신경망 분석
(Neural Network Model)

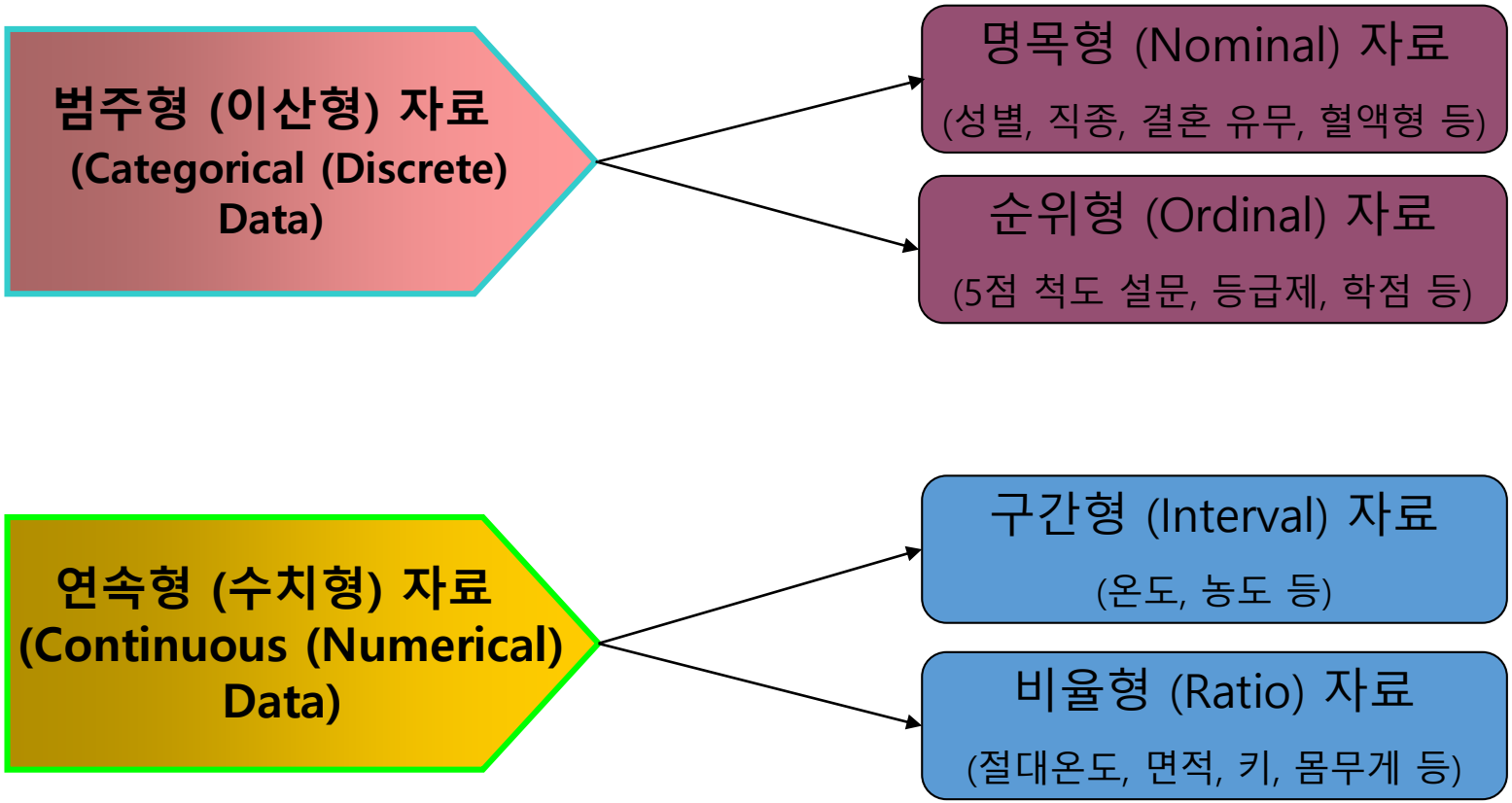
동국대학교 통계학과

이 영 섭

yung@dongguk.edu

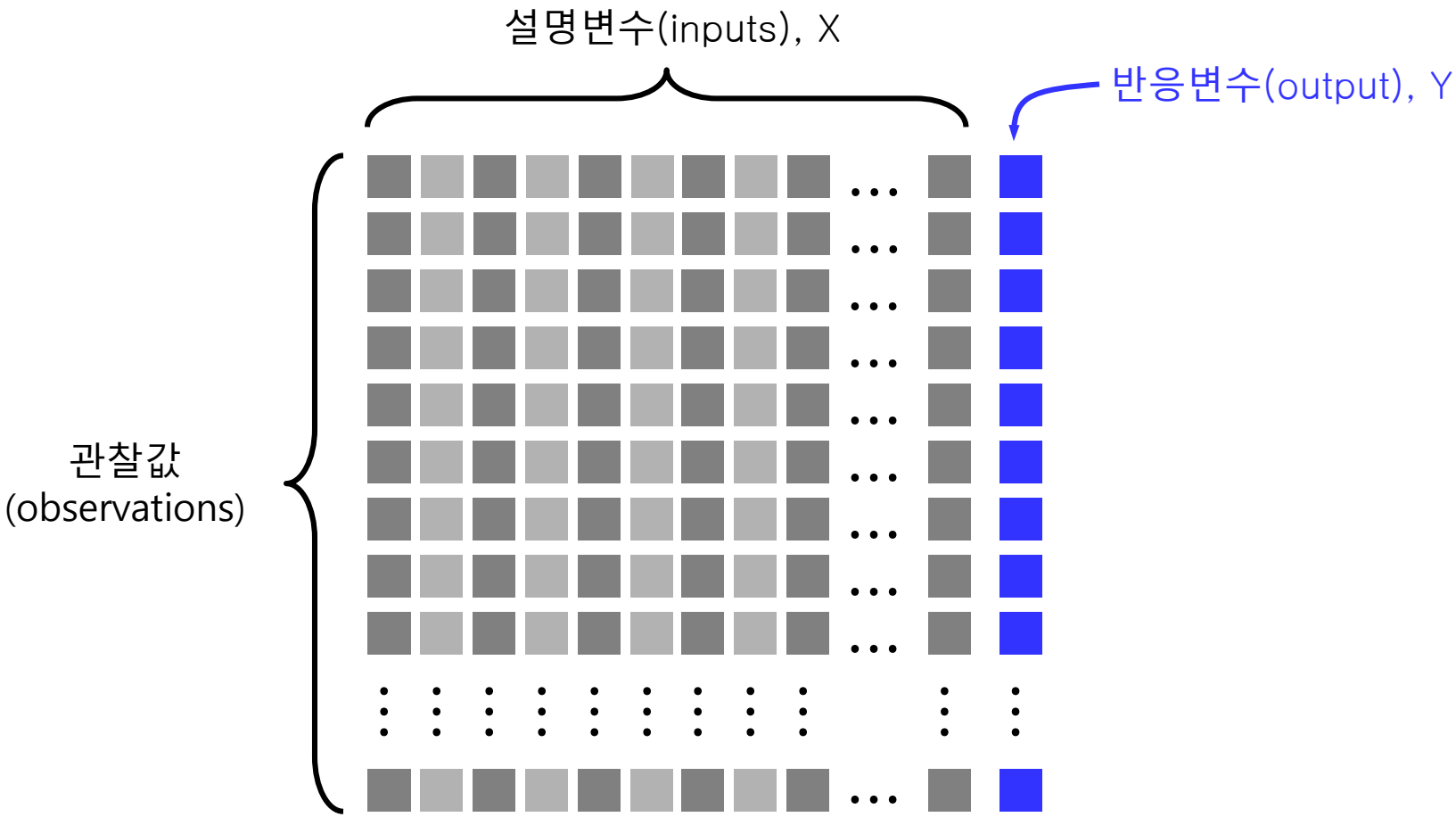
예측분석 모형
(Prediction Model)

데이터 종류



예측분석모형 데이터 형태 (데이터 구조)

(Supervised, Prediction, Classification or Regression)

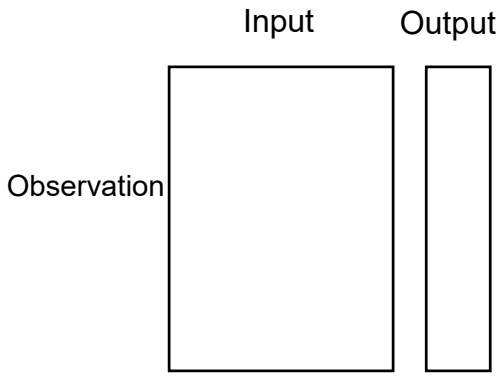


Notations

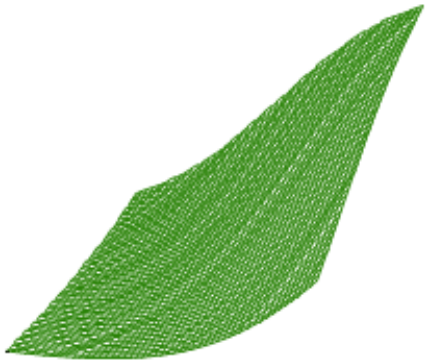
- Output (Response, Predictand, Dependent, Outcome, Target, Classes Categorical Responses, Labels, endogenous, Y, 반응변수, 종속변수, 목표변수, 부류, 레이블)

- Input (Variable, Predictors, Independent, Features, Explanatory Variables, Attributes, Covariates, Regressors, Fields, Properties, Magnitudes, Measurements, Dimensions, Characteristic, exogenous, term (word), X, 설명변수, 독립변수, 특징)

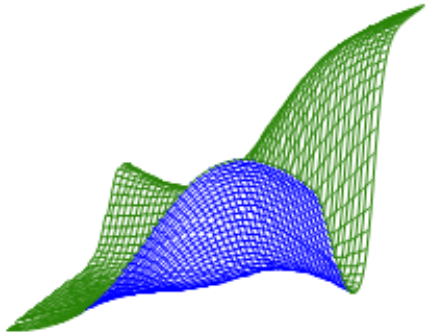
- Observation (Object, ID, Sample, Case, Record, Example, Entity, Event, Unit, Instance, Pattern, Point, Vector, Transaction, Tuple Document, n, 관찰값)



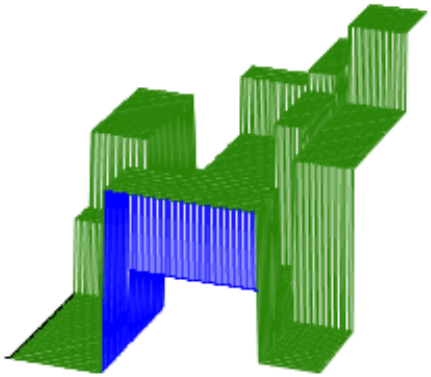
대표적인 데이터마이닝 분석 모형



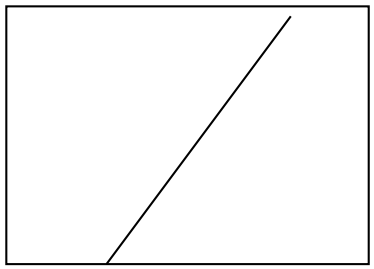
일반화선형모형
(Generalized Linear Models):
회귀모형(Regression) /
로지스틱회귀(Logistic Regression)



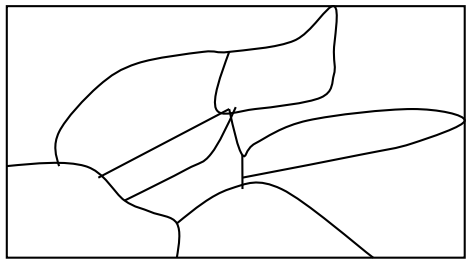
신경망 모형
(Neural
Networks)



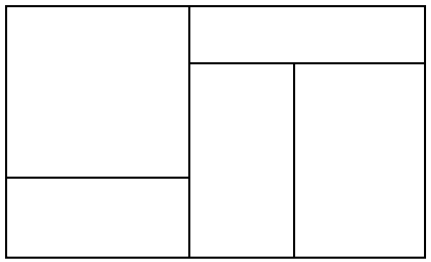
의사결정나무
(Decision
Trees)



Linear regression



Neural networks



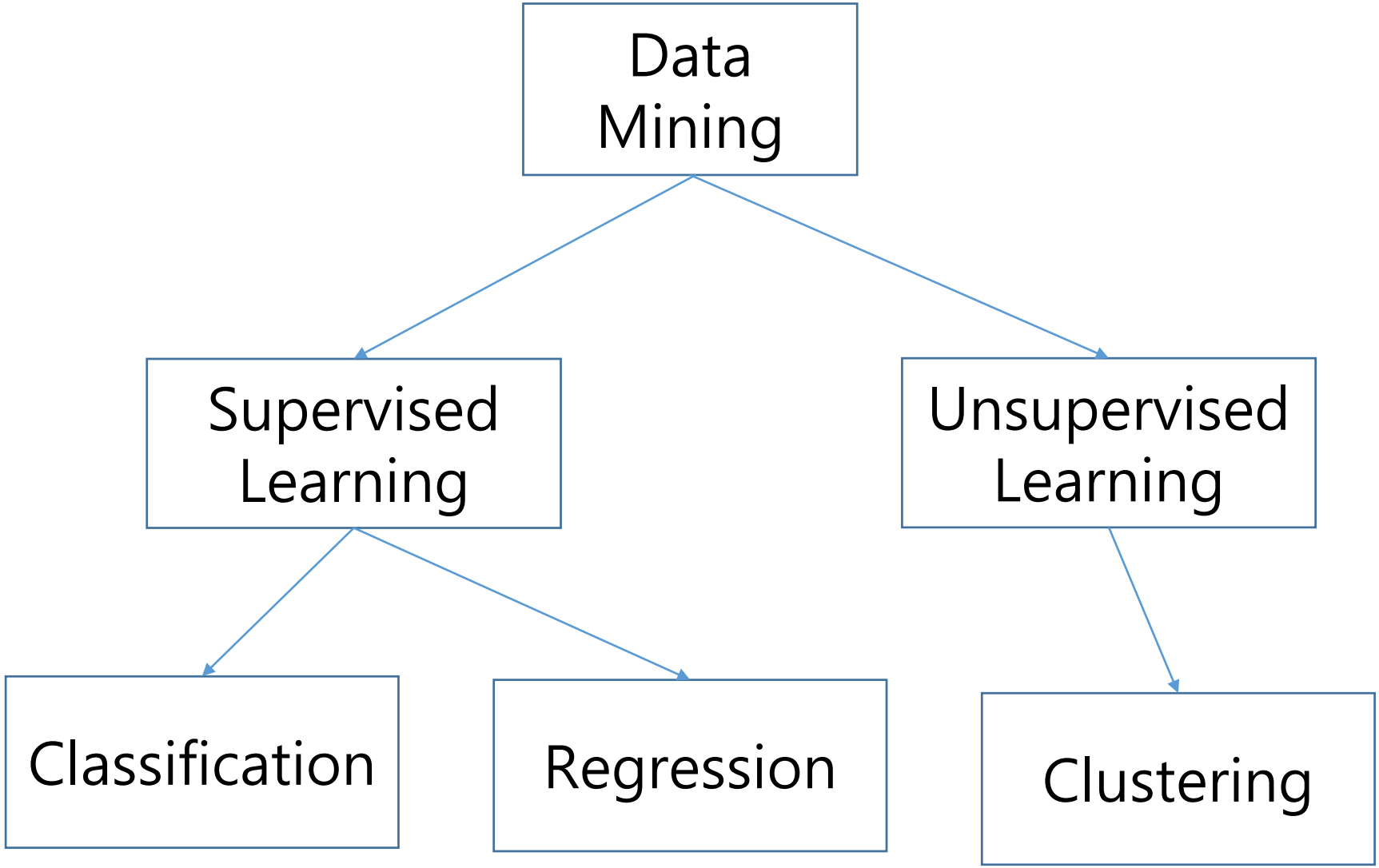
Classification trees

참고: 데이터마이닝기법 ≈ 빅데이터분석기법 ≈ 기계학습(머신러닝)기법

일반화선형모형(GLM)

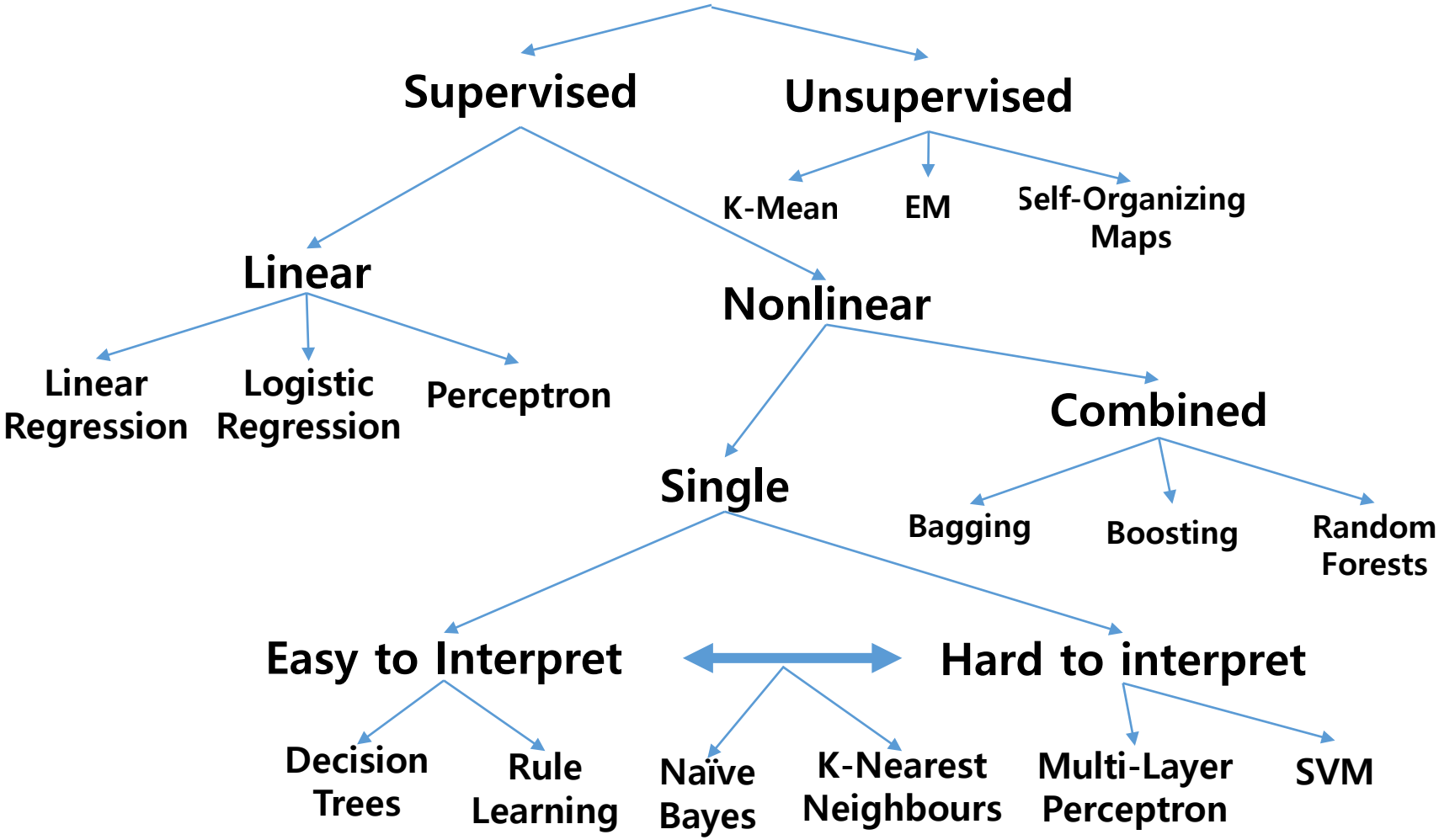
반응변수(Y) / 설명변수(X)	연속형	범주형
연속형	선형회귀 모형	로지스틱 모형
범주형	분산 분석	분할표 분석 로지스틱 모형 로그선형 모형
연속형 + 범주형	공분산 분석 가변수 모형	로지스틱 모형

데이터마이닝 분석 (빅데이터 분석)기법 분류



참고: 데이터마이닝기법 ≈ 빅데이터분석기법 ≈ 기계학습(머신러닝)기법

A Taxonomy of Machine Learning Techniques :



http://slideplayer.com/slide/2412766/

데이터마이닝 분석 (빅데이터 분석)기법 분류

	Unsupervised	Supervised
Continuous	● Clustering & Dimensionality Reduction o SVD o PCA o K-means	● Regression o Linear o Polynomial ● Decision Trees ● Random Forests
Categorical	● Association Analysis o Apriori o FP-Growth ● Hidden Markov Model	● Classification o KNN o Trees o Logistics Regression o Naïve-Bayes o SVM

출처: https://nyghtowl.io/category/data-science/machine-learning/

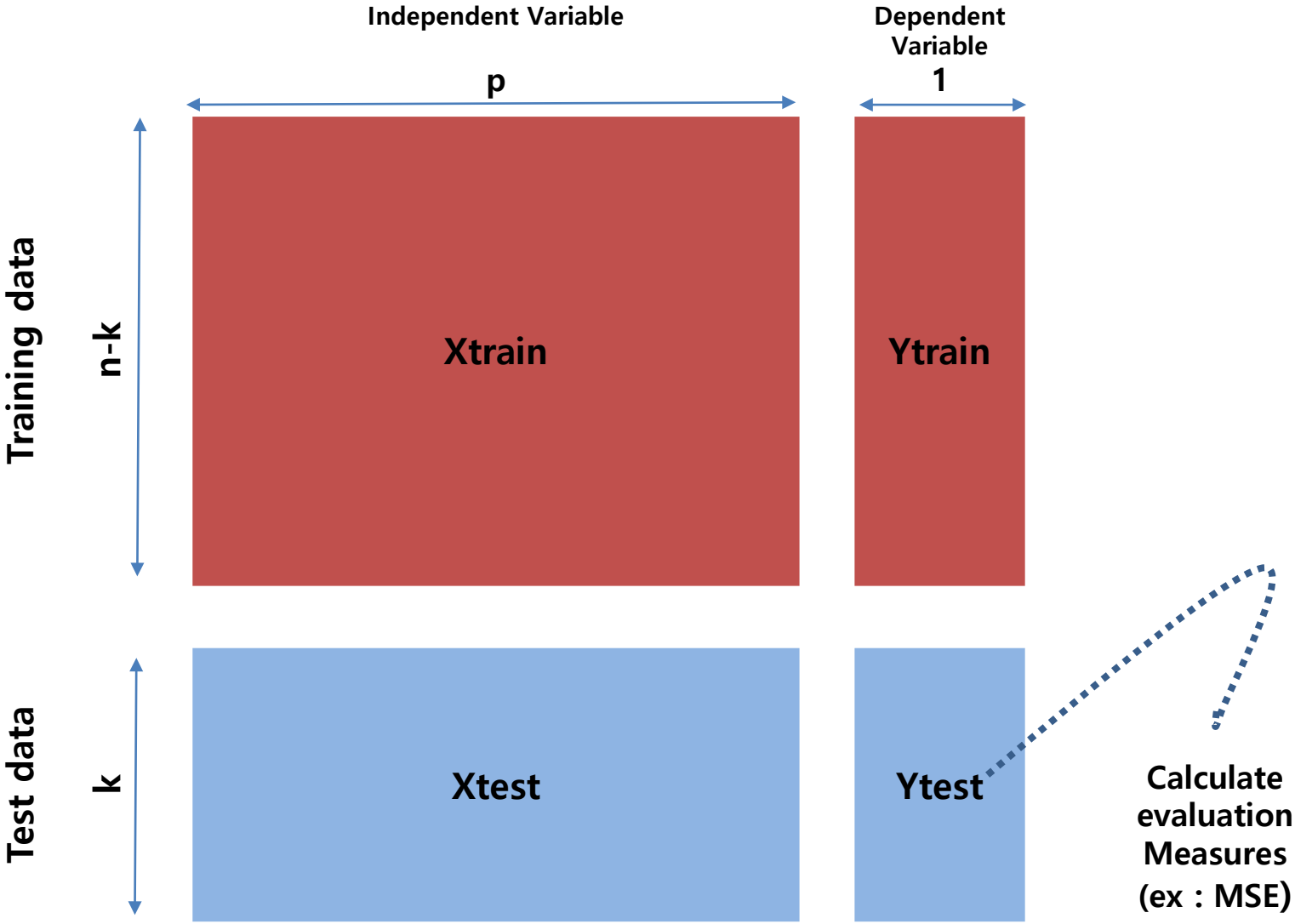
예측분석모형 데이터 형태

(Supervised, Prediction, Classification or Regression)

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class*.
- Find a *model* for class attribute as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

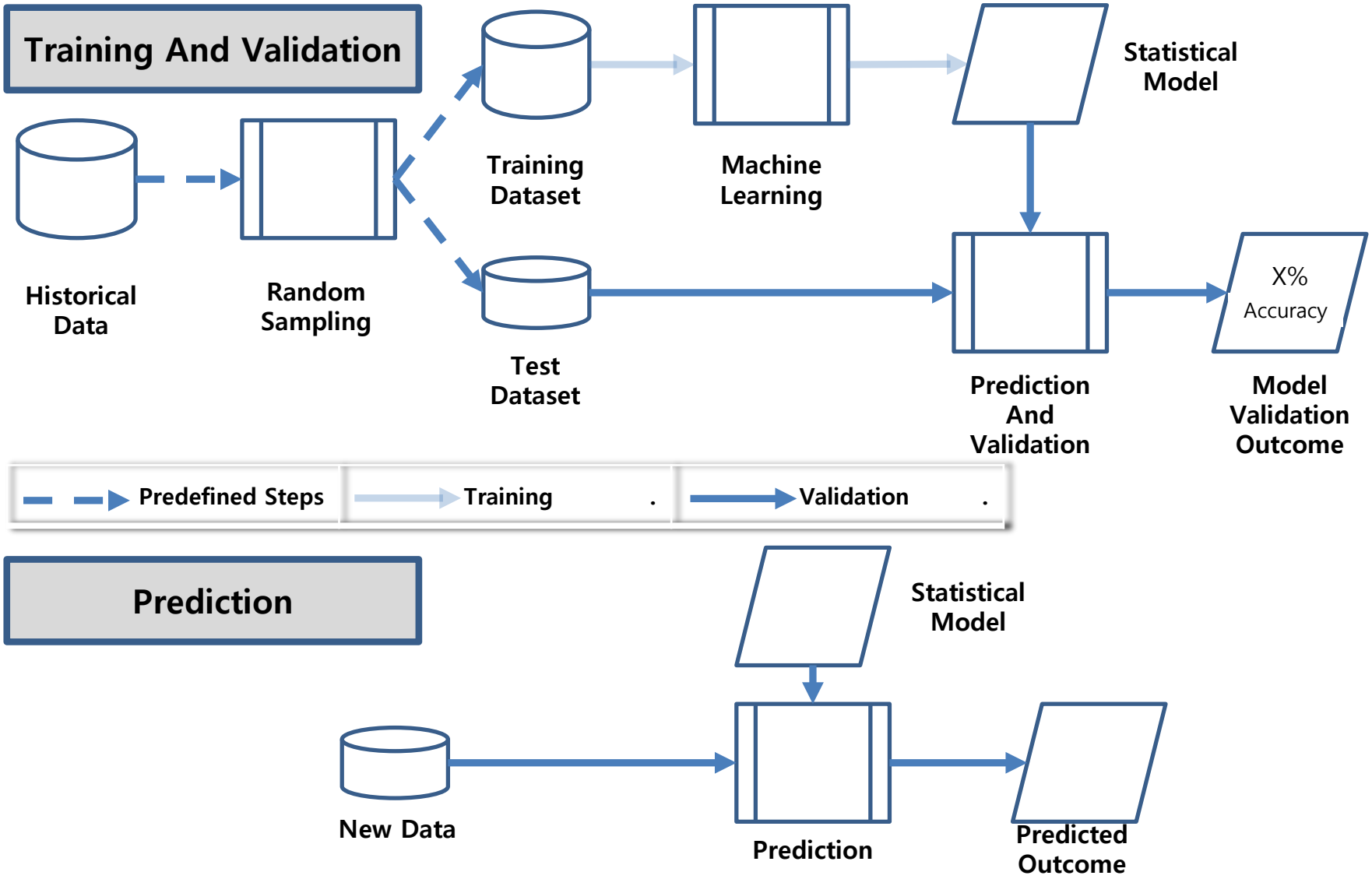
예측분석모형 데이터 형태

(Supervised, Prediction, Classification or Regression)



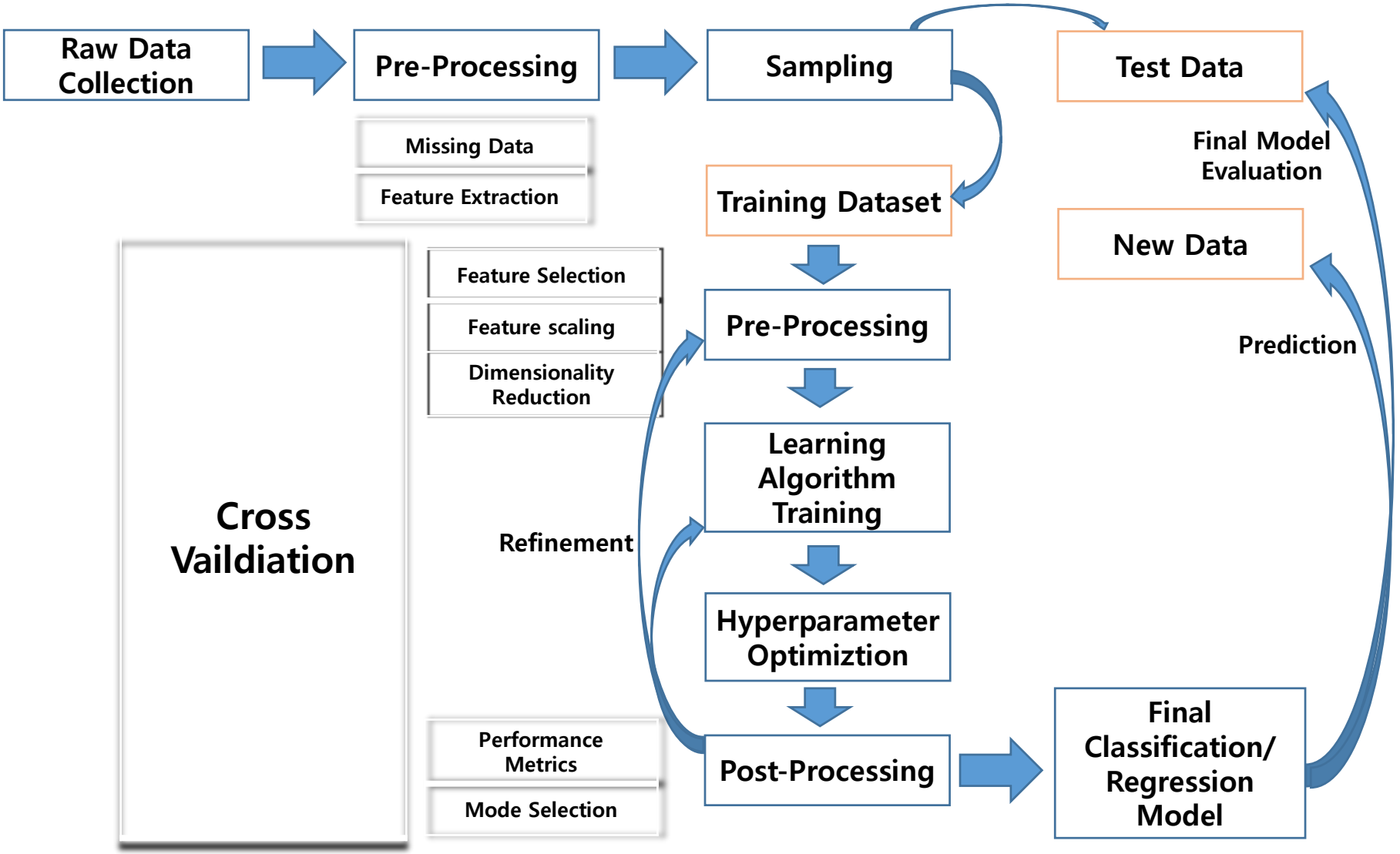
출처: https://www.lucypark.kr/courses/2015-dm/multiple-linear-regression.html

Supervised Learning Process (통계적인 관점)

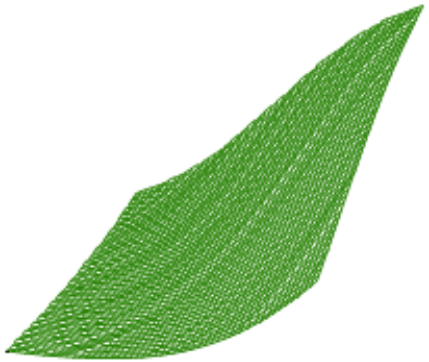


출처: http://www.edureka.co/blog/introduction-to-supervised-learning/

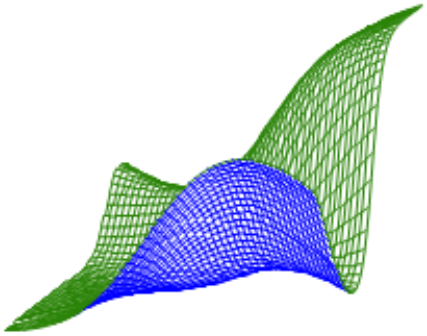
Supervised Learning Process (기계학습 관점)



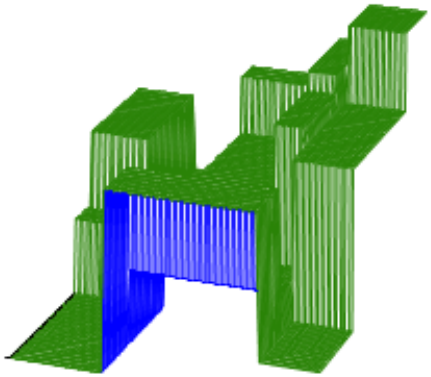
대표적인 데이터마이닝 분석 모형



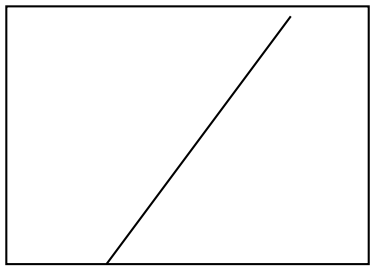
일반화선형모형
(Generalized Linear Models):
회귀모형(Regression) /
로지스틱회귀(Logistic Regression)



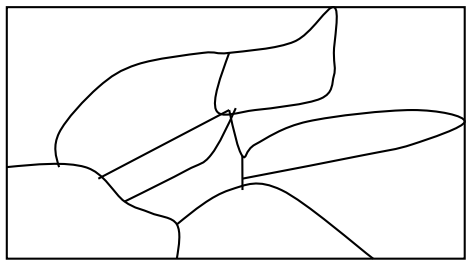
신경망 모형
(Neural
Networks)



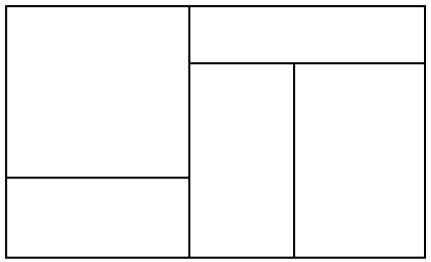
의사결정나무
(Decision
Trees)



Linear regression



Neural networks



Classification trees

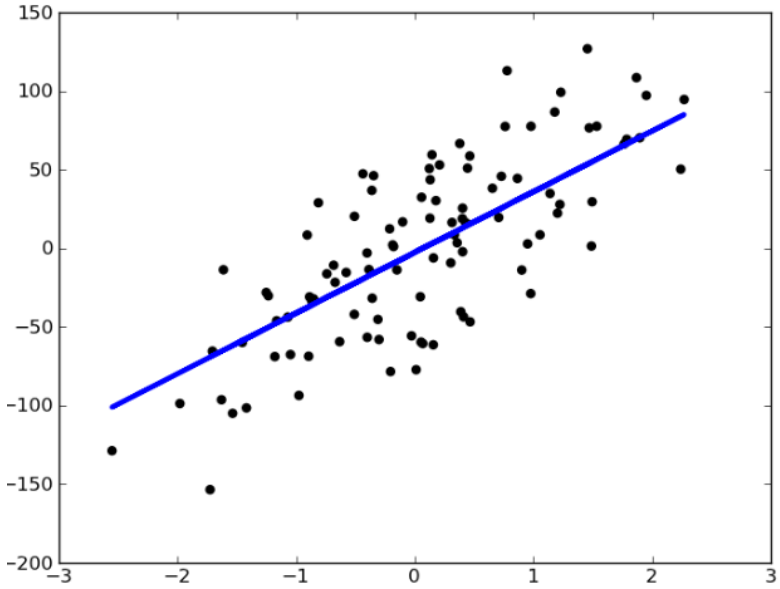
참고: 데이터마이닝기법 ≈ 빅데이터분석기법 ≈ 기계학습(머신러닝)기법

일반화 선형 모델 (Generalized Linear Model, GLM)

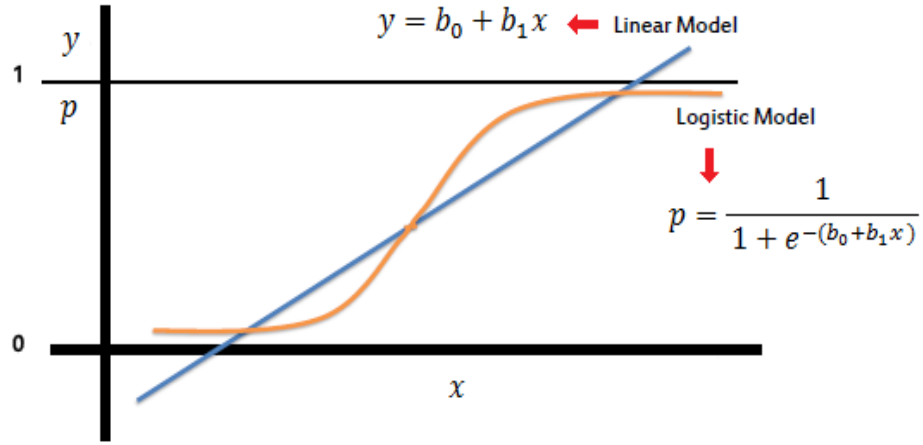
반응변수(Y) / 설명변수(X)	연속형	범주형
연속형	선형회귀 모형	로지스틱 모형
범주형	분산 분석	분할표 분석 (하나의 X) 로지스틱 모형 또는 로그 선형 모형(여러 개의 Xs)
연속형 + 범주형	공분산 분석 선형회귀(가변수) 모형	로지스틱 모형

일반화 선형 모형(Generalized Linear Models)

회귀(Regression) 모형



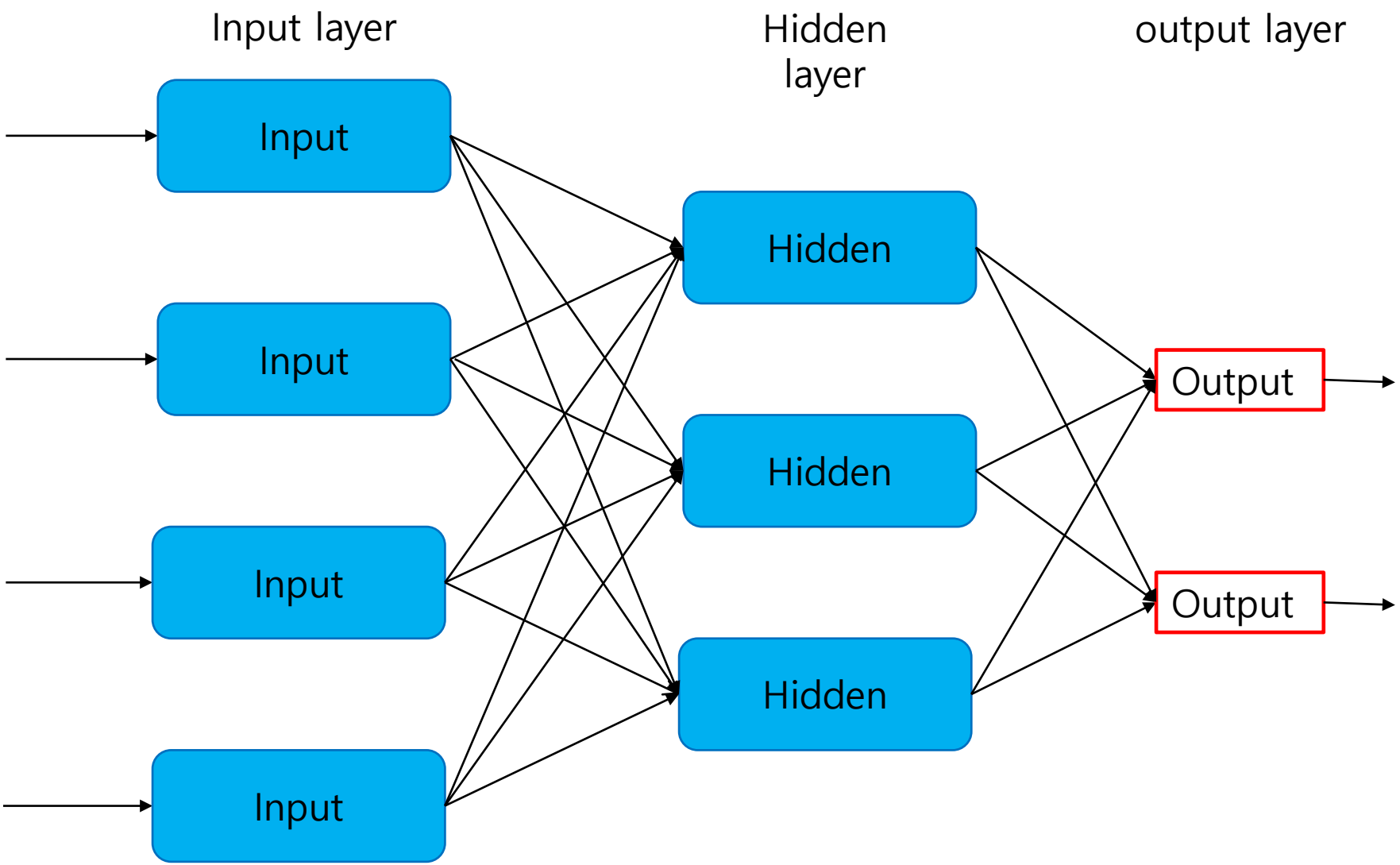
로지스틱회귀(Logistic Regression) 모형



일반화 선형 모형(Generalized Linear Models)

	B	C	D	E	F	G	H	I	J	K	L	M	N
1	site	ymd	Tem_Mean	Tem_max	Tem_min	Rain	WS_Mean	WS_Max	Solar_sum		**Pollen(Y)**	Pollen 0/1	
162	서울	2014-08-08	26.1	30.6	21.9		3.8	6.9	19.64		0	0	
163	서울	2014-08-09	26.5	31.6	22.3		4.6	7.3	22.35		0	0	
164	서울	2014-08-10	23.4	27.7	19.6	42.5	2.6	9.3	6.66		0	0	
165	서울	2014-08-11	23.7	28.6	18.3		2.2	5.2	19.44		1	1	
166	서울	2014-08-12	25.8	30.4	22.3		2.7	4.9	16.57		36	1	
167	서울	2014-08-13	24.2	26.7	22.5	0	2.1	4.6	6.14		1	1	
168	서울	2014-08-14	22.3	24.2	21.4	0.5	2	3.8	4.32		3	1	
169	서울	2014-08-15	24.9	29.6	20.9		1.7	4.5	12.53		9	1	
170	서울	2014-08-16	26.1	31.8	22.2		1.8	5.1	15.5		0	0	
171	서울	2014-08-17	24	25	22.9	3	1.9	4.2	3.2		0	0	
172	서울	2014-08-18	22.3	24.4	20.9	11.5	3.1	5	6.42		0	0	
173	서울	2014-08-19	22.3	25.3	20.4	4.5	1.5	3.5	4.37		0	0	
174	서울	2014-08-20	24.8	28	22.4	0.5	1.1	2.5	6.74		15	1	
175	서울	2014-08-21	23.1	24.6	21.8	46.5	2.4	5.2	4.94		4	1	
176	서울	2014-08-22	24.7	28.2	21.8	15	3.1	5.9	13.56		64	1	
177	서울	2014-08-23	25.2	29.7	22.8	0.5	2.7	4.4	12.29		1	1	
178	서울	2014-08-24	25.6	30.7	20.7	0.1	1.3	4.5	15.04		4	1	
179	서울	2014-08-25	25.9	28.6	24.1	0	3	4.8	8.43		0	0	
180	서울	2014-08-26	25.4	29.6	21.6	11.5	2.5	4.8	11.38		14	1	
181	서울	2014-08-27	24.6	28.6	21.1		2.8	4	14.77		3	1	
182	서울	2014-08-28	25.1	30.7	19.8		2.2	4.6	18.41		63	1	
183	서울	2014-08-29	25.2	30.1	21	6.5	1.8	5	14.21		20	1	
184	서울	2014-08-30	24.6	30.7	19.9		1.9	4.3	17.6		50	1	
185	서울	2014-08-31	24.1	29	20.7		1.4	3	12.18		35	1	
186	서울	2014-09-01	24.1	29.7	20.4	1.5	1.5	5.1	13.6		16	1	
187	서울	2014-09-02	23.1	27.7	19.3	6	2.5	5.6	10.67		39	1	

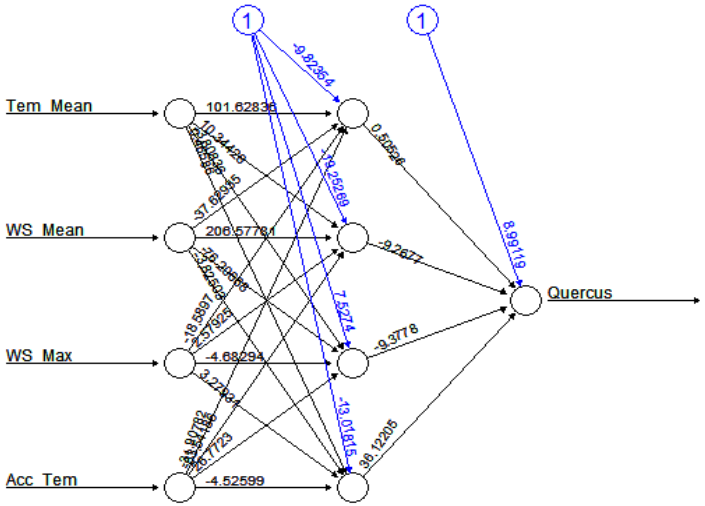
신경망 모형(Neural network model)



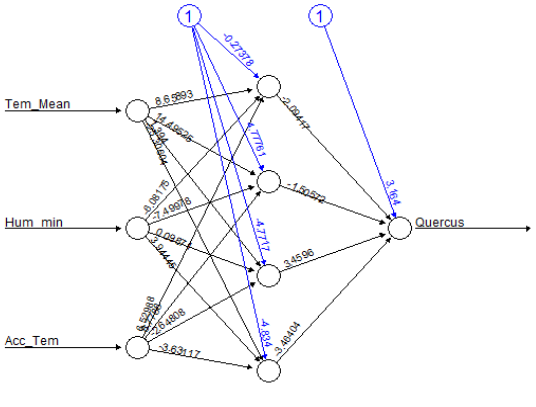
신경망 모형(Neural network model)

- **기상요소와 참나무 꽃가루농도의 신경망모형(neural networks)**

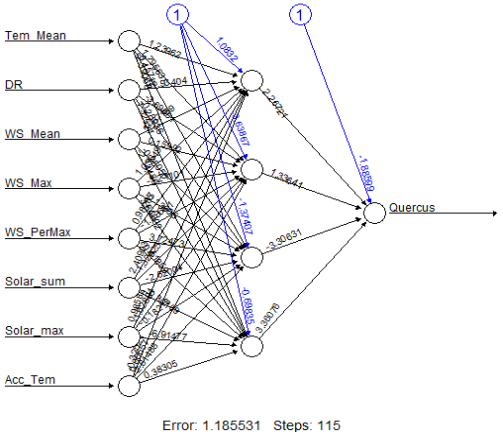
신경망모형 결과(hidden node 갯수 =4)



전체지역



서울지역

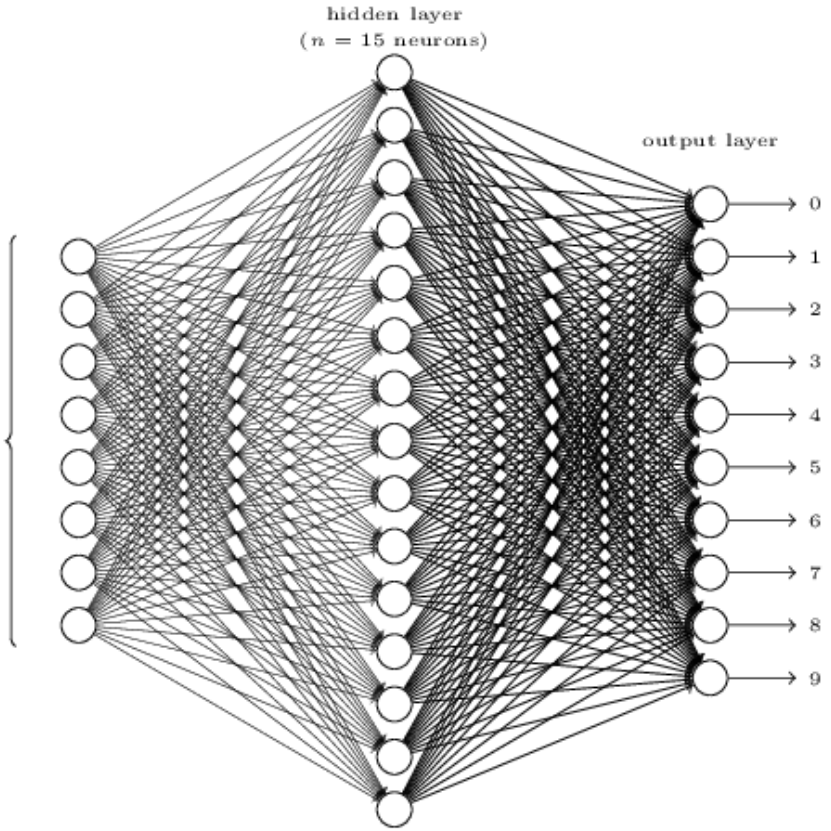


대전지역

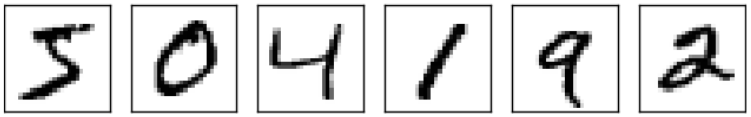
신경망 모형(Neural network model)

Using neural nets to recognize handwritten digits

Training :



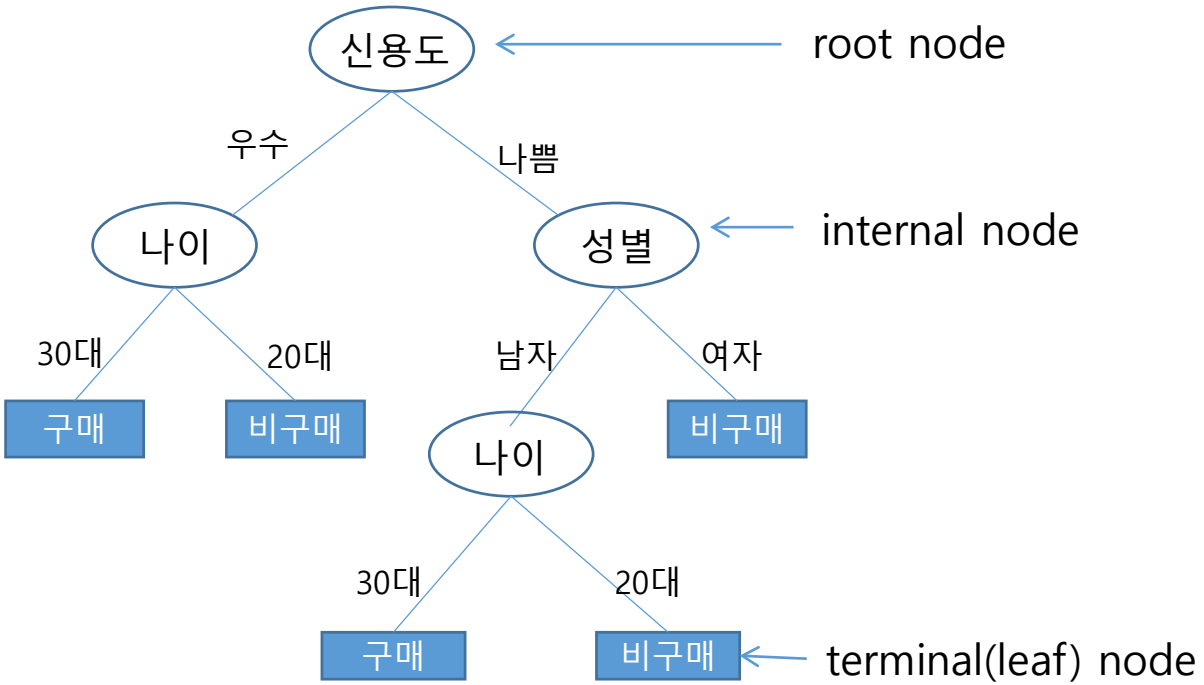
Test:



출처: http://neuralnetworksanddeeplearning.com/chap1.html

의사결정나무 (Decision trees)

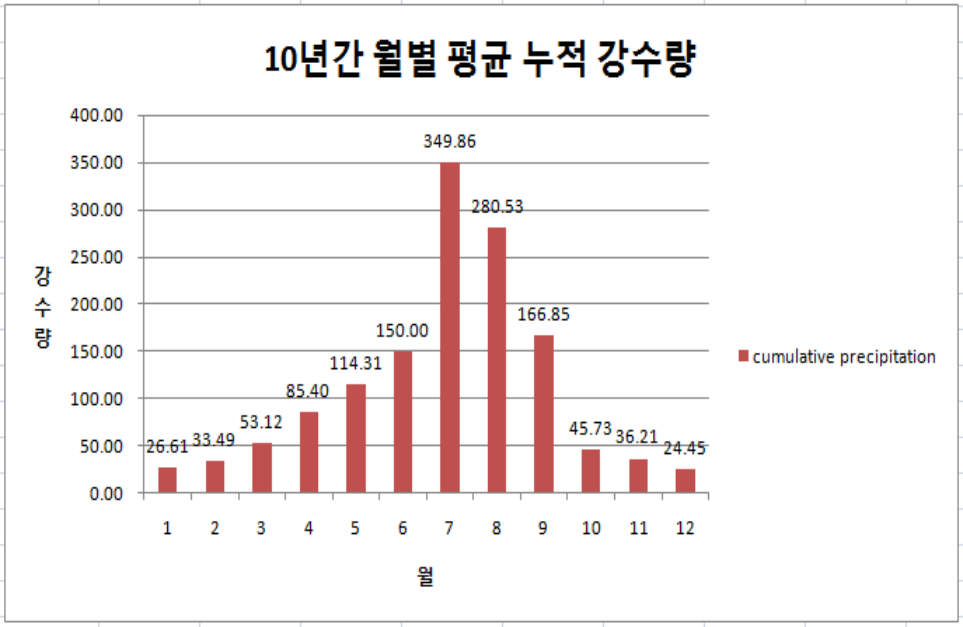
1	구매비구매 데이터			
2				
3	성별	나이	신용도	구매유무
4	여자	30대	나쁨	비구매
5	여자	20대	우수	비구매
6	남자	30대	나쁨	구매
7	남자	30대	우수	구매
8	여자	20대	나쁨	비구매
9	남자	20대	나쁨	비구매
10	남자	30대	나쁨	구매
11	여자	20대	우수	비구매
12	남자	30대	나쁨	구매
13	여자	30대	우수	구매
14	남자	20대	나쁨	비구매
15	남자	20대	우수	비구매
16	남자	20대	나쁨	비구매
17	여자	30대	나쁨	비구매
18	여자	30대	우수	구매
19	남자	30대	우수	구매
20	여자	20대	나쁨	비구매
21	여자	30대	우수	구매
22	남자	30대	나쁨	구매



의사결정나무 (Decision trees)

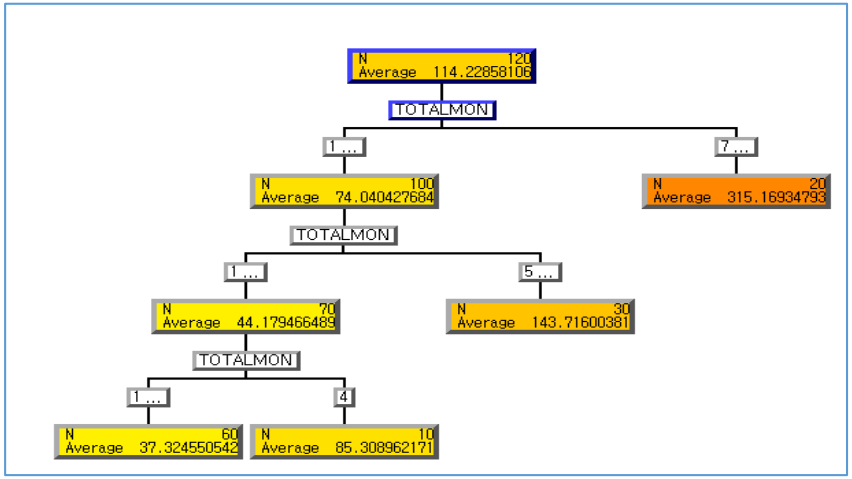
강수량 월별 그룹화(계절 구분)

자료 구성	
시간적	10년간(2001~2010) 월 평균 누적강수량
공간적	남한 전체 606지점(synop+aws)



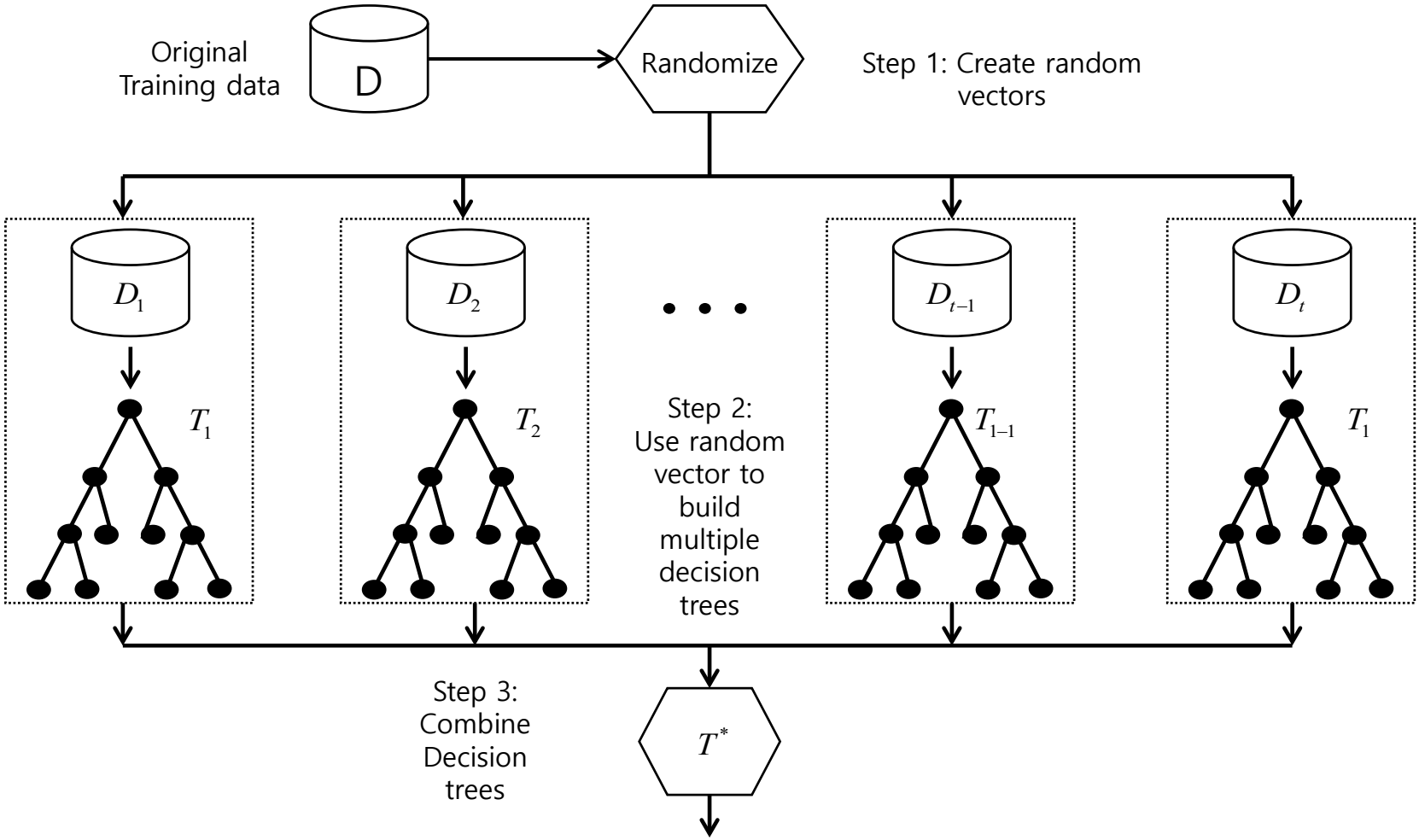
◆ 통계적 의사결정나무(Decision Tree) 기법 적용 하여 계절구분

- 제 1 계절 : 7월, 8월
- 제 2 계절 : 5월, 6월, 9월
- 제 3 계절 : 1월, 2월, 3월, 4월 ,10월, 11월, 12월



앙상블기법(Ensemble methods)

Bagging, Boosting, Random forest, etc.

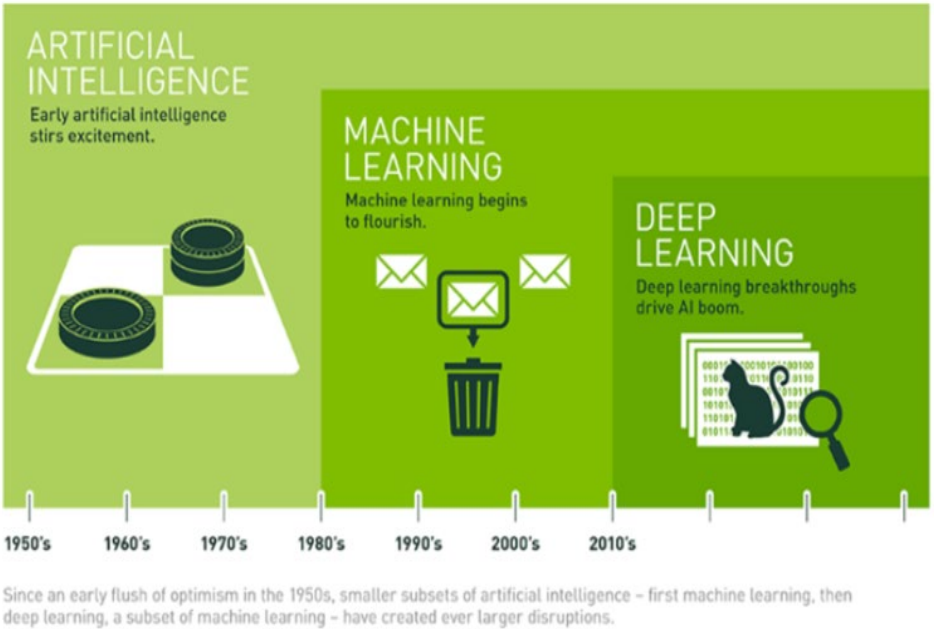


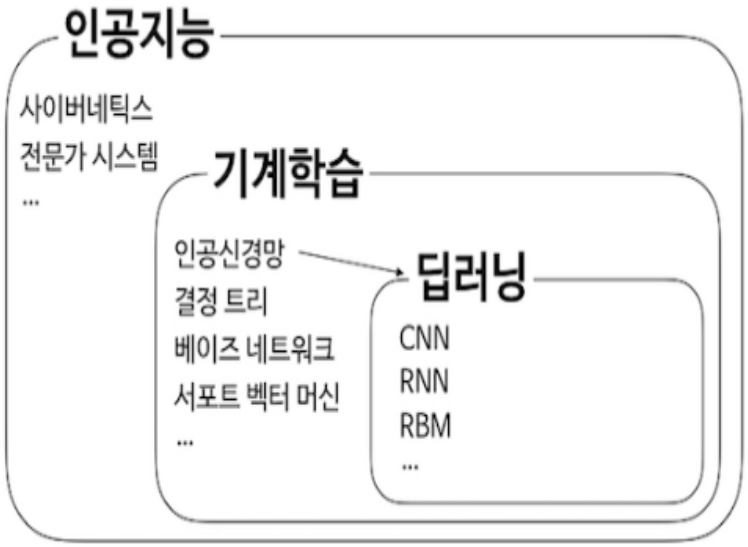
출처 : INTRODUCTION TO DATA MINING/ PANG-NING TAN, (2005)

신경망모형
(Neural Networks)

인공지능(AI), 머신러닝(기계학습, Machine Learning)), 딥러닝

- 인간의 지능을 컴퓨터로 구현하는 것이 인공지능이다.
- 이런 인공지능을 구현하기 위한 컴퓨터의 학습 방법이 머신 러닝이다.
- 딥러닝은 바로 머신 러닝을 실현하기 위한 기술인 것이다.





인공지능이 작동되기 위해서는 더 많고 더 좋은 데이터를 가질수록 더 좋은 결과를 가져오게 됨.
빅데이터를 단시간에 분석할 수 있는 AI 기술이 발전함
=> (빅)데이터는 21세기의 원유

출처: http://www.yoonsupchoi.com/2017/08/08/ai-medicine-4/
출처: http://betanews.heraldcorp.com:8080/article/708317.html (2017.06.15. 베타뉴스 기사)

인공지능(AI), 머신러닝(기계학습, Machine Learning)), 딥러닝

출처: http://m.dbguide.net/about.db?cmd=view&boardConfigUid=19&boardUid=190830

인공지능의 바둑에의 적용은 페이스북의 Darkforest가 앞섰으나, 구글의 AlphaGo가 이세돌에게 이긴 2016년 3월 같은 시기에 일본 명예기성인 고바야시 고이치에게 패배해 주목받지 못함.

- 인공지능은 알고리즘도중요하지만 이를 튜닝할 수 있는 데이터사이언티스트의 역할도 중요함.

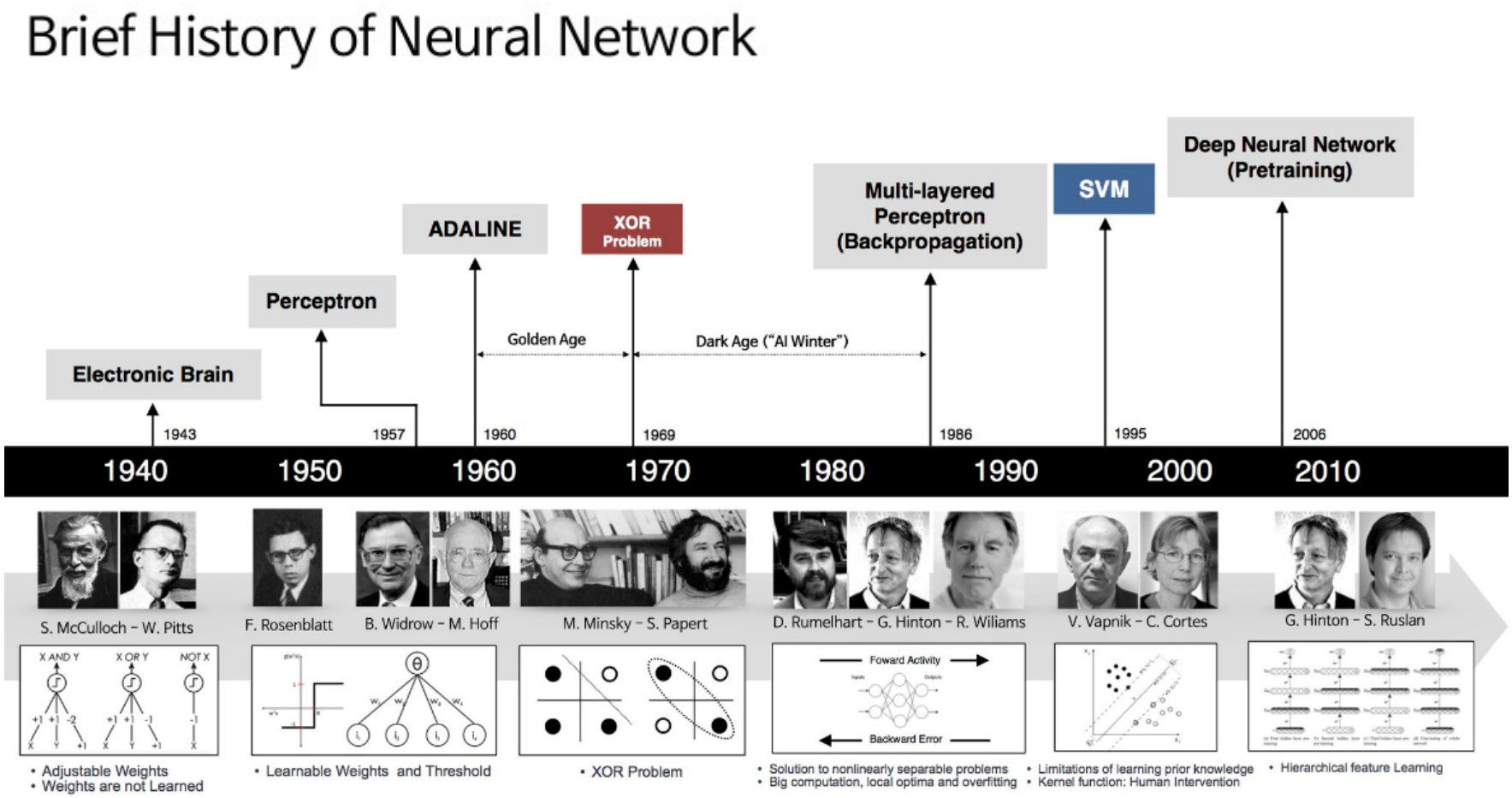
Darkforest와 AlphaGo의 비교

구 분	Darkforest (Facebook)	AlphaGo (Google)
착수 예측	▪ 인공신경망 구축 – 종류 : 컨볼루션 신경망 – 신경망 층 : 12개 ▪ 데이터 – 총 바둑 게임 25만개 – 학습(train) : 22만개 – 시험(test) : 3만개 – 정확도*: 57.1%	▪ 인공신경망 구축 – 종류 : 컨볼루션 신경망(지도학습, 강화학습) – 신경망 층 : 13개 ▪ 데이터 – 총 바둑 게임 16만개에서 3천만개 바둑게임 상태 추출 – 학습과 시험에는 바둑게임 상태가 사용됨 – 학습(train) : 2,900만개 – 시험(test) : 100만개 – 정확도 : 57.0% (지도학습)
탐색 알고리즘	▪ MTCS와 인공신경망의 동기화/비동기화 전략으로 구현	▪ MTCS의 정책과 가치 구현에 인공신경망 적용
성 능	아마추어 1 ~ 2단	프로 2 ~ 5단
계산 환경**	GPU : 44개 (CPU 소수)	CPU : 1,920개, GPU : 280개
기존 연구와 차별성	MTCS와 인공신경망의 동기화	강화학습, MTCS와 인공신경망의 결합

* 테스트 결과가 실제 결과와 얼마나 같은지를 나타내는 지표(i.e. 바둑 프로기사가 착수하는 패턴을 얼마나 학습했는지)이나, 이 정확도가 높다고 해서 바둑 게임에서의 승률이 반드시 높다는 보장은 할 수 없음

** 인공지능 바둑 프로그램을 구현할 때 사용된 최대 계산자원을 기준으로 함

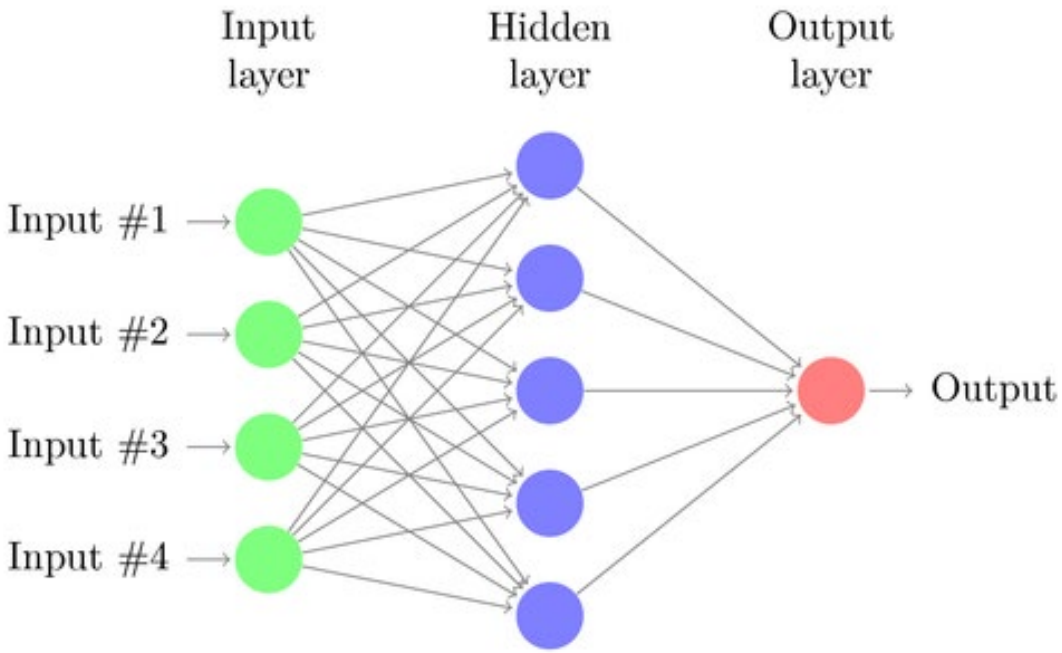
출처: 4차산업혁명시대의 금융업에서의 빅데이터, 이종석(신한카드 빅데이터 센터장), 국민미래포럼 2017.09



source: https://medium.com/ibm-data-science-experience/deep-learning-with-data-science-experience-8478cc0f81ac

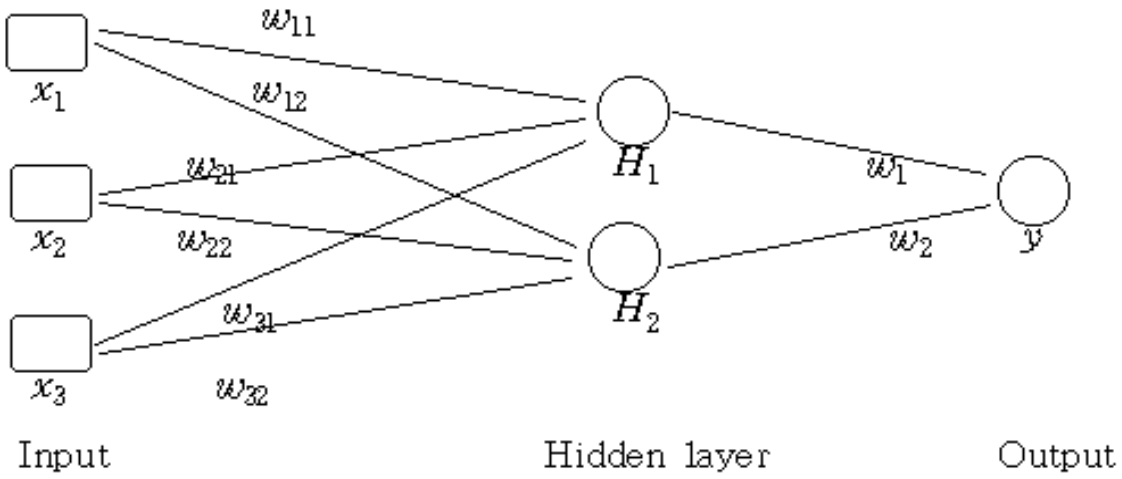
신경망모형 (Neural Networks) 개요

- 인간의 두뇌구조를 모방한 지도학습 방법
- 여러 개의 뉴런들이 상호 연결하여 입력에 상응하는 최적의 출력 값을 예측
- 장점: 좋은 예측력
- 단점: 해석의 어려움



출처: http://www.texample.net/tikz/examples/neural-network/

신경망모형 (Neural Networks) 구조



$$g_0^{-1}(E(y)) = w_0 + w_1 H_1 + w_2 H_2$$

$$\begin{cases} H_1 = g_1(w_{01} + w_{11}x_1 + w_{21}x_2 + w_{31}x_3) \\ H_2 = g_2(w_{02} + w_{12}x_1 + w_{22}x_2 + w_{32}x_3) \end{cases}$$

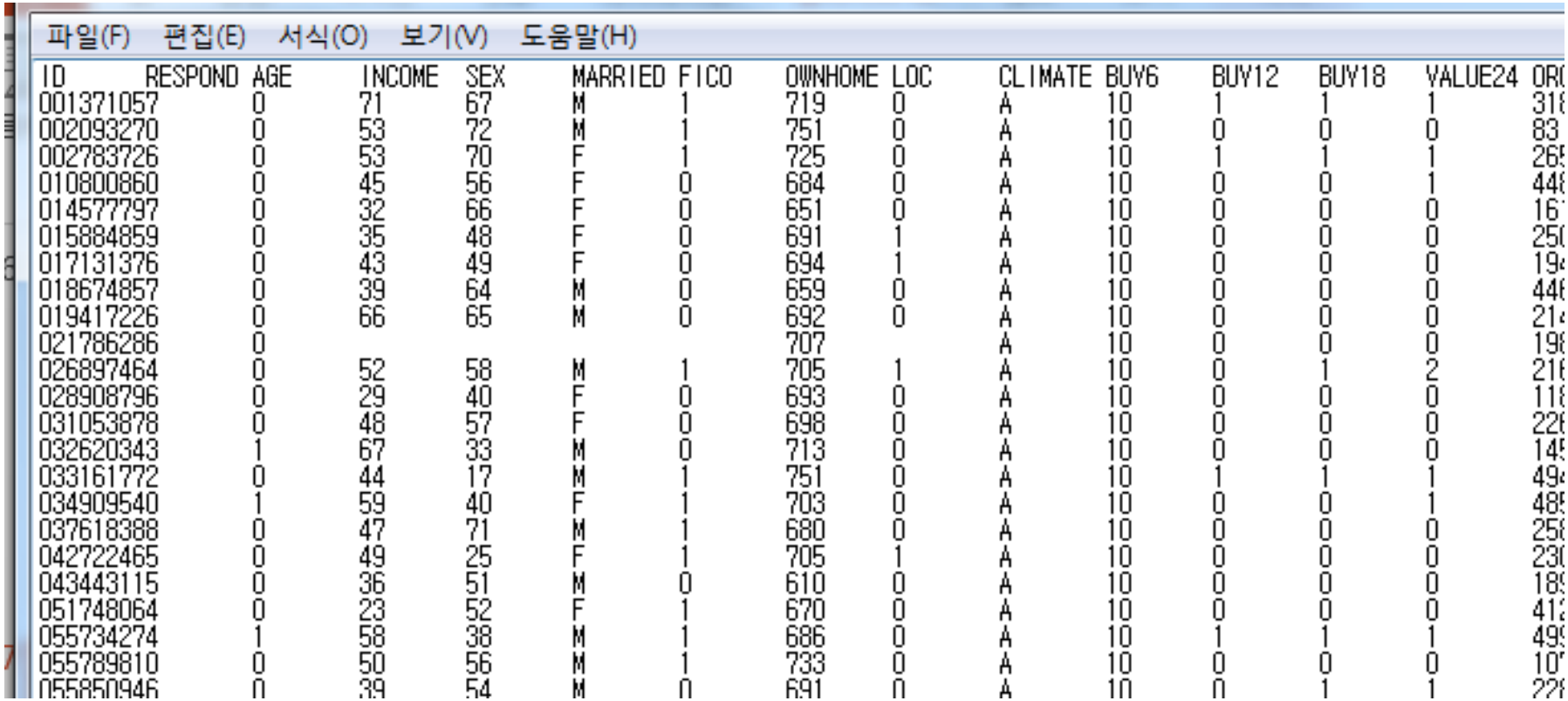
g_0^{-1} : Link function
$g_i(\cdot)$: activation function (e.g. the hyperbolic tangent(tanh), logistic, arctan, Elliott)
w_{ij} : weights from i node to j node
w_0, w_{01}, w_{02} : bias(intercept)
H_1, H_2 : hidden units

$g_i(\cdot)$: 활성화 함수. 생물학적 뉴런의 경우에는 마치 전구를 켜고 끄듯 (1 또는 0) 어떤 조건이 되면 뉴런이 활성화 됨. 그래서 활성화 함수라고 함.

Example: BUYTEST

범주	변수	변수의 내용
인구속성	Age	나이(년)
	Income	년수입(1000$)
	Married	1:결혼, 0:미혼
	Sex	F:여자, M:남자
	Ownhouse	1:소유, 0:미소유
지역속성	Loc	거주지 (A-H)
	Climate	거주지의 기온(10,20,30
거래회수	Buy6	최근 6개월 간의 구입회수
	Buy12	최근 12개월 간의 구입회수
	Buy18	최근 18개월 간의 구입회수
거래금액	Value24	지난 24개월 간의 구입총액
신용상태	Fico	신용점수
거래속성	Orgsrc	고객분류(C,D,I,O,P,R,U)
	Discbuy	할인고객 여부(1:할인고객, 0)
	Return24	지난 24개월간 상품의 반품여부 (1:반품, 0)
응답	Respond	광고메일에 응답여부(1:응답, 0)
	Purchtot	광고메일에 의한 구입총액
	C1-C7	광고메일에 의한 품목별 구입액
고객번호	ID	고객번호

R 을 이용한 신경망 모형 구축 예제



파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

ID	RESPOND	AGE	INCOME	SEX	MARRIED	FICO	OWNHOME	LOC	CLIMATE	BUY6	BUY12	BUY18	VALUE24	ORI
001371057	0	71	67	M	1	719	0	A	10	1	1	1		318
002093270	0	53	72	M	1	751	0	A	10	0	0	0		83
002783726	0	53	70	F	1	725	0	A	10	1	1	1		265
010800860	0	45	56	F	0	684	0	A	10	0	0	1		448
014577797	0	32	66	F	0	651	0	A	10	0	0	0		16
015884859	0	35	48	F	0	691	1	A	10	0	0	0		250
017131376	0	43	49	F	0	694	1	A	10	0	0	0		19
018674857	0	39	64	M	0	659	0	A	10	0	0	0		446
019417226	0	66	65	M	0	692	0	A	10	0	0	0		21
021786286	0					707		A	10	0	0	0		198
026897464	0	52	58	M	1	705	1	A	10	0	1	2		216
028908796	0	29	40	F	0	693	0	A	10	0	0	0		118
031053878	0	48	57	F	0	698	0	A	10	0	0	0		226
032620343	1	67	33	M	0	713	0	A	10	0	0	0		145
033161772	0	44	17	M	1	751	0	A	10	1	1	1		49
034909540	1	59	40	F	1	703	0	A	10	0	0	1		485
037618388	0	47	71	M	1	680	0	A	10	0	0	0		258
042722465	0	49	25	F	1	705	1	A	10	0	0	0		230
043443115	0	36	51	M	0	610	0	A	10	0	0	0		189
051748064	0	23	52	F	1	670	0	A	10	0	0	0		412
055734274	1	58	38	M	1	686	0	A	10	1	1	1		499
055789810	0	50	56	M	1	733	0	A	10	0	0	0		107
055850946	0	39	54	M	0	691	0	A	10	0	1	1		228

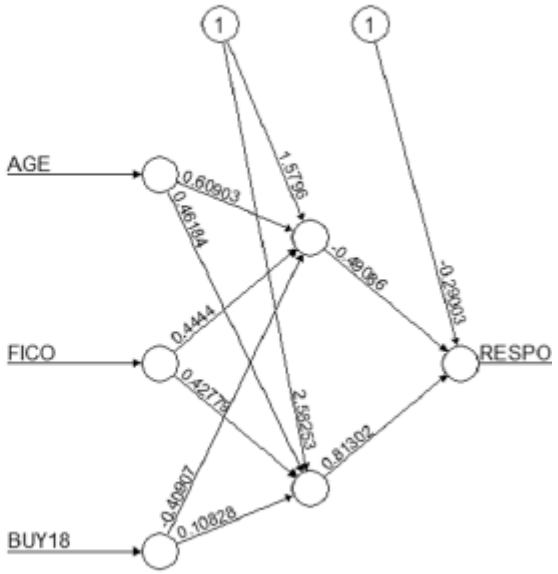
R 을 이용한 신경망 모형 구축 예제

```
> library(neuralnet)
>
> set.seed(123)
> nn1<-neuralnet(RESPOND~AGE+FICO+BUY18, data=buytest1, hidden=2,
+ stepmax = 1e+04, threshold = 0.05, act.fct='logistic')
> print(nn1)
Call: neuralnet(formula = RESPOND ~ AGE + FICO + BUY18, data = buytest1,

1 repetition was calculated.

        Error Reached Threshold Steps
1 331.9247771     0.04752704326  4580
```

```
> summary(nn1)
                   Length Class      Mode
call                    7 -none-     call
response             9727 -none-     numeric
covariate           29181 -none-     numeric
model.list              2 -none-     list
err.fct                 1 -none-     function
act.fct                 1 -none-     function
linear.output           1 -none-     logical
data                   26 data.frame list
net.result              1 -none-     list
weights                 1 -none-     list
startweights            1 -none-     list
generalized.weights     1 -none-     list
result.matrix          14 -none-     numeric
```

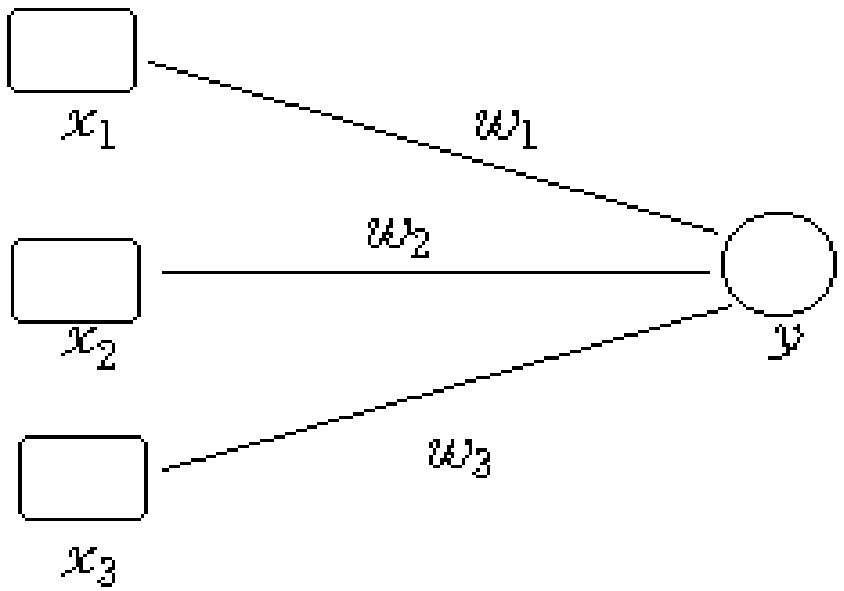
```
> print(nn1$weights)
[[1]]
[[1]][[1]]
                [,1]          [,2]
[1,]  1.5795954154 2.5825266021
[2,]  0.6090329379 0.4618394941
[3,]  0.4443999466 0.4277913738
[4,] -0.4090696043 0.1082817696

[[1]][[2]]
                [,1]
[1,] -0.2900289408
[2,] -0.4908575790
[3,]  0.8130214335
```

```
> head(nn1$net.result[[1]])
           [,1]
1 0.04113811544
2 0.03863306023
3 0.06011519497
4 0.10202768888
5 0.09328557101
6 0.07980124124
> plot(nn1)
```



신경망모형 (Neural Networks) 구조

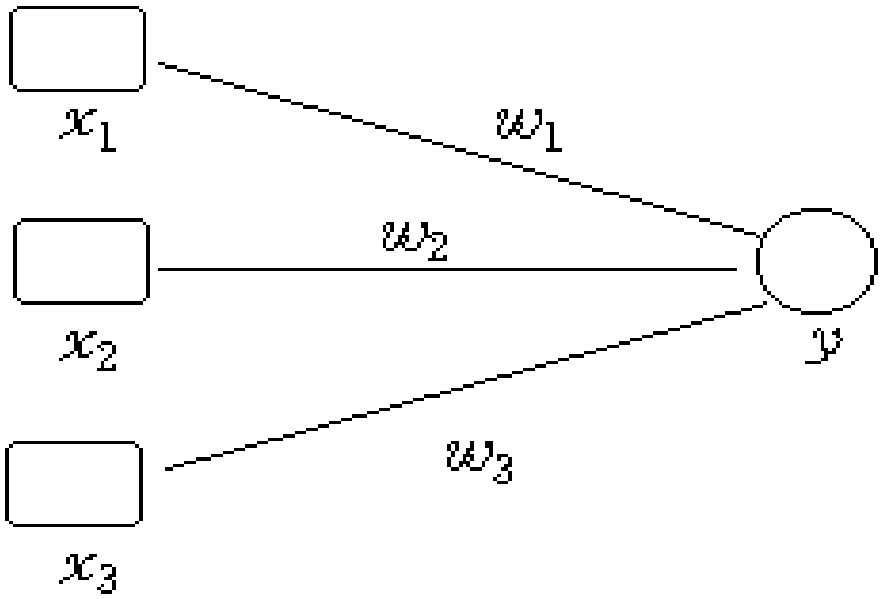
- 다중선형회귀모형 (Multiple Linear Regression)



$g_0^{-1}(E(Y|x)) = E(Y|x) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$

g_0^{-1} : 항등함수

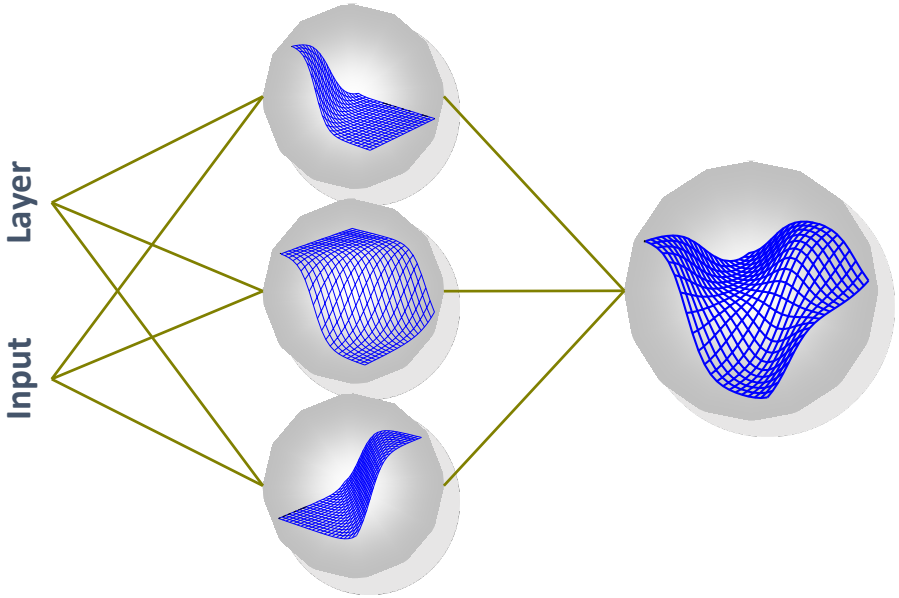
- 로지스틱 회귀모형 (Logistic Regression)



$$g_0^{-1}(E(Y|x)) = \ln\left(\frac{E(Y|x)}{1-E(Y|x)}\right) = w_0 + w_1x_1 + w_2x_2 + w_3x_3$$

g_0^{-1} : 로지스틱 함수

은닉층(hidden layer) 노드: 활성화 함수(Activation (Squashing) Function)

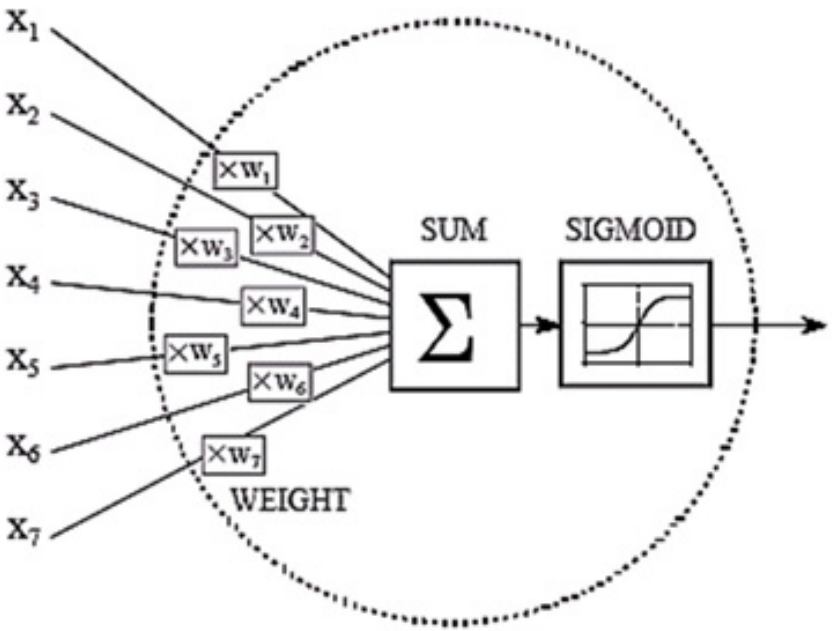


Activation Functions	
Name	**Formula**
Identity	$A(x) = x$
Sigmoid	$A(x) = \frac{1}{1+e^{-x}}$
Tanh	$A(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Step	$A(x) = \begin{matrix} -1 & if\ x < 0 \\ 1 & if\ x \geq 0 \end{matrix}$

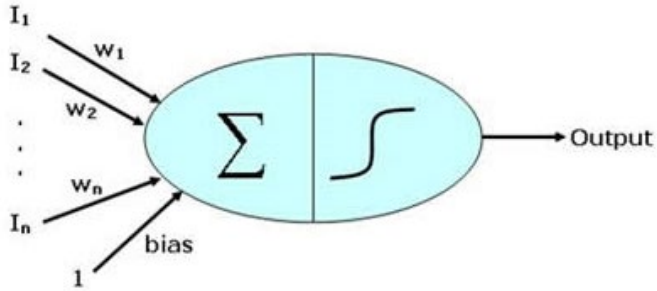
가장 많이 사용하는 activation function은 sigmoid(logistic) 함수와 hyperbolic tangent(Tanh) 함수이다.

간단한 함수: Linear(Identity), Threshold(Step) 함수
Gaussian function: Radial basis Function Network

은닉층(hidden layer) 노드: 활성화 함수(Activation (Squashing) Function)



출처:
http://futurehumanevolution.com/artificial-intelligence-future-human-evolution/artificial-neural-networks



Activation Functions	
Name	Formula
Identity	$A(x) = x$
Sigmoid	$A(x) = \frac{1}{1+e^{-x}}$
Tanh	$A(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Step	$A(x) = \begin{matrix} -1 & if\ x < 0 \\ 1 & if\ x \geq 0 \end{matrix}$

출처:
http://web.rememberingemil.org/Projects/NeuralNetworks.aspx.html

노드(node)는 두 부분으로 나누어 진다.
1. Combination 부분: 모든 input 변수들과 해당 weight(w_{ij})의 곱들의 합(SUM)으로 이루어진 부분. (위의 그림에서 SUM 부분)

SUM (또는 net) = $w_{01} + w_{11}x_1 + w_{21}x_2 + w_{31}x_{3+...}$

2. Activation function 부분: 위의 SUM의 결과에다가 다양한 activation 함수를 적용하여 output 값을 내는 부분.(위의 그림에서 SIGMOID 부분)

활성화 함수(Activation (Squashing) Function)

1. Sigmoid(Logistic) 함수:

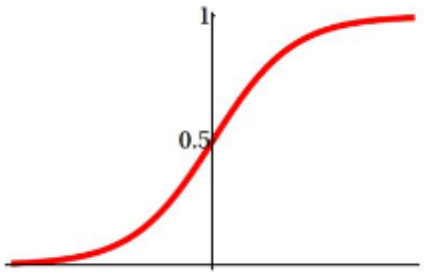
$$Sigmoid(x) = Logistic(x) = f(x) = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$$

이 함수의 결과값은 0과 1사이의 값을 가진다.

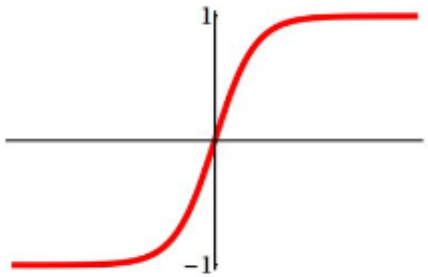
2. Tanh (hyperbolic tangent) 함수:

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

이 함수의 결과값은 -1과 1사이의 값을 가진다.

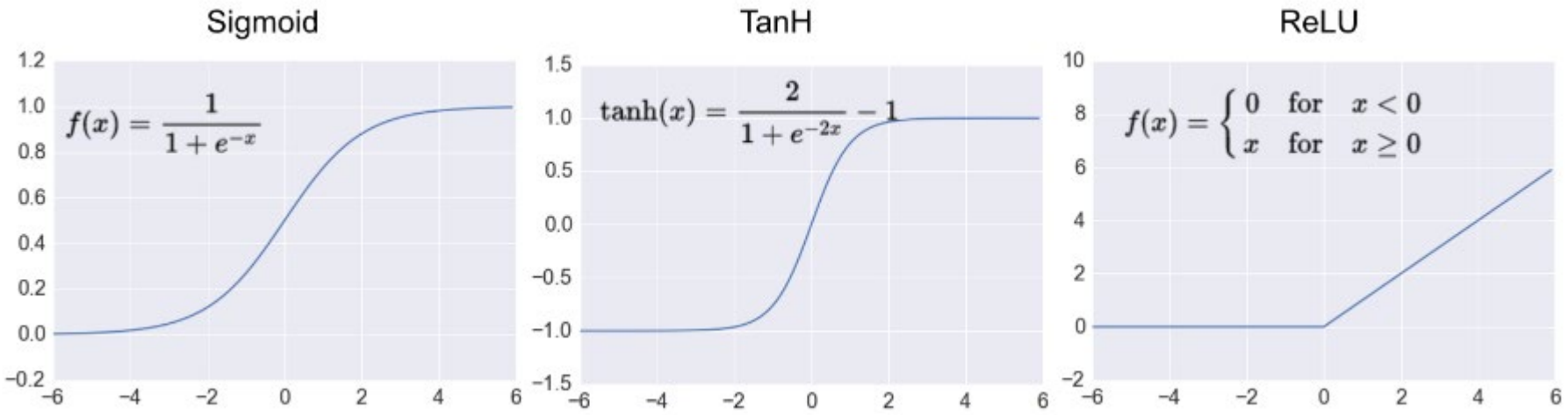


sigmoid

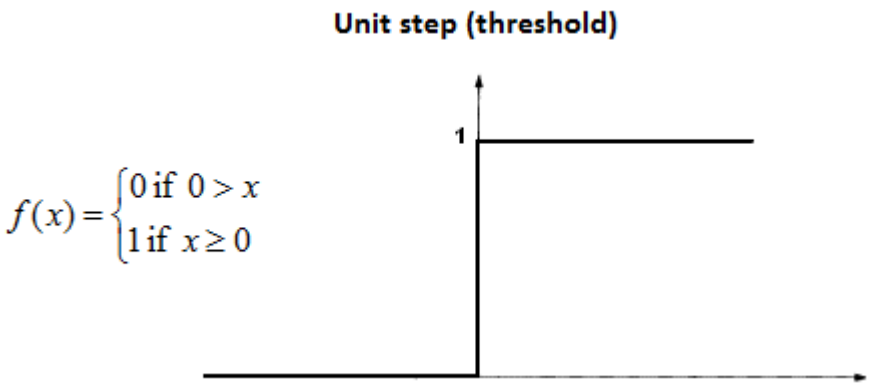


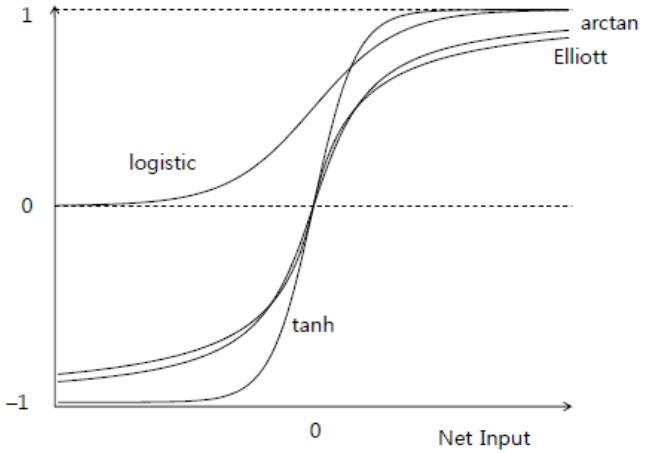
tanh

활성화 함수(Activation (Squashing) Function)

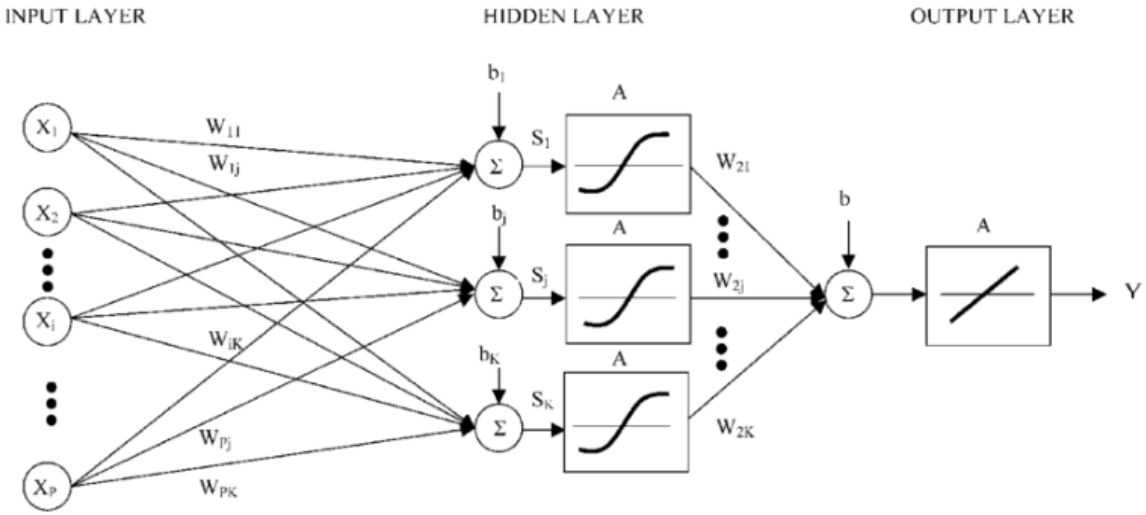


출처: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

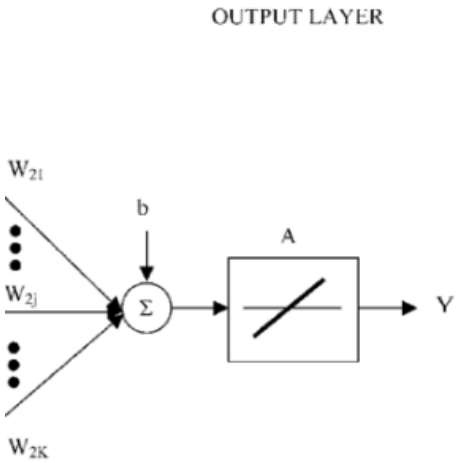




출력층(Output layers) 노드



Output Layers



For Regression: $Y=A(\sum) = \sum$ A: identity function

For classification:

$Y=A(\sum) = \frac{e^{\Sigma}}{\sum_{i=1}^{K} e^{\Sigma}}$ A: softmax function for Y=1,2,...,K levels

or

$Y=A(\sum) = \frac{1}{1+e^{-\Sigma}}$ A: logistic function function for Y=0,1 levels

통계 모형과 신경망 모형의 용어 차이

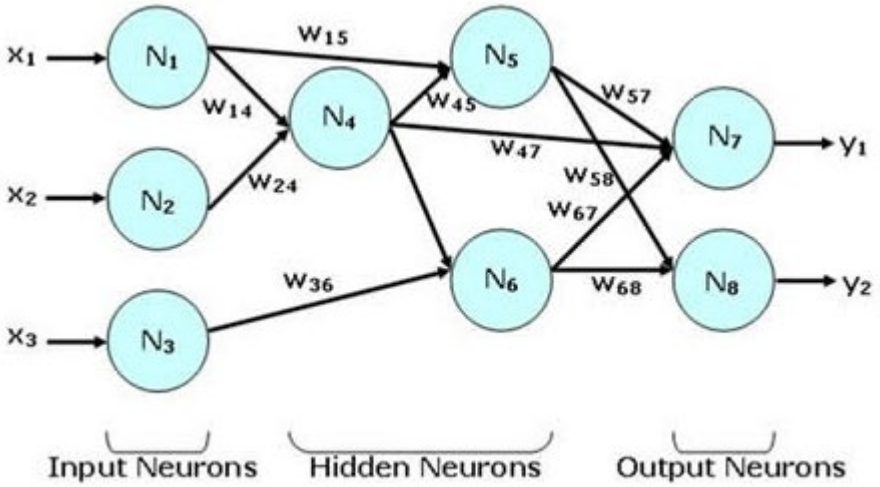
Statistical Model	Neural Netorks Model
variables	features
independent variables	inputs
predicted values	outputs
dependent variables	targets or training values
residuals	errors
estimation	raining, learning, adaptation, or self-organization.
an estimation criterion	an error function, cost function, or Lyapunov function

통계 모형과 신경망 모형의 용어 차이

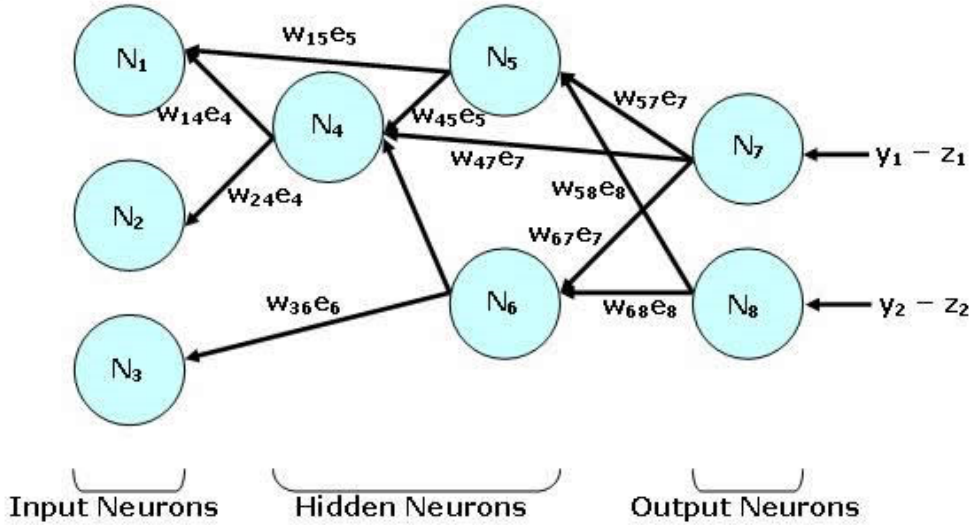
Statistical Model	Neural Netorks Model
observations	patterns or training pairs
parameter estimates	(synaptic) weights
interactions	higher-order neurons
transformations	functional links
regression and discriminant analysis	supervised learning or heteroassociation
data reduction	unsupervised learning, encoding, or autoassociation
cluster analysis	competitive learning or adaptive vector quantization
interpolation and extrapolation	generalization

신경망모형 (Neural Networks)

< Feed-forward Neural Networks >

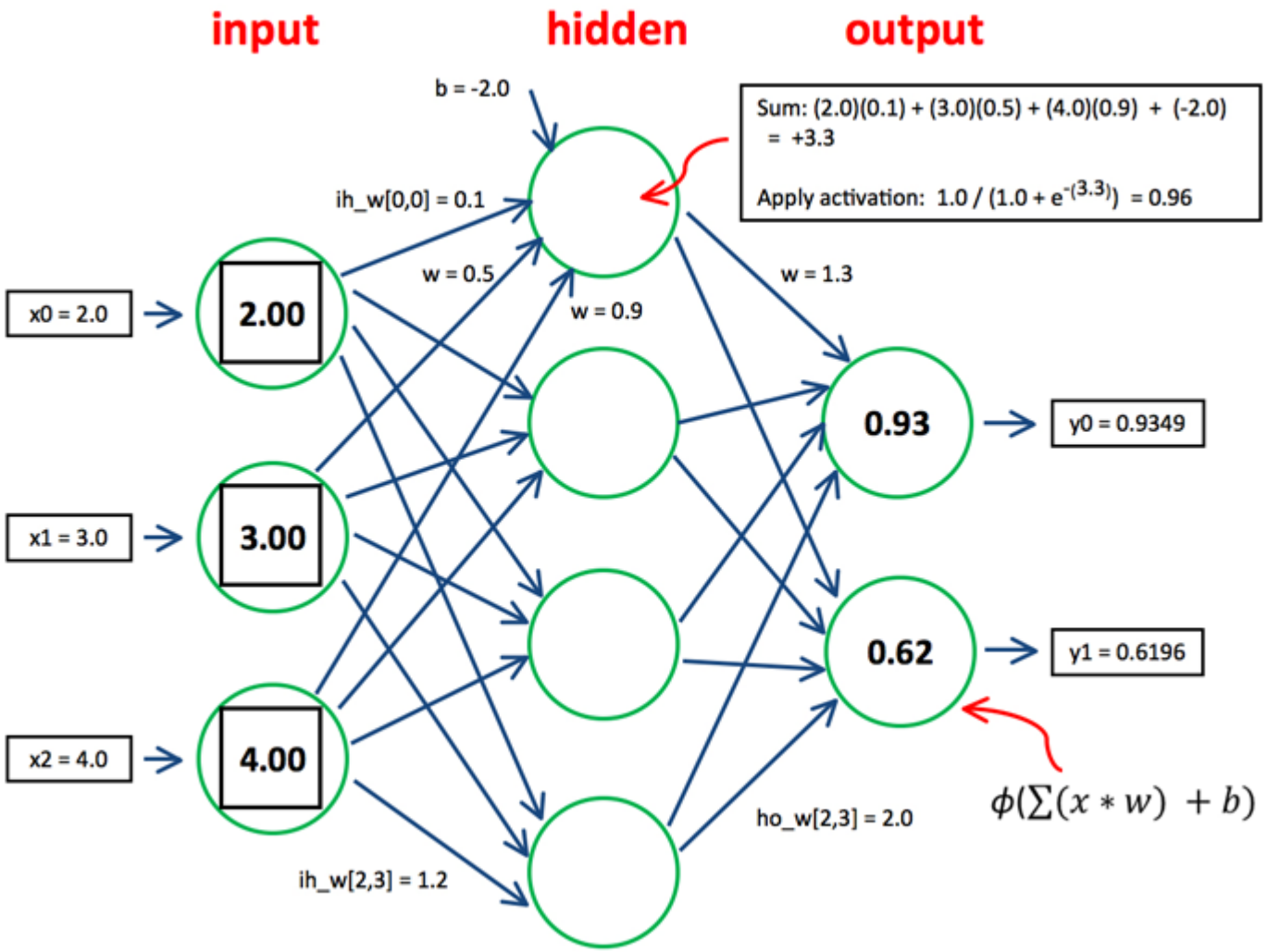


< Back-propagation Learning >



출처: http://web.rememberingemil.org/Projects/NeuralNetworks.aspx.html

신경망모형 (Neural Networks)

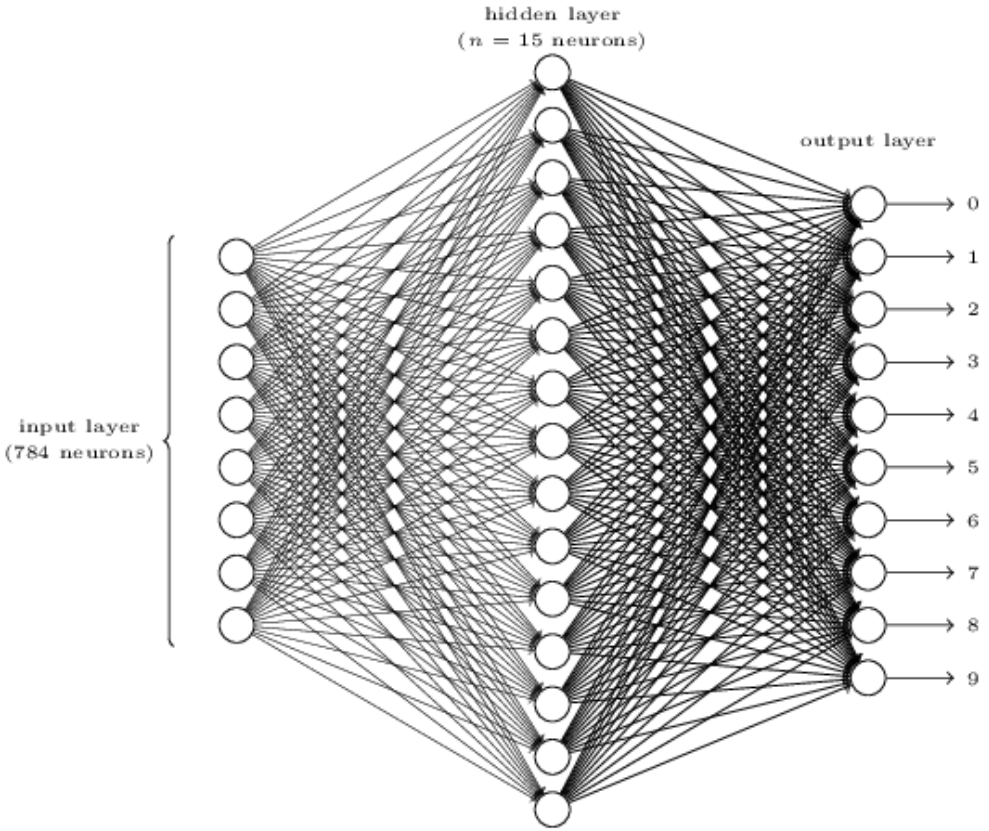


출처:
https://visualstudiomagazine.com/articles/2013/05/01/~/media/ECG/visualstudiomagazine/Images/2013/05/0513vsm_McCaffreyNeuralNet2.ashx

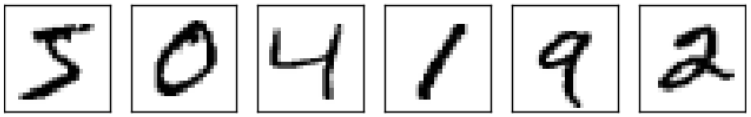
신경망모형 (Neural Networks)

Using neural nets to recognize handwritten digits

Training :



Test:



출처: http://neuralnetworksanddeeplearning.com/chap1.html

2.2 단순선형회귀모형

단순선형 회귀모형: $Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i, \quad i = 1, 2, \cdots, n$

i : 전체 n개의 관측값 중 i번째 값을 나타내는 첨자

ε_i : 평균이 0, 분산이 σ^2인 오차를 나타내는 확률변수.

ε_i 들은 확률적으로 서로 독립

X_i : 값이 주어진 상수

==> Y_i는 $X = X_i$로 주어질 때 평균이 $\beta_0 + \beta_1 X_i$이고, 분산이 σ^2인 확률변수

Y_i들 역시 독립

기울기 β_1 : 설명변수 X가 한 단위 증가할 때 움직이는 Y의 평균값의 변화량

출처: 회귀분석(제2판), 김충락, 강근석 지음, 교우사 (2010)

2.3 회귀계수의 추정

2.3.1 최소제곱법

표본: $(X_1, Y_1), \cdots, (X_n, Y_n)$

최소제곱법(method of least squares) : 오차의 크기를 "전체적"으로 작게 하는 방법.

- 각 오차의 제곱의 합을 최소로 하는 회귀식을 구하는 방법
- 독일의 수학자 가우스(Carl Friedrich Gauss)에 의해 사용됨.

최소제곱법 :

관측치 Y_i와 모집단회귀식 $\beta_0 + \beta_1 X_i$와의 차이인 오차 ε_i들의 제곱의 합이 최소가 되도록 회귀계수를 추정하는 방법.

즉,

$$S = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2$$

을 최소화하는 β_0와 β_1의 값을 구한다.

출처: 회귀분석(제2판), 김충락, 강근석 지음, 교우사 (2010)

회귀분석 복습

제곱합의 β_0와 β_1에 대한 편미분:

$$\frac{\partial S}{\partial \beta_0} = (-2)\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i) = 0$$

$$\frac{\partial S}{\partial \beta_1} = (-2)\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i)X_i = 0$$

정규방정식(normal equation)

$$nb_0 + b_1\sum_{i=1}^{n} X_i = \sum_{i=1}^{n} Y_i$$

$$b_0\sum_{i=1}^{n} X_i + b_1\sum_{i=1}^{n} X_i^2 = \sum_{i=1}^{n} X_i Y_i$$

최소제곱추정량(least squares estimator):

$$b_1 = \frac{S_{XY}}{S_{XX}}$$

$$b_0 = \overline{Y} - b_1\overline{X}$$

위에서

$$S_{XY} = \sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})$$

$$S_{XX} = \sum_{i=1}^{n}(X_i - \overline{X})^2$$

b_0와 b_1의 의미: 표본회귀식의 절편과 기울기

b_1 : 설명변수 X가 한 단위 변화할 때 반응변수 Y의 평균변화량

- **b_0와 b_1은 X와 Y변수가 갖는 단위들과 함께 해석되어야 한다**

출처: 회귀분석(제2판), 김충락, 강근석 지음, 교우사 (2010)

3.1 중선형회귀모형

3.1.1 모집단 중회귀모형

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \beta_{p-1} X_{i,p-1} + \varepsilon_i , \quad i = 1, 2, \cdots, n$$

$\varepsilon_i \sim iid \ N(0, \sigma^2)$

X_{ij} = j번째 설명변수 X_j의 i번째 관측치

β_j = Y와 X_j간의 기울기. 다른 설명변수들의 값들이 고정되었을 때

3.2 회귀계수의 추정

3.2.1 최소제곱법

오차제곱합 : $S = \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_{i1} - \cdots - \beta_{p-1} X_{i,p-1})^2$

$$\frac{\partial S}{\partial \beta_0} = (-2) \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_{i1} - \cdots - \beta_{p-1} X_{i,p-1}) = 0$$

$$\frac{\partial S}{\partial \beta_j} = (-2) \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_{i1} - \cdots - \beta_{p-1} X_{i,p-1}) X_{ij} = 0 , \quad j = 1, 2, \cdots, p-1$$

출처: 회귀분석(제2판), 김충락, 강근석 지음, 교우사 (2010)

회귀분석 복습

$b_0, b_1, \cdots, b_{p-1}$: β_j들의 최소제곱추정량

정규방정식: p차원 연립방정식

$$nb_0 + b_1\sum_{i=1}^{n} X_{i1} + \cdots + b_{p-1}\sum_{i=1}^{n} X_{i,p-1} = \sum_{i=1}^{n} Y_i$$
$$b_0\sum_{i=1}^{n} X_{ij} + b_1\sum_{i=1}^{n} X_{i1}X_{ij} + \cdots + b_{p-1}\sum_{i=1}^{n} X_{i,p-1}X_{ij} = \sum_{i=1}^{n} X_{ij}Y_i, \quad j=1,2,\cdots,p-1$$

추정된 회귀식: $\hat{Y}_i = b_0 + b_1X_{i1} + \cdots + b_{p-1}X_{i,p-1}$

잔차: $e_i = Y_i - \hat{Y}_i = Y_i - (b_0 + b_1X_{i1} + \cdots + b_{p-1}X_{i,p-1})$

행렬과 벡터를 이용한 유도

오차제곱합: $S = \sum_{i=1}^{n}\varepsilon_i^2 = \boldsymbol{\varepsilon}'\boldsymbol{\varepsilon} = (\mathbf{y}-\mathbf{X}\boldsymbol{\beta})'(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})$

$$\frac{\partial S}{\partial \boldsymbol{\beta}} = -2\mathbf{X}'\mathbf{y} + 2(\mathbf{X}'\mathbf{X})\boldsymbol{\beta}$$ (p.471 부록 A.7.2 참조)

정규방정식: $(\mathbf{X}'\mathbf{X})\mathbf{b} = \mathbf{X}'\mathbf{y}$

최소제곱추정량: $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$: 추정량 b 가 y 들의 선형 함수임.

$\mathbf{X}'\mathbf{X}$ 의 역행렬을 구하기 위해 QR 분해법 등의 계산 방법 이용.

출처: 회귀분석(제2판), 김충락, 강근석 지음, 교우사 (2010)

적합된 반응벡터: $\hat{y} = Xb$

단순회귀의 경우:

$$X'X = \begin{bmatrix} n & \sum_{i=1}^{n} X_i \\ \sum_{i=1}^{n} X_i & \sum_{i=1}^{n} X_i^2 \end{bmatrix}, \quad X'y = \begin{bmatrix} \sum_{i=1}^{n} Y_i \\ \sum_{i=1}^{n} X_i Y_i \end{bmatrix}$$

$$(X'X)^{-1} = \frac{1}{n\sum_{i=1}^{n} X_i^2 - (\sum_{i=1}^{n} X_i)^2} \begin{bmatrix} \sum_{i=1}^{n} X_i^2 & -\sum_{i=1}^{n} X_i \\ -\sum_{i=1}^{n} X_i & n \end{bmatrix}$$

==> $(X'X)^{-1}X'y$ = 식(2.5) p.51

추정된 회귀식: $\hat{Y}_i = x_i'b$

잔차: $e_i = Y_i - x_i'b$

잔차벡터(residual vector) : $e = y - \hat{y} = y - Xb$

여기에서 $\hat{y} = (\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_n)$, $e = (e_1, e_2, \ldots, e_n)$, Xb 는 기대함수 $X\beta$ 에서 β 대신 추정량 b 를 대입한 것.

출처: 회귀분석(제2판), 김충락, 강근석 지음, 교우사 (2010)

Gradient Descent algorithm 이해하기

Lets take the example of predicting the price of a new price from housing data:

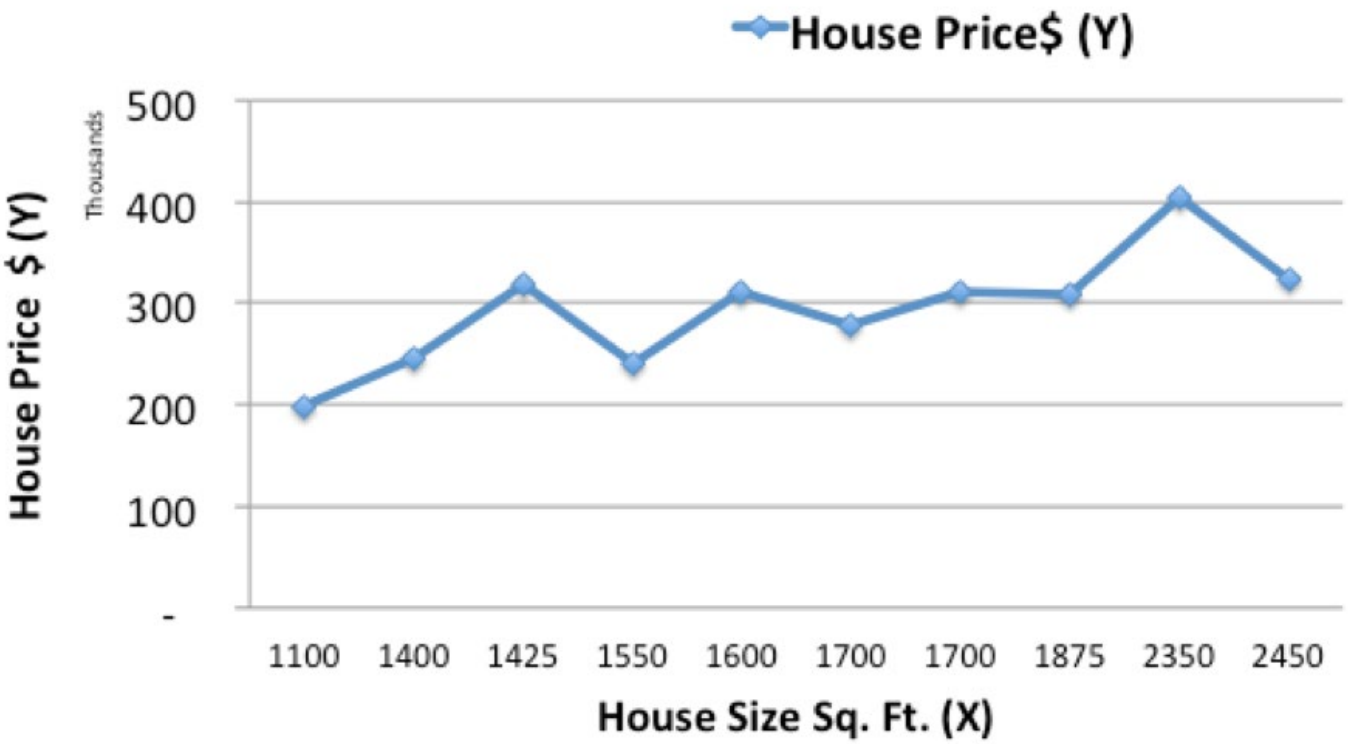
Now, given **historical housing data,** the task is to create a model that predicts the price of a new house given the house size.

House Size sq.ft (X)	1400	1600	1700	1875	1100	1550	2350	2450	1425	1700
House Price$ (Y)	245,000	312,000	279,000	308,000	199,000	219,000	405,000	324,000	319,000	255,000

The task – for a new house, given its size (X), what will its price (Y) be?

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

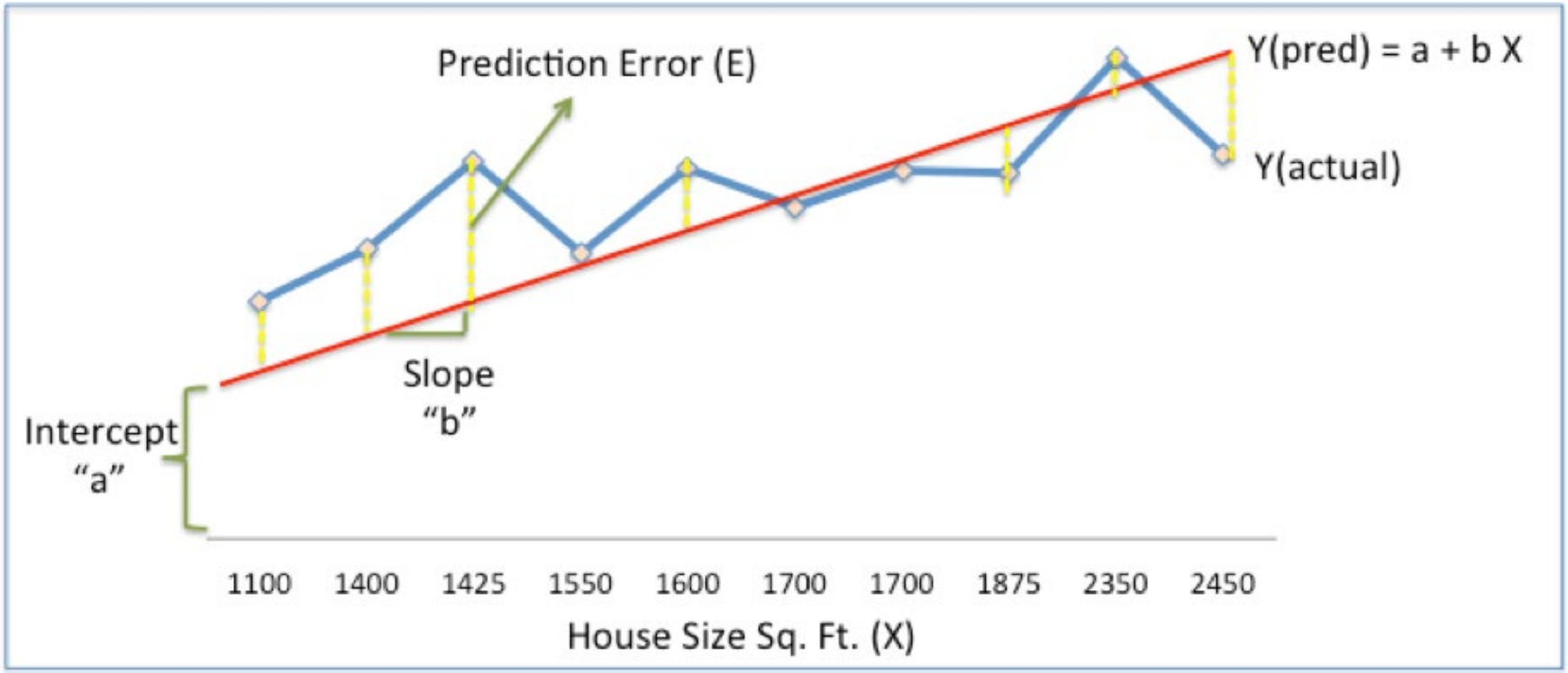
Lets start off by plotting the historical housing data:



출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

Now, we will use a simple linear model, where we fit a line on the historical data, to predict the price of a new house (Ypred) given its size(X)



출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

In the above chart, the red line gives the predicted house price (Ypred) given house size (X).
Ypred = a+bX

The blue line gives the actual house prices from historical data (Yactual)
The difference between Yactual and Ypred (given by the yellow dashed lines) is the prediction error (E)

So, we need to find a line with optimal values of a,b (called weights) that best fits the historical data by reducing the prediction error and improving prediction accuracy.

So, our objective is to find optimal **a, b** that minimizes the error between actual and predicted values of house price (1/2 is for mathematical convenience since it helps in calculating gradients in calculus)

Sum of Squared Errors (SSE) = ½ Sum (Actual House Price – Predicted House Price)^2
= ½ Sum(Y – Ypred)^2

(Please note that there are other measures of Error. SSE is just one of them.)

This is where Gradient Descent comes into the picture. Gradient descent is an optimization algorithm that finds the optimal weights (a,b) that reduces prediction error.

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

Lets now go step by step to understand the **Gradient Descent algorithm:**

Step 1: Initialize the weights(a & b) with random values and calculate Error (SSE)

Step 2: Calculate the gradient i.e. change in SSE when the weights (a & b) are changed by a very small value from their original randomly initialized value. This helps us move the values of a & b in the direction in which SSE is minimized.

Step 3: Adjust the weights with the gradients to reach the optimal values where SSE is minimized

Step 4: Use the new weights for prediction and to calculate the new SSE

Step 5: Repeat steps 2 and 3 till further adjustments to weights doesn't significantly reduce the Error

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

We will now go through each of the steps in detail (I worked out the steps in excel, which I have pasted below). But before that, we have to standardize the data as it makes the optimization process faster.

HOUSING DATA	
House Size (X)	House Price (Y)
1,100	1,99,000
1,400	2,45,000
1,425	3,19,000
1,550	2,40,000
1,600	3,12,000
1,700	2,79,000
1,700	3,10,000
1,875	3,08,000
2,350	4,05,000
2,450	3,24,000

Min-Max Standardization	
X (X-Min/Max-min)	Y (Y-Min/Max-Min)
0.00	0.00
0.22	0.22
0.24	0.58
0.33	0.20
0.37	0.55
0.44	0.39
0.44	0.54
0.57	0.53
0.93	1.00
1.00	0.61

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

Step 1: To fit a line Ypred = a + b X, start off with random values of a and b and calculate prediction error (SSE)

a	b	X	Y	YP=a+bX	SSE=1/2(Y-YP)^2
0.45	0.75	0.00	0.00	0.45	0.101
		0.22	0.22	0.62	0.077
		0.24	0.58	0.63	0.001
		0.33	0.20	0.70	0.125
		0.37	0.55	0.73	0.016
		0.44	0.39	0.78	0.078
		0.44	0.54	0.78	0.030
		0.57	0.53	0.88	0.062
		0.93	1.00	1.14	0.010
		1.00	0.61	1.20	0.176
				Total SSE	**0.677**

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Step 2: Calculate the error gradient w.r.t the weights

$\partial SSE/\partial a = -(Y-YP)$
$\partial SSE/\partial b = -(Y-YP)X$

Here, $SSE=\frac{1}{2}(Y-YP)^2 = \frac{1}{2}(Y-(a+bX))^2$
You need to know a bit of calculus, but that's about it!!

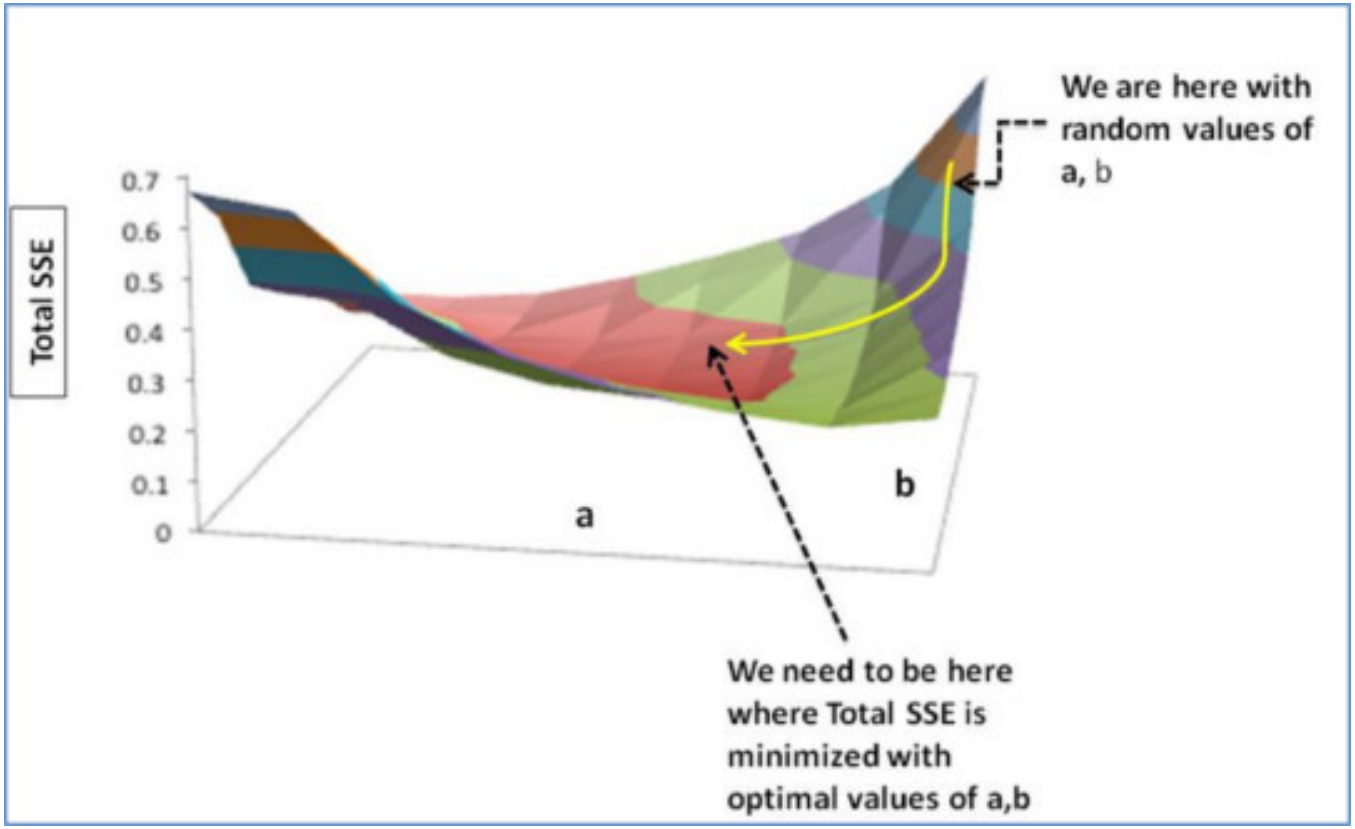
$\partial SSE/\partial a$ and $\partial SSE/\partial b$ are the **gradients** and they give the direction of the movement of a,b w.r.t to SSE.

a	b	X	Y	YP=a+bX	SSE		∂SSE/∂a = -(Y-YP)	∂SSE/∂b = -(Y-YP)X
0.45	0.75	0.00	0.00	0.45	0.101		0.45	0.00
		0.22	0.22	0.62	0.077		0.39	0.09
		0.24	0.58	0.63	0.001		0.05	0.01
		0.33	0.20	0.70	0.125		0.50	0.17
		0.37	0.55	0.73	0.016		0.18	0.07
		0.44	0.39	0.78	0.078		0.39	0.18
		0.44	0.54	0.78	0.030		0.24	0.11
		0.57	0.53	0.88	0.062		0.35	0.20
		0.93	1.00	1.14	0.010		0.14	0.13
		1.00	0.61	1.20	0.176		0.59	0.59
					Total SSE	**0.677**	**Sum**	**3.300**	**1.545**

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

Step 3: Adjust the weights with the gradients to reach the optimal values where SSE is minimized



출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

We need to update the random values of a,b so that we move in the direction of optimal a, b.

Update rules:

• a – ∂SSE/∂a

• b – ∂SSE/∂b

So, update rules:

1. New a = a – r * **∂SSE/∂a =** 0.45-0.01*3.300 = 0.42

2. New b = b – r * **∂SSE/∂b=** 0.75-0.01*1.545 = 0.73

here, r is the learning rate = 0.01, which is the pace of adjustment to the weights.

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Gradient Descent algorithm 이해하기

Step 4: Use new a and b for prediction and to calculate new Total SSE

a	b	X	Y	YP=a+bX		SSE		∂SSE/∂a	∂SSE/∂b
0.42	0.73	0.00	0.00	0.42		0.087		0.42	0.00
		0.22	0.22	0.58		0.064		0.36	0.08
		0.24	0.58	0.59		0.000		0.01	0.00
		0.33	0.20	0.66		0.107		0.46	0.15
		0.37	0.55	0.69		0.010		0.14	0.05
		0.44	0.39	0.74		0.063		0.36	0.16
		0.44	0.54	0.74		0.021		0.20	0.09
		0.57	0.53	0.84		0.048		0.31	0.18
		0.93	1.00	1.10		0.005		0.10	0.09
		1.00	0.61	1.15		0.148		0.54	0.54
					Total SSE	**0.553**	**Sum**	**2.900**	**1.350**

You can see with the new prediction, the total SSE has gone down (0.677 to 0.553). That means prediction accuracy has improved.

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

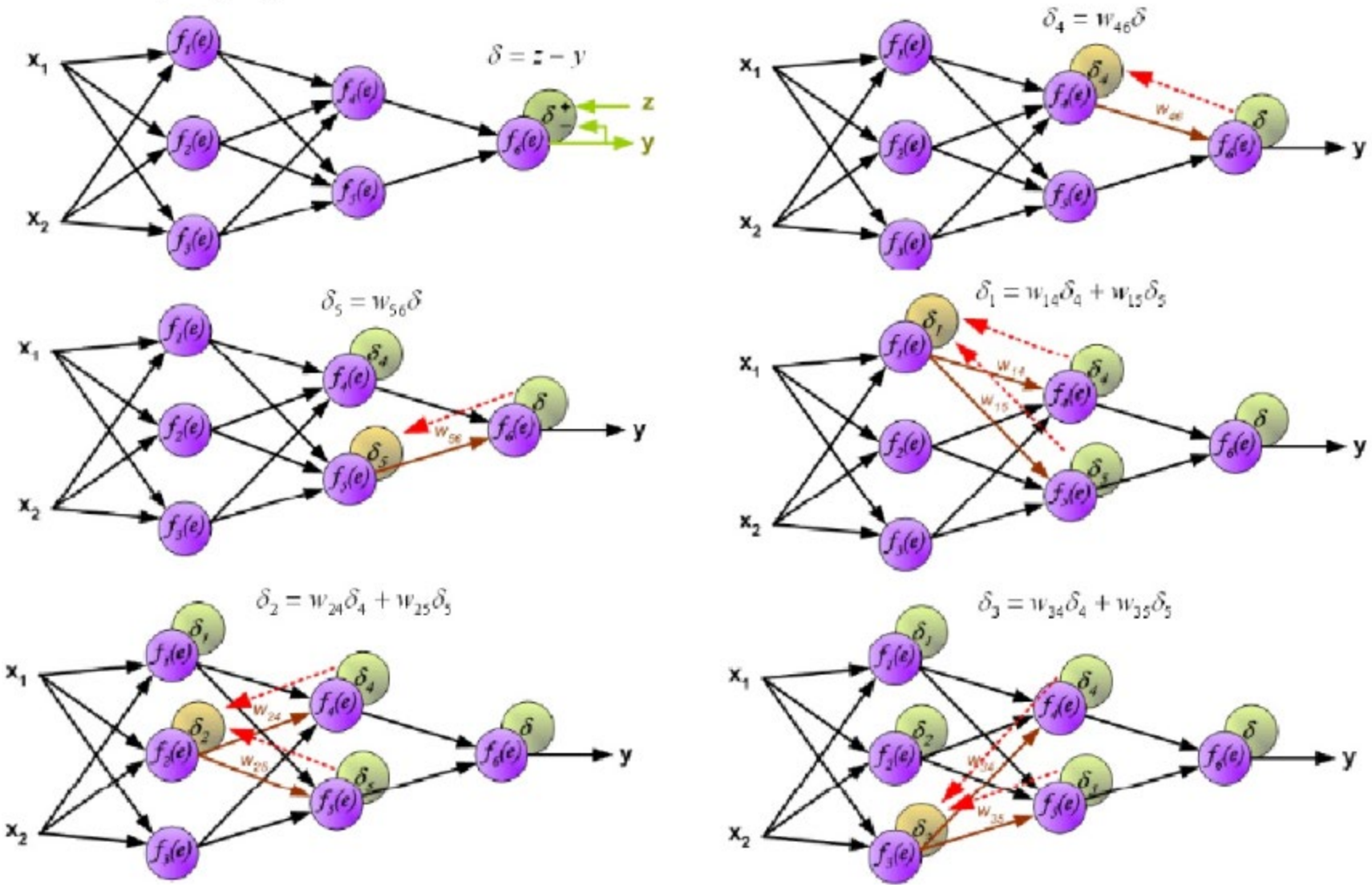
Gradient Descent algorithm 이해하기

Step 5: Repeat step 3 and 4 till the time further adjustments to a, b doesn't significantly reduces the error. At that time, we have arrived at the optimal a,b with the highest prediction accuracy.

This is the Gradient Descent Algorithm. This optimization algorithm and its variants form the core of many machine learning algorithms like Neural Networks and even Deep Learning.

출처: http://www.kdnuggets.com/2017/04/simple-understand-gradient-descent-algorithm.html
Keep it simple! How to understand Gradient Descent algorithm by Jahnavi Mahanta

Back-propagation Learning: Gradient Descent algorithm



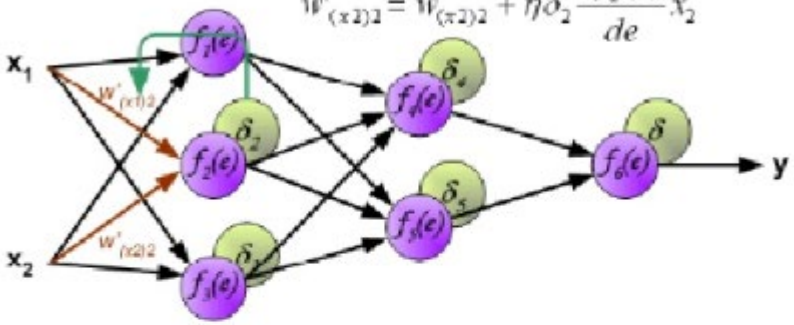
출처: http://www.slideshare.net/fullscreen/aorriols/lecture11-neural-networks/4

Back-propagation Learning: Gradient Descent algorithm

3.bis. Do the same of each neuron

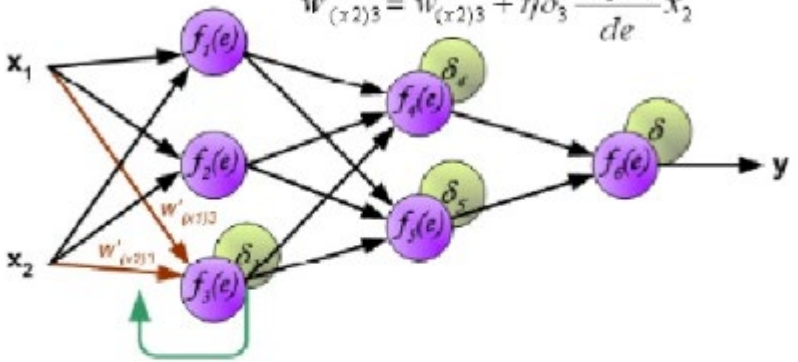
$$w'_{(x1)2} = w_{(x1)2} + \eta\delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta\delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta\delta_3 \frac{df_3(e)}{de} x_1$$

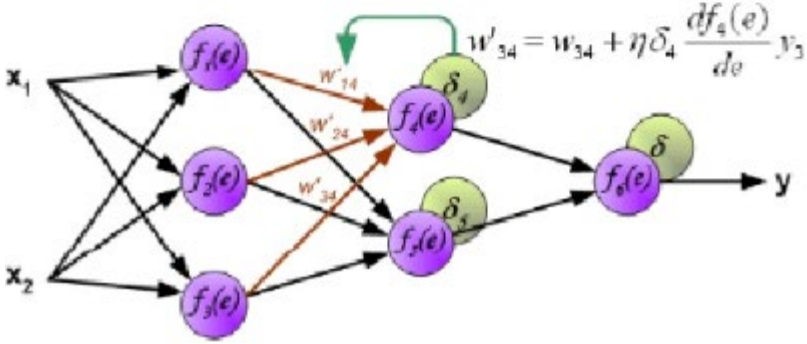
$$w'_{(x2)3} = w_{(x2)3} + \eta\delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

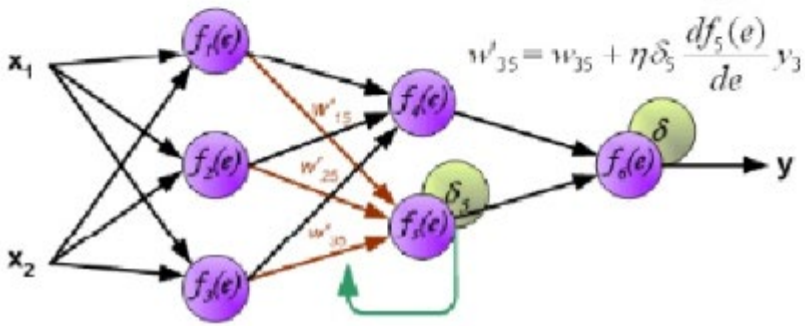
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$



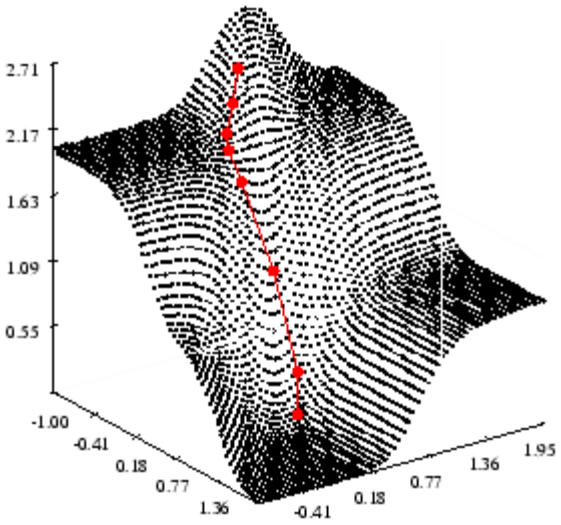
출처: http://www.slideshare.net/fullscreen/aorriols/lecture11-neural-networks/4

Back-propagation Learning: Gradient Descent algorithm

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

Error Surface: Learning weights



출처: http://fedc.wiwi.hu-berlin.de/fedc_homepage/xplore/tutorials/sfehtmlnode97.html

Backpropagation Example

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Overview

For this tutorial, we're going to use a neural network with two inputs, two hidden neurons, two output neurons. Additionally, the hidden and output neurons will include a bias.

Here's the basic structure:

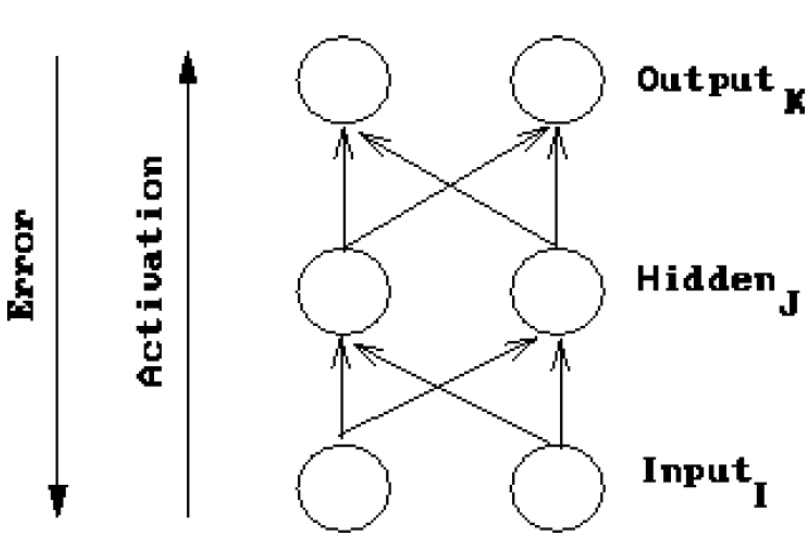
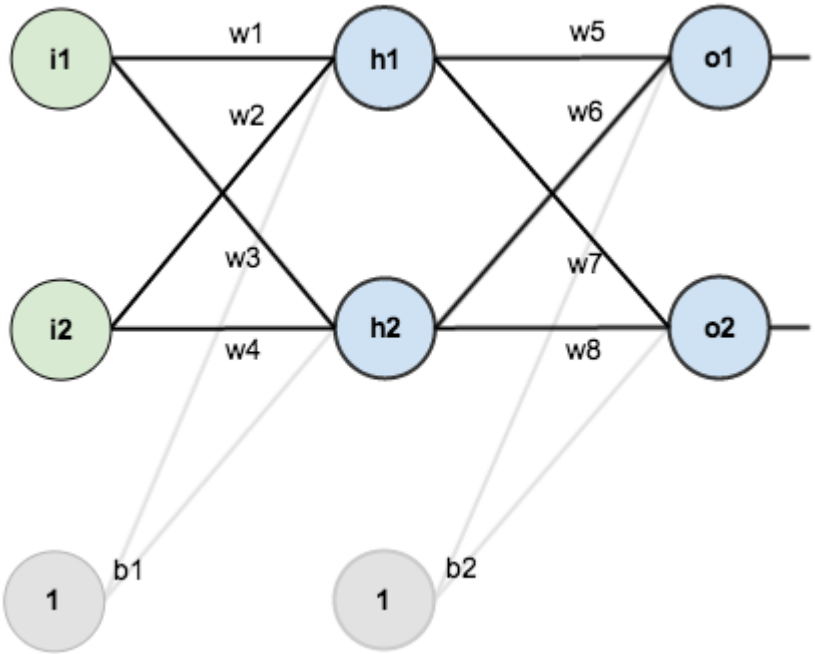
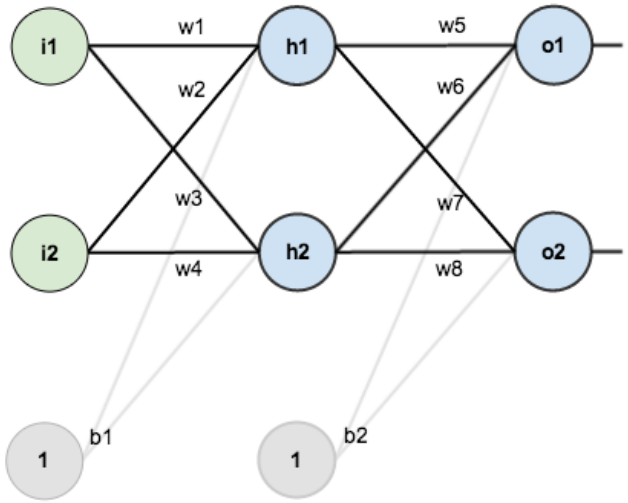


Figure: Neural network processing

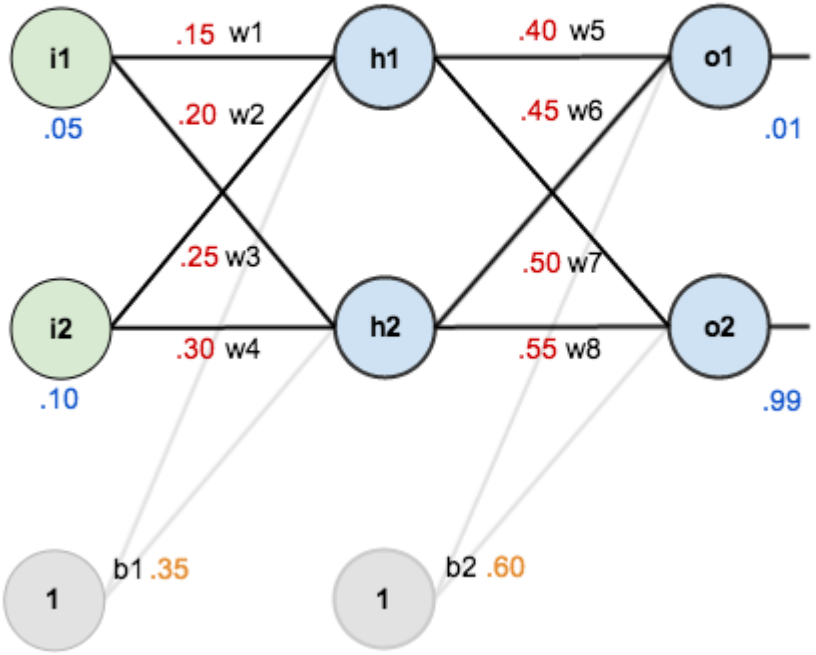


출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example



In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



a training example:

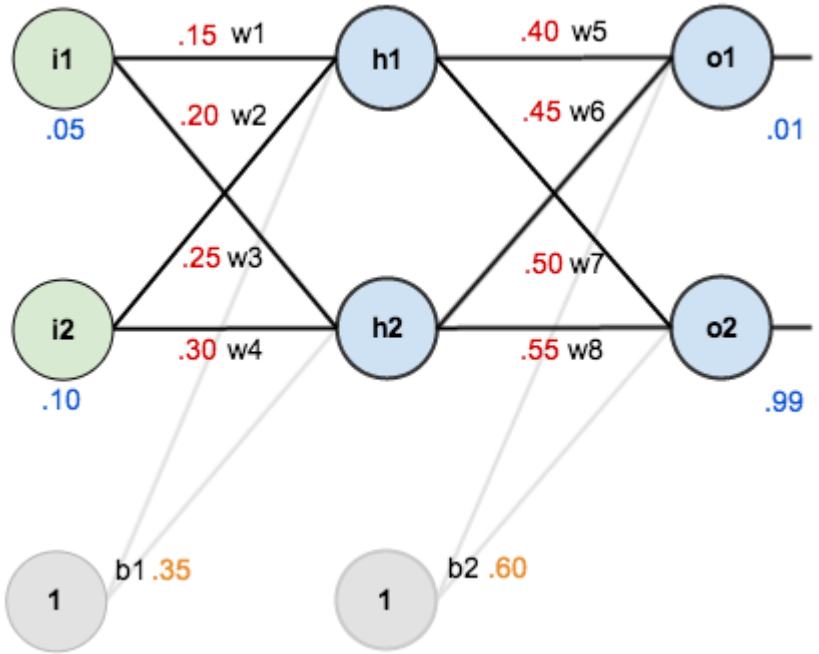
i1(x1)	i2(x2)	o1(y1)	o2(y2)
0.05	0.10	0.01	0.99

The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

For the rest of this tutorial we're going to work with a single training set: given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.

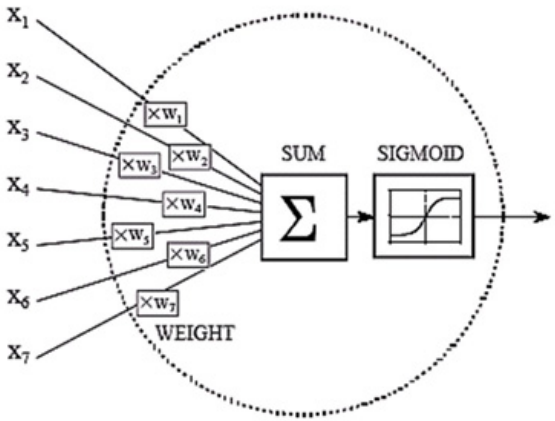
출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

The Forward Pass

To begin, lets see what the neural network currently predicts given the weights and biases above and inputs of 0.05 and 0.10. To do this we'll feed those inputs forward though the network.

We figure out the *total net input* to each hidden layer neuron, *squash* the total net input using an *activation function* (here we use the *logistic function*), then repeat the process with the output layer neurons.

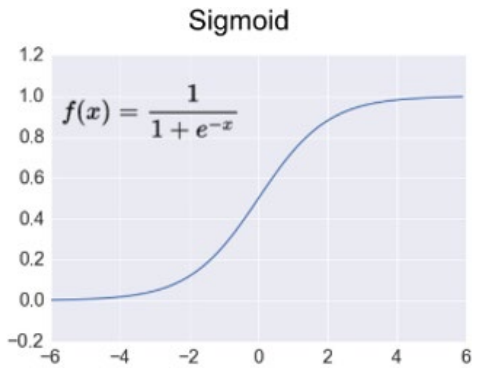
Total net input is also referred to as just *net input* by some sources.



활성화 함수 (activation function): Sigmoid(Logistic) 함수:

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

이 함수의 결과값은 0과 1사이의 값을 가진다.



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

a training example:

i1(x1)	i2(x2)	o1(y1)	o2(y2)
0.05	0.10	0.01	0.99

Here's how we calculate the total net input for h_1 :

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

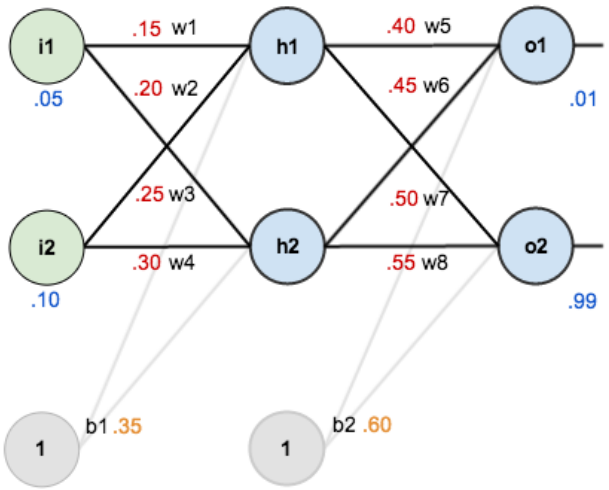
We then squash it using the logistic function to get th output of h_1 :

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

Carrying out the same process for h_2 we get:

$$out_{h2} = 0.596884378$$

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

a training example:

i1(x1)	i2(x2)	o1(y1)	o2(y2)
0.05	0.10	0.01	0.99

Here's the output for o1 :

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

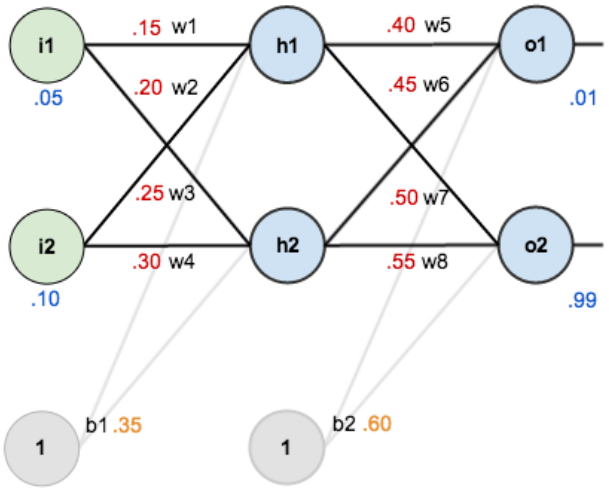
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

And carrying out the same process for o2 we get:

$$out_{o2} = 0.772928465$$

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Calculating the Total Error

We can now calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Some sources refer to the target as the *ideal* and the output as the *actual*.

The 1/2 is included so that exponent is cancelled when we differentiate later on.
The result is eventually multiplied by a learning rate anyway so it doesn't matter that we introduce a constant here .

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

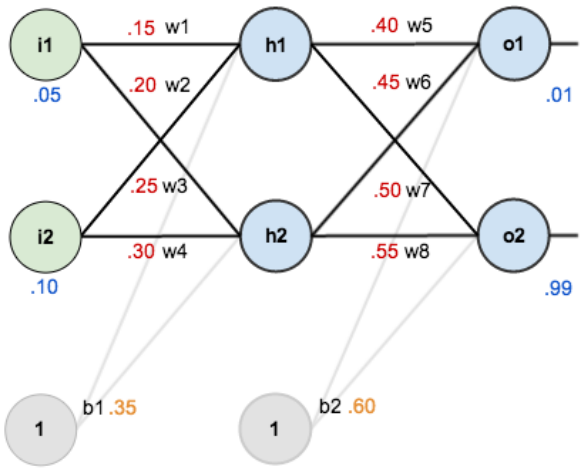
a training example:

i1(x1)	i2(x2)	o1(y1)	o2(y2)
0.05	0.10	0.01	0.99

For example, the target output for o1 is 0.01 but the neural network output 0.75136507, therefore its error is:

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



Repeating this process for o2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

The Backwards Pass

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Output Layer

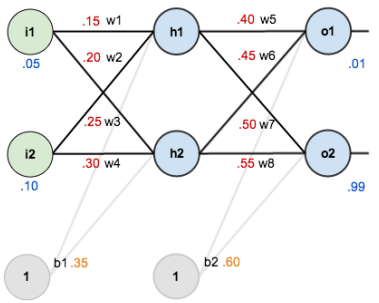
Consider w_5 . We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$

$\frac{\partial E_{total}}{\partial w_5}$ is read as "the partial derivative of E_{total} with respect to w_5". You can also say "the gradient with respect to w_5".

By applying the chain rule we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:

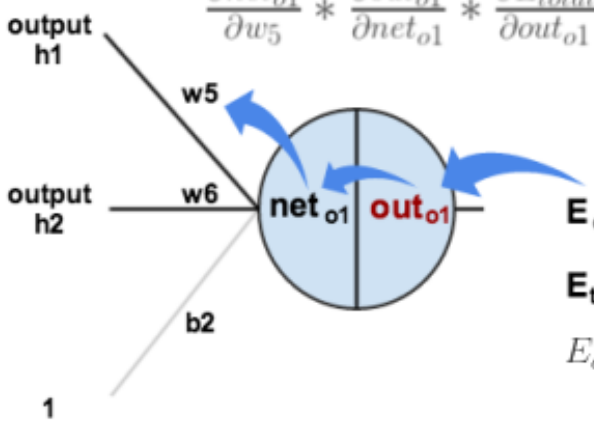


출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

Visually, here's what we're doing:

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$E_{o1} = ½(target_{o1} - out_{o1})^2$

$E_{total} = E_{o1} + E_{o2}$

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

We need to figure out each piece in this equation.

First, how much does the total error change with respect to the output?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

$-(target - out)$ is sometimes expressed as $out - target$

When we take the partial derivative of the total error with respect to out_{o1}, the quantity $\frac{1}{2}(target_{o2} - out_{o2})^2$ becomes zero because out_{o1} does not affect it which means we're taking the derivative of a constant which is zero.

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Next, how much does the output of o1 change with respect to its total net input?

The partial derivative of the logistic function is the output multiplied by 1 minus the output:

Note: sigmoid(logistic) 함수 미분

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

$$\frac{\partial f(x)}{\partial x} = (-1)(1+e^{-x})^{-2}e^{-x}(-1) = \frac{e^{-x}}{(1+e^{-x})^2}$$
$$= \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} = \frac{1}{(1+e^{-x})}\left(1 - \frac{1}{(1+e^{-x})}\right)$$
$$= f(x)\big(1-f(x)\big)$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1-out_{o1}) = 0.75136507(1-0.75136507) = 0.186815602$$

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Finally, how much does the total net input of o1 change with respect to w_5 ?

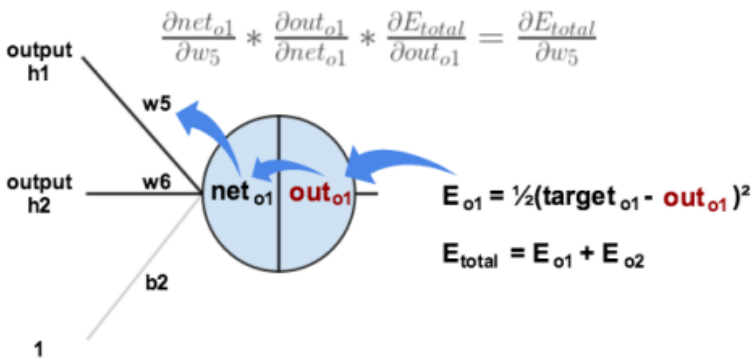
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

You'll often see this calculation combined in the form of the delta rule:

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

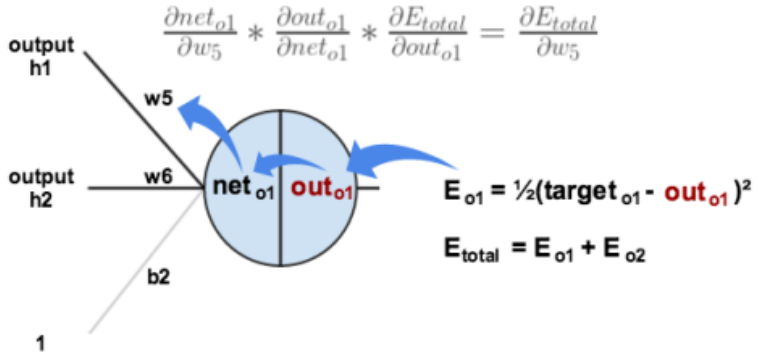
$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

To decrease the error, we then subtract this value from the current weight (optionally multiplied by some learning rate, eta, which we'll set to 0.5):

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

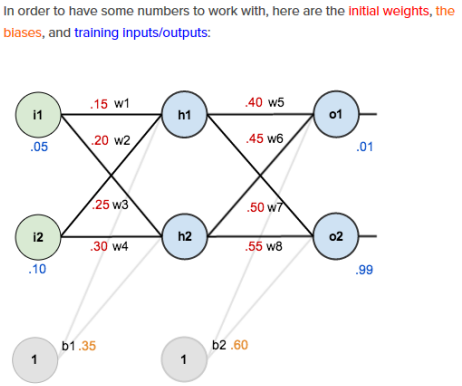
Some sources use α (alpha) to represent the learning rate, others use η (eta), and others even use ϵ (epsilon).

We can repeat this process to get the new weights w_6, w_7, and w_8 :

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$



We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

We perform the actual updates in the neural network *after* we have the new weights leading into the hidden layer neurons (ie, we use the original weights, not the updated weights, when we continue the backpropagation algorithm below).

Hidden Layer

Next, we'll continue the backwards pass by calculating new values for w_1, w_2, w_3, anc

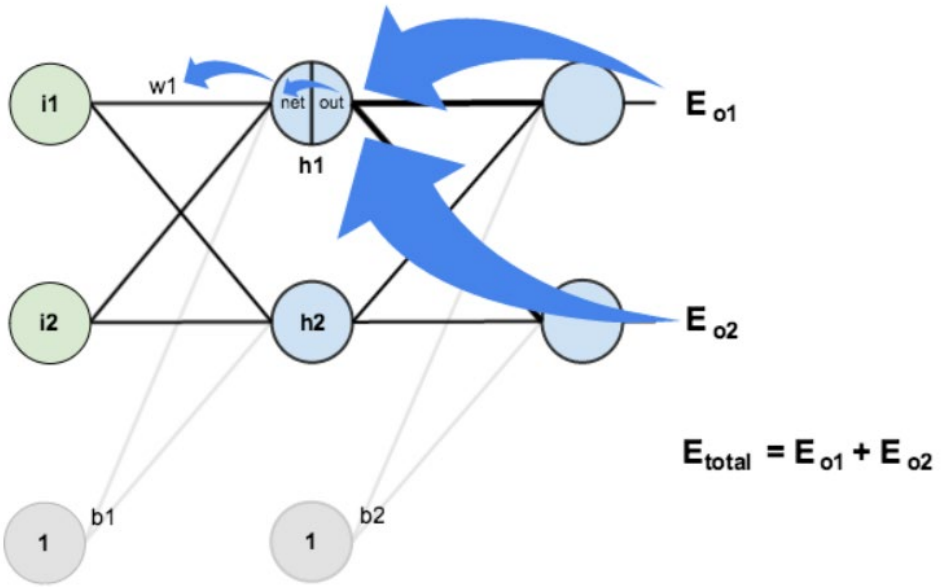
Big picture, here's what we need to figure out:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Visually: ➔

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

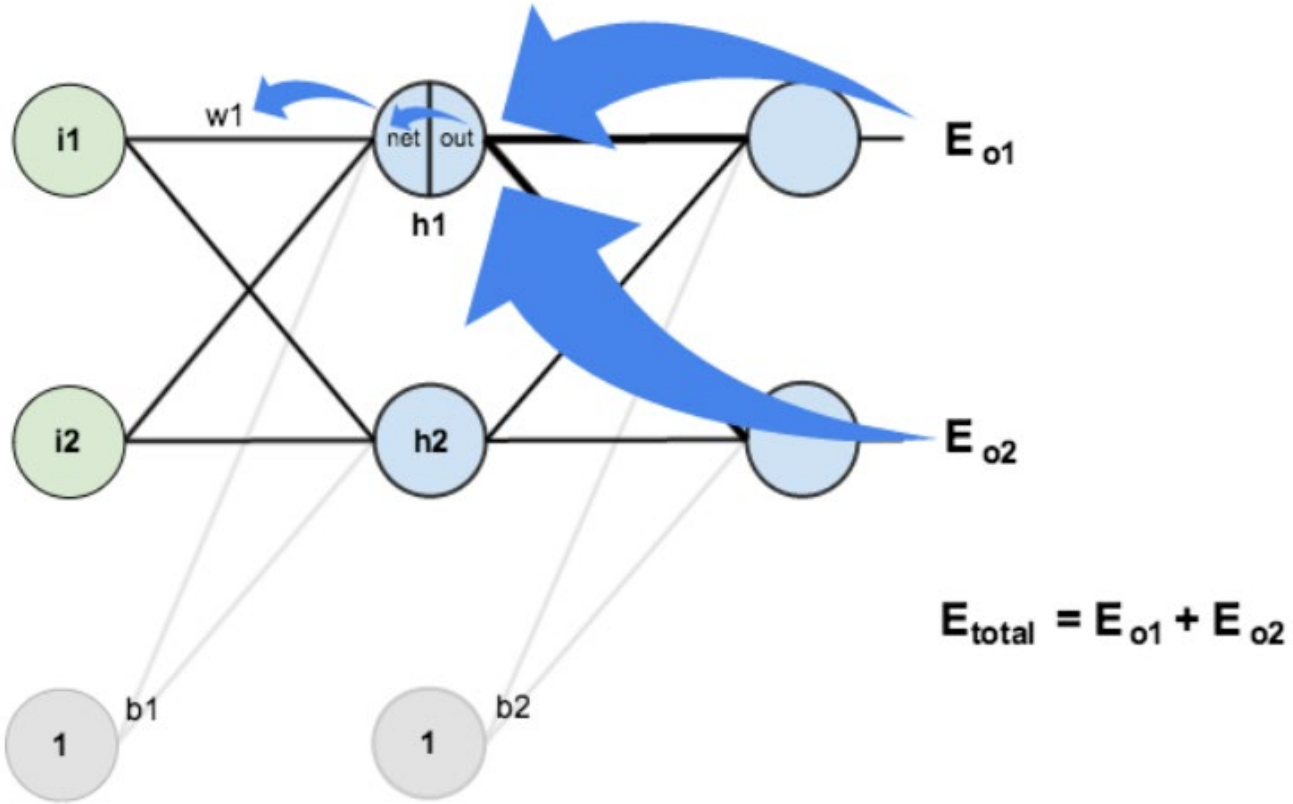


출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

We're going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that out_{h1} affects both out_{o1} and out_{o2} therefore the $\frac{\partial E_{total}}{\partial out_{h1}}$ needs to take into consideration its effect on the both output neurons:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

Plugging them in:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

Therefore:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:

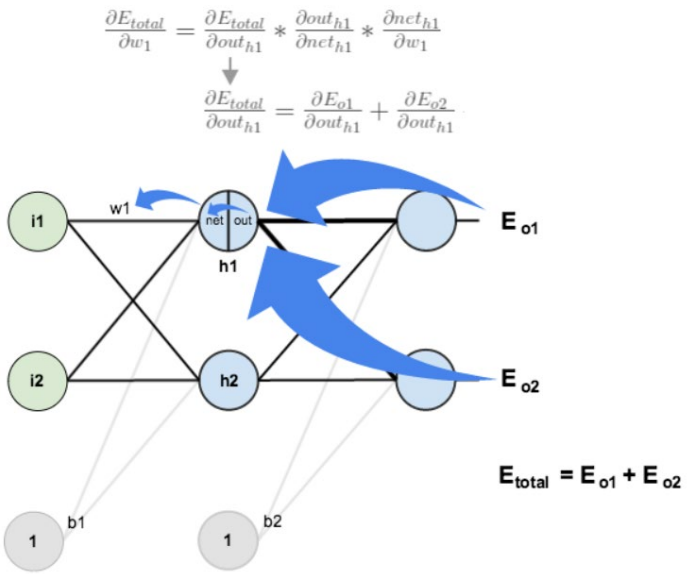
$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

You might also see this written as:

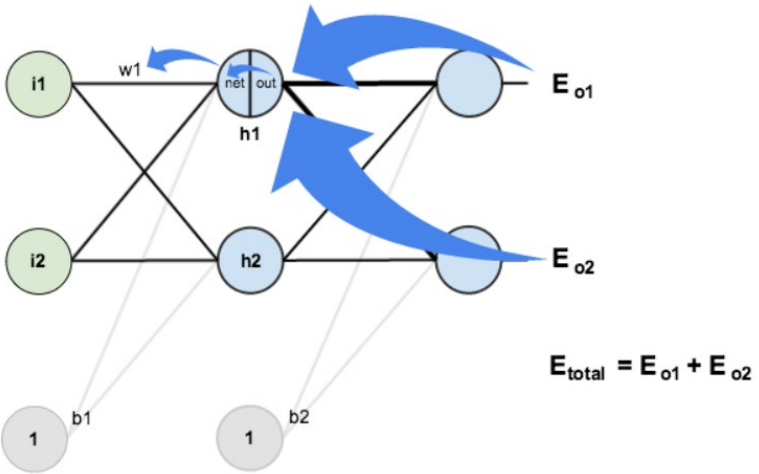
$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}\right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \delta_o * w_{ho}\right) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

We can now update w_1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

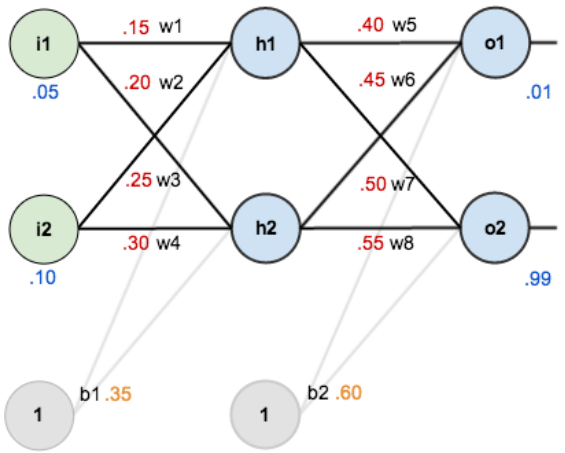
Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

In order to have some numbers to work with, here are the initial weights, the biases, and training inputs/outputs:



출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

Backpropagation Example

Finally, we've updated all of our weights! When we fed forward the 0.05 and 0.1 inputs originally, the error on the network was 0.298371109.

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

After this first round of backpropagation, the total error is now down to 0.291027924.

It might not seem like much, but after repeating this process 10,000 times, for example, the error plummets to 0.0000351085.

At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

출처: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ by Matt Mazur

신경망 모형
miscellaneous

Terminology

• one **epoch** = one forward pass and one backward pass of *all* the training examples

• **batch size** = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

• number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

모형 구축 과정

하나의(예, 첫번째) obs에 대해서 one pass = one forward pass + one backward pass 하고 나면 **w** (weights) 값들이 변한다
(예, initial weights 전체를 $\mathbf{w_0}$ 라고 한다면 첫번째 obs 를 pass하면 $\mathbf{w_0}$=>$\mathbf{w_1}$ 으로 조정 변화된다.
이 $\mathbf{w_1}$으로 두번째 obs의 forward pass를 위한 initial 값이 된다. 그 후 backward pass를 거치면 $\mathbf{w_1}$ => $\mathbf{w_2}$ 가 된다.
이렇게 하여 순차적으로 모든 obs을 다 거치면 마지막 **w** 는 $\mathbf{w_n}$ 이 될 것이다.
$\mathbf{w_0}$=> $\mathbf{w_1}$=> $\mathbf{w_2}$=> ...=>$\mathbf{w_n}$
이러면 한번의 epoch 가 되는 셈이다.

이번에는 마지막 $\mathbf{w_n}$을 가지고 모든 obs 에 한번 더 적용하면 각 obs마다 E (error)가 나올 것이면 이 n개의 E 들의 평균이 이 첫번째 epoch에서의 error1 이 된다.

obs 끼리 상관이 존재할 때는 obs의 순서에 따라 **w** 가 영향을 받을 수 있으므로 첫번째, 두번째,.. obs들의 순서를 정할 때는 random 하게 뽑아서 할 수도 있다.

모형 구축 과정 (continued)

두 번째 epoch 시작의 첫번째 obs의 **w**는 앞의 $\mathbf{w_n}$이 initial 값이 된다.
이렇게 하여 두번째 epoch의 마지막 $\mathbf{w_n}$ 으로 모든 obs에 적용하여 평균한 error2가 된다.
epoch를 거듭할 수록 이 error1, error2, ...들의 변화량을 보면서 최종 NN 모형을 결정한다.

만약 batch 가 2이면, 한번에 2개의 obs 를 하나의 같은 **w** 에 적용하는 것이다.
예를 들면, 첫번째, 두번째 obs 는 하나의 $\mathbf{w_0}$ (initial)에 적용하여 one forward pass 후에는 첫번째 obs 에 대한 E1 과 두번째 obs 에 대한 E2 가 생긴다. 이 둘의 평균(MSE)을 E로 하여 backward pass를 하여 $\mathbf{w_0}$을 조정하여 하나의 $\mathbf{w_1}$을 만든다($\mathbf{w_0}$ => $\mathbf{w_1}$).
이 $\mathbf{w_1}$으로 두번째 batch 인 세번째, 네번째 obs 에 적용할 initial **w** 가 된다.
이렇게 하여 모든 obs 에 iteration 하면 하나의 epoch 가 되고, 마지막 **w**를 다시 한번 더 모든 obs 에 적용하여 평균 error 가 이 epoch의 error 가 된다.

앞 페이지의 경우는 batch가 1인 경우이다.
최근에는 컴퓨터의 발달로 batch가 1인 경우가 일반적이다. 즉, 하나의 obs 마다 E를 계산하는 것이 일반적이다.

신경망 모형 구축을 위한 Tip

▣ An approach:
- Start with none (no hidden layer)
- Add nodes one at a time, monitoring performance
- Stop adding units when the generalization error begins to increase

▣ NN's train best when all the input and output values are between 0 and 1, or -1 and 1

▣ **입력변수 표준화**: input 변수들을 변환(표준화, 정규화 등)하는 이유는, 만약 linear combination(앞에서 net 또는 SUM) 값이 너무 클 경우, 그 값을 sigmoid activation 함수로 변환하면 거의 1에 saturated 된다. 즉, net 값이 어느 정도 커버리면, 아무리 net 값이 차이가 나더라도 그 값들을 sigmoid 함수로 변환하여 나온 값은 모두 1에 가까워져 큰 차이가 없게 된다. 따라서 아무리 net 값이 차이가 나더라도 sigmoid 값은 1 가까이에서는 차이가 나지 않는다. 그래서 net 값을 계산하기전에 모든 input 변수들을 표준화한 후에 그 값들의 linear combination(net) 하는 것이 더 합리적이라 할 수 있다.

▣ 일반적으로 input variable 들은 표준화(standardization, 평균 =0, 표준편차=1) 시킨 후 입력한다. (표준화 방법은 Min-max 방법 등 여러가지가 있으나 평균=0, 표준편차=1로 표준화하는 것을 많이 사용한다).

▣ 연속형인 Output variable 도 표준화 하면 Gradient Descent 알고리즘으로 optimization 할 때 처리속도가 빠르다.

신경망모형의 장·단점

장점

–신경망의 가장 뛰어난 장점은 좋은 예측 성과에 있다. 신경망은 잡음이 많은 데이터(noisy data)에도 매우 안정적이며 예측변수와 반응변수 간의 관계가 매우 복잡할 때도 이를 파악할 수 있는 능력이 뛰어나다.

단점

–블랙박스와 같은 모형의 구조를 설명하지 못한다.
–신경망은 변수선정 메커니즘을 자체적으로 갖고 있지 않다.
–신경망의 매우 높은 유연성은 학습과정에 충분한 데이터를 가질 때에만 가능하다.
–신경망 모형의 기술적인 문제로서 신경망의 가중치는 전역 최적점이 아닌, 지역 최적점에 빠질 위험이 존재한다.
–신경망의 유용성을 결정할 수 있는 실제적인 고려사항으로는 "학습 시간의 적절성"이있다. 신경망은 상대적으로 많은 계산시간이 필요하며 다른 분류기법보다 계산시간이 훨씬 길다. 학습 시간을 정하는 것은 중요한 의사 결정 요인이다.

참고: Activation function 미분

참고: The standard logistic function has an easily calculated derivative:

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

$$\frac{d}{dx} f(x) = \frac{e^x \cdot (1+e^x) - e^x \cdot e^x}{(1+e^x)^2}$$

$$\frac{d}{dx} f(x) = \frac{e^x}{(1+e^x)^2} = f(x)(1-f(x))$$

The derivative of the logistic function has the property that:

$$\frac{d}{dx} f(x) = \frac{d}{dx} f(-x).$$

Net input (net = $w_{01} + w_{11}x_1 + w_{21}x_2 + w_{31}x_3$)

$$Logistic(net) = \frac{1}{1+e^{-net}}$$

$$HyperbolicTangent(net) = \frac{2}{1+e^{-2net}} - 1$$

$$Linear(net) = net$$

$$\frac{dLogistic}{dnet} = Logistic(net) \cdot \big(1 - Logistic(net)\big)$$

$$\frac{dTanH(net)}{dnet} = 1 - HyperbolicTangent(net)^2$$

$$\frac{dLinear(net)}{dnet} = 1$$

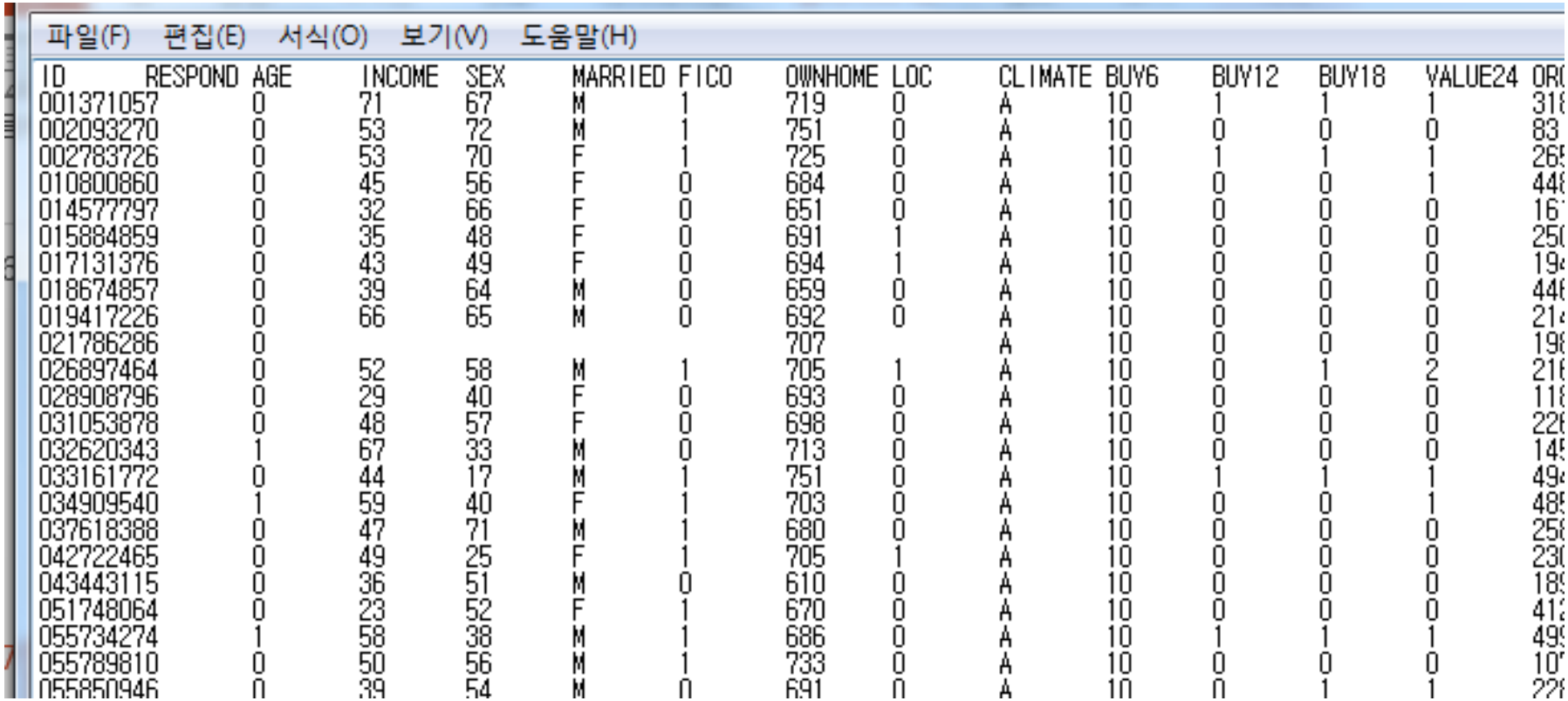
R 실습

R 실습: 신경망 관련 R 패키지

```
nnet(x, y, weights, size, Wts, mask,
    linout = FALSE, entropy = FALSE, softmax = FALSE,
    censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
    maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
    abstol = 1.0e-4, reltol = 1.0e-8, ...)

neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)

deepnet::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=0.5,
                  visible_dropout=0.5)

mxnet::mx.symbol.Activation ← 'relu', 'sigmoid', 'softrelu', 'tanh'
```

Example: BUYTEST

범주	변수	변수의 내용
인구속성	Age	나이(년)
	Income	년수입(1000$)
	Married	1:결혼, 0:미혼
	Sex	F:여자, M:남자
	Ownhouse	1:소유, 0:미소유
지역속성	Loc	거주지 (A-H)
	Climate	거주지의 기온(10,20,30
거래회수	Buy6	최근 6개월 간의 구입회수
	Buy12	최근 12개월 간의 구입회수
	Buy18	최근 18개월 간의 구입회수
거래금액	Value24	지난 24개월 간의 구입총액
신용상태	Fico	신용점수
거래속성	Orgsrc	고객분류(C,D,I,O,P,R,U)
	Discbuy	할인고객 여부(1:할인고객, 0)
	Return24	지난 24개월간 상품의 반품여부 (1:반품, 0)
응답	Respond	광고메일에 응답여부(1:응답, 0)
	Purchtot	광고메일에 의한 구입총액
	C1-C7	광고메일에 의한 품목별 구입액
고객번호	ID	고객번호



파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

ID	RESPOND	AGE	INCOME	SEX	MARRIED	FICO	OWNHOME	LOC	CLIMATE	BUY6	BUY12	BUY18	VALUE24	ORI
001371057	0	71	67	M	1	719	0	A	10	1	1	1	318	
002093270	0	53	72	M	1	751	0	A	10	0	0	0	83	
002783726	0	53	70	F	1	725	0	A	10	1	1	1	265	
010800860	0	45	56	F	0	684	0	A	10	0	0	1	448	
014577797	0	32	66	F	0	651	0	A	10	0	0	0	161	
015884859	0	35	48	F	0	691	1	A	10	0	0	0	250	
017131376	0	43	49	F	0	694	1	A	10	0	0	0	194	
018674857	0	39	64	M	0	659	0	A	10	0	0	0	446	
019417226	0	66	65	M	0	692	0	A	10	0	0	0	214	
021786286	0					707		A	10	0	0	0	198	
026897464	0	52	58	M	1	705	1	A	10	0	1	2	216	
028908796	0	29	40	F	0	693	0	A	10	0	0	0	118	
031053878	0	48	57	F	0	698	0	A	10	0	0	0	226	
032620343	1	67	33	M	0	713	0	A	10	0	0	0	145	
033161772	0	44	17	M	1	751	0	A	10	1	1	1	494	
034909540	1	59	40	F	1	703	0	A	10	0	0	1	485	
037618388	0	47	71	M	1	680	0	A	10	0	0	0	258	
042722465	0	49	25	F	1	705	1	A	10	0	0	0	230	
043443115	0	36	51	M	0	610	0	A	10	0	0	0	189	
051748064	0	23	52	F	1	670	0	A	10	0	0	0	412	
055734274	1	58	38	M	1	686	0	A	10	1	1	1	499	
055789810	0	50	56	M	1	733	0	A	10	0	0	0	107	
055850946	0	39	54	M	0	691	0	A	10	0	1	1	228	

R 실습

```
> setwd("c:/Bigdata")
> buytest = read.table('buytest.txt',sep='\t',header=T)
> head(buytest,10)
          ID RESPOND AGE INCOME SEX MARRIED FICO OWNHOME LOC CLIMATE BUY6
1  001371057       0  71     67   M       1  719       0   A      10    1
2  002093270       0  53     72   M       1  751       0   A      10    0
3  002783726       0  53     70   F       1  725       0   A      10    1
4  010800860       0  45     56   F       0  684       0   A      10    0
5  014577797       0  32     66   F       0  651       0   A      10    0
6  015884859       0  35     48   F       0  691       1   A      10    0
7  017131376       0  43     49   F       0  694       1   A      10    0
8  018674857       0  39     64   M       0  659       0   A      10    0
9  019417226       0  66     65   M       0  692       0   A      10    0
10 021786286       0  NA     NA          NA  707      NA   A      10    0


> nrow(buytest)
[1] 10000
> complete=complete.cases(buytest[,c("RESPOND","AGE","FICO","BUY18")])
> buytest1<-buytest[complete,]
> nrow(buytest1)
[1] 9727
> buytest1$AGE <- scale(buytest1$AGE)
> buytest1$FICO <- scale(buytest1$FICO)
```

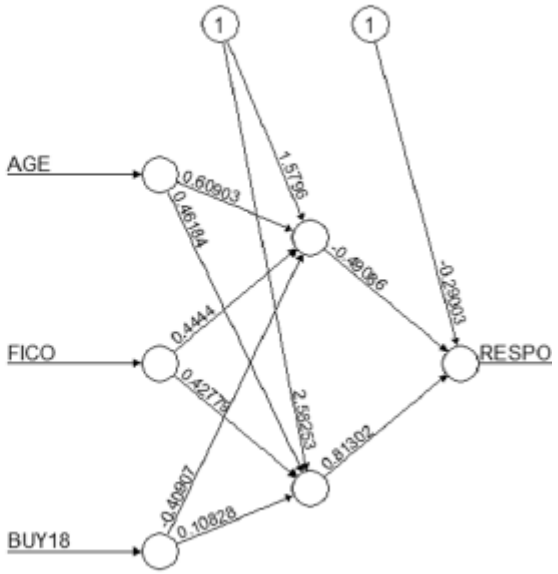
R 실습

```
> library(neuralnet)
>
> set.seed(123)
> nn1<-neuralnet(RESPOND~AGE+FICO+BUY18, data=buytest1, hidden=2,
+ stepmax = 1e+04, threshold = 0.05, act.fct='logistic')
> print(nn1)
Call: neuralnet(formula = RESPOND ~ AGE + FICO + BUY18, data = buytest1,

1 repetition was calculated.

        Error Reached Threshold Steps
1 331.9247771     0.04752704326  4580
```

```
> summary(nn1)
                   Length Class      Mode
call                    7 -none-     call
response             9727 -none-     numeric
covariate           29181 -none-     numeric
model.list              2 -none-     list
err.fct                 1 -none-     function
act.fct                 1 -none-     function
linear.output           1 -none-     logical
data                   26 data.frame list
net.result              1 -none-     list
weights                 1 -none-     list
startweights            1 -none-     list
generalized.weights     1 -none-     list
result.matrix          14 -none-     numeric
```

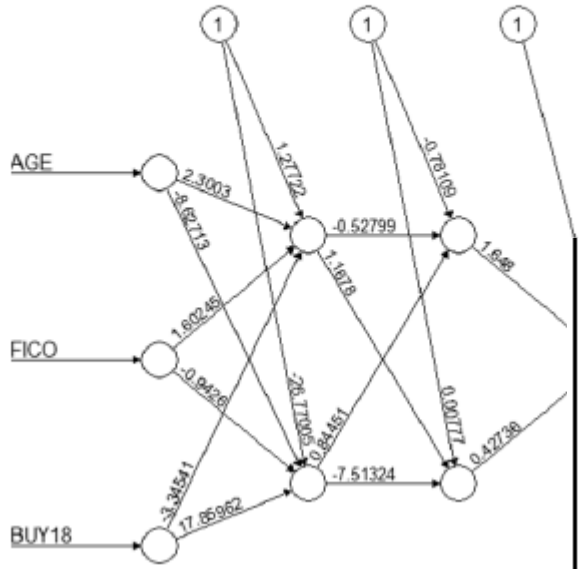
```
> print(nn1$weights)
[[1]]
[[1]][[1]]
                [,1]          [,2]
[1,]  1.5795954154 2.5825266021
[2,]  0.6090329379 0.4618394941
[3,]  0.4443999466 0.4277913738
[4,] -0.4090696043 0.1082817696

[[1]][[2]]
                [,1]
[1,] -0.2900289408
[2,] -0.4908575790
[3,]  0.8130214335
```

```
> head(nn1$net.result[[1]])
           [,1]
1 0.04113811544
2 0.03863306023
3 0.06011519497
4 0.10202768888
5 0.09328557101
6 0.07980124124
> plot(nn1)
```



R 실습

```
> set.seed(123)
> nn2<-neuralnet(RESPOND~AGE+FICO+BUY18,data=buytest1, hidden=c(2,2),
+ stepmax = 1e+04, threshold = 0.05, act.fct='logistic')
> plot(nn2)
```




```
> pred<-compute(nn1,covariate=buytest1[,c("AGE","FICO","BUY18")])
> head(pred$net.result,10)
            [,1]
1  0.04113811544
2  0.03863306023
3  0.06011519497
4  0.10202768888
5  0.09328557101
6  0.07980124124
7  0.06178220883
8  0.08008775981
9  0.03613287891
11 0.10220514830
> summary(pred$net.result)
       V1
 Min.   :0.02525871
 1st Qu.:0.05164143
 Median :0.06630618
 Mean   :0.07494953
 3rd Qu.:0.08878173
 Max.   :0.28616476
```

감사합니다

동국대학교 통계학과

이 영 섭

yung@dongguk.edu