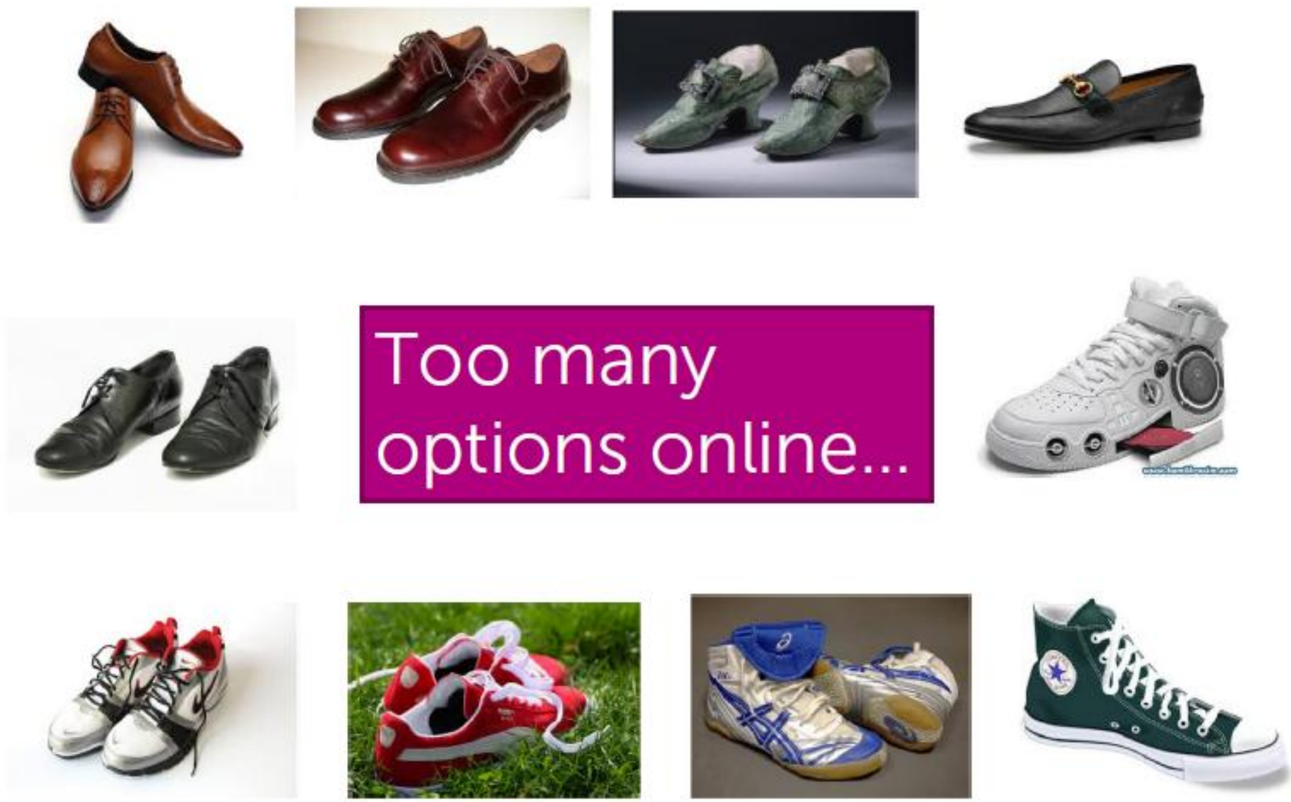
INTRODUCTION TO DATA SCIENCE

This lecture is
based on course by E. Fox and C. Guestrin, Univ of Washington

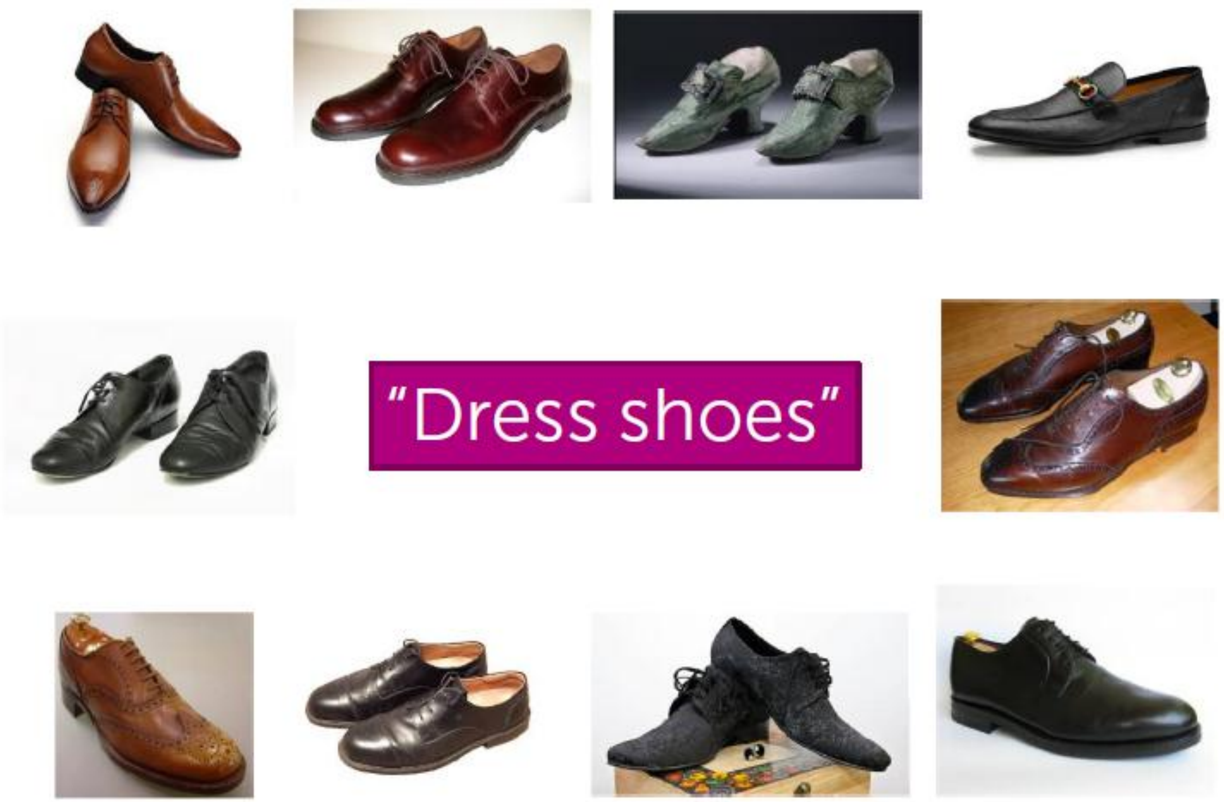
Visual product recomender

I want to buy new shoes, but...



Visual product recomender

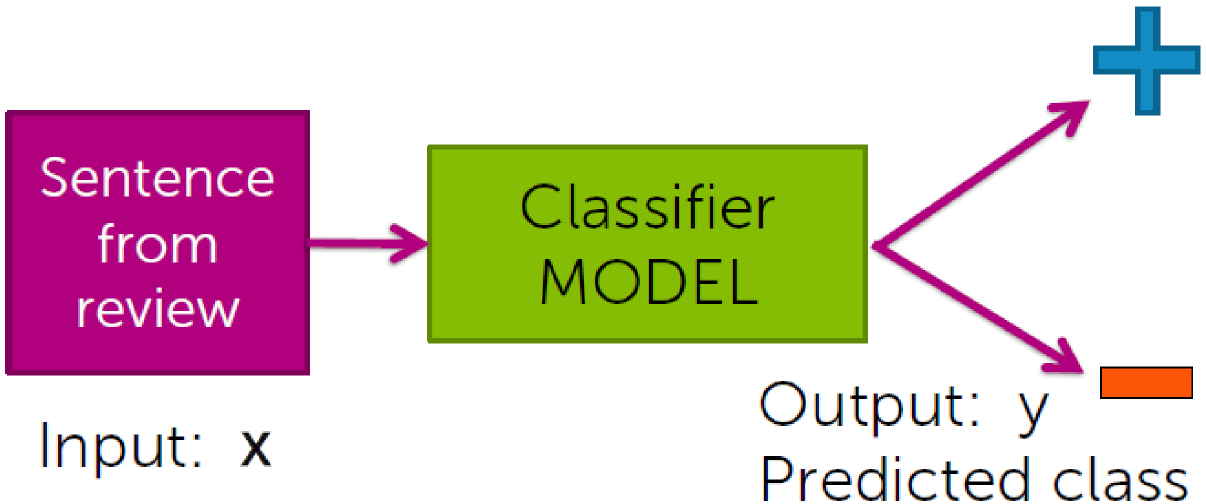
Text search doesn't help...



Visual product recomender

Features are key to machine learnng

Goal: revisit classifiers, but using more complex, non-linear features



Features are key to machine learning

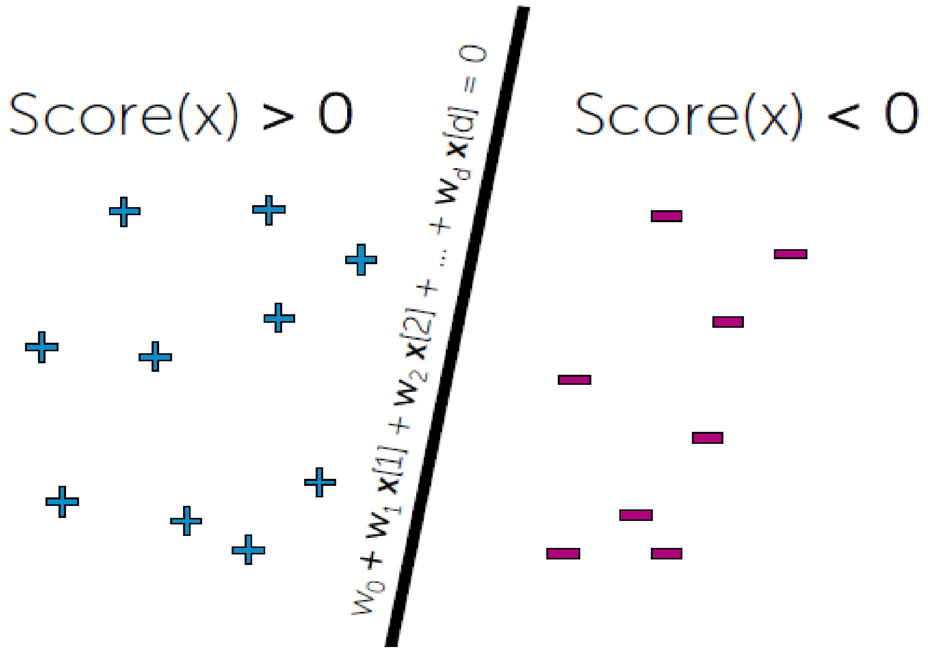
Neural networks

↓

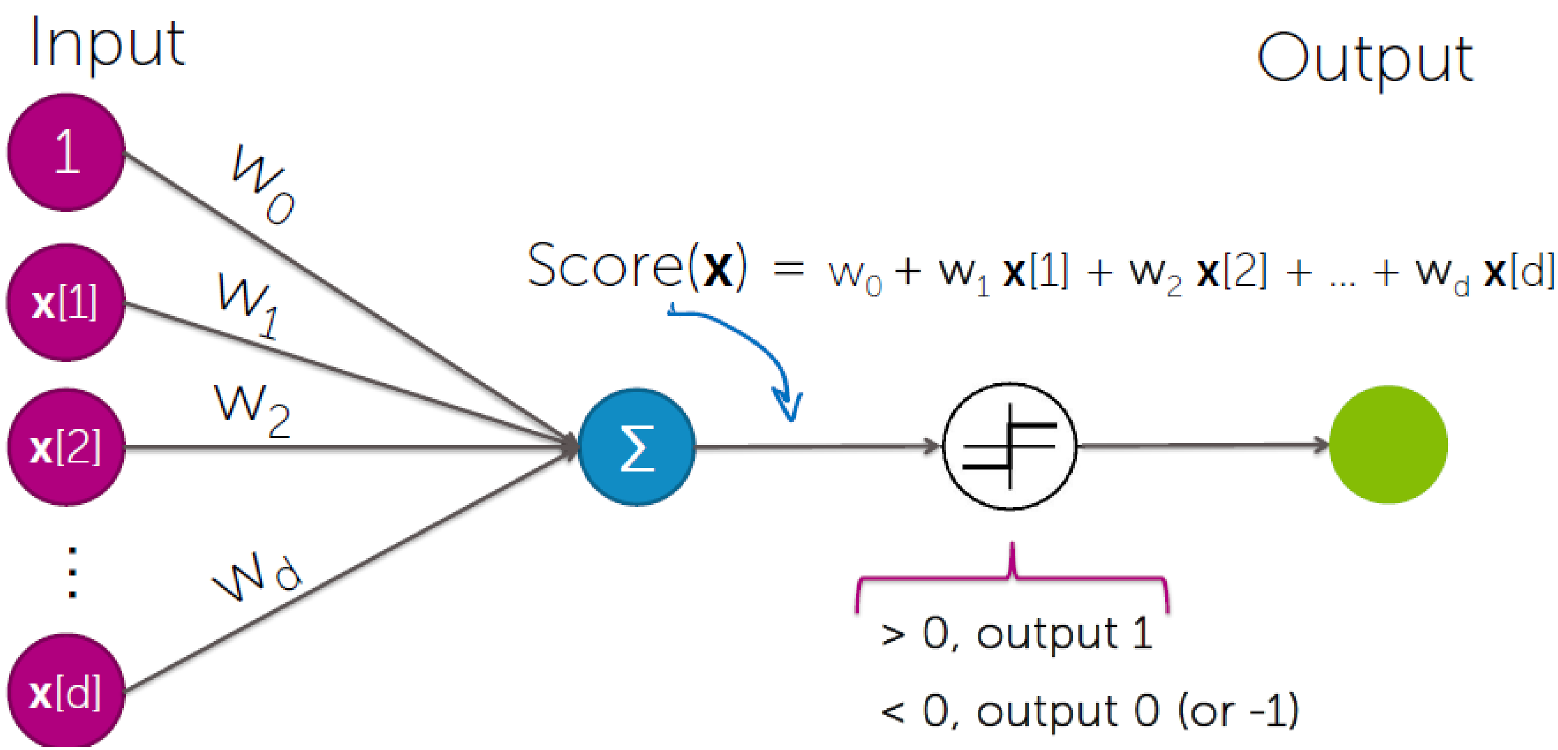
Learning *very* non-linear features

Recall: Linear classifiers

$$\text{Score}(x) = w_0 + w_1\,\mathbf{x}[1] + w_2\,\mathbf{x}[2] + \ldots + w_d\,\mathbf{x}[d]$$

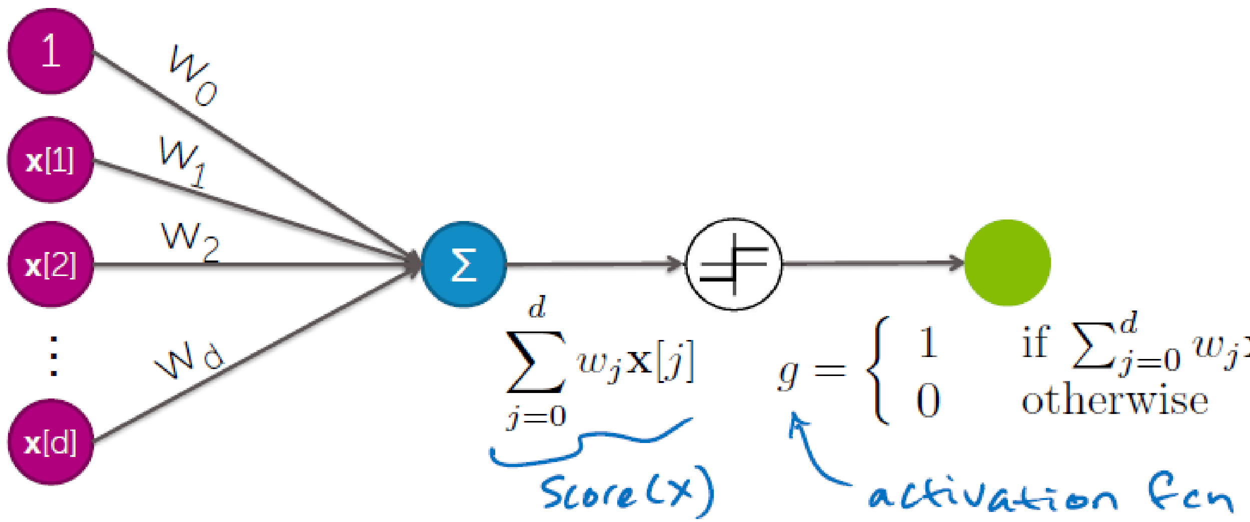


Graph representation of classifier: useful for defining neural networks



Single layer neural network

Perceptron as a neural network



$$\sum_{j=0}^{d} w_j \mathbf{x}[j]$$

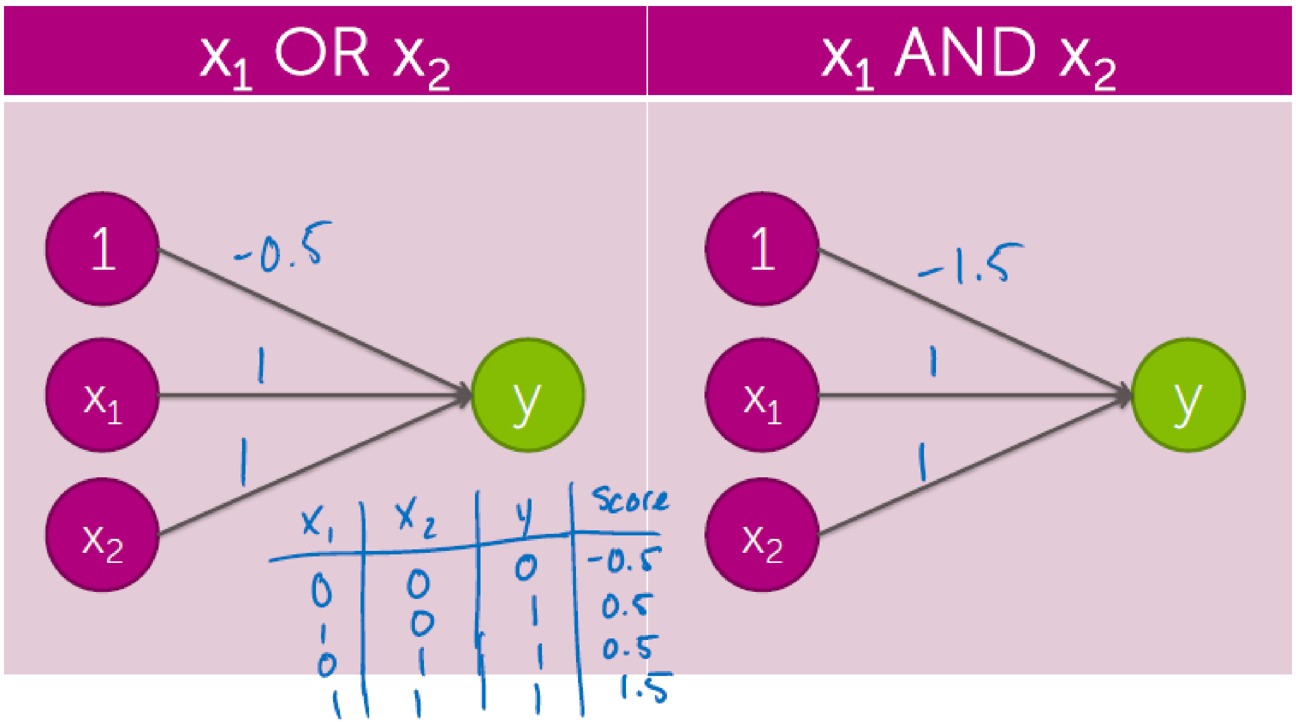
$$g = \begin{cases} 1 & \text{if } \sum_{j=0}^{d} w_j \mathbf{x}[j] > 0 \\ 0 & \text{otherwise} \end{cases}$$

This is one **neuron**:

- Input edges **x**[1],…,**x**[d], along with intercept **x**[0]=1
- Sum passed through an activation function g

Single layer neural network

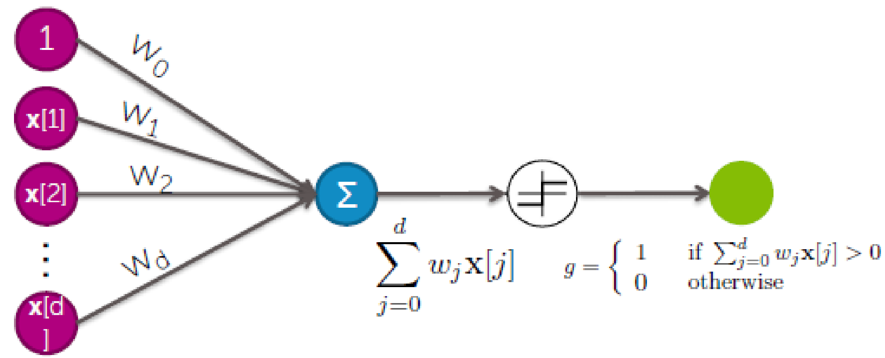
What can a linear classifier represent?



x_1	x_2	y	Score
0	0	0	-0.5
1	0	1	0.5
0	1	1	0.5
1	1	1	1.5

Single layer neural network

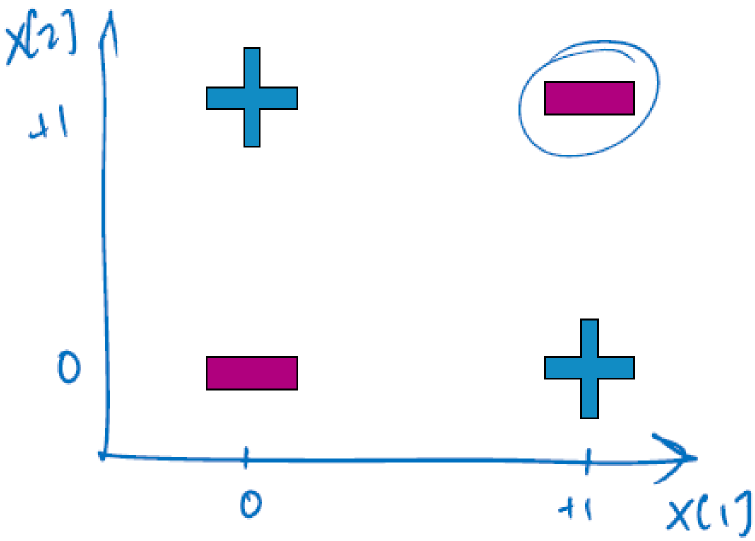
Perceptron, linear classification, Boolean fns: x[j]∈{0,1}



- Can learn **x**[1] OR **x**[2]?
  - -0.5 + **x**[1] + **x**[2]
- Can learn **x**[1] AND **x**[2]?
  - -1.5 + **x**[1] + **x**[2]
- Can learn any conjunction or disjunction?
  - 0.5 + **x**[1] + … + **x**[d]
  - (-d+0.5) + **x**[1] + … + **x**[d]
- Can learn majority?
  - (-0.5*d) + **x**[1] + … + **x**[d]
- What are we missing? The dreaded XOR!, etc.

Single layer neural network

What can't a simple linear classifier represent?



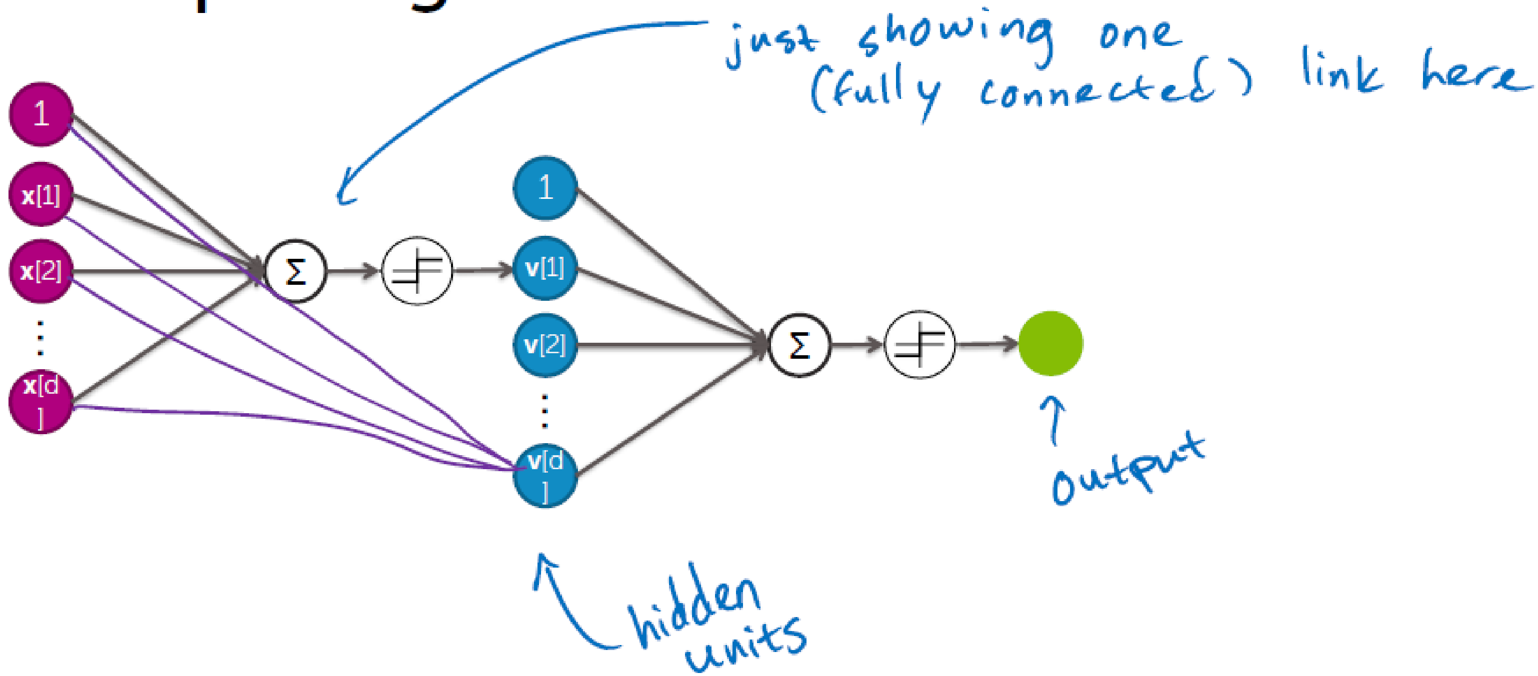
XOR
the counterexample to everything

Need non-linear features

XOR = **x**[1] AND NOT **x**[2] OR NOT **x**[1] AND **x**[2]

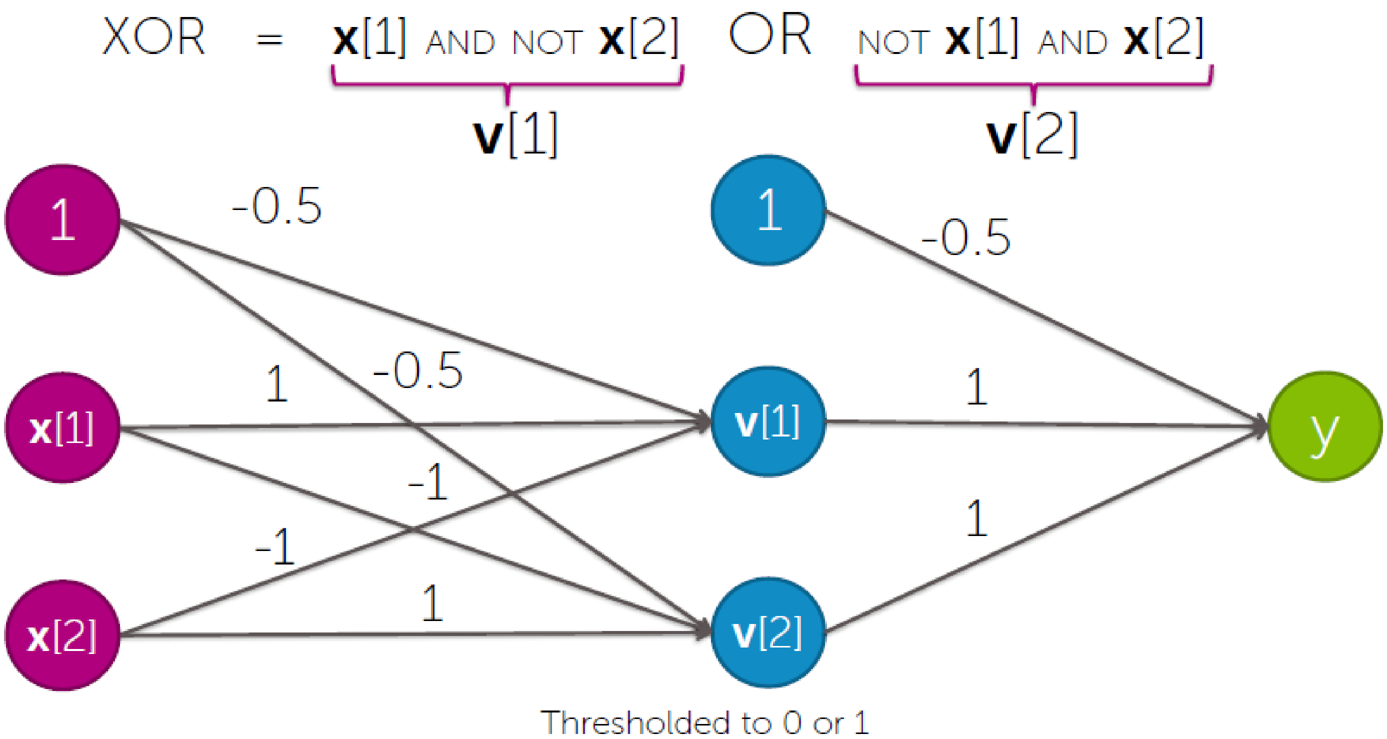
Introducing a hiden layer

Composing individual neurons



Introducing a hiden layer

Solving the XOR problem: Going beyond linear classification by adding a layer



Introducing a hiden layer

Solving the XOR problem: Going beyond linear classification by adding a layer

$y = \mathbf{x}[1] \text{ XOR } \mathbf{x}[2] = (\mathbf{x}[1] \text{ AND } \neg\mathbf{x}[2]) \text{ OR } (\mathbf{x}[2] \text{ AND } \neg\mathbf{x}[1])$

$\mathbf{v}[1] = (\mathbf{x}[1] \text{ AND } \neg\mathbf{x}[2])$ ← x[1]=1, x[2]=0 → 1

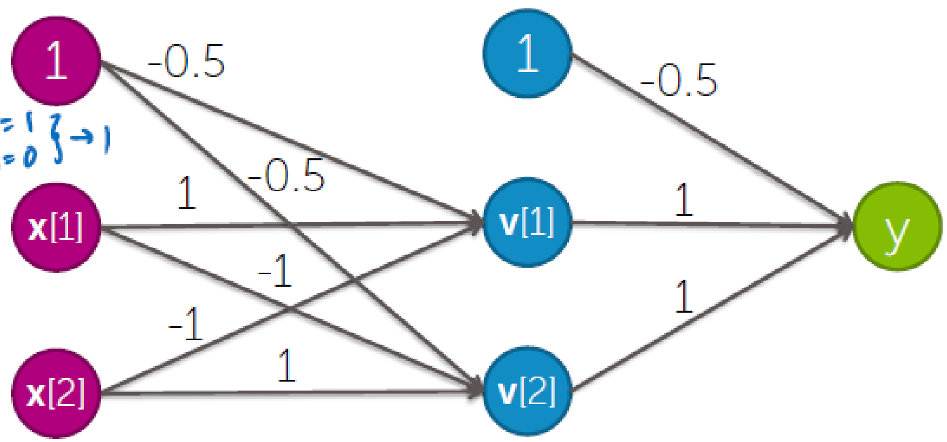
$= g(-0.5+\mathbf{x}[1]-\mathbf{x}[2])$

$\mathbf{v}[2] = (\mathbf{x}[2] \text{ AND } \neg\mathbf{x}[1])$ ← x[2]=1, x[1]=0 } → 1

$= g(-0.5+\mathbf{x}[2]-\mathbf{x}[1])$

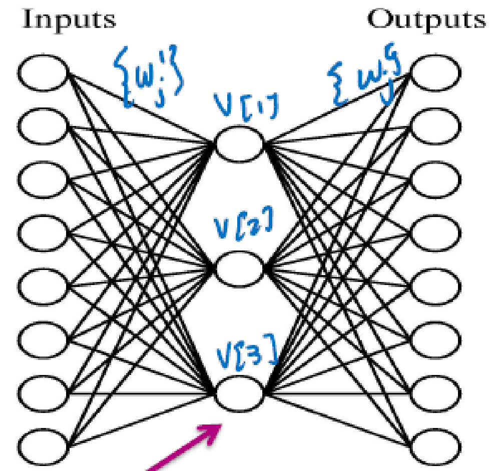
$y = \mathbf{v}[1] \text{ OR } \mathbf{v}[2]$

$= g(-0.5+\mathbf{v}[1]+\mathbf{v}[2])$



Thresholded to 0 or 1

Introducing a hiden layer



Hidden layer

Single unit:

activation fcn

$$out(\mathbf{x}) = g(w_0 + \sum_j w_j \mathbf{x}[j])$$

score(x)

1-hidden layer:

weights on output layer

$$out(\mathbf{x}) = g(w_0 + \sum_k w_k g(w_0^k + \sum_j w_j^k \mathbf{x}[j]))$$

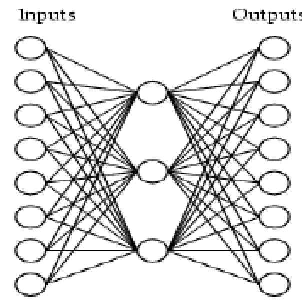
j th weight for the k th hidden unit

$\mathbf{v}[k]$

No longer convex function!

Neural net with hiden layer

Example data for neural net with hidden layer



A target function:

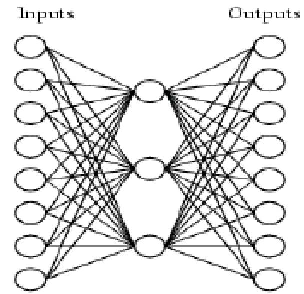
Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

0 0 0 → 0
0 0 1
⋮
1 1 1

Can this be learned??

Neural net with hiden layer

A network:



Learned weights for hidden layer

using diff g!

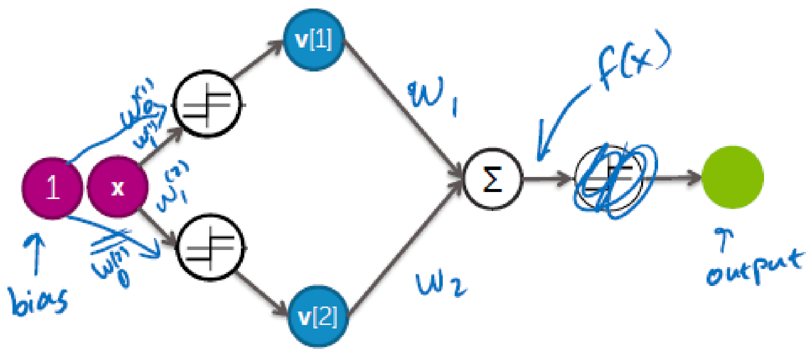
Learned hidden layer representation:

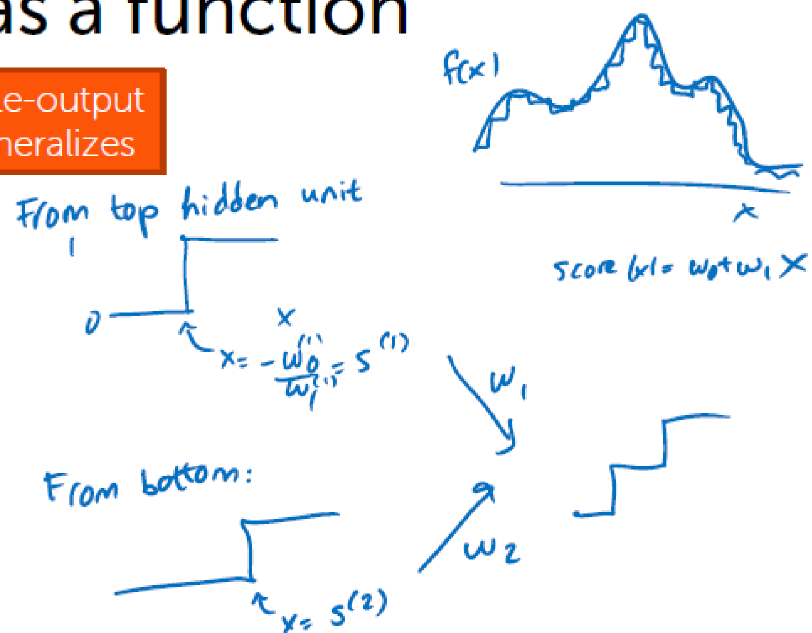
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Neural net with hiden layer

2-layer neural network as a function approximator

Single-input/single-output example, but generalizes





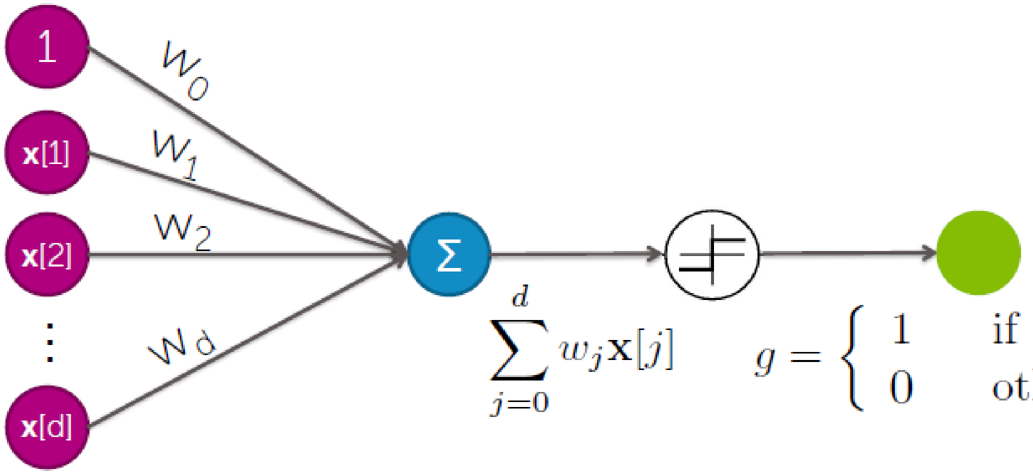
Composing unit step functions
→ piecewise constant function approximator

How to get better approximation? more hidden units!

Nice overview: http://neuralnetworksanddeeplearning.com/chap4.html

More general NN

So far focused on thresholding activation



$$\sum_{j=0}^{d} w_j \mathbf{x}[j] \qquad g = \begin{cases} 1 & \text{if } \sum_{j=0}^{d} w_j \mathbf{x}[j] > 0 \\ 0 & \text{otherwise} \end{cases}$$

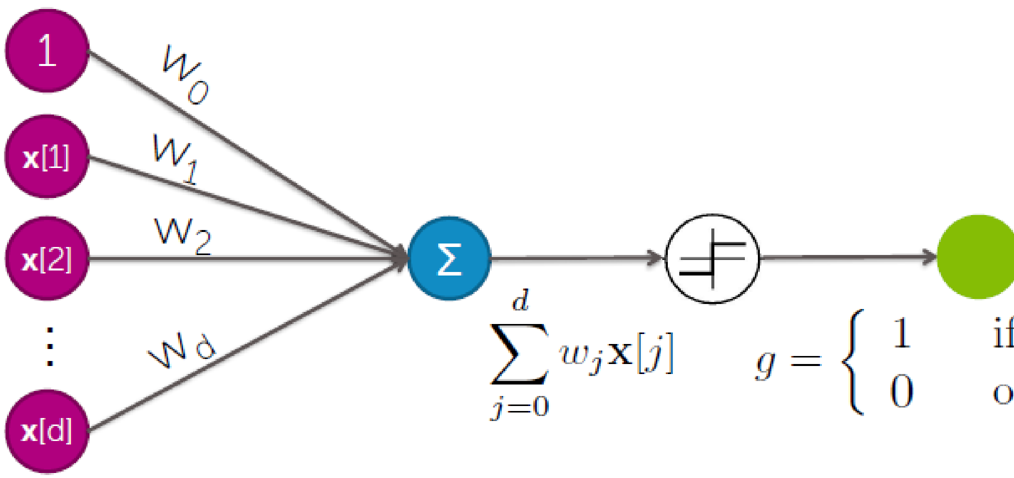
Not actually used in practice!

Can't compute gradients

(will come back to this)

More general NN

So far focused on thresholding activation



$$\sum_{j=0}^{d} w_j \mathbf{x}[j] \qquad g = \begin{cases} 1 & \text{if } \sum_{j=0}^{d} w_j \mathbf{x}[j] > 0 \\ 0 & \text{otherwise} \end{cases}$$

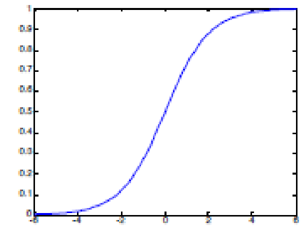
Not actually used in practice!

Can't compute gradients

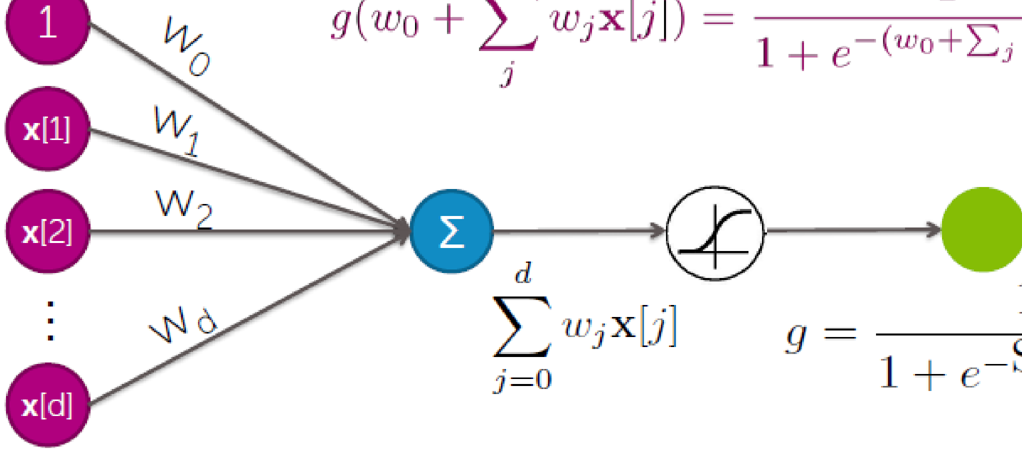
(will come back to this)

More general NN

Sigmoid neuron



$$g(w_0 + \sum_j w_j \mathbf{x}[j]) = \frac{1}{1 + e^{-(w_0 + \sum_j w_j \mathbf{x}[j])}}$$



$$\sum_{j=0}^{d} w_j \mathbf{x}[j]$$

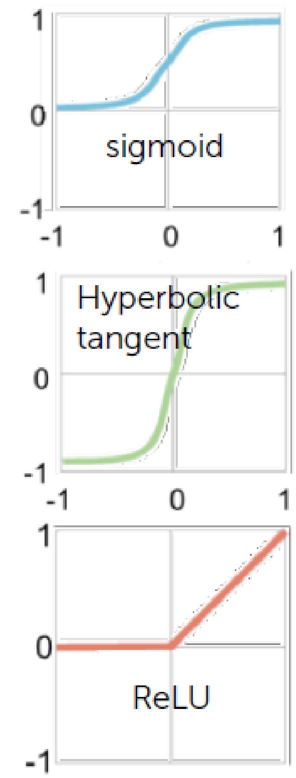
$$g = \frac{1}{1 + e^{-\text{Score}(\mathbf{x})}}$$

Just change g!

- Notice the output range [0,1]. What was it before? *0 or 1*
- Look familiar? *logistic regression*
- Why would we want to do this? *differentiable!*

More general NN

Some choices of activation function g

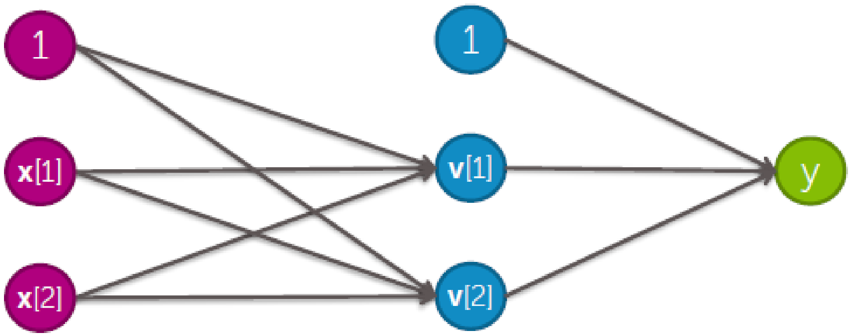


- Sigmoid
  - Historically popular, but (mostly) fallen out of favor
    - Neuron's activation saturates (weights get very large → gradients get small)
    - Not zero-centered → other issues in the gradient steps
  - When put on the output layer, called "softmax" because interpreted as class probability (soft assignment)
- Hyperbolic tangent $g(x) = \tanh(x)$
  - Saturates like sigmoid unit, but zero-centered
- Rectified linear unit (ReLU) $g(x) = x^{+} = \max(0,x)$
  - Most popular choice these days
  - Fragile during training and neurons can "die off"… be careful about learning rates
  - "Noisy" or "leaky" variants
- Softplus $g(x) = \log(1+\exp(x))$
  - Smooth approximation to rectifier activation

More general NN

A general neural network

- Layers and layers and layers of linear models and non-linear transformations



- Around for about 50 years
  - Fell in "disfavor" in 90s
- In last few years, big resurgence
  - Impressive accuracy on several benchmark problems
  - Powered by huge datasets, GPUs, & modeling/learning alg improvements

More general NN

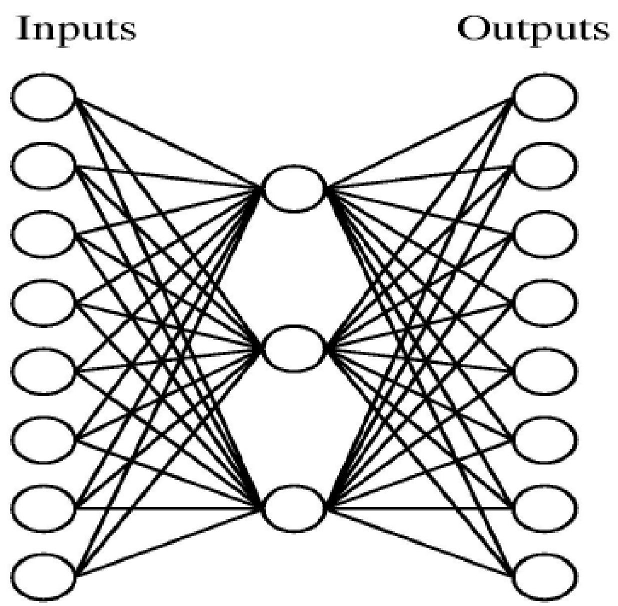
Overfitting in NNs

Are NNs likely to overfit?

- *Yes*, they can represent arbitrary functions!!!

Avoiding overfitting?

- More training data
- Fewer hidden nodes / better topology
  - **Rule of thumb**: 3-layer NNs outperform 2-layer NNs, but going deeper rarely helps (different story for convolutional networks!)
- Regularization
- Early stopping

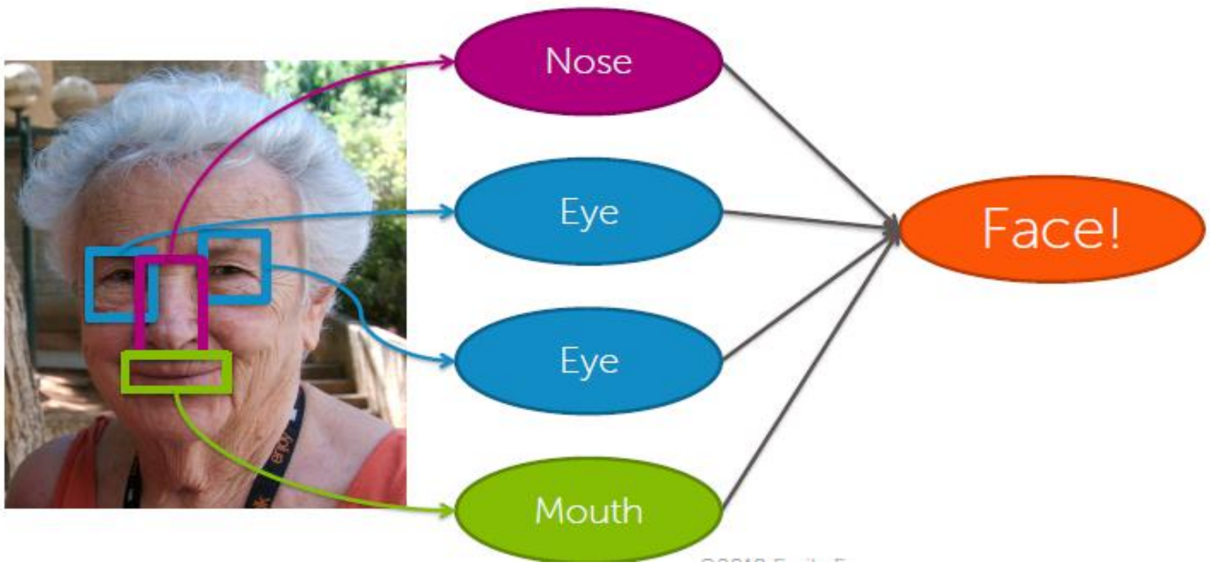


Application to computer vision

Image features

Features = local detectors

- Combined to make prediction
- (in reality, features are more low-level)

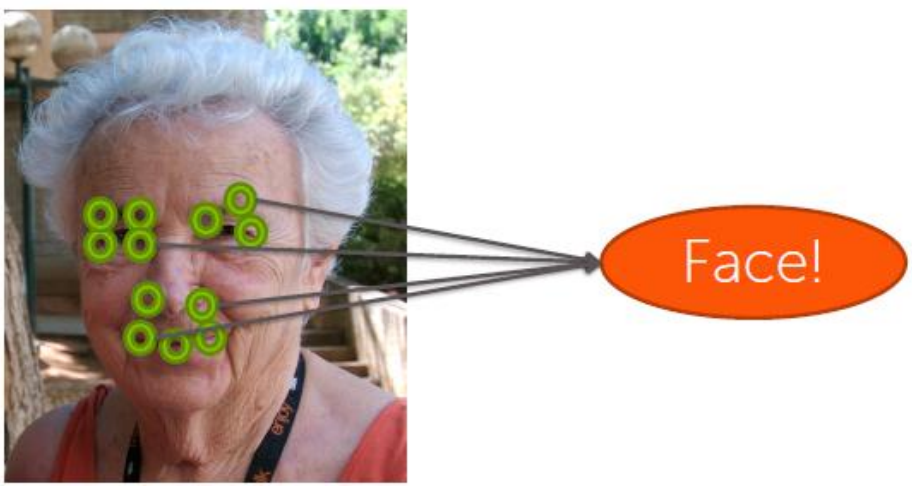


Application to computer vision

Typical local detectors look for locally "interesting points" in image

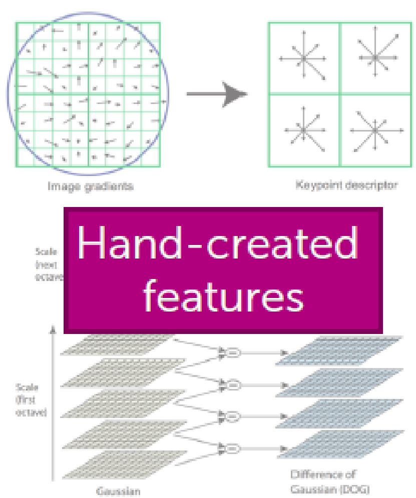
Image features: collections of locally interesting points

– Combined to build classifiers



Application to computer vision

Many hand created features exist for finding interest points…



SIFT [Lowe '99]

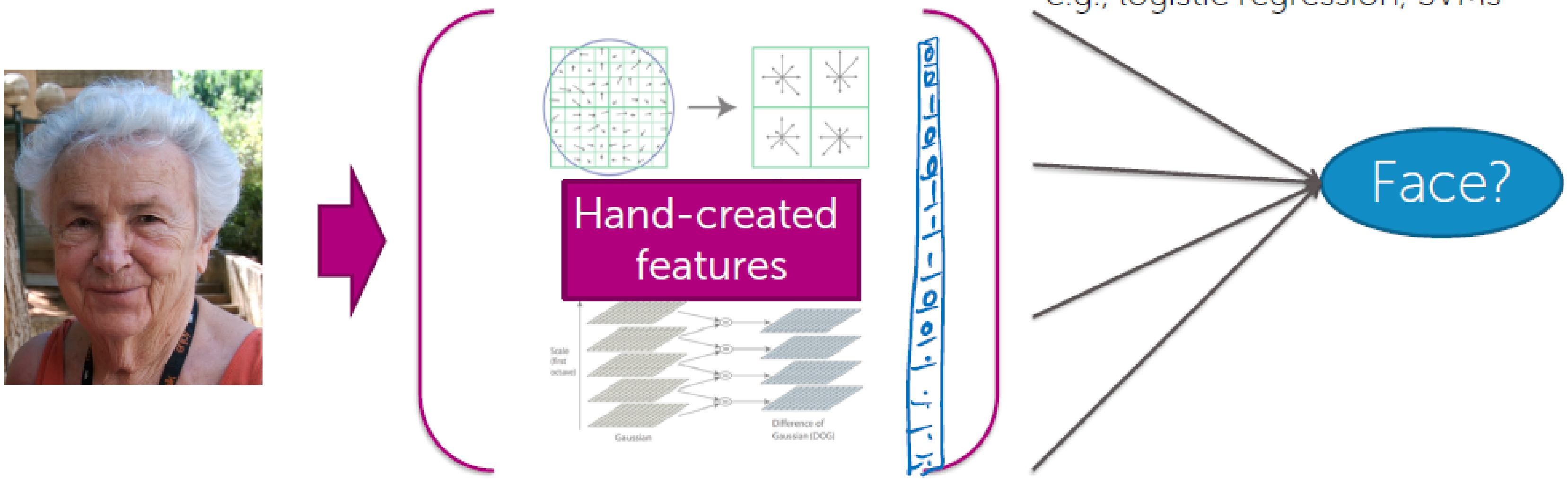
- *Spin Images* [Johnson & Herbert '99]
- *Textons* [Malik et al. '99]
- *RIFT* [Lazebnik '04]
- *GLOH* [Mikolajczyk & Schmid '05]
- *HoG* [Dalal & Triggs '05]
- …

… but very painful to design

Application to computer vision

Standard image classification approach

Input | Extract features | Use simple classifier
e.g., logistic regression, SVMs



Application to computer vision

Deep learning:
implicitly learns features