Support Vector Machines & Kernels

Doing *really* well with linear decision surfaces

Outline

- Prediction
  - Why might predictions be wrong?
- Support vector machines
  - Doing really well with linear models
- Kernels
  - Making the non-linear linear

Why Might Predictions be Wrong?

- True non-determinism
  - Flip a biased coin
  - p(heads) = θ
  - Estimate θ
  - If $\theta > 0.5$ predict 'heads', else 'tails'

Lots of ML research on problems like this:

- Learn a model
- Do the best you can in expectation

Why Might Predictions be Wrong?

- Partial observability
  - Something needed to predict y is missing from observation $\mathbf{x}$
  - N-bit parity problem
    - $\mathbf{x}$ contains N-1 bits (hard PO)
    - $\mathbf{x}$ contains N bits but learner ignores some of them (soft PO)

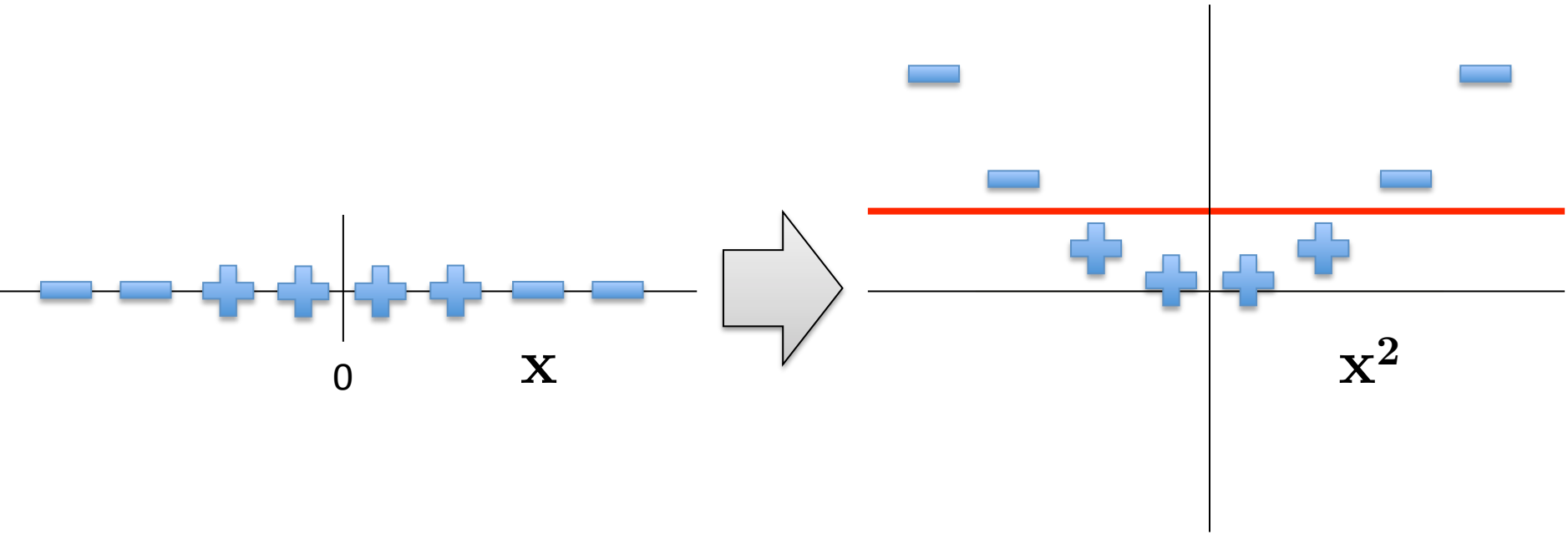
- Noise in the observation $\mathbf{x}$
  - Measurement error
  - Instrument limitations

Why Might Predictions be Wrong?

- True non-determinism
- Partial observability
  - hard, soft
- Representational bias
- Algorithmic bias
- Bounded resources

Representational Bias

- Having the right features ($\mathbf{x}$) is crucial



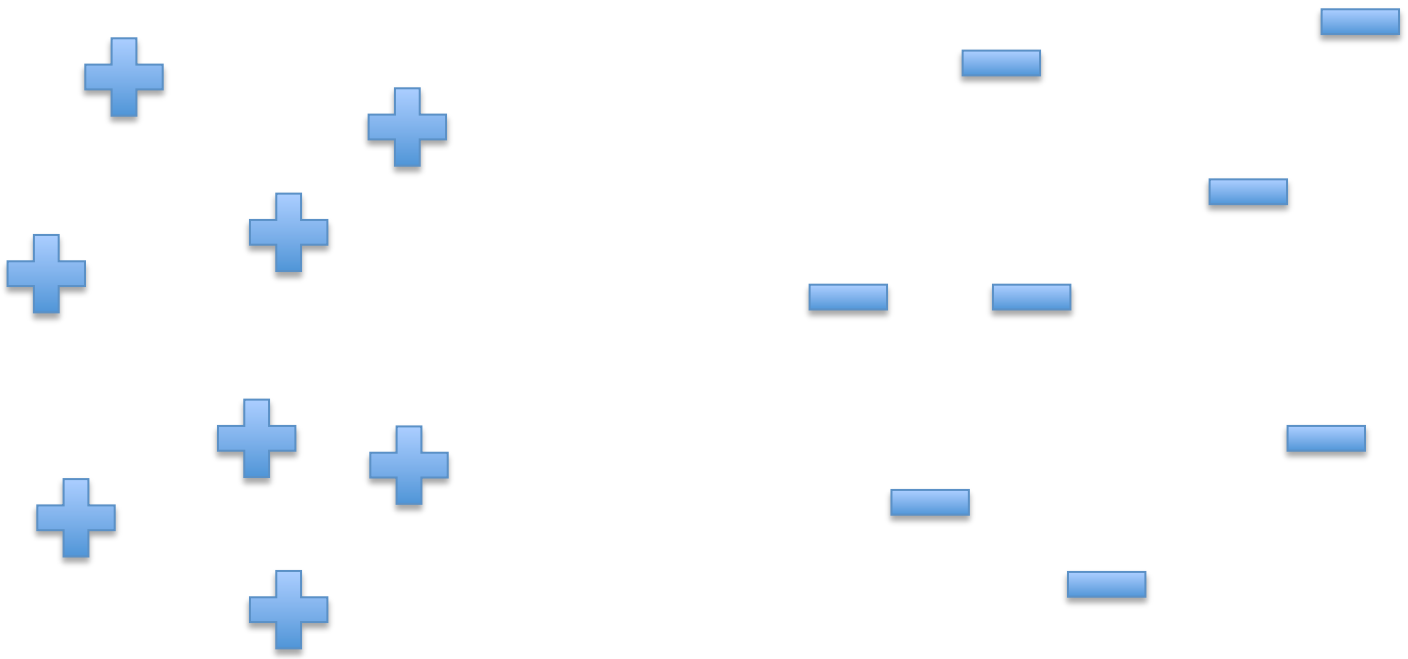
Support Vector Machines

Doing ***Really*** Well with Linear Decision Surfaces

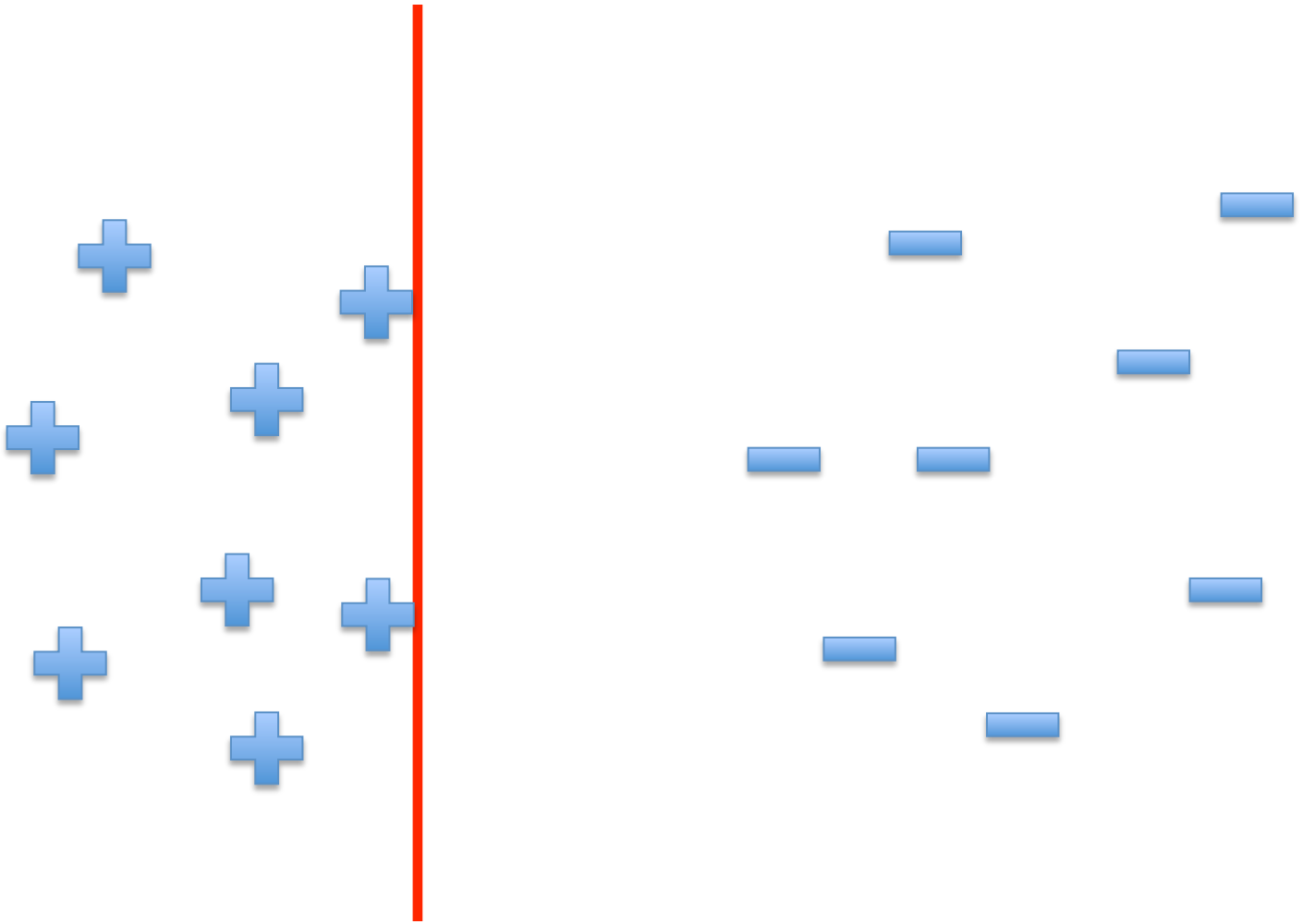
Strengths of SVMs

- Good generalization
  - in theory
  - in practice
- Works well with few training instances
- Find globally best model
- Efficient algorithms
- Amenable to the kernel trick

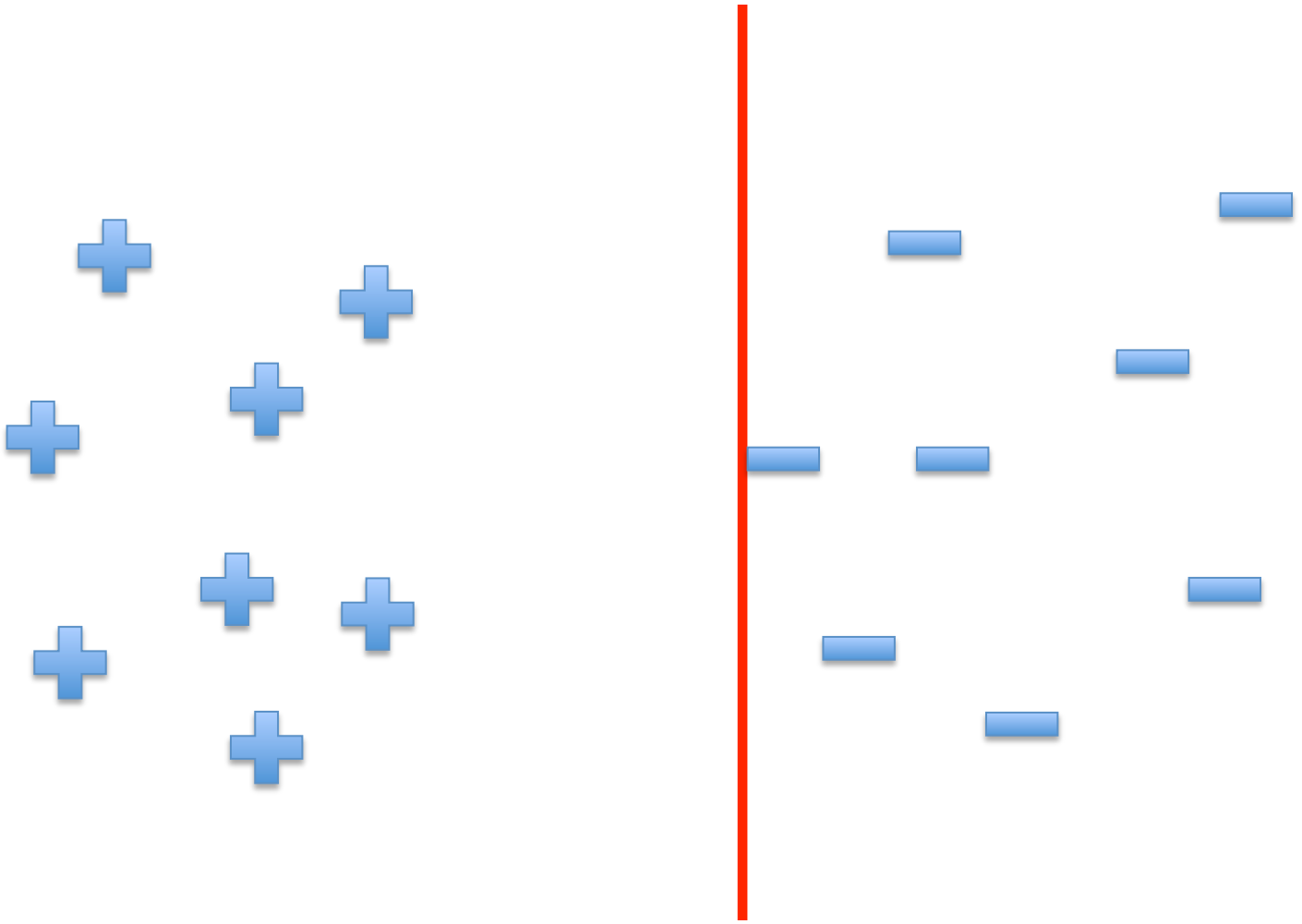
Intuitions



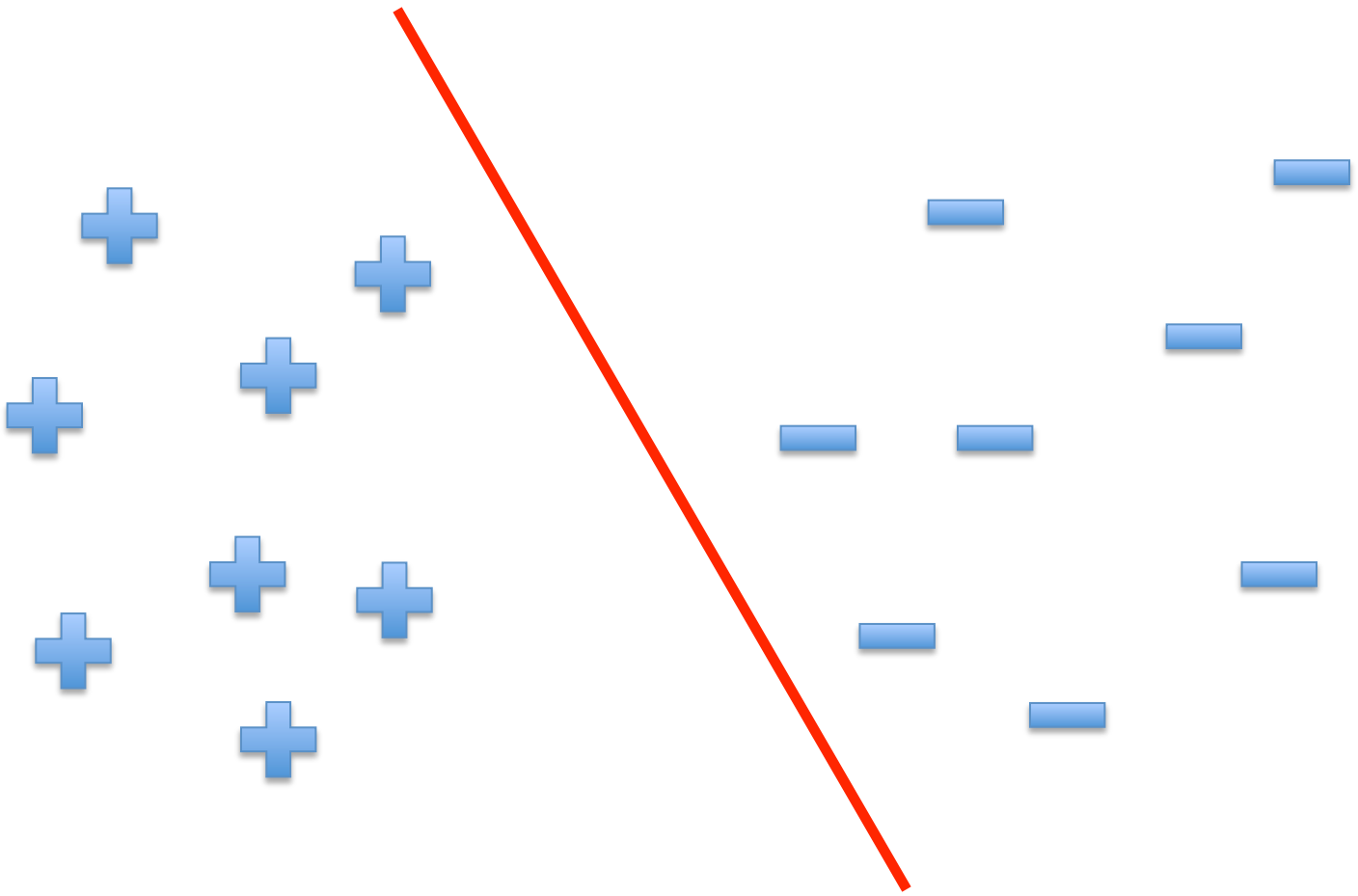
Intuitions



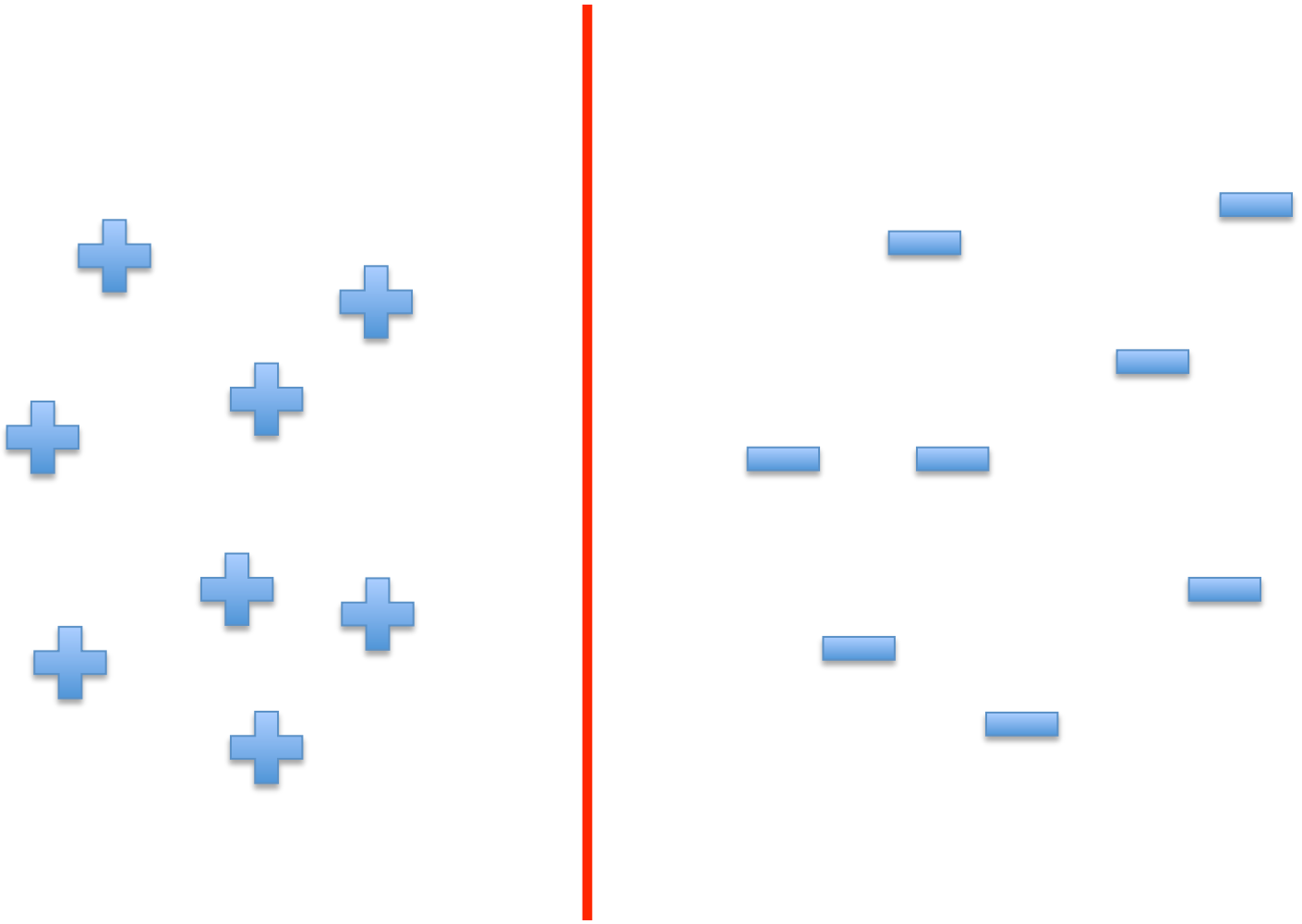
Intuitions



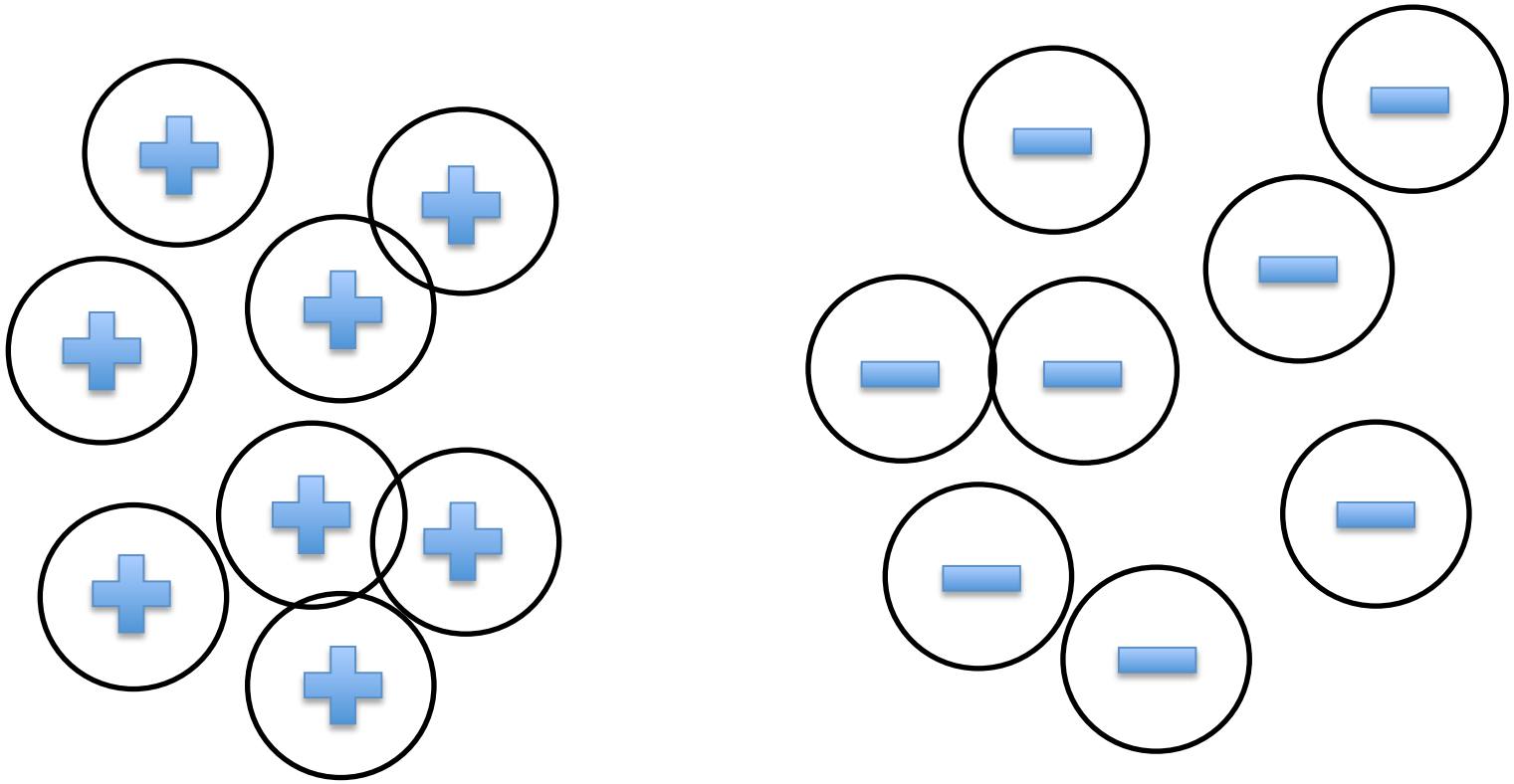
Intuitions



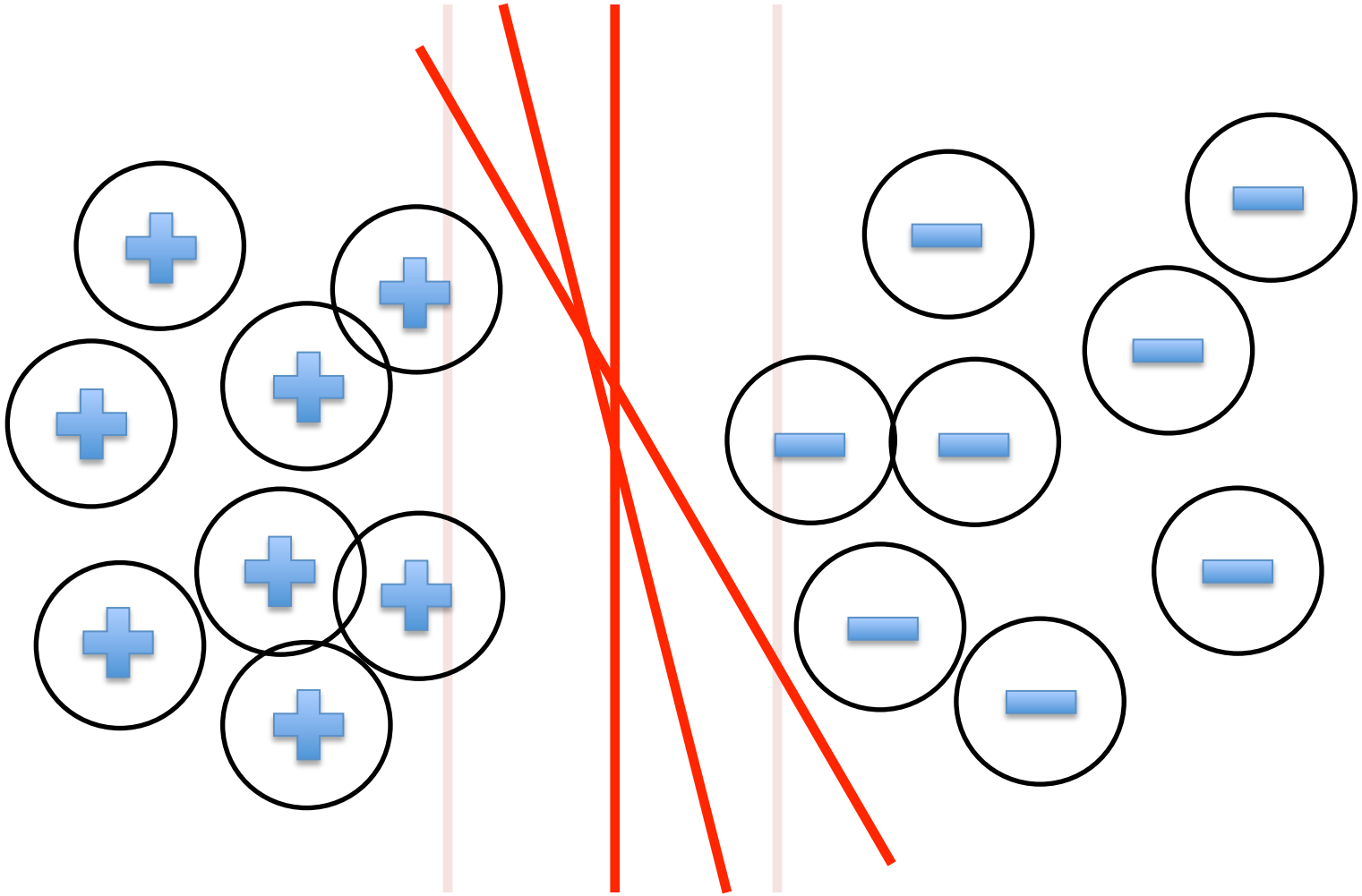
A “Good” Separator



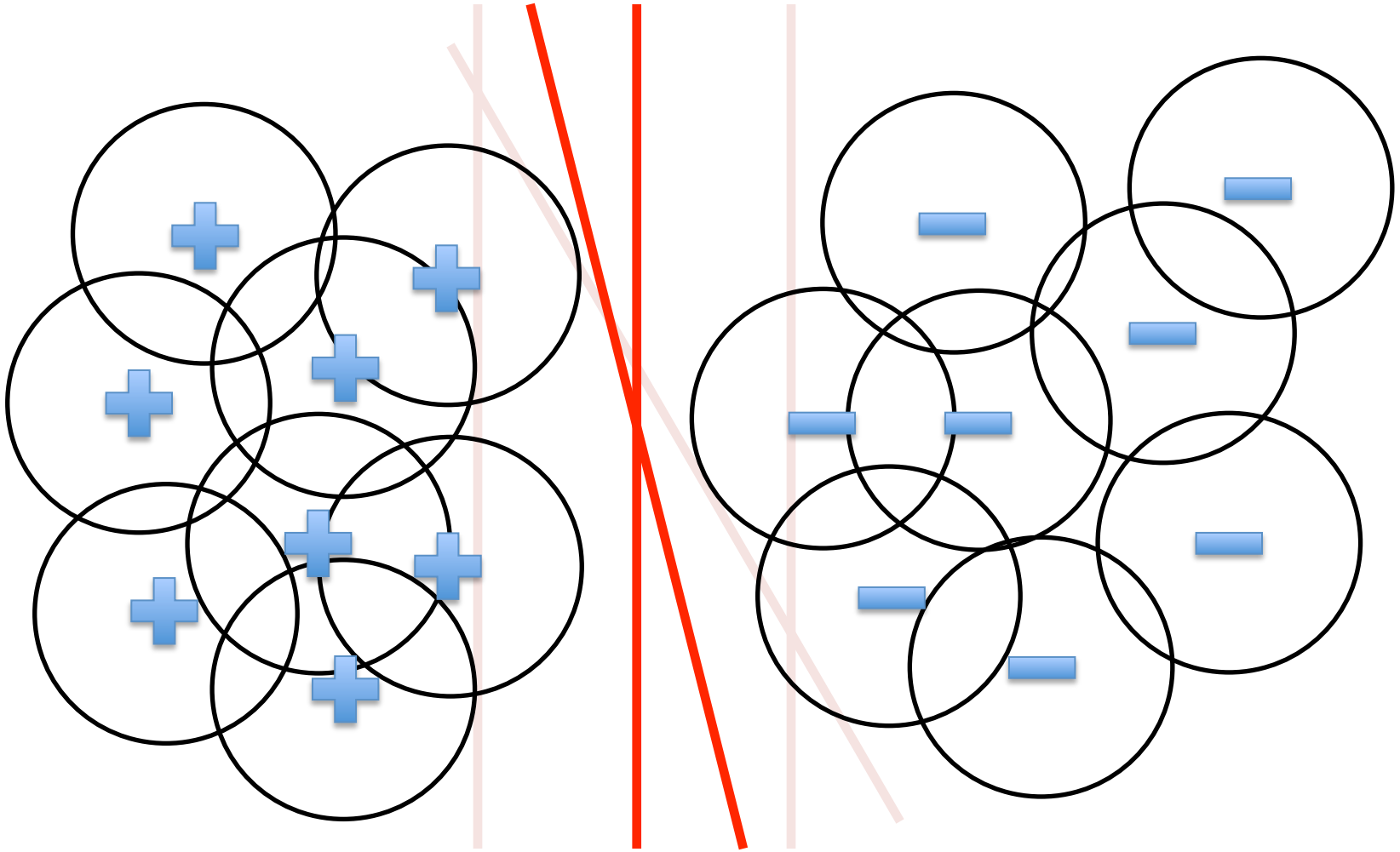
Noise in the Observations



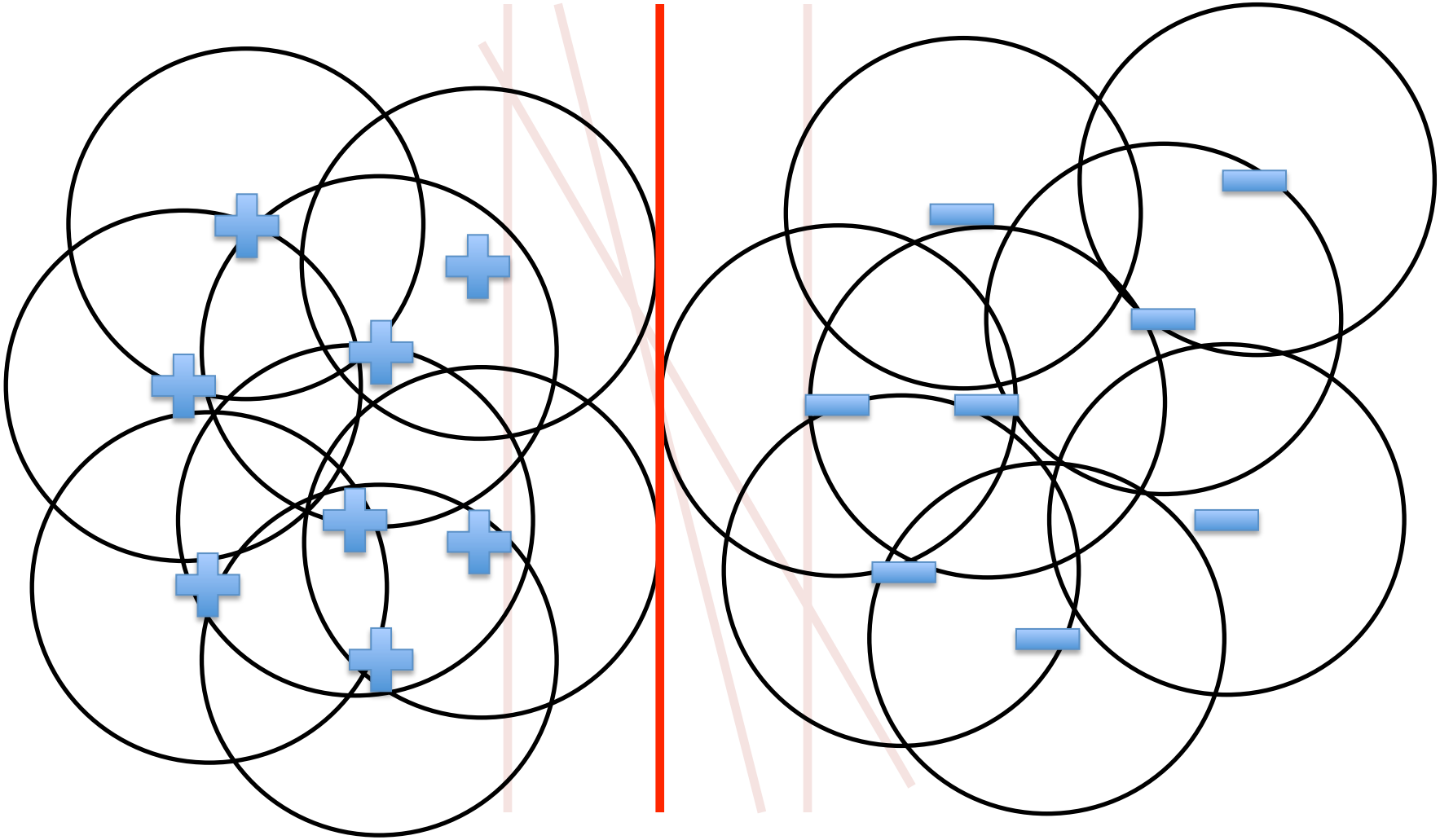
Ruling Out Some Separators



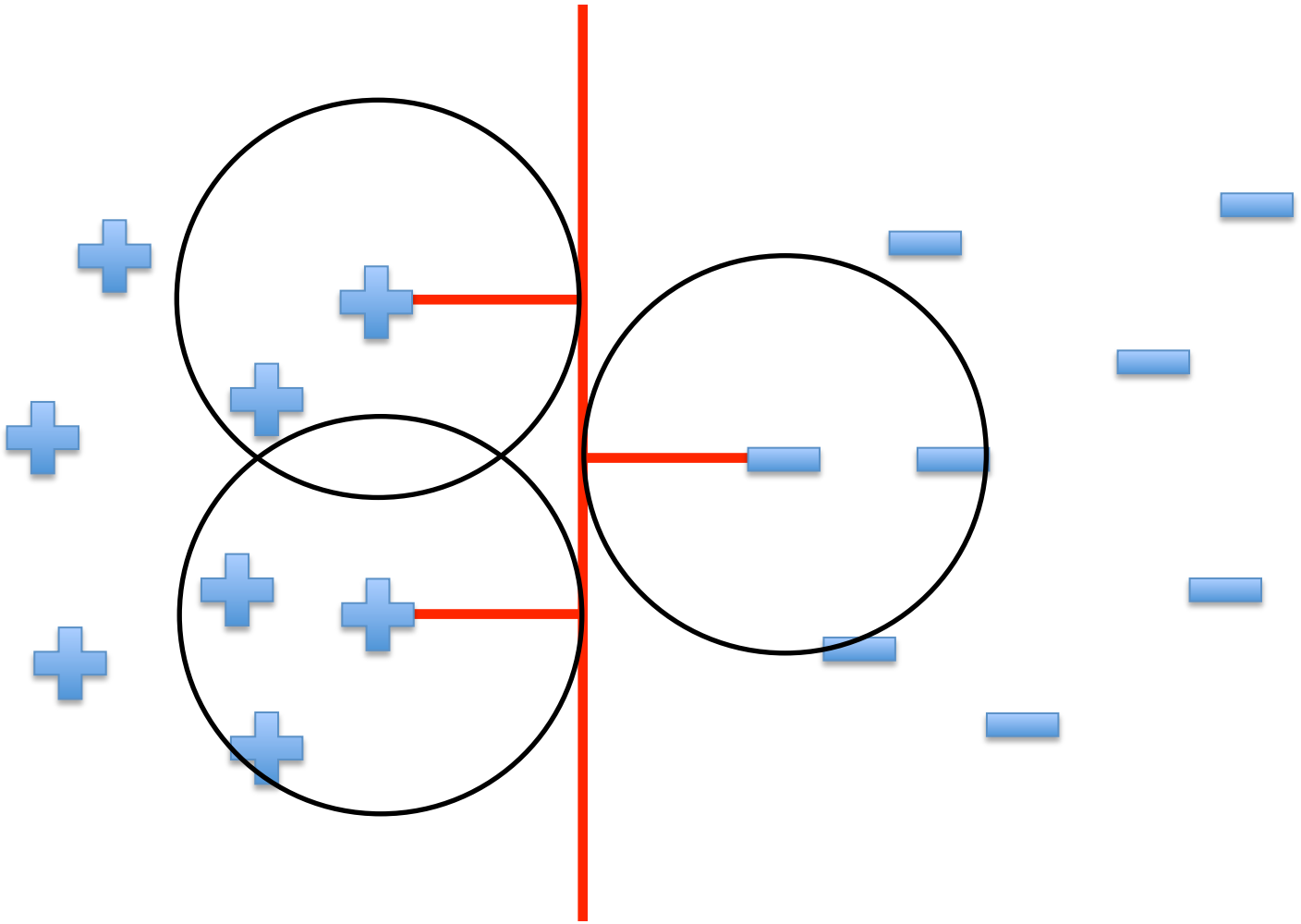
Lots of Noise



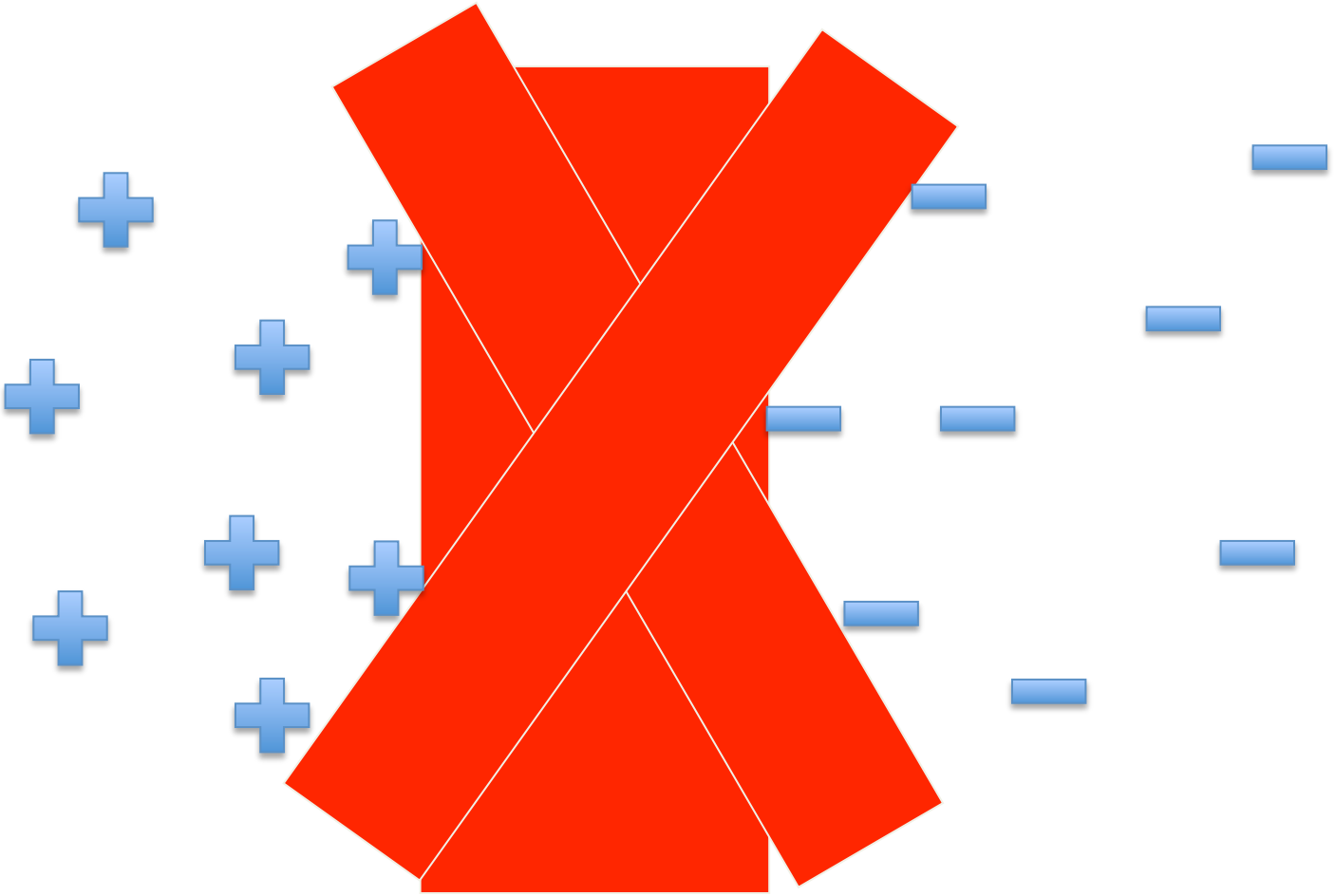
Only One Separator Remains



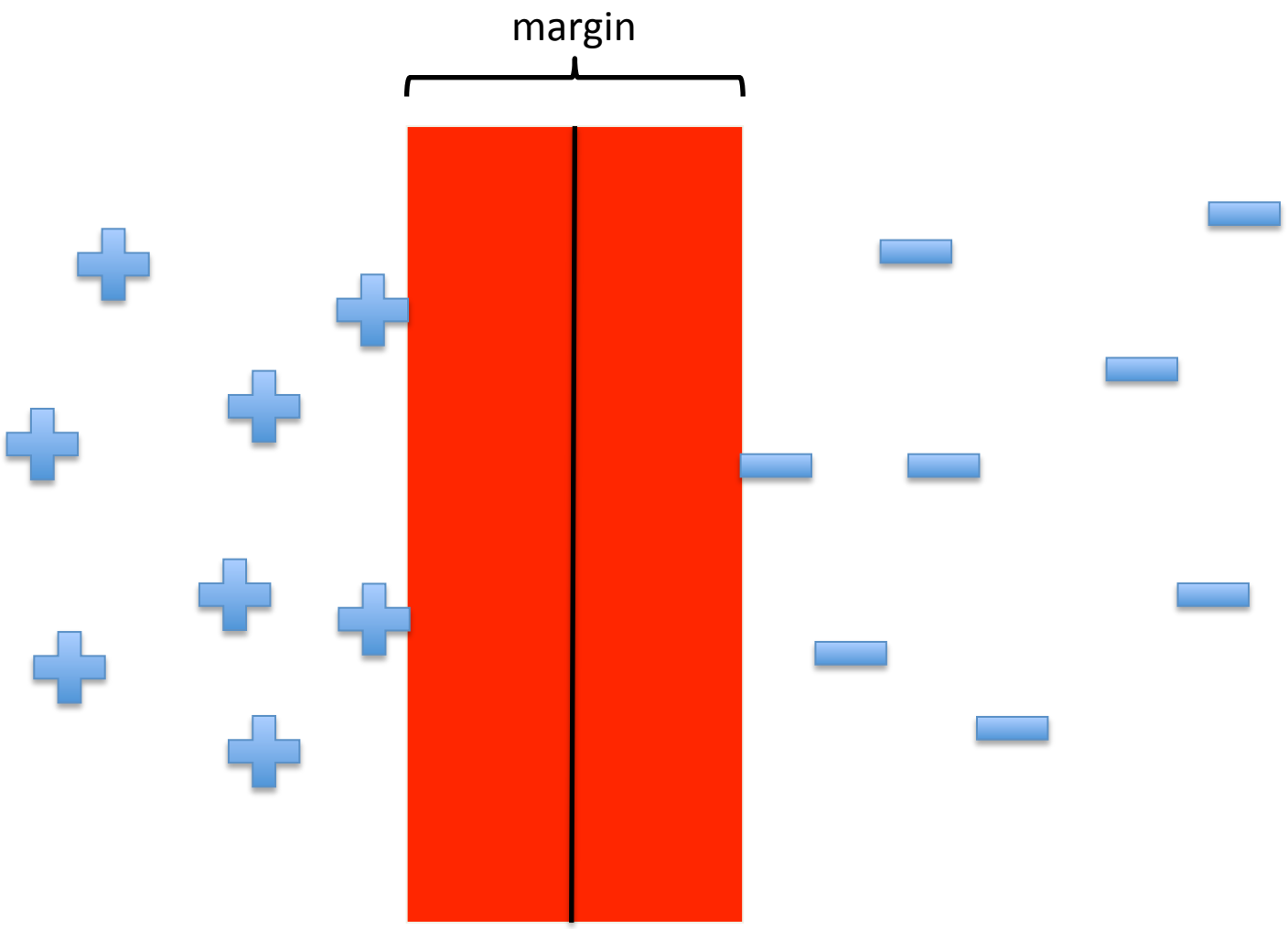
Maximizing the Margin



“Fat” Separators



“Fat” Separators



Why Maximize Margin

Increasing margin reduces *capacity*

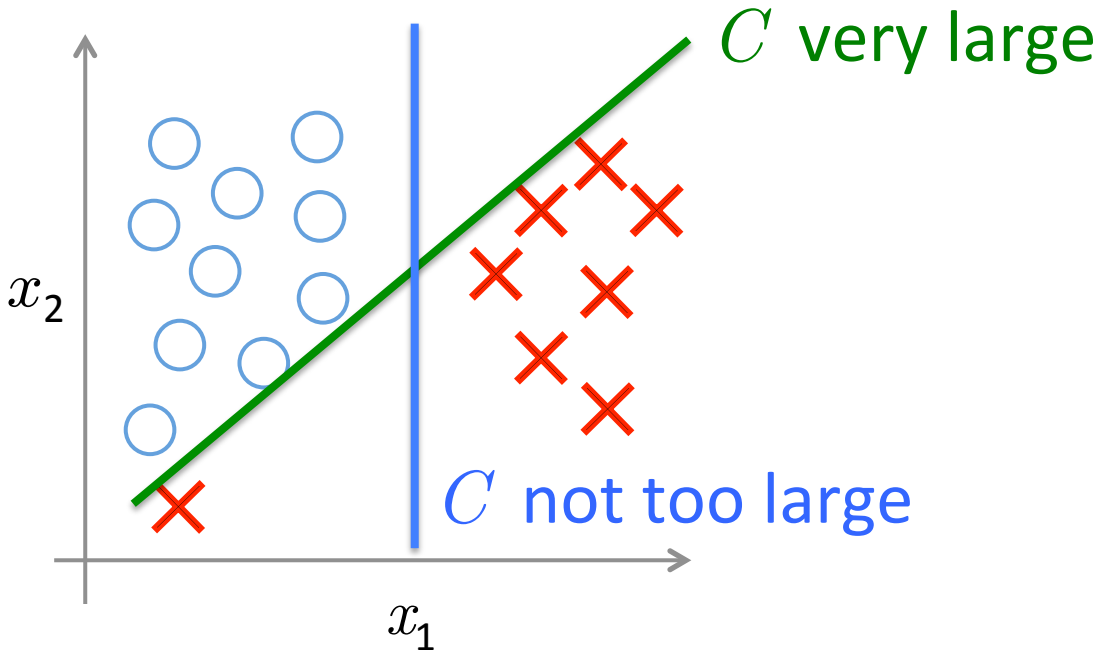
- i.e., fewer possible models

Lesson from Learning Theory:

- If the following holds:
  - H is sufficiently constrained in size
  - and/or the size of the training data set n is large,

  then low training error is likely to be evidence of low generalization error

Large Margin Classifier in Presence of Outliers

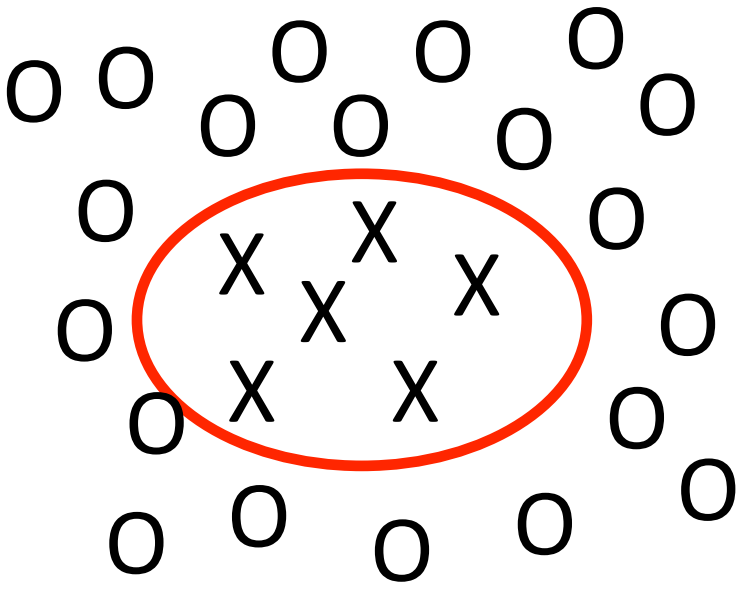


Based on slide by Andrew Ng

Strengths of SVMs

- Good generalization in theory
- Good generalization in practice
- Work well with few training instances
- Find globally best model
- Efficient algorithms
- Amenable to the kernel trick …

What if Surface is Non-Linear?



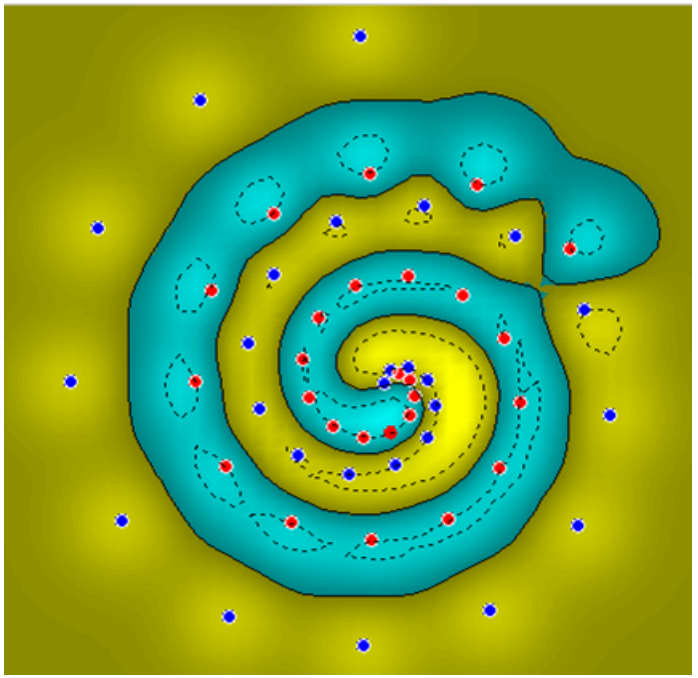
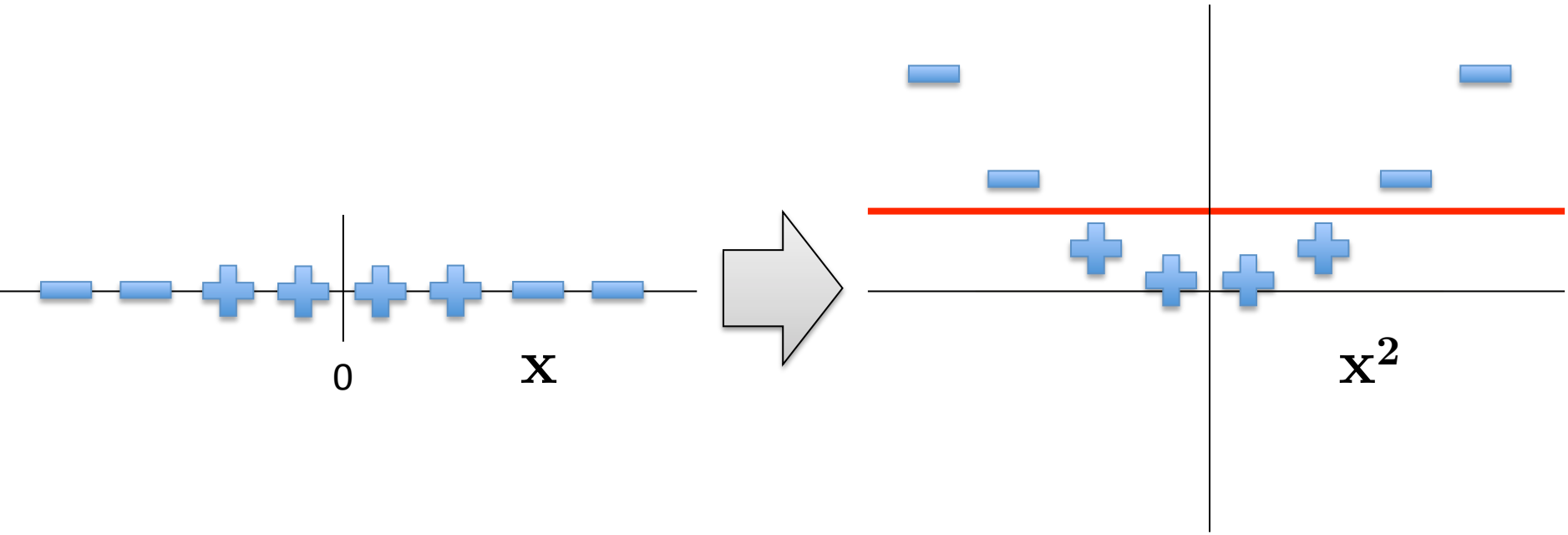


Image from http://www.atrandomresearch.com/iclass/

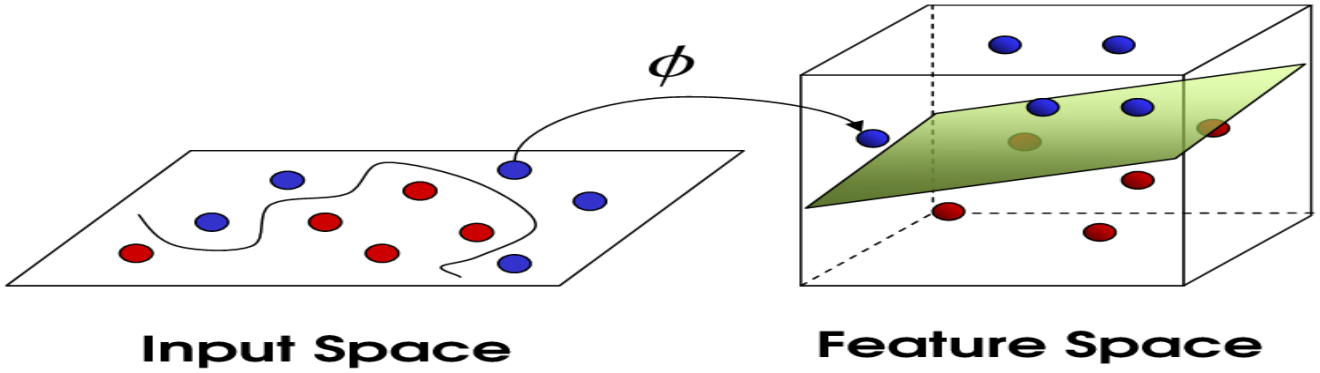
Kernel Methods

Making the Non-Linear Linear

When Linear Separators Fail



Mapping into a New Feature Space



$$\Phi : \mathcal{X} \mapsto \hat{\mathcal{X}} = \Phi(\mathbf{x})$$

- For example, with $\mathbf{x}_i \in \mathbb{R}^2$

$$\Phi([x_{i1}, x_{i2}]) = [x_{i1}, x_{i2}, x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2]$$

- Rather than run SVM on $\mathbf{x}_i$, run it on $\Phi(\mathbf{x}_i)$
  - Find non-linear separator in input space
- What if $\Phi(\mathbf{x}_i)$ is really big?
- Use kernels to compute it implicitly!

Image from http://web.engr.oregonstate.edu/ ~afern/classes/cs534/

Kernels

- Find kernel K such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

- Computing $K(\mathbf{x}_i, \mathbf{x}_j)$ should be efficient, much more so than computing $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$

- Use $K(\mathbf{x}_i, \mathbf{x}_j)$ in SVM algorithm rather than $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$
- Remarkably, this is possible!

The Polynomial Kernel

Let $\mathbf{x}_i = [x_{i1}, x_{i2}]$ and $\mathbf{x}_j = [x_{j1}, x_{j2}]$

Consider the following function:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= \langle \mathbf{x}_i, \mathbf{x}_j \rangle^2 \\ &= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\ &= (x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2}) \\ &= \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \end{aligned}$$

where

$$\Phi(\mathbf{x}_i) = [x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}]$$

$$\Phi(\mathbf{x}_j) = [x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2}]$$

The Polynomial Kernel

- Given by $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^d$
  - $\Phi(\mathbf{x})$ contains all monomials of degree d
- Useful in visual pattern recognition
  - Example:
    - 16x16 pixel image
    - 10^{10} monomials of degree 5
    - Never explicitly compute $\Phi(\mathbf{x})$!
- Variation: $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$
  - Adds all lower-order monomials (degrees 1,…, d)!

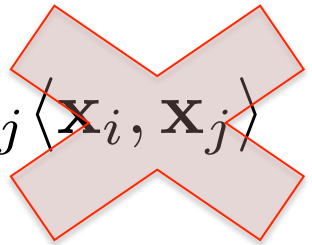
The Kernel Trick

"Given an algorithm which is formulated in terms of a positive definite kernel K_1, one can construct an alternative algorithm by replacing K_1 with another positive definite kernel K_2"

- SVMs can use the kernel trick

Incorporating Kernels into SVM

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$



$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{s.t. } a_i \geq 0 \quad \forall i$$

$$\sum_i \alpha_i y_i = 0$$

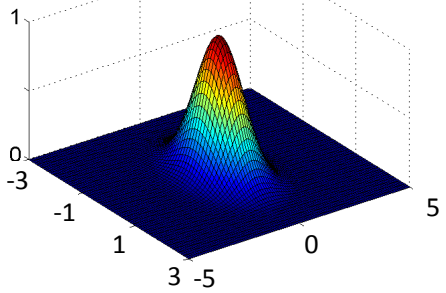
The Gaussian Kernel

- Also called Radial Basis Function (RBF) kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

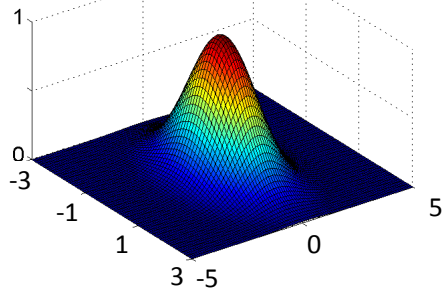
  - Has value 1 when $\mathbf{x}_i = \mathbf{x}_j$
  - Value falls off to 0 with increasing distance
  - Note: Need to do feature scaling before using Gaussian Kernel

$\sigma^2 = 0.5$

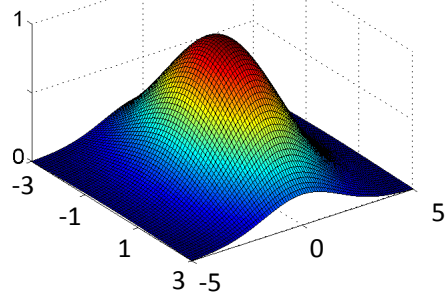


lower bias, higher variance

$\sigma^2 = 1$

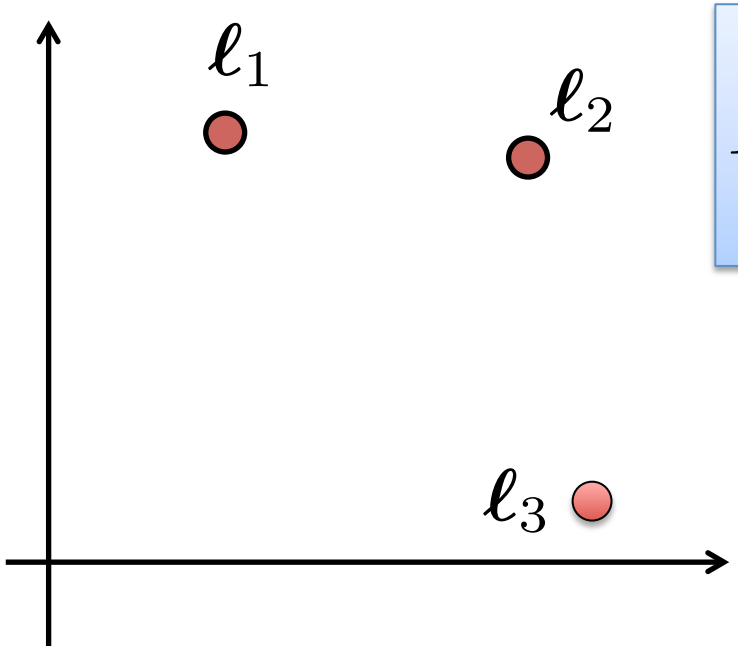


$\sigma^2 = 3$



higher bias, lower variance

Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

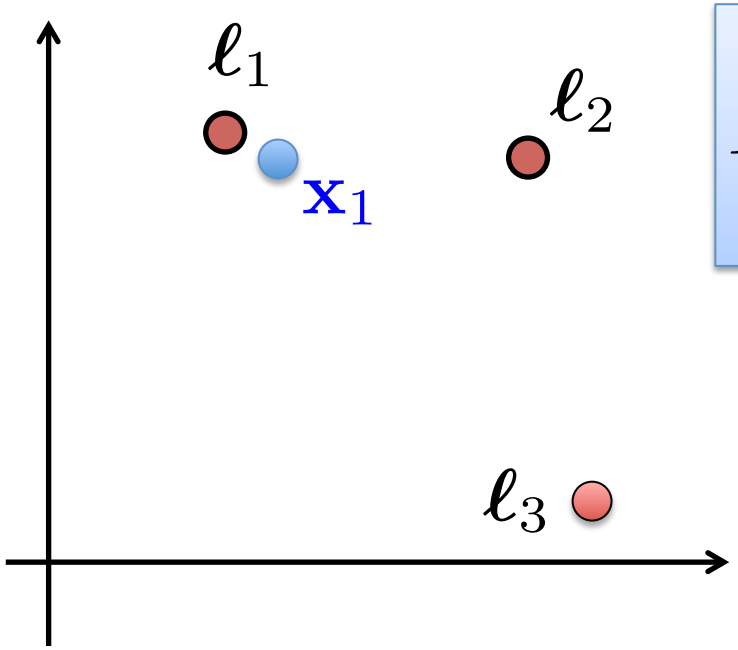
Imagine we've learned that:

$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \boldsymbol{\ell}_1) + \theta_2 K(\mathbf{x}, \boldsymbol{\ell}_2) + \theta_3 K(\mathbf{x}, \boldsymbol{\ell}_3) \geq 0$

Based on example by Andrew Ng

Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

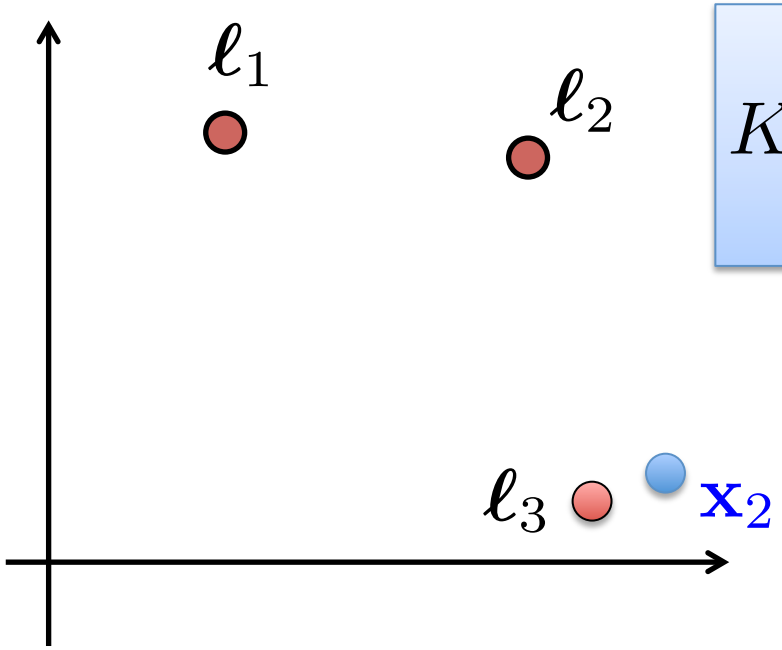
Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \ell_1) + \theta_2 K(\mathbf{x}, \ell_2) + \theta_3 K(\mathbf{x}, \ell_3) \geq 0$

- For $\mathbf{x}_1$, we have $K(\mathbf{x}_1, \ell_1) \approx 1$, other similarities ≈ 0

$$\begin{aligned} &\theta_0 + \theta_1(1) + \theta_2(0) + \theta_3(0) \\ &= -0.5 + 1(1) + 1(0) + 0(0) \\ &= 0.5 \geq 0 \text{ , so predict +1} \end{aligned}$$

Based on example by Andrew Ng

Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

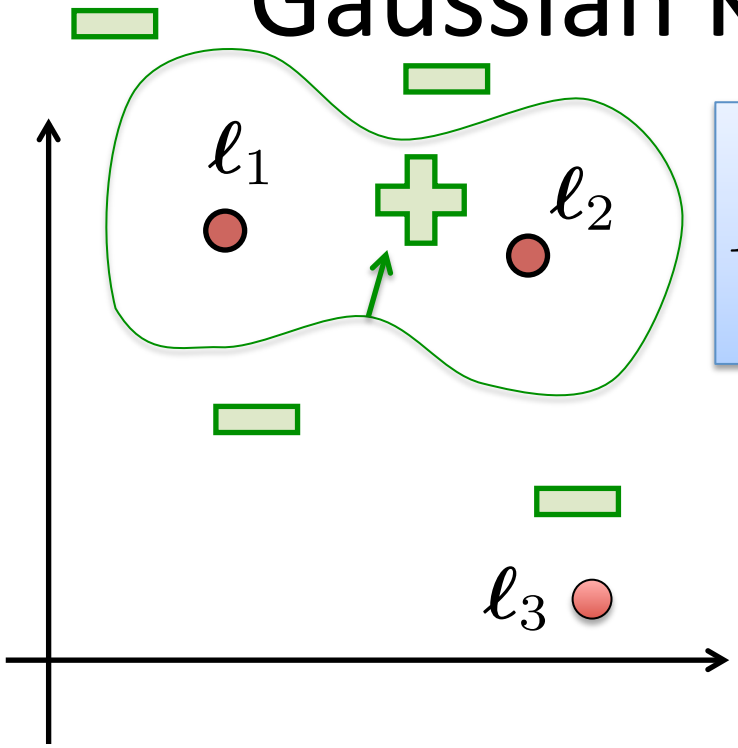
Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \ell_1) + \theta_2 K(\mathbf{x}, \ell_2) + \theta_3 K(\mathbf{x}, \ell_3) \geq 0$

- For $\mathbf{x}_2$, we have $K(\mathbf{x}_2, \ell_3) \approx 1$, other similarities ≈ 0

$$\begin{aligned} &\theta_0 + \theta_1(0) + \theta_2(0) + \theta_3(1) \\ &= -0.5 + 1(0) + 1(0) + 0(1) \\ &= -0.5 < 0 \text{ , so predict -1} \end{aligned}$$

Based on example by Andrew Ng

Gaussian Kernel Example



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \boldsymbol{\ell}_1) + \theta_2 K(\mathbf{x}, \boldsymbol{\ell}_2) + \theta_3 K(\mathbf{x}, \boldsymbol{\ell}_3) \geq 0$

Rough sketch of decision surface

Other Kernels

- Sigmoid Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\alpha \mathbf{x}_i^\intercal \mathbf{x}_j + c\right)$$

  - Neural networks use sigmoid as activation function
  - SVM with a sigmoid kernel is equivalent to 2-layer perceptron

- Cosine Similarity Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^\intercal \mathbf{x}_j}{\|\mathbf{x}_i\| \ \|\mathbf{x}_j\|}$$

  - Popular choice for measuring similarity of text documents
  - L_2 norm projects vectors onto the unit sphere; their dot product is the cosine of the angle between the vectors

An Aside: The Math Behind Kernels

What does it *mean* to be a kernel?

- $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ for some Φ

What does it *take* to be a kernel?

- The Gram matrix $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - Symmetric matrix
  - Positive semi-definite matrix:

    $\mathbf{z}^{\mathsf{T}}\mathrm{G}\mathbf{z} \geq 0$ for every non-zero vector $\mathbf{z} \in \mathbb{R}^n$

Establishing "kernel-hood" from first principles is non-trivial

A Few Good Kernels...

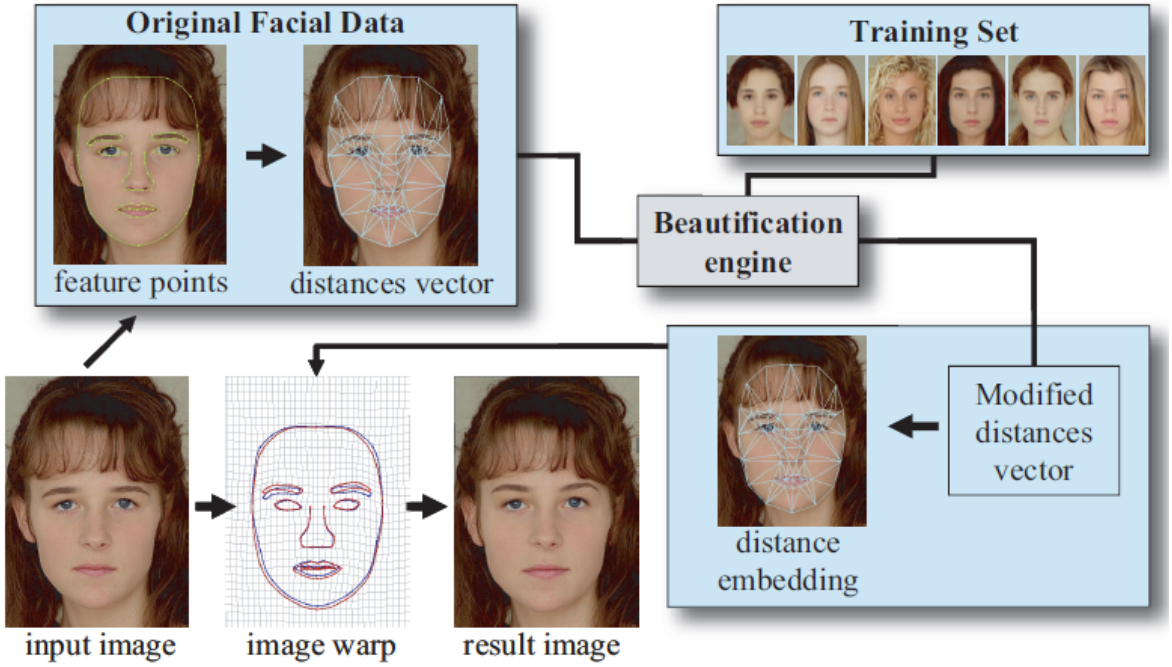
- Linear Kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$
- Polynomial kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^d$
  - c ≥ 0 trades off influence of lower order terms
- Gaussian kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$
- Sigmoid kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\alpha \mathbf{x}_i^\intercal \mathbf{x}_j + c\right)$

Many more...

- Cosine similarity kernel
- Chi-squared kernel
- String/tree/graph/wavelet/etc kernels

Application: Automatic Photo Retouching

(Leyvand et al., 2008)



Practical Advice for Applying SVMs

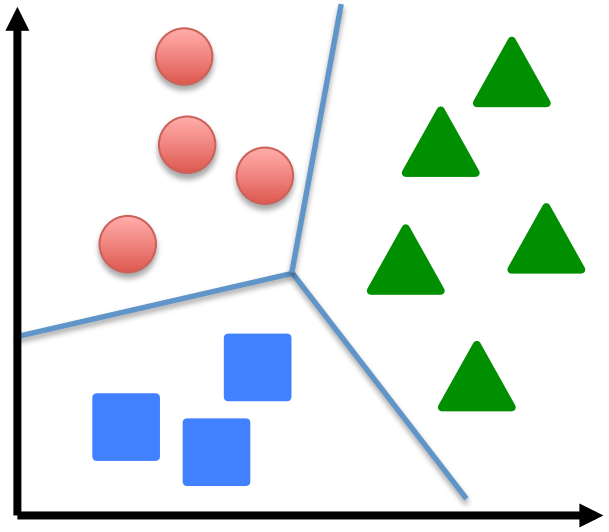
- Use SVM software package to solve for parameters
  - e.g., SVMlight, libsvm, cvx (fast!), etc.

- Need to specify:
  - Choice of parameter C
  - Choice of kernel function
    - Associated kernel parameters

e.g., $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^d$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Multi-Class Classification with SVMs



$$y \in \{1, \ldots, K\}$$

- Many SVM packages already have multi-class classification built in
- Otherwise, use one-vs-rest
  - Train K SVMs, each picks out one class from rest, yielding $\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(K)}$
  - Predict class i with largest $(\boldsymbol{\theta}^{(i)})^{\mathsf{T}}\mathbf{x}$

Based on slide by Andrew Ng

Conclusion

- SVMs find optimal linear separator
- The kernel trick makes SVMs learn non-linear decision surfaces

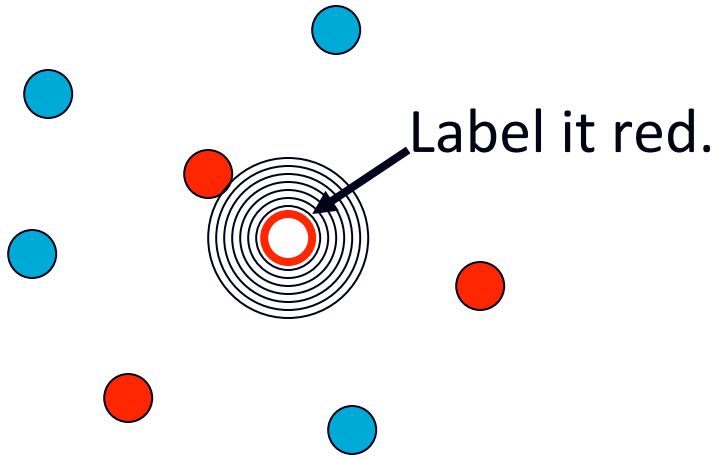
- Strength of SVMs:
  - Good theoretical and empirical performance
  - Supports many types of kernels

- Disadvantages of SVMs:
  - "Slow" to train/predict for huge data sets (but relatively fast!)
  - Need to choose the kernel (and tune its parameters)

k-Nearest Neighbor & Instance-based Learning

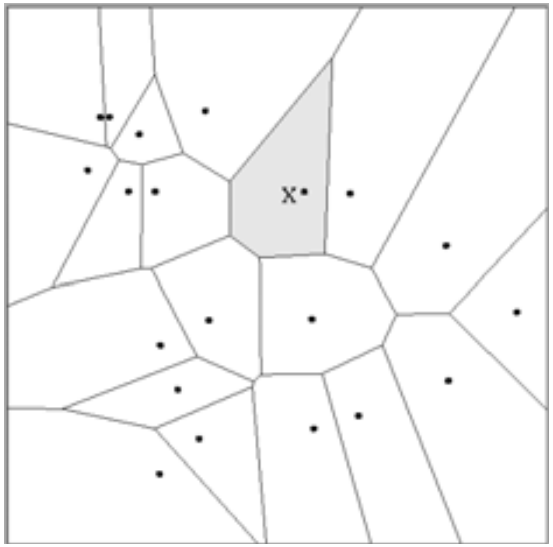
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point



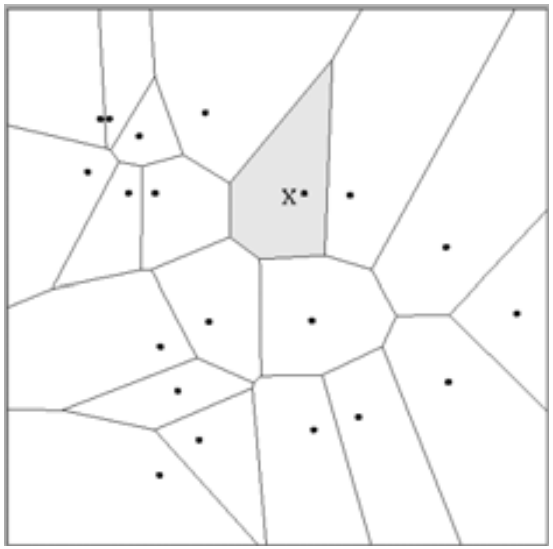
1-Nearest Neighbor

- A type of instance-based learning
  - Also known as "memory-based" learning
- Forms a Voronoi tessellation of the instance space

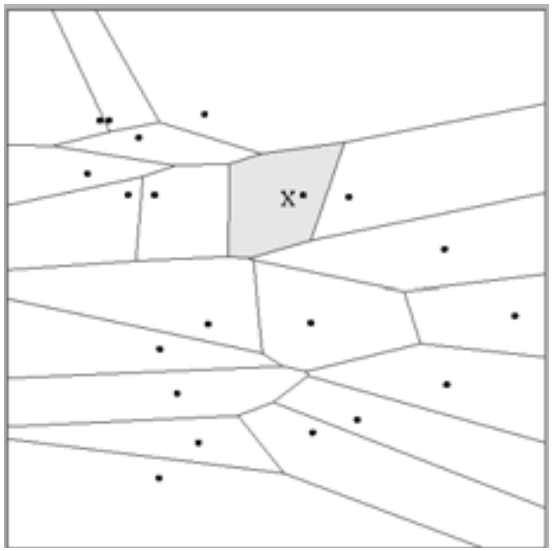


Distance Metrics

- Different metrics can change the decision surface



Dist(**a,b**) =$(a_1 - b_1)^2 + (a_2 - b_2)^2$



Dist(**a,b**) =$(a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

- Standard Euclidean distance metric:
  - Two-dimensional: Dist(a,b) = sqrt($(a_1 - b_1)^2 + (a_2 - b_2)^2$)
  - Multivariate: Dist(a,b) = sqrt($\sum (a_i - b_i)^2$)

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

Four Aspects of an Instance-Based Learner:

1. A distance metric
2. How many nearby neighbors to look at?
3. A weighting function (optional)
4. How to fit with the local points?

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

Zen Gardens

Mystery of renowned zen garden revealed [CNN Article]
Thursday, September 26, 2002 Posted: 10:11 AM EDT (1411 GMT)

LONDON (Reuters) -- For centuries visitors to the renowned Ryoanji Temple garden in Kyoto, Japan have been entranced and mystified by the simple arrangement of rocks.

The five sparse clusters on a rectangle of raked gravel are said to be pleasing to the eyes of the hundreds of thousands of tourists who visit the garden each year.

Scientists in Japan said on Wednesday they now believe they have discovered its mysterious appeal.

"We have uncovered the implicit structure of the Ryoanji garden's visual ground and have shown that it includes an abstract, minimalist depiction of natural scenery," said Gert Van Tonder of Kyoto University.

The researchers discovered that the empty space of the garden evokes a hidden image of a branching tree that is sensed by the unconscious mind.

"We believe that the unconscious perception of this pattern contributes to the enigmatic appeal of the garden," Van Tonder added.

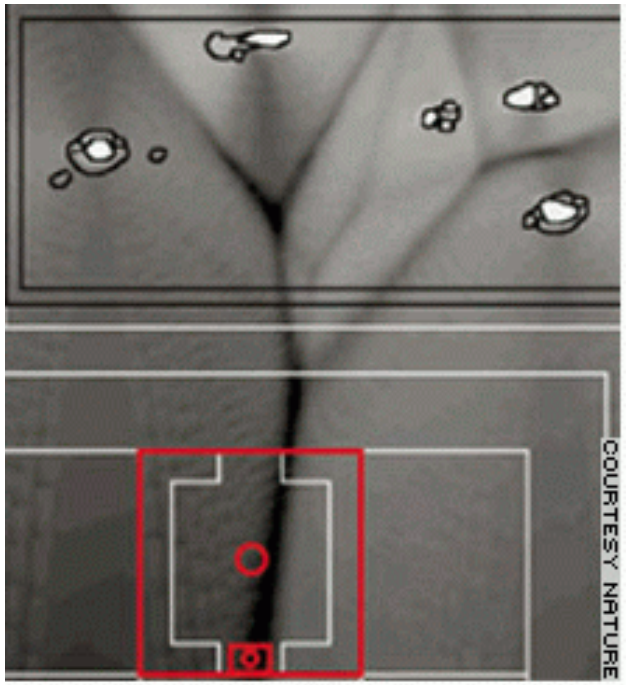
He and his colleagues believe that whoever created the garden during the Muromachi era between 1333-1573 knew exactly what they were doing and placed the rocks around the tree image.

By using a concept called medial-axis transformation, the scientists showed that the hidden branched tree converges on the main area from which the garden is viewed.

The trunk leads to the prime viewing site in the ancient temple that once overlooked the garden. It is thought that abstract art may have a similar impact.

"There is a growing realisation that scientific analysis can reveal unexpected structural features hidden in controversial abstract paintings," Van Tonder said

Ryoanji Temple garden in Kyoto

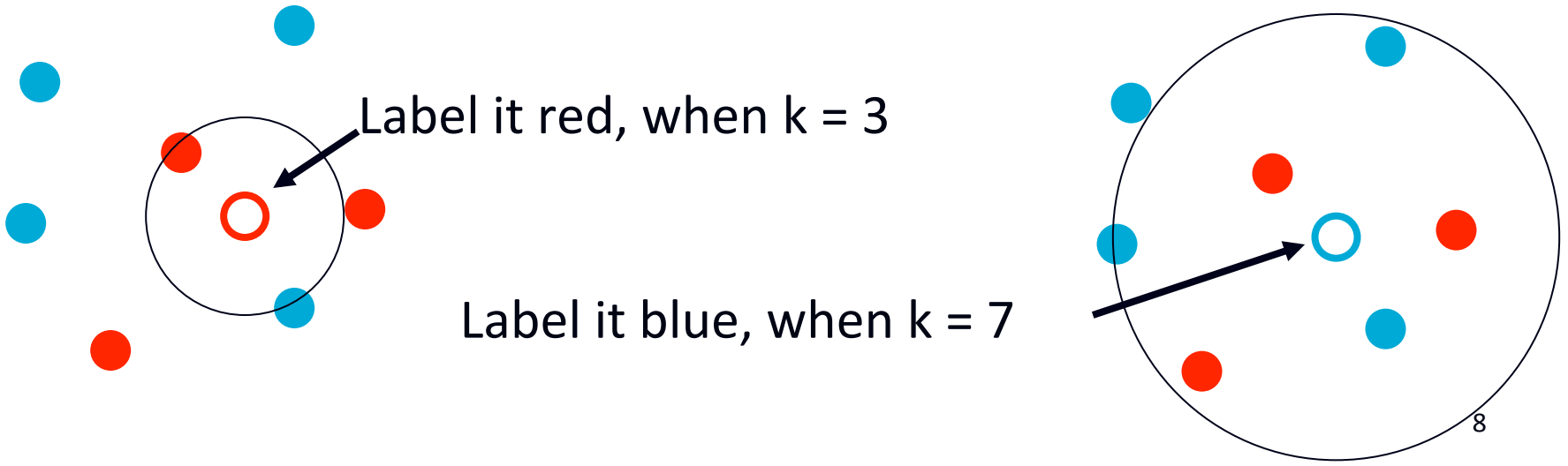


Layout shows the rock clusters (top) and the preferred viewing spot of the garden from the main hall (the circle in the middle of the square).

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

k – Nearest Neighbor

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its *k* nearest neighbors



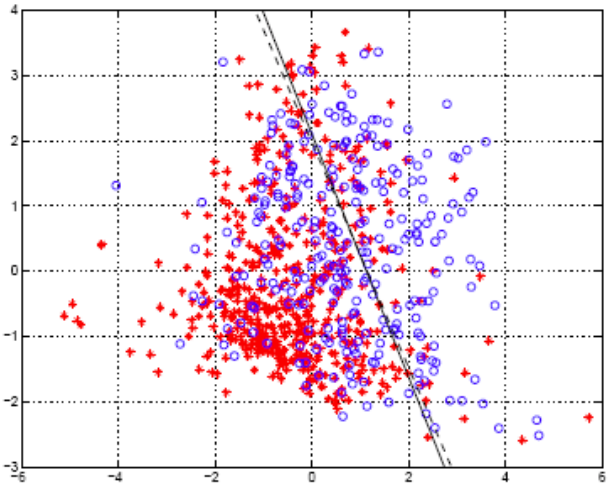
Logistic Regression

Classification Based on Probability

- Instead of just predicting the class, give the probability of the instance being that class
  - i.e., learn $p(y \mid \boldsymbol{x})$
- Comparison to perceptron:
  - Perceptron doesn't produce probability estimate
  - Perceptron (and other discriminative classifiers) are only interested in producing a discriminative model
- Recall that:

$$0 \leq p(\text{event}) \leq 1$$

$$p(\text{event}) + p(\neg\text{event}) = 1$$



Logistic Regression

- Takes a probabilistic approach to learning discriminative functions (i.e., a classifier)
- $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ should give $p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta})$
  - Want $0 \leq h_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq 1$

Can't just use linear regression with a threshold

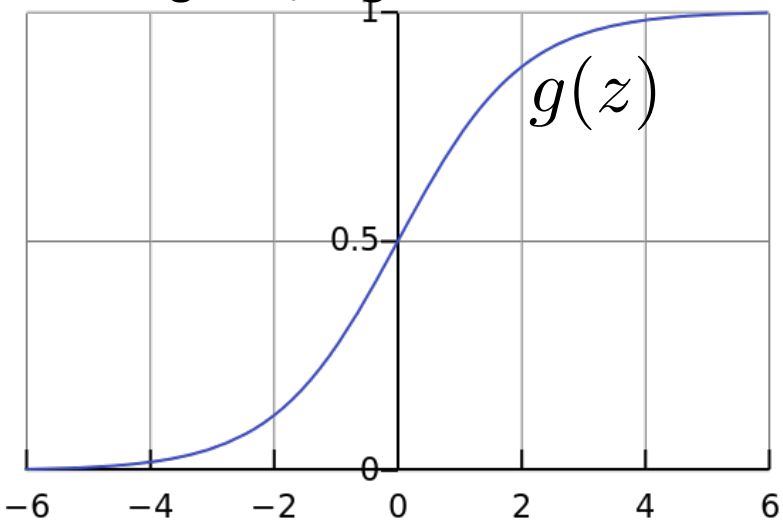
- Logistic regression model:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}\right)$$

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

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}}}$$

Logistic / Sigmoid Function



Interpretation of Hypothesis Output

$h_{\boldsymbol{\theta}}(\boldsymbol{x})$ = estimated $p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta})$

Example: Cancer diagnosis from tumor size

$$\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0.7$$

→ Tell patient that 70% chance of tumor being malignant

Note that: $p(y = 0 \mid \boldsymbol{x}; \boldsymbol{\theta}) + p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}) = 1$

Therefore, $p(y = 0 \mid \boldsymbol{x}; \boldsymbol{\theta}) = 1 - p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta})$

Another Interpretation

- Equivalently, logistic regression assumes that

$$\log \frac{p(y=1 \mid \boldsymbol{x}; \boldsymbol{\theta})}{p(y=0 \mid \boldsymbol{x}; \boldsymbol{\theta})} = \theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d$$

odds of y = 1

Side Note: the odds in favor of an event is the quantity p / (1 – p), where p is the probability of the event

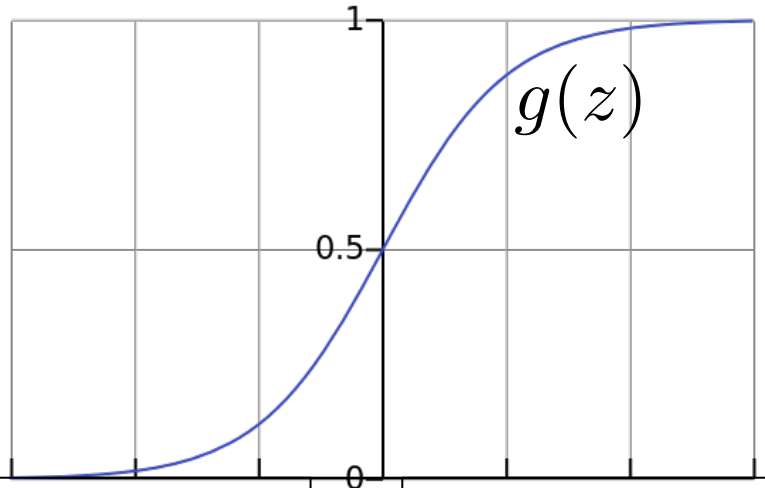
E.g., If I toss a fair dice, what are the odds that I will have a 6?

- In other words, logistic regression assumes that the log odds is a linear function of $\boldsymbol{x}$

Logistic Regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}\right)$$

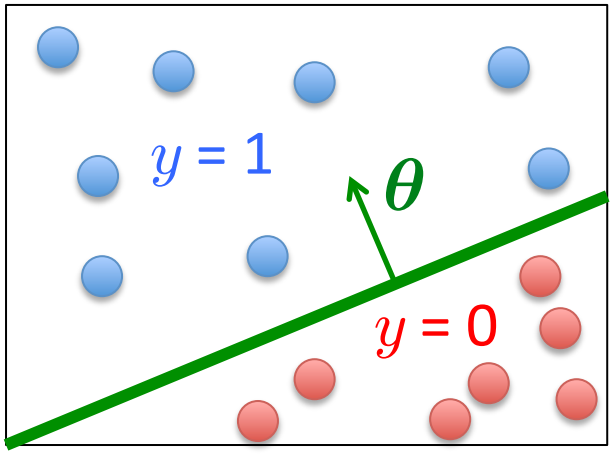
$$g(z) = \frac{1}{1+e^{-z}}$$



$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large <u>negative</u> values for negative instances

$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large <u>positive</u> values for positive instances

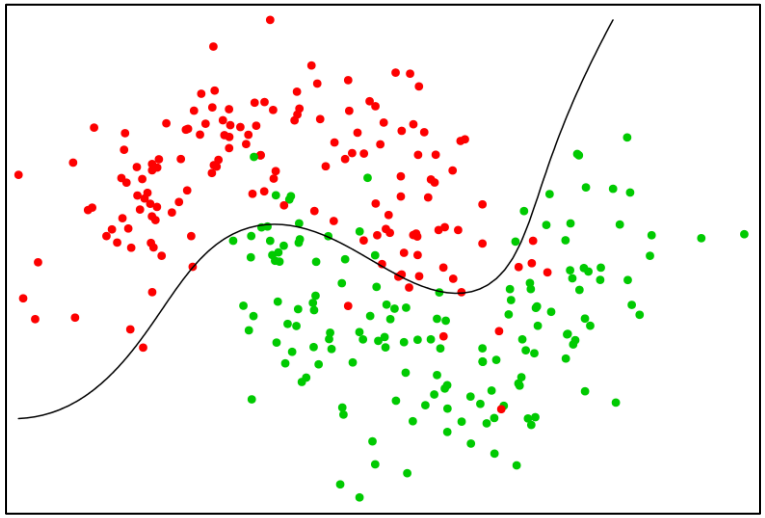
- Assume a threshold and...
  - Predict $y = 1$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0.5$
  - Predict $y = 0$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$



Non-Linear Decision Boundary

- Can apply basis function expansion to features, same as with linear regression

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \\ x_1^2 x_2 \\ x_1 x_2^2 \\ \vdots \end{bmatrix}$$



Logistic Regression

- Given $\left\{ \left(\boldsymbol{x}^{(1)}, y^{(1)} \right), \left(\boldsymbol{x}^{(2)}, y^{(2)} \right), \ldots, \left(\boldsymbol{x}^{(n)}, y^{(n)} \right) \right\}$

  where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d, \; y^{(i)} \in \{0, 1\}$

- Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^\intercal \boldsymbol{x}\right)$

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

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \ldots & x_d \end{bmatrix}$$

Logistic Regression Objective Function

- Can't just use squared loss as in linear regression:

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

  - Using the logistic regression model

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \boldsymbol{x}}}$$

    results in a non-convex optimization

Deriving the Cost Function via Maximum Likelihood Estimation

- Likelihood of data is given by: $l(\boldsymbol{\theta}) = \prod_{i=1}^{n} p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$

- So, looking for the $\boldsymbol{\theta}$ that maximizes the likelihood

$$\boldsymbol{\theta}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{n} p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

- Can take the log without changing the solution:

$$\boldsymbol{\theta}_{\mathrm{MLE}} = \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{n} p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

Deriving the Cost Function via Maximum Likelihood Estimation

- Expand as follows:

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p(y^{(i)} \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left[y^{(i)} \log p(y^{(i)} = 1 \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta}) + \left(1 - y^{(i)}\right) \log \left(1 - p(y^{(i)} = 1 \mid \boldsymbol{x}^{(i)}; \boldsymbol{\theta})\right) \right]$$

- Substitute in model, and take negative to yield

Logistic regression objective:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n} \left[y^{(i)} \log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1 - y^{(i)}\right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right) \right]$$

Intuition Behind the Objective

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)} \log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1 - y^{(i)}\right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\right]$$

- Cost of a single instance:

$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

- Can re-write objective function as

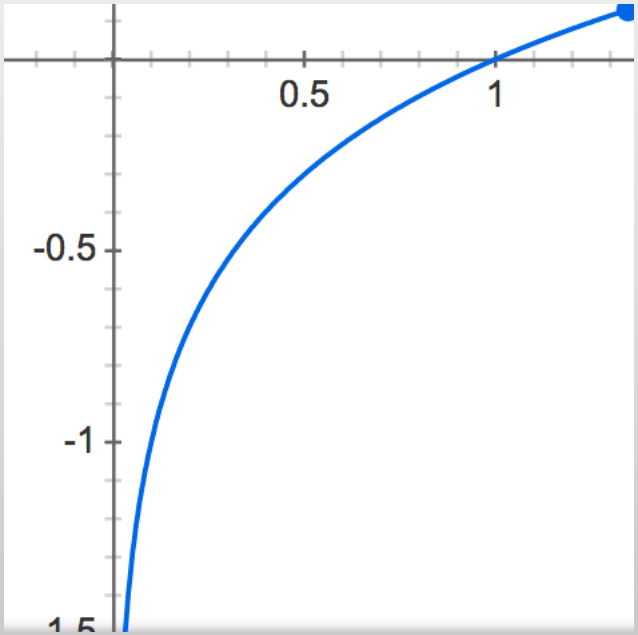
$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} \text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}), y^{(i)}\right)$$

Compare to linear regression: $J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$

Intuition Behind the Objective

$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

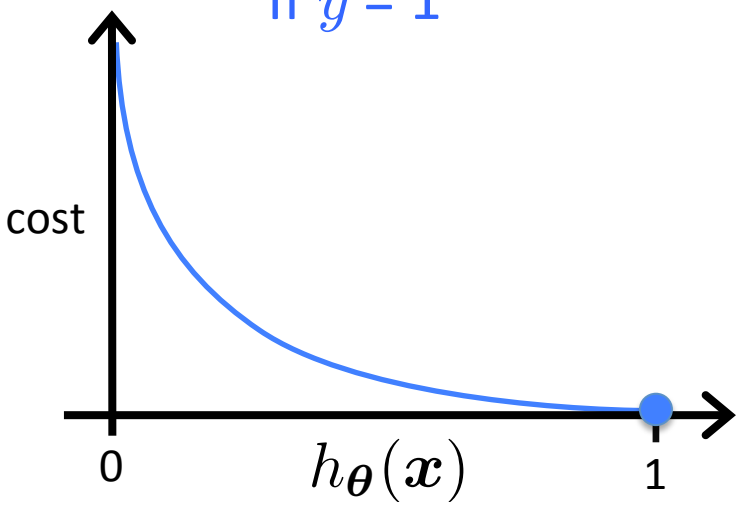
Aside: Recall the plot of $\log(z)$



Intuition Behind the Objective

$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$

If y = 1



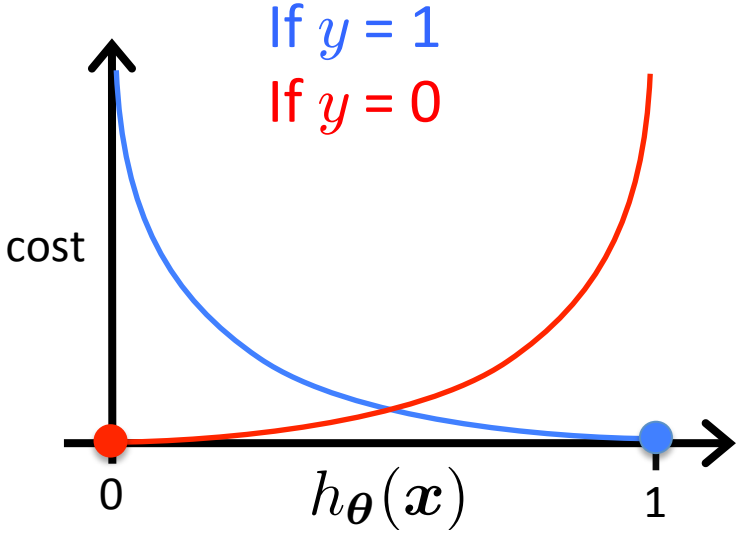
If y = 1

- Cost = 0 if prediction is correct
- As $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \to 0, \text{cost} \to \infty$
- Captures intuition that larger mistakes should get larger penalties
  - e.g., predict $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0$, but y = 1

Based on example by Andrew Ng

Intuition Behind the Objective

$$\text{cost}\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y\right) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if } y = 0 \end{cases}$$



If y = 0

- Cost = 0 if prediction is correct
- As $(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) \to 0, \text{cost} \to \infty$
- Captures intuition that larger mistakes should get larger penalties

Based on example by Andrew Ng

Regularized Logistic Regression

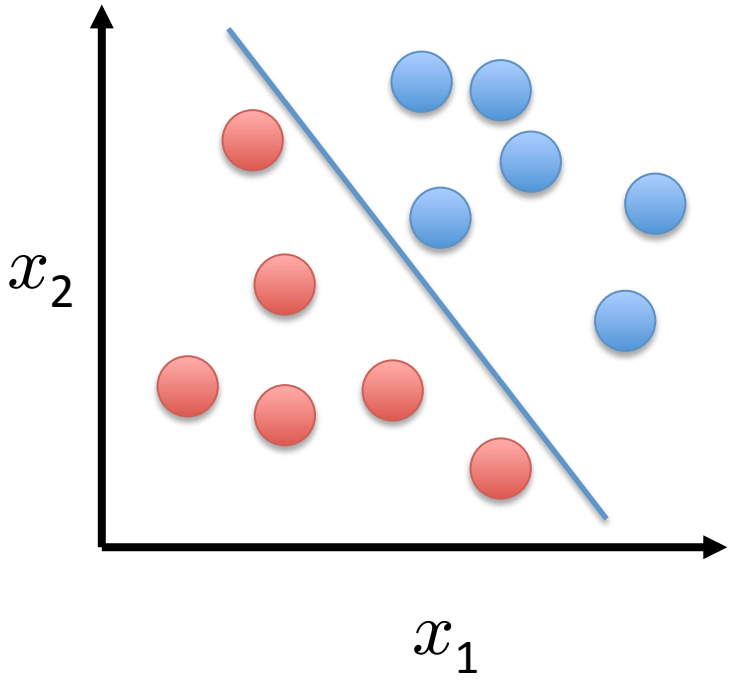
$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{n}\left[y^{(i)}\log h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \left(1-y^{(i)}\right)\log\left(1-h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})\right)\right]$$

- We can regularize logistic regression exactly as before:

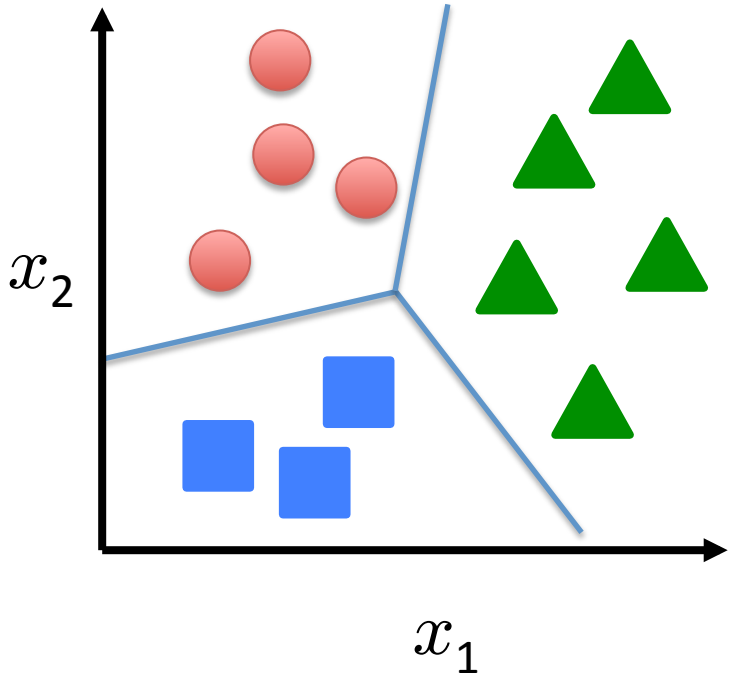
$$\begin{aligned} J_{\text{regularized}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2 \\ &= J(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}_{[1:d]}\|_2^2 \end{aligned}$$

Multi-Class Classification

Binary classification:



Multi-class classification:



Disease diagnosis: healthy / cold / flu / pneumonia

Object classification: desk / chair / monitor / bookcase

Multi-Class Logistic Regression

- For 2 classes:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x})} = \frac{\exp(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x})}{1+\exp(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x})}$$

weight assigned to $y = 0$

weight assigned to $y = 1$

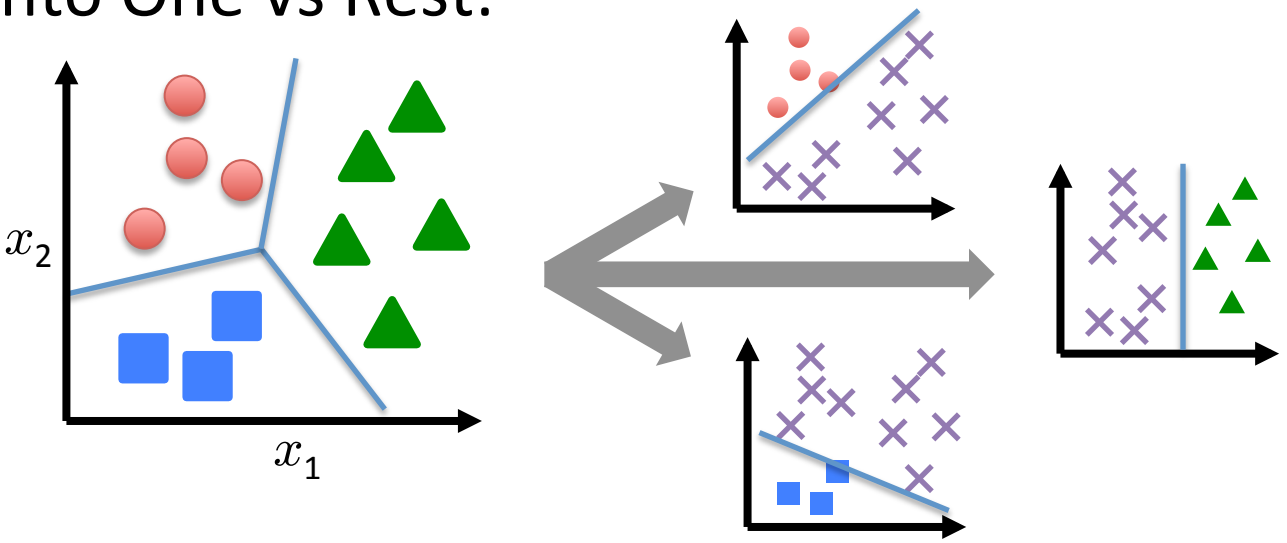
- For C classes {1, ..., C}:

$$p(y = c \mid \boldsymbol{x}; \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_C) = \frac{\exp(\boldsymbol{\theta}_c^{\mathsf{T}}\boldsymbol{x})}{\sum_{c=1}^{C}\exp(\boldsymbol{\theta}_c^{\mathsf{T}}\boldsymbol{x})}$$

  - Called the **softmax** function

Multi-Class Logistic Regression

Split into One vs Rest:



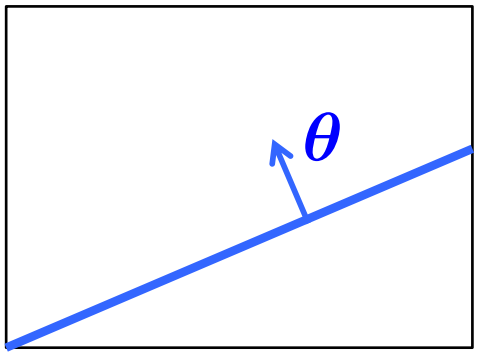
- Train a logistic regression classifier for each class i to predict the probability that $y = i$ with

$$h_c(\boldsymbol{x}) = \frac{\exp(\boldsymbol{\theta}_c^\mathsf{T}\boldsymbol{x})}{\sum_{c=1}^{C}\exp(\boldsymbol{\theta}_c^\mathsf{T}\boldsymbol{x})}$$

Linear Classification: The Perceptron

Linear Classifiers

- A **hyperplane** partitions $\mathbb{R}^d$ into two half-spaces
  - Defined by the normal vector $\boldsymbol{\theta} \in \mathbb{R}^d$
    - $\boldsymbol{\theta}$ is orthogonal to any vector lying on the hyperplane



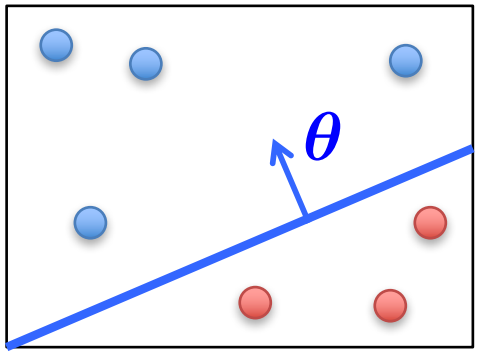
  - Assumed to pass through the origin
    - This is because we incorporated bias term θ_0 into it by $x_0 = 1$

- Consider classification with +1, -1 labels ...

Linear Classifiers

- **Linear classifiers**: represent decision boundary by hyperplane

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$



$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

– Note that: $\boldsymbol{\theta}^\intercal \boldsymbol{x} > 0 \implies y = +1$

$\boldsymbol{\theta}^\intercal \boldsymbol{x} < 0 \implies y = -1$

The Perceptron

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\mathsf{T}\boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{2}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

either 2 or -2

  - If the prediction matches the label, make no change
  - Otherwise, adjust $\boldsymbol{\theta}$

The Perceptron

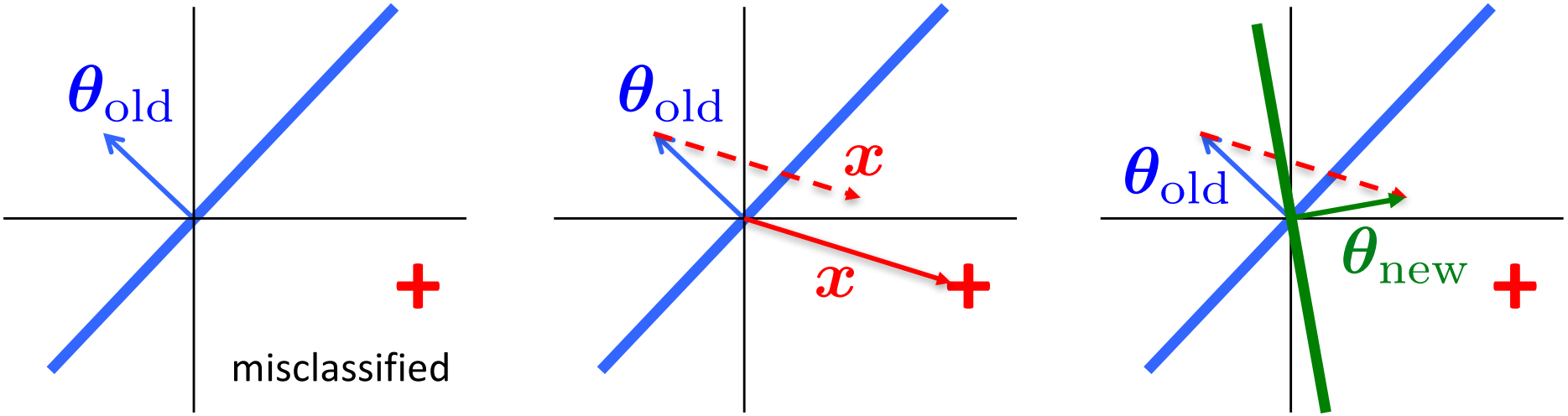
- The perceptron uses the following update rule each time it receives a new training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{2}\underbrace{\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)}_{\text{either 2 or -2}} x_j^{(i)}$$

- Re-write as $\theta_j \leftarrow \theta_j + \alpha y^{(i)} x_j^{(i)}$ (only upon misclassification)
  - Can eliminate α in this case, since its only effect is to scale $\boldsymbol{\theta}$ by a constant, which doesn't affect performance

Perceptron Rule: If $\boldsymbol{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)}\boldsymbol{x}^{(i)}$

Why the Perceptron Update Works



Why the Perceptron Update Works

- Consider the misclassified example (y = +1)
  - Perceptron wrongly thinks that $\boldsymbol{\theta}_{\text{old}}^{\mathsf{T}}\boldsymbol{x} < 0$
- Update:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + y\boldsymbol{x} = \boldsymbol{\theta}_{\text{old}} + \boldsymbol{x} \qquad (\text{since } y = +1)$$

- Note that

$$\begin{aligned}\boldsymbol{\theta}_{\text{new}}^{\mathsf{T}}\boldsymbol{x} &= (\boldsymbol{\theta}_{\text{old}} + \boldsymbol{x})^{\mathsf{T}}\boldsymbol{x} \\ &= \boldsymbol{\theta}_{\text{old}}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{x}^{\mathsf{T}}\boldsymbol{x}\end{aligned}$$

$\|\boldsymbol{x}\|_2^2 > 0$

- Therefore, $\boldsymbol{\theta}_{\text{new}}^{\mathsf{T}}\boldsymbol{x}$ is less negative than $\boldsymbol{\theta}_{\text{old}}^{\mathsf{T}}\boldsymbol{x}$
  - So, we are making ourselves more correct on this example!

Online Perceptron Algorithm

Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
    Receive training example $(\boldsymbol{x}^{(i)}, y^{(i)})$
    if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} \leq 0$ // prediction is incorrect
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)}\boldsymbol{x}^{(i)}$

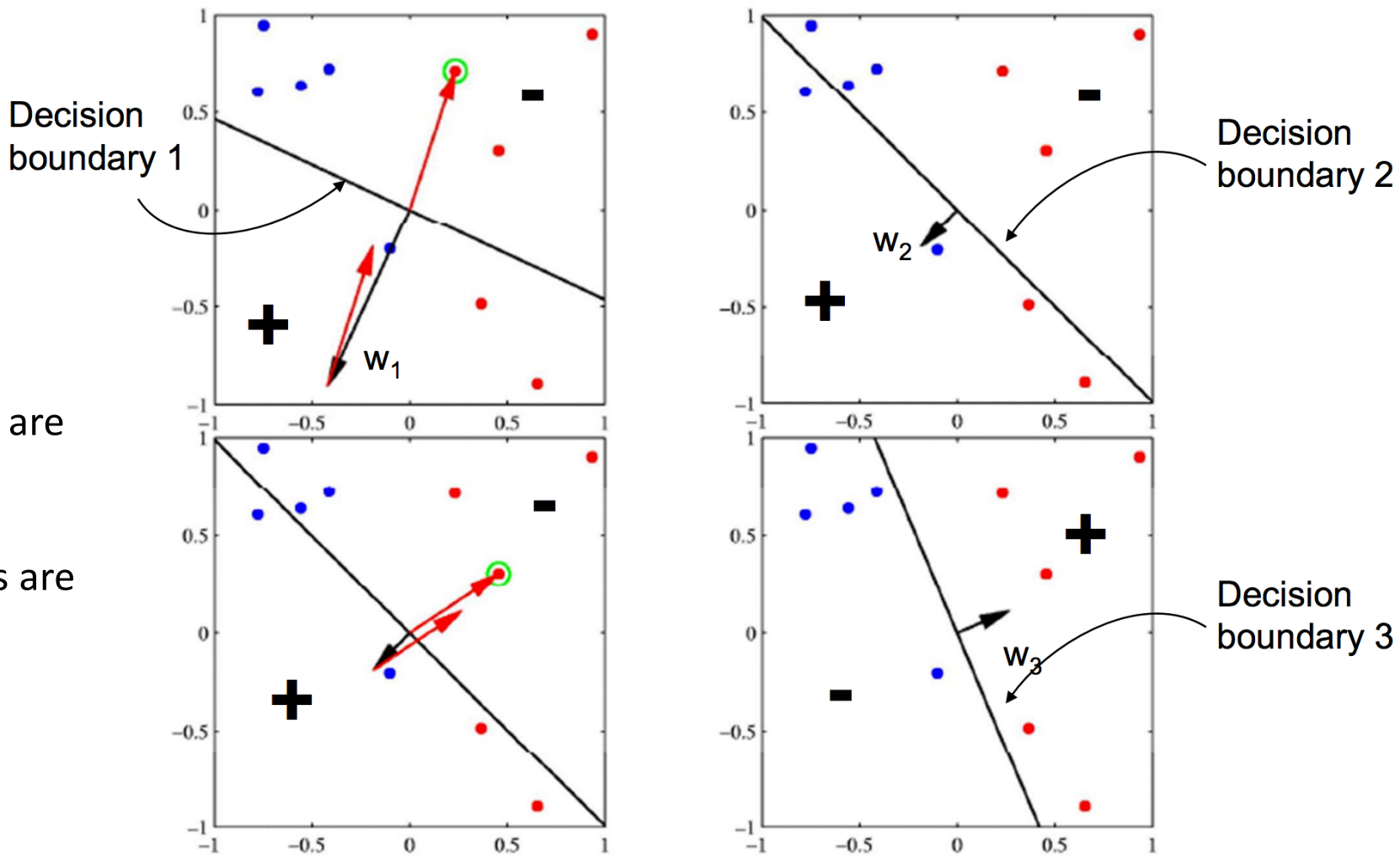
Online learning – the learning mode where the model update is performed each time a single observation is received

Batch learning – the learning mode where the model update is performed after observing the entire training set

Based on slide by Alan Fern

Online Perceptron Algorithm

When an error is made, moves the weight in a direction that corrects the error



Red points are labeled +

Blue points are labeled -

See the perceptron in action: www.youtube.com/watch?v=vGwemZhPlsA

Based on slide by Alan Fern

Batch Perceptron

Given training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
Let $\boldsymbol{\theta} \leftarrow [0, 0, \ldots, 0]$
Repeat:
    Let $\boldsymbol{\Delta} \leftarrow [0, 0, \ldots, 0]$
    for $i = 1 \ldots n$, do
        if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} \leq 0$ // prediction for ith instance is incorrect
            $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta} + y^{(i)}\boldsymbol{x}^{(i)}$
    $\boldsymbol{\Delta} \leftarrow \boldsymbol{\Delta}/n$ // compute average update
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\boldsymbol{\Delta}$
Until $\|\boldsymbol{\Delta}\|_2 < \epsilon$

- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Guaranteed to find a separating hyperplane if one exists

Improving the Perceptron

- The Perceptron produces many θ's during training
- The standard Perceptron simply uses the final θ at test time
  - This may sometimes not be a good idea!
  - Some other θ may be correct on 1,000 consecutive examples, but one mistake ruins it!

- **Idea:** Use a combination of multiple perceptrons
  - (i.e., neural networks!)
- **Idea:** Use the intermediate θ's
  - **Voted Perceptron**: vote on predictions of the intermediate θ's
  - **Averaged Perceptron**: average the intermediate θ's

Based on slide by Piyush Rai

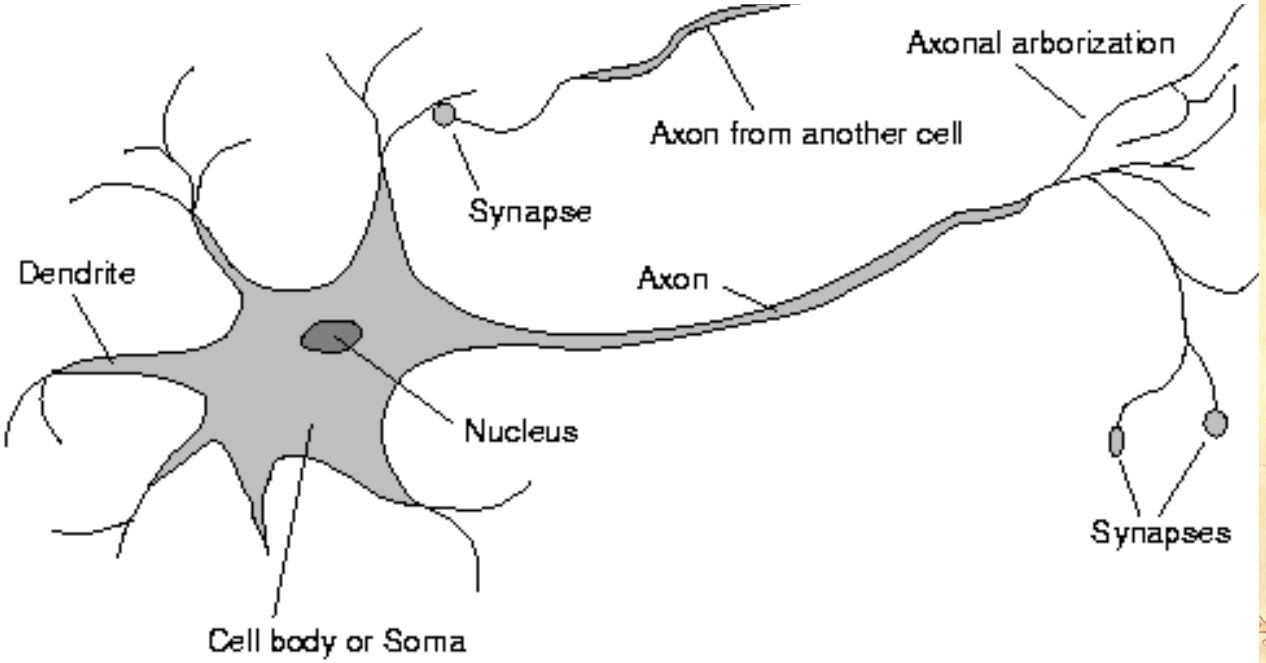
Neural Networks

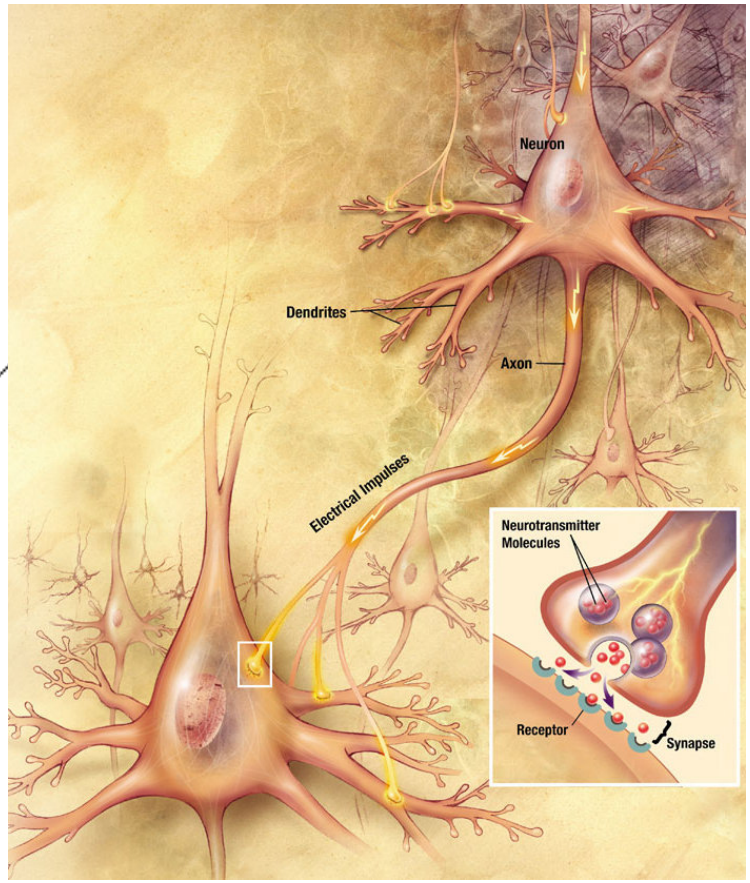
Neural Function

- Brain function (thought) occurs as the result of the firing of **neurons**
- Neurons connect to each other through **synapses**, which propagate **action potential** (electrical impulses) by releasing **neurotransmitters**
  - Synapses can be **excitatory** (potential-increasing) or **inhibitory** (potential-decreasing), and have varying **activation thresholds**
  - Learning occurs as a result of the synapses' **plasticicity**: They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain!

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Biology of a Neuron



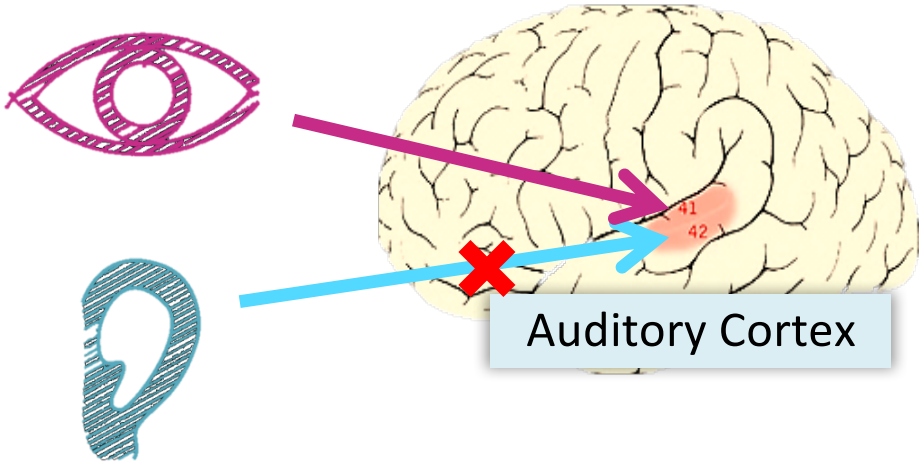


Brain Structure

- Different areas of the brain have different functions
  - Some areas seem to have the same function in all humans (e.g., Broca's region for motor speech); the overall layout is generally consistent
  - Some areas are more plastic, and vary in their function; also, the lower-level structure and function vary greatly
- We don't know how different functions are "assigned" or acquired
  - Partly the result of the physical layout / connection to inputs (sensors) and outputs (effectors)
  - Partly the result of experience (learning)
- We *really* don't understand how this neural structure leads to what we perceive as "consciousness" or "thought"

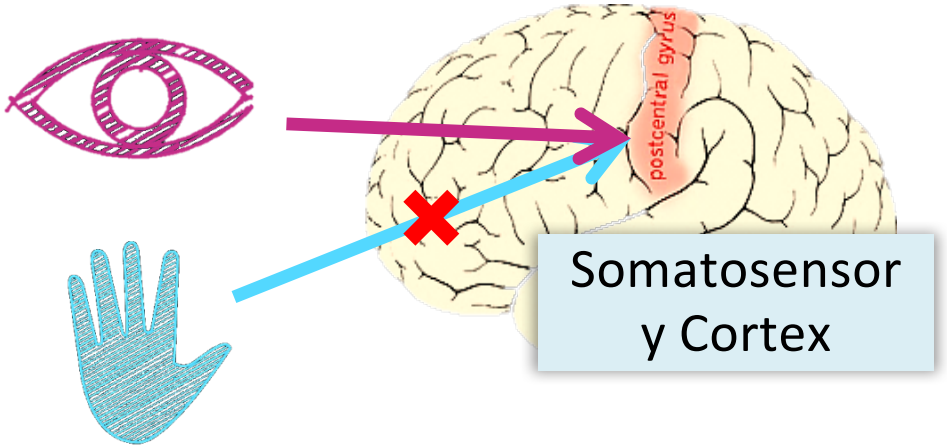
Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

The "One Learning Algorithm" Hypothesis



Auditory cortex learns to see

[Roe et al., 1992]

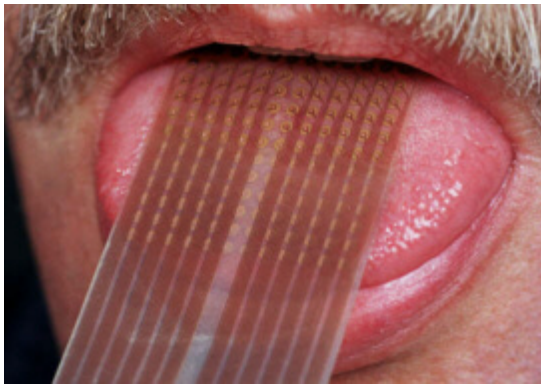


Somatosensory cortex learns to see

[Metin & Frost, 1989]

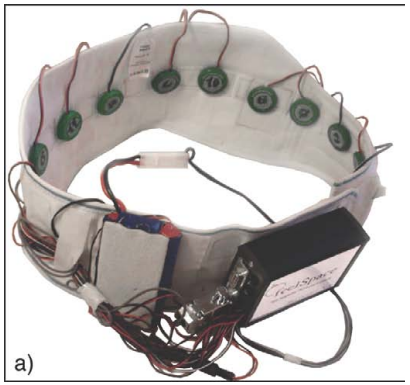
Based on slide by Andrew Ng

Sensor Representations in the Brain

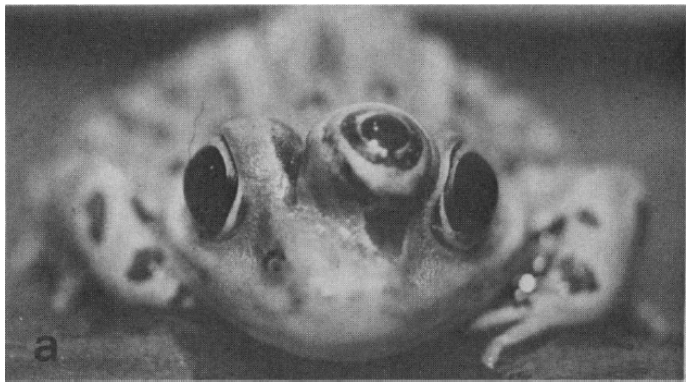



Seeing with your tongue

Human echolocation (sonar)



Haptic belt: Direction sense



Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

Comparison of computing power

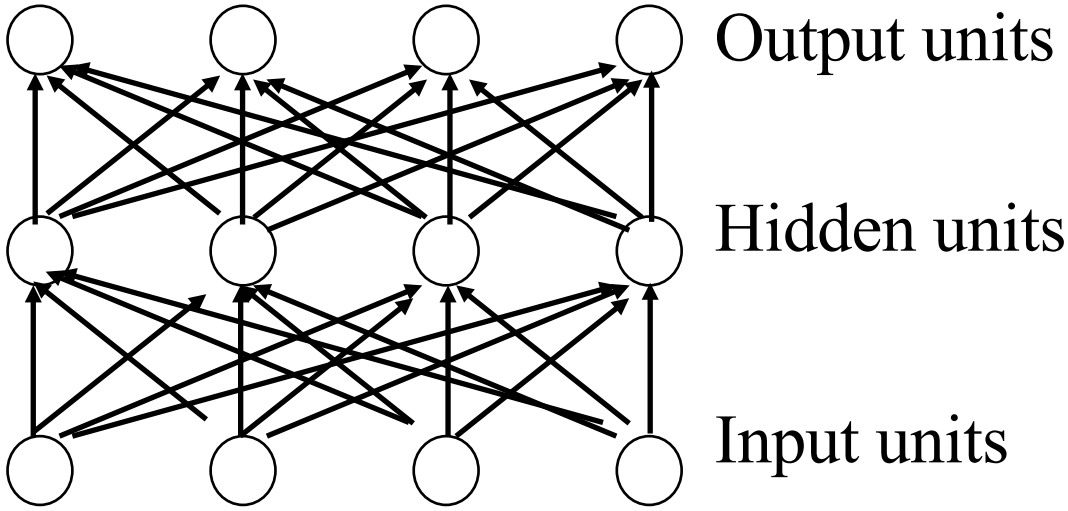
INFORMATION CIRCA 2012	Computer	Human Brain
Computation Units	10-core Xeon: 10^9 Gates	10^{11} Neurons
Storage Units	10^9 bits RAM, 10^{12} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-9} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec

- Computers are way faster than neurons…
- But there are a lot more neurons than we can reasonably model in modern digital computers, and they all fire in parallel
- Neural networks are designed to be massively parallel
- The brain is effectively a billion times faster

Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

Neural networks

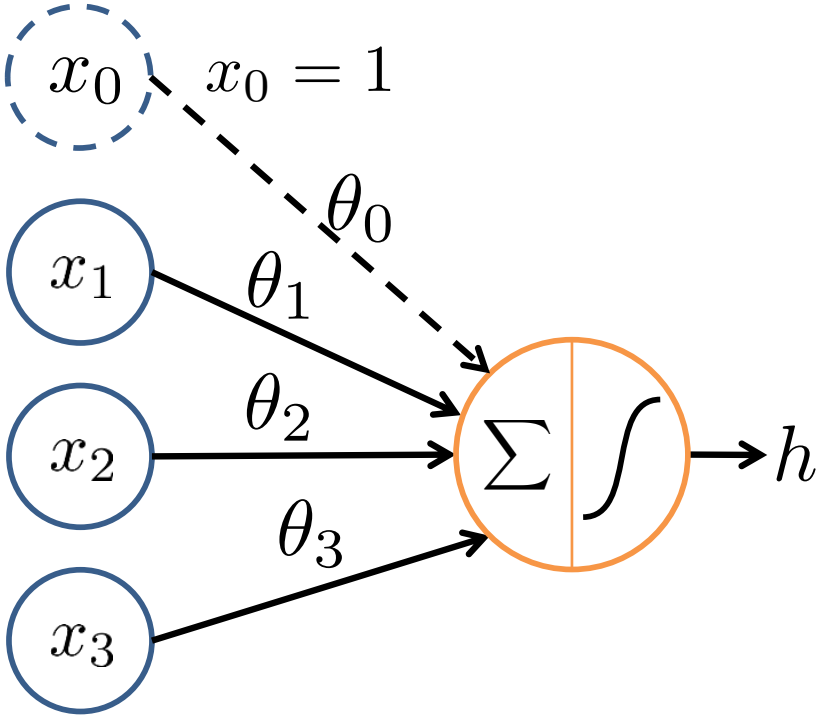


Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Neuron Model: Logistic Unit



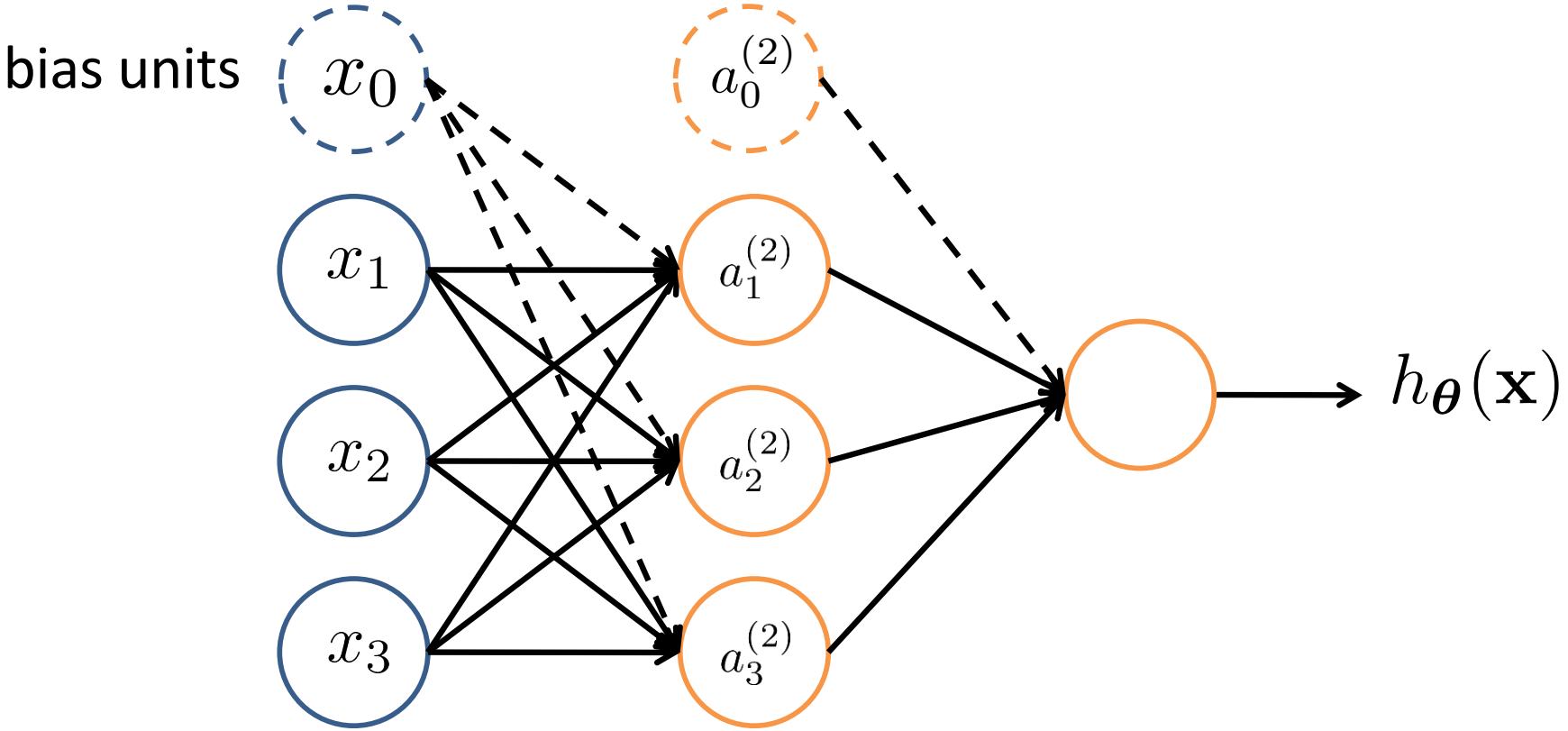
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\mathsf{T}}\mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Based on slide by Andrew Ng

Neural Network



Layer 1 (Input Layer)

Layer 2 (Hidden Layer)

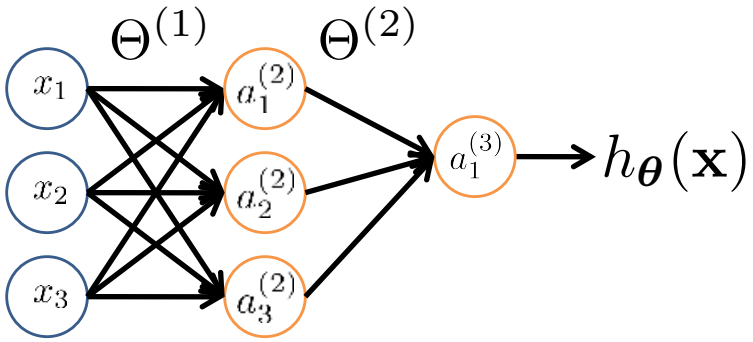
Layer 3 (Output Layer)

Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Neural Network



$a_i^{(j)} =$ "activation" of unit i in layer j

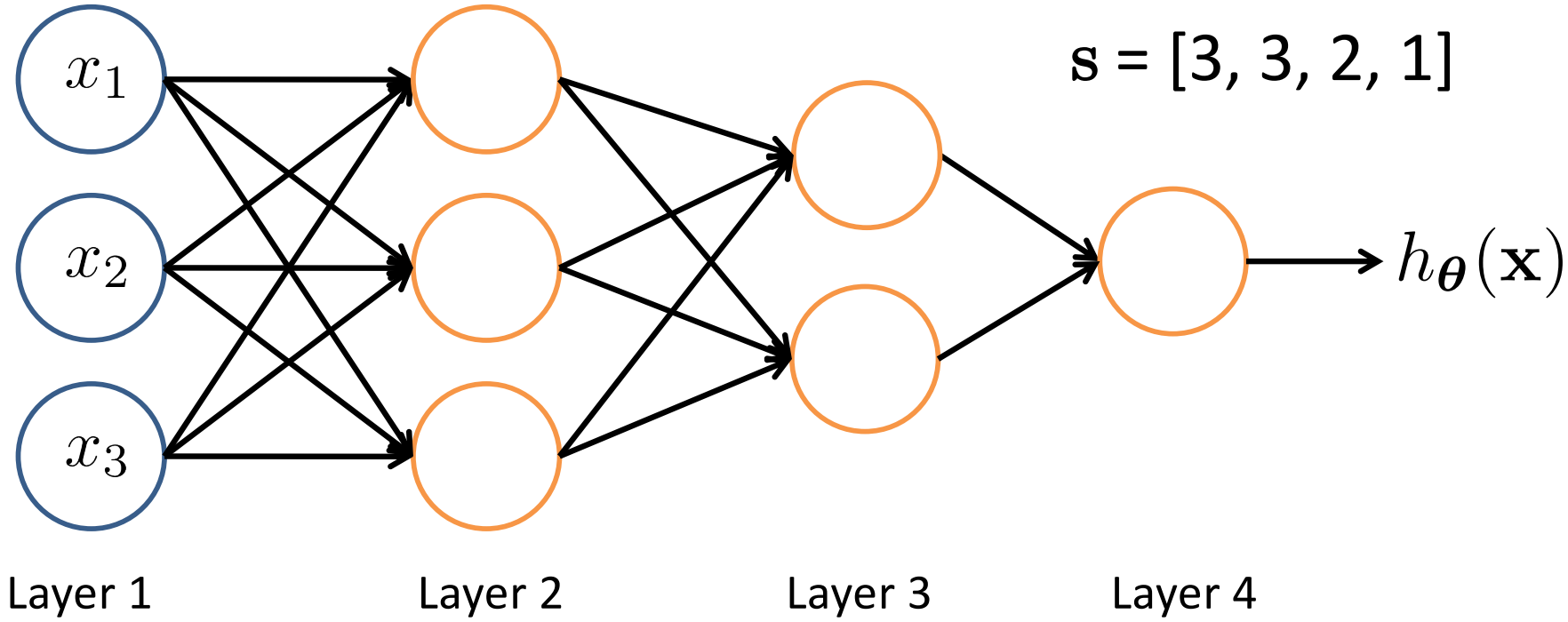
$\Theta^{(j)} =$ weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

Other Network Architectures



L denotes the number of layers

$\mathbf{s} \in \mathbb{N}^{+L}$ contains the numbers of nodes at each layer

- Not counting bias units
- Typically, $s_0 = d$ (# input features) and $s_{L-1} = K$ (# classes)

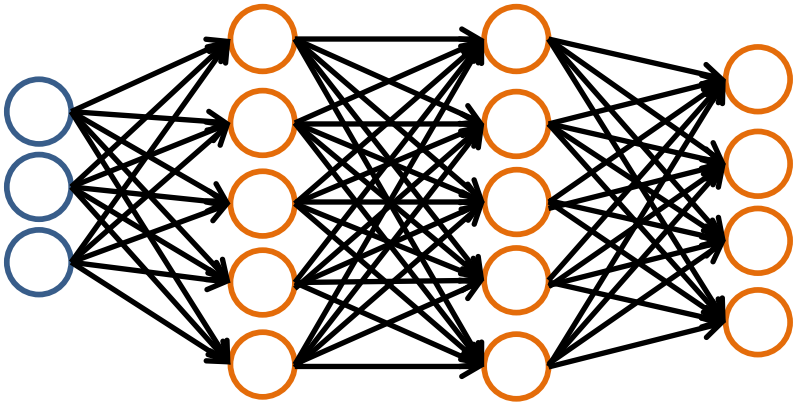
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

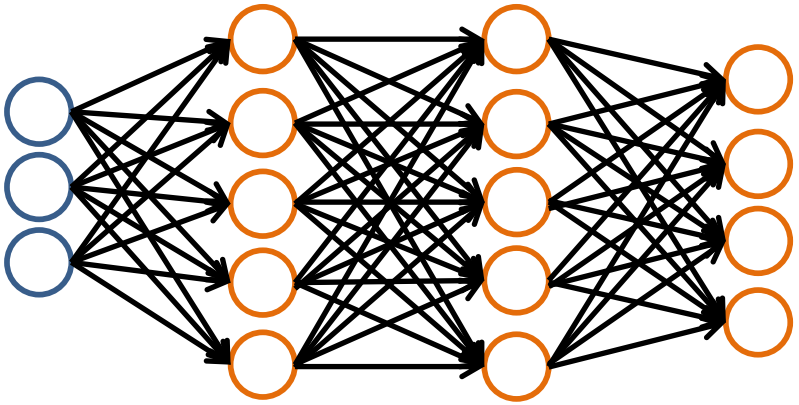
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$ when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$ when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$ when truck

Neural Network Classification



Given:

$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

$\mathbf{s} \in \mathbb{N}^{+L}$ contains # nodes at each layer

- $s_0 = d$ (# features)

Binary classification

y = 0 or 1

1 output unit $(s_{L-1} = 1)$

Multi-class classification (K classes)

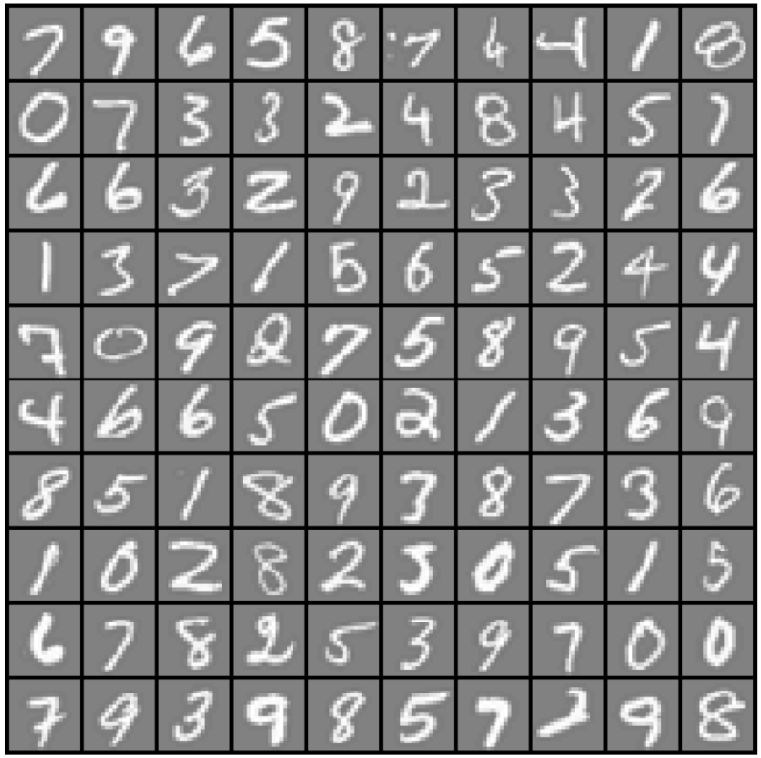
$\mathbf{y} \in \mathbb{R}^K$ e.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

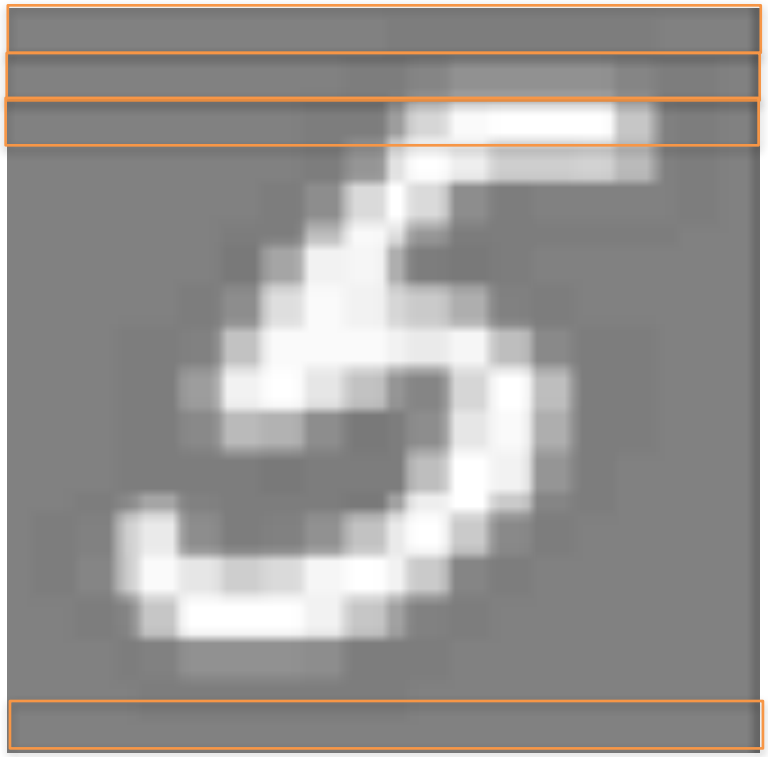
pedestrian car motorcycle truck

K output units $(s_{L-1} = K)$

Understanding Representations

Layering Representations





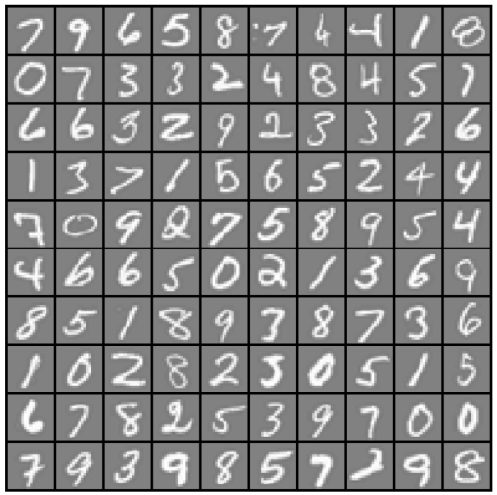
$x_1 \ \dots \ x_{20}$
$x_{21} \ \dots \ x_{40}$
$x_{41} \ \dots \ x_{60}$

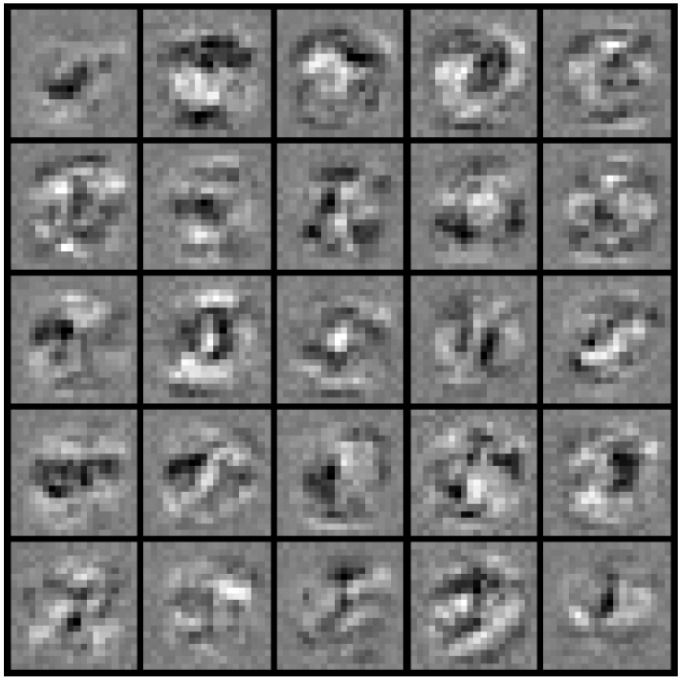
$x_{381} \ \dots \ x_{400}$

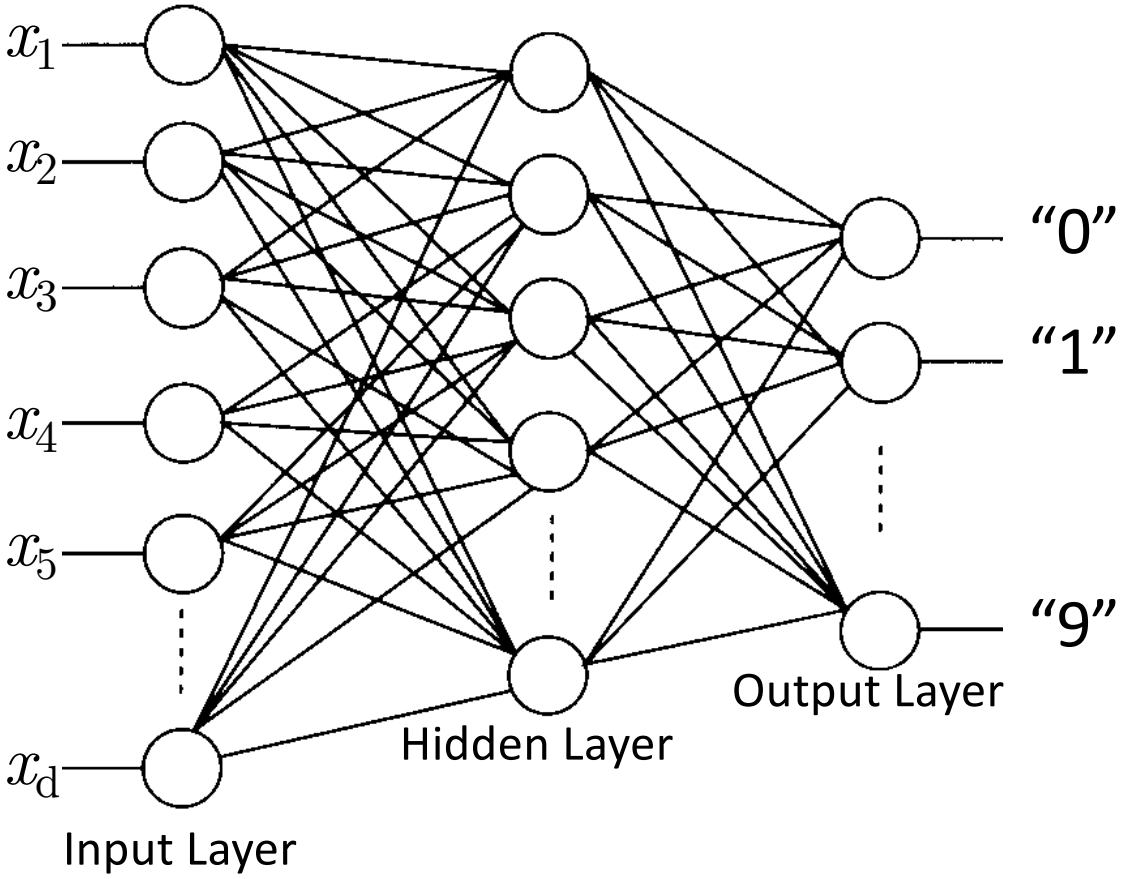
20 × 20 pixel images
d = 400 10 classes

Each image is “unrolled” into a vector $\mathbf{x}$ of pixel intensities

Layering Representations







Visualization of Hidden Layer

LeNet 5 Demonstration: http://yann.lecun.com/exdb/lenet/

Unsupervised Learning:
K-Means &
Gaussian Mixture Models

Unsupervised Learning

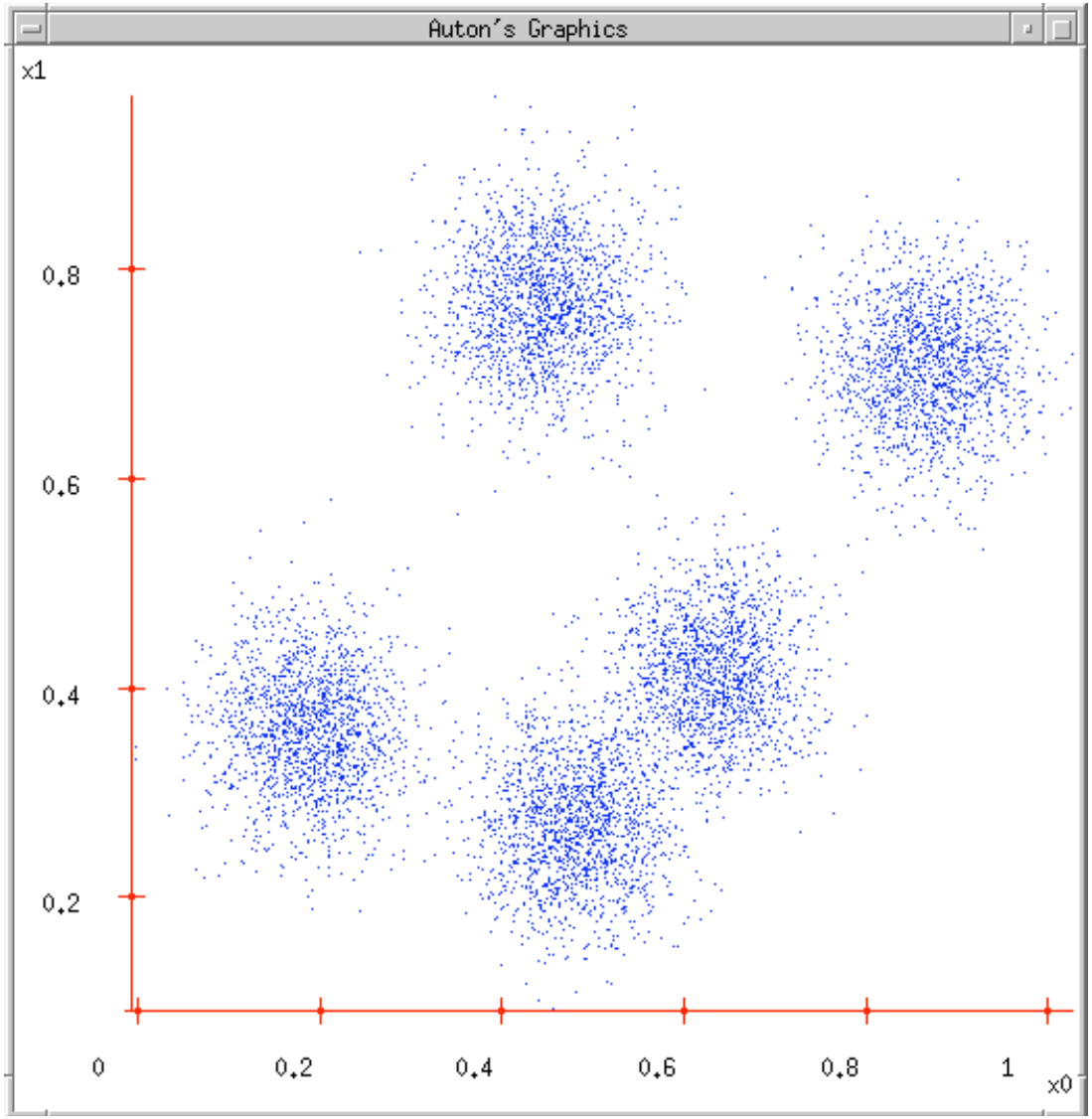
- Supervised learning used labeled data pairs $(\mathbf{x}, \mathrm{y})$ to learn a function $f: \mathrm{X} \rightarrow \mathrm{Y}$
  - But, what if we don't have labels?

- No labels = **unsupervised learning**
- Only some points are labeled = **semi-supervised learning**
  - Labels may be expensive to obtain, so we only get a few

- **Clustering** is the unsupervised grouping of data points. It can be used for **knowledge discovery**.

K-Means Clustering

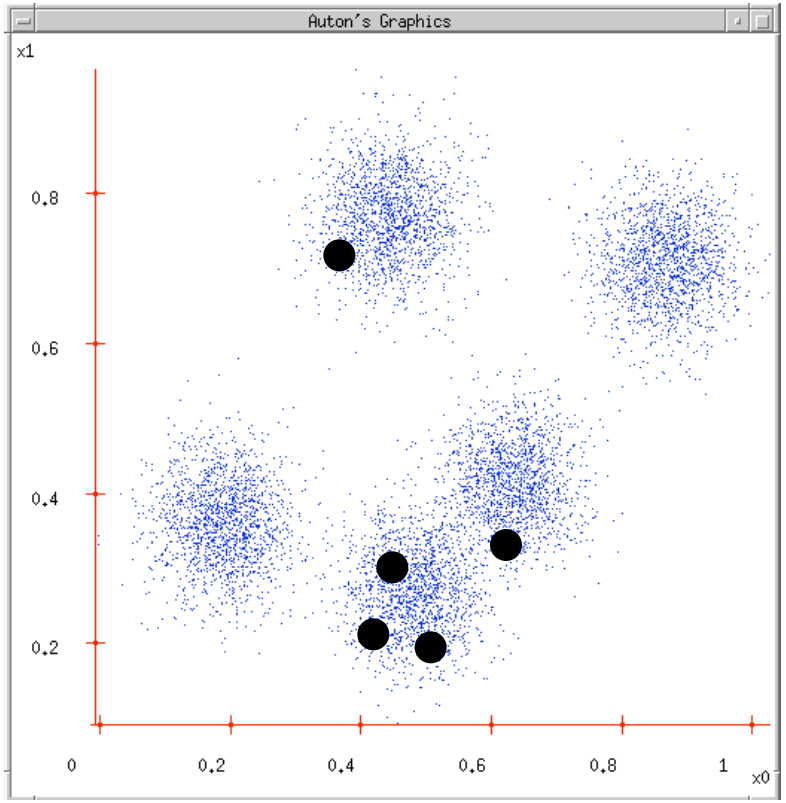
Clustering Data



K-Means Clustering

K-Means (k , X)

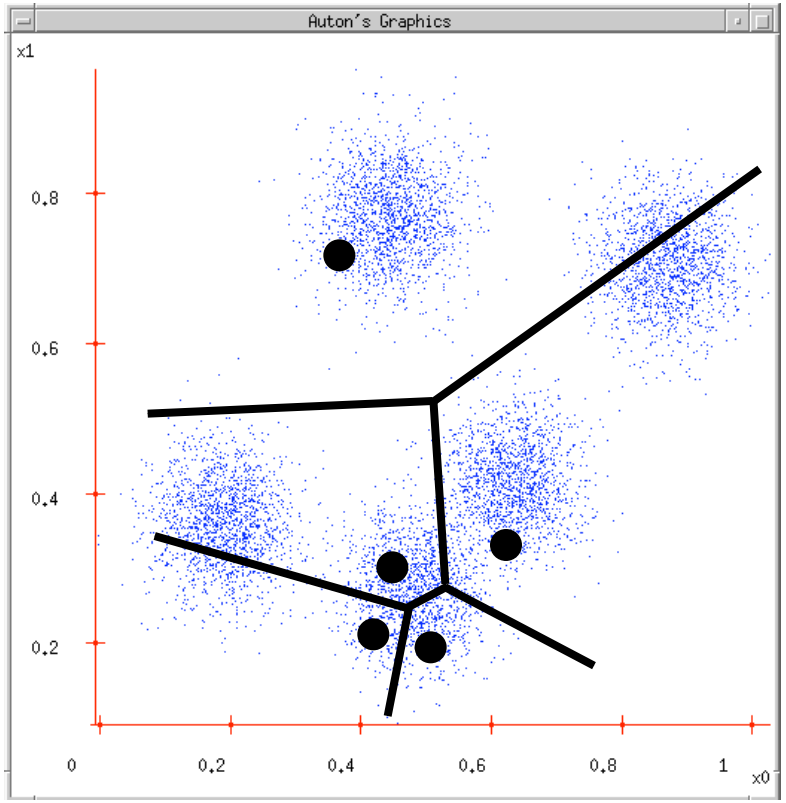
- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster



K-Means Clustering

K-Means (k , X)

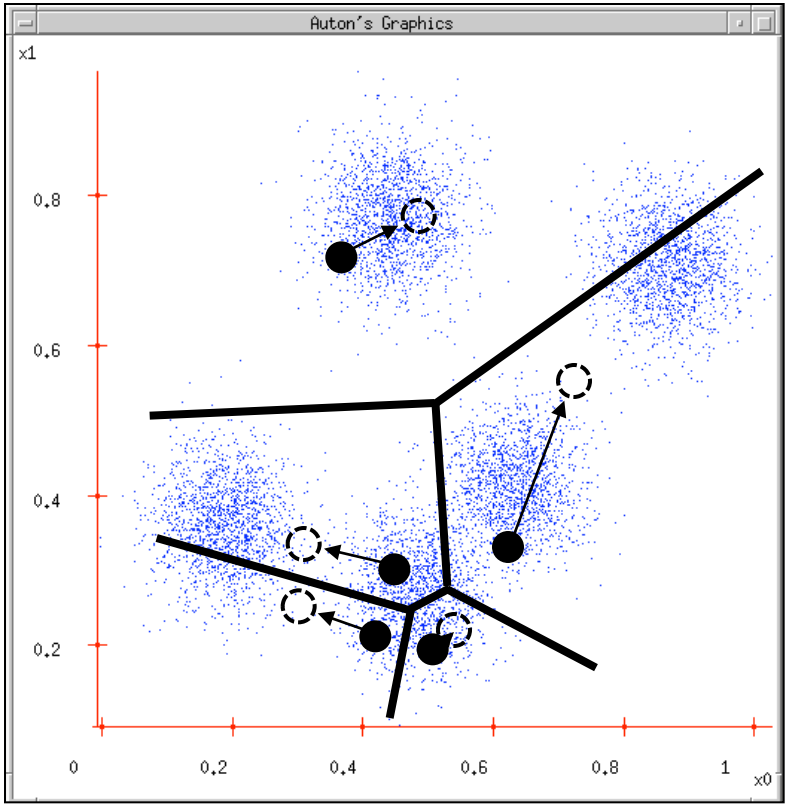
- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster



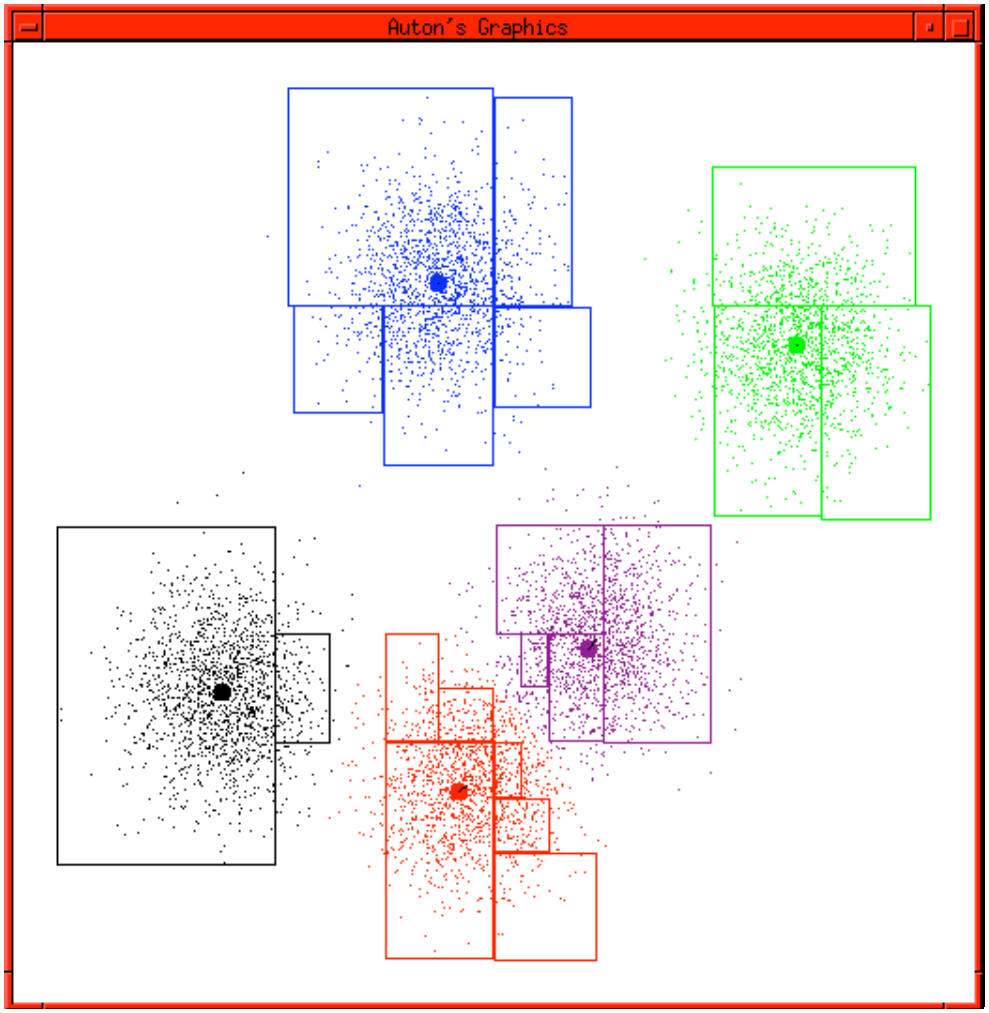
K-Means Clustering

K-Means (k , X)

- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster



K-Means Animation



Example generated by Andrew Moore using Dan Pelleg's super-duper fast K-means system:

Dan Pelleg and Andrew Moore. Accelerating Exact k-means Algorithms with Geometric Reasoning. Proc. Conference on Knowledge Discovery in Databases 1999.

K-Means Objective Function

- K-means finds a local optimum of the following objective function:

$$\arg\min_{\boldsymbol{\mathcal{S}}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in \mathcal{S}_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|_2^2$$

where $\boldsymbol{\mathcal{S}} = \{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ is a partitioning over $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ s.t. $X = \bigcup_{i=1}^{k} \mathcal{S}_i$ and $\boldsymbol{\mu}_i = \text{mean}(\mathcal{S}_i)$

Problems with K-Means

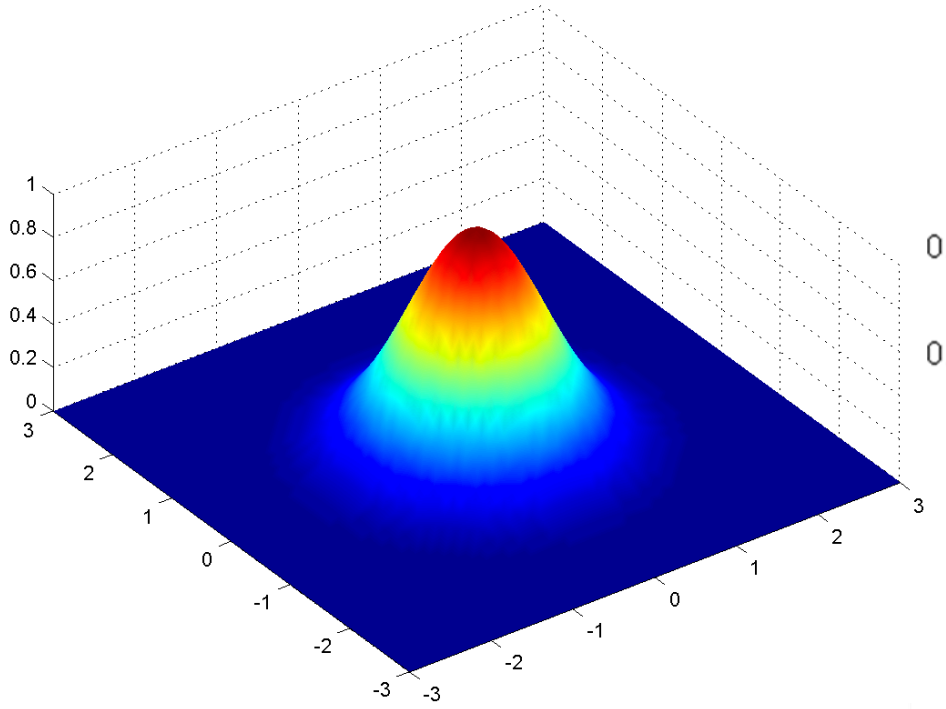
- ***Very*** sensitive to the initial points
  - Do many runs of K-Means, each with different initial centroids
  - Seed the centroids using a better method than randomly choosing the centroids
    - e.g., Farthest-first sampling

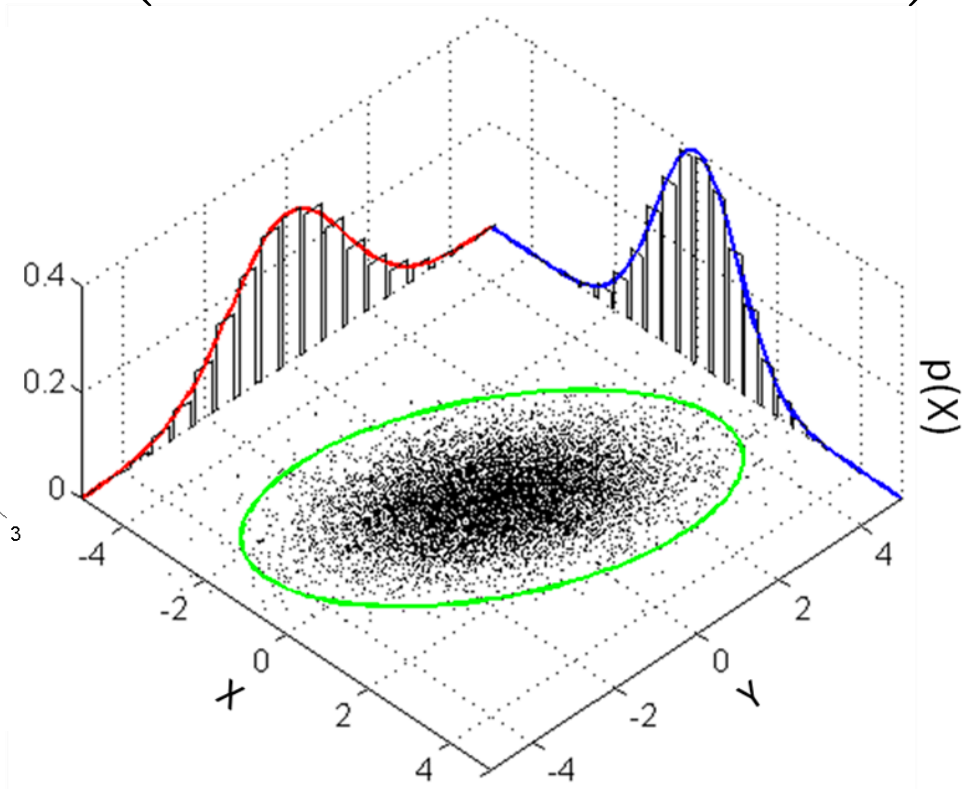
- Must manually choose k
  - Learn the optimal k for the clustering
    - Note that this requires a performance measure

Gaussian Mixture Models

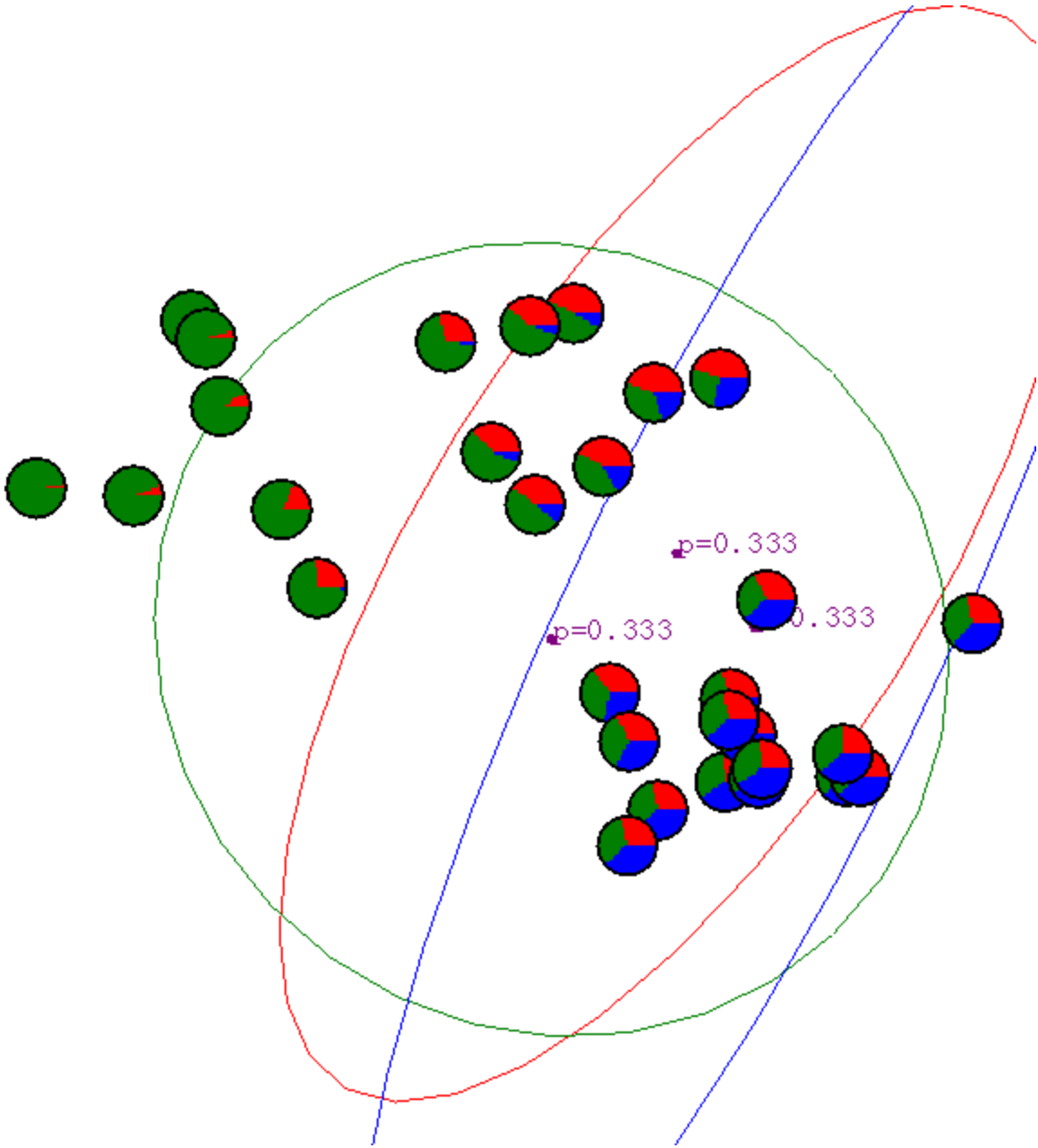
- Recall the Gaussian distribution:

$$P(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$



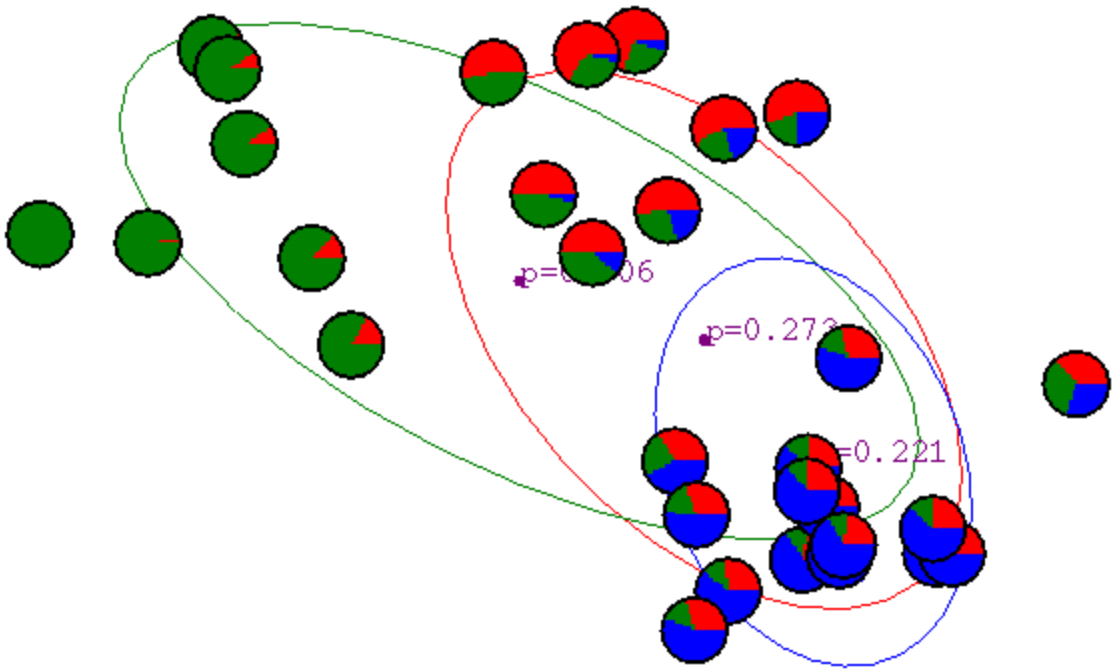


Gaussian Mixture Example: Start

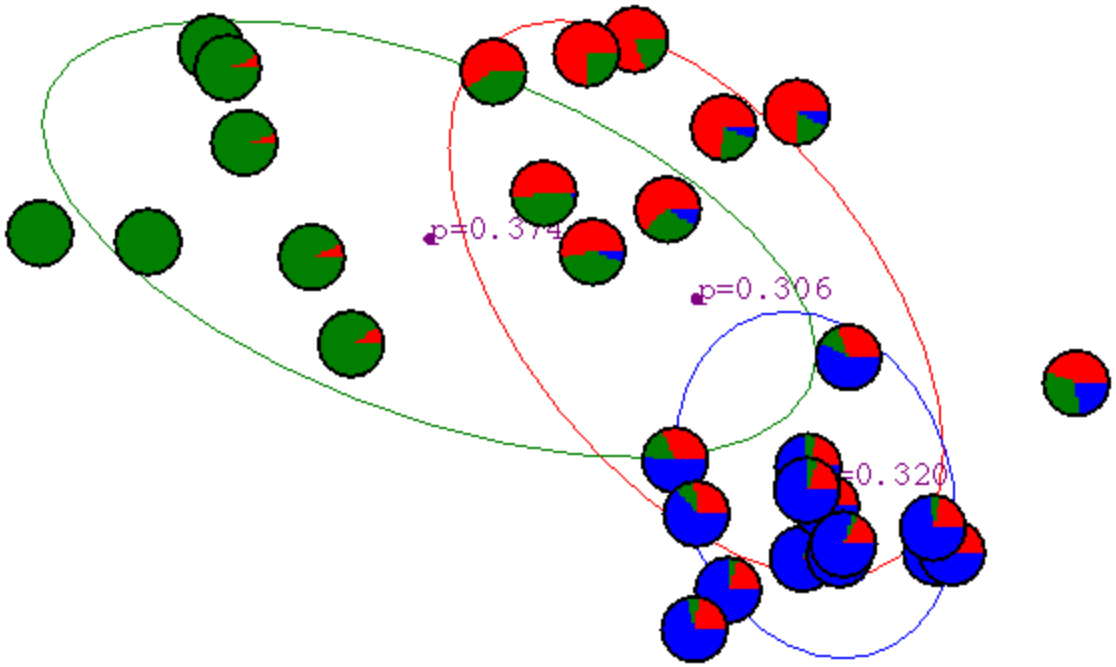


Advance apologies: in Black and White this example will be incomprehensible

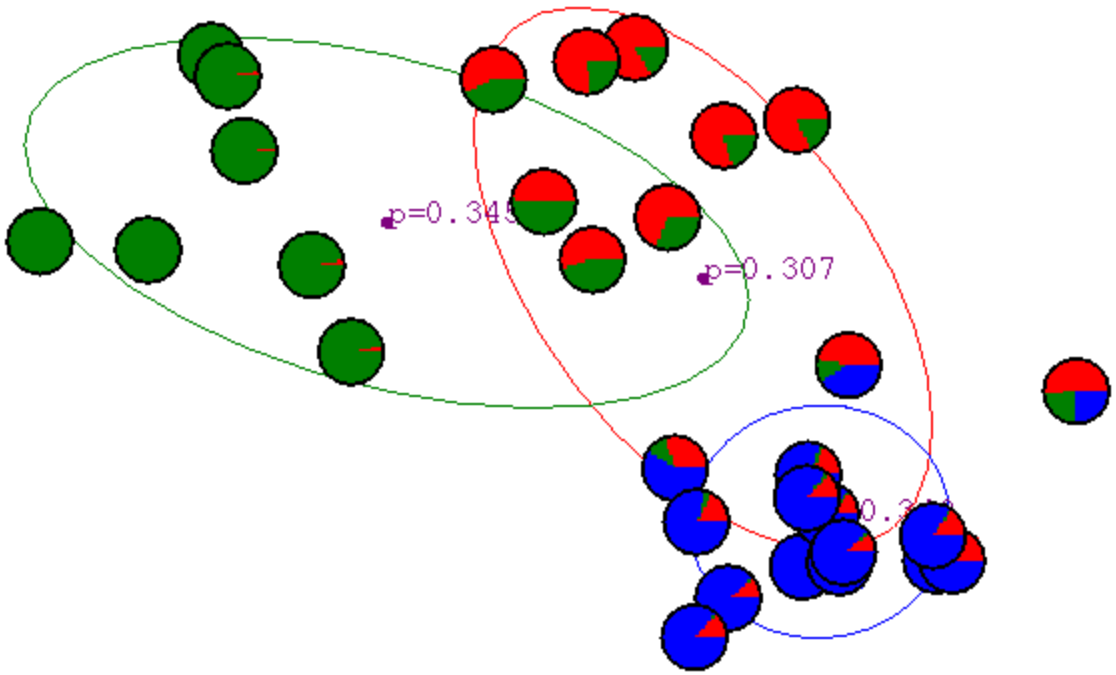
After first iteration



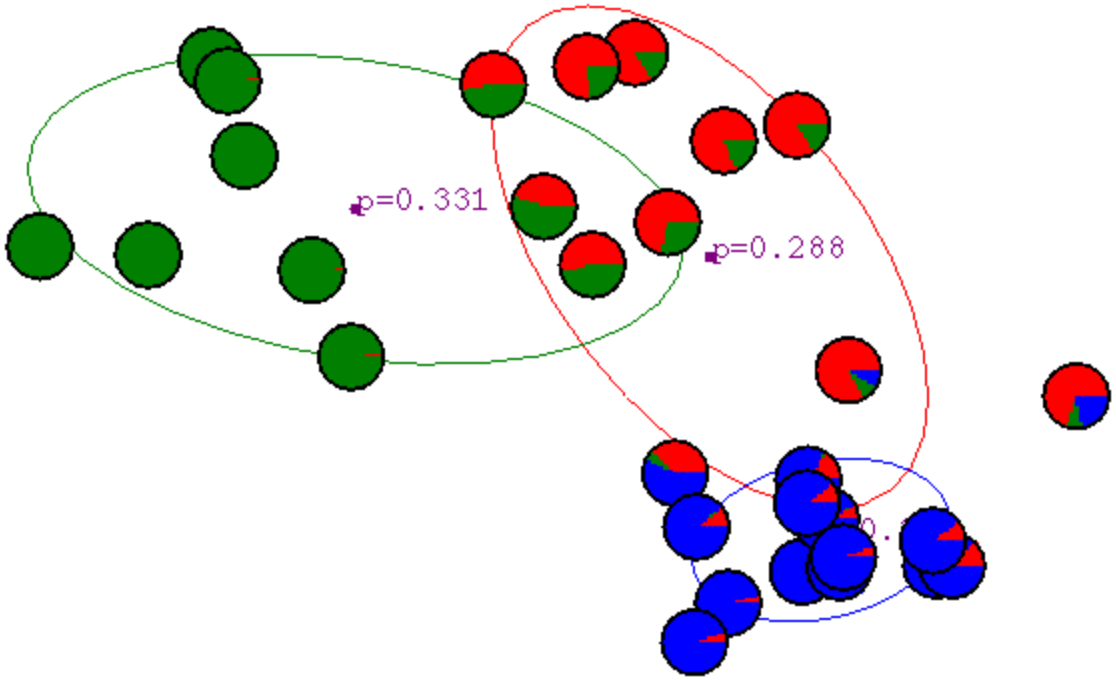
After 2nd iteration



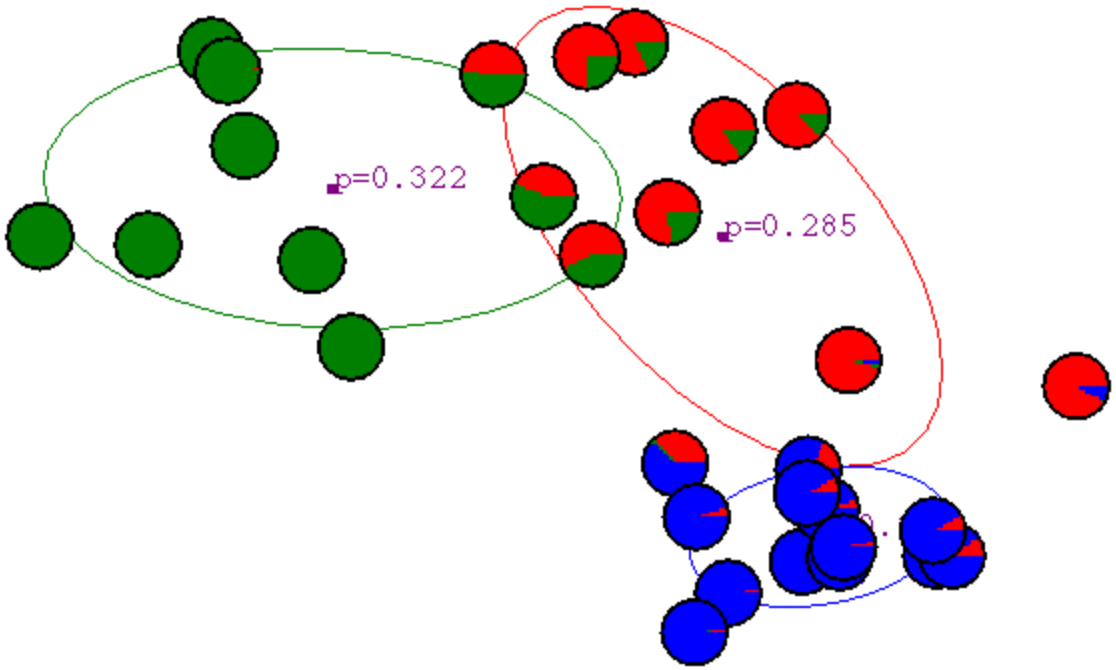
After 3rd iteration



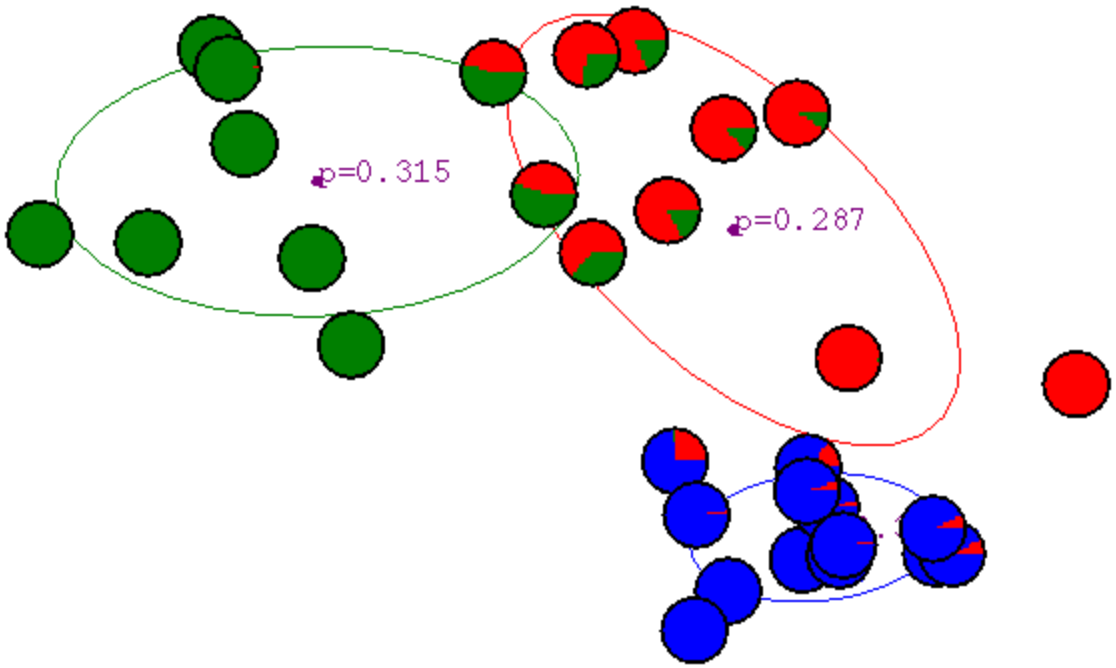
After 4th iteration



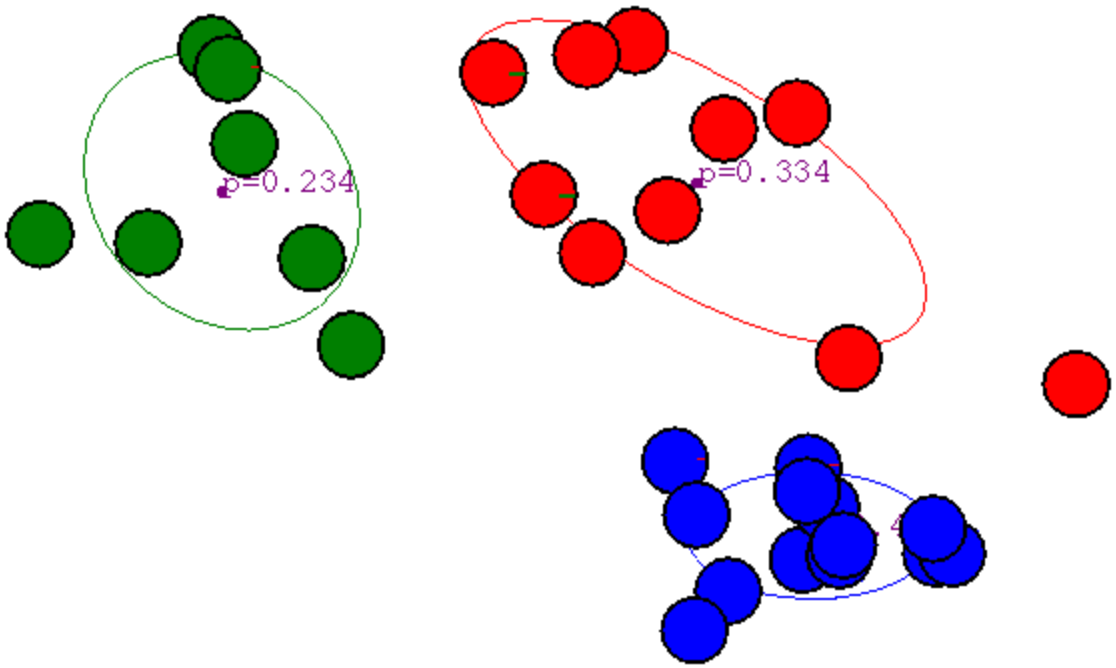
After 5th iteration



After 6th iteration

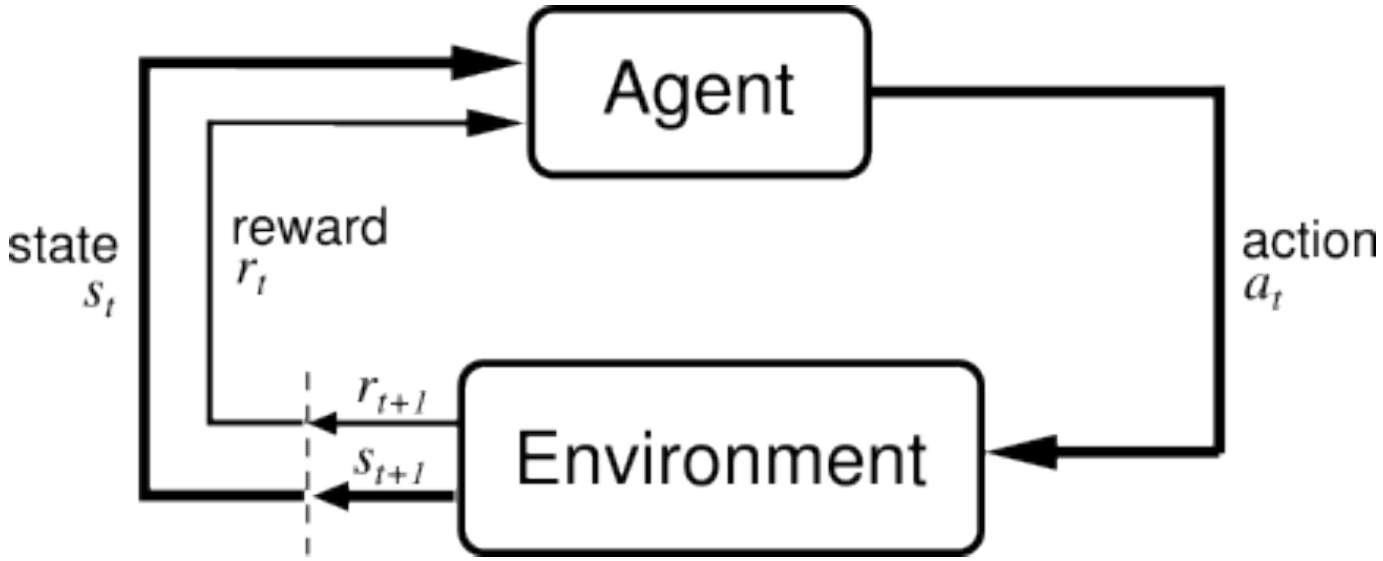


After 20th iteration



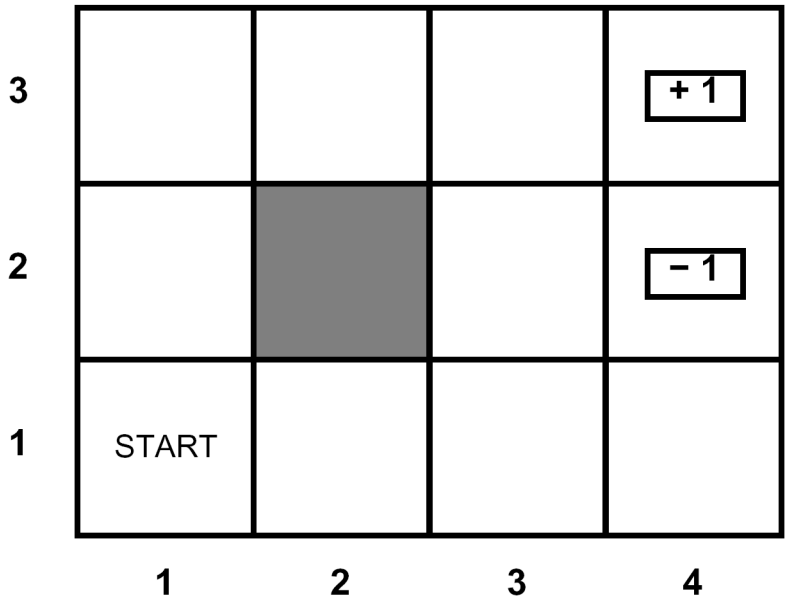
Reinforcement Learning

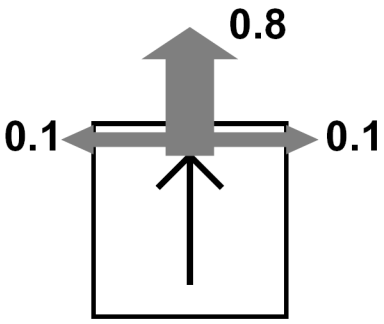
- Basic idea:
  - Receive feedback in the form of rewards
  - Agent's utility is defined by the reward function
  - Must (learn to) act so as to maximize expected rewards



Grid World

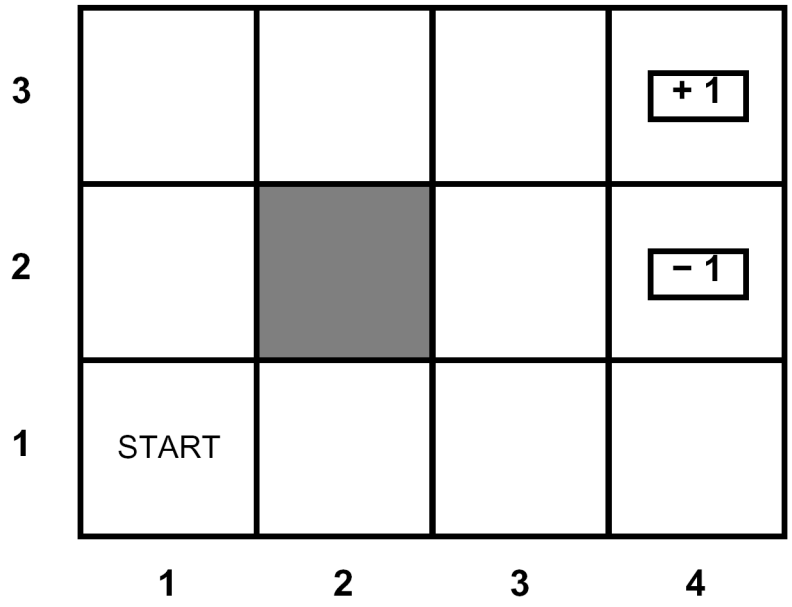
- The agent lives in a grid
- Walls block the agent's path
- The agent's actions do not always go as planned:
  - 80% of the time, the action North takes the agent North (if there is no wall there)
  - 10% of the time, North takes the agent West; 10% East
  - If there is a wall in the direction the agent would have been taken, the agent stays put
- Small "living" reward each step
- Big rewards come at the end
- Goal: maximize sum of rewards*



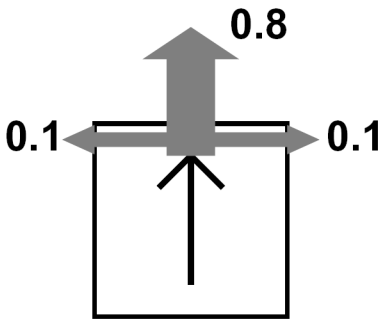


Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s,a,s')
    - Prob that a from s leads to s'
    - i.e., P(s' | s,a)
    - Also called the model
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state (or distribution)
  - Maybe a terminal state



- MDPs are a family of non-deterministic search problems
  - Reinforcement learning: MDPs where we don't know the transition or reward functions



What is Markov about MDPs?

- Andrey Markov (1856-1922)
- "Markov" generally means that given the present state, the future and the past are independent
- For Markov decision processes, "Markov" means:

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

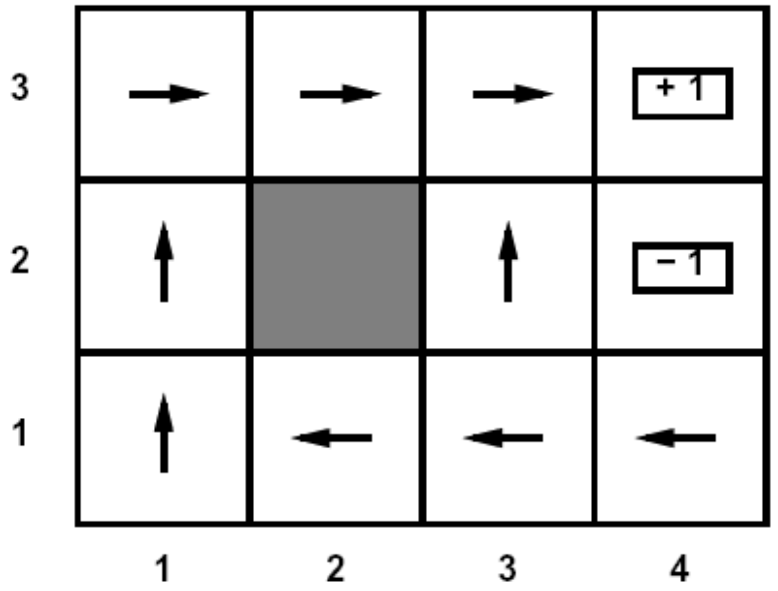
$$=$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

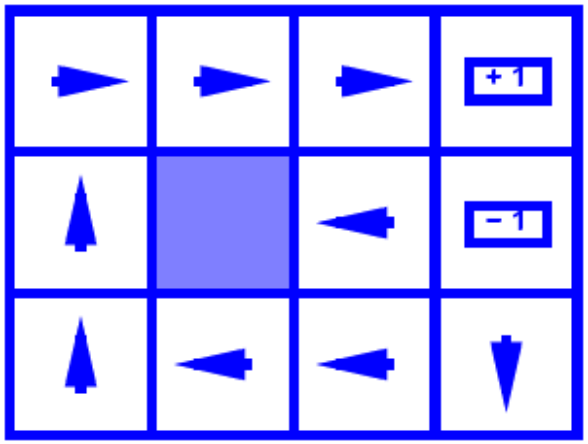
Solving MDPs

- In deterministic single-agent search problems, want an optimal plan, or sequence of actions, from start to a goal
- In an MDP, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy maximizes expected utility if followed
  - Defines a reflex agent

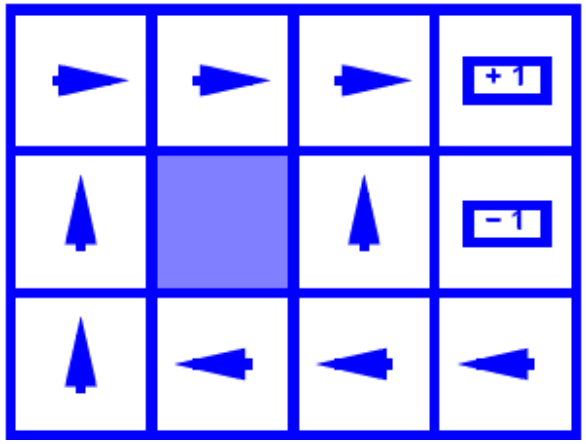
Optimal policy when R(s, a, s') = -0.03 for all non-terminals s



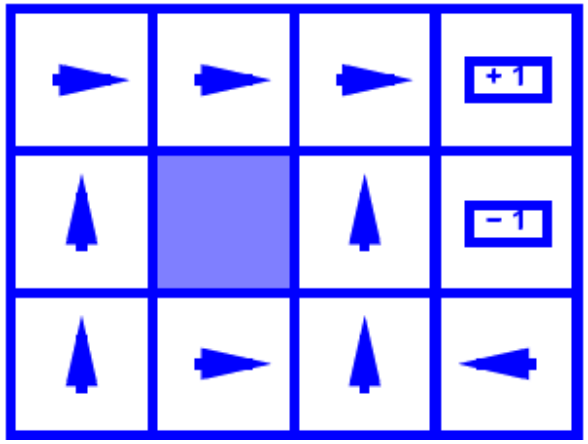
Example Optimal Policies



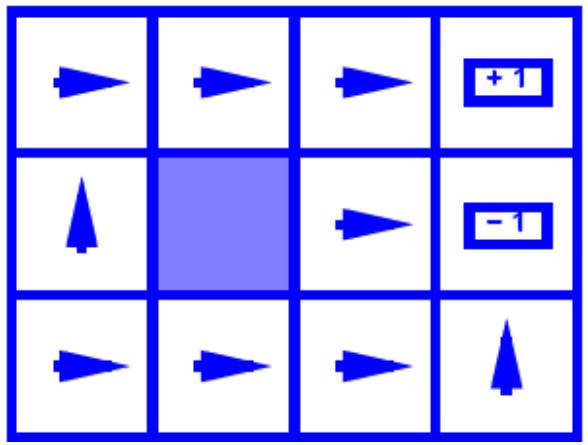
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4

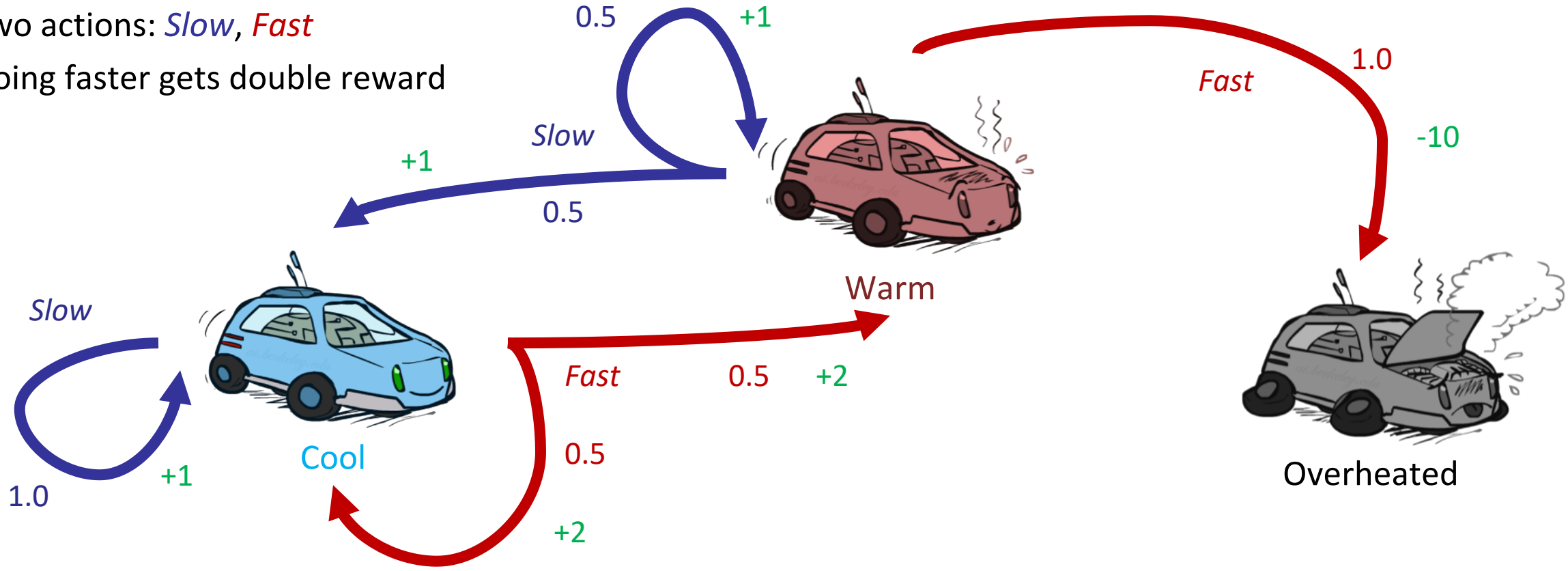


R(s) = -2.0

MDP Example: Car Control

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward

- Find a policy from states to actions
  - That is if the car is Cool, warm or overheated what should you do? Go Fast or Go Slow?



Policy Search

- Problem: often the feature-based policies that work well aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - We'll see this distinction between modeling and prediction again later in the course

- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards