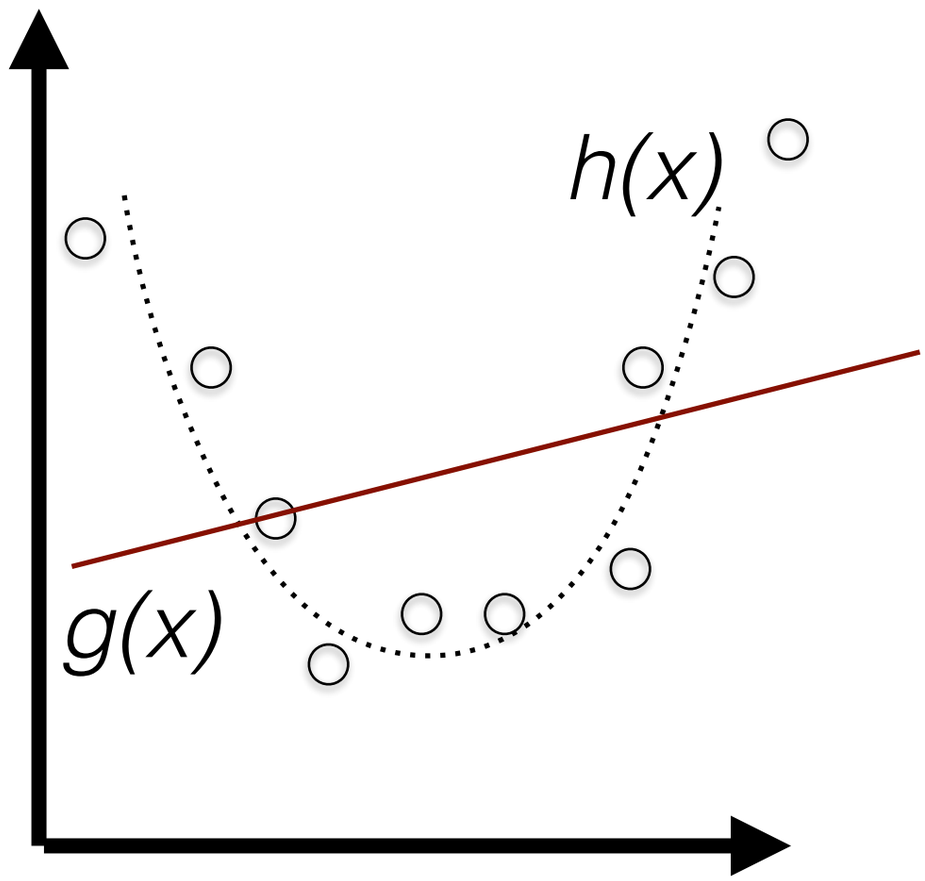
Connectionism 4

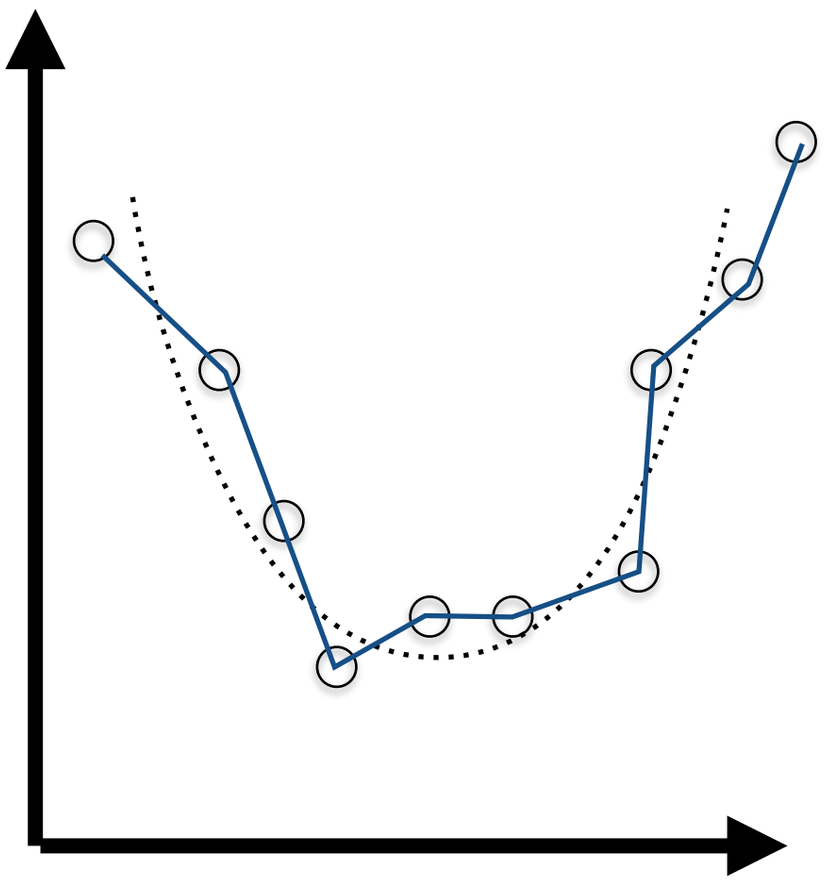
1: Bias, Generalization, and the craft of training networks

2: Introduction to representation

Topic: Bias and Generalization

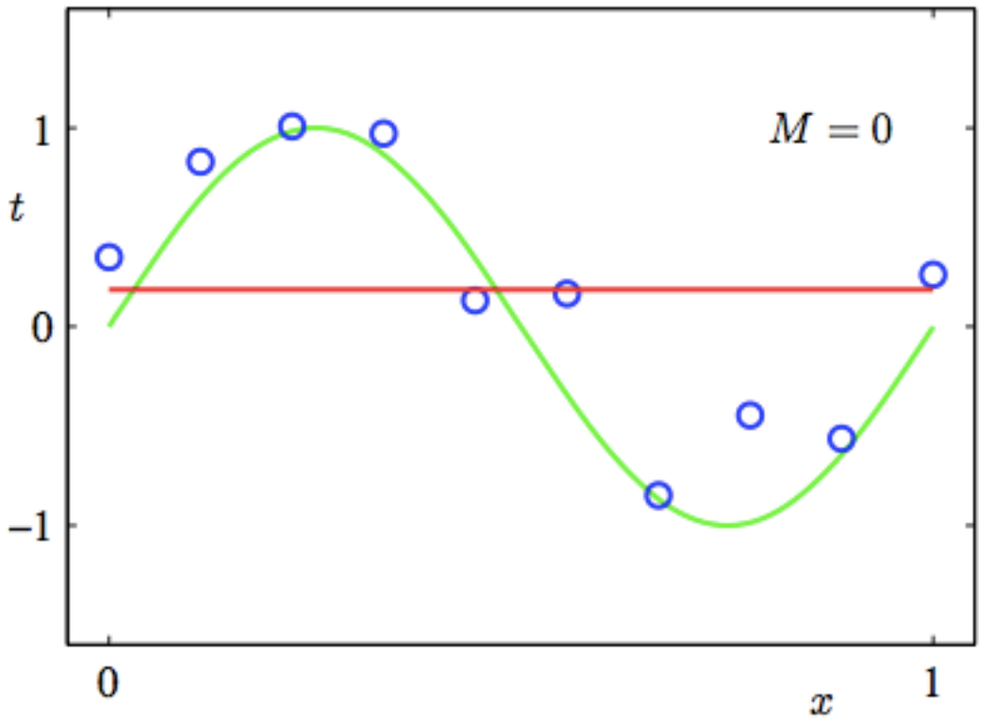
Data generated using this: $y = h(x) + \epsilon$

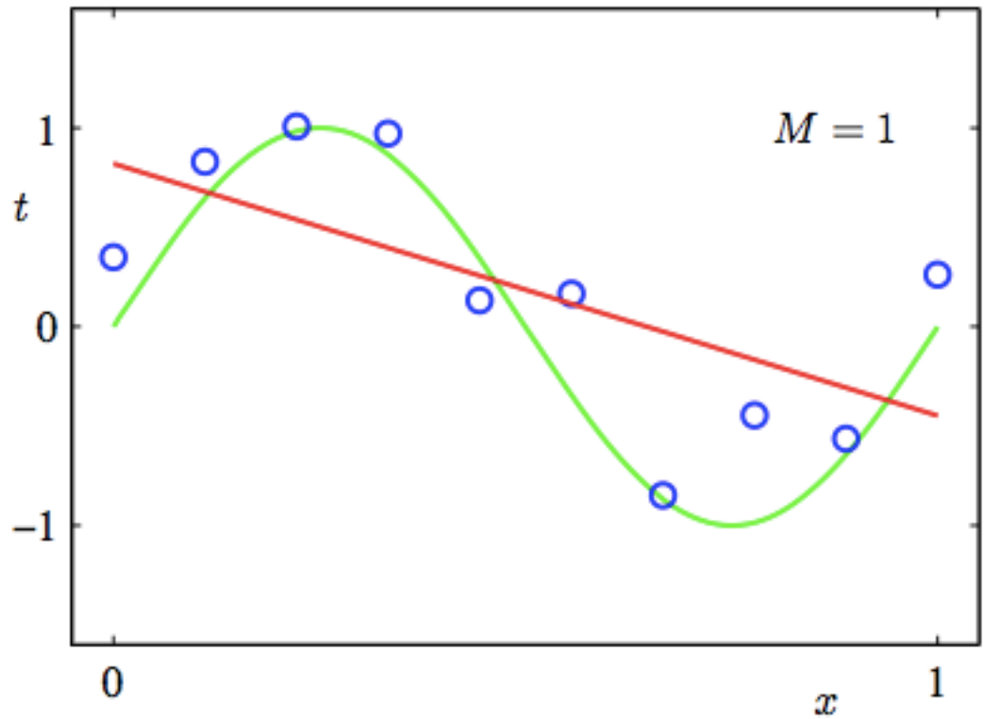


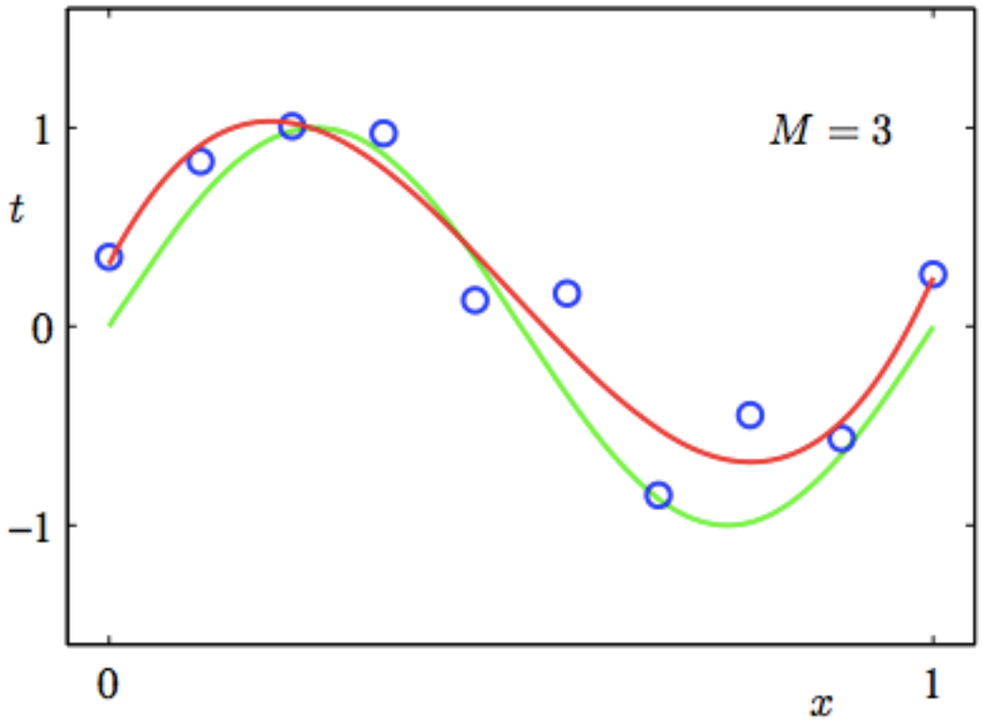


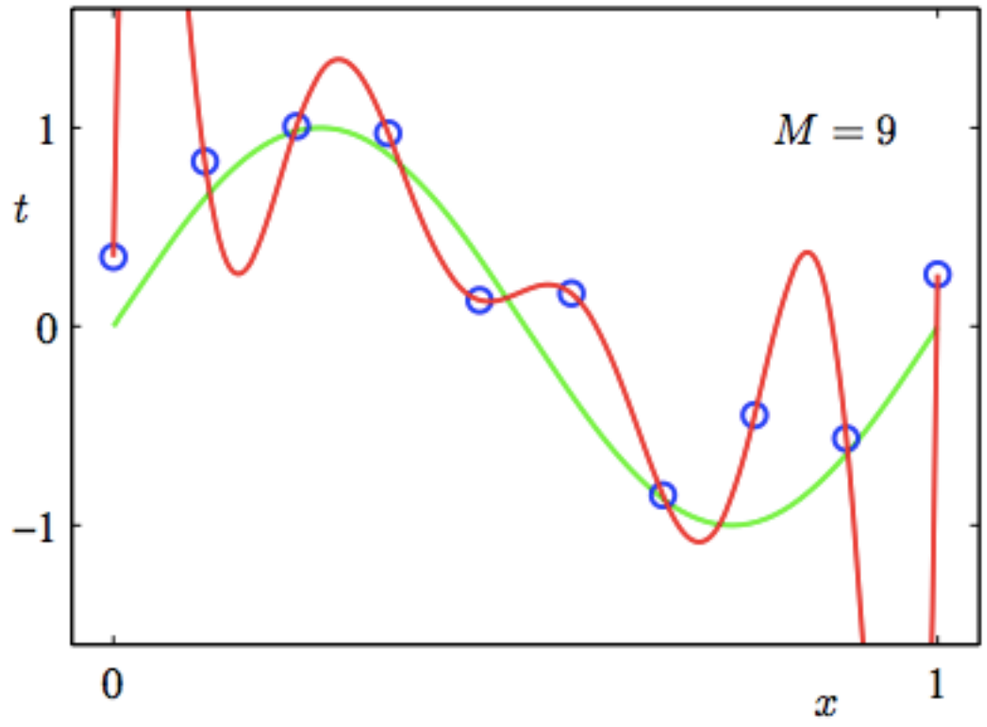
The linear model is not very good: it has a **high bias.** I.e., it is not sufficiently influenced by the data

This model is **overfitted**. It is overly influenced by the data.


M = 0
t
x
1
0
−1
0
1


M = 1
t
x
1
0
−1
0
1


M = 3
t
x
1
0
−1
0
1


M = 9
t
x
1
0
−1
0
1

What kind of model might exhibit high bias?
What kind might be prone to overfitting?

large network with 10000 hidden units

small network with 3 hidden units

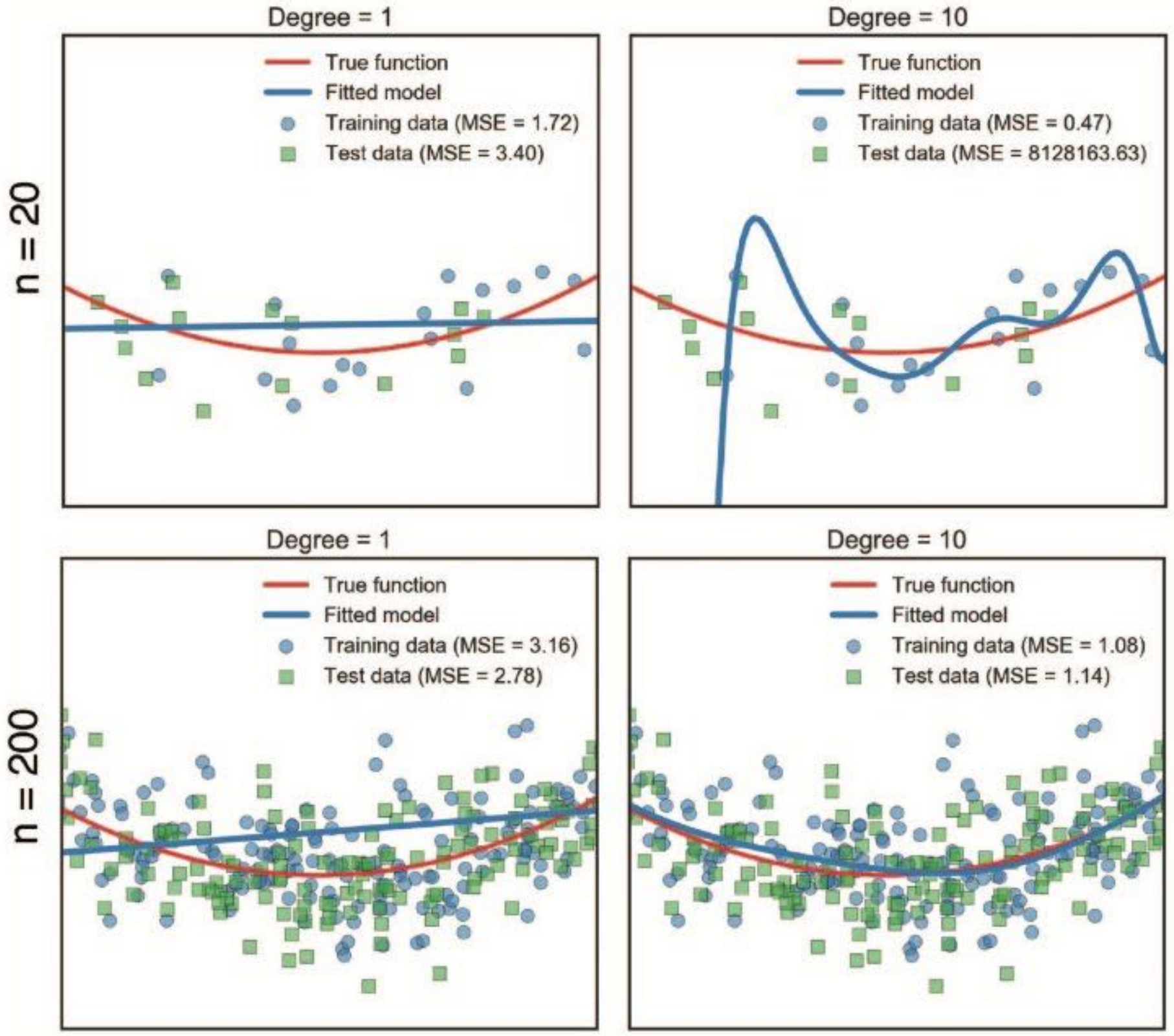


Fig. 4. Large samples guard against overfitting. See text for explanation.

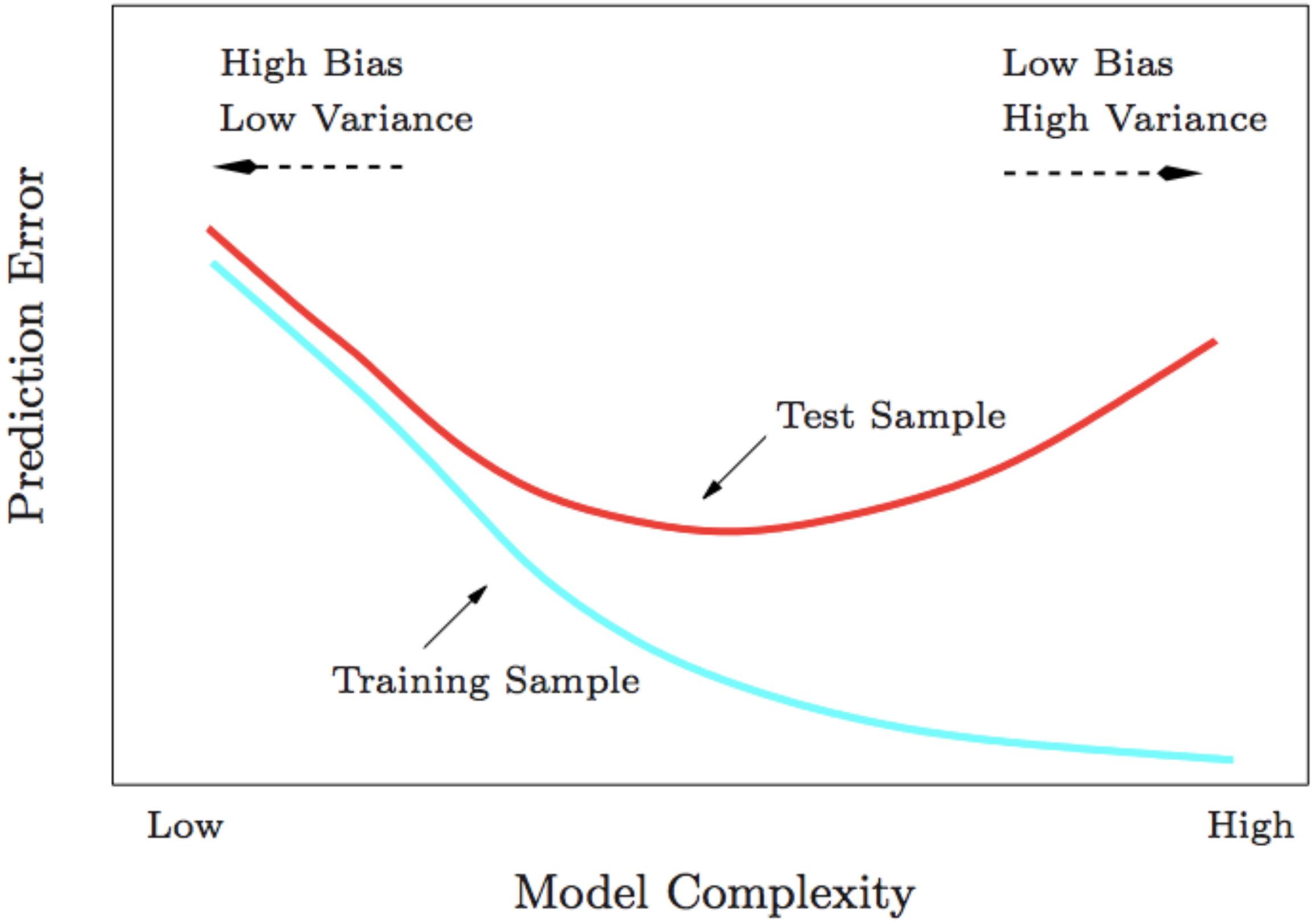
Stop training when you are learning the ‘noise’

10001->001
00100->111
01100->011
01001->000

Training set

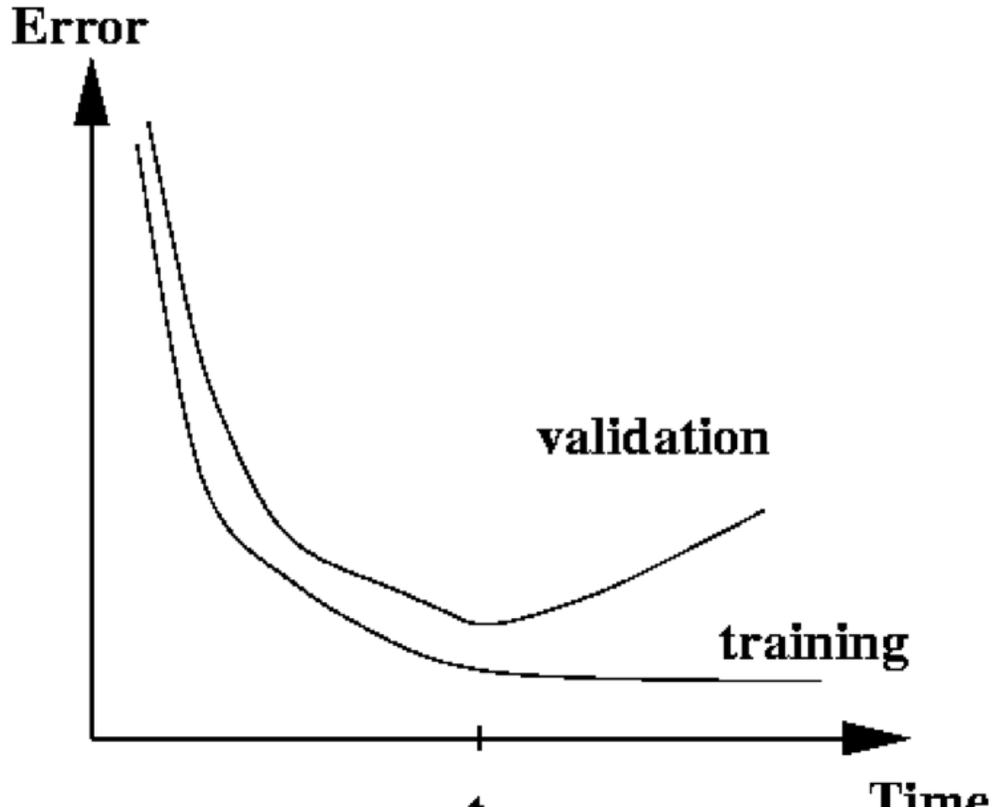
11000->110
11010->101

Independent Test Set


High Bias
Low Variance
Low Bias
High Variance
Prediction Error
Test Sample
Training Sample
Low
High
Model Complexity

10001->001
00100->111
01100->011
01001->000

Training set



11010->100
11011->001

Independent Validation Set (used to determine stopping time only)

11000->110
11010->101

Independent Test Set (used to determine performance of the trained network.

Topic: Tips and Tricks

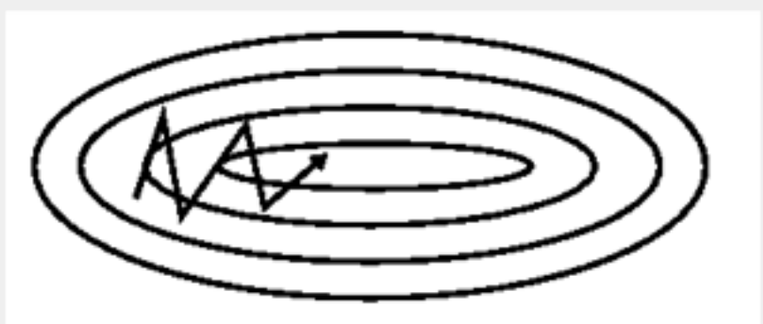
Adding Momentum

If you think of training as rolling down an error surface, then momentum makes sense: if things are going well, proceed with renewed confidence.

Make your current wt change depend not only on this error, but also on the size of your last weight change

$$\Delta w_{ij}(t) = \eta \delta_i o_j + m \Delta w_{ij}(t-1)$$

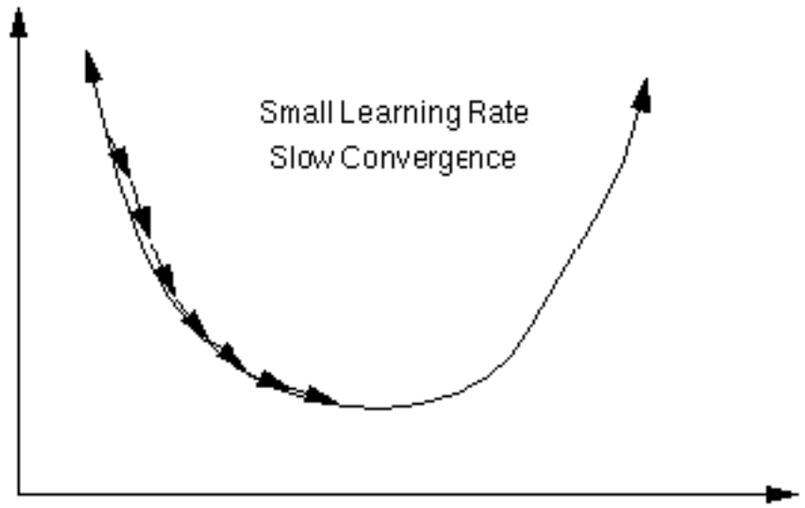
(Fig. 1a)



(Fig. 1b)

Q: How do learning rate and momentum interact?

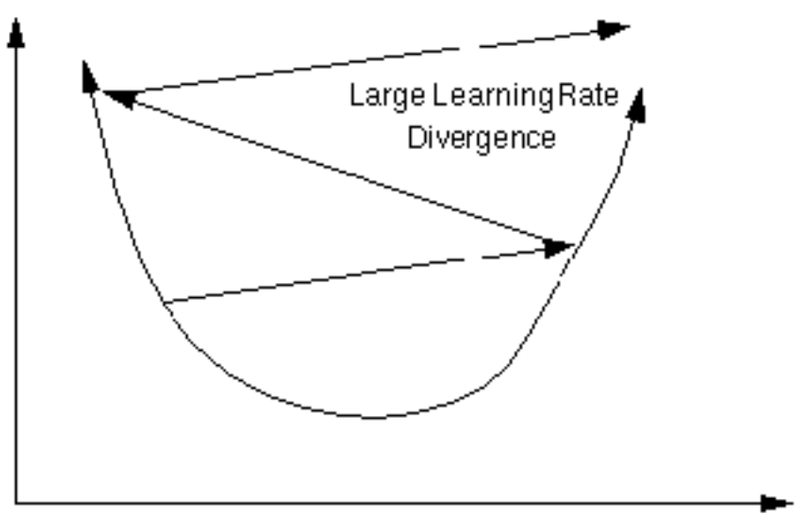
The Importance of Learning Rate....



$$\Delta w_{ij} = -\eta \delta_i o_j$$

Learning Rate

Some training techniques dynamically adapt learning rate. Simulated Annealing is one such. Helps to avoid getting stuck in local minima.

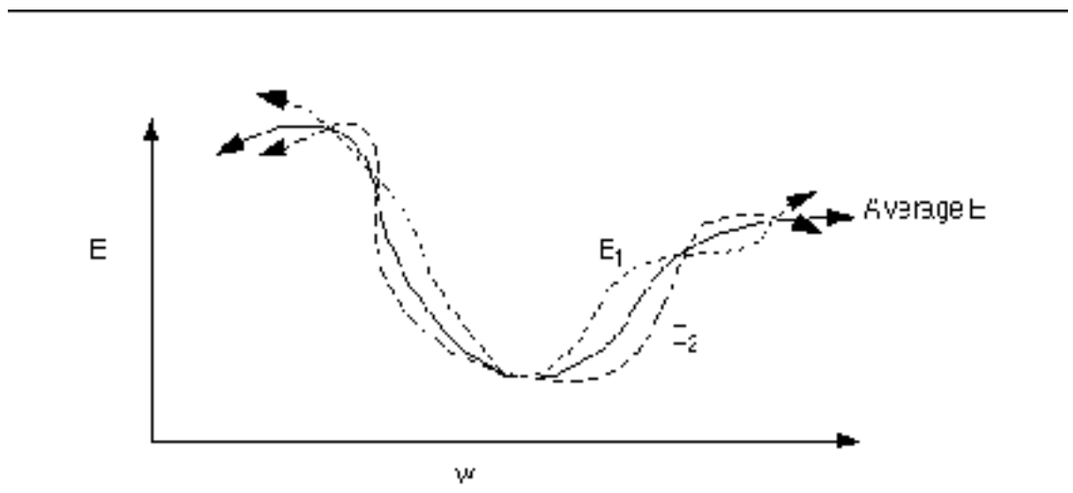


True Gradient Descent:

Accumulate delta-w values for all patterns in your training set, and modify weights only after you have seen each and every pattern. A.k.a. *Batch Training*.

Stochastic Gradient Descent:

Update weights after each pattern. This too may help to avoid local minima. A.k.a. *Online Learning*.



Other Tricks to Avoid Overfitting....

[1] Weight Decay: large weights are reduced in proportion to their size. Based on the observation that large weights and overfitting go hand in hand.

NEURAL NETWORK LAB

Neural Network Weight Decay and Restriction

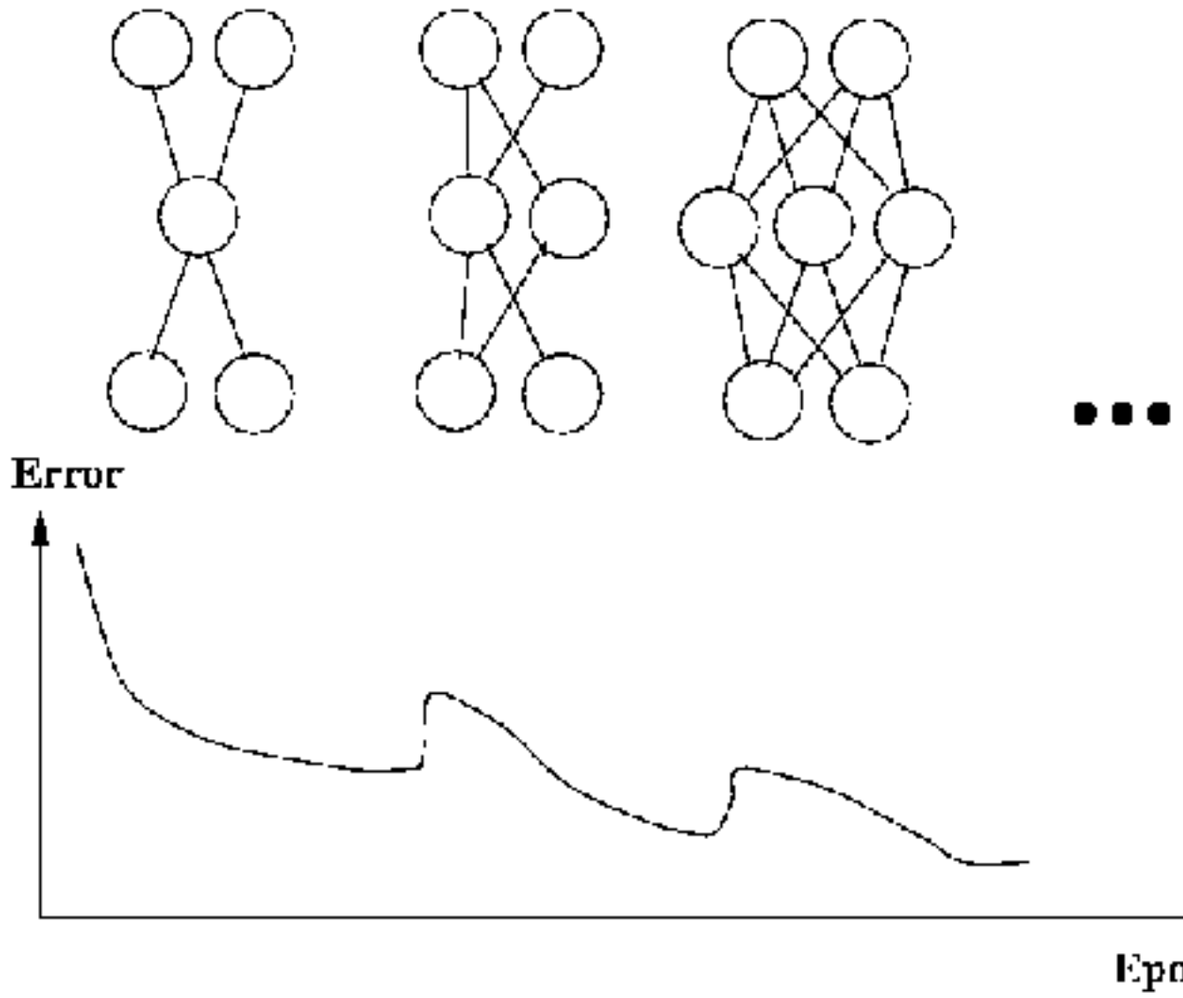
07/28/2014

Weight decay and weight restriction are two closely related, optional techniques that can be used when training a neural network. This article explains exactly what weight decay and weight restriction are, and how to use them with an existing neural network application or implement them in a custom application.

[2] Add Noise: $x_i \longrightarrow x_i + \epsilon$

This is sometimes effective in preventing overfitting.

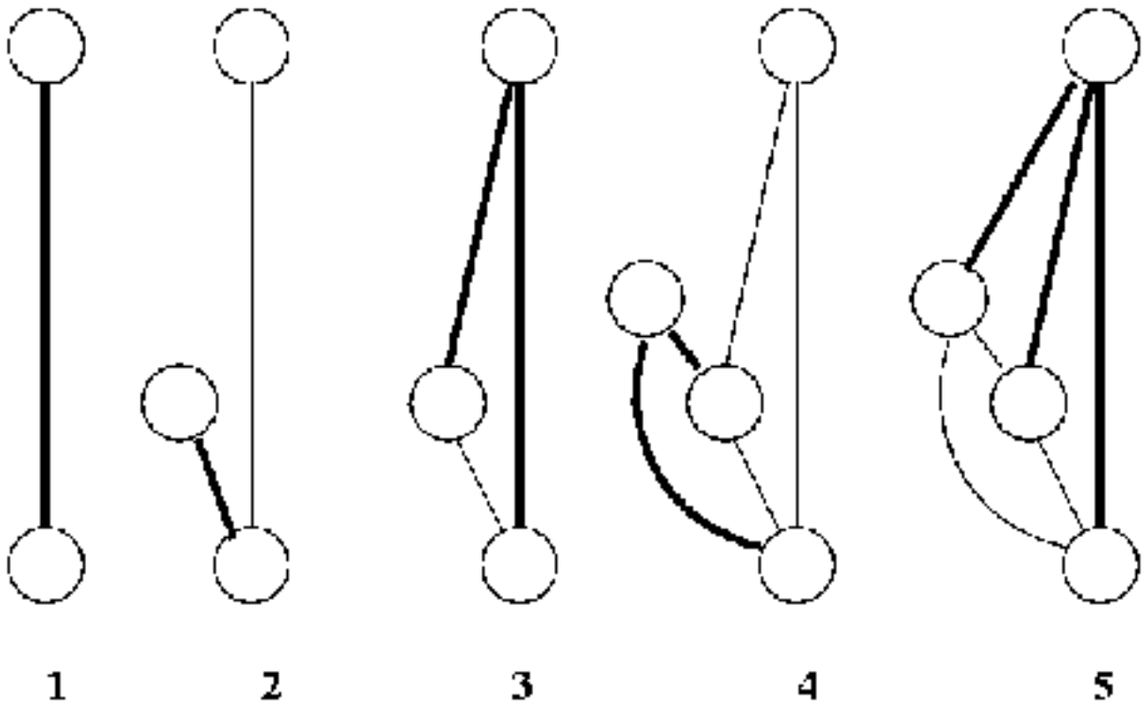
Don’t make your network any larger than it need be..



Many techniques for ‘growing’ and ‘pruning’ networks exist.
E.g. Cascade Correlation (Scott Fahlman).

Topic: Constructive Algorithms, or How to grow a network

Example: Cascade Correlation (Scott Fahlma



We incrementally add units to the network, training just a few weights at a time.

[1] Train wts from input to output until stable
[2] Add a new unit, providing only incoming weights
    Train these weights to be maximally correlated with the remaining network error
[3] Freeze input wts, add new outgoing wts, retrain wts to output until stable
[4] Add new unit with incoming wts from all input and hidden units
[5] repeat and continue

Step 2 is clever! It gets the unit "into the zone" so that it can best contribute to reducing the residual error

Topic: An Informal Taxonomy of Problems

What Kinds of Things Can Networks Learn?

A very rough taxonomy.....

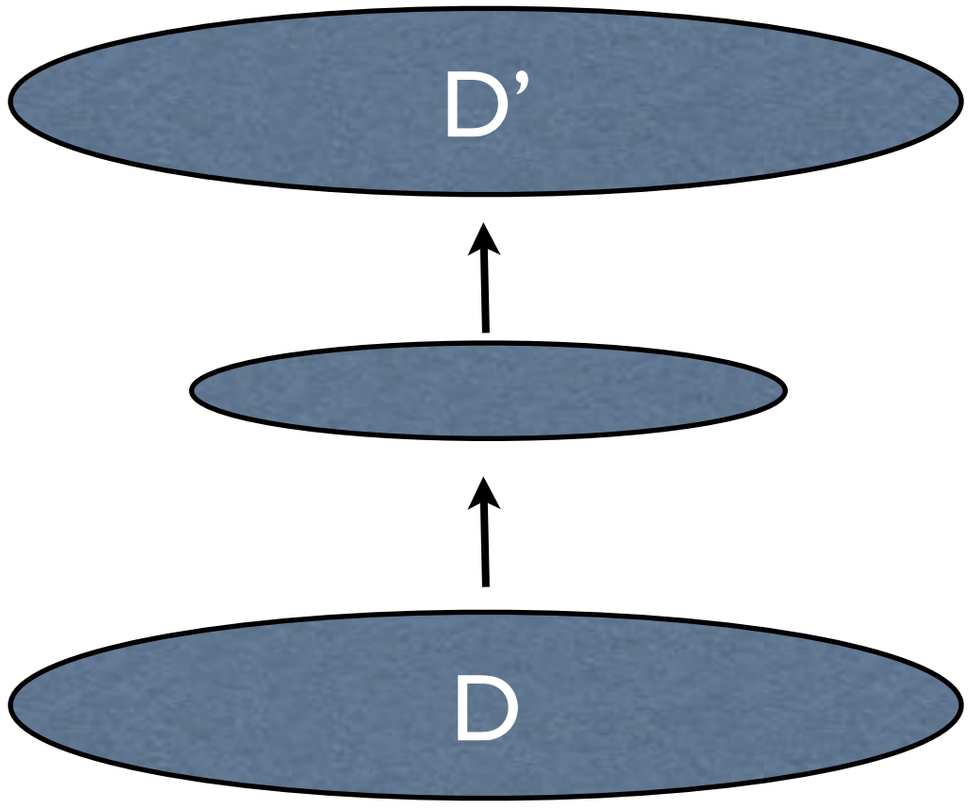
1. Mapping from a rich domain, D, to a similar domain, D'

Examples:

Image processing

Audio processing

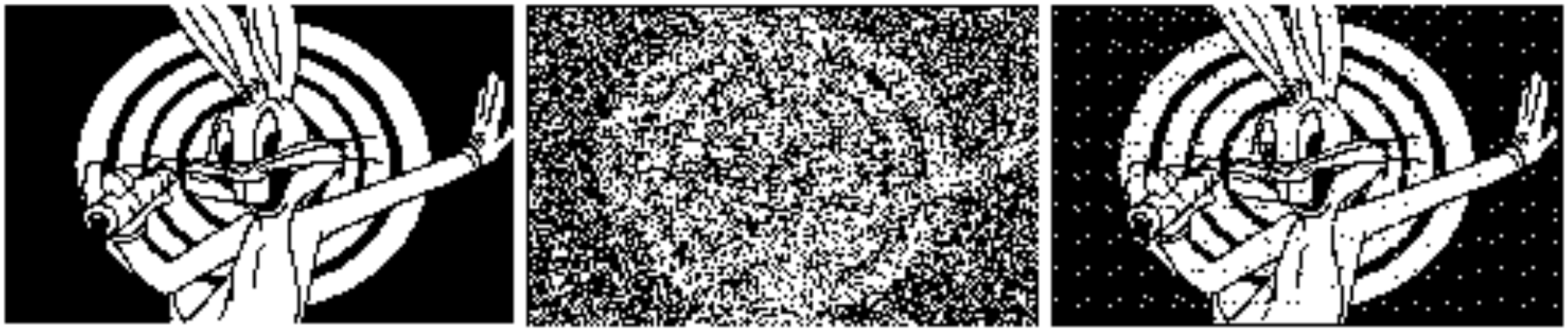
more generally: filtering.....



Training images

After training present these

And the network produces these

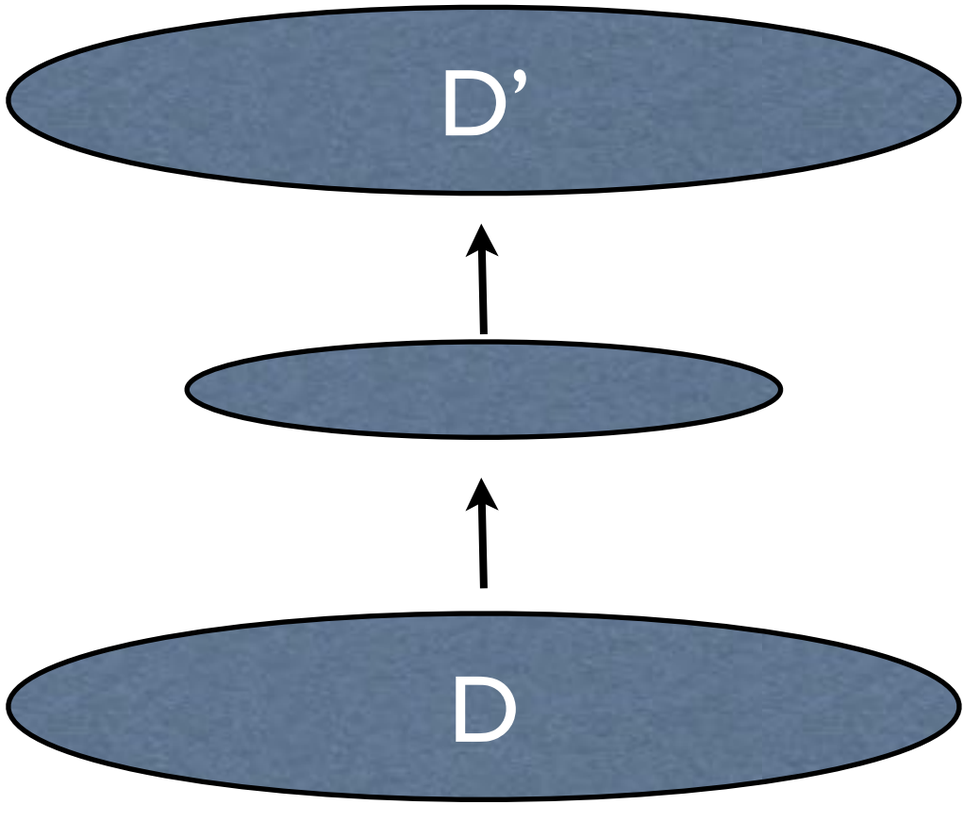


Original

Degraded

Reconstruction

Often: unsupervised methods may be used (simple association)



Implementing non-linear filtering for DSP

Prototype extraction and Noise removal

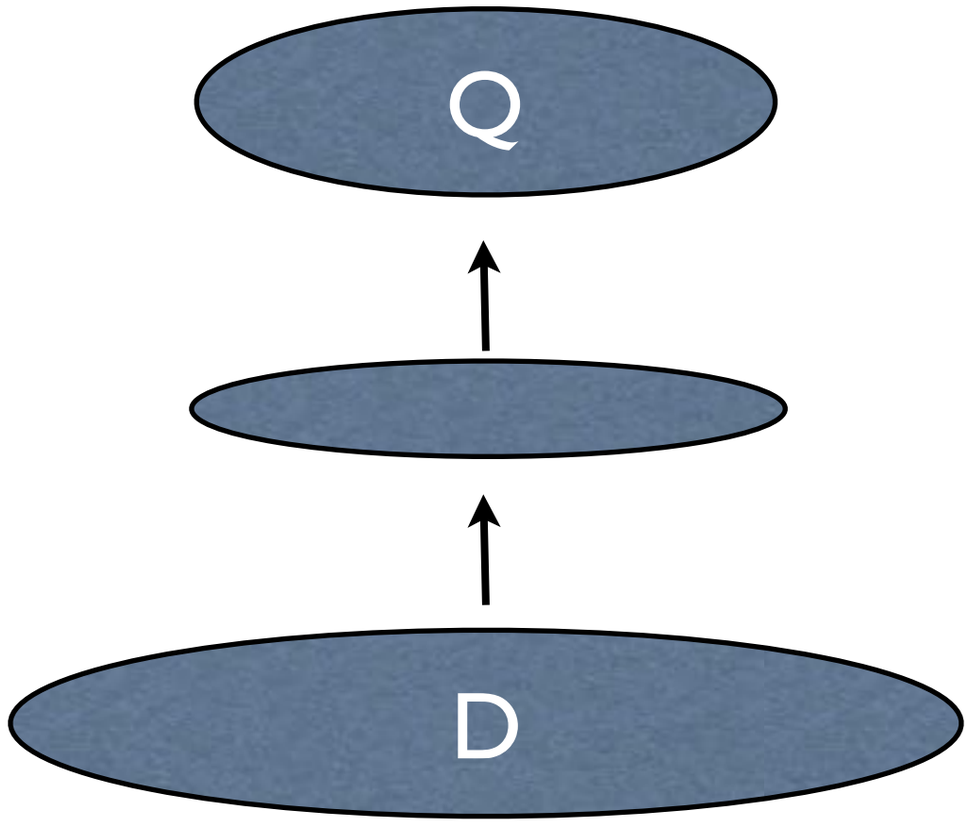
From a 'cognitive' viewpoint, this may be derided as 'associationism'

Can you think of plausible roles for this kind of model?

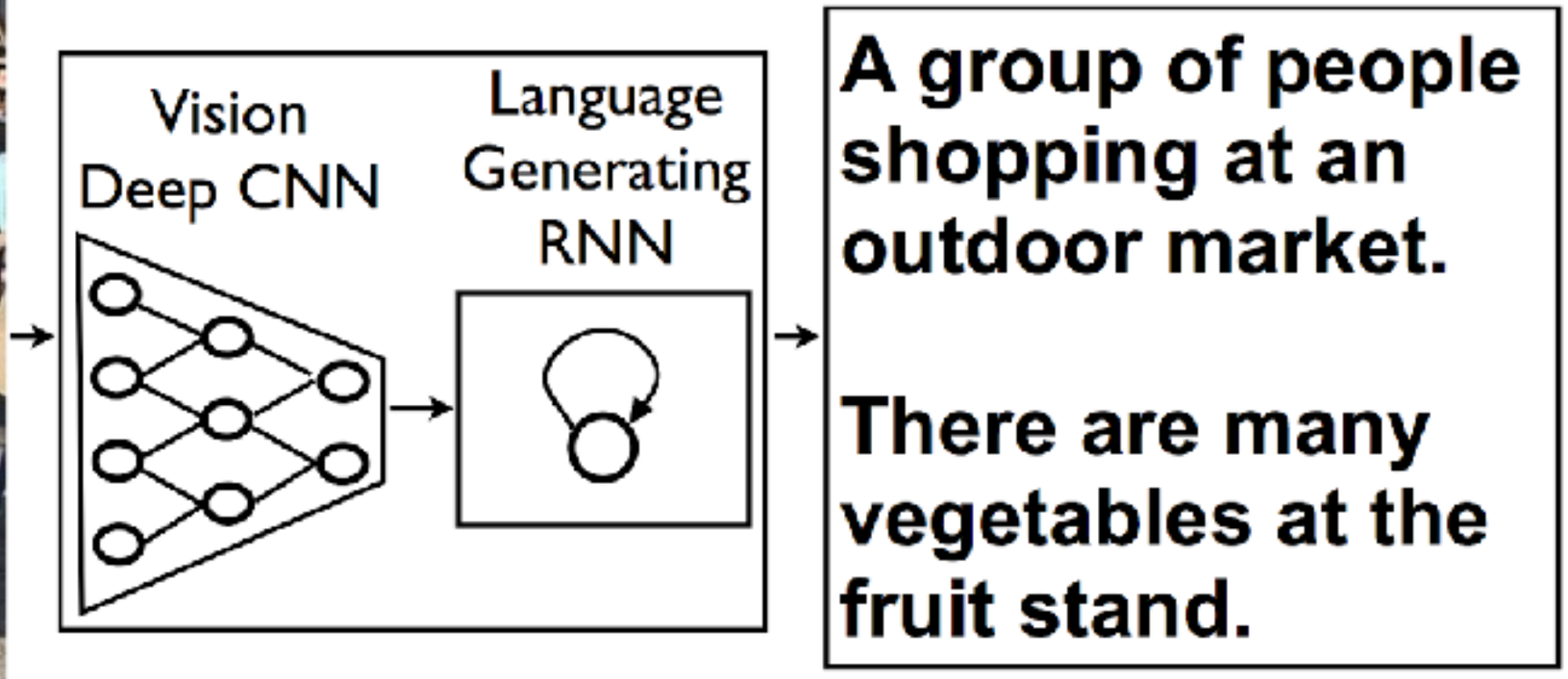
2: Mapping from one domain, D, to an entirely different, continuous, domain

One use is for non-linear multiple regression

Another example: mapping from sensory input to motor output



Startling recent result from Google

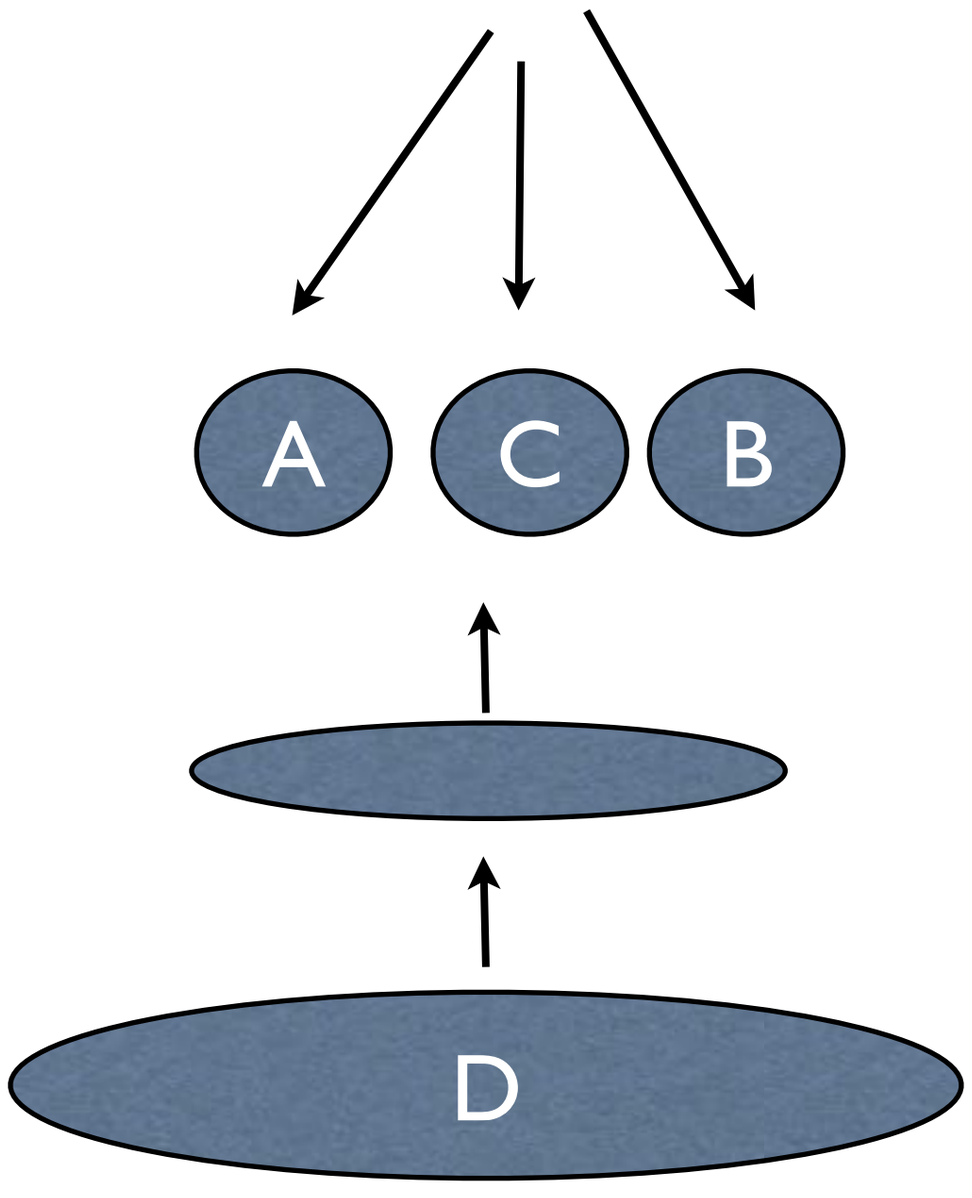


3: Mapping to a discrete, categorical domain

Common situation: we want outputs to be interpretable as probability estimates that the input belongs to 1 of *n* discrete and non-overlapping categories.

Outputs should all lie in (0,1) and should sum to 1.

Mutually exclusive categories



Classification 1: 2 output classes

First stab: Use a single linear output unit, with targets of 1 and 0 for the two classes. Classify based on a *discriminant value* of 0.5.

Problem: Much of our training effort will lie in approximating the target values (0,1) when all we care about is which side of the discriminant we are on. So: use a sigmoidal output function (logistic function):

$$f(u) \equiv \frac{1}{1+e^{-u}} \qquad f'(u) \equiv f(u)(1-f(u))$$

Some details: when using a non-linear output function, we should choose an appropriate non-linear error function. For logistic output units, we use the *cross-entropy* error function:

$$E = -t \ln y - (1-t) \ln(1-y)$$

...which has a particularly simple 'delta' function:

$$\frac{\partial E}{\partial net} = \ldots = y - t$$

Classification 2: Multinomial Classification

Simple generalization: for *n* classes, use *n* output units. Error is based on cross-entropy, output function is the *softmax* output function

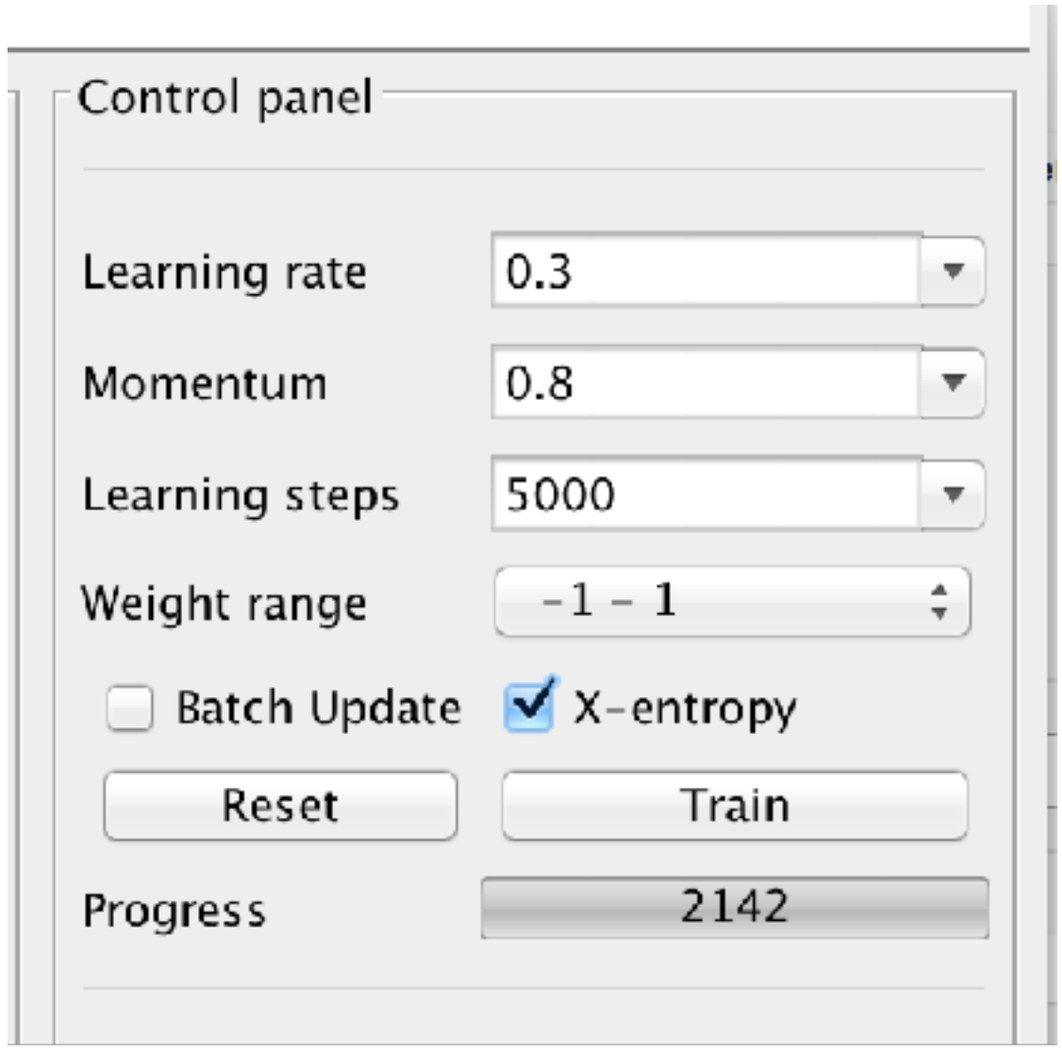
$$f(net_i) = \frac{e^{net_i}}{\sum_o e^{net_o}}$$

And again (fortunately):

$$\frac{\partial E}{\partial net} = \ldots = y - t$$

Take home lesson:

If you are doing a classification problem, where your outputs should be regarded as probabilities, use the cross-entropy error function.



It would be a good idea at this point to take time out to read the following excellent account of the history of neural network research, and how it has gone into and out of favour:

http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/

And here's a second overview. Both go from the beginning to Deep Learning

http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks

Deep Learning

Recent application of ANNs to hard problems in the field of artificial intelligence & data mining, make use of networks with many layers.

This has become known as Deep Learning, and is very much in vogue, e.g. in Google.

Deep learning typically maps from a low-level input, such as a picture, to a high level *symbolic* representation.

The mapping is through many levels. Mapping from one level to the next may be supervised or unsupervised.

The return of the hype:

"the Building Google Brain"

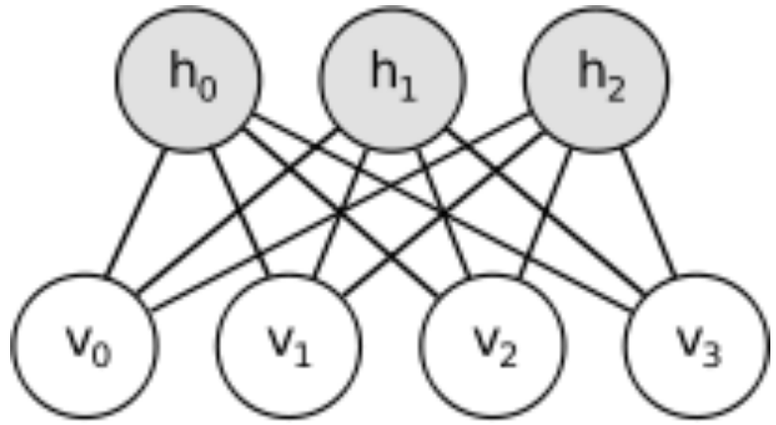
Deep Learning Revives the Neural Net Revolution

Geoff Hinton

One common approach.

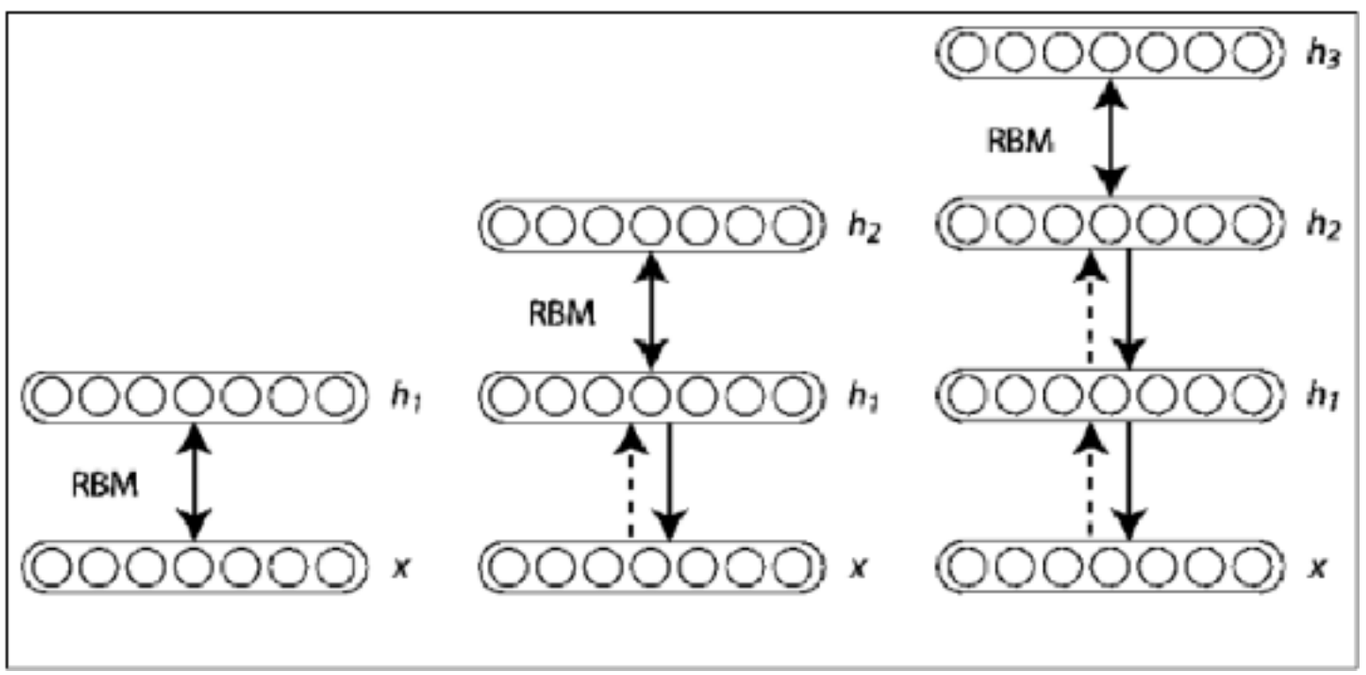
Start with a big data set of richly structured data (e.g. images).

Use unsupervised learning (restricted Boltzmann machines) to learn statistical features of this set.

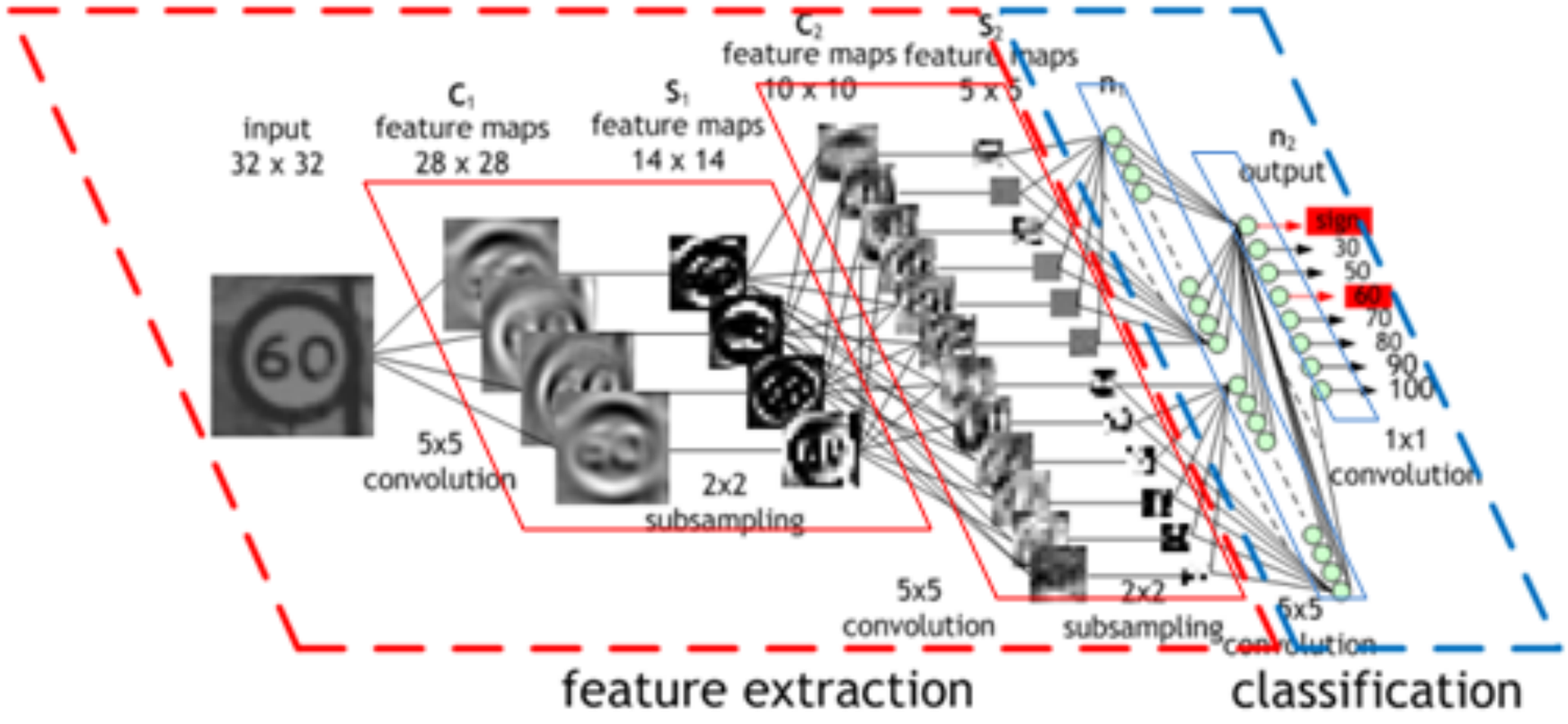


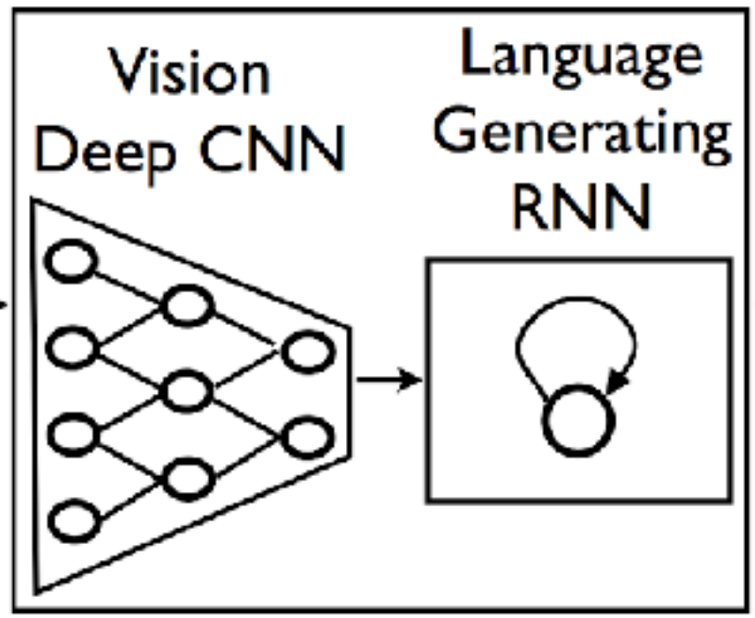
Then learn features from this layer in further layer.

Repeat several times.



After training a stack of such layers, the final representation can be used as input to a fairly standard neural network, e.g. to do classification.

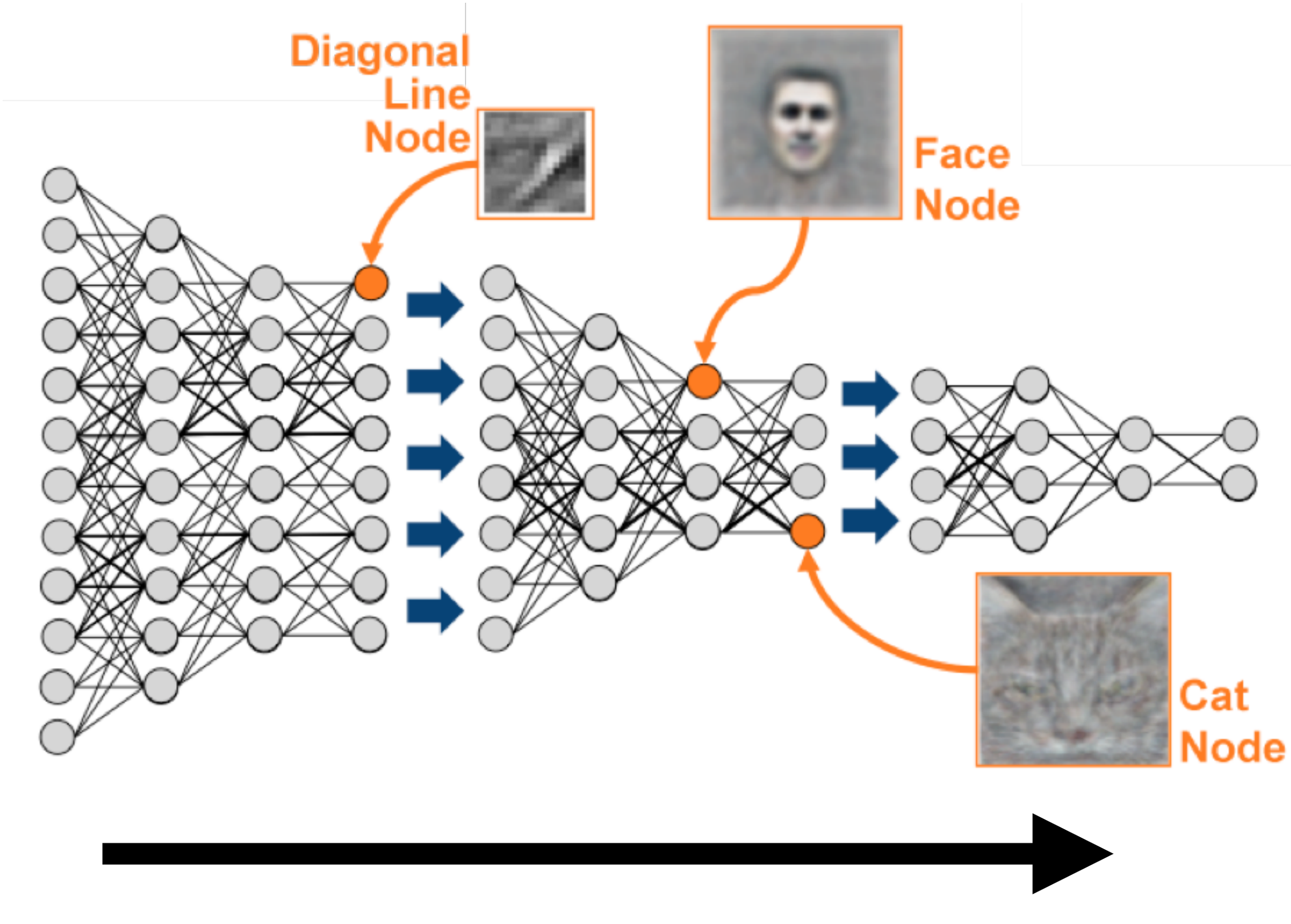


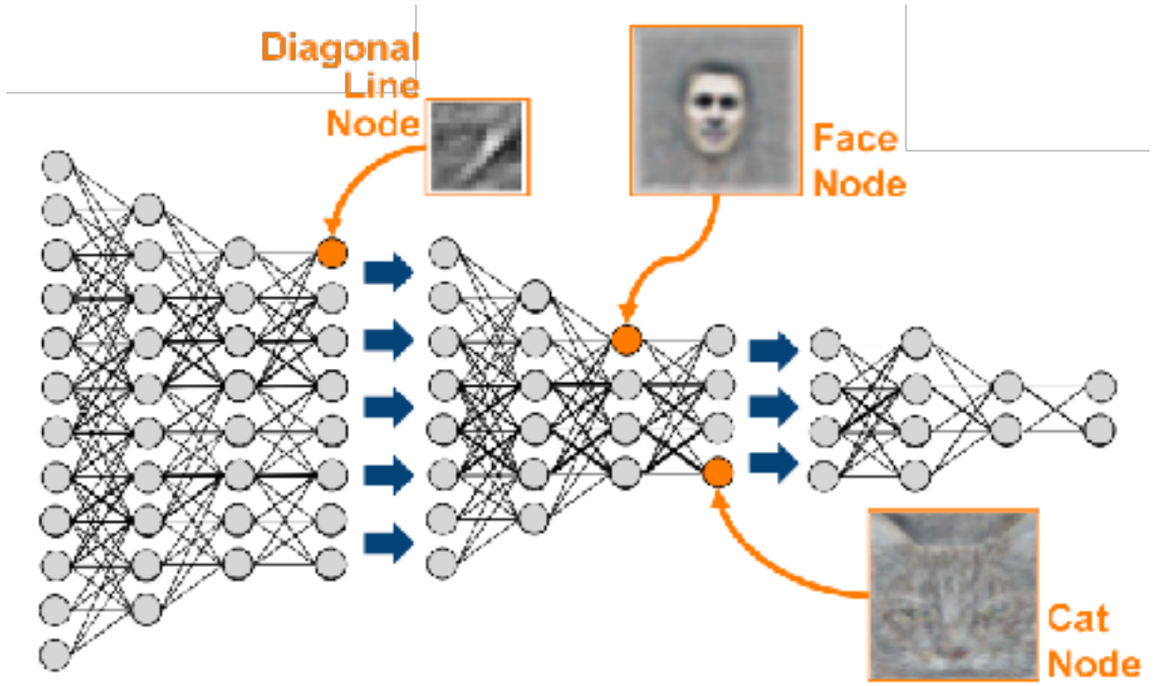


A group of people shopping at an outdoor market.

There are many vegetables at the fruit stand.

“Deep Learning” Network





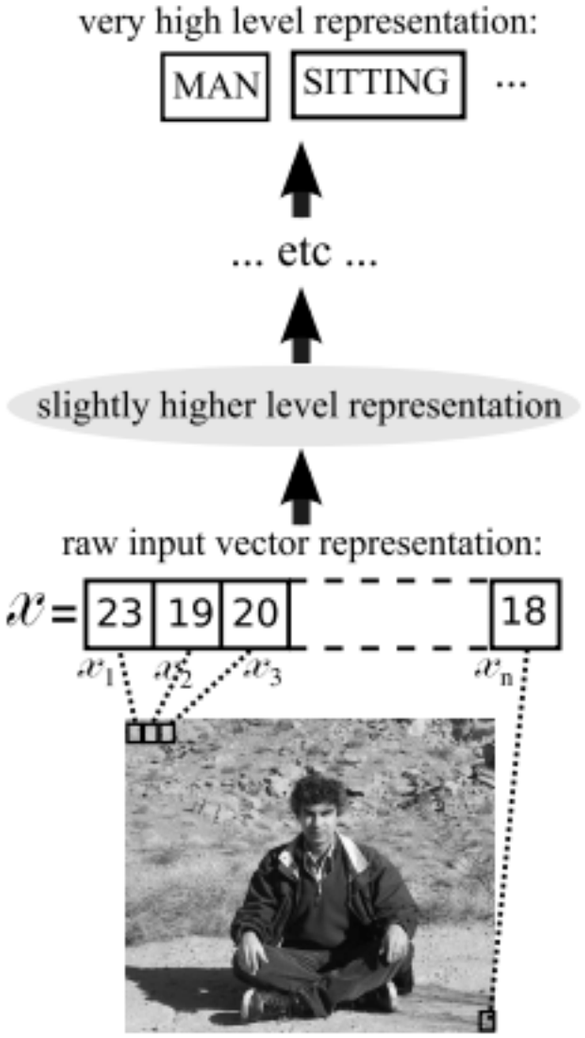
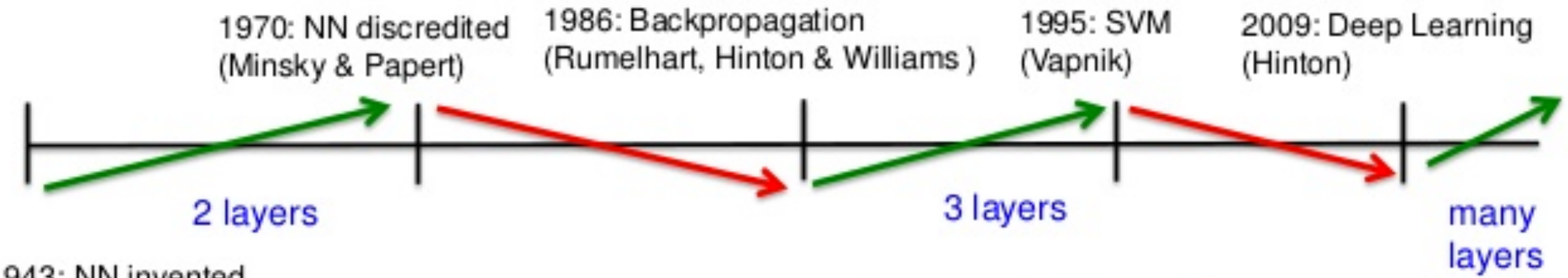


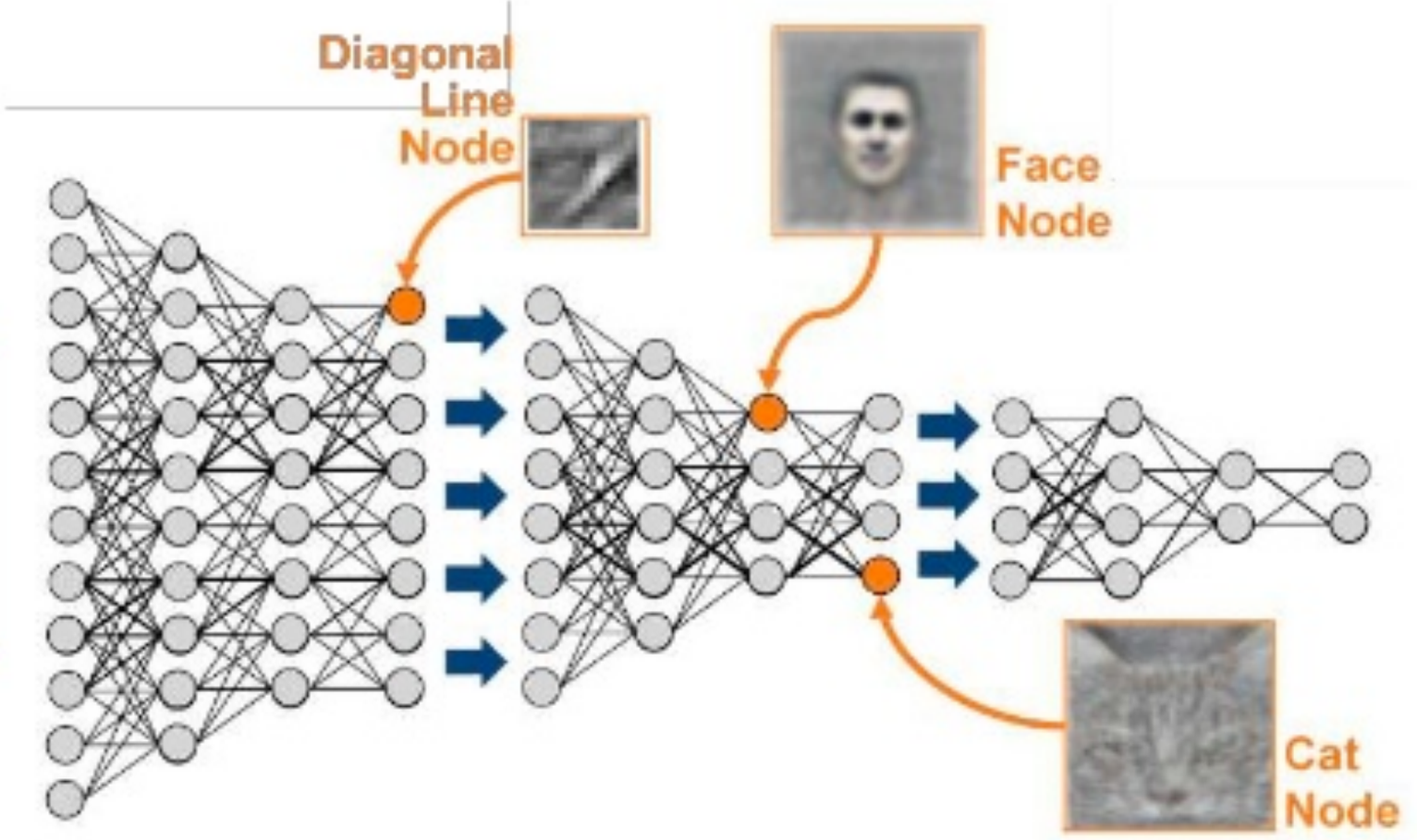
Figure 1: We would like the raw input image to be transformed into gradually higher levels of representation, representing more and more abstract functions of the raw input, e.g., edges, local shapes, object parts, etc. In practice, we do not know in advance what the "right" representation should be for all these levels of abstractions, although linguistic concepts might help guessing what the higher levels should implicitly represent.

Food for thought: how does deep learning relate to the view of the nervous system offered by Hubel and Wiesel?

Deep Learning: Neural Nets Strike Back(again)



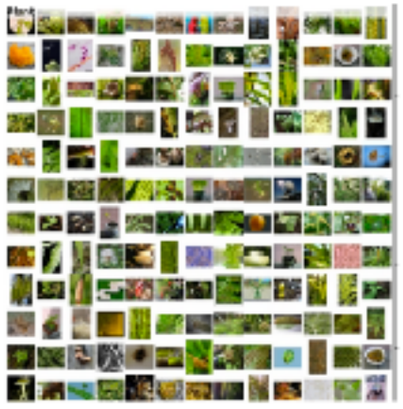
-Model Size: 10B parameters
-Used by: Yahoo!, Google, Microsoft, Baidu, IBM, Scyfer ☺



Google's famous neural-net learned cat. This is the optimal input to one of the neurons. (Source)

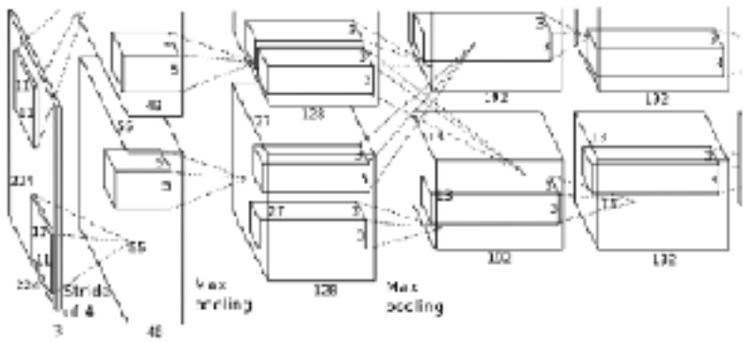
Deep Learning came about because of a conjunction of several elements:

The Deep Learning "Computer Vision Recipe"



Big Data: ImageNet

+



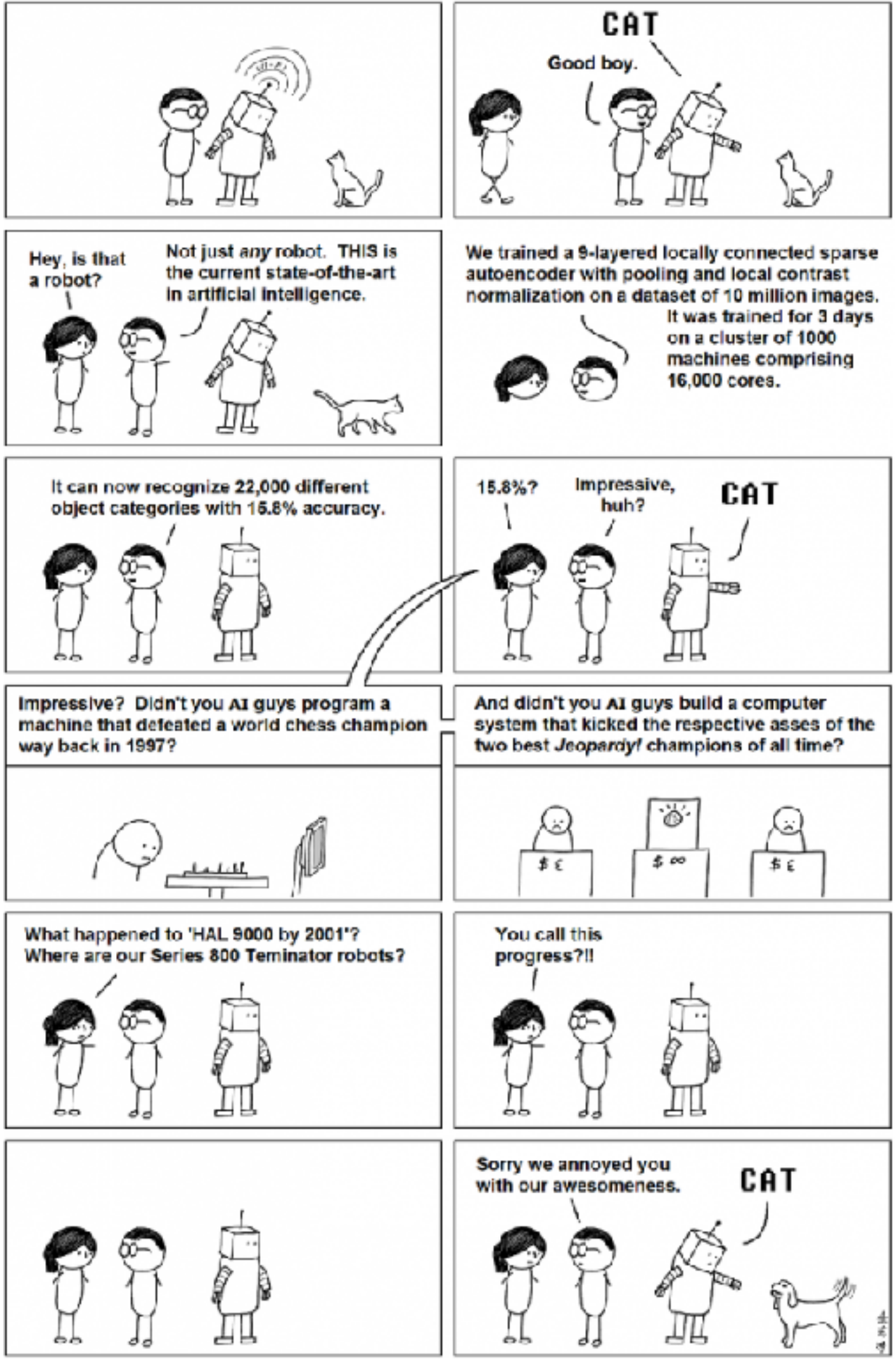
Deep Convolutional Neural Network

+

Backprop on GPU

=

Learned Weights


CAT
Good boy.
Hey, is that a robot?
Not just *any* robot. THIS is the current state-of-the-art in artificial intelligence.
We trained a 9-layered locally connected sparse autoencoder with pooling and local contrast normalization on a dataset of 10 million images.
It was trained for 3 days on a cluster of 1000 machines comprising 16,000 cores.
It can now recognize 22,000 different object categories with 15.8% accuracy.
15.8%?
Impressive, huh?
CAT
Impressive? Didn't you AI guys program a machine that defeated a world chess champion way back in 1997?
And didn't you AI guys build a computer system that kicked the respective asses of the two best *Jeopardy!* champions of all time?
What happened to 'HAL 9000 by 2001'? Where are our Series 800 Teminator robots?
You call this progress?!!
Sorry we annoyed you with our awesomeness.
CAT

Big Overarching Topic: Representations and Features

Representations

localist

<1 0 0 0 0 0 0 0>

<0 1 0 0 0 0 0 0>

<0 0 1 0 0 0 0 0>

<0 0 0 1 0 0 0 0>

<0 0 0 0 1 0 0 0>

distributed

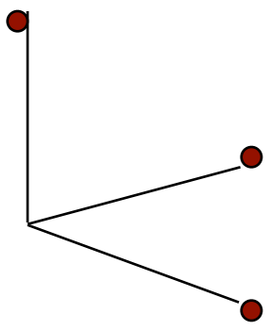
<0.2 0.5 0.1 0.4 0.5 0.9 0.1 0.5>

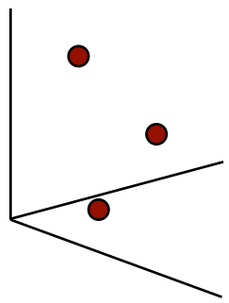
<0.4 0.4 0.2 0.6 0.4 0.7 0.2 0.1>

<0.3 0.7 0.2 0.5 0.9 0.9 0.5 0.7>

<0.1 0.8 0.1 0.4 0.3 0.2 0.6 0.7>

<0.6 0.4 0.7 0.2 0.6 0.3 0.4 0.1>





robustness

graceful degradation

similarity relationships

- Localist = 1 unit responds to 1 thing (e.g., digits, grandmother cell).

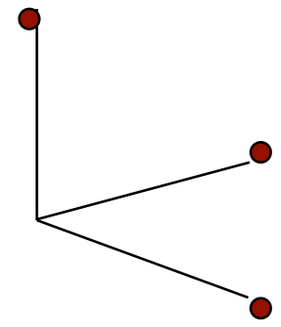
- Distributed = Many units respond to 1 thing, one unit responds to many things.

- With distributed representations, units correspond to stimulus *features* as opposed to complete stimuli

Note: the localist vs distributed representation issue applies really only to feedforward, input/output mapping networks. If we regard RNNs as dynamical systems, then the same question appears rather differently: how rich is our state description. More on this later….

Localist Representations

Assume a 3-D input space:

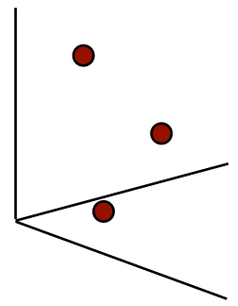


How similar are these vectors:
(0,1,0) and (1,0,0)
(0,0,1) and (0,1,0)

$$d = |x - y| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

Euclidian distance

Distributed Representations



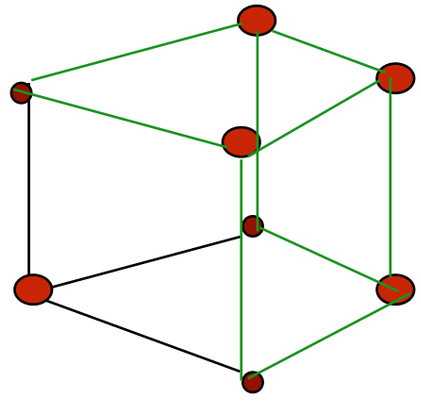
How similar are these vectors?

(0.5,0.2,0.8) and (0.3,0.4,0.2)
(0.5,0.2,0.8) and (0.9,0.1,0.6)

$$d = |x - y| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

Distributed Representations

Here we allow each input to be 0 or 1



We have a much more constrained form of distributed representation.

What distance measure might we use here?

2 possible encodings:

1: all values are 1 or 0
...corresponds to the corners of a hypercube
.... with *n* bits, we can represent 2^*n* patterns
.... distances between patterns fall into a few discrete sets.

2: all real values between 0 and 1 are allowed
...corresponds to the entire volume of the hypercube
...arbitrary precision/arbitrary closeness between patterns

Distributed Representations: some properties

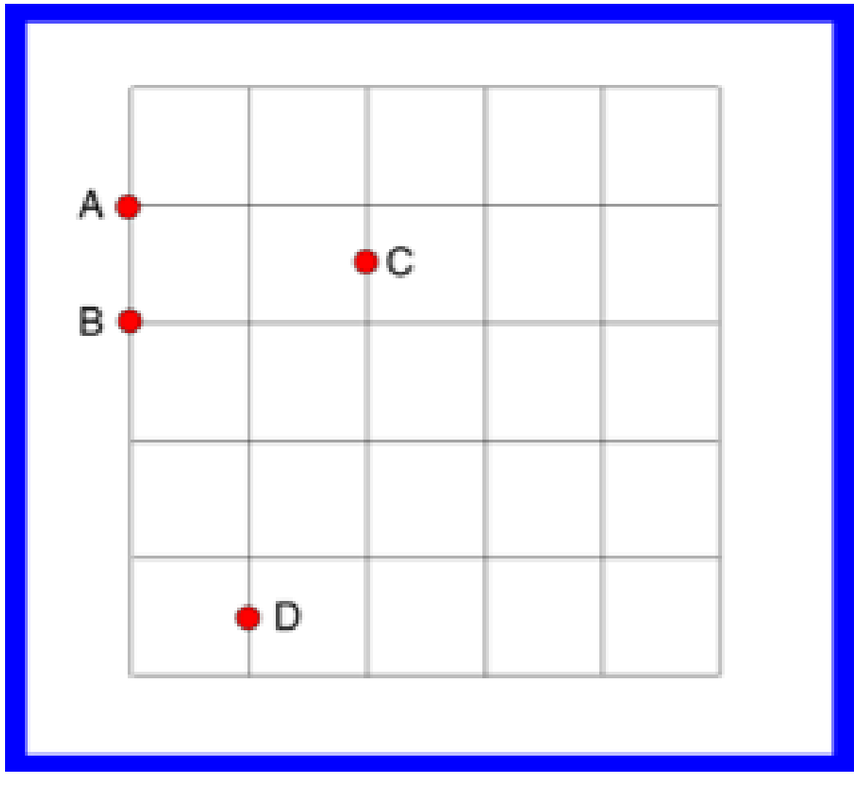
- Graceful Degradation/Robustness
- Similarity and Generalization
- Content Addressability
- Pattern Completion
- Structured Mapping

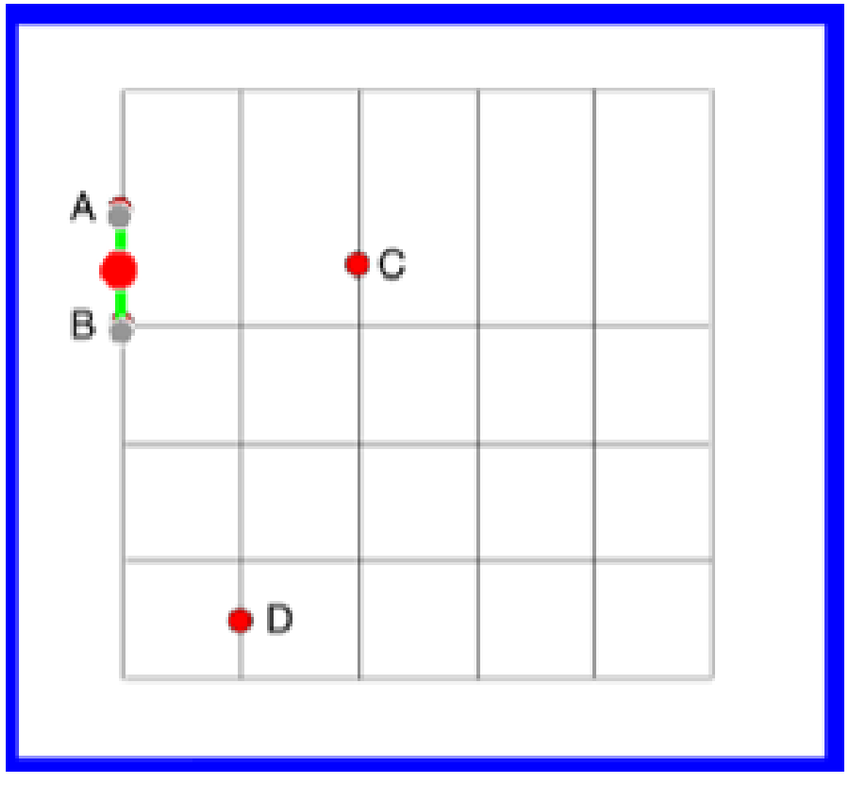
Topic: Clustering to express similarity relations

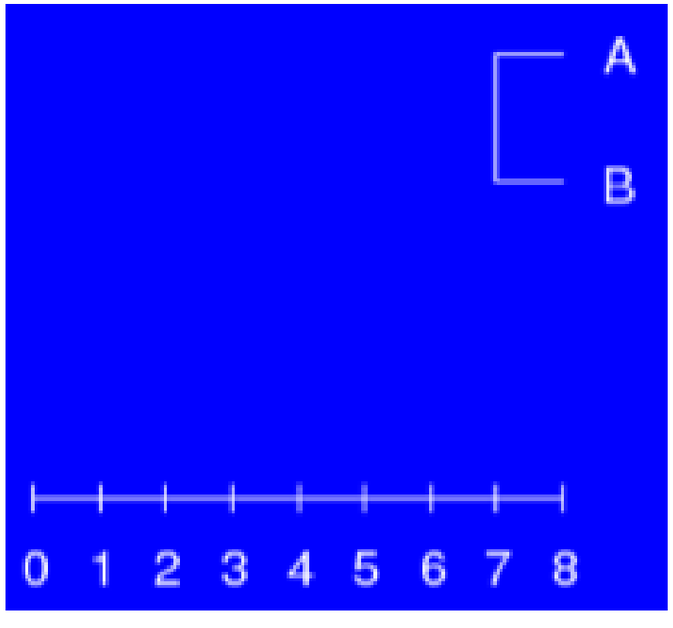
Both data sets, and sets of network activations, are encountered as big sets of numbers. Similarity relations within these sets of numbers determine the properties of the data sets.

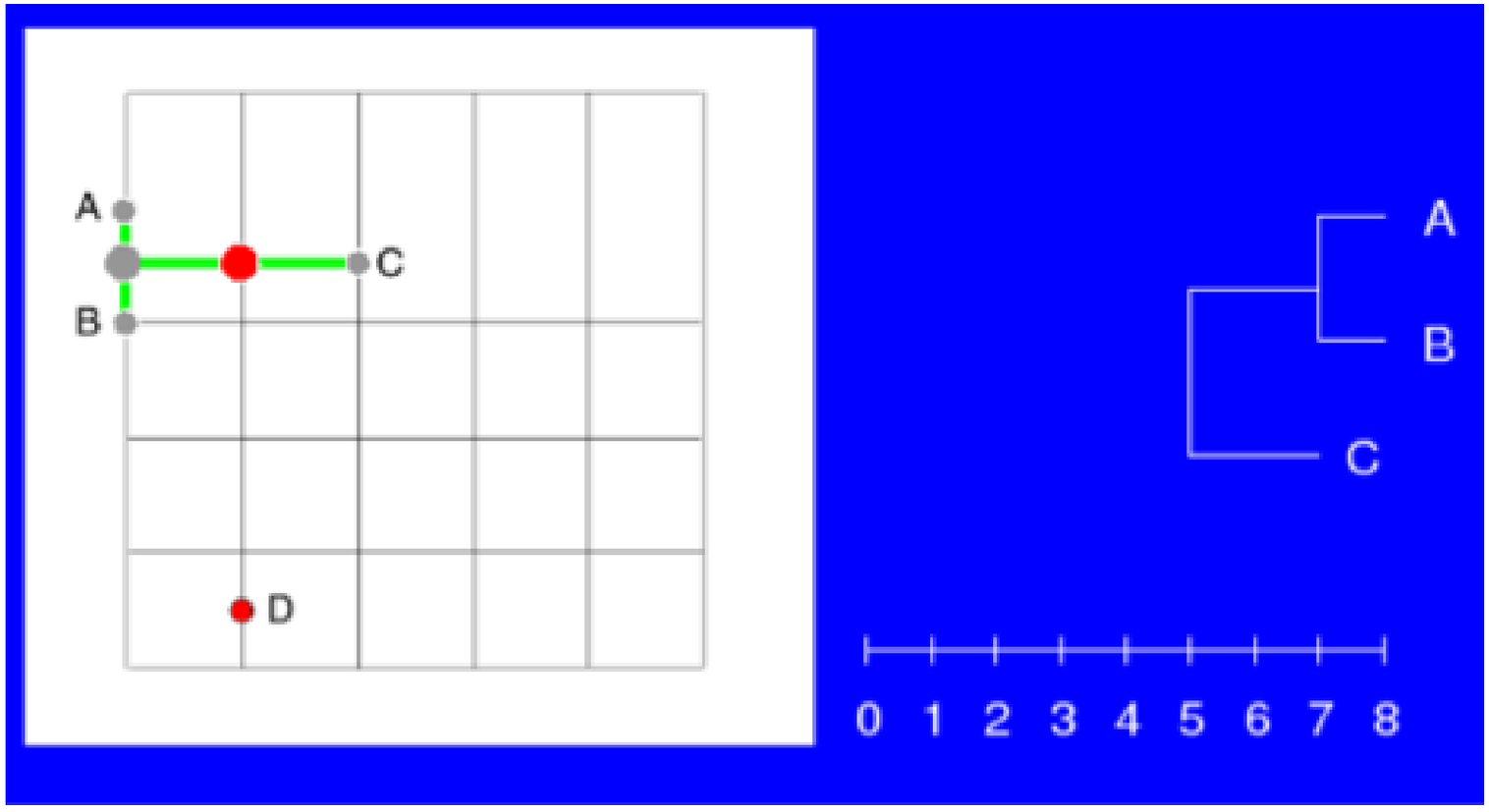
We need to be able to derive a way of looking at similarity relations among vectors.

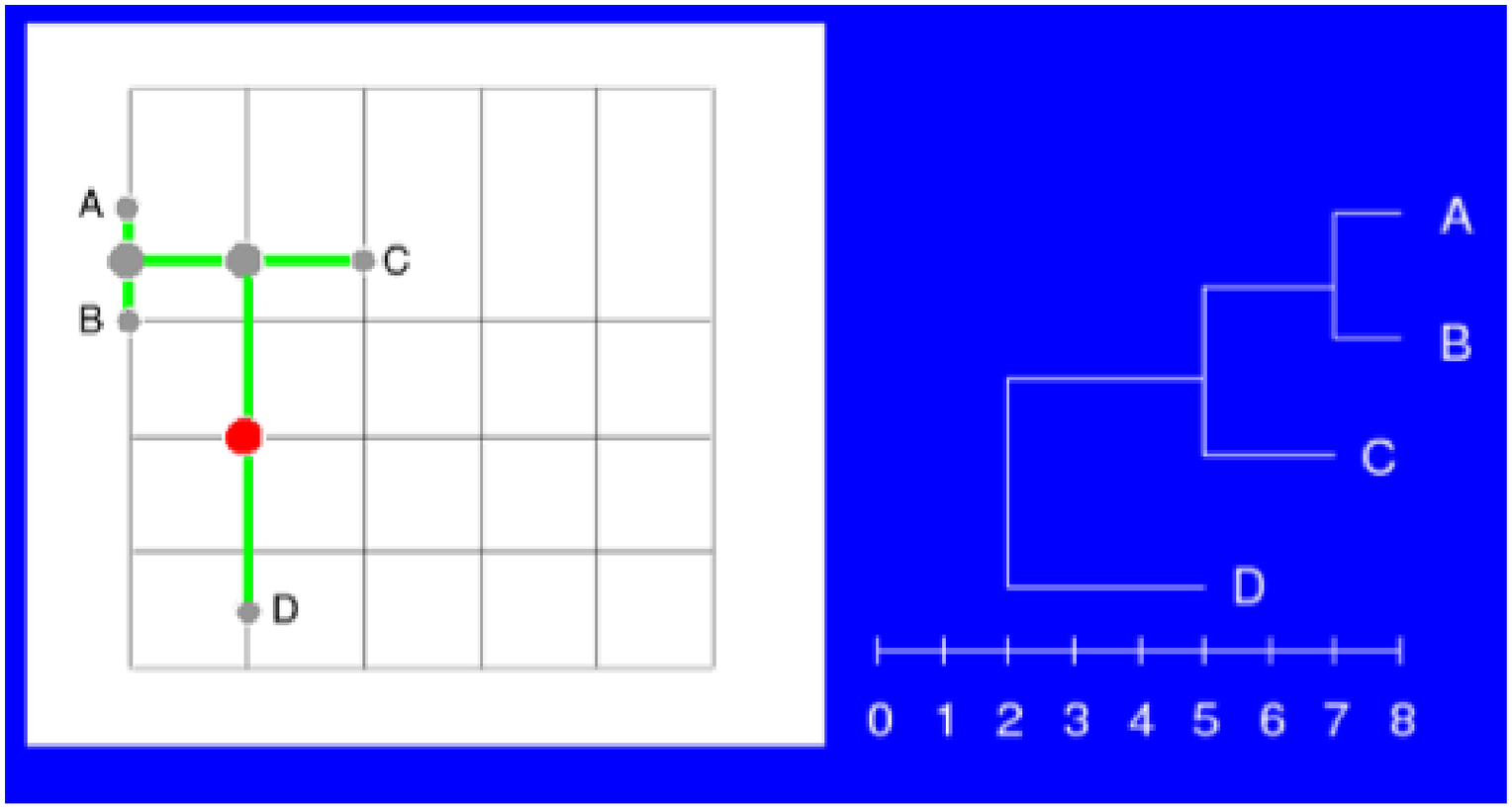
One such technique is Hierarchical Clustering.


A
B
C
D


A
B
C
D

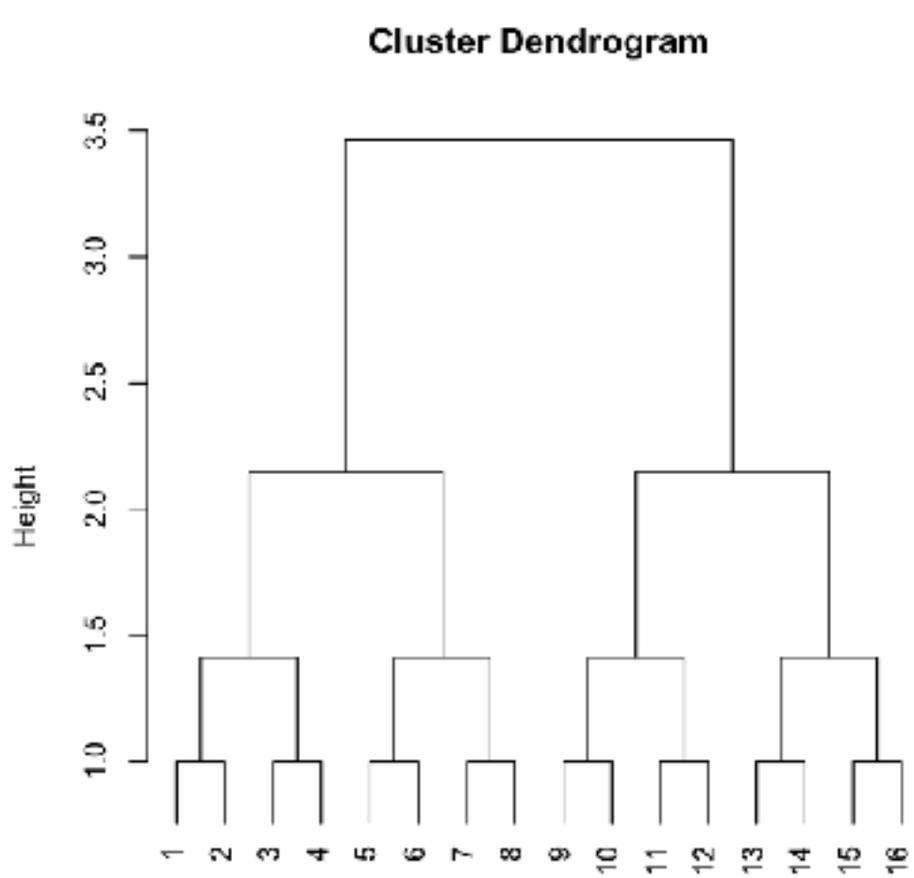

A
B
0 1 2 3 4 5 6 7 8


A
B
C
D
A
B
C
0 1 2 3 4 5 6 7 8


A
B
C
D
A
B
C
D
0 1 2 3 4 5 6 7 8

We will experiment with hierarchical clustering in Lab 5.

As an exercise, what can you say about the set of patterns that cluster like this?



Topic: Horses for Courses: Developing Appropriate Representations

Perhaps the first and most important decision we make in developing a connectionist model is to decide how the elements that will feature within our model are represented.

This does not necessarily imply any commitment to a theory of cognitive representation, though you might choose to make that link.

Example: encoding phonemes

Sejnowski and Rosenberg (1987): Nettalk

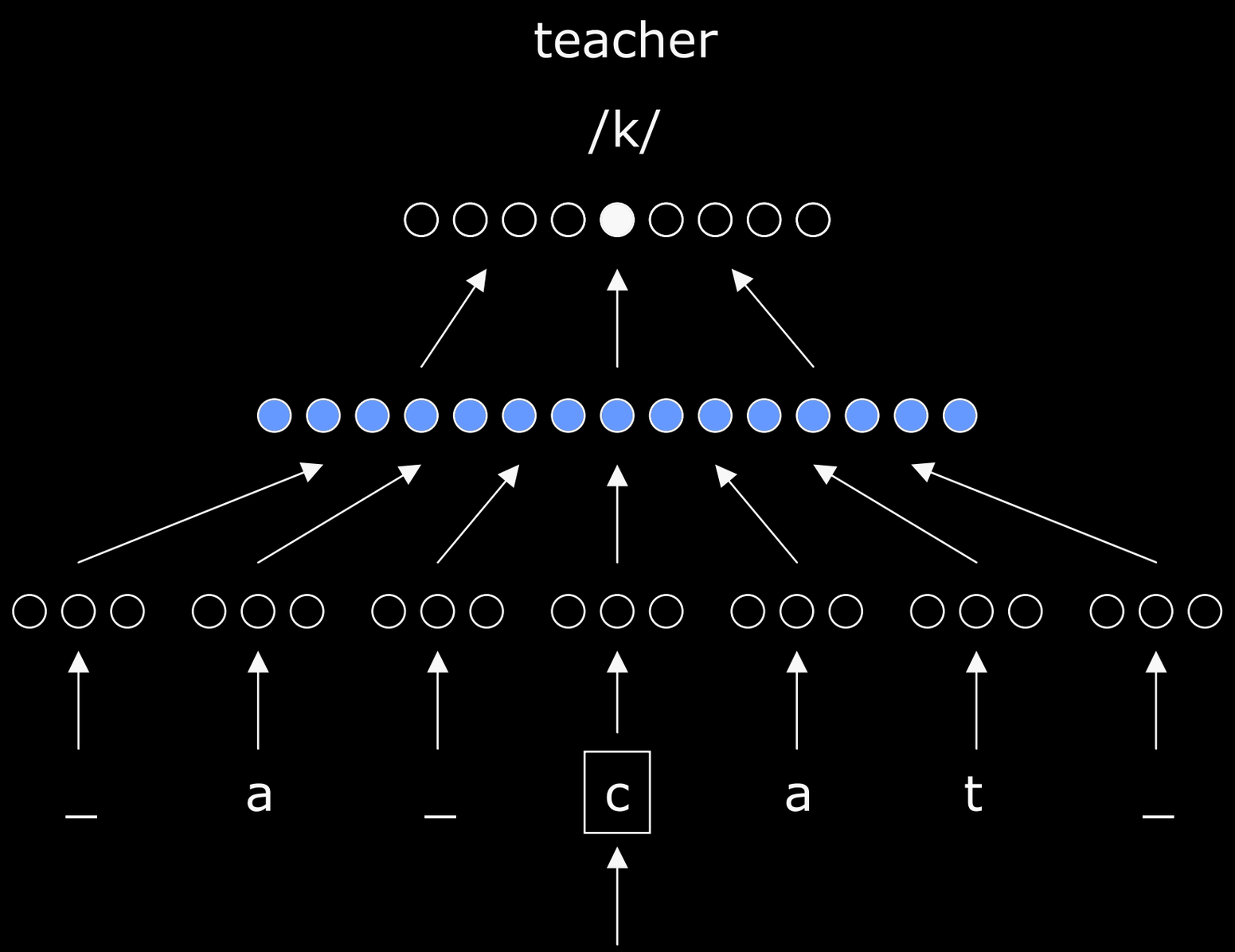
Three-layer network trained to 'read' English text; i.e. to map from spelling to phonemes

"caught" => /kɔt/
"laugh" => /læf/

Each presentation (input) consists of 7 letters: target letter with three letters on either side.

[_ c a U g h t]
localist encoding
(26 + 3 units) * 7

The NETtalk Network



(after Hinton, 1989)

AR35, Seth Bullock, 2001

Feature-based output encoding

There are approximately 43 distinct sounds in most varieties of English. These are the *phonemes* of the language. Phonemes function contrastively:

/pæt/ vs /bæt/

Phoneme is an abstract notion. /p/ is realized differently at the beginning and end of syllables.

English orthography is a poor guide to pronunciation. Nettalk maps from a noisy orthography to a more principled phonemic spelling. This is only one small step in the chain from text to speech.....

Similarity among phonemes:

Phonemes exhibit organization, with similarity based on place, type and manner of articulation

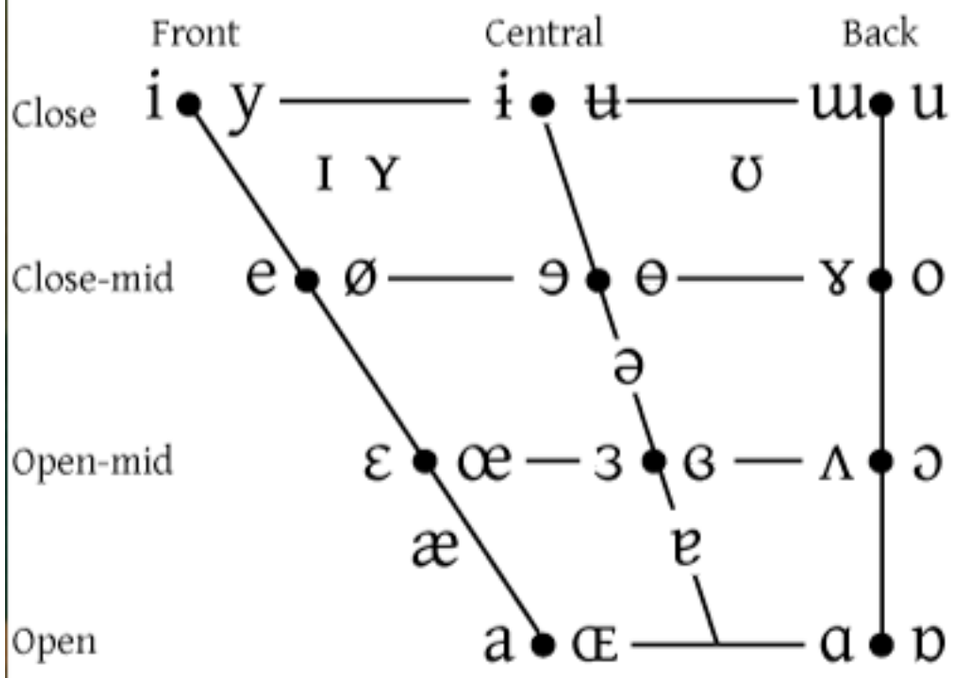
In many phonological theories, this is captured by a discrete set of features. Each phoneme is a set of features (sometimes an unordered bag, sometimes a hierarchically structured collections... pick your theory)

Nettalk had 26 outputs, corresponding to 23 articulatory features + 3 to encode stress and syllable boundaries.

CONSONANTS (PULMONIC)

	Bilabial	Labiodental	Dental	Alveolar	Postalveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal	Glottal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap or Flap				ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lateral fricative				ɬ ɮ							
Approximant		ʋ		ɹ		ɻ	j	ɰ			
Lateral approximant				l		ɭ	ʎ	ʟ			

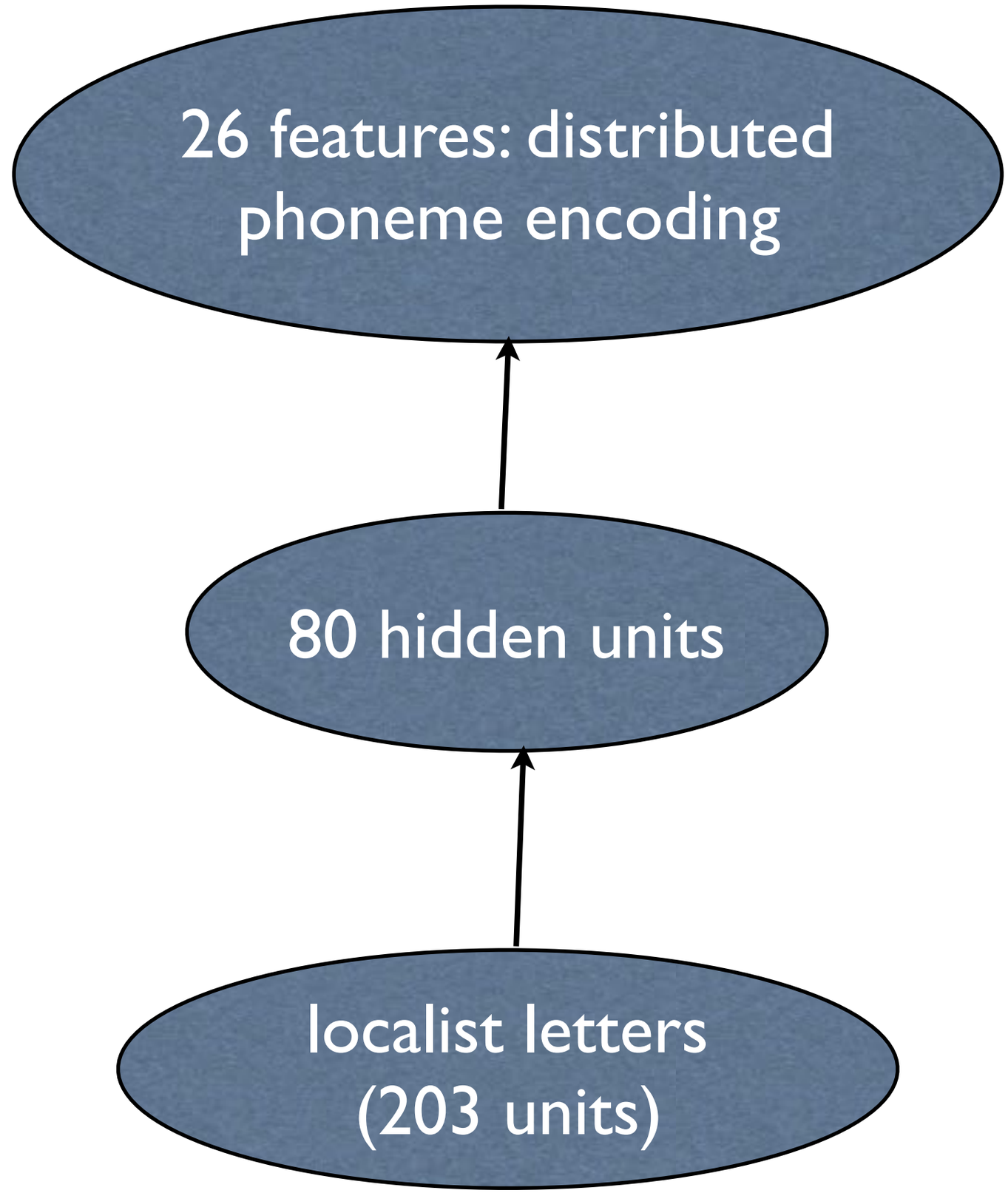
VOWELS



English Consonants

p	b		m		f		v	
					θ		ð	
t	d		n		s	ʃ	z	ʒ
k	g		ŋ					
		ɹ						
		l						

Nettalk


26 features: distributed phoneme encoding
80 hidden units
localist letters (203 units)

Things to consider:

Why is the letter representation localist and the phoneme representation distributed?

What theory drives the choice of one form of representation over another??

Topic: Invariance and Representation

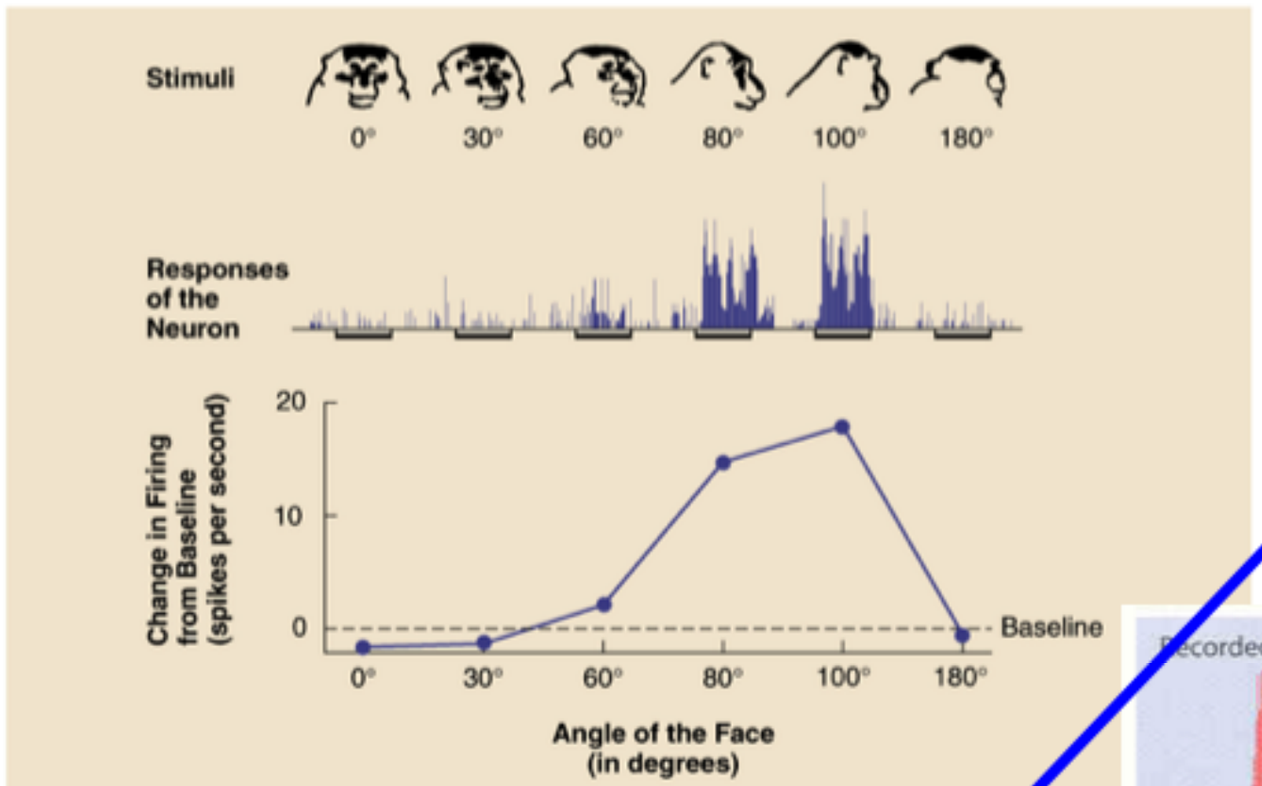
Perceptual Invariance: Different appearances of an object can be perceived as ``the same", despite, e.g., changes in position or illumination, distortions, or partial occlusion by other objects.

Huge challenge for artificial systems

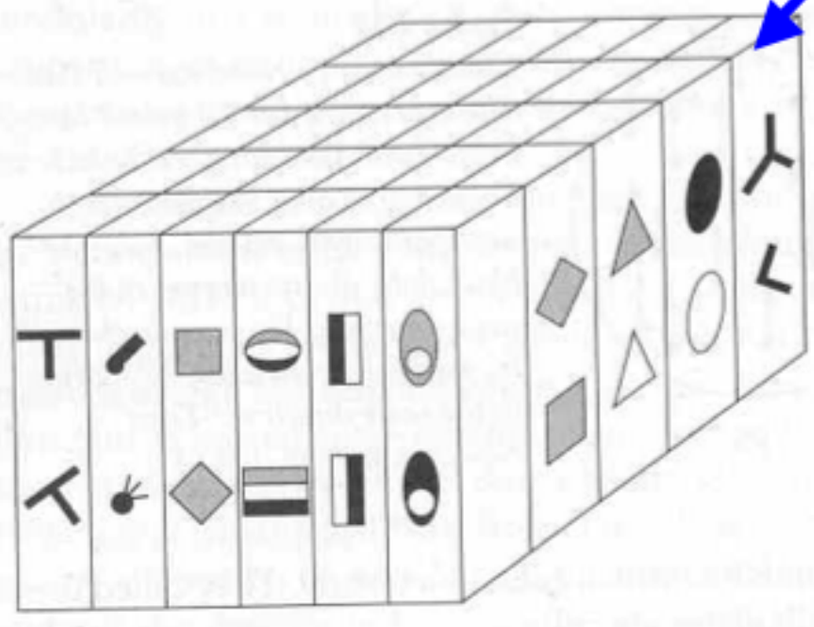
Need to develop representations which are insensitive to ‘surface’ variation and which correspond to the ways in which we carve up the flux of existence.

Tuning in monkey IT

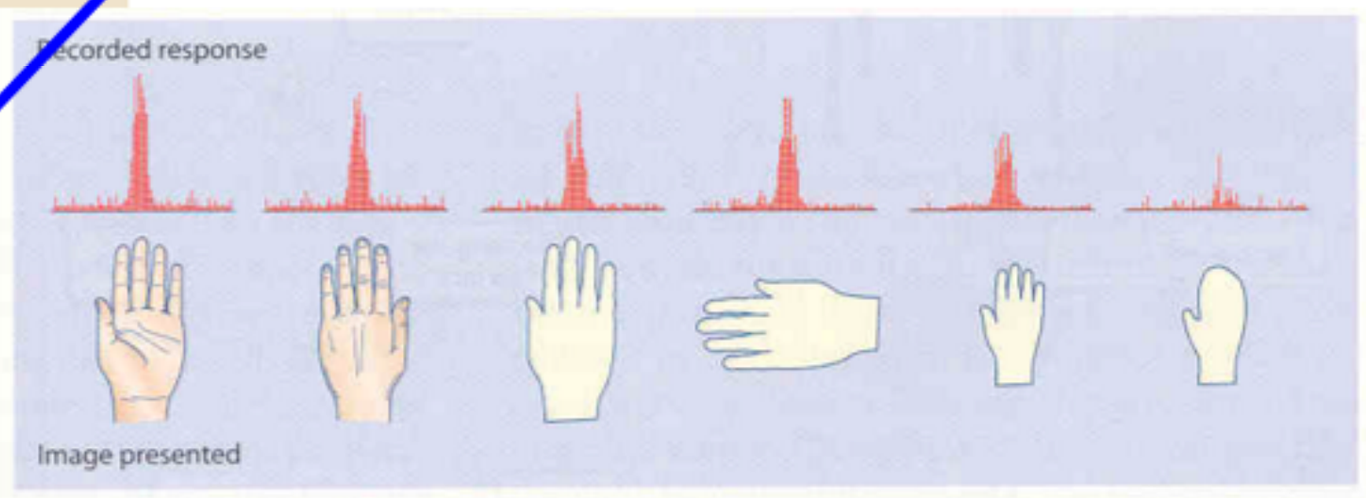
► Firing Rate of a Monkey Inferotemporal Cortex Neuron

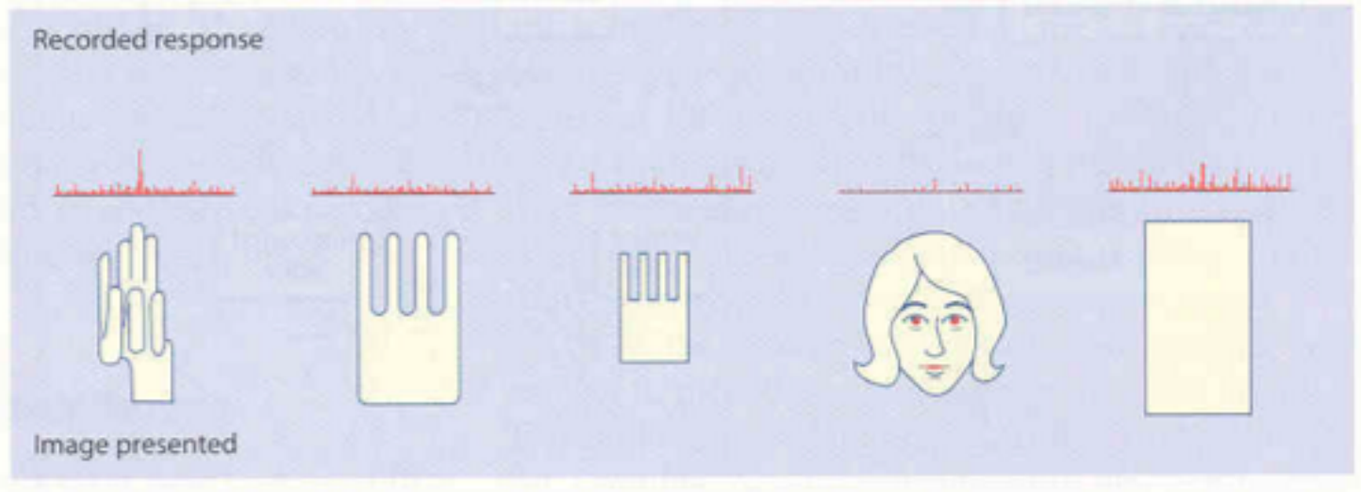


Source: Adapted from Gross et al., *Pattern Recognition Mechanisms*, p. 179-201. Berlin: Springer Verlag 1985.



Cells respond to specific stimuli
And are clustered into columns with similar tuning





Building a feature detector:

Does a given feature exist in the input?

Assume you are not primarily interested in *where* a feature is, just in whether it is present or not . . .

Human visual pattern recognition:

We recognize visual elements even though they may be

* At different locations
* Of different sizes
* and in different orientations.

Small example

How could we build a feature detector that has something like this ability?

Consider just changes in location, and make the input space "simple":

Positive	Negative
0 1 1 1 0 0 0 0	0 1 1 0 0 0 0 0
0 1 0 1 1 1 0 0	1 0 1 0 1 0 1 0
1 0 1 0 0 1 1 1	1 1 0 1 1 0 1 1

Imagine you are a hidden unit: what sum do you compute?

How might we train a network to recognize the pattern "...1 1 1..." irrespective of where it occurs?

This is the feature we seek

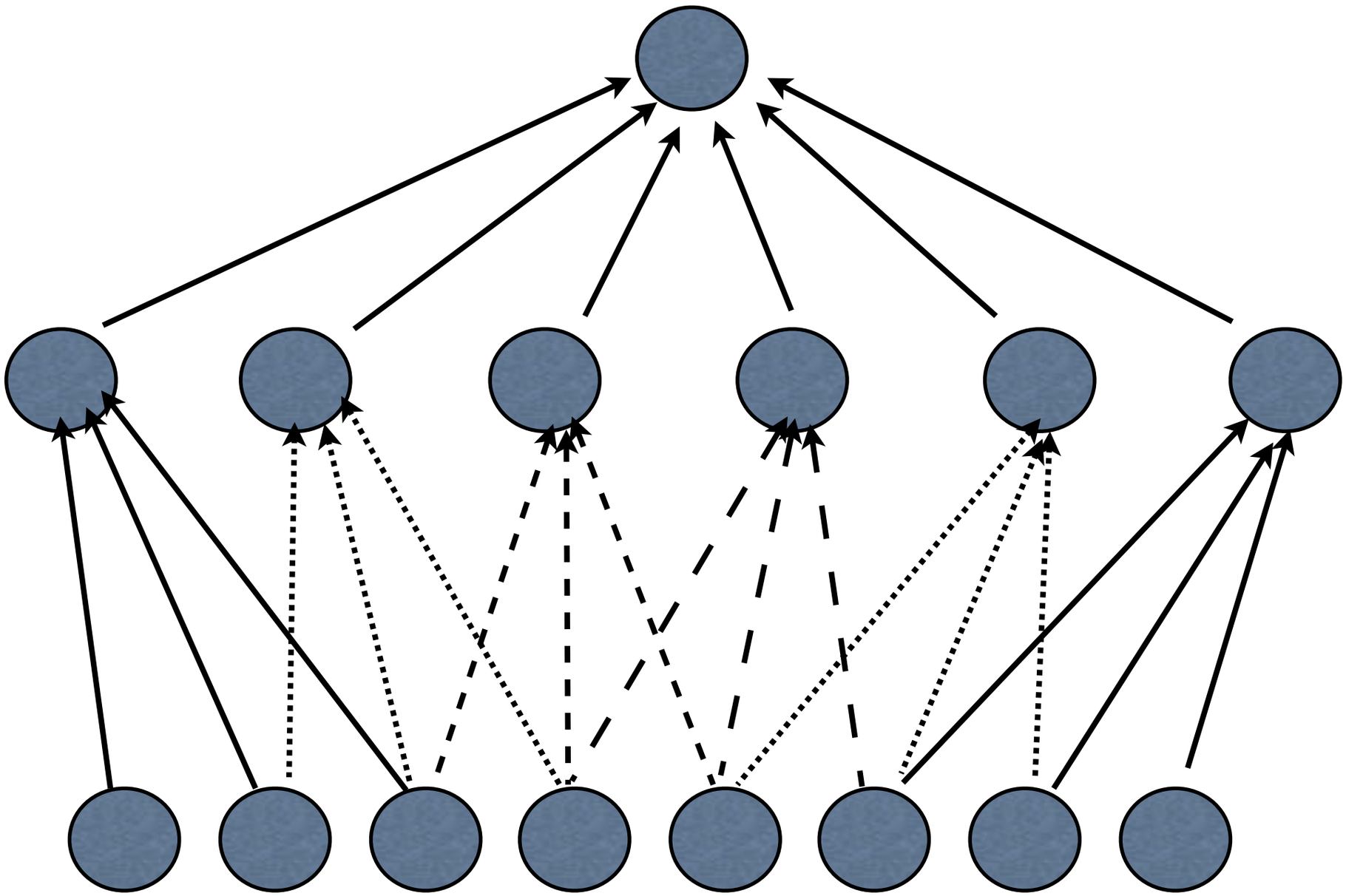
Positive	Negative
0 1 1 1 0 0 0 0	0 1 1 0 0 0 0 0
0 1 0 1 1 1 0 0	1 0 1 0 1 0 1 0
1 0 1 0 0 1 1 1	1 1 0 1 1 0 1 1

Designing a solution:

How much of the solution should we "build in"?
Constant tension between *specificity* and *generality*

For now, we restrict the vision of each hidden node to a portion of the input vector.

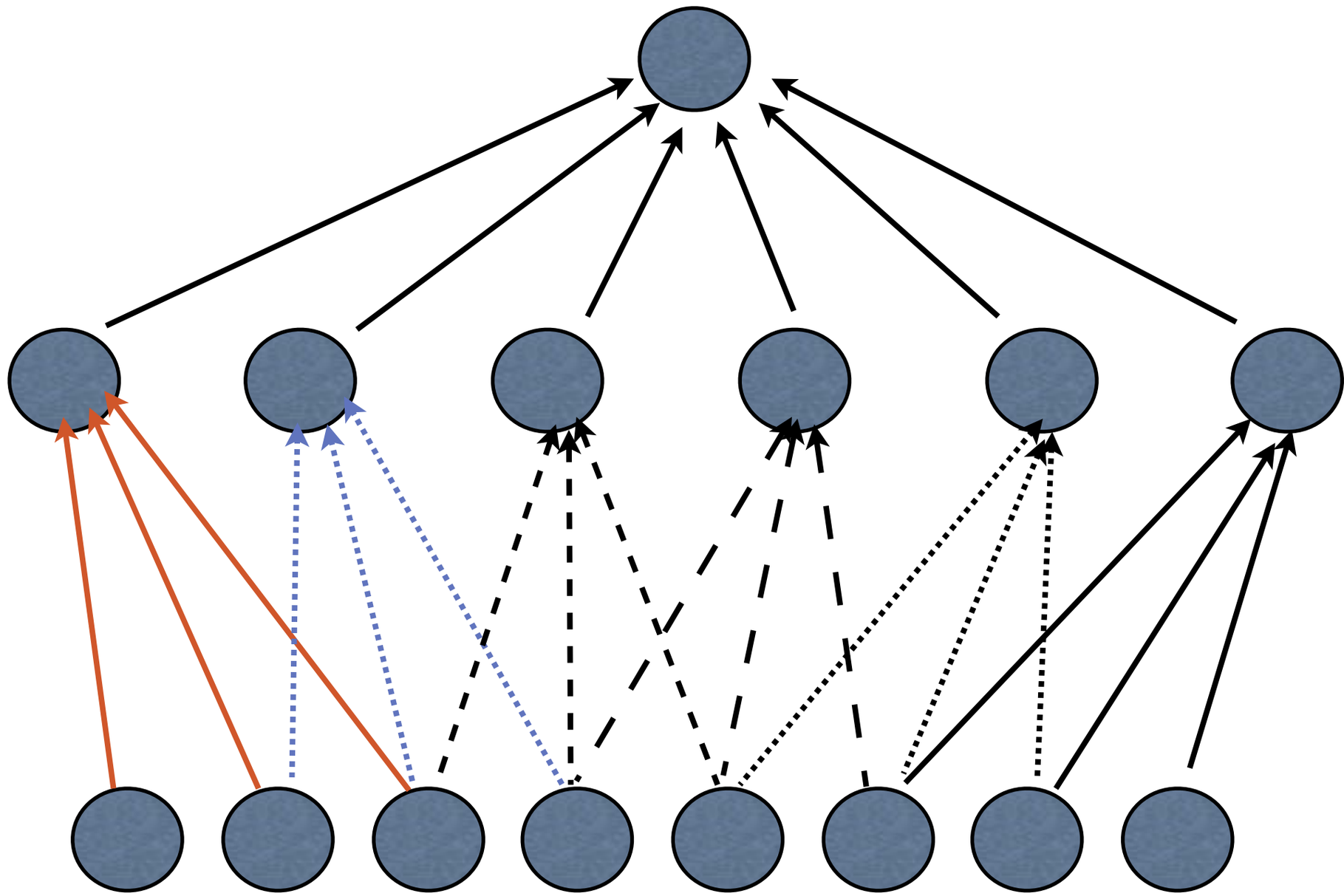
Let each hidden unit 'see' exactly three adjacent inputs.



Goal is shift-invariance, so we want ...1 1 1... to have a comparable effect, no matter where it occurs in the input string

Tactic: constrain each group of 3 weights (+bias) to be identical

All hidden-output wts are likewise identical



Now, whenever ...1 1 1... occurs, *some* hidden unit is maximally activated.

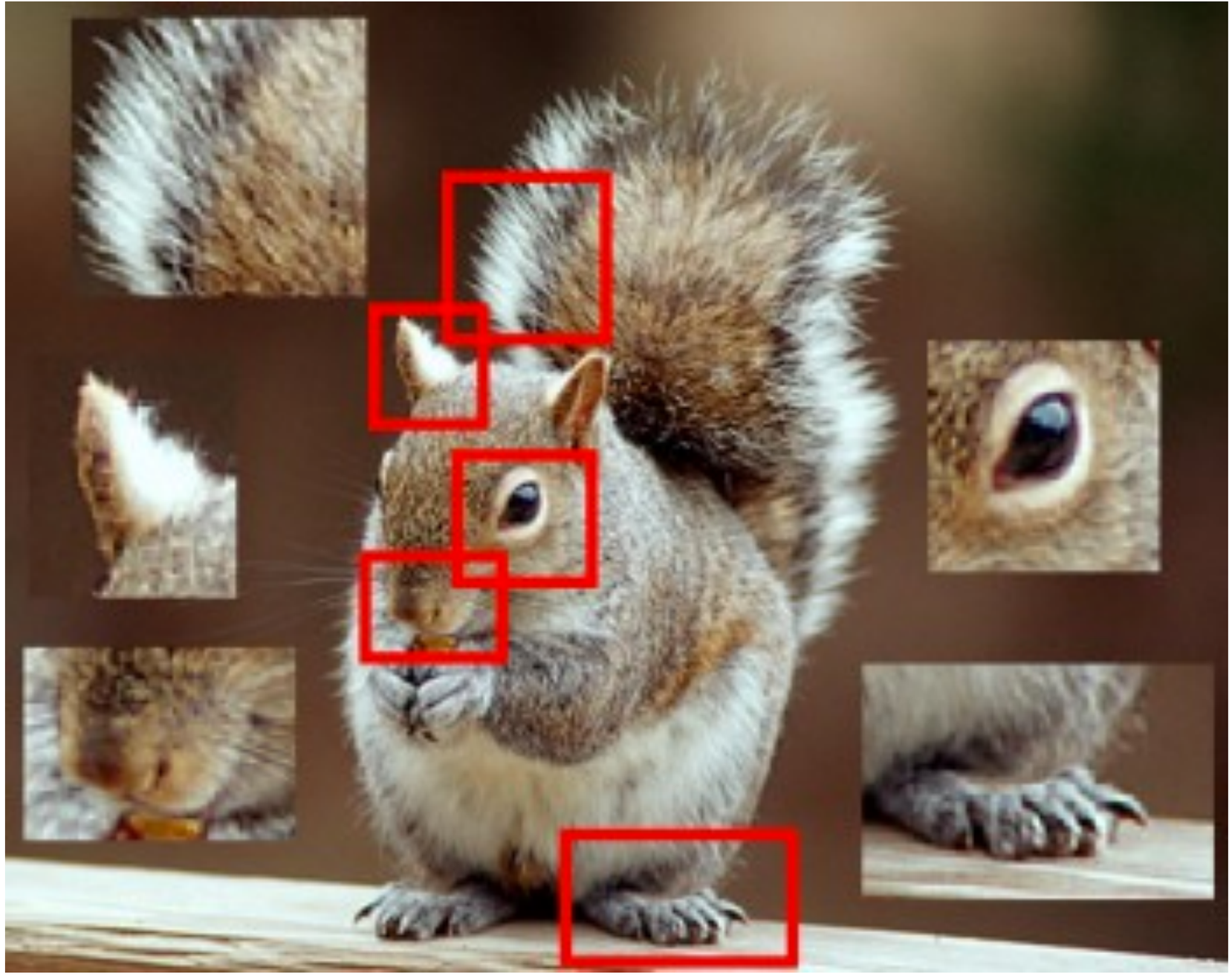
Conversely, unless ...111... occurs no unit is maximally activated.

Note the introduction of fixed-size ***receptive fields*** constrained the size of our hidden layer: the fields must span the input.

Receptive fields have natural biological counterparts

What are the pros and cons of the solution we have adopted here?

Gaussian receptive fields are common in visual processing (on-center/off-surround; off-center/on-surround). Other cells may react preferentially to bars of specific orientations.....



In a powerful deep learning network, we might have feature detectors for many kinds of features

Each set of units with a fixed weight vector is essentially analysing the whole image, one piece at a time, looking for the pattern it responds most sensitively to.

This is the "convolution" you may hear of in deep networks.