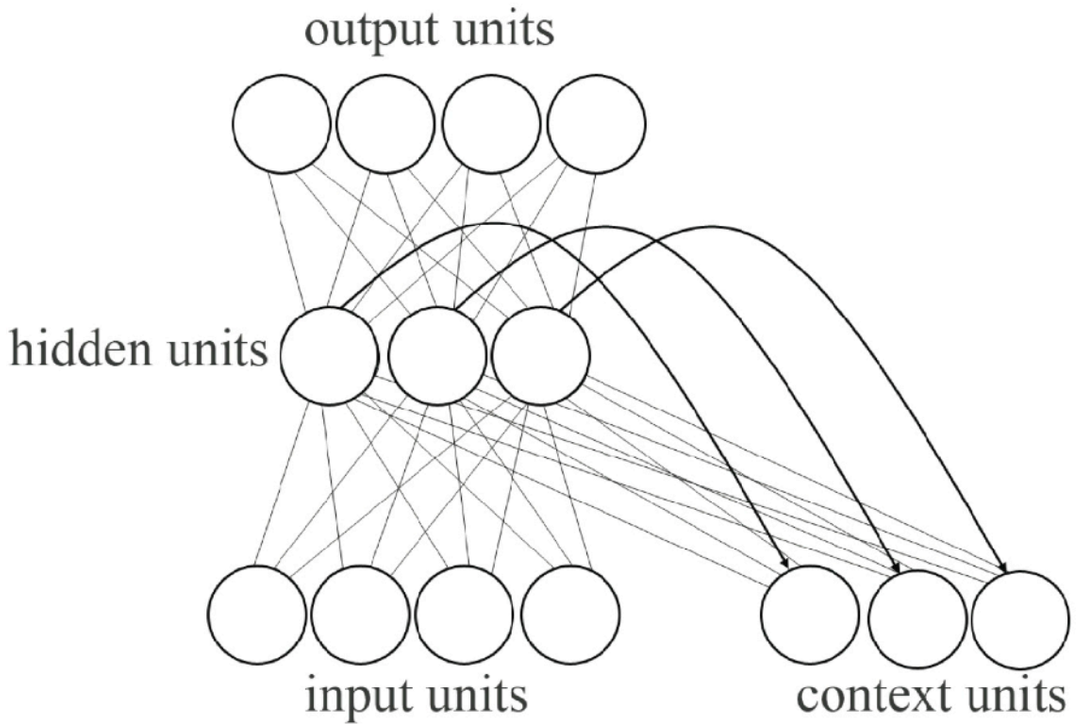
Computational Cognitive Modeling

Neural networks and deep learning (part 2)

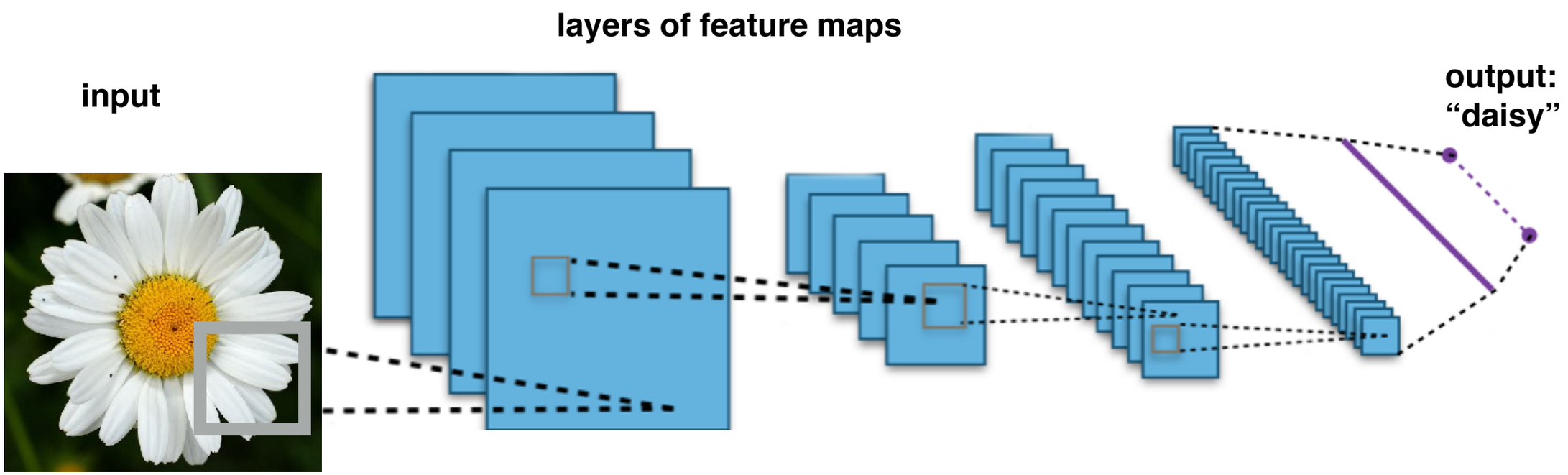
Brenden Lake & Todd Gureckis

Two very important types of neural networks

Recurrent neural networks (RNNs)



Convolutional neural networks (ConvNets)

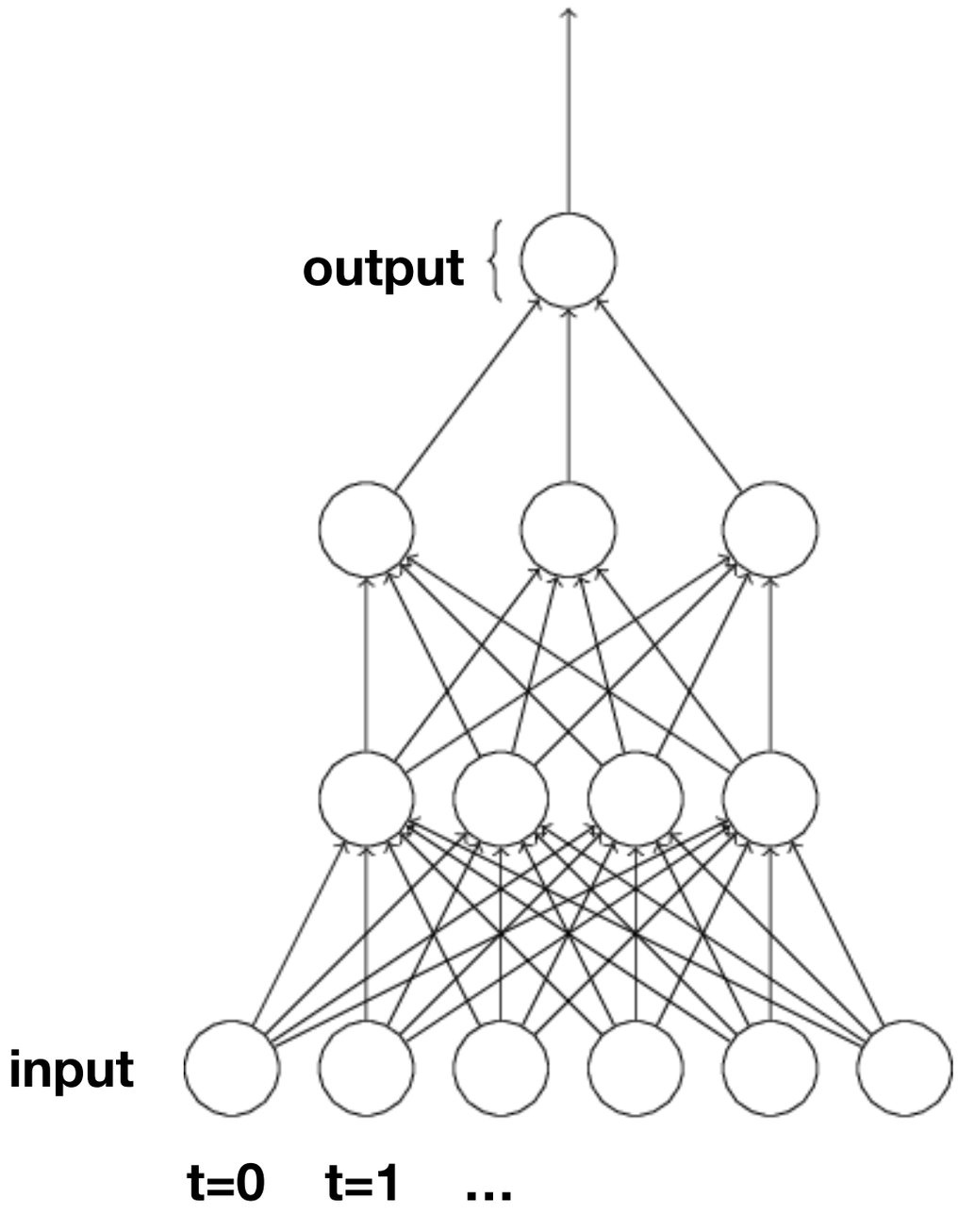


Data often has important temporal structure

- 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

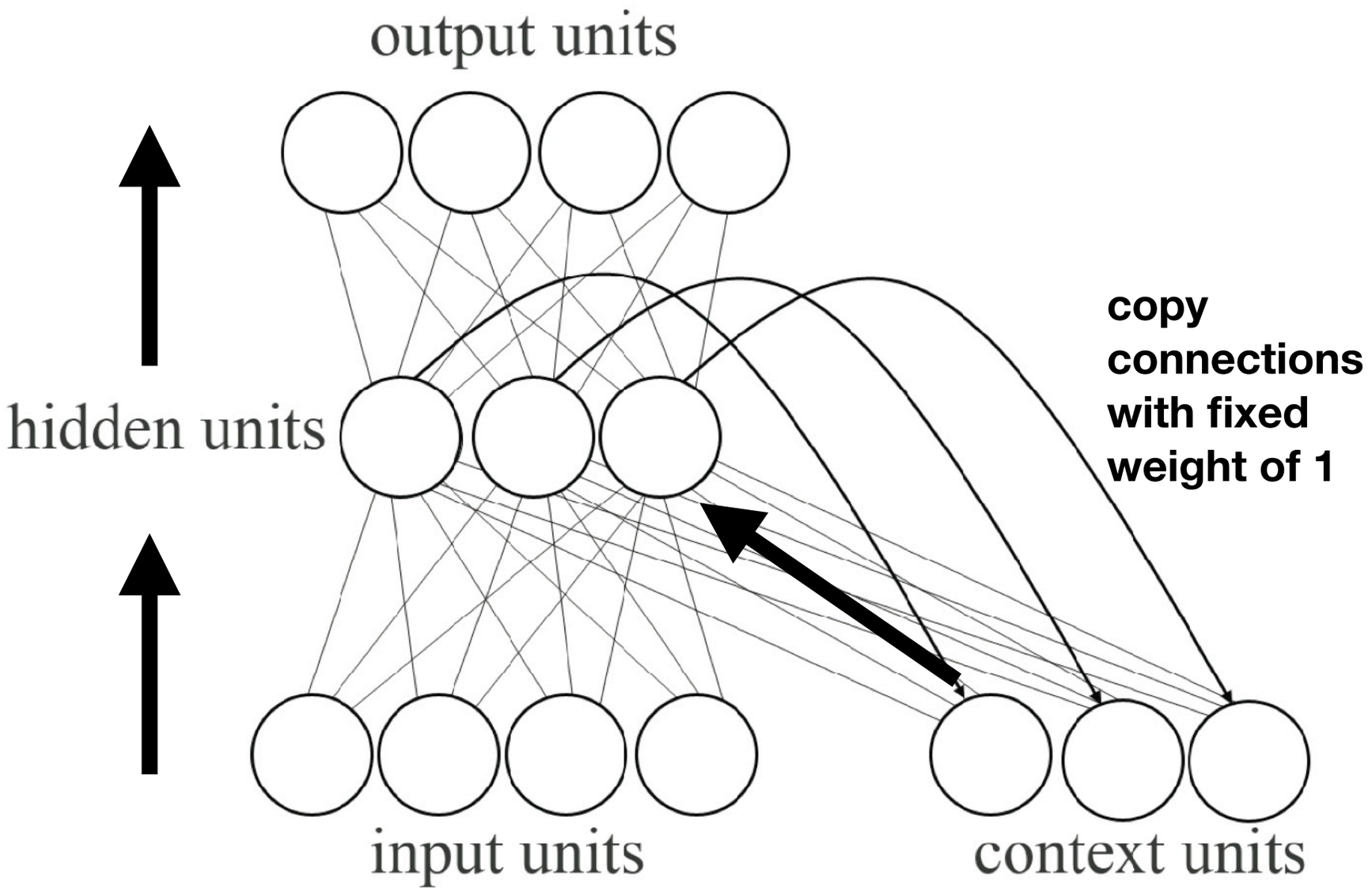
Time underlies many interesting human behaviors. Thus, the question of how to represent time in connectionist models is very important. One approach is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation). The current report develops a proposal 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

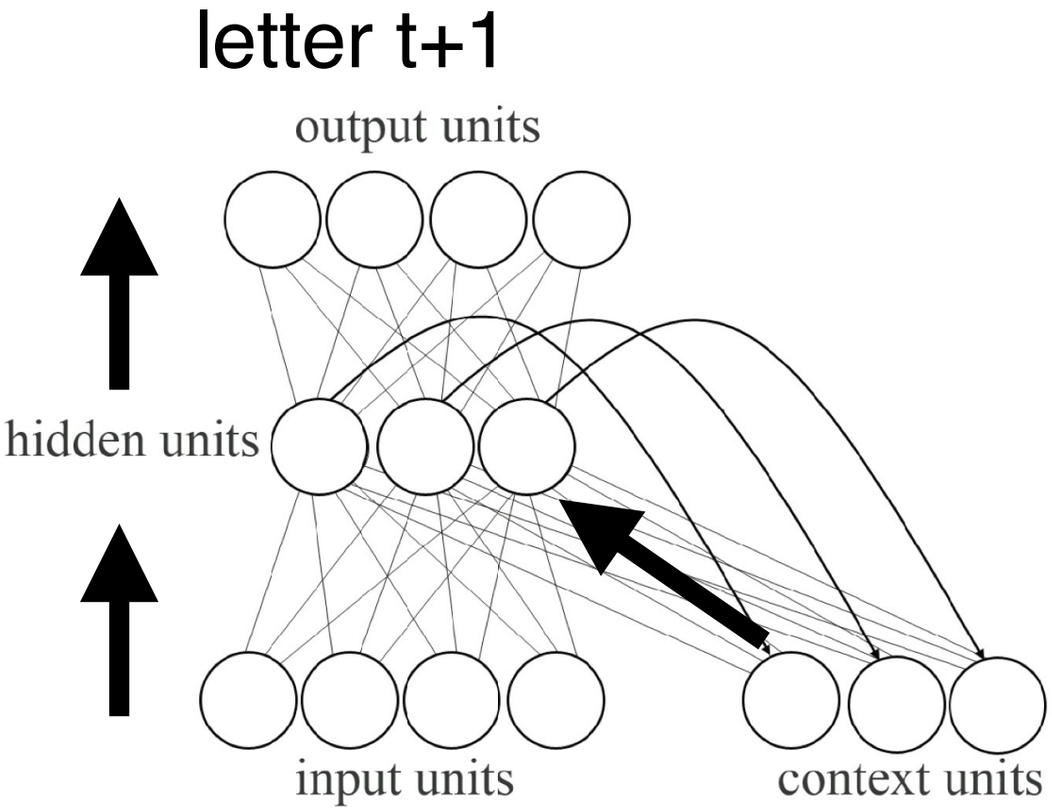
Time is clearly important in cognition. It is inextricably bound up with many behaviors (such as language) which express themselves as temporal sequences. Indeed, it is difficult to know how one might deal with such basic problems as goal-directed behavior, planning, or causation without some way of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

diibaguuubadiidiiguuuguuudiidiibadii….
(random mix of “ba”, “dii”, and “guuu”)

task:

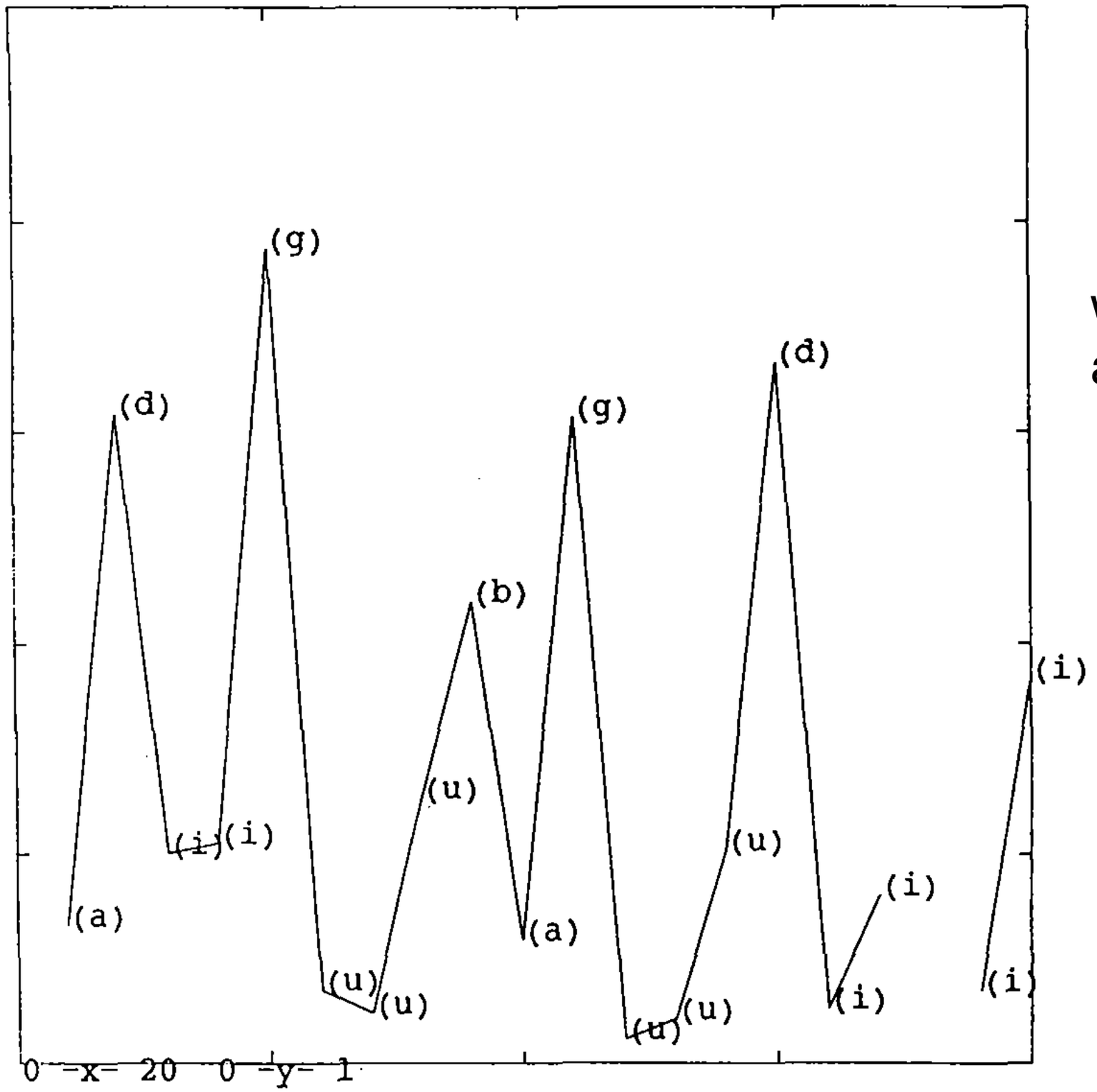
predict the next letter

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)



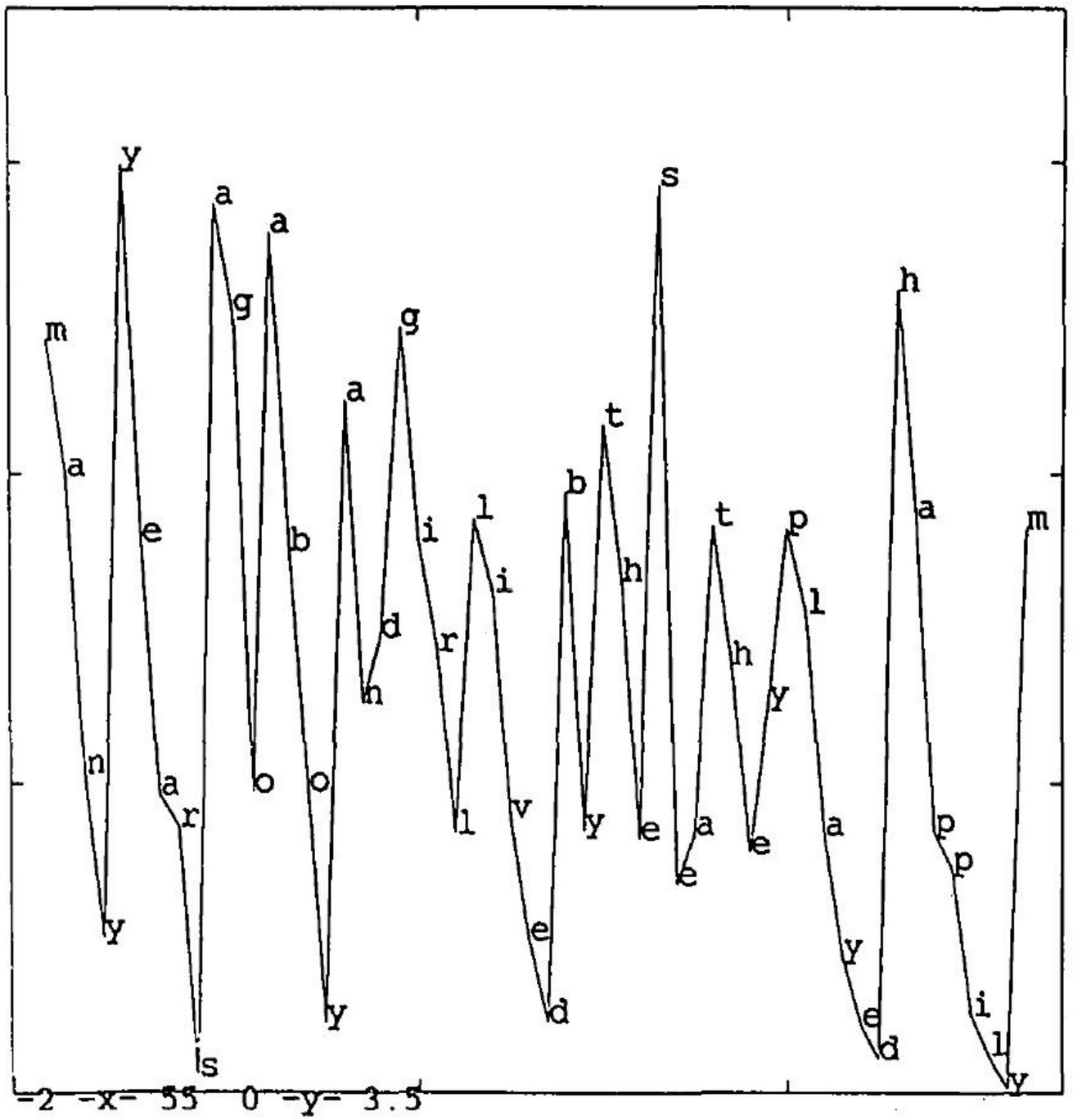
time

words “ba”, “dii”, and “guuu”

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

What is a “word”?

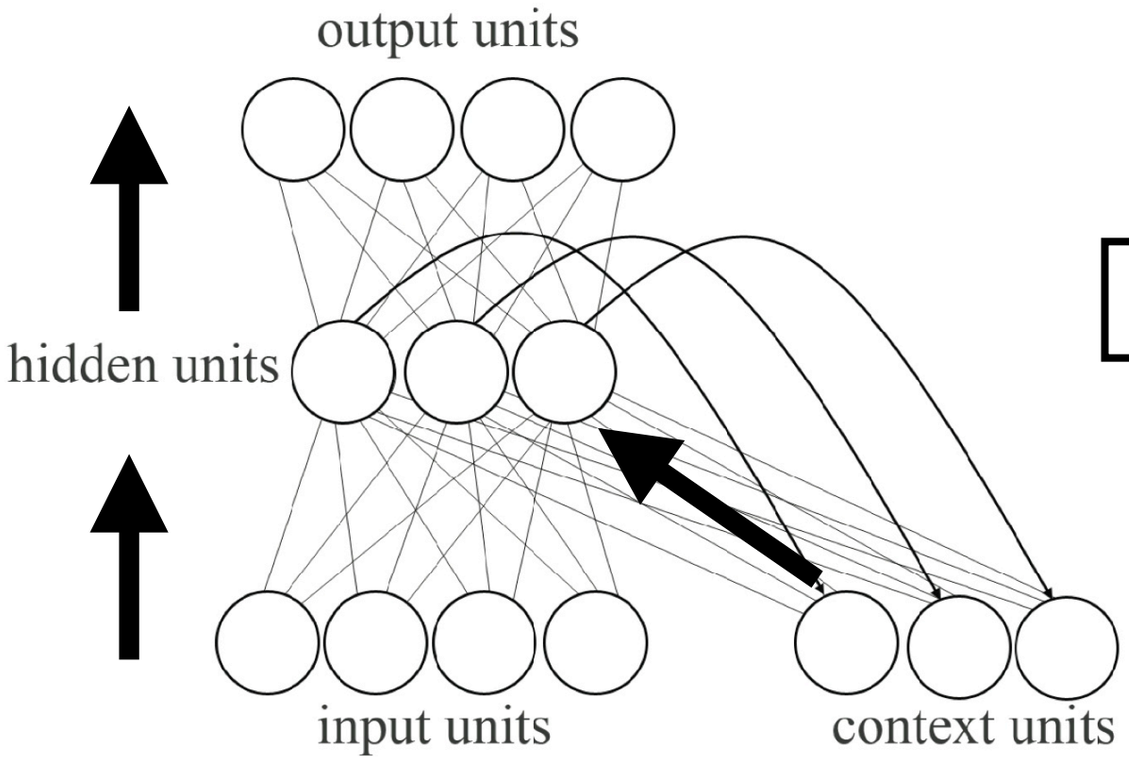
A similar simulation with a sequence of artificial sentences such as “manyyearsagoaboyandgirllivedbythesea..”



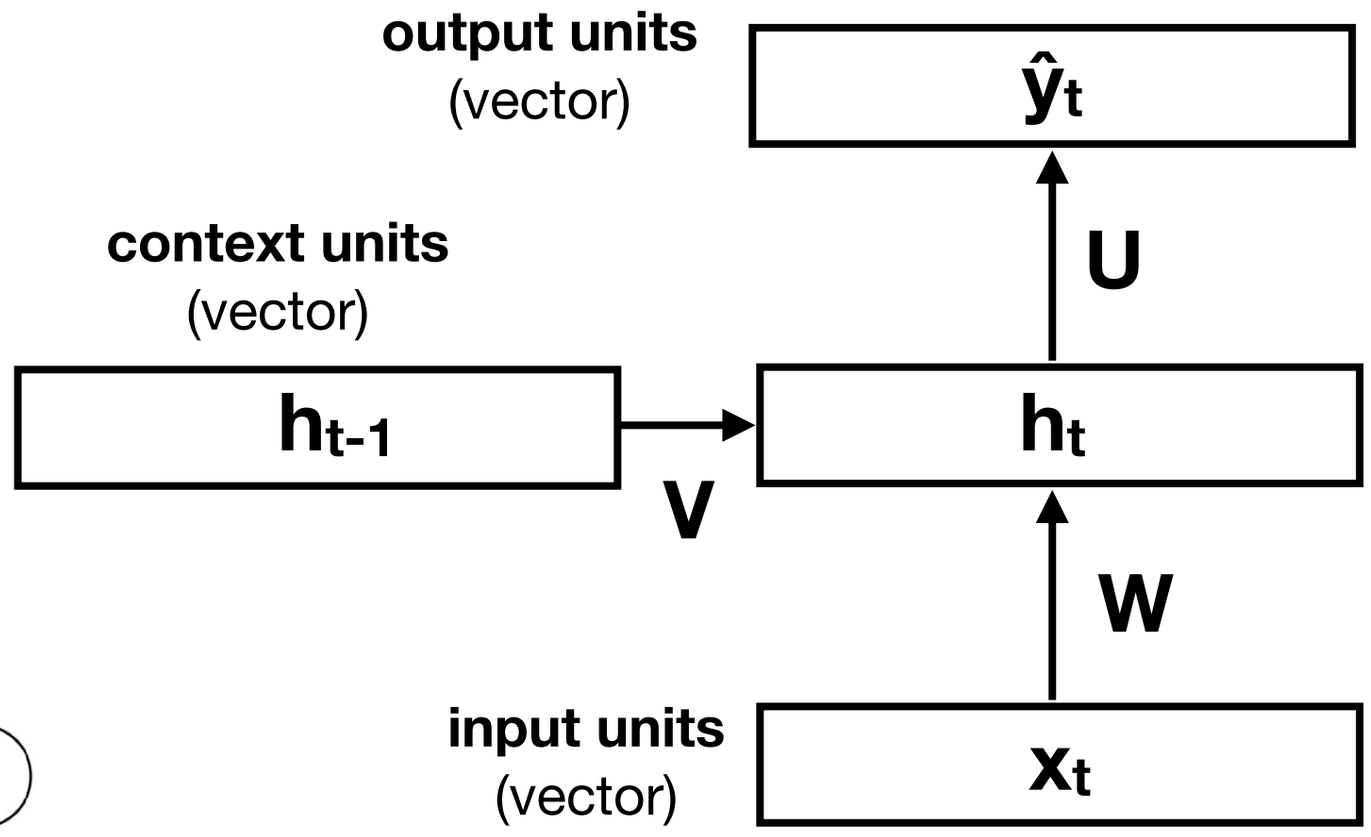
- Much previous work in cognitive science and linguistics assumed basic linguistic units such as phonemes, morphemes, words, etc.
- But the definition of a “word” isn’t that clear
- How many words is each of these phrases? (examples from PDP Handbook)
  - ‘line drive’, ‘flagpole’, ‘carport’, ‘gonna’,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

Training a recurrent neural network

old notation



new notation

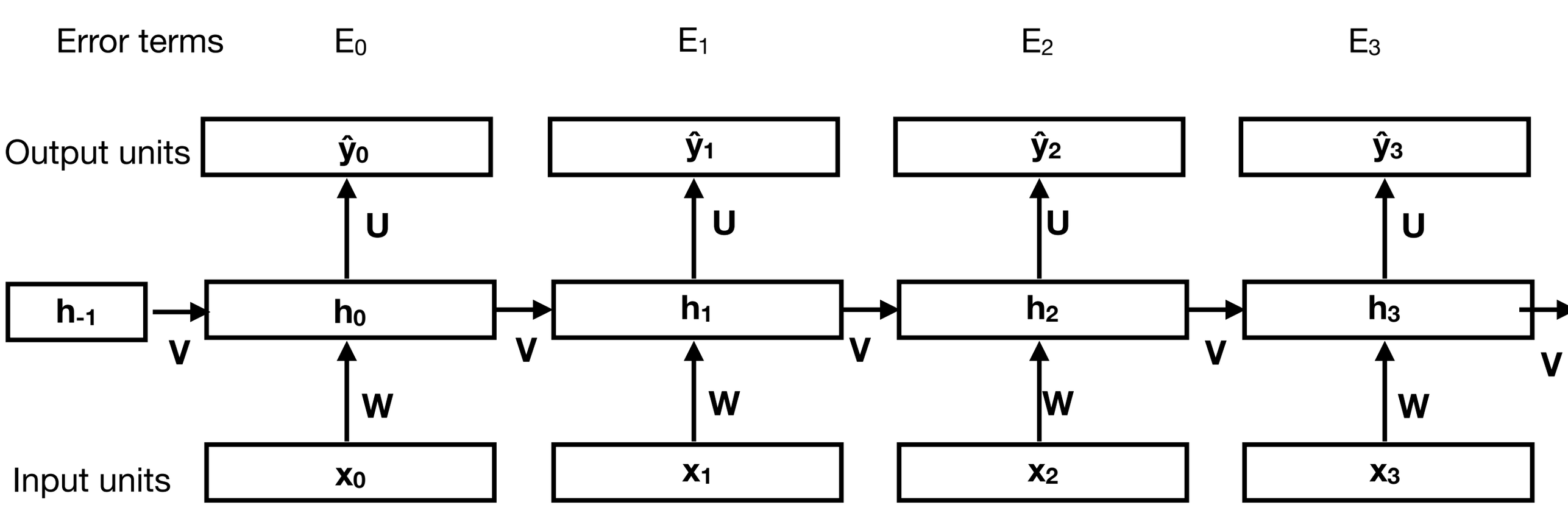


weight matrices **V, W, U**

index **t** is represents the step in time

Unrolling a recurrent network in time

Global error $E = \sum_t E_t$



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

i i b a

Output units $\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$

U U U U

h_{-1} h_0 h_1 h_2 h_3

V V V V V

W W W W

Input units x_0 x_1 x_2 x_3

d i i b

Input sequence:

diibaguuu

weight matrices V, W, U

Reminder: Backpropagation algorithm for computing gradient

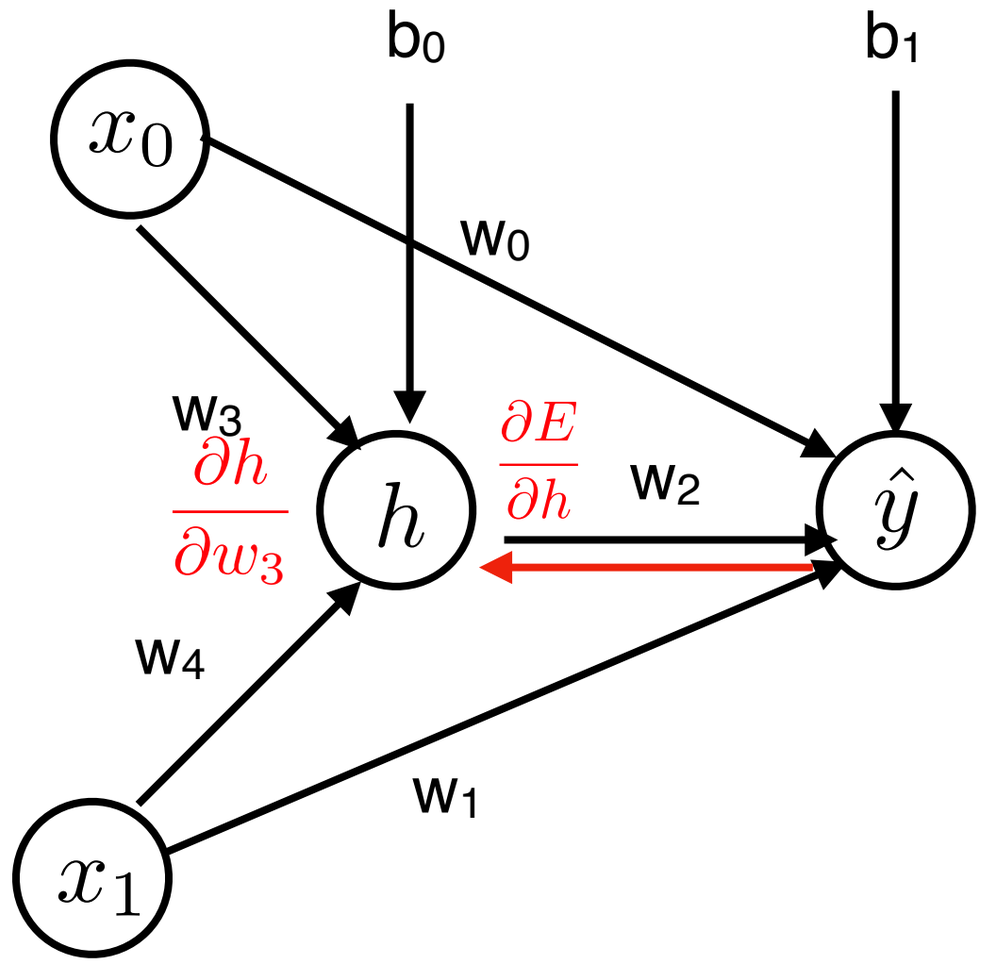
$$E(w,b) = (\hat{y} - y)^2$$
$$= (g(\text{net}) - y)^2$$

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

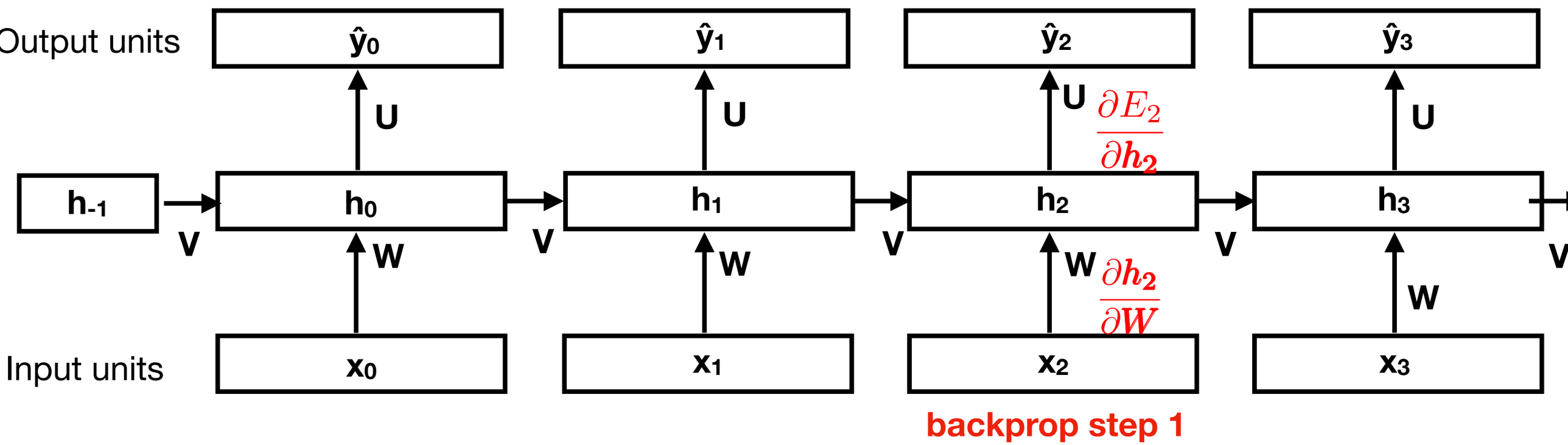
Step 2) Compute how hidden unit activation changes as a function of weight



Backpropagation through time

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

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



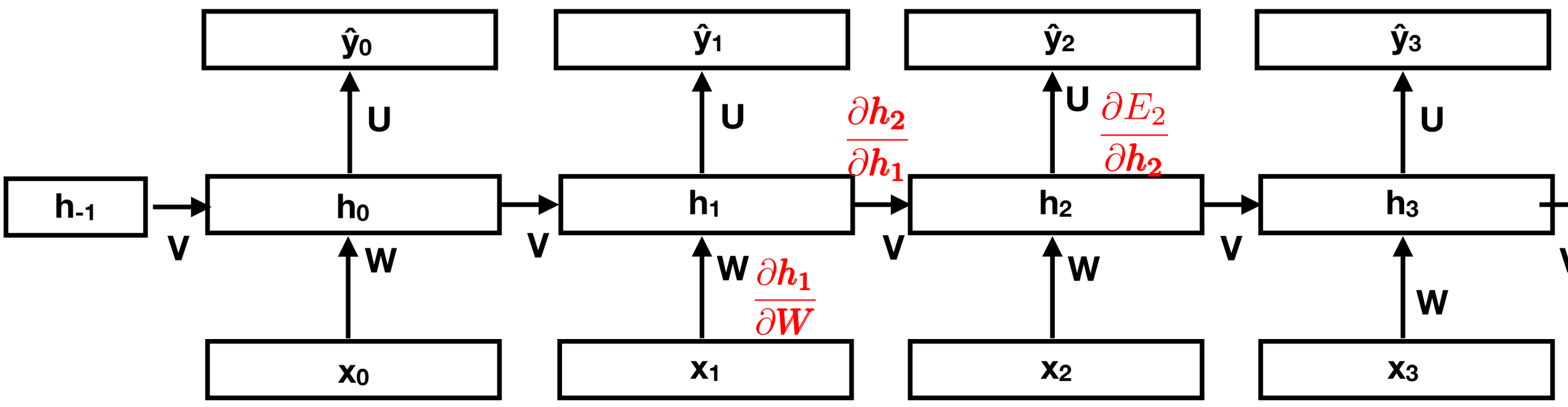
backprop step 1

weight matrices V, W, U

Backpropagation through time

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

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



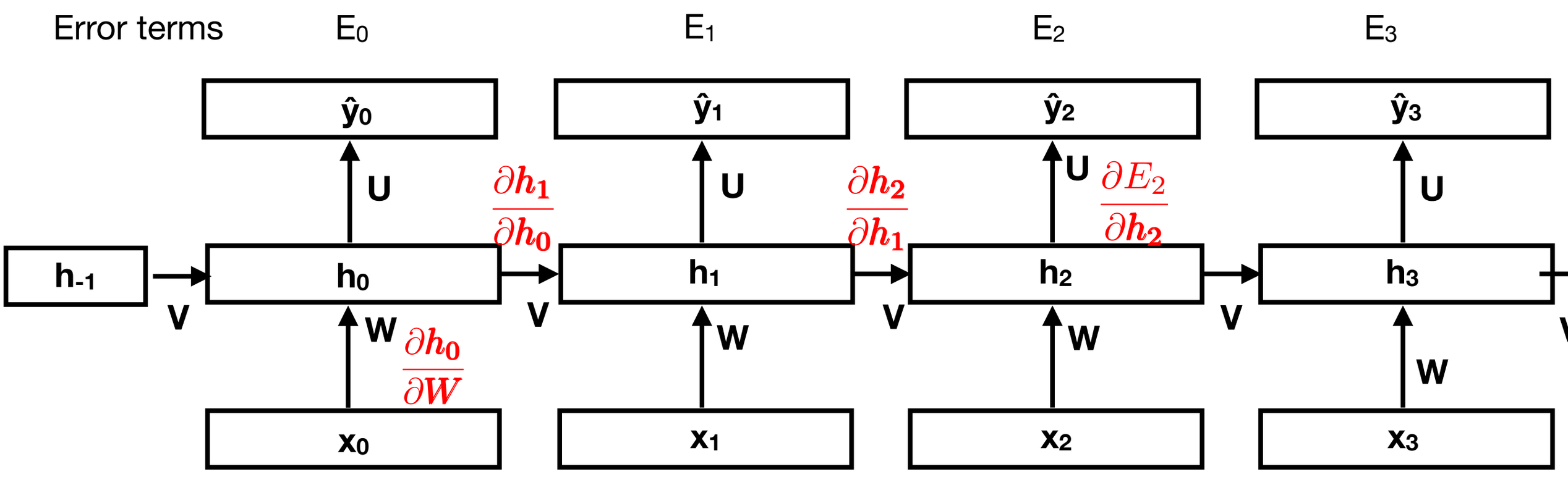
backprop step 2

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!

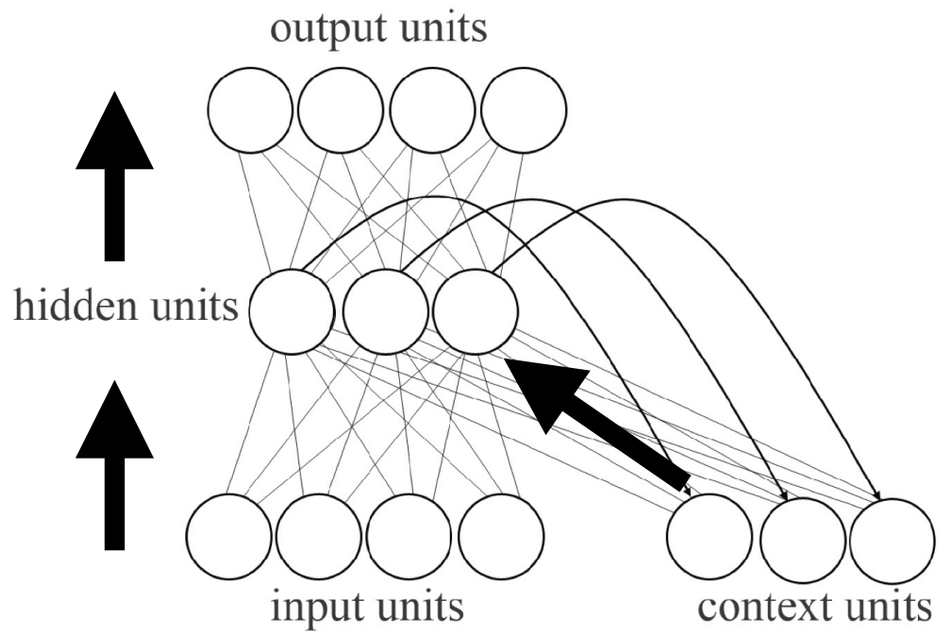
Fortunately, PyTorch (or TensorFlow, etc.) can compute all of the gradients for you!

```
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(dim=1)

    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 + h
        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 torch.zeros(1, self.hidden_size, requires_grad=True)
```

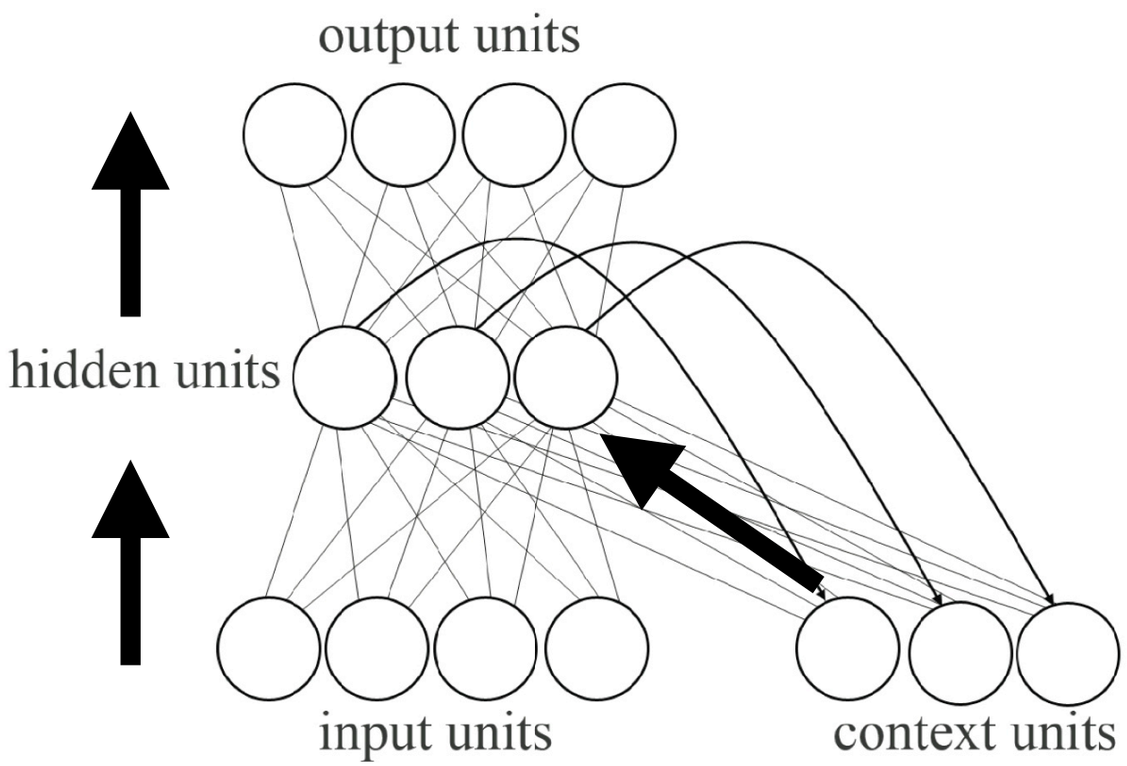



```
def train(seq_tensor, rnn):
    # seq_tensor: [seq_length x 1 x nsymbols tensor]
    # rnn : instance of SRN class
    hidden = rnn.initHidden()
    rnn.train()
    rnn.zero_grad()
    loss = 0
    seq_length = seq_tensor.shape[0]
    for i in range(seq_length-1):
        output, hidden = rnn(seq_tensor[i],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

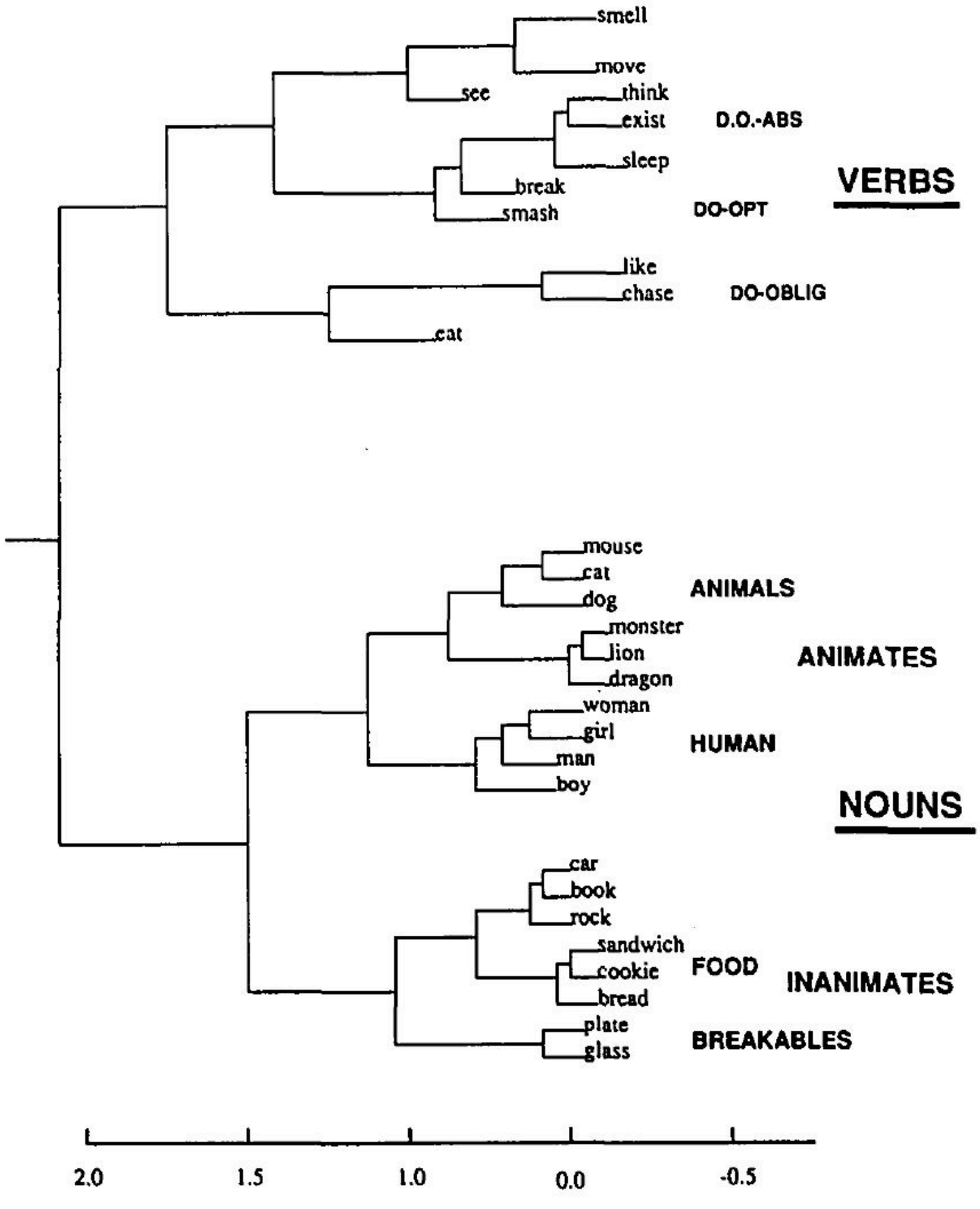
Discovering lexical classes from simple sentences

"one-hot" encoding for words

TABLE 5
Fragment of Training Sequences for Sentence Simulation

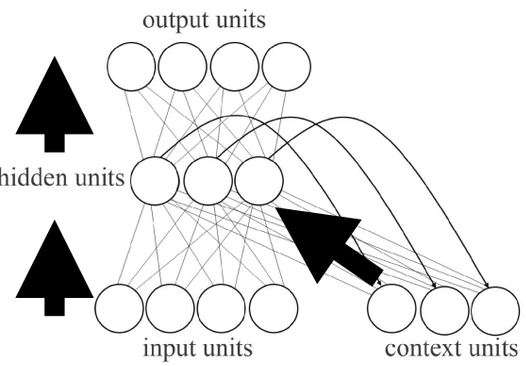
Input	Output
00000000000000000000000000000010 (*woman*)	00000000000000000000000000010000 (*smash*)
00000000000000000000000000010000 (*smash*)	00000000000000000000001000000000 (*plate*)
00000000000000000000001000000000 (*plate*)	00000100000000000000000000000000 (*cat*)
00000100000000000000000000000000 (*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

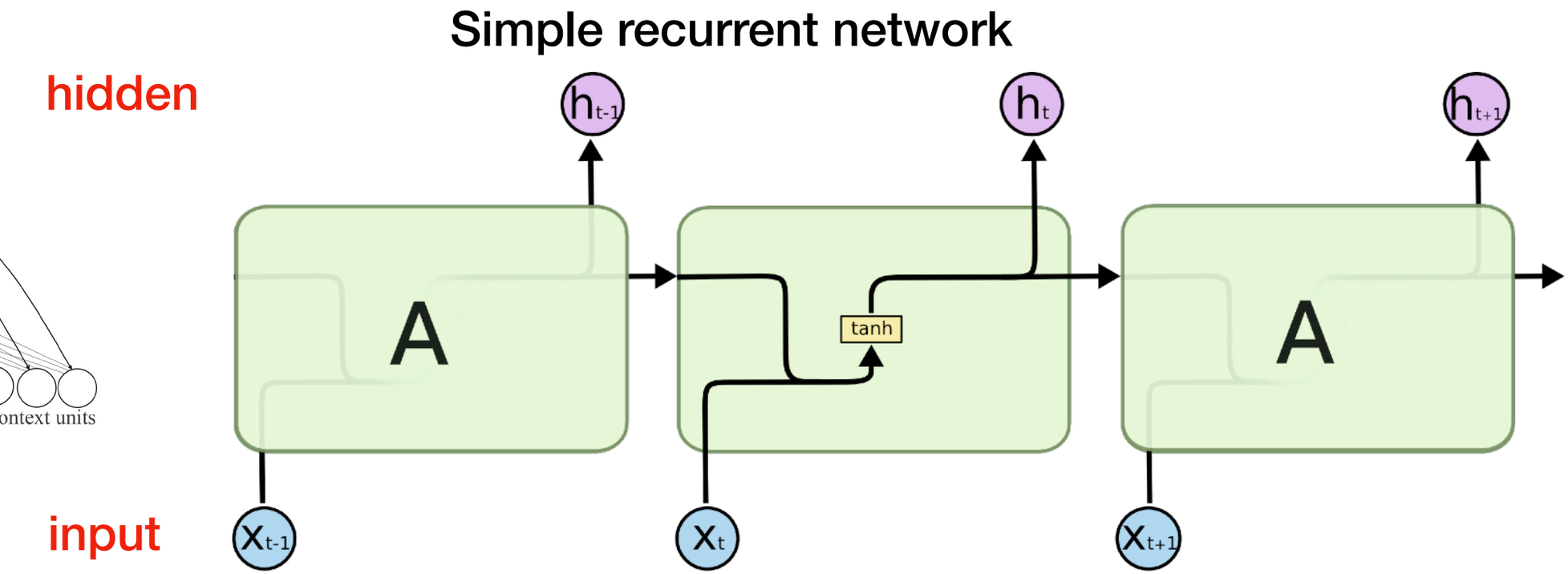


- Rich structure from just learning to **predict the next word** from the **previous words**
- This diagram shows results of clustering the average pattern over the hidden units for each word (across many presentations)
- From just the prediction task, the network “discovers” nouns vs. verbs, and also 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 network



Long short-term memory (LSTM) network

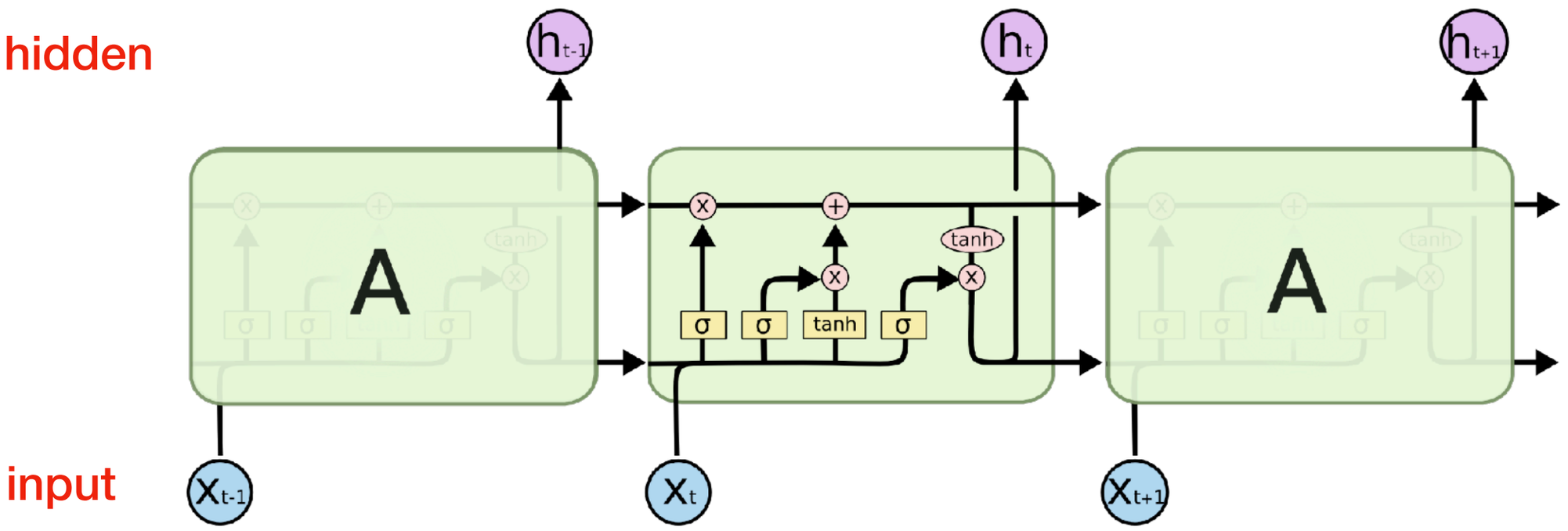
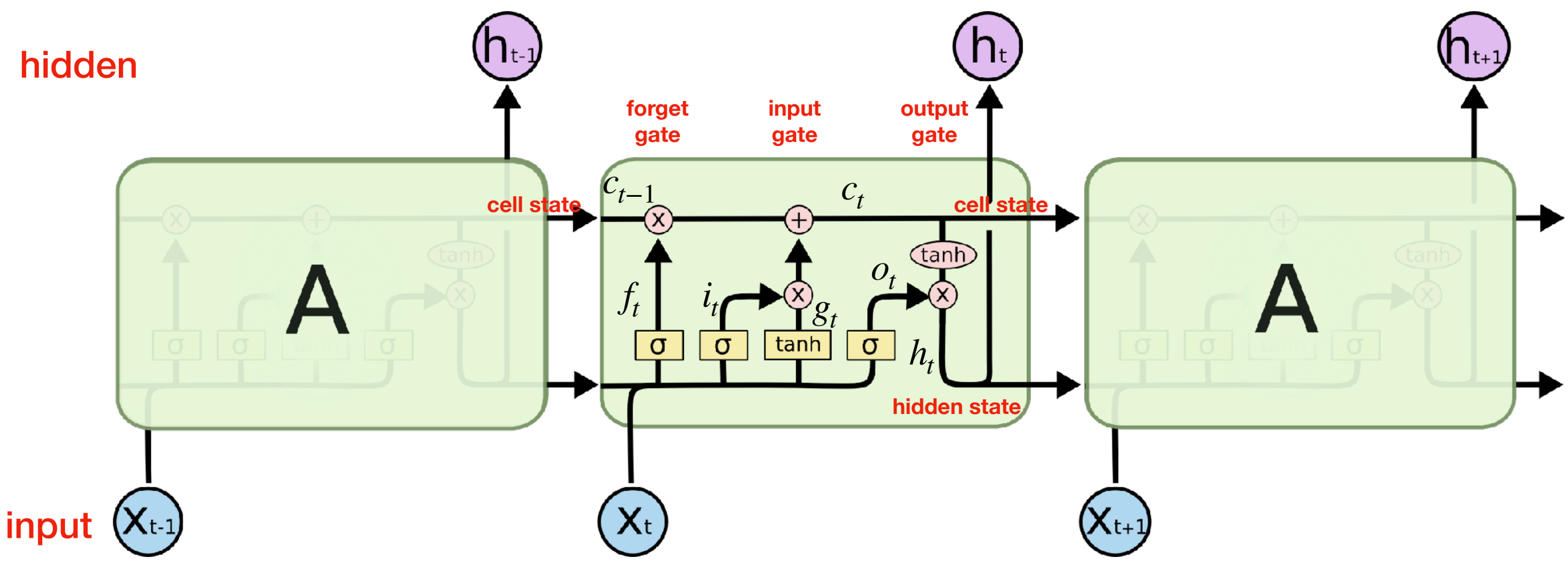


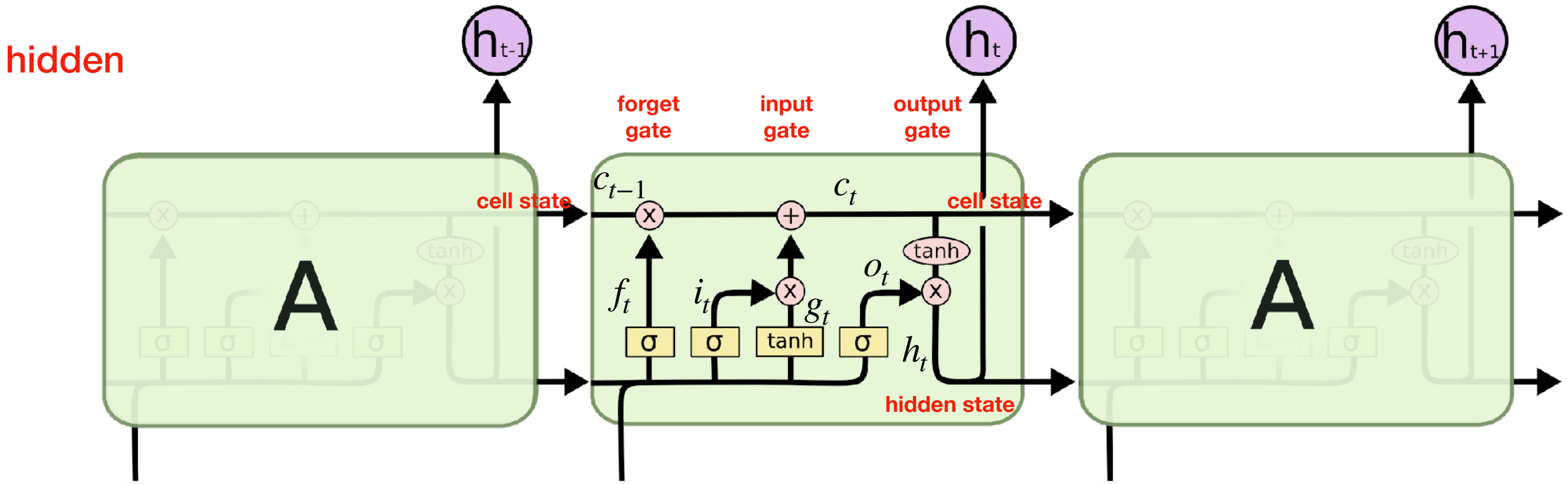
Image credit: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long short-term memory (LSTM)

- 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)
- 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 RNN 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.



Long short-term memory (LSTM)



updating the cell state

forget some cell elements *f*, add to other elements *i*

$$c_t = f_t c_{(t-1)} + i_t g_t$$

forget gate

compute which cell elements to forget (*f*)

$$f_t = \textbf{logistic}(W_{if} x_t + W_{hf} h_{(t-1)})$$

input gate

compute which cell elements to add to (*i*), and what to add (*g*)

$$i_t = \textbf{logistic}(W_{ii} x_t + W_{hi} h_{(t-1)})$$

$$g_t = \textbf{tanh}(W_{ig} x_t + W_{hg} h_{(t-1)})$$

output gate

compute which cell elements to output (*o*), and output them (*h*)

$$o_t = \textbf{logistic}(W_{io} x_t + W_{ho} h_{(t-1)})$$

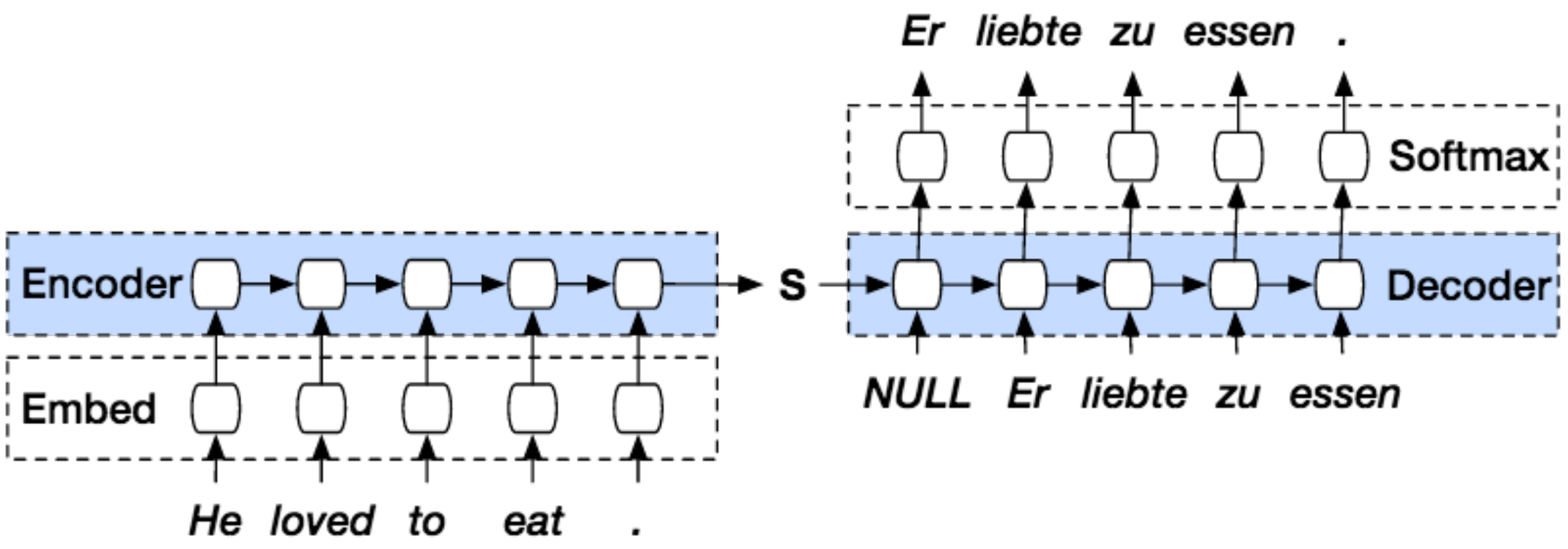
$$h_t = o_t \tanh(c_t)$$

Important notes:

- all vectors including cell, hidden, forget, etc. have same dimensionality (except *x*)
- biases not shown in equations for simplicity

Neural machine translation with recurrent neural networks

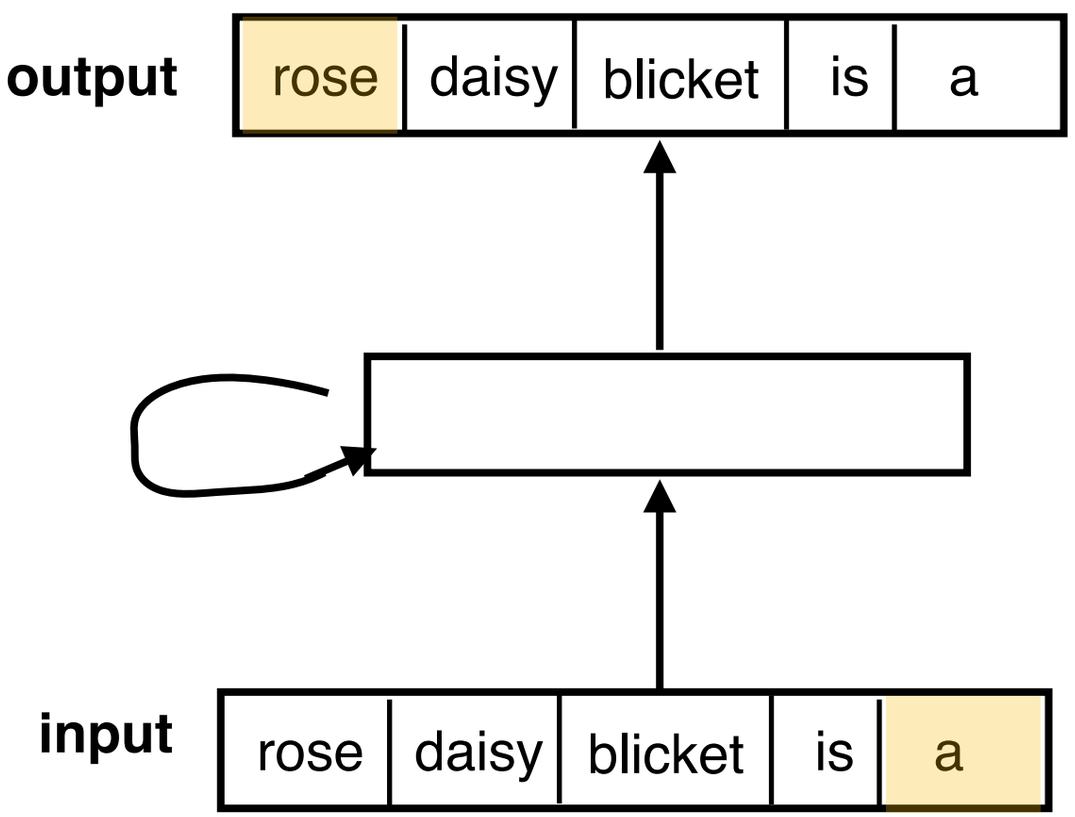
- Recurrent neural networks can solve sequence-to-sequence (seq2seq) problems by having a recurrent encoder, which embeds a sentence in the hidden space, and another recurrent decoder that translates it.
- This is the main technical juice behind Google Translate! (circa 2018)
- Trained using sentence pairs and backprop through time



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

Critiques of recurrent neural networks

(Marcus, 1998, Rethinking eliminative connectionism)

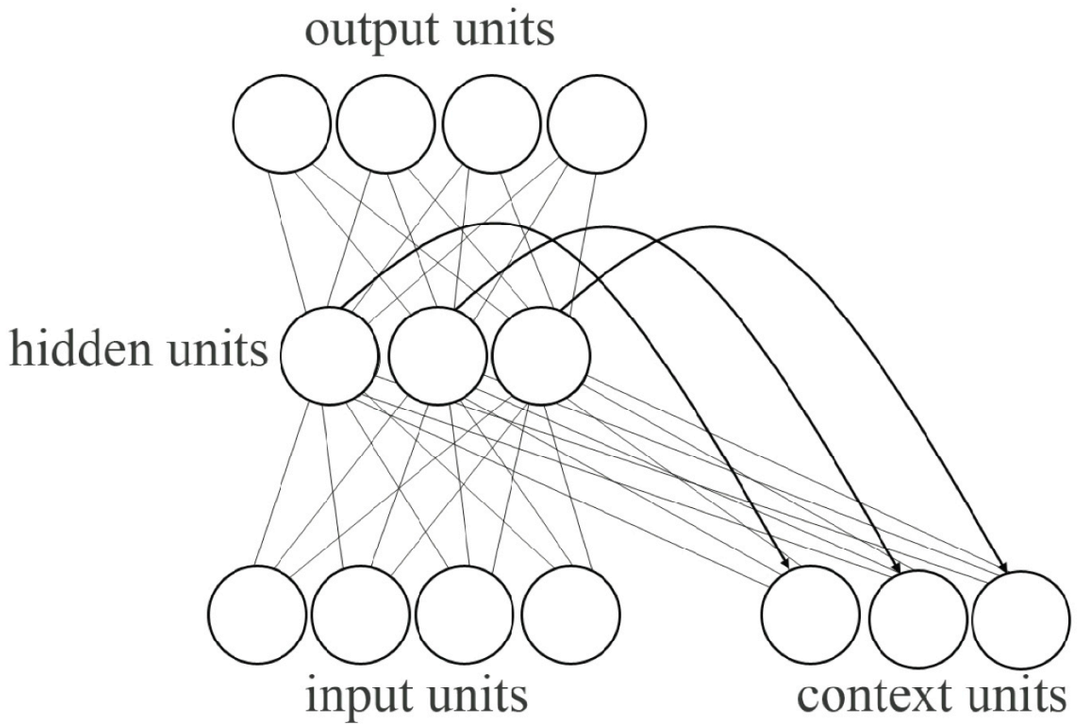


Sentence input… “A rose is a”

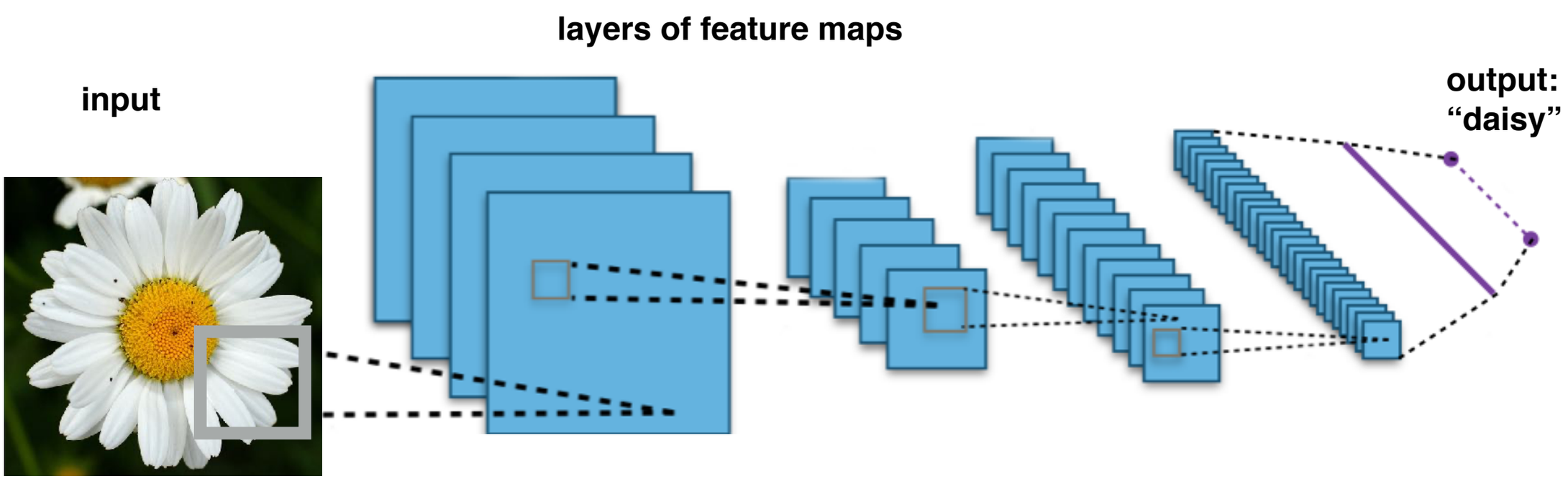
- Recurrent neural networks (RNNs) generalize very well WITHIN a specific set of words / symbols
- However, they generalize very poorly to new words OUTSIDE the training set
- Say the network is trained on sentences such as, “a rose is a rose”, “a daisy is a daisy”, and “a violet is a violet”
- It will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

Two very important types of neural networks

Recurrent neural networks (RNNs)

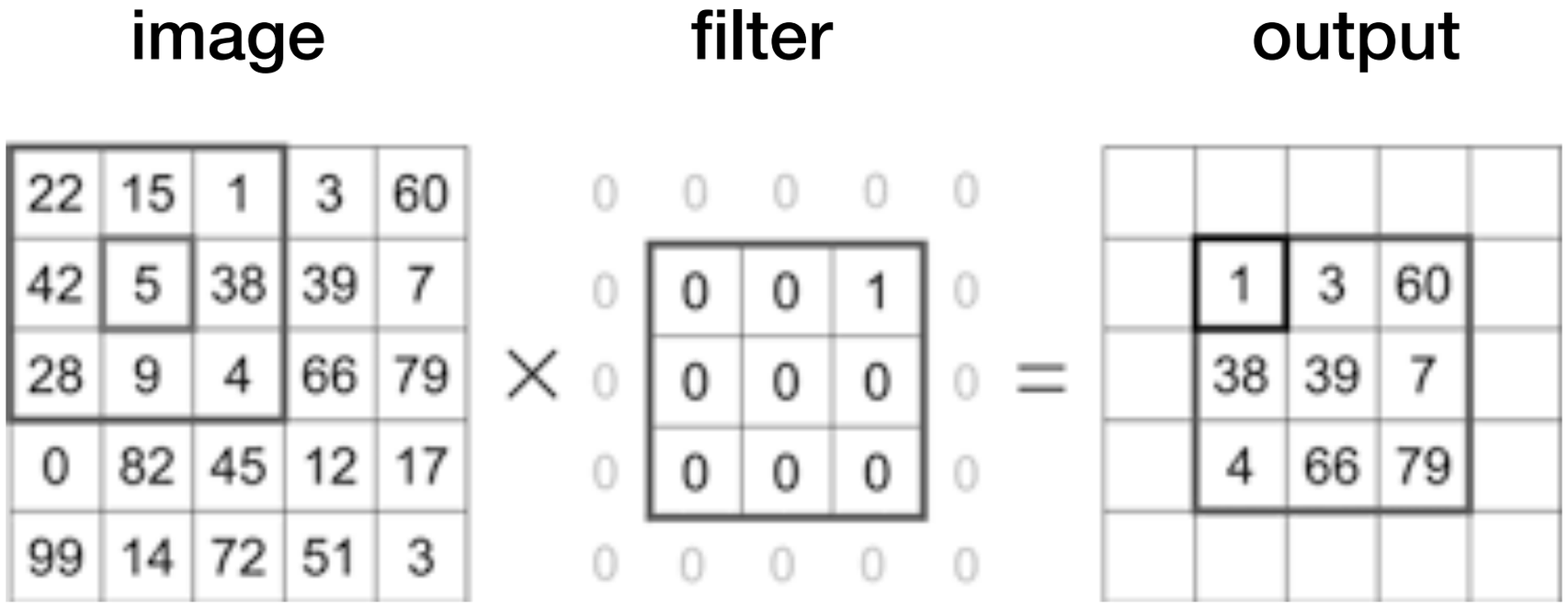


Convolutional neural networks (ConvNets)



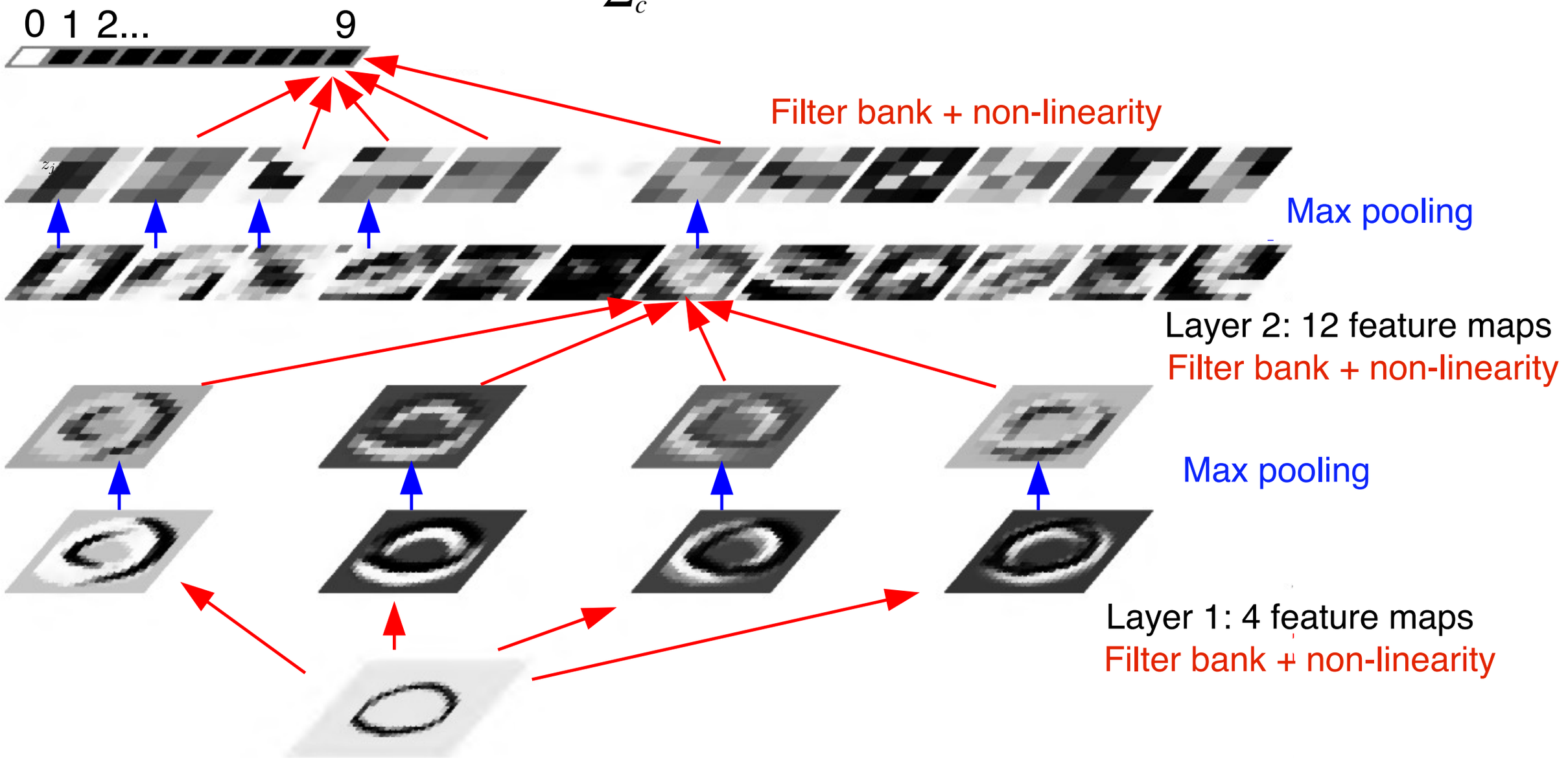
convolving an image with a filter

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

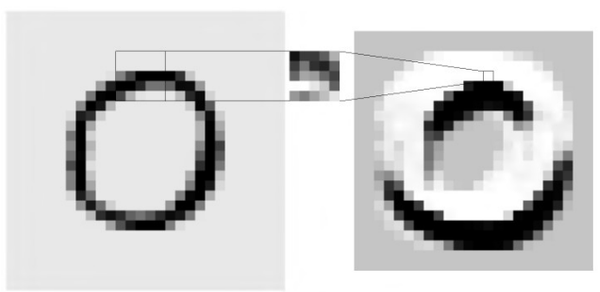


Deep convolutional neural network (convnet) for vision

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

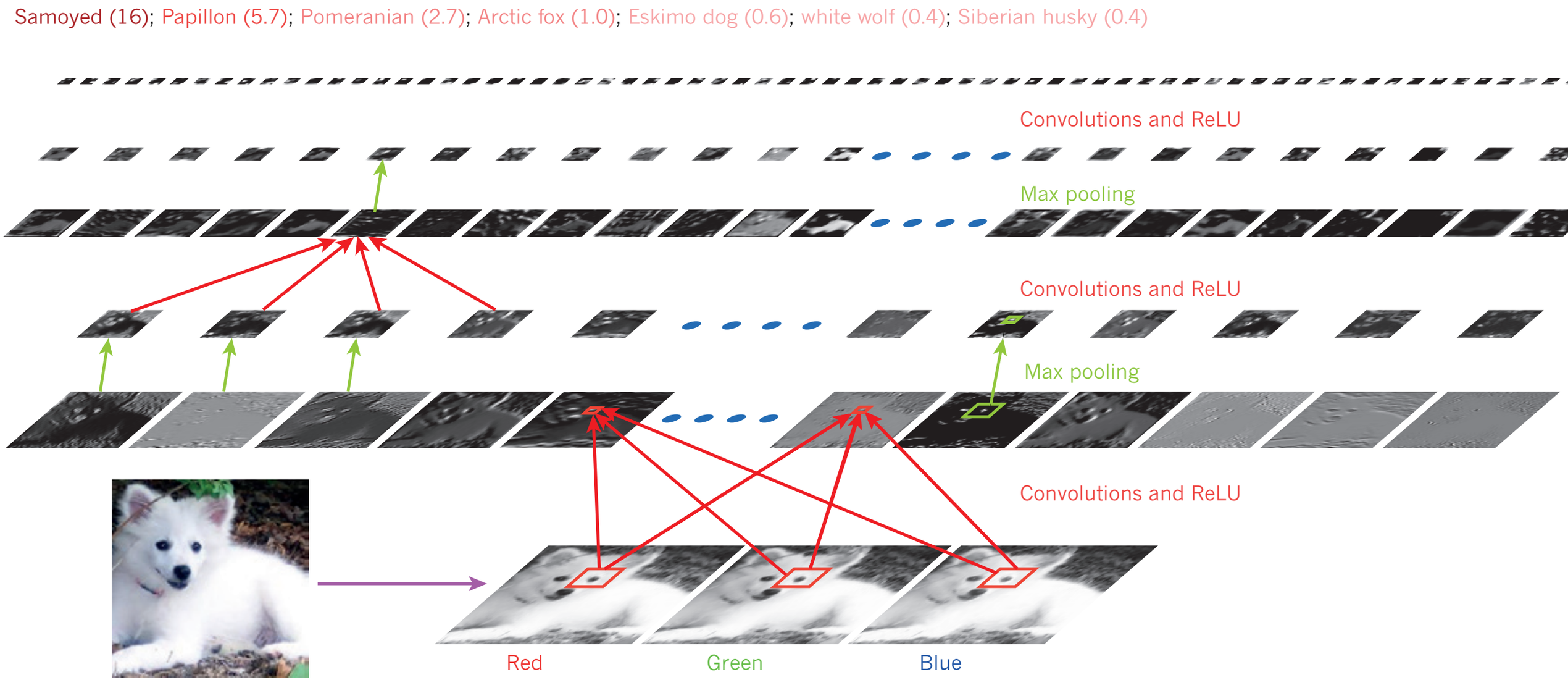


example convolution



Slide courtesy of Yann LeCun

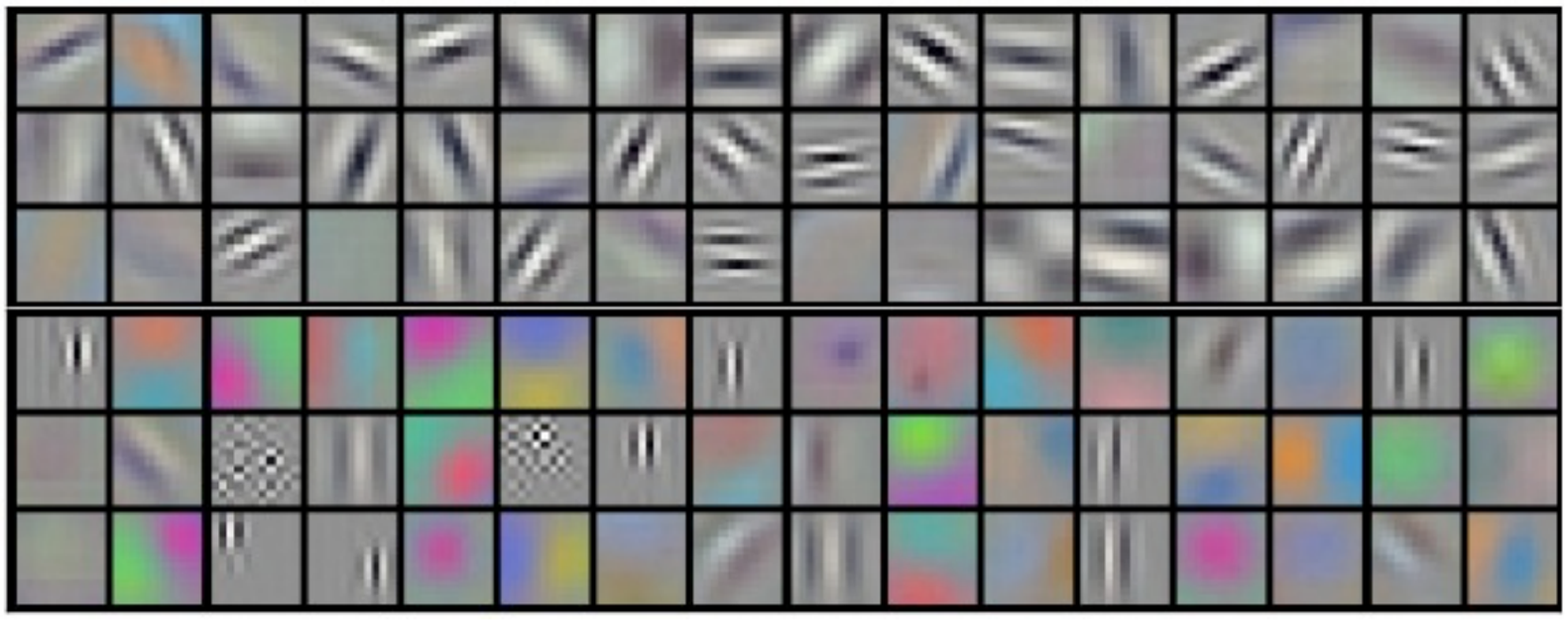
Deep convolutional neural network



From LeCun, Bengio, & Hinton (2015).

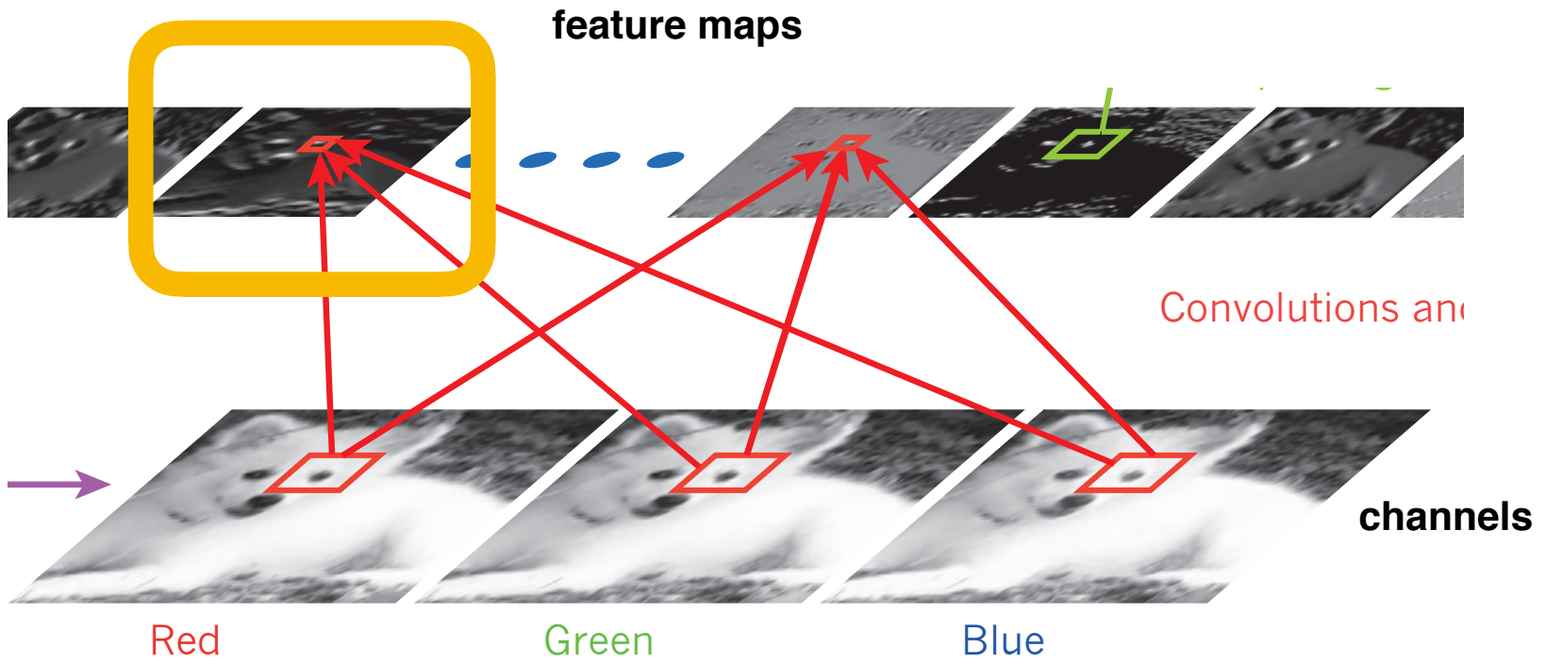
Learned filters in a deep convnet

- 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	0	2	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	0	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

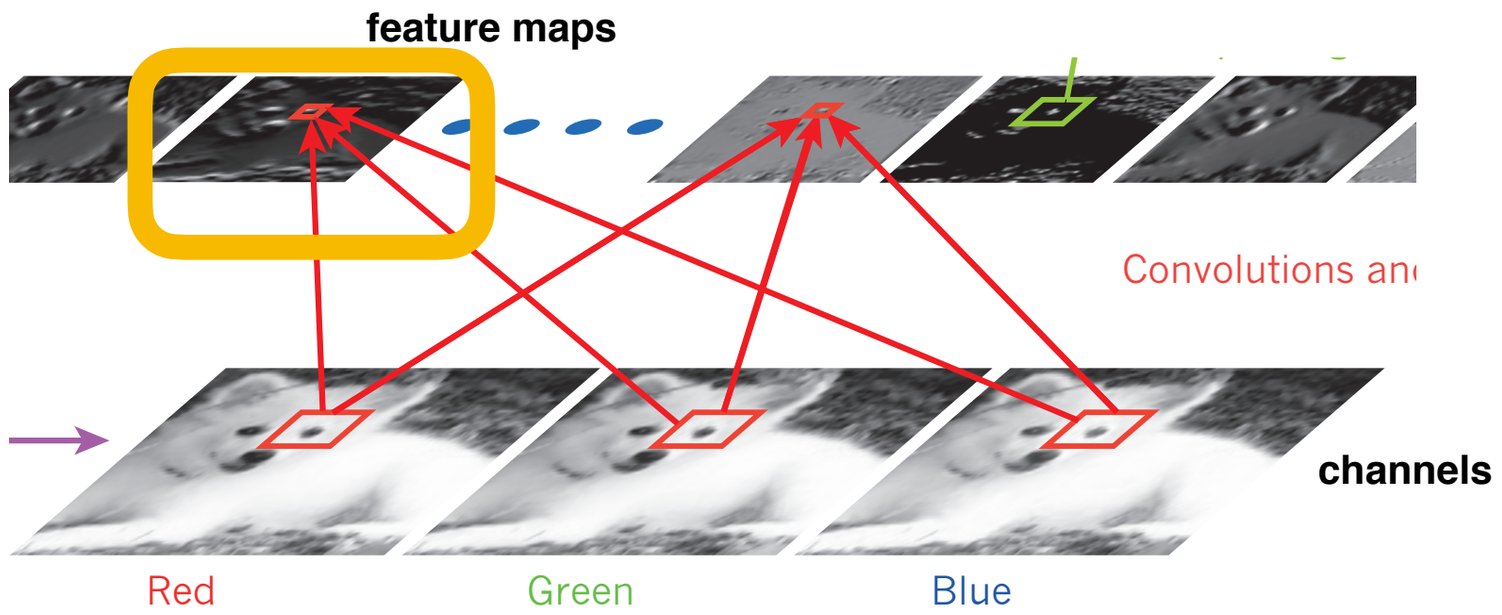
o[:,:,1]

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]

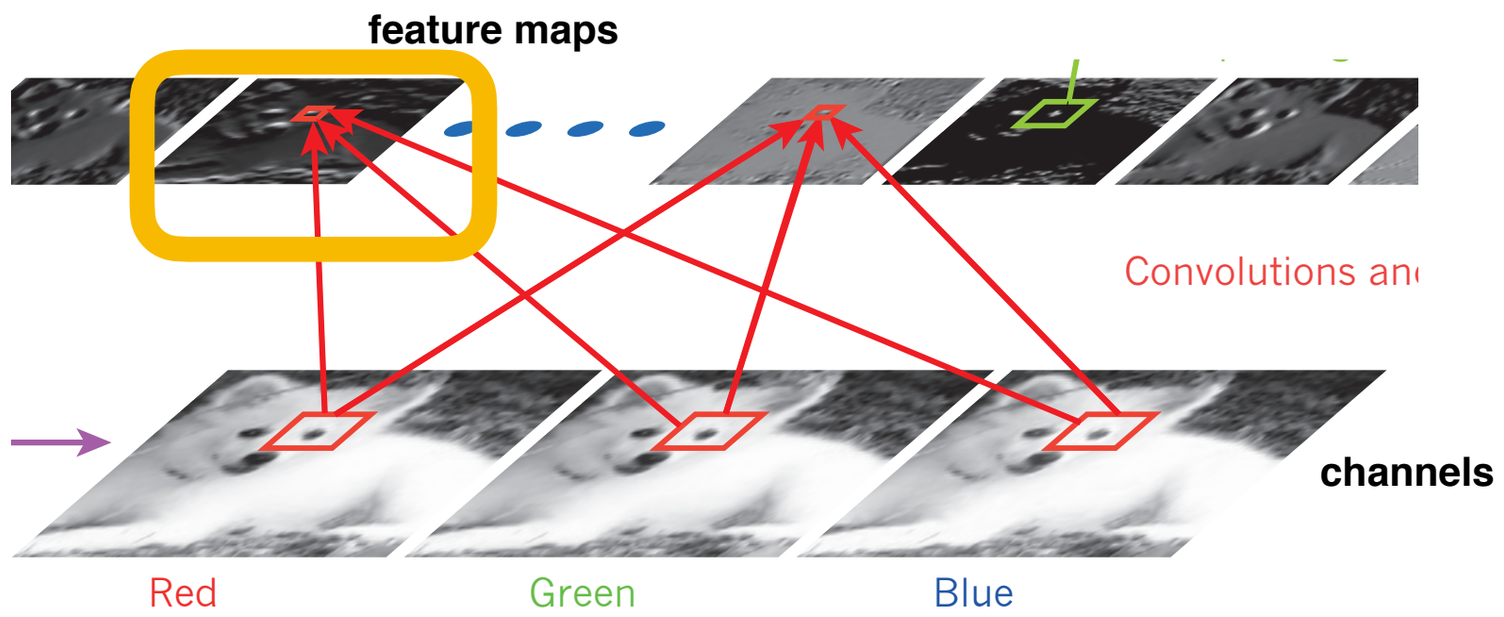
8	6	5
7	9	2
-1	7	3

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**

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]

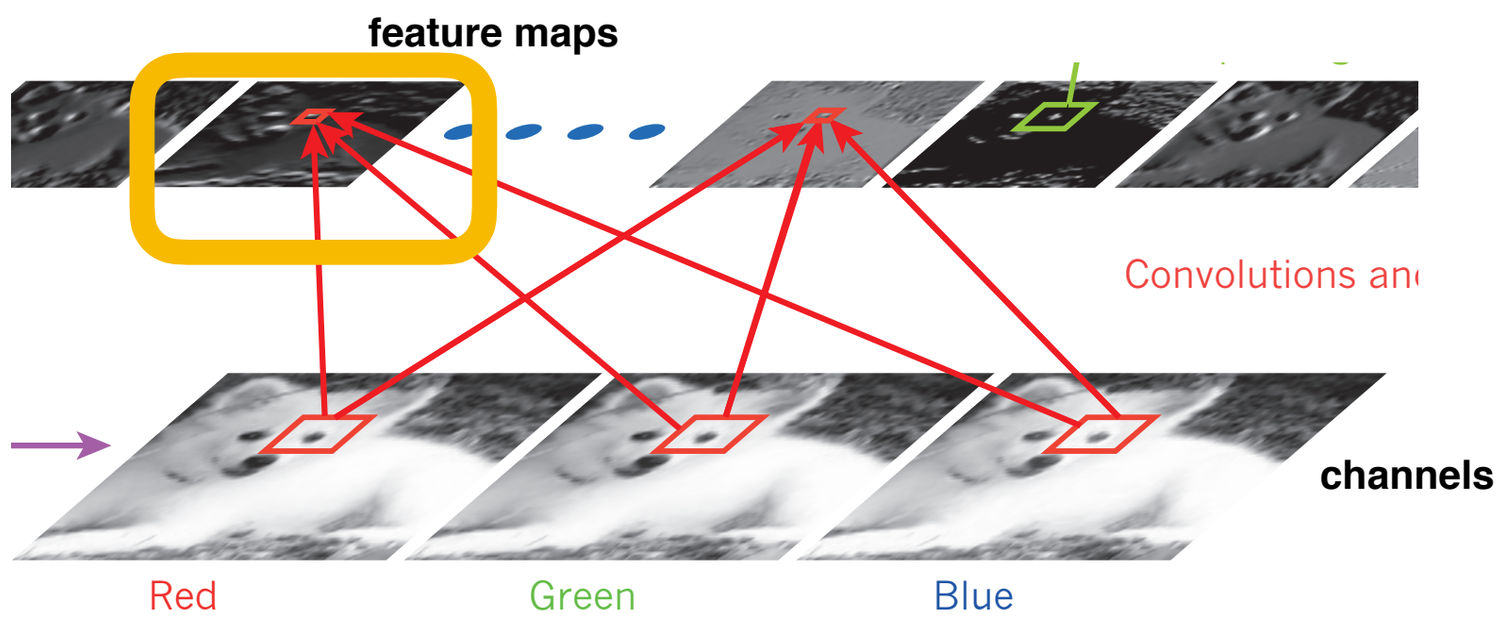
8	6	5
7	9	2
-1	7	3

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**

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)

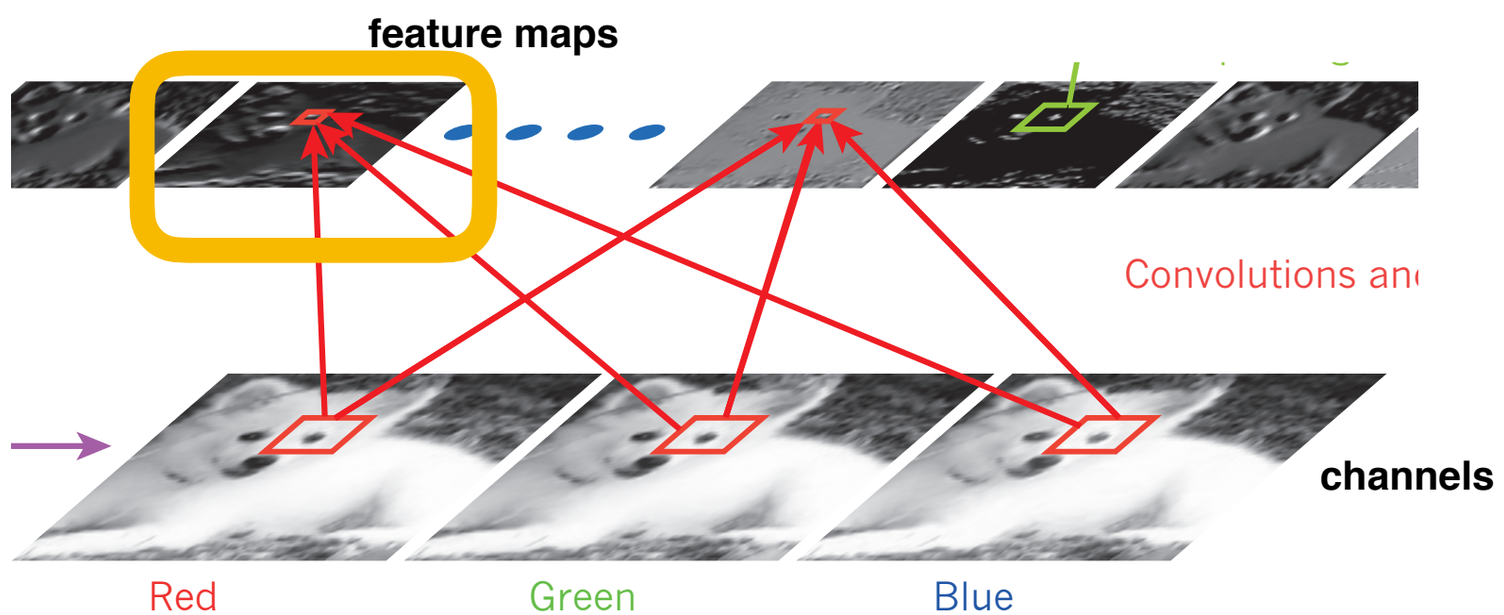
b0[:,:,0]

1

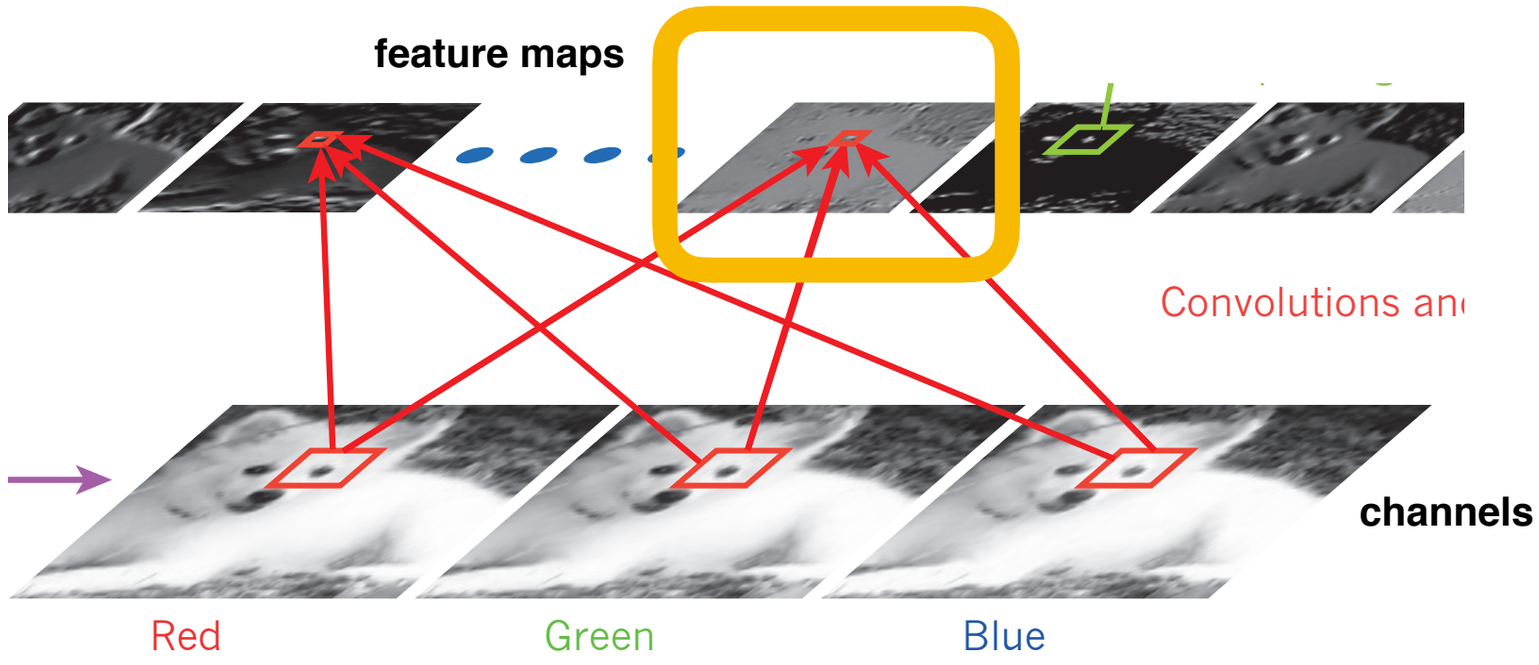
Bias b1 (1x1x1)

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

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]`

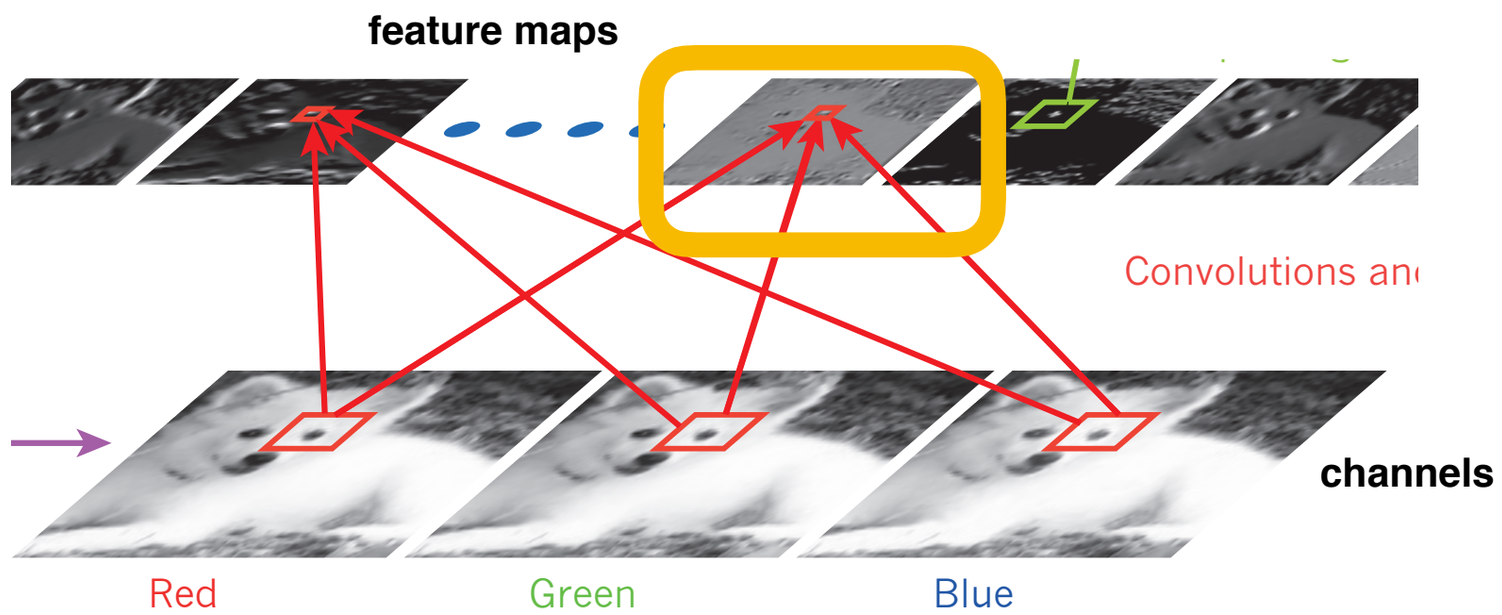
8	6	5
7	9	2
-1	7	3

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]`

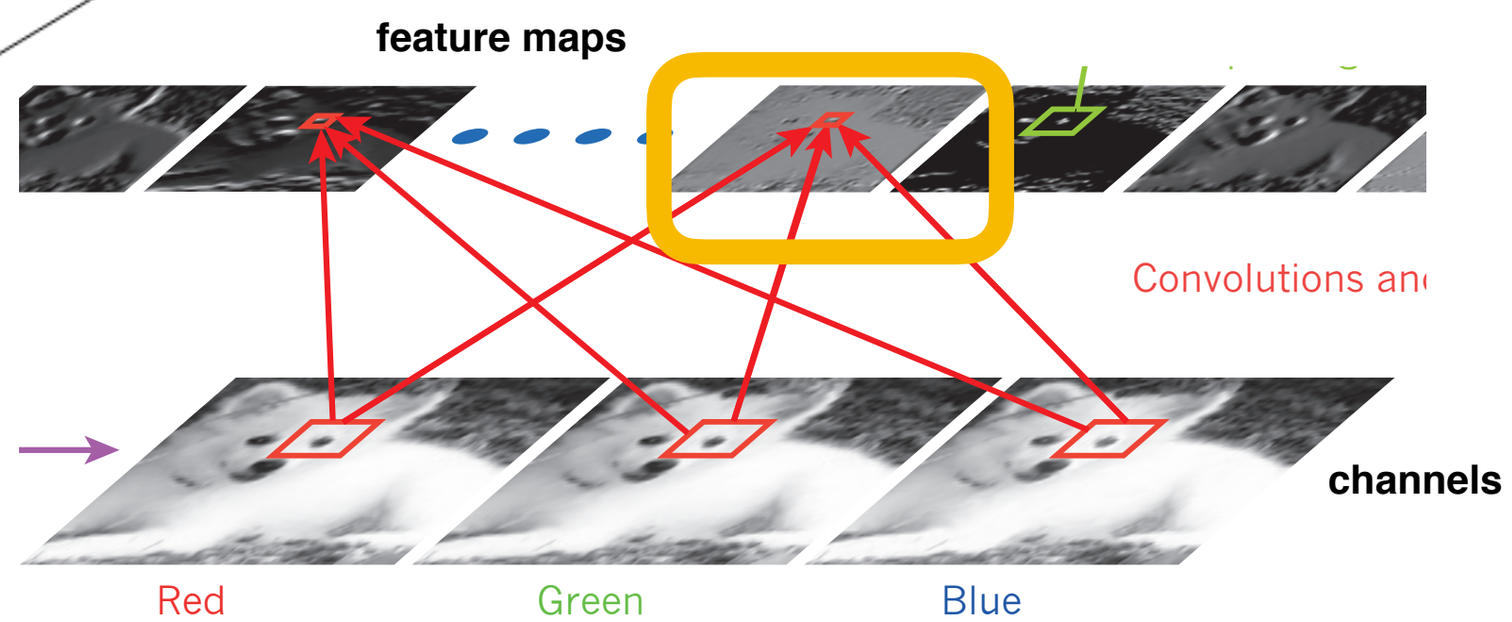
8	6	5
7	9	2
-1	7	3

Feature map 0

`o[:,:,1]`

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

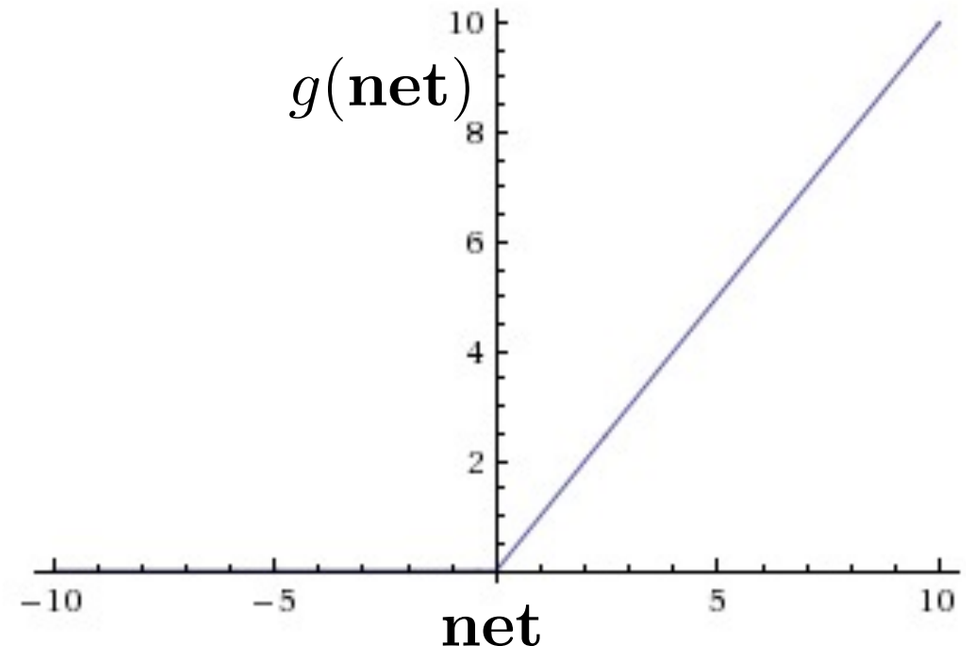
Feature map 1



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

$g(\mathbf{net})$ →

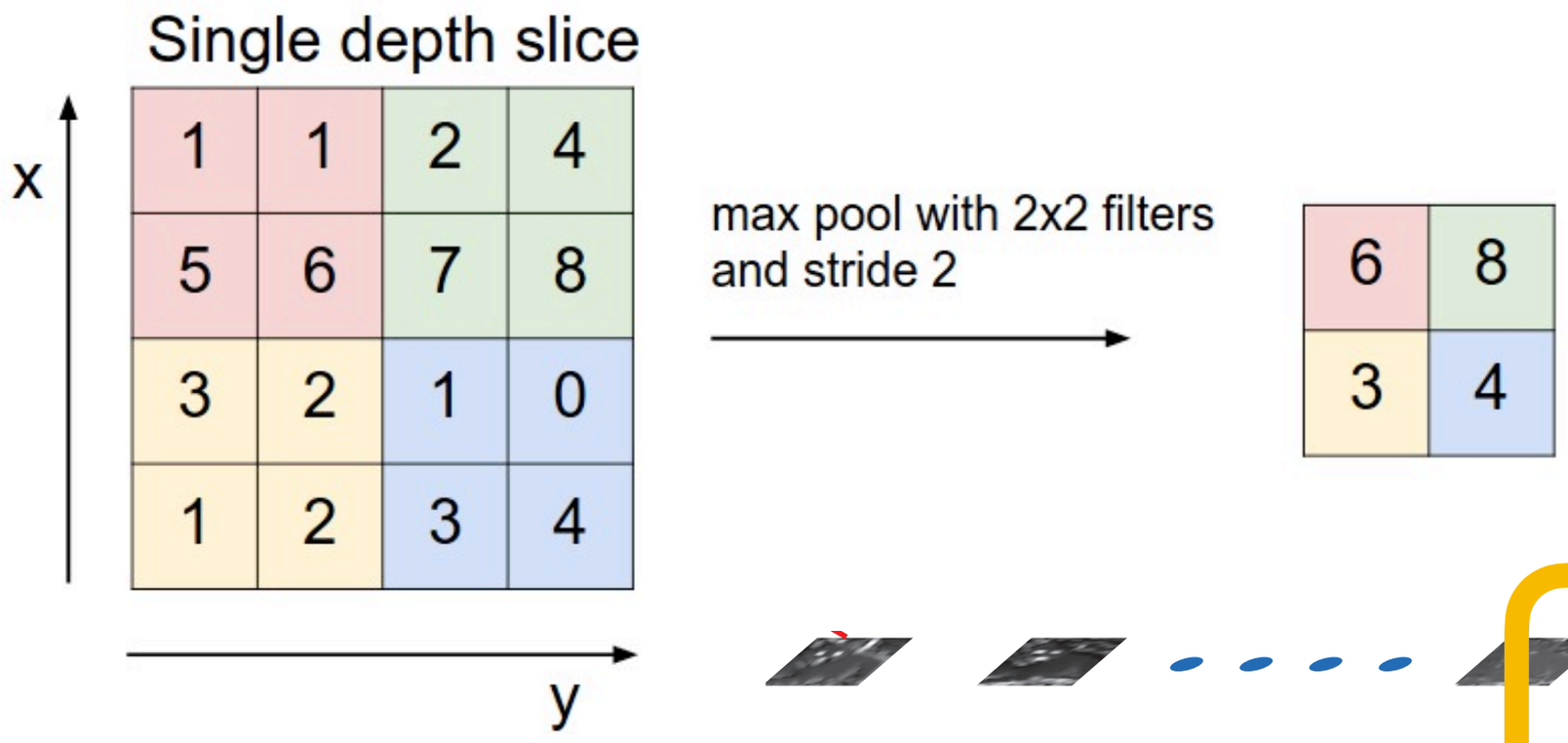
8	6	5
7	9	2
0	7	3

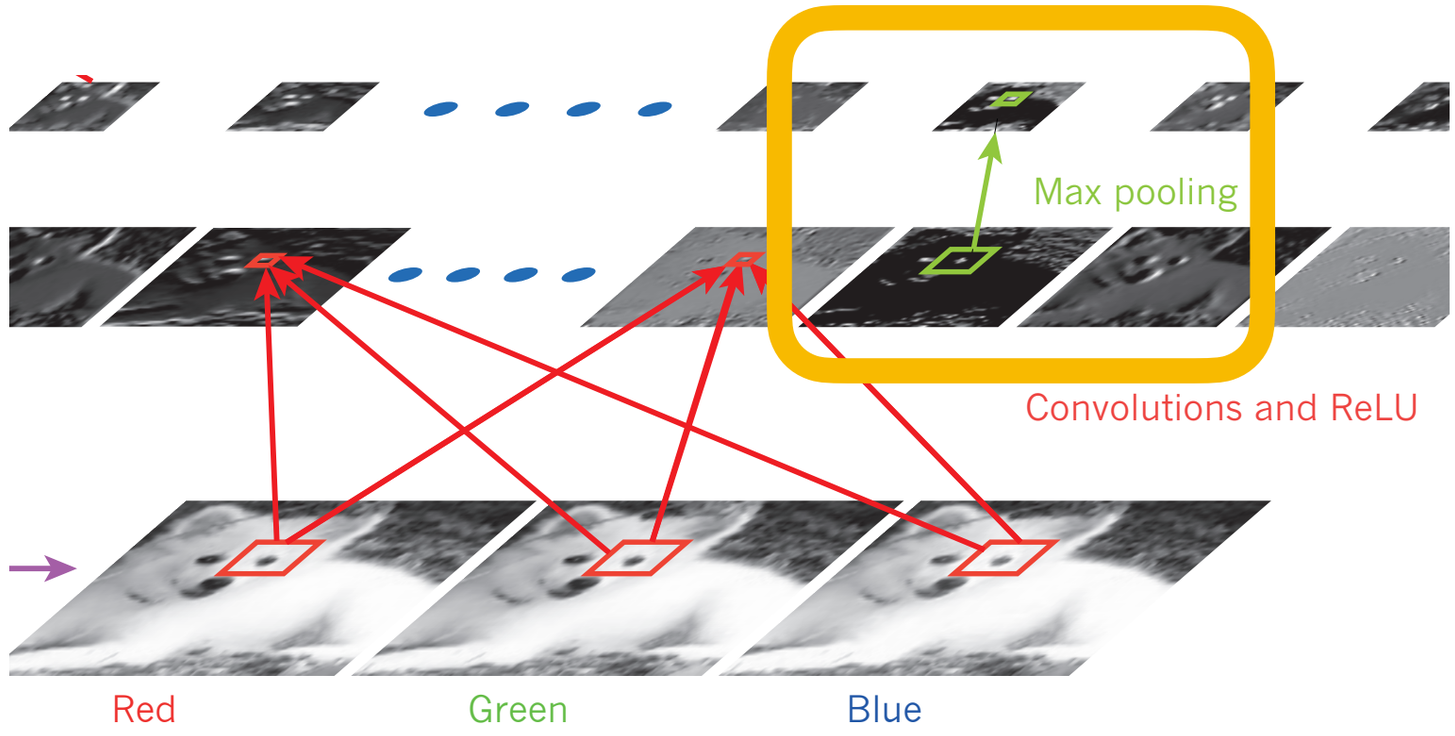
2	3	7
0	1	0
0	0	0

Next step: Pooling

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

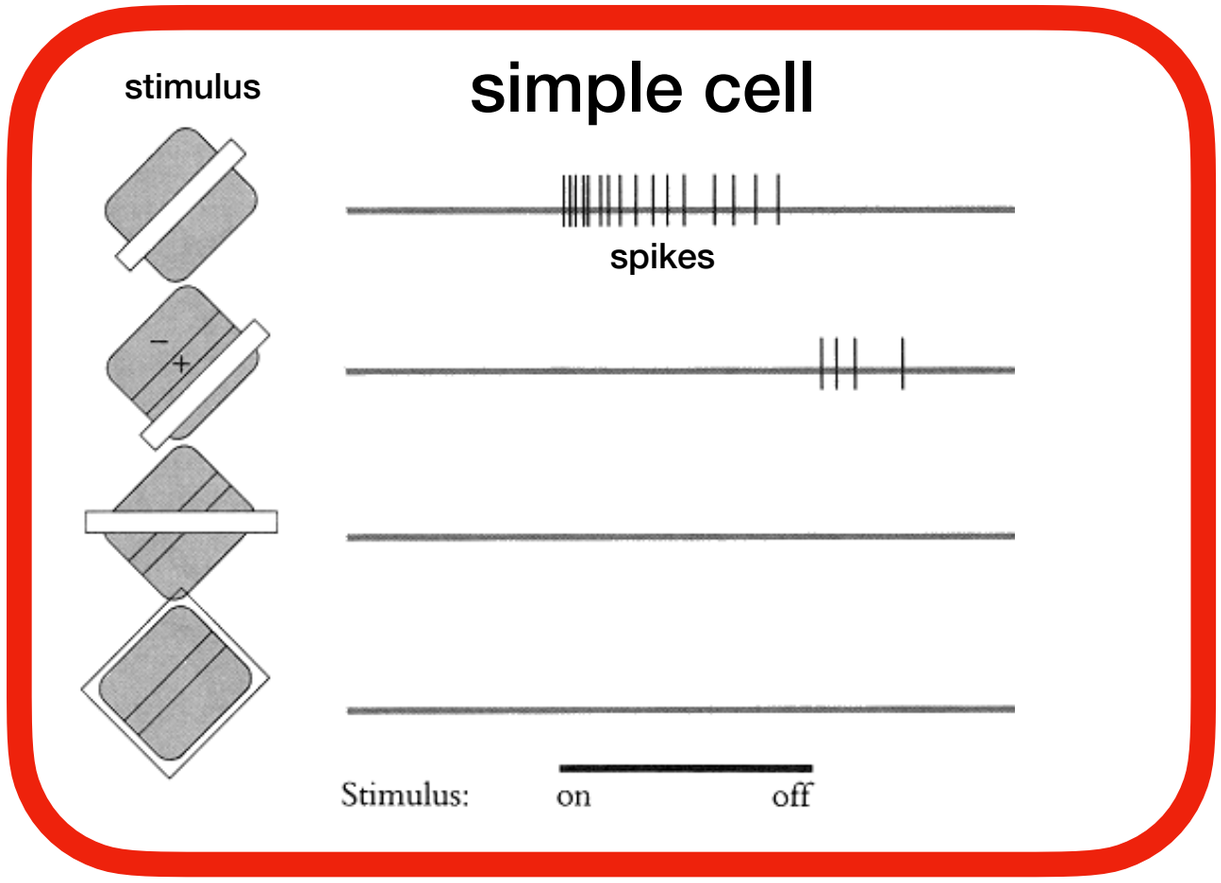
Max pooling



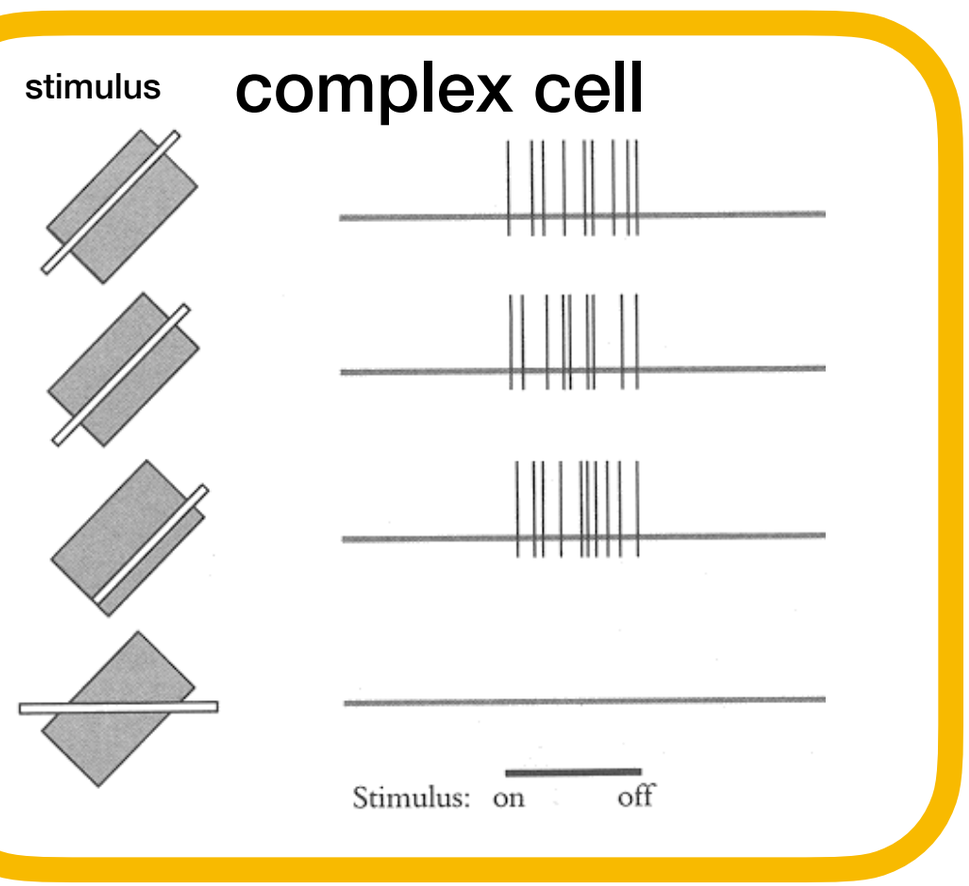


Connection to biological architecture of primary visual cortex

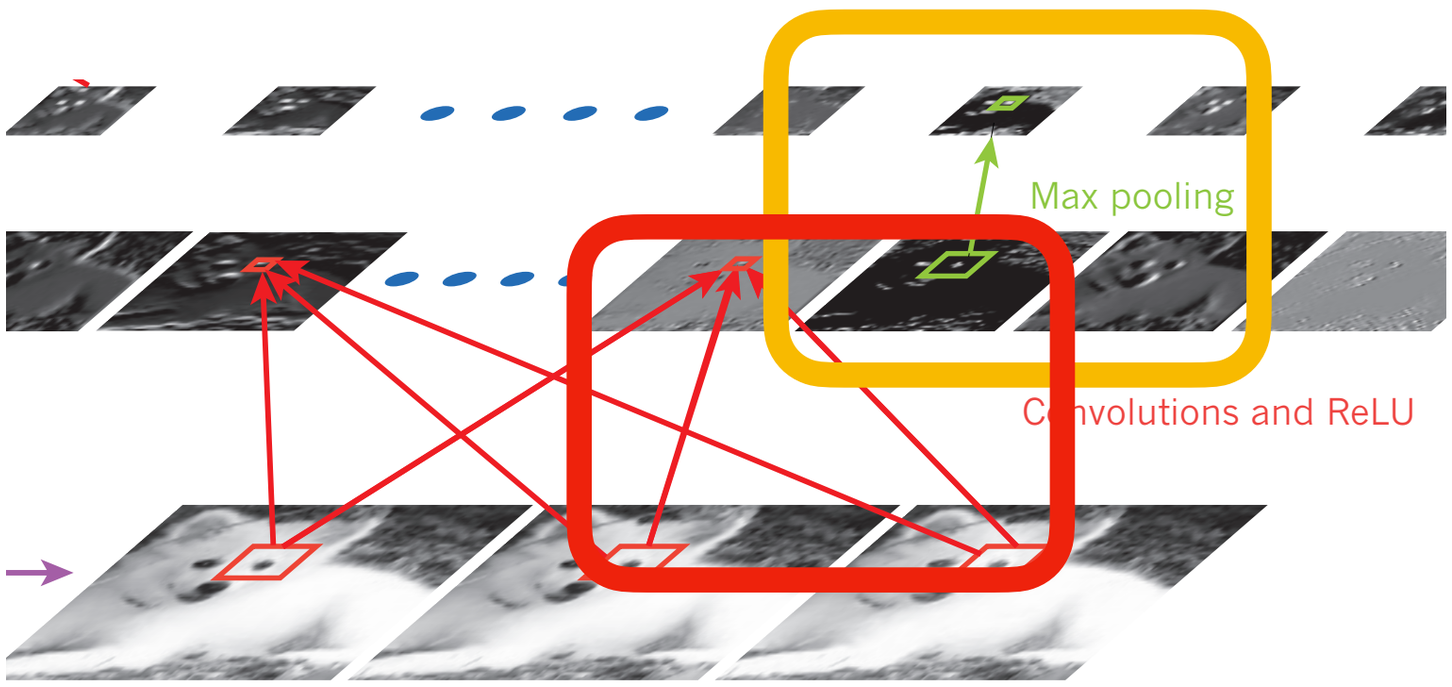
(Hubel & Wiesel)



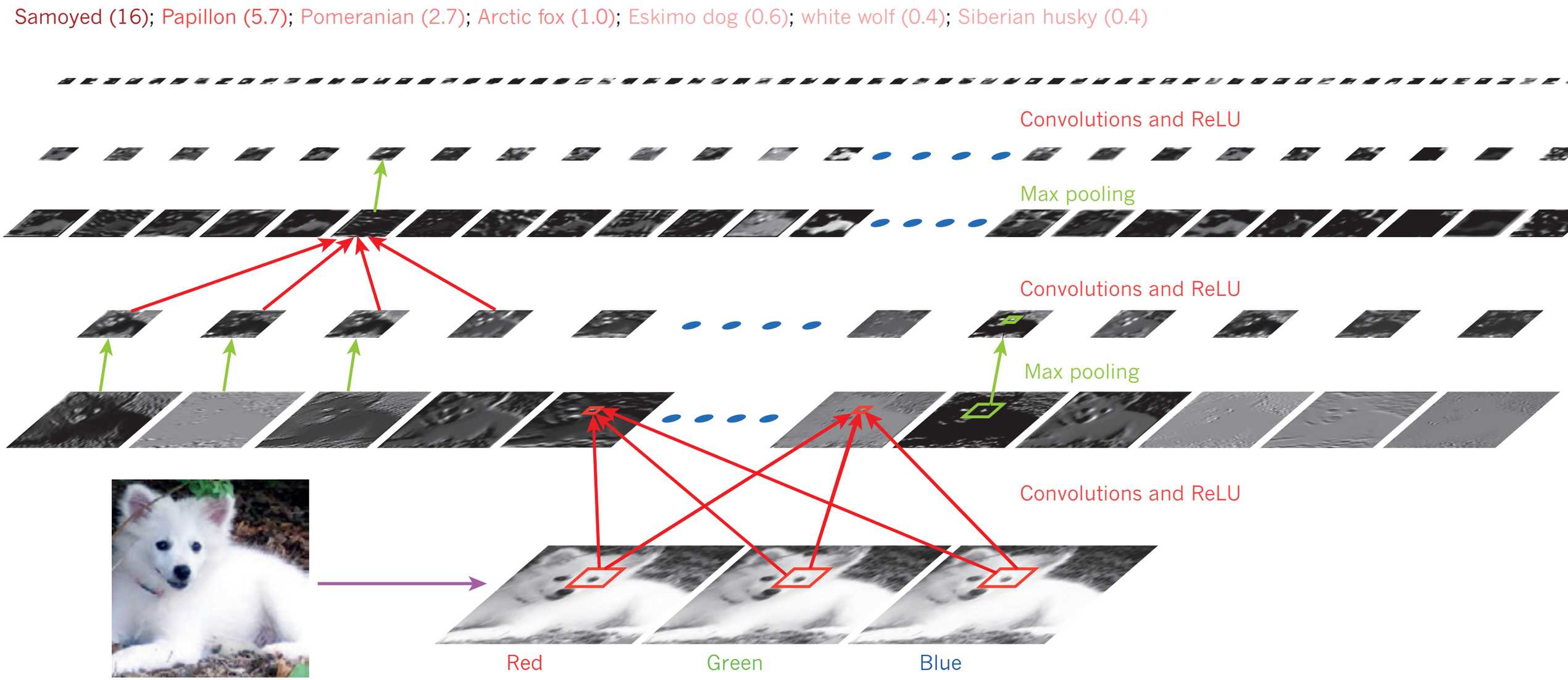
localized edge detector



invariance through pooling

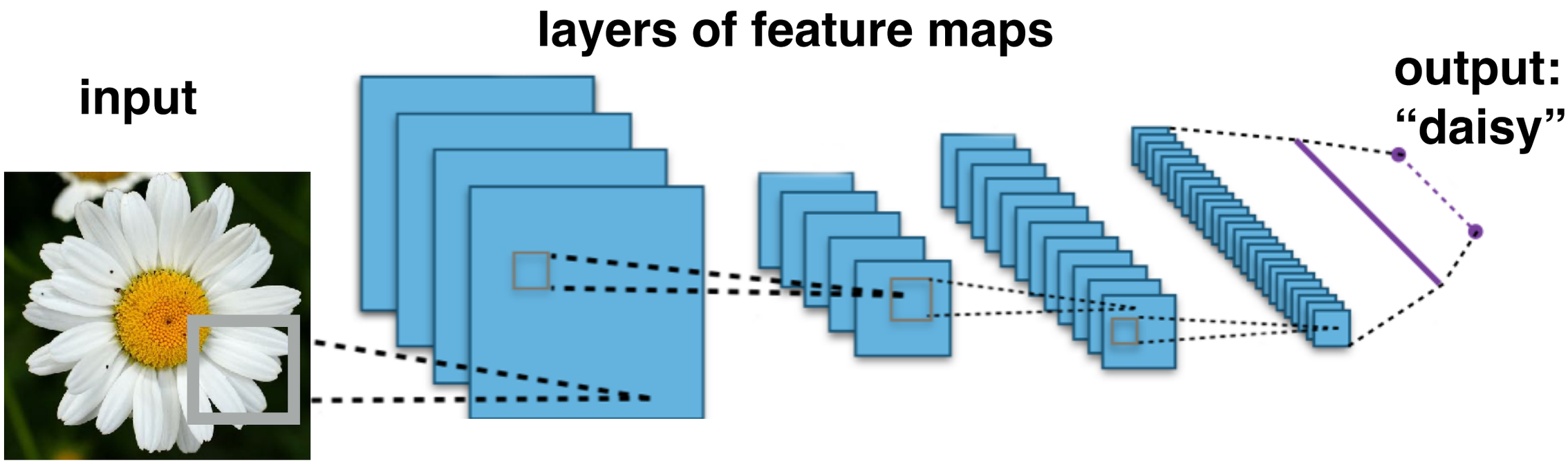


Deep convolutional neural network



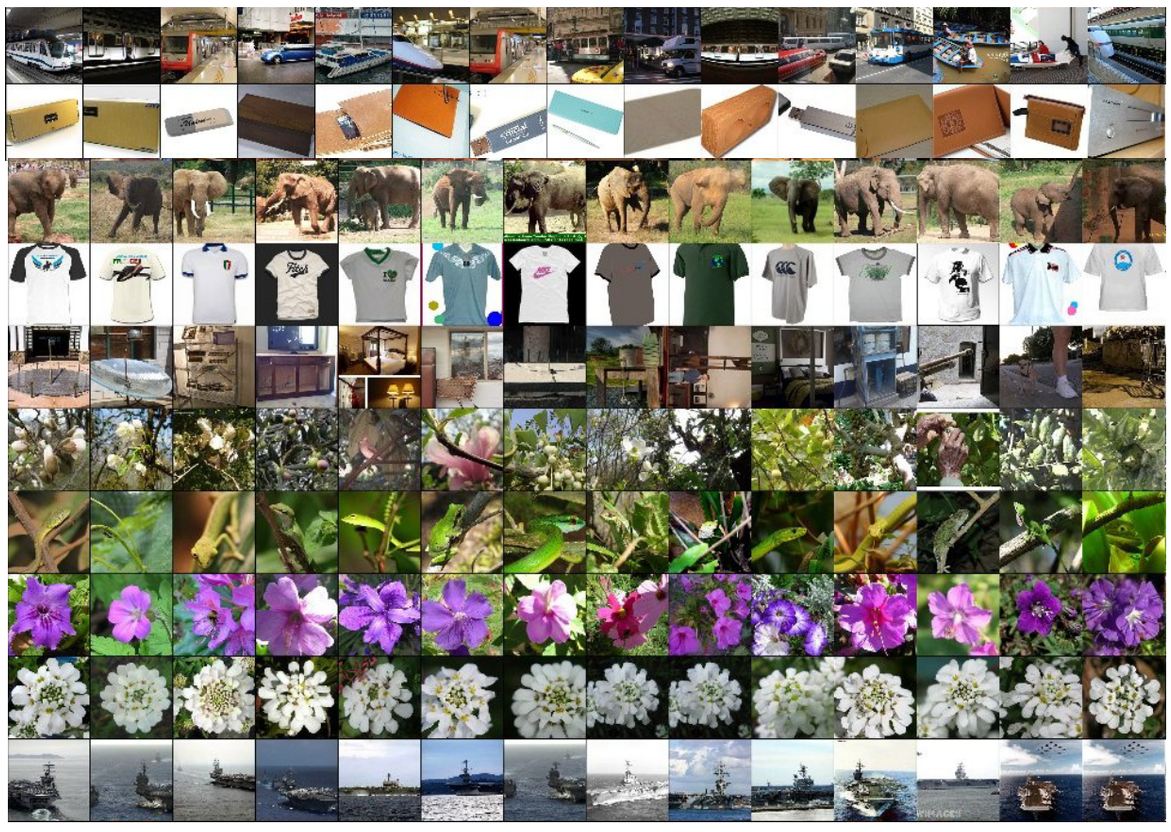
From LeCun, Bengio, & Hinton (2015).

How do we train it? Backpropagation



Training data (ImageNet)

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



<u>AlexNet</u>
8 layers
~60 million parameters

<u>GoogLeNet</u>
22 layers
~5 million parameters

<u>Residual networks</u>
hundreds of layers…

Deep convnets for understanding psychological and neural representations

- 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, J., Abbott, J., & Griffiths, T. (2016). Adapting Deep Network Features to Capture Psychological Representations.

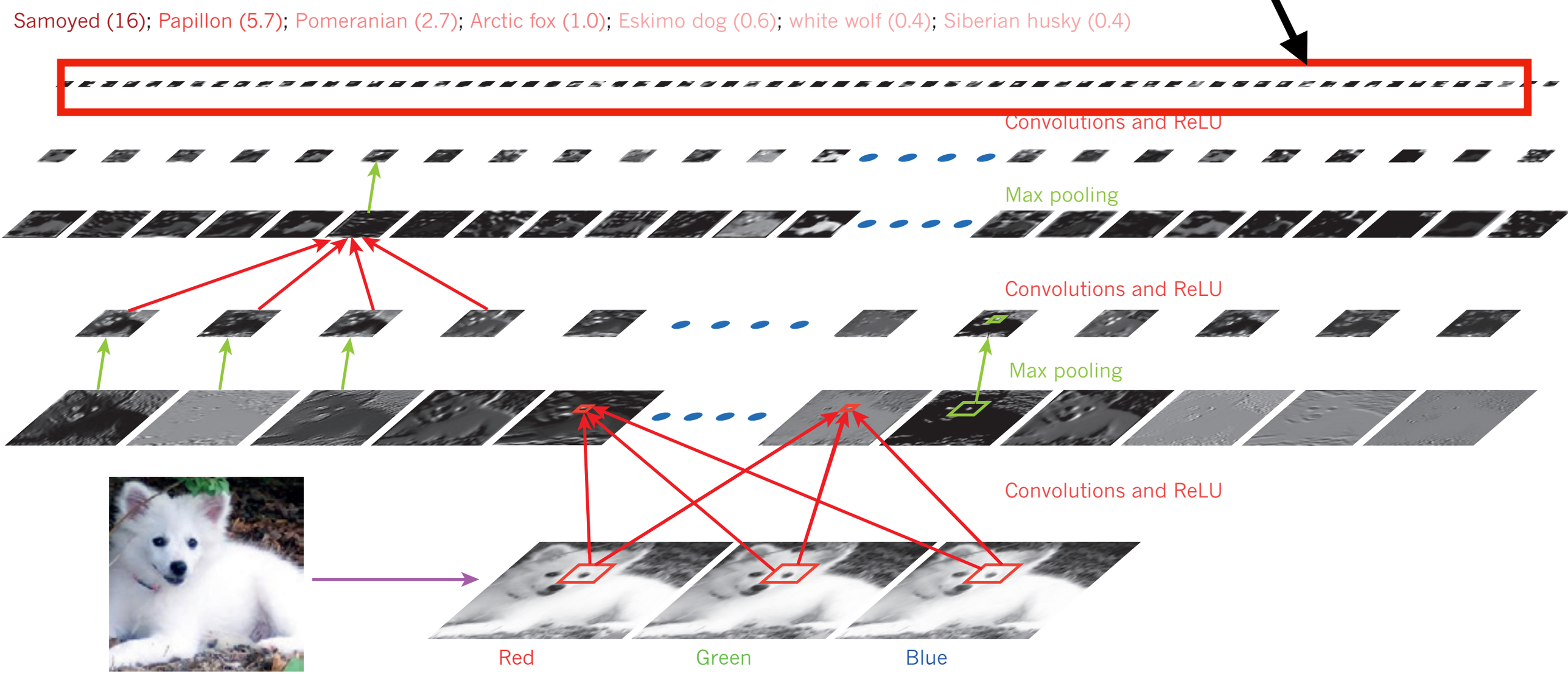
Some images look more similar to us than others. Can a neural network trained for classification help to explain similarity judgments from humans?

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

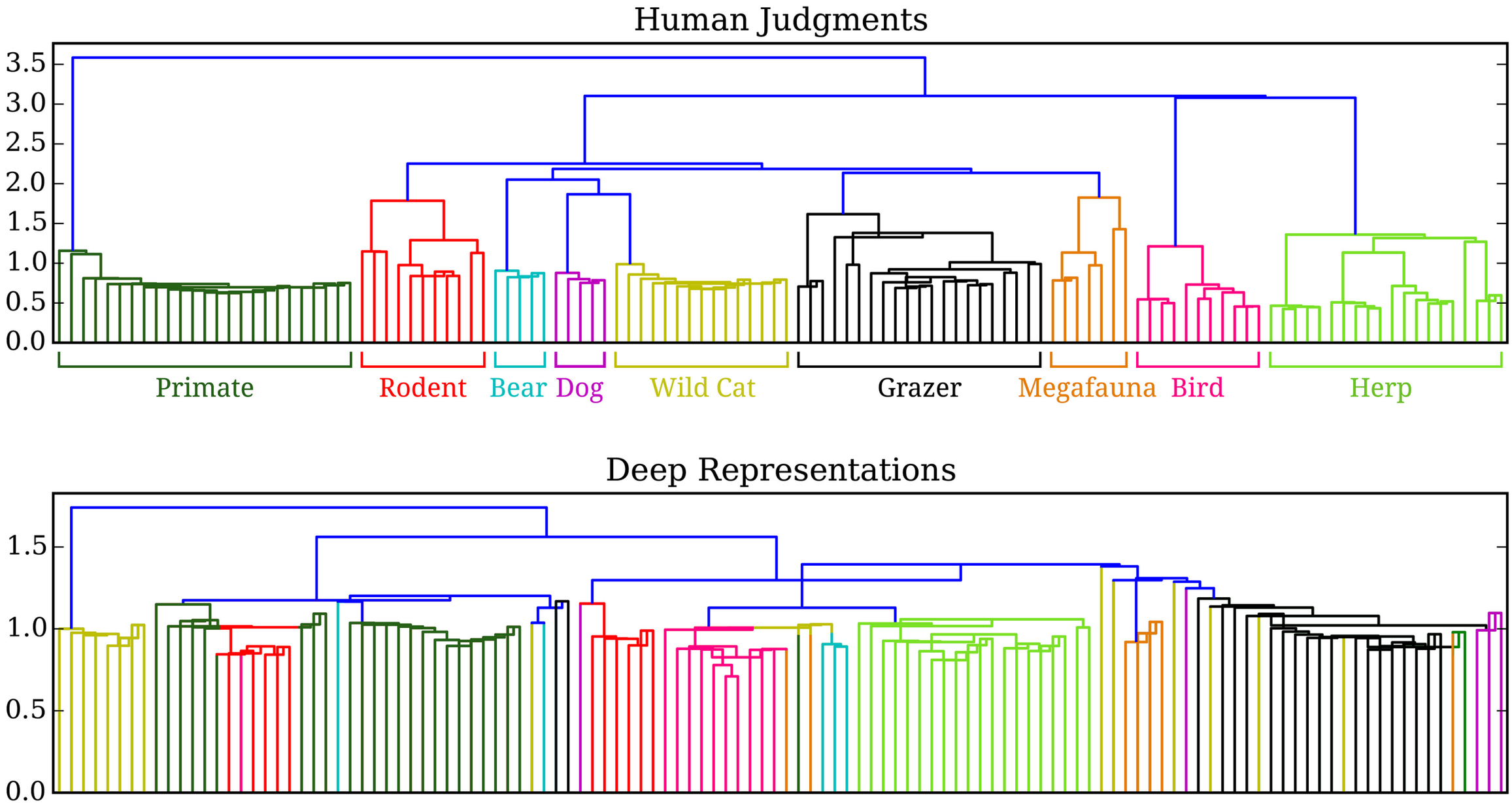
computing image-to-image similarity

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

summarizing images as high-level feature vector f

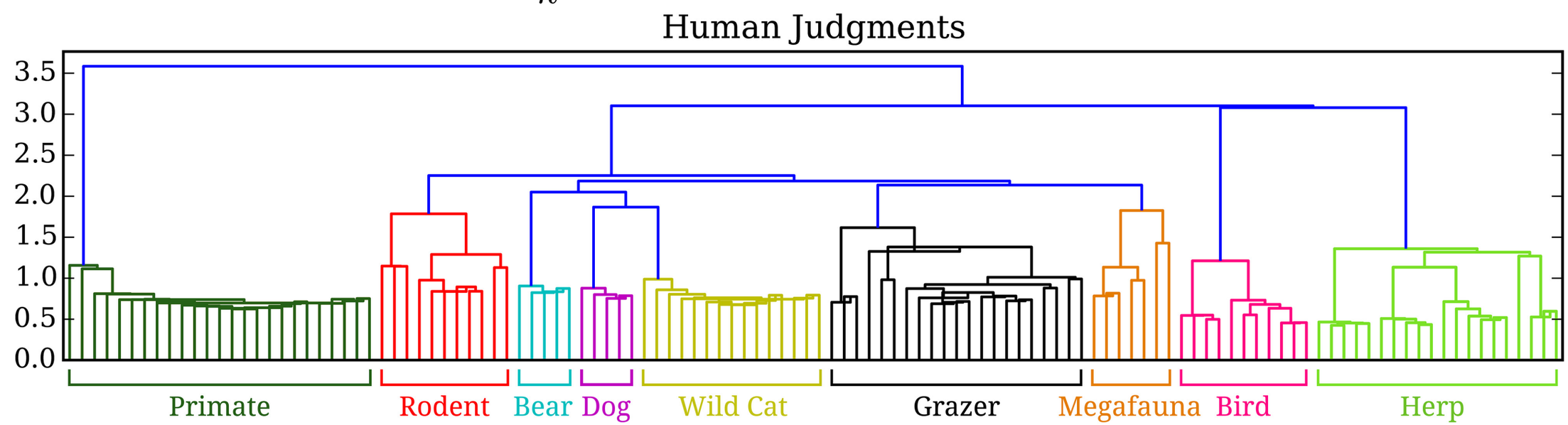


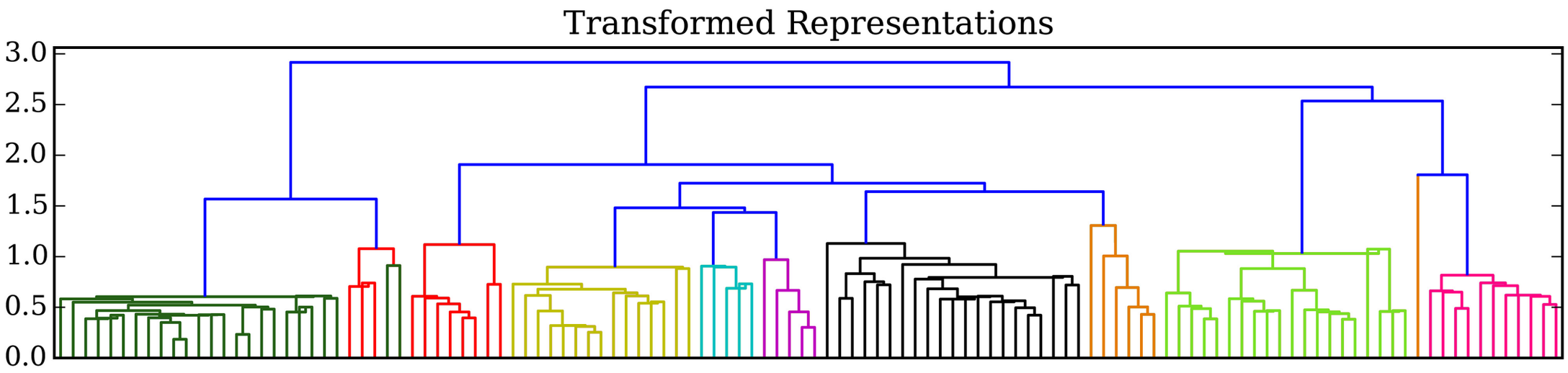
Hierarchical clustering reveals substantial differences in representation (best network explains about 40% of the variance in human judgments)



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)

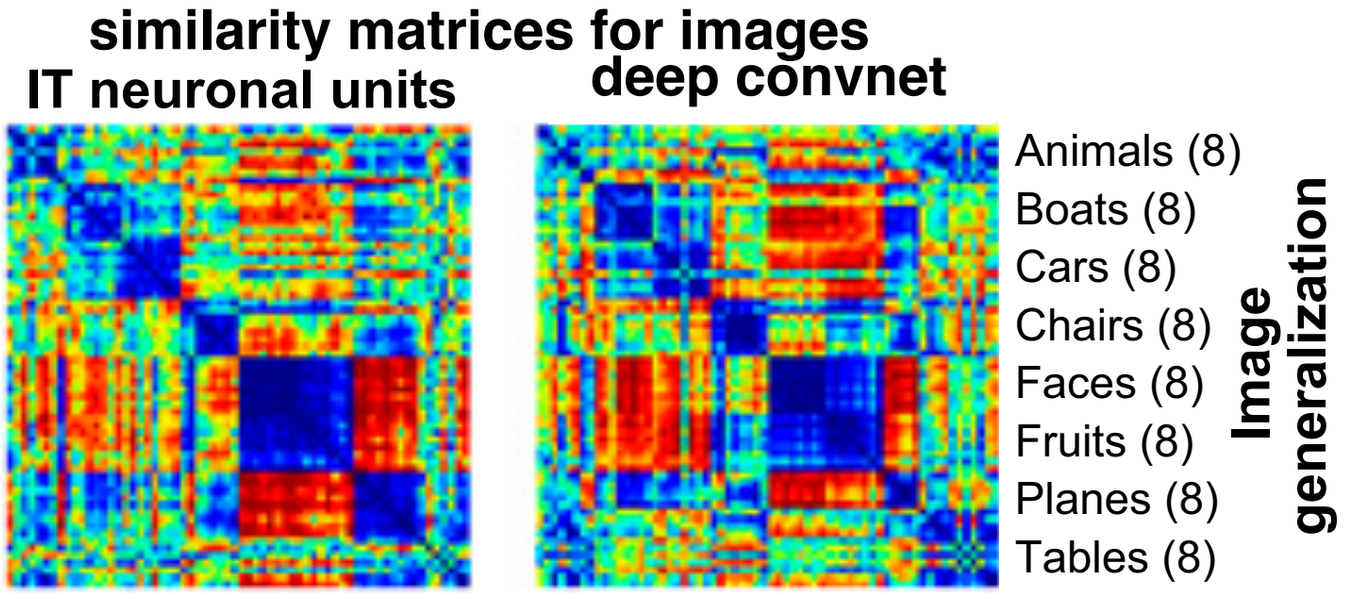
$$sim(i,j) = \sum_k w_k f_{ik} f_{jk} \qquad w_k : \text{weight for feature k}$$

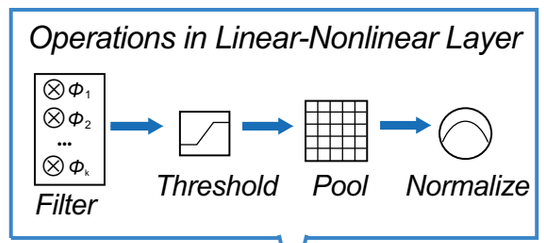




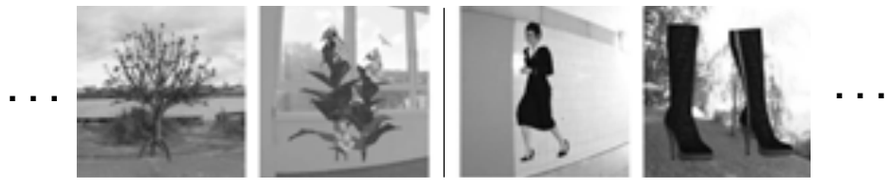
Predicting neural recordings with a deep convnet

(Yamins, D. L., Hong, H., Cadieu, C. F., Solomon, E. A., Seibert, D., & DiCarlo, J. J. (2014). Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, *111*(23), 8619-8624.)

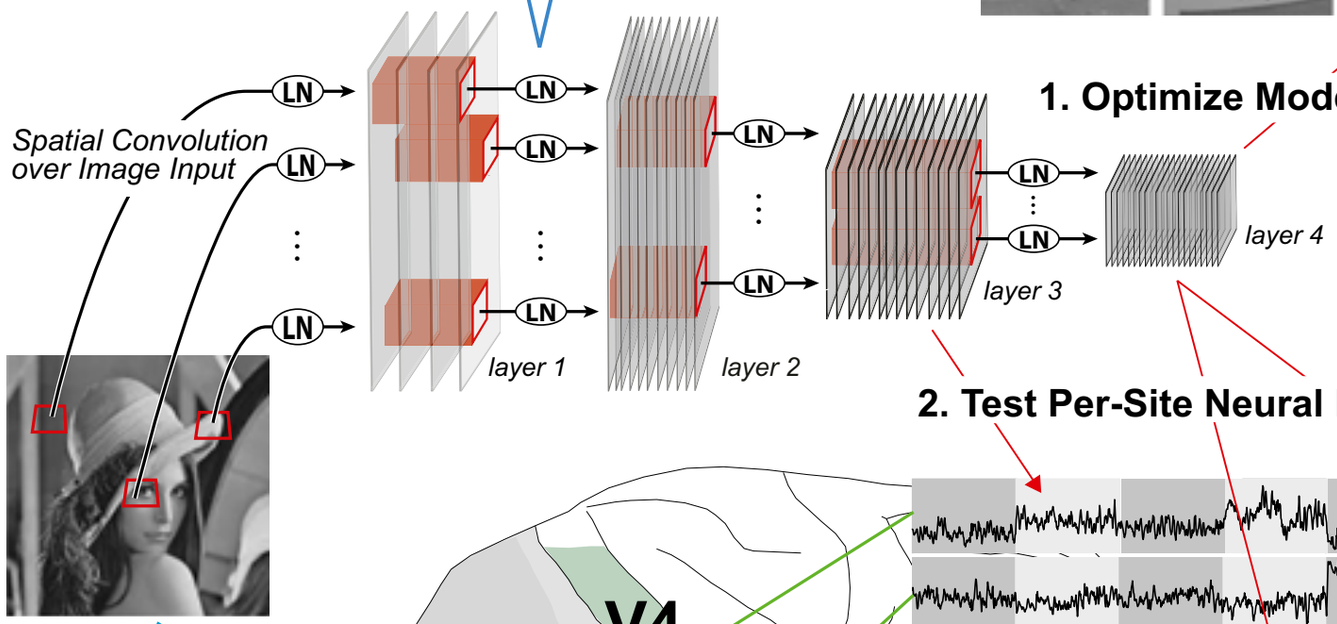




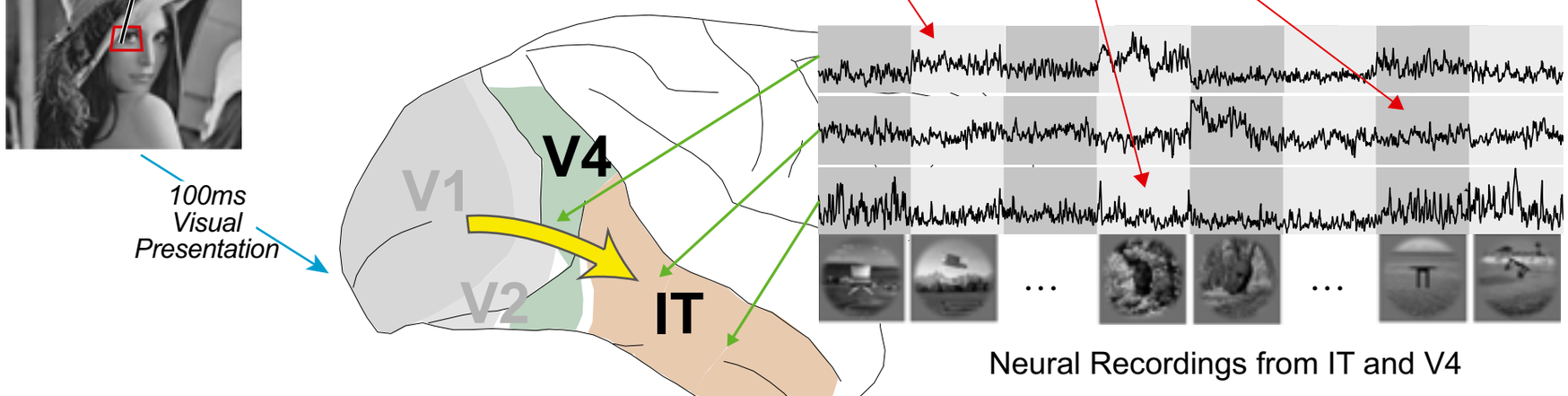
Behavioral Tasks
e.g. Trees vs non-Trees



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, B. M., Zaremba, W., Fergus, R., & Gureckis, T. M. (2015). Deep Neural Networks Predict Category Typicality Ratings for Images.

typical dog

atypical dog

• For people, typicality influences performance in practically any category-related task

- 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)

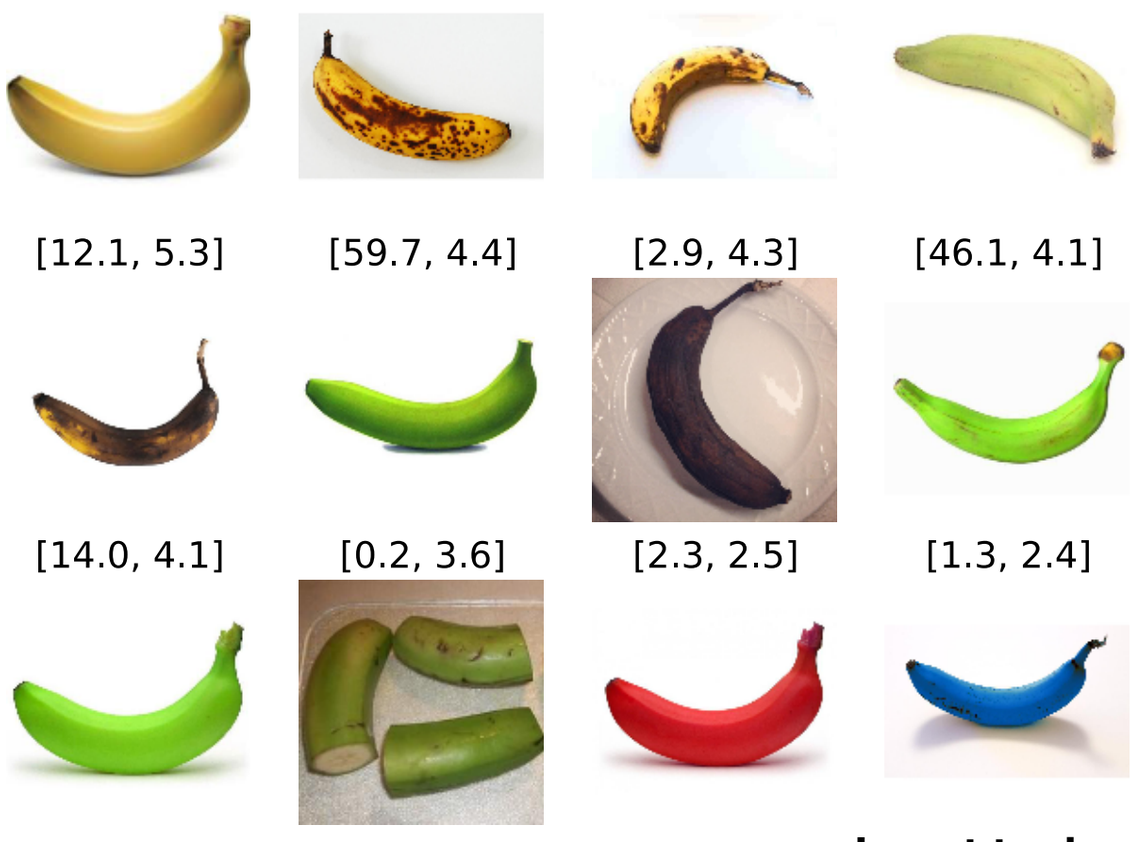
How well does this picture fit your idea or image of the category? (rated on 1-7 scale)

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]



Least typical

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

Category: Bathtub (ρ=0.68)

Human typicality ratings

Most typical →

Least typical

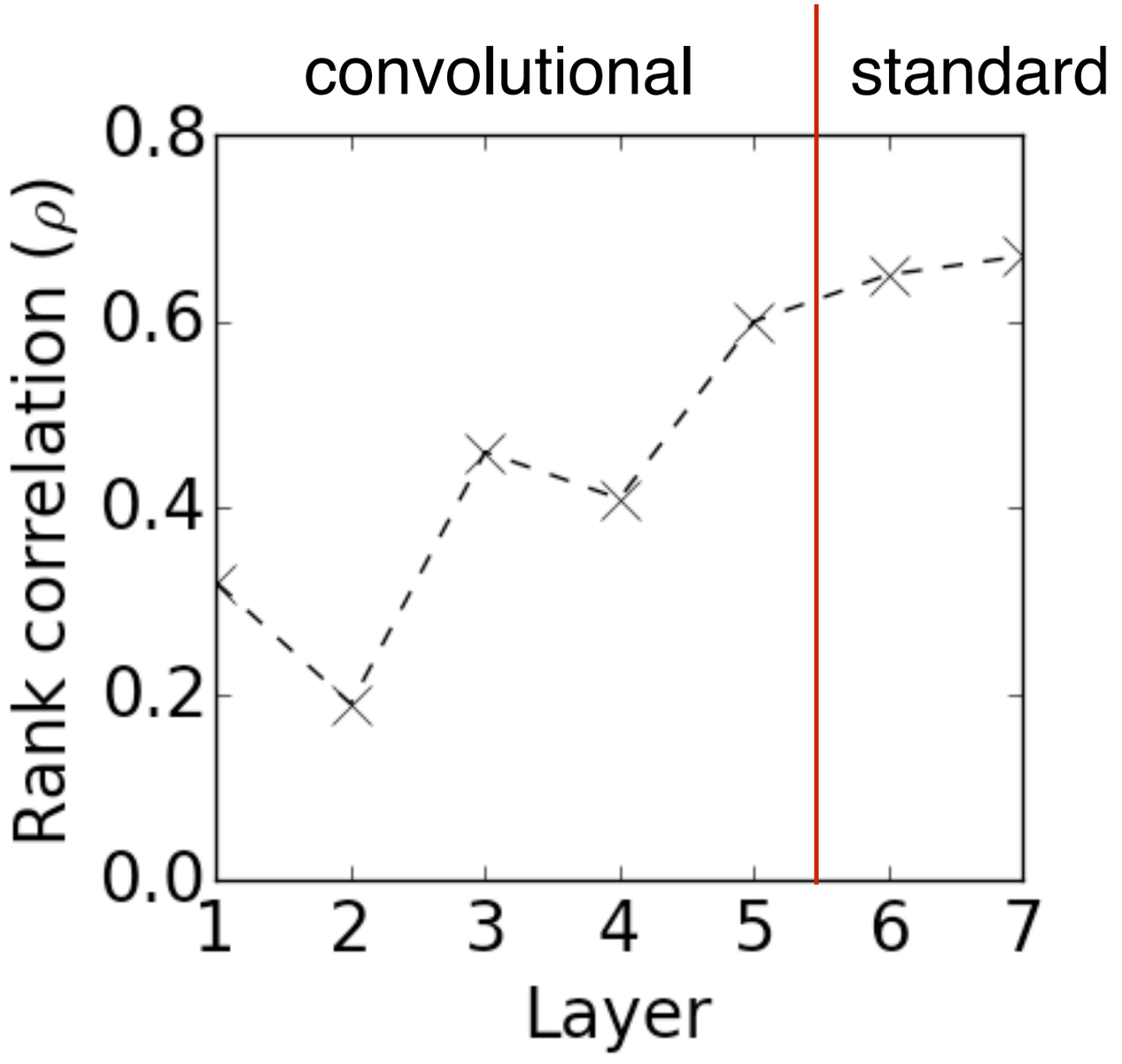
Convnet typicality ratings

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

Summary of typicality predictions

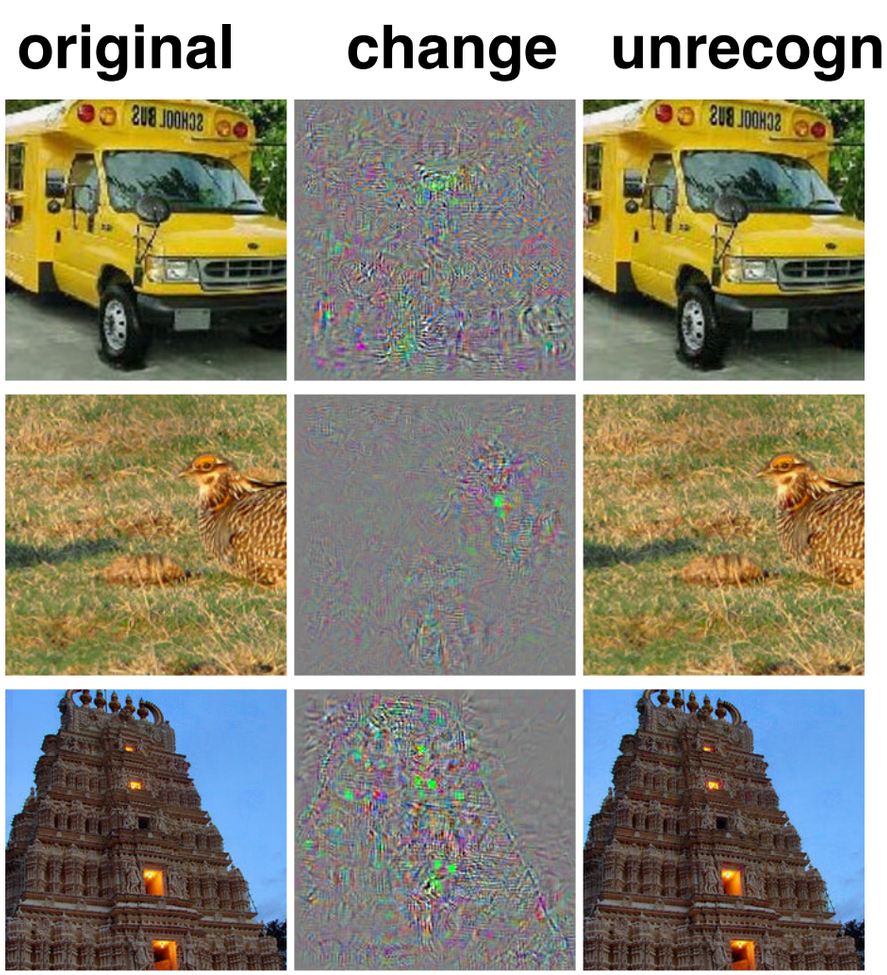
	Rank Correlation
Banana	0.82
Bathtub	0.68
Coffee Mug	0.62
Envelope	0.79
Pillow	0.67
Soap dispenser	0.74
Table lamp	0.69
Teapot	0.38
Average	**0.67**

Prediction quality varies as a function of network depth.

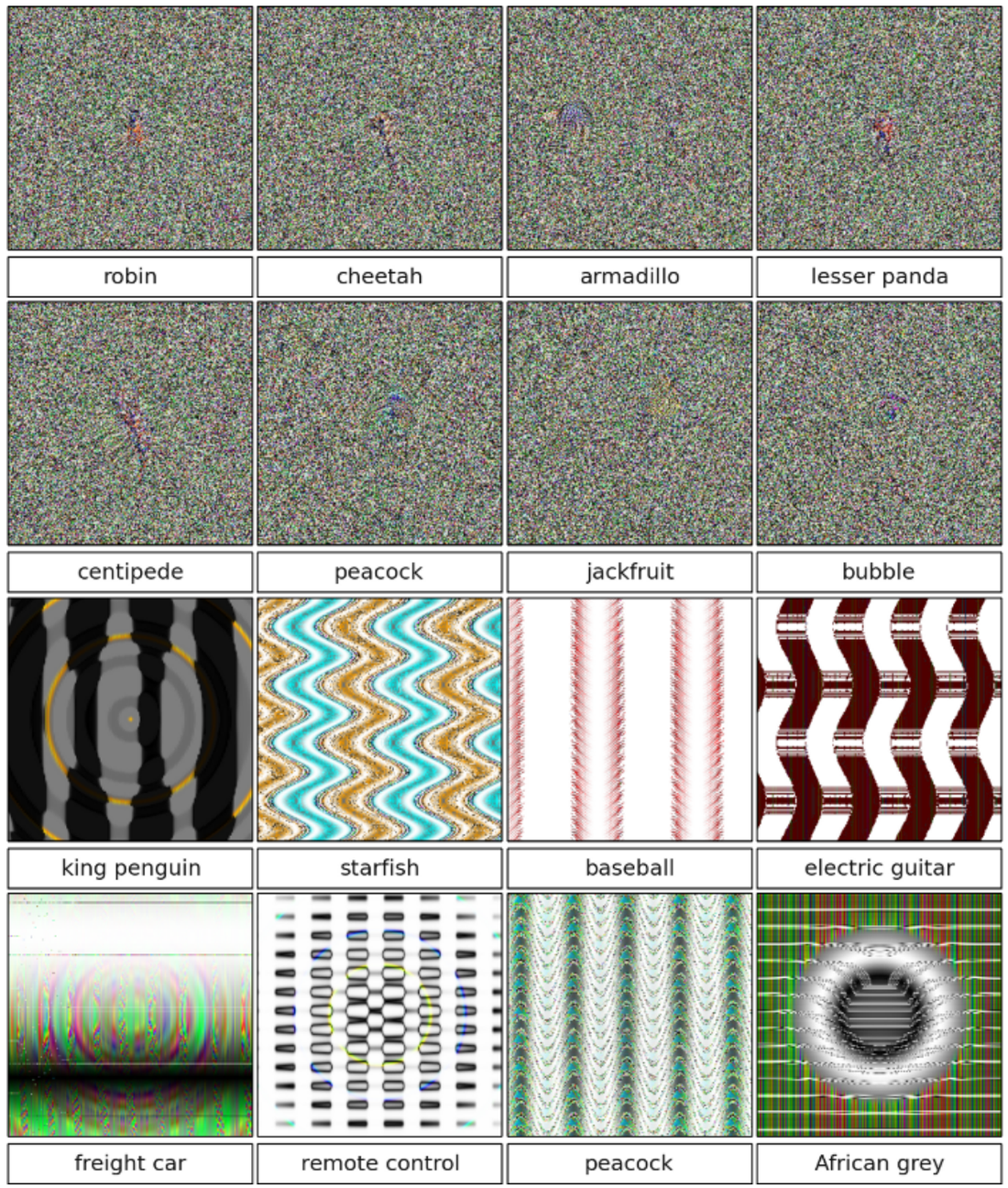


Critiques of deep convolutional networks

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

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

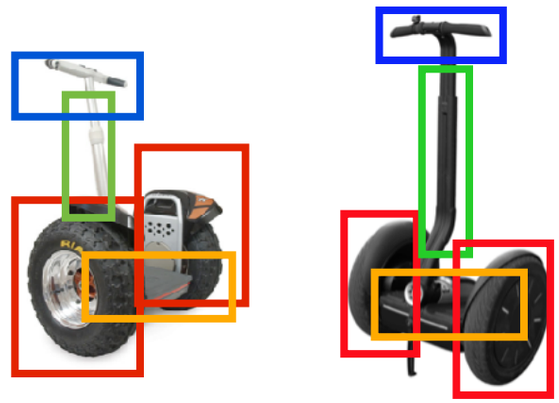
People learn from less data

"one-shot learning"

where are the others?

People learn richer concepts

parsing



generating new concepts

generating new examples