CLASSIFICATION—ADVANCED

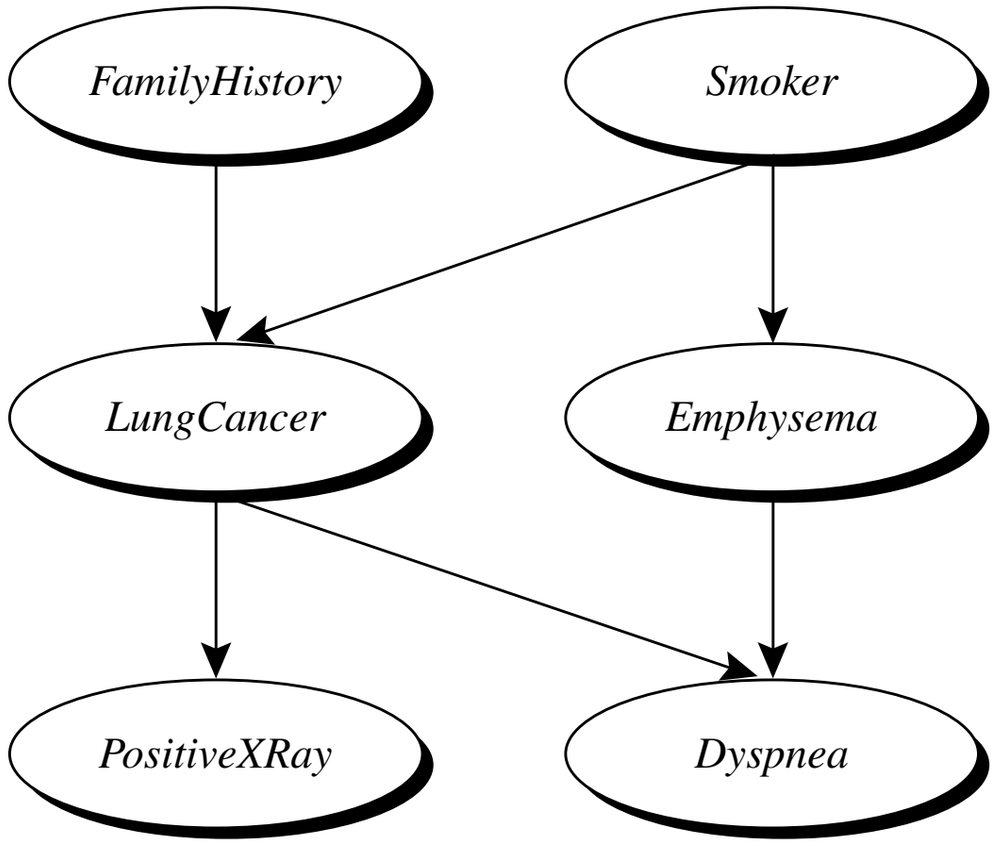
Hari Sundaram

hs1@illinois.edu

http://sundaram.cs.illinois.edu

adapted from slides by Jiawei Han and Kevin Chang

BAYESIAN BELIEF NETWORKS



Support Vector Machines Neural Networks Etc

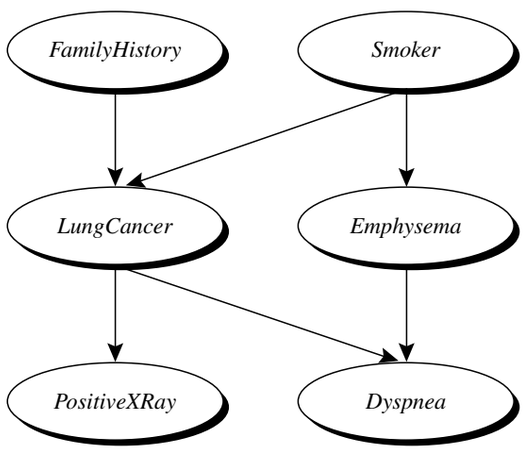
Summary

a set of conditional probability tables (CPTs)

	FH, S	*FH, ~S*	*~FH, S*	*~FH, ~S*
LC	0.8	0.5	0.7	0.1
~LC	0.2	0.5	0.3	0.9

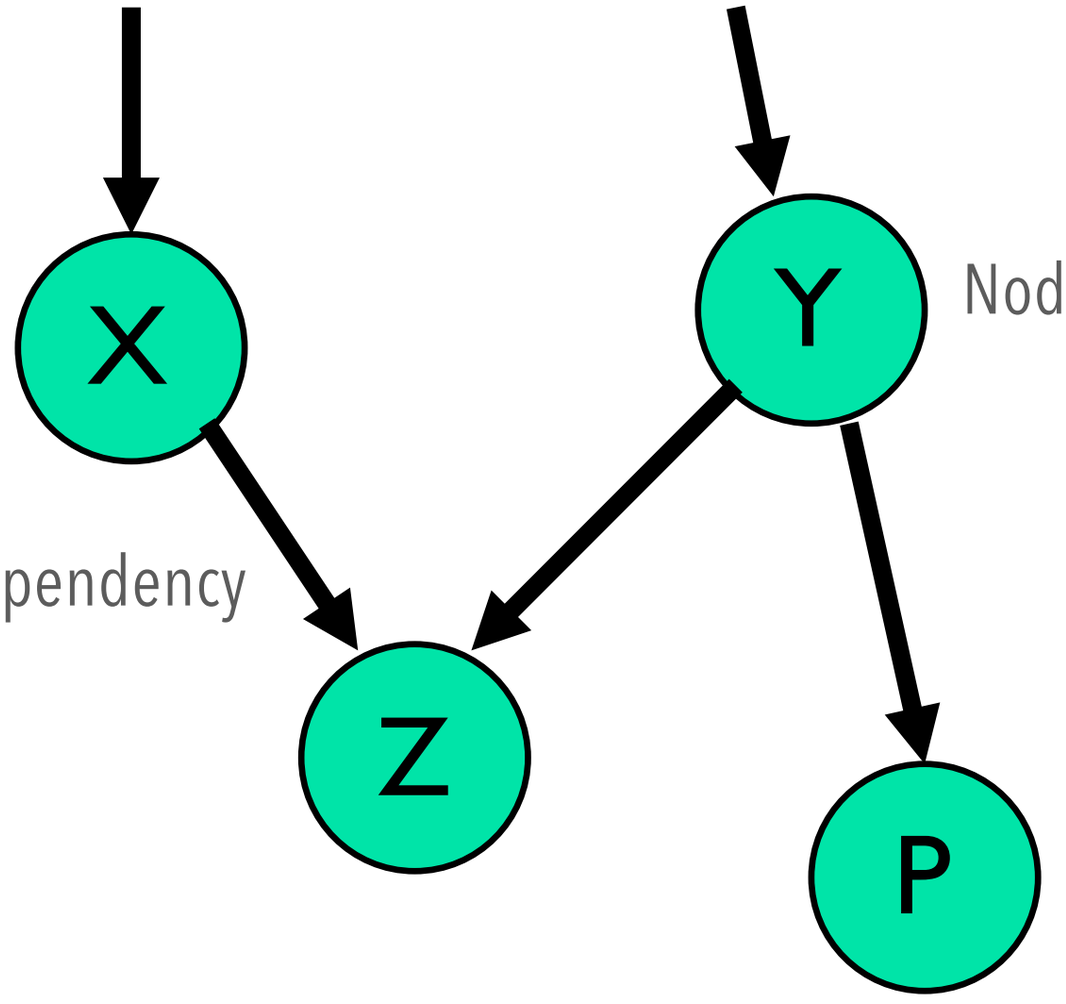
bayesian belief networks

A directed acyclic graph (called a structure)



Bayesian belief networks allow class conditional independencies between subsets of variables

X and Y are the parents of Z, and Y is the parent of P

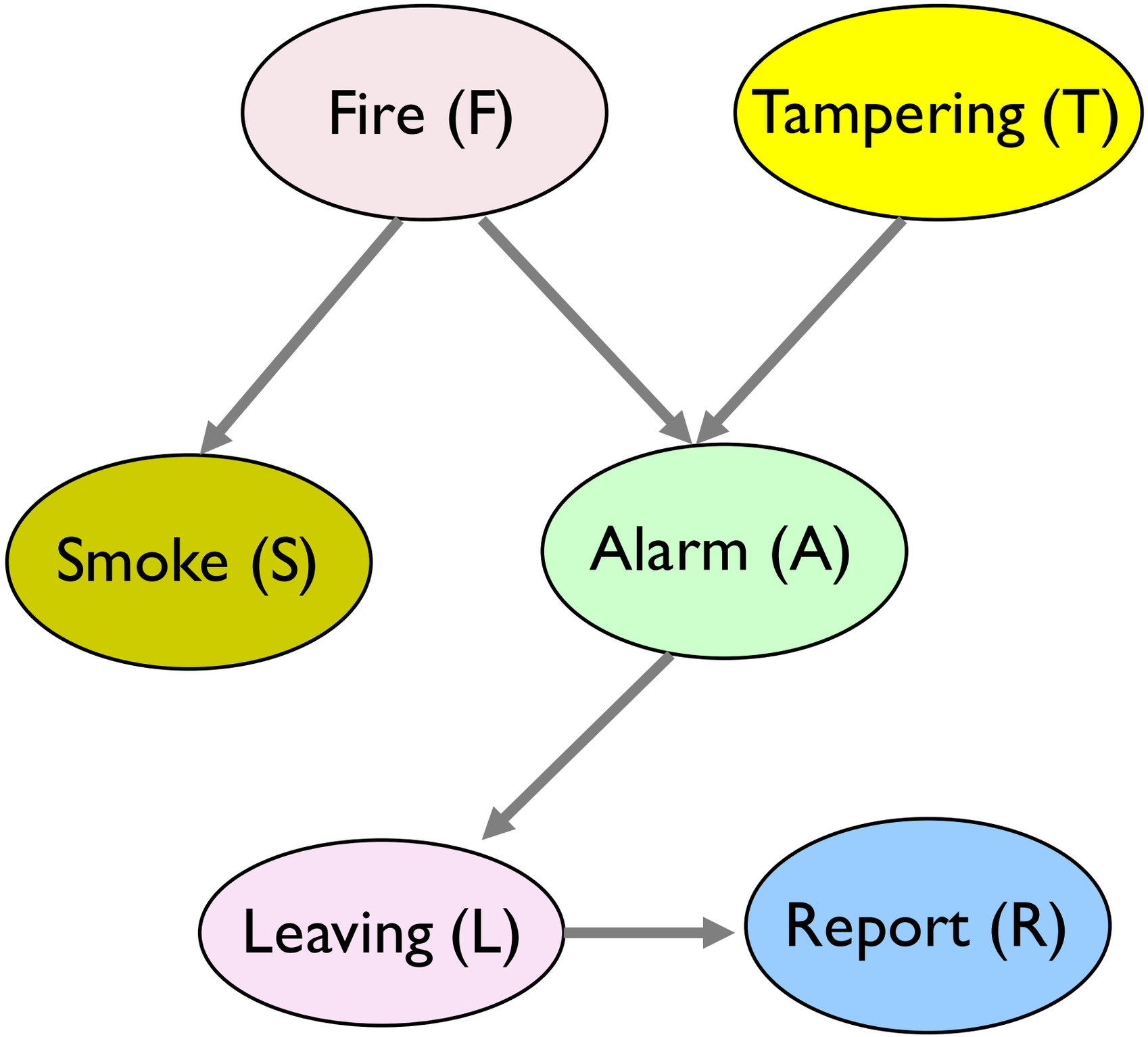


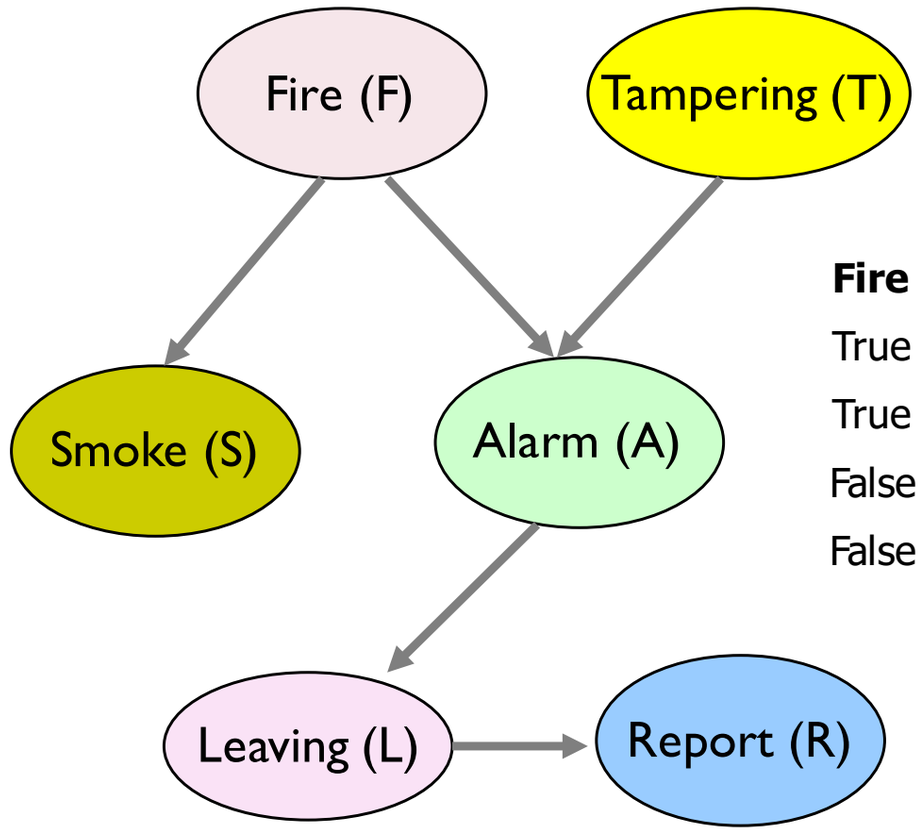
Nodes: random variables

Links: dependency

no loops/cycles

No dependency between Z and P

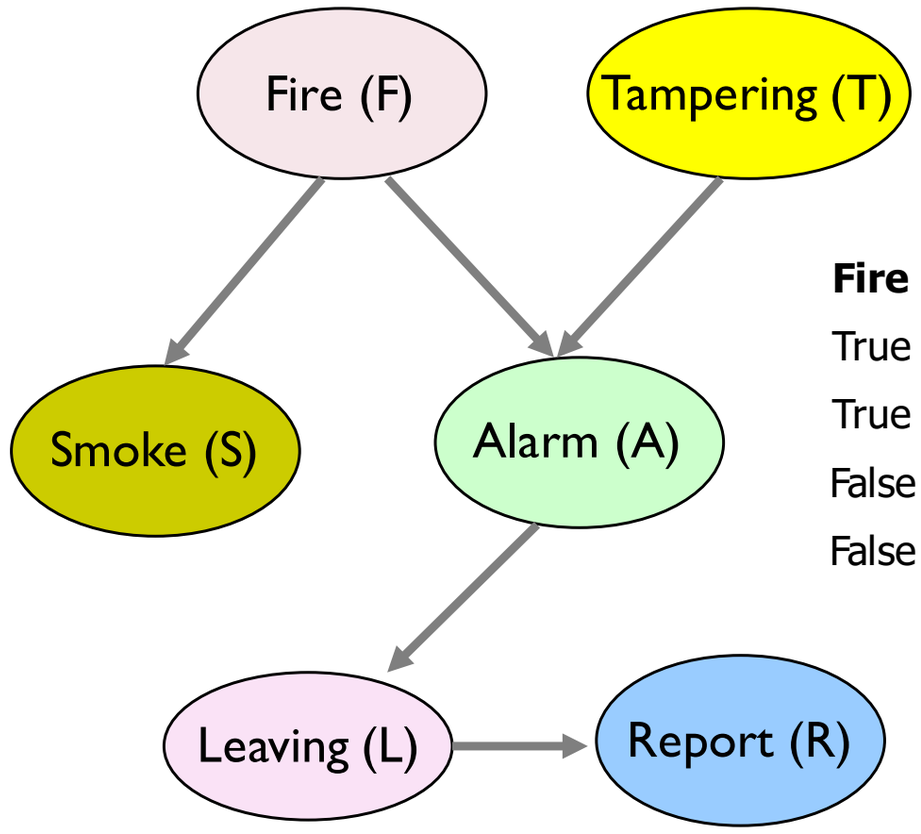

Fire (F)
Tampering (T)
Smoke (S)
Alarm (A)
Leaving (L)
Report (R)



Fire	Smoke	P(S=T|F)
True	True	.90
False	True	.01

Fire	Tampering	Alarm	P(A=T | F,T)
True	True	True	.5
True	False	True	.99
False	True	True	.85
False	False	True	.0001

CPT shows the conditional probability for each possible combination of its parents



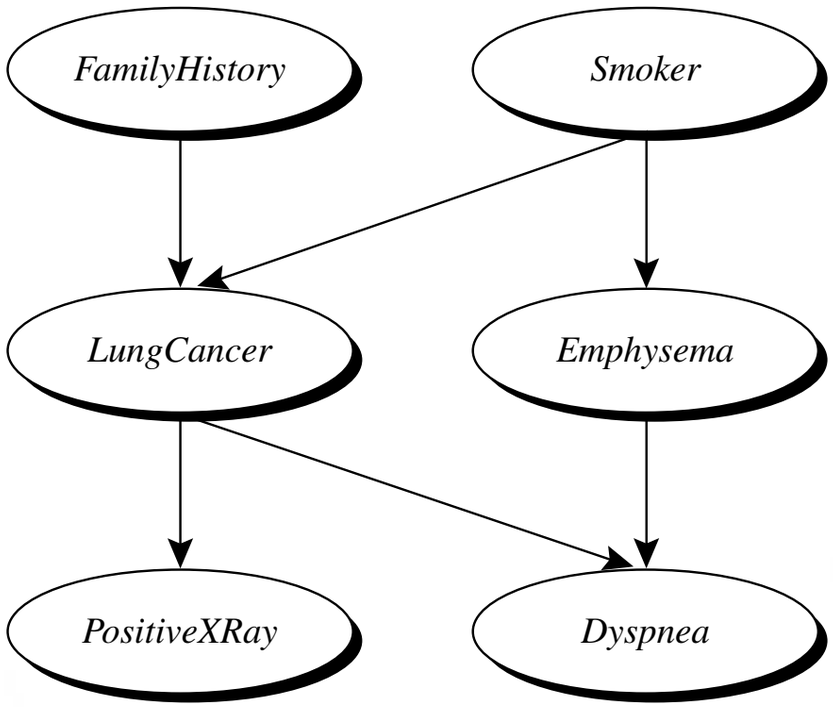
Fire	Smoke	P(S=T|F)
True	True	.90
False	True	.01

Fire	Tampering	Alarm	P(A=T | F,T)
True	True	True	.5
True	False	True	.99
False	True	True	.85
False	False	True	.0001

CPT shows the conditional probability for each possible combination of its parents

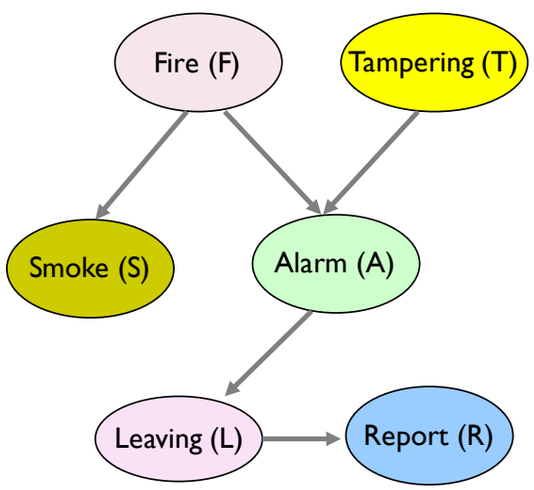
$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | Parents(x_i))$$

does a belief network allow for causal inference?



how do we construct these networks?

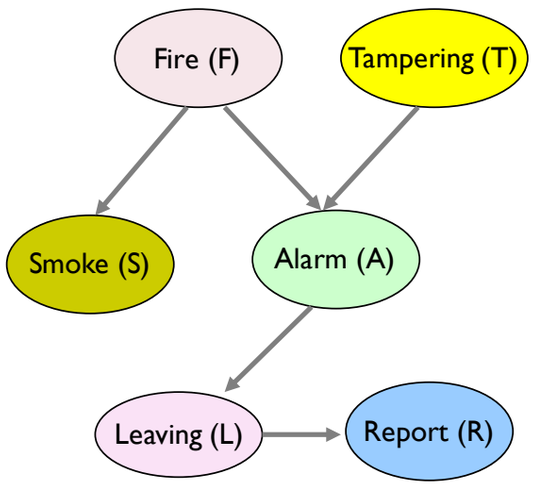
Subjective construction



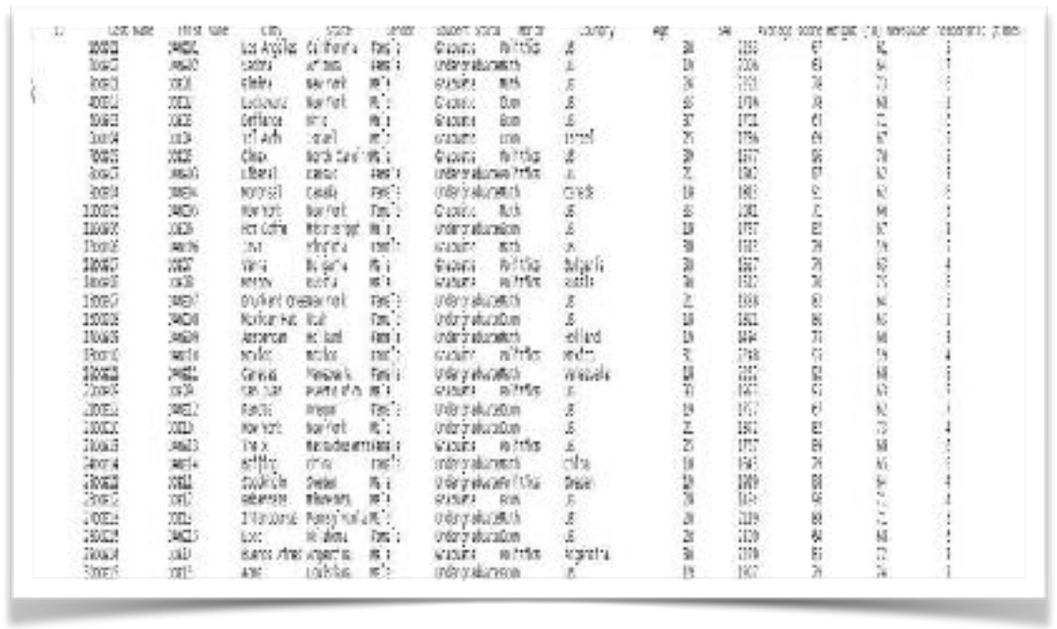
Identification of (direct) causal structure

People are quite good at identifying direct causes from a given set of variables & whether the set contains all relevant direct causes

Markovian assumption



Each variable becomes independent of its non-effects once its direct causes are known



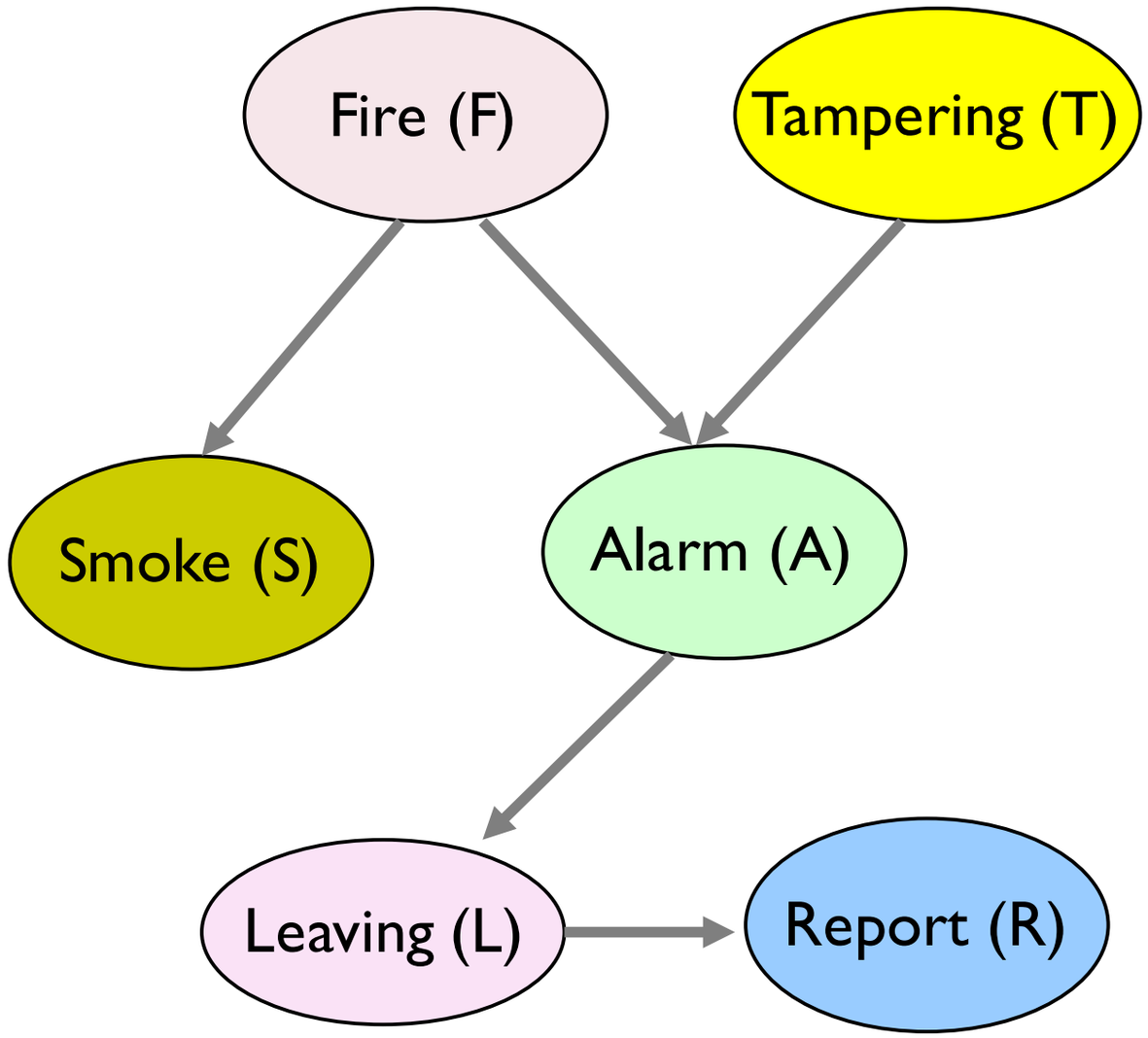
Learning from data

from medical records or student admission record

Learn parameters given
network structure or learn
both structure and
parameters

Maximum likelihood principle

TRAINING A BBN



D. Heckerman. A Tutorial on Learning with Bayesian Networks. In Learning in Graphical Models, M. Jordan, ed. MIT Press, 1999.

Scenario 1: Given both the network structure and all variables observable: compute only the CPT entries

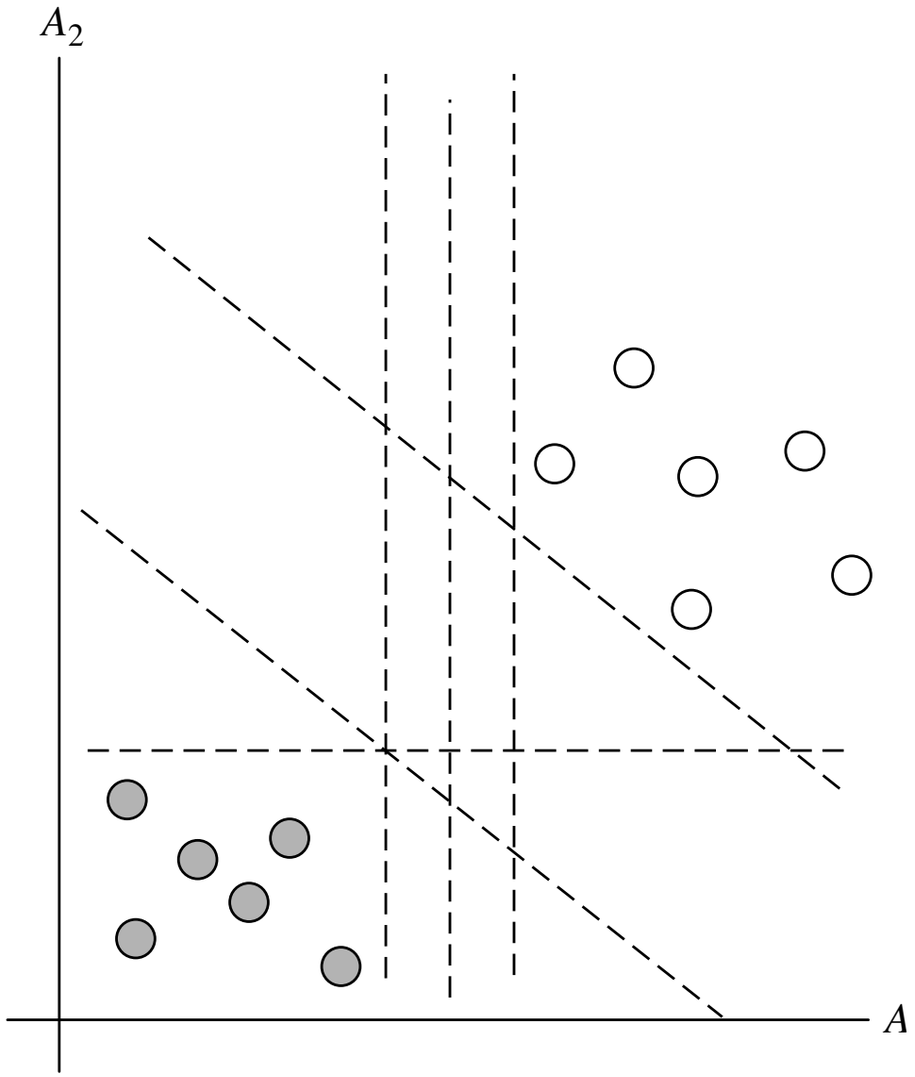
Scenario 2: Network structure known, some variables hidden: gradient descent (greedy hill-climbing) method, i.e., search for a solution along the steepest descent of a criterion function

- Weights are initialized to random probability values
- At each iteration, it moves towards what appears to be the best solution at the moment, w.o. backtracking
- Weights are updated at each iteration & converge to local optimum

Scenario 3: Network structure unknown, all variables observable: search through the model space to reconstruct network topology

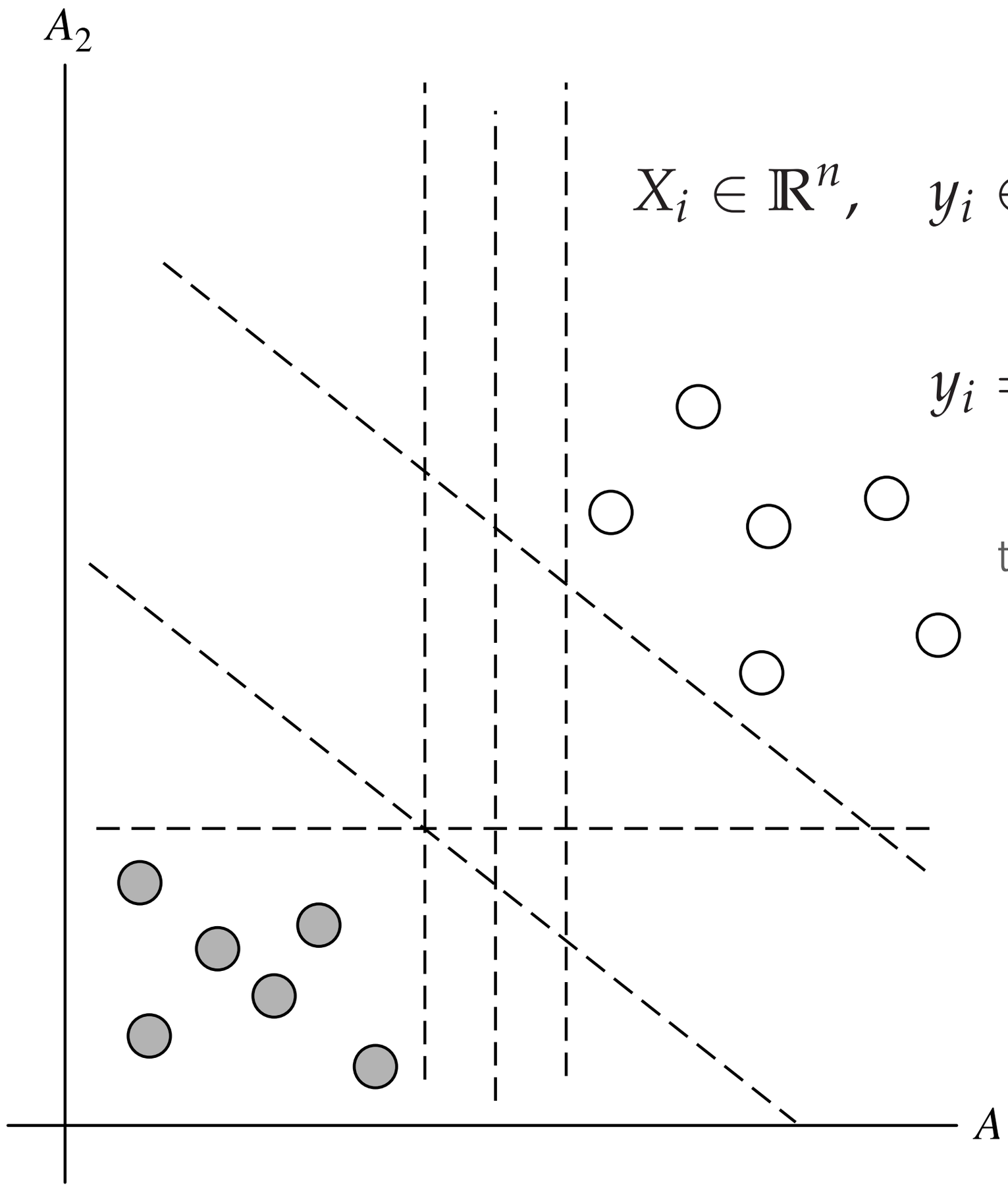
Scenario 4: Unknown structure, all hidden variables: No good algorithms known for this purpose

SUPPORT VECTOR MACHINES



Bayesian Belief Networks Neural Networks k-NN

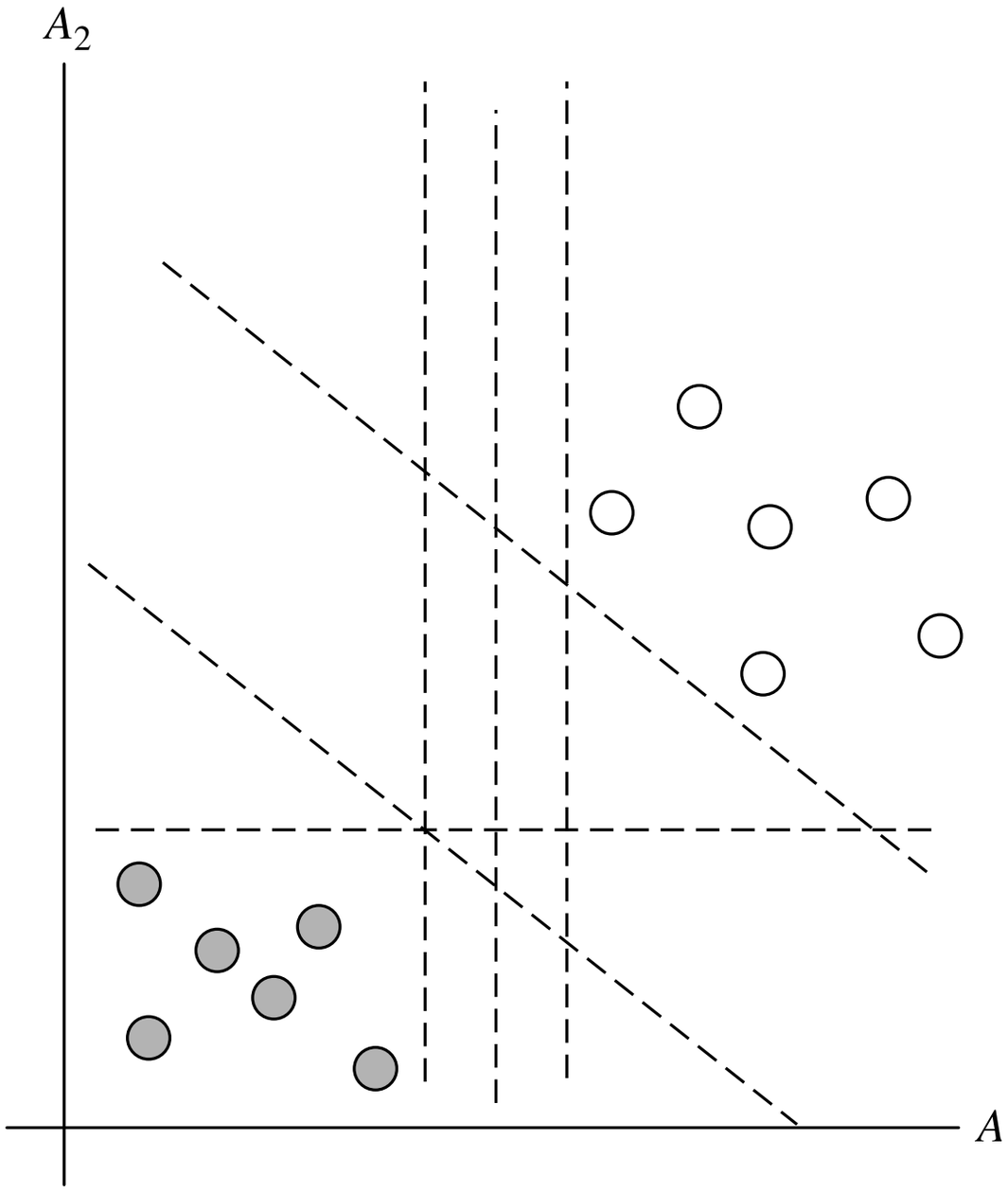
Multiclass Classification Summary



$$X_i \in \mathbb{R}^n, \quad y_i \in \{+1, -1\}$$

$$y_i = f(X_i)$$

the goal



ADVANTAGES

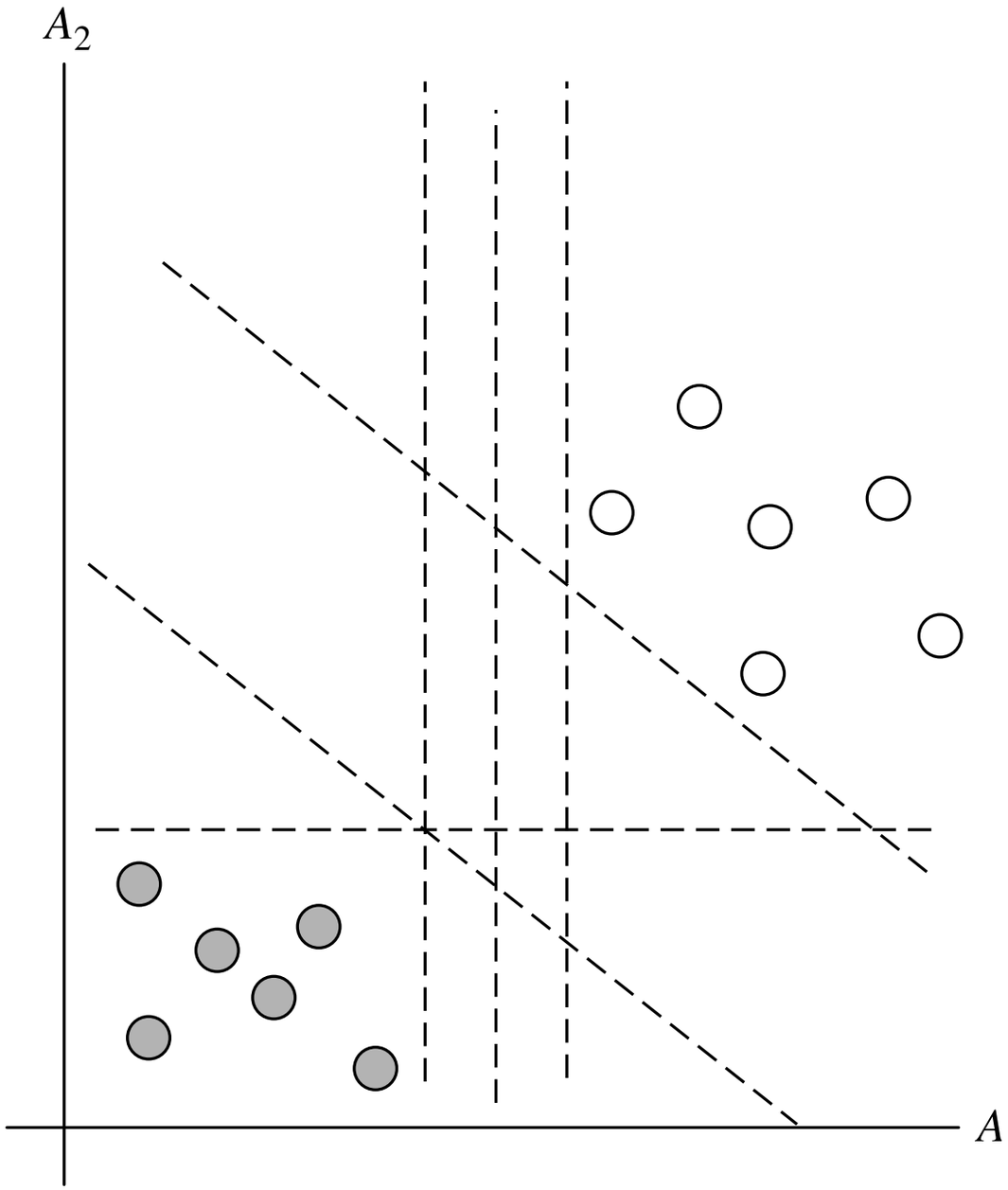
Prediction accuracy is generally high

As compared to Bayesian methods – in general

Robust, works when training examples contain errors

Fast evaluation of the learned target function

Bayesian networks are normally slow



CRITICISMS

Long training time

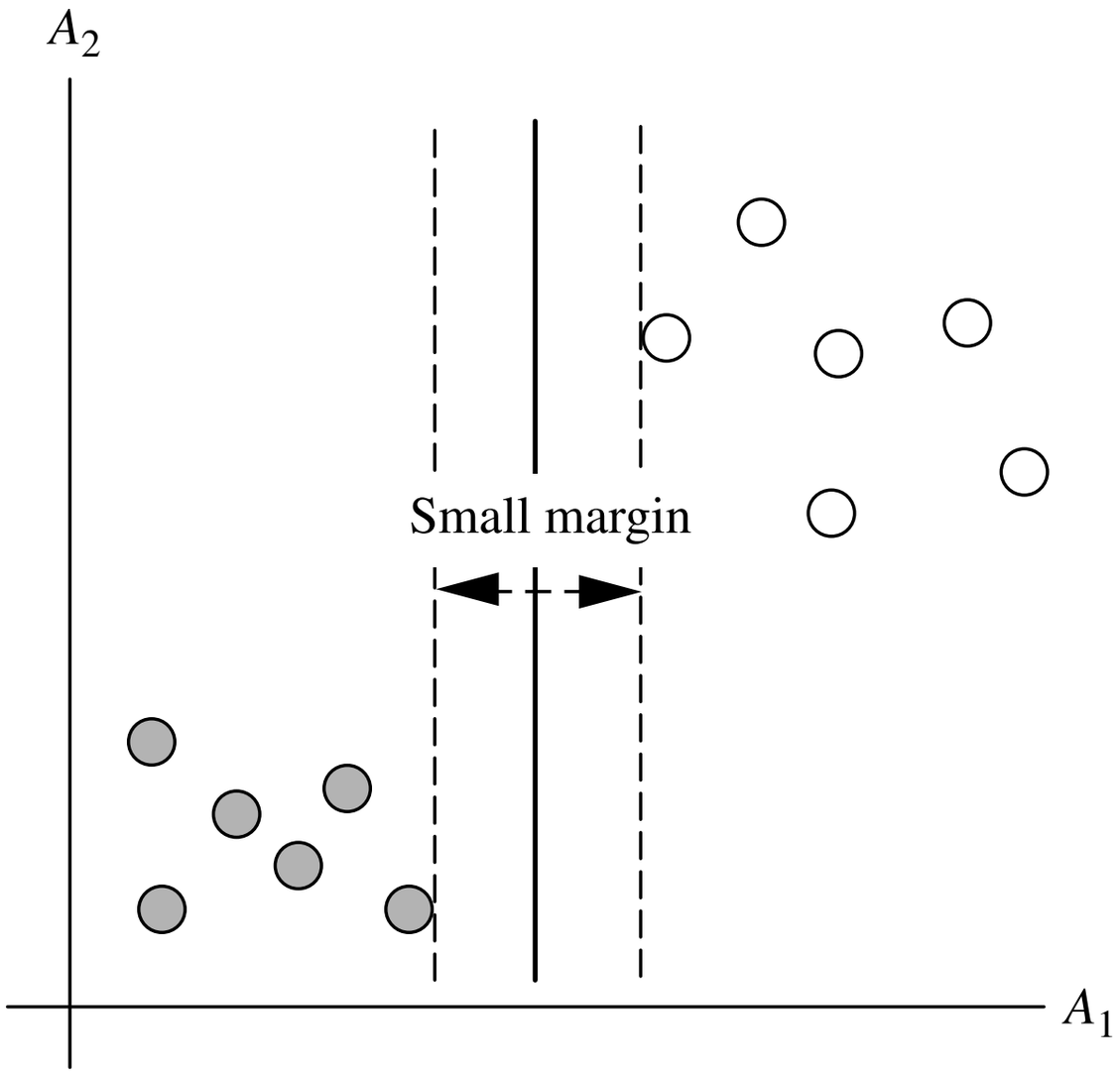
Difficult to understand the learned function (weights)

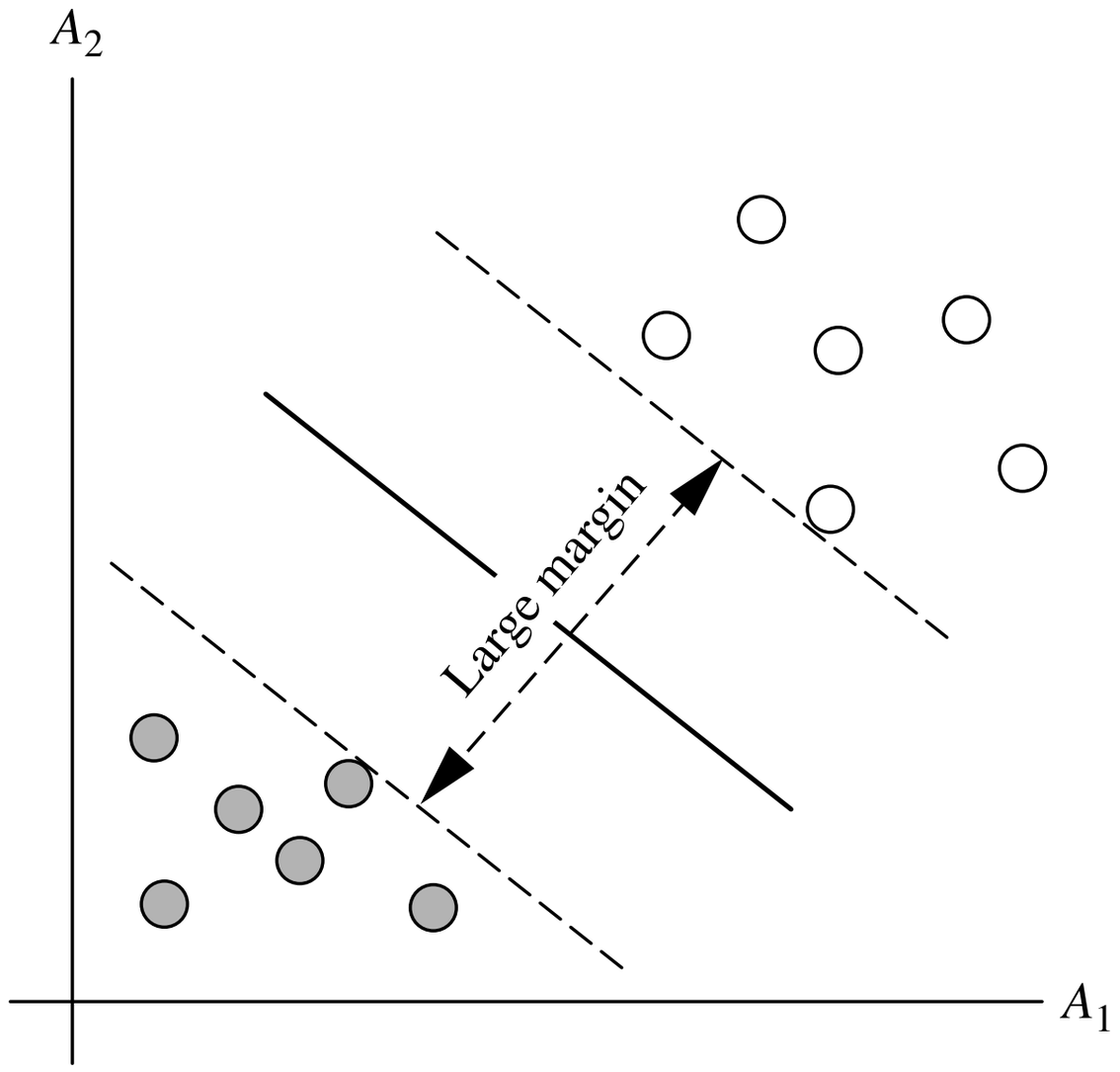
Bayesian networks can be used easily for pattern discovery

Not easy to incorporate domain knowledge

Easy in the form of priors on the data or distributions

maximum margin hyperplane





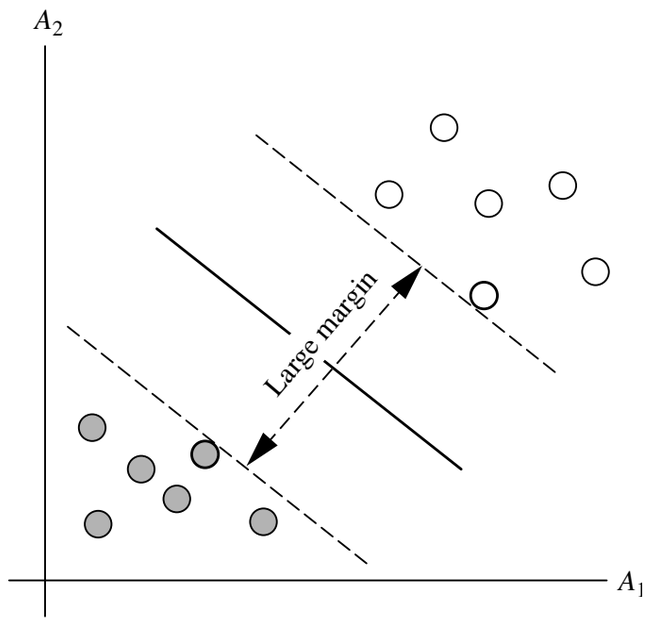
$$\boldsymbol{W} \cdot \boldsymbol{X} + b = 0$$ equation of a separating hyperplane

It uses a nonlinear mapping to transform the original training data into a higher dimension

With the new dimension, it searches for the linear optimal separating hyperplane (i.e., "decision boundary")

support vector machines

With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane

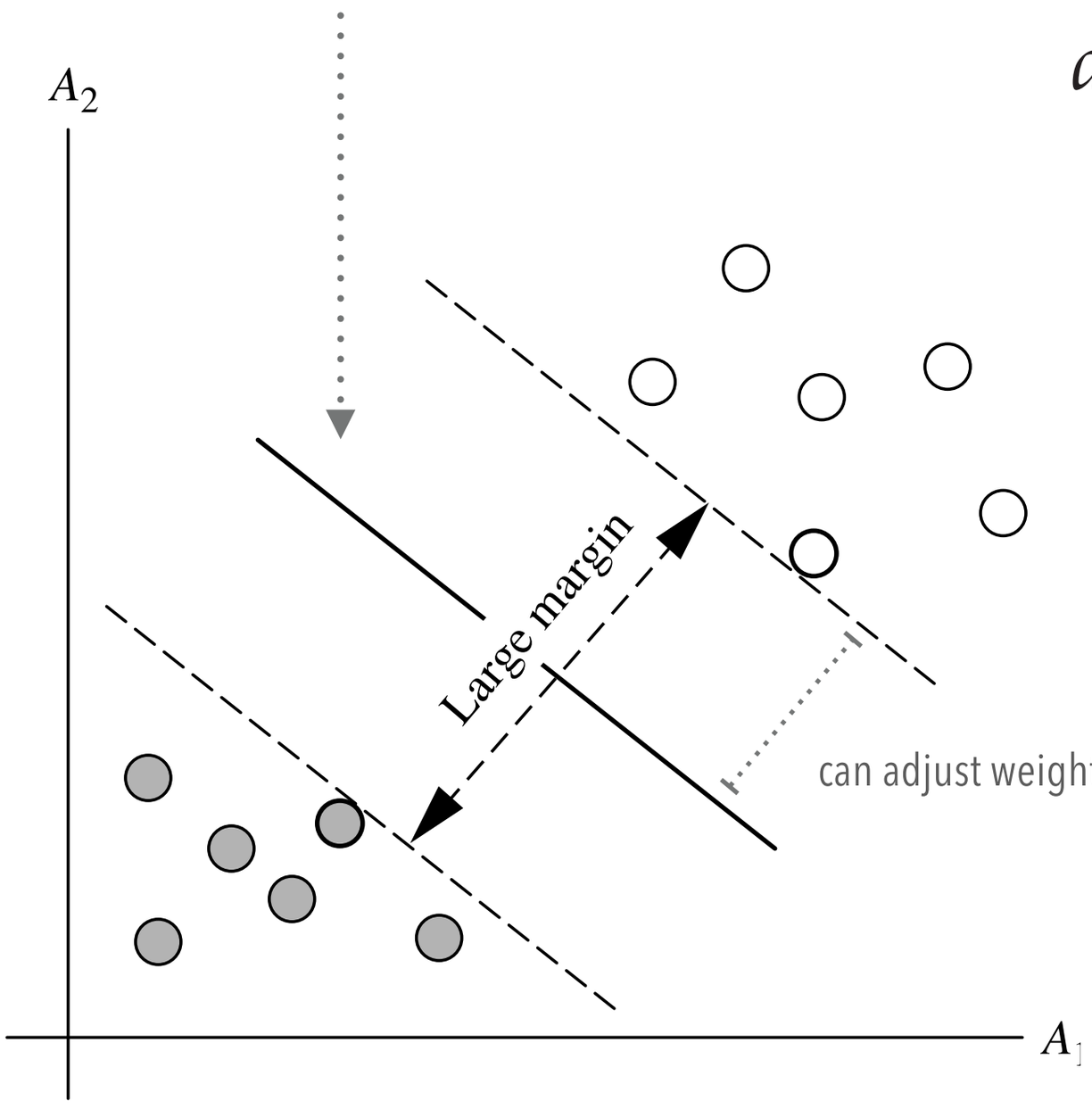


SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors)

$$w_1 x_1 + w_2 x_2 + b_0 = 0$$

decision

$$d(\boldsymbol{X}) = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{X}_i \boldsymbol{X}^T + b_0$$



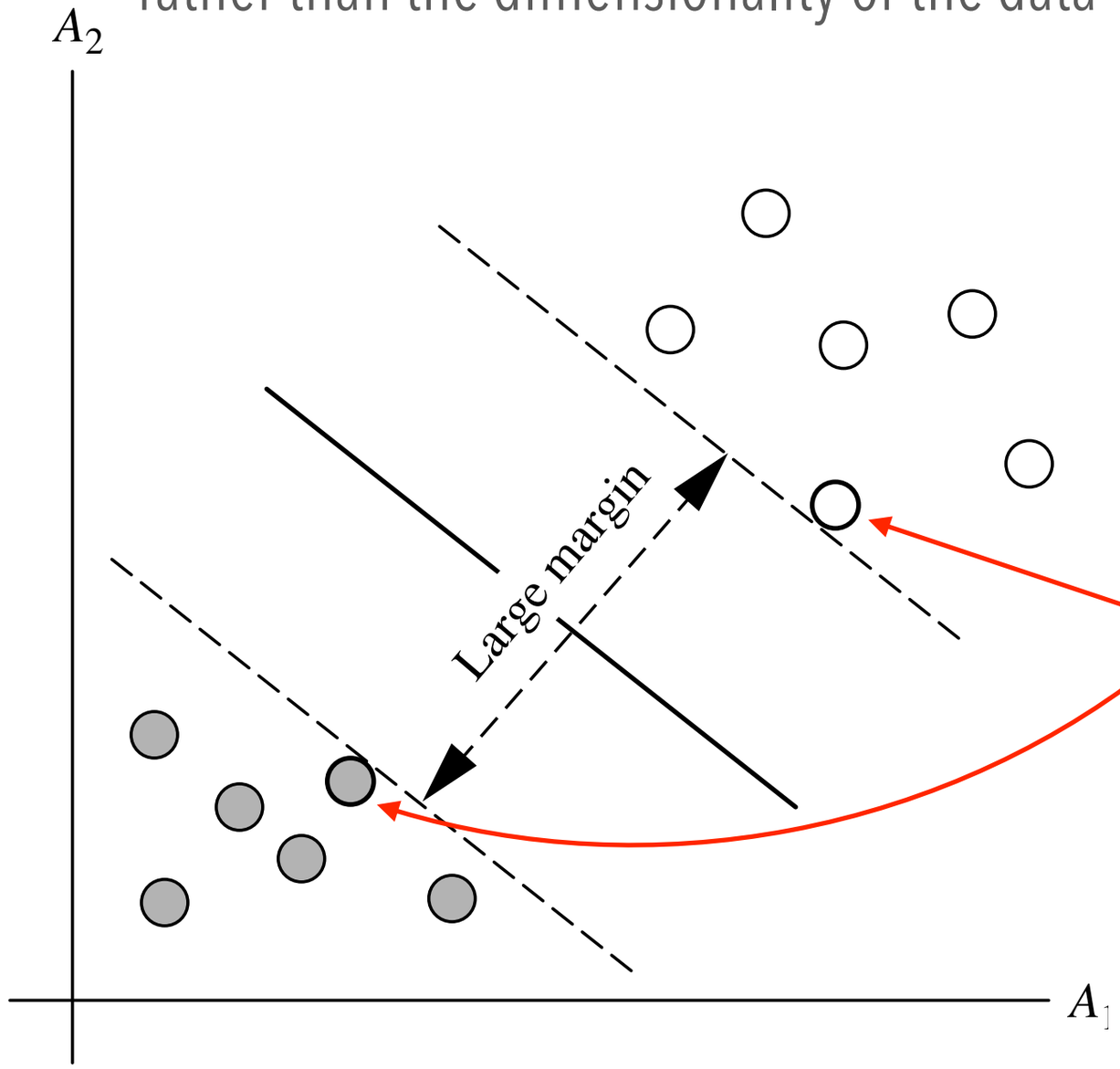
can adjust weights so that distance to the margin is 1

general case

$$H_1 : w_1 x_1 + w_2 x_2 + b_0 \geq +1, \quad y_i = +1$$
$$y_i(w_1 x_{i,1} + w_2 x_{i,2} + b_0) \geq +1 \quad \forall i$$
$$H_2 : w_1 x_1 + w_2 x_2 + b_0 \leq -1, \quad y_i = -1$$

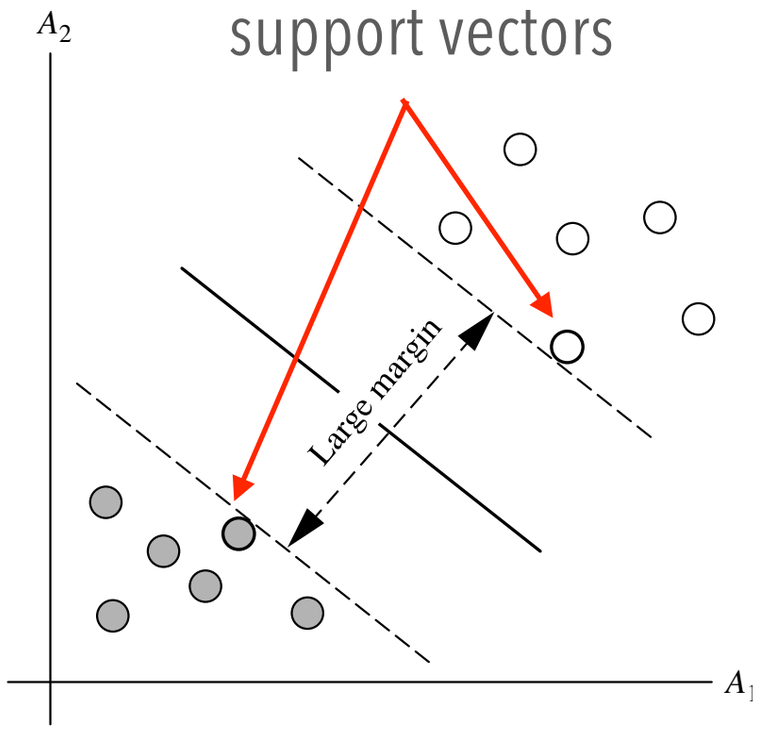
why are SVM's so effective?

The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data



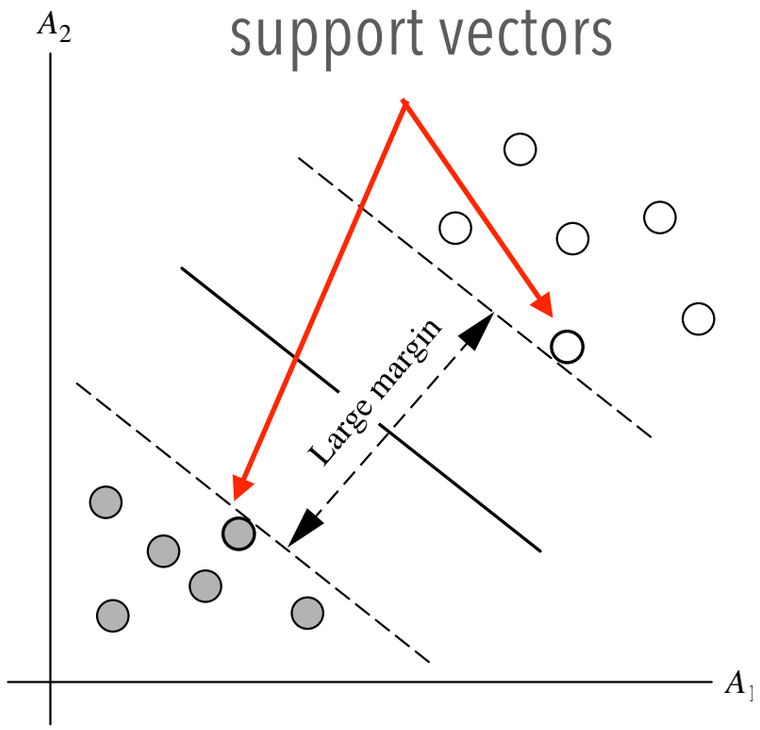
The support vectors are the essential or critical training examples–they lie closest to the decision boundary ; i.e. the maximum margin hyperplane (MMH)

what is so special about the support vectors?



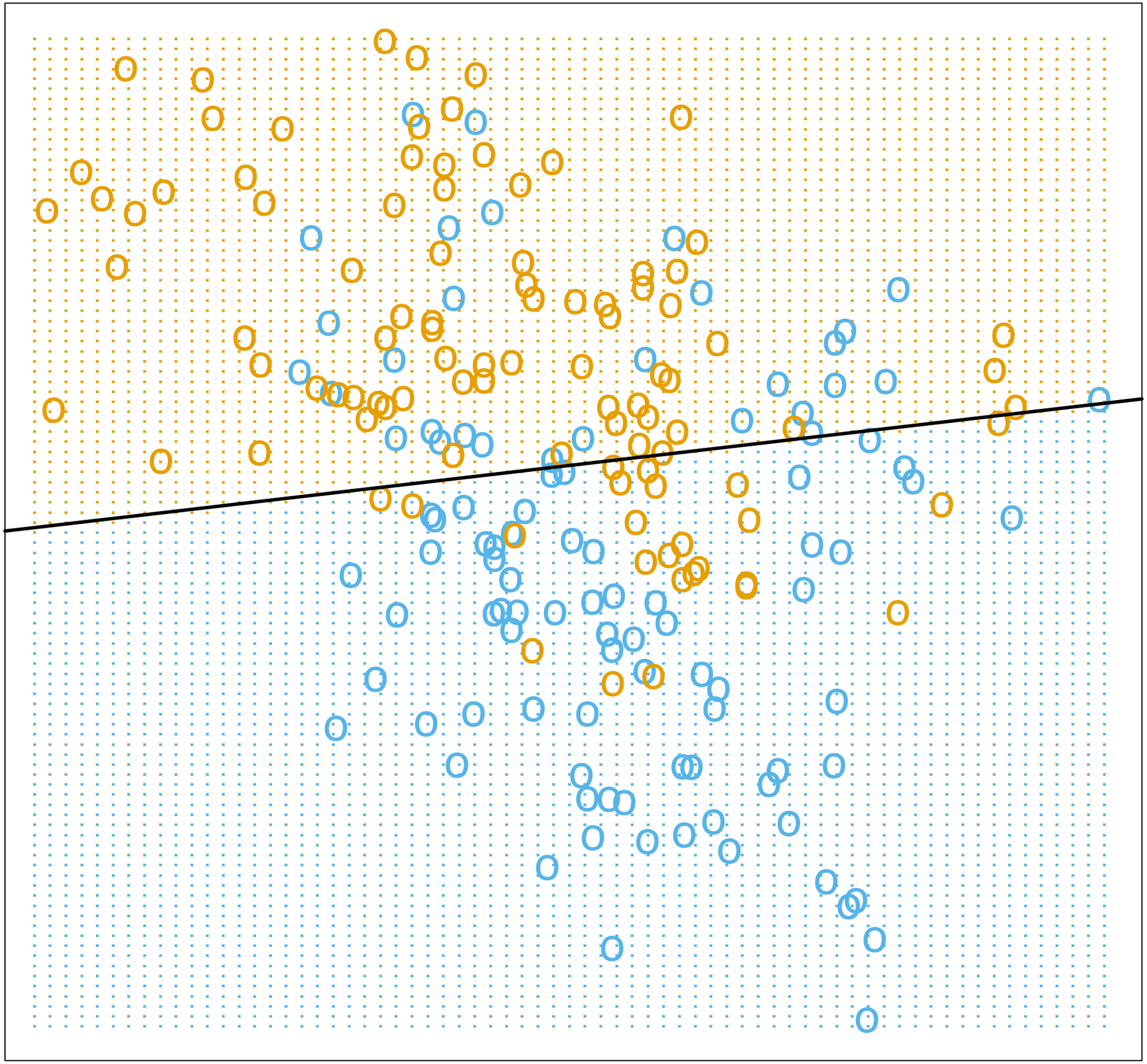
The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality

what about generalization?



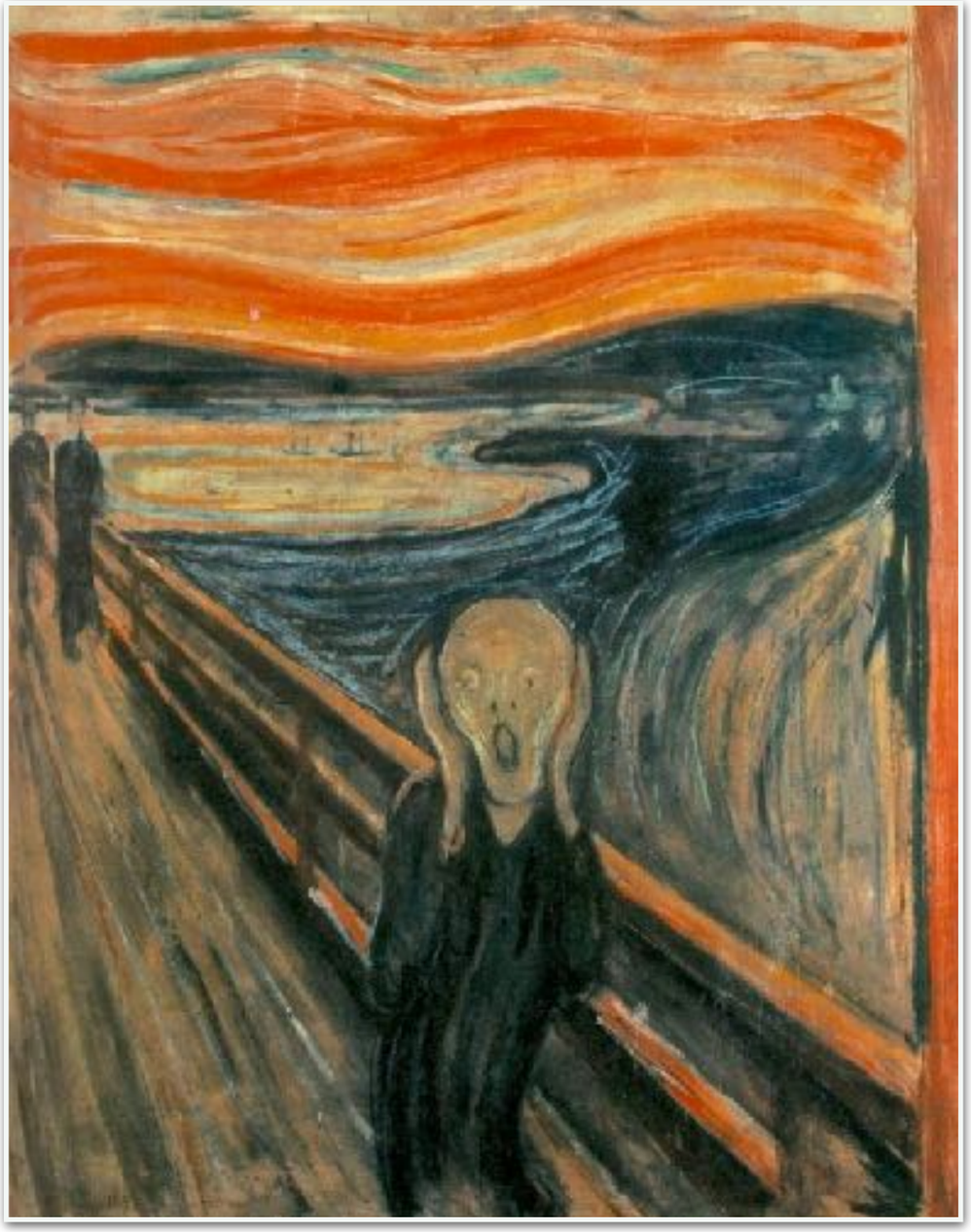
but, wait! all the cases we've discussed are linearly separable!

so why are SVM's a big deal?



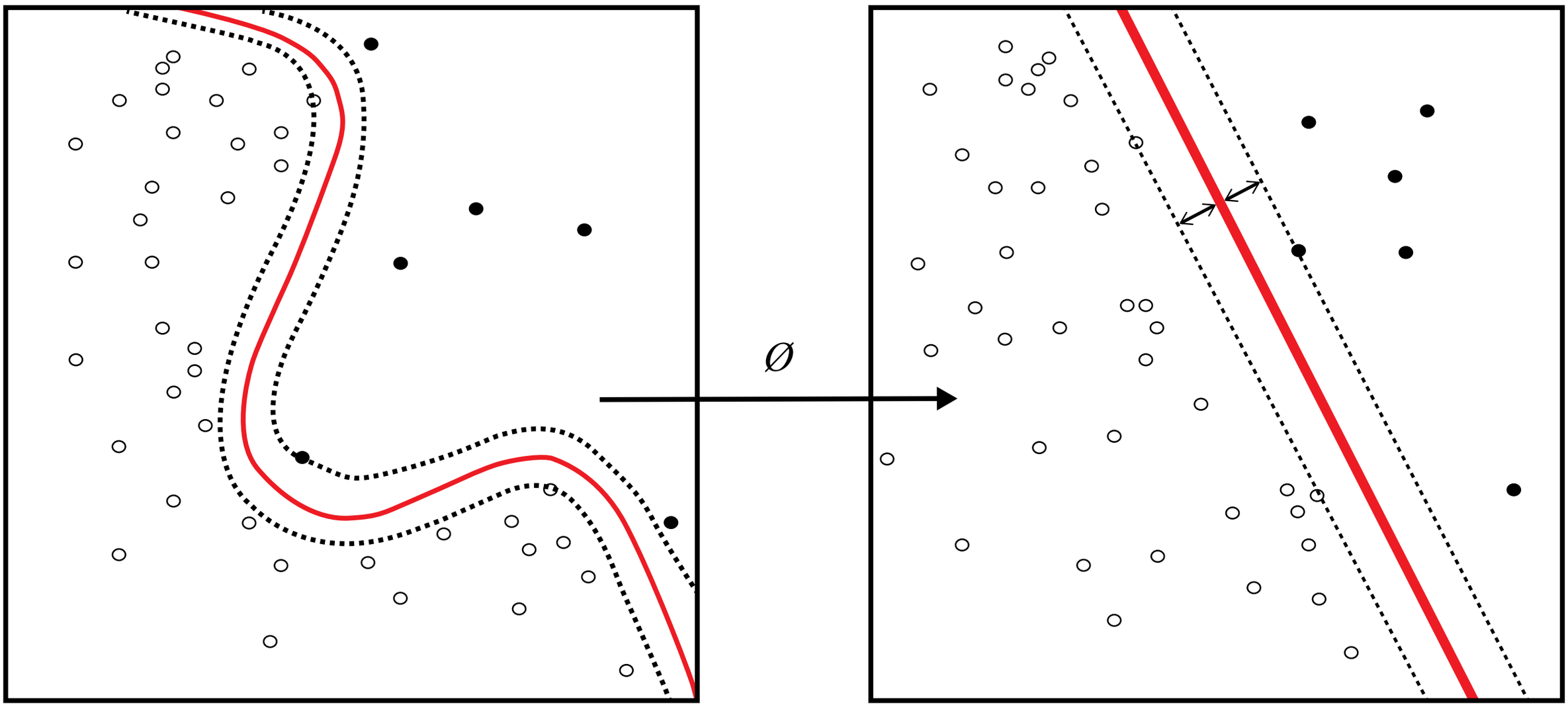
as we've discussed, for complex boundaries, we will use a non linear mapping to a high dimensional space

$$Z_i = \phi(X_i)$$



Weren't we done with kernel functions?

I stopped and leaned against the balustrade, almost dead with fatigue. Over the blue-black fjord hung clouds, red as blood–as tongues of flame. My friends passed on, and alone, trembling with anguish, I listened to the great, infinite cry of nature. **E. Munch**



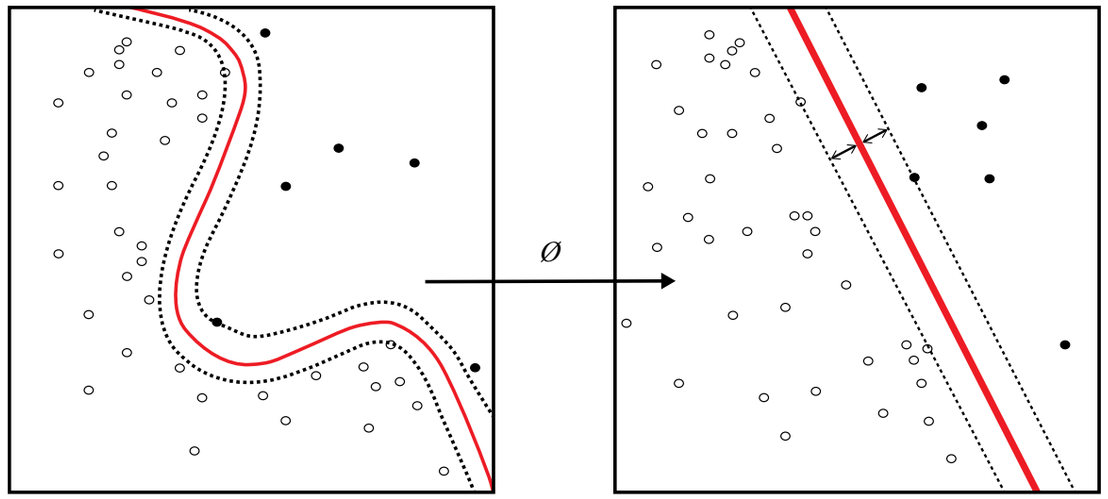
$$Z_i = \phi(X_i)$$

use linear discriminants
in the mapped space!

decision

$$d(\boldsymbol{X}) = \sum_{i=1}^{l} y_i \alpha_i \boldsymbol{X}_i \boldsymbol{X}^T + b_0$$

this only involves dot products



$$d(\boldsymbol{X}) = \sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{X}_i) \cdot \phi(\boldsymbol{X}^T) + b_0$$

still dot products, but
this is a little more tricky

enter the kernel trick

what if:

$$K(X_i, X) = \phi(X_i) \cdot \phi(X)$$

K is a function

$$K(X_i, X) = \phi(X_i) \cdot \phi(X)$$

there exists some function K, which is equivalent to the dot product in some transformed space

now not only we don't have to find ϕ, but we <u>don't</u> have to compute the dot product!

$$K(X_i, X) = \phi(X_i) \cdot \phi(X)$$

this sounds too good to be true!

is there a catch?

dimensions to which we map

if $d \geq N$, we are guaranteed 100% accuracy

how do we find these kernels?

$$K(X_i, X) = \phi(X_i) \cdot \phi(X)$$

there exists some function K, which is equivalent to the dot product in some transformed space

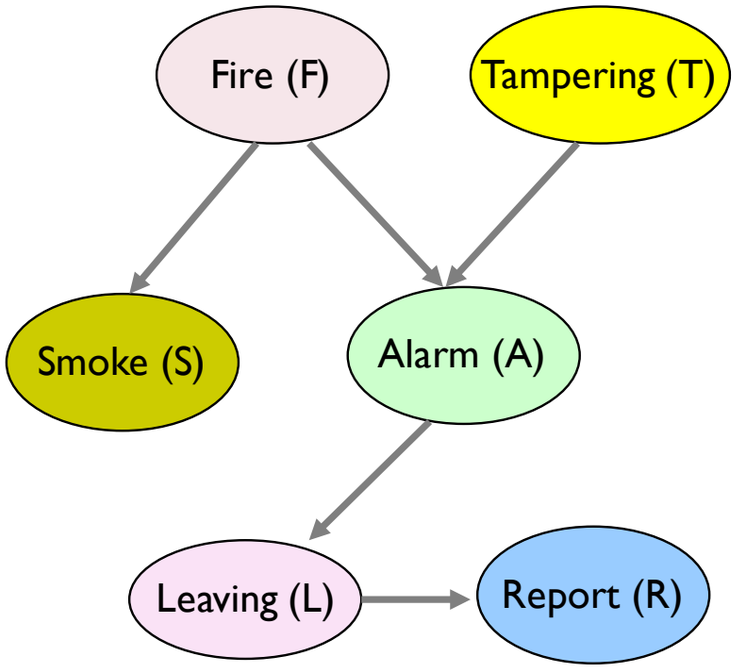
polynomial

$$K(\boldsymbol{X}_i, \boldsymbol{X}) = (\boldsymbol{X}_i \cdot \boldsymbol{X} + 1)^h$$

$$K(\boldsymbol{X}_i, \boldsymbol{X}) = \exp\left(\frac{-||\boldsymbol{X}_i - \boldsymbol{X}||^2}{2\sigma^2}\right)$$

Gaussian

what about multi-class classification?

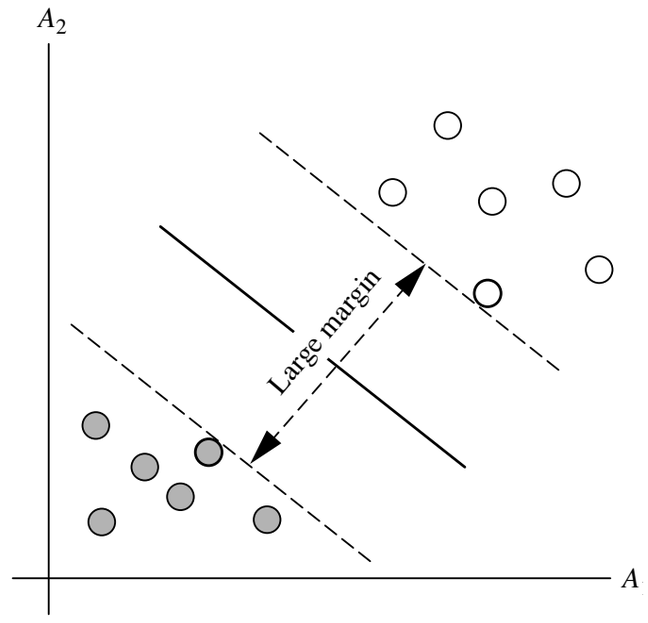


Bayesian belief networks allow class conditional independencies between subsets of variables

DAG's

Markov

SUMMARY

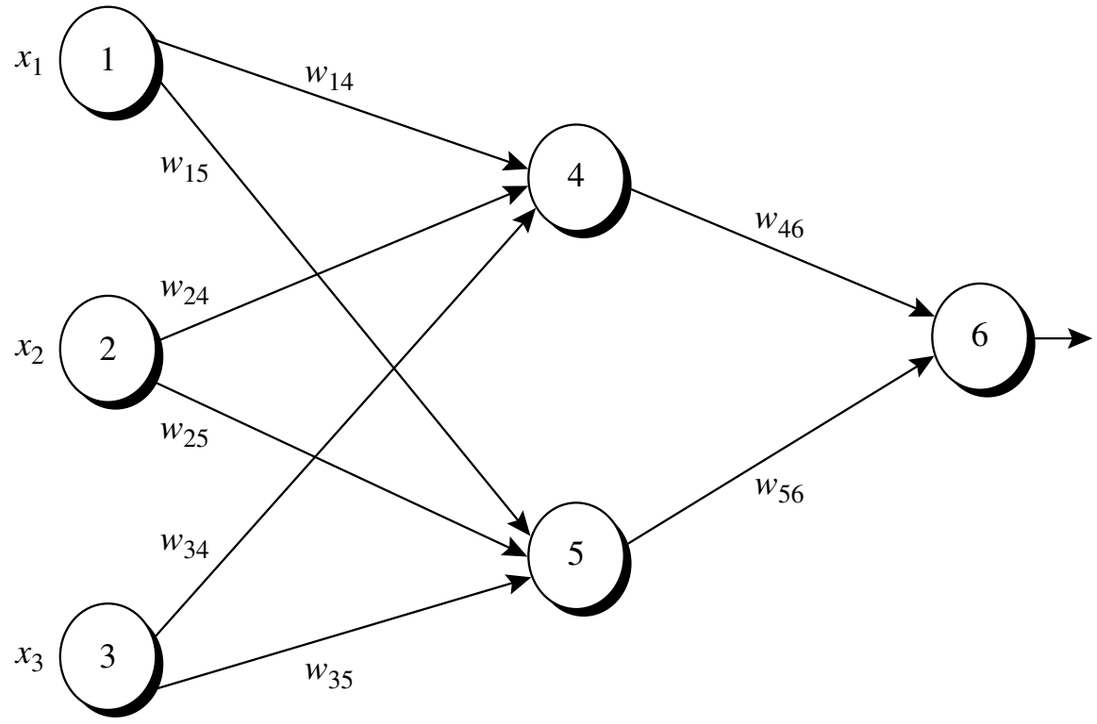


linear discriminants

max margin

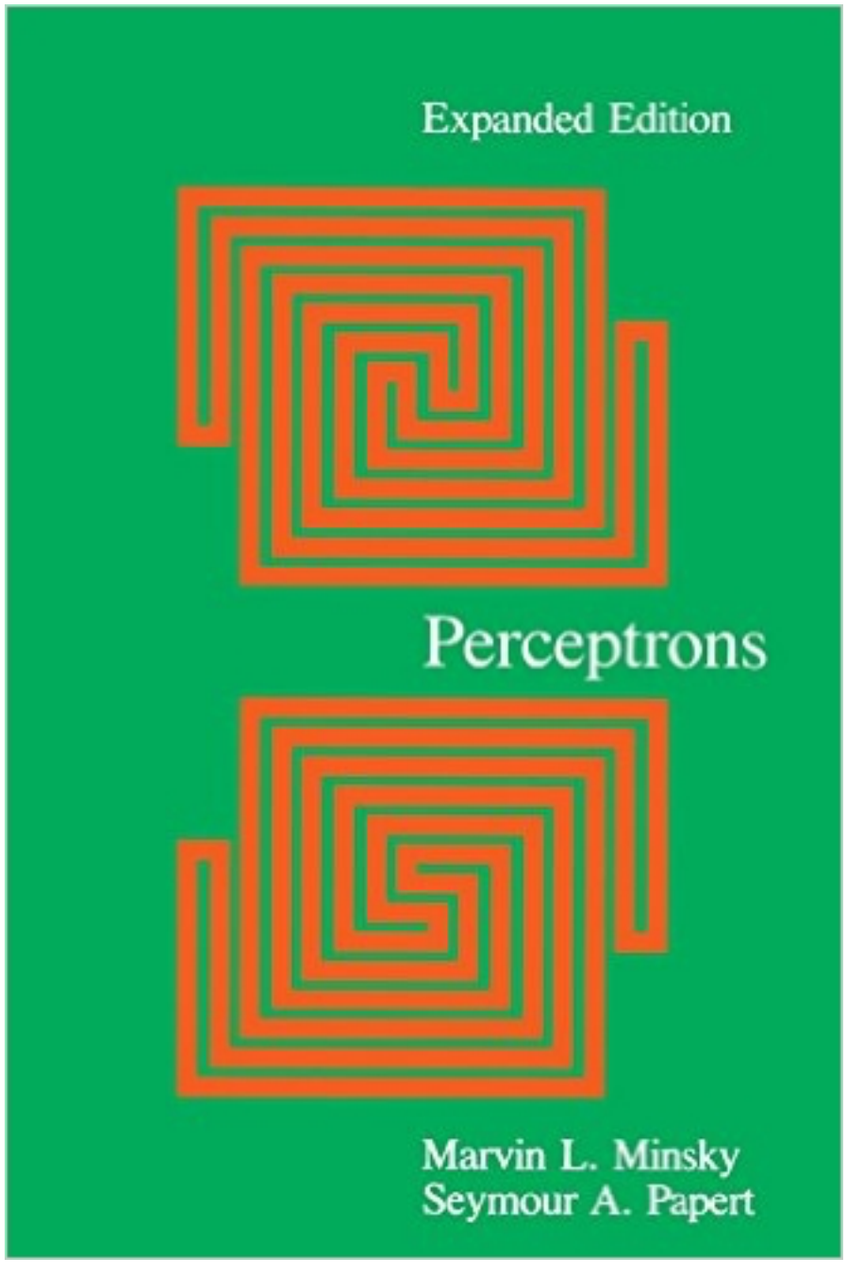
kernel trick

NEURAL NETWORKS



Bayesian Networks Support Vector Machines Etc

Summary

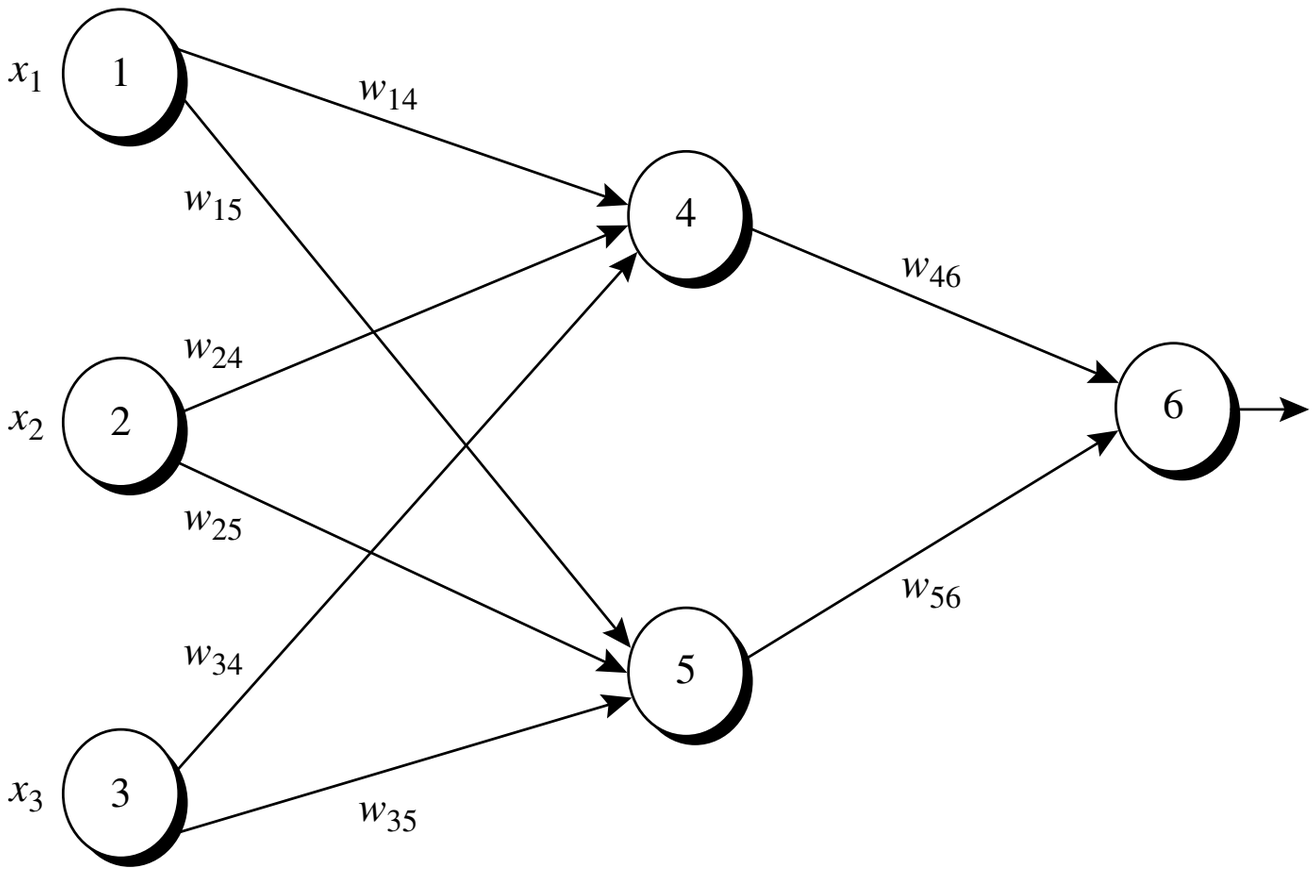


cannot represent XOR gates

$$f(\boldsymbol{x}) = b + \sum_i \alpha_i x_i$$

AI winter lasted till mid 80's

BACKPROPAGATION



Backpropagation: A neural network learning algorithm

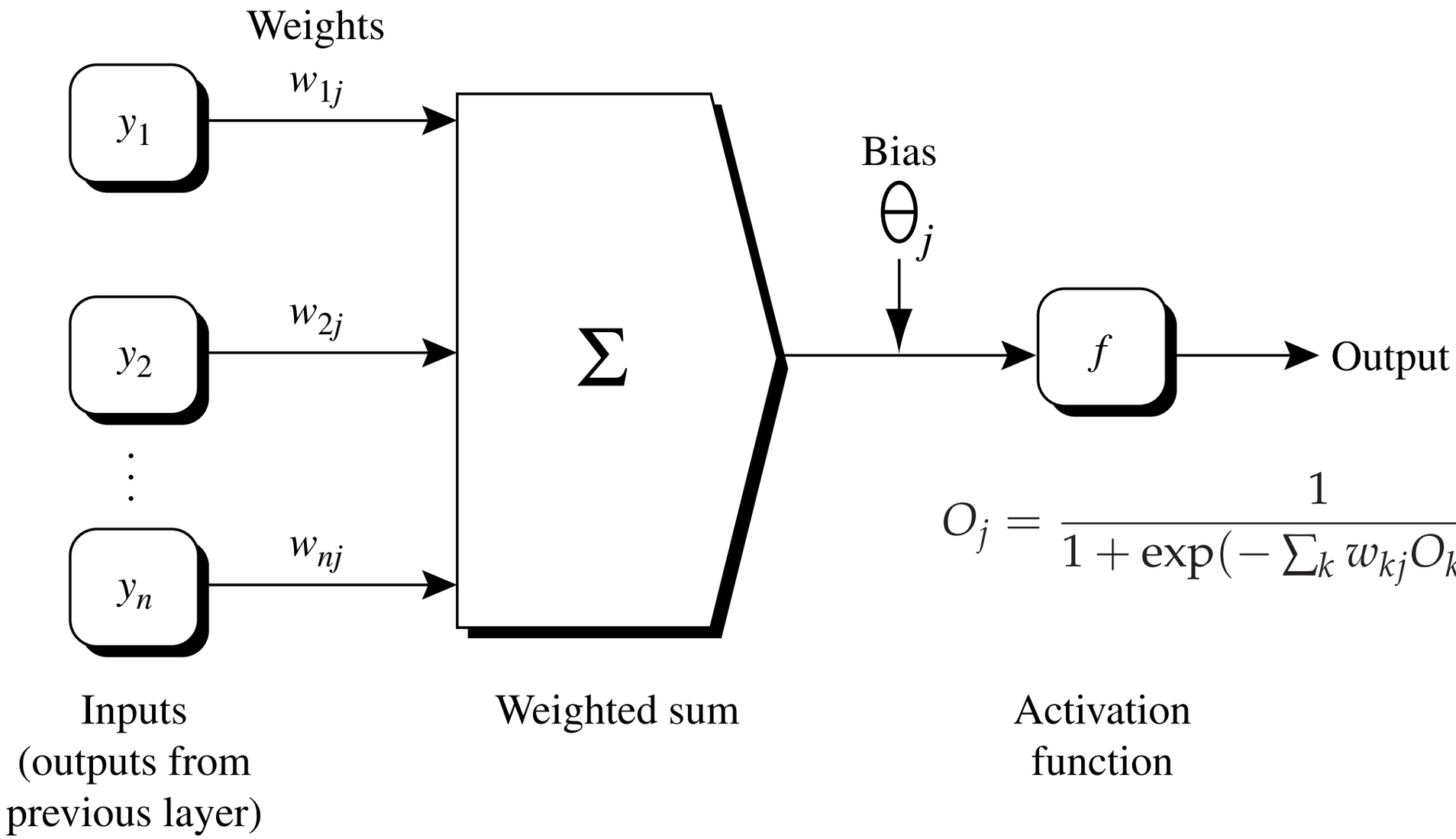
Started by psychologists and neurobiologists to develop and test computational analogues of neurons

A neural network: A set of connected input/output units where each connection has a weight associated with it

- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples

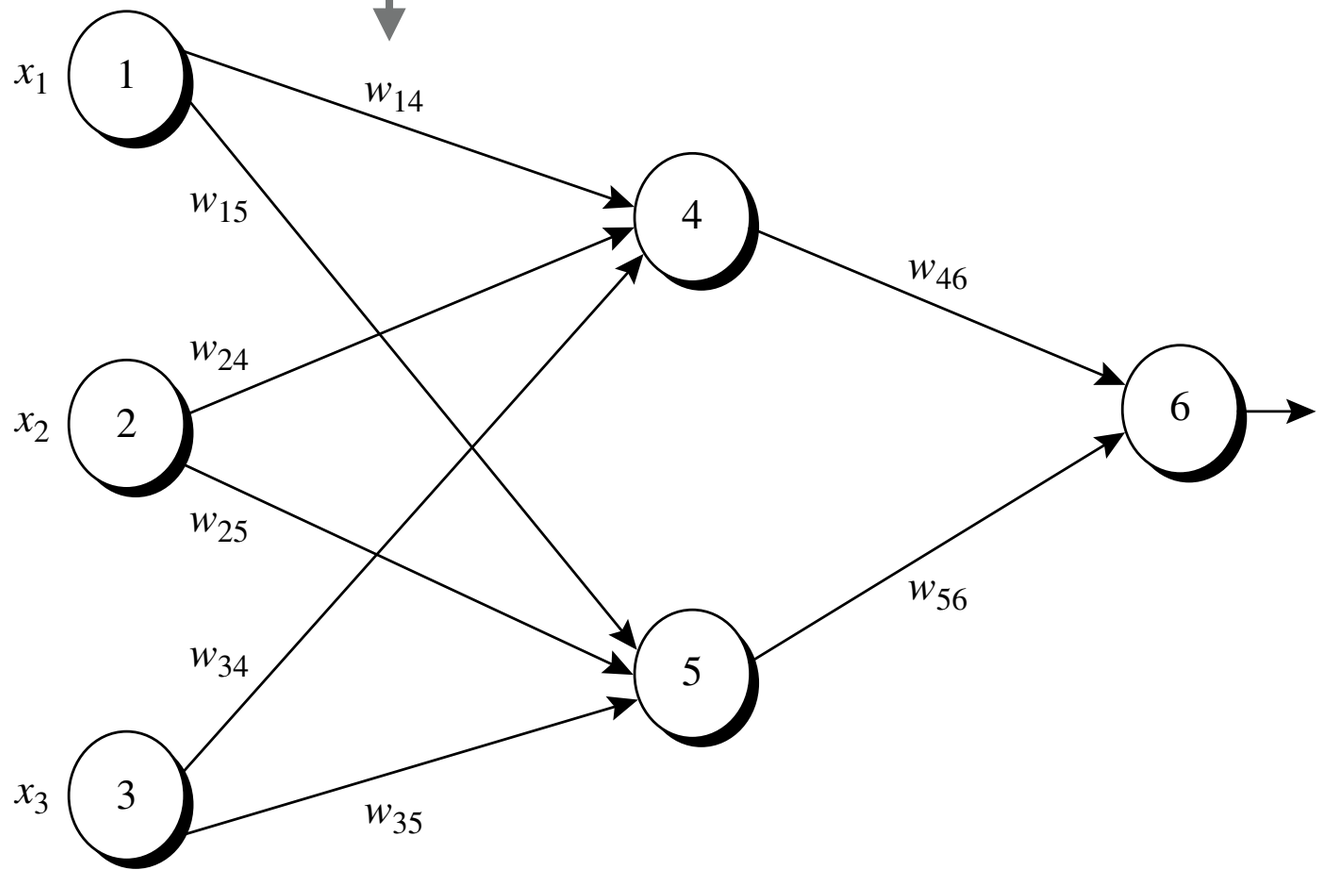
Also referred to as connectionist learning due to the connections between units

An n-dimensional input vector x is mapped into variable y by means of the scalar product and a nonlinear function mapping



The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

They are then weighted and fed simultaneously to a hidden layer



The inputs to the network correspond to the attributes measured for each training tuple

The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction

The network is feed-forward

Inputs are fed simultaneously into the units making up the input layer

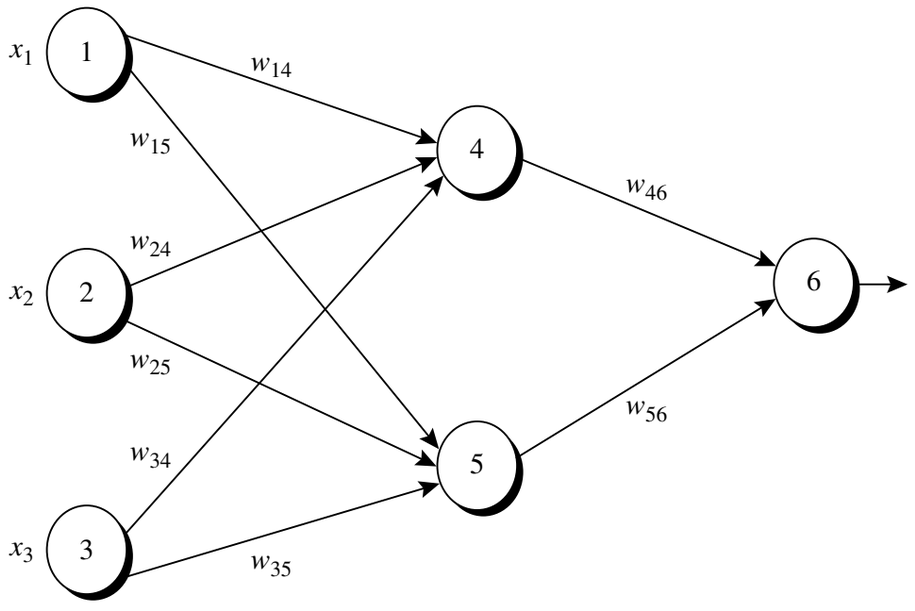
how do we decide on the network topology?

Iteratively process a set of training tuples & compare the network's prediction with the actual known target value

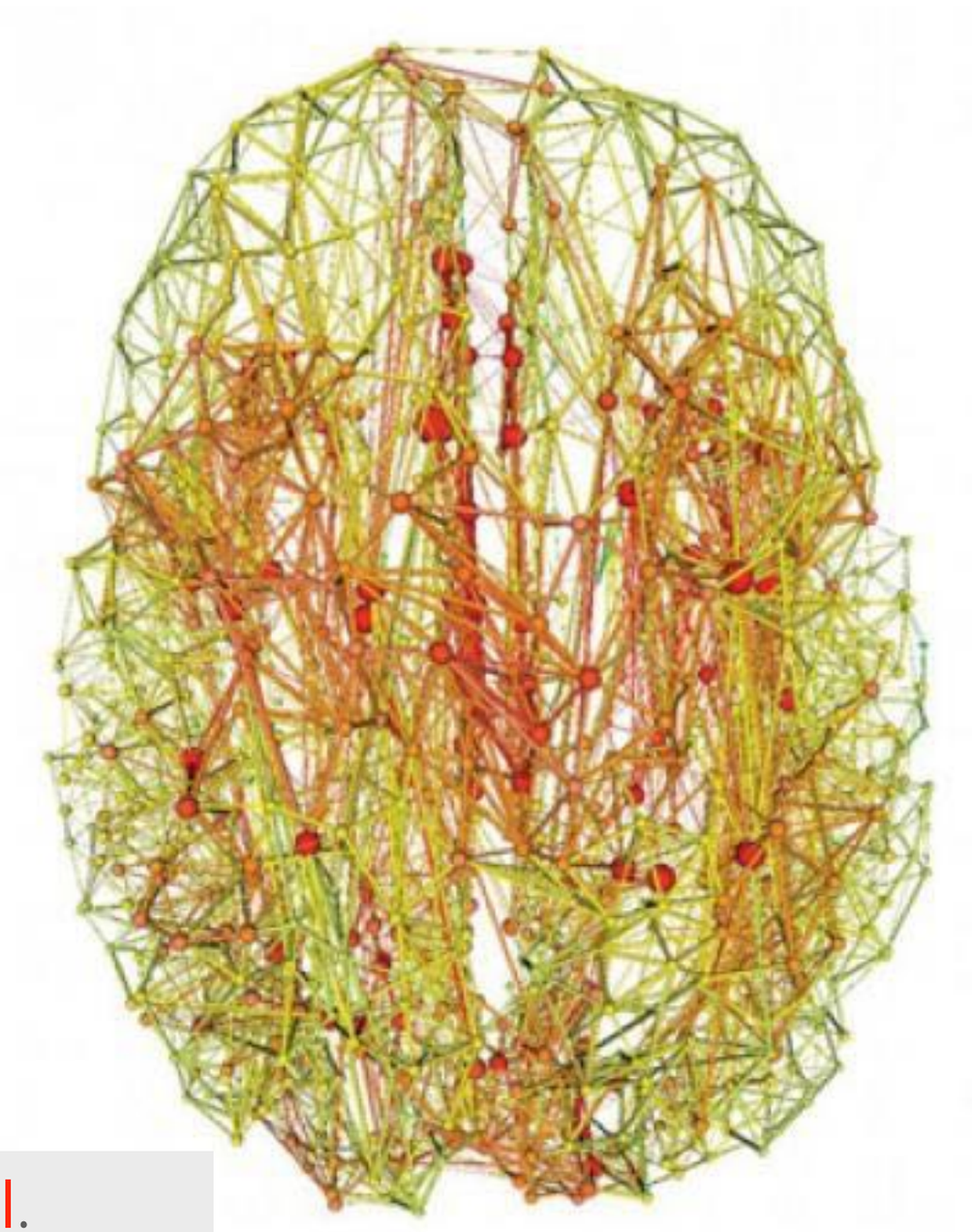
For each training tuple, the weights are modified to minimize the mean squared error between the network's prediction and the actual target value

backpropagation

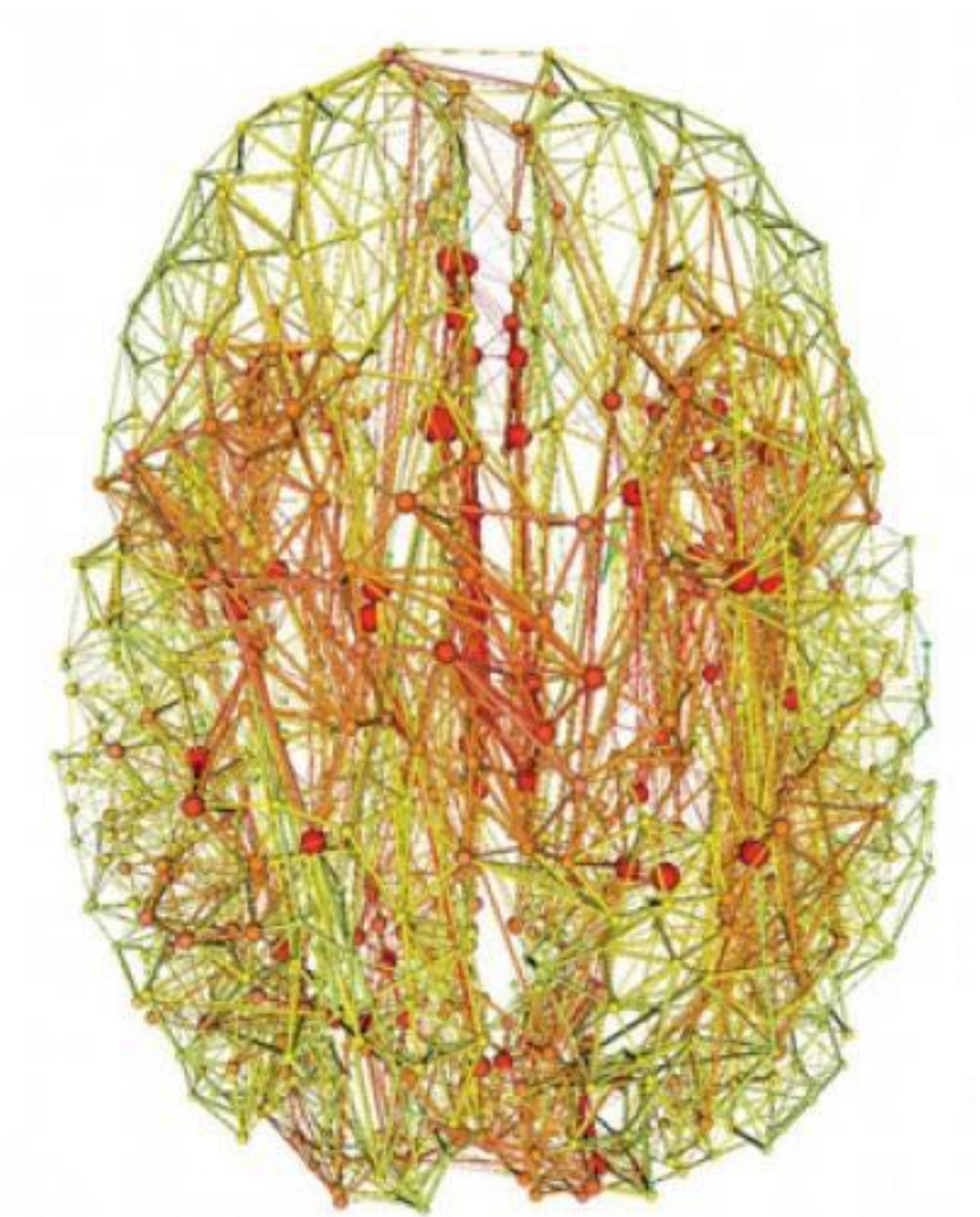
Modifications are made in the "backwards" direction: from the output layer, through each hidden layer down to the first hidden layer,



deep networks



Bengio, Y. (2009). Learning deep architectures for AI. Foundations and trends® in Machine Learning, 2(1), 1-127.

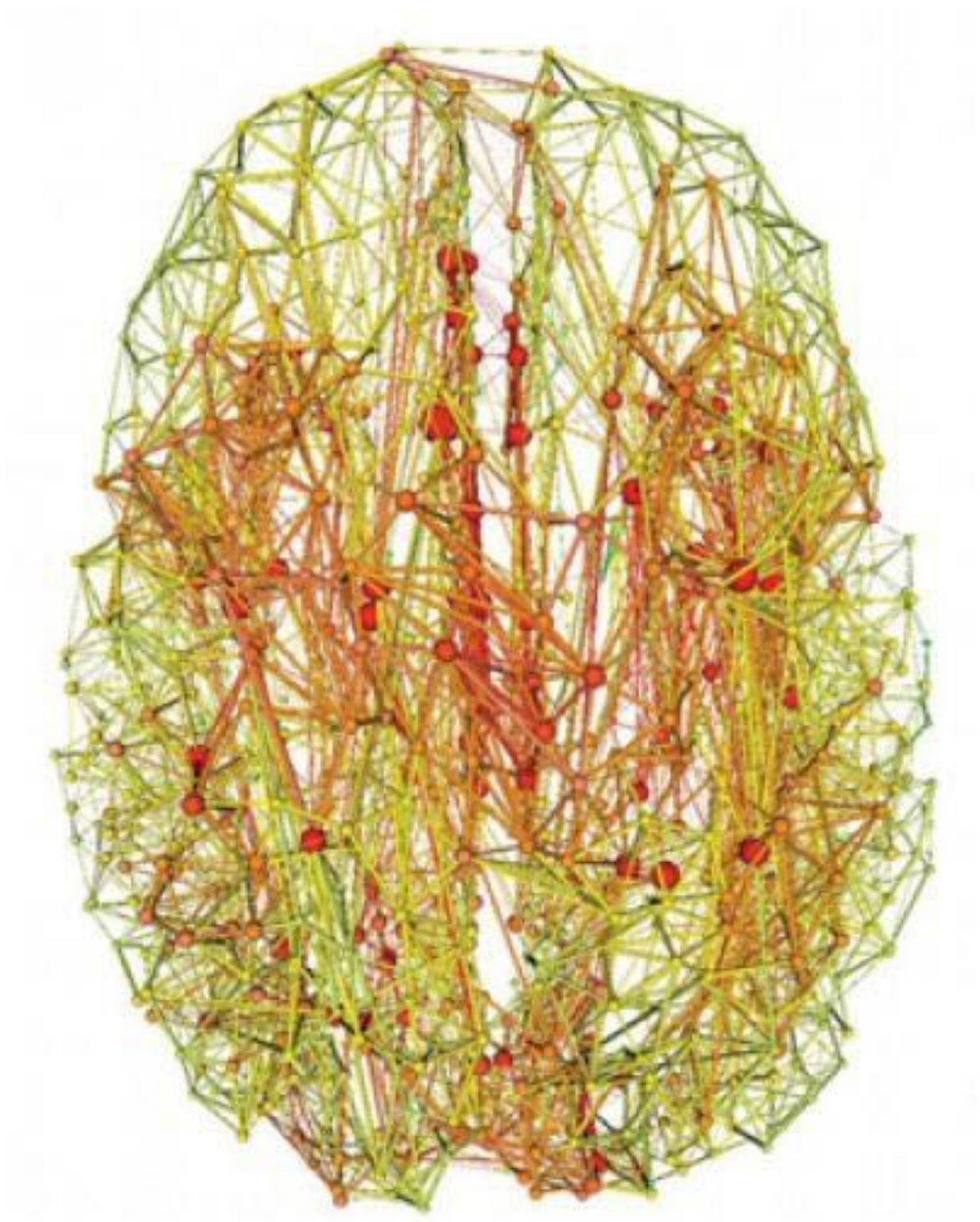


SOME CONSIDERATIONS

Ability to learn complex, highly-varying functions, i.e., with a number of variations much greater than the number of training examples.

Ability to learn with little human input the low-level, intermediate, and high-level abstractions that would be useful to represent the kind of complex functions needed for AI tasks.

Ability to learn from a very large set of examples: computation time for training should scale well with the number of examples, i.e., close to linearly.



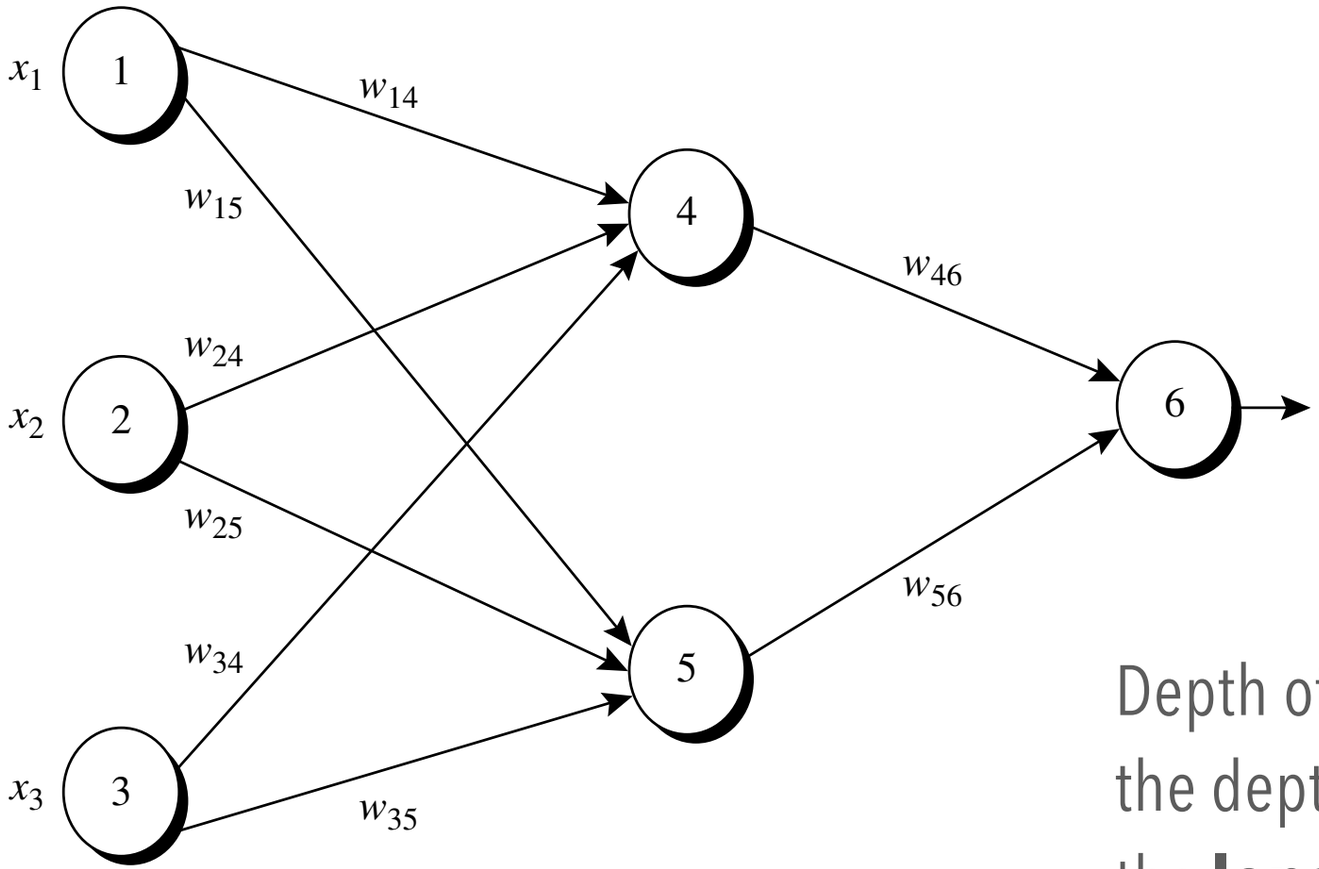
SOME CONSIDERATIONS

Ability to learn from mostly unlabeled data, i.e., to work in the semi-supervised setting, where not all the examples come with complete and correct semantic labels.

Ability to exploit the synergies present across a large number of tasks, i.e., multi-task learning. These synergies exist because all the AI tasks provide different views on the same underlying reality.

Strong unsupervised learning (i.e., capturing most of the statistical structure in the observed data), which seems essential in the limit of a large number of tasks and when future tasks are not known ahead of time.

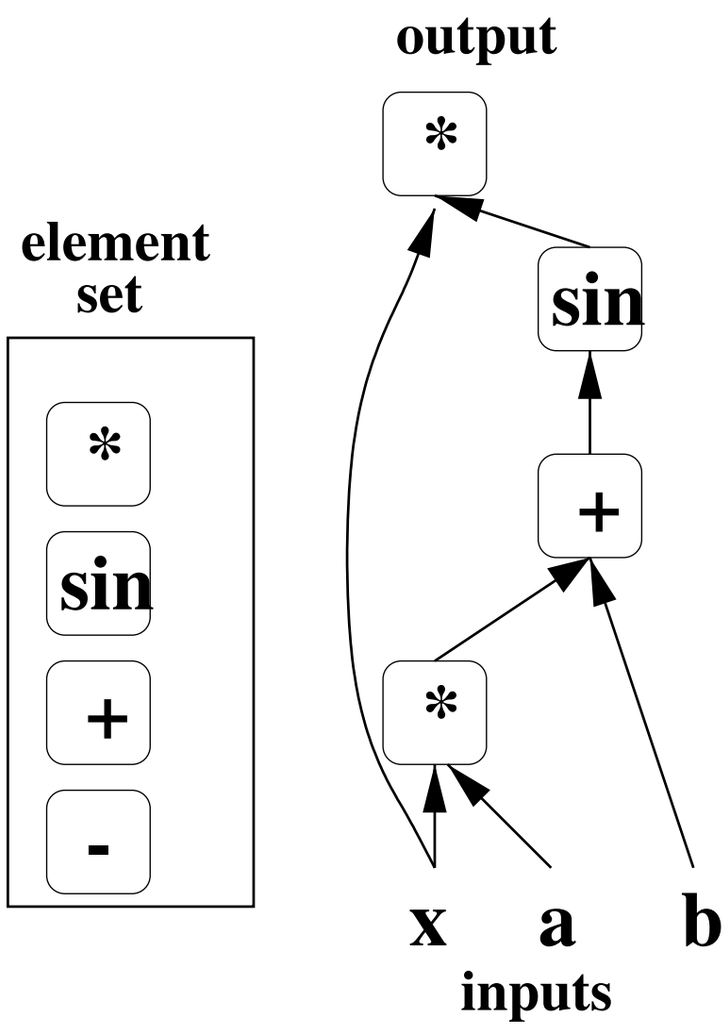
what is shallow?

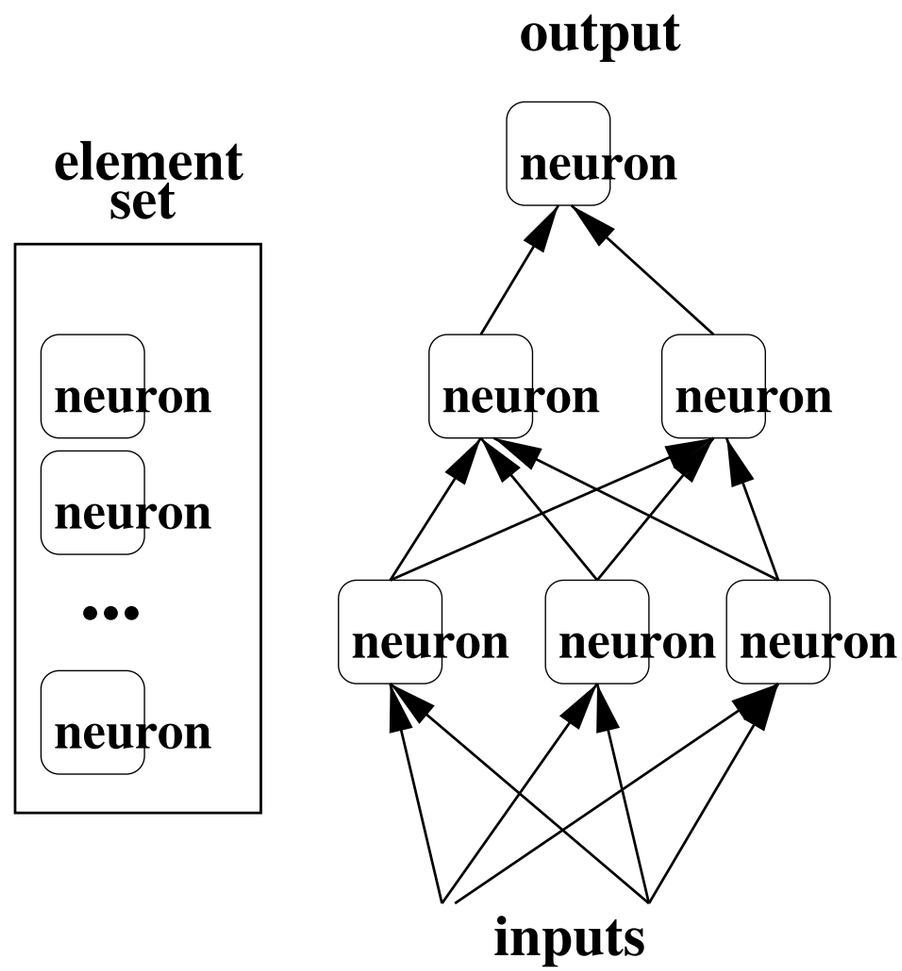


Depth of architecture refers to the depth of that graph, i.e., the **longest path** from an input node to an output node

the expression of a function is compact when it has few computational elements, i.e., few degrees of freedom that need to be tuned by learning.

$$f(x) = x \sin(ax + b)$$





DEPTH

If we include affine operations and their possible composition with sigmoids in the set of computational elements, linear regression and logistic regression have depth 1, i.e., have a single level.

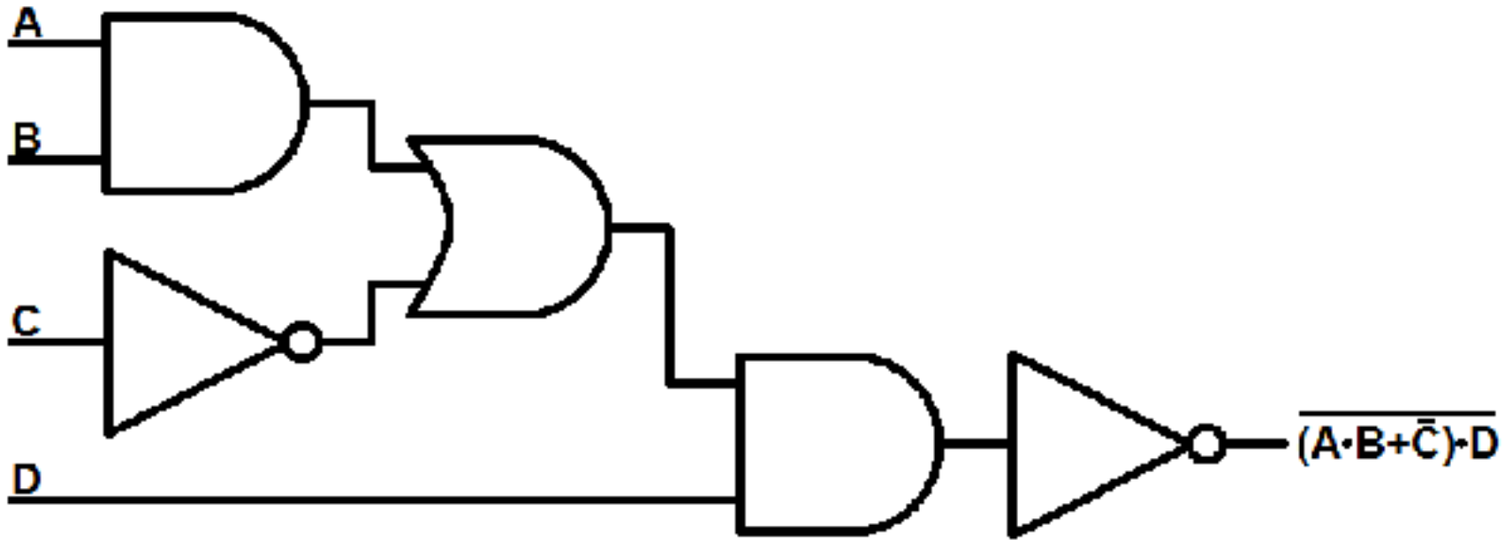
When we put a fixed kernel computation K(u,v) in the set of allowed operations, along with affine operations, kernel machines with a fixed kernel can be considered to have two levels. The first level has one element computing $K(x,x_i)$ for each prototype x_i (a selected representative training example) and matches the input vector x with the prototypes x_i. The second level performs an affine combination.

Boosting usually adds one level to its base learners: that level computes a vote or linear combination of the outputs of the base learners.

Based on current knowledge of brain anatomy, it appears that the cortex can be seen as a deep architecture, with 5–10 levels just for the visual system.

More precisely, functions that can be compactly represented by a depth k architecture might require an exponential number of computational elements to be represented by a depth k−1 architecture.

when a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one.

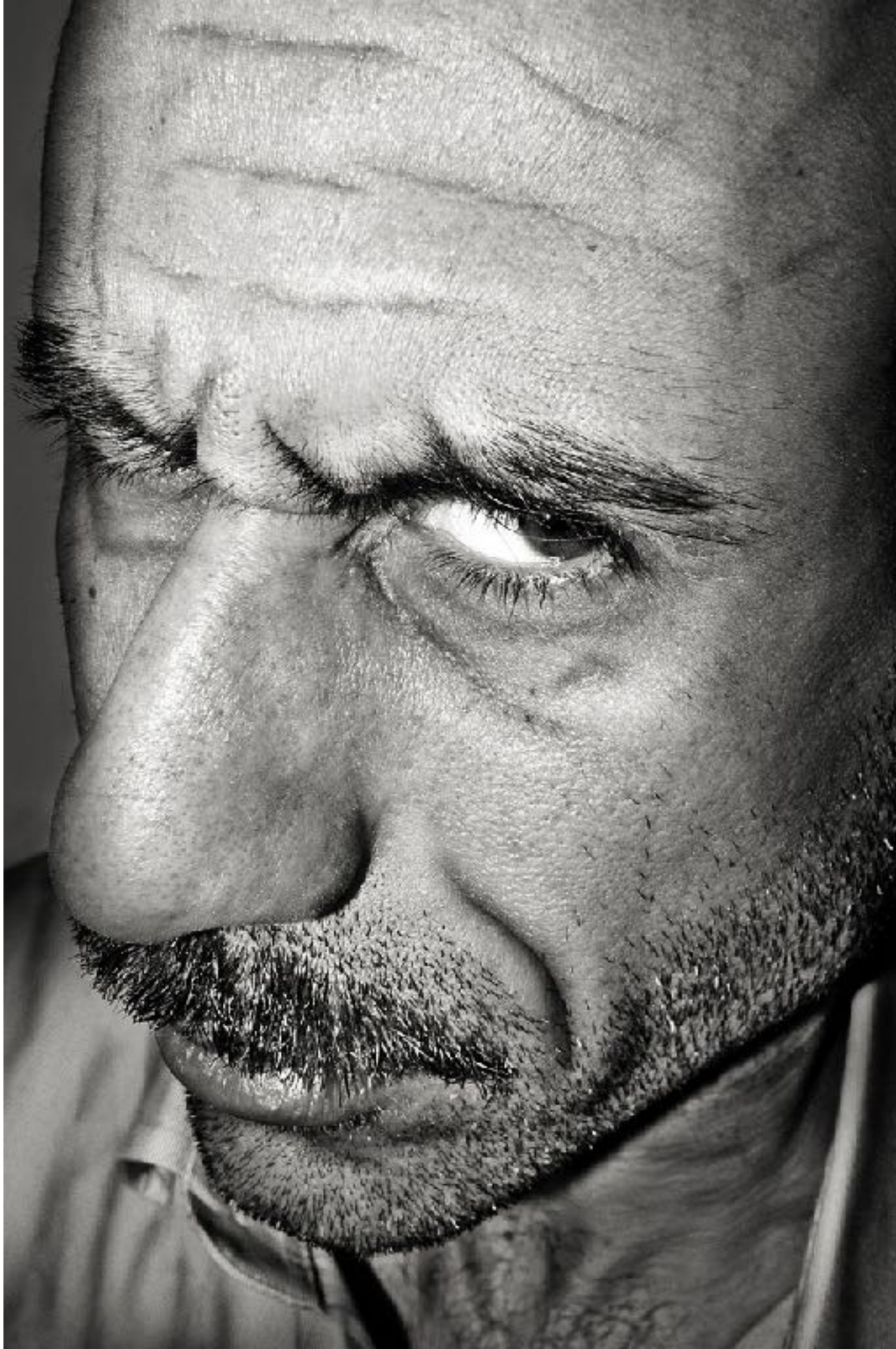


consider a function described by logic gates

there are functions computable with a polynomial-size logic gates circuit of depth k that require exponential size when restricted to depth $k-1$

does the result generalize?

there are strong indications that deep networks are more "efficient"



An estimator that is local in input space obtains good generalization for a new input x by mostly exploiting training examples in the neighborhood of x.

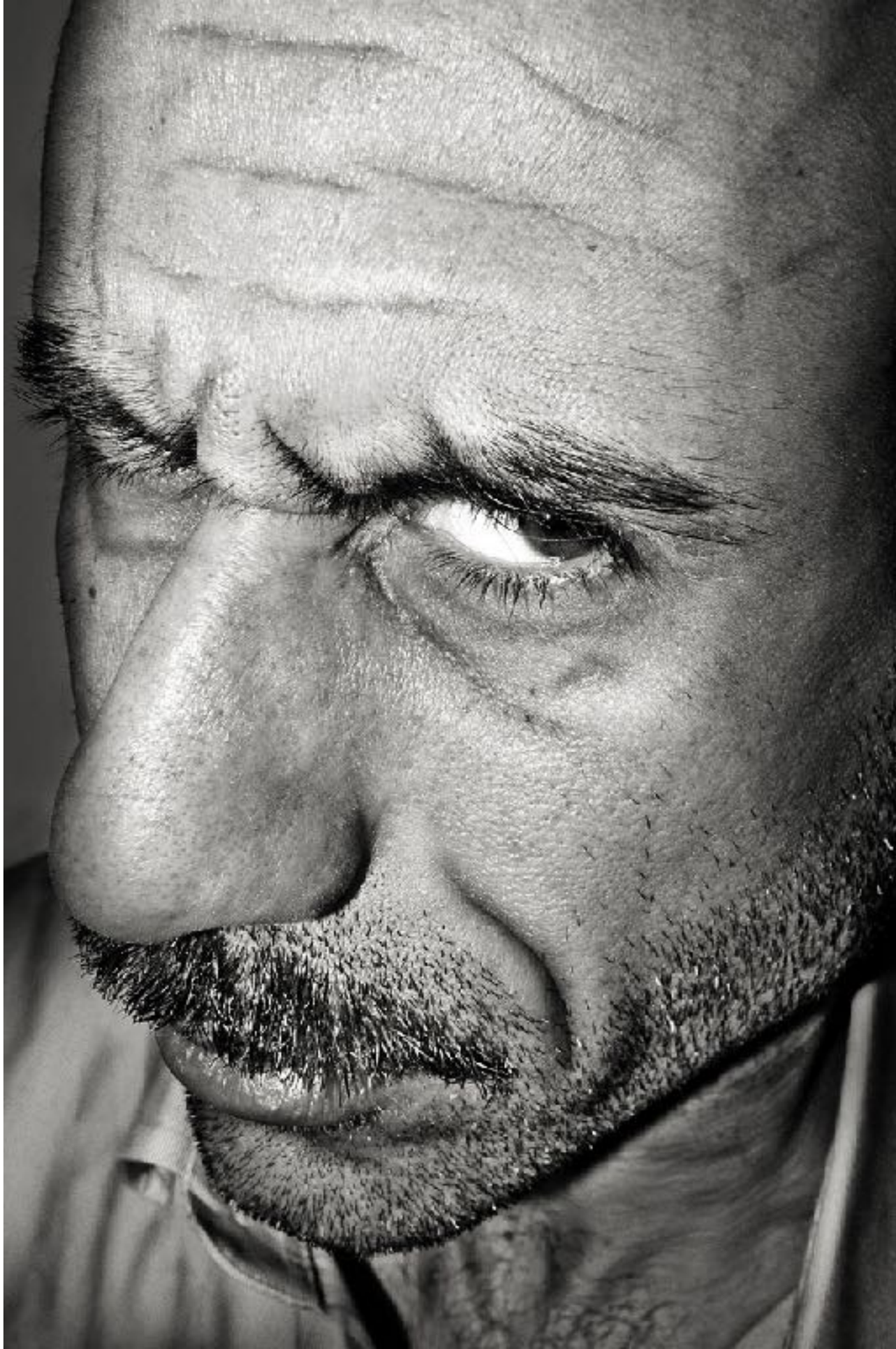
$$f(\boldsymbol{x}) = b + \sum_i \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i)$$

smoothness prior

such a prior is often insufficient to generalize when the target function is highly varying in input space.

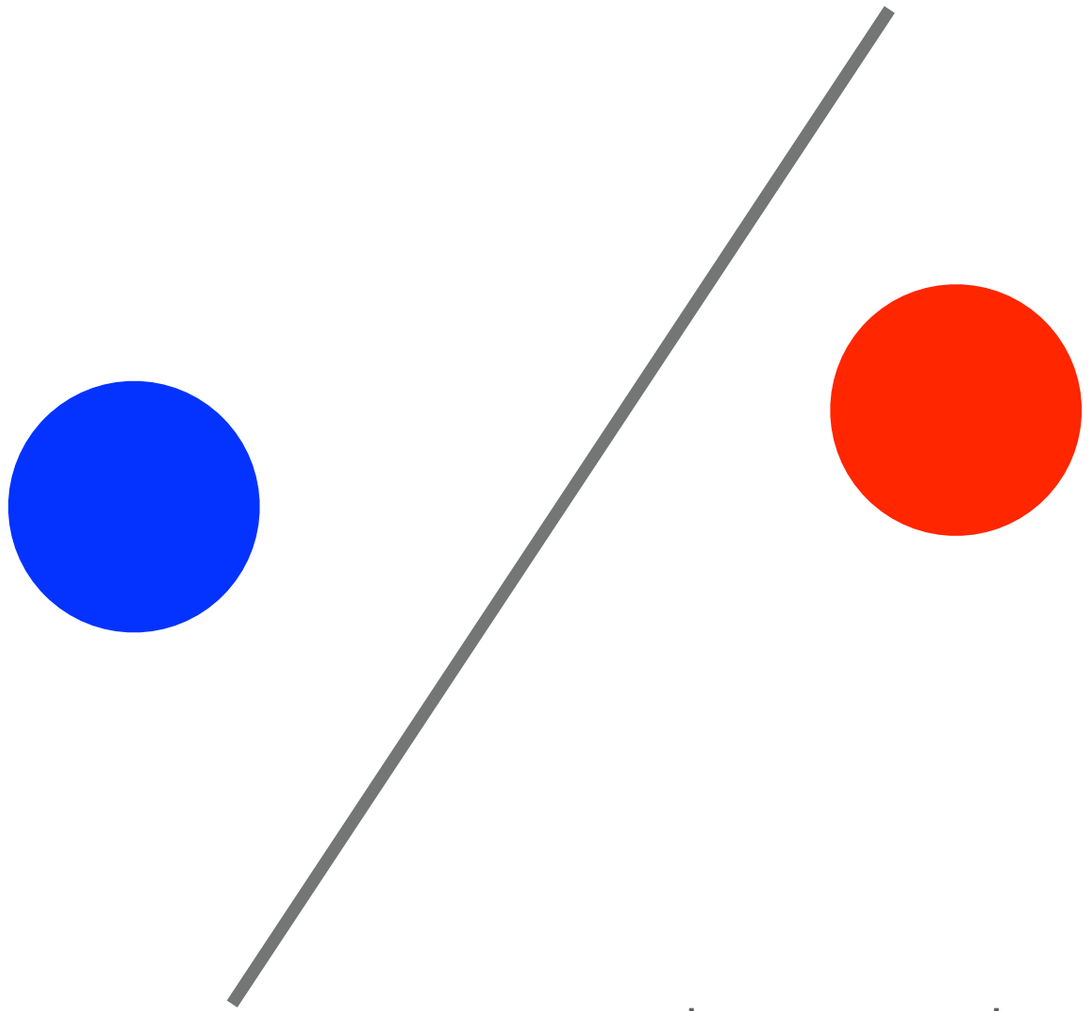
Are the depth 1, 2 and 3 architectures **too shallow** to represent efficiently more complicated functions needed for AI tasks?

there might be no universally right depth

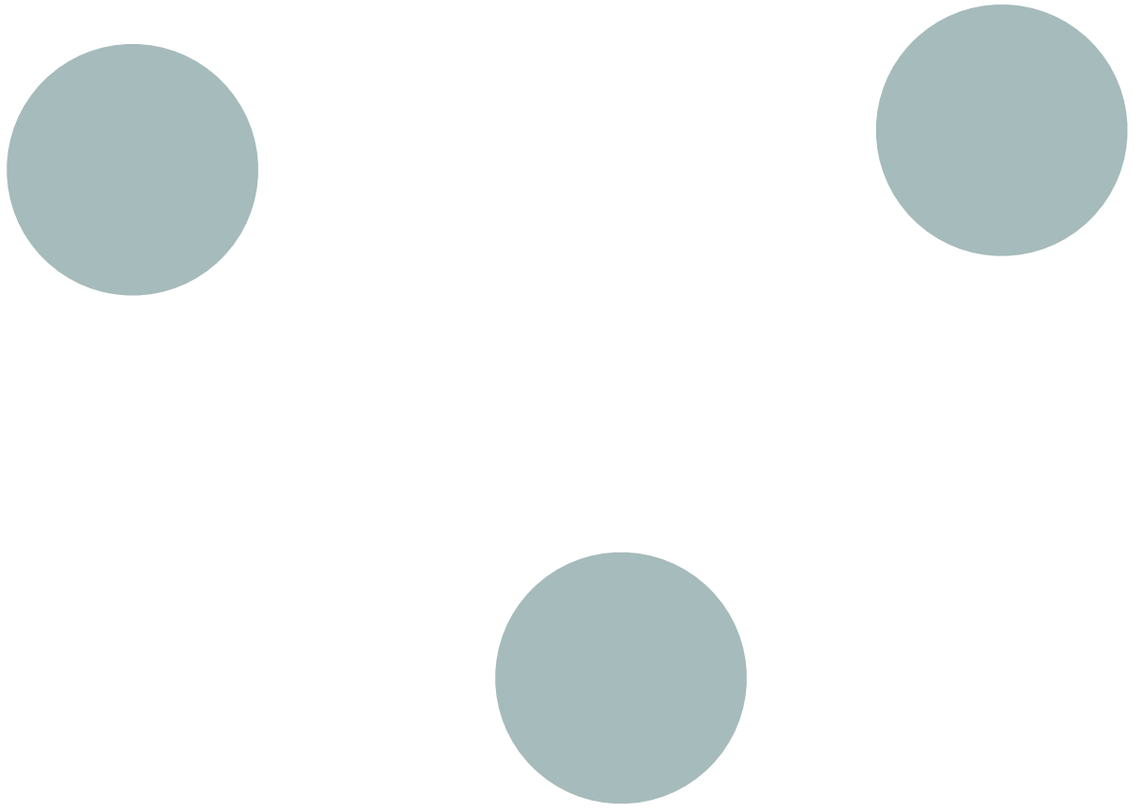


VC dimension

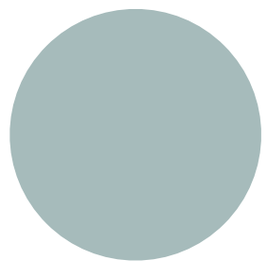
the largest number of points that a model parameterized by θ makes no error in correctly assigning labels

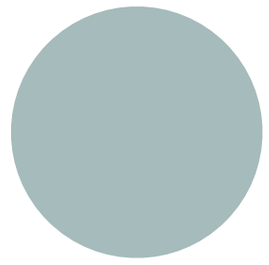


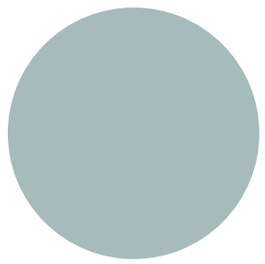
can these points be shattered by a line?

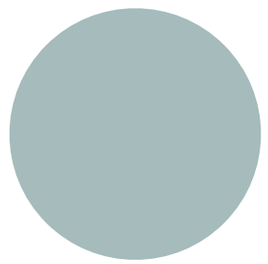


can these points be shattered by a line?









can these points be shattered by a line?

$$17 = 00\,1\,000\ldots0$$

17^{th} position, local

17 = 10001

distributed, compact

For the same number of possible configurations, a distributed representation can potentially be exponentially more compact than a very local one.

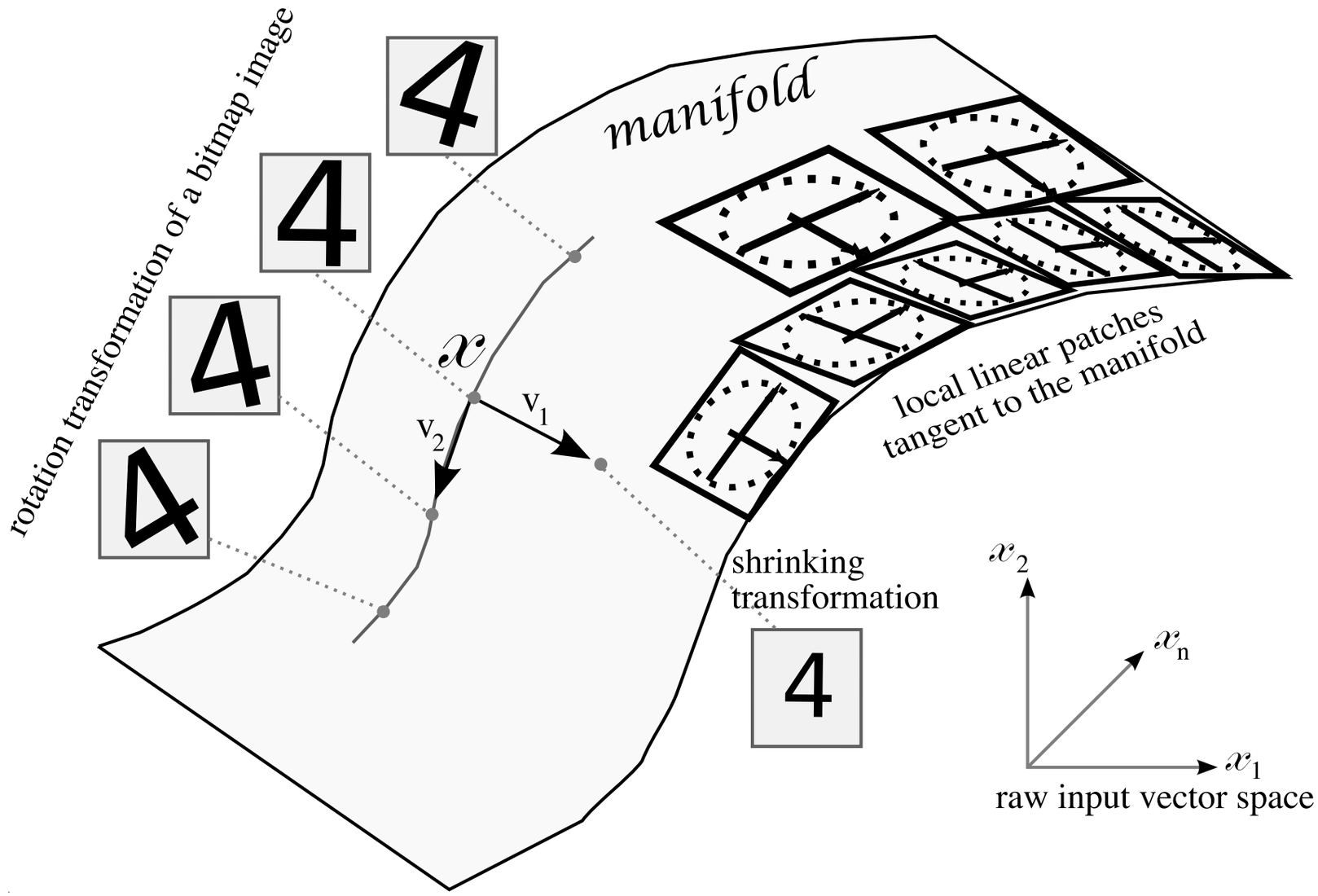


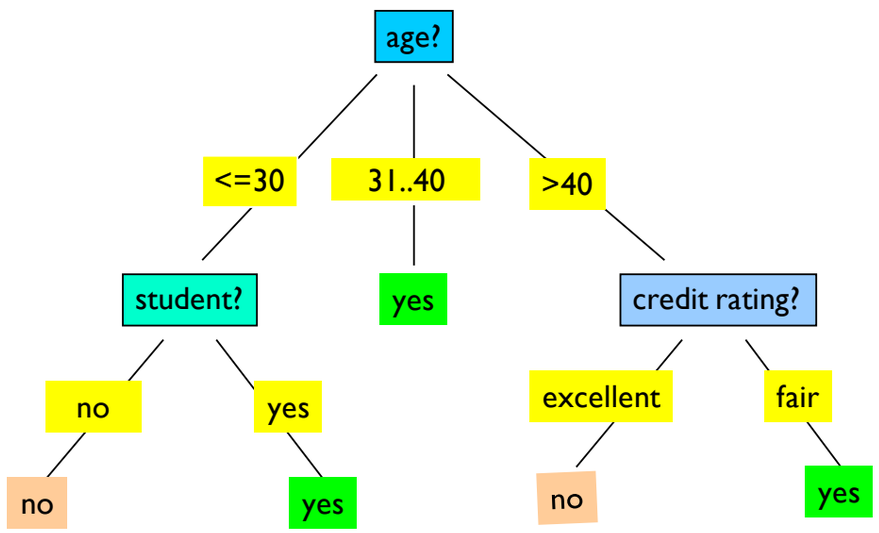
Fig. 3.1 The set of images associated with the same object class forms a manifold or a set of disjoint manifolds, i.e., regions of lower dimension than the original space of images. By rotating or shrinking, e.g., a digit 4, we get other images of the same class, i.e., on the same manifold. Since the manifold is locally smooth, it can in principle be approximated locally by linear patches, each being tangent to the manifold. Unfortunately, if the manifold is highly curved, the patches are required to be small, and exponentially many might be needed with respect to manifold dimension. Graph graciously provided by Pascal Vincent.

Sparsity: encouraging many units to take the value 0

allows for representations that are in between being fully local (i.e., maximally sparse) and non-sparse (i.e., dense) distributed representations.

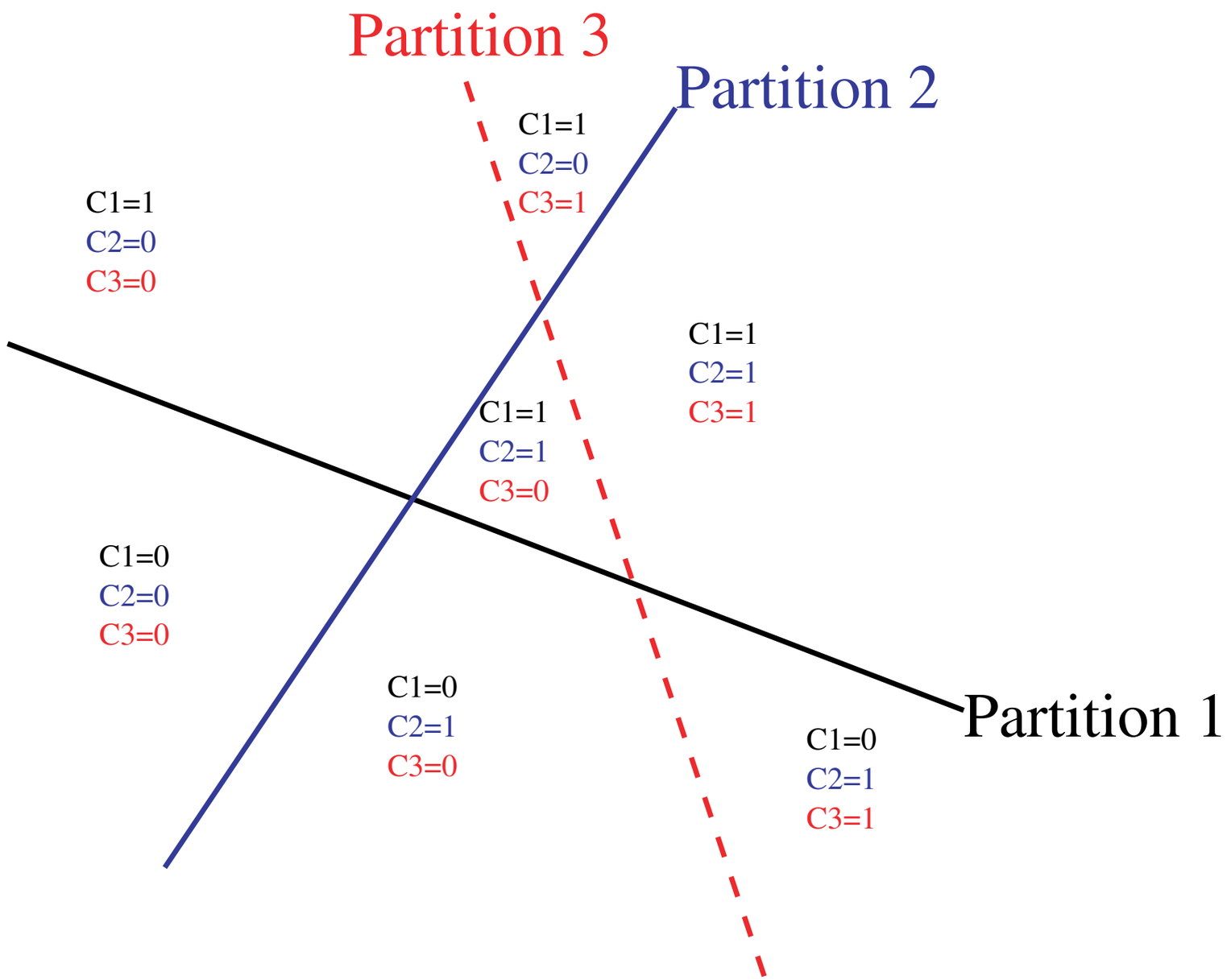
Neurons in the cortex are believed to have a distributed and sparse representation, with around 1-4% of the neurons active at any one time.

decision trees are local

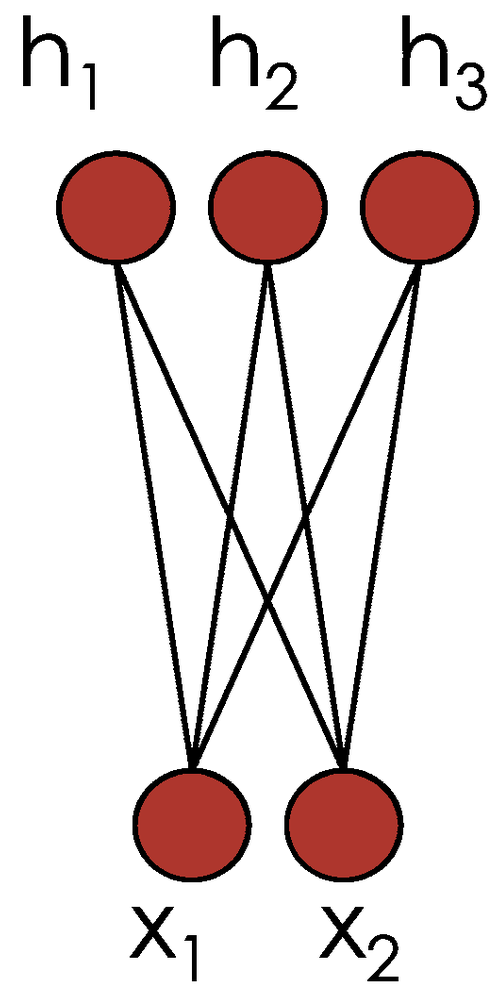


rely on a partition of the input space and using separate parameters for each region with each region associated with a leaf of the decision tree

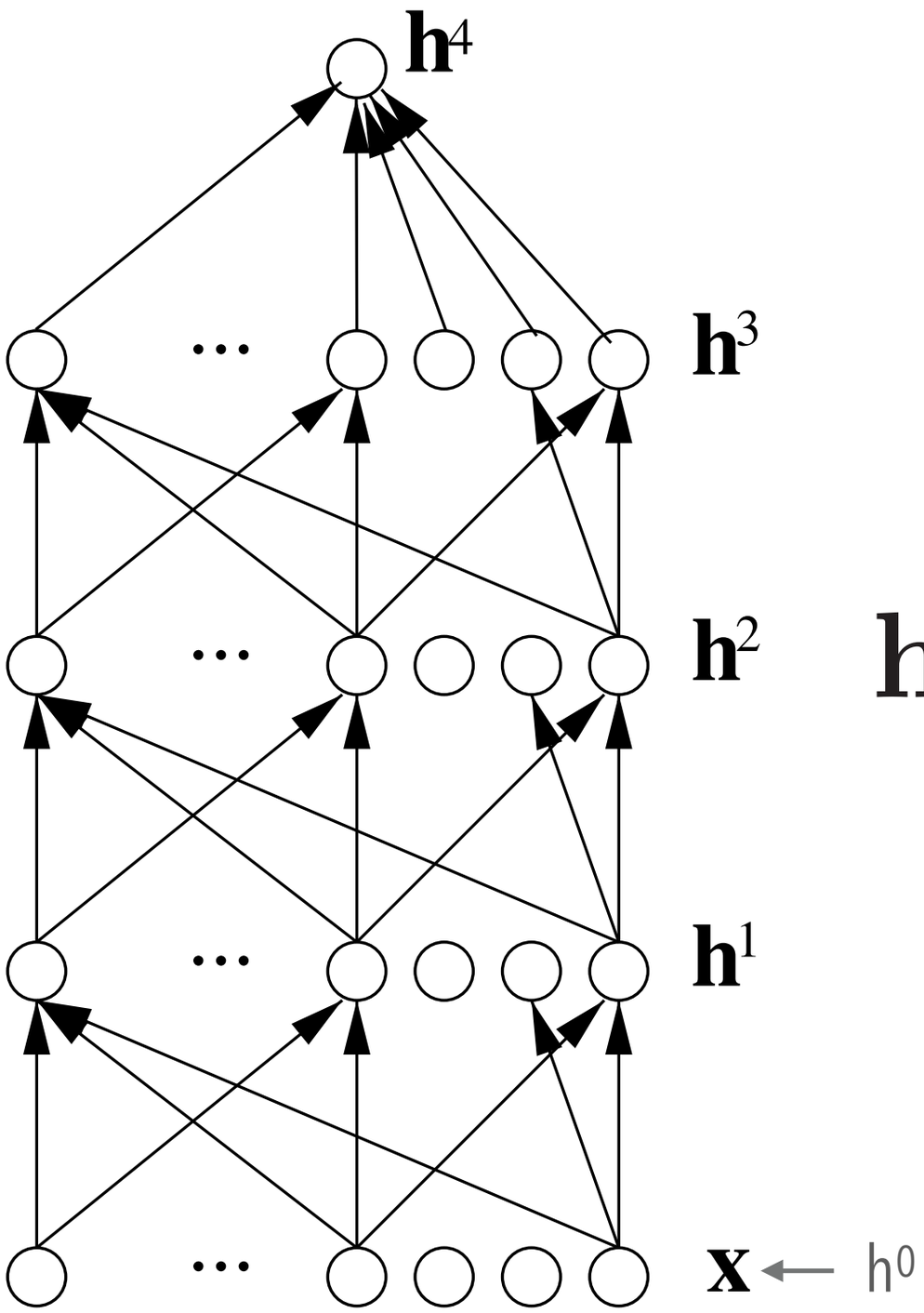
however tree ensembles are implicitly distributed



RBM with three hidden units

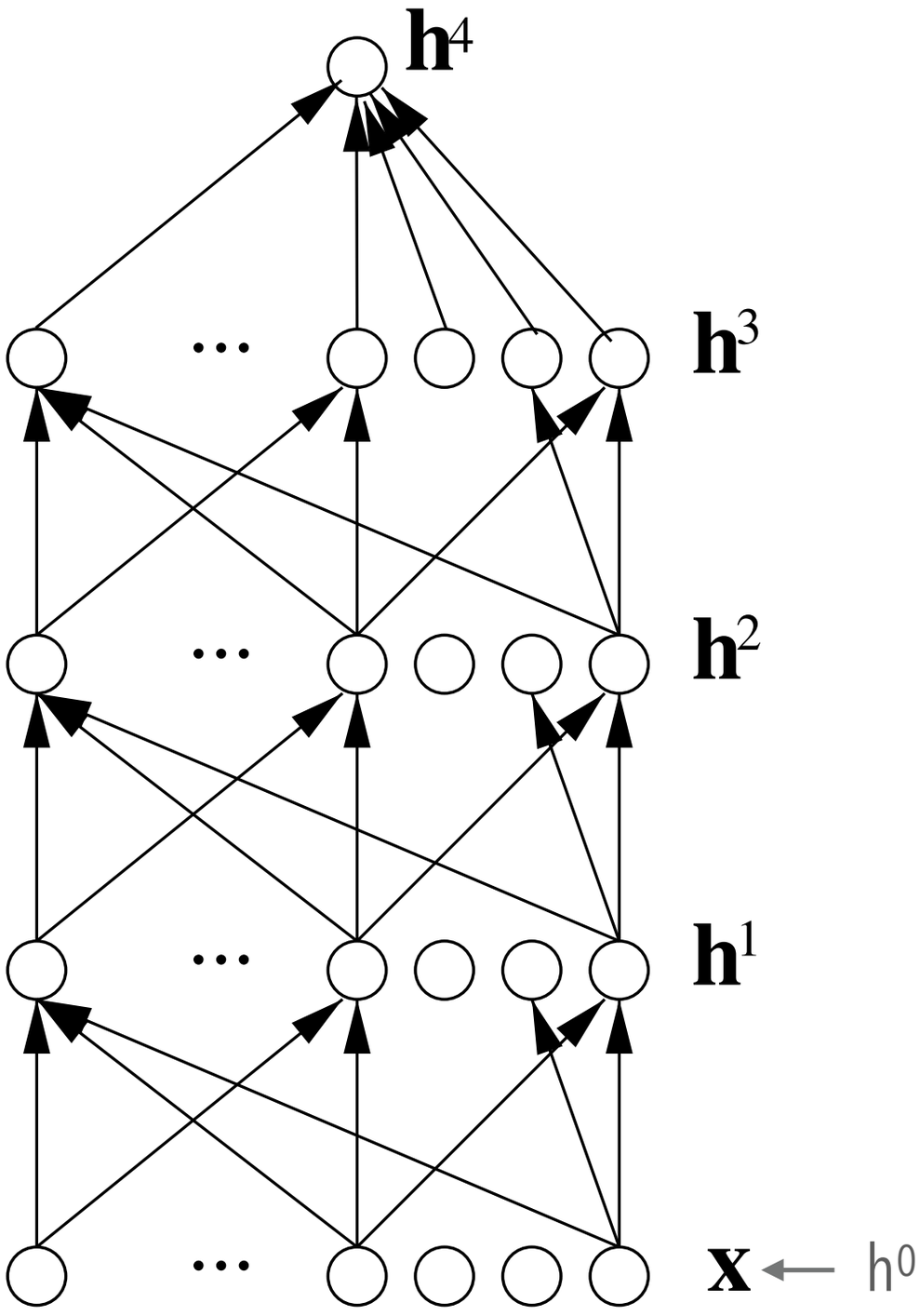


An ensemble of trees can discriminate among a number of regions exponential in the number of trees

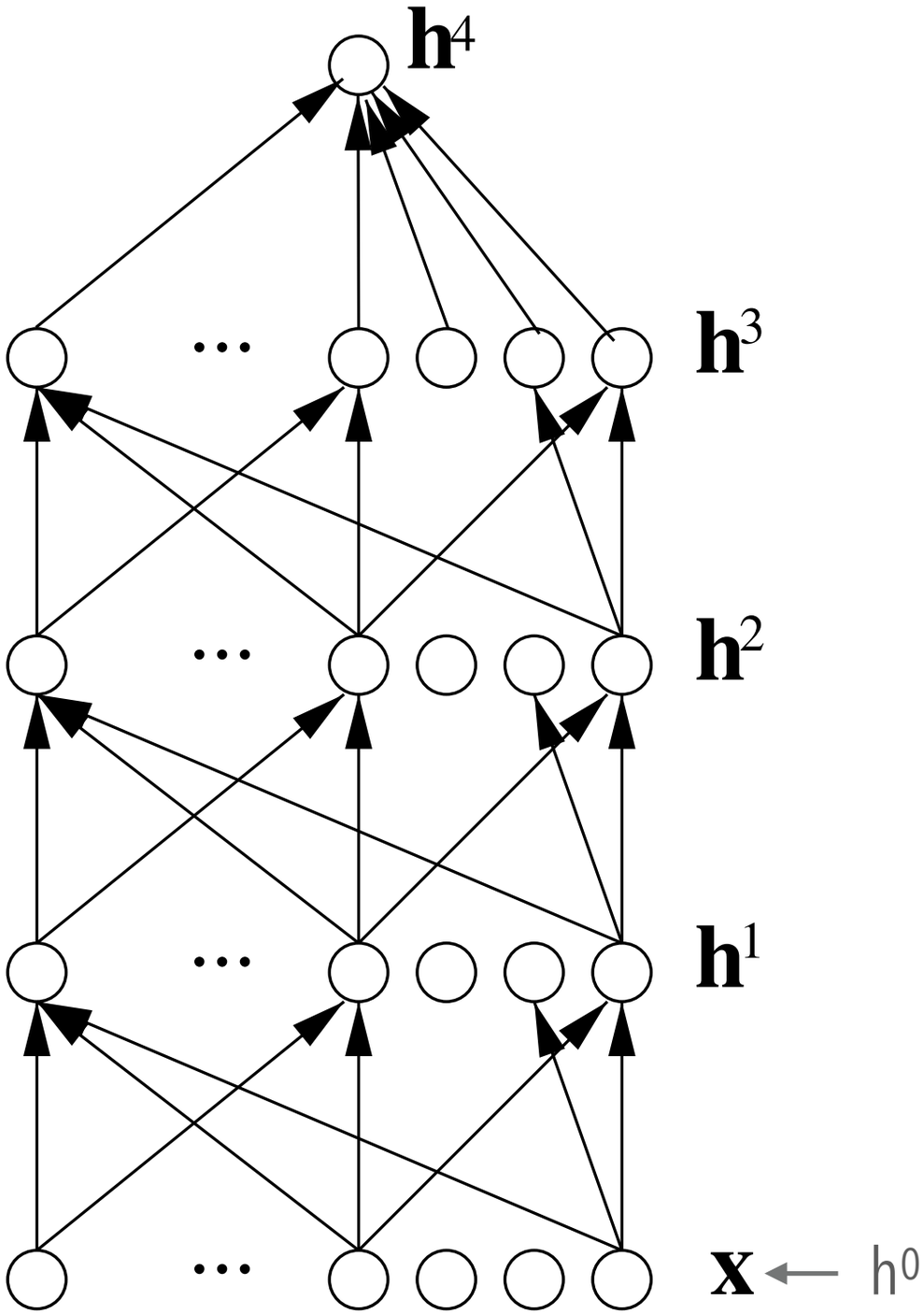


$$\mathbf{h}_i^\ell = \frac{e^{\mathbf{b}_i^\ell + W_i^\ell \mathbf{h}^{\ell-1}}}{\sum_j e^{\mathbf{b}_j^\ell + W_j^\ell \mathbf{h}^{\ell-1}}}$$

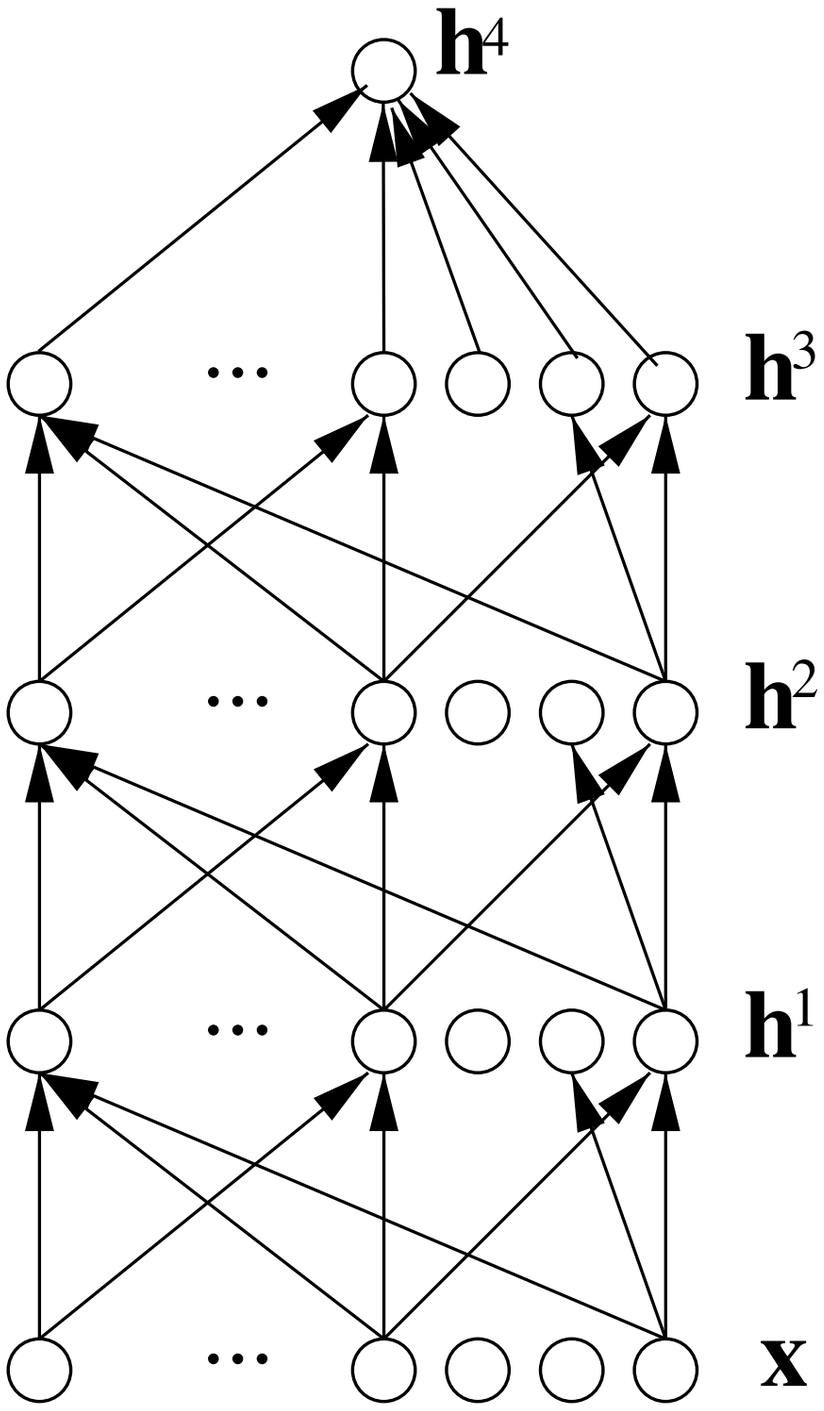
$$\mathbf{h}^k = \tanh(\mathbf{b}^k + W^k \mathbf{h}^{k-1})$$



what could be a major difference with shallow networks?



learning weights of the various hidden layers is hard!



why?

what is it about depth that makes it hard?

consider a jury

what was his impact in the final decision?







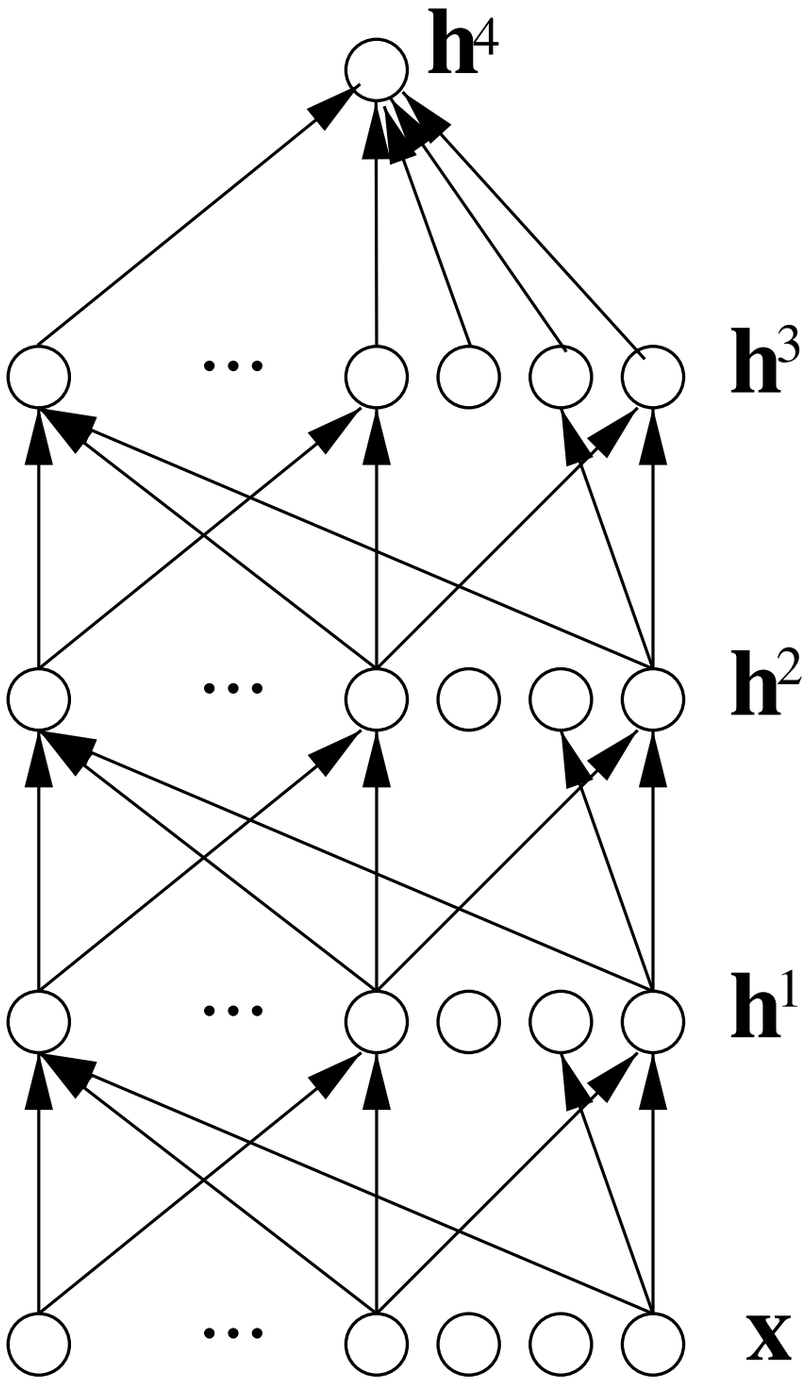






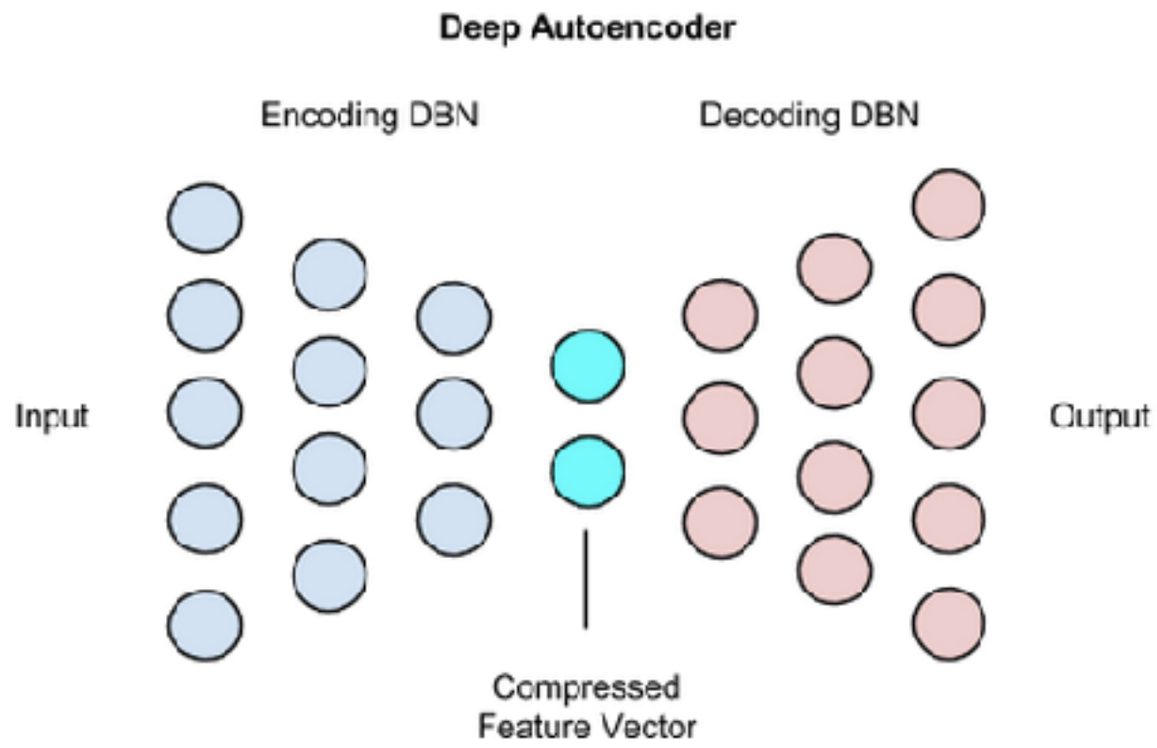
gradients become less meaningful with lots of layers, with random initialization

When starting from **random initialization**, the solutions obtained with deeper neural networks appear to correspond to poor solutions that perform worse than the solutions obtained for networks with 1 or 2 hidden layers



the major change: unsupervised pre-training

unsupervised pre-training helps generalization by allowing for a 'better' tuning of lower layers of a deep architecture.

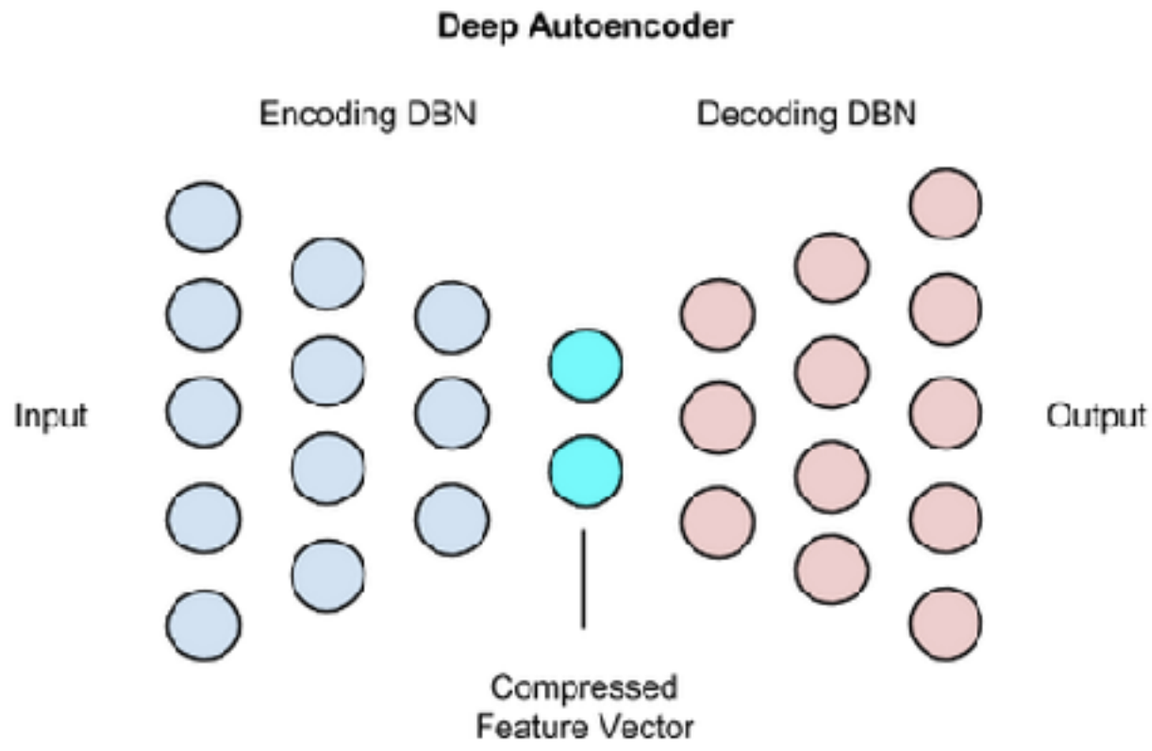


auto-encoders

$$x = f(g(x))$$

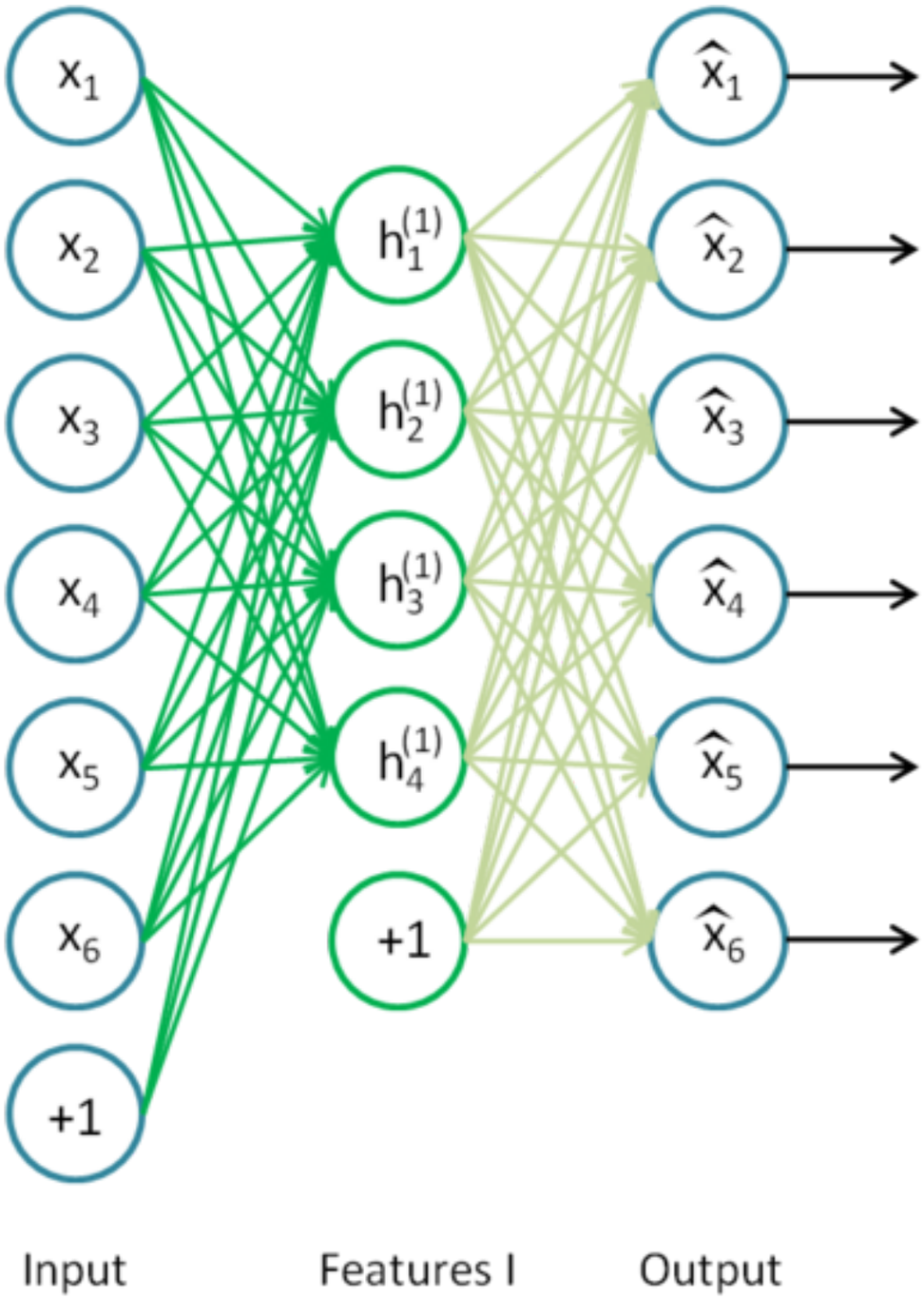
decoder encoder

why don't auto encoders simply learn **the** identity function**?**

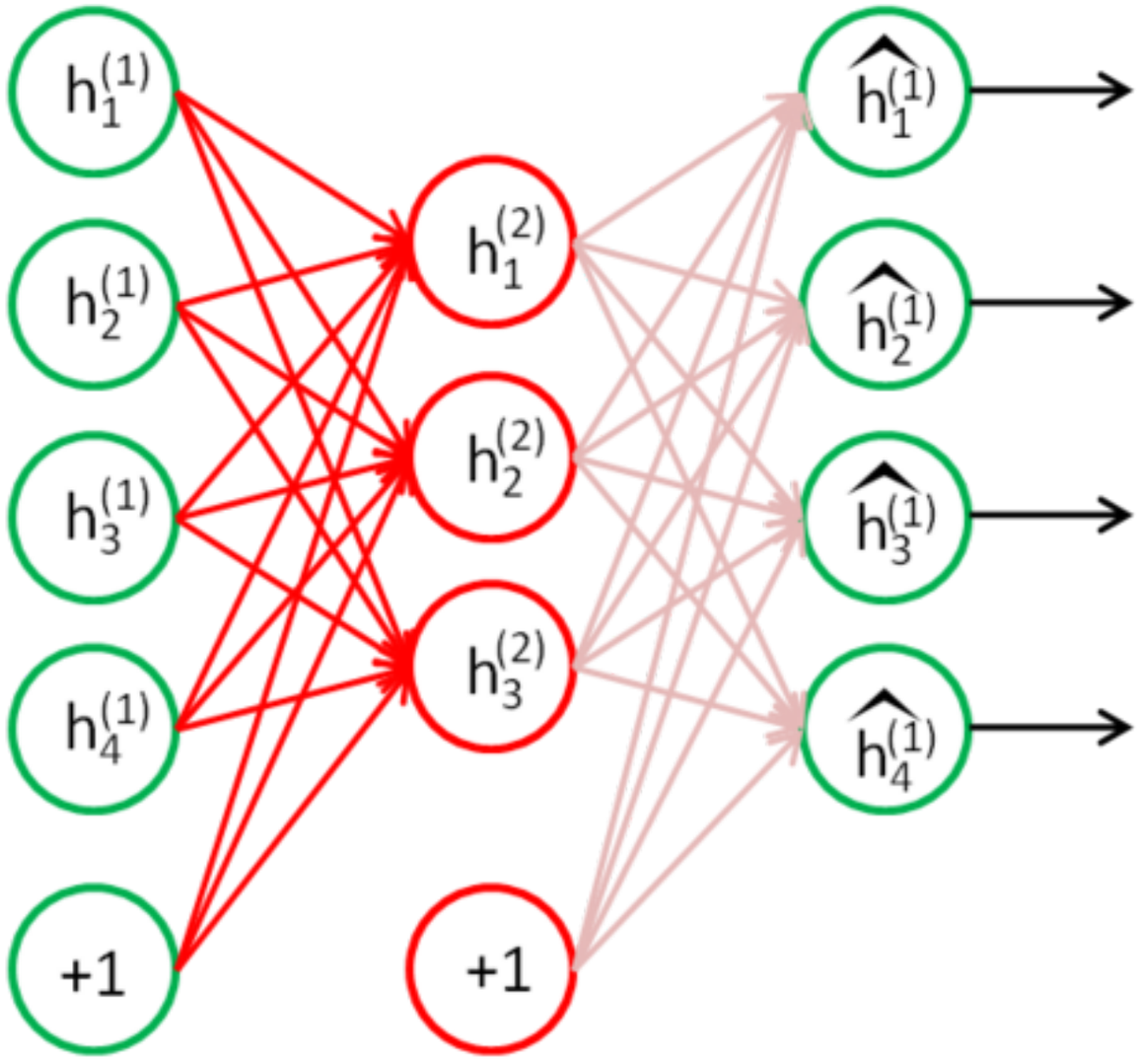


$$x = f(g(x))$$

stacked auto-encoders

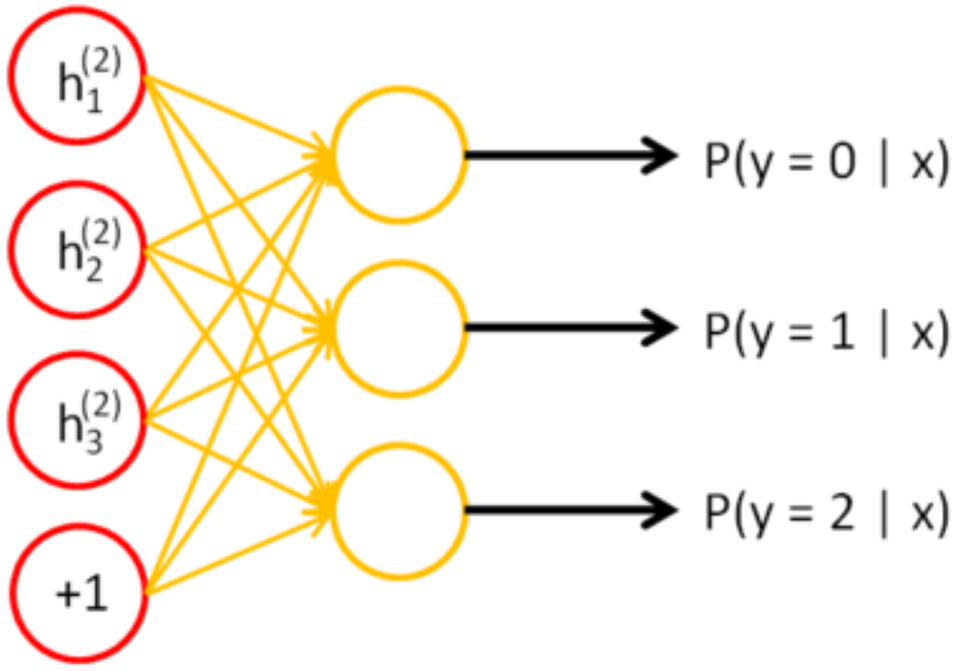


stacked auto-encoders



Input (Features I) Features II Output

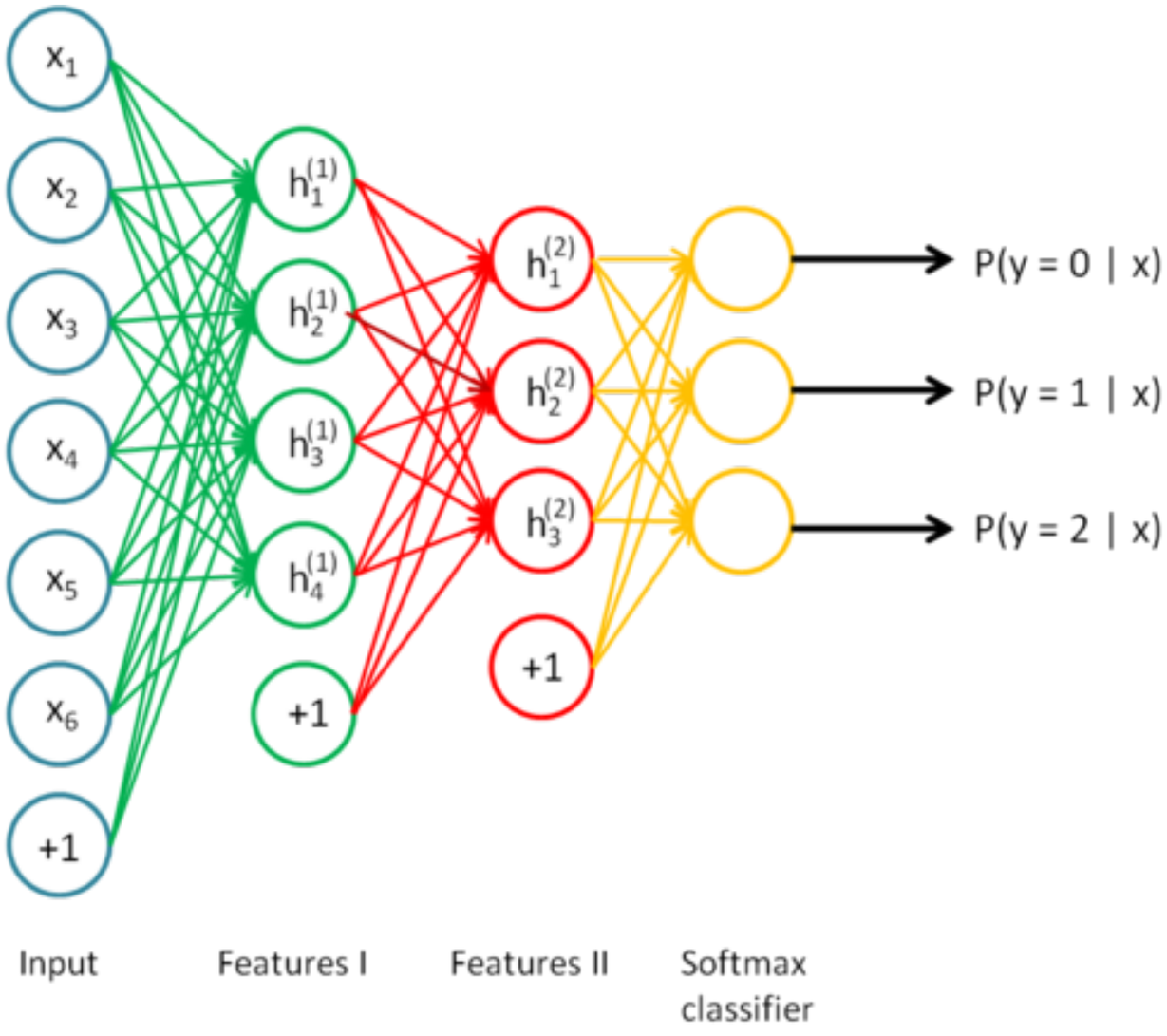
stacked auto-encoders



Input (Features II)

Softmax classifier

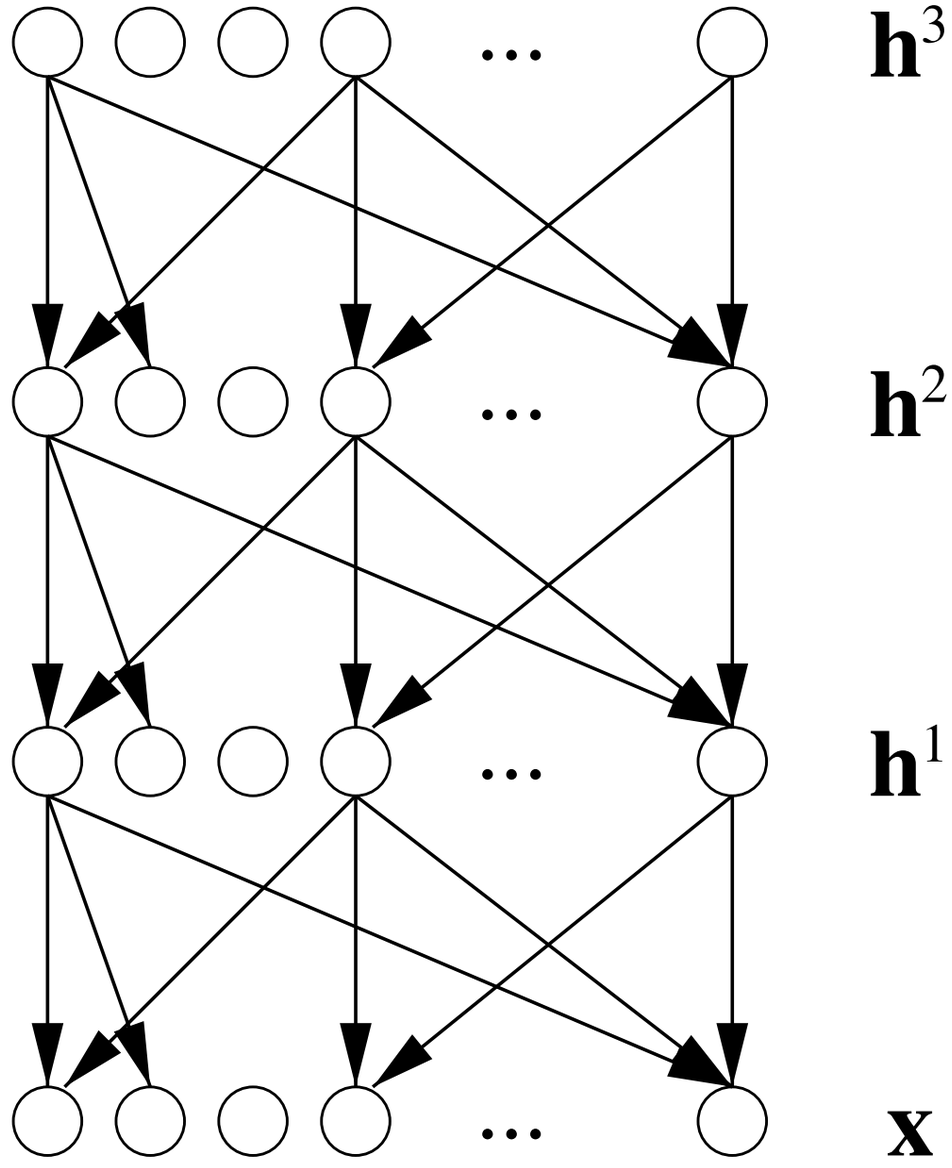
stacked auto-encoders



$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = P(\mathbf{h}^\ell) \left(\prod_{k=1}^{\ell-1} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x} | \mathbf{h}^1)$$

$$P(\mathbf{h}_i^k = 1 | \mathbf{h}^{k+1}) = \text{sigm}(\mathbf{b}_i^k + \sum_j W_{i,j}^{k+1} \mathbf{h}_j^{k+1})$$

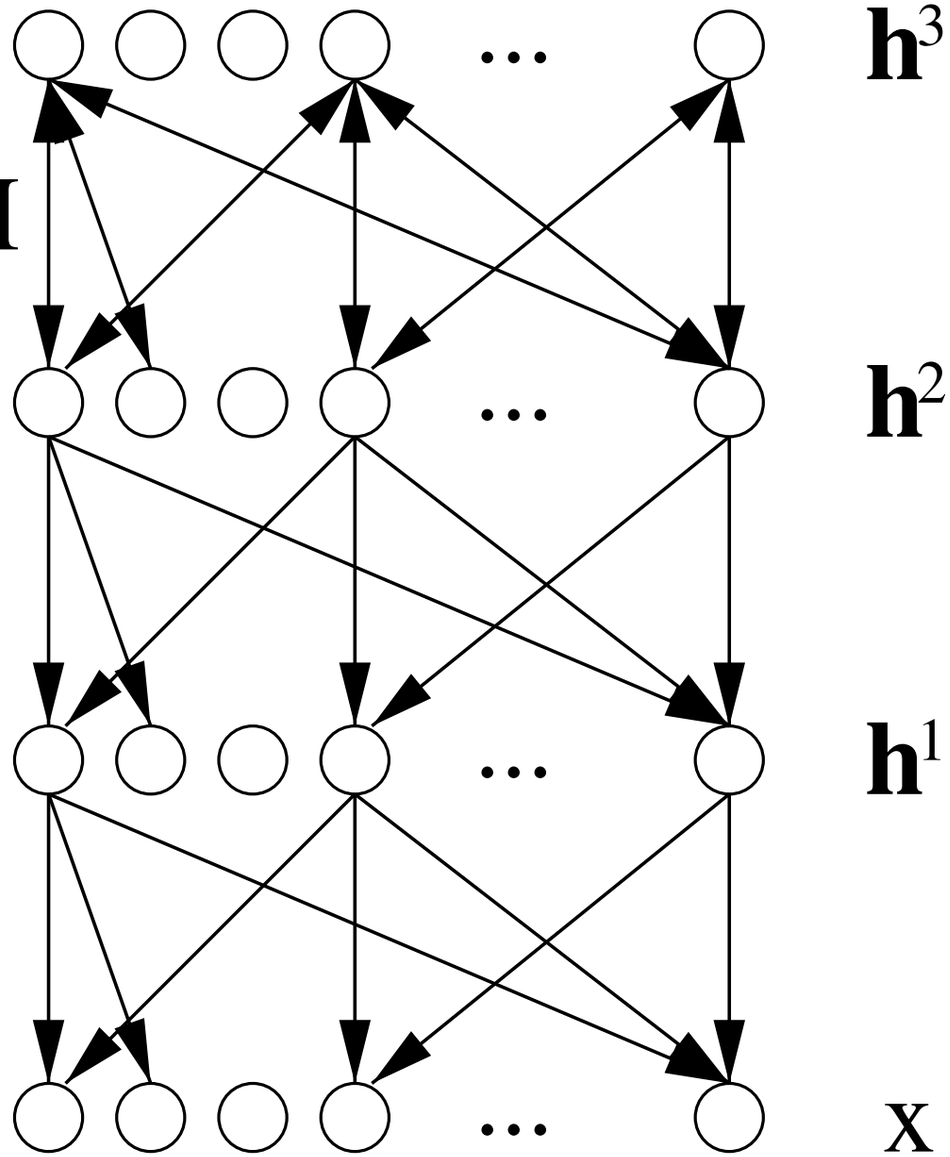
deep generative architectures



note the direction of the arrows

noticed the undirected arrows

$$P(\mathbf{h}^2, \mathbf{h}^3) \sim \text{RBM}$$



$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell) \left(\prod_{k=1}^{\ell-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x} | \mathbf{h}^1).$$

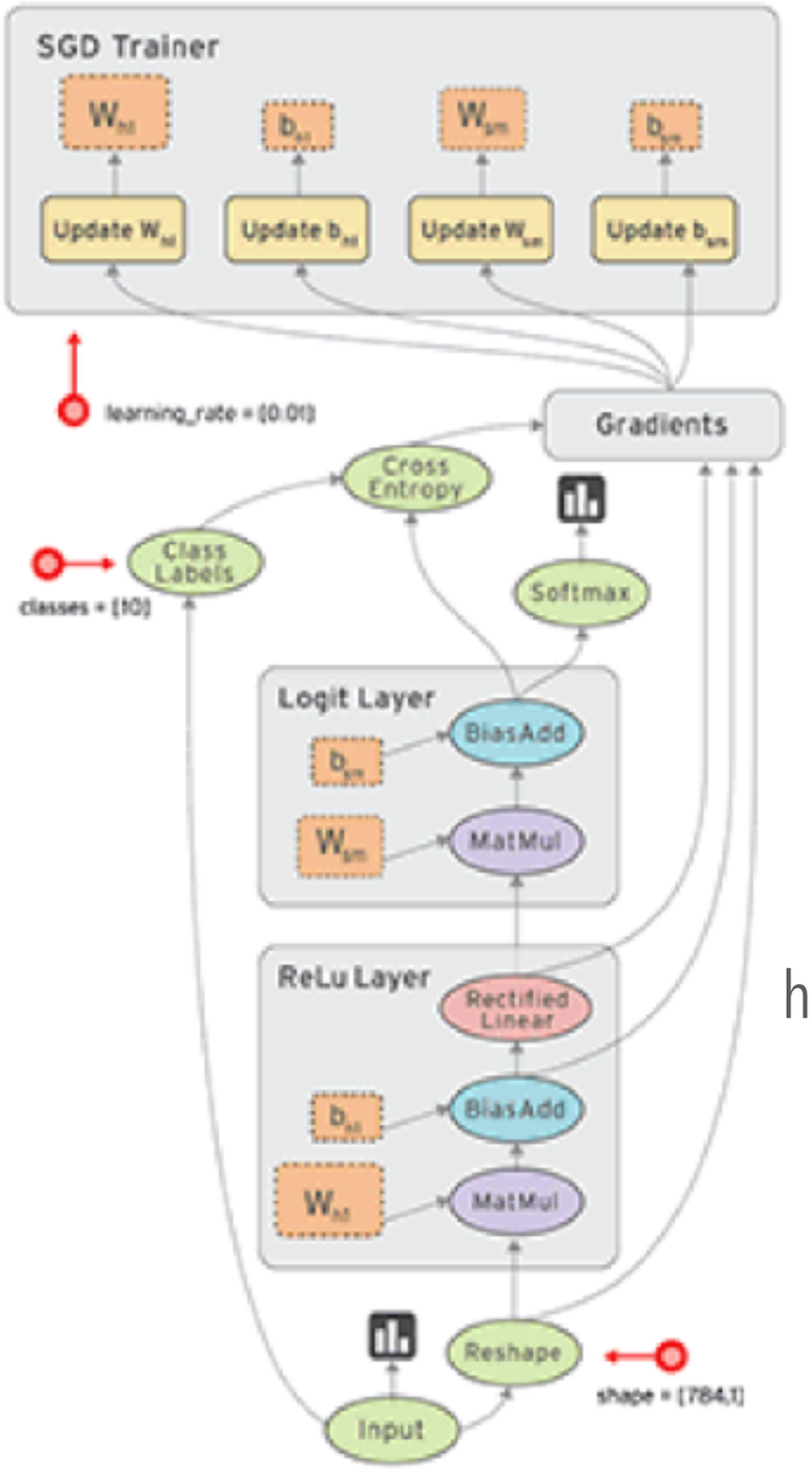
$$P(\mathbf{h}^{\ell-1}, \mathbf{h}^\ell) \propto e^{\mathbf{b}'\mathbf{h}^{\ell-1} + \mathbf{c}'\mathbf{h}^\ell + \mathbf{h}^{\ell\prime} W \mathbf{h}^{\ell-1}}$$

restricted boltzmann machine

$$\boldsymbol{X}_i \rightarrow \boldsymbol{X}_{i+1}$$

Gibbs Sampling

The point of Gibbs sampling is that given a multivariate distribution it is simpler to sample from a conditional distribution than to marginalize by integrating over a joint distribution.



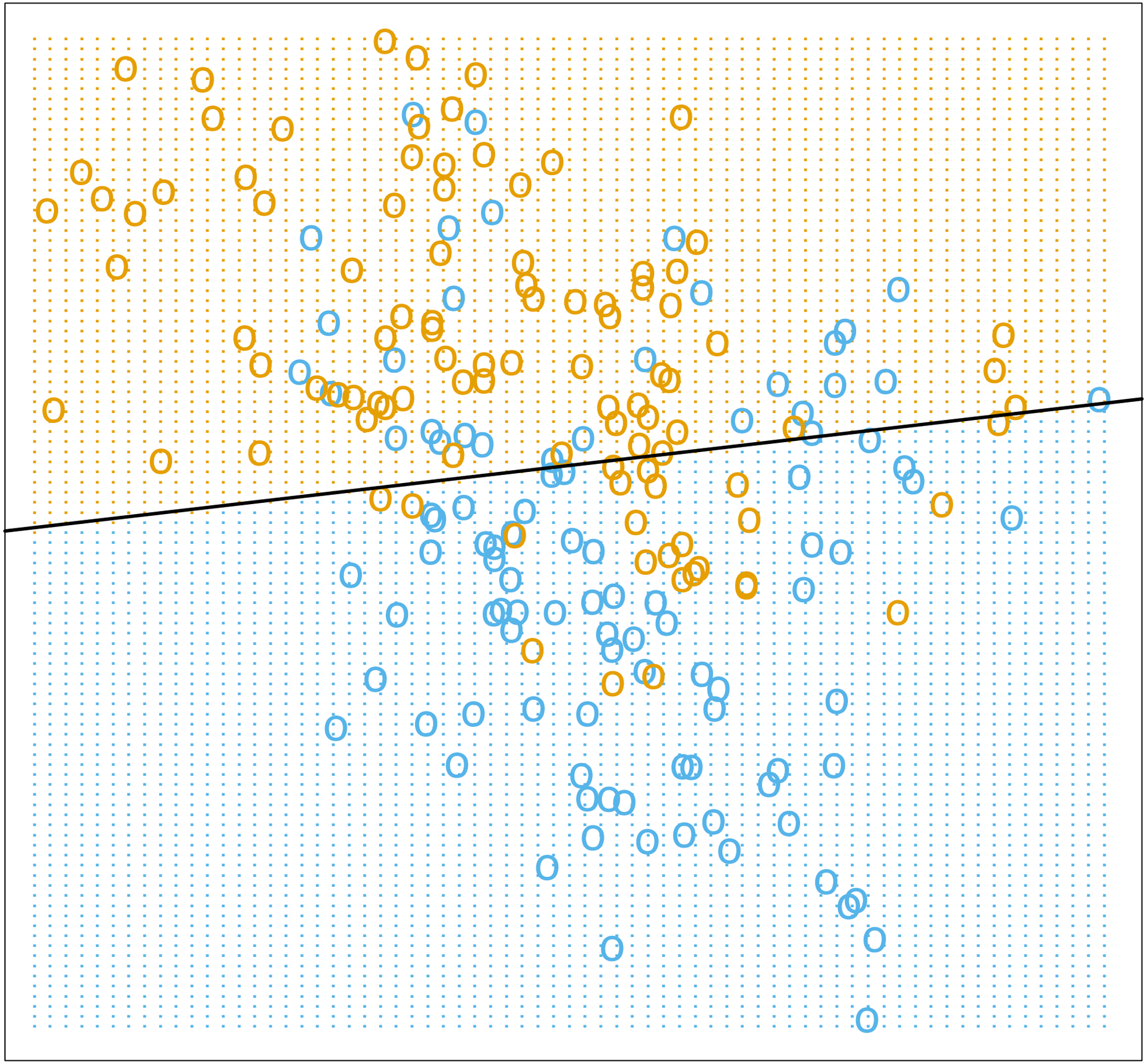
https://www.tensorflow.org

http://deeplearning.net/software/theano/index.html#

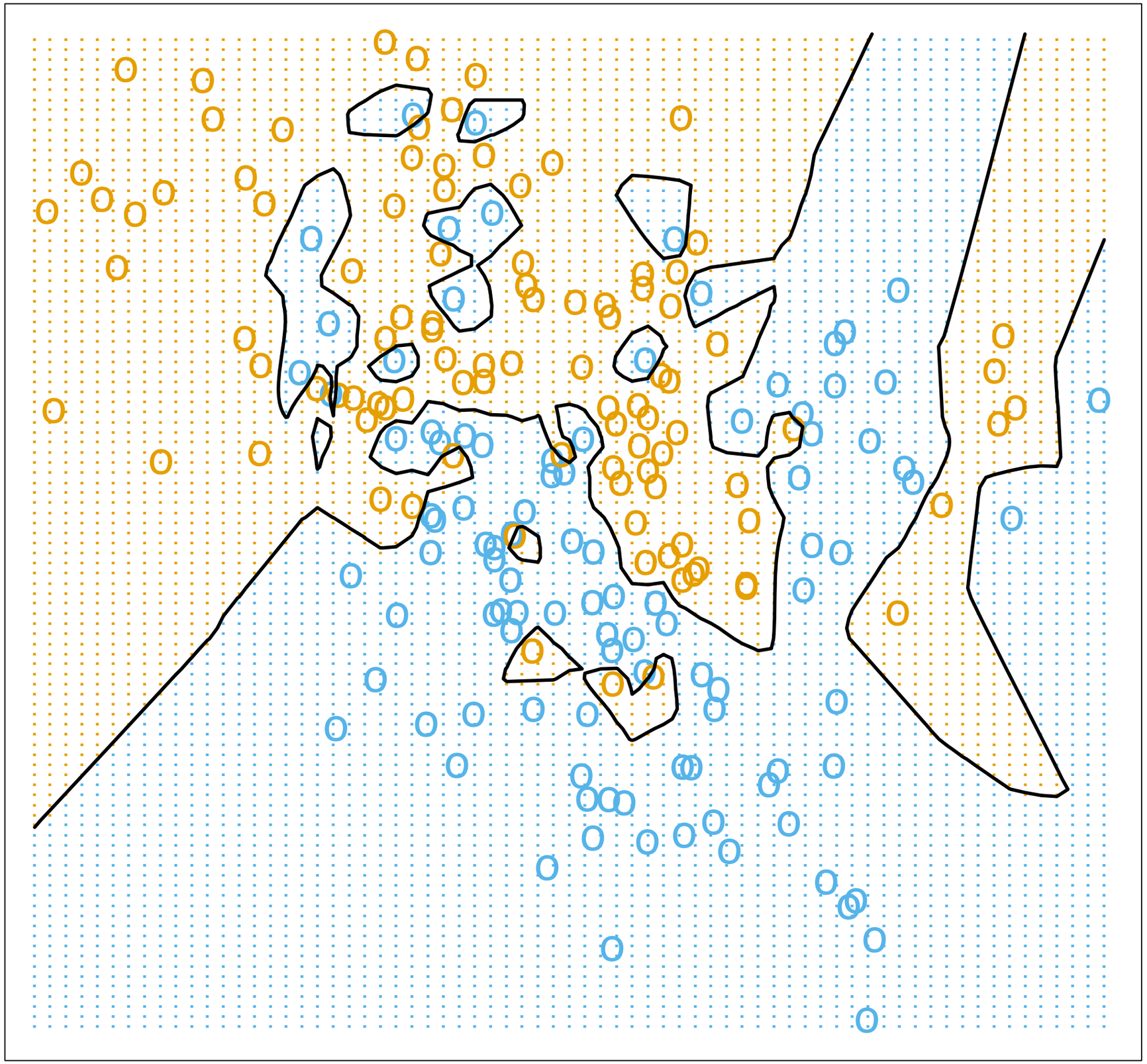
ETC.

Bayesian Networks Support Vector Machines Neural Networks

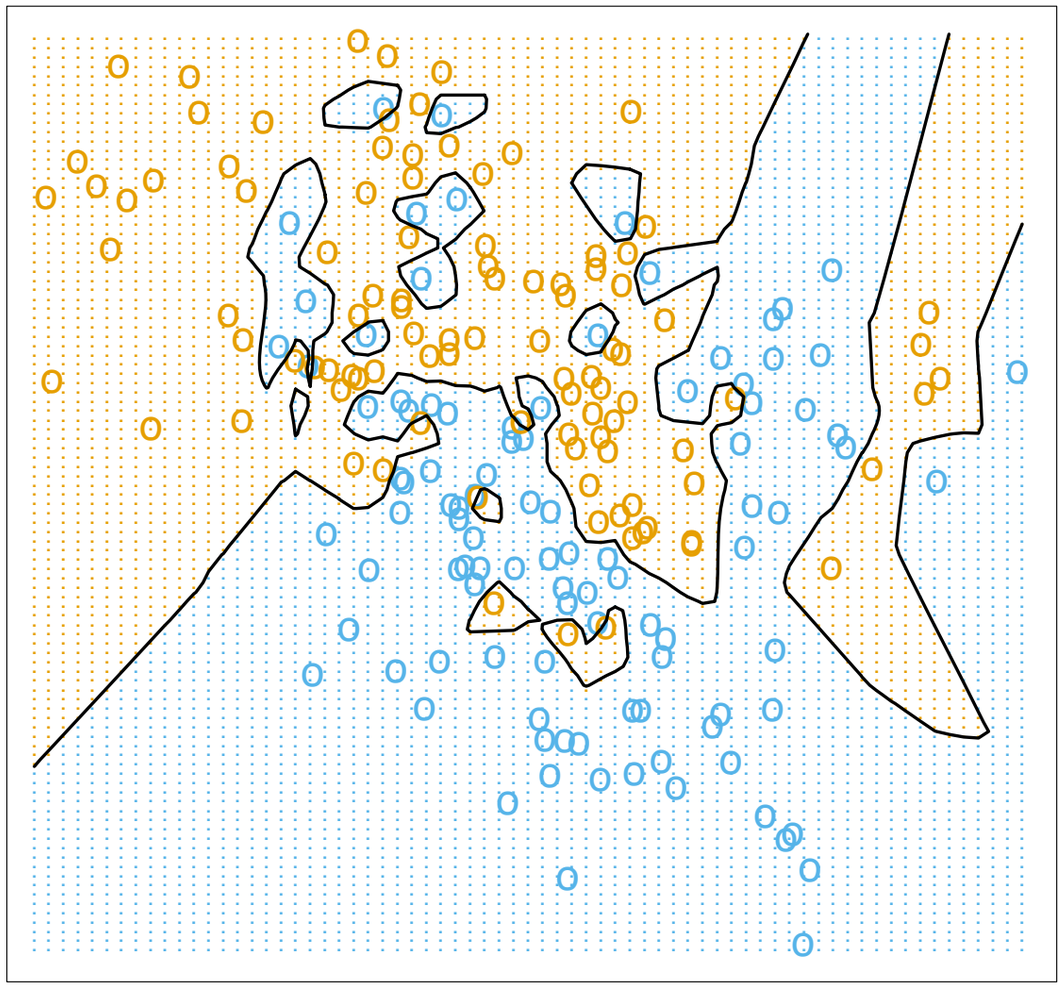
Summary

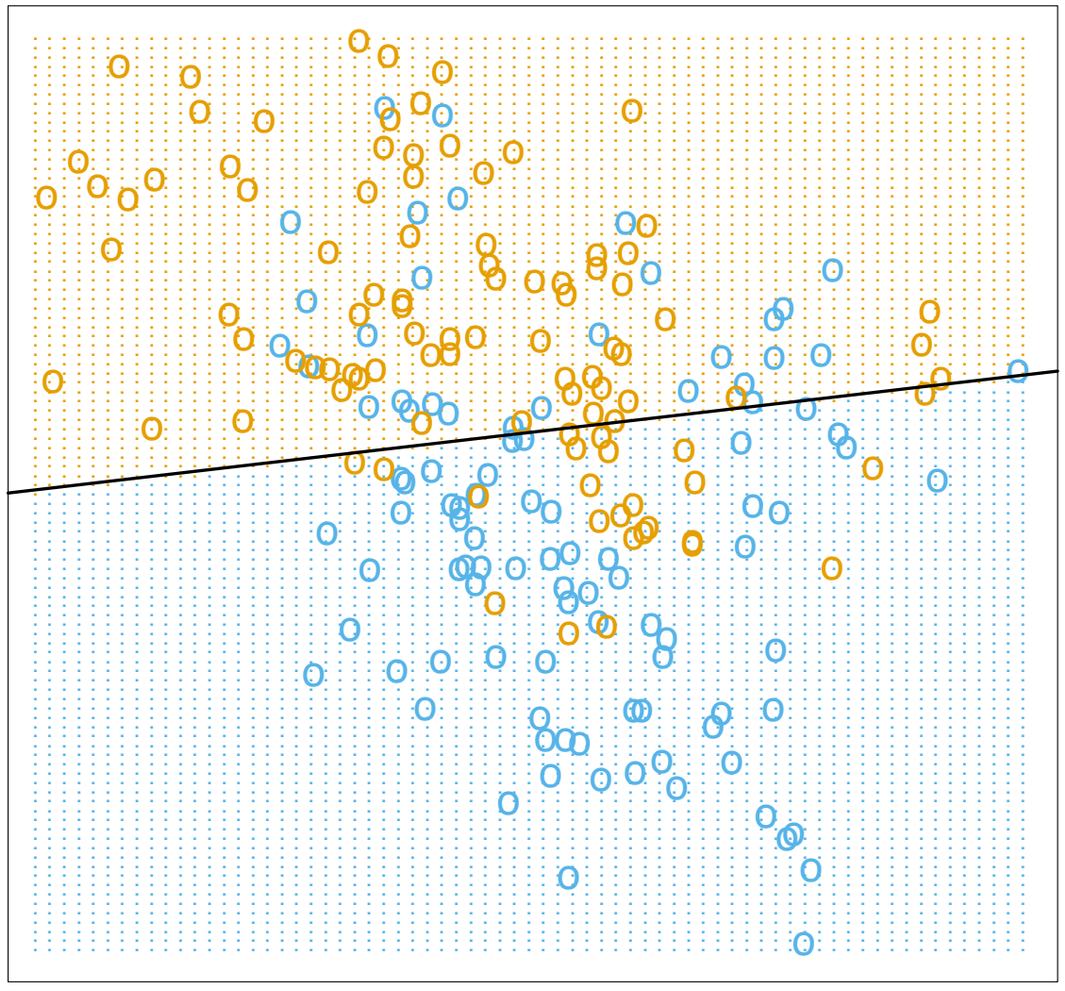


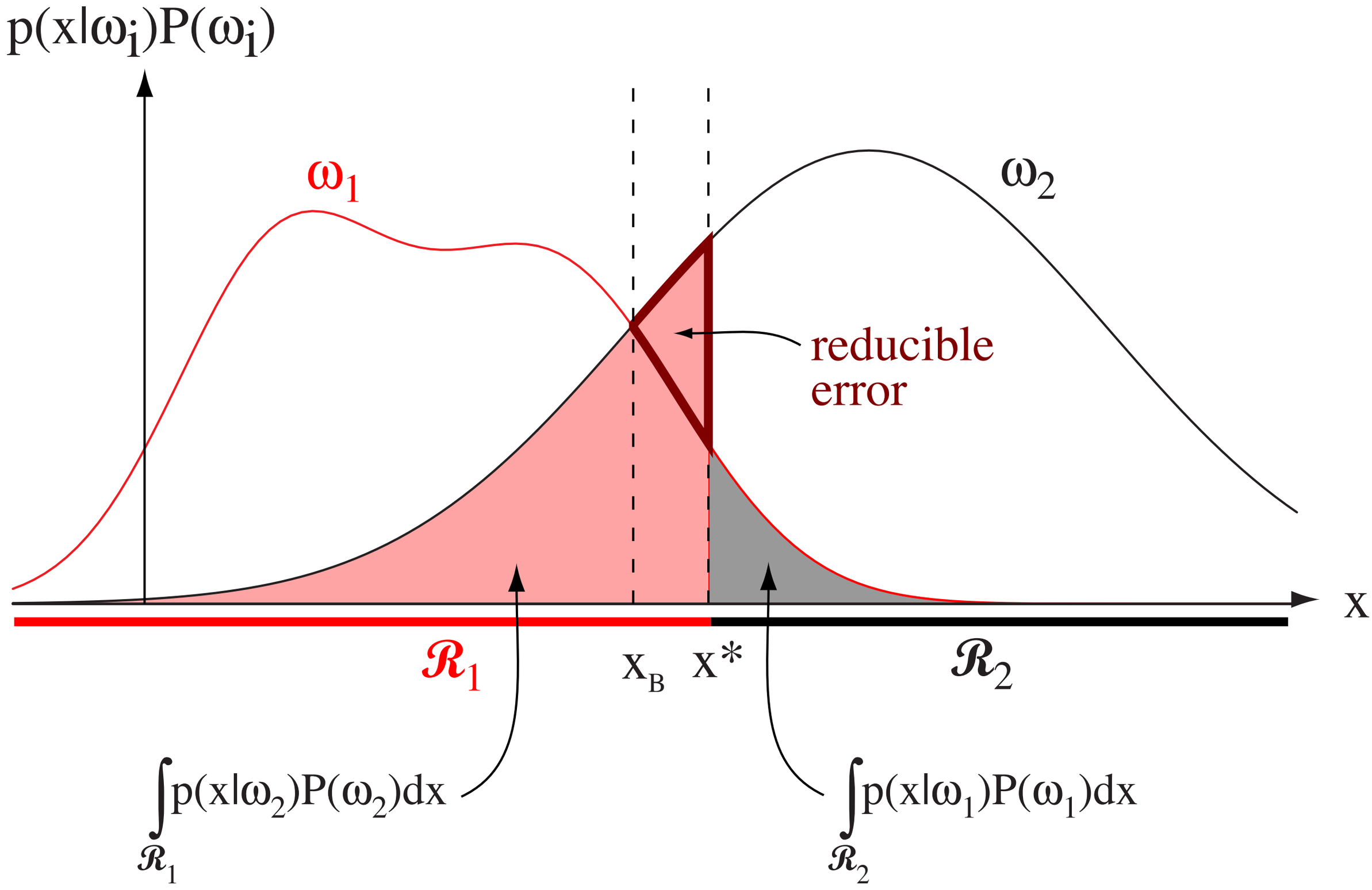
1-Nearest Neighbor Classifier



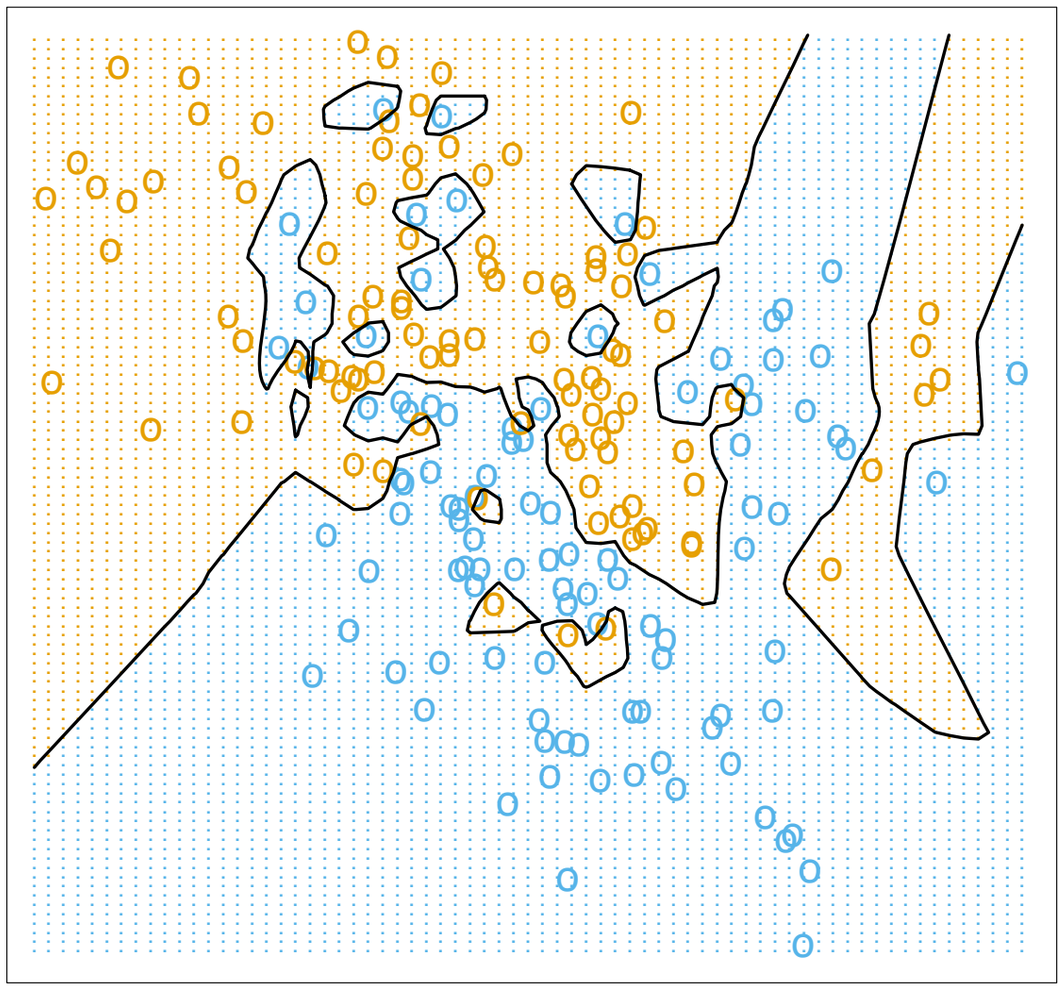
what is the difference?






p(x|ωi)P(ωi)
ω1
ω2
reducible
error
x
R1
R2
xB
x*
∫R1 p(x|ω2)P(ω2)dx
∫R2 p(x|ω1)P(ω1)dx

K-NN



All instances correspond to points in the n-D space

The nearest neighbor are defined in terms of Euclidean distance, $d(X_1, X_2)$

Target function could be discrete or real valued

For discrete-valued, k-NN returns the most common value among the k training examples nearest to x_q

MULTI CLASS

Method 1. One-vs.-all: Learn a classifier one at a time

- Given m classes, train m classifiers: one for each class
- Classifier j: treat tuples in class j as positive & all others as negative
- To classify a tuple X, the set of classifiers vote as an ensemble

Method 2. All-vs.-all: Learn a classifier for each pair of classes

- Given m classes, construct m(m-1)/2 binary classifiers
- A classifier is trained using tuples of the two classes
- To classify a tuple X, each classifier votes. X is assigned to the class with maximal vote

Build a classifier using the labeled data

Use it to label the unlabeled data,
and those with the most confident
label prediction are added to the
set of labeled data

Repeat

semi-supervised

Uses labeled and unlabeled data to build a classifier

Each learner uses a mutually independent set of features of each tuple to train a good classifier, say f_1

Then f_1 and f_2 are used to predict the class label for unlabeled data X

The tuple having the most confident prediction from f_1 is added to the set of labeled data for f_2, & vice versa

co-training

ACTIVE LEARNING

Class labels are expensive to obtain

Active learner: query human (oracle) for labels

Pool-based approach: Uses a pool of unlabeled data

L: a small subset of D is labeled, U: a pool of unlabeled data in D

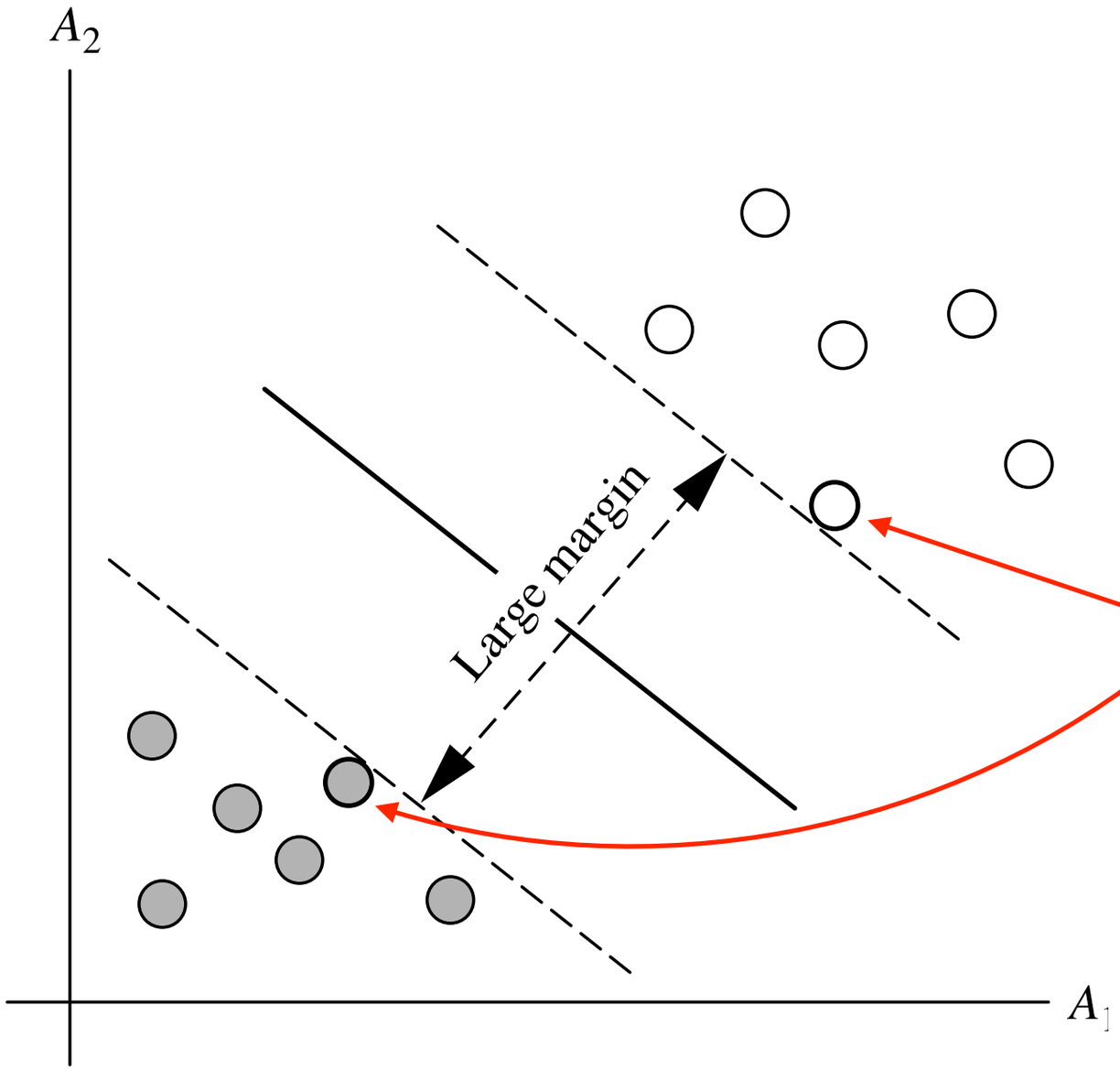
Use a query function to carefully select one or more tuples from U and request labels from an oracle (a human annotator)

The newly labeled samples are added to L, and learn a model

Goal: Achieve high accuracy using as few labeled data as possible

How to choose the data tuples to be queried?

choose the least certain ones



The support vectors are the essential or critical training examples–they lie closest to the decision boundary (MMH)

transfer learning

works best when the data shares regularities