AIVLE

KT AIVLE School

# 1일차 정리

AIVLE

Make it possible

# 전체 Process(CRISP-DM)

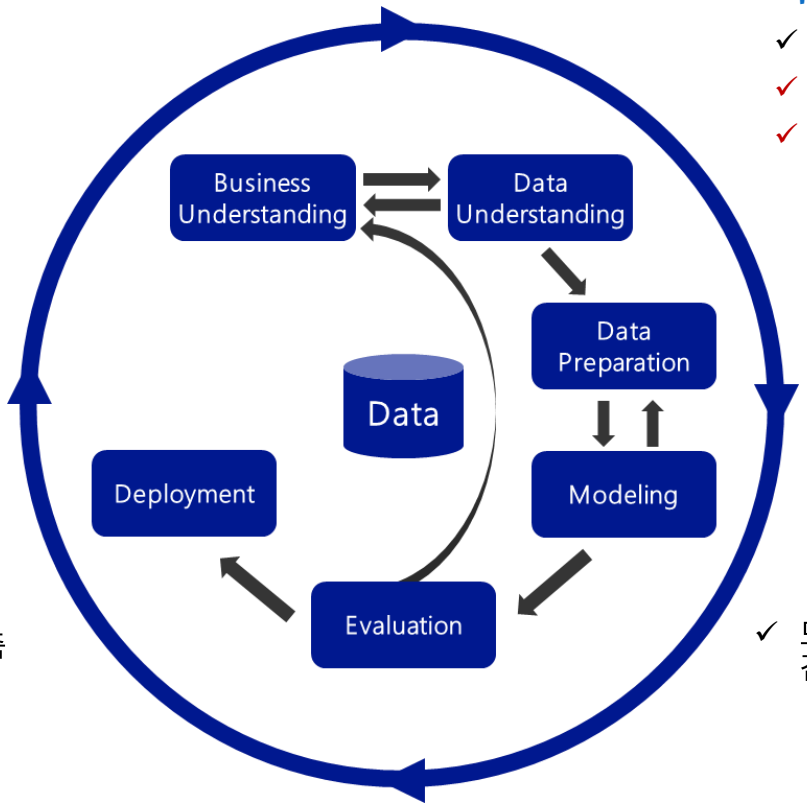
## 무엇이 문제인가?

- ✓ 비즈니스 **문제정의**
- ✓ 데이터분석 방향, 목표
- ✓ 초기 **가설 수립**

$x \rightarrow y$

## 어디? 진짜?

- ✓ 원본식별
- ✓ **분석을 위한 구조 만들기**
- ✓ **데이터분석EDA & CDA**




- ✓ 모델링을 위한 데이터 구조 만들기
  - 모든 셀은 **값**이 있어야 한다.
  - 모든 값은 **숫자** 여야 한다.
  - (필요시) 숫자의 **범위**가 일치

- ✓ 모델 관리
- ✓ AI 서비스 구축

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

# 회귀모델 평가

오차의 크기

$\hat{y}$ : 예측값

| | | 오차 | 제곱 오차 | 절대값 오차 | 오차율 |
|---|---|---|---|---|---|
| $y$ | $\hat{y}$ | $y - \hat{y}$ | $(y - \hat{y})^2$ | $\lvert y - \hat{y} \rvert$ | $\left\lvert \frac{y - \hat{y}}{y} \right\rvert$ |
| 6 | 4 | | | | |
| 5 | 6 | | | | |
| 12 | 9 | | | | |
| 2 | 2 | | | | |
| | | 평균 | MSE<br>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*

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

$$\boldsymbol{medv = w_1 \cdot lstat + w_0}$$

**단계② : (예측) 결과를 뽑는다.**

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

**단계③ : 오차를 계산한다.**

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

*forward propagation*

**단계④ : 오차를 줄이는 방향으로 가중치를 조정**
- *Optimizer : GD, Adam...*

$$\boldsymbol{w_1: 1 \rightarrow 0.8}$$
$$\boldsymbol{w_0: 3 \rightarrow 3.3}$$

*back propagation*

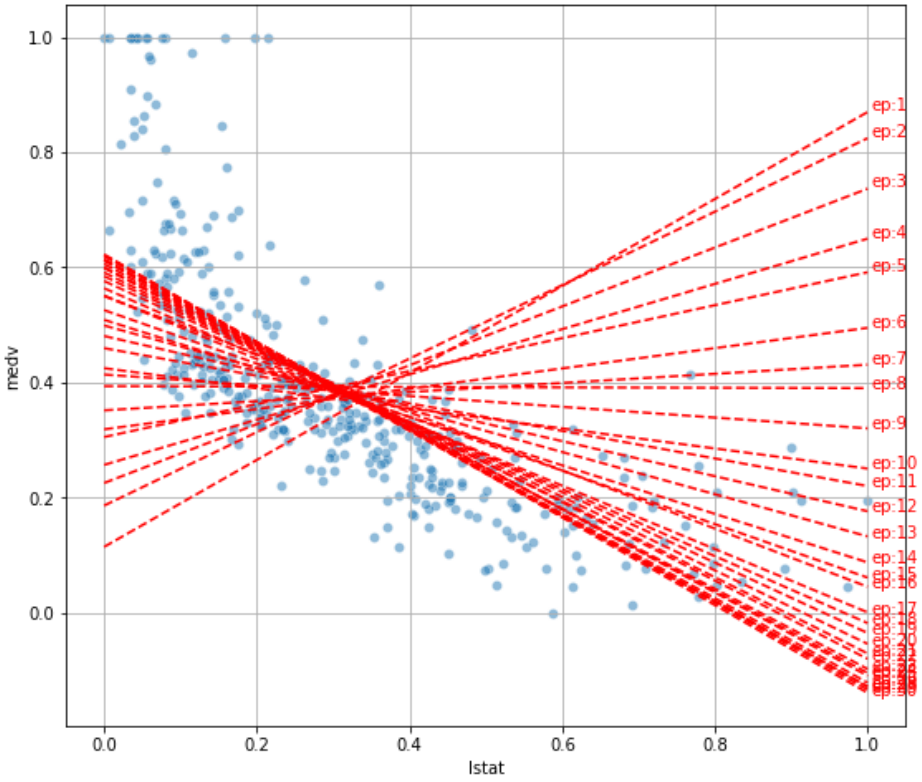
**단계⑤ : 다시 단계①로 올라가 반복한다.**
- *max iteration에 도달.(오차의 변동이 (거의) 없으면 끝.)*

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

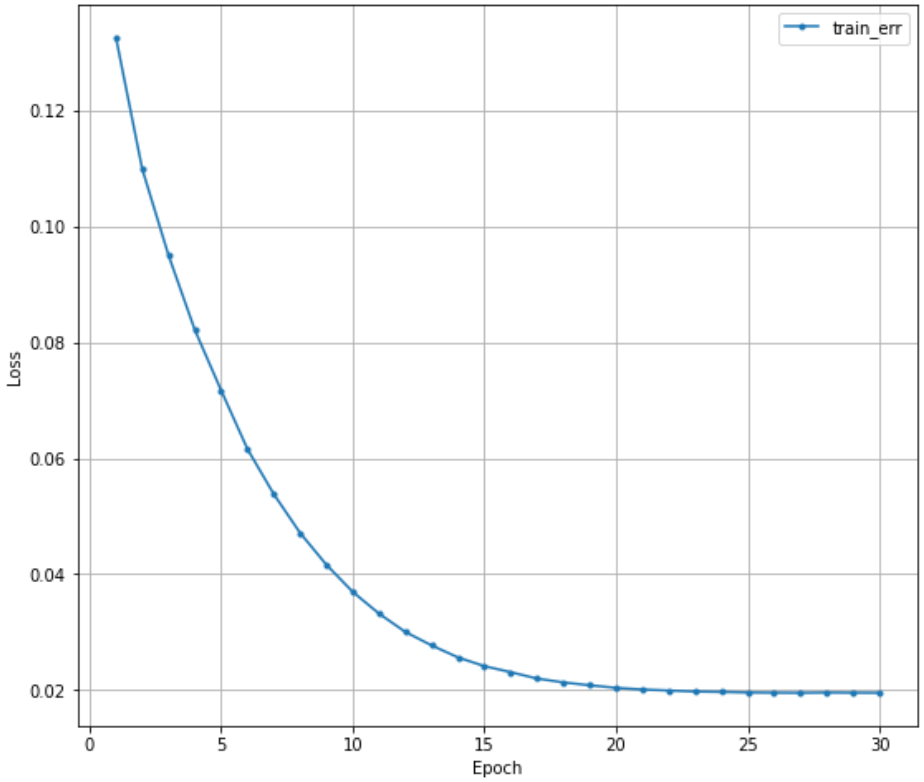
# 딥러닝 개념 - 학습 절차

## ✓30번 조정하며 최적의 Weight를 찾아가는 과정

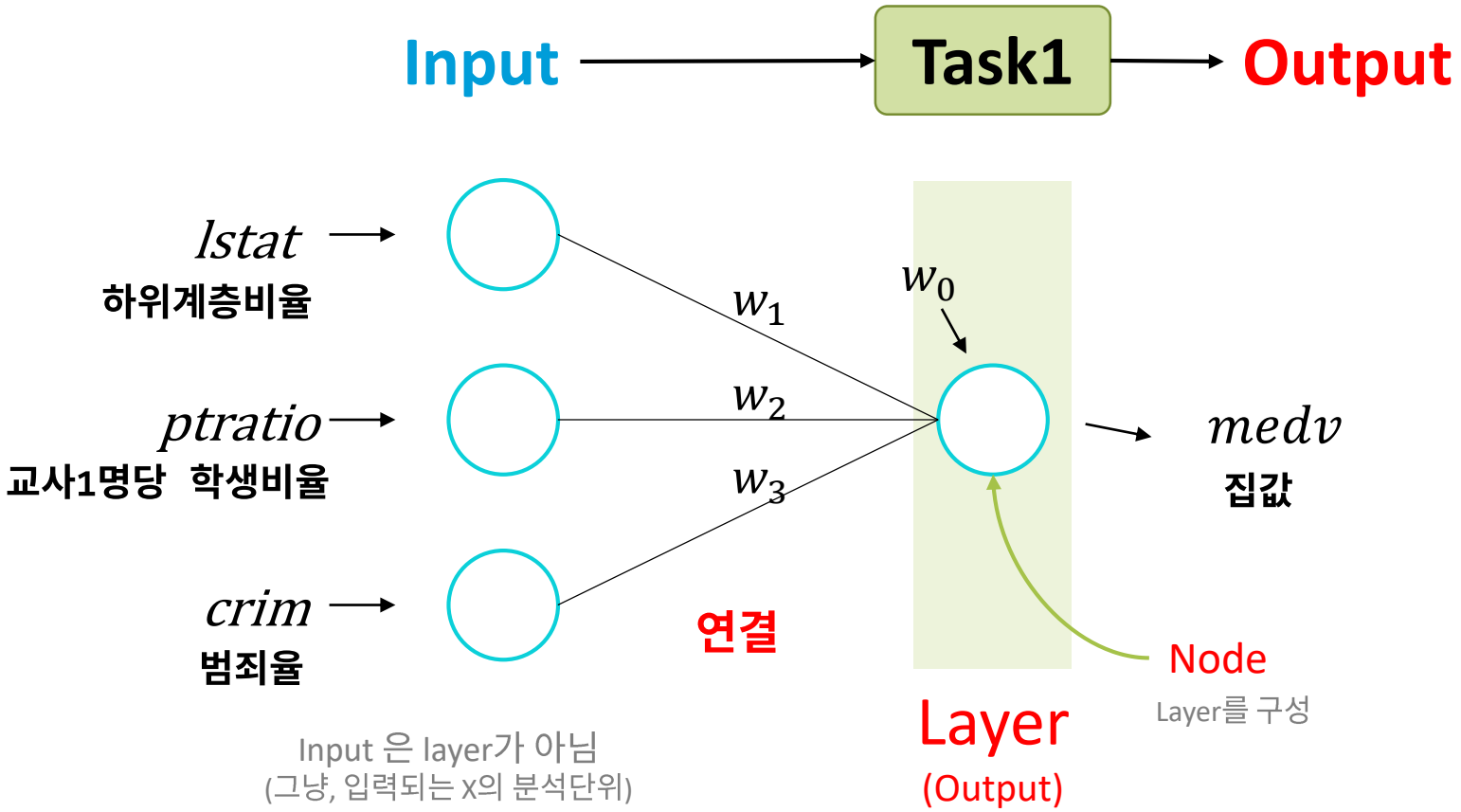
### 모델의 가중치가 업데이트되는 과정




### 모델의 오차가 줄어드는 과정

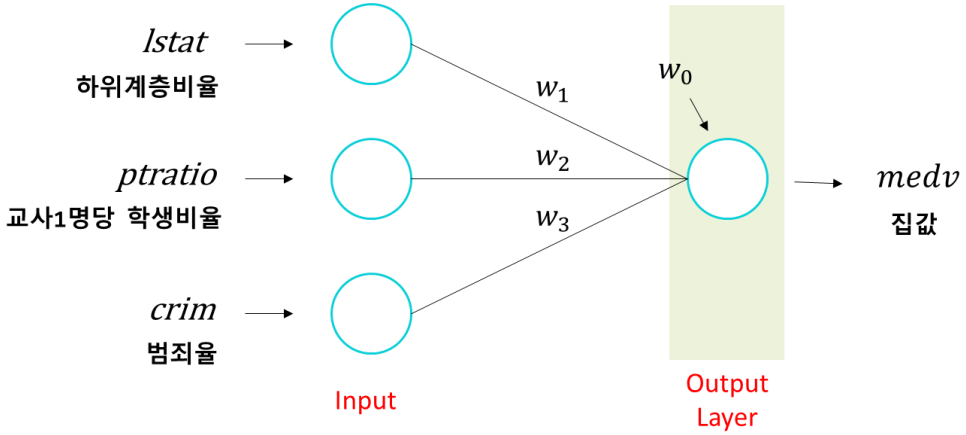


# 딥러닝 구조




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

# 딥러닝 코드 - Dense




✓ **input_shape = ( , )**

- **분석단위**에 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()
```

*3*

*output*

*input*

```
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

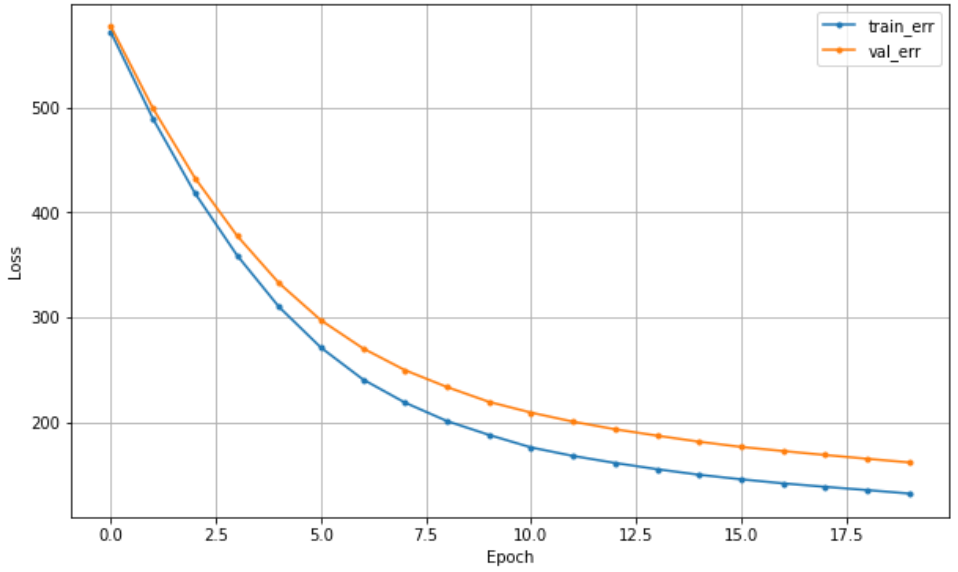
## ✓ optimizer

- 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**

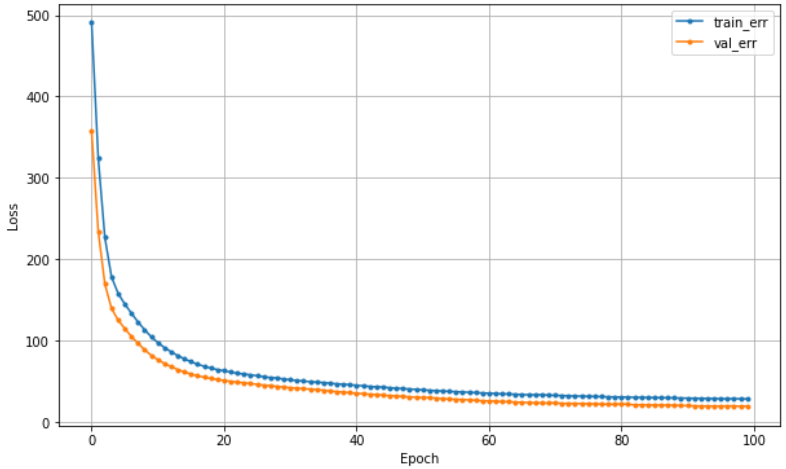
```
history = model.fit(x_train, y_train,
                    epochs = 20,
                    validation_split=0.2).history
```

```
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

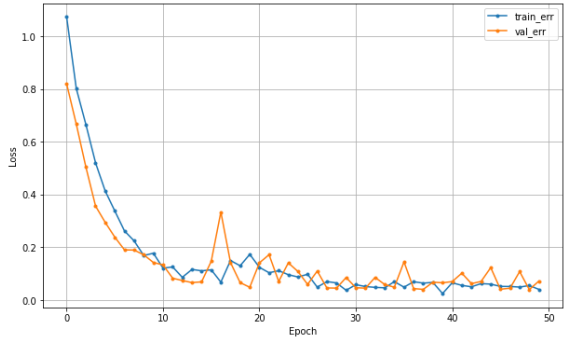
✓ **바람직한 곡선의 모습**

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



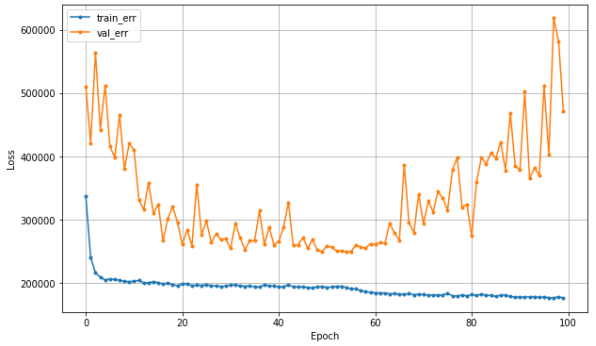

- Loss가 줄어들기는 하나, 들쑥날쑥

➔ Learning_rate을 줄여 봅시다.

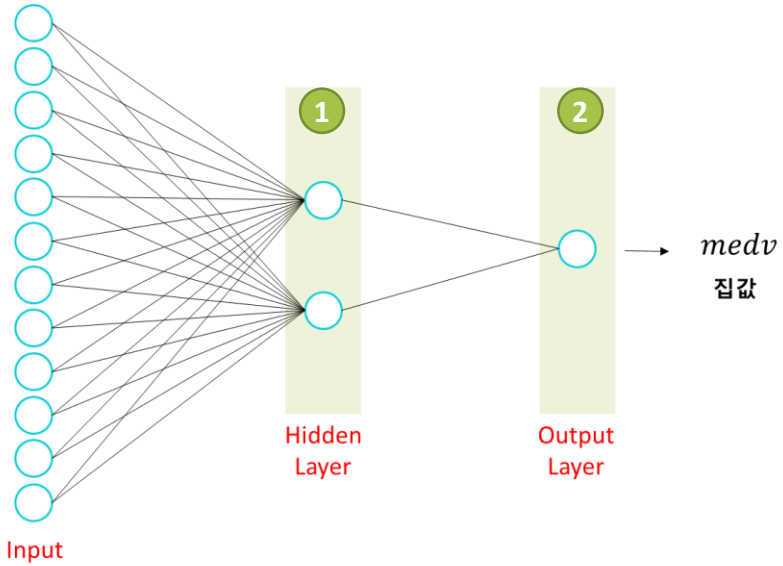



- Val_loss가 줄어들다가 다시 상승(과적합)

➔ Epochs와 learning_rate을 조절해 봅시다.



# 딥러닝 구조 - Hidden Layer




✓ **layer 여러 개 : 리스트[ ] 로 입력**

✓ **hidden layer**

- input_shape 는 첫번째 layer만 필요
- activation
  - 히든 레이어는 활성함수를 필요로 합니다.
  - 활성함수는 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()
```

1 2

```
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

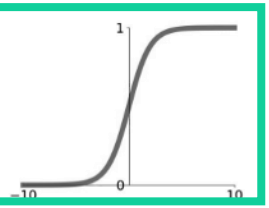
## ✓그래서 활성화 함수는...

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

Hidden layer
국룰

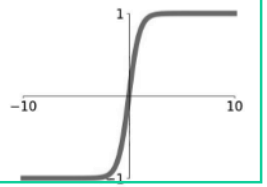
**Sigmoid**

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



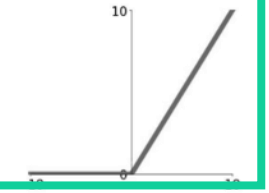
**tanh**

$$\tanh(x)$$



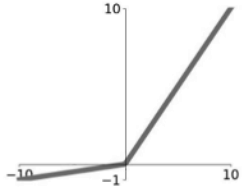
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

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

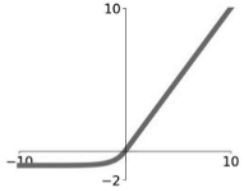


**Maxout**

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

**ELU**

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




# 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 |

AIVLE

KT AIVLE School

# 2일차 정리

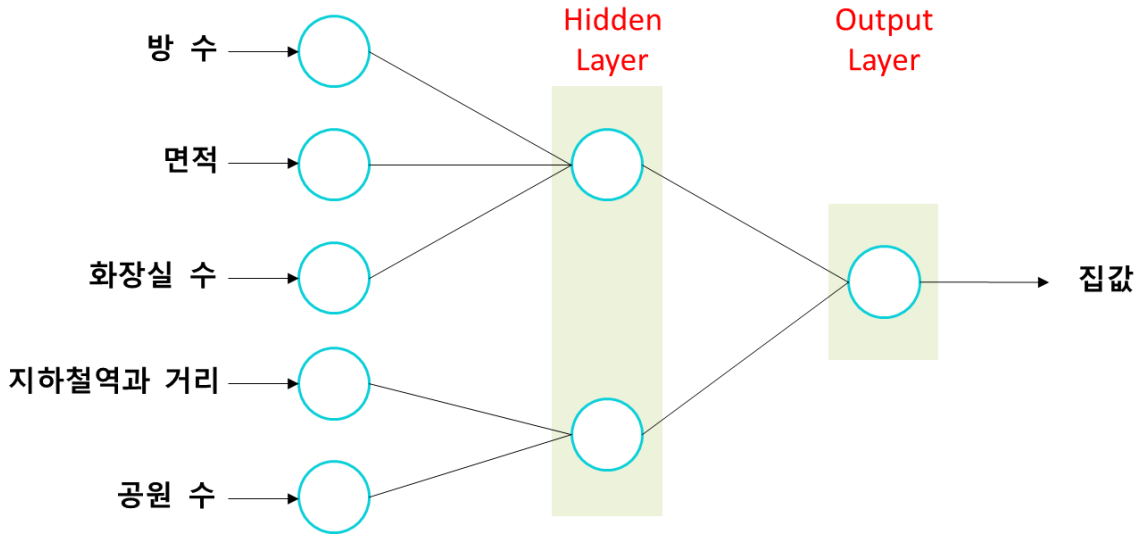
AIVLE

Make it possible

# Hidden Layer에서 무슨 일이 일어나는가?

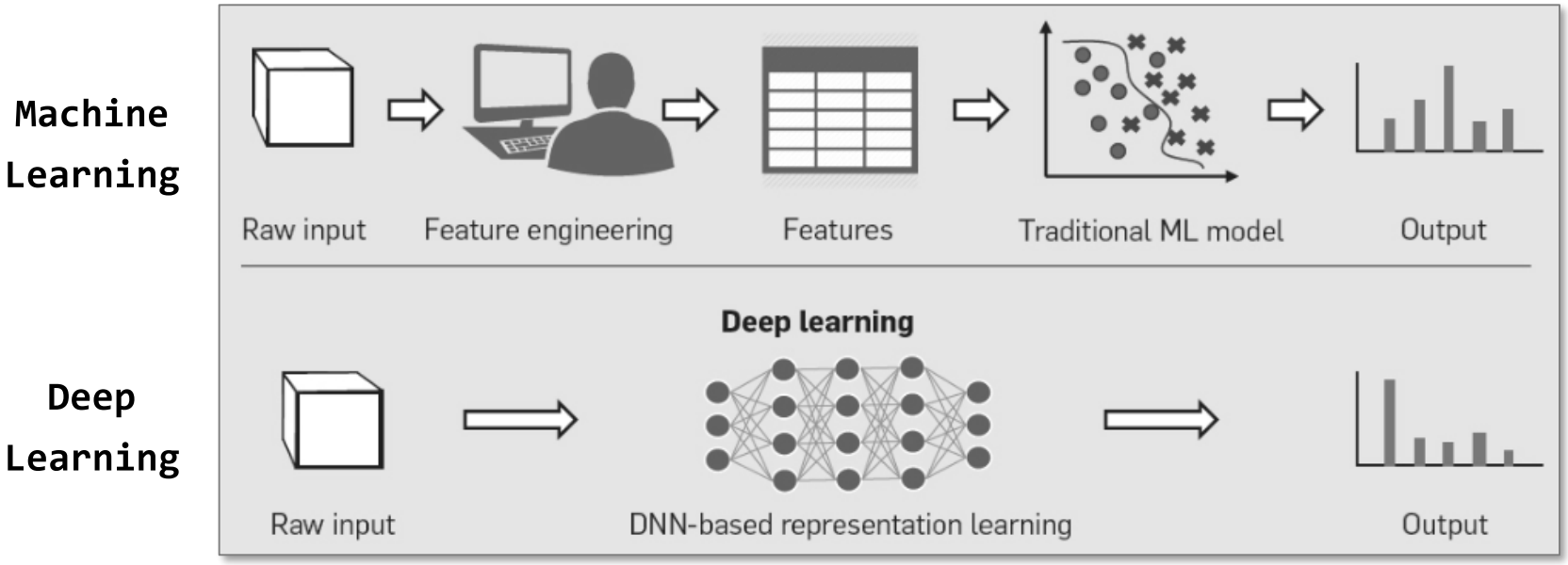
## ✓ 처음으로 돌아와서…Hidden Layer에서는 어떤 일이 일어났나요?

- 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**이 필요합니다.^^)~~
- Hidden Layer에서는 기존 데이터가 **새롭게 표현(Representation)** 되었습니다.
  **Feature Engineering**이 진행된 것입니다!

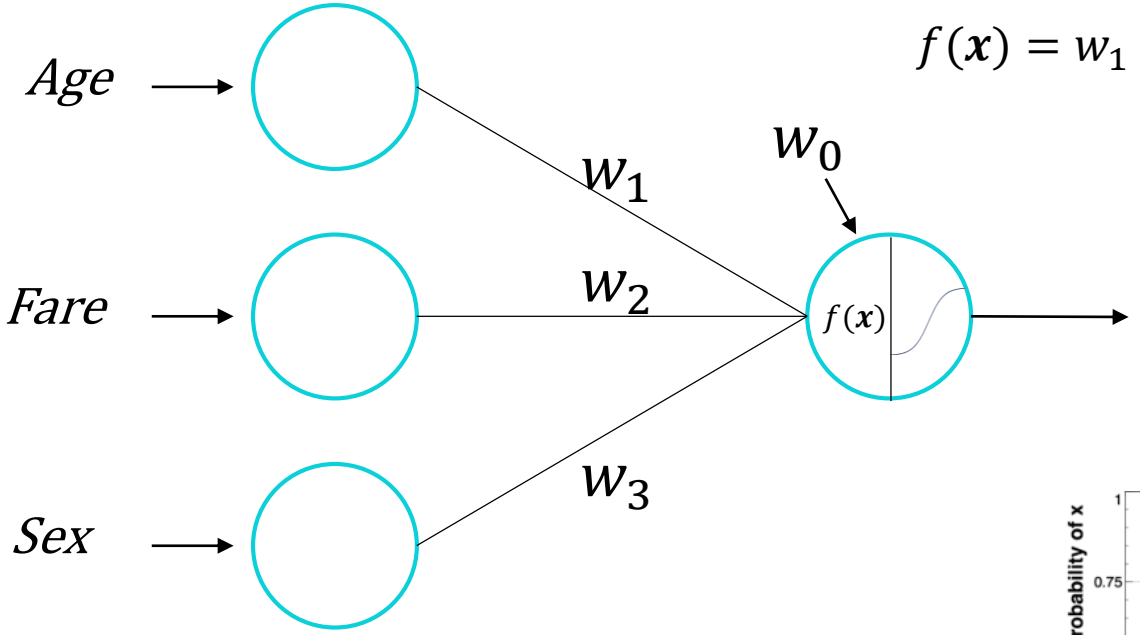


# Feature Representation

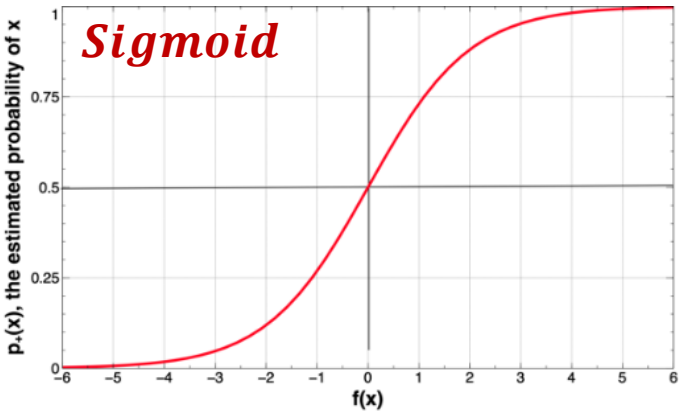
✓**Deep Learning ➔ Representation Learning**



# 딥러닝 구조 - 이진분류




$$f(\boldsymbol{x}) = w_1 \cdot \boldsymbol{Age} + w_2 \cdot \boldsymbol{Fare} + w_3 \cdot \boldsymbol{Sex} + w_0$$




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

✓ **Node의 결과를 변환해주는 함수가 필요**

- 그것을 **활성 함수**Activation Function 라고 합니다.

# 딥러닝 구조 – 활성 함수Activation Function

## ✓node의 결과를 변환시켜 주는 역할

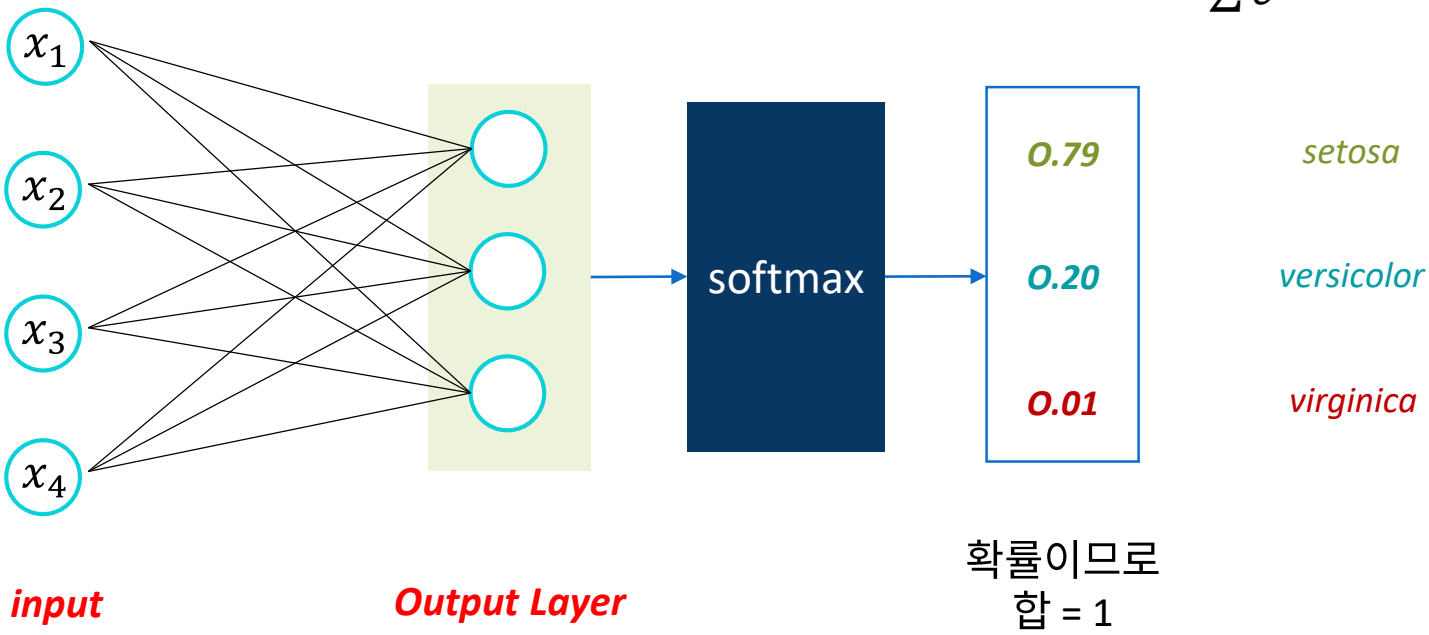
| Layer | Activation Function | | 기능 |
|---|---|---|---|
| Hidden Layer | ReLU | | 좀 더 깊이 있는 학습(Deep Learning)을 시키려고.<br>(Hidden Layer를 여러 층 쌓으려고)<br>(선형 모델을 비선형 모델로 바꾸려고) |
| Output Layer | 회귀 | X | X |
| | 이진분류 | **sigmoid** | 결과를 0, 1로 변환하기 위해 |
| | 다중분류 | **softmax** | 각 범주에 대한 결과를 범주별 확률 값으로 변환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

# 딥러닝 구조 - Output Layer

## ✓Softmax

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

$$softmax(x)_i = \frac{e^{x_i}}{\sum e^{x_i}}, \qquad e = 2.718 \ldots$$




# 요약 : 회귀 vs 이진분류 vs 다중분류

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

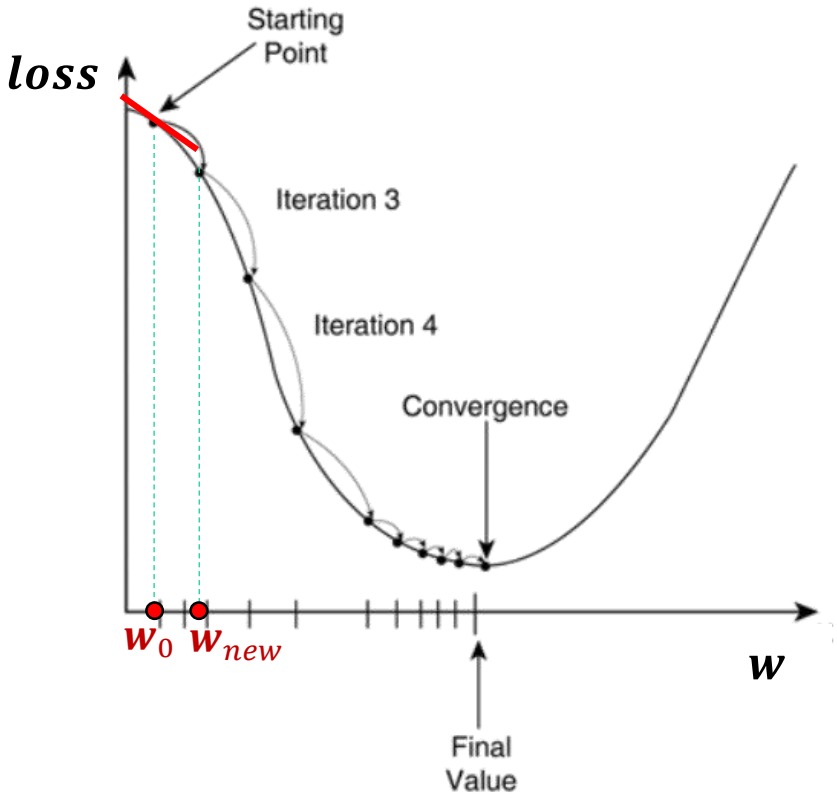
# [참조]가중치 업데이트

✓ **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, $Learning\ Rate$

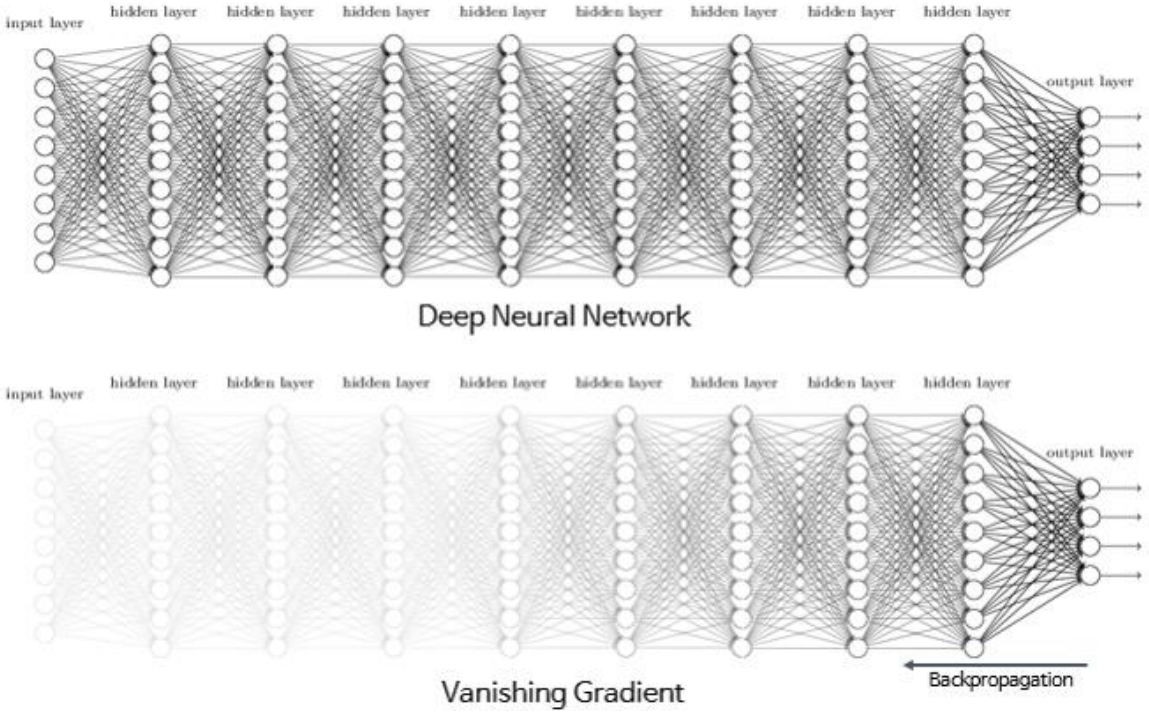
$$w_{new} = w_0 - \eta \times \frac{\partial f(w)}{\partial w}$$

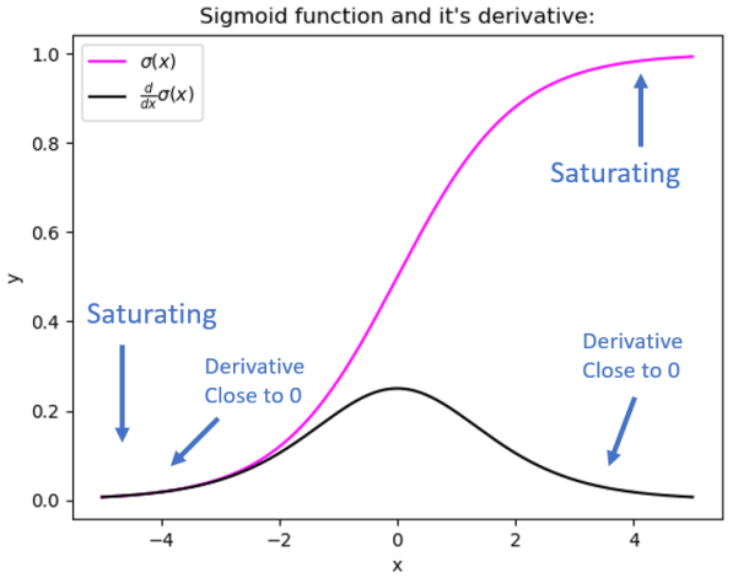


# [참조] Vanishing Gradient(기울기 소실)

## ✓ 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



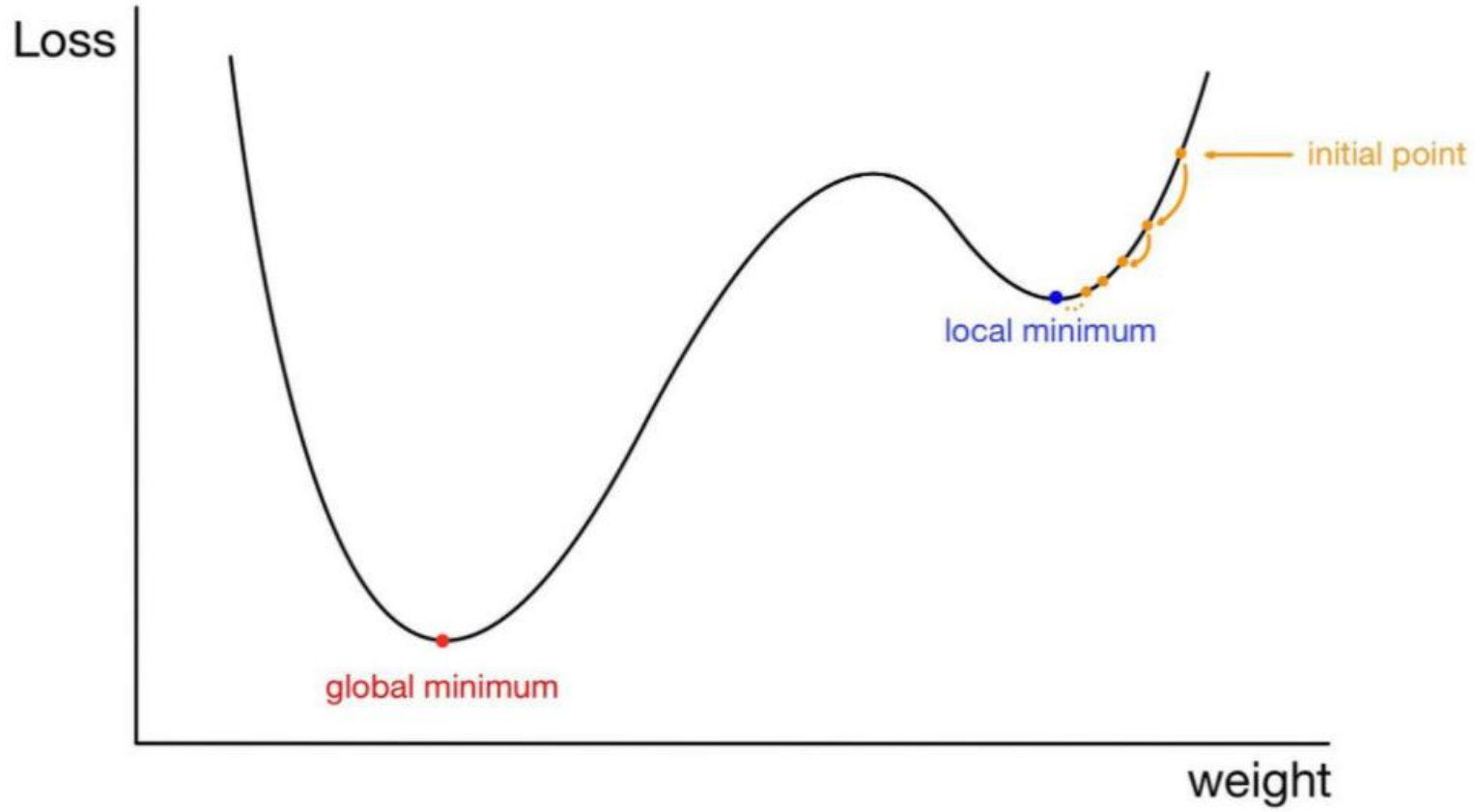
# [참조] Vanishing Gradient(기울기 소실)

## ✓기울기 소실 문제를 최소화 하기 위한 노력

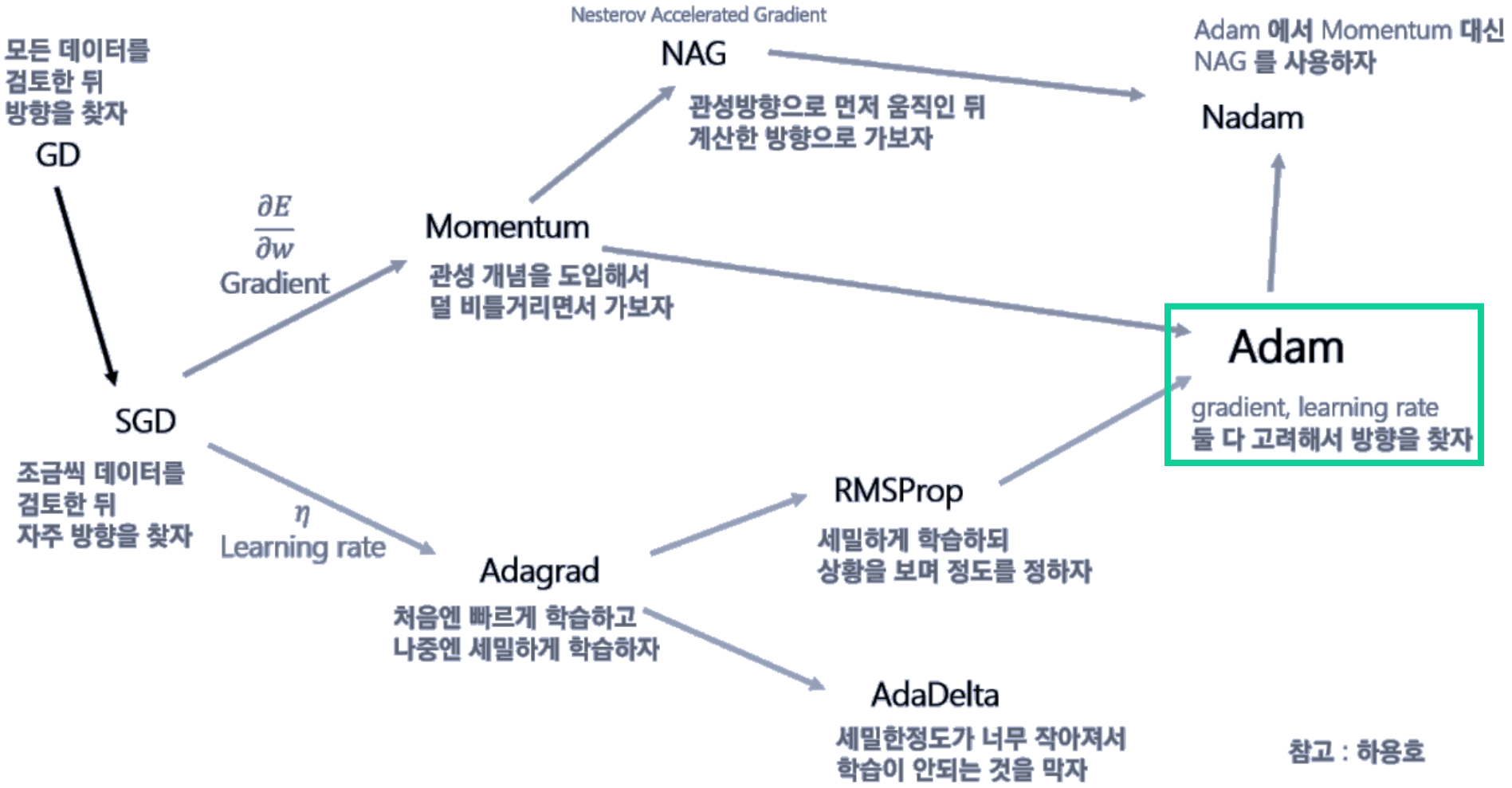
- 활성화 함수 조정
  - ReLU(Rectified Linear Unit)
    - 음수 입력 - 0 출력, 양수 입력 - 그대로 출력 ➔ 양의 기울기 유지하여 기울기 소실 문제를 완화
  - ReLU의 변형된 활성화 함수
    - Leaky ReLU, PReLU, ELU ➔ 음수 입력에 대해서도 매우 작은 기울기를 허용

- 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<br>metric | max_depth<br>min_samples_leaf | n_estimators<br>max_depth<br>learning_rate | Hidden Layer 수, node 수<br>learning_rate, epochs<br>… |

# [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.

AIVLE

KT AIVLE School

# 3일차 정리

AIVLE
Make it possible

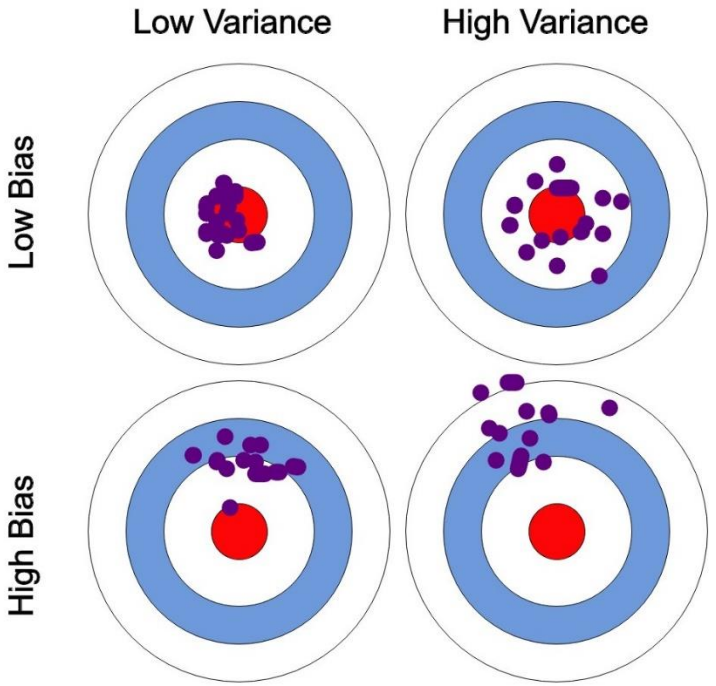

# 요약 : 성능 관리

## ✓딥러닝 모델 성능 높이기

- 데이터
  - 입력 데이터 정제, 적절한 전처리
  - 데이터 늘리기 :
    - 열(적절한 feature 추가) ➔ 성능 향상(Bias 줄이기)
    - 행(데이터 건수 늘리기) ➔ 성능 편차 줄이기(Variance 줄이기)

- 모델 구조
  - 은닉층, 노드 수 늘리기 : 성능이 증가할 때 까지
  - 반복문 / keras-tuner

- 학습
  - Epochs : 10 ~ 50 에서 시작
    - Model check point / early stopping 으로 최적 모델 저장 가능
  - learning_rate : 0.1 ~ 0.001 사이에서 시작




# 요약 : 성능 관리

## ✓과적합 문제

- 모델링 목적 : **모집단 전체**에서 두루 잘 맞추는 (적당한) 모델 만들기
- 과적합 : **학습 데이터**에서만 높은 성능, 다른 데이터에서는 낮은 성능

## ✓과적합 문제 해결

- 데이터 건수 늘리기
- 모델 복잡도 조절하기
  - 반복 학습 횟수(epochs) 적절히 ➔ early stopping
  - 너무 많은 가중치 줄이기 ➔ 가중치 규제(Regularization)

## ✓모델 저장하기

- 최종 모델 저장 : model.save()
- 체크포인트에서 모델 저장 : ModelCheckpoint()
  - 성능이 개선되면 저장하기 가능.

# Function API 코드 연습

## Sequential

- Sequential 함수 안에 리스트로 레이어 입력

```
clear_session()

model = Sequential([
        Dense(18 ,input_shape = (nfeatures, ),
              activation = 'relu' ),
        Dense(4, activation='relu') ,
        Dense(1) ])

model.summary()
```

## Functional

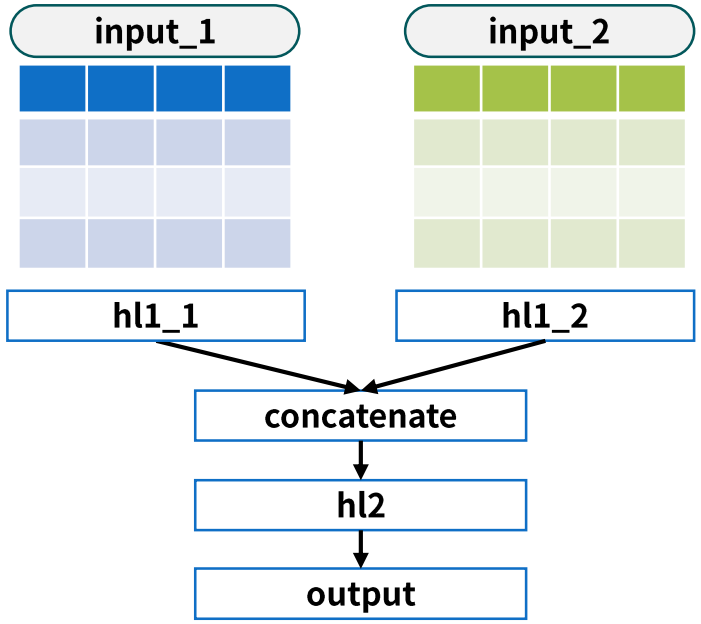
- ① Input 함수
- ② 레이어 : 앞 레이어 연결 지정
- ③ Model 함수로 시작과 끝 연결해서 선언

```
clear_session()

il = Input(shape=(nfeatures, ))
hl1 = Dense(18, activation='relu')(il)
hl2 = Dense(4, activation='relu')(hl1)
ol = Dense(1)(hl2)

model = Model(inputs = il, outputs = ol)

model.summary()
```

# 다중 입력 예제

## ✓ 카시트 판매 데이터를 두가지 입력으로 구분

- 입력1 – 판매 관련 정보 : Advertising, Price, ShelveLoc, US, Urban, CompPrice
- 입력2 – 외부 환경 정보 : Income, Population, Age, Education




```
# 모델 구성
input_1 = Input(shape=(nfeatures1,), name='input_1')
input_2 = Input(shape=(nfeatures2,), name='input_2')

# 입력을 위한 레이어
hl1_1 = Dense(10, activation='relu')(input_1)
hl1_2 = Dense(20, activation='relu')(input_2)

# 두 히든레이어 옆으로 합치기(= pd.concat)
cbl = concatenate([hl1_1, hl1_2])

# 추가레이어
hl2 = Dense(8, activation='relu')(cbl)
output = Dense(1)(hl2)

# 모델 선언
model = Model(inputs = [input_1, input_2], outputs = output)
```