M2177.003100
Deep Learning

[1: Foundations of Deep Learning]

Electrical and Computer Engineering
Seoul National University

(last compiled at 16:00:00 on 2018/09/03)

Outline

References

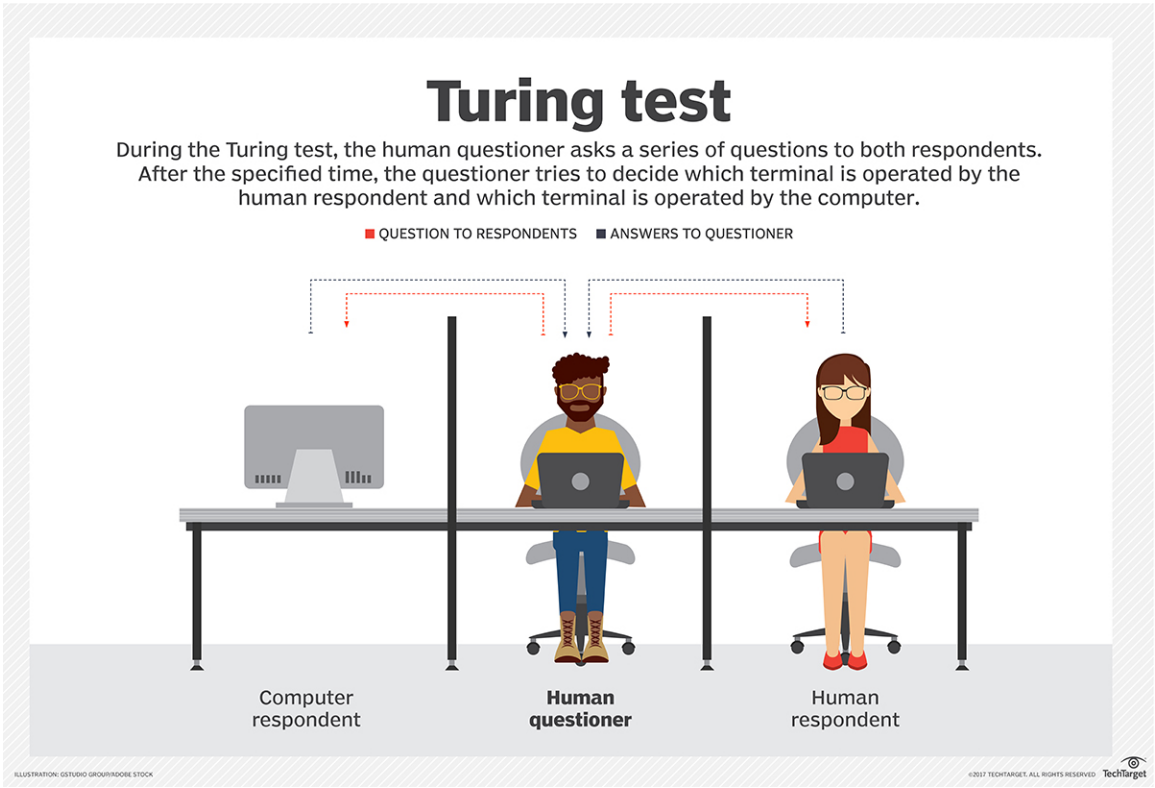
- *Deep Learning* by Goodfellow, Bengio and Courville ▸ Link
  - Chapters 1–5
- online resources:
  - *Deep Learning Specialization (coursera)* ▸ Link
  - *Stanford CS231n: CNN for Visual Recognition* ▸ Link
  - *Machine Learning Yearning* ▸ Link

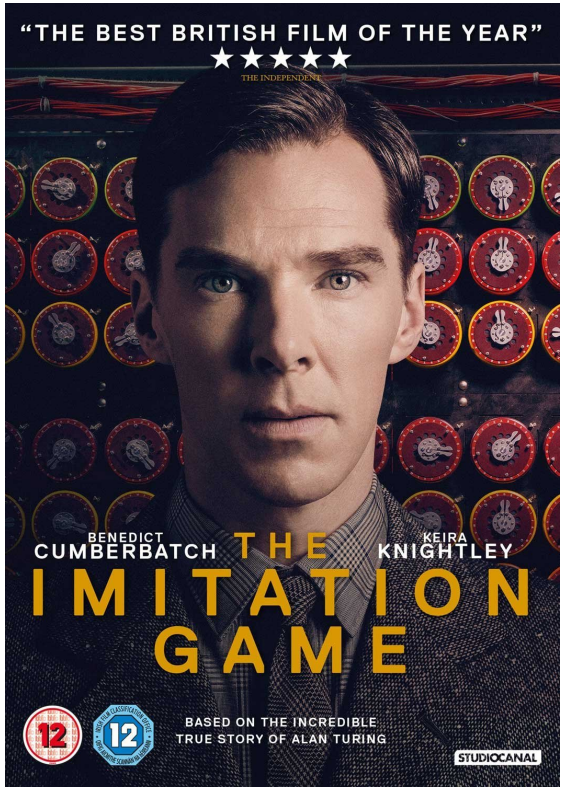
Outline

Artificial intelligence (AI)

- objective
  - to create a machine that can think and/or act like humans
    ↑
    think/act rationally
  - AI = computational intelligence
- rationality in engineering
  - refers to maximizing expected utility
- evaluation metric
  - human-level performance (suggested from day one)

- Turing test: the imitation game metric

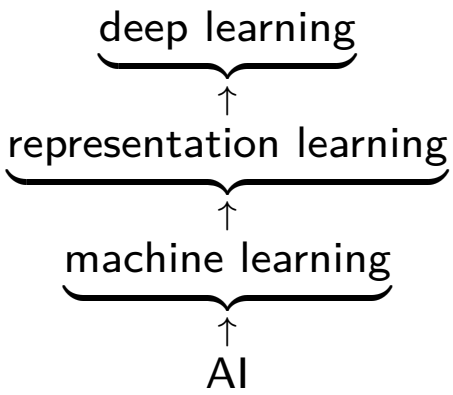


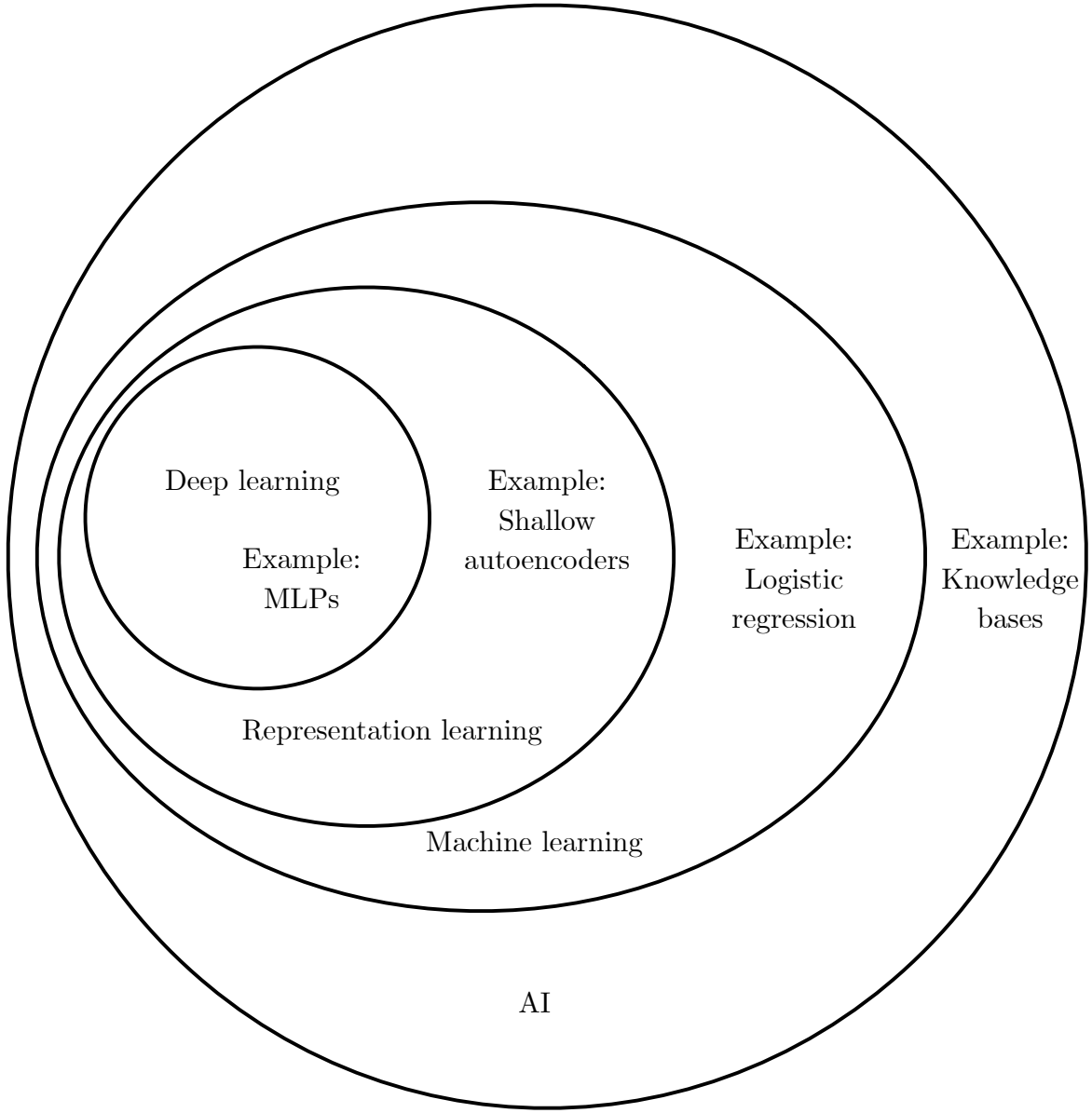


(source: http://searchenterpriseai.techtarget.com)

- Google Duplex ▸ Clip
  - human-level intelligence?

Comparison

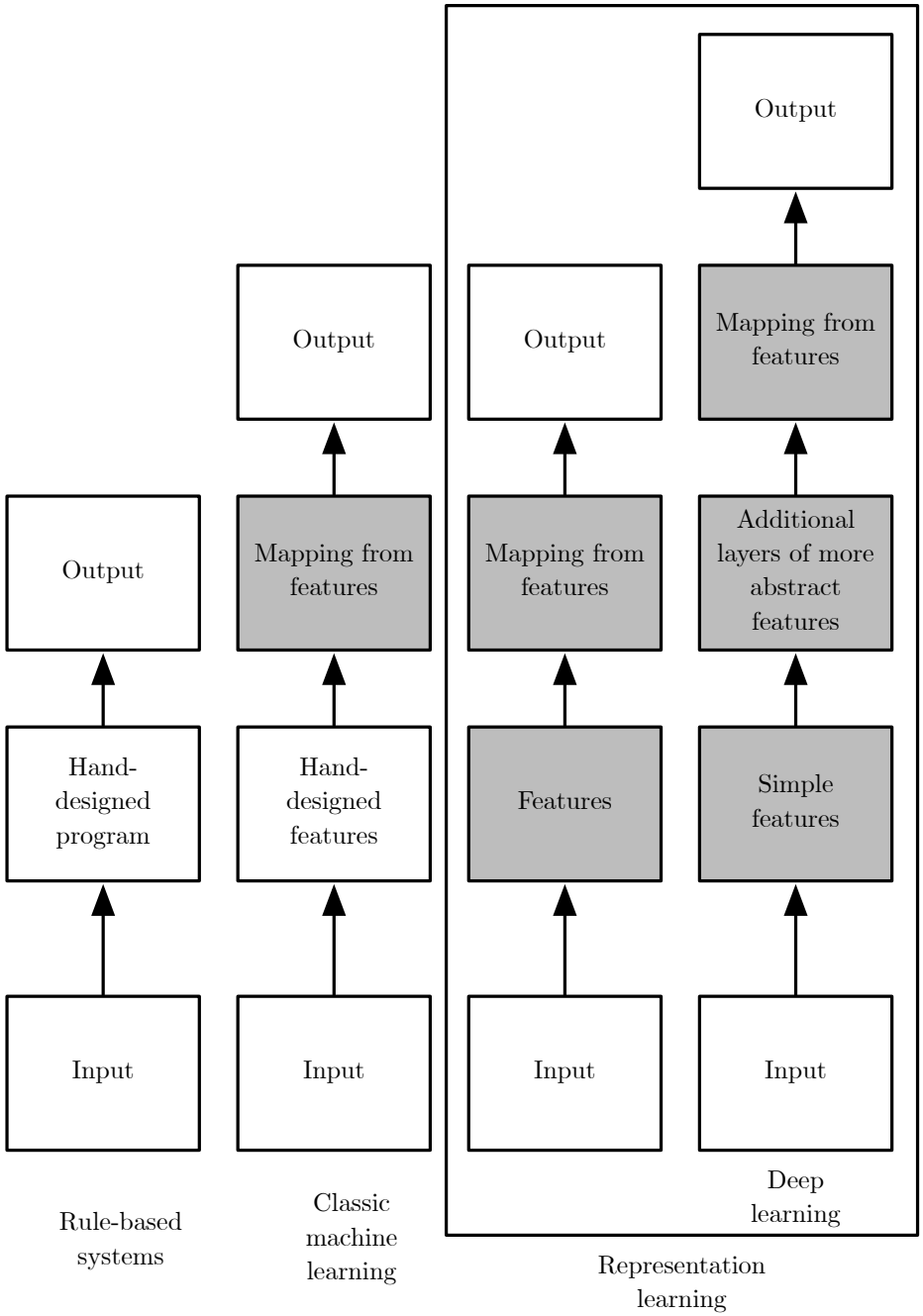

deep learning
representation learning
machine learning
AI


Deep learning
Example:
MLPs
Example:
Shallow
autoencoders
Example:
Logistic
regression
Example:
Knowledge
bases
Representation learning
Machine learning
AI

Deep learning

- hierarchical representation learning
  - implementation: neural nets
  - fueled by big data
  - workhorse: GPU
- each layer in neural nets
  - feature representation
- main applications
  - tasks humans can do well

(shaded boxes: components that are able to learn from data)



Three waves of DL development

1. "cybernetics": 1940's–1960's [neuroscience]
   - perceptron, δ-rule, stochastic gradient decent
   - essentially linear models $\Rightarrow$ limitations (*e.g.* XOR)

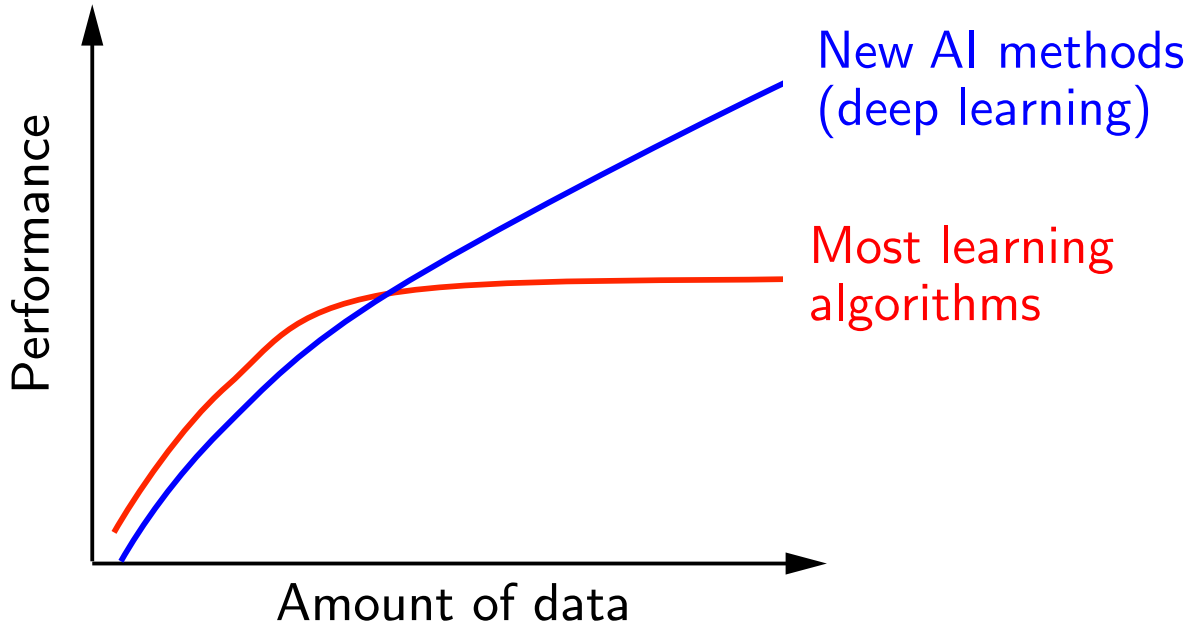
2. "connectionism": 1980's–1990's [cognitive science]
   - simple units achieve intelligence when networked together
   - neural nets, distributed representation[1], back propagation
   - training issues $\Rightarrow$ eclipsed by kernel/graphical models

3. "deep learning": 2006–present [multidisciplinary]
   - early breakthroughs: restricted Boltzmann machine, CNN
   - unsupervised $\rightarrow$ supervised/reinforcement $\rightarrow$ unsupervised ...

[1] each input to a system should be represented by many features, and each feature should be involved in the representation of many possible inputs

Deep learning is in its prime time

- three key driving forces
  1. big data
  2. parallel/distributed computing
  3. advances in algorithms



(source: Ng)

Status quo

- subhuman performance
  - general intelligence
  - domains with small/pricey data, expensive human experts (*e.g.* medical)
- human-level performance
  - some perception tasks: visual/speech recognition
- superhuman performance
  - domains with extremely big data (*e.g.* recommendation, online AD)
  - some perception tasks (*e.g.* massive surveillance), game play

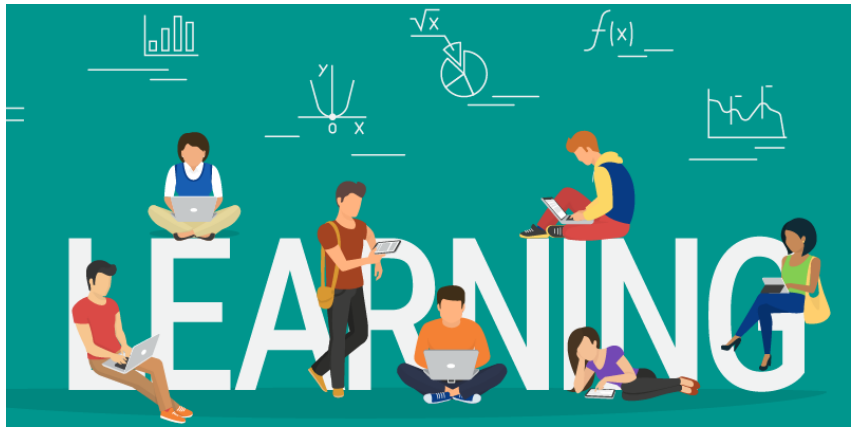
Outline

Machine learning

- learning from data
- what do we mean by learning?
  - Mitchell (1997):

    "A computer program is said to learn from experience E
    with respect to some class of tasks T and performance measure P,
    if its performance at tasks in T,
    as measured by P,
    improves with experience E."

- common types:
  - supervised
  - unsupervised
  - reinforcement
  - many more



Tasks in ML

- described in terms of how to process an **example**
- an "example":
  - a collection of **features** quantitatively measured from object/event
  - represented as a vector $\boldsymbol{x} \in \mathbb{R}^n$ (each entry x_i : a feature)

  e.g. features of an image: pixels values
- common ML tasks:

T1. classification

T2. classification with missing inputs

T3. regression

T4. transcription

T5. machine translation

T6. structured output

T7. anomaly detection

T8. synthesis and sampling

T9. imputation of missing values

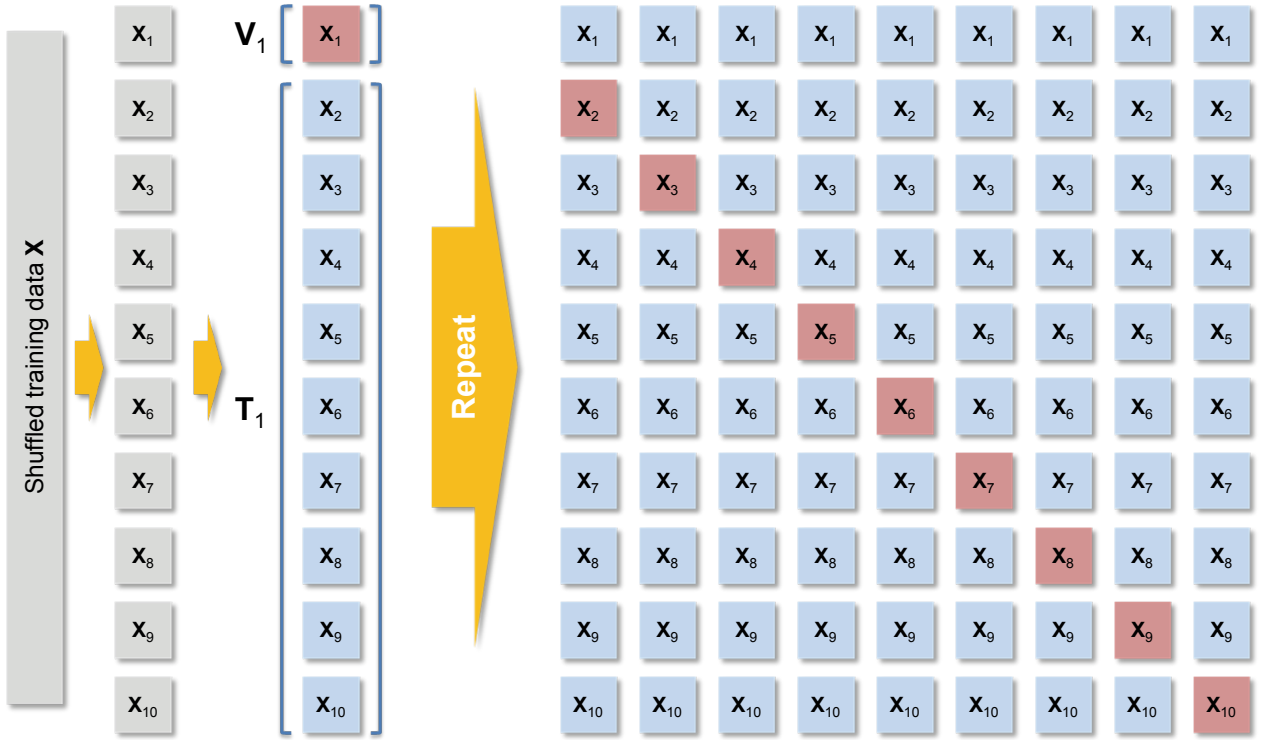
T10. denoising

T11. density/pmf estimation

Data set

- a collection of examples
  - training set: for fitting
  - validation set ("dev set"): for model selection
  - test set: for estimation

10-fold
cross-validation:



Performance measure

- specific to task T

  e.g. classification: accuracy, error rate E ← we focus on this for a while

  density estimation: average log-probability the model assigns to examples

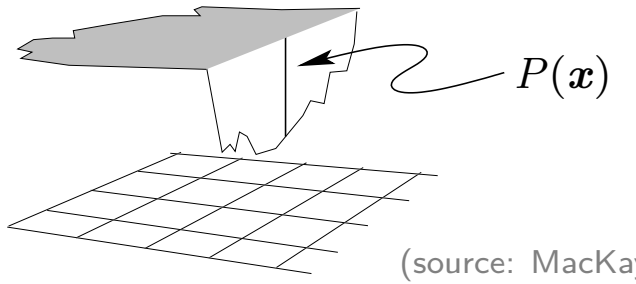
- evaluated using data sets
  - training/dev/test sets $\Rightarrow \mathrm{E}_{\mathrm{train}}$, $\mathrm{E}_{\mathrm{dev}}$, $\mathrm{E}_{\mathrm{test}}$

- often challenging to choose
  1. difficult to decide what to measure

     e.g. penalize frequent mid-sized mistakes or rare large mistakes?
  2. know ideal measure but measurement is undoable

     e.g. density estimation

a lake whose depth at $\boldsymbol{x} = (x, y)$ is $P(\boldsymbol{x})$



(source: MacKay)

Central challenge in ML

- generalization
  - ability to perform well on previously unobserved examples
- generalization error $\mathrm{E}_{\mathrm{gen}}$
  - expected error on a new example $\Rightarrow$ implausible to calculate
- training error $\mathrm{E}_{\mathrm{train}}$
  - measured on a training set $\Rightarrow$ bad proxy for $\mathrm{E}_{\mathrm{gen}}$
- test error $\mathrm{E}_{\mathrm{test}}$
  - measured on a test set (not used in training) $\Rightarrow$ better proxy for $\mathrm{E}_{\mathrm{gen}}$

Two specific objectives

- objective: $\boxed{\mathrm{E}_{\mathrm{gen}} = 0}$ in theory or $\boxed{\mathrm{E}_{\mathrm{test}} \simeq 0}$ in practice
- split into two objectives:
  1. $\mathrm{E}_{\mathrm{test}} \simeq \mathrm{E}_{\mathrm{train}}$
  2. $\mathrm{E}_{\mathrm{train}} \simeq 0$
- objective 1: make $\mathrm{E}_{\mathrm{test}} \simeq \mathrm{E}_{\mathrm{train}}$
  - failure: overfitting $\rightarrow$ high variance
  - cure: regularization, more data + lower capacity(layer수나 feature수를 줄임)
- objective 2: make $\mathrm{E}_{\mathrm{train}} \simeq 0$
  - failure: underfitting $\rightarrow$ high bias
  - cure: optimization, more complex model

Capacity of a model

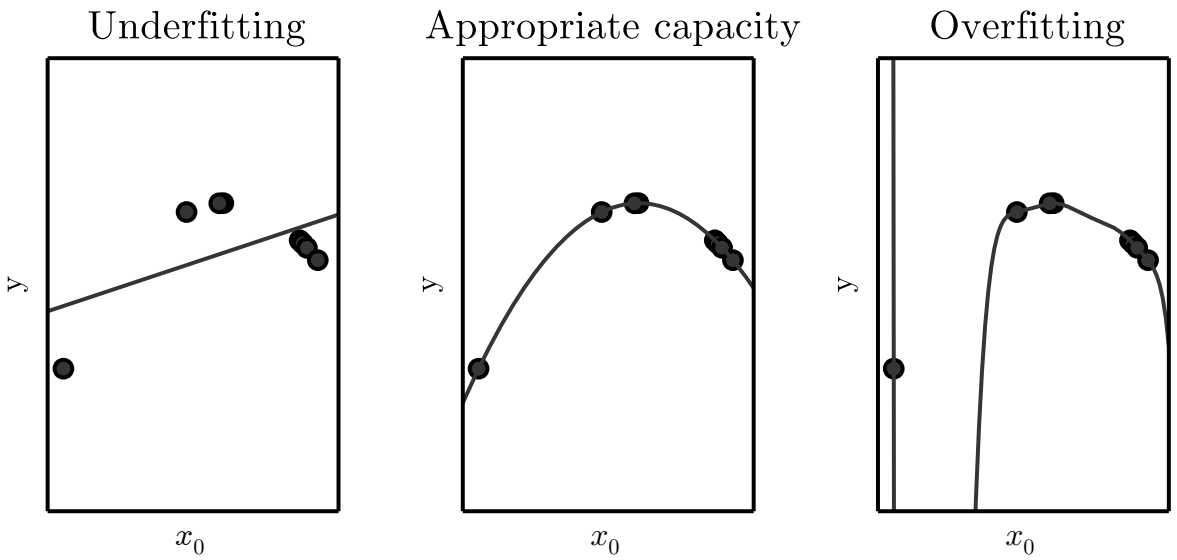
capacity 조절으로 under/overfitting 조절 가능

- the ability of the model to fit various functions

  ↑

  representation (+ learning algorithm)

- altering capacity controls over/underfitting
  - example (truth: quadratic; fit: linear, quadratic, degree-9)



Choosing a model (conventional advice)

전통적 : simple한게 제일 좋음
현대적 : 복잡한 모델 쓰고 regularization 잘하는게 좋음

- Occam's razor (a principle of parsimony)
  - among competing hypotheses, choose the "simple" one
- why? **VC generalization bound**: for any $\epsilon > 0$ and $N > 0$

$$\mathbb{P}[\ \overbrace{|\mathrm{E}_{\mathrm{train}}(f) - \mathrm{E}_{\mathrm{test}}(f)|}^{\text{generalization gap}} > \epsilon\ ]\ \leq\ \underbrace{4 \cdot (2N)^{\overbrace{d_{\mathrm{VC}}}^{\text{capacity}}} \cdot e^{-\frac{1}{8}\epsilon^2 N}}_{\text{VC bound}}$$

(bad event)

  - N : # of training examples
  - f : a model (d_{VC} : its *VC dimension*, a measure of model capacity)
- in words: discrepancy between $\mathrm{E}_{\mathrm{train}}$ and $\mathrm{E}_{\mathrm{test}}$
  - grows as model capacity grows

(but shrinks as N increases)
↑
power of big data

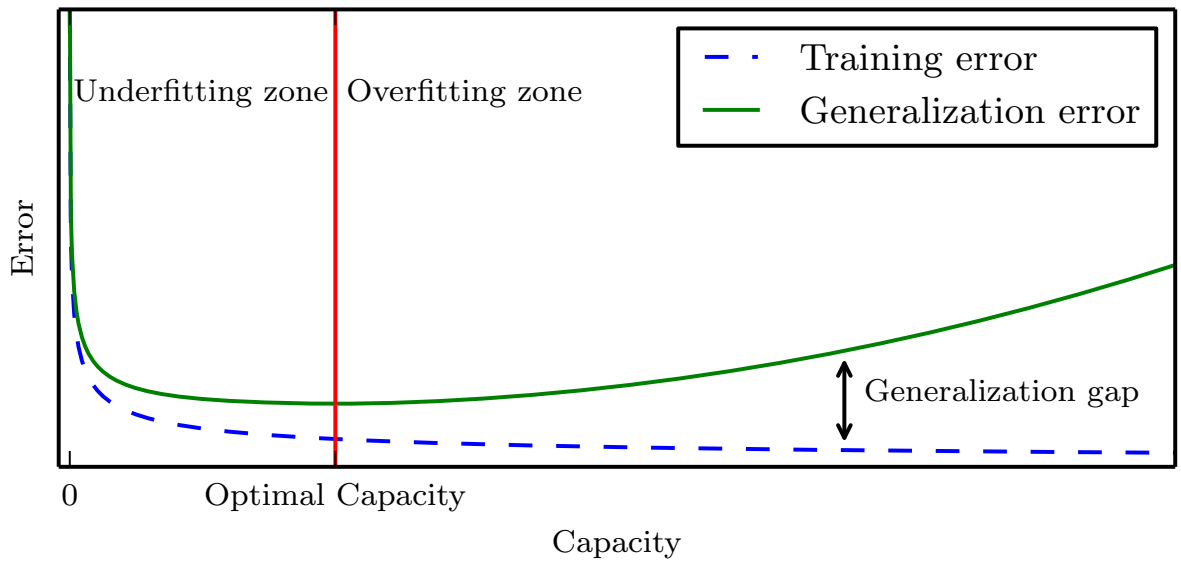
A tradeoff: the main challenge in ML

- approximation-generalization tradeoff or bias-variance tradeoff

complex model is better

$$\underbrace{\mathrm{E}_{\text{test}} \simeq \overbrace{\mathrm{E}_{\text{train}}}} \simeq 0$$

simple model is better



- in theory: choose simpler functions
  - better generalization (smaller gap between training/test error)
- in practice: must still choose a sufficiently complex hypothesis
  - to achieve low training error

Two major weapons to fight the tradeoff

- **optimization**: bias reduction (better approximation)
  - finds model parameters that minimize error

  e.g. stochastic gradient descent

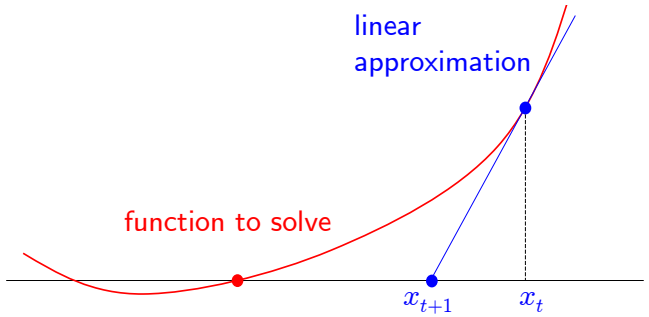
- **regularization**: variance reduction (better generalization)
  - constrains model capacity by reflecting prior knowledge

  e.g. dropout, weight decay

Optimization examples

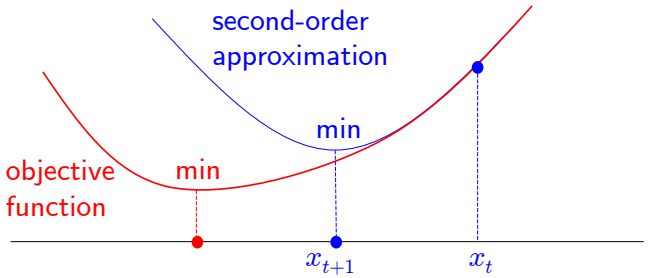
- Newton's method
  - zero-finding

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$



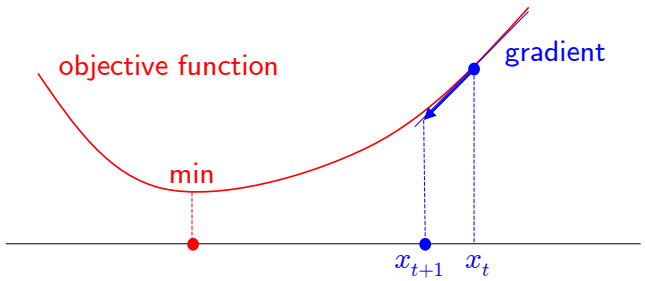
  - minimization/maximization

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$$



- gradient descent
  - minimization/maximization

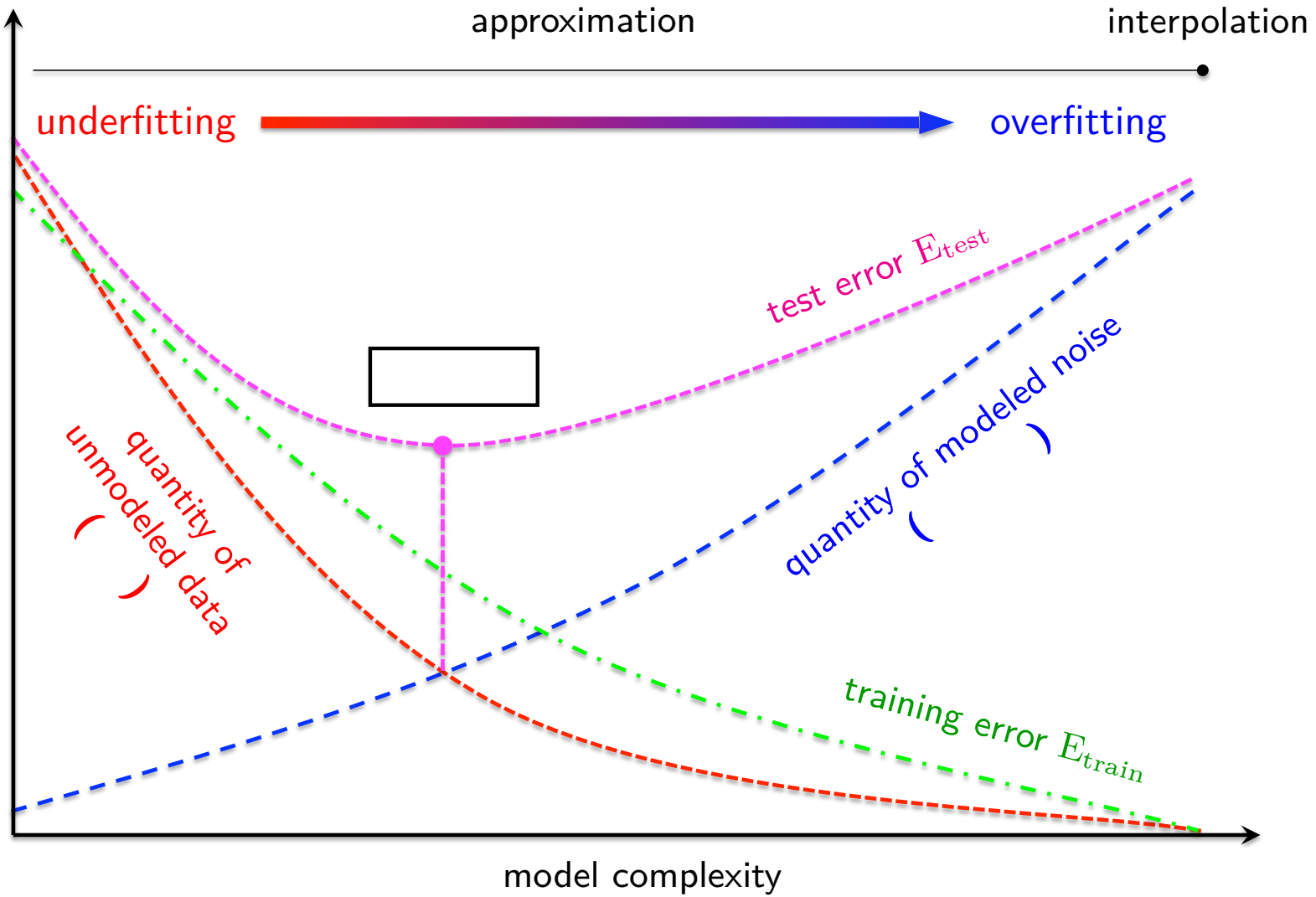
$$x_{t+1} = x_t - \epsilon f'(x_t)$$



Choosing a model (modern advice)

- complex model + effective regularization + big data
- complex model
  - higher chance of fitting data $\rightarrow \mathrm{E}_{\mathrm{train}} \simeq 0$
- regularization + big data
  - reduces generalization gap $\rightarrow \mathrm{E}_{\mathrm{test}} \simeq \mathrm{E}_{\mathrm{train}}$

Big picture


approximation
interpolation
underfitting
overfitting
test error $\mathrm{E}_{\mathrm{test}}$
quantity of modeled noise
quantity of unmodeled data
training error $\mathrm{E}_{\mathrm{train}}$
model complexity

Outline

Linear models

- basis for more sophisticated models
- has many advantages $\rightarrow$ worth trying first
  - simplicity: easy to implement, test, and interpret
  - generalization: higher chance of $\mathrm{E}_{\mathrm{test}} \simeq \mathrm{E}_{\mathrm{train}}$ than complex models
  - extension: nonlinear transform, kernel trick, neural nets
- can solve three important problems
  1. classification
  2. regression
  3. probability estimation (*aka* logistic regression)
     - come with different but related algorithms

Example: credit card application

- given:
  - applicant information $\longrightarrow$
- decide:
  - approve a credit card or not?

feature	value
age	23 years
gender	female
annual salary	$30,000
years in residence	1 year
years in job	1 year
current debt	$15,000
...	...

Formalization

- let $\mathcal{X} = \mathbb{R}^d$ be the input space
  - $\mathbb{R}^d$: the d-dimensional Euclidean space
  - input vector $\mathbf{x} \in \mathcal{X}$: $\mathbf{x} = (x_1, x_2, \ldots, x_d)$
- let $\mathcal{Y} = \{+1, -1\}$ be the output space
  - denotes a binary yes/no decision
- in our credit example
  - coordinates of input $\mathbf{x}$:
    salary, debt, and other fields in a credit card application
  - binary output y: approved or denined

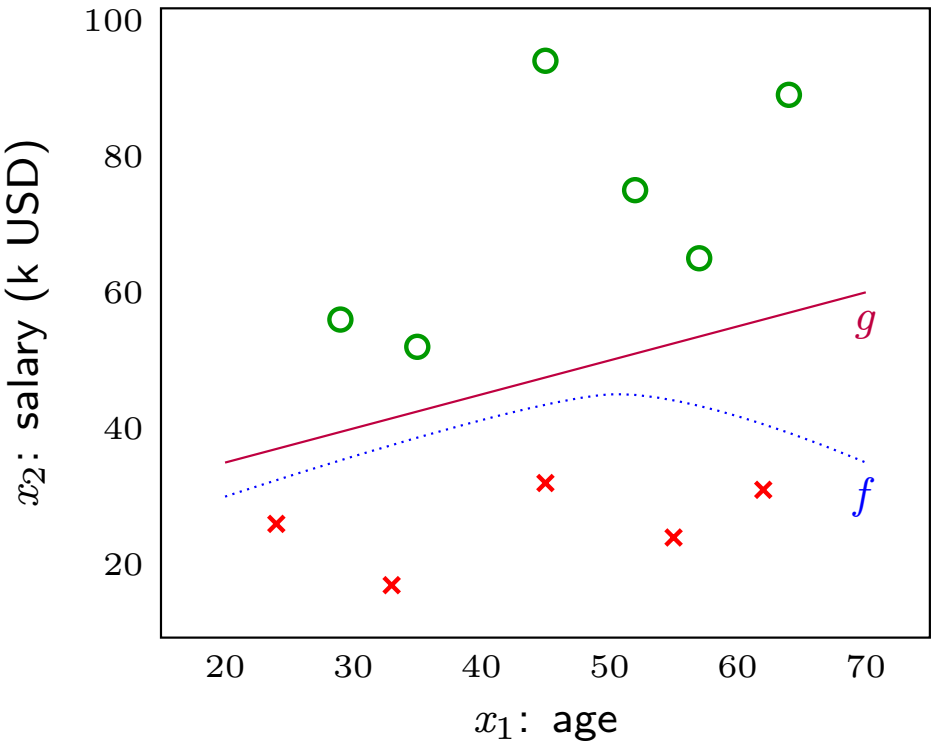
component	symbol	credit approval metaphor
input	$\mathbf{x}$	customer application
output	y	approve or deny
target function	$f : \mathcal{X} \to \mathcal{Y}$	ideal approval formula
data	$(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})$	historical records
hypothesis	$g : \mathcal{X} \to \mathcal{Y}$	formula to be used

- f: unknown target function
- $\mathcal{X}$: input space (set of all possible inputs $\mathbf{x}$)
- $\mathcal{Y}$: output space (set of all possible outputs)
- N: the number of input-output examples (*i.e.* training examples)
- $\mathbb{X} \triangleq \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$: data set where $y^{(n)} = f(\mathbf{x}^{(n)})$

Example

- $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ where x_1: age and x_2: annual salary in USD
- $N = 11$, $d = 2$, $\mathcal{X} = \mathbb{R}^2$, and $\mathcal{Y} = \{\text{approve}, \text{deny}\}$
- data set $\mathcal{D}$:

n	x_1	x_2	y
1	29	56k	approve
2	64	89k	approve
3	33	17k	deny
4	45	94k	approve
5	24	26k	deny
6	55	24k	deny
7	35	52k	approve
8	57	65k	approve
9	45	32k	deny
10	52	75k	approve
11	62	31k	deny



Decision making

- to make a decision
  - weighted coordinates are combined to form a 'credit score'
  - the resulting score is then compared to a threshold
- in our credit card approval example
  - for input $\mathbf{x} = (x_1, \ldots, x_d)$, 'attributes of an applicant':

approve the application if $\sum_{i=1}^{d} w_i x_i > \text{threshold}$

deny the application if $\sum_{i=1}^{d} w_i x_i < \text{threshold}$

The perceptron

- this linear formula can be written more compactly:

$$g(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \tag{1}$$

$$= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right) \tag{2}$$

where b is called the bias and $\text{sign}(z)$[2] $= \begin{cases} +1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \end{cases}$

- this model: called the **perceptron**
  - a simple linear classifier

[2] value of $\text{sign}(z)$ when $z = 0$ is a simple technicality we can ignore

- different parameters $\boldsymbol{\theta} = (\underbrace{w_1, w_2, \ldots, w_d}_{\text{weights}}, \underbrace{b}_{\text{bias}})$
  - yield different hyperplanes $w_1 x_1 + w_2 x_2 + \cdots w_d x_d + b = 0$
- for simplification
  - treat bias b as a weight $w_0 \equiv b$
  - introduce an artificial coordinate x0 is 1
- with this convention, $\mathbf{w}^\top \mathbf{x} = \sum_{i=0}^{d} w_i x_i$
  - this gives the perceptron in vector form:

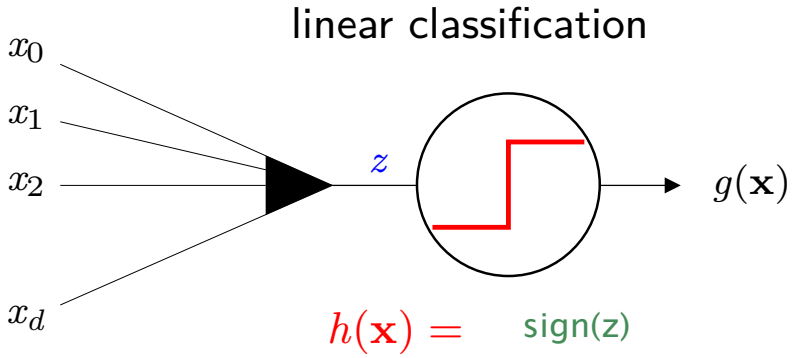
$$\boxed{g(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x})} \tag{3}$$

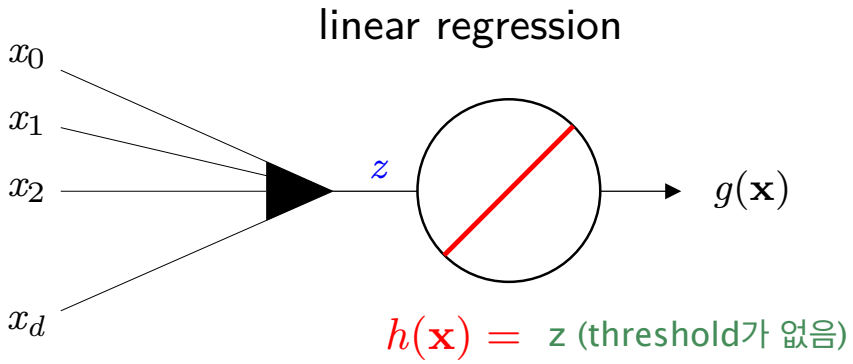
  - $\mathbf{w}^\top \mathbf{x}$: called signal

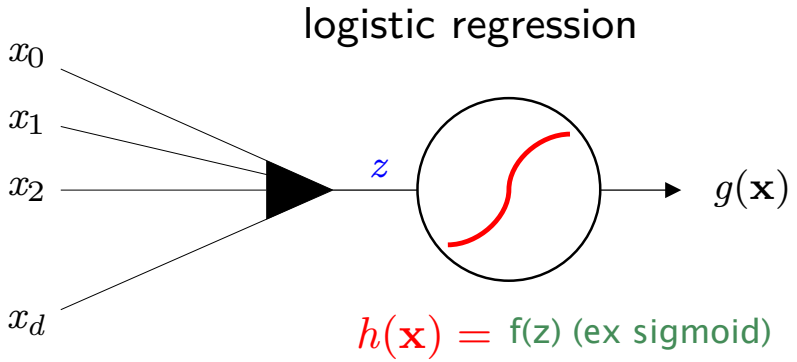
Linear models

- based on "signal" z:

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







Comparison

	linear classification	linear regression	logistic regression
$\mathcal{Y}$	$\{-1, +1\}$	$\mathbb{R}$	$\{-1, +1\}$ 0 <= output <= 1
$\hat{y} = g(\mathbf{x})$	$\text{sign}(\mathbf{w}^\top \mathbf{x})$	$\mathbf{w}^\top \mathbf{x}$	$\theta^\star(\mathbf{w}^\top \mathbf{x})$
$\text{e}(\hat{y}, y)$	0-1 loss $[\![\hat{y} \neq y]\!]$	squared error $(\hat{y} - y)^2$	cross-entropy error $[\![y{=}+1]\!] \ln \frac{1}{\hat{y}} + [\![y{=}-1]\!] \ln \frac{1}{1-\hat{y}}$
$\text{E}_{\text{train}}(h)$	$\frac{1}{N} \sum_{n=1}^{N} [\![h(\mathbf{x}^{(n)}) \neq y^{(n)}]\!]$	$\frac{1}{N} \sum_{n=1}^{N} (h(\mathbf{x}^{(n)}) - y^{(n)})^2$	$\frac{1}{N} \sum_{n=1}^{N} \ln\left(1 + e^{-y^{(n)} \mathbf{w}^\top \mathbf{x}^{(n)}}\right)$
training	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 $\theta(z) = 1/(1 + e^{-z})$

Outline

Miscellaneous

- scalar a, vector $\boldsymbol{a}$, matrix $\boldsymbol{A}$, tensor $\mathbf{A}$
- element-wise product (aka inner product)

$$\boldsymbol{A} \odot \boldsymbol{B}$$

- broadcasting: implicit copying of a vector to many locations

  e.g. $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{b}$

  where $C_{i,j} = A_{i,j} + b_j$ (*i.e.* vector b is added to each row of matrix $\boldsymbol{A}$)

- $f = (\boldsymbol{x}; \boldsymbol{\theta})$: f is a function of $\boldsymbol{x}$ parameterized by $\boldsymbol{\theta}$
  - neural net parameters: weight and bias
- parameter vs hyperparameter
  - learn by machine vs set by human

Outline

Information theory

- can quantify how much information is present in a signal
  - originally invented in communication

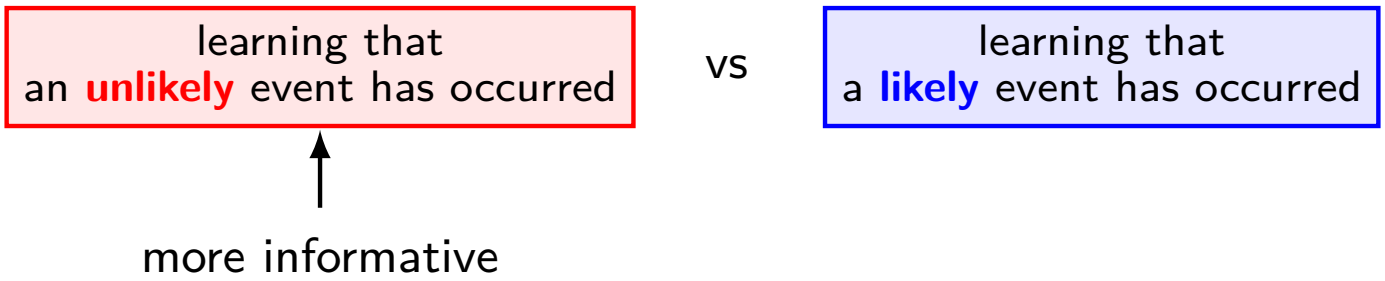
  i.e. send messages from discrete alphabets over a noisy channel

usage:

- communication
  - design optimal codes
  - calculate the expected length of messages sampled from specific probability distributions using various encoding scheme

- ML
  - characterize probability distributions
  - quantify similarity between them

Basic intuition



- example:
  - "the sun rose this morning": so uninformative $\Rightarrow$ unnecessary to send
  - "a solar eclipse occurred this morning": very informative

- we quantify information in a way that formalizes this intuition
  - certain events: no information content
  - likely events: low information content
  - less likely events: higher information content
  - independent events: additive information[3]

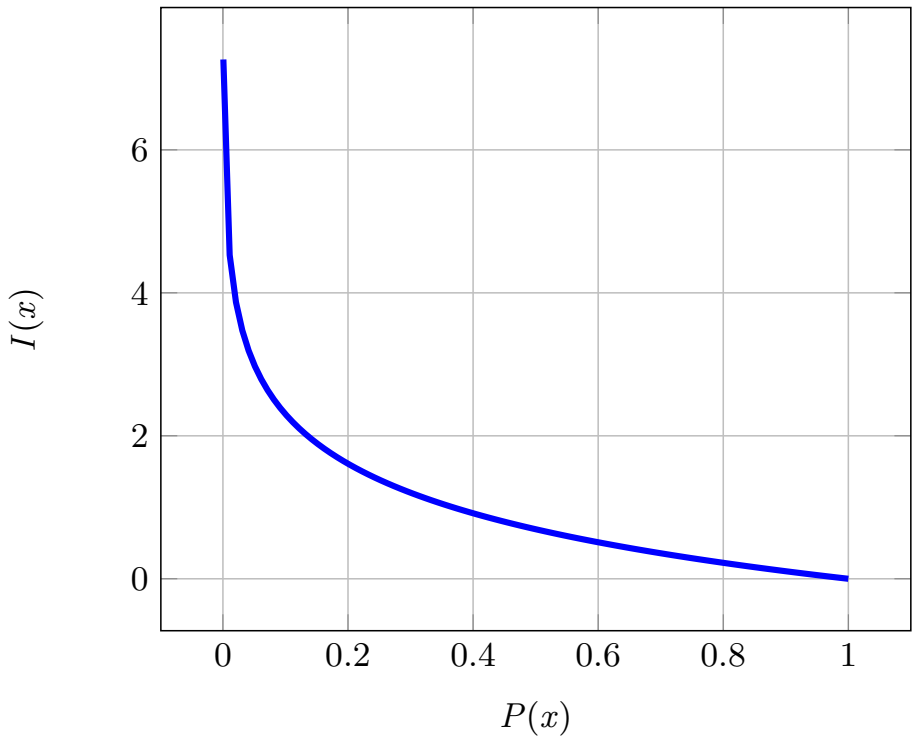
[3] *e.g.* finding out that a tossed coin has come up as heads twice should convey twice as much information as finding out that a tossed coin has come up as heads once

Self-information

- to satisfy all these properties, we define **self-information** of an event $\mathrm{x} = x$:

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

  - unit: **nats** (base e) or bias (base 2)



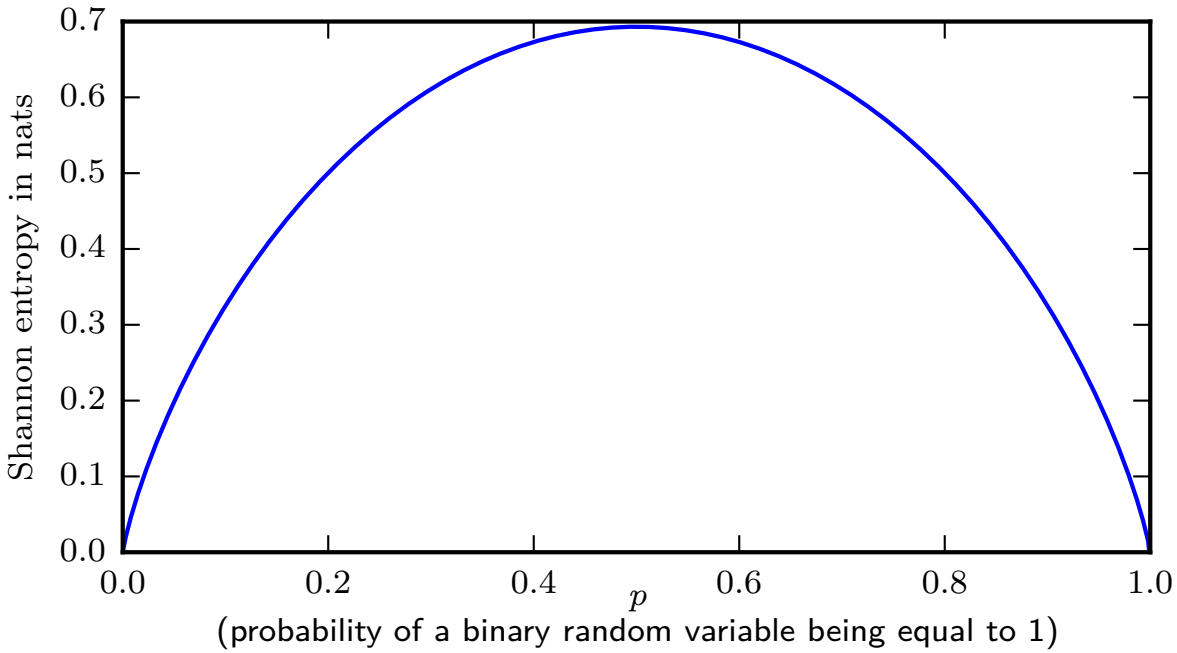
Shannon entropy

- self-information deals only with a single outcome
- **Shannon entropy** allows us to quantify
  - the amount of uncertainty in an entire probability distribution

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

- $H(P)$: Shannon entropy of a distribution P
  - expected amount of information in an event drawn from P
  - gives a *lower bound* on the *number of bits needed* on an average to encode symbols drawn from P

- example: low vs high entropy Bernoulli distributions (parameter p)
  - Shannon entropy: $-(1-p)\log(1-p) - p\log p$



  - low entropy: distributions that are nearly deterministic (p: near 0 or 1)
  - high entropy: distributions that are closer to uniform (max at $p = 0.5$)

Kullback-Leibler (KL) divergence (relative entropy)

- assume: two separate probability distributions $P(\mathrm{x})$ and $Q(\mathrm{x})$
- **KL divergence** can measure how different these two distributions are:

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

- for discrete variables: $D_{\mathrm{KL}}(P \,||\, Q)$ gives
  - extra amount of information needed to send a message (↑ contains symbols drawn from P)

  when using a code designed to minimize the length of messages (↑ contain symbols drawn from Q)

Properties of KL divergence

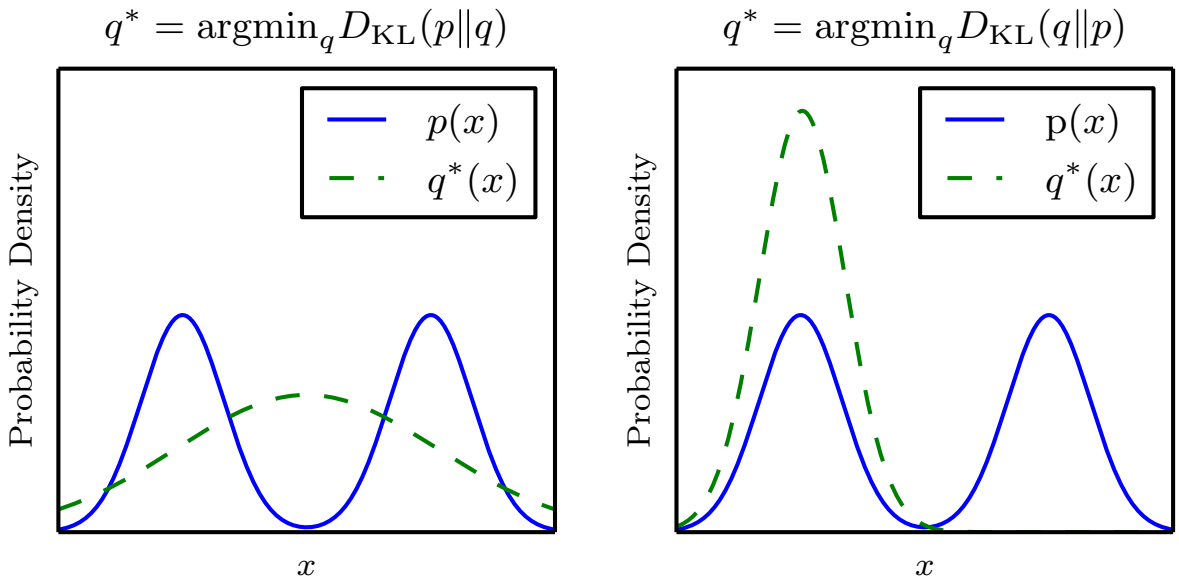
- non-negative
- zero if and only if P and Q are
  - the same distribution (discrete variables)
  - equal "almost everywhere" (continuous variables)
- conceptualized as measuring distance between two distributions
- not symmetric: $D_{\mathrm{KL}}(P \,||\, Q) \neq D_{\mathrm{KL}}(Q \,||\, P)$

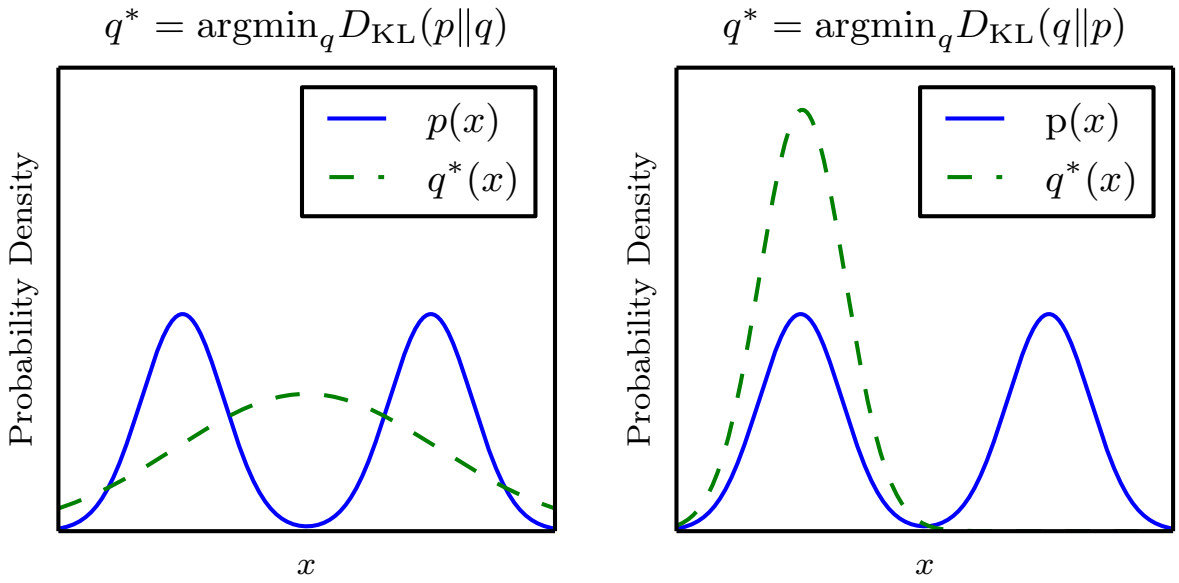
  $\Rightarrow$ not a true distance measure

- task: approximate distribution $\overbrace{p(x)}^{\text{truth}}$ with distribution $\overbrace{q(x)}^{\text{approximatation}}$
  - dilemma: minimize $D_{\mathrm{KL}}(p \,||\, q)$ or $D_{\mathrm{KL}}(q \,||\, p)$?

- choice: problem -dependent
  - some apps require: high $q(x)$ anywhere $p(x)$ is high $\Rightarrow \min D_{\mathrm{KL}}(p \,||\, q)$
  - others require: rarely high $q(x)$ anywhere $p(x)$ is low $\Rightarrow \min D_{\mathrm{KL}}(q \,||\, p)$

e.g. p : a mixture of two Gaussians, q : a single Gaussian





p: truth, q: approximation	$D_{\text{KL}}(p \,\|\| \, q)$	$D_{\text{KL}}(q \,\|\| \, p)$
names	forward	reverse
	inclusive	exclusive
characterizing properties	mean-seeking	mode-seeking
	zero avoiding	zero forcing
normalization wrt p	required	not required
computation	inconvenient	convenient

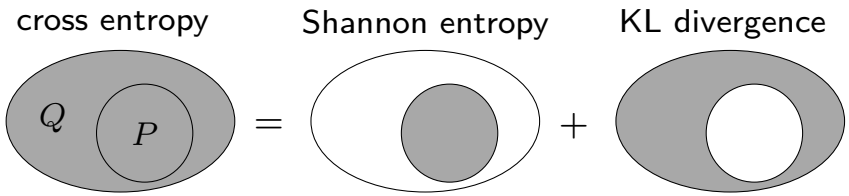
- more details: ▸ Link 1 ▸ Link 2
- some generative models minimize $D_{\text{KL}}(q \,\|\| \, p) \Rightarrow$ mode collapsing issue

Cross-entropy

- defined for two probability distributions P and Q:

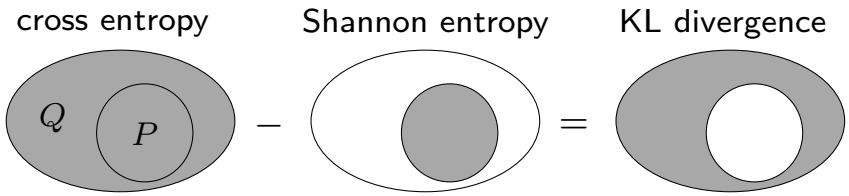
$$\begin{aligned} H(P, Q) &= H(P) + D_{\mathrm{KL}}(P \,||\, Q) \\ &= -\mathbb{E}_{\mathrm{x} \sim P}[\log Q(x)] \end{aligned}$$

  - closely related to KL divergence



- minimizing cross-entropy wrt Q = minimizing KL divergence
  - this is also related to **maximum** likelihood **estimation**

Comparison



- self information:

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

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

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

- KL divergence (relative entropy):
  - expected # of extra bits per message to encode events from true P if using an optimal code for Q (rather than P)

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

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

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

Outline

Maximum likelihood estimator

- consider a set of m examples $\mathbb{X} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$
  - drawn independently from $\underbrace{p_{\text{data}}(\mathbf{x})}_{\uparrow}$

the true but unknown data generating distribution

- let $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$ be a parametric family of $\underbrace{\text{probability distributions}}_{\uparrow}$

(over the same space as p_{data}, indexed by $\boldsymbol{\theta}$)

any configuration $\boldsymbol{x}$ $\xrightarrow{p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})}$ a real number $\hat{p}_{\text{data}}(\boldsymbol{x})$ [estimator of true prob $p_{\text{data}}(\boldsymbol{x})$]

- maximum likelihood estimator for $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_{\mathrm{ML}} = \operatorname*{argmax}_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X}; \boldsymbol{\theta}) \tag{4}$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{5}$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) \tag{6}$$

$$= \operatorname*{argmax}_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \boxed{\operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}; \boldsymbol{\theta})} \tag{7}$$

(4) $\rightarrow$ (5): from independence assumption

(5) $\rightarrow$ (6): to get more convenient but equivalent optimization

(6) $\rightarrow$ (7): optimization criterion is expressed as an expectation

- wrt the empirical distribution $\hat{p}_{\mathrm{data}}$ defined by training data

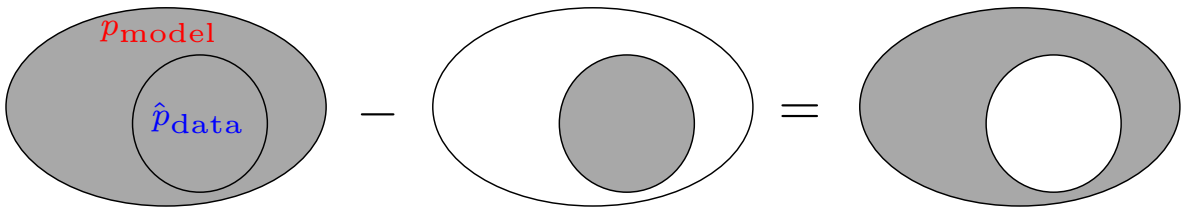
Interpretation

- MLE = minimizing the dissimilarity between
  - empirical distribution $\hat{p}_{\text{data}}$ (defined by training set), and
  - model distribution p_{model}

- dissimilarity measure:
  - KL divergence (from p_{model} to $\hat{p}_{\text{data}}$)

$$
\begin{aligned}
D_{\text{KL}}(\hat{p}_{\text{data}} \,||\, p_{\text{model}}) &= \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}[\log \hat{p}_{\text{data}}(\boldsymbol{x}) - \log p_{\text{model}}(\boldsymbol{x})] \\
&= \underbrace{\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}[-\log p_{\text{model}}(\boldsymbol{x})]}_{H(\hat{p}_{\text{data}}, p_{\text{model}})} - \underbrace{\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}[-\log \hat{p}_{\text{data}}(\boldsymbol{x})]}_{H(\hat{p}_{\text{data}})}
\end{aligned}
$$

cross entropy — entropy



- entropy $H(\hat{p}_{\text{data}})$
  - a function only of the data generating process, not the model
- this means when we train the model to minimize the KL divergence
  - we need only minimize

$$-\mathbb{E}_{\mathbf{x}\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\boldsymbol{x})]$$

  - which is the same as the maximization in eq (7)
- minimizing this KL divergence

  $=$ minimizing cross-entropy between the distributions

- any loss consisting of a negative log-likelihood (NLL)
  - is a crossentrophy between
    - empirical distribution (defined by training set) and
    - probability distribution defined by model

  e.g. mean squared error: cross entropy between
    - empirical distribution and a Gaussian model

> we can thus see maximum likelihood as an attempt to
>
> - make model distribution match empirical distribution $\hat{p}_{\text{data}}$

- ideally, we would like to match
  - true data-generating distribution p_{data}
  - but have no direct access to this distribution

Bottom line

maximum likelihood

$=$ minimization of negative log-likelihood (NLL)

$=$ minimization of cross entropy

$=$ minimization of KL divergence

- when maximizing likelihood or minimizing KL divergence
  - optimal $\boldsymbol{\theta}$: the same
  - values of objective functions: different

Outline

Modeling assumptions

- to model $p(\boldsymbol{x})$ given data, we should make some assumptions
  - "You can't do inference without making assumptions." (MacKay)

1. **smoothness** assumption
   - points that are close each other are more likely to share a label

2. **cluster** assumption
   - the data form discrete clusters; points in the same cluster are likely to share a label

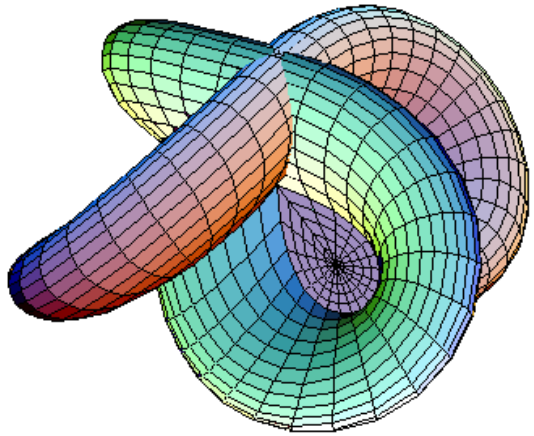
3. **manifold** assumption
   - the data lie on a manifold of much lower dimension than input space

manifold
=> high dimension에서 모든 공간에 uniform하게 분포하지 않고 보통 특정 주변에 존재하는 특징

high dimension input
=> 중요한 점을 갖는 lower dimension input으로 바꿈

A manifold

- refers to a connected region
- mathematically
  - a set of points, associated with a neighborhood around each point

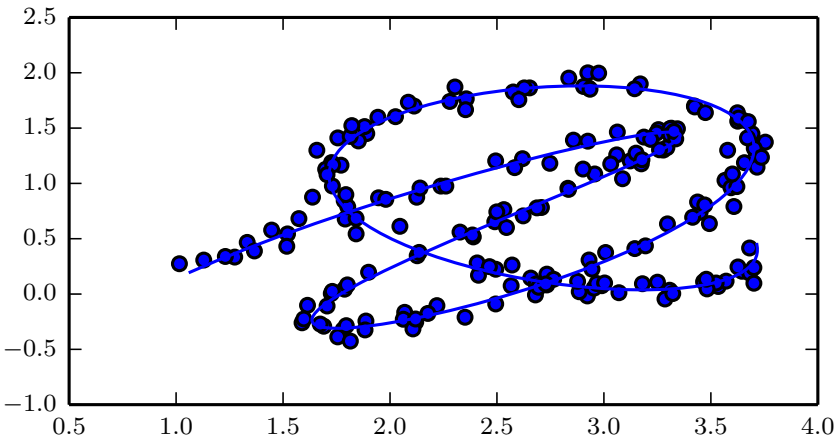


- in everyday life
  - we experience the surface of the world as a 2D plane
  - but it is in fact a manifold in 3D space

Manifold in ML context

- the term "manifold" tends to be used more loosely
- a manifold in ML: designates a connected set of points
  - that can be approximated well by only a small number of degrees of freedom/dimensions embedded in a higher-dimensional space
  - each dimension = a local direction of variation

e.g. training data lie near a 1D manifold embedded in 2D space



solid line: underlying manifold

↑

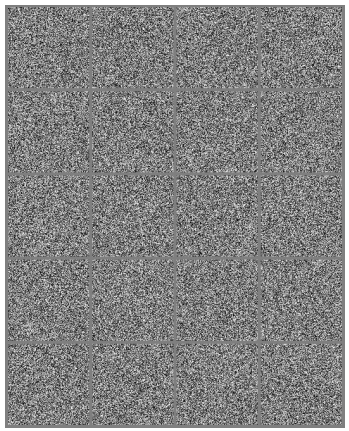
the learner should infer

Evidence #1 for manifold assumption

images, textx, sounds 등의 real media 는 uniform이 아니라 편중되어있음

probability distribution over real images/texts/sounds: highly

- random image/text/sound (uniform noise)
  - essentially never resemble structured inputs from these domains



random image
(uniformly sampled pixels)

```
alskdfjljl;sdafjld;l
l409ul,msf 3 4-0i24-
  0;lwef lk w e-92
   3pkjwf v`23-
qeg;rmsfPIOUJ V 23P042
   ;LMFA., 32- 2
  3elk23p;om ;34
```

random text
(uniformly sampled letters)

- real image/text/sound => 극히 편중되어있음
  - occupy a **negligible proportion** of volume of image/text/sound space

Evidence #2 for manifold assumption

given a real-world example, we can imagine highly similar examples

e.g. given an image, we can gradually

- dim/brighten the lights
- move/rotate objects in the image
- alter the colors on the surfaces of objects

$\Rightarrow$ corresponds to tracing out a manifold in image space

- this helps to establish:
  - the examples we encounter: connected to each other by other examples
  - similar examples: reached by transformations to traverse the manifold

Manifold learning

=>highly concentrated한 maniflod를 찾아야함

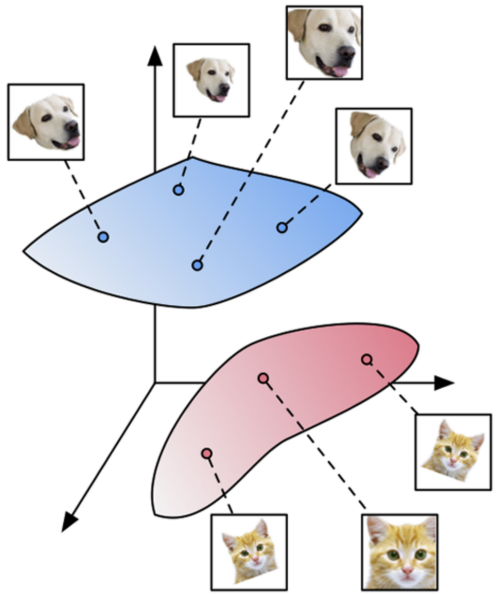
- ML problems seem hopeless
  - if want to learn functions with interesting variations across all of $\mathbb{R}^n$
- **manifold learning** algorithms surmount this obstacle by assuming

  (1) most of $\mathbb{R}^n$ consists of invalid inputs, and

  (2) interesting inputs occur only along a collection of manifolds

  => manifold 내부의 이동이나 manifold간의 이동에 관심이 있음

  => manifold는 전체 데이터 중 극히 일부에 존재



(source: Chung)

- these manifolds contain a small subset of points
  - interesting variations in the output of the learned function
    - occur only along directions that lie on manifold, or
    - happen only when we move from one manifold to another
- key idea: probability concentration

Extracting manifold coordinates

when the data lies on a low-dimensional manifold:

- it can be most natural for ML algorithms to represent the data
  - in terms of coordinates on the manifold (not in $\mathbb{R}^n$)

e.g. roads: viewed as 1D manifolds embedded in 3D space

$\Rightarrow$ we give directions in terms of 1D addresses (not 3D coordinates)

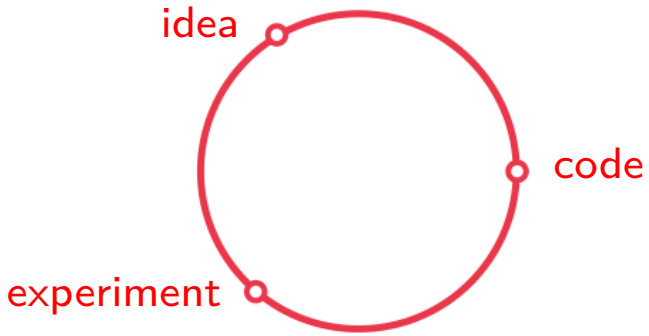
- extracting these manifold coordinates:
  - challenging but promising to improve many ML algorithms

2D manifold corresponding to two angles of rotation

Outline

Motivation

- deep learning 코딩 => 실험 => 재코딩...
  - highly iterative process



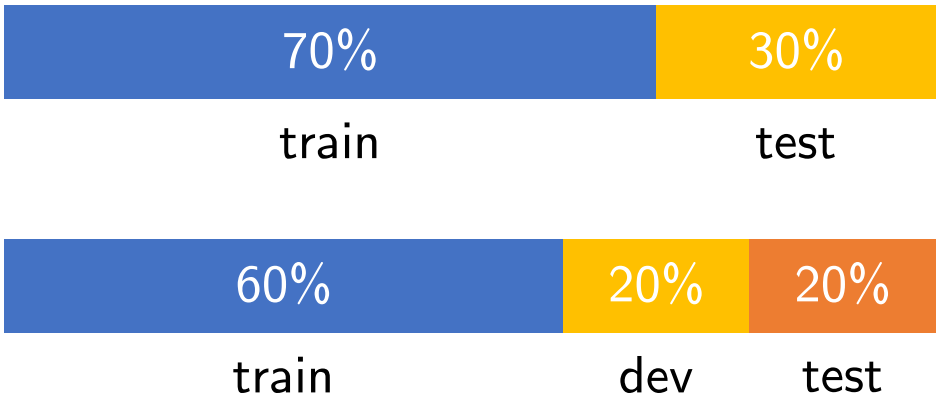
- many knobs to tweak
  - data, metric, optimizer, regularizer, hyperparameters/architecture, ...

- how to accelerate this iterative process?
  - before autoML comes on earth

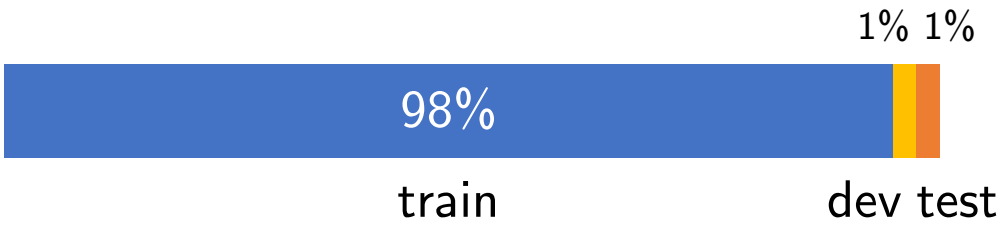
Data breakdown

generalization performace가 궁극적인 목표
train: train을 위한 data
dev: validation을 위한 data
test: generalization performance를 approximate
big data의 경우 dev와 test의 비율을 줄여도 충분히 많음

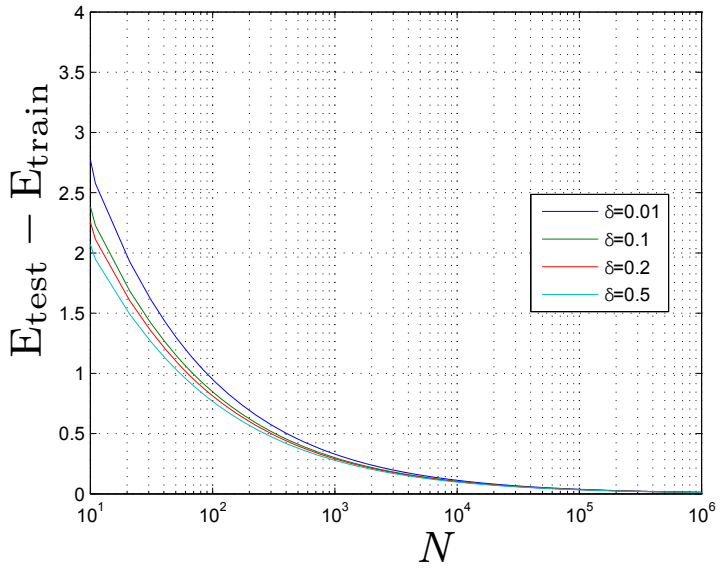
- small data ($n \approx$ 100-10,000):

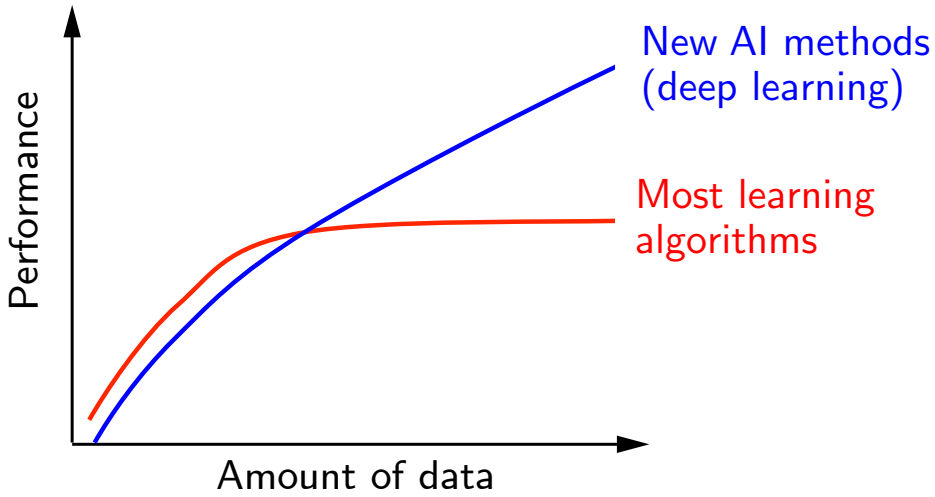


- big data ($n \approx$ 1,000,000):



Power of big data





- as $N \to \infty$
  - $\mathrm{E_{out}} - \mathrm{E_{in}} \to 0$ regardless of model/statistical confidence[4]
  - performance generally improves

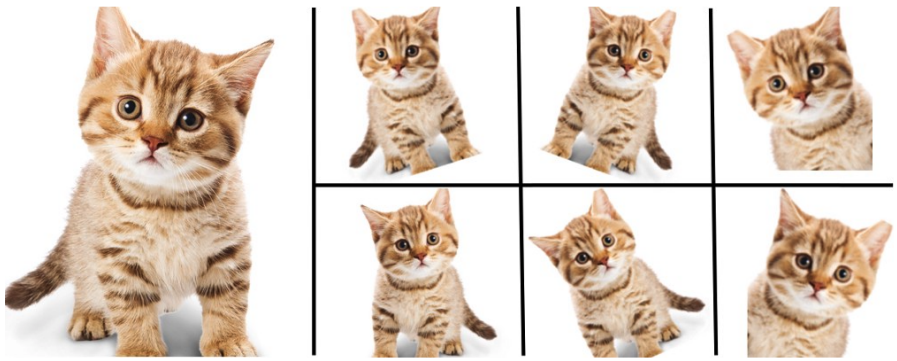
[4] δ in left plot

How much data?

- highly dependent on specific / each problems
- a rough rule of thumb (Goodfellow et al., 2016):
  - 5000 labeled examples per category
    - ▷ to achieve acceptable performance by supervised deep learning
  - at least 10 million labeled examples
    - ▷ to match/exceed human performance
- active research areas
  - pre-training and/or transfer learning
  - un/semi-supervised learning to use unlabeled data

When you do not have enough data

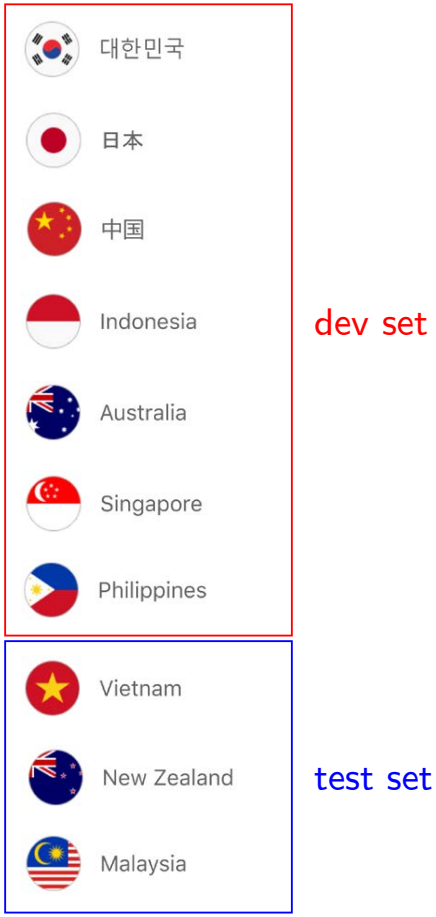
1. data augmentation 작은 변형을 통해 새로운 데이터를 만들어내는 효과
   - rotation, noise, translation
2. simulation
   simulation
   => 답이 있는 문제의 경우 실제로 simulate해봄
   - AlphaGo Zero
3. generation => 답이 없지만 비슷한 상황을 만들어서 train
   - generative adversarial net (GAN)



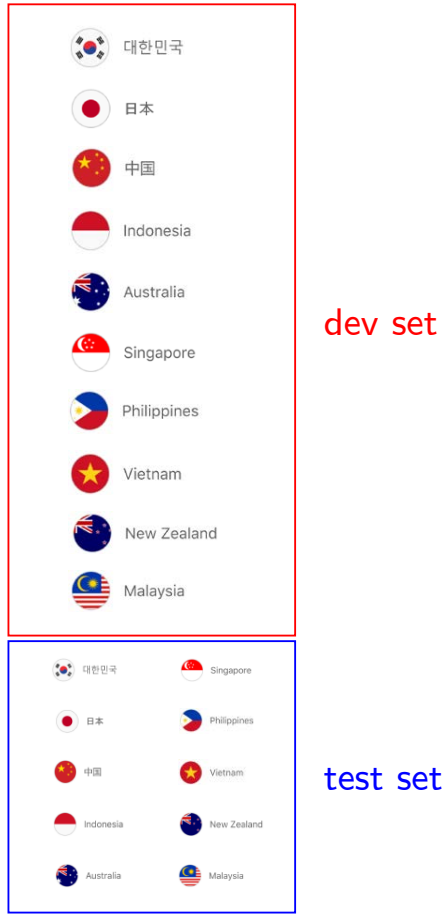
Match data distributions

dev set과 test set의 data distribution이 같을수록 좋음

xpnt dev distr ≠ test distr



- dev distr ≈ test distr (better)



Orthogonalization

어떤 기능을 위해서는 combination이 아니라 independent한 (= orthogonal knobs) 것이 어야 함(simple)

(source: Porsche)

- to open window, press 0.3 of [bttn 1] + 0.2 of [bttn 2] + 0.5 of [bttn 3]
- just press bttn [open]
- orthogonal _____ knobs → more effective control

validation error => test set에 대한 estimation

- orthogonalization in training ML models

desired task	**if you fail, try the following:** (orthogonal knob)	(less orthogonal knob)
fit train set well	bigger network better optimizer	early stopping
fit dev set well	regularization bigger training set	
fit test set well	bigger dev set	
perform well in real world	change dev set change cost function	

- early stopping (terminating training prematurely)
  - affects both training and validation performance $\Rightarrow$ less orthogonal
  - sometimes not recommended in deep learning training

Choosing a metric

- using a single real number evaluation metric
  - clear objective $\Rightarrow$ can speed up the iterative process

- optimizing and satisficing metrics
  - M metrics $\Rightarrow$ 1 optimizing metric + $(M-1)$ satisficing metrics
  - example

optimizing: 최적화할 요소
satisficing: 상수로 놓을 요소(minimum설정)

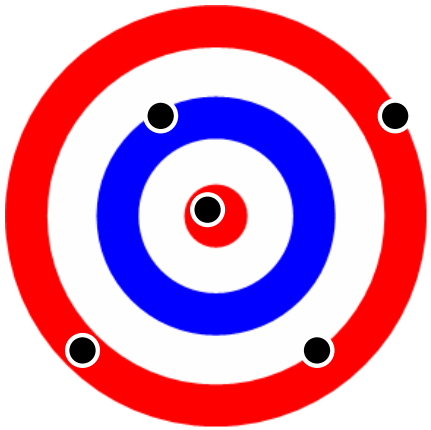
classifier	accuracy	runtime
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

$$\text{maximize} \quad \overbrace{\text{accuracy}}^{\text{optimizing metric}}$$
$$\text{s.t.} \quad \underbrace{\text{runtime}}_{\text{satisficing metric}} \leq 100\text{ms}$$

$\Rightarrow$ optimal: B

Setting (and adjusting) a target

- learning target: set by a metric + dev/test sets
  - bullets: shot by training sets



- change your metric and/or dev/test sets
  - if you experience bad generalization

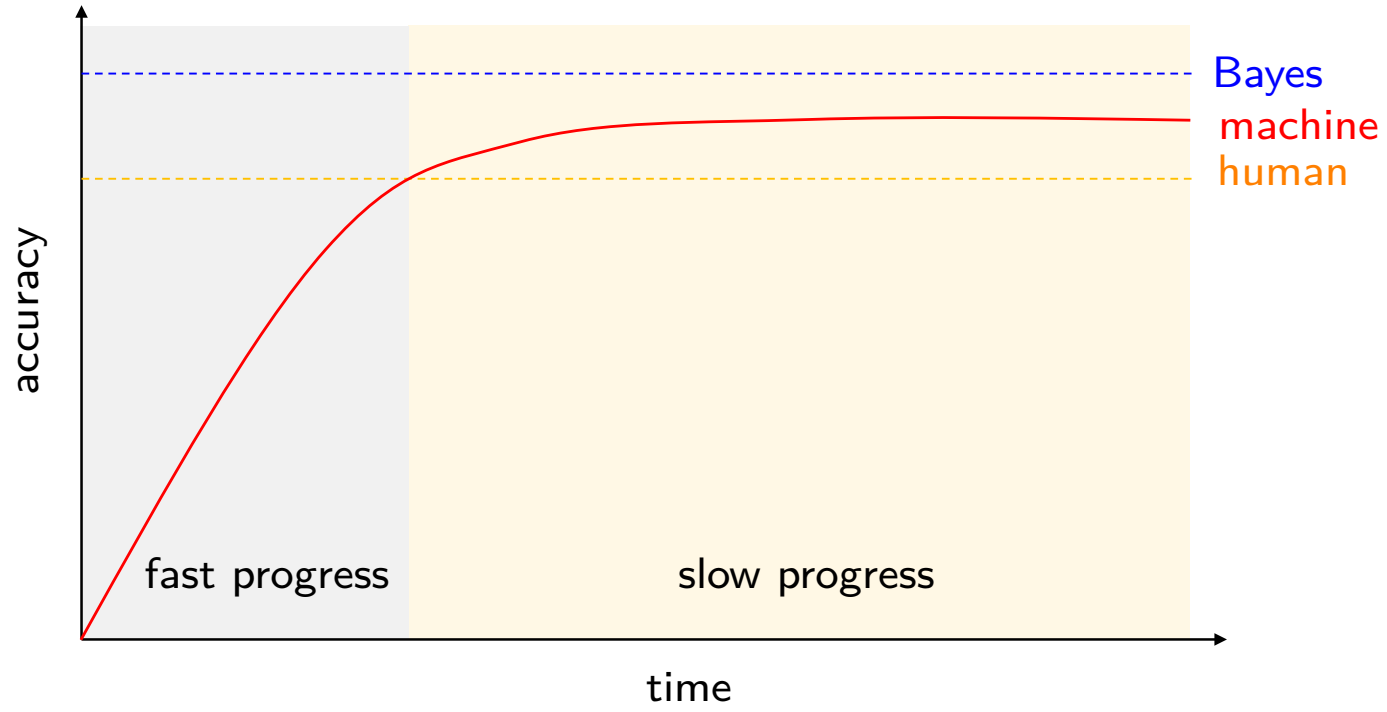
    (*i.e.* have low test error but cannot handle new inputs well)

generalization이 실패하면
metric이나
train/dev/test set을 변경해야함

Referencing human-level performance

- bayes error (irreducible error): lowest possible error
- human error
  - often close to Bayes error (especially for natural perception tasks)

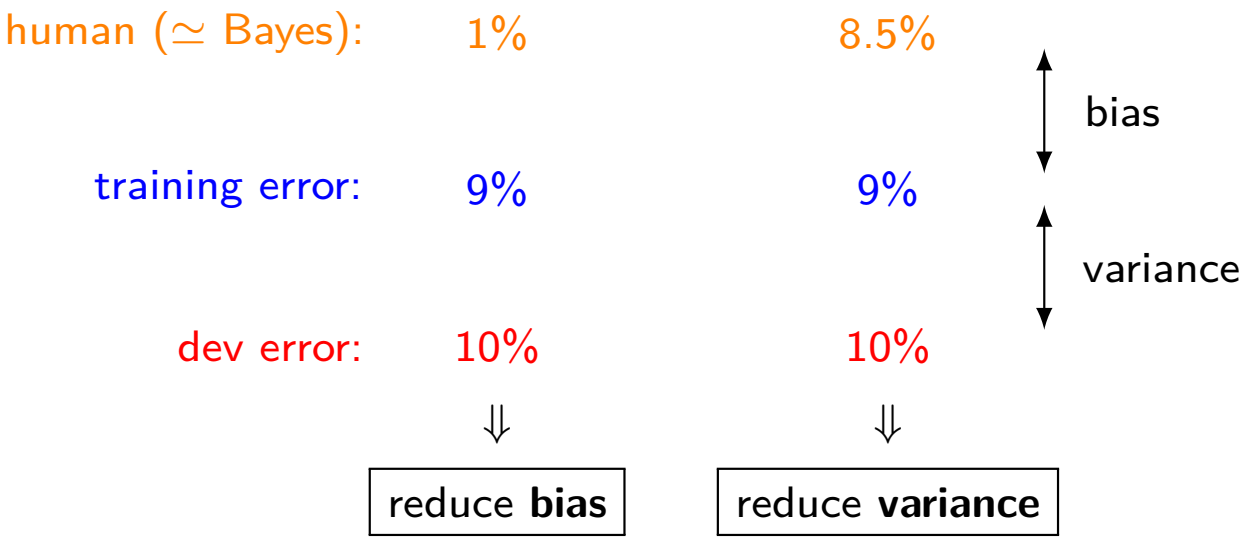
$\Rightarrow$ used as a proxy for Bayes error $\Rightarrow$ target for ML



- when ML performance $<$ human performance: tools exist
  - more labeled data from humans
  - manual error analysis (why did humans get things right?)
  - better bias-variance analysis

- when ML performance $>$ human performance:
  - the above tools no longer useful
  - more difficult to improve machine learning

Bias-variance analysis



- reducing bias

bias 줄이기 =>
complexity를 늘린다
optimizer를 좋은거 쓴다

  - more complex model, longer training, better optimization
  - better hyperparameter/architecture

- reducing variance

variance 줄이기 =>
more data
regularization

  - more data, regularization
  - better hyperparameter/architecture

Outline

Summary

- deep learning: hierarchical representation learning
  - driving forces: big data, parallel hw (GPU), advanced algorithms
- machine learning: learn from data to achieve generalization
  - objectives: making $\mathrm{E}_{\mathrm{test}} \simeq \mathrm{E}_{\mathrm{train}}$ + making $\mathrm{E}_{\mathrm{train}} \simeq 0$
  - challenge: approximation-generalization or bias-variance tradeoff
  - weapons: big data, optimization, regularization
  - example: linear models for classification/regression/prob estimation
- data sets: train/dev/test
  - breakdown in big data era: train/dev/test $\simeq$ 98%/1%/1%
  - handling data scarcity: data augmentation, simulation, generation
- machine learning strategy: needed to accelerate iterative process
  - orthogonalization, optimizing/satisficing metrics, bias-variance analysis