1. Intro

05/10/2020

Machine learning is programming computers to improve a performance criterion using example data or past experience.
Machine learning (or data mining) is the task of producing knowledge from data.

1.1 Well-posed learning problems

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

To have a well-defined problem we have to identify:

1. class of tasks
2. measure of performance
3. source of experience

-The chosen algorithm is the last thing to focus on

e.g.:

Task T: playing checkers

Performance measure P: percent of games won

Training experience E: playing practice games against itself

1.2 Designing a Learning system

1.2.1 Choosing the training experience

One key attribute is whether the training experience provides direct or indirect feedback regarding the choices made by the performance system.

- **Direct training:** individual checkers board states and the correct move for each. (easier)

- **Indirect training: c**onsists of the move sequences and final outcomes of various games played.

Problem of credit assignment: importance of a move in the entire game.

A **second important attribute of the training experience** is the degree to which the learner controls the sequence of training examples. For example, the learner might rely on the teacher to select informative board states and to provide the correct move for each. Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.

A **third important attribute of the training experience** is how well it represents the distribution of examples over which the final system performance P must be measured. In general, learning is most reliable when the training examples follow a distribution similar to that of future test examples.

e.g: If its training experience E consists only of games played against itself, there is an obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.

To complete the design of the learning system, we must now choose:

1. the exact type of knowledge to be learned
2. a representation for this target knowledge
3. a learning mechanism

1.2.2 Choosing the target function

What type of knowledge will be learned and how this will be used by the performance program. In our case, the program just need to learn hiw ti chiise the best move:

- ChooseMove: B → M → output: best move
- V: B → R (Real numbers) → output: board states with higher scores:
    - value V(b) for an arbitrary board state b in B, as follows:
        1. if b is a final board state that is won, then V(b) = 100
        2. if b is a final board state that is lost, then V(b) = -100
        3. if b is a final board state that is drawn, then V(b) = 0

4. if b is a not a final state in the game, then V(b) = V(b'), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

1.2.3 Choosing a Representation for the Target Function

V will be calculated as a linear combination of the following board features:

x1: the number of black pieces on the board
x2: the number of red pieces on the board
x3: the number of black kings on the board
x4: the number of red kings on the board

x5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)
X6: the number of red pieces threatened by black

Our elaborated learning task is now:

Task T: playing checkers
Performance measure P: percent of games won in the world tournament
Training experience E: games played against itself
Targetfunction: V:Board + 8
Targetfunction representation

1.2.4 Choosing a function approximation algorithm

We require a set of **training examples, each describing a specific board state *b* and the training value Vtrain(b) for b.**

In other words: a training example is an ordered pair of the form (b,Vtrain(b))

e.g.: the following describes a board state b in which black has won the game

(note x2 = 0 indicates that red has no remaining pieces) and for which the target function value Vtrain(b) is therefore +100.

1.2.4.1 Estimating training values

We require training examples that assign specific scores to specific board states.

Q: How to assign training values to the more numerous intermediate board states that occur before the game's end?

Assign the training value of Vtrain(b) for b to be V'(Successor(b)), where V' is the learner's current approximation to V and where Successor(b), this is:

Rule for estimating training values: Vtrain(b) ←V'(Successor (b))

we can see this will make sense if V' tends to be more accurate for board states closer to game's end.

1.2.4.2 Adjusting the weights

Last step: specify the learning algorithm for choosing the weights *Wi* to best fit the set of training examples {<b,Vtrain(b)>}.

Define the set of weights or **best hypothesis,** which minimizes the squared error between the training values and the values prediced by the hypothesis V'

We require an algorithm that will **incrementally refine the weights as new training examples become available** and that will be robust to errors in these estimated training values.

This algorithm can be viewed as performing a stochastic gradient-descent search through the space of possible hypotheses (weight values) to minimize the squared error E.

LMS (least mean square) weight update rule.

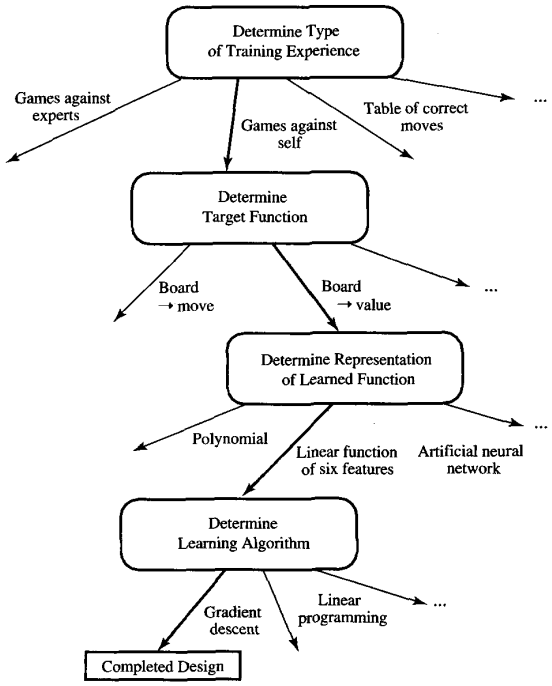
- Use the current weights to calculate V'(b)
- For each weight *wi*, update it as: $wi \leftarrow wi + n$ (Vtrain(b) - V'(b)) xi

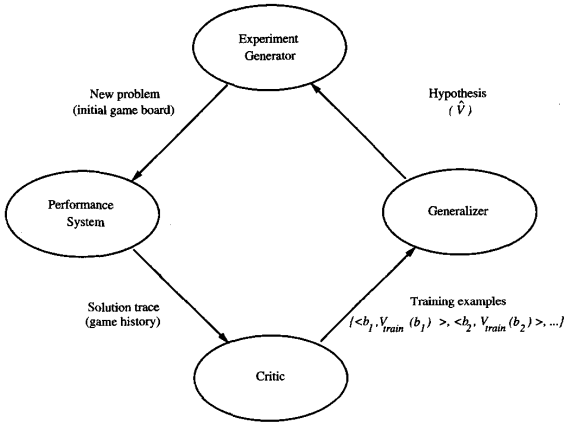
n~0.1 | *When Vtrain(b) - V'(b) = 0 no weights are changed.*

1.2.5 The final design

4 program modules:

1. Prformance system: must solve the given performance task
2. Critic: as input history of the game an produce a set of training examples
3. Generalizer: produce an hypothesis that estimate the target function.
4. Experiment Generator: produce a new problem, to maximize the learning rate of the overall system.





1.3 Perspectives and issues in machine learning

One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits. For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights w0 through w6. **The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples.**

1.4 Machine Learning Problems

Learning a function f : X → Y , given a training set D containing sampled information about f.
Learning a function f means computing an approximated function ˆf that returns values as close as possible to f , specially for samples x not present in the training set D.
Xd = {x|x in D} in X and |Xd|<<|X|

1.5 Classification

Classification based on input

1.5.1 Supervised Learning

D = {<xi,yi>}

$$X \equiv \begin{cases} A_1 \times \ldots \times A_n,\ A_i \text{ finite sets } \textbf{(Discrete)} \\ \Re^n \textbf{ (Continuous)} \end{cases}$$

$$Y \equiv \begin{cases} \Re^k \textbf{ (Regression)} \\ \{C_1, \ldots, C_k\},\ \textbf{(Classification)} \end{cases}$$

Special case:
$X \equiv A_1 \times \ldots A_n$ (A_i finite) and $Y \equiv \{0, 1\}$ **(Concept Learning)**

1.5.2 Unsupervised Learning

D = {xi}

- Learning what normally happens
- No output available
- Clustering: grouping similar instances

1.5.3 Reinforcment Learning

D = {<Ai(1) , . . . , Ai(n), Ri>}

- Learning a policy: what to do in every state
- No supervised output available, only sparse and time-delayed rewards

1.6 Learning task

Given a training set D = {<Xi , c(Xi)>}, find the best approximation h*in H of the target function c : X → {0, 1}.
Steps:
1. Define the hypothesis space H (i.e., a representation of the hypotheses)
2. Define a performance metric to determine the best approximation
3. Define an appropriate algorithm

A hypothesis h is consistent with a set of training examples D of target concept c if and only if h(x) = c(x) for each training example hx, c(x)i in D.

$$\text{Consistent}(h,D) = (\text{For every } x \text{ in } D)\ h(x) = c(x)$$

The real goal of a machine learning system is to find the best hypothesis h that predicts correct values of h(x') for instances x' not in D with respect to the unknown values c(x').

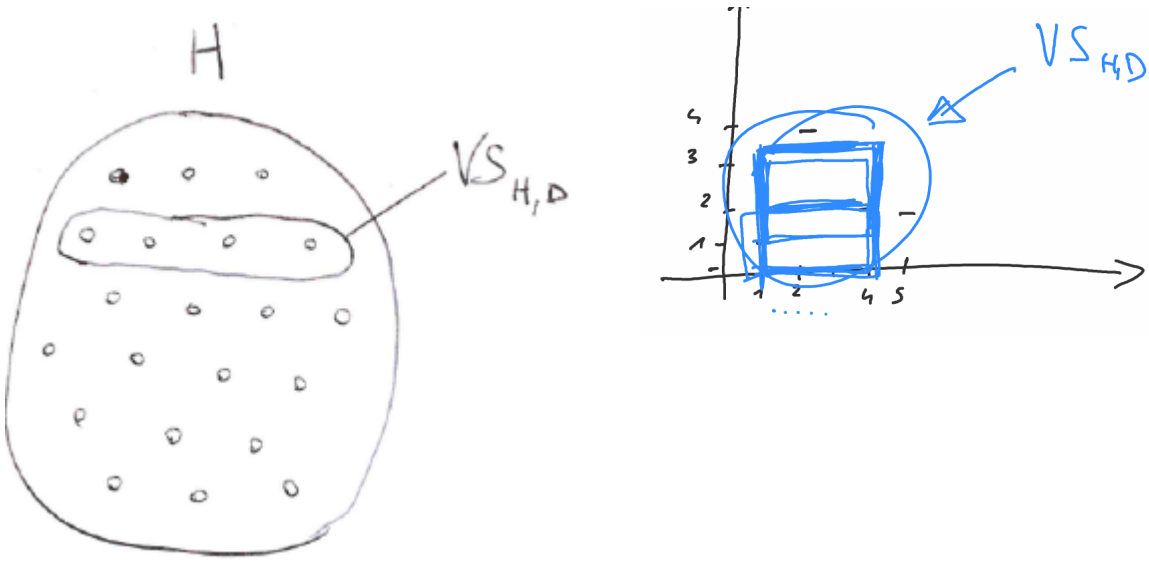
A performance measure is based on evaluating c(xi) = h(xi) for all xi in D.

Inductive learning hypothesis: Any hypothesis that approximates the target function well over a sufficiently large set of training and unobserved examples.

1.6.1 Representation of hypothesis space

The version space, VSH,D, is the subset of hypotheses from H consistent with all training examples in D.

$$VS_{H,D} \quad \{h \text{ in } H \mid \text{Consistent}(h,D)\}$$



1.6.2 LIST-THEN-ELIMINATE ALGORITHM

1. VS ← a list containing every hypothesis in H
2. For each training example, <x, c(x)>: remove from VS any hypothesis h for which h(x) ≠ c(x)

3. Output the list of hypotheses in VS

1.6.3 Output of a learning system

1. VSH',D (H' can represent all the subsets of X) → cannot predict instances not in D
2. VSH,D (H can not represent all the subsets of X) → limited prediction instances not in D
3. h* in VSH,D (h is one hypothesis chosen within the VS) → maximum prediction power on values not in D

In case of **noisy data set**, statistical methods must be used to implement robust algorithms and to avoid the possibility of having no consistent hypothesis.

2. Classification Evaluation

References
T. Mitchell. Machine Learning. Chapter 5

2.1 Statistical Evaluation

Performance evaluation in classification based on accuracy or error rate.

Consider a typical classification problem:

f: X→Y

X instance space

D: prob distribution over X

S: dataset: a sample from X

Consider a hypothesis h, **solution of a learning algorithm** obtained from S and estimate the accuracy:

Errors

True error: is the prob that h will misclassify an instance drawn at random according to D. (true error cannot be computed)

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

Sample error: is the proportion of examples h misclassifies (computed only on a small data sample)

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x) \neq h(x))$$

where delta = 1 if f(x)≠h(x), 0 otherwise. **Accuracy(h)** = 1 - error(h)

the goal of a learning system is to be accurate in h(x), for every x not in S
If accuracyS (h) is very high, but accuracyD(h) is poor, then our system would not be very useful.

Probabilities

bias = E[errorS(h)]-errorD(h)

1. If S is the training set used to compute h, errorS (h) is optimistically biased
2. For unbiased estimate, h and S must be chosen independently E[errorS (h)] = errorD(h)
3. Even with unbiased S, errorS (h) may still vary from errorD(h). The smaller the set S, the greater the expected variance.

Confidence intervals

With approximately N% probability, $error_{\mathcal{D}}(h)$ lies in interval

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

where

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Estimators

How to compute errorS (h)

1. Partition the data set D (D = T Union S, T Inters. S = Vuoto, |T| = 2/3|D|)
2. Compute a hypothesis h using training set T
3. Evaluate errorS (h)

errorS (h) is an **unbiased estimator** for errorD(h); since true error is not computable, but we need it to evaluate h, we must use an estimate

Trade off between training and testing

- More training and less testing improve performance: but Es does not approximate well Ed
- More training and less testing reduces variance

USE: 2/3 TRAINING, 1/3 TESTING

Comparisons

True comparison: d = Ed(h1) - Ed(h2)

Its estimator: d^= Es1(h1) - Es2(h2)

d^ is unbiased iff all parameters are independent: E[d^] = d

Overfitting: h in H **overfits** training data if there is h' in H:

Es(h) < Es(h') Ed(h) > Ed(h')

Rem.: h is solution of learning algorithm L when using training set T → h = L(T)

K-fold Cross Validation

1. Partition data set D into k disjoint sets S1, S2, . . . , Sk (|Si | > 30)
2. For i = 1, . . . , k do

   use Si as test set, and the remaining data as training set Ti
   Ti = {D − Si}
   hi = L(Ti)

   deltai = Esi(hi)
3. Return

   error(L,D) = 1/k Sum deltai

Comparing Algorithm

E(S in D)[errorD(LA(S)) − errorD(LB(S))]

where L(S) is the hypothesis output by learner L using training set S
i.e., the expected difference in true error between hypotheses output by learners LA and LB; can be approximated by a K-Fold Cross Validation.

Using A and B →

$$\bar{\delta} \equiv \frac{1}{k}\sum_{i=1}^{k} \delta_i$$

Note: if $\bar{\delta} < 0$ we can estimate that L_A is better than L_B.

2.2 Performance metrics

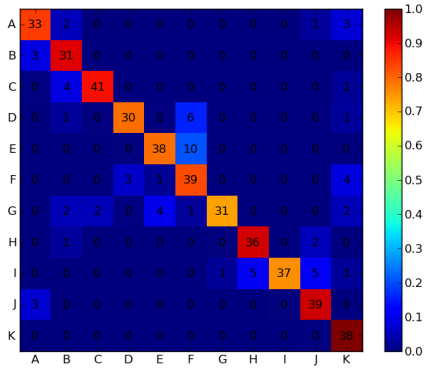
Accuracy is not always a good performance metric, especially with unbalanced data.

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

Error rate = | errors | / | instances | = (FN + FP) / (TP + TN + FP +FN)
Accuracy = 1 - Error rate = (TP + TN) / (TP + TN + FP + FN)
Problems when datasets are unbalanced.

Confusion Matrix

In a classification problem with many classes, we can compute how many times an instance of class Ci is classified in class Cj.



Main diagonal contains accuracy; outside the diagonal the errors.

3. Decision Trees

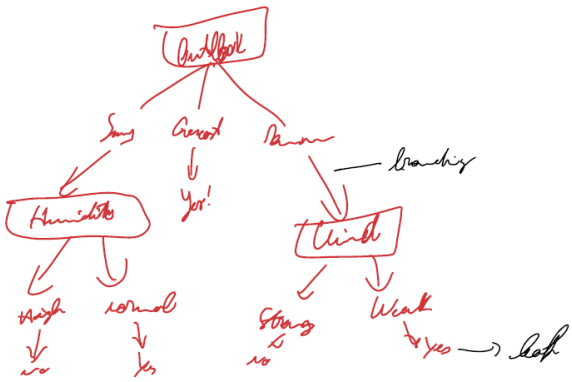
References
T. Mitchell. Machine Learning. Chapter 3

To compute the "best" consistent hypothesis with respect to (wrt) D:

1. Define htpothesis Space H
2. Implement an algorithm that searches for the best hypothesis

Given a discrete input space with m attributes (A1*...*Am) and a classification problem f: X → C **decision tree** has 3 characteristics:

1. Internal node → attribute Ai
2. Branch → value of ai,j in Ai
3. Leaf → assign a classification value c in C



Decision trees represent a **disjunction of conjunctions of constraints** on the attribute values of instances.

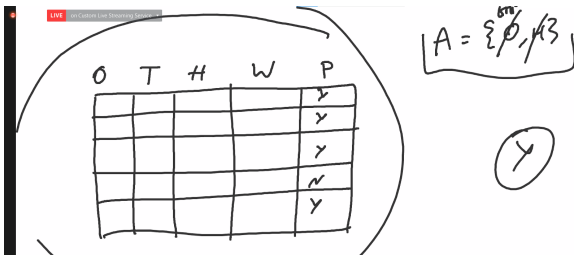
(Outlook = Sunny ∧ Humidity = Normal) ∨ (Outlook = Overcast) ∨ (Outlook = Rain ∧ Wind = Weak)

A **rule** is generated for **each path to a leaf node.**
IF (Outlook = Sunny) ∧ (Humidity = High)
THEN PlayTennis = No

ID3 Algorithm

1 Create a Root node for the tree
2 if all Examples are **positive** (you always play tennis), then return the node Root with **label +**

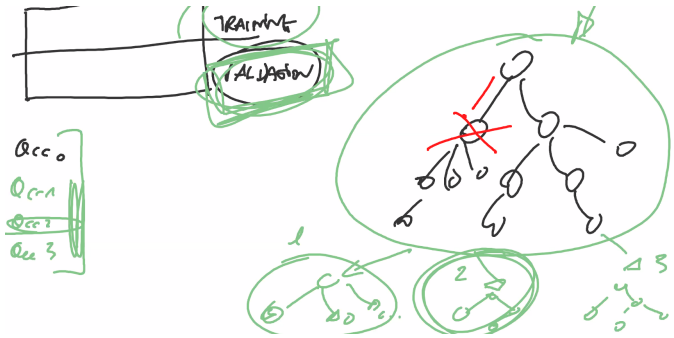
3 if all Examples are **negative**, then return the node Root with **label –**



4 if **Attributes** is **empty**, then **return the node Root with label = most common value** of Target attribute in Examples
5 Otherwise

- For each value vi of A
  - if Examples vi is empty then add a leaf node with label = most common value of Target attribute in Examples
  - else
    add the tree ID3(Examplesvi , Target attribute, Attributes–{A})

If there isn't an attribute, it means that we don't consider that attribute important for the choose.



Output tree depends on attribute order

Information gain measures **how well a given attribute separates** the training examples according to their target classification.

ID3 selects the attribute that induces **highest information gain.**

Entropy

Information gain measured as reduction in **entropy** (how much a dataset is impure).

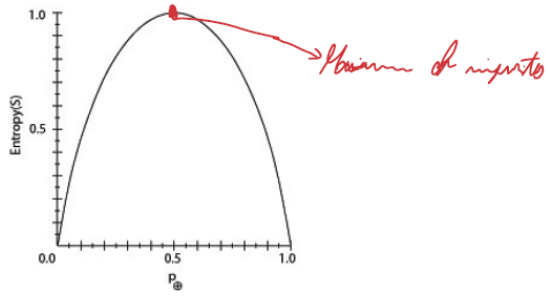
- $p_{\oplus}$ is the proportion of positive examples in S (+/N)
- $p_{\ominus}$ (= 1 – $p_{\oplus}$) is the proportion of negative examples in S
- Entropy measures the impurity of S

Example

Consider the set S = [9+, 5−]

Entropy(S) = −(9/14)log2(9/14) − (5/14)log2(5/14) = 0.940

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$



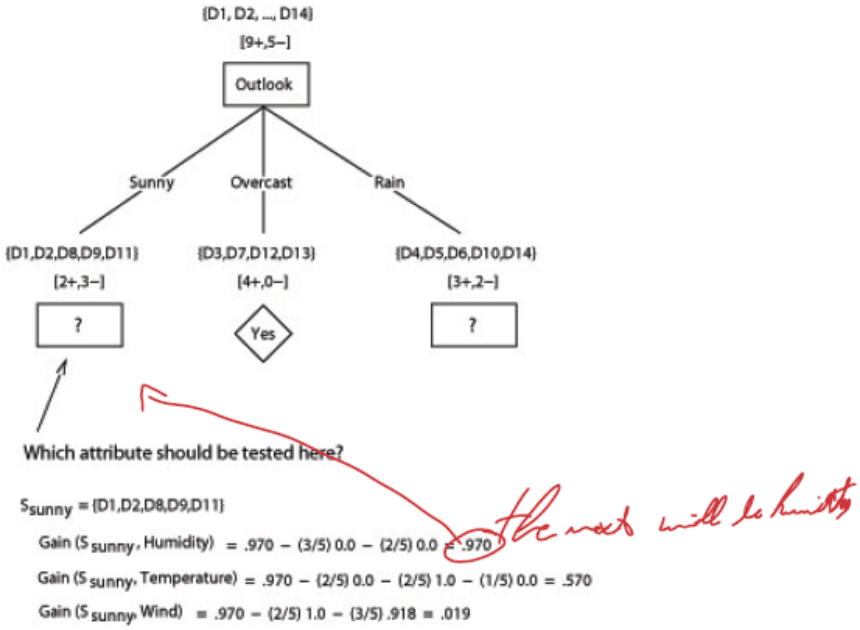
In case of multi-valued target functions (c-wise classification)

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

Gain(S, A) = expected reduction in entropy of S caused by knowing the value of attribute A.

Gain(S, A) ≡ Entropy(S) − Sum[(|Sv|/|S|) Entropy(Sv)]

Sv = {s in S|A(s) = v}



If you end before the leaf you will choose with less accuracy

Overfitting in Decision Trees

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

Prune

Reduced-Error pruning

Split data into training and validation set

- Do until further pruning (potatura) is harmful (decreases accuracy):
  1 Evaluate impact on validation set of pruning each possible node
  2 Greedily remove the one that most improves validation set accuracy

Rule Post-Pruning

- Convert the learned tree into a set of rules
- Generalize each rule independently
- Sort rules for use

Specific Attributes

- Attributes with Many Values

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- Attributes with Costs
  - Tan and Schlimmer (1990)

    $$\frac{Gain^2(S, A)}{Cost(A)}$$

  - Nunez (1988) ($w \in [0, 1]$ determines importance of cost)

    $$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

- Unknown Attribute Values

  Assign most common value

4. Probability and Bayes

4.1 Probability

Uncertainty: not secure about the outcomes

Omega sample space (set of possibilities)

omega in Omega sample point

Probability space: Function P : Omega → R such that:

- 0≤P(omega) ≤ 1
- Sum P(omega) = 1

Event: any subset of Omega

Probability of an event A is a function assigning A to [0,1]

P(A) = Sum P(omega)

A **random variable** (outcome of a random phenomenon) is a function from the sample space to some range X : Omega → R or B etc.

P induces a **probability distribution** for a random variable X:

P(X = xi) = Sum P(w)

A **proposition** is the event (subset of) where an assignment to a random variable holds.

event a = A = true = { omega in Omega such that A(omega) = true}

4.2 Syntax and Semantics

Prior or unconditional probabilities → normal probability (P(odd = true) = 0.5) without knowing anything

A **probability distribution** is a function assigning a probability value to all possible assignments of a random variable. (for Real is continuous)

e.g.: P(Weather) =< 0.72, 0.1, 0.08, 0.1 >

Joint probability distribution for a set of random variables gives the probability of every atomic joint event on those random variables.

Joint probability distribution of the random variables *Weather* and *Cavity*: $P(Weather, Cavity)$ = a 4×2 matrix of values:

Weather =	*sunny*	*rain*	*cloudy*	*snow*
Cavity = *true*	0.144	0.02	0.016	0.02
Cavity = *false*	0.576	0.08	0.064	0.08

Conditional/Posterior Probability: I know the outcome of a random variable, how does this affect probability of other random variables?

P(a|b) = P(a^b)/P(b) if P(b) ≠ 0

Product rule

P(a^b) = P(a|b)P(b) = P(b|a)P(a)

Total probabilities

P(a) = P(a|b)P(b) + P(a|¬b)P(¬b)

In general,

P(X) = Sum P(X|Y = yi)P(Y = yi)

Chain rule

P(X1,X2) = P(X1)P(X2|X1)

P(X1, . . . ,Xn) = Prod. P(Xi |X1, . . . ,Xi−1)

4.3 Inference by enumeration

	toothache		¬ *toothache*	
	catch	¬ *catch*	*catch*	¬ *catch*
cavity	**.108**	**.012**	**.072**	**.008**
¬ *cavity*	**.016**	**.064**	**.144**	**.576**

For any proposition ϕ, sum the atomic events where it is true: $P(\phi) = \sum_{\omega:\omega\models\phi} P(\omega)$

e.g.: P(¬cavity|toothache) = P(¬cavity ^ toothache)/P(toothache)

4.4 Independence

A and B are independent iff

P(A|B) = P(A) or P(B|A) = P(B) or P(A, B) = P(A)P(B)

P(Toothache, Catch, Cavity,Weather) = P(Toothache, Catch, Cavity)P(Weather)

- P(Toothache, Cavity, Catch) has 23 – 1 = 7 independent entries
- If I have a cavity, the probability that the probe catches in it does not depend on whether I have a toothache:
  (1) P(catch|toothache, cavity) = P(catch|cavity)
- The same independence holds if I haven't got a cavity:
  (2) P(catch|toothache,¬cavity) = P(catch|¬cavity)
- Catch is conditionally independent of Toothache given Cavity:
  P(Catch|Toothache, Cavity) = P(Catch|Cavity)
- Equivalent statements:
  P(Toothache|Catch, Cavity) = P(Toothache|Cavity)
  P(Toothache, Catch|Cavity) = P(Toothache|Cavity)P(Catch|Cavity)

General formulation:
X conditionally independent from Y given Z iff:

- P(X|Y , Z) = P(X|Z)

  P(X, Y |Z) = P(X|Y , Z)P(Y |Z) = P(X|Z)P(Y |Z)

  In general,
  P(Y1, . . . ,Yn|Z) = P(Y1|Y2, . . .Yn, Z)P(Y2|Y3 . . . Yn, Z) · · · P(Yn|Z)

  Yi conditionally independent from Yj given Z

  P(Y1, . . . ,Yn|Z) = P(Y1|Z)P(Y2|Z) · · · P(Yn|Z)

Chain rule + Conditional independence
P(Toothache, Catch, Cavity) = P(Toothache|Catch, Cavity)P(Catch, Cavity)
= P(Toothache|Catch, Cavity)P(Catch|Cavity)P(Cavity)
= P(Toothache|Cavity)P(Catch|Cavity)P(Cavity) = 2 + 2 + 1 = 5 independent numbers (instead of 2^3 – 1)

4.5 Bayes' Rule

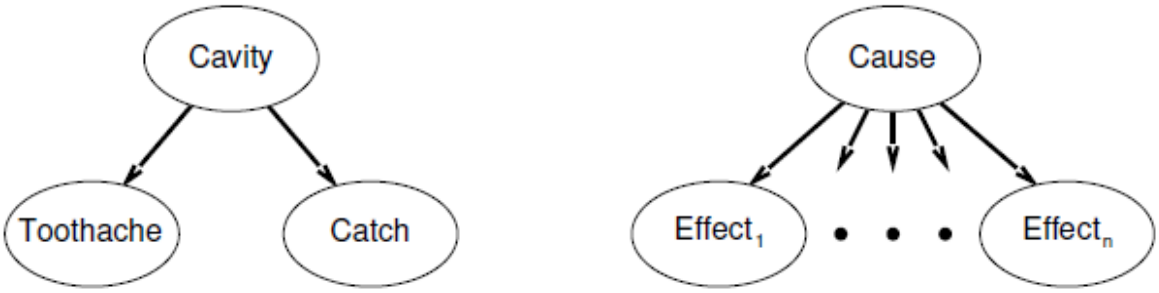
Product rule: P(a ^ b) = P(a|b)P(b) = P(b|a)P(a)
⇒ **Bayes' rule** P(a|b) =P(b|a)P(a)/P(b)
Or P(Y |X) = P(X|Y)P(Y)/P(X) = alfa*P(X|Y)P(Y)
P(Cause|Effect) = P(Effect|Cause)P(Cause) / P(Effect)

With conditional independence...

General/chained situation

Yi , . . .Yn conditionally independent each other given Z
P(Z|Y1, . . . ,Yn) =P(Y1|Z) · · · P(Yn|Z) P(Z)
P(Cause|Effect1, . . . , Effectn) = alfa* P(Cause) PROD P(Effecti |Cause)

4.5.1 Bayesian networks

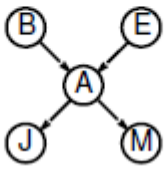


- a directed, acyclic graph (link "directly influences")
- a conditional distribution for each node given its parents: P(Xi |Parents(Xi))

In the simplest case, conditional distribution represented as a **conditional probability table (CPT)** giving the distribution over Xi for each combination of parent values.

All **joint probabilities** computed with the chain rule:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{Parents}(X_i))$$



5. Bayesian Learning

References
T. Mitchell. Machine Learning. Chapter 6

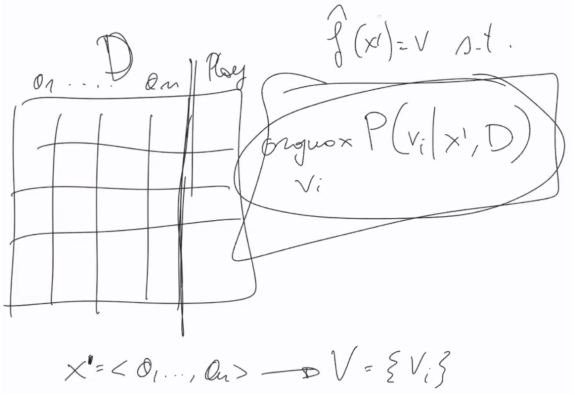
5.1 Bayes Methods

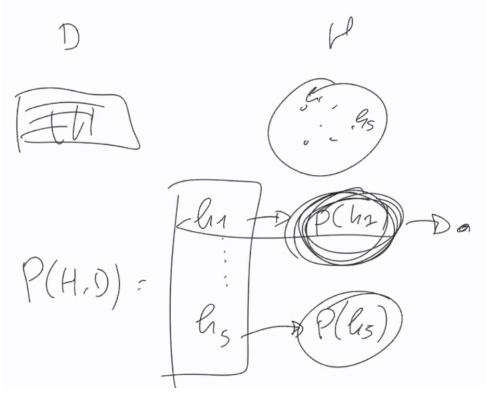
Provide **practical learning algorithms** and **conceptual frameworks**

Probabilistic estimation

Classification as Probabilistic estimation

Given f to learn : X → V, D (based on which f will learn), a new instance x', best prediction f(x') = v* ⇒ v* = argmax P(v|x',D) (v that maximizes prob)





Generally we want the most probable hypothesis h **given D**, hence, the ***Maximum a posteriori*** hypothesis h map: (we look for h that maximizes probability):

$$h_{MAP} \equiv \arg\max_{h \in H} P(h|D) = \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$
$$= \arg\max_{h \in H} P(D|h)P(h)$$

Maximum Likelihood hypothesis of generating Data we are observing

$$h_{ML} = \arg\max_{h \in H} P(D|h)$$

N.B.: h MAP(x') may not be the most probable classification !!! (we take the class returned)

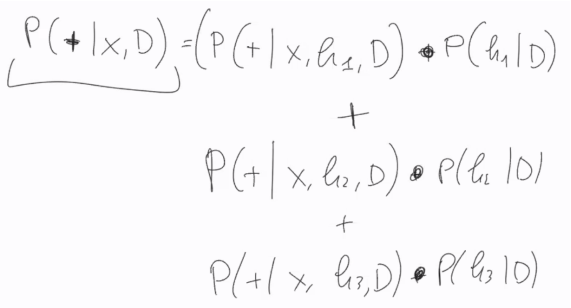
Bayes Optimal Classifier

Consider target function f : X → V, V = {v1, ..., vk}, data set D and a new instance x !in D:

$$P(v_j|x,D) = \text{SUM } P(v_j|x, h_i)P(h_i|D)$$

(given new example and D, the new example is classified as vj)

We are independent from the dataset, because, hi are given, so they are independent from other hypothesis based on dataset.



Computes most prob class, Vob, for new instance x:

$$v_{OB} = \arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|x, h_i)P(h_i|D)$$

Example:

$$P(h_1|D) = 0.4, \quad P(\ominus|x, h_1) = 0, \quad P(\oplus|x, h_1) = 1$$
$$P(h_2|D) = 0.3, \quad P(\ominus|x, h_2) = 1, \quad P(\oplus|x, h_2) = 0$$
$$P(h_3|D) = 0.3, \quad P(\ominus|x, h_3) = 1, \quad P(\oplus|x, h_3) = 0$$

therefore

$$\sum_{h_i \in H} P(\oplus|x, h_i)P(h_i|D) = 0.4$$
$$\sum_{h_i \in H} P(\ominus|x, h_i)P(h_i|D) = 0.6$$

and

$$v_{OB} = \arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j|x, h_i)P(h_i|D) = \ominus$$

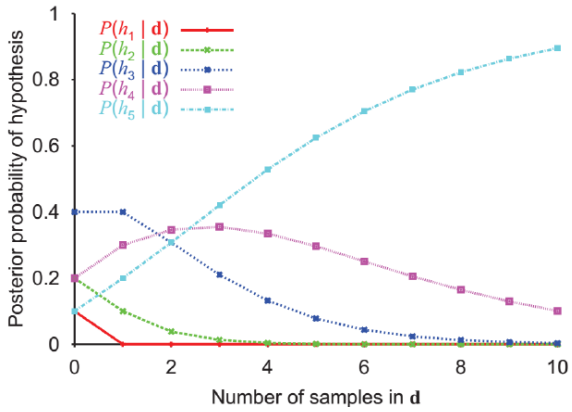
Optimal learner: no other classification method (same hyp space etc.) can outperform this one; it maximizes prob. new instance is classified correctly. Label new instances.

alfa in Bayes rule:

1. First candy is lime: $D_1 = \{l\}$

$P(h_i|\{d_1\}) = \alpha P(\{d_1\}|h_i)P(h_i)$ (Bayes rule)

$$\begin{aligned} \mathrm{P}(H|D_1) &= \alpha < 0, 0.25, 0.5, 0.75, 1 > \cdot < 0.1, 0.2, 0.4, 0.2, 0.1 > \\ &= \alpha < 0, 0.05, 0.2, 0.15, 0.1 > \\ &= < 0, 0.1, 0.4, 0.3, 0.2 > \end{aligned}$$



New example: consider theta = number of cherries in [0,1]

Data set: D = {c cherries, l lime}, N = c + l

P(c|h theta) = theta

P(l |h theta) = 1 – theta

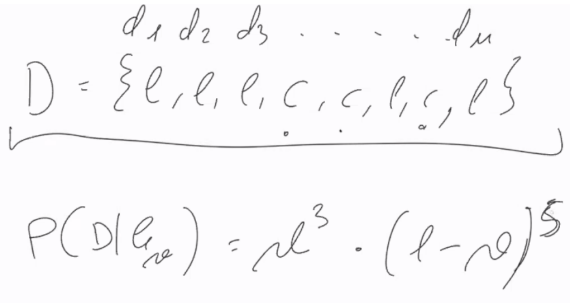
$$h_{ML} = \underset{h_\theta}{\operatorname{argmax}} P(D|h_\theta) = \underset{h_\theta}{\operatorname{argmax}} L(D|h_\theta)$$

with $L(D|h_\theta) = \log P(D|h_\theta)$

$$P(D|h_\theta) = \prod_{j=1\ldots N} P(d_j|h_\theta) = \theta^c \cdot (1-\theta)^l$$

$$L(D|h_\theta) = c \log\theta + l \log(1-\theta)$$

$$\frac{dL(D|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \Rightarrow \theta_{ML} = \frac{c}{c+l} = \frac{c}{N}$$



Theta ml: is the proportion that best explains the data you are observing; it's the ratio of cherries over total in your bag and theta ML is the number that maximizes the probability of seeing the data you are observing given the hypothesis Htheta.

Theta ml is the most probable class you can obtain.

In general:

Theta is a vector of parameters.

$$\Theta_{ML} = \underset{\Theta}{\operatorname{argmax}} \log P(d_i|\Theta)$$

Bernoulli

Probability distribution of a binary random variable $X \in \{0, 1\}$

$$P(X = 1) = \theta \quad P(X = 0) = 1 - \theta$$

(e.g., observing head after flipping a coin, extracting a lime candy, ...).

$$P(X = k; \theta) = \theta^k (1 - \theta)^{1-k}$$

Multi-variate Bernoulli

Joint probability distribution of independent variables

$$P(X_1 = k_1, \ldots ; \theta_1, \ldots, \theta_n) = \prod_{i=1}^{n} P(X_i = k_i; \theta_i) = \prod_{i=1}^{n} \theta_i^{k_i} (1 - \theta_i)^{1-k_i}$$

Binomial

Probability distribution of k outcomes from n Bernoulli trials

$$P(X = k; n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}$$

Multinomial

Generalization of binomial

$$P(X_1 = k_1, \ldots, X_d = k_d; n, \theta_1, \ldots, \theta_d) = \frac{n!}{k_1! \ldots k_n!} \theta_1^{k_1} \cdot \ldots \cdot \theta_d^{k_d}$$

(e.g., rolling a d-sided dice n times and observing k times a particular value, extracting k lime candies after n extractions form a bag containing d different flavors, ...).

Summary

Probabilistic method: not eff cause you have to calculate all the distributions
Maximum likelihood: it's convenient because you can simply iterate calculus

5.2 Naive Bayes Classifier

Naive Bayes Classifier uses conditional independence to approximate the solution; works under the assumptions that are independent.

$P(X,Y \mid Z) = P(X \mid Y, Z)P(Y \mid Z) = P(X \mid Z)P(Y \mid Z)$

$$
\begin{aligned}
v_{MAP} &= \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2 \ldots a_n, D) \\
&= \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2 \ldots a_n | v_j, D) P(v_j | D)}{P(a_1, a_2 \ldots a_n | D)} \\
&= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2 \ldots a_n | v_j, D) P(v_j | D)
\end{aligned}
$$

Bayes rule

Eliminate denominator because is positive

Class of new instance x:

$$
v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j | D) \prod_i P(a_i | v_j, D)
$$

5.2.1 Naive Bayes Algorithm

Target function $f : X \mapsto V$, $X = A_1 \times \ldots \times A_n$, $V = \{v_1, ..., v_k\}$, data set D, new instance $x = \langle a_1, a_2 \ldots a_n \rangle$.

$$
\hat{P}(v_j | D) = \frac{|\{< \ldots, v_j >\}|}{|D|}
$$

$$
\hat{P}(a_i | v_j, D) = \frac{|\{< \ldots, a_i, \ldots, v_j >\}|}{|\{< \ldots, v_j >\}|}
$$

Naive_Bayes_Learn(A, V, D)

- for each target value $v_j \in V$
  - $\hat{P}(v_j | D) \leftarrow$ estimate $P(v_j | D)$
  - for each attribute A_k
    - for each attribute value $a_i \in A_k$
      - $\hat{P}(a_i | v_j, D) \leftarrow$ estimate $P(a_i | v_j, D)$

Proportion of times you see that class above the total.

Classify_New_Instance(x)

$$
v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \hat{P}(v_j | D) \prod_{a_i \in x} \hat{P}(a_i | v_j, D)
$$

Typical solution is Bayesian estimate with prior estimates (p-prior, m-weight)

$$
\hat{P}(a_i | v_j, D) = \frac{|\{< \ldots, a_i, \ldots, v_j >\}| + mp}{|\{< \ldots, v_j >\}| + m}
$$

$$P(PlayTennis = yes) = P(y) = 9/14 = 0.64$$
$$P(PlayTennis = no) = P(n) = 5/14 = 0.36$$
$$P(Wind = strong|y) = 3/9 = 0.33$$
$$P(Wind = strong|n) = 3/5 = 0.60$$
$$\dots$$
$$P(y)\ P(sun|y)\ P(cool|y)\ P(high|y)\ P(strong|y) = .005$$
$$P(n)\ P(sun|n)\ P(cool|n)\ P(high|n)\ P(strong|n) = .021$$

$$\rightarrow v_{NB} = n$$

The strong wind influence more to say no

5.3 Learn to classify text

A set of documents ad input and a learn target function f: Docs → {c1,..,ck}

We compute a vocabulary V = {wk} (size n) with all the words appeared

Representations:

1. Boolean features: 1 if appear 0 otherwise (Multivariate Bernoulli)
2. Ordinal features: number of occurrences (Multinomial)

5.3.1 With Naive Bayes approach

Compute a Data set D = {<di, ci>} di documents

$$c_{NB} = \underset{c_j \in C}{\operatorname{argmax}} P(c_j|D) \prod_i P(d_i|c_j, D) \qquad P(d_i|c_j, D) = \prod_{i=1}^{length(d_i)} P(a_i = w_k|c_j, D)$$

where P(ai = wk |cj) is probability that word in position i is wk , given cj

Multi-variate Bernoulli Naive Bayes distribution

n-dimensional vector 1 if word wk appears in document d, 0 otherwise

$$P(d|c_j, D) = \prod_{i=1}^{''} P(w_i|c_j, D)^{I(w_i \in d)} \cdot (1 - P(w_i|c_j, D))^{1-I(w_i \in d)}$$

$$\hat{P}(w_i|c_j, D) = \frac{t_{i,j} + 1}{t_j + 2}$$

$t_{i,j}$: number of documents in D of class c_j containing word w_i
t_j: number of documents in D of class c_j
1, 2: parameters for Laplace smoothing

Multinomial Naive Bayes distribution

n-dimensional vector with number of occurrences of word wi in document d

P(d|cj ,D) = Mu(d; n,) = . . .

$$\hat{P}(w_i|c_j, D) = \frac{\sum_{d \in D} tf_{i,j} + \alpha}{\sum_{d \in D} tf_j + \alpha \cdot |V|}$$

$tf_{i,j}$: term frequency (number of occurrences) of word w_i in document d of class c_j
tf_j: all term frequencies of document d of class c_j
α: smoothing parameter ($\alpha = 1$ for Laplace smoothing)

Algorithm (using Bernoulli)

Estimate $\hat{P}(c_j)$ and $\hat{P}(w_i|c_j)$ using *Bernoulli distribution*.

LEARN_NAIVE_BAYES_TEXT_BE(D, C)

$V \leftarrow$ all distinct words in D
for each target value $c_j \in C$ do
  $docs_j \leftarrow$ subset of D for which the target value is c_j
  $t_j \leftarrow |docs_j|$: total number of documents in c_j
  $\hat{P}(c_j) \leftarrow \frac{t_j}{|D|}$
  for each word w_i in V do
    $t_{i,j} \leftarrow$ number of documents in c_j containing word w_i
    $\hat{P}(w_i|c_j) \leftarrow \frac{t_{i,j}+1}{t_j+2}$

6. Probabilistic models

References

C. Bishop. Pattern Recognition and Machine Learning. Sect. 4.2, 4.3

Approaches that allows to estimate probability that given an instance we have a certain classes.

Two families of models:
Generative: estimate P(x|Ci) and then compute P(Ci |x) with Bayes
Discriminative: estimate P(Ci |x) directly

6.1 Probabilistic generative models

$$P(C_1|\mathbf{x}) = \frac{P(\mathbf{x}|C_1)P(C_1)}{P(\mathbf{x})} = \frac{P(\mathbf{x}|C_1)P(C_1)}{P(\mathbf{x}|C_1)P(C_1) + P(\mathbf{x}|C_2)P(C_2)}$$
$$= \frac{1}{1+\exp(-a)} = \sigma(a)$$

with:

$$a = \ln \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_2)P(C_2)}$$

and

$\sigma(a) = \frac{1}{1+\exp(-a)}$ the *sigmoid function.*

Multi-class

$$P(C_k|\mathbf{x}) = \frac{P(\mathbf{x}|C_k)P(C_k)}{\sum_j P(\mathbf{x}|C_j)P(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)}$$

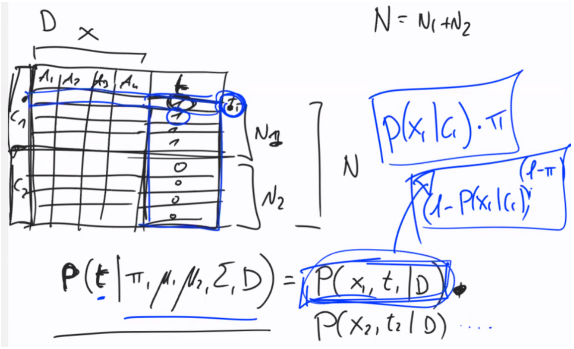
(normalized exponential or softmax function)

with $a_k = \ln P(\mathbf{x}|C_k)P(C_k)$

6.1.1 Maximum likelihood

Likelihood function

$$P(\mathbf{t}|\pi, \mu_1, \mu_2, \mathbf{\Sigma}, D) = \prod_{n=1}^{N} [\pi \mathcal{N}(\mathbf{x}_n; \mu_1, \mathbf{\Sigma})]^{t_n} [(1-\pi)\mathcal{N}(\mathbf{x}_n; \mu_2, \mathbf{\Sigma})]^{(1-t_n)}$$



For 2 classes, we obtain

$$\pi = \frac{N_1}{N} \quad \rightarrow \text{# class 1.} \qquad \rightarrow \text{Total}$$

$$\boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n \mathbf{x}_n \qquad \boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{n=1}^{N} (1-t_n)\mathbf{x}_n$$

$$\boldsymbol{\Sigma} = \frac{N_1}{N} S_1 + \frac{N_2}{N} S_2$$

with $S_i = \frac{1}{N_i}\sum_{n \in C_i} (\mathbf{x}_n - \boldsymbol{\mu}_i)(\mathbf{x}_n - \boldsymbol{\mu}_i)^T$, $i = 1, 2$

Posterior distributions with parametric models:

For two classes

$$P(C_1|\mathbf{x}) = \sigma(a)$$

For $k \geq 2$ classes

$$P(C_i|\mathbf{x}) = \frac{exp(a_k)}{\sum_j exp(a_j)}$$

$$a_k = \mathbf{w}^T \mathbf{x} + w_0$$

$$\mathbf{w}^T\mathbf{x} + w_0 = (w_0\ \mathbf{w})\begin{pmatrix}1\\ \mathbf{x}\end{pmatrix}$$

$$\tilde{\mathbf{w}} = \begin{pmatrix}w_0\\ \mathbf{w}\end{pmatrix}, \tilde{\mathbf{x}} = \begin{pmatrix}1\\ \mathbf{x}\end{pmatrix}$$

$$a_k = \mathbf{w}^T\mathbf{x} + w_0 = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$$

Likelihood for a parametric model $\mathcal{M}_\Theta$: $P(\mathbf{t}|\Theta, D)$, $D = \langle \mathbf{X}, \mathbf{t}\rangle$

Maximum likelihood solution:

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}} \ln P(\mathbf{t}|\Theta, \mathbf{X})$$

When $\mathcal{M}_\Theta$ belongs to the exponential family, likelihood $P(\mathbf{t}|\Theta, \mathbf{X})$ can be expressed in the form $P(\mathbf{t}|\tilde{\mathbf{w}}, \mathbf{X})$, with maximum likelihood

$$\tilde{\mathbf{w}}^* = \underset{\tilde{\mathbf{w}}}{\operatorname{argmax}} \ln P(\mathbf{t}|\tilde{\mathbf{w}}, \mathbf{X})$$

6.2 Probabilistic discriminative models

Estimate dirrectly

$$P(C_i|\tilde{\mathbf{x}}, D) = \frac{exp(a_k)}{\sum_j exp(a_j)} \qquad a_k = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$$

with maximum likelihood

Max the likelihood.

$$\tilde{\mathbf{w}}^* = \underset{\tilde{\mathbf{w}}}{\operatorname{argmax}} \ln P(\mathbf{t}|\tilde{\mathbf{w}}, \mathbf{X})$$

without estimating the model parameters.

Simplified notation (dataset omitted): $P(\mathbf{t}|\tilde{\mathbf{w}})$

6.2.1 Logistic regression

For 2 classes, likelihood function:

Note:tn: value in the data set corresponding toxn,yn: posterior prediction of the current model wforxn.

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}$$

with $y_n = p(C_1|\mathbf{x}_n) = \sigma(\mathbf{w}^T\mathbf{x}_n)$

$$E(\mathbf{w}) \equiv -\ln p(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N}[t_n \ln y_n + (1 - t_n)\ln(1 - y_n)]$$

Solution concept: solve the optimization problem

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

Cross-entropy error function:

To minimize error we apply Newton-Raphson:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\mathbf{x}_n$$

Gradient descent step

$$\mathbf{w} \leftarrow \mathbf{w} - \boldsymbol{H}^{-1} \nabla E(\mathbf{w})$$

$\boldsymbol{H} = \nabla\nabla E(\mathbf{w})$ is the Hessian matrix of $E(\mathbf{w})$ (second derivatives with respect to $\mathbf{w}$).

Given

$$\tilde{\mathbf{X}} = \begin{pmatrix} \tilde{\mathbf{x}}_1^T \\ \dots \\ \tilde{\mathbf{x}}_N^T \end{pmatrix} \qquad \mathbf{t} = \begin{pmatrix} t_1 \\ \dots \\ t_N \end{pmatrix},$$

$\mathbf{y}(\tilde{\mathbf{w}}) = (y_1, \dots, y_n)^T$ posterior predictions of model $\tilde{\mathbf{w}}$

$\boldsymbol{R}(\tilde{\mathbf{w}})$: diagonal matrix with $R_{nn} = y_n(1 - y_n)$

we have

$$\nabla E(\tilde{\mathbf{w}}) = \tilde{\mathbf{X}}^T (\mathbf{y}(\tilde{\mathbf{w}}) - \mathbf{t})$$

$$\boldsymbol{H}(\tilde{\mathbf{w}}) = \nabla\nabla E(\tilde{\mathbf{w}}) = \sum_{n}^{N} y_n(1 - y_n)\tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T = \tilde{\mathbf{X}}^T \boldsymbol{R}(\tilde{\mathbf{w}})\tilde{\mathbf{X}}$$

Iterative method:

1. Initialize $\mathbf{w}$
2. Repeat until termination condition

$$\mathbf{w} \leftarrow \mathbf{w} - (\mathbf{X}^T \boldsymbol{R} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - \mathbf{t})$$

Multiclass:

K classes

$$P(C_k|\tilde{\mathbf{x}}) = \frac{exp(a_k)}{\sum_j exp(a_j)} \qquad a_k = \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}} \qquad k = 1, \ldots, K$$

$$\tilde{\mathbf{X}} = \begin{pmatrix} \tilde{\mathbf{x}}_1^T \\ \ldots \\ \tilde{\mathbf{x}}_N^T \end{pmatrix} \qquad \mathbf{T} = \begin{pmatrix} t_1^T \\ \ldots \\ t_N^T \end{pmatrix} \text{ 1-of-}K \text{ encoding of labels}$$

$\mathbf{y}_n^T = (y_{n1} \ldots y_{nK})^T$ posterior prediction of $\tilde{\mathbf{x}}_n$ for model $\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K$

$$\mathbf{Y}(\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K) = \begin{pmatrix} \mathbf{y}_1^T \\ \ldots \\ \mathbf{y}_N^T \end{pmatrix} \text{ posterior predictions of model } \tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K$$

Discriminative model

$$P(\mathbf{T}|\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K) = \prod_{n=1}^{N} \prod_{k=1}^{K} P(C_k|\tilde{\mathbf{x}}_n)^{t_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} y_{nk}^{t_{nk}}$$

with $y_{nk} = \mathbf{Y}[n, k]$ and $t_{nk} = \mathbf{T}[n, k]$.

Cross-entropy error function

$$E(\tilde{\mathbf{w}}_1, \ldots \tilde{\mathbf{w}}_K) = -\ln P(\mathbf{T}|\tilde{\mathbf{w}}_1, \ldots \tilde{\mathbf{w}}_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

Iterative algorithm

gradient $\nabla_{\tilde{\mathbf{w}}_j} E(\tilde{\mathbf{w}}_1, \ldots \tilde{\mathbf{w}}_K) = \ldots$

Hessian matrix $\nabla_{\tilde{\mathbf{w}}_k} \nabla_{\tilde{\mathbf{w}}_j} E(\tilde{\mathbf{w}}_1, \ldots \tilde{\mathbf{w}}_K) = \ldots$

SUMMARY

Given a target function $f : X \to C$, and data set D

assume a parametric model for the posterior probability $P(C_k|\tilde{\mathbf{x}}, \tilde{\mathbf{w}})$
$\sigma(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}})$ (2 classes) or $\frac{exp(\tilde{\mathbf{w}}_k^T\tilde{\mathbf{x}})}{\sum_{j=1}^{K} exp(\tilde{\mathbf{w}}_j^T\tilde{\mathbf{x}})}$ (k classes)

Define an error function $E(\tilde{\mathbf{w}})$ (negative log likelihood)

Solve the optimization problem

$$\tilde{\mathbf{w}}^* = \underset{\tilde{\mathbf{w}}}{\operatorname{argmin}}\, E(\tilde{\mathbf{w}})$$

Classify new sample $\tilde{\mathbf{x}}'$ as C_{k^*} where $k^* = \operatorname{argmax}_{k=1,\ldots,K} P(C_k|\tilde{\mathbf{x}}', \tilde{\mathbf{w}}^*)$

10. Instance based learning

10.1 K-nearest neighbors

$F : X \rightarrow C$ with $D = \{(x_n,t_n)^N_{n=1}\}$,

Classification with K-NN,

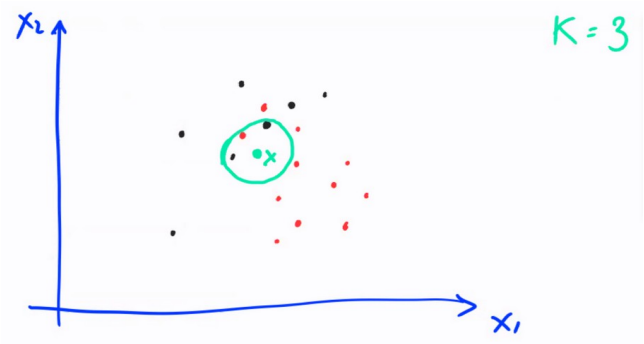
1. Find K nearest neighbors of new instance x

2. Assign to x the most common label among the majority of neighbors

Likelihood of c to new x:

$$p(c|\mathbf{x}, D, K) = \frac{1}{K} \sum_{\mathbf{x}_n \in N_K(\mathbf{x}_n, D)} \mathbb{I}(t_n = c),$$

with $Nk(x_n,D)$ nearest point and I(e) ={1 if e is true, 0 if e is false.



Requires storage of all the data set, and depends on a distance function.

Increasing K brings to smoother regions (reducing overfitting).

With K=1 a point is the closest to itself (perfect, but expensive, overfit).

With K>1 reduce overfit, but doesn't ensure better performance.

Distance function: $\|x-x_n\|^2=x^Tx+x^T_nx_n-2x^Tx_n$.

can be kernelized by using a kernel $k(x,x_n)$

Regression $X \rightarrow R$

1. Compute $N_K(x_q,D)$: K-nearest neighbors of x_q

2. Fit a regression model $y(x;w)$ on$N_K(x_q,D)$

3. Return $y(x_q;w)$

Advantages of KNN: input space doesn't converge to an optimal solution, so use KNN transforming input space to feature space (expensive).

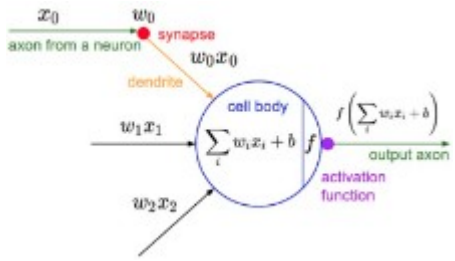
11. Artificial Neural Networks

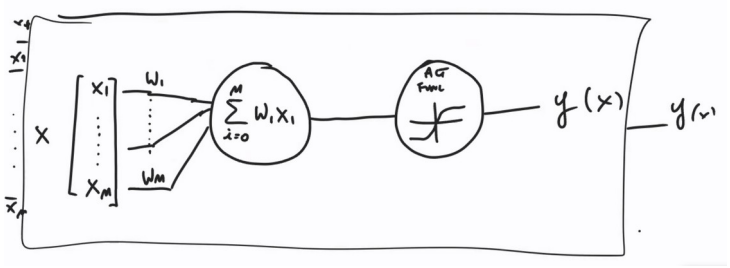
Approximate functions that take vectors as input and retrieve vectors as output

D ={$(x_n,t_n)^N_{n=1}$} such that $t_n \approx f(x_n)$

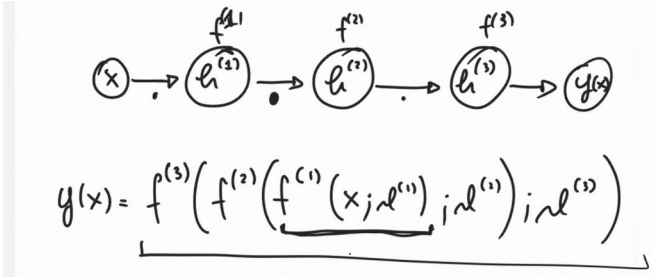
Define $y = \hat{f}(x;\theta)$ and learn parameters θ so that ˆf approximates f.

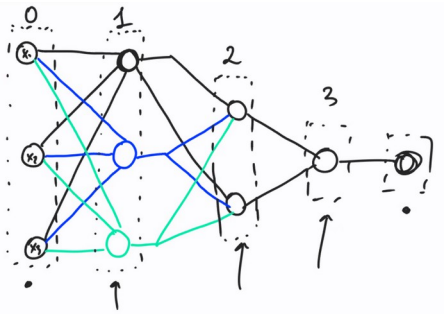
11.1 FeedForward Networks





Layers have to take as input only the output of the previous layer. Intermediate layers are called hidden layers.





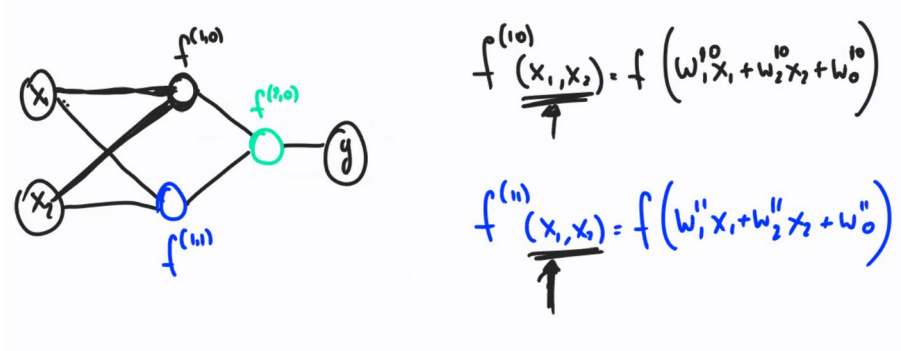
FNNs are chain structures.

The length of the chain is the depth of the network.

Final layer also called output layer.

Deep learning follows from the use of networks with a large number oflayers (large depth).

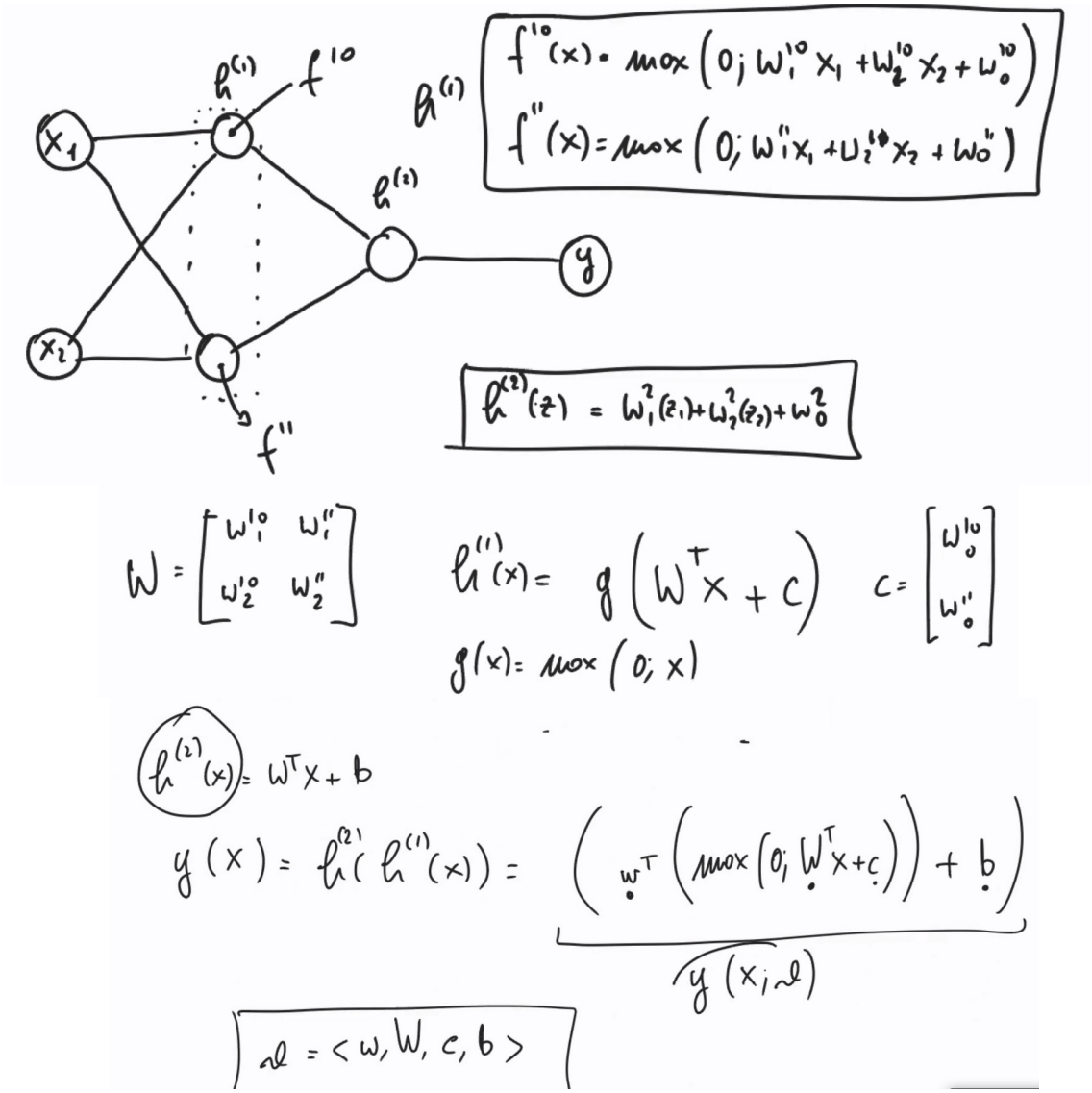
Why to use FNN → to model non linear models, the FNN learning manages complex combination of many parametric functions.



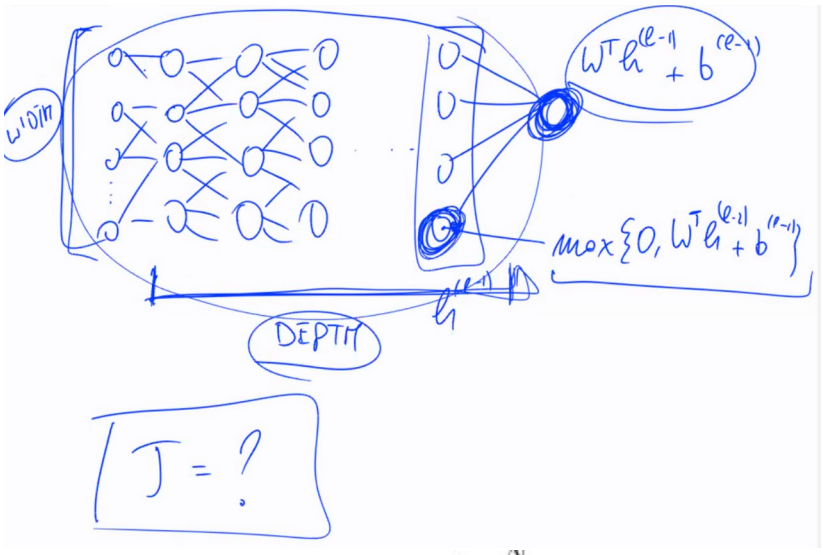
11.2 XOR Example

Not modelable with linear model → linear regression retrieve not optimal results.

Dataset:D ={((0, 0)^T, 0), ((0, 1)^T, 1), ((1, 0)^T, 1), ((1, 1)^T, 0)}



Mean squared error (MSE) loss function:



$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(t_n - y(\mathbf{x}_n))^2$$

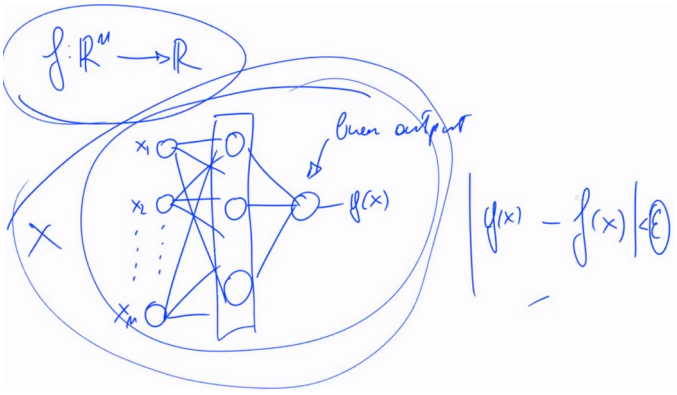
Solution:

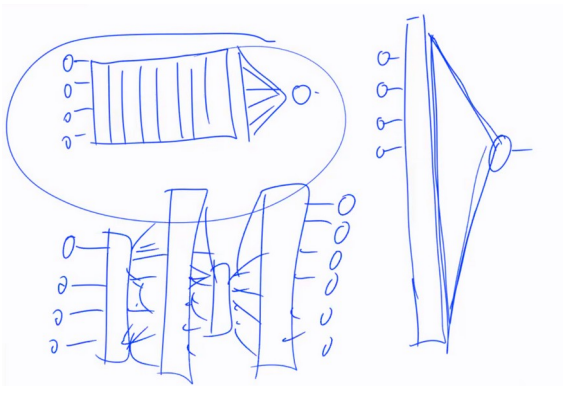
$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, b = 0$$

11.3 Architectural design

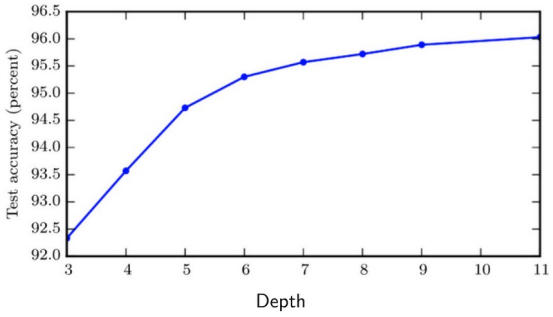
11.3.1 Depth

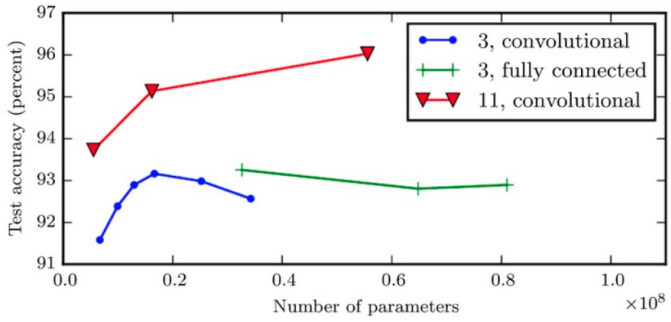
Universal approximation theorem: a FFN with a linear output layer (returns linear combination of hidden layer) and at least one hidden layer with any “squashing” activation function (e.g.,sigmoid) can approximate any Borel measurable function with any desired amount of error, provided that enough hidden units are used.It works also for other activation functions (e.g., ReLU)





Note: It's better increase numer of layer than the number of component for each layer.





11.3.2 Cost function

Model implicitly defines a conditional distributionp(t|x,θ) (Probability that an input from output instance space, in the output provides by the network correspond with the one provided by the function).

Cost function: Maximum likelihood principle (cross-entropy)

$$J(\boldsymbol{\theta}) = E_{\mathbf{x},\mathbf{t}\sim\mathcal{D}}\left[-\ln(p(\mathbf{t}|\mathbf{x},\boldsymbol{\theta}))\right]$$

Example:
Assuming additive Gaussian noise we have

$$p(\mathbf{t}|\mathbf{x},\boldsymbol{\theta}) = \mathcal{N}(\mathbf{t}|f(\mathbf{x};\boldsymbol{\theta}),\beta^{-1}I)$$

and hence

$$J(\boldsymbol{\theta}) = E_{\mathbf{x},\mathbf{t}\sim\mathcal{D}}\left[\frac{1}{2}||\mathbf{t}-f(\mathbf{x};\boldsymbol{\theta})||^2\right]$$

Maximum likelihood estimation with Gaussian noise corresponds to mean squared error minimization.

11.4 Output units activation functions

In case of:

1. Regression

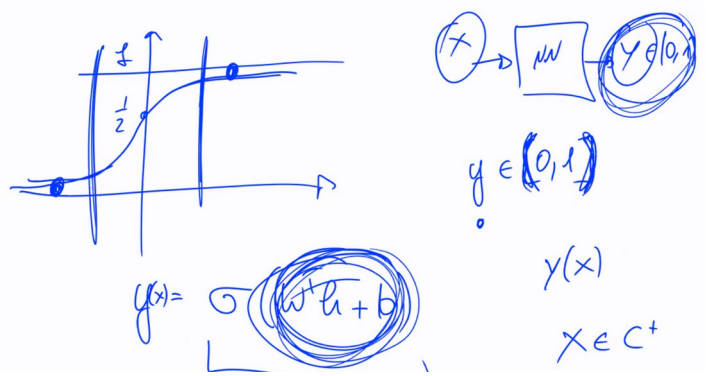
Output is the linear cobination of the inputs

Likelihood → Gaussian

$y = W^t h + b$, using Gaussian → $p(t|x) = N(t|y,B^{-1})$

Cost function: maximum likelihood (cross-entropy) that is equivalent to minimizing mean squared error.

2. Binary classification



Sigmoid units: $y=\sigma(w^Th+b)$

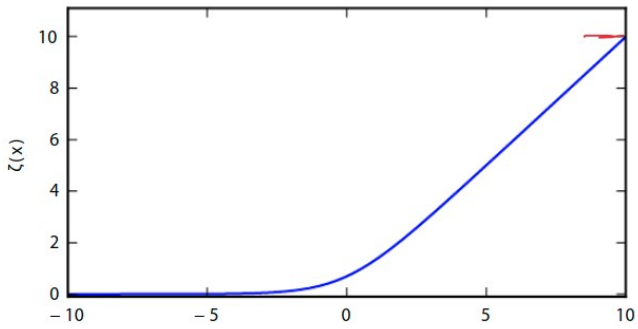
x is hidden in h

Likelihood → Bernoulli

$J(\theta) = E_{x,t\sim D}[-\ln p(t|x)]$

$-\ln p(t|x) = -\ln \sigma(\alpha)^t(1-\sigma(\alpha))^{1-t} = -\ln\sigma((2t-1)\alpha) = \text{softplus}((1-2t)\alpha)$

with $\alpha=wTh+b$.



The softplus function

3. Multi-class classification

Softmax units: $yi=\text{softmax}(\alpha i) = \exp(\alpha i)/\sum j \exp(\alpha j)$

Likelihood –> Multinomial

$J_i(\theta) = E_{x,t\sim D}[-\ln \text{softmax}(\alpha i)]$ with $\alpha_i=w^T_ih+b_i$.

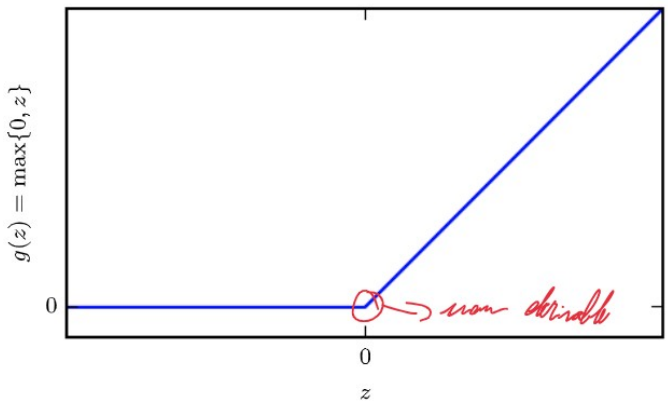
4.(bonus) Hidden unit activation functions

Since predicting which activation function will work best is usually impossible, we use:

Rectified Linear Units(ReLU):

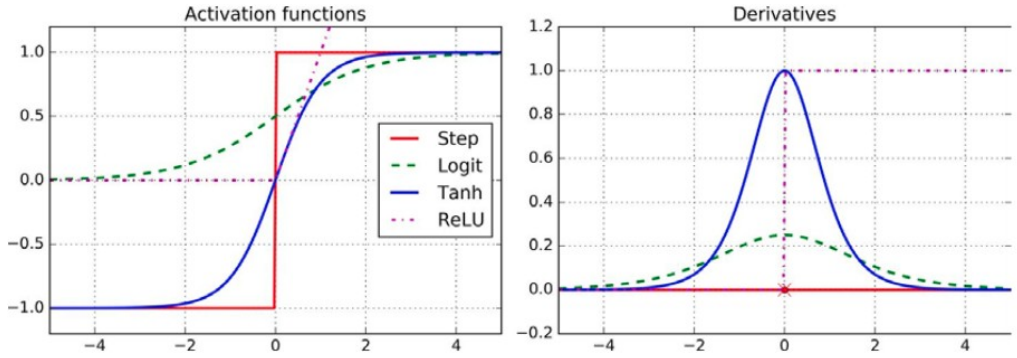
$$g(\alpha) = \max(0,\alpha).\ (\alpha \text{ is the linear comb. of the inputs}).$$

Easy to optimize, not differentiable at 0

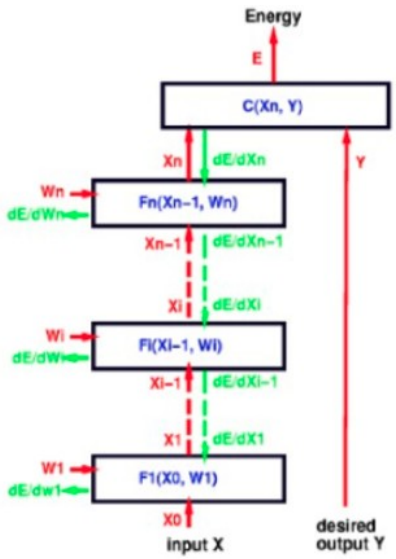


Sigmod and hyperbolic tangent:

g(α) =σ(α) and g(α) = tanh(α) Closely related astanh(α) = 2σ(2α)−1.



11.5 Gradient computation



To train the network we need tocompute the gradients with respectto the network parameters **θ**.

Back-propagation or backprop algorithm is used to propagate gradient computation from the cost through the whole network.

back-propagation is only used to compute the gradients

back-propagation is not a training algorithm

back-propagation is not specific to FNNs

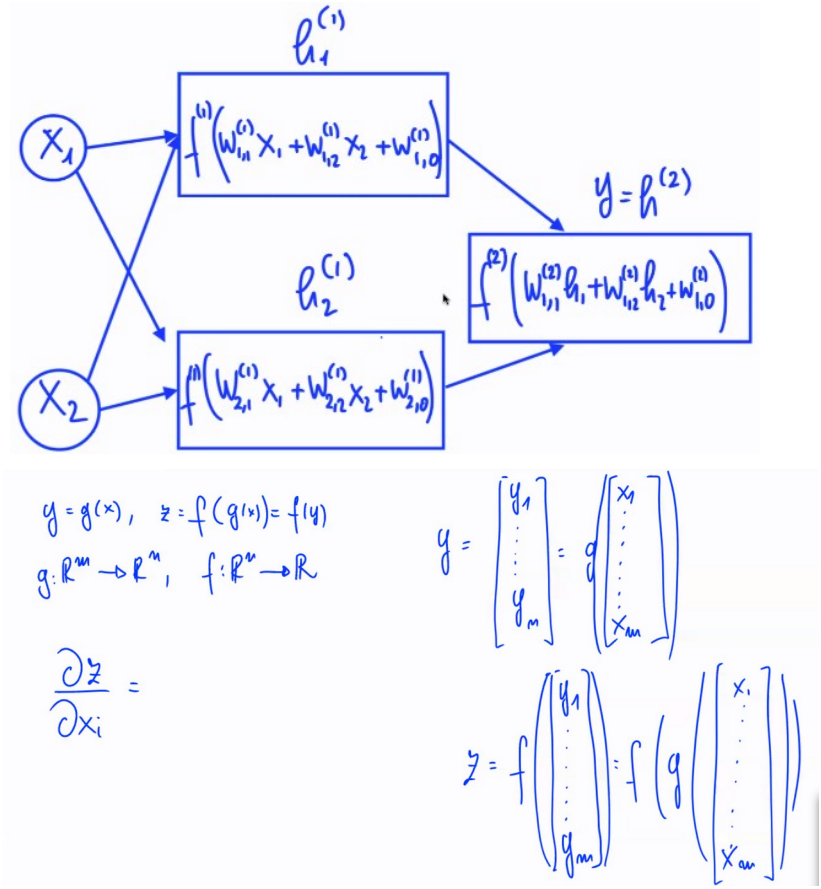
Output depends on the parameters of the network (weights).

Goal: find the weights to minimize difference from wanted and obtained

Backpropagation is just a preliminary step to compute the weights.

You are learning the network when you minimize lost function.

Example of back propagation:



Compute the gradient of the cost function w.r.t. the parameters $\nabla_\theta J(\theta)$

Chain rule

Let: $y = g(x)$ and $z = f(g(x)) = f(y)$
Applying the chain rule we have:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Gradient is the genralization of partial derivative from unidimensional to multidimensional.

For vector functions, $g : \mathbb{R}^m \mapsto \mathbb{R}^n$ and $f : \mathbb{R}^n \mapsto \mathbb{R}$ we have:

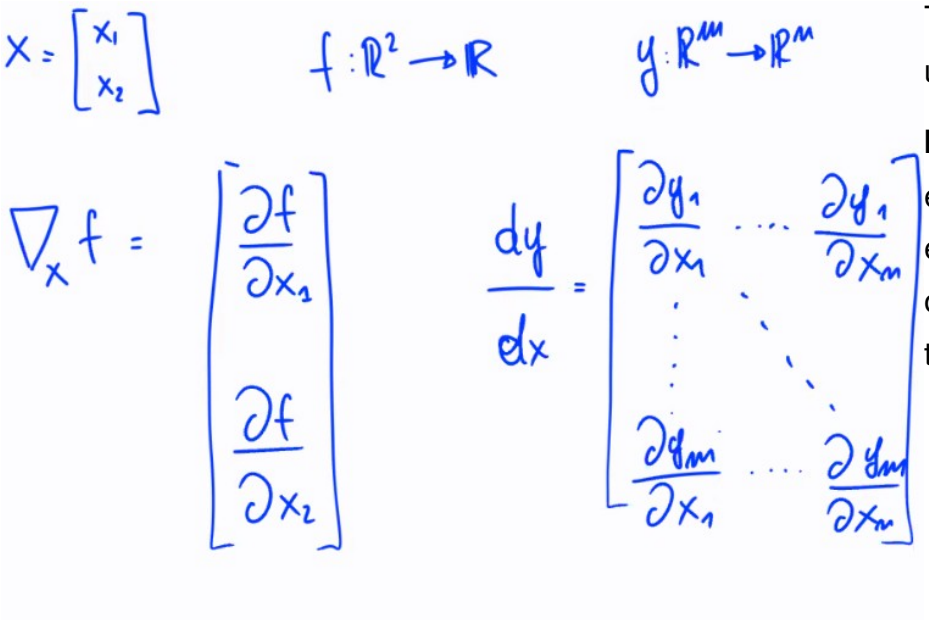
$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i},$$

It tells us how the components of the function grows when you get far from the selected point in term of distance.

equivalently in vector notation:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z,$$

with $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ the $n \times m$ Jacobian matrix of g.



Think of it as in unidimensional.

back-prop can be used either to produce an expression for the output of the function or one for the gradient.

Algorithm

Forward step

Require: Network depth l
Require: $W^{(i)}, i \in \{1, \ldots, l\}$ weight matrices
Require: $\mathbf{b}^{(i)}, i \in \{1, \ldots, l\}$ bias parameters
Require: $\mathbf{x}$ input value
Require: $\mathbf{t}$ target value

$h^{(0)} = \mathbf{x}$
for $k = 1, \ldots, l$ **do**
  $\boldsymbol{\alpha}^{(k)} = \mathbf{b}^{(k)} + W^{(k)}\mathbf{h}^{(k-1)}$
  $\mathbf{h}^{(k)} = f(\boldsymbol{\alpha}^{(k)})$
end for
$\mathbf{y} = \mathbf{h}^{(l)}$
$J = L(\mathbf{t}, \mathbf{y})$

(This version of backprop is specific for fully connected MLPs.)

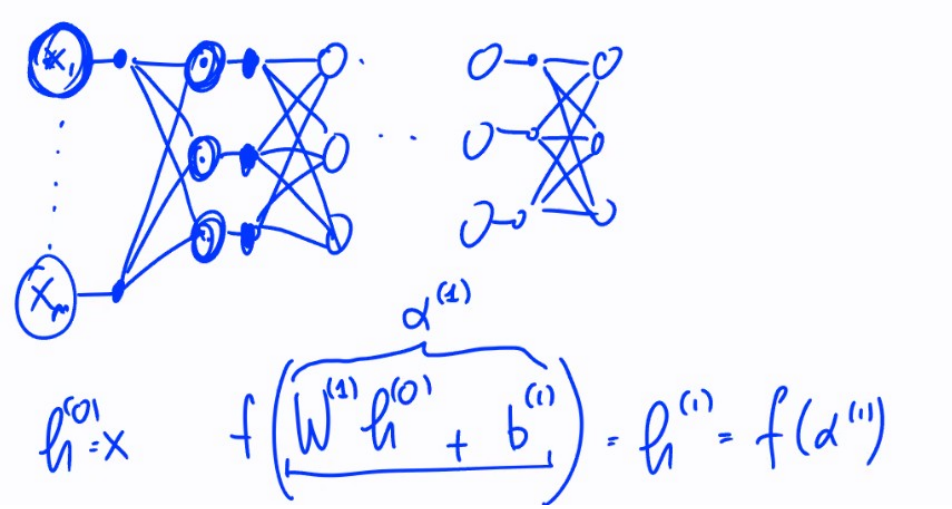
Backward step

$\mathbf{g} \leftarrow \nabla_{\mathbf{y}} J = \nabla_{\mathbf{y}} L(\mathbf{t}, \mathbf{y})$
for $k = l, l-1, \ldots, 1$ **do**
  Propagate gradients to the pre-nonlinearity activations:
  $\mathbf{g} \leftarrow \nabla_{\boldsymbol{\alpha}^{(k)}} J = \mathbf{g} \odot f'(\boldsymbol{\alpha}^{(k)})$ {$\odot$ denotes elementwise product}
  $\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g}$
  $\nabla_{W^{(k)}} J = \mathbf{g}(\mathbf{h}^{(k-1)})^T$
  Propagate gradients to the next lower-level hidden layer:
  $\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(k-1)}} J = (W^{(k)})^T \mathbf{g}$
end for

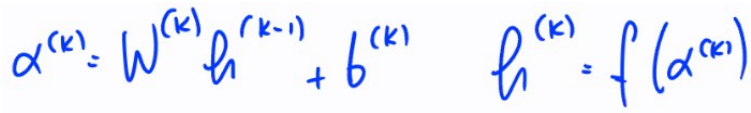
More general versions for acyclic graphs exist.

Dynamic programming is used to avoid doing the same computations multiple times.

Gradients can be computed either in symbolic or numerical form.



and soon in chain $b^{(1)}$ bias



f may change among layers

t,y referred to only one pair instance output.

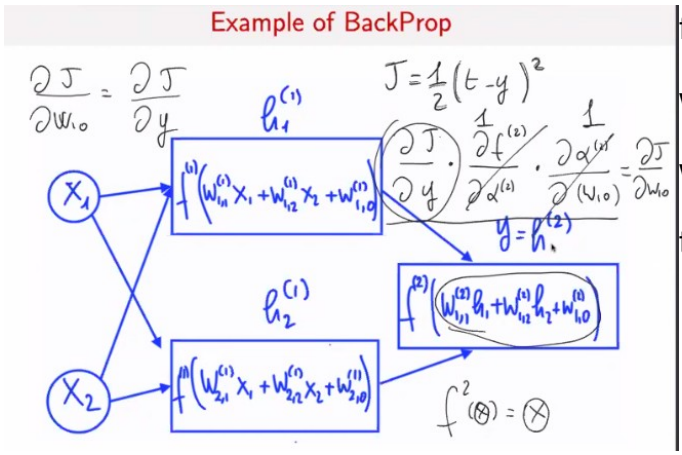
Our task is to compute all the partial derivative with respect to all these parameters because error change based on these parameters.

we proceed backward:

1. compute partial derivative referred to weights

2. apply product rule

for this specific network:

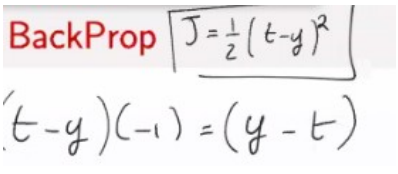


f is linear, so → x

we know alfa2 that is the internal of f2

whenever you derive y you always get 1

then derivate j wrt of y



new example

$$\frac{\partial J}{\partial w_{12}} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial \alpha^{(2)}} \cdot \frac{\partial \alpha^{(2)}}{\partial w_{12}} \quad h_2^{(1)} = (y - t) \cdot h_2$$

it's called backpropagation because we compute from the end to the beginning

Continue with the example…

$$\alpha_i^{(1)} = w_{i,0}^{(1)} + w_{i,1}^{(1)} x_1 + w_{i,2}^{(1)} x_2 \quad i = 1, 2$$

$$h_i^{(1)} = f^{(1)}(\alpha_i^{(1)}) = \texttt{ReLU}(\alpha_i^{(1)}) \quad i = 1, 2$$

$$\alpha^{(2)} = w_{1,0}^{(2)} + w_{1,1}^{(1)} h_1^{(1)} + w_{1,2}^{(1)} h_2^{(1)}$$

$$h^{(2)} = f^{(2)}(\alpha^{(2)}) = \alpha^{(2)}$$

$$y = h^{(2)}$$

Loss function MSE

$$L(t, y) = \tfrac{1}{2}(t - y)^2$$

$$\boldsymbol{\theta} = \langle w_{1,0}^{(1)}, w_{1,1}^{(1)}, w_{1,2}^{(1)}, w_{2,0}^{(1)}, w_{2,1}^{(1)}, w_{2,2}^{(1)}, w_{1,0}^{(2)}, w_{1,1}^{(2)}, w_{1,2}^{(2)} \rangle$$

Forward step

Given $x_1, x_2, w_{i,j}^{(k)}, t$

compute $\alpha_1^{(1)}, \alpha_2^{(1)}, \alpha^{(2)}, h_1^{(1)}, h_2^{(1)}, h^{(2)}, y, J = L(t, y)$

Backward step

Given $x_1, x_2, w_{i,j}^{(k)}, t, \alpha_1^{(1)}, \alpha_2^{(1)}, \alpha^{(2)}, h_1^{(1)}, h_2^{(1)}, h^{(2)}, y, J = L(t, y)$

compute $\frac{\partial J}{\partial w_{i,j}^{(k)}}$

Gradient computation

$$\frac{\partial J(\boldsymbol{\theta})}{\partial y} = \frac{\partial}{\partial y} \frac{1}{2}(t - y)^2 = y - t$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_{i,j}^{(2)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial y} \frac{\partial y}{\partial w_{i,j}^{(2)}} \quad \text{with} \quad \frac{\partial y}{\partial w_{1,0}^{(2)}} = 1, \quad \frac{\partial y}{\partial w_{1,1}^{(2)}} = h_1^{(1)} \quad \frac{\partial y}{\partial w_{1,2}^{(2)}} = h_2^{(1)}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial h_i^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial y} \frac{\partial y}{\partial h_i^{(1)}} \quad \text{with} \quad \frac{\partial y}{\partial h_1^{(1)}} = w_{1,1}^{(2)} \quad \frac{\partial y}{\partial h_2^{(1)}} = w_{1,2}^{(2)}$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \alpha_i^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial h_i^{(1)}} \frac{\partial h_i^{(1)}}{\partial \alpha_i^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial h_i} \texttt{step}(\alpha_i^{(1)})$$

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w_{i,j}^{(1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial \alpha_i^{(1)}} \frac{\partial \alpha_i^{(1)}}{\partial w_{i,j}^{(1)}} \quad \text{with} \quad \frac{\partial \alpha_i^{(1)}}{\partial w_{i,0}^{(1)}} = 1, \quad \frac{\partial \alpha_i^{(1)}}{\partial w_{i,1}^{(1)}} = x_1 \quad \frac{\partial y}{\partial w_{i,2}^{(1)}} = x_2$$

Gradient computation (in vector notation)

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{1,1}^{(1)} w_{1,2}^{(1)} \\ w_{2,1}^{(1)} w_{2,2}^{(1)} \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} w_{1,0}^{(1)} \\ w_{2,0}^{(1)} \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} w_{1,2}^{(2)} \end{bmatrix}, \mathbf{b}^{(2)} = \begin{bmatrix} w_{1,0}^{(2)} \end{bmatrix}$$

$$f^{(1)}(z) = \texttt{ReLU}(z),\ f^{(2)}(z) = z$$

$$\mathbf{h}^{(0)} = \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\boldsymbol{\alpha}^{(1)} = \begin{bmatrix} \alpha_1^{(1)} \\ \alpha_2^{(1)} \end{bmatrix} = \mathbf{W}^{(1)}\mathbf{h}^{(0)} + \mathbf{b}^{(1)}$$

$$\mathbf{h}^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \end{bmatrix} = f^{(1)}(\boldsymbol{\alpha}^{(1)}) = \begin{bmatrix} f^{(1)}(\alpha_1^{(1)}) \\ f^{(1)}(\alpha_2^{(2)}) \end{bmatrix} = \begin{bmatrix} \texttt{ReLu}(\alpha_1^{(1)}) \\ \texttt{ReLu}(\alpha_2^{(2)}) \end{bmatrix}$$

$$\boldsymbol{\alpha}^{(2)} = \left[\alpha^{(2)}\right] = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}$$

$$\mathbf{h}^{(2)} = \left[h^{(2)}\right] = f^{(2)}(\boldsymbol{\alpha}^{(2)}) = \left[f^{(2)}(\alpha^{(2)})\right] = \left[\alpha^{(2)}\right] = \boldsymbol{\alpha}^{(2)}$$

$$\mathbf{y} = \mathbf{h}^{(2)} = \boldsymbol{\alpha}^{(2)}$$

$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(2)}} J = \nabla_y J = \nabla_y \tfrac{1}{2}(t-y)^2 = \tfrac{\partial(\frac{1}{2}(t-y)^2)}{\partial y} = y - t$$

$$\mathbf{g} \leftarrow \nabla_{\alpha^{(2)}} J = \mathbf{g} \odot f^{(2)\prime}(\alpha^{(2)}) = \mathbf{g} \odot \tfrac{\partial \alpha^{(2)} - t}{\partial \alpha^{(2)}} = \mathbf{g} \odot 1 = \mathbf{g}$$

$$\nabla_{\mathbf{b}^{(2)}} J \leftarrow \frac{\partial J}{\partial w_{1,0}^{(2)}} = \mathbf{g}$$

$$\nabla_{\mathbf{W}^{(2)}} J \leftarrow \begin{bmatrix} \frac{\partial J}{\partial w_{1,1}^{(2)}} & \frac{\partial J}{\partial w_{1,2}^{(2)}} \end{bmatrix} = \mathbf{g} \cdot (\mathbf{h}^{(1)})^T$$

$$\mathbf{g} \leftarrow \nabla_{\mathbf{h}^{(1)}} J = \begin{bmatrix} \frac{\partial J}{\partial h_1^{(1)}} \\ \frac{\partial J}{\partial h_2^{(1)}} \end{bmatrix} = (\mathbf{W}^{(2)})^T \cdot \mathbf{g}$$

$$\mathbf{g} \leftarrow \nabla_{\boldsymbol{\alpha}^{(1)}} J = \mathbf{g} \odot f^{(1)\prime}(\boldsymbol{\alpha}^{(1)}) = \mathbf{g} \odot \begin{bmatrix} \frac{\partial \texttt{ReLU}(\alpha_1^{(1)})}{\partial \alpha_1^{(1)}} \\ \frac{\partial \texttt{ReLU}(\alpha_2^{(1)})}{\partial \alpha_2^{(1)}} \end{bmatrix} = \mathbf{g} \odot \begin{bmatrix} \texttt{step}(\alpha_1^{(1)}) \\ \texttt{step}(\alpha_2^{(1)}) \end{bmatrix}$$

$$\nabla_{\mathbf{b}^{(1)}} J \leftarrow \begin{bmatrix} \frac{\partial J}{\partial w_{1,0}^{(1)}} \\ \frac{\partial J}{\partial w_{2,0}^{(2)}} \end{bmatrix} = \mathbf{g}$$

$$\nabla_{\mathbf{W}^{(1)}} J \leftarrow \begin{bmatrix} \frac{\partial J}{\partial w_{1,1}^{(1)}} & \frac{\partial J}{\partial w_{1,2}^{(1)}} \\ \frac{\partial J}{\partial w_{2,1}^{(1)}} & \frac{\partial J}{\partial w_{2,2}^{(1)}} \end{bmatrix} = \mathbf{g} \cdot (\mathbf{h}^{(1)})^T$$

11.6 Stochastic Gradient Descent

Now we want to train the network use stochastic

Require: Learning rate $\eta \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of m examples from the dataset D
    Compute gradient estimate: $\mathbf{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
    Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} - \eta \mathbf{g}$
    $k \leftarrow k + 1$
  end while

Observe: $\nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{t})$ obtained with backprop

We start with some randome values of the parameters (typically small).

Training rate is not static, reduce it during the execution.

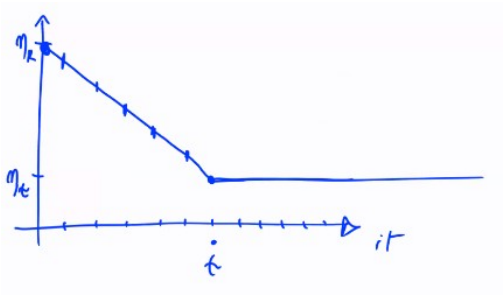
η usually changes according to some rule through the iterations

Until iteration τ ($k \leq \tau$):

$$\eta^{(k)} = \left(1 - \frac{k}{\tau}\right)\eta^{(k)} + \frac{k}{\tau}\eta^{(\tau)}$$

After iteration τ ($k > \tau$):

$$\eta^{(k)} = \eta^{(\tau)}$$



Momentum can accelerate learning
Motivation: Stochastic gradient can largely vary through the iterations

Require: Learning rate $\eta \geq 0$
Require: Momentum $\mu \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
  $k \leftarrow 1$
  $\mathbf{v}^{(1)} \leftarrow 0$
  while stopping criterion not met **do**
    Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of m examples from the dataset D
    Compute gradient estimate: $\mathbf{g} = \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
    Compute velocity: $\mathbf{v}^{(k+1)} \leftarrow \mu\mathbf{v}^{(k)} - \eta\mathbf{g}$, with $\mu \in [0, 1)$
    Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + \mathbf{v}^{(k+1)}$
    $k \leftarrow k + 1$
  end while

Only change on velocity parameter → as you learn you keep track of the rate of change.

We apply a momentum to our gradient.

Idea: you want to make gradient reduce smoothly.

velocity → change rate (called momentum)→ get it from the next iteration.

Momentumμmight also increase according to some rule through theiterations.

11.6.1 SGD with Nesterov momentum

Momentum is applied before computing the gradient

$$\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}^{(k)} + \mu\mathbf{v}^{(k)}$$

Sometimes it improves convergence rate.

Based on analysis of the gradient of the loss function it

$$\mathbf{g} = \frac{1}{m}\nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \mathbf{t}^{(i)})$$

is possible todetermine, at any step of the algorithm, whether the learning rate shouldbe increased or decreased.

11.7 Regularization

overfitting → model perform better on the data than on the train set.

we have to reduce it because it may be that adding a new instance it will not be classified well. to reduce overfit, add regularization term,

Add a regularization term E_{reg} to the cost function

$$E_{\text{reg}}(\boldsymbol{\theta}) = \sum_j |\theta_j|^q.$$

Resulting cost function:

$$\bar{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda E_{\text{reg}}(\boldsymbol{\theta}).$$

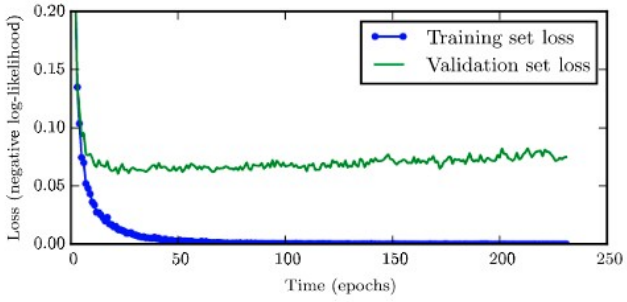
or augment dataset:

1. Data transforming (e.g. image rotation, scaling, varying, illumination conditions) by doing this you have a much larger dataset and it is more difficult to recognize it, so the model will be more “complete”.

2. Add noise

11.7.1 Early stopping

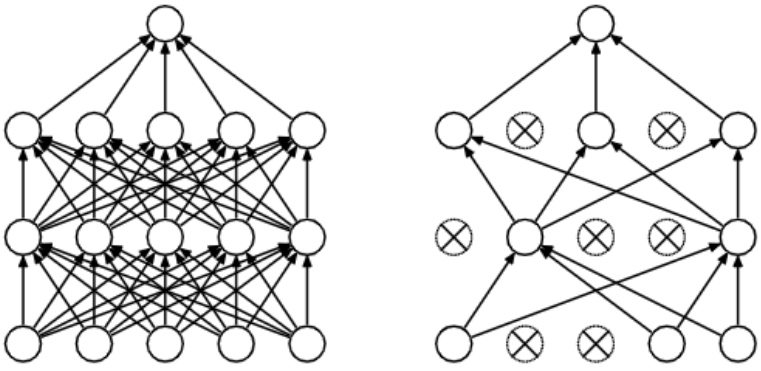
stop earlier to avoid overfitting



When to stop? Use cross-validation to determine best values.

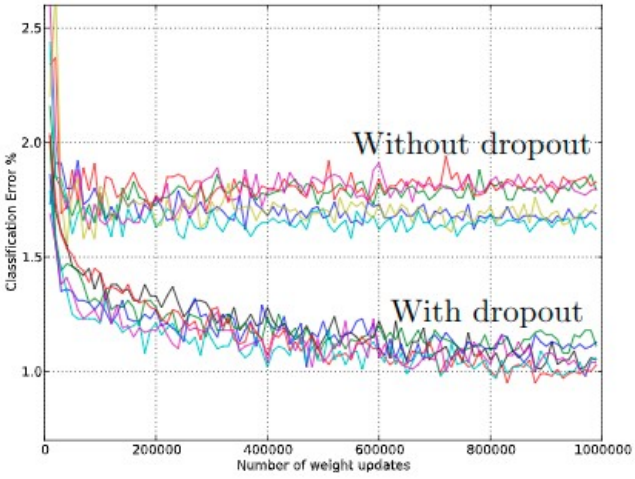
11.7.2 Dropout

Randomly remove network units with some probability alfa



(a) Standard Neural Net (b) After applying dropout.

Image from Srivastava *et al.*. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting"



FNN → cascade model

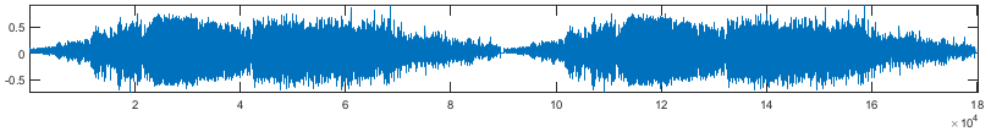
in NN the central feature, they are able to learn transformation of the input (the precedents approaches not).

12. Convolutional Neural Networks

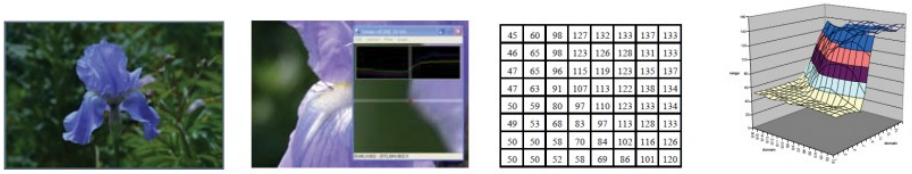
12.1 Convolution

Audio signal – vector (tensor – vectors of vectors) of variable length

Audio



...	0.0468	0.0468	0.0468	0.0390	0.0390	0.0390	0.0546	0.0625	0.0625	0.0390	0.0312	0.0468	0.0625	...



Note: multi-channel 2D matrices (3D tensor)

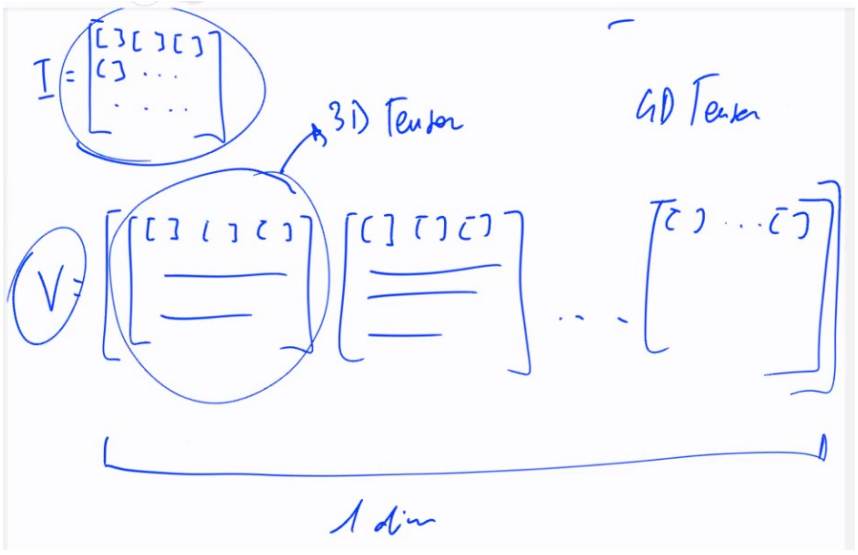


Figure 1: Example of tensor with image and video

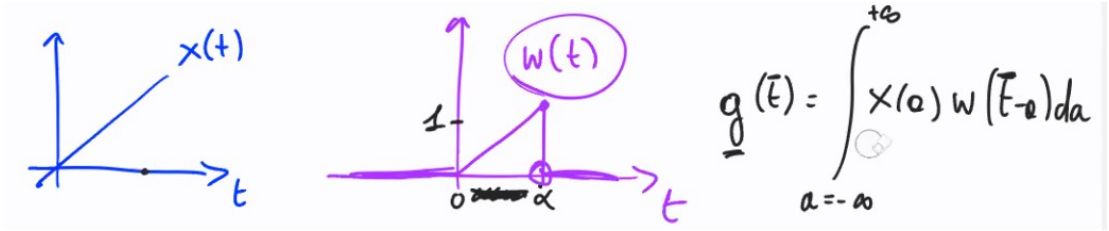
Continuous functions

Convolution operation is defined as:

$$(x * w)(t) \equiv \int_{a=-\infty}^{\infty} x(a)\, w(t-a)\, da$$

Discrete functions

$$(x * w)(t) \equiv \sum_{a=-\infty}^{\infty} x(a)\, w(t-a)$$



integral over a, so a variable and t constant

fix a certain t –> t^

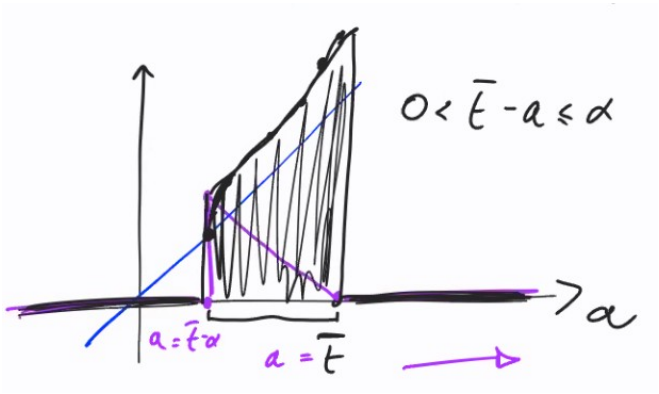
0 <= w(t^ - a) <= alfa

w(t^-a) = 0 <=> t^ = a

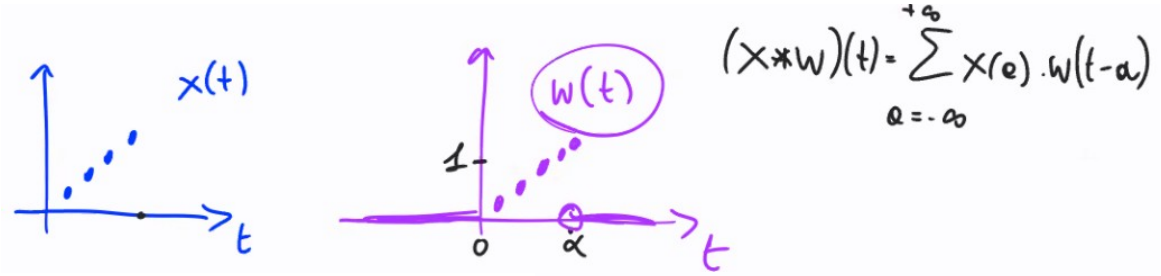
t^-alfa in [0,1]

con a = t^-alfa maximum

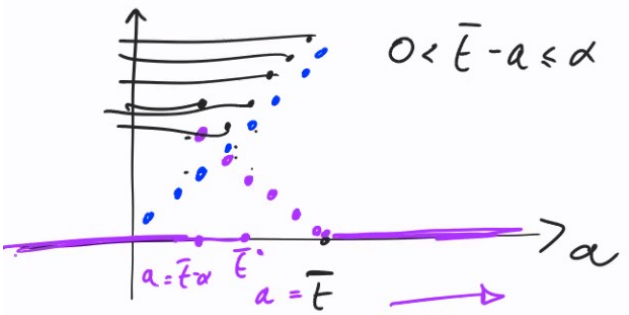
The value of g at t^ is this area →



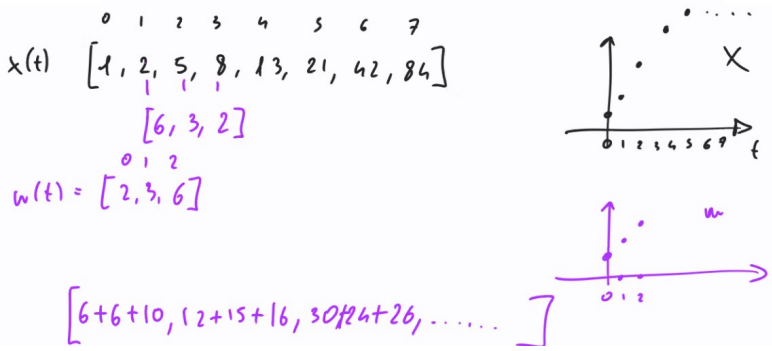
For discrete case:



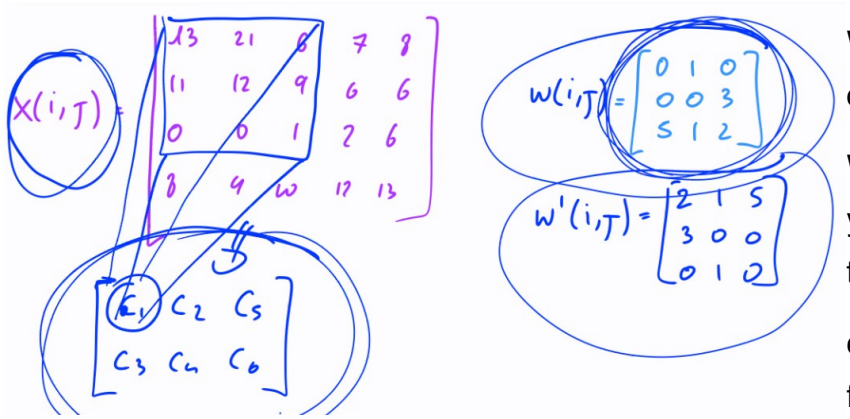
w(t-a) = flip w



Convolution: flip w, multiply w for x (if w points are less than x, slice it).



Multidimensional case:



With x less than w, we can't do convolution

When you learn the wights you are learning a trasnformation of the input

obs: w is such a kernel function.

Discrete limited 2D functions:

$$(I * K)(i,j) \equiv \sum_{m \in S_1} \sum_{n \in S_2} I(m,n) K(i-m, j-n)$$

I: 2D input, K: 2D kernel, S_i: finite sets.

Discrete limited 3D functions:

$$(I * K)(i,j,k) \equiv \sum_{m \in S_1} \sum_{n \in S_2} \sum_{u \in S_3} I(m,n,u) K(i-m, j-n, k-u)$$

I: 3D input, K: 3D kernel, S_i: finite sets.
Commutative

$$(I * K)(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

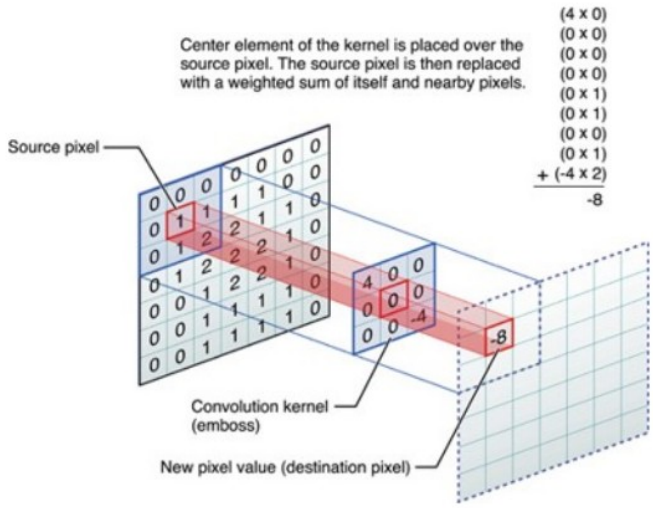
Cross-correlation

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

implemented in machine learning libraries (called convolution).

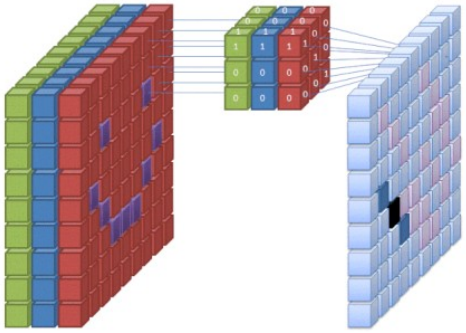
Also changing the operation, the learning algorithm will adapt.

2D Convolution for 2D input image (gray scale) with 2D kernel

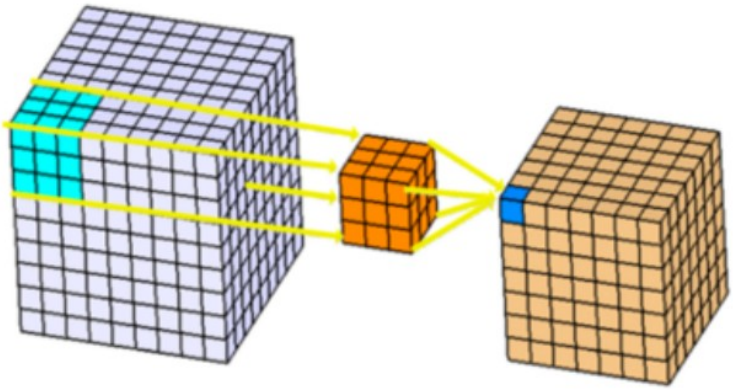


The transformation of that pixel depends on his neighbours.

2D Convolution for 3D input image (RBG channels) with 3D kernel (3 channels)



3D Convolution with 3D input and 3D kernel



Terminology

Input size (w in × h in × d in) dimensions of the input

Kernel size (w k × h k × d k) dimensions of the kernel

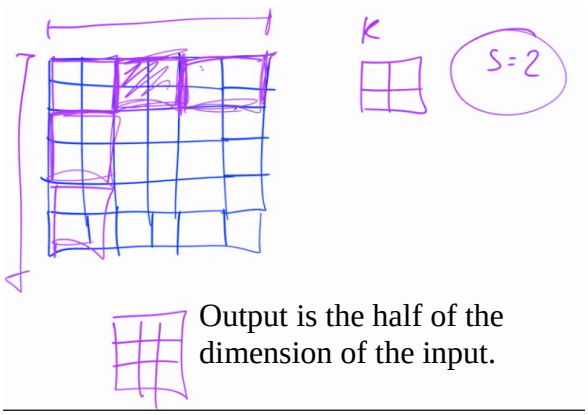
Feature map or Depth slice output of convolution between an input and one kernel

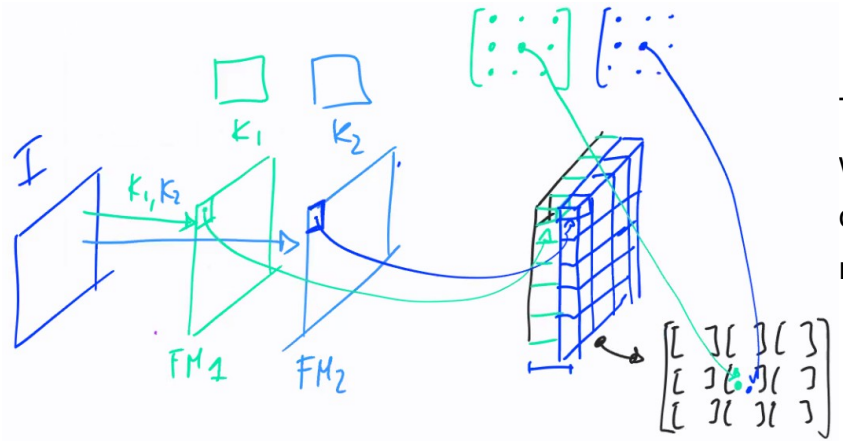
Depth (d) number of kernels (i.e., of feature maps)

Padding (p) nr. of fillers for outer rows/columns (typically zeros)

Stride (s) step of sliding kernel (1 does not skip any pixel)

Receptive field region in the input space that a particular feature is looking at (i.e., is affected by)





This with multiple kernels; if we consider this as a layer, the ouput will be the input of the next layer.

1 kernel applied to the input produces 1 feature map

Examples:
32 x 32 x 1 image * 5 x 5 x 1 kernel $\rightarrow$ 1 feature map 28 x 28
32 x 32 x 3 image * 5 x 5 x 3 kernel $\rightarrow$ 1 feature map 28 x 28

If we use d kernels, we can generate d feature maps (output of depth d)

Examples:
32 x 32 x 1 image * 6 kernels 5 x 5 x 1 $\rightarrow$ 6 feature maps 28 x 28
32 x 32 x 3 image * 6 kernels 5 x 5 x 3 $\rightarrow$ 6 feature maps 28 x 28

Note: 6 feature maps 28 x 28 are represented as a 28 x 28 x 6 tensor.

In general

Input: $w_{in} \times h_{in} \times d_{in}$

Kernels: d_{out} of size $w_k \times h_k \times d_k$ (with $d_k = d_{in}$)

Output: feature maps $w_{out} \times h_{out} \times d_{out}$

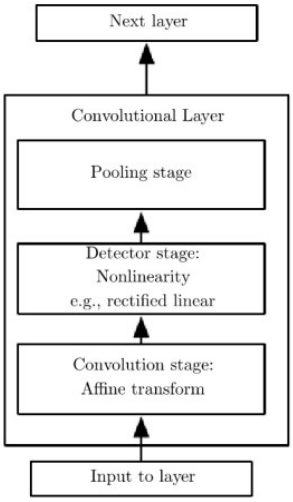
with w_{out}, h_{out} computed according to stride and padding (see next slides)

Number of kernel parameters: $w_k \cdot h_k \cdot d_k \cdot d_{out}$

Note: for 3D convolutions $d_{in} > d_k$ and output with multiple kernels is a 4D tensor $w_{out} \times h_{out} \times z_{out} \times d_{out}$, with z_{out} computed according to slide and padding in the third dimension.

Depends only on the size of the kernel

12.2 Convolutional Neural Networks Layers



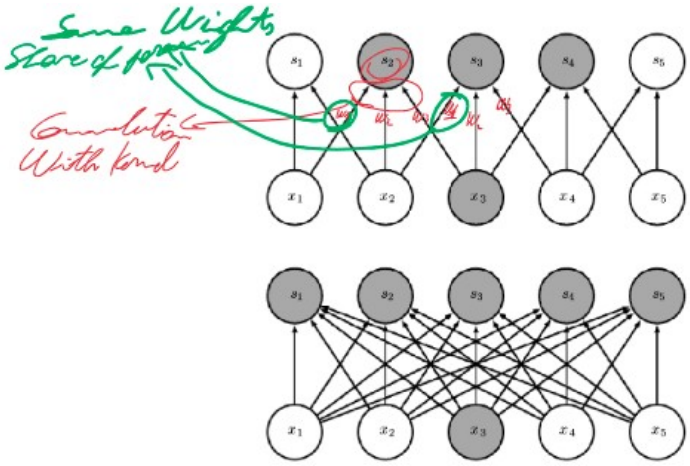
12.2.1 2D Convolution stage

We'll use 2D with several kernels

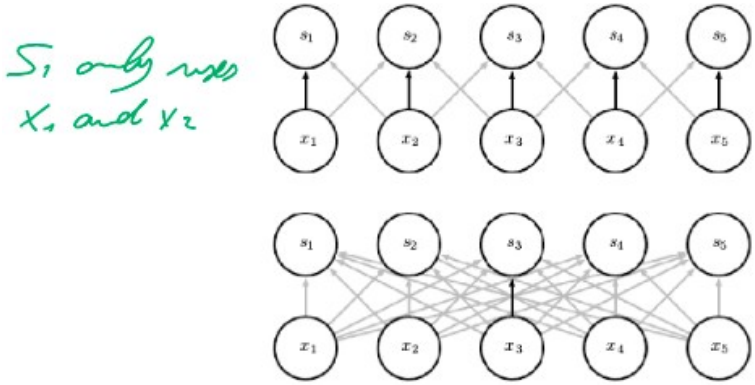
Note: It's better to have a deeper network than a wider one.

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Sparse connectivity: outputs depend only on a few inputs



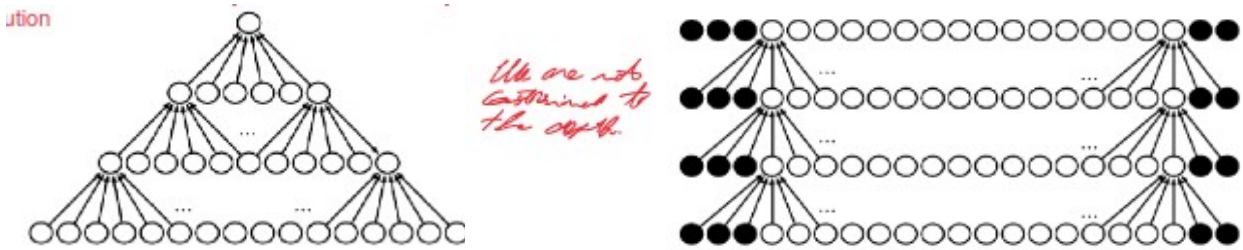
Parameter sharing: Learn only one set of parameters (for the kernel) shared to all the units. K parameters instead of m×n (note:k<<m).



Padding (Number of nodes we are adding to both sides of the network**):**

1. valid padding (p=0): only valid values (output depends on kernel size)

2. same padding(p=W_k/2): output has same size of input



12.2.2 Detector stage

Use non-linear activation functions.

- ReLU, tanh, etc.

12.2.3 Pooling stage

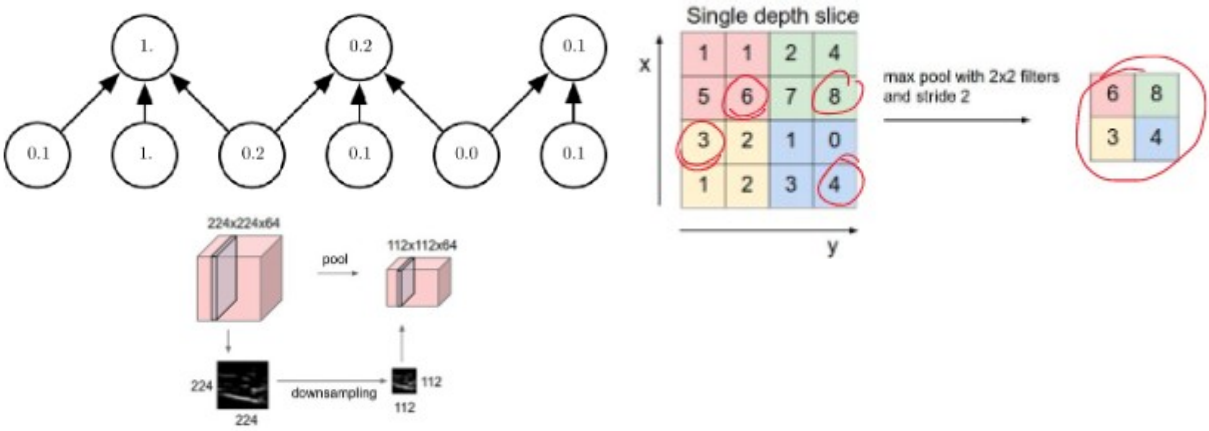
Implements invariance to local translations.

max pooling returns the maximum value in a rectangular region.

average pooling returns the average value in a rectangular region.

When applied with *stride*, it reduces the size of the output layer.

Example: max pooling with width 3 and stride 2



Consider input of size $w_{in} \times h_{in} \times d_{in}$, d_{out} kernels of size $w_k \times h_k \times d_{in}$, stride s and padding p. Dimensions of output feature map are given by:

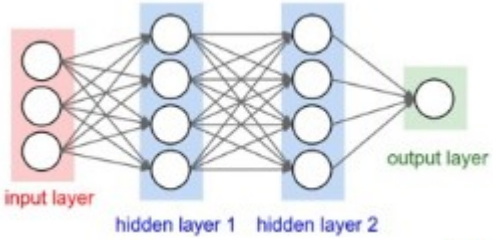
$$w_{out} = (w_{in} - w_k + 2p)/s + 1$$

$$h_{out} = (h_{in} - h_k + 2p)/s + 1$$

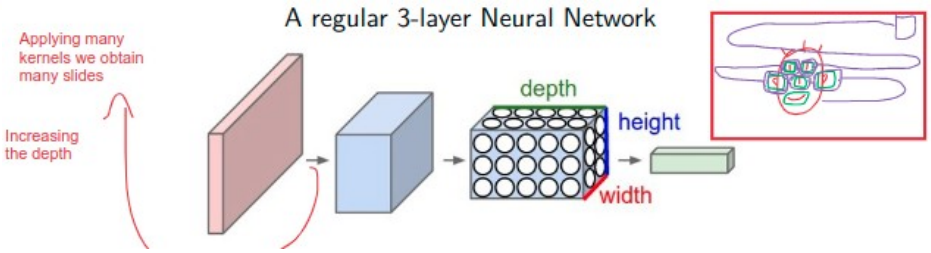
Number of trainable parameters of the convolutional layer is:

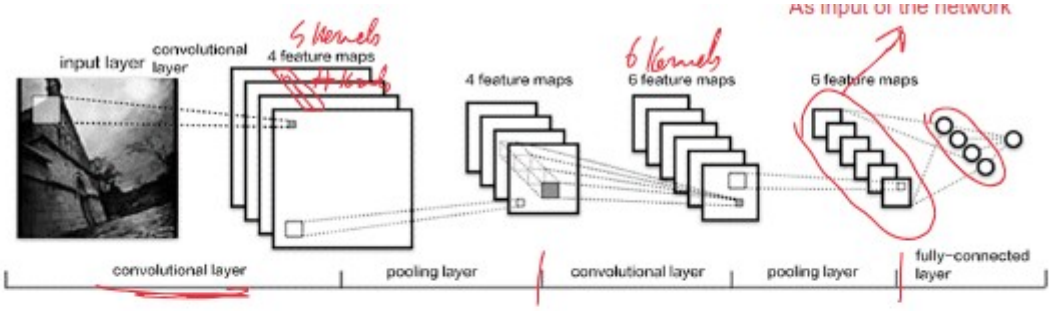
$$|\theta| = \underbrace{w_k \cdot h_k \cdot d_{in} \cdot d_{out}}_{\text{kernel weights}} + \underbrace{d_{out}}_{\text{bias}}$$

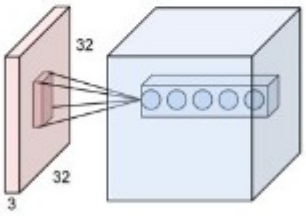
12.3 CNNs for images (2D input)

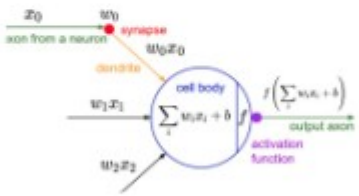


Why we use CNN for images? If you take a kernel and you slide it all ovr the image, you will detect a zone you need, e.g.: ears, nose, etc.



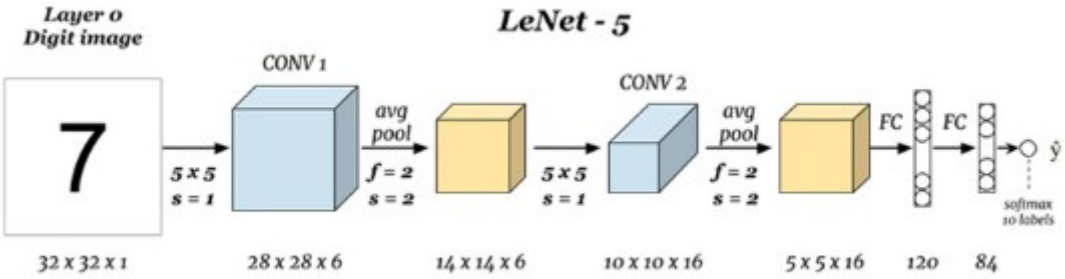


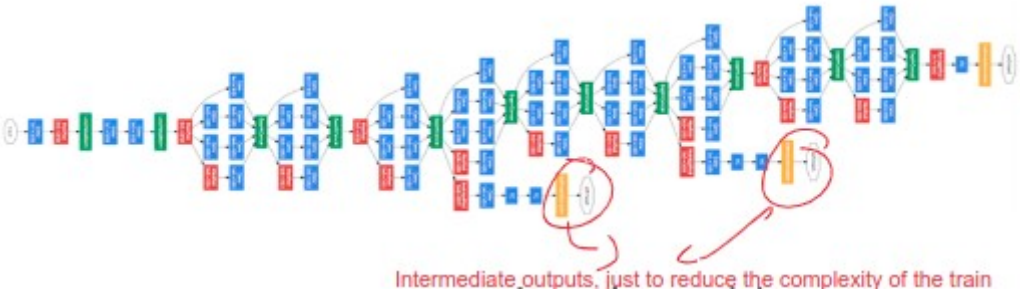




Each neuron is connected to a local 'horizontal' region of the input volume, but to **all** channels (depth)

The neurons still compute a dot product of their weights with the input followed by a non-linearity





12.4 Common uses of famous CNNs

Train a new model on a dataset

Use pre-trained models (e.g., trained on ImageNet) to:

- predict ImageNet categories for new images
- extract features to train another model (e.g., SVM)

Refine pre-trained models on a new dataset (new set of classes)

12.5 Transfer Learning

Definitions

- D is a *domain* defined by data points $\mathbf{x}_i \in \mathbf{X}$ distributed according to $\mathcal{D}(\mathbf{x})$
- T is a *learning task* defined by labels $\mathbf{y} \in \mathbf{Y}$, a target function $f : \mathbf{X} \rightarrow \mathbf{Y}$, and distribution $P_{\mathcal{D}}(\mathbf{y}|\mathbf{x})$

Given

- D_S and T_S a source domain and learning task

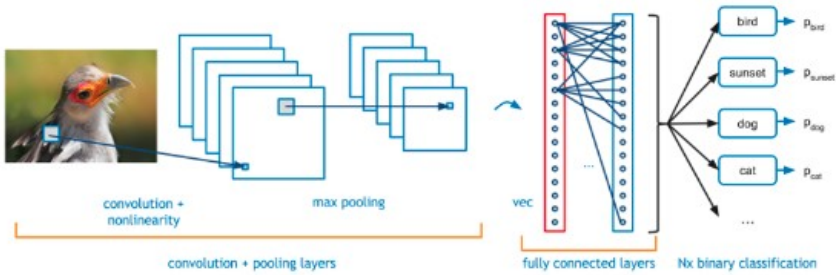
- D_T and T_T a target domain and learning task

In general, D_S != D_T and T_S != T_T

Goal: improve learning of $f_T: X_T \rightarrow Y_T$ using knowledge in D_S and T_S (i.e., after training $f_S: X_S \rightarrow Y_S$)

12.5.1 Examples

Image classification using a CNN



CNN pre-trained on Imagenet

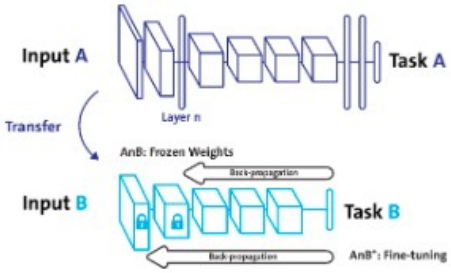
Image classification using a CNN

Use a pre-trained model for a different domain and/or learning task
E.g. Boat recognition in ARGOS dataset:

- thousands of boat images (target domain)
- classification of 20 boat classes (target learning task)

1st solution - Fine-tuning

- use same network architecture with pre-trained model
- network parameters 'copied' from the pre-trained model
- no random initialization



Strategies

- training of all network parameters
- 'freeze' parameters of some layers (usually the first ones)

Pro Full advantage of the CNN!
Con 'Heavy' training

2nd solution - CNN as feature extractor

1. extract features at a specific layer of CNN, usually:
   - last convolutional layer (flattened)
   - dense layers
2. collect extracted features $\mathbf{x}'$ of training/validation split and associate corresponding labels t in a new dataset $D' = \{(\mathbf{x}'_1, t_1), \ldots, (\mathbf{x}'_N, t_n)\}$
3. train a new classifier C' using dataset D', e.g.
   - ANN (extreme case of fine-tuning)
   - SVM
   - linear classifier
   - ...
4. classify extracted features of test set using the classifier C'

Pro No need to train the CNN!
Con Cannot modify features, source and target domains should be as 'compatible' as possible

13. Multiple Learners

Reference: Intro to ML Ch. 17, Pattern etc. Ch. 14.

Train many different learners/models and then combine results

Committes: set of models trained on a dataset.

13.1 Voting (parallel training)

1. use D to train a set of models $y_m(\mathbf{x})$, for $m = 1, \ldots M$
2. make predictions with

$$y_{voting}(\mathbf{x}) = \sum_{m=1}^{M} w_m y_m(\mathbf{x}) \quad \text{(regression)}$$

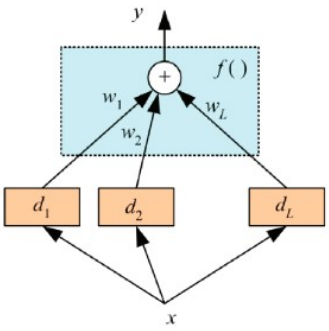


Figure 1: Voting

$$y_{voting}(\mathbf{x}) = \underset{c}{\operatorname{argmax}} \sum_{m=1}^{M} w_m I(y_m(\mathbf{x}) = c) \quad \text{weighted majority (classification)}$$

with $w_m \geq 0$, $\sum_m w_m = 1$ (prior probability of each model),
$I(e) = 1$ if e is true, 0 otherwise.

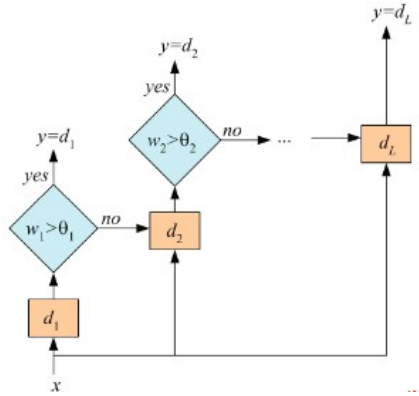


Figure 2: Cascading

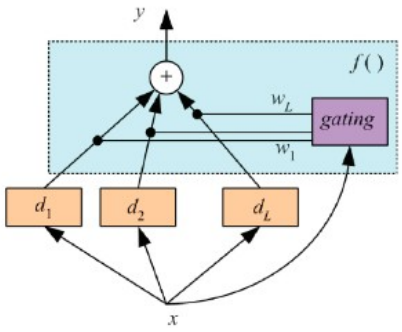


Figure 3: Mixture of experts

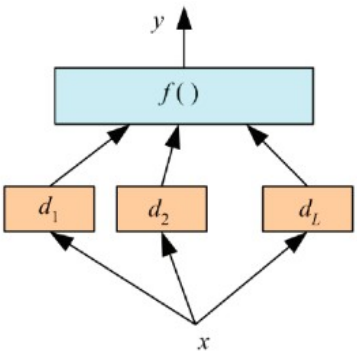


Figure 4: Stacking

13.2 Bagging (Parallel learning)

1. Generate M bootstraps datasets
2. Use each to train a model $y_m(x)$
3. make prediction with a voting scheme.

$$y_{bagging}(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x})$$

Solve the problem of data availability

Only one model or slightly different models

Better than training any individual model.

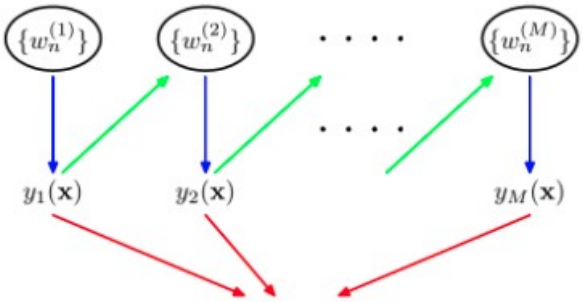
Bootstrap datasets chosen randomly

13.3 Boosting (Sequential learning)

Base classifiers trained sequentially on weighted data.

Weights depend on performance of previous classifiers (greater weights to points misclasified previously).

Predictions based on weighted majority of votes.



$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$

13.3.1 AdaBoost

Given $D = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$, where $\mathbf{x}_n \in \mathbf{X}$, $t_n \in \{-1, +1\}$

1. Initialize $w_n^{(1)} = 1/N$, $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   - Train a weak learner $y_m(\mathbf{x})$ by minimizing the weighted error function:

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n), \text{ with } I(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

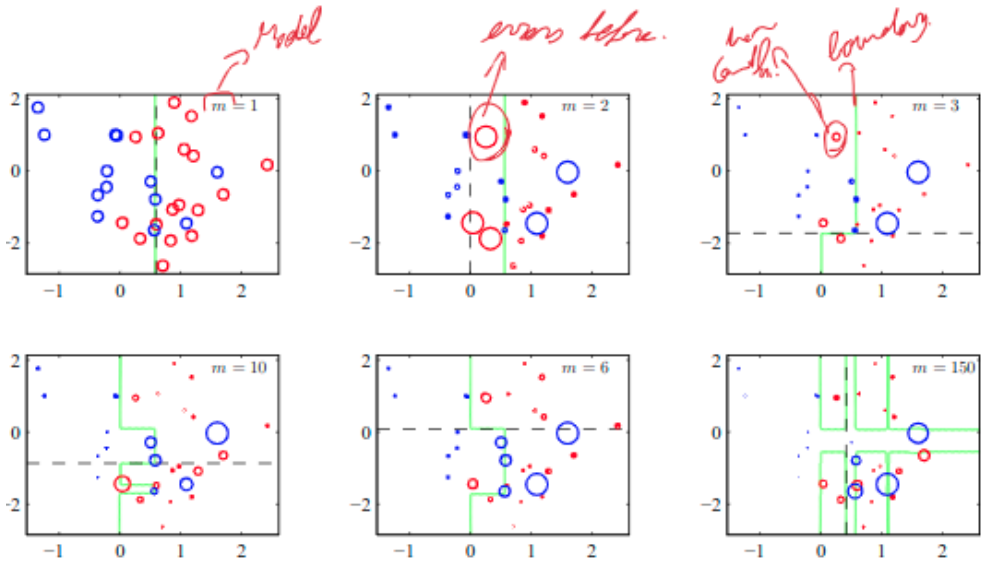
By Some way you update weights by giving more importance to the instances that have been misclassification so far, by showing that these approach actually correspond to sequentially minimizing an esoosnential error.

   - Evaluate: $\epsilon_m = \dfrac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$ and $\alpha_m = \ln\left[\dfrac{1-\epsilon_m}{\epsilon_m}\right]$
   - Update the data weighting coefficients:

$$w_n^{(m+1)} = w_n^{(m)} \exp[\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)]$$

3. Output the final classifier

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(\mathbf{x})\right)$$



13.4 Exponential error minimization

AdaBoost can be explained as the sequential minimization of anexponential error function.

$$E = \sum_{n=1}^{N} \exp[-t_n f_M(\mathbf{x}_n)],$$

Goal: minimize E

$$f_M(\mathbf{x}) = \frac{1}{2} \sum_{m=1}^{M} \alpha_m y_m(\mathbf{x}), \quad t_n \in \{-1, +1\}$$

Sequential minimization. Instead of minimizing E globally

- assume $y_1(\mathbf{x}), \ldots, y_{M-1}(\mathbf{x})$ and $\alpha_1, \ldots, \alpha_{M-1}$ fixed;
- minimize w.r.t. $y_M(\mathbf{x})$ and α_M.

Making $y_M(\mathbf{x})$ and α_M explicit we have:

$$\begin{aligned} E &= \sum_{n=1}^{N} \exp\left[-t_n f_{M-1}(\mathbf{x}_n) - \frac{1}{2} t_n \alpha_M y_M(\mathbf{x}_n)\right] \\ &= \sum_{n=1}^{N} w_n^{(M)} \exp\left[-\frac{1}{2} t_n \alpha_M y_M(\mathbf{x}_n)\right], \end{aligned}$$

with $w_n^{(M)} = \exp[-t_n f_{M-1}(\mathbf{x}_n)]$ constant as we are optimizing w.r.t. α_M and $y_M(\mathbf{x})$.

From sequential minimization of E, we obtain

$$w_n^{(m+1)} = w_n^{(m)} \exp[\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)] \text{ and } \alpha_m = \ln\left[\frac{1-\epsilon_m}{\epsilon_m}\right]$$

predictions are made with

$$\text{sign}(f_M(\mathbf{x})) = \text{sign}\left(\frac{1}{2}\sum_{m=1}^{M}\alpha_m y_m(\mathbf{x})\right)$$

which is equivalent to

$$Y_M(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{M}\alpha_m y_m(\mathbf{x})\right)$$

thus proving that AdaBoost minimizes such error function.

-Advantages:

fast, simple and easy to program

no prior knowledge about base learner is required

no parameters to tune (except forM)

can be combined with any method for finding base learners

theoretical guarantees given sufficient data and base learners withmoderate accuracy

-Issues:

Performance depends on data and the base learners(can fail with insufficient data or when base learners are too weak)

Sensitive to noise

14. Unsupervised Learning

Target values not available.

14.1 K-means

Computing K means of data generated from K Gaussian distributions.

1. Choose k = number of clusters.

2. Put initial partition that classifies the data into k clusters. Randomly or systematically.

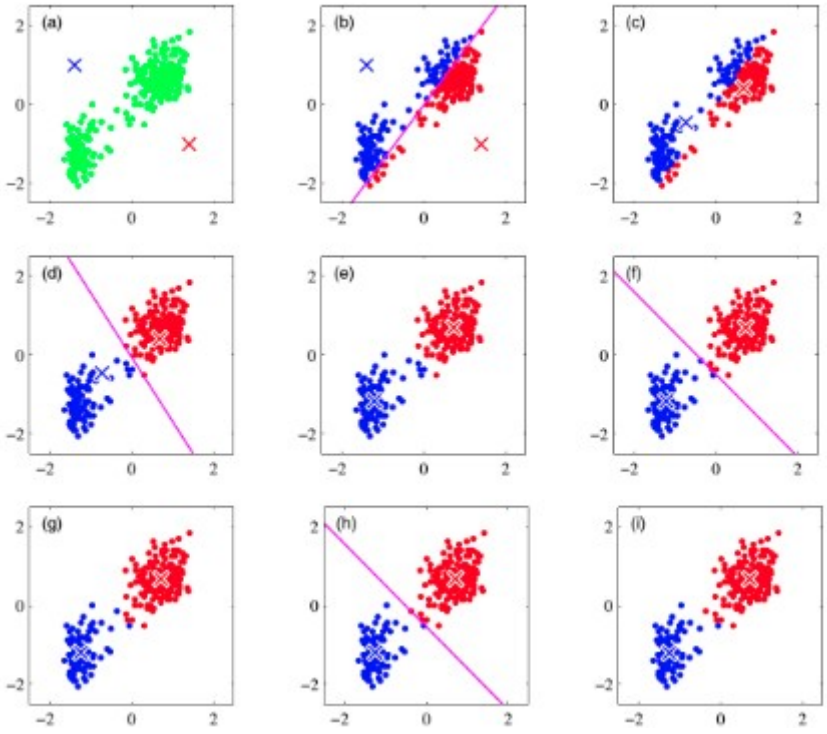
3. Take each sample in sequence and compute its distance from centroid, if it is nearer to another cluster switch and update.

4. Repeat until convergence.

Convergence occur if:

- the sum of distances from each training sample is decreasing

- there are finite partitions



Not robust to outlirs, very far data can influece centroid.

Improvements:

- use K-means clustering only if there are many data available

- use median instead of mean

- define better distance functions

14.2 Gaussian Mixture Model

Mixed probability distribution P formed by k different Gaussian distributions.

Each instance xn generated by

1 Choosing Gaussian k according to prior probabilities [π1,...,πK]

2 Generating an instance at random according to that Gaussian, thus using μk, Σk

Introduce new variables $z_k \in \{0, 1\}$, with $\mathbf{z} = (z_1, \ldots, z_K)^T$ using a 1-out-of-K encoding (only one component is 1, all the others are 0).

Let's define

$$P(z_k = 1) = \pi_k$$

thus

$$P(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

For a given value of $\mathbf{z}$:

$$P(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Thus

$$P(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

Joint distribution: $P(\mathbf{x}, \mathbf{z}) = P(\mathbf{x}|\mathbf{z})P(\mathbf{z})$ (chain rule).

When $\mathbf{z}$ are variables with 1-out-of-K encoding and $P(z_k = 1) = \pi_k$

$$P(\mathbf{x}) = \sum_{\mathbf{z}} P(\mathbf{z})P(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

GMM distribution $P(\mathbf{x})$ can be seen as the marginalization of a distribution $P(\mathbf{x}, \mathbf{z})$ over variables $\mathbf{z}$.

z_{nk}= 1 denotes x_n sampled from Gaussian k

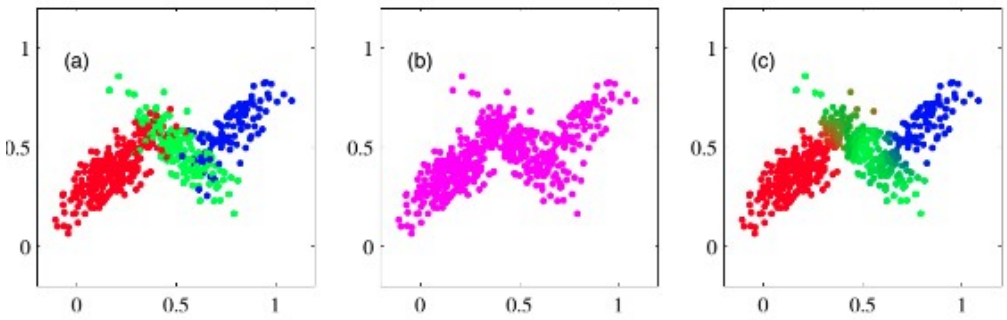
z_n are called **latent variables.**

Let's define the posterior

$$\gamma(z_k) \equiv P(z_k = 1|\mathbf{x}) = \frac{P(z_k = 1)\, P(\mathbf{x}|z_k = 1)}{P(\mathbf{x})}$$

$$\gamma(z_k) = \frac{\pi_k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

Note:
π_k: prior probability of z_k
$\gamma(z_k)$: posterior probability after observation of $\mathbf{x}$.



a) $P(\mathbf{x}, \mathbf{z})$ with 3 latent variables $\mathbf{z}$ (red, green, blue)
b) $P(\mathbf{x})$ marginalized distribution
c) $\gamma(z_{n,k})$ posterior distribution

14.3 Expectation Maximization (EM)

Note: generalization of K-means algorithm

Maximum likelihood

$$\underset{\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}}{\operatorname{argmax}} \ln P(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

At maximum:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

$$\pi_k = \frac{N_k}{N}, \quad with \;\; N_k = \sum_{n=1}^{N} \gamma(z_{nk})$$

- Initialize $\pi_k^{(0)}, \boldsymbol{\mu}_k^{(0)}, \boldsymbol{\Sigma}_k^{(0)}$
- Repeat until termination condition $t = 0, \ldots, T$
  - **E step**

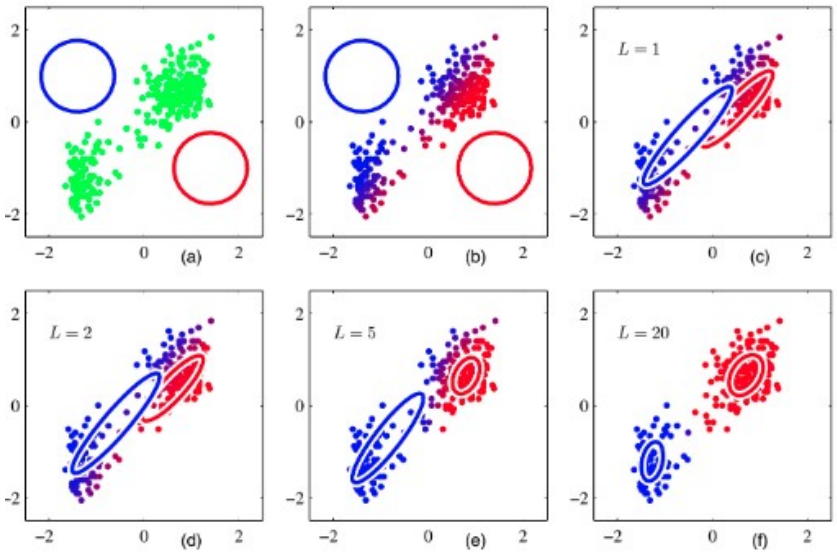
$$\gamma(z_{nk})^{(t+1)} = \frac{\pi_k^{(t)} \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{j=1}^{K} \pi_j^{(t)} \mathcal{N}(\mathbf{x}_k; \boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)})}$$

  - **M step**

$$\boldsymbol{\mu}_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})^{(t+1)} \mathbf{x}_n$$

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})^{(t+1)} (\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)})^T$$

$$\pi_k^{(t+1)} = \frac{N_k}{N}, \quad with \;\; N_k = \sum_{n=1}^{N} \gamma(z_{nk})^{(t+1)}$$



Converges to local maximum likelihood; Provides estimates of the latent variables variables z_{nk} ;Not only gaussian

Define likelihood function $Q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ defined on variables $\mathbf{Y} = \mathbf{X} \cup \mathbf{Z}$, using observed $\mathbf{X}$ and current parameters $\boldsymbol{\theta}$ to estimate $\mathbf{Z}$

EM Algorithm:

Estimation (E) step: Calculate $Q(\boldsymbol{\theta}'|\boldsymbol{\theta})$ using current hypothesis $\boldsymbol{\theta}$ and observed data $\mathbf{X}$ to estimate probability distribution over $\mathbf{Y}$

$$Q(\boldsymbol{\theta}'|\boldsymbol{\theta}) \leftarrow E[\ln P(\mathbf{Y}|\boldsymbol{\theta}')|\boldsymbol{\theta}, \mathbf{X}]$$

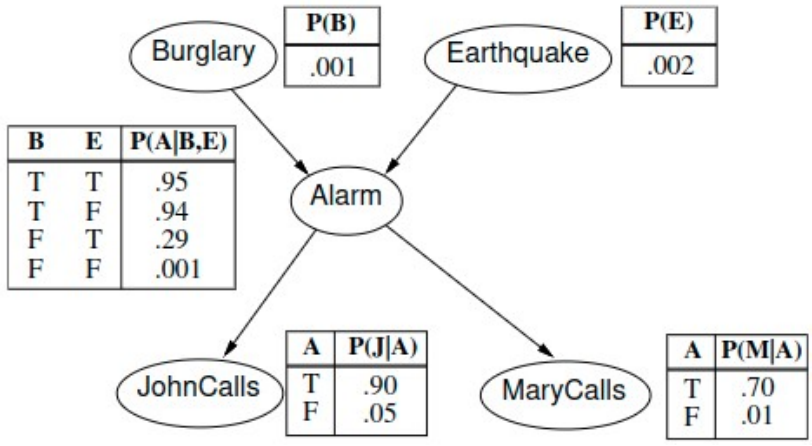
Maximization (M) step: Replace hypothesis $\boldsymbol{\theta}$ by the hypothesis $\boldsymbol{\theta}'$ that maximizes this Q function

$$\boldsymbol{\theta} \leftarrow \underset{\boldsymbol{\theta}'}{\operatorname{argmax}} \, Q(\boldsymbol{\theta}'|\boldsymbol{\theta})$$

14.4 Bayesian Network

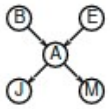
Examples of cavity, tooth etc…

Example:



All joint probabilities computed with the chain rule:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{Parents}(X_i))$$



e.g., $P(j \wedge m \wedge a \wedge \neg b \wedge \neg e)$

A CPT for Boolean variable X_i with k Boolean parents has 2^k rows for the combinations of parent values.

If each variable has no more than k parents, the complete network requires $O(n \cdot 2^k)$ numbers.

When structure known and all variables observable conditional probabilities can be estimated with maximum likelihood.

Unsupervised learning can be seen as learning with a BN with one hidden variable (the class of the instances). This can be generalized to general BN with multiple hidden variables.

Example:

Consider three random variables $X \in \{0,1\}$, $A \in \{a_1, a_2\}$, $B \in \{b_1, b_2\}$, with X unobservable.

How can we learn BN parameters for $P(X)$, $P(A|X)$, and $P(B|X)$ from instances $D = \{d_1, ..., d_n\}$, with $d_k = \langle a_k, b_k \rangle$?

Define:
$P(X = 0) = \theta_0$, $P(A = a_1|X = 0) = \theta_1$, $P(A = a_1|X = 1) = \theta_2$,
$P(B = b_1|X = 0) = \theta_3$, $P(B = b_1|X = 1) = \theta_4$

$\boldsymbol{\theta} = \langle \theta_0, \theta_1, \theta_2, \theta_3, \theta_4 \rangle$

Apply EM method to find maximum likelihood wrt $\boldsymbol{\theta}$ from D.

Estimation of BN parameters:

$$\begin{aligned} P(X = x_j) &= \frac{1}{n} E[\hat{N}(X = x_j)] \\ P(A = a_i|X = x_j) &= \frac{E[\hat{N}(A = a_i, X = x_j)]}{E[\hat{N}(X = x_j)]} \end{aligned}$$

Note that

$$E[\hat{N}(\cdot)] = E[\sum_k I(\cdot|d_k)] = \sum_k P(\cdot|d_k)$$

Estimation of BN parameters:

$$\begin{aligned} P(X = x_j) &= \frac{1}{n} \sum_{k=1}^{n} P(X = x_j|d_k) \\ P(A = a_i|X = x_j) &= \frac{\sum_{k=1}^{n} P(A = a_i, X = x_j|d_k)}{\sum_{k=1}^{n} P(X = x_j|d_k)} \\ P(B = b_l|X = x_j) &= \ldots \end{aligned}$$

Apply Bayes rule

$$\begin{aligned} P(x_j|d_k) &= P(x_j|\langle a_k, b_k \rangle) = \frac{P(a_k|x_j)P(b_k|x_j)}{\sum_i P(a_k|x_i)P(b_k|x_i)P(x_i)} = \phi_1(\boldsymbol{\theta}) \\ P(a_i, x_j|d_k) &= P(a_i|x_j, d_k)P(x_j|d_k) = \phi_2(\boldsymbol{\theta}) \\ P(b_l, x_j|d_k) &= P(b_l|x_j, d_k)P(x_j|d_k) = \phi_3(\boldsymbol{\theta}) \end{aligned}$$

... to define $Q(\boldsymbol{\theta}'|\boldsymbol{\theta})$

Unsupervised learning useful to deal with unknown variables

EM algorithm is a general method to estimate likelihood for mixed distributions

Concepts to be extended to continuous latent variables