M2608.001300
Machine Learning Fundamentals & Applications

[4: Logistic Regression]

Electrical and Computer Engineering
Seoul National University

Outline

Readings

- *Learning from Data* by Abu-Mostafa, Magdon-Ismail, and Lin
  - Chapter 3: The Linear Model (Sections 3.3 & 3.4)
- *Introduction to Statistical Learning* by James *et al.*
  - Figures 1 and 2 are from this book (Chapter 4)
- *Machine Learning* by Murphy
  - Gradient descent examples are from this book (Chapter 8)

Outline

Introduction

- core of linear models
  - 'signal' $s = \underline{\mathbf{w}^T\mathbf{x}}$: combines input variables linearly
  - we have seen two models based on this

1. linear regression
   - signal itself = output
   - for predicting real (unbounded) response

2. linear classification
   - signal is thresholded at zero to produce ± 1 output
   - for binary decisions

Motivating example

heart attack prediction:

- based on cholesterol level, blood pressure, age, weight, . . .

- cannot predict a heart attack with any certainty
  - but can predict how likely it is to occur given these factors

- a more suitable model than binary decision would be:
  - output y that varies continuously between 0 and 1
  - the closer y is to 1, the more likely heart attack will occur

Logistic regression: 'soft' binary classification

- we introduce the third linear model:
  - outputs probability of a binary response

  e.g. heart attack or not, dead or alive
  - returns 'soft labels' (probability)

- our new model: called "*logistic* regression"
  - output: real (like regression) but bounded (like classification)
    sigmoid
- comparison: linear classification vs logistic regression
  - both deal with a binary event
  - logistic regression: allowed to be "uncertain"

  $\Rightarrow$ intermediate values between 0 and 1 reflect this uncertainty

Example: probability of default

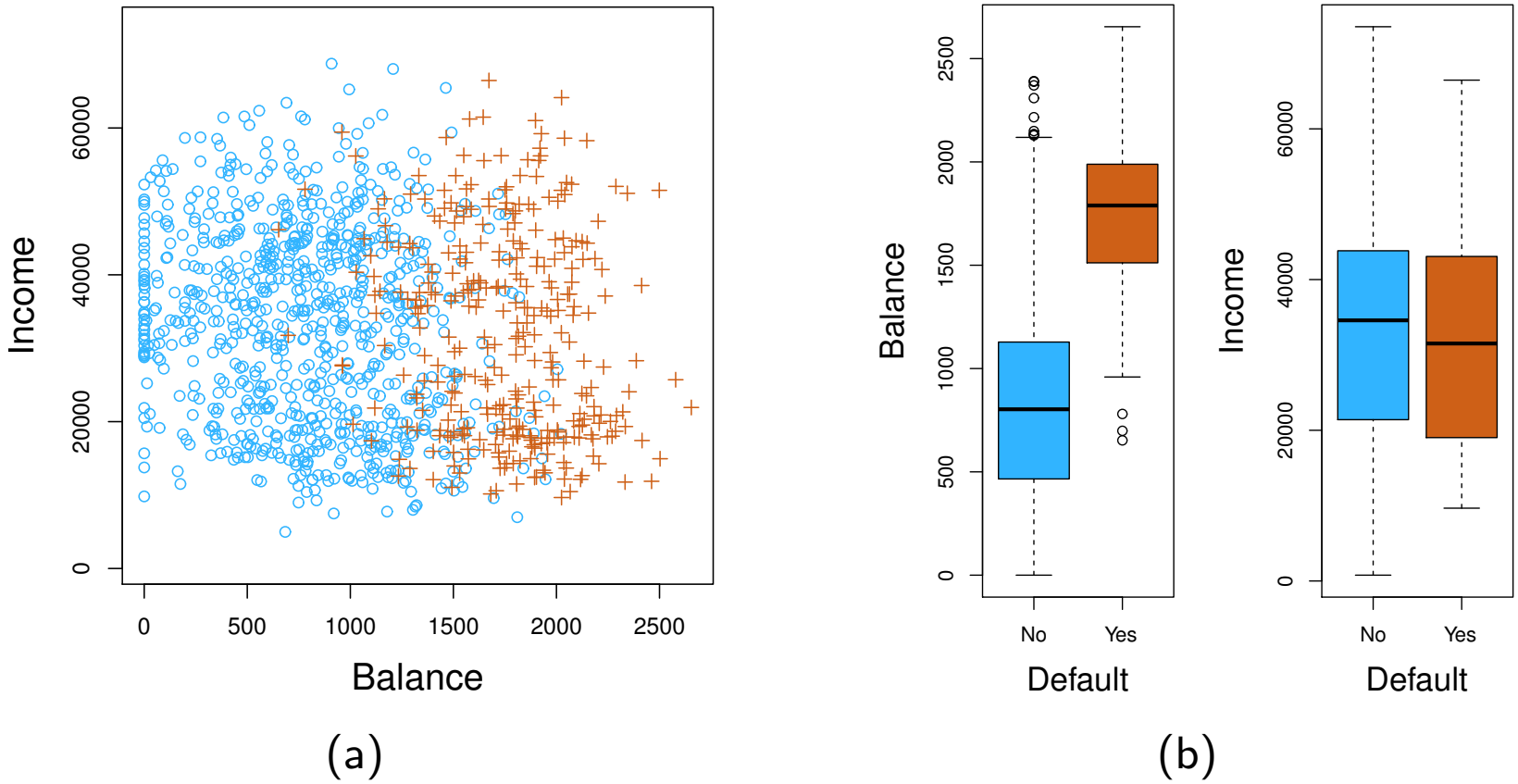
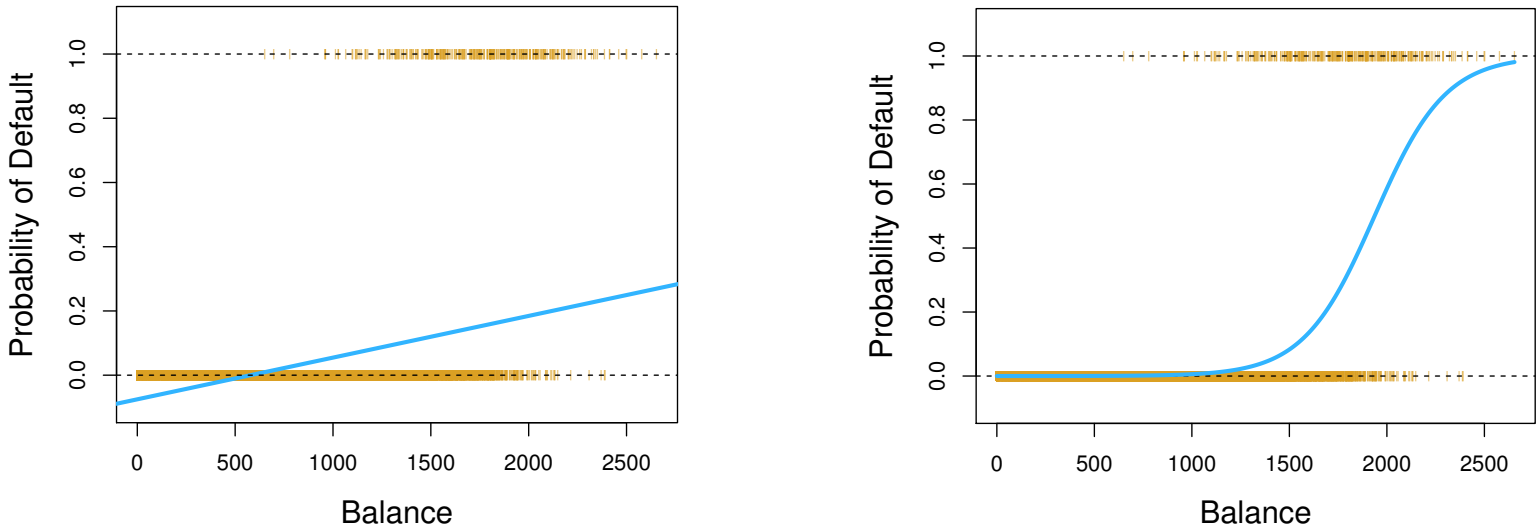


Figure 1: annual incomes and monthly credit card balances: (a) the individuals who defaulted on their credit card payments are shown in orange, and those who did not are shown in blue (b) boxplots of balance/income as a function of default status

- $\mathbb{P}[\text{default} = \texttt{yes} \,|\, \text{balance}]$: probability of default given balance
  - can be estimated/predicted by regression analysis
  - logistic regression: more appropriate than linear regression



(a) linear regression (b) logistic regression

Figure 2: estimated probability of default: (a) some probabilities are negative (b) all probabilities lie between 0 and 1

Logistic regression model

- linear classification: uses a hard threshold on signal $s = \mathbf{w}^{\mathrm{T}}\mathbf{x}$

$$h(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

- linear regression: uses no threshold at all

$$h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

- our new model: needs something between these two
  - smoothly restricts output to probability range [0,1]

- one choice: the logistic regression model

$$h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

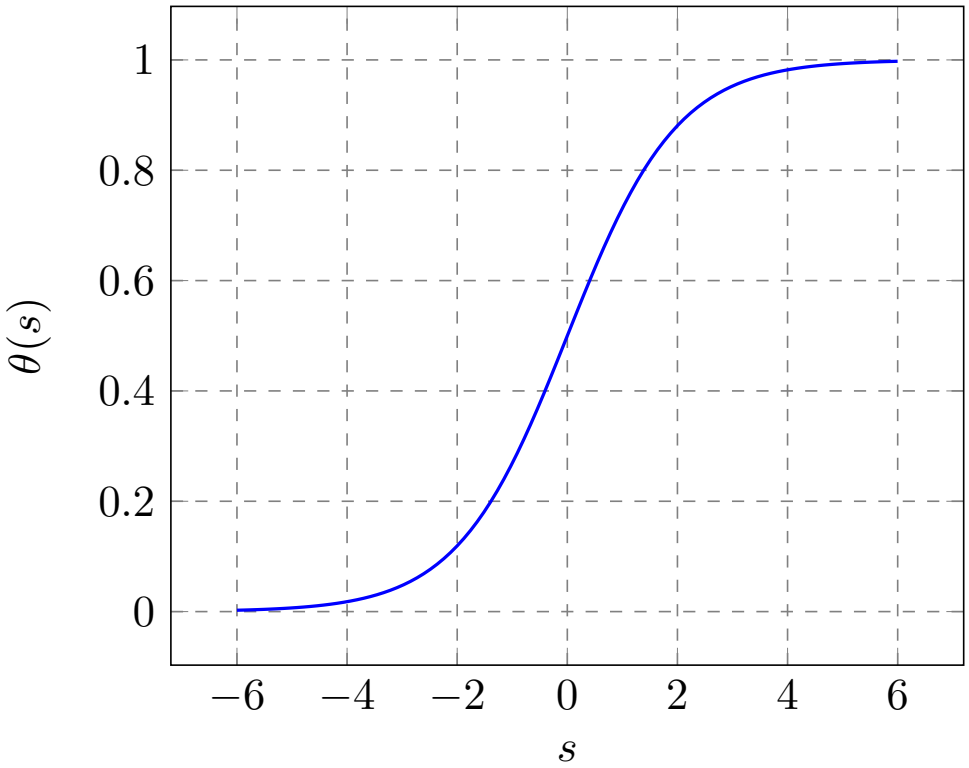
  - θ is so-called *logistic* function

θ: logistic sigmoid

Logistic function θ

- definition
  - for $-\infty < s < \infty$:

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$



- output lies between 0 and 1
  - can be interpreted as probability for binary events

- the above formula of $\theta(s)$
  - allows us to define an error measure that has analytical and computational advantages (details will follow)

- other names of logistic function θ

  "soft threshold" in contrast to the hard one in classification

  "sigmoid" its shape looks like flattened-out 's'

- note:

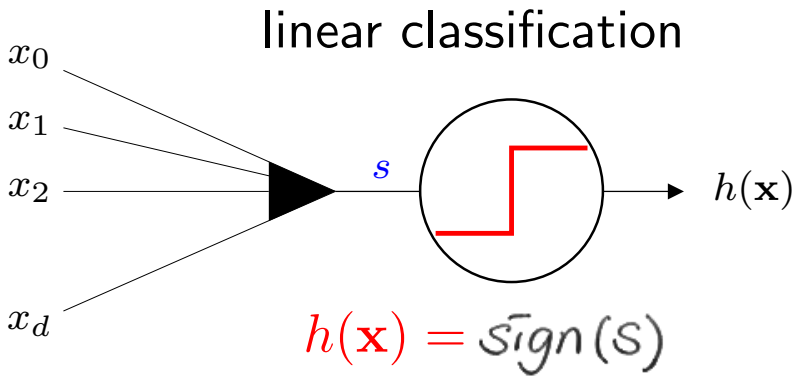
$$1 - \theta(s) = 1 - \frac{1}{1+e^{-s}} = \frac{e^{-s}}{1+e^{-s}} = \theta(-s)$$

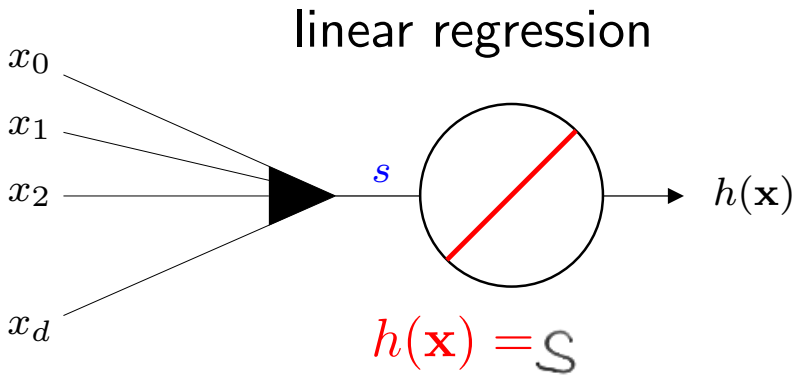
★ $1-\theta(s)=\theta(-s)$
이므로 뒤에서 정리해줌

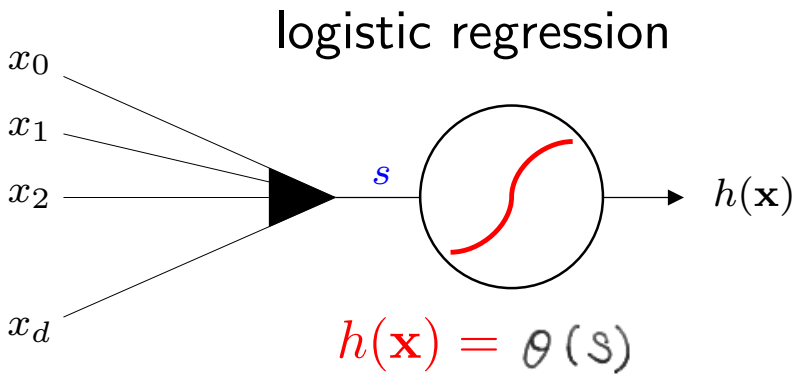
Linear models

- based on "signal" s:

$$s = \sum_{i=0}^{d} w_i x_i$$







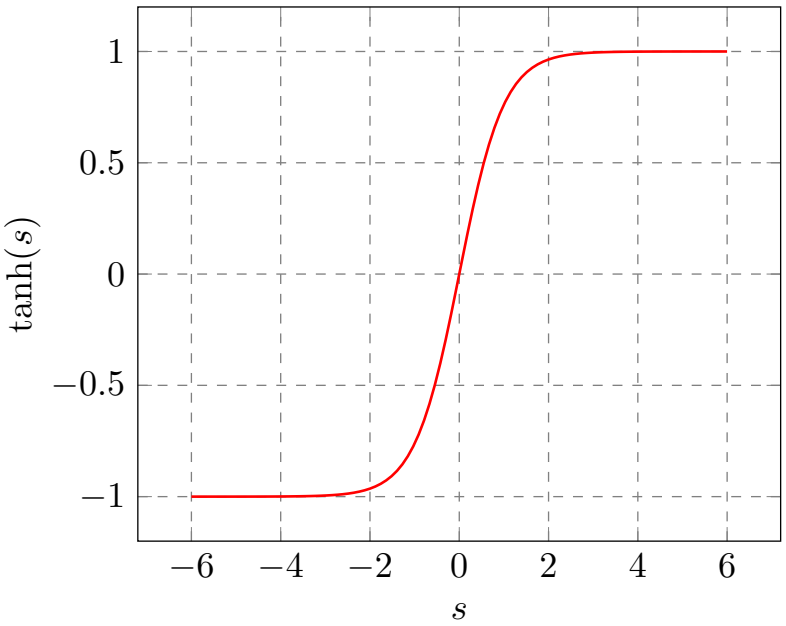
Example: heart attack

- prediction of heart attacks
  - input $\mathbf{x}$: cholesterol level, age, weight, etc.
  - signal $s = \mathbf{w}^{\mathrm{T}}\mathbf{x}$: "risk score"
- linear classification
  - $h(\mathbf{x})$ returns ± 1: heart attack $(+1)$ or not (-1) for sure
- linear regression
  - $h(\mathbf{x})$ returns risk score s itself
- logistic regression
  - $h(\mathbf{x})$ returns $\theta(s)$: probability of heart attack

Another popular sigmoid function

- hyperbolic tangent:

$$\tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}}$$



- properties
  - $\tanh(s)$ converges to a hard threshold for large $|s|$
  - converges to no threshold for small $|s|$

Comparison

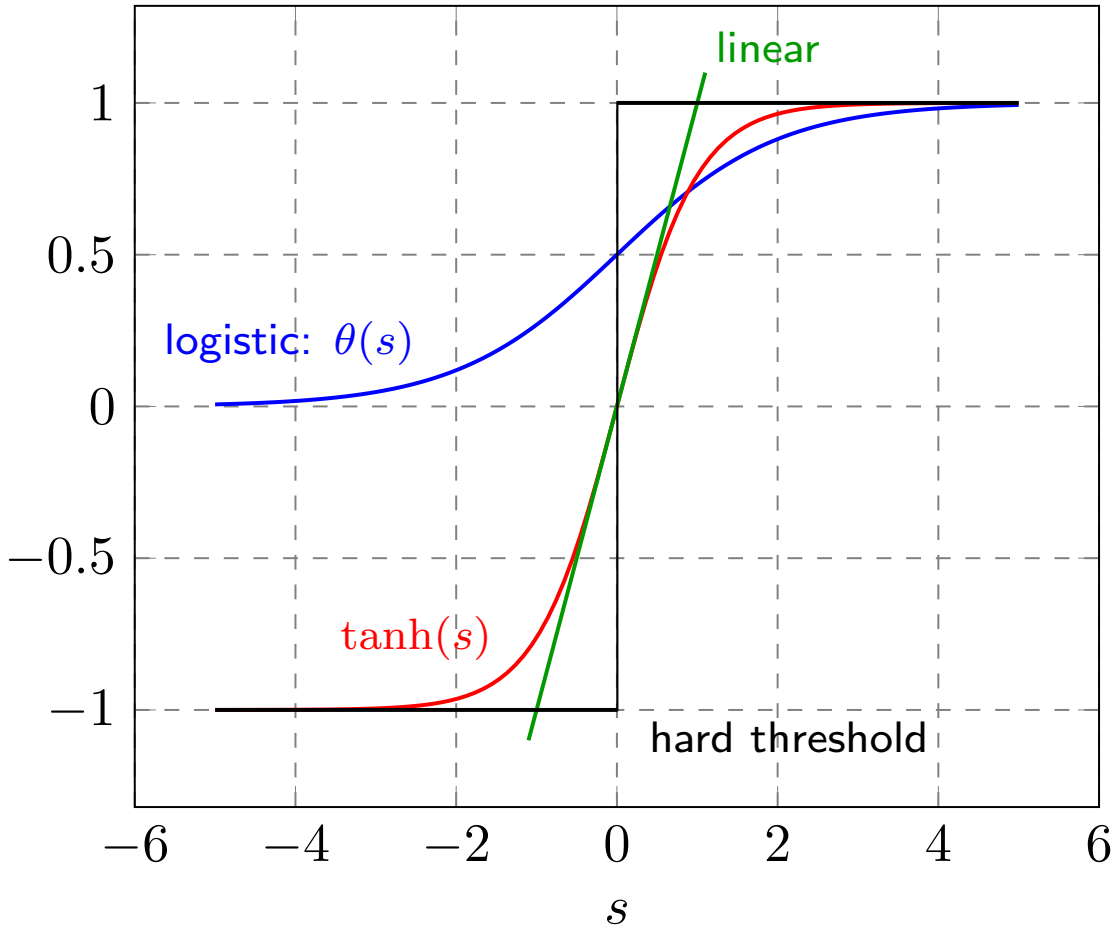
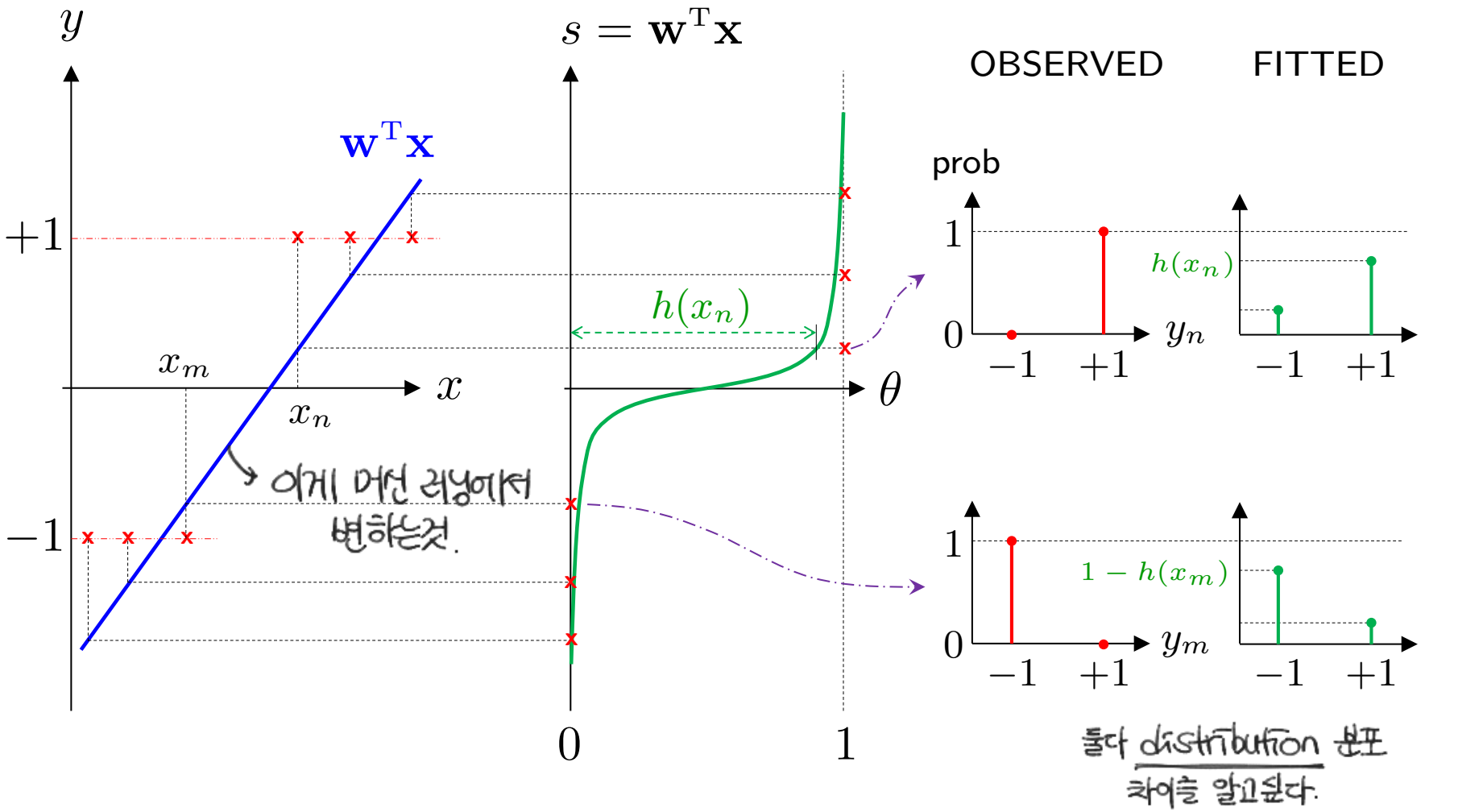


Figure 3: linear, logistic, $\tanh$, and hard threshold

Outline

Big picture



$$\mathbb{P}[y = +1|\mathbf{x}] = f(\mathbf{x}) \sim h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

$$P(y|\mathbf{x}) = h(\mathbf{x})^{[\![y=+1]\!]}(1-h(\mathbf{x}))^{[\![y=-1]\!]} = \theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

Learning target in logistic regression

- learning target:
  - probability of event $y = +1$ given input $\mathbf{x}$

$$f(\mathbf{x}) = \mathbb{P}[\, y = +1 \mid \mathbf{x} \,]$$

event에 대한 확률

  - *e.g.* probability of a patient being at risk for heart attack given the characteristics of the patient

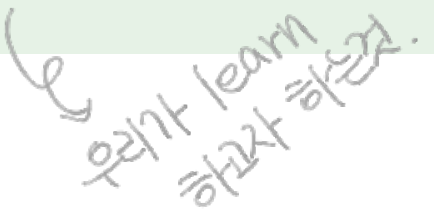
- training data do not give us the value of f explicitly
  - rather, give us samples generated by this probability
  - *e.g.* patients with/without heart attacks

$\Rightarrow$ learning target: inherently noisy

- target is $P(y|\mathbf{x})$ rather than $y = f(\mathbf{x})$

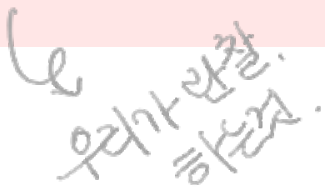
ideal (noiseless) data

$$(\mathbf{x}_1, f_1 = 0.99 = \mathbb{P}[+1|\mathbf{x}_1])$$
$$(\mathbf{x}_2, f_2 = 0.13 = \mathbb{P}[+1|\mathbf{x}_2])$$
$$\vdots$$
$$(\mathbf{x}_N, f_N = 0.51 = \mathbb{P}[+1|\mathbf{x}_N])$$



actual (noisy) data

$$(\mathbf{x}_1, y_1 = +1)$$
$$(\mathbf{x}_2, y_2 = -1)$$
$$\vdots$$
$$(\mathbf{x}_N, y_N = -1)$$



함수가 아닌 distribution 을 learn.

- we thus view data as generated by target distribution $P(y|\mathbf{x})$

$$P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases} \tag{1}$$

- to learn from such data, we need a proper error measure
  - it gauges how close a given hypothesis h is to target f
  - in terms of noisy ± 1 examples

What makes an h good?

- 'fitting' data $\mathcal{D}$ means finding a good h
  - h is good if $\begin{cases} h(\mathbf{x}_n) \approx 1 & \text{whenever } y_n = +1 \\ h(\mathbf{x}_n) \approx 0 & \text{whenever } y_n = -1 \end{cases}$
- a simple error measure that captures this

$$\mathrm{E}_{\text{in}}(h) = \frac{1}{N} \sum_{n=1}^{N} \left(h(\mathbf{x}_n) - \frac{1}{2}(1 + y_n) \right)^2$$

hit exponential 이라서.

  - not very convenient (hard to minimize)

- preview: we will use cross-entropy error measure

$$\mathrm{E}_{\mathrm{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n}\right) \tag{2}$$

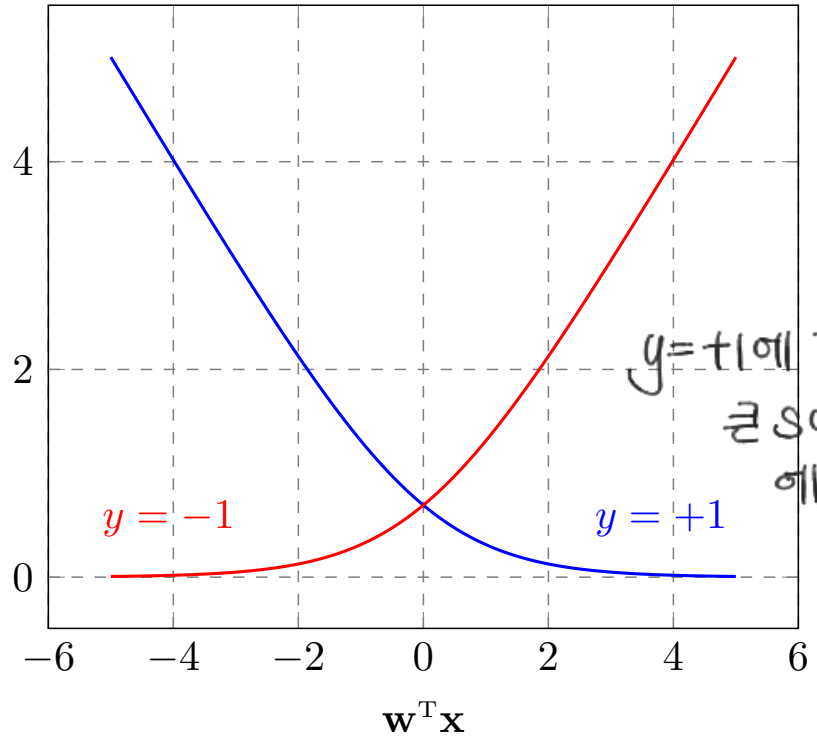
  - looks complicated and ugly; however (2) is
  1. based on an intuitive probabilistic interpretation (next page)
  2. has nice property for gradient-based optimization (although not as easy as linear regression)

(1. 에러의 성질 가져
2. convex 꼴을 가진다. 가장 낮은점 쉽게

- $y_n = +1$ encourages $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n \gg 0$, so $\theta(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) \approx 1$, and vice versa

포인트 wise 에러

$$\ln\left(1 + e^{-y\mathbf{w}^{\mathrm{T}}\mathbf{x}}\right)$$



$y=+1$에 가까워지면 큰 s에 대해 에러↓

↳ 에러의 성질을 가진다.

Defining error measure

1) 확률적 : ML. (maximum likelihood)
2) 정보이론 : cross entropy.

- standard error measure in logistic regression: based on "*likelihood*"
  - how 'likely' is it to get output y from input $\mathbf{x}$, if target distribution $P(y|\mathbf{x})$ was indeed captured by $h(\mathbf{x})$?

- based on (1), that likelihood would be

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1 \\ 1 - h(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

$$= \boxed{\theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x})} \qquad (3)$$

우리가 사용하는 θ 함수는 θ(s)+θ(-s)=1 의 성질을 가진다.

  - recall: $h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ and $1 - \theta(s) = \theta(-s)$
  - this simplicity in (3): reason for defining $\theta(s)$ as $e^s/(1+e^s)$

Criterion for choosing h: maximum likelihood

- assume $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ are independently generated
  - $\Rightarrow$ probability of getting all y_n's from corresponding $\mathbf{x}_n$'s:

$$P(y_1|\mathbf{x}_1)P(y_2|\mathbf{x}_2)\cdots P(y_N|\mathbf{x}_N) = \prod_{n=1}^{N} P(y_n|\mathbf{x}_n)$$

  - this is the *likelihood* of training data

- the method of *maximum likelihood*
  - selects the hypothesis h that maximizes this probability

- we can equivalently minimize a more convenient quantity

$$-\frac{1}{N}\ln\left(\prod_{n=1}^{N}P(y_n|\mathbf{x}_n)\right)=\frac{1}{N}\sum_{n=1}^{N}\ln\frac{1}{P(y_n|\mathbf{x}_n)}$$

$\because$ $-\frac{1}{N}\ln(\cdot)$ is a monotonically decreasing function

log likelihood

NLL : negative ✓ : max(ML) = min(NLL)

- substituting with Eq. (3), we would be minimizing (wrt $\mathbf{w}$):

$$\frac{1}{N}\sum_{n=1}^{N}\ln\frac{1}{\theta(y_n\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)}=\frac{1}{N}\sum_{n=1}^{N}\ln\begin{cases}\frac{1}{h(\mathbf{x}_n)} & \text{for } y_n=+1\\ \frac{1}{1-h(\mathbf{x}_n)} & \text{for } y_n=-1\end{cases} \tag{4}$$

$$=\frac{1}{N}\sum_{n=1}^{N}\left\{[\![y_n=+1]\!]\ln\frac{1}{h(\mathbf{x}_n)}+[\![y_n=-1]\!]\ln\frac{1}{1-h(\mathbf{x}_n)}\right\} \tag{5}$$

positive의 비율 → 확률

  - more details on (5) will follow

정보이론에서 다시

E_{in} for logistic regression

- the fact that we are *minimizing* quantity (4)

  $\Rightarrow$ allows us to treat it as an 'error measure'

  에러로 생각하자.

- substituting the functional form for $\theta(y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n)$ produces
  - in-sample error measure for logistic regression

$$\boxed{\mathrm{E}_{\text{in}}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n}\right)} \tag{6}$$

pointwise 에러 같은 존재.

  - implied point-wise error

$$\mathrm{e}(h(\mathbf{x}_n), y_n) = \ln\left(1 + e^{-y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n}\right)$$

Optimization involved

$$\max \quad \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) \tag{7}$$

$$\Leftrightarrow \quad \max \quad \ln\left(\prod_{n=1}^{N} P(y_n|\mathbf{x}_n)\right) \tag{8}$$

$$\equiv \quad \max \quad \sum_{n=1}^{N} \ln P(y_n|\mathbf{x}_n) \tag{9}$$

$$\Leftrightarrow \quad \min \quad -\frac{1}{N}\sum_{n=1}^{N} \ln P(y_n|\mathbf{x}_n) \tag{10}$$

$$\equiv \quad \min \quad \frac{1}{N}\sum_{n=1}^{N} \ln \frac{1}{P(y_n|\mathbf{x}_n)} \tag{11}$$

$$\equiv \quad \min \quad \frac{1}{N}\sum_{n=1}^{N} \ln \frac{1}{\theta(y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n)} \tag{12}$$

$$\equiv \quad \min \quad \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+e^{-y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n}\right) \tag{13}$$

모델링

우리가 θ를 $\frac{1}{1+e^{-s}}$ 함수를 사용하게 때문

cross-entropy

∴ Max. likelihood = Min. cross entropy.

$$\mathrm{E}_{\mathrm{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+e^{-y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n}\right)$$

- (11) → (12): by logistic modeling

Remarks

- this error measure: small when $y_n \mathbf{w}^{\mathrm{T}} \mathbf{x}_n$ is large and *positive*
  - which would imply that $\mathrm{sign}(\mathbf{w}^{\mathrm{T}} \mathbf{x}_n) = y_n$
- therefore, as our intuition expect
  - the error measure encourages $\mathbf{w}$ to 'classify' each $\mathbf{x}_n$ correctly
- alternative derivation of $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ is possible
  - based on the notion of *cross entropy*

Cross entropy

- consider two pmfs $\{p, 1-p\}$ and $\{q, 1-q\}$ with binary outcomes
  - *e.g.* $\mathbb{P}[\text{pass}] = p$ and $\mathbb{P}[\text{fail}] = 1-p$
- cross entropy for these two pmfs: defined by

$$p \log \frac{1}{q} + (1-p) \log \frac{1}{1-q} \tag{14}$$

  - $\log$ denotes binary logarithm ($\log_2$)
  - see Appendix for more details
- cross entropy measures the 'error' for approximating
  - 'observed' pmf $\{p, 1-p\}$ by ~~'fitted'~~ = learned pmf $\{q, 1-q\}$

Alternative derivation based on cross entropy

- recall:

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{for } y = +1 \\ 1 - h(\mathbf{x}) & \text{for } y = -1 \end{cases} \tag{15}$$

$$= \theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x}) \tag{16}$$

- (15) can also be written as

$$P(y|\mathbf{x}) = h(\mathbf{x})^{[\![y=+1]\!]}(1 - h(\mathbf{x}))^{[\![y=-1]\!]} \tag{17}$$

1일때 1, 아니면 0.

- likelihood of data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ now becomes

$$\prod_{n=1}^{N} P(y_n|\mathbf{x}_n) = \prod_{n=1}^{N} h(\mathbf{x}_n)^{[\![y_n=+1]\!]}(1 - h(\mathbf{x}_n))^{[\![y_n=-1]\!]} \tag{18}$$

- negative log-likelihood (NLL) is given by

$$
\begin{aligned}
NLL(\mathbf{w}) &\triangleq -\log \left\{ \prod_{n=1}^{N} P(y_n|\mathbf{x}_n) \right\} \\
&= -\log \left\{ \prod_{n=1}^{N} h(\mathbf{x}_n)^{[\![y_n=+1]\!]} (1-h(\mathbf{x}_n))^{[\![y_n=-1]\!]} \right\} \\
&= \sum_{n=1}^{N} \left\{ [\![y_n=+1]\!] \log \frac{1}{h(\mathbf{x}_n)} + [\![y_n=-1]\!] \log \frac{1}{1-h(\mathbf{x}_n)} \right\} \qquad (19)
\end{aligned}
$$

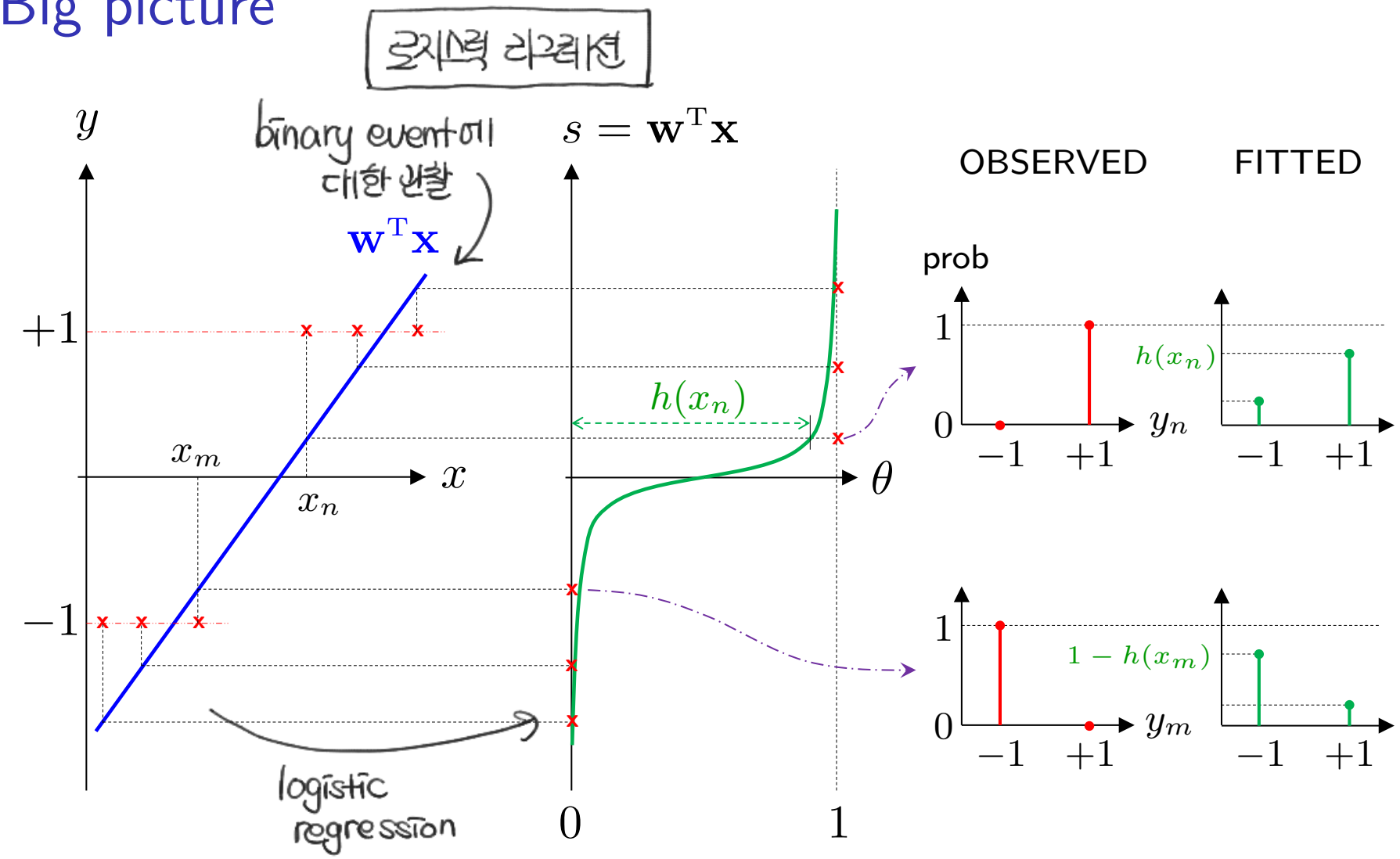
↑ 1/N 하면 예전 확률 비슷하게 됨.

  - this is also called cross-entropy error function
  - related to log loss and KL divergence

- nth term in (19): cross entropy for two pmfs measured on $(\mathbf{x}_n, y)$
  - observed: $\{[\![y_n = +1]\!], 1 - [\![y_n = +1]\!]\}$
  - fitted: $\{h(\mathbf{x}_n), 1 - h(\mathbf{x}_n)\}$
  - in (14): $p = [\![y_n = +1]\!]$ and $q = h(\mathbf{x}_n)$

- minimizing $NLL(\mathbf{w})$ is equivalent to
  - maximizing (18)
  - minimizing (5)
  - minimizing (6) for $h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$

Big picture



$$\mathbb{P}[y = +1|\mathbf{x}] = f(\mathbf{x}) \sim h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

$$P(y|\mathbf{x}) = h(\mathbf{x})^{[\![y=+1]\!]}(1-h(\mathbf{x}))^{[\![y=-1]\!]} = \theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

Outline

Training linear models

- linear classification (the perceptron)
  - minimize E_{in} by solving a combinatorial optimization problem
  - PLA/pocket algorithm
- linear regression
  - minimize E_{in} by setting $\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}) = 0$
  - analytic pseudo-inverse algorithm
- logistic regression
  - minimize E_{in} by setting $\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}) = 0$
  - iterative optimization (*e.g.* gradient descent)

Training logistic regression 어떻게 train?

허서안 matrix 보고 0이 배고파서 positive semi definite 이면 convex 꼴. 맞아

- take an approach similar to linear regression: set $\nabla E_{in}(W) = 0$
- unfortunately, $\nabla \mathrm{E}_{\mathrm{in}}$ for logistic regression:
  - not easy to manipulate analytically

$$\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\frac{y_n \mathbf{x}_n}{1+e^{y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n}} \tag{20}$$

앞의 식을 미분

$$= \frac{1}{N}\sum_{n=1}^{N} -y_n \mathbf{x}_n \theta(-y_n \mathbf{w}^{\mathrm{T}}\mathbf{x}_n) \tag{21}$$

$\Rightarrow$ an analytic solution is not feasible!

$\Rightarrow$ 따라서 iteratively 문제를 푼다.

Iterative optimization

- we will iteratively set the gradient to zero
  - instead of analytically setting it to zero
- gradient descent: a very general algorithm
  - can train many other models with smooth error measures
  - has particularly nice properties for logistic regression

- gradient: a fancy word for the derivative of vector functions
  - direction of greatest *increase* of a function (*i.e.* "steepest")
  - zero at (local) max/min (∵ no single direction of increase)

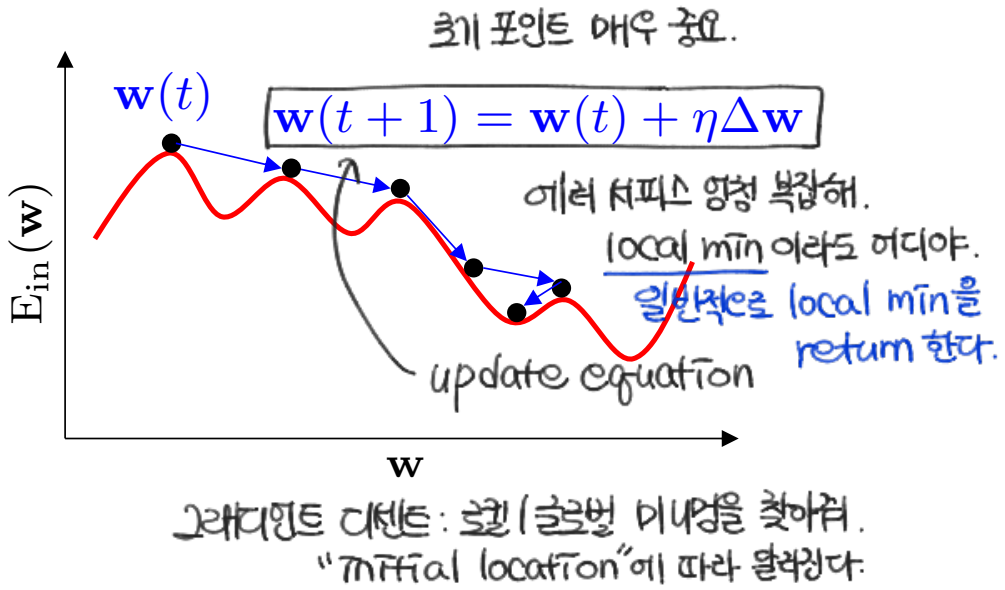
Gradient descent

· logistic regression 에서 중요.
· optimization 문제에서 많이 사용된다.

- general technique for minimizing twice-differentiable function
  - *e.g.* $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ in logistic regression and neural networks
- physical analogy: a ball rolling down a hilly surface
  - if placed on a hill, it will roll down
  - it comes to rest at the bottom of a valley

- same basic idea underlies gradient descent
  - $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ is a 'surface' in high-dimensional space
  - in step 0, we start somewhere on this surface, at $\mathbf{w}(0)$
  - and try to roll down the surface, thereby decreasing E_{in}

방향과 거리를 정해 뛰어야
(RH Iteration 에서 결정사항)



- two things to decide:
  1. which direction?
  2. how much?

$$\Delta\mathbf{w} = -\nabla\mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))$$

(η: learning rate

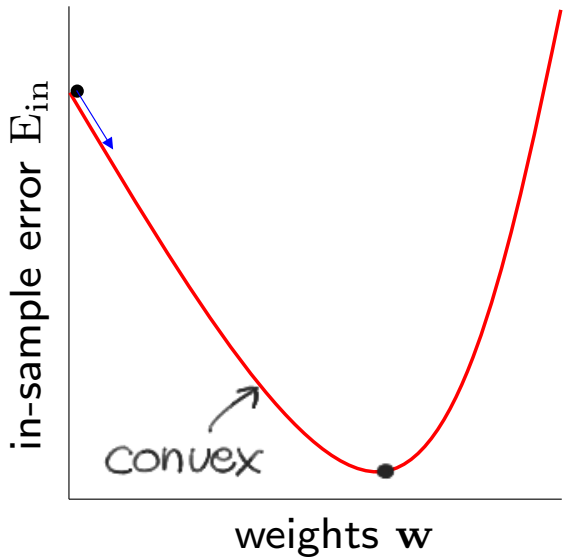
$-\nabla$: negative gradient
방향으로 뛴다.

→ Iterative 알고리즘

- caution: local optimum
  - initial location is crucial

Advantage of using cross-entropy error

- there is only one valley!
  - no matter where we start
  - same (unique) *global minimum*



- $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ for cross-entropy error:
  - a convex function of $\mathbf{w}$
    (check: Hessian of E_{in} is positive semidefinite)
    (Hessian: 2번미분, positive semidefinite: not negative)
  - $\Rightarrow$ no local minima

How to 'roll' down E_{in}-surface

- assume:
  - you are at weights $\mathbf{w}(t)$, and
  - take a step of size η in the direction of *unit* vector $\hat{\mathbf{v}}$

unit vector

- new weights:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \hat{\mathbf{v}}$$

- question: which is the best direction of $\hat{\mathbf{v}}$?
  - pick $\hat{\mathbf{v}}$ s.t. $\mathrm{E}_{\mathrm{in}}(\mathbf{w}(t+1))$ becomes as small as possible

  $\Leftrightarrow$ maximize $\mathrm{E}_{\mathrm{in}}(\mathbf{w}(t)) - \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t+1))$

  $\Leftrightarrow$ minimize $\mathrm{E}_{\mathrm{in}}(\mathbf{w}(t+1)) - \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))$

Recall

- Taylor expansion of vector function $f(\mathbf{x})$ to "1st order"

$$f(\mathbf{x}+\boldsymbol{\delta}) - f(\mathbf{x}) \approx \nabla f(\mathbf{x})^{\mathrm{T}} \boldsymbol{\delta}$$

- for arbitrary vector $\mathbf{x}$ and unit vector $\mathbf{y}$

$$\mathbf{x}^{\mathrm{T}}\mathbf{y} \geq -||\mathbf{x}||$$

  - equality holds if and only if

$$\mathbf{y} = -\frac{\mathbf{x}}{||\mathbf{x}||}$$

(*i.e.*, the angle between $\mathbf{x}$ and $\mathbf{y}$ is 180°)

Negative gradient gives the steepest descent

- approximating the change in E_{in}

$$\begin{aligned}\Delta \mathrm{E}_{\text{in}} &= \mathrm{E}_{\text{in}}(\mathbf{w}(t+1)) - \mathrm{E}_{\text{in}}(\mathbf{w}(t)) \\ &= \mathrm{E}_{\text{in}}(\mathbf{w}(t) + \eta\hat{\mathbf{v}}) - \mathrm{E}_{\text{in}}(\mathbf{w}(t)) \\ &\approx \eta \nabla \mathrm{E}_{\text{in}}(\mathbf{w}(t))^{\mathrm{T}} \hat{\mathbf{v}} \\ &\geq -\eta ||\nabla \mathrm{E}_{\text{in}}(\mathbf{w}(t))||\end{aligned}$$

- the best (steepest) direction to move is the negative gradient:

$$\boxed{\hat{\mathbf{v}} = -\frac{\nabla \mathrm{E}_{\text{in}}(\mathbf{w}(t))}{||\nabla \mathrm{E}_{\text{in}}(\mathbf{w}(t))||}} \tag{22}$$

가장 steepest

Fixed learning-rate gradient descent

- gradient descent algorithm (with fixed η):

1: initialize weights at time step $t = 0$ to $\mathbf{w}(0)$
2: **for** $t = 0, 1, 2, \ldots$ **do**
3: compute gradient $\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))$
4: set direction to move: $\hat{\mathbf{v}}_t = -\frac{\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))}{||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||}$
5: update weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \hat{\mathbf{v}}_t$

$$\boxed{\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \frac{\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))}{||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||}} \tag{23}$$

6: iterate to next step until it is time to stop
7: return final weights

Remarks

- to use gradient descent, one must compute gradient
  - this can be done explicitly for logistic regression: Eq. (21)
- we can apply gradient descent to logistic regression
  - to axpproximately minimize $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ in Eq. (6)

→ convex 이면 정확하게

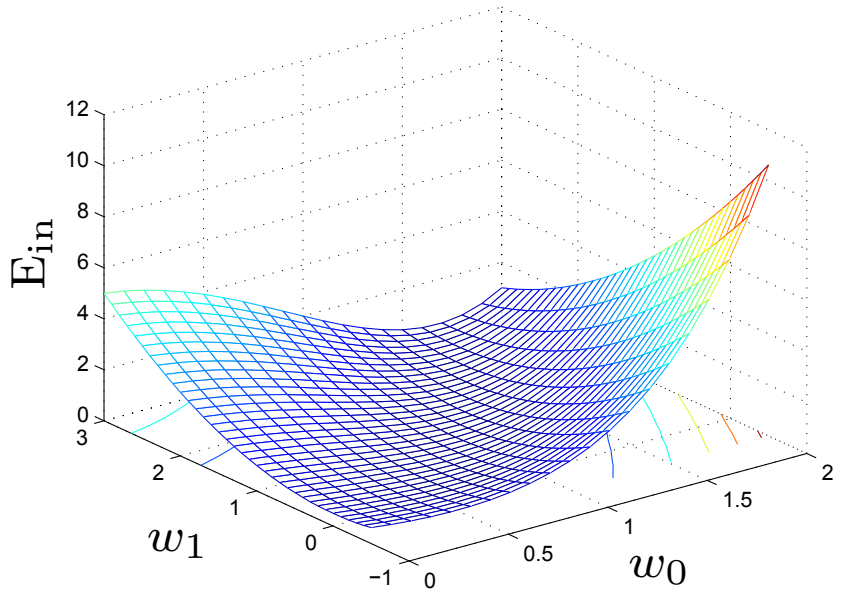
- parameter η has to be specified
  - step size η is called learning rate (가장 중요한 하이퍼 파라미터)
  - typically good choice: around ~~0.1~~ (empirical)

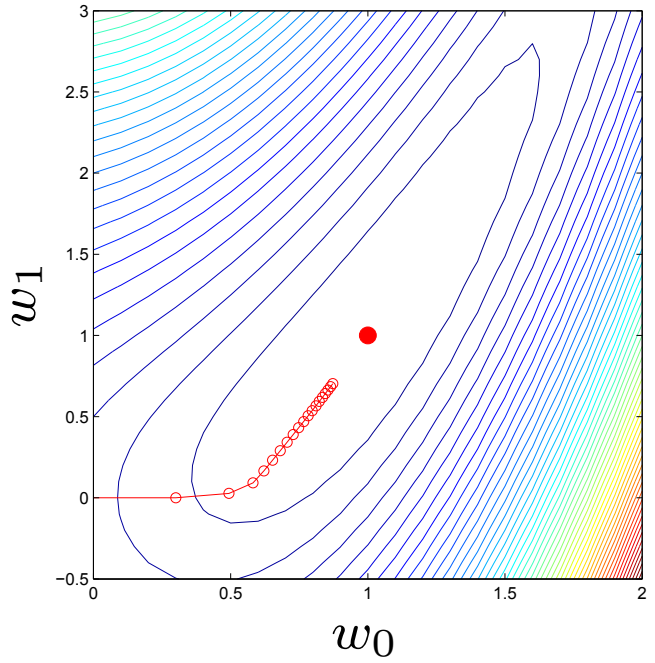
case by case, 작은값

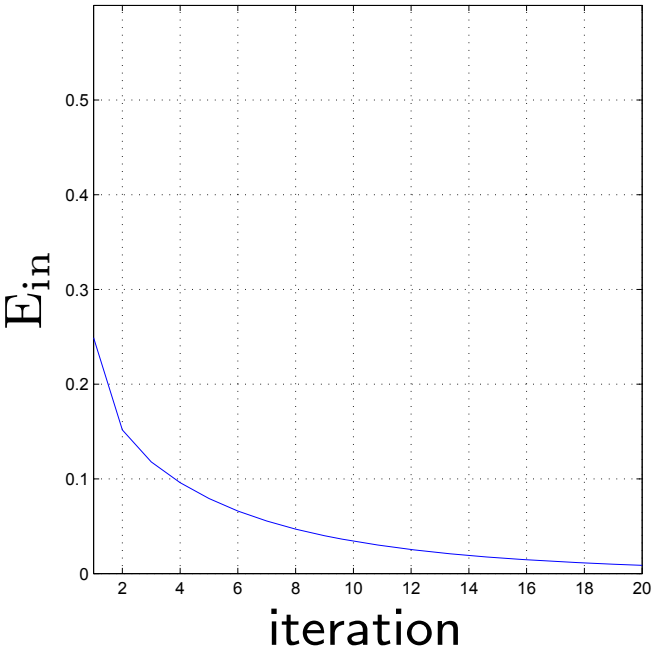
  - how does the value of η affect gradient descent?

Example

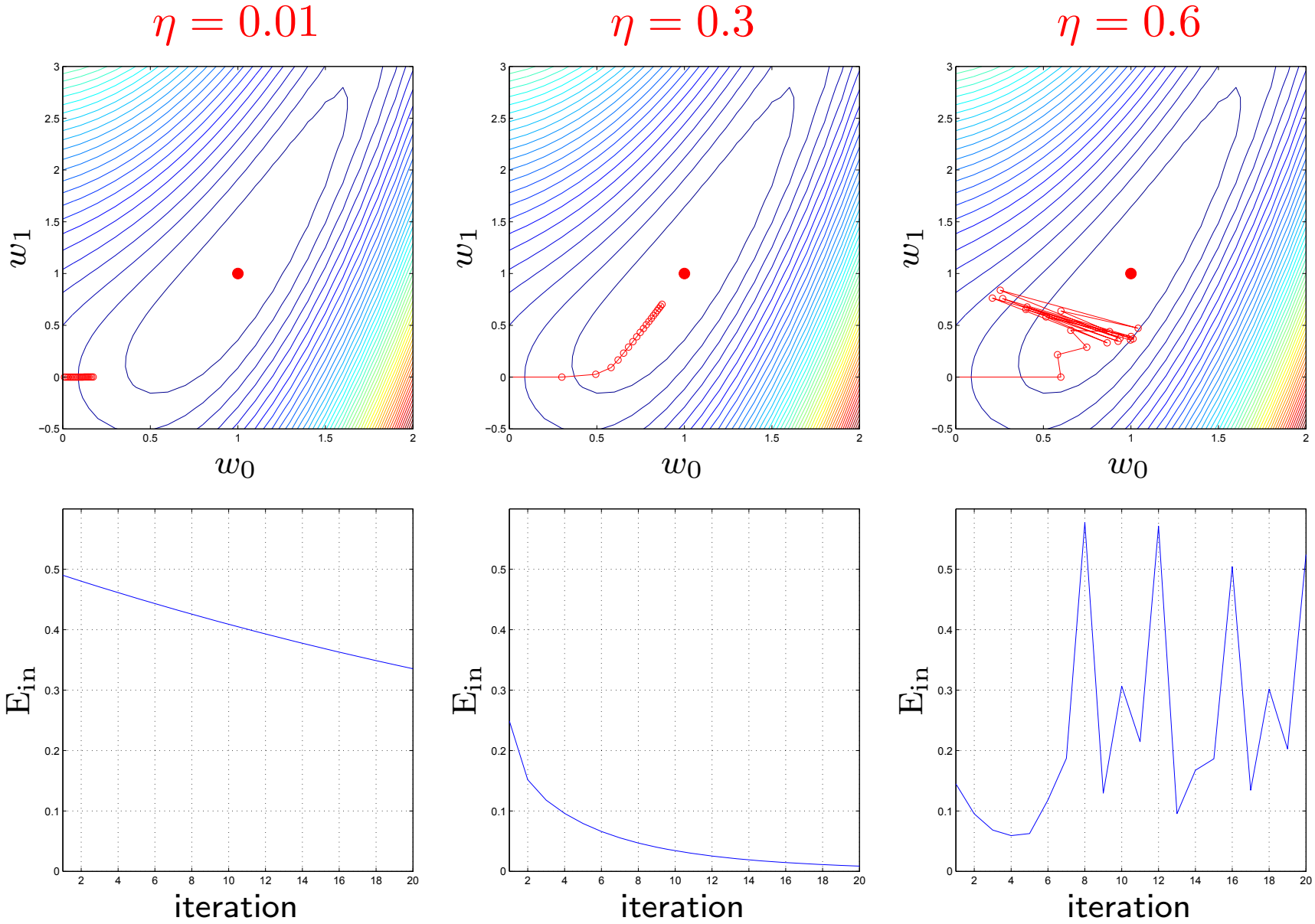
- assume a simple function for E_{in}
  - global minimum 0 at (1,1)





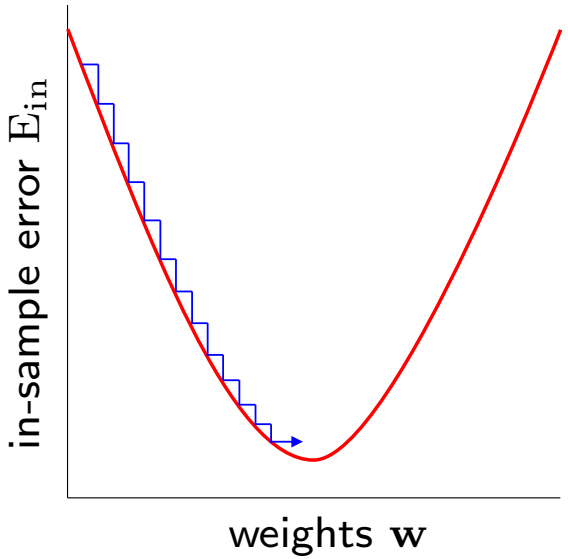


- gradient descent starting at (0,0)
  - # iterations (steps) $= 20$
  - step size (learning rate) $\eta = 0.3$

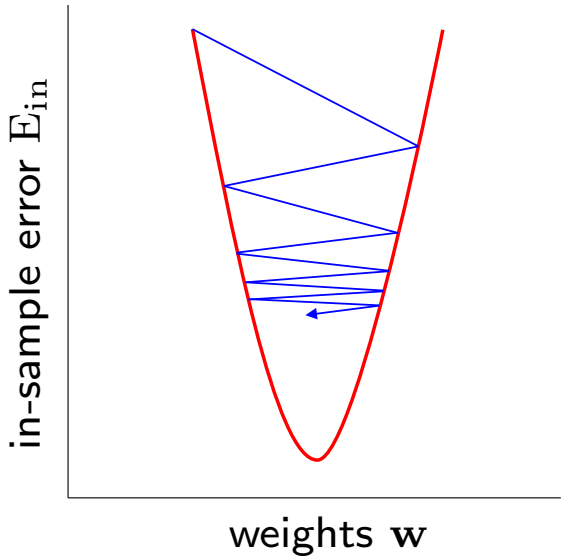

$\eta = 0.01$
$\eta = 0.3$
$\eta = 0.6$
w_1
w_0
$\mathrm{E_{in}}$
iteration

Effect of step size η

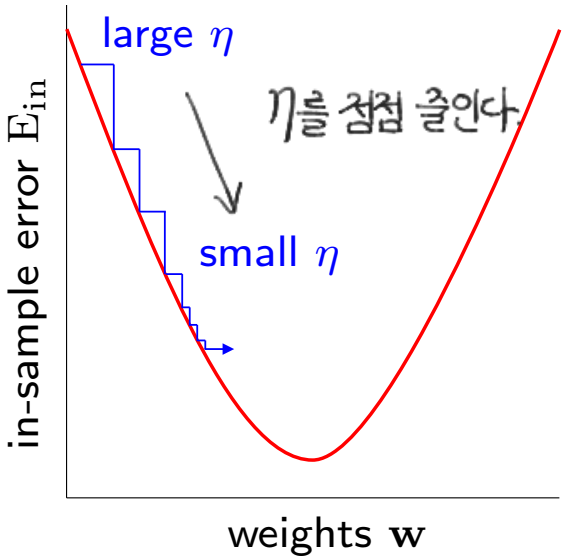
- η affects the convergence behavior of gradient descent:



η too small



η too large



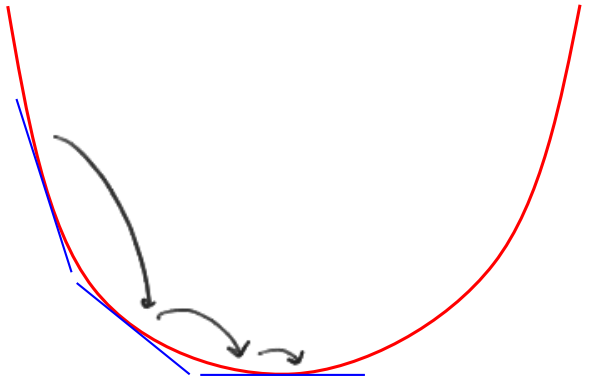
variable η — just right

How to choose η?

- a fixed step size (if too small)
  - insufficient when you are far from the local minimum
- too large a step size when you are close to the minimum
  - leads to bouncing around, possibly even increasing E_{in}
- ideal strategy:
  - take large steps when far from the minimum
    (this is to get in the right ballpark quickly)
  - and then small (more careful) steps when close the minimum
- a simple heuristic can accomplish this

Achieving variable step size η

- observe that the norm of gradient
  - typically large when far from the minimum
  - small when close to the minimum
- thus, we could set $\eta_t = \eta ||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||$
  - to obtain the desired behavior for the variable step size
  - also conveniently cancels normalizing term of $\hat{\mathbf{v}}$ in Eq. (22)



Gradient descent (revised)

- gradient descent algorithm (with redefined learning rate):

1: initialize weights at time step $t = 0$ to $\mathbf{w}(0)$

2: **for** $t = 0, 1, 2, \ldots$ **do**

3: compute gradient $\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))$

4: set the direction to move: $\hat{\mathbf{v}}_t = -\frac{\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))}{||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||}$

5: set learning rate: $\eta_t = \eta ||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||$

6: update weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta_t \hat{\mathbf{v}}_t$

$$\boxed{\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))} \quad (24)$$

7: iterate to next step until it is time to stop

8: return final weights

Remarks

- update rule (24)

$$\boxed{\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))}$$

  - needs to examine all examples at each iteration: $O(N)$ time
  - can be approximated in $O(1)$ by stochastic gradient descent

SGD (모든 ex 다 볼 필요 없게 바꾼것)

- least mean square (LMS) rule (Widrow-Hoff learning rule)
  - when the cost function is mean squared error

$$\begin{aligned} \mathbf{w} &:= \mathbf{w} - \eta \nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}) \\ &= \mathbf{w} + \eta \underbrace{(\mathbf{y} - \mathbf{w}^T \mathbf{x})}_{\text{error}} \cdot \underbrace{\mathbf{x}}_{\text{input}} \end{aligned}$$

Outline

Logistic regression algorithm

1: initialize weights at time step $t = 0$ to $\mathbf{w}(0)$

2: **for** $t = 0, 1, 2, \ldots$ **do**

3: compute the gradient

$$\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t)) = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^{\mathrm{T}}(t) \mathbf{x}_n}}$$

4: set the direction to move: $\mathbf{v}_t = -\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))$

5: update weights: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$

6: iterate to next step until it is time to stop

7: return final weights $\mathbf{w}$

Remarks

- vector $\mathbf{v}_t$ is no longer restricted to unit length
  - due to "cancellation" of $||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||$ term

- we have two more loose ends to tie
  1. how to choose initial weights $\mathbf{w}(0)$
  2. how to set the criterion for stopping gradient descent

Initialization

- in some cases (*e.g.* logistic regression) → convex
  - setting $\mathbf{w}(0) = \mathbf{0}$ works well
- however, in general
  - it is safer to initialize weights randomly
  - to avoid getting stuck on a perfectly symmetric hilltop
- in practice
  - choose each weight independently from a normal distribution with zero mean and small variance

Two basic criteria for termination

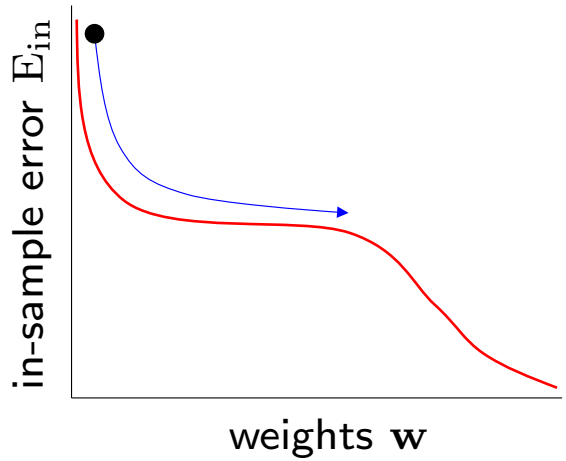
끝내기

- non-trivial topic in optimization
- one simple approach (as in pocket algorithm)
  - set an upper bound on the number of iterations
  - problem: no guarantee on quality of final weights
- another natural termination criterion: 즉, $|\nabla E_{in}|$이 작은곳에서 stop.
  - stop once $||\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}(t))||$ drops below a threshold
  - eventually this must happen, but we do not know when
- for logistic regression, combination works well:
  1. large upper bound for the number of iterations
  2. small lower bound for the size of gradient

Third criterion for termination

- problem with relying solely on the size of gradient
  - you might stop prematurely

- when relatively flat region is reached (more common than we suspect)
  - algorithm will prematurely stop when we may want to continue



- one solution is to require both:

  (i) change in error is small, and (ii) error itself is small

Termination criteria in practice

언제 끝내야 하는가?

- ultimately, a combination of criteria works reasonably well:
  1. maximum number of iterations
  2. marginal error improvement (개선이 ↓)
  3. small value for the error itself (에러 자체가 ↓)

Variants of gradient decent

1st order method

- **batch** gradient descent
  - use (all N examples) in each iteration
- **stochastic** gradient descent
  - use (1 example) in each iteration
- mini-batch gradient descent
  - use b examples in each iteration
  - b: mini-batch size (typically $2 \sim 100$)

Outline

Three linear models

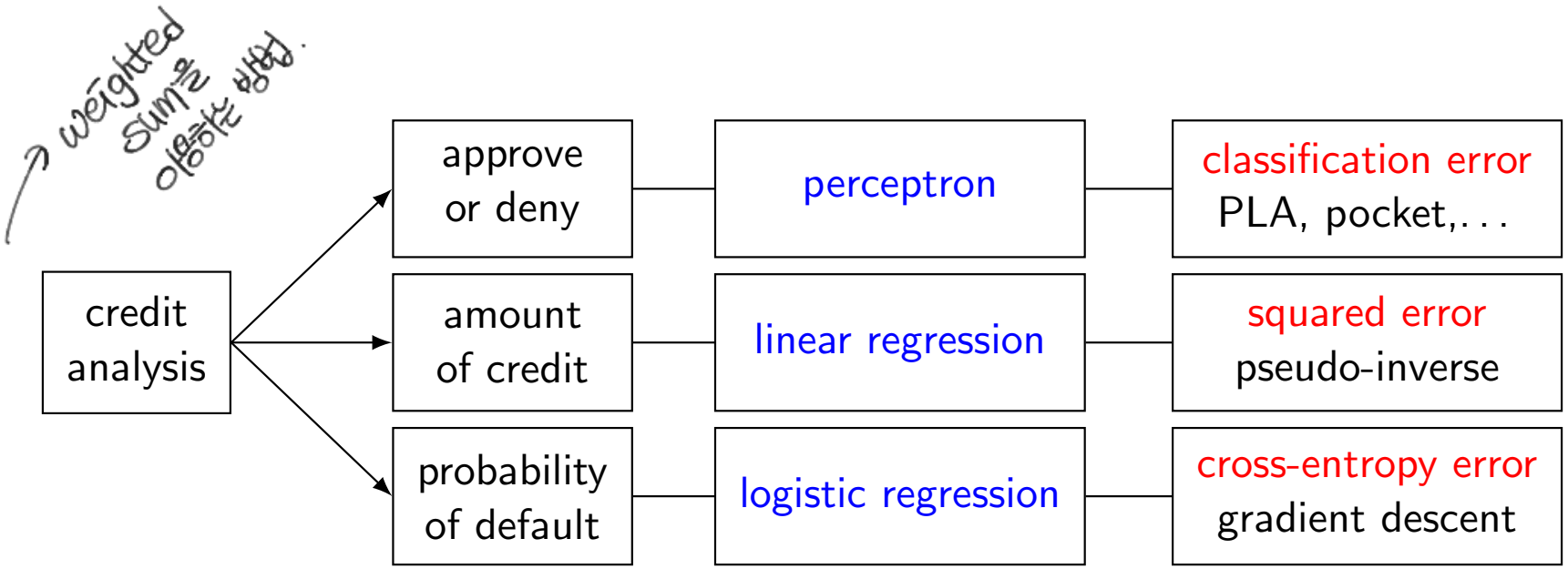


Figure 4: comparison of the three linear models considered

- have their respective goals, error measures, and algorithms
  - nonetheless, share similar sets of linear hypotheses
- you should "*first* try" a linear model when learning from data!

Comparison

	linear classification	linear regression	logistic regression
$\mathcal{Y}$	$\{-1, +1\}$	$\mathbb{R}$	$\{-1, +1\}$, $\mathbb{R}$
$\hat{y} = h(\mathbf{x})$	$\text{sign}(\mathbf{w}^{\text{T}}\mathbf{x})$	$\mathbf{w}^{\text{T}}\mathbf{x}$	$\theta^{\star}(\mathbf{w}^{\text{T}}\mathbf{x})$
e$(\hat{y}, y)$	0-1 loss $[\![\hat{y} \neq y]\!]$	squared error $(\hat{y} - y)^2$	cross-entropy error $[\![y{=}+1]\!] \ln \frac{1}{\hat{y}}$ $+[\![y{=}-1]\!] \ln \frac{1}{1-\hat{y}}$
$\text{E}_{\text{in}}(h)$	$\frac{1}{N}\sum_{n=1}^{N}[\![h(\mathbf{x}_n){\neq}y_n]\!]$	$\frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n){-}y_n)^2$	$\frac{1}{N}\sum_{n=1}^{N}\ln\left(1{+}e^{-y_n\mathbf{w}^{\text{T}}\mathbf{x}_n}\right)$
opt.	combinatorial optimization (NP-hard)	set $\nabla \text{E}_{\text{in}}(\mathbf{w}) = 0$ (closed-form solution exists)	set $\nabla \text{E}_{\text{in}}(\mathbf{w}) = 0$ iterative optimization (*e.g.* gradient descent)

$\star$ logistic sigmoid $\theta(s) = 1/(1 + e^{-s})$

Summary

- logistic regression: estimates probabilities of binary events
  - approximates $\mathbb{P}[y = +1|\mathbf{x}] = f(\mathbf{x})$ by $h(\mathbf{x}) = \theta(\mathbf{w}^{\mathrm{T}}\mathbf{x})$
  - likelihood: $P(y|\mathbf{x}) = h(\mathbf{x})^{[\![y=+1]\!]}(1 - h(\mathbf{x}))^{[\![y=-1]\!]} = \theta(y\mathbf{w}^{\mathrm{T}}\mathbf{x})$
  - training: minimize cross-entropy error iteratively
- gradient descent: iteratively solves unconstrained optimization
  - general technique for minimizing twice-differentiable functions
  - batch gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w})$ [over all data]
  - stochastic gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \eta\nabla \mathrm{e}_n(\mathbf{w})$ [pointwise]
  - mini-batch gradient descent, batch normalization
- linear models should be a first try: simple/robust/work & generalize well

Outline

(Shannon) entropy and relative entropy

- entropy
  - entropy $H(X)$ of discrete rv X with alphabet $\mathcal{X}$ and pmf p:

$$H(X) = H(p) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$

- relative entropy (KL divergence)
  - relative entropy $D(p||q)$ of pmf p wrt pmf q:

$$D(p||q) = \sum_{x} p(x) \log \frac{p(x)}{q(x)}$$

  - measures the information lost when q is used to approximate p

Cross entropy

- cross entropy
  - cross entropy for two pmfs p and q:

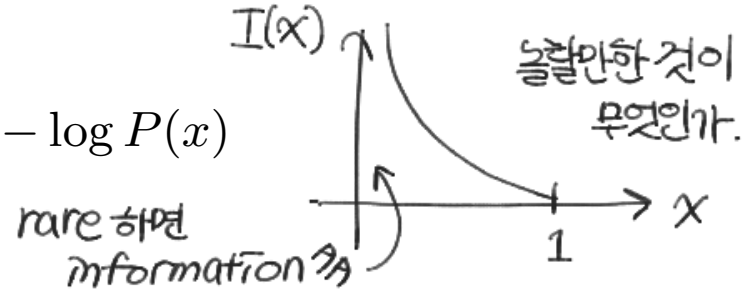
$$\begin{aligned} H(p,q) &= H(p) + D(p||q) \\ &= \sum_x p(x) \left(-\log p(x) + \log p(x) - \log q(x)\right) \\ &= -\sum_x p(x) \log q(x) \end{aligned}$$

  - measures the average number of bits needed to identify an event from a set of possibilities, if a coding scheme is used based on a given probability distribution q, rather than the "true" distribution p

Comparison

- self information:

$$I(x) = -\log P(x)$$



- Shannon entropy:
  - min # of bits per msg needed (on average) to encode events from P (분포)

$$H(P) = \mathbb{E}_{\mathbf{x}\sim P}[I(x)] = -\mathbb{E}_{\mathbf{x}\sim P}[\log P(x)]$$

random valuable
small x가 instance
P는 x가 따르는 분포.

- KL divergence: (두 분포사이 거리)
  - expected # of extra bits per msg to encode events from true P if using an optimal code for Q (rather than P)

Q||P 이 들어 다르다.

$$D_{\mathrm{KL}}(P||Q) = \mathbb{E}_{\mathbf{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = -H(P) - \mathbb{E}_{\mathbf{x}\sim P}[\log Q(x)]$$

- cross entropy:
  - expected # of total bits per msg to encode events from true P if using an optimal code for Q (rather than P)

(P: 고분이 아는. data 발생방법.
Q: 우리가 알고있는것.

P,Q사이의 cross entropy

$$\boxed{H(P,Q) = H(P) + D_{\mathrm{KL}}(P||Q)} = -\mathbb{E}_{\mathbf{x}\sim P}[\log Q(x)]$$

totall bit extra bit