

# 빅데이터 분석기초

## - Machine learning · Deep learning -

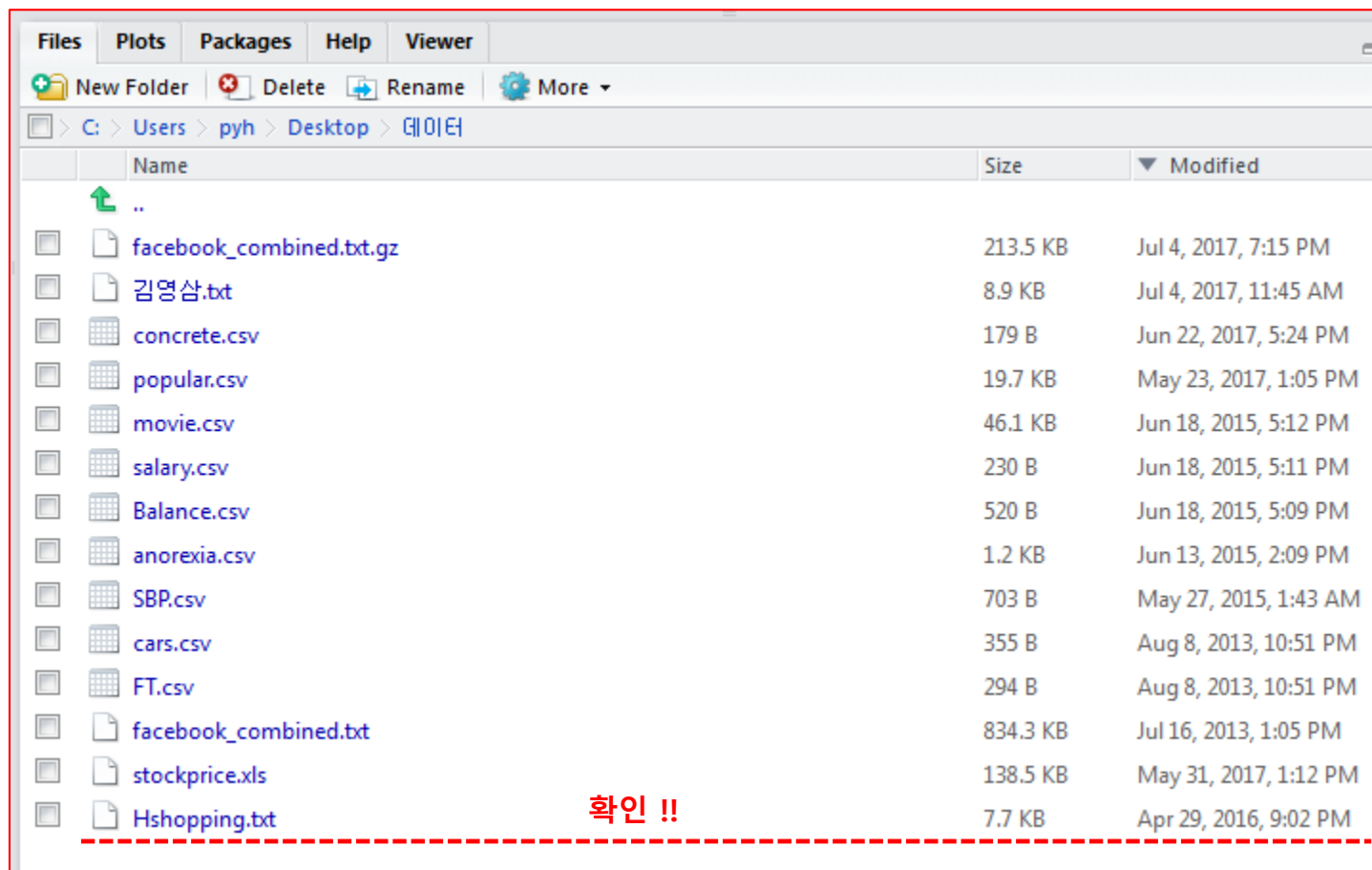
# 2022. 3

부천대학교 토목과 박영훈 교수



# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (고객 반품 예측, "Hshopping.txt" 활용)



# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("nnet")  
library(nnet)  
cb=read.delim("Hshopping.txt", stringsAsFactors = FALSE)  
head(cb,n=20)
```

	ID	성별	나이	구매금액	출연자	반품여부
1	1	1	33	2	2	0
2	2	2	21	3	2	1
3	3	1	45	1	1	0
4	4	1	50	2	1	0
5	5	1	21	3	1	1
6	6	1	22	3	1	1
7	7	2	27	3	2	1
8	8	2	26	3	2	1
9	9	2	28	2	2	1
10	10	2	24	3	2	1
11	11	1	18	2	2	1
12	12	1	64	1	1	0
13	13	1	55	1	1	0
14	14	1	75	1	1	0
15	15	2	73	2	2	0
16	16	2	29	1	2	1
17	17	1	30	3	2	0
18	18	2	31	3	2	1
19	19	1	54	2	2	0
20	20	2	34	3	2	1

# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("nnet")
library(nnet)
cb=read.delim("Hshopping.txt", stringsAsFactors = FALSE)
str(cb)
```

```
'data.frame':   500 obs. of  6 variables:
 $ ID          : int   1 2 3 4 5 6 7 8 9 10 ...
 $ 성별        : int   1 2 1 1 1 1 2 2 2 2 ...
 $ 나이        : int   33 21 45 50 21 22 27 26 28 24 ...
 $ 구매금액    : int   2 3 1 2 3 3 3 3 2 3 ...
 $ 출연자      : int   2 2 1 1 1 1 2 2 2 2 ...
 $ 반품여부    : int   0 1 0 0 1 1 1 1 1 1 ...
```

# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("nnet")
library(nnet)
cb=read.delim("Hshopping.txt", stringsAsFactors = FALSE)
cb$반품여부=factor(cb$반품여부)
install.packages("caret", dependencies = TRUE) → 데이터셋 분리 package
library(caret)
set.seed(1)
inTrain=createDataPartition(y=cb$반품여부, p=0.6, list=FALSE) → 데이터셋에서 60% 분리
cb.train=cb[inTrain,] → 데이터셋에서 60% 분리된 데이터셋을 cb.train으로 할당
cb.test=cb[-inTrain,] → 데이터셋에서 60% 분리된 데이터셋을 제외한 데이터셋을 cb.test으로 할당
```



# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

`str(cb.train)`

```
'data.frame': 301 obs. of 6 variables:
 $ ID      : int  4 6 7 9 10 11 13 15 17 18 ...
 $ 성별    : int  1 1 2 2 2 1 1 2 1 2 ...
 $ 나이    : int  50 22 27 28 24 18 55 73 30 31 ...
 $ 구매금액: int  2 3 3 2 3 2 1 2 3 3 ...
 $ 출연자  : int  1 1 2 2 2 2 1 2 2 2 ...
 $ 반품여부: Factor w/ 2 levels "0","1": 1 2 2 2 2 2 1 1 1 2 ...
```

500명의 60%의 300명이 `cb.train` 데이터셋으로 할당

# 머신러닝 (Machine learning)\_인공신경망

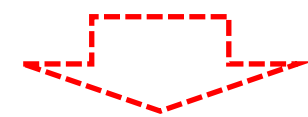
- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("nnet")
library(nnet)
cb=read.delim("Hshopping.txt", stringsAsFactors = FALSE)
cb$반품여부=factor(cb$반품여부)
install.packages("caret", dependencies = TRUE)
library(caret)
set.seed(1)
inTrain=createDataPartition(y=cb$반품여부, p=0.6, list=FALSE)
cb.train=cb[inTrain,]
cb.test=cb[-inTrain,]
set.seed(1234567)
nn_model=nnet(반품여부~성별+나이+구매금액+출연자, data=cb.train, size=3, maxit=1000)
```

- 성별, 나이, 구매금액, 출연자에 따른 반품 여부 학습
- cb.train 데이터셋 적용, 은닉층의 노드수(size)=3, 학습 반복수(maxit)=1000번

```
# weights: 19
initial value 213.055901
iter 10 value 186.774140
iter 20 value 112.128209
iter 30 value 79.153325
iter 40 value 77.474908
iter 50 value 73.522520
iter 60 value 66.366894
iter 70 value 64.676339
iter 80 value 64.347763
iter 90 value 64.280156
iter 100 value 64.121042
iter 110 value 64.038900
iter 120 value 64.021142
iter 130 value 63.930980
iter 140 value 63.221849
iter 150 value 61.809592
iter 160 value 60.229743
iter 170 value 60.007861
iter 180 value 59.961155
iter 190 value 59.958479
iter 200 value 59.954248
final value 59.954171
converged
```



# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
```

```
cb.test$nn_pred_prob=predict(nn_model, cb.test, type="raw")
```

```
head(cb.test, n=30)
```

	ID	성별	나이	구매금액	출연자	반품여부	nn_pred	nn_pred_prob
1	1	1	33	2	2	0	0	0.3196240848
2	2	2	21	3	2	1	1	0.9930982462
3	3	1	45	1	1	0	0	0.0091319806
5	5	1	21	3	1	1	1	0.7571775278
8	8	2	26	3	2	1	1	0.9776174199
12	12	1	64	1	1	0	0	0.0010564718
14	14	1	75	1	1	0	0	0.0003966508
16	16	2	29	1	2	1	1	0.6909776516
19	19	1	54	2	2	0	0	0.0154878711
22	22	2	34	3	2	1	1	0.8787121431
23	23	1	54	1	1	0	0	0.0030397389
24	24	1	53	2	1	0	0	0.0064706847
26	26	1	64	2	1	0	0	0.0018126424
27	27	2	25	3	2	1	1	0.9822090792
28	28	2	28	3	2	1	1	0.9649281742
29	29	2	64	2	2	0	0	0.0141665331
34	34	1	54	1	2	0	0	0.0076638167
36	36	2	25	1	2	0	1	0.8368975875
37	37	2	67	1	1	0	0	0.0020263346
39	39	2	35	3	2	1	1	0.8539145265
40	40	1	37	3	2	1	0	0.3747105281
42	42	1	39	3	2	1	0	0.2925032834
46	46	1	58	1	1	0	0	0.0019519579
47	47	1	59	2	1	0	0	0.0031481504
49	49	1	67	2	1	0	0	0.0013283004
51	51	2	57	1	2	0	0	0.0178837817
53	53	2	17	2	2	1	1	0.9905537022
55	55	1	29	2	2	0	0	0.5005600616
56	56	2	21	2	2	1	1	0.9760030643
59	59	2	59	1	2	0	0	0.0135894631

실제

예측





# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

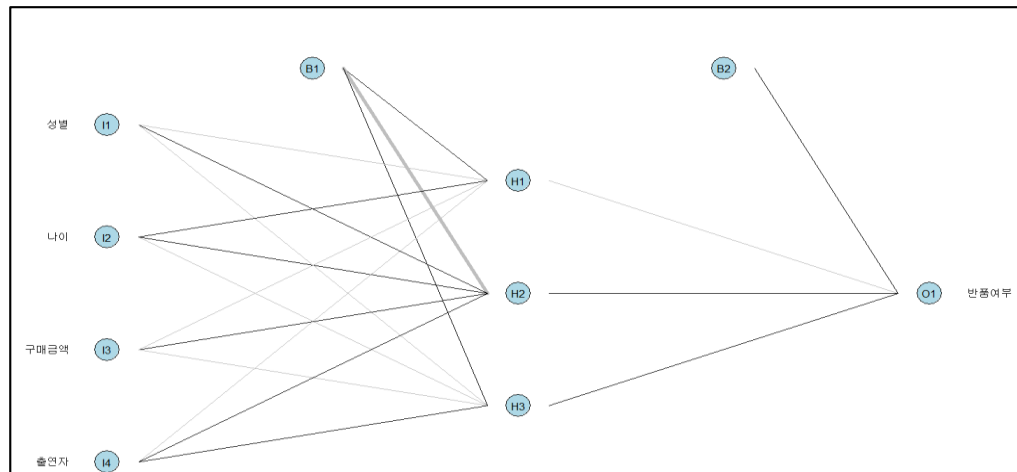
## 입력 및 출력

```
install.packages("devtools")
```

```
library(devtools)
```

```
source_url('https://gist.githubusercontent.com/Peque/41a9e20d6687f2f3108d/raw/85e14f3a292e126f1454864427e3a189c2fe33f3/nnet_plot_update.r')
```

```
plot.nnet(nn_model)
```



# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

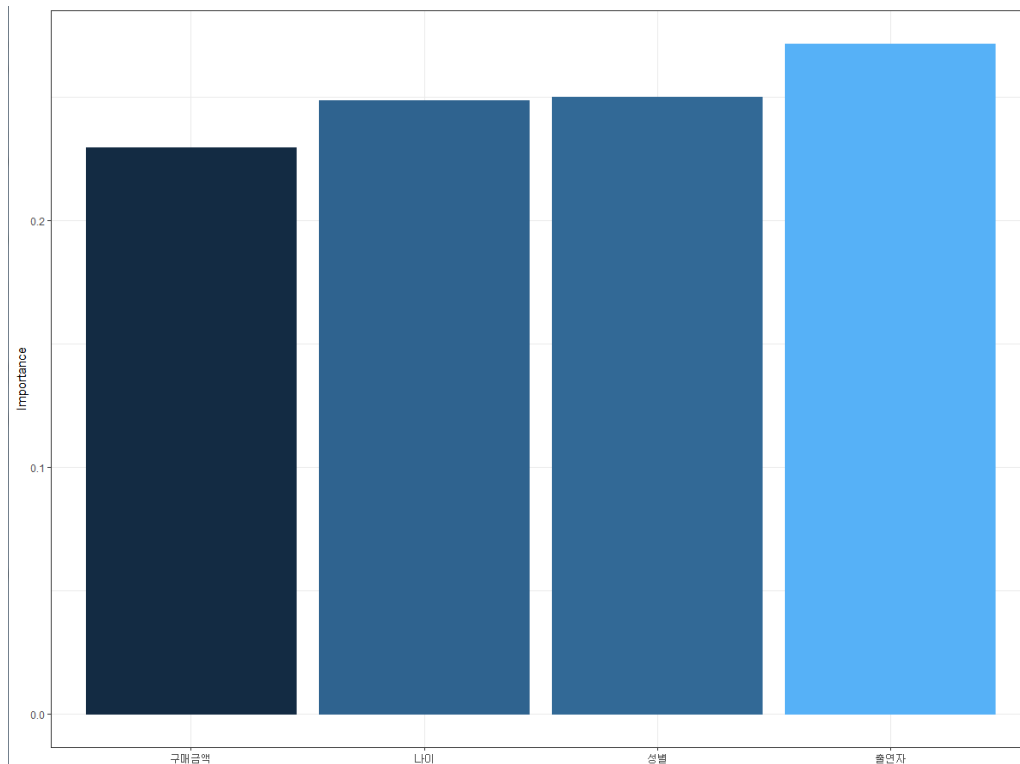
## 입력 및 출력

```
install.packages("NeuralNetTools")
```

```
library(NeuralNetTools)
```

```
garson(nn_model)
```

변수 중요도



- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

Prediction \ Reference	0	1
	0 123 9	1 14 53

Accuracy : 0.8844  
95% CI : (0.8316, 0.9  
No Information Rate : 0.6884  
P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364  
McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978  
Specificity : 0.8548  
Pos Pred Value : 0.9318  
Neg Pred Value : 0.7910  
Prevalence : 0.6884  
Detection Rate : 0.6181  
Detection Prevalence : 0.6633  
Balanced Accuracy : 0.8763

'Positive' class : 0

- 실제 반품하지 않았고 인공지능망에 의한 예측에서도 반품하지 않을 고객 123명
- 실제 반품하지 않았는데 인공지능망에 의한 예측에서 반품할 고객은 14명 (오류)
- 실제 반품했으나 인공지능망에 의한 예측에서 반품하지 않을 고객은 9명 (오류)
- 실제 반품했으며 인공지능망에 의한 예측에서도 반품할 고객 53명

# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

	Reference	
Prediction	0	1
0	123	9
1	14	53

Accuracy : 0.8844

95% CI : (0.8316, 0.9253)

No Information Rate : 0.6884

P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364

McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978

Specificity : 0.8548

Pos Pred Value : 0.9318

Neg Pred Value : 0.7910

Prevalence : 0.6884

Detection Rate : 0.6181

Detection Prevalence : 0.6633

Balanced Accuracy : 0.8763

'Positive' Class : 0

$$(123+53)/(123+9+14+53) = 0.8844(88.44\%)$$

# 머신러닝 (Machine learning)\_인공신경망

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

	Reference	
Prediction	0	1
0	123	9
1	14	53

Accuracy : 0.8844

95% CI : (0.8316, 0.9253)

No Information Rate : 0.6884

P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364

McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978

Specificity : 0.8548

Pos Pred Value : 0.9318

Neg Pred Value : 0.7910

Prevalence : 0.6884

Detection Rate : 0.6181

Detection Prevalence : 0.6633

Balanced Accuracy : 0.8763

'Positive' Class : 0

95% 신뢰구간(confidence interval)에서 정확도(accuracy)  
0.8316~0.9253(83.16%~92.53%)

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

	Reference	
Prediction	0	1
0	123	9
1	14	53

Accuracy : 0.8844  
95% CI : (0.8316, 0.9253)  
No Information Rate : 0.6884  
P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364  
McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978

Specificity : 0.8548

Pos Pred Value : 0.9318

Neg Pred Value : 0.7910

Prevalence : 0.6884

Detection Rate : 0.6181

Detection Prevalence : 0.6633

Balanced Accuracy : 0.8763

'Positive' Class : 0

민감도 : 실제 true(0)인 고객이 예측에서도 true(0)인 비율  
(123)/(123+14)=0.8978(89.78%)

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

		Reference	
Prediction		0	1
0	123	9	
1	14		53

Accuracy : 0.8844  
95% CI : (0.8316, 0.9253)  
No Information Rate : 0.6884  
P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364  
McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978  
Specificity : 0.8548  
Pos Pred Value : 0.9318  
Neg Pred Value : 0.7910  
Prevalence : 0.6884  
Detection Rate : 0.6181  
Detection Prevalence : 0.6633  
Balanced Accuracy : 0.8763

'Positive' Class : 0

특이도 : 실제 false(1)인 고객이 예측에서도 false(1)인 비율  
 $(53)/(53+9)=0.8548(85.48\%)$

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=as.factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

	Reference	
Prediction	0	1
0	123	9
1	14	53

Accuracy : 0.8844  
95% CI : (0.8316, 0.9253)  
No Information Rate : 0.6884  
P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364  
McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978  
Specificity : 0.8548  
Pos Pred Value : 0.9318  
Neg Pred Value : 0.7910  
Prevalence : 0.6884  
Detection Rate : 0.6181  
Detection Prevalence : 0.6633  
Balanced Accuracy : 0.8763

'Positive' Class : 0

예측에서 true(0)인 고객이 실제에서도 true(0)인 비율  
(123)/(123+9)=0.9318(93.18%)



- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

```
install.packages("caret",dependencies = TRUE) : library(caret)
cb.test$nn_pred=predict(nn_model, cb.test, type="class")
cb.test$nn_pred=as.factor(cb.test$nn_pred)
predicted=as.factor(predict(nn_model, newdata=cb.test, type="class"))
confusionMatrix(predicted,cb.test$반품여부)
```

### Confusion Matrix and Statistics

	Reference	
Prediction	0	1
0	123	9
1	14	53

Accuracy : 0.8844  
95% CI : (0.8316, 0.9253)  
No Information Rate : 0.6884  
P-Value [Acc > NIR] : 7.026e-11

Kappa : 0.7364  
McNemar's Test P-Value : 0.4042

Sensitivity : 0.8978  
Specificity : 0.8548  
Pos Pred Value : 0.9318  
Neg Pred Value : 0.7910  
Prevalence : 0.6884  
Detection Rate : 0.6181  
Detection Prevalence : 0.6633  
Balanced Accuracy : 0.8763

'Positive' Class : 0

예측에서 false(1)인 고객이 실제에서도 false(1)인 비율  
 $(53)/(14+53)=0.7910(79.10\%)$

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

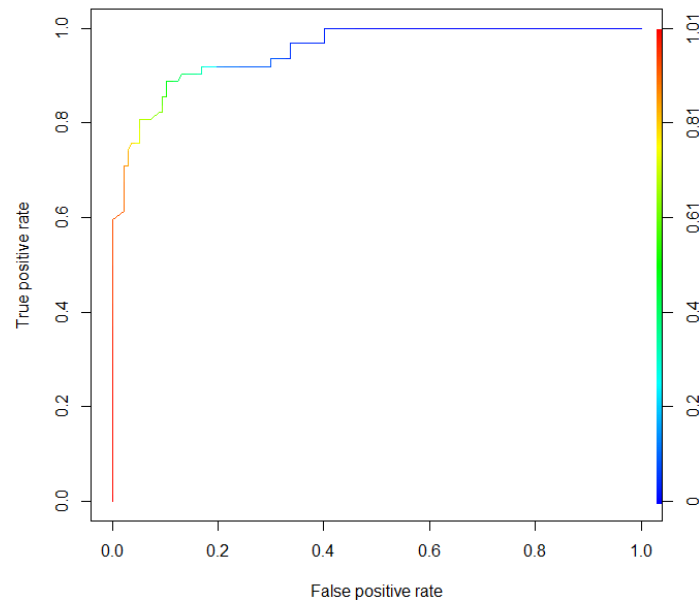
```
install.packages("ROCR")
```

```
library(ROCR)
```

```
nn_pred=prediction(predict(nn_model, newdata=cb.test, type="raw"), cb.test$반품여부)
```

```
nn_model.perf1=performance(nn_pred, "tpr", "fpr")
```

```
plot(nn_model.perf1, colorize=TRUE)
```



- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

인공 신경망을 활용하여 고객의 '성별'과 '나이'만 고려한 예측 모델을 만들고 정확도를 평가하여라  
Nn\_model=nnet(반품여부~성별+나이, data=cb.train, size=3, maxit=1000)

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

인공 신경망을 활용하여 고객의 '성별'과 '나이'만 고려한 예측 모델을 만들고 정확도를 평가하여라  
Nn\_model=nnet(반품여부~성별+나이, data=cb.train, size=3, maxit=1000)

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

인공 신경망을 활용하여 고객의 '성별'과 '나이'만 고려한 예측 모델을 만들고 정확도를 평가하여라  
Nn\_model=nnet(반품여부~성별+나이, data=cb.train, size=3, maxit=1000)

- 인공 신경망 적용 (package "nnet", 고객 반품 예측, "Hshopping.txt" 활용)

## 입력 및 출력

인공 신경망을 활용하여 고객의 '성별'과 '나이'만 고려한 예측 모델을 만들고 정확도를 평가하여라  
Nn\_model=nnet(반품여부~성별+나이, data=cb.train, size=3, maxit=1000)

## \* 머신러닝\_의사결정나무 (Desion Tree)

```
install.packages("caret")
install.packages("C50")
install.packages("ROCR")
install.packages('e1071', dependencies=TRUE)
library(caret)
library(C50)
library(ROCR)
library(e1071)
cb=read.delim("Hshopping.txt", stringsAsFactors=FALSE)
head(cb)
str(cb)
cb$반품여부=factor(cb$반품여부)

set.seed(1)
inTrain=createDataPartition(y=cb$반품여부, p=0.6, list=FALSE)
cb.train=cb[inTrain,]
cb.test=cb[-inTrain,]

dim(cb.train); dim(cb.test)
```

## \* 머신러닝\_ C5.0 (Desion Tree)

```
c5_options =C5.0Control(winnow = FALSE, noGlobalPruning= FALSE)
c5_model =C5.0(반품여부~ 성별+나이+구매금액+출연자, data=cb.train, control=c5_options,
rules=FALSE)
summary(c5_model)
plot(c5_model)
cb.test$c5_pred =predict(c5_model, cb.test, type="class")

cb.test$c5_pred_prob =predict(c5_model, cb.test, type="prob")

head(cb.test)

confusionMatrix(cb.test$c5_pred, cb.test$반품여부)

c5_pred =prediction(cb.test$c5_pred_prob[,2],cb.test$반품여부)

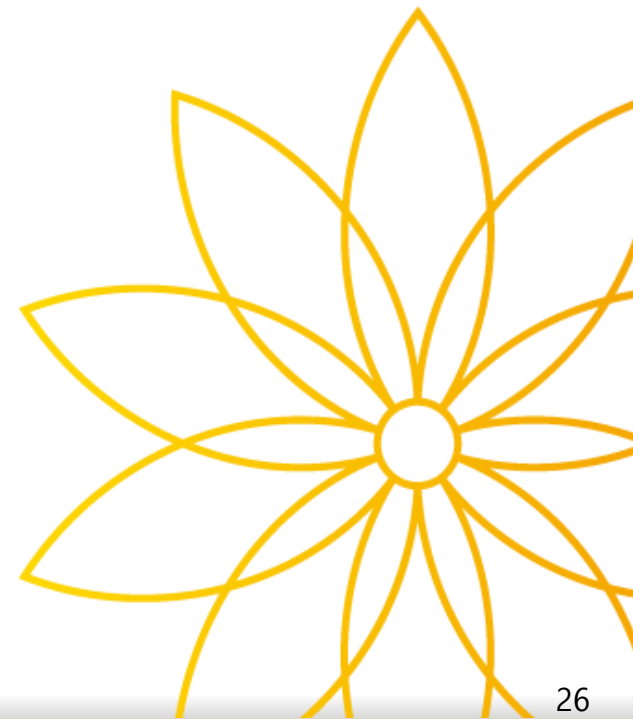
c5_model.perf1 =performance(c5_pred, "tpr", "fpr") # ROC-chart
c5_model.perf2 =performance(c5_pred, "lift", "rpp") # Lift chart
plot(c5_model.perf1, colorize=TRUE)
plot(c5_model.perf2, colorize=TRUE)
```



## \* 머신러닝\_ Randomforest

```
install.packages("randomForest")  
library(randomForest)  
rf_model=randomForest(반품여부~성별+나이+구매금액+출연자, data=cb.train, ntree=10)  
rf_model  
plot(rf_model)  
varImpPlot(rf_model)  
cb.test$rf_pred=predict(rf_model, cb.test, type="response")  
cb.test$rf_pred_prob=predict(rf_model, cb.test, type="prob")  
confusionMatrix(cb.test$rf_pred, cb.test$반품여부)  
rf_pred=prediction(cb.test$rf_pred_prob[,2],cb.test$반품여부)  
rf_model.perf1 =performance(rf_pred, "tpr", "fpr") # ROC-chart  
plot(rf_model.perf1, colorize=TRUE)  
performance(rf_pred, "auc")@y.values[[1]]
```

## IV. 인공지능 (Artificial Intelligence)\_딥러닝 (Deep learning)



# 딥러닝 (Deep learning)

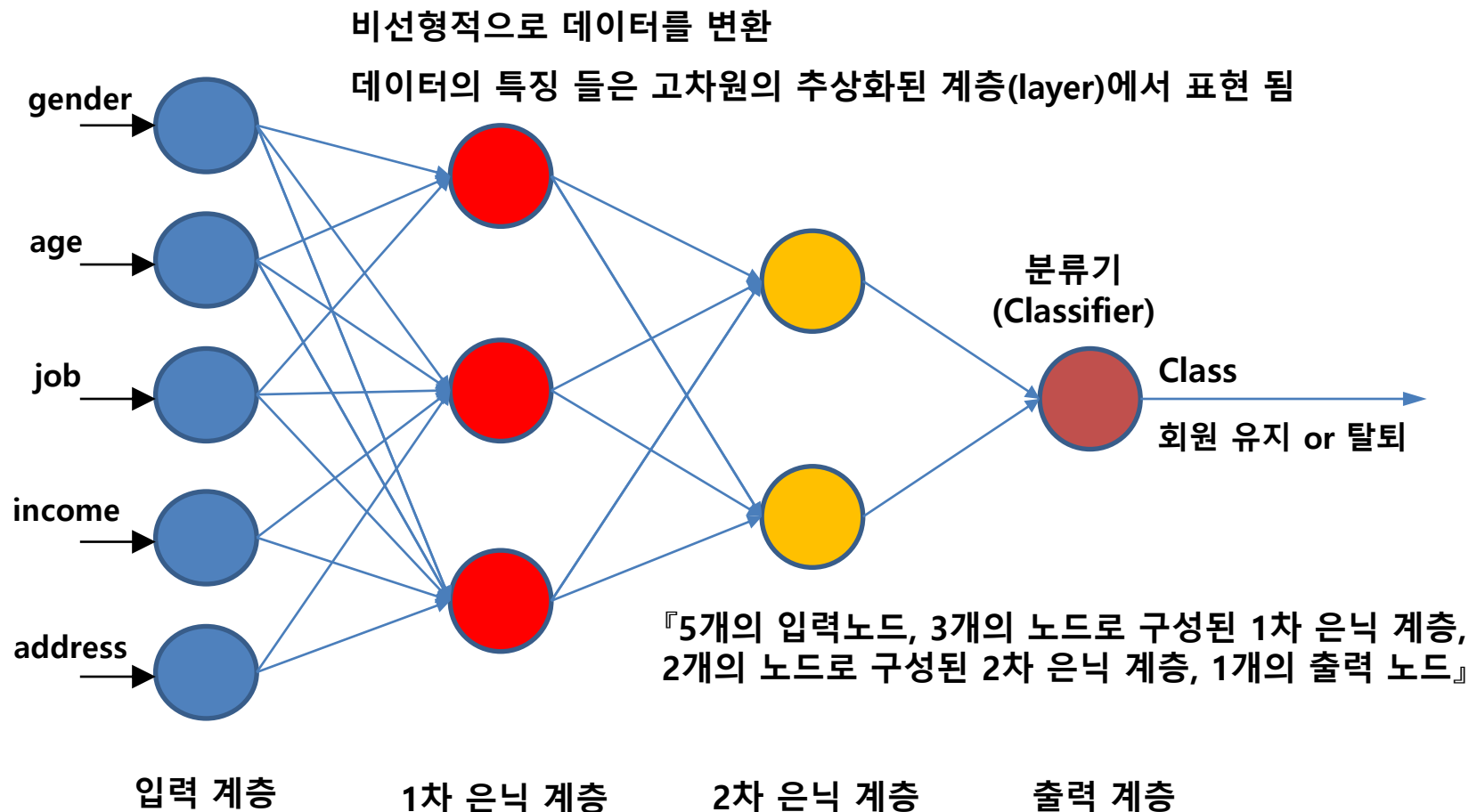
- 머신러닝의 한 분야 (딥러닝 = 다층 머신러닝, 지도 및 비지도 학습)
- 빠른 CPU와 저렴한 메모리
- 데이터의 특징 들은 고차원의 추상화된 계층(layer)에서 표현 됨  
(계층(layer) : 비선형적으로 데이터를 변환하는 여러 단계)

\* 지도학습 (Supervised learning) : 학습 데이터(training data)에 결과가 포함 돼 있는 경우

\* 비지도학습 (Unsupervised learning) : 학습 데이터에 결과가 포함 되어 있지 않음

데이터 내 관계를 스스로 발견하는 알고리즘을 의미

# 딥러닝 (Deep learning)



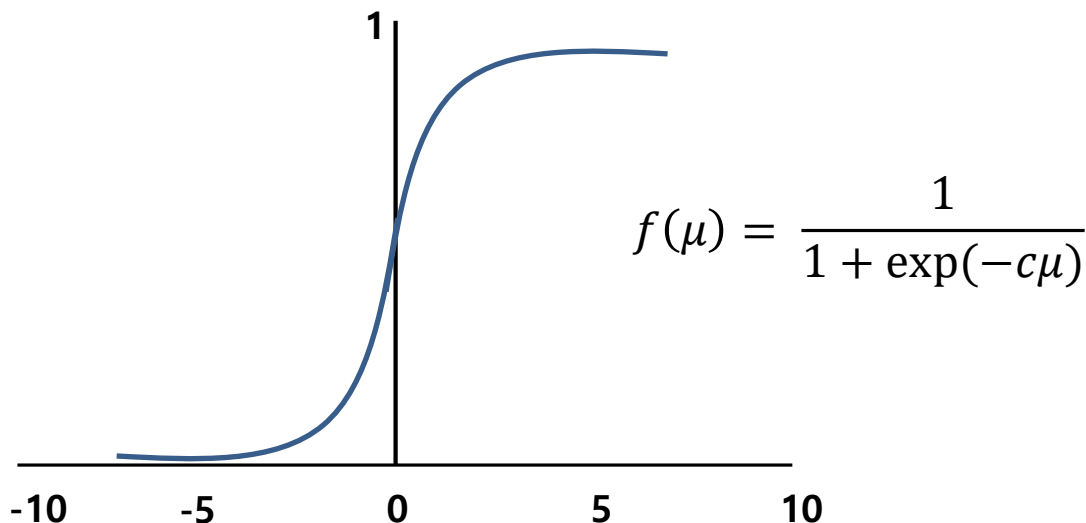
[2 개의 은닉 계층이 있는 피드포워드(Feed Forward) 신경망]

# 딥러닝 (Deep learning)

- 활성화 함수(Activation function) : 네트워크에 비선형성을 적용하는데 필요

- 활성화 함수 적용 결과는 네트워크 내 다음 뉴런으로 전달
- 뉴런의 출력은 일반적으로 0~1 또는 -1 ~ 1 사이의 값을 가짐
- 학습과정에서 계산을 많이 줄 일 수 있음

- 시그모이드 함수(sigmoid function)



# 딥러닝 (Deep learning)

- 활성화 함수(Activation function) : 네트워크에 비선형성을 적용하는데 필요

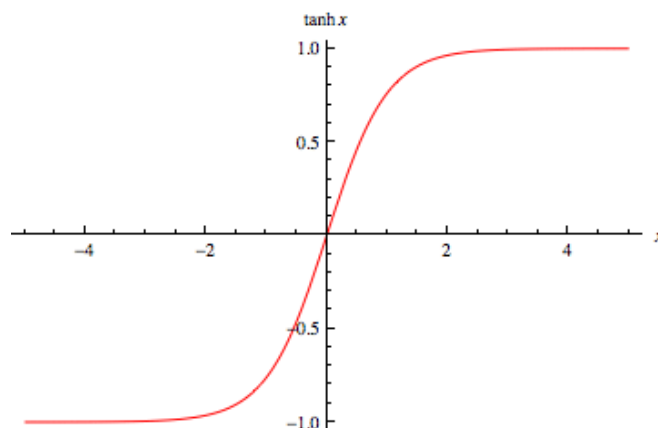
- 선형 함수(linear function)

$$f(\mu) = \mu$$

- 초평면 탄젠트 함수(Hyperbolic tangent function)

$$f(\mu) = \tanh(\mu) \quad -1 \sim 1 \text{ 출력, 시그모이드 함수의 많은 특성 공유}$$

시그모이드 함수 보다 복잡한 비선형 관계를 모델링하는데 효율적

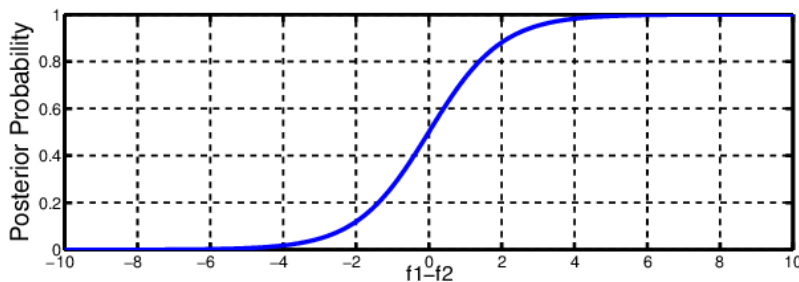


# 딥러닝 (Deep learning)

- 활성화 함수(Activation function) : 네트워크에 비선형성을 적용하는데 필요

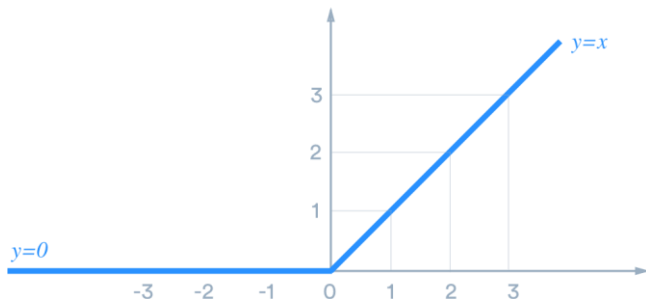
- 소프트맥스 함수(Softmax function)

$$f(\mu) = \frac{\exp\left(\frac{\mu}{T}\right)}{\sum_k \exp\left(\frac{\mu}{T}\right)}$$



- 정류된 선형 유닛 (ReLU Rectified linear unit)

$$f(\mu) = \max(0, \mu)$$



- 이미지 인식에 많이 활용
- 뉴런의 결과가 양수인 경우만 활성화
- 최대값 계산으로 초평면 탄젠트 활성화 함수나 시그모이드 함수를 사용하는 신경망보다 더 빠른 계산 지원
- 대략 절반의 뉴런을 0으로 설정 → 신경망 희소성(sparsity) 허용

# 딥러닝 (Deep learning)

## · 신경망 학습

### 네트워크 초기화

- 무작위로 초기화



### 피드 포워드

- 네트워크를 통해 입력계층부터, 은닉 계층, 출력 계층 까지 활성화 함수와 가중치 적용을 거쳐 정보가 전달
- 활성화 함수로는 노드 입력의 가중치 합에 대한 시그모이드 함수가 사용



### 오차 평가

- 네트워크에서 계산된 예측 결과를 실제 결과와 비교
- 둘 사이의 오차가 이미 정해진 가중치보다 작으면 알고리즘 통과



### 전파

- 출력 계층의 오차는 가중치를 다시 조정하는데 사용
- 알고리즘은 오차를 네트워크에서 거꾸로 전파
- 가중치 변경과 관련해서 오차 값 변화의 경사(gradient)를 계산



### 조정

- 오차를 줄이는 방향으로 경사의 변화를 사용해서 가중치 조정
- 각 뉴런의 가중치와 편향은 활성화 함수의 도함수, 예측값과 실제값 차이, 뉴런 결과 등에 의해 조정

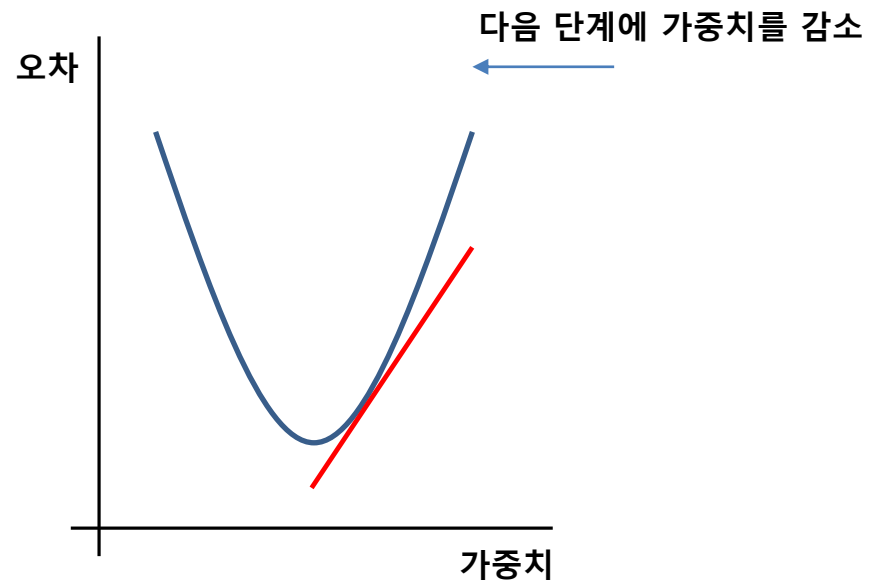
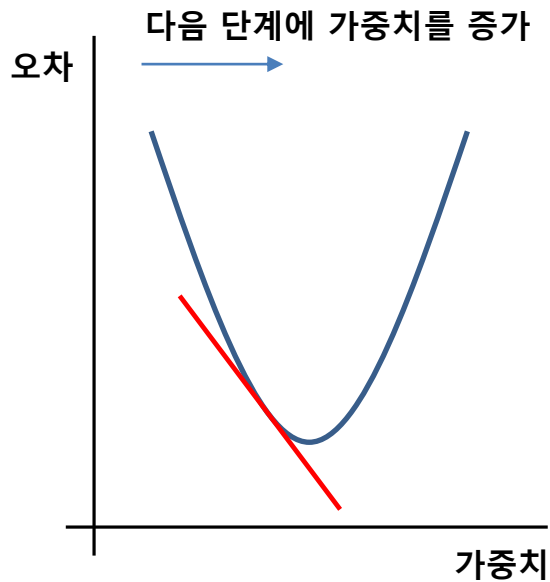


# 딥러닝 (Deep learning)

## · 신경망 학습

### - 경사 (gradient)

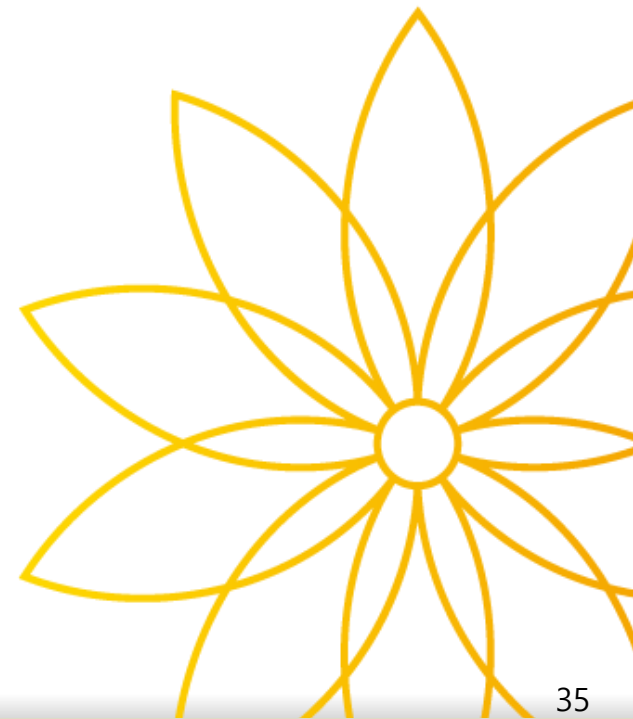
편미분이 음이면 가중치 증가, 편미분이 양이면 가중치 감소



# 딥러닝 (Deep learning) (심층신경망(DNN))

- 객체 인식, 장면 이해, 의료 등 다양한 응용 분야에서 좋은 성능을 보여 줌
- 입력 변수의 값을 조합하고 가중치를 부여해 새로운 실수를 만들어 출력 계층으로 전달
- 출력 계층에서 분류나, 예측을 수행
- DNN 학습이 이루어 지는 동안 학습 단계에서 계산되는 예측 결과가 실제 결과와 가능한 같도록 가중치 조정 됨
- 개별적인 회귀 모델들의 조합으로 볼 수 있다

## IV.1 DEEP LEARNING, 'R'



# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")  
library(neuralnet)  
set.seed(2021)  
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)  
attribute
```

```
> attribute  
      sample(seq(-2, 2, length = 50), 50, replace = FALSE)  
1                -1.26530612  
2                -1.42857143  
3                 1.26530612  
4                -1.51020408  
5                -0.28571429  
6                -1.59183673  
7                 0.20408163  
8                 1.10204082  
9                -2.00000000  
10               -1.83673469  
11               -0.77551020  
12               -1.18367347  
13               -1.34693878  
14                0.61224490  
15                1.02040816  
16               -0.93877551  
17                1.67346030
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")  
library(neuralnet)  
set.seed(2021)  
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)  
attribute  
response=attribute^2  
respons
```

```
> response  
      sample(seq(-2, 2, length = 50), 50, replace = FALSE)  
1                1.600999584  
2                2.040816327  
3                1.600999584  
4                2.280716368  
5                0.081632653  
6                2.533944190  
7                0.041649313  
8                1.214493961  
9                4.000000000  
10               3.373594336  
11               0.601416077  
12               1.401082882  
13               1.814244065  
14               0.374843815  
15               1.041232820  
16               0.881299459
```

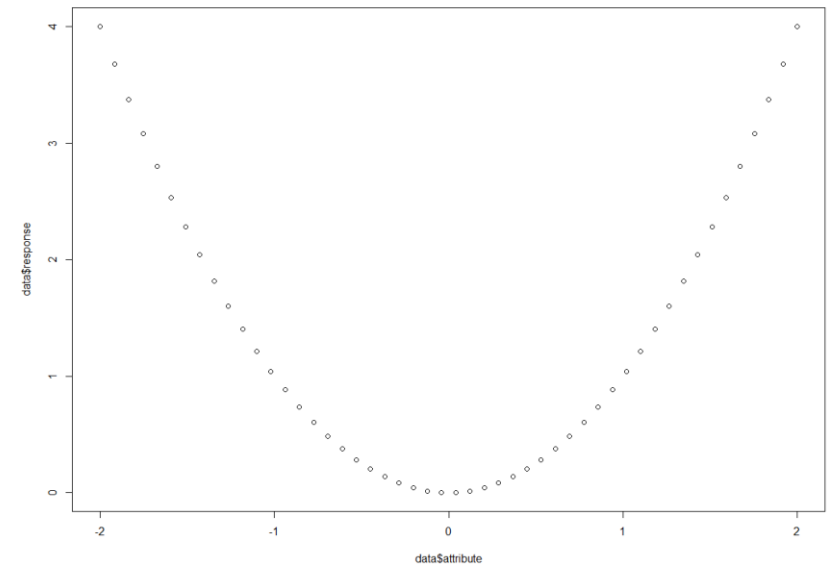
# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")
library(neuralnet)
set.seed(2021)
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)
attribute
response=attribute^2
response
data=cbind(attribute, response)
colnames(data)=c("attribute", "response")
data
```

```
> data
      attribute  response
1  -1.26530612  1.600999584
2  -1.42857143  2.040816327
3   1.26530612  1.600999584
4  -1.51020408  2.280716368
5  -0.28571429  0.081632653
6  -1.59183673  2.533944190
7   0.20408163  0.041649313
8   1.10204082  1.214493961
9  -2.00000000  4.000000000
10 -1.83673469  3.373594336
11 -0.77551020  0.601416077
12 -1.18367347  1.401082882
13 -1.34693878  1.814244065
14  0.61224490  0.374843815
```

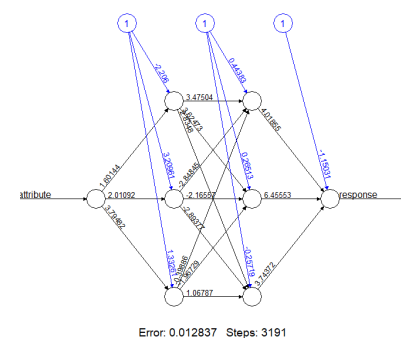
# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")  
library(neuralnet)  
set.seed(2021)  
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)  
attribute  
response=attribute^2  
response  
data=cbind(attribute, response)  
colnames(data)=c("attribute", "response")  
data  
plot(data$attribute,data$response )
```



# 딥러닝 (Deep learning) (심층신경망(DNN))

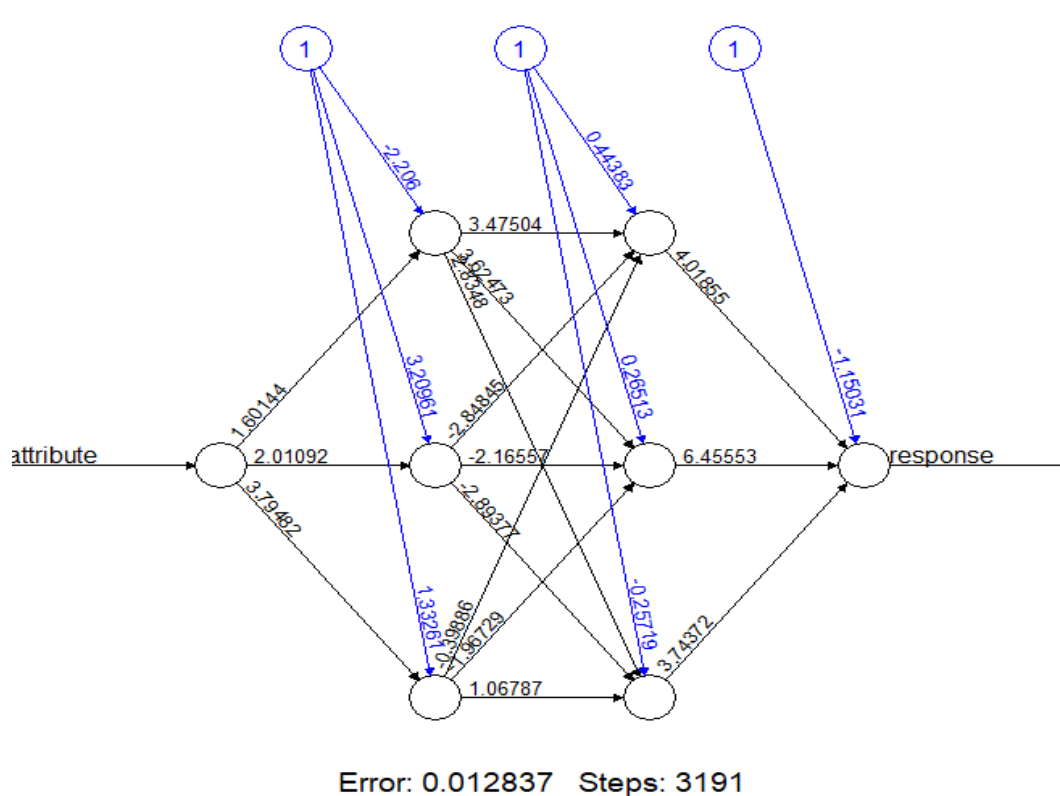
```
install.packages("neuralnet")
library(neuralnet)
set.seed(2021)
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)
attribute
response=attribute^2
respons
data=cbind(attribute, response)
colnames(data)=c("attribute", "response")
data
plot(data$attribute,data$response )
fit=neuralnet(response~attribute, data=data, hidden = c(3,3), threshold=0.01)
plot(fit)
```





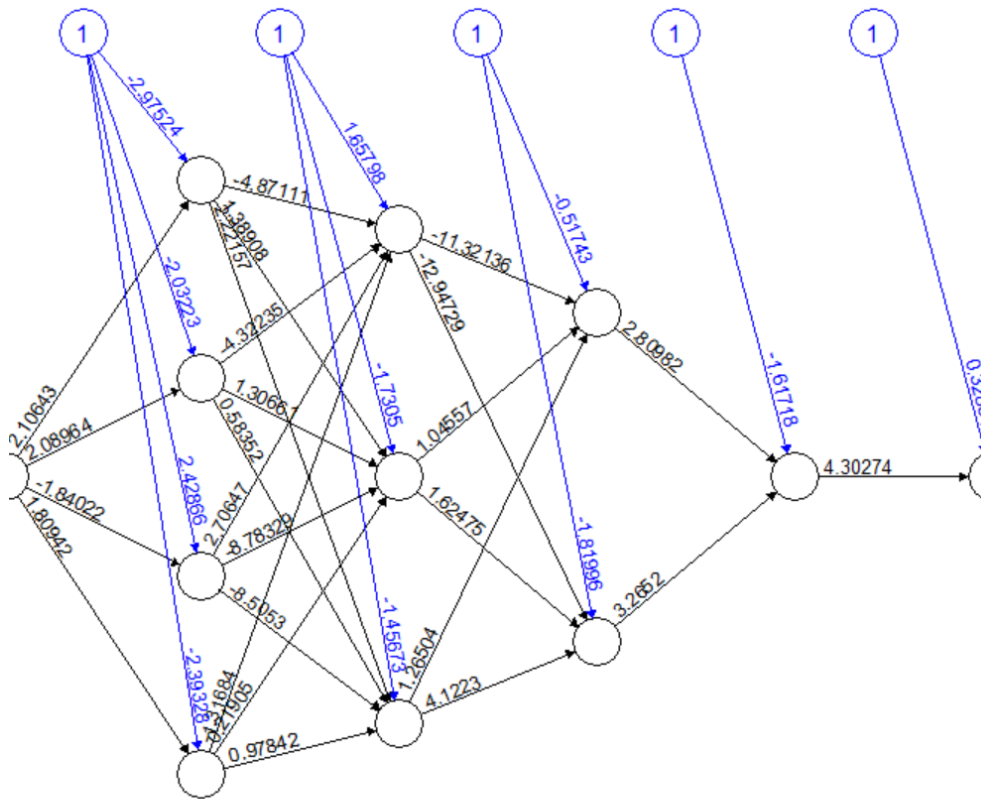
# 딥러닝 (Deep learning) (심층신경망(DNN))

```
fit=neuralnet(response~attribute, data=data, hidden = c(3,3), threshold=0.01)  
plot(fit)
```



# 딥러닝 (Deep learning) (심층신경망(DNN))

```
fit3=neuralnet(response~attribute, data=data, hidden = c(4,3,2,1), threshold=0.01)
plot(fit3)
```



Error: 24.362241 Steps: 370

# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")
library(neuralnet)
set.seed(2021)
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)
attribute
response=attribute^2
response
data=cbind(attribute, response)
colnames(data)=c("attribute", "response")
data
plot(data$attribute,data$response )
fit=neuralnet(response~attribute, data=data, hidden = c(3,3), threshold=0.01)
plot(fit)
testdata=as.matrix(sample(seq(-2,2,length=10), 10, replace=FALSE), ncol=1)
testdata
```

```
> testdata
      [,1]
[1,] 0.2222222
[2,] -1.5555556
[3,] 1.1111111
[4,] 0.6666667
[5,] -1.1111111
[6,] 1.5555556
[7,] 2.0000000
[8,] -0.2222222
[9,] -0.6666667
[10,] -2.0000000
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")
library(neuralnet)
set.seed(2021)
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)
response=attribute^2
data=cbind(attribute, response)
colnames(data)=c("attribute", "response")
plot(data$attribute,data$response )
fit=neuralnet(response~attribute, data=data, hidden = c(3,3), threshold=0.01)
testdata=as.matrix(sample(seq(-2,2,length=10), 10, replace=FALSE), ncol=1)
pred=compute(fit, testdata)
pred
```

```
> pred
$neurons
$neurons[[1]]
      [,1]      [,2]
[1,]      1 0.2222222
[2,]      1 -1.5555556
[3,]      1 1.1111111
[4,]      1 0.6666667
[5,]      1 -1.1111111
[6,]      1 1.5555556
[7,]      1 2.0000000
[8,]      1 -0.2222222
[9,]      1 -0.6666667
[10,]     1 -2.0000000
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

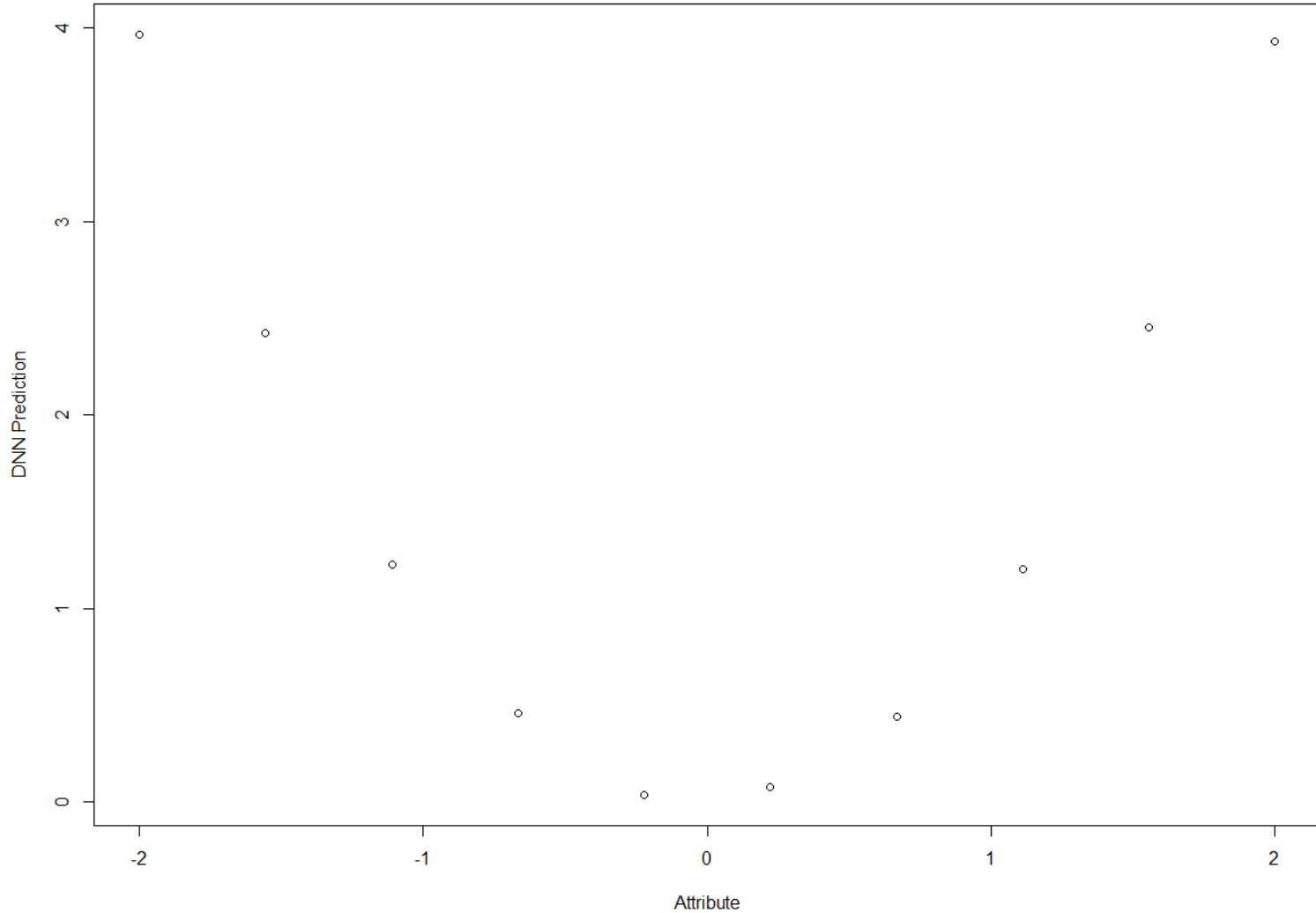
```
install.packages("neuralnet")
library(neuralnet)
set.seed(2021)
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)
response=attribute^2
data=cbind(attribute, response)
colnames(data)=c("attribute", "response")
plot(data$attribute,data$response )
fit=neuralnet(response~attribute, data=data, hidden = c(3,3), threshold=0.01)
testdata=as.matrix(sample(seq(-2,2,length=10), 10, replace=FALSE), ncol=1)
pred=compute(fit, testdata)
result=cbind(testdata, pred$net.result, testdata^2)
colnames(result)=c("Attribute", "Prediction", "Actual")
result
round(result,4)
```

```
> round(result,4)
      Attribute Prediction Actual
[1,]    0.2222    0.0785  0.0494
[2,]   -1.5556    2.4213  2.4198
[3,]    1.1111    1.2013  1.2346
[4,]    0.6667    0.4395  0.4444
[5,]   -1.1111    1.2254  1.2346
[6,]    1.5556    2.4521  2.4198
[7,]    2.0000    3.9317  4.0000
[8,]   -0.2222    0.0364  0.0494
[9,]   -0.6667    0.4554  0.4444
[10,]  -2.0000    3.9675  4.0000
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

```
install.packages("neuralnet")
library(neuralnet)
set.seed(2021)
attribute=as.data.frame(sample(seq(-2,2,length=50),50, replace=FALSE), ncol=1)
response=attribute^2
data=cbind(attribute, response)
colnames(data)=c("attribute", "response")
plot(data$attribute,data$response )
fit=neuralnet(response~attribute, data=data, hidden = c(3,3), threshold=0.01)
testdata=as.matrix(sample(seq(-2,2,length=10), 10, replace=FALSE), ncol=1)
pred=compute(fit, testdata)
result=cbind(testdata, pred$net.result, testdata^2)
colnames(result)=c("Attribute", "Prediction", "Actual")
result
round(result,4)
plot(pred$net.result~testdata, xlab="Attribute", ylab="DNN Prediction")
```

# 딥러닝 (Deep learning) (심층신경망(DNN))



# 딥러닝 (Deep learning) (심층신경망(DNN))

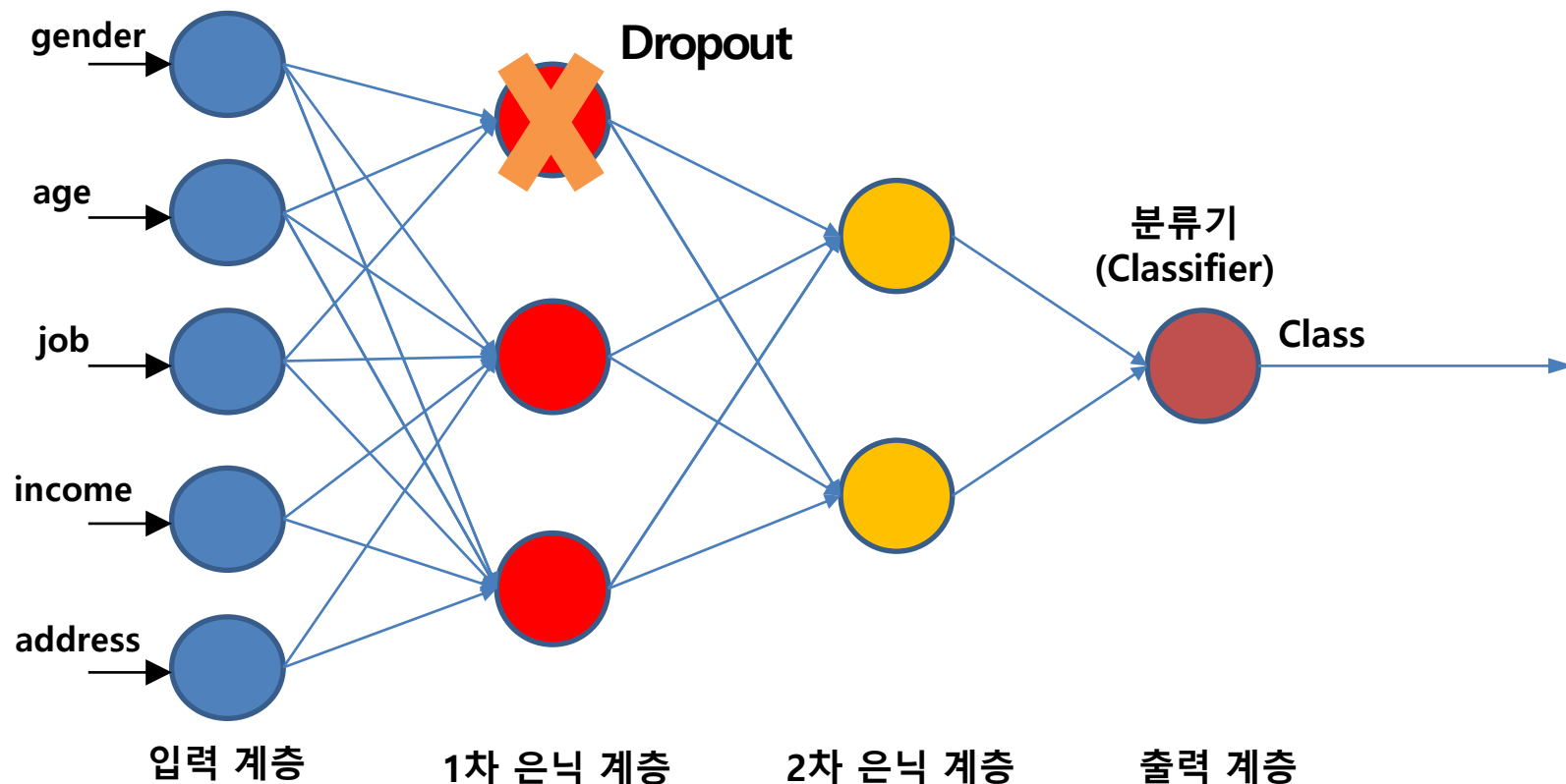
- 얼마나 많은 뉴런이 필요 한가 ?
  - 많은 뉴런의 과적합의 위험이 있음
  - 최적의 DNN을 일반화 하기 위해서는 용인 할 수 있는 오차 범위 내에서 가능한 적은 수의 뉴런을 사용
- 최적의 계층의 수는 ?
  - 모델 선택의 문제
  - 경험
- DNN 성능 개선 방법
  - Dropout : 은닉 계층 뉴런을 무작위로 포기 하는 것
    - 약한 학습 (weak learning) 유도
    - 약한 모델은 각각 낮은 예측 능력을 가지지만 여러 약한 모델의 예측에 가중치가 적용 되어 하나로 합쳐지면 훨씬 강력한 예측력을 가짐
    - 변수간 상관관계가 높아 발생하는 다중공산성 문제를 해결하는 원리
    - 뉴런간 다중 공산성 감소 → 모델의 일반화



# 딥러닝 (Deep learning) (심층신경망(DNN))

## · DNN 성능 개선 방법

- Dropout : 은닉 계층 뉴런을 무작위로 포기 하는 것



[2 개의 은닉 계층이 있는 피드포워드(Feed Forward) 신경망]

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")  
library(neuralnet)  
install.packages("Metrics")  
library(Metrics)  
data("Boston", package="MASS")  
data=Boston  
head(data)
```

```
> head(data)  
      crim zn  indus chas   nox   rm   age   dis rad tax ptratio  black  lstat medv  
1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900  1 296    15.3 396.90  4.98 24.0  
2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671  2 242    17.8 396.90  9.14 21.6  
3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671  2 242    17.8 392.83  4.03 34.7  
4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622  3 222    18.7 394.63  2.94 33.4  
5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622  3 222    18.7 396.90  5.33 36.2  
6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622  3 222    18.7 394.12  5.21 28.7
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

														1940년 이전 건물 비율	세율	소득 하위층 비율				
> head(data)																				
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv						
1	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	타운별 범죄율	타운별 비상업지구 비율				
2	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6			Nox 집중도 Nitrogen oxide (질소산화물)			
3	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7				평균 거주 가구		
4	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4					Boston 고용센터의 평균거리	
5	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2						타운별 교사-초등학생 비율
6	0.02985	0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7						

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
> head(data)
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
1	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
2	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
3	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
4	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
5	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
6	0.02985	0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7

crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
nox	nitric oxides concentration (parts per 10 million)
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built prior to 1940
dis	weighted distances to five Boston employment centres
rad	index of accessibility to radial highways
tax	full-value property-tax rate per USD 10,000
ptratio	pupil-teacher ratio by town
b	$1000(B - 0.63)^2$ where $B$ is the proportion of blacks by town
lstat	percentage of lower status of the population
medv	median value of owner-occupied homes in USD 1000's

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
head(data)
```

```
> head(data)
      crim      zn      indus      chas      nox      rm      age      dis      rad      tax      ptratio      black
1 -0.4193669  0.2845483 -1.2866362 -0.2723291 -0.1440749  0.4132629 -0.1198948  0.140075 -0.9818712 -0.6659492 -1.4575580  0.4406159
2 -0.4169267 -0.4872402 -0.5927944 -0.2723291 -0.7395304  0.1940824  0.3668034  0.556609 -0.8670245 -0.9863534 -0.3027945  0.4406159
3 -0.4169290 -0.4872402 -0.5927944 -0.2723291 -0.7395304  1.2814456 -0.2655490  0.556609 -0.8670245 -0.9863534 -0.3027945  0.3960351
4 -0.4163384 -0.4872402 -1.3055857 -0.2723291 -0.8344581  1.0152978 -0.8090878  1.076671 -0.7521778 -1.1050216  0.1129203  0.4157514
5 -0.4120741 -0.4872402 -1.3055857 -0.2723291 -0.8344581  1.2273620 -0.5106743  1.076671 -0.7521778 -1.1050216  0.1129203  0.4406159
6 -0.4166314 -0.4872402 -1.3055857 -0.2723291 -0.8344581  0.2068916 -0.3508100  1.076671 -0.7521778 -1.1050216  0.1129203  0.4101651
      lstat      medv
1 -1.0744990  0.1595278
2 -0.4919525 -0.1014239
3 -1.2075324  1.3229375
4 -1.3601708  1.1815886
5 -1.0254866  1.4860323
6 -1.0422909  0.6705582
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
```

```
library(neuralnet)
```

```
install.packages("Metrics")
```

```
library(Metrics)
```

```
data("Boston", package="MASS")
```

```
data=Boston
```

```
data=scale(data)
```

```
head(data)
```

```
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
```

```
data=data[,keeps]
```

```
head(data)
```

```
> head(data)
```

	crim	indus	nox	rm	age	dis	tax	ptratio	lstat	medv
1	-0.4193669	-1.2866362	-0.1440749	0.4132629	-0.1198948	0.140075	-0.6659492	-1.4575580	-1.0744990	0.1595278
2	-0.4169267	-0.5927944	-0.7395304	0.1940824	0.3668034	0.556609	-0.9863534	-0.3027945	-0.4919525	-0.1014239
3	-0.4169290	-0.5927944	-0.7395304	1.2814456	-0.2655490	0.556609	-0.9863534	-0.3027945	-1.2075324	1.3229375
4	-0.4163384	-1.3055857	-0.8344581	1.0152978	-0.8090878	1.076671	-1.1050216	0.1129203	-1.3601708	1.1815886
5	-0.4120741	-1.3055857	-0.8344581	1.2273620	-0.5106743	1.076671	-1.1050216	0.1129203	-1.0254866	1.4860323
6	-0.4166314	-1.3055857	-0.8344581	0.2068916	-0.3508100	1.076671	-1.1050216	0.1129203	-1.0422909	0.6705582

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
```

```
> apply(data,2, function(x) sum(is.na(x))) # 결측치 확인 -> 결측치 없다.
      crim      indus      nox      rm      age      dis      tax ptratio      lstat      medv
      0         0         0         0         0         0         0         0         0         0
~ |
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
n
```

```
> n
[1] 506
```



# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
train=sample(1:n, 400, FALSE)
head(train)
```

```
> head(train)
[1] 92 73 425 68 240 61
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
train=sample(1:n, 400, FALSE)
fit=neuralnet(f,data=data[train,], hidden=c(10,12,20), algorithm="rprop+",
err.fct = "sse", act.fct="logistic", threshold=0.1, linear.output=TRUE)
plot(fit)
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
fit=neuralnet(f,data=data[train,], hidden=c(10,12,20), algorithm="rprop+",  
err.fct = "sse", act.fct="logistic", threshold=0.1, linear.output=TRUE)
```

↓  
임계치

↓  
선형 활성화 함수 적용 여부

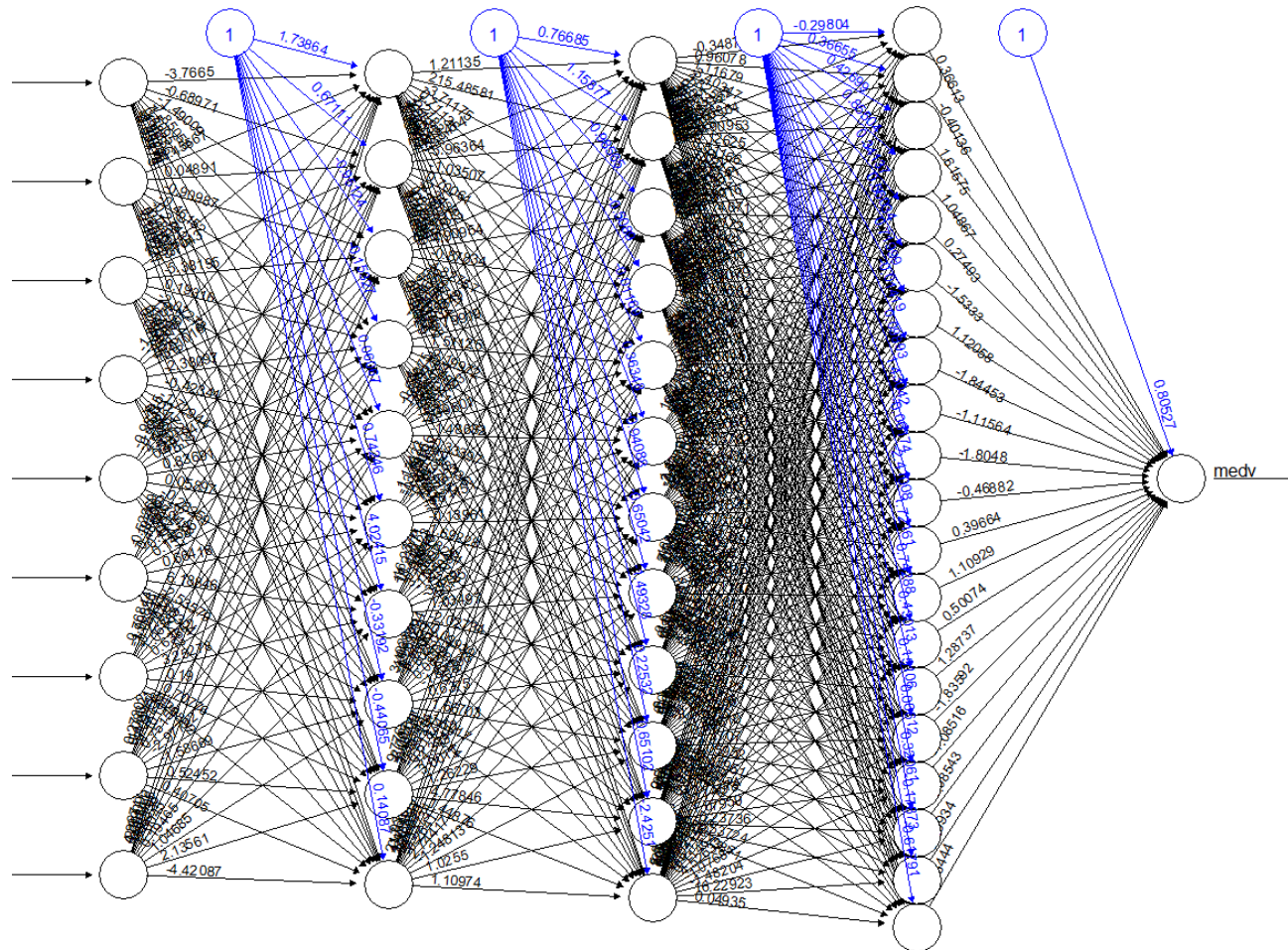
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat

↓  
학습알고리즘 : resilient backpropagation with backtracking  
(역전파 알고리즘 : algorithm="backprop", learningrate=0.01)

↓  
오차함수 : sum of squared error(제곱 오차의 합)

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측



# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
train=sample(1:n, 400, FALSE)
fit=neuralnet(f,data=data[train,], hidden=c(10,12,20), algorithm="rprop+",
err.fct = "sse", act.fct="logistic", threshold=0.1, linear.output=TRUE)
pred=compute(fit, data[-train, 1:9])
round(cor(pred$net.result, data[-train, 10])**2,6)
```

```
> round(cor(pred$net.result, data[-train, 10])**2,6)
      [,1]
[1,] 0.809458
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
train=sample(1:n, 400, FALSE)
fit=neuralnet(f,data=data[train,], hidden=c(10,12,20), algorithm="rprop+",
err.fct = "sse", act.fct="logistic", threshold=0.1, linear.output=TRUE)
pred=compute(fit, data[-train, 1:9])
round(cor(pred$net.result, data[-train, 10])**2,6)
mse(data[-train,10], pred$net.result)
(mean squared error)
```

```
> mse(data[-train,10], pred$net.result)
[1] 0.2607602
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
train=sample(1:n, 400, FALSE)
fit=neuralnet(f,data=data[train,], hidden=c(10,12,20), algorithm="rprop+",
err.fct = "sse", act.fct="logistic", threshold=0.1, linear.output=TRUE)
pred=compute(fit, data[-train, 1:9])
round(cor(pred$net.result, data[-train, 10])**2,6)
rmse(data[-train,10], pred$net.result)
(root mean squared error)
```

```
> rmse(data[-train,10], pred$net.result)
[1] 0.5106468
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측

```
Install.packages("neuralnet")
library(neuralnet)
install.packages("Metrics")
library(Metrics)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
apply(data,2, function(x) sum(is.na(x)))
f=medv~crim+indus+nox+rm+age+dis+tax+ptratio+lstat
set.seed(2016)
n=nrow(data)
train=sample(1:n, 400, FALSE)
fit=neuralnet(f,data=data[train,], hidden=c(10,12,20), algorithm="rprop+",
err.fct = "sse", act.fct="logistic", threshold=0.1, linear.output=TRUE)
pred=compute(fit, data[-train, 1:9])
Response=data[-train,10]
Predicted_Value=pred$net.result
plot(Response~Predicted_Value)
```



# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측 (deepnet package 활용)

```
install.packages("deepnet")  
library(deepnet)  
data("Boston", package="MASS")  
data=Boston  
data=scale(data)  
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")  
data=data[,keeps]  
n=nrow(data)  
train=sample(1:n, 400, FALSE)  
X=data[train, 1:9]  
Y=data[train, 10]
```

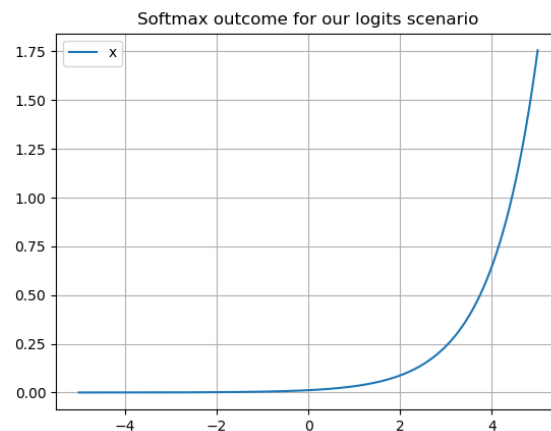
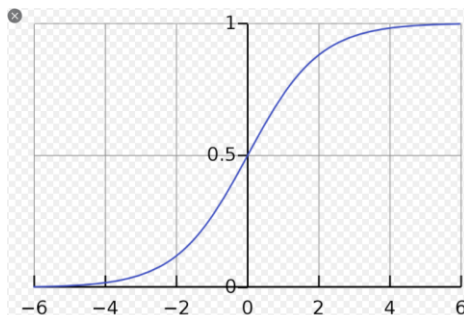
```
fitB=nn.train(x=X, y=Y, initW=NULL, initB=NULL, hidden=c(10,12,20),  
learningrate=0.58, momentum=0.74, learningrate_scale=1,  
activationfun="sigm", output="linear", numepochs=970, batchsize=60,  
hidden_dropout=0, visible_dropout=0)
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측 (deepnet package 활용)

```
fitB=nn.train(x=X, y=Y, initW=NULL, initB=NULL, hidden=c(10,12,20),  
learningrate=0.58, momentum=0.74, learningrate_scale=1,  
activationfun="sigm", output="linear", numepochs=970, batchsize=60,  
hidden_dropout=0, visible_dropout=0)
```

- initW : 뉴런의 가중치 (NULL=무작위)
- initB : 뉴런의 편향 (NULL=무작위)
- learning rate : 학습률 (신경망의 수렴 속도 조절)
- momentum : 모멘텀 (경사 하강 업데이트 과거 경사들에 대한 가중치 평균 포함  
오차 함수의 높은 곡률에서 특히 잡음을 줄임  
→ 네트워크가 일반적으로 겪는 시행착오를 줄이는 데 도움을 줌)
- learningrate\_scale=1 : 학습률 스케일
- activationfun : 은닉 계층 활성화 함수 (linear, tanh 함수도 가능)
- output : 출력 계층 활성화 함수 (sigmoid, softmax 함수도 가능)
- numepochs : 학습 횟수
- batch : 데이터 배치 크기
- hidden\_dropout
- visible\_dropout



# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측 (deepnet package 활용)

```
install.packages("deepnet")
library(deepnet)
data("Boston", package="MASS")
data=Boston
data=scale(data)
keeps=c("crim", "indus", "nox", "rm", "age", "dis", "tax", "ptratio", "lstat", "medv")
data=data[,keeps]
n=nrow(data)
train=sample(1:n, 400, FALSE)
X=data[train, 1:9]
Y=data[train, 10]
fitB=nn.train(x=X, y=Y, initW=NULL, initB=NULL, hidden=c(10,12,20),
learningrate=0.58, momentum=0.74, learningrate_scale=1,
activationfun="sigm", output="linear", numepochs=970, batchsize=60,
hidden_dropout=0, visible_dropout=0)
Xtest=data[-train, 1:9]
preB=nn.predict(fitB, Xtest)
round(cor(preB, data[-train,10])^2,6)
```

```
> round(cor(preB, `data[-train,10]`)^2,6)
      [,1]
[1,] 0.893618
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- Boston 주택 가격 예측 (deepnet package 활용)

```
install.packages("Metrics")
```

```
library(Metrics)
```

```
mse(data[-train,10], preB)
```

```
> mse(data[-train,10], preB)  
[1] 0.1110024
```

```
rmse(data[-train,10], preB)
```

```
> rmse(data[-train,10], preB)  
[1] 0.3331703
```

- 최적의 DNN 모델을 찾는 과정

- 네트워크의 구조 선택
- 뉴런의 계수 선택
- 학습 알고리즘 선택
- 파라미터 튜닝 의 조합

\* 꽤 복잡한 조합

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")  
head(PimaIndiansDiabetes2)
```

Prima 인디언 부족에서 21살 이상의 여성 768명에 대한 기록(당뇨와 소화기 질환)

```
> head(PimaIndiansDiabetes2)  
  pregnant glucose pressure triceps insulin mass pedigree age diabetes  
1         6    148      72      35      NA  33.6    0.627   50      pos  
2         1     85      66      29      NA  26.6    0.351   31      neg  
3         8    183      64      NA      NA  23.3    0.672   32      pos  
4         1     89      66      23     94  28.1    0.167   21      neg  
5         0    137      40      35    168  43.1    2.288   33      pos  
6         5    116      74      NA      NA  25.6    0.201   30      neg  
> |
```

- pregenat : 임신 횟수
- glucose : 플라스마 포도당의 집중도
- pressure : 혈압 (mm Hg)
- tricrps : 삼두근 두께(mm)
- insulin : 2시간 혈청의 인슐린(mm U/ml)
- mass : 비만도(BMI)
- pedigree : 당뇨 가족력 함수
- age : 나이
- diabetes : 당뇨 검사 결과 (음성(neg)/양성(pos))

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
```

```
> ncol(PimaIndiansDiabetes2)
[1] 9
> nrow(PimaIndiansDiabetes2)
[1] 768
> str(PimaIndiansDiabetes2)
'data.frame': 768 obs. of 9 variables:
 $ pregnant: num 6 1 8 1 0 5 3 10 2 8 ...
 $ glucose : num 148 85 183 89 137 116 78 115 197 125 ...
 $ pressure: num 72 66 64 66 40 74 50 NA 70 96 ...
 $ triceps : num 35 29 NA 23 35 NA 32 NA 45 NA ...
 $ insulin : num NA NA NA 94 168 NA 88 NA 543 NA ...
 $ mass : num 33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 NA ...
 $ pedigree: num 0.627 0.351 0.672 0.167 2.288 ...
 $ age : num 50 31 32 21 33 30 26 29 53 54 ...
 $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1 2 1 2 1 2 2 ...
```

NA : Not Available(missing value(결측치))

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")  
head(PimaIndiansDiabetes2)  
ncol(PimaIndiansDiabetes2)  
nrow(PimaIndiansDiabetes2)  
str(PimaIndiansDiabetes2)  
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x)))
```

 결측치 숫자 확인

```
> sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x)))  
pregnant  glucose  pressure  triceps  insulin    mass  pedigree    age  diabetes  
      0         5        35       227      374      11        0        0         0
```



# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x)))  결측치 숫자 확인
temp=PimaIndiansDiabetes2
temp$insulin=NULL
temp$triceps=NULL
head(temp)
```

```
> head(temp)
```

	pregnant	glucose	pressure	mass	pedigree	age	diabetes
1	6	148	72	33.6	0.627	50	pos
2	1	85	66	26.6	0.351	31	neg
3	8	183	64	23.3	0.672	32	pos
4	1	89	66	28.1	0.167	21	neg
5	0	137	40	43.1	2.288	33	pos
6	5	116	74	25.6	0.201	30	neg

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x))) 결측치 숫자 확인
temp=PimaIndiansDiabetes2
temp$insulin=NULL
temp$triceps=NULL
temp=na.omit(temp) missing value imputation, complete case
str(temp)
```

```
> str(temp)
'data.frame': 724 obs. of 7 variables:
 $ pregnant: num 6 1 8 1 0 5 3 2 4 10 ...
 $ glucose : num 148 85 183 89 137 116 78 197 110 168 ...
 $ pressure: num 72 66 64 66 40 74 50 70 92 74 ...
 $ mass : num 33.6 26.6 23.3 28.1 43.1 25.6 31 30.5 37.6 38 ...
 $ pedigree: num 0.627 0.351 0.672 0.167 2.288 ...
 $ age : num 50 31 32 21 33 30 26 53 30 34 ...
 $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1 2 1 2 2 1 2 ...
 - attr(*, "na.action")= 'omit' Named int [1:44] 8 10 16 50 61 76 79 82 146 173 ...
 .. attr(*, "names")= chr [1:44] "8" "10" "16" "50" ...
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x)))  결측치 숫자 확인
temp=PimaIndiansDiabetes2
temp$insulin=NULL
temp$triceps=NULL
temp=na.omit(temp)  missing value imputation, complete case
str(temp)
nrow(temp)
ncol(temp)
```

```
> nrow(temp)
[1] 724
> ncol(temp)
[1] 7
```

```
> ncol(PimaIndiansDiabetes2)  (was)
[1] 9
> nrow(PimaIndiansDiabetes2)
[1] 768
> str(PimaIndiansDiabetes2)
'data.frame': 768 obs. of 9 variables:
 $ pregnant: num  6 1 8 1 0 5 3 10 2 8 ...
 $ glucose : num  148 85 183 89 137 116 78 115 197 125 ...
 $ pressure: num  72 66 64 66 40 74 50 NA 70 96 ...
 $ triceps : num  35 29 NA 23 35 NA 32 NA 45 NA ...
 $ insulin : num  NA NA NA 94 168 NA 88 NA 543 NA ...
 $ mass : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 NA ...
 $ pedigree: num  0.627 0.351 0.672 0.167 2.288 ...
 $ age : num  50 31 32 21 33 30 26 29 53 54 ...
 $ diabetes: Factor w/ 2 levels "neg","pos": 2 1 2 1 2 1 2 1 2 2 ...
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x)))  결측치 숫자 확인
temp=PimaIndiansDiabetes2
temp$insulin=NULL
temp$triceps=NULL
temp=na.omit(temp)  missing value imputation, complete case
str(temp)
nrow(temp)
ncol(temp)
y=temp$diabetes
temp$diabetes=NULL
head(temp)
```

```
> head(temp)
  pregnant glucose pressure mass pedigree age
1         6    148      72  33.6    0.627  50
2         1     85      66  26.6    0.351  31
3         8    183      64  23.3    0.672  32
4         1     89      66  28.1    0.167  21
5         0    137      40  43.1    2.288  33
6         5    116      74  25.6    0.201  30
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x))) 결측치 숫자 확인
temp=PimaIndiansDiabetes2
temp$insulin=NULL
temp$triceps=NULL
temp=na.omit(temp) missing value imputation, complete case
y=temp$diabetes
temp$diabetes=NULL
temp=scale(temp)
temp=cbind(as.factor(y), temp)
head(temp)
```

```
> head(temp)
  pregnant glucose pressure mass pedigree age
1 2 0.6345831 0.8493456 -0.03235514 0.1644481 0.4581039 1.4150969
2 1 -0.8522718 -1.1994329 -0.51701289 -0.8516733 -0.3724334 -0.1998088
3 2 1.2293251 1.9875559 -0.67856547 -1.3307019 0.5935176 -0.1148137
4 1 -0.8522718 -1.0693517 -0.51701289 -0.6339330 -0.9261249 -1.0497592
5 2 -1.1496428 0.4916224 -2.61719644 1.5434699 5.4563736 -0.0298187
6 1 0.3372121 -0.1913038 0.12919744 -0.9968335 -0.8238123 -0.2848038
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
data("PimaIndiansDiabetes2", package="mlbench")
head(PimaIndiansDiabetes2)
ncol(PimaIndiansDiabetes2)
nrow(PimaIndiansDiabetes2)
str(PimaIndiansDiabetes2)
sapply(PimaIndiansDiabetes2, function(x) sum(is.na(x))) 결측치 숫자 확인
temp=PimaIndiansDiabetes2
temp$insulin=NULL
temp$triceps=NULL
temp=na.omit(temp) missing value imputation, complete case
temp$diabetes=ifelse(temp$diabetes=="pos", 1,2)
y=temp$diabetes
temp$diabetes=NULL
temp=scale(temp)
temp=cbind(as.factor(y), temp)
str(temp)
class(temp)
summary(temp)
```

```
> str(temp)
num [1:724, 1:7] 2 1 2 1 2 1 2 2 1 2 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:724] "1" "2" "3" "4" ...
..$ : chr [1:7] "" "pregnant" "glucose" "pressure" ...
> class(temp)
[1] "matrix" "array"
> summary(temp)
      v1      pregnant      glucose      pressure      mass      pedigree      age
Min.   :1.000   Min.   : -1.1496   Min.   : -2.5328   Min.   : -3.90962   Min.   : -2.071019   Min.   : -1.1939   Min.   : -1.0498
1st Qu.:1.000   1st Qu.: -0.8523   1st Qu.: -0.7198   1st Qu.: -0.67856   1st Qu.: -0.721029   1st Qu.: -0.6914   1st Qu.: -0.7948
Median :1.000   Median : -0.2575   Median : -0.1588   Median : -0.03236   Median : -0.009744   Median : -0.2882   Median : -0.3698
Mean   :1.344   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.000000   Mean   : 0.0000   Mean   : 0.0000
3rd Qu.:2.000   3rd Qu.: 0.6346   3rd Qu.: 0.6542   3rd Qu.: 0.61386   3rd Qu.: 0.599929   3rd Qu.: 0.4596   3rd Qu.: 0.6501
Max.   :2.000   Max.   : 3.9057   Max.   : 2.5079   Max.   : 4.00646   Max.   : 5.027315   Max.   : 5.8536   Max.   : 4.0499
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
n=nrow(temp)
head(n)      > head(n)
[1] 724
n_train=600
n_test=n-n_train
n_test      > n_test
[1] 124
train=sample(1:n, n_train, FALSE)
install.packages("RSNNS")
library(RSNNS)
set.seed(2016)
X=temp[train, 1:6]
Y=temp[train,7]
head(X)
head(y)
```

```
> head(x)
      pregnant glucose pressure mass pedigree age
139           0     129       80  31.2    0.703  29
112           8     155       62  34.0    0.543  46
648           0     179       50  37.8    0.455  22
105           2      85       65  39.6    0.930  27
367           6     124       72  27.6    0.368  29
96            6     144       72  33.9    0.255  40
> head(y)
[1] pos neg pos neg pos neg
Levels: neg pos
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

2개 은닉층, 12개 및 8개 뉴런

```
fitMLP=mlp(x=X, y=Y, size=c(12,8), maxit=1000, initFunc =  
                                                    무작위 가중치로 초기화  
"Randomize_Weights", initFuncParams = c(-0.3,0.3), learnFunc =  
"Std_Backpropagation", learnFuncParams = c(0.2,0), updateFunc =  
"Topological_Order", updateFuncParams = c(0), hiddenActFunc =  
                                                    은닉계층 활성화 함수 = 로지스틱 함수  
"Act_Logistic", shufflePatterns = TRUE, linOut = TRUE)
```



# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (RSNNS package 활용)

```
fitMLP=mlp(x=X, y=Y, size=c(12,8), maxit=1000, initFunc =  
"Randomize_Weights", initFuncParams = c(-0.3,0.3), learnFunc =  
"Std_Backpropagation", learnFuncParams = c(0.2,0), updateFunc =  
"Topological_Order", updateFuncParams = c(0), hiddenActFunc =  
"Act_Logistic", shufflePatterns = TRUE, linOut = TRUE)
```

```
predMLP=sign(predict(fitMLP, temp[-train, 1:6]))
```

```
table(predMLP, sign(temp[-train,7]), dnn=c("Predicted", "Observed"))  
error_rate=(1-sum(predMLP==sign(temp[-train,7]))/124)  
round(error_rate,3)
```

```
> table(predMLP, sign(temp[-train,7]), dnn=c("Predicted", "Observed"))  
      Observed  
Predicted -1  1  
      -1  59 10  
       1  19 36  
> error_rate=(1-sum(predMLP==sign(temp[-train,7]))/124)  
> round(error_rate,3)  
[1] 0.234
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (AMORE package 활용)

```
detach("package:RSNNS", unload=TRUE)
install.packages("AMORE")
library(AMORE)
net=newff(n.neurons = c(6,12,8,1), learning.rate.global = 0.01,
momentum.global = 0.5, error.criterium = "LMLS", Stao=NA, hidden.layer =
"sigmoid", output.layer = "purelin", method="ADAPTgdwm")
X=temp[train,]
Y=temp[train,7]
head(x)
head(Y)
fit=train(net, P=X, T=Y, error.criterium="LMLS", report=TRUE, show.step=100,
n.shows=5)
```

```
> fit=train(net, P=X, T=Y, error.criterium="LMLS", report=TRUE, show.step=100, n.shows=5)
index.show: 1 LMLS 0.185937506411178
index.show: 2 LMLS 0.183252239645921
index.show: 3 LMLS 0.179248091392619
index.show: 4 LMLS 0.175802413683804
index.show: 5 LMLS 0.172289628795579
```

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (AMORE package 활용)

입력변수 개수, 1<sup>st</sup> hidden layer neuron 개수....., 출력변수 개수

net=newff(n.neurons = c(6,12,8,1), learning.rate.global = 0.01,

momentum.global = 0.5, error.criterium = "LMLS", Stao=NA, hidden.layer =

"sigmoid", output.layer = "purelin", method="ADAPTgdwm")

학습 방법은 적응형 경사 하강 알고리즘 적용  
(adaptive gradient descend with momentum)

- MSE 최소화 적용시에는 이상치 존재를 알 수 없음.
- 강건한 오차 측정치 (robust error metric) LMLS(least mean log squared error, 평균 제곱 오차) 적용
- AMORE 패키지에서는 다른 강건한 오차 측정치 tao, 표준평균제곱 오차도 사용 가능

# 딥러닝 (Deep learning) (심층신경망(DNN))

- 분류를 잘하는 DNN (AMORE package 활용)

```
fit=train(net, P=X, T=Y, error.criterium="LMLS", report=TRUE, show.step=100,
n.shows=5)
pred=sign(sim(fit$net, temp[-train,]))
table(pred, sign(temp[-train,7]), dnn=c("Predicted", "Observed"))
error_rate=(1-sum(pred==sign(temp[-train,7]))/124)
round(error_rate, 3)
```

```
> table(pred, sign(temp[-train,7]), dnn=c("Predicted", "Observed"))
      Observed
Predicted -1  1
      -1  63 12
       1  15 34
> error_rate=(1-sum(pred==sign(temp[-train,7]))/124)
> round(error_rate, 3)
[1] 0.218
```