Computational cognitive modeling

Neural networks and deep learning

Feb. 20

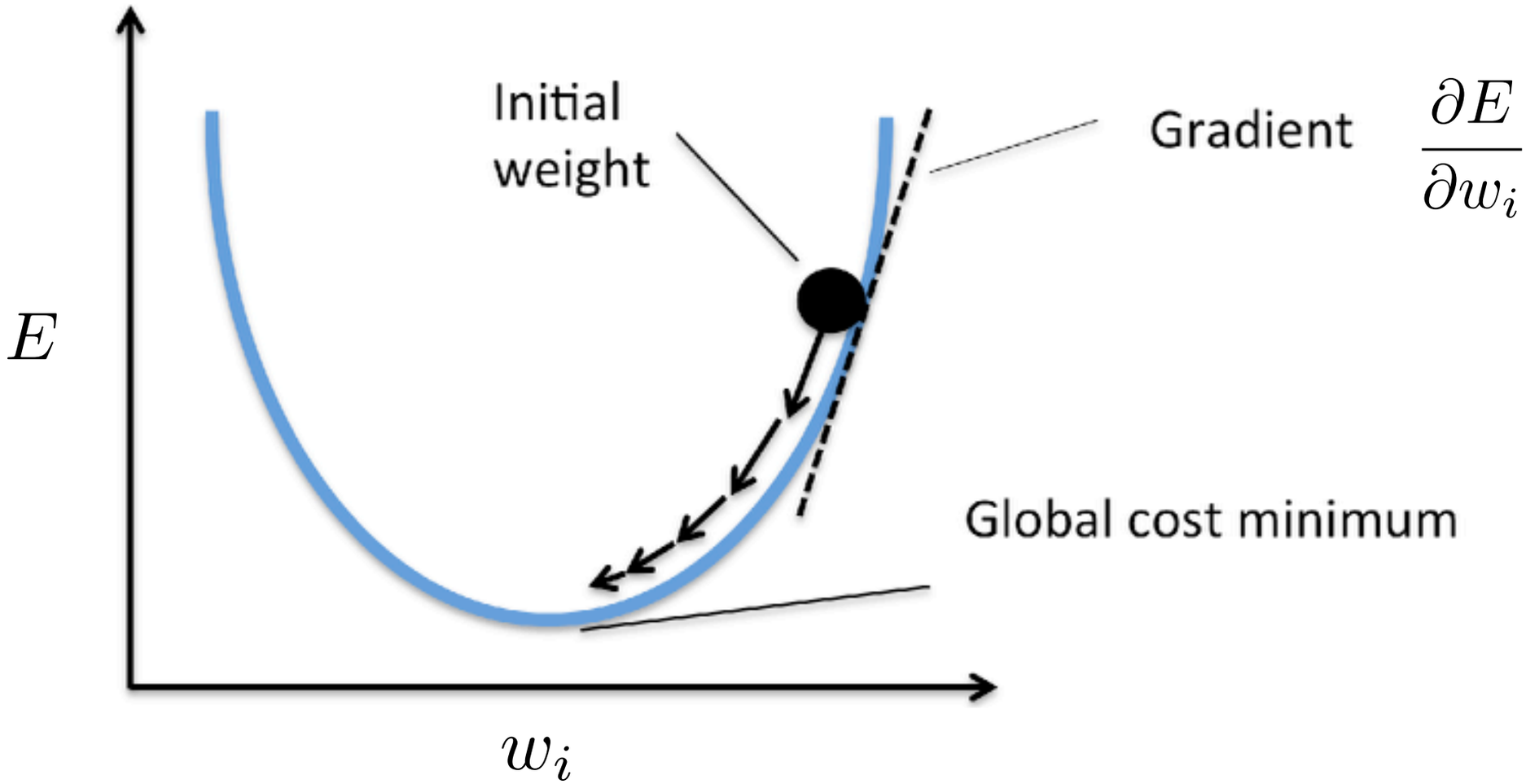
Fifth week course evaluation

Please check your email for the following:

Subject line: "[NYU Psychology] Spring 2018 Fifth-Week Course Evaluation for PSYCH-GA 3405 002: COMPUTATIONAL COGNITIVE MODELING"

Sender: Nicole Kozlowski

Review: stochastic gradient descent



Computing the gradient tells us which direction to go for steepest descent:

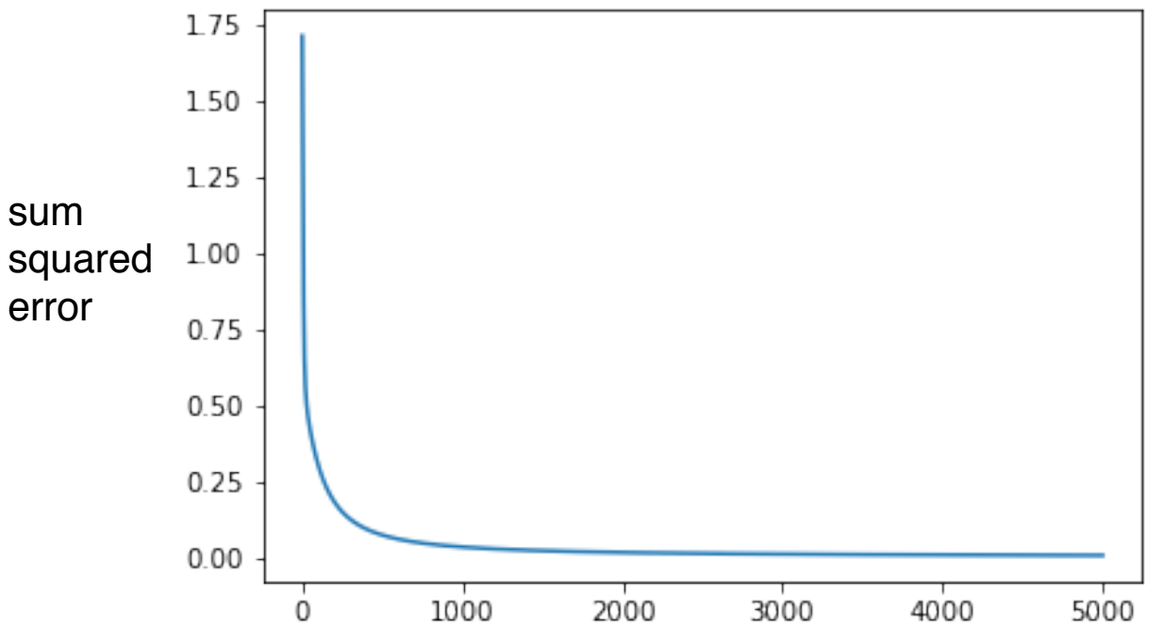
$$w_i \leftarrow w_i - \alpha \frac{\partial E}{\partial w_i} \qquad \alpha : \text{learning rate}$$

Credit https://rasbt.github.io/mlxtend/

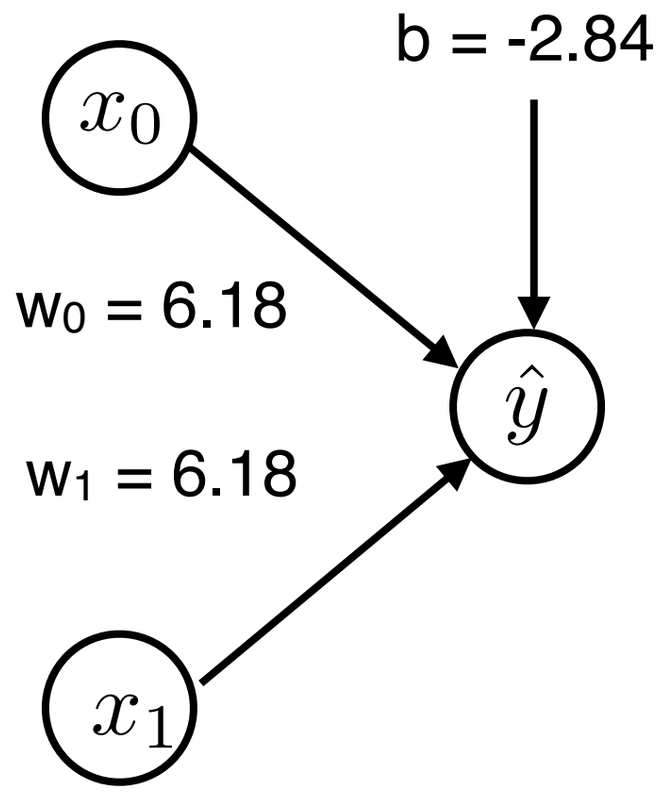
Review: Learning via stochastic gradient descent

We repeatedly cycle through each pattern in the training set, making updates after each pattern as in the previous slide.

Error over time



Final network

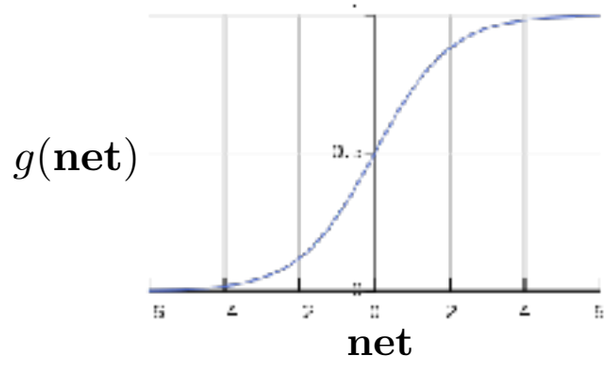


Learned function (logical OR)

x_0	x_1	$\hat{y}$	y
0	0	0.05	0
0	1	0.97	1
1	0	0.97	1
1	1	0.99	1

Pretty good fit!

$$g(\mathbf{net}) = \frac{1}{1 + e^{-\mathbf{net}}}$$



Review: Backpropagation algorithm for computing gradient

Updates for these weights the same as before:

$$\frac{\partial E}{\partial w_0} \quad \frac{\partial E}{\partial w_1} \quad \frac{\partial E}{\partial w_2}$$

What about the other weights?

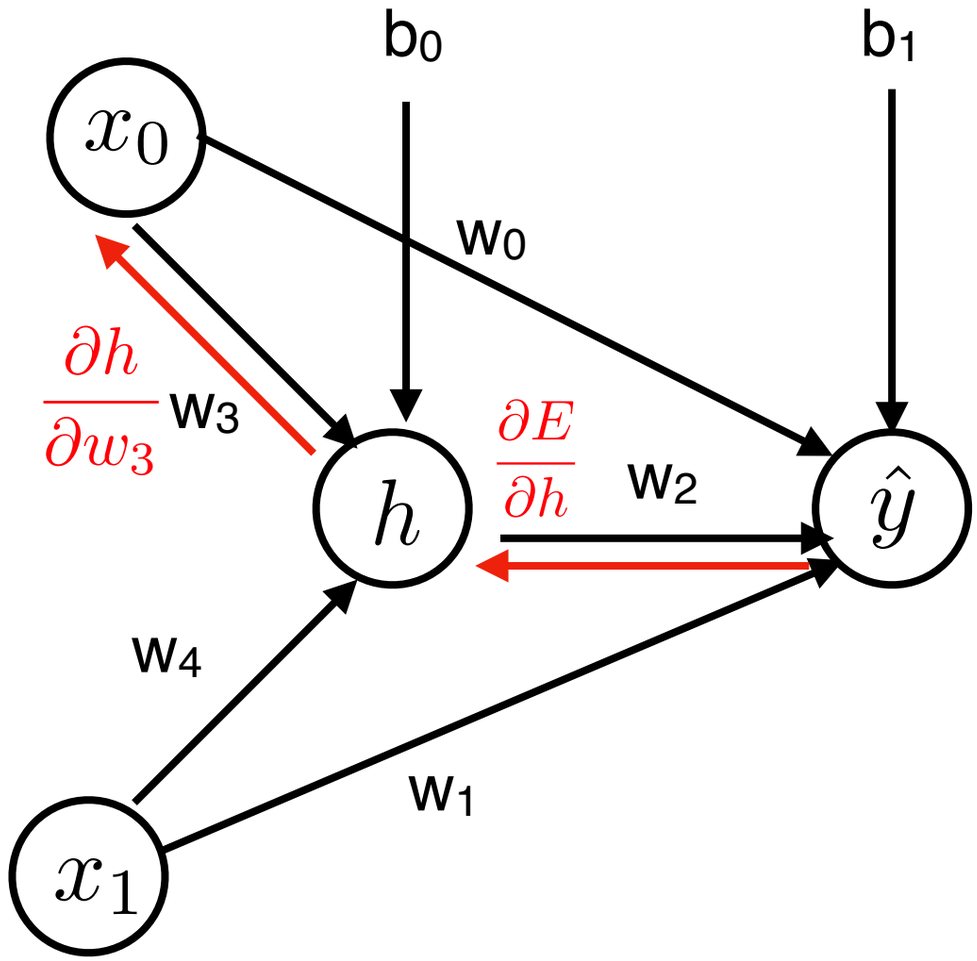
$$\frac{\partial E}{\partial w_3} \quad \frac{\partial E}{\partial w_4}$$

Multi-step strategy:

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial w_3}$$

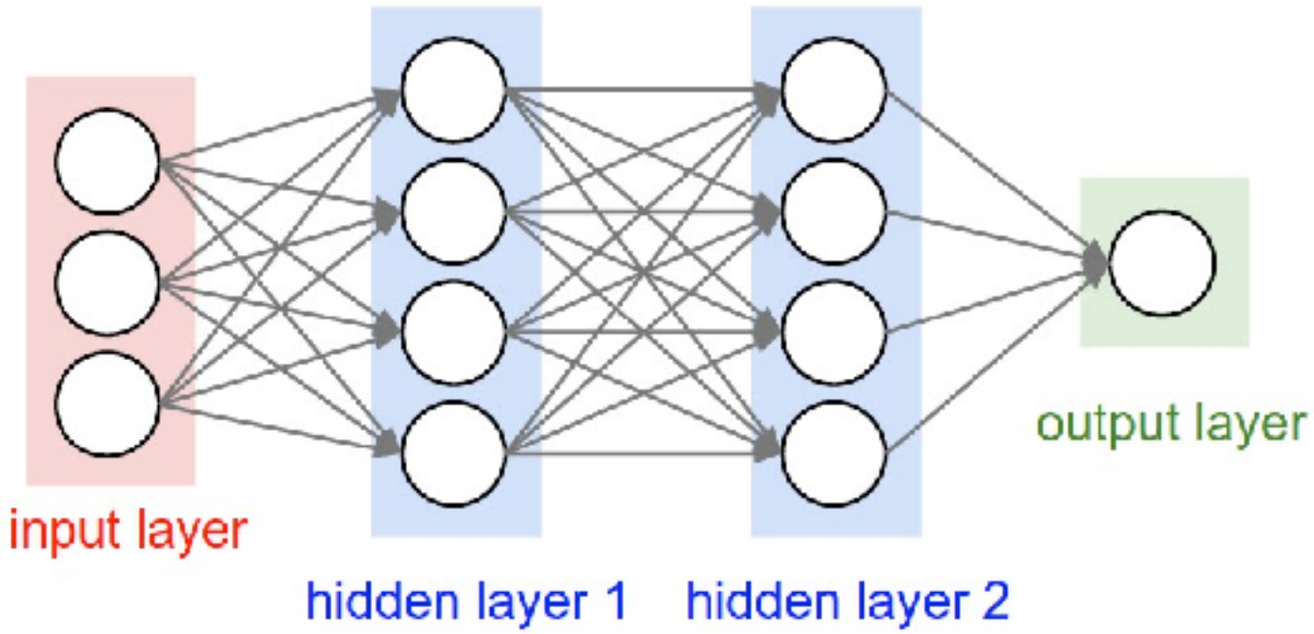
Step 1) Compute how error changes as a function of hidden unit activation

Step 2) Compute how hidden unit activation changes as a function of weight

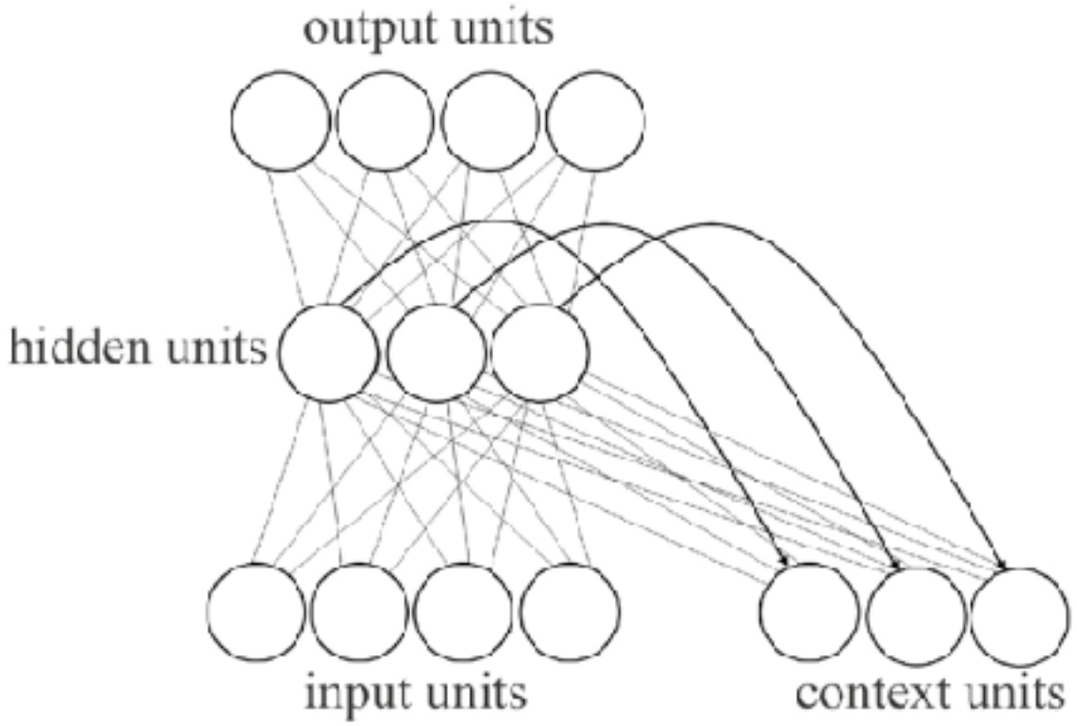


Backpropagation allows us to efficiently optimize a wide range of “deep neural network” models

Deep fully-connected neural network



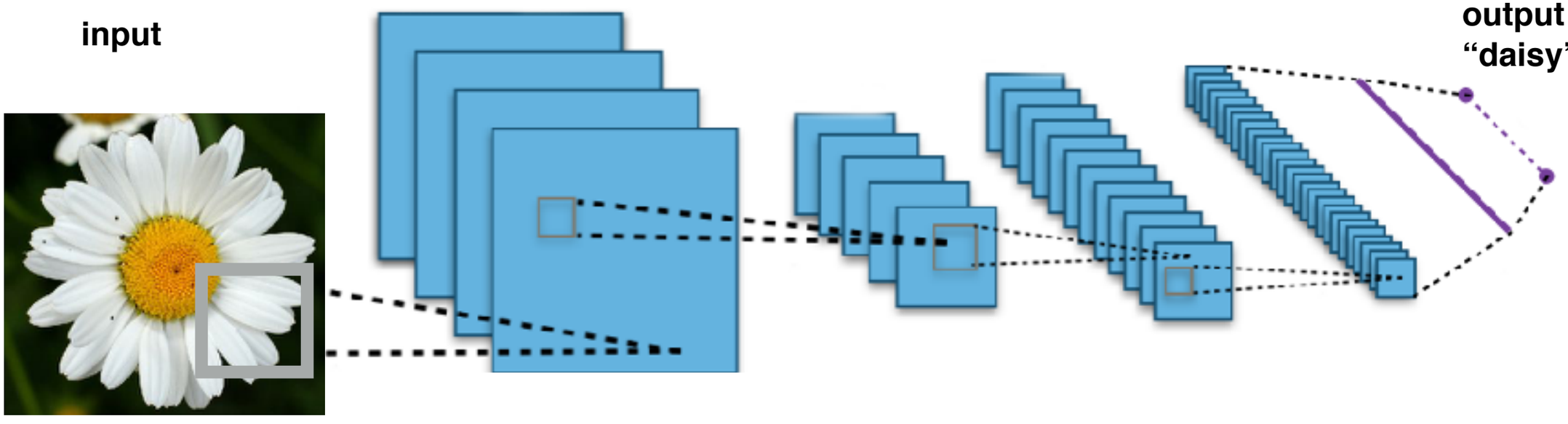
Recurrent neural network



Deep convolutional neural network

layers of feature maps

input

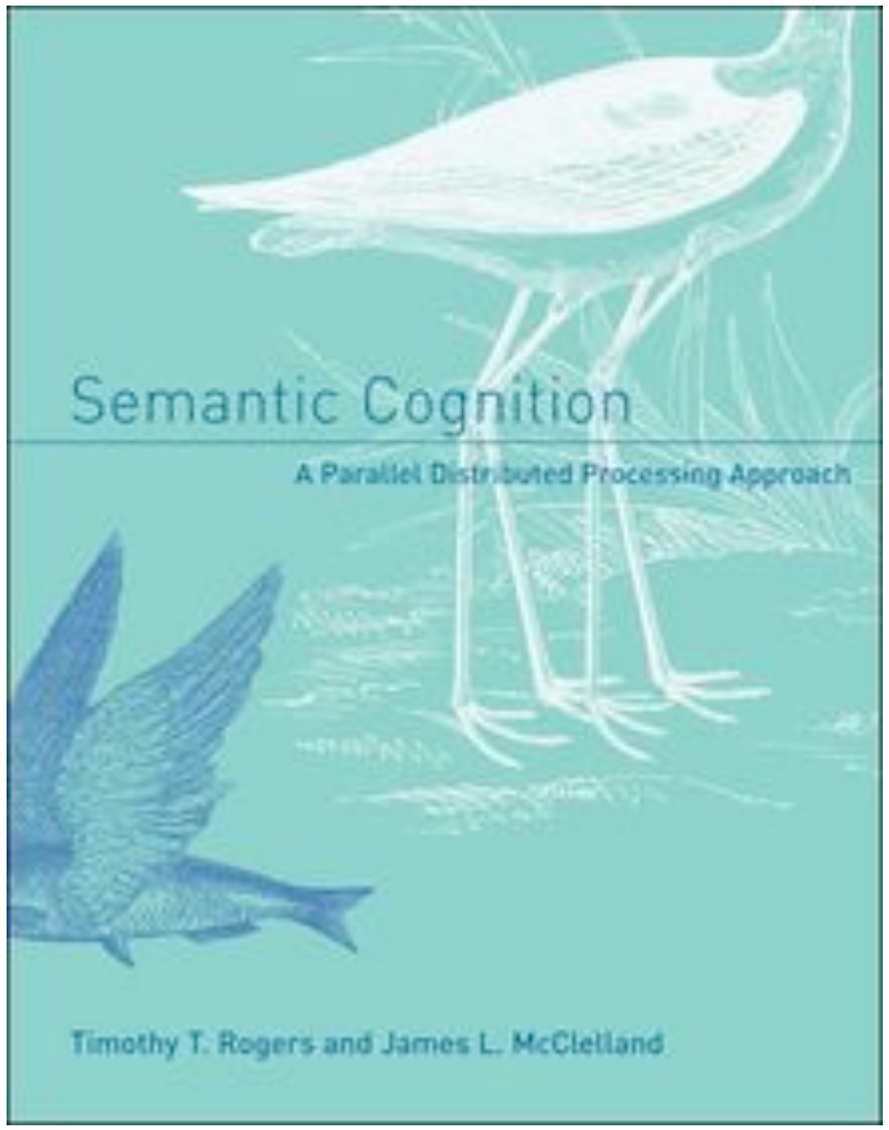


Modeling semantic cognition with a multi-layer neural net trained with backpropagation

THE PARALLEL DISTRIBUTED PROCESSING APPROACH TO SEMANTIC COGNITION

James L. McClelland and Timothy T. Rogers‡*

How do we know what properties something has, and which of its properties should be generalized to other objects? How is the knowledge underlying these abilities acquired, and how is it affected by brain disorders? Our approach to these issues is based on the idea that cognitive processes arise from the interactions of neurons through synaptic connections. The knowledge in such interactive and distributed processing systems is stored in the strengths of the connections and is acquired gradually through experience. Degradation of semantic knowledge occurs through degradation of the patterns of neural activity that probe the knowledge stored in the connections. Simulation models based on these ideas capture semantic cognitive processes and their development and disintegration, encompassing domain-specific patterns of generalization in young children, and the restructuring of conceptual knowledge as a function of experience.



How do we know that Socrates is mortal? Aristotle suggested that we reason from two propositions, in this case: Socrates is a man; and all men are mortal. This classical SYLLOGISM forms the basis of many theories of how we attribute properties to individuals. First we categorize them, then we consult properties known to apply to members of the category. Another answer — the one that we and a growing community of researchers would give — is that the knowledge that Socrates is mortal is latent in the connections among the neurons in the brain that process semantic information. In this article, we contrast this approach with other proposals, including a hierarchical propositional approach that grows out of the classical tradition. We show how it can address several findings on the acquisition of SEMANTIC KNOWLEDGE in development and its disintegration in dementia. It can also capture a set of phenomena that have motivated the idea that semantic cognition rests on innately specified, intuitive, domain-specific theories. Although challenges remain to be addressed, this approach provides an integrated account of a wide range of phenomena, and provides a promising basis for addressing the remaining issues.

SYLLOGISM
A formal structure for deduction in argument, consisting of a major and a minor premise from which a conclusion logically follows.

**Center for the Neural Basis of Cognition and Department of Psychology, Carnegie Mellon University, 4400 Fifth Avenue, Pittsburgh, Pennsylvania 15213-2683, USA.*
‡Medical Research Council Cognition and Brain Sciences Unit, 15 Chaucer Road, Cambridge CB2 2EF, UK.
e-mails: jlm@cnbc.cmu.edu; tim.rogers@mrc-cbu.cam.ac.uk
doi:10.1038/nrn1076

The hierarchical propositional approach

In the early days of computer simulation models, researchers assumed that human semantic cognition was based on the use of categories and propositions. Quillian[1] proposed that if the concepts were organized into a hierarchy progressing from specific to general categories, then propositions true of all members of a superordinate category could be stored only once, at the level of the superordinate category. For example, propositions true of all living things could be stored at the top of the tree (FIG. 1). Other propositions, true of all animals but not of plants, could be stored at the next level down, and so on, with specific facts about an individual concept stored directly with it. To determine whether a proposition were true of a particular concept, one could access the concept, and see whether the proposition was stored there. If not, one could search at successively higher levels until the property was found, or until the top of the hierarchy was reached.

Quillian's proposal was appealing in part for its economy of storage: propositions true of many items could often be specified just once. The proposal also allowed for immediate generalization of what is known

Key questions in semantic cognition

- **Semantic cognition**: our intuitive understanding of objects and their properties

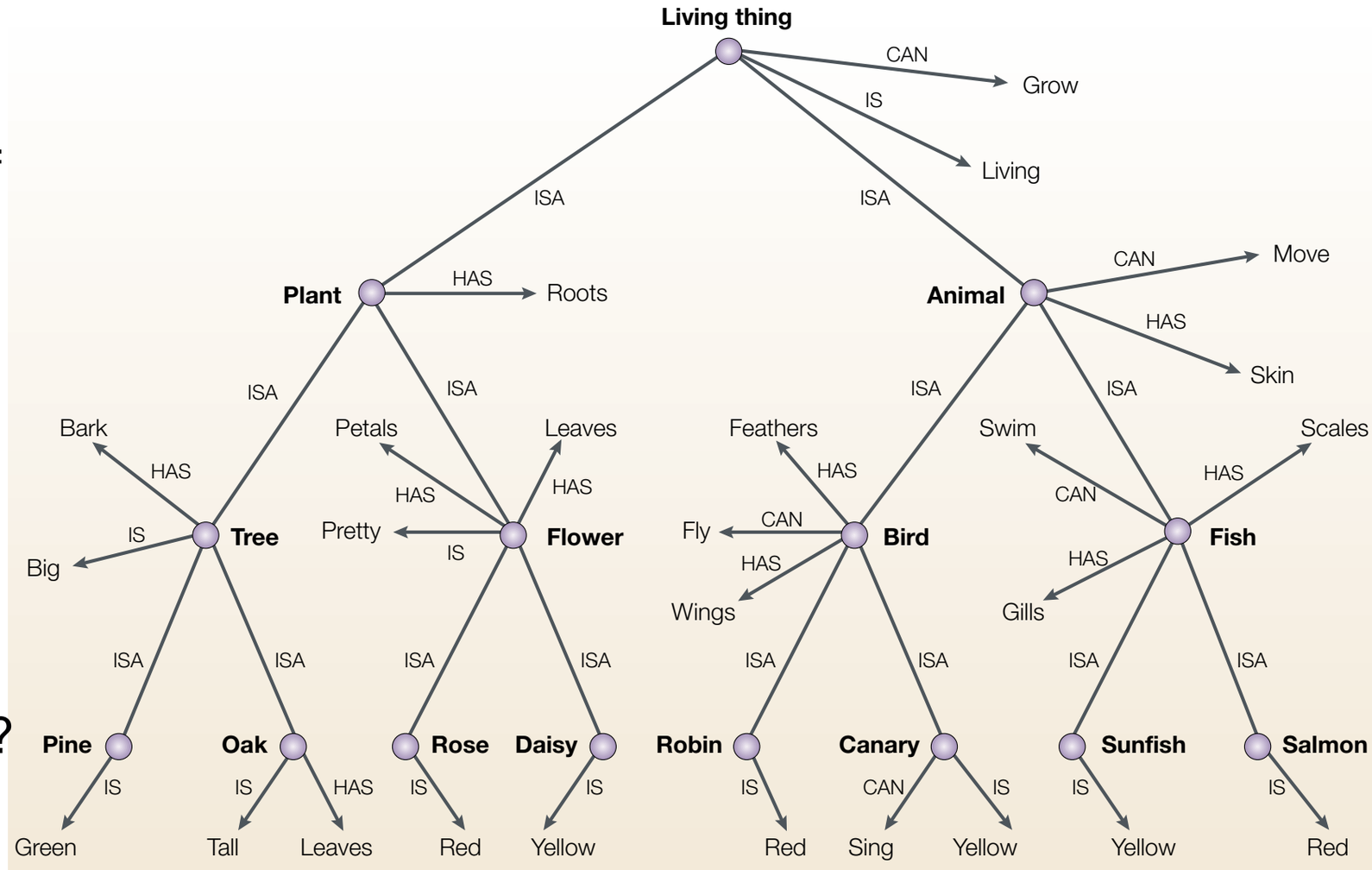
- How do we know what properties something has, and which of its properties should be generalized to other objects?

- How is the knowledge underlying these abilities acquired, and how is it affected by brain disorders?

- Can we understand semantic cognition as gradual optimization in a multi-layer neural network?

Quillian's hierarchical propositional model

- Quillian proposed that concepts are represented in a hierarchy organized from specific to general.
- Propositions true of all members of specific categories are stored only once, at the highest-level
- **Strengths of the model**
  - economy of storage
  - powerful inferences when adding a new concept
- **Problems for the model**
  - How do you handle exceptions? (e.g., a penguin that can't fly)
  - How do you decide which level to store a property?



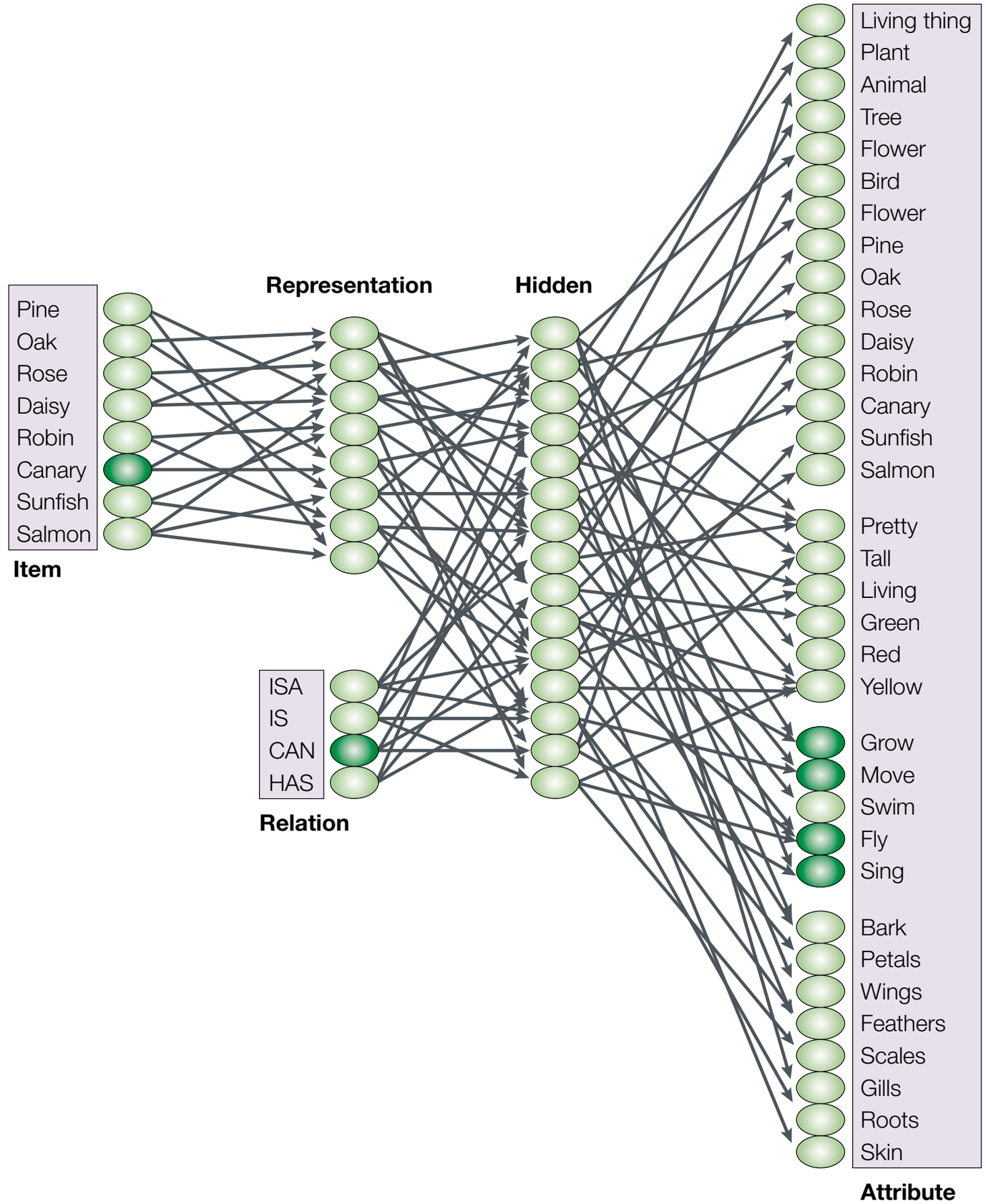
(Quillian, 1968)

A neural network model of semantic cognition

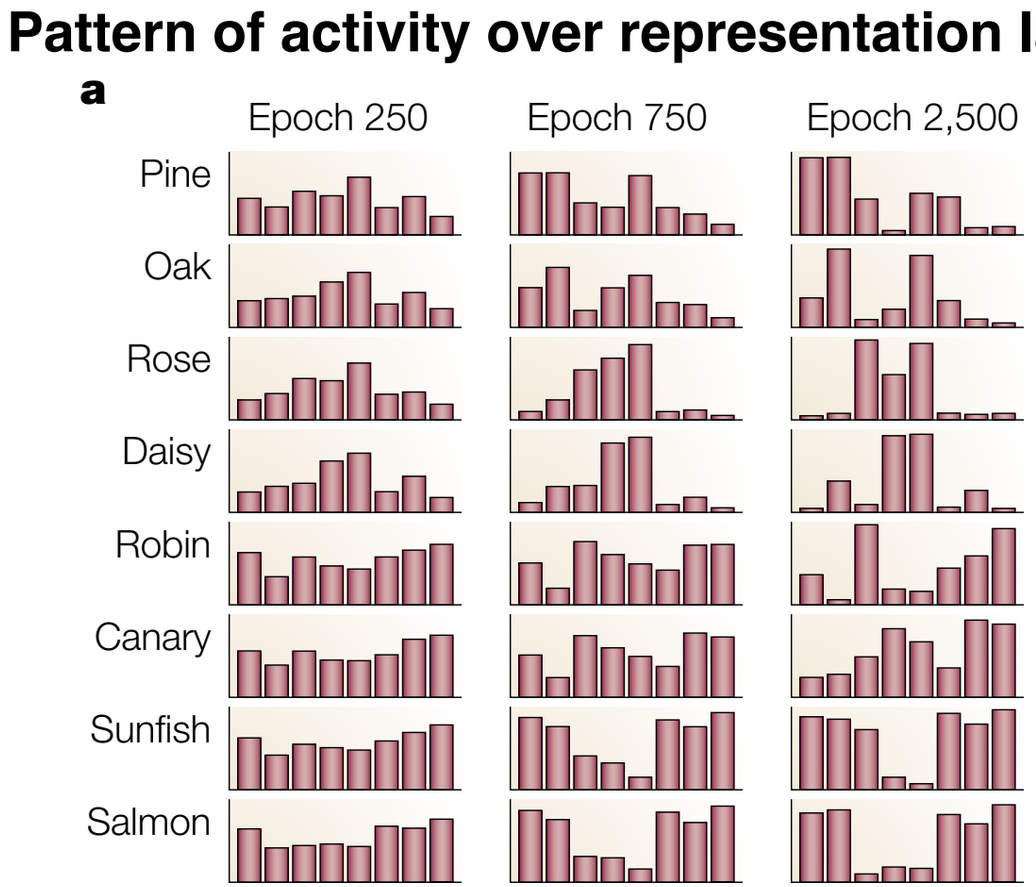
- Network is trained to answer queries involving an **item** (e.g., “Canary”) and a **relation** (e.g., “CAN”), outputting all **attributes** that are true of the item/relation pair (e.g., “grow, move, fly, sing”)

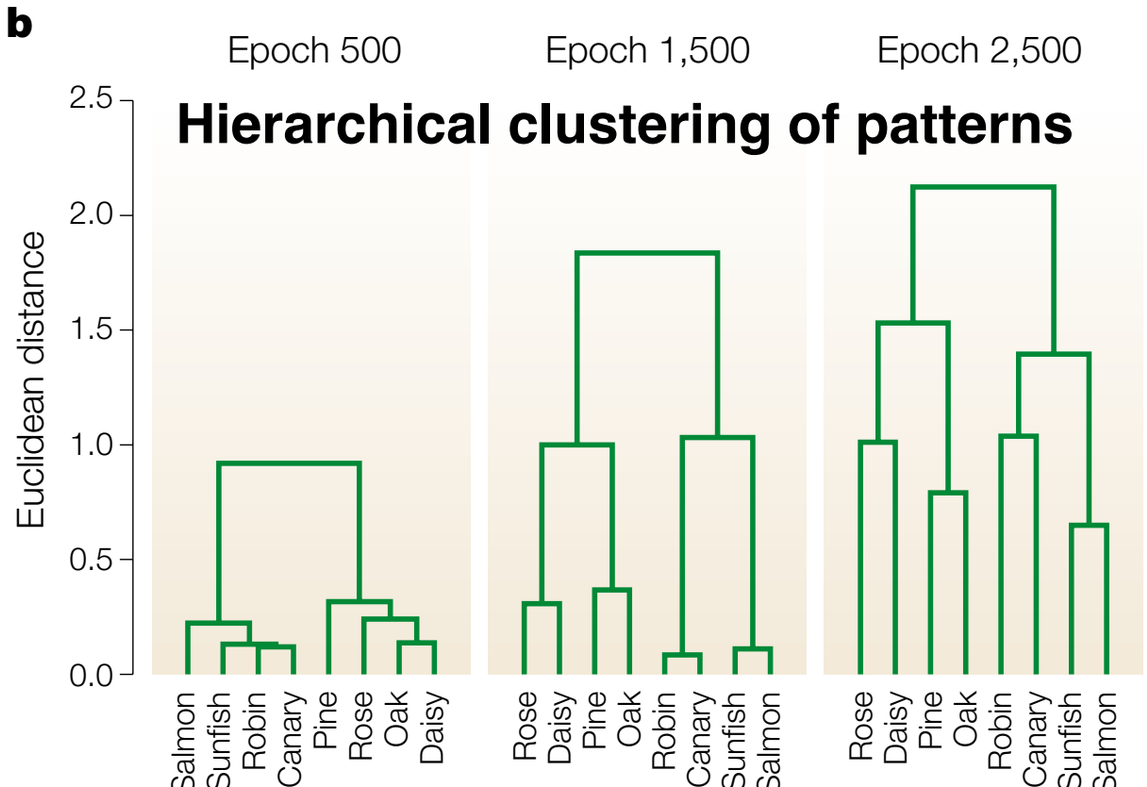
- Unlike Quillian’s model, knowledge is not stored explicitly in a hierarchy. It is stored implicitly in the web of connection weights.

- Starting with random weights, the network is trained on all facts stored in the Quillian hierarchy on the previous slide.



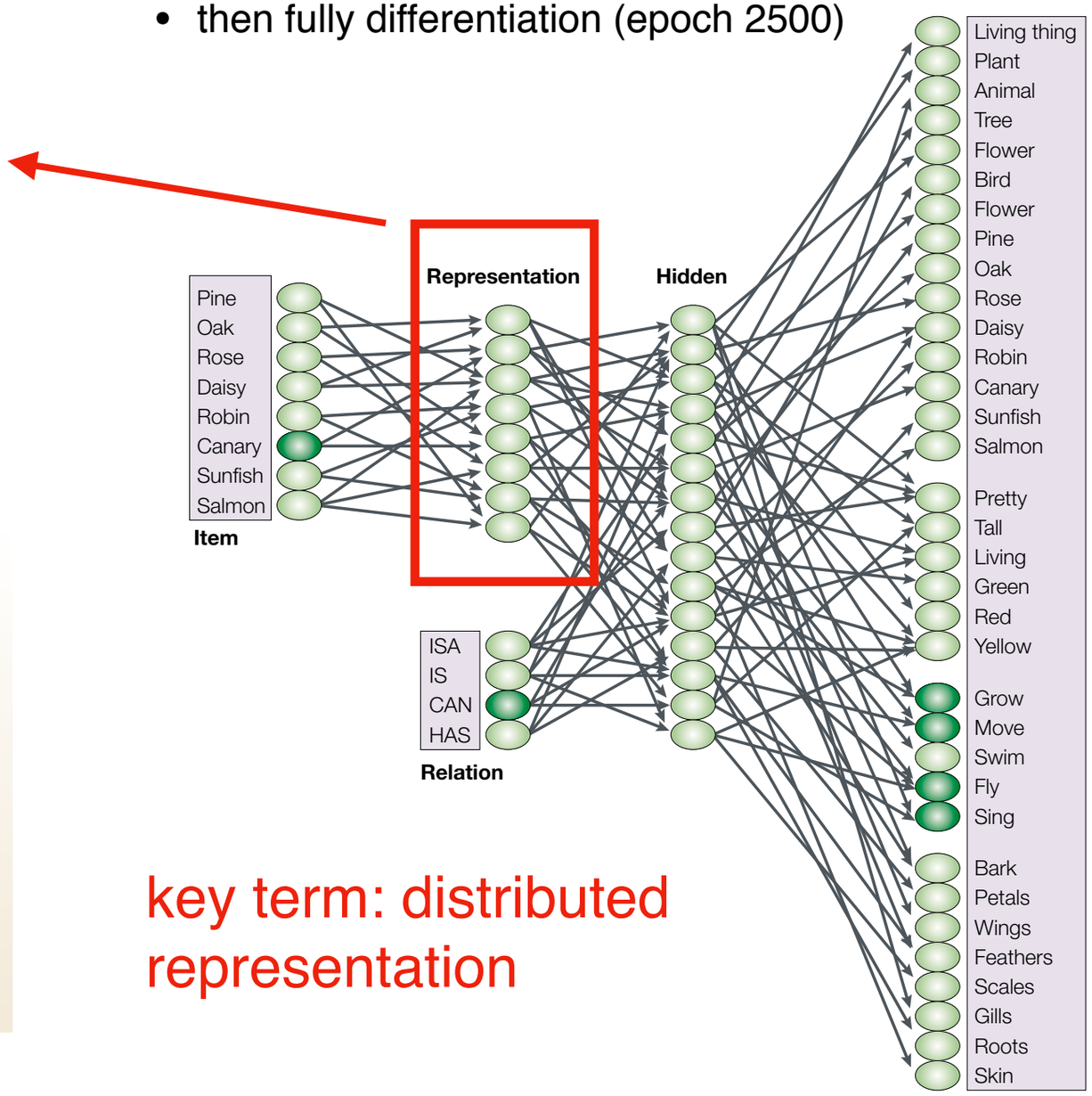
Modeling cognitive development of semantic representation through training with backpropagation





During training, model goes through stages that resemble broad-to-specific differentiation in children's cognitive development (e.g., Mandler vs. Bauer, 1988)

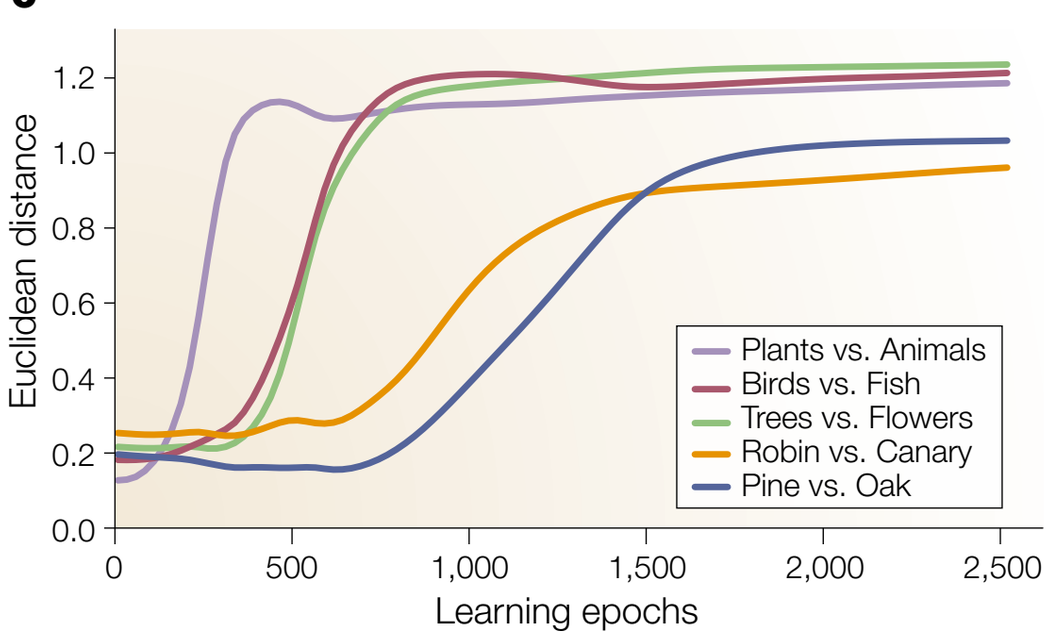
- first differentiates plants vs. animals (epoch 250)
- then birds vs. fish and trees vs. flowers (epoch 750)
- then fully differentiation (epoch 2500)



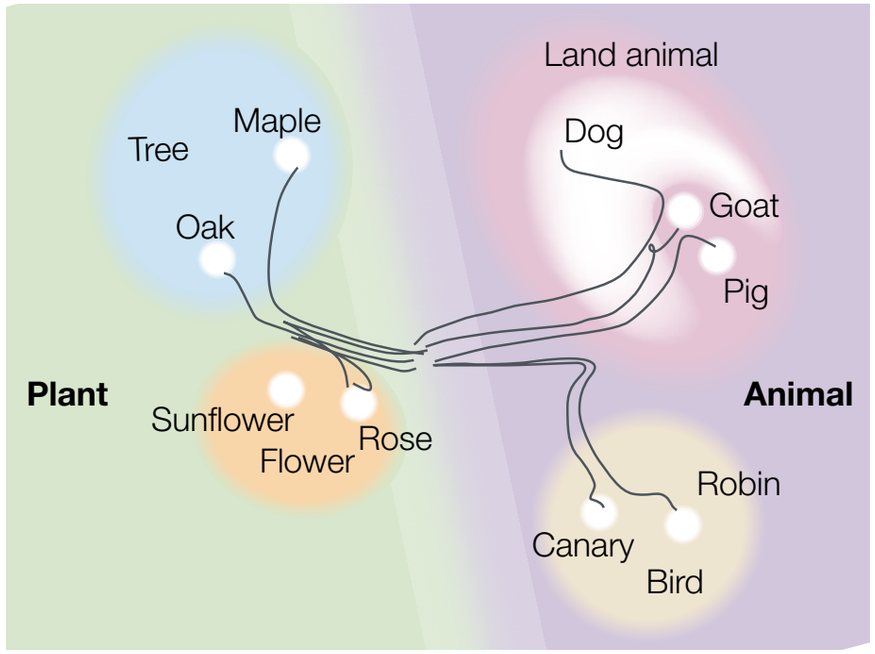
key term: distributed representation

Modeling cognitive development of semantic representation through training with backpropagation

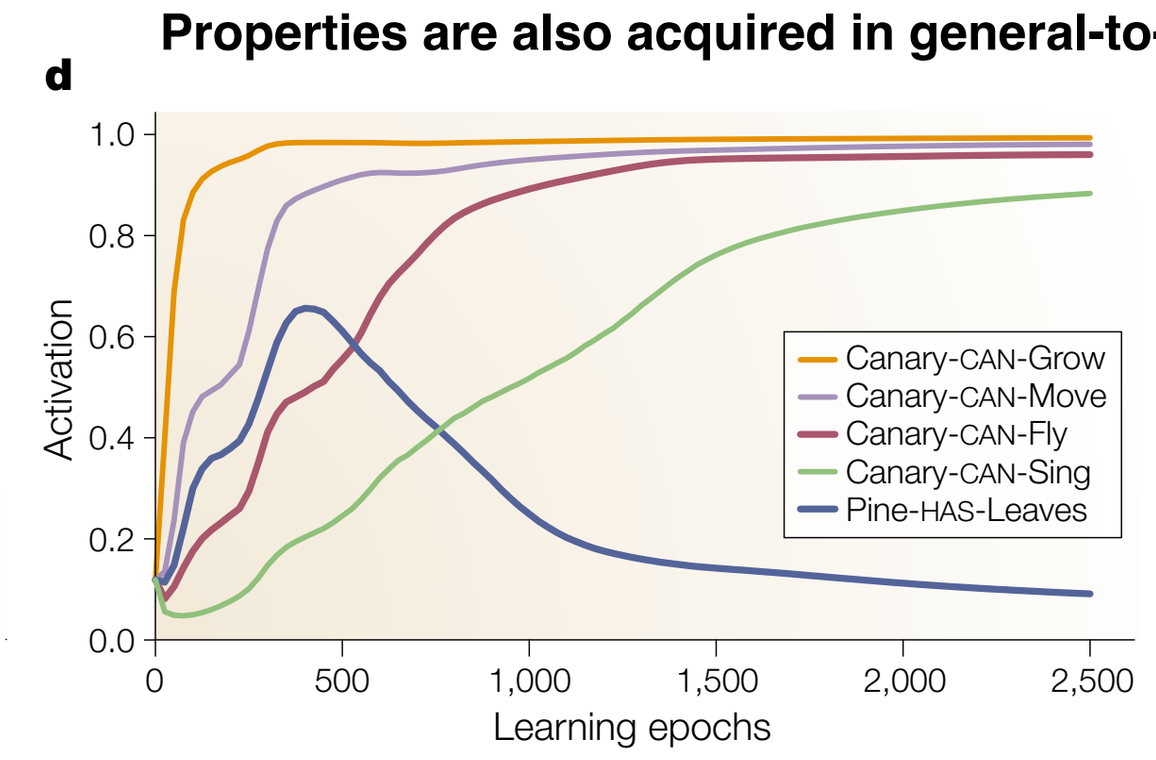
Broad-to-specific stages of conceptual development



PCA embedding over time of representation layer



Properties are also acquired in general-to-specific manner



Modeling semantic dementia by adding noise to the neural network

- For human patients with semantic dementia (a form of progressive brain damage), specific categories and properties are lost first, while more general information is preserved.

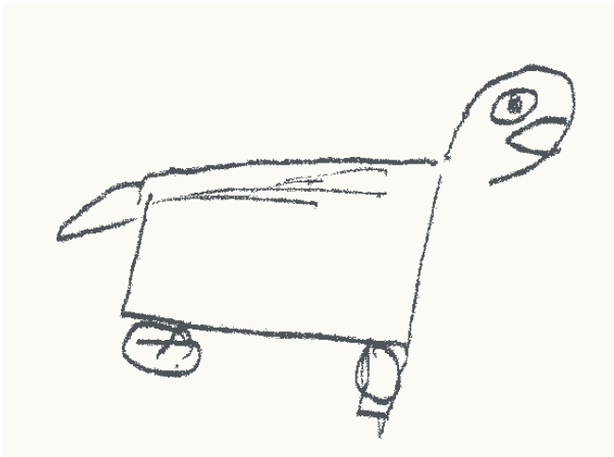
Patient naming pictures of birds

Picture naming responses for JL

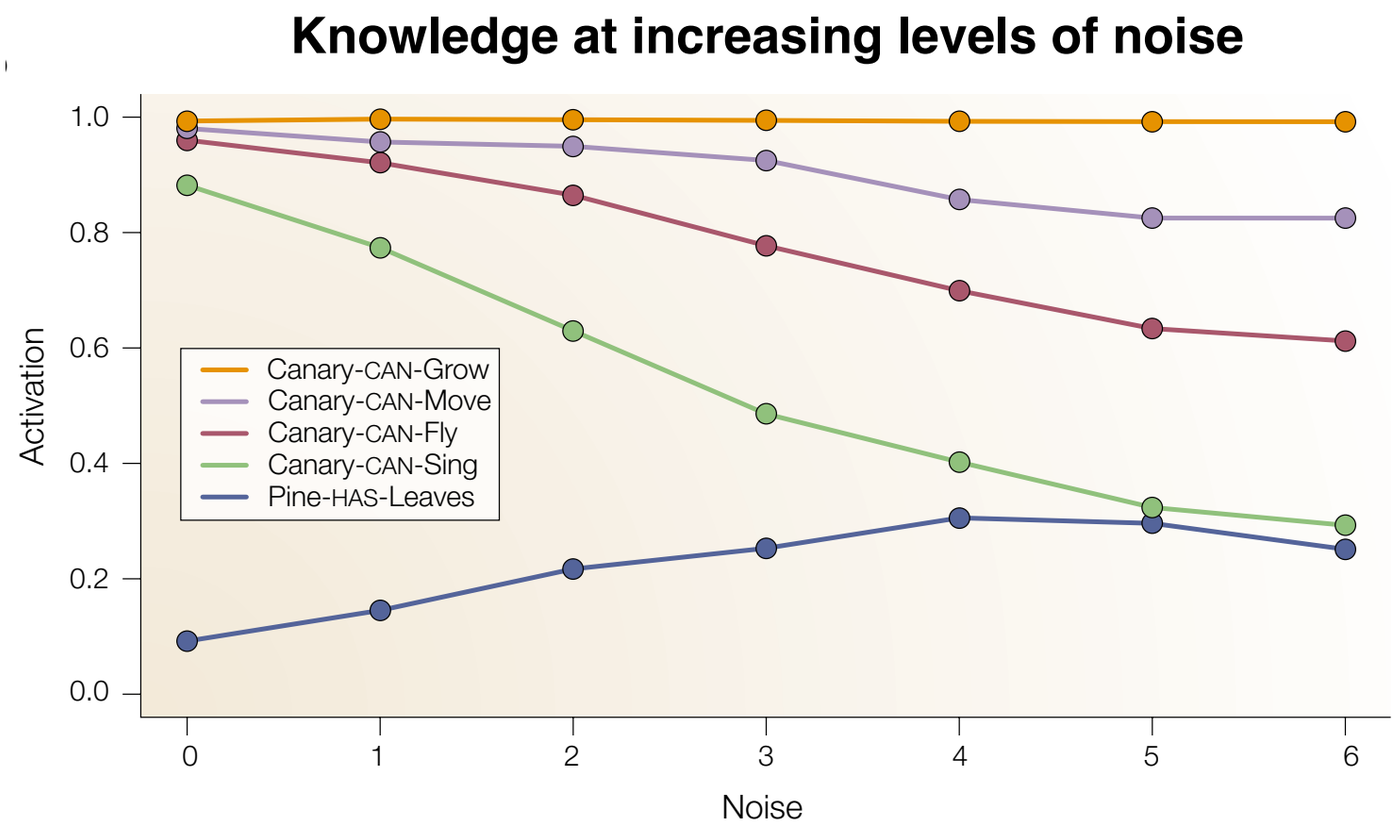
Item	Sept. 91	March 92	March 93
Bird	+	+	Animal
Chicken	+	+	Animal
Duck	+	Bird	Dog
Swan	+	Bird	Animal
Eagle	Duck	Bird	Horse
Ostrich	Swan	Bird	Animal
Peacock	Duck	Bird	Vehicle
Penguin	Duck	Bird	Part of animal
Rooster	Chicken	Chicken	Dog

Drawing a camel shows loss of specifics

c IF's delayed copy of a camel



- If random noise is added to the representation layer of the neural network, a similar pattern of losing the specifics but keeping the general information is observed.



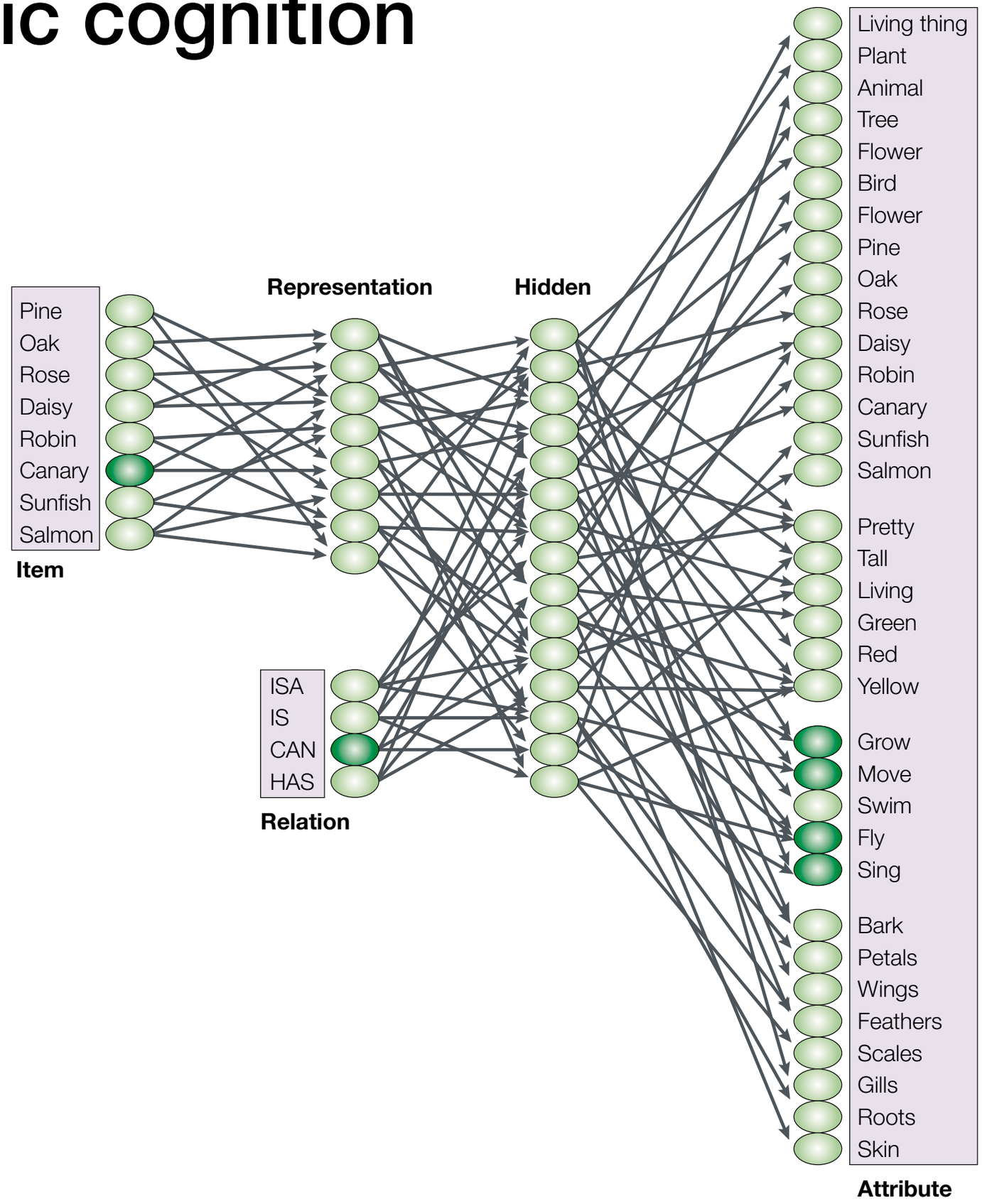
Summary: A neural network model of semantic cognition

- Network is trained to answer queries involving an **item** (e.g., “Canary”) and a **relation** (e.g., “CAN”), outputting all **attributes** that are true of the item/ relation pair (e.g., “grow, move, fly, sing”)

- Trained with stochastic gradient descent, as we learned about in class

- The model helps us to understand the broad-to-specific pattern of differentiation in children’s cognitive development

- It also helps us to understand the specific-to-general deterioration in semantic dementia



Key principles of neural network models of cognition

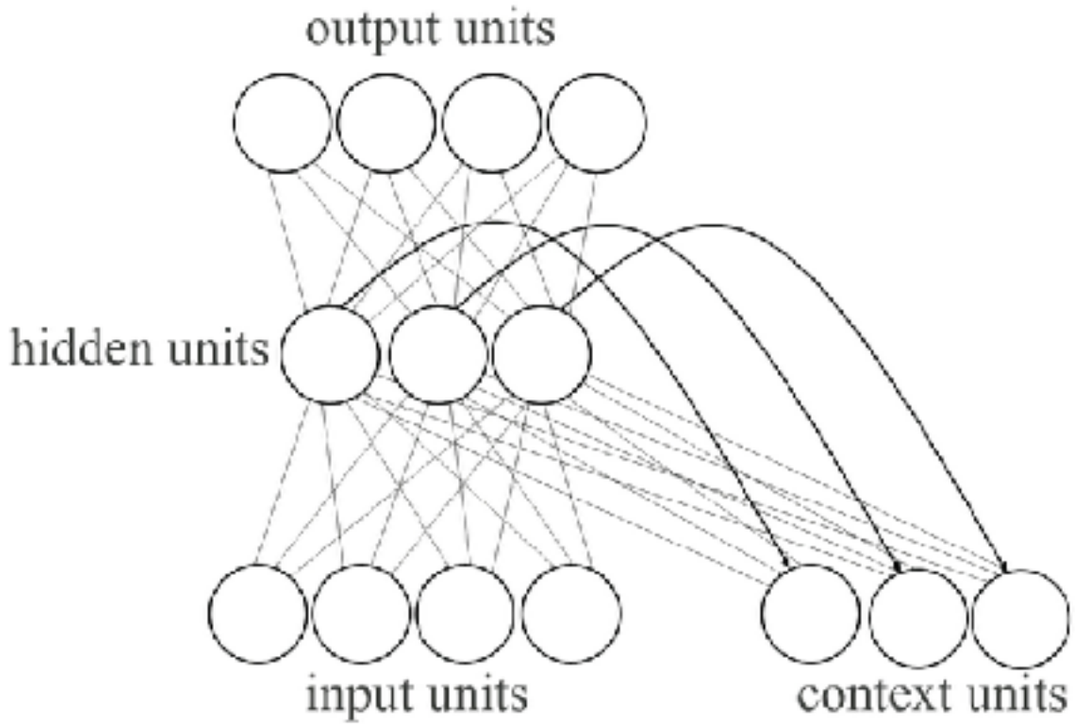
- *Neurally-inspired computation.* Taking inspiration from the low-level (how neurons compute) is key to understanding the high-level (intelligence and cognition)

- *Intelligence is an emergent phenomenon.* Complex behavior can emerge from a very large number of simple, interactive computations.

- *Simulation is central.* It's hard to predict how complexity will emerge. Computational modeling and simulation are essential for understanding intelligence.

Today's lecture: Two very important types of neural network models

Recurrent neural network

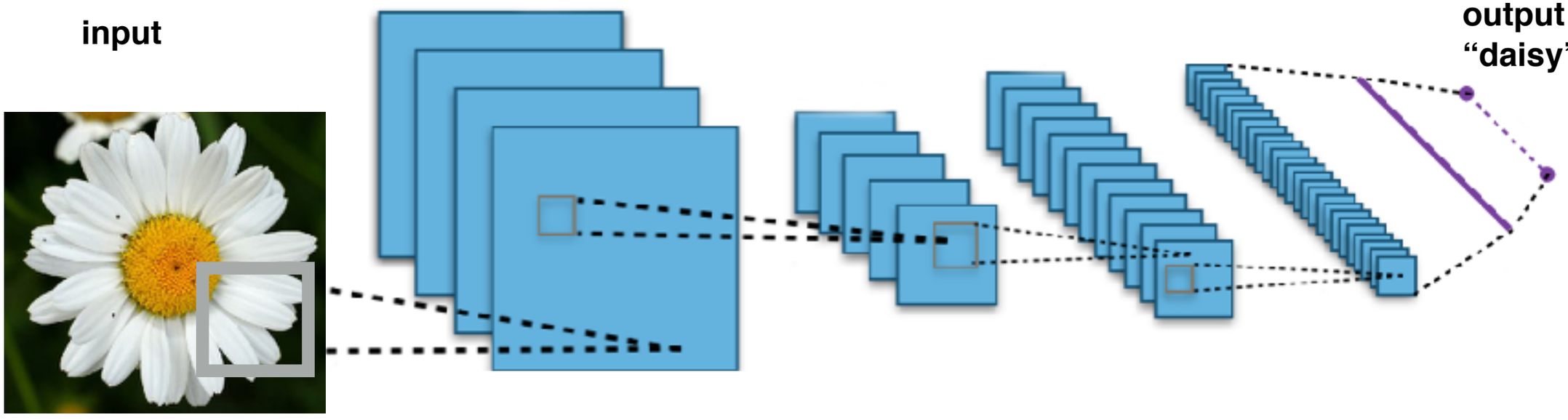


Deep convolutional neural network

layers of feature maps

input

output: "daisy"

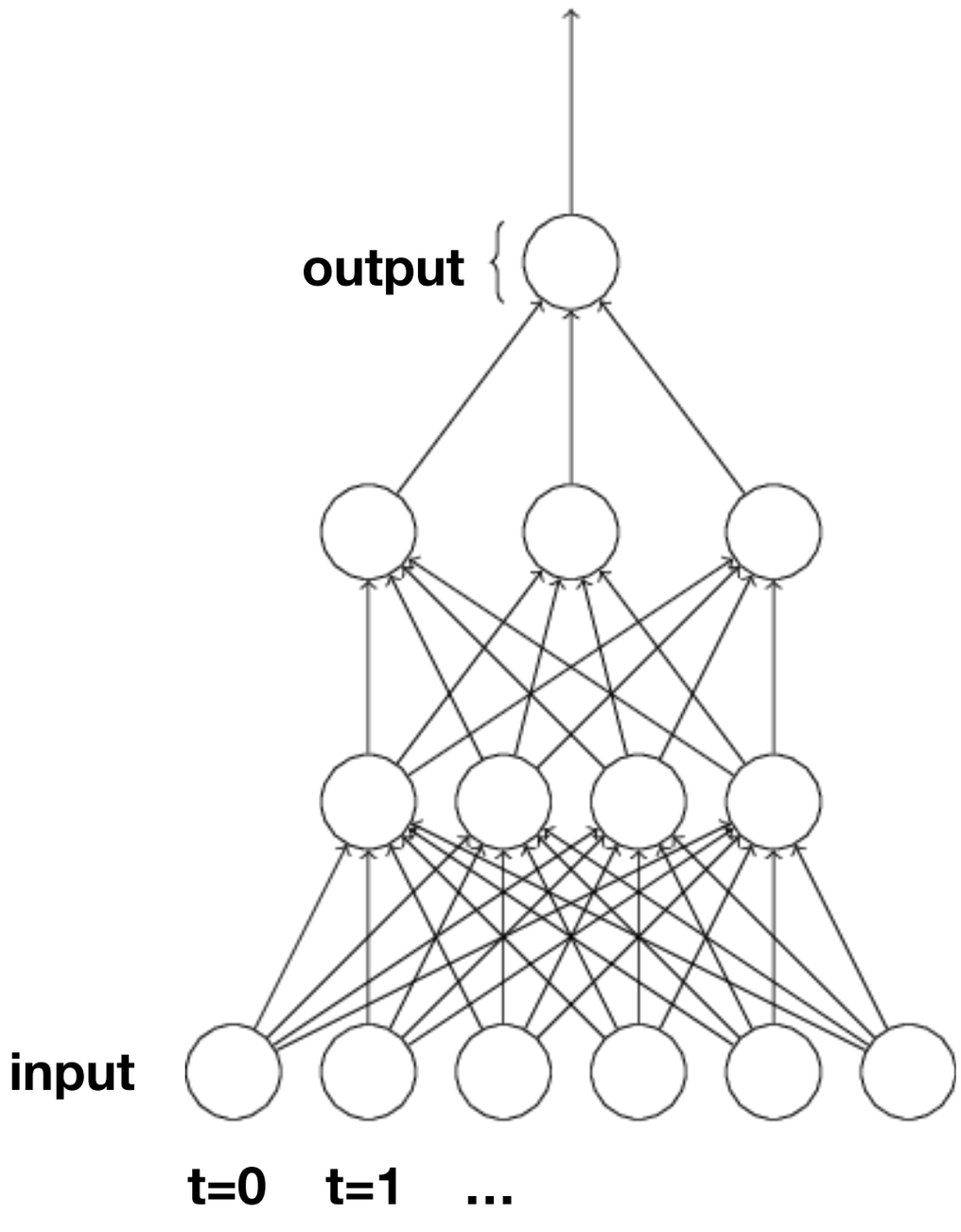


Data often has important temporal structure

- Language/text
- Speech
- Video
- Financial
- Weather
- All of human behavior unfolds in time

How do we represent time in a neural network?

Naive approach: represent time spatially in a standard network



Problems with this approach:

- The network has a fixed buffer. How large do you make the buffer?
- Two identical patterns translated in time, such as **[0 1 1 0 0 0], [0 0 0 1 1 0]** have no natural overlap in the (untrained) architecture

COGNITIVE SCIENCE **14,** 179–211 (1990)

Finding Structure in Time

JEFFREY L. ELMAN
University of California, San Diego

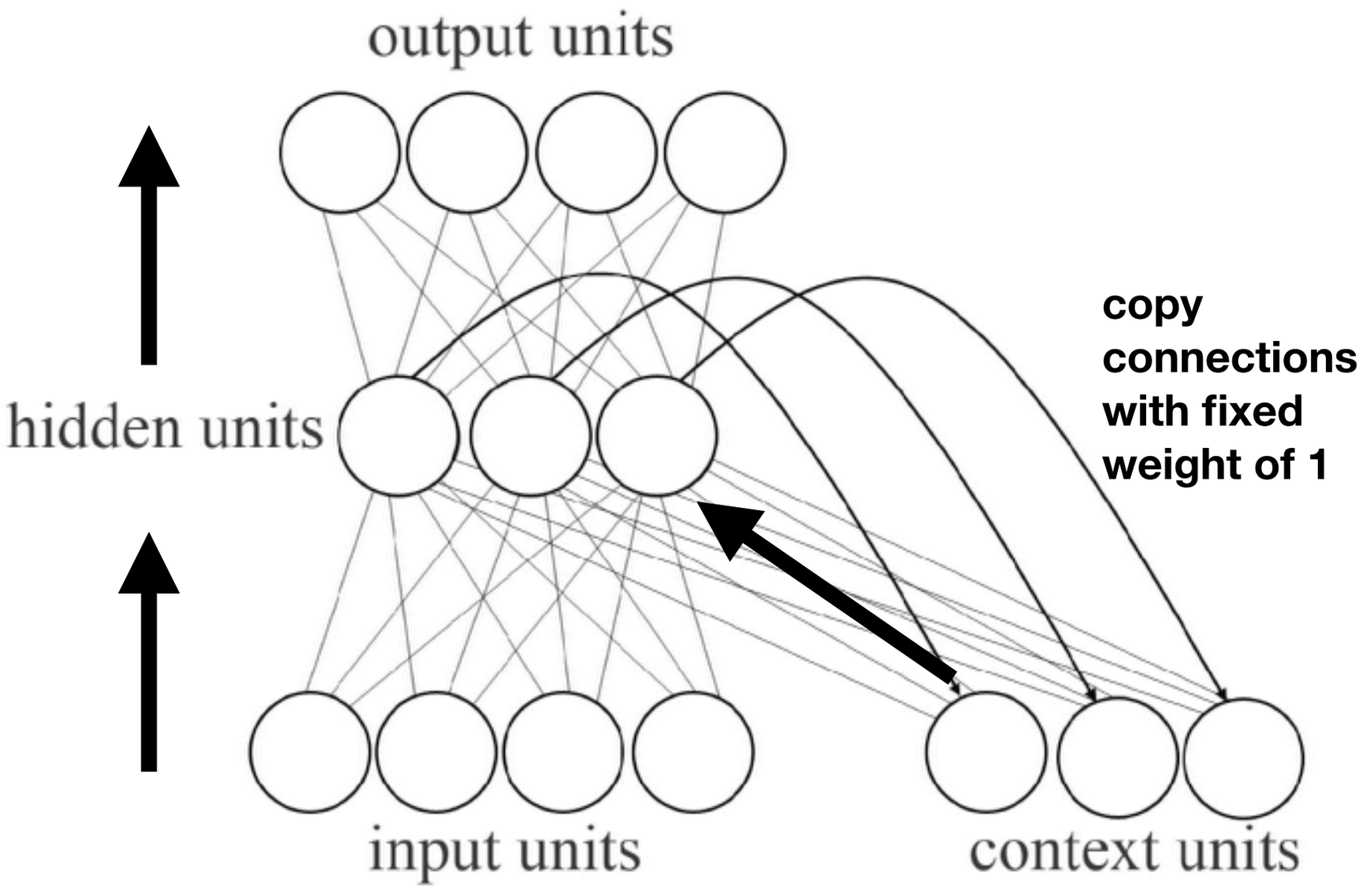
Time underlies many interesting human behaviors. Thus, the question of how to represent time in connectionist models is very important. One approach is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation). The current report develops a proposal along these lines first described by Jordan (1986) which involves the use of recurrent links in order to provide networks with a dynamic memory. In this approach, hidden unit patterns are fed back to themselves; the internal representations which develop thus reflect task demands in the context of prior internal states. A set of simulations is reported which range from relatively simple problems (temporal version of XOR) to discovering syntactic/semantic features for words. The networks are able to learn interesting internal representations which incorporate task demands with memory demands; indeed, in this approach the notion of memory is inextricably bound up with task processing. These representations reveal a rich structure, which allows them to be highly context-dependent, while also expressing generalizations across classes of items. These representations suggest a method for representing lexical categories and the type/token distinction.

INTRODUCTION

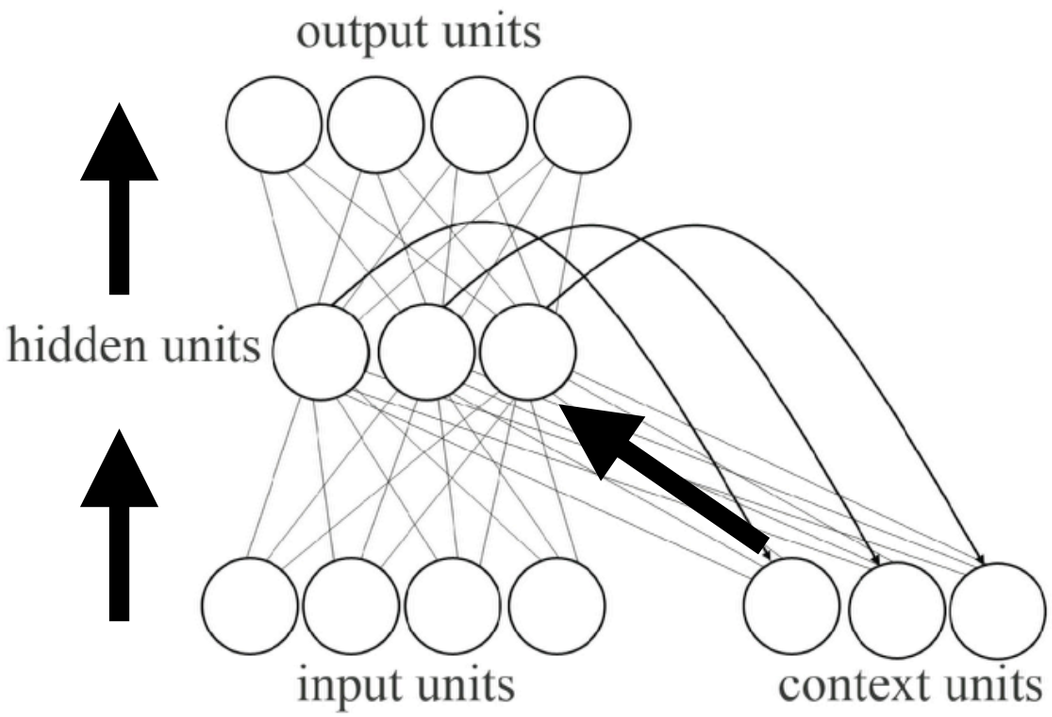
Time is clearly important in cognition. It is inextricably bound up with many behaviors (such as language) which express themselves as temporal sequences. Indeed, it is difficult to know how one might deal with such basic problems as goal-directed behavior, planning, or causation without some way of representing time.

The question of how to represent time might seem to arise as a special problem unique to parallel-processing models, if only because the parallel nature of computation appears to be at odds with the serial nature of tem-

Simple recurrent network (SRN; or Elman network)



A simple example task



input data:

diibaguuubadiidiiguuuguuudiidiiba dii…. (random mix of “ba”, “dii”, and “guu”)

task:

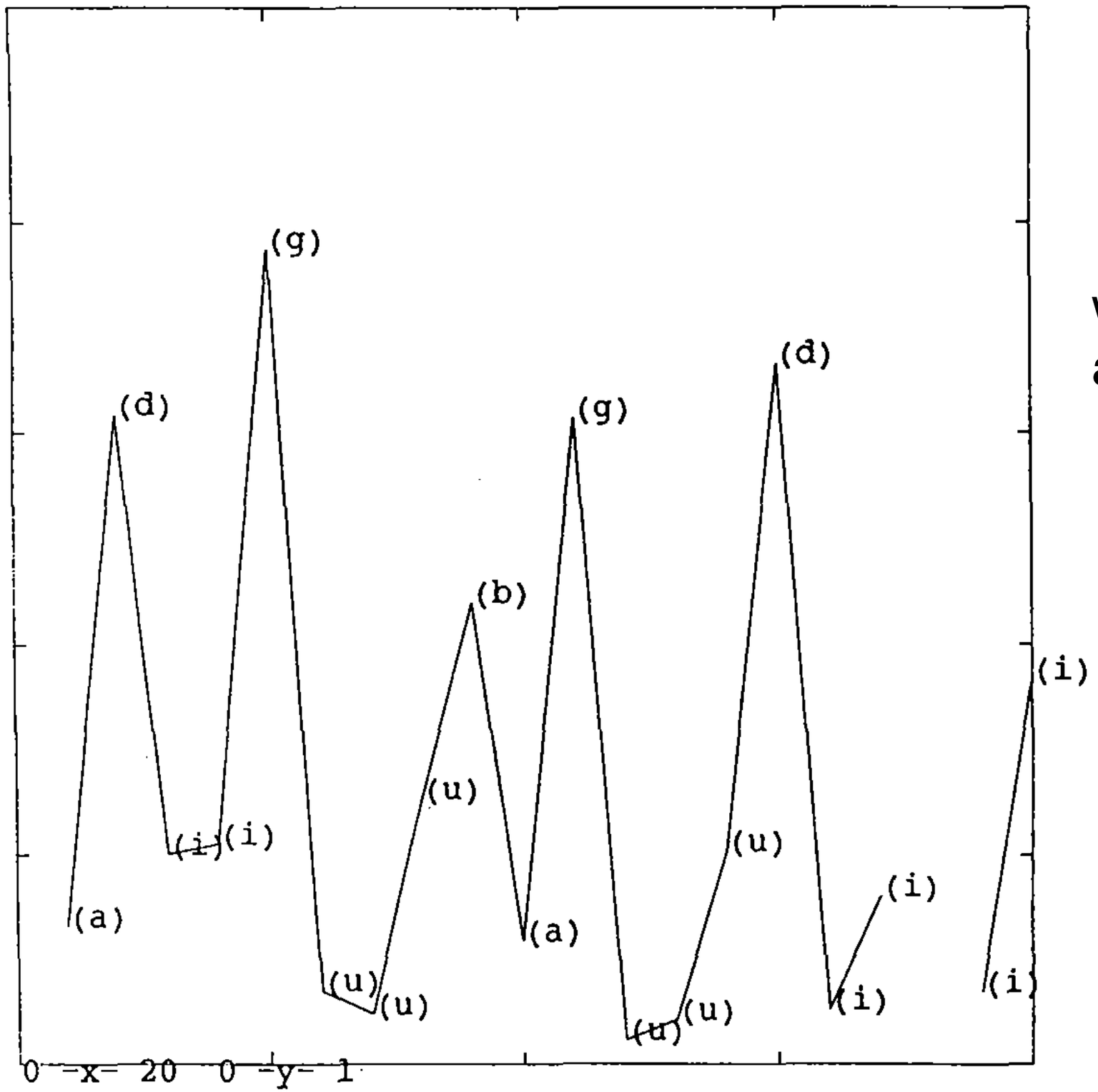
predict the next letter

representation of input

	Consonant	Vowel	Interrupted	High	Back	Voiced
b	1	0	1	0	0	1
d	1	0	1	1	0	1
g	1	0	1	0	1	1
a	0	1	0	0	1	1
i	0	1	0	1	0	1
u	0	1	0	1	1	1

Performance of the trained network

error (root mean squared error)



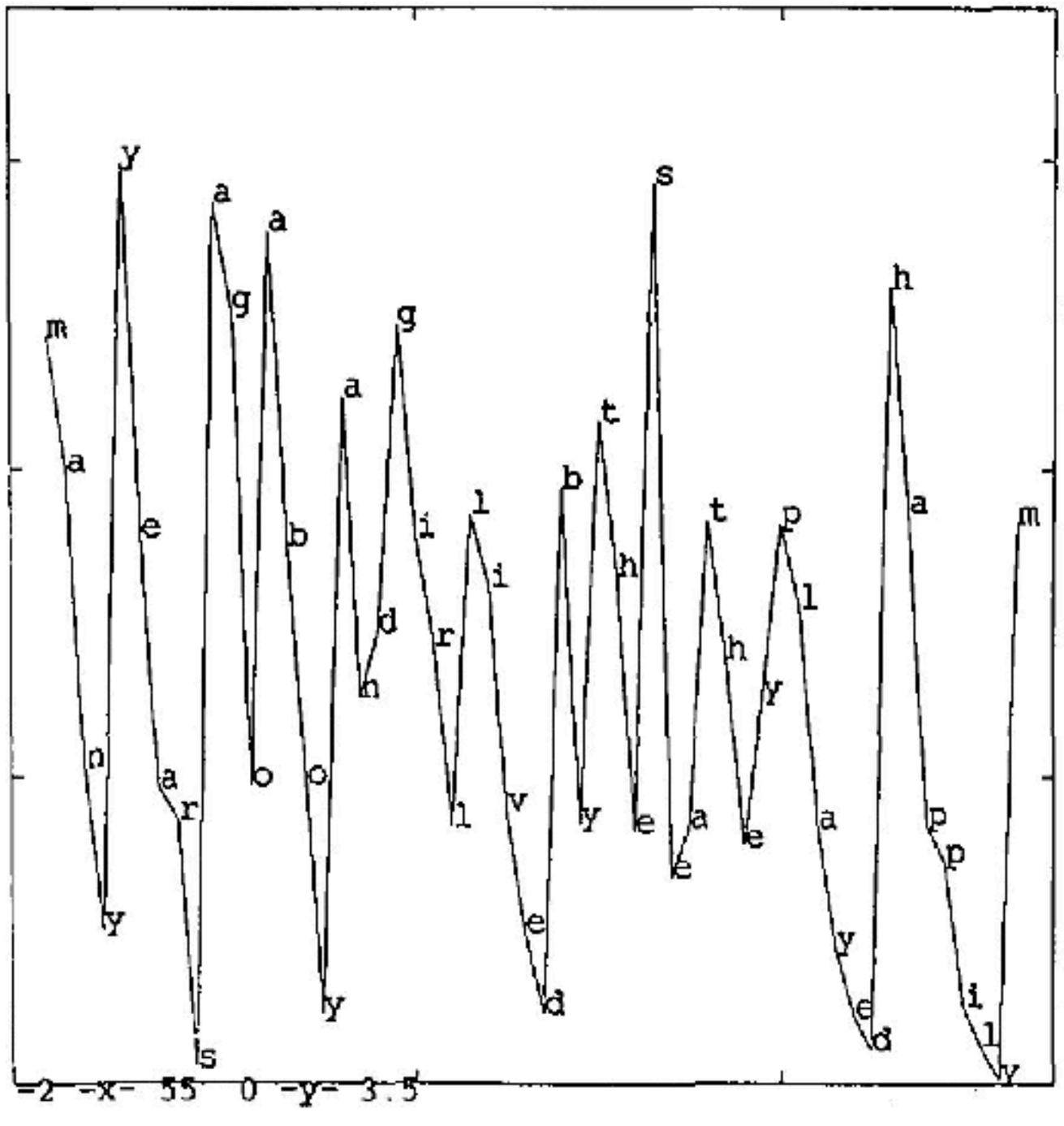
time

words “ba”, “dii”, and “guu”

Network shows higher error when predicting unpredictable characters (b, d, g)

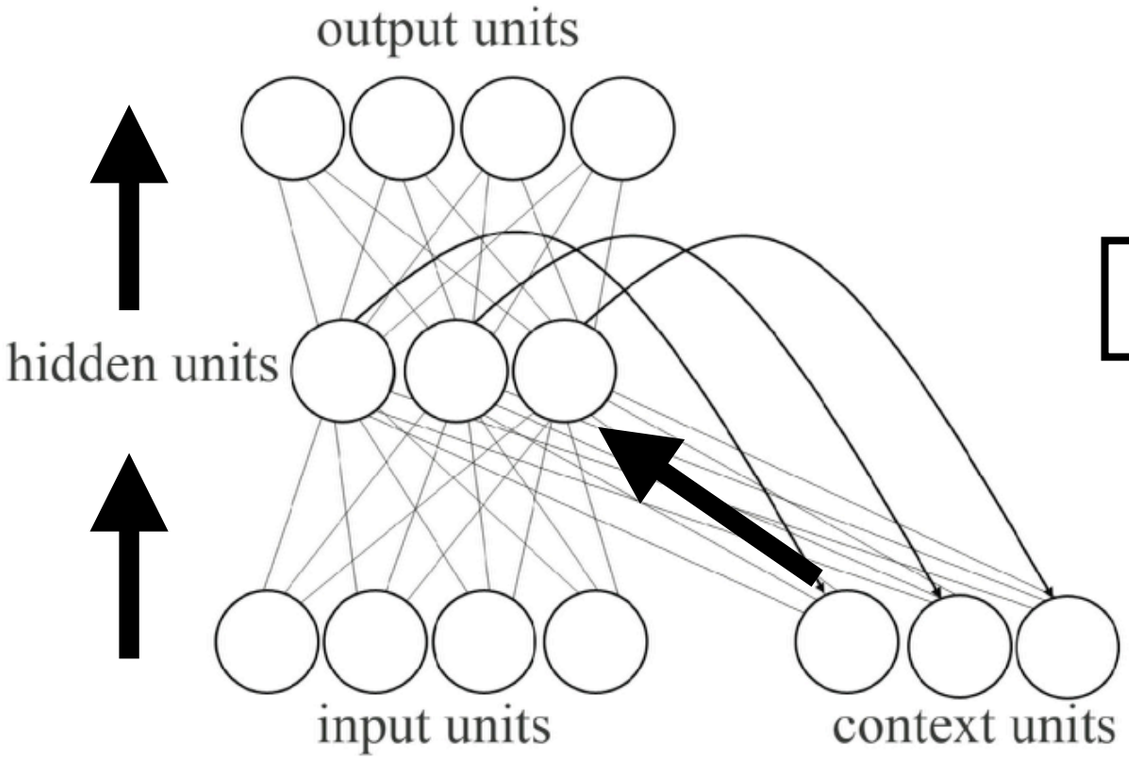
What is a “word”?

A similar simulations with a sequence of artificial sentences such as “manyyearsagoaboyandgirllivedbythesea..”

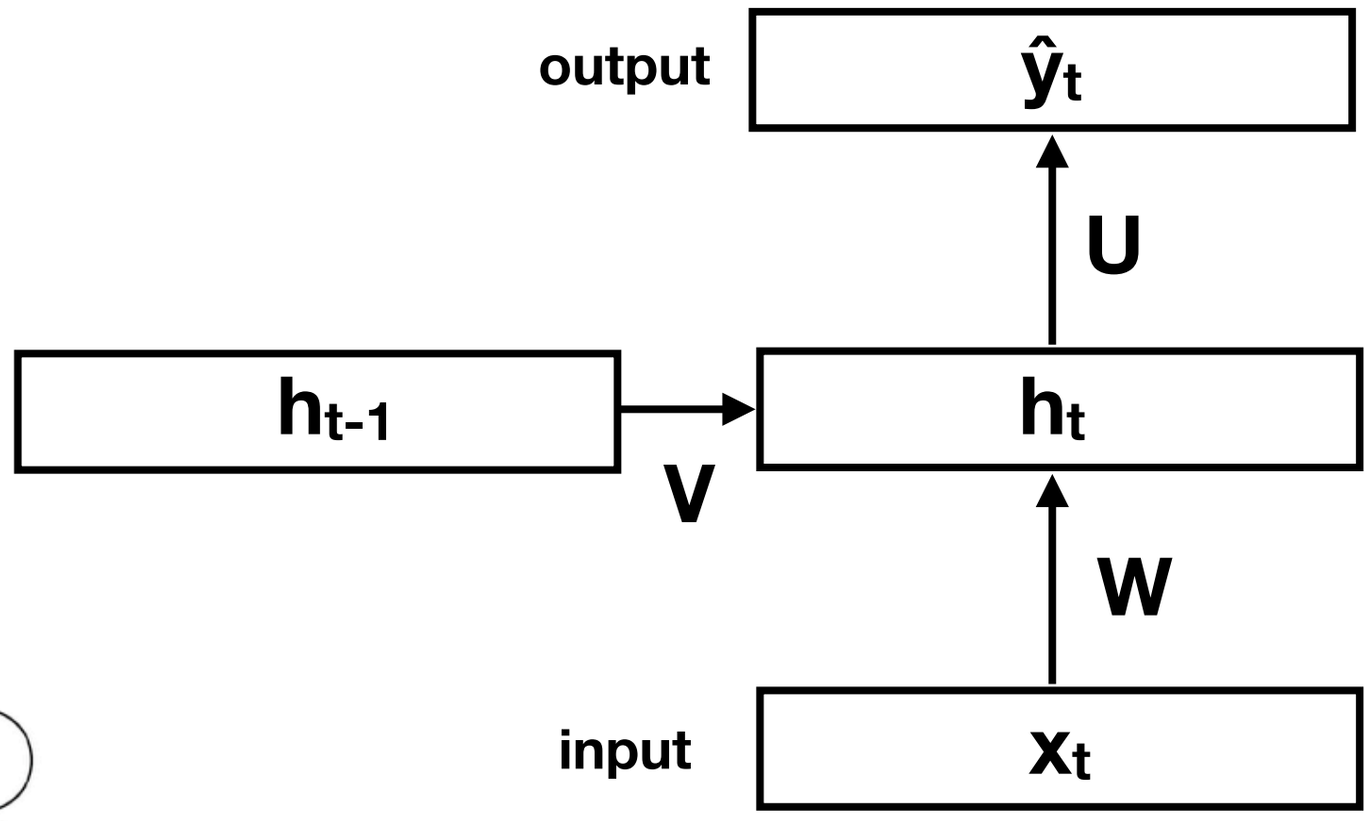


- Much previous work in cognitive science and linguistics assumed basic linguistic units such as phonemes, morphemes, words, etc.
- But the definition of a “word” isn’t that clear
- How many words is each of these phrases? (examples from PDP Handbook)
  - ‘line drive’, ‘flagpole’, ‘carport’, ‘gonna’, ‘wanna’, ‘hafta’, ‘isn’t’ and ‘didn’t Maybe “words” are just peaks in the error prediction signal?
- Elman’s paper and the SRN were the first to break away from commitments to basic linguistic units (phonemes, morphemes, words, etc.)

Training a recurrent neural network



notation for SRN



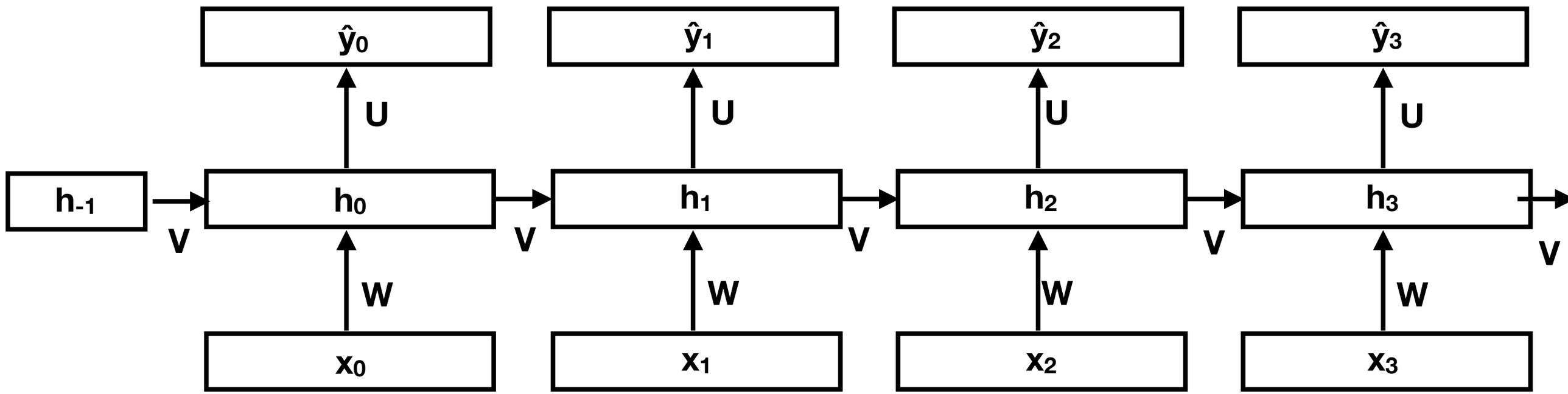
weight matrices V, W, U

index t is represents tie step

Unrolling a recurrent network in time

Global error $E = \sum_t E_t$

Error terms E_0 E_1 E_2 E_3

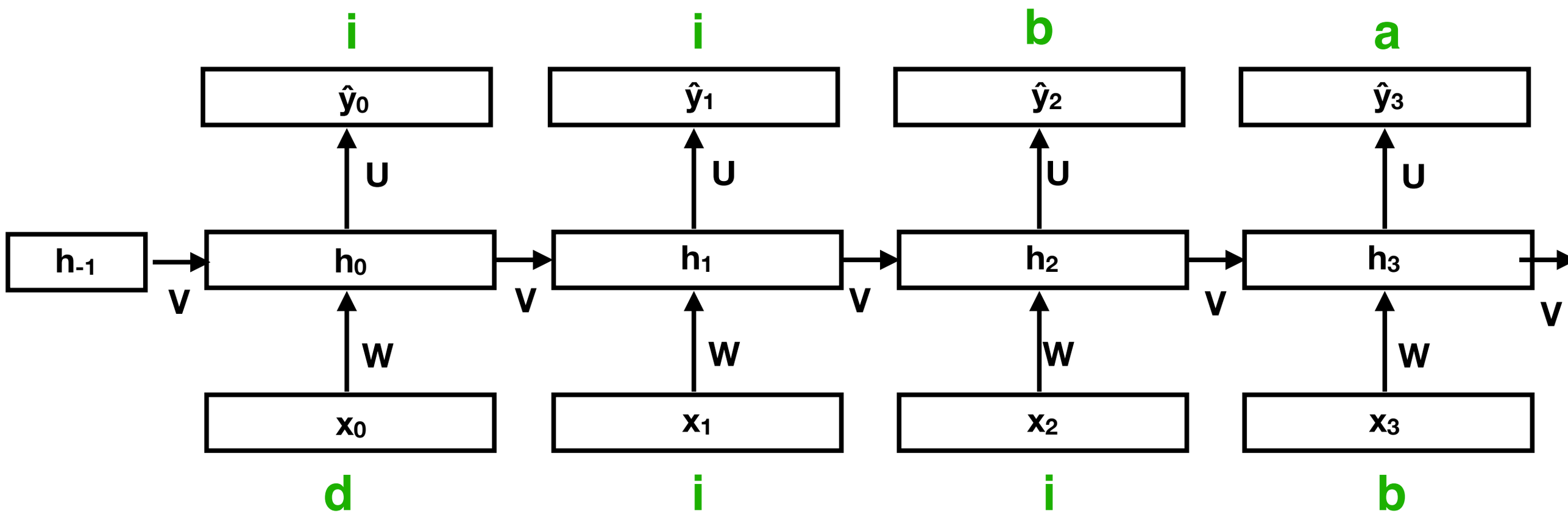


weight matrices V, W, U

Unrolling a recurrent network in time

Global error $E = \sum_t E_t$

Error terms E_0 E_1 E_2 E_3



weight matrices V, W, U

Input sequence:

diibaguuu

Reminder: Backpropagation algorithm for computing gradient

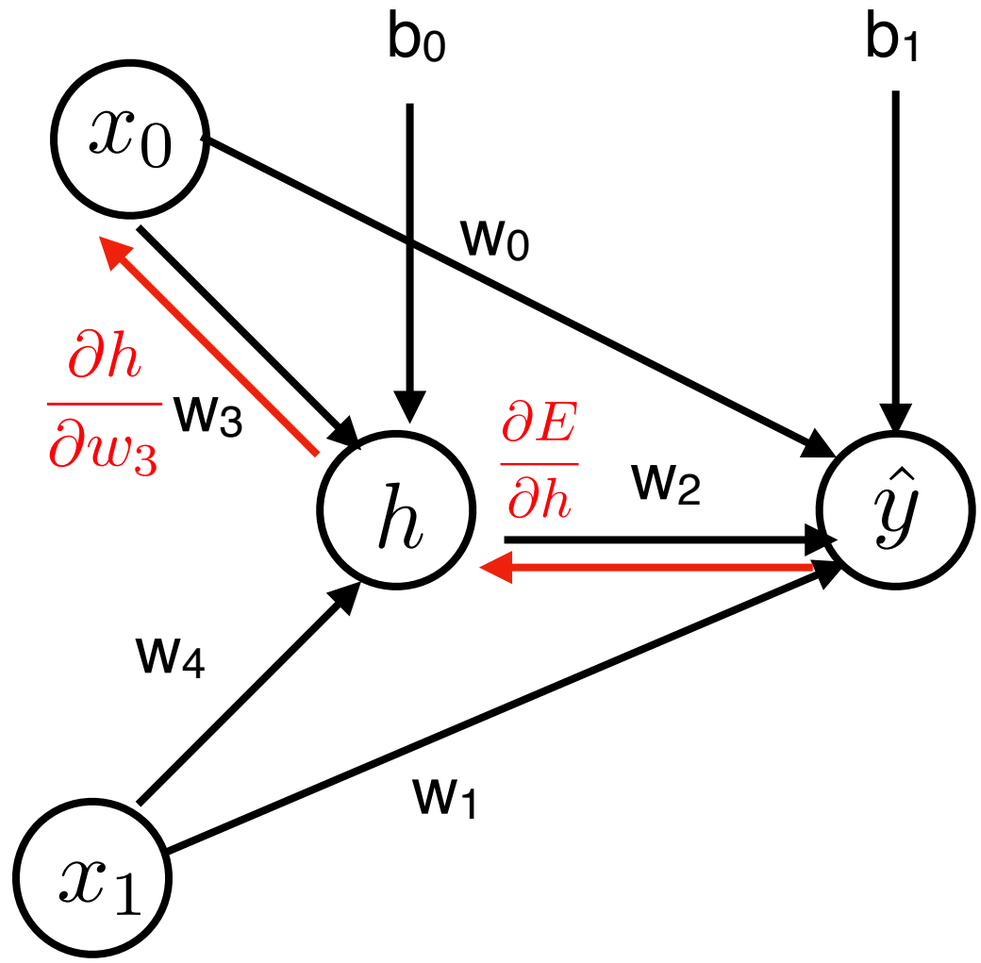
$$E(w,b) = (\hat{y} - y)^2$$
$$= (g(\text{net}) - y)^2$$

Multi-step strategy:

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial w_3}$$

Step 1) Compute how error changes as a function of hidden unit activation

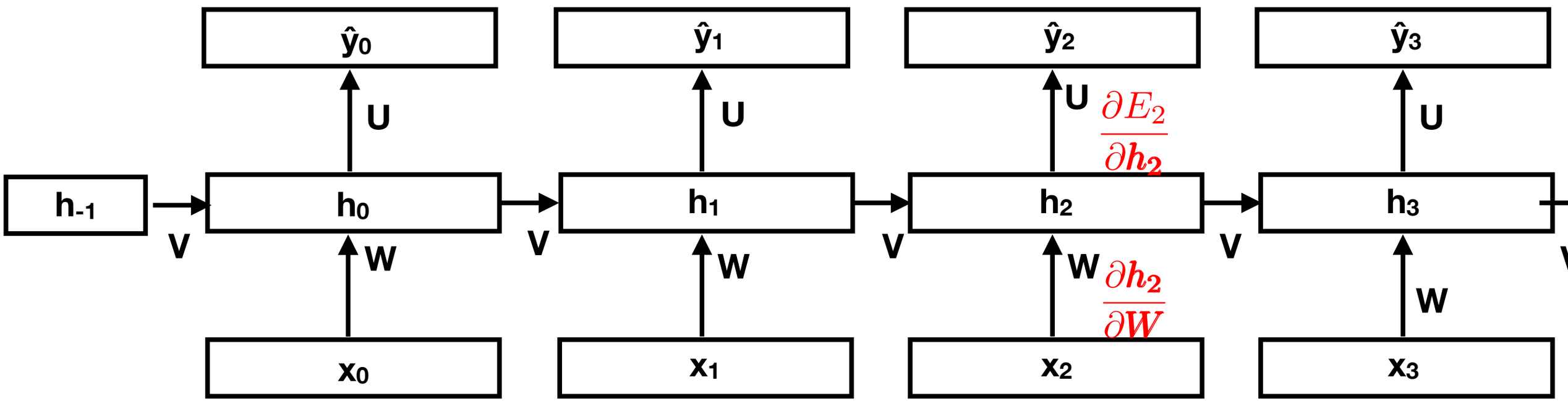
Step 2) Compute how hidden unit activation changes as a function of weight



Backpropagation through time

Global error $E = \sum_t E_t \qquad \frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$

Illustration for computing $\frac{\partial E_2}{\partial \boldsymbol{W}}$

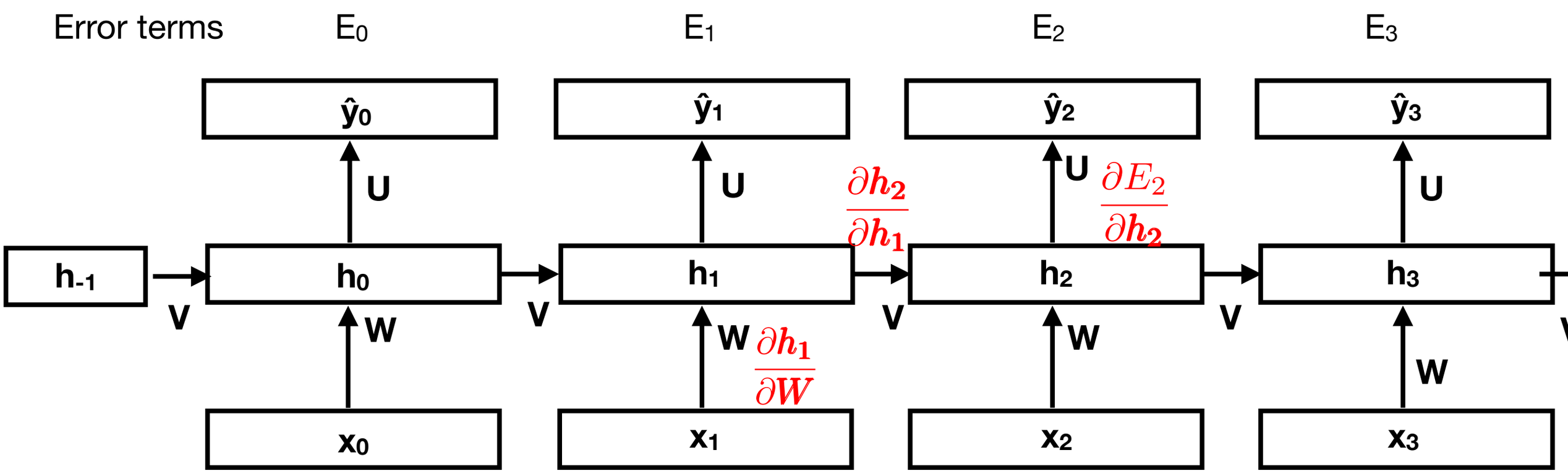


weight matrices V, W, U

Backpropagation through time

Global error $E = \sum_t E_t \qquad \frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$

Illustration for computing $\frac{\partial E_2}{\partial W}$



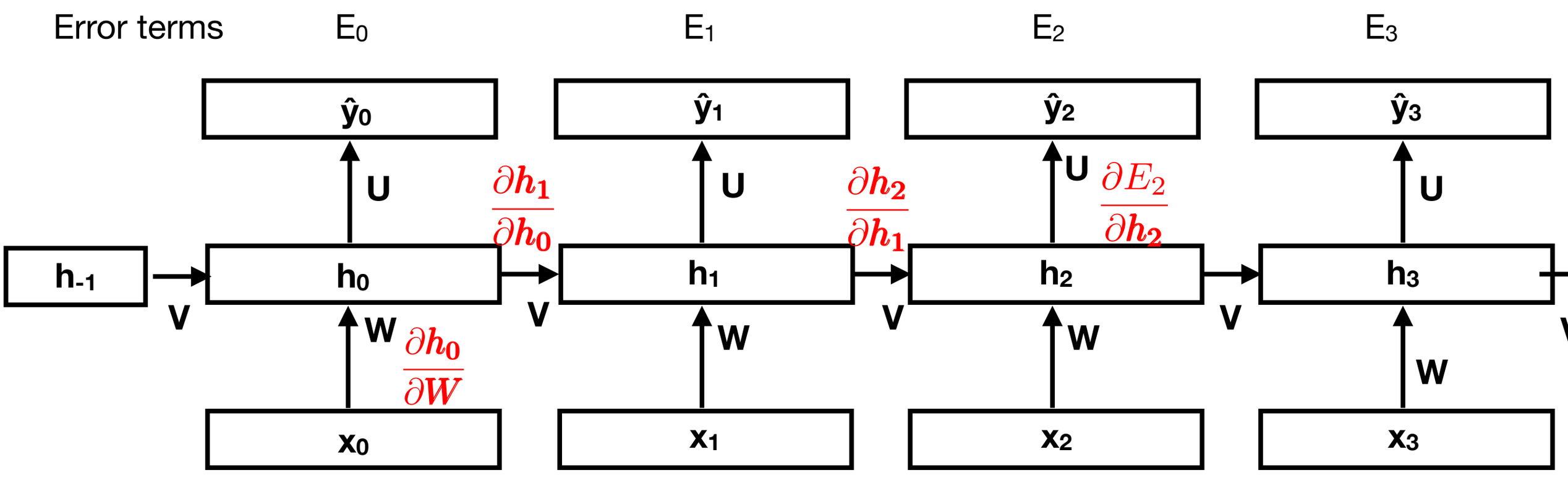
backprop step 2

weight matrices V, W, U

Backpropagation through time

Global error $E = \sum_t E_t$ $\quad \frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$

Illustration for computing $\frac{\partial E_2}{\partial W}$



backprop step 3

weight matrices V, W, U

Now, we sum the gradients from the last three slides.

And compute gradients for other time steps, E_0 , E_1 again summing over shared weights…

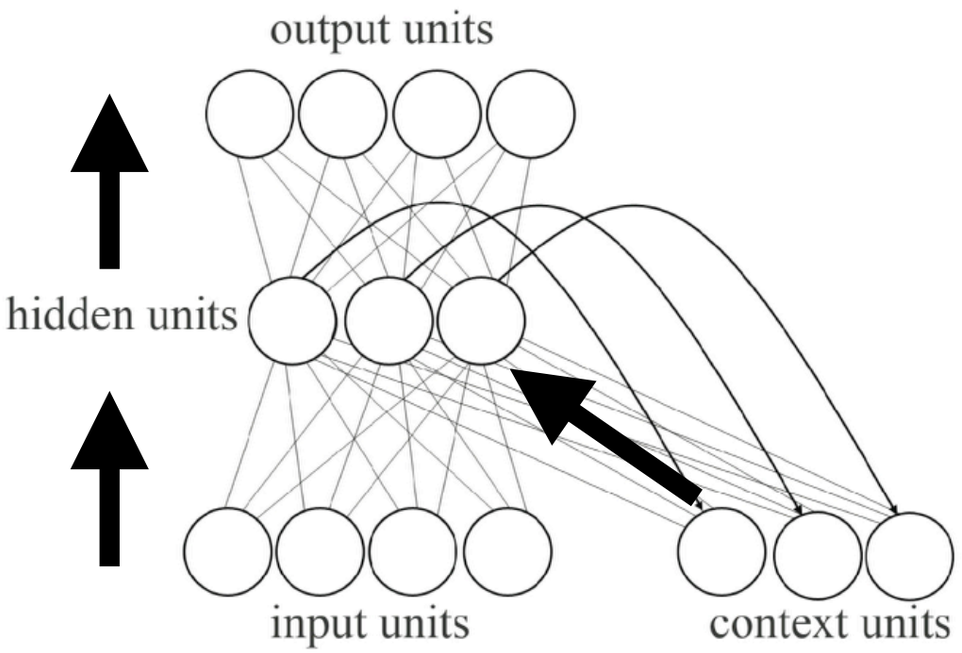
Conceptually, still no different than wiggling W and seeing how error changes!

Fortunately, PyTorch (or TensorFlow, etc.) can compute all of the gradients for you! (see code in Homework 2)

```
class SRN(nn.Module):

    def __init__(self, nsymbols, hidden_size):
        #  nsymbols: number of possible input/output symbols
        super(SRN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(nsymbols + hidden_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, nsymbols)
        self.softmax = nn.LogSoftmax()

    def forward(self, input, hidden):
        # input: 1 x nsymbol tensor, with one-hot encoding of a single symbol
        # hidden: 1 x hidden_size tensor, which is the previous hidden state
        combined = torch.cat((input, hidden), 1) # tensor size 1 x (nsymbol + hidden_size)
        hidden = self.i2h(combined) # 1 x hidden_size
        hidden = F.sigmoid(hidden)
        output = self.h2o(hidden) # 1 x nsymbol
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return Variable(torch.zeros(1, self.hidden_size))
```

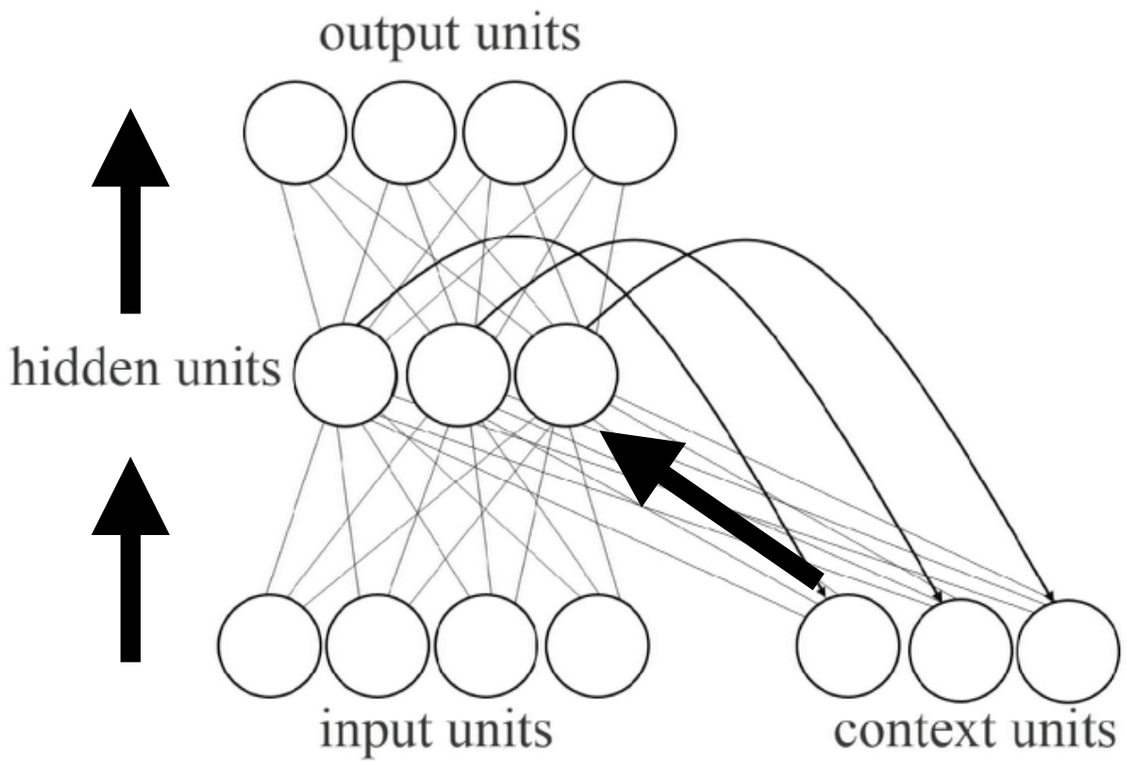
```
def train(seq_tensor, rnn):
    # seq_tensor: [seq_length x 1 x nsymbols tensor]
    # rnn : instance of SRN class
    hidden = rnn.initHidden()
    rnn.train()
    rnn.zero_grad()
    loss = 0
    seq_length = seq_tensor.size()[0]
    for i in range(seq_length-1):
        output, hidden = rnn(seq_tensor[i], hidden)
        loss += criterion(output, variableToIndex(seq_tensor[i+1]))
    loss.backward()
    optimizer.step()
    return loss.data.numpy()[0] / float(seq_length-1)
```

(this line computes all of the gradients for you.)



Discovering lexical classes from simple sentences

(also Elman, 1990)



“man eat cookie”
“woman see book”
“dragon eat human”
…

Can the SRN discover lexical classes like nouns and verbs?

Discovering lexical classes from simple sentences

Template for generating simple sentences

Categories of Lexical Items Used in Sentence Simulation

Category	Examples
NOUN-HUM	man, woman
NOUN-ANIM	cat, mouse
NOUN-INANIM	book, rock
NOUN-AGRESS	dragon, monster
NOUN-FRAG	glass, plate
NOUN-FOOD	cookie, break
VERB-INTRAN	think, sleep
VERB-TRAN	see, chase
VERB-AGPAT	move, break
VERB-PERCEPT	smell, see
VERB-DESTROY	break, smash
VERB-EAT	eat

Templates for Sentence Generator

WORD 1	WORD 2	WORD 3
NOUN-HUM	VERB-EAT	NOUN-FOOD
NOUN-HUM	VERB-PERCEPT	NOUN-INANIM
NOUN-HUM	VERB-DESTROY	NOUN-FRAG
NOUN-HUM	VERB-INTRAN	
NOUN-HUM	VERB-TRAN	NOUN-HUM
NOUN-HUM	VERB-AGPAT	NOUN-INANIM
NOUN-HUM	VERB-AGPAT	
NOUN-ANIM	VERB-EAT	NOUN-FOOD
NOUN-ANIM	VERB-TRAN	NOUN-ANIM
NOUN-ANIM	VERB-AGPAT	NOUN-INANIM
NOUN-ANIM	VERB-AGPAT	
NOUN-INANIM	VERB-AGPAT	
NOUN-AGRESS	VERB-DESTROY	NOUN-FRAG
NOUN-AGRESS	VERB-EAT	NOUN-HUM
NOUN-AGRESS	VERB-EAT	NOUN-ANIM
NOUN-AGRESS	VERB-EAT	NOUN-FOOD

“man eat cookie”
“woman see book”
“dragon eat human”
...

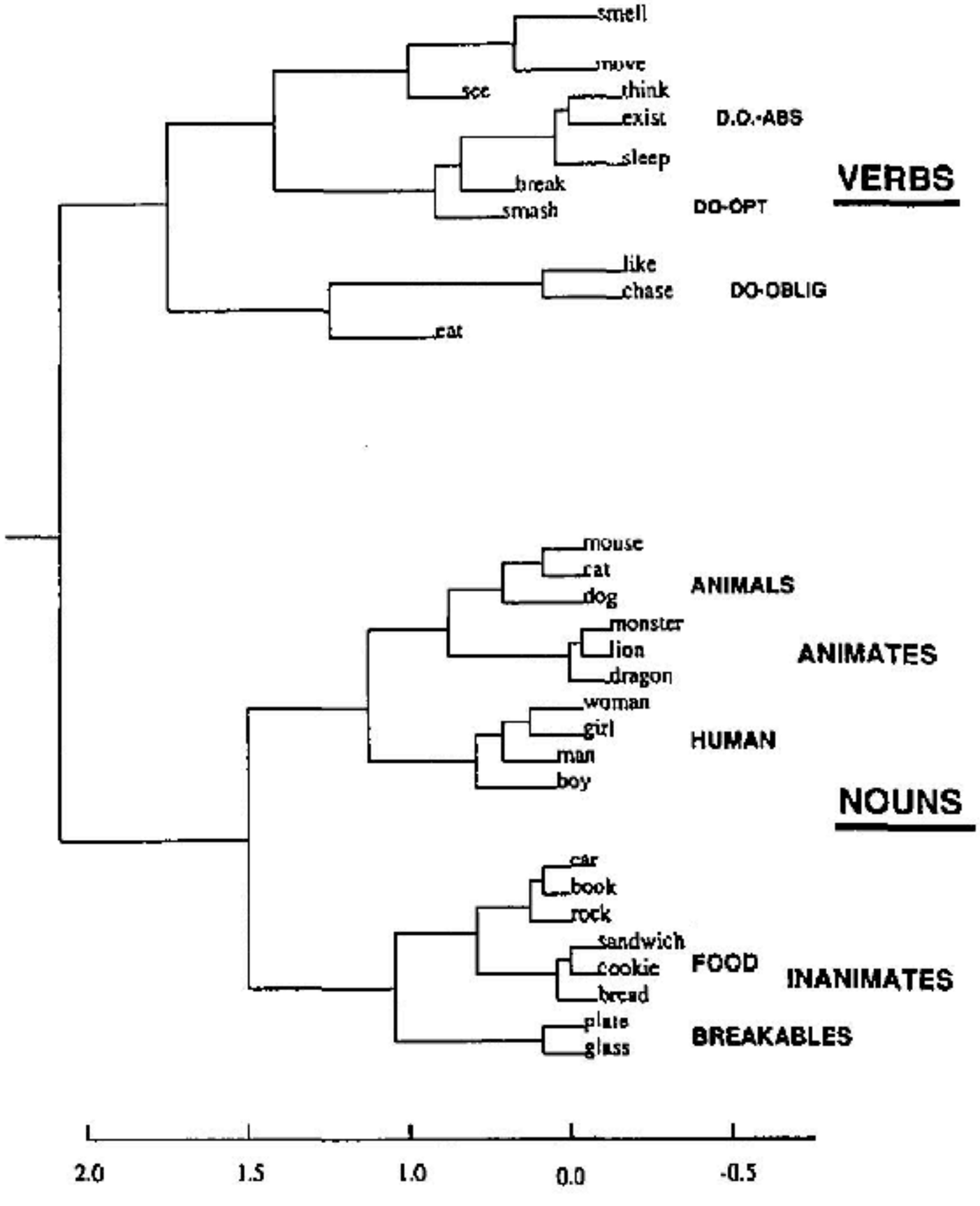
Discovering lexical classes from simple sentences

"one-hot" encoding for words

TABLE 5
Fragment of Training Sequences for Sentence Simulation

Input	Output
00000000000000000000000000000010 (*woman*)	00000000000000000000000000010000 (*smash*)
00000000000000000000000000010000 (*smash*)	00000000000000000000001000000000 (*plate*)
00000000000000000000001000000000 (*plate*)	00000100000000000000000000000000 (*cat*)
00000100000000000000000000000000 (*cat*)	00000000000000000000100000000000 (*move*)
00000000000000000000100000000000 (*move*)	00000000000000000100000000000000 (*man*)
00000000000000000100000000000000 (*man*)	00010000000000000000000000000000 (*break*)
00010000000000000000000000000000 (*break*)	00001000000000000000000000000000 (*car*)
00001000000000000000000000000000 (*car*)	01000000000000000000000000000000 (*boy*)
01000000000000000000000000000000 (*boy*)	00000000000000000000100000000000 (*move*)
00000000000000000000100000000000 (*move*)	00000000000010000000000000000000 (*girl*)
00000000000010000000000000000000 (*girl*)	00000000001000000000000000000000 (*eat*)
00000000001000000000000000000000 (*eat*)	00100000000000000000000000000000 (*bread*)
00100000000000000000000000000000 (*bread*)	00000000100000000000000000000000 (*dog*)
00000000100000000000000000000000 (*dog*)	00000000000000000000100000000000 (*move*)
00000000000000000000100000000000 (*move*)	00000000000000000001000000000000 (*mouse*)
00000000000000000001000000000000 (*mouse*)	00000000000000000001000000000000 (*mouse*)
00000000000000000001000000000000 (*mouse*)	00000000000000000000100000000000 (*move*)
00000000000000000000100000000000 (*move*)	10000000000000000000000000000000 (*book*)
10000000000000000000000000000000 (*book*)	00000000000000001000000000000000 (*lion*)

Discovering lexical classes from simple sentences



- The network just learned to **predict the next word** from the **previous word**
- This diagram shows results of clustering the average pattern over the hidden units for each word (across many presentations)
- From just the prediction task, the network “discovers” nouns vs. verbs, and also animate vs inanimate, etc. without building in these lexical classes in any way

It's a relatively small step to state-of-the-art recurrent neural networks for text processing and machine translation

- Simple recurrent networks (SRNs) are not used very often anymore. Instead, they have been replaced by more powerful Long short term memory (LSTM) recurrent networks (Hochreiter & Schmidhuber, 1997) or Gated Recurrent Unit (GRU) networks
- Conceptually, LSTMs are recurrent networks just like SRNs, except the "hidden state" can more easily pass from one step to another without modification, unless the network decides to modify it. This gives the network a much longer memory.
- For state-of-the-art text modeling, networks are trained to predict the next word given the previous, just as we studied in the previous examples. Training is with backprop through time

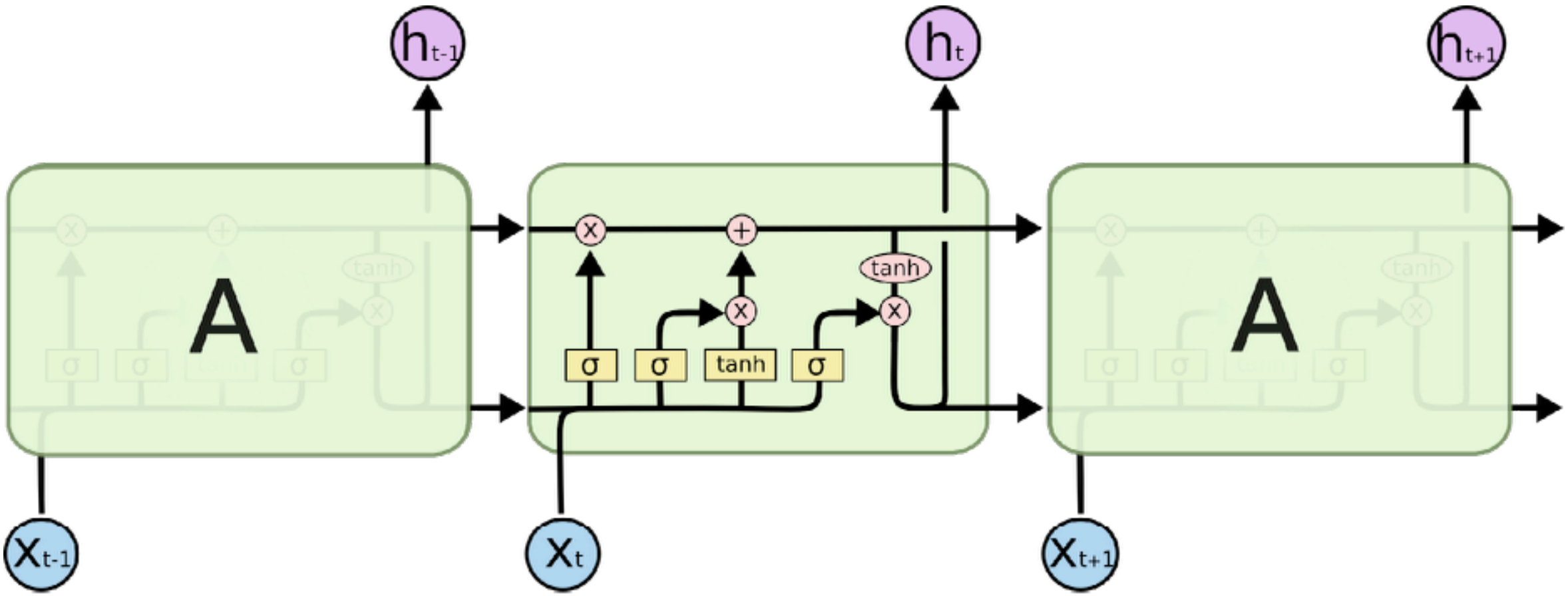
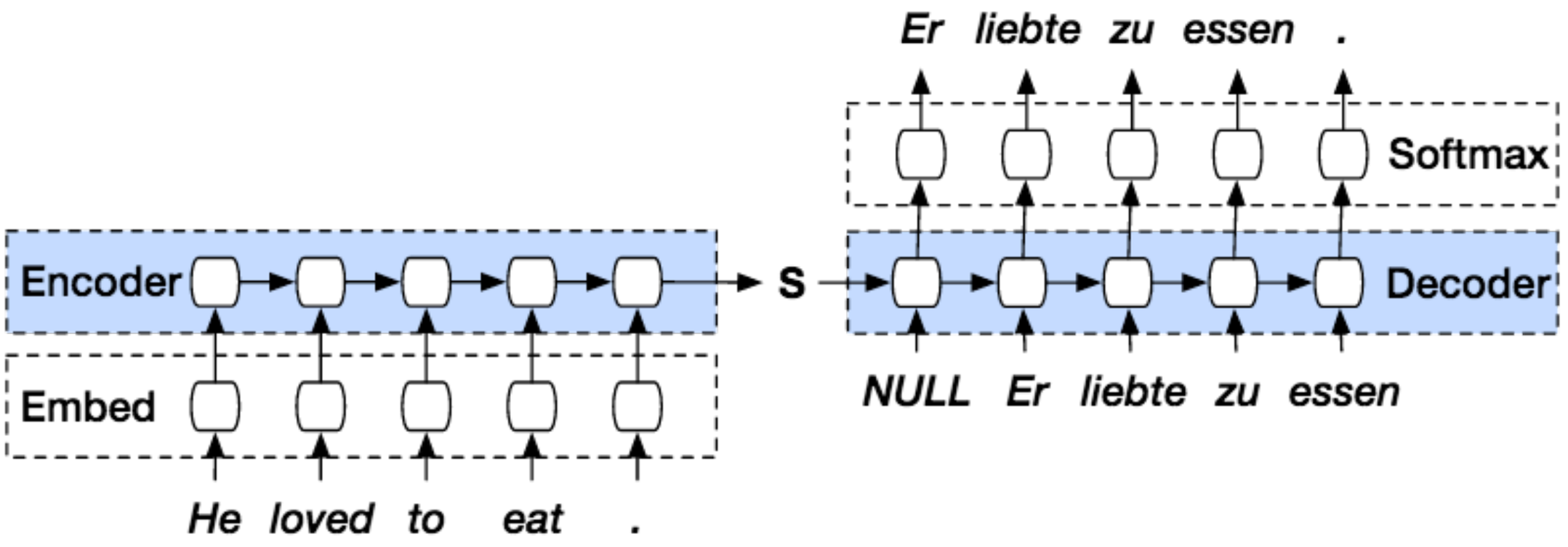


Image credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Neural machine translation with recurrent neural networks

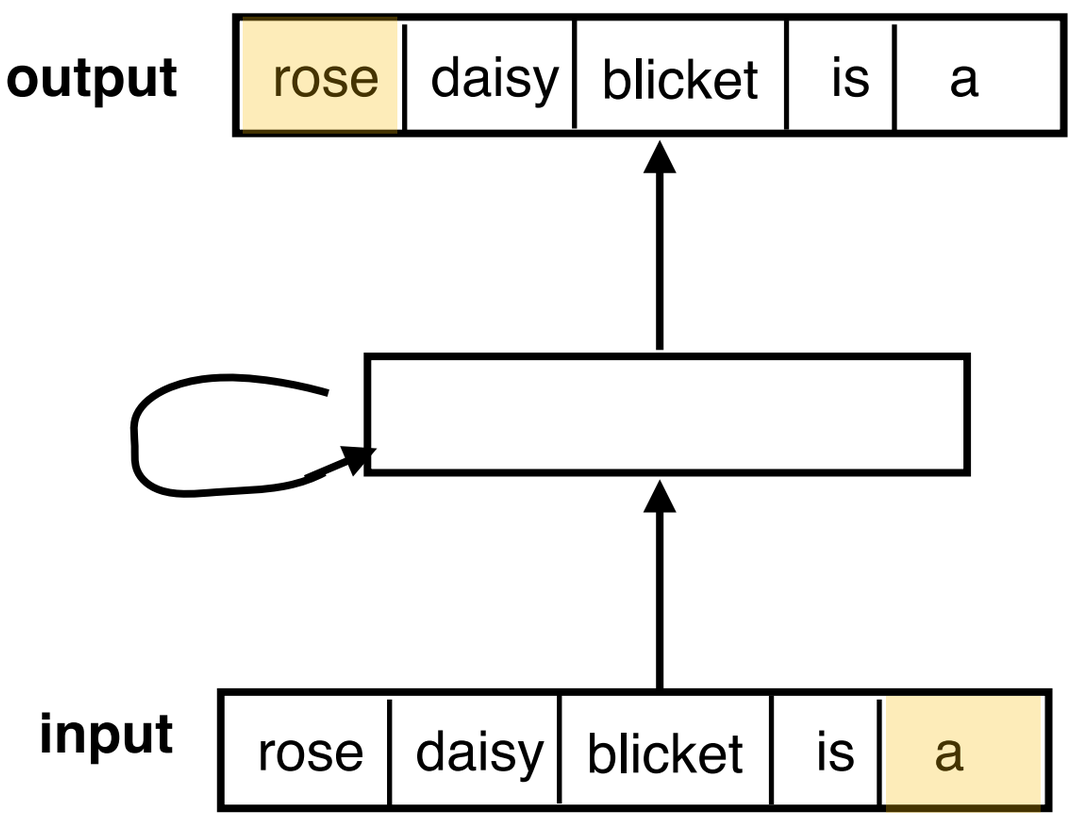
- Recurrent neural networks can solve sequence-to-sequence (seq2seq) problems by having a recurrent encoder, which embeds a sentence in the hidden space, and another recurrent decoder that translates it.
- This is the main technical juice behind Google Translate!
- Trained using sentence pairs and backprop through time



(e.g.,Sutskever, Vinyals, & Le, 2014)

Critiques of recurrent neural networks

(Marcus, 1998, Rethinking eliminative connectionism)

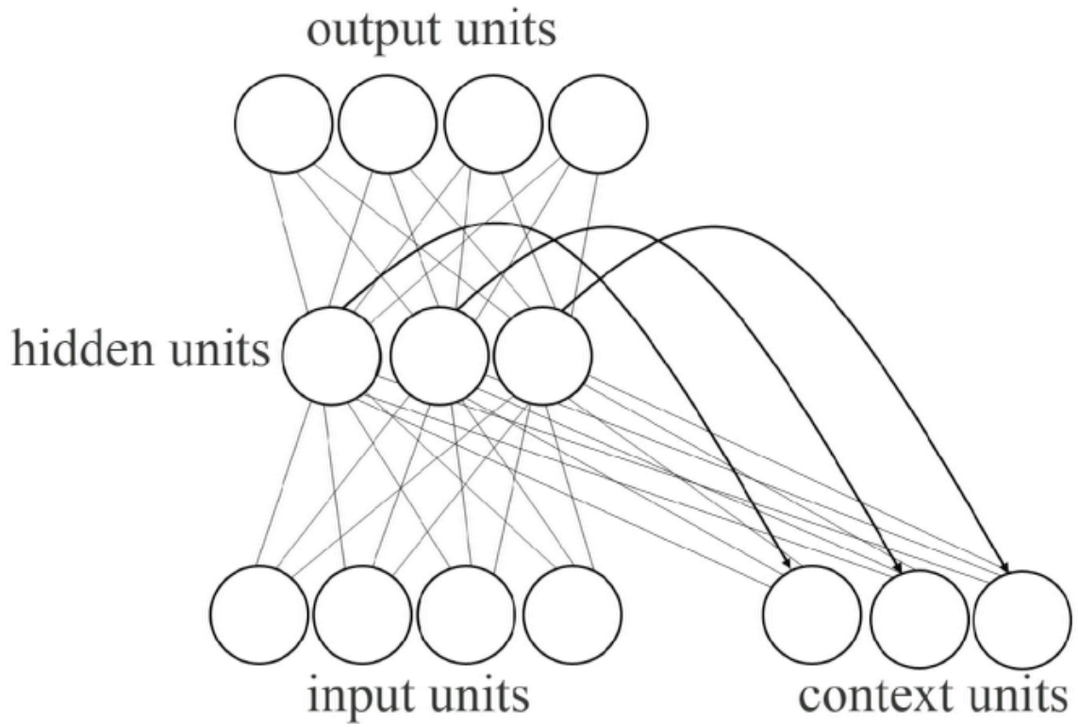


Sentence input… “A rose is a”

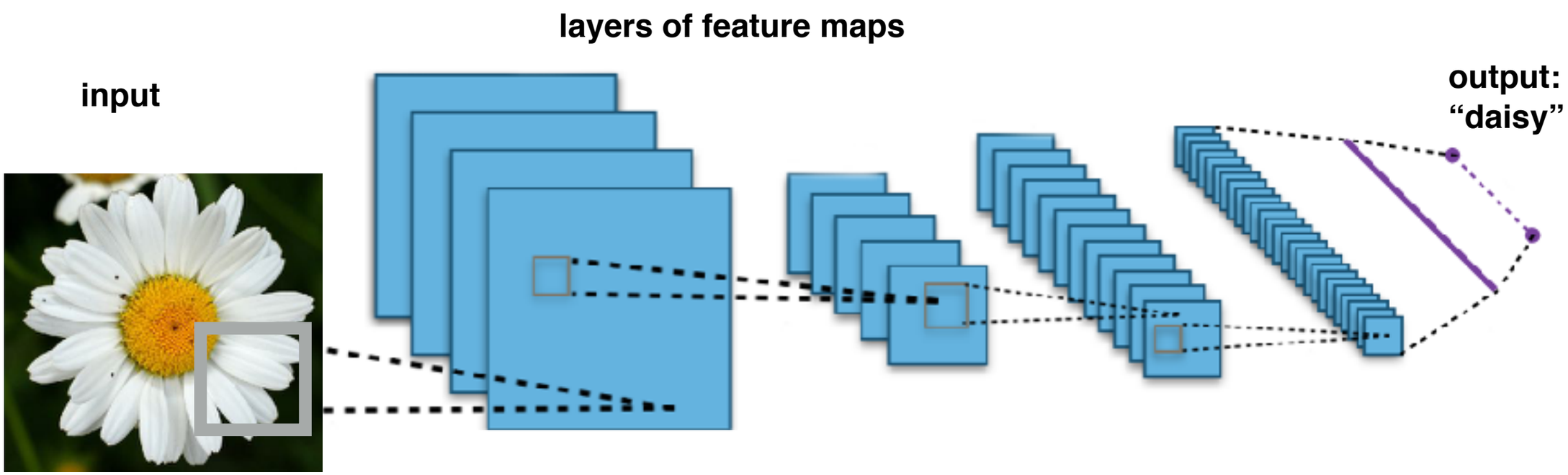
- Recurrent neural networks (RNNs) generalize very well WITHIN a specific set of words / symbols
- However, they generalize very poorly to new words OUTSIDE the training set
- Say the network is trained on sentences such as, “a rose is a rose”, “a daisy is a daisy”, and “a violet is a violet”
- It will NOT generalize to a new word, “a blicket is a blicket”
- People can easily learn rules that apply to arbitrary new variables

Today's lecture: Two very important types of neural network models

Recurrent neural network

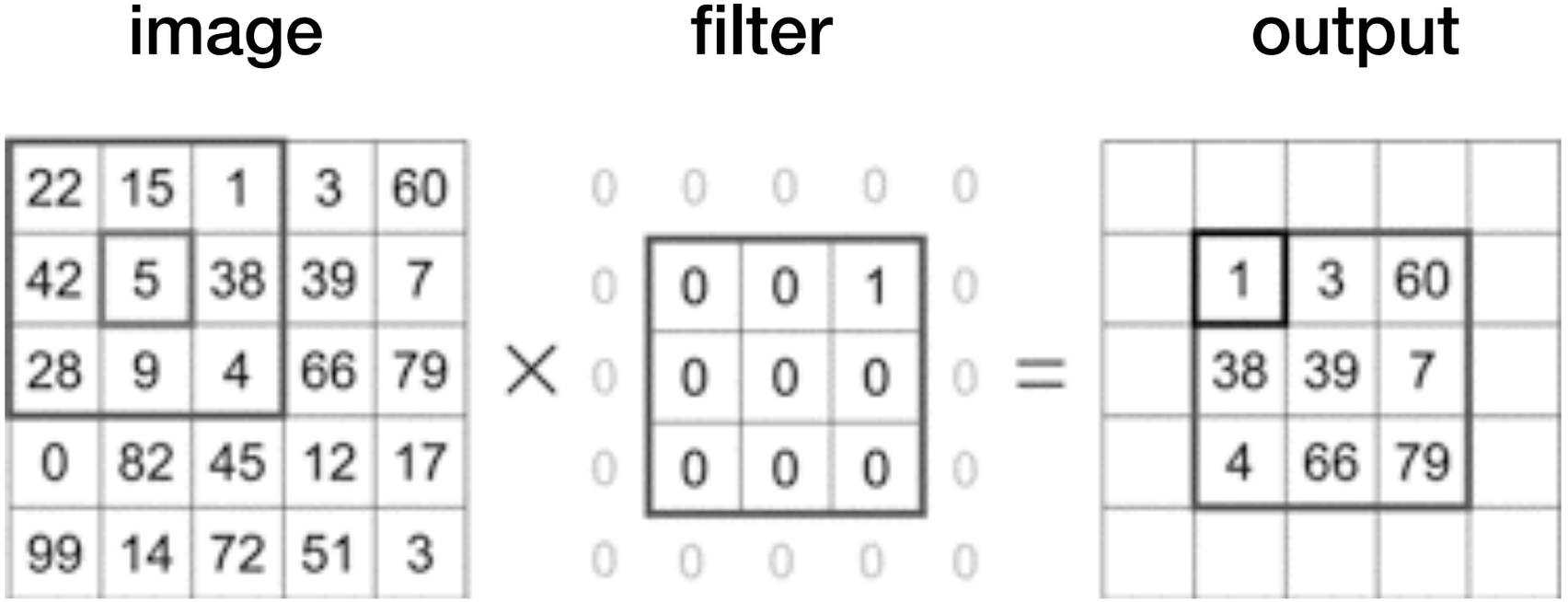


Deep convolutional neural network



convolving an image with a filter

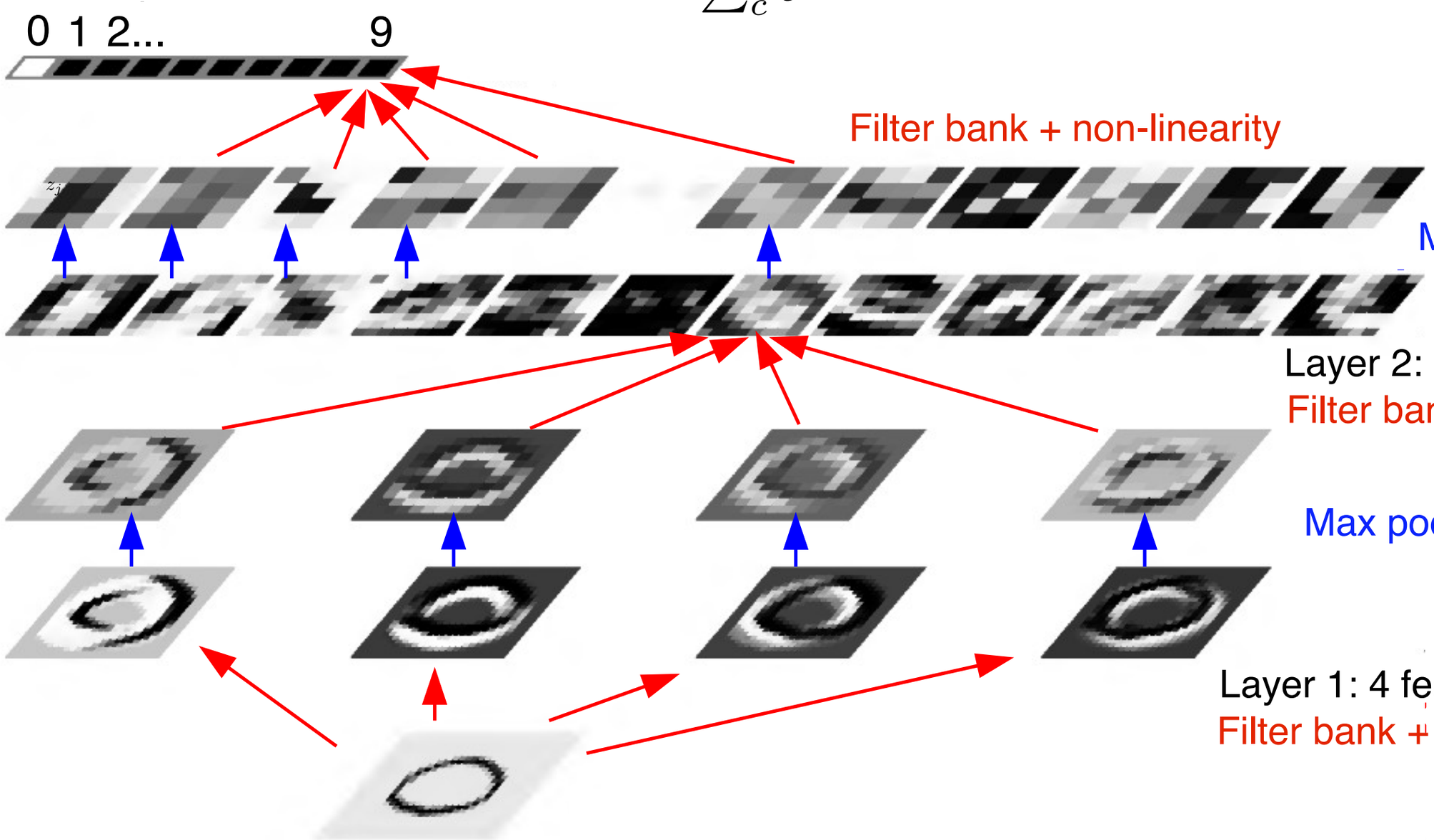
We slide the filter across the image to produce an output image



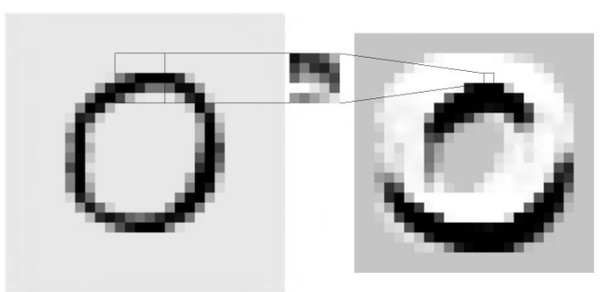
Deep convolutional neural network for vision

output classes (c)

$$g(\mathbf{net}_i) = \frac{e^{-\mathbf{net}_i}}{\sum_c e^{-\mathbf{net}_c}}$$

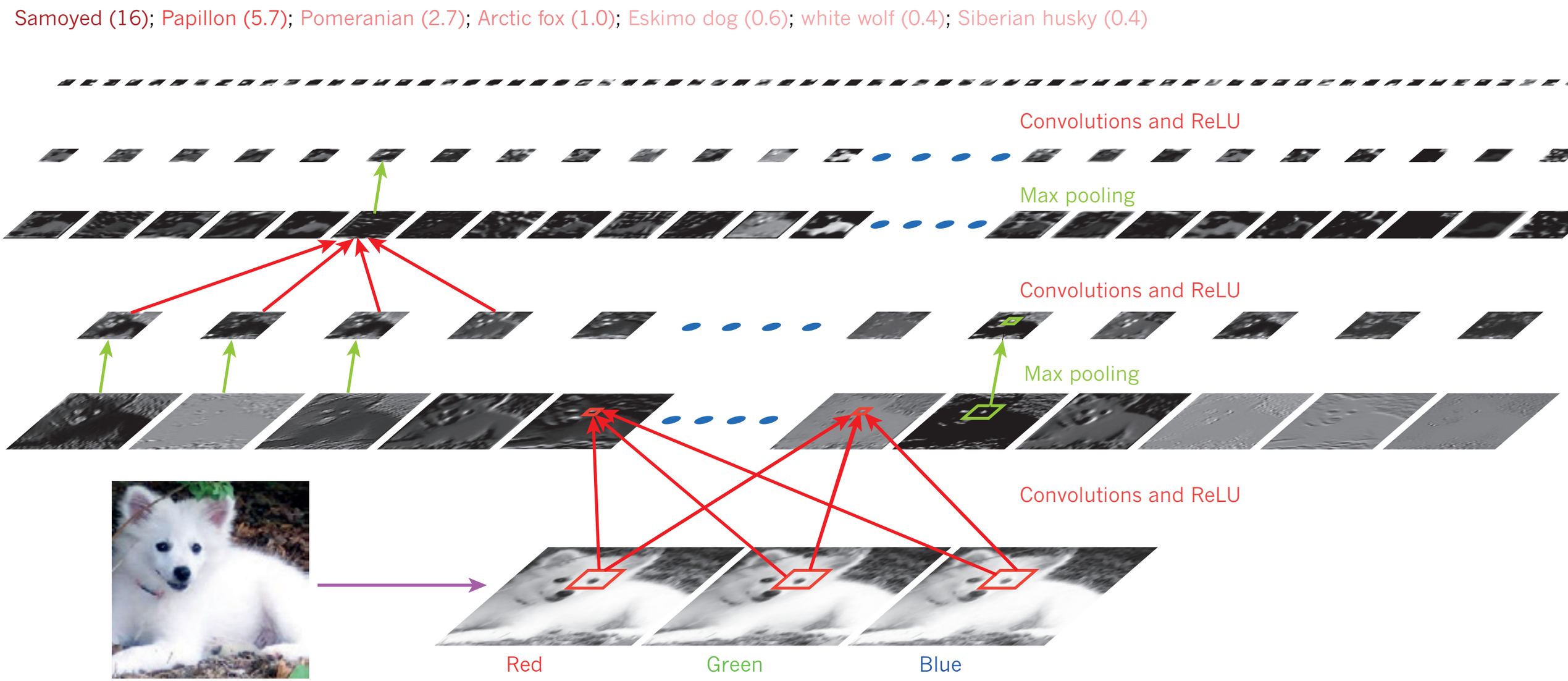


example convolution



Slide courtesy of Yann LeCun

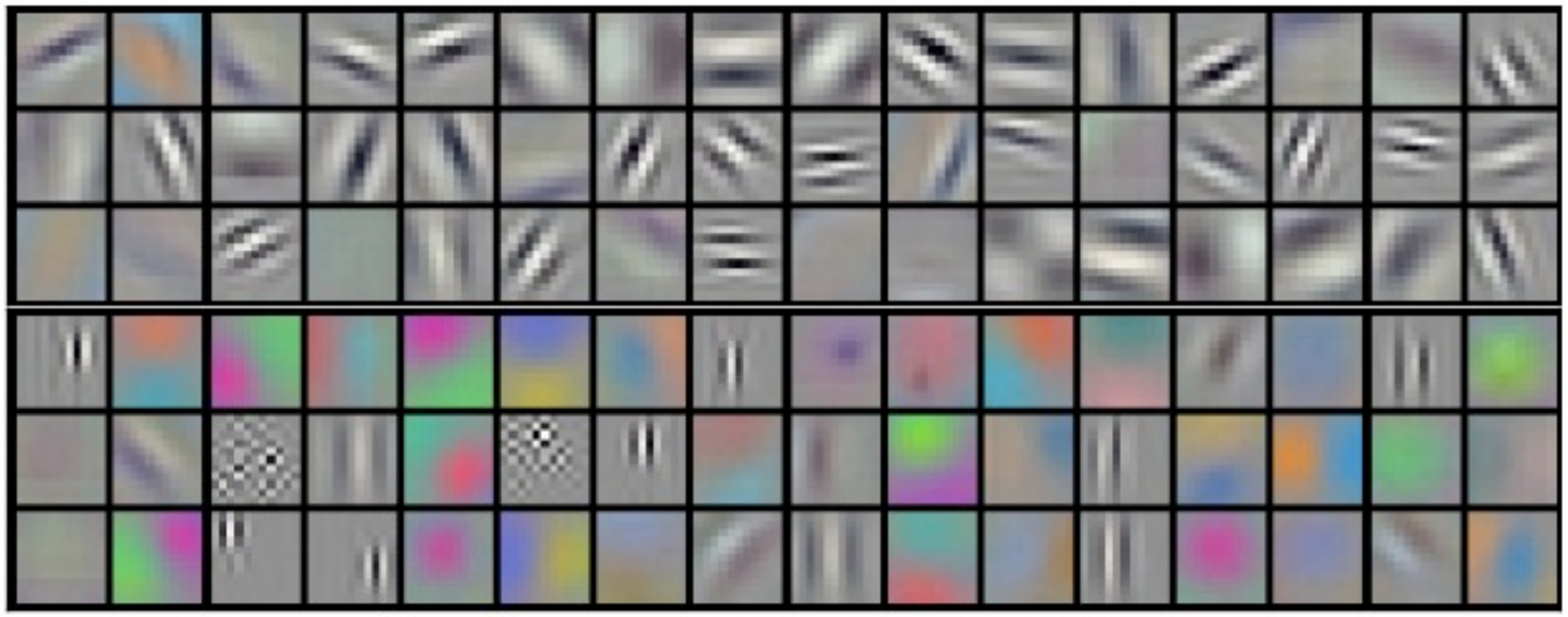
Deep convolutional neural network



From LeCun, Bengio, & Hinton (2015).

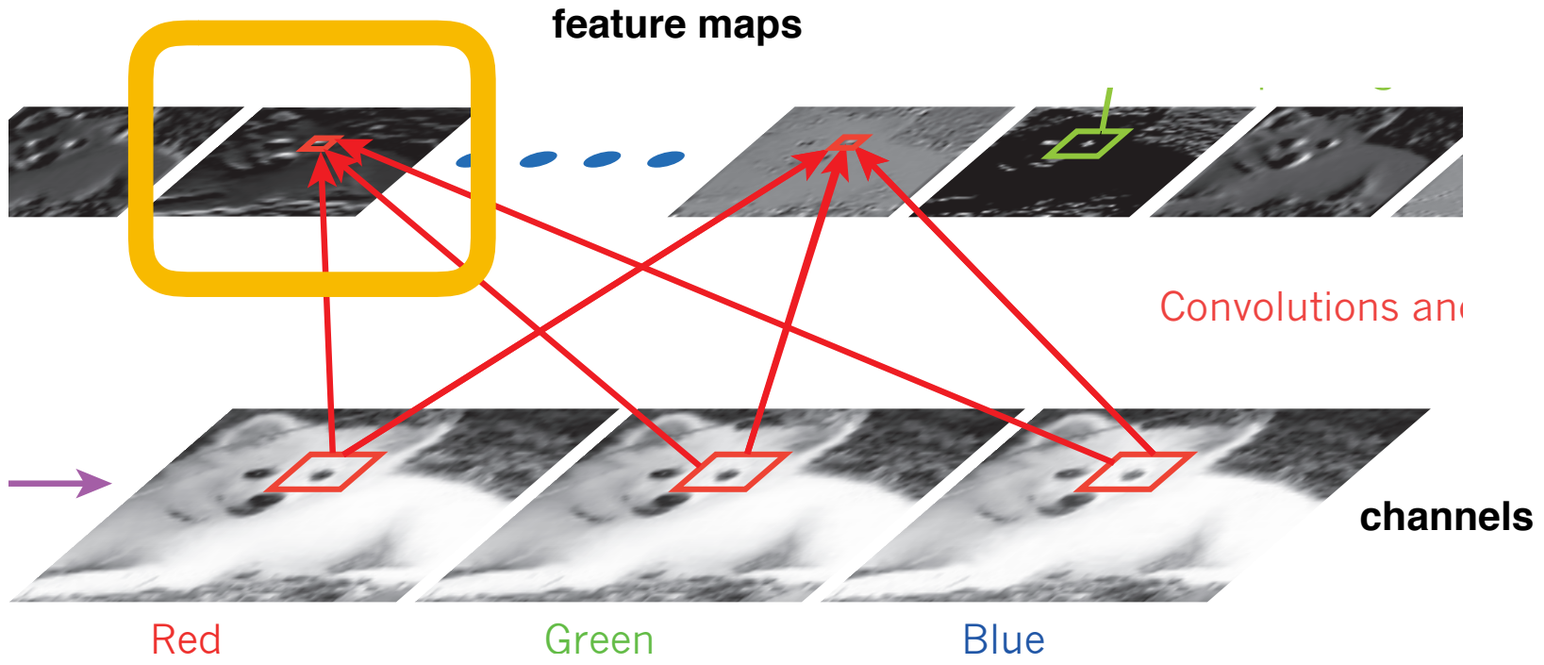
Learned filters in a deep convnet

- Key assumption of “translation invariance”: If a filter (e.g., a horizontal edge detector) is useful in one part of the image, it is probably useful anywhere in the image



(some filters learned from Krizhevsky et al., 2012)

Let’s go through computing with convolutions to build a feature map…



Filter 0 **Filter 1** **Next layer of network**

Channel 0 (e.g., red)

Channel 1 (e.g., green)

Channel 2 (e.g., blue)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	0	0	1	0	0
0	2	2	0	2	0	0
0	1	0	2	0	1	0
0	2	0	0	0	2	0
0	0	0	2	0	2	0
0	0	0	0	2	1	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	2	0	1	0
0	0	0	0	0	2	0
0	0	0	2	1	0	0
0	2	1	1	1	2	0
0	1	1	0	2	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	1	0	0	0
0	1	1	2	2	0	0
0	2	2	1	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

dot product (element-wise multiply then sum)

dot product

dot product

Filter W0 (3x3x3)

w0[:,:,0]

0	1	0
-1	1	0
0	1	1

w0[:,:,1]

-1	-1	1
0	1	0
0	1	0

w0[:,:,2]

1	-1	-1
0	0	0
0	0	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	-1
0	0	1
1	1	-1

w1[:,:,1]

1	1	-1

w1[:,:,2]

0	-1	0
-1	1	0
-1	-1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

sum

Output Volume (3x3x2)

o[:,:,0]

8	6	5
7	9	2
-1	7	3

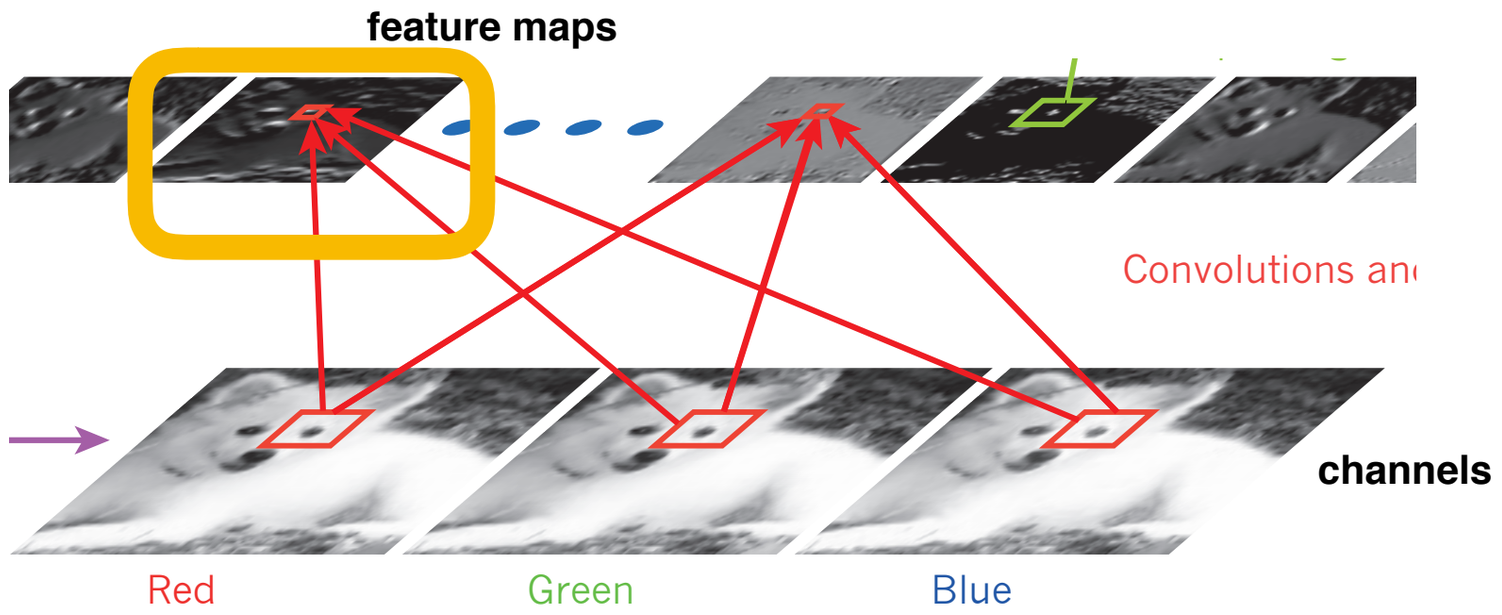
o[:,:,1]

2	3	7
-2	1	-2
-5	-1	-2

Feature map 0

Feature map 1

Credit for demo: http://cs231n.github.io/convolutional-networks/



Filter 0 **Filter 1** **Next layer of network**

Channel 0 (e.g., red)

Channel 1 (e.g., green)

Channel 2 (e.g., blue)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	0	0	2	2	0
0	2	2	1	2	1	0
0	1	0	2	1	0	0
0	2	0	2	0	1	0
0	0	0	2	0	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	2	0	1	0
0	0	0	0	0	2	0
0	0	0	2	1	0	0
0	2	1	1	1	2	0
0	1	1	2	2	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	1	0	0	0
0	1	1	2	2	0	0
0	2	2	1	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	1	0
-1	1	0
0	1	1

w0[:,:,1]

-1	-1	1
0	1	0
0	1	0

w0[:,:,2]

1	-1	-1
0	0	0
0	0	0

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	-1
0	0	1
1	1	-1

w1[:,:,1]

0	-1	-1
-1	1	1
1	1	-1

w1[:,:,2]

0	-1	0
-1	1	0
-1	-1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

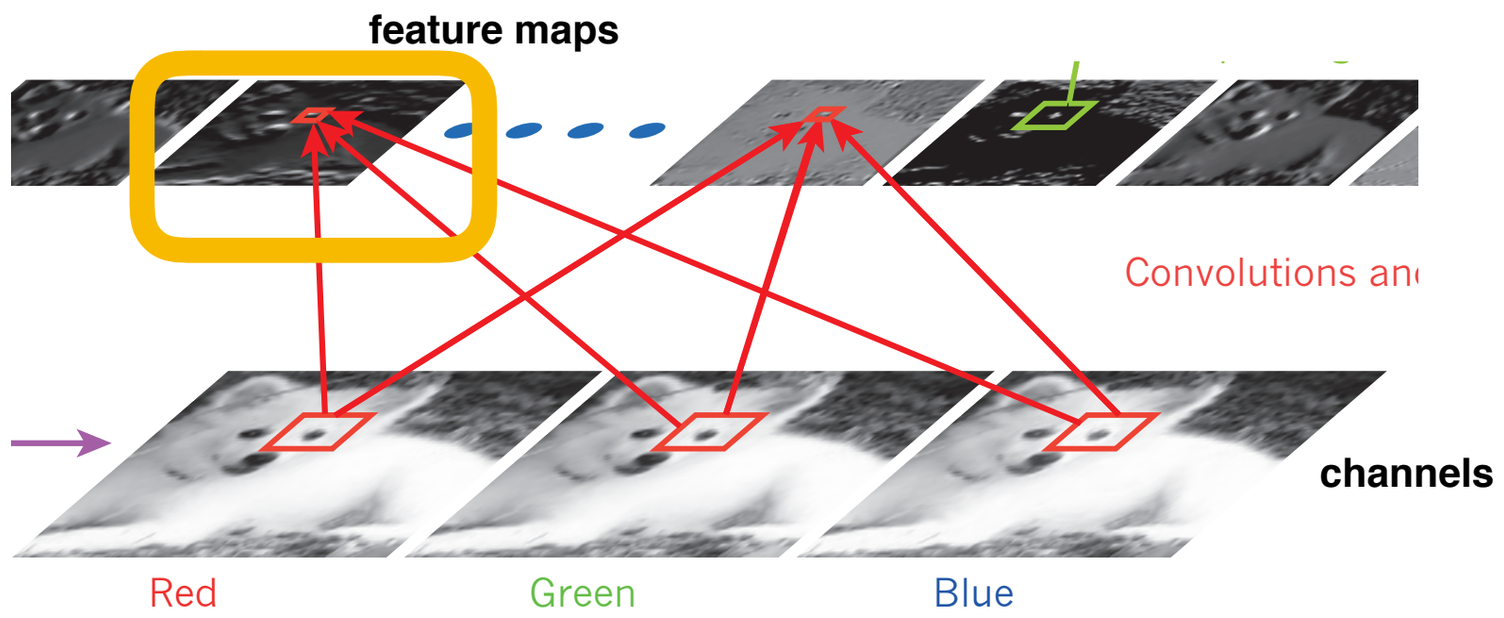
8	6	5
7	9	2
-1	7	3

Feature map 0

o[:,:,1]

2	3	7
-2	1	-2
-5	-1	-2

Feature map 1



Filter 0 **Filter 1** **Next layer of network**

Input Volume (+pad 1) (7x7x3)

`x[:,:,0]`

0	0	0	0	0	0	0
0	2	0	0	2	2	0
0	2	2	1	2	1	0
0	1	0	2	1	0	0
0	2	0	2	0	1	0
0	0	0	2	0	2	0
0	0	0	0	0	0	0

Channel 0

(e.g., red)

`x[:,:,1]`

0	0	0	0	0	0	0
0	1	0	2	0	1	0
0	0	0	0	0	2	0
0	0	0	2	1	0	0
0	2	1	1	1	2	0
0	1	1	2	2	2	0
0	0	0	0	0	0	0

Channel 1

(e.g., green)

`x[:,:,2]`

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	1	0	0	0
0	1	1	2	2	0	0
0	2	2	1	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

Channel 2

(e.g., blue)

Filter W0 (3x3x3)

`w0[:,:,0]`

0	1	0
-1	1	0
0	1	1

`w0[:,:,1]`

-1	-1	1
0	1	0
0	1	0

`w0[:,:,2]`

1	-1	-1
0	0	0
0	0	0

Bias b0 (1x1x1)

`b0[:,:,0]`

1

Filter W1 (3x3x3)

`w1[:,:,0]`

0	-1	-1
0	0	1
1	1	-1

`w1[:,:,1]`

0	-1	-1
-1	1	1
1	1	-1

`w1[:,:,2]`

0	-1	0
-1	1	0
-1	-1	0

Bias b1 (1x1x1)

`b1[:,:,0]`

0

Output Volume (3x3x2)

`o[:,:,0]`

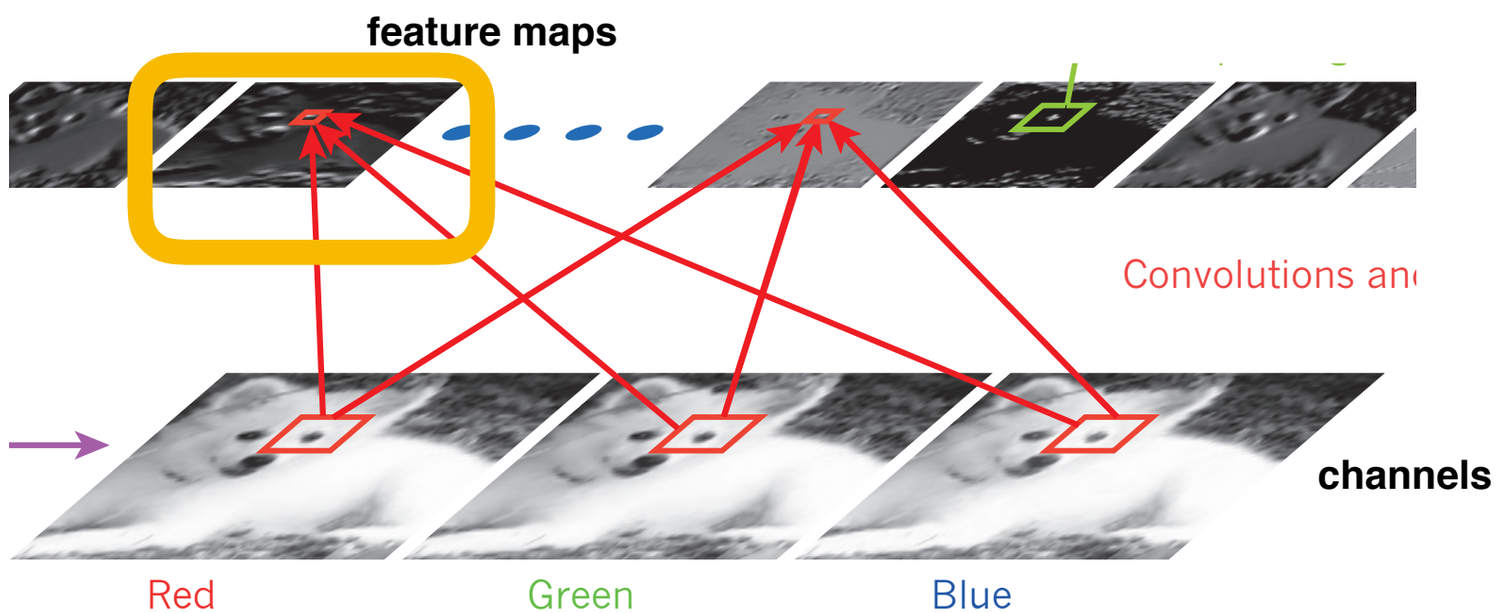
8	6	5
7	9	2
-1	7	3

Feature map 0

`o[:,:,1]`

2	3	7
-2	1	-2
-5	-1	-2

Feature map 1



Filter 0 **Filter 1** **Next layer of network**

Channel 0
(e.g., red)

Channel 1
(e.g., green)

Channel 2
(e.g., blue)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	2	0	0	2	2	0
0	2	2	1	2	1	0
0	1	0	2	1	0	0
0	2	0	2	0	1	0
0	0	0	2	0	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	2	0	1	0
0	0	0	0	0	2	0
0	0	0	2	1	0	0
0	2	1	1	1	2	0
0	1	1	2	2	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	1	0	0	0
0	1	1	2	2	0	0
0	2	2	1	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

0	1	0
-1	1	0
0	1	1

w0[:,:,1]

-1	-1	1
0	1	0
0	1	0

w0[:,:,2]

1	-1	-1
0	0	0
0	0	0

Filter W1 (3x3x3)

w1[:,:,0]

0	-1	-1
0	0	1
1	1	-1

w1[:,:,1]

0	-1	-1
-1	1	1
1	1	-1

w1[:,:,2]

0	-1	0
-1	1	0
-1	-1	0

Output Volume (3x3x2)

o[:,:,0]

8	6	5
7	9	2
-1	7	3

Feature map 0

o[:,:,1]

2	3	7
-2	1	-2
-5	-1	-2

Feature map 1

Bias b0 (1x1x1)

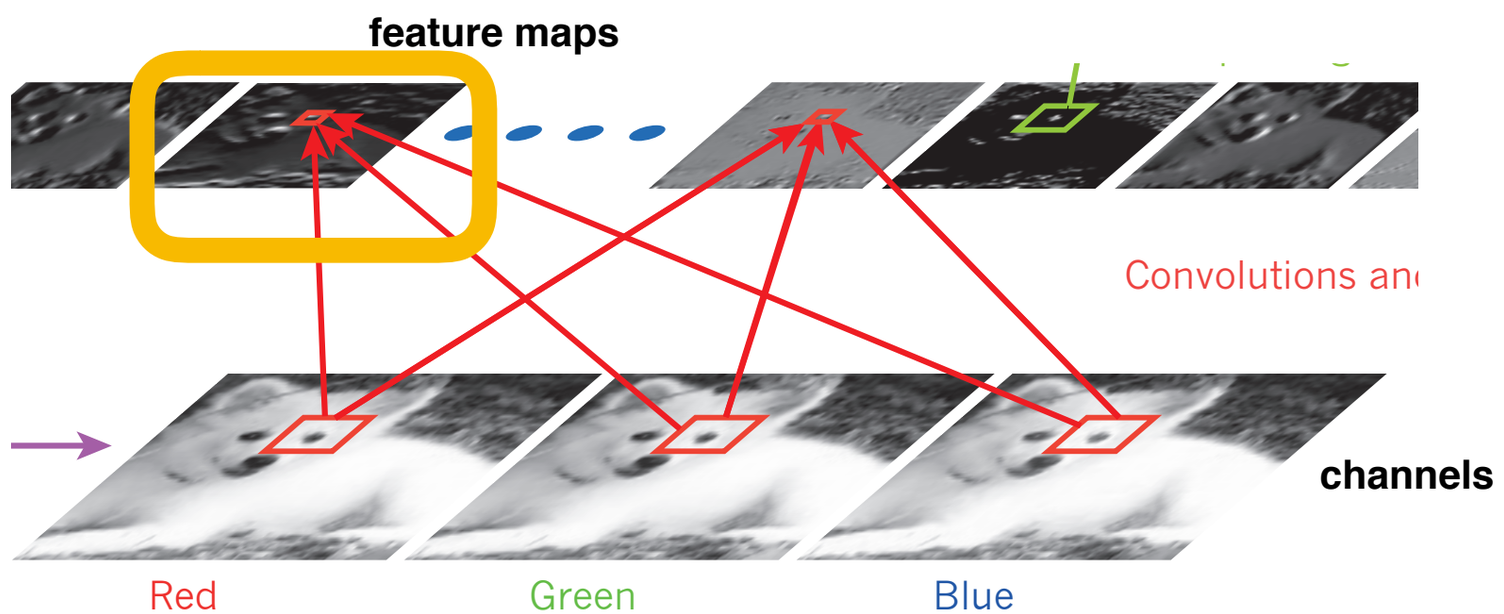
b0[:,:,0]

1

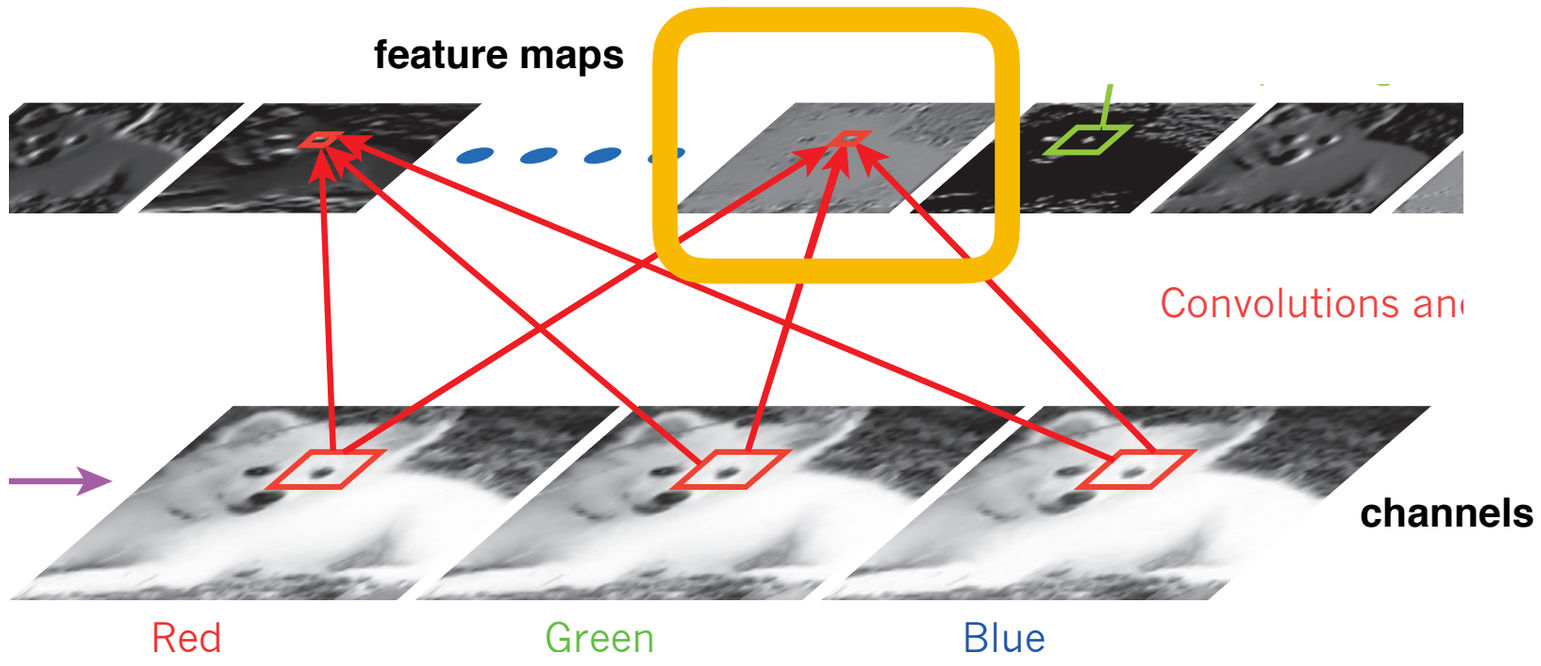
Bias b1 (1x1x1)

b1[:,:,0]

0



Let’s skip to compute the next feature map…



Channel 0

(e.g., red)

Channel 1

(e.g., green)

Channel 2

(e.g., blue)

Filter 0

Filter 1

Next layer of network

Input Volume (+pad 1) (7x7x3)

`x[:,:,0]`

0	0	0	0	0	0	0
0	2	0	0	2	2	0
0	2	2	1	2	1	0
0	1	0	2	1	0	0
0	2	0	2	0	1	0
0	0	0	2	0	2	0
0	0	0	0	0	0	0

`x[:,:,1]`

0	0	0	0	0	0	0
0	1	0	2	0	1	0
0	0	0	0	0	2	0
0	0	0	2	1	0	0
0	2	1	1	1	2	0
0	1	1	2	2	2	0
0	0	0	0	0	0	0

`x[:,:,2]`

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	1	0	0	0
0	1	1	2	2	0	0
0	2	2	1	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

`w0[:,:,0]`

0	1	0
-1	1	0
0	1	1

`w0[:,:,1]`

-1	-1	1
0	1	0
0	1	0

`w0[:,:,2]`

1	-1	-1
0	0	0
0	0	0

Filter W1 (3x3x3)

`w1[:,:,0]`

0	-1	-1
0	0	1
1	1	-1

`w1[:,:,1]`

0	-1	-1
-1	1	1
1	1	-1

`w1[:,:,2]`

0	-1	0
-1	1	0
-1	-1	0

Bias b0 (1x1x1)

`b0[:,:,0]`

1

Bias b1 (1x1x1)

`b1[:,:,0]`

0

Output Volume (3x3x2)

`o[:,:,0]`

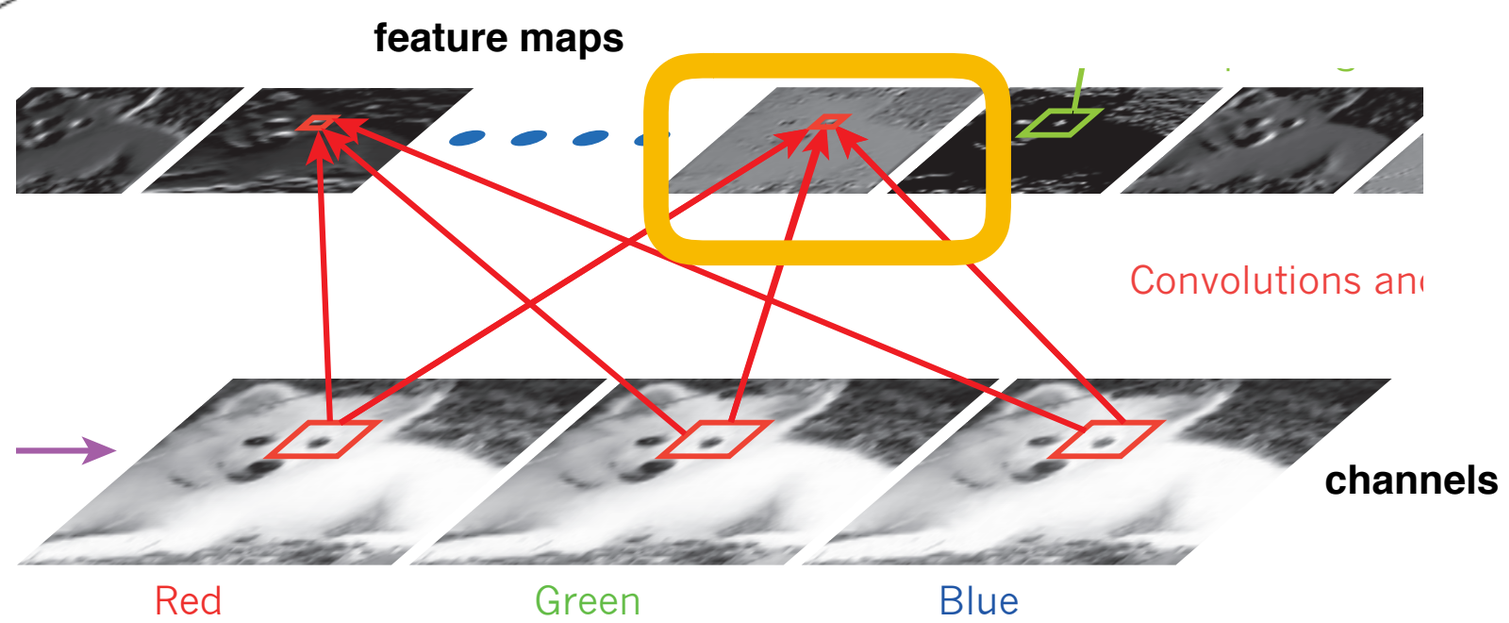
8	6	5
7	9	2
-1	7	3

Feature map 0

`o[:,:,1]`

2	3	7
-2	1	-2
-5	-1	-2

Feature map 1



Filter 0 **Filter 1** **Next layer of network**

Input Volume (+pad 1) (7x7x3)

`x[:,:,0]`

0	0	0	0	0	0	0
0	2	0	0	2	2	0
0	2	2	1	2	1	0
0	1	0	2	1	0	0
0	2	0	2	0	1	0
0	0	0	2	0	2	0
0	0	0	0	0	0	0

Channel 0

(e.g., red)

`x[:,:,1]`

0	0	0	0	0	0	0
0	1	0	2	0	1	0
0	0	0	0	0	2	0
0	0	0	2	1	0	0
0	2	1	1	1	2	0
0	1	1	2	2	2	0
0	0	0	0	0	0	0

Channel 1

(e.g., green)

`x[:,:,2]`

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	1	1	0	0	0
0	1	1	2	2	0	0
0	2	2	1	0	0	0
0	0	0	1	1	2	0
0	0	0	0	0	0	0

Channel 2

(e.g., blue)

Filter W0 (3x3x3)

`w0[:,:,0]`

0	1	0
-1	1	0
0	1	1

`w0[:,:,1]`

-1	-1	1
0	1	0
0	1	0

`w0[:,:,2]`

1	-1	-1
0	0	0
0	0	0

Bias b0 (1x1x1)

`b0[:,:,0]`

1

Filter W1 (3x3x3)

`w1[:,:,0]`

0	-1	-1
0	0	1
1	1	-1

`w1[:,:,1]`

0	-1	-1
-1	1	1
1	1	-1

`w1[:,:,2]`

0	-1	0
-1	1	0
-1	-1	0

Bias b1 (1x1x1)

`b1[:,:,0]`

0

Output Volume (3x3x2)

`o[:,:,0]`

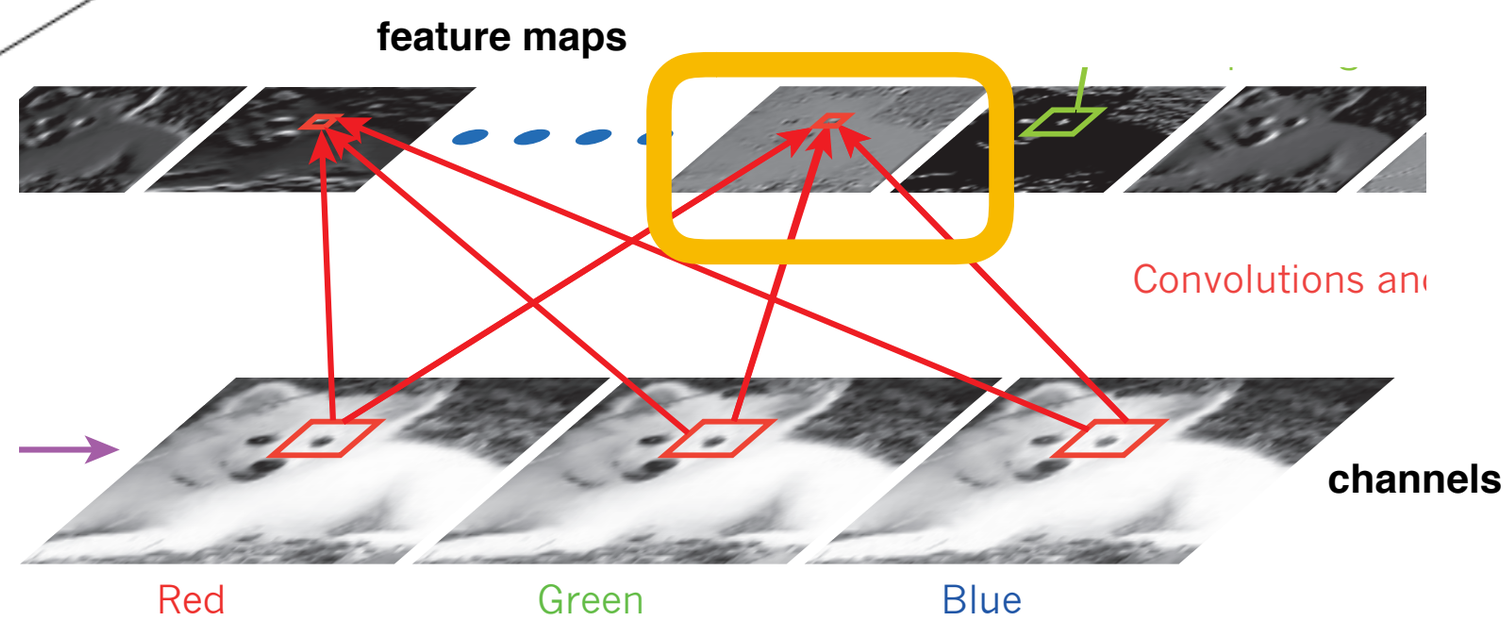
8	6	5
7	9	2
-1	7	3

Feature map 0

`o[:,:,1]`

2	3	7
-2	1	-2
-5	-1	-2

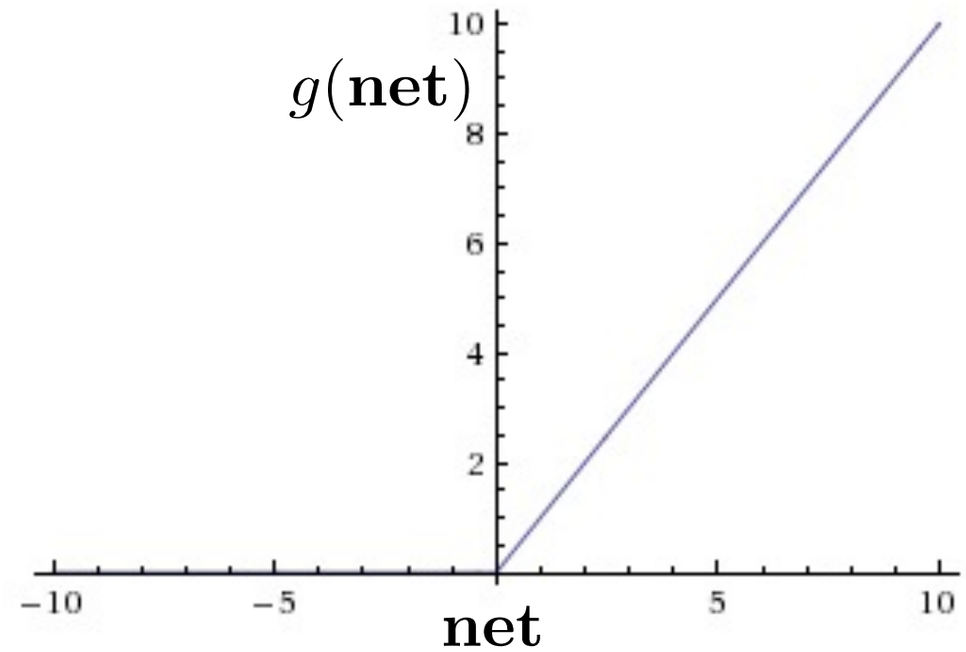
Feature map 1



After convolutions, apply a non-linear activation function (e.g., ReLUs)

$$g(\mathbf{net}) = \begin{cases} \mathbf{net} & \mathbf{net} \geq 0 \\ 0 & \mathbf{net} < 0 \end{cases}$$

Rectified linear unit (ReLU)



Feature map 0

8	6	5
7	9	2
-1	7	3

Feature map 1

2	3	7
-2	1	-2
-5	-1	-2

$g(\mathbf{net})$ →

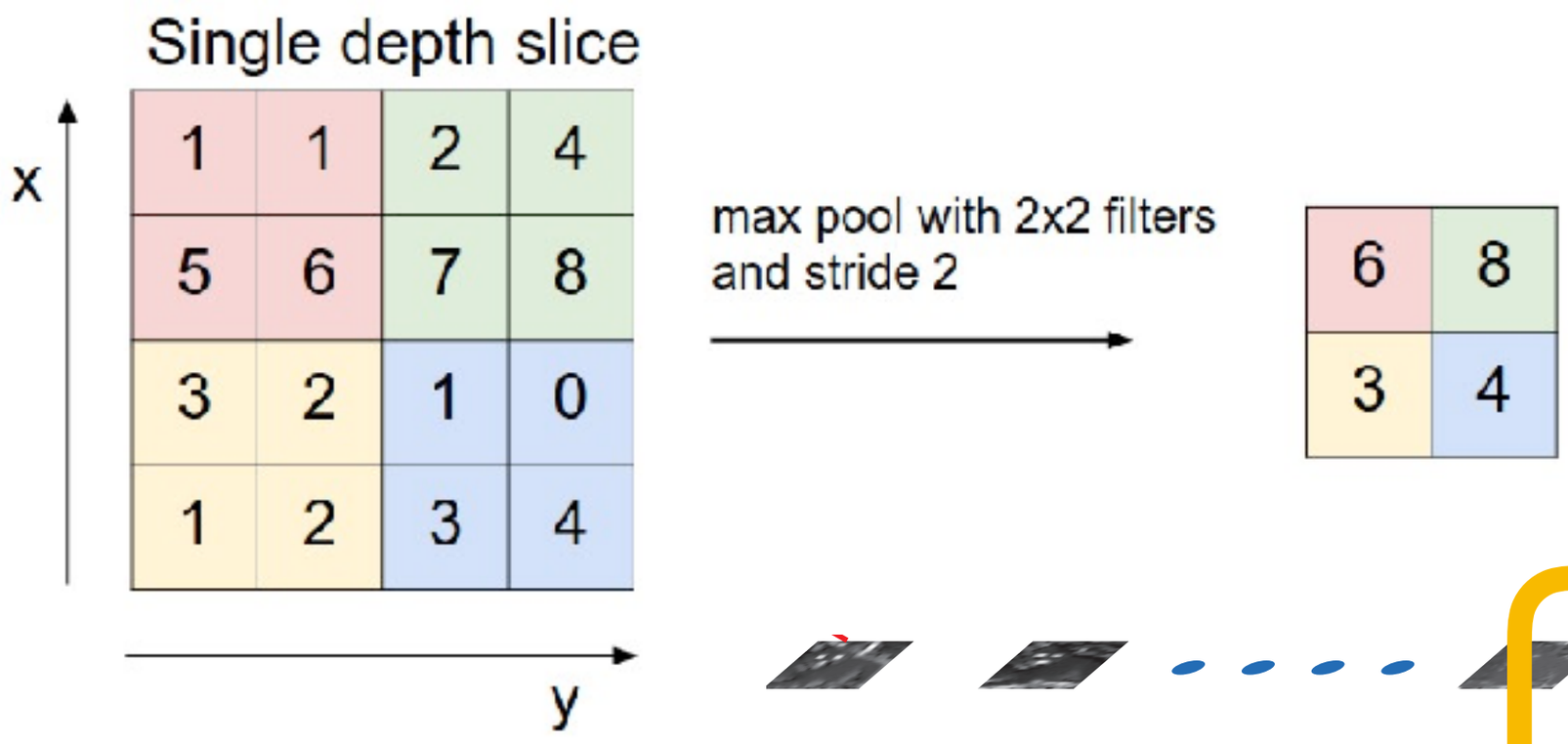
8	6	5
7	9	2
0	7	3

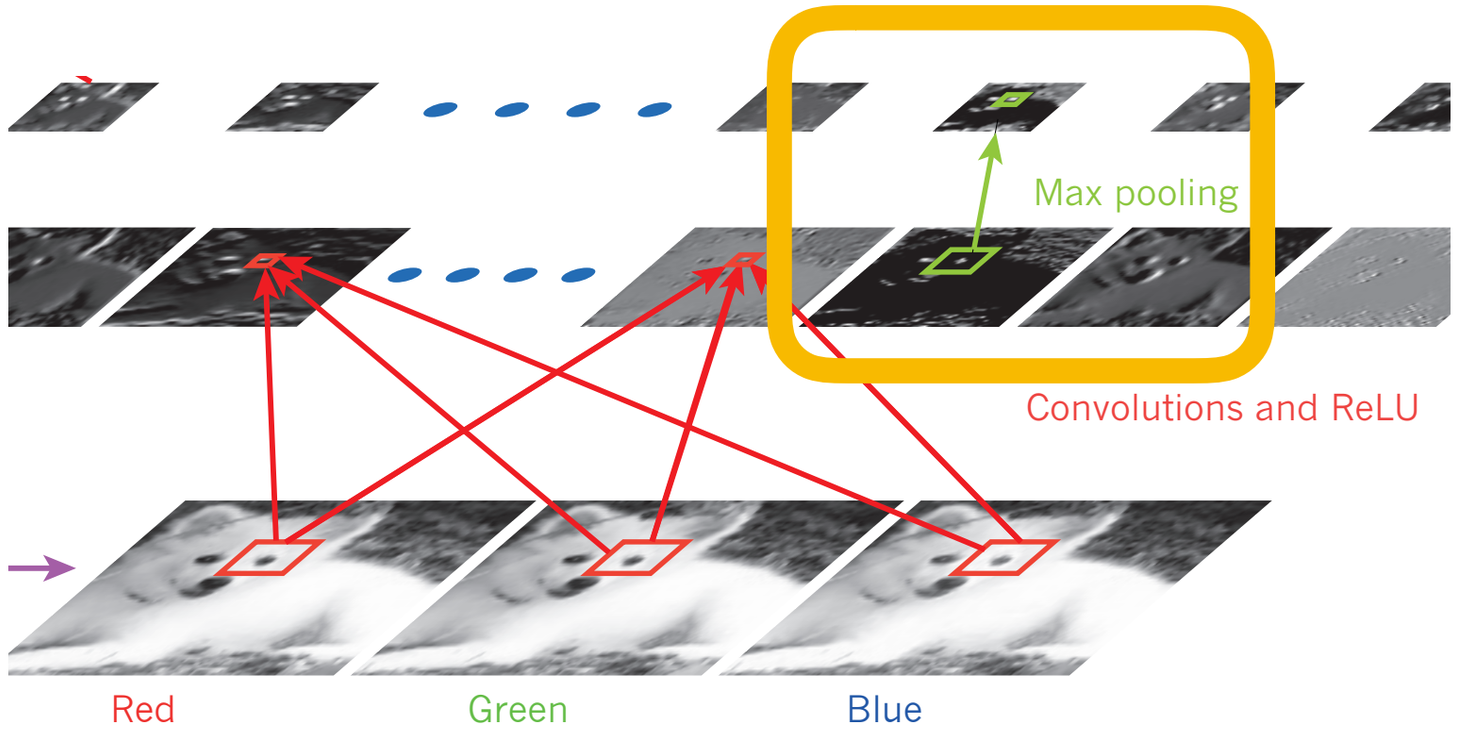
2	3	7
0	1	0
0	0	0

Next step: Pooling

- Downsamples a feature map to a coarser resolution
- Provides additional translation invariance

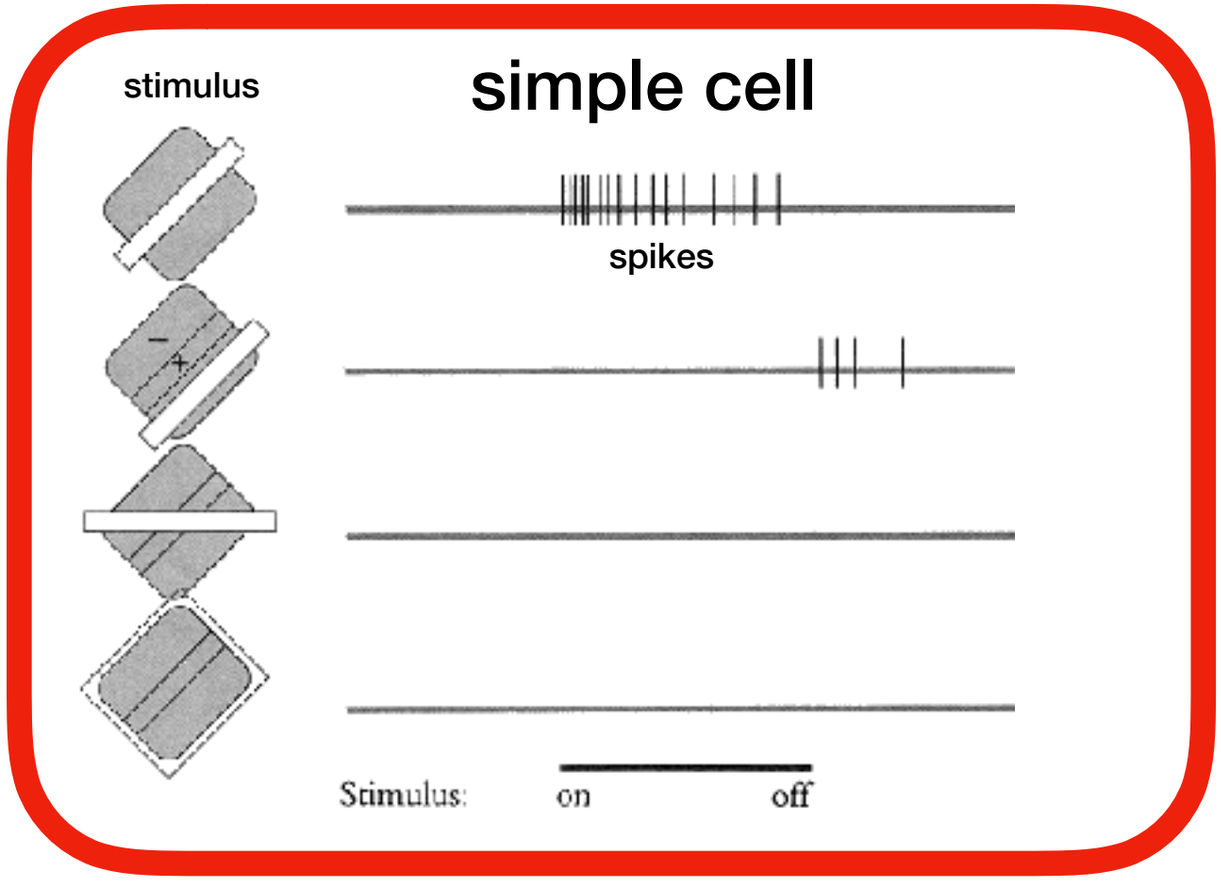
Max pooling



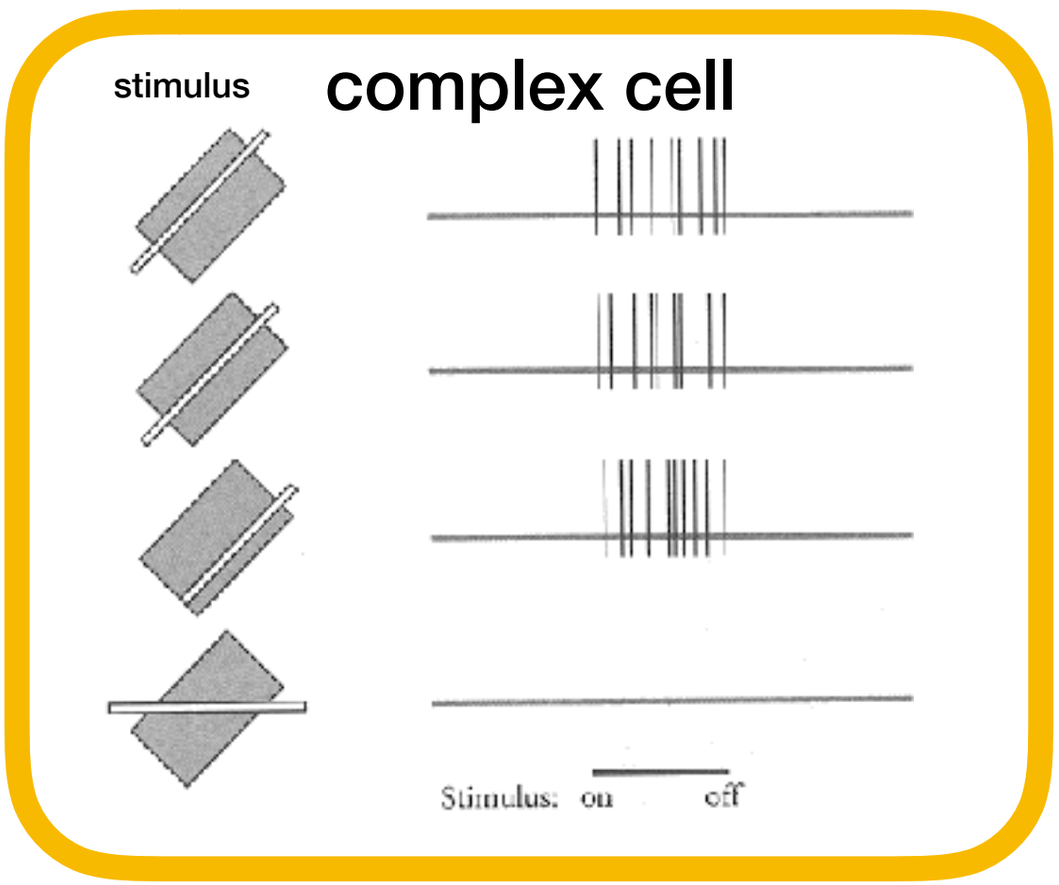


Connection to biological architecture of primary visual cortex

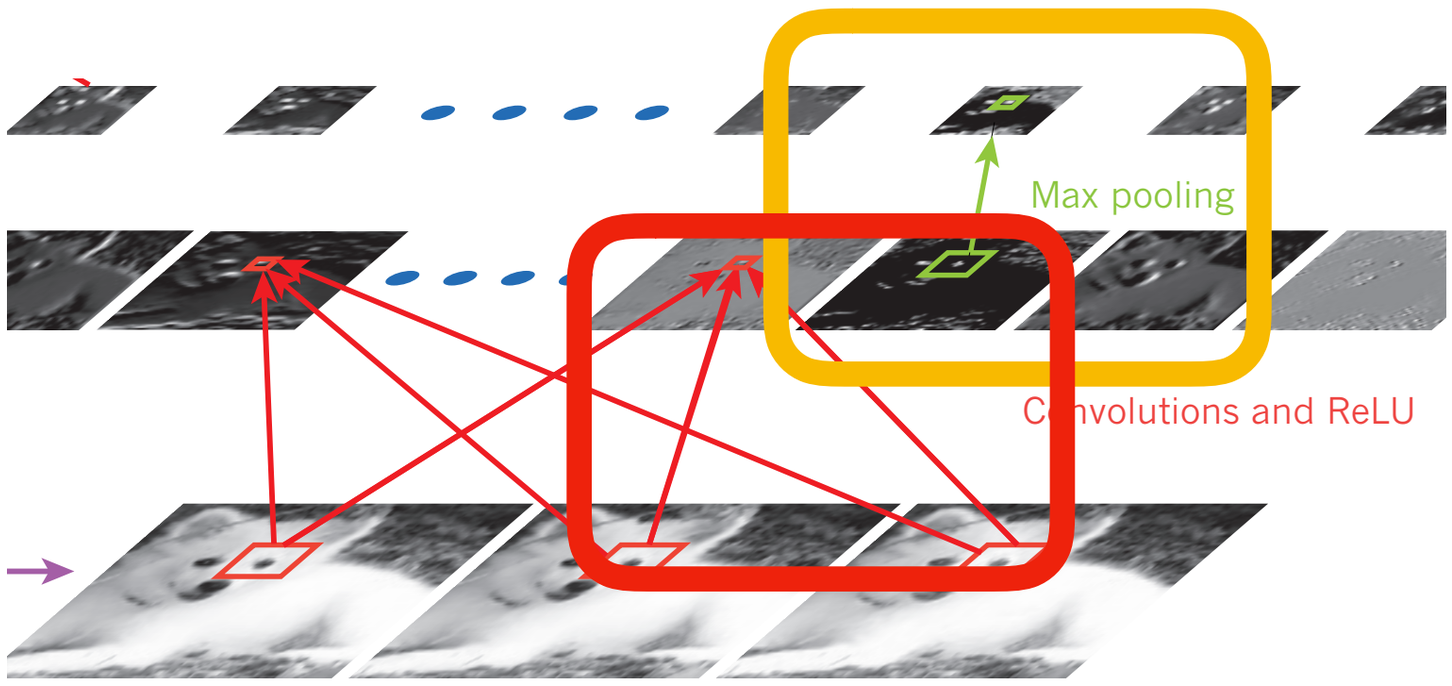
(Hubel & Wiesel)



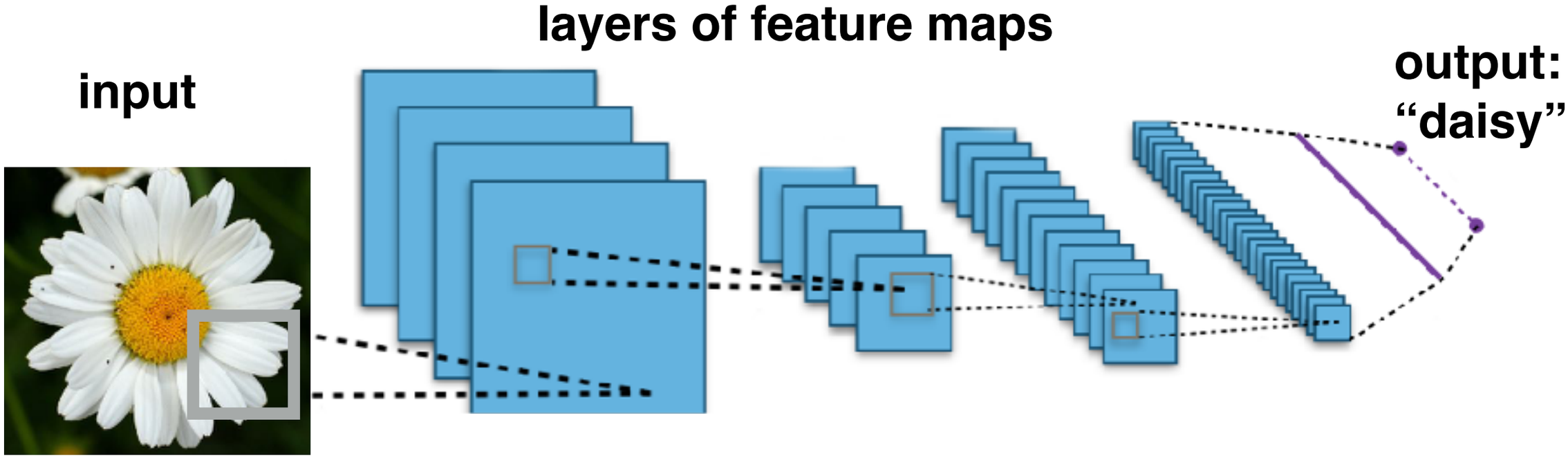
localized edge detector



invariance through pooling

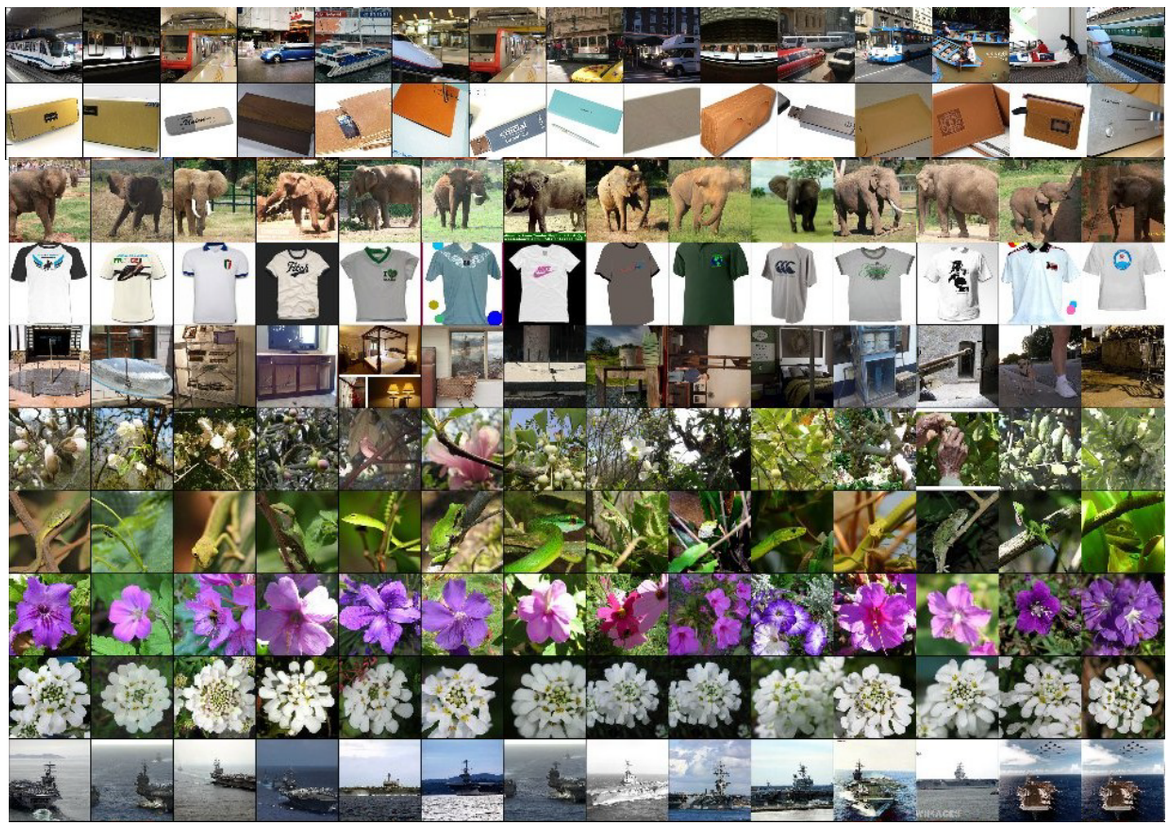


How do we train it? Backpropagation



Training data (ImageNet)

- 1.2 million images
- 1000 categories
- ~1200 images per category



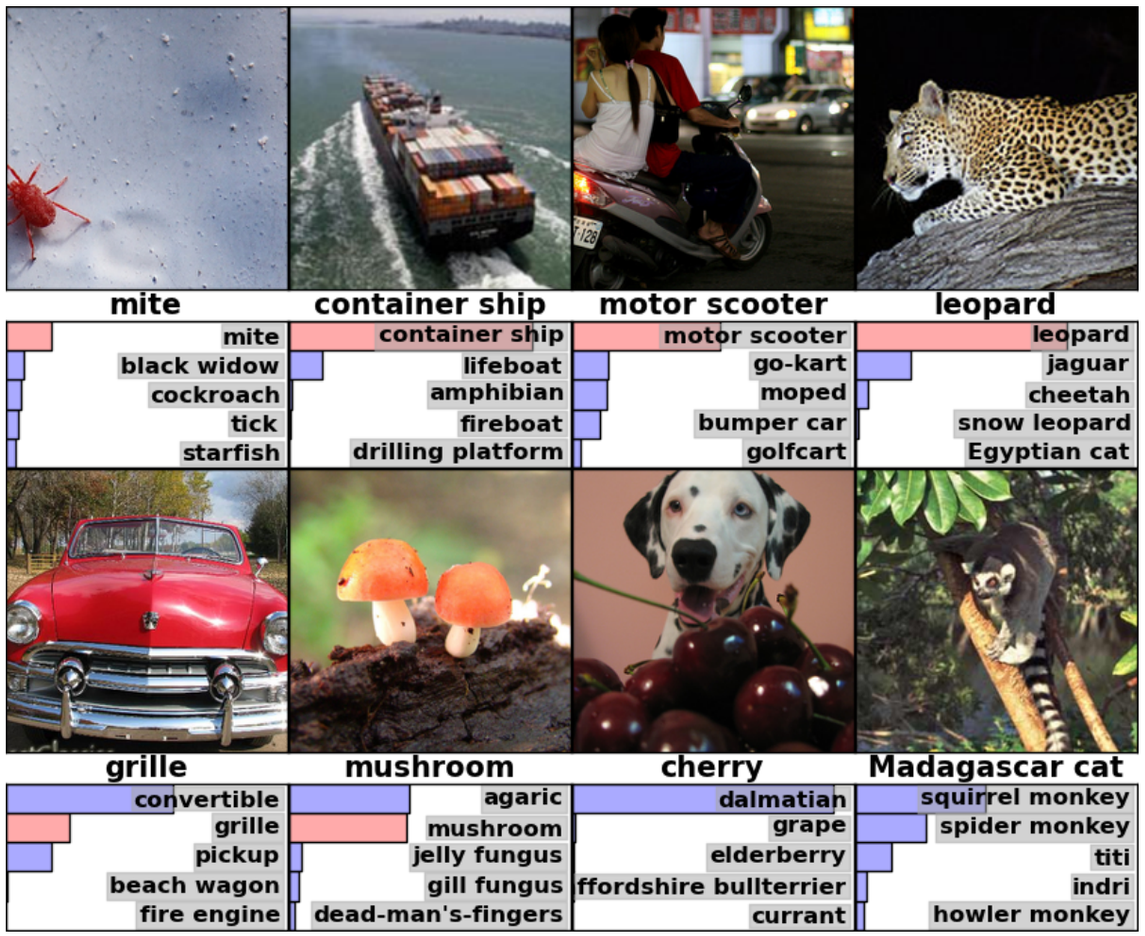
AlexNet
8 layers
~60 million parameters

GoogLeNet
22 layers
~5 million parameters

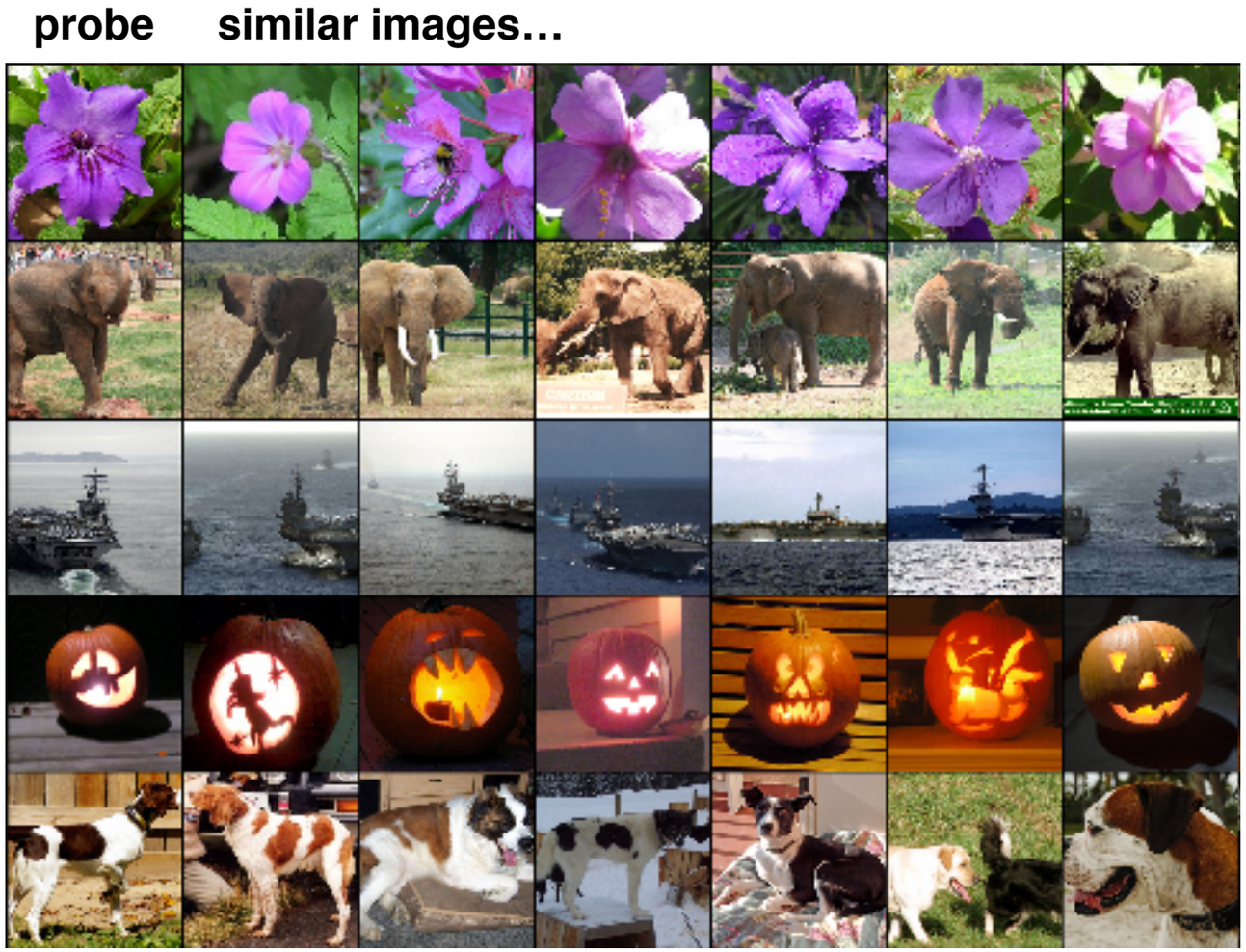
Residual network
152 layers

Example results on test images

object recognition



finding similar images



(some filters learned by Krizhevsky et al., 2012)

Deep convolutional neural networks have been applied recently in cognitive and neural modeling

- Peterson, Abbott, and Griffiths (2016) explored convents for **predicting similarity ratings** between images.
- Lake, Zaremba, Fergus, and Gureckis (2015) showed that deep convnets can **predict human typicality ratings** for some classes of natural images
- Yamins et al. (2014) showed that deep convnets can **predict neural response in high-level visual areas**

Predicting human similarity ratings with a neural network

(Peterson, Abbott, and Griffiths, 2016)

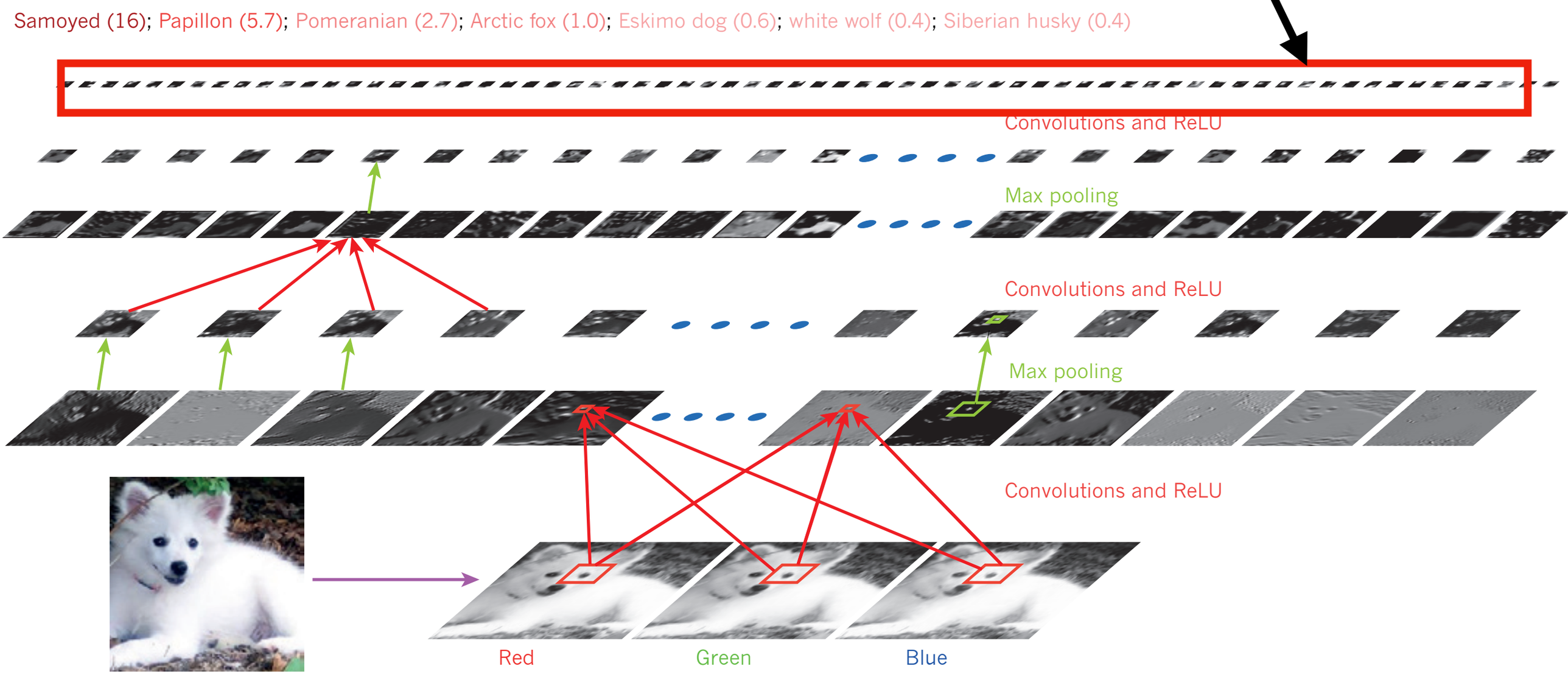
Some images look more similar to us than others. Can a neural network trained for classification help to explain similarity human judgments?

example animal images (they collected 120 x 120 pairwise ratings)

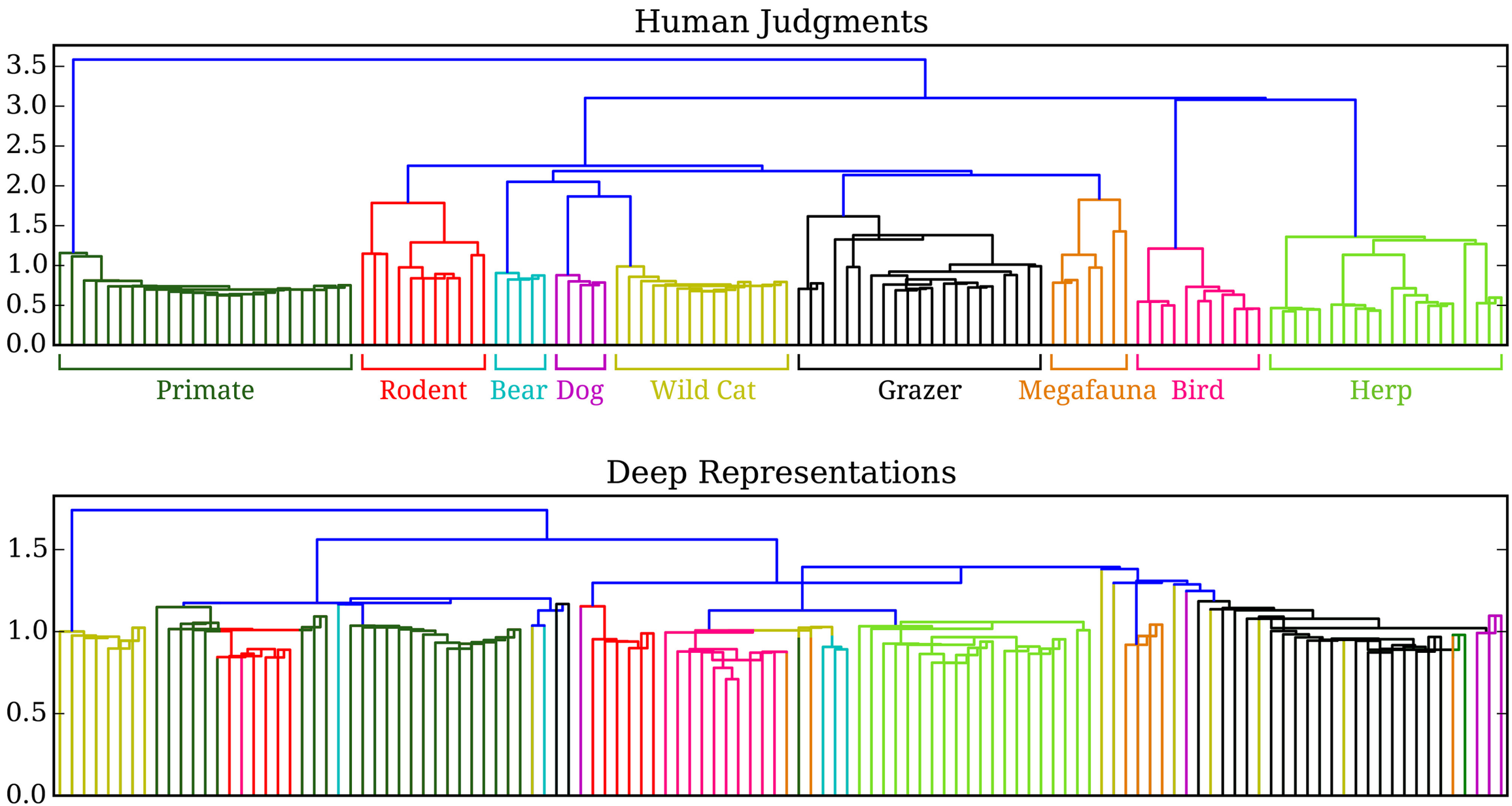
computing image-to-image similarity

$$sim(i,j) = \sum_k f_{ik} f_{jk}$$ similarity computed as dot product

summarizing images as high-level feature vector f

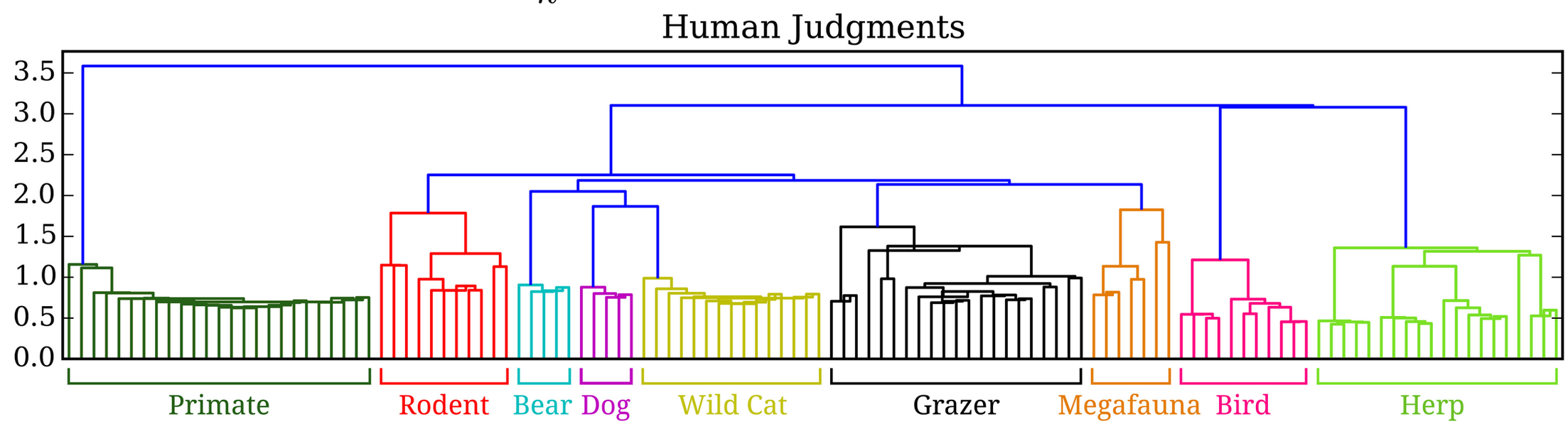


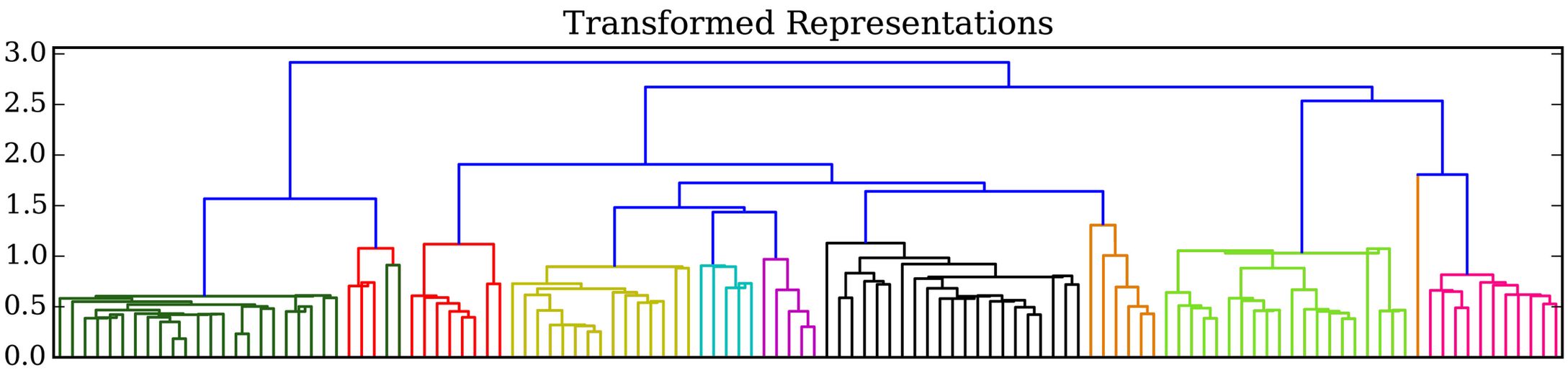
Hierarchical clustering reveals substantial differences in representation (best network explains about 40% of the variance in human judgments)



When fitting weights that allow the network features to be re-scaled, the network fits much better (best network explains about 84% of the variance in human judgments using out-of-sample predictions)

$$sim(i,j) = \sum_k w_k f_{ik} f_{jk} \qquad w_k : \text{weight for feature k}$$



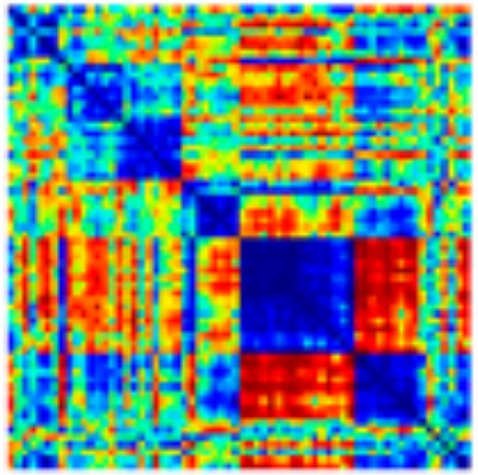


Predicting neural recordings with a deep convnet

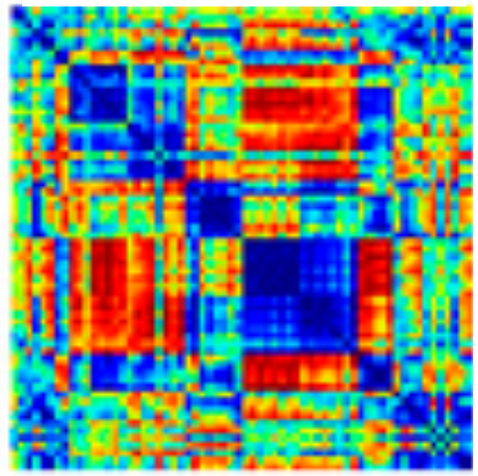
(Yamins et al., 2014)

similarity matrices for images

IT neuronal units



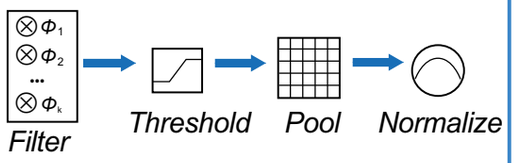
deep convnet



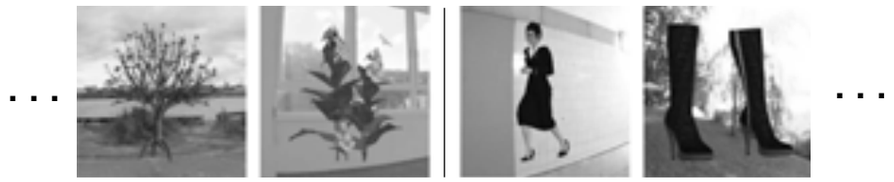
Animals (8)
Boats (8)
Cars (8)
Chairs (8)
Faces (8)
Fruits (8)
Planes (8)
Tables (8)

Image generalization

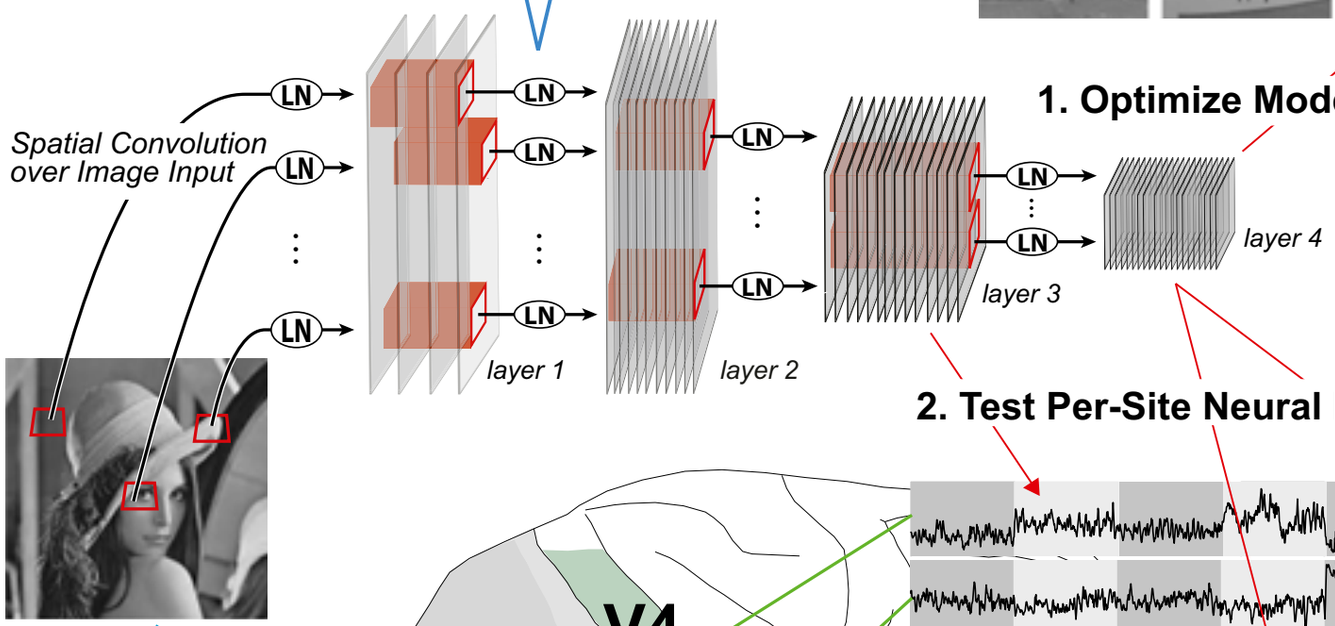
Operations in Linear-Nonlinear Layer



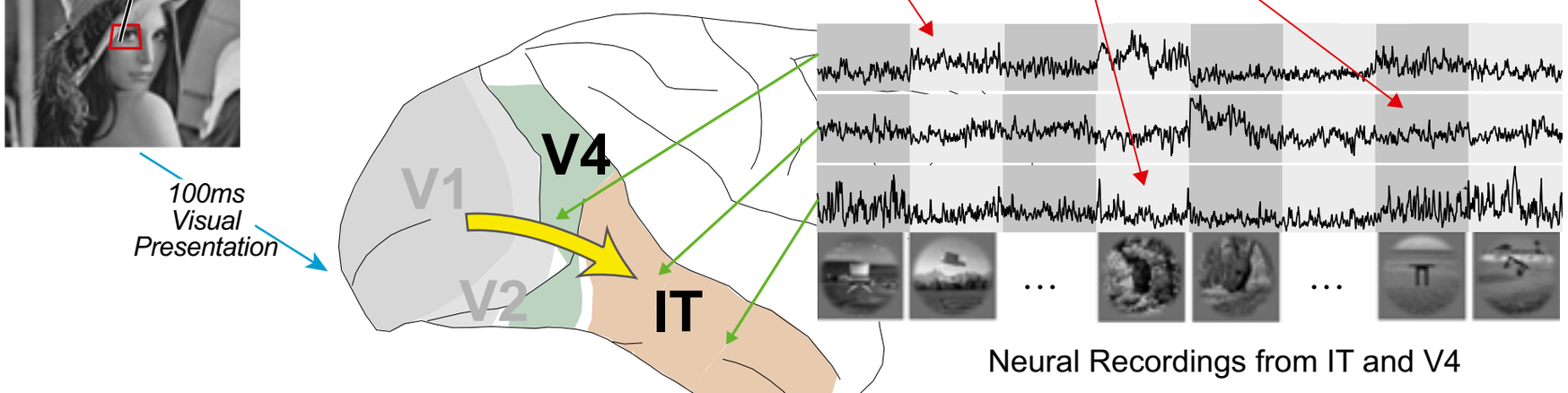
Behavioral Tasks
e.g. Trees vs non-Trees



1. Optimize Model for Task Performance



2. Test Per-Site Neural Predictions



Neural Recordings from IT and V4

Understanding category typicality with deep convnets

(Lake, Zaremba, Fergus, & Gureckis, 2015)

typical dog

atypical dog

• For people, typicality influences performance in practically any category-related task (Murphy, 2002)

- speed of categorization
- ease of production
- ease of learning
- usefulness for inductive inference

• No known model successfully predicts typicality ratings from raw images -- How do convnets perform?

Category: Banana (ρ=0.82)

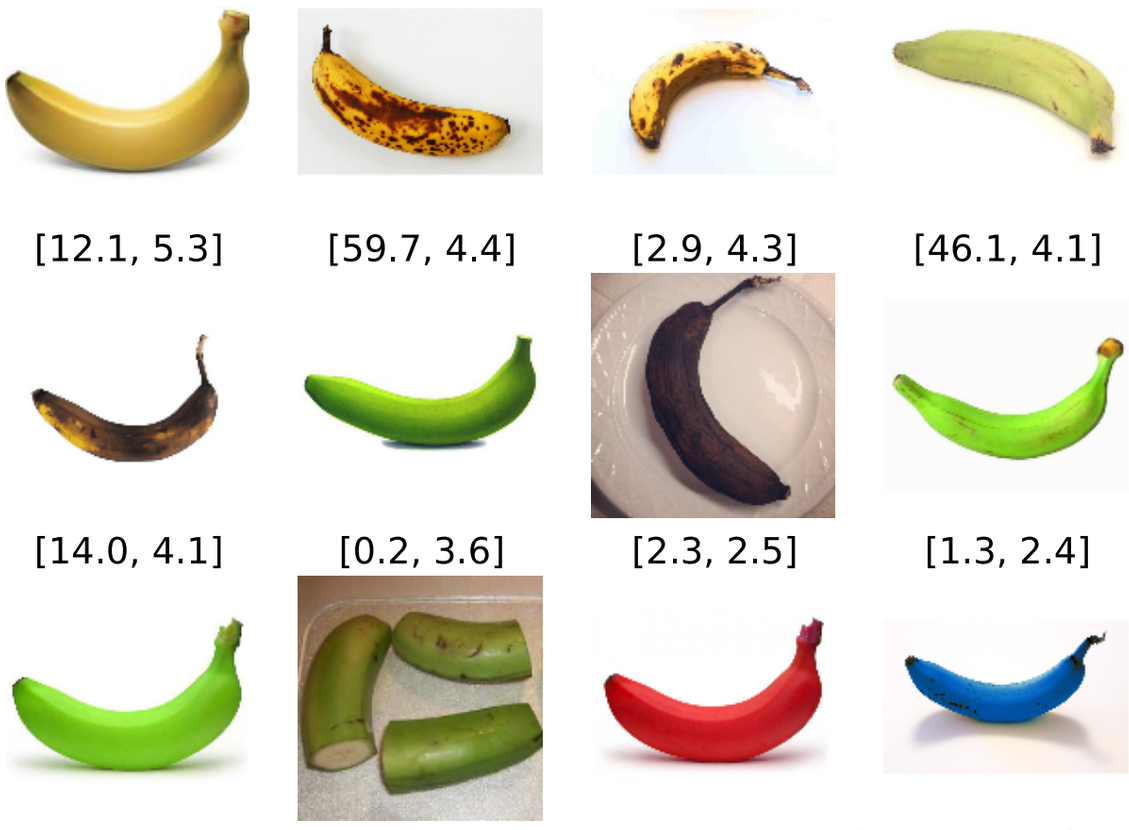
How well does this picture fit your idea or image of the category? (rated on 1-7 scale)

Human typicality ratings

Most typical →

[97.8, 6.8] [98.0, 6.8] [96.6, 6.8] [99.7, 6.6]

[96.9, 6.6] [99.3, 6.0] [78.6, 5.8] [99.5, 5.5]



[12.1, 5.3] [59.7, 4.4] [2.9, 4.3] [46.1, 4.1]

[14.0, 4.1] [0.2, 3.6] [2.3, 2.5] [1.3, 2.4]

Least typical

rating key: [machine (0-100), human (1-7)]

Category: Bathtub (ρ=0.68)

Human typicality ratings

Most typical →

Least typical

Convnet typicality ratings

rating key: [machine (0-100), human (1-7)]

Category: Envelope (ρ=0.79)

Human typicality ratings

Most typical →

[91.5, 6.7] [75.2, 6.6] [96.4, 6.6] [98.5, 6.6]

[97.7, 6.6] [82.8, 6.2] [69.5, 5.3] [59.7, 5.2]

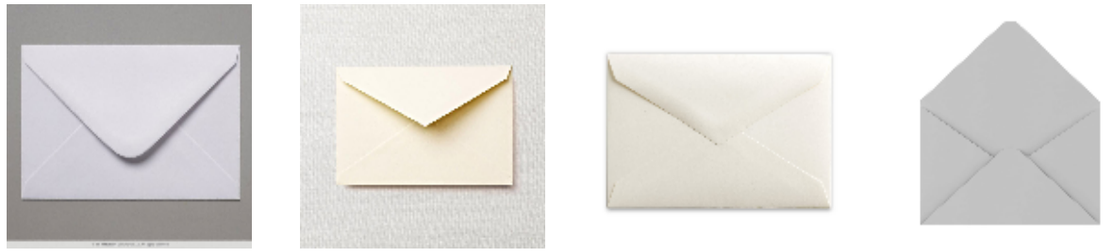
[31.2, 5.1] [32.5, 5.1] [10.8, 4.8] [5.8, 4.2]

[10.4, 4.1] [50.6, 3.8] [41.9, 3.4] [24.9, 3.2]

Least typical

Convnet typicality ratings

[98.5, 6.6] [97.7, 6.6] [96.4, 6.6] [91.5, 6.7]



[82.8, 6.2] [75.2, 6.6] [69.5, 5.3] [59.7, 5.2]

[50.6, 3.8] [41.9, 3.4] [32.5, 5.1] [31.2, 5.1]

[24.9, 3.2] [10.8, 4.8] [10.4, 4.1] [5.8, 4.2]

rating key: [machine (0-100), human (1-7)]

Category: Teapots (ρ=0.38)

Human typicality ratings

Most typical →

Least typical

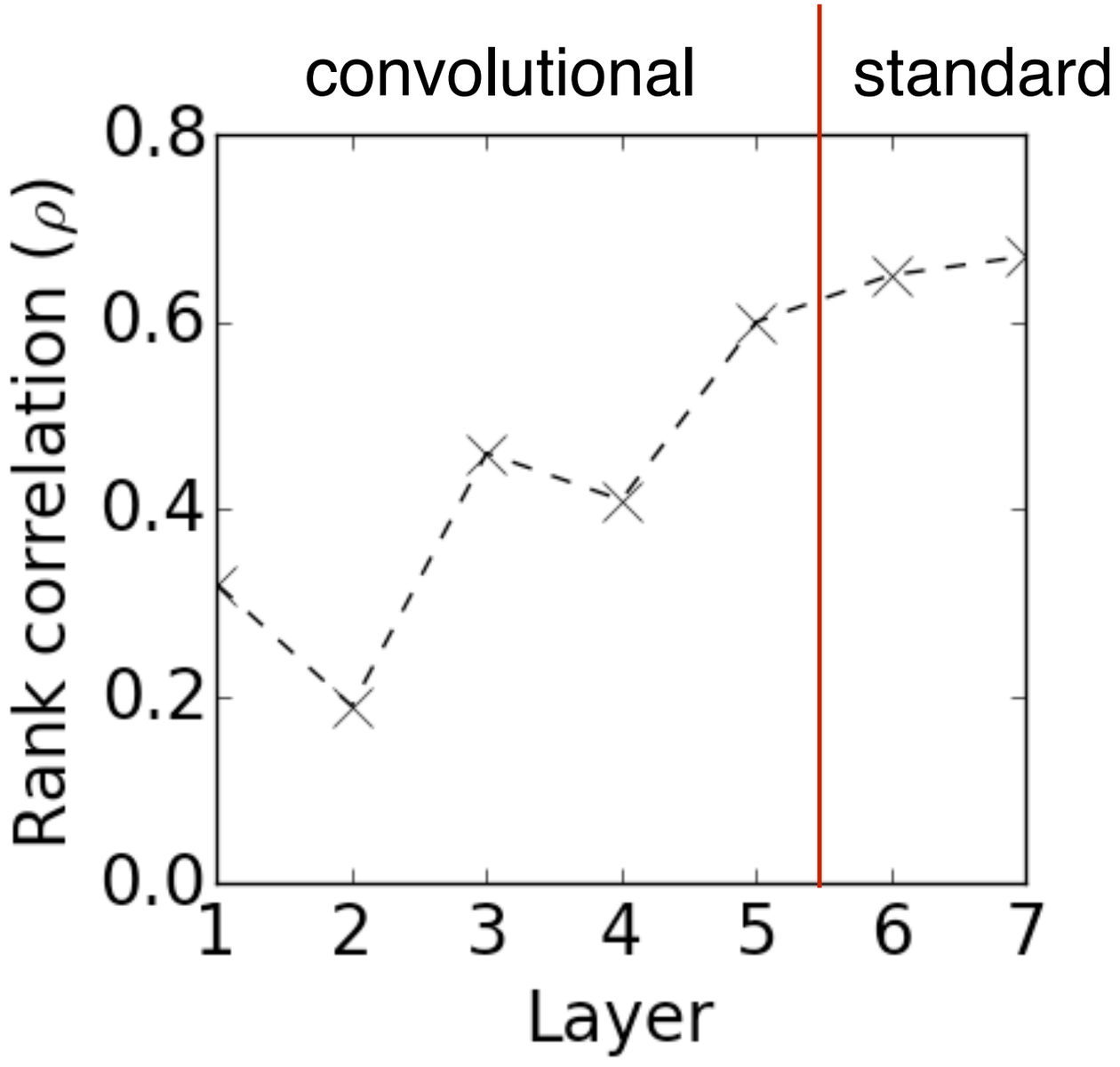
Convnet typicality ratings

rating key: [machine (0-100), human (1-7)]

Summary

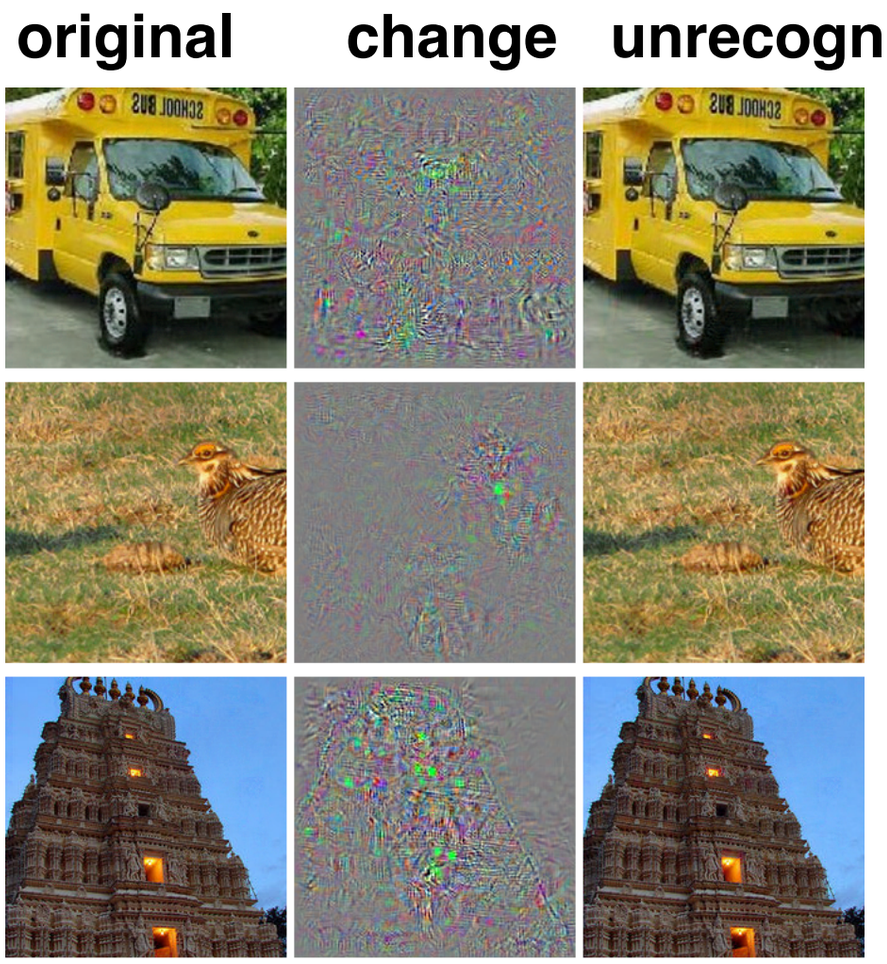
	Rank Correlation
Banana	0.82
Bathtub	0.68
Coffee Mug	0.62
Envelope	0.79
Pillow	0.67
Soap dispenser	0.74
Table lamp	0.69
Teapot	0.38
Average	**0.67**

Prediction as a function of network depth.

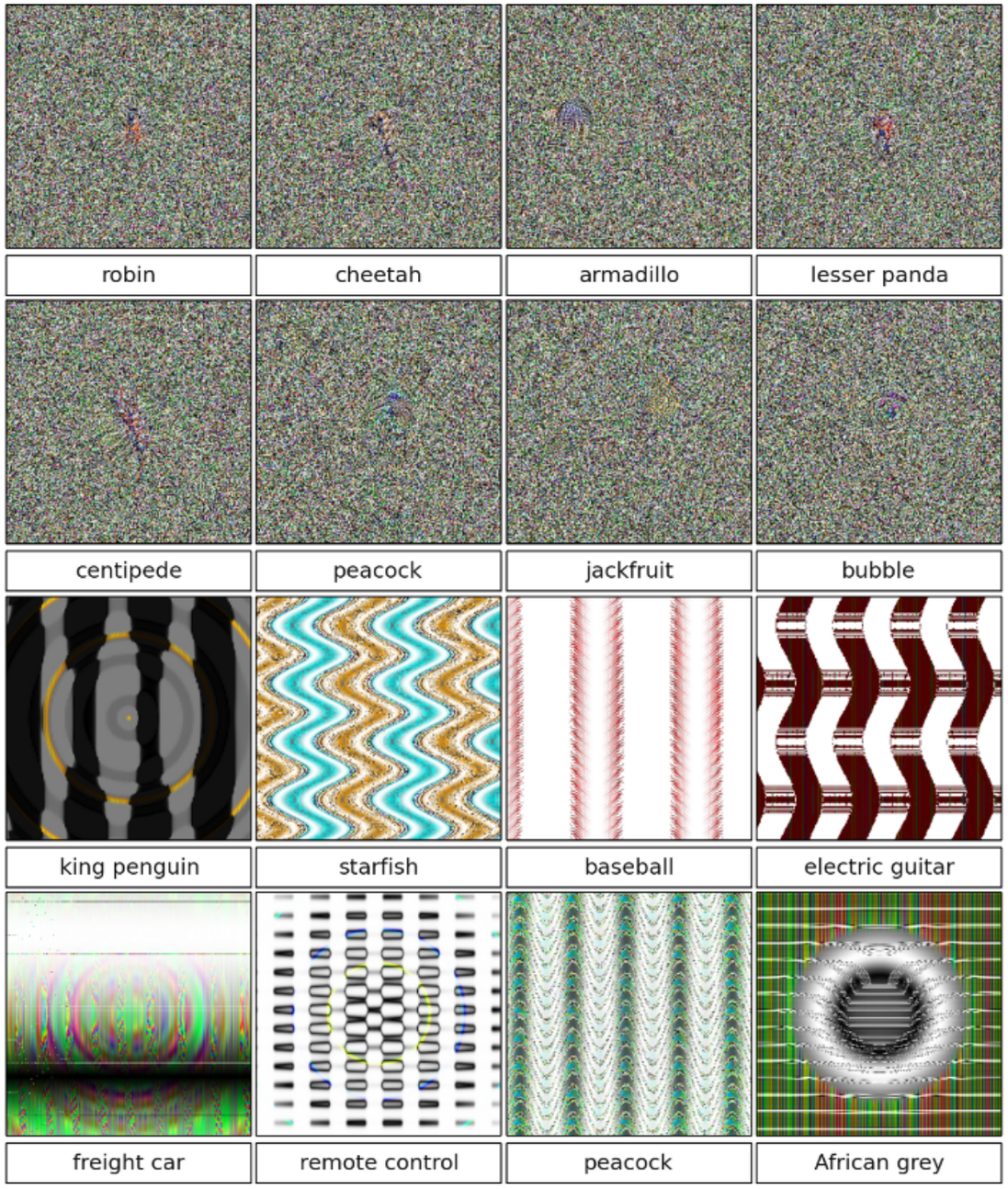


Critiques of deep convolutional networks

It is easy to fool them with adversarial examples generated to fool networks



(Szegedy et al., 2013)



(Nguyen et al., 2015)

Critiques of deep convolutional networks

Compared to deep convnets, people can learn much richer concepts from less data.

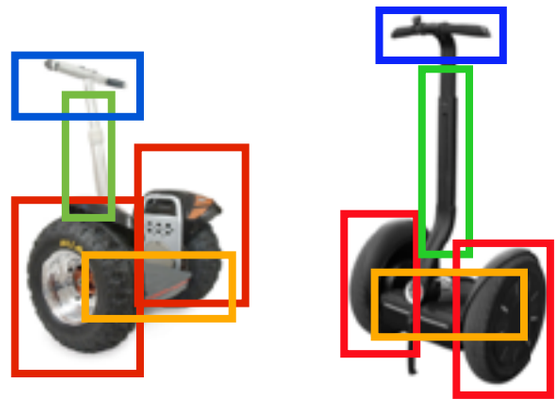
People learn from less data

"one-shot learning"

where are the others?

People learn richer concepts

parsing



generating new concepts

generating new examples