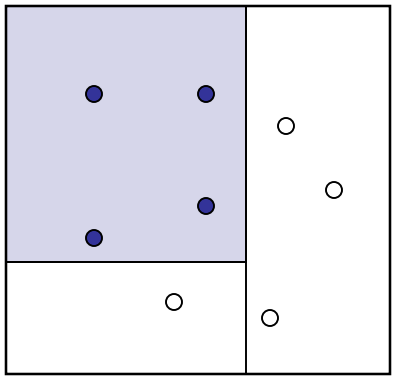
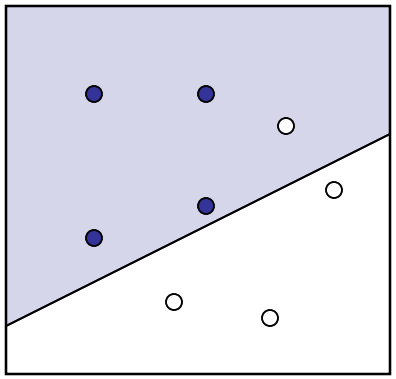
Classification

- Basic concepts
- Decision tree
- Naïve Bayesian classifier
- Model evaluation
- **Support Vector Machines**
- Regression
- Neural Networks and Deep Learning
- Lazy Learners (k Nearest Neighbors)
- Bayesian Belief Networks

Classification: Discriminant Function

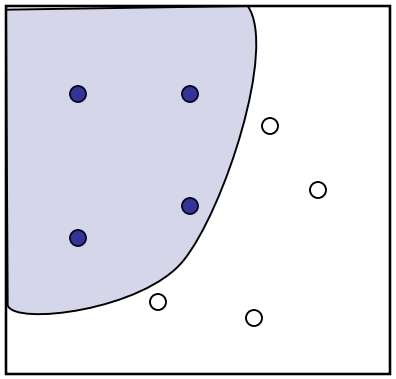


Decision Tree



Linear Functions

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$



Nonlinear Functions

Linear Discriminant Function

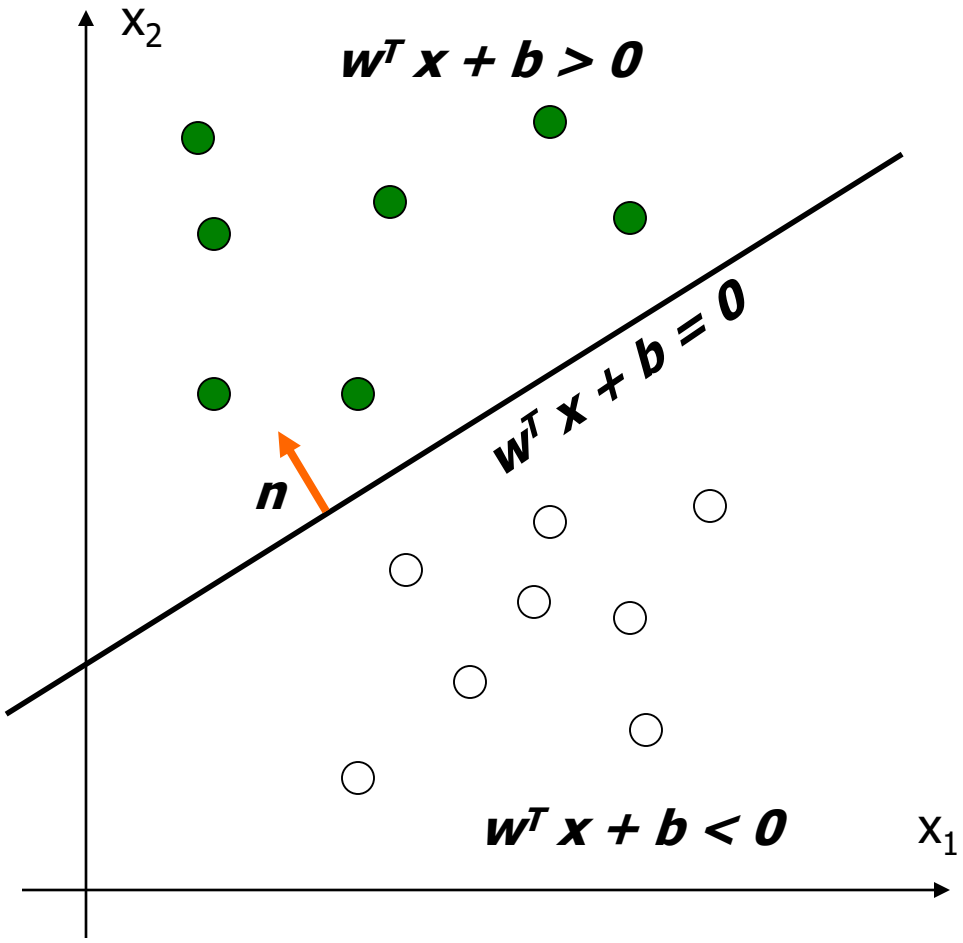
- g(x) is a linear function:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

- A hyper-plane in the feature space

- Unit normal vector of the hyper-plane:

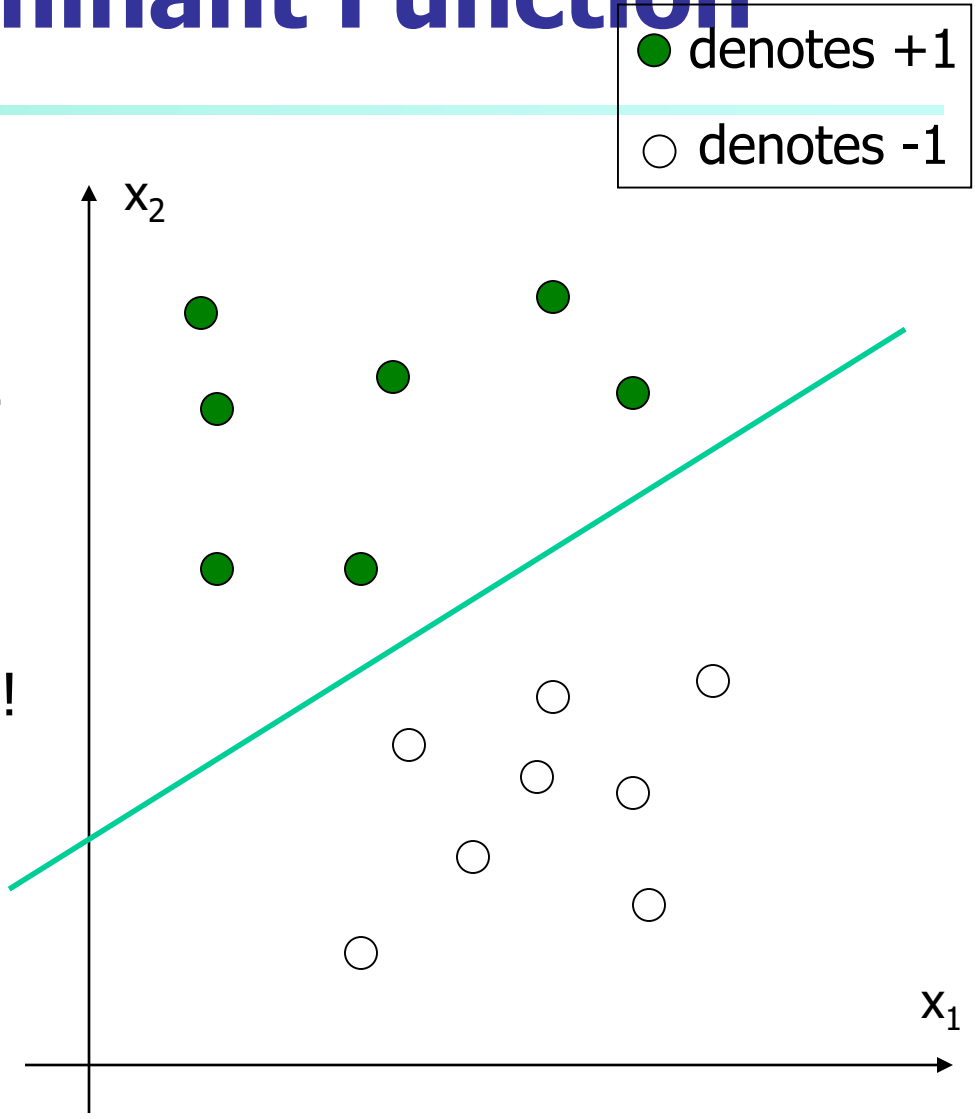
$$\mathbf{n} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$



Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?

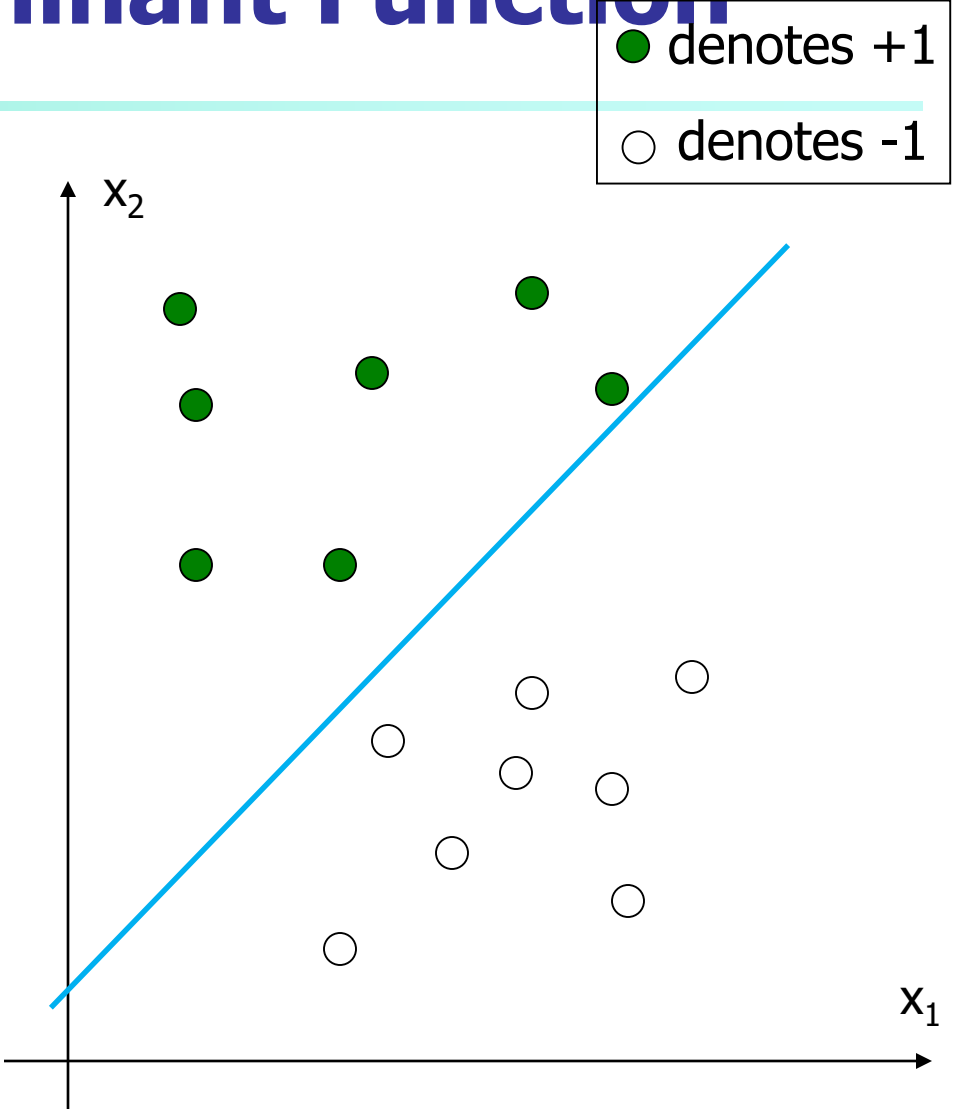
- Infinite number of answers!



Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?

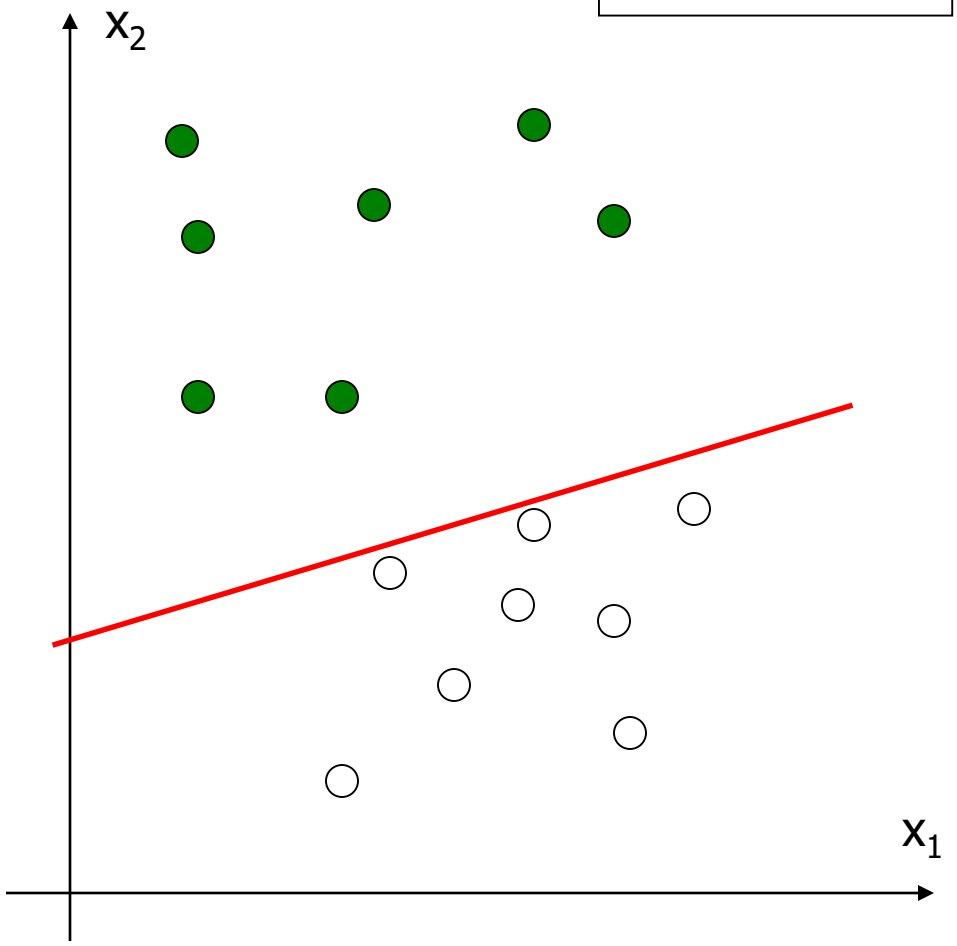
- Infinite number of answers!



Linear Discriminant Function

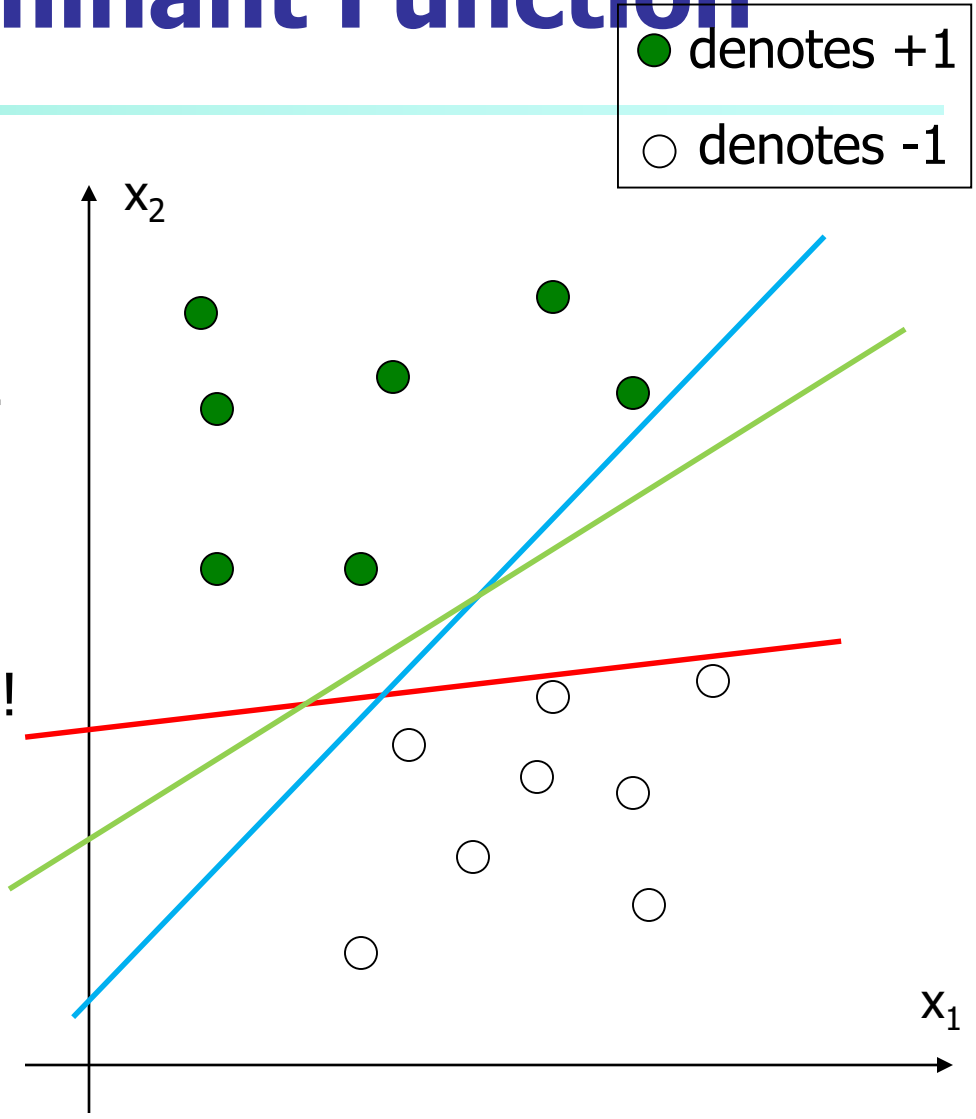
- How would you classify these points using a linear discriminant function in order to minimize the error rate?

- Infinite number of answers!



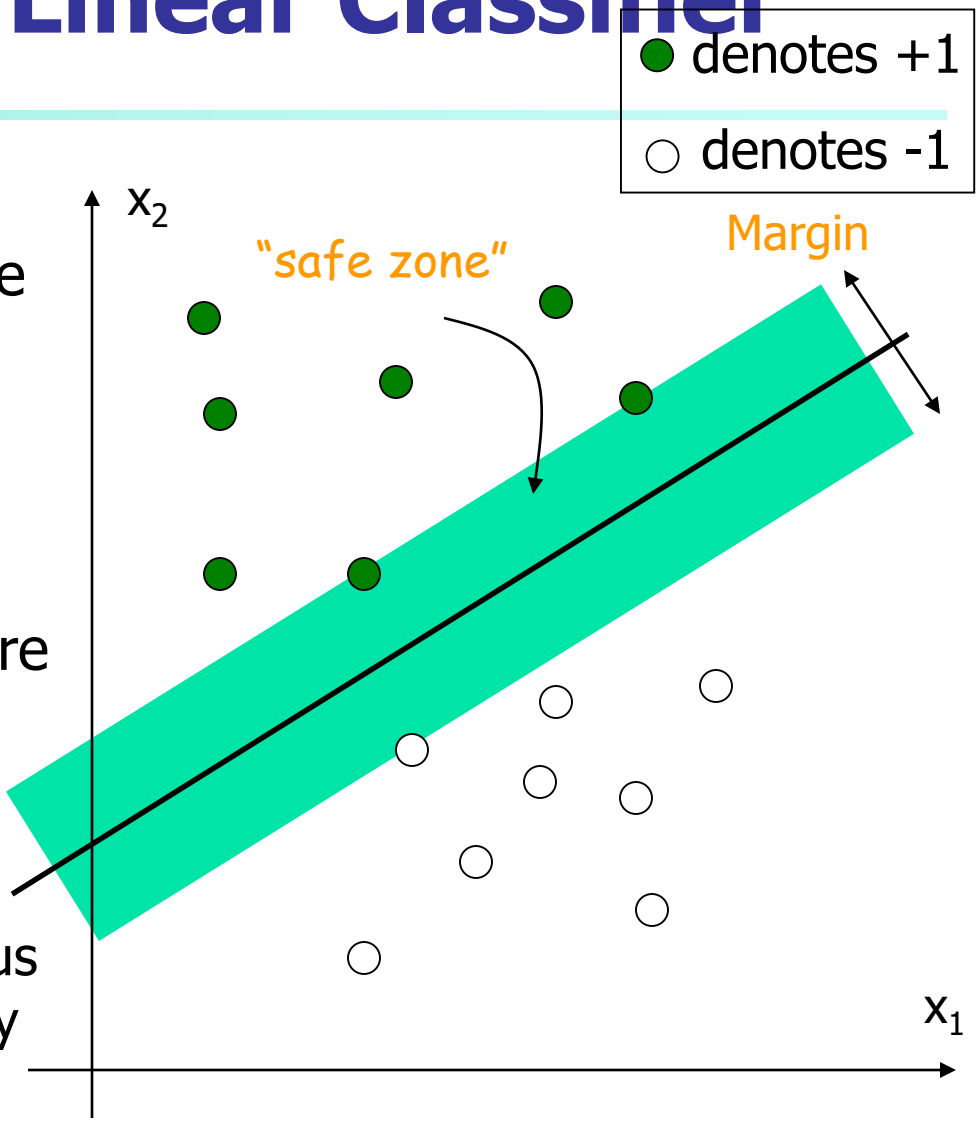
Linear Discriminant Function

- How would you classify these points using a linear discriminant function in order to minimize the error rate?
- Infinite number of answers!
- Which one is the best?



Large Margin Linear Classifier

- The linear discriminant function (classifier) with the maximum margin is the best
- Margin is defined as the width that the boundary could be increased by before hitting a data point
- Why it is the best?
  - Robust to outliners and thus strong generalization ability



Large Margin Linear Classifier

- Given a set of data points:

$\{(\mathbf{x}_i, y_i)\},\ i = 1, 2, \cdots, n$, where

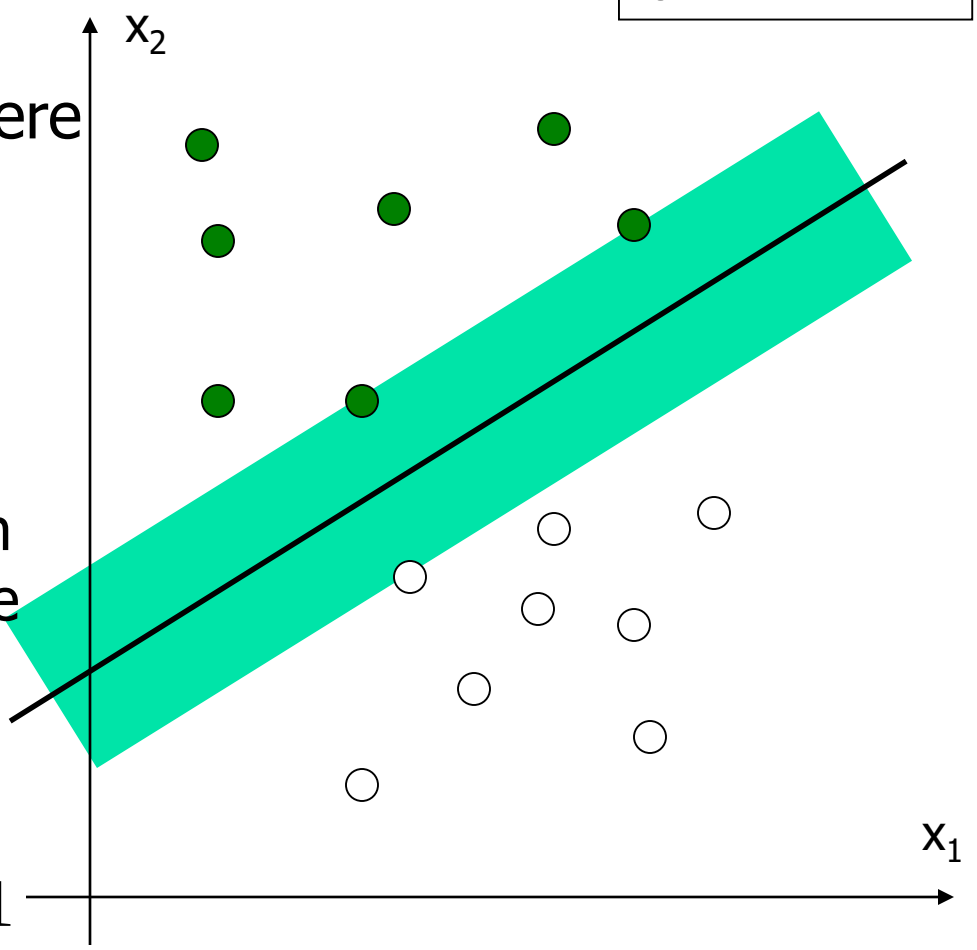
$$\text{For } y_i = +1,\quad \mathbf{w}^T\mathbf{x}_i + b > 0$$

$$\text{For } y_i = -1,\quad \mathbf{w}^T\mathbf{x}_i + b < 0$$

- With a scale transformation on both *w* and *b*, the above is equivalent to

$$\text{For } y_i = +1,\quad \mathbf{w}^T\mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1,\quad \mathbf{w}^T\mathbf{x}_i + b \leq -1$$



Large Margin Linear Classifier

- For the boundary points

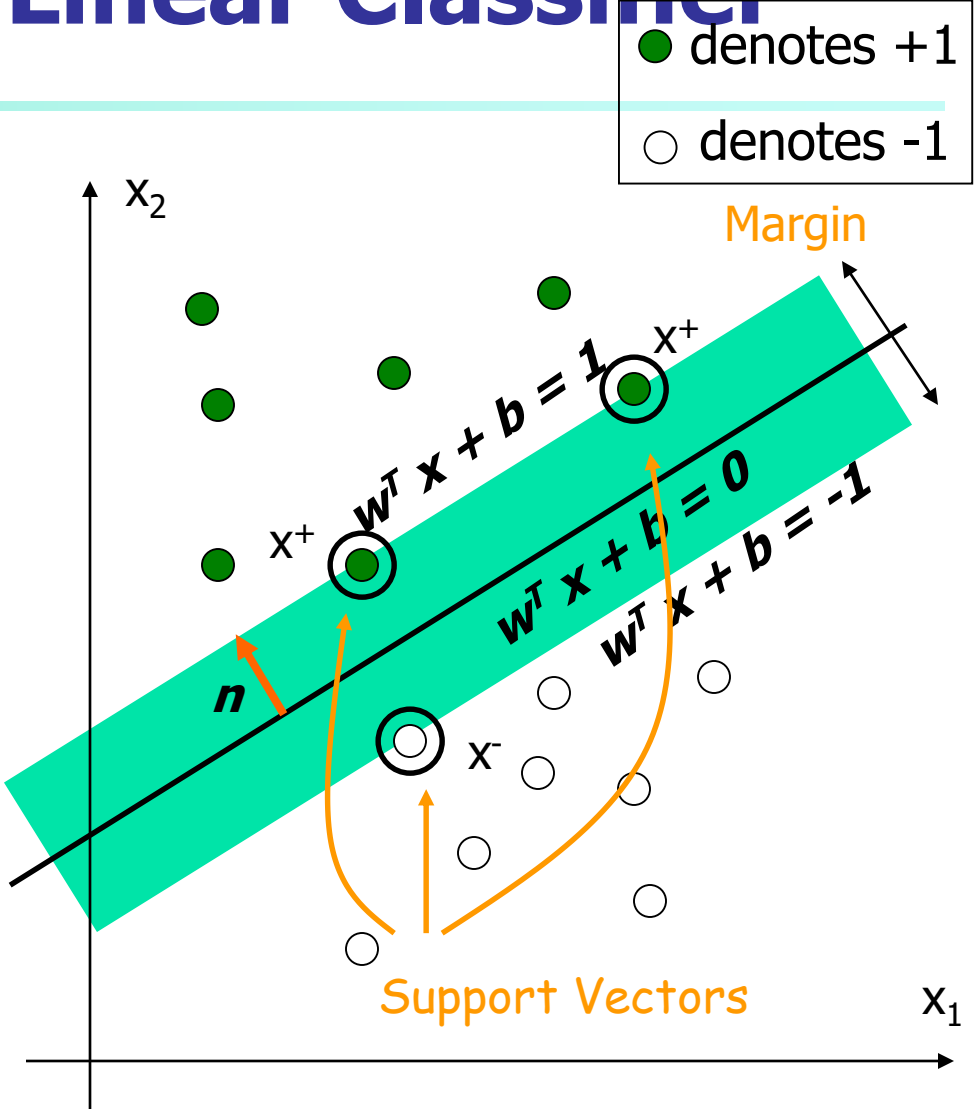
$$\mathbf{w}^T\mathbf{x}^+ + b = 1$$

$$\mathbf{w}^T\mathbf{x}^- + b = -1$$

- The margin width is:

$$M = (\mathbf{x}^+ - \mathbf{x}^-)\cdot\mathbf{n}$$

$$= (\mathbf{x}^+ - \mathbf{x}^-)\cdot\frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$



Large Margin Linear Classifier

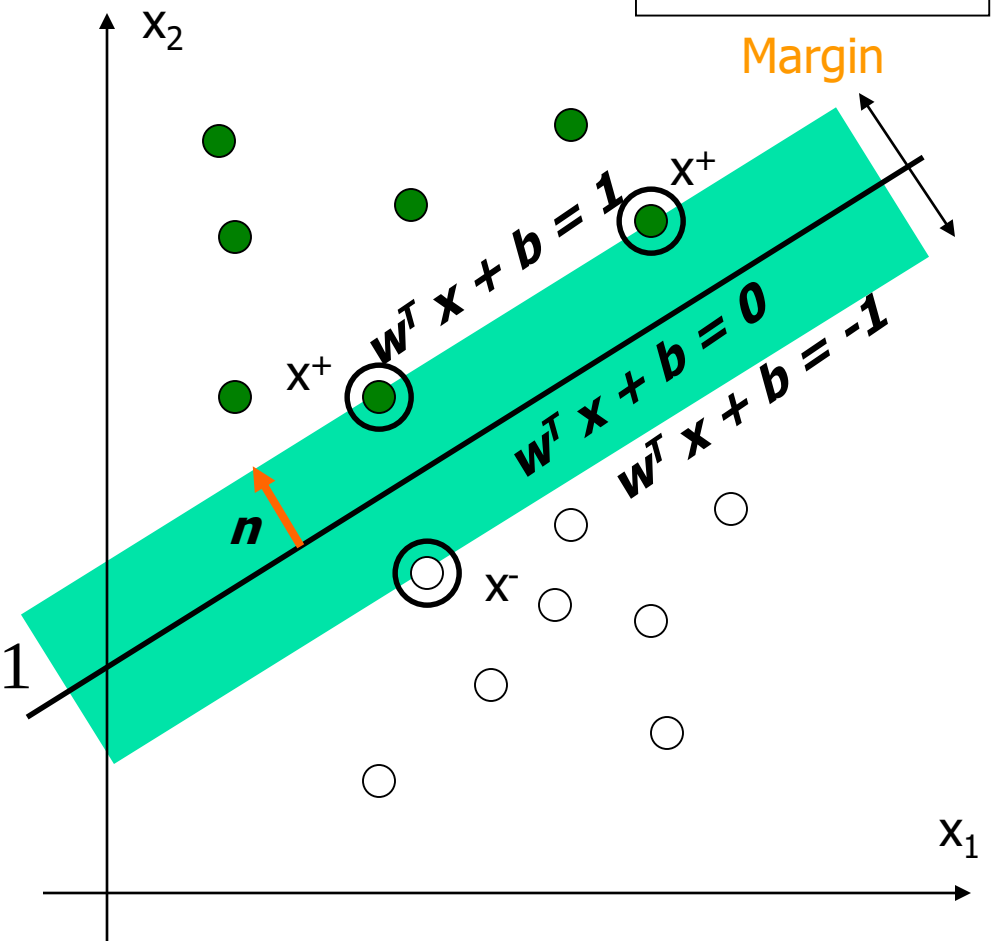
- Formulation:

$$\text{maximize} \quad \frac{2}{\|\mathbf{w}\|}$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T \mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T \mathbf{x}_i + b \leq -1$$



Large Margin Linear Classifier

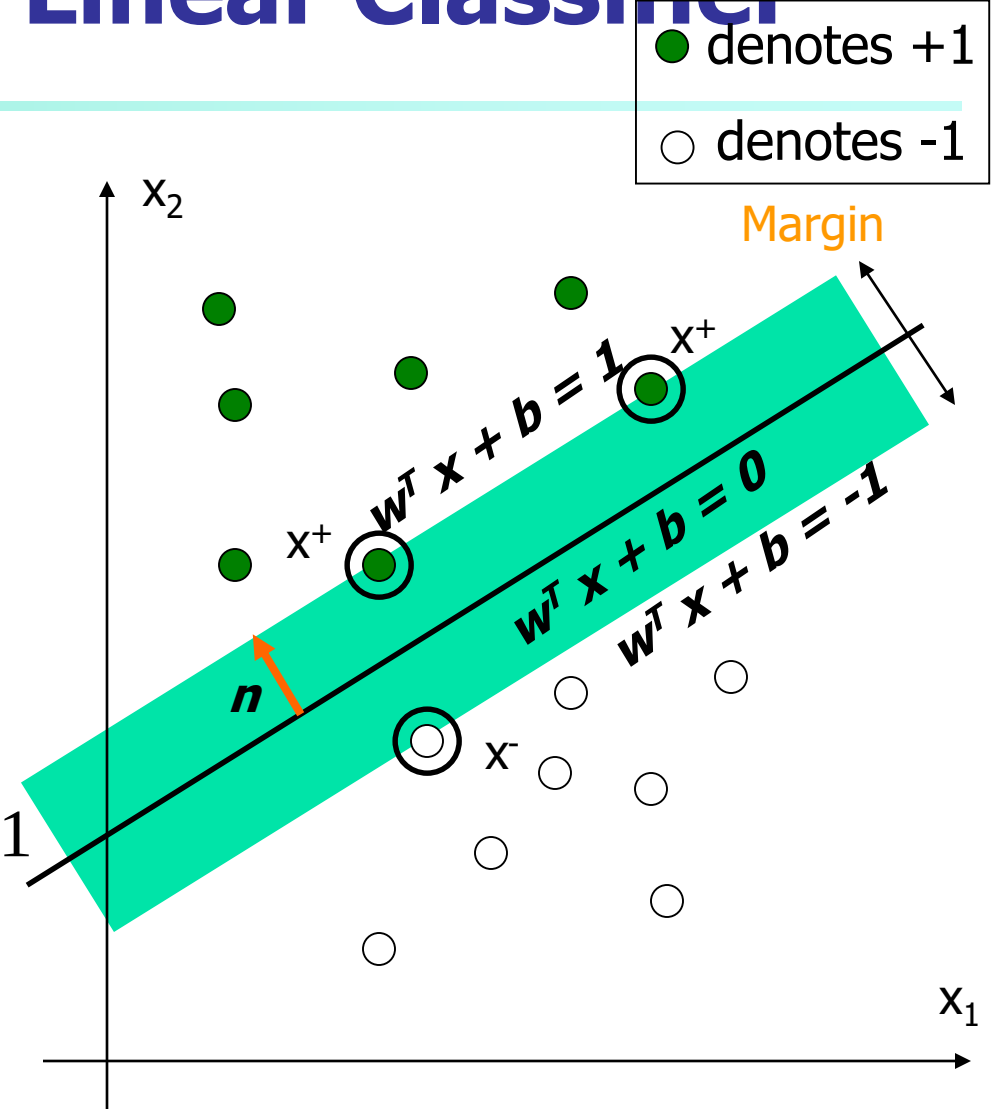
- Formulation:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

such that

$$\text{For } y_i = +1, \quad \mathbf{w}^T\mathbf{x}_i + b \geq 1$$

$$\text{For } y_i = -1, \quad \mathbf{w}^T\mathbf{x}_i + b \leq -1$$



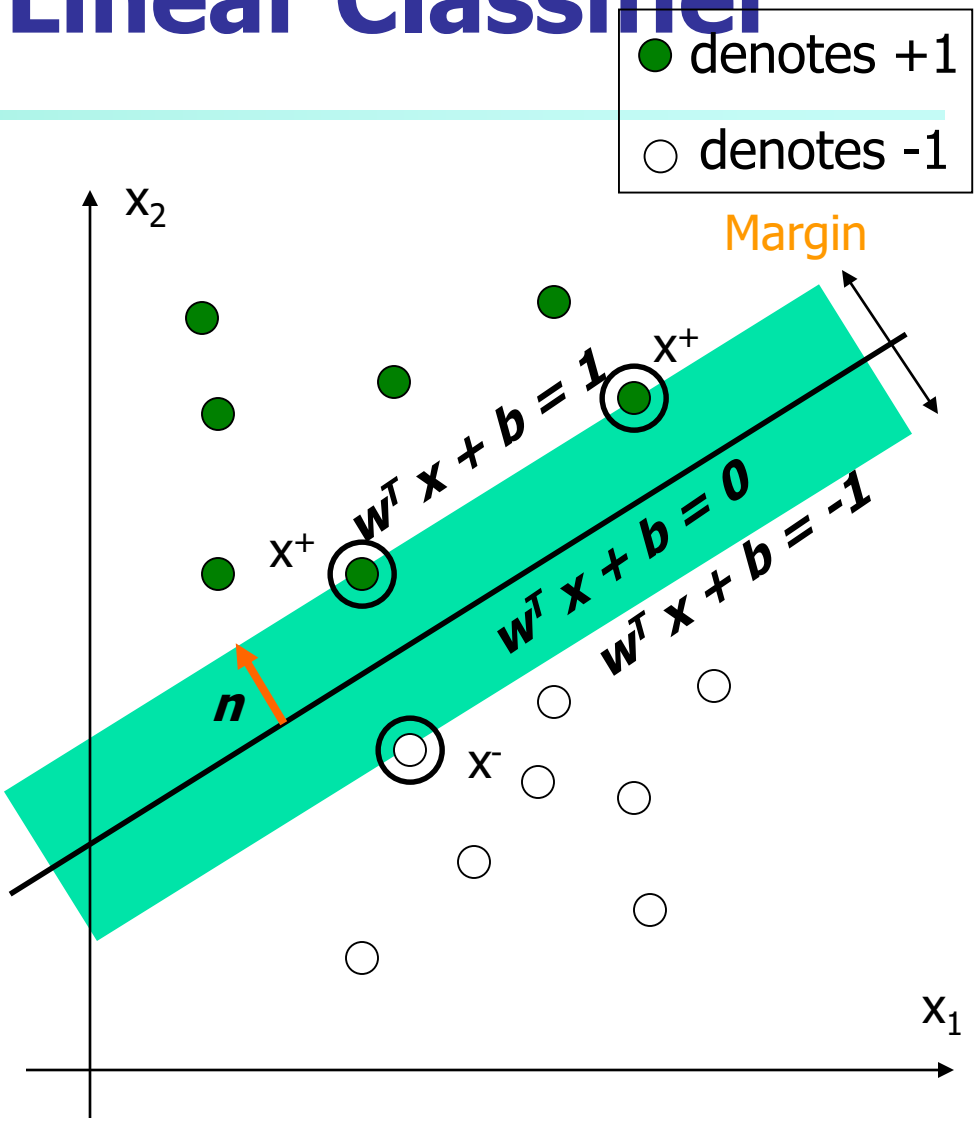
Large Margin Linear Classifier

- Formulation:

$$\text{minimize} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$



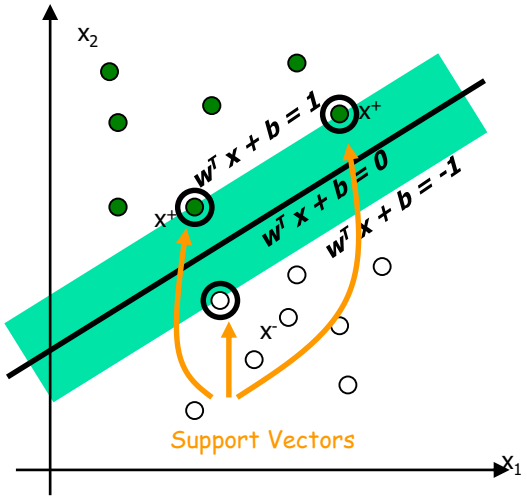
Solving the Optimization Problem

- The solution has the form:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = \sum_{i \in \mathrm{SV}} \alpha_i y_i \mathbf{x}_i$$

get b from $y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 = 0,$

where $\mathbf{x}_i$ is support vector



Solving the Optimization Problem

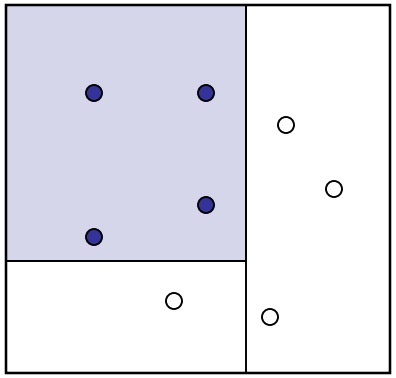
- The linear discriminant function is:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \sum_{i \in \mathrm{SV}} \alpha_i \mathbf{x}_i^T \mathbf{x} + b$$

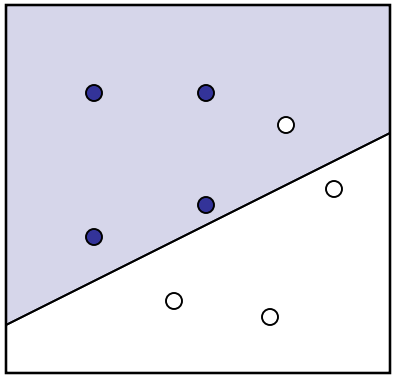
- Notice it relies on a *dot product* between the test point $\boldsymbol{x}$ and the support vectors $\boldsymbol{x_i}$

- Also keep in mind that solving the optimization problem involved computing the dot products $\boldsymbol{x_i^T x_j}$ between all pairs of training points

Classification: Discriminant Function

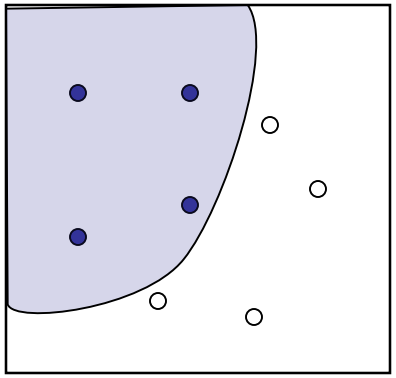


Decision Tree



Linear Functions

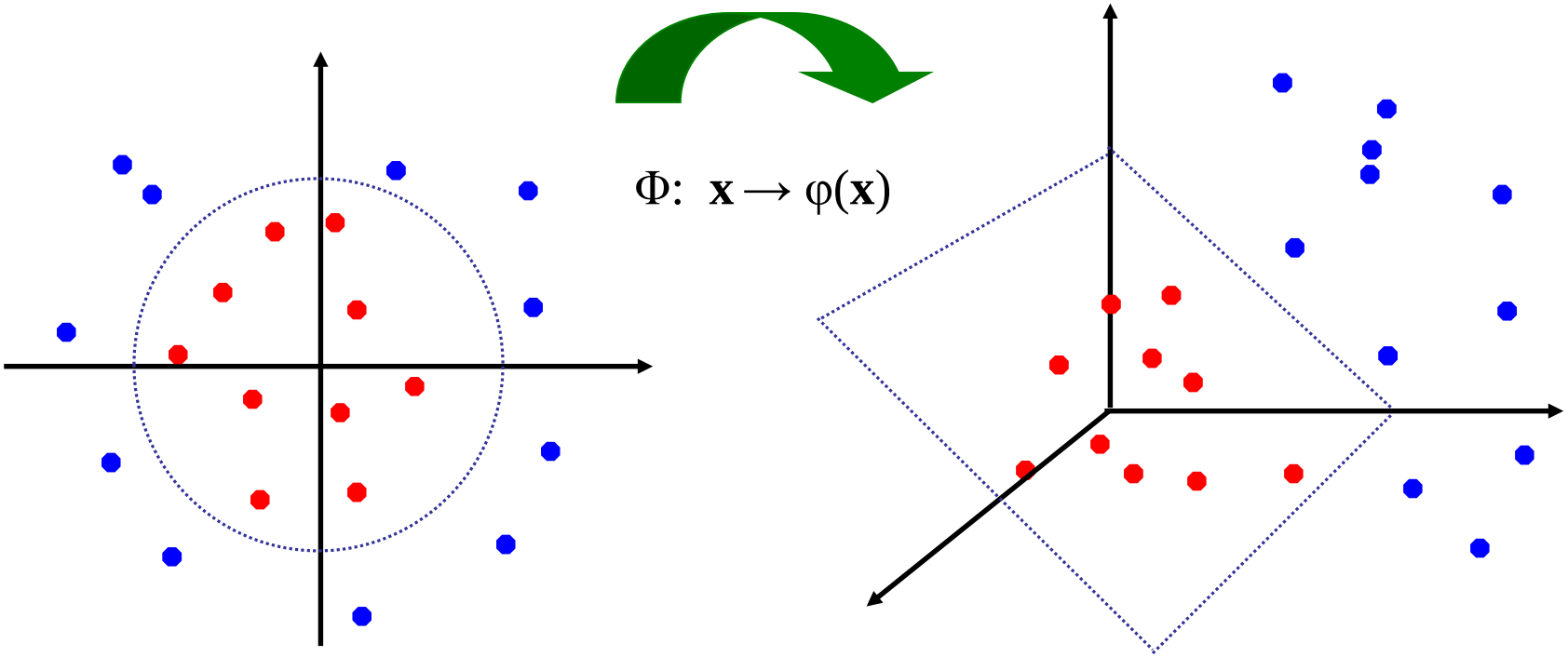
$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$



Nonlinear Functions

Non-linear SVMs: Feature Space

- General idea: the original input space can be mapped to some higher-dimensional feature space where the training set is separable:



This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Nonlinear SVMs: The Kernel Trick

- With this mapping, our discriminant function is now:

$$g(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + b = \sum_{i \in \text{SV}} \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b$$

- No need to know this mapping explicitly, because we only use the dot product of feature vectors in both the training and test.

- A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Nonlinear SVMs: The Kernel Trick

- Examples of commonly-used kernel functions:
  - Linear kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
  - Polynomial kernel: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
  - Gaussian (Radial-Basis Function (**RBF**)) kernel:

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

Summary: Support Vector Machine

- 1. Large Margin Classifier
  - Better generalization ability & less over-fitting

- 2. The Kernel Trick
  - Map data points to higher dimensional space in order to make them linearly separable.
  - Since only dot product is used, we do not need to represent the mapping explicitly.

Classification

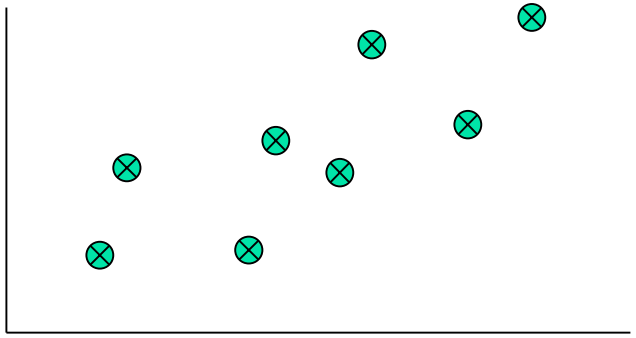
- Basic concepts
- Decision tree
- Naïve Bayesian classifier
- Model evaluation
- Support Vector Machines
- **Regression**
- Neural Networks and Deep Learning
- Lazy Learners (k Nearest Neighbors)
- Bayesian Belief Networks

Machine Learning Problems

	Supervised Learning	*Unsupervised Learning*
Discrete	classification or categorization	clustering
Continuous	regression	dimensionality reduction

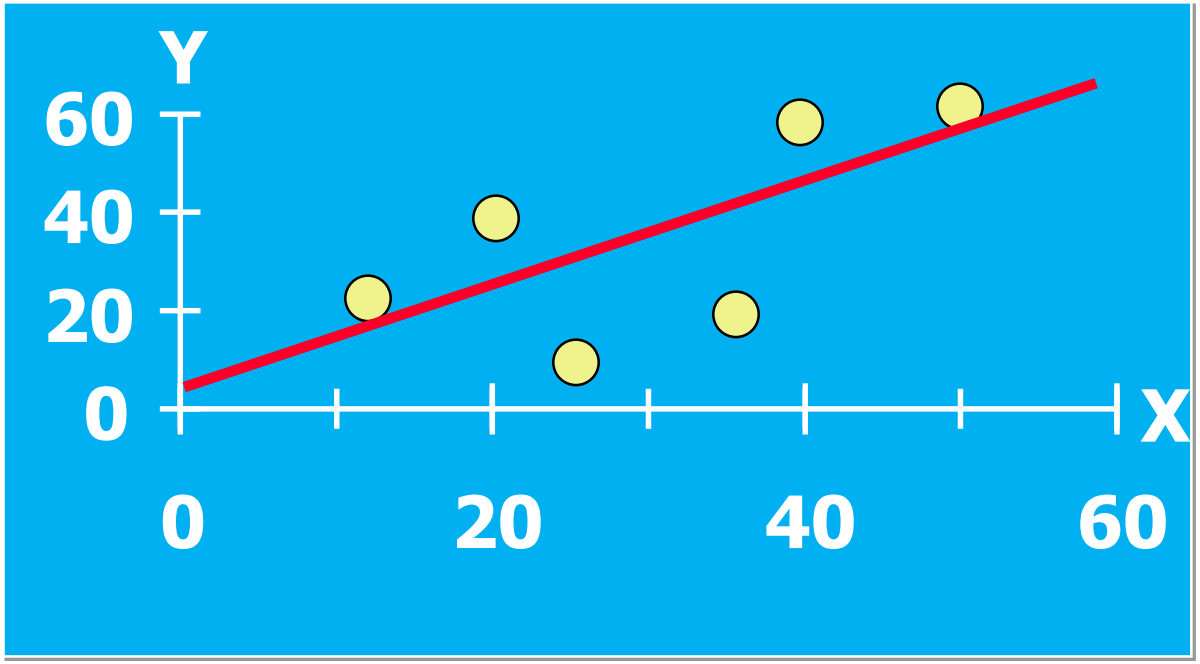
Regression

- Given input/output samples (X, y), we learn a function f such that y = f(X), which can be used on new data.
  - **Classification:** y is discrete (class labels).
  - **Regression:** y is continuous, e.g. linear regression.



Linear Regression

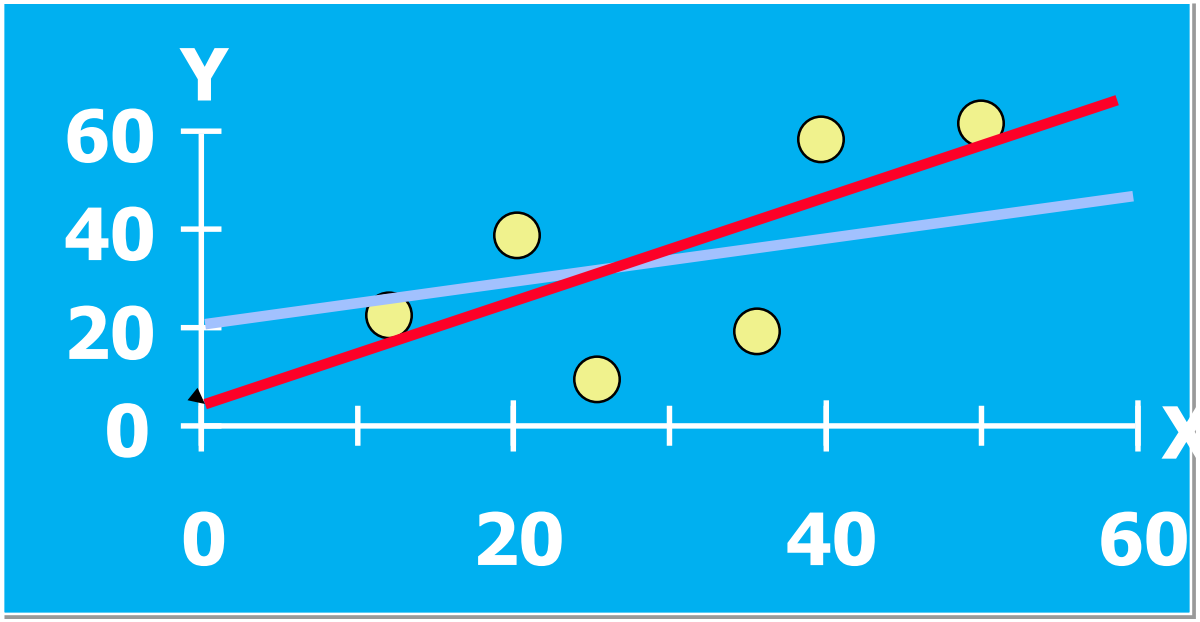
- Linear regression: use a linear function to model the relationship between a dependent (target) variable y and explanatory variables X.
- Simple linear regression: one explanatory variable



Linear Regression

How do you determine which line 'fits best'?

$$Y = \beta_0 + \beta_1 X$$



Least Squares

$$Y = \beta_0 + \beta_1 X$$

- 'Best Fit' can be defined by a cost function - Difference Between Actual Y Values & Predicted Y Values (residue)

- Least Squares Minimizes the Sum of the Squared Differences (errors) (SSE)

$$\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i \right)^2$$

Linear Regression

Data: Inputs are continuous vectors of length K. Outputs are continuous scalars.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \mathbb{R}$$

Prediction: Output is a linear function of the inputs.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_K x_K$$

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

(We assume x_1 is 1)

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

Least Squares

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

We minimize the sum of the squares:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Why?

Reduces distance between true measurements and predicted hyperplane (line in 1D)

Learning the parameters

- Closed form
  - set partial derivatives equal to zero and solve for parameters)
- Gradient descent (GD)
- Stochastic gradient descent (SGD)

Derivation of Parameters (1)

- Least Squares: Find β that minimize the objective function - squared error (SSE)

$$\sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \beta_1 x_i \right)^2$$

$$0 = \frac{\partial \sum \varepsilon_i^2}{\partial \beta_0} = \frac{\partial \sum \left(y_i - \beta_0 - \beta_1 x_i \right)^2}{\partial \beta_0}$$

$$= -2\left(n\bar{y} - n\beta_0 - n\beta_1 \bar{x} \right)$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Derivation of Parameters (2)

- Least Squares: Find β that minimize the objective function - squared error (SSE)

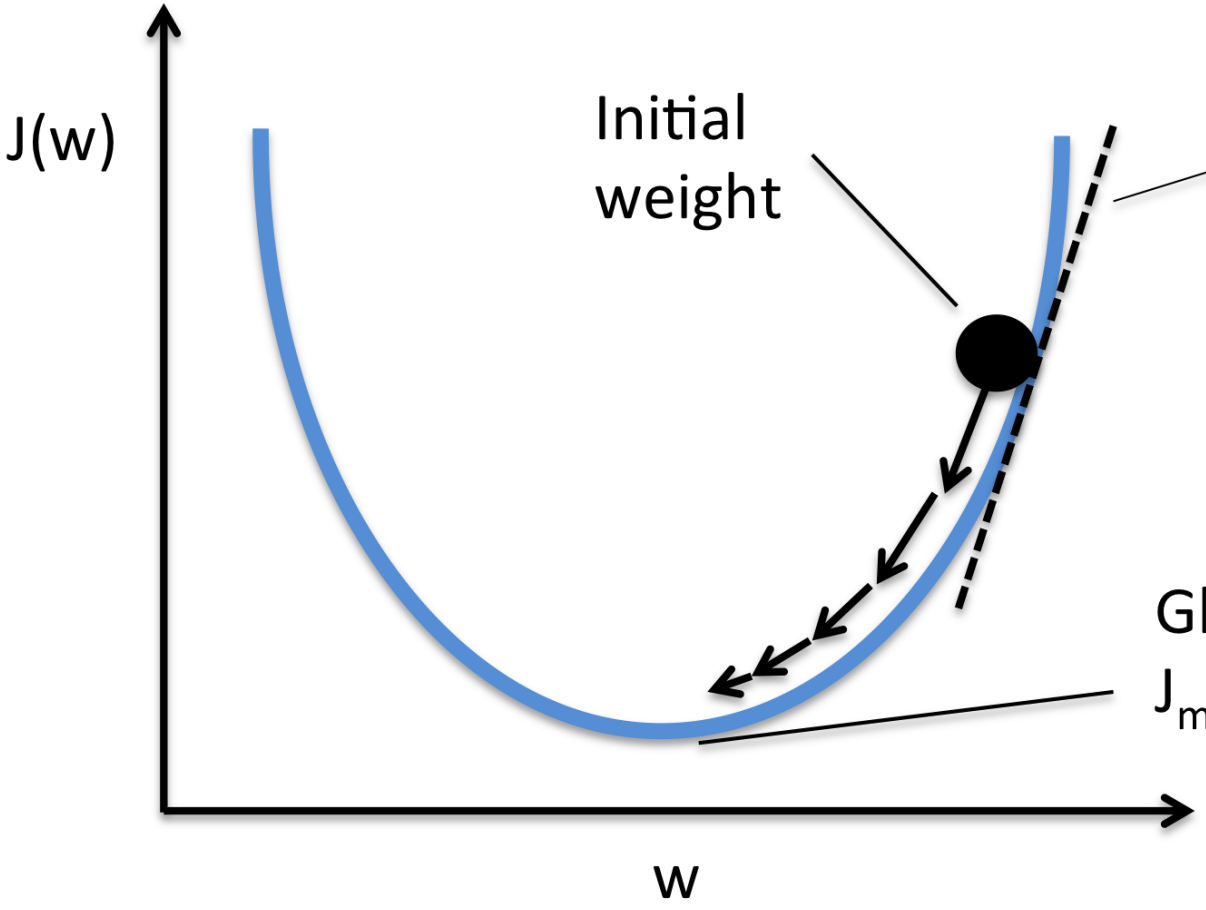
$$0 = \frac{\partial \sum \varepsilon_i^2}{\partial \beta_1} = \frac{\partial \sum (y_i - \beta_0 - \beta_1 x_i)^2}{\partial \beta_1}$$
$$= -2 \sum x_i (y_i - \beta_0 - \beta_1 x_i)$$
$$= -2 \sum x_i (y_i - \bar{y} + \beta_1 \bar{x} - \beta_1 x_i)$$

$$\beta_1 \sum x_i (x_i - \bar{x}) = \sum x_i (y_i - \bar{y})$$
$$\beta_1 \sum (x_i - \bar{x})(x_i - \bar{x}) = \sum (x_i - \bar{x})(y_i - \bar{y})$$
$$\hat{\beta}_1 = \frac{SS_{xy}}{SS_{xx}}$$

Learning the parameters

- Closed form
  - set partial derivatives equal to zero and solve for parameters
- Gradient descent (GD)
  - Start with initial values, gradually move to towards the minimal loss function value based on gradient
- Stochastic gradient descent (SGD)

Gradient Descent Illustration



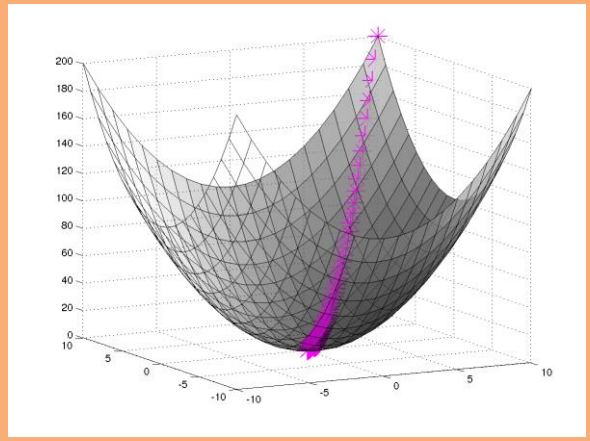
$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j},$$

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w},$$

Gradient Descent

Algorithm 1 Gradient Descent

1: **procedure** GD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$**while** not converged **do**
4: $\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: $\quad$**return** $\boldsymbol{\theta}$



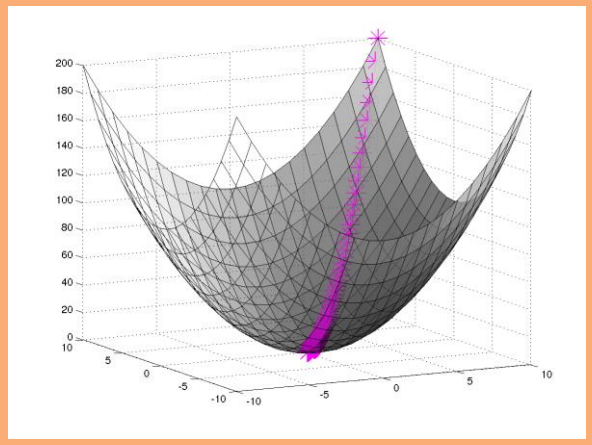
Gradient of the objective function (i.e. vector of partial derivatives):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_N} J(\boldsymbol{\theta}) \end{bmatrix}$$

Gradient Descent

Algorithm 1 Gradient Descent

1: **procedure** GD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$**while** not converged **do**
4: $\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: $\quad$**return** $\boldsymbol{\theta}$



Convergence. We could check whether the L2 norm of the gradient is below some small tolerance.

$$||\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})||_2 \leq \epsilon$$

Alternatively whethre the reduction in the objective function from one iteration to the next is small.

Partial Derivatives for Linear Reg.

Let $J(\boldsymbol{\theta}) = \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$

where $J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$.

$$
\begin{aligned}
\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) &= \frac{d}{d\theta_k} \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\
&= \frac{1}{2} \frac{d}{d\theta_k}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 \\
&= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \\
&= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \frac{d}{d\theta_k}(\sum_{k=1}^{K} \theta_k x_k^{(i)} - y^{(i)}) \\
&= (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}
\end{aligned}
$$

Partial Derivatives for Linear Reg.

Let $J(\boldsymbol{\theta}) = \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$

where $J^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$.

$$\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) = (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$

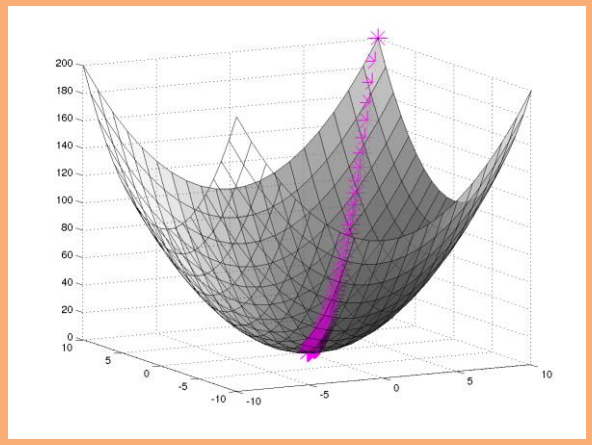
$$\frac{d}{d\theta_k} J(\boldsymbol{\theta}) = \frac{d}{d\theta_k} \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$

Gradient Descent for Least Sqaures

Algorithm 1 Gradient Descent

1: **procedure** GD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$**while** not converged **do**
4: $\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
5: $\quad$**return** $\boldsymbol{\theta}$



$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_N} J(\boldsymbol{\theta}) \end{bmatrix}$$

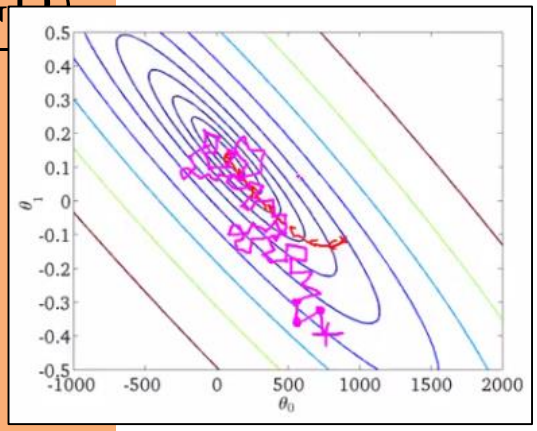
$$\frac{d}{d\theta_k} J(\boldsymbol{\theta}) = \frac{d}{d\theta_k} \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$$

$$= \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_k^{(i)}$$

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

1: **procedure** SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$**while** not converged **do**
4: $\quad\quad$**for** $i \in$ shuffle($\{1, 2, \ldots, N\}$) **do**
5: $\quad\quad\quad$**for** $k \in \{1, 2, \ldots, K\}$ **do**
6: $\quad\quad\quad\quad\theta_k \leftarrow \theta_k + \lambda \frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta})$
7: $\quad$**return** $\boldsymbol{\theta}$



Update parameters based on gradient of random data samples

Often preferred over gradient descent because it gets close to the minimum much faster

Non-Linear basis function

- So far we only used the observed values x_1,x_2,…
- However, linear regression can be applied in the same way to **functions** of these values
  - Eg: to add a new variable x_1^2 and x_1x_2 so each example becomes: x_1, x_2, …. x_1^2, x_1x_2
- As long as these functions can be directly computed from the observed values the parameters are still linear in the data and the problem remains a multi-variate linear regression problem

$$y = w_0 + w_1 x_1^2 + \ldots + w_k x_k^2 + \varepsilon$$

Non-linear basis functions

- What type of functions can we use?
- A few common examples:

- Polynomial: $\phi_j(x) = x^j$ for j=0 … n

- Gaussian: $f_j(x) = \frac{(x - \mu_j)}{2\sigma_j^2}$

- Sigmoid: $f_j(x) = \frac{1}{1 + \exp(-s_j x)}$

- Logs: $f_j(x) = \log(x + 1)$

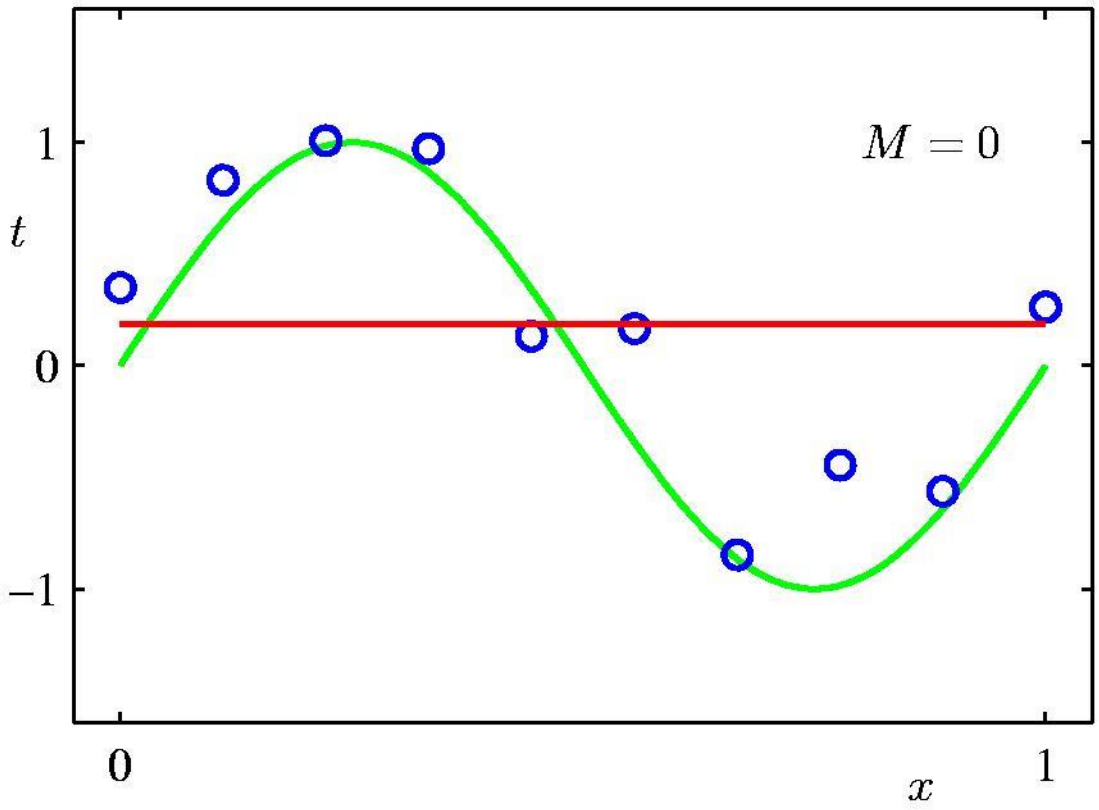
General linear regression problem

- Using our new notations for the basis function linear regression can be written as

$$y = \sum_{j=0}^{n} w_j f_j(x)$$

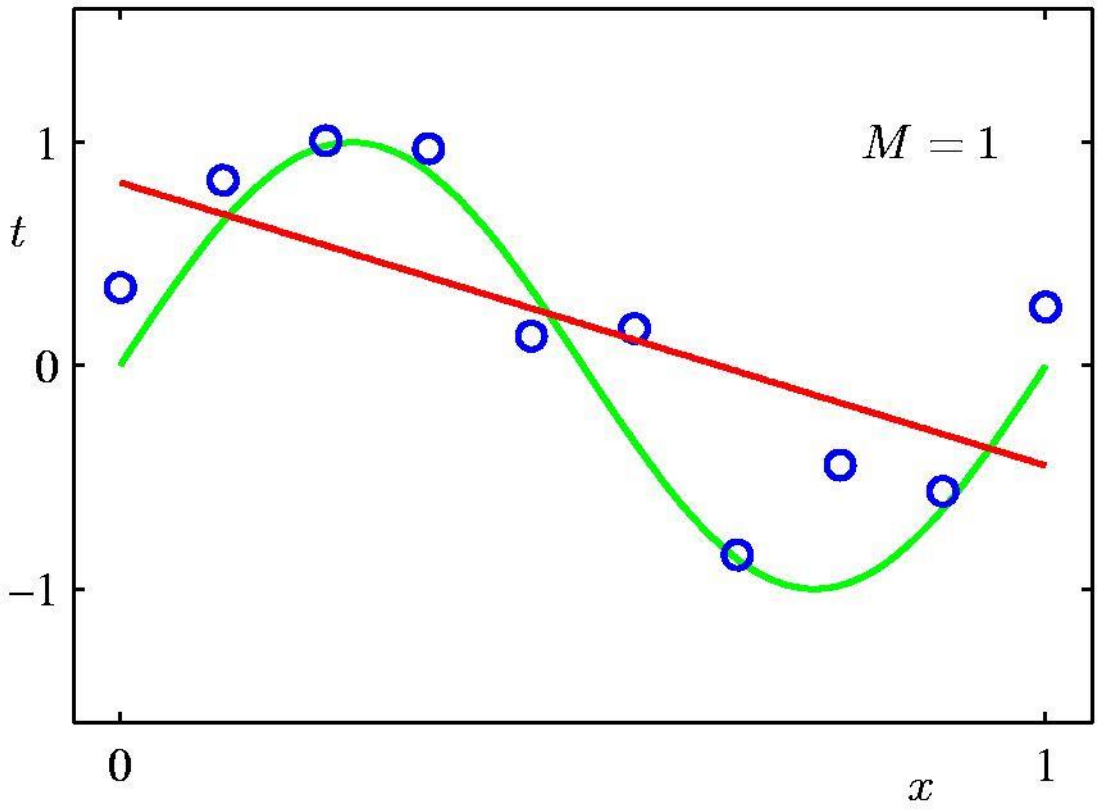
- Where $\phi_j(\mathbf{x})$ can be either x_j for multivariate regression or one of the non-linear basis functions we defined
- … and $\phi_0(\mathbf{x})=1$ for the intercept term

0th Order Polynomial

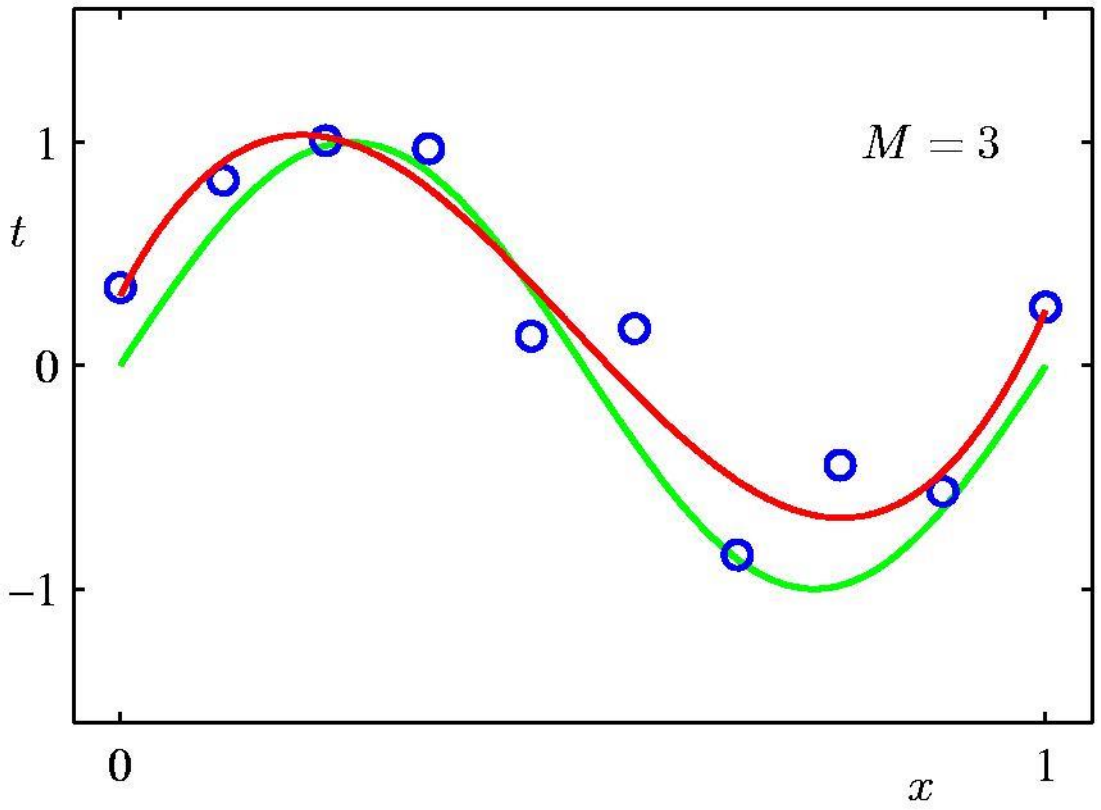


n=10

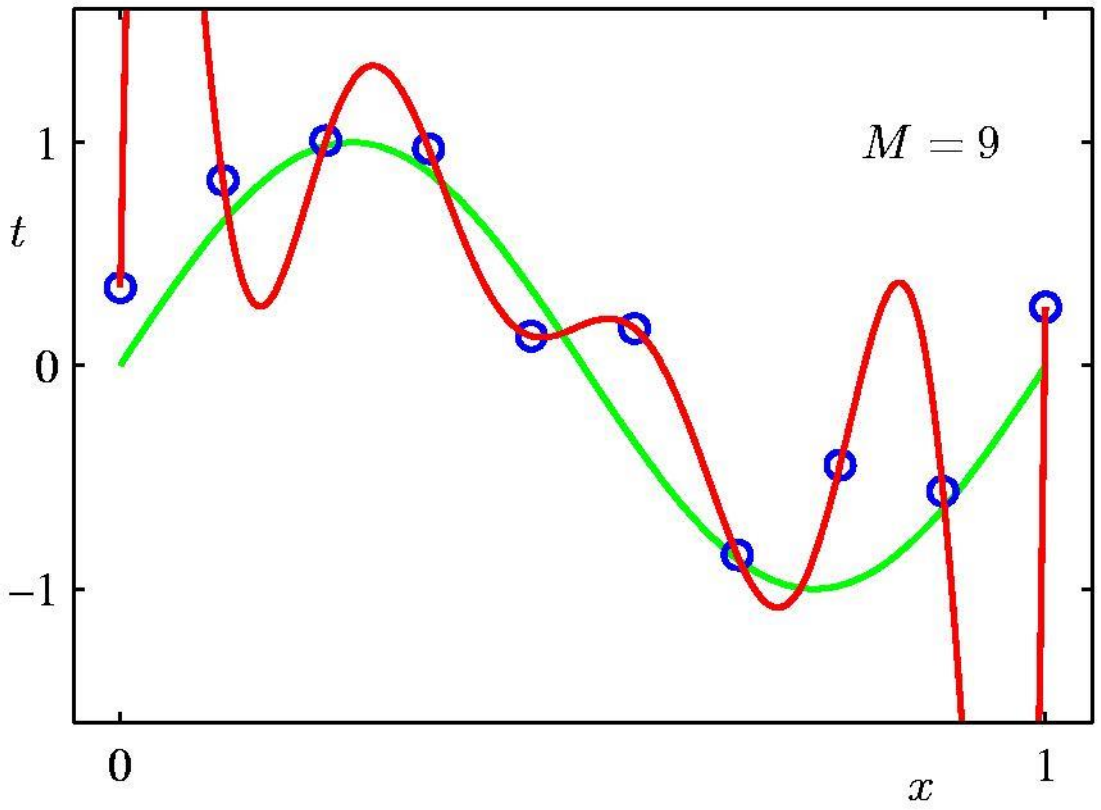
1st Order Polynomial



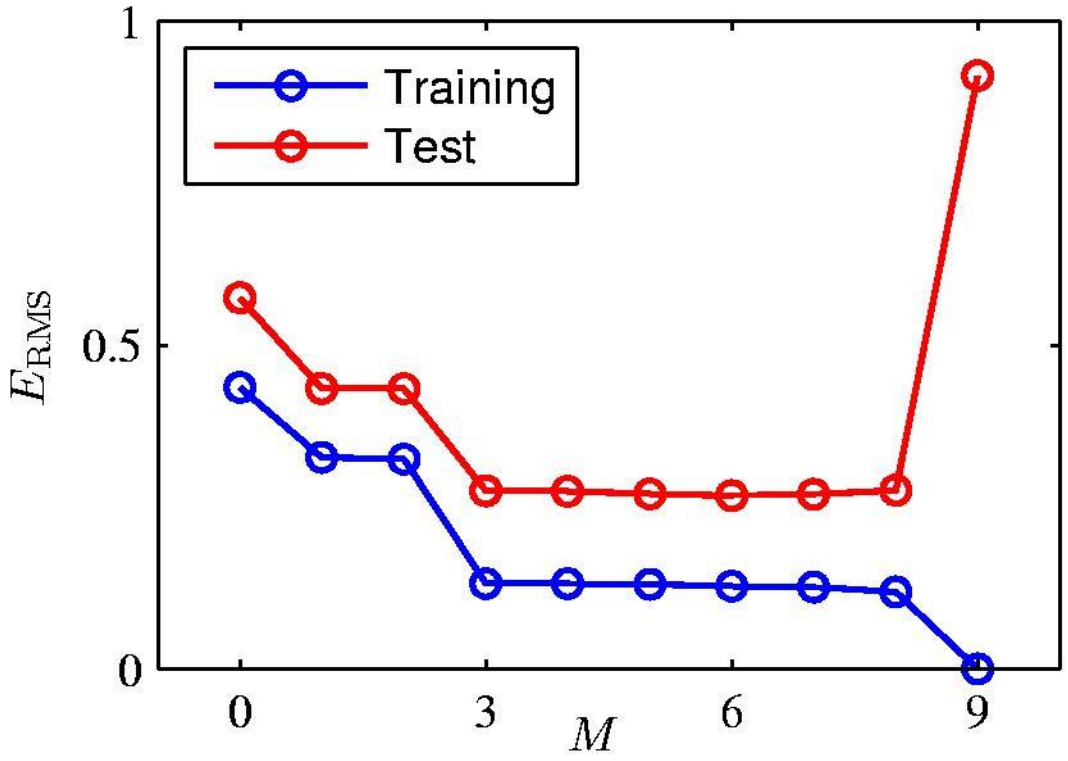
3rd Order Polynomial



9th Order Polynomial



Over-fitting



Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^{\star})/N}$

Polynomial Coefficients

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

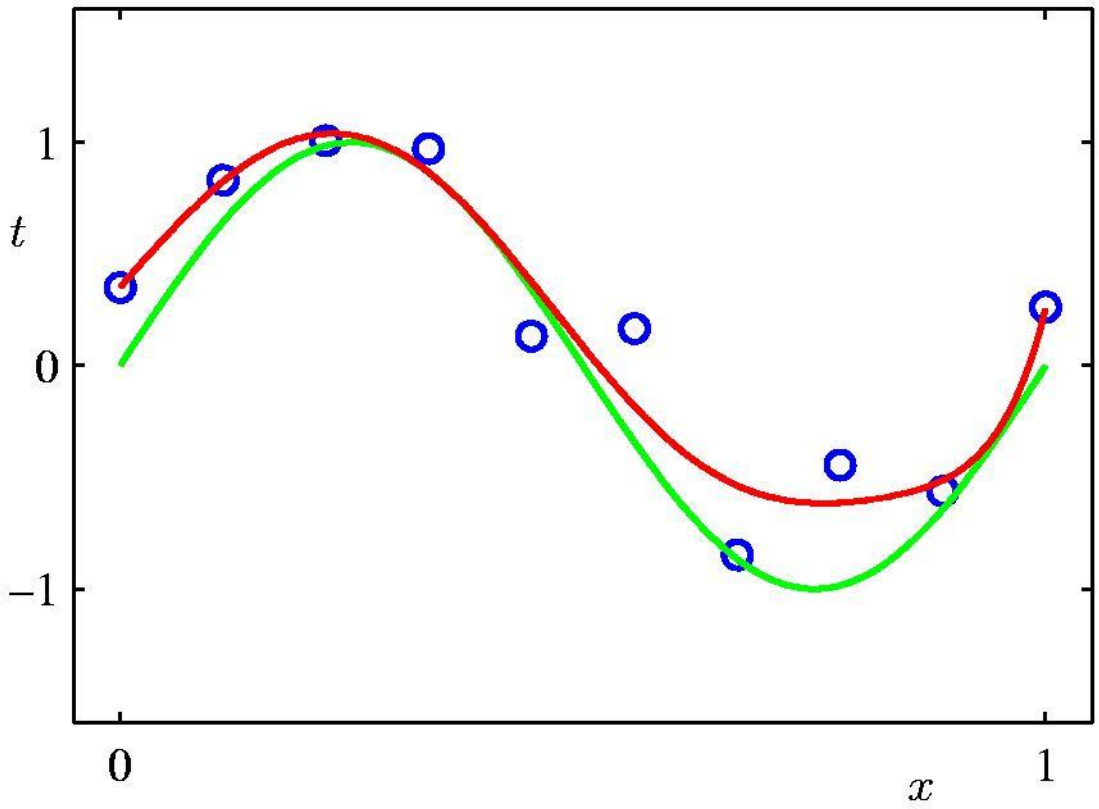
Regularization

Penalize large coefficient values

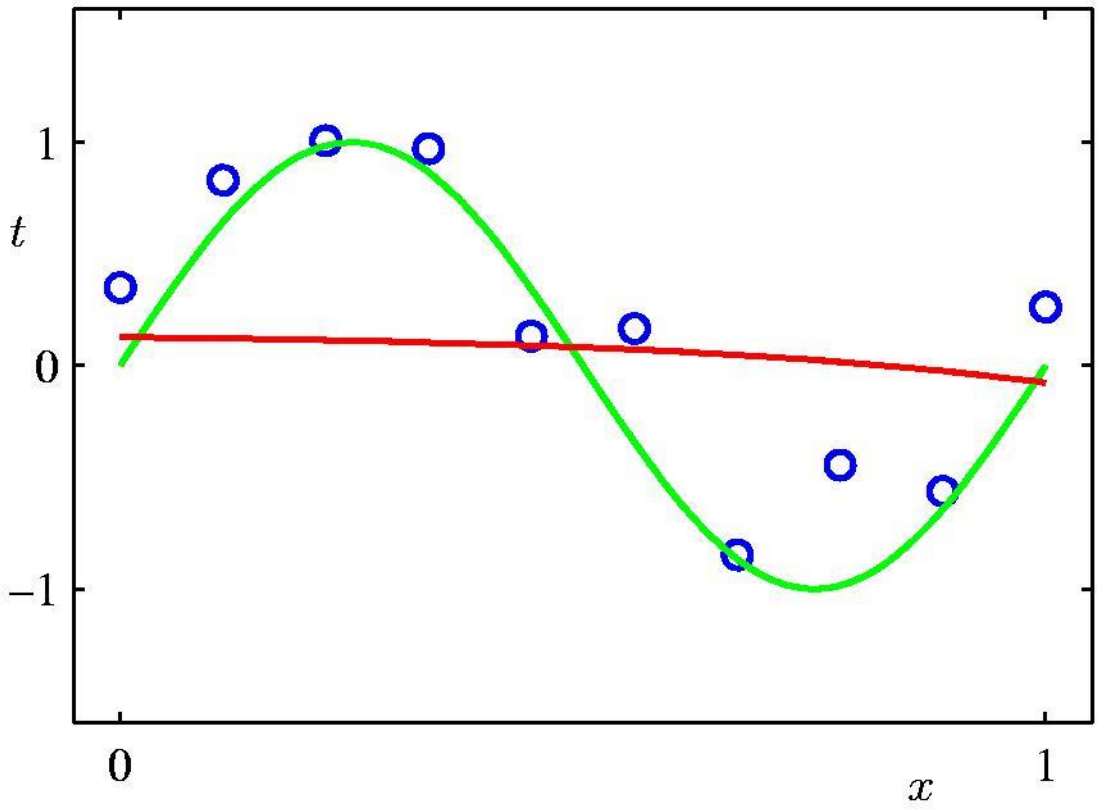
$$J_{\mathbf{X},\mathbf{y}}(\mathbf{w}) = \frac{1}{2}\sum_i \left(y^i - \sum_j w_j f_j(\mathbf{x}^i) \right)^2 - \frac{l}{2}\|\mathbf{w}\|^2$$

Regularization:

$\ln \lambda = +18$



Over Regularization:



Polynomial Coefficients

	none	exp(18)	huge
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

LASSO

- Adds an **L1 regularizer** to Linear Regression

$$J_{\text{LASSO}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda||\boldsymbol{\theta}||_1$$

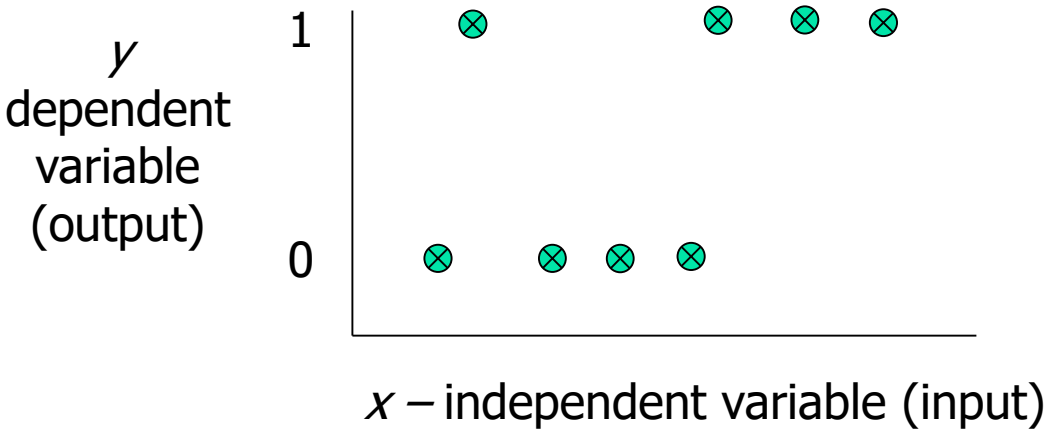
$$= \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})^2 + \lambda\sum_{k=1}^{K}|\theta_k|$$

Intepretability

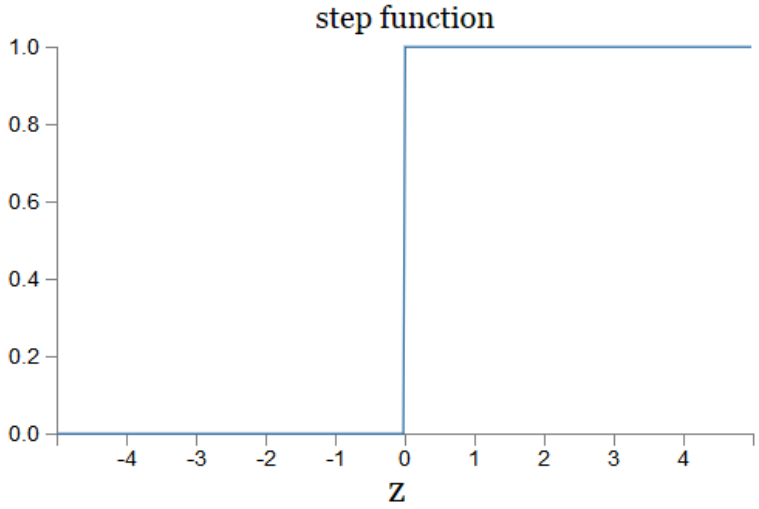
- Coefficients suggest importance/correlation with the output
  - A large positive coefficient implies that the output will increase when this input is increased (positively correlated)
  - A large negative coefficient implies that the output will decrease when this input is increased (negatively correlated)
  - A small or 0 coefficient suggests that the input is uncorrelated with the output (at least at the 1^{st} order)
- Linear regression can be used to find best "indicators"

Regression for Classification

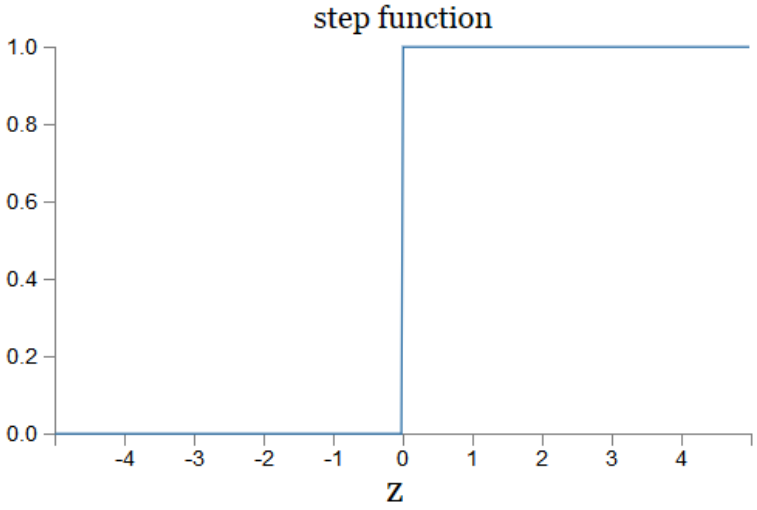
- Given input/output samples (X, y), we learn a function f such that y = f(X), which can be used on new data.
  - **Classification:** y is discrete (class labels).
  - **Regression:** y is continuous, e.g. linear regression.

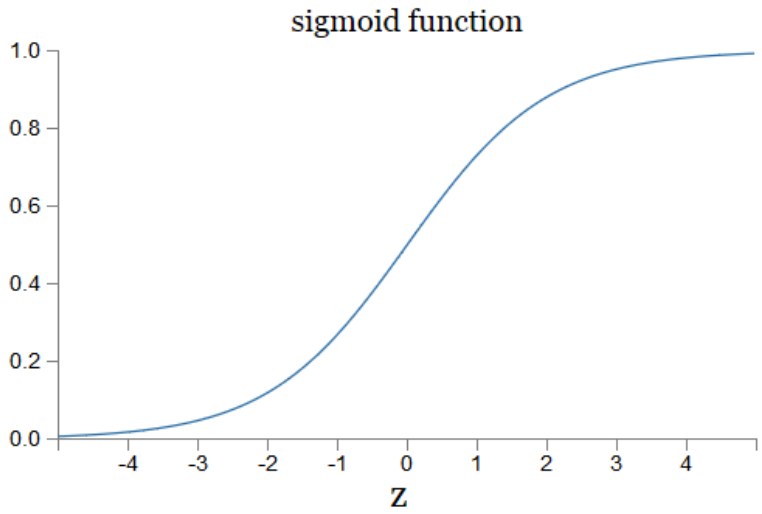


From real value to discrete value

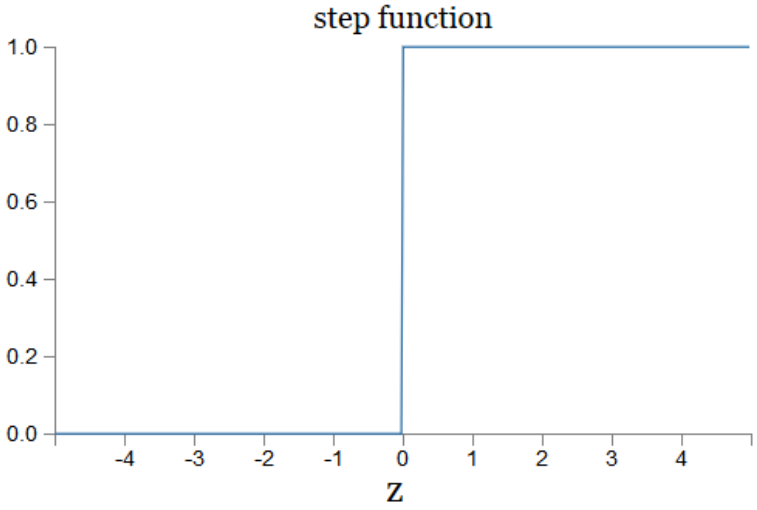


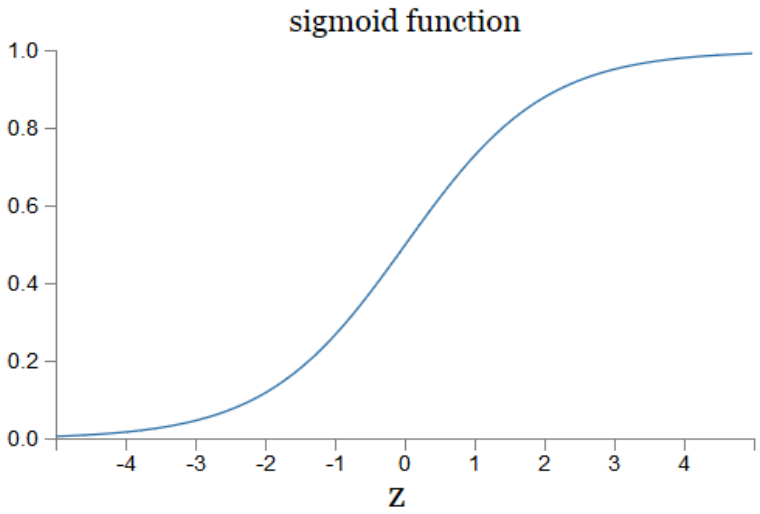
From real value to discrete value





From real value to discrete value





Non-differentiable

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\
&= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z)).
\end{aligned}
$$

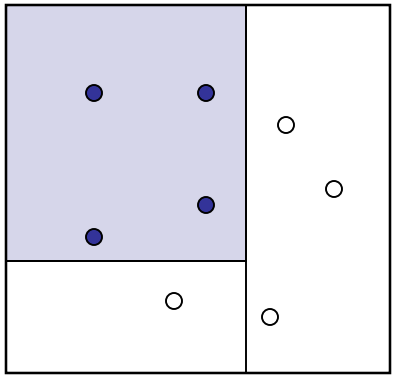
Logistic Regression

Data: Inputs are continuous vectors of length K. Outputs are discrete valued.

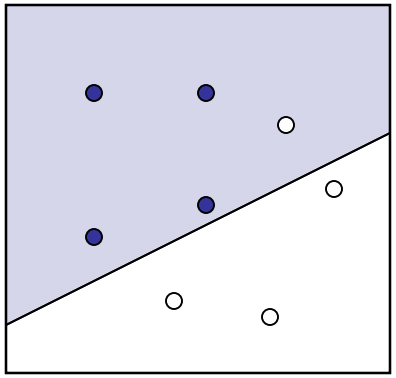
$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

Prediction: Output is a logistic function of the linear function of the inputs. $\hat{y} = h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$

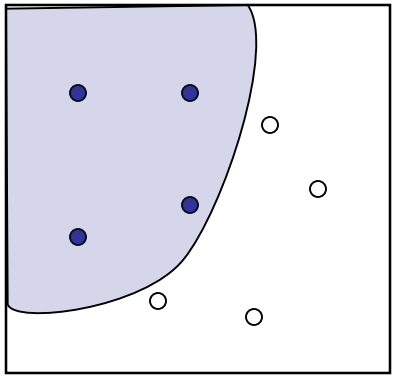
Classification: Discriminant Function



Decision Tree



Linear Functions



Nonlinear Functions

Logistic Regression

Data: Inputs are continuous vectors of length K. Outputs are discrete valued.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

Prediction: Output is a logistic function of the linear function of the inputs. $\hat{y} = h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$

Learning: finds the parameters that minimize some objective function.

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

Recall: Least Squares for Linear Regression

Learning: finds the parameters that minimize some objective function**.**

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

We minimize the sum of the squares:

$$J(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Why?
Reduces distance between true measurements and predicted values

Maximum Likelihood for Logistic Regression

Learning: finds the parameters that maximizes the log likelihood of observing the data:

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X; \theta) \\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}}
\end{aligned}
$$

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))
\end{aligned}
$$

Review: Derivative Rules

Common Functions	Function	Derivative
Constant	c	0
Line	x	1
	ax	a
Square	x^2	2x
Square Root	$\sqrt{x}$	$(\frac{1}{2})x^{-\frac{1}{2}}$
Exponential	e^x	e^x
	a^x	$\ln(a)\, a^x$
Logarithms	$\ln(x)$	$1/x$
	$\log_a(x)$	$1 / (x \ln(a))$
Trigonometry (x is in radians)	$\sin(x)$	$\cos(x)$
	$\cos(x)$	$-\sin(x)$
	$\tan(x)$	$\sec^2(x)$
Inverse Trigonometry	$\sin^{-1}(x)$	$1/\sqrt{(1-x^2)}$
	$\cos^{-1}(x)$	$-1/\sqrt{(1-x^2)}$
	$\tan^{-1}(x)$	$1/(1+x^2)$

Rules	Function	Derivative
Multiplication by constant	cf	cf′
Power Rule	x^n	nx^{n-1}
Sum Rule	f + g	f′ + g′
Difference Rule	f - g	f′ – g′
Product Rule	fg	f g′ + f′ g
Quotient Rule	f/g	$(f' g - g' f)/g^2$
Reciprocal Rule	1/f	$-f'/f^2$
Chain Rule (as "Composition of Functions")	f ∘ g	(f′ ∘ g) × g′
Chain Rule (using ′)	f(g(x))	f′(g(x))g′(x)
Chain Rule (using $\frac{d}{dx}$)	$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$	

Stochastic Gradient Descent (Ascent)

Gradient Descent Update: $\theta := \theta + \alpha \nabla_\theta \ell(\theta)$

Partial derivative with one training example (x,y):

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1 - g(\theta^T x)) - (1-y) g(\theta^T x) \right) x_j \\
&= \left(y - h_\theta(x) \right) x_j
\end{aligned}$$

Stochastic Gradient Descent Update:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Regression Summary

- Regression methods
  - Linear regression
  - Logistic regression
- Optimization
  - Gradient descent
  - Stochastic gradient descent
- Regularization

Classification

- Basic concepts
- Decision tree
- Naïve Bayesian classifier
- Model evaluation
- Support Vector Machines
- Regression
- **Neural Networks and Deep Learning**
- Lazy Learners (k Nearest Neighbors)
- Bayesian Belief Networks

Deep Learning: MIT Technology Review - 10 Breakthrough Technologies 2013

10 Breakthrough Technologies The List + Past Lists +

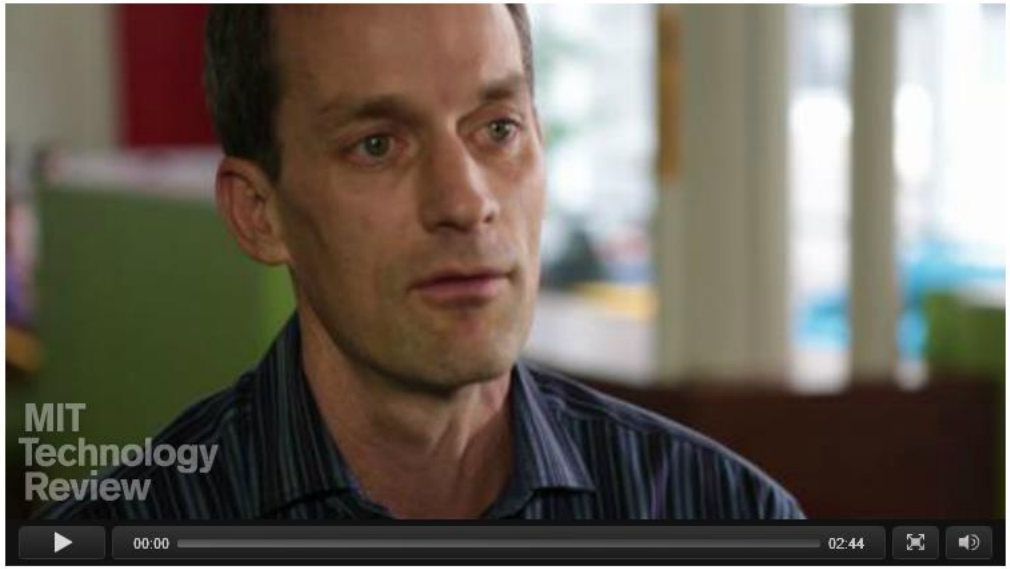
01 **Deep Learning** With massive amounts of
02 Smart Watches The designers of the Pebble watch
03 Ultra-Efficient Solar Power Doubling the efficiency of solar
04 Memory Implants A maverick neuroscientist

Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.

Advertisement

When Ray Kurzweil met with Google CEO Larry Page last July, he wasn't looking for a job. A respected inventor who's become a machine-intelligence futurist, Kurzweil wanted to discuss his upcoming book *How to Create a Mind*. He told Page, who had read an early draft, that he wanted to start a company to develop his ideas about how to build a truly intelligent computer: one that could understand language and then make inferences and decisions on its own.



nature International weekly journal of science

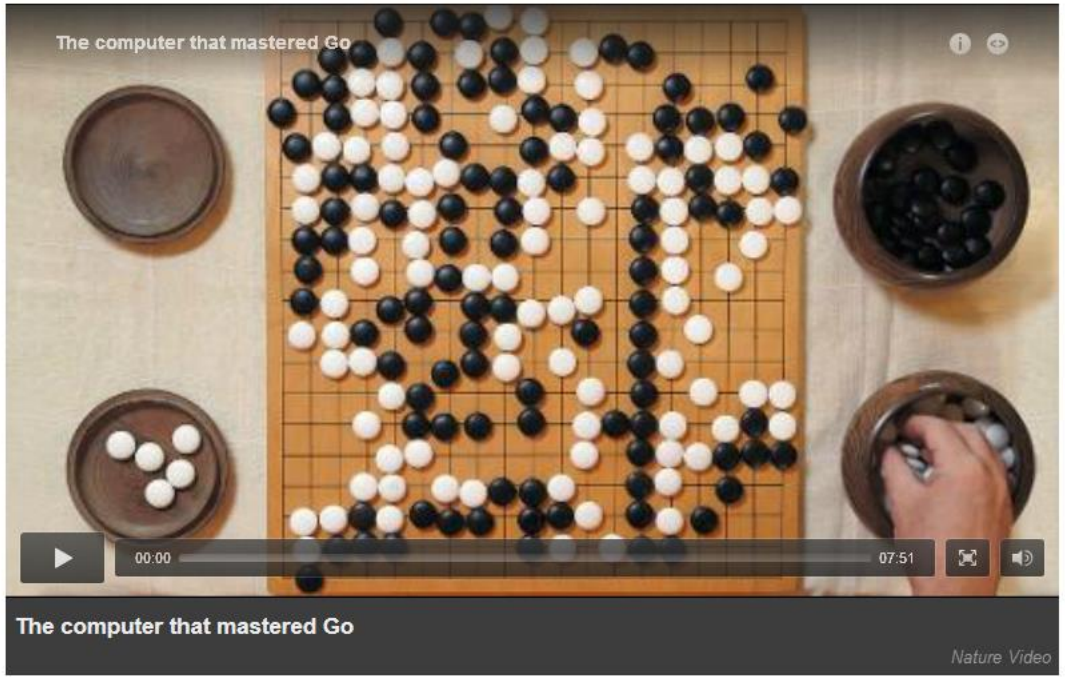
NATURE | NEWS

Google AI algorithm masters ancient game of Go

Deep-learning software defeats human professional for first time.

Elizabeth Gibney

27 January 2016



Applications

- Image recognition
- Speech recognition
- Natural language processing

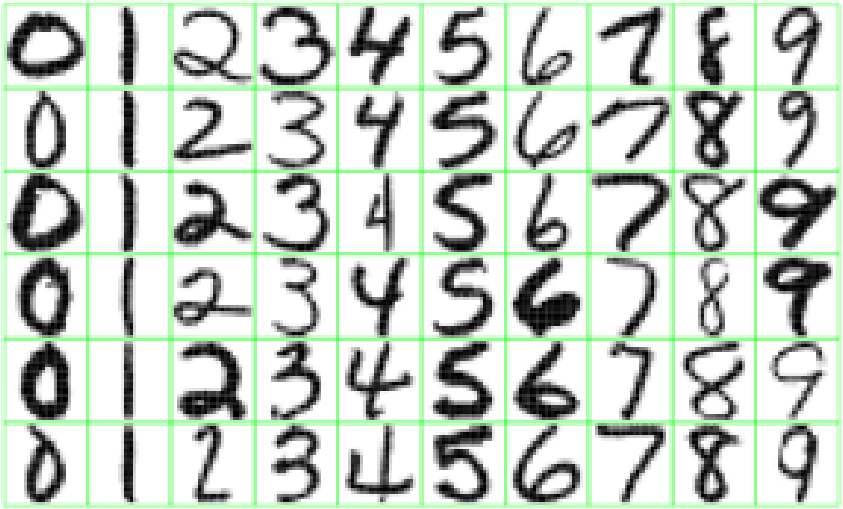
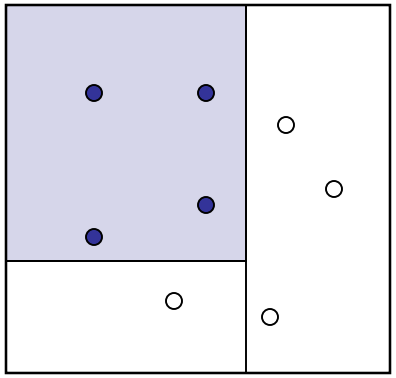
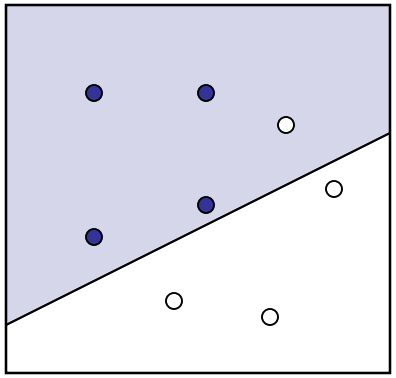


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

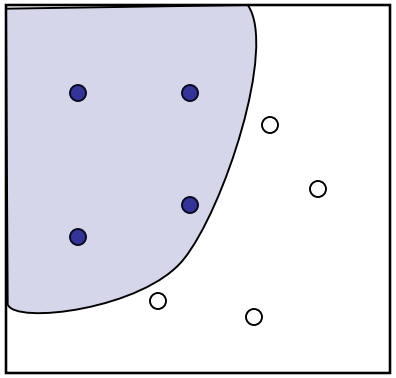
Classification: Discriminant Function



Decision Tree

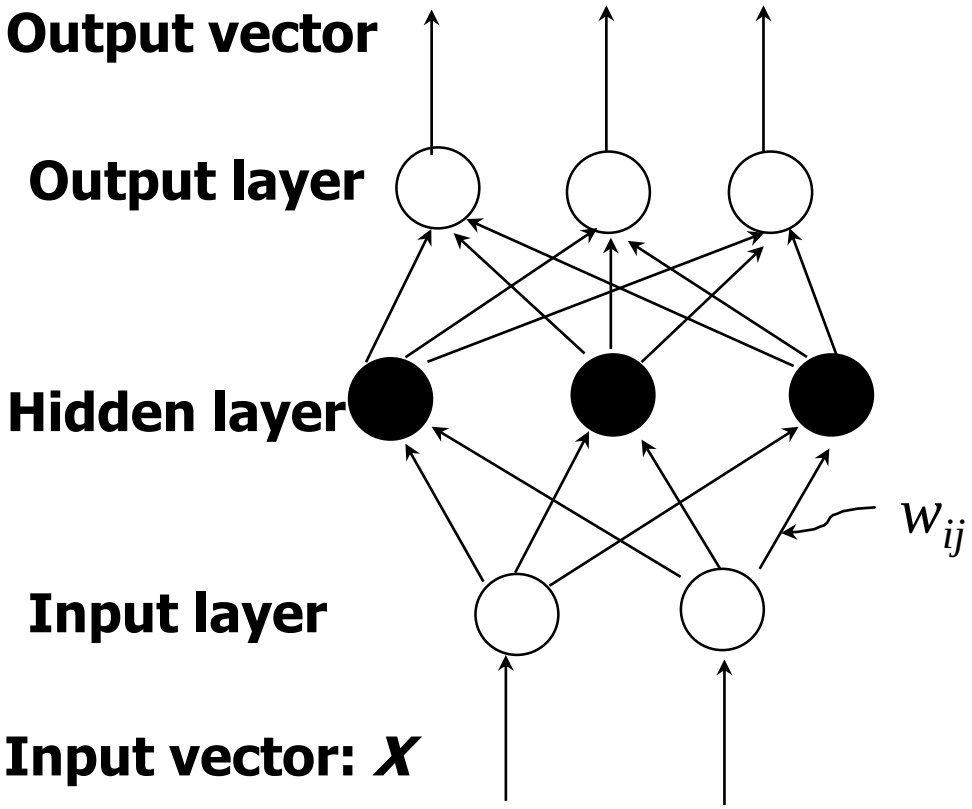


Linear Functions



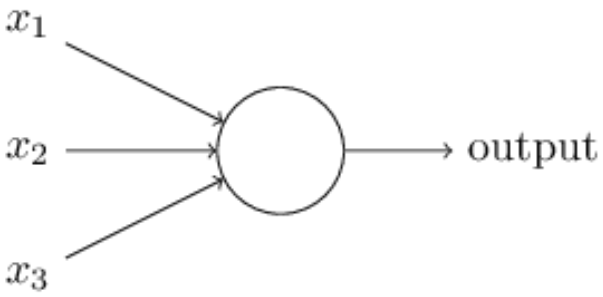
Nonlinear Functions

Neural Network and Deep Learning

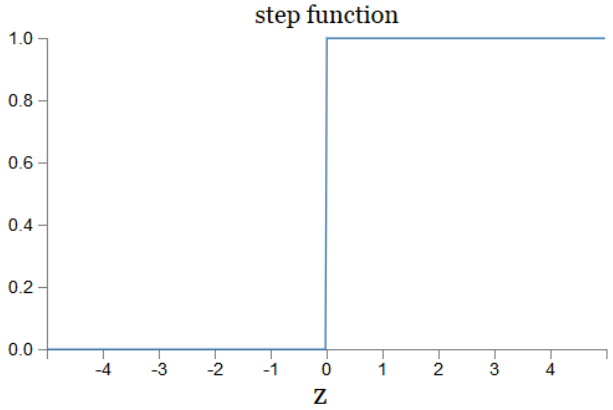


- A neural network is a multi-layer structure of connected input/output units (artificial neurons)
- Learning by adjusting the weights so as to predict the correct class label of the input tuples
- Deep learning use more layers than shallow learning

Artificial Neuron – Perceptron

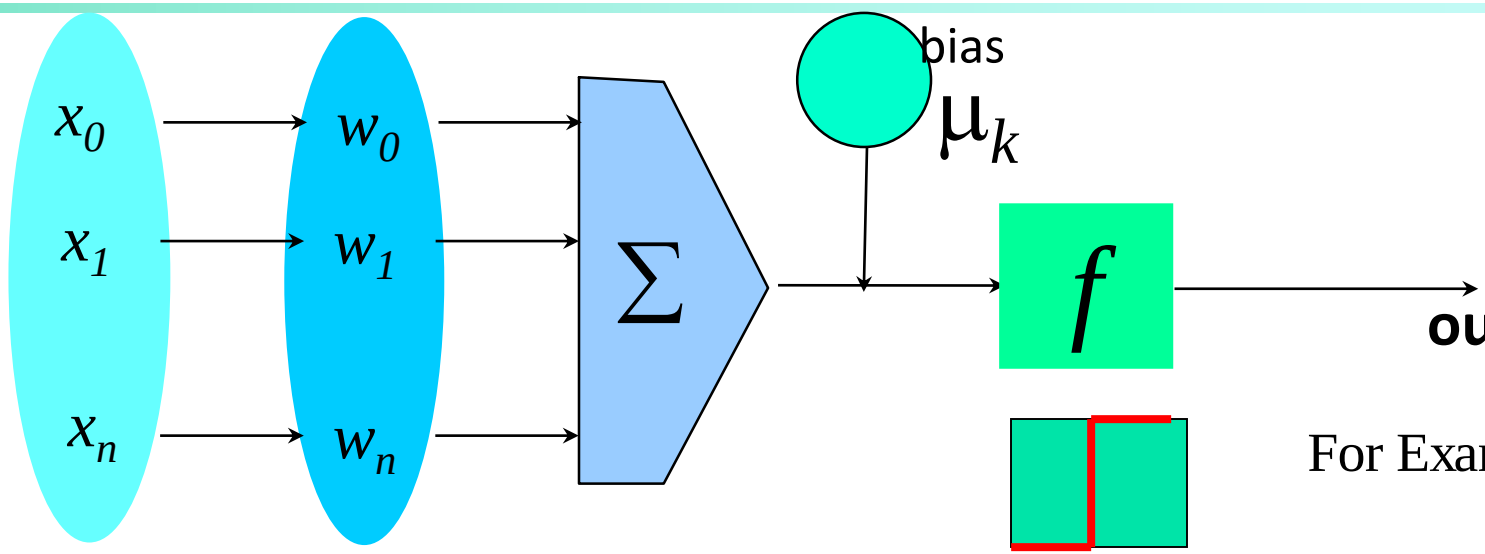


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



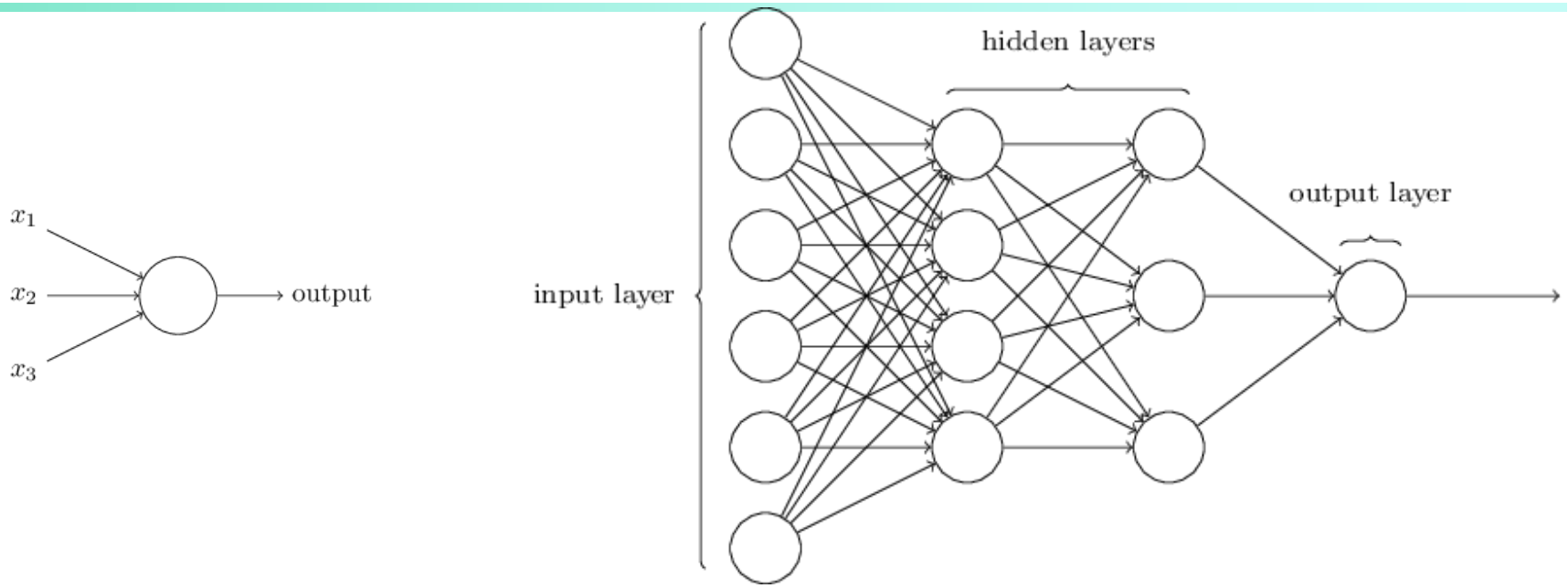
Perceptron

Neuron: A Hidden/Output Layer Unit



- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping
- The inputs to unit are outputs from the previous layer. They are multiplied by their corresponding weights to form a weighted sum, which is added to the bias associated with unit. Then a nonlinear activation function is applied to it.

From Neuron to Neural Network



- The **input layer** correspond to the attributes measured for each training tuple
- They are then weighted and fed simultaneously to **hidden layers**
- The weighted outputs of the last hidden layer are input to **output layer**, which emits the network's prediction
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

Neural Networks

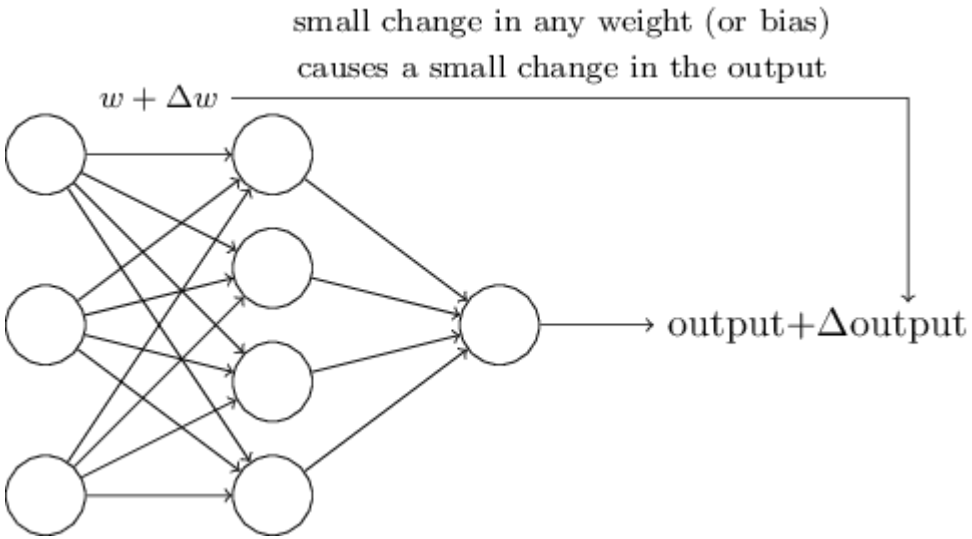
- A family of parametric, non-linear, and hierarchical representation learning functions

- $a_L\left(x; \theta_{1,\ldots,\mathrm{L}}\right) = h_L\left(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}),\ \theta_L\right)$
  - x:input, θ_l: parameters for layer l, $a_l = h_l(x, \theta_l)$: (non-)linear function
- Given training corpus $\{X, Y\}$ find optimal parameters

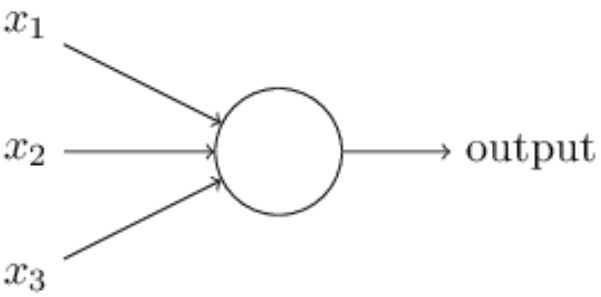
$$\theta^* \leftarrow \arg\min_\theta \sum_{(x,y)\subseteq(X,Y)} \ell\left(y, a_L\left(x; \theta_{1,\ldots,\mathrm{L}}\right)\right)$$

Learning Weights (and Bias) in the Network

- If a small change in a weight (or bias) cause only a small change in output, we could modify weights and biases gradually to train the network
- Does the perceptron work?



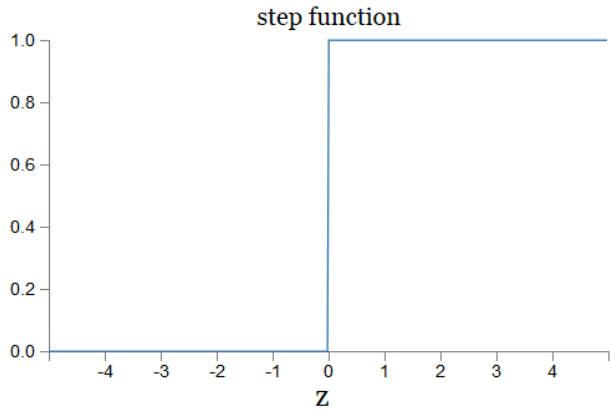
Artificial Neuron – Perceptron to sigmoid neuron

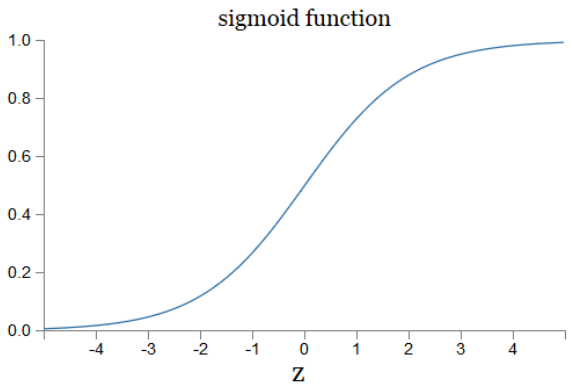


$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$



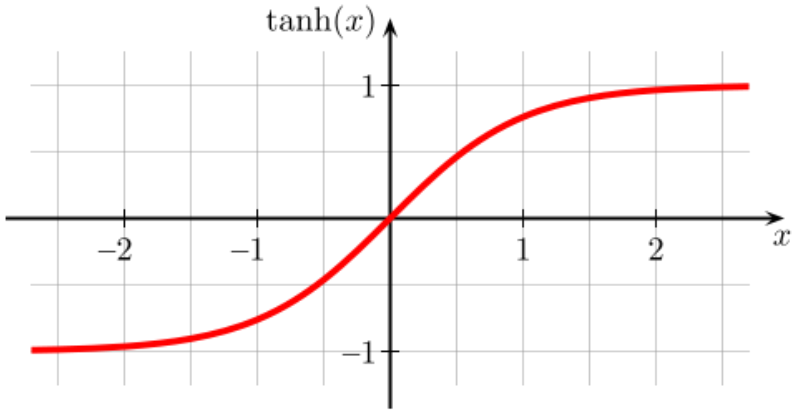


Perceptron

sigmoid neuron

Popular Activation Functions

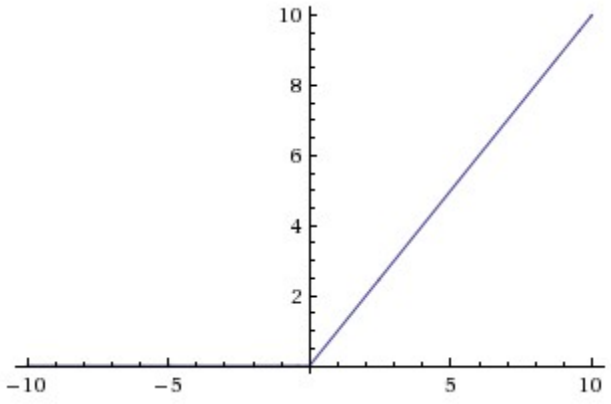
Tanh(x)



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

$$\tanh(x) = 2\sigma(2x) - 1$$

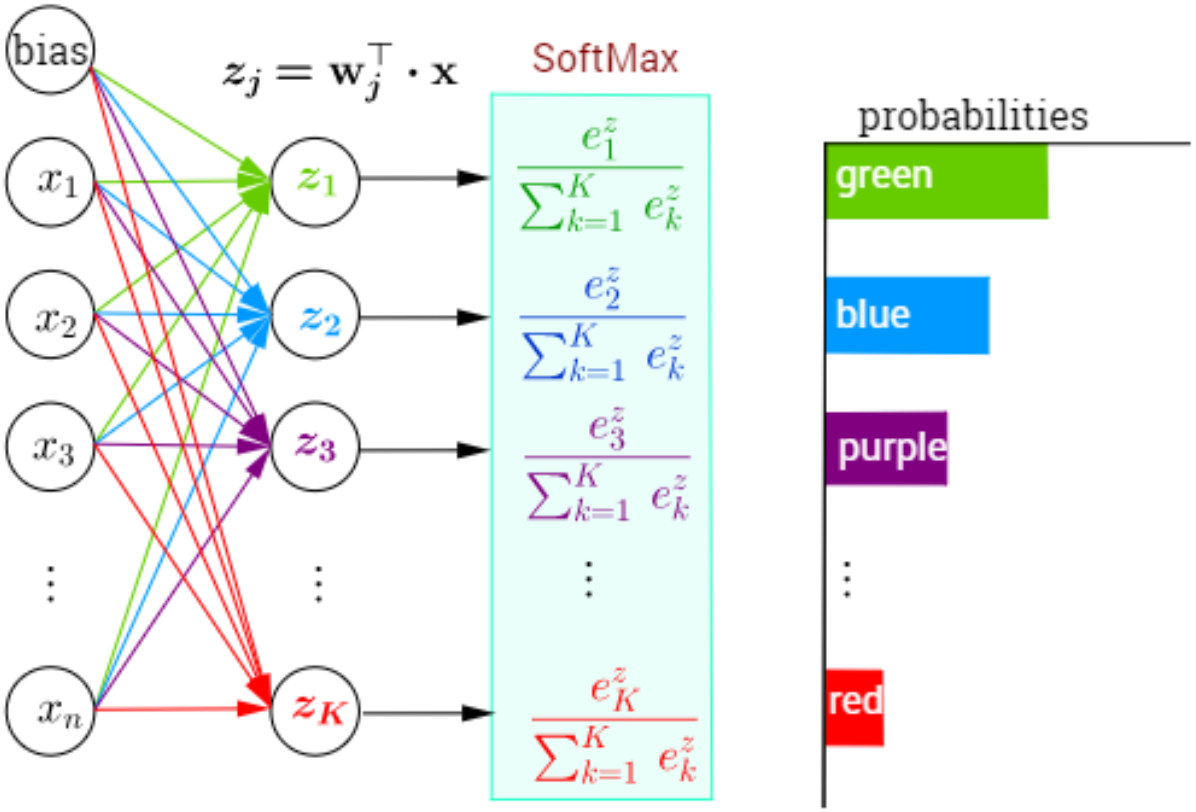
ReLU (Rectified Linear Unit)



$$f(x) = \max(0, x)$$

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Learning by Backpropagation

- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights to small random numbers, associated with biases
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

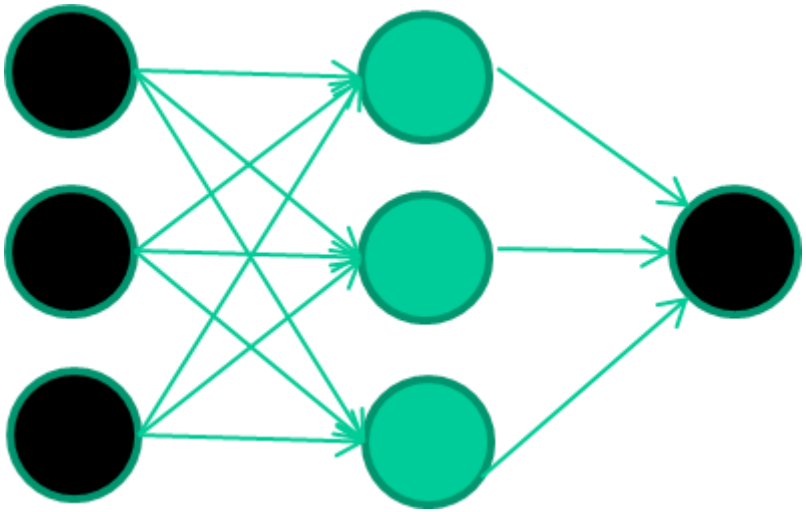
Stochastic Gradient Descent

- Gradient Descent (batch GD)
  - The cost gradient is based on the complete training set, can be costly and longer to converge to minimum
- Stochastic Gradient Descent (SGD, iterative or online-GD)
  - Update the weight after each training sample
  - The gradient based on a single training sample is a stochastic approximation of the true cost gradient
  - Converges faster but the path towards minimum may zig-zag
- Mini-Batch Gradient Descent (MB-GD)
  - Update the weights based on small group of training samples

Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

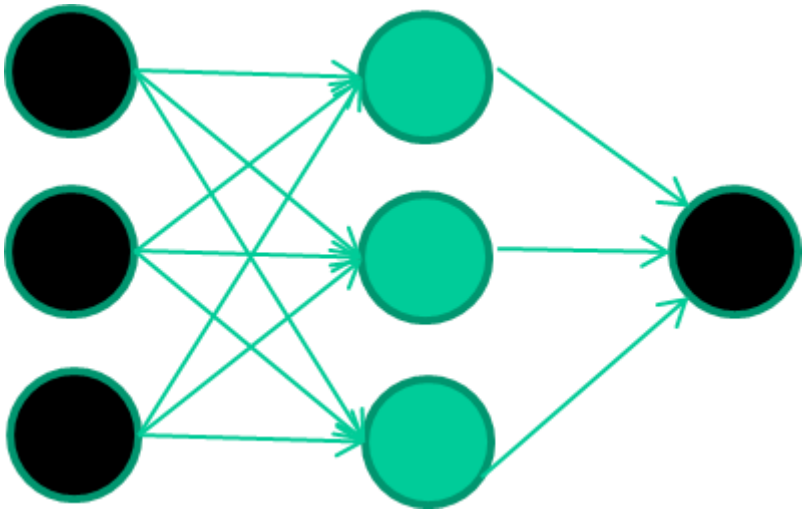
etc …



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

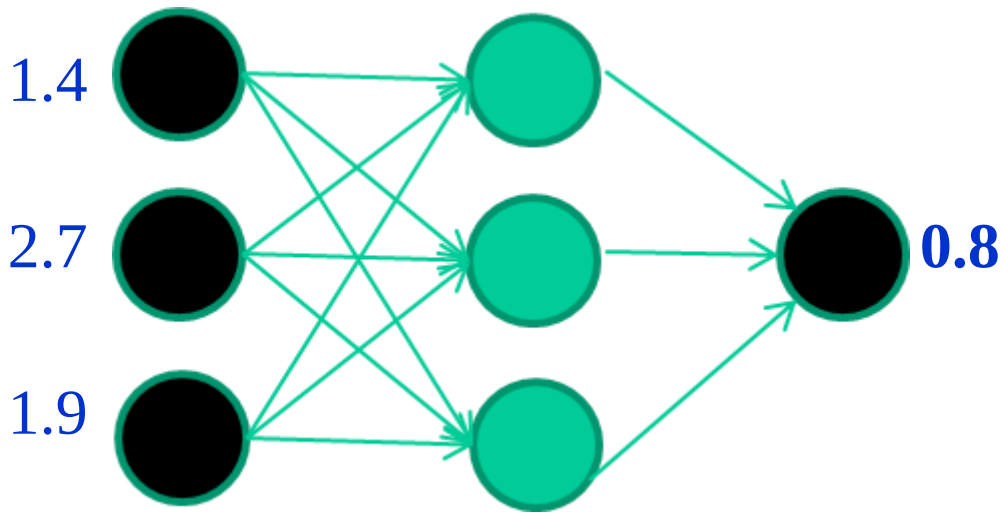
Initialise with random weights



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output

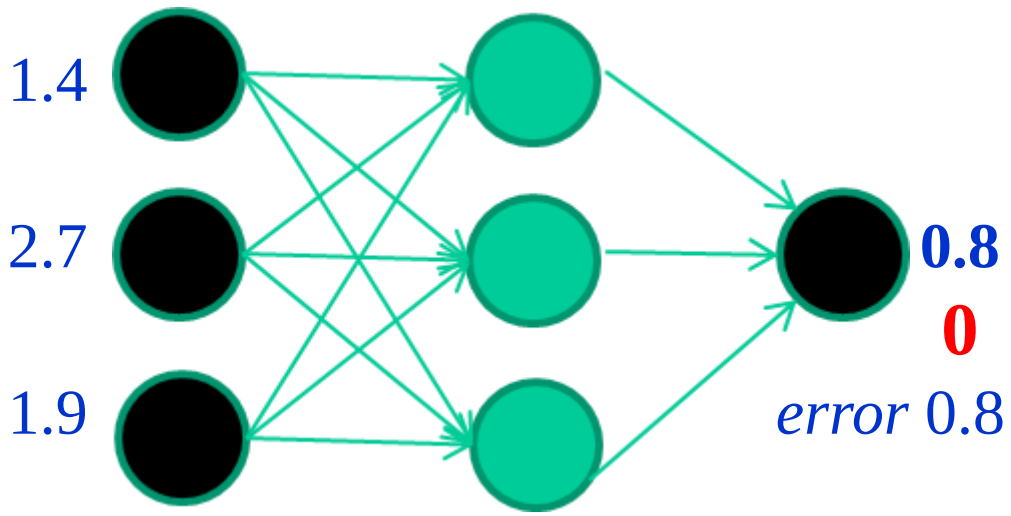


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

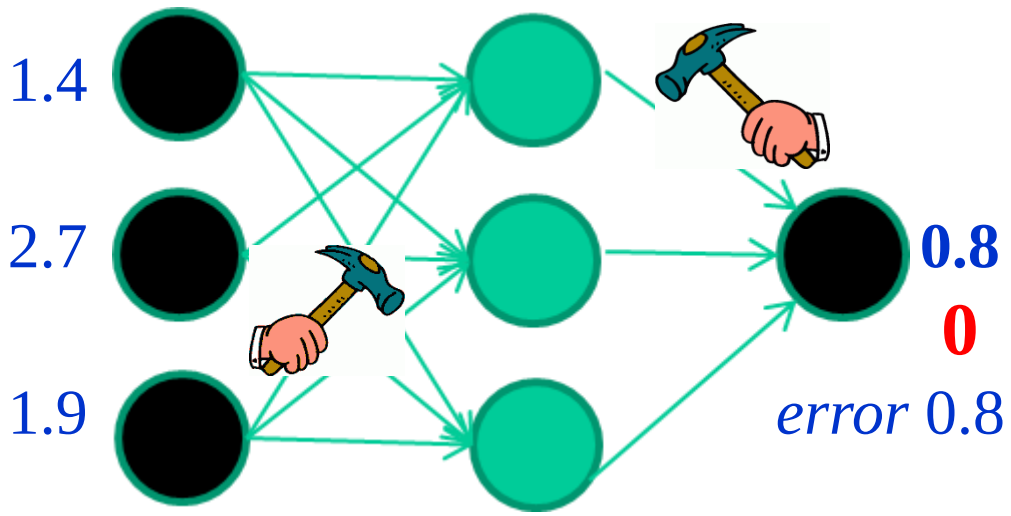
Compare with target output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

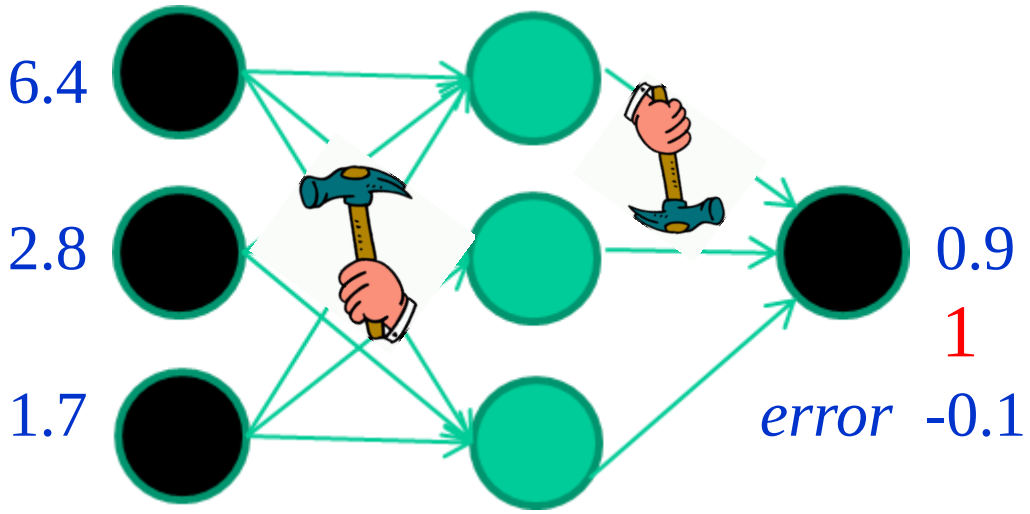
Adjust weights based on error



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

And so on ….



Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments *(stochastic gradient descent)*

Learning for neural networks

- Shallow networks
- Deep networks with multiple layers - deep learning

Feature detectors

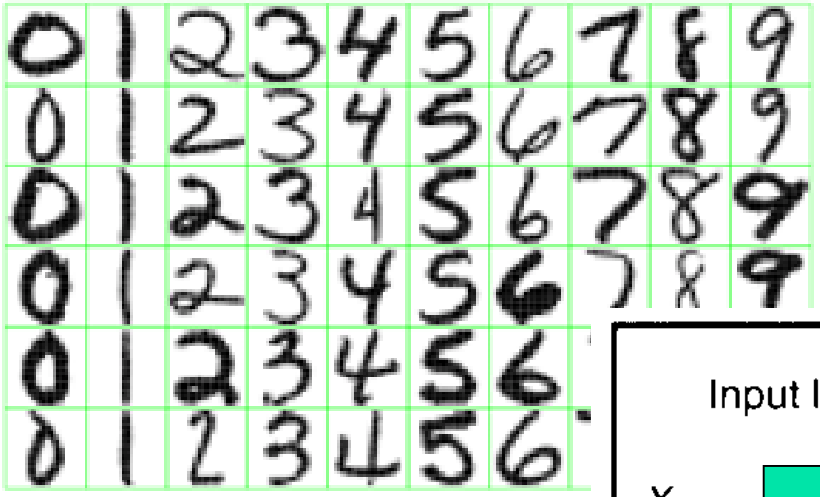
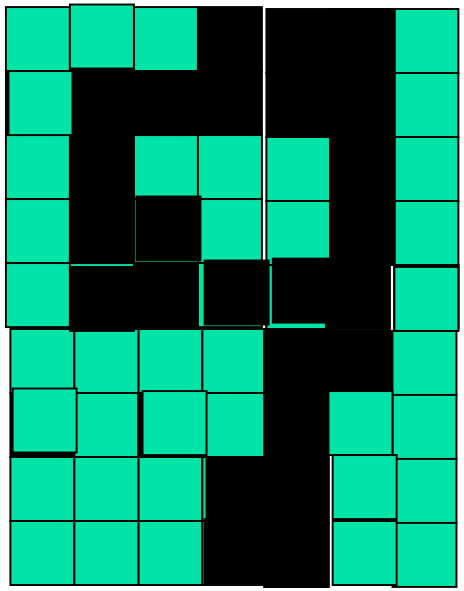
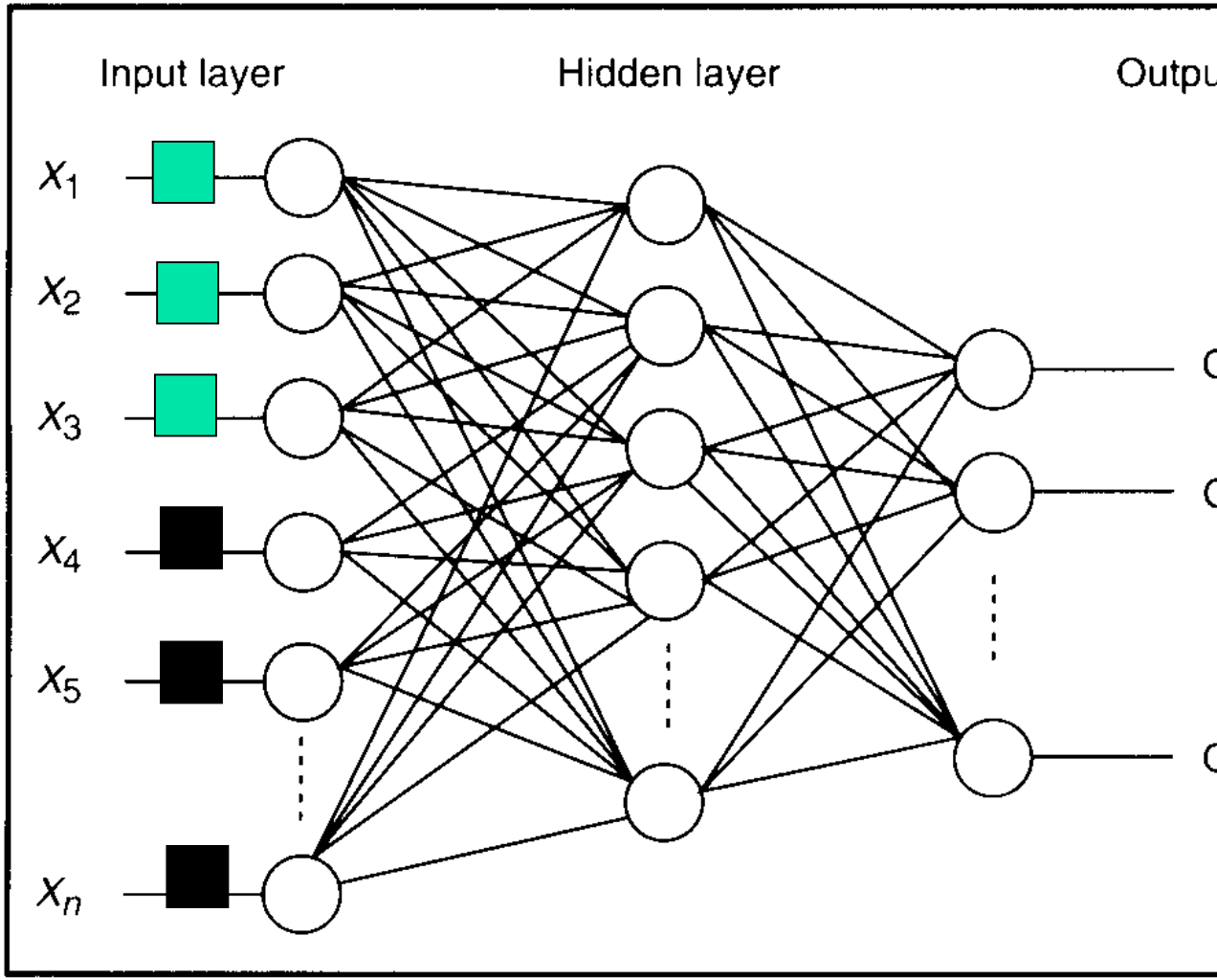
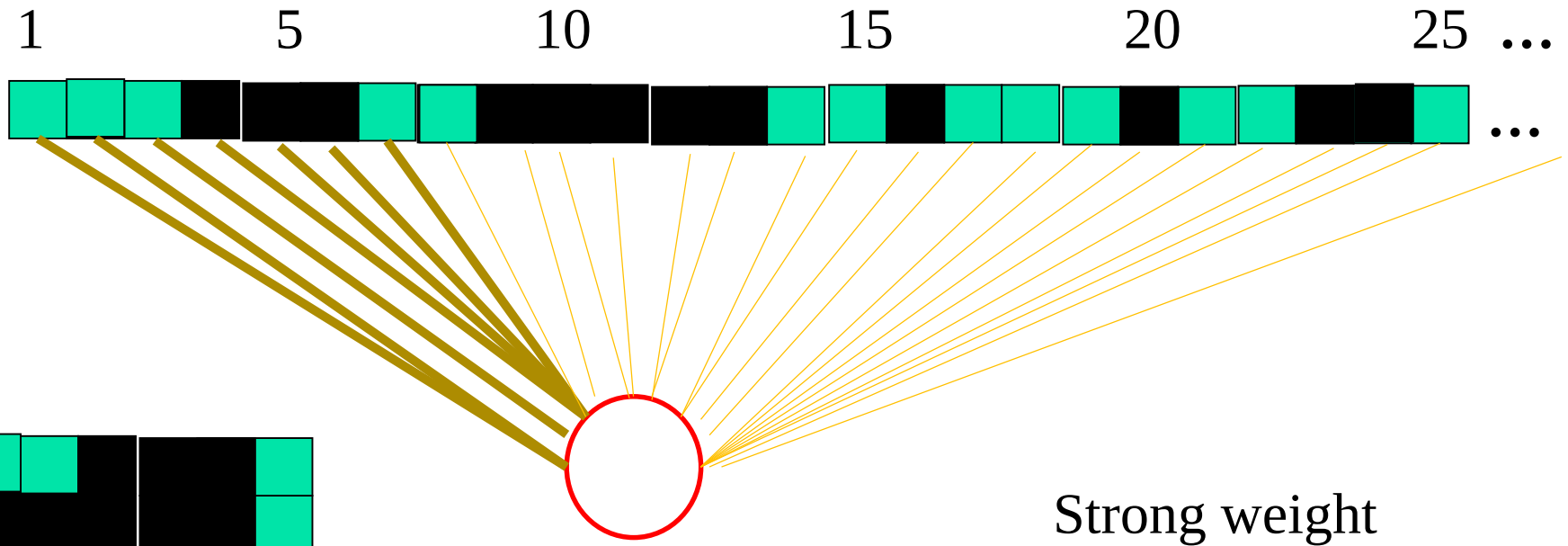


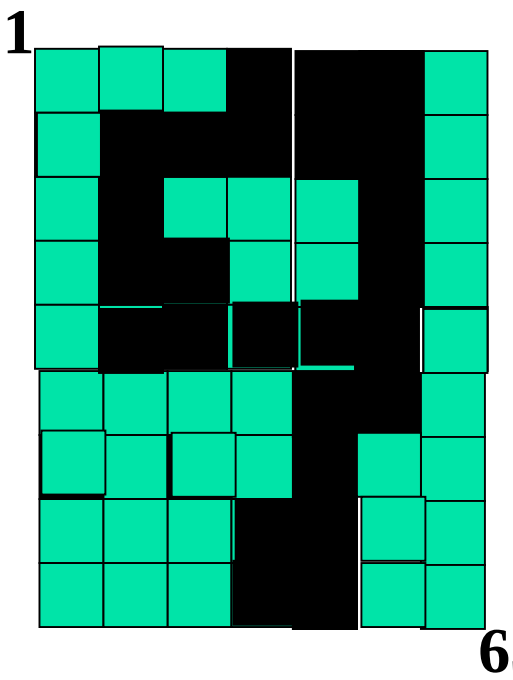
Figure 1.2: *Examples of handwritten d*
postal envelopes.





Hidden layer units become *self-organised feature detectors*

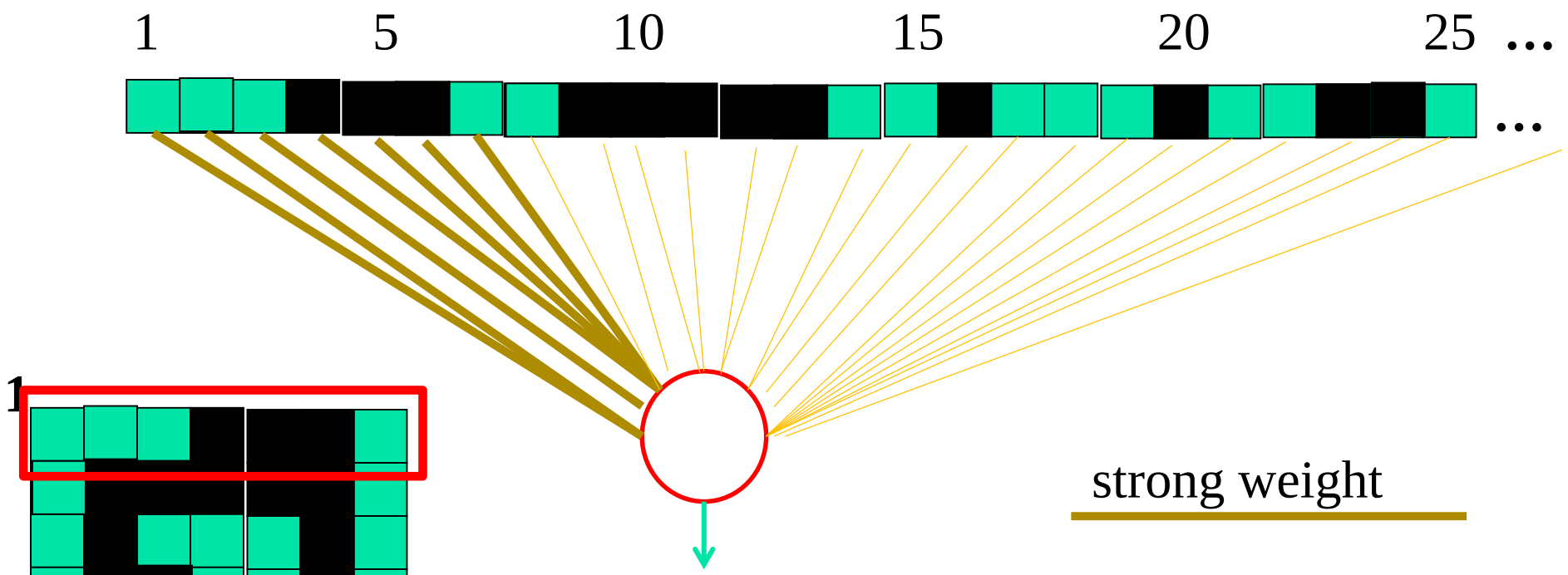




Strong weight

low/zero weight

What does this unit detect?



it will send strong signal for a horizontal line in the top row, ignoring everywhere else

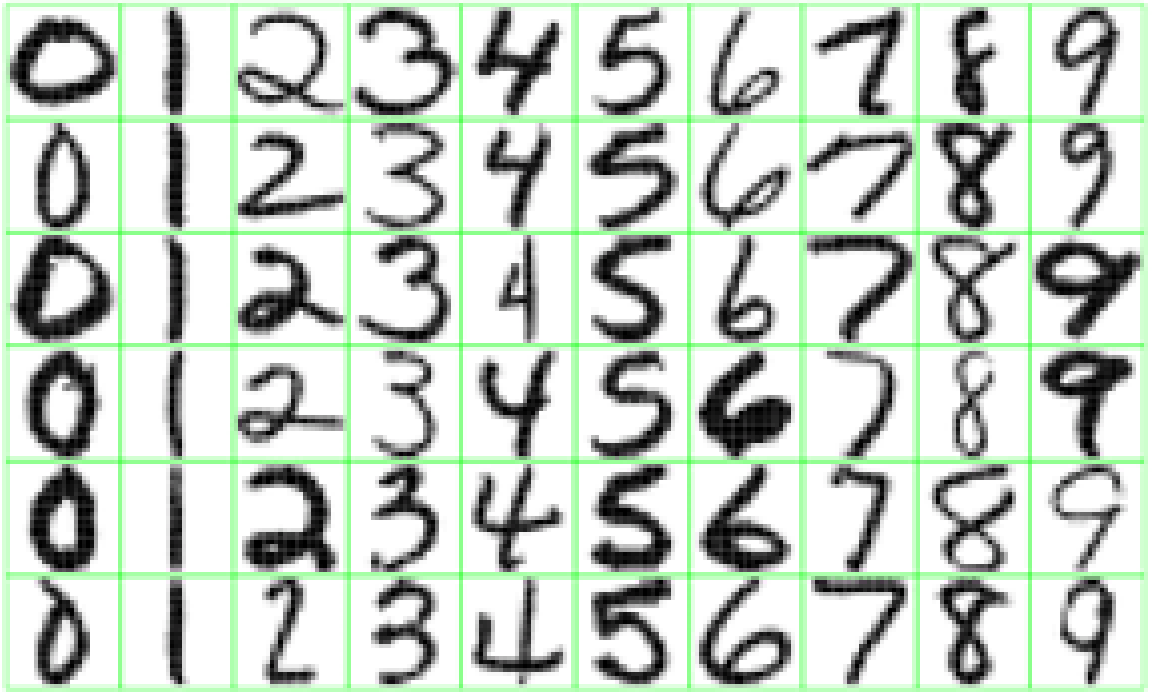


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features might you expect a good NN to learn, when trained with data like this?

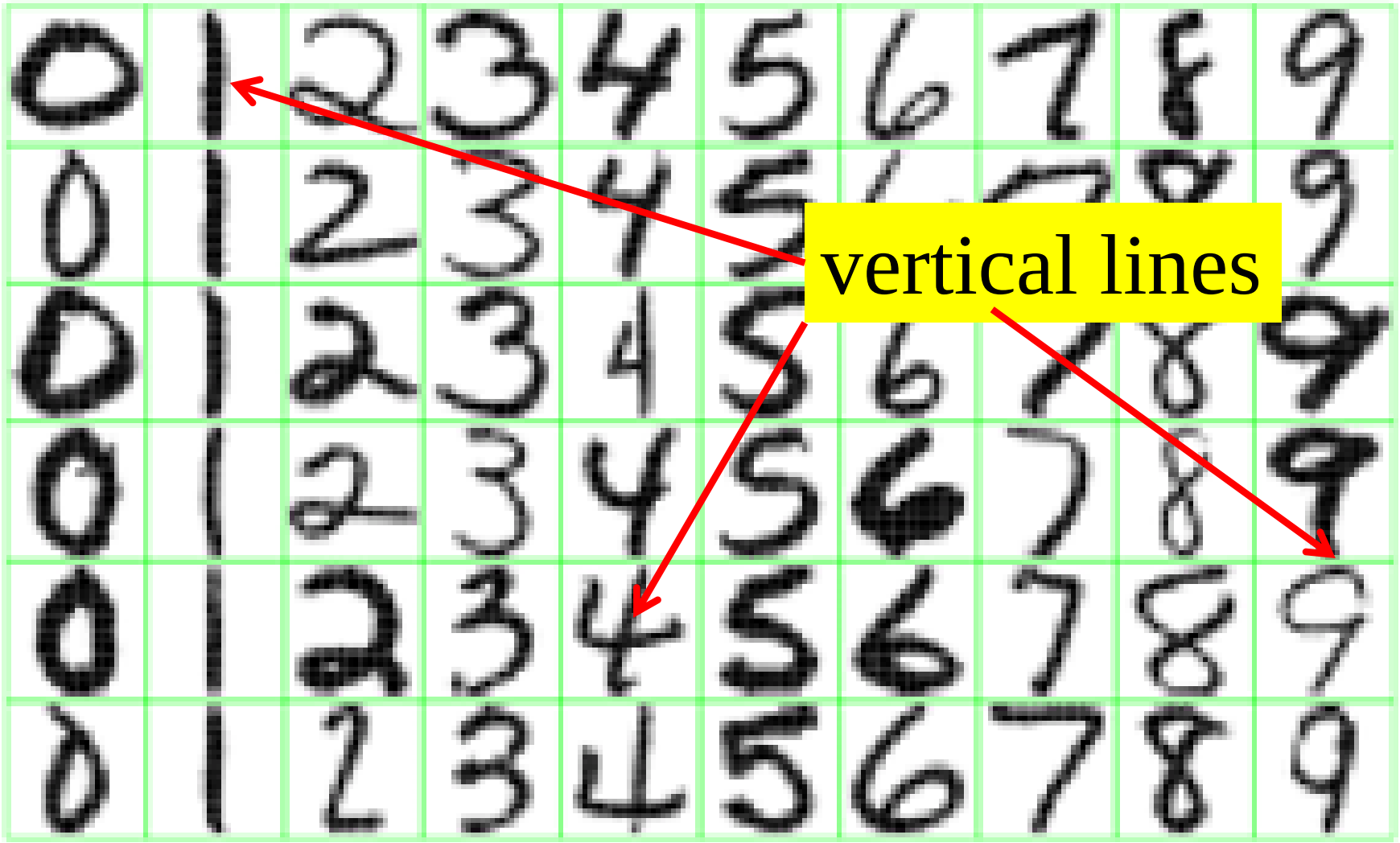
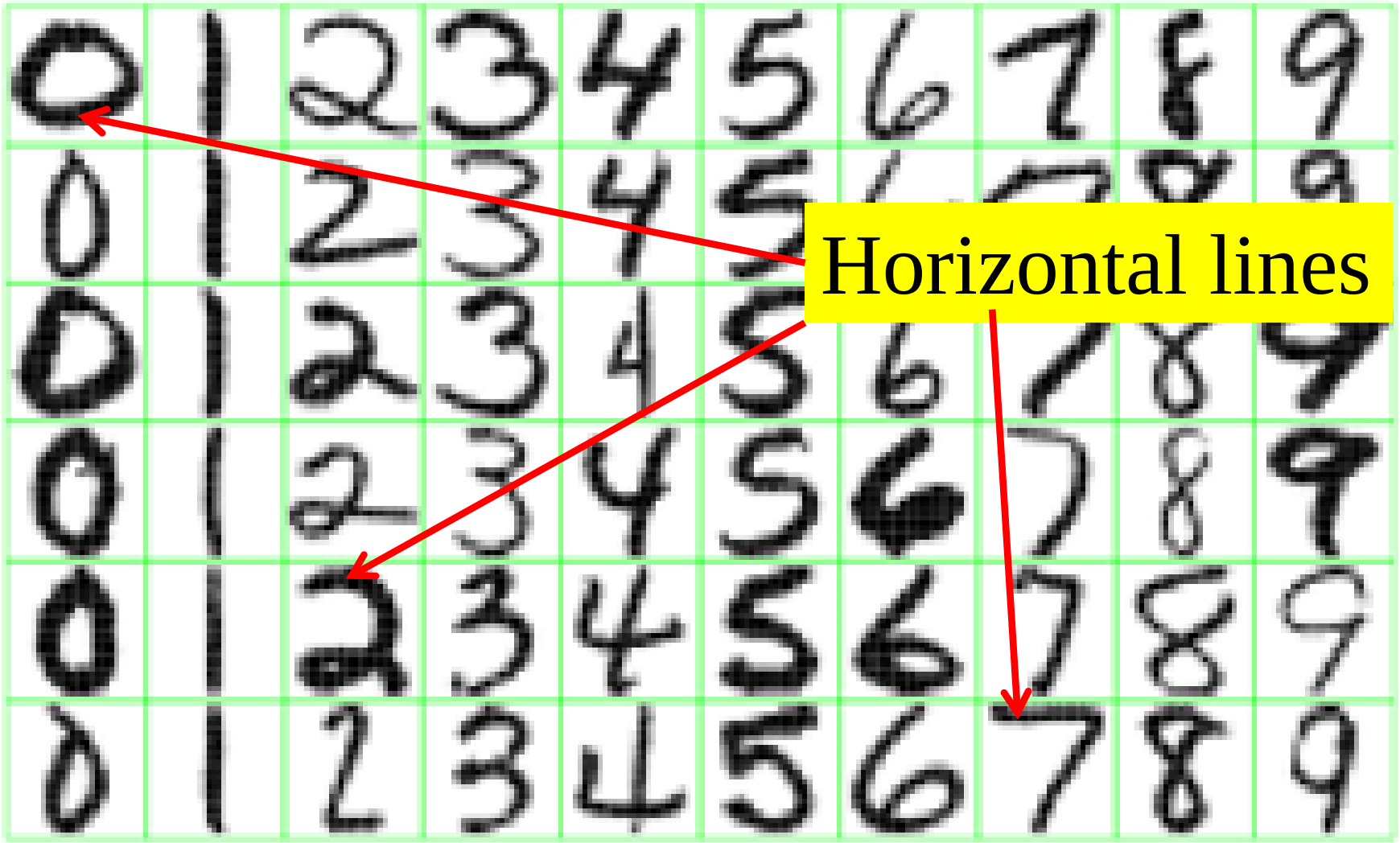


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

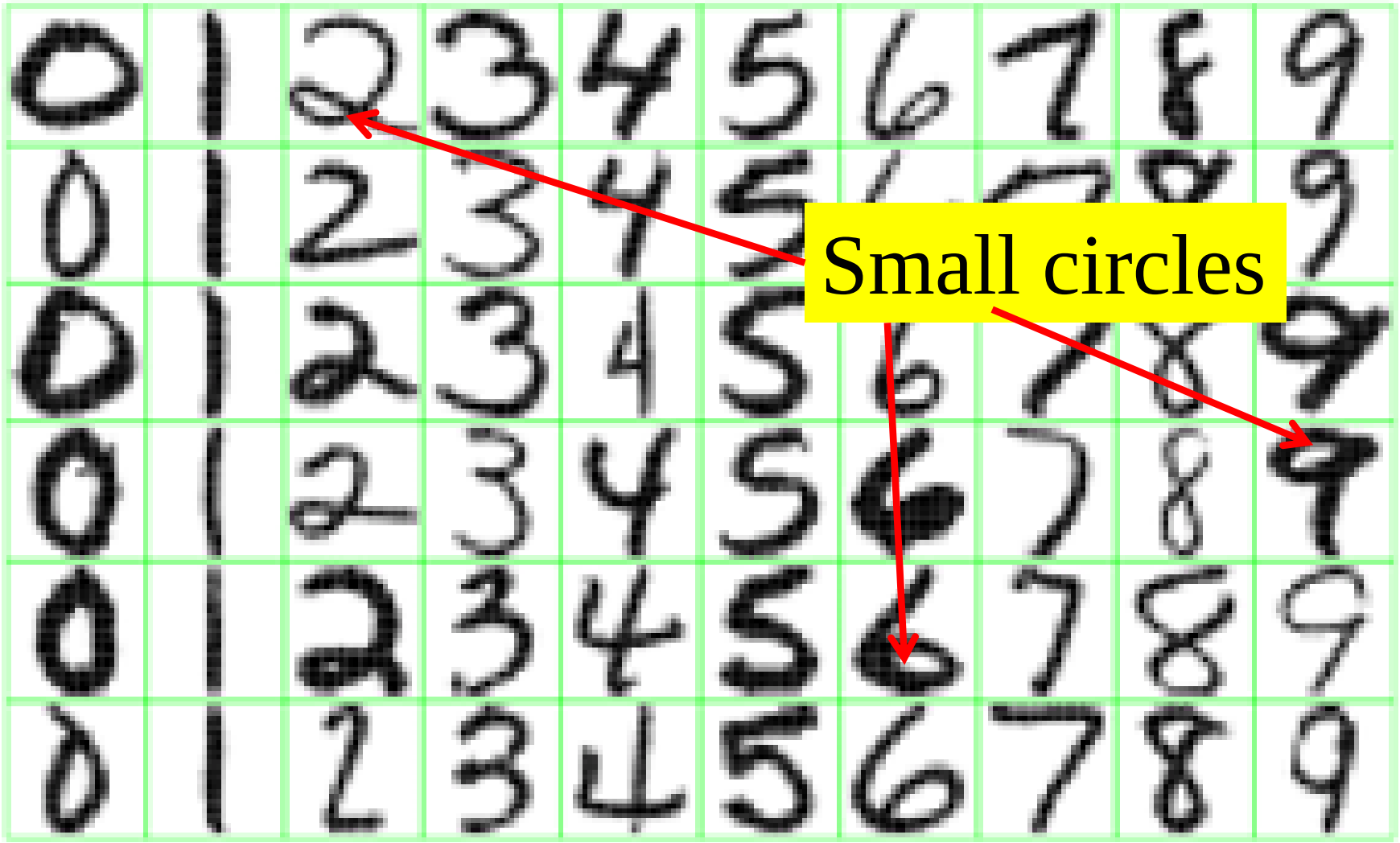


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

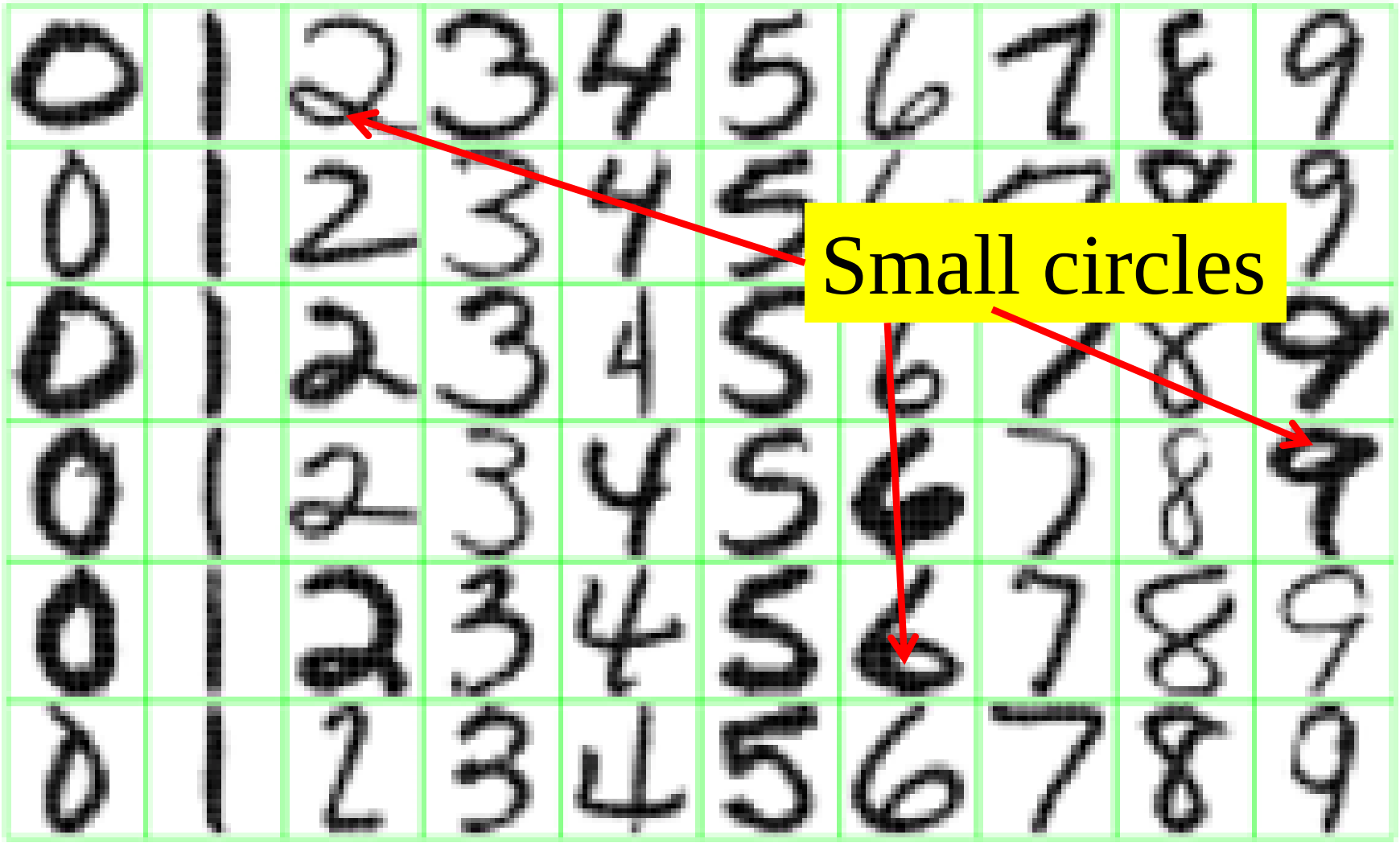
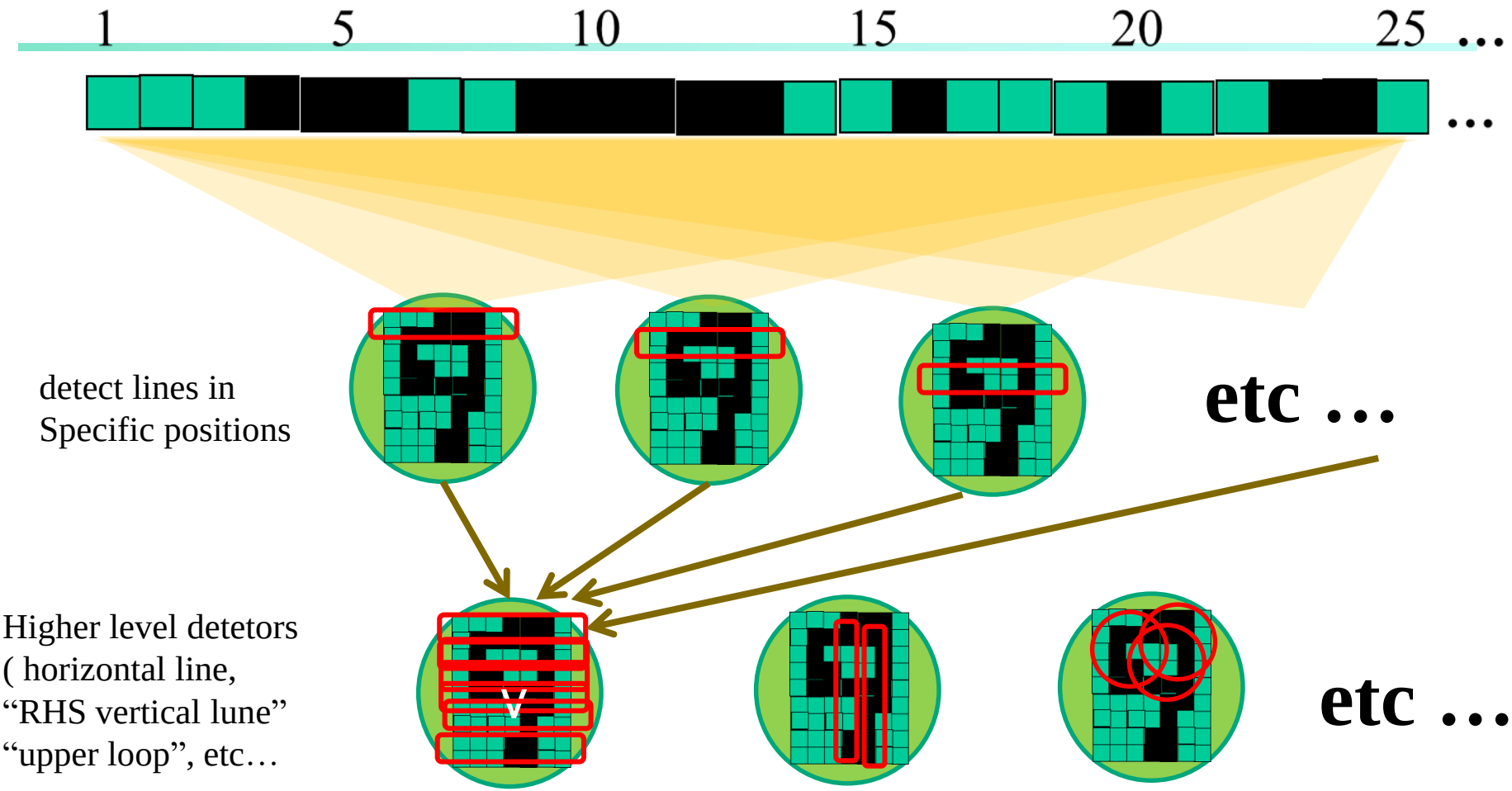


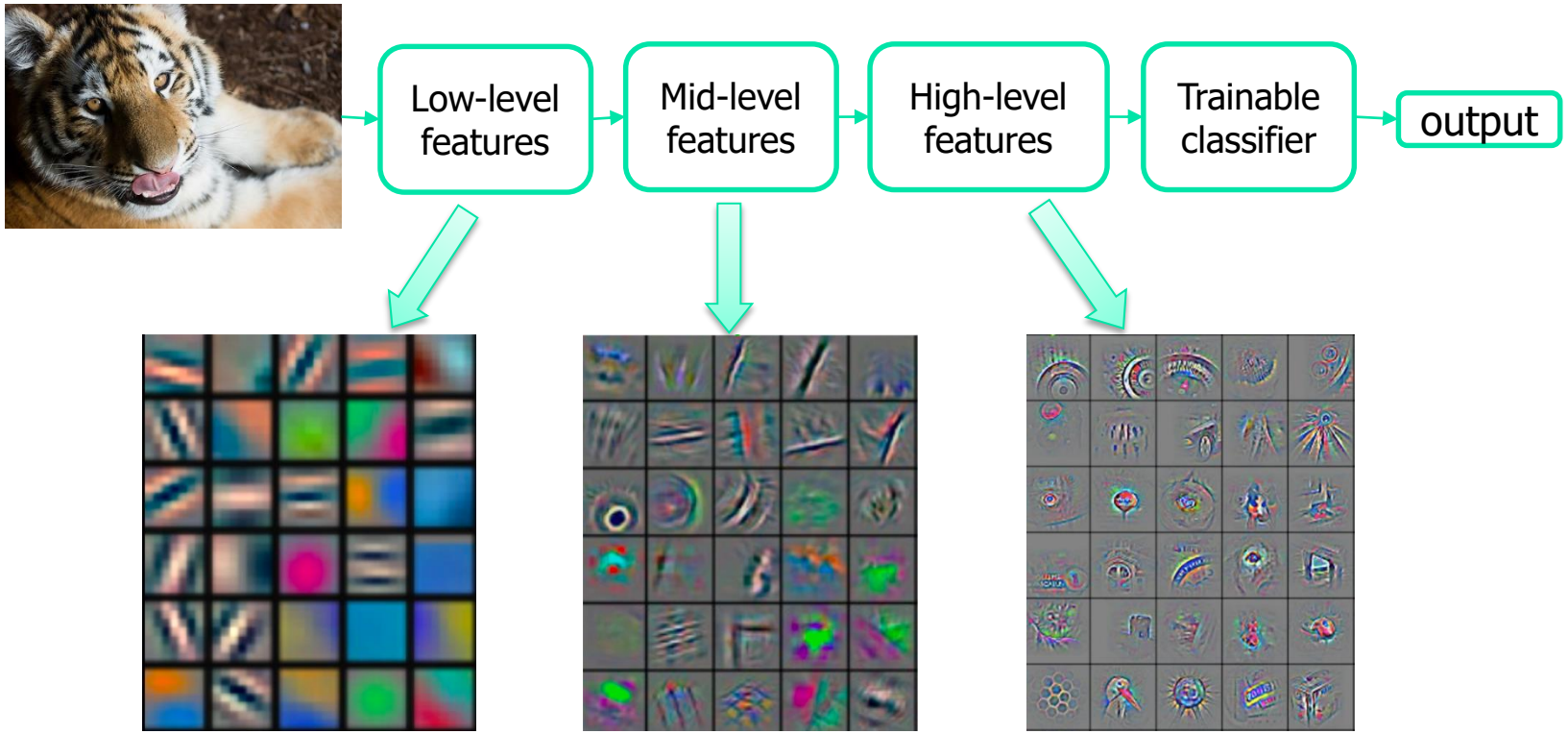
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

Hierarchical Feature Learning



Hierarchical Feature Learning

- Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.



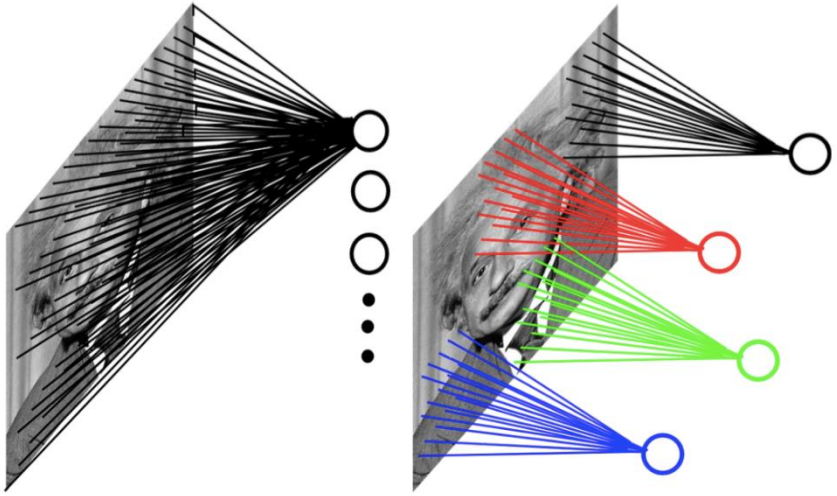
Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)

Deep Learning Architectures

- Commonly used architectures
  - convolutional neural networks
  - recurrent neural networks

Convolutional Neural Network

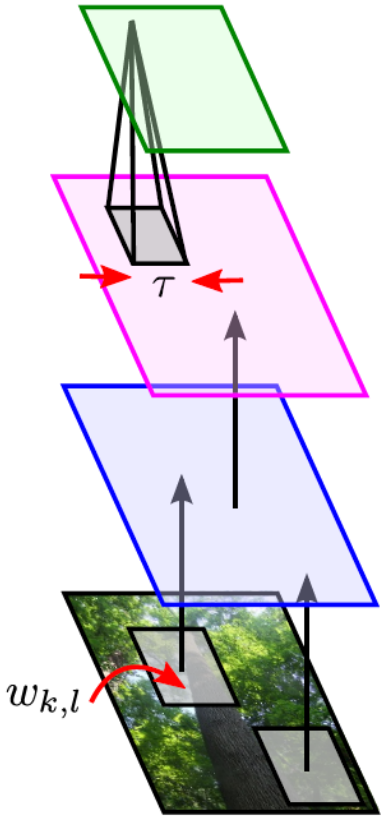
- Input can have very high dimensions
  - Using a fully-connected neural network would need a large amount of parameters.
- CNNs are a special type of neural network using shared weights and local connection
  - The number of parameters needed by CNNs is much smaller.



Example: 200x200 image

a) fully connected: 40,000 hidden units => 1.6 billion parameters
b) CNN: 5x5 filter, 100 filters => 2,500 parameters

Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

convolutional stage

Shared weights

ach sub-region yields a eature map, representing its feature.

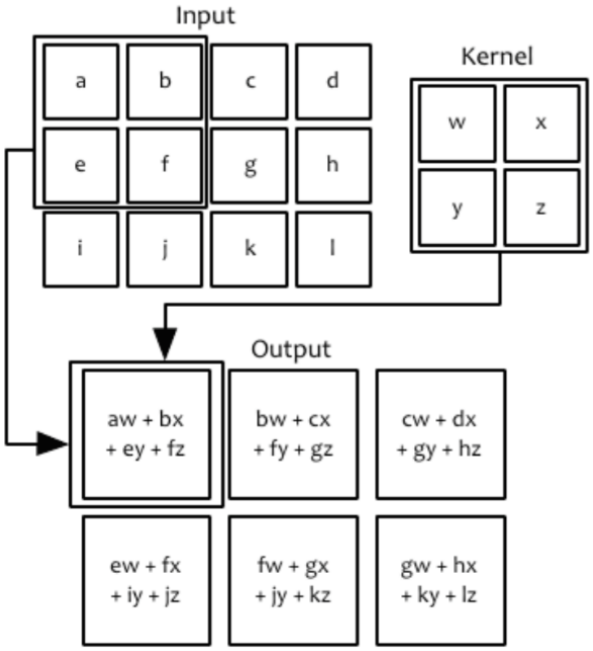
$z_{i,j}$

Images are segmented into sub-regions.

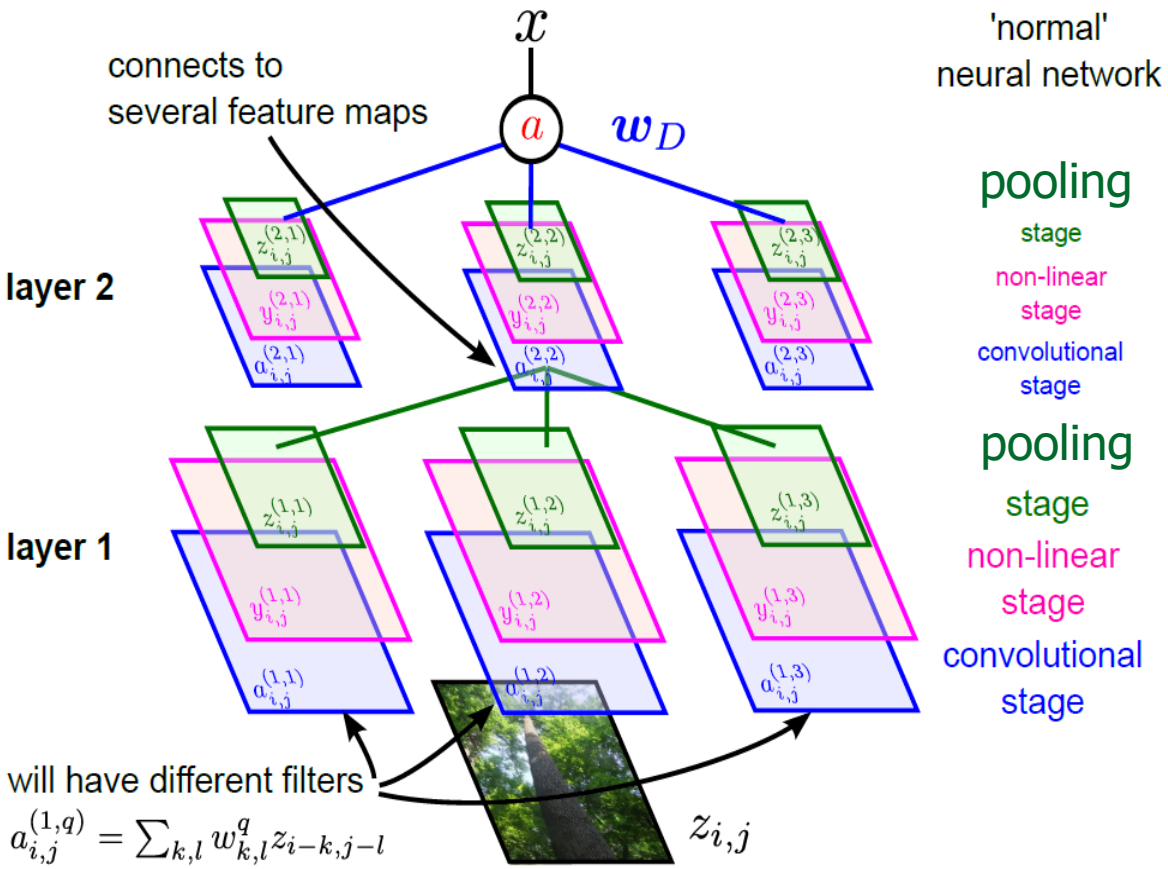
input image

CNN Architecture: Convolutional Layer

- The convolutional layer consists of a set of filters.
- Each filter covers a spatially small portion of the input data.
- Each filter is convolved across the dimensions of the input data (dot product), producing a multidimensional feature map.



Full CNN

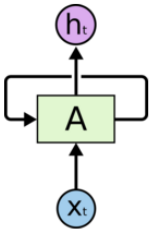

x
a
w_D
connects to
several feature maps
'normal'
neural network
pooling
stage
non-linear
stage
convolutional
stage
pooling
stage
non-linear
stage
convolutional
stage
layer 2
layer 1
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$

Recurrent Neural Networks

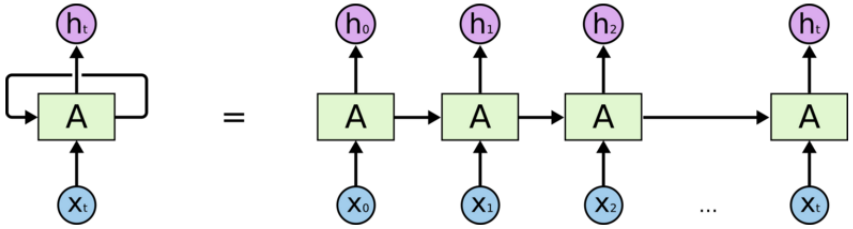
- Standard Neural Networks (also Convolutional Networks) :
  - Assume input examples as vectors of fixed length (e.g., an image) and produce a fixed-size vector as output (e.g., probabilities of different classes).
  - These models use a fixed amount of computational steps (e.g. the number of layers in the model).
- Recurrent Neural Networks
  - Model data with temporal or sequential structures
  - Varying length of input and outputs

Recurrent Neural Networks

- Recurrent Neural Networks are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.



An unrolled recurrent neural network.

At time t, given some input x_t and outputs a value h_t.

A loop allows information to be passed from one step of the network to the next.

Neural Network as a Classifier

- Weakness
  - Long training time
  - Require a large number of training data
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - Successful on an array of real-world data, e.g., hand-written letters
  - High tolerance to noisy data
  - Well-suited for continuous-valued inputs *and outputs*
  - Algorithms are inherently parallel

Deep learning frameworks

framework	base language	multi-GPU?	pros	cons
TensorFlow	Python and C++	yes		
Torch	Lua	yes	1.) easy to set up 2.) helpful error messages 3.) large amount of sample code and tutorials	can be somewhat difficult to set up in CentOS
Caffe	C++	yes		general
Theano	Python	By default, no. Can use more than one but requires a workaround	1.) expressive Python syntax 2.) higher-level spin-off frameworks 3.) large amount of sample code and tutorials	error messages are cryptic
IBM Deep Learning Platform	SystemML, et al.	expected yes	runs from out of box	
Neon	Python	yes	runs fast	

https://www.microway.com/hpc-tech-tips/deep-learning-frameworks-survey-tensorflow-torch-theano-caffe-neon-ibm-machine-learning-stack/

TensorFlow playground
http://playground.tensorflow.org/

Classification

- Basic concepts
- Decision tree
- Naïve Bayesian classifier
- Model evaluation
- Support Vector Machines
- Regression
- Neural Networks and Deep Learning
- Lazy Learners (k Nearest Neighbors)

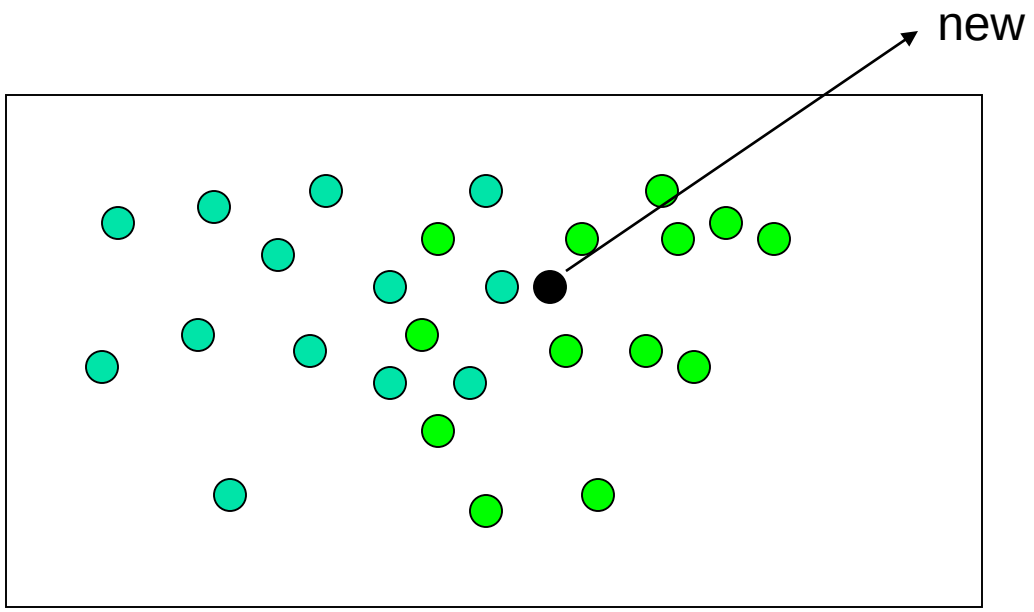
Lazy vs. Eager Learning

- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the previously discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions
  - Eager: must commit to a single hypothesis (global function) that covers the entire instance space

K-nearest neighbor method

- Majority vote within the k nearest neighbors

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$



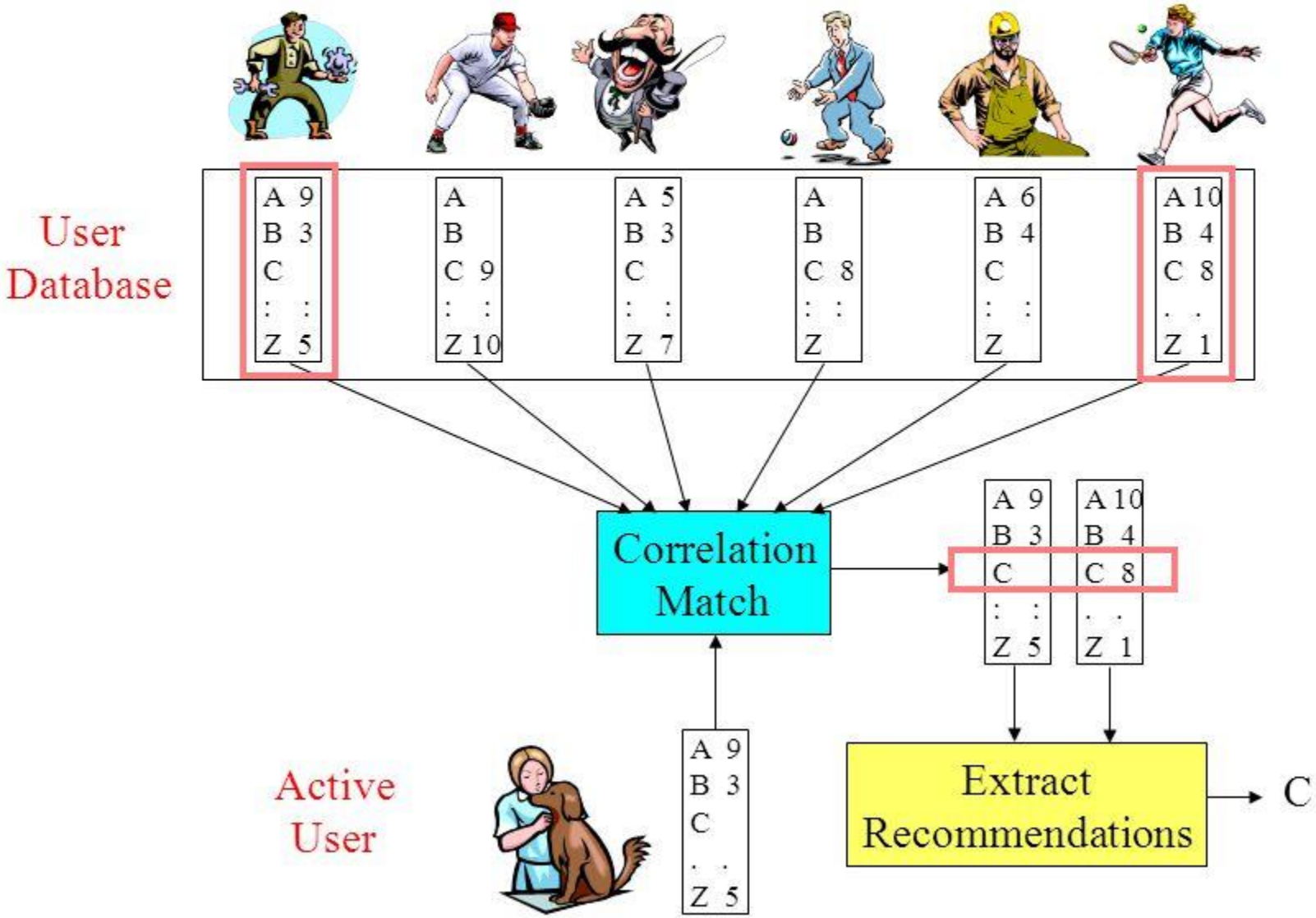
K= 1: positive
K= 3: negative

K-nearest neighbor method

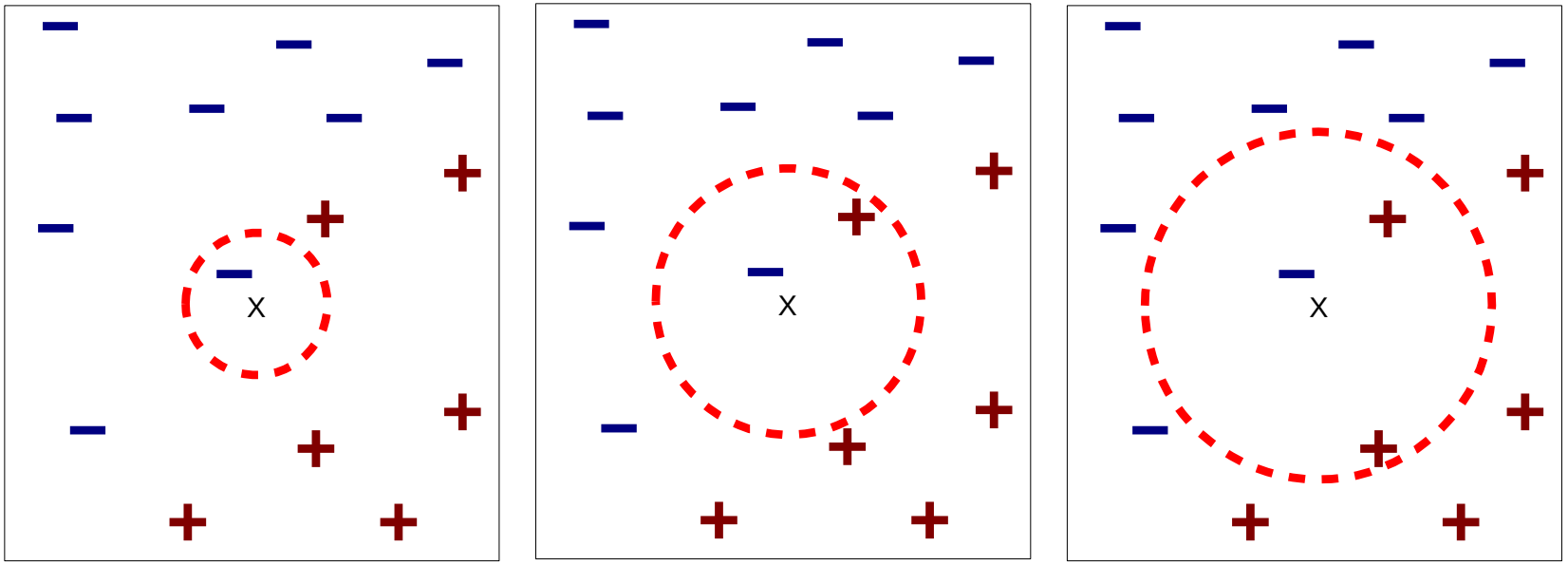
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query x_q

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

Collaborative Filtering



Definition of Nearest Neighbor

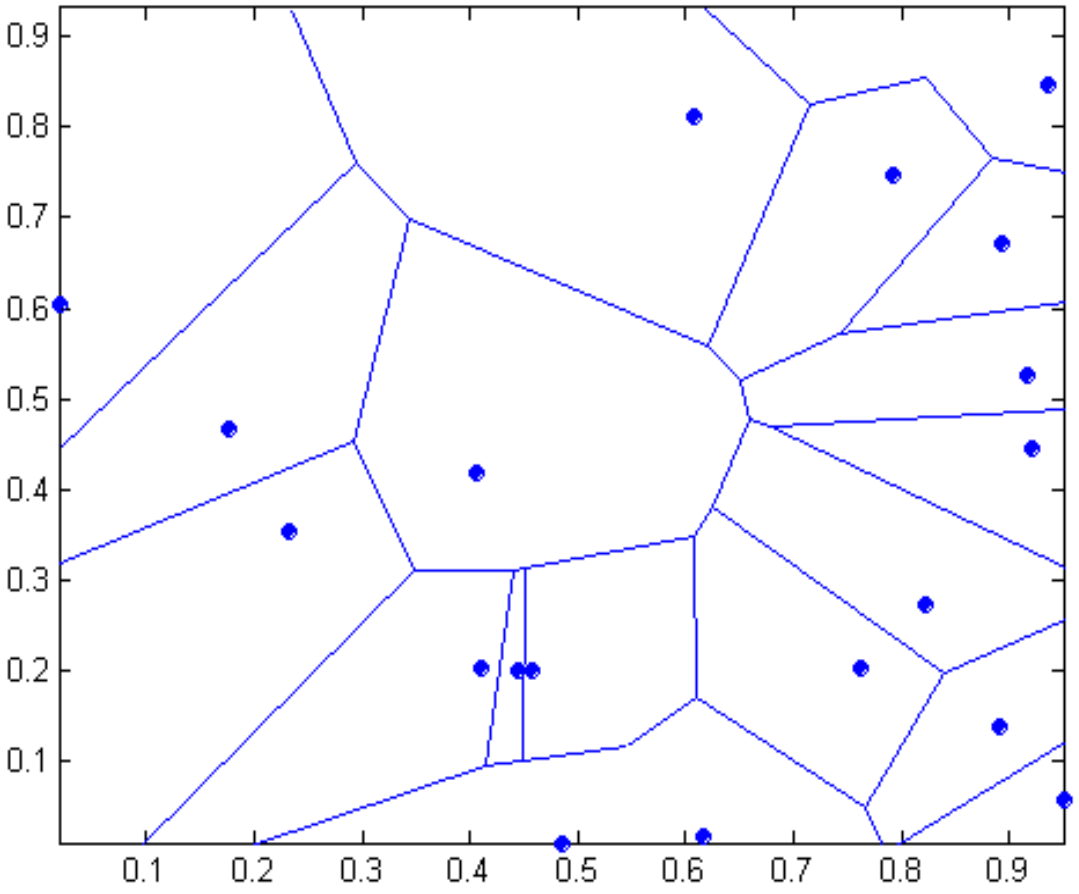


(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

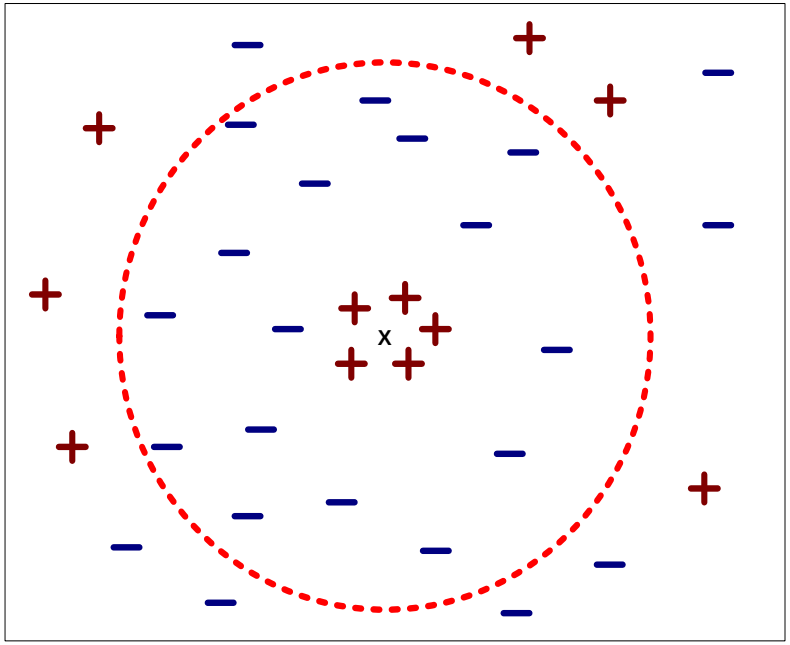
1 nearest-neighbor

Voronoi Diagram



Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



Similarity/distance between data objects

Data objects

- as points: distance between points
- as vectors: cosine between vectors
- as random variables: correlation
- as sets: Jaccard distance between sets
- as strings: Hamming distance

Distance between two data points

- Euclidean distance

$$d(i,j)=\sqrt{(|x_{i_1}-x_{j_1}|^2+|x_{i_2}-x_{j_2}|^2+...+|x_{i_p}-x_{j_p}|^2)}$$

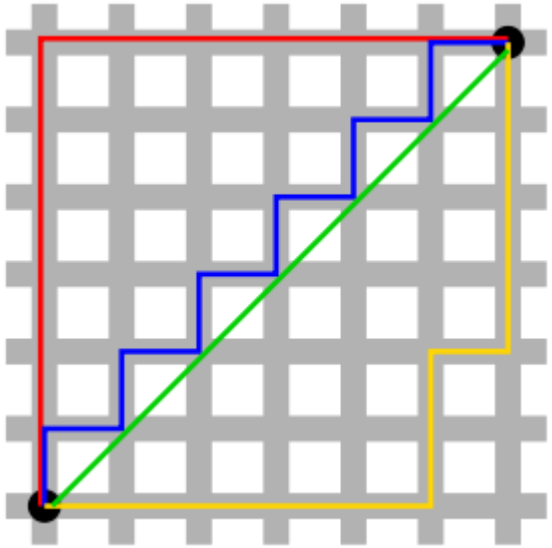
- Manhattan distance

$$d(i,j)=|x_{i_1}-x_{j_1}|+|x_{i_2}-x_{j_2}|+...+|x_{i_p}^{1}-x_{j_p}|$$

- *Minkowski distance*

$$d(i,j)=\sqrt[q]{(|x_{i_1}-x_{j_1}|^q+|x_{i_2}-x_{j_2}|^q+...+|x_{i_p}-x_{j_p}|^q)}$$

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$



Distance between two attributes values

- To compute $|x_{if} - x_{jf}|$
  - *f* is numeric (interval or ratio scale)
    - Scaling issues -> normalization
  - *f* is ordinal
    - Mapping by rank $z_{if} = \frac{r_{if} - 1}{M_f - 1}$
  - *f* is nominal
    - Mapping function

      $|x_{if} - x_{jf}| = 0$ if $x_{if} = x_{jf}$, or 1 otherwise
    - Hamming distance (edit distance) for strings

Normalization of attributes

- scaled attributes to fall within a small, specified range
- Min-max normalization: [min_A, max_A] to [new_min_A, new_max_A]

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex. Let income [\$12,000, \$98,000] normalized to [0.0, 1.0]. Then \$73,000 is mapped to $\frac{73{,}600 - 12{,}000}{98{,}000 - 12{,}000}(1.0 - 0) + 0 = 0.716$
- Z-score normalization (μ: mean, σ: standard deviation):

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - Ex. Let μ = 54,000, σ = 16,000. Then $\frac{73{,}600 - 54{,}000}{16{,}000} = 1.225$

Weighted distance

- Assigning weights to different attributes

$$d(A,B) = \sqrt{\sum_i w_i (A_i - B_i)^2},$$

- If wi is inverse variance, it's a form of Mahalanobis distance
- Supervised metric learning: learn the weights wi using labeled data

Euclidean distance

- Euclidean distance may not be meaningful (counter intuitive) for high dimensional data, e.g. user movie ratings

3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

vs

0	3	0	3	0	3	0	3	0	3	0	3	0	3	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Similarity/distance between data objects

Data objects

- as points: distance between points
- **as vectors: cosine between vectors**
- **as random variables: correlation**
- as sets: Jaccard distance between sets
- as strings: Hamming distance

Cosine similarity between two vectors

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

- Cosine measure $\frac{X_i \bullet X_j}{\|X_i\| \cdot \|X_j\|}$
- From -1 to 1

Cosine similarity

- Consine similarity
  - Invariant to multiplicative scaling
  - Variant to additive scaling

3	3	3	3	3	3	6	6	6	6	6	6		
1	1	1	1	1	1	4	4	4	4	4	4		

vs

2	2	2	2	2	2	8	8	8	8	8	8		
1	1	1	1	1	1	4	4	4	4	4	4		

Correlation between two random variables (numerical data)

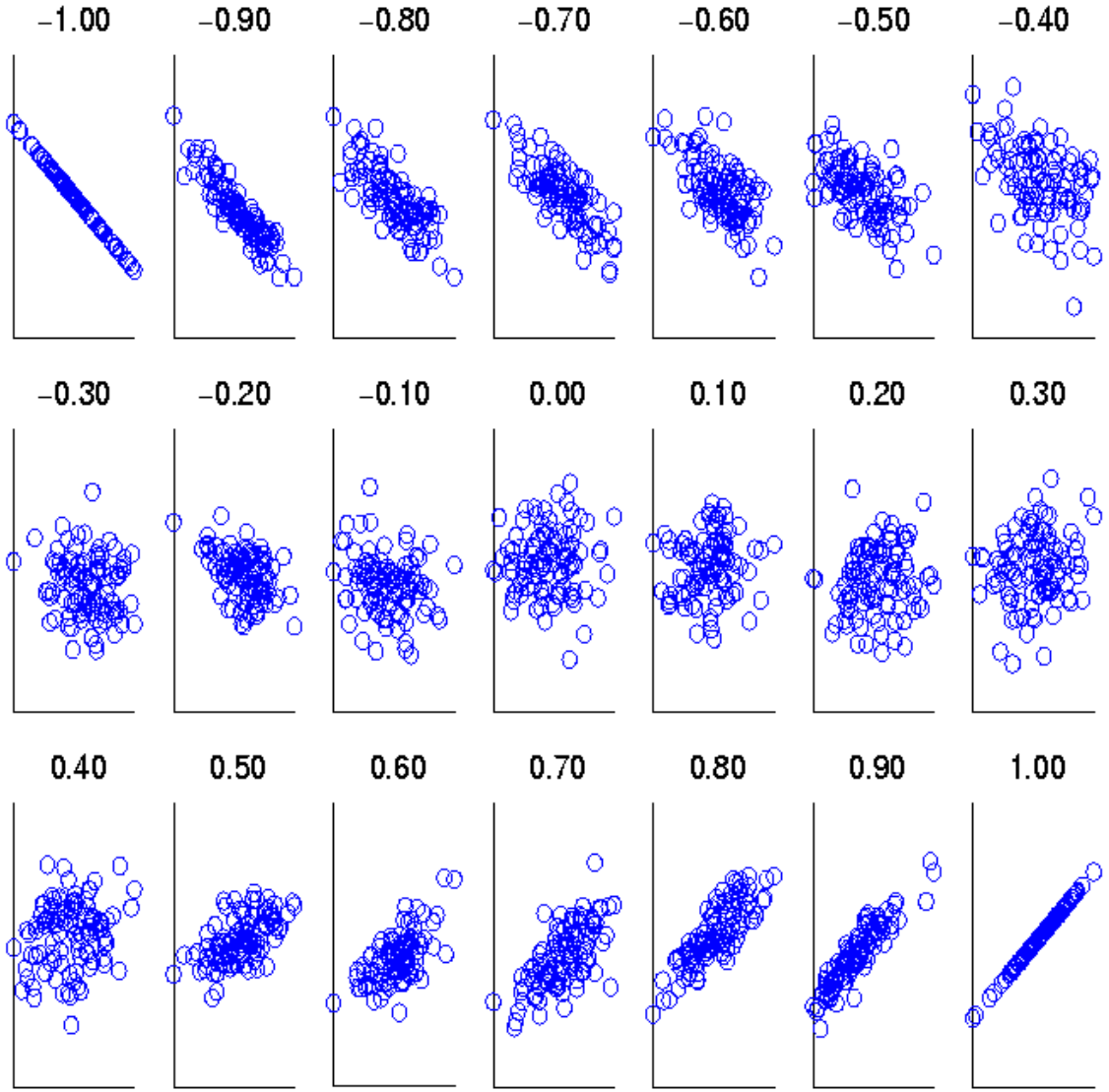
- Correlation coefficient (also called Pearson's product moment coefficient)

$$r_{A,B} = \frac{\sum (A - \overline{A})(B - \overline{B})}{(n-1)\sigma_A \sigma_B} = \frac{\sum (AB) - n\overline{A}\,\overline{B}}{(n-1)\sigma_A \sigma_B}$$

where n is the number of tuples, $\overline{A}$ and $\overline{B}$ are the respective means of A and B, σ_A and σ_B are the respective standard deviation of A and B, and $\Sigma(AB)$ is the sum of the AB dot-product.

- $r_{A,B} > 0$, A and B are positively correlated (A's values increase as B's)
- $r_{A,B} = 0$: independent
- $r_{A,B} < 0$: negatively correlated

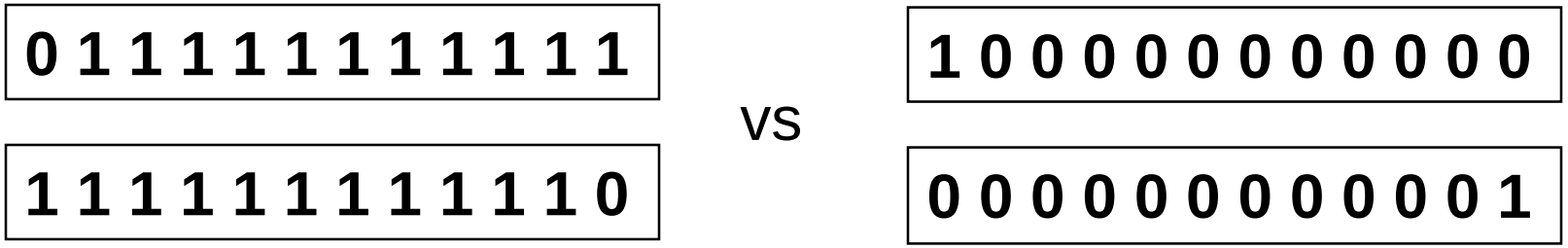
Visualization of Correlation



Scatter plots showing the Pearson correlation from –1 to 1.

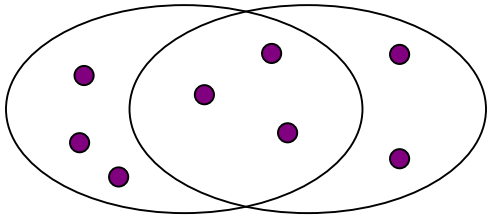
Data object at a set

- For transaction data, document data
  - Shared items are more important to consider



Jaccard distance between two sets

- The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union: $sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
- **Jaccard distance:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

Similarity/distance between data objects

Data objects

- as points: distance between points
- as vectors: cosine between vectors
- as random variables: correlation
- as sets: Jaccard distance between sets
- as strings: Hamming distance

Supervised Learning

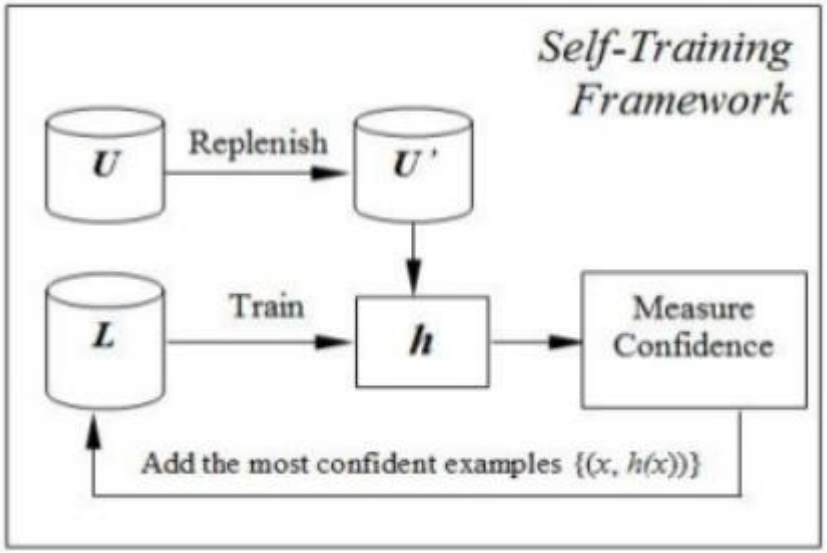
- Decision tree
- Naïve Bayesian classifier
- Support Vector Machines
- Regression
- Neural Networks and Deep Learning
- Lazy Learners (k Nearest Neighbors)

Semi-Supervised Classification

- Supervised learning: learning from labeled data
- Labeled data can be rare or expensive, unlabeled data are much easier to get
- Semi-supervised learning: Use both labeled and unlabeled data
  - Self-training
  - Co-training
- Active learning: iterative supervised learning

Self-training

- Build a classifier using the labeled data
- Repeatedly Use it to label the unlabeled data, and those with the most confident label prediction are added to the set of labeled data
- May reinforce errors

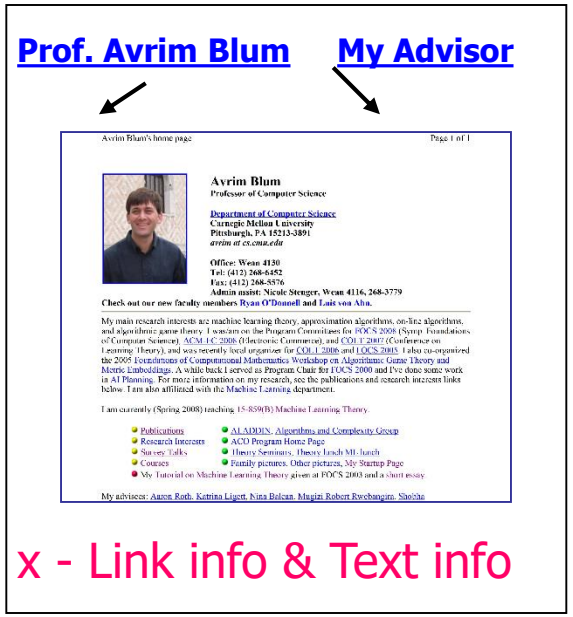


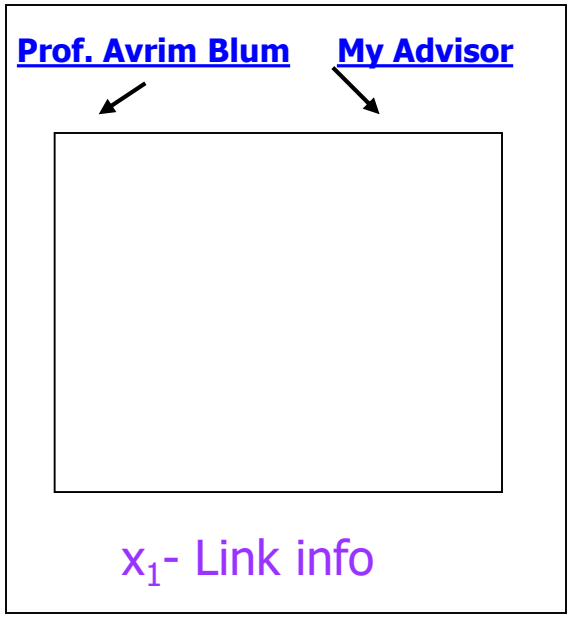
Co-training

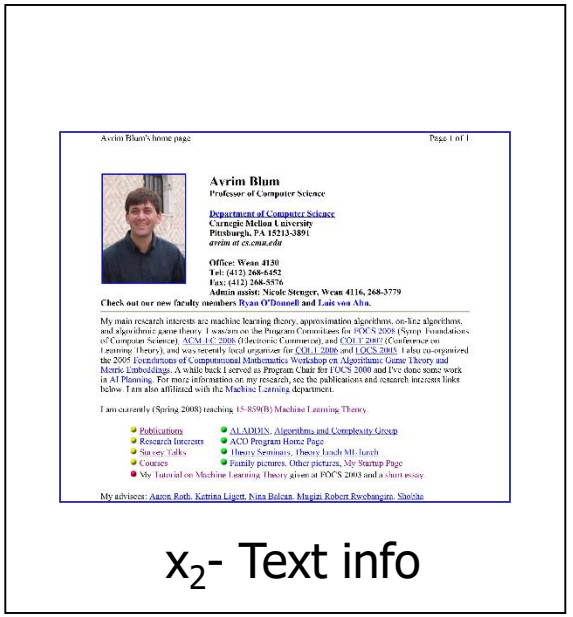
[Blum&Mitchell'98]

Many problems have two different sources of info you can use to determine label.

E.g., classifying webpages: can use words on page or words on links pointing to the page.

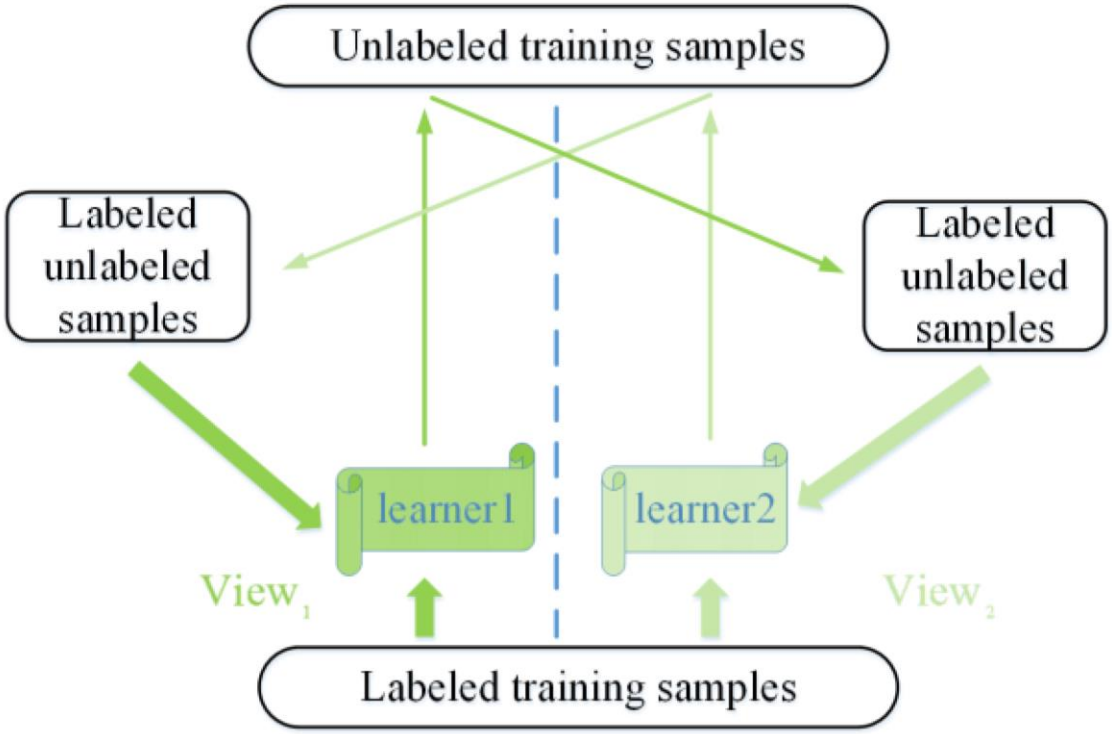






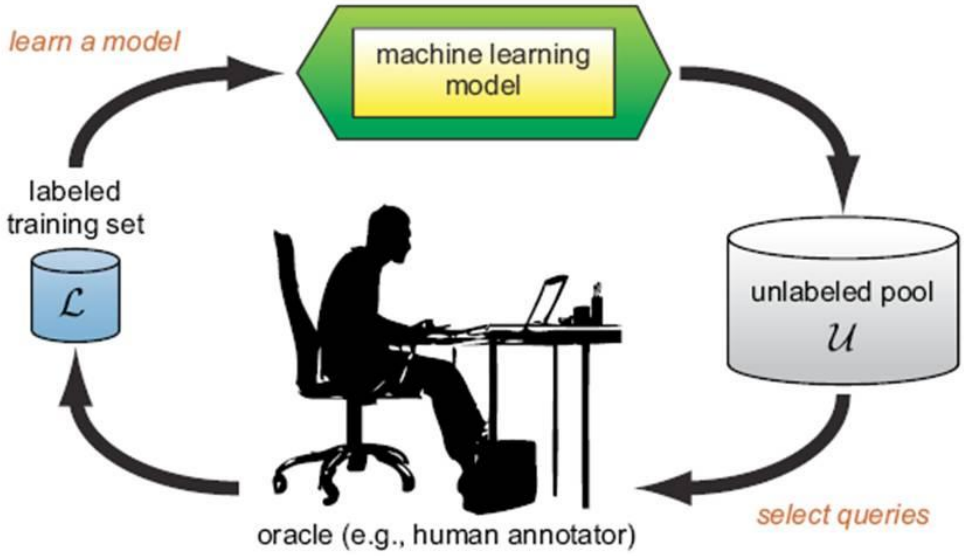
Co-training

- Learn a separate classifier for each view using labeled data
- Iteratively use each classifier on the unlabeled data to construct additional labeled training data for the other classifier.



Active Learning

- Use a query function to select one or more tuples from Unlabeled data and request labels from an oracle (a human annotator)
- The newly labeled samples are added to labeled data to train the model
- Evaluated through *learning curves*: Accuracy as a function of the number of instances queried



Summary

- Supervised learning: classification and regression
  - Decision tree
  - Naïve Bayesian classifier
  - Support Vector Machines
  - Linear regression and logistic regression
  - Neural Networks and Deep Learning
  - Lazy Learners (k Nearest Neighbors)
- Ensemble methods
- Model evaluation and selection
- Semi-supervised learning