

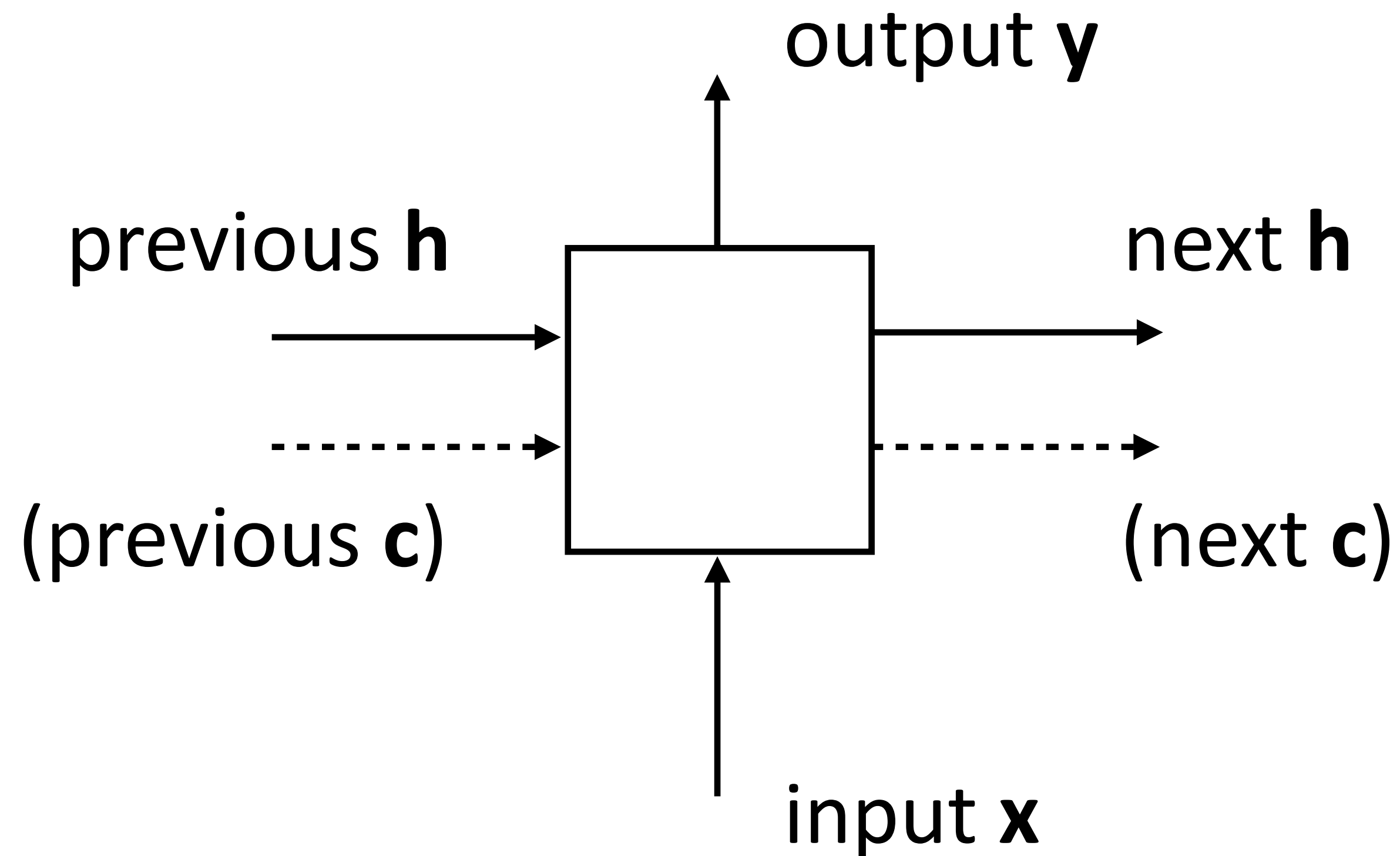
CNNs and Neural CRFs

Wei Xu

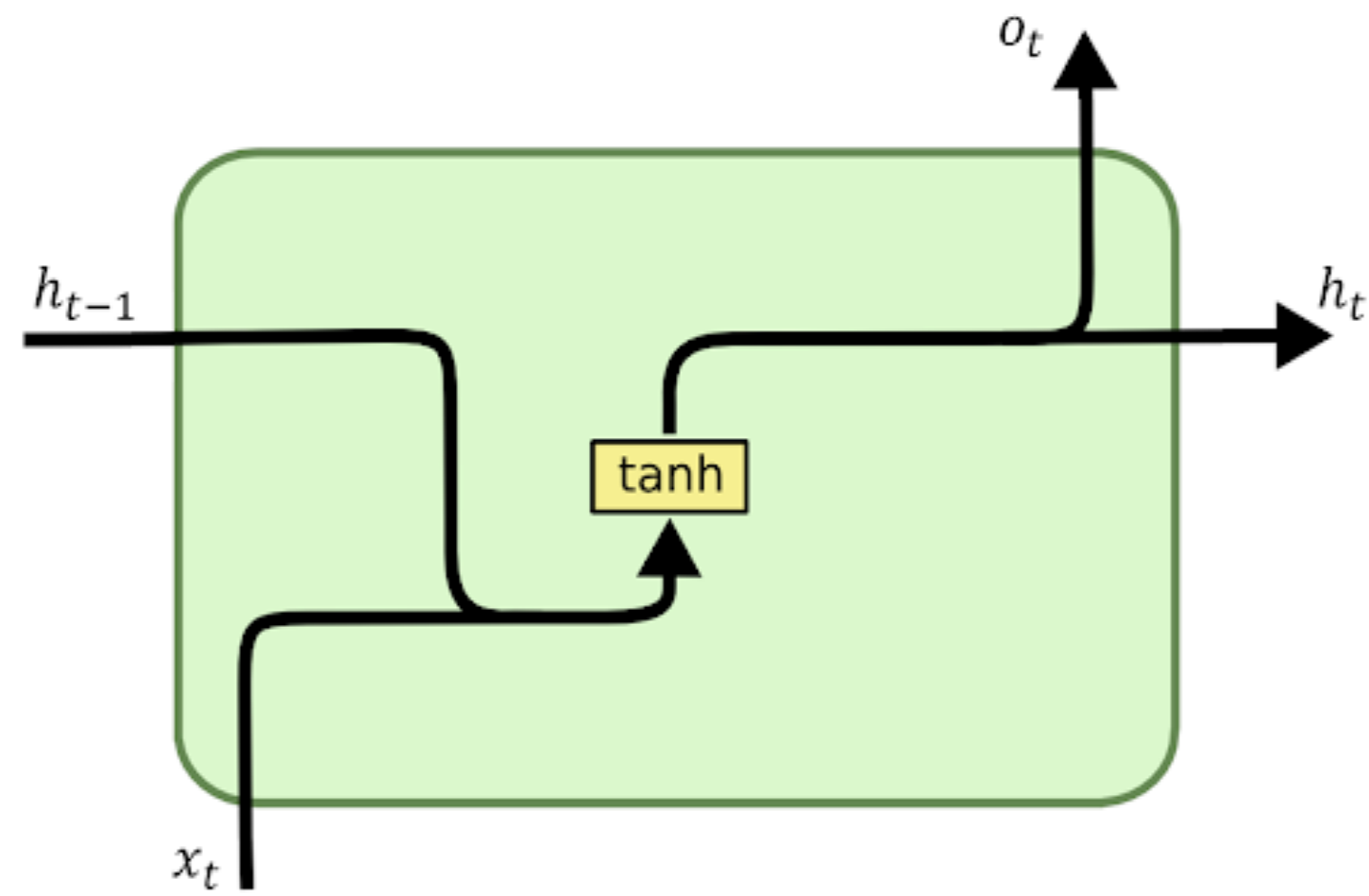
(many slides from Greg Durrett, Stanford 231n)

Recall: RNNs

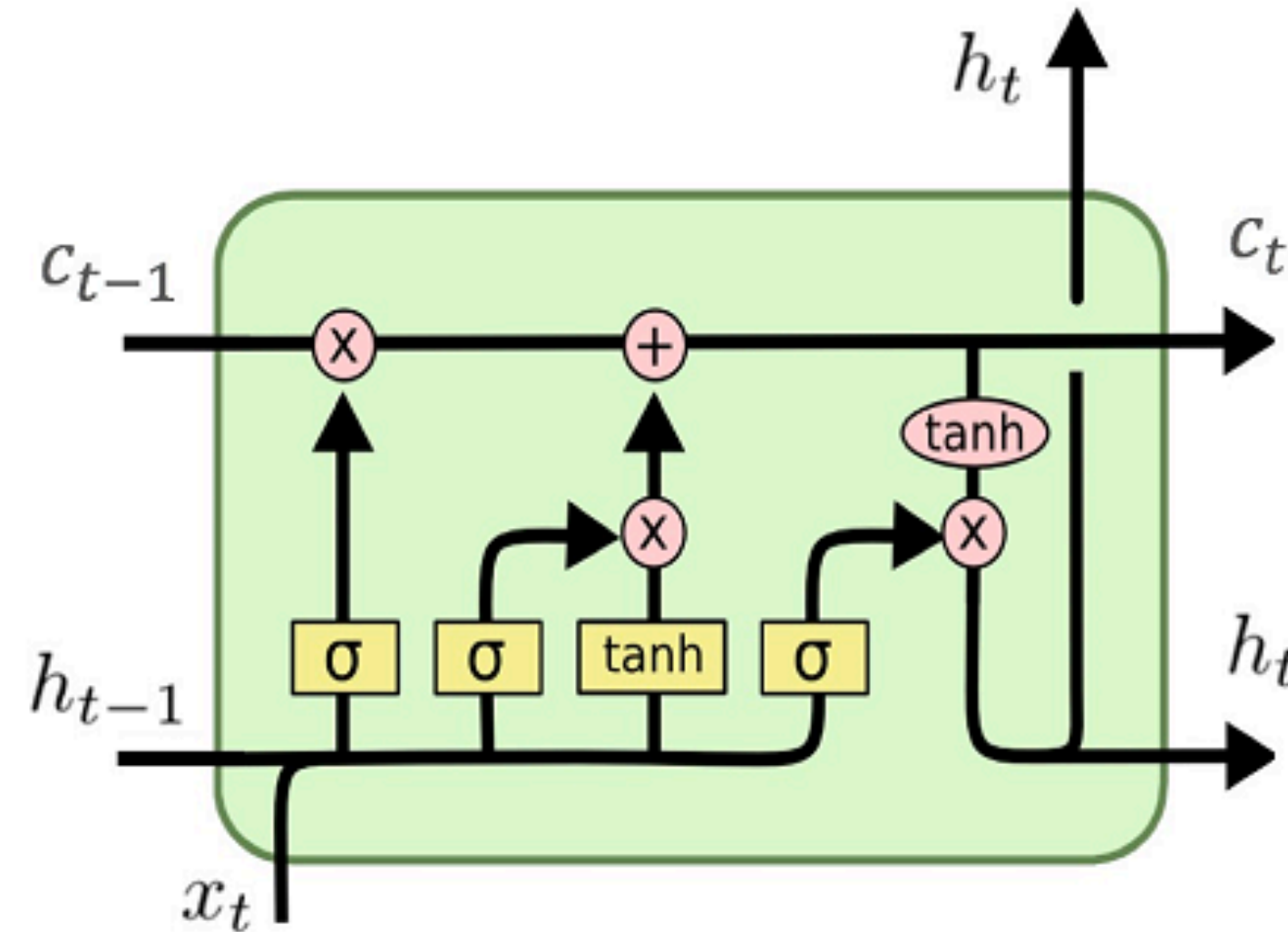
- ▶ Cell that takes some input \mathbf{x} , has some hidden state \mathbf{h} , and updates that hidden state and produces output \mathbf{y} (all vector-valued)



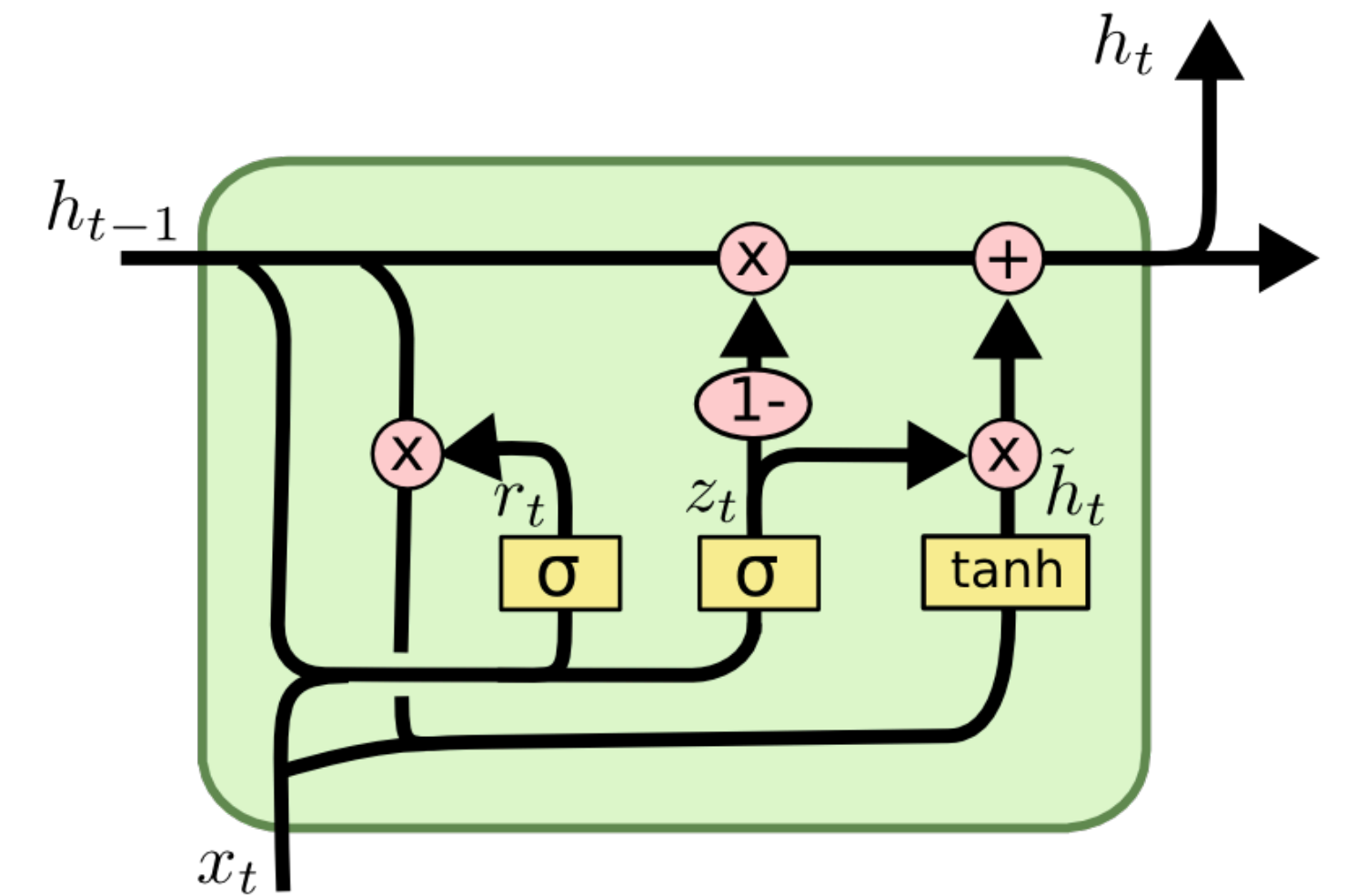
Recall: LSTM & GRU



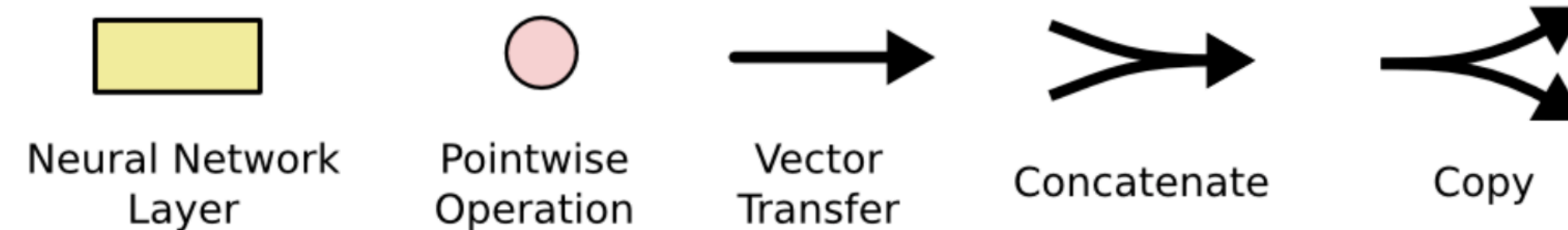
Standard RNN
(with a single layer)



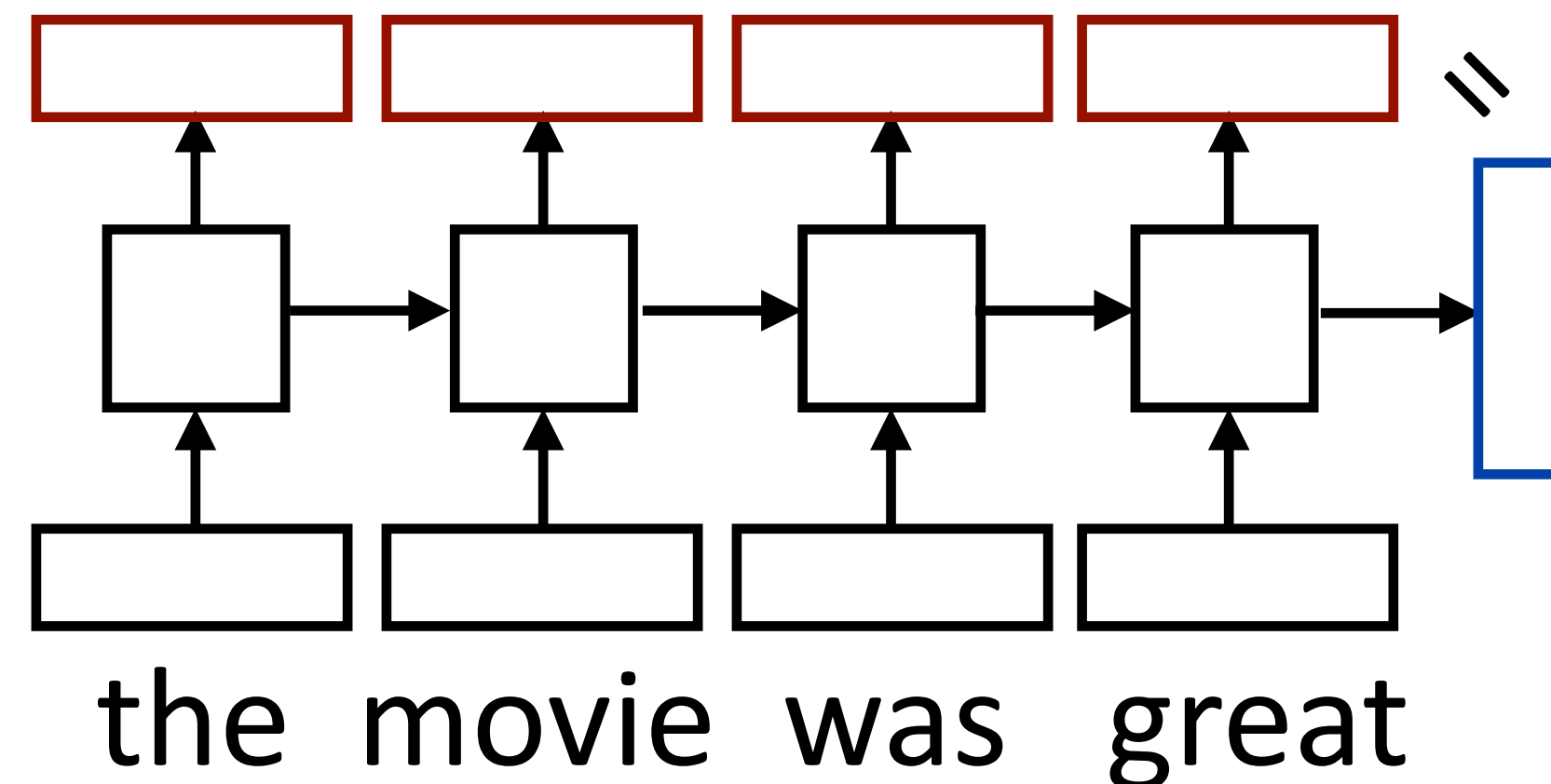
LSTM
(Long Short-term Memory)



GRU
(Gated Recurrent Unit)



Recall: RNN Abstraction



- ▶ **Encoding of the sentence** — can pass this a decoder or make a classification decision about the sentence
- ▶ **Encoding of each word** — can pass this to another layer to make a prediction (can also pool these to get a different sentence encoding)
- ▶ RNN can be viewed as a transformation of a sequence of vectors into a sequence of context-dependent vectors

This Lecture

- ▶ CNNs
- ▶ CNNs for Sentiment, Entity Linking
- ▶ Neural CRFs

Administrivia

- ▶ Reading — Goldberg 9 (CNN); Eisenstein 3.4, 7.6

A Primer on Neural Network Models for Natural Language Processing

Yoav Goldberg
Draft as of October 5, 2015.

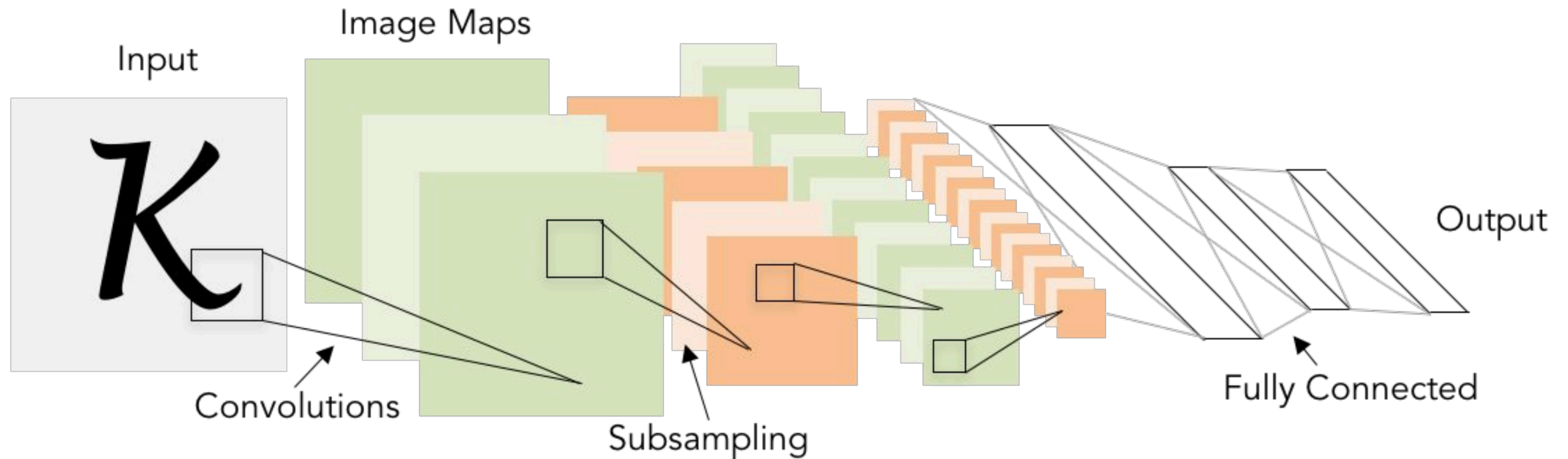
The most up-to-date version of this manuscript is available at <http://www.cs.biu.ac.il/~yogo/nlp.pdf>. Major updates will be published on arxiv periodically. I welcome any comments you may have regarding the content and presentation. If you spot a missing reference or have relevant work you'd like to see mentioned, do let me know.
first.last@gmail

Abstract

Over the past few years, neural networks have re-emerged as powerful machine-learning models, yielding state-of-the-art results in fields such as image recognition and speech processing. More recently, neural network models started to be applied also to textual natural language signals, again with very promising results. This tutorial surveys neural network models from the perspective of natural language processing research, in an attempt to bring natural-language researchers up to speed with the neural techniques. The tutorial covers input encoding for natural language tasks, feed-forward networks, convolutional networks, recurrent networks and recursive networks, as well as the computation graph abstraction for automatic gradient computation.

CNNs

Convolutional Neural Networks



LeCun et al. (1998)

A Bit of History

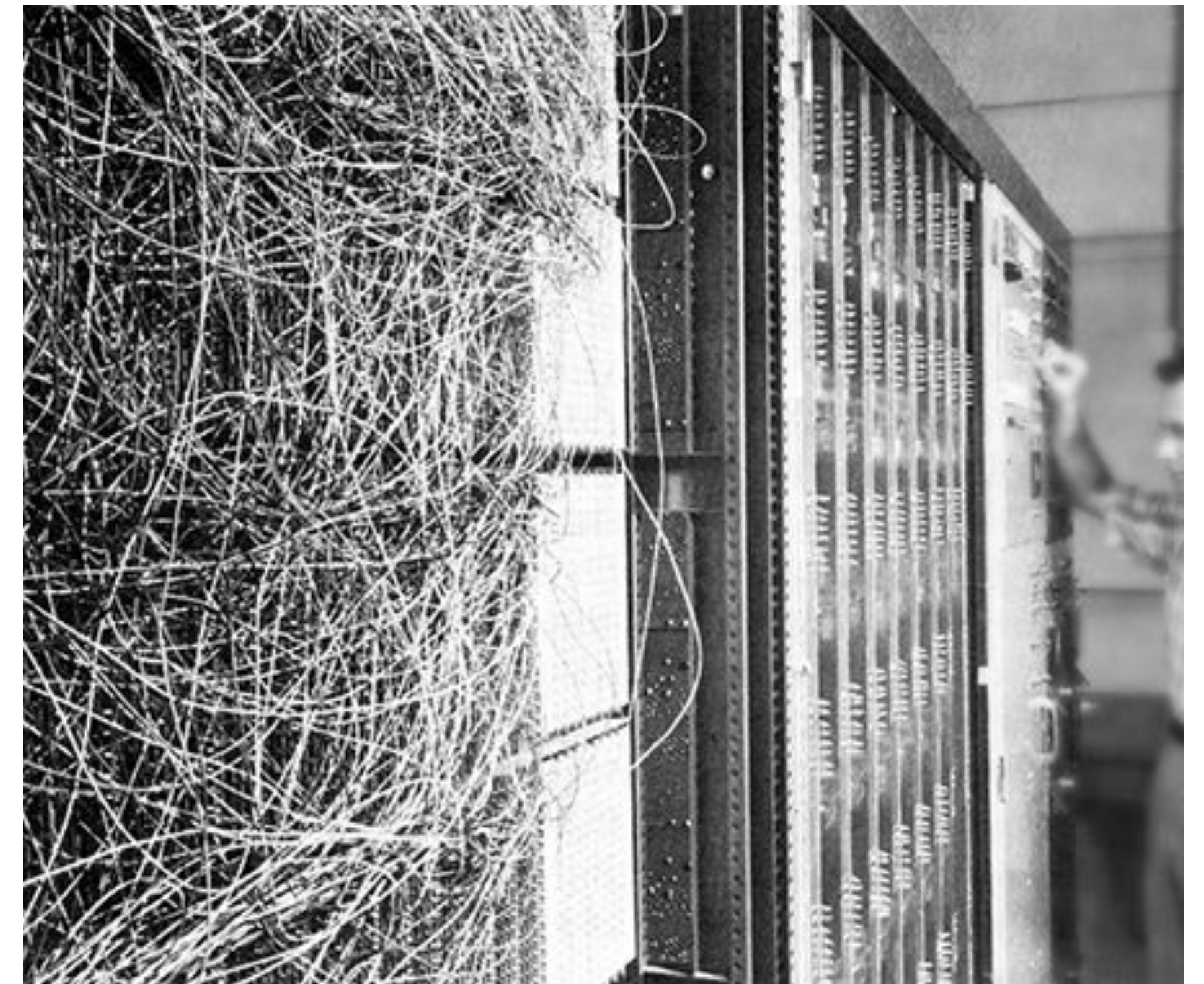
- ▶ The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
- ▶ Perceptron (Frank Rosenblatt, 1957)
- ▶ Artificial Neuron (McCulloch & Pitts, 1943)

McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad \text{if } \sum_{i=0}^n x_i \geq 0$$
$$= 0 \quad \text{if } \sum_{i=0}^n x_i < 0$$

Perceptron

$$y = 1 \quad \text{if } \sum_{i=0}^n w_i * x_i \geq 0$$
$$= 0 \quad \text{if } \sum_{i=0}^n w_i * x_i < 0$$

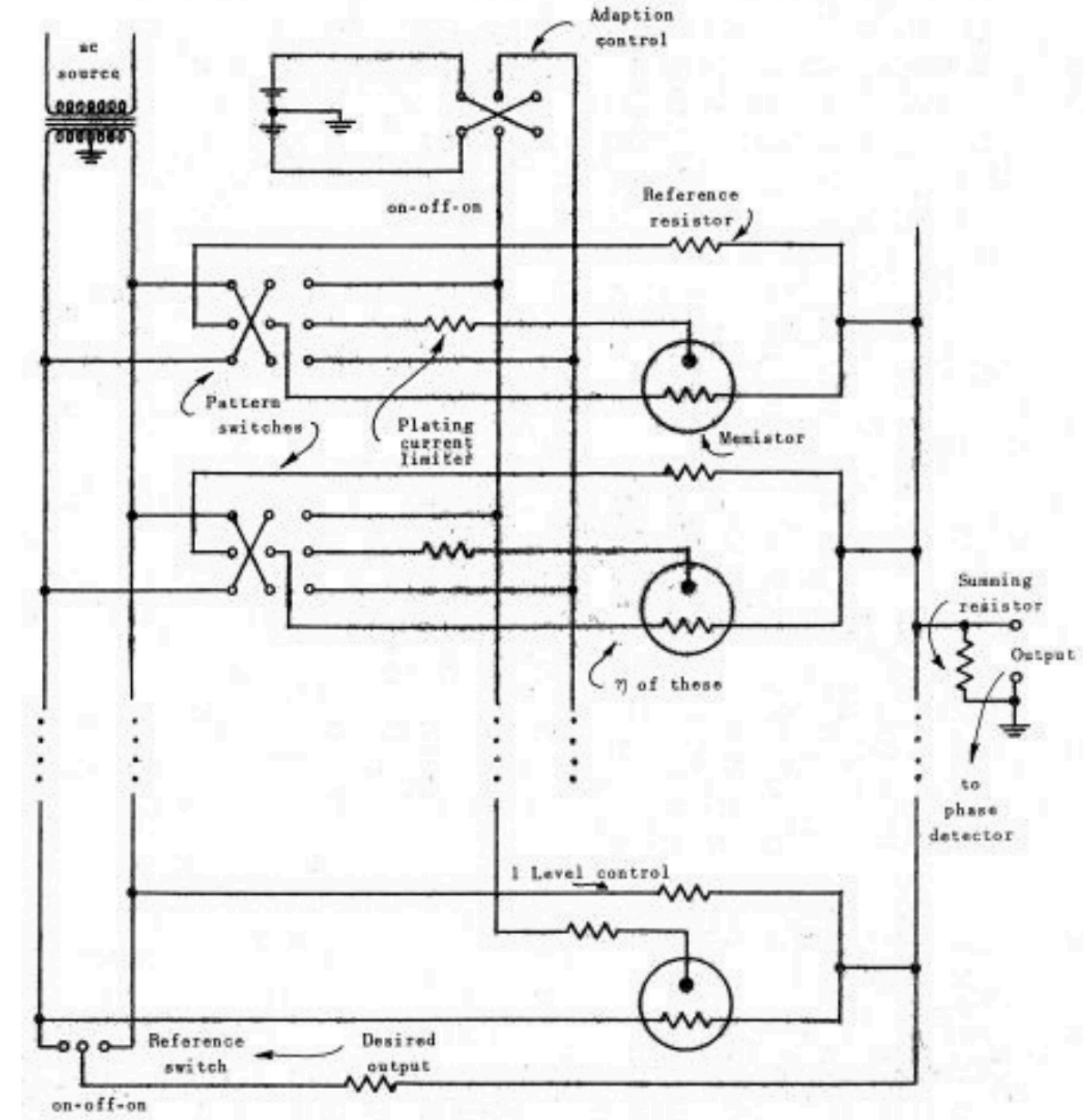
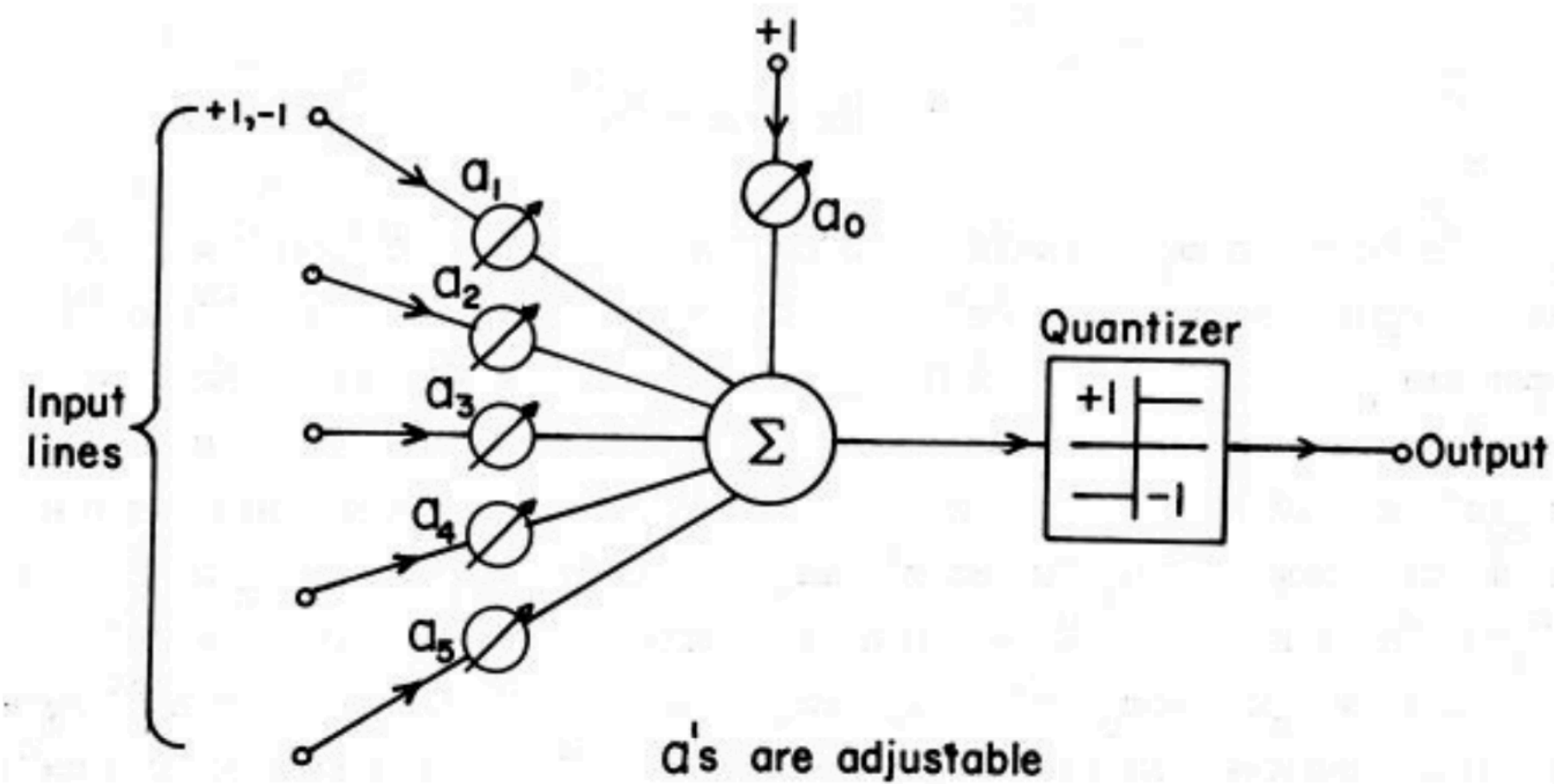


The IBM Automatic Sequence Controlled Calculator, called Mark I by Harvard University's staff. It was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.

https://www.youtube.com/watch?time_continue=71&v=cNxadbrN_al&feature=emb_logo

A Bit of History

- ▶ Adaline/Madeline - single and multi-layer “artificial neurons” (Widrow and Hoff, 1960)



A Bit of History

- ▶ First time back-propagation became popular (Rumelhart et al, 1986)

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors². Learning becomes more interesting but

more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j , to unit j is a linear function of the outputs, y_i , of the units that are connected to j and of the weights, w_{ji} , on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j , which is a non-linear function of its total input

$$y_j = \frac{1}{1 + e^{-x_j}} \quad (2)$$

† To whom correspondence should be addressed.

A Bit of History

- ▶ First time back-propagation became popular (Rumelhart et al, 1986)

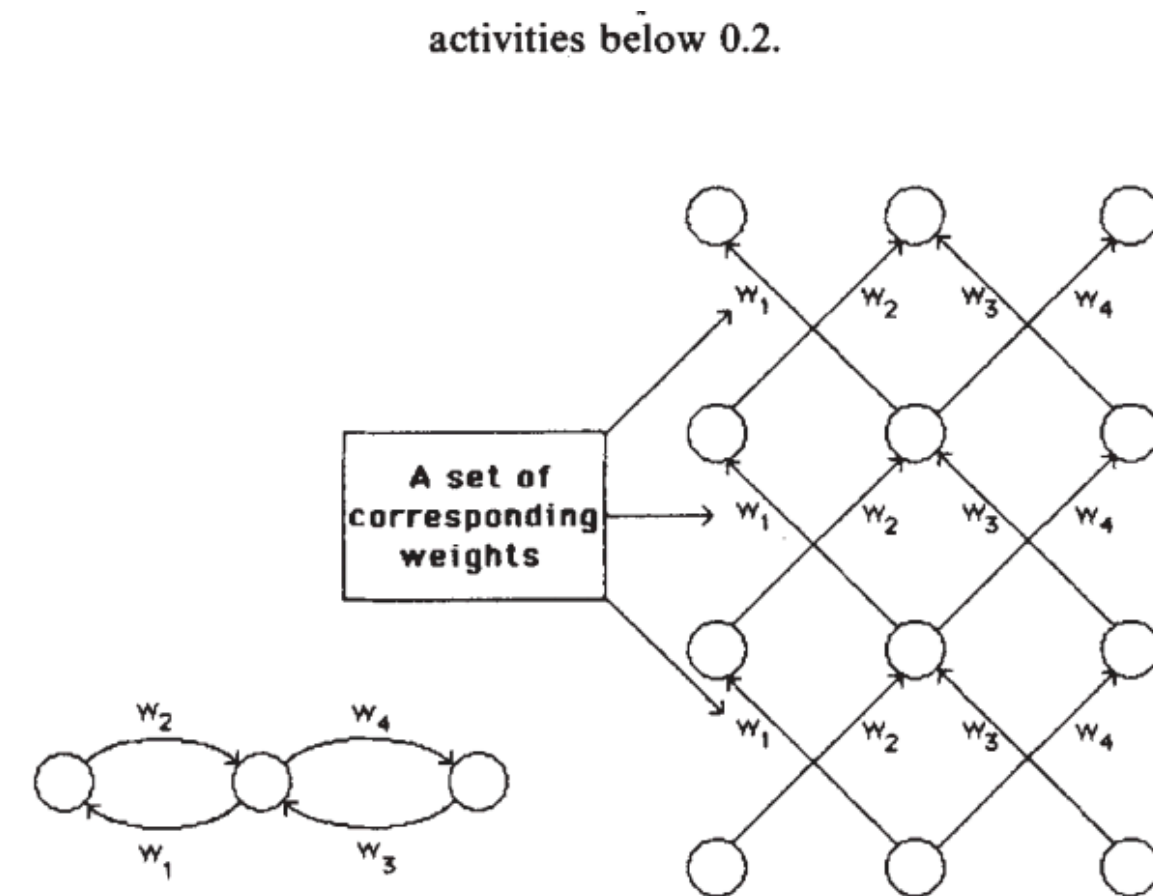


Fig. 5 A synchronous iterative net that is run for three iterations and the equivalent layered net. Each time-step in the recurrent net corresponds to a layer in the layered net. The learning procedure for layered nets can be mapped into a learning procedure for iterative nets. Two complications arise in performing this mapping: first, in a layered net the output levels of the units in the intermediate layers during the forward pass are required for performing the backward pass (see equations (5) and (6)). So in an iterative net it is necessary to store the history of output states of each unit. Second, for a layered net to be equivalent to an iterative net, corresponding weights between different layers must have the same value. To preserve this property, we average $\partial E/\partial w$ for all the weights in each set of corresponding weights and then change each weight in the set by an amount proportional to this average gradient. With these two provisos, the learning procedure can be applied directly to iterative nets. These nets can then either learn to perform iterative searches or learn sequential structures⁴.

To break symmetry we start with small random weights. Variants on the learning procedure have been discovered independently by David Parker (personal communication) and by Yann Le Cun³.

One simple task that cannot be done by just connecting the input units to the output units is the detection of symmetry. To detect whether the binary activity levels of a one-dimensional array of input units are symmetrical about the centre point, it is essential to use an intermediate layer because the activity in an individual input unit, considered alone, provides no evidence about the symmetry or non-symmetry of the whole input vector, so simply adding up the evidence from the individual input units is insufficient. (A more formal proof that intermediate units are required is given in ref. 2.) The learning procedure discovered an elegant solution using just two intermediate units, as shown in Fig. 1.

Another interesting task is to store the information in the two family trees (Fig. 2). Figure 3 shows the network we used, and Fig. 4 shows the 'receptive fields' of some of the hidden units after the network was trained on 100 of the 104 possible triples.

So far, we have only dealt with layered, feed-forward networks. The equivalence between layered networks and recurrent networks that are run iteratively is shown in Fig. 5.

The most obvious drawback of the learning procedure is that the error-surface may contain local minima so that gradient descent is not guaranteed to find a global minimum. However, experience with many tasks shows that the network very rarely gets stuck in poor local minima that are significantly worse than the global minimum. We have only encountered this undesirable behaviour in networks that have just enough connections to perform the task. Adding a few more connections creates extra dimensions in weight-space and these dimensions provide paths around the barriers that create poor local minima in the lower dimensional subspaces.

A Bit of History

- ▶ Long Short-term Memory (Hochreiter & Schmidhuber, 1997)

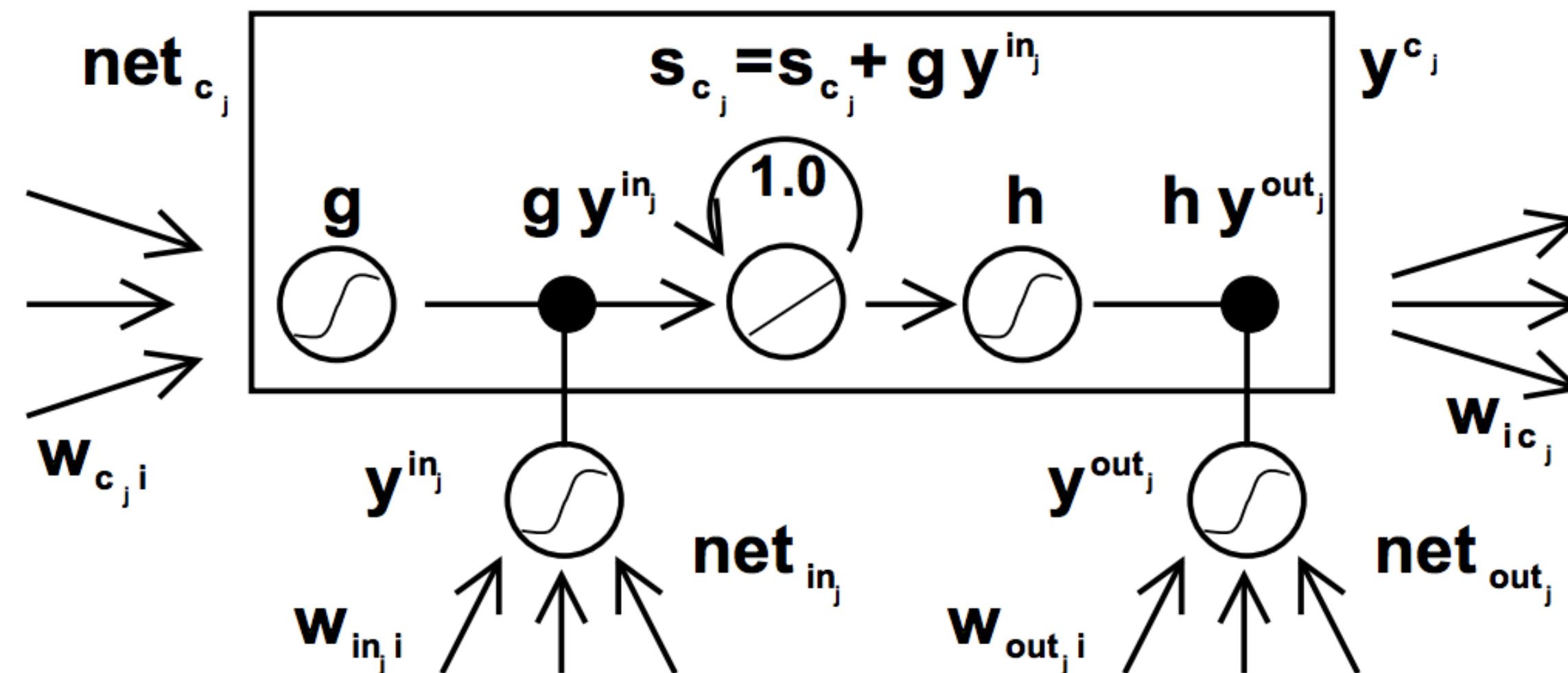
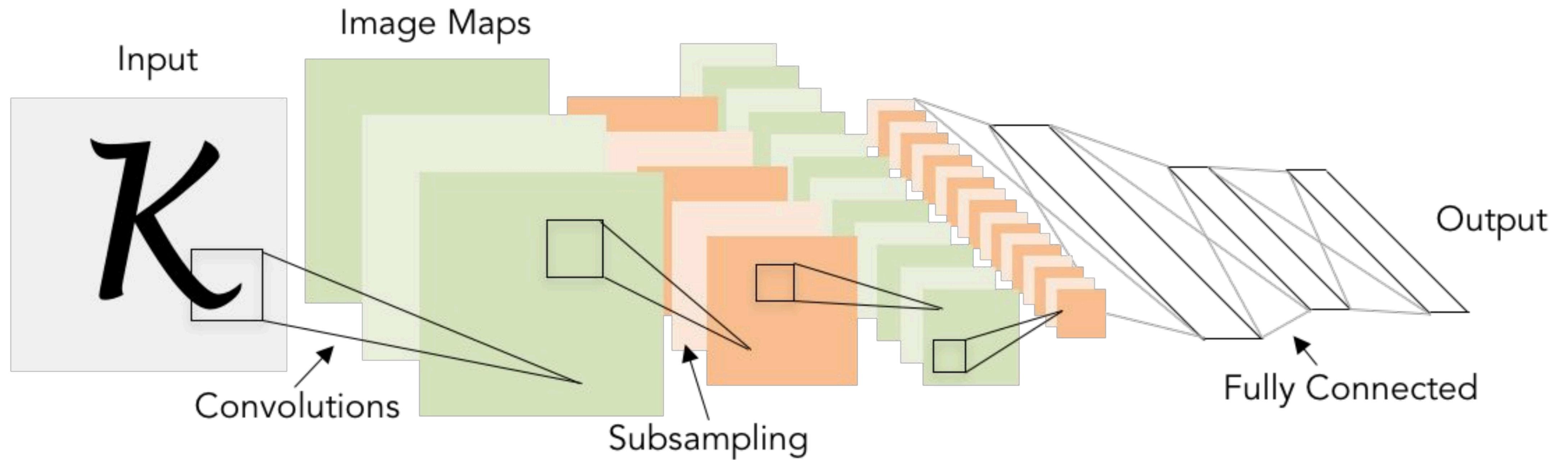


Figure 1: Architecture of memory cell c_j (the box) and its gate units in_j, out_j . The self-recurrent connection (with weight 1.0) indicates feedback with a delay of 1 time step. It builds the basis of the “constant error carousel” CEC. The gate units open and close access to CEC. See text and appendix A.1 for details.

A Bit of History



LeCun et al. (1998)

A Bit of History

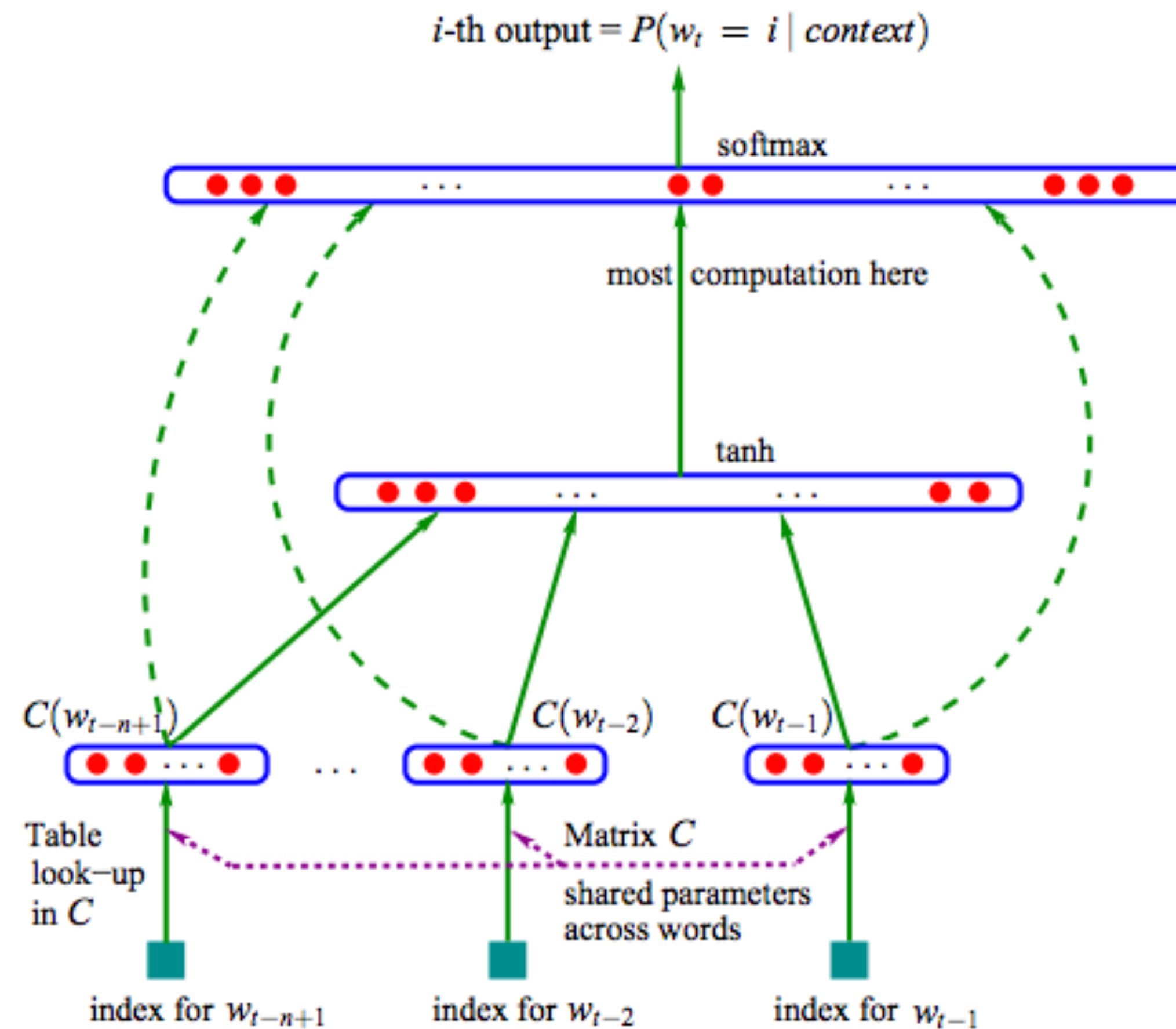
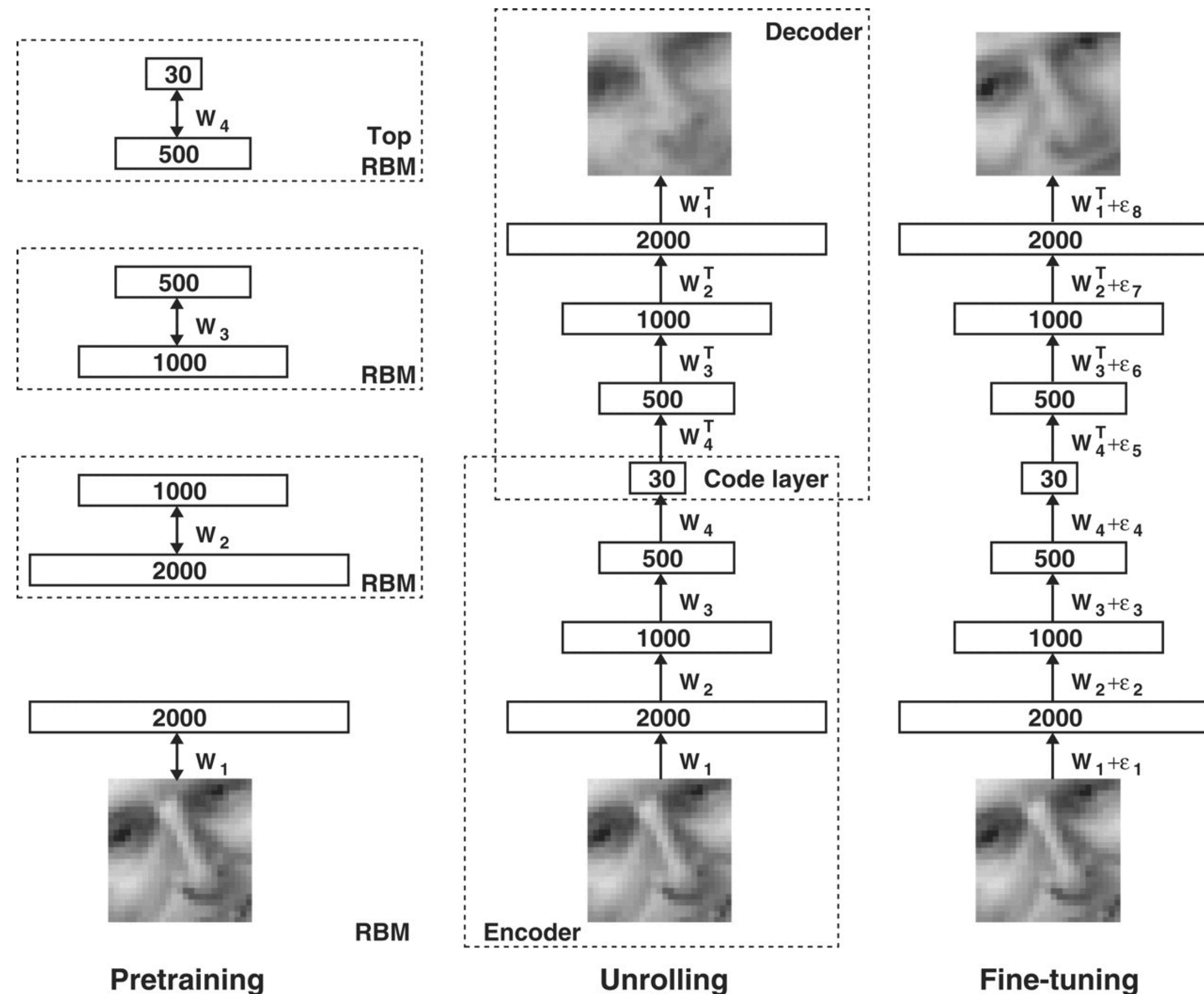


Figure 1: Neural architecture: $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i -th word feature vector.

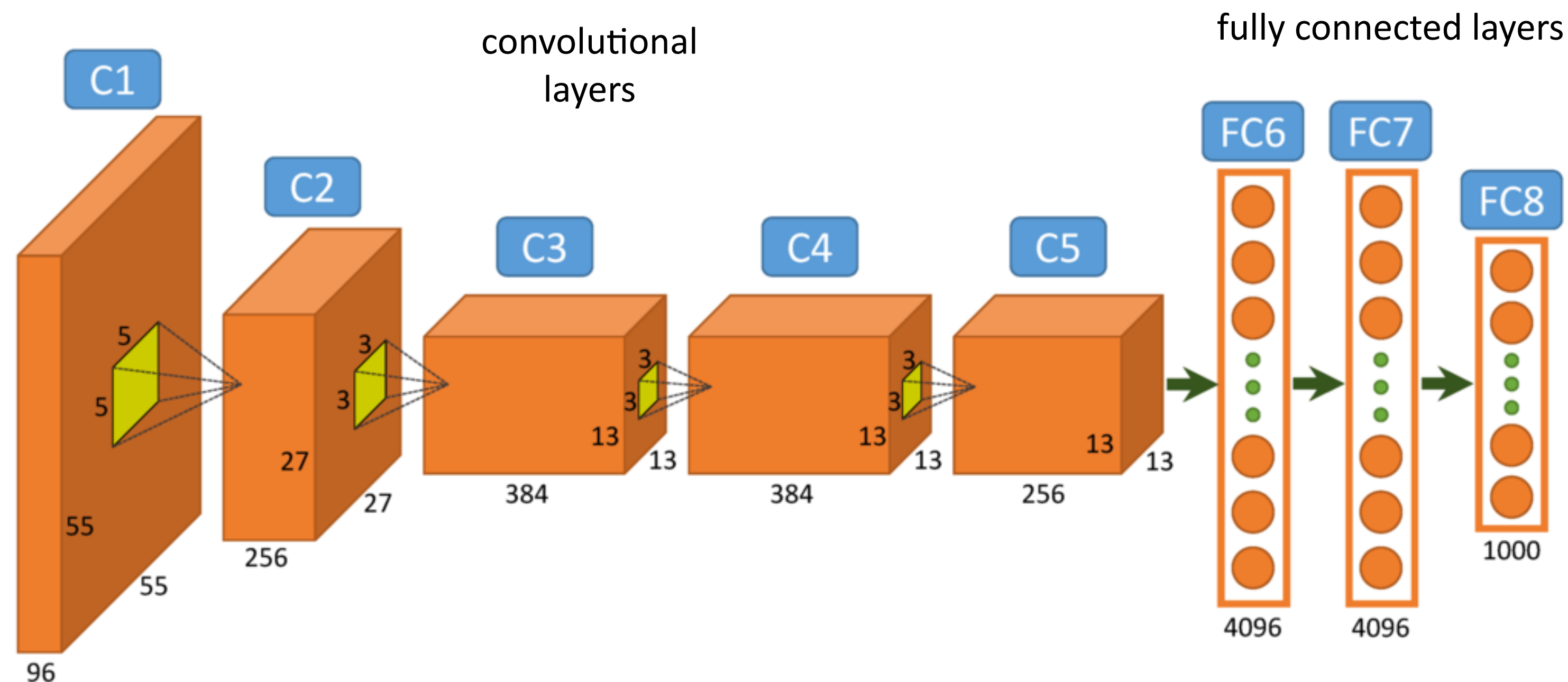
A Bit of History

- Reinvigorated research in deep learning (Hinton & Salakhutdinov, 2006)



Convolutional Neural Networks

- ▶ AlexNet - one of the first strong results
- ▶ more filters per layer as well as stacked convolutional layers
- ▶ use of ReLU for the non-linear part instead of sigmoid or Tanh



Krizhevsky et al. (2012)

ImageNet - Object Recognition

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images



Output:

Scale

T-shirt

Steel drum

Drumstick

Mud turtle



Output:

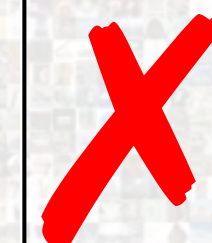
Scale

T-shirt

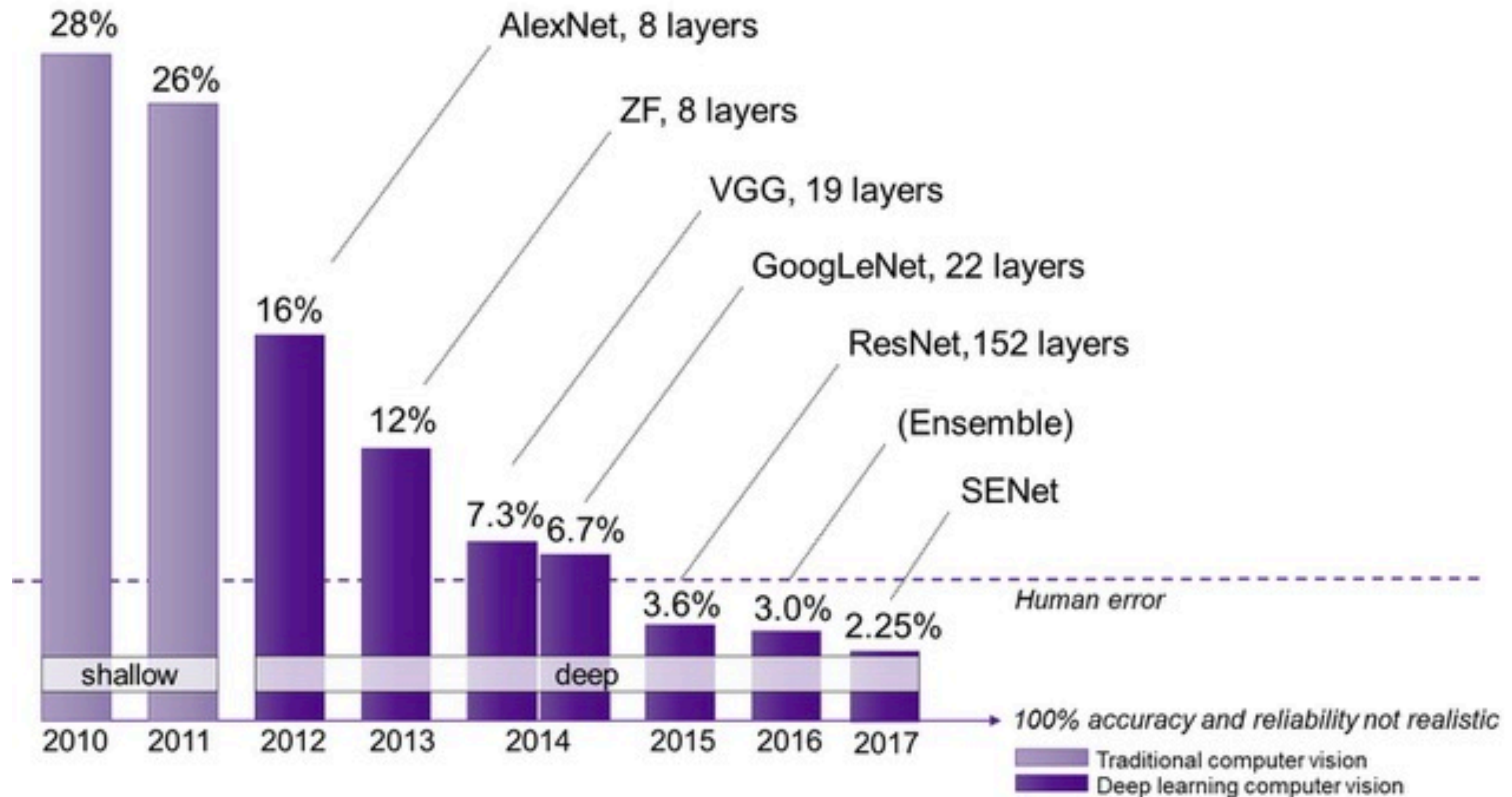
Giant panda

Drumstick

Mud turtle



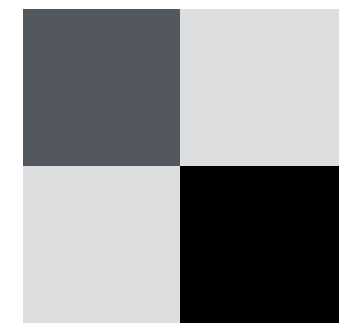
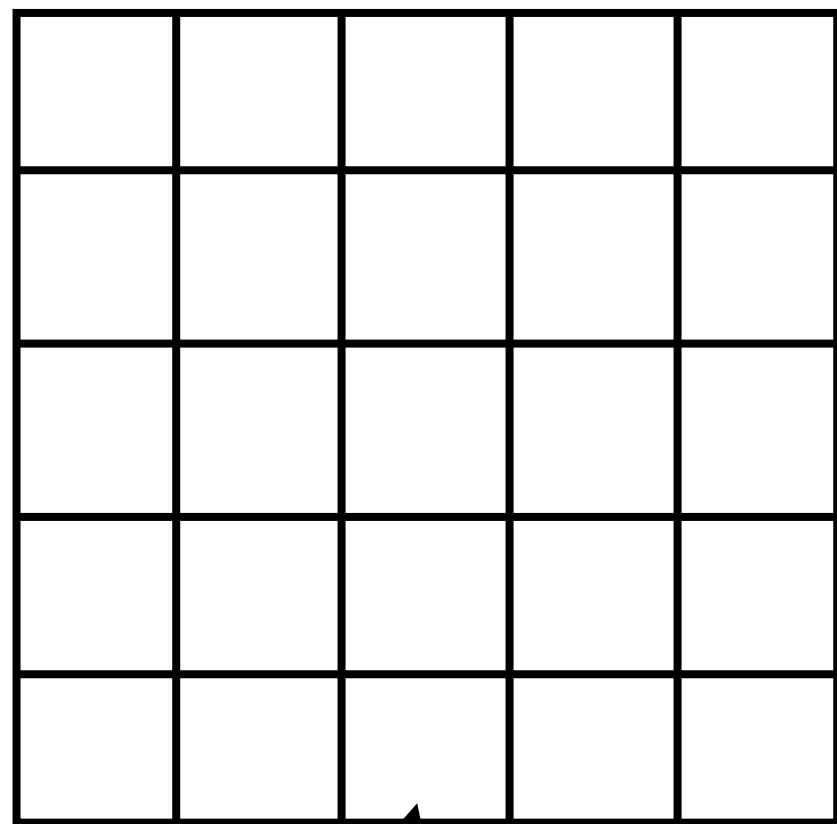
ImageNet - Object Recognition



Convolutional Layer

- ▶ Applies a *filter* over patches of the input and returns that filter's activations
- ▶ Convolution: take dot product of filter with a patch of the input

image: $n \times n \times k$ filter: $m \times m \times k$



sum over dot products

$$\text{activation}_{ij} = \sum_{i_o=0}^{m-1} \sum_{j_o=0}^{m-1} \text{image}(i + i_o, j + j_o) \cdot \text{filter}(i_o, j_o)$$

offsets

Each of these cells is a vector with multiple values

Images: RGB values (3 dim)

Convolutional Layer

- ▶ An animated example: $k = 1$, and a filter of size 3×3 .

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

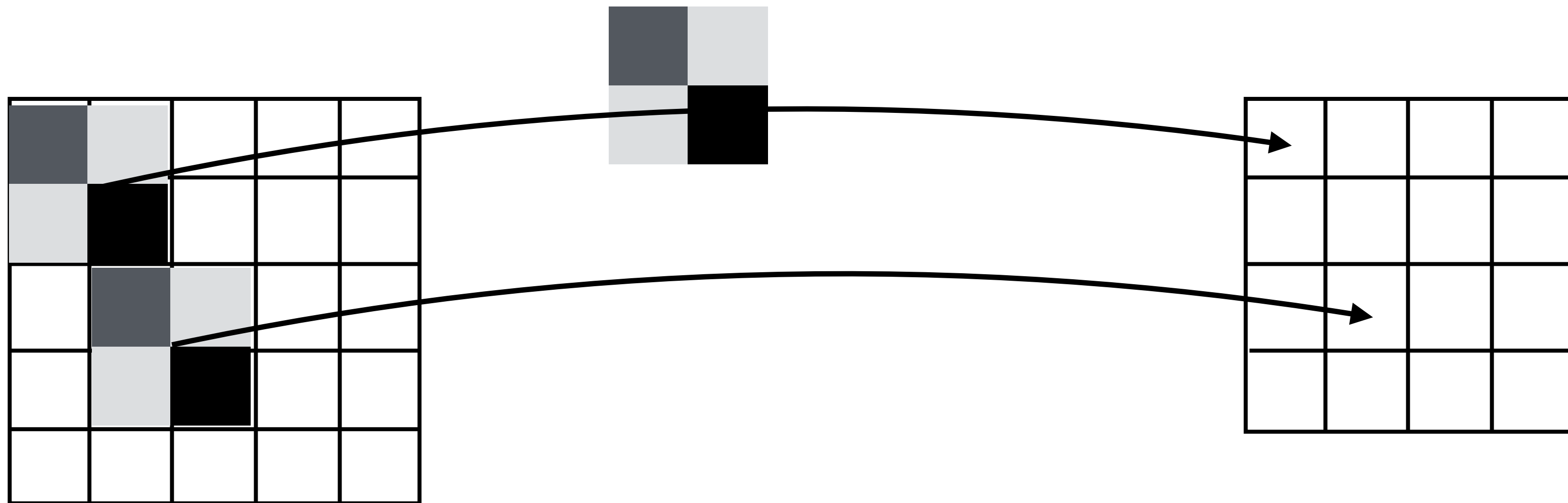
4		

Convolved
Feature

Convolutional Layer

- ▶ Applies a *filter* over patches of the input and returns that filter's activations
- ▶ Convolution: take dot product of filter with a patch of the input

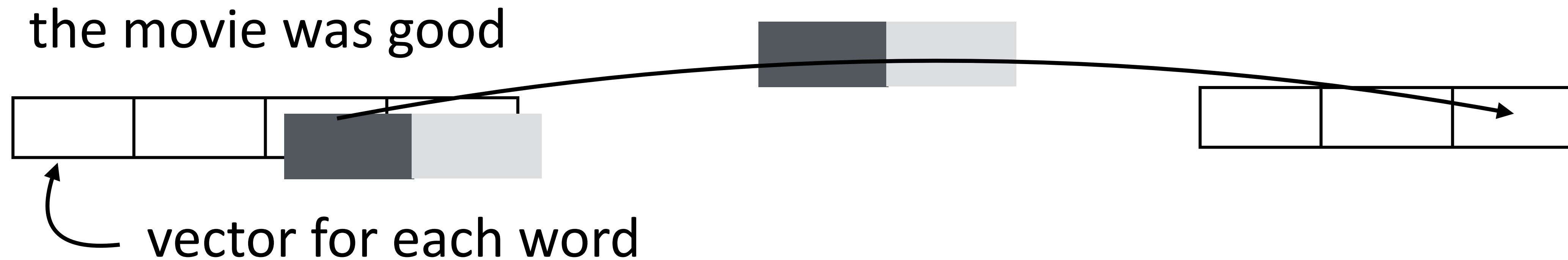
image: $n \times n \times k$ filter: $m \times m \times k$ activations: $(n - m + 1) \times (n - m + 1) \times 1$



Convolutions for NLP

- ▶ Input and filter are 2-dimensional instead of 3-dimensional

sentence: n words \times k vec dim filter: $m \times k$ activations: $(n - m + 1) \times 1$

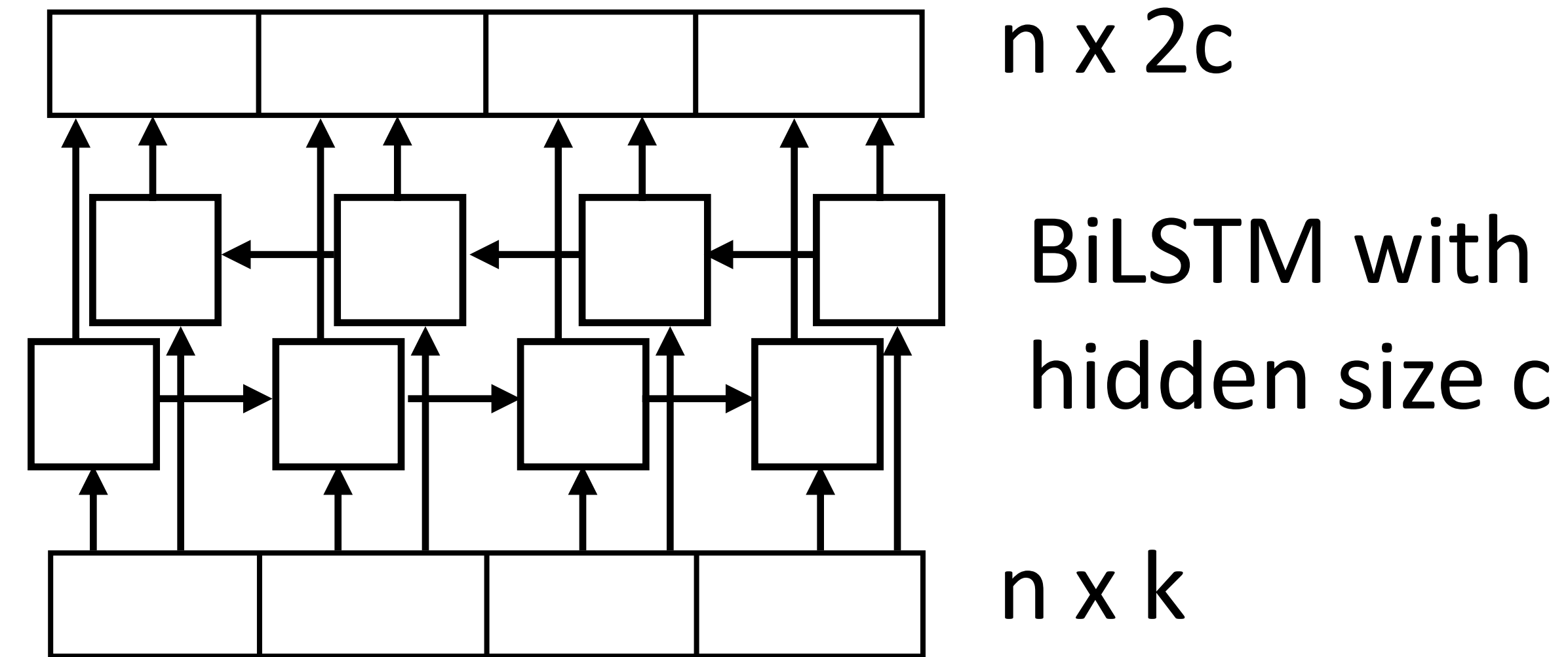


- ▶ Combines evidence locally in a sentence and produces a new (but still variable-length) representation

Compare: CNNs vs. LSTMs



the movie was good

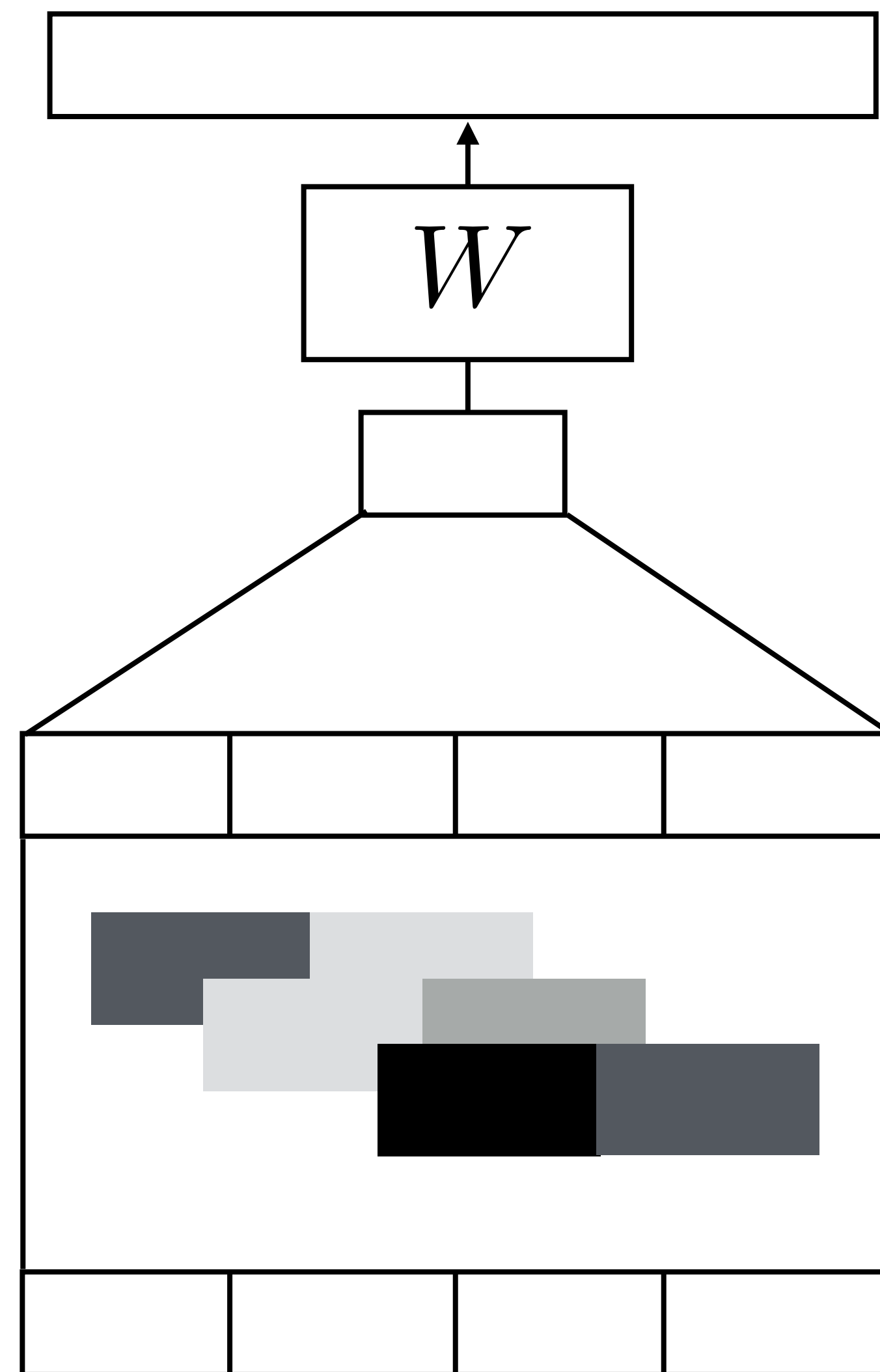


the movie was good

- ▶ Both LSTMs and convolutional layers transform the input using context
- ▶ LSTM: “globally” looks at the entire sentence (but local for many problems)
- ▶ CNN: local depending on filter width + number of layers

CNNs for Sentiment

CNNs for Sentiment Analysis



$$P(y|\mathbf{x})$$

projection + softmax

c-dimensional vector

max pooling over the sentence

$n \times c$

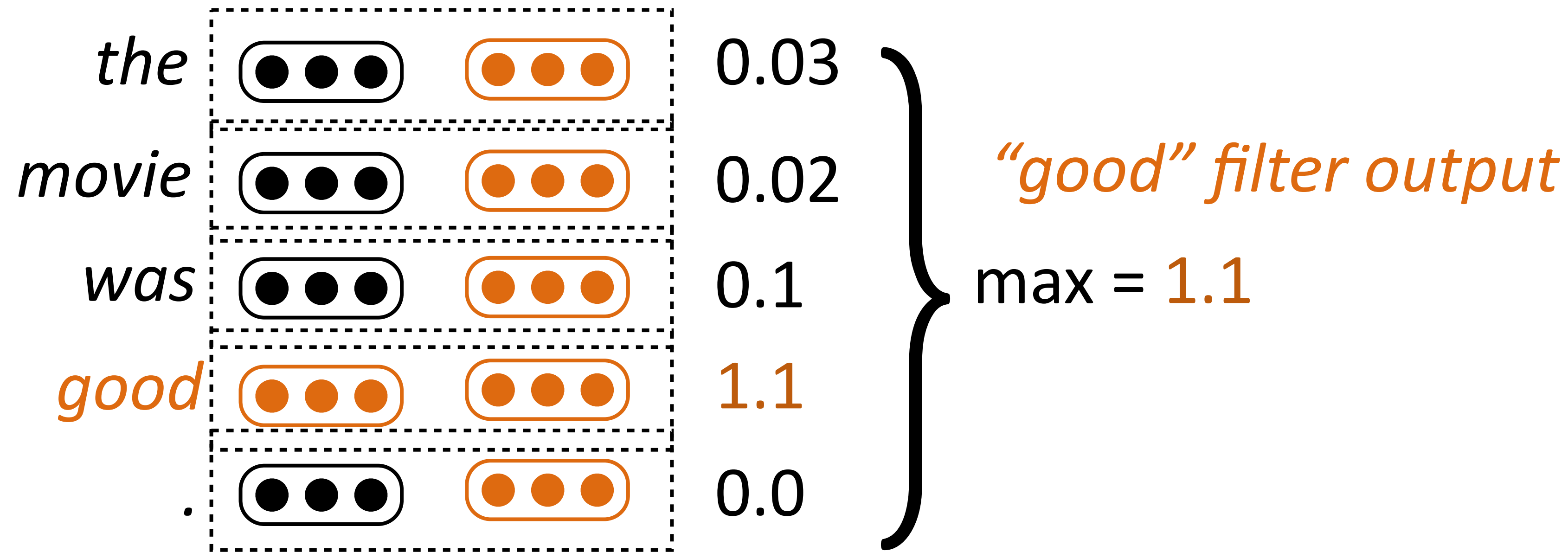
c filters,
 $m \times k$ each

$n \times k$

- ▶ Max pooling: return the max activation of a given filter over the entire sentence; like a logical OR (sum pooling is like logical AND)

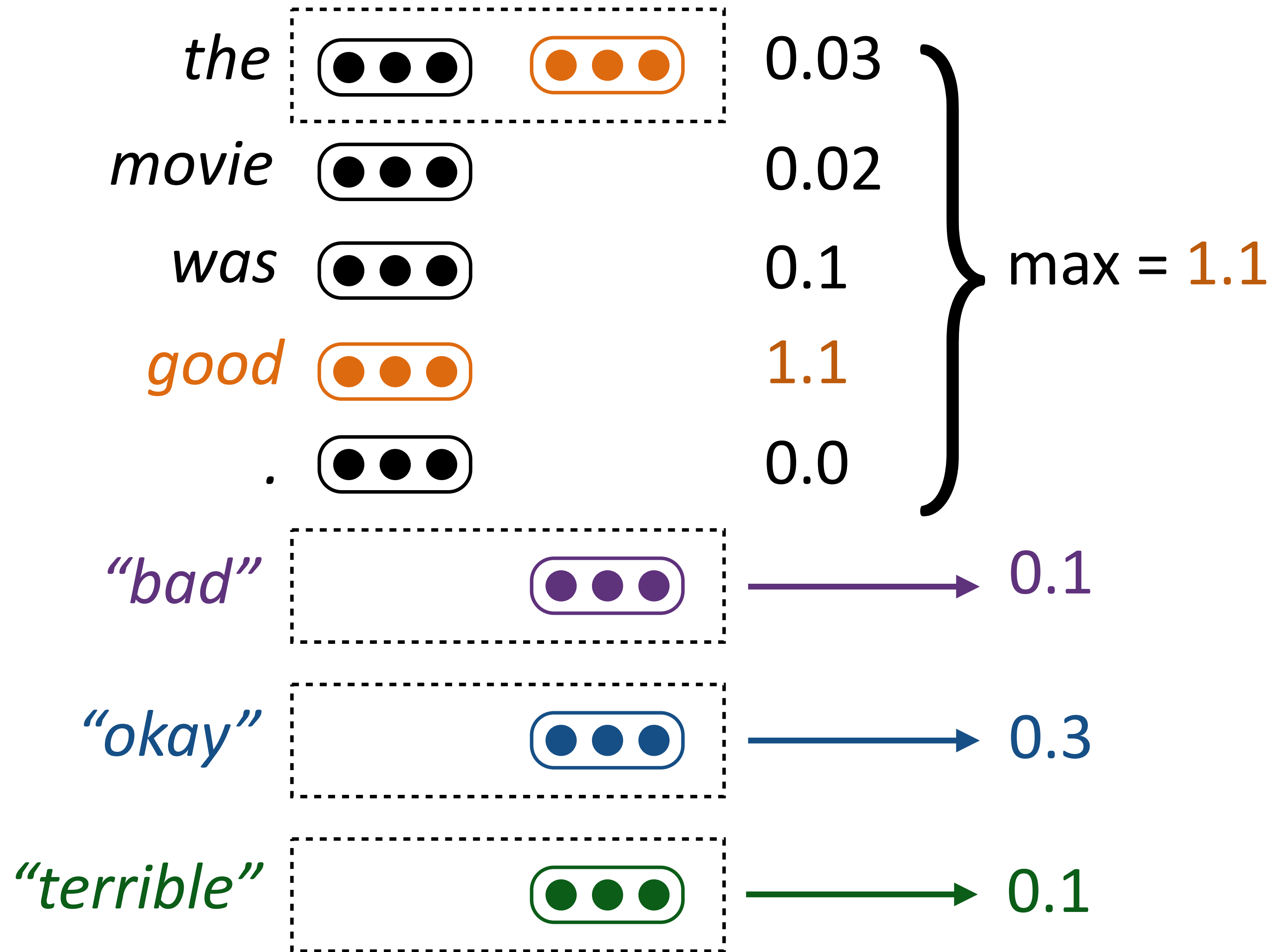
the movie was good

Understanding CNNs for Sentiment

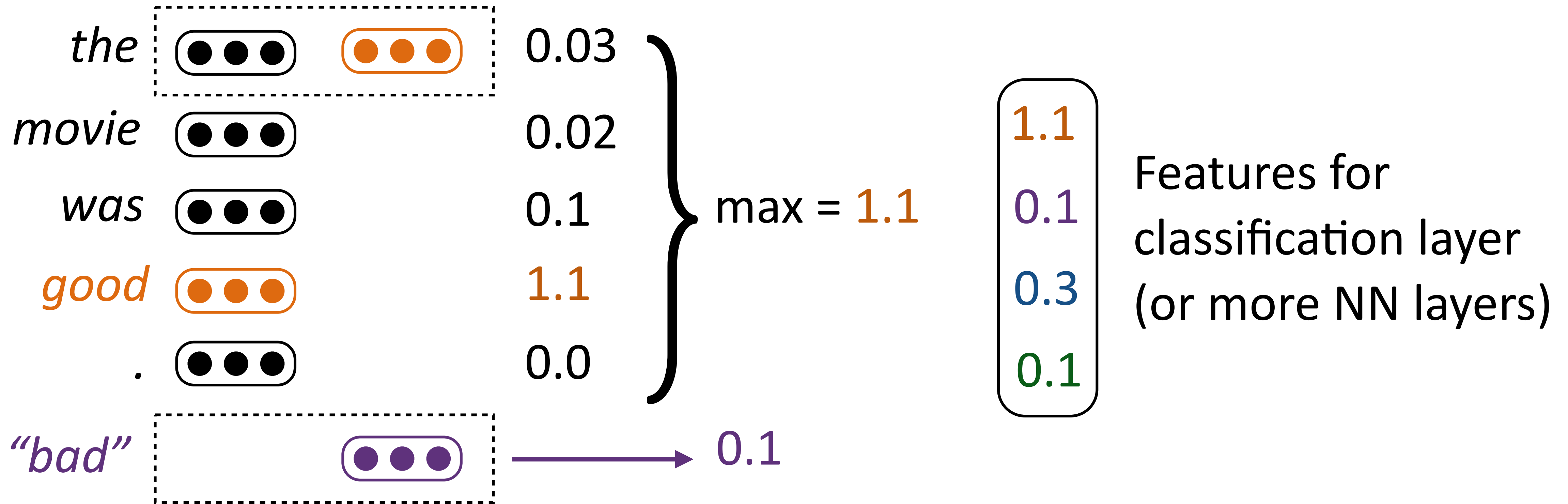


- ▶ Filter “looks like” the things that will cause it to have high activation

Understanding CNNs for Sentiment

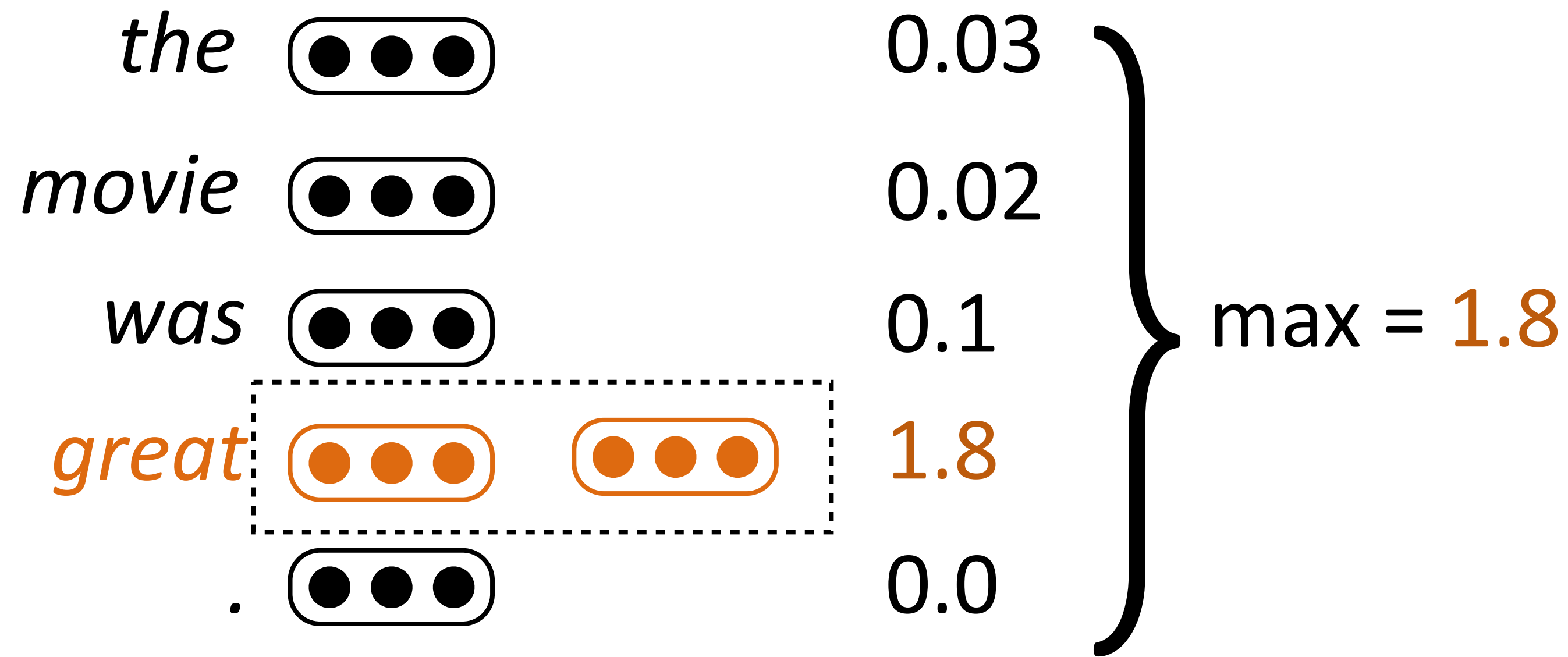


Understanding CNNs for Sentiment



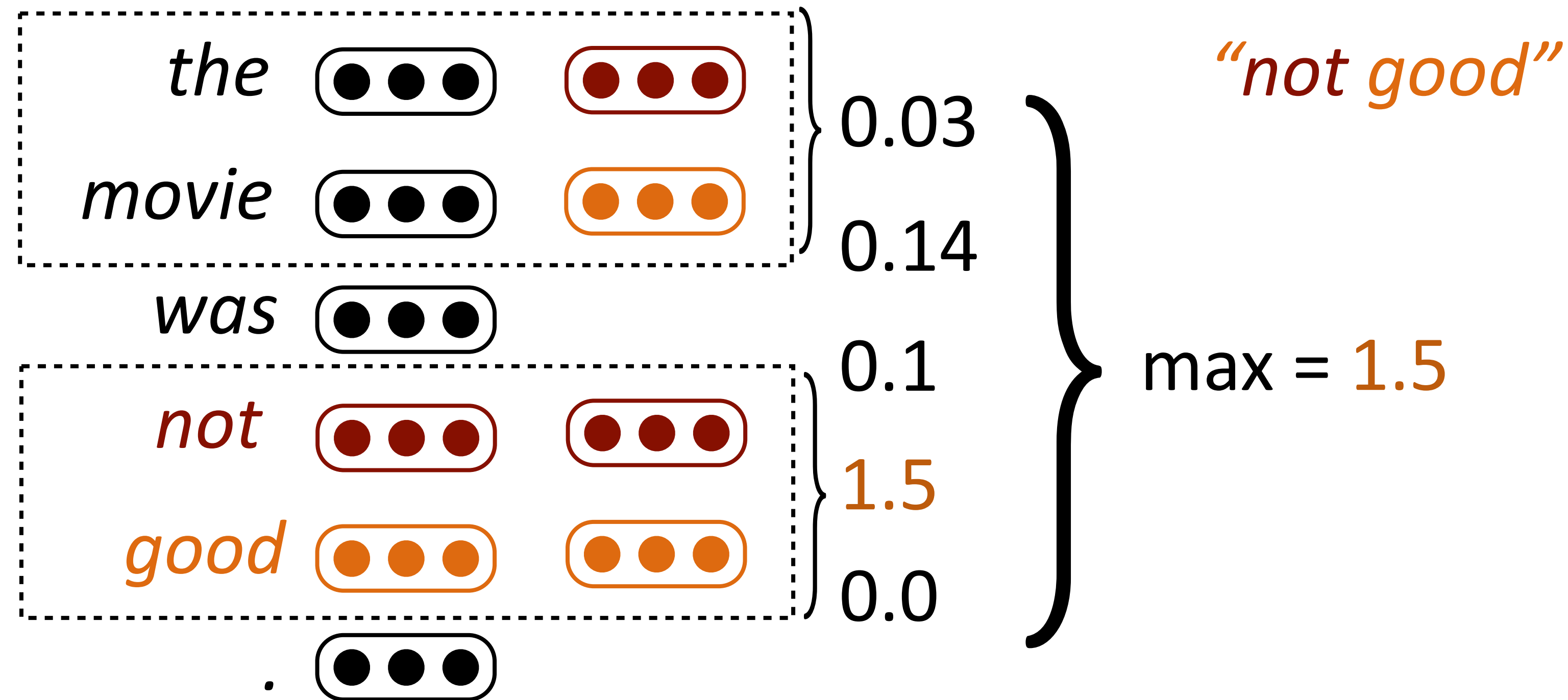
- ▶ Takes variable-length input and turns it into fixed-length output
- ▶ Filters are initialized randomly and then learned

Understanding CNNs for Sentiment



- ▶ Word vectors for similar words are similar, so convolutional filters will have similar outputs

Understanding CNNs for Sentiment



- ▶ Analogous to bigram features in bag-of-words models
- ▶ Indicator feature of text containing bigram \leftrightarrow max pooling of a filter that matches that bigram

What can CNNs learn?

- ▶ CNNs let us take advantage of word similarity

really not very good vs. really not very enjoyable

- ▶ CNNs are translation-invariant like bag-of-words

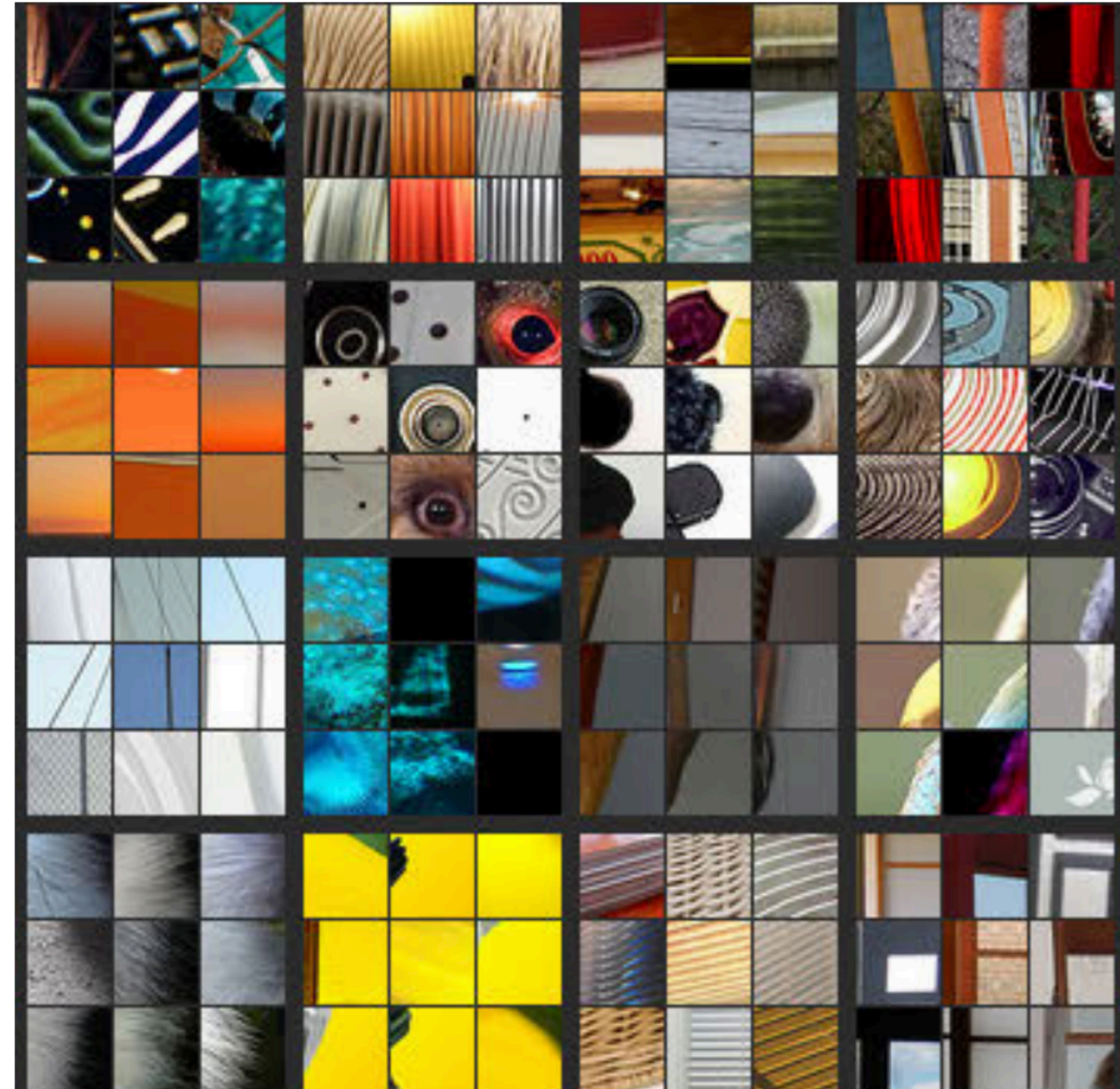
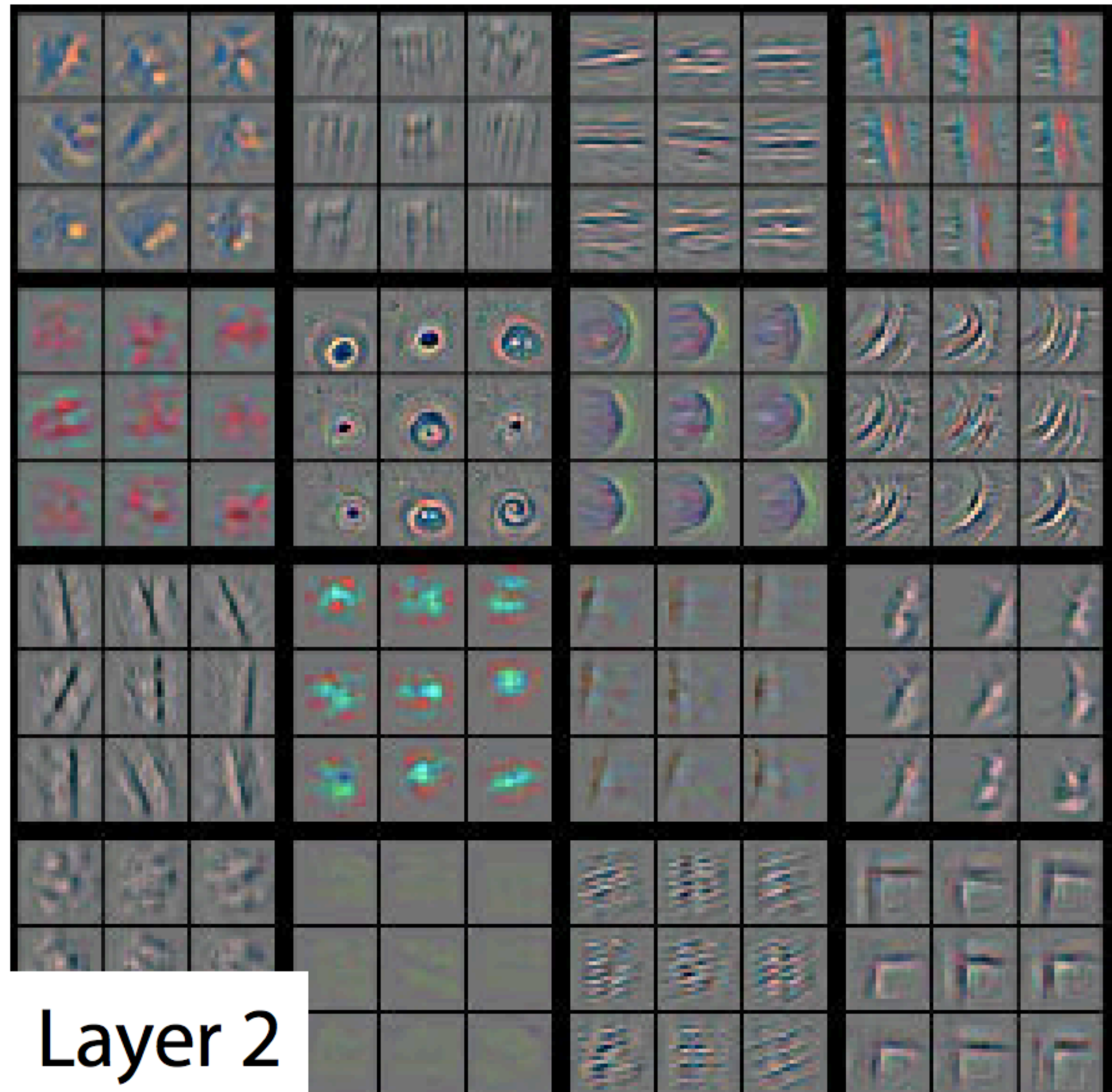
The movie was bad, but blah blah blah ... vs. ... blah blah blah, but the movie was bad.

- ▶ CNNs can capture local interactions with filters of width > 1

It was not good, it was actually quite bad vs. it was not bad, it was actually quite good

Deep Convolutional Networks

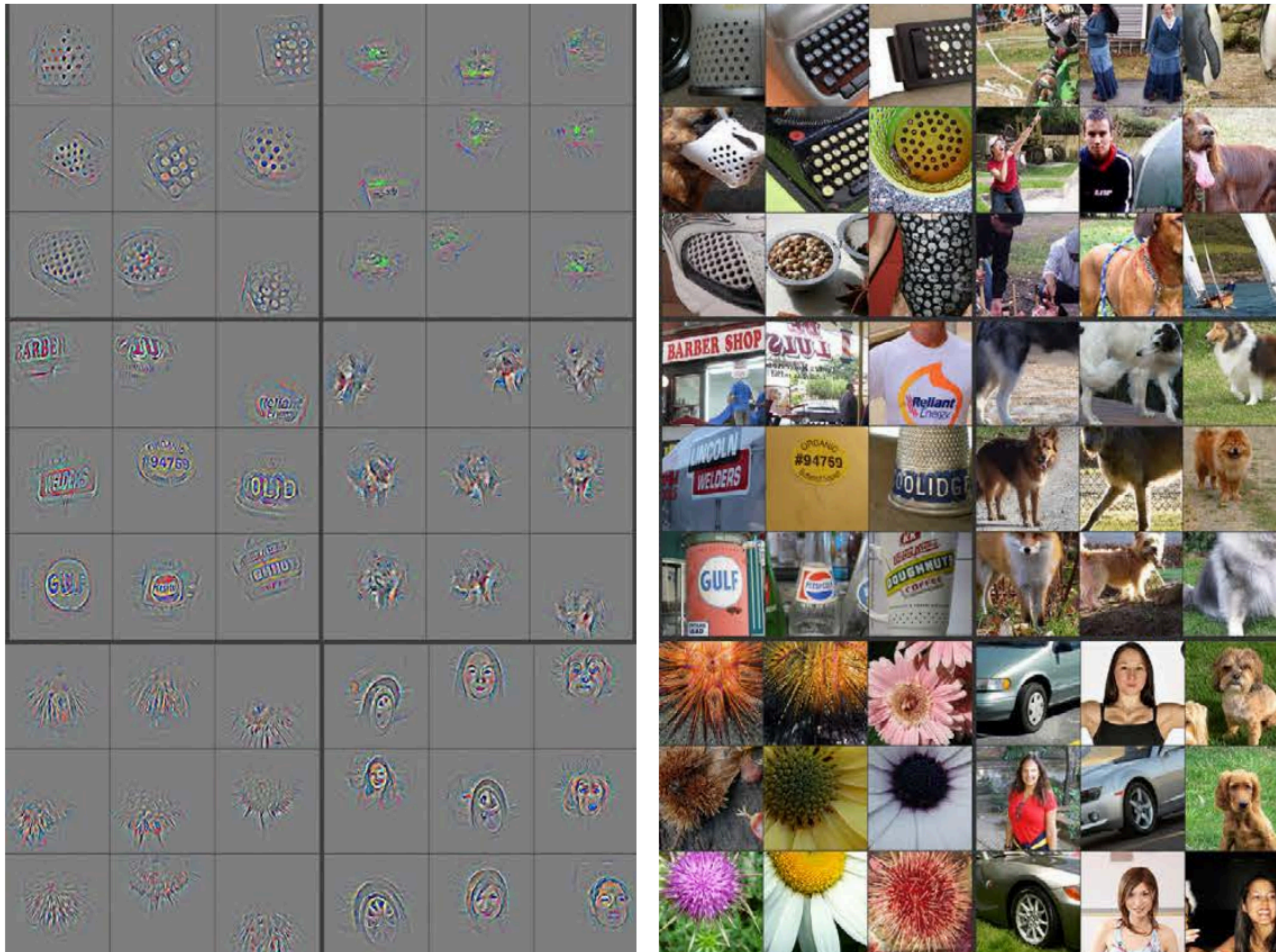
- ▶ Low-level filters: extract low-level features from the data



Zeiler and Fergus (2014)

Deep Convolutional Networks

- ▶ High-level filters: match larger and more “semantic patterns”



Zeiler and Fergus (2014)

CNNs: Implementation

- ▶ Input is `batch_size x n x k` matrix, filters are `c x m x k` matrix (`c` filters)
- ▶ Typically use filters with `m` ranging from 1 to 5 or so (multiple filter widths in a single convnet)
- ▶ All computation graph libraries support efficient convolution operations

```
CLASS torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')
```

[SOURCE]

Applies a 1D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, L) and output (N, C_{out}, L_{out}) can be precisely described as:

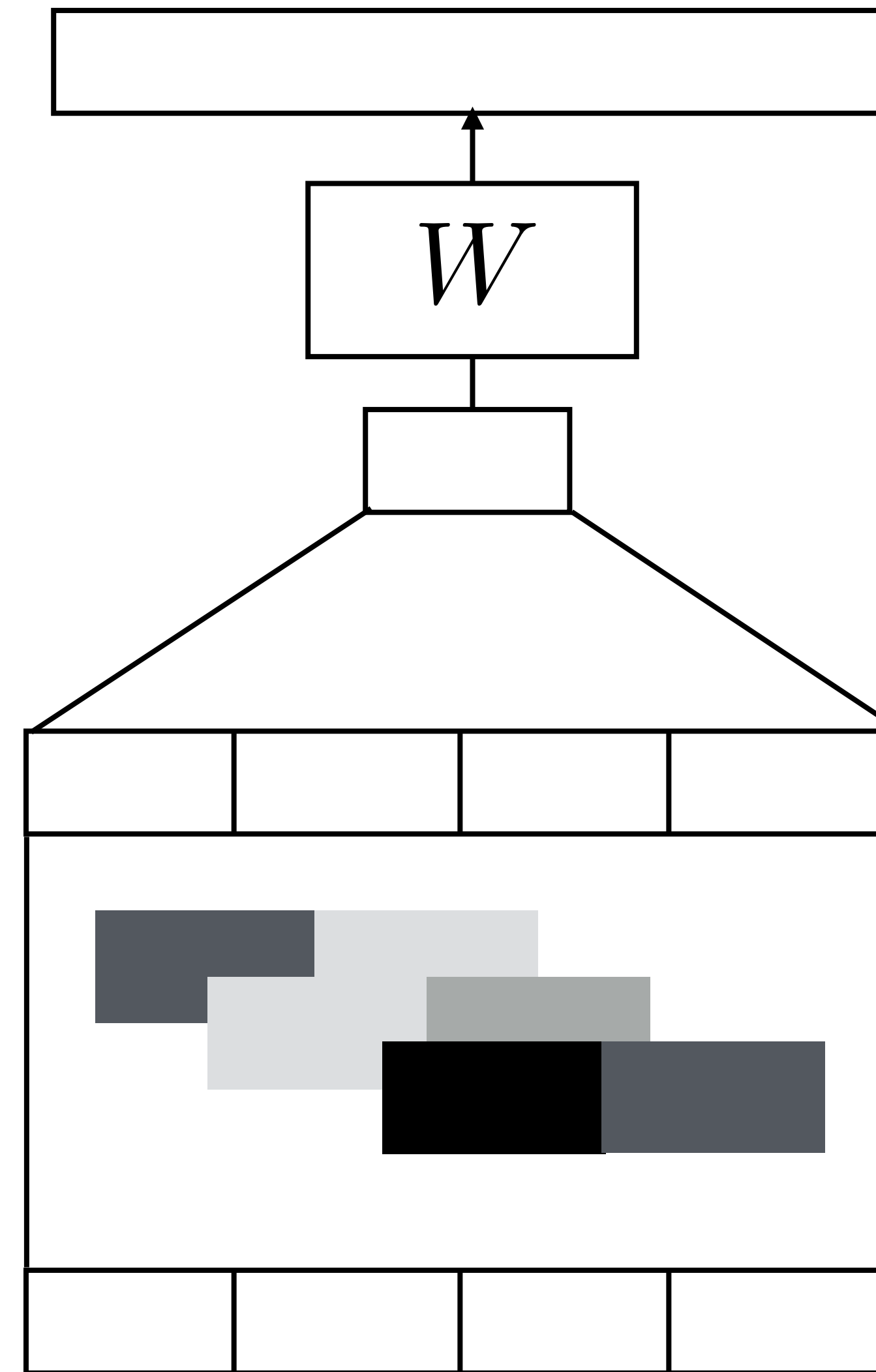
$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k)$$

where \star is the valid **cross-correlation** operator, N is a batch size, C denotes a number of channels, L is a length of signal sequence.

- `stride` controls the stride for the cross-correlation, a single number or a one-element tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points.

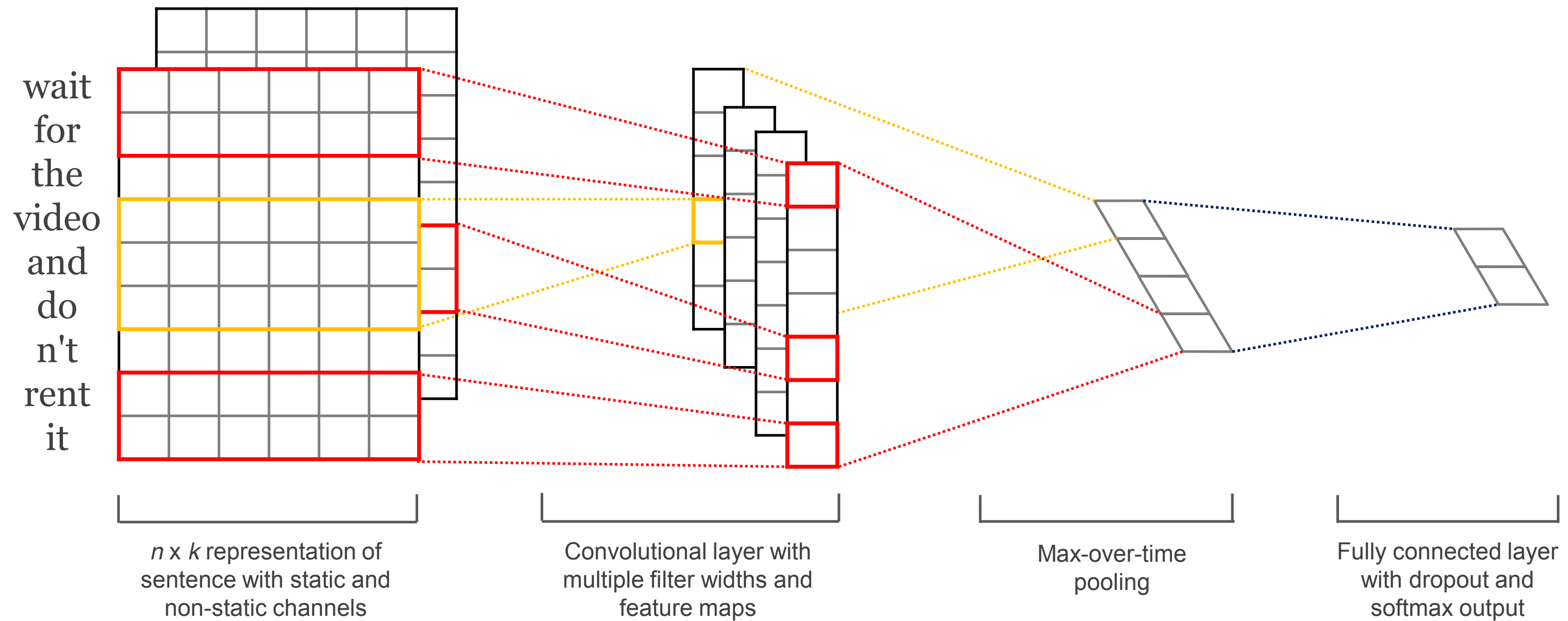
CNNs for Sentence Classification

- ▶ Question classification, sentiment, etc.
- ▶ Conv+pool, then use feedforward layers to classify
- ▶ Can use multiple types of input vectors (fixed initializer and learned)



the movie was good

CNNs for Sentence Classification



Sentence Classification

movie review sentiment subjectivity/objectivity detection product reviews

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-multichannel	81.1	47.4	88.1	93.2	92.2	85.0	89.4
NBSVM (Wang and Manning, 2012)	79.4	—	—	93.2	—	81.8	86.3

question type classification

- ▶ Also effective at document-level text classification

Entity Linking

- ▶ CNNs can produce good representations of both sentences and documents like typical bag-of-words features
- ▶ Can distill topic representations for use in entity linking

that they had **disqualified** **Armstrong**
from his seven consecutive

cycling domain



Lance Armstrong

geopolitical domain



Armstrong County

Entity Linking

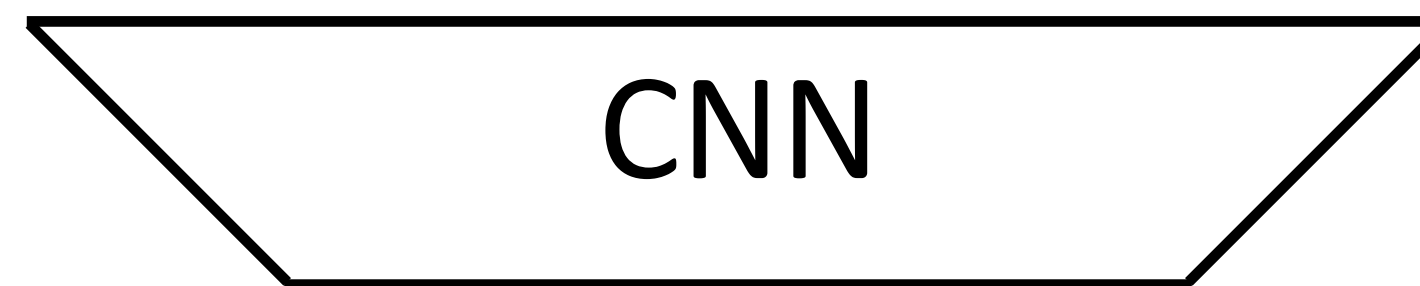
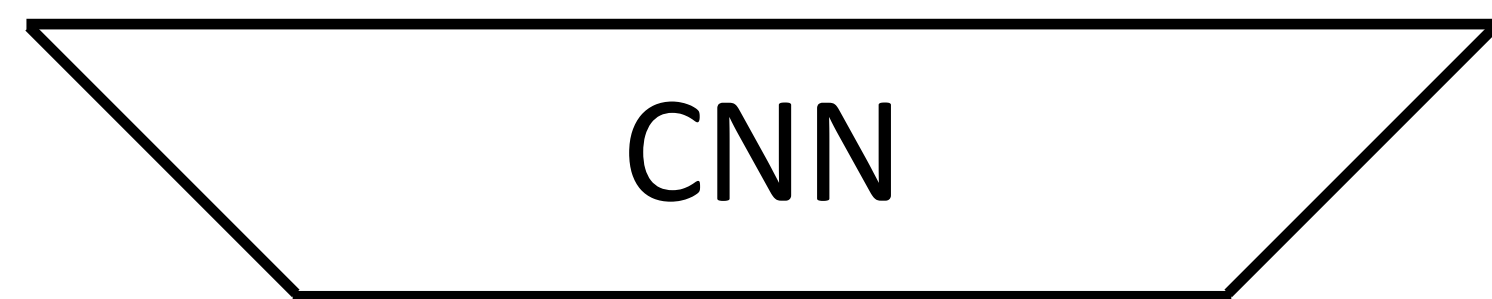
Although he originally won the event, the United States Anti-Doping Agency announced in August 2012 that they had disqualified **Armstrong** from his seven consecutive Tour de France wins from 1999–2005.



Lance Edward Armstrong is an American former professional road cyclist



Armstrong County is a county in Pennsylvania...



Document topic vector d

Article topic vector a_{Lance}

Article topic vector a_{County}

$$s_{\text{Lance}} = d \cdot a_{\text{Lance}}$$

$$s_{\text{County}} = d \cdot a_{\text{County}}$$

$$P(y|\mathbf{x}) = \text{softmax}(\mathbf{s})$$

Francis-Landau et al. (2016)

Neural CRF Basics

NER Revisited



- ▶ Features in CRFs: $I[\text{tag}=\text{B-LOC} \ \& \ \text{curr_word}=\text{Hangzhou}]$, $I[\text{tag}=\text{B-LOC} \ \& \ \text{prev_word}=\text{to}]$, $I[\text{tag}=\text{B-LOC} \ \& \ \text{curr_prefix}=\text{Han}]$
- ▶ Linear model over features
- ▶ Downsides:
 - ▶ Lexical features mean that words need to be seen in the training data
 - ▶ Linear model can't capture feature conjunctions as effectively (doesn't work well to look at more than 2 words with a single feature)

LSTMs for NER

B-PER I-PER O O O B-LOC O O O B-ORG O O

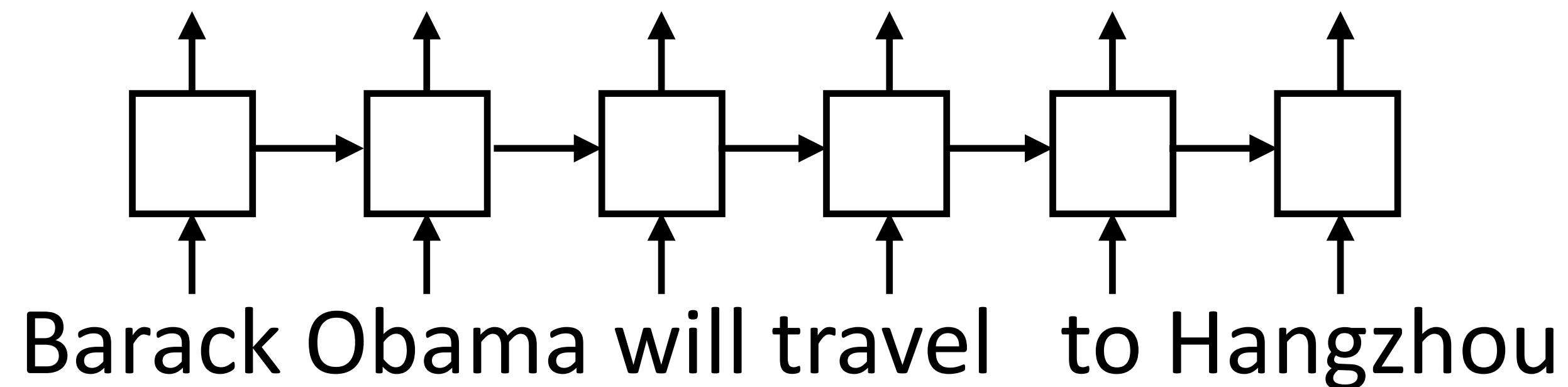
Barack Obama will travel to Hangzhou today for the G20 meeting .

PERSON

LOC

ORG

B-PER I-PER O O O B-LOC



- ▶ Transducer (LM-like model)
- ▶ What are the strengths and weaknesses of this model compared to CRFs?

LSTMs for NER

B-PER I-PER O O O B-LOC O O O B-ORG O O

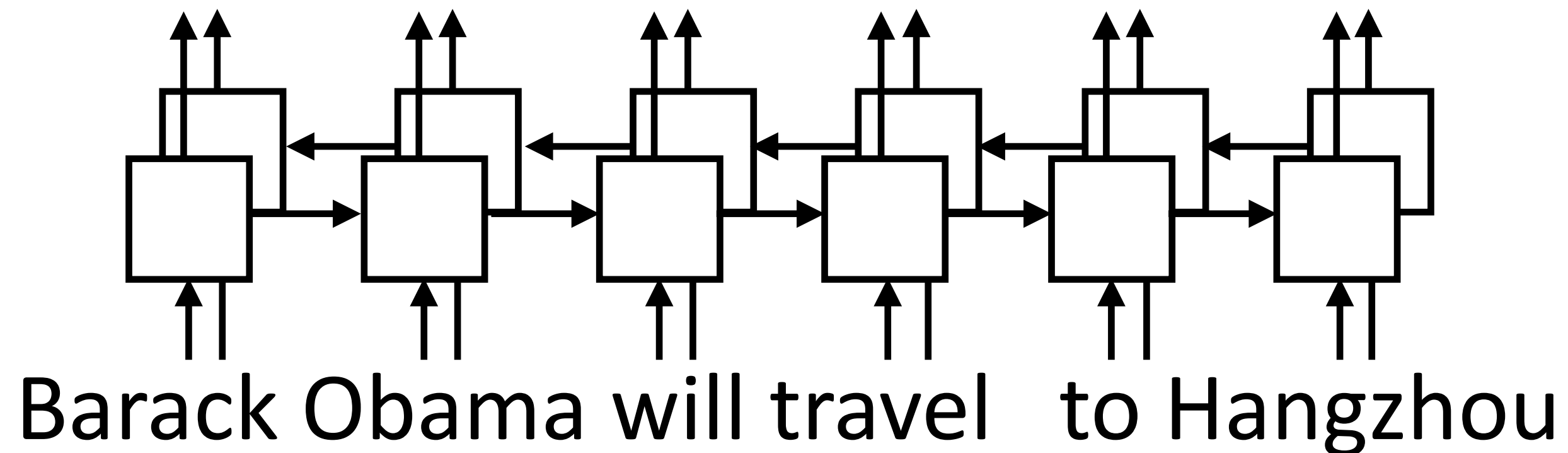
Barack Obama will travel to Hangzhou today for the G20 meeting .

PERSON

LOC

ORG

B-PER I-PER O O O B-LOC

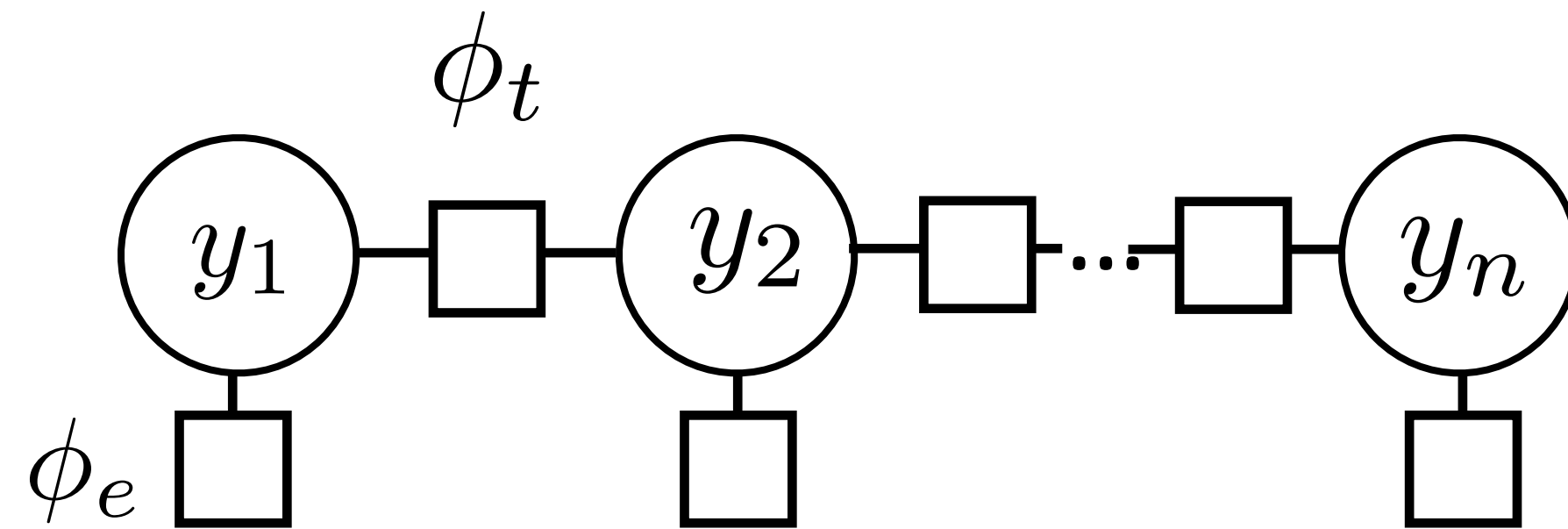


- ▶ Bidirectional transducer model
- ▶ What are the strengths and weaknesses of this model compared to CRFs?

Recall: Sequential CRFs

► Model:
$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \prod_{i=2}^n \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^n \exp(\phi_e(y_i, i, \mathbf{x}))$$

► Normalizing constant
$$Z = \sum_{\mathbf{y}} \prod_{i=2}^n \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^n \exp(\phi_e(y_i, i, \mathbf{x}))$$



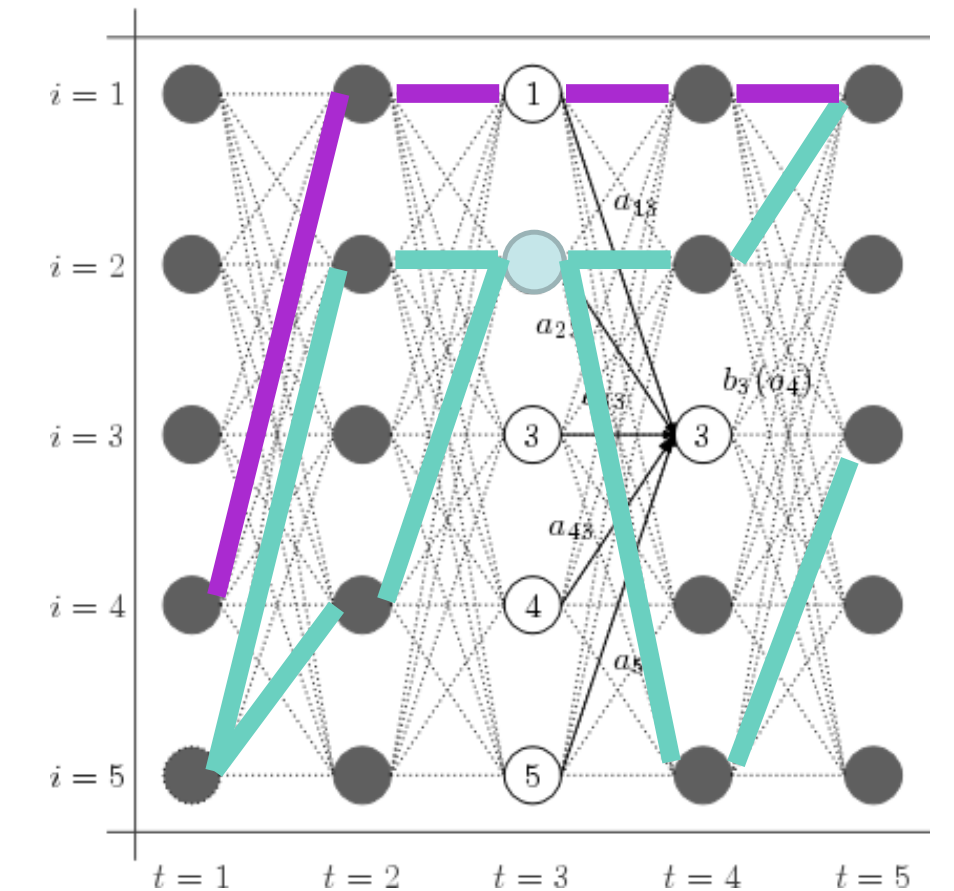
Recall: Sequential CRFs

- ▶ Inference: use Viterbi algorithm $P(\mathbf{y}_{\max}|\mathbf{x}) = \max_{y_1, \dots, y_n} P(\mathbf{y}|\mathbf{x})$

- ▶ Learning: run forward-backward to compute marginals

$$P(y_i = s|\mathbf{x}) = \sum_{y_1, \dots, y_{i-1}, y_{i+1}, \dots, y_n} P(\mathbf{y}|\mathbf{x})$$

$P(y_i = s_1, y_{i+1} = s_2|\mathbf{x})$, then update gradient



Neural CRFs

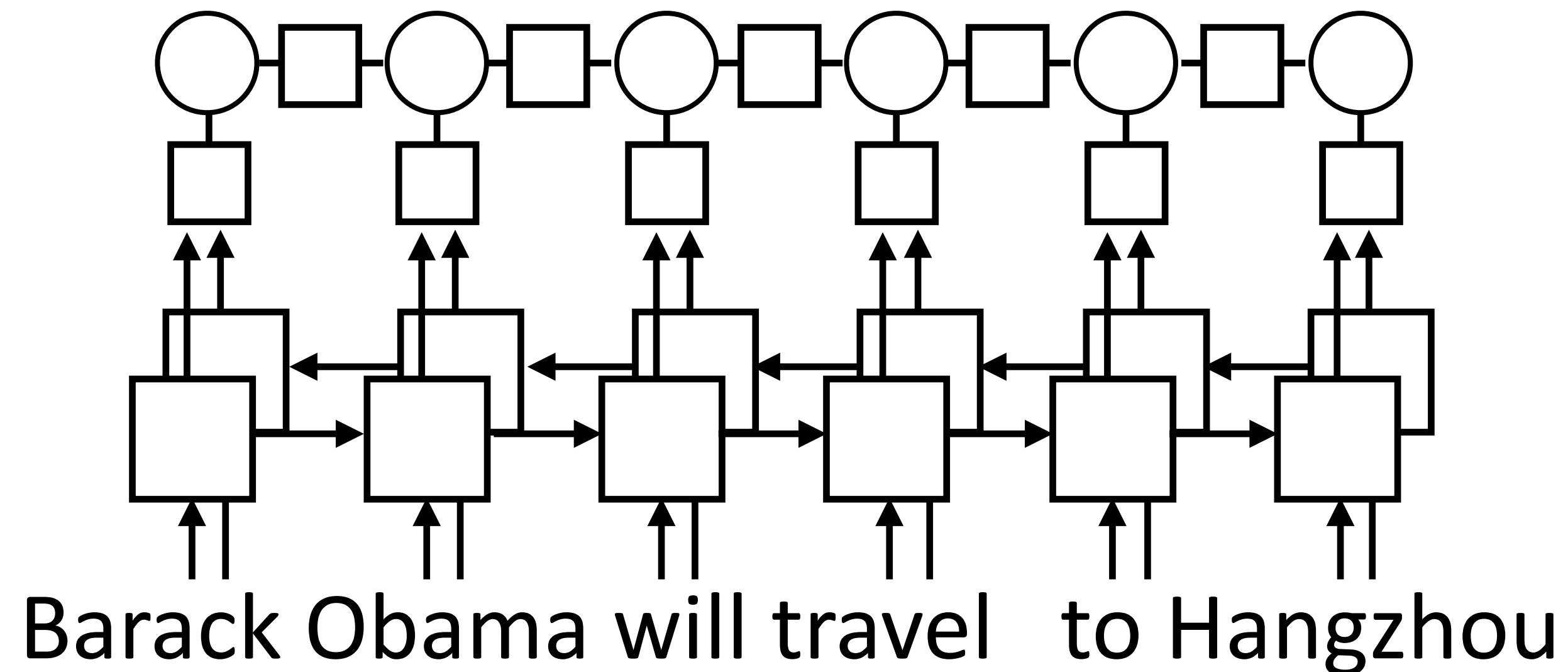
B-PER I-PER O O O B-LOC O O O B-ORG O O

Barack Obama will travel to Hangzhou today for the G20 meeting .

PERSON

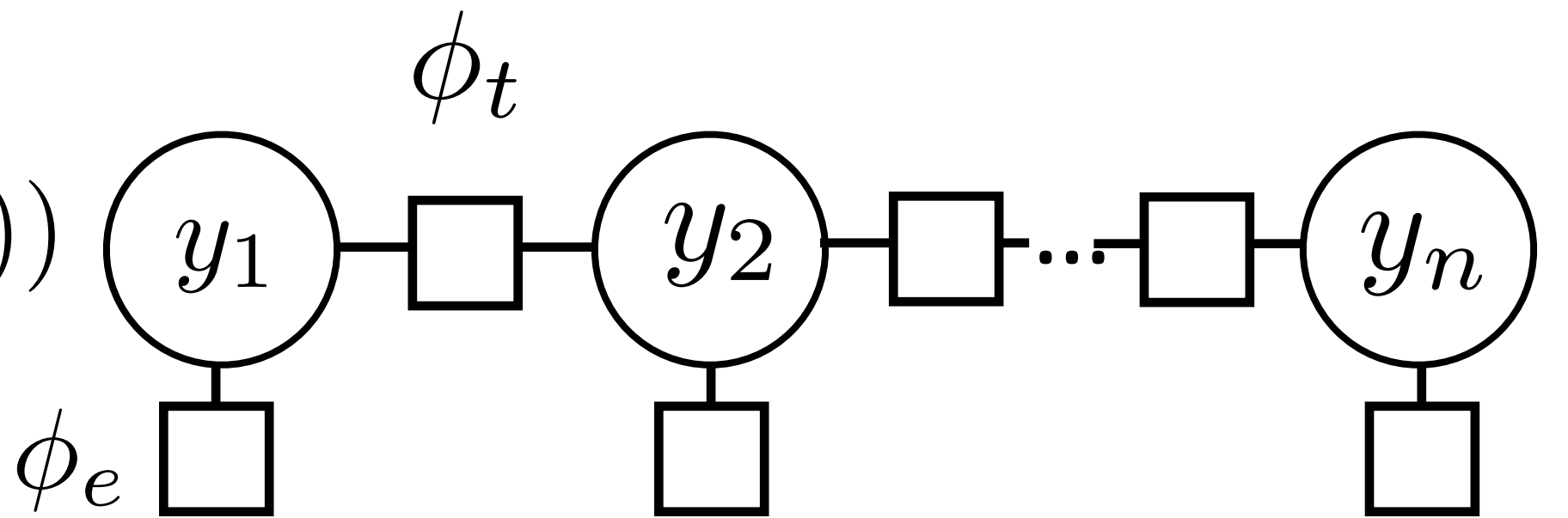
LOC

ORG



- ▶ Neural CRFs: bidirectional LSTMs (or some NN) compute emission potentials, capture structural constraints in transition potentials

Neural CRFs

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \prod_{i=2}^n \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^n \exp(\phi_e(y_i, i, \mathbf{x}))$$


- ▶ Conventional: $\phi_e(y_i, i, \mathbf{x}) = w^\top f_e(y_i, i, \mathbf{x})$
- ▶ Neural: $\phi_e(y_i, i, \mathbf{x}) = W_{y_i}^\top f(i, \mathbf{x})$ W is a `num_tags x len(f)` matrix
- ▶ $f(i, \mathbf{x})$ could be the output of a feedforward neural network looking at the words around position i , or the i th output of an LSTM, ...
- ▶ Neural network computes unnormalized potentials that are consumed and “normalized” by a structured model
- ▶ Inference: compute f , use Viterbi

Computing Gradients

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z} \prod_{i=2}^n \exp(\phi_t(y_{i-1}, y_i)) \prod_{i=1}^n \exp(\phi_e(y_i, i, \mathbf{x}))$$

▶ Conventional: $\phi_e(y_i, i, \mathbf{x}) = w^\top f_e(y_i, i, \mathbf{x})$

▶ Neural: $\phi_e(y_i, i, \mathbf{x}) = W_{y_i}^\top f(i, \mathbf{x})$

$\frac{\partial \mathcal{L}}{\partial \phi_{e,i}} = -P(y_i = s|\mathbf{x}) + I[s \text{ is gold}]$ “error signal”, compute with F-B

▶ For linear model: $\frac{\partial \phi_{e,i}}{w_i} = f_{e,i}(y_i, i, \mathbf{x})$ chain rule say to multiply together, gives our update

▶ For neural model: compute gradient of phi w.r.t. parameters of neural net

Neural CRFs

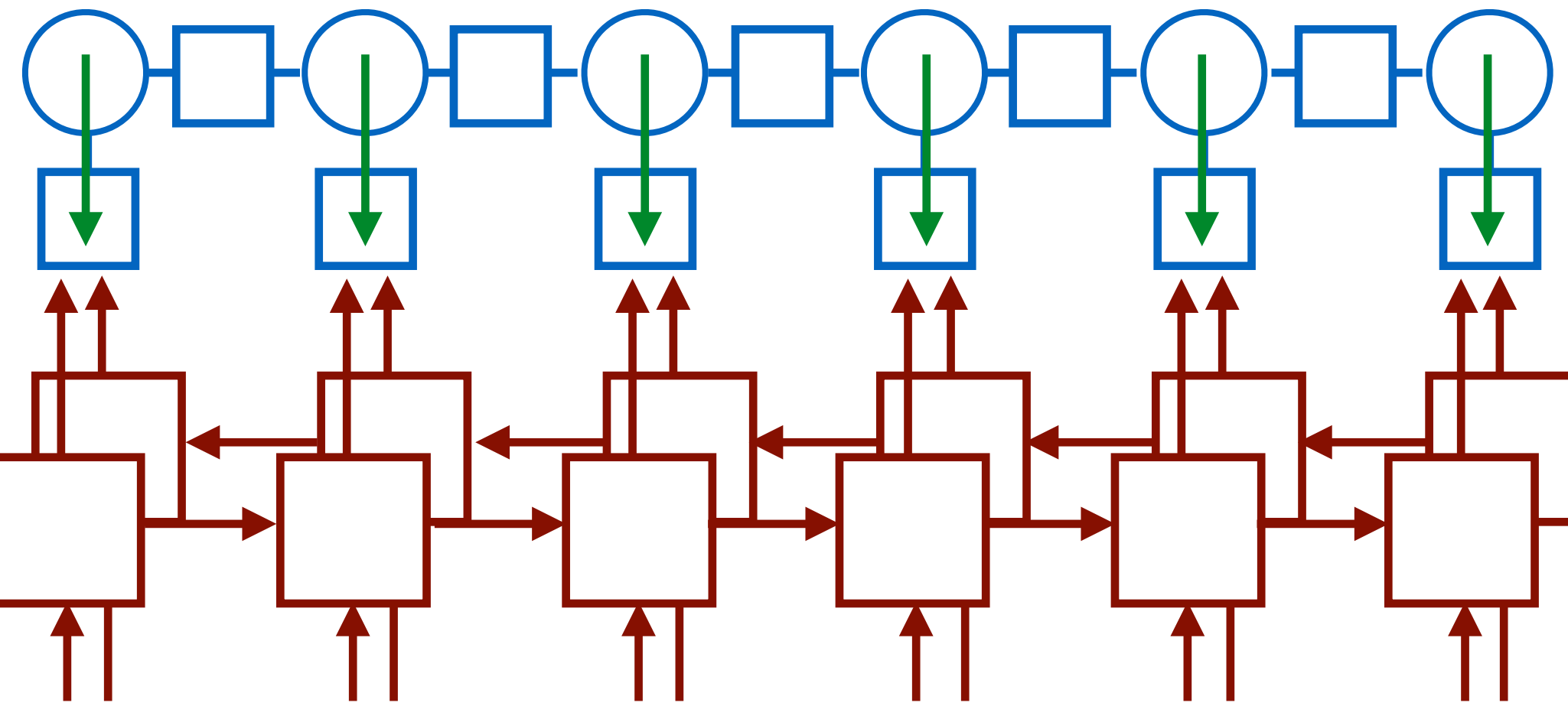
B-PER I-PER O O O B-LOC O O O B-ORG O O

Barack Obama will travel to Hangzhou today for the G20 meeting .

PERSON

LOC

ORG



Barack Obama will travel to Hangzhou

2) Run forward-backward

3) Compute error signal

1) Compute $f(\mathbf{x})$

4) Backprop (no knowledge of sequential structure required)

FFNN Neural CRF for NER

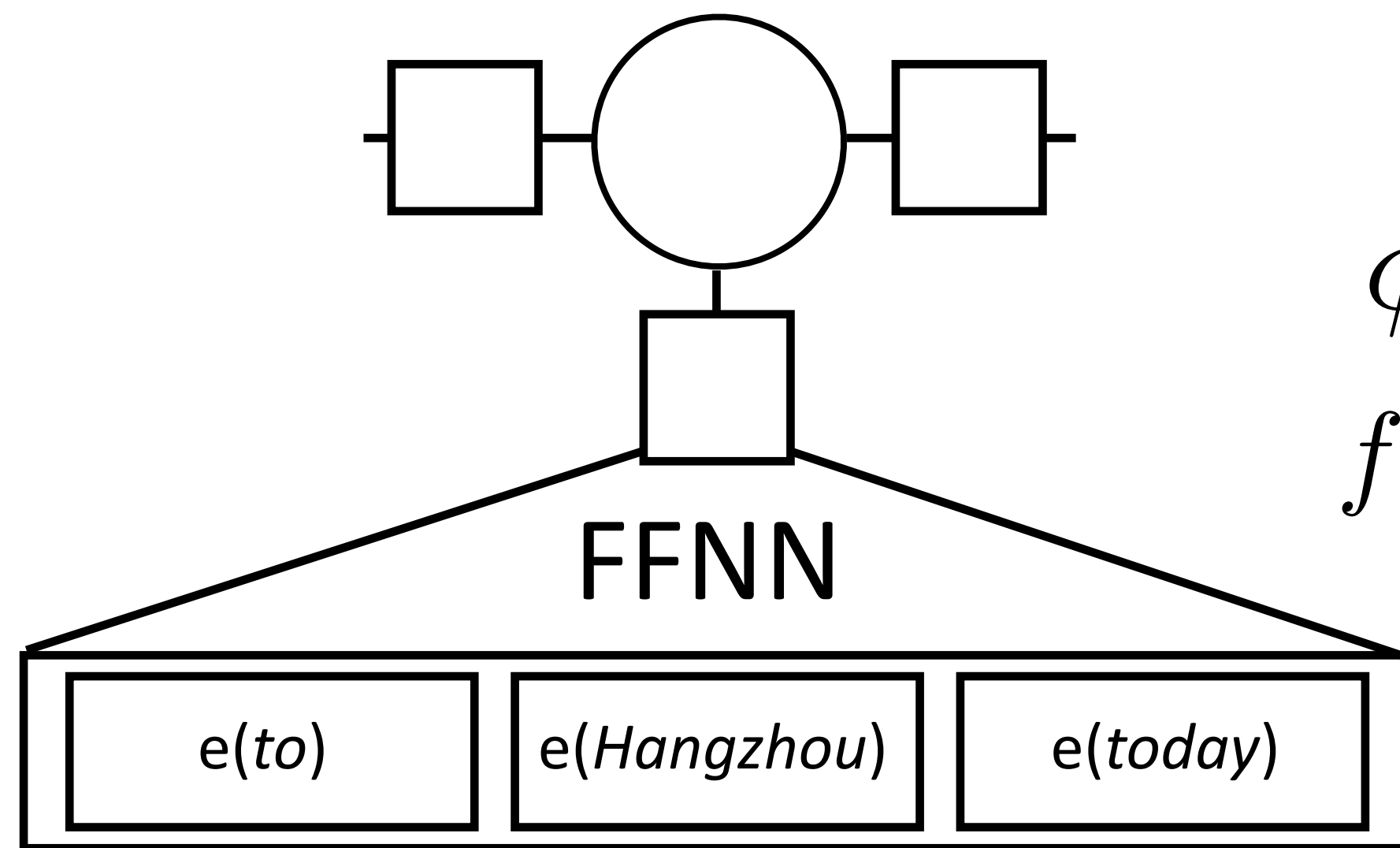
B-PER I-PER O O O B-LOC O O O B-ORG O O

Barack Obama will travel to Hangzhou today for the G20 meeting .

PERSON

LOC

ORG



$$\phi_e = Wg(Vf(\mathbf{x}, i))$$

$$f(\mathbf{x}, i) = [\text{emb}(\mathbf{x}_{i-1}), \text{emb}(\mathbf{x}_i), \text{emb}(\mathbf{x}_{i+1})]$$

previous word curr word next word

to Hangzhou today

LSTMs for NER

B-PER I-PER O O O B-LOC O O O B-ORG O O

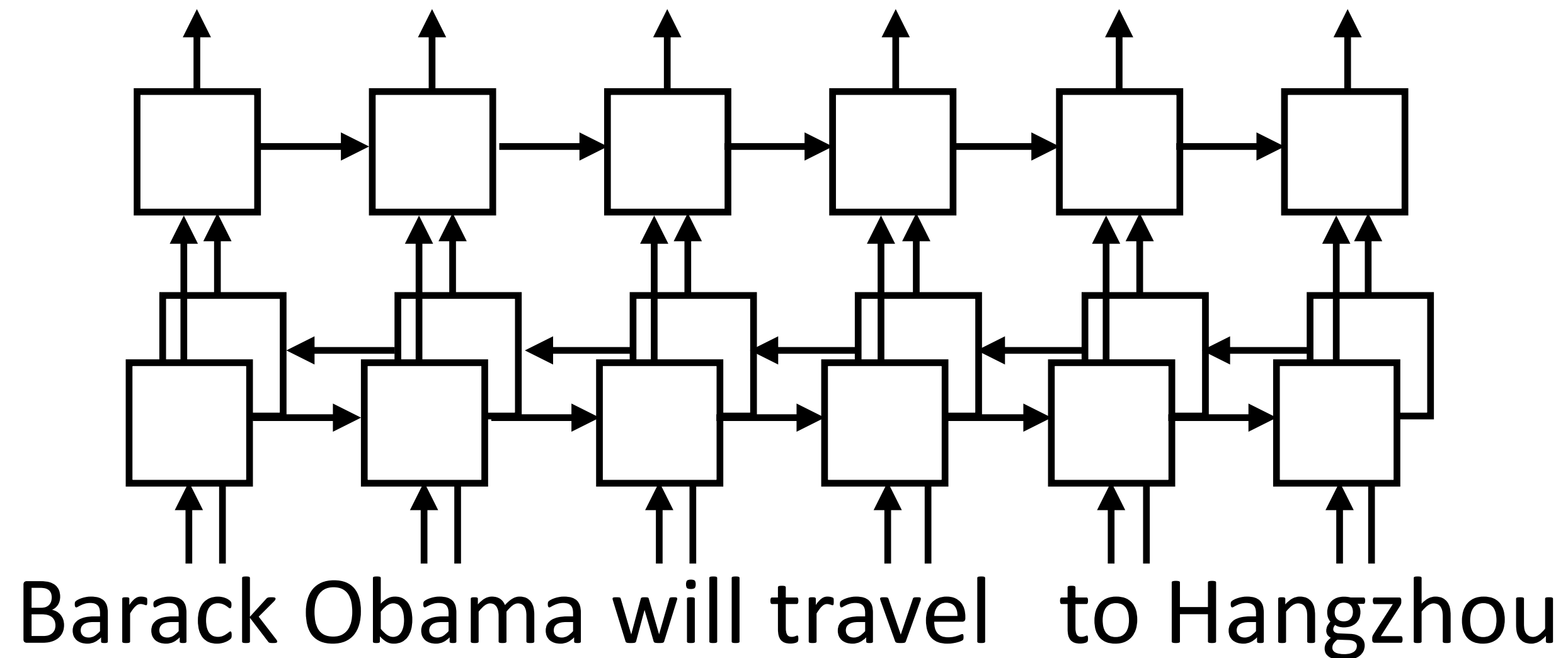
Barack Obama will travel to Hangzhou today for the G20 meeting .

PERSON

LOC

ORG

B-PER I-PER O O O B-LOC

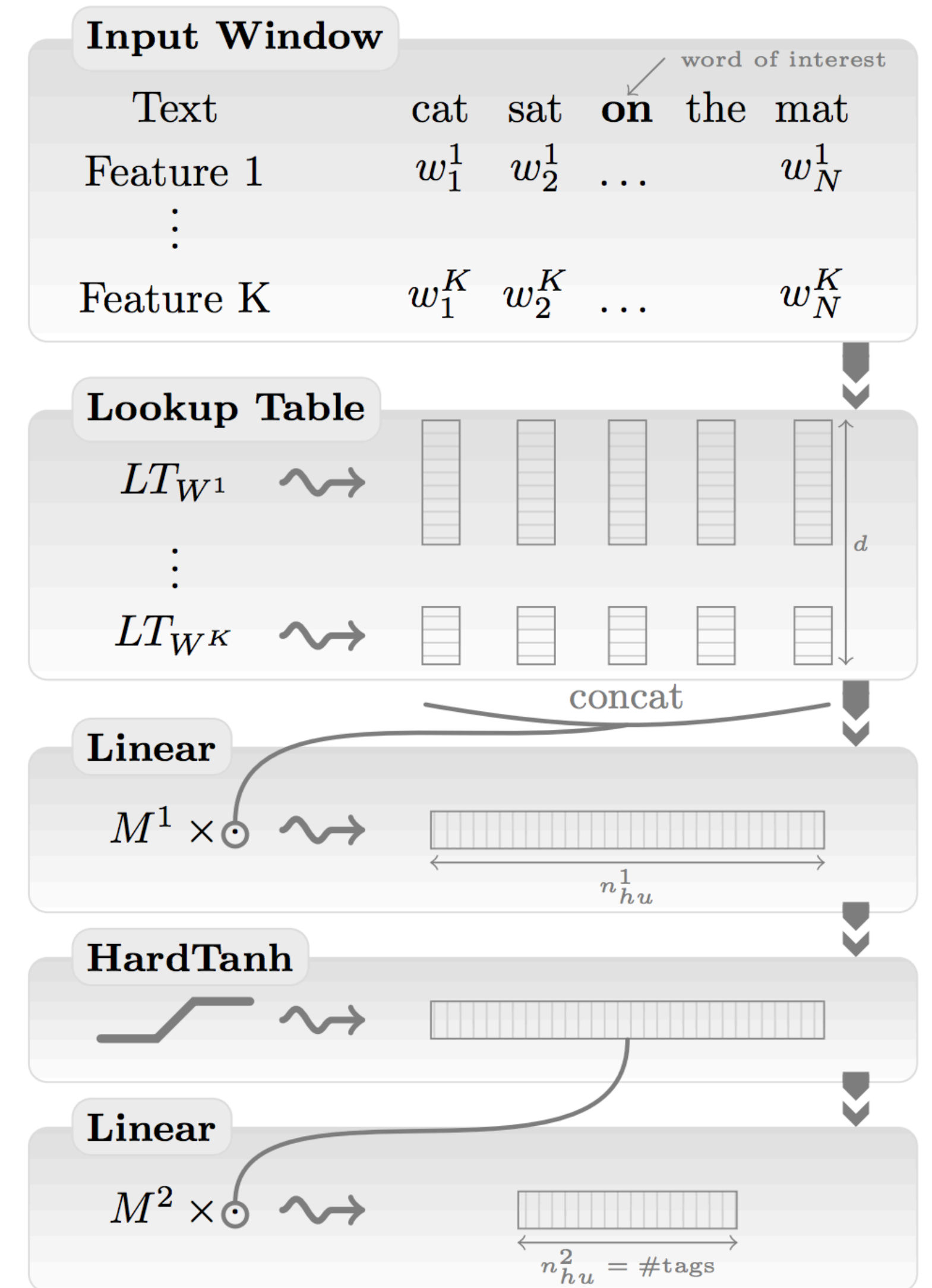


- ▶ How does this compare to neural CRF?

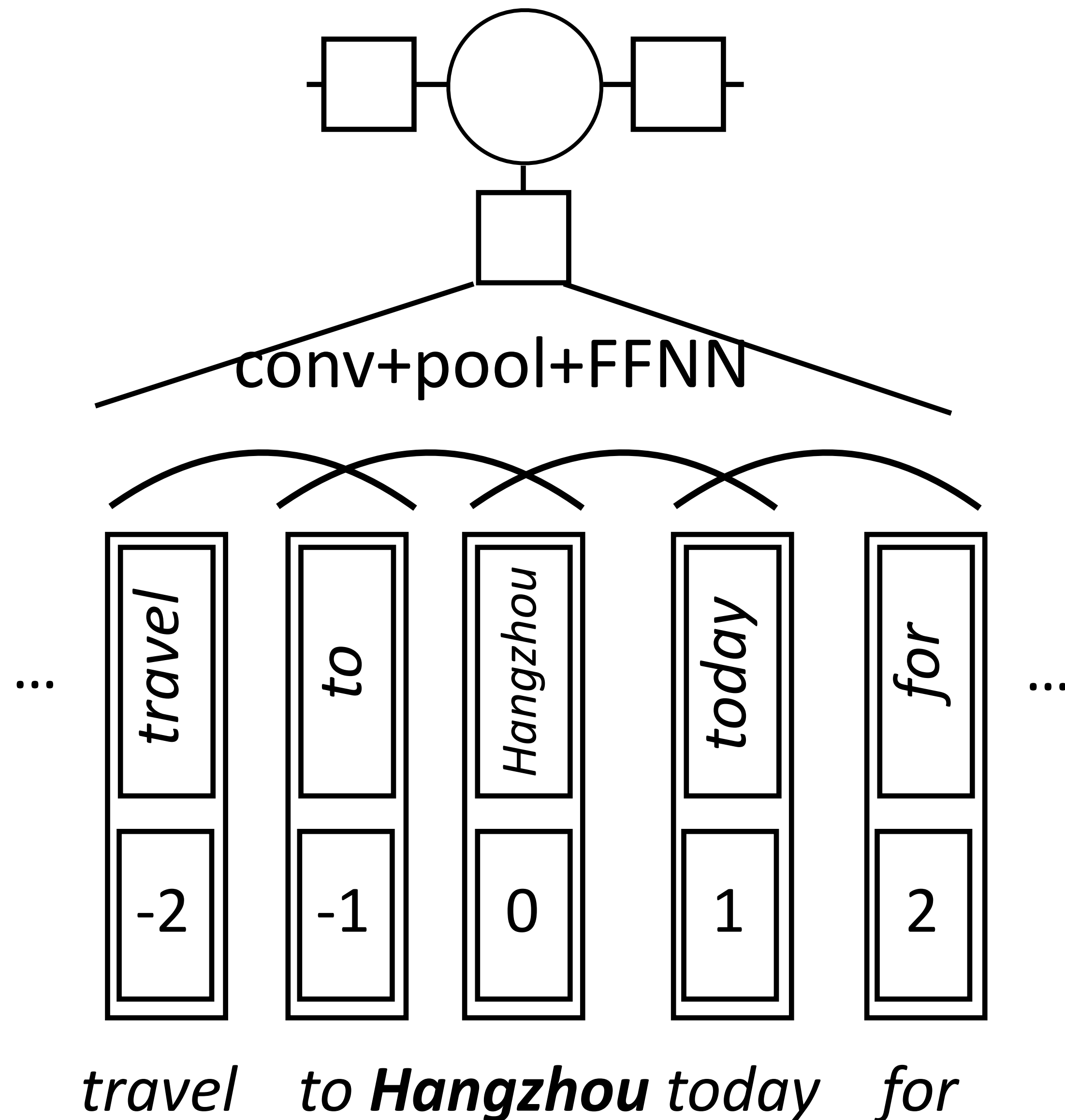
“NLP (Almost) From Scratch”

Approach	POS (PWA)	CHUNK (F1)	NER (F1)	SRL (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99
NN+WLL+LM1	97.05	91.91	85.68	58.18
NN+SLL+LM1	97.10	93.65	87.58	73.84
NN+WLL+LM2	97.14	92.04	86.96	58.34
NN+SLL+LM2	97.20	93.63	88.67	74.15

- ▶ WLL: independent classification; SLL: neural CRF
- ▶ LM2: word vectors learned from a precursor to word2vec/GloVe, trained for 2 weeks (!) on Wikipedia



CNN Neural CRFs



- ▶ Append to each word vector an *embedding of the relative position* of that word
- ▶ Convolution over the sentence produces a position-dependent representation

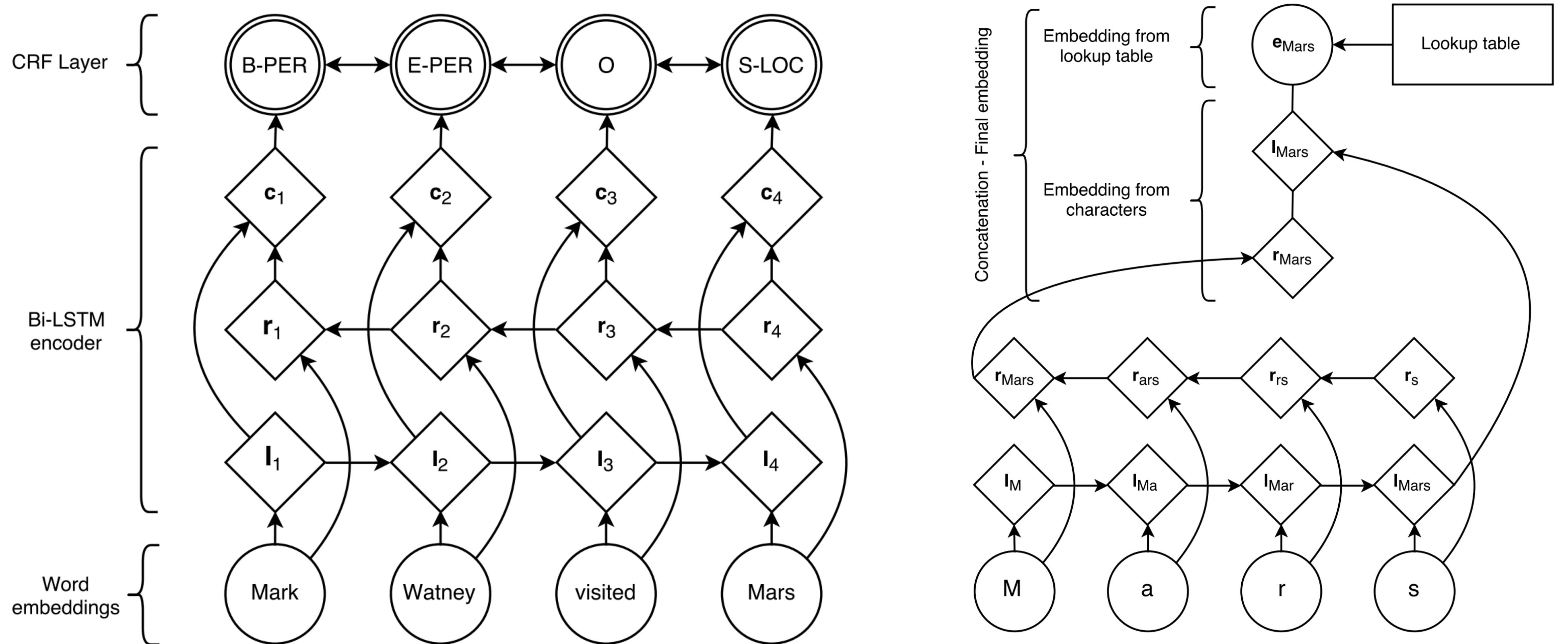
CNN NCRFs vs. FFNN NCRFs

Approach	POS (PWA)	CHUNK (F1)	NER (F1)	SRL (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
	<i>Window Approach</i>			
NN+SLL+LM2	97.20	93.63	88.67	–
	<i>Sentence Approach</i>			
NN+SLL+LM2	97.12	93.37	88.78	74.15

- ▶ Sentence approach (CNNs) is comparable to window approach (FFNNs) except for SRL where they claim it works much better

Neural CRFs with LSTMs

- ▶ Neural CRF using character LSTMs to compute word representations



Chiu and Nichols (2015), Lample et al. (2016)

Neural CRFs with LSTMs

- ▶ Chiu+Nichols: character CNNs instead of LSTMs
- ▶ Lin/Passos/Luo: use external resources like Wikipedia
- ▶ LSTM-CRF captures the important aspects of NER: word context (LSTM), sub-word features (character LSTMs), outside knowledge (word embeddings)

Model	F ₁
Collobert et al. (2011)*	89.59
Lin and Wu (2009)	83.78
Lin and Wu (2009)*	90.90
Huang et al. (2015)*	90.10
Passos et al. (2014)	90.05
Passos et al. (2014)*	90.90
Luo et al. (2015)* + gaz	89.9
Luo et al. (2015)* + gaz + linking	91.2
Chiu and Nichols (2015)	90.69
Chiu and Nichols (2015)*	90.77
LSTM-CRF (no char)	90.20
LSTM-CRF	90.94

Takeaways

- ▶ CNNs are a flexible way of extracting features analogous to bag of n-grams, can also encode positional information
- ▶ All kinds of NNs can be integrated into CRFs for structured inference. Can be applied to NER, other tagging, parsing, ...