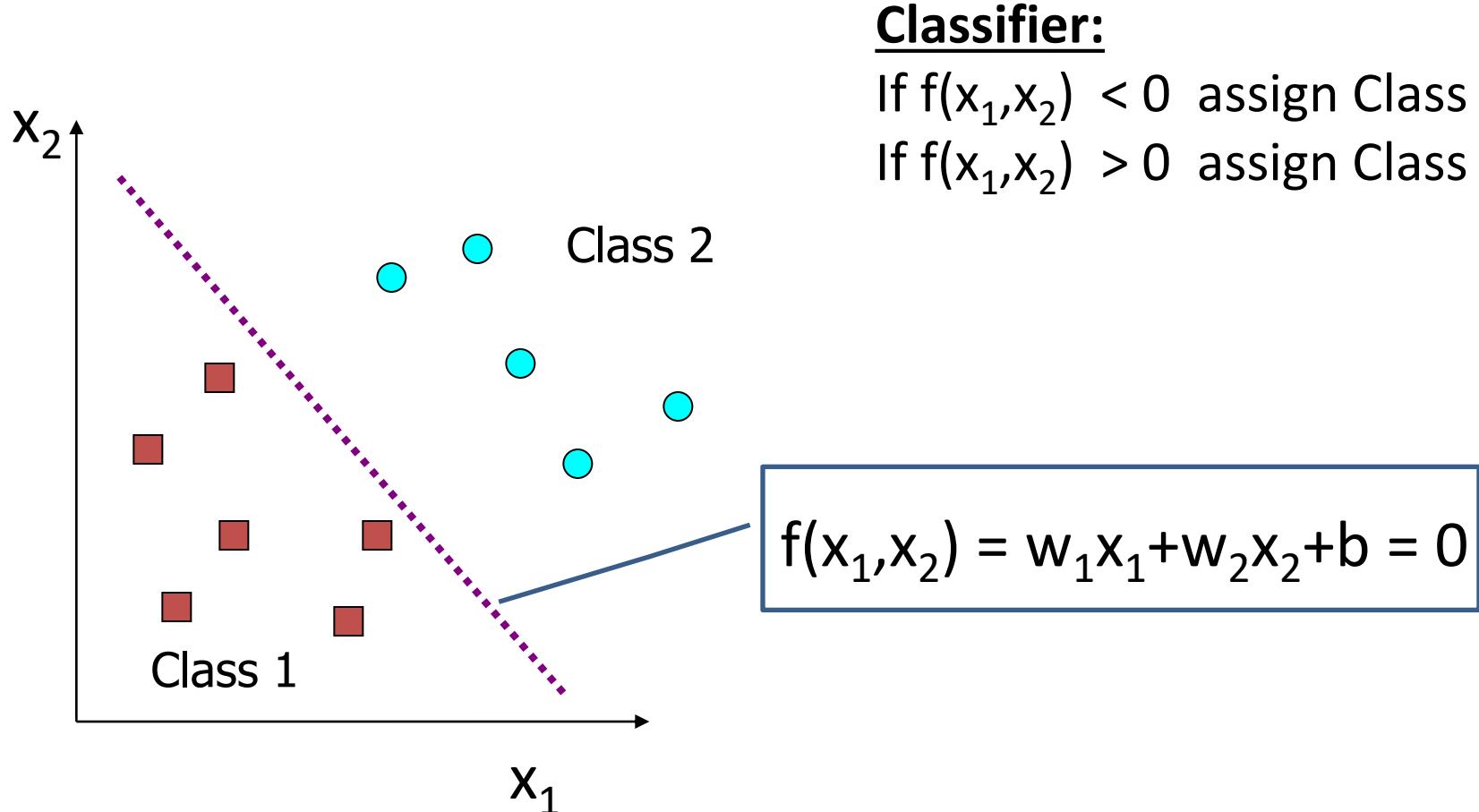


Support Vector Machines: An Introduction

Mostly informal

Linear Classifier



Perceptron

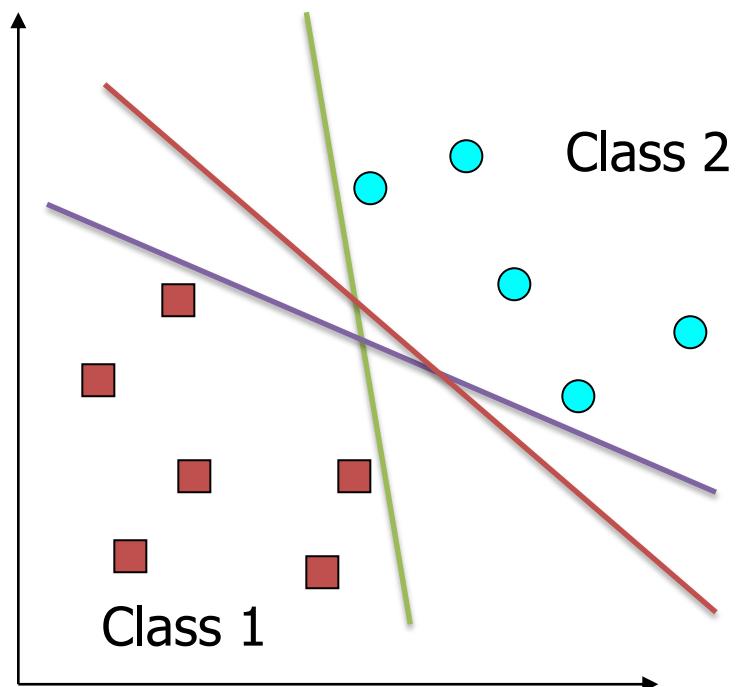
- Perceptron is the name given to the linear classifier.
- If there exists a Perceptron that correctly classifies all training examples, then we say that the training set is **linearly separable**.
- Different Perceptron learning techniques are available.

Learning a linear classifier

- A linear classifier can be learnt even when the data is NOT a linearly separable one.
- We can define a criterion J and try to minimize this which corresponds to the learnt linear classifier.
- A linear regression method can be directly used.

Perceptron – Let us begin with linearly separable data

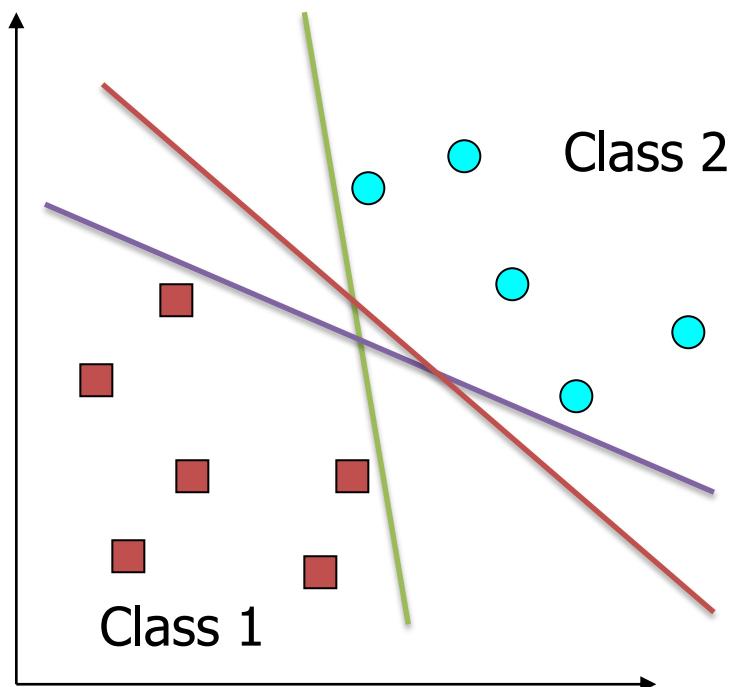
- If the data is linearly separable then many Perceptrons are possible that correctly classifies the training set.



All being doing equally good on training set, which one is good on the unseen test set?

Perceptron – Let us begin with linearly separable data

- If the data is linearly separable then many Perceptrons are possible that correctly classifies the training set.

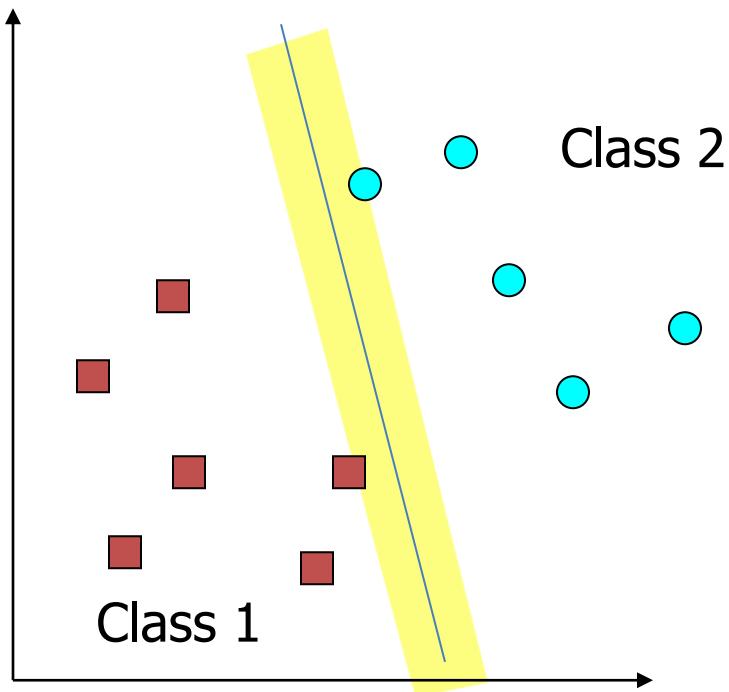
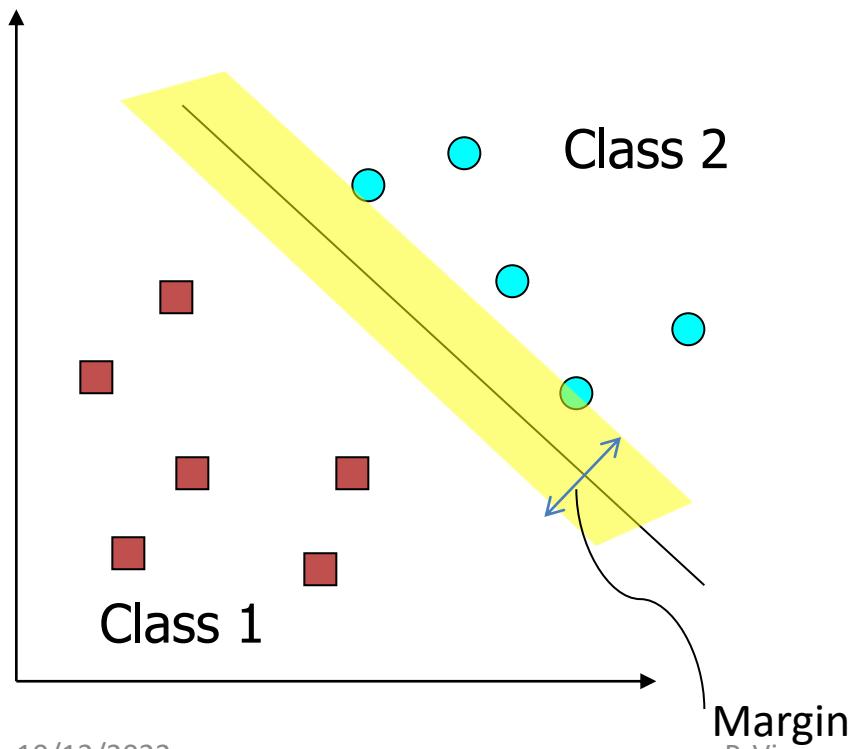


All being doing equally good on training set, which one is good on the unseen test set?

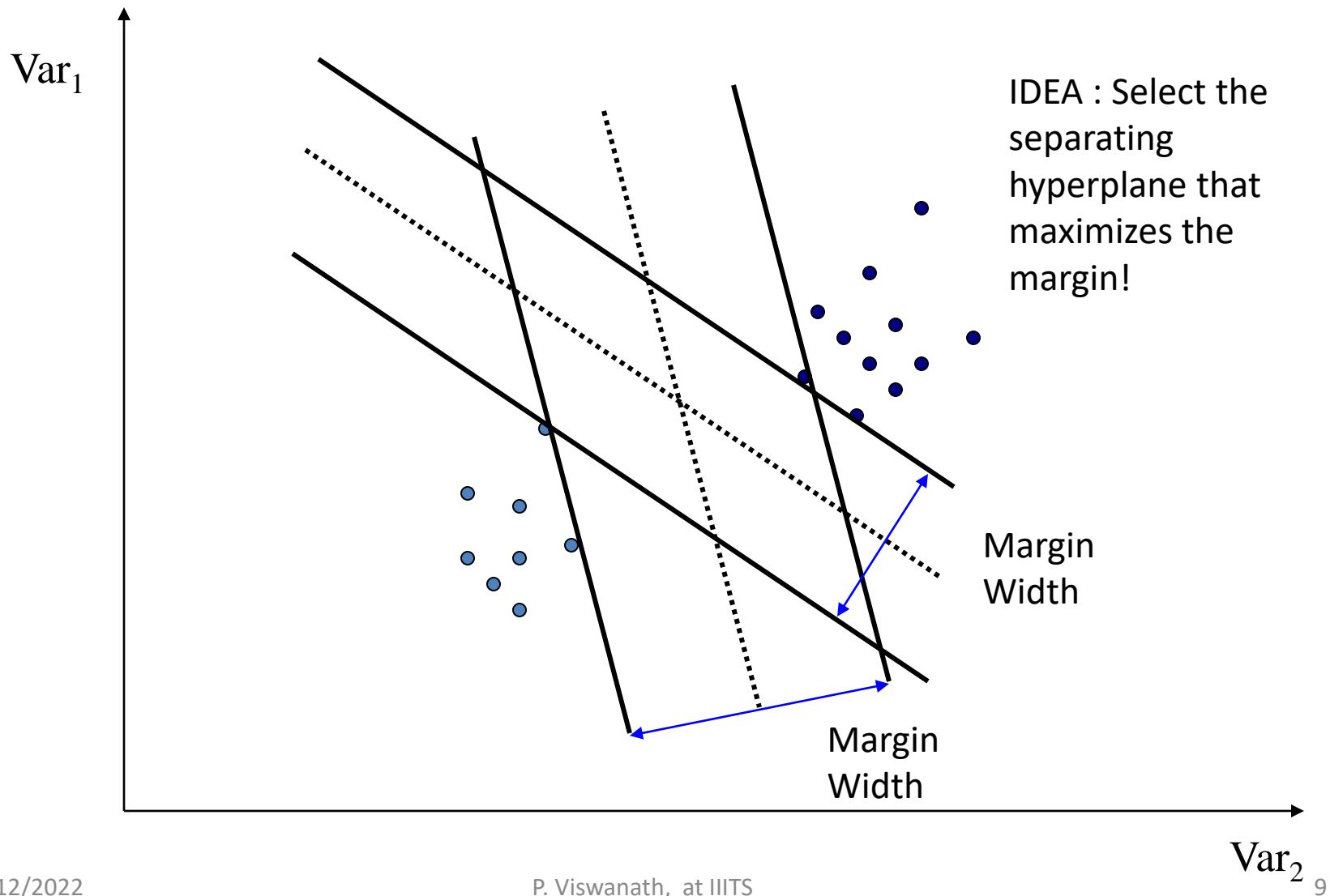
The solution we are getting depends on the starting point i.e., the parameter vector we chose to begin the gradient descent like method.

Hard Linear SVM

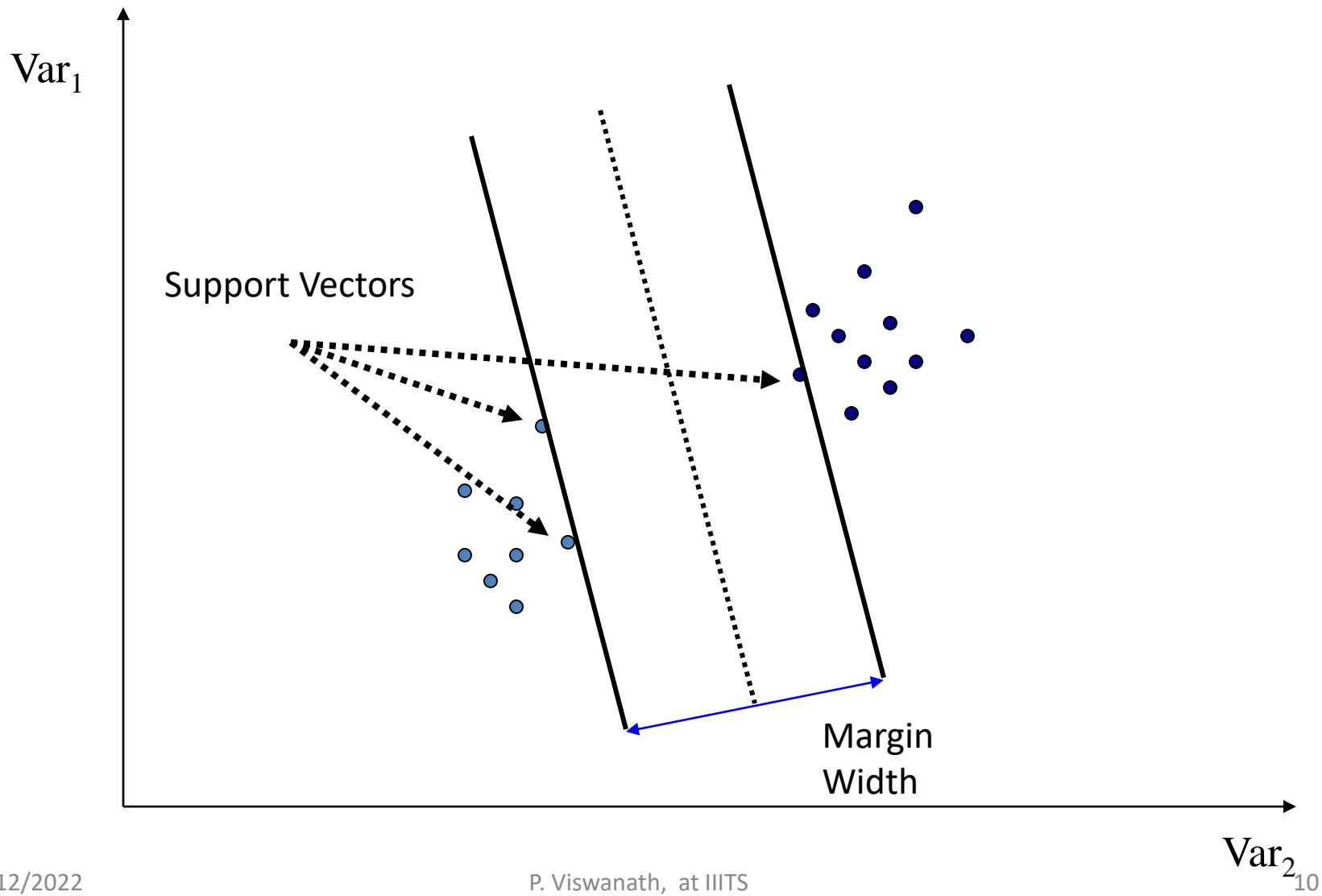
- The best perceptron for a linearly separable data is called “hard linear SVM”.
- For each linear function we can define its margin.
- That linear function which has the maximum margin is the best one.



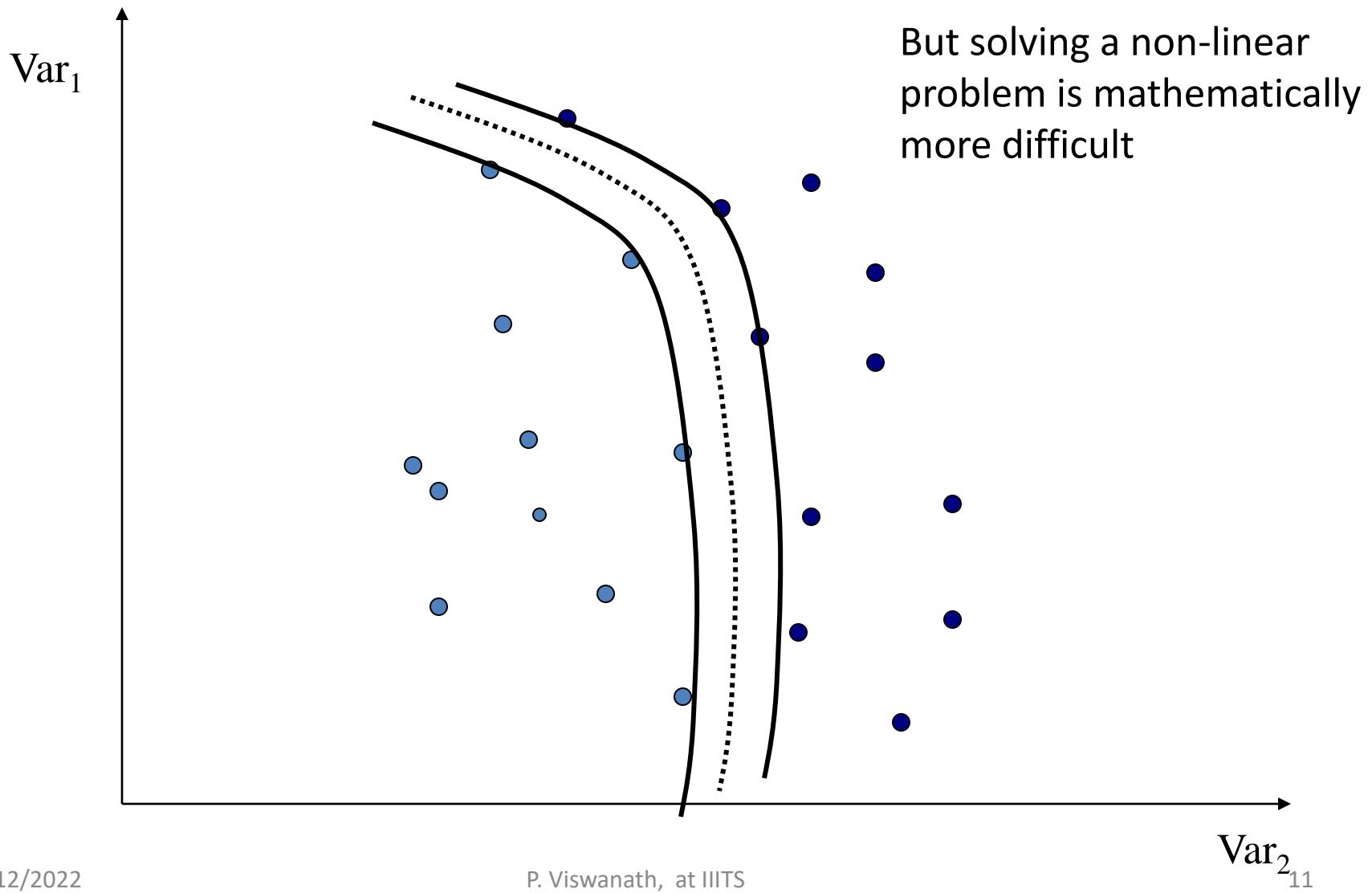
Maximizing the Margin



Support Vectors

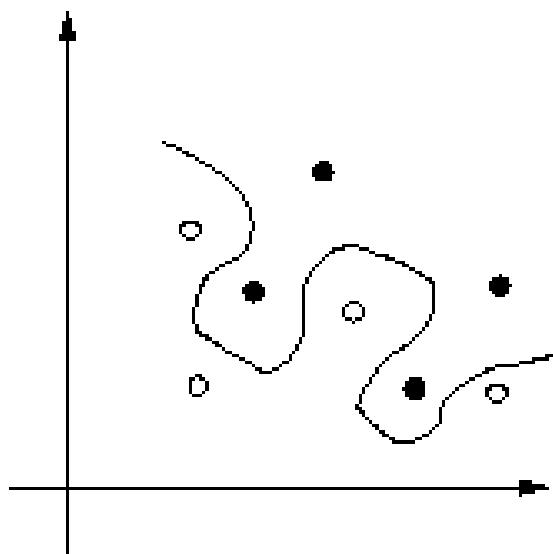


What if the data is not linearly separable?



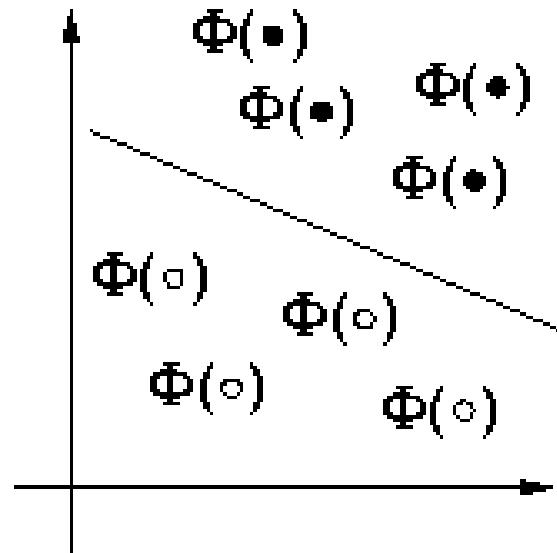
Kernel Mapping

Data space X



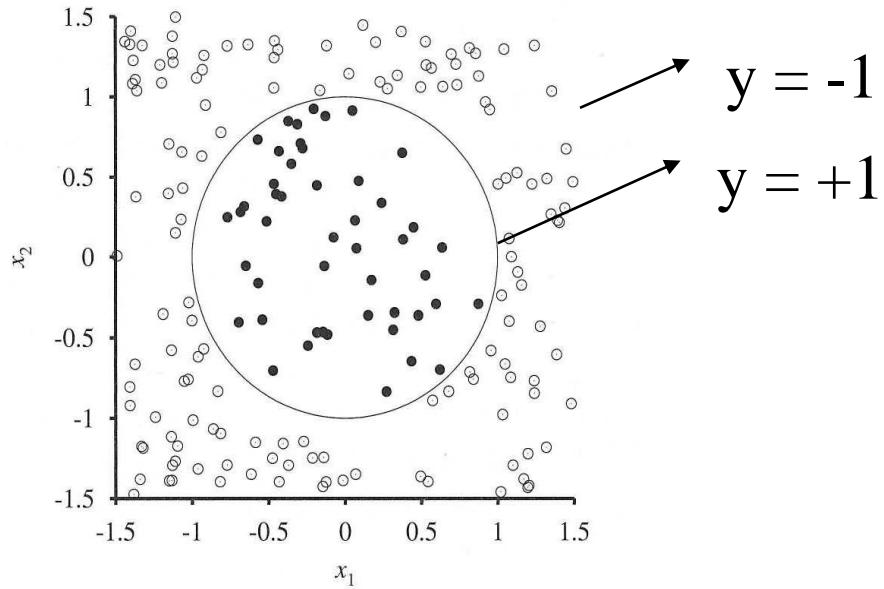
Φ

Feature Space F

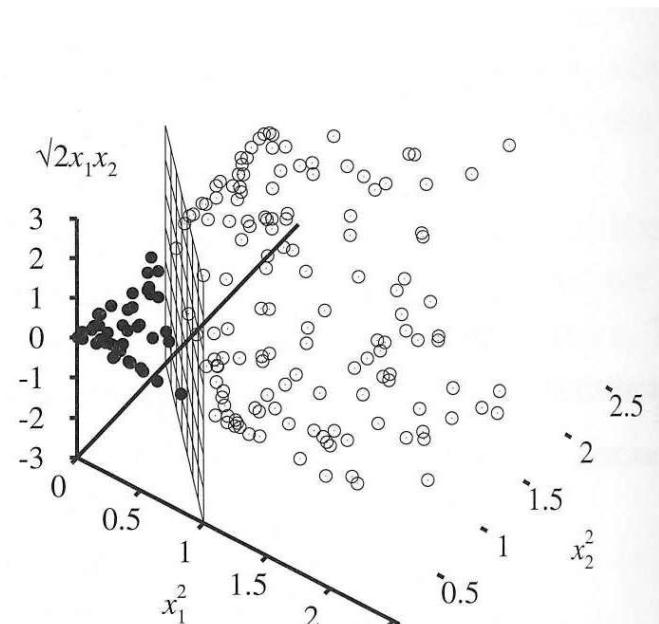


An example

Input Space



Feature Space



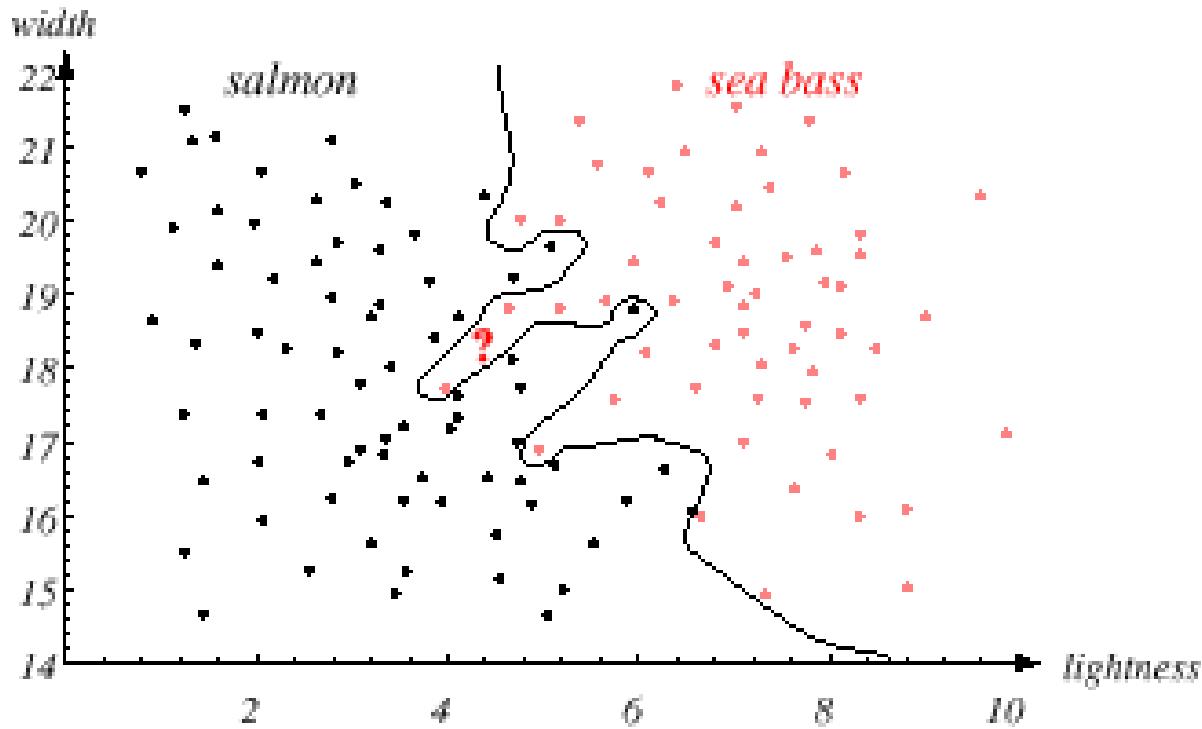
The Trick !!

- There is no need to do this mapping explicitly.
- For some mappings, the dot product in the feature space can be expressed as a function in the Input space.
- $\varphi(X_1) \cdot \varphi(X_2) = k(X_1, X_2)$

Eg: Consider a two dimensional problem with $X = (x_1, x_2)^t$. Let $\phi(X) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^t$. Then $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j) = (X_i \cdot X_j)^2$.

- So, if the solution is going to involve only dot products then it can be solved using kernel trick (of course, appropriate kernel function has to be chosen).
- The problem is, with powerful kernels like “Gaussian kernel” it is possible to learn a non-linear classifier which does extremely well on the training set.

Discriminant functions: non-linear



This makes zero mistakes with the training set.

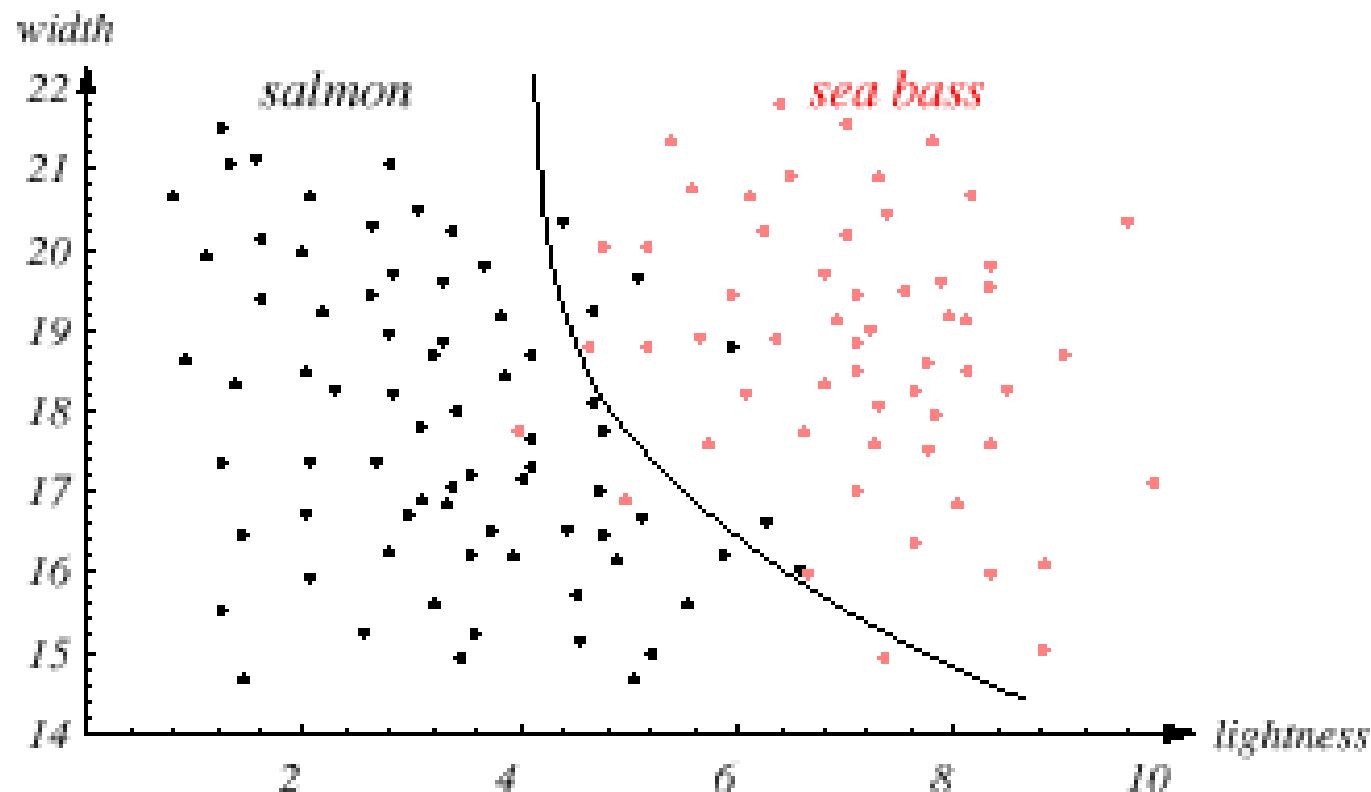
Other important issues ...

- This classifier is doing very well as for the training data is considered.
- But this does not guarantee that the classifier works well with a data element which is not there in the training set (that is, with unseen data).
- This is **overfitting** the classifier with the training data.
- May be we are respecting noise also (There might be mistakes while taking the measurements).
- The ability “to perform better with unseen test patterns too” is called the **generalization ability** of the classifier.

Generalization ability

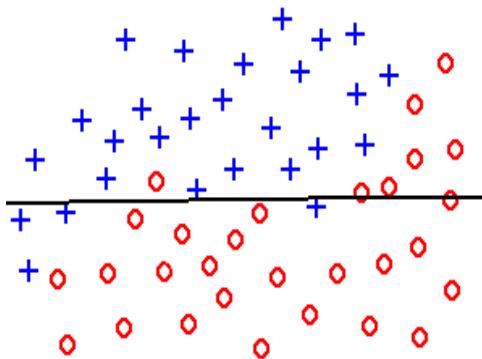
- This is discussed very much.
- It is argued that the simple one will have better generalization ability (eg: Occam's razor: **Between two solutions, if everything else is same then choose the simple one**).
- How to quantify this?
- (*Training error + a measure of complexity*) should be taken into account while designing the classifier.
- Support vector machines are proved to have better generalization ability.

Discriminant functions ...



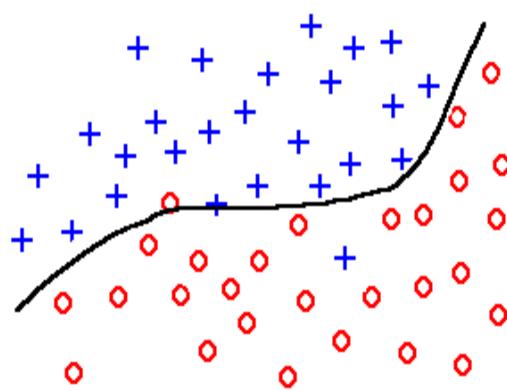
- This has some training error, but this is a relatively simple one.

Overfitting and underfitting



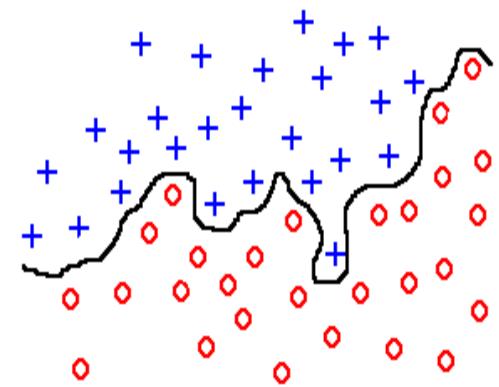
underfitting

High bias,
Low variance



good fit

Low bias,
Low variance



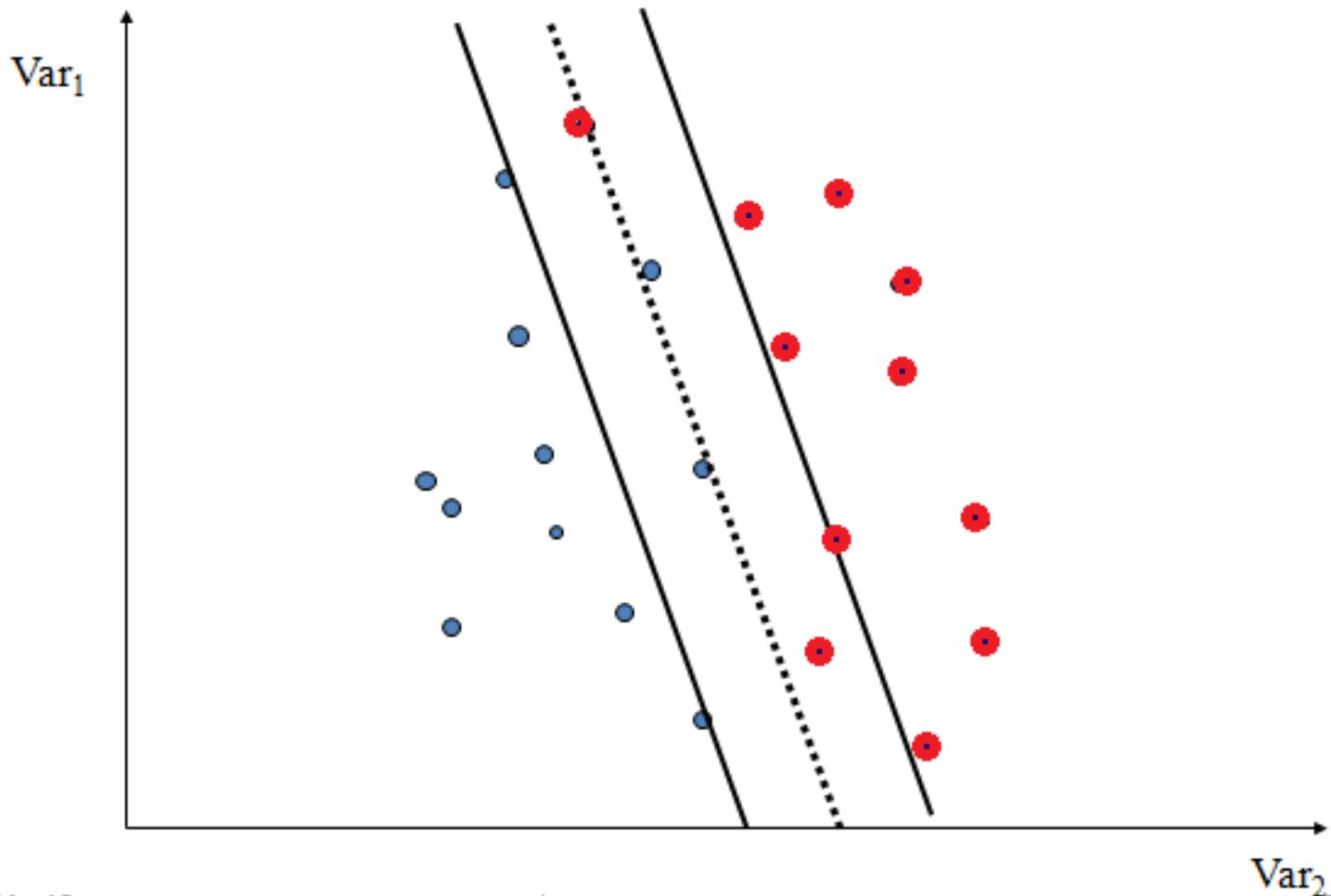
overfitting

Low bias,
High variance

Soft SVM

- Allow for some mistakes with the training set !
- But, this is to achieve a better margin.

Soft SVM



Support Vector Machines

(Formal : Version 1)

Linear Classifier

- Consider a two class problem, $\Omega = \{\omega_1, \omega_2\}$

A pattern $X = (x_1, \dots, x_d)^t$

- The discriminant function

$$g(X) = w_1x_1 + \dots + w_dx_d + w_0 = W^tX + w_0$$

- $g(X) = 0$ defines a hyperplane in the feature space.

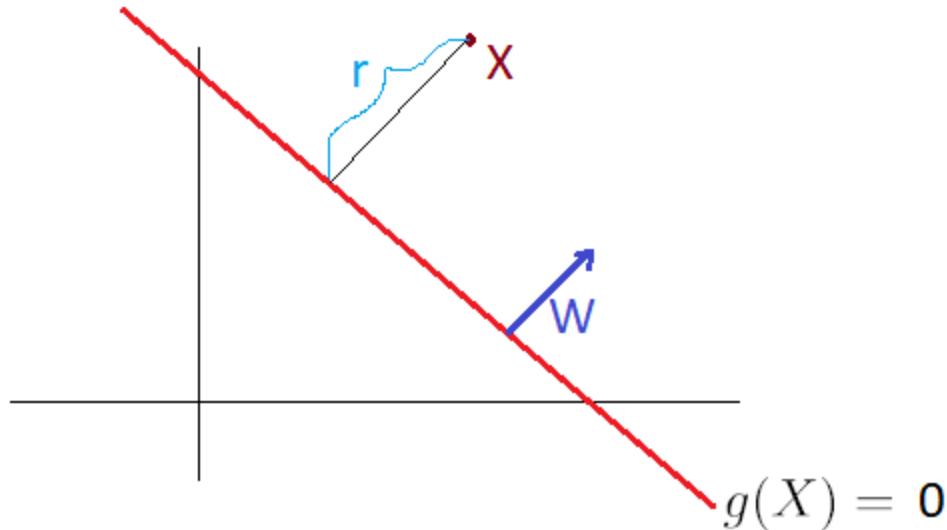
- Classification rule:

$$g(X) > 0 \Rightarrow \text{class is } \omega_1$$

$$g(X) < 0 \Rightarrow \text{class is } \omega_2$$

$$g(X) = 0 \Rightarrow \text{class is decided arbitrarily}$$

$$g(X) = w_1x_1 + \cdots + w_dx_d + w_0 = W^t X + w_0$$



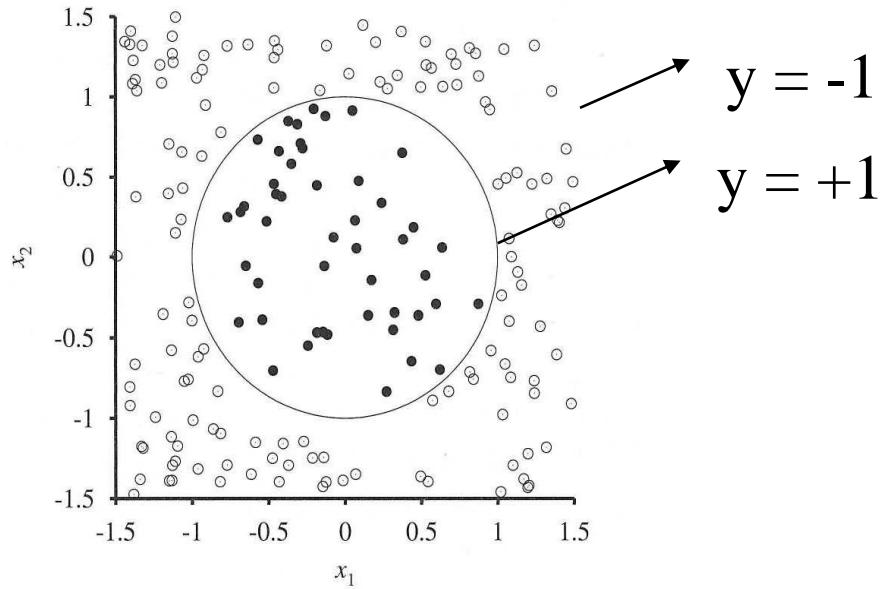
- W is perpendicular to $g(X) = 0$ hyper-plane.
- r , distance of arbitrary X from the hyper-plane
is: $r = \frac{g(X)}{\|W\|}$

Non-linear vs linear

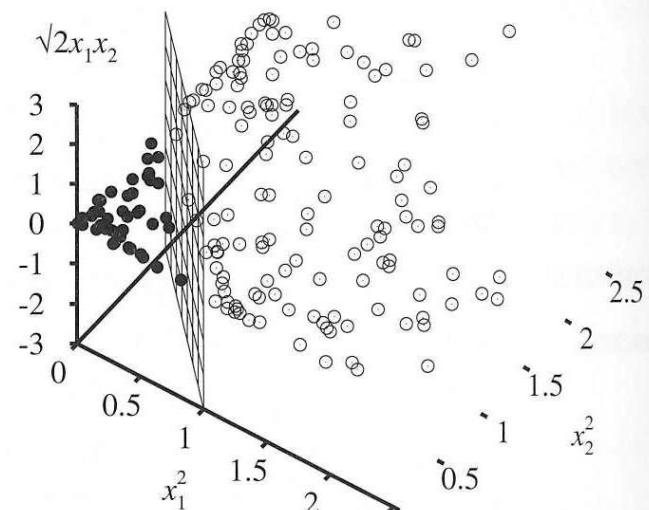
- For every function in the input space, there is an equivalent linear function in a feature space.
- Feature space is a high-dimensional one.
- A mathematically valid way to work (do the optimization) in the input space but to get the solution (linear function) in the feature space is by using the *kernel trick*.

An example

Input Space



Feature Space



The Kernel Trick !

- There is no need to do this mapping explicitly.
- For some mappings, the dot product in the feature space can be expressed as a function in the Input space.
- $\varphi(X_1) \cdot \varphi(X_2) = k(X_1, X_2)$

Eg: Consider a two dimensional problem with $X = (x_1, x_2)^t$. Let $\phi(X) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^t$. Then $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j) = (X_i \cdot X_j)^2$.

Support Vector Machines

- Decision surface is a hyperplane (line in 2D) in **feature** space (similar to the Perceptron)
- Arguably, one of the most important discovery in machine learning
- In a nutshell:
 - map the data to a predetermined very high-dimensional space via a kernel function
 - Find the hyperplane that maximizes the margin between the two classes
 - If data are not separable find the hyperplane that maximizes the margin and minimizes the (penalty associated with) misclassifications

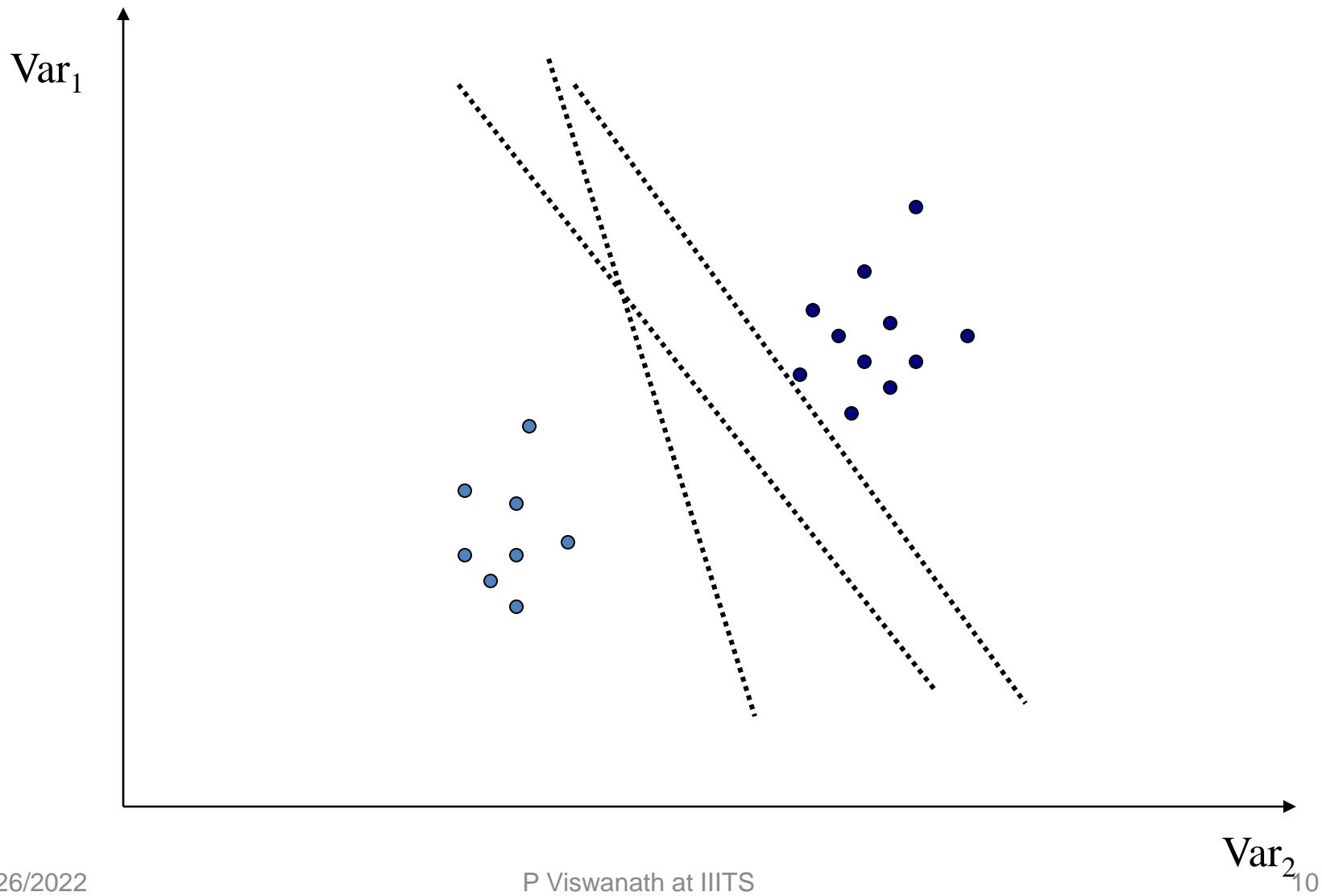
Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

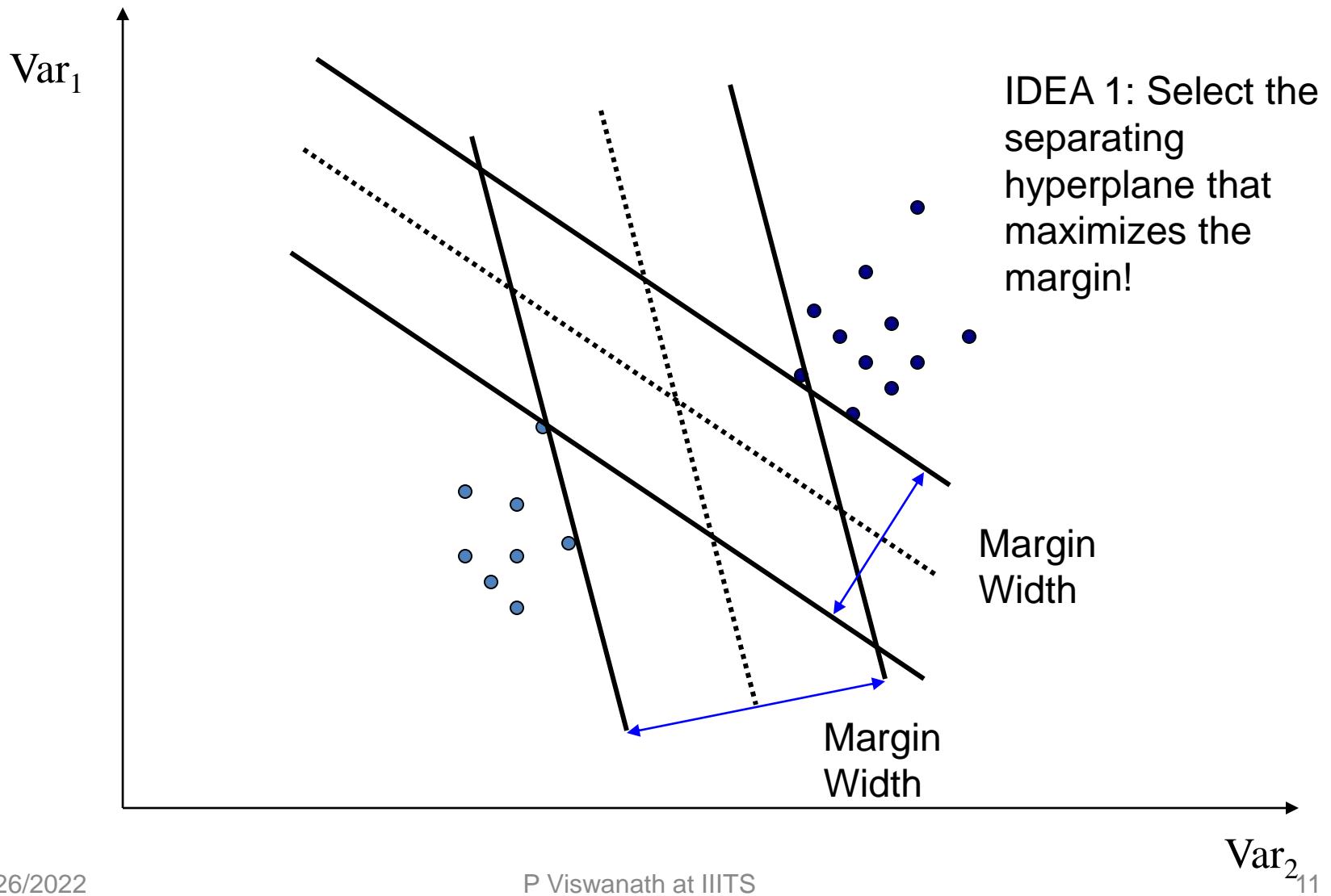
Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

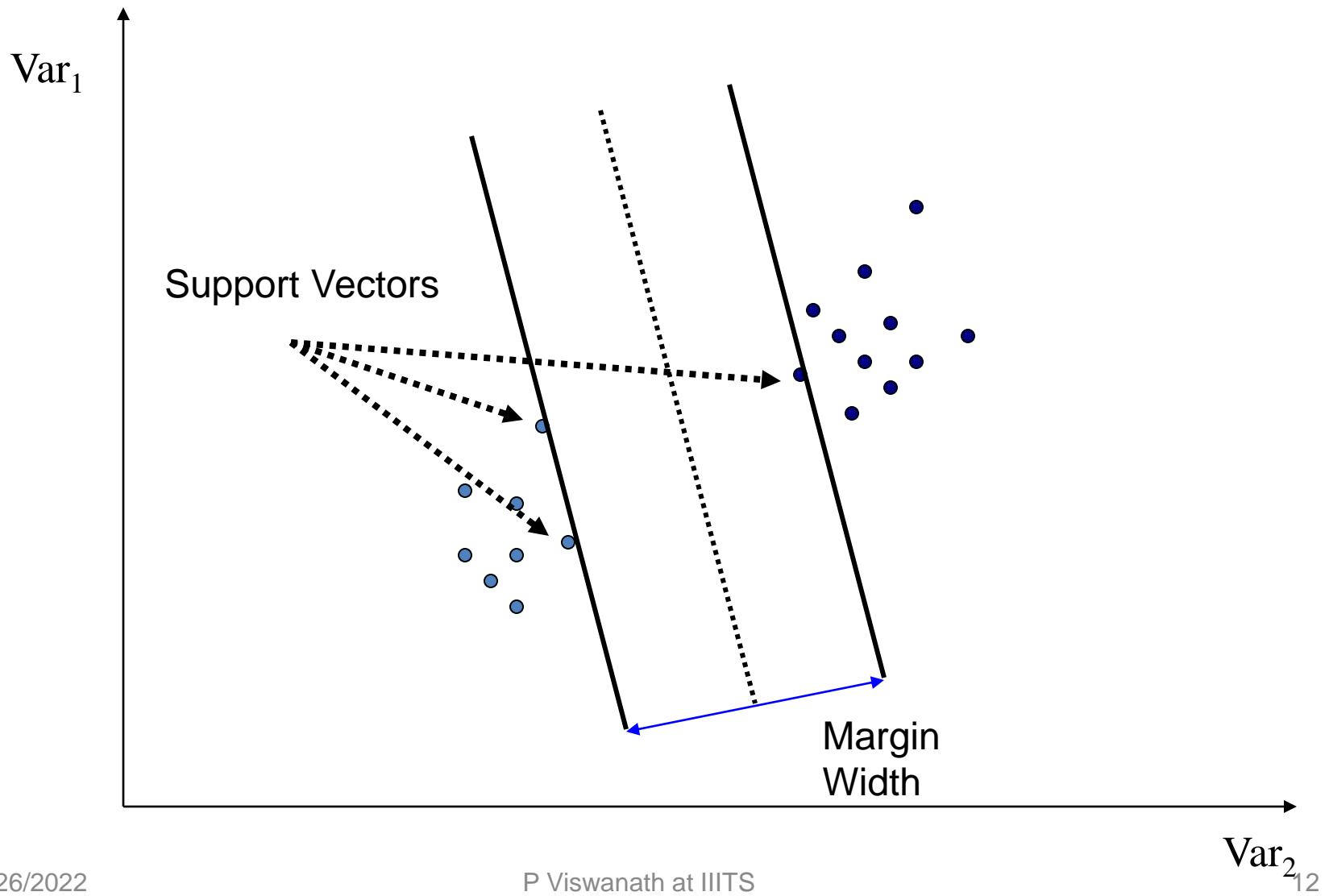
Which Separating Hyperplane to Use?



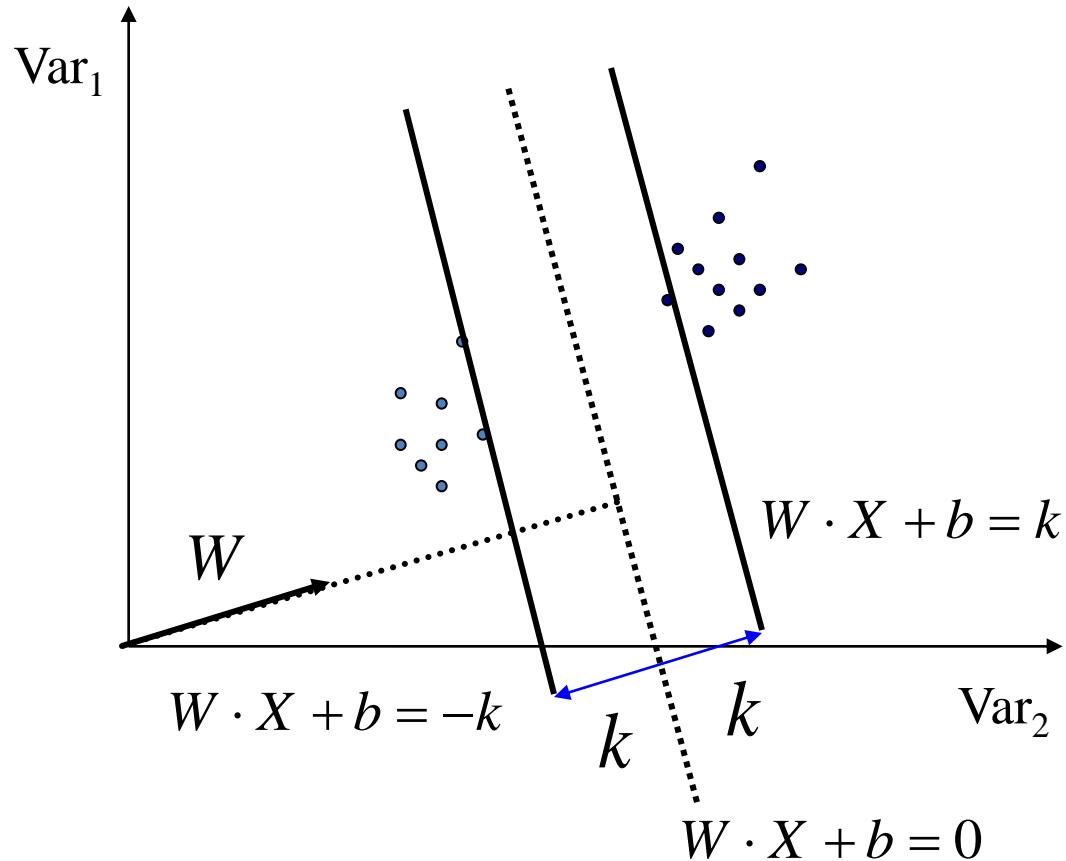
Maximizing the Margin



Support Vectors



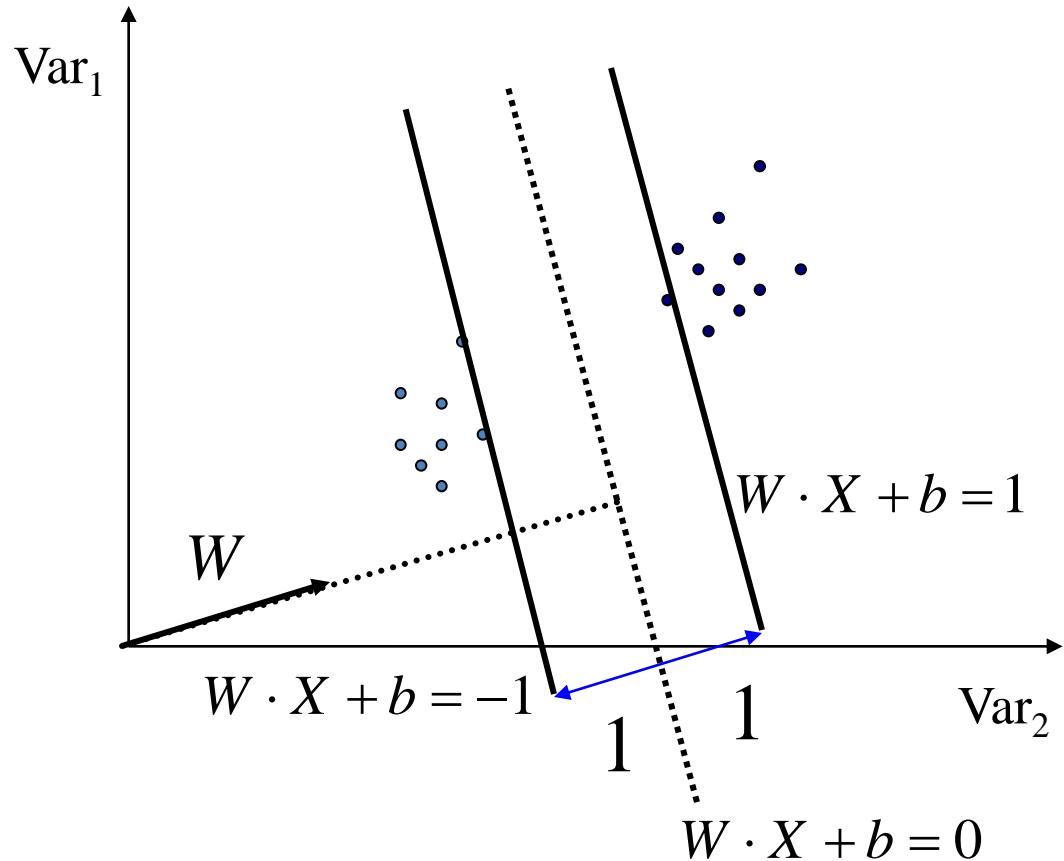
Setting Up the Optimization Problem



So, the problem is:

$$\begin{aligned} &\text{Maximize } \frac{2|k|}{\|W\|} \\ \text{s.t. } &W \cdot X + b \geq k, \forall X \text{ in Class 1} \\ &W \cdot X + b \leq -k, \forall X \text{ in Class 2} \end{aligned}$$

Setting Up the Optimization Problem



There is a scale and unit for data so that $k=1$. Then problem becomes:

So, the problem is:

$$\begin{aligned} & \text{Maximize } \frac{2}{\|W\|} \\ \text{s.t. } & W \cdot X + b \geq 1, \forall X \text{ in Class 1} \\ & W \cdot X + b \leq -1, \forall X \text{ in Class 2} \end{aligned}$$

Setting Up the Optimization Problem

- If class 1 corresponds to 1 and class 2 corresponds to -1, we can rewrite

$$\begin{aligned} & . \quad W \cdot X_i + b \geq 1, \forall X_i \text{ with } y_i = 1 \\ & W \cdot X_i + b \leq -1, \forall X_i \text{ with } y_i = -1 \end{aligned}$$

- as

$$. \quad y_i(W \cdot X_i + b) \geq 1, \forall X_i$$

- So the problem becomes:

$$\begin{aligned} & \text{Maximize } \frac{2}{\|W\|} \\ & \text{s.t. } y_i(W \cdot X_i + b) \geq 1, \forall X_i \end{aligned}$$

or

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|W\|^2 \\ & \text{s.t. } y_i(W \cdot X_i + b) \geq 1, \forall X_i \end{aligned}$$

Linear, Hard-Margin SVM Formulation

- Find W, b that solves

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \|W\|^2 \\ & \text{s.t. } y_i(W \cdot X_i + b) \geq 1, \forall X_i \end{aligned}$$

- Problem is convex so, there is a unique global minimum value (when feasible)
- Non-solvable if the data is not linearly separable
- Quadratic Programming
 - Very efficient computationally with modern constraint optimization engines (handles thousands of constraints).

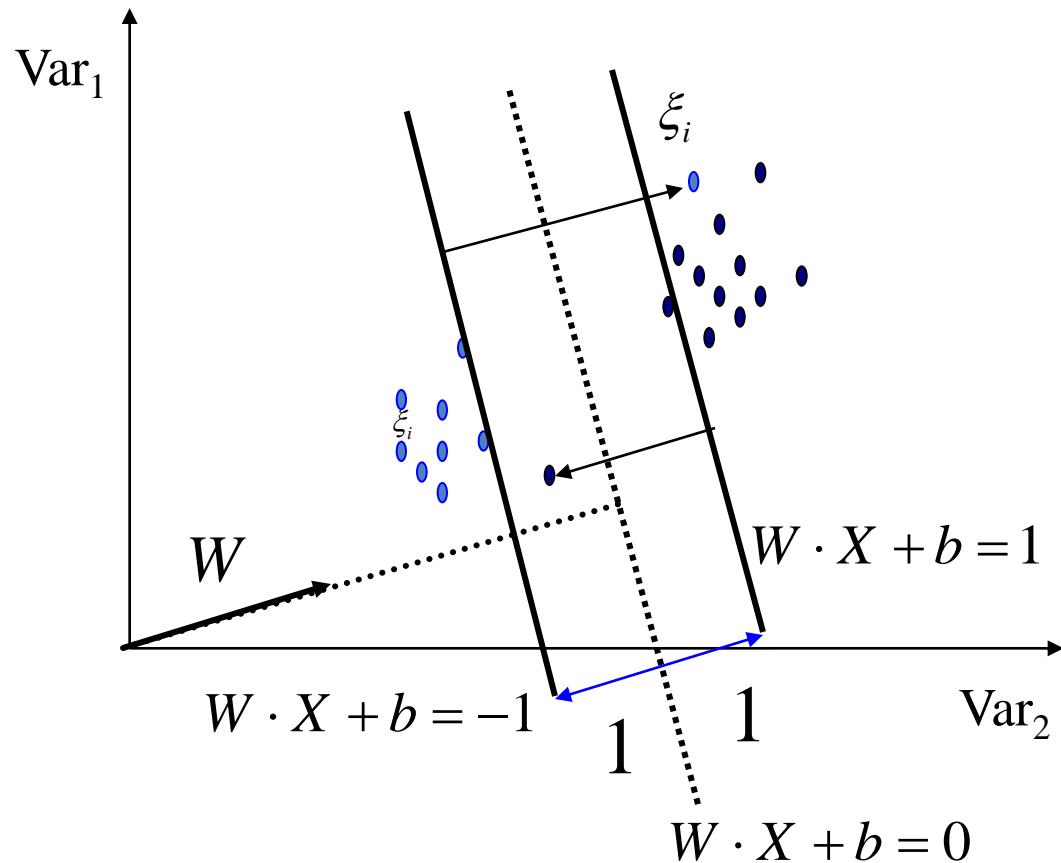
Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

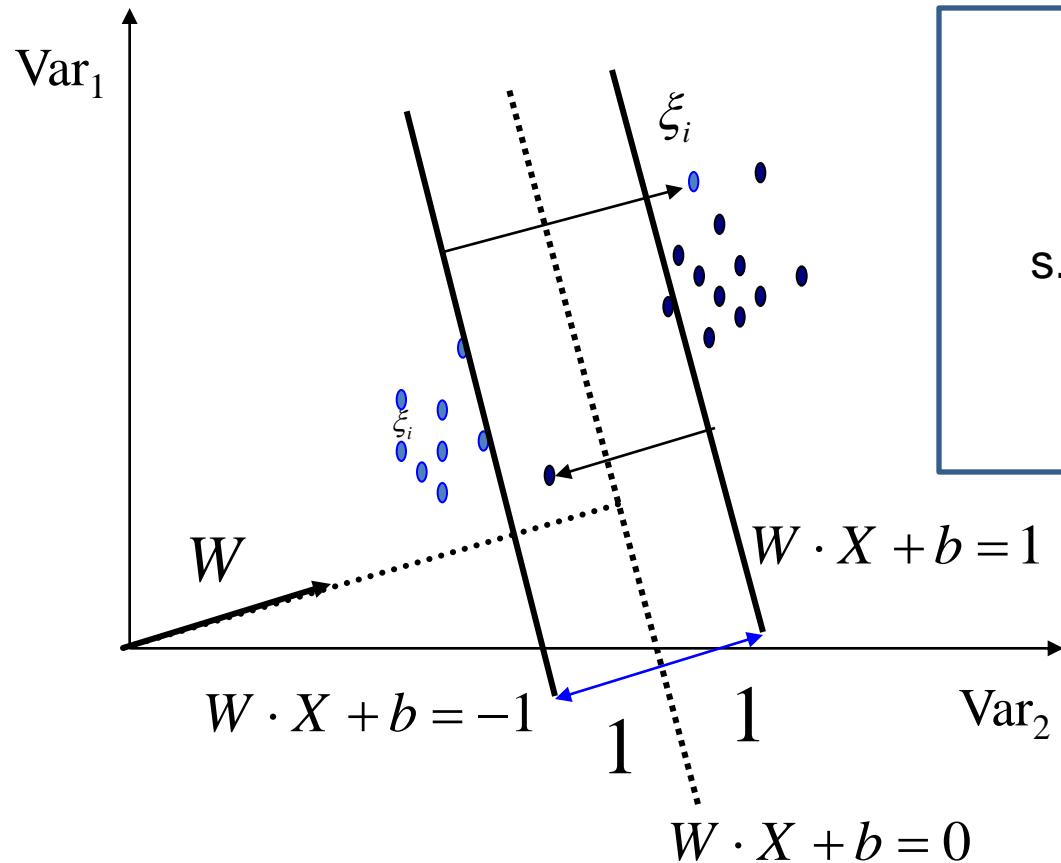
Non-Linearly Separable Data



Introduce slack
variables ξ_i

Allow some
instances to fall
within the margin,
but penalize them

Non-Linearly Separable Data



$$\text{Min} \frac{1}{2} \|W\|^2 + C \sum_i \xi_i$$

$$\text{s.t. } y_i(W \cdot X_i + b) \geq 1 - \xi_i, \forall X_i$$

$$\text{and } \xi_i \geq 0$$

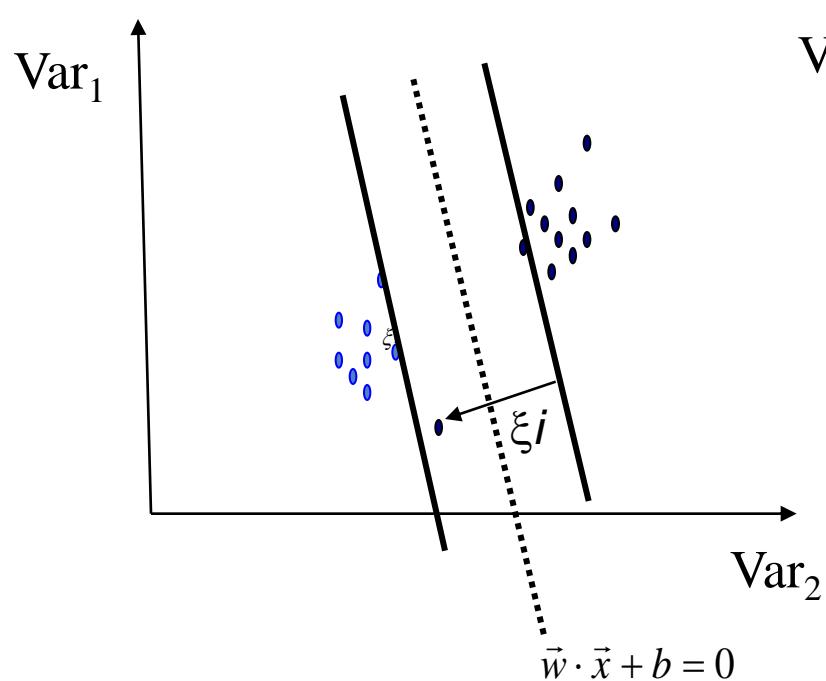
C trades-off margin width
and misclassifications

Linear, Soft-Margin SVMs

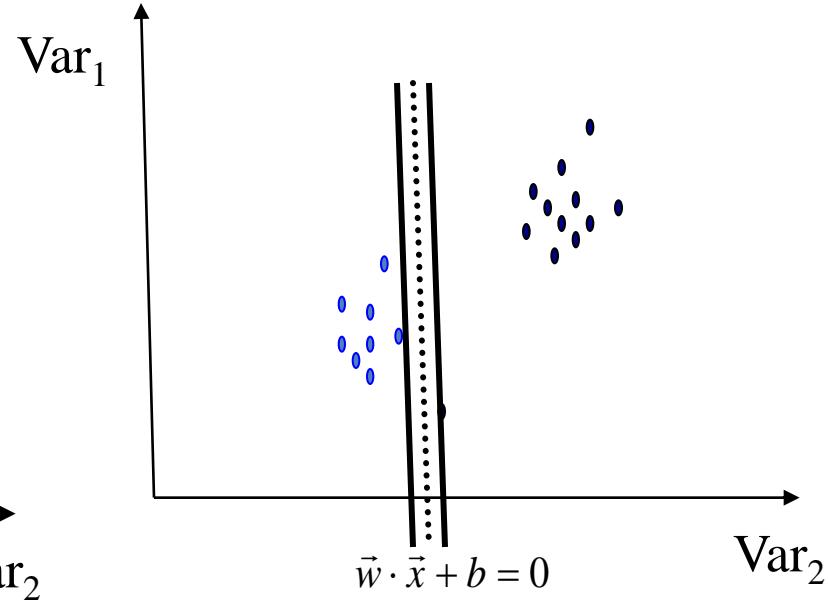
$$\min \frac{1}{2} \|w\|^2 + C \sum_i \xi_i \quad \begin{aligned} & y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall x_i \\ & \xi_i \geq 0 \end{aligned}$$

- Algorithm tries to maintain ξ_i to zero while maximizing margin
- Notice: algorithm does not minimize the *number* of misclassifications (NP-complete problem) but the sum of distances from the margin hyperplanes
- Other formulations use ξ_i^2 instead
- As $C \rightarrow \infty$, we get closer to the hard-margin solution

Robustness of Soft vs Hard Margin SVMs



Soft Margin SVM



Hard Margin SVM

Soft vs Hard Margin SVM

- Soft-Margin always have a solution
- Soft-Margin is more robust to outliers
 - Smoother surfaces (in the non-linear case)
- Hard-Margin does not require to guess the cost parameter (requires no parameters at all)

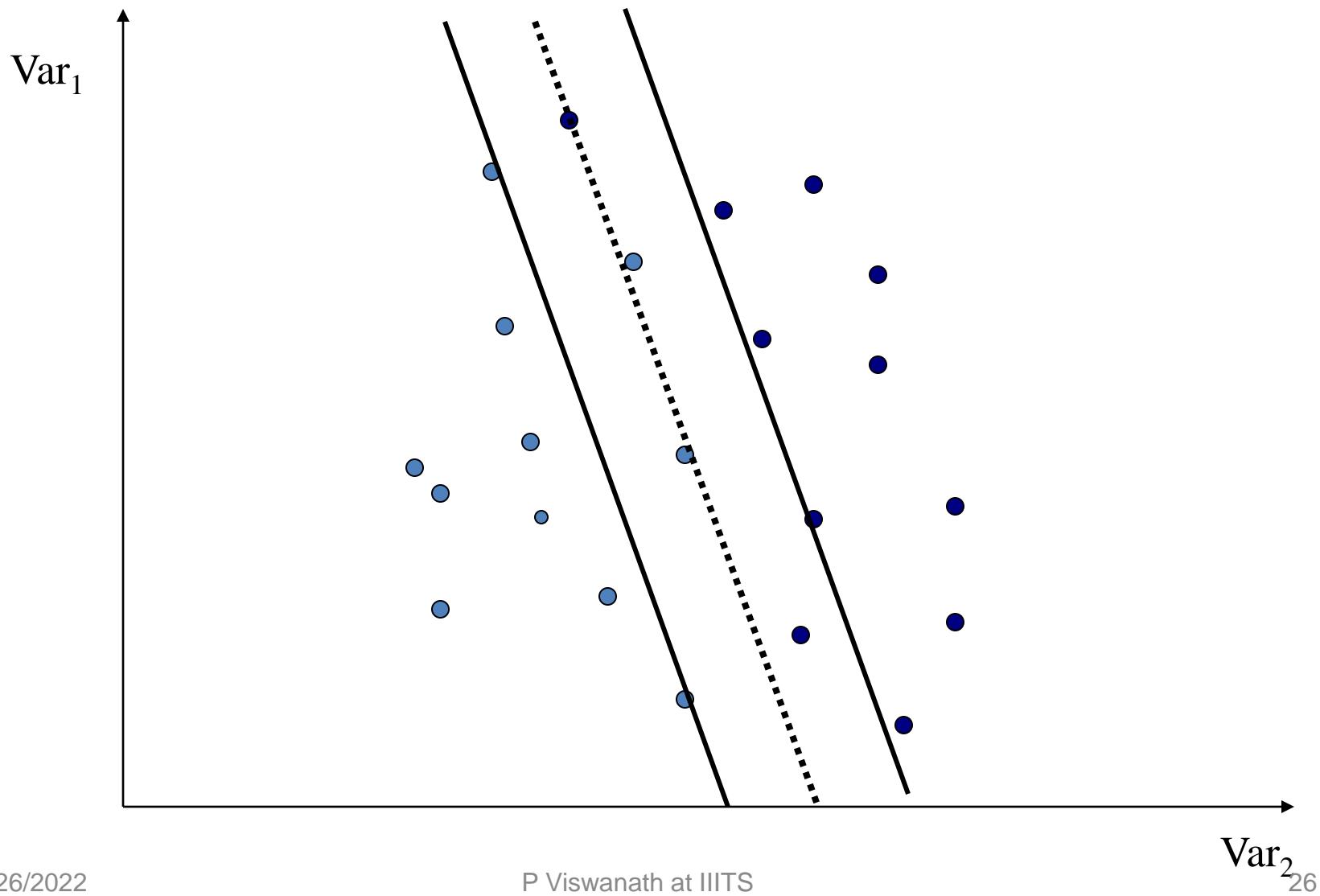
Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

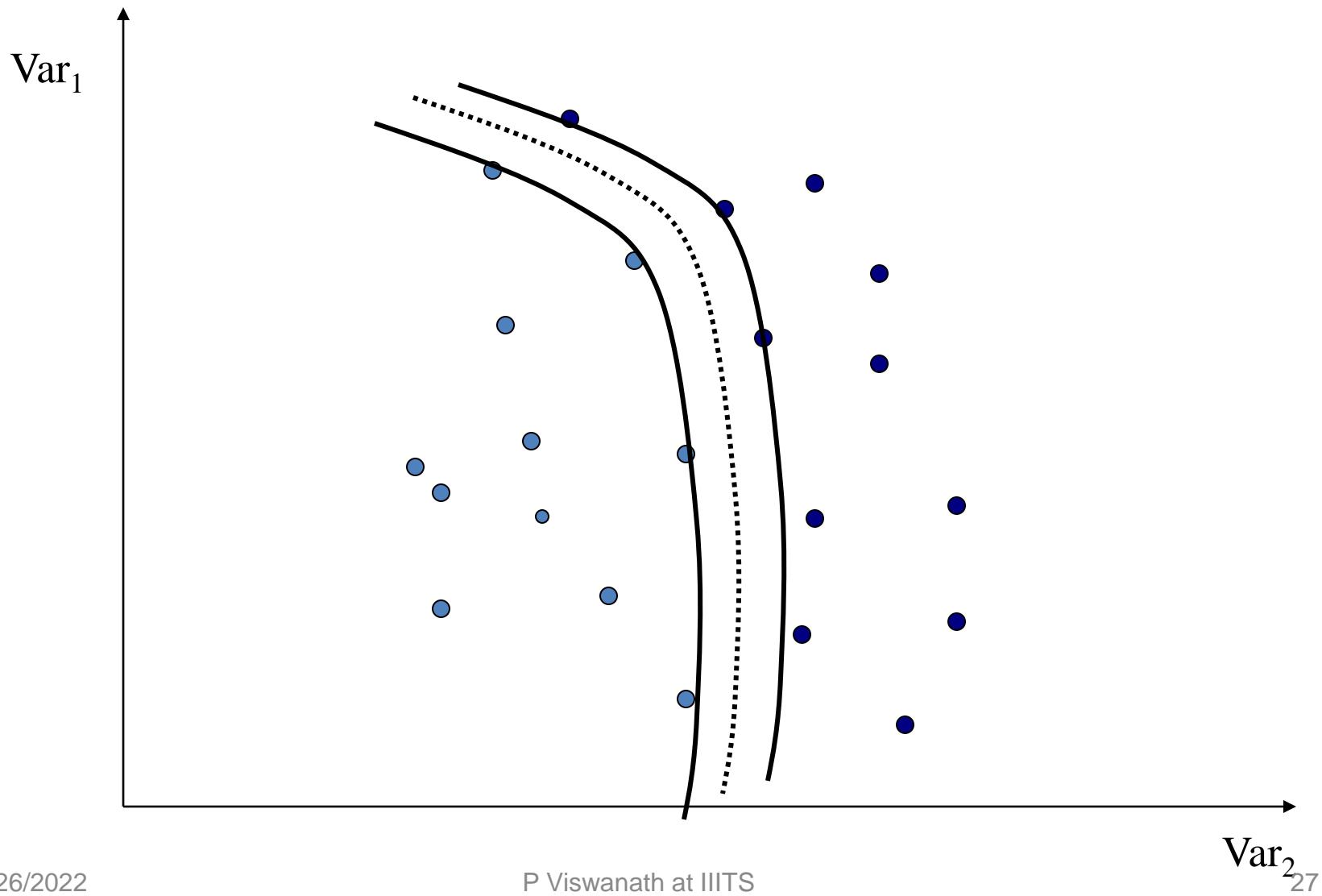
Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

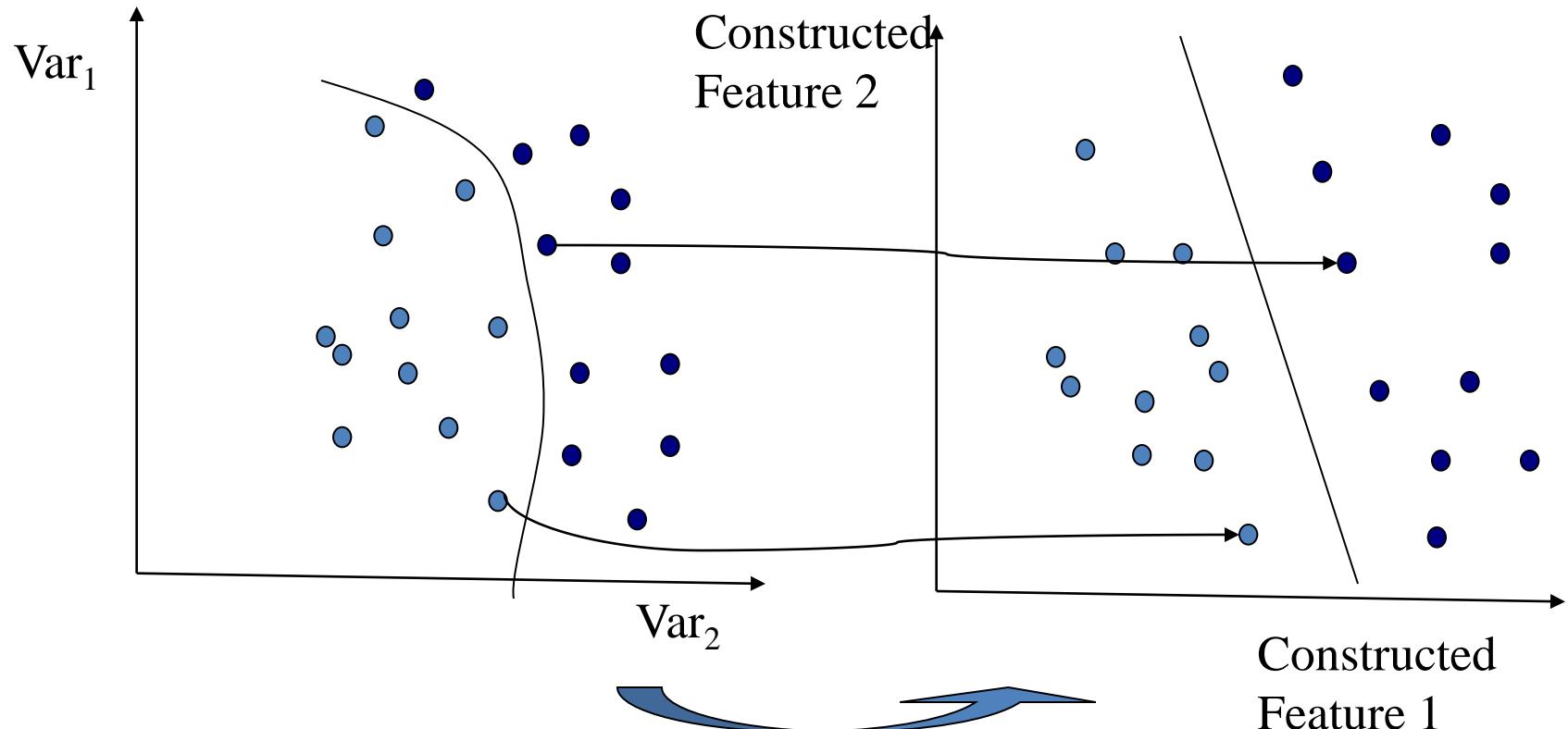
Disadvantages of Linear Decision Surfaces



Advantages of Non-Linear Surfaces



Linear Classifiers in High-Dimensional Spaces



Find function $\Phi(x)$ to map to
a different space

Mapping Data to a High-Dimensional Space

- Find function $\Phi(x)$ to map to a different space, then SVM formulation becomes:
- $$\min \frac{1}{2} \|W\|^2 + C \sum_i \xi_i \quad \begin{aligned} s.t. \quad & y_i(W \cdot \Phi(X) + b) \geq 1 - \xi_i, \forall X_i \\ & \xi_i \geq 0 \end{aligned}$$
- Data appear as $\Phi(X)$, weights W are now weights in the new space
- Explicit mapping expensive if $\Phi(X)$ is very high dimensional
- Solving the problem without explicitly mapping the data is desirable

The Dual of the SVM Formulation

- Original SVM formulation
 - n inequality constraints
 - n positivity constraints
 - n number of ξ variables

$$\min_{W,b} \frac{1}{2} \|W\|^2 + C \sum_i \xi_i$$

$$\text{s.t. } y_i (W \cdot \Phi(X) + b) \geq 1 - \xi_i, \forall X_i \\ \xi_i \geq 0$$

- The (Wolfe) dual of this problem
 - one equality constraint
 - n positivity constraints
 - n number of α variables (Lagrange multipliers)
 - Objective function more complicated
- NOTICE: Data only appear as $\Phi(X_i) \cdot \Phi(X_j)$

$$\min_{\alpha_i} \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\Phi(X_i) \cdot \Phi(X_j)) - \sum_i \alpha_i$$

$$\text{s.t. } \mathbf{C} \geq \alpha_i \geq 0, \forall X_i \\ \sum_i \alpha_i y_i = 0$$

The Kernel Trick

- $\Phi(x_i) \cdot \Phi(x_j)$: means, map data into new space, then take the inner product of the new vectors
- We can find a function such that: $K(X_i, X_j) = \Phi(X_i) \cdot \Phi(X_j)$, i.e., the image of the inner product of the data is the inner product of the images of the data
- Then, we do not need to explicitly map the data into the high-dimensional space to solve the optimization problem (for training)
- How do we classify without explicitly mapping the new instances?
Turns out

$$\text{sgn}(W \cdot X + b) = \text{sgn}\left(\sum_i \alpha_i y_i K(X_i, X) + b\right)$$

where b solves $\alpha_j(y_j \sum_i \alpha_i y_i K(X_i, X_j) + b - 1) = 0$,

for any j with $\alpha_j \neq 0$

Examples of Kernels

- Assume we measure two quantities

- Consider the function:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1x_2, x_1, x_2, 1)$$

- We can verify that:

$$K(X_1, X_2) = (X_1 \cdot X_2 + 1)^2$$

- These type of kernels are called Polynomial kernels.

Polynomial and Gaussian Kernels

$$K(X \cdot Z) = (X \cdot Z + 1)^p$$

- is called the polynomial kernel of degree p .
- Another commonly used Kernel is the Gaussian (maps to an infinite dimensional space):

$$K(X \cdot Z) = \exp(-\|X - Z\|^2 / 2\sigma^2)$$

The Mercer Condition

- Is there a mapping $\Phi(x)$ for any given symmetric function $K(x,z)$? **No.**
- The SVM dual formulation requires calculation $K(x_i, x_j)$ for each pair of training instances. The matrix $G_{ij} = K(x_i, x_j)$ is called the Gram matrix
- There is a feature space $\Phi(x)$ when the Kernel is such that G is always semi-positive definite (Mercer condition)

Support Vector Machines

- Three main ideas:
 1. Define what an optimal hyperplane is (in way that can be identified in a computationally efficient way): maximize margin
 2. Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications
 3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

Other Types of Kernel Methods

- SVMs that perform regression
- SVMs that perform clustering
- ν-Support Vector Machines: maximize margin while bounding the number of margin errors
- Leave One Out Machines: minimize the bound of the leave-one-out error
- SVM formulations that take into consideration difference in cost of misclassification for the different classes
- Kernels suitable for sequences of strings, or other specialized kernels
- Kernel-PCA, Kernel-SVD

Comparison with Neural Networks

Neural Networks

- Hidden Layers map to lower dimensional spaces
- Search space has multiple local minima
- Training is expensive
- Classification extremely efficient
- Requires number of hidden units and layers
- Very good accuracy in typical domains

SVMs

- Kernel maps to a very-high dimensional space
- Search space has a unique minimum
- Training is extremely efficient
- Classification extremely efficient
- Kernel and cost the two parameters to select
- Very good accuracy in typical domains
- Extremely robust

MultiClass SVMs

- One-versus-all
 - Train n binary classifiers, one for each class against all other classes.
 - Predicted class is the class of the most confident classifier
- One-versus-one
 - Train $n(n-1)/2$ classifiers, each discriminating between a pair of classes
 - Several strategies for selecting the final classification based on the output of the binary SVMs
- Truly MultiClass SVMs
 - Generalize the SVM formulation to multiple categories

Conclusions

- SVMs express learning as a mathematical program taking advantage of the rich theory in optimization
- SVM uses the kernel trick to map indirectly to extremely high dimensional spaces
- SVMs extremely successful, robust, efficient, and versatile while there are good theoretical indications as to why they generalize well

Suggested Further Reading

- <http://www.kernel-machines.org/tutorial.html>
- C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
- P.H. Chen, C.-J. Lin, and B. Schölkopf. A tutorial on nu -support vector machines. 2003.
- N. Cristianini. ICML'01 tutorial, 2001.
- K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181-201, May 2001.
- B. Schölkopf. SVM and kernel methods, 2001. Tutorial given at the NIPS Conference.
- Hastie, Tibshirani, Friedman, The Elements of Statistical Learning, Springel 2001

Appendix

Linear SVM

- We like to draw two hyperplanes such that

$$W \cdot X_i + b \geq 1 \quad \text{if } y_i = +1$$

$$W \cdot X_i + b \leq -1 \quad \text{if } y_i = -1$$

$$y_i(W \cdot X_i + b) \geq 1 \quad \text{for all } i, \quad \text{OR}$$

$$1 - y_i(W \cdot X_i + b) \leq 0 \quad \dots \quad \dots \quad \dots \quad (1)$$

- The two parallel hyperplanes are

$$H_1 : W \cdot X + b = 1$$

$$H_2 : W \cdot X + b = -1$$

Linear SVM

- Distance between origin and H_1 is $\frac{b-1}{\|W\|}$
- Distance between origin and H_2 is $\frac{b+1}{\|W\|}$
- Then, the margin $= \frac{b-1}{\|W\|} - \frac{b+1}{\|W\|}$
 $= \frac{2}{\|W\|}$
- Then the problem is :
Minimize $\frac{1}{2}\|W\|^2$ (Objective function)
Subject to constraints: $1 - y_i(W \cdot X_i + b) \leq 0$, for all i
- Note that the objective is *convex* and the constraints are *linear*
- Lagrangian method can be applied.

Constrained Optimization Problem

- Minimize $f(v)$
Subject to the constraints $g_j(v) \leq 0, 1 \leq j \leq n.$
- Lagrangian,

$$\mathcal{L} = f(v) + \sum_{j=1}^n \alpha_j g_j(v)$$

where v is called *primary* variables and α_j are the Lagrangian multipliers which are also called *dual* variables.

- \mathcal{L} has to be minimized with respect to primal variables and maximized with respect to dual variables.

Constrained Optimization Problem

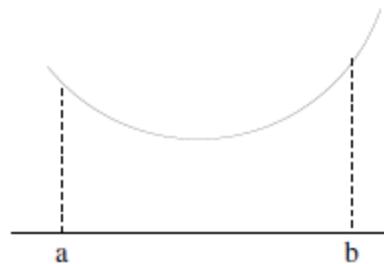
- The K.K.T (Karush-Kuhn-Tucker) conditions “necessary” at optimal v are:
 1. $\nabla_v \mathcal{L} = 0$
 2. $\alpha_j \geq 0$, for all $j = 1, \dots, n$
 3. $\alpha_j g_j(v) = 0$, for all $j = 1, \dots, n$
- If $f(v)$ is convex and $g_j(v)$ is linear for all j , then it turns out that K.K.T conditions are “necessary and sufficient” for the optimal v .

Convex Function

- A real valued function f defined in (a, b) is said to be convex if

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

for $a < x, y < b$, and $0 < \lambda < 1$



- This definition can be extended to functions in higher dimensional spaces.

Activate

Lagrangian

- Minimize $\frac{1}{2}||W||^2$ (Objective function)
- Subject to constraints: $1 - y_i(W \cdot X_i + b) \leq 0$, for all i
- Lagrangian,

$$\mathcal{L}(W, b, \alpha) = \frac{1}{2}||W||^2 + \sum_i \alpha_i [1 - y_i(W \cdot X_i + b)]$$

- Here,

$$\alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix}$$

Activate WiFi
Go to Settings 

K.K.T. Conditions

- $\nabla_W \mathcal{L} = W - \sum_i \alpha_i y_i X_i = 0$
 $\Rightarrow W = \sum_i \alpha_i y_i X_i \quad \cdots \cdots \cdots \rightarrow (1)$
- $\frac{\partial \mathcal{L}}{\partial b} = -\sum \alpha_i y_i = 0$
 $\Rightarrow \sum \alpha_i y_i = 0 \quad \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \rightarrow (2)$
- $\alpha_i \geq 0, \text{ for } i = 1 \text{ to } n \quad \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \rightarrow (3)$
- $\alpha_i [1 - y_i(W \cdot X_i + b)] = 0 \quad \cdots \cdots \cdots \cdots \cdots \cdots \cdots \cdots \rightarrow (4)$
for $i = 1 \text{ to } n.$

-
- Solve these equations (1), (2), (3), (4) to get W and b .
 - While it is possible to do this, it is tedious !

Wolfe Dual Formulation

- Other easy and advantageous ways to solve the optimization problem does exist which can be easily extended to non-linear SVMs.
- This is to get \mathcal{L} where we eliminate W and b and which has $\alpha_1, \dots, \alpha_n$.
- We know, \mathcal{L} has to be maximized w.r.t. the dual variables $\alpha_1, \dots, \alpha_n$.

Wolfe Dual Formulation

- The Lagrangian \mathcal{L} is :

$$= \frac{1}{2}||W||^2 + \sum_i \alpha_i - \sum_i \alpha_i y_i X_i \cdot W - \sum_i \alpha_i y_i b$$

$$= \frac{1}{2}||W||^2 + \sum_i \alpha_i - W \cdot \underbrace{\left[\sum_i \alpha_i y_i X_i \right]}_W - b \underbrace{\left[\sum_i \alpha_i y_i \right]}_0$$

$$= -\frac{1}{2}||W||^2 + \sum_i \alpha_i$$

$$= \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

Activate WiFi
Go to Settings

Wolfe Dual Formulation

- Maximize \mathcal{L} w.r.t α

$$\mathcal{L} = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

such that $\sum \alpha_i y_i = 0$, and $\alpha_i \geq 0$ for all i .

- We need to find the Lagrangian multipliers α_i ($1 \leq i \leq n$) only.
- Primal variables W and b are eliminated.
- There exists various numeric iterative methods to solve this constrained convex quadratic optimization problem.
- Sequential minimal optimization (SMO)* is one such technique which is a simple and relatively fast method.

- If $\alpha_i > 0$ (from (4)) $\Rightarrow X_i$ is on a hyperplane, i.e., X_i is a support vector.
Note: X_j lies on the hyperplane $\nRightarrow \alpha_i > 0$
- Similarly, X_i does not lie on hyperplane $\Rightarrow \alpha_i = 0$
That is, for interior points $\alpha_i = 0$.

The Solution

- Once α is known, We can find $W = \sum_i \alpha_i y_i X_i$
- The classifier is

$$f(X) = \text{Sign} \{W \cdot X + b\} = \text{Sign} \left\{ \sum_i \alpha_i y_i X_i \cdot X + b \right\}$$

- b can be found from (4)
 - For any $\alpha_j \neq 0$, we have $y_j(W \cdot X_j + b) = 1$
 - Multiplying with y_j on both sides we get,
$$W \cdot X_j + b = y_j$$
 - So, $b = y_j - W \cdot X_j = y_j - \sum_i \alpha_i y_i X_i \cdot X_j$

Some observations

- In the dual problem formulation and in its solution we have only dot products between some of the training patterns.
- Once we have the matrix \hat{K} , the problem and its solution are independent of the dimensionality d .

Non-linear SVM

- We know that every non-linear function in X -space (input space) can be seen as a linear function in an appropriate Y -space (feature space).
- Let the mapping be $Y = \phi(X)$.
- Once the $\phi(\cdot)$ is defined, one has to replace in the problem as well as in the solution, for certain products, as explained below.
- Whenever we see $X_i \cdot X_j$, replace this by $\phi(X_i) \cdot \phi(X_j)$.
- While it is possible to explicitly define the $\phi(\cdot)$ and generate the training set in the Y -space, and then obtain the solution...
- it is tedious and amazingly unnecessary also.

$$A = x^T - x = V A' x^T$$

Kernel Function

- For certain mappings, $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j)$. That is, dot product in the Y-space can be obtained as a function in the X-space itself. There is no need to explicitly generate the patterns in the Y-space.
- Eg: Consider a two dimensional problem with $X = (x_1, x_2)^t$. Let $\phi(X) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)^t$. Then $\phi(X_i) \cdot \phi(X_j) = \kappa(X_i, X_j) = (X_i \cdot X_j)^2$.
- This *kernel trick* is one of the reasons for the success of SVMs.

Wolfe Dual Formulation

- Maximize \mathcal{L} w.r.t α

$$\mathcal{L} = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j X_i \cdot X_j$$

such that $\sum \alpha_i y_i = 0$, and $0 \leq \alpha_i \leq C$ for all i .

- $C = \infty \Rightarrow$ Hard Margin
- $C = 0 \Rightarrow$ Very Very Soft Margin

Training Methods

- Any convex quadratic programming technique can be applied.
- But with larger training sets, most of the standard techniques can become very slow and space occupying. For example, many techniques need to store the kernel matrix whose size is n^2 where n is the number of training patterns.
- These considerations have driven the design of specific algorithms for SVMs that can exploit the sparseness of the solution, the convexity of the optimization problem, and the implicit mapping into feature space.
- One such a simple and fast method is Sequential Minimal Optimization (SMO).

- Builtin library functions, toolboxes are readily available in Python, ...

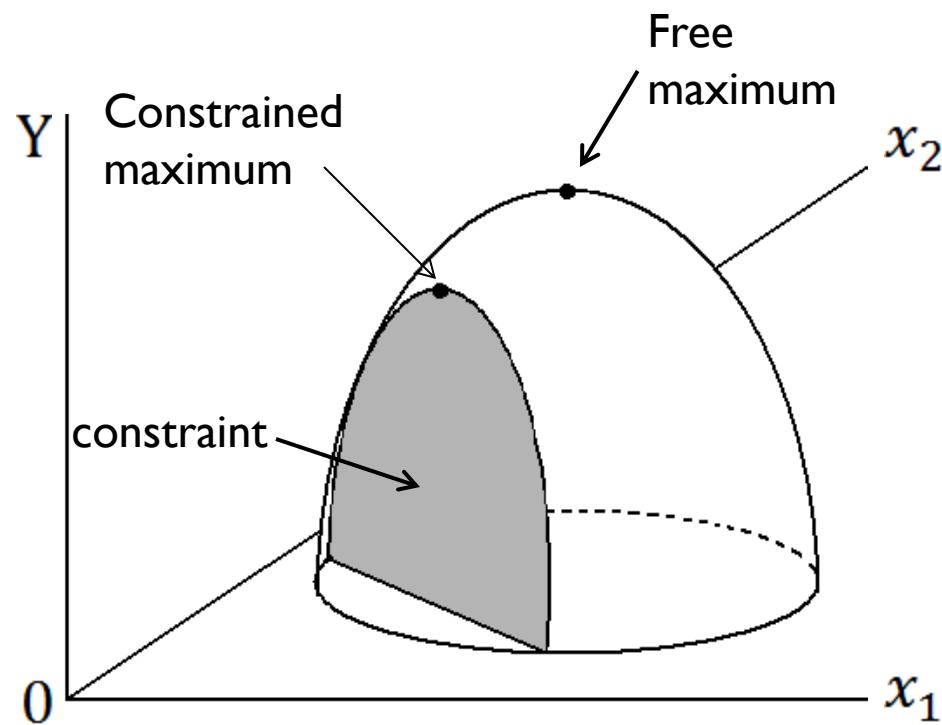


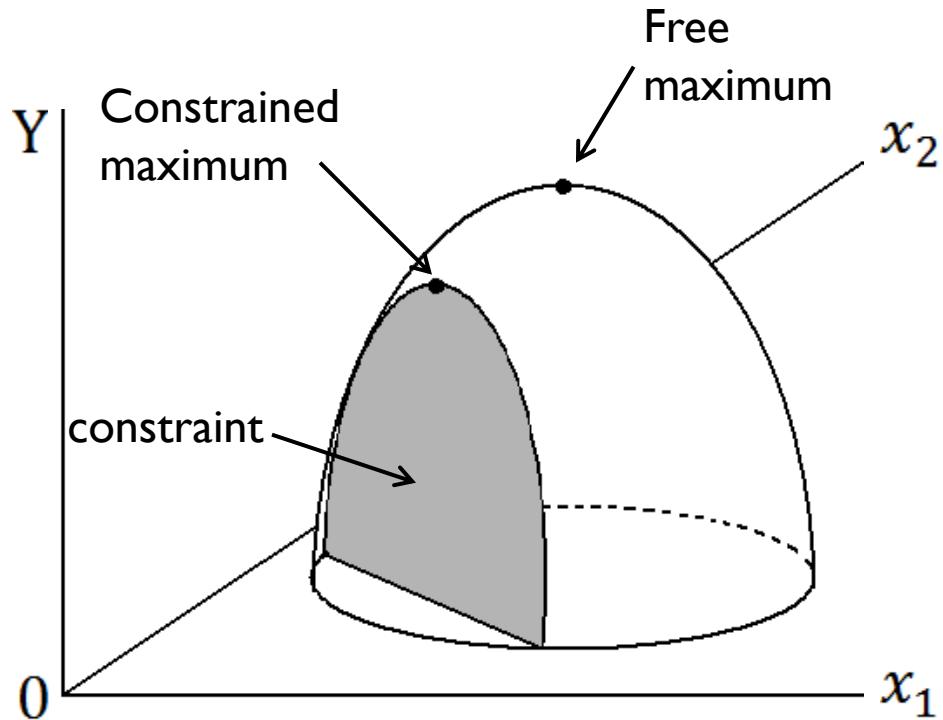
Constrained Optimization

We want to minimize (or maximize) the objective function, at the same time the solution should obey certain constraints.

Constrained Optimization

Graphically, the difference between the free optima and the constrained optima can be shown as:





- The free optima occurs at the peak of the surface.
- If we specify a specific relationship between variables x_1 and x_2 (a constraint) then the search for an optimum is restricted to a slice of the surface. The constrained maximum occurs at the peak of the slice.

Constrained Optimization

- Since economists deal with the allocation of scarce resources among alternative uses, the concept of constraints or restrictions is important.
- There are two approaches to solving constrained optima problems:
 - (i) substitution method
 - (ii) Lagrange multipliers

Substitution Method

- Consider a firm producing commodity y with the following production function:

$$y = 5x_1 x_2$$

- Without any constraints, the firm can produce an unlimited quantity by utilizing an unlimited amount of x_1 and x_2 .

Substitution Method

- But suppose the firm has a budget constraint:

Let $p_{x_1} = \$2/unit$

$p_{x_2} = \$1/unit$

- For simplicity, assume that the maximum amount the firm can spend on these two inputs is \$100.
- So we have the following constraint:

$$2x_1 + x_2 = 100$$

Substitution Method

- Suppose the economic question facing this firm is maximizing production subject to this budget constraint.
- The solution via the substitution method is to substitute:
 - First, write the constraint in terms of x_2 :

$$2x_1 + x_2 = 100$$

$$\therefore x_2 = 100 - 2x_1$$

Substitution Method

$$x_2 = 100 - 2x_1$$

- Then substitute this value into the production function, such that:

$$y = 5x_1 x_2$$

$$y = 5x_1(100 - 2x_1)$$

$$= 500x_1 - 10x_1^2$$

- With this substitution, the constrained maxima problem is reduced to a free maxima problem with one independent variable.

Substitution Method

- Now apply the usual optimization procedure:

$$\frac{dy}{dx_1} = 500 - 20x_1 = 0$$

$$-20x_1 = -500$$

(critical value) $\therefore x_1 = \frac{-500}{-20} = 25$

Substitution Method

$$\frac{dy}{dx_1} = 500 - 20x_1 = 0$$

$$\frac{d^2y}{dx_1^2} = -20 < 0 \quad \therefore \text{relative max}$$

$$\therefore \text{if } x_1 = 25 \text{ then } 100 = 2(25) + x_2$$

$$\therefore 100 - 50 = 50 = x_2$$

- The method of substitution is one way to solve constrained optima problems. This is manageable in some cases. In others, the constraint may be very complicated and substitution becomes complex.

Lagrange Multipliers

- The constrained optima problem can be stated as finding the extreme value of $y = f(x_1, x_2)$ subject to $g(x_1, x_2) = 0$.
- So Lagrange (a mathematician) formed the augmented function.

$$L = f(x_1, x_2) + \alpha(g(x_1, x_2))$$



denotes the augmented function called the Lagrangian, will behave like the function if the constraint is followed.

Lagrange Multipliers

- Given the augmented function, the first order condition for optimization (where the independent variables are x_1 , x_2 and λ) is as follows:

$$\left. \begin{array}{l} \frac{\partial L}{\partial x_1} = f_1 + \alpha g_1 = 0 \\ \frac{\partial L}{\partial x_2} = f_2 + \alpha g_2 = 0 \\ \frac{\partial L}{\partial \alpha} = g(x_1, x_2) = 0 \end{array} \right\} \text{Solve simultaneously for critical values}$$

Lagrange Multipliers

- Using the previous example:

$$L = 5x_1x_2 + \alpha(100 - 2x_1 - x_2)$$

note: $100 = 2x_1 + x_2$

$\therefore 100 - 2x_1 - x_2 = 0$ to be on the budget line

$$\left. \begin{array}{l} \frac{\partial L}{\partial x_1} = 5x_2 - 2\alpha = 0 \\ \frac{\partial L}{\partial x_2} = 5x_1 - \alpha = 0 \\ \frac{\partial L}{\partial \alpha} = 100 - 2x_1 - x_2 = 0 \end{array} \right\} \begin{array}{l} \text{3 unknowns:} \\ x_1, x_2, \alpha \\ \text{3 equations} \end{array}$$

Lagrange Multipliers

$$\left. \begin{array}{l} \frac{\partial L}{\partial x_1} = 5x_2 - 2\alpha = 0 \\ \frac{\partial L}{\partial x_2} = 5x_1 - \alpha = 0 \\ \frac{\partial L}{\partial \alpha} = 100 - 2x_1 - x_2 = 0 \end{array} \right\} \begin{array}{l} \text{3 unknowns:} \\ x_1, x_2, \alpha \\ \text{3 equations} \end{array}$$

- Solving these 3 equations simultaneously:

$$5x_2 - 2\alpha = 0$$

$$5x_2 = 2\alpha$$

$$\therefore x_2 = \frac{2\alpha}{5}$$

Lagrange Multipliers

$$\left. \begin{array}{l} \frac{\partial L}{\partial x_1} = 5x_2 - 2\alpha = 0 \\ \frac{\partial L}{\partial x_2} = 5x_1 - \alpha = 0 \\ \frac{\partial L}{\partial \alpha} = 100 - 2x_1 - x_2 = 0 \end{array} \right\} \begin{array}{l} \text{3 unknowns:} \\ x_1, x_2, \alpha \\ \text{3 equations} \end{array}$$

- Solving these 3 equations simultaneously (cont'd):

$$x_1 = \frac{\alpha}{5}$$

Lagrange Multipliers

$$x_1 = \frac{\alpha}{5}$$

$$x_2 = \frac{2\alpha}{5}$$

- Solving these 3 equations simultaneously (cont'd):

$$\therefore 100 - 2\left(\frac{\alpha}{5}\right) - \left(\frac{2\alpha}{5}\right) = 0$$

$$500 = 4\alpha$$

$$100 = \frac{4\alpha}{5}$$

$$\alpha = 125$$

Lagrange Multipliers

- This solution yields the same answer as the substitution method, i.e.,

$$x_1 = 25 \qquad x_2 = 50$$

Lagrange Multipliers

- Economists prefer using the Lagrange technique over the substitution method, because:
 - (i) easier to handle for most cases and
 - (ii) provides additional information.
- The Lagrange multiplier gives how much sensitive the constraint is

KKT Conditions

- In the presence of inequality constraints, one can use KKT conditions which are necessary conditions for the optimality.
 - These are sufficient also, provided the objective is convex and constraints are linear. (This is what exactly happens in the case of SVMs).

Constrained Optimization Problem

- Minimize $f(v)$
Subject to the constraints $g_j(v) \leq 0, \quad 1 \leq j \leq n.$
- Lagrangian,

$$\mathcal{L} = f(v) + \sum_{j=1}^n \alpha_j g_j(v)$$

where v is called *primary* variables and α_j are the Lagrangian multipliers which are also called *dual* variables.

- \mathcal{L} has to be minimized with respect to primal variables and maximized with respect to dual variables.

K.K.T Conditions

$$\begin{array}{l} (i) \quad \nabla_v L = 0 \\ (ii) \quad \alpha_j \geq 0 \\ (iii) \quad \alpha_j g_j(v) = 0 \\ (iv) \quad g_j(v) \leq 0 \end{array} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{for all } j = 1 \text{ to } n$$

- K.K.T. Conditions, in general are necessary, i.e., at optimal point these are satisfied. So, if these are not satisfied we know that the point we are concerned is not optimal.
- However, when the objective is a convex function and constraints are all linear functions, then K.K.T conditions are sufficient also.
- The (iv) th one is within the problem statement. So normally first 3 conditions are called the KKT conditions.

Example.

Minimize $f(x) = (x - 4)^2 + 5$, such that $x \geq 6$.

$$L = (x - 4)^2 + 5 + \alpha(-x + 6)$$

KKT conditions:

$$(1) \frac{\partial L}{\partial x} = 0, \text{ so } x = \frac{1}{2}(\alpha + 8)$$

$$(2) \alpha(-x + 6) = 0$$

$$(3) \alpha \geq 0$$

(2) and (3) along with the problem constraint, gives $x = 6$.

Example

Minimize $f(v_1, v_2) = v_1 + v_2$, such that $v_1^2 + v_2^2 \leq 1$.

Solution: $L = (v_1 + v_2) + \alpha(v_1^2 + v_2^2 - 1)$.

KKT Conditions

$$(1) \frac{\partial L}{\partial v_1} = 1 + 2\alpha v_1 = 0, \quad \frac{\partial L}{\partial v_2} = 1 + 2\alpha v_2 = 0.$$

$$So, v_1 = v_2 = -\frac{1}{2\alpha}. \quad So \alpha \neq 0.$$

$$(2) \alpha \geq 0$$

$$(3) \alpha(v_1^2 + v_2^2 - 1) = 0.$$

From (2) and (3) since $\alpha > 0$, we have $v_1^2 + v_2^2 - 1 = 0$.

$$This gives \alpha = \frac{1}{\sqrt{2}}.$$

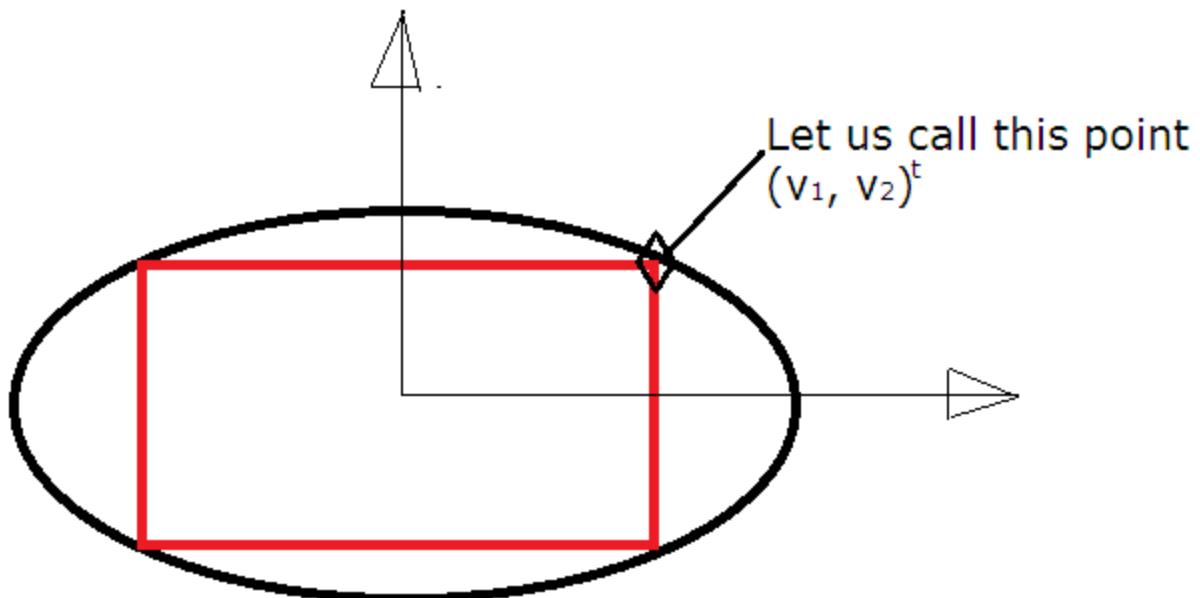
$$So, we get v_1 = v_2 = -\frac{1}{\sqrt{2}}$$

With both equality and inequality constraints

- We see an example.

An Example.

- Find maximum perimeter rectangle that is inscribed in the ellipse $x^2 + 4y^2 = 4$.



- Maximize the perimeter= $4(v_1 + v_2)$, subject to $v_1^2 + 4v_2^2 - 4 = 0$. Also, note we have constraints $v_1 \geq 0$, and $v_2 \geq 0$.
- Lagrangian, $L(v_1, v_2, \alpha_1, \alpha_2, \alpha_3) = -4(v_1 + v_2) + \alpha_1(v_1^2 + 4v_2^2 - 4) + \alpha_2(-v_1) + \alpha_3(-v_2)$.

KKT Conditions

- $\frac{\partial L}{\partial v_1} = 0$
- $\frac{\partial L}{\partial v_2} = 0$
- $\frac{\partial L}{\partial \alpha_1} = 0$
- $v_1 \alpha_2 = 0, \quad v_2 \alpha_3 = 0$
- $\alpha_2 \geq 0, \quad \alpha_3 \geq 0$

- We get $\alpha_1 = \frac{\sqrt{5}}{2}, \alpha_2 = \alpha_3 = 0$.
- We get the solution, ...

Decision Tree Induction

Non-metric Methods

- Numerical Attributes
 - Nearest-neighbor -- distance
 - Neural networks: two similar inputs leads to similar outputs
 - SVMs: Dot Product

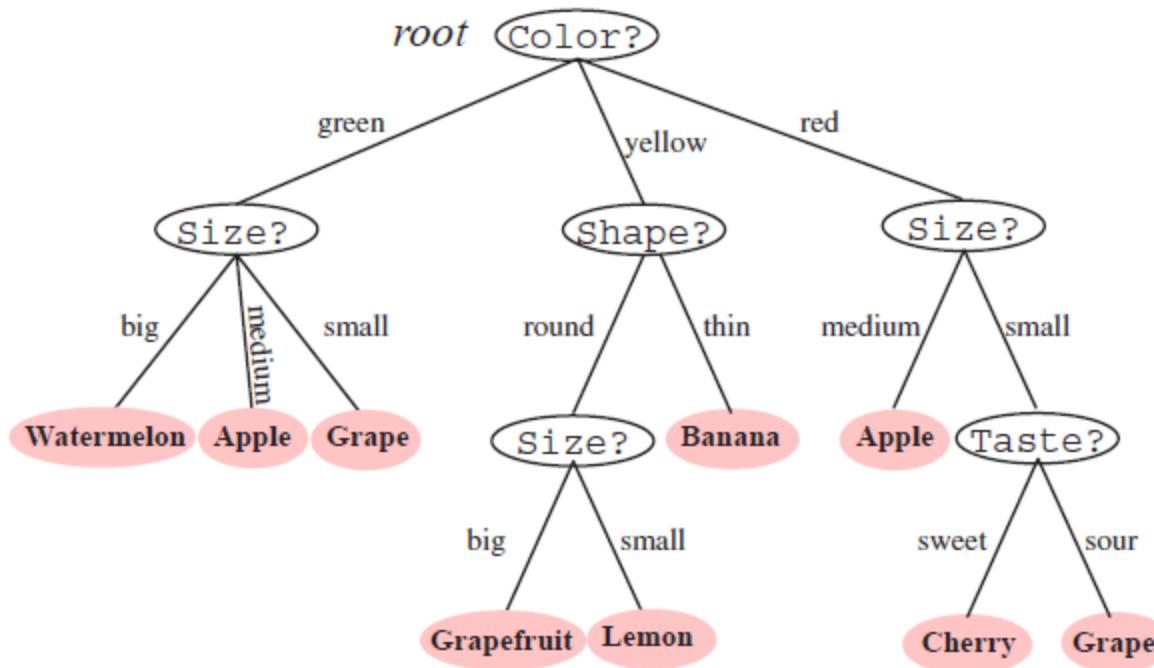
Non-metric data

- Nominal attributes
- Color, taste
- Strings: DNA
-

- Probability based
 - Naïve Bayes
- Rule based
 - Decision trees

Decision Tree

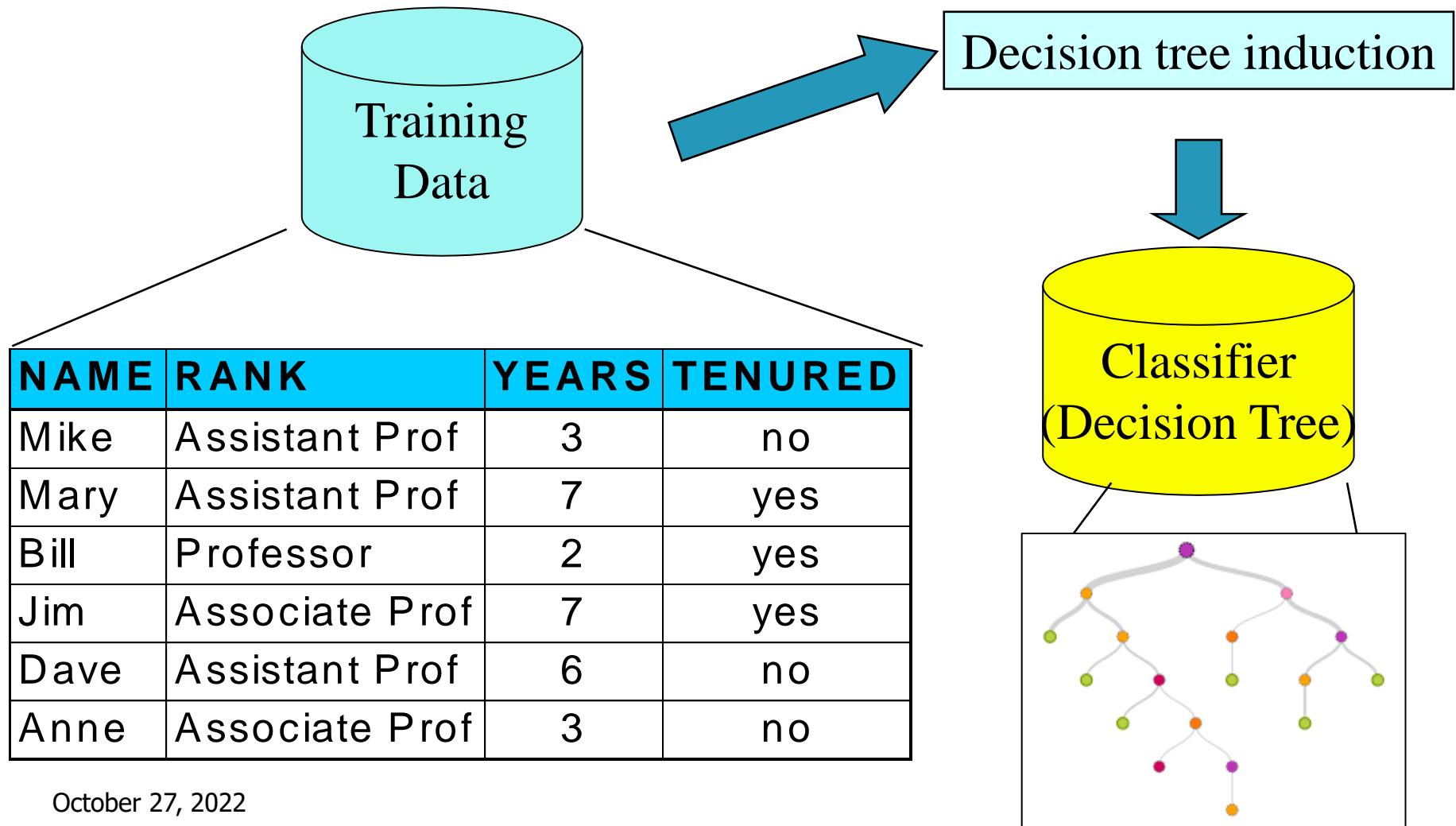
- Rules in the form of a hierarchy.



- Why are decision trees so popular?

- Why are decision trees so popular?
- Interpretability
 - You can give human understandable explanation for the decision being made.

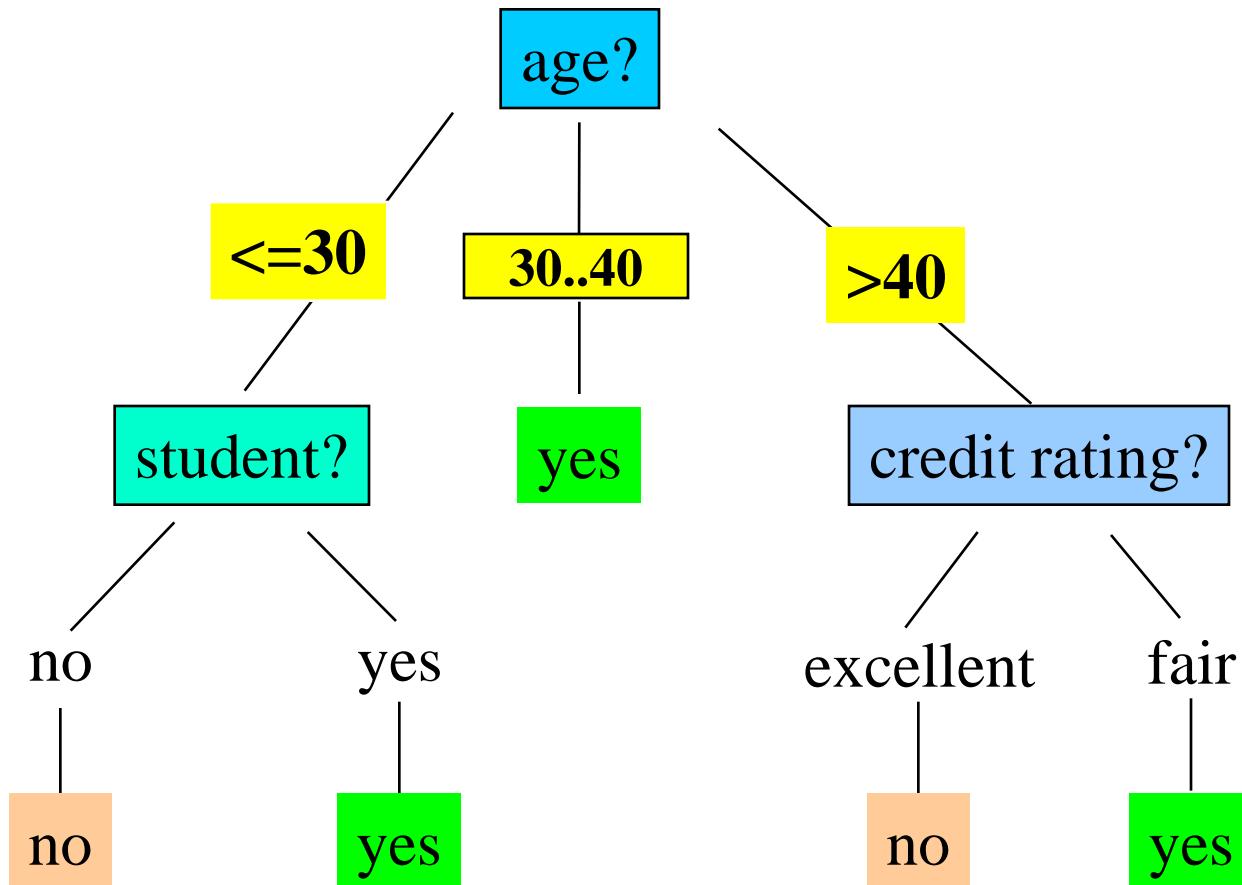
We need to work with a training set



You need to work with a training set

age	income	student	cred_rati	buys_comp
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Output: A Decision Tree for “buys_computer”



How to create a Decision Tree

- From root to leaves it is created
- For root, choose the attribute to be used for testing. This gives child nodes for the root.
- For each child, again choose the attribute to be tested.

Issues

- Criteria for choosing an attribute?
- You can achieve 100% accuracy with training set?!
 - Overfitting
- When you stop building the tree?
- Are there various types of DT induction methods?? ID3, C4.5 and CART.

Decision tree induction

- They adopt a greedy (i.e., nonbacktracking), top-down recursive divide-and-conquer approach.

- Node → subset of training patterns
- Root → training set.
- Leaf → class label.

Impurity measures

- Entropy impurity (information impurity)

$$i(N) = - \sum_j P(\omega_j) \log_2 P(\omega_j)$$

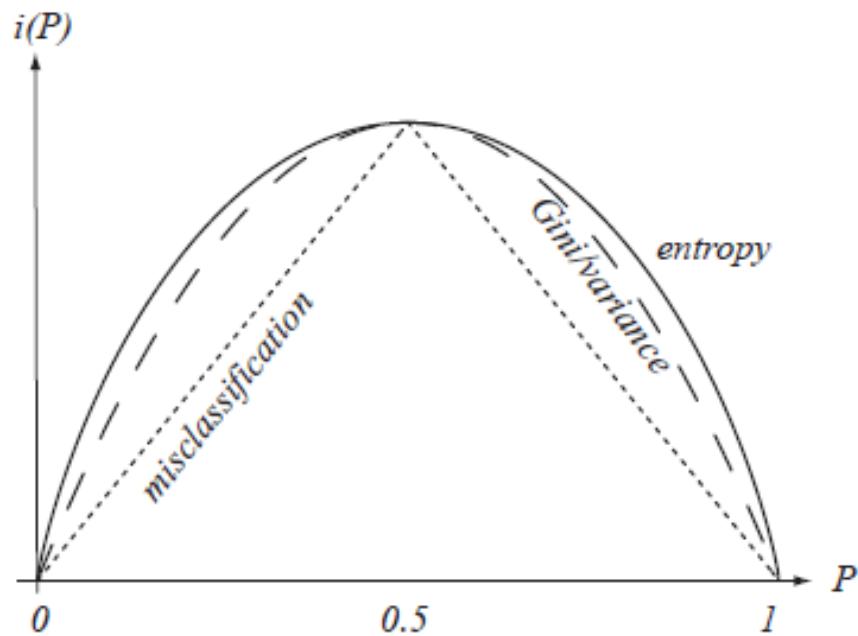
- Gini impurity (variance impurity)

$$i(N) = 1 - \sum_j P^2(\omega_j)$$

- Misclassification impurity

$$i(N) = 1 - \max_j P(\omega_j)$$

For a two category case



Which test?

- That which drops the impurity greater.
 - Try to become pure quickly.

$$\Delta i(N) = i(N) - (P_L i(N_L) + (1 - P_L) i(N_R)),$$

Which test?

- That which drops the impurity greater.
 - Try to become pure quickly.

$$\Delta i(N) = i(N) - (P_L i(N_L) + (1 - P_L) i(N_R)),$$

where N_L and N_R are the left and right descendent nodes,
 $i(N_L)$ and $i(N_R)$ their impurities,
and P_L is the fraction of patterns at node N that will go to N_L

Which test?

- That which drops the impurity greater.
 - Try to become pure quickly.

$$\Delta i(N) = i(N) - (P_L i(N_L) + (1 - P_L) i(N_R)),$$

where N_L and N_R are the left and right descendent nodes,
 $i(N_L)$ and $i(N_R)$ their impurities,
and P_L is the fraction of patterns at node N that will go to N_L

Then the “best” test value s is the choice for T that maximizes $\Delta i(T)$.

Which test?

- That which drops the impurity greater.
 - Try to become pure quickly.

$$\Delta i(N) = i(N) - (P_L \ i(N_L) + (1 - P_L) \ i(N_R)),$$



Which test?

- That which drops the impurity greater.
 - Try to become pure quickly.

$$\Delta i(N) = i(N) - (P_L \ i(N_L) + (1 - P_L) \ i(N_R)),$$



Information gain

- This is drop in entropy impurity !!
- For an attribute A , often written as $Gain(A)$

Gain(age) ??

age	income	student	cred_rati	buys_comp
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

(yes, no) = (9, 5)

Gain(age) ??

age	income	student	cred_rati	buys_comp
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

$$(\text{yes}, \text{no}) = (9, 5)$$

$$\begin{aligned} i(\text{root}) &= I(9,5) \\ I(9,5) &= -\frac{9}{14} \log \frac{9}{14} - \frac{5}{14} \log \frac{5}{14} \\ &= 0.94 \end{aligned}$$

age	Yes	No	Impurity
<=30	2	3	0.971
31...40	4	0	0
>40	3	2	0.971

$$\begin{aligned} \Delta i(\text{age}) &= 0.94 - \left(\frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) \right) \\ &= 0.69 \end{aligned}$$

We call this $\text{Gain}(\text{age}) = 0.69$.

For other attributes, their GAIN

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit_rating) = 0.048$$

- So we choose age as the splitting attribute.

- Similarly one can use other impurity measures

Gini Index (IBM IntelligentMiner)

- If a data set T contains examples from n classes, gini index, $gini(T)$ is defined as

$$gini(T) = 1 - \sum_{j=1}^n p_j^2$$

where p_j is the relative frequency of class j in T .

- If a data set T is split into two subsets T_1 and T_2 with sizes N_1 and N_2 respectively, the *gini* index of the split data contains examples from n classes, the *gini* index $gini(T)$ is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute provides the smallest $gini_{split}(T)$ is chosen to split the node (*need to enumerate all possible splitting points for each attribute*).

- But, there is one drawback with this approach!

- A split with large branching factor is often chosen.
 - So, telephone number is chosen.

$$\Delta i(s) = i(N) - \sum_{k=1}^B P_k i(N_k)$$

$$\sum_{k=1}^B P_k = 1.$$

So, we penalize large branching factors

- This is called ***gain ratio*** (very often used with *information gain*).

$$\Delta i_B(s) = \frac{\Delta i(s)}{-\sum_{k=1}^B P_k \log_2 P_k}.$$

- Branching factor is more, the denominator is more.

Extracting Classification Rules from Trees

- Represent the knowledge in the form of **IF-THEN** rules
- One rule is created for each path from the root to a leaf
- Each attribute-value pair along a path forms a conjunction
- The leaf node holds the class prediction
- Rules are easier for humans to understand
- Example

IF *age* = “<=30” AND *student* = “no” THEN *buys_computer* = “no”

IF *age* = “<=30” AND *student* = “yes” THEN *buys_computer* = “yes”

IF *age* = “31...40” THEN *buys_computer* = “yes”

IF *age* = “>40” AND *credit_rating* = “excellent” THEN *buys_computer* = “yes”

IF *age* = “>40” AND *credit_rating* = “fair” THEN *buys_computer* = “no”

Avoid Overfitting in Classification

- The generated tree may overfit the training data
 - Too many branches, some may reflect anomalies due to noise or outliers
 - Result is in poor accuracy for unseen samples
- Two approaches to avoid overfitting
 - Prepruning: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
 - Difficult to choose an appropriate threshold
 - Postpruning: Remove branches from a “fully grown” tree—get a sequence of progressively pruned trees
 - Use a set of data different from the training data to decide which is the “best pruned tree”

Classification in Large Databases

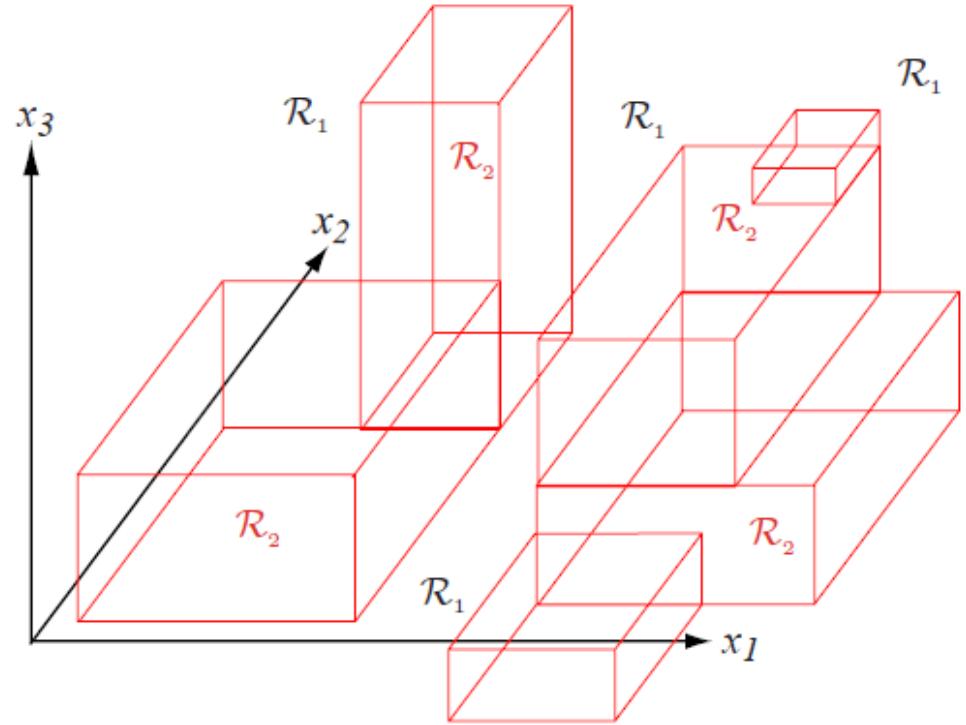
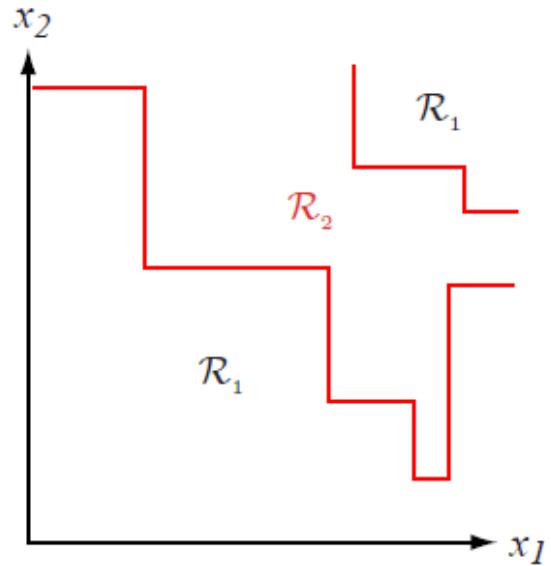
- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why decision tree induction in classification?
 - relatively faster learning speed (than other classification methods)
 - convertible to simple and easy to understand classification rules
 - can use SQL queries for accessing databases
 - comparable classification accuracy with other methods

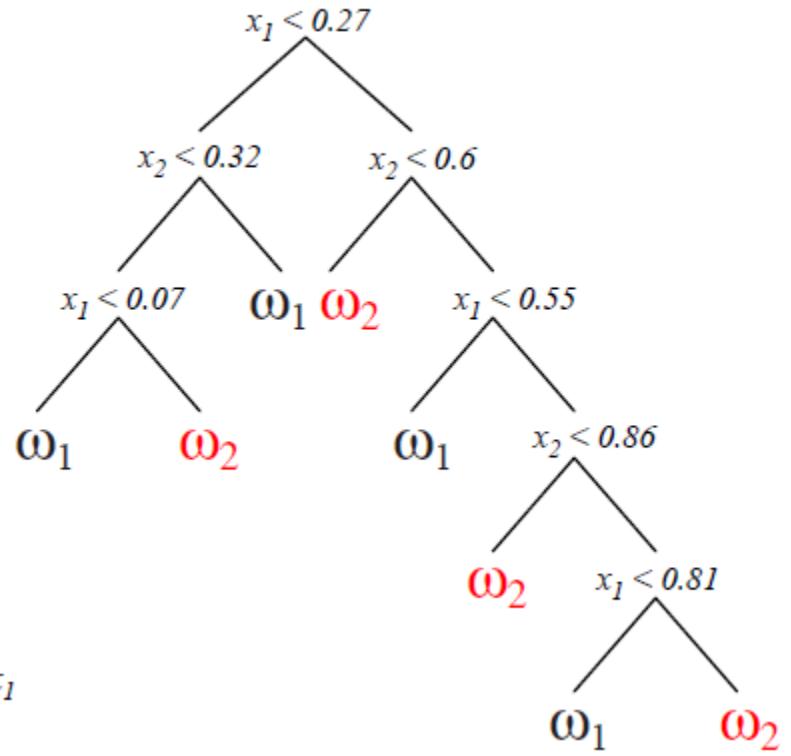
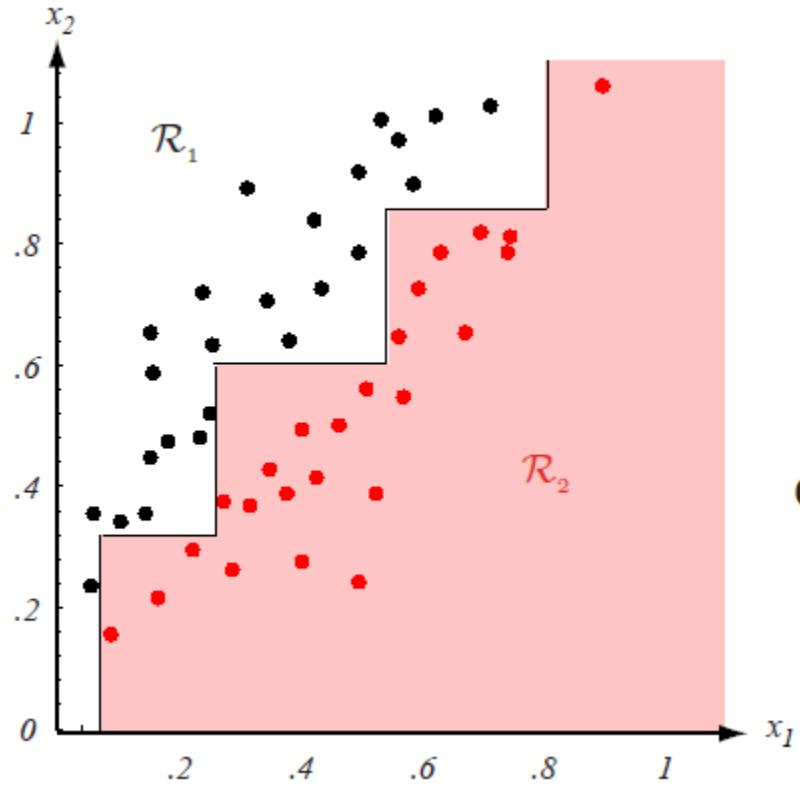
Scalable Decision Tree Induction Methods in Data Mining Studies

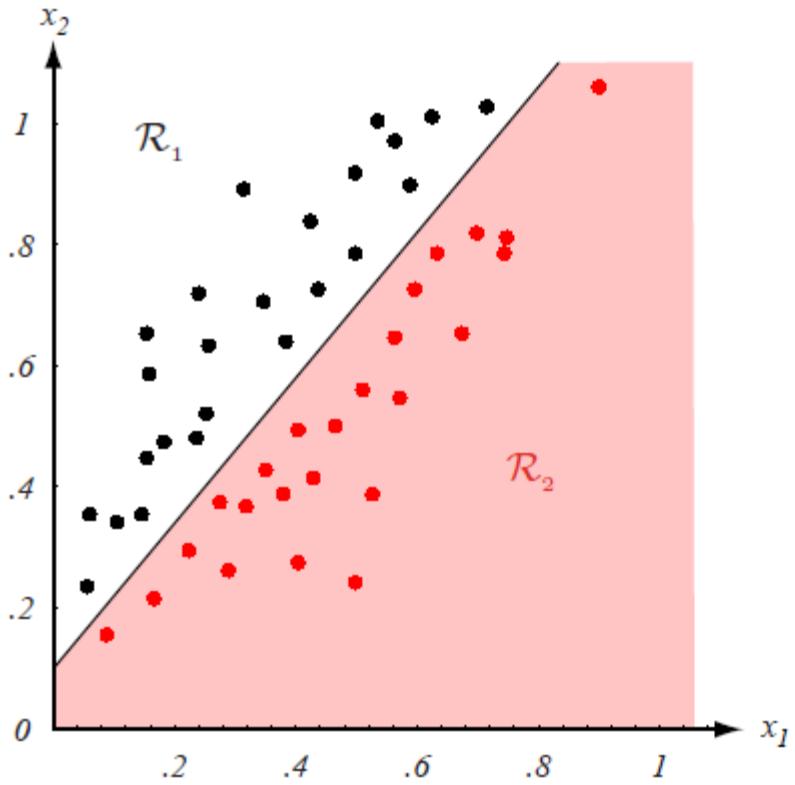
- **SLIQ** (EDBT'96 — Mehta et al.)
 - builds an index for each attribute and only class list and the current attribute list reside in memory
- **SPRINT** (VLDB'96 — J. Shafer et al.)
 - constructs an attribute list data structure
- **PUBLIC** (VLDB'98 — Rastogi & Shim)
 - integrates tree splitting and tree pruning: stop growing the tree earlier
- **RainForest** (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
 - separates the scalability aspects from the criteria that determine the quality of the tree
 - builds an AVC-list (attribute, value, class label)

Drawbacks

- What we discussed are axis parallel
- For continuous valued attributes cut-points can be found.
 - Can be discretized (CART does).







$$-1.2x_1 + x_2 < 0.1$$

```

graph TD
    A[ -1.2x1 + x2 < 0.1 ] --> B[w2]
    A --> C[w1]
  
```

Machine Learning

Decision Trees

Nominal Data

- So far we consider patterns to be represented by feature vectors of real or integer values.
- Easy to come up with a distance (similarity) measure by using a variety of mathematical norms.
- What happens if features are not numbers?
- May not have a numerical representation
- Distance measures might not make sense

Nominal Data: Examples

- Consider the use of information about teeth in the classification of fish and sea mammals.

E.g.:

- Some teeth are small and fine (as in baleen whales) for straining tiny prey from the sea. Others (as in sharks) coming in multiple rows.
- Consider describing a piece of fruit by the four properties of color, texture, taste and smell.
 - color = red, texture = shiny,
 - taste = sweet and size = small
- Another common approach is to describe the pattern by a variable length string of nominal attributes, such as sequence of base pairs string in a segment of DNA,
 - E.g.: "AGCTTCAGATTCCA."

How to use this data for classification/regression?

- How can we best use such nominal data for classification?
- Most importantly, how can we efficiently learn categories using such non-metric data?
- If there is structure in strings, how can it be represented?

How to use this data for classification/regression?

- Visualizing using n-dimensional space might be difficult how to map, say, smell, onto an axis?
- There might only be few discrete values (an article is highly interesting, somewhat interesting, not interesting, etc.)
- Even though that helps, do remember you cannot take distance measure in that space

(e.g., Euclidean distance in r-g-b color space does not correspond to human perception of color)

Decision Trees

- A classification based on a sequence of questions on
 - A particular feature (E.g., is the fruit sweet or not?) or
 - A particular set of features (E.g., is this article relevant and interesting?)
- Answer can be either
 - Yes/no
 - Choice (relevant & interesting, interesting but not relevant, relevant but not interesting, etc.)
 - Usual a finite number of discrete values

Decision Trees

- It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.
- This approach is particularly useful for non-metric data, since all of the questions can be asked in a “yes/no” or “true/false” or “value(property) \in set of values” style that does not require any notion of metric.
- ***Such sequence of questions is displayed in a directed decision tree or simply tree.***

Credits: Pattern Classification, Duda hart

Decision Trees

- *Such sequence of questions is displayed in a directed decision tree or simply tree.*
- Where by convention **root node** the first or root node is displayed at the top.
- Root is connected by **successive (directional) links or branches** to other nodes.
- These are similarly connected until we reach **terminal or leaf nodes**, which have no further links.

Decision Tree

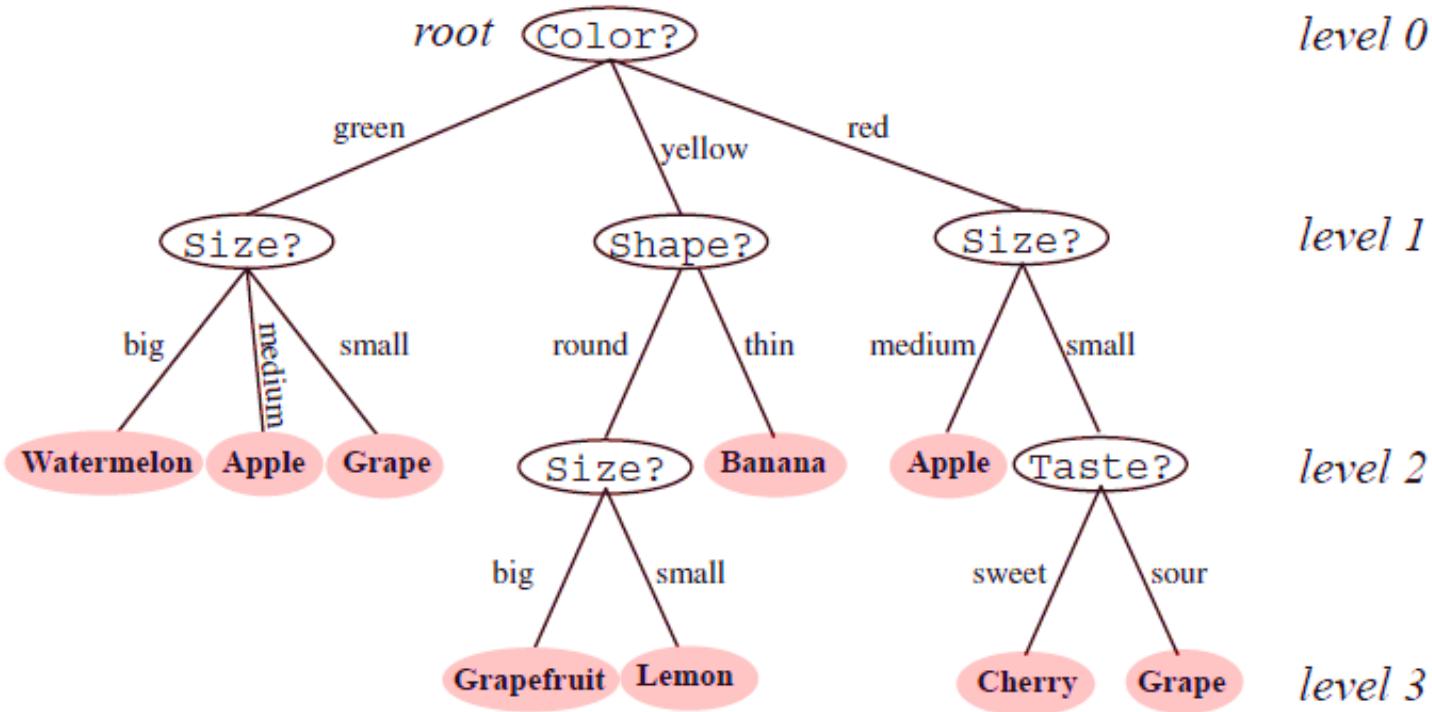


Figure 8.1: Classification in a basic decision tree proceeds from top to bottom. The questions asked at each node concern a particular property of the pattern, and the downward links correspond to the possible values. Successive nodes are visited until a terminal or leaf node is reached, where the category label is read. Note that the same question, **Size?**, appears in different places in the tree, and that different questions can have different numbers of branches. Moreover, different leaf nodes, shown in pink, can be labeled by the same category (e.g., **Apple**).

DT: How they are used for classification?

- The classification of a particular pattern begins at the root node, which asks for the value of a particular property of the pattern.
- The different links from the root node correspond to the different possible values.
- Based on the answer we follow the appropriate link to a subsequent or descendent node.
- In the trees, the links must be mutually distinct and exhaustive, i.e., one and only one link will be followed.

DT: How they are used for classification?

- The next step is to make the decision at the sub-tree appropriate subsequent node, which can be considered the root of a sub-tree.
- We continue this way until we reach a leaf node, which has no further question.
- Each leaf node bears a category label and the test pattern is assigned the category of the leaf node reached.

Creation of a Decision Tree

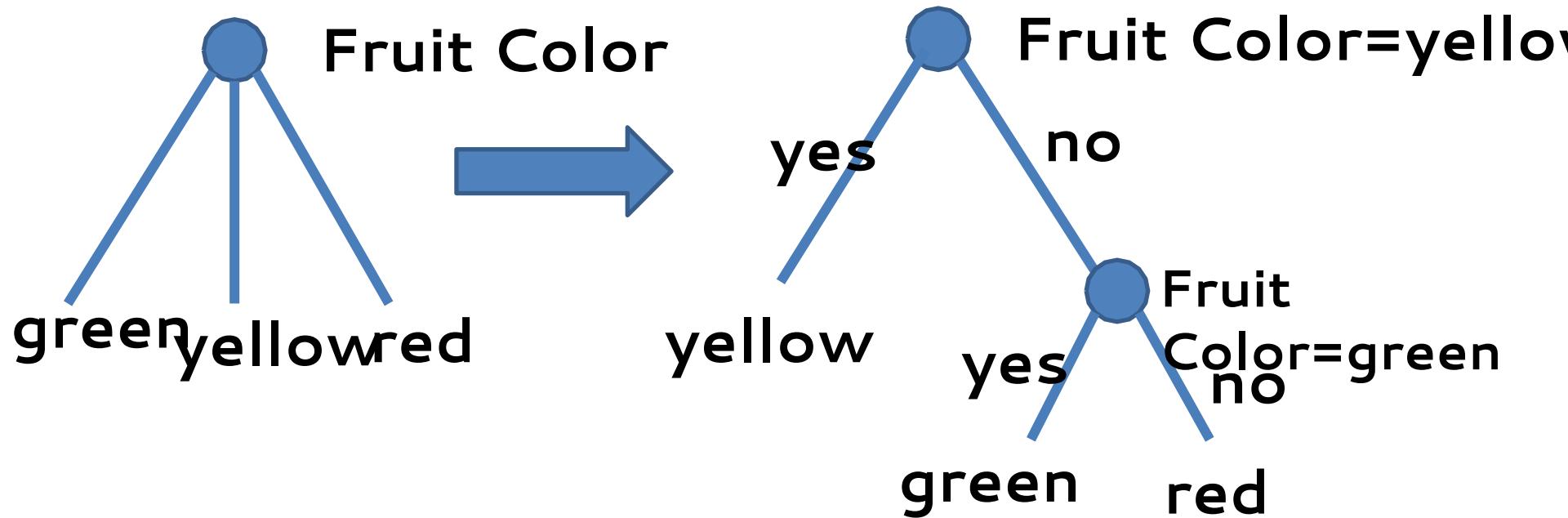
- Use supervised learning
 - Samples with tagged label (just like before)
- Process
 - Number of splits
 - Query selection
 - Rule for stopping splitting and pruning
 - Rule for labelling the leaves
 - Variable combination and missing data

Number of splits

- Each decision outcome at a node is called a *split*, since it corresponds to splitting a subset of the training data.
- The root node splits the full training set; each successive decision splits a proper subset of the data.
- The number of splits at a node is closely related to which property need to be tested and specifying *which* particular split will be made at a node.
- The number of links descending from a node is sometimes called branching the node's *branching factor* or *branching ratio*, denoted B .

Binary vs Multi-way Splits

- Binary vs. Multi-way
 - Can always make a multi-way split into binary splits



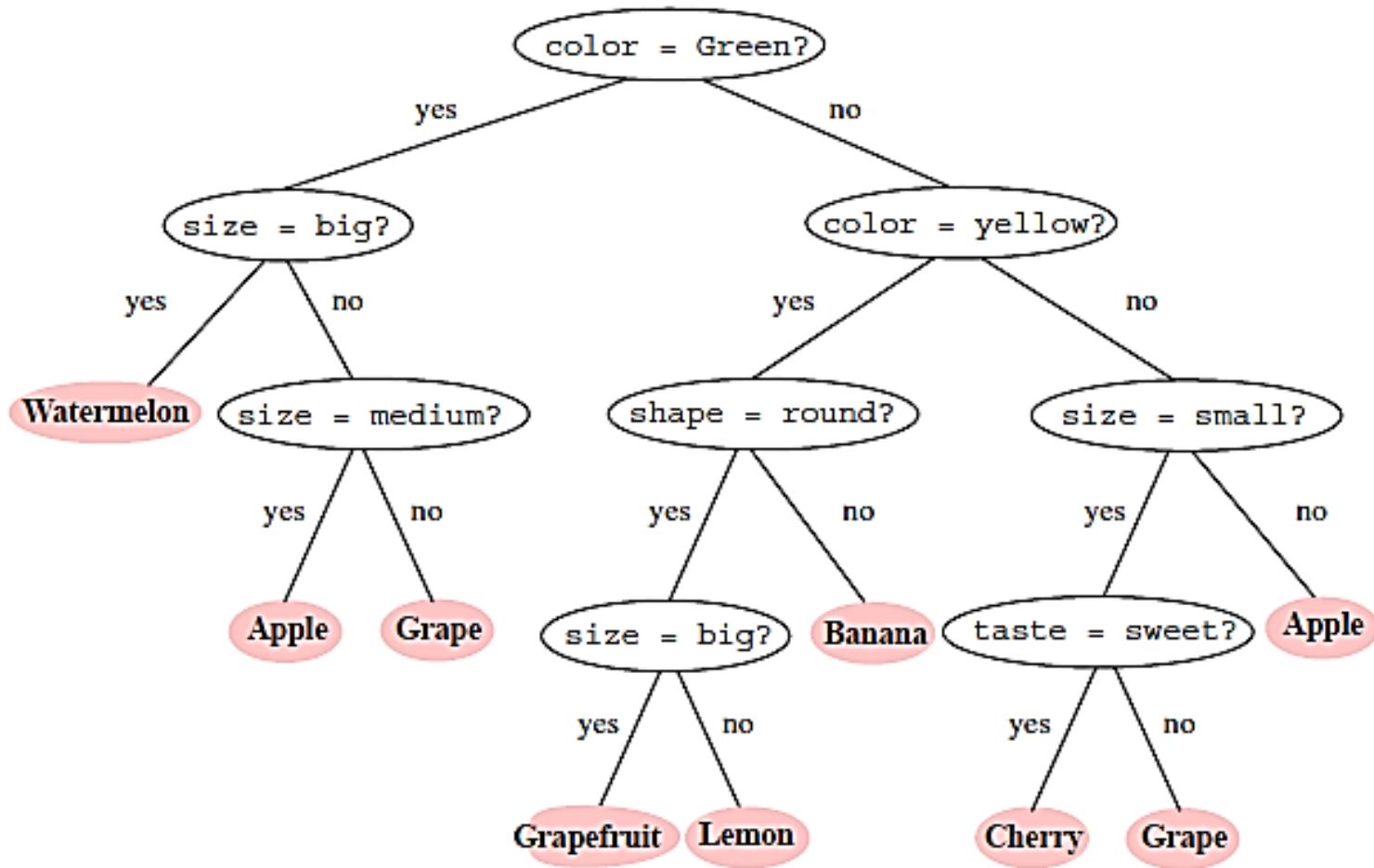


Figure 8.2: A tree with arbitrary branching factor at different nodes can always be represented by a functionally equivalent binary tree, i.e., one having branching factor $B = 2$ throughout. By convention the “yes” branch is on the left, the “no” branch on the right. This binary tree contains the same information and implements the same classification as that in Fig. 8.1.

Test selection

- If a feature is an ordered variable,
 - we might ask is $x > c$, for some c .
- If a feature is a category, we might ask is x in a particular category
 - *Yes* sends samples to left and *no* sends samples to right
- Simple rectangular partitions of the feature space
 - More complicated ones: is $x > 0.5$ & $y < 0.3$

Test selection

- The fundamental principle underlying tree creation is that of simplicity: we prefer decisions that lead to a simple, compact tree with few nodes.
- *This is a version of Occam's razor, that the simplest model that explains data is the one to be preferred.*

"Occam's razor is the problem-solving principle states that "entities should not be multiplied without necessity", or more simply, the simplest explanation is usually the right one."

- *To this end, we seek a property test T at each node N that makes the purity data reaching the immediate*

Decision boundary

- For example, suppose that the test at each node has the form “is $x_i \leq x_{i,s}$?” This leads to hyperplane decision boundaries that are perpendicular to the coordinate axes, and to decision regions of the form illustrated in Fig.

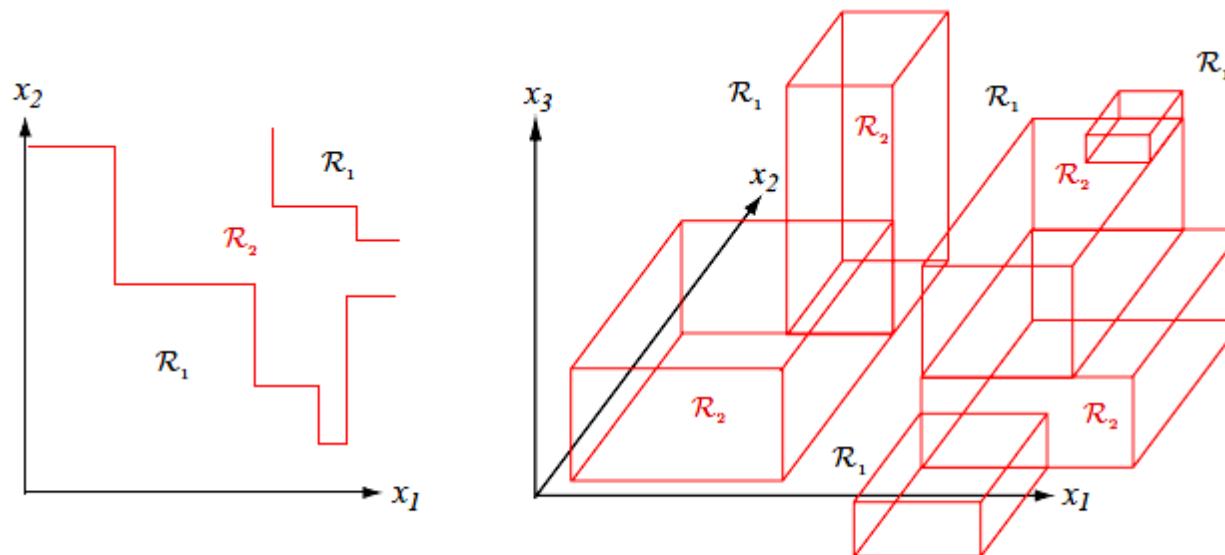


Figure 8.3: Monothetic decision trees create decision boundaries with portions perpendicular to the feature axes. The decision regions are marked \mathcal{R}_1 and \mathcal{R}_2 in these two-dimensional and three-dimensional two-category examples. With a sufficiently large tree, any decision boundary can be approximated arbitrarily well.

Criteria for Splitting

- Intuitively, to make the populations of the samples in the two children nodes purer than the parent node
- What do you mean by pure?
- In formalizing this notion, it turns out to be more convenient to define the *impurity*, rather than purity.
- General formulation
 - At node n , with k classes
 - Impurity depends on probabilities of samples at that node being in a certain class

$$i(n) = f(P(w_1 | n), P(w_2 | n), \dots, P(w_k | n))$$

Impurity

- Several different mathematical measures of impurity have been proposed, all of which have basically the same behaviour.
 - Entropy Impurity
 - Variance Impurity
 - Gini Impurity
 - Misclassification impurity

Machine Learning

Decision Trees

Impurity

- Several different mathematical measures of impurity have been proposed, all of which have basically the same behaviour.
 - Entropy Impurity
 - Variance Impurity
 - Gini Impurity
 - Misclassification impurity

Entropy Impurity

- Let $i(N)$ denote the impurity of a node N . In all cases, we want $i(N)$ to be 0 if all of the patterns that reach the node bear the same category label, and to be large if the categories are equally represented.
- The most popular measure is the *entropy impurity (information impurity)*.
- **Entropy**: A measure of “randomness” or “unpredictability.
- In information theory, the number of bits that are needed to code the transmission.

Entropy Impurity

$$i(N) = - \sum_j P(\omega_j) \log_2 P(\omega_j),$$

- where $P(\omega_j)$ is the fraction of patterns at node N that are in category ω_j .*
- By the well-known properties of entropy, if all the patterns are of the same category, the impurity is 0; otherwise it is positive, with the greatest value occurring when the different classes are equally likely.

variance impurity

- Another definition of impurity is particularly useful in the two-category case.
- Given the desire to have zero impurity when the node represents only patterns of a single category, the simplest polynomial form is:

$$i(N) = P(\omega_1)P(\omega_2)$$

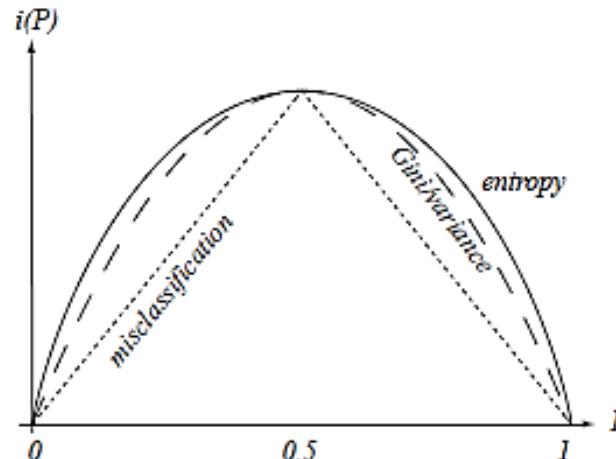
- It is related to the variance of a distribution associated with the two categories.

Gini impurity

- A generalization of the variance impurity, applicable to two or more categories, is the Gini impurity:

$$i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = 1 - \sum_j P^2(\omega_j).$$

- This is just the expected error rate at node N if the category label is selected randomly from the class distribution present at N .
- This criterion is more strongly peaked at equal probabilities than is the entropy impurity.



Misclassification impurity

- It measures the minimum probability that a training pattern would be misclassified at N .

$$i(N) = 1 - \max_j P(\omega_j),$$

- this measure is the most strongly peaked at equal probabilities.
- We now come to the key question – *given a partial tree down to node N, what value s should we choose for the property test T?*

Finding an optimal decision for a node

- The drop in impurity is defined by

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R),$$

where N_L and N_R are the left and right descendent nodes, $i(N_L)$ and $i(N_R)$ their impurities, and P_L is the fraction of patterns at node N that will go to N_L when property test T is used.

- Then the “best” test value s is the choice for T that maximizes

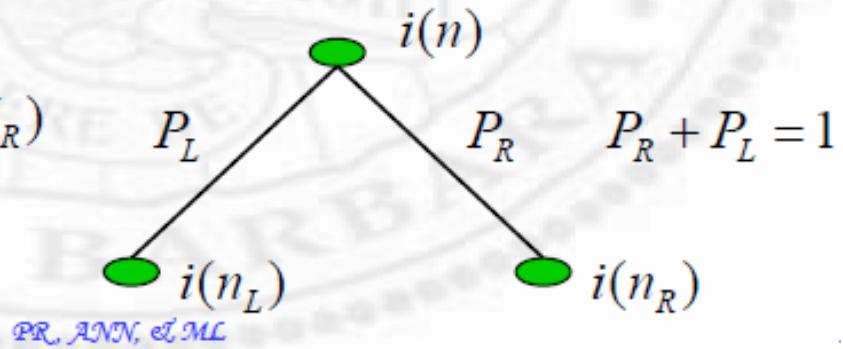
$\Delta i(T)$ (gain).

- If the form of the decisions is based on the nominal attributes, we may have to perform extensive or exhaustive search over all possible subsets of the training set to find the rule maximizing Δi .
- If the attributes are real-valued, one could use gradient descent algorithms to find a splitting

Split Decision

- ❖ Before split – fixed impurity
- ❖ After split – impurity depends on decisions
- ❖ The goal is maximize the drop in impurity
- ❖ Difference between
 - Impurity at root
 - Impurity at children (weighted by population)

$$\Delta i(n) = i(n) - P_L i(N_L) - P_R i(N_R)$$

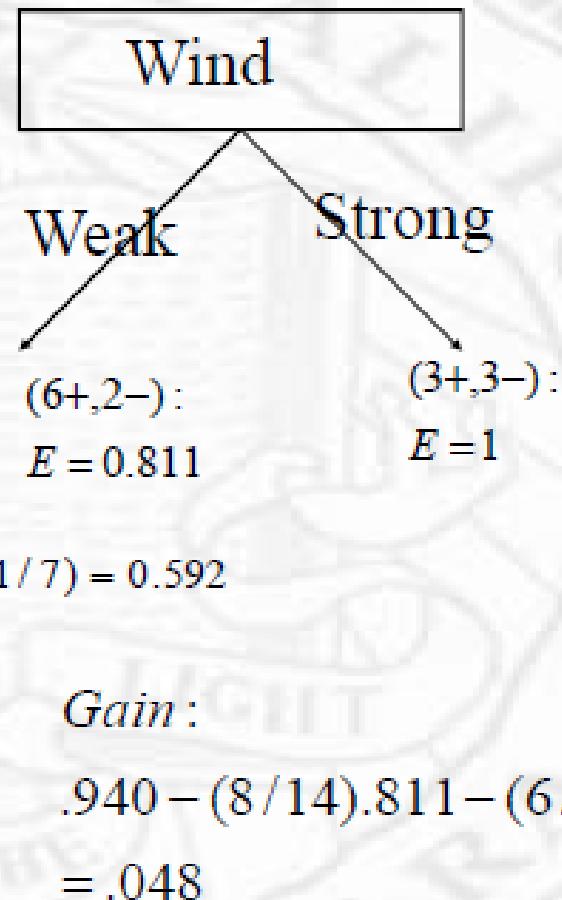
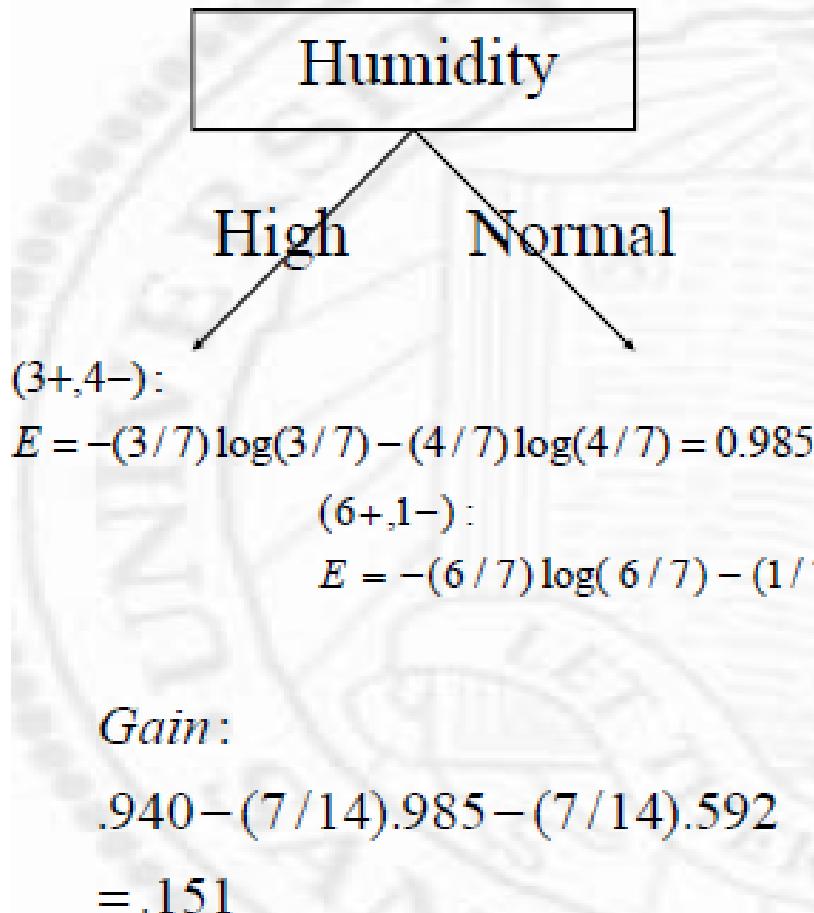


Example

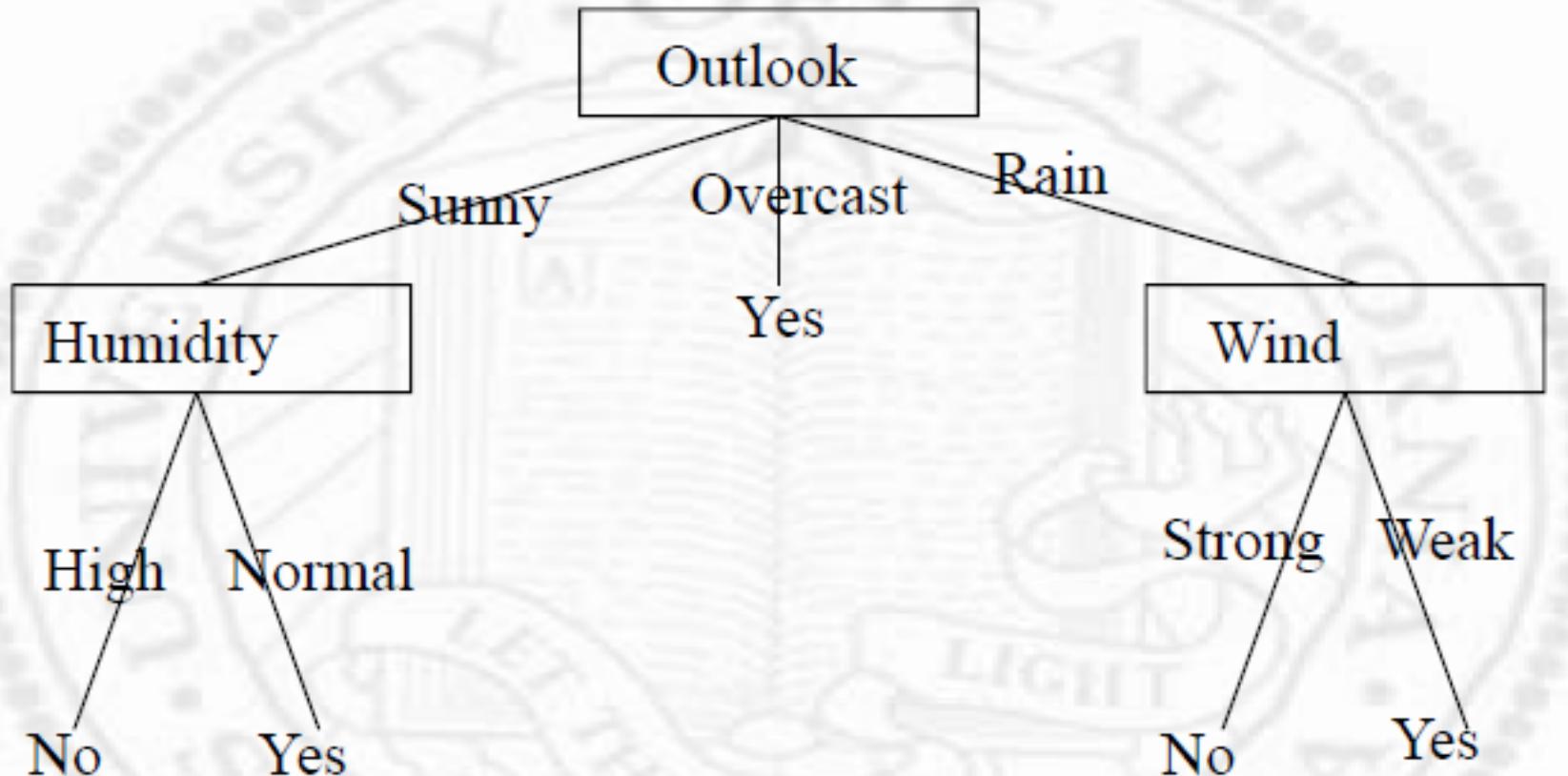
Day	Outlook	Temperature	Humidity	Wind	Play tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cold	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

$$\text{root} : (9+, 5-) : E = -(9/14) \log(9/14) - (5/14) \log(5/14) = 0.94$$



Example



Multiway Splits

- In general, more splits allow impurity to drop
- Splits reduce the samples in each branch
 - With few samples, it is likely that one sample might dominate (1 sample, impurity=0, 2 samples, 50% chance impurity=0)
- Proper scaling of change of impurity
 - Large split is penalized

$$\Delta i(s) = i(N) - \sum_{k=1}^B P_k i(N_k),$$



$$\Delta i_B(s) = \frac{\Delta i(s)}{- \sum_{k=1}^B P_k \log_2 P_k}.$$



Larger the entropy \rightarrow bad split

Bagging – Random Forest

Overfitting

- You can perfectly fit to any training data
- Zero bias, high variance

Two approaches used to solve this for Decision Trees:

1. Stop growing the tree when further splitting the data does not yield an improvement
2. Grow a full tree, then prune the tree, by eliminating nodes.

Yet another approach to reduce variance

Use an ensemble of classifiers.

Two ensemble methods

Bagging: This is known to reduce variance.

Boosting: A weak method is progressively made in to a stronger one. This can reduce bias.

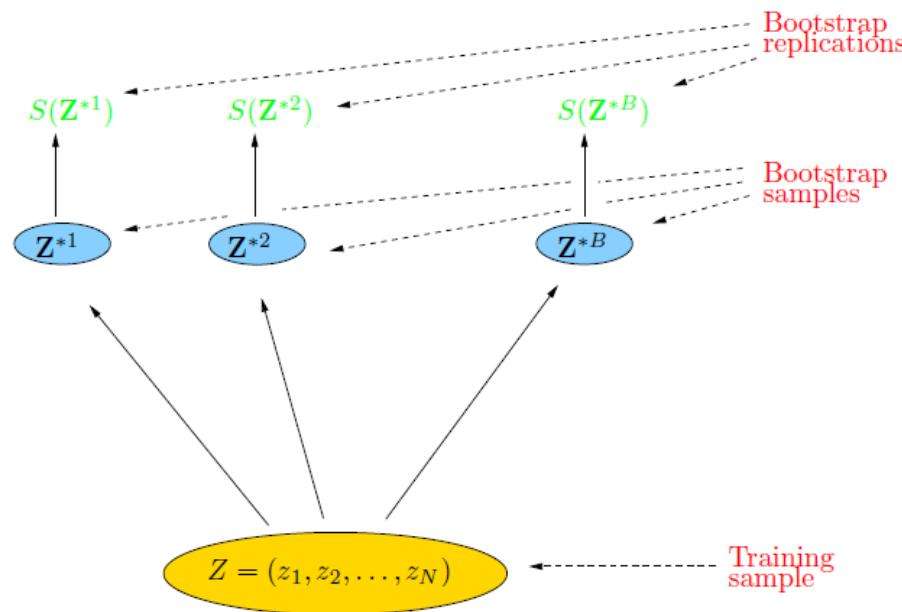
Bagging

- Bagging or *bootstrap aggregation* a technique for reducing the variance of an estimated prediction function.
- For classification, a *committee* of trees each cast a vote for the predicted class.

Bootstrap

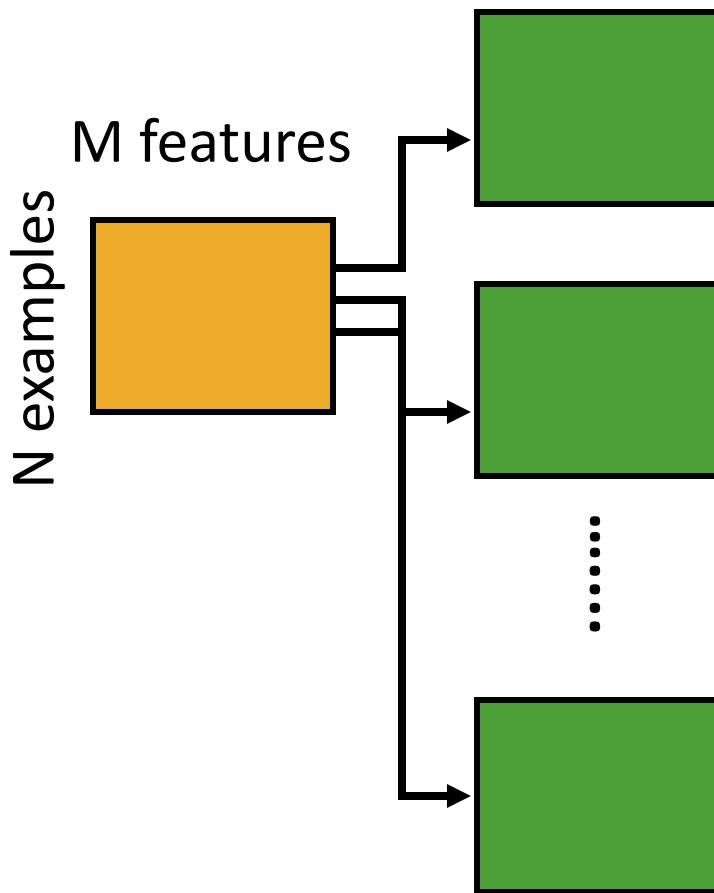
The basic idea:

randomly draw datasets *with replacement* from the training data, each sample *the same size as the original training set*



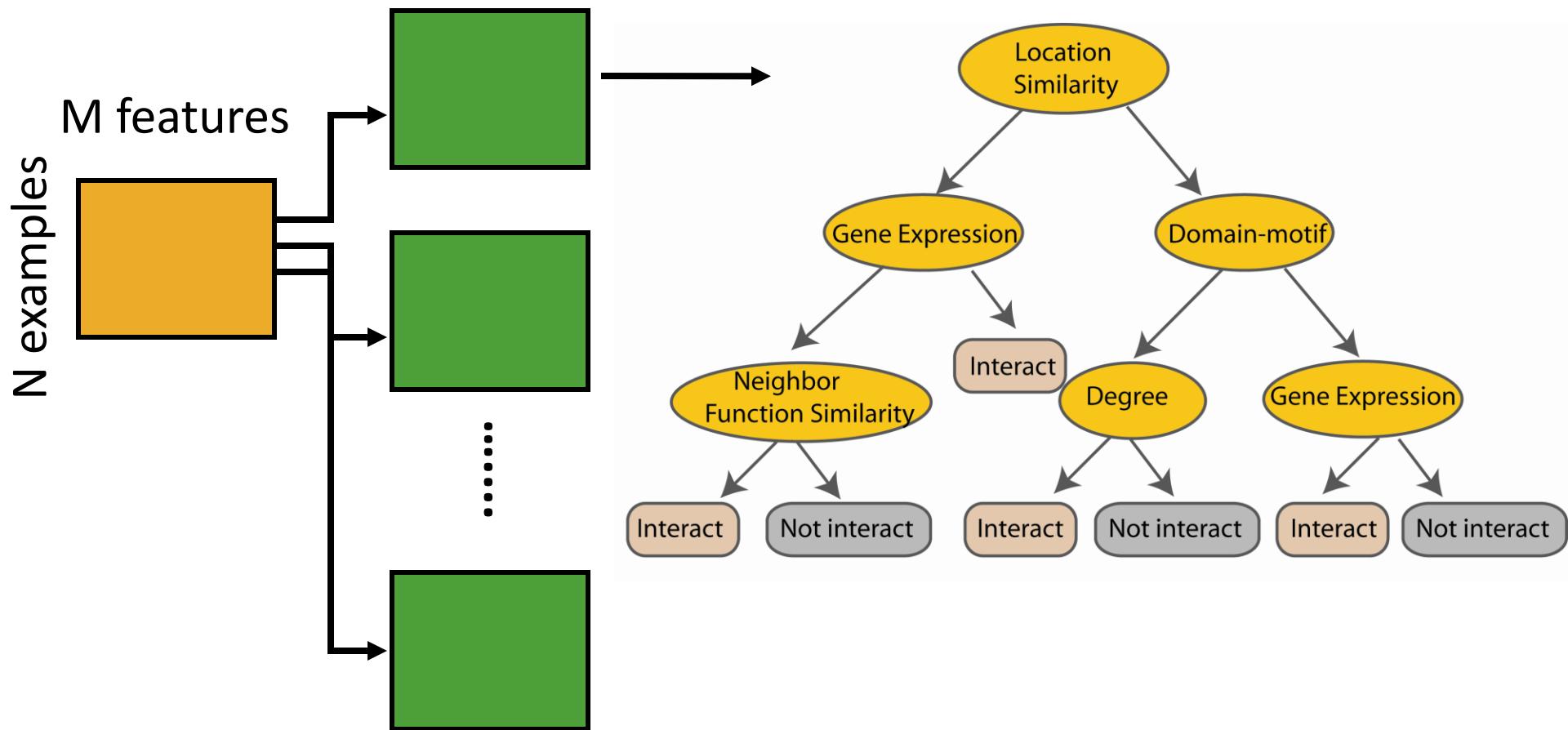
Bagging

Create bootstrap samples
from the training data

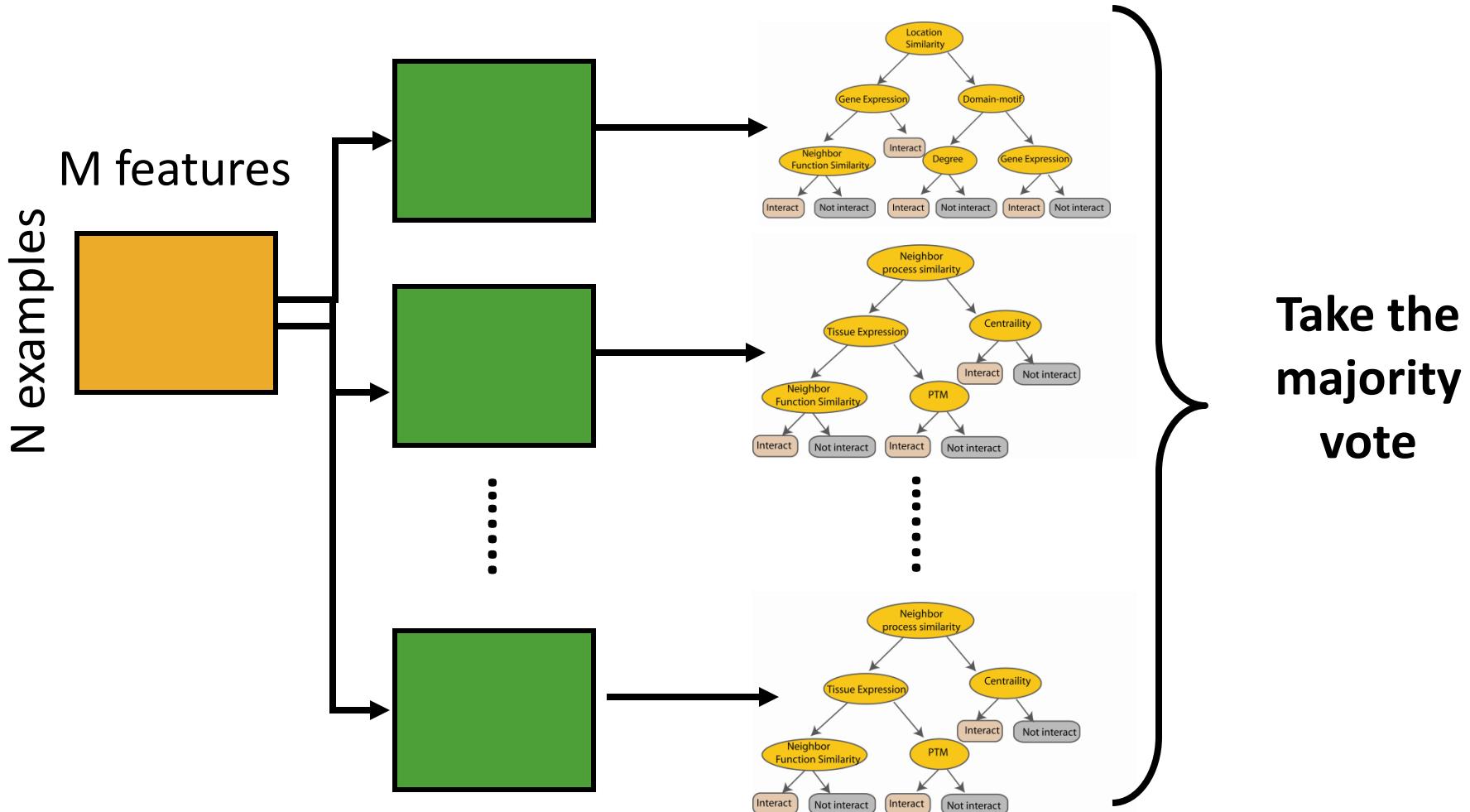


Random Forest Classifier

Construct a decision tree



Random Forest Classifier

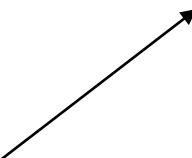


Bagging

$$Z = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

Z^{*b} where $b = 1, \dots, B$..

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x).$$



The prediction at input x when bootstrap sample b is used for training

<http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf> (Chapter 8.7)

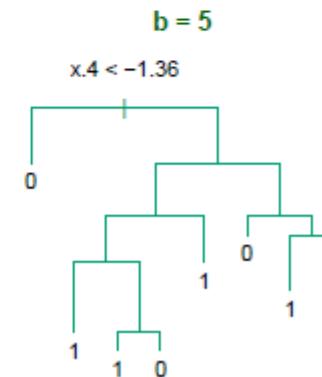
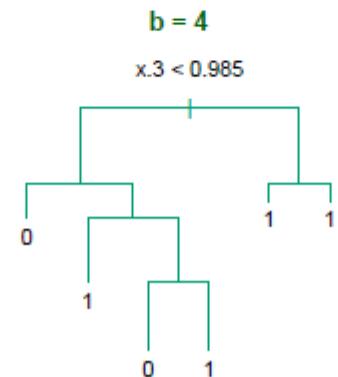
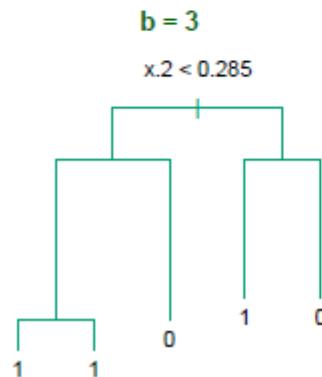
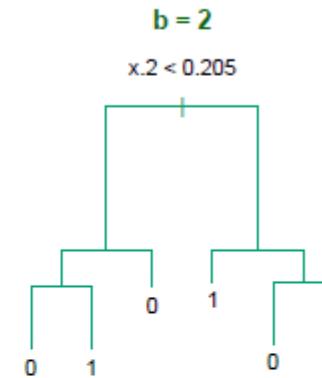
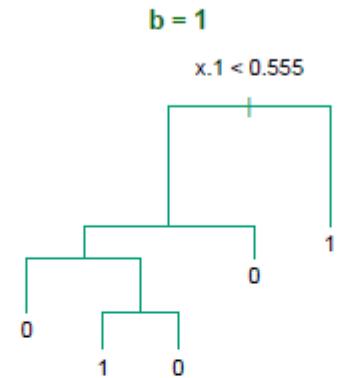
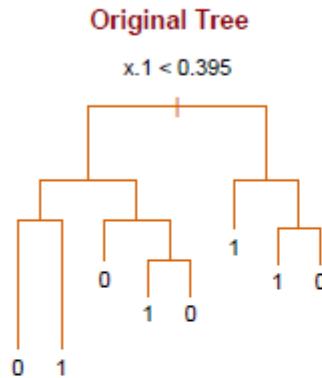
Bagging : an simulated example

Generated a sample of size $N = 30$, with two classes and $p = 5$ features, each having a standard Gaussian distribution with pairwise Correlation 0.95.

The response Y was generated according to
 $\Pr(Y = 1/x_1 \leq 0.5) = 0.2$,
 $\Pr(Y = 0/x_1 > 0.5) = 0.8$.

Bagging

Notice the bootstrap trees are different than the original tree



Bagging

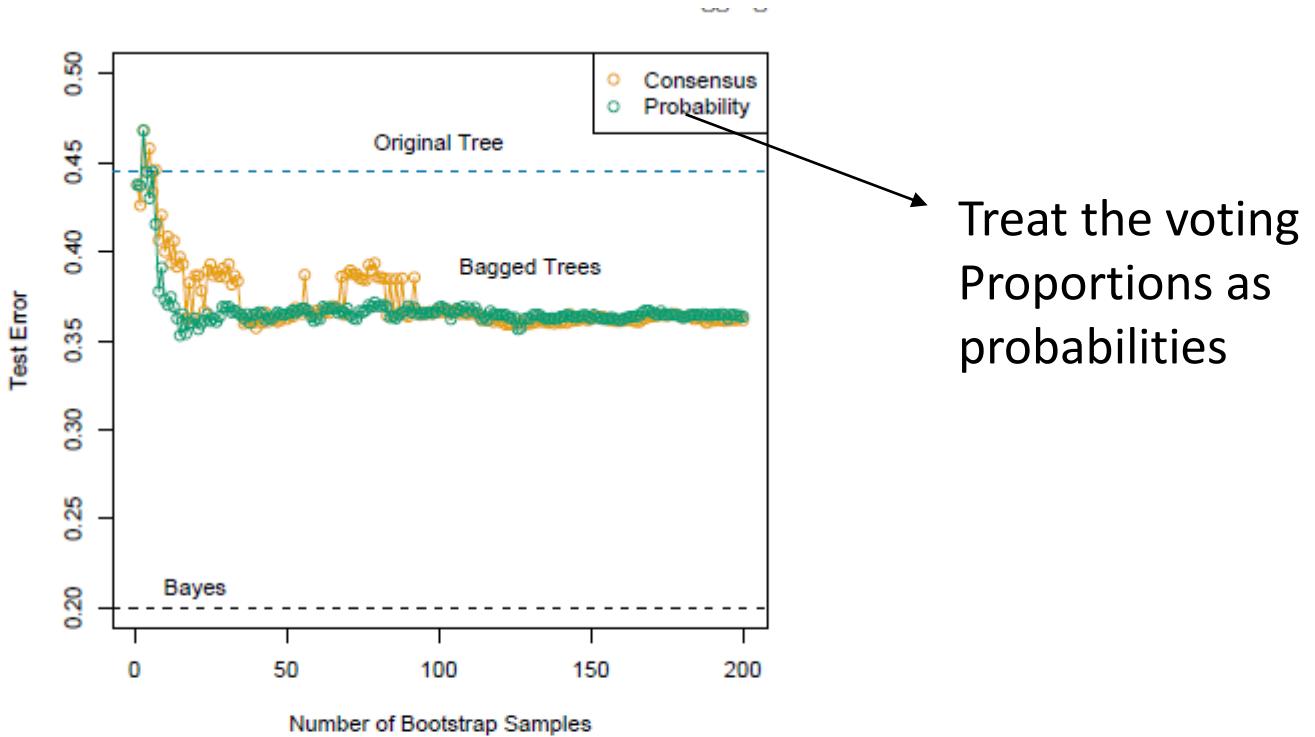


FIGURE 8.10. Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.

bagging helps under squared-error loss, in short because averaging reduces

Hastie

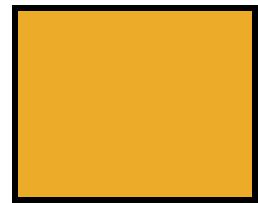
Random forest classifier

Random forest classifier, an extension to bagging which uses *de-correlated* trees.

Random Forest Classifier

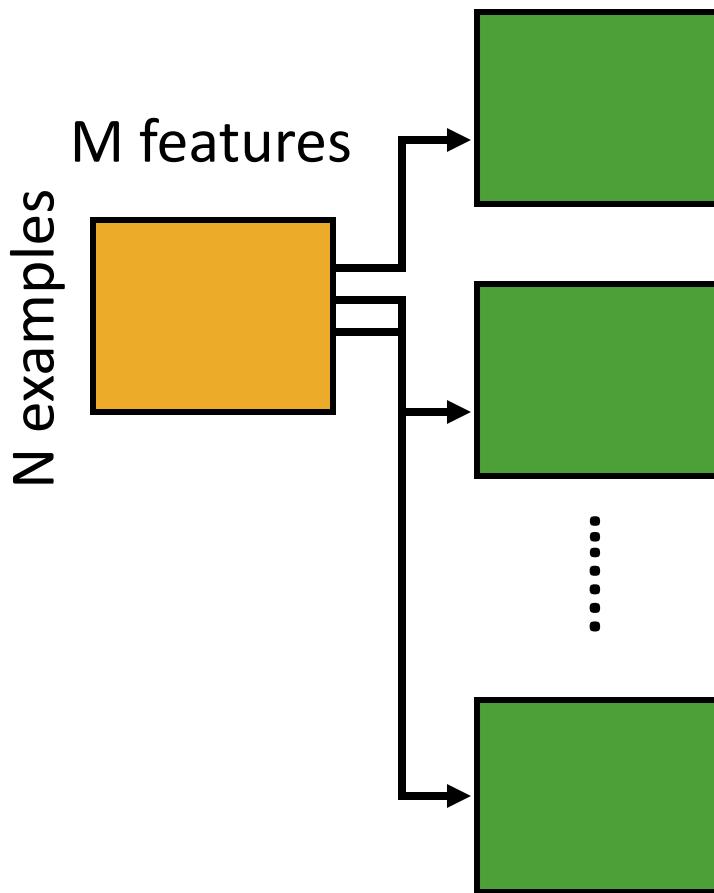
Training Data

M features



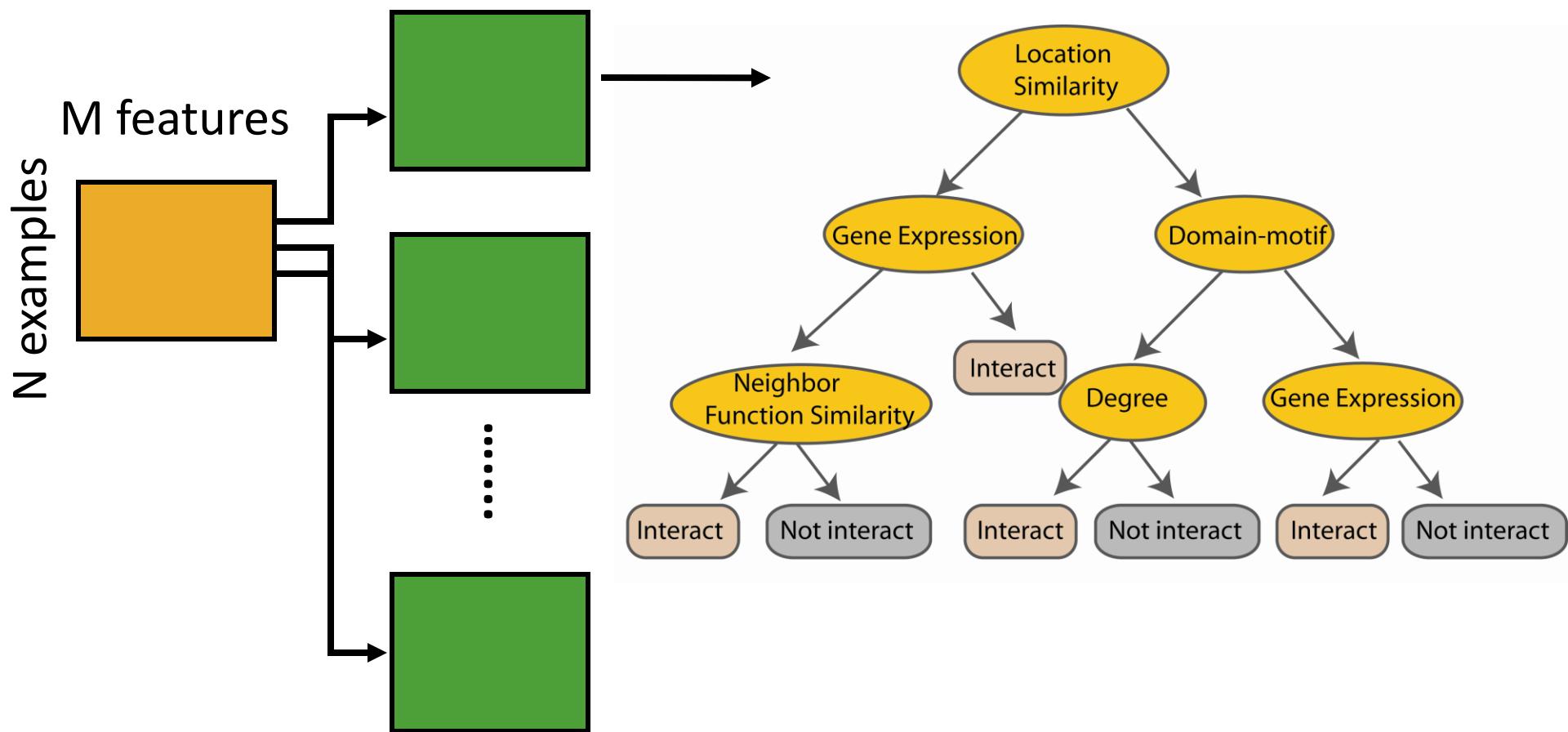
Random Forest Classifier

Create bootstrap samples
from the training data



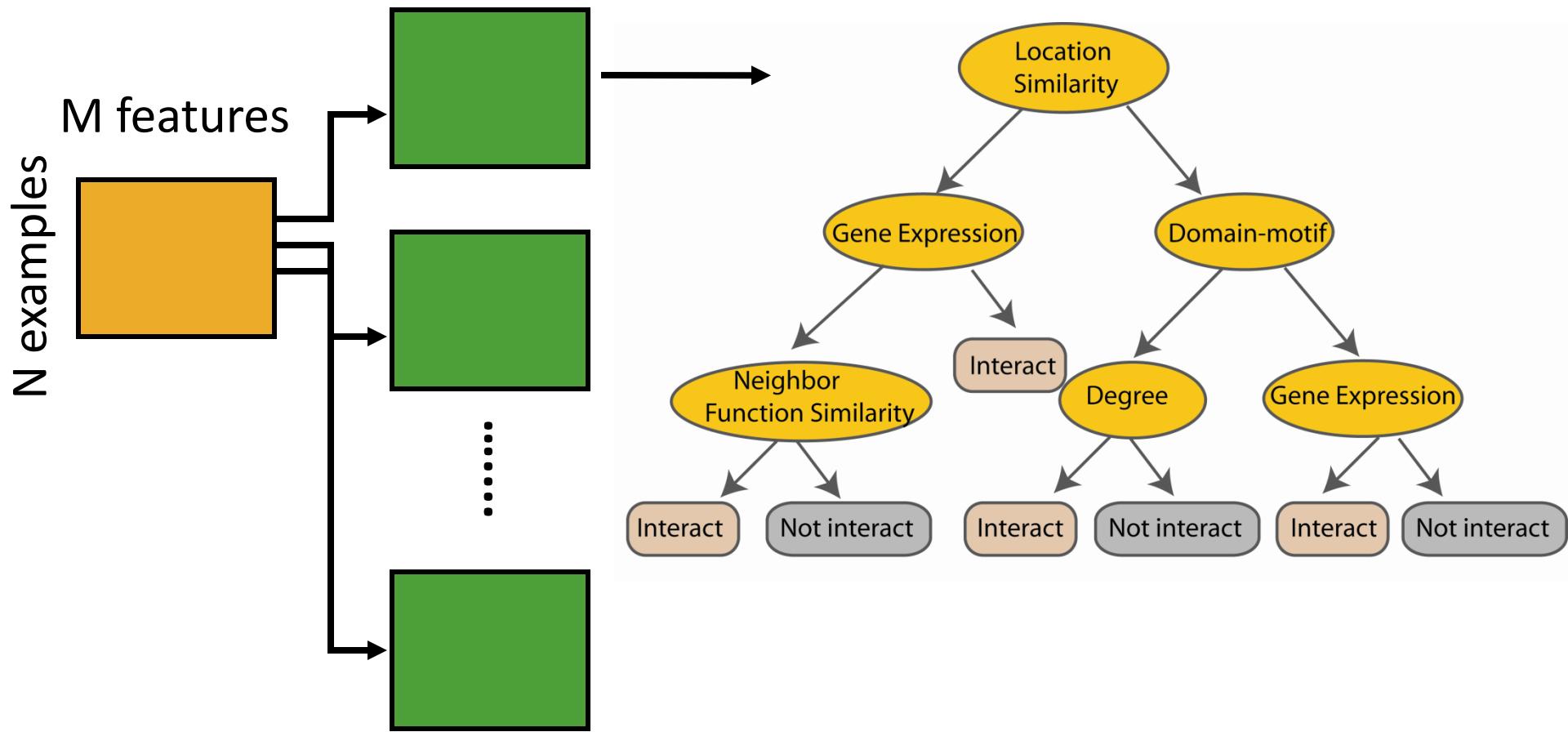
Random Forest Classifier

Construct a decision tree



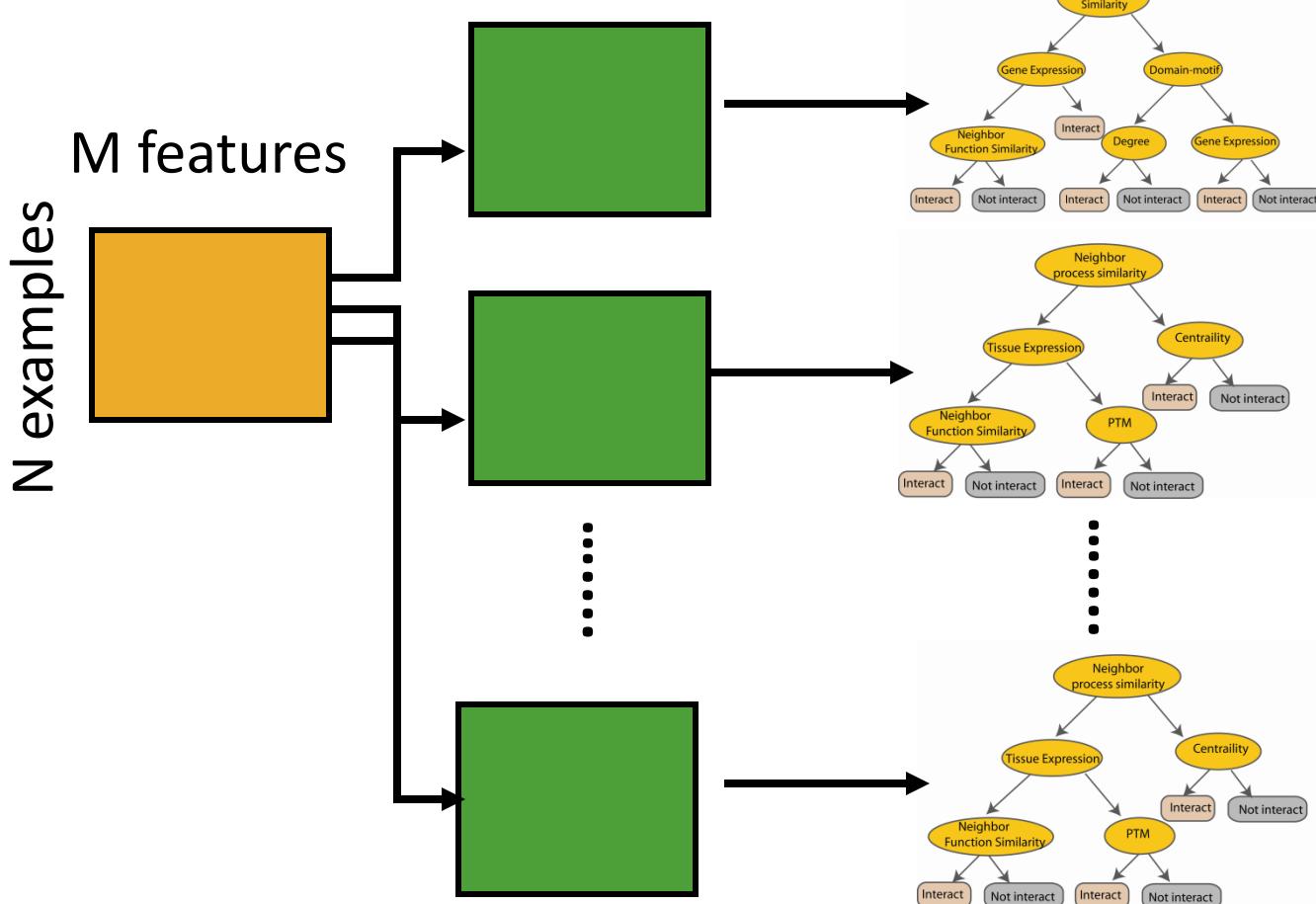
Random Forest Classifier

At each node in choosing the split feature
choose only among $m < M$ features

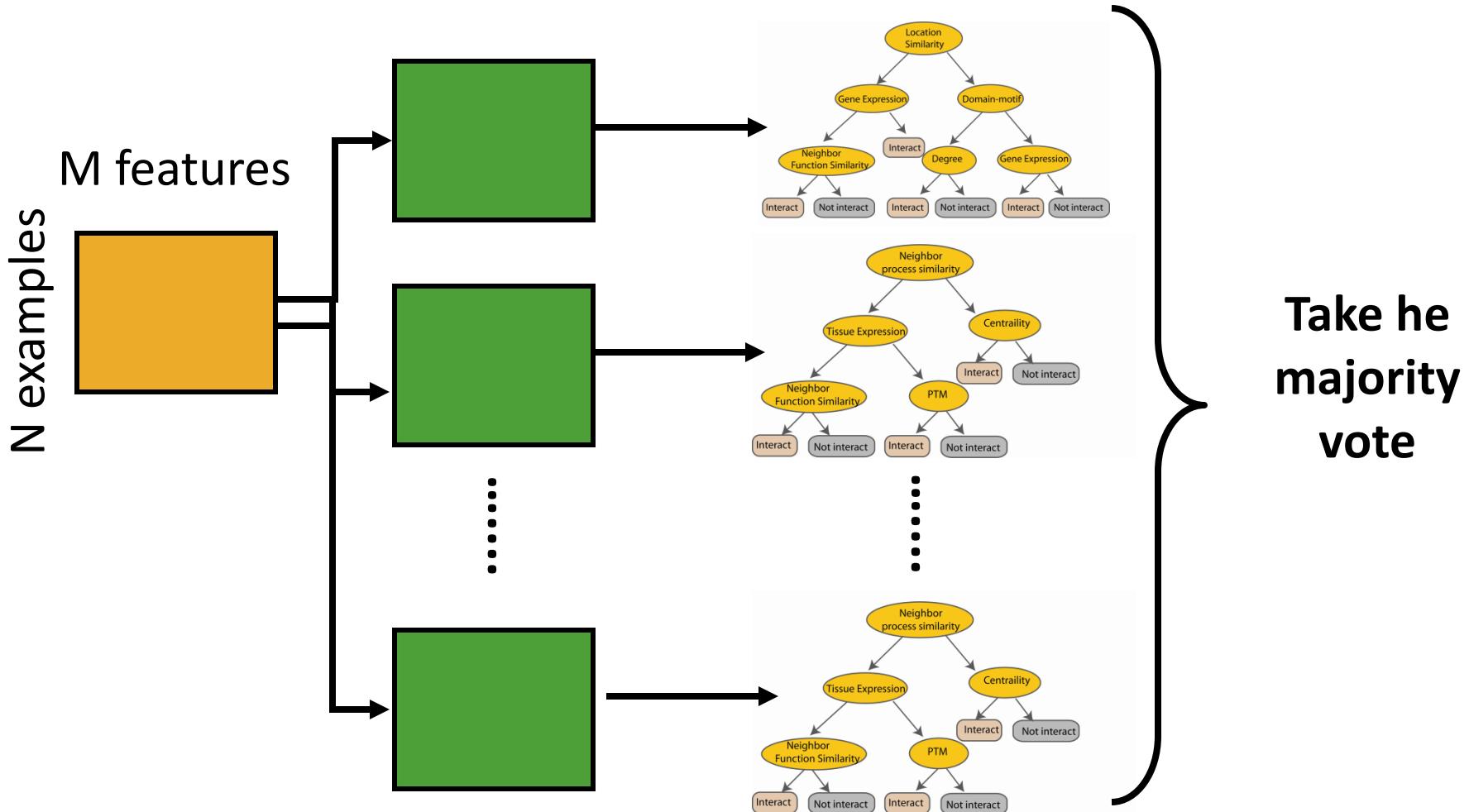


Random Forest Classifier

Create decision tree
from each bootstrap sample



Random Forest Classifier



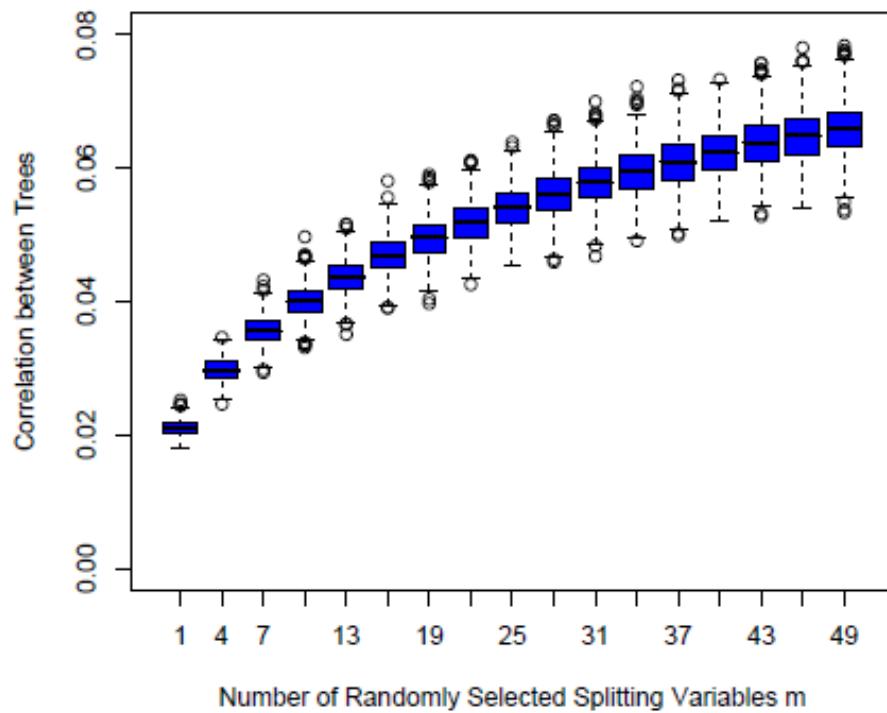


FIGURE 15.9. *Correlations between pairs of trees drawn by a random-forest regression algorithm, as a function of m . The boxplots represent the correlations at 600 randomly chosen prediction points x .*

Random forest

Available package:

http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

To read more:

<http://www-stat.stanford.edu/~hastie/Papers/ESLII.pdf>

Mixture Models and EM

If our data is unlabeled?

- If labeled data (training set) is given:
 - We can extract one class data.
 - We assume the parametric form of the distribution for the class of data. Eg: Gaussian.
 - We can employ maximum likelihood parameter estimation.
- This is what we saw in maximum likelihood parametric density estimation.

Two classes: a and b

- Observations $x_1 \dots x_n$
 - K=2 Gaussians with unknown μ, σ^2
 - estimation trivial if we know the source of each observation



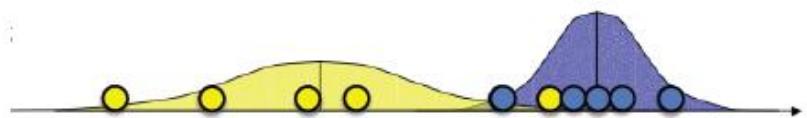
Two classes: a and b

- Observations $x_1 \dots x_n$
 - K=2 Gaussians with unknown μ, σ^2
 - estimation trivial if we know the source of each observation



$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$

$$\sigma_b^2 = \frac{(x_1 - \mu_1)^2 + \dots + (x_n - \mu_n)^2}{n_b}$$



If our data is unlabeled?

- If we know the probability distribution from which the data is drawn,
 - We can label the data ..
 - By employing the Bayes classifier

If our data is unlabeled?

- Distributions are available.

That is, $P(a), P(b), p(x_i|a)$ and $p(x_i|b)$ are given.

Let $p(x_i|a) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$, then

the posterior $P(a|x_i) = \frac{p(x_i|a)P(a)}{p(x_i)}$, and the posterior $P(b|x_i)$ can be used in finding the class label for x_i

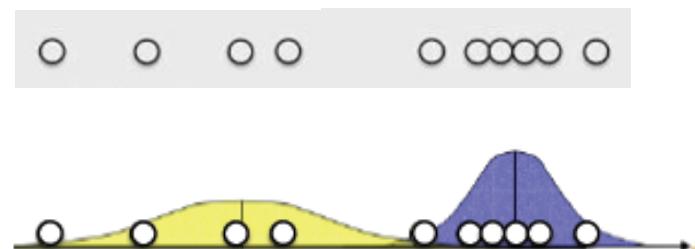
If our data is unlabeled?

- Distributions are available.

That is, $P(a), P(b), p(x_i|a)$ and $p(x_i|b)$ are given.

Let $p(x_i|a) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$, then

the posterior $P(a|x_i) = \frac{p(x_i|a)p(a)}{p(x_i)}$, and the posterior $P(b|x_i)$ can be used in finding the class label for x_i



- Labels are needed to get distributions
- Distributions are needed to get labels.
-

- Labels are needed to get distributions
- Distributions are needed to get labels.
- Chicken and egg problem.



How the nature solved this chicken and egg problem?

- Neither chicken, nor egg was first!
- Both evolved over time.
- Initially very hazy distinction between them, but as time progressed it became two clear distinct things.
- So, we too employ this, but we call this solution **the EM algorithm**.
- Later, we learn that K-means clustering algorithm is a grandson of this algorithm.

Mixture models

- Recall types of clustering methods
 - hard clustering: clusters do not overlap
 - element either belongs to cluster or it does not
 - soft clustering: clusters may overlap
 - strength of association between clusters and instances
- Mixture models
 - probabilistically-grounded way of doing soft clustering
 - each source: a generative model (Gaussian or multinomial)
 - parameters (e.g. mean/covariance are unknown)
- Expectation Maximization (EM) algorithm
 - automatically discover all parameters for the K “sources”

EM with Gaussian assumptions becomes GMM.

Further, GMM, with more assumptions can become K-means ☺

GAUSSIAN MIXTURE MODEL (GMM)

Expectation Maximization (EM)

- Chicken and egg problem
 - need (μ_a, σ_a^2) and (μ_b, σ_b^2) to guess source of points
 - need to know source to estimate (μ_a, σ_a^2) and (μ_b, σ_b^2)

Expectation Maximization (EM)

- Chicken and egg problem
 - need (μ_a, σ_a^2) and (μ_b, σ_b^2) to guess source of points
 - need to know source to estimate (μ_a, σ_a^2) and (μ_b, σ_b^2)

- EM algorithm
 - Start with two randomly placed Gaussians (μ_a, σ_a^2) , (μ_b, σ_b^2) .
 - While (not converged) do
 - E-step: Find $P(a|x_i), P(b|x_i)$ for each data element. This gives label for x_i . Fishy: This label is a random variable !
 - M-step: Adjust (μ_a, σ_a^2) and (μ_b, σ_b^2) to fit points assigned to them.

EM: 1-d example

Source parameters are randomly fixed to begin with.



$$p(x_i|a) = \frac{1}{\sqrt{2\pi\sigma_a^2}} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$$

$$a_i = P(a|x_i) = \frac{p(x_i|a)P(a)}{p(x_i)}$$

$$b_i = 1 - a_i$$

(a_i, b_i) is the label for x_i



EM: 1-d example



(a_i, b_i) is the label for x_i

Prior $P(a)$ can be estimated from $\frac{a_1 + a_2 + \dots + a_n}{n}$

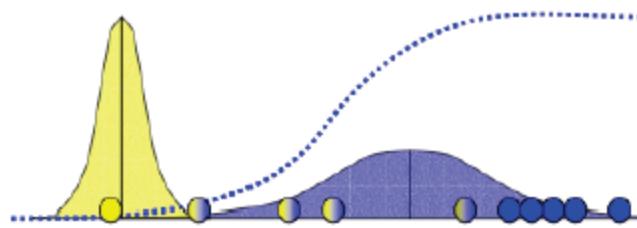
So, $P(b) = \frac{b_1 + b_2 + \dots + b_n}{n}$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \dots + a_n x_{n_b}}{a_1 + a_2 + \dots + a_n}$$

$$\sigma_a^2 = \frac{a_1 (x_1 - \mu_a)^2 + \dots + a_n (x_n - \mu_a)^2}{a_1 + a_2 + \dots + a_n}$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \dots + b_n x_{n_b}}{b_1 + b_2 + \dots + b_n}$$

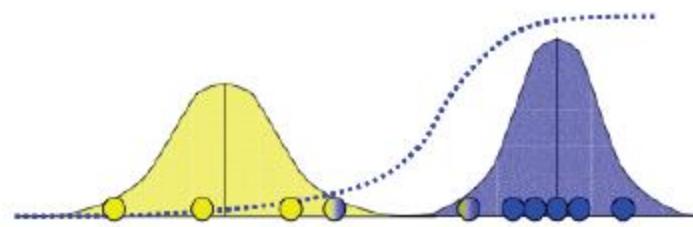
$$\sigma_b^2 = \frac{b_1 (x_1 - \mu_b)^2 + \dots + b_n (x_n - \mu_b)^2}{b_1 + b_2 + \dots + b_n}$$



Now we are with a new estimation of the source.
A better estimate. Repeat till convergence.

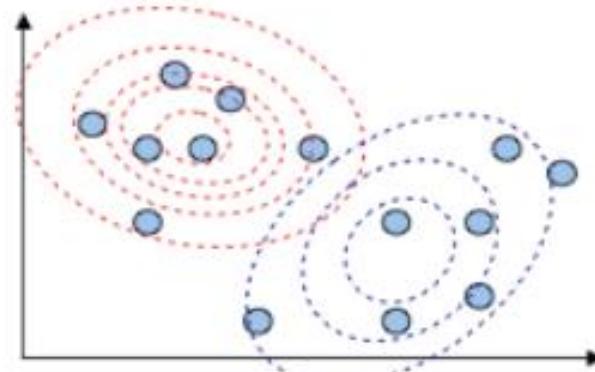
EM: 1-d example

after convergence, the output:



Extension to $d > 1$, $c > 2$

- Assume c component mixture (c classes).
- Start with randomly chosen means (randomly choose k distinct data elements).
- Similarly randomly chosen covariance matrix for each class. Usually we begin with identity matrix.



E-step: Find label for each x_i . Let this be the random variable given by $(P_{1i}, P_{2i}, \dots, P_{ci})$. This is done for each $x_i, 1 \leq i \leq n$.

E-step: Find label for each x_i . Let this be the random variable given by $(P_{1i}, P_{2i}, \dots, P_{ci})$. This is done for each $x_i, 1 \leq i \leq n$.

M-step: Let $\mu^{(1)}$ be the mean of class 1. Then, $\mu^{(1)} = \frac{\sum_{i=1}^n P_{1i}x_i}{\sum_{i=1}^n P_{1i}}$. Similarly mean vector for other classes, $\mu^{(2)}, \dots, \mu^{(c)}$ can be found.

E-step: Find label for each x_i . Let this be the random variable given by $(P_{1i}, P_{2i}, \dots, P_{ci})$. This is done for each $x_i, 1 \leq i \leq n$.

M-step: Let $\mu^{(1)}$ be the mean of class 1. Then, $\mu^{(1)} = \frac{\sum_{i=1}^n P_{1i} x_i}{\sum_{i=1}^n P_{1i}}$. Similarly mean vector for other classes, $\mu^{(2)}, \dots, \mu^{(c)}$ can be found.

Covariance Matrix for class 1, $\Sigma^{(1)} = \frac{\sum_{i=1}^n P_{1i} (x_i - \mu^{(1)}) (x_i - \mu^{(1)})^t}{\sum_{i=1}^n P_{1i}}$. Similarly covariance matrix for other classes, $\Sigma^{(2)}, \dots, \Sigma^{(c)}$ can be found.

- We stop EM algorithm here.
- In exams, I can ask some numeric problem for 1D two class case. {Do not worry about multidimensional problem (as of now)}.
- Theoretically, the iterative process can get stuck in a local maximum.

K-means is an approximation of GMM

- Initially pick k distinct random seed points (in GMM: the set of initial mean vectors)
- We assume that

$$\boldsymbol{\Sigma}^{(1)} = \boldsymbol{\Sigma}^{(2)} = \dots = \boldsymbol{\Sigma}^{(c)} = \mathbf{I}$$

- The Bayes classifier becomes “the minimum distance classifier”.
- Let the label be deterministic (not a random variable). Choose the nearest’s mean’s label (this is what the minimum distance classifier will do).
- GMM becomes k-means clustering algorithm.

GMM: Grandfather

K-Means: Grandson



***K*-means Clustering**

- Partitioning Clustering Approach
 - a typical clustering analysis approach via **iteratively** partitioning training data set to learn a partition of the given data space
 - learning a partition on a data set to produce several non-empty clusters (usually, the number of clusters given in advance)
 - in principle, optimal partition achieved via **minimising the sum of squared distance to its “representative object”** in each cluster

$$E = \sum_{k=1}^K \sum_{\mathbf{x} \in C_k} d^2(\mathbf{x}, \mathbf{m}_k)$$

e.g., Euclidean distance $d^2(\mathbf{x}, \mathbf{m}_k) = \sum_{n=1}^N (x_n - m_{kn})^2$

- Given a K , find a partition of K clusters to optimise the chosen partitioning criterion (cost function)
 - global optimum: exhaustively search all partitions
- The *K-means* algorithm: a heuristic method
 - K-means algorithm (MacQueen'67): each cluster is represented by the centre of the cluster and the algorithm converges to stable centroids of clusters.
 - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications.

K-means Algorithm

- Given the cluster number K , the *K-means* algorithm is carried out in three steps after initialisation:

Initialisation: set seed points (randomly)

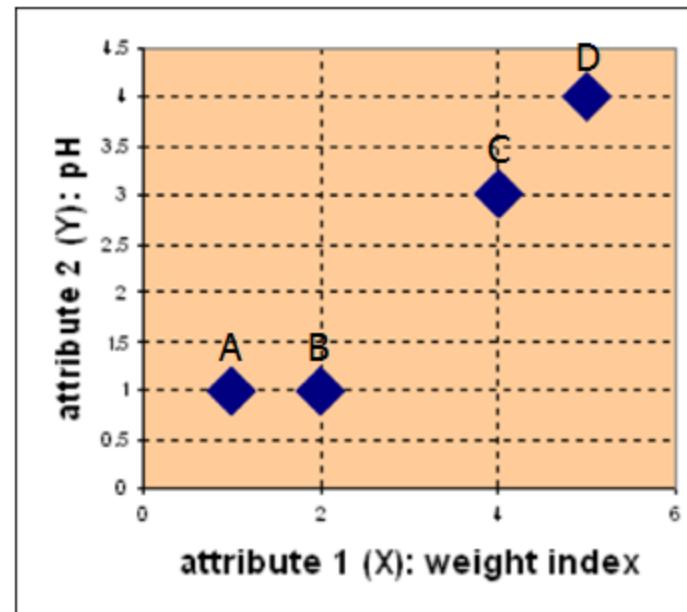
- 1) Assign each object to the cluster of the nearest seed point measured with a specific distance metric
- 2) Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., *mean point*, of the cluster)
- 3) Go back to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

Example

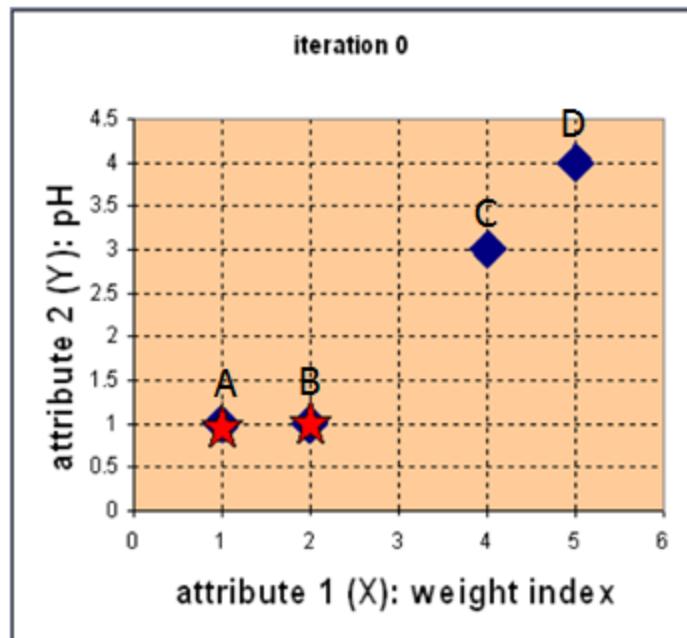
- Problem

Suppose we have 4 types of medicines and each has two attributes (pH and weight index). Our goal is to group these objects into $K=2$ group of medicine.

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Example



Step 1: Use initial seed points for partitioning

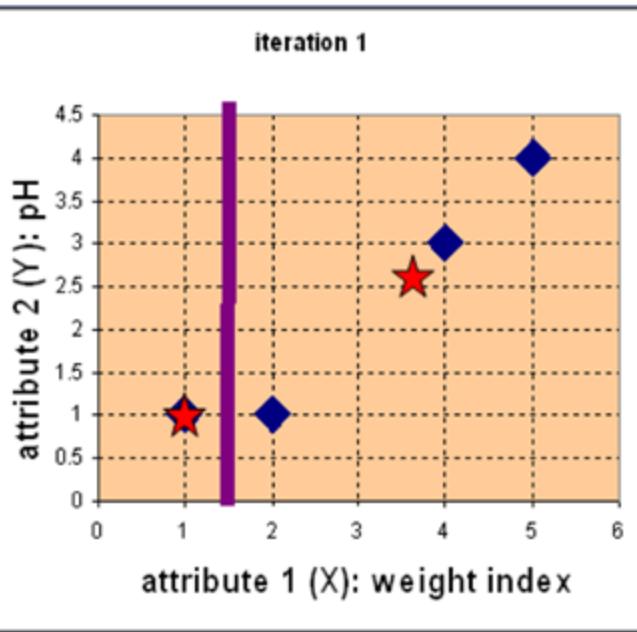
$$c_1 = A, c_2 = B$$

Step 2 Assign each object to the cluster with the nearest seed point

$$d(D, c_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$d(D, c_2) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24$$

Step 3 Compute new centroids of the current partition



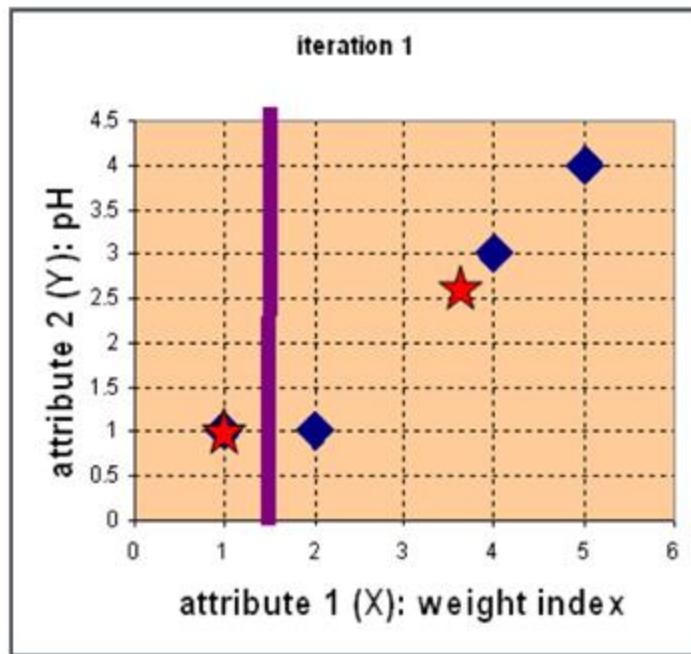
Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

$$c_1 = (1, 1)$$

$$\begin{aligned} c_2 &= \left(\frac{2 + 4 + 5}{3}, \frac{1 + 3 + 4}{3} \right) \\ &= \left(\frac{11}{3}, \frac{8}{3} \right) \end{aligned}$$

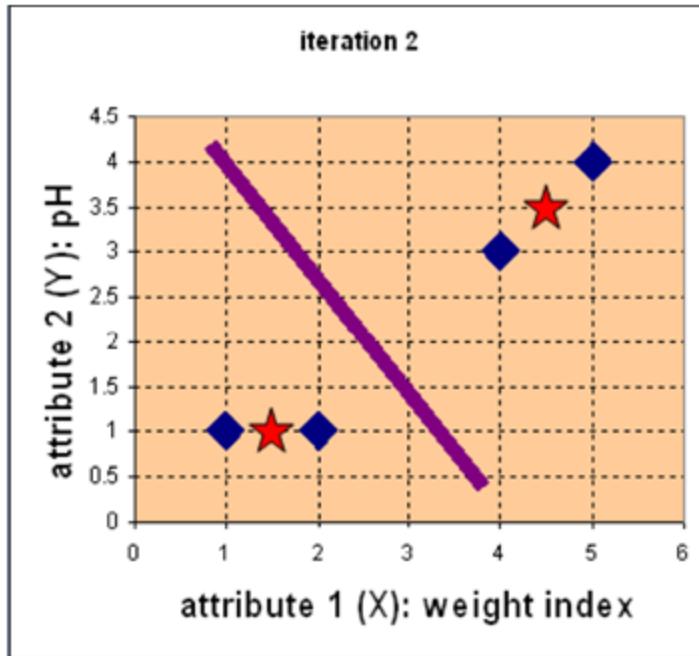
Step 2 Assign each object to the cluster with the nearest seed point

Renew membership based on new centroids



Step 3 Compute new centroids of the current partition

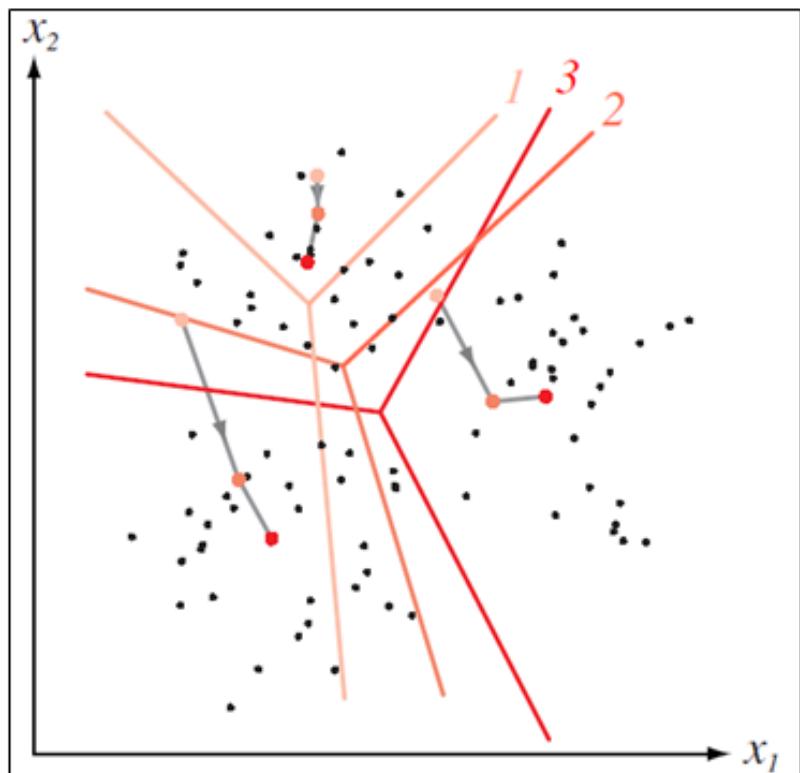
Repeat until its convergence



$$c_1 = \left(\frac{1+2}{2}, \frac{1+1}{2} \right) = \left(1\frac{1}{2}, 1 \right)$$

$$c_2 = \left(\frac{4+5}{2}, \frac{3+4}{2} \right) = \left(4\frac{1}{2}, 3\frac{1}{2} \right)$$

How K-means partitions?



When K centroids are set/fixed, they partition the whole data space into K mutually exclusive subspaces to form a partition.

A partition amounts to a
Voronoi Diagram

Changing positions of centroids leads to a new partitioning.

Relevant Issues

- Efficient in computation
 - $O(tKn)$, where n is number of objects, K is number of clusters, and t is number of iterations. Normally, $K, t \ll n$.
- Local optimum
 - sensitive to initial seed points
 - converge to a local optimum: maybe an unwanted solution
- Other problems
 - Need to specify K , the *number* of clusters, in advance
 - Unable to handle noisy data and outliers (*K-Medoids* algorithm)
 - Not suitable for discovering clusters with non-convex shapes
 - Applicable only when mean is defined, then what about categorical data? (*K-mode* algorithm)
 - how to evaluate the *K-mean* performance?

Summary

- *K-means* algorithm is a simple yet popular method for clustering analysis
- Its performance is determined by initialisation and appropriate distance measure
- There are several *variants* of *K-means* to overcome its weaknesses
 - *K-Medoids*: resistance to noise and/or outliers
 - *K-Modes*: extension to categorical data clustering analysis
 - CLARA: extension to deal with large data sets
 - Mixture models (EM algorithm): handling uncertainty of clusters

Multilayer Perceptron

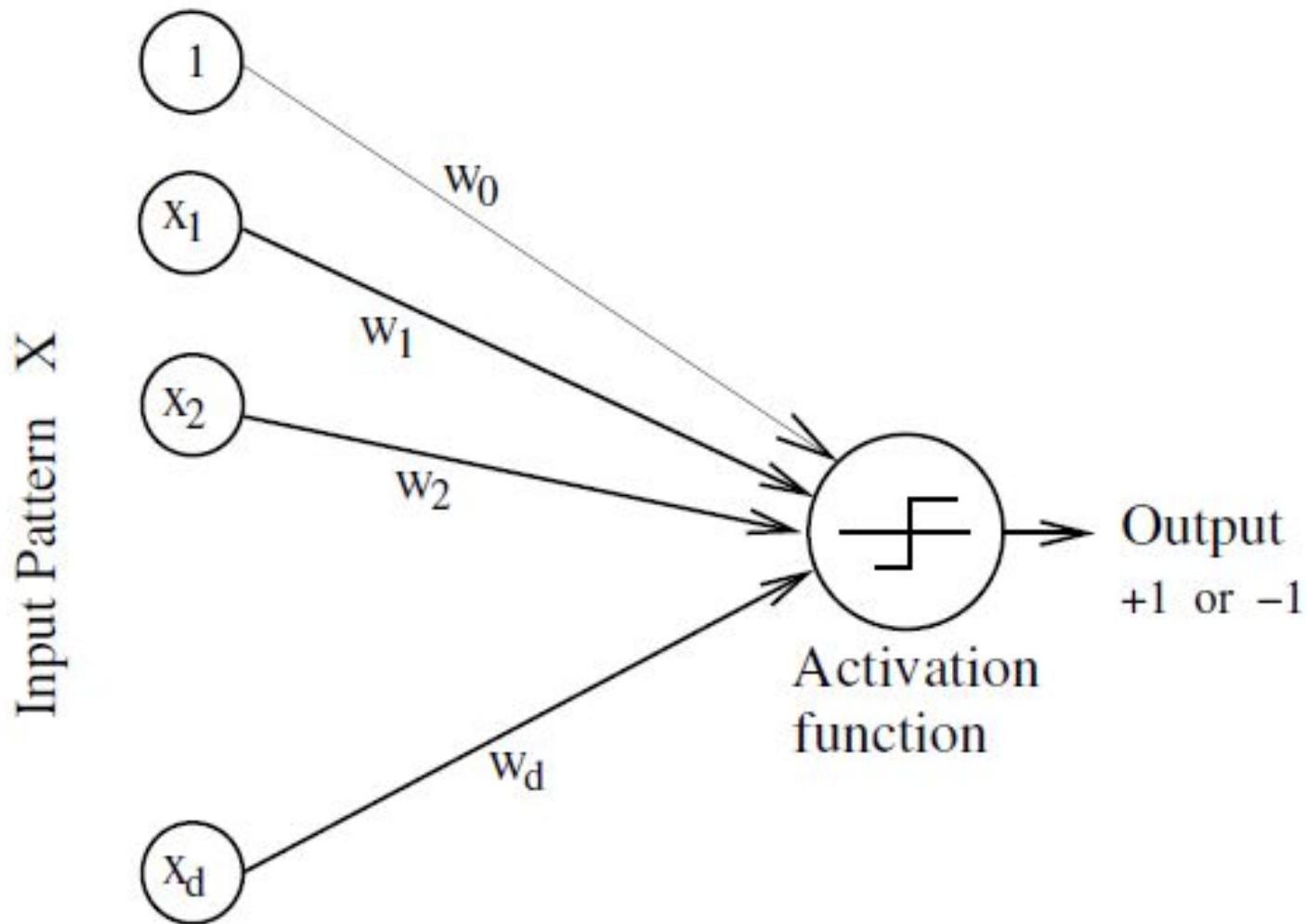
Multilayer Feed-forward Neural
Network

Non-linear discriminants

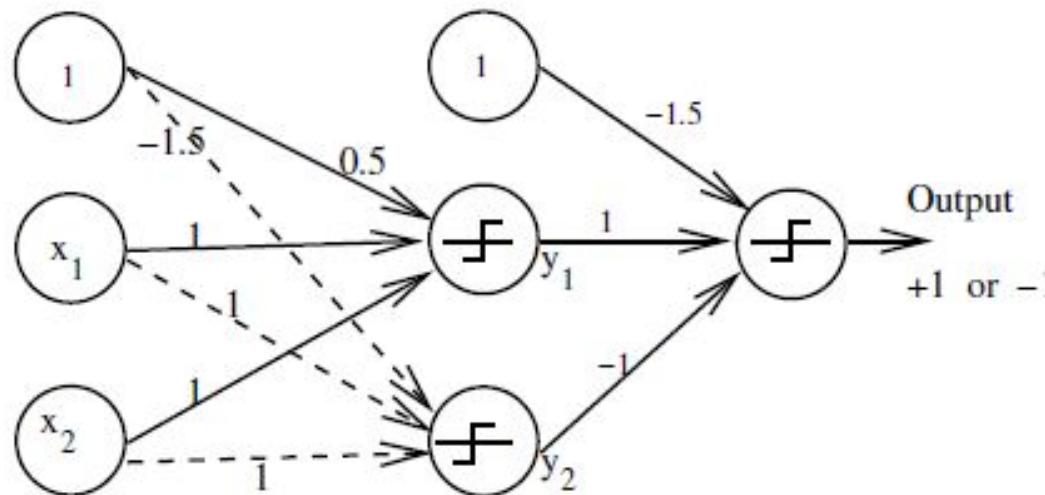
- Non-linear discriminants are more powerful than linear discriminants.
- The number of parameters to be learned are larger than that with linear discriminants and hence can create some problems.
- These can overcome the drawbacks of the single ^a layer networks (Perceptrons).
- One of the popular methods for training a multilayer network is based on gradient descent procedure called the *backpropagation algorithm*.

^aSome say Perceptron has single layer others say it has two layers

Two layer network



Three layer network : XOR Problem



x_1	x_2	y_1	y_2	Output
+1	+1	+1	+1	-1
+1	-1	+1	-1	+1
-1	+1	+1	-1	+1
-1	-1	-1	-1	-1

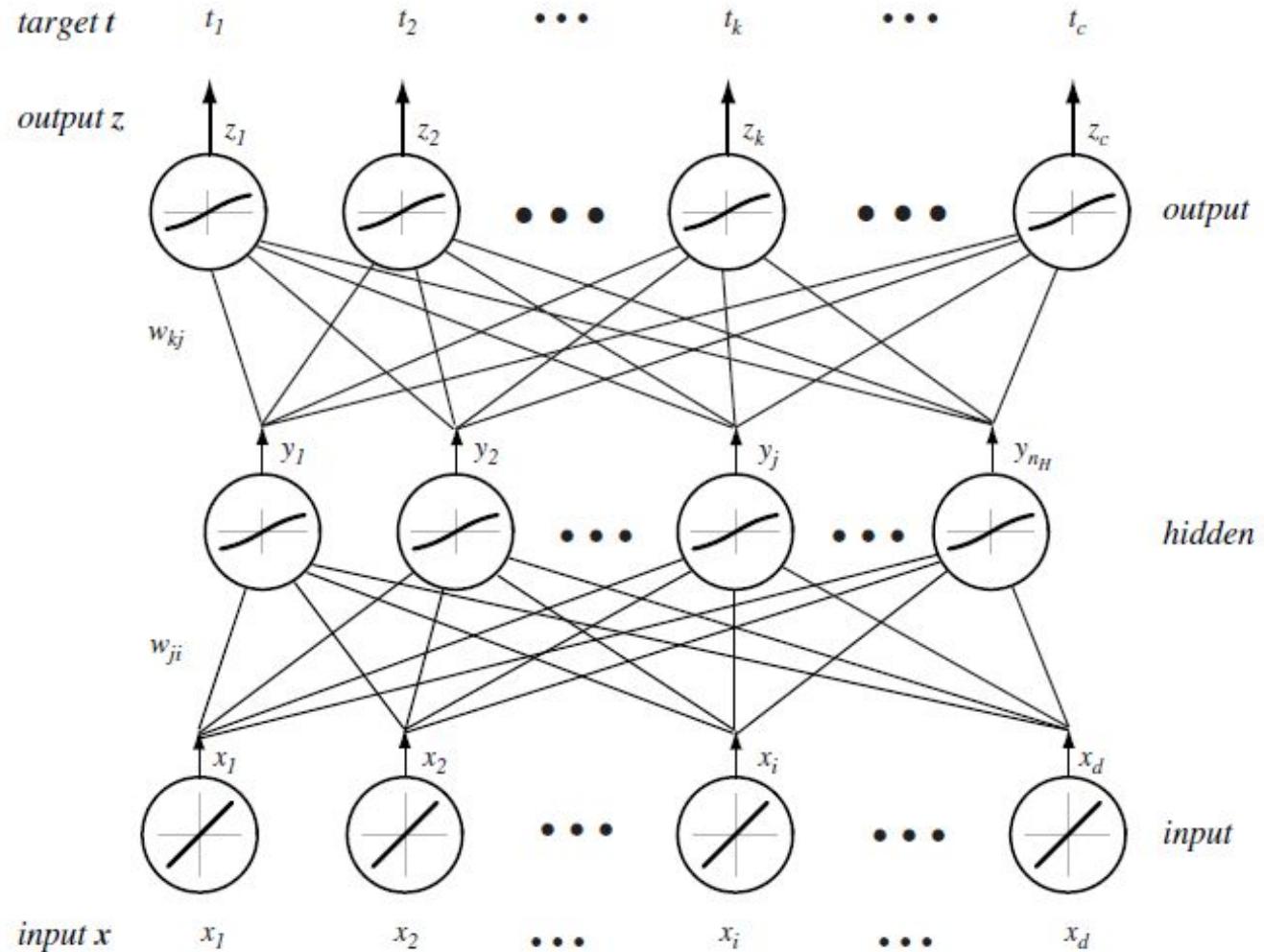
Multilayer Networks

- With sufficient number of hidden units, Multilayer networks can represent any function.

Indeed, a three layer network with sufficient number of hidden units can implement any function. This is mathematically proved.

- To generalize it to the c class problem, the number of output units are c where each one computes the discriminant function $g_k(X)$.
- The activation function need not be the only *sign (Signum)* function. Indeed we often require the activation function to be continuous and differentiable. Also, activation functions can be different for different units.

Three layer network

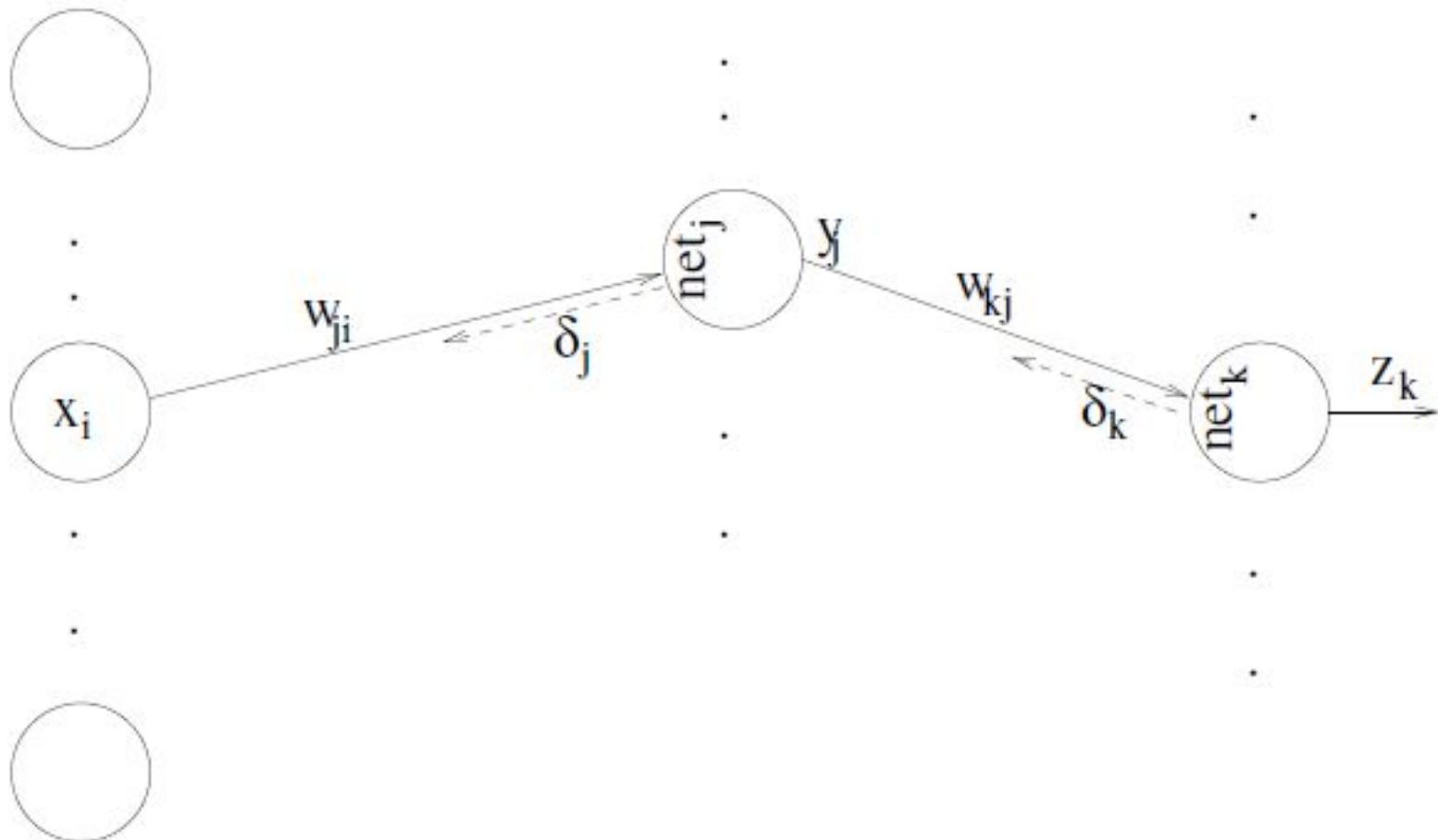


Three layer network

- Let the weight connecting the i th input unit to the j th hidden unit be w_{ji} . Similarly, let the weight connecting the j th hidden unit to the k th output unit be w_{kj} .
- Let the number of hidden units are n_H .
- Let the output from the k th output unit be z_k .
- Then the discriminant $g_k(X)$ is:

$$g_k(X) \equiv z_k = f \left(\sum_{j=1}^{n_H} w_{kj} f \left(\sum_{i=1}^d w_{ji} x_i + w_{j0} \right) + w_{k0} \right)$$

Error Backpropagation



Error Backpropagation

- Let the desired output (target) of the output unit k be t_k .
- The actual output of the unit be z_k .
- The objective,

$$J(W) = \frac{1}{2} \sum_{r=1}^c (t_r - z_r)^2$$

- We apply gradient descent to get a W for which $J(W)$ is minimum.
- If w is the weight of an edge then the update rule after the m^{th} iteration should be

$$w_{m+1} = w_m - \eta \frac{\partial J}{\partial w} = w_m + \Delta w$$

Error Backpropagation

Update rule for weights between hidden and output units:

- Let w_{kj} be the weight between j^{th} hidden node and k^{th} output node.
- We need to find $\partial J / \partial w_{kj}$
- $J(W) = \frac{1}{2} \sum_{r=1}^c (t_r - z_r)^2$, So $\frac{\partial J}{\partial z_k} = -(t_k - z_k)$
- $$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial \text{net}_k} \underbrace{\frac{\partial \text{net}_k}{\partial w_{kj}}}_{y_j} = \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial \text{net}_k} y_j = \underbrace{-(t_k - z_k) f'(\text{net}_k)}_{\frac{\partial J}{\partial \text{net}_k} = -\delta_k} y_j = -\delta_k y_j$$
- $\Delta w_{kj} = \eta y_j \delta_k$

Error Backpropagation

Update rule for weights between input and hidden units:

- Let w_{ji} be the weight between i^{th} input node and j^{th} hidden node.

$$\begin{aligned}\frac{\partial J}{\partial w_{ji}} &= \underbrace{\frac{\partial J}{\partial net_j}}_{-\delta_j} \underbrace{\frac{\partial net_j}{\partial w_{ji}}}_{x_i} = \left(\sum_{r=1}^c \underbrace{\frac{\partial J}{\partial \text{net}_r}}_{-\delta_r} \underbrace{\frac{\partial \text{net}_r}{\partial y_j}}_{w_{rj}} \underbrace{\frac{\partial y_j}{\partial \text{net}_j}}_{f'(\text{net}_j)} \right) x_i \\ &= \underbrace{\sum_{r=1}^c -\delta_r w_{rj} f'(\text{net}_j) x_i}_{\frac{\partial J}{\partial net_j} = -\delta_j}\end{aligned}$$

- $\Delta w_{ji} = \eta x_i \delta_j$

Learning – training the network

- We will call the process of running 1 example through the network (and training the network on that 1 example) a **weight update iteration**.
- Training the network once on each example of your training set is called an **epoch**. Typically, you have to train the network for many epochs before it converges.

Training Protocols

- The exact behavior of the backpropagation depends on the starting point.
- We cannot start with $W = 0$, i.e., all weights being zeros. Because the updates will be zero and the training cannot proceed.
- So one should start with a random weight vector.
 - Two common approaches
 1. Offline or Batch Protocol
 2. Online Protocol

Batch Backpropagation

- In the batch training protocol, all the training patterns are presented first and their corresponding weight updates summed; only then are the actual weights in the network are updated.
- Stopping criteria:
 - When weight updation between successive epochs is small enough.
 - Or, overall error accumulated for all training examples is small enough.

Online or Stochastic Gradient Descent

- For each example –
 - Present the example to the network
 - update the weights.
 - Goto the next example.
- When all examples are presented, we say one epoch is completed.

Activation Function

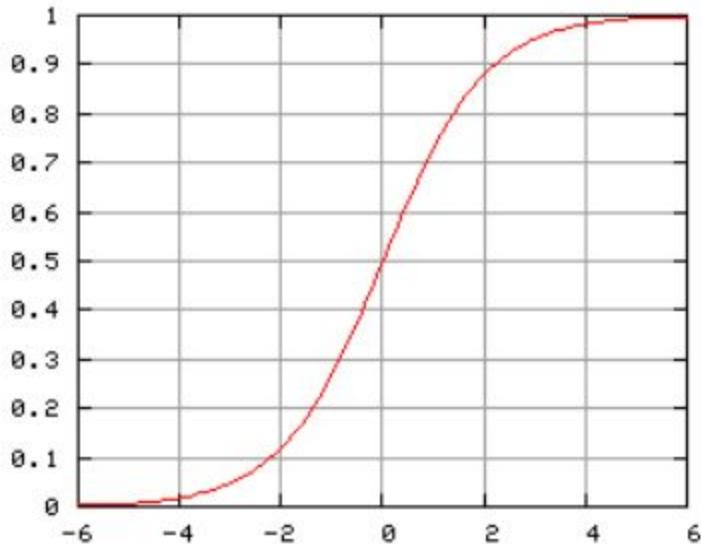
- Backpropagation will work with any activation function $f(\cdot)$, provided that it is continuous and differentiable.
- But the activation function needs to be non-linear, otherwise it will not have any computational power over Perceptrons.
- A second desirable property is that $f(\cdot)$ has some bounded range. It should have some limited values for maximum and minimum. This will keep the weights to be bounded, and thus keeping the training time limited.

Activation function

Sigmoid function

$$f'(t) = f(t)(1 - f(t))$$

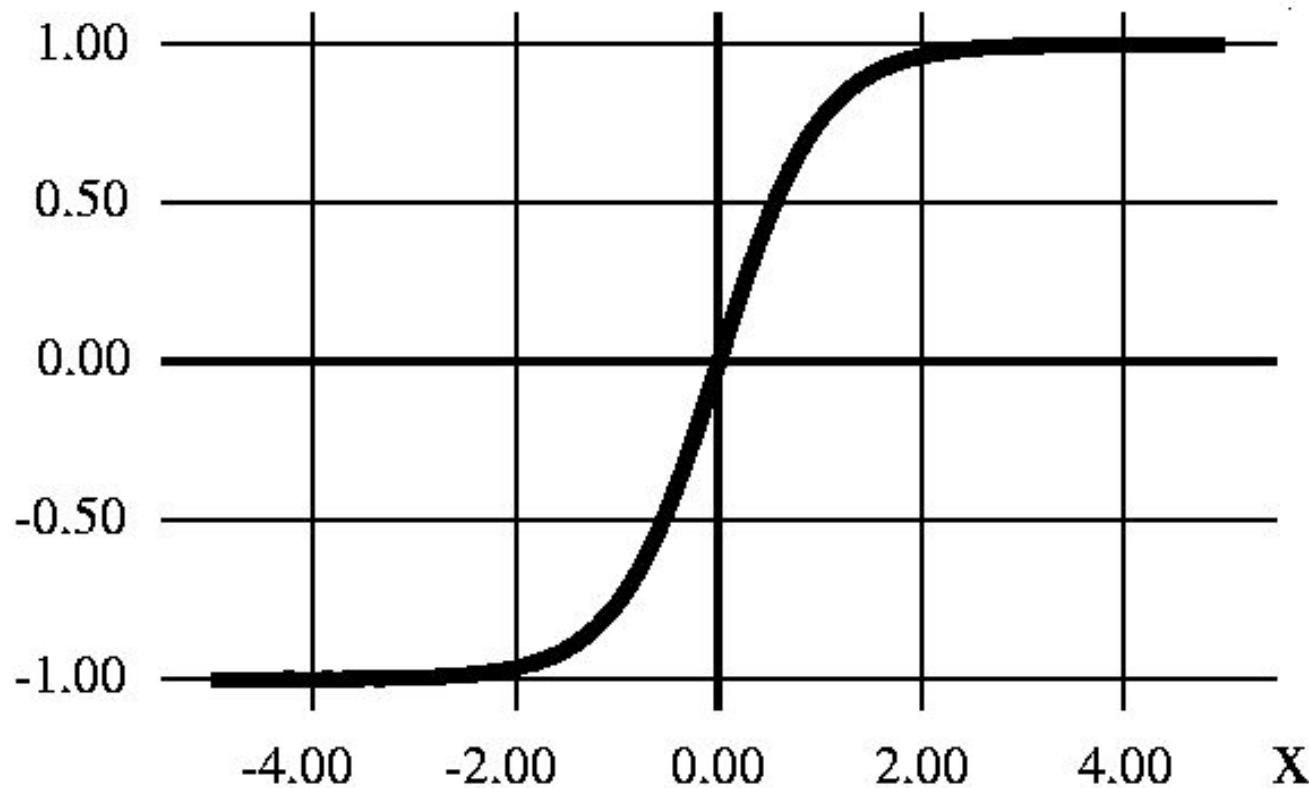
Various parametrized sigmoid functions are there, but, we will not talk about them.



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

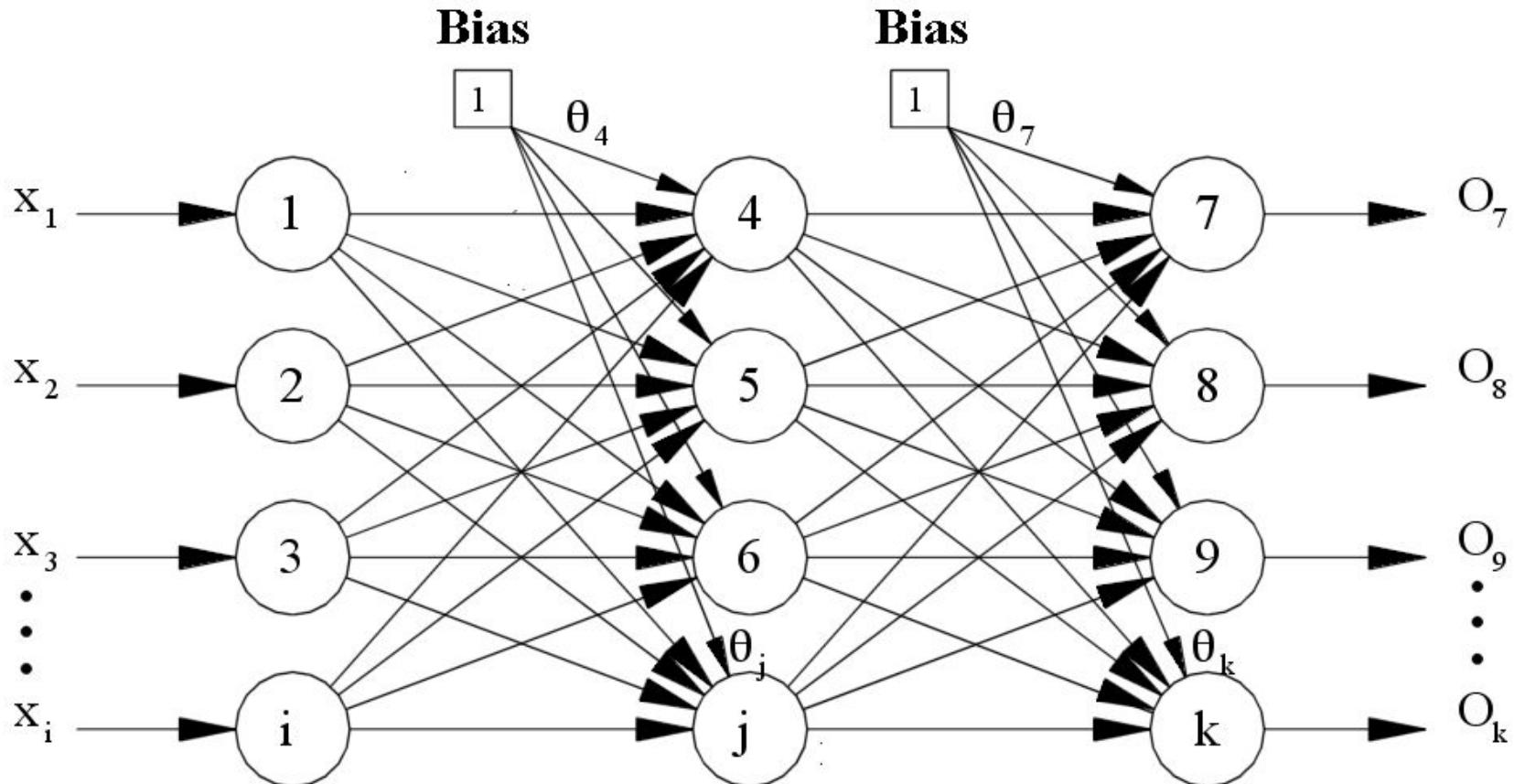
hyperbolic tangent function

- $\tanh(x)$



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

Universal approximation requires bias term



Scaling Input

- A feature with whose values could be of larger magnitude (say it is in thousands), can swamp the features whose values are smaller (say in fractions).
- It is better to normalize the training set to have *zero-mean* and *unit-variance* for each feature.

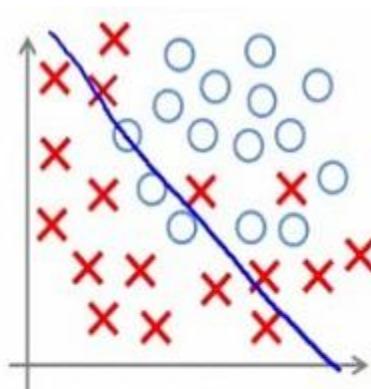
Improvements

- It can be shown that with infinite number of training patterns, multilayer networks converges to the Bayes classifier.
- So, larger the training set, better the classifier.
- Sometimes it is possible for us to create training patterns based on domain knowledge, etc.
 - In OCR data, the images of the characters can be rotated slightly to create new patterns.
 - Probabilistic dependency between the features can be taken into account to create new patterns.
- Drawbacks with larger training sets is that of increased space and training time requirements.

Number of Hidden Units

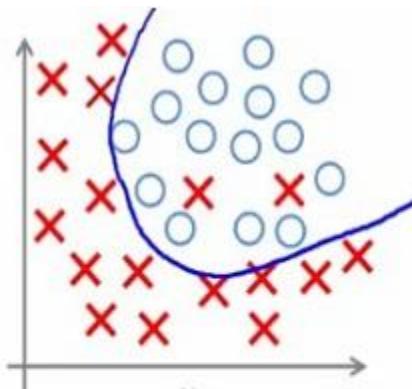
- The number of input and output units are dictated by the problem, but the number of hidden units (n_H) is not.
- Large n_H is required to learn complicated functions. So large n_H means more expressive power for the net.
 - Drawback: Over-fitting. Good performance over the training set does not necessarily mean good performance over the independent test set. One should not respect noisy patterns.
- Small n_H means, the net does not have enough free parameters to fit the training data well, and again the test error is high.

Over-fitting versus under-fitting

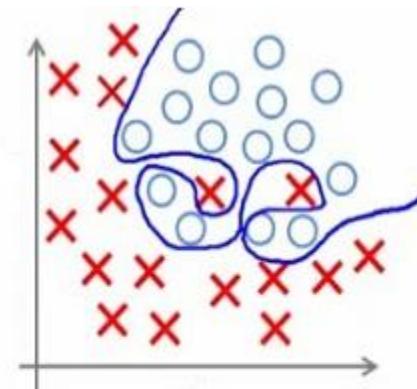


Under-fitting

(too simple to
explain the
variance)



Appropriate-fitting



Over-fitting

(forcefitting -- too
good to be true)

- Regularization is a principled way to overcome the over-fitting problem.

Number of Hidden Units

- n_H determines the total number of weights in the net – which can be considered as number of degrees of freedom – and thus it can be seen that we should not have more weights than the total number of training patterns.
- A convenient rule of thumb is to choose the total number of weights is roughly $N/10$ where N is the number of training patterns.

Learning Rates

- In principle, small learning rates will lead to convergence, but slowly.
- The Hessian matrix (second order information) can guide to optimal learning rate, as we saw for Perceptrons.
- Large learning rates can have negative effect.
- In practice, however thumb-rules are used. For example, $\eta \simeq 0.1$ might be alright, which can be lowered or increased based the changes in the J values.
- Adding Momentum can speed-up the process.
 - A fraction of Previous update can be added ...
- $$\Delta w_{ij}(t) = -\eta \frac{\partial J}{\partial w_{ij}} + \alpha \Delta w_{ij}(t - 1)$$

When to stop the training?

- Excessive training can give us a classifier which is doing very well with the training data.
- This, anyhow, does not guarantee better performance with the test set.
- Cross validation technique can be used to decide when to stop the training.
- Some times early stopping is advocated.

Stopping Criterion

- $$W_{new} = W_{old} + \Delta W$$

Stop when ΔW is sufficiently small.
That is, if ($\|\Delta W\| < \epsilon$), STOP.

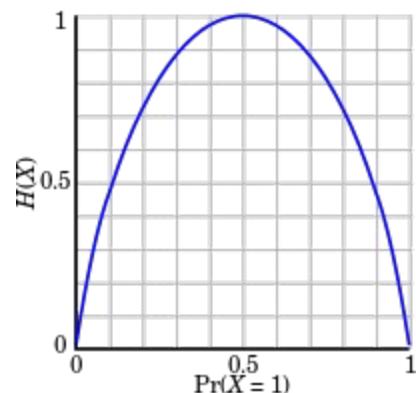
ϵ could be something like 0.01
- Or, after the fixed number of epochs, the training can be stopped.

Other Loss Functions

- **Cross-entropy** is also a widely used loss function.
- **Entropy:** For a distribution, it is a measure of uncertainty.

$$H(X) = - \sum_{X=x_i} p(x_i) \log p(x_i)$$

- For a binary random variable



Cross (relative) entropy

- Distribution q is away from distribution p , by

$$H(p, q) = - \sum_{x_i} p(x_i) \log q(x_i)$$

- Distribution p is the target, q is the network's output.
- For binary classification, a single output neuron is enough.

Let the two classes are labeled 0 and 1.

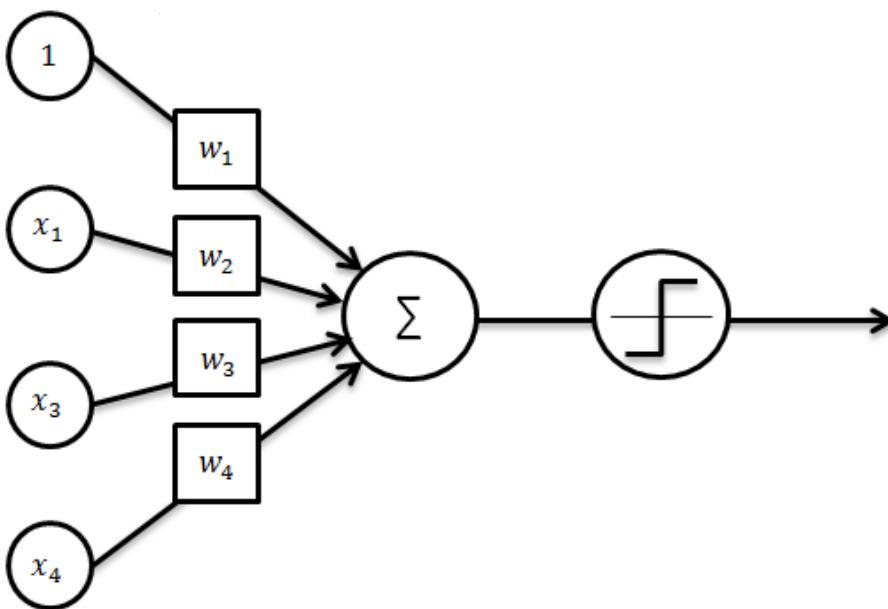
The cross entropy is: $-t \log z - (1 - t) \log(1 - z)$.

$$J(W) = - \sum_{r=1}^c t_r \log z_r + (1 - t_r) \log(1 - z_r)$$

Sigmoid (logistic activation)

- Remember, we used the same activation, i.e., the Sigmoid one.
- Now, one can apply back-propagation with this error being defined.
- For details, it is quite similar to what we did with sum of squared deviations. Reference is given below.
 - But, notation used in the following reference is different.

Perceptron



Perceptron

Perceptron is a single layer neural network.

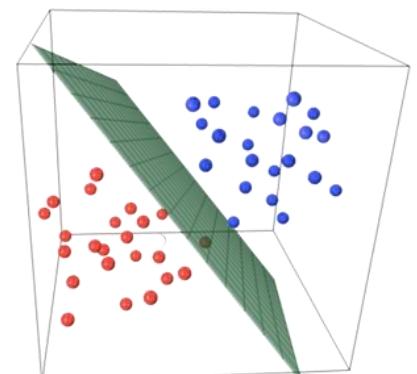
Perceptron is a linear classifier (binary). Also, it is used in supervised learning.

It helps to classify the given input data.

The perceptron consists of 4 parts.

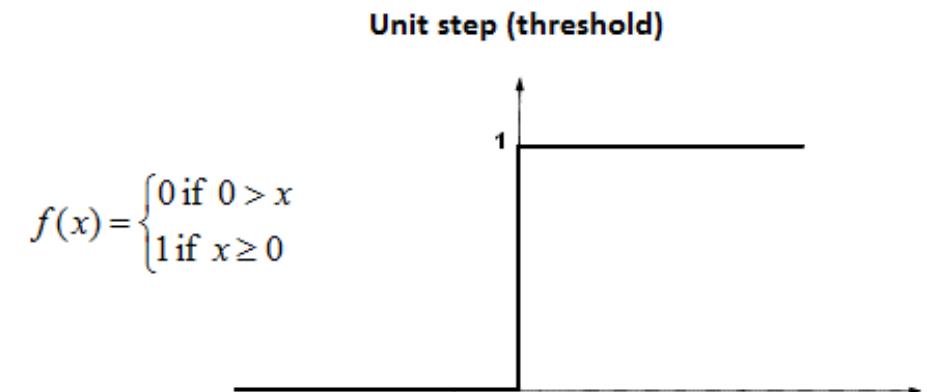
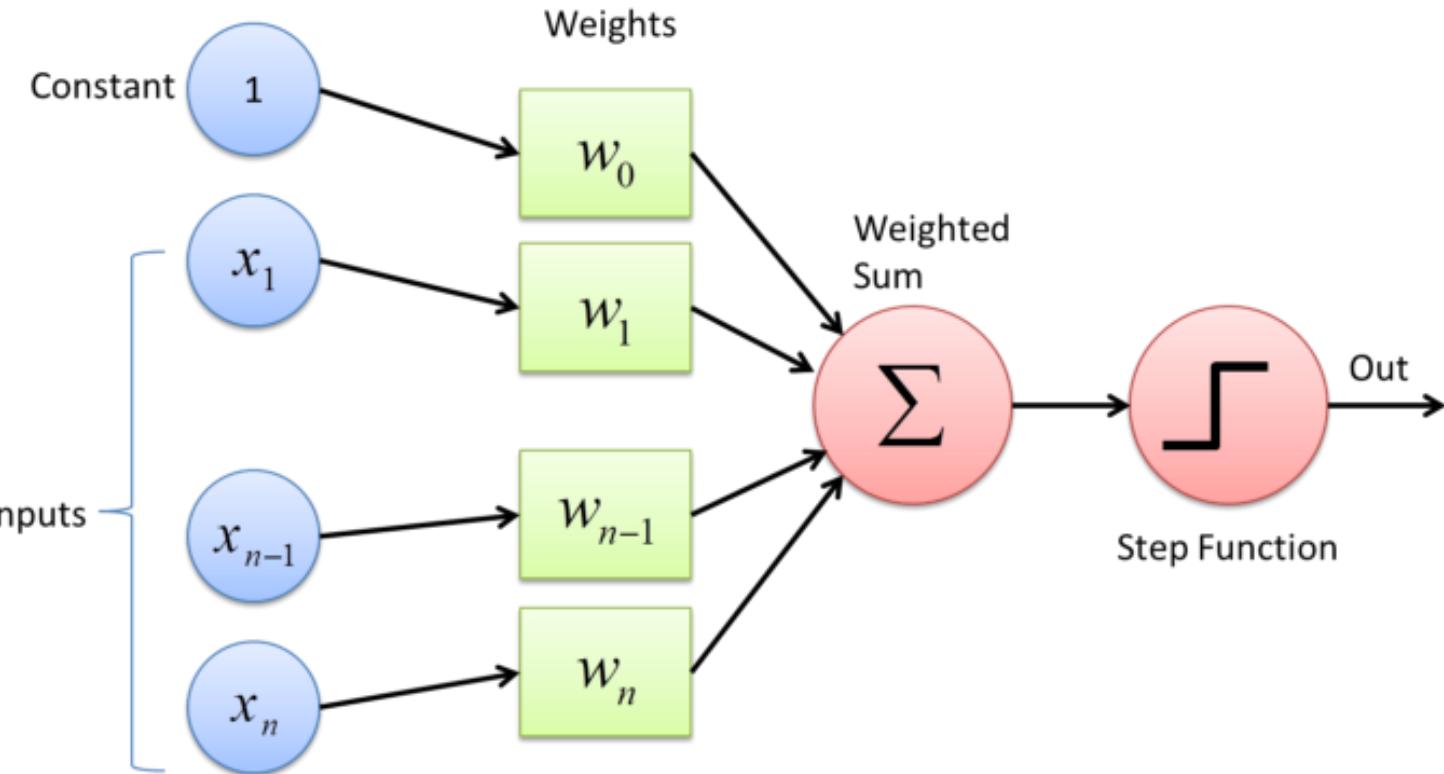
- Input values or One input layer
- Weights and Bias
- Net sum
- Activation Function

The Neural Networks work the same way as the perceptron.

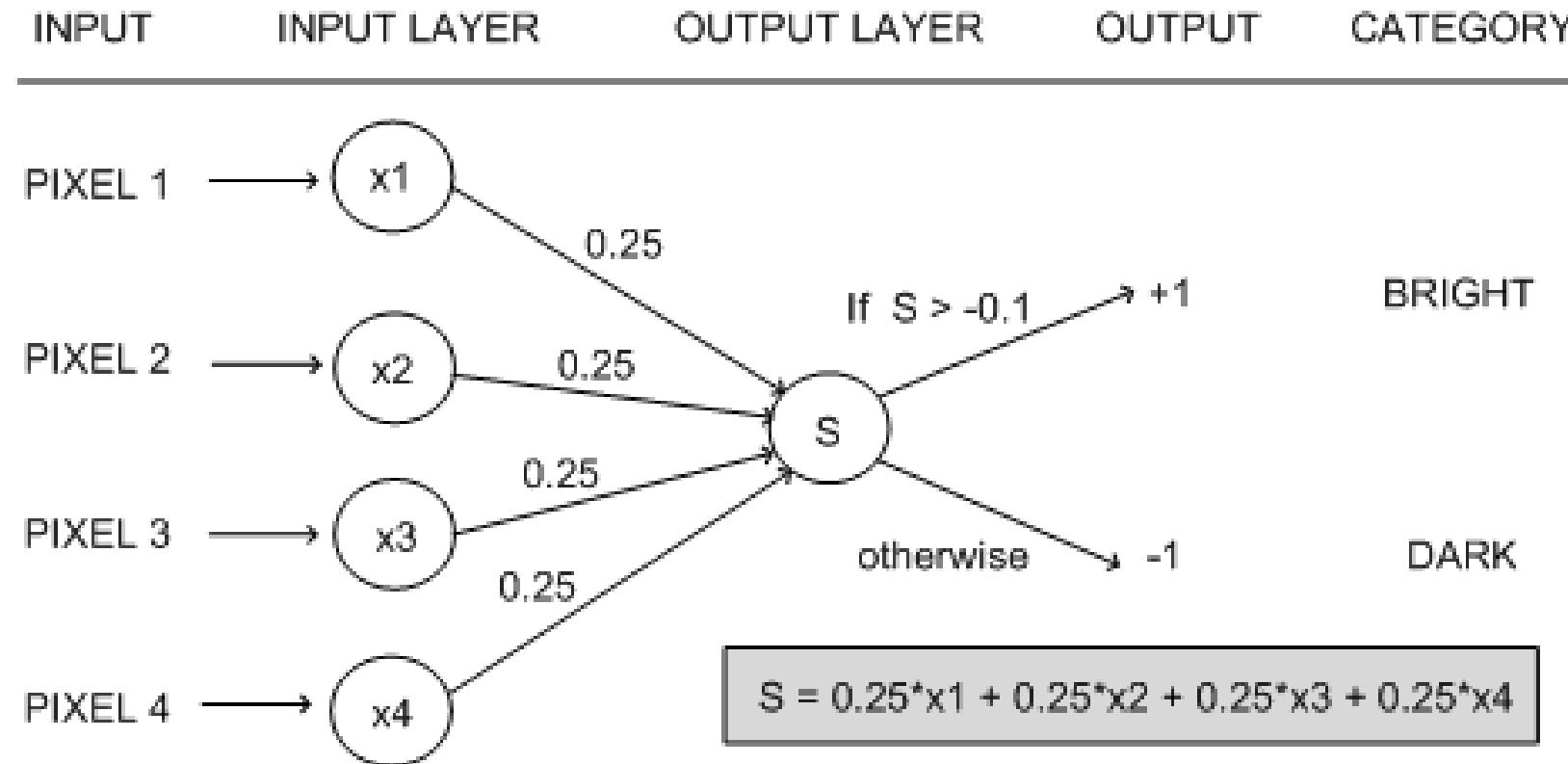


[Image source](#)

Formalization



Formalization



Activation Function

In short, the activation functions are used to map the input between the required values like (0, 1) or (-1, 1).

It's just a thing function that you use to get the output of node. It is also known as ***Transfer Function***.

The Activation Functions can be basically divided into 2 types-

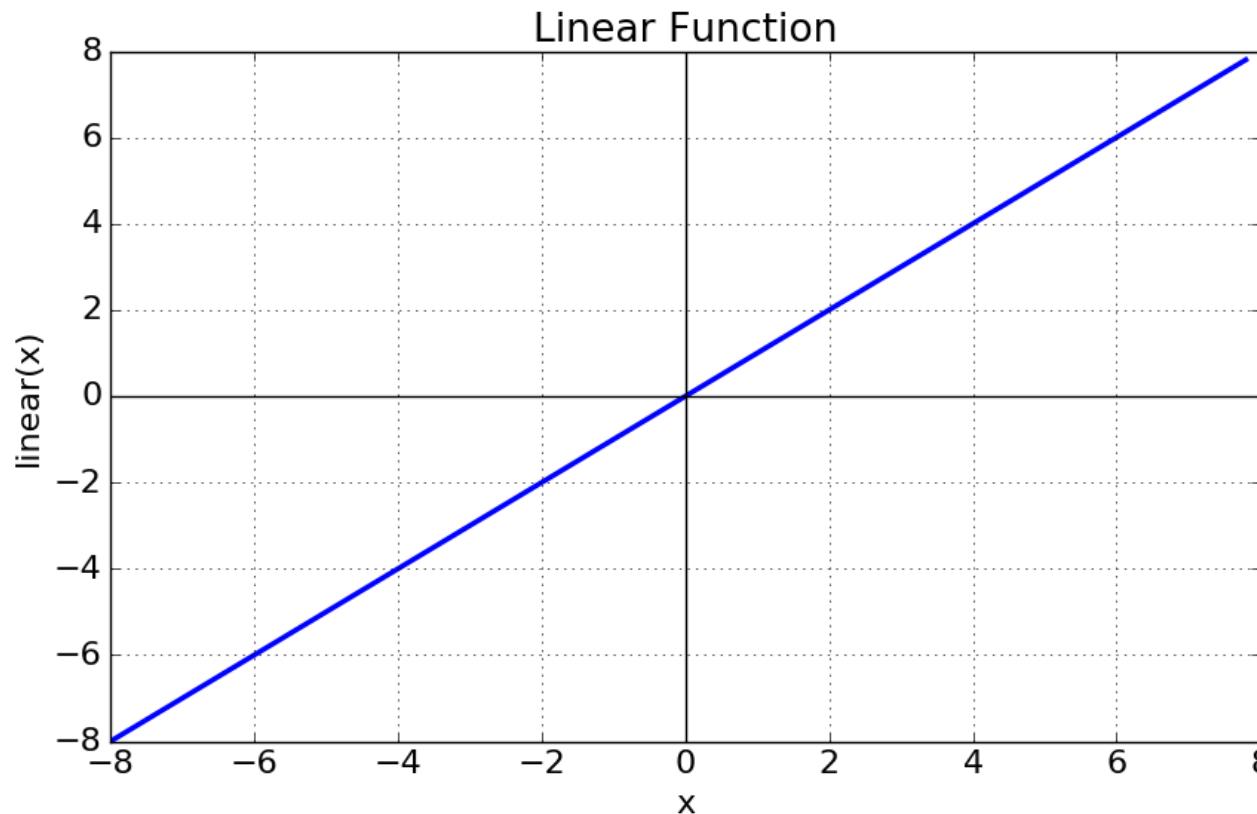
- Linear Activation Function
- Non-linear Activation Functions

Linear or Identity Activation Function

Equation : $f(x) = x$

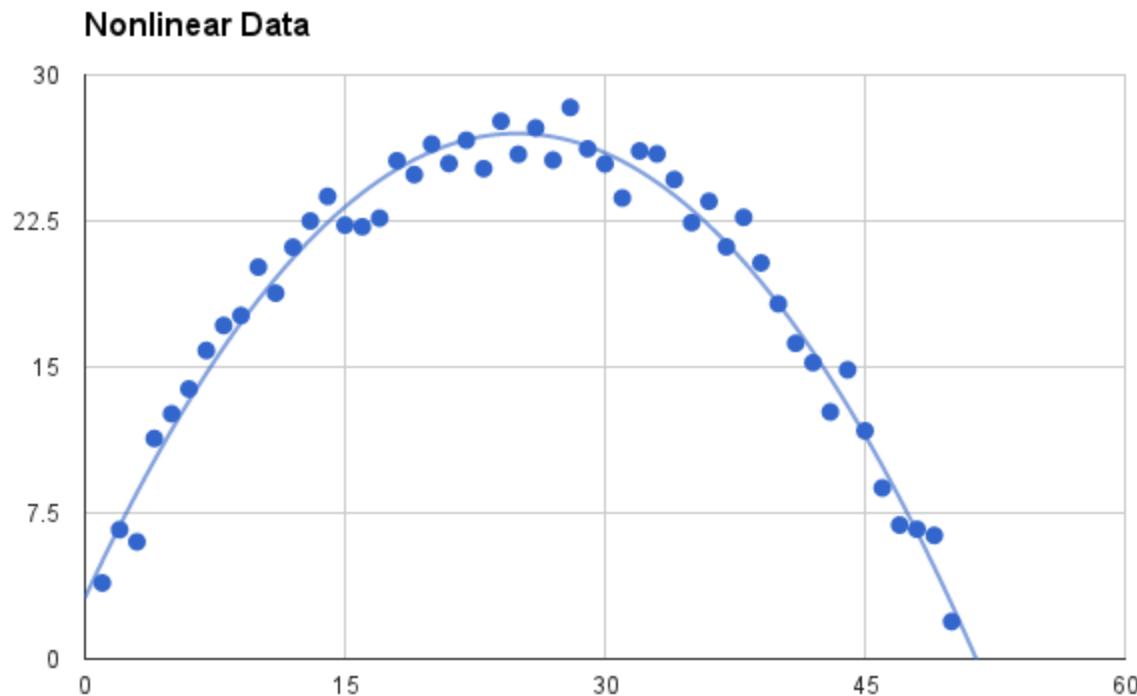
Range : (-infinity to infinity)

It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.



Non-linear Activation Function

The Nonlinear Activation Functions are the most used activation functions.



It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the output.

Non-linear Activation Function

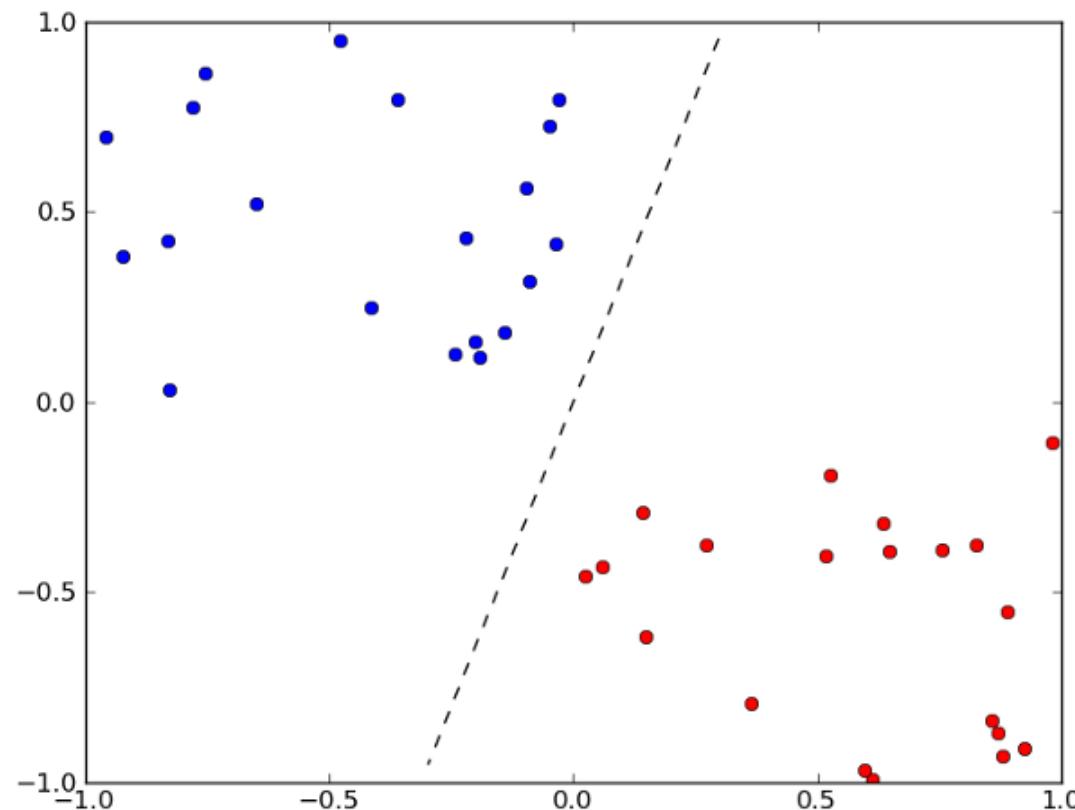
The main terminologies needed to understand for nonlinear functions are:

- Derivative or Differential: Change in y-axis w.r.t. change in x-axis. It is also known as slope.
- Monotonic function: A function which is either entirely non-increasing or non-decreasing.

The Nonlinear Activation Functions are mainly divided on the basis of their **range or curves-**

- Sigmoid or Logistic Activation Function
- Tanh or hyperbolic tangent Activation Function
- ReLU (Rectified Linear Unit) Activation Function

Perceptron is usually used to classify the data into two parts. Therefore, it is also known as a Linear Binary Classifier.

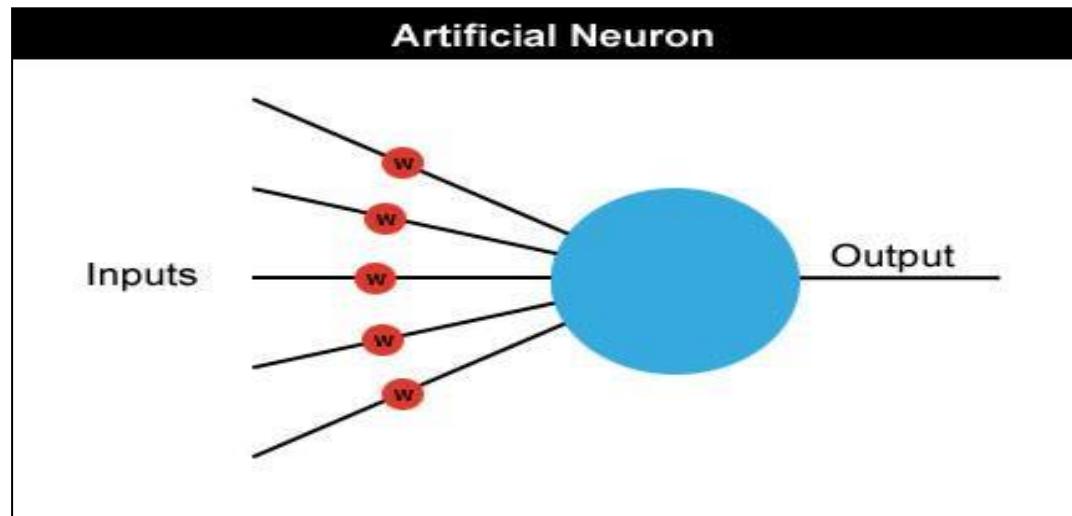


Neural networks

Neural networks are made up of many artificial neurons.

Each input into the neuron has its own weight associated with it illustrated by the red circle.

A weight is simply a floating point number and it's these we adjust when we eventually come to train the network.



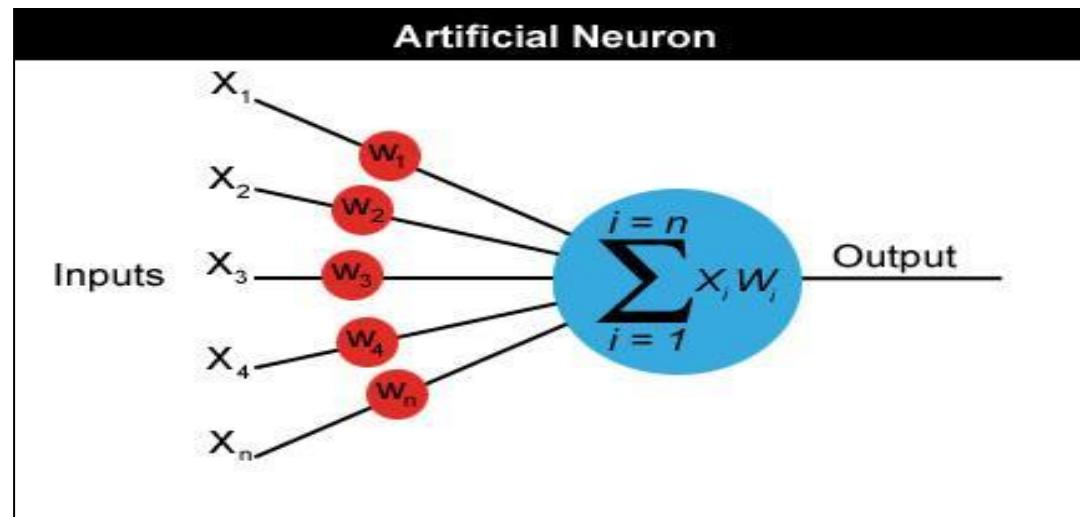
Neural networks

A neuron can have any number of inputs from one to n, where n is the total number of inputs.

The inputs may be represented therefore as $x_1, x_2, x_3 \dots x_n$.

And the corresponding weights for the inputs as $w_1, w_2, w_3 \dots w_n$.

$$\text{Output } a = x_1w_1 + x_2w_2 + x_3w_3 \dots + x_nw_n$$

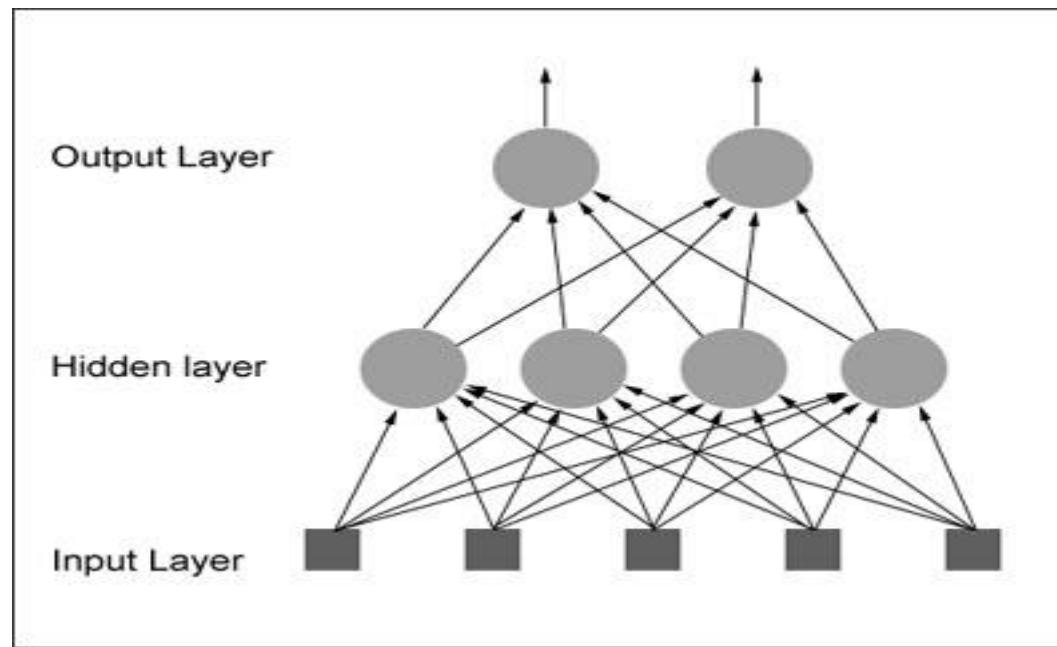


How do we actually *use* an artificial neuron?

feedforward network: The neurons in each layer feed their output forward to the next layer until we get the final output from the neural network.

There can be any number of hidden layers within a feedforward network.

The number of neurons can be completely arbitrary.



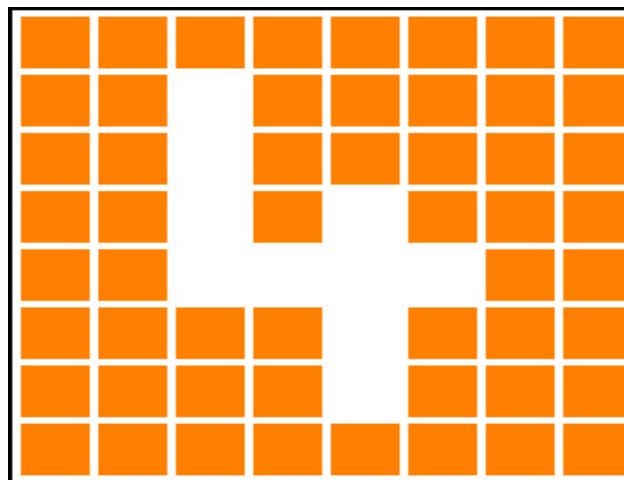
Neural Networks by an Example

let's design a neural network that will detect the number '4'.

Given a panel made up of a grid of lights which can be either on or off, we want our neural net to let us know whenever it thinks it sees the character '4'.

The panel is eight cells square and looks like this:

the neural net will have **64 inputs**, each one representing a particular cell in the panel and a hidden layer consisting of a number of neurons (more on this later) all feeding their output into just **one neuron in the output layer**

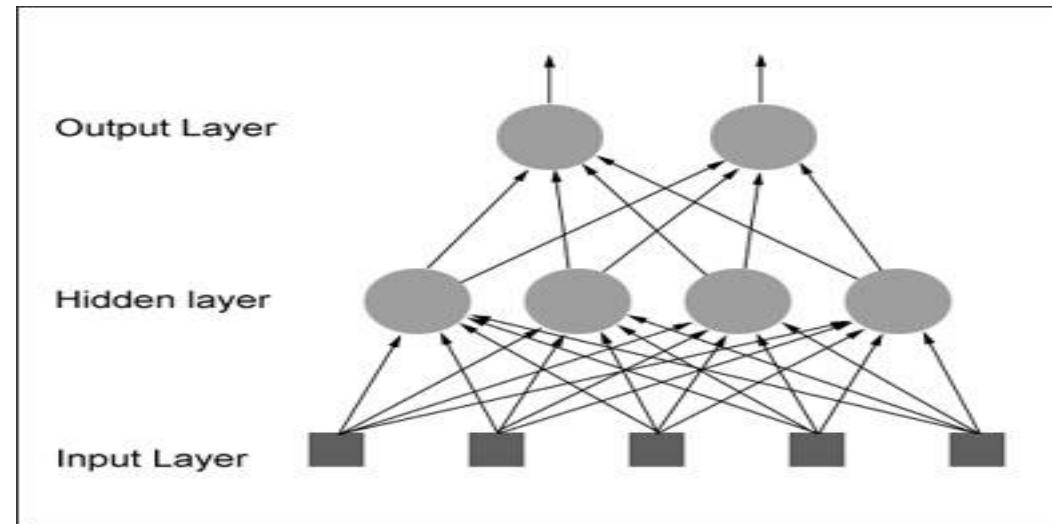


Neural Networks by an Example

initialize the neural net with random weights

feed it a series of inputs which represent, in this example, the different panel configurations

For each configuration we check to see what its output is and **adjust the weights accordingly** so that whenever it sees something looking like a number 4 it outputs a 1 and for everything else it outputs a zero.

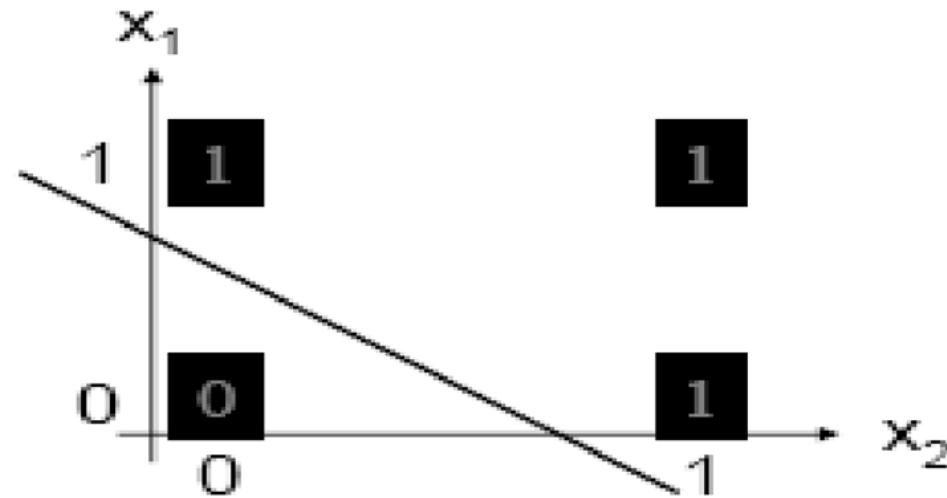


Perceptron Learning Theorem

Recap: A perceptron (threshold unit) can *learn* anything that it can *represent* (i.e. anything separable with a hyperplane)

OR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

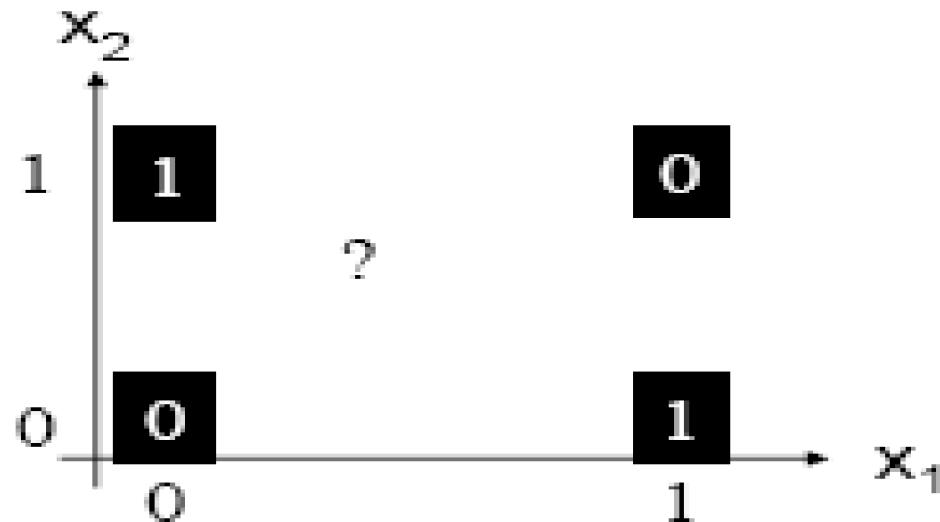


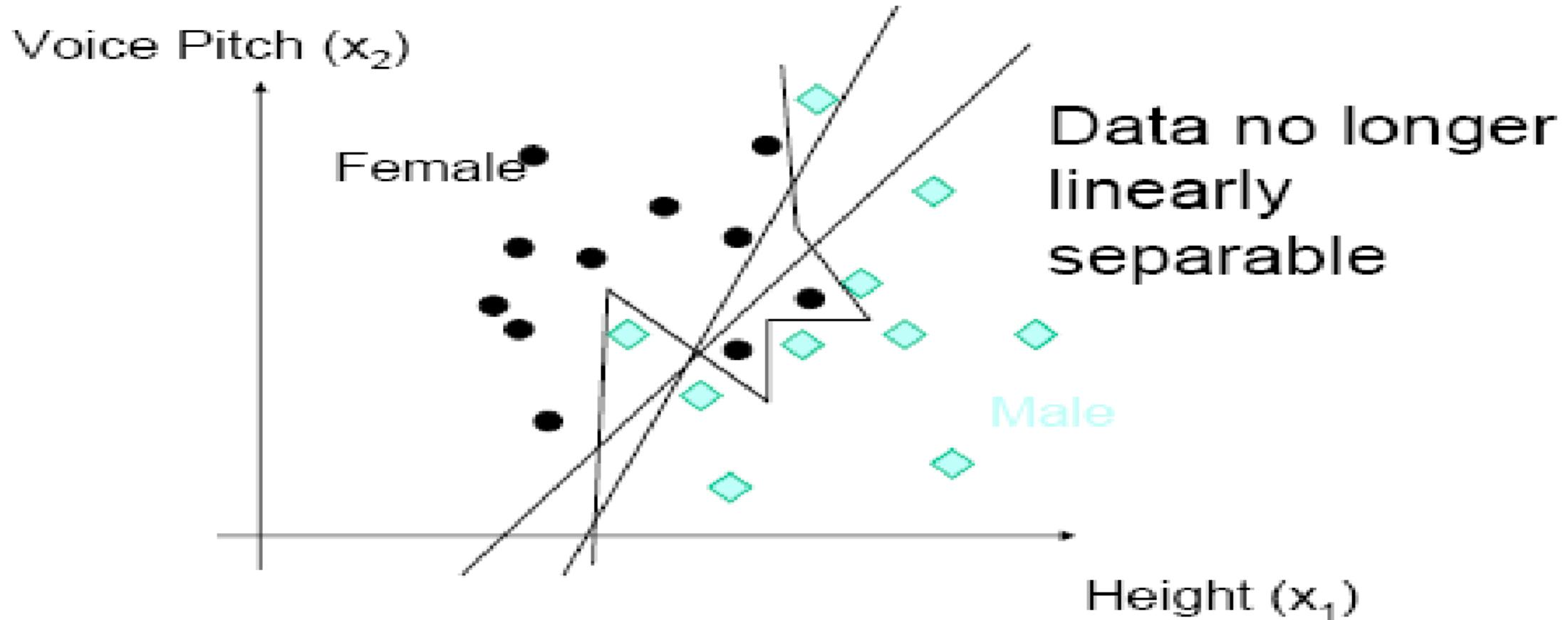
The Exclusive OR problem

A Perceptron cannot represent Exclusive OR since it is not linearly separable.

XOR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

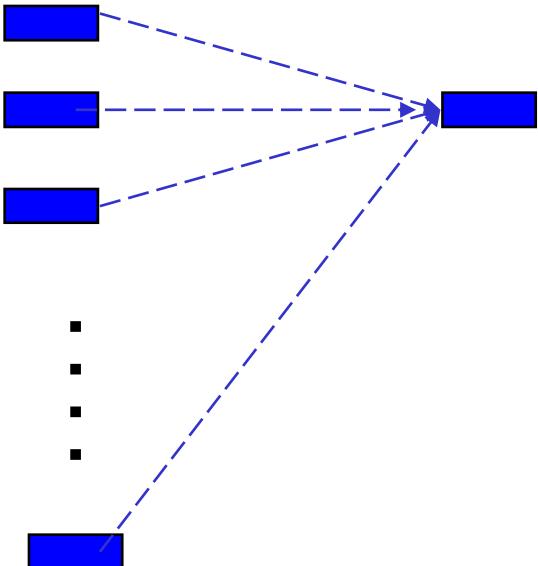




What is a good decision boundary ?

Properties of NN architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units



Each unit is a
perceptron



Often include bias as an extra weight

Perceptron training algorithm

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example (x_i, y_i) :
- If current prediction $\text{sgn}(w^T x_i)$ does not match y_i then update weights:

$$w \leftarrow w + \eta y_i x_i$$

where η is a *learning rate* that should decay slowly* over time

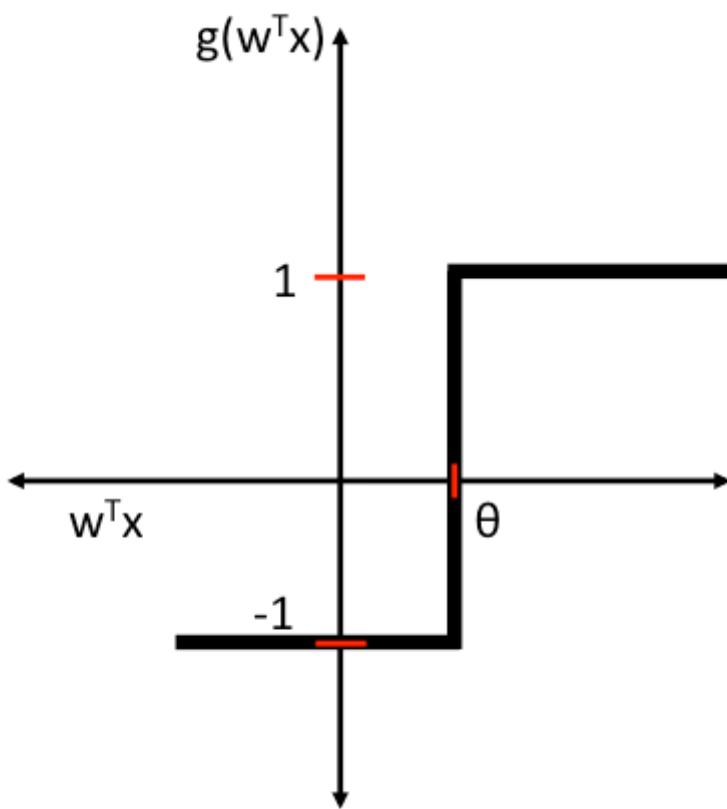
$$g(\mathbf{z}) = \begin{cases} 1 & \text{if } \mathbf{z} \geq \theta \\ -1 & \text{otherwise.} \end{cases}$$

where

$$\begin{aligned}\mathbf{z} &= w_1x_1 + \cdots + w_mx_m = \sum_{j=1}^m x_jw_j \\ &= \mathbf{w}^T \mathbf{x}\end{aligned}$$

\mathbf{w} is the feature (weight) vector, and \mathbf{x} is an m -dimensional sample from the training dataset:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$



Unit step function.

Perceptron Learning Rule

Rosenblatt's initial perceptron rule is fairly simple and can be summarized by the following steps:

Initialize the weights to 0 or small random numbers.

For each training sample $x(i)$:

- Calculate the *output* value.
 - Update the weights.
-
- The output value is the class label predicted by the unit step function that we defined earlier ($\text{output} = g(z)$)
 - the weight update can be written more formally as $w_j := w_j + \Delta w_j$

$$\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$$

where η is the learning rate (a constant between 0.0 and 1.0), “target” is the true class label, and the “output” is the predicted class label.

It is important to note that all weights in the weight vector are being updated on a single dataset, we will

$$\Delta w_0 = \eta (\text{target}^{(i)} - \text{output}^{(i)})$$

$$\Delta w_1 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_1^{(i)}$$

$$\Delta w_2 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_2^{(i)}$$

In the two scenarios where the perceptron predicts the class label correctly, the weights remain unchanged:

$$\Delta w_j = \eta(-1^{(i)} - -1^{(i)}) x_j^{(i)} = 0$$

$$\Delta w_j = \eta(1^{(i)} - 1^{(i)}) x_j^{(i)} = 0$$

However, in case of a wrong prediction, the weights are being “pushed” towards the direction of the positive or negative target class, respectively:

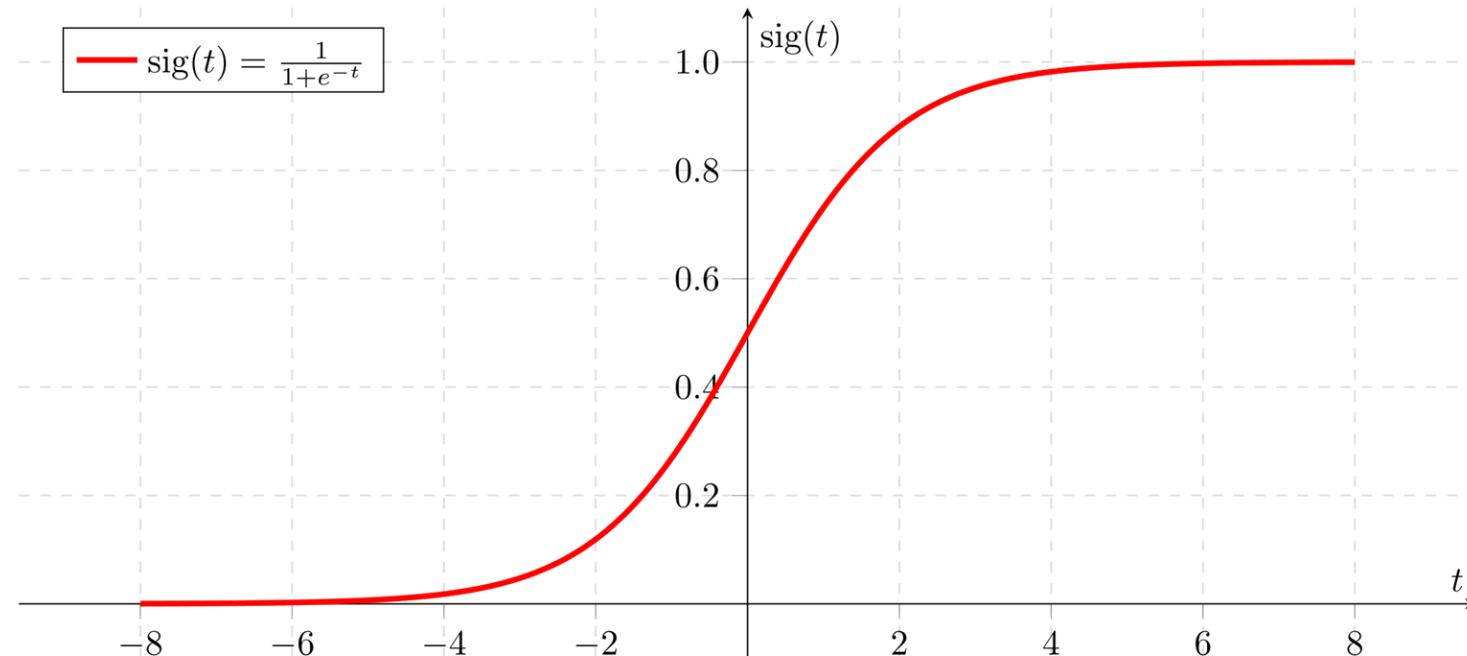
$$\Delta w_j = \eta(1^{(i)} - -1^{(i)}) x_j^{(i)} = \eta(2) x_j^{(i)}$$

$$\Delta w_j = \eta(-1^{(i)} - 1^{(i)}) x_j^{(i)} = \eta(-2) x_j^{(i)}$$

Sigmoid function

- Squash the linear response of the classifier to the interval [0,1]:

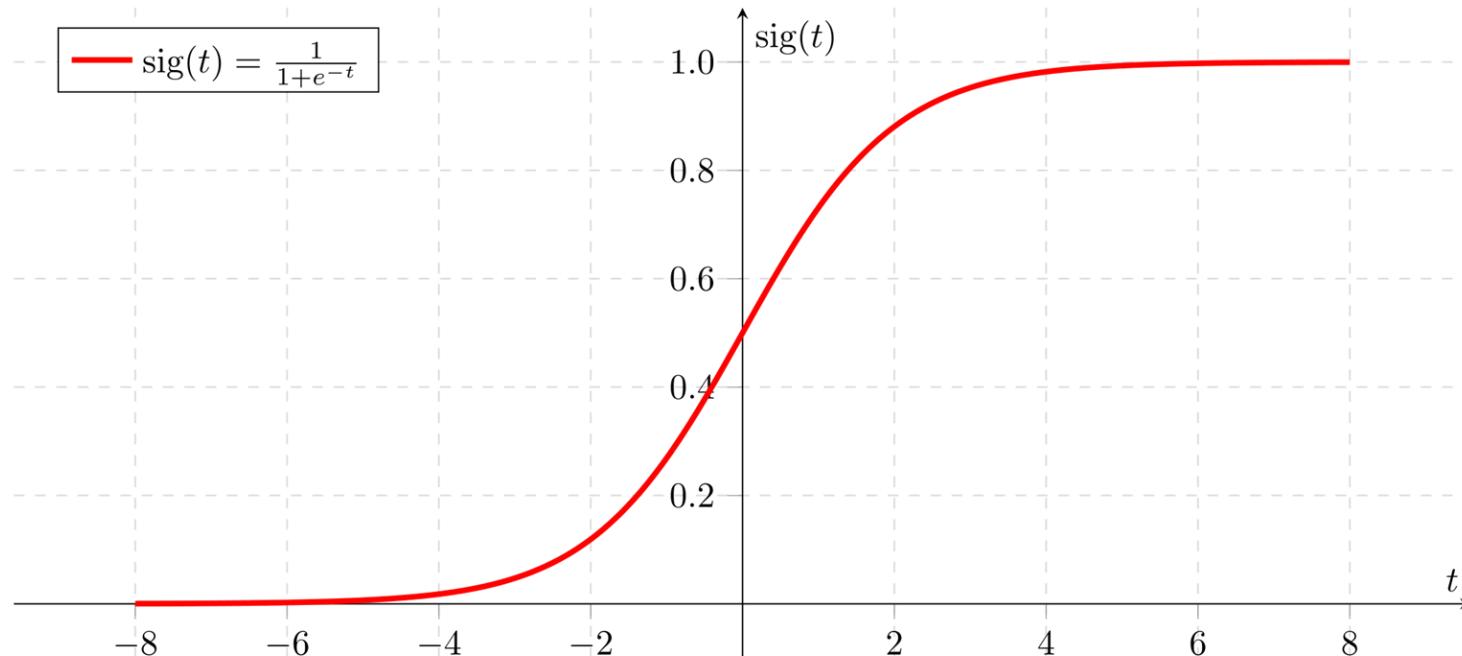
$$\sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$



Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

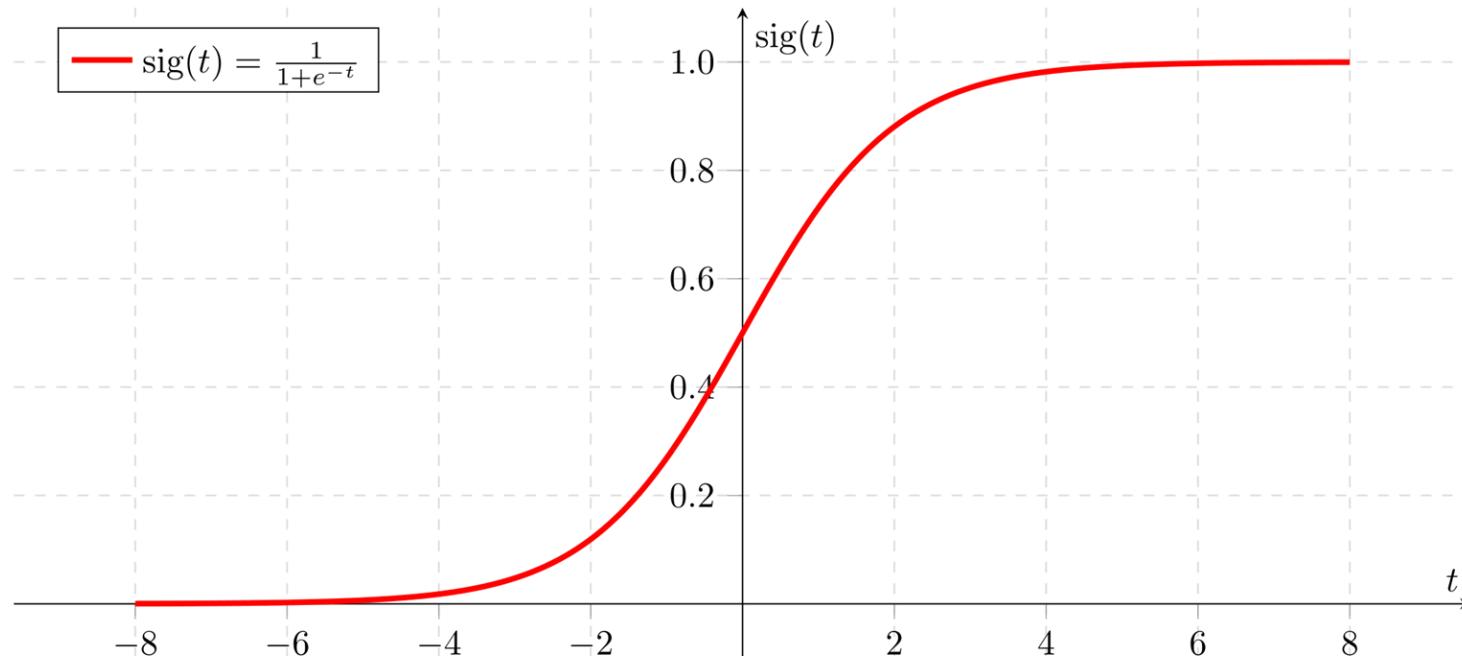


Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- Sigmoid is *symmetric*: $1 - \sigma(t) = \sigma(-t)$

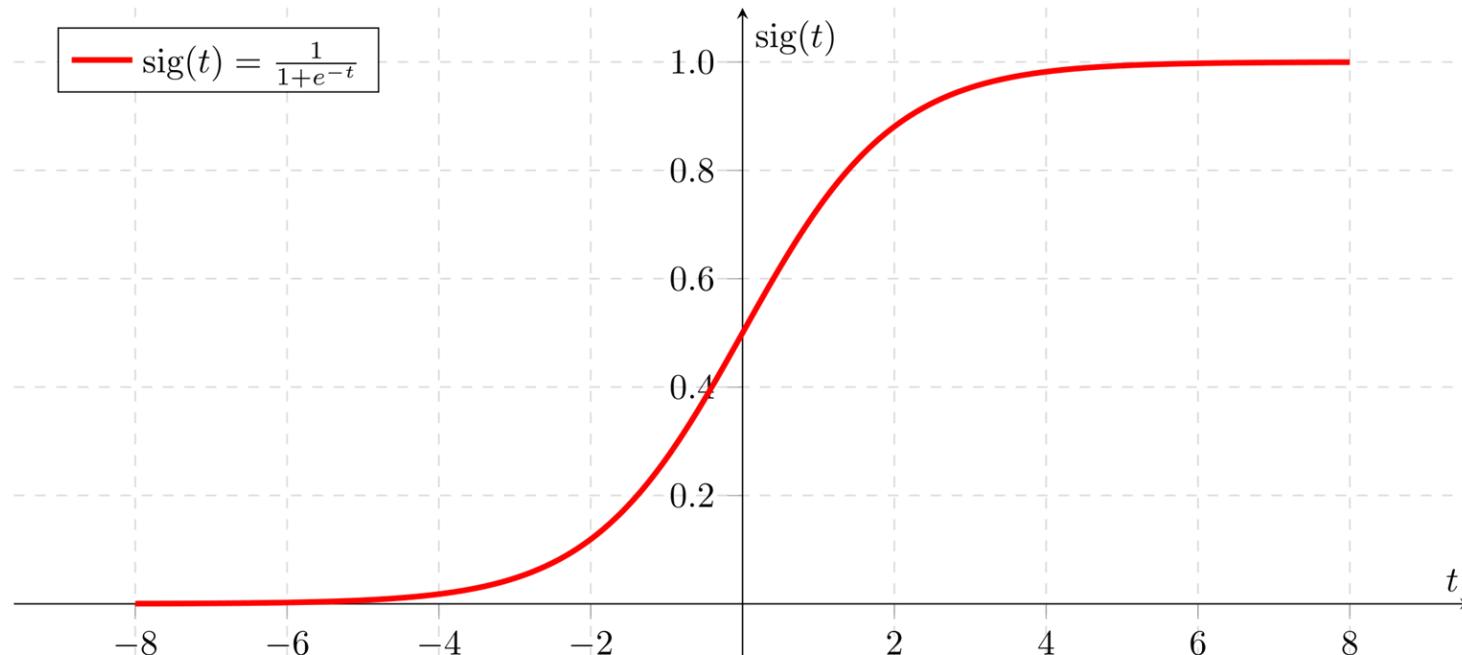


Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

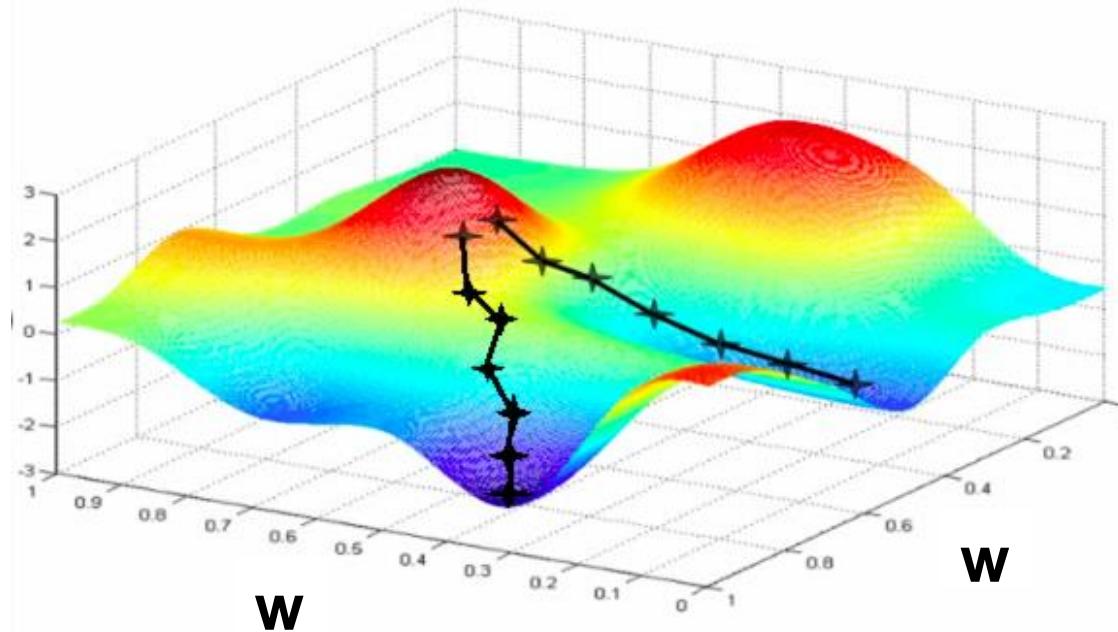
- What happens if we scale w by a constant?



Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- At each step, find $\nabla \hat{L}(w)$, the *gradient* of the loss w.r.t. w , and take a small step in the *opposite* direction

$$w \leftarrow w - \eta \nabla \hat{L}(w)$$



Stochastic gradient descent (SGD)

- At each iteration, take a single data point (x_i, y_i) and perform a parameter update using $\nabla l(w, x_i, y_i)$, the gradient of the loss for that point:

$$w \leftarrow w - \eta \nabla l(w, x_i, y_i)$$

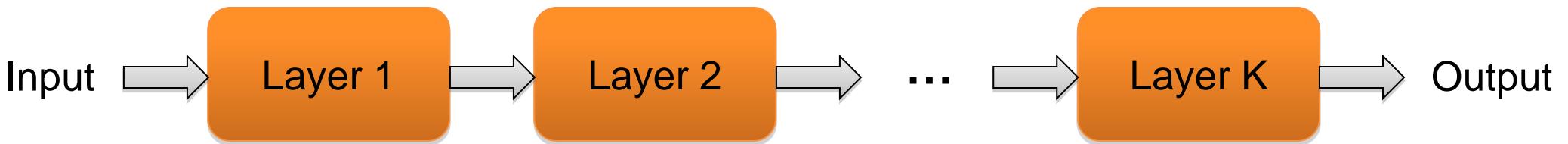
- This is called an *online* or *stochastic* update

How to train a multi-layer network?

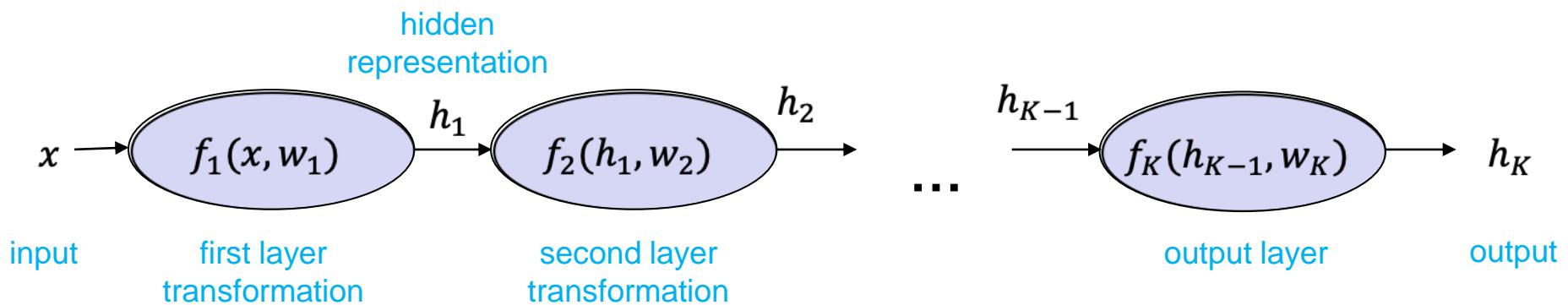


Multi-layer neural networks

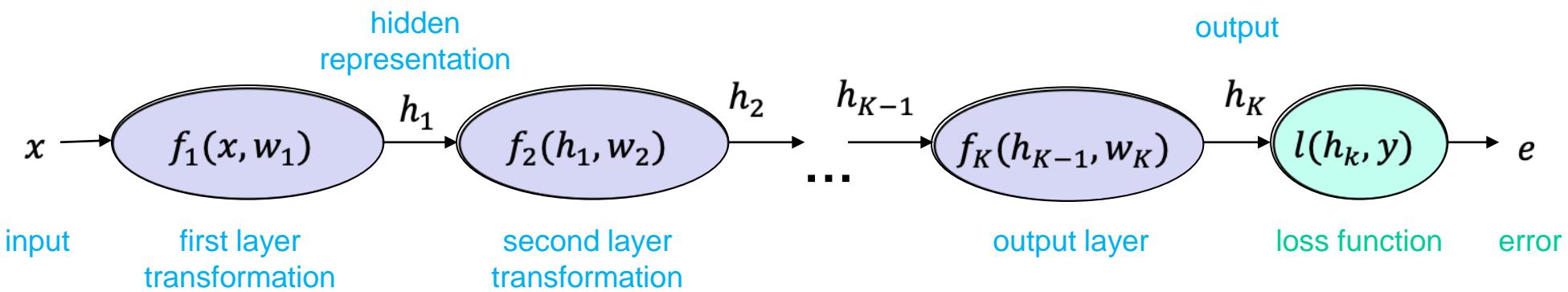
- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):



- More precisely:



Training a multi-layer network

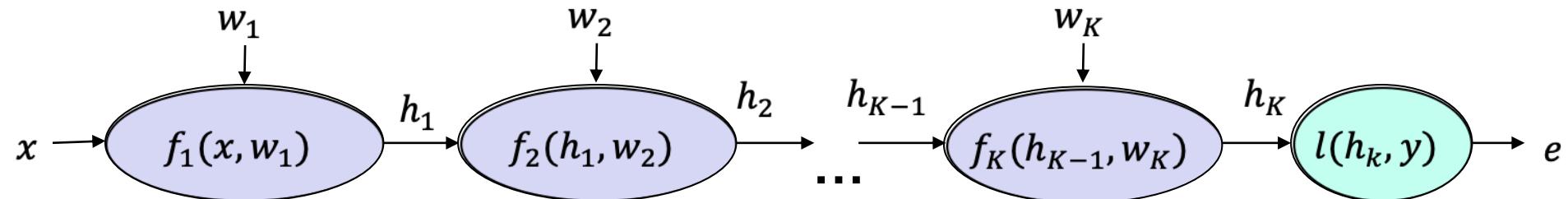


- What is the SGD update for the parameters w_k of the k th layer?

$$w_k \leftarrow w_k - \eta \frac{\partial e}{\partial w_k}$$

- To train the network, we need to find the **gradient of the error** w.r.t. the parameters of each layer, $\frac{\partial e}{\partial w_k}$

Computation graph

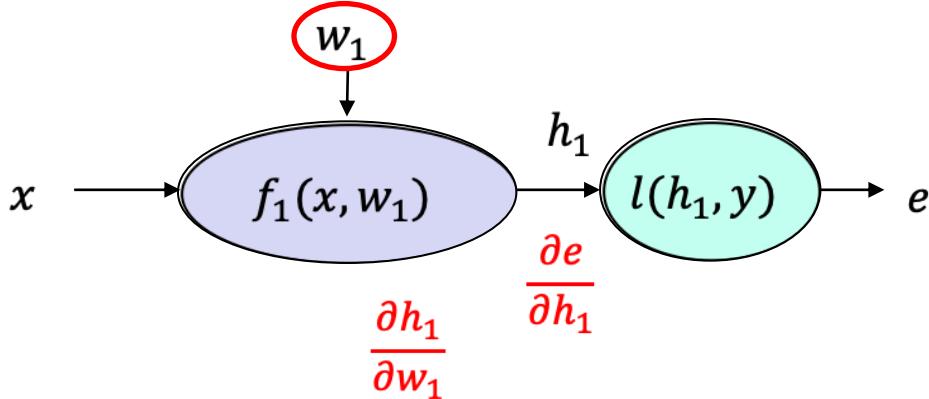


Chain rule

Let's start with $k = 1$

$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) =$$



Example: $e = (y - w_1^T x)^2$

$$h_1 = f_1(x, w_1) = w_1^T x$$

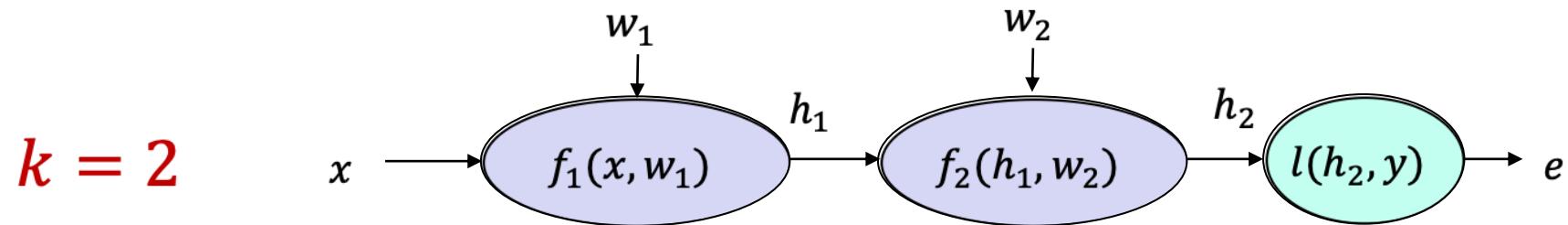
$$e = l(h_1, y) = (y - h_1)^2$$

$$\frac{\partial h_1}{\partial w_1} =$$

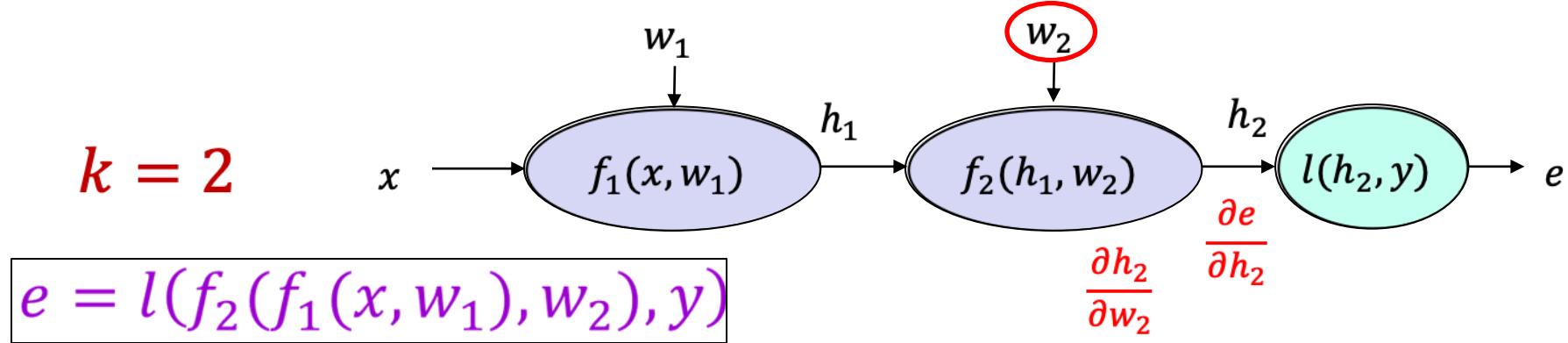
$$\frac{\partial e}{\partial h_1} =$$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1} = \dots$$

Chain rule

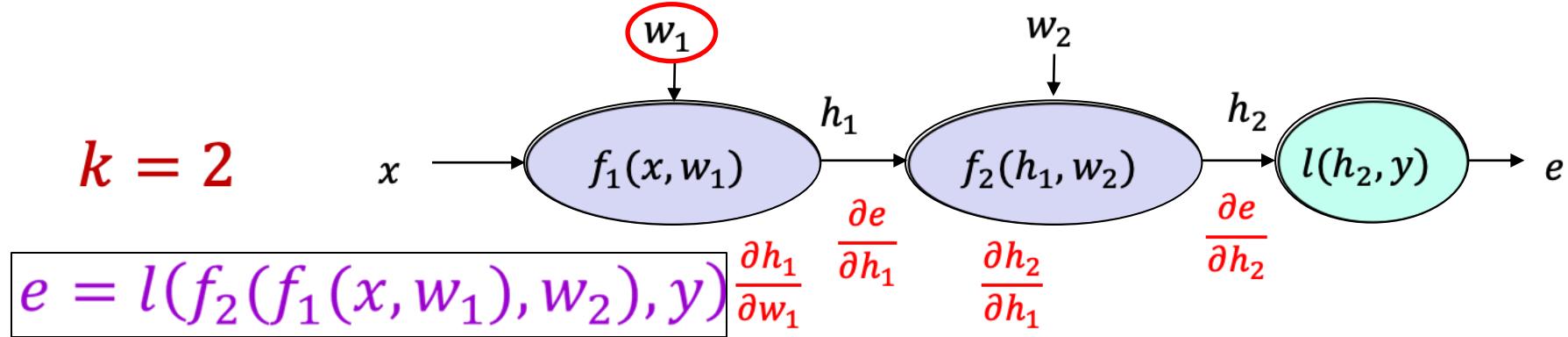


Chain rule



$$\frac{\partial e}{\partial w_2} =$$

Chain rule



$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2} \frac{\partial h_2}{\partial w_2}$$



Example: $e = -\log(\sigma(w_1^T x))$ (assume $y = 1$)

$$h_1 = f_1(x, w_1) = w_1^T x$$

$$\frac{\partial h_1}{\partial w_1} =$$

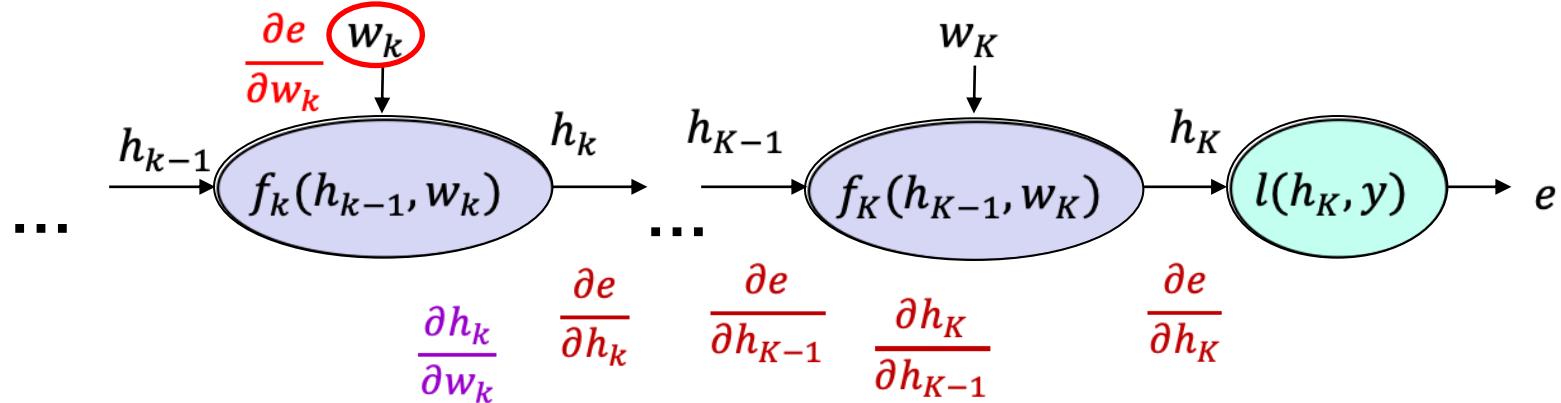
$$h_2 = f_2(h_1) = \sigma(h_1)$$

$$\frac{\partial h_2}{\partial h_1} =$$

$$e = l(h_2, 1) = -\log(h_2)$$

$$\frac{\partial e}{\partial h_2} =$$

Chain rule



General case:

$$\frac{\partial e}{\partial w_k} = \boxed{\frac{\partial e}{\partial h_K} \quad \frac{\partial h_K}{\partial h_{K-1}} \quad \dots \quad \frac{\partial h_{k+1}}{\partial h_k}} \frac{\partial h_k}{\partial w_k}$$

Upstream gradient

$$\frac{\partial e}{\partial h_k}$$

Local gradient

Backpropagation summary

Parameter update:

$$\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial w_k}$$

w_k

$$\frac{\partial h_k}{\partial w_k} \text{ Local gradient}$$

f_k

$$\frac{\partial h_k}{\partial h_{k-1}} \text{ Local gradient}$$

h_{k-1}

Downstream gradient:

$$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k} \frac{\partial h_k}{\partial h_{k-1}}$$

Upstream
gradient:

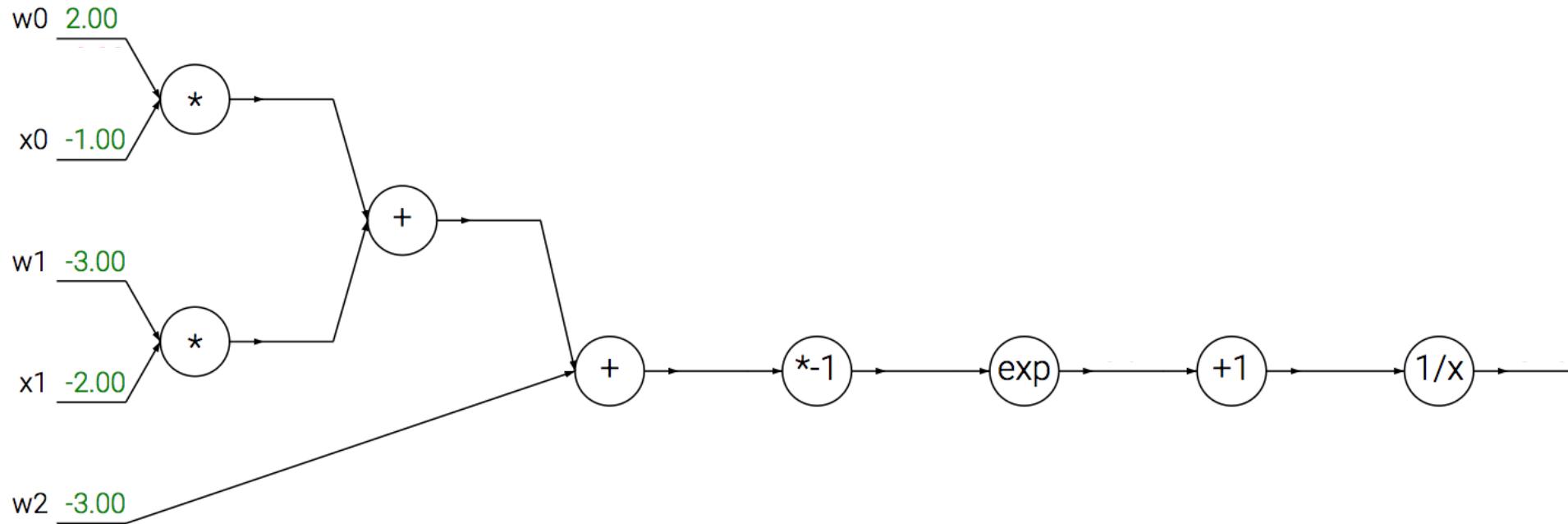
$$\frac{\partial e}{\partial h_k}$$

h_k

→ Forward pass
← Backward pass

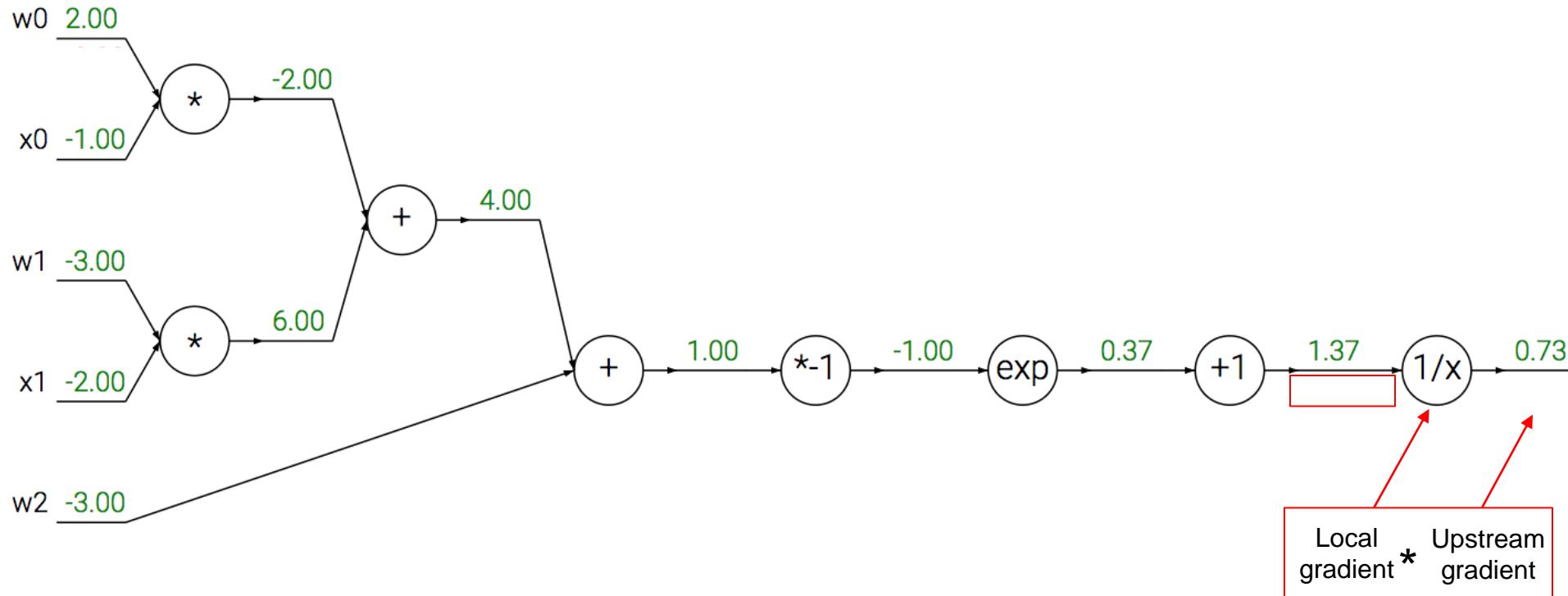
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0 x_0 + w_1 x_1 + w_2)]}$$



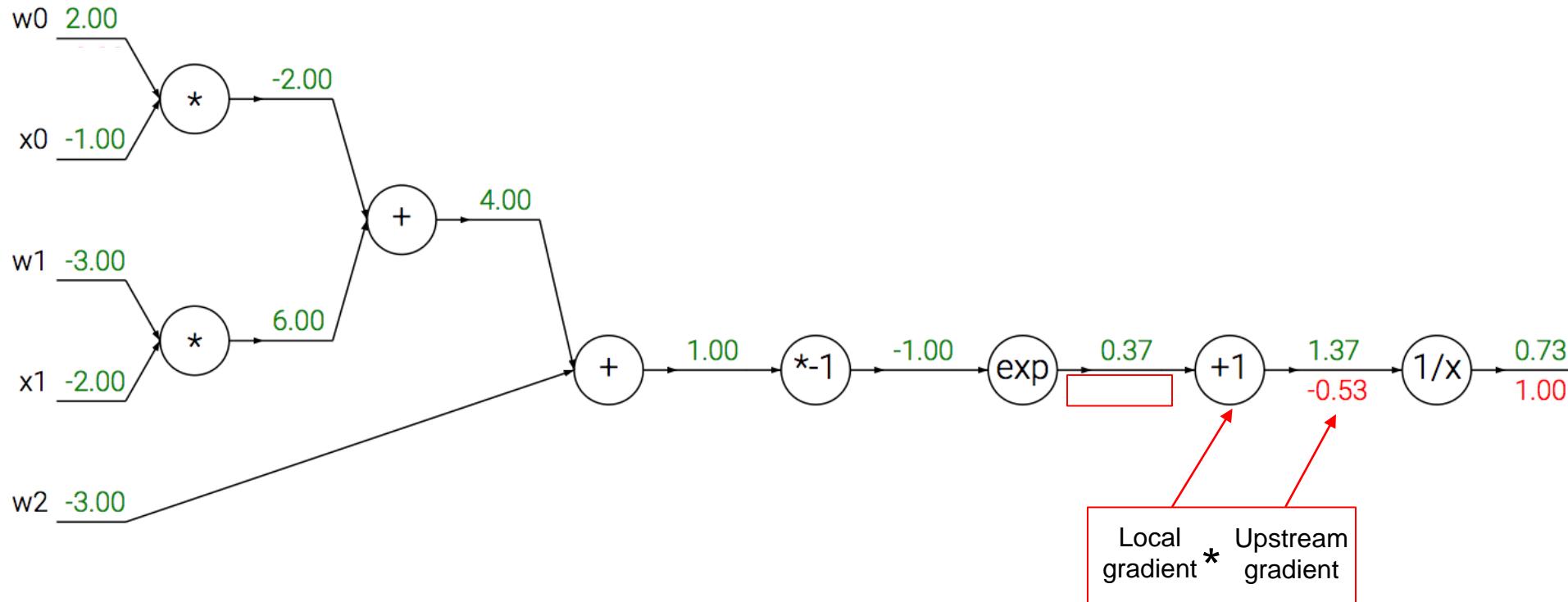
$$(1/x)' = -1/x^2$$

$$-\frac{1}{1.37^2} * 1 = -0.53$$

Source: [Stanford 231n](#)

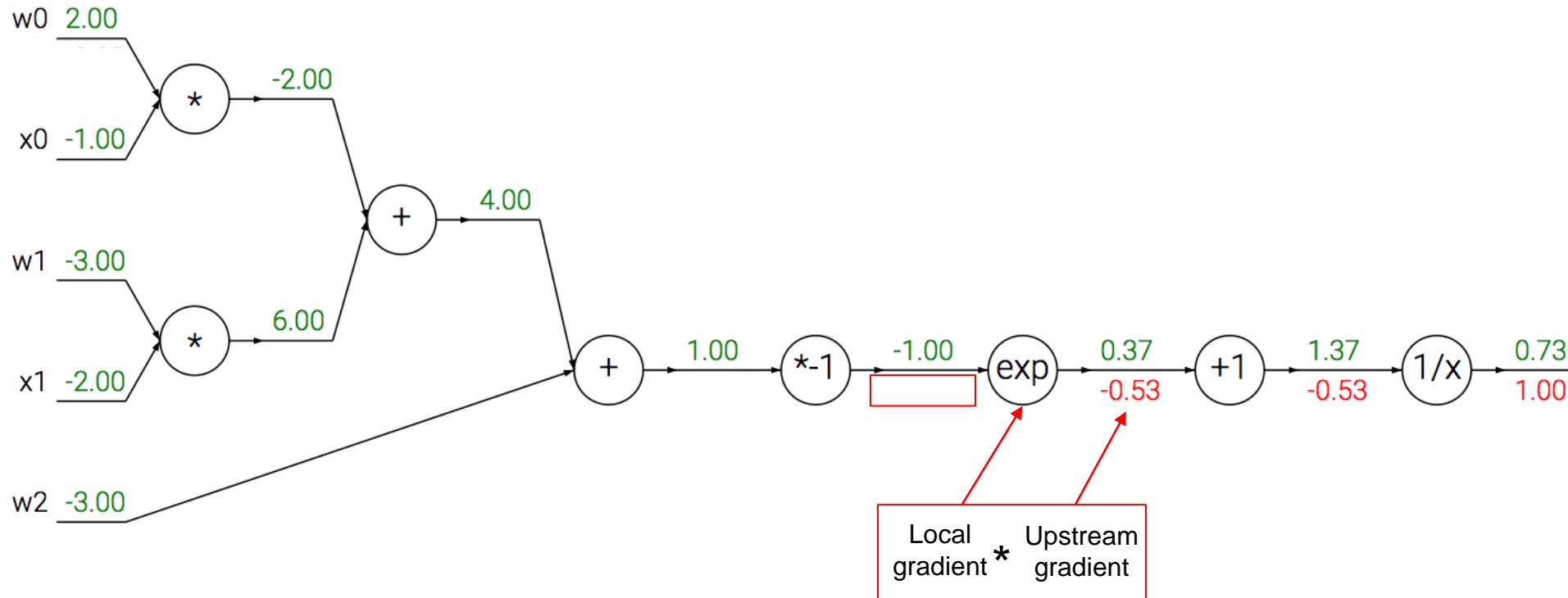
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



A detailed example

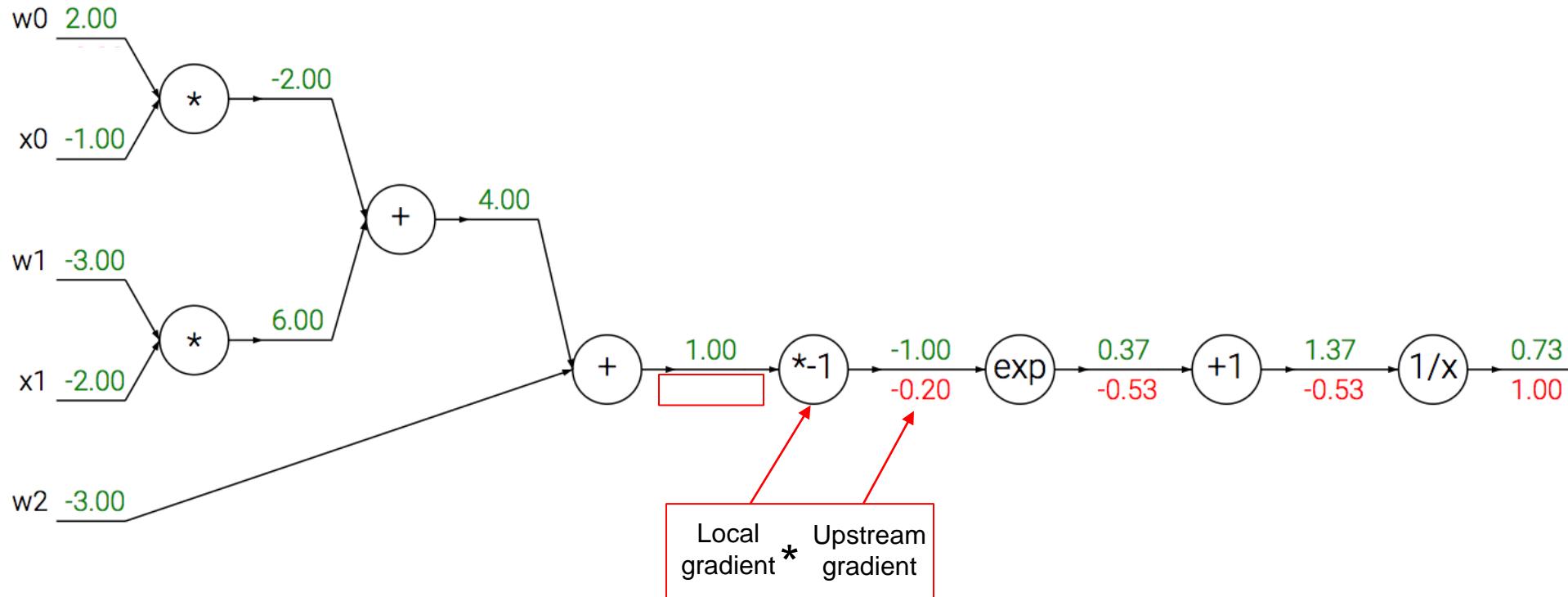
$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



$$\exp(-1) * (-0.53) = -0.20$$

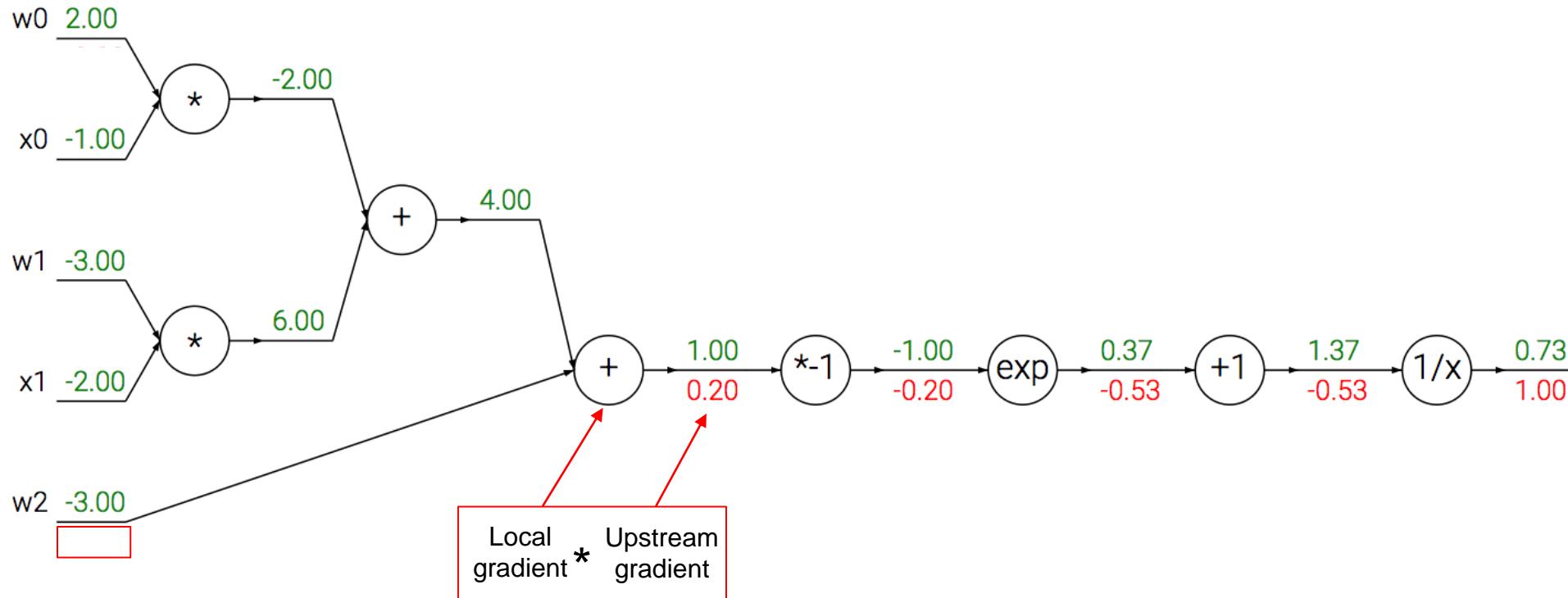
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0 x_0 + w_1 x_1 + w_2)]}$$



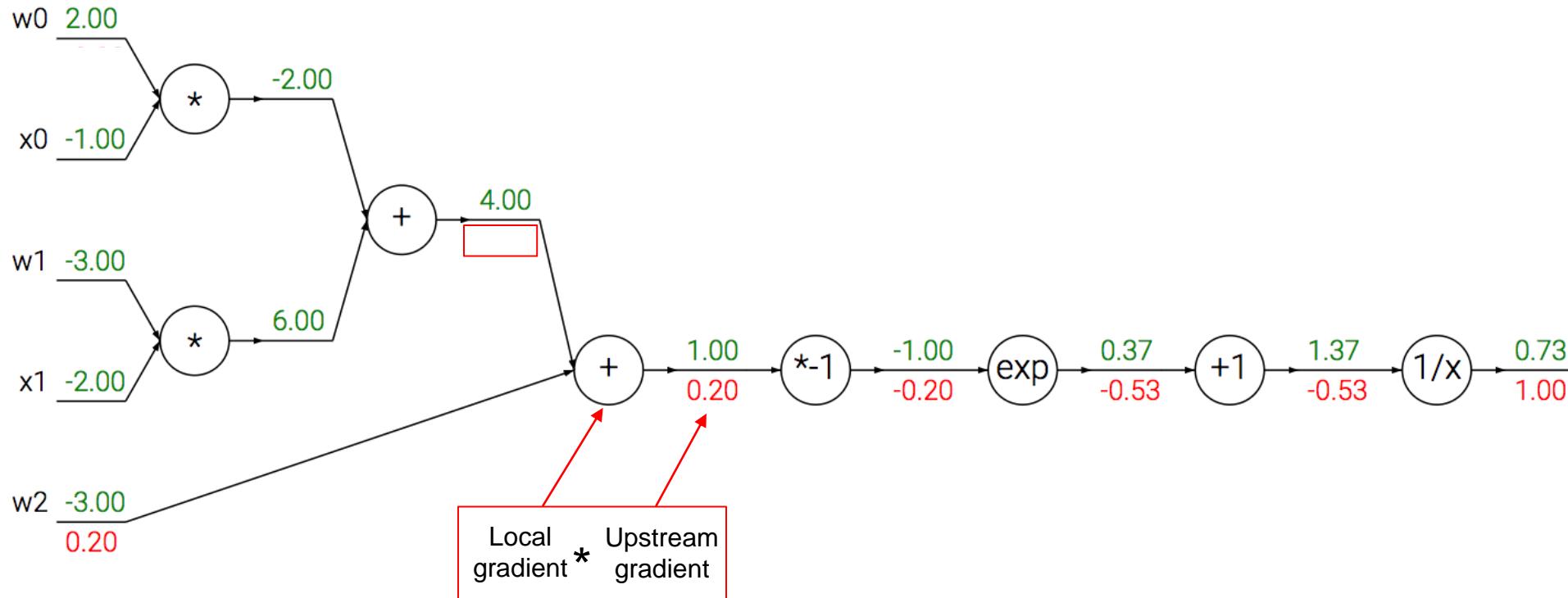
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0 x_0 + w_1 x_1 + w_2)]}$$



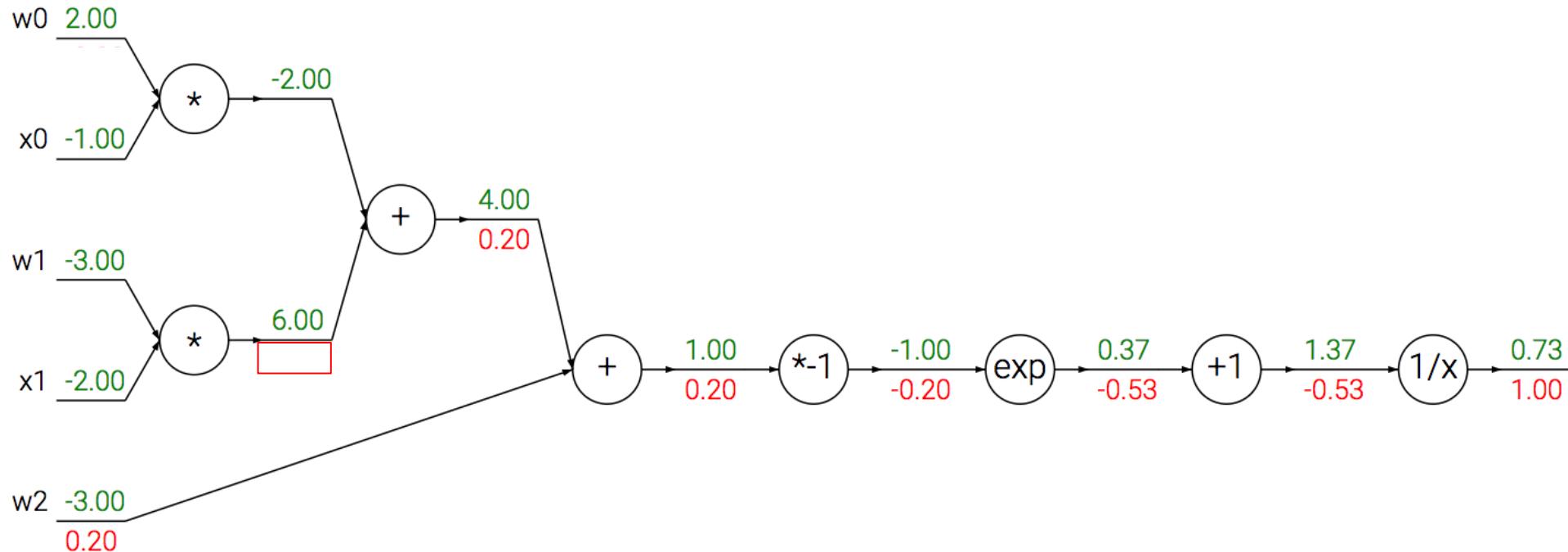
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



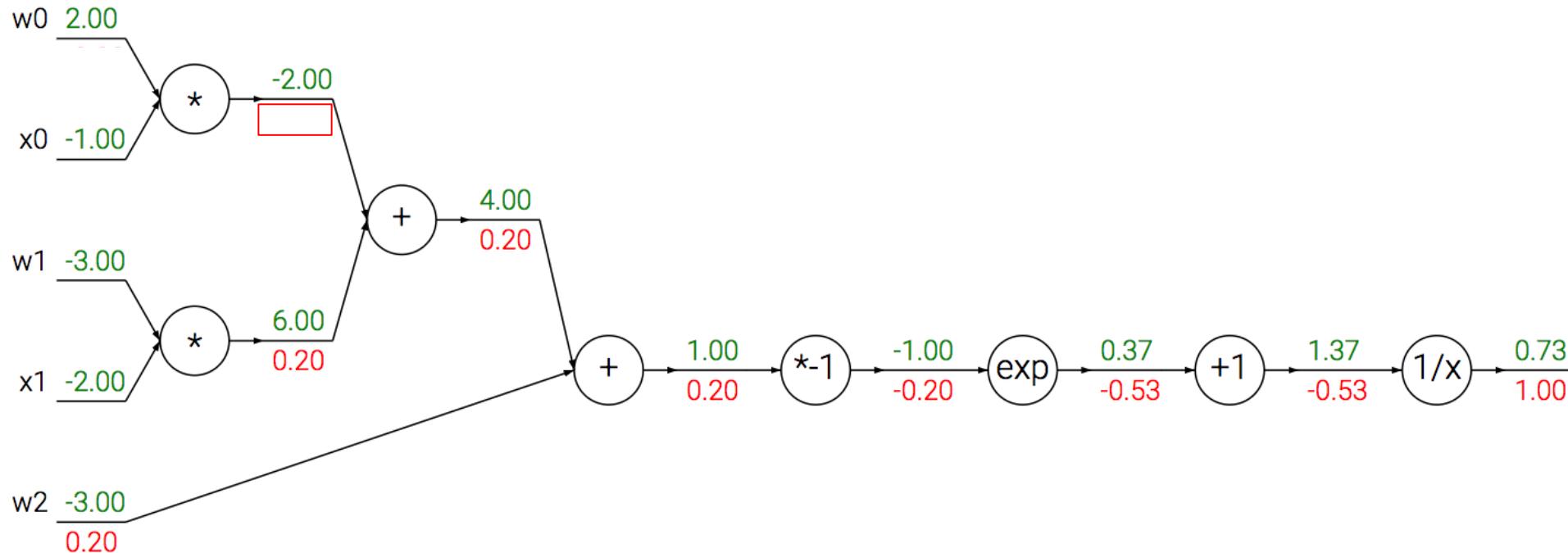
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



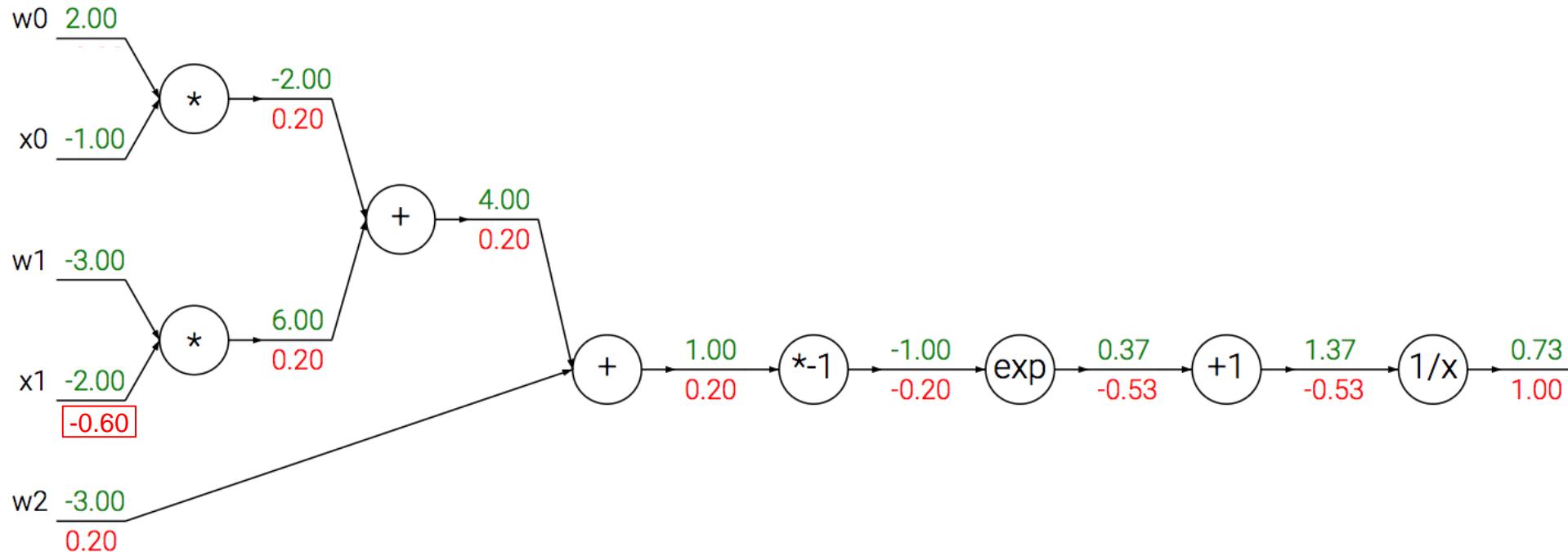
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



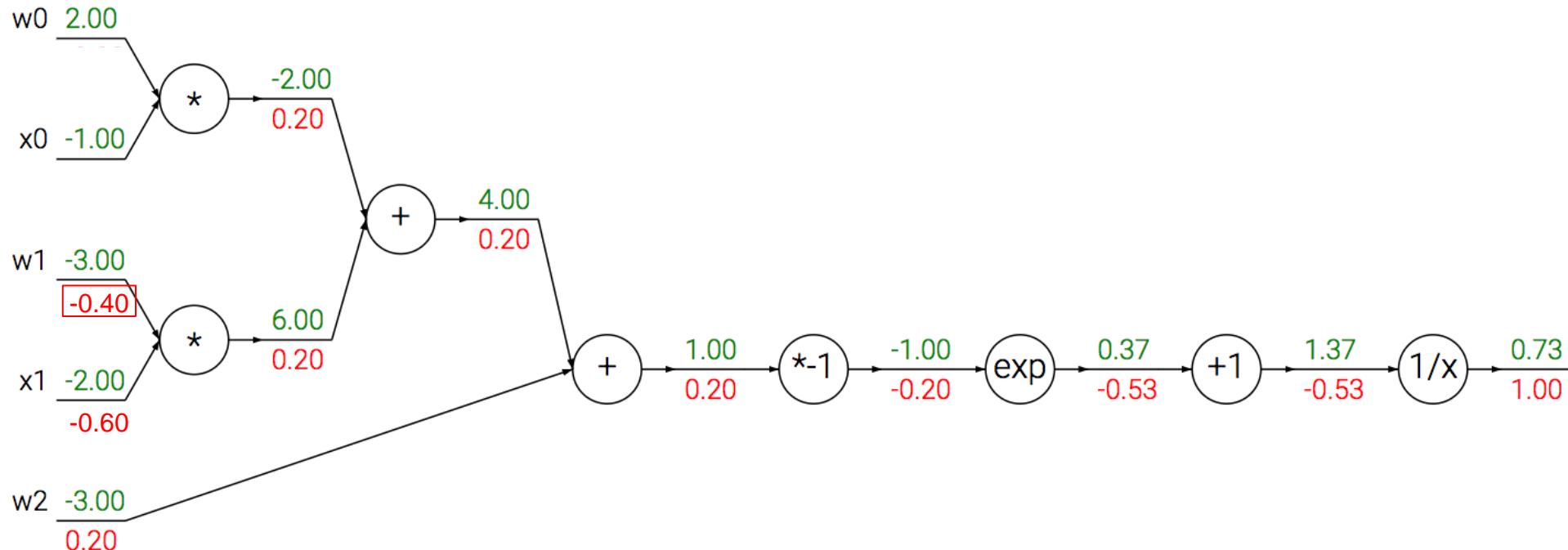
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



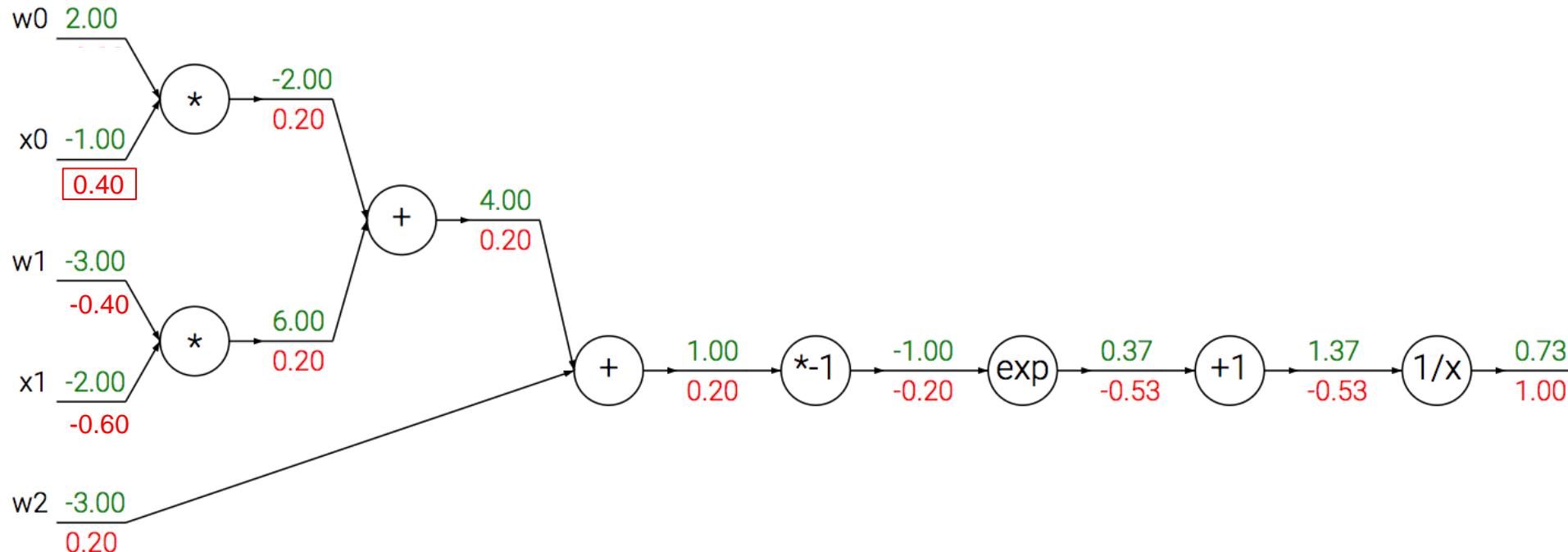
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



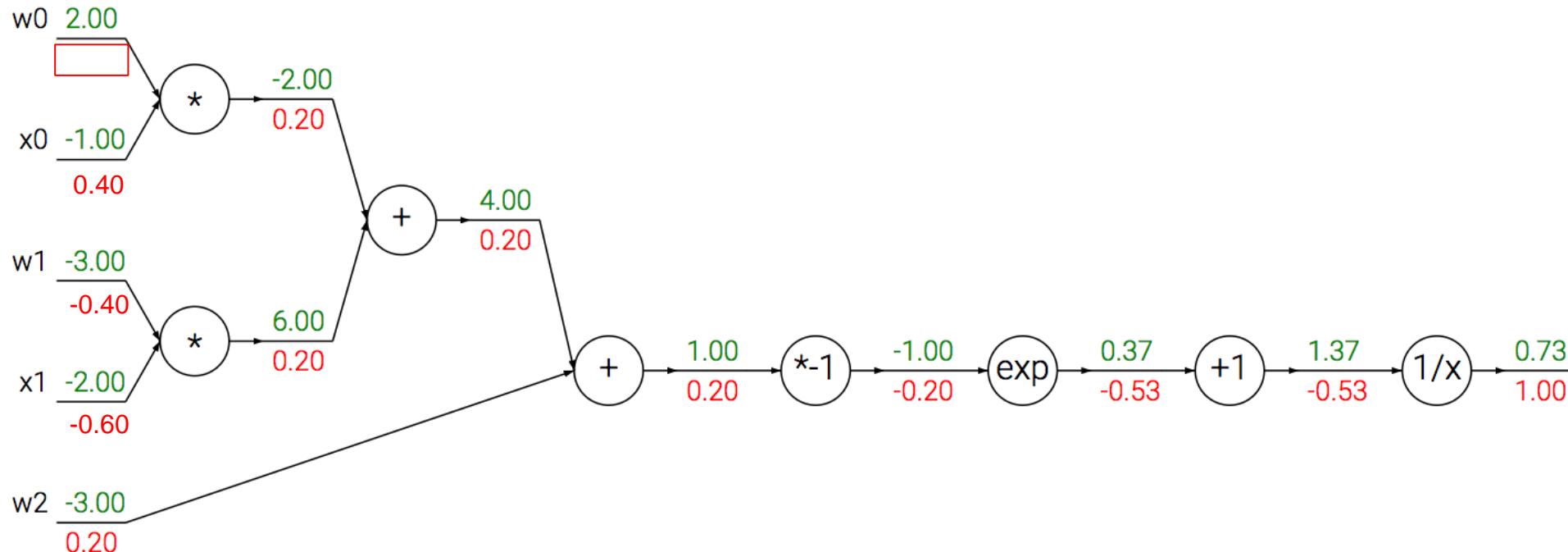
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



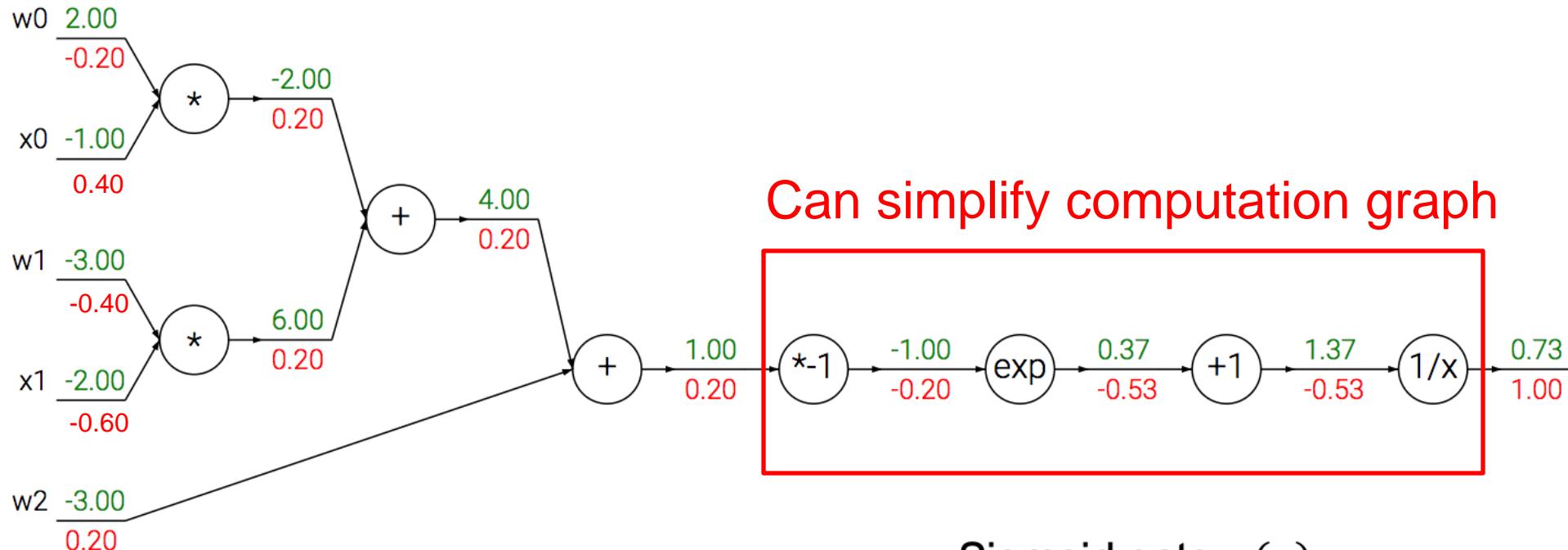
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$



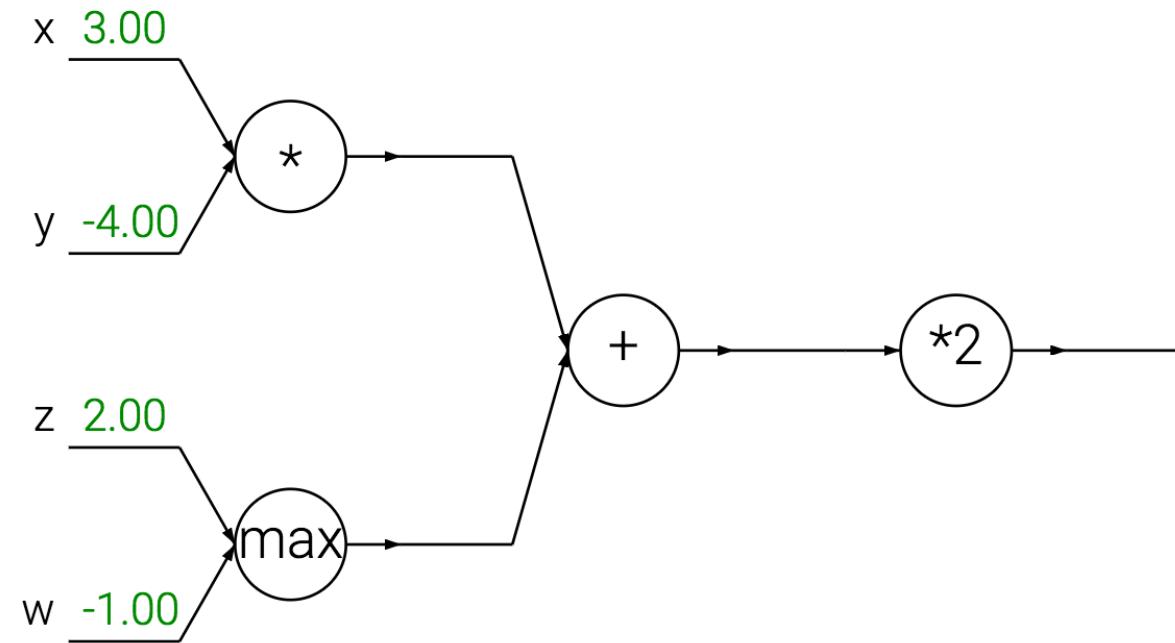
A detailed example

$$f(x, w) = \frac{1}{1 + \exp[-(w_0x_0 + w_1x_1 + w_2)]}$$

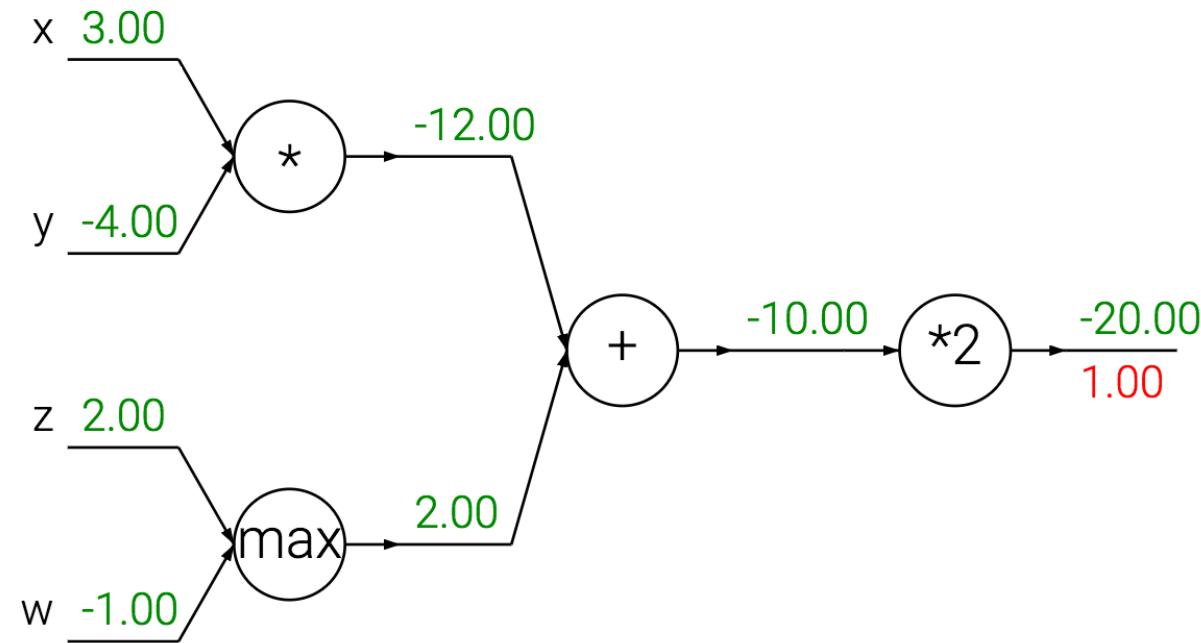


Sigmoid gate $\sigma(x)$
 $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
 $\sigma(1)(1 - \sigma(1)) = 0.73 * (1 - 0.73) = 0.20$

Patterns in gradient flow

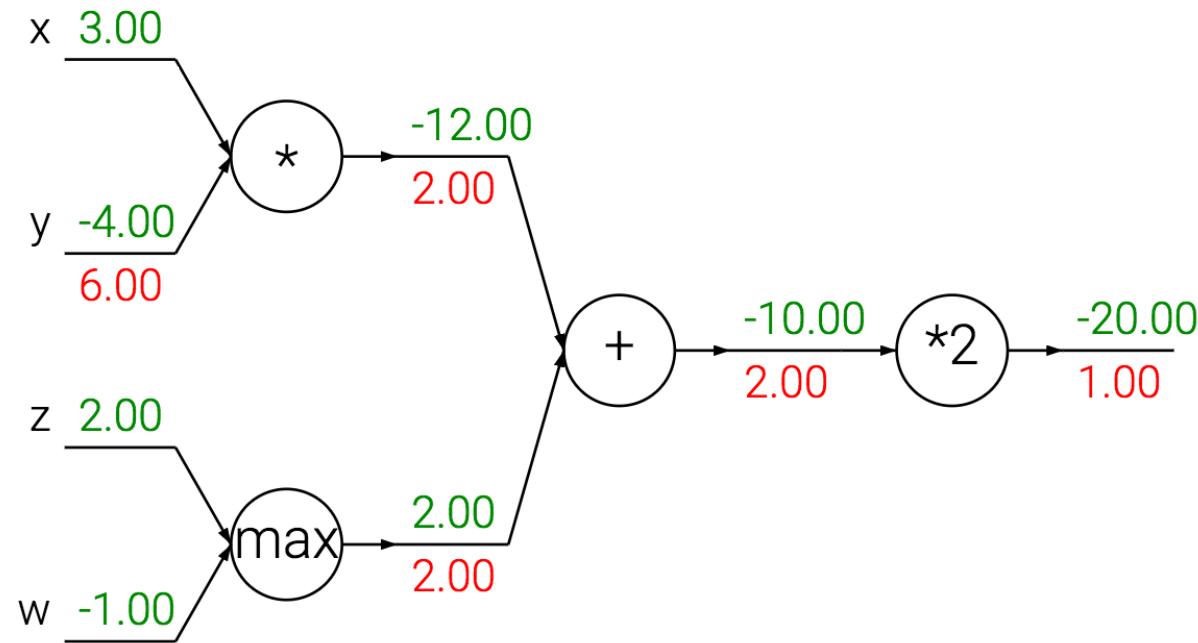


Patterns in gradient flow



Add gate: “gradient distributor”

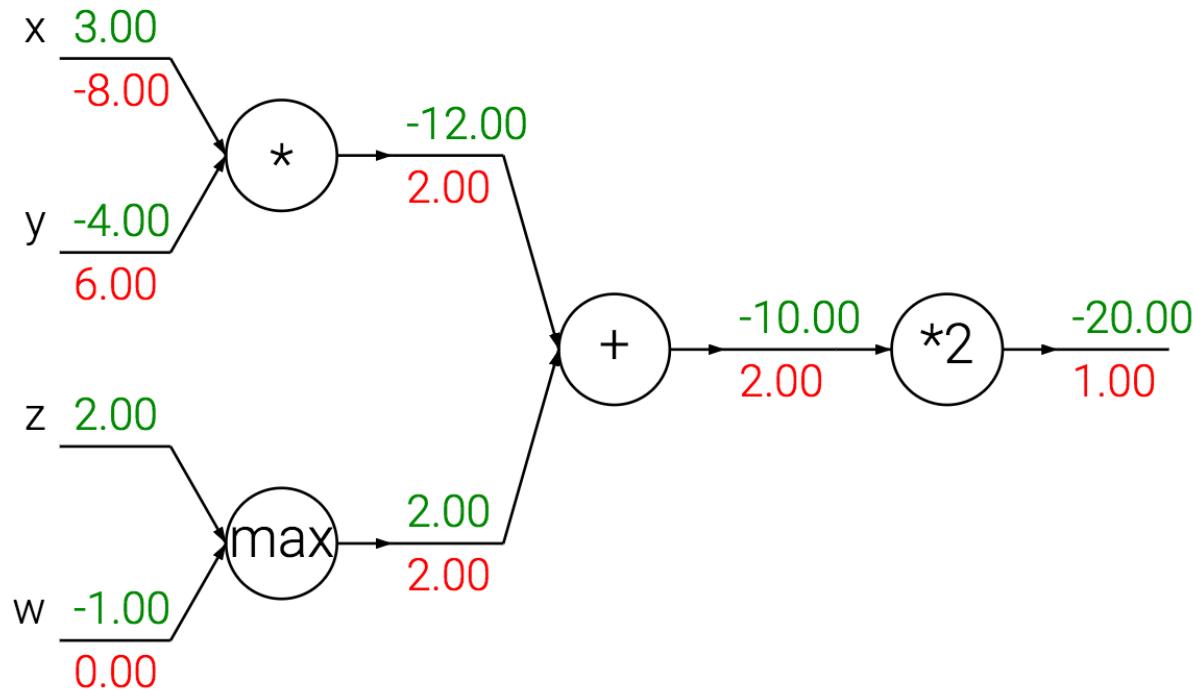
Patterns in gradient flow



Add gate: “gradient distributor”

Multiply gate: “gradient switcher”

Patterns in gradient flow



Add gate: “gradient distributor”

Multiply gate: “gradient switcher”

Max gate: “gradient router”

General tips

- Derive error signal (upstream gradient) directly, avoid explicit computation of huge local derivatives
 - Write out expression for a single element of the Jacobian, then deduce the overall formula
 - Keep consistent indexing conventions, order of operations
 - Use dimension analysis
-
- **For further reading:**
 - Lecture 4 of [Stanford 231n](#)
 - [Yes you should understand backprop](#) by Andrej Karpathy

Acknowledgement

Thanks to the following courses and corresponding researchers for making their teaching/research material online

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University
- And Many More

Dimensionality Reduction

Using Principal Component Analysis

Motivation

- Clustering
 - One way to summarize a complex real-valued data point with a single categorical variable
- Dimensionality reduction
 - Another way to simplify complex high-dimensional data
 - Summarize data with a lower dimensional real valued vector

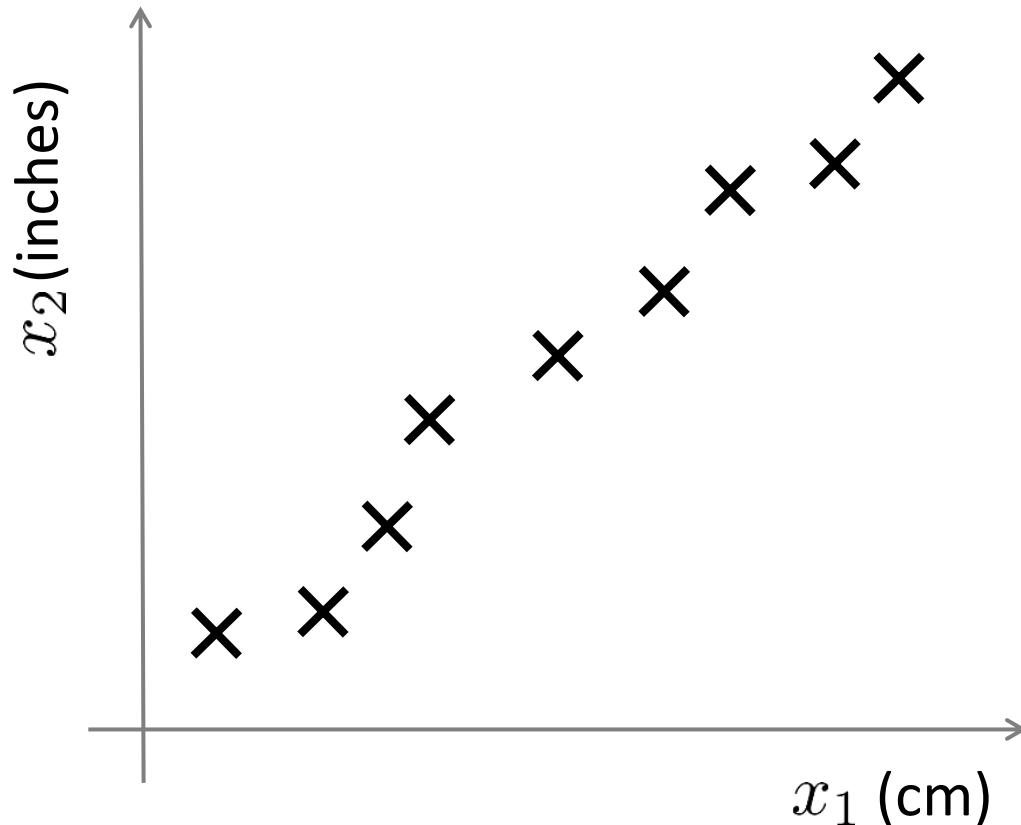
Motivation

- Clustering
 - One way to summarize a complex real-valued data point with a single categorical variable
- Dimensionality reduction
 - Another way to simplify complex high-dimensional data
 - Summarize data with a lower dimensional real valued vector



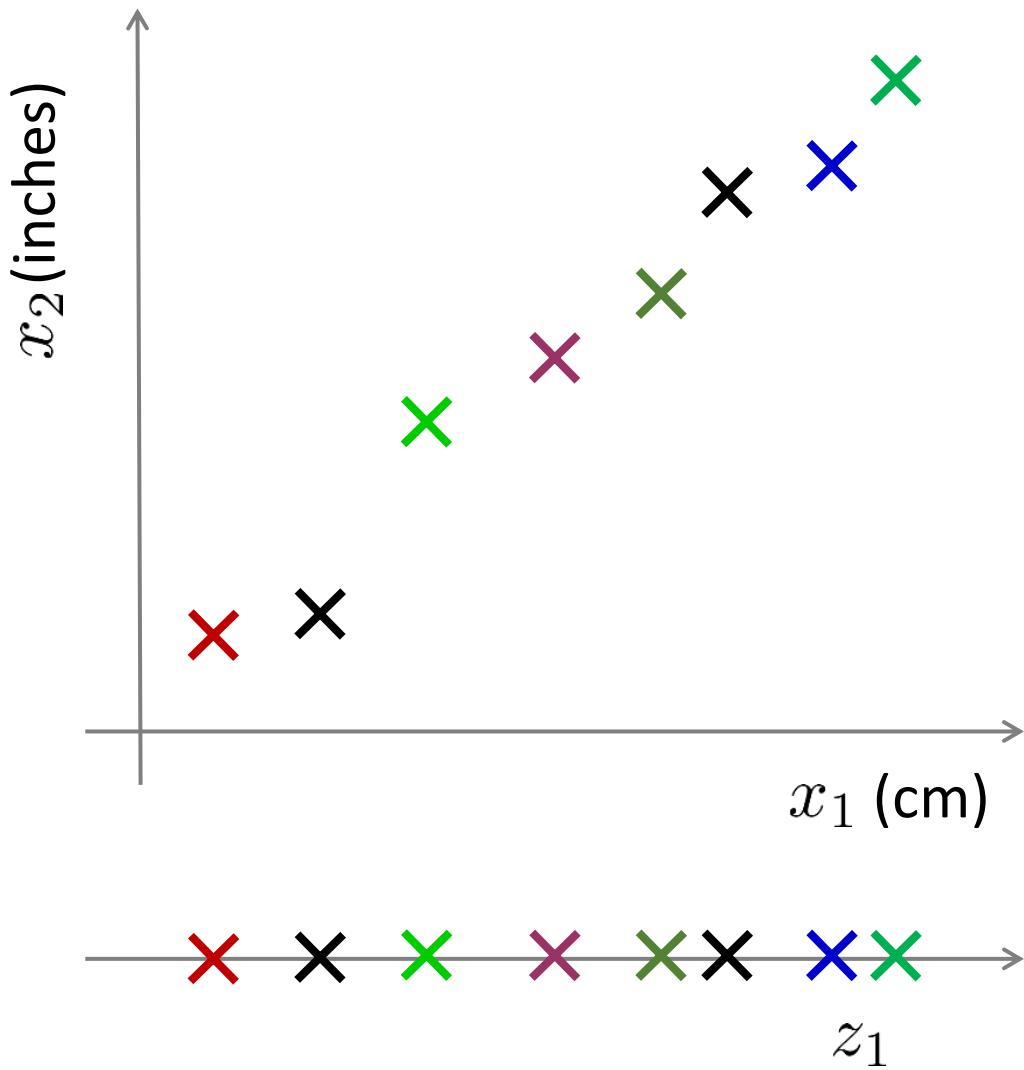
- Given data points in d dimensions
 - Convert them to data points in $r < d$ dimensions
 - With minimal loss of information

Data Compression



Reduce data from
2D to 1D

Data Compression



Reduce data from
2D to 1D

$$x^{(1)} \rightarrow z^{(1)}$$

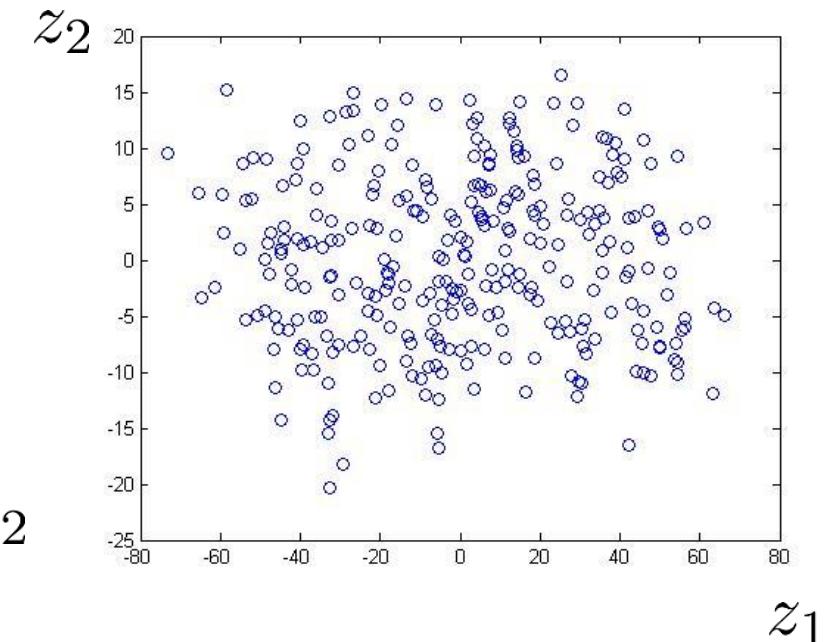
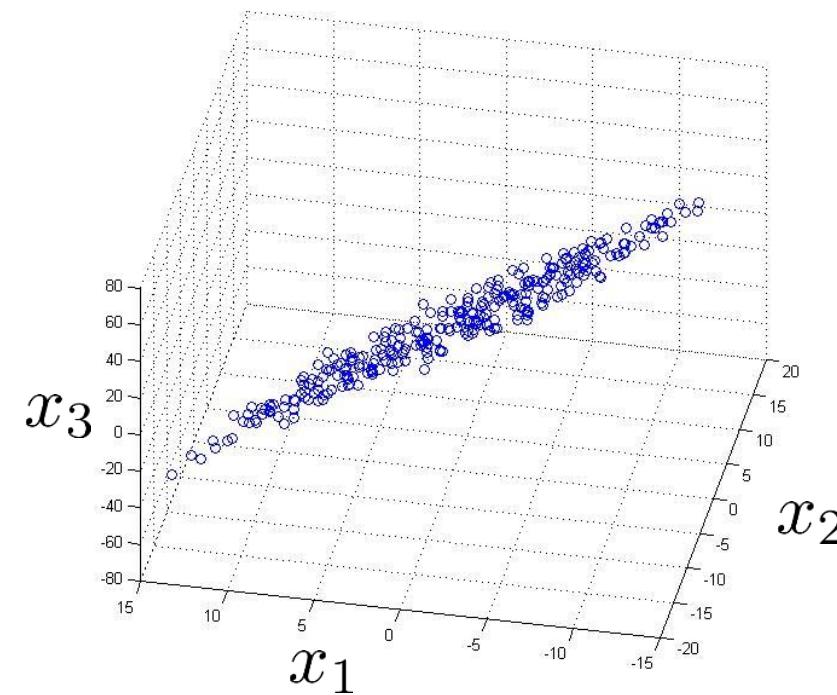
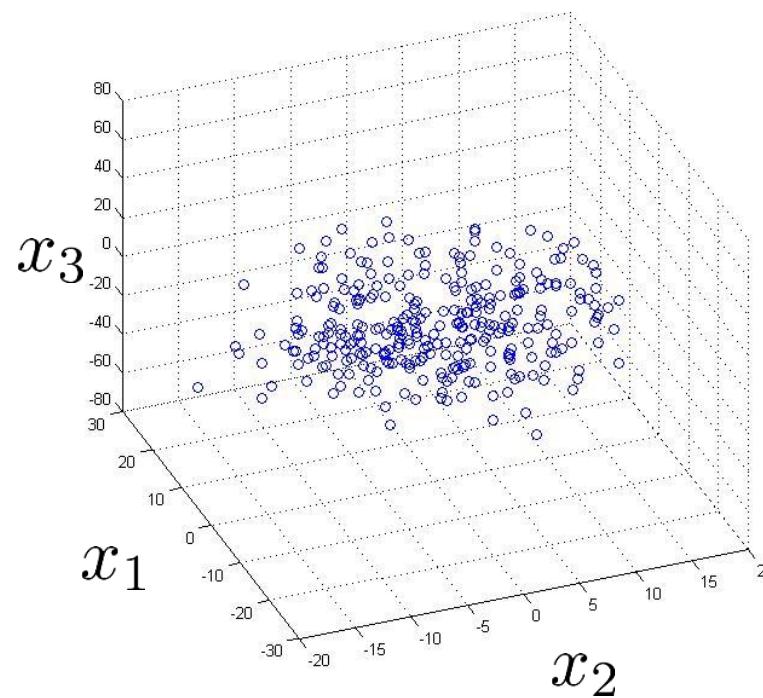
$$x^{(2)} \rightarrow z^{(2)}$$

⋮

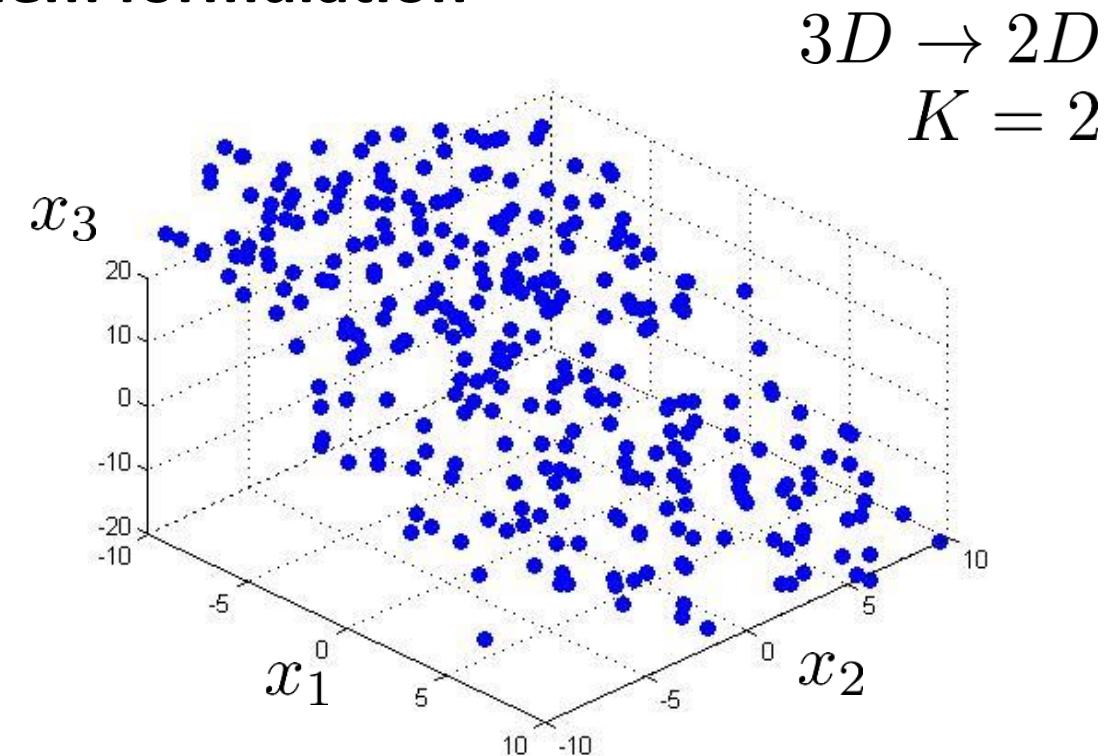
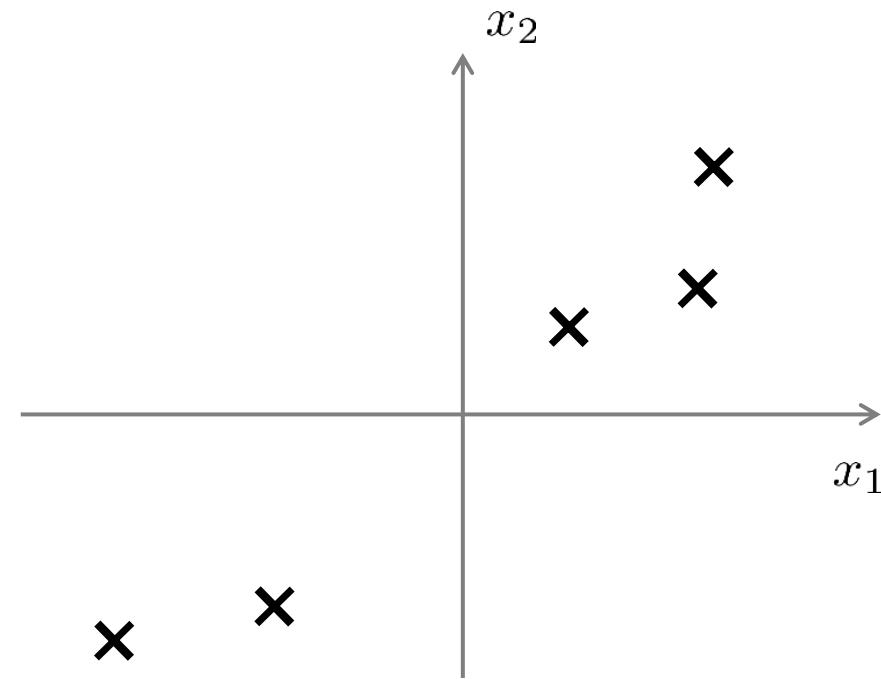
$$x^{(m)} \rightarrow z^{(m)}$$

Data Compression

Reduce data from 3D to 2D



Principal Component Analysis (PCA) problem formulation



Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \dots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Principal Component Analysis

Goal: Find r -dim projection that best preserves variance

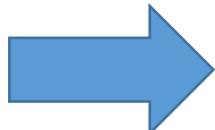
1. Compute mean vector μ and covariance matrix Σ of original points
2. Compute eigenvectors and eigenvalues of Σ
3. Select top r eigenvectors
4. Project points onto subspace spanned by them:

$$y = A(x - \mu)$$

where y is the new point, x is the old one,
and the rows of A are the eigenvectors

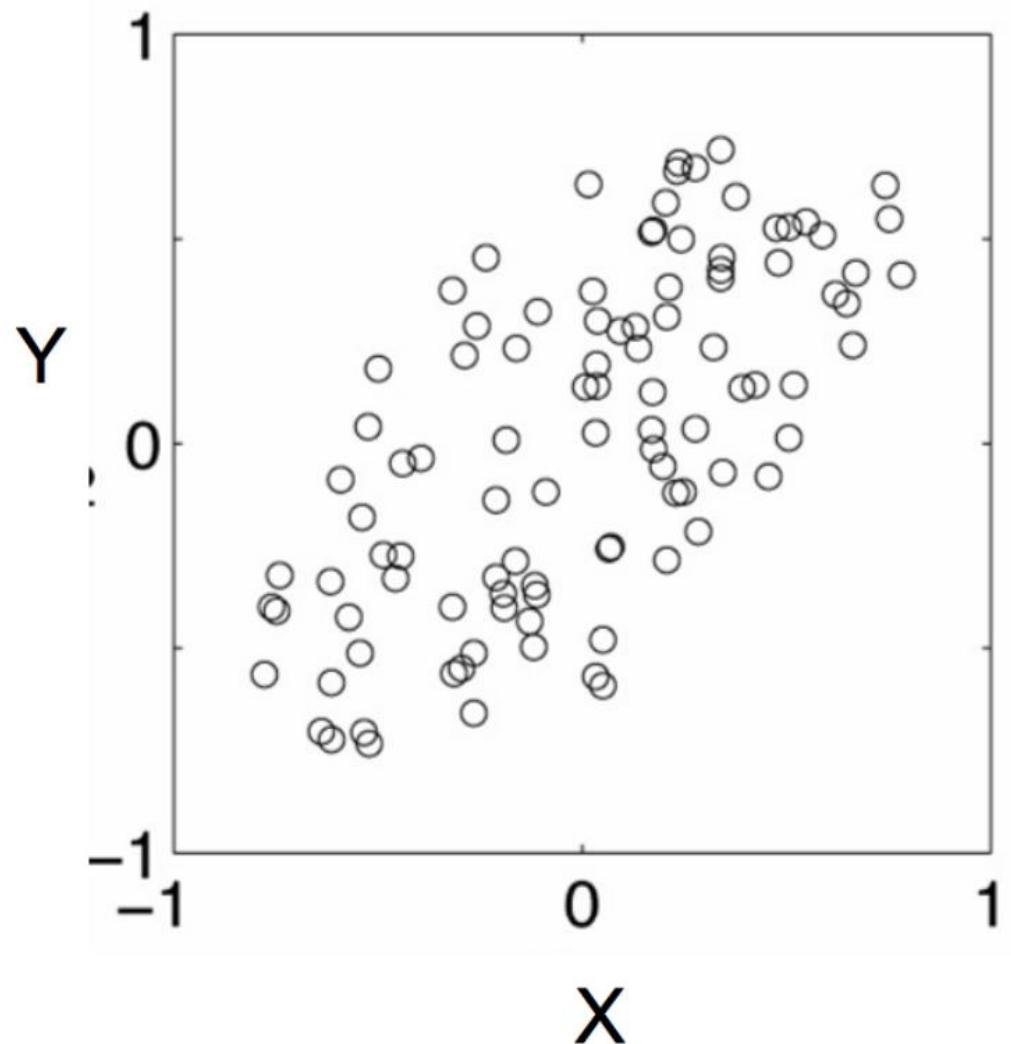
Covariance

- Variance and Covariance:
 - Measure of the “spread” of a set of points around their center of mass(mean)
- Variance:
 - Measure of the deviation from the mean for points in one dimension
- Covariance:
 - Measure of how much each of the dimensions vary from the mean with **respect to each other**



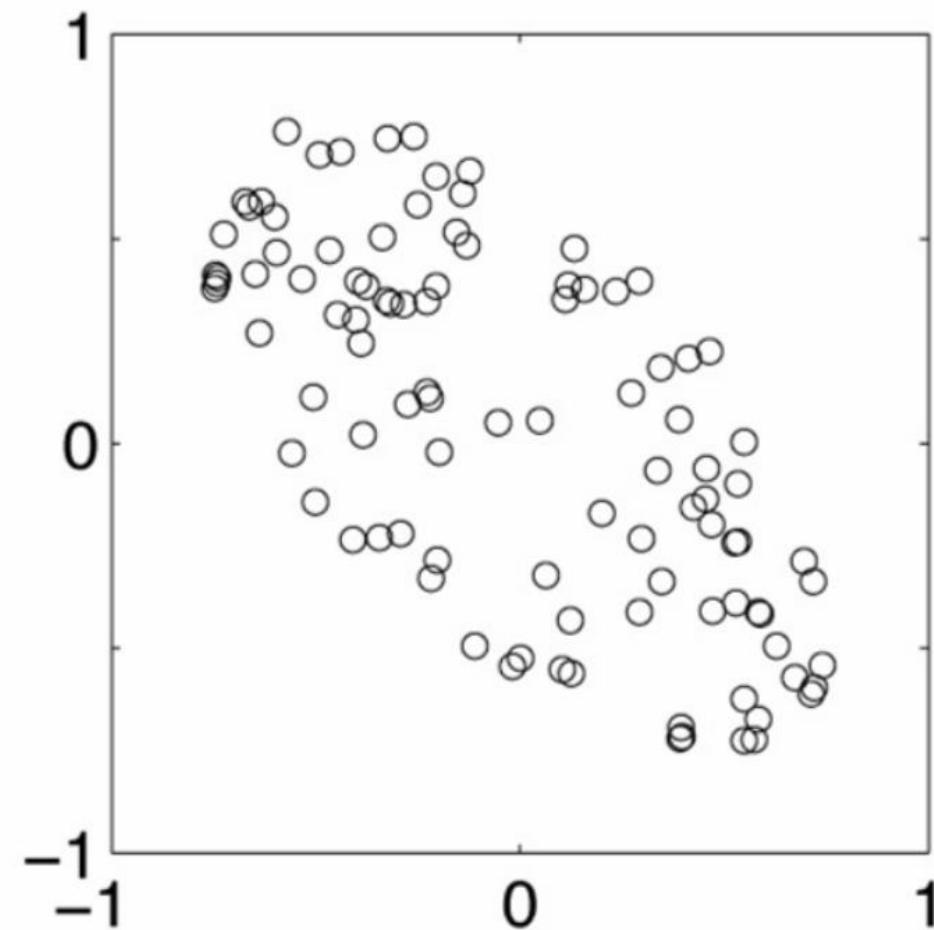
- Covariance is measured between two dimensions
 - Covariance sees if there is a relation between two dimensions
 - Covariance between one dimension is the variance

positive covariance



Positive: Both dimensions increase or decrease together

negative covariance



Negative: While one increase the other decrease

Covariance

- Used to find relationships between dimensions in high dimensional data sets

$$q_{jk} = \frac{1}{N} \sum_{i=1}^N (X_{ij} - E(X_j))(X_{ik} - E(X_k))$$


The Sample mean

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ : Eigenvector or characteristic vector

X: Eigenvalue or characteristic value



- *The zero vector can not be an eigenvector*
- *The value zero can be eigenvalue*

Eigenvector and Eigenvalue

$$Ax = \lambda x$$

A: Square Matrix

λ : Eigenvector or characteristic vector

X: Eigenvalue or characteristic value

Example

Show $x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ is an eigenvector for $A = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix}$

$$\text{Solution: } Ax = \begin{bmatrix} 2 & -4 \\ 3 & -6 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\text{But for } \lambda = 0, \lambda x = 0 \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Thus, x is an eigenvector of A , and $\lambda = 0$ is an eigenvalue.

Eigenvector and Eigenvalue

$$Ax = \lambda x$$



$$Ax - \lambda x = 0$$

$$(A - \lambda I)x = 0$$

If we define a new matrix B:



$$B = A - \lambda I$$

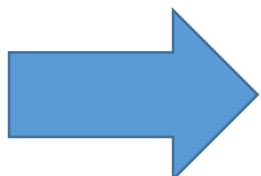
$$Bx = 0$$

If B has an inverse:



$$x = B^{-1}0 = 0$$

X BUT! an eigenvector cannot be zero!!



x will be an eigenvector of A if and only if B does not have an inverse, or equivalently $\det(B)=0$:

$$\det(A - \lambda I) = 0$$

Eigenvector and Eigenvalue

Example 1: Find the eigenvalues of

$$A = \begin{bmatrix} 2 & -12 \\ 1 & -5 \end{bmatrix}$$

$$\begin{aligned} |\lambda I - A| &= \begin{vmatrix} \lambda - 2 & 12 \\ -1 & \lambda + 5 \end{vmatrix} = (\lambda - 2)(\lambda + 5) + 12 \\ &= \lambda^2 + 3\lambda + 2 = (\lambda + 1)(\lambda + 2) \end{aligned}$$

two eigenvalues: $-1, -2$

Note: The roots of the characteristic equation can be repeated. That is, $\lambda_1 = \lambda_2 = \dots = \lambda_k$. If that happens, the eigenvalue is said to be of multiplicity k.

Example 2: Find the eigenvalues of

$$A = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$$|\lambda I - A| = \begin{vmatrix} \lambda - 2 & -1 & 0 \\ 0 & \lambda - 2 & 0 \\ 0 & 0 & \lambda - 2 \end{vmatrix} = (\lambda - 2)^3 = 0$$

$\lambda = 2$ is an eigenvector of multiplicity 3.

Principal Component Analysis

Input:

$$\mathbf{x} \in \mathbb{R}^D: \mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$$

Set of basis vectors: $\mathbf{u}_1, \dots, \mathbf{u}_K$

Summarize a D dimensional vector X with K dimensional feature vector $h(\mathbf{x})$

$$h(\mathbf{x}) = \begin{bmatrix} \mathbf{u}_1 \cdot \mathbf{x} \\ \mathbf{u}_2 \cdot \mathbf{x} \\ \vdots \\ \mathbf{u}_K \cdot \mathbf{x} \end{bmatrix}$$

Principal Component Analysis

$$\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_K]$$

Basis vectors are orthonormal

$$\mathbf{u}_i^T \mathbf{u}_j = 0$$

$$\|\mathbf{u}_j\| = 1$$

New data representation $h(\mathbf{x})$

$$z_j = \mathbf{u}_j \cdot \mathbf{x}$$

$$h(\mathbf{x}) = [z_1, \dots, z_K]^T$$

Principal Component Analysis

$$\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_K]$$

New data representation $h(\mathbf{x})$

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

$$h(\mathbf{x}) = \mathbf{U}^T (\mathbf{x} - \mu_0)$$

Empirical mean of the data


$$\mu_0 = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i$$

The space of all face images

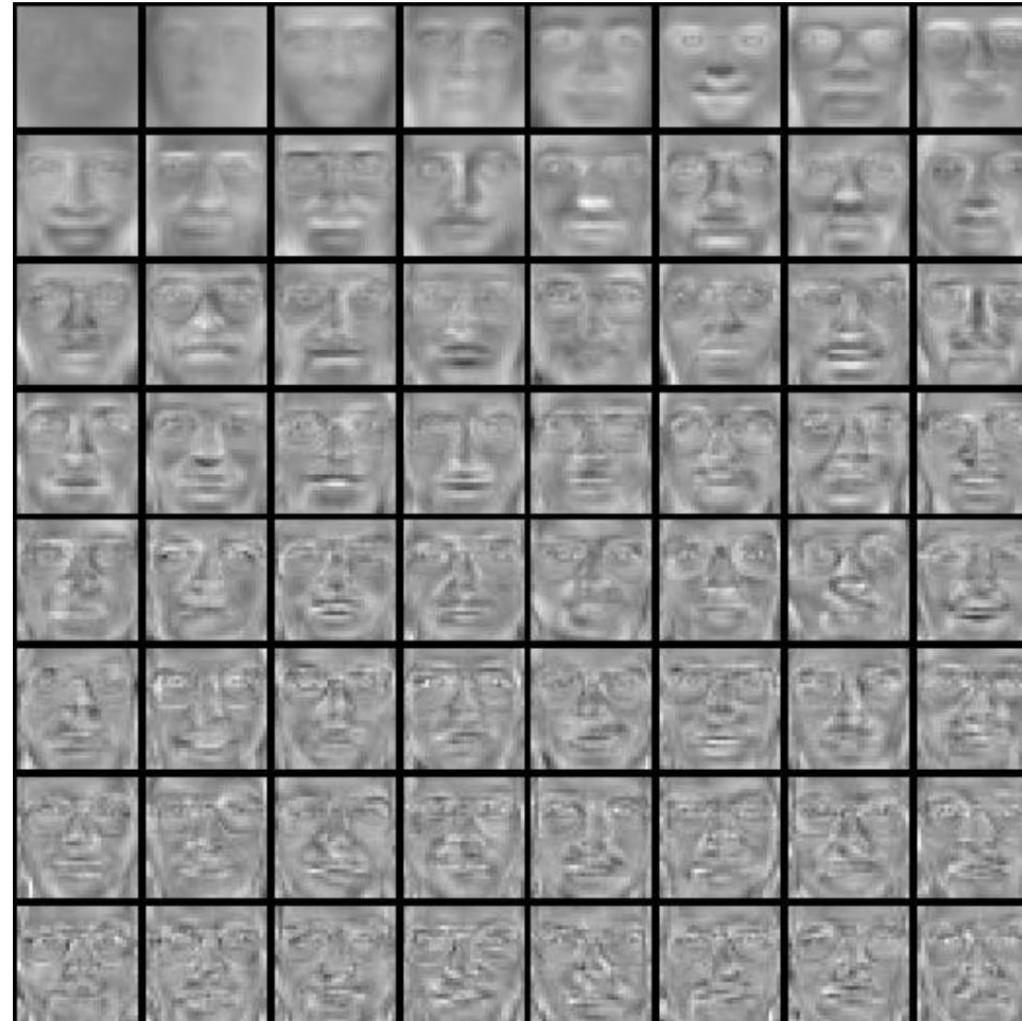
- When viewed as vectors of pixel values, face images are extremely high-dimensional
 - 100×100 image = 10,000 dimensions
 - Slow and lots of storage
- But very few 10,000-dimensional vectors are valid face images
- We want to effectively model the subspace of face images



Eigenfaces example

Top eigenvectors: u_1, \dots, u_k

Mean: μ



Representation and reconstruction

- Face \mathbf{x} in “face space” coordinates:



$$\begin{aligned}\mathbf{x} &\rightarrow [\mathbf{u}_1^T(\mathbf{x} - \mu), \dots, \mathbf{u}_k^T(\mathbf{x} - \mu)] \\ &= w_1, \dots, w_k\end{aligned}$$

- Reconstruction:

$$\begin{aligned}\hat{\mathbf{x}} &= \mathbf{\mu} + w_1 \mathbf{u}_1 + w_2 \mathbf{u}_2 + w_3 \mathbf{u}_3 + w_4 \mathbf{u}_4 + \dots \\ &= \mathbf{\mu} + w_1 \mathbf{u}_1 + w_2 \mathbf{u}_2 + w_3 \mathbf{u}_3 + w_4 \mathbf{u}_4 + \dots\end{aligned}$$

The diagram illustrates the reconstruction of a face. On the left is the original face image. An equals sign follows, then a mean face image, then a plus sign, then a row of seven basis face images. Below this is the equation for the reconstruction of the face using the mean face and the weighted sum of the basis faces.

Reconstruction

$P = 4$



$P = 200$



$P = 400$



After computing eigenfaces using 400 face images from ORL face database

Application: Image compression



Original Image

- Divide the original 372x492 image into patches:
 - Each patch is an instance that contains 12x12 pixels on a grid
 - View each as a 144-D vector

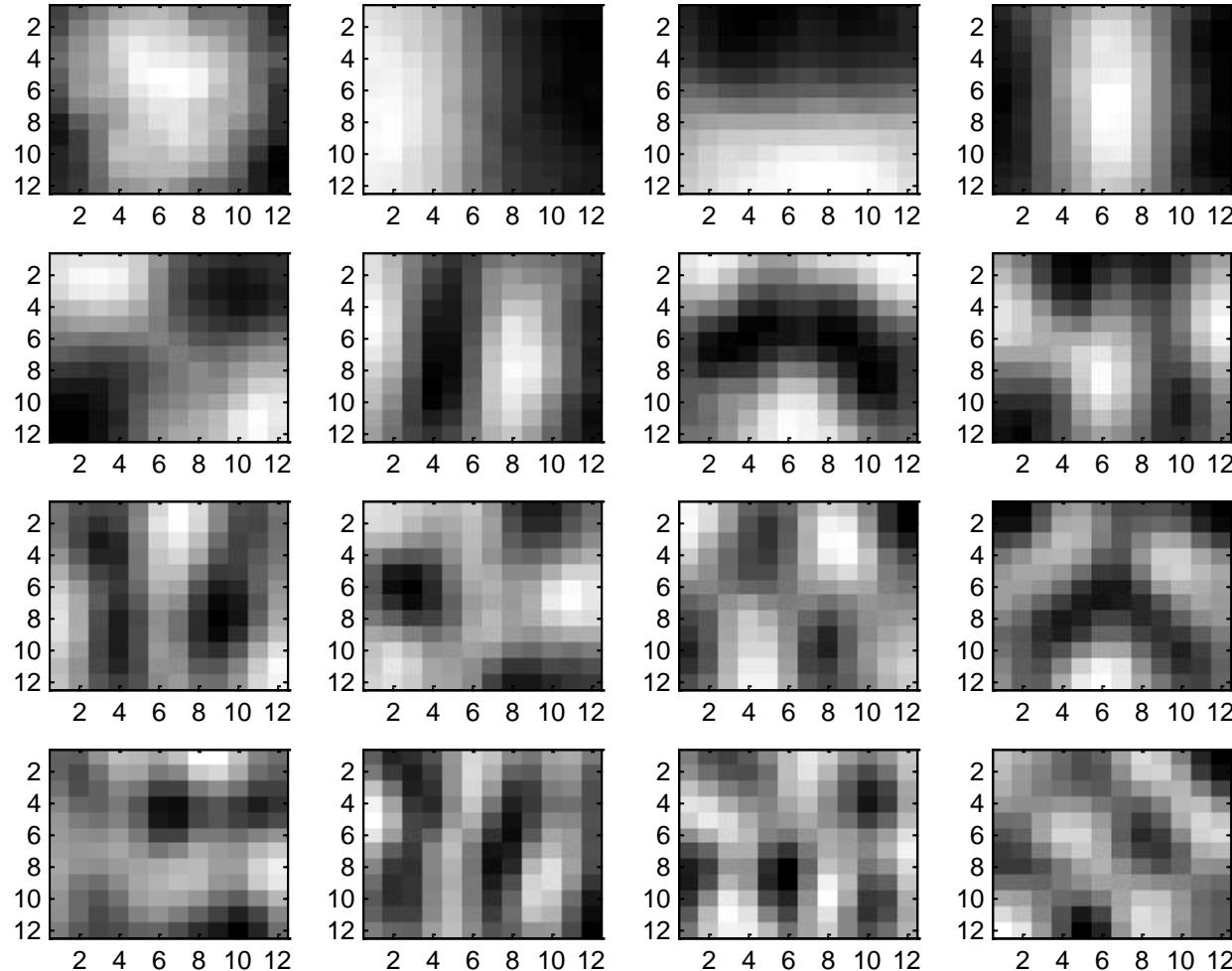
PCA compression: 144D → 60D



PCA compression: 144D → 16D



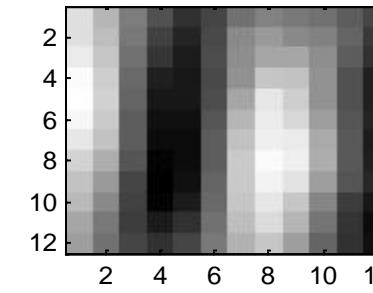
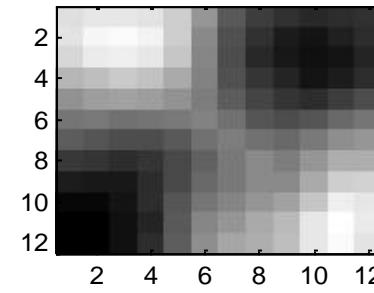
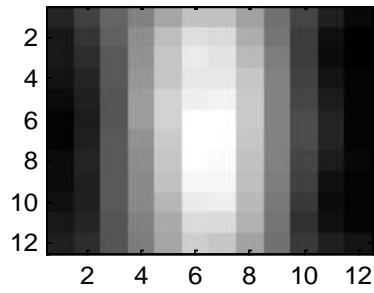
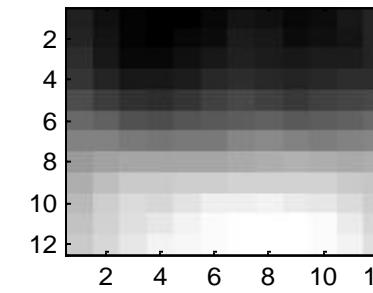
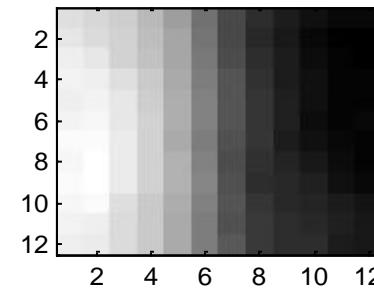
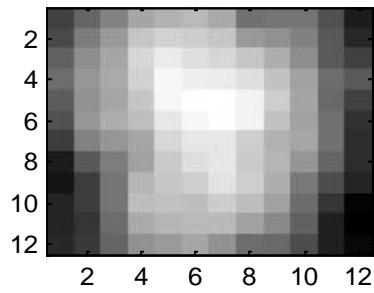
16 most important eigenvectors



PCA compression: 144D → 6D



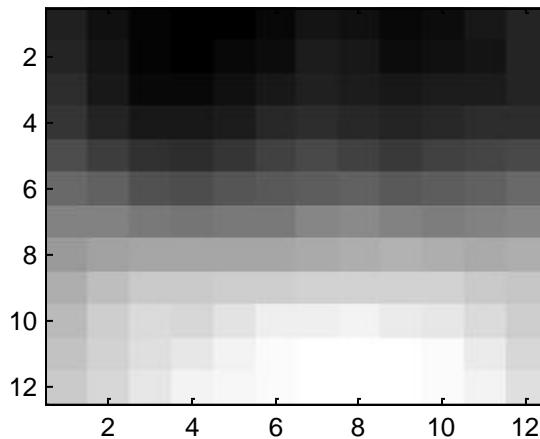
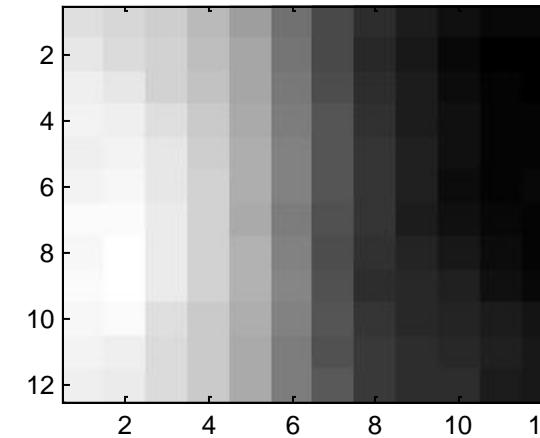
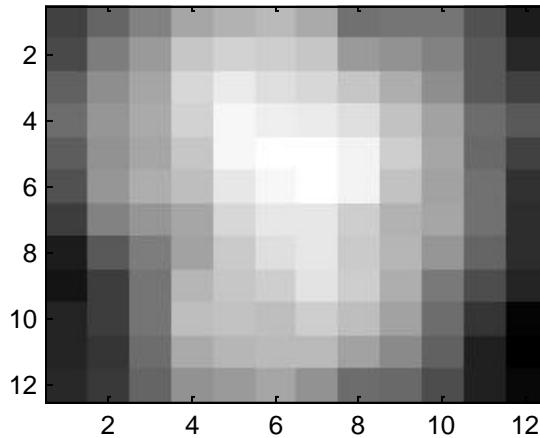
6 most important eigenvectors



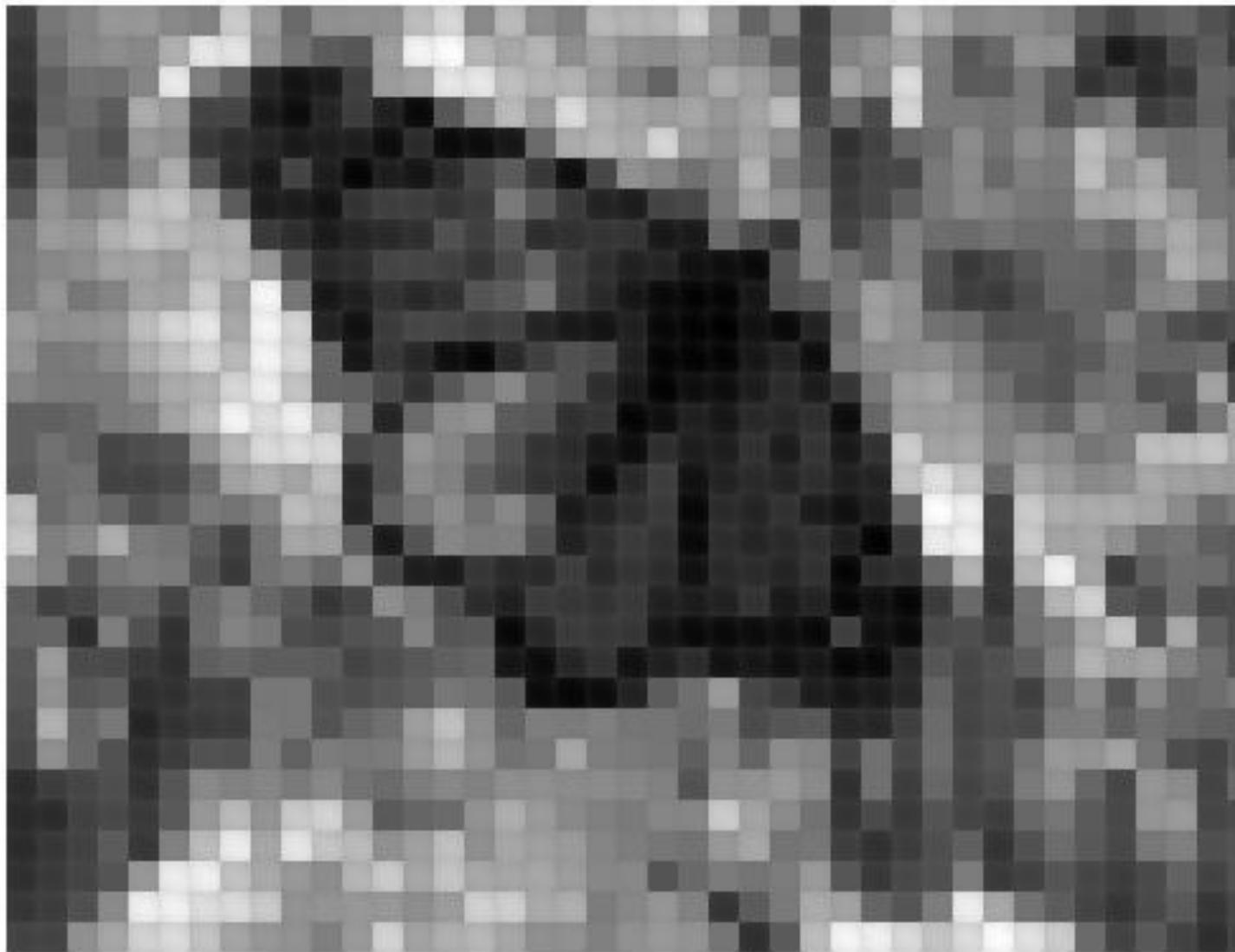
PCA compression: 144D \rightarrow 3D



3 most important eigenvectors



PCA compression: 144D \rightarrow 1D



An example

Data

x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Data	
<i>x</i>	<i>y</i>
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Mean subtracted Data

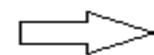
<i>x</i>	<i>y</i>
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Data	
<i>x</i>	<i>y</i>
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Mean subtracted Data

<i>x</i>	<i>y</i>
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01



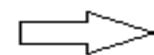
$$cov = \begin{pmatrix} .61 & .61 \\ .61 & .71 \end{pmatrix}$$

Data	
<i>x</i>	<i>y</i>
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Mean subtracted Data

<i>x</i>	<i>y</i>
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01



$$cov = \begin{pmatrix} .61 & .61 \\ .61 & .71 \end{pmatrix}$$

Data	
<i>x</i>	<i>y</i>
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Mean subtracted Data

<i>x</i>	<i>y</i>
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-.101



$$cov = \begin{pmatrix} .61 & .61 \\ .61 & .71 \end{pmatrix}$$

Eigenvalues are 1.28, 0.049

Corresponding Eigenvectors are $\begin{pmatrix} -.67 \\ -.73 \end{pmatrix}$ and $\begin{pmatrix} -.73 \\ .67 \end{pmatrix}$

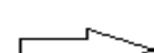
What is the projection matrix?

If we are interested in 1D data

Data	
x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Mean subtracted Data	
x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-.101



$$cov = \begin{pmatrix} .61 & .61 \\ .61 & .71 \end{pmatrix}$$

Eigenvalues are 1.28, 0.049

Corresponding Eigenvectors are $\begin{pmatrix} -.67 \\ -.73 \end{pmatrix}$ and $\begin{pmatrix} -.73 \\ .67 \end{pmatrix}$

What is the projection matrix?

$$\mathbf{P} = \begin{pmatrix} -.67 \\ -.73 \end{pmatrix}^t$$

Transformed Data (Single eigenvector)

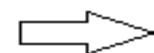
x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056

The data after transforming using only the most significant eigenvector

Data	
<i>x</i>	<i>y</i>
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9



Mean subtracted Data	
<i>x</i>	<i>y</i>
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-.101



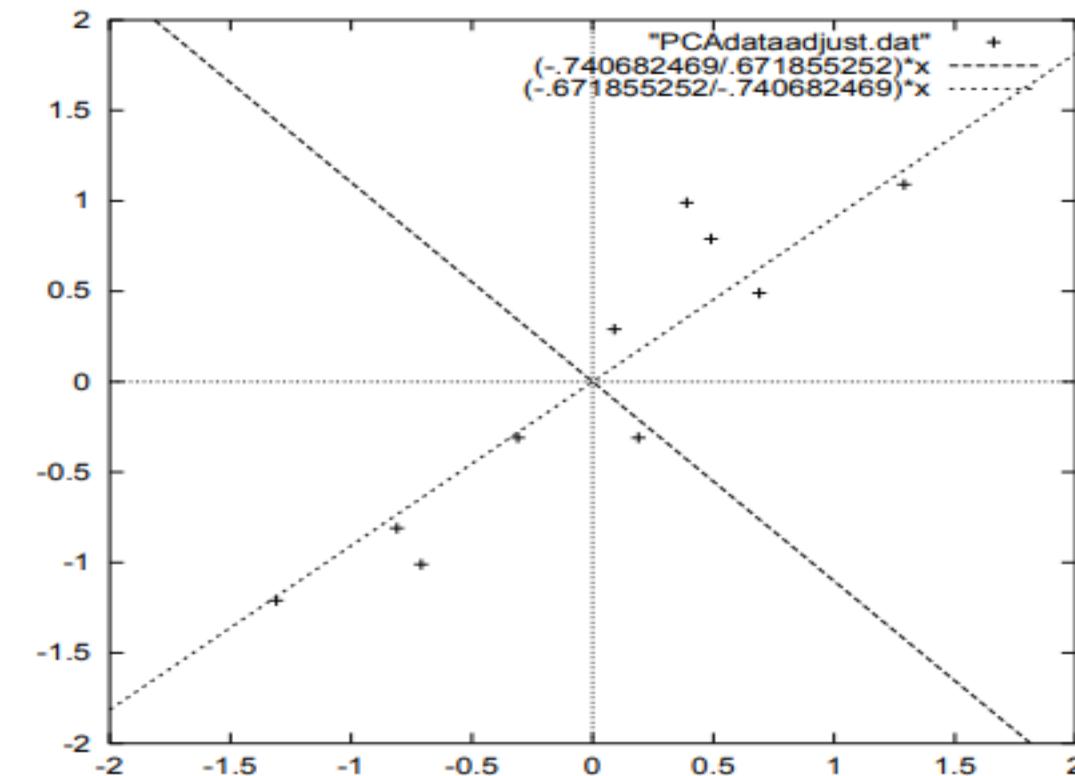
$$cov = \begin{pmatrix} .61 & .61 \\ .61 & .71 \end{pmatrix}$$

Eigenvalues are 1.28, 0.049

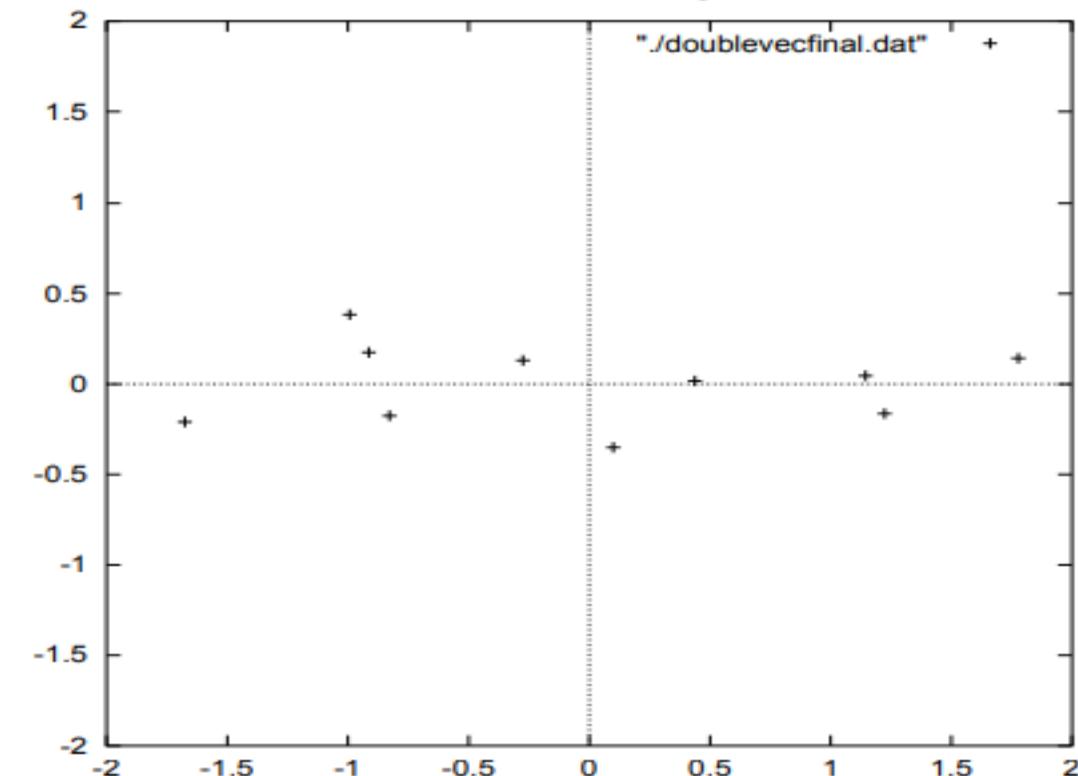
Corresponding Eigenvectors are $\begin{pmatrix} -.67 \\ -.73 \end{pmatrix}$ and $\begin{pmatrix} -.73 \\ .67 \end{pmatrix}$

We can get a 2D data by using both eigenvectors,

Mean adjusted data with eigenvectors overlayed



Data transformed with 2 eigenvectors



Dimensionality reduction

- PCA (Principal Component Analysis):
 - Find projection that maximize the variance
- ICA (Independent Component Analysis):
 - Very similar to PCA except that it assumes non-Gaussian features
- Multidimensional Scaling:
 - Find projection that best preserves inter-point distances
- LDA(Linear Discriminant Analysis):
 - Maximizing the component axes for class-separation
- ...
- ...