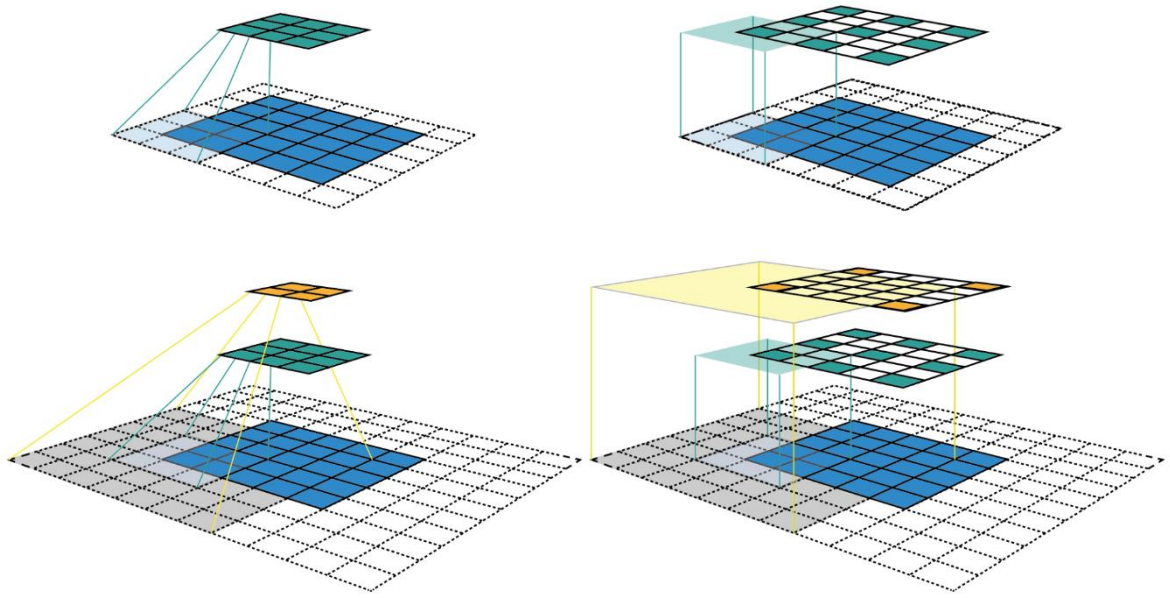
University of Ljubljana, Faculty of Computer and Information Science

Kernel methods and neural networks



Prof Dr Marko Robnik-Šikonja

Intelligent Systems, November 2018

Topics overview

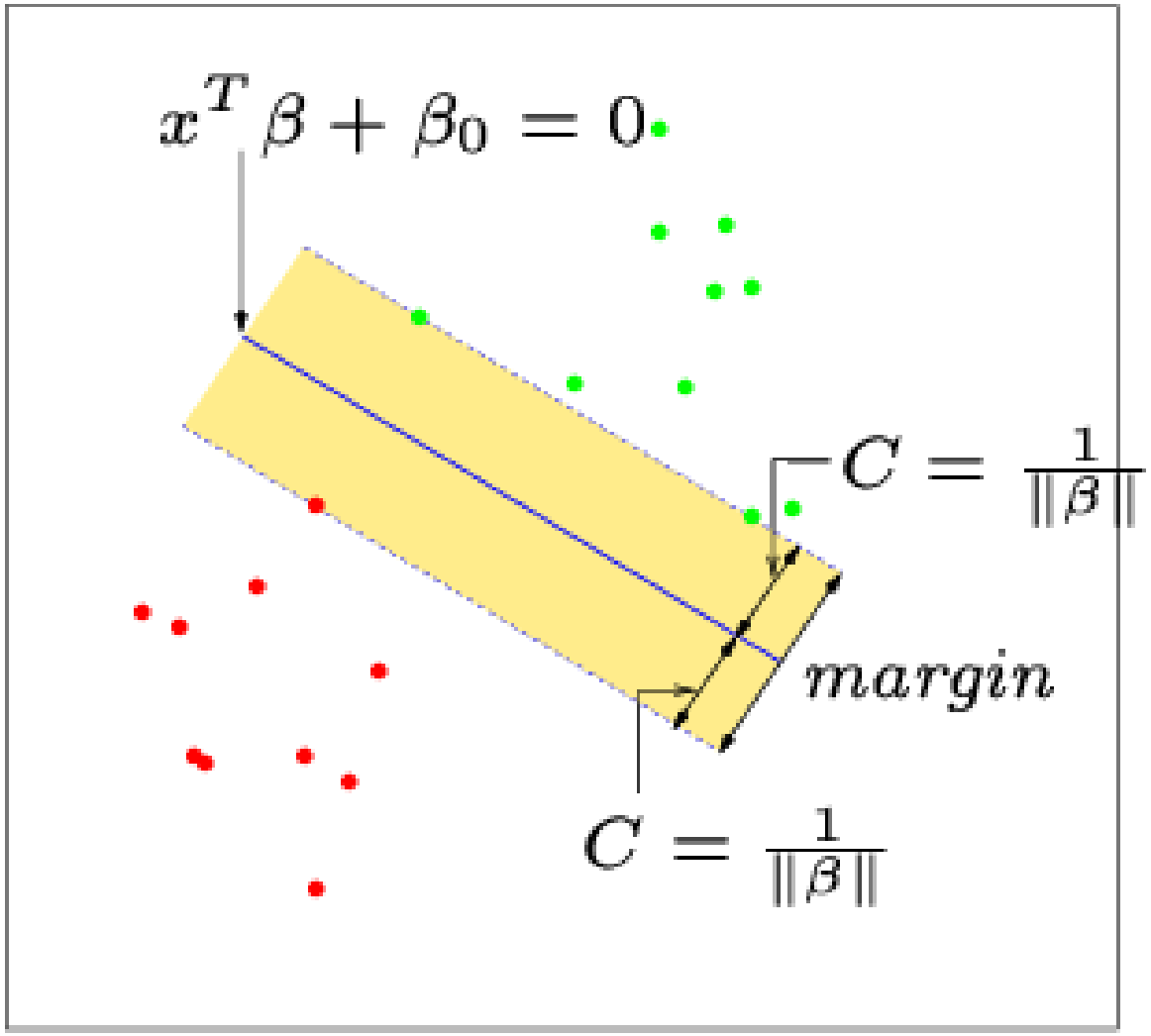
- support vector machines
- neural networks
- imbalanced learning

Support vector machines

- Imagine a situation where you have a two class classification problem with two predictors X_1 and X_2.
- Suppose that the two classes are “linearly separable” i.e. one can draw a straight line in which all points on one side belong to the first class and points on the other side to the second class.
- Then a natural approach is to find the straight line that gives the biggest separation between the classes i.e. the points are as far from the line as possible
- This is the basic idea of a support vector classifier.

An illustration

- C is the minimum perpendicular distance between each point and the separating line.
- We find the line which maximizes C.
- This line is called the “optimal separating hyperplane”
- The classification of a point depends on which side of the line it falls on.



More than two dimensions

- This idea works just as well with more than two predictor variables.
- For example, with three predictors you want to find the plane that produces the largest separation between the classes.
- With more than three dimensions it becomes hard to visualize a plane but it still exists. In general they are caller hyper-planes.

Non-separating classes

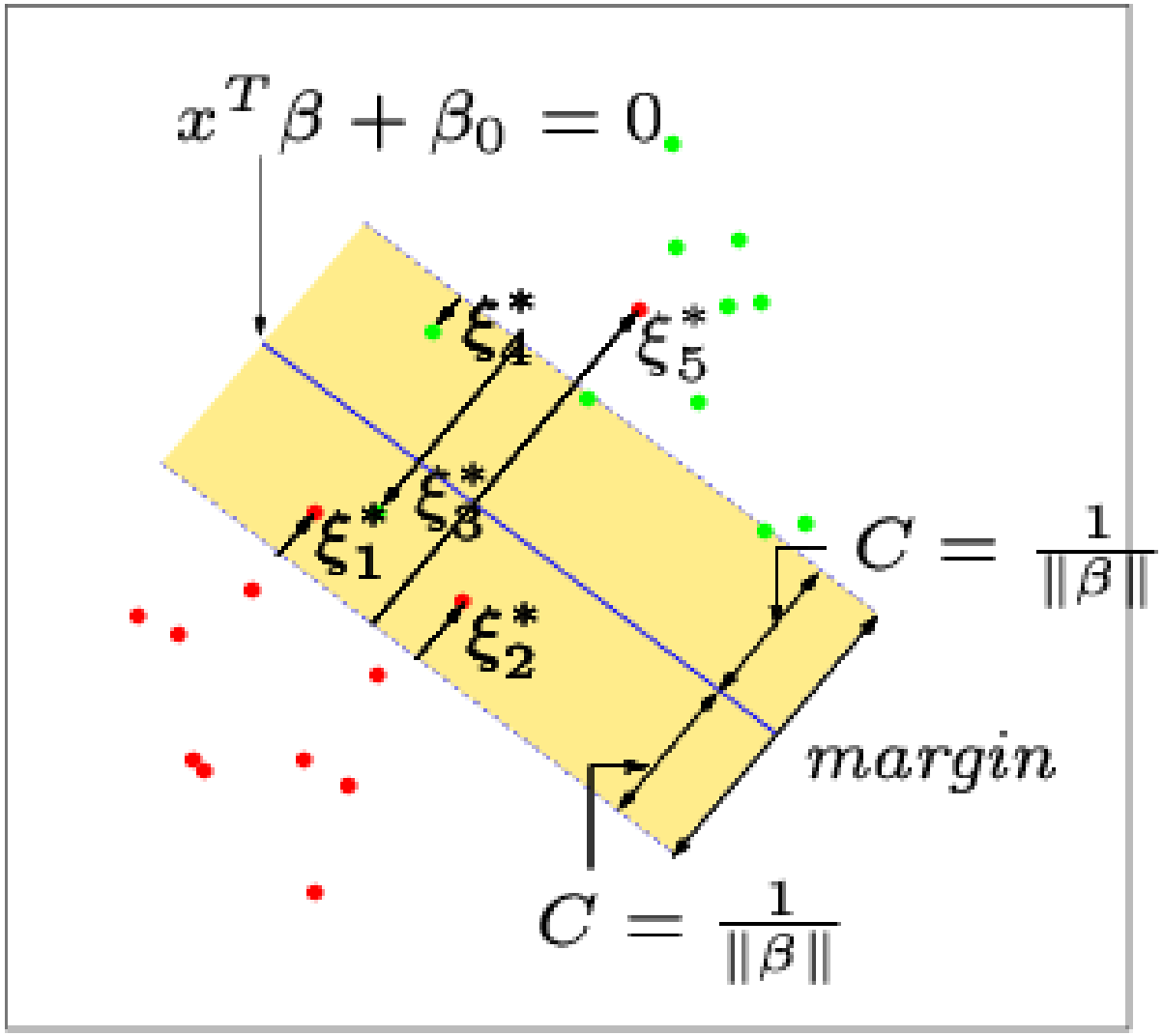
- Of course in practice it is not usually possible to find a hyper-plane that perfectly separates two classes.
- In other words, for any straight line or plane that we draw, there will always be at least some points on the wrong side of the line.
- In this situation we try to find the plane that gives the best separation between the points that are correctly classified, subject to the points on the wrong side of the line not being off by too much.
- It is easier to see with a picture!

Non-separating example

- Let ξ^*_i represent the amount that the i-th point is on the wrong side of the margin (the dashed line).
- Then we want to maximize C subject to

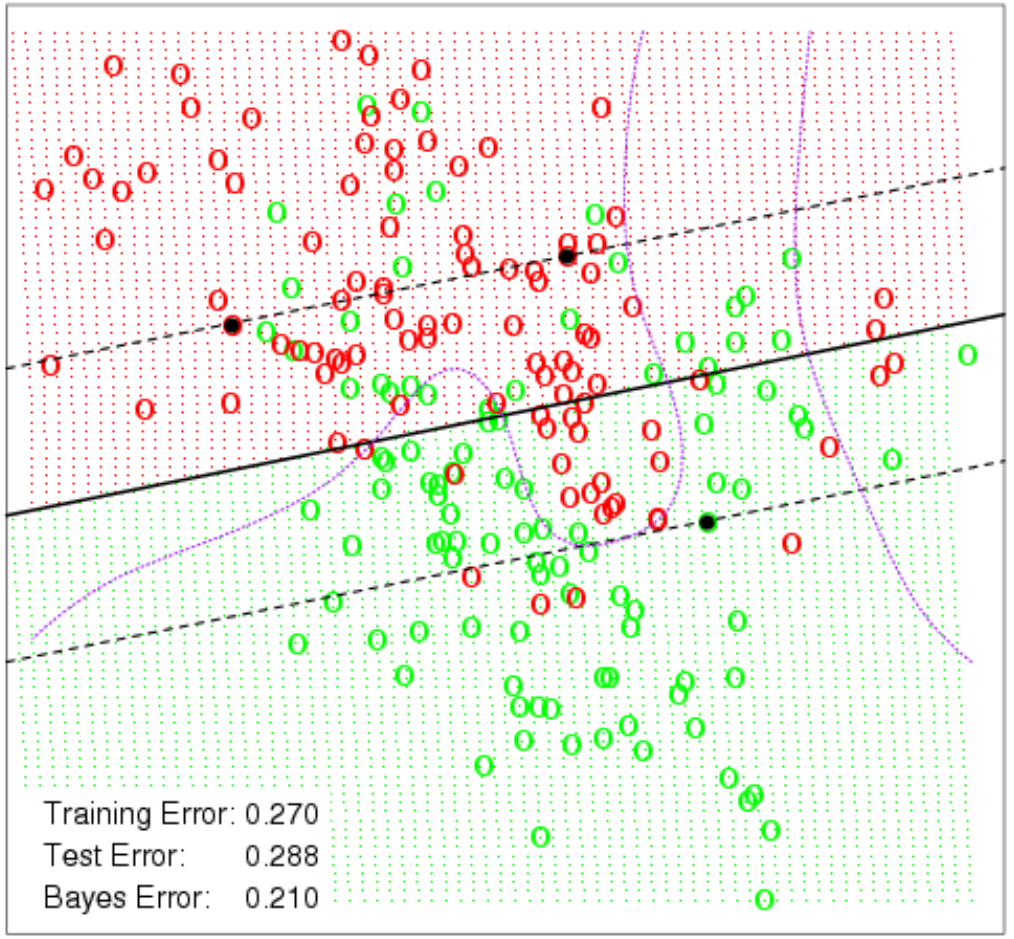
$$\frac{1}{C}\sum_{i=1}^{n}\xi_i^* \leq \text{Constant}$$

- The constant is a tuning parameter that we choose.



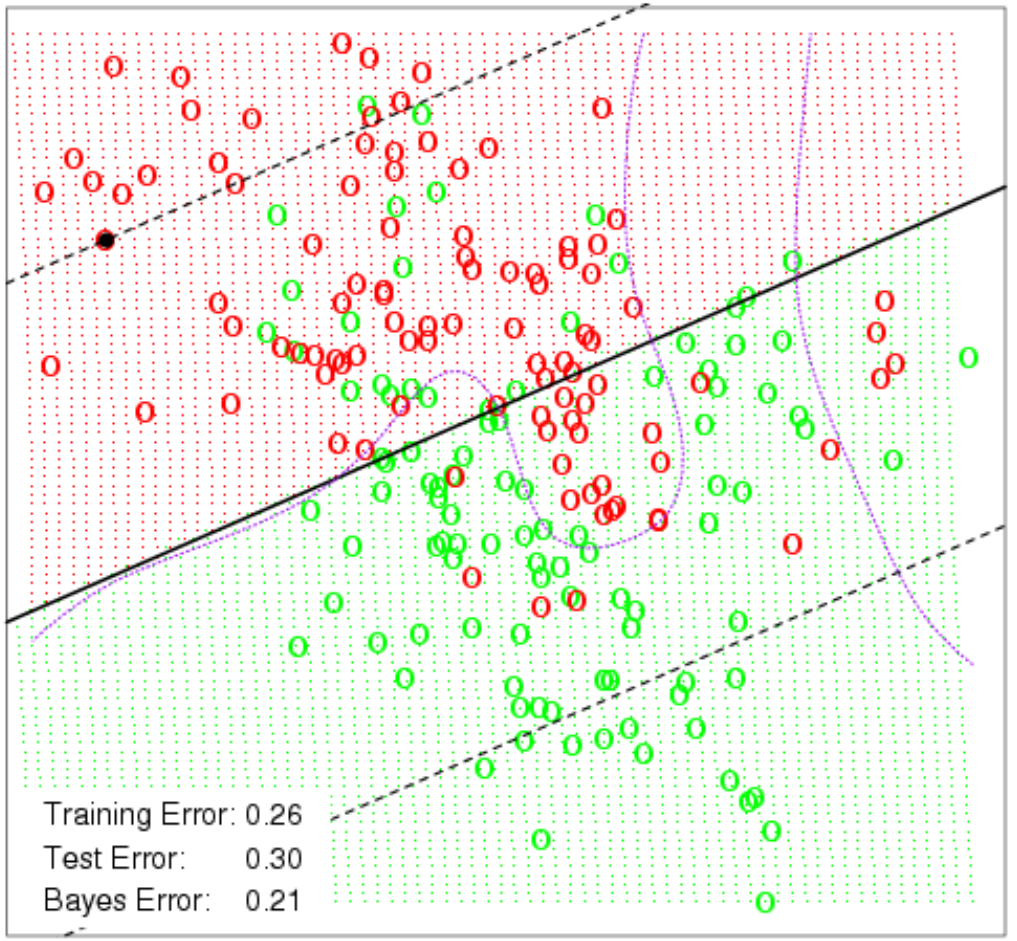
A simulation example with a small constant

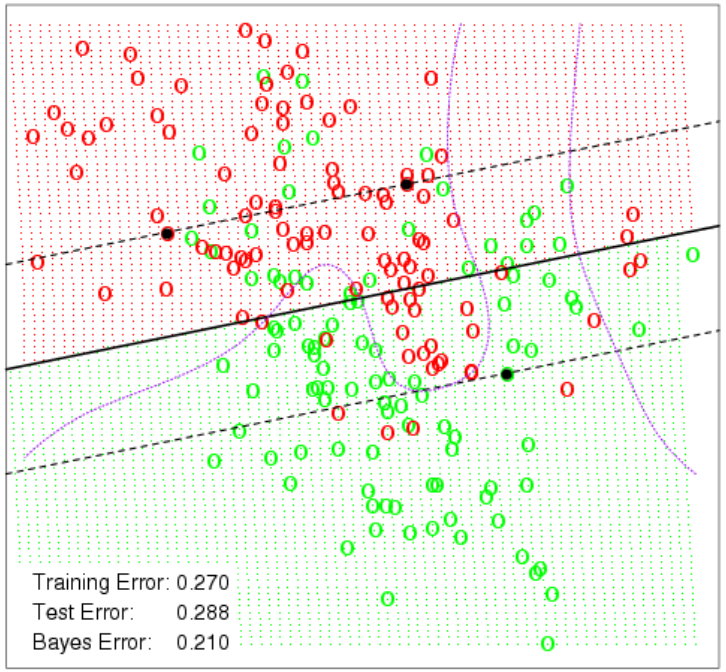
- The distance between the dashed lines represents the margin or 2C.
- The purple lines represent the Bayes decision boundaries



The same example with a larger constant

- Using a larger constant allows for a greater margin and creates a slightly different classifier.
- Notice, however, that the decision boundary must always be linear.





Non-linear support vector classifier

- The support vector classifier is fairly easy to think about. However, because it only allows for a linear decision boundary it may not be all that powerful.
- Recall that linear regression is extended to non-linear regression using a basis function i.e.

$$Y_i = \beta_0 + \beta_1 b_1(X_i) + \beta_2 b_2(X_i) + \cdots + \beta_p b_p(X_i) + \varepsilon_i$$

A basis approach

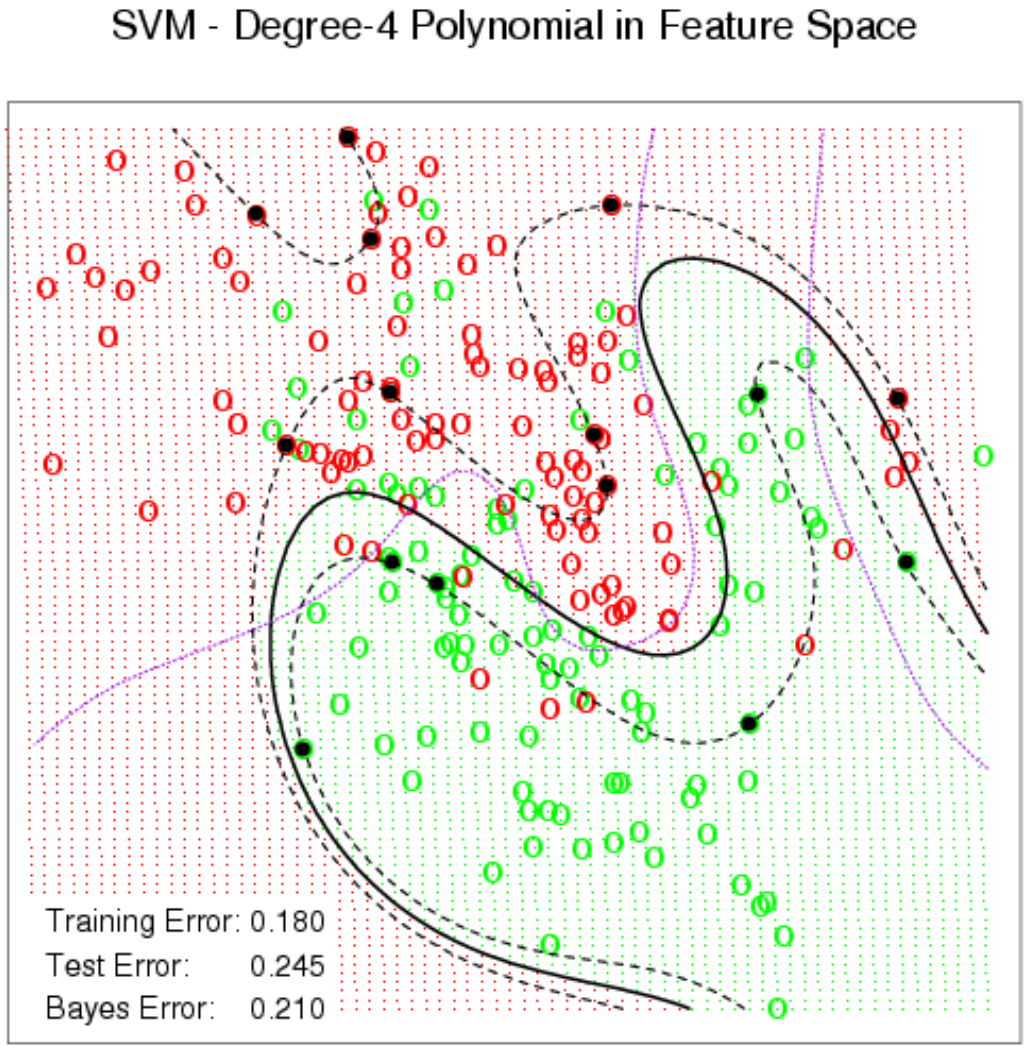
- Conceptually, we can take a similar approach with the support vector classifier.
- The support vector classifier finds the optimal hyper-plane in the space spanned by X_1, X_2,…, X_p.
- Instead we can create transformations (or a basis) $b_1(x)$, $b_2(x)$, …, $b_M(x)$ and find the optimal hyper-plane in the space spanned by $b_1(\mathbf{X})$, $b_2(\mathbf{X})$, …, $b_M(\mathbf{X})$.
- This approach produces a linear plane in the transformed space but a non-linear decision boundary in the original space.
- This is called the support vector machine classifier.

In reality

- While conceptually the basis approach is how the support vector machine works, there are some technical details which means that we don't actually choose $b_1(x)$, $b_2(x)$, …, $b_M(x)$.
- Instead we choose a kernel function which takes the place of the basis.
- Common kernel functions include
  - Linear
  - Polynomial
  - Radial Basis
  - Sigmoid

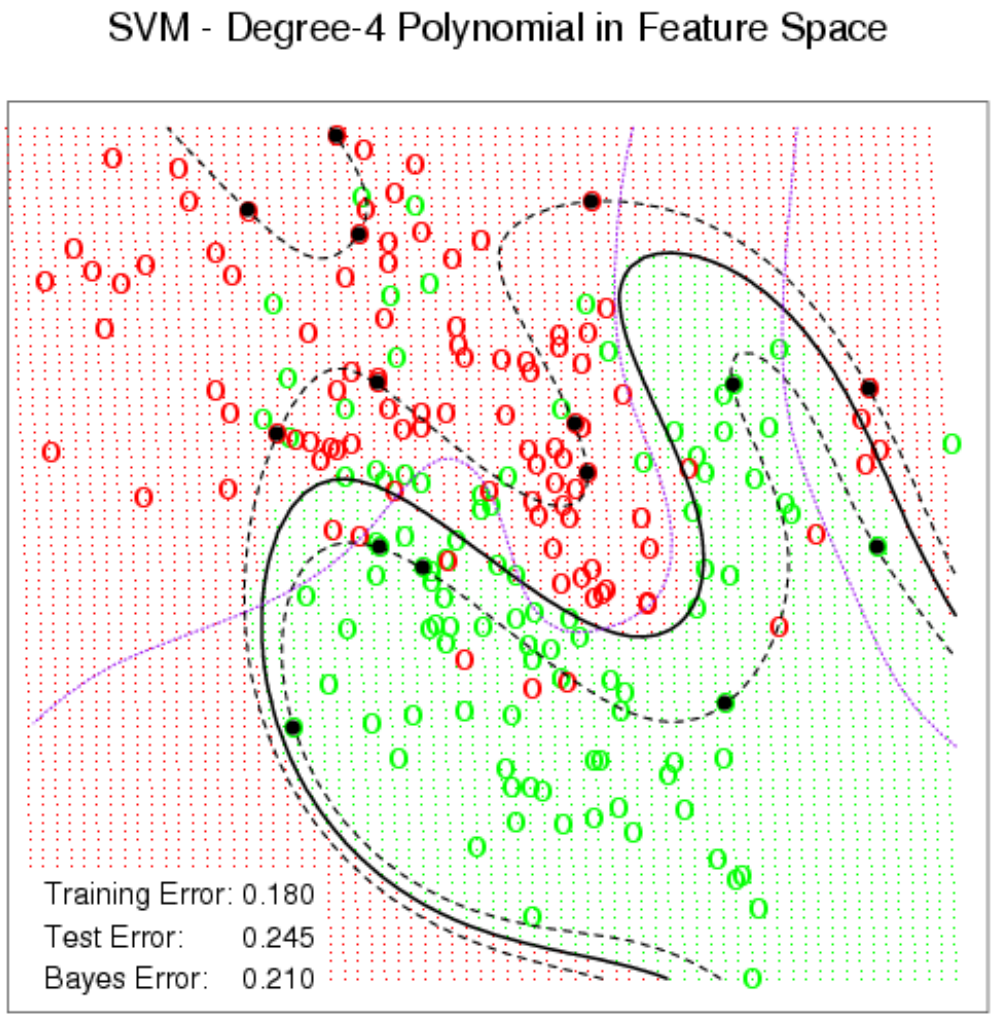
Polynomial kernel on Sim data

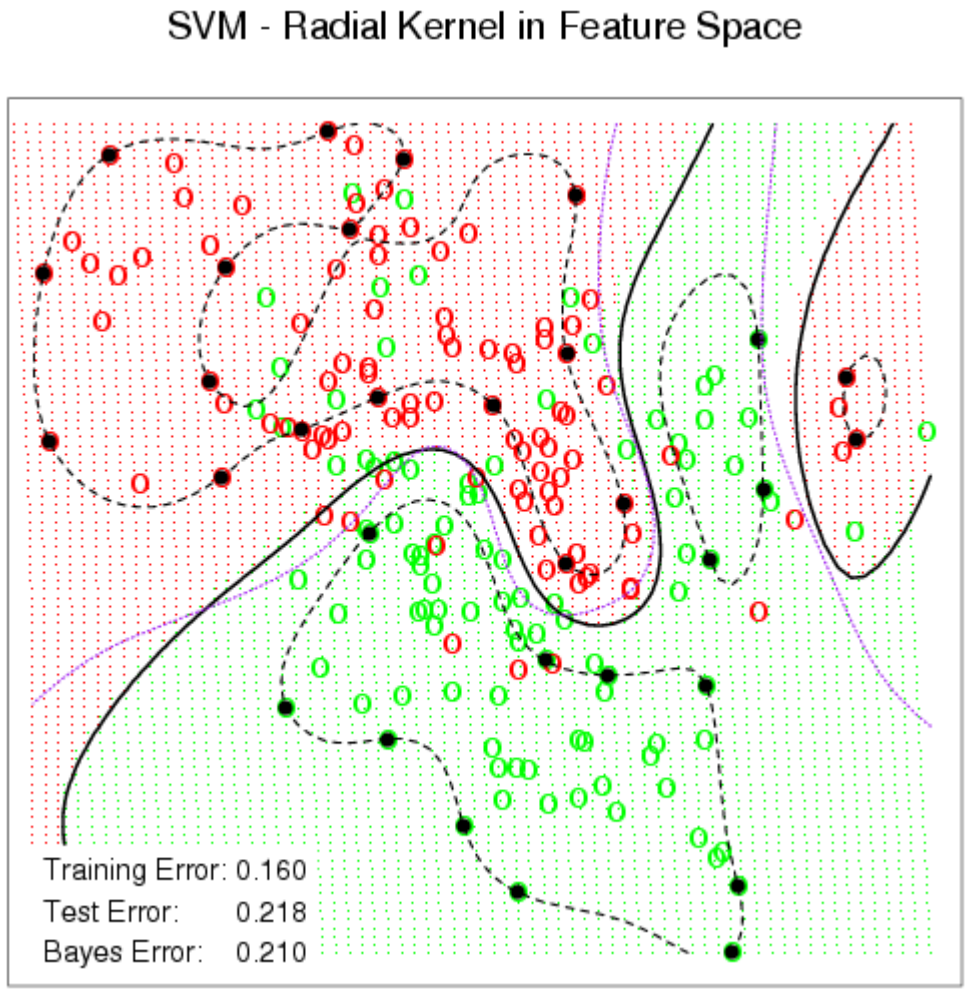
- Using a polynomial kernel we now allow SVM to produce a non-linear decision boundary.
- Notice that the test error rate is a lot lower.



Radial basis kernel

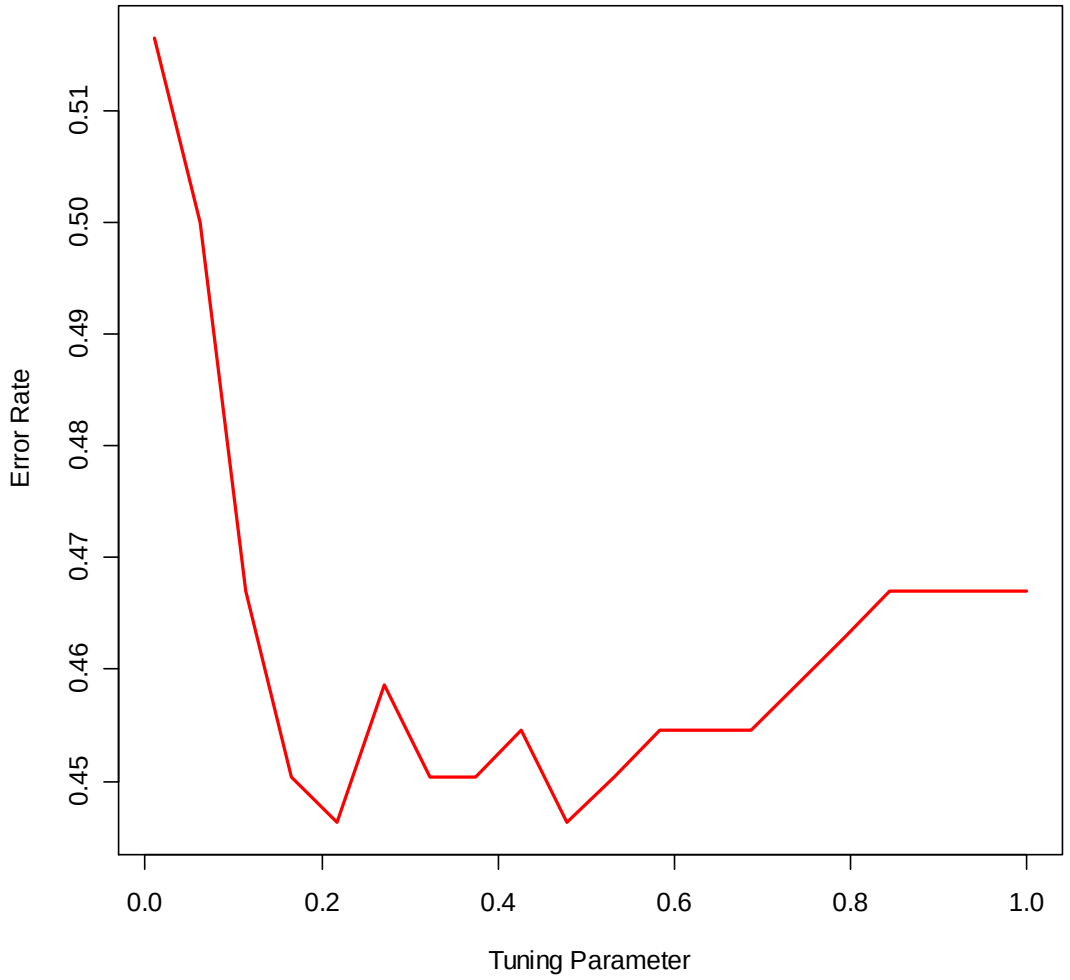
- Using a radial basis kernel you get an even lower error rate.





Error rates on S&P data

- Here results of a radial basis kernel are shown with the error rate for different values of the tuning parameter.
- The results on this data were similar to GAM but not as good as boosting.



SVMs: more than 2 classes?

The SVM as defined works for $K = 2$ classes. What do we do if we have $K > 2$ classes?

OVA One versus All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

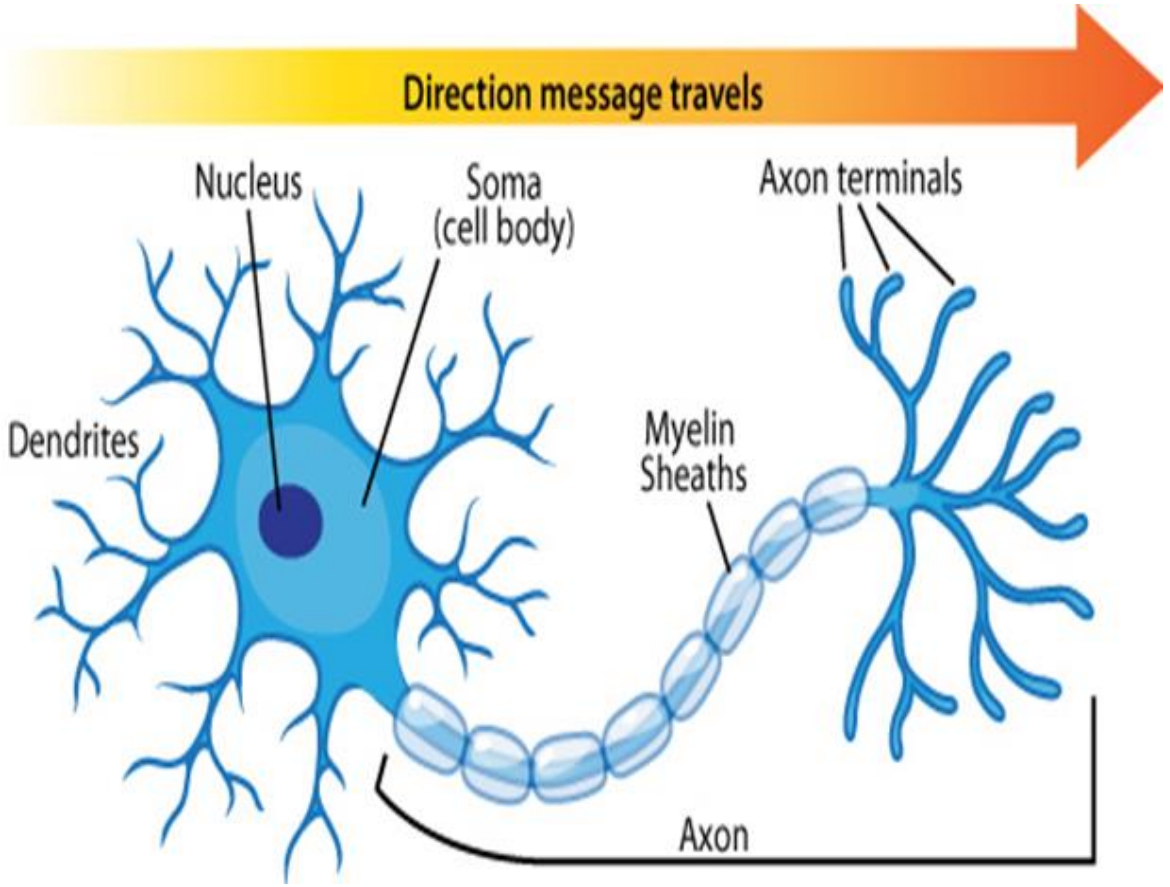
OVO One versus One. Fit all $\binom{K}{2}$ pairwise classifiers $\hat{f}_{k\ell}(x)$. Classify x^* to the class that wins the most pairwise competitions.

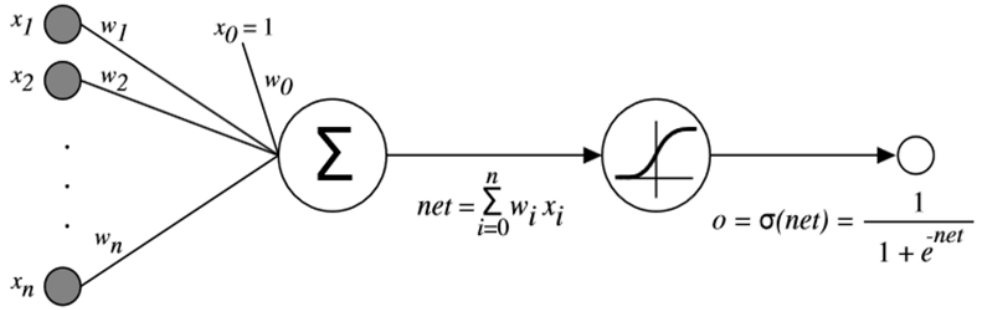
Which to choose? If K is not too large, use OVO.

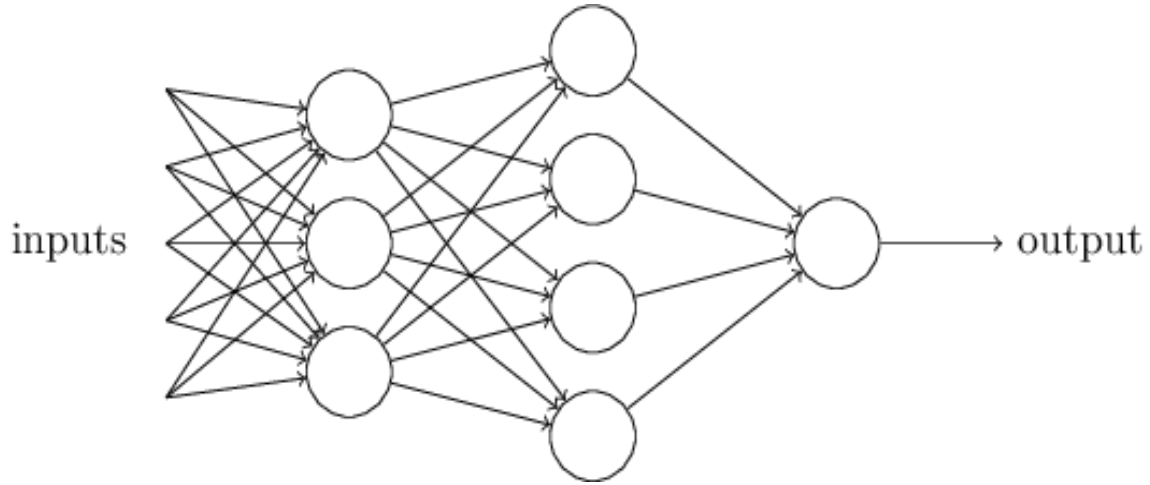
Artificial neural networks

- many approaches, we shall cover the basic ideas
- currently revival of interest, see deep neural networks
- http://www.deeplearningbook.org

Brain analogy





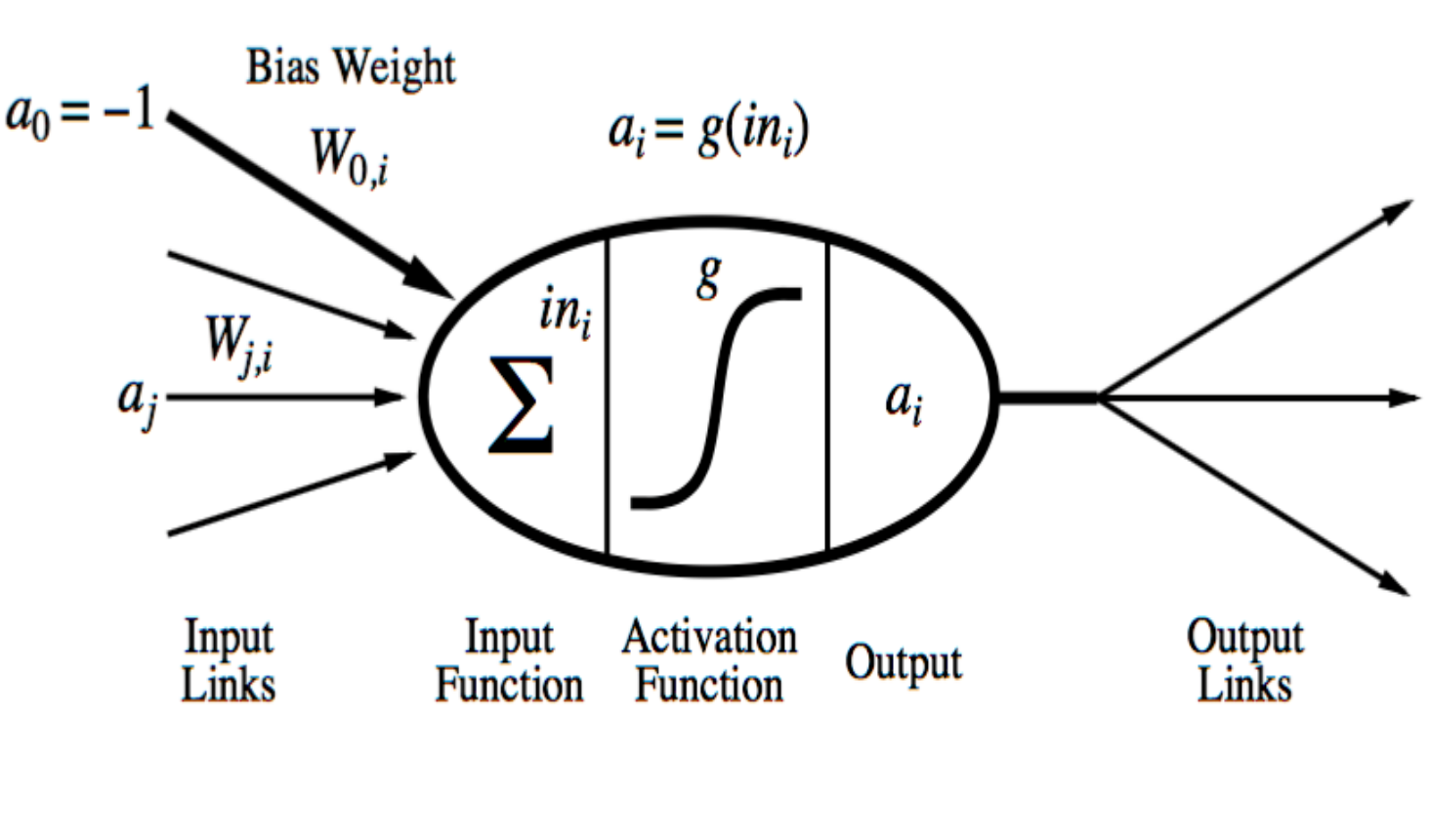


learning by error backpropagation

Neuron

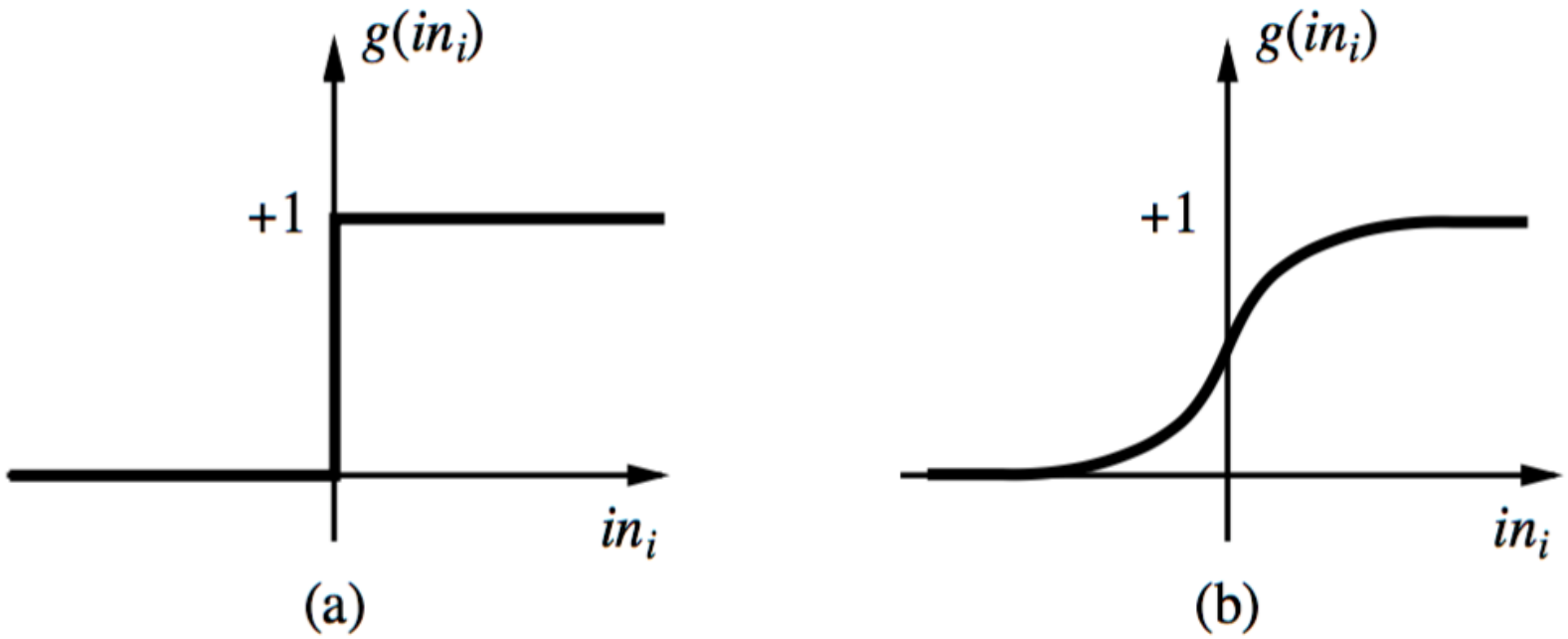
- Computational units, passing messages (information) in the network, typically organized into layers

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



Activation functions

- examples: step function, sigmoid (logistic)

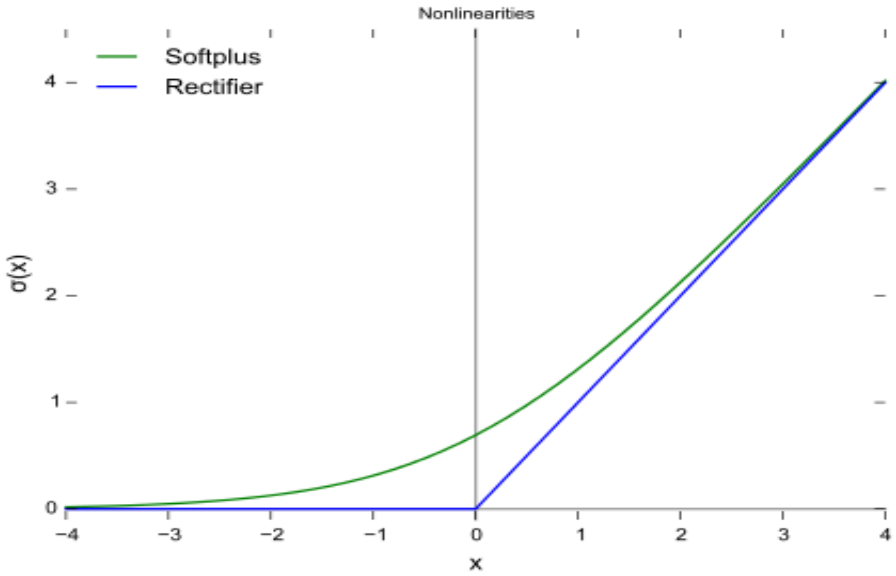


Activation functions

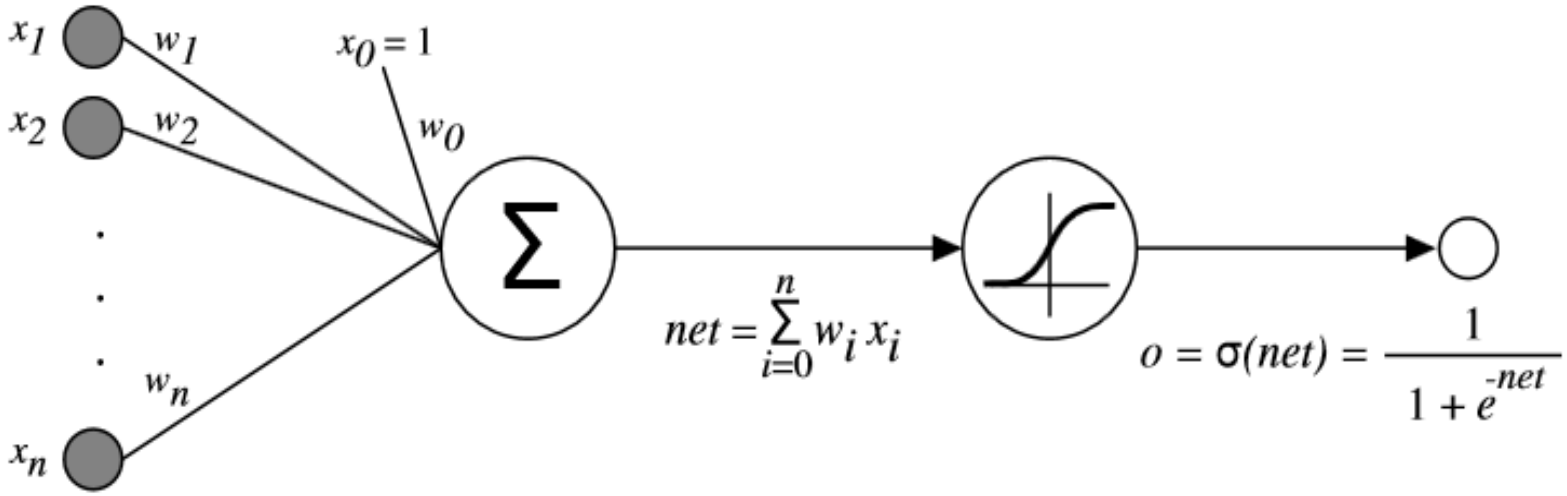
- ReLU (rectified linear unit)

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

- softplus / approximation of ReLU with continuous derivation
- $f(x) = \ln(1+e^x)$



Why nonlinear?



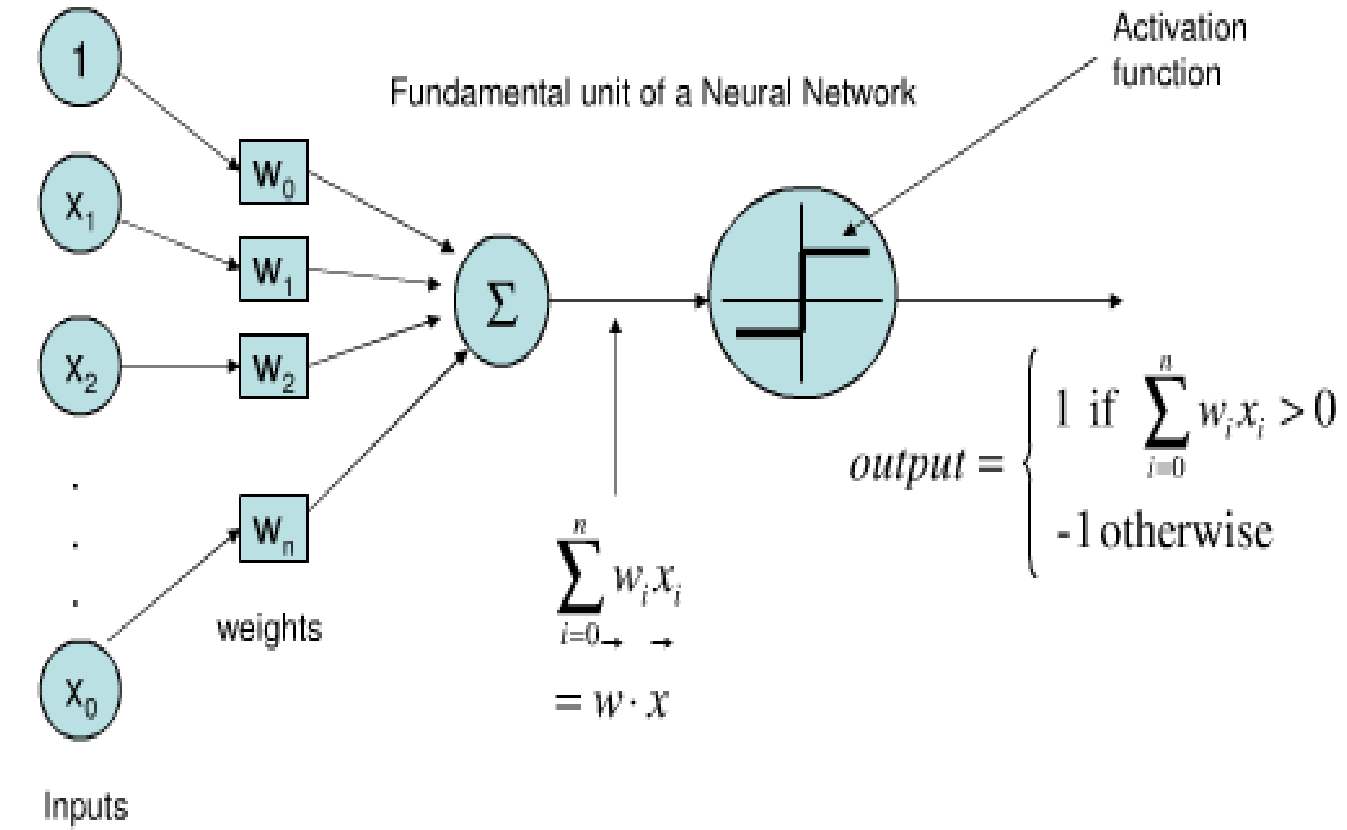
What is a derivative of a sigmoid?

Backpropagation learning algorithm for NN

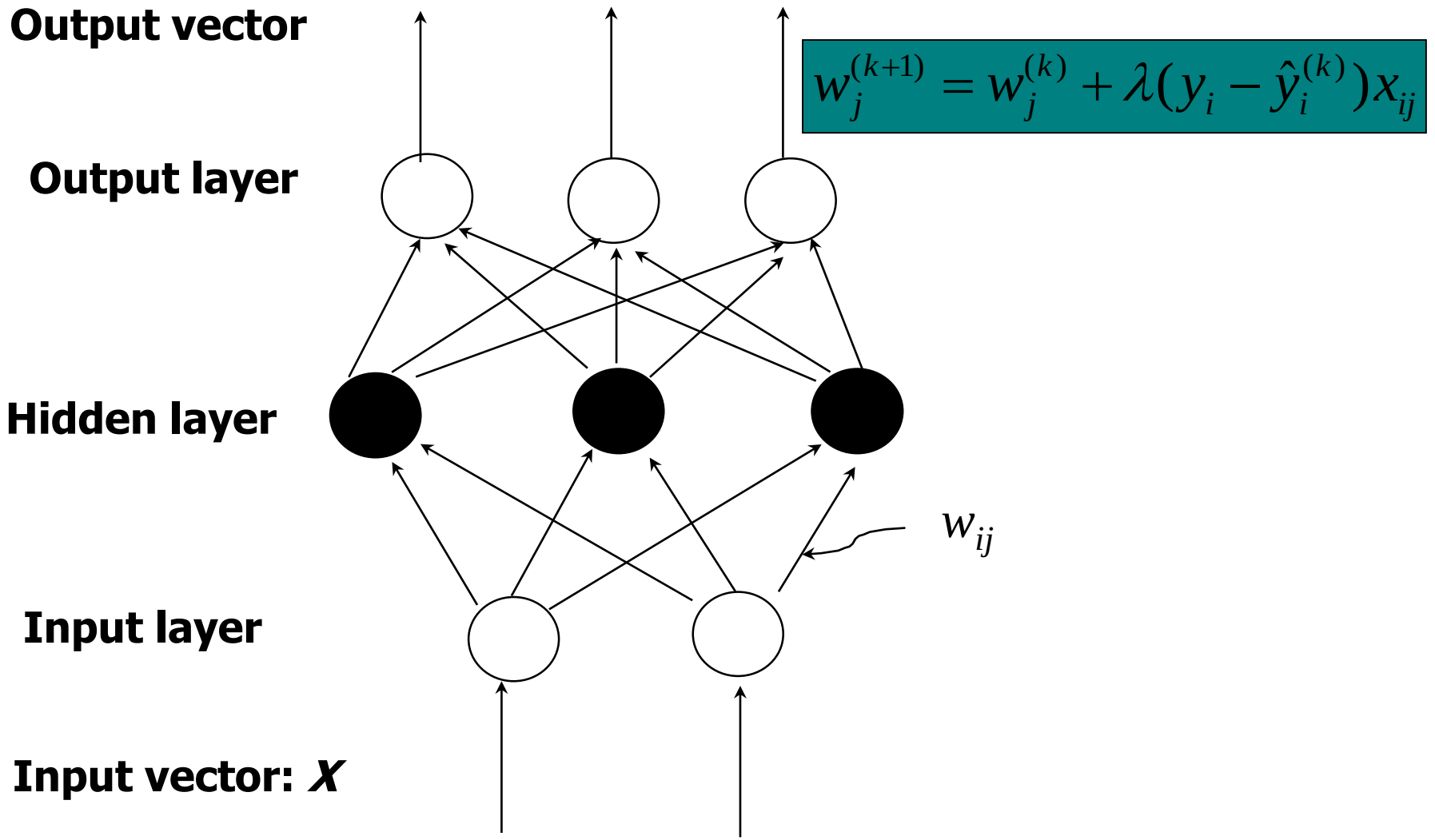
- Backpropagation: A neural network learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- A neural network: A set of connected input/output units where each connection has a weight associated with it
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples
- Also referred to as connectionist learning due to the connections between units

Perceptron

- model nevrona



A multi-layer feed-forward NN

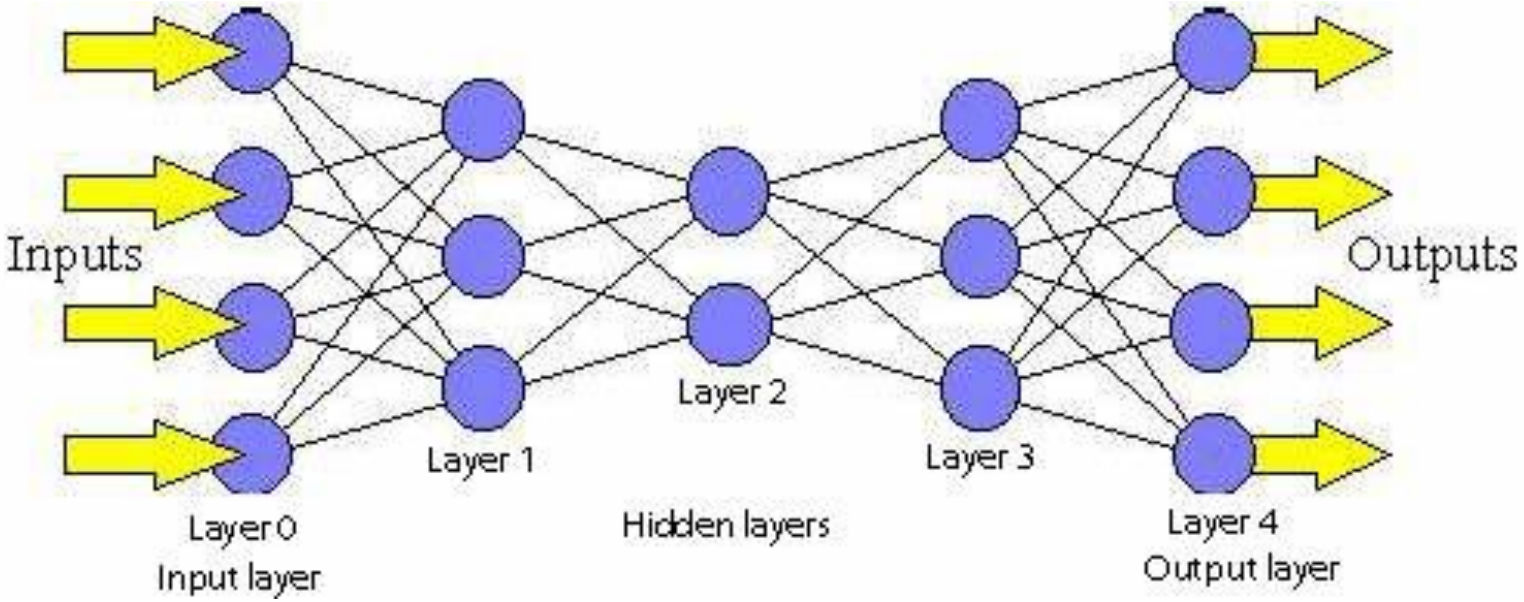


How a multi-layer NN works?

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary; if more than 1 hidden layer is used, the network is called deep neural network
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- If we have backwards connections the network is called recurrent neural network
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

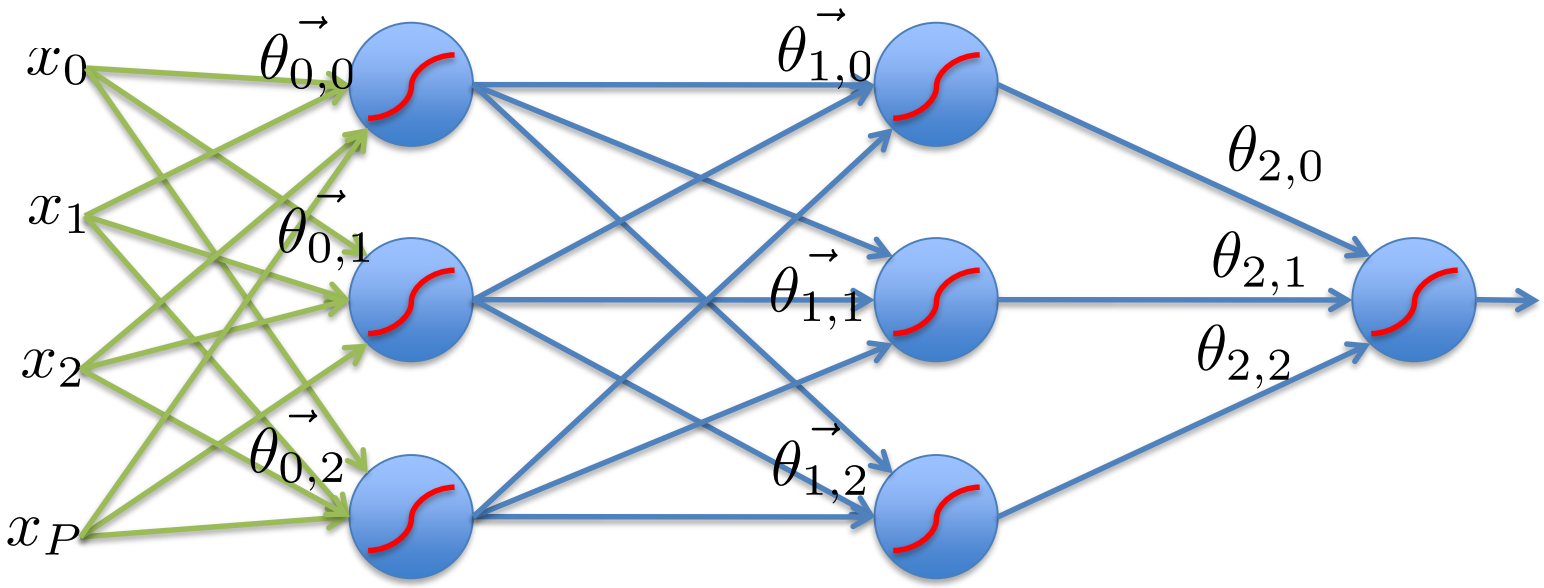
Feed-Forward Network

- neurons are activated progressively throug layers from input to output



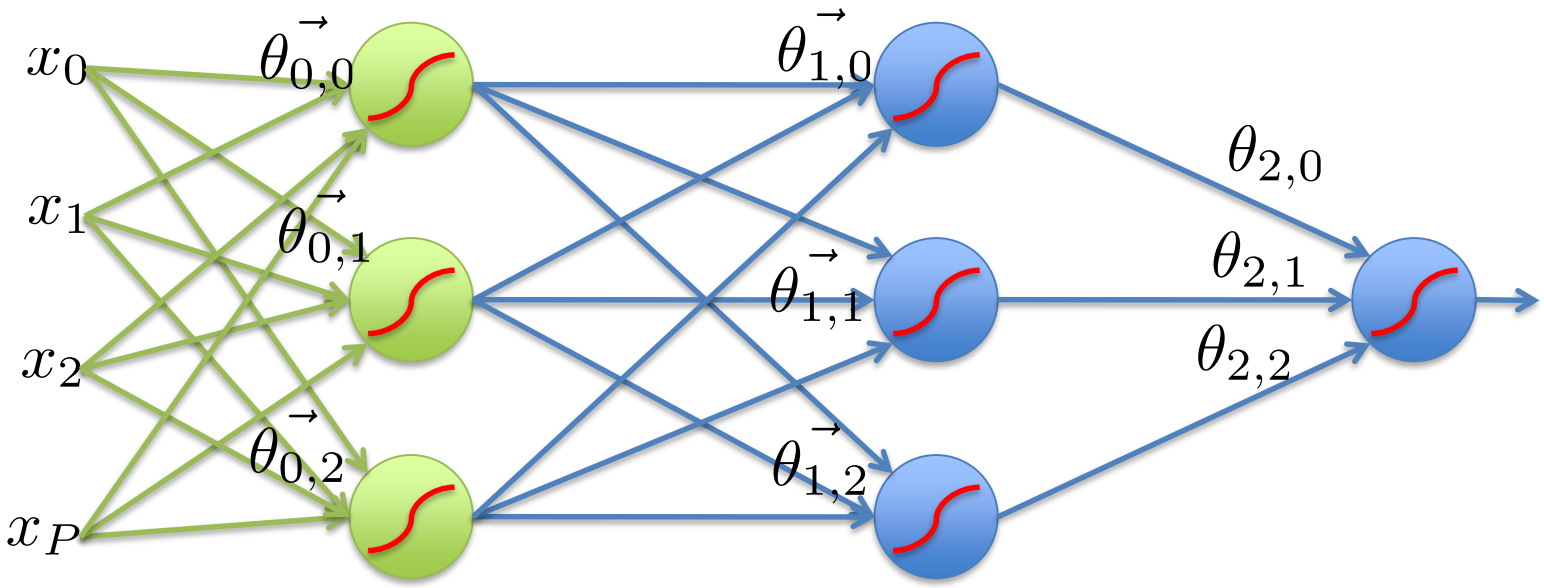
Feed-Forward Network

- Values are propagated through the network to the output, which returns the prediction

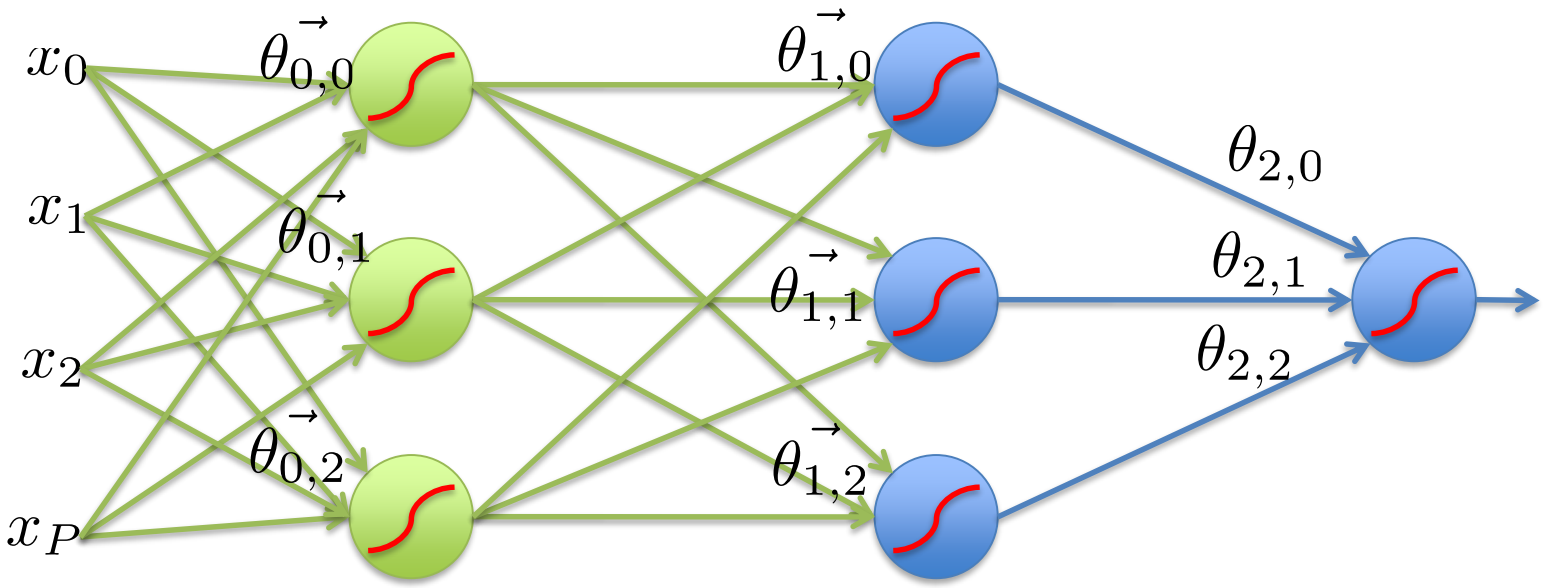


Next 6 slides by Andrew Rosenberg

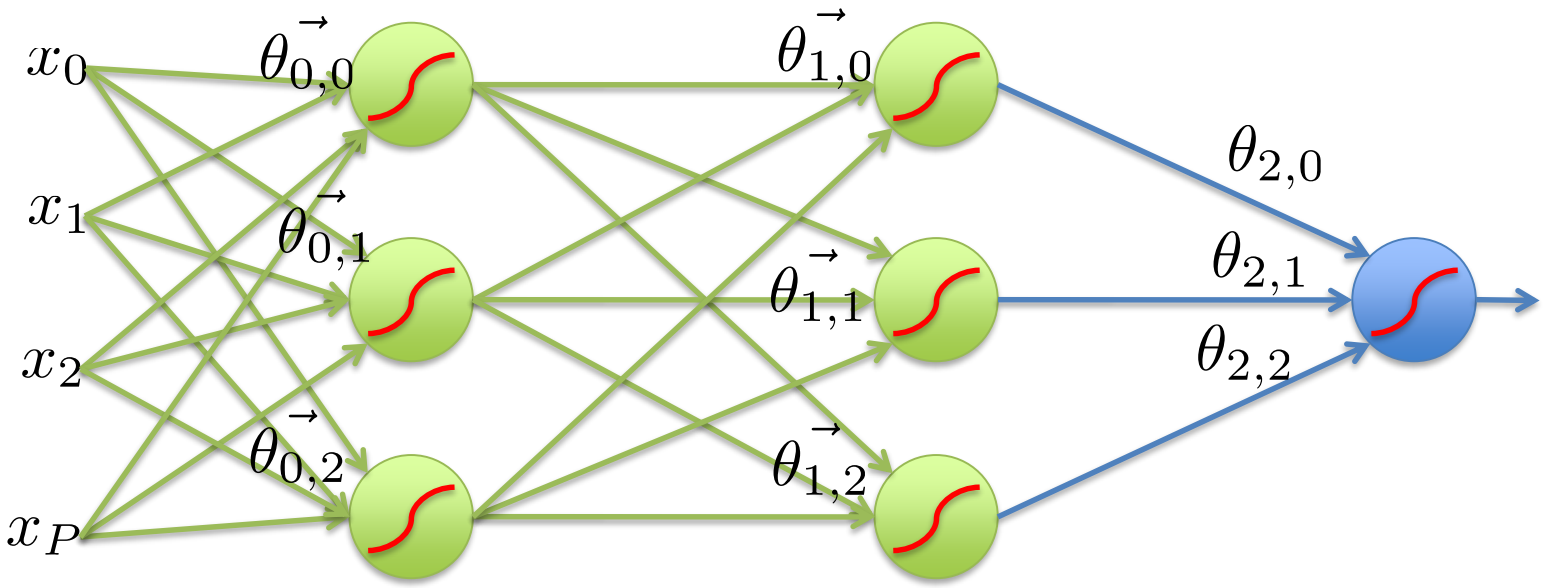
Feed-Forward Networks



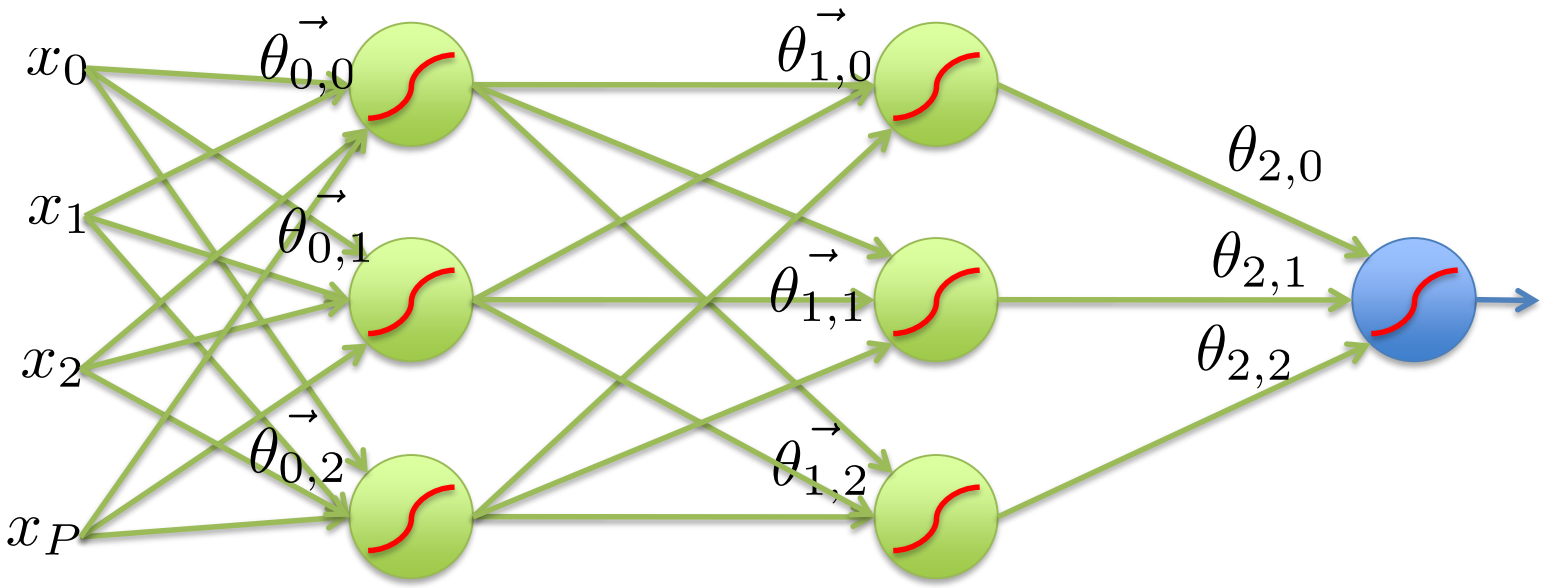
Feed-Forward Networks



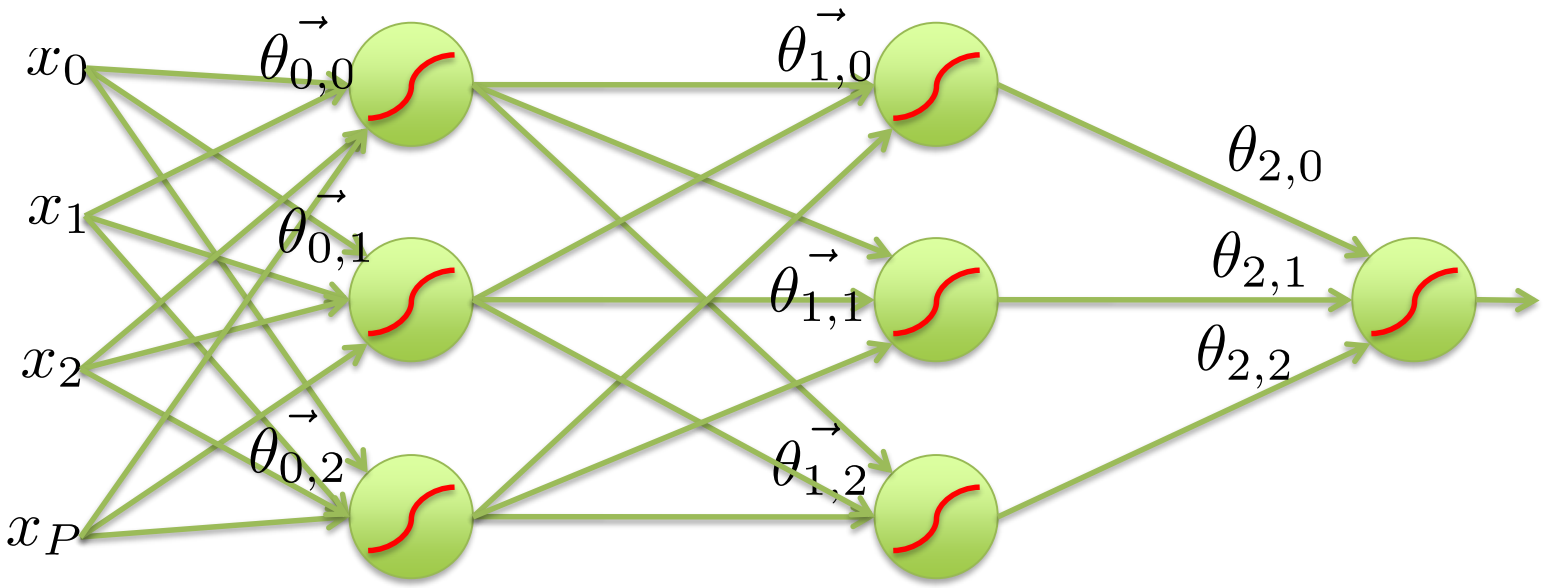
Feed-Forward Networks



Feed-Forward Networks



Feed-Forward Networks



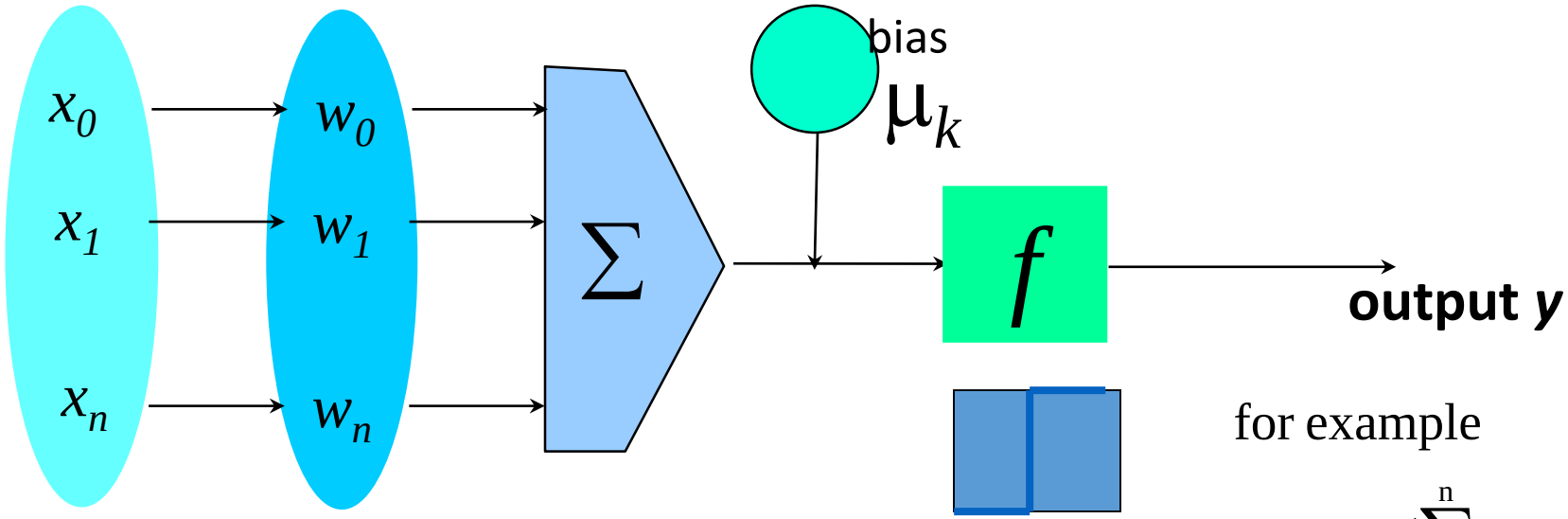
Defining a network topology

- Decide the network topology: Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the input values for each attribute measured in the training tuples to [0.0—1.0]
- One input unit per domain value, each initialized to 0
- Output, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is unacceptable, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation algorithm

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- 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.)

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.

Neural network as a classifier

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  - Poor interpretability: difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs *and outputs*
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques exist for the extraction of explanations from trained neural networks

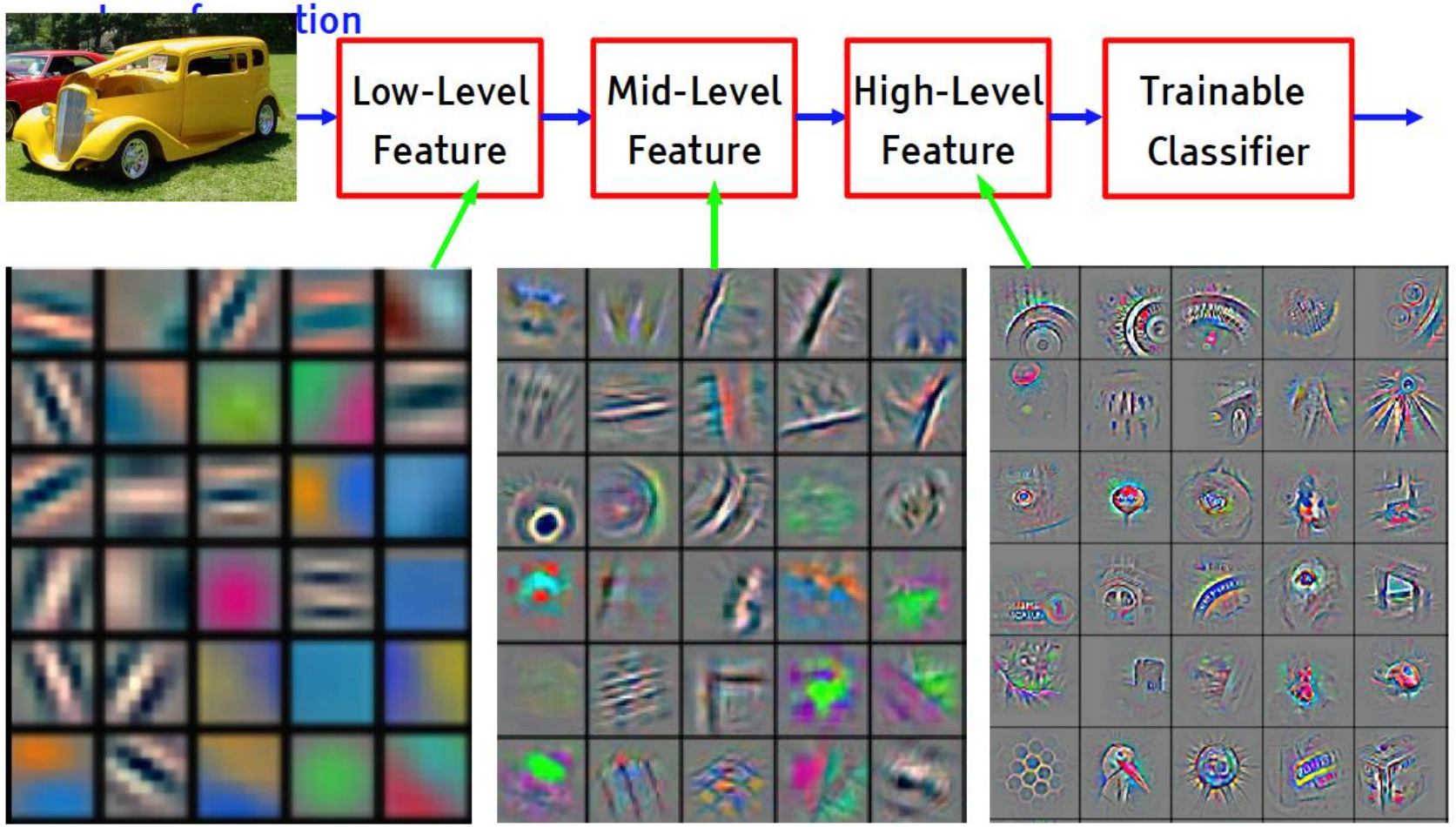
Efficiency and interpretability

- Efficiency of backpropagation: Each epoch (one iteration through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be exponential to n, the number of inputs, in worst case
- For easier comprehension: Rule extraction by network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- Sensitivity analysis: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

Prevention of overfitting

- Weight-decay
- Weight-sharing
- Early stopping
- Model averaging
- Bayesian fitting of neural nets
- Dropout
- Generative pre-training
- etc.

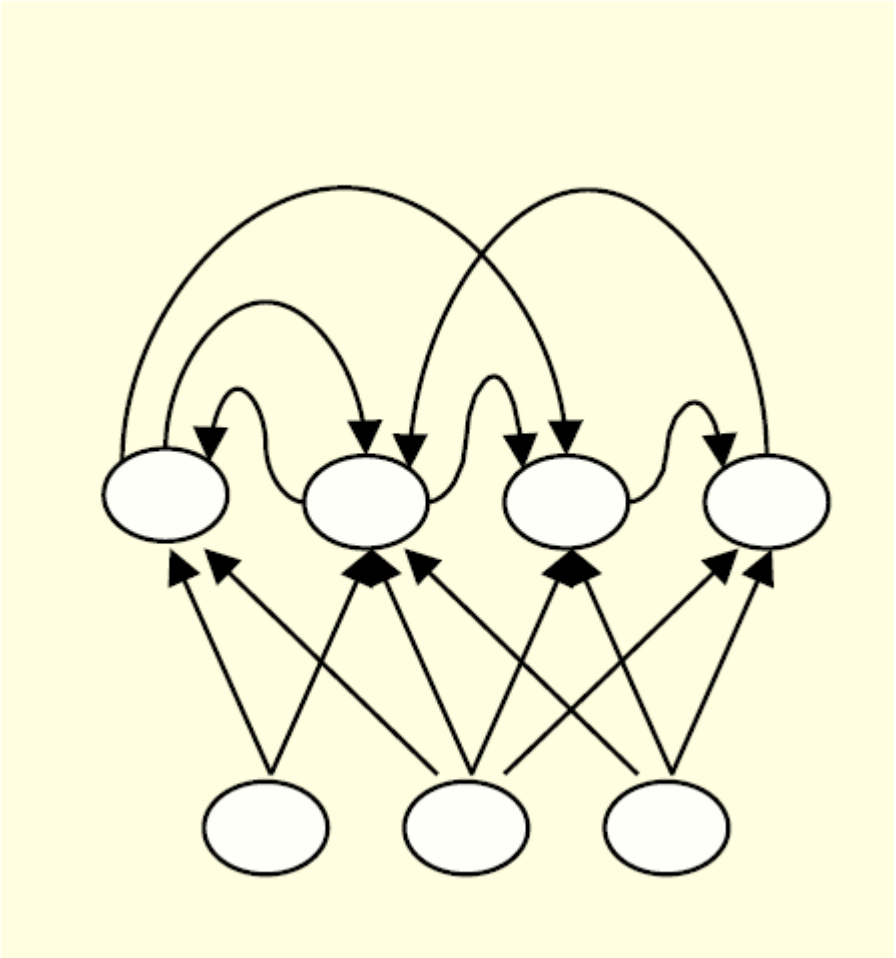
Deep learning =
learning of hierarchical represenation



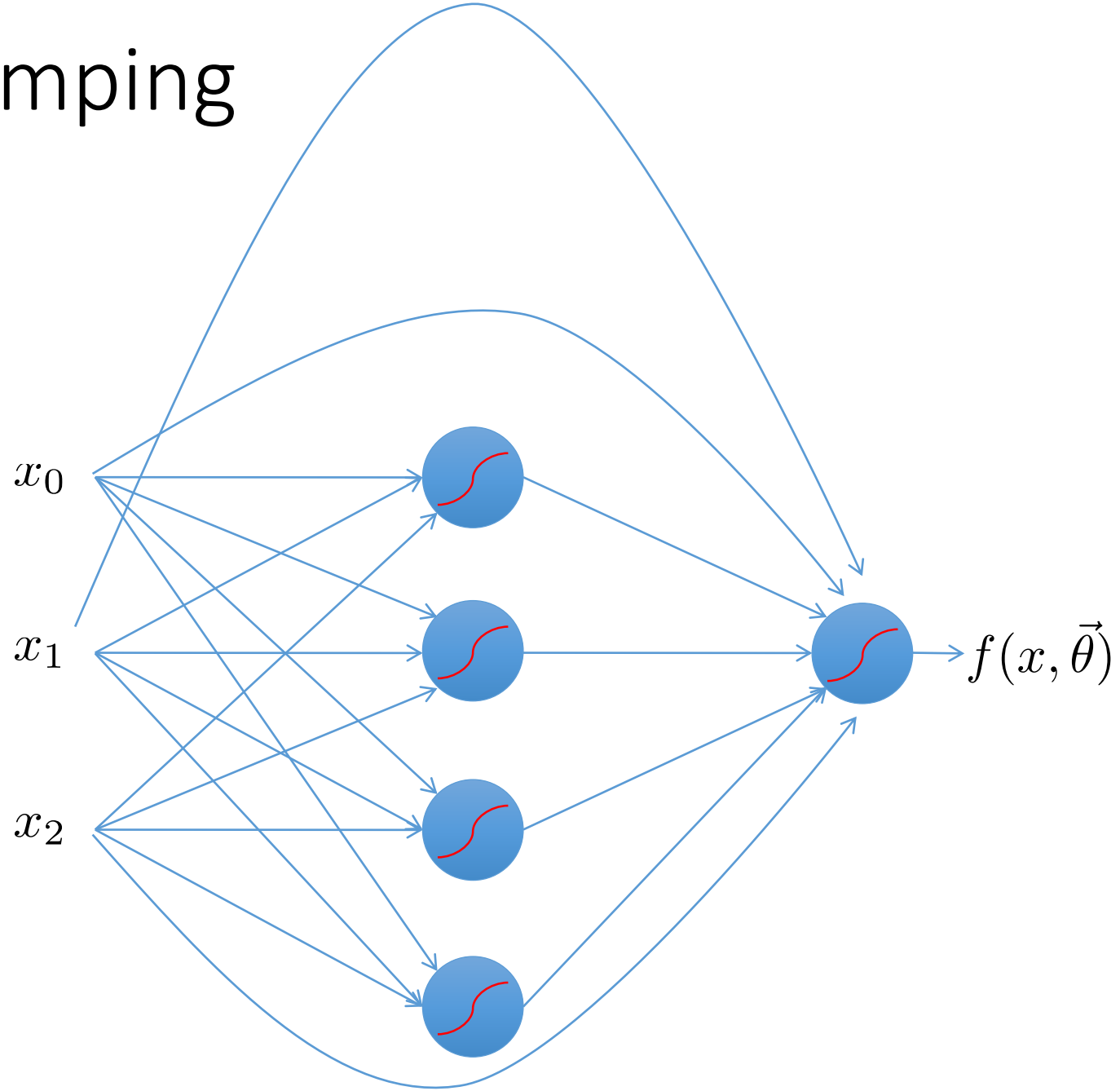
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Recurrent networks

- back connections
- biologically more realistic
- store information from the past
- more difficult to learn

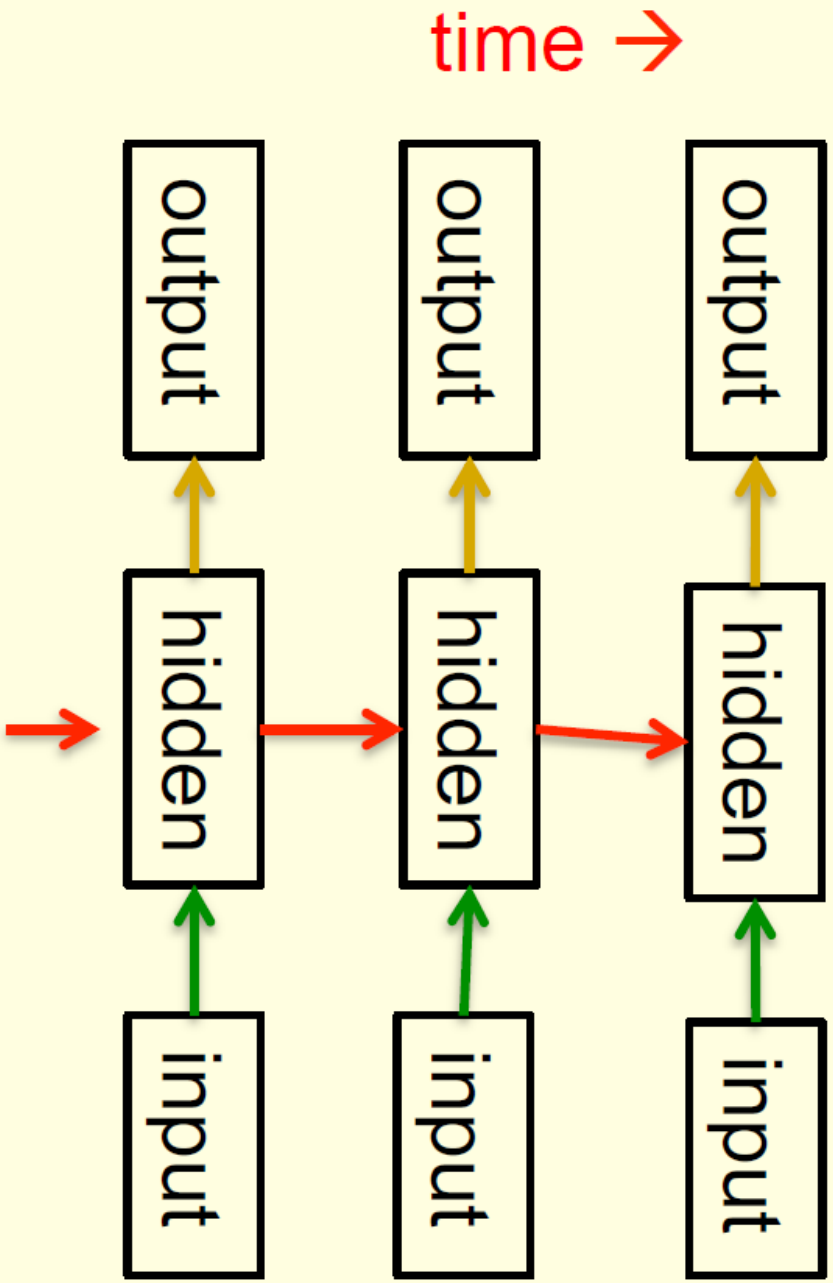


Level jumping



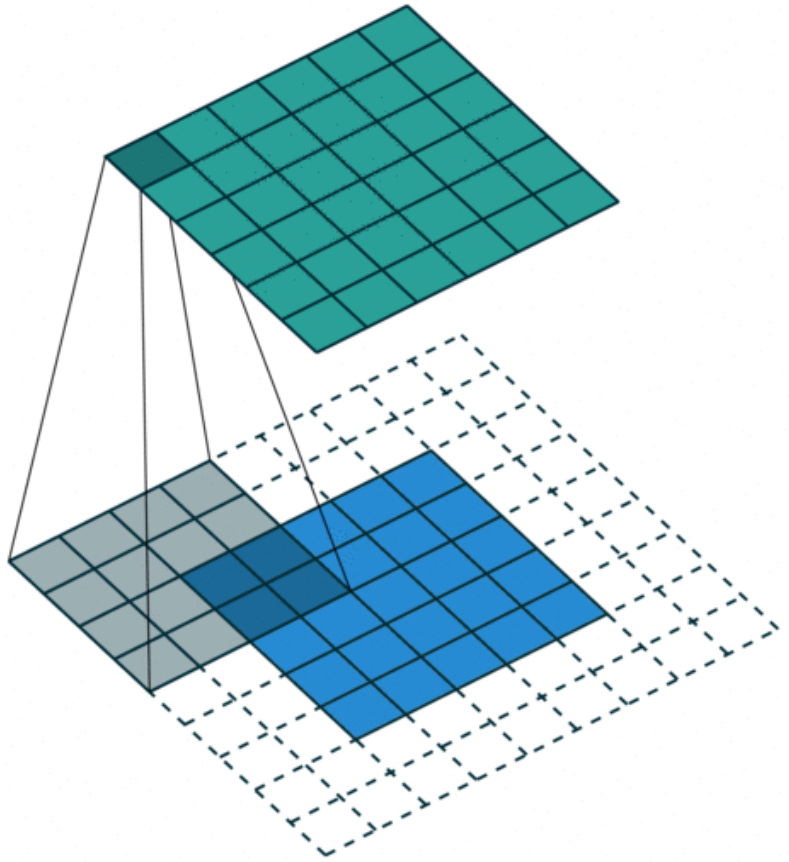
Recurrent networks for sequence learning

- equivalent to deep networks with one hidden level per time slot
- but: hidden layers share weight (less parameters)



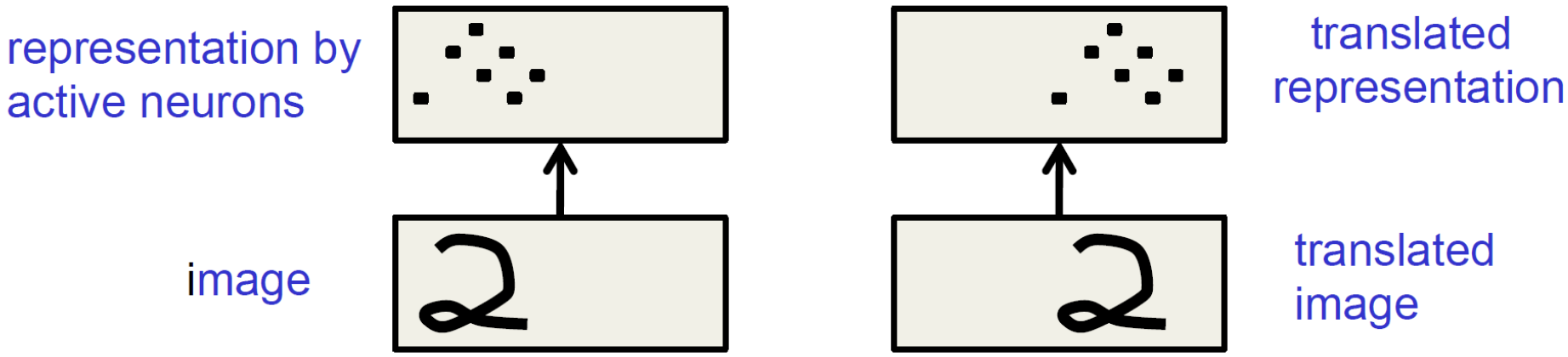
Convolutional neural networks (CNN)

- currently, the most successful approach in image analysis, successful in language
- idea: many copies of small detectors used all over the image, recursively combined,
- detectors are learned, combination are learned



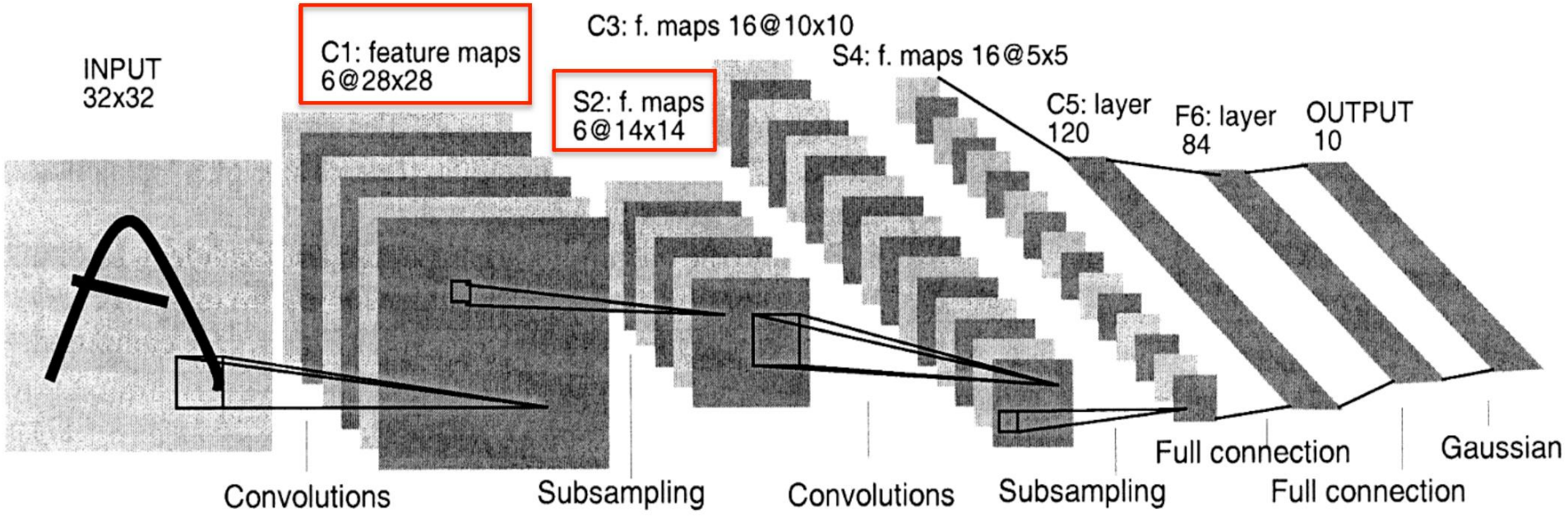
Convolutional networks

- a useful feature is learned and used on several positions
- prevents dimension hopping
- max-pooling



LeNet5 architecture

- handwritten digit recognition



Errors

- 80 errors in 10,000 test cases

Sources

- Ian Goodfellow and Yoshua Bengio and Aaron Courville: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org
- Jürgen Schmidhuber: Deep learning in neural networks: An overview. *Neural Networks* 61:85-117, 2015.
- Geoffrey Hinton, Nitish Srivastava, Kevin Swersky: *Neural Networks for Machine Learning*. Coursera, 2017
- Richard Socher: *Deep Learning for Natural Language Processing*. Coursera, 2017
- Keras library in R or python
- TensorFlow
- PyTorch