Supervised Learning (contd)
Linear Separation

Mausam
(based on slides by UW-AI faculty)

Images as Vectors

Binary handwritten characters

```
00000000010000000000
00000000110000000000
00000000101000000000
00000001000010000000
00000010000010000000
00000100000000100000
00001000000000100000
00001100111111110000
00001111110000010000
00011000000000011000
00010000000000001100
00110000000000000100
00110000000000000110
00100000000000000010
00100000000000000010
01100000000000000010
01000000000000000000
00000000000000000000
```

```
00000000011110000000
00000001100001100000
00000011000000110000
00001100000000011000
00001000000000001000
00001100000000010000
00000111000000100000
00000011100111100000
00000000111100000000
00000011000111000000
00001100000000110000
00011000000000011000
00110000000000001000
00100000000000001100
00010000000000011000
00011000000000010000
00001000000000110000
00000011111111000000
```

Greyscale images

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

.....

⋮

Treat an image as a high-dimensional vector
(e.g., by reading pixel values left to right, top to bottom row)

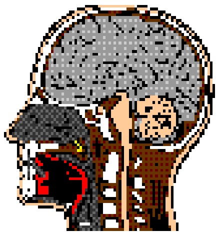
$$\mathbf{I} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{N-2} \\ p_N \end{bmatrix}$$

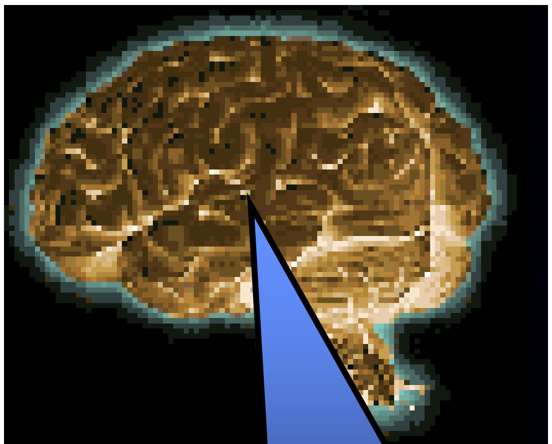
Pixel value p_i can be 0 or 1 (binary image) or 0 to 255 (greyscale)

The human brain is extremely good at classifying images

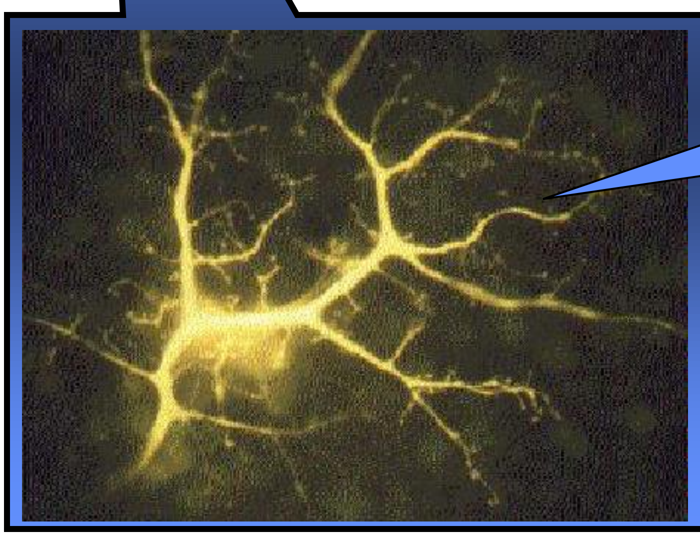
Can we develop classification methods by emulating the brain?

Brain Computer: What is it?





Human brain contains a massively interconnected net of 10^{10}-10^{11} (10 billion) neurons (cortical cells)

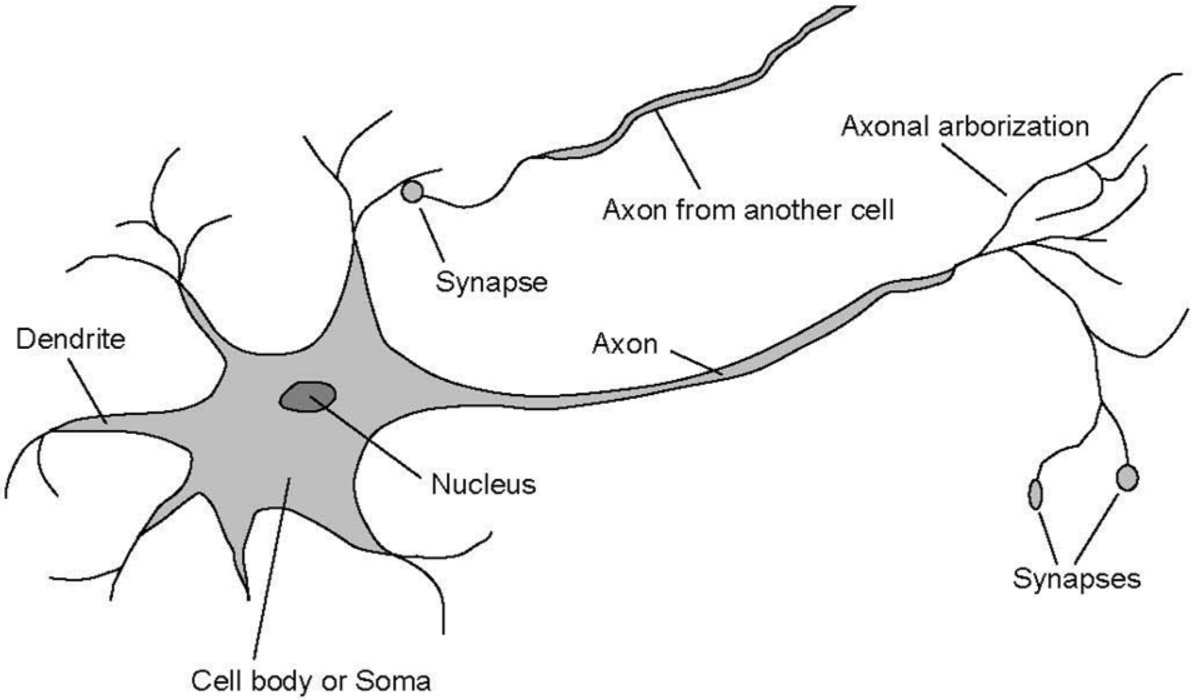


Biological Neuron

- The simple "arithmetic computing" element

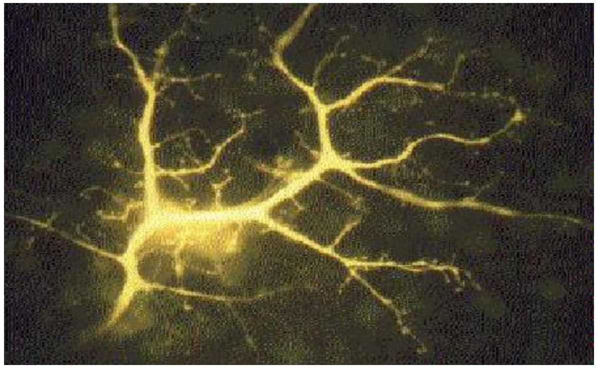
Brains

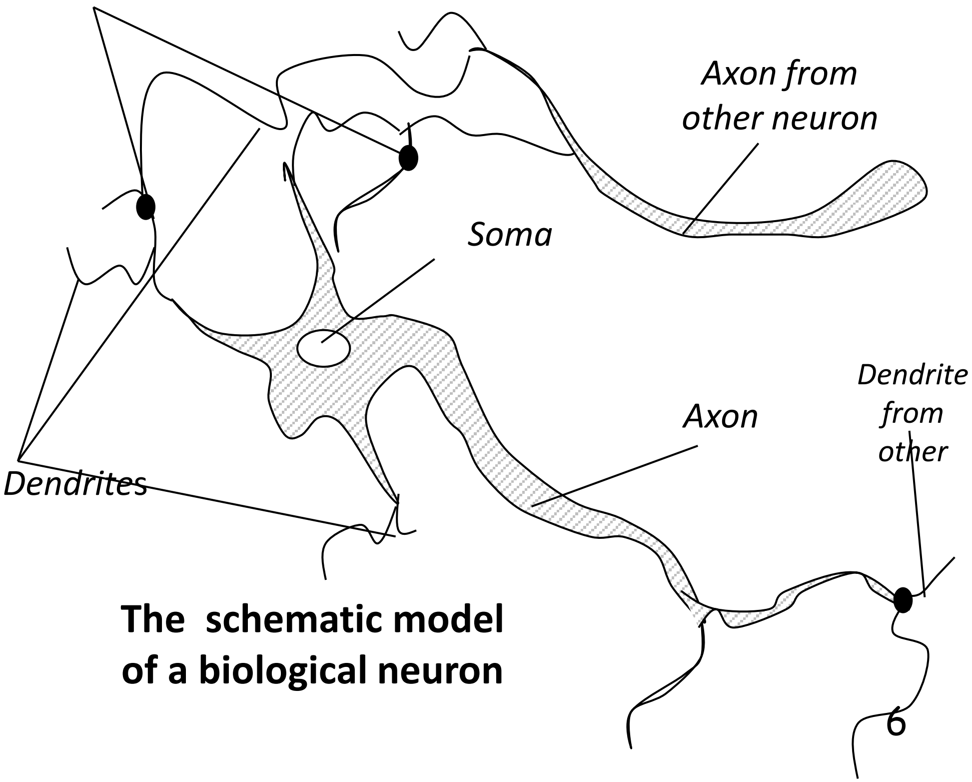
10^{11} neurons of >20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



Biological Neurons

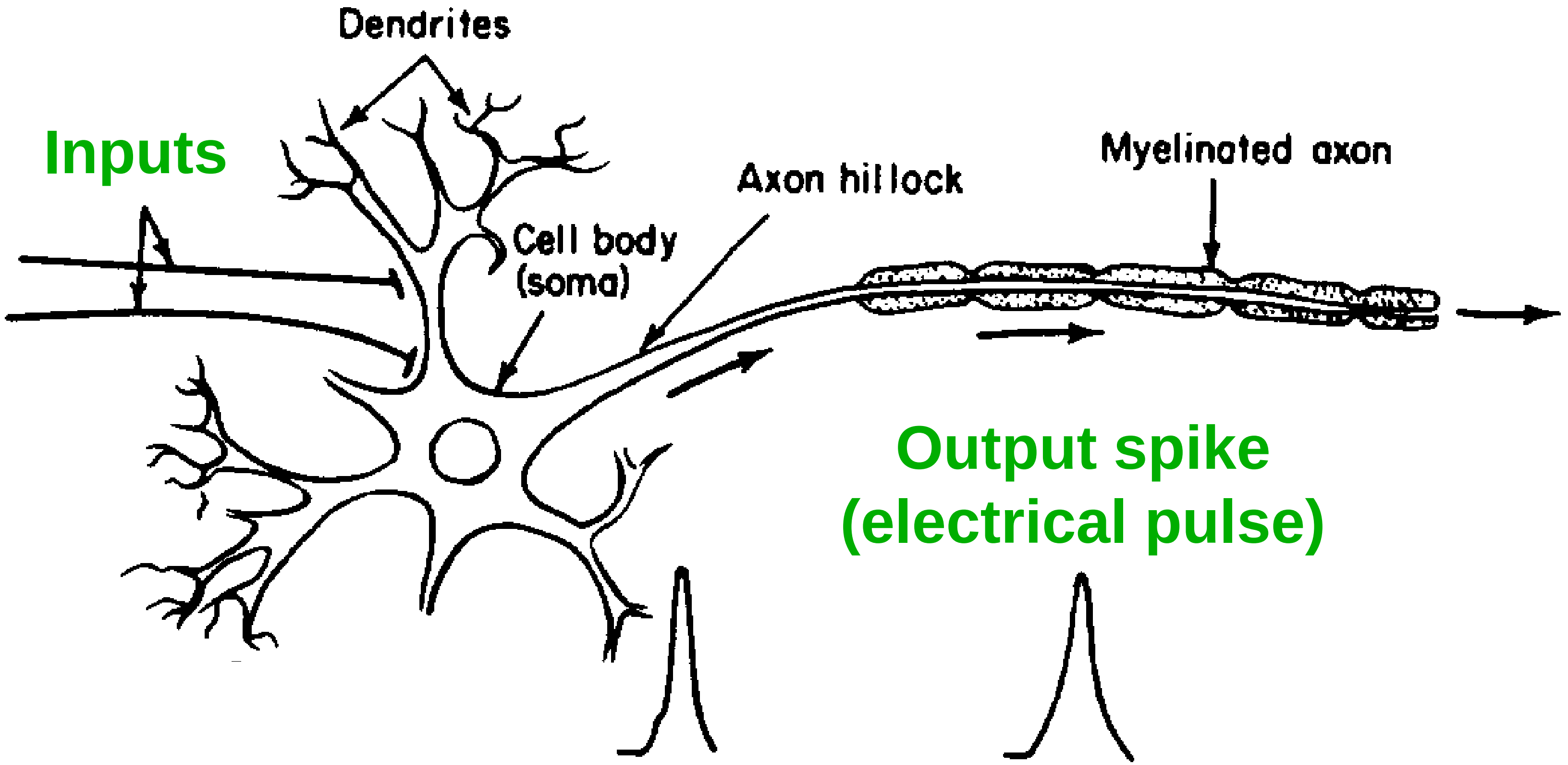
1. ***Soma* or *body cell* -** is a large, round central body in which almost all the logical functions of the neuron are realized.
2. ***The axon* (output)**, is a nerve fibre attached to the soma which can serve as a final output channel of the neuron. An axon is usually highly branched.
3. ***The dendrites* (inputs)-** represent a highly branching tree of fibres. These long irregularly shaped nerve fibres (processes) are attached to the soma.
4. ***Synapses*** are specialized contacts on a neuron which are the termination points for the axons from other neurons.





The schematic model of a biological neuron

Neurons communicate via spikes

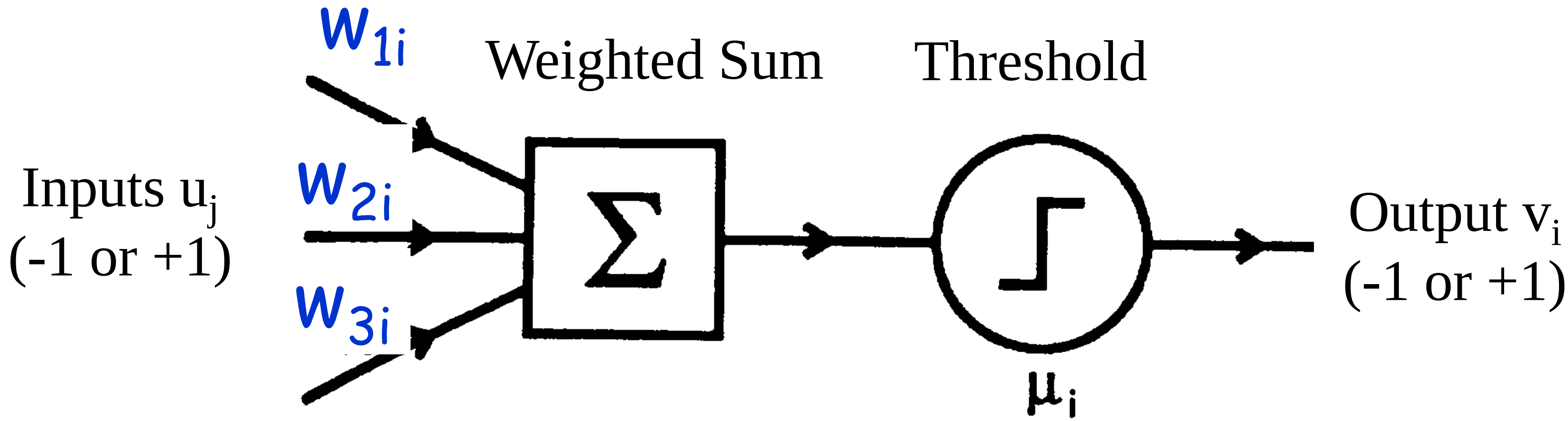


Output spike roughly dependent on whether sum of all inputs reaches a threshold

Neurons as "Threshold Units"

Artificial neuron:

- m binary inputs (-1 or 1), 1 output (-1 or 1)
- Synaptic weights w_{ji}
- Threshold μ_i



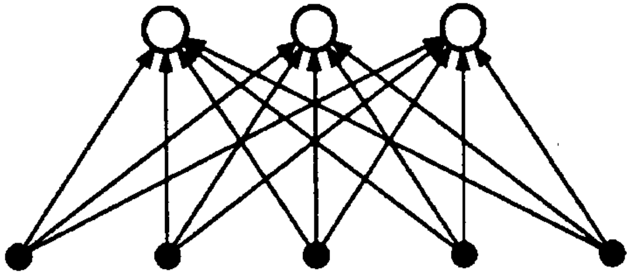
$$v_i = \Theta\left(\sum_j w_{ji} u_j - \mu_i\right)$$

$\Theta(x) = 1$ if $x > 0$ and -1 if $x \le 0$

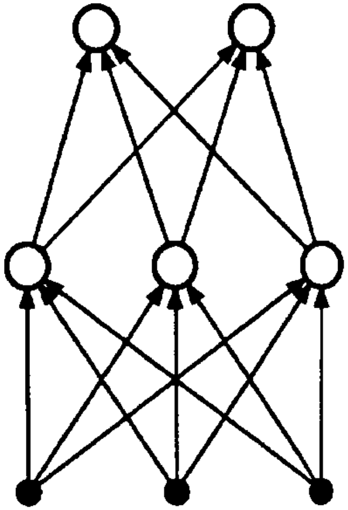
"Perceptrons" for Classification

- Fancy name for a type of layered "feed-forward" networks (no loops)
- Uses artificial neurons ("units") with binary inputs and outputs

Single-layer



Multilayer



Perceptrons and Classification

Consider a single-layer perceptron

- Weighted sum forms a *linear hyperplane*

$$\sum_j w_{ji} u_j - \mu_i = 0$$

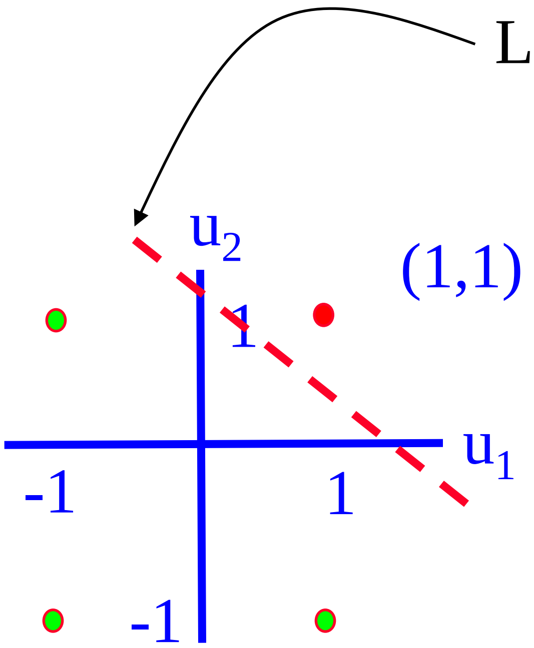
- Everything *on one side* of this hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)

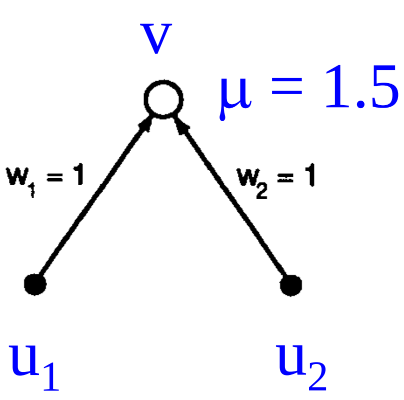
Any function that is <u>linearly separable</u> can be computed by a perceptron

Linear Separability

Example: AND is linearly separable

u_1	u_2	AND
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	1





$v = 1$ iff $u_1 + u_2 - 1.5 > 0$

Similarly for OR and NOT

How do we *learn* the appropriate weights given only examples of (input,output)?

Idea: Change the weights to decrease the error in output

Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

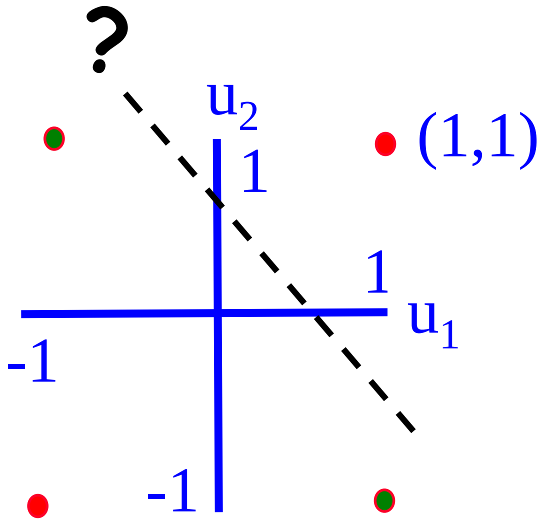
$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

What about the XOR function?

u_1	u_2	XOR
-1	-1	1
1	-1	-1
-1	1	-1
1	1	1



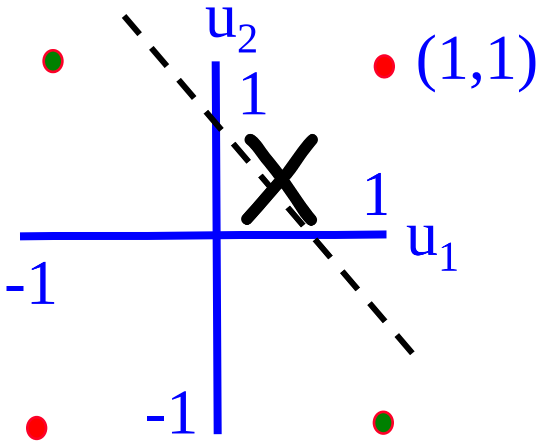
Can a perceptron separate the +1 outputs from the -1 outputs?

Linear Inseparability

Perceptron with threshold units fails if classification task is not linearly separable

- Example: XOR
- No single line can separate the "yes" (+1) outputs from the "no" (−1) outputs!

Minsky and Papert's book showing such negative results put a damper on neural networks research for over a decade!



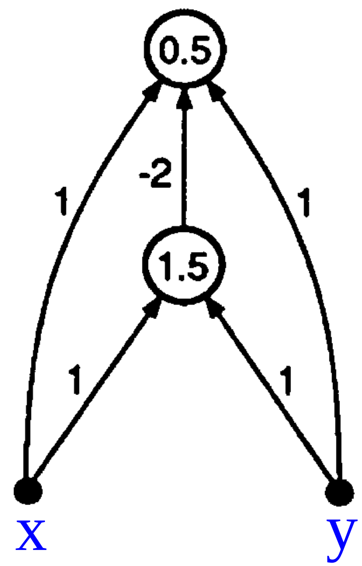
How do we deal with linear inseparability?

Idea 1: Multilayer Perceptrons

Removes limitations of single-layer networks

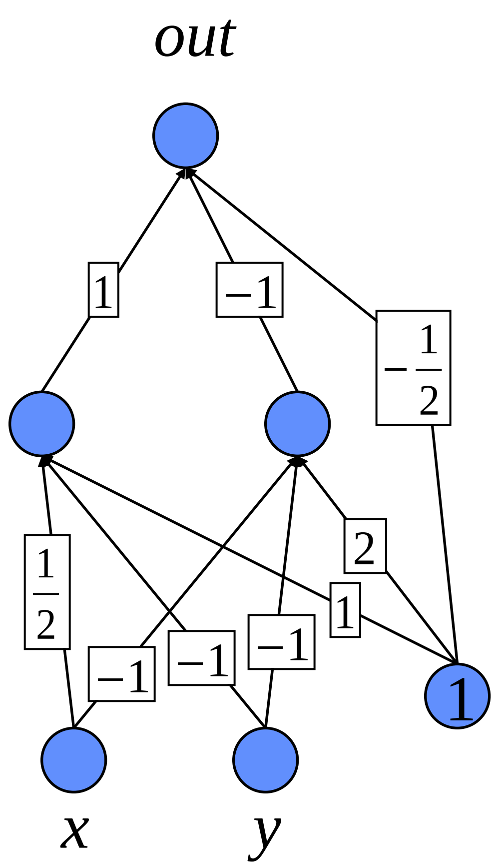
- Can solve XOR

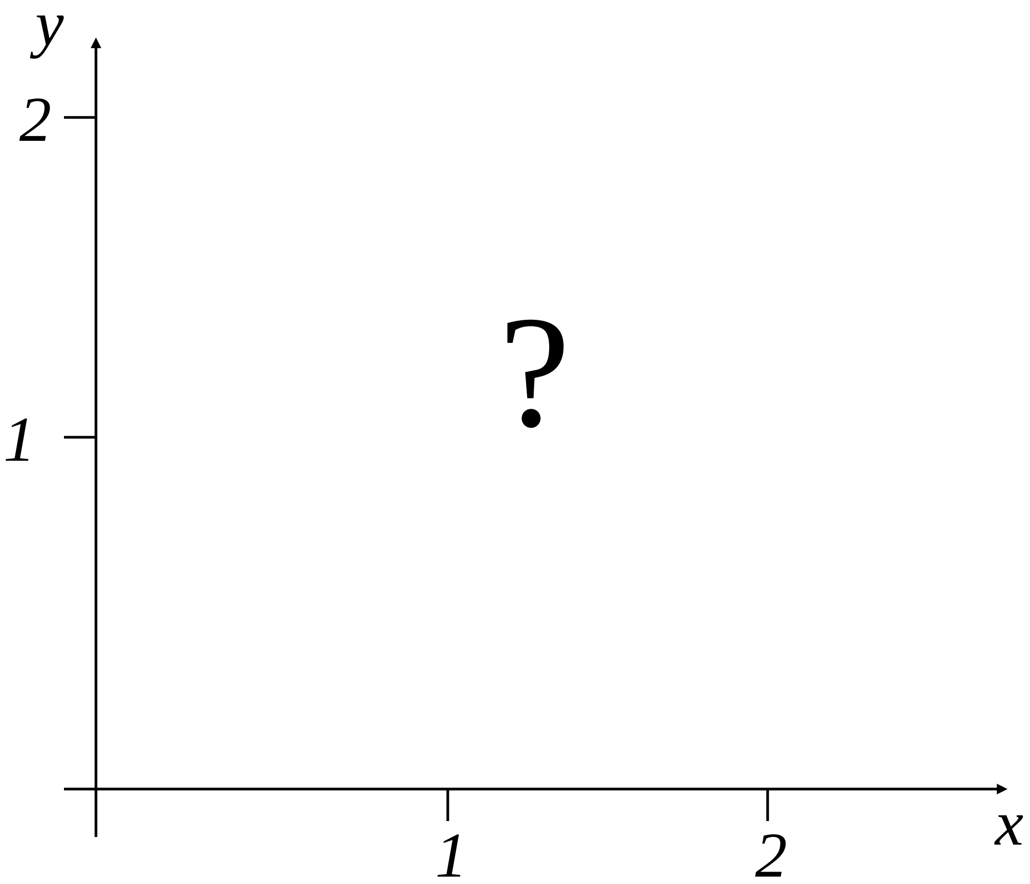
Example: Two-layer perceptron that computes XOR



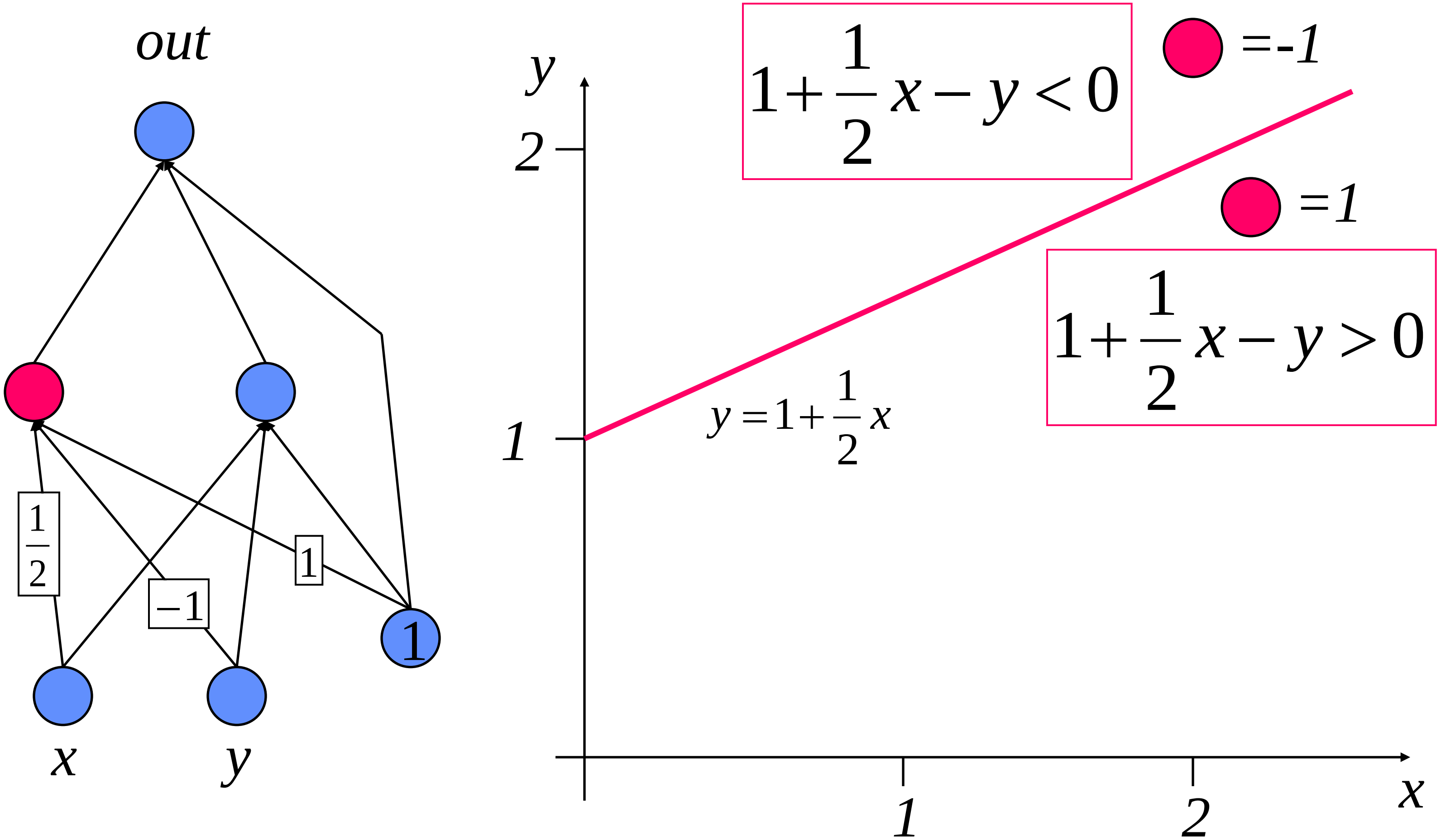
Output is +1 if and only if $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

Multilayer Perceptron: What does it do?

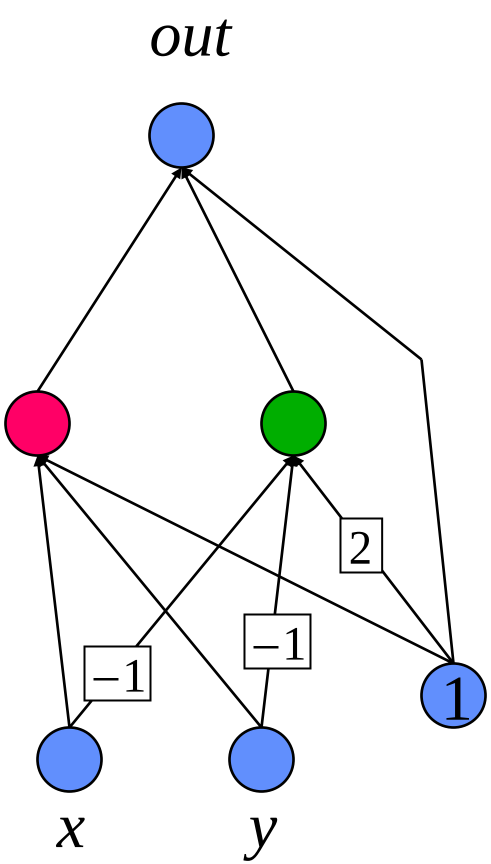


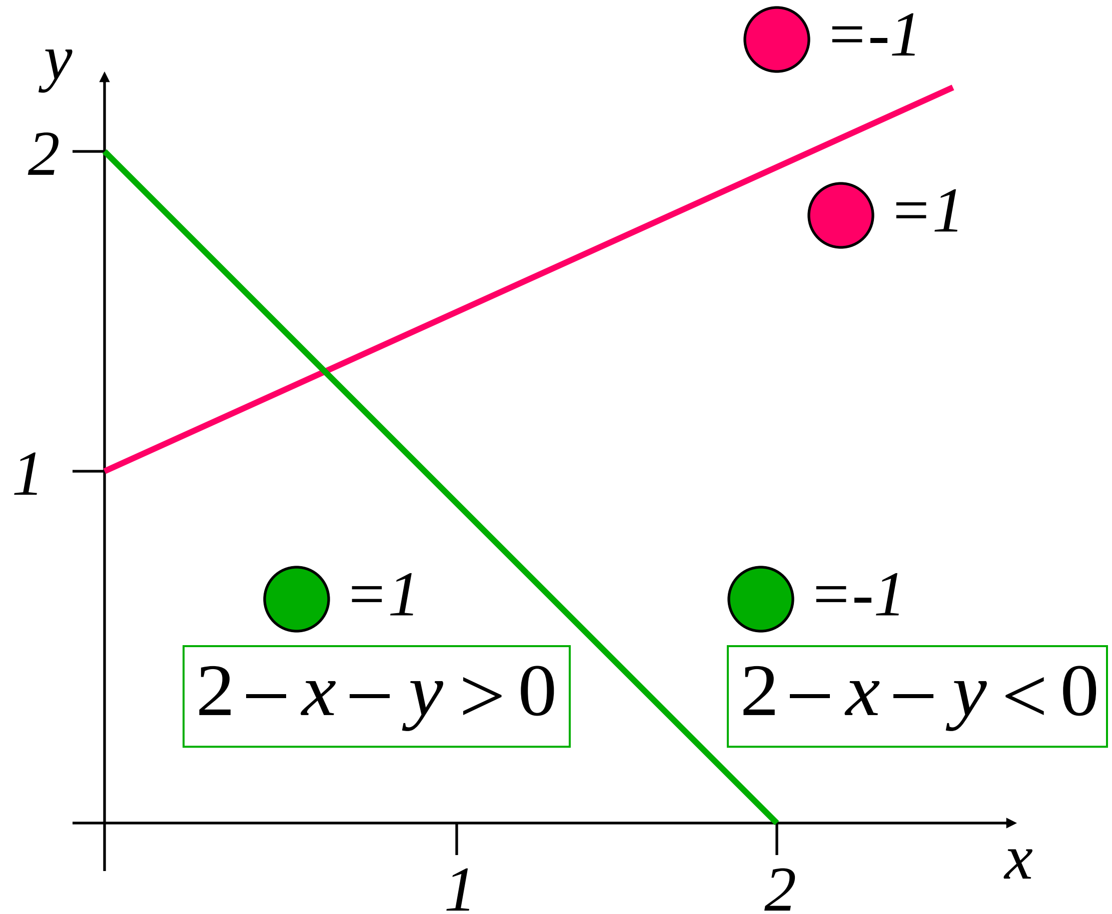


Multilayer Perceptron: What does it do?

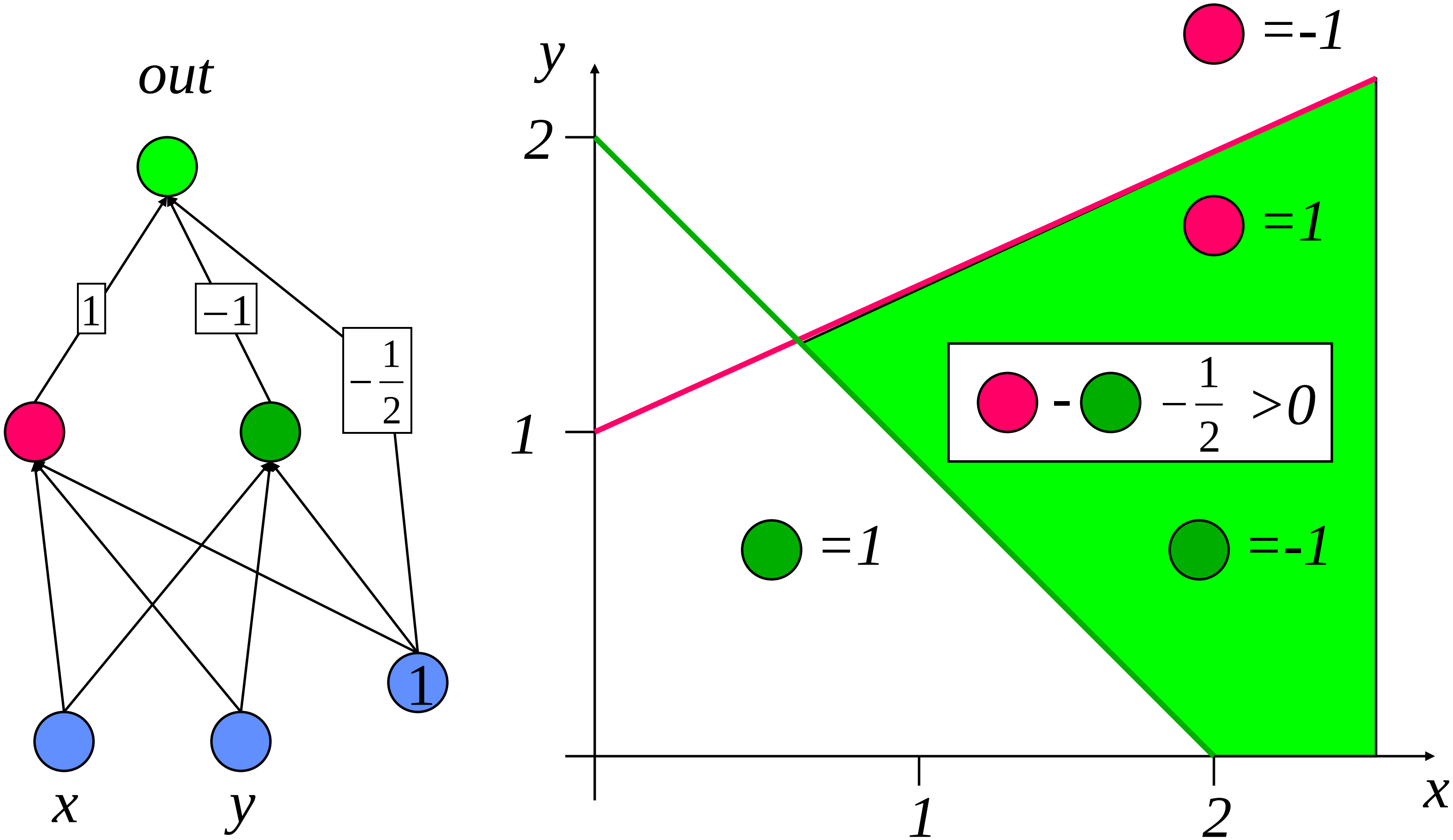


Multilayer Perceptron: What does it do?

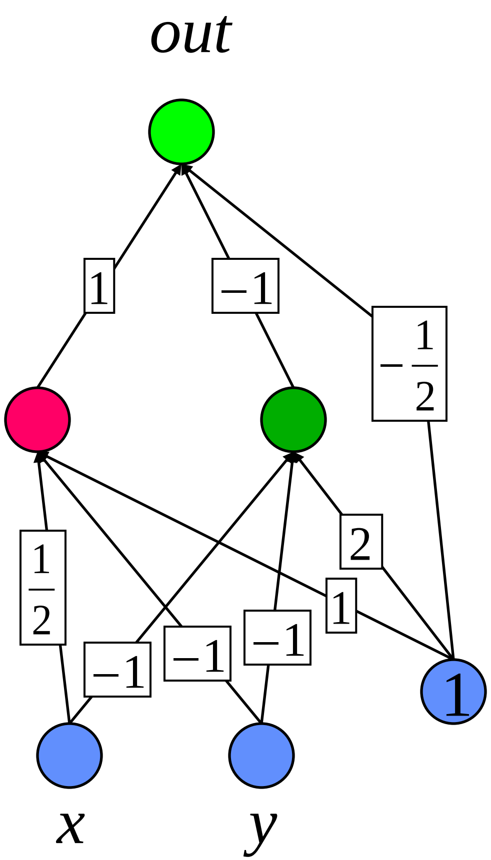


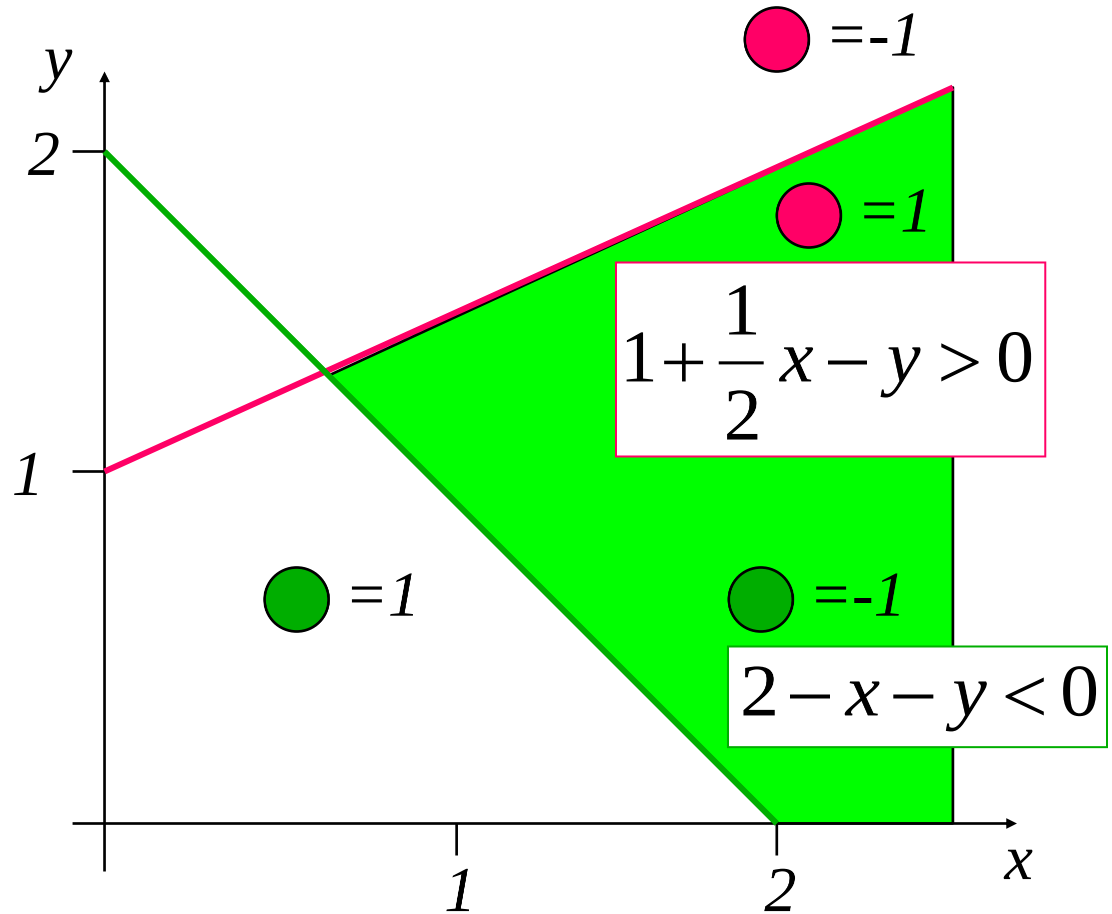


Multilayer Perceptron: What does it do?

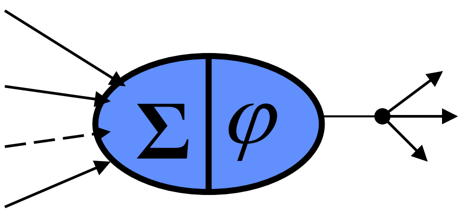


Perceptrons as Constraint Satisfaction Networks



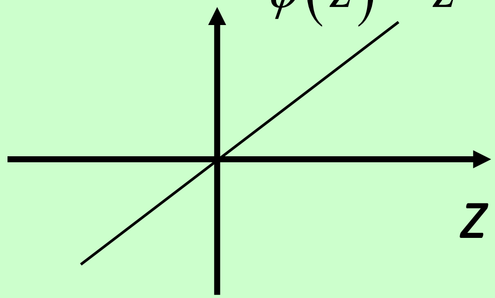


Artificial Neuron: Most Popular Activation Functions



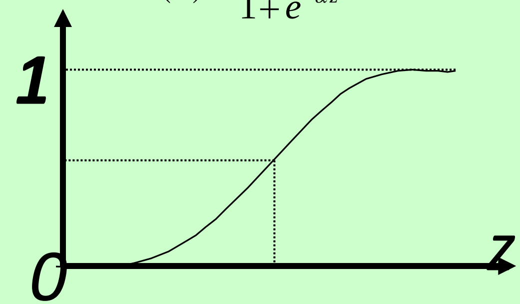
Linear activation

$$\phi(z) = z$$



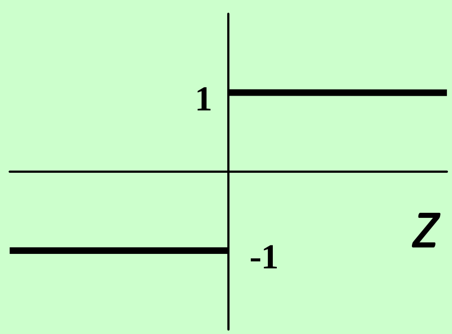
Logistic activation

$$\phi(z) = \frac{1}{1+e^{-\alpha z}}$$



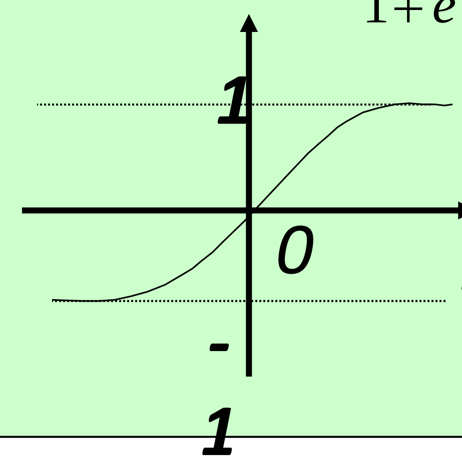
Threshold activation

$$\phi(z) = \text{sign}(z) = \begin{cases} 1, & if \quad z \geq 0, \\ -1, & if \quad z < 0. \end{cases}$$



Hyperbolic tangent activation

$$\varphi(u) = tanh(\gamma u) = \frac{1-e^{-2\gamma u}}{1+e^{-2\gamma u}}$$



Neural Network Issues

- Multi-layer perceptrons can represent any function
- Training multi-layer perceptrons hard
    - Backpropagation

- Early successes
    - Keeping the car on the road

- Difficult to debug
    - Opaque

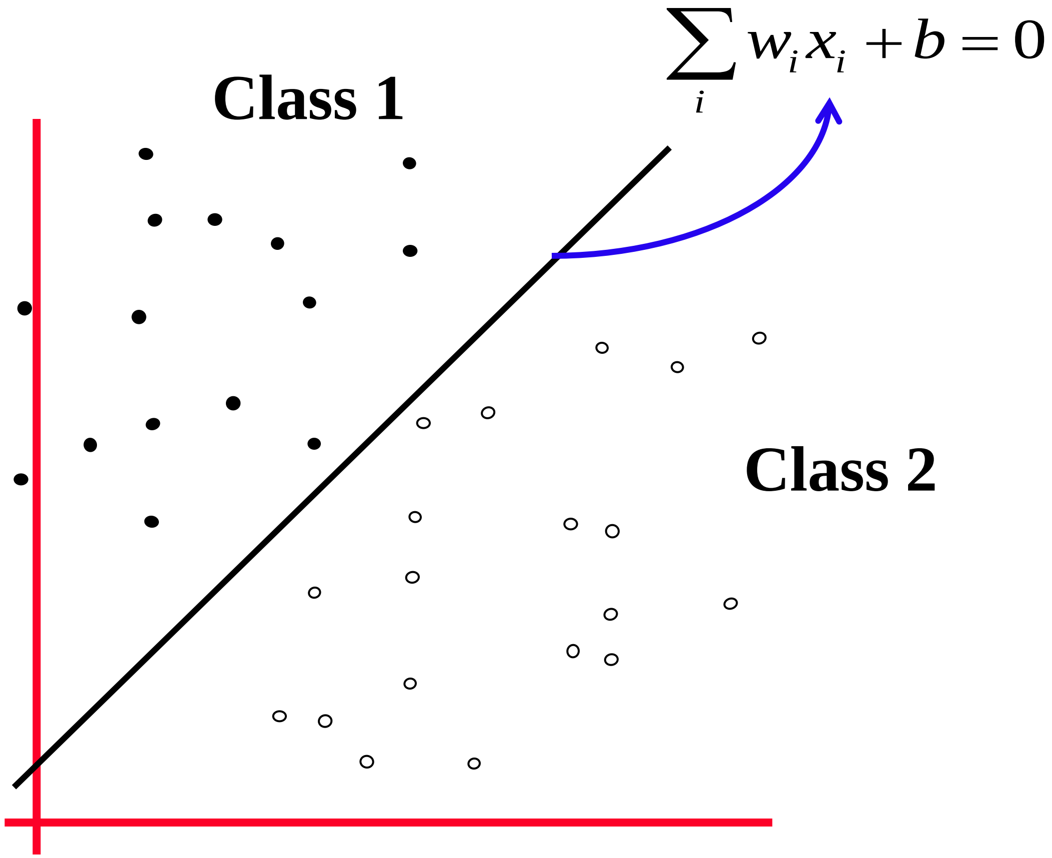
Back to Linear Separability

- **Recall: Weighted sum in perceptron forms a *linear hyperplane***

$$\sum_i w_i x_i + b = 0$$

- Due to threshold function, everything *on one side* of this hyperplane is labeled as class 1 (output = +1) and everything *on other side* is labeled as class 2 (output = -1)

Separating Hyperplane

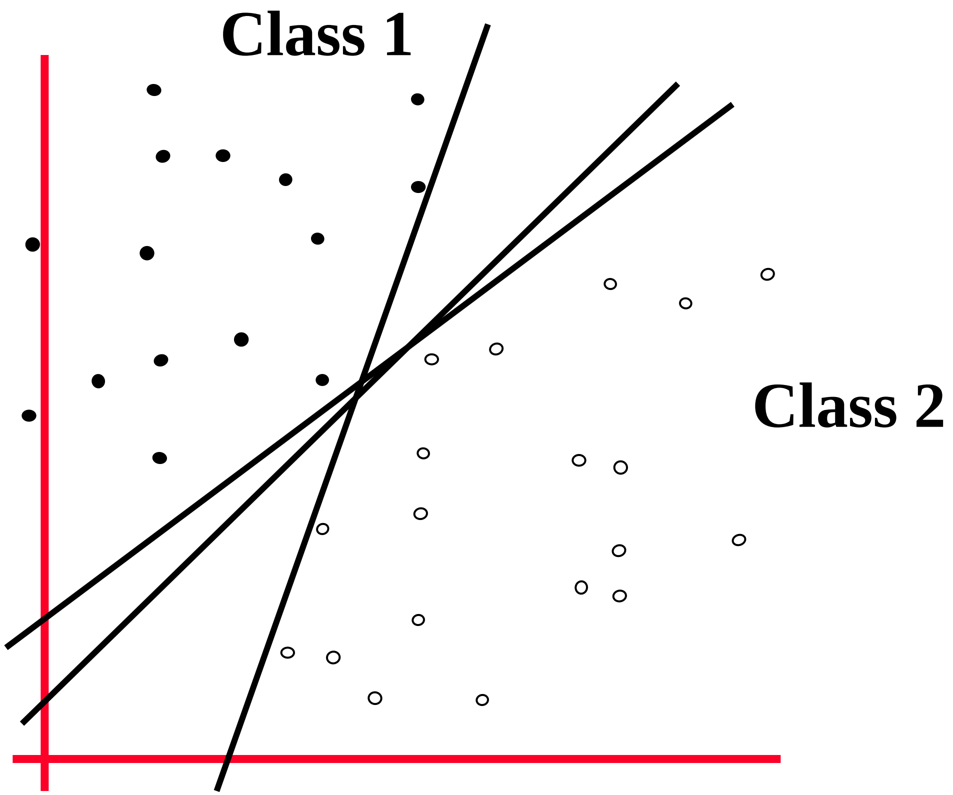


- • denotes +1 output
- ◦ denotes -1 output

Need to choose **w** and *b* based on training data

Separating Hyperplanes

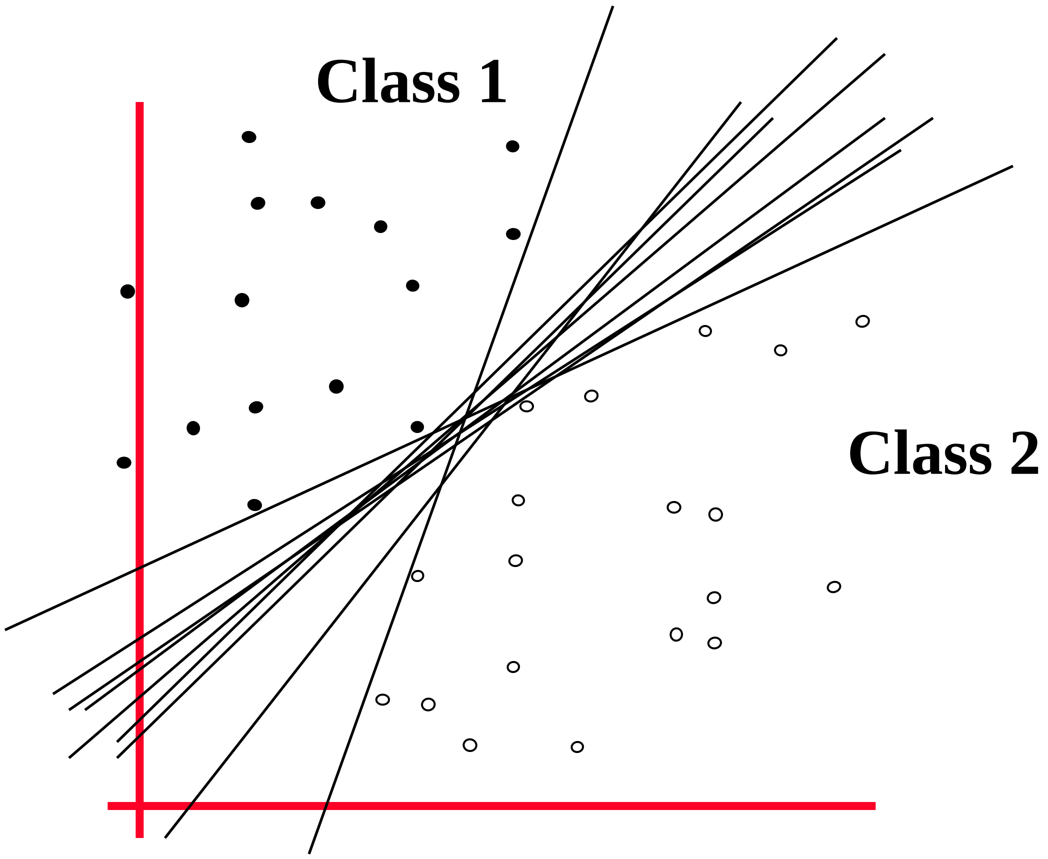
Different choices of **w** and *b* give different hyperplanes



- • denotes +1 output
- ◦ denotes -1 output

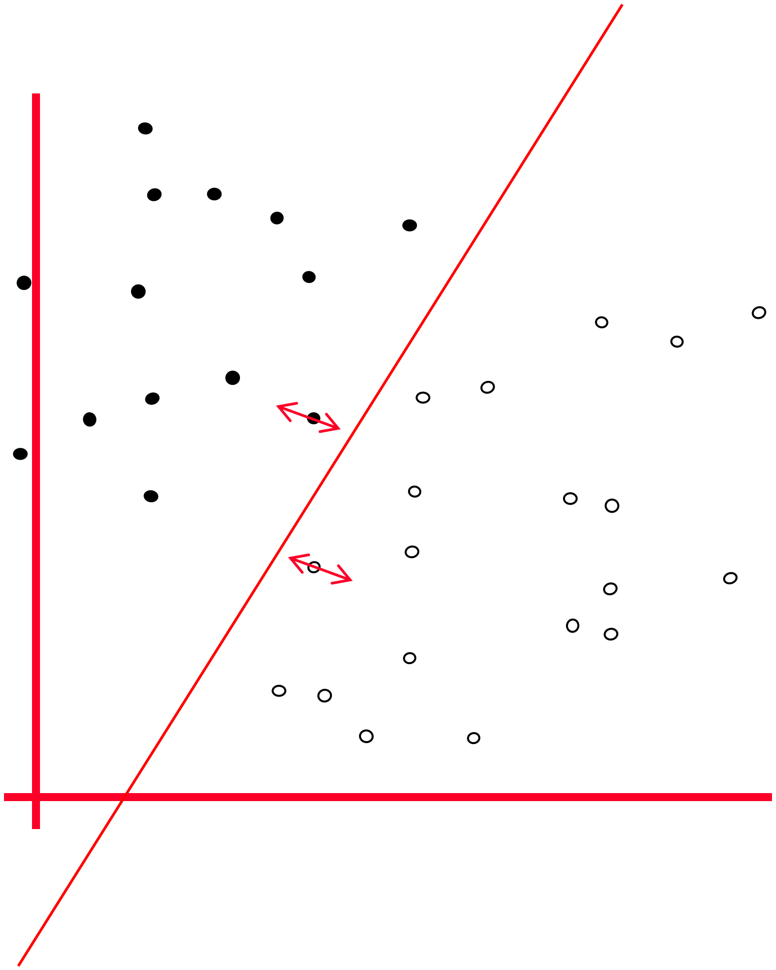
(This and next few slides adapted from Andrew Moore's)

Which hyperplane is best?



- • denotes +1 output
- ◦ denotes -1 output

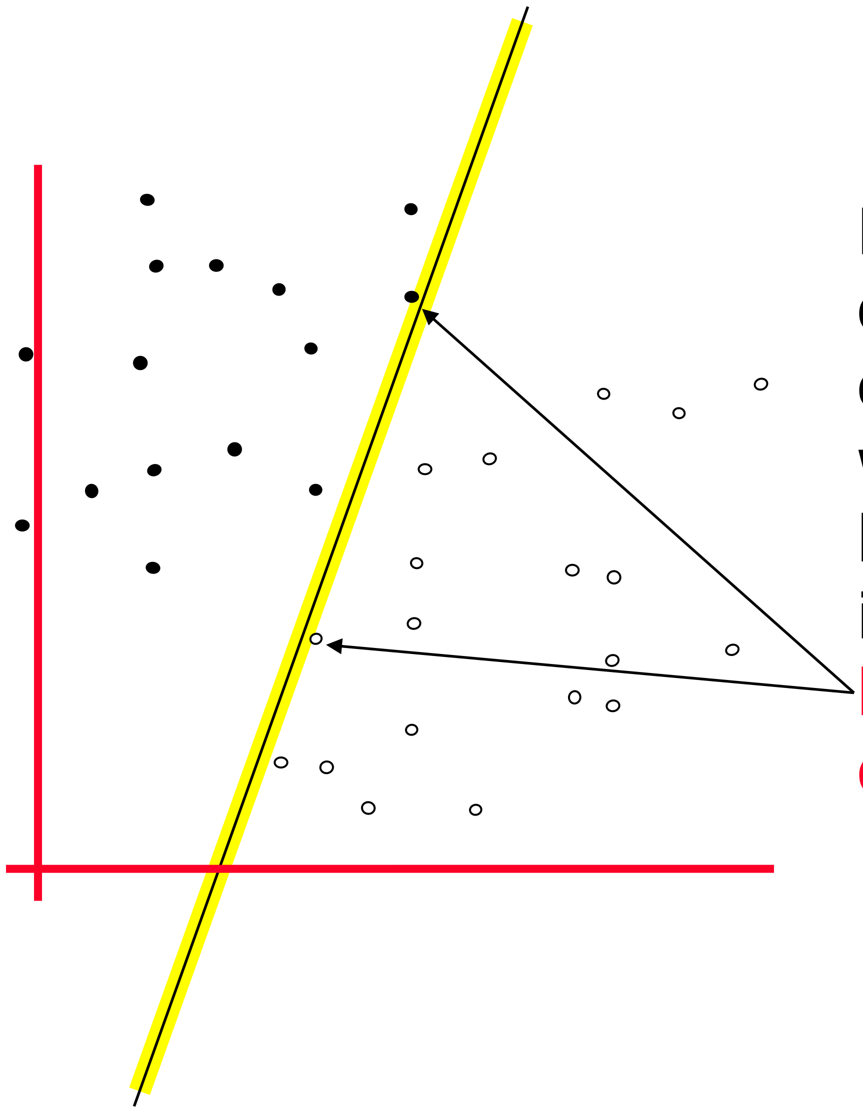
How about the one right in the middle?



Intuitively, this boundary seems good

Avoids misclassification of new test points if they are generated from the same distribution as training points

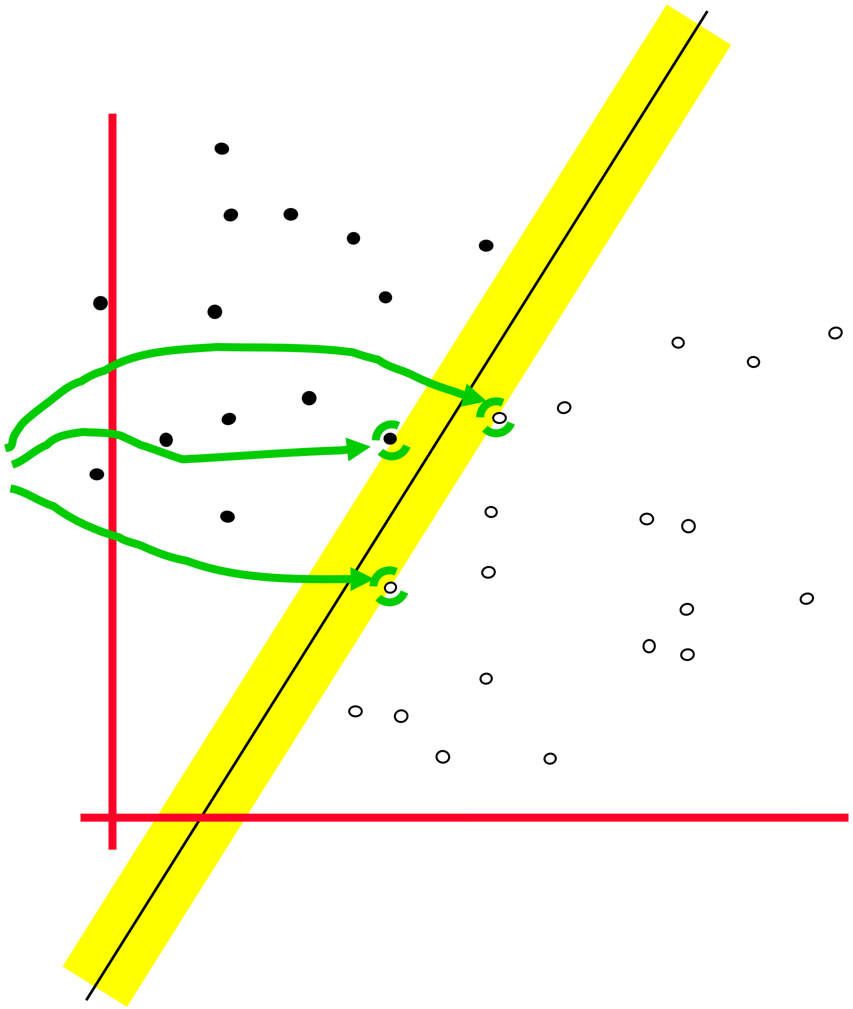
Margin



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

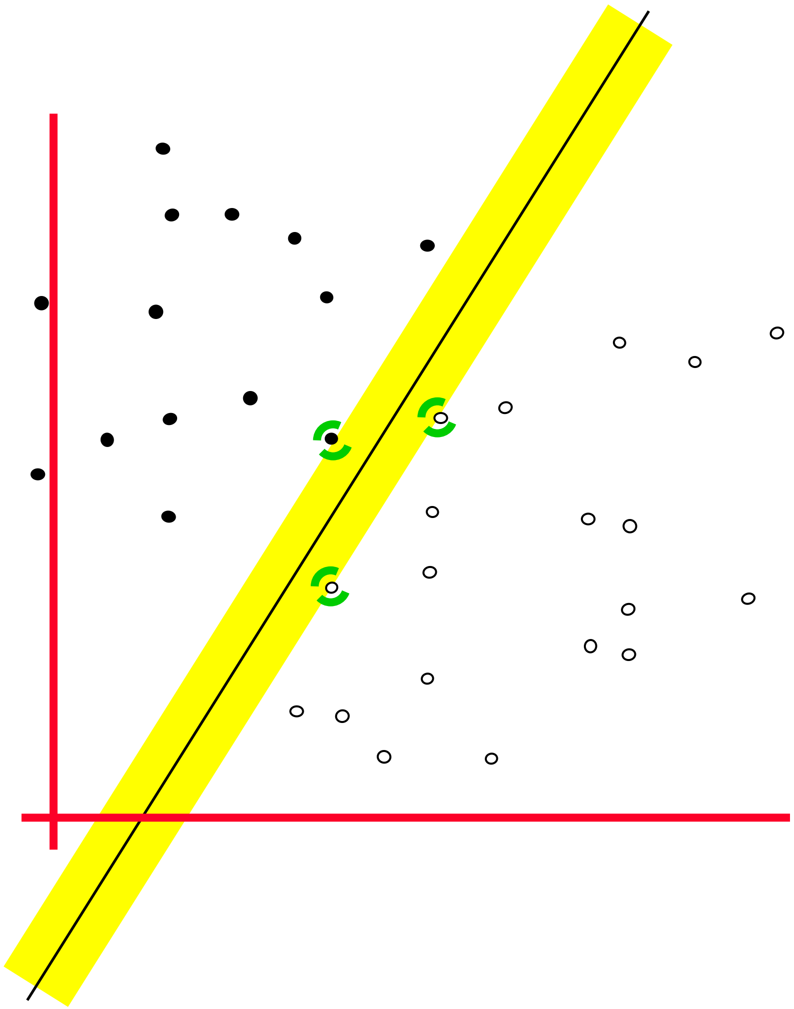
Maximum Margin and Support Vector Machine

Support Vectors are those datapoints that the margin pushes up against



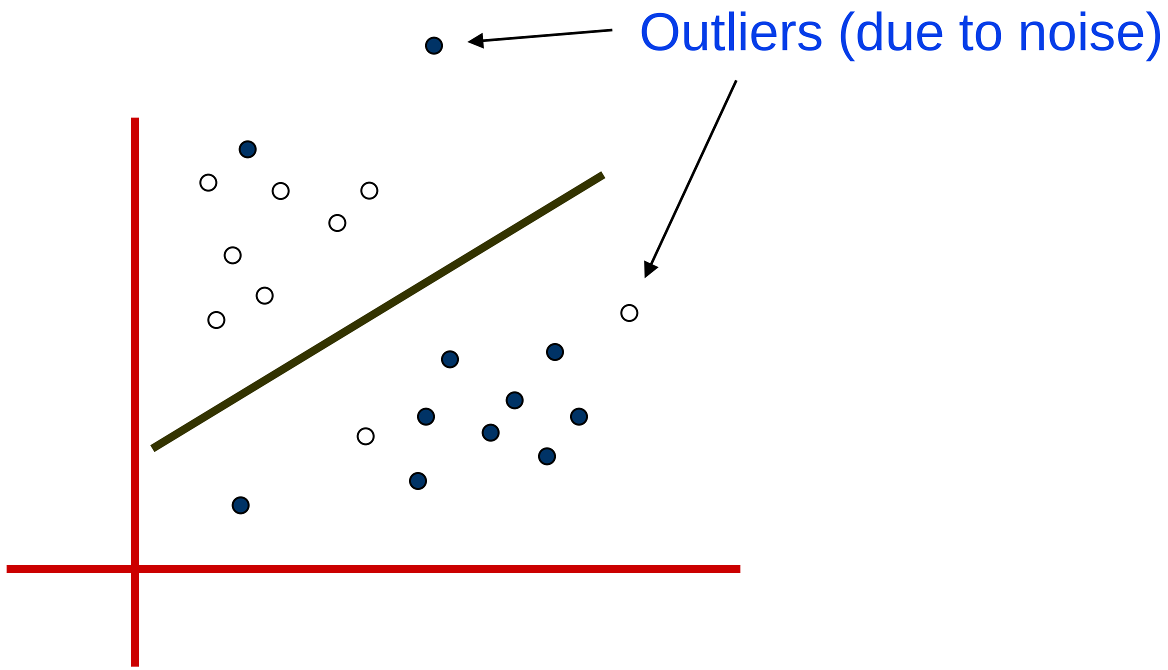
The maximum margin classifier is called a Support Vector Machine (in this case, a Linear SVM or LSVM)

Why Maximum Margin?

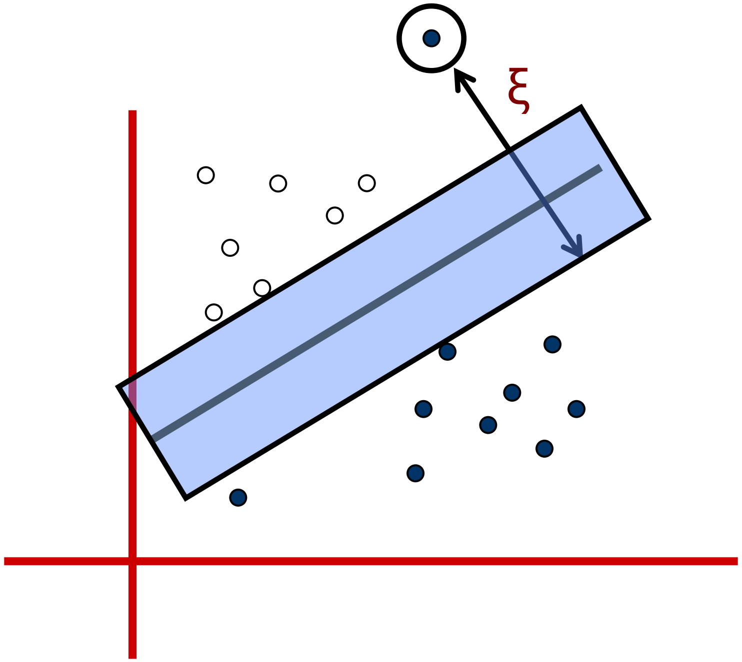


- Robust to small perturbations of data points near boundary
- There exists theory showing this is best for generalization to new points
- Empirically works great

What if data is not linearly separable?



Approach 1: Soft Margin SVMs

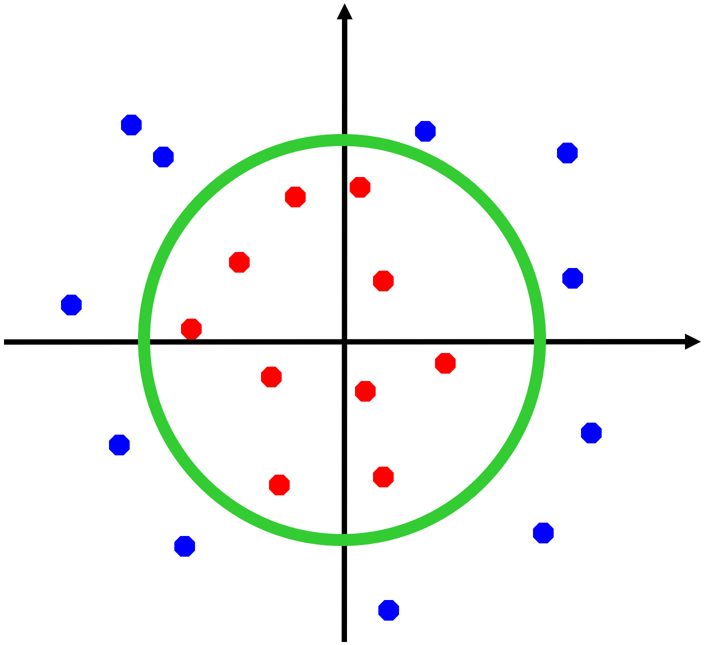


Allow *errors* ξ_i (deviations from margin)

Trade off margin with errors.

Minimize: margin + error-penalty

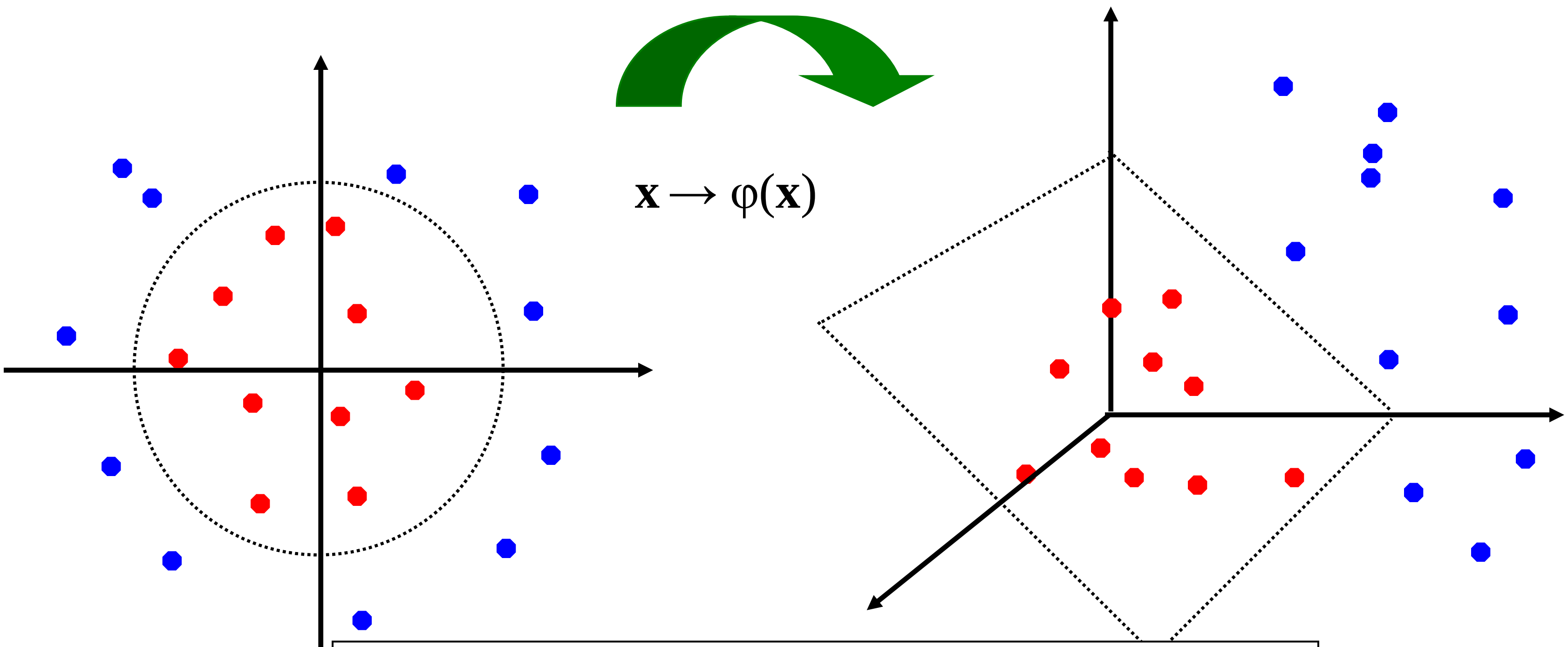
What if data is not linearly separable: Other ideas?



Not linearly separable

What if data is not linearly separable?

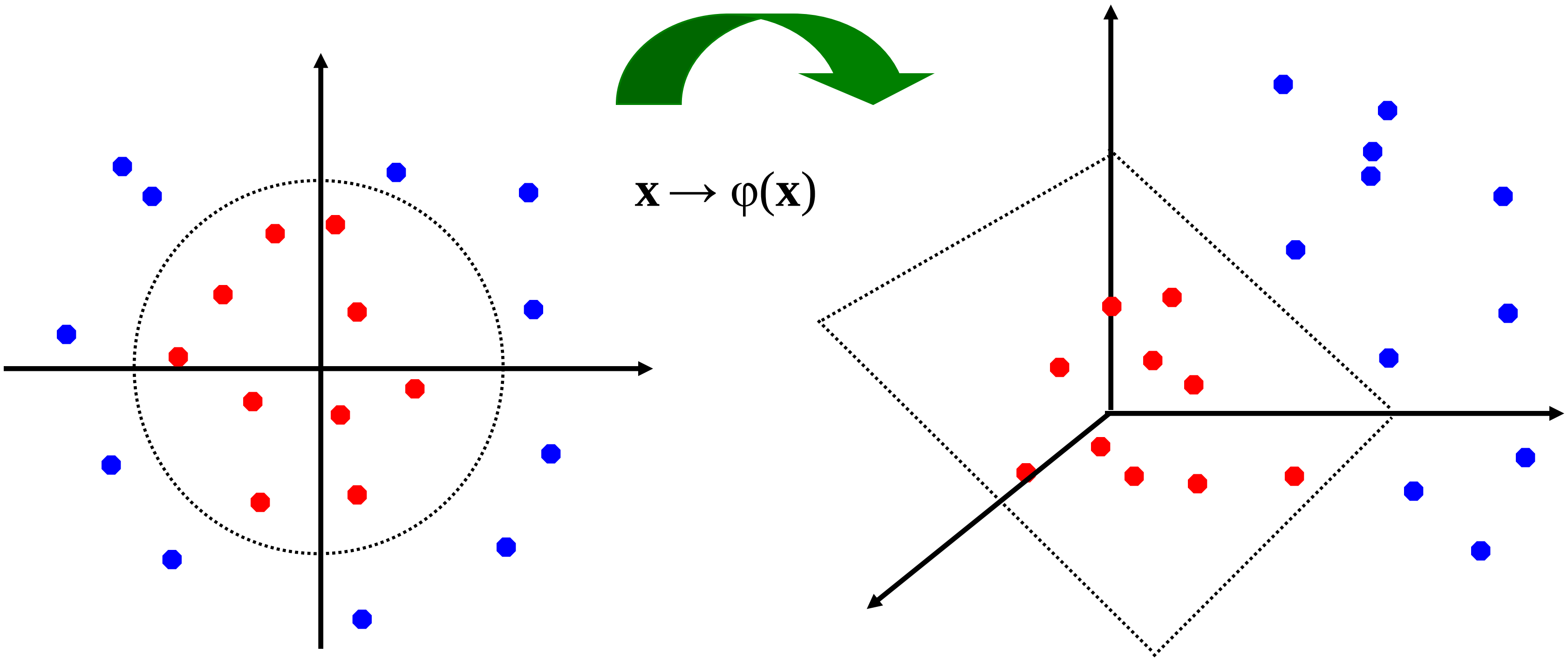
Approach 2: Map original input space to higher-dimensional feature space; use linear classifier in higher-dim. space



$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

Kernel: additional bias to convert into high d space

Problem with high dimensional spaces



- Computation in high-dimensional feature space can be costly
- The high dimensional projection function $\varphi(x)$ may be too complicated to compute
- Kernel trick to the rescue!

The Kernel Trick

Dual Formulation: SVM maximizes the quadratic function:

$$\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to } \alpha_i \geq 0 \text{ and } \sum_i \alpha_i y_i = 0$$

Insight:

The data points only appear as inner product

- No need to compute high-dimensional $\varphi(x)$ explicitly! Just replace inner product $x_i \cdot x_j$ with a kernel function $K(x_i,x_j) = \varphi(x_i) \cdot \varphi(x_j)$
- E.g., Gaussian kernel

  $K(x_i,x_j) = \exp(-||x_i-x_j||^2/2\sigma^2)$
- E.g., Polynomial kernel

  $K(x_i,x_j) = (x_i \cdot x_j+1)^d$

K-Nearest Neighbors

A simple *non-parametric* classification algorithm

Idea:

- Look around you to see how your neighbors classify data
- Classify a new data-point according to a *majority vote* of your k nearest neighbors

Distance Metric

How do we measure what it means to be a neighbor (what is "close")?

Appropriate distance metric depends on the problem

Examples:

x discrete (e.g., strings): Hamming distance

$d(x_1,x_2)$ = # features on which x_1 and x_2 differ

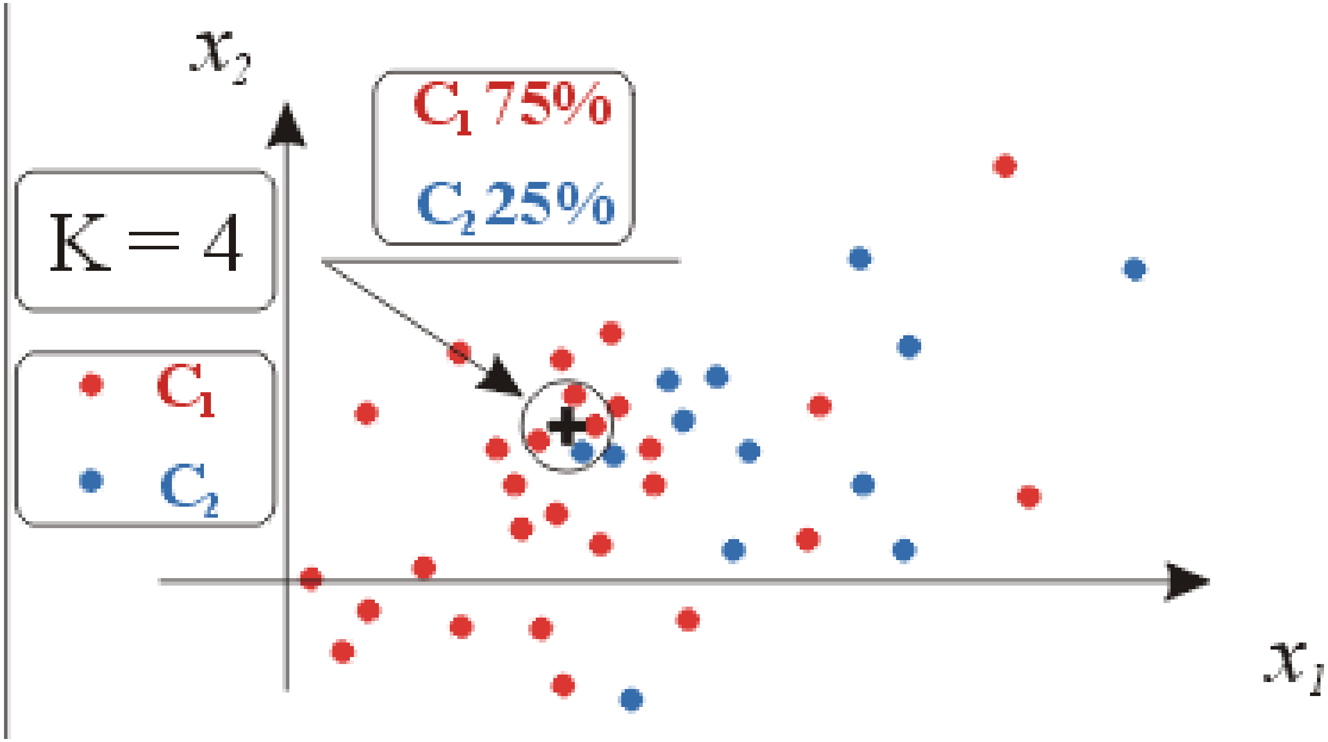
x continuous (e.g., vectors over reals): Euclidean distance

$d(x_1,x_2) = || x_1 - x_2 ||$ = square root of sum of squared differences between corresponding elements of data vectors

Example

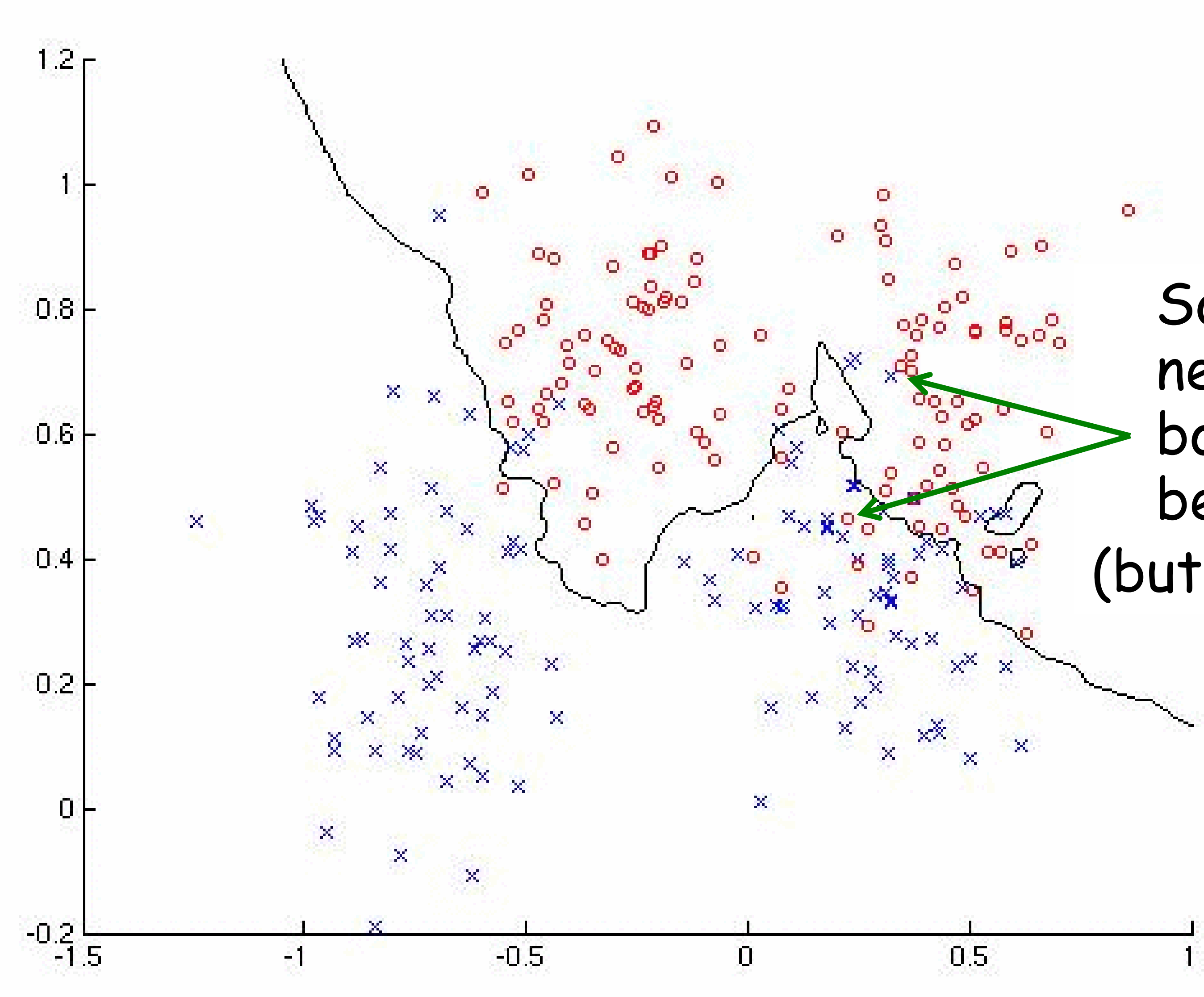
Input Data: 2-D points (x_1,x_2)

Two classes: C_1 and C_2. New Data Point ✚



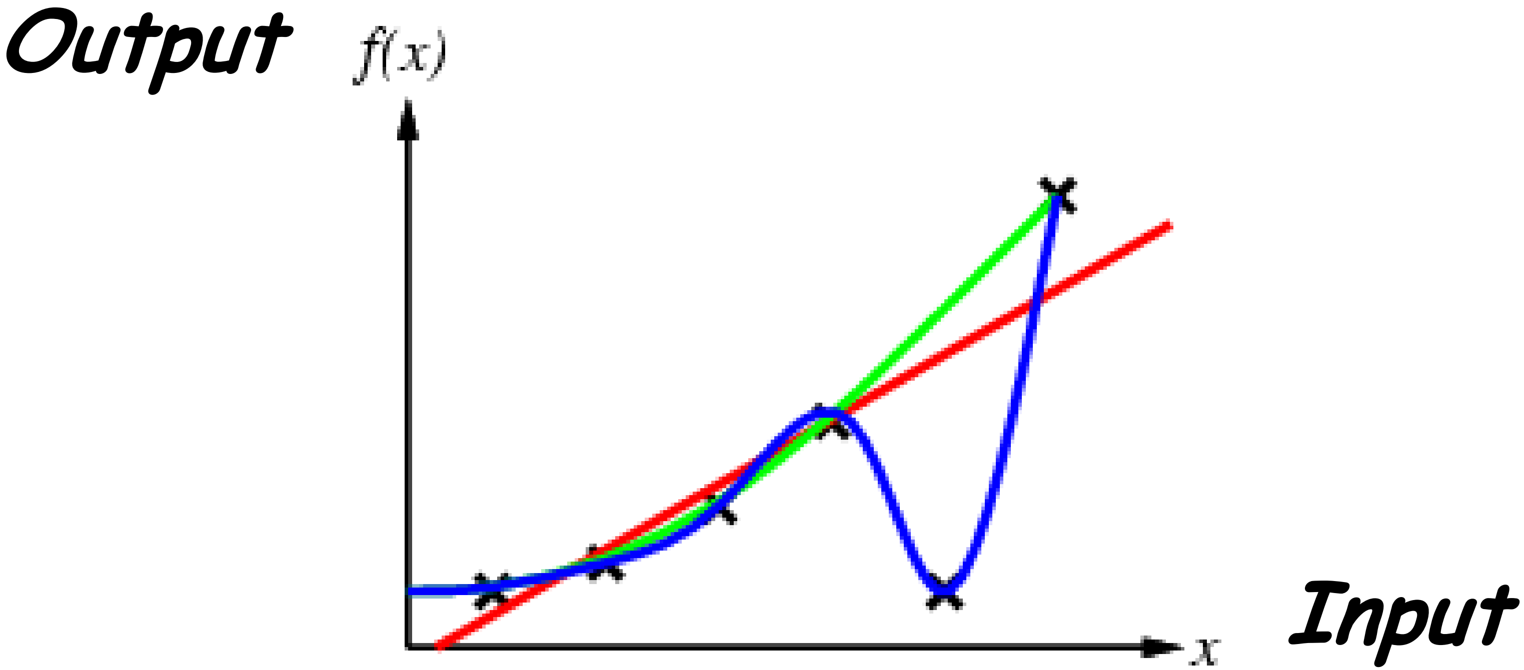
K = 4: Look at 4 nearest neighbors of +
3 are in C_1, so classify + as C_1

Decision Boundary using K-NN



Some points near the boundary may be misclassified (but maybe noise)

What if we want to learn continuous-valued functions?



Regression

K-Nearest neighbor

take the average of k-close by points

Linear/Non-linear Regression

fit parameters (gradient descent)
minimizing the regression error/loss

Neural Networks

remove the threshold function

Large Feature Spaces

Easy to overfit

Regularization

add penalty for large weights

prefer weights that are zero or close to zero

minimize

regression error + C.regularization penalty

Regularizations

L1 : diamond

L2 : circle

Derivatives

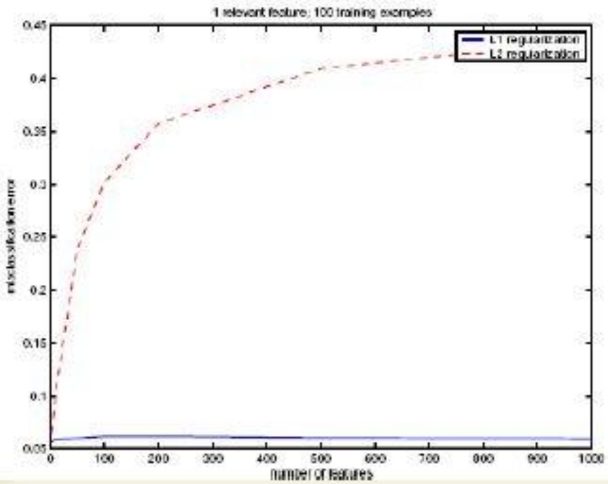
L1 : constant

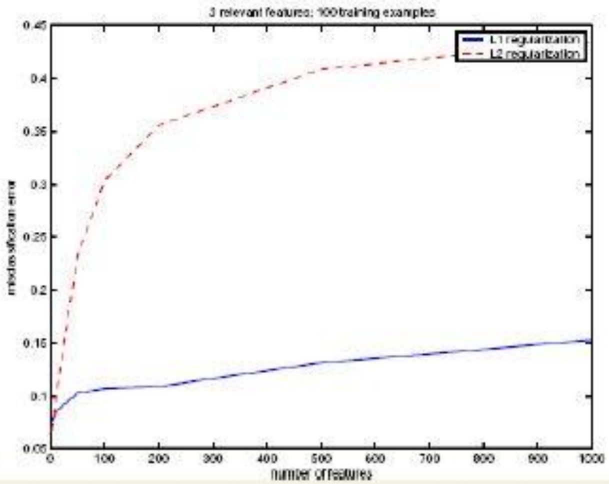
L2 : high for large weights

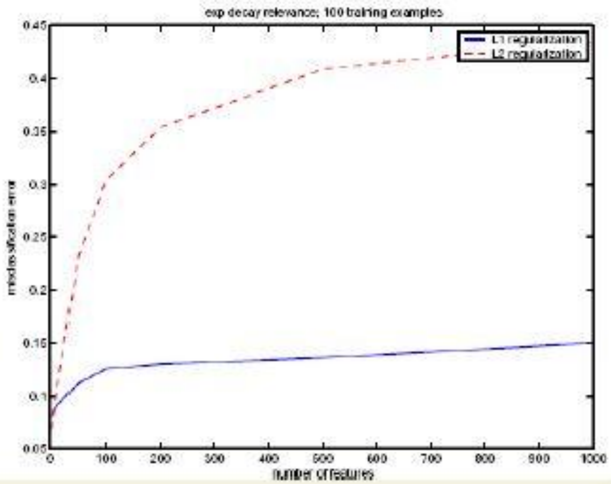
L1 harder to optimize, but not too hard.

- discontinuous but convex

L1 vs. L2







1 relevant feature, 100 training examples

3 relevant feature, 100 training examples

exp decay in relevance, 100 training examples