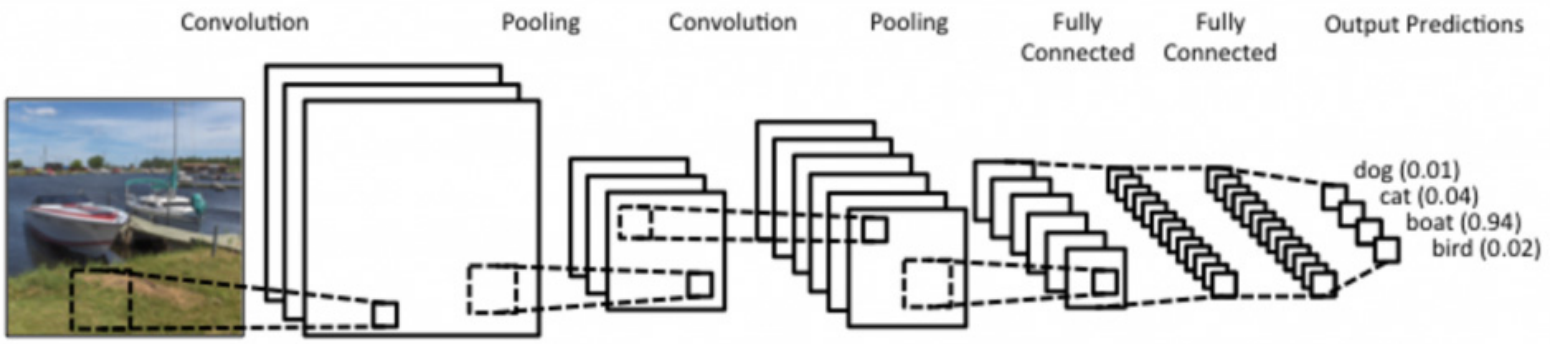
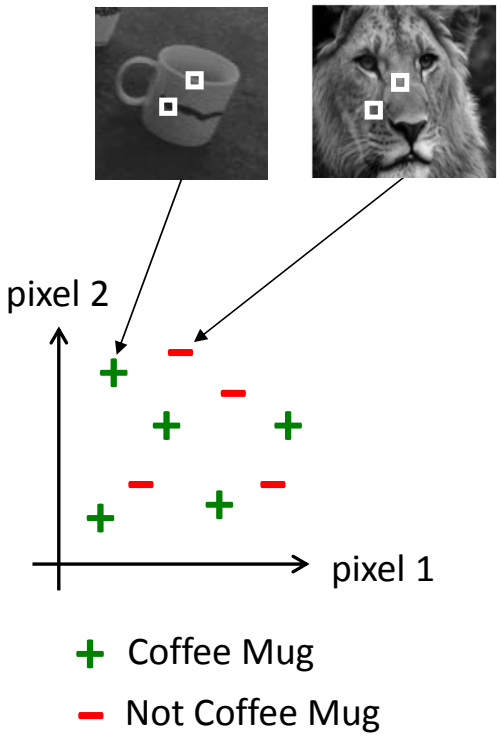
Introduction to Convolutional Neural Networks



Evangelos Kalogerakis

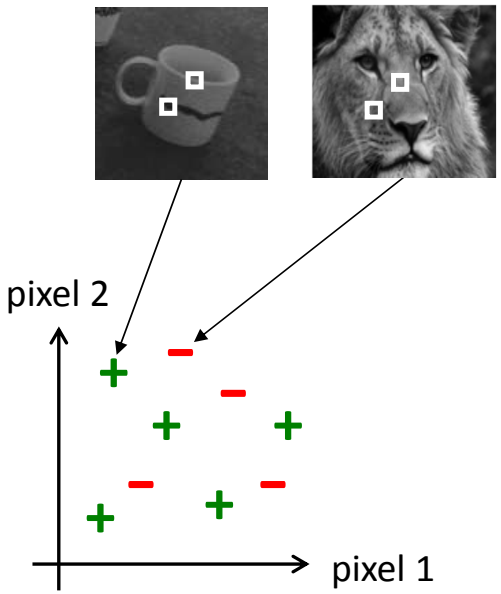
UMASS AMHERST

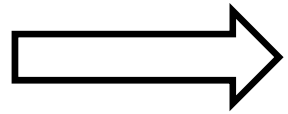
Motivation

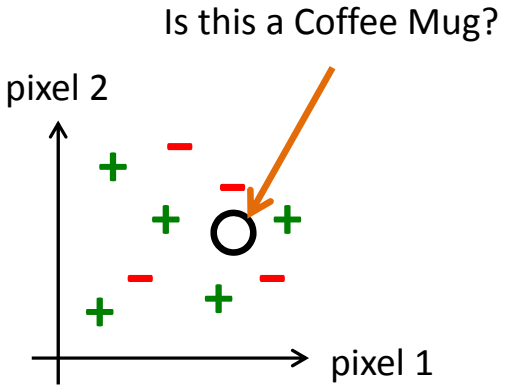


modified slides originally by Adam Coates

Motivation





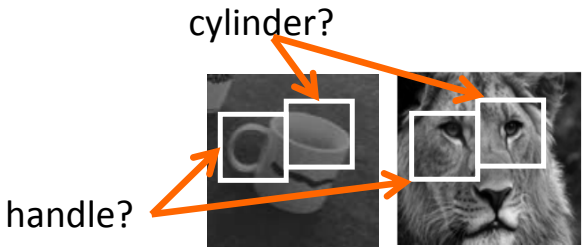


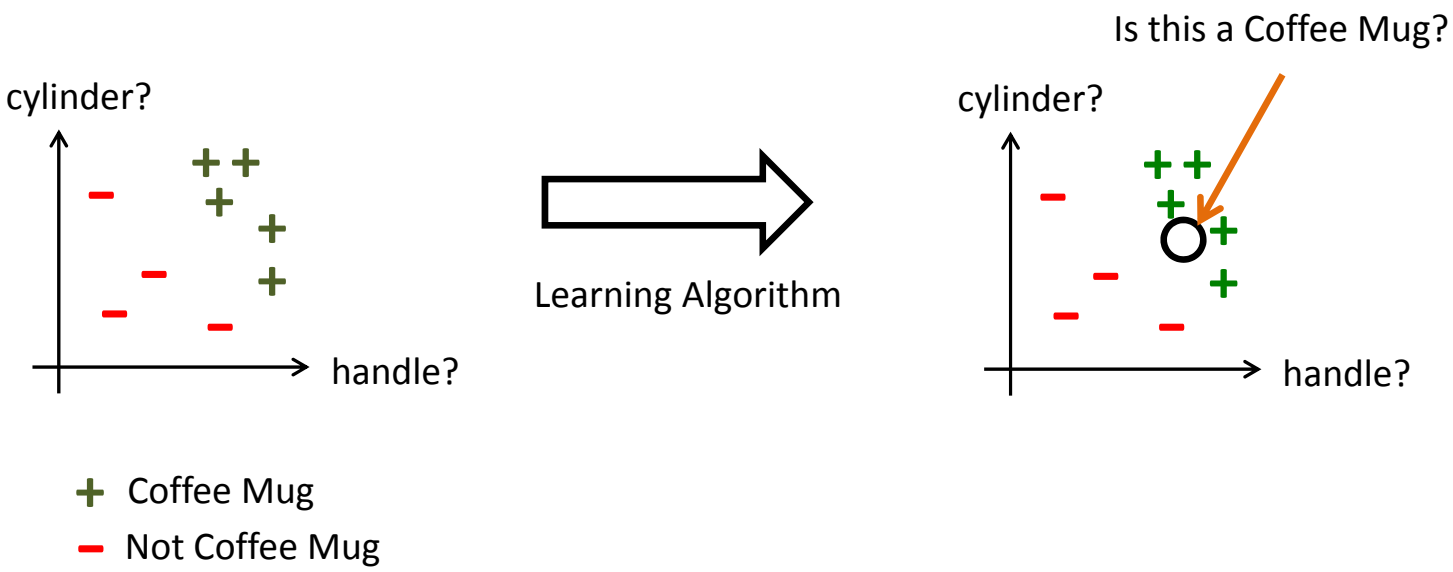
+ Coffee Mug

− Not Coffee Mug

modified slides originally by Adam Coates

Need stronger feature representations!



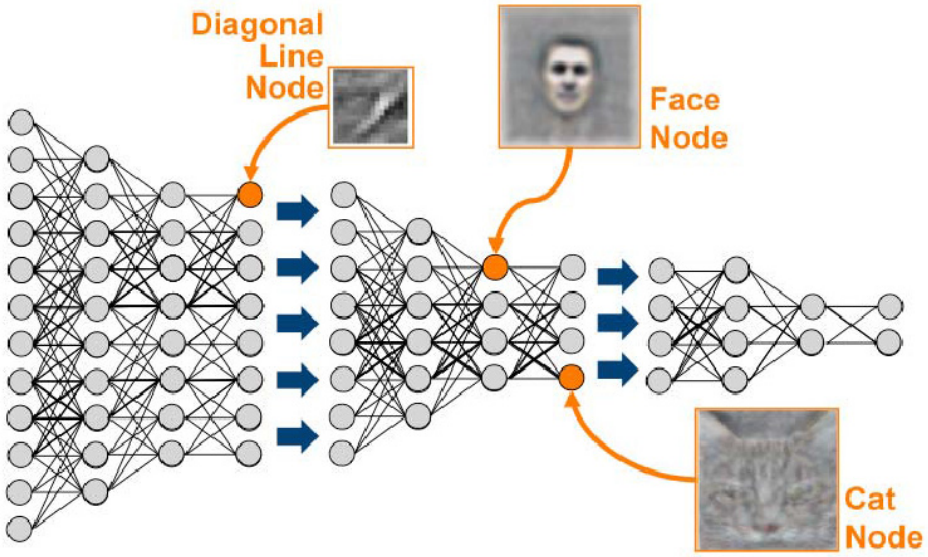


modified slides originally by Adam Coates

From “swallow” to “deep” mappings (networks)

Images, shapes, natural language have **compositional structure and patterns.**

Deep neural networks can learn powerful feature representations capturing those.



Classification basics: Logistic Regression

Suppose you want to predict **mug** or **no mug** in an image.

Output: $\mathbf{y} = \mathbf{1}$ $[coffee\,mug]$, $\mathbf{y} = 0$ $[no\,coffee\,mug]$

Input: $\mathbf{x} = \{x_1, x_2, \ldots\}$ [pixel intensities, gradients, SIFT, etc]

Classification basics: Logistic Regression

Suppose you want to predict **mug** or **no mug** in an image.

Output: $\mathbf{y} = \mathbf{1}$ [*coffee mug*], $\mathbf{y} = 0$ [*no coffee mug*]

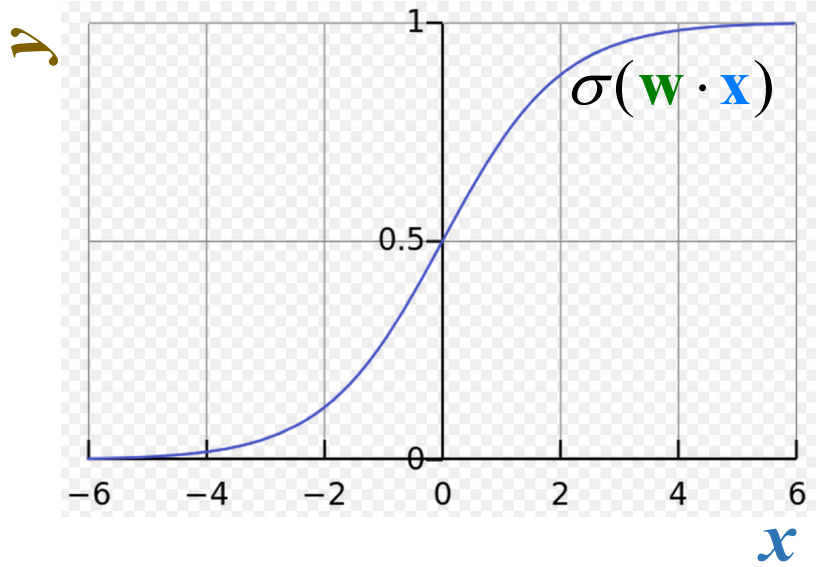
Input: $\mathbf{x} = \{x_1, x_2, \ldots\}$ [pixel intensities, gradients, SIFT, etc]

Classification function:

$$P(\mathbf{y} = \mathbf{1} \mid \mathbf{x}) = \mathbf{f}(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x})$$

where $\mathbf{w}$ is a **weight vector (parameters)**

$$\sigma(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x})}$$



Logistic regression: training

Need to estimate parameters $\boldsymbol{w}$ from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ **(**mugs/no mugs) *(*$i=1 \ldots N$ training images*)*

Find parameters that **maximize probability of training data**

$$\max_{\mathbf{w}} \prod_{i=1}^{N} P(\mathbf{y}=1 \mid \mathbf{x}_i)^{[\mathbf{y_i}==1]} [1 - P(\mathbf{y}=1 \mid \mathbf{x}_i)]^{[\mathbf{y_i}==0]}$$

Logistic regression: training

Need to estimate parameters $\boldsymbol{w}$ from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ **(**mugs/no mugs) *(i=1…N* training images*)*

Find parameters that **maximize probability of training data**

$$\max_{\mathbf{w}} \prod_{i=1}^{N} \sigma(\mathbf{w} \cdot \mathbf{x}_i)^{[\mathbf{y_i}==1]} [1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i)]^{[\mathbf{y_i}==0]}$$

Logistic regression: training

Need to estimate parameters $\boldsymbol{w}$ from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ **(**mugs/no mugs) *(i=1…N* training images*)*

Find parameters that **maximize the log probability of training data**

$$\max_{\mathbf{w}} \log\{\prod_{i=1}^{N} \sigma(\mathbf{w}\cdot\mathbf{x}_i)^{[\mathbf{y_i}==1]}[1-\sigma(\mathbf{w}\cdot\mathbf{x}_i)]^{[\mathbf{y_i}==0]}\}$$

Logistic regression: training

Need to estimate parameters $\boldsymbol{w}$ from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ **(**mugs/no mugs) *(*$i=1 \ldots N$ training images*)*

Find parameters that **maximize the log probability of training data**

$$\max_{\mathbf{w}} \sum_{i=1}^{N} [\mathbf{y_i} == 1] \log \sigma(\mathbf{w} \cdot \mathbf{x}_i) + [\mathbf{y_i} == 0] \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i))$$

Logistic regression: training

Need to estimate parameters $\boldsymbol{w}$ from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ **(**mugs/no mugs) *(*$i=1 \ldots N$ training images*)*

Find parameters that **minimize the negative log probability of training data**

$$\min_{\mathbf{w}} -\sum_{i=1}^{N} [\mathbf{y_i} == 1] \log \sigma(\mathbf{w} \cdot \mathbf{x}_i) + [\mathbf{y_i} == 0] \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i))$$

Logistic regression: training

Need to estimate parameters w from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ (mugs/no mugs) $(i=1 \ldots N$ training images$)$

This is called (**negative**) **log likelihood:**

$$\min_{\mathbf{w}} -\sum_{i=1}^{N} [\mathbf{y}_i == 1] \log \sigma(\mathbf{w} \cdot \mathbf{x}_i) + [\mathbf{y}_i == 0] \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i))$$

$L(w)$

Logistic regression: training

Need to estimate parameters w from training data e.g., images of objects $\mathbf{x_i}$ and given labels $\mathbf{y_i}$ (mugs/no mugs) *(i=1…N training images)*

We now have an **optimization problem:**

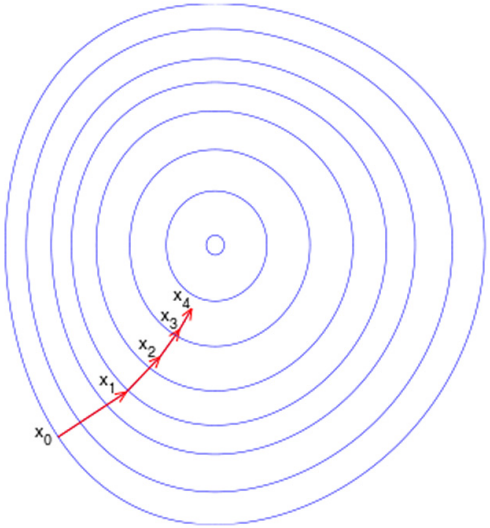
$$\min_{\mathbf{w}} -\sum_{i=1}^{N} [\mathbf{y}_i == 1] \log \sigma(\mathbf{w} \cdot \mathbf{x}_i) + [\mathbf{y}_i == 0] \log(1 - \sigma(\mathbf{w} \cdot \mathbf{x}_i))$$

L(w)

$$\frac{\partial L(\mathbf{w})}{\partial w_d} = \sum_i x_{i,d} [\, y_i - \sigma(\mathbf{w} \cdot \mathbf{x}_i) \,]$$

(partial derivative for d[th] parameter)

How can we minimize/maximize a function?



Gradient descent: Given a random initialization of parameters and a step rate η, update them according to:

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla L(\mathbf{w})$$

See also **quasi-Newton** and **IRLS** methods

Regularization

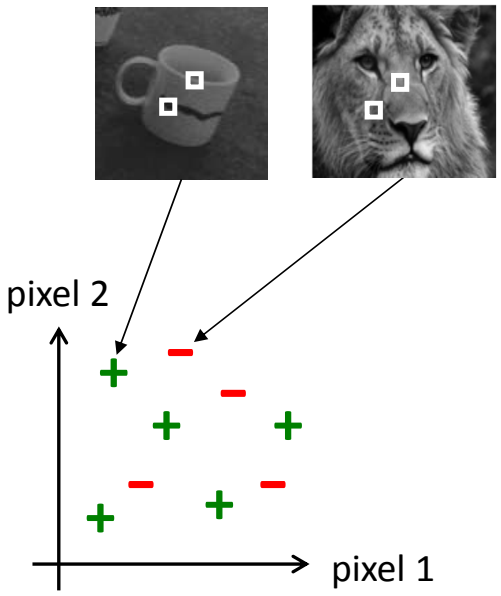
Overfitting: few training data and number of parameters is large!

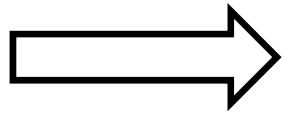
Penalize large weights:

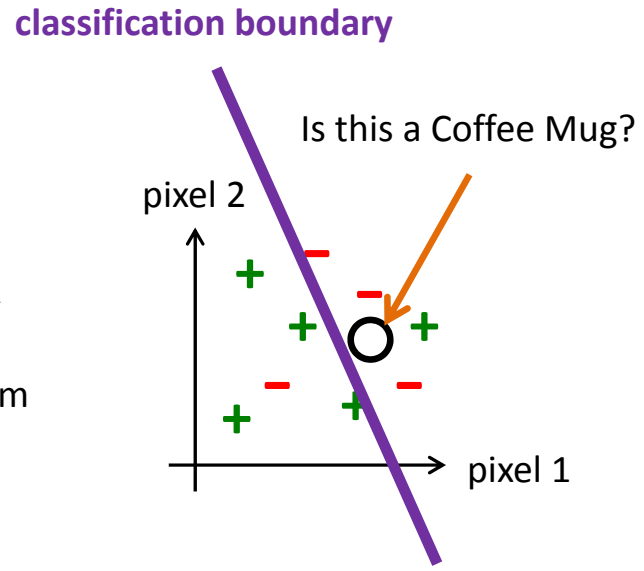
$$\min_{\mathbf{w}} L(\mathbf{w}) + \lambda \sum_{d} \mathbf{w}_d{}^2$$

Called **ridge regression (or L2 regularization)**

Back to our original example…

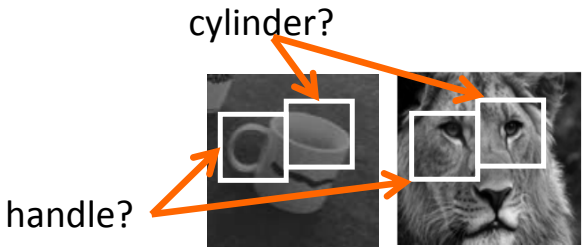


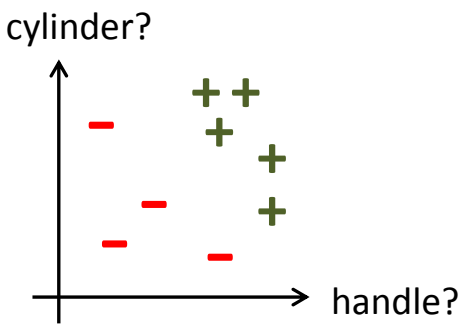


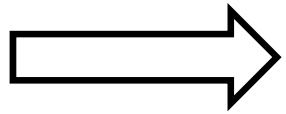


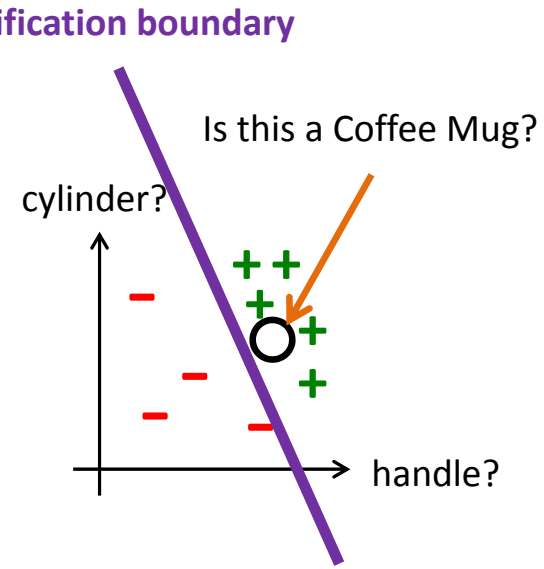
modified slides originally by Adam Coates

How can we learn better feature representations?







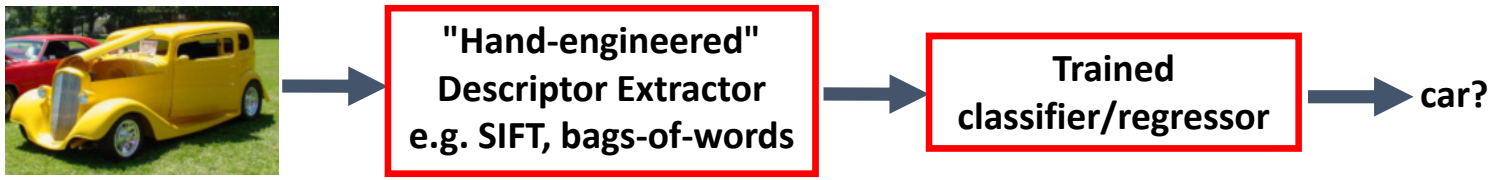


+ Coffee Mug
− Not Coffee Mug

modified slides originally by Adam Coates

"Traditional" recognition pipeline

Fixed/engineered descriptors + **trained** classifier/regressor



modified slides originally by Adam Coates

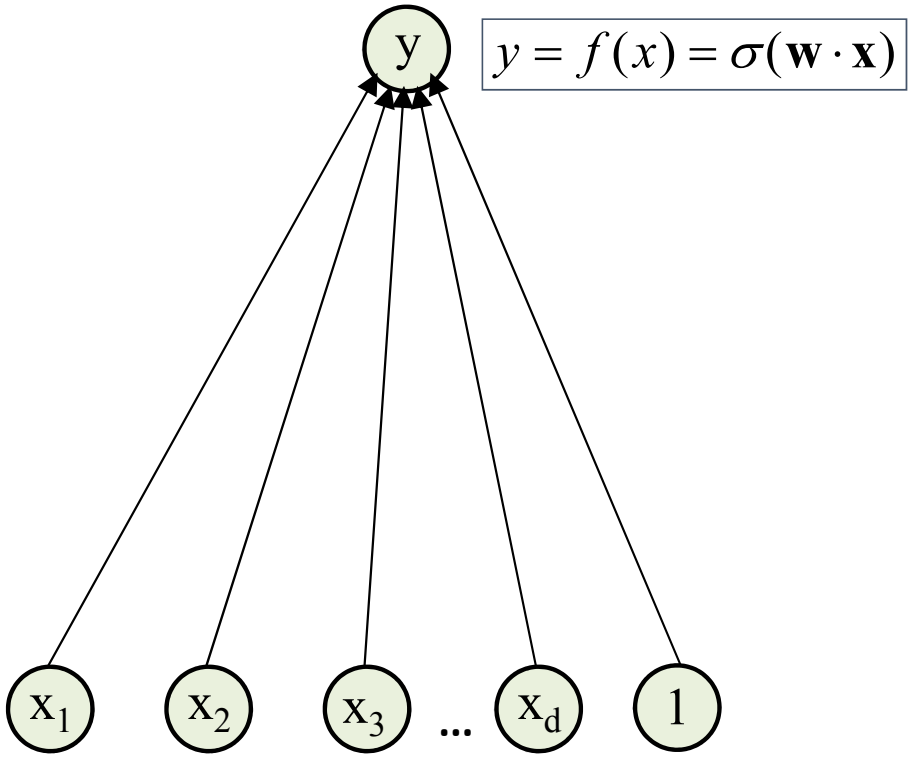
"New" recognition pipeline

Trained descriptors + **trained** classifier/regressor

modified slides originally by Adam Coates

From “swallow” to “deep” mappings (networks)

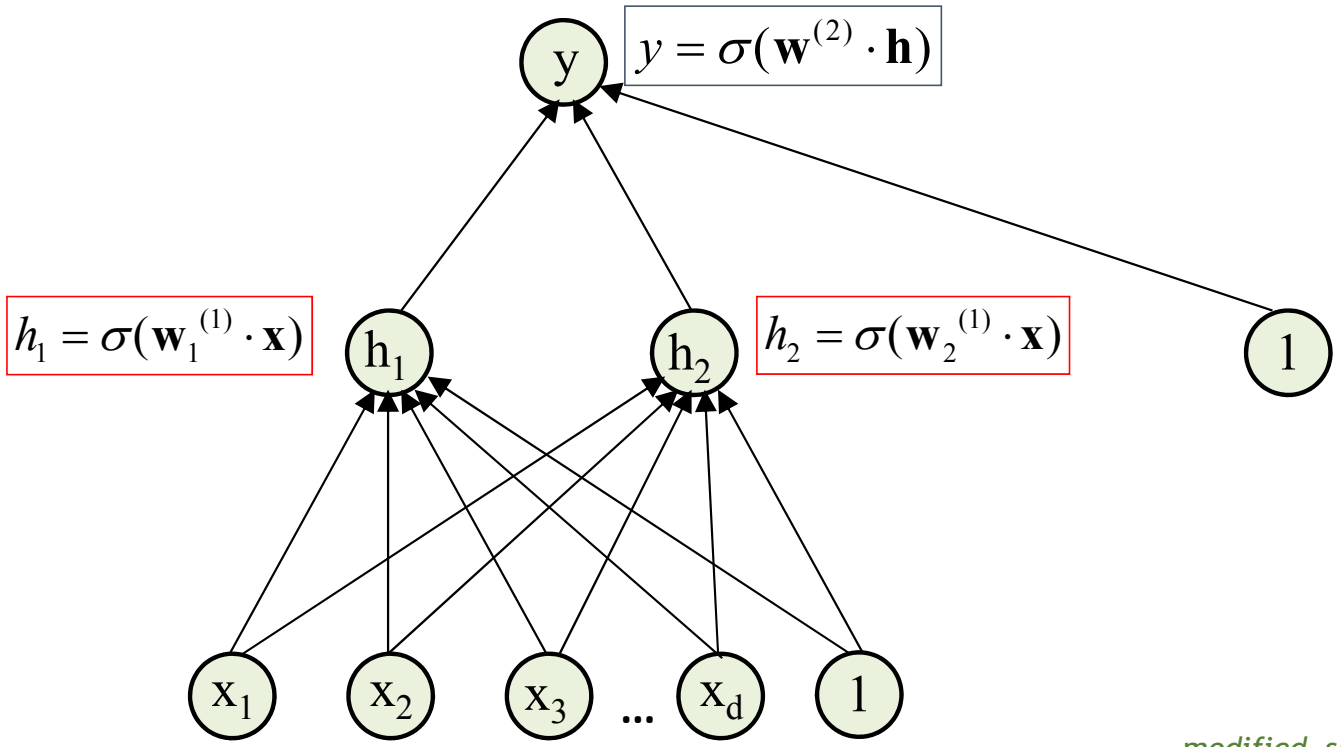
Logistic regression: output is a **direct function of inputs**. Think of it as a net:



modified slides originally by Adam Coates

Neural network

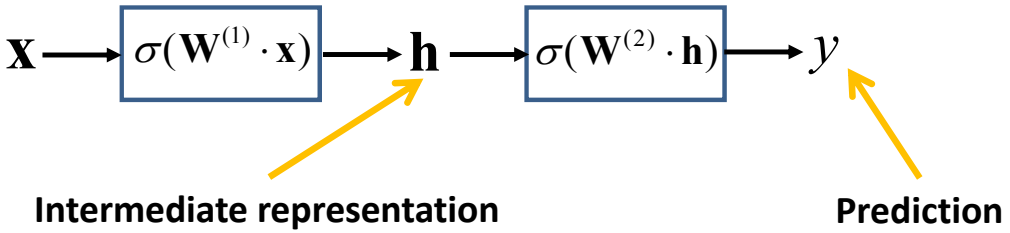
Introduce latent nodes that play the role of **learned feature representations.**



modified slides originally by Adam Coates

Neural network

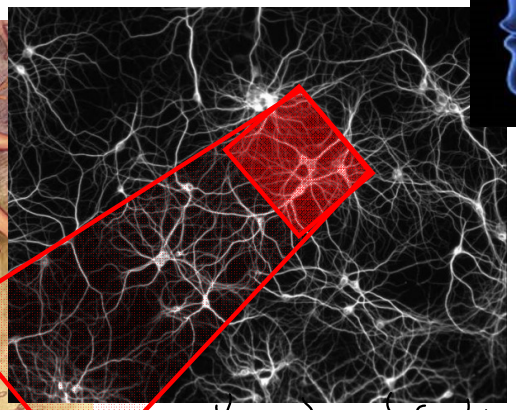
Same as logistic regression but now our output function has **multiple stages** ("layers", "modules").

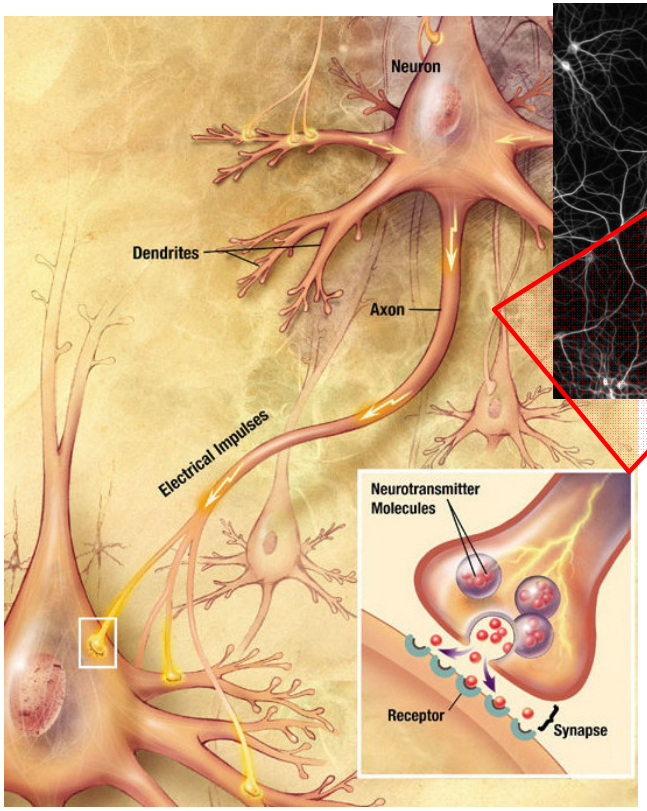


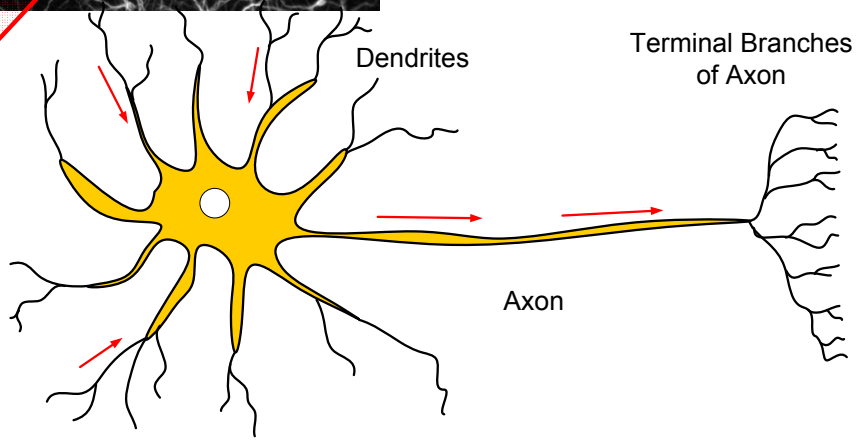
$$where \quad \mathbf{W}^{(\cdot)} = \begin{bmatrix} \mathbf{w}_1^{(\cdot)} \\ \mathbf{w}_2^{(\cdot)} \\ \dots \\ \mathbf{w}_m^{(\cdot)} \end{bmatrix}$$

modified slides originally by Adam Coates

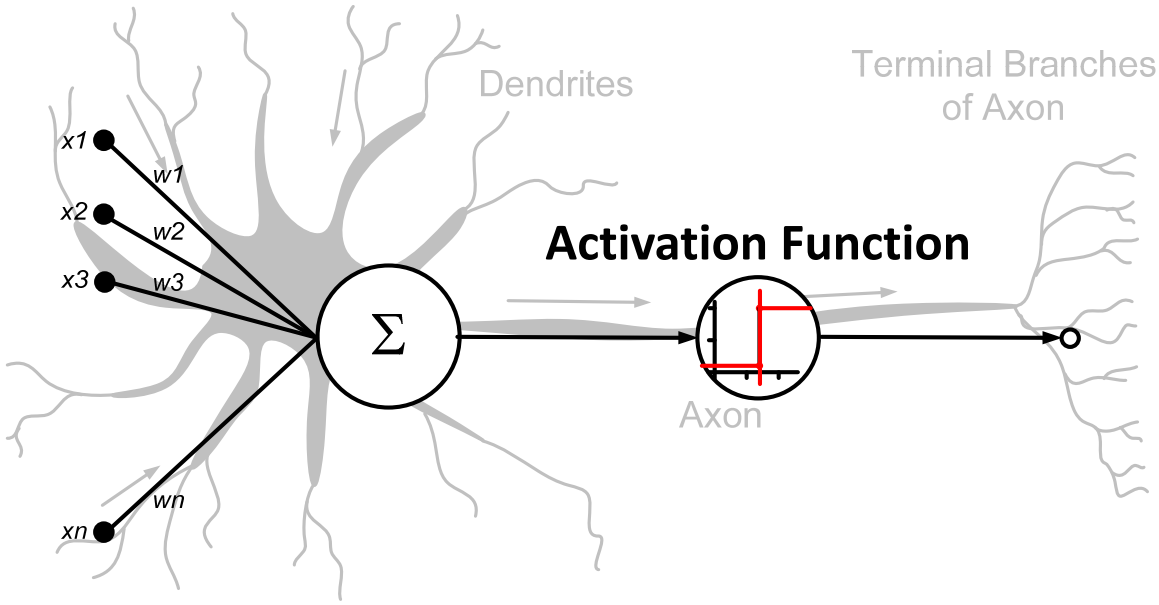
Biological Neurons







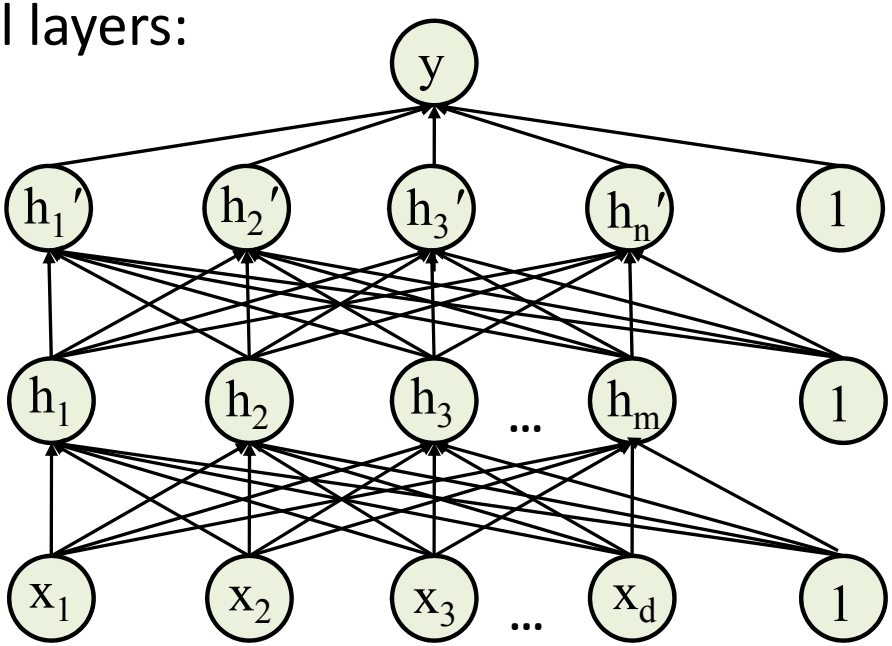
Analogy with biological networks



Slide credit : Andrew L. Nelson

Neural network

Stack up several layers:



modified slides originally by Adam Coates

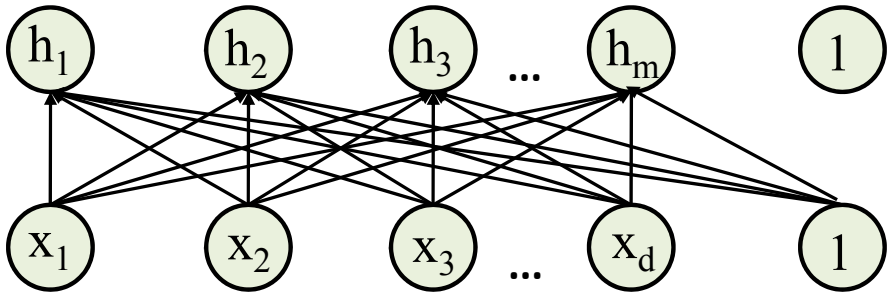
Forward propagation

Process to compute output:

modified slides originally by Adam Coates

Forward propagation

Process to compute output:

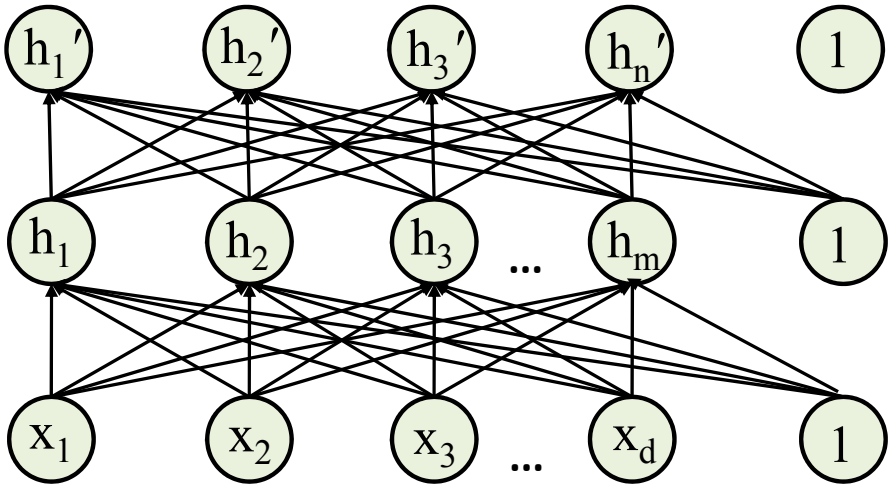


$$\mathbf{x} \rightarrow \boxed{\sigma(\mathbf{W}^{(1)} \cdot \mathbf{x})} \rightarrow \mathbf{h}$$

modified slides originally by Adam Coates

Forward propagation

Process to compute output:

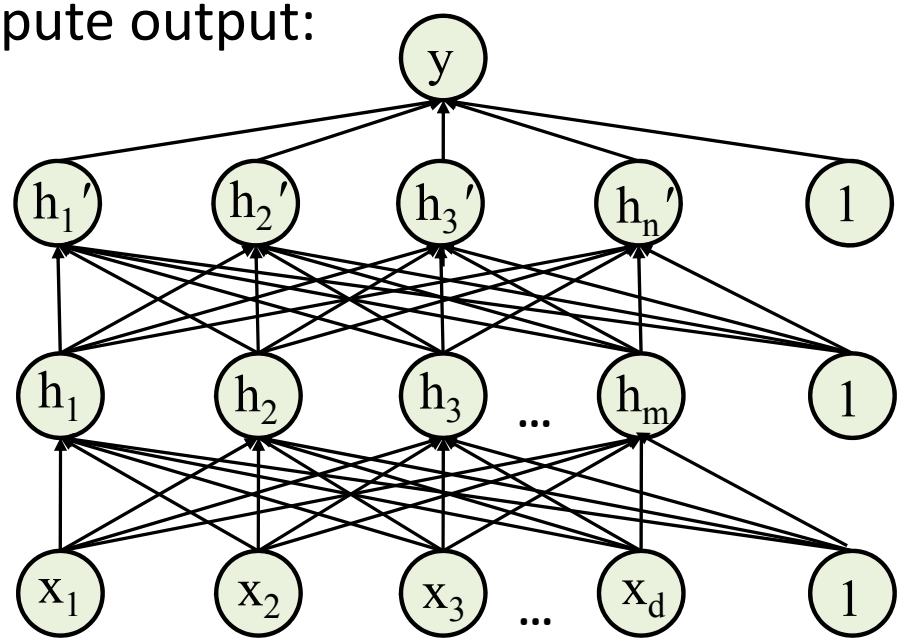


$$\mathbf{x} \rightarrow \boxed{\sigma(\mathbf{W}^{(1)} \cdot \mathbf{x})} \rightarrow \mathbf{h} \rightarrow \boxed{\sigma(\mathbf{W}^{(2)} \cdot \mathbf{h})} \rightarrow \mathbf{h}'$$

modified slides originally by Adam Coates

Forward propagation

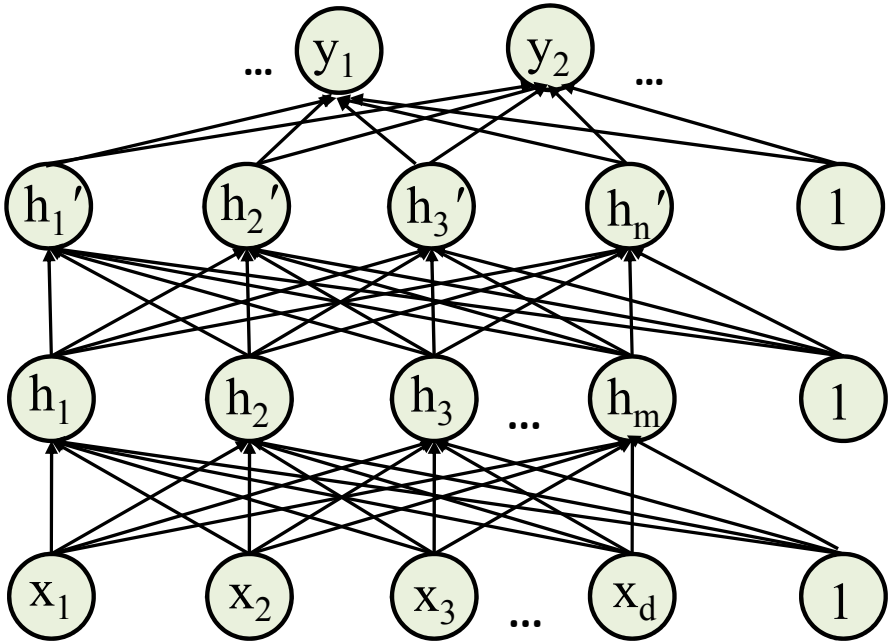
Process to compute output:



$$\mathbf{x} \rightarrow \boxed{\sigma(\mathbf{W}^{(1)} \cdot \mathbf{x})} \rightarrow \mathbf{h} \rightarrow \boxed{\sigma(\mathbf{W}^{(2)} \cdot \mathbf{h})} \rightarrow \mathbf{h}' \rightarrow \boxed{\sigma(\mathbf{W}^{(3)} \cdot \mathbf{h}')} \rightarrow y$$

modified slides originally by Adam Coates

Multiple outputs



$$\mathbf{x} \rightarrow \boxed{\sigma(\mathbf{W}^{(1)} \cdot \mathbf{x})} \rightarrow \mathbf{h} \rightarrow \boxed{\sigma(\mathbf{W}^{(2)} \cdot \mathbf{h})} \rightarrow \mathbf{h}' \rightarrow \boxed{\sigma(\mathbf{W}^{(3)} \cdot \mathbf{h}')} \rightarrow \mathbf{y}$$

modified slides originally by Adam Coates

How can you learn the parameters?

Use a loss function e.g., for classification:

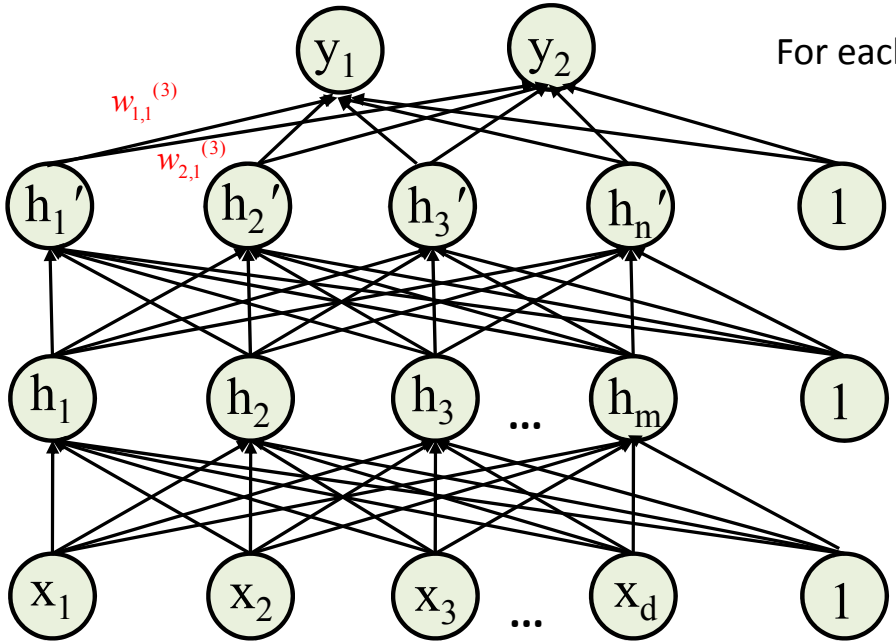
$$L(\mathbf{w}) = -\sum_{i=1} \sum_{output\ t} [\mathbf{y_{i,t}} == 1] \log f_t(\mathbf{x}_i) + [\mathbf{y_{i,t}} == 0] \log(1 - f_t(\mathbf{x}_i))$$

In case of regression i.e., for predicting continuous outputs:

$$L(\mathbf{w}) = \sum_{i} \sum_{output\ t} [\mathbf{y}_{i,t} - f_t(\mathbf{x}_i)]^2$$

Backpropagation

For each training example i (omit index i for clarity):

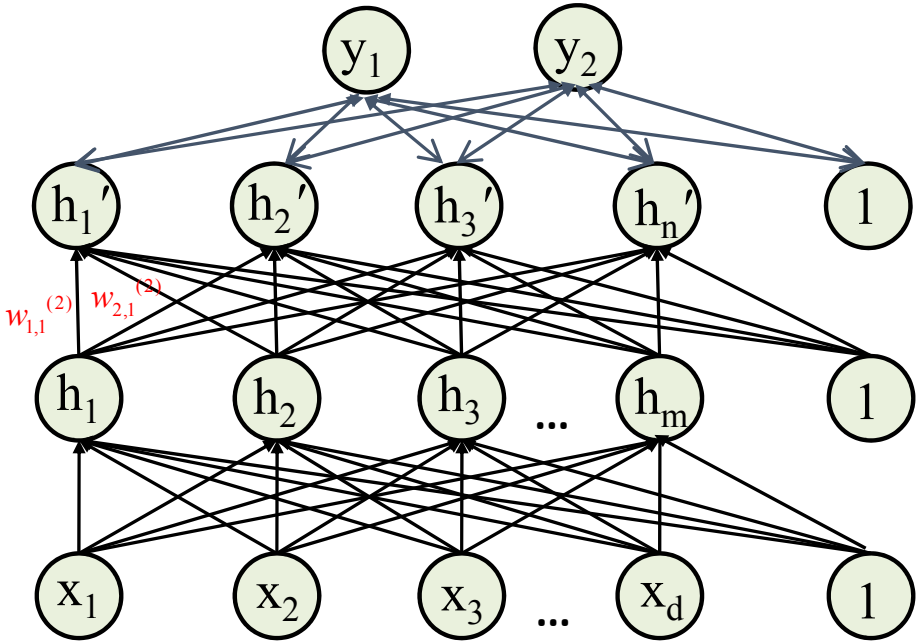


For each output: $\delta_t^{(3)} = y_t - f(\mathbf{x})$

$$\frac{\partial L(\mathbf{w})}{\partial w_{t,n}^{(3)}} = \delta_t^{(3)} h_n$$

Backpropagation

For each training example i (omit index i for clarity):



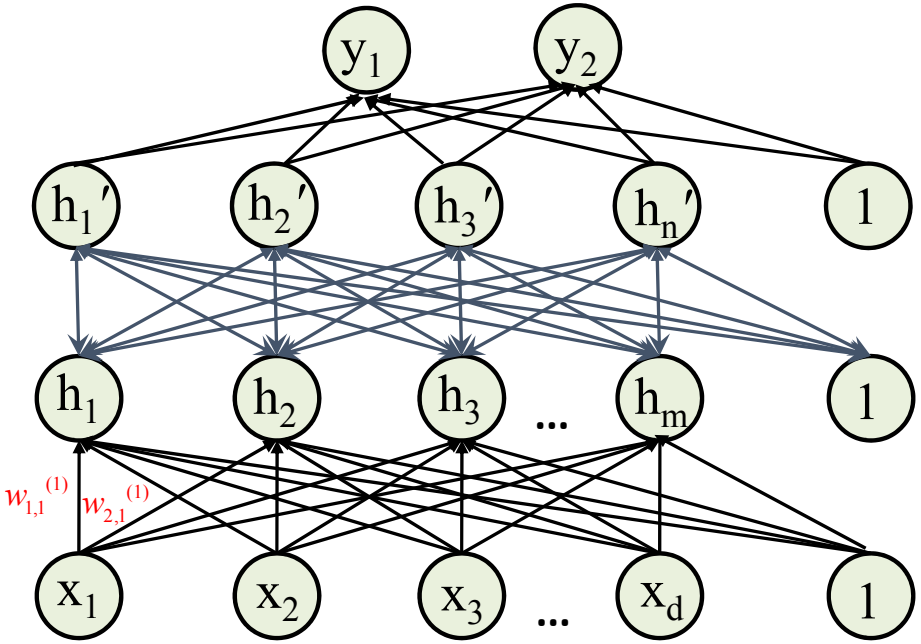
$$\delta_n^{(2)} = \sigma'(\mathbf{w_n}^{(2)} \cdot \mathbf{h}) \sum_t w_{t,n}^{(3)} \delta_t^{(3)}$$

Note: $\sigma'(\cdot) = \sigma(\cdot)[1 - \sigma(\cdot)]$

$$\frac{\partial L(\mathbf{w})}{\partial w_{n,m}^{(2)}} = \delta_n^{(2)} h_m$$

Backpropagation

For each training example i (omit index i for clarity):



$$\delta_m^{(1)} = \sigma'(\mathbf{w}_m^{(1)} \cdot \mathbf{x}) \sum_n w_{n,m}^{(2)} \delta_n^{(2)}$$

$$\frac{\partial L(\mathbf{w})}{\partial w_{m,d}^{(1)}} = \delta_m^{(1)} x_d$$

Is this magic?

All these are derivatives derived analytically using the **chain rule**!

Gradient descent is expressed through **backpropagation of messages** δ following the structure of the model

Training algorithm

For each training example [in a batch]

1. **Forward propagation** to compute outputs per layer
2. **Back propagate** messages δ from top to bottom layer
3. Multiply messages δ with inputs to compute **derivatives per layer**
4. **Accumulate the derivatives** from that training example

Apply the gradient descent rule

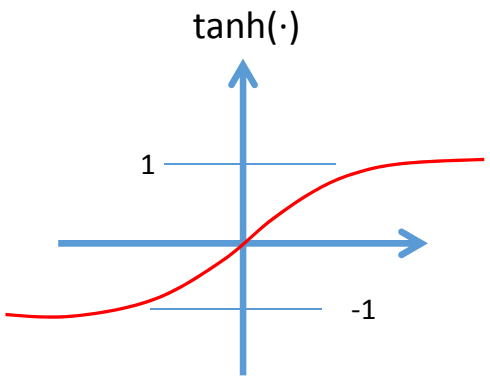
Yet, this does not work so easily...

Yet, this does not work so easily...

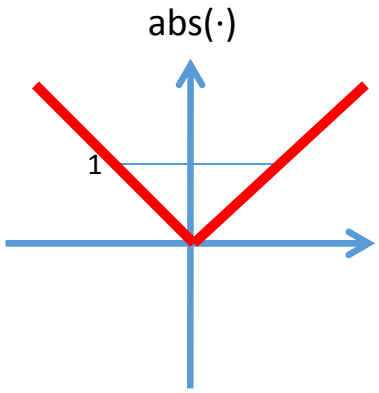
- **Non-convex:** Local minima; convergence criteria.
- Optimization becomes difficult with **many layers**.
- Hard to diagnose and **debug malfunctions**.
- **Many things turn out to matter**:
  - Choice of nonlinearities.
  - Initialization of parameters.
  - Optimizer parameters: step size, schedule.

Non-linearities

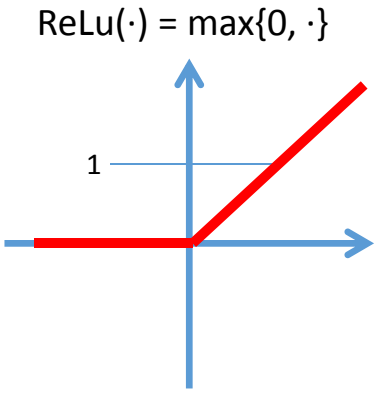
- **Choice of functions inside network matters.**
  - Sigmoid function yields highly non-convex loss functions
  - Some other choices often used:



$tanh'(\cdot) = 1 - tanh(\cdot)^2$



$abs'(\cdot) = sign(\cdot)$

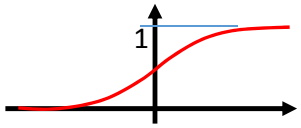


$ReLu'(\cdot) = [\cdot > 0]$

"Rectified Linear Unit"
→ Increasingly popular.
[Nair & Hinton, 2010]

Initialization

- **Usually small random values.**
  - Try to choose so that typical input to a neuron avoids saturating



- **Initialization schemes for weights used as input to a node:**
  - tanh units: Uniform[-r, r]; sigmoid: Uniform[-4r, 4r].
  - See [Glorot et al., AISTATS 2010]

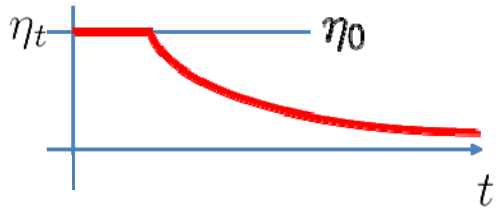
$$r = \sqrt{6/(\text{fan-in} + \text{fan-out})}$$

- **Unsupervised pre-training**

Step size

- **Fixed step-size**
  - try many, choose the best...
  - pick size with least test error on a validation set after T iterations

- **Dynamic step size**
  - decrease after T iterations



  - if simply the objective is not decreasing much, cut step by half

Momentum/L2 regularization

Modify stochastic/batch gradient descent:

$$\text{Before}: \quad \Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} L(\mathbf{w}), \quad w = w - \Delta \mathbf{w}$$

$$\text{With momentum}: \Delta \mathbf{w} = \mu \Delta \mathbf{w}_{previous} + \eta \nabla_{\mathbf{w}} L(\mathbf{w}), \quad w = w - \Delta \mathbf{w}$$

"Smooth" estimate of gradient from several steps of gradient descent:

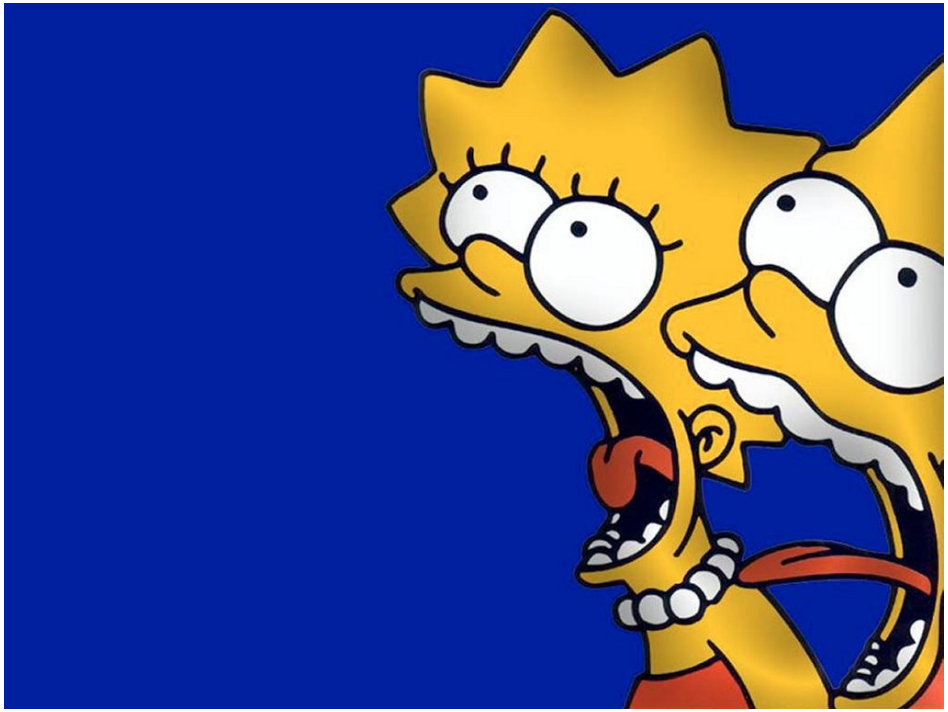
- High-curvature directions cancel out, low-curvature directions "add up" and accelerate.

Other techniques: Adagrad, Adadelta…

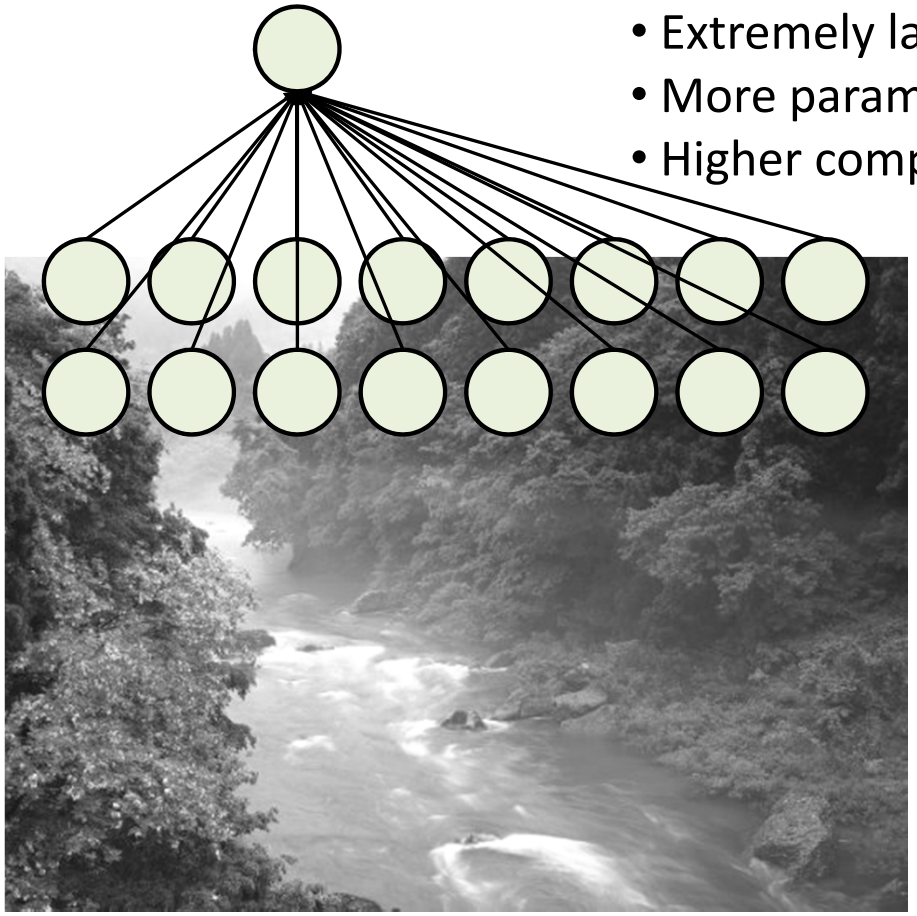
Add **L2 regularization** to the loss function:

$$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} (L(\mathbf{w}) + \lambda \| \mathbf{w} \|_2)$$

Yet, things will not still work well!



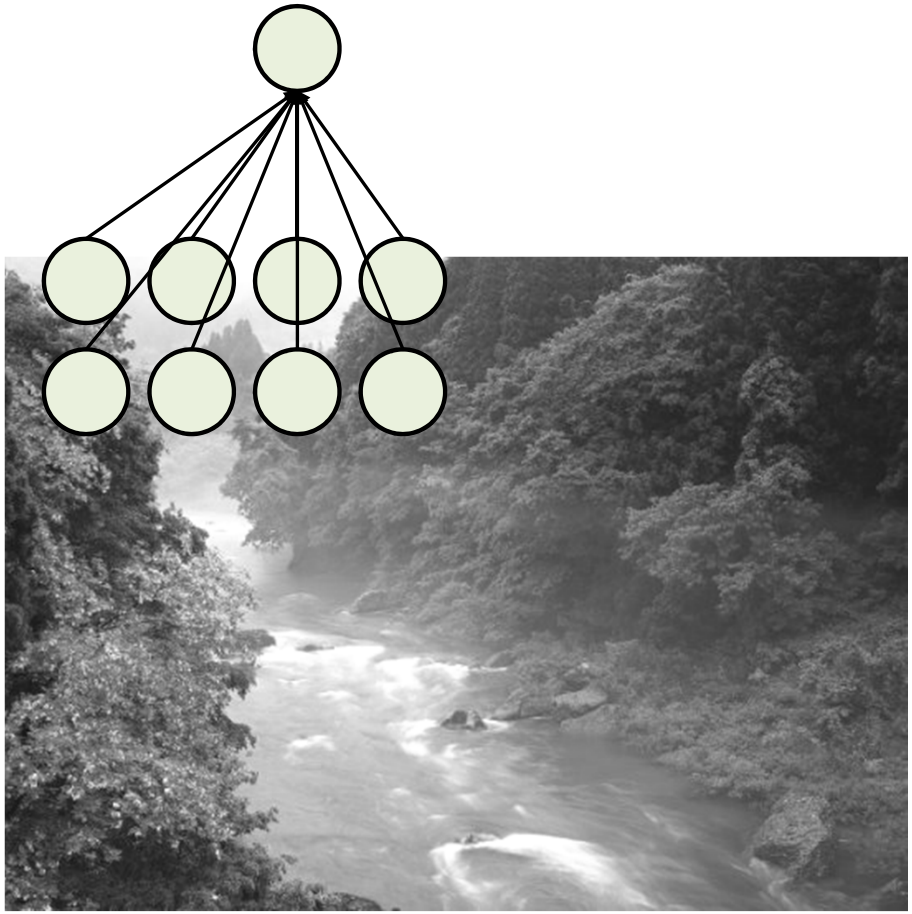
Main problem



- Extremely large number of connections.
- More parameters to train.
- Higher computational expense.

modified slides originally by Adam Coates

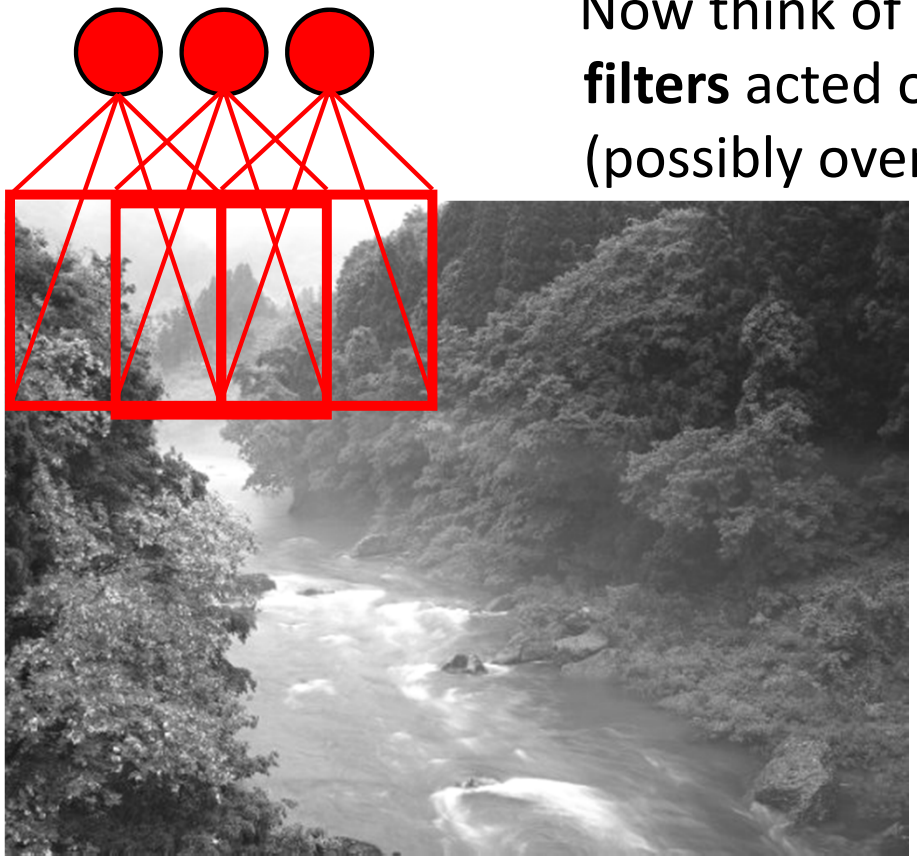
Local connectivity



Reduce parameters with
local connections!

modified slides originally by Adam Coates

Neurons as convolution filters

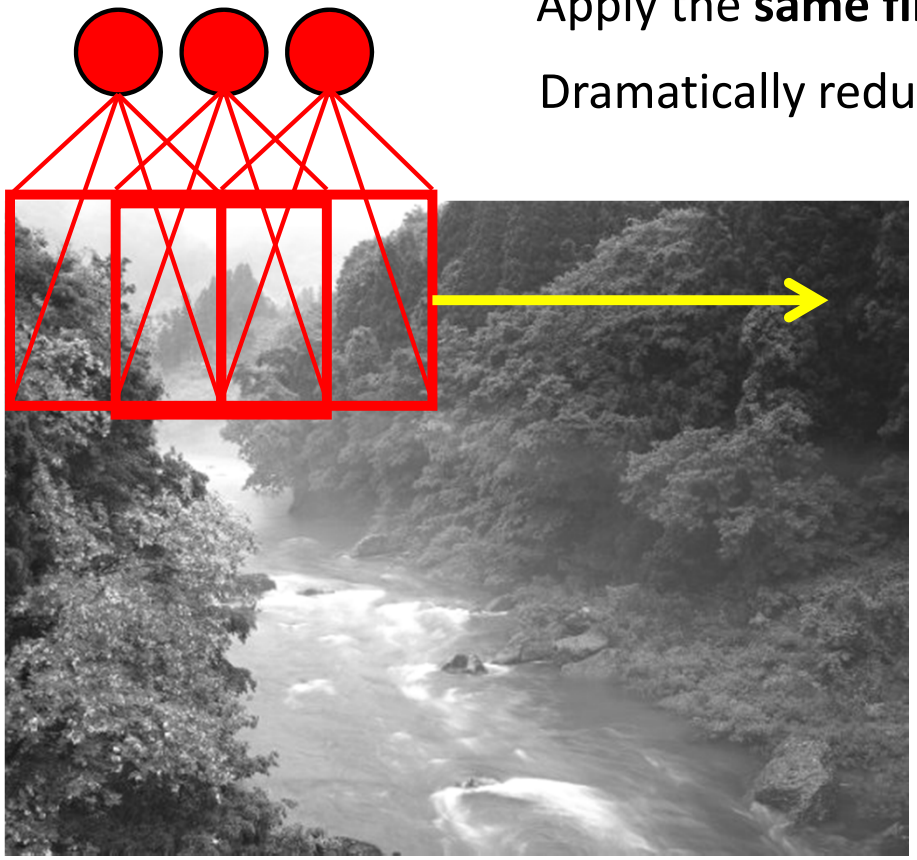


Now think of neurons as convolutional **filters** acted on small adjacent (possibly overlapping) windows

Window size is called **“receptive field” size** and spacing is called **“step” or “stride”**

modified slides originally by Adam Coates

Extract repeated structure



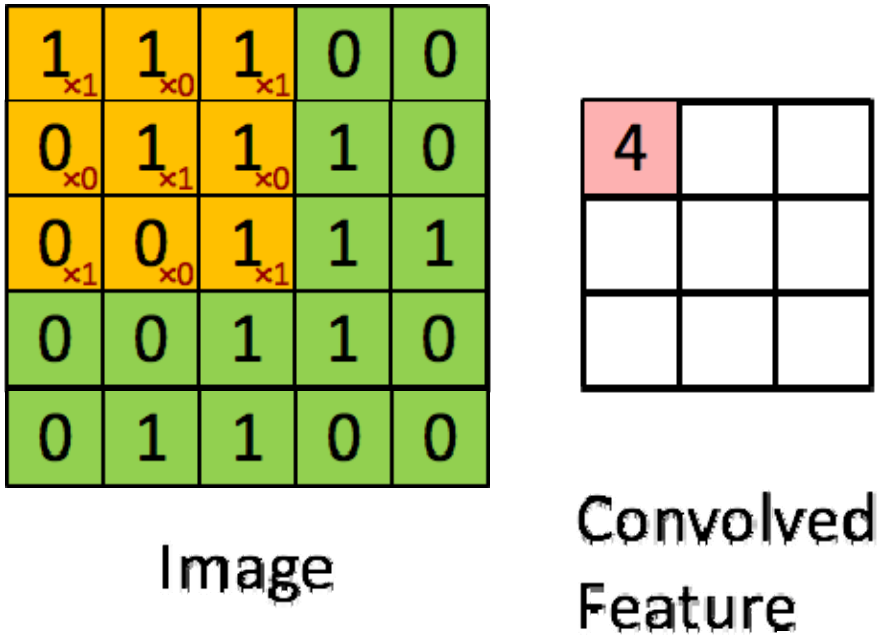
Apply the **same filter** (weights) throughout the image

Dramatically reduces the number of parameters

modified slides originally by Adam Coates

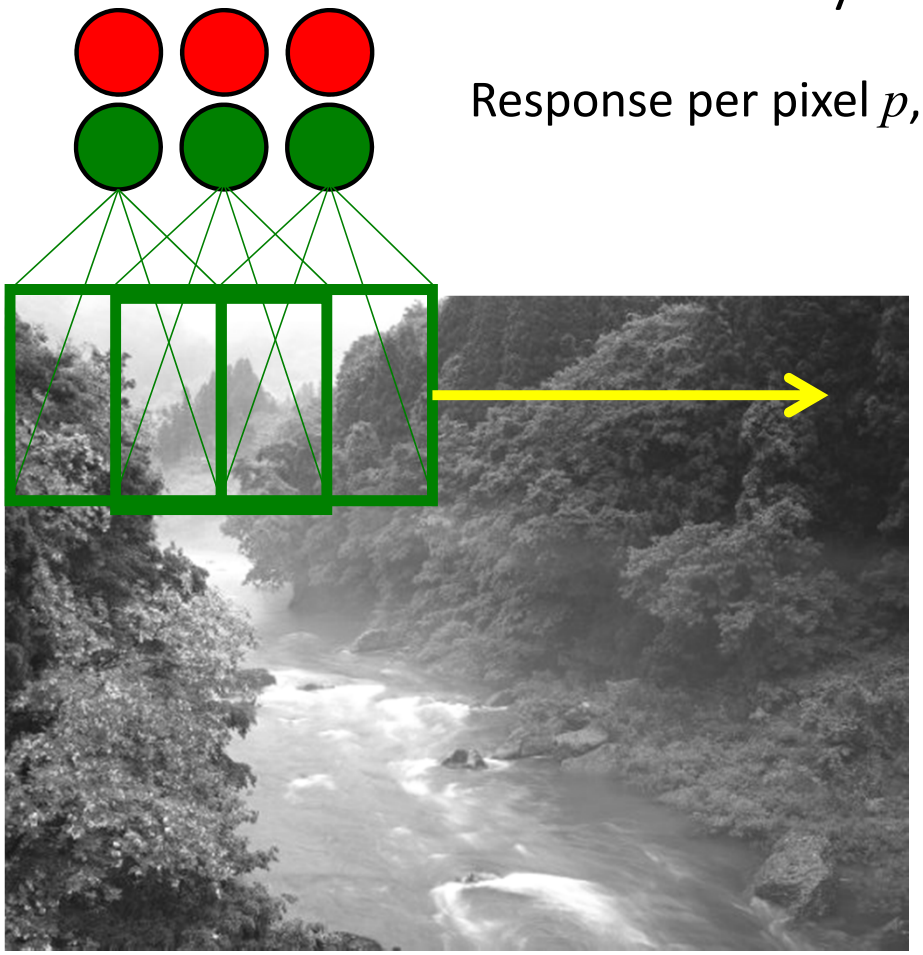
Convolution reminder [animated]

(red numbers are filter values)



Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Can have many filters!

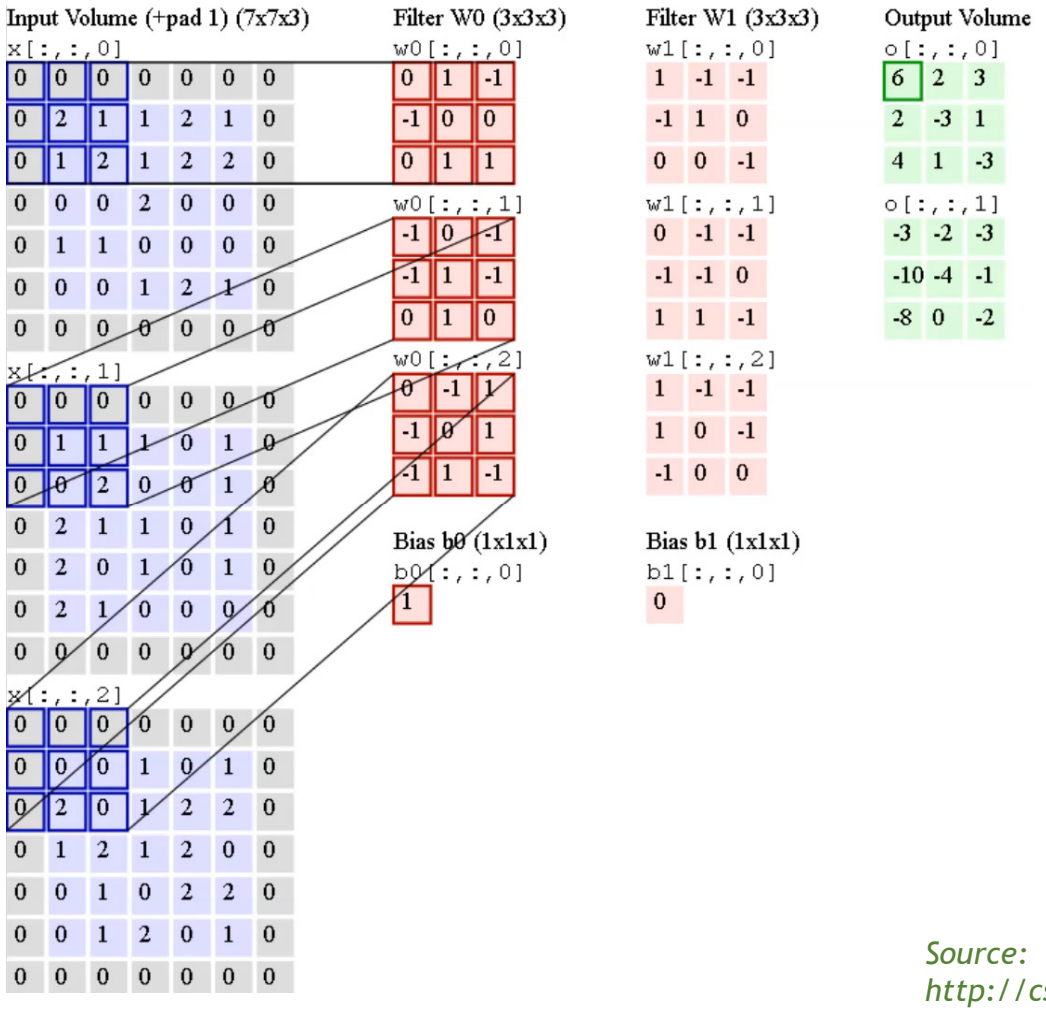


Response per pixel p, per filter f for a transfer function g:

$$h_{p,f} = g(\mathbf{w}_f \cdot \mathbf{x}_p)$$

modified slides originally by Adam Coates

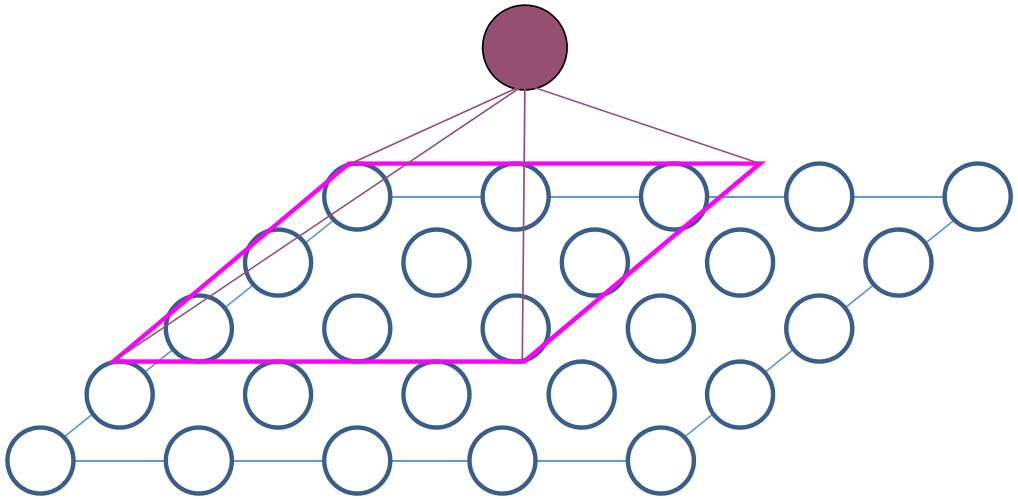
Example: multiple 3D filters working on multiple channels



Source:
http://cs231n.github.io/convolutional-networks/

Pooling

Apart from hidden layers dedicated to convolution, we can have layers dedicated to extract **locally invariant** descriptors



Max pooling:

$$h_{p',f} = \max_{p}(\mathbf{x_p})$$

Mean pooling:

$$h_{p',f} = \underset{p}{avg}(\mathbf{x_p})$$

Fixed filter (e.g., Gaussian):

$$h_{p',f} = w_{gaussian} \cdot \mathbf{x_p}$$

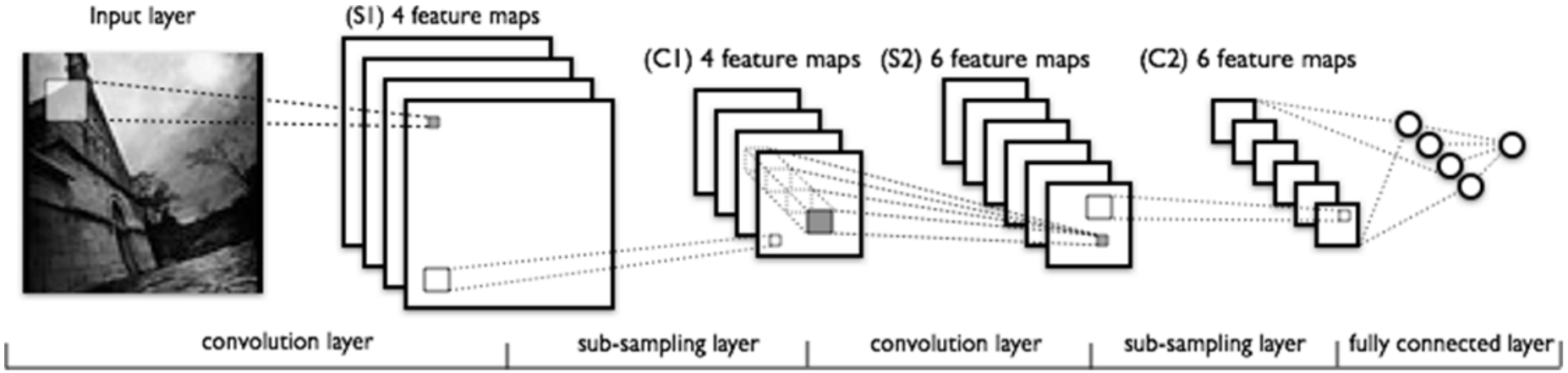
Progressively reduce the resolution of the image, so that the next convolutional filters are applied on larger scales

[Scherer et al., ICANN 2010]
[Boureau et al., ICML 2010]

A mini convolutional neural network

Interchange convolutional and pooling (subsampling) layers.

In the end, **unwrap all feature maps into a single feature vector** and pass it through the classical (**fully connected**) neural network.

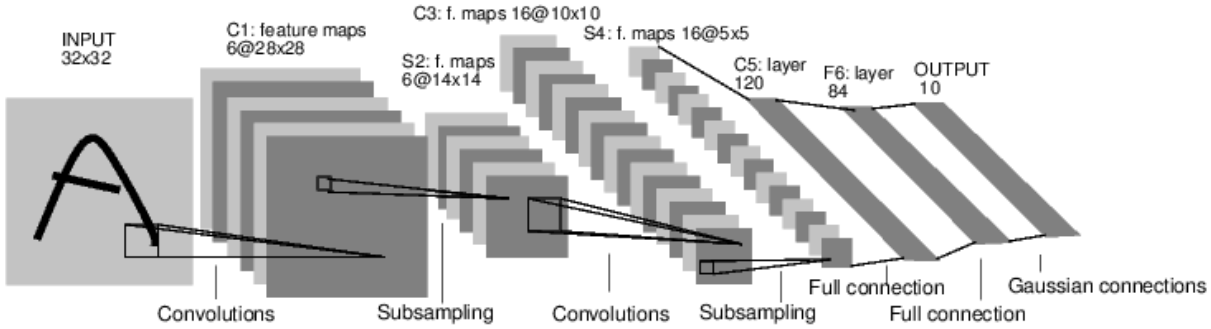


Source: http://deeplearning.net/tutorial/lenet.html

LeNet

Initial architecture from LeCun et al., 1998:

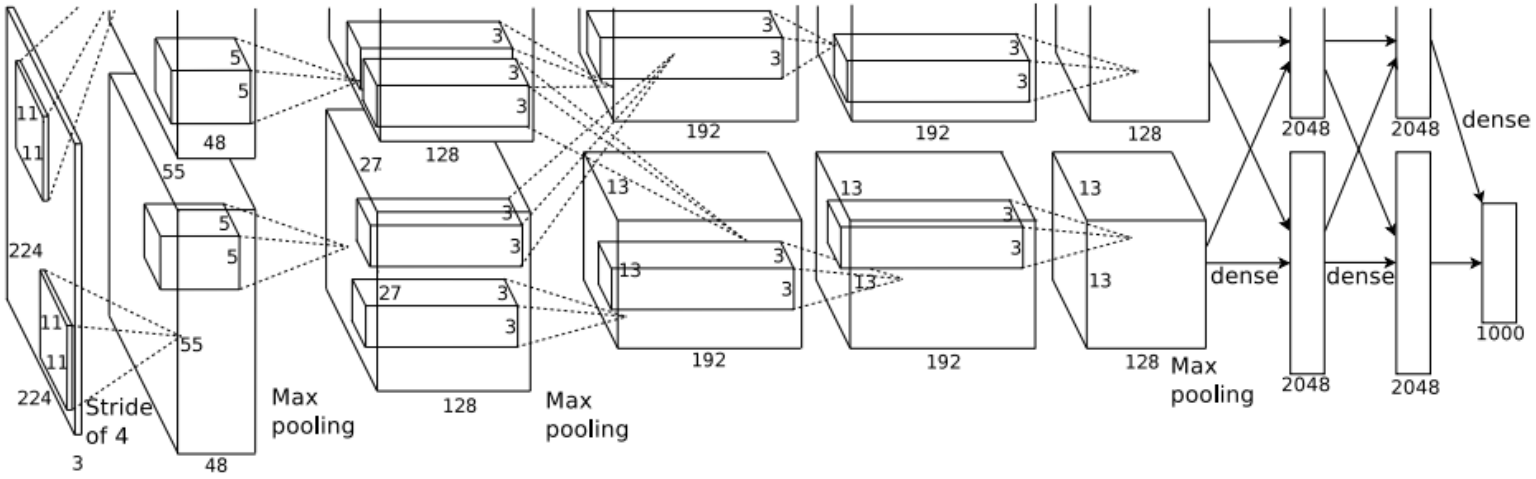
- Convolutional layers with tanh non-linearity
- Max-pooling layers
- Stochastic gradient descent
- Applied to digit recognition



AlexNet

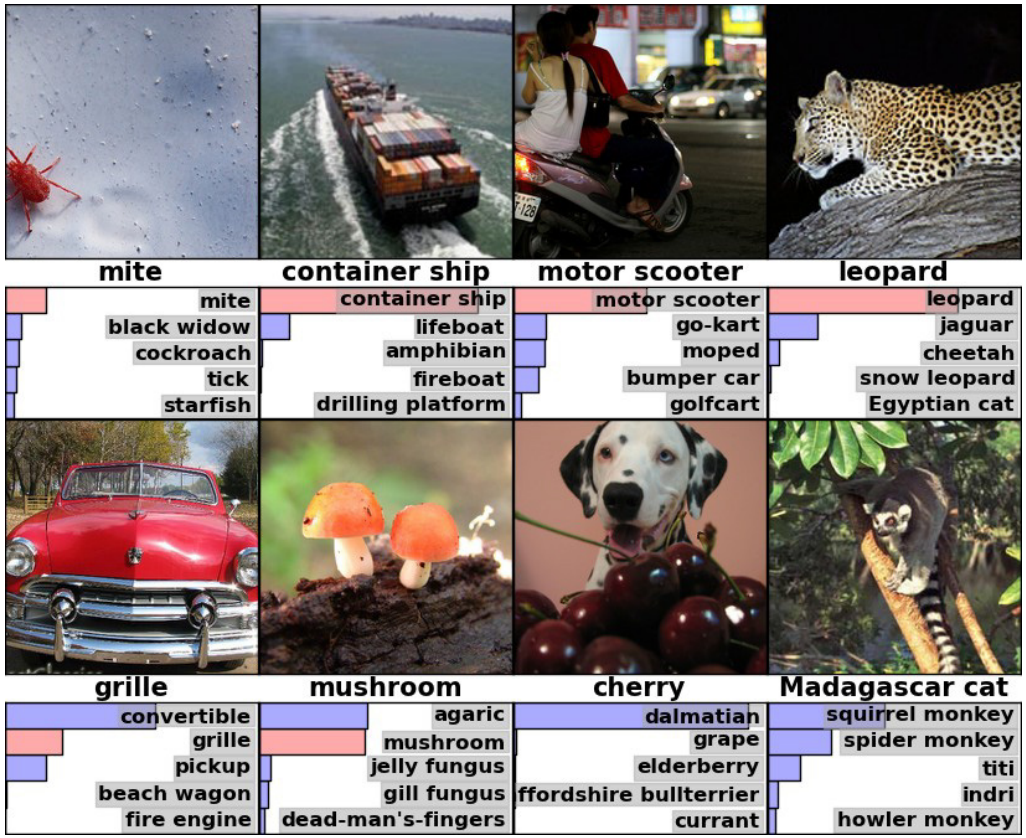
Proposed architecture from Krizhevsky et al., NIPS 2012:

- Convolutional layers with Rectified linear units
- Max-pooling layers
- Stochastic gradient descent on GPU with momentum, L2 regularization, dropout
- Applied to image classification (ImageNet competition – top runner & game changer)



Application: ImageNet classification

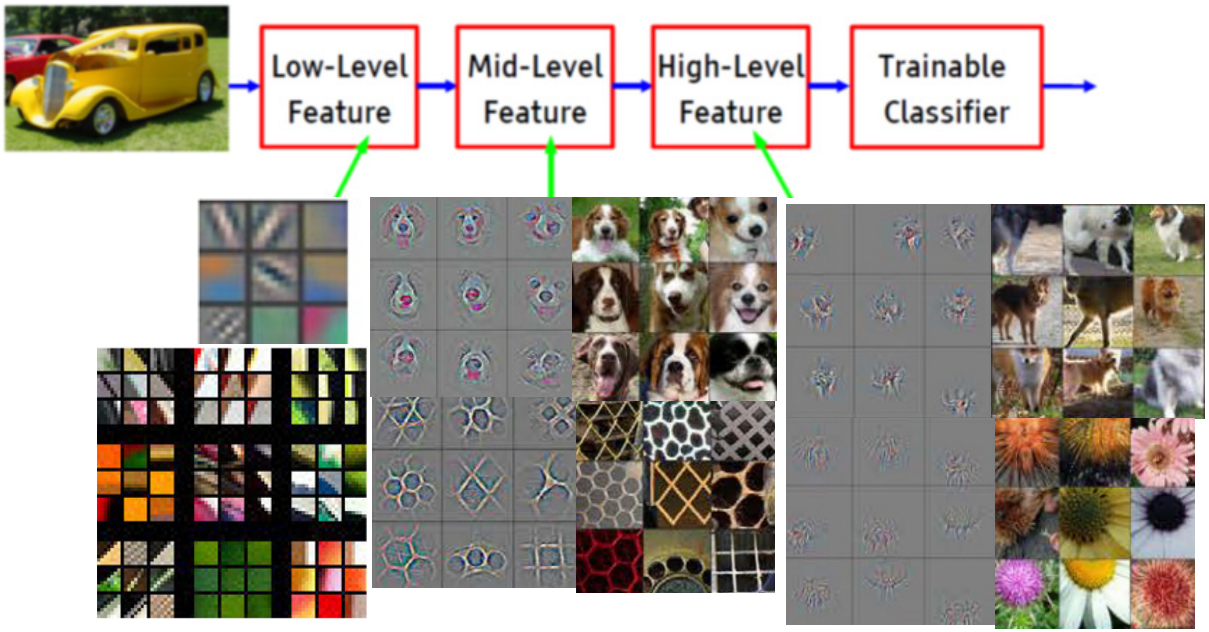
Top result in ILSVRC 2012 [~85%, Top-5 accuracy]



Krizhevsky et al., NIPS 2012

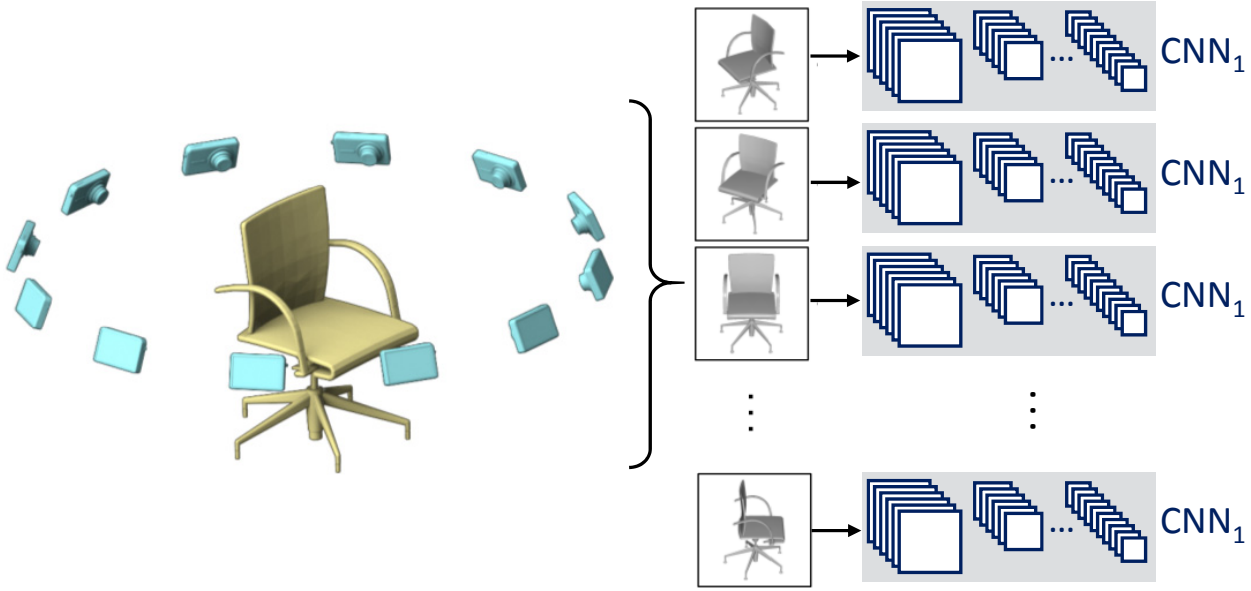
Learned representations

Think of convolution filters as optimized **feature templates capturing various hierarchical patterns** (edges, local structures, sub-parts, parts…)



see Matthew D. Zeiler and Rob Fergus, Visualizing and Understanding Convolutional Networks, 2014

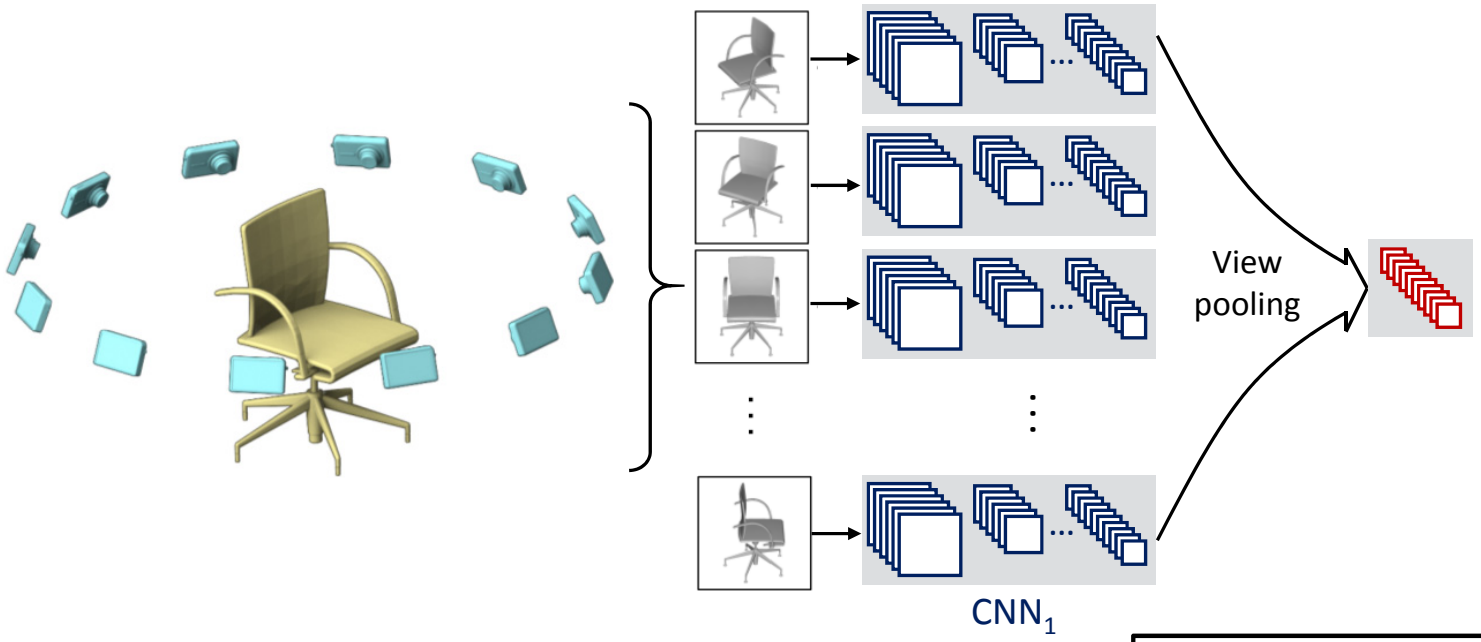
Multi-view CNNs for shape analysis



CNN_1: a ConvNet extracting image features

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015

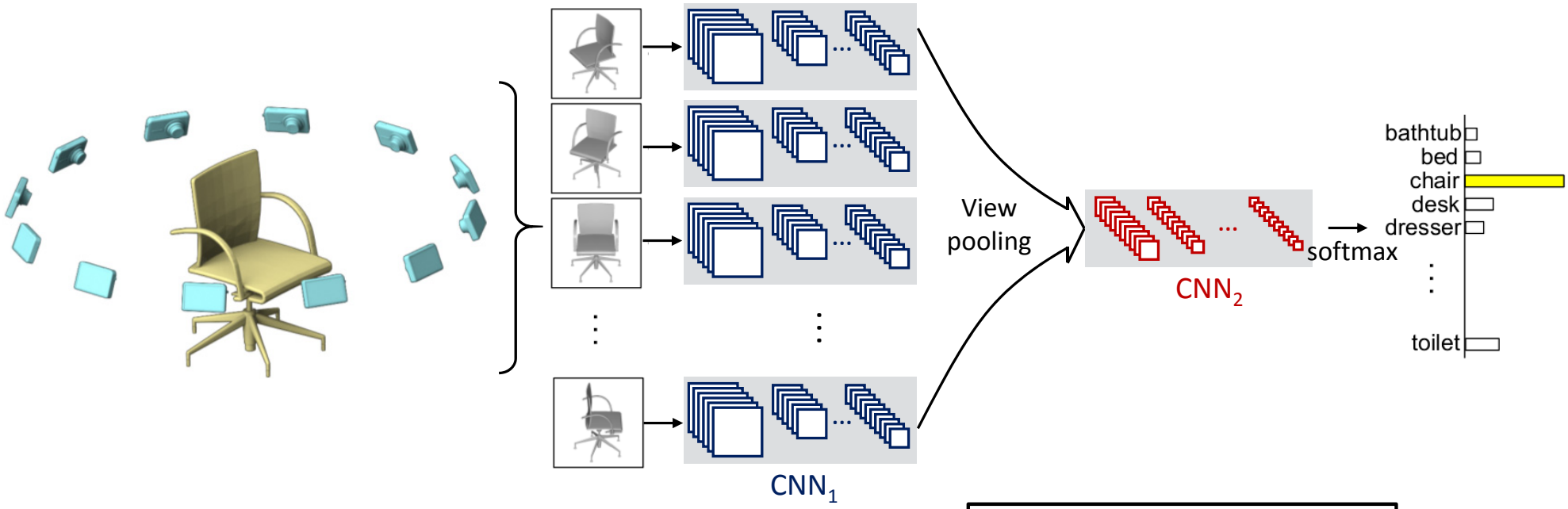
All image features are combined by view pooling…



View pooling: element-wise max-pooling across all views

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015

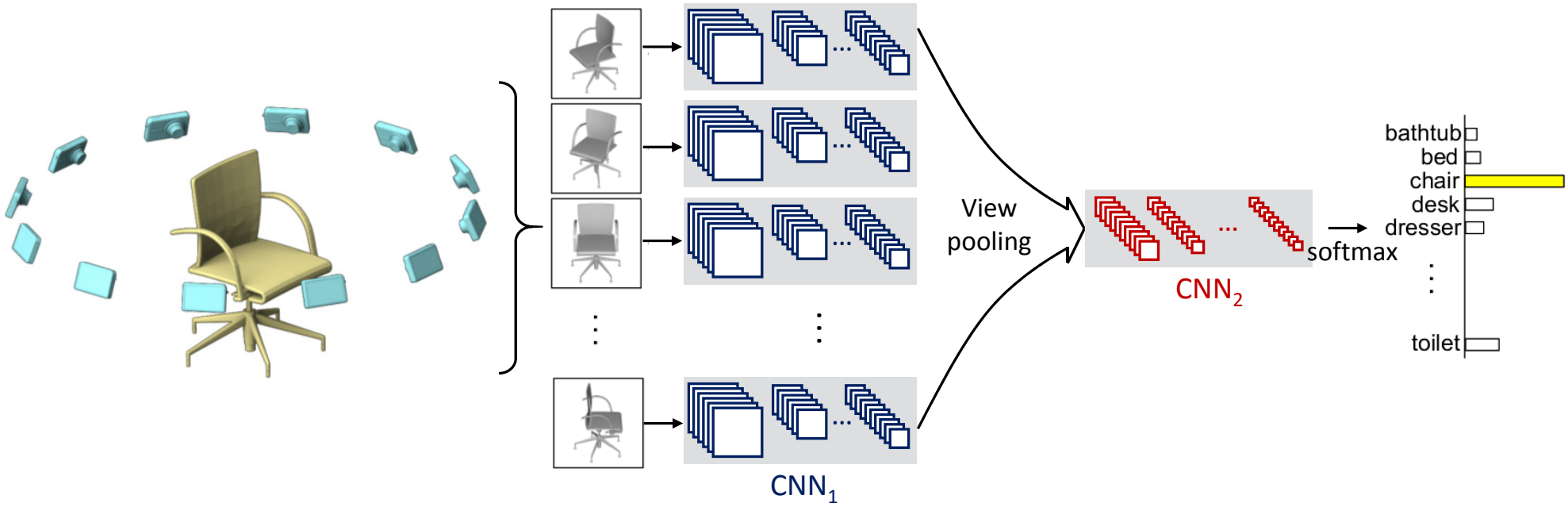
… then passed through CNN_2 and to generate final prediction



CNN_2: a second ConvNet producing shape descriptors

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015

Train on image datasets!



CNNs pre-trained on ImageNet **(leverage large image datasets for training shape analysis techniques!)**

Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015

… and then fine-tune on 3D datasets!

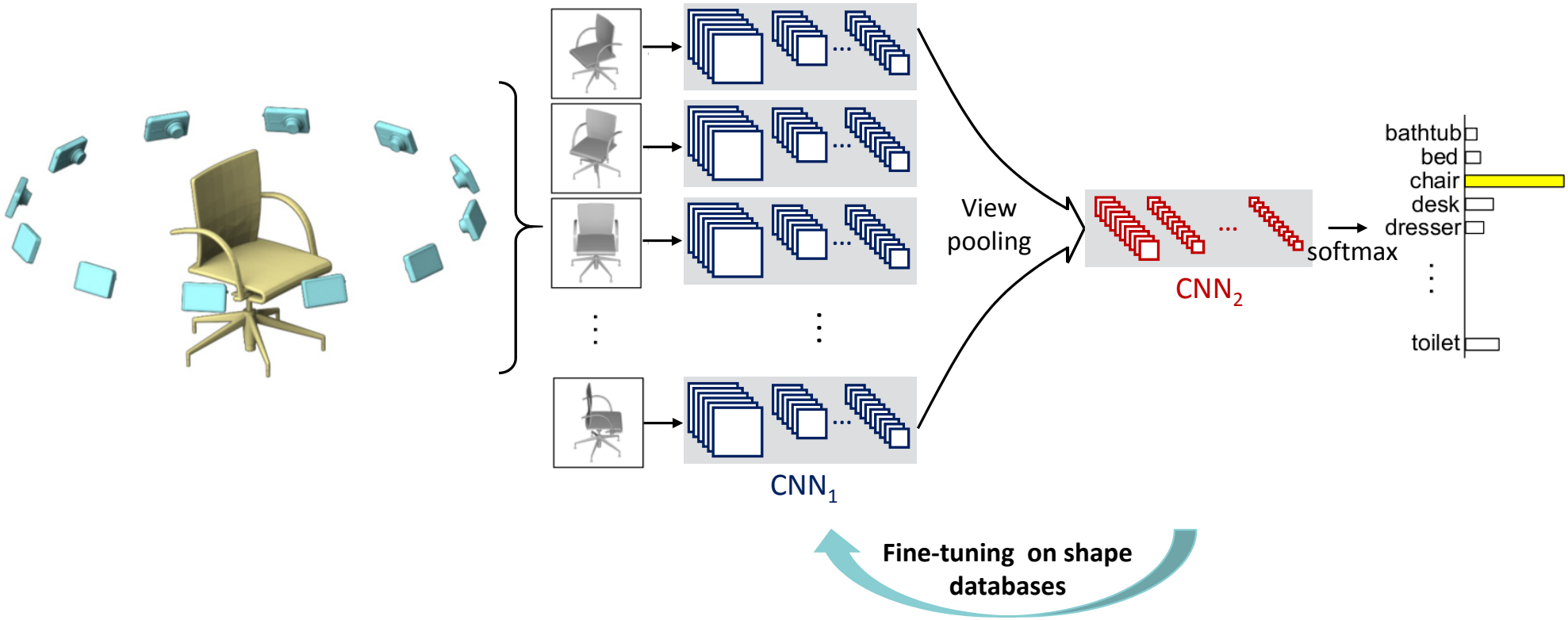


Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller Multi-view Convolutional Neural Networks for 3D Shape Recognition, ICCV 2015

Volumetric CNNs

Key idea: represent a shape as a volumetric image with binary voxels.

Learn filters operating on these volumetric data.

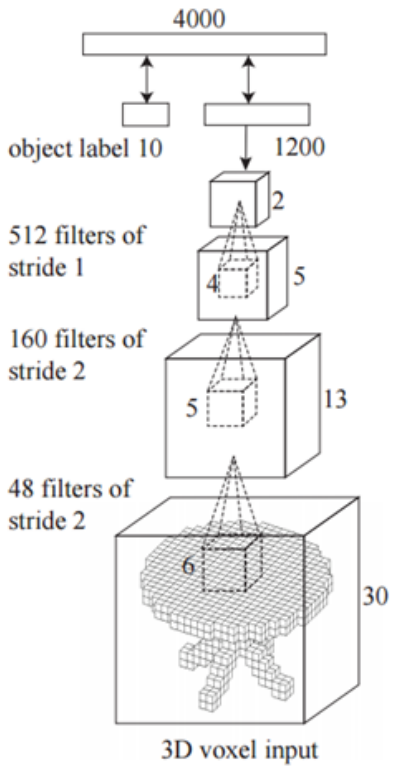


Image from Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao
3D ShapeNets: A Deep Representation for Volumetric Shapes, 2015

Volumetric CNNs

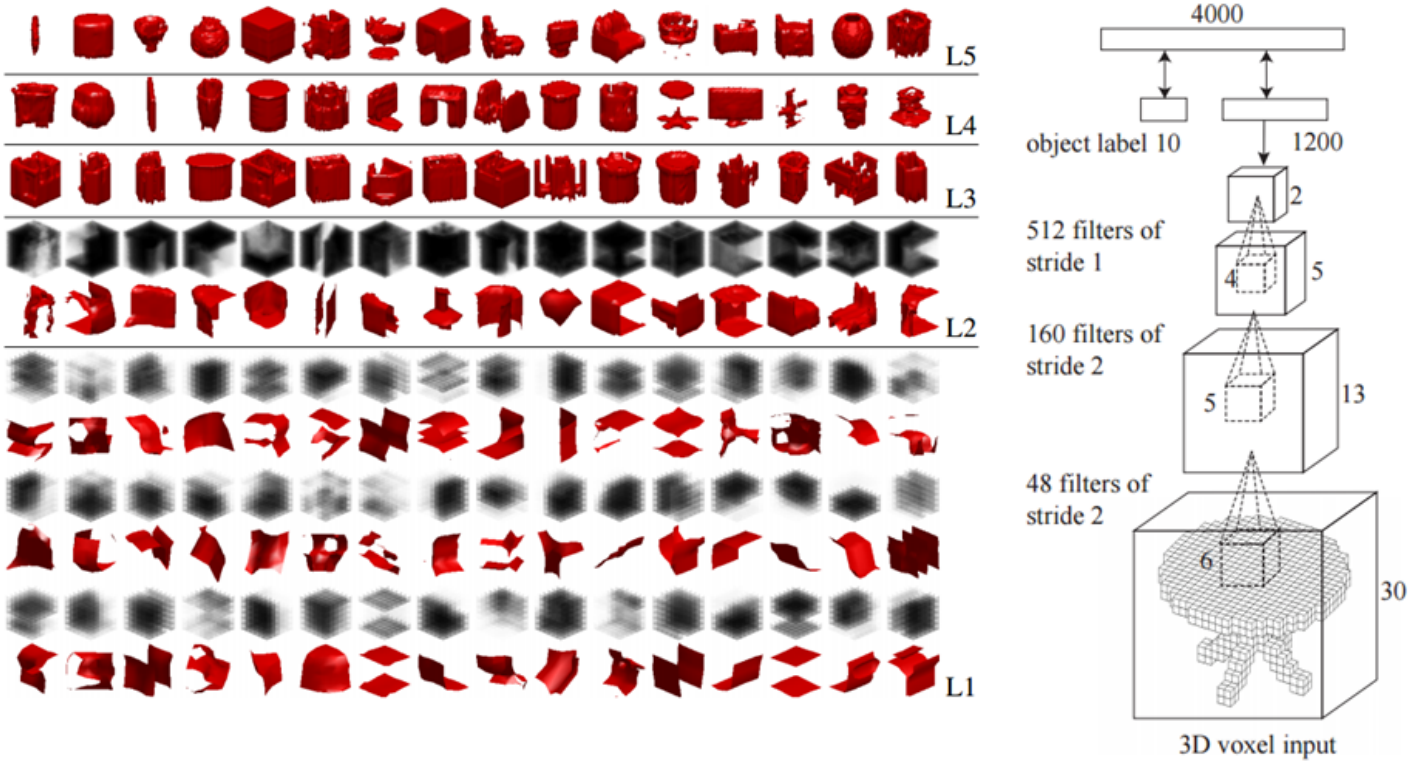


Image from Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao
3D ShapeNets: A Deep Representation for Volumetric Shapes, 2015

Comparison

Shape retrieval evaluation in ModelNet40:

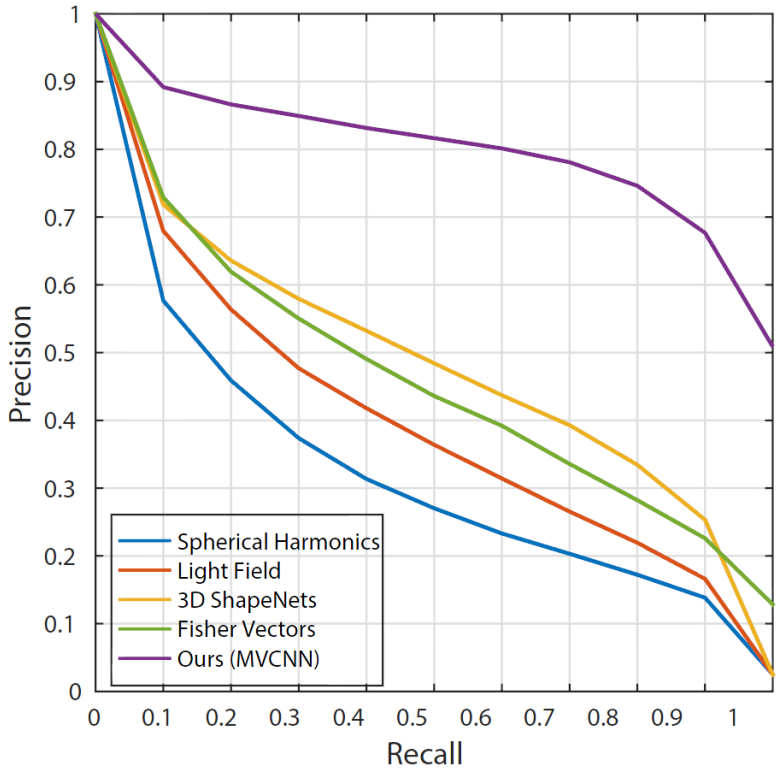


Image from Hang Su, Subhransu Maji, Evangelos Kalogerakis, Erik Learned-Miller, Multi-view Convolutional Neural Networks for 3D Shape Recognition, 2015

Summary

CNNs can learn **highly discriminative, hierarchical, powerful feature representations** for image and shape analysis.

Deep learning and CNNs have revolutionized computer vision, robotics, NLP, machine learning: **solve hard tasks, achieve performance comparable to humans**.

Why do we still use far-from-optimal, 'old-style' descriptors in CG?

Deep learning has also shown very promising results in image and shape synthesis [see **Data-Driven Shape Analysis and Processing, EG'16 STAR report**].