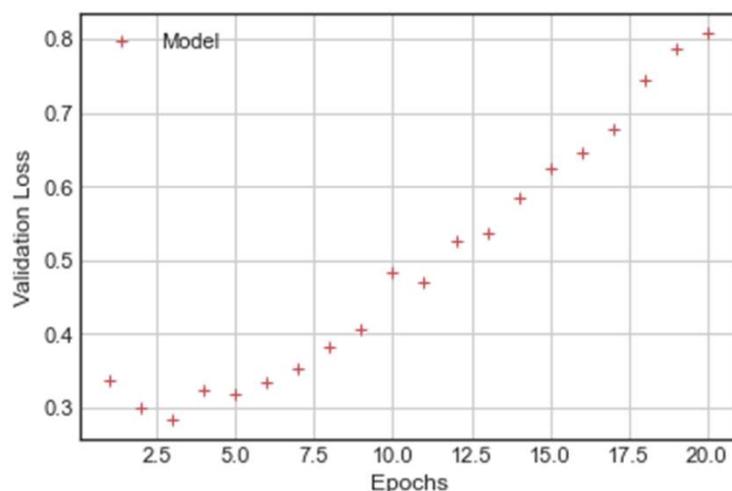
```
In [82]: model_hist = model.fit(X_train, y_train,  
                           epochs = 20,  
                           batch_size = 512,  
                           validation_data = (X_test, y_test))
```

딥러닝 학습 기술

Using IMDB Dataset, Evaluate

```
In [83]: import matplotlib.pyplot as plt  
plt.style.use('seaborn-white')
```

```
epochs = range(1, 21)  
model_val_loss = model_hist.history['val_loss']  
plt.plot(epochs, model_val_loss, 'r+', label = 'Model')  
plt.xlabel('Epochs')  
plt.ylabel('Validation Loss')  
plt.legend()  
plt.grid()  
plt.show()
```

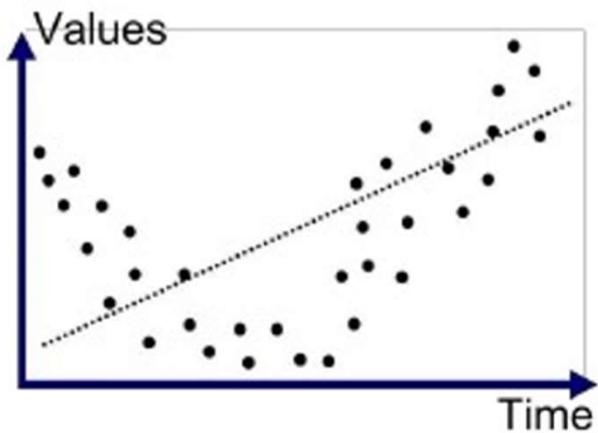


3 epoch 이후로 epoch이 늘어날 수록 validation loss가 늘어남을 알 수 있다!

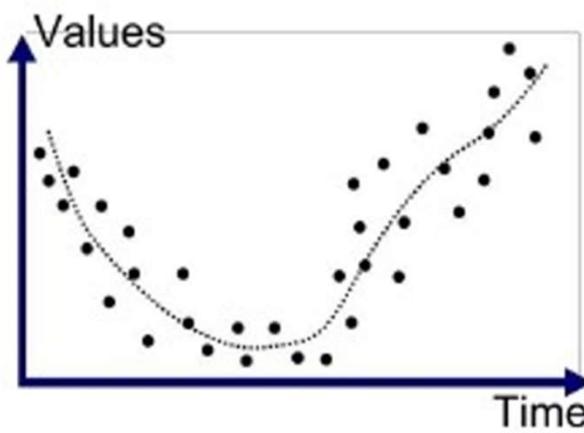


경희대학교
KYUNG HEE UNIVERSITY

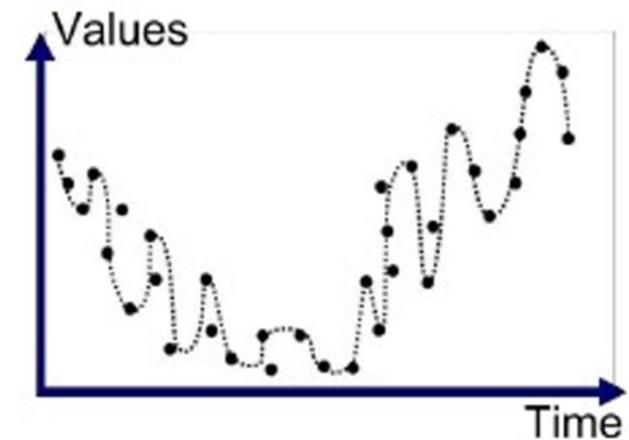
Under fitting, Over fitting



Underfitted



Good Fit/Robust



Overfitted

Under fitting

- 학습 데이터를 충분히 학습하지 않아 성능이 매우 안 좋은 경우
- 모델이 지나치게 단순한 경우
- 해결 방안
 - 충분한 학습 데이터 수집
 - 보다 더 복잡한 모델 사용
 - 에폭수(epochs)를 늘려 충분히 학습

Over fitting

- 모델이 학습 데이터에 지나치게 맞추어진 상태
- 새로운 데이터에서는 성능 저하
- 데이터에는 잡음이나 오류가 포함
- 학습 데이터가 매우 적을 경우
- 모델이 지나치게 복잡한 경우
- 학습 횟수가 매우 많을 경우
- 해결방안
 - 다양한 학습 데이터 수집 및 학습
 - 모델 단순화: 파라미터가 적은 모델을 선택하거나, 학습 데이터의 특성 수를 줄임
 - 정규화(Regularization)을 통한 규칙 단순화
 - 적정한 하이퍼 파라미터 찾기

Prevent Overfitting and Underfitting

- 모델의 크기 축소
- 가중치 초기화(Weight Initializer)
- 옵티마이저(Optimizer)
- 배치 정규화(Batch Normalization)
- 규제화(Regularization)
- 드롭아웃(Dropout)

Adjust Model Size

- 가장 단순한 방법
- 모델의 크기를 줄인다는 것은 학습 파라미터(trainable parameter)의 수를 줄이는 것

Reduce model size

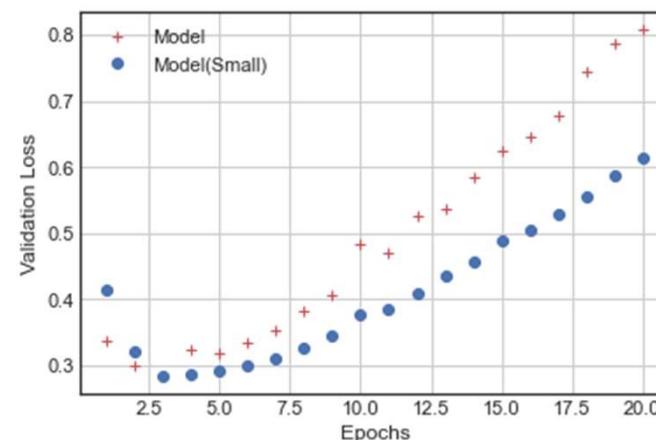
```
In [84]: model_s = Sequential([Dense(7, activation = 'relu', input_shape = (10000, ), name = 'input'),
                           Dense(7, activation = 'relu', name = 'hidden2'),
                           Dense(1, activation = 'sigmoid', name = 'output2')])

model_s.compile(optimizer = 'rmsprop',
                 loss = 'binary_crossentropy',
                 metrics = ['acc'])

model_s.summary()
```

Model: "sequential_4"

Layer (type)	Output Shape	Param #
<hr/>		
input (Dense)	(None, 7)	70007
hidden2 (Dense)	(None, 7)	56
output2 (Dense)	(None, 1)	8
<hr/>		
Total params:	70,071	
Trainable params:	70,071	
Non-trainable params:	0	
<hr/>		



Increase model size

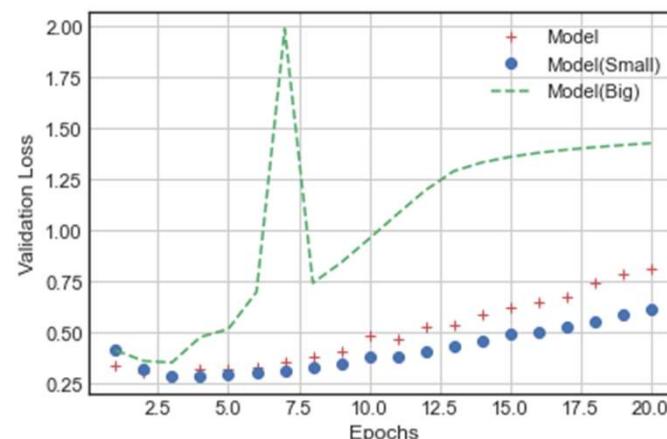
```
In [87]: model_b = Sequential([Dense(1024, activation = 'relu', input_shape = (10000, ), name = 'input'),
                           Dense(1024, activation = 'relu', name = 'hidden3'),
                           Dense(1, activation = 'sigmoid', name = 'output3')])

model_b.compile(optimizer = 'rmsprop',
                 loss = 'binary_crossentropy',
                 metrics = ['acc'])

model_b.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
<hr/>		
input (Dense)	(None, 1024)	10241024
hidden3 (Dense)	(None, 1024)	1049600
output3 (Dense)	(None, 1)	1025
<hr/>		
Total params:	11,291,649	
Trainable params:	11,291,649	
Non-trainable params:	0	



Regularization

- 과대적합(Overfitting)을 방지하는 방법 중 하나
- 과대적합은 가중치의 매개변수 값이 커서 발생하는 경우가 많음
이를 방지하기 위해 **큰 가중치 값에 큰 규제를 가하는 것**
- 규제란 가중치의 절댓값을 가능한 작게 만드는 것으로, 가중치의 모든 원소를 0에 가깝게 하여 모든 특성이 출력에 주는 영향을 최소한으로 만드는 것(기울기를 작게 만드는 것)을 의미
- 가중치의 분포가 더 균일하게 됨
- 복잡한 네트워크 일수록 네트워크의 복잡도에 제한을 두어 가중치가 작은 값을 가지도록 함
- 규제란 과대적합이 되지 않도록 모델을 강제로 제한한다는 의미
- 적절한 규제값을 찾는 것이 중요
- 네트워크 손실함수에 큰 가중치와 연관된 비용을 추가
 - L1 규제: 가중치의 절댓값에 비례하는 비용이 추가
 - L2 규제: 가중치의 제곱에 비례하는 비용이 추가(흔히 가중치 감쇠라고도 불림)

L1 Regularization

- 가중치의 절대값합
- L2 규제와 달리 어떤 가중치는 0이 되는데 이는 모델이 가벼워짐을 의미
- $Loss = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{y}_i) + \frac{\lambda}{2} |w|$

L1 Regularization

```
In [123]: l1_model = Sequential([Dense(16, kernel_regularizer=l1(0.0001), activation = 'relu', input_shape = (10000, )),
                           Dense(16, kernel_regularizer = l1(0.0001), activation = 'relu'),
                           Dense(1, activation = 'sigmoid')])

l1_model.compile(optimizer = 'rmsprop',
                  loss = 'binary_crossentropy',
                  metrics=['acc'])

l1_model.summary()

plot_model(l1_model, show_shapes = True)
```

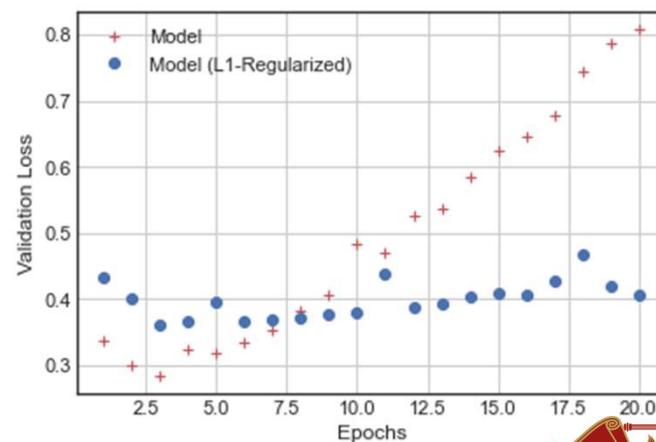
Model: "sequential_9"

Layer (type)	Output Shape	Param #
<hr/>		
dense_25 (Dense)	(None, 16)	160016
dense_26 (Dense)	(None, 16)	272
dense_27 (Dense)	(None, 1)	17
<hr/>		

Total params: 160,305

Trainable params: 160,305

Non-trainable params: 0



L2 Regularization

- 가중치의 제곱합
- 손실 함수 일정 값을 더함으로써 과적합을 방지
- λ 값이 크면 가중치 감소가 커지고, 작으면 가하는 규제가 적어진다.
- 더 Robust한 모델을 생성하므로 L1보다 많이 사용됨
- $Loss = \frac{1}{n} \sum_{i=1}^n L(y_i, \hat{y}_i) + \frac{\lambda}{2} w^2$

L2 Regularization

```
In [120]: from tensorflow.keras.regularizers import l1, l2, l1_l2

l2_model = Sequential([Dense(16, kernel_regularizer=l2(0.001), activation = 'relu', input_shape = (10000, )),
                      Dense(16, kernel_regularizer = l2(0.001), activation = 'relu'),
                      Dense(1, activation = 'sigmoid')])

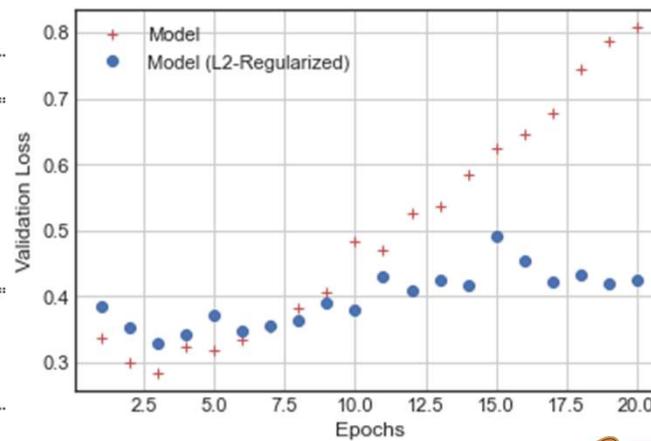
l2_model.compile(optimizer = 'rmsprop',
                  loss = 'binary_crossentropy',
                  metrics=['acc'])

l2_model.summary()

plot_model(l2_model, show_shapes = True)
```

Model: "sequential_8"

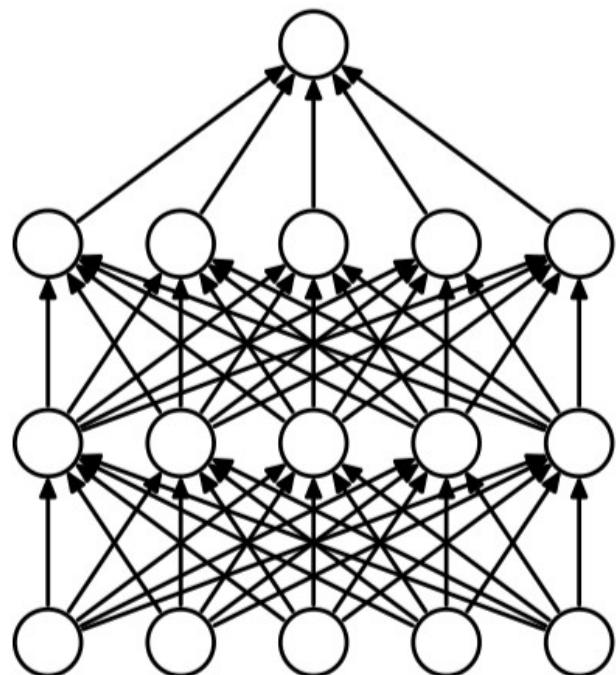
Layer (type)	Output Shape	Param #
dense_22 (Dense)	(None, 16)	160016
dense_23 (Dense)	(None, 16)	272
dense_24 (Dense)	(None, 1)	17
<hr/>		
Total params:	160,305	
Trainable params:	160,305	
Non-trainable params:	0	



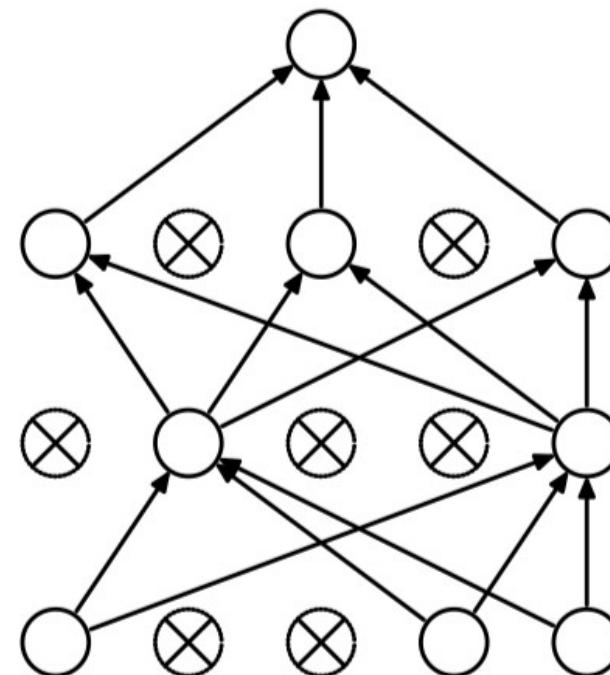
Dropout

- 신경망을 위해 사용되는 규제 기법 중 가장 효과적이고 널리 사용되는 방법
- 과적합을 방지하기 위한 방법
- 학습할 때 사용하는 노드의 수를 전체 노드 중에서 일부만을 사용
- 신경망의 레이어에 드롭아웃을 적용하면 훈련하는 동안 무작위로 층의 일부 특성(노드)를 제외
 - 예를 들어, [1.0, 3.2, 0.6, 0.8, 1.1]라는 벡터에 대해 드롭아웃을 적용하면 **무작위로 0으로 바뀜** -> [0, 3.2, 0.6, 0.8, 0]
 - 보통 0.2 ~ 0.5 사이의 비율로 지정됨
- 테스트 단계에서는 그 어떤 노드도 드롭아웃 되지 않고, 대신 해당 레이어의 출력 노드를 드롭아웃 비율에 맞게 줄여줌

Dropout



(a) Standard Neural Net



(b) After applying dropout.



경희대학교
KYUNG HEE UNIVERSITY

Dropout

```
In [130]: from tensorflow.keras.layers import Dropout

dropout_model = Sequential([Dense(16, activation = 'relu', input_shape = (10000, )),
                            Dropout(0.5),
                            Dense(16, activation = 'relu'),
                            Dropout(0.5),
                            Dense(1, activation = 'sigmoid')])

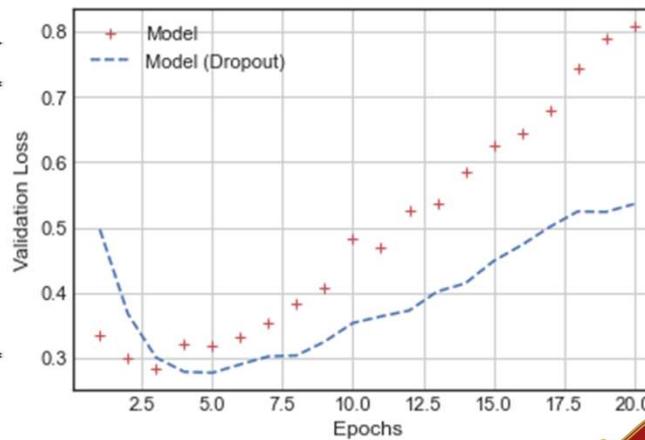
dropout_model.compile(optimizer = 'rmsprop',
                      loss = 'binary_crossentropy',
                      metrics=['acc'])

dropout_model.summary()

plot_model(dropout_model)
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
<hr/>		
dense_31 (Dense)	(None, 16)	160016
dropout (Dropout)	(None, 16)	0
dense_32 (Dense)	(None, 16)	272
dropout_1 (Dropout)	(None, 16)	0
dense_33 (Dense)	(None, 1)	17
<hr/>		
Total params:	160,305	
Trainable params:	160,305	
Non-trainable params:	0	
<hr/>		



Optimizer

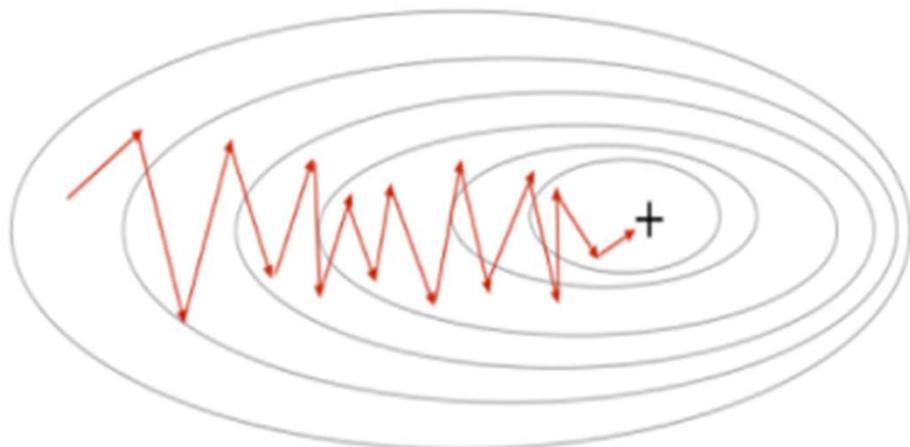
Stochastic Gradient Descent

- 전체를 한번에 계산하지 않고, **확률적으로** 일부 샘플을 뽑아 조금씩 나누어 학습을 시키는 과정
- 반복할 때마다 다루는 데이터의 수가 적기 때문에 한 번 처리하는 속도는 빠름
- 한 번 학습할 때 필요한 메모리만 있으면 되므로 매우 큰 데이터셋에 대해서도 학습이 가능
- 확률적이기 때문에, 배치 경사하강법보다 불안정
- 손실함수의 최솟값에 이를 때까지 다소 위아래로 요동치면서 이동
- 땀라서 위와 같은 문제 때문에 **미니 배치 경사하강법**(mini-batch gradient descent)로 학습을 진행
- 요즘에는 보통 SGD라고하면 미니 배치 경사하강법을 의미하기도 함
- SGD의 단점: 단순하지만 문제에 따라서 시간이 매우 오래 걸림
- SGD 수식
- $W \leftarrow W - \gamma \frac{\delta L}{\delta W}, \gamma: learning rate$

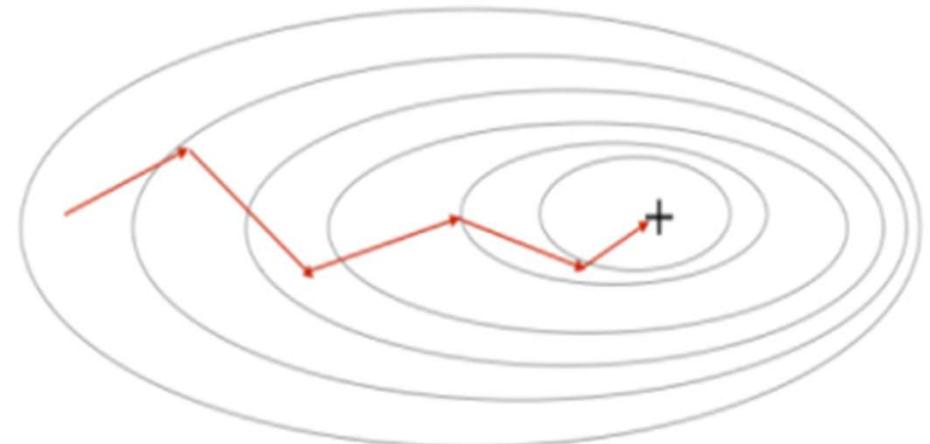
Optimizer

Stochastic Gradient Descent vs Mini-Batch Gradient Descent

Stochastic Gradient Descent



Mini-Batch Gradient Descent



경희대학교
KYUNG HEE UNIVERSITY

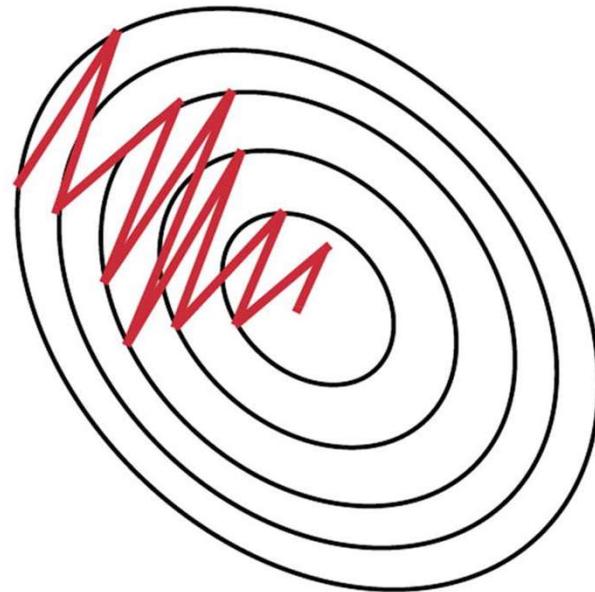
Optimizer

Momentum

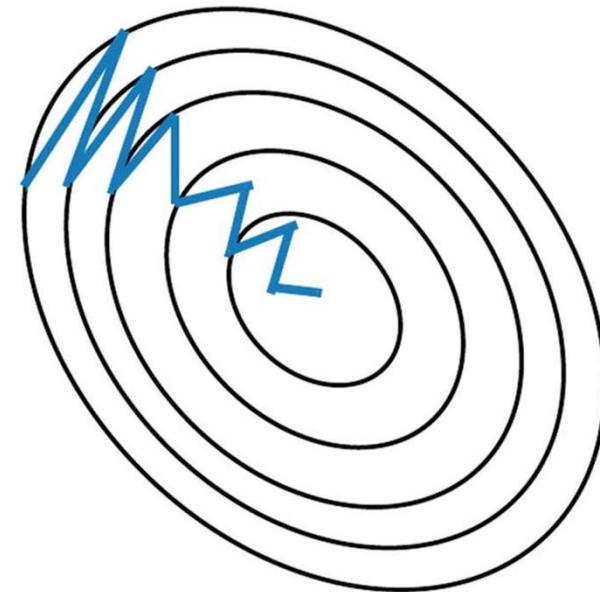
- 운동량을 의미, 관성과 관련
- 공이 그릇의 경사면을 따라서 내려가는 듯한 모습
- 이전의 속도를 유지하려는 성향
- 경사하강을 좀 더 유지하려는 성격을 지님
- 단순히 SGD만 사용하는 것보다 적게 방향이 변함
- Momentum 수식
 - $v \leftarrow \alpha v - \gamma \frac{\delta L}{\delta w}$
 - $W \leftarrow W + v$
 - $v_{init} = 0$

Optimizer

SGD without momentum vs SGD with momentum



Stochastic Gradient
Descent **without**
Momentum



Stochastic Gradient
Descent **with**
Momentum

Optimizer

Nesterov Acceleration Gradient

- 모멘텀의 방향으로 조금 앞선 곳에서 손실함수의 그라디언트를 구함
- 시간이 지날수록 조금 더 빨리 최솟값에 도달
- 일반적으로 모멘텀 벡터가 올바른 방향을 가리킬 것이라는 아이디어에서 착안
- Nesterov 수식
 - $m \leftarrow \beta m - \eta \nabla_{\theta} J(\theta + \beta m)$
 - $\theta \leftarrow \theta + m$
 - $\theta + \beta m$: 현재 위치 (가중치) + 모멘텀 * 속도 (모멘텀 벡터)
 - $\nabla_{\theta} J(\theta + \beta m)$: loss function의 미분값
 - η : learning rate

Optimizer

AdaGrad (Adaptive Gradient)

- 가장 가파른 경사를 따라 빠르게 하강하는 방법
- 학습률을 변화시키며 진행하며 적응적 학습률이라고도 부름
- 경사가 급할 때는 빠르게 변화, 완만할 때는 느리게 변화
- 간단한 문제에서는 좋을 수는 있지만 딥러닝(Deep Learning)에서는 자주 쓰이지 않음
- 학습률이 너무 감소되어 전역 최소값(global minimum)에 도달하기 전에 학습이 빨리 종료될 수 있기 때문
- AdaGrad 수식
 - $s \leftarrow s + \nabla_{\theta}J(\theta) \otimes \nabla_{\theta}J(\theta)$
 - $\theta \leftarrow \theta - \eta \nabla_{\theta}J(\theta) \oslash \sqrt{s + \varepsilon}$

Optimizer

RMSProp (Root Mean Square Propagation)

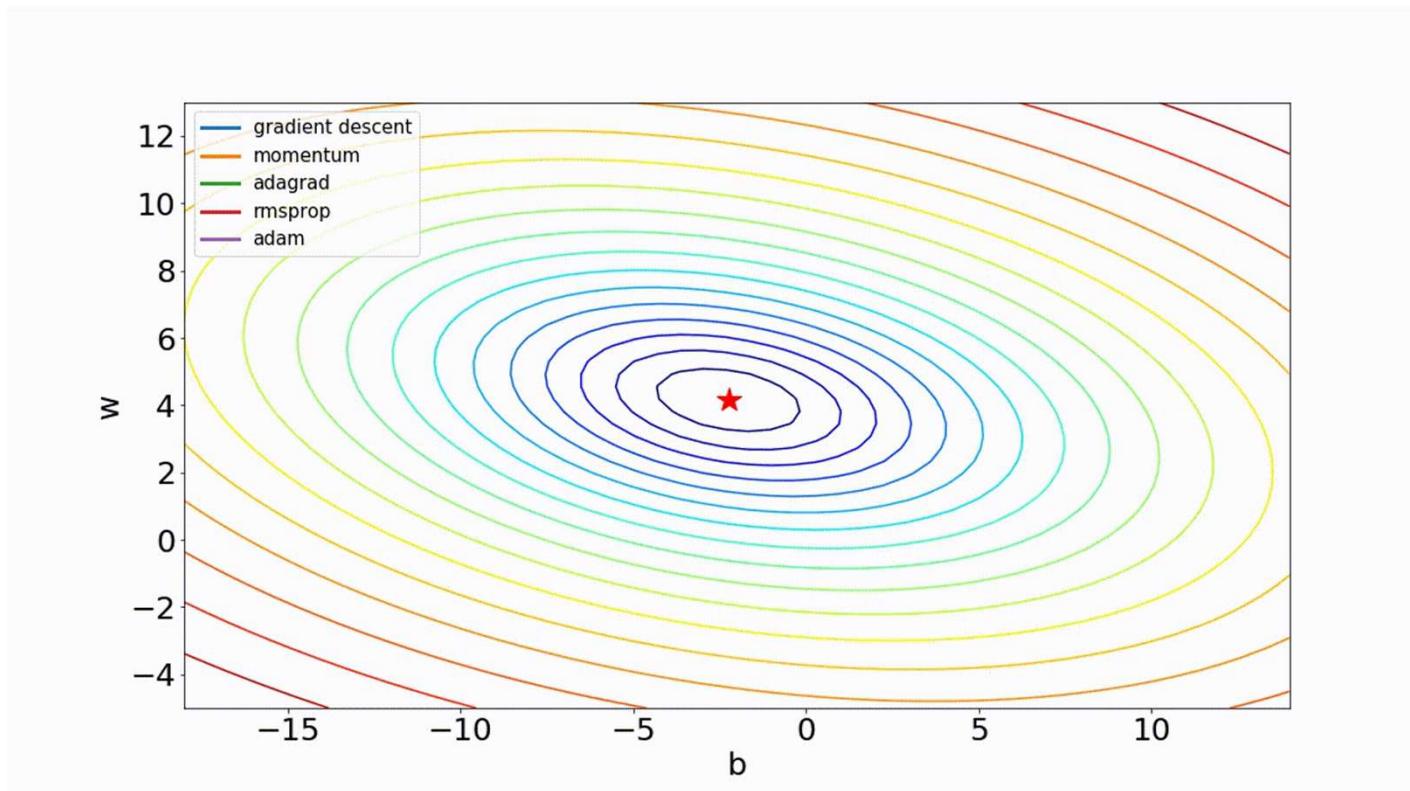
- AdaGrad를 보완하기 위한 방법으로 등장
- 합 대신 지수의 평균값을 활용
- 학습이 안되기 시작하면 학습률이 커져서 잘 되게하고, 학습률이 너무 크면 학습률을 다시 줄임
- RMSProp 수식
 - $s \leftarrow \beta s + (1 - \beta) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$
 - $\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{s + \epsilon}$

Optimizer

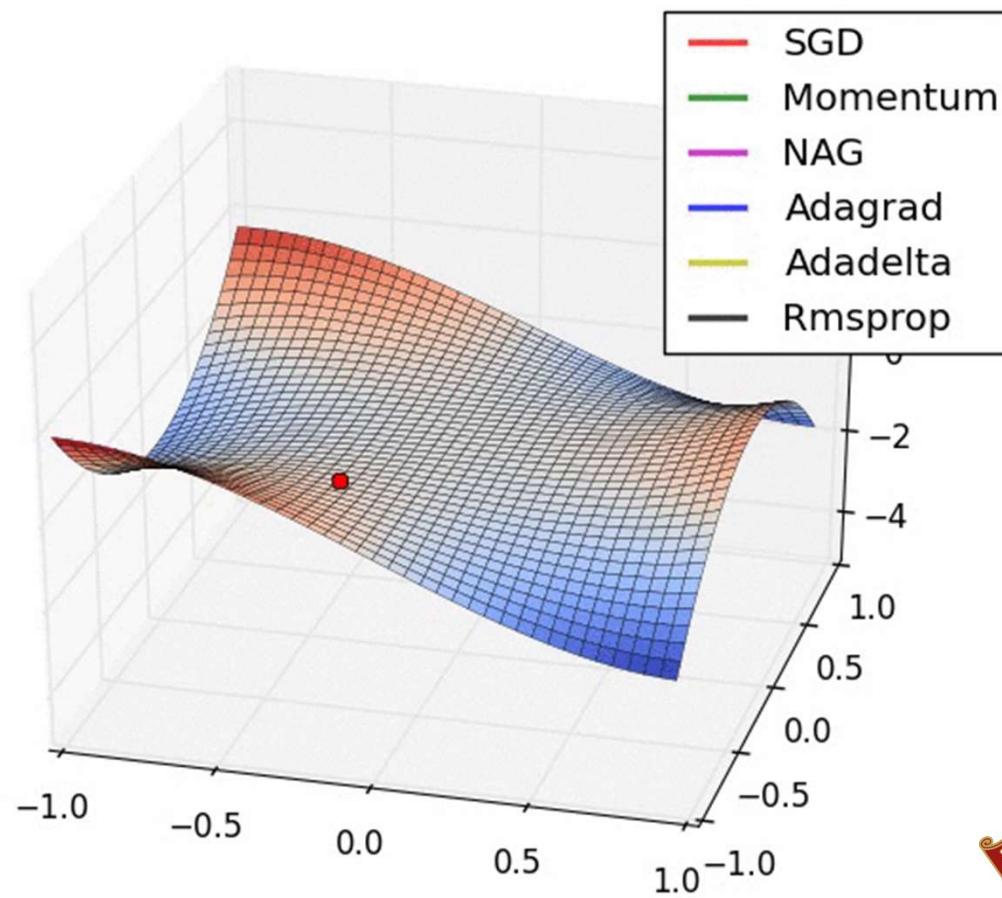
Adam (Adaptive Momentum Estimation)

- 모멘텀 최적화와 RMSProp의 아이디어를 합친 것
- 지난 그래디언트의 지수 감소 평균을 따르고(Momentum), 지난 그래디언트 제곱의 지수 감소된 평균(RMSProp)을 따름
- 가장 많이 사용되는 최적화 방법
- Adam 수식
 - $m \leftarrow \beta_1 m - (1 - \beta_1) \nabla_{\theta} J(\theta)$
 - $s \leftarrow \beta_2 s + (1 - \beta_2) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$
 - $\hat{m} \leftarrow \frac{m}{1 - \beta_1^t}$
 - $\hat{s} \leftarrow \frac{s}{1 - \beta_2^t}$
 - $\theta \leftarrow \theta + \eta \hat{m} \oslash \sqrt{\hat{s} + \epsilon}$

Optimizer Comparison



Optimizer Comparison



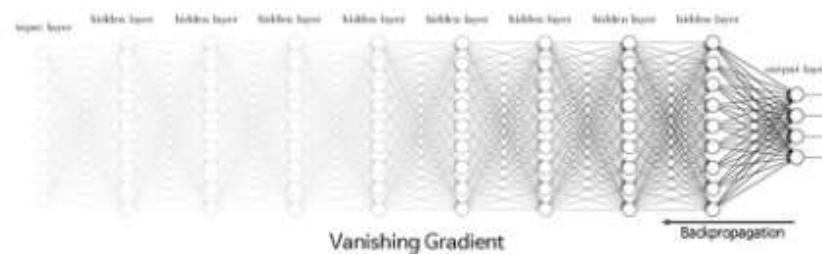
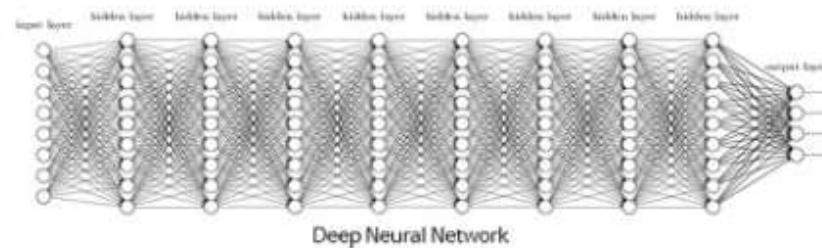
경희대학교
KYUNG HEE UNIVERSITY

Weights Initialization

- 활성화함수가 Sigmoid 함수 일 때, 은닉층의 개수가 늘어 날수록 가중치가 역전파되면서 가중치 소실문제 발생
 - 0 ~ 1 사이의 값으로 출력되면서 0 또는 1에 가중치 값이 퍼짐
이는 미분값이 점점 0에 가까워짐을 의미하기도 함
 - ReLU 함수 등장(비선형 함수)
- 가중치 초기화 문제(은닉층의 활성화값 분포)
 - 가중치의 값이 일부 값으로 치우치게 되면, 활성화 함수를 통과한 값이 치우치게 되고, 표현할 수 있는 신경망의 수가 적어짐
 - 따라서, 활성화값이 골고루 분포되는 것이 중요

Weights Initialization

Sigmoid: Vanishing Gradient Problem

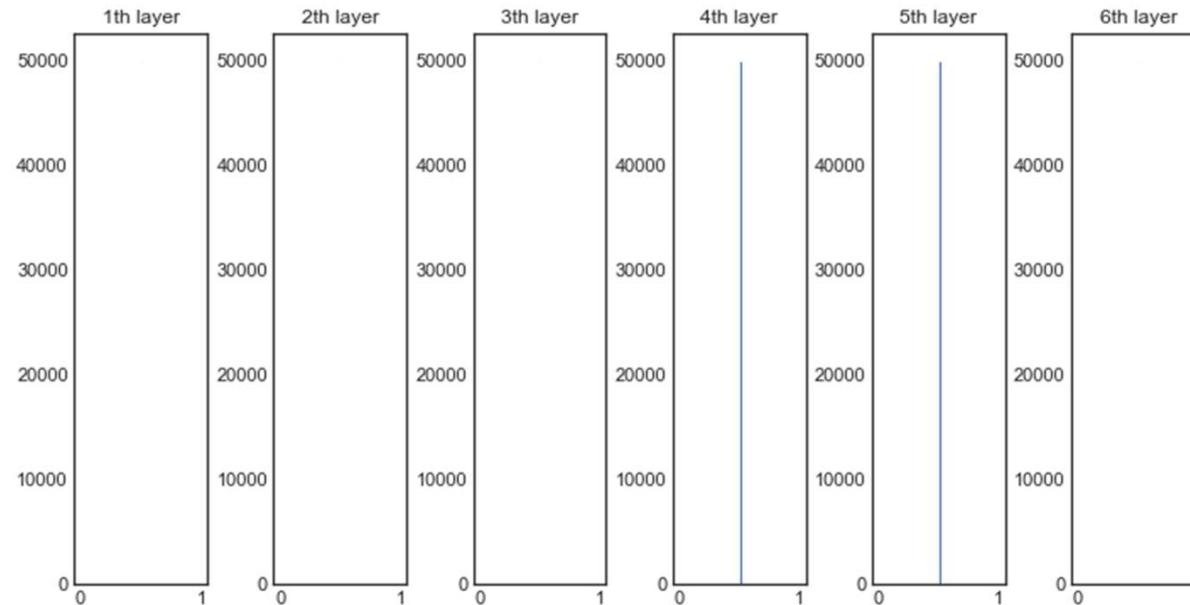


경희대학교
KYUNG HEE UNIVERSITY

선형 함수에서 가중치 초기화

초기값: 0 (zeros)

- 학습이 올바르게 진행되지 않음
- 0으로 설정하면 오차역전파법에서 모든 가중치의 값이 똑같이 갱신됨

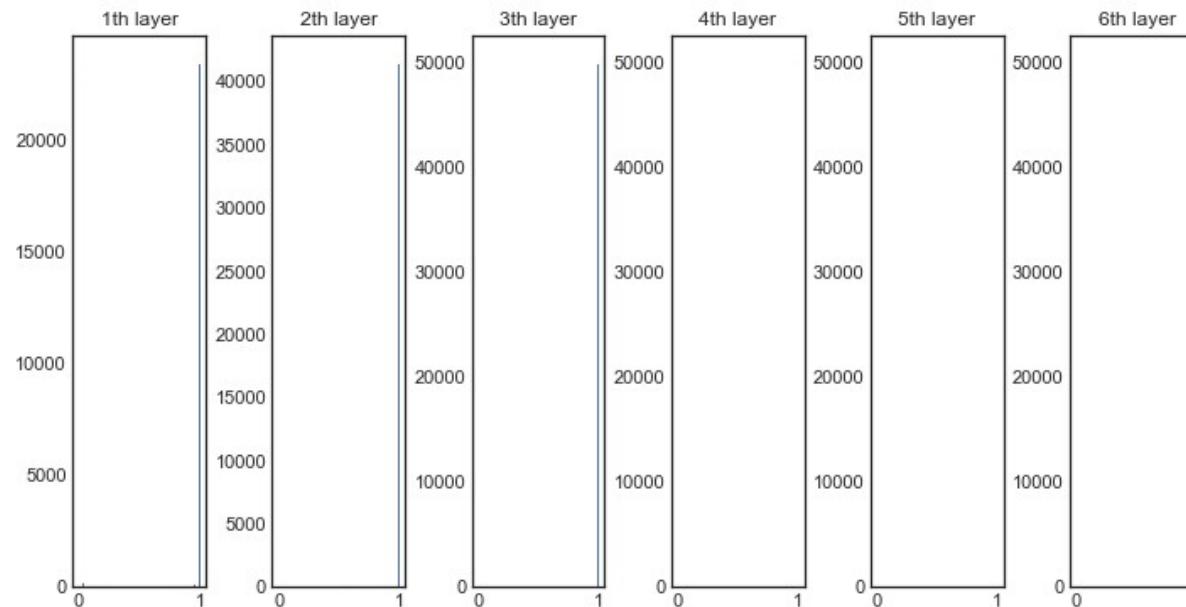


경희대학교
KYUNG HEE UNIVERSITY

선형 함수에서 가중치 초기화

초기값: 균일분포(Uniform)

- 활성화 값이 균일하지 않음(활성화함수 : sigmoid)
- 역전파로 전해지는 기울기 값이 사라짐

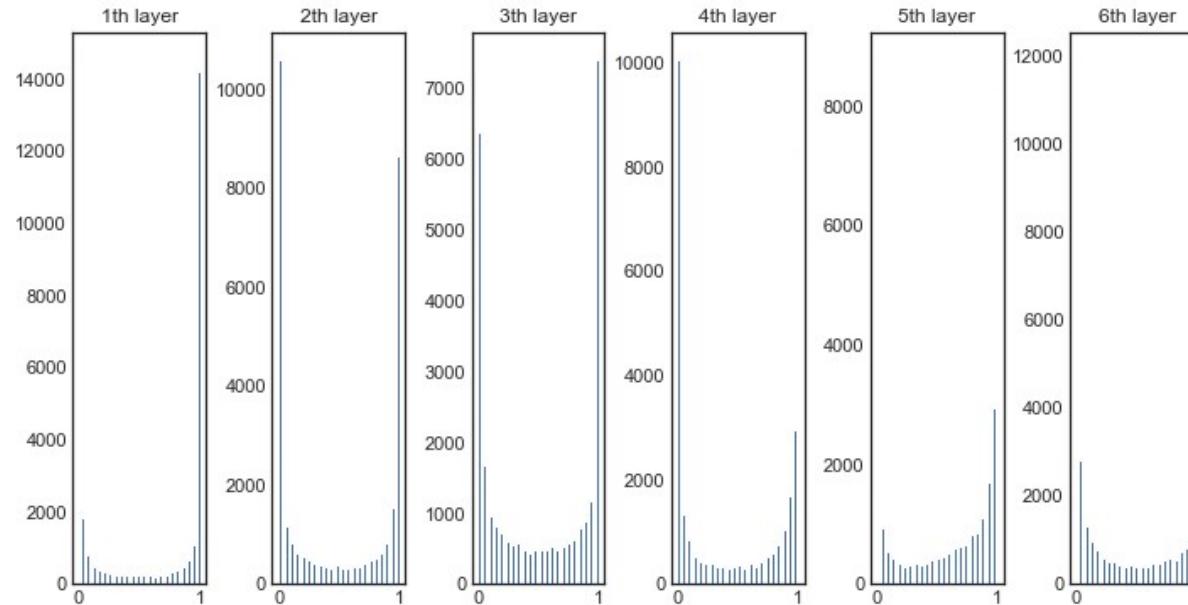


경희대학교
KYUNG HEE UNIVERSITY

선형 함수에서 가중치 초기화

초기값: 정규분포(normalization)

- 활성화함수를 통과하면 양쪽으로 퍼짐
- 0과 1에 퍼지면서 기울기 소실문제(gradient vanishing) 발생

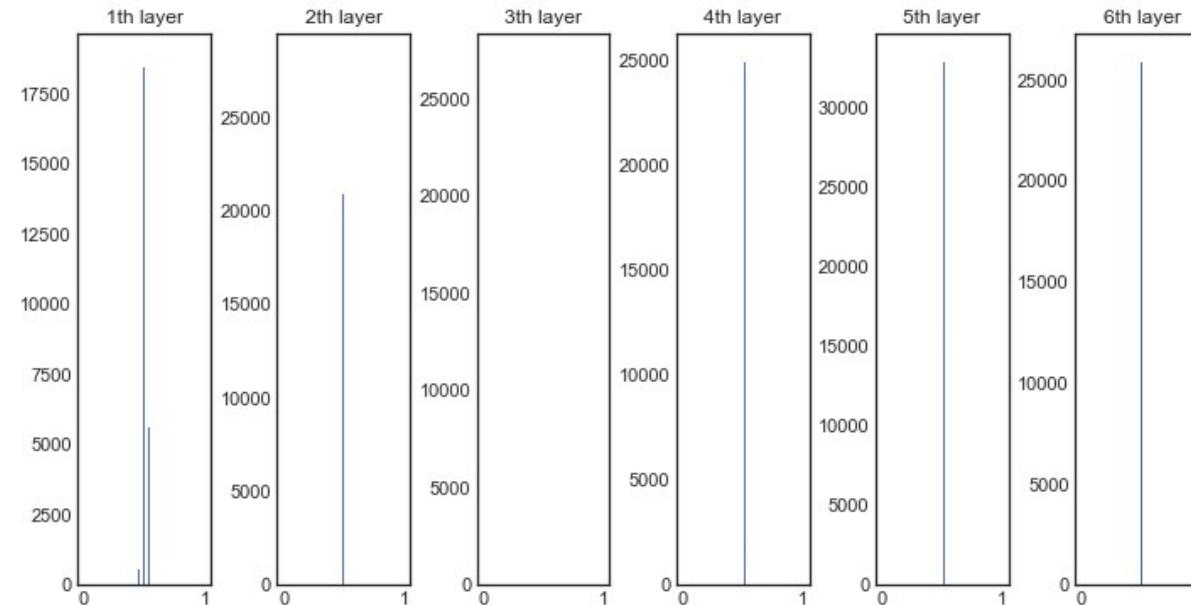


경희대학교
KYUNG HEE UNIVERSITY

선형 함수에서 가중치 초기화

아주 작은 정규분포 값으로 가중치 초기화

- 0과 1로 퍼지지는 않았고, 한 곳에 치우쳐 짐
- 해당 신경망이 표현할 수 있는 문제가 제한됨

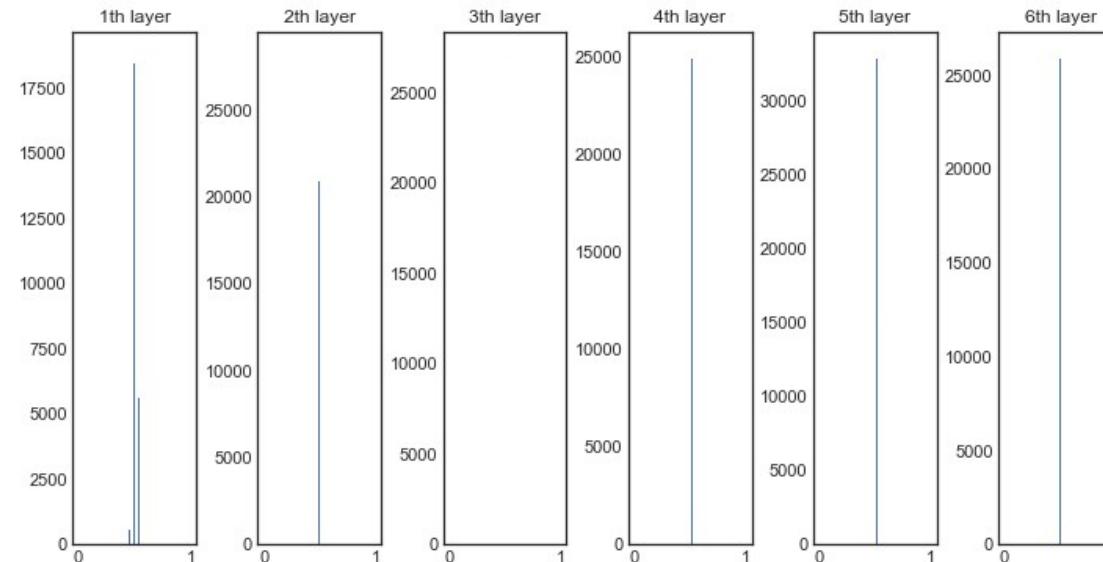


경희대학교
KYUNG HEE UNIVERSITY

선형 함수에서 가중치 초기화

초기값: Xavier (Glorot)

- 은닉층의 노드의 수가 n 이라면 표준편차가 $\frac{1}{\sqrt{n}}$ 인 분포
- 더 많은 가중치에 역전파가 전달 가능하고, 비교적 많은 문제를 표현할 수 있음.

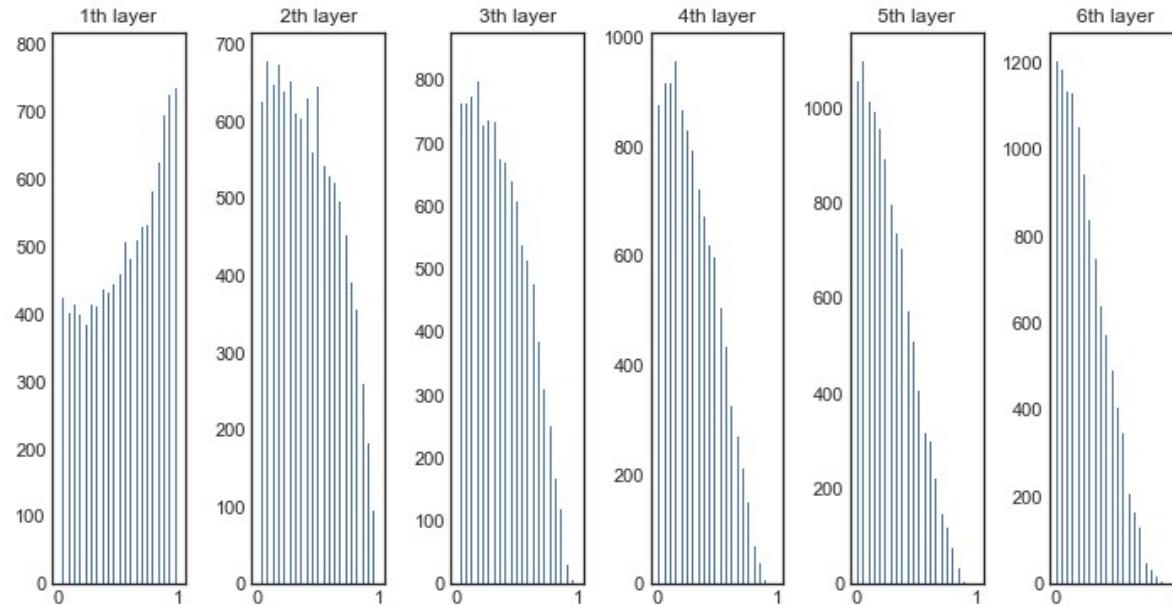


경희대학교
KYUNG HEE UNIVERSITY

선형 함수에서 가중치 초기화

초기값: Xavier (Glorot) – tanh

- 활성화 함수: tanh
- sigmoid 함수보다 더 깔끔한 종모양으로 분포

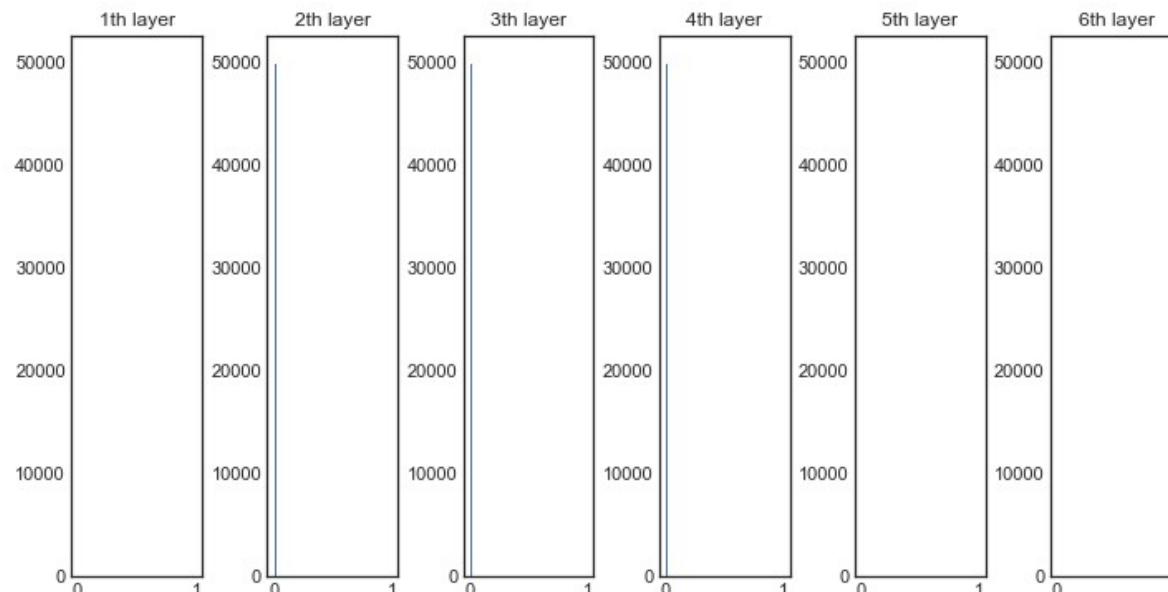


경희대학교
KYUNG HEE UNIVERSITY

비선형 함수에서 가중치 초기화

초기값: 0 (Zeros)

- 활성화함수: ReLU

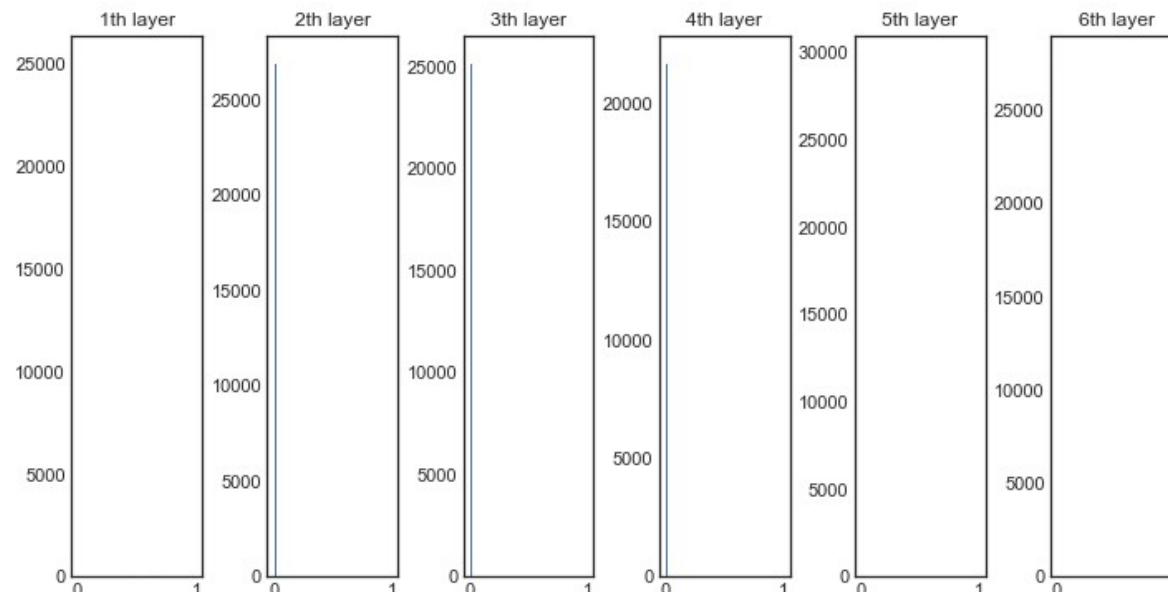


경희대학교
KYUNG HEE UNIVERSITY

비선형 함수에서 가중치 초기화

초기값: 정규분포 (Normalization)

- 활성화함수: ReLU

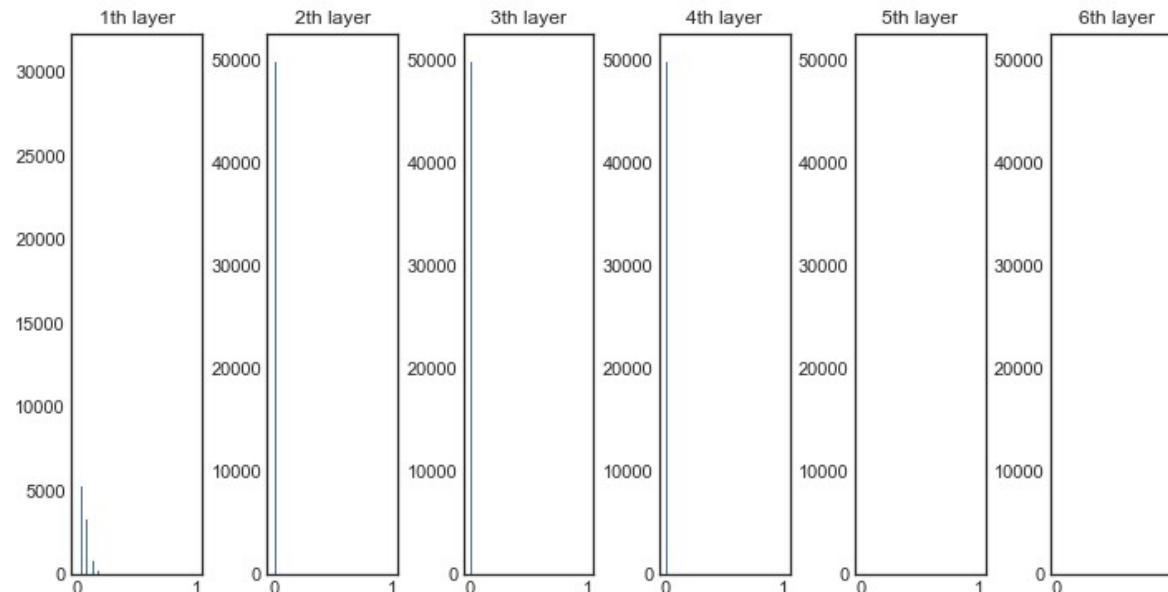


경희대학교
KYUNG HEE UNIVERSITY

비선형 함수에서 가중치 초기화

초기값: 표준편차가 0.01인 정규분포

- 활성화함수: ReLU

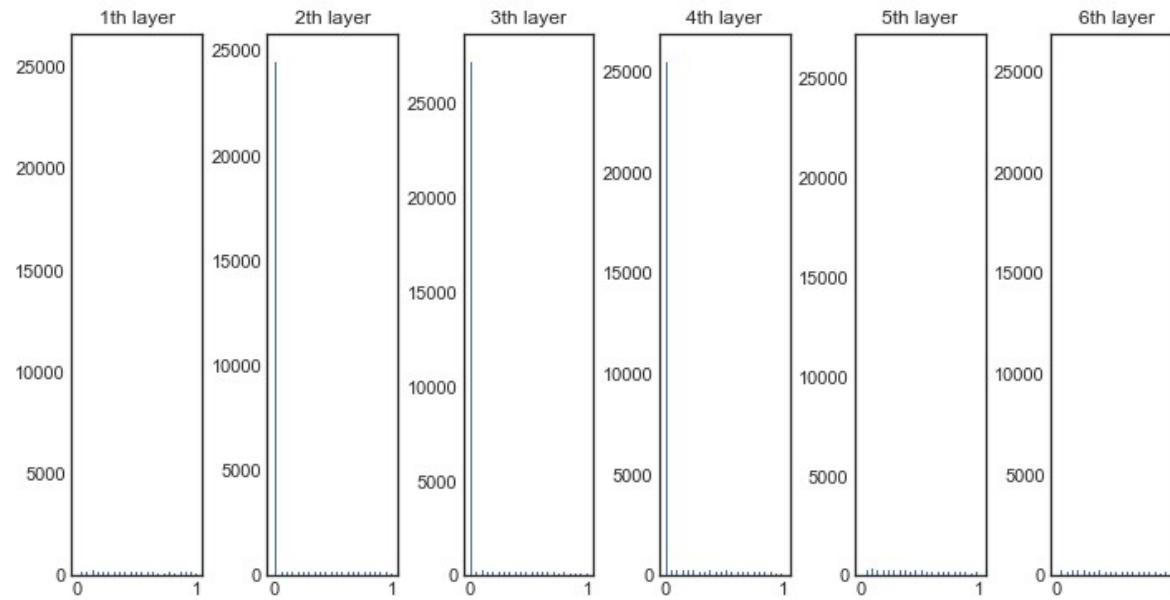


경희대학교
KYUNG HEE UNIVERSITY

비선형 함수에서 가중치 초기화

초기값: He

- 은닉층의 노드의 수가 n 이라면 표준편차가 $\frac{2}{\sqrt{n}}$ 인 분포
- 활성화값 분포가 균일하게 분포되어 있음



경희대학교
KYUNG HEE UNIVERSITY

초기화 전략

- Glorot Initialization (Xavier)

- 활성화 함수
 - 없음
 - tanh
 - sigmoid
 - softmax

- He Initialization

- 활성화 함수
 - ReLU
 - LeakyReLU
 - ELU 등

초기화 전략

```
In [118]: 1 from tensorflow.keras.layers import Dense, LeakyReLU, Activation  
2 from tensorflow.keras.models import Sequential  
3  
4 model = Sequential([Dense(30, kernel_initializer = 'he_normal', input_shape = [10, 10]),  
5                     LeakyReLU(alpha = 0.2),  
6                     Dense(1, kernel_initializer = 'he_normal'),  
7                     Activation('softmax')])  
8  
9 model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
=====		
dense_19 (Dense)	(None, 10, 30)	330
leaky_re_lu (LeakyReLU)	(None, 10, 30)	0
dense_20 (Dense)	(None, 10, 1)	31
activation_1 (Activation)	(None, 10, 1)	0
=====		

Total params: 361

Trainable params: 361

Non-trainable params: 0



경희대학교
KYUNG HEE UNIVERSITY

Hyper Parameter

- 사람이 직접 설정해야 하는 매개변수
- 학습이 되기 전 미리 설정되어 상수취급

Hyper Parameter

Learning Rate

- 학습률에 따라 학습정도가 달라짐
- 적절한 학습률을 찾는 것이 핵심

Hyper Parameter

Epochs

- 학습 횟수를 너무 작게, 또는 너무 크게 지정하면 과소적합 또는 과대적합 발생
- 여러 번 진행하면서 최적의 학습 횟수(epochs)값을 찾아야함

Hyper Parameter

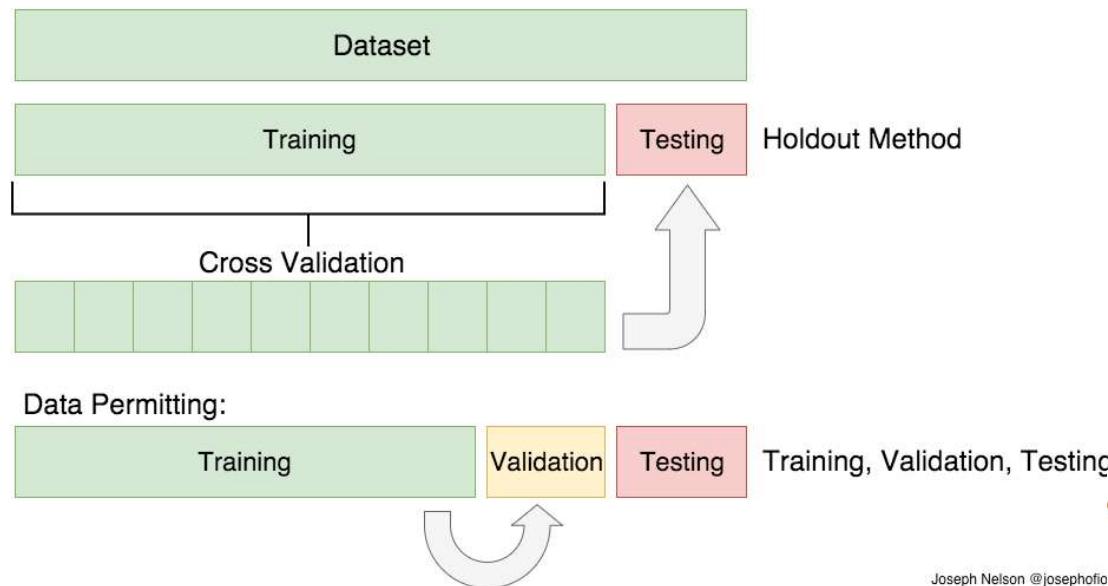
Mini Batch Size

- 미니 배치 학습
 - 한번 학습할 때 메모리의 부족현상을 막기 위해 전체 데이터의 일부를 여러 번 학습하는 방식
- 한번 학습할 때마다 얼마만큼의 미니배치 크기를 사용할지 결정
- 배치 크기가 작을수록 학습 시간이 많이 소요되고, 클수록 학습 시간이 학습 시간은 적게 소요된다.

Hyper Parameter

Validation Data

- 주어진 데이터를 학습 + 검증 + 테스트 데이터로 구분하여 과적합을 방지
- 일반적으로 전체 데이터의 2~30%를 테스트 데이터, 나머지에서 20%정도를 검증용 데이터, 남은 부분을 학습용 데이터로 사용



Joseph Nelson @josephofio



경희대학교
KYUNG HEE UNIVERSITY

Fashion MNIST Model



Fashion MNIST Model

Module Import

```
In [133]: 1 import tensorflow as tf
2 from tensorflow.keras.datasets.fashion_mnist import load_data
3 from tensorflow.keras.models import Sequential, Model
4 from tensorflow.keras import models
5 from tensorflow.keras.layers import Dense, Input
6 from tensorflow.keras.optimizers import Adam
7 from tensorflow.keras.utils import plot_model
8
9 from sklearn.model_selection import train_test_split
10
11 import numpy as np
12 import matplotlib.pyplot as plt
13 plt.style.use('seaborn-white')
```

Fashion MNIST Model

Data Load

```
In [134]: 1 tf.random.set_seed(111)
2
3 (X_train_full, y_train_full), (X_test, y_test) = load_data()
4
5 X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full,
6                                         test_size = 0.3,
7                                         random_state = 111)
```

```
In [135]: 1 print("학습 데이터: {} 레이블: {}".format(X_train_full.shape, y_train_full.shape))
2 print("학습 데이터: {} 레이블: {}".format(X_train.shape, y_train.shape))
3 print("검증 데이터: {} 레이블: {}".format(X_val.shape, y_val.shape))
4 print("테스트 데이터: {} 레이블: {}".format(X_test.shape, y_test.shape))
```

```
학습 데이터: (60000, 28, 28)    레이블: (60000,)
학습 데이터: (42000, 28, 28)    레이블: (42000,)
검증 데이터: (18000, 28, 28)    레이블: (18000,)
테스트 데이터: (10000, 28, 28)  레이블: (10000,)
```

Fashion MNIST Model

Data Preprocessing

```
In [140]: 1 X_train = (X_train.reshape(-1, 28 * 28)) / 255.  
2 X_val = (X_val.reshape(-1, 28 * 28)) / 255.  
3 X_test = (X_test.reshape(-1, 28 * 28)) / 255.
```

Fashion MNIST Model

Construct Model (functional API)

```
In [141]: 1 input_ = Input(shape = (784, ), name = 'input_')
2 hidden1 = Dense(512, activation = 'relu', name = 'hidden1')(input_)
3 hidden2 = Dense(256, activation = 'relu', name = 'hidden2')(hidden1)
4 hidden3 = Dense(128, activation = 'relu', name = 'hidden3')(hidden2)
5 hidden4 = Dense(64, activation = 'relu', name = 'hidden4')(hidden3)
6 hidden5 = Dense(32, activation = 'relu', name = 'hidden5')(hidden4)
7 output = Dense(10, activation = 'softmax', name = 'output')(hidden5)
8 model = Model(inputs = [input_], outputs = output)
9 model.summary()
```

Model: "model_6"

Layer (type)	Output Shape	Param #
<hr/>		
input_ (InputLayer)	[(None, 784)]	0
hidden1 (Dense)	(None, 512)	401920
hidden2 (Dense)	(None, 256)	131328
hidden3 (Dense)	(None, 128)	32896
hidden4 (Dense)	(None, 64)	8256
hidden5 (Dense)	(None, 32)	2080
output (Dense)	(None, 10)	330
<hr/>		

Total params: 576,810
Trainable params: 576,810
Non-trainable params: 0



경희대학교
KYUNG HEE UNIVERSITY

Fashion MNIST Model

model compile and training

```
In [143]: 1 model.compile(loss = 'sparse_categorical_crossentropy',
2                     optimizer = Adam(learning_rate = 0.01),
3                     metrics = ['acc'])
```

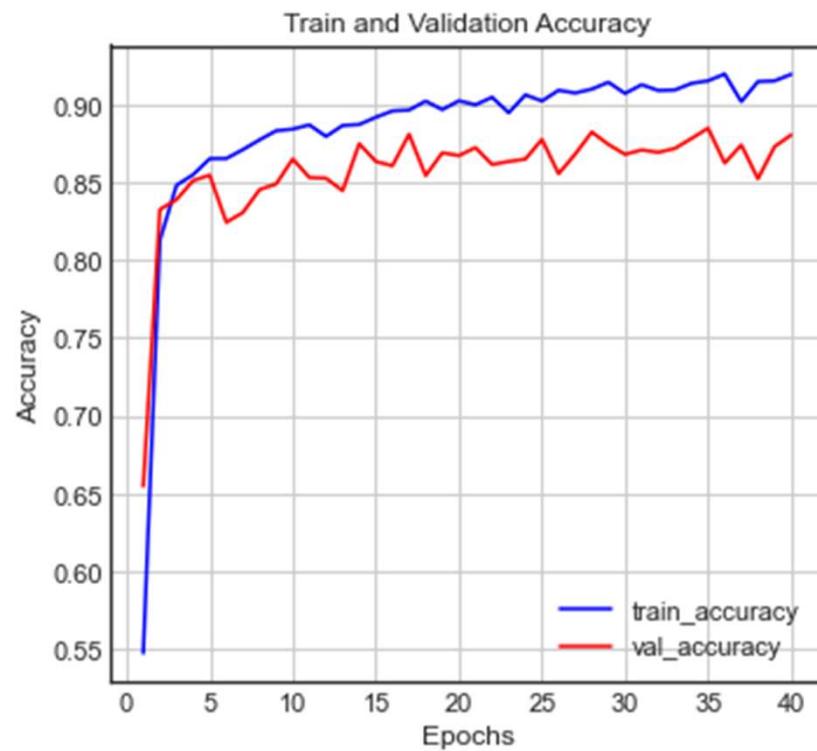
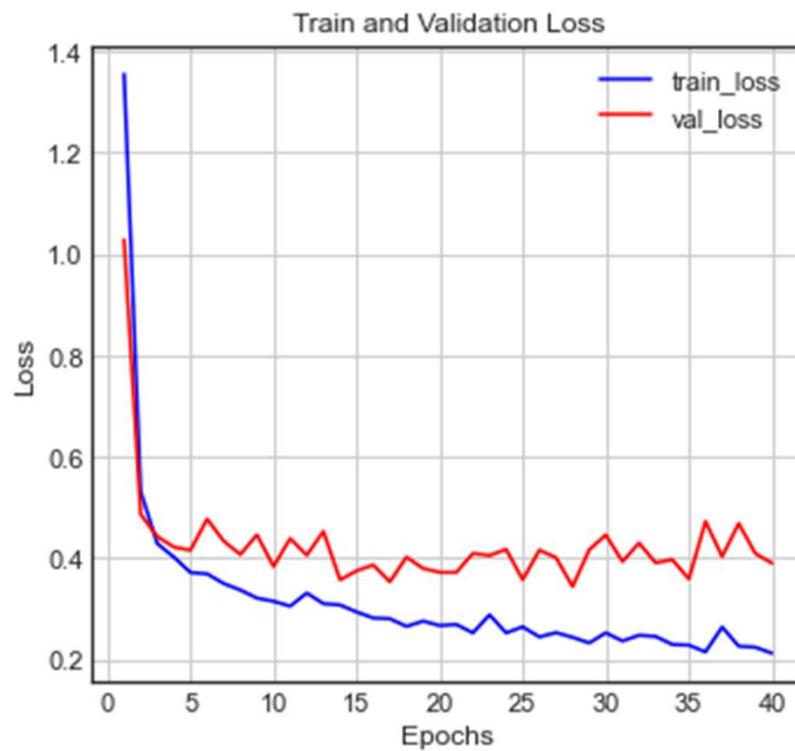
```
In [144]: 1 history = model.fit(X_train, y_train,
2                           epochs = 40,
3                           batch_size = 512,
4                           validation_data = (X_val, y_val))
```

```
Epoch 1/40
83/83 [=====] - 2s 17ms/step - loss: 1.3554 - acc: 0.5476 - val_loss: 1.0288 - val_acc: 0.6547
Epoch 2/40
83/83 [=====] - 1s 16ms/step - loss: 0.5318 - acc: 0.8128 - val_loss: 0.4867 - val_acc: 0.8326
Epoch 3/40
83/83 [=====] - 1s 16ms/step - loss: 0.4290 - acc: 0.8481 - val_loss: 0.4428 - val_acc: 0.8388
Epoch 4/40
83/83 [=====] - 1s 15ms/step - loss: 0.4022 - acc: 0.8549 - val_loss: 0.4221 - val_acc: 0.8511
Epoch 5/40
83/83 [=====] - 1s 16ms/step - loss: 0.3718 - acc: 0.8651 - val_loss: 0.4161 - val_acc: 0.8546
Epoch 6/40
83/83 [=====] - 1s 15ms/step - loss: 0.3693 - acc: 0.8653 - val_loss: 0.4774 - val_acc: 0.8243
Epoch 7/40
83/83 [=====] - 1s 15ms/step - loss: 0.3502 - acc: 0.8710 - val_loss: 0.4339 - val_acc: 0.8306
Epoch 8/40
83/83 [=====] - 1s 15ms/step - loss: 0.3374 - acc: 0.8772 - val_loss: 0.4075 - val_acc: 0.8453
Epoch 9/40
83/83 [=====] - 1s 15ms/step - loss: 0.3214 - acc: 0.8831 - val_loss: 0.4463 - val_acc: 0.8489
Epoch 10/40
83/83 [=====] - 1s 15ms/step - loss: 0.3159 - acc: 0.8941 - val_loss: 0.4020 - val_acc: 0.8550
```



Fashion MNIST Model

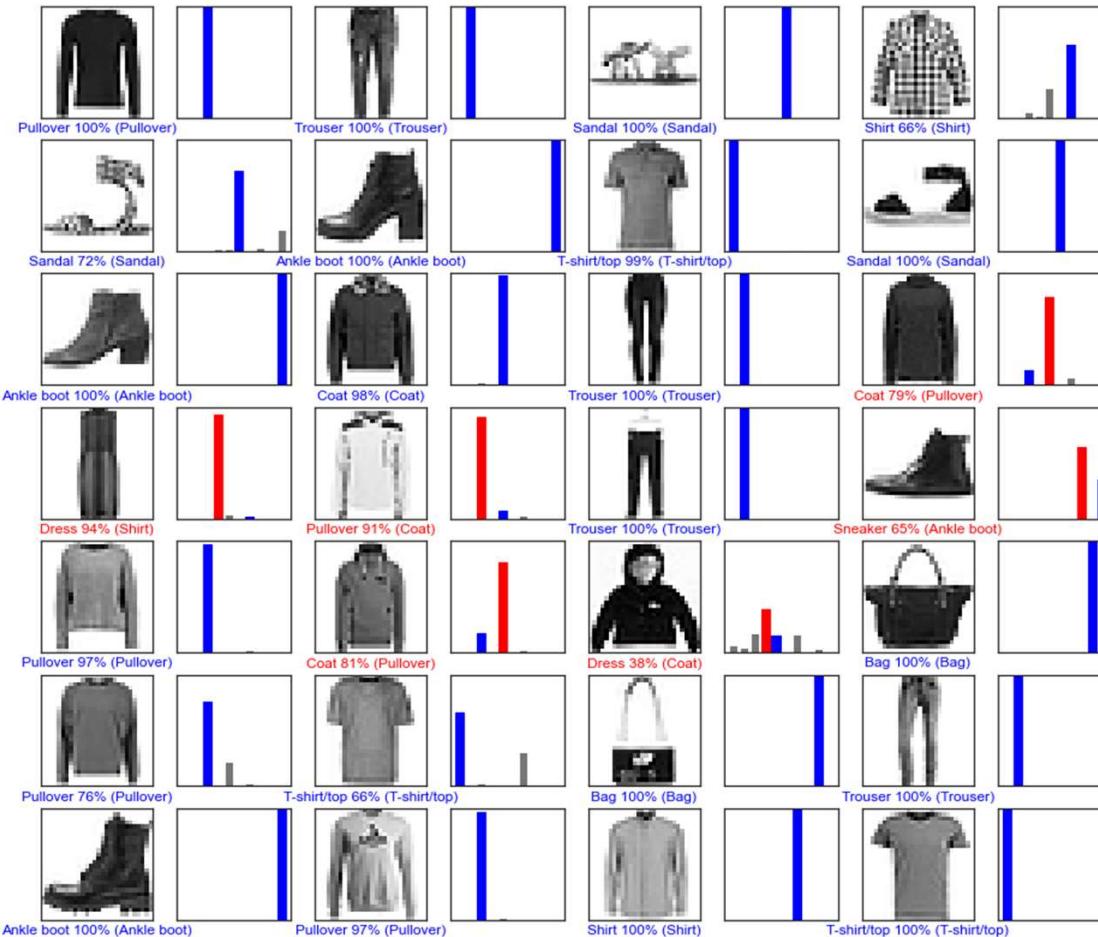
model compile and training



경희대학교
KYUNG HEE UNIVERSITY

Fashion MNIST Model

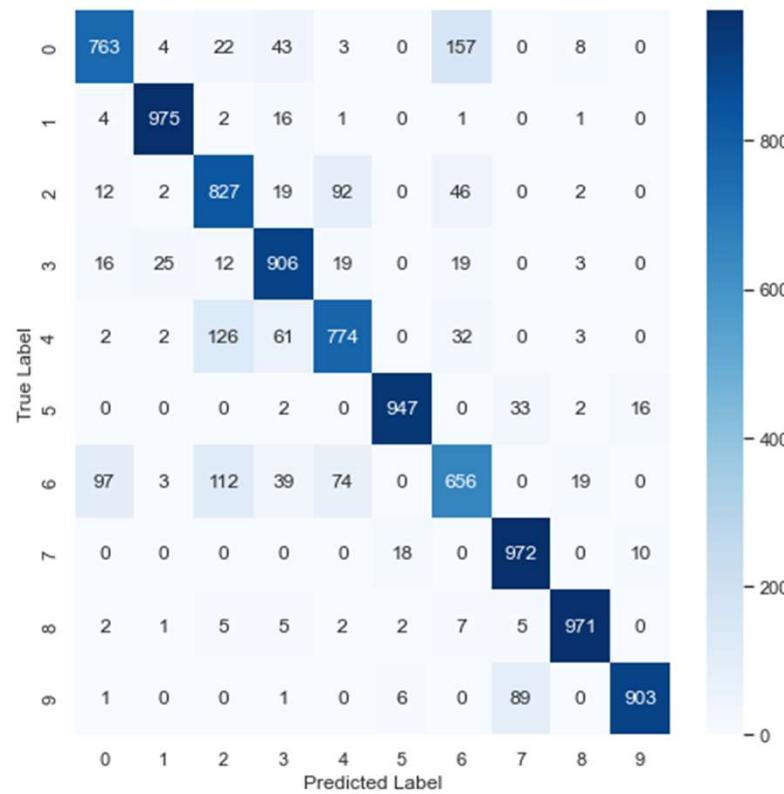
model evaluate



경희대학교
KYUNG HEE UNIVERSITY

Fashion MNIST Model

model evaluate, Confusion Matrix



Fashion MNIST Model

model evaluate, Classification Report

	precision	recall	f1-score	support
0	0.85	0.76	0.80	1000
1	0.96	0.97	0.97	1000
2	0.75	0.83	0.79	1000
3	0.83	0.91	0.87	1000
4	0.80	0.77	0.79	1000
5	0.97	0.95	0.96	1000
6	0.71	0.66	0.68	1000
7	0.88	0.97	0.93	1000
8	0.96	0.97	0.97	1000
9	0.97	0.90	0.94	1000
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000