
CS688: Web-Scale Image Search
Deep Neural Nets and
Features

Sung-Eui Yoon
(윤성익)

Course URL:
<http://sglab.kaist.ac.kr/~sungeui/IR>

KAIST

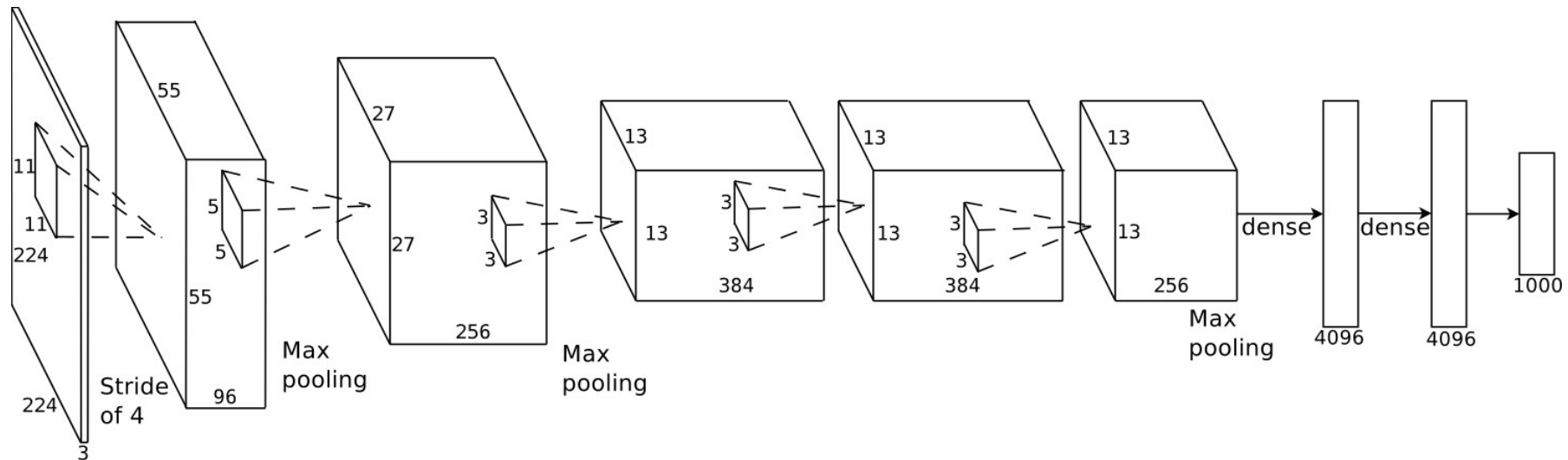


Class Objectives

- **Study neural nets, especially, convolution neural nets (CNNs)**
- **See its applications to computer vision problems and image search**

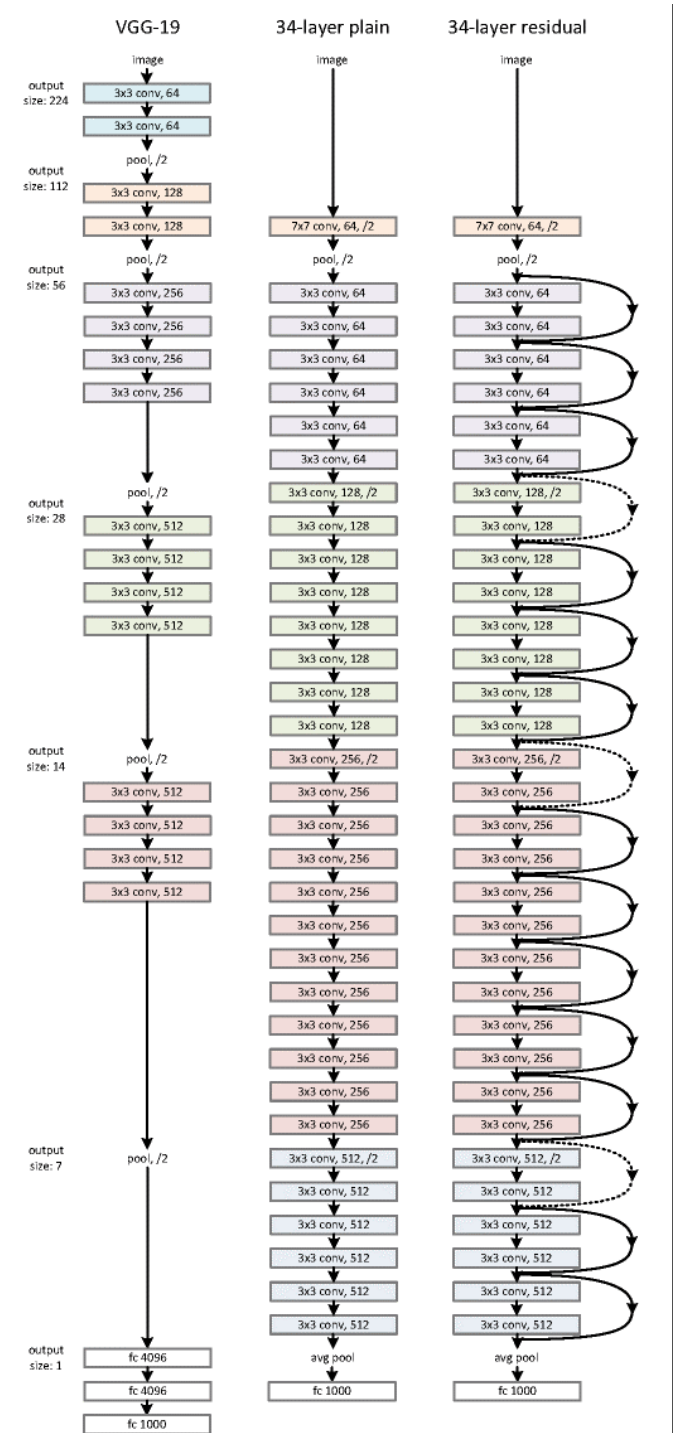
High-Level Messages

- Deep neural nets provide low-level and high-level features
 - We can use those features for image search
- Achieve the best results in many computer vision related problems



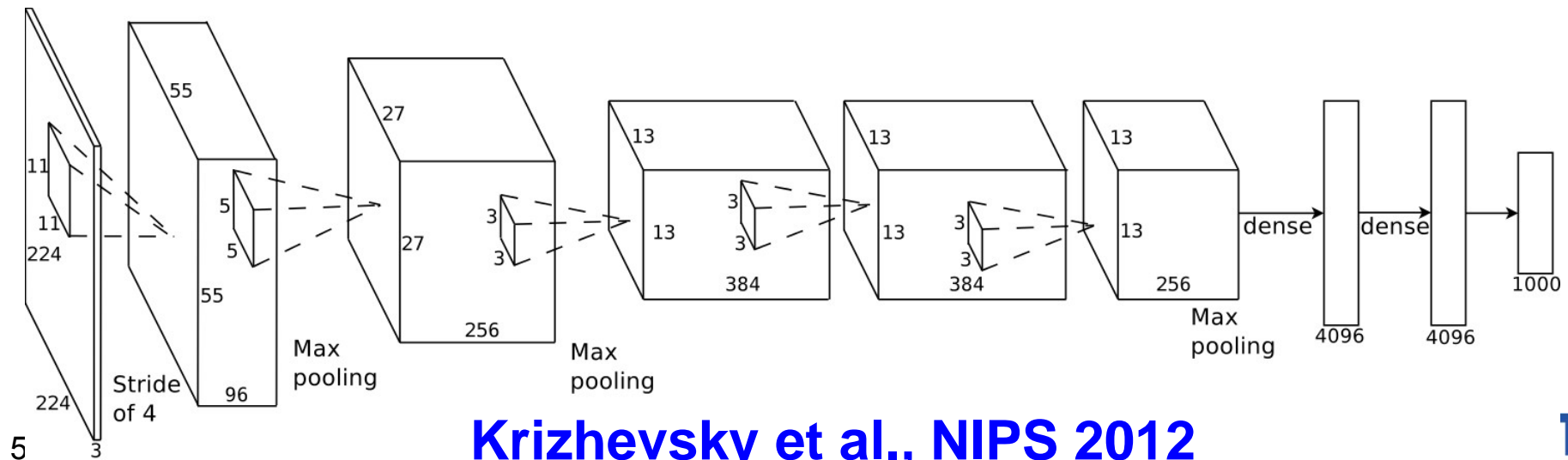
High-Level Messages

- Many features and codes are available
 - Caffe [Krizhevsky et al., NIPS 2012]
 - Very deep convolutional networks [Simonyan et al., ICLR 15]; using up to 19 layers
 - Deep Residual Learning [He et al., CVPR 16]; using up to 152 layers
- Model Zoo
github.com/BVLC/caffe/wiki/Model-Zoo



High-Level Messages

- Perform the end-to-end optimization w/ lots of training data
 - Aims not only features, but the accuracy of any end-to-end systems including image search



Deep Learning for Vision

Adam Coates

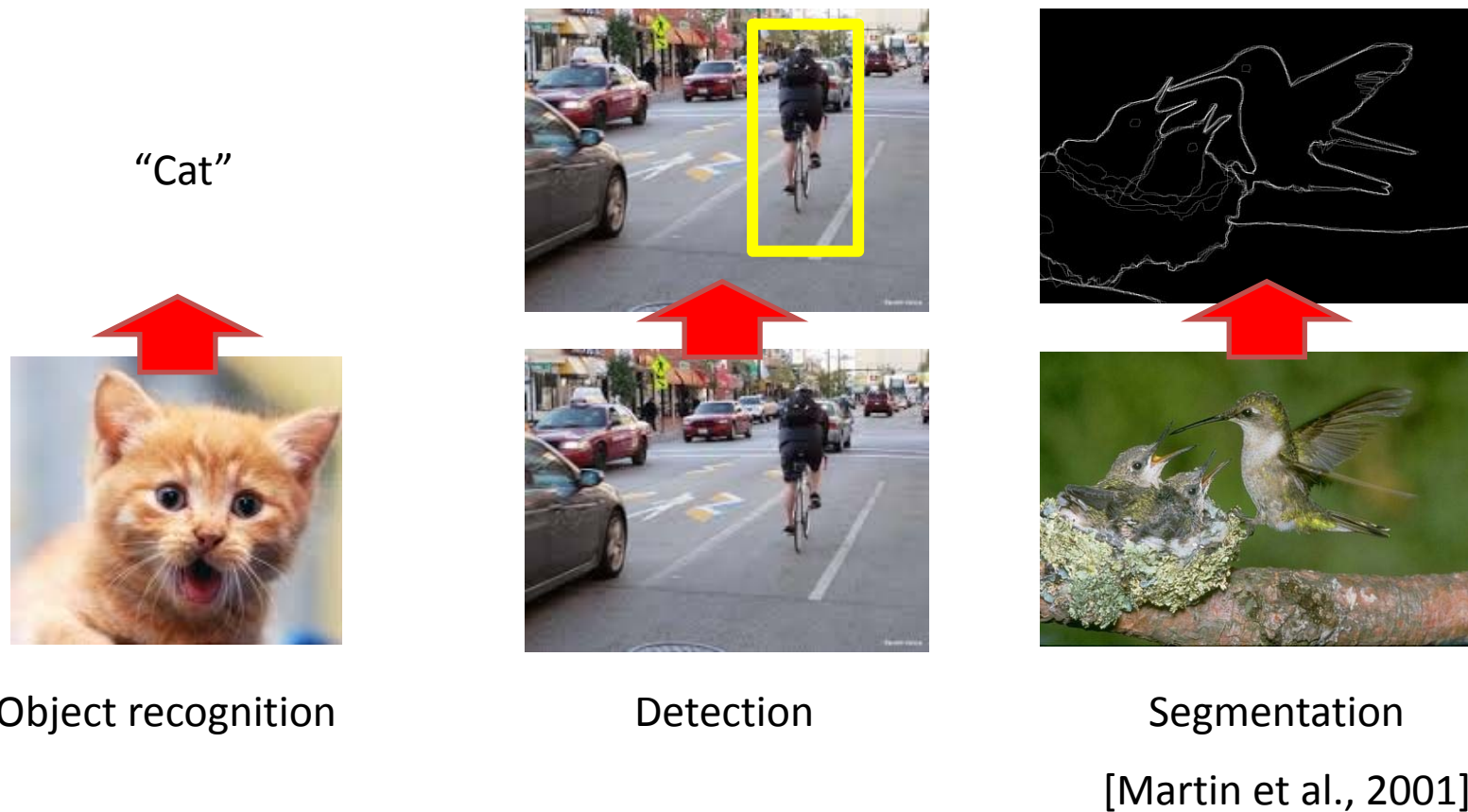
Stanford University

(Visiting Scholar: Indiana University, Bloomington)



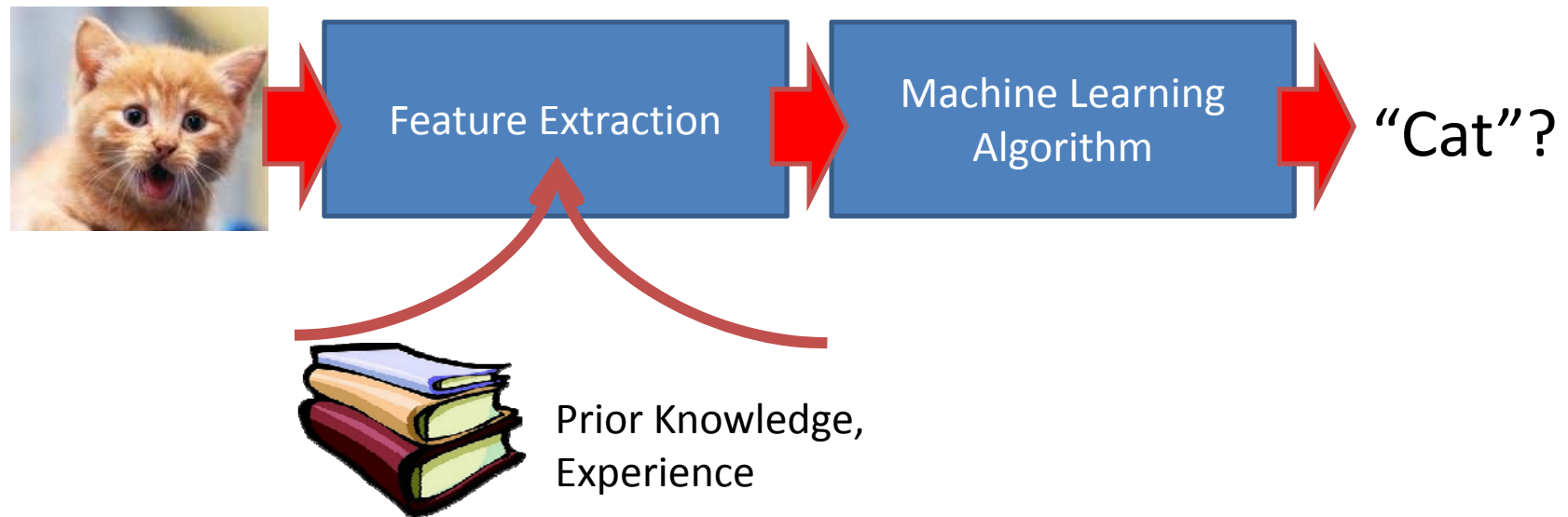
What do we want ML to do?

- Given image, predict complex high-level patterns:



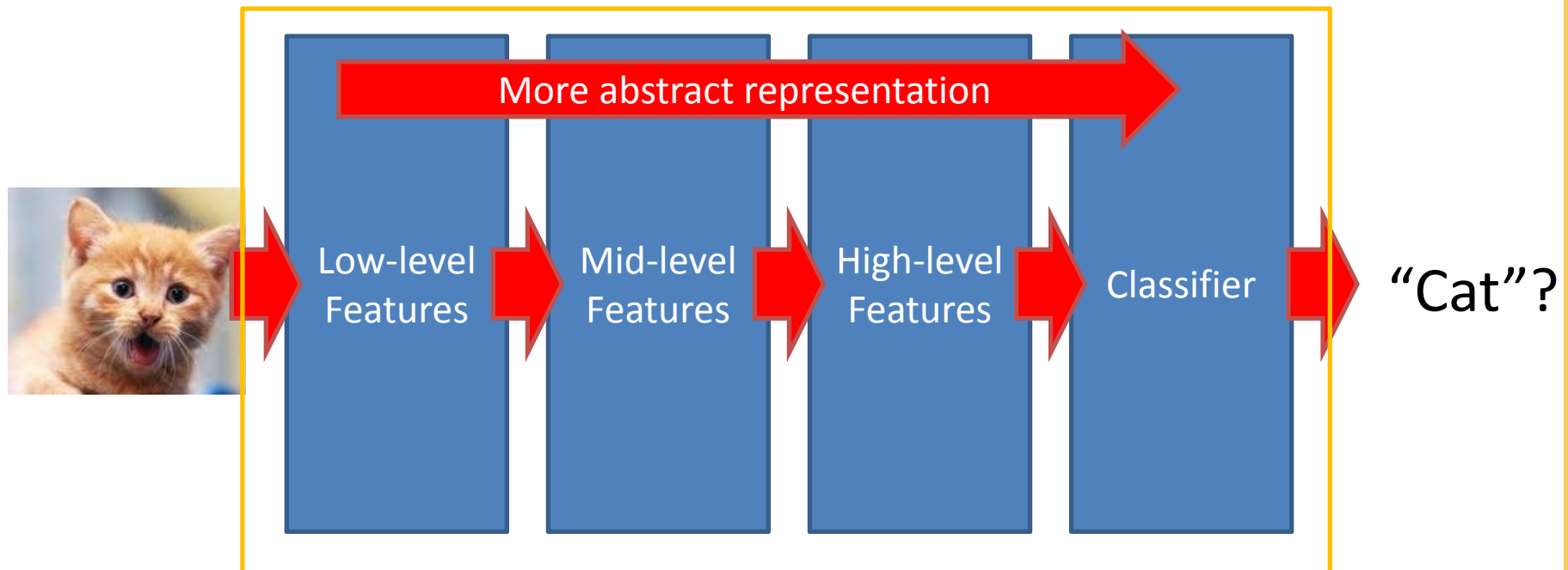
How is ML done?

- Machine learning often uses hand-designed feature extraction.



“Deep Learning”

- Deep Learning
 - Train *multiple layers* of features from data.
 - Try to discover useful *representations*



“Deep Learning”

- Why do we want “deep learning”?
 - Some decisions require many stages of processing.
 - We already hand-engineer “layers” of representation.
 - Algorithms scale well with data and computing power.
 - In practice, one of the most consistently successful ways to get good results in ML.

Have we been here before?

- Yes: Basic ideas common to past ML and neural networks research.
- No.
 - Faster computers; more data.
 - Better optimizers; better initialization schemes.
 - “Unsupervised pre-training” trick
[[Hinton et al. 2006](#); [Bengio et al. 2006](#)]
 - Lots of empirical evidence about what works.
 - Made useful by ability to “mix and match” components.
[See, e.g., [Jarrett et al., ICCV 2009](#)]

Real impact

- DL systems are high performers in many tasks over *many domains*.

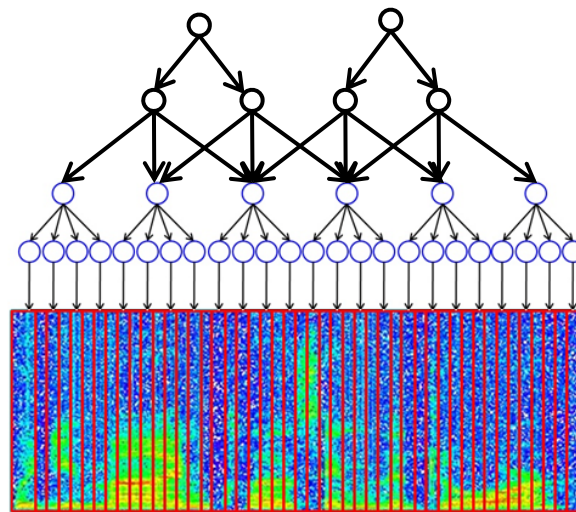


leopard



Image recognition

[E.g., Krizhevsky et al., 2012]

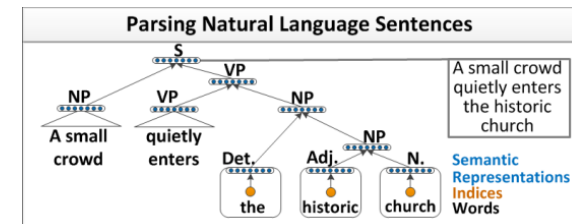


Spectrogram

[Honglak Lee]

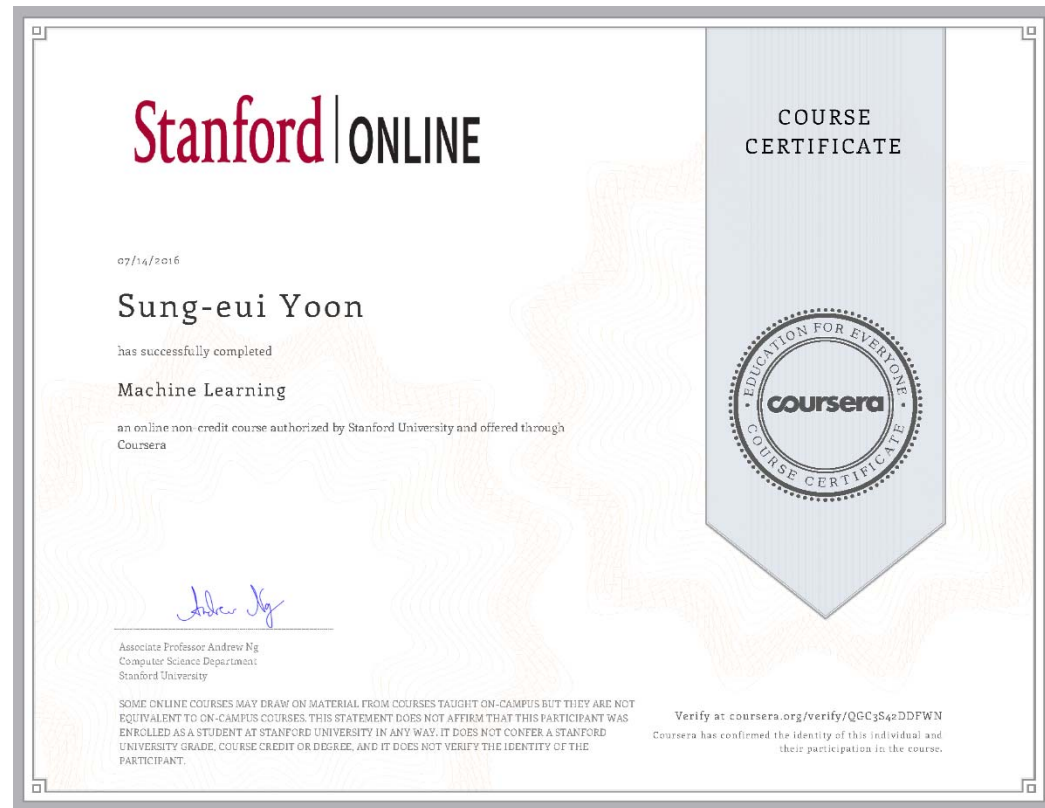
Speech recognition

[E.g., Heigold et al., 2013]



NLP

[E.g., Socher et al., ICML 2011; Collobert & Weston, ICML 2008]



Crash Course

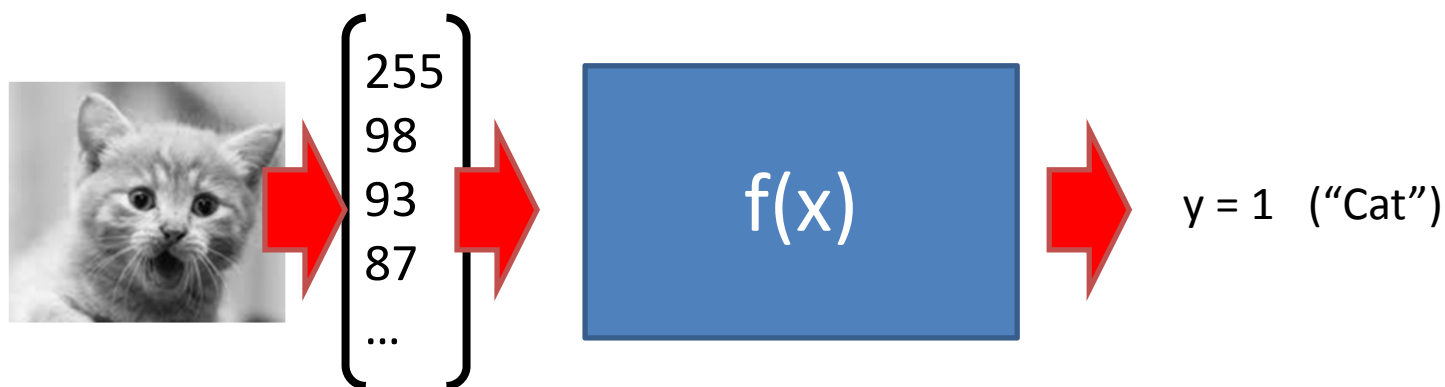
MACHINE LEARNING REFRESHER

Supervised Learning

- Given *labeled* training examples:

$$\mathcal{X} = \{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$$

- For instance: $x^{(i)}$ = vector of pixel intensities.
 $y^{(i)}$ = object class ID.



- Goal: find $f(x)$ to predict y from x on training data.
 - Hopefully: learned predictor works on “test” data.

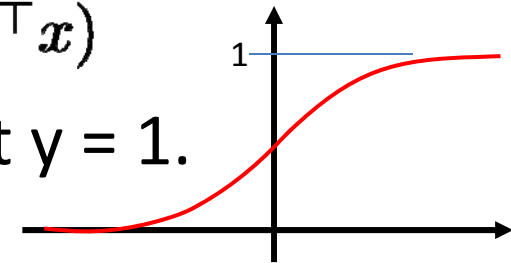
Logistic Regression

- Simple binary classification algorithm

- Start with a function of the form:

$$f(x; \theta) \equiv \sigma(\theta^T x) = \frac{1}{1 + \exp(-\theta^T x)}$$

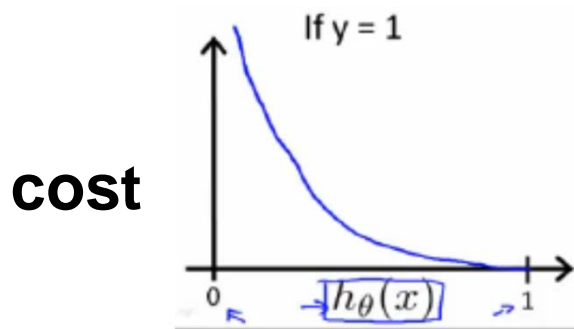
- Interpretation: $f(x)$ is probability that $y = 1$.



- Find choice of θ that minimizes objective:

$$\mathcal{L}(\theta) = - \sum_i^n 1\{y^{(i)} = 1\} \log(f(x^{(i)}; \theta)) + 1\{y^{(i)} = 0\} \log(1 - f(x^{(i)}; \theta))$$

$\mathbb{P}(y^{(i)} = 1 | x^{(i)})$
 $\mathbb{P}(y^{(i)} = 0 | x^{(i)})$



From Ng's slide

Optimization

- How do we tune θ to minimize $\mathcal{L}(\theta)$?
- One algorithm: gradient descent
 - Compute gradient:

$$\nabla_{\theta} \mathcal{L}(\theta) = \sum_i^m x^{(i)} \cdot (y^{(i)} - f(x^{(i)}; \theta))$$

- Follow gradient “downhill”:

$$\theta := \theta - \eta \nabla_{\theta} \mathcal{L}(\theta)$$

- Stochastic Gradient Descent (SGD): take step using gradient from only small batch of examples.
 - Scales to larger datasets. [[Bottou & LeCun, 2005](#)]

Is this enough?

- Loss is convex \rightarrow we always find minimum.
- Works for simple problems:
 - Certain pixels are highly informative --- e.g., center pixel.



- Fails for even slightly harder problems.
 - Is this a coffee mug?



Why is vision so hard?



“Coffee Mug”

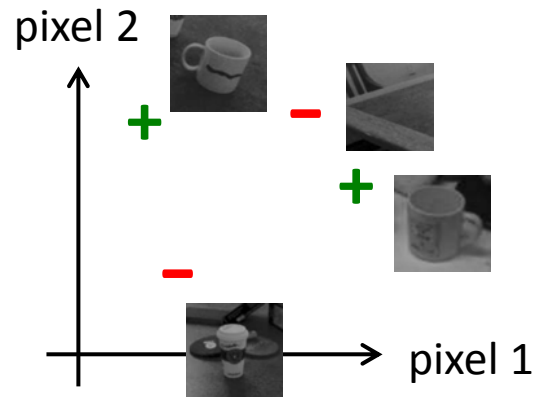
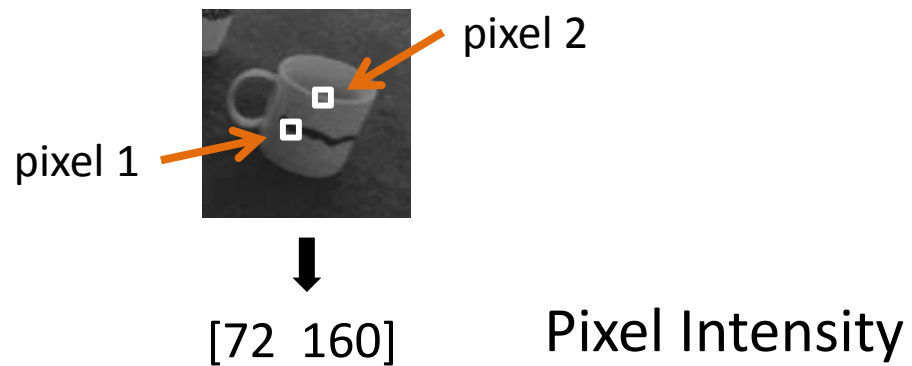


Pixel Intensity

177	153	118	91	85	100	124	145
151	124	93	77	86	115	148	168
115	93	78	83	108	145	177	191
88	79	84	104	136	168	190	197
82	85	103	127	152	170	180	182
91	101	120	138	150	157	159	159
103	114	127	136	140	140	140	141
111	119	126	130	130	129	128	130

Pixel intensity is a very poor representation.

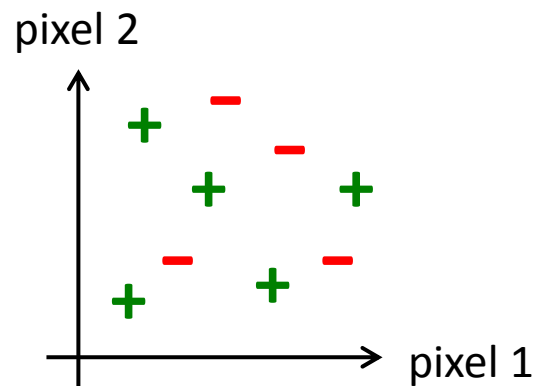
Why is vision so hard?



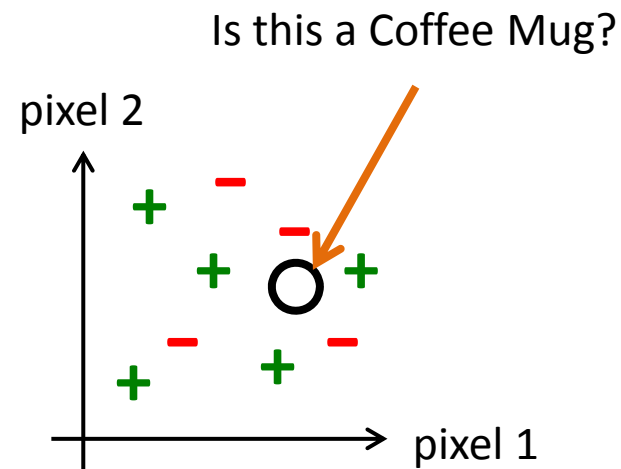
+ Coffee Mug

- Not Coffee Mug

Why is vision so hard?



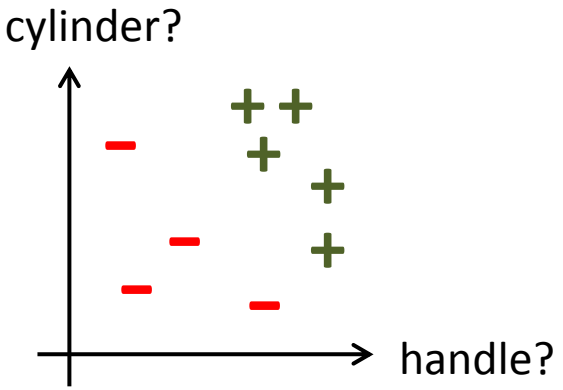
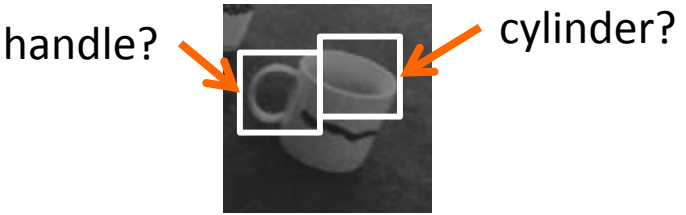
Learning Algorithm



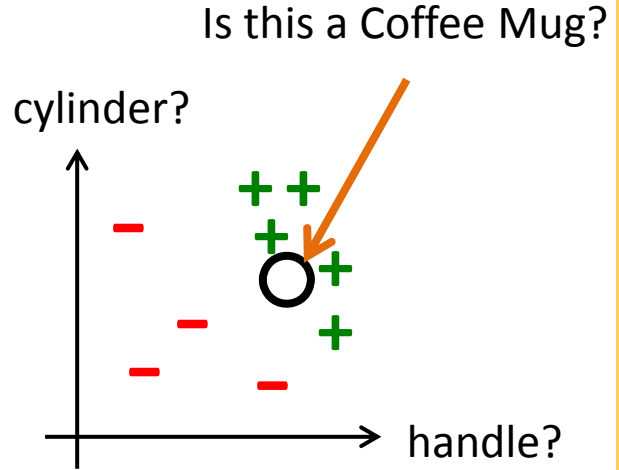
+ Coffee Mug

- Not Coffee Mug

Features



Learning Algorithm

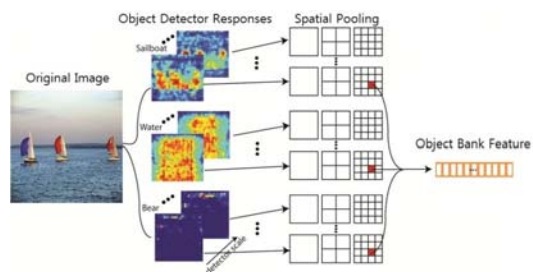


- + Coffee Mug
- Not Coffee Mug

Features

- Huge investment devoted to building application-specific feature representations.

Object Bank [Li et al., 2010]

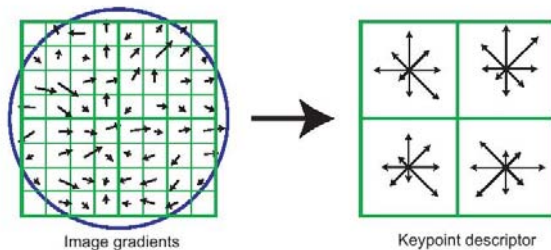


Super-pixels

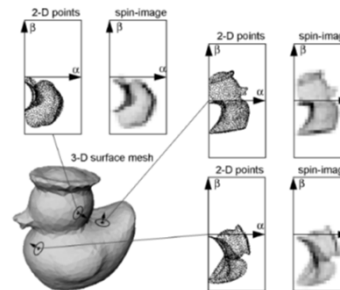
[Gould et al., 2008; Ren & Malik, 2003]



SIFT [Lowe, 1999]



Spin Images [Johnson & Hebert, 1999]



Extension to neural networks

SUPERVISED DEEP LEARNING



Basic idea

- We saw how to do supervised learning when the “features” $\phi(x)$ are fixed.
 - Let’s extend to case where features are given by tunable functions with their own parameters.

$$\mathbb{P}(y = 1|x) = f(x; \theta, W) = \sigma(\theta^\top \underbrace{\sigma(Wx)})$$

Outer part of function is same as logistic regression.

Inputs are “features”---one feature for each row of W:

$$\begin{bmatrix} \sigma(w_1x) \\ \sigma(w_2x) \\ \dots \\ \sigma(w_Kx) \end{bmatrix}$$

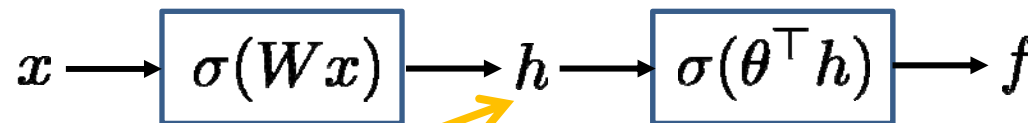
Basic idea

- To do supervised learning for two-class classification, minimize:

$$\mathcal{L}(\theta, W) = - \sum_i^m 1\{y^{(i)} = 1\} \log(f(x^{(i)}; \theta, W)) + \\ 1\{y^{(i)} = 0\} \log(1 - f(x^{(i)}; \theta, W))$$

- Same as logistic regression, but now $f(x)$ has multiple stages (“layers”, “modules”):

$$f(x; \theta, W) = \sigma(\theta^\top \sigma(Wx))$$

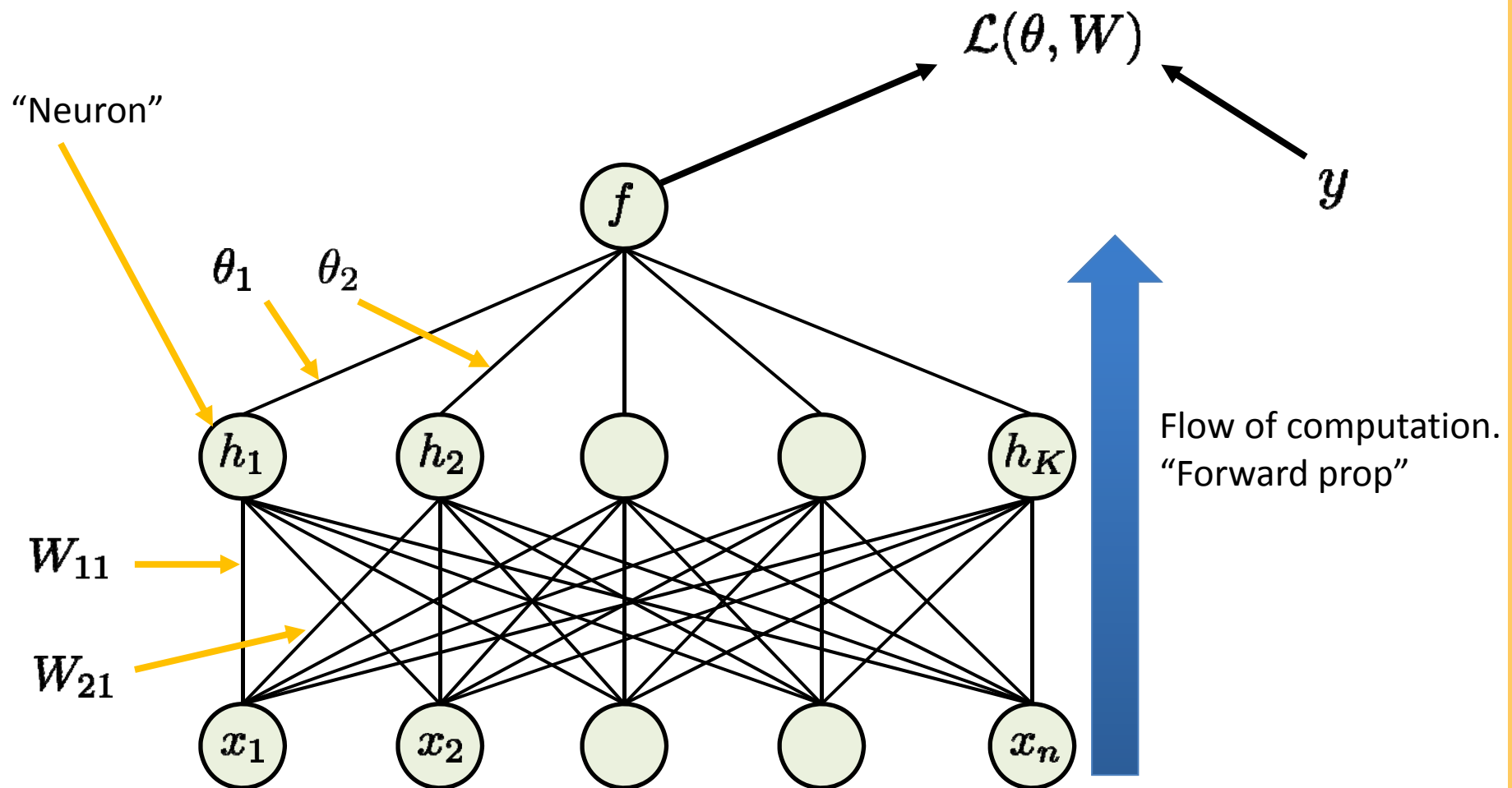


Intermediate representation (“features”)

Prediction for $\mathbb{P}(y = 1|x)$

Neural network

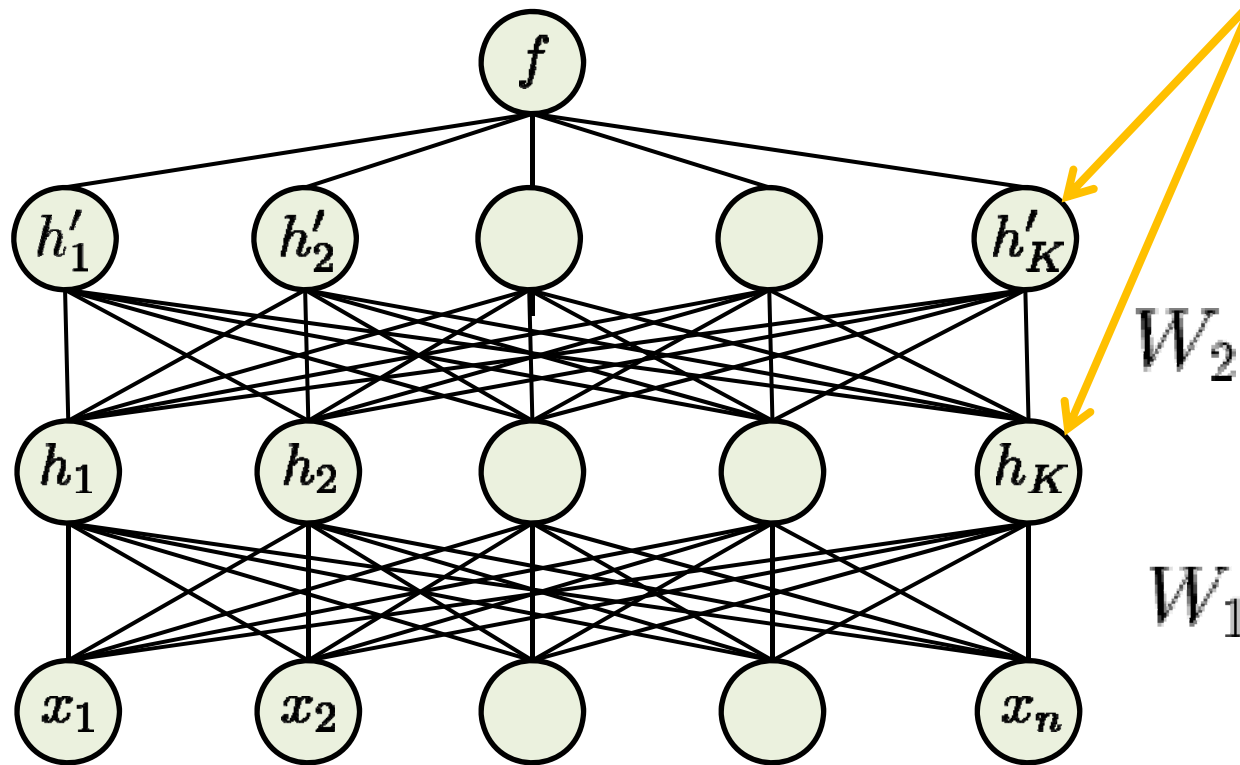
- This model is a sigmoid “neural network”:



Neural network

- Can stack up several layers:

Must learn multiple stages of internal “representation”.



$$x \rightarrow \sigma(W_1 x) \rightarrow h \rightarrow \sigma(W_2 h) \rightarrow h' \rightarrow \sigma(\theta^\top h') \rightarrow f$$

Back-propagation

- Minimize:

$$\mathcal{L}(\theta, W) = - \sum_i^m 1\{y^{(i)} = 1\} \log(f(x^{(i)}; \theta, W)) + \\ 1\{y^{(i)} = 0\} \log(1 - f(x^{(i)}; \theta, W))$$

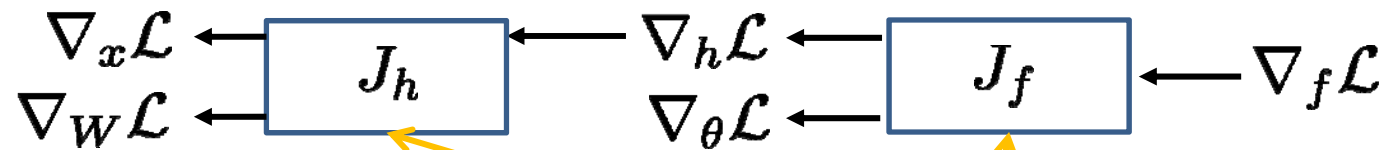
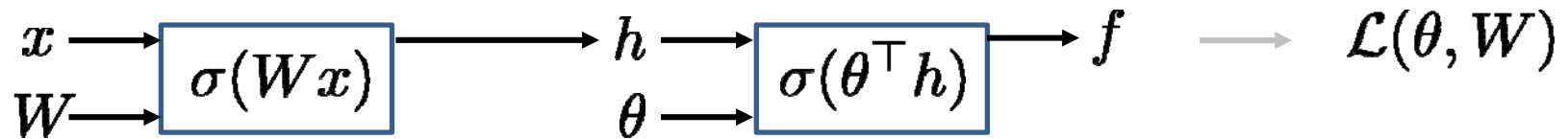
- To minimize $\mathcal{L}(\theta, W)$ we need gradients:

$$\nabla_{\theta} \mathcal{L}(\theta, W) \text{ and } \nabla_W \mathcal{L}(\theta, W)$$

- Then use gradient descent algorithm as before.
- Formula for $\nabla_{\theta} \mathcal{L}(\theta, W)$ can be found by hand (same as before); but what about W ?
 - Beyond the scope of this course

Back-propagation

- Can re-apply chain rule to get gradients for all intermediate values and parameters.



“Backward” modules for each forward stage.

Training Procedure

- Collect labeled training data
 - For SGD: Randomly shuffle after each epoch!

$$\mathcal{X} = \{(x^{(i)}, y^{(i)}) : i = 1, \dots, m\}$$

- For a batch of examples:
 - Compute gradient w.r.t. all parameters in network.

$$\Delta_{\theta} := \nabla_{\theta} \mathcal{L}(\theta, W)$$

$$\Delta_W := \nabla_W \mathcal{L}(\theta, W)$$

- Make a small update to parameters.

$$\theta := \theta - \eta_{\theta} \Delta_{\theta}$$

$$W := W - \eta_W \Delta_W$$

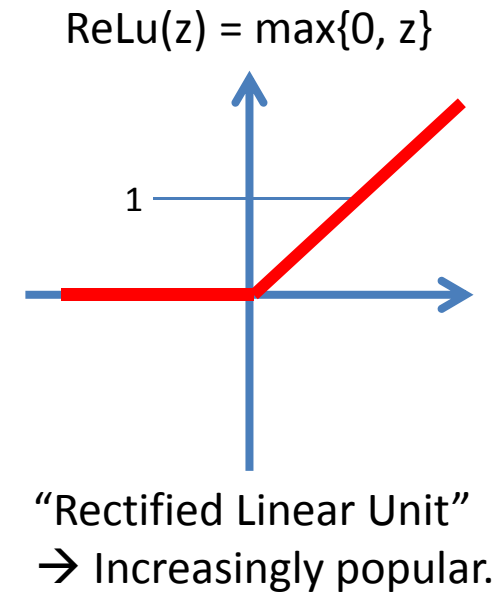
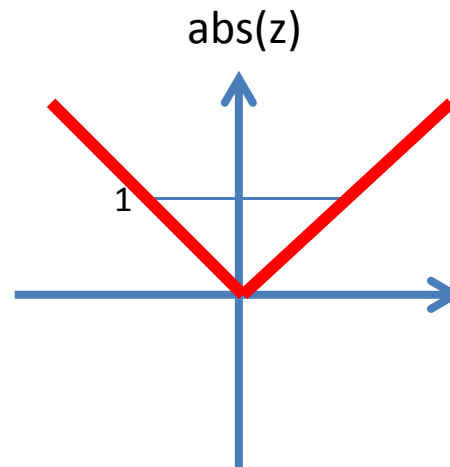
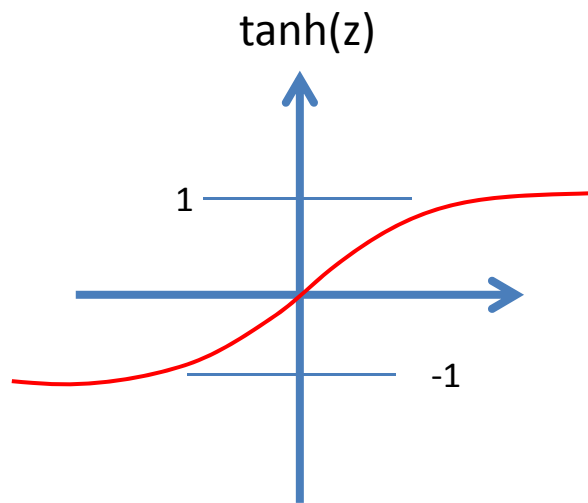
- Repeat until convergence.

Training Procedure

- Historically, this has not worked so easily.
 - Non-convex: Local minima; convergence criteria.
 - Optimization becomes difficult with many stages.
 - “Vanishing gradient problem”
 - Hard to diagnose and debug malfunctions.
- Many things turn out to matter:
 - Choice of nonlinearities.
 - Initialization of parameters.
 - Optimizer parameters: step size, schedule.

Nonlinearities

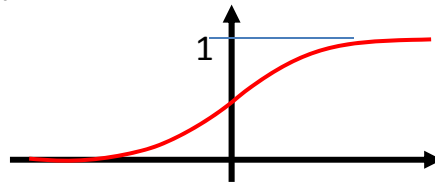
- Choice of functions inside network matters.
 - Sigmoid function turns out to be difficult.
 - Some other choices often used:



[Nair & Hinton, 2010]

Initialization

- Usually small random values.
 - Try to choose so that typical input to a neuron avoids saturating / non-differentiable areas.



- Occasionally inspect units for saturation / blowup.
 - Larger values may give faster convergence, but worse models!
- Initialization schemes for particular units:
 - tanh units: Unif[-r, r]; sigmoid: Unif[-4r, 4r].

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

See [[Glorot et al., AISTATS 2010](#)]

Application

SUPERVISED DL FOR VISION

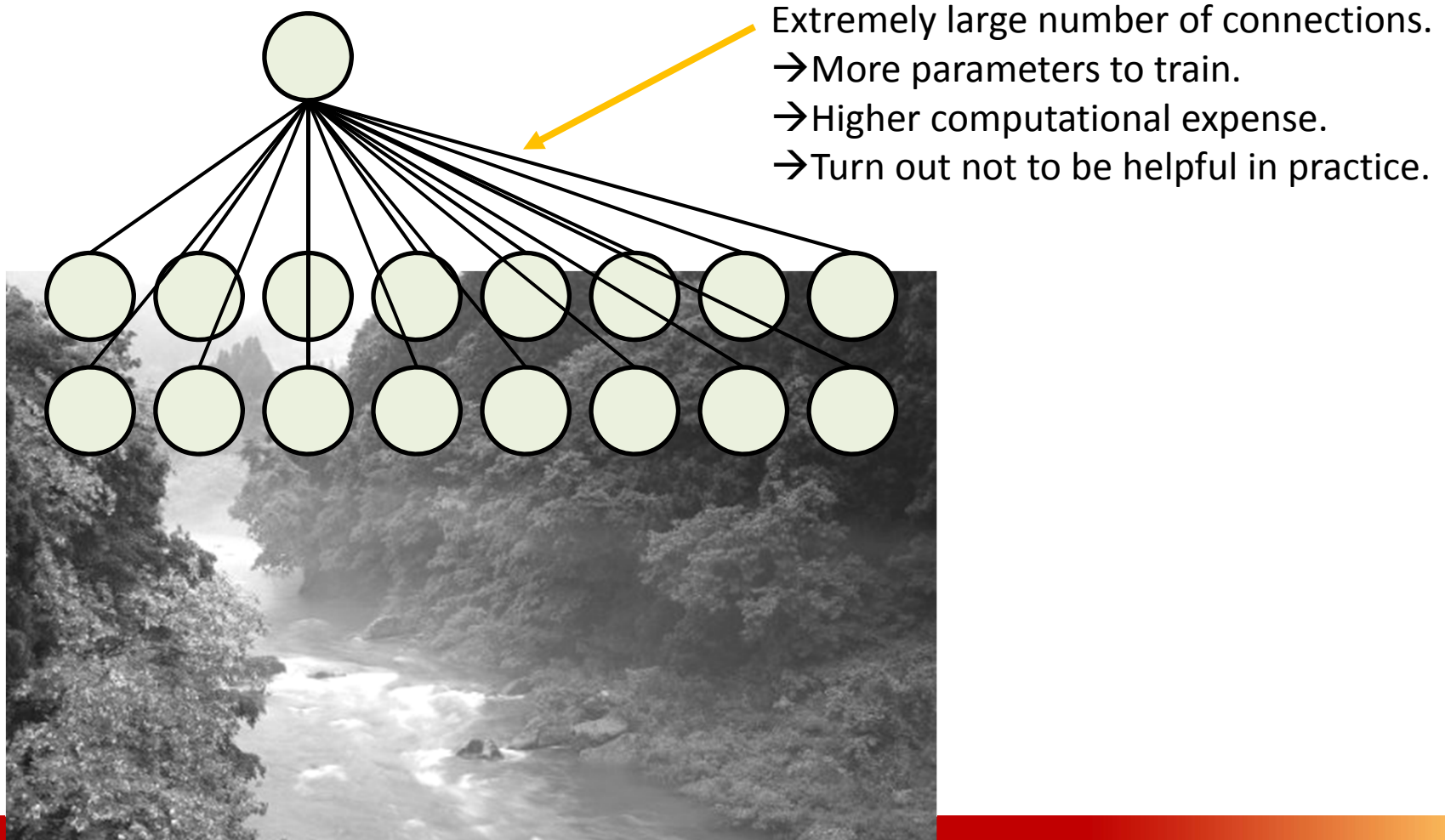


Working with images

- Major factors:
 - Want to have “selective” features and “invariant” features.
 - Try to exploit knowledge of images to accelerate training or improve performance.
- Generally try to avoid wiring detailed visual knowledge into system --- prefer to learn.

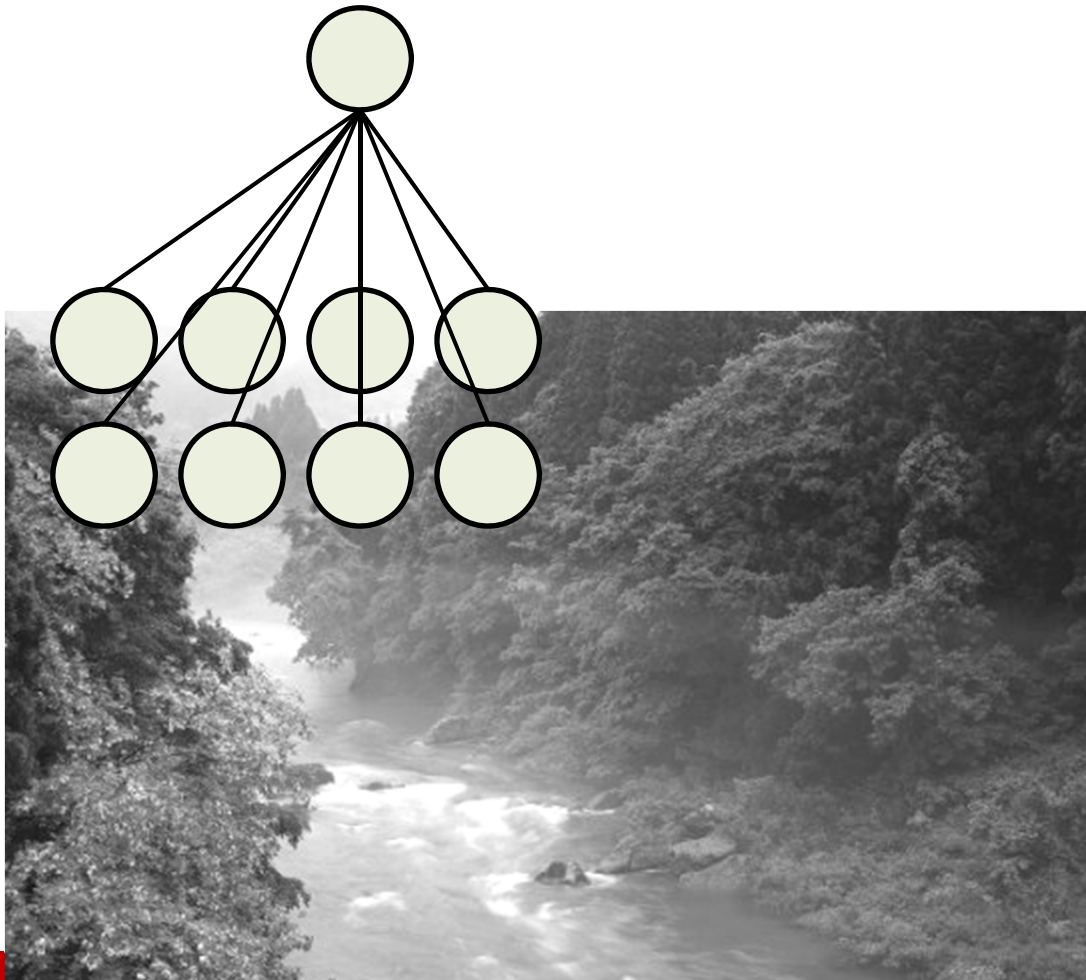
Local connectivity

- Neural network view of single neuron:



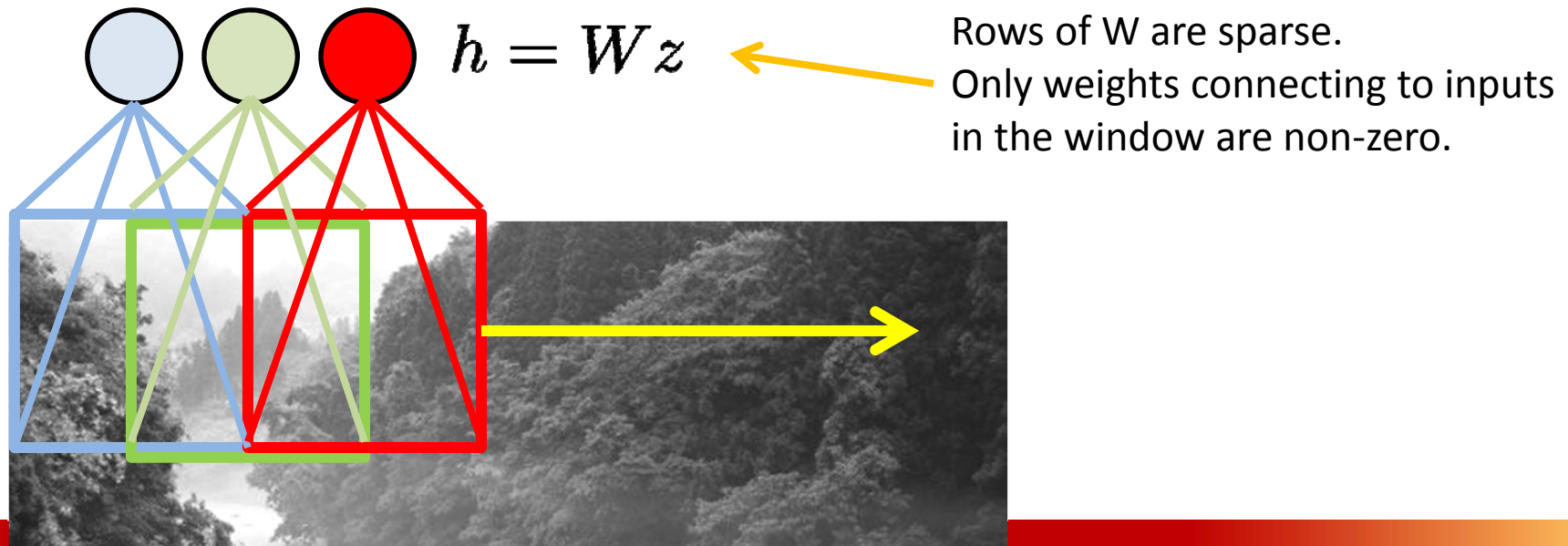
Local connectivity

- Reduce parameters with local connections.
 - Weight vector is a spatially localized “filter”.



Local connectivity

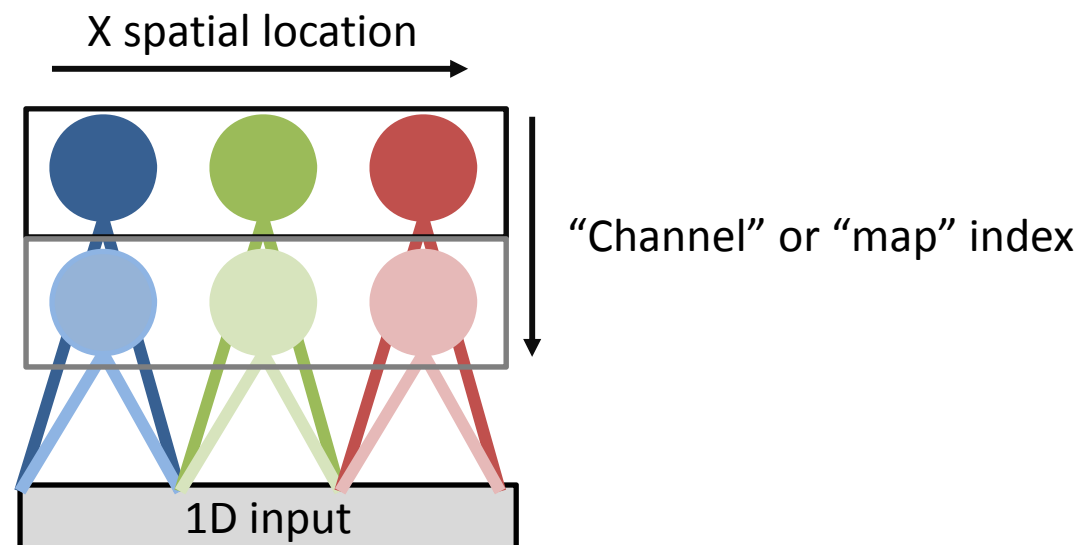
- Sometimes think of neurons as viewing small adjacent windows.
 - Specify connectivity by the size (“receptive field” size) and spacing (“step” or “stride”) of windows.
 - Typical RF size = 5 to 20
 - Typical step size = 1 pixel up to RF size.



Local connectivity

- Spatial organization of filters means output features can also be organized like an image.
 - X,Y dimensions correspond to X,Y position of neuron window.
 - “Channels” are different features extracted from same spatial location. (Also called “feature maps”, or “maps”.)

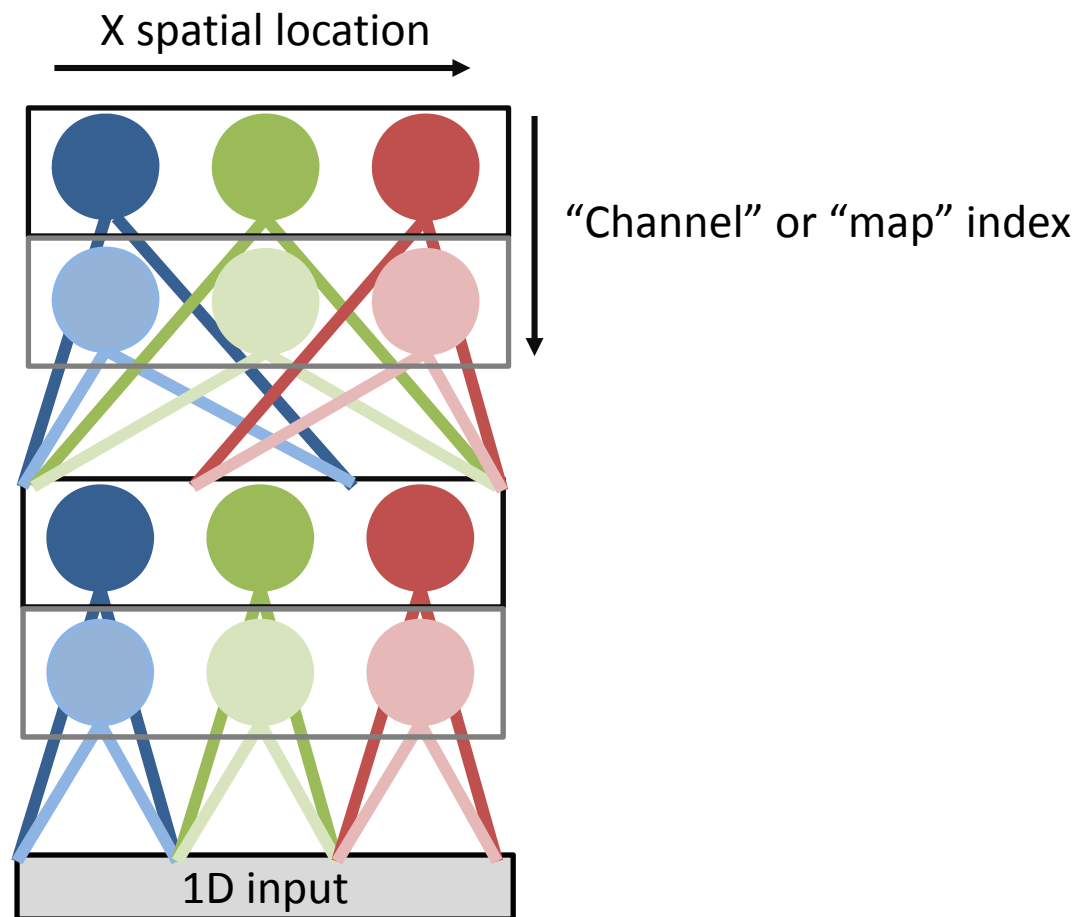
1-dimensional example:



Local connectivity

- We can treat output of a layer like an image and re-use the same tricks.

1-dimensional example:

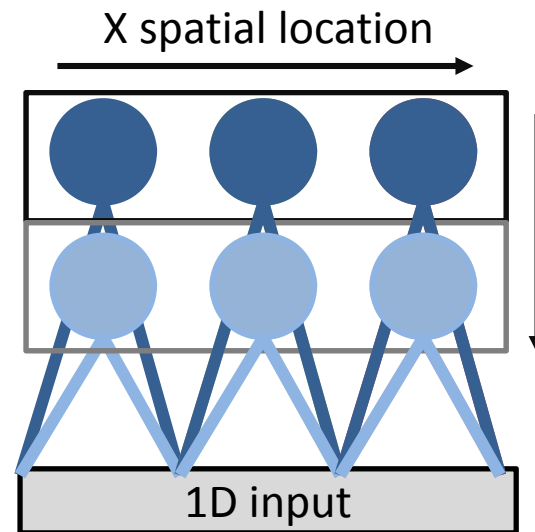


Weight-Tying

- Even with local connections, may still have too many weights.
 - Trick: constrain some weights to be equal if we know that some parts of input should learn same kinds of features.
 - Images tend to be “stationary”: different patches tend to have similar low-level structure.

Weight-Tying or Convolutional Network

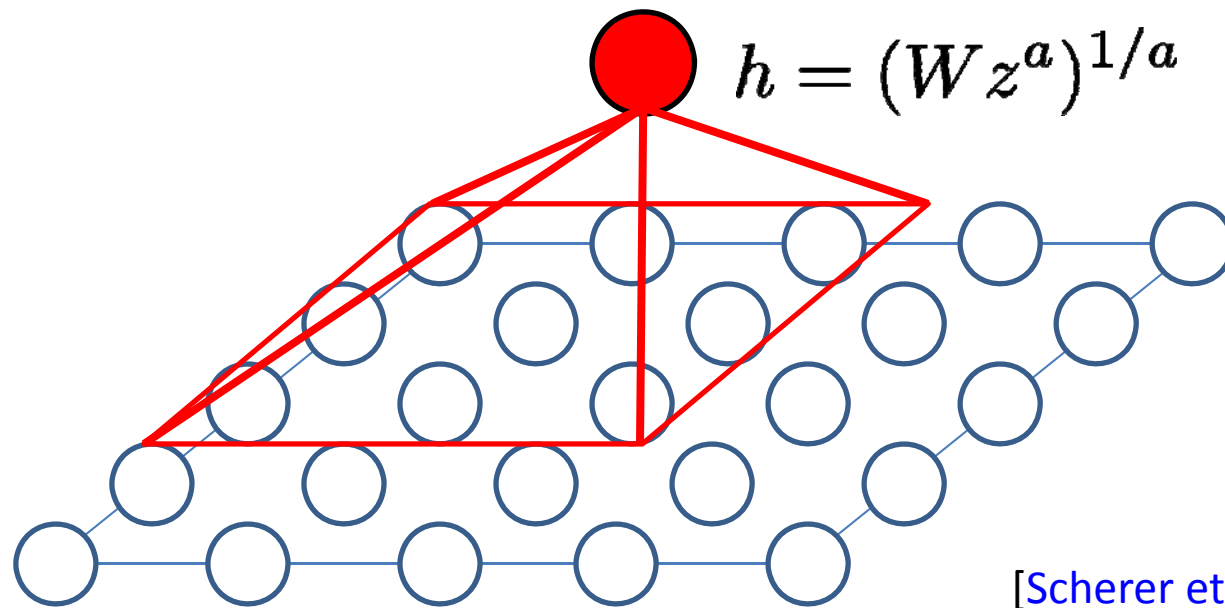
- Reduce parameters by making them equal.



- Each unique filter is spatially convolved with the input to produce responses for each map. [[LeCun et al., 1989](#); [LeCun et al., 2004](#)]

Pooling

- Functional layers designed to represent invariant features.
- Usually locally connected with specific nonlinearities.
 - Combined with convolution, corresponds to hard-wired translation invariance.
- Usually fix weights to local box or gaussian filter.
 - Easy to represent max-, average-, or 2-norm pooling.

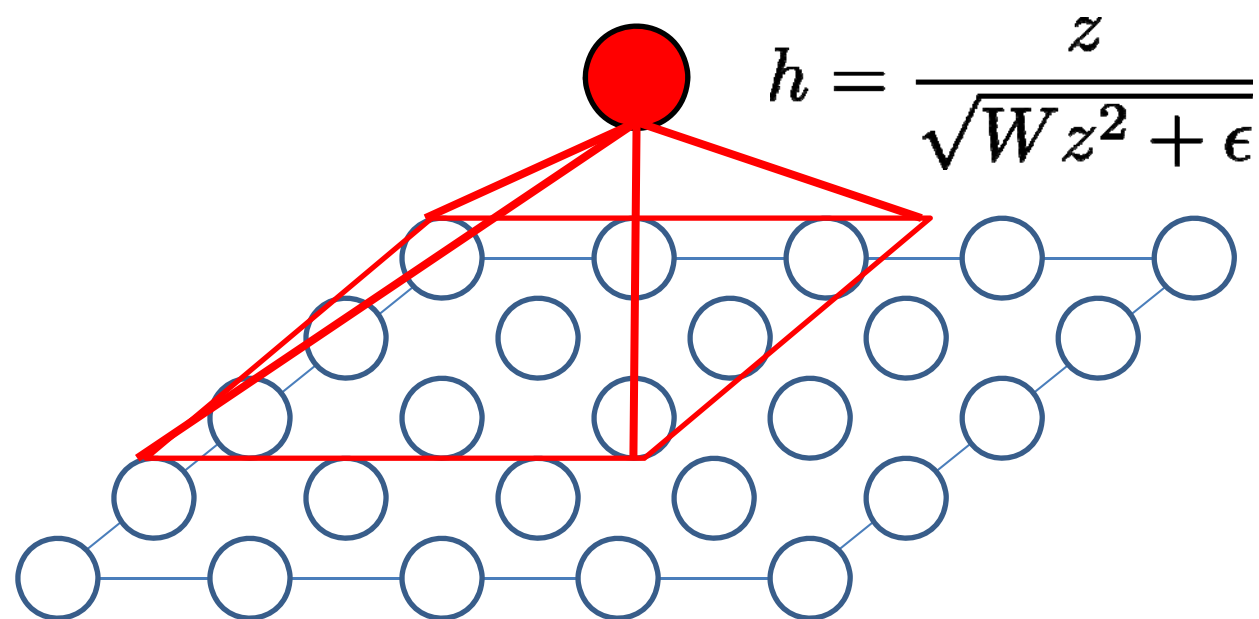


[Scherer et al., ICANN 2010]

[Boureau et al., ICML 2010]

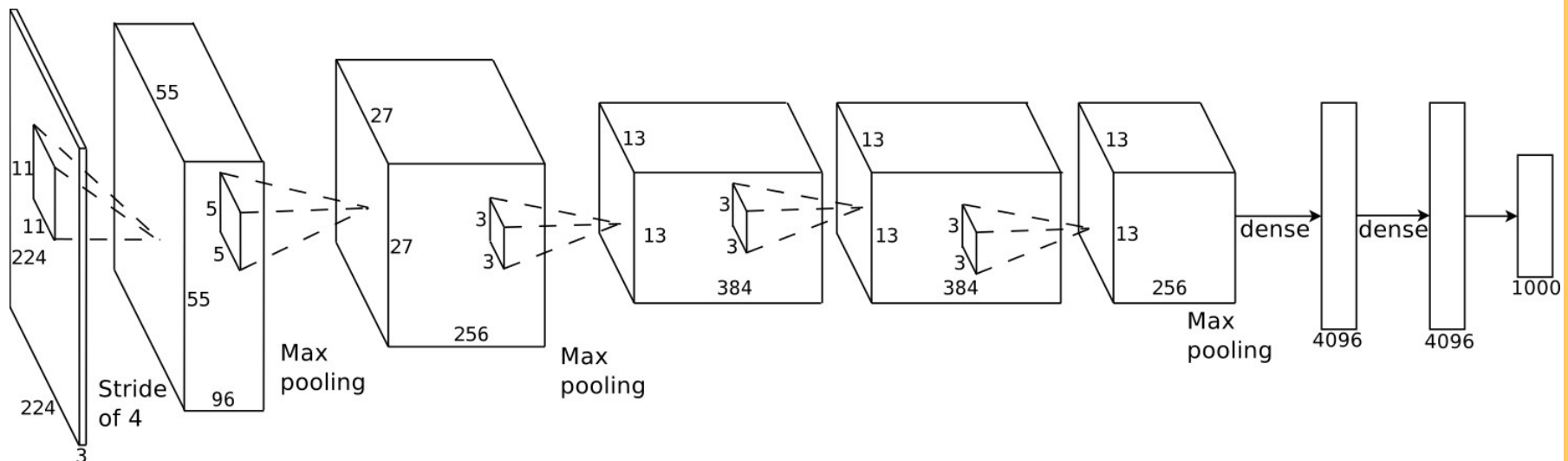
Contrast Normalization

- Empirically useful to soft-normalize magnitude of groups of neurons.
 - Sometimes we subtract out the local mean first.



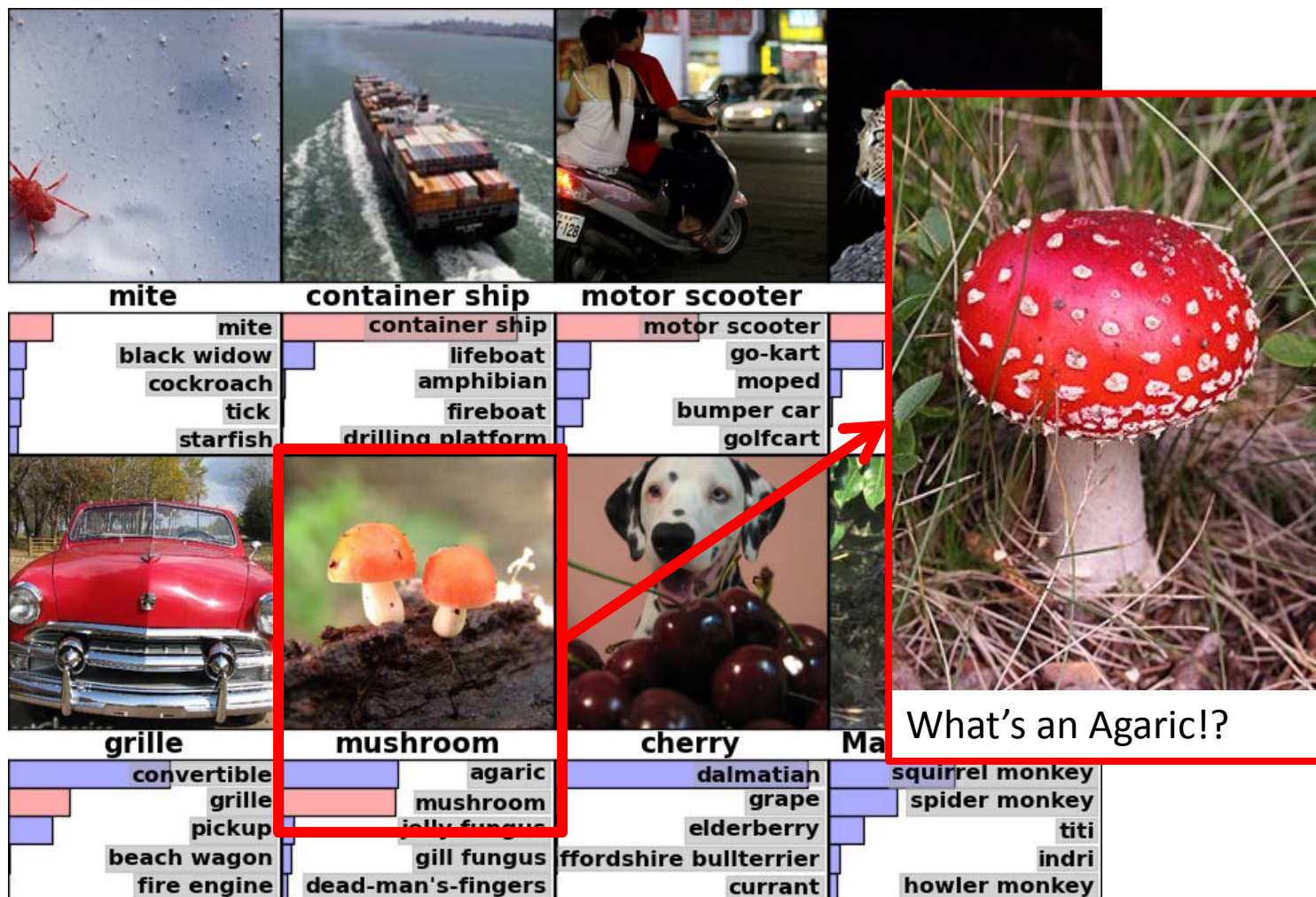
Application: Image-Net

- System from [Krizhevsky et al., NIPS 2012](#):
 - Convolutional neural network.
 - Max-pooling.
 - Rectified linear units (ReLU).
 - Contrast normalization.
 - Local connectivity.



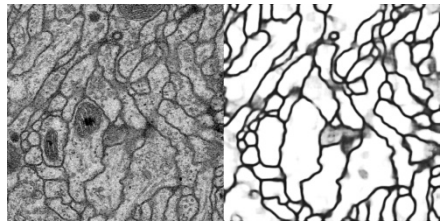
Application: Image-Net

- Top result in LSVRC 2012: ~85%, Top-5 accuracy.



More applications

- Segmentation: predict classes of pixels / super-pixels.



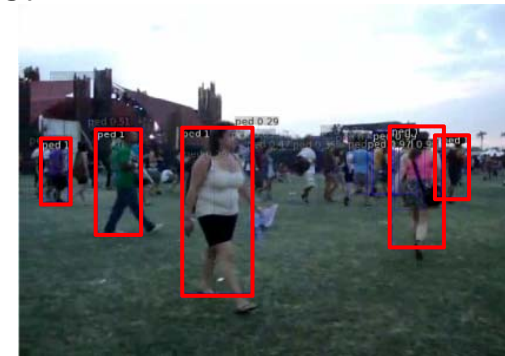
Farabet et al., ICML 2012 →

← Ciresan et al., NIPS 2012

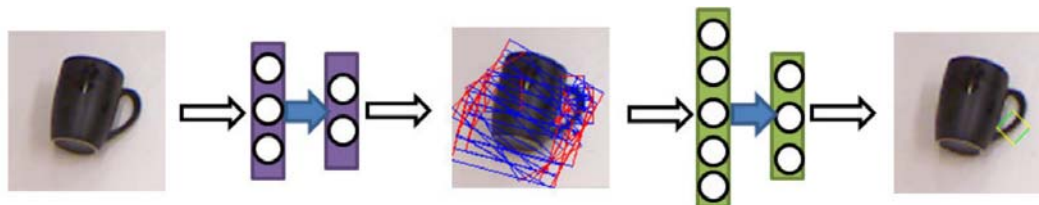


- Detection: combine classifiers with sliding-window architecture.
 - Economical when used with convolutional nets.

Pierre Sermanet (2010) →



- Robotic grasping. [Lenz et al., RSS 2013]



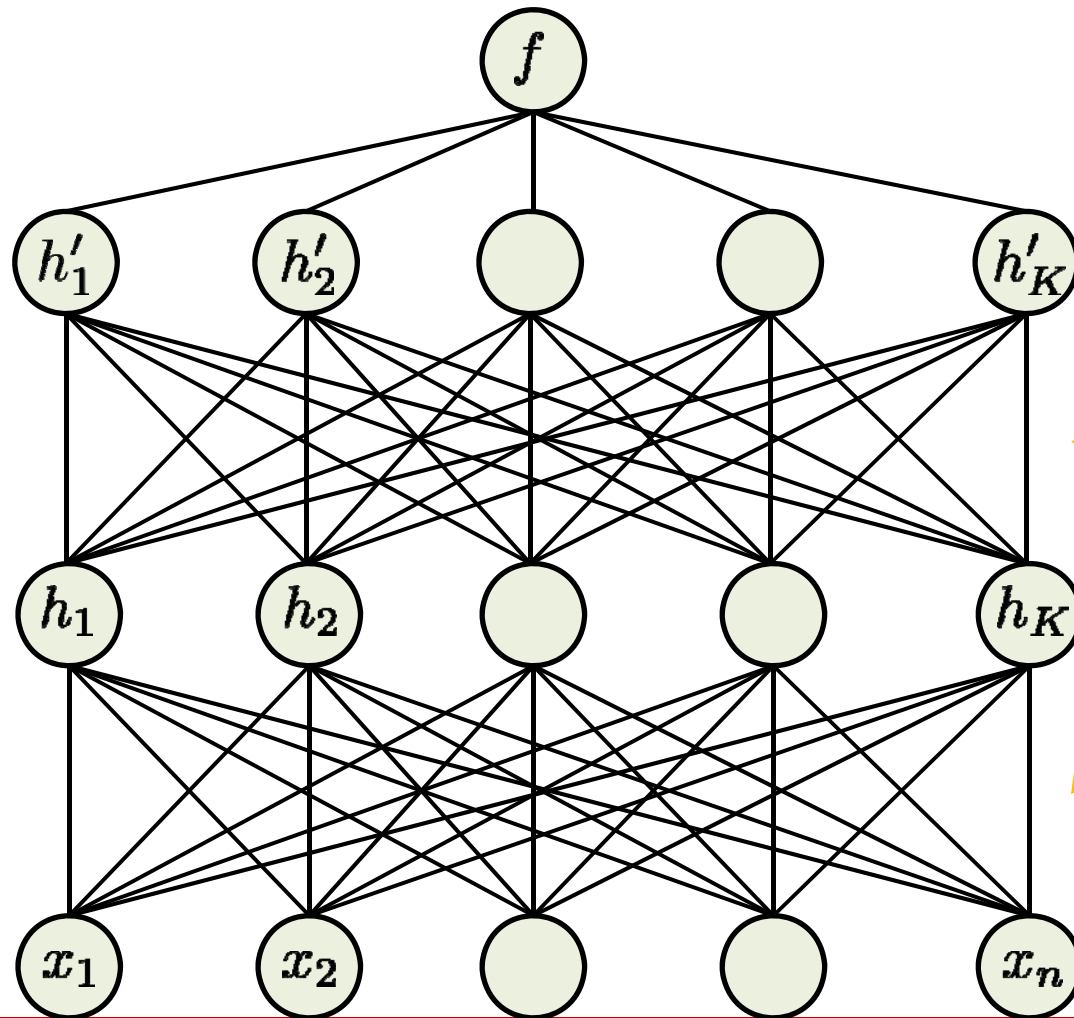
<http://www.youtube.com/watch?v=f9Cuzql1SkE>

UNSUPERVISED DL



Representation Learning

- In supervised learning, train “features” to accomplish top-level objective.

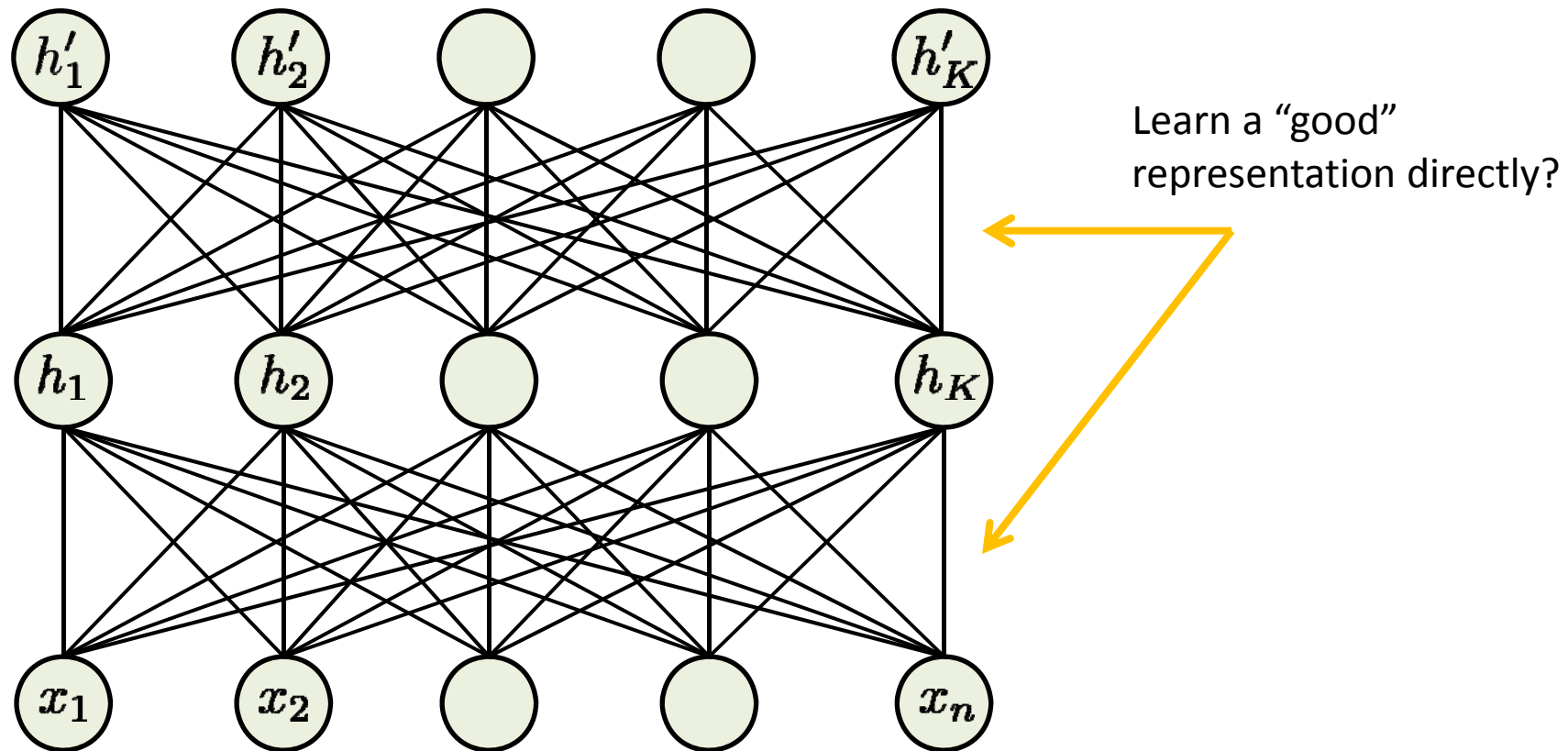


But what if we have too few labels to train all these parameters?



Representation Learning

- Can we train the “representation” without using top-down supervision?



Representation Learning

- What makes a good representation?
 - Distributed: roughly, K features represents more than K types of patterns.
 - E.g., K binary features that can vary independently to represent 2^K patterns.
 - Invariant: robust to local changes of input; more abstract.
 - E.g., pooled edge features: detect edge at several locations.
 - Disentangling factors: put separate concepts (e.g., color, edge orientation) in separate features.

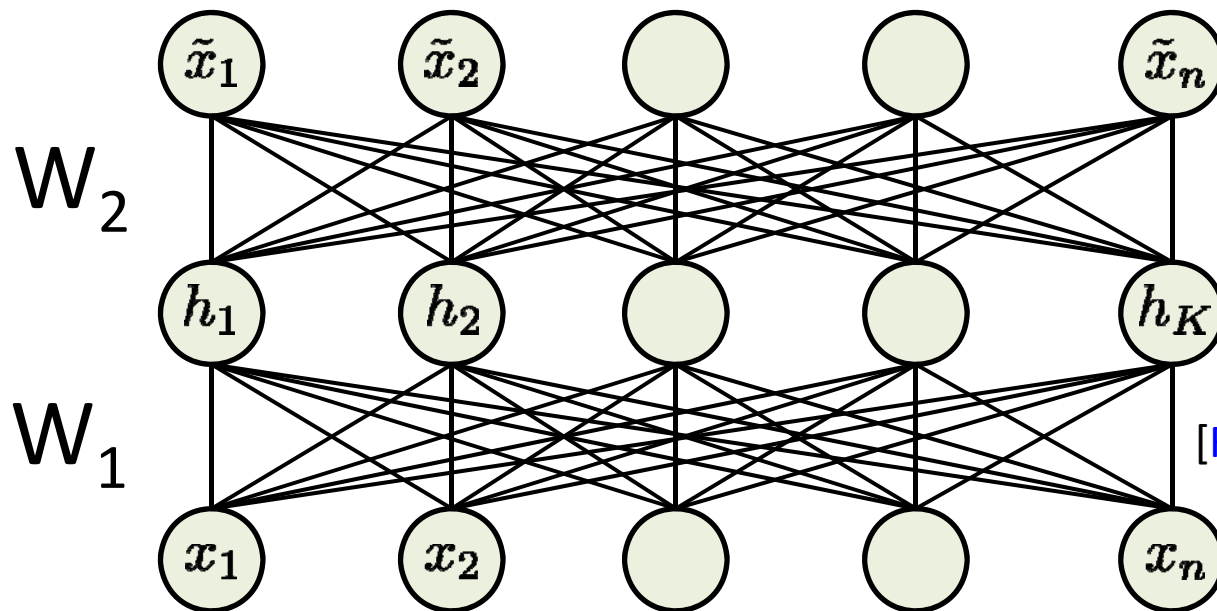
Sparse auto-encoder

- Train two-layer neural network by minimizing:

$$\underset{W_1, W_2}{\text{minimize}} \sum_i \|W_2 h(W_1 x^{(i)}) - x^{(i)}\|_2^2 + \lambda \|h(W_1 x^{(i)})\|_1$$

$$h(z) = \text{ReLU}(z)$$

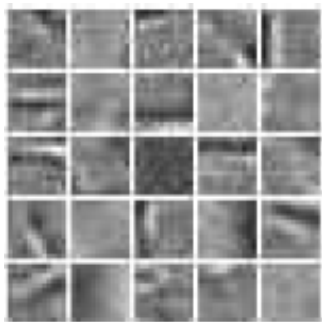
- Remove “decoder” and use learned features (h).



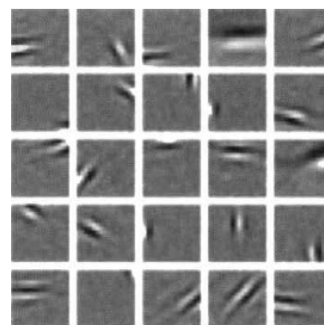
[Ranzato et al., NIPS 2006]

What features are learned?

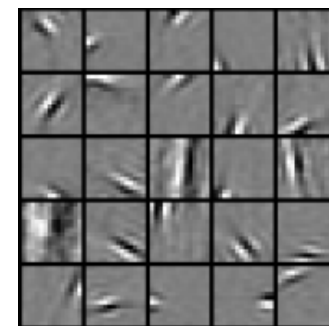
- Applied to image patches, well-known result:



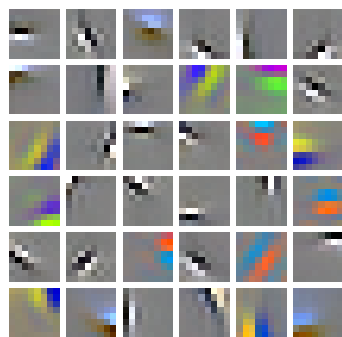
Sparse auto-encoder
[Ranzato et al., 2007]



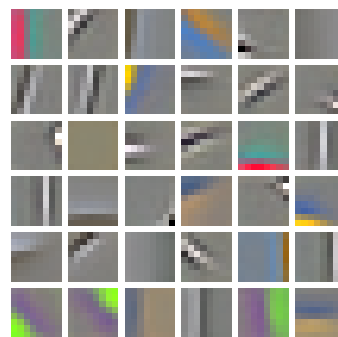
Sparse coding
[Olshausen & Field, 1996]



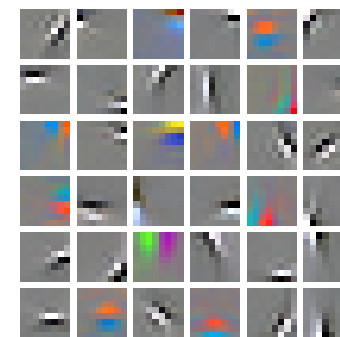
Sparse RBM
[Lee et al., 2007]



Sparse auto-encoder



K-means



Sparse RBM

Summary

- Supervised deep-learning
 - Practical and highly successful in practice. A general-purpose extension to existing ML.
 - Optimization, initialization, architecture matter!
- Unsupervised deep-learning
 - Pre-training often useful in practice.
 - Difficult to train many layers of features without labels.

Resources

Tutorials

Stanford Deep Learning tutorial:

<http://ufldl.stanford.edu/wiki>

Deep Learning tutorials list:

<http://deeplearning.net/tutorials>

IPAM DL/UFL Summer School:

<http://www.ipam.ucla.edu/programs/gss2012/>

ICML 2012 Representation Learning Tutorial

<http://www.iro.umontreal.ca/~bengioy/talks/deep-learning-tutorial-2012.html>

References

<http://www.stanford.edu/~acoates/bmvc2013refs.pdf>

Overviews:

Yoshua Bengio,

“Practical Recommendations for Gradient-Based Training of Deep Architectures”

Yoshua Bengio & Yann LeCun,

“Scaling Learning Algorithms towards AI”

Yoshua Bengio, Aaron Courville & Pascal Vincent,

“Representation Learning: A Review and New Perspectives”

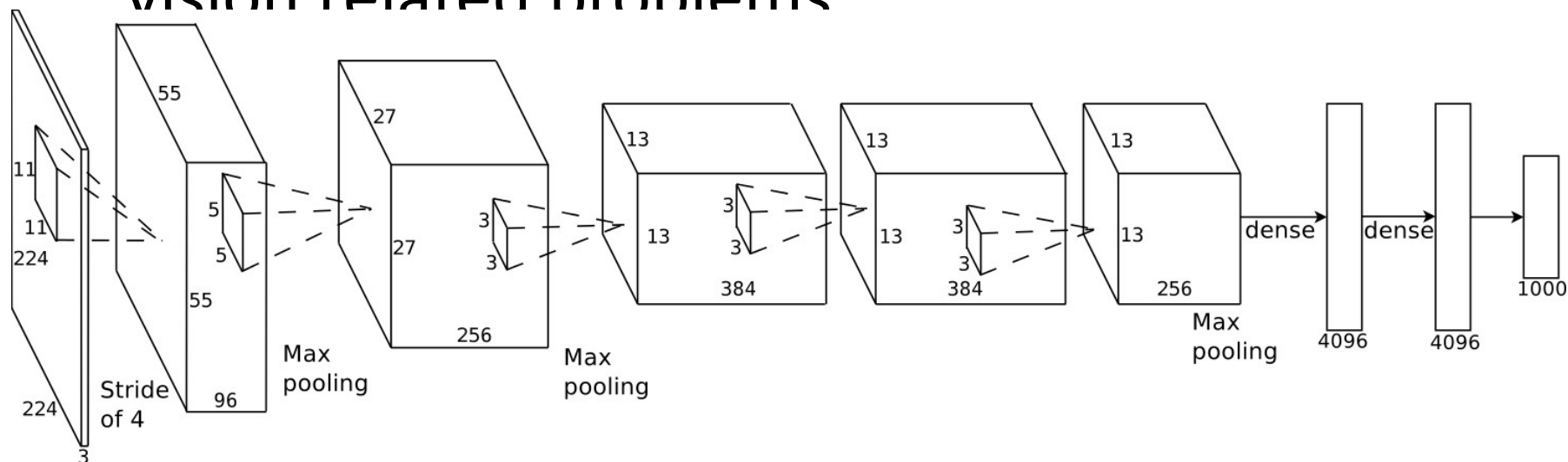
Software:

Theano GPU library: <http://deeplearning.net/software/theano>

SPAMS toolkit: <http://spams-devel.gforge.inria.fr/>

High-Level Messages

- Deep neural nets provide low-level and high-level features
 - We can use those features for image search
- Achieve the best results in many computer vision related problems



Krizhevsky et al., NIPS 2012

PA3

- **Apply binary code embedding and inverted index to PA2**
 - Spherical hashing for binary code embedding
 - **K-means clustering** or product quantization (PQ) for inverted index