Discriminative Models for Text Classification

Mausam

(Slides by Michael Collins, Emily Fox, Dan Jurafsky, Dan Klein, Chris Manning, Ray Mooney, Noah Smith, Andrew Rosenberg, Dan Weld, Alex Yates, Luke Zettlemoyer)

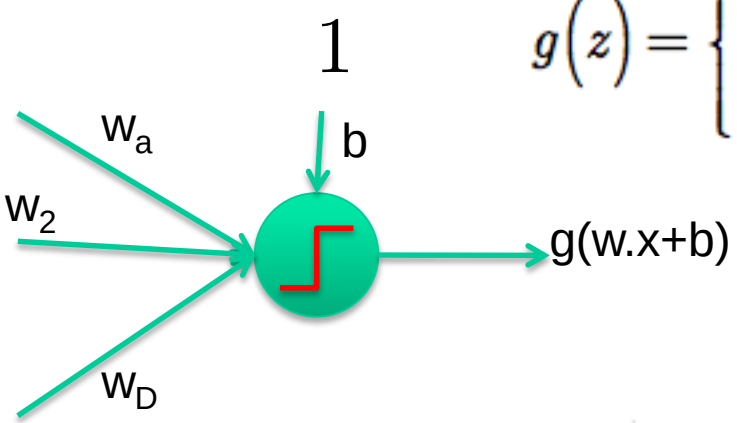
ML Classifiers

- Two probabilistic approaches to predicting classes y*
  - Joint: work with a *joint* probabilistic model of the data, weights are (often) local conditional probabilities
    - E.g., represent p(y,x) as Naïve Bayes model, compute $y^* = \text{argmax}_y\ p(y,x)$
  - Conditional: work with *conditional* probability p(y|x)
    - We can then direct compute $y^* = \text{argmax}_y\ p(y|x)$ Can develop *feature rich* models for p(y|x).

- But, why estimate a distribution at all?
  - Linear predictor: $y^* = \text{argmax}_y\ w \bullet \phi(x,y)$
  - Perceptron algorithm
    - Online
    - Error driven
    - Simple, additive updates

Perceptron Loss

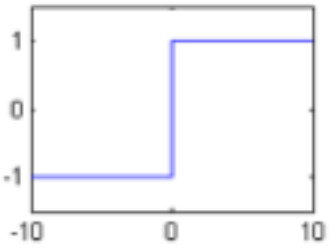
- Classification squashing function



$$g(z) = \begin{cases} -1 \text{ when } z < 0 \\ +1 \text{ when } z \geq 0 \end{cases}$$

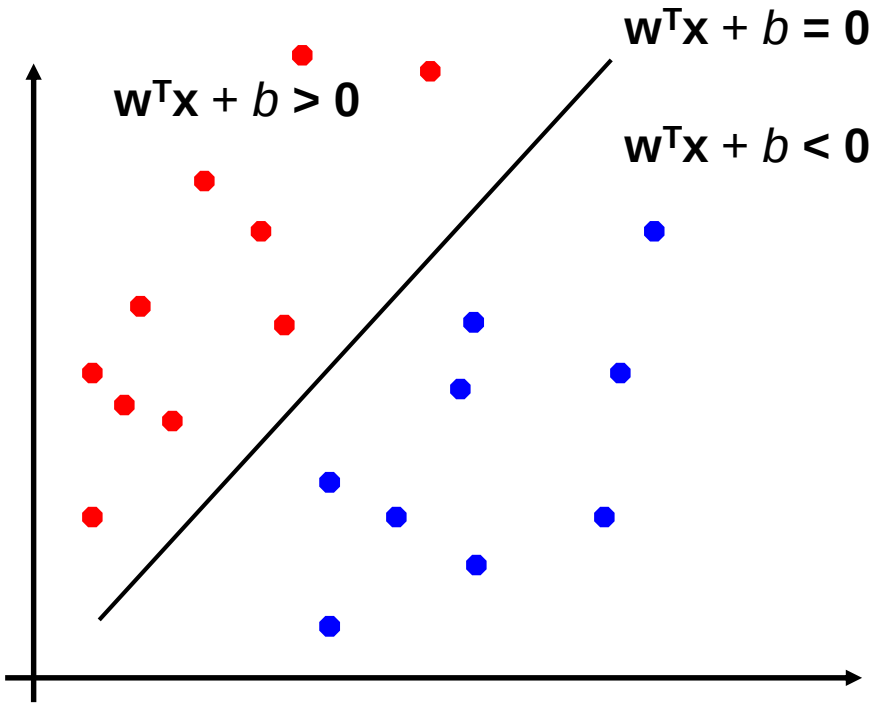
Perceptron

- Strictly classification error



Perceptron: Linear Separators

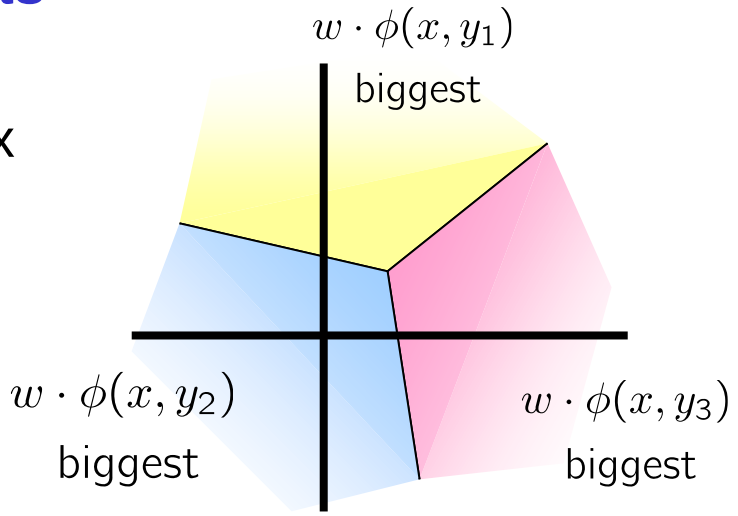
- Binary classification can be viewed as the task of separating classes in feature space:



$o(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$

Multiclass Perceptron Decision Rule

- Compare all possible outputs
  - Highest score wins
  - Boundaries are more complex
  - Harder to visualize



$$y^* = \arg\max_y w \cdot \phi(x, y)$$

Linear Models: Perceptron

- The perceptron algorithm
  - Iteratively processes the training set, reacting to training errors
  - Can be thought of as trying to drive down training error
- The (online) perceptron algorithm:
  - Start with zero weights
  - Visit training instances (x_i,y_i) one by one
    - Make a prediction

$$y^* = \arg\max_y w \cdot \phi(x_i, y)$$

    - If correct ($y^*==y_i$): no change, goto next example!
    - If wrong: adjust weights

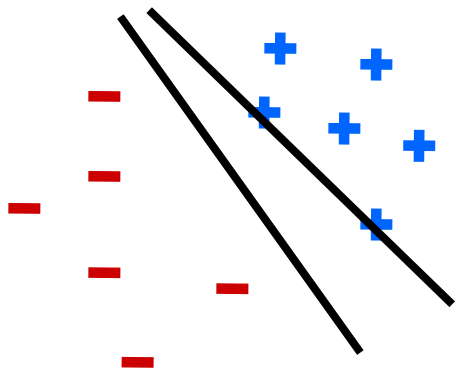
$$w = w + \eta\{\phi(x_i, y_i) - \phi(x_i, y^*)\}$$

Properties of Perceptrons

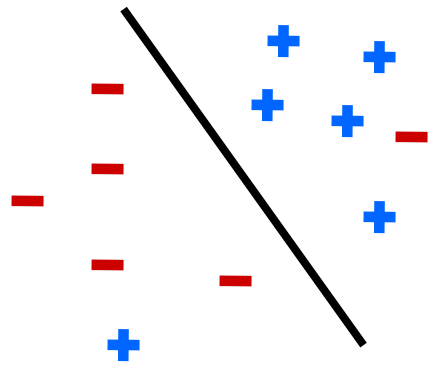
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{k}{\delta^2}$$

Separable

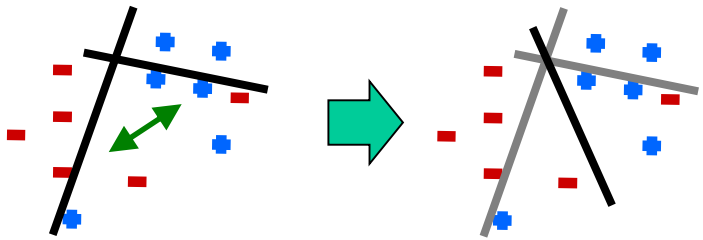


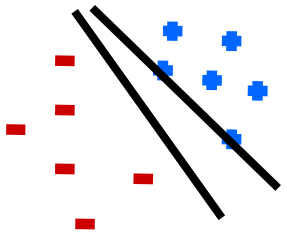
Non-Separable



Problems with the Perceptron

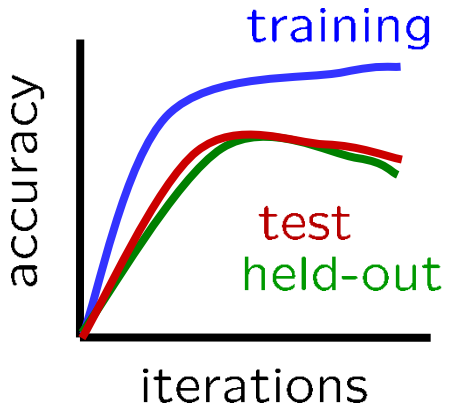
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)





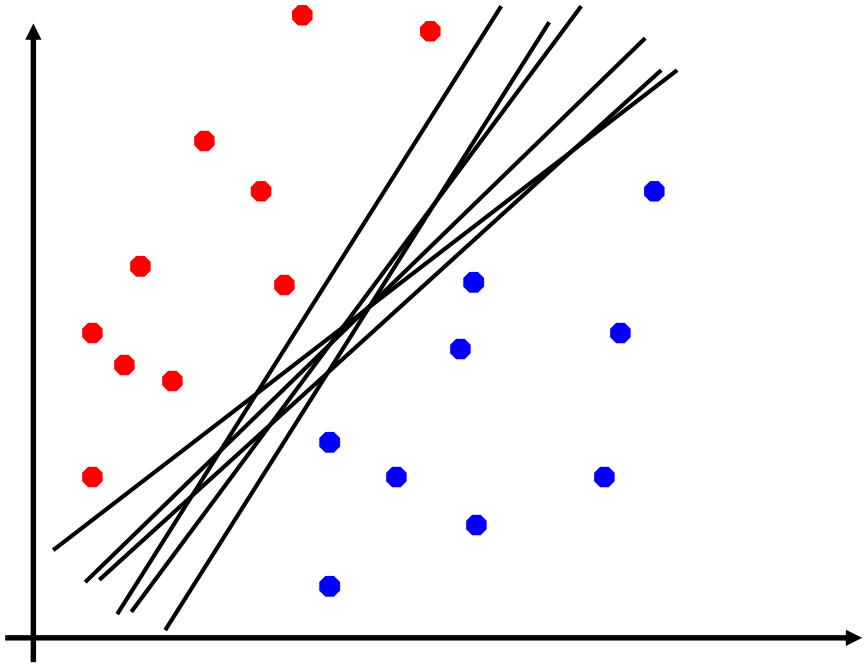
- Mediocre generalization: finds a "barely" separating solution

- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



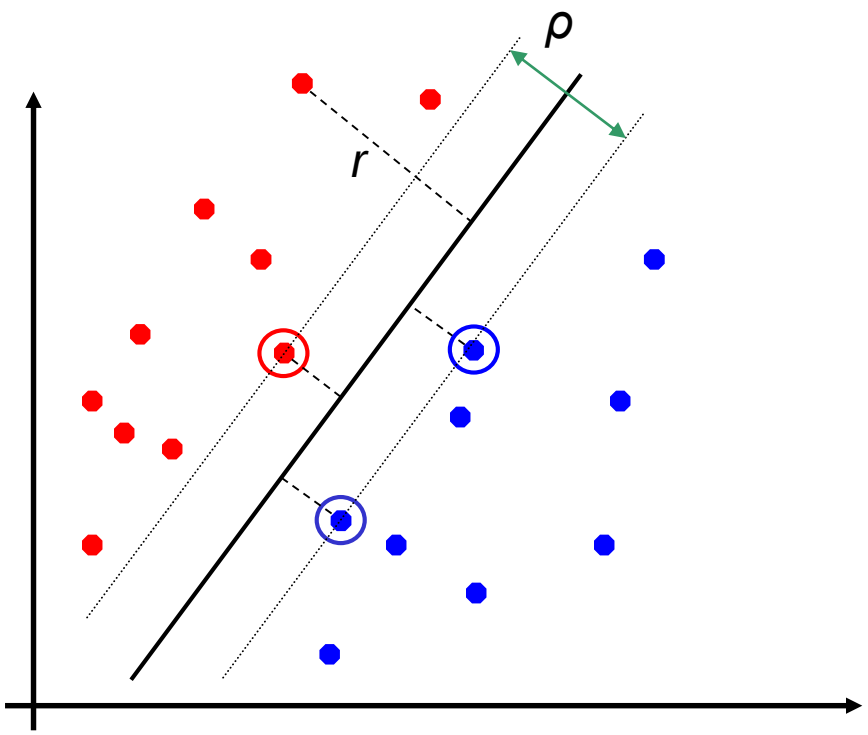
Perceptrons look for any separator

- Which of the linear separators is optimal?



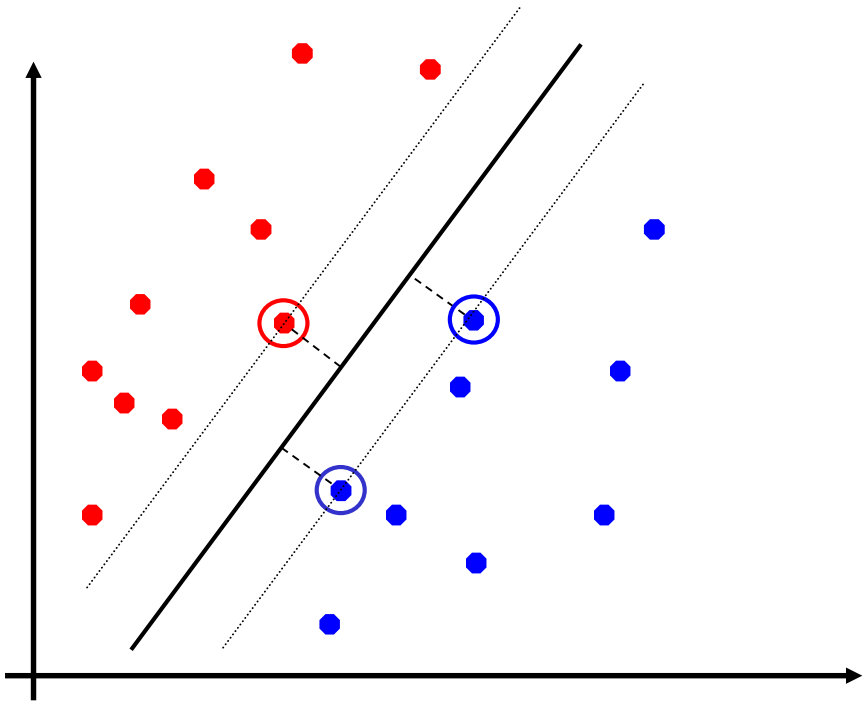
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \dfrac{\mathbf{w}^T\mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors.



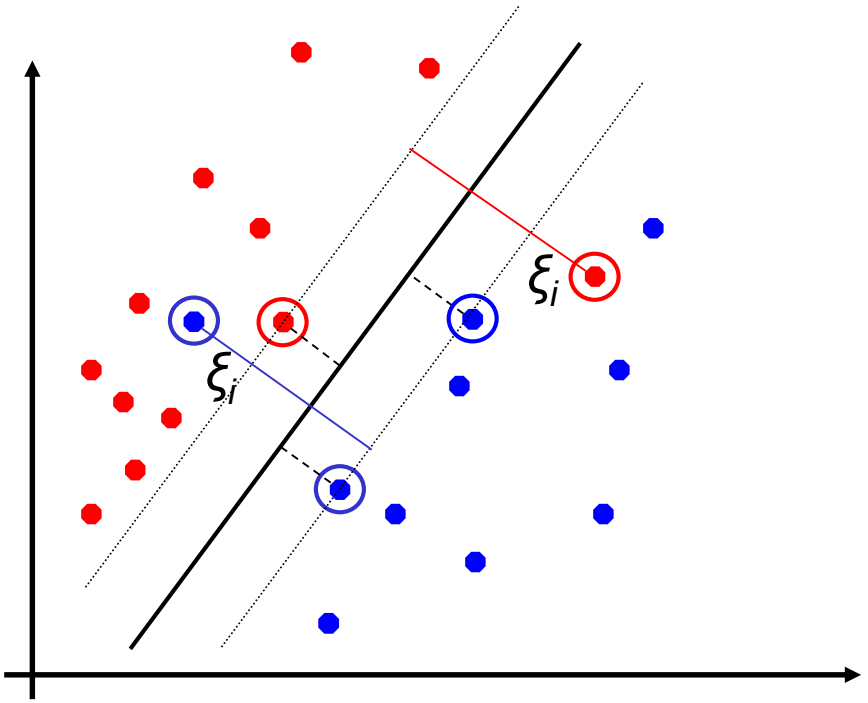
Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples are ignorable.



Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Linear SVMs: Overview

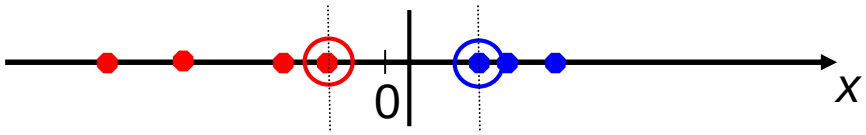
- The classifier is a *separating hyperplane.*
- Most "important" training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the dual formulation of the problem and in the solution training points appear only inside inner products:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

$$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$$

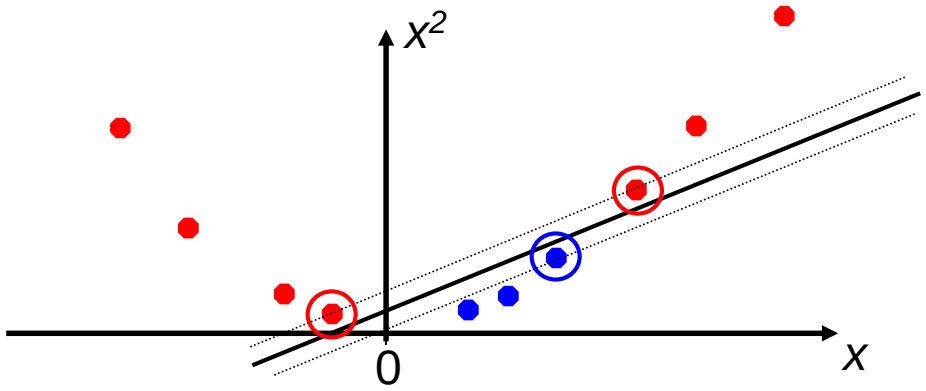
Non-linear SVMs

- Datasets that are linearly separable with some noise work out great:



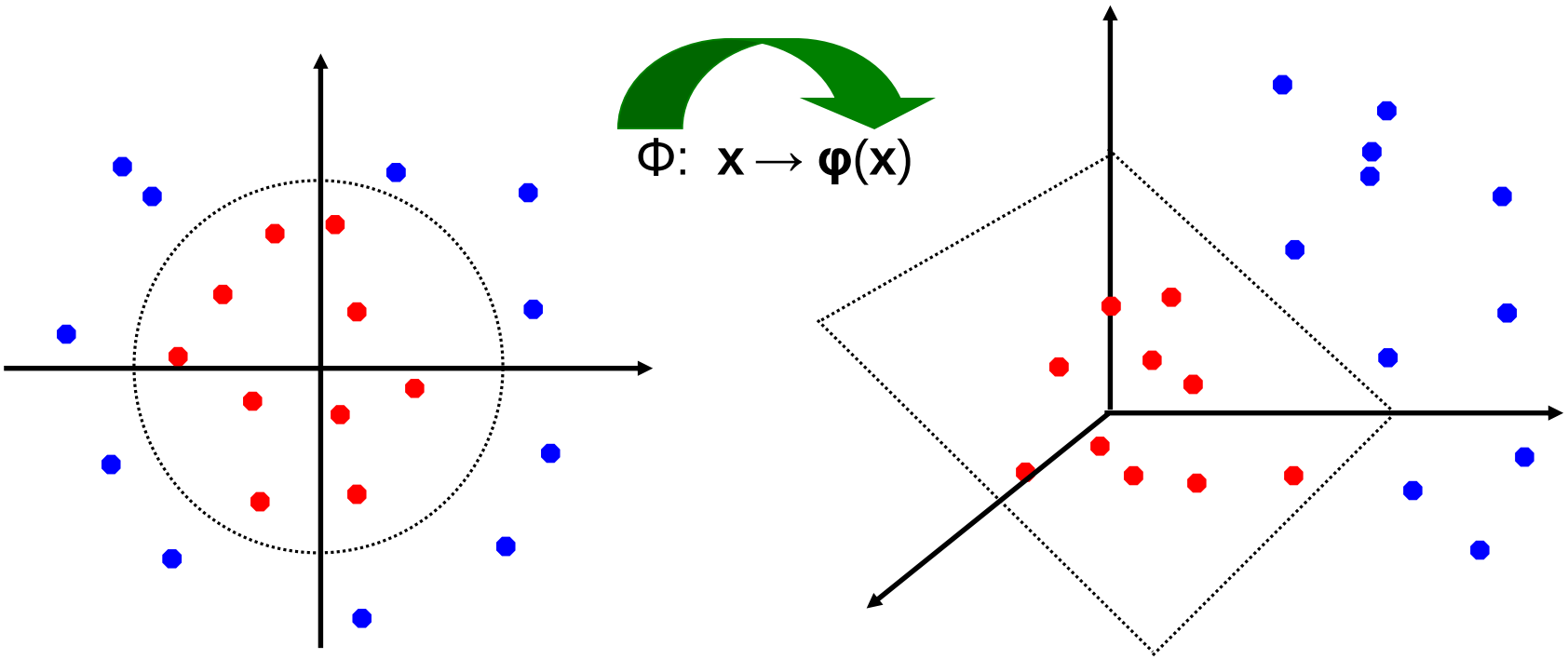
- But what are we going to do if the dataset is just too hard?

- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

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



Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i,\mathbf{x}_j)= \mathbf{x}_i^T\mathbf{x}_j$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is $\mathbf{x}$ itself

- Polynomial of power p: $K(\mathbf{x}_i,\mathbf{x}_j)= (1+ \mathbf{x}_i^T\mathbf{x}_j)^p$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ has $\binom{d+p}{p}$ dimensions

- Gaussian (radial-basis function): $K(\mathbf{x}_i,\mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2}}$
  - Mapping Φ: $\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$, where $\boldsymbol{\varphi}(\mathbf{x})$ is *infinite-dimensional*: every point is mapped to *a function* (a Gaussian); combination of functions for support vectors is the separator.

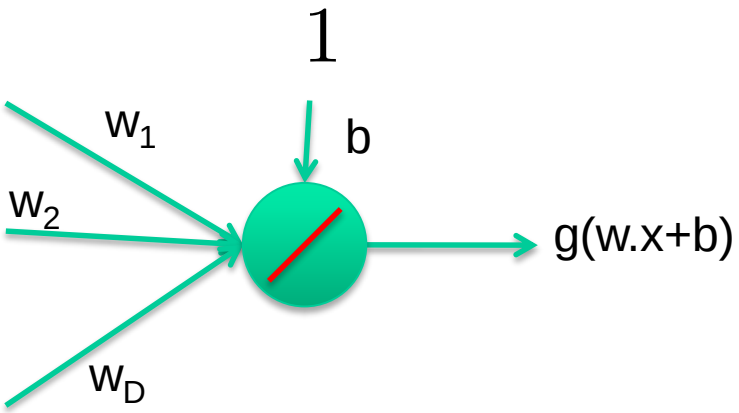
- Higher-dimensional space still has *intrinsic* dimensionality *d* (the mapping is not *onto*), but linear separators in it correspond to *non-linear* separators in original space.

Performance Comparison

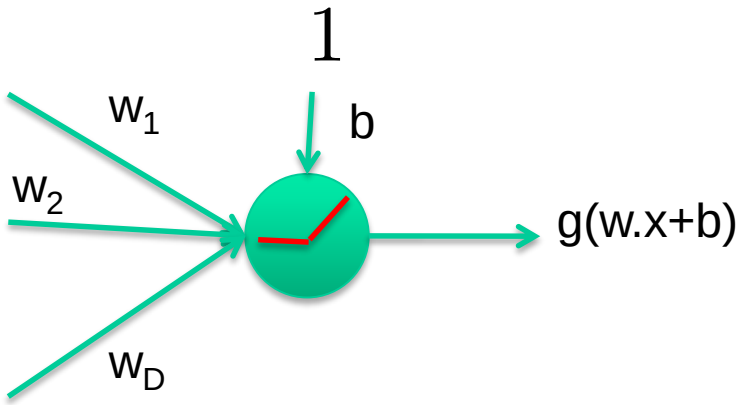
					linear SVM		rbf-SVM
	NB	Rocchio	Dec. Trees	kNN	C=0.5	C=1	
earn	96.0	96.1	96.1	97.8	98.0	**98.2**	98.1
acq	90.7	92.1	85.3	91.8	95.5	**95.6**	94.7
money-fx	59.6	67.6	69.4	75.4	**78.8**	78.5	74.3
grain	69.8	79.5	89.1	82.6	91.9	93.1	**93.4**
crude	81.2	81.5	75.5	85.8	89.4	**89.4**	88.7
trade	52.2	77.4	59.2	77.9	79.2	**79.2**	76.6
interest	57.6	72.5	49.1	**76.7**	75.6	74.8	69.1
ship	80.9	83.1	80.9	79.8	**87.4**	86.5	85.8
wheat	63.4	79.4	85.5	72.9	86.6	**86.8**	82.4
corn	45.2	62.2	87.7	71.4	87.5	**87.8**	84.6
microavg.	72.3	79.9	79.4	82.6	86.7	**87.5**	86.4

SVM classifier break-even F from (Joachims, 2002a, p. 114). Results are shown for the 10 largest categories and for microaveraged performance over all 90 categories on the Reuters-21578 data set.

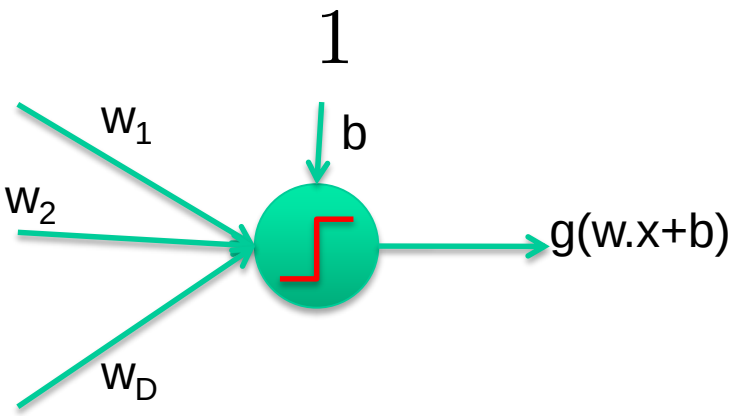
Neurons: Activation Functions



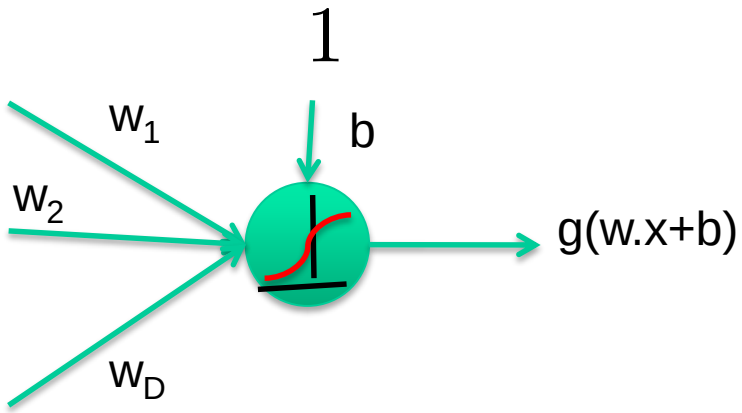
Linear Neuron



Rectified Linear Unit



Perceptron



Logistic Neuron

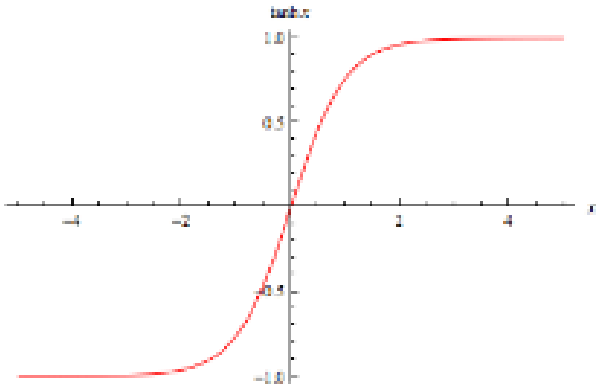
Activations Popular in NLP

- $\text{softmax} : \mathbb{R}^k \to \mathbb{R}^k$

$$\langle x_1, x_2, \ldots, x_k \rangle \mapsto \left\langle \frac{e^{x_1}}{\sum_{j=1}^{k} e^{x_j}}, \frac{e^{x_2}}{\sum_{j=1}^{k} e^{x_j}}, \ldots, \frac{e^{x_k}}{\sum_{j=1}^{k} e^{x_j}} \right\rangle$$

- $\tanh : \mathbb{R} \to [-1, 1]$

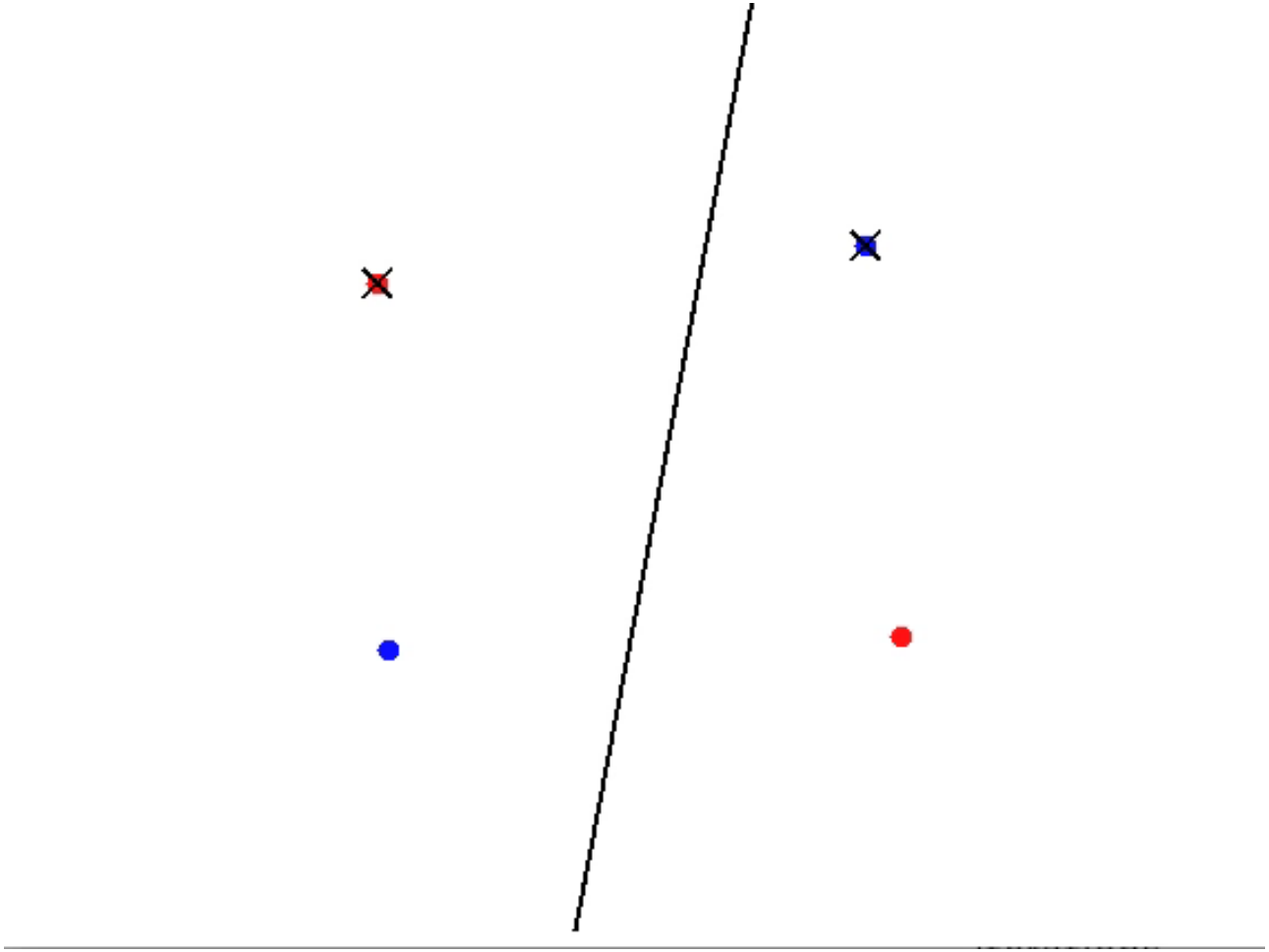
$$x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Generalized to be *elementwise*, so that it maps $\mathbb{R}^k \to [-1, 1]^k$.

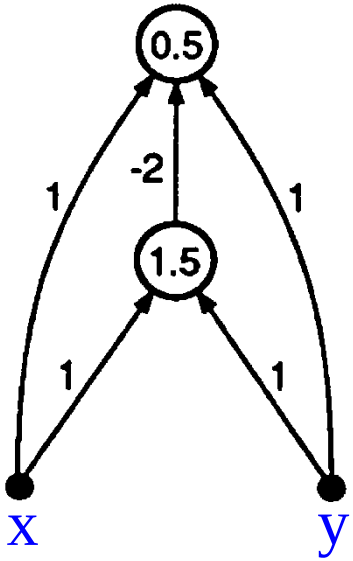
XOR using Perceptron

- The inseparable case



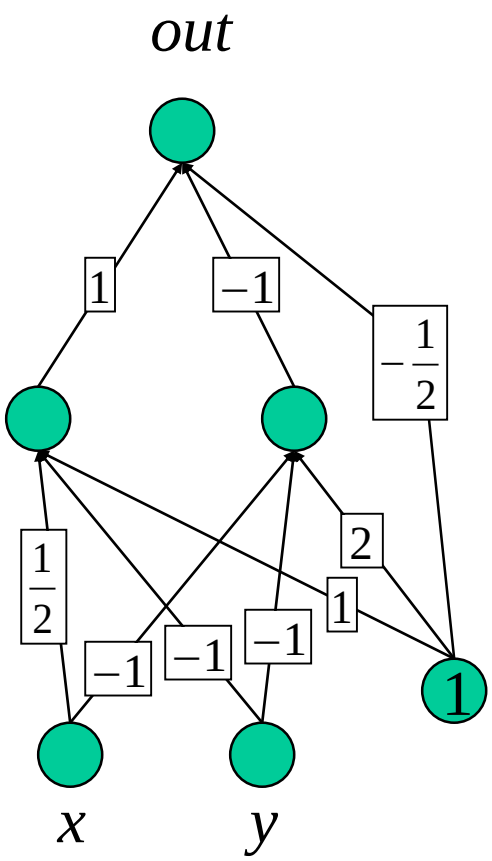
Idea 1: Multilayer Perceptrons

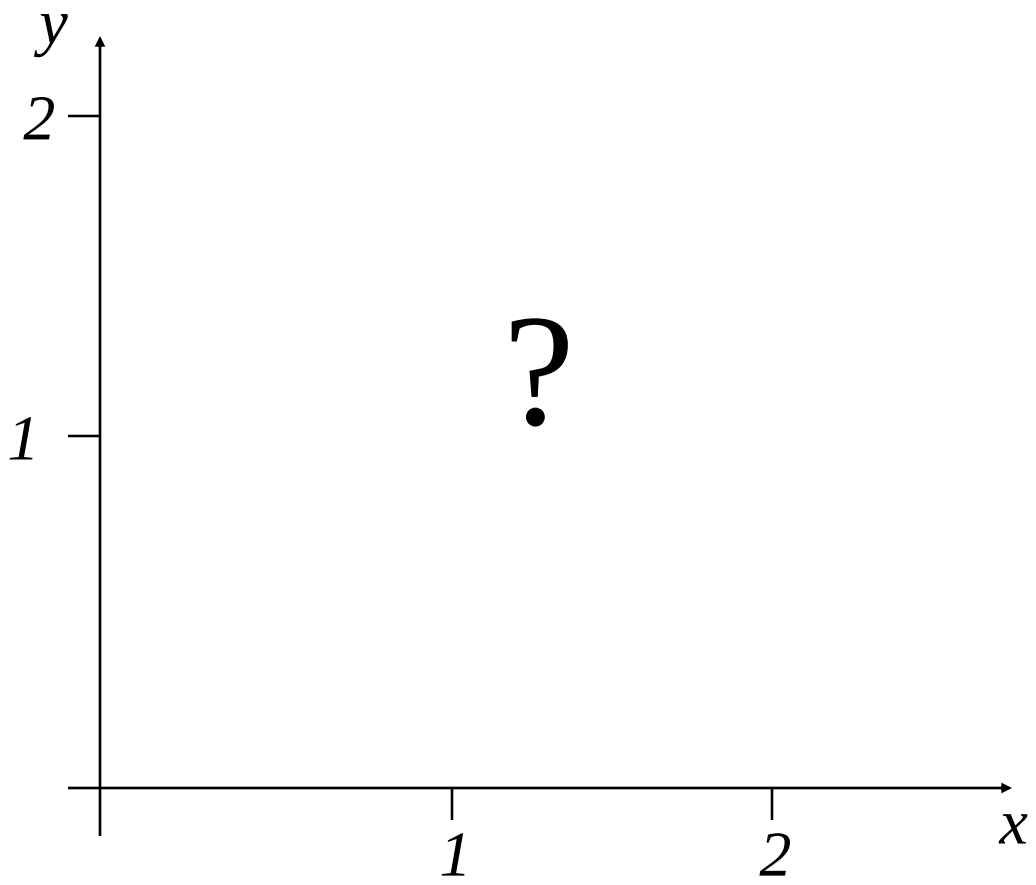
- Removes limitations of single-layer networks
  - Can solve XOR
- Example: Two-layer perceptron that computes XOR



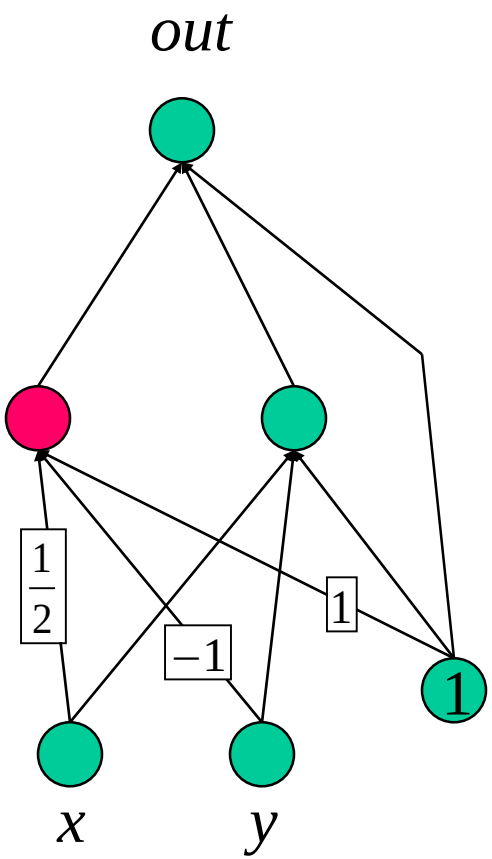
- Output is +1 if and only if $x + y - 2\Theta(x + y - 1.5) - 0.5 > 0$

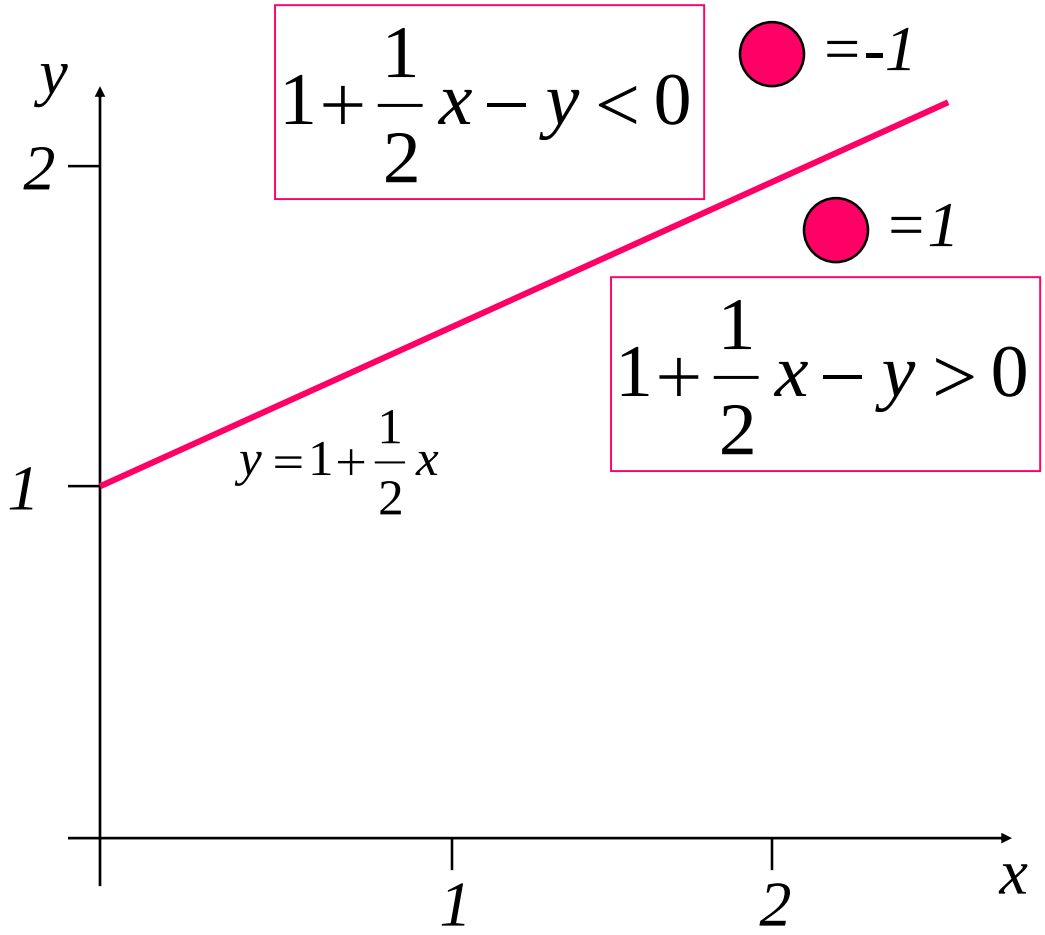
Multilayer Perceptron: What does it do?



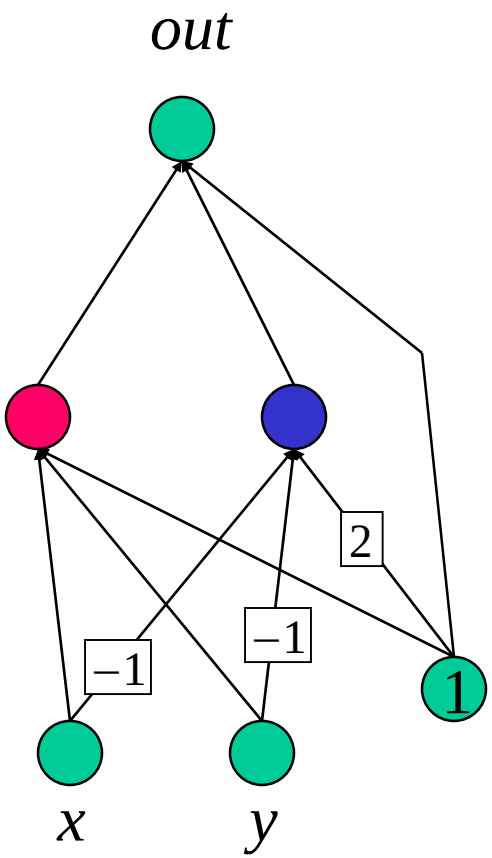


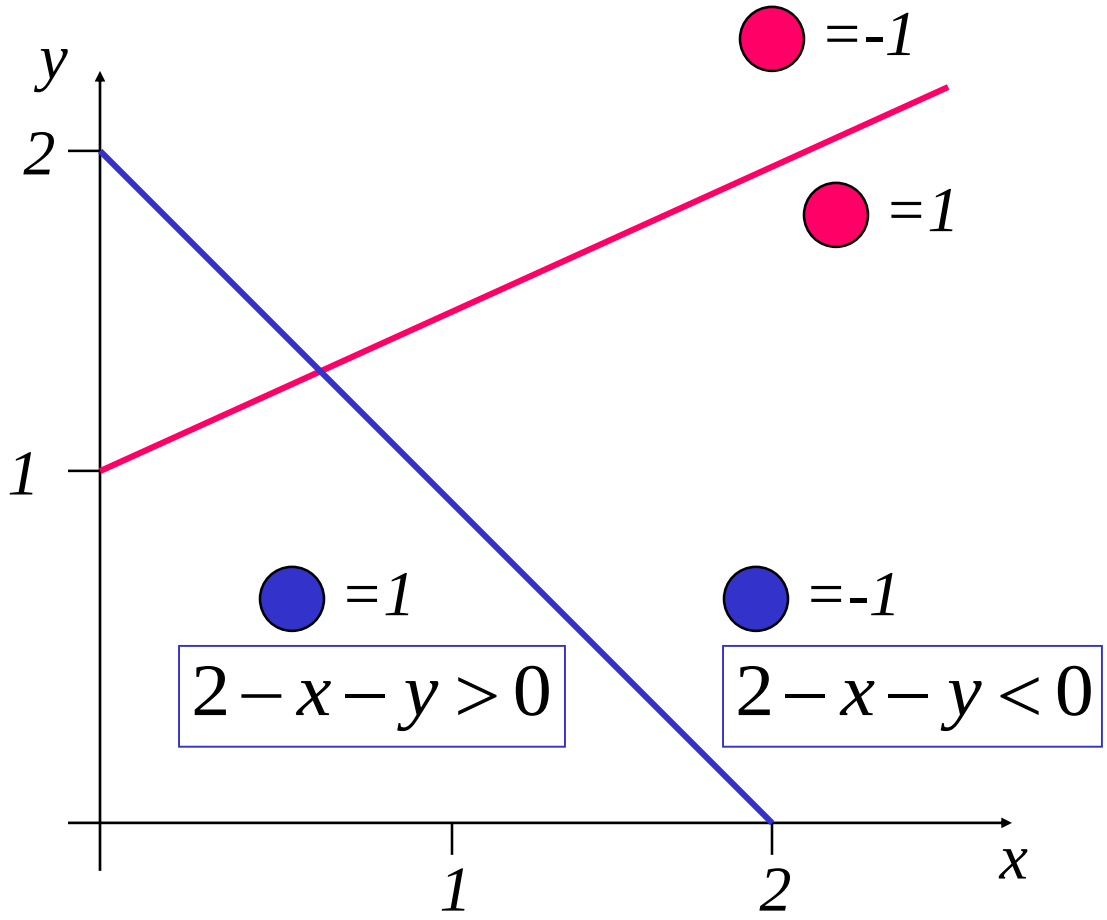
Multilayer Perceptron: What does it do?



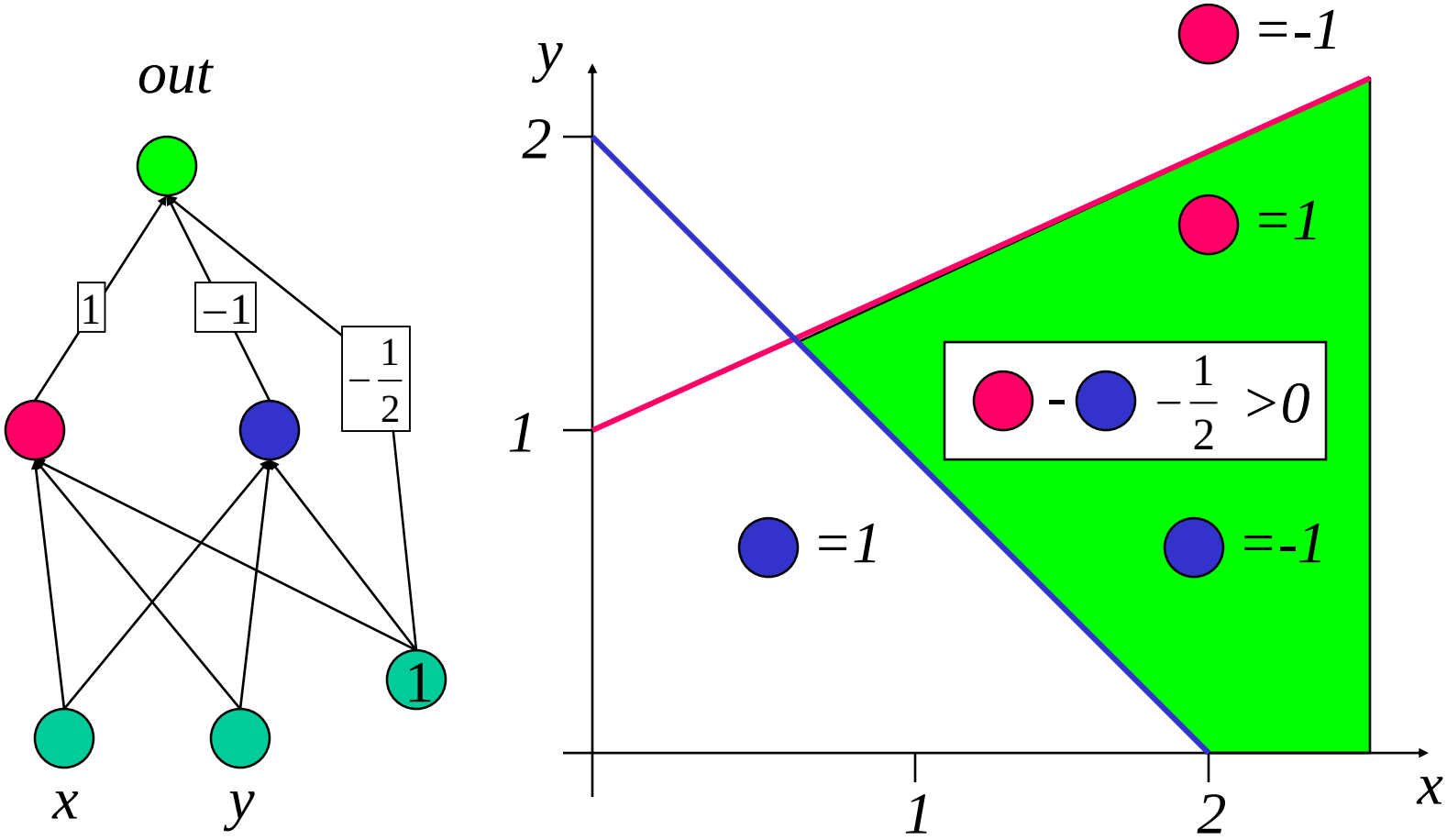


Multilayer Perceptron: What does it do?



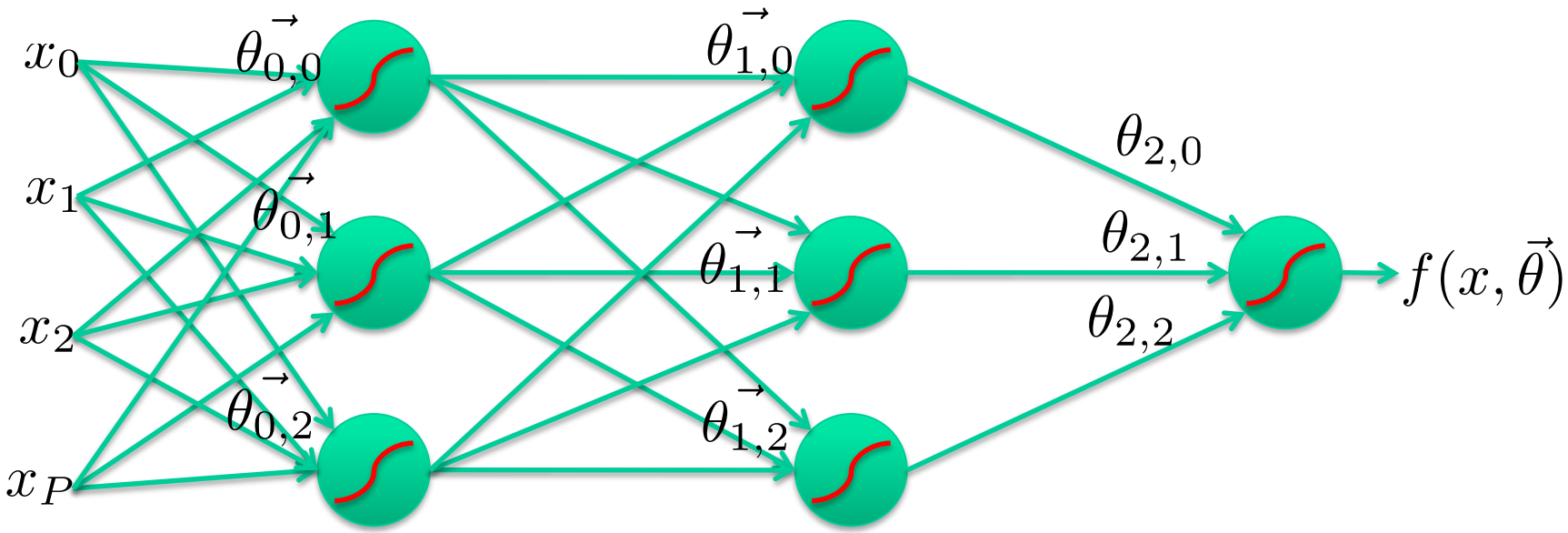


Multilayer Perceptron: What does it do?



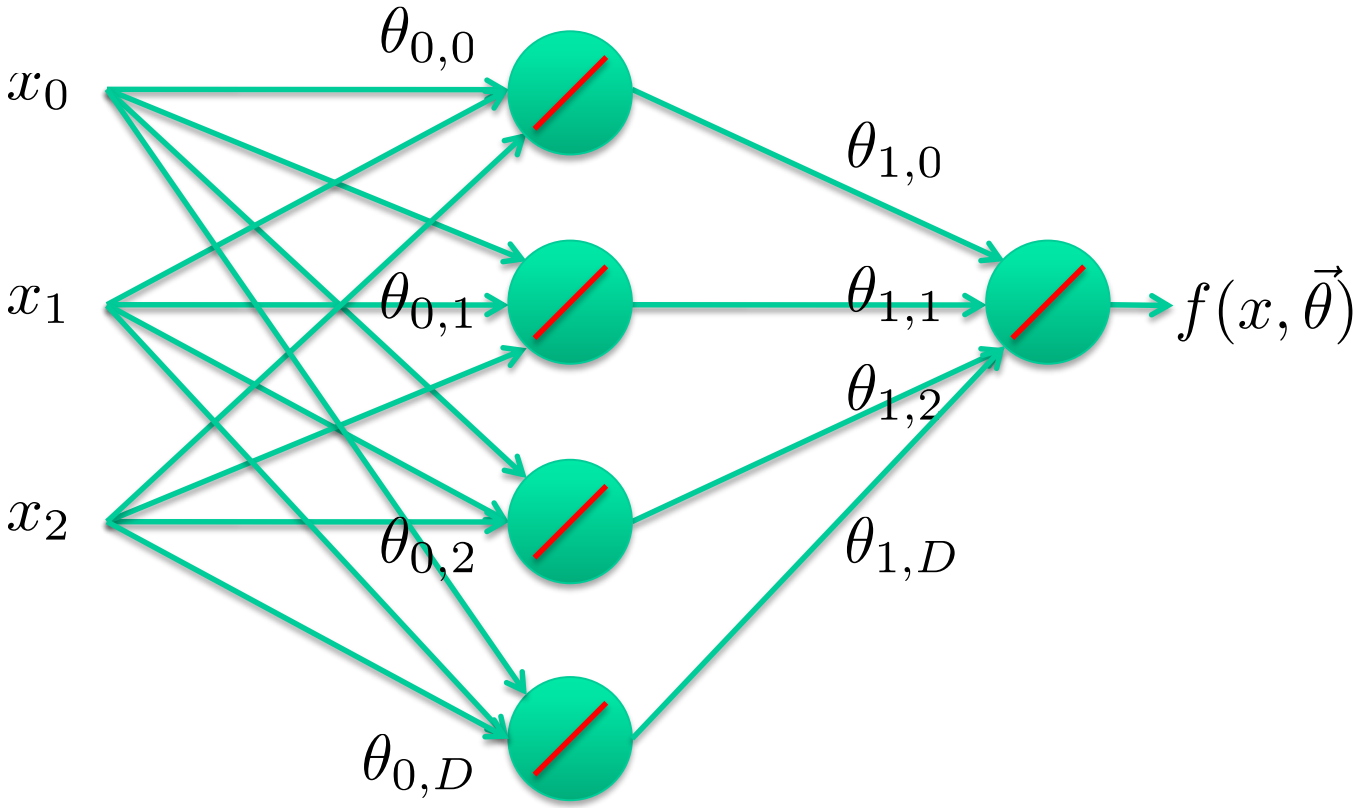
Multilayer Networks

- Cascade Neurons together
- The output from one layer is the input to the next
- Each Layer has its own sets of weights



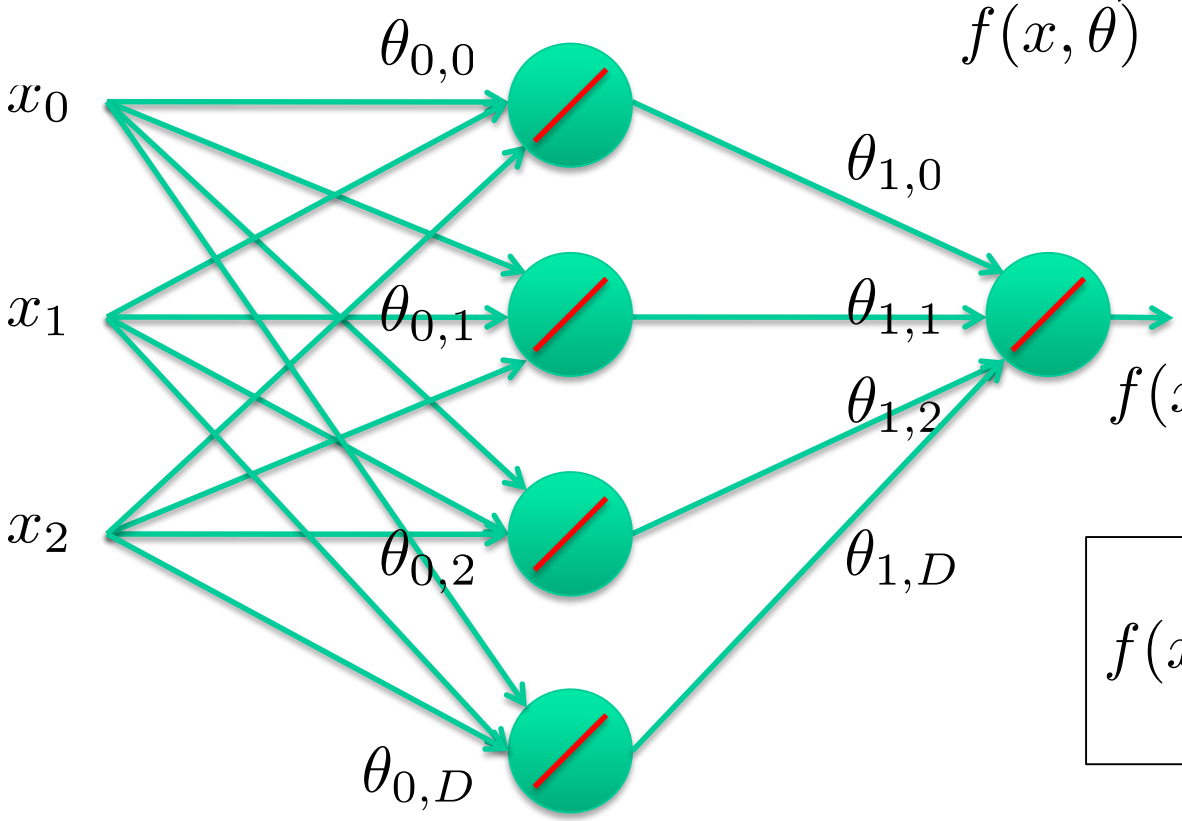
Linear Regression Neural Networks

- What happens when we arrange **linear neurons** in a multilayer network?



Linear Regression Neural Networks

- Nothing special happens.
  - The product of two linear transformations is itself a linear transformation.



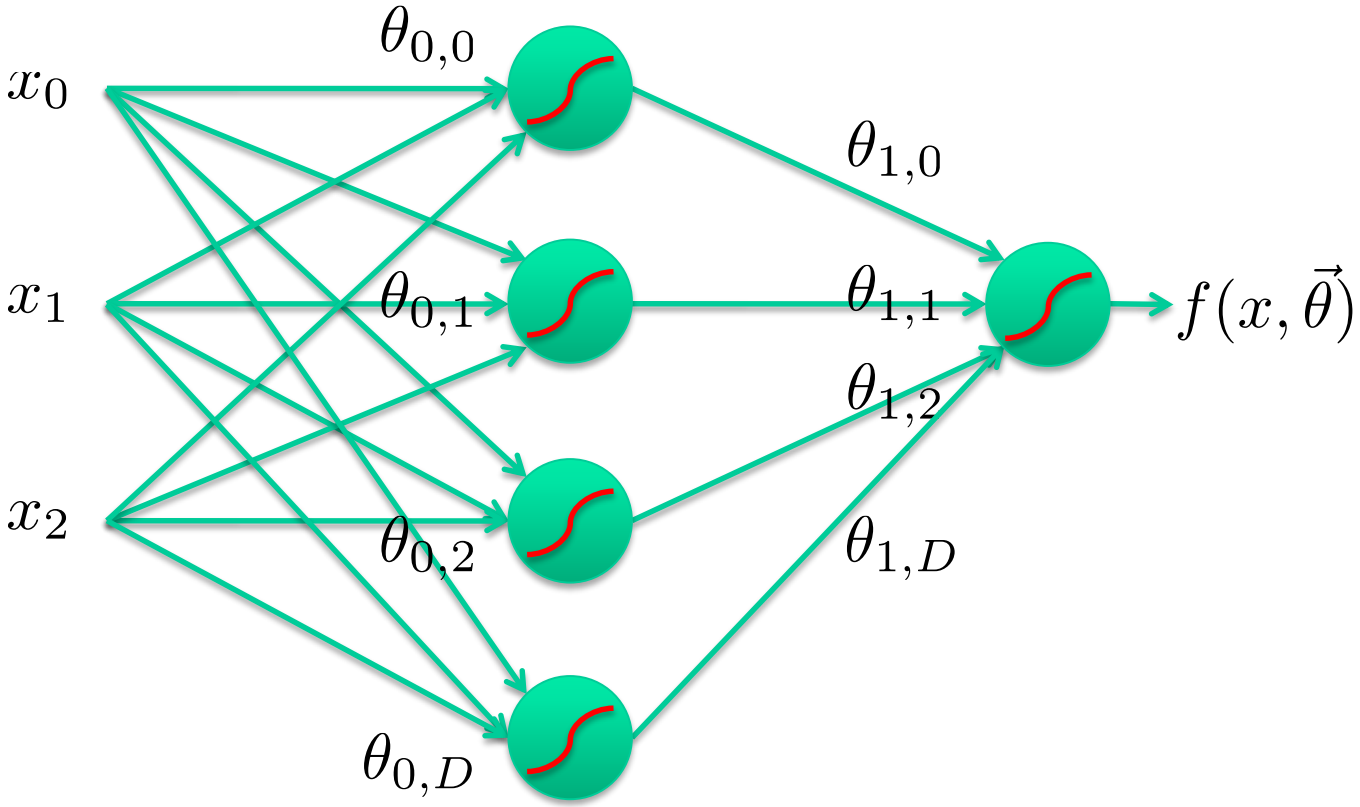
$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} \sum_{n=0}^{N-1} \theta_{0,i,n} x_n$$

$$f(x,\vec{\theta}) = \sum_{i=0}^{D} \theta_{1,i} [\theta_{0,i}^T \vec{x}]$$

$$\boxed{f(x,\vec{\theta}) = \sum_{i=0}^{D} [\hat{\theta}_i^T \vec{x}]}$$

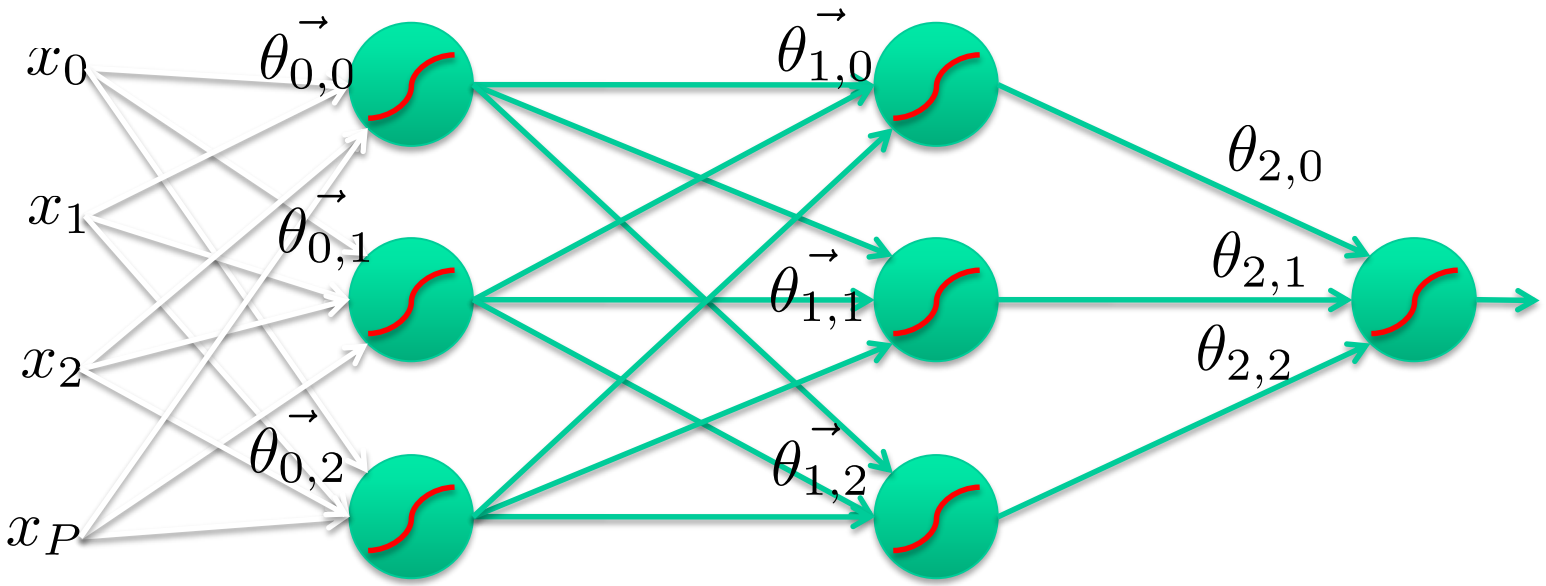
Neural Networks

- We want to introduce non-linearities to the network.
  - Non-linearities allow a network to identify complex regions in space



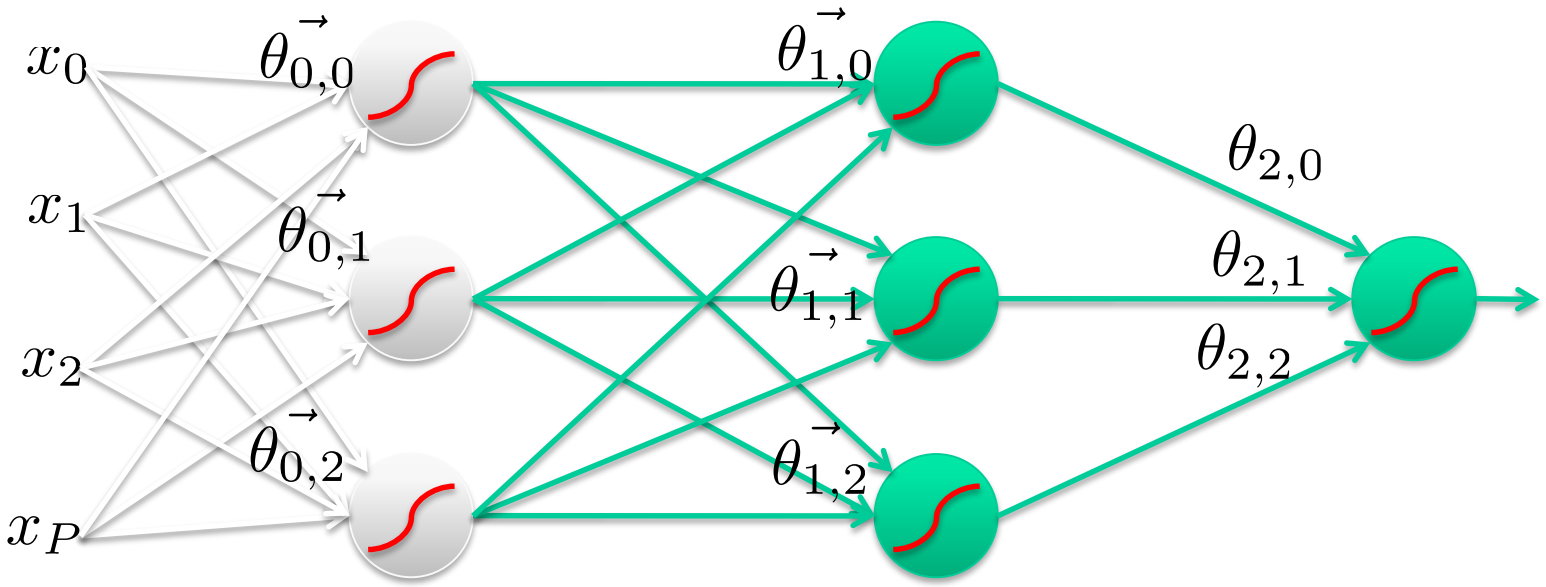
Feed-Forward Networks

- Predictions are fed forward through the network to classify



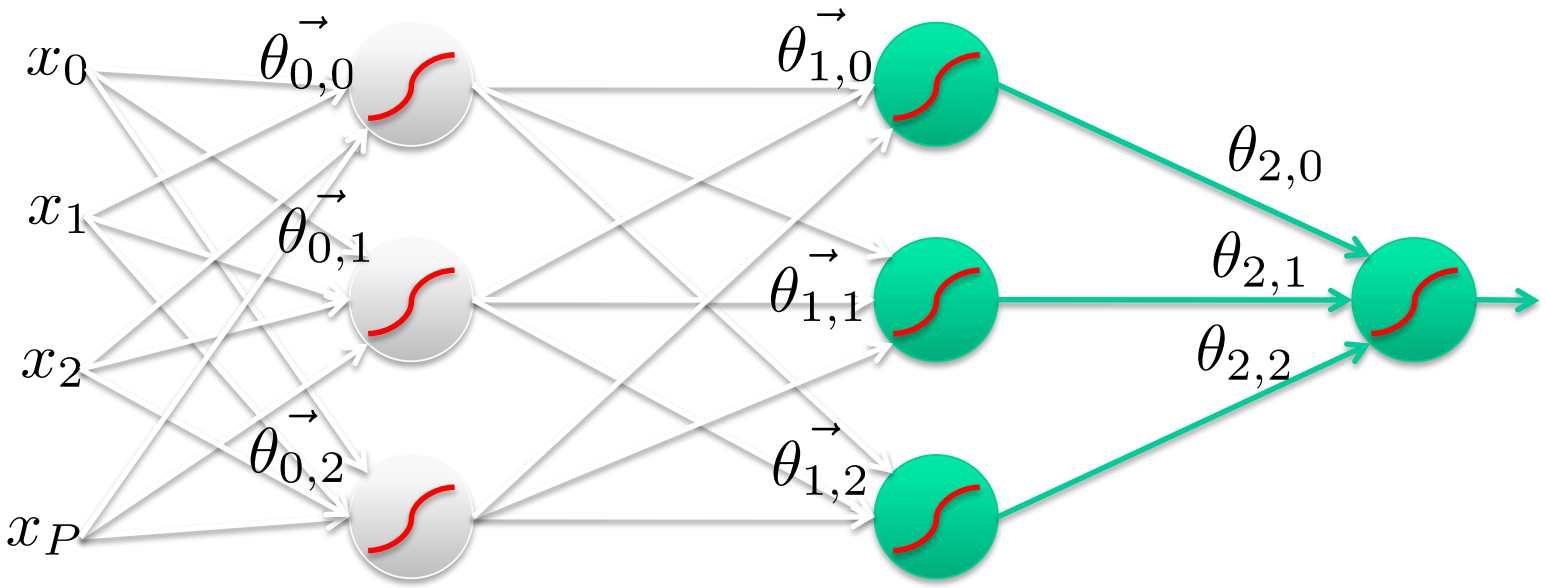
Feed-Forward Networks

- Predictions are fed forward through the network to classify



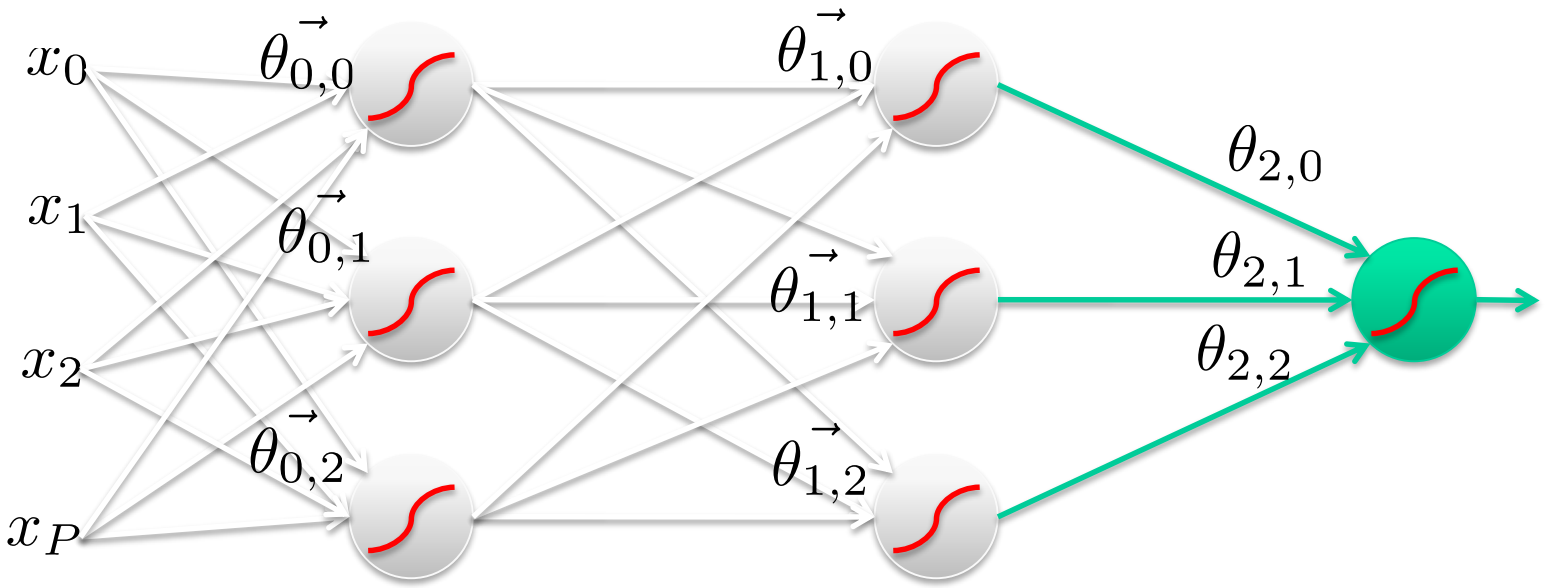
Feed-Forward Networks

- Predictions are fed forward through the network to classify



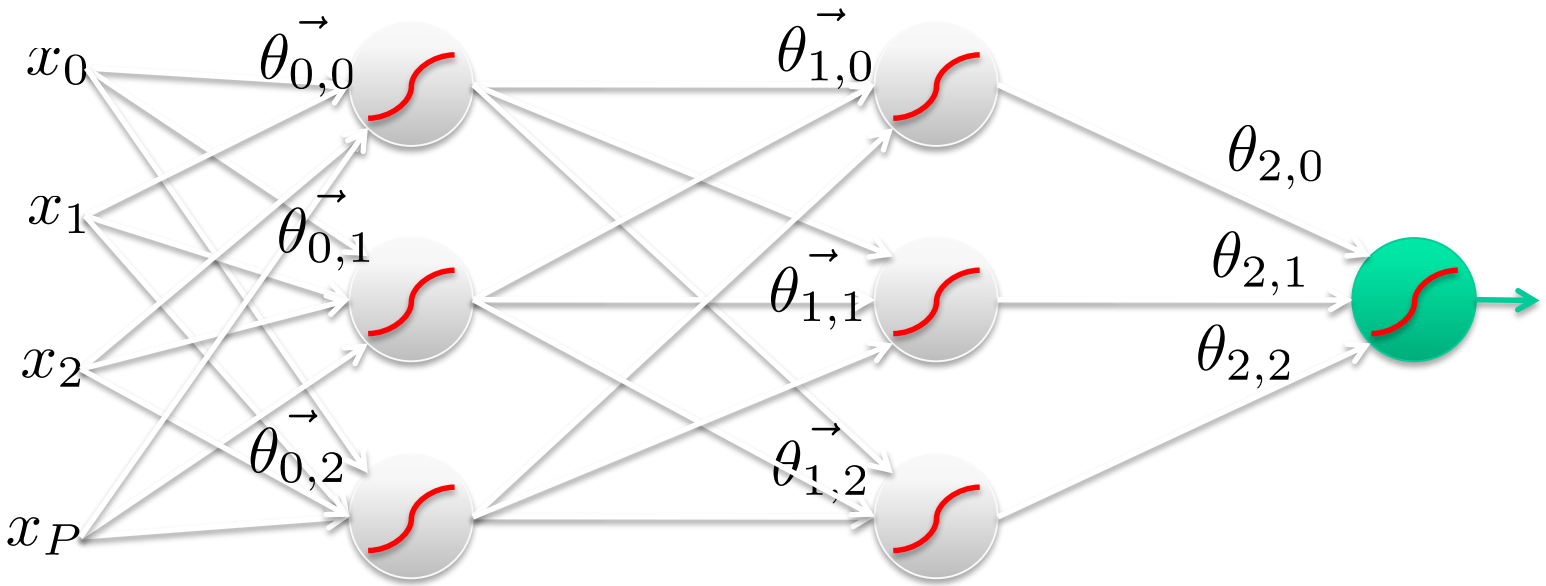
Feed-Forward Networks

- Predictions are fed forward through the network to classify



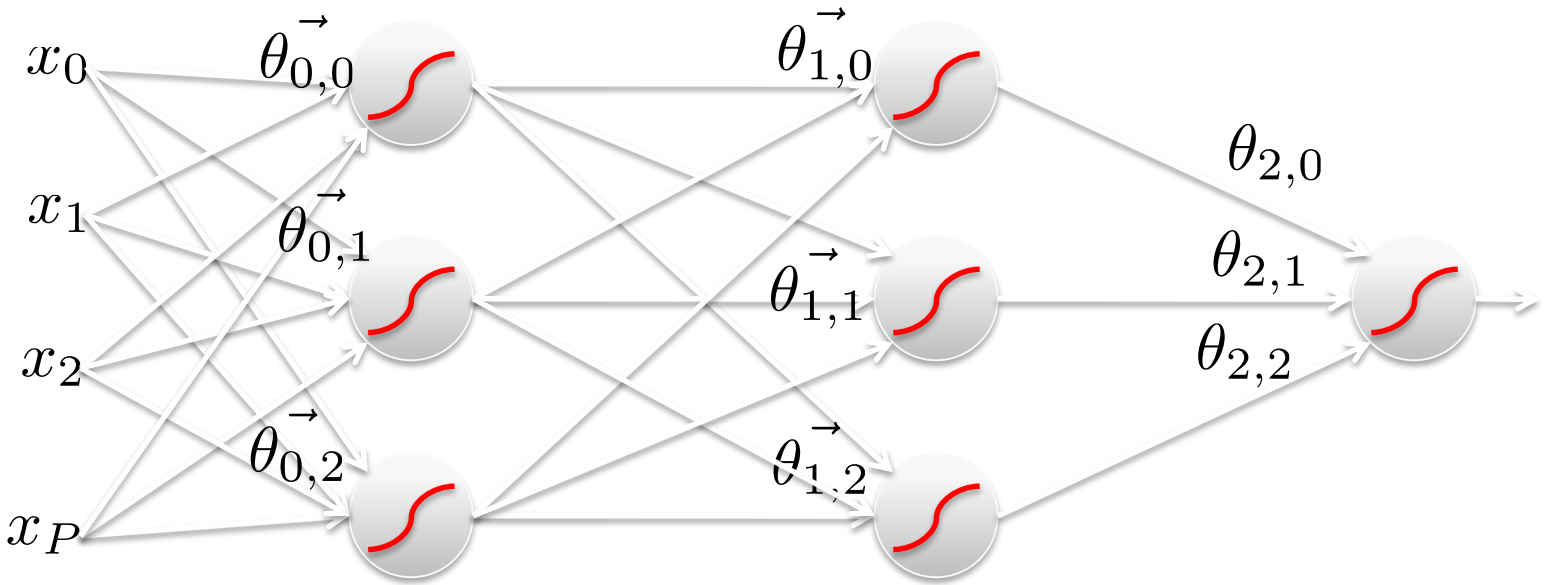
Feed-Forward Networks

- Predictions are fed forward through the network to classify



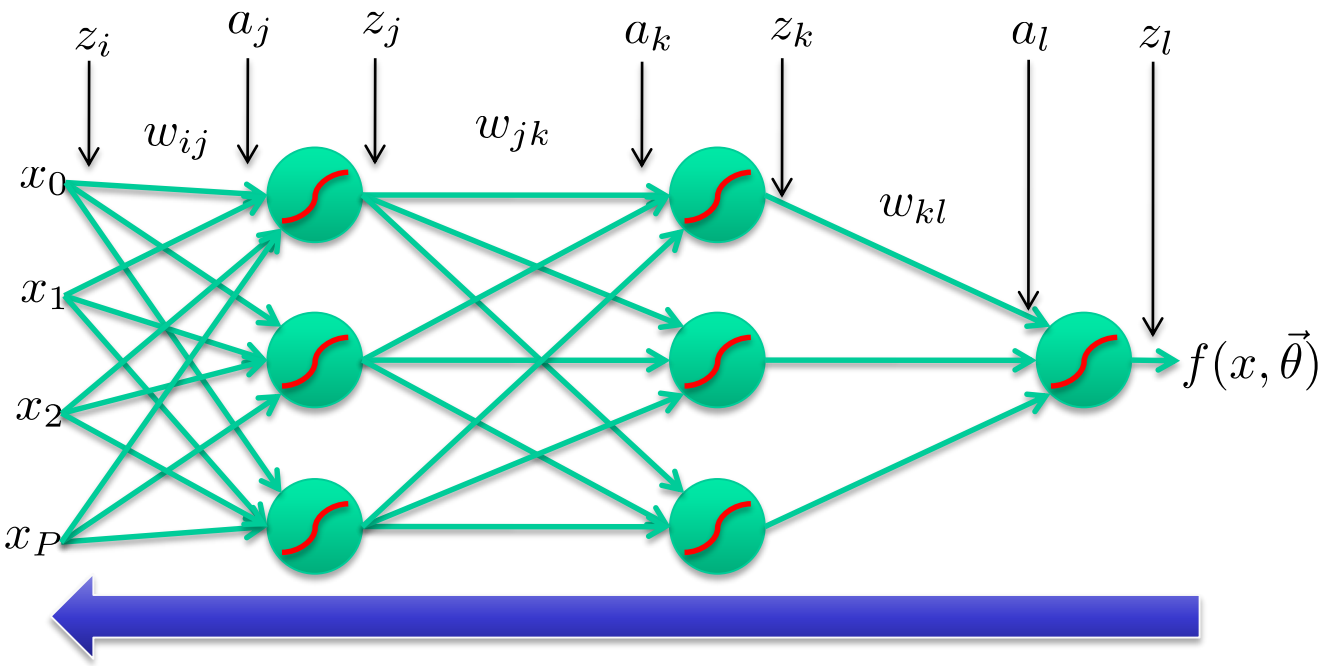
Feed-Forward Networks

- Predictions are fed forward through the network to classify

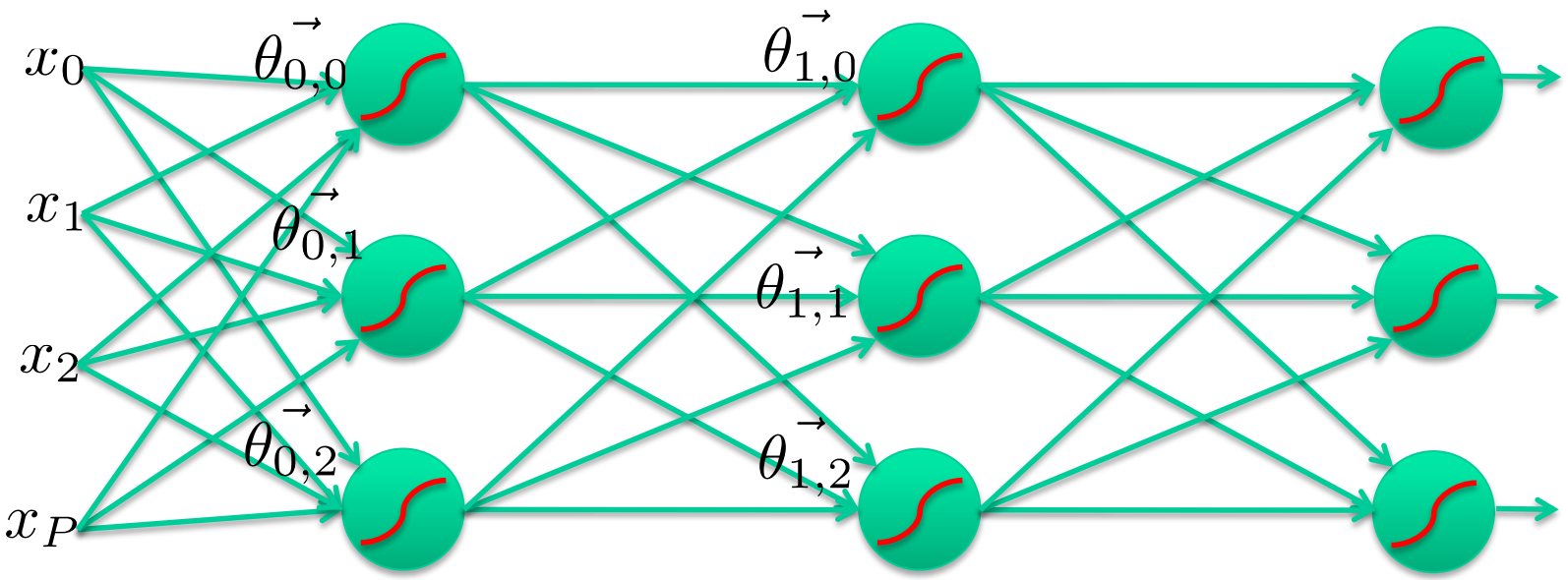


Error Back-propagation

- Error backprop unravels the multivariate chain rule and solves the gradient for each partial component separately.
- The target values for each layer come from the next layer.
- This feeds the errors back along the network.



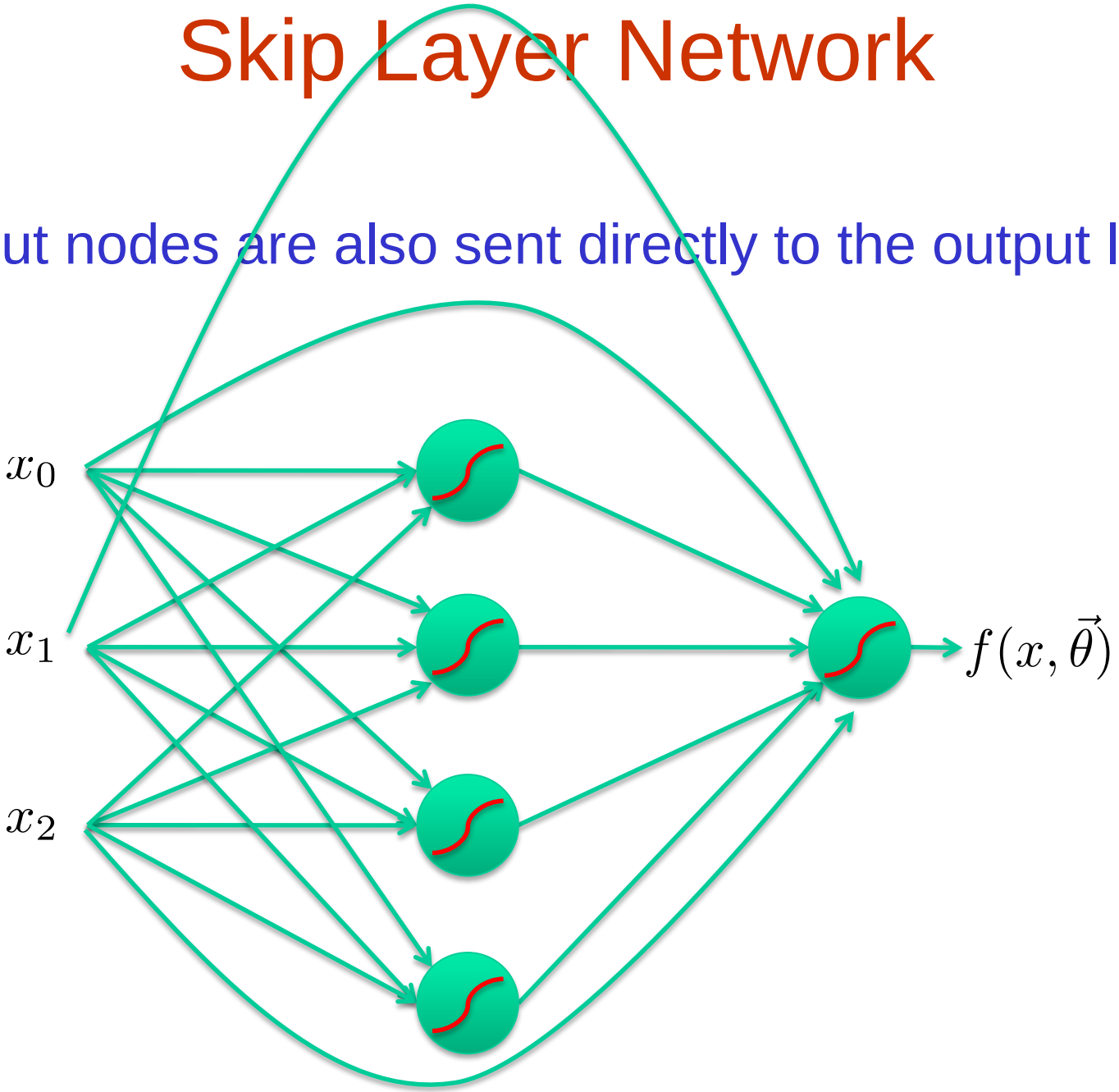
Multiple Outputs



- Used for N-way classification.
- Each Node in the output layer corresponds to a different class
- No guarantee that the sum of the output vector will equal 1.

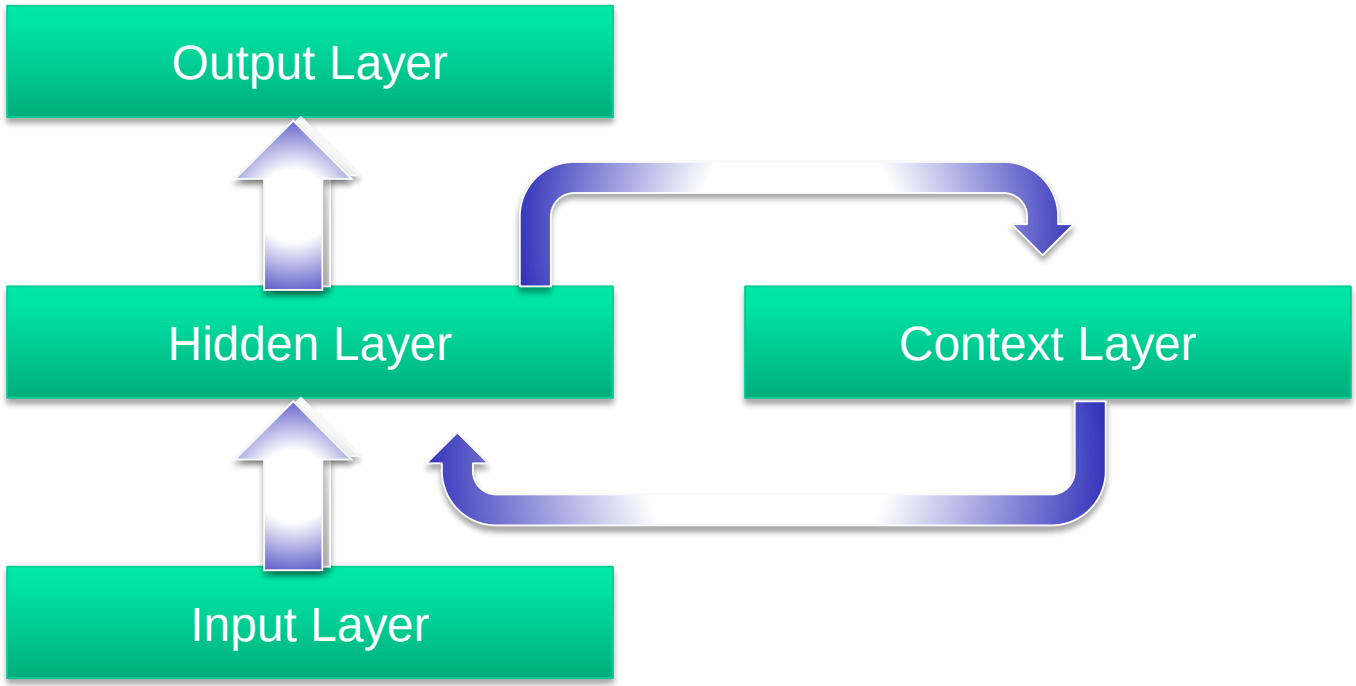
Skip Layer Network

- Input nodes are also sent directly to the output layer.



Recurrent Neural Networks

- Output or hidden layer information is stored in a context or memory layer.



Neural Networks

- Have significant expressiveness
- Bounded VC dim
  - some believe that this makes them superior to SVMs
- Ability to provide background knowledge as a network

- Uninterpretable
  - Very hard to debug
- Overfitting
  - Dropout, Pooling layers, etc
- Hard to deal with words explicitly
  - word vector representations

Choosing a classifier

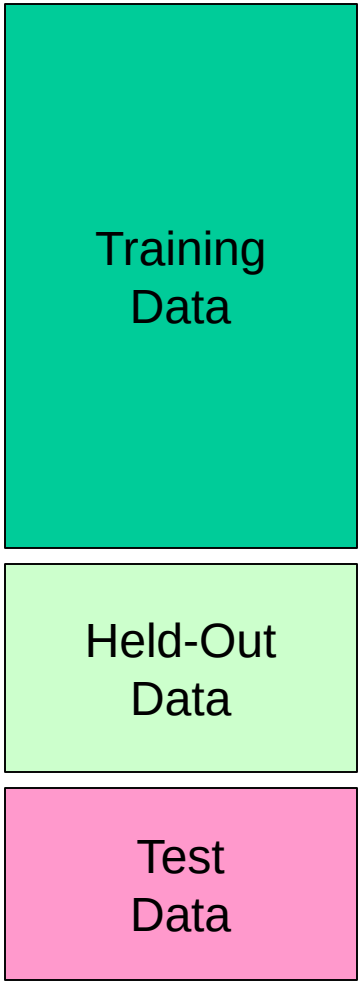
Technique	Train time	Test time	"Accuracy"	Interpre-tability	Bias-Variance	Data Complexity
Naïve Bayes	$\|W\| + \|C\| \|V\|$	$\|C\| * \|V_d\|$	Medium-low	Medium	High-bias	Low
k-NN	$\|W\|$	$\|V\| * \|V_d\|$	Medium	Low	High-variance	High
SVM	$\|C\|\|D\|^3 \|V\|_{ave}$	$\|C\|* \|V_d\|$	High	Low	Mixed	Medium-low
Neural Nets	?	$\|C\|* \|V_d\|$	High	Low	High-variance	High
Log-linear	?	$\|C\|* \|V_d\|$	High	Medium	High-variance/ mixed	Medium

"Accuracy" – reputation for accuracy in experimental settings.

Note that it is impossible to say beforehand which classifier will be most accurate on any given problem.

C = set of classes. W = bag of training tokens. V = set of training types. D = set of train docs. V_d = types in test document d. V_{ave} = average number of types per doc in training.

Summary: Three Views of Classification



- Joint prob. Models (Naïve Bayes):
  - Parameters from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data
- Conditional prob. Models (Log-linear):
  - Parameters from gradient ascent
  - Parameters: linear, probabilistic model, and discriminative
  - Training: gradient ascent (usually batch), regularize to stop overfitting
- Discriminative non-prob. models:
  - Perceptrons: Parameters from reactions to mistakes; linear
  - SVMs: parameters from gradient ascent; can be any shape depending on kernel; regularize to stop overfitting

All Methods

- Define a loss function
  - - log-likelihood
  - squared loss
  - hinge loss
  - - #errors
  - regularization terms
- Take derivative wrt parameters
- Perform gradient updates