

KT AIVLE School

# 1일차 정리

# 전체 Process(CRISP-DM)

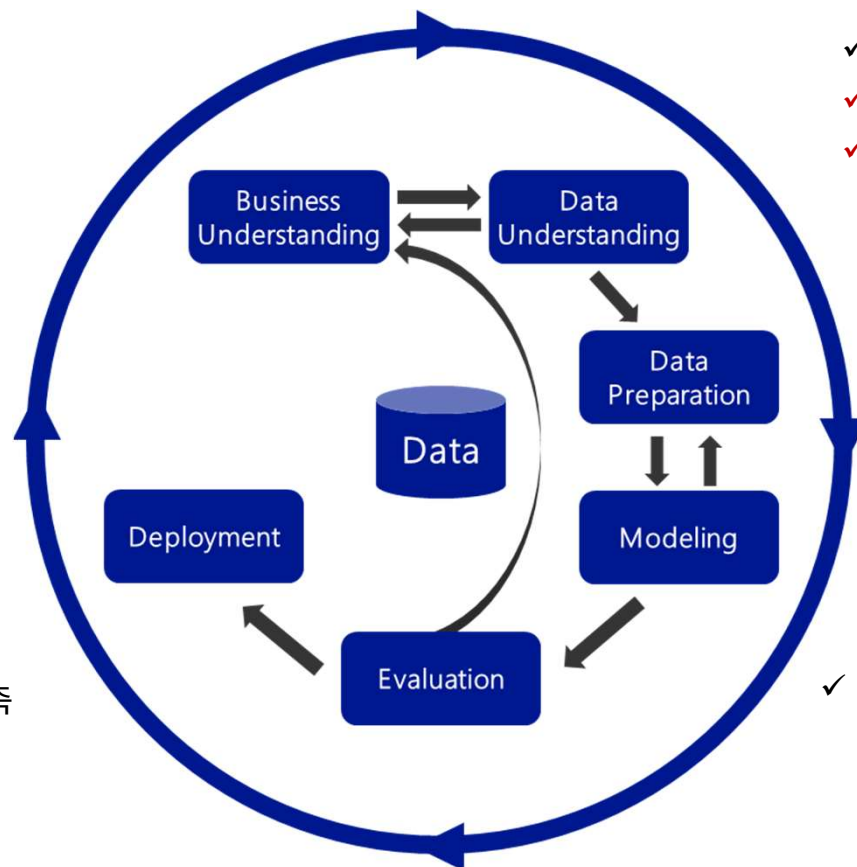
## 무엇이 문제인가?

- ✓ 비즈니스 문제정의
  - ✓ 데이터분석 방향, 목표
  - ✓ 초기 가설 수립
- $x \rightarrow y$

## 어디? 진짜?

- ✓ 원본식별
- ✓ 분석을 위한 구조 만들기
- ✓ 데이터분석 EDA & CDA

- ✓ 모델 관리
- ✓ AI 서비스 구축



- ✓ 모델링을 위한 데이터 구조 만들기
  - 모든 셀은 값이 있어야 한다.
  - 모든 값은 숫자여야 한다.
  - (필요시) 숫자의 범위가 일치

- ✓ 모델을 만들고 검증하기

## 문제가 해결 되었는가?

- ✓ 기술적 관점 평가
- ✓ 비즈니스 관점 평가

# 모델링을 위해 필요한 것 두가지 - ② 알고리즘

## ✓알고리즘 한판 정리

	선형회귀	로지스틱회귀	KNN	SVM	Decision Tree	Random Forest	Gradient Boost (GBM, XGB, LGBM)
<b>개념</b>	✓오차를 최소화 하는 직선, 평면	✓오차를 최소화 하는 직선, 평면 ✓직선을 로지스틱 함수로 변환 (0~1 사이 값으로)	✓예측할 데이터와 train set과의 거리 계산 ✓가까운 [k개 이웃의 y]의 평균으로 예측	✓마진을 최대화 하는 초 평면 찾기 ✓데이터 커널 변환	✓정보전달량 = 부모 불순도 - 자식 불순도 ✓정보 전달량이 가장 큰 변수를 기준으로 split	✓여러 개의 트리 ✓각각 예측 값의 평균 ✓행과 열에 대한 랜덤 : 조금씩 다른 트리들 생성	✓여러 개의 트리 ✓트리를 더해서 하나의 모델로 생성 ✓더해지는 트리는 오차를 줄이는 모델
<b>전제 조건</b>	✓NaN조치 ✓가변수화 ✓x들 간 독립	✓NaN조치 ✓가변수화 ✓x들 간 독립	✓NaN조치 ✓가변수화 ✓스케일링	✓NaN조치 ✓가변수화 ✓스케일링	✓NaN조치 ✓가변수화	✓NaN조치 ✓가변수화	✓NaN조치 ✓가변수화
<b>성능</b>	✓변수 선택 중요 ✓x가 많을 수록 복잡	✓변수 선택 중요 ✓x가 많을 수록 복잡	✓주요 hyper-parameter - n_neighbors : k 작을수록 복잡 - metric : 거리계산법	✓주요 hyper-parameter - C : 클수록 복잡 - gamma : 클수록 복잡	✓주요 hp - max_depth : 클수록 복잡 - min_samples_leaf : 작을수록 복잡	✓주요 hp 기본값으로도 충분! - n_estimators - max_features ✓기본값으로 생성된 모델 ==> 과적합 회피	✓주요 hp - n_estimators - learning_rate ✓XGB, LGBM : 과적합 회피를 위한 규제

# 회귀모델 평가

		오차의 크기			
$\hat{y}$ : 예측값		오차	제곱 오차	절대값 오차	오차율
$y$	$\hat{y}$	$y - \hat{y}$	$(y - \hat{y})^2$	$ y - \hat{y} $	$\left  \frac{y - \hat{y}}{y} \right $
6	4				
5	6				
12	9				
2	2				

평균

MSE  
RMSE

MAE

MAPE

평균 오차      평균 오차율

# 딥러닝 개념 - 학습 절차

✓ `model.fit(x_train, y_train)` 하는 순간...

단계① : 가중치에 (초기)값을 할당한다.

- 초기값은 랜덤으로 지정

단계② : (예측) 결과를 뽑는다.

단계③ : 오차를 계산한다.

단계④ : 오차를 줄이는 방향으로 가중치를 조정

- *Optimizer* : GD, Adam...

단계⑤ : 다시 단계①로 올라가 반복한다.

- *max iteration* 에 도달.(오차의 변동이 (거의) 없으면 끝.)

- 가중치(weight)의 다른 용어 **파라미터(parameter)**

$$medv = 1 \cdot lstat + 3$$

medv	lstat	$\hat{y}$
20	10	13
10	11	14
8	15	18

$$mse = \frac{\sum (y - \hat{y})^2}{n} = \frac{7^2 + 6^2 + 8^2}{3}$$

$$w_1: 1 \rightarrow 0.8$$

$$w_0: 3 \rightarrow 3.3$$

$$medv = w_1 \cdot lstat + w_0$$

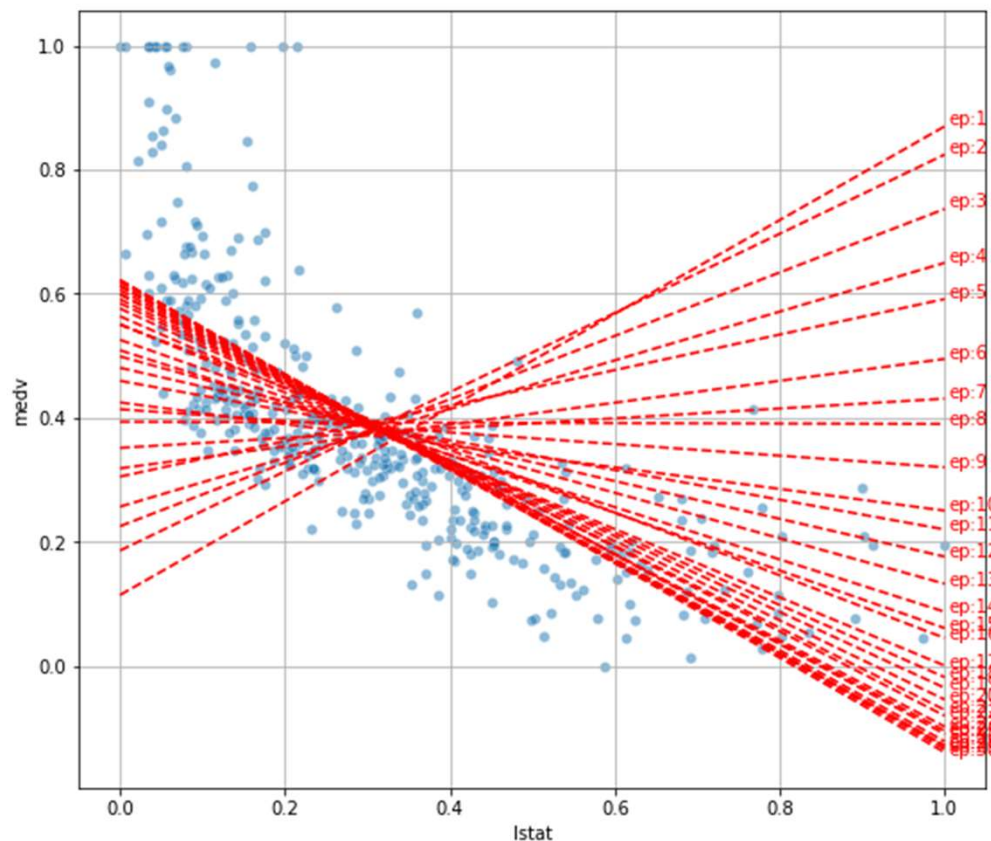
*forward propagation*

*back propagation*

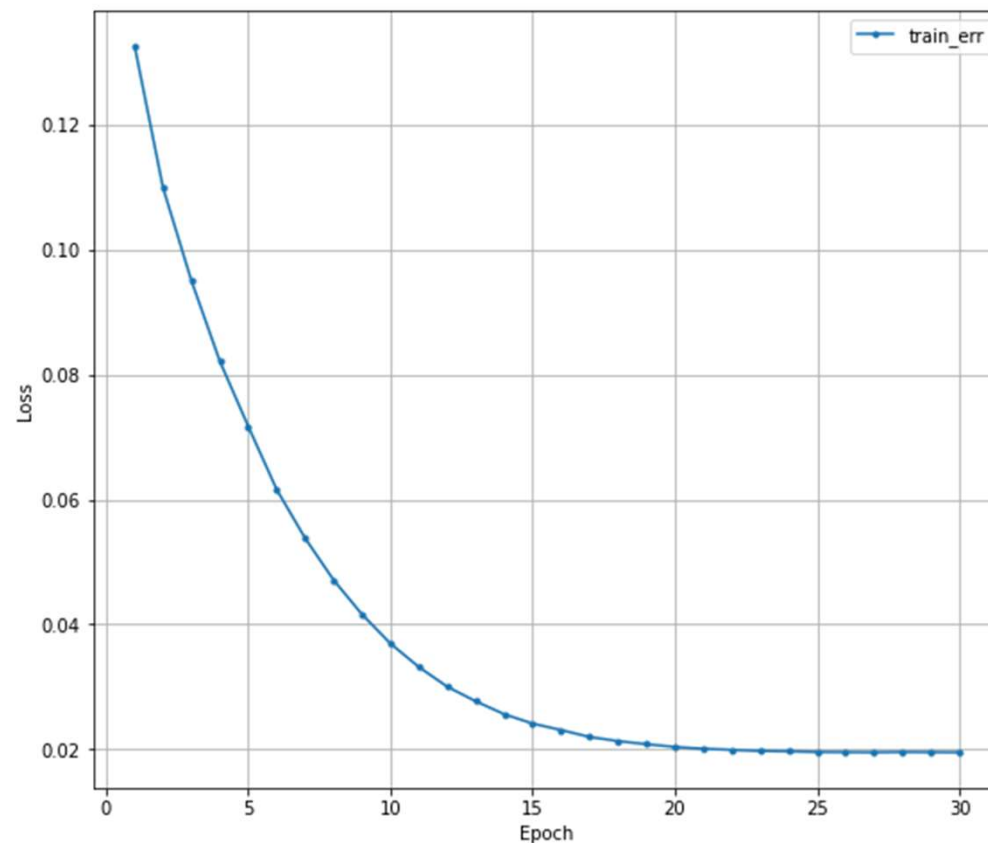
# 딥러닝 개념 - 학습 절차

✓ 30번 조정하며 최적의 Weight를 찾아가는 과정

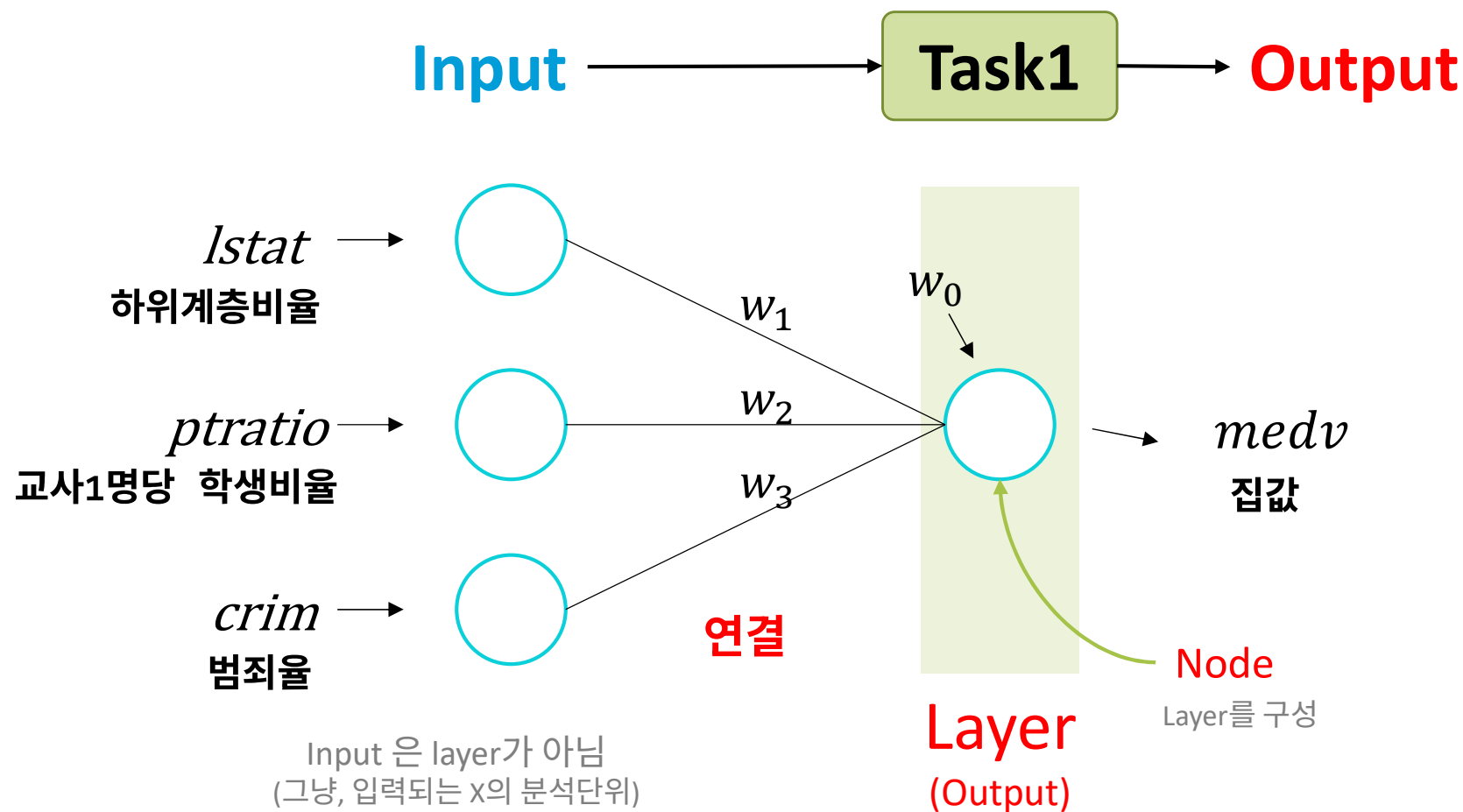
모델의 가중치가 업데이트되는 과정



모델의 오차가 줄어드는 과정

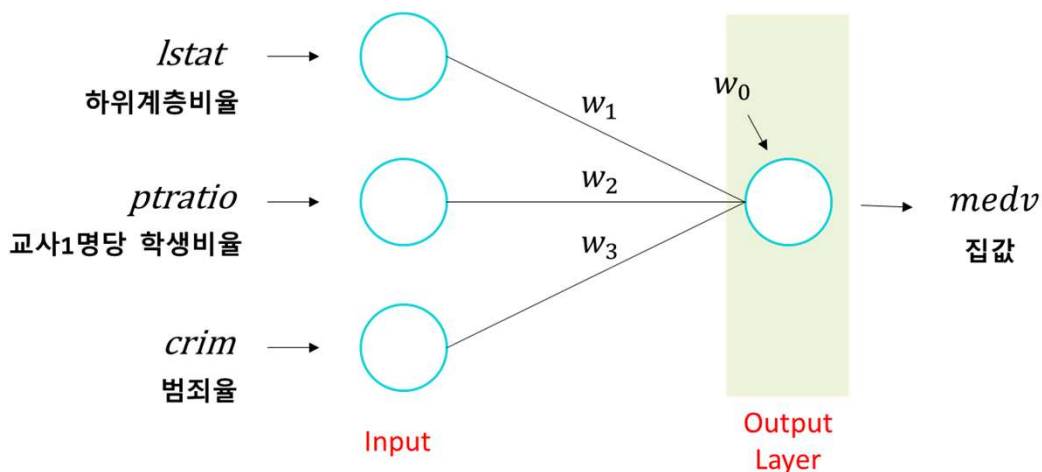


# 딥러닝 구조



$$medv = w_1 \cdot lstat + w_2 \cdot ptratio + w_3 \cdot crim + w_0$$

# 딥러닝 코드 - Dense



✓ **input\_shape = ( , )**

▪ 분석단위에 대한 shape

- 1차원 : (feature 수, )
- 2차원 : (rows, columns)

✓ **output**

▪ 예측 결과가 1개 변수(y가 1개 변수)

## Python Code

```
# 메모리 정리
clear_session()

# Sequential 타입
model = Sequential([Dense(1, input_shape = (nfeatures,))])

# 모델요약
model.summary()
```

Annotations in the code: **output** points to the `1` in `Dense(1, ...)`, and **input** points to `nfeatures`. A red **3** is placed above `nfeatures`.

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	4

=====  
 Total params: 4  
 Trainable params: 4  
 Non-trainable params: 0  
 =====



# 딥러닝 코드 - Compile

## ✓ 컴파일(Compile)

- 선언된 모델에 대해 몇 가지 설정을 한 후, 컴퓨터가 이해할 수 있는 형태로 변환하는 작업

### Python Code

```
model.compile(  
    optimizer = Adam(learning_rate = 0.1)  
    , loss='mse')
```

## ✓ loss function(오차함수)

- 오차 계산을 무엇으로 할지 결정
- mse : mean squared error, 회귀모델은 보통 mse로 오차 계산

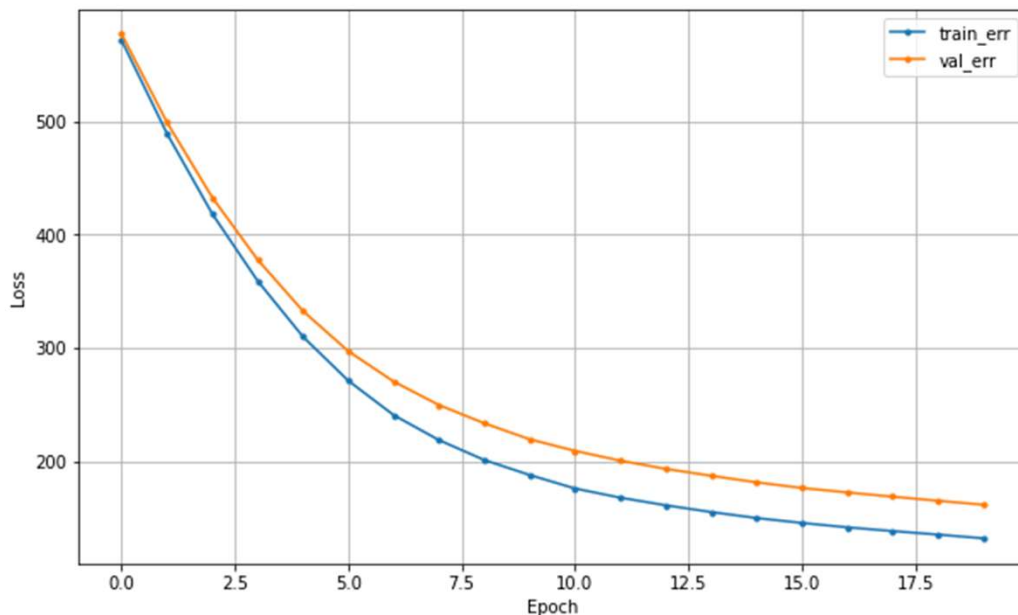
## ✓ optimizer

- 오차를 최소화 하도록 가중치를 조절하는 역할
- optimizer = 'adam' : learning\_rate 기본값 = 0.001
- optimizer = Adam(lr = 0.1) : 옵션 값 조정 가능
  - lr과 learning\_rate은 같지만, learning\_rate 사용을 권장

# 딥러닝 코드 - 학습곡선

## ✓ .history

- 학습을 수행하는 과정 중에
- 가중치가 업데이트 되면서
- 그때그때마다의 성능을 측정하여 기록
- 학습 시 계산된 오차 기록
- 그것을 저장한 후 차트를 그리면...



## Python Code

```
history = model.fit(x_train, y_train,
                    epochs = 20,
                    validation_split=0.2).history
```

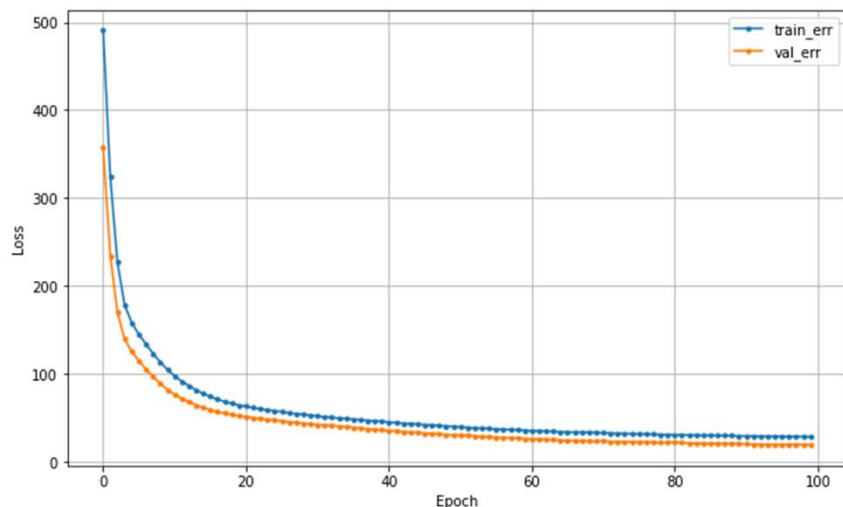
Epoch 1/20	11/11 [=====]	- 1s 31ms/step - loss: 571.5110 - val_loss: 577.0120
Epoch 2/20	11/11 [=====]	- 0s 8ms/step - loss: 489.2647 - val_loss: 499.1079
Epoch 3/20	11/11 [=====]	- 0s 11ms/step - loss: 418.2319 - val_loss: 432.6833
Epoch 4/20	11/11 [=====]	- 0s 11ms/step - loss: 359.0570 - val_loss: 377.7811
Epoch 5/20	11/11 [=====]	- 0s 7ms/step - loss: 309.7421 - val_loss: 332.4446
Epoch 6/20	11/11 [=====]	- 0s 7ms/step - loss: 270.8658 - val_loss: 296.9759
Epoch 7/20	11/11 [=====]	- 0s 6ms/step - loss: 240.5217 - val_loss: 270.1676
Epoch 8/20	11/11 [=====]	- 0s 12ms/step - loss: 218.4201 - val_loss: 249.3737
Epoch 9/20	11/11 [=====]	- 0s 7ms/step - loss: 200.8222 - val_loss: 233.2946
Epoch 10/20	11/11 [=====]	- 0s 10ms/step - loss: 187.6137 - val_loss: 219.1513
Epoch 11/20	11/11 [=====]	- 0s 7ms/step - loss: 175.6799 - val_loss: 208.9160
Epoch 12/20	11/11 [=====]	- 0s 5ms/step - loss: 167.5694 - val_loss: 200.2585
Epoch 13/20	11/11 [=====]	- 0s 7ms/step - loss: 160.8632 - val_loss: 193.0237
Epoch 14/20	11/11 [=====]	- 0s 6ms/step - loss: 154.9114 - val_loss: 186.9379
Epoch 15/20	11/11 [=====]	- 0s 9ms/step - loss: 149.6200 - val_loss: 181.1366
Epoch 16/20	11/11 [=====]	- 0s 6ms/step - loss: 145.2706 - val_loss: 176.1777
Epoch 17/20	11/11 [=====]	- 0s 7ms/step - loss: 141.4094 - val_loss: 172.2429
Epoch 18/20	11/11 [=====]	- 0s 8ms/step - loss: 138.0926 - val_loss: 168.4736
Epoch 19/20	11/11 [=====]	- 0s 7ms/step - loss: 135.0007 - val_loss: 164.8660
Epoch 20/20	11/11 [=====]	- 0s 13ms/step - loss: 131.7069 - val_loss: 161.3870

# 딥러닝 코드 - 학습곡선

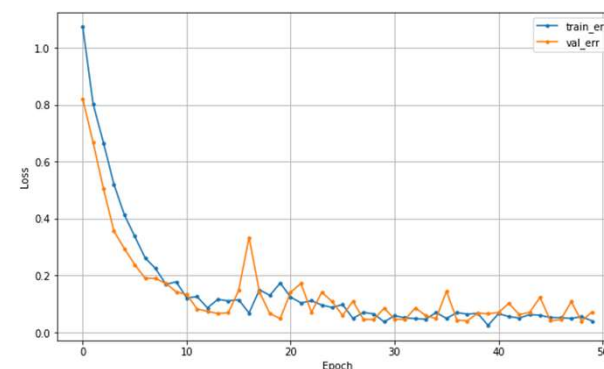


## ✓ 바람직한 곡선의 모습

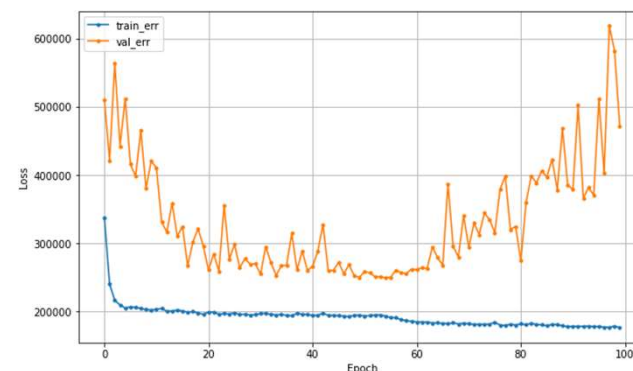
- Epoch가 증가하면서 Loss가 큰 폭으로 축소
- 점차 Loss 감소 폭이 줄어들면서 완만해짐.



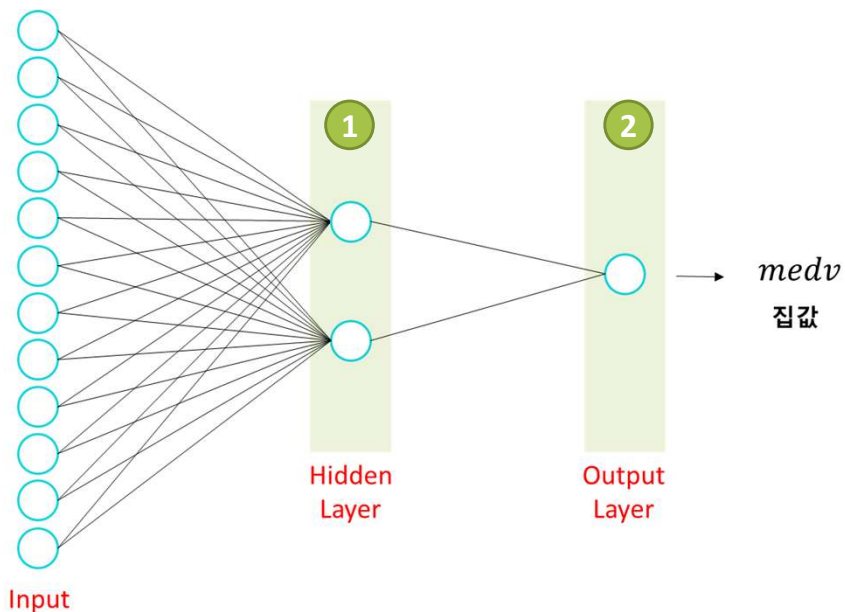
- Loss가 줄어드기는 하나, 들쭉날쭉  
→ Learning\_rate을 줄여 보시다.



- Val\_loss가 줄어들다가 다시 상승(과적합)  
→ Epochs와 learning\_rate을 조절해 보시다.



# 딥러닝 구조 - Hidden Layer



✓ layer 여러 개 : **리스트[ ]** 로 입력

✓ hidden layer

- input\_shape 는 첫번째 layer만 필요
- activation
  - 히든 레이어는 활성화함수를 필요로 합니다.
  - 활성화함수는 보통 'relu'를 사용

✓ output layer

- 예측 결과가 1개

## Python Code

```
# 메모리 정리
clear_session()

# Sequential 타입 모델 선언
model3 = Sequential([Dense(2, input_shape = (nfeatures,)
                        , activation = 'relu')
                    , Dense(1)])

# 모델요약
model3.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 2)	26
dense_1 (Dense)	(None, 1)	3

```
=====
Total params: 29
Trainable params: 29
Non-trainable params: 0
=====
```

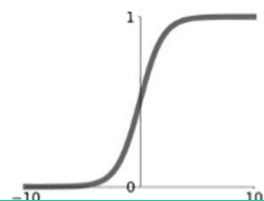
# 활성화 함수 Activation Function

## ✓ 그래서 활성화 함수는...

- Hidden Layer에서는 : 선형함수를 비선형 함수로 변환
- Output Layer에서는 : 결과값을 다른 값으로 변환해 주는 역할
  - 주로 분류 Classification 모델에서 필요

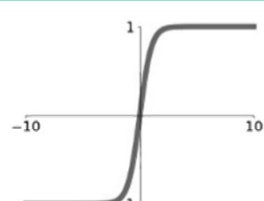
### Sigmoid

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



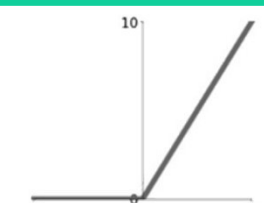
### tanh

$$\tanh(x)$$



### ReLU

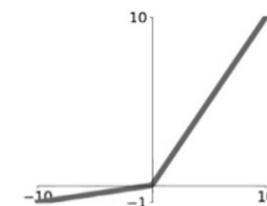
$$\max(0, x)$$



Hidden layer  
국룰

### Leaky ReLU

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

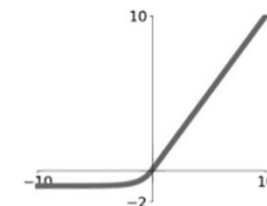


### Maxout

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

### ELU

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



# 요약 : 회귀 모델링

## ✓ 딥러닝 전처리

- NaN 조치, 가변수화, 스케일링

## ✓ Layer

- 첫번째 Layer는 input\_shape를 받는다.(분석단위의 shape)
  - 2차원 데이터셋의 분석단위 1차원 → shape는 ( feature수, )
- Output layer의 node 수 : 1
- Activation Function
  - Hidden layer에 필요 :
    - 비선형 모델로 만들려고 → hidden layer를 여럿 쌓아서 성능을 높이려고.
  - 회귀 모델링에서 Output Layer에는 활성화 함수 필요하지 않음!

구분	Hidden Layer	Output Layer		Compile	
	Activation	Activation	Node수	optimizer	loss
Regression	relu	X	1	adam	mse

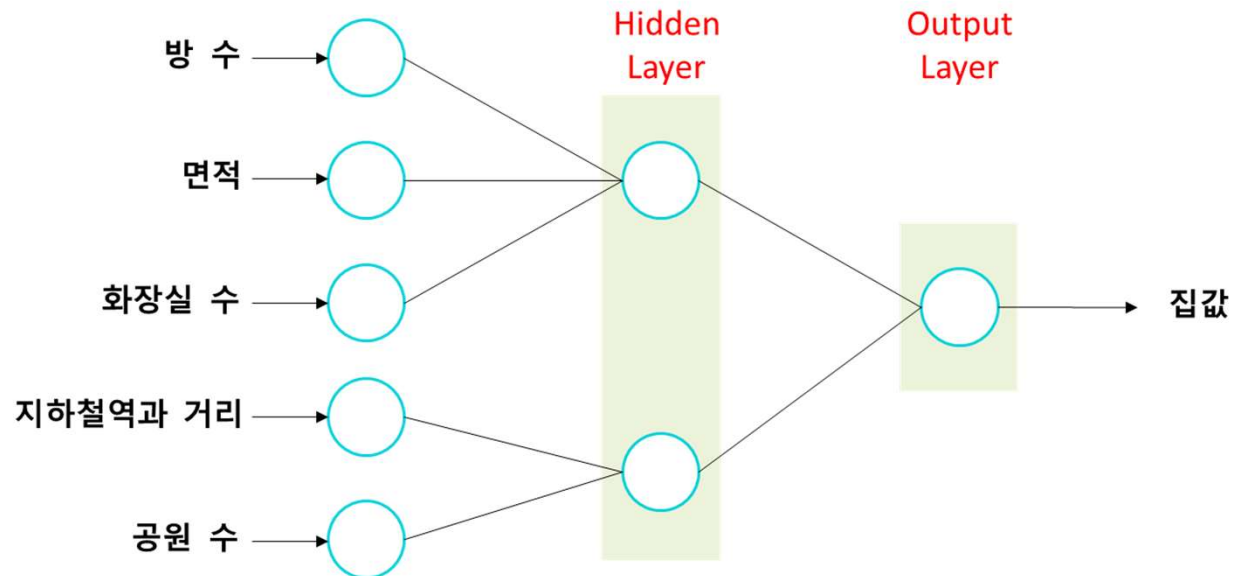
KT AIVLE School

# 2일차 정리

# Hidden Layer에서 무슨 일이 일어나는가?

## ✓ 처음으로 돌아와서...Hidden Layer에서는 어떤 일이 일어났나요?

- 기존 데이터를 받아들여,
- (우리는 정확히 알기 어렵지만) 뭔가 **새로운 특징(New Feature)**을 만들어 냈습니다.
- 그 특징은 분명히 예측된 값과 실제 값 사이의 오차를 **최소화** 해주는
- **유익한 특징**일 것입니다. (여기서-우리는-믿음-아-필요합니다-^^)
- Hidden Layer에서는 기존 데이터가 **새롭게 표현(Representation)** 되었습니다.  
**Feature Engineering**이 진행된 것입니다!

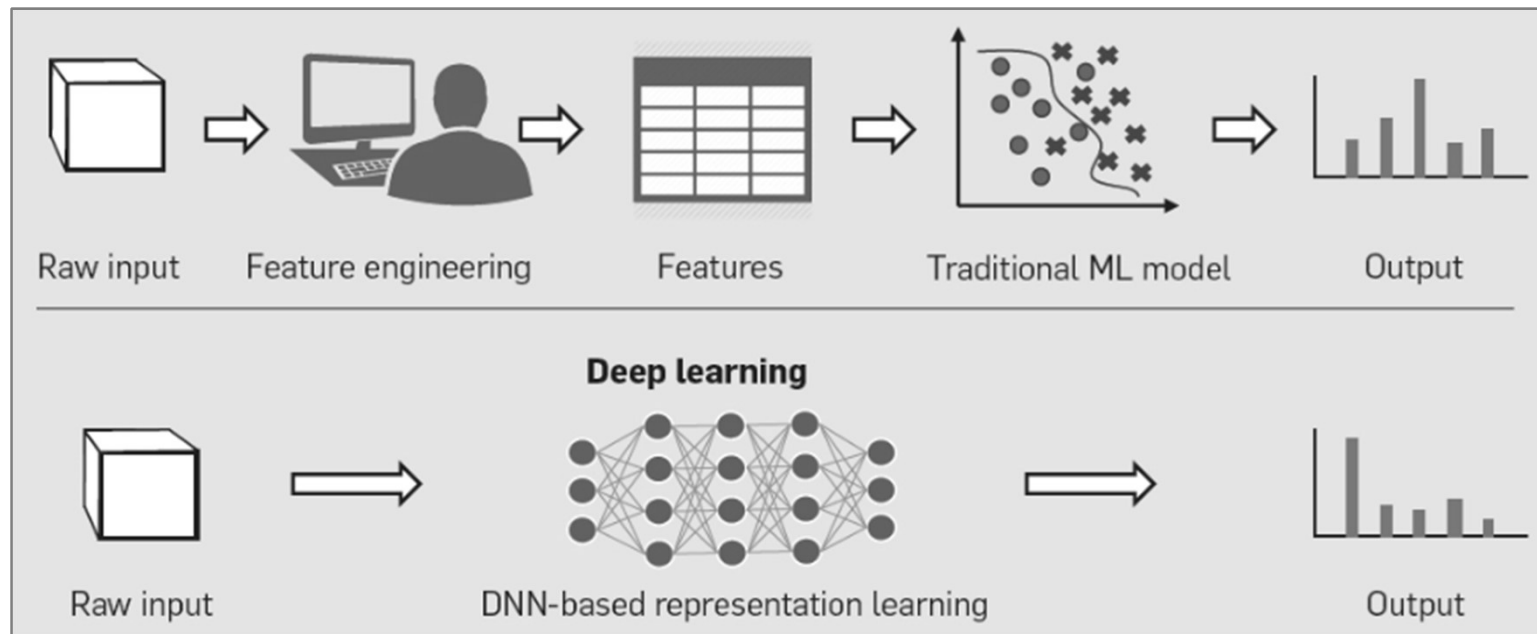




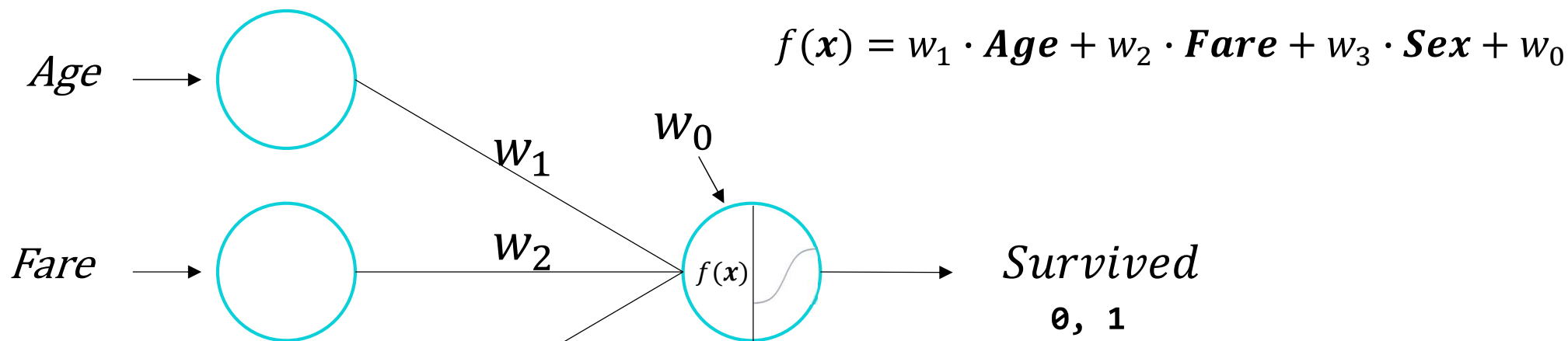
# Feature Representation

## ✓ Deep Learning → Representation Learning

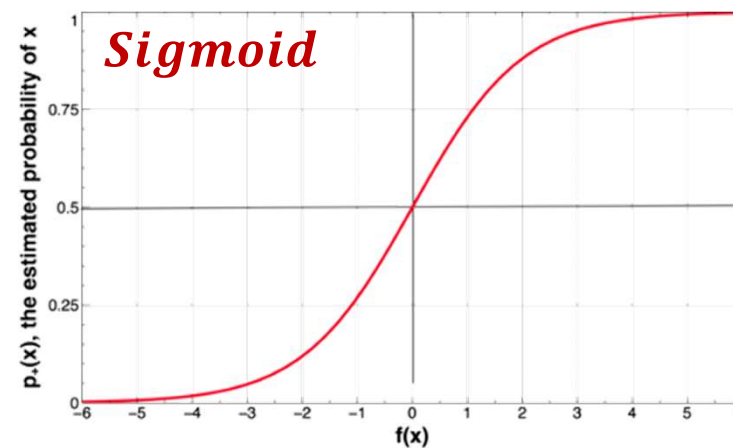
**Machine Learning**



# 딥러닝 구조 - 이진분류



- ✓ Node의 결과를 변환해주는 함수가 필요
  - 그것을 **활성 함수** Activation Function 라고 합니다.



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

# 딥러닝 구조 – 활성화 함수 Activation Function

✓ node의 결과를 변환시켜 주는 역할

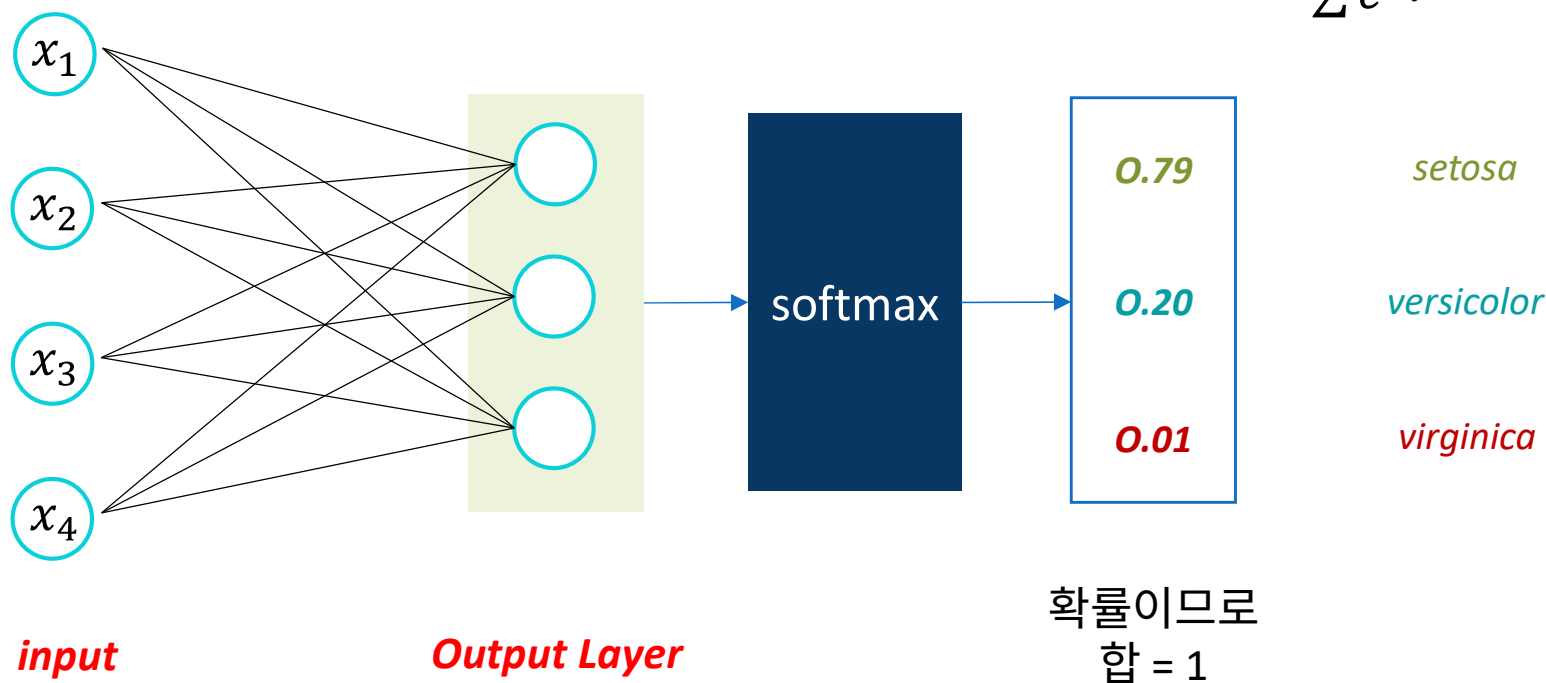
Layer	Activation Function		기능
Hidden Layer	ReLU		좀 더 깊이 있는 학습(Deep Learning)을 시키려고. (Hidden Layer를 여러 층 쌓으려고) (선형 모델을 비선형 모델로 바꾸려고)
Output Layer	회귀	X	X
	이진분류	sigmoid	결과를 0, 1로 변환하기 위해
	다중분류	softmax	각 범주에 대한 결과를 범주별 확률 값으로 변환

# 딥러닝 구조 - Output Layer

## ✓ Softmax

- 각 Class 별(Output Node)로 예측한 값을, 하나의 확률 값으로 변환.

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum e^{x_i}}, \quad e = 2.718 \dots$$



# SUMMARY



## 요약 : 회귀 vs 이진분류 vs 다중분류

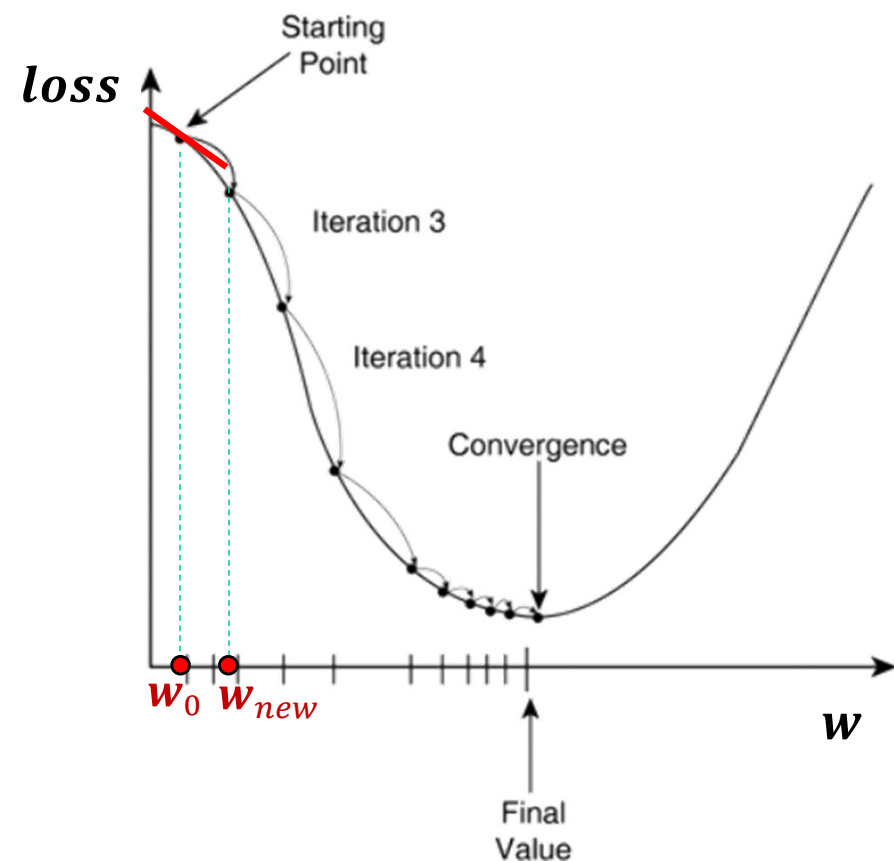
		Regression	Two-Class	Multi-Class
전처리	x	가변수화, 스케일링		
	y			정수 인코딩 원핫 인코딩
모델링	은닉층	activation = 'relu'		
	출력층	activation : 없음 Node 수 : 1	activation = 'sigmoid' node 수 : 1	activation = 'softmax' node 수 : y의 class 수
	컴파일 (loss)	mse	binary_crossentropy	sparse_categorical_crossentropy categorical_crossentropy
검증	예측결과 처리		np.where(pred>.5, 1, 0)	np.argmax(pred, axis = 1)

# [참조]가중치 업데이트

- ✓ Gradient : 기울기(벡터)
- ✓ Gradient Decent(경사 하강법, optimizer의 기본)

- $w$ 의 초기값 지정 :  $w_0$
- 초기값에서의 기울기(방향) 확인 :  $\frac{df(w)}{dw}$ ,  $w = w_0$ 
  - 기울기가 - 이면  $x$  는 오른쪽(+방향)
  - 기울기가 + 이면  $x$  는 왼쪽(- 방향)
- 조금 **조정** :  $\eta \times \frac{df(w)}{dw}$ 
  - $\eta$  : eta, 조정하는 비율, *Learning Rate*

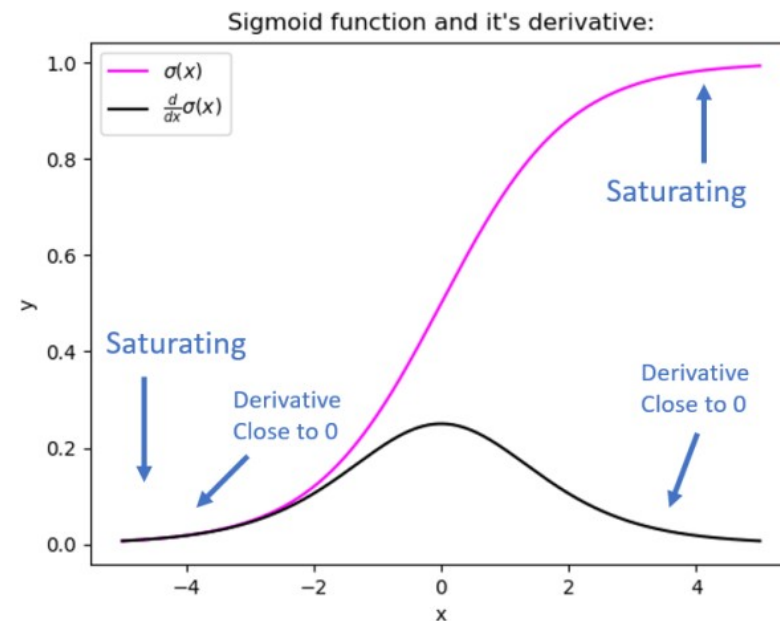
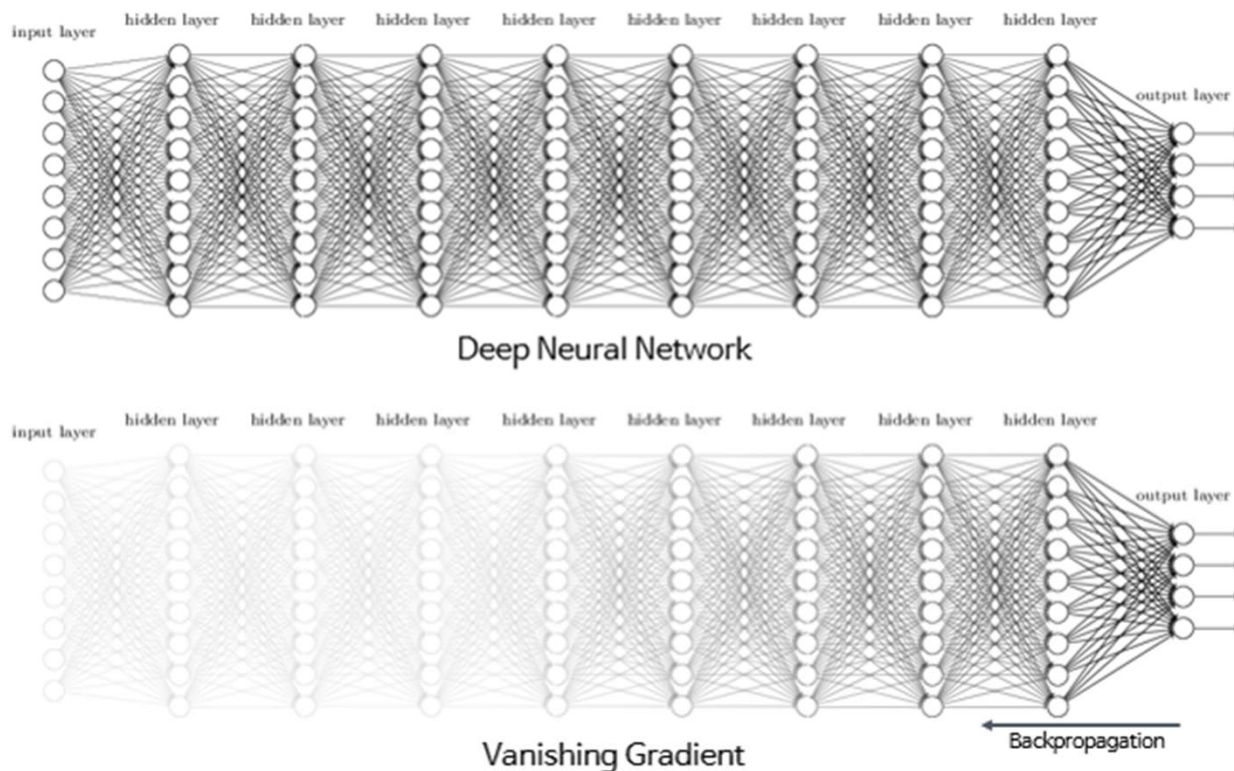
$$w_{new} = w_0 - \eta \times \frac{\partial f(w)}{\partial w}$$



# [참조] Vanishing Gradient(기울기 소실)

## ✓ 기울기 소실

- 네트워크의 깊은 부분으로 갈수록 기울기가 점점 작아져서, 가중치가 거의 또는 전혀 업데이트되지 않게 되는 현상
- 초기 activation = 'sigmoid' 를 사용 → 특히 기울기 소실 문제가 심각



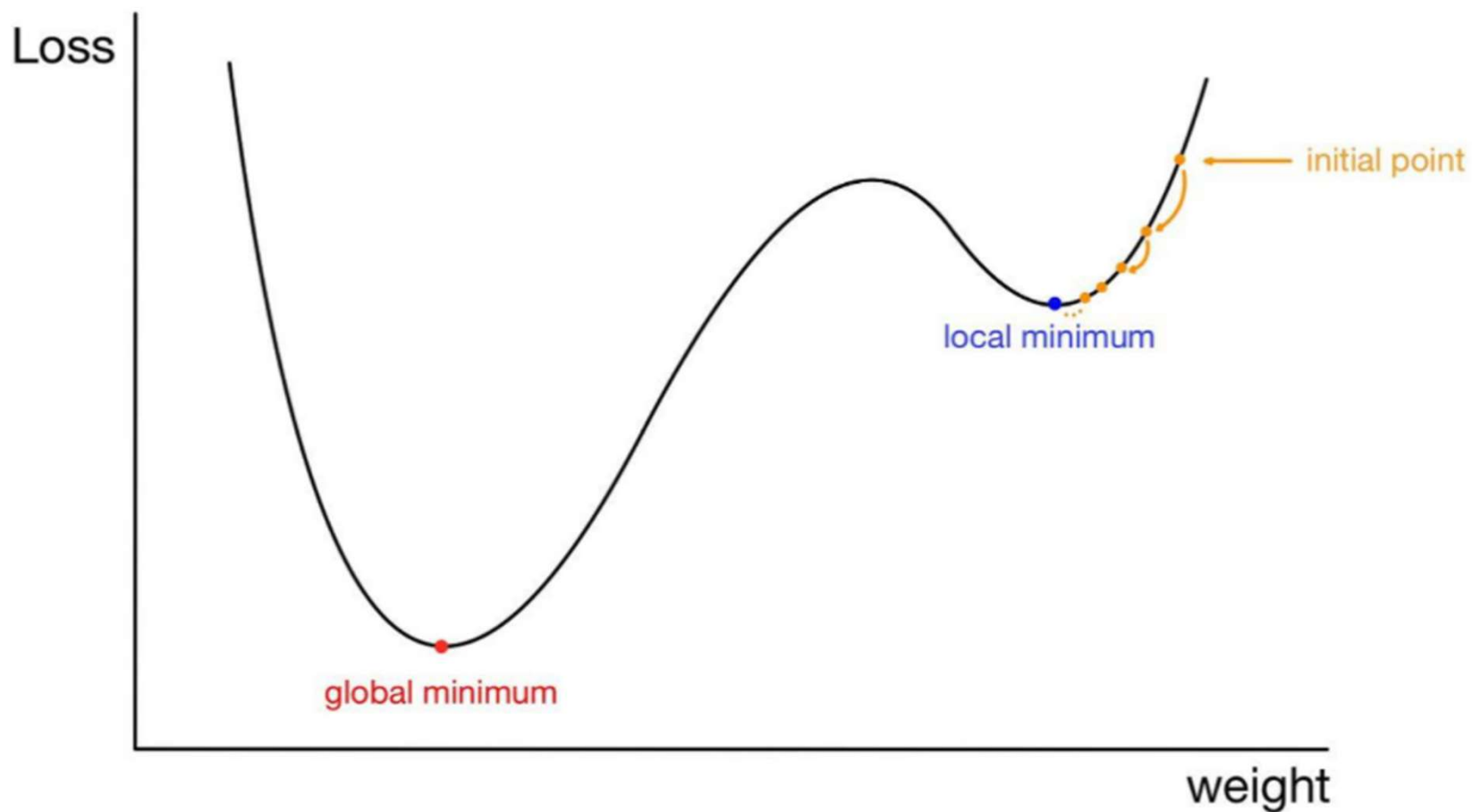
# [참조] Vanishing Gradient(기울기 소실)

## ✓기울기 소실 문제를 최소화 하기 위한 노력

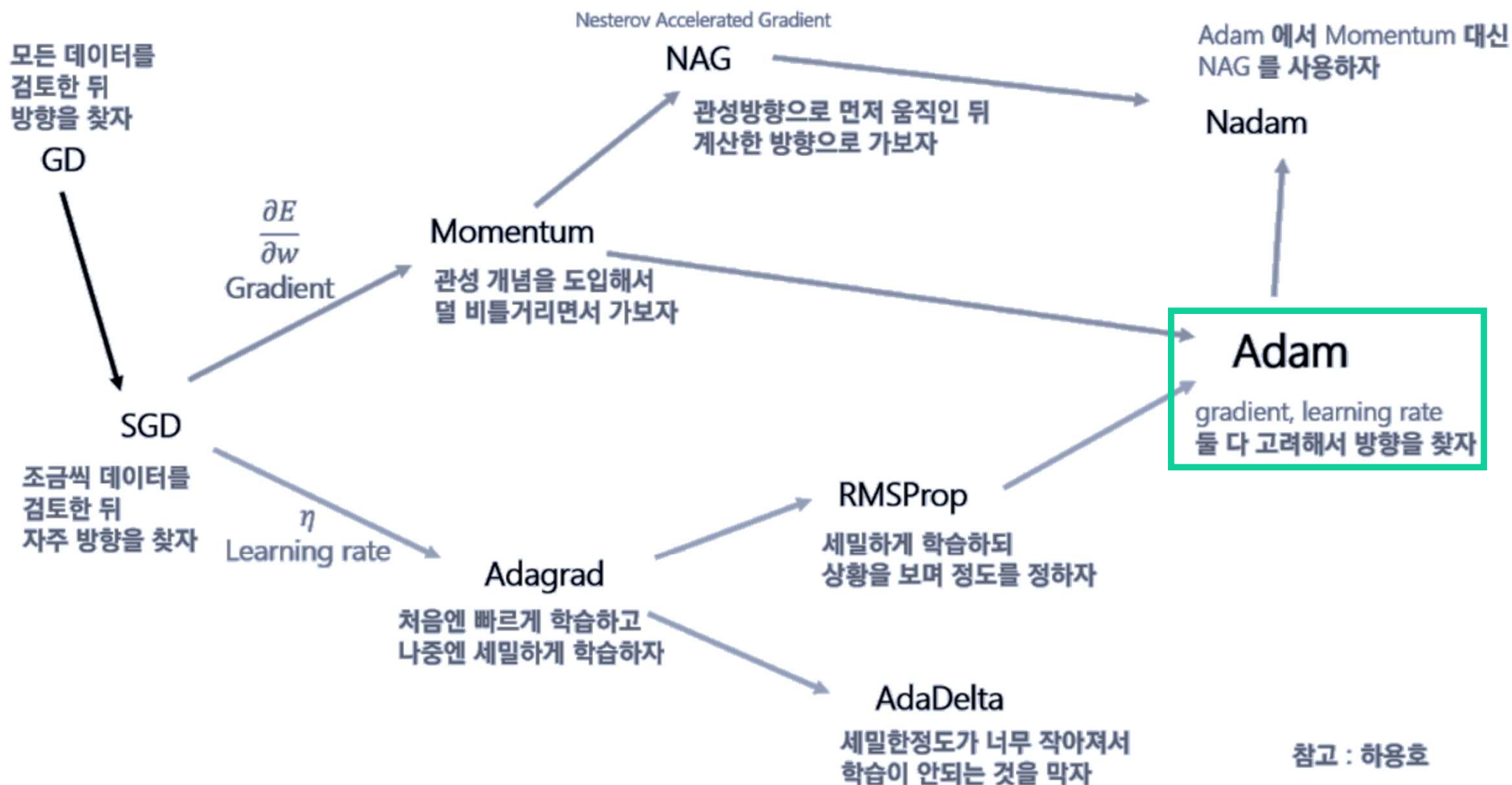
- 활성화 함수 조정
  - ReLU(Rectified Linear Unit)
    - 음수 입력 - 0 출력, 양수 입력 - 그대로 출력 → 양의 기울기 유지하여 기울기 소실 문제를 완화
  - ReLU의 변형된 활성화 함수
    - Leaky ReLU, PReLU, ELU → 음수 입력에 대해서도 매우 작은 기울기를 허용
- 그 외 방법들
  - 가중치 초기화 기법
  - 배치 정규화(Batch Normalization)
  - Residual Connections
  - Gradient Clipping



# [참조]Local Minima problem



# [참조]Local Minima problem → Optimizer



# [참조] 모델의 성능 최적화

## ✓ 모델링의 목표

- ~~완벽한~~적절한 예측력을 얻기 위해
- 적절한 복잡도의 모델을 생성

➔ 일반화 성능

## ✓ 모델의 복잡도

- 학습데이터 안에 포함된 패턴을 모델에 반영한 정도
- 대체로 하이퍼 파라미터 조정에 따라 복잡도가 달라짐

## ✓ 하이퍼 파라미터(hyper-parameter)

- 우리가 조정해 줘야 할 대상

KNN	DT	XGB	DL
n_neighbors metric	max_depth min_samples_leaf	n_estimators max_depth learning_rate	Hidden Layer 수, node 수 learning_rate, epochs ...

# [참조] 모델의 성능 최적화

## ✓ 하이퍼 파라미터 튜닝

- 튜닝 기본 방법 : 다양한 값들도 시도
  - Random Search : 지정한 범위 내에서 무작위로 시도
  - Grid Search : 지정한 범위 내에서, 모든 경우의 수 만큼 시도
- 최적의 모델 선정 → 검증 성능으로 평가
- 검증 성능을 기반으로 최적의 모델을 선정하면, 과적합을 피할 수 있음.