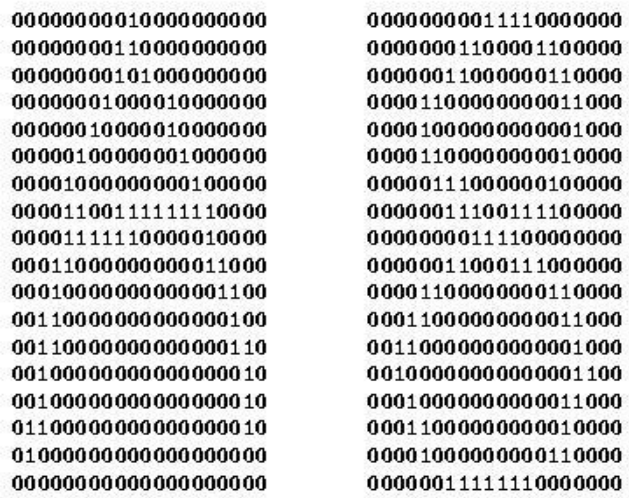
# Supervised Learning (contd)
# Linear Separation

Mausam
(based on slides by UW-AI faculty)

# Images as Vectors

Binary handwritten characters



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)

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

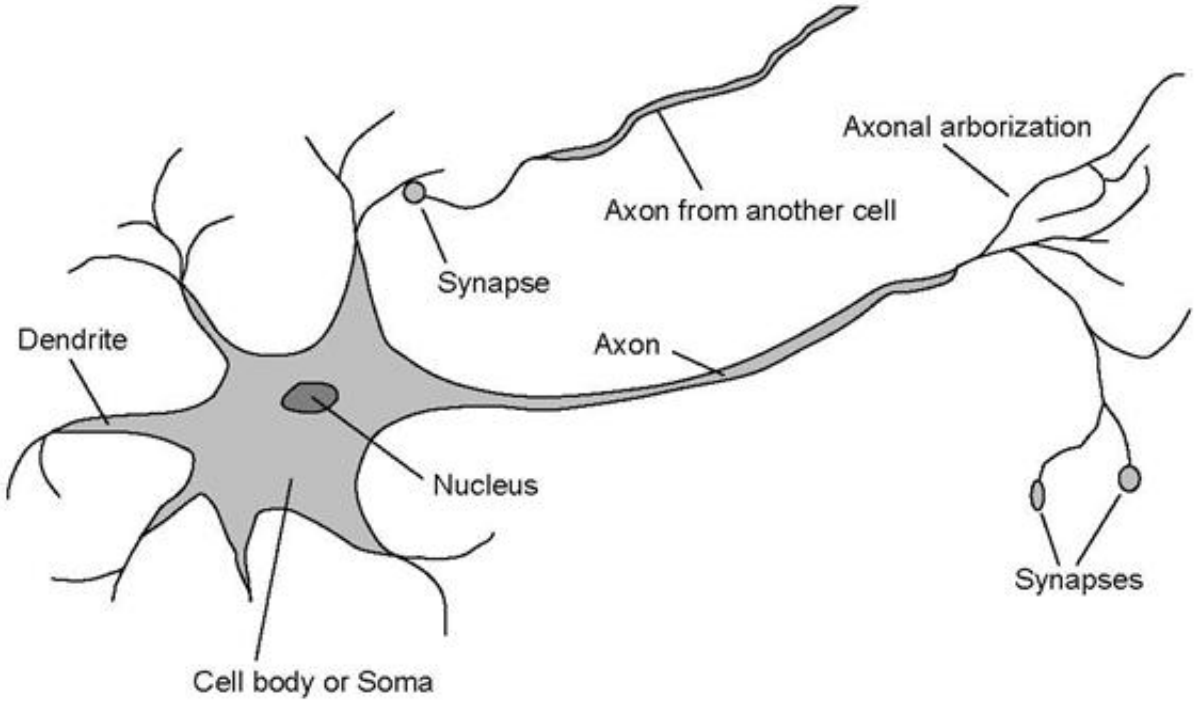
Pixel value $p_i$ can be 0 or 1 (binary image) or 0 to 255 (greyscale)

# The human brain is extremely good at classifying images

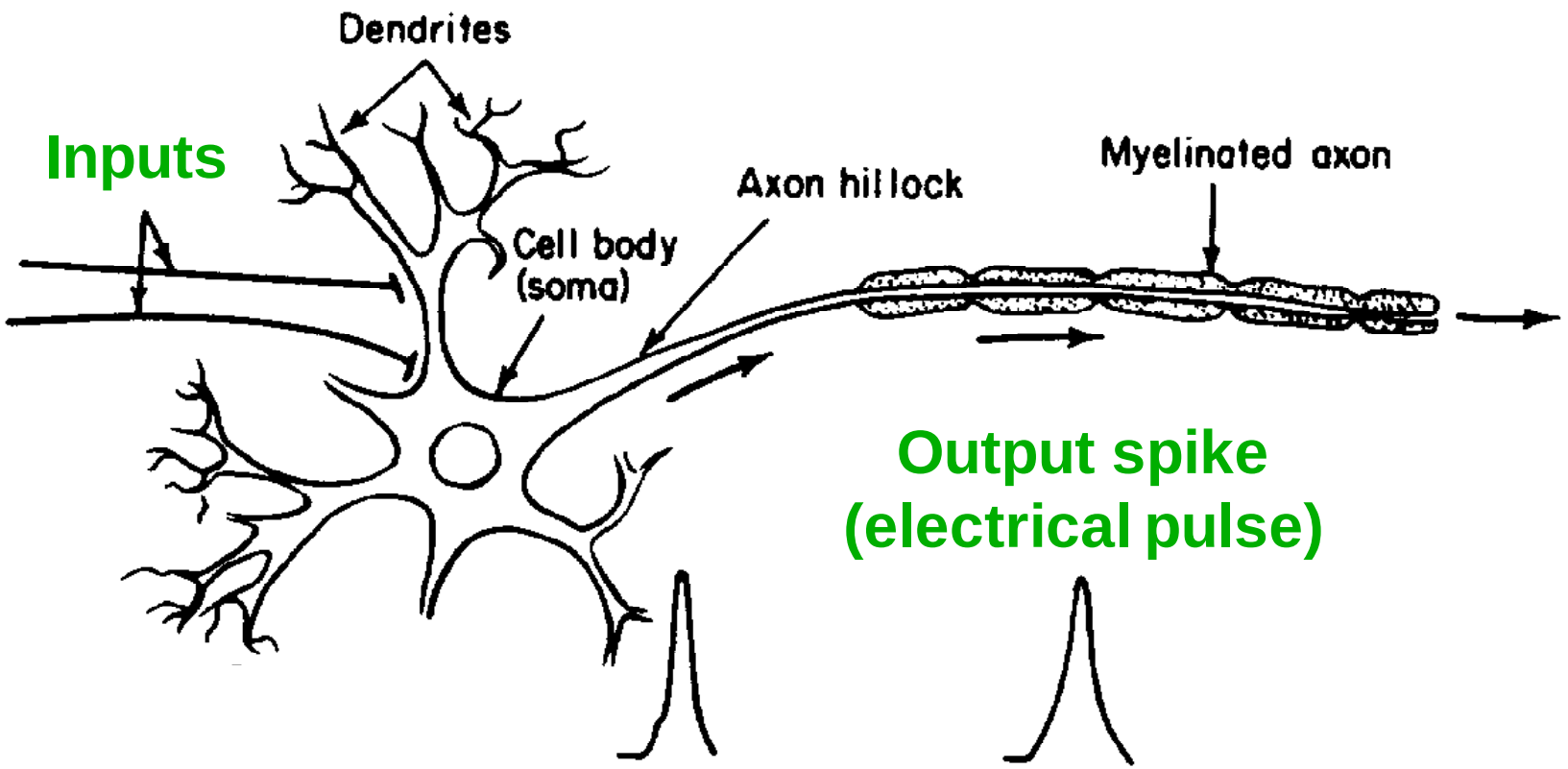
Can we develop classification methods by emulating the brain?

# Brains

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



# Neurons communicate via spikes

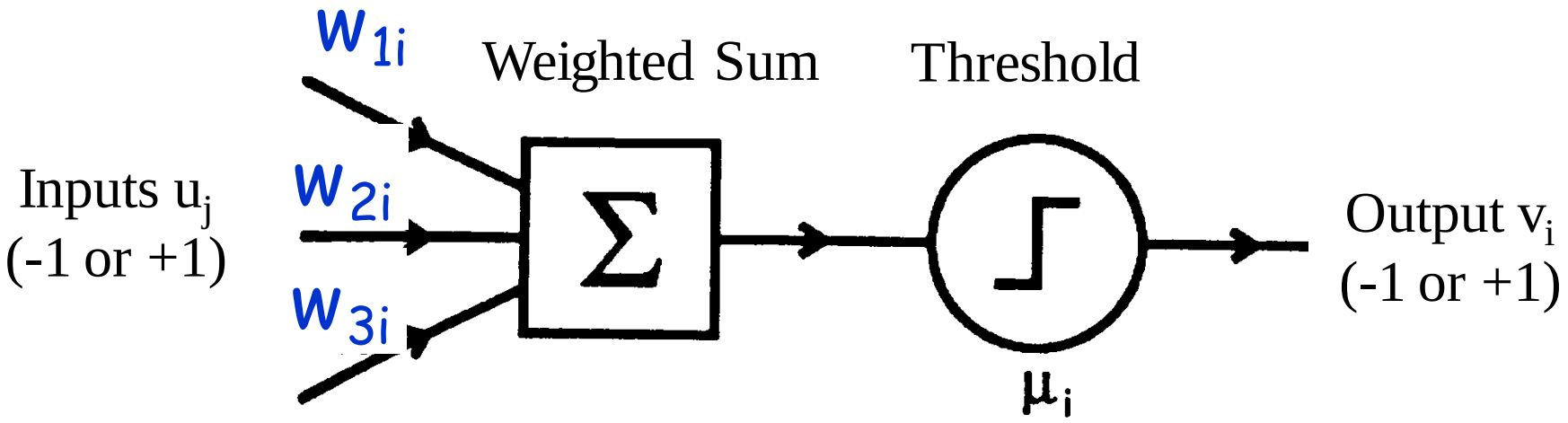



Output spike roughly dependent on whether sum of all inputs reaches a threshold

# Neurons as "Threshold Units"

**Artificial neuron:**

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




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

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

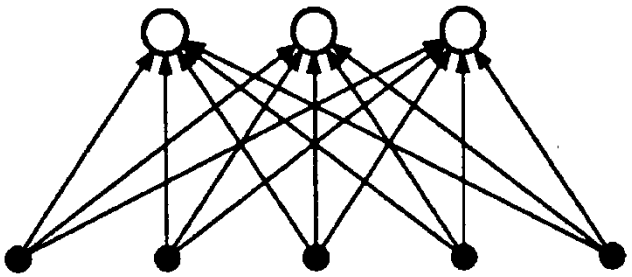
# "Perceptrons" for Classification

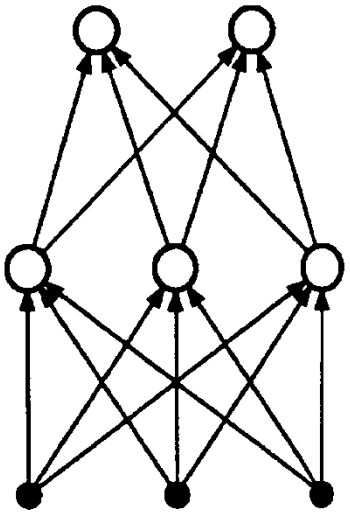
Fancy name for a type of layered "feed-forward" networks (no loops)

Uses artificial neurons ("units") with binary inputs and outputs

Multilayer

Single-layer





# Perceptrons and Classification

Consider a single-layer perceptron

- Weighted sum forms a *linear hyperplane*

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

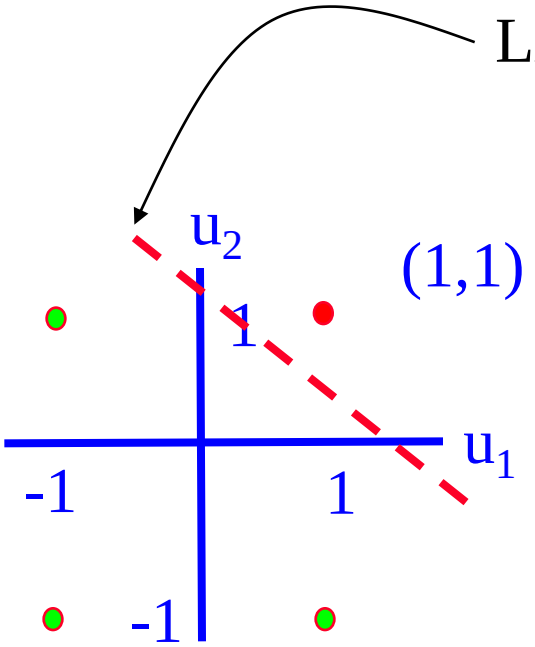
- Everything *on one side* of this hyperplane is in class 1 (output = +1) and everything *on other side* is class 2 (output = -1)

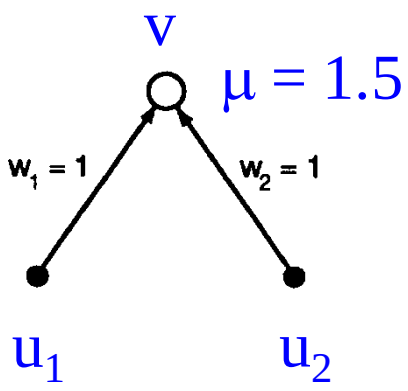
Any function that is linearly separable can be computed by a perceptron

# 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

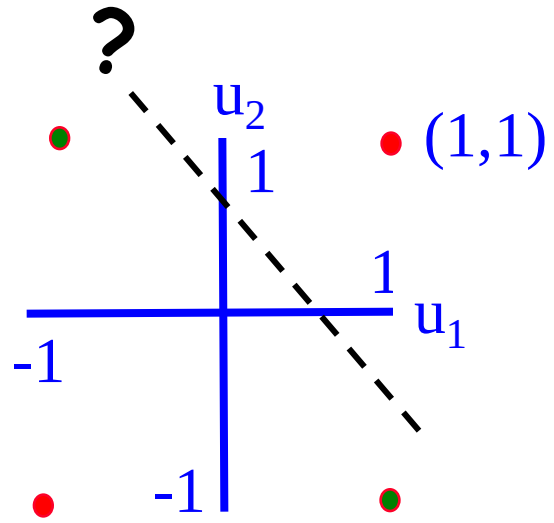
$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- $o$ is perceptron output
- $\eta$ 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 |



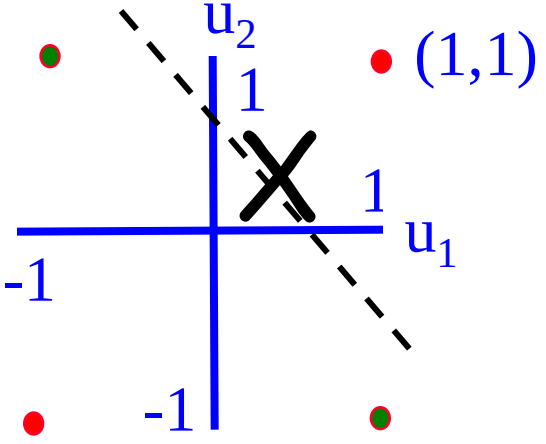

Can a perceptron separate the +1 outputs from the -1 outputs?

# 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!



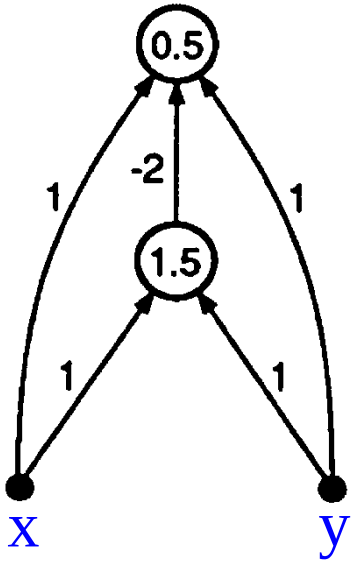
# How do we deal with linear inseparability?

# Idea 1: Multilayer Perceptrons

Removes limitations of single-layer networks

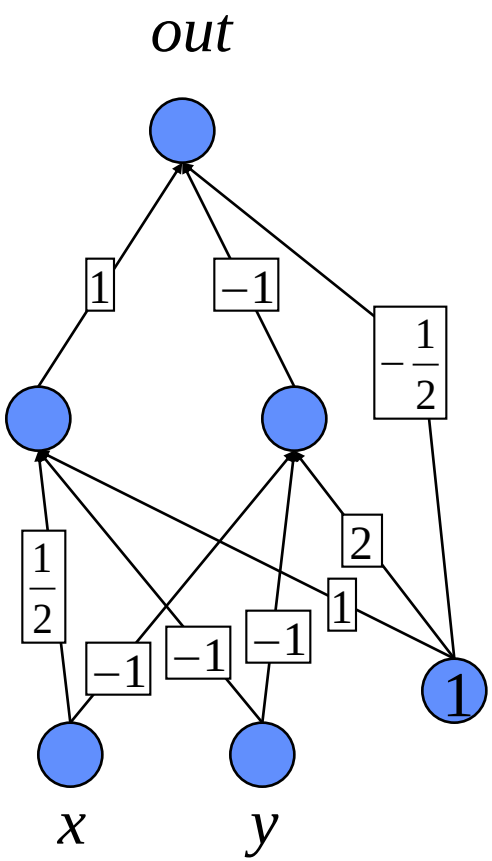
- Can solve XOR

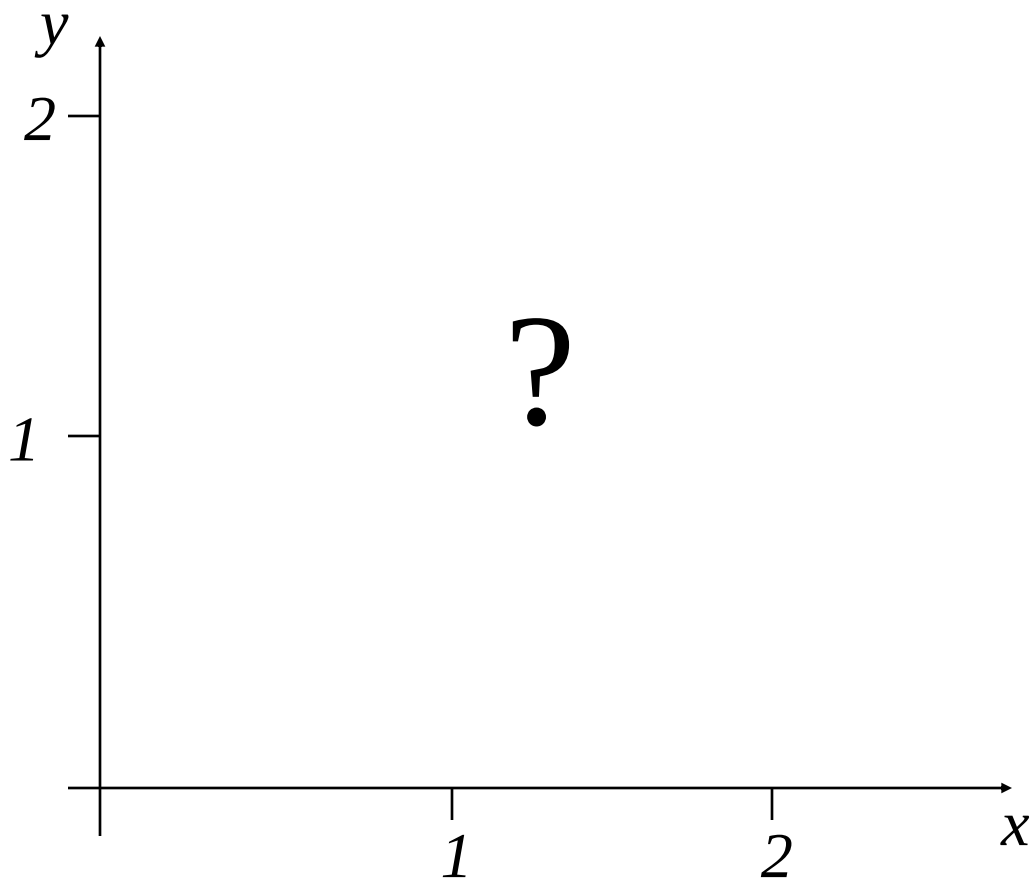
Example: Two-layer perceptron that computes XOR



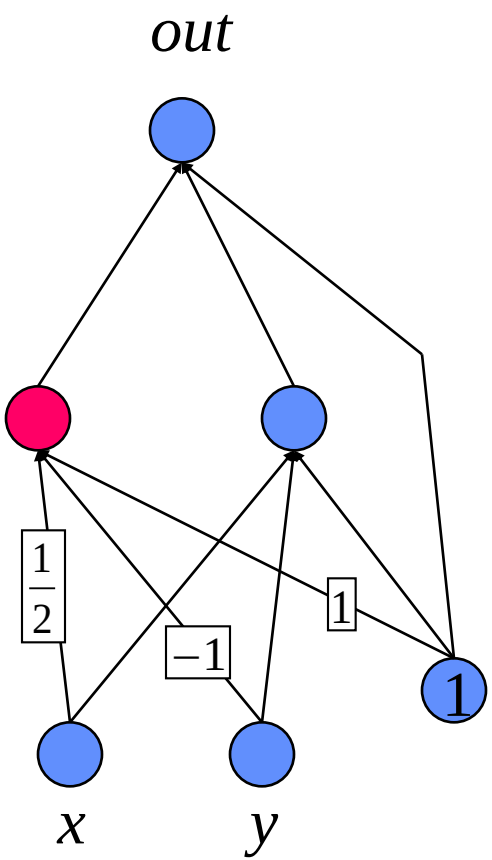

Output is +1 if and only if $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

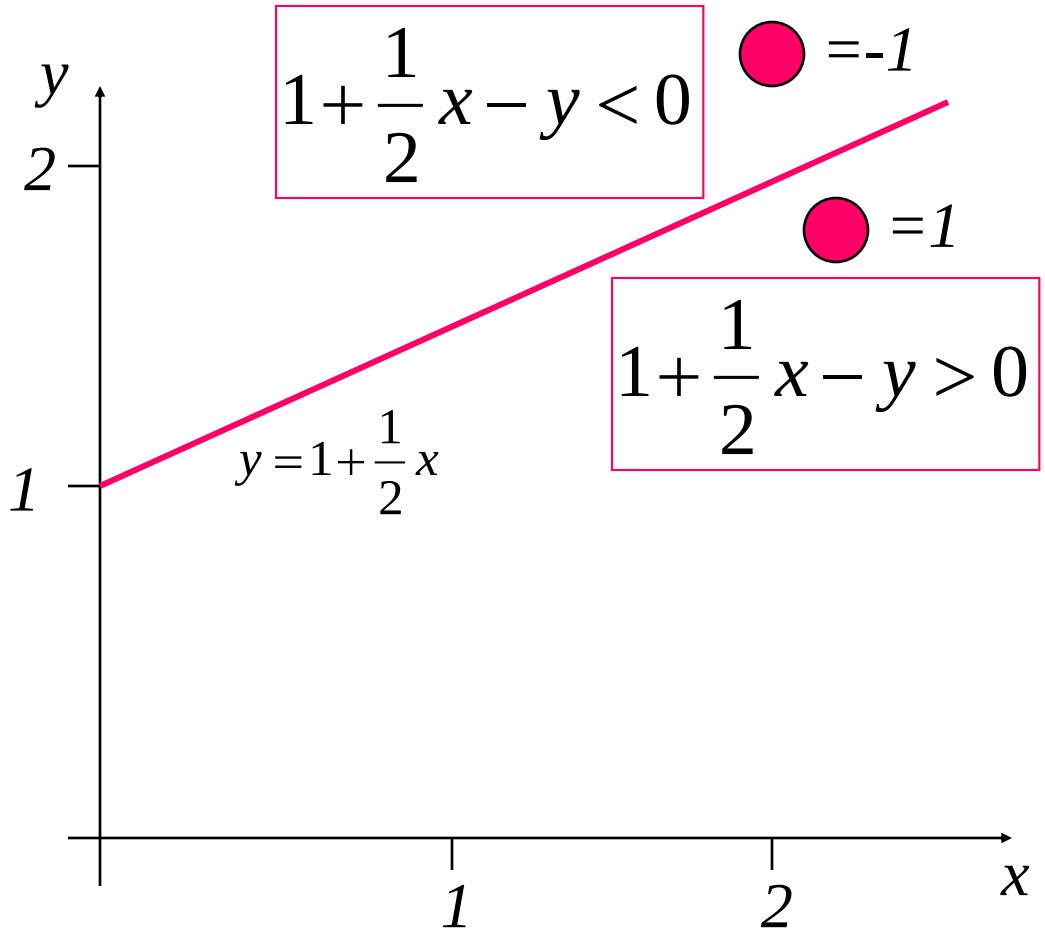
# Multilayer Perceptron: What does it do?



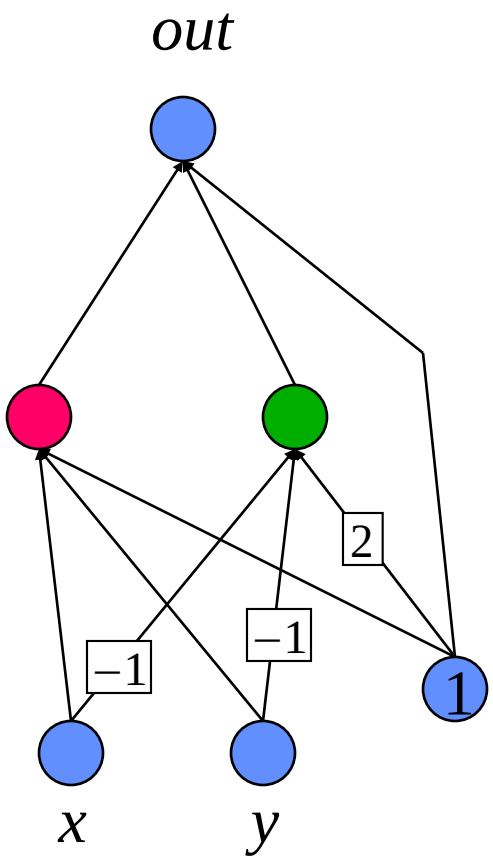



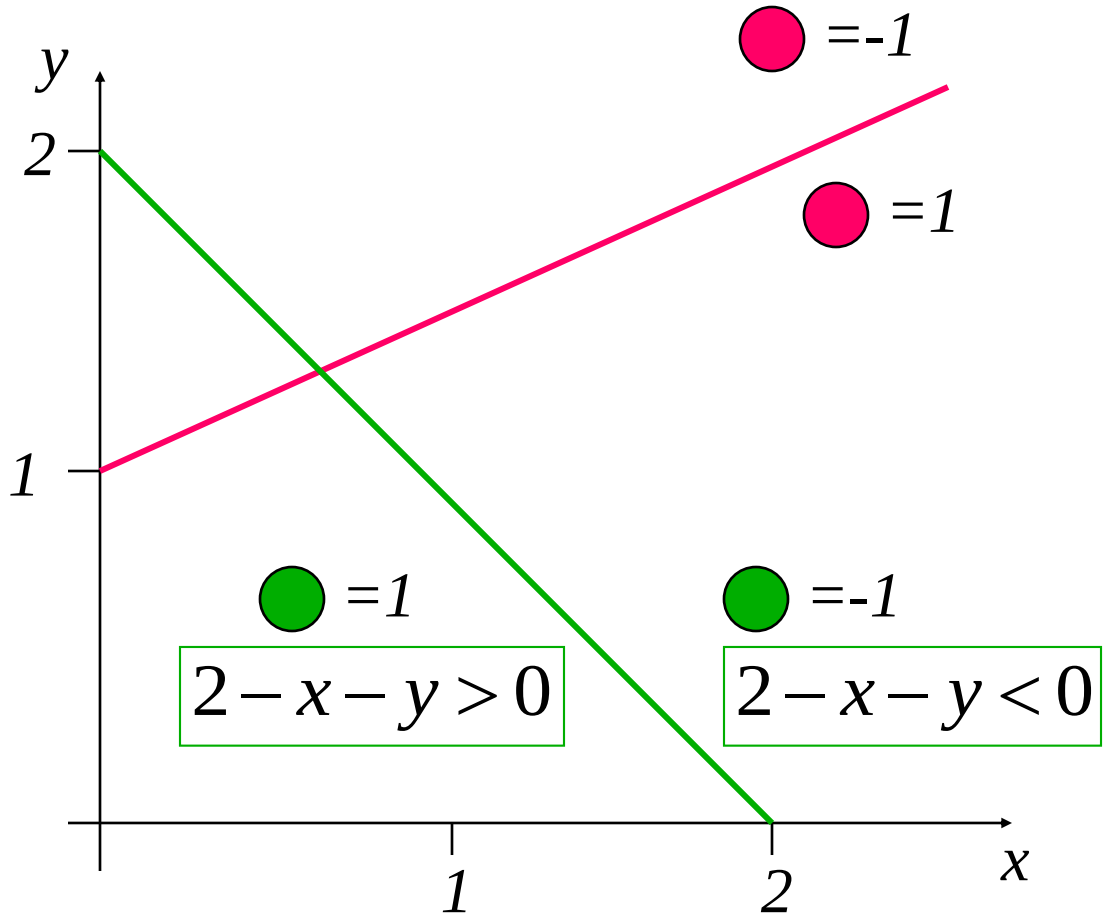
# Multilayer Perceptron: What does it do?



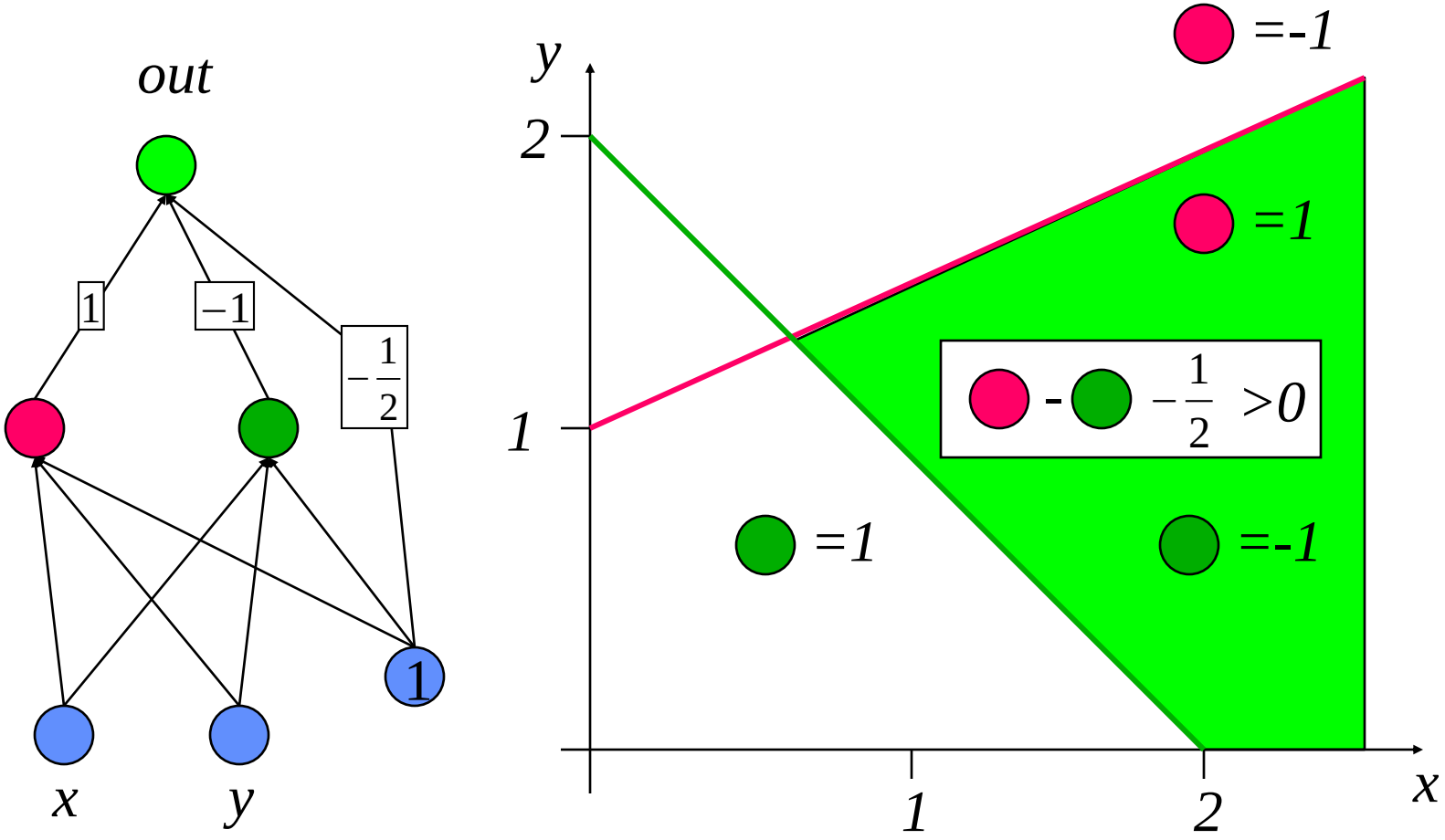



# Multilayer Perceptron: What does it do?

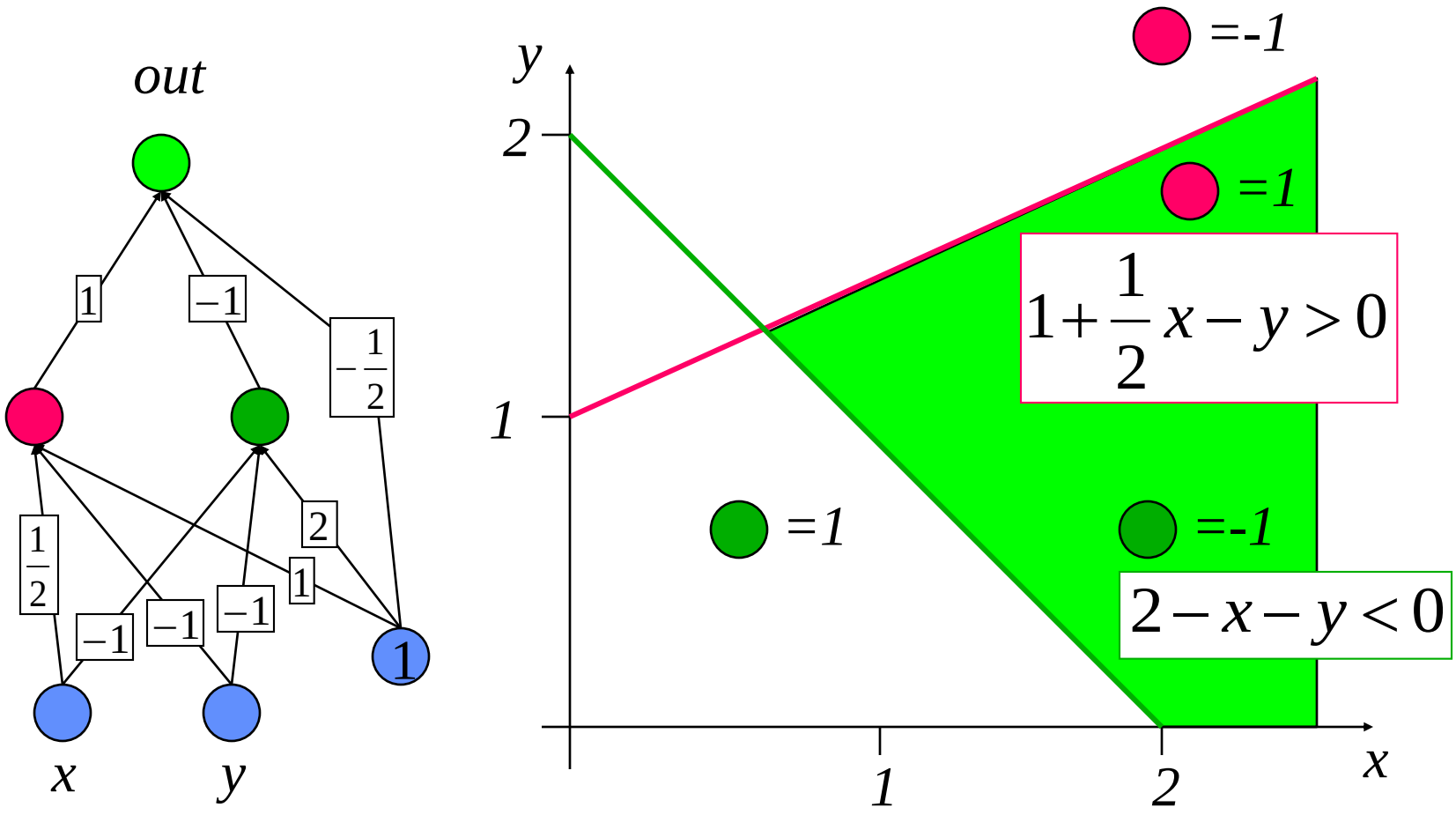


# Multilayer Perceptron: What does it do?



# Perceptrons as Constraint Satisfaction Networks



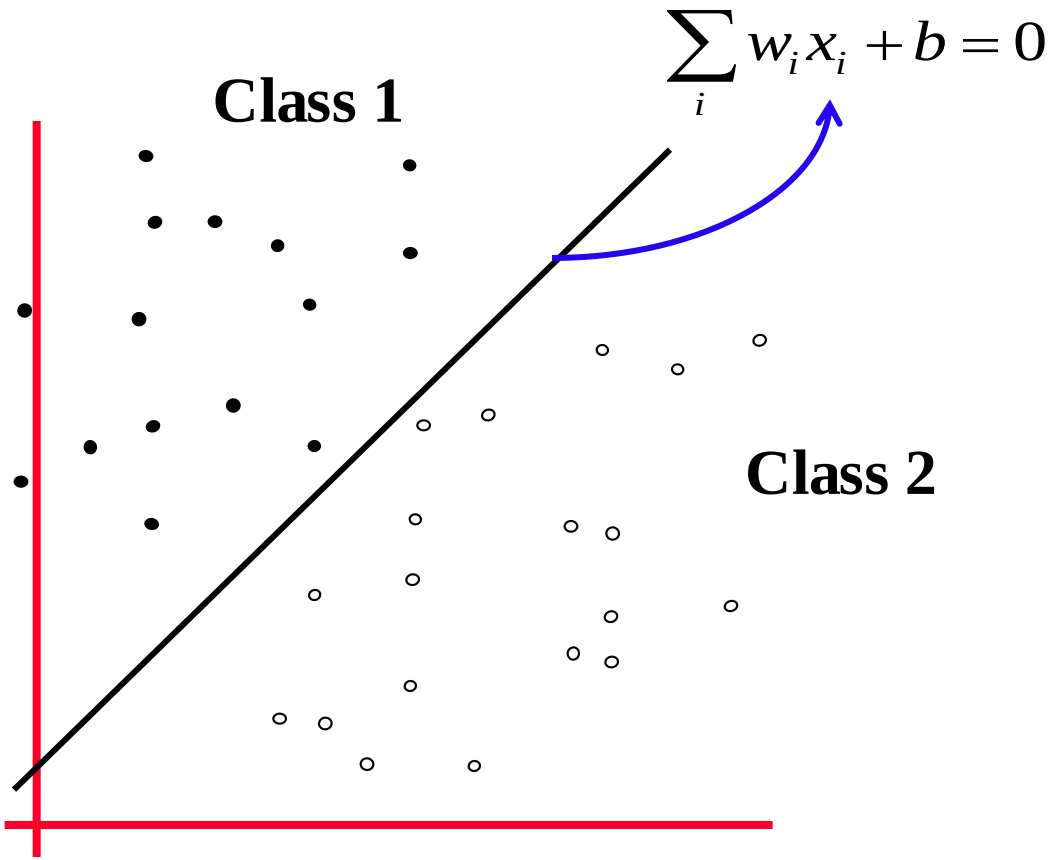
# Back to Linear Separability

- **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

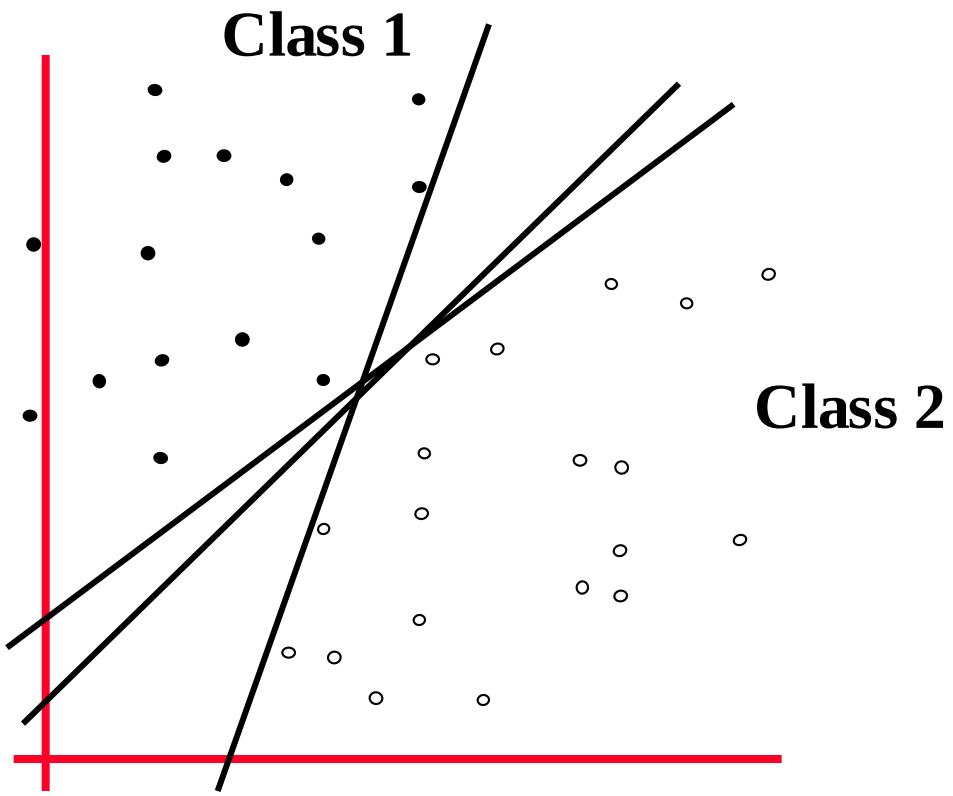



- denotes +1 output
- denotes -1 output

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

# Separating Hyperplanes

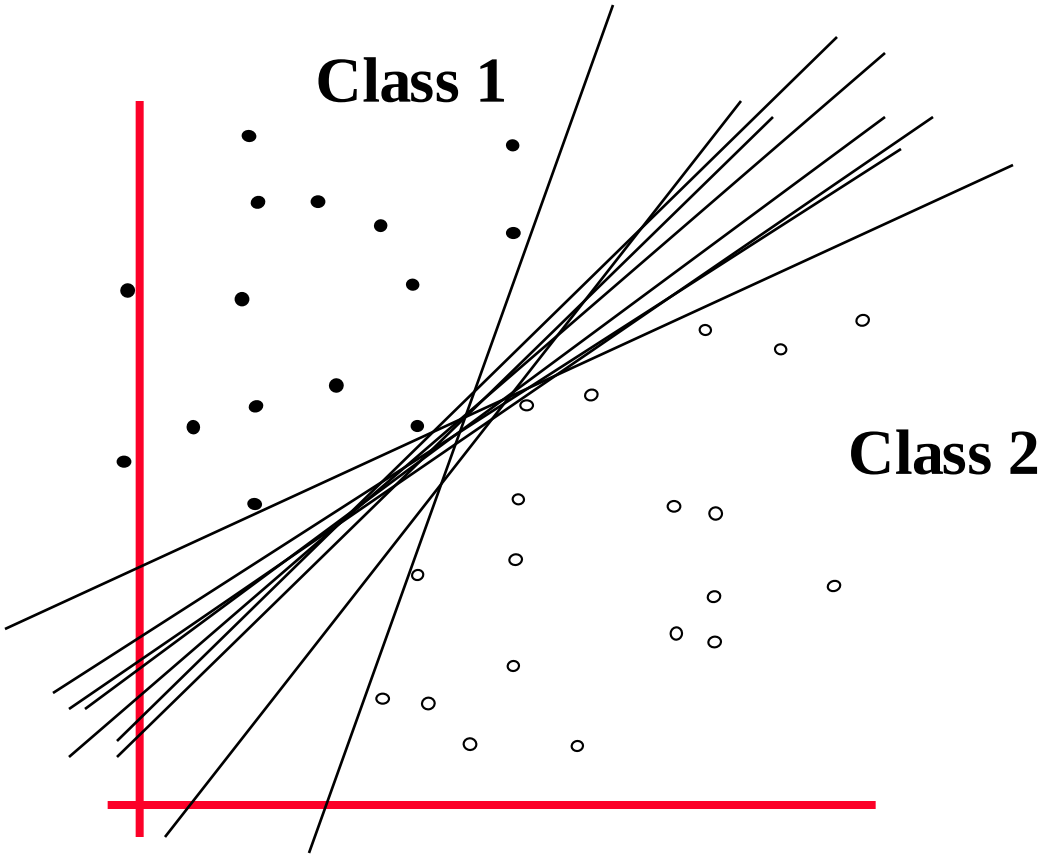
Different choices of **w** and *b* give different hyperplanes




- denotes +1 output
- denotes -1 output

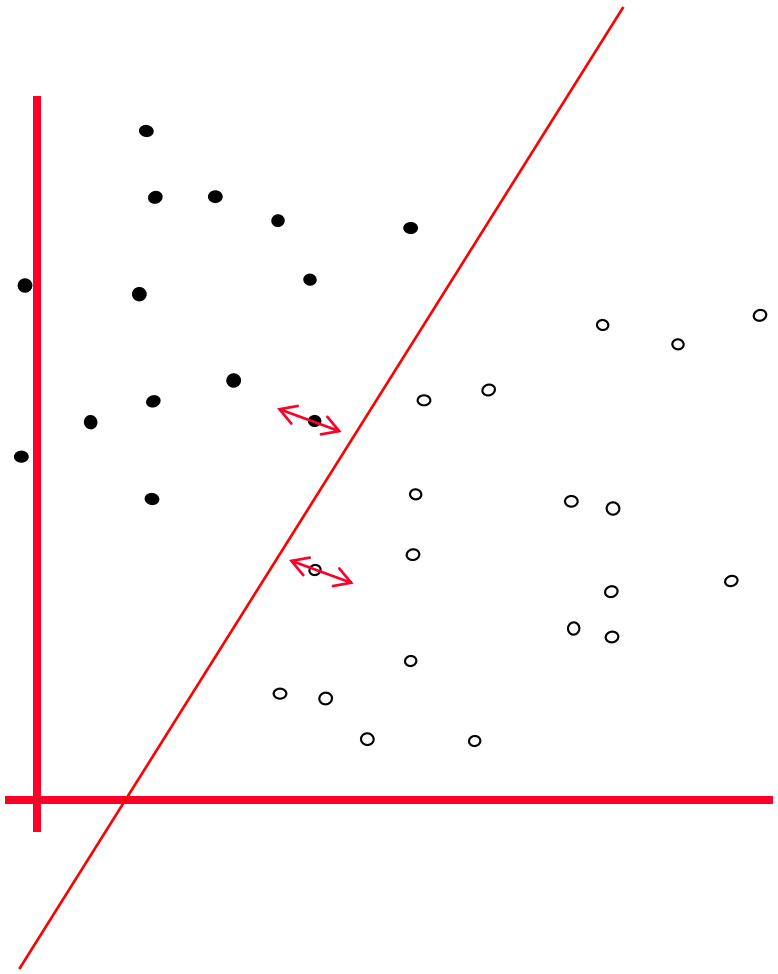
(This and next few slides adapted from Andrew Moore's)

# Which hyperplane is best?




- denotes +1 output
- denotes -1 output

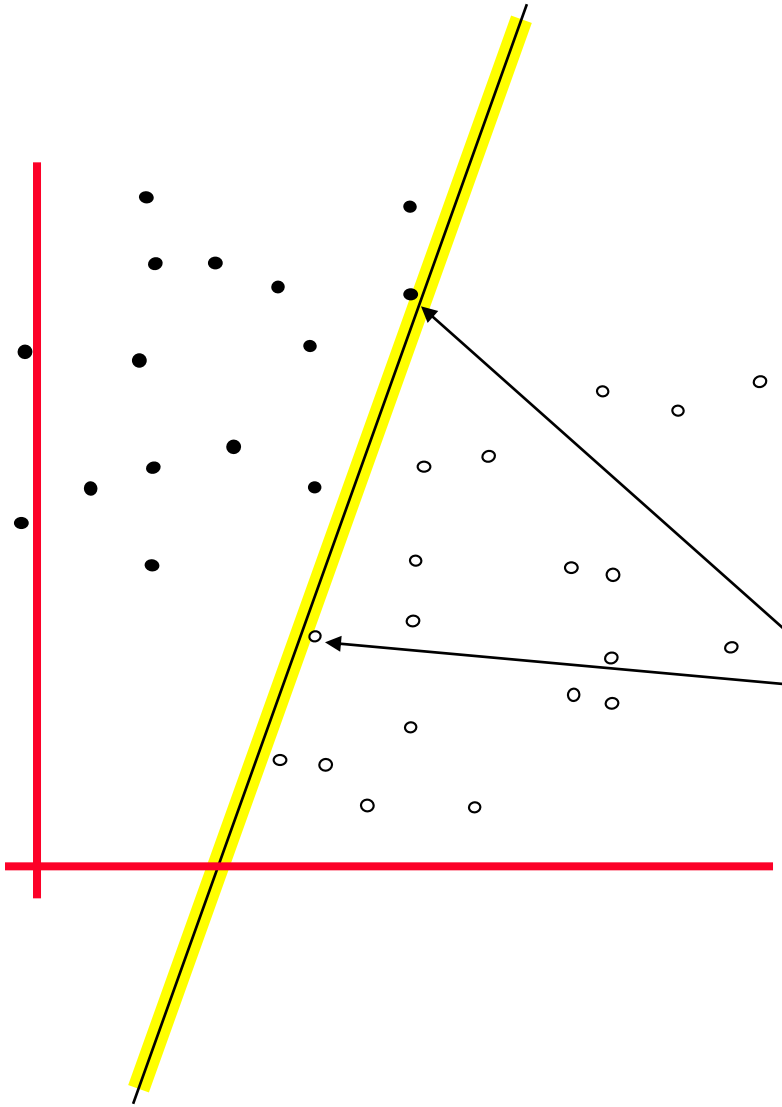
# How about the one right in the middle?



Intuitively, this boundary seems good

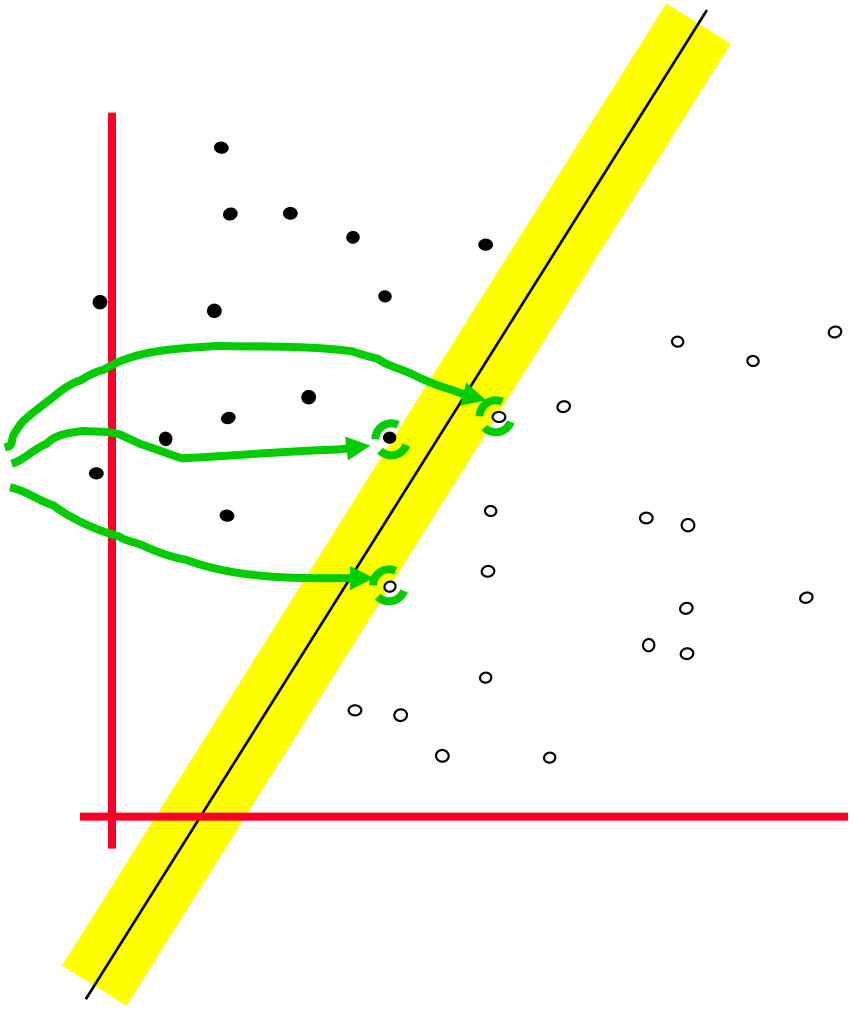
Avoids misclassification of new test points if they are generated from the same distribution as training points

# Margin



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

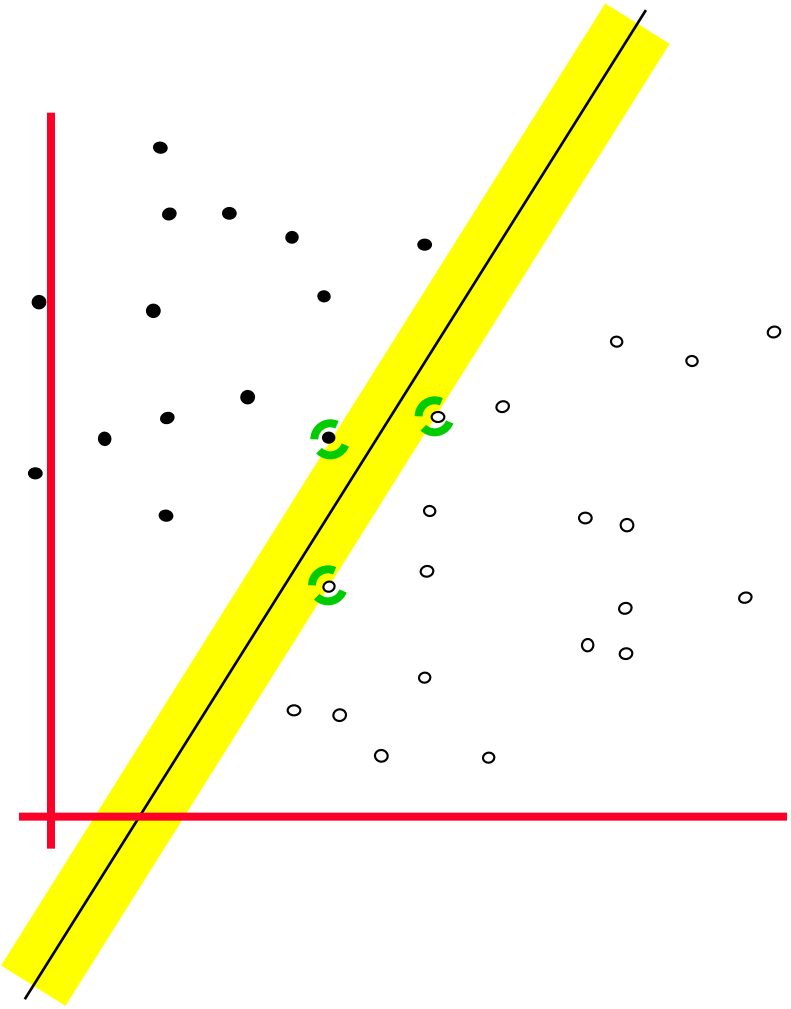
# Maximum Margin and Support Vector Machine



Support Vectors are those datapoints that the margin pushes up against

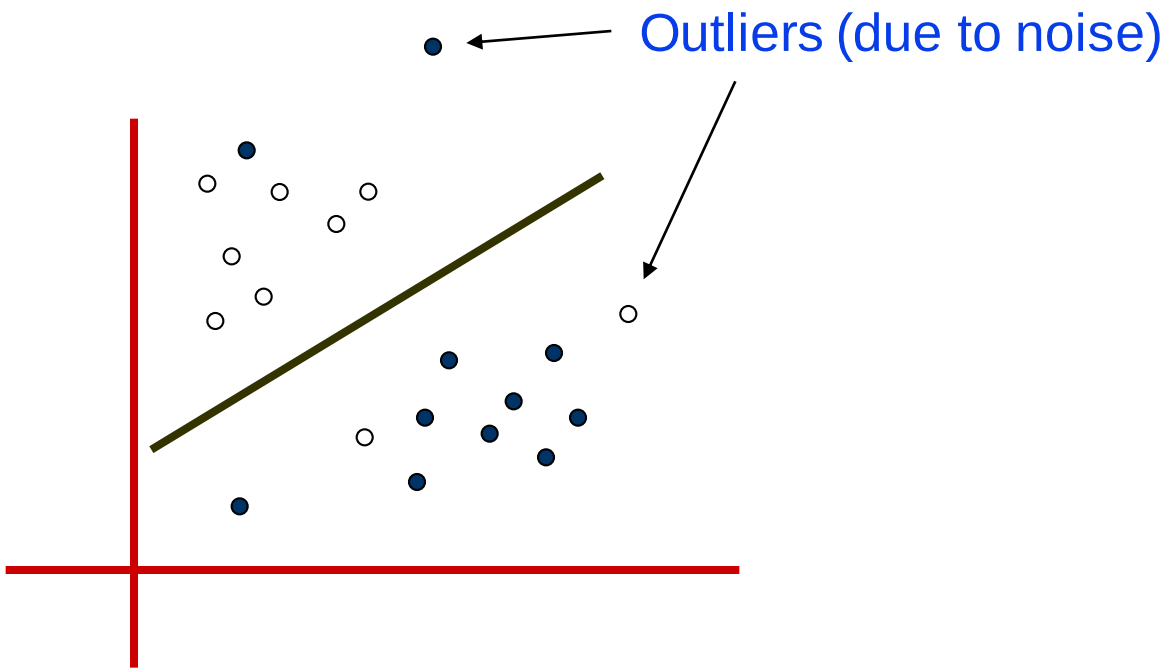
The maximum margin classifier is called a Support Vector Machine (in this case, a Linear SVM or LSVM)

# Why Maximum Margin?

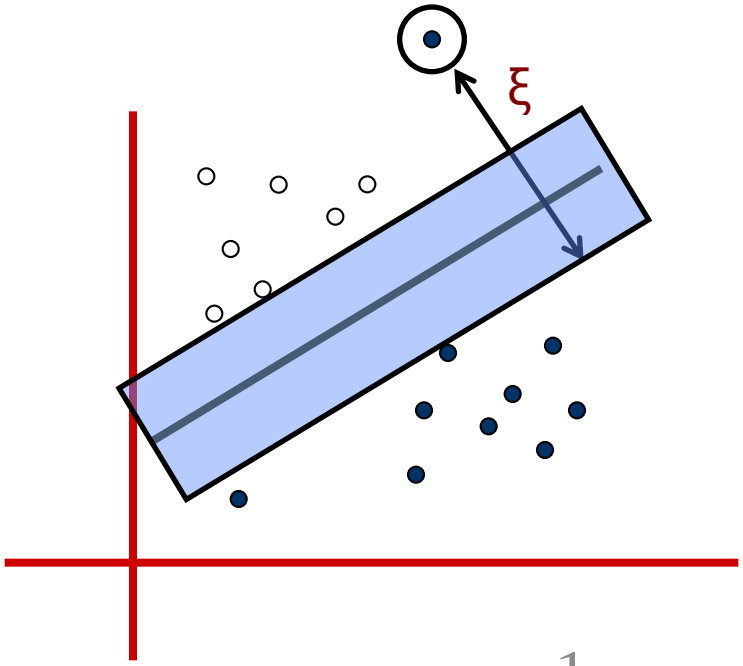


- Robust to small perturbations of data points near boundary
- There exists theory showing this is best for generalization to new points
- Empirically works great

# What if data is not linearly separable?



# Approach 1: Soft Margin SVMs



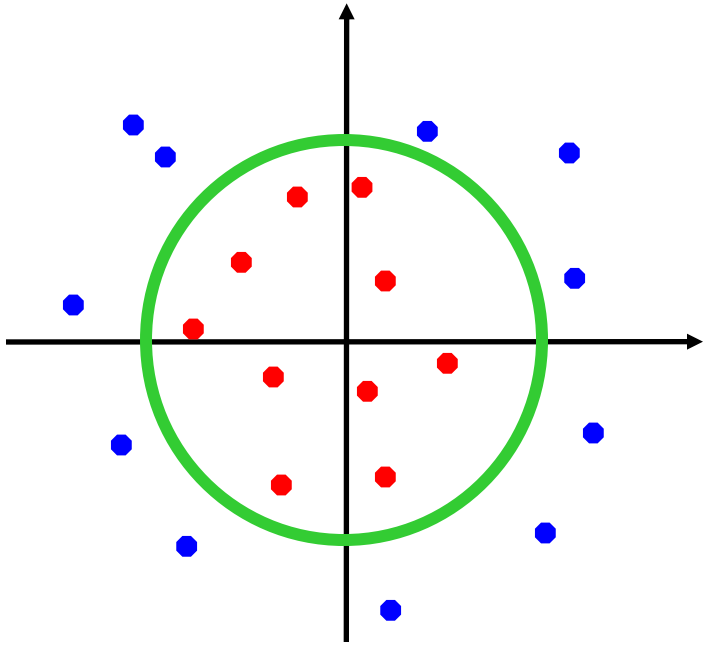

Allow *errors* $\xi_i$ (deviations from margin)

Trade off margin with errors.

Minimize: $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$ subject to:

$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and } \xi_i \geq 0, \forall i$

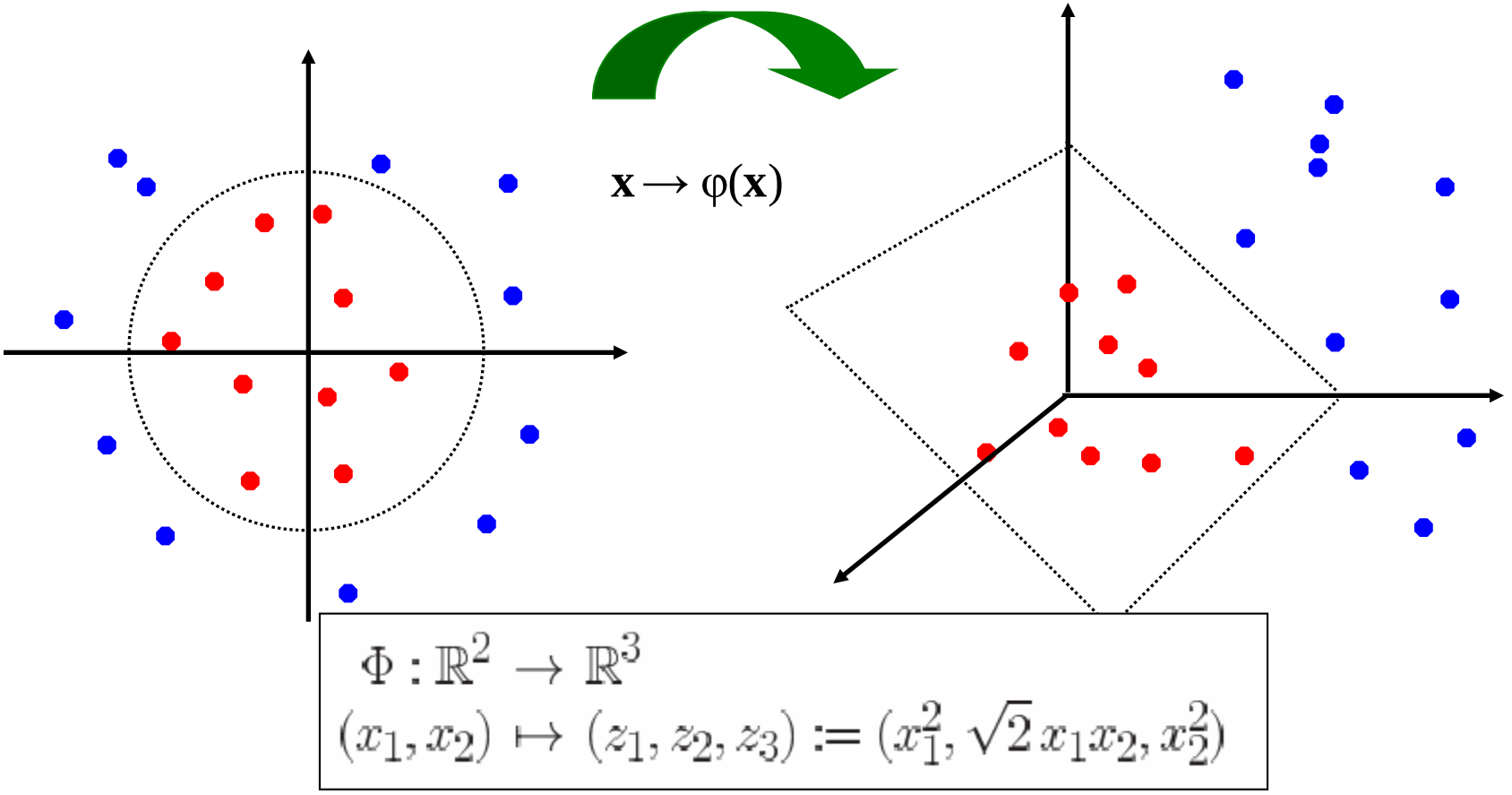
# What if data is not linearly separable: Other ideas?



Not linearly separable

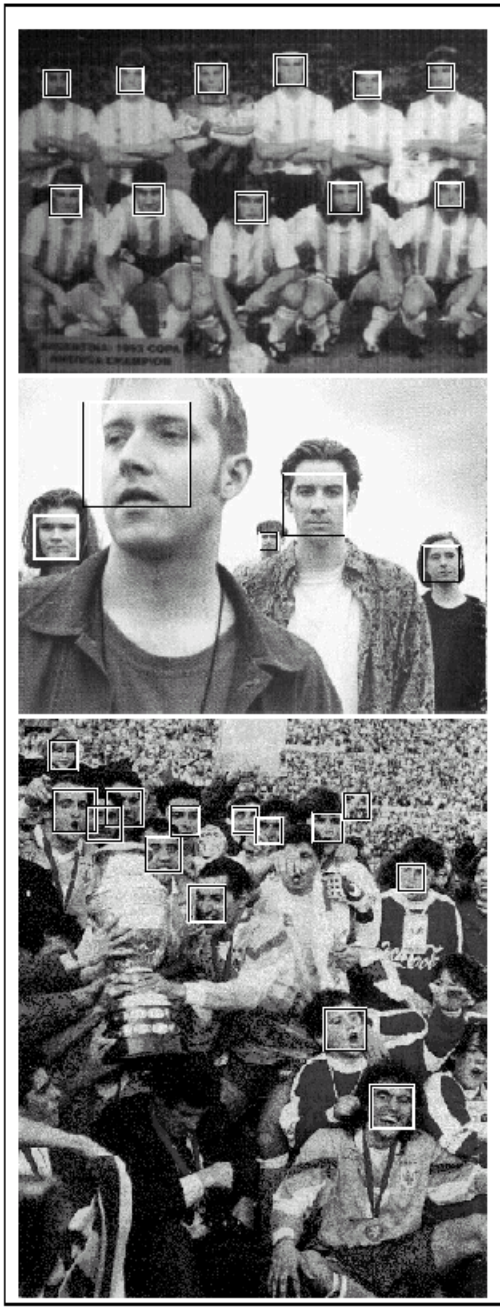
# What if data is not linearly separable?

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




Kernel: additional bias to convert into high d space

# Face Detection using SVMs



| | Test Set A | | Test Set B | |
|---|---|---|---|---|
| | Detect Rate | False Alarms | Detect Rate | False Alarms |
| SVM | 97.1 % | 4 | 74.2% | 20 |
| Sung *et al.* | 94.6 % | 2 | 74.2% | 11 |

Kernel used: Polynomial of degree 2

(Osuna, Freund, Girosi, 1998)

# 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**

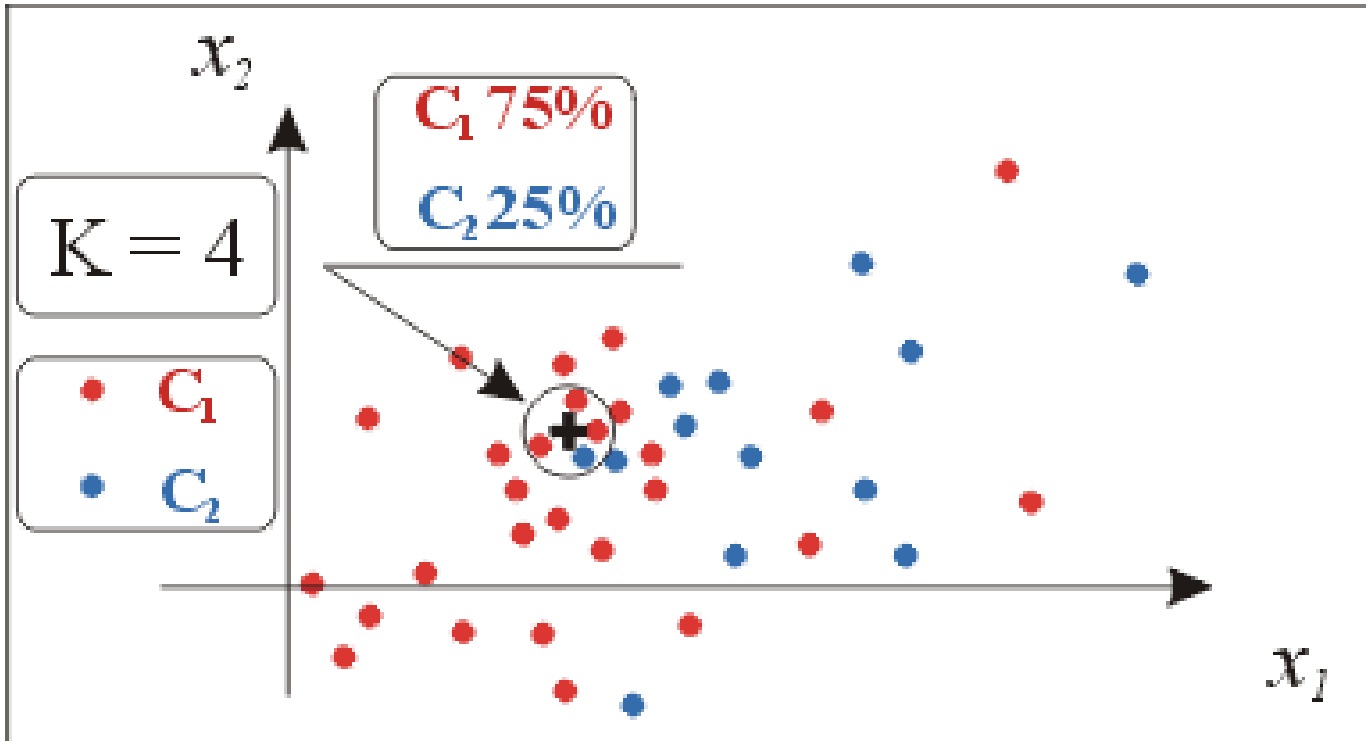
Examples:

- 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 $(x_1,x_2)$

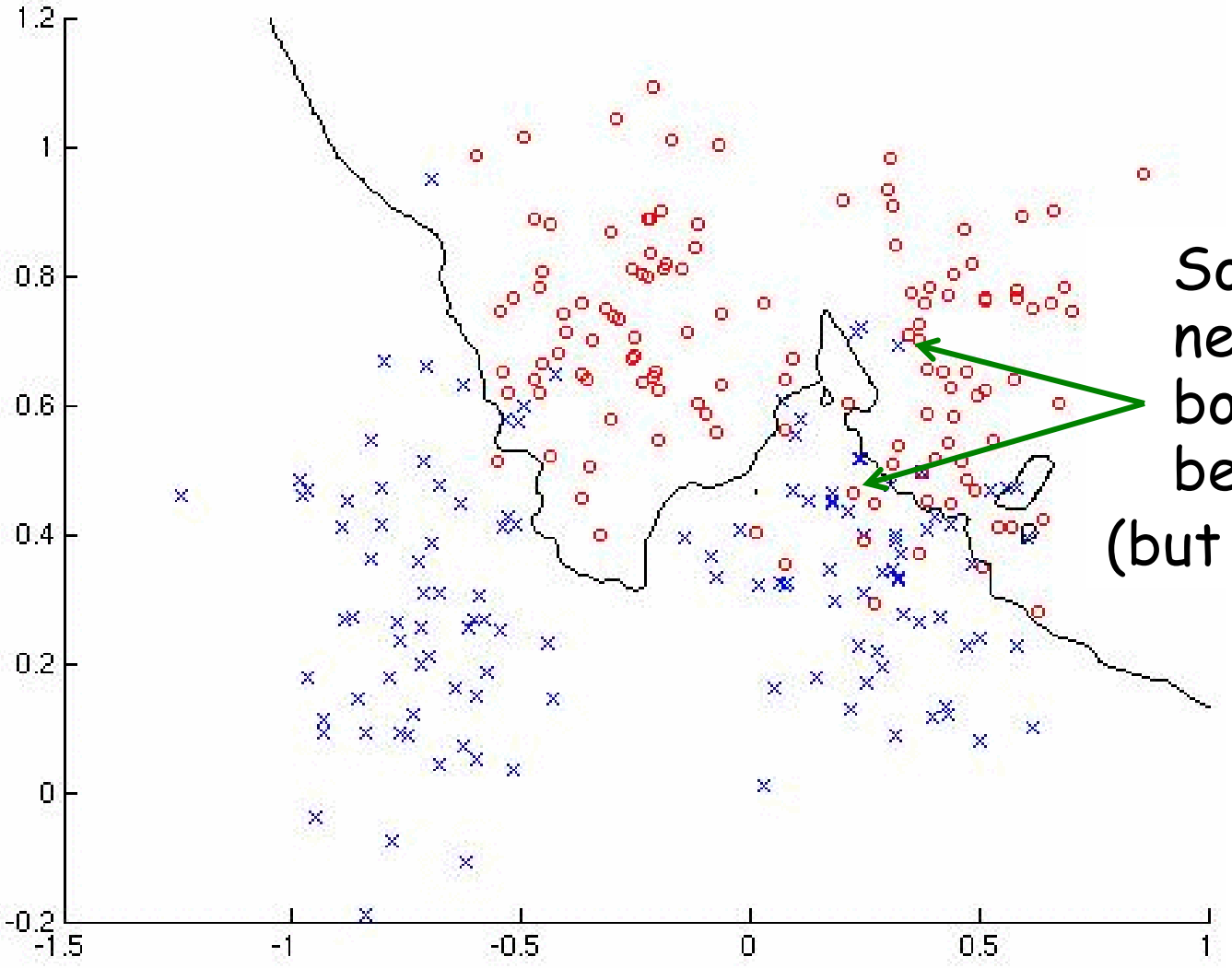
Two classes: $C_1$ and $C_2$. New Data Point +




K = 4: Look at 4 nearest neighbors of +

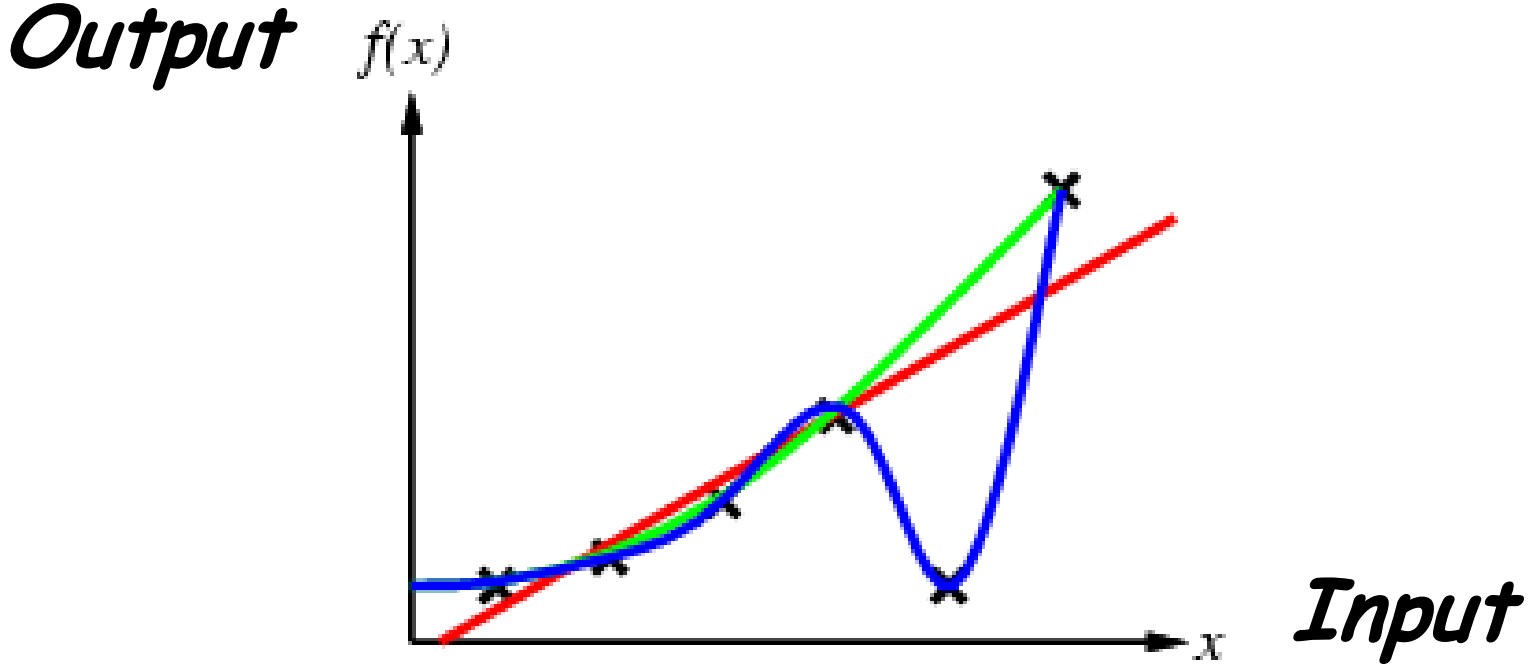
3 are in $C_1$, so classify + as $C_1$

# 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

learning multi-layer networks: backpropagation

# 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