CS378: Natural Language Processing

Lecture 16: Neural Network (Sequence) Continued

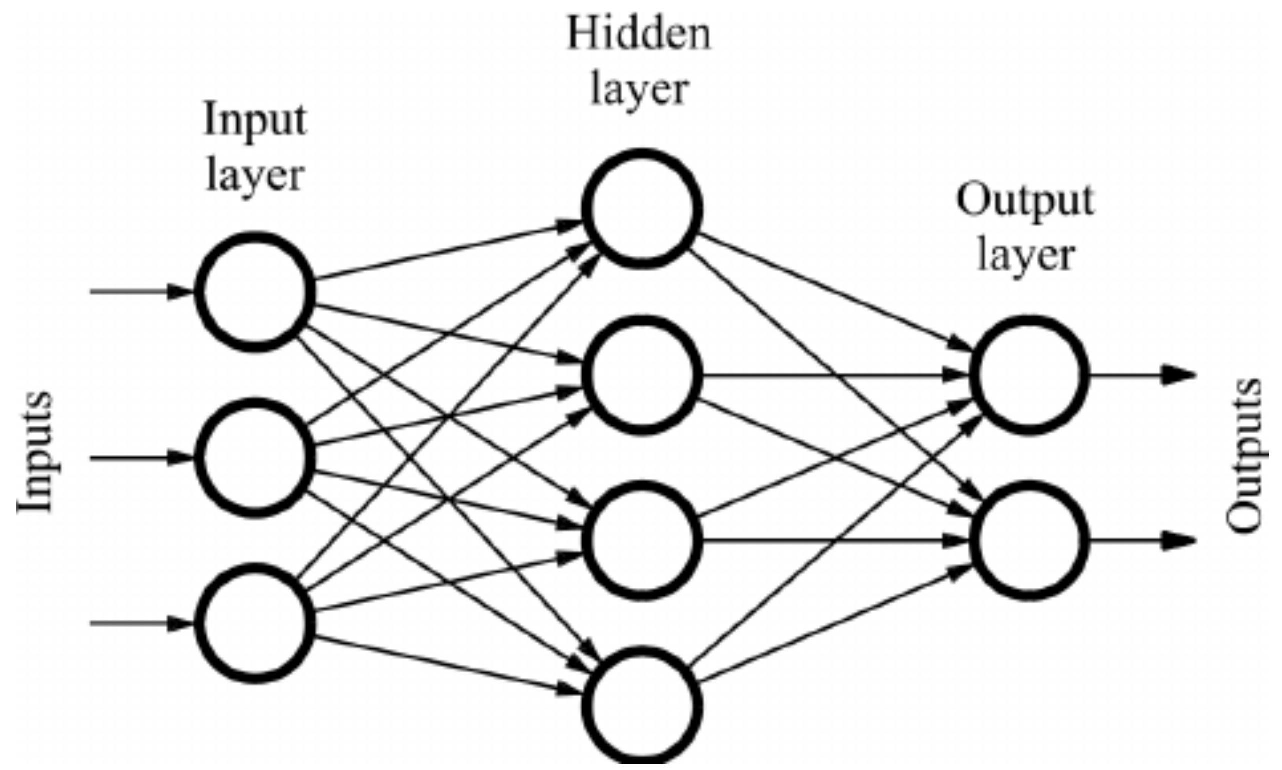
The University of Texas at Austin

Eunsol Choi

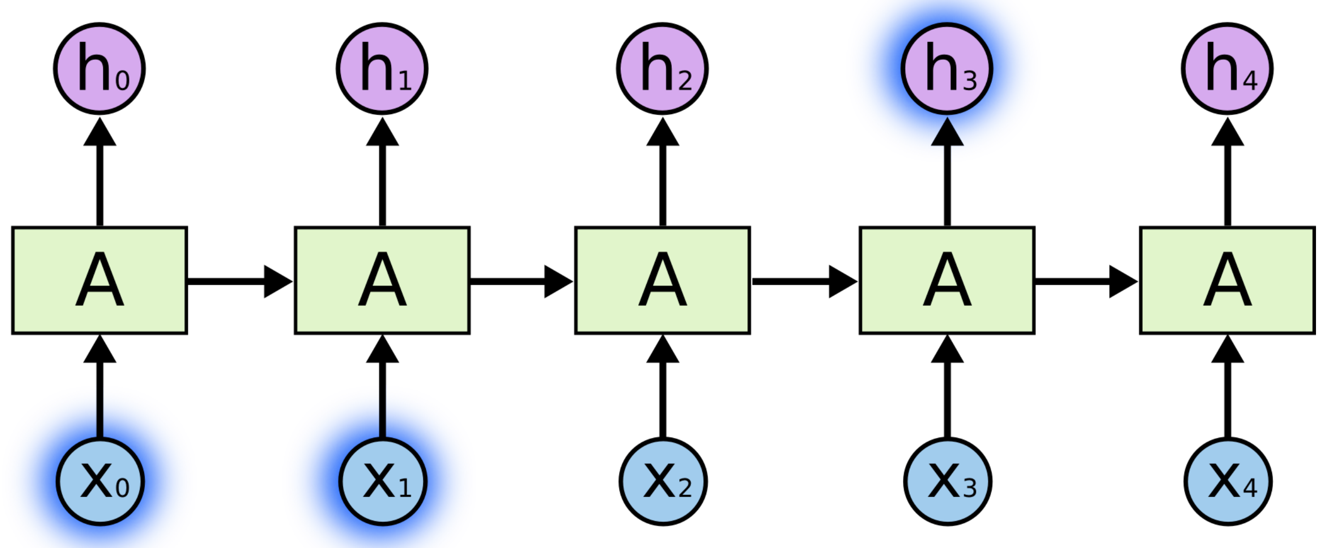
Slides from Greg Durrett, Yoav Artzi, Yejin Choi, Princeton NLP

Neural Networks in NLP

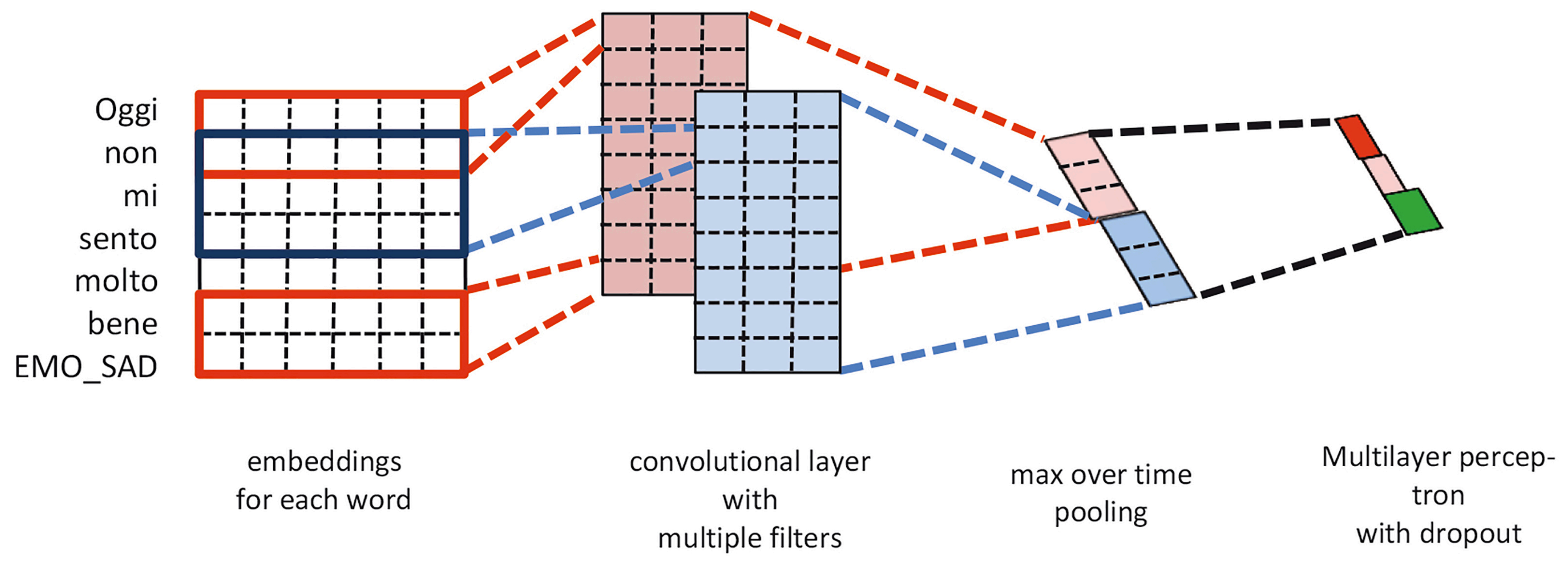
Feed-forward NNs



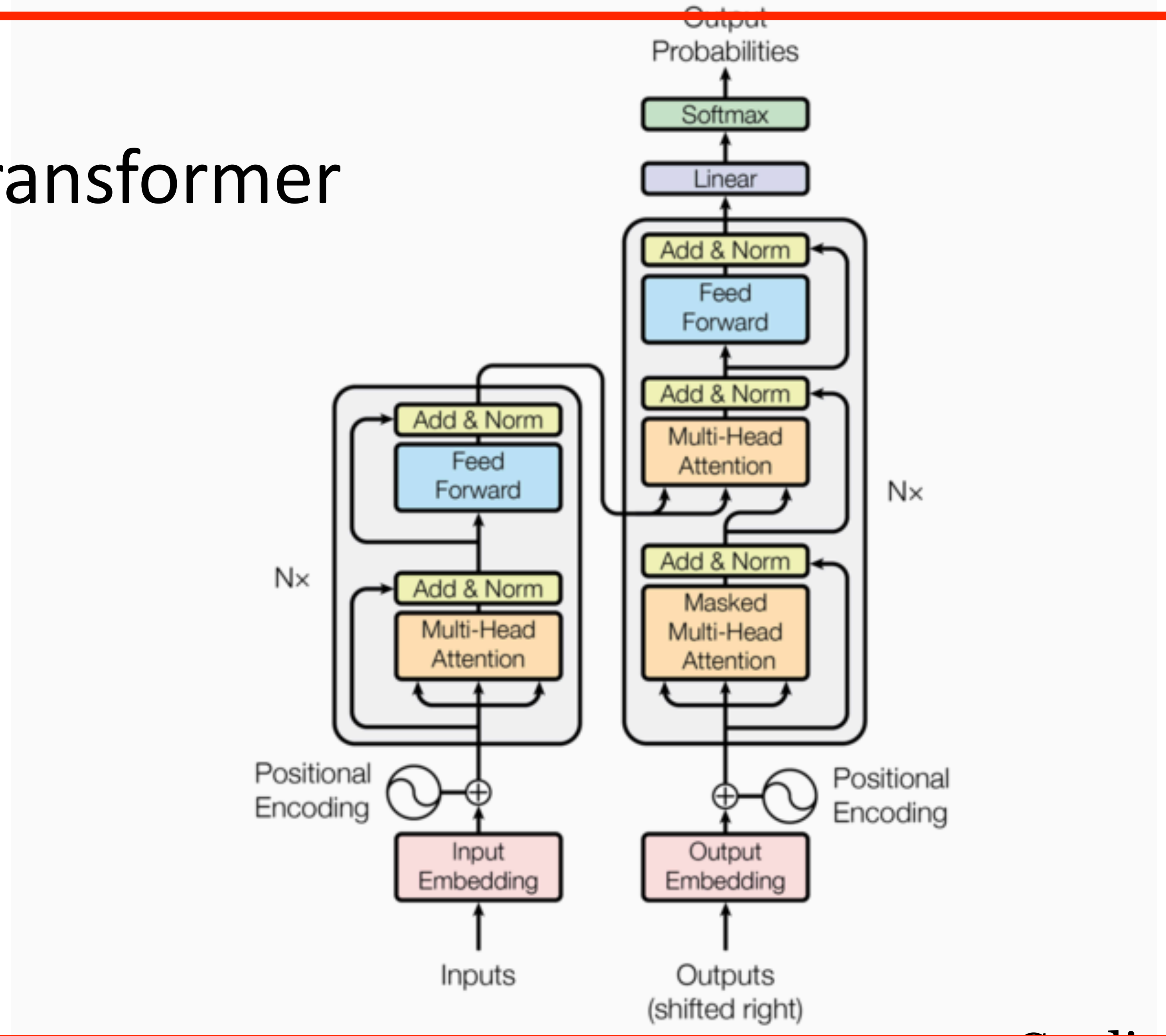
Recurrent NNs



Convolutional NNs



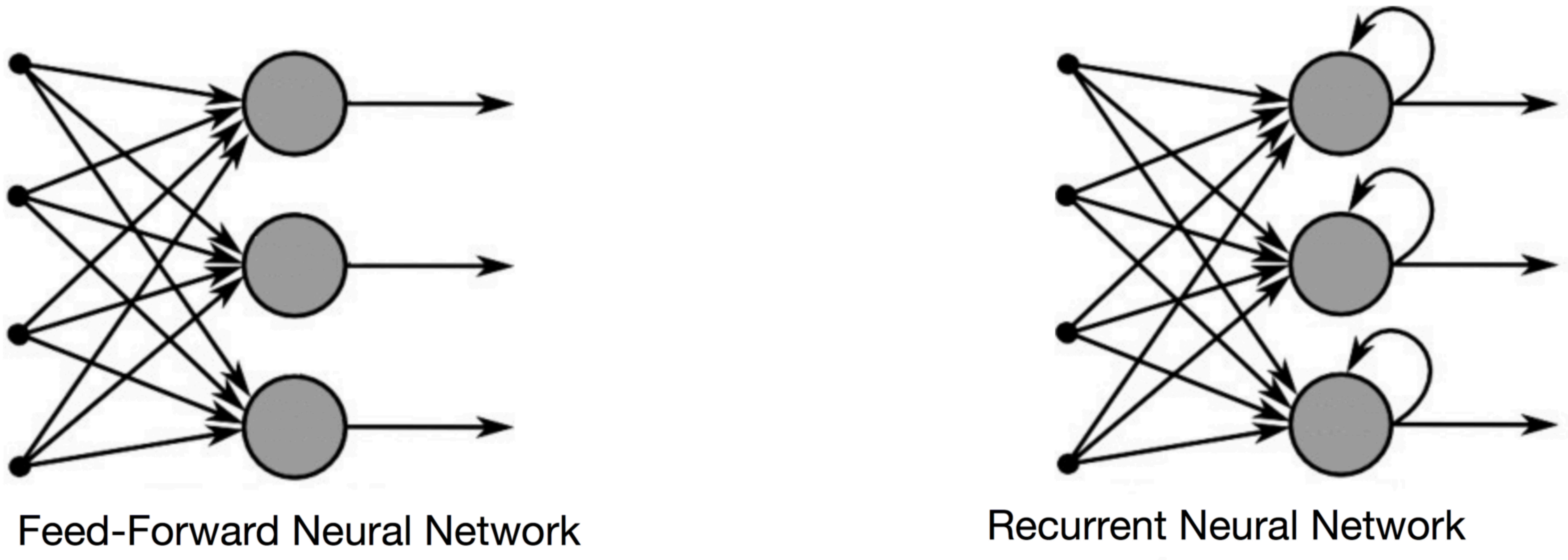
Transformer



Always coupled with word embeddings…

Credits: Princeton NLP course

Recap: RNNs



- Maps from dense input sequence to dense hidden state representation sequence

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \rightarrow h_1, \ldots, h_n$$

- Simple definition of R: $R(h_{i-1}, x_i) = \tanh(Wx_i + Vh_{i-1} + b)$

Recap: RNNs

- $S = \mathbb{R}^{d_{hid}}$ - hidden state space $(h_1, h_2 \ldots)$
- $\Sigma = \mathbb{R}^{d_{in}}$ - input state space $(x_1, x_2 \ldots.)$
- $s_0 \in S$ - initial state vector (h_0)
- $R : \mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{hid}} \to \mathbb{R}^{d_{hid}}$ - transition function

- Maps from dense input sequence to dense hidden state representation sequence

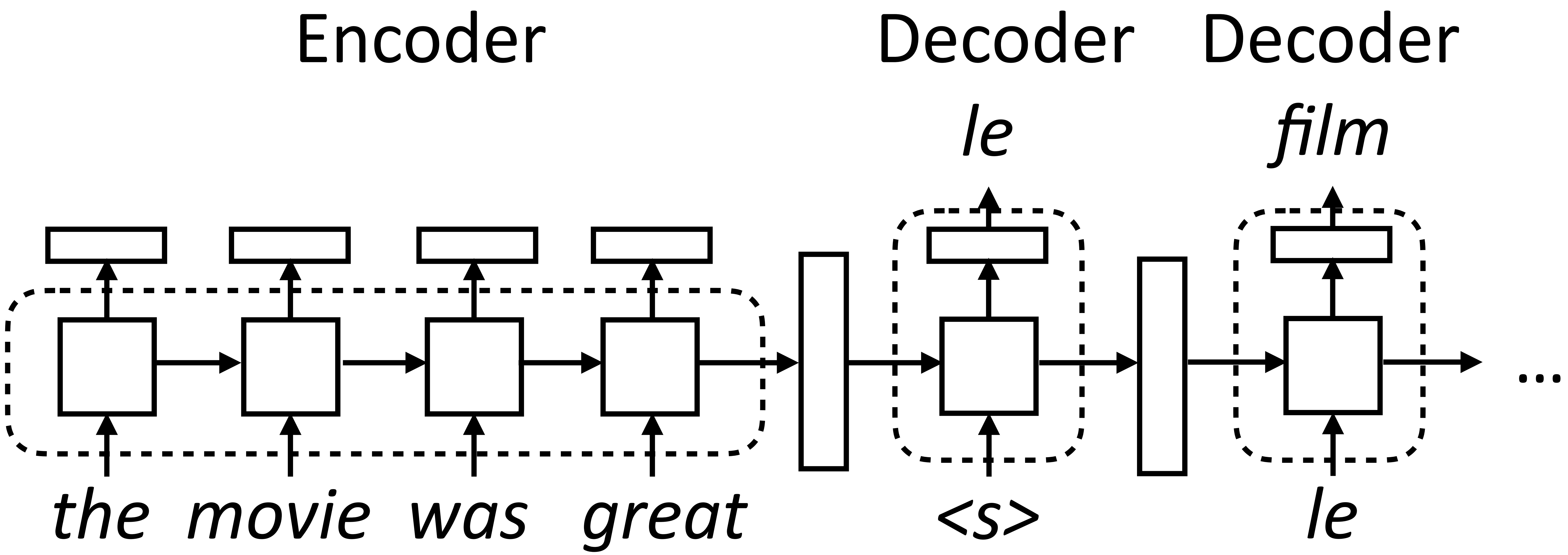
$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \to h_1, \ldots, h_n$$

- For all $i \in \{1, \ldots, n\}$,
  - $h_i = R\left(h_{i-1}, \boldsymbol{x}_i\right)$
  - Simple definition of R: $R(h_{i-1}, x_i) = \tanh(Wx_i + Vh_{i-1} + b)$

- R is parameterized, where the parameters are shared across all steps.

$$h_4 = R\left(h_3, \boldsymbol{x}_4\right) = \ldots = R(R\left(R\left(R\left(h_0, \boldsymbol{x}_1\right), \boldsymbol{x}_2\right), \boldsymbol{x}_3\right), \boldsymbol{x}_4)$$

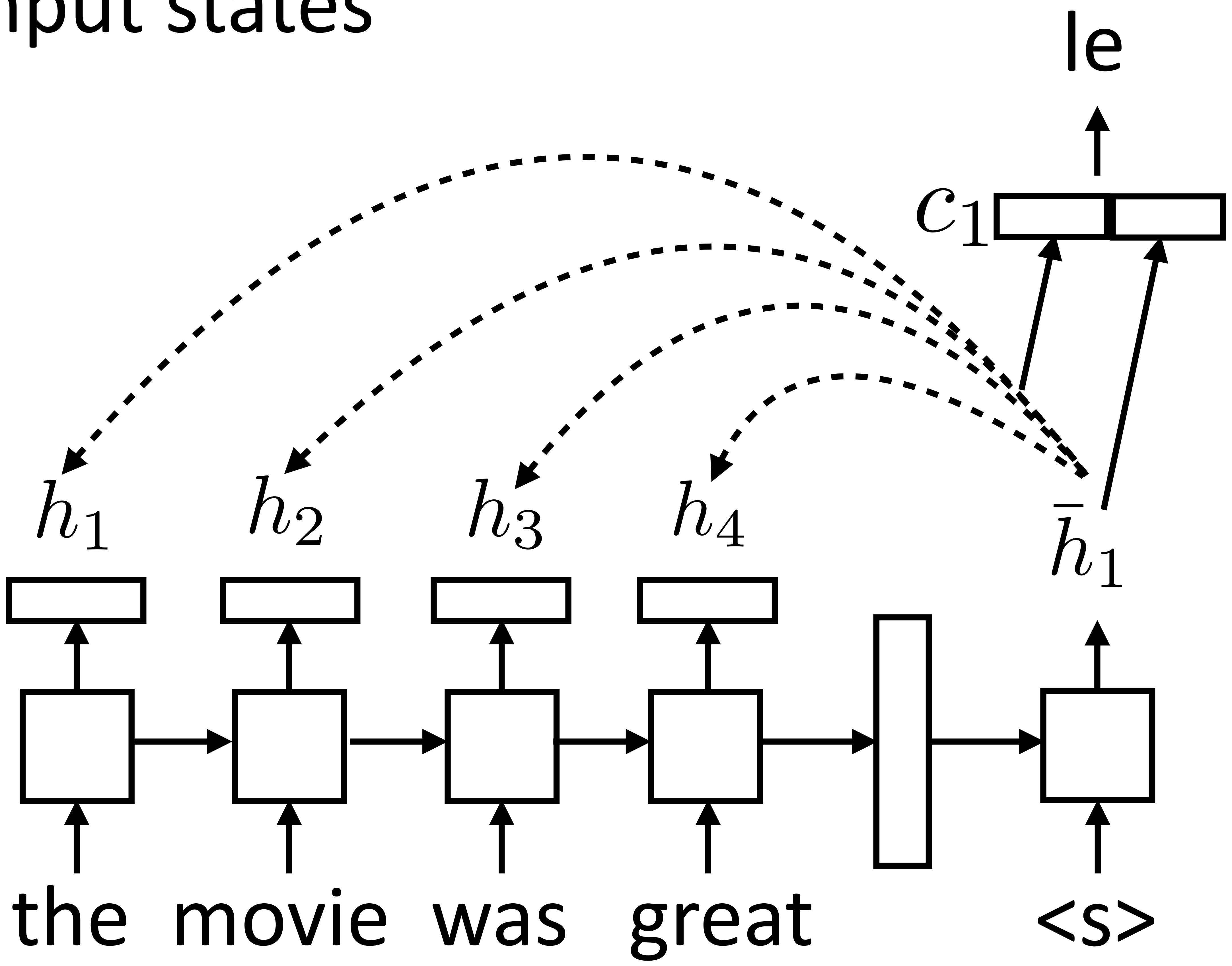
Recap: Seq2Seq Models



- Encoder: a RNN encoding a sequence of tokens, produces a vector.
- Decoder: separate RNN module (***different*** parameters).
  - Takes two inputs: hidden state and previous token.
  - Outputs token and a new hidden state.

Recap: Attention

- For each decoder state, compute weighted sum of input states



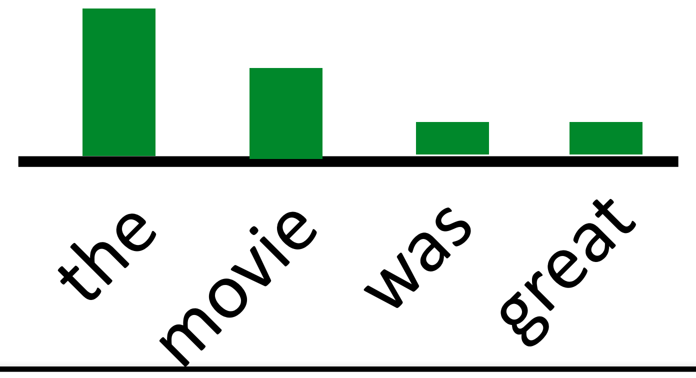
- No attn: $P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h}_i)$

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

Weighted sum of input hidden states (vector)

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



Attention weight for input x_j at decoding y_i

$$e_{ij} = f(\bar{h}_i, h_j)$$

- Some function f (next slide)

Limitations of RNN

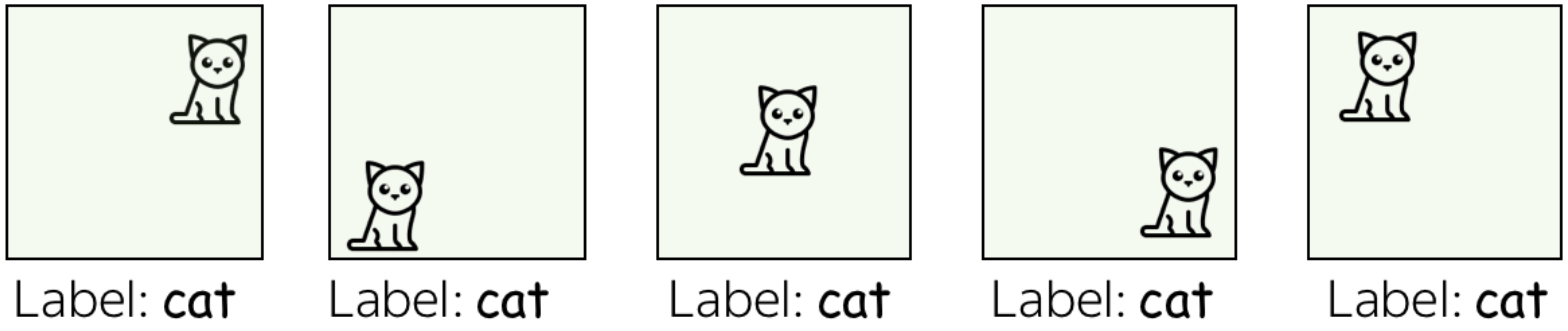
- You have to process input sequentially (has to process $x_{t-1}, x_{t-2} \ldots, x_1$) to process x_t
- Does it have to be this way?

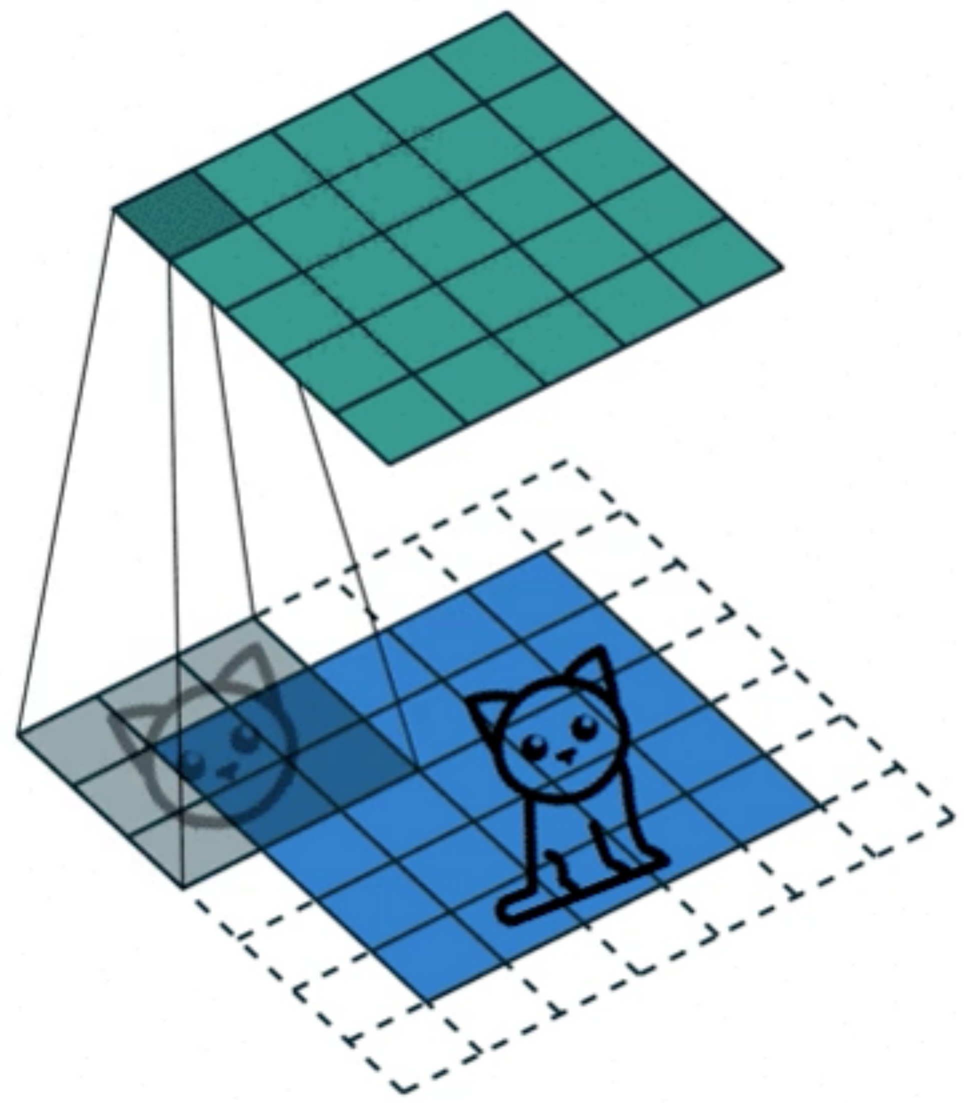
Neural Network for Sequence

- Recurrent Neural Network (RNN)
  - LSTM, GRU
- Encoder-Decoder model
  - Output is also a variable length sequence
  - Attention mechanism
- **Variants of Neural Network**
  - **Convolutional Neural Network**
  - **Transformer**

Convolutional Neural Network

- Computer vision neural network architecture
- Scan the input piece-by-piece
- Can handle input of different sizes with few parameters





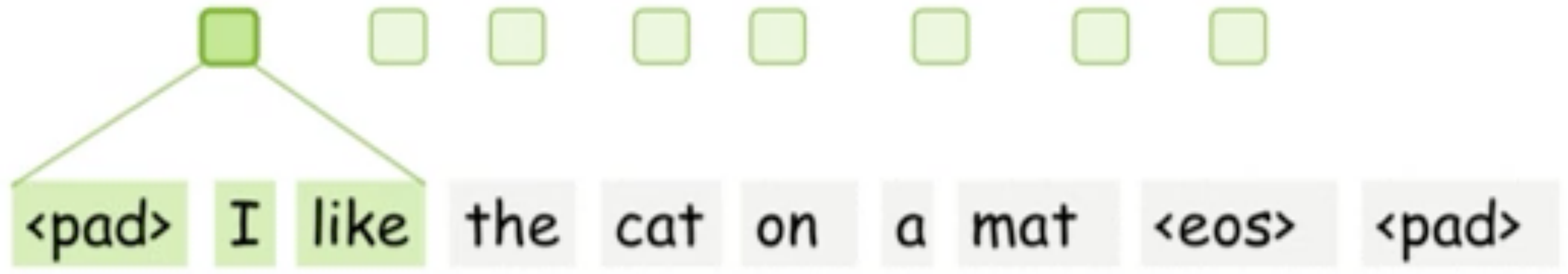
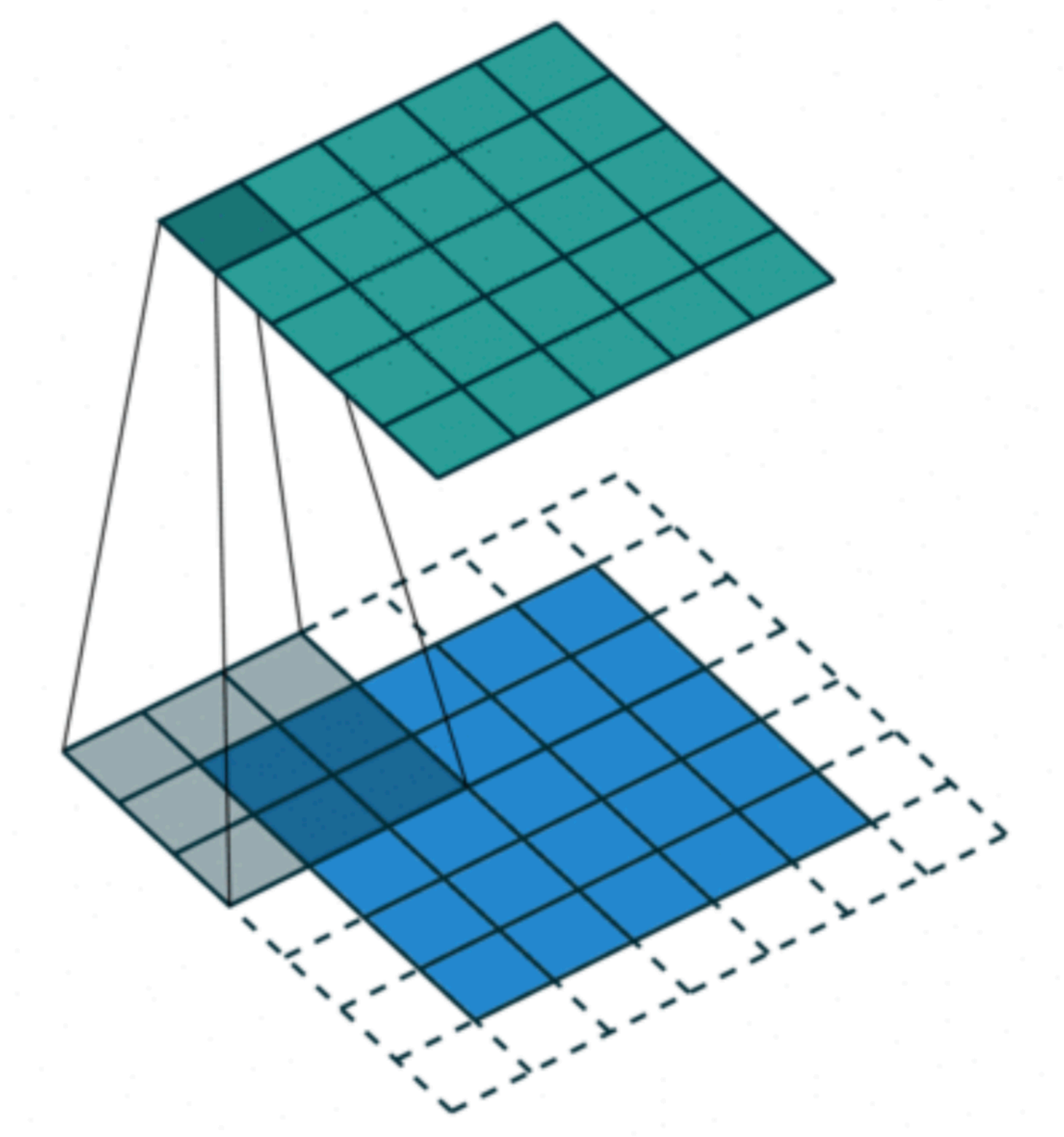


Image credit: https://lena-voita.github.io/nlp_course/models/convolutional.html

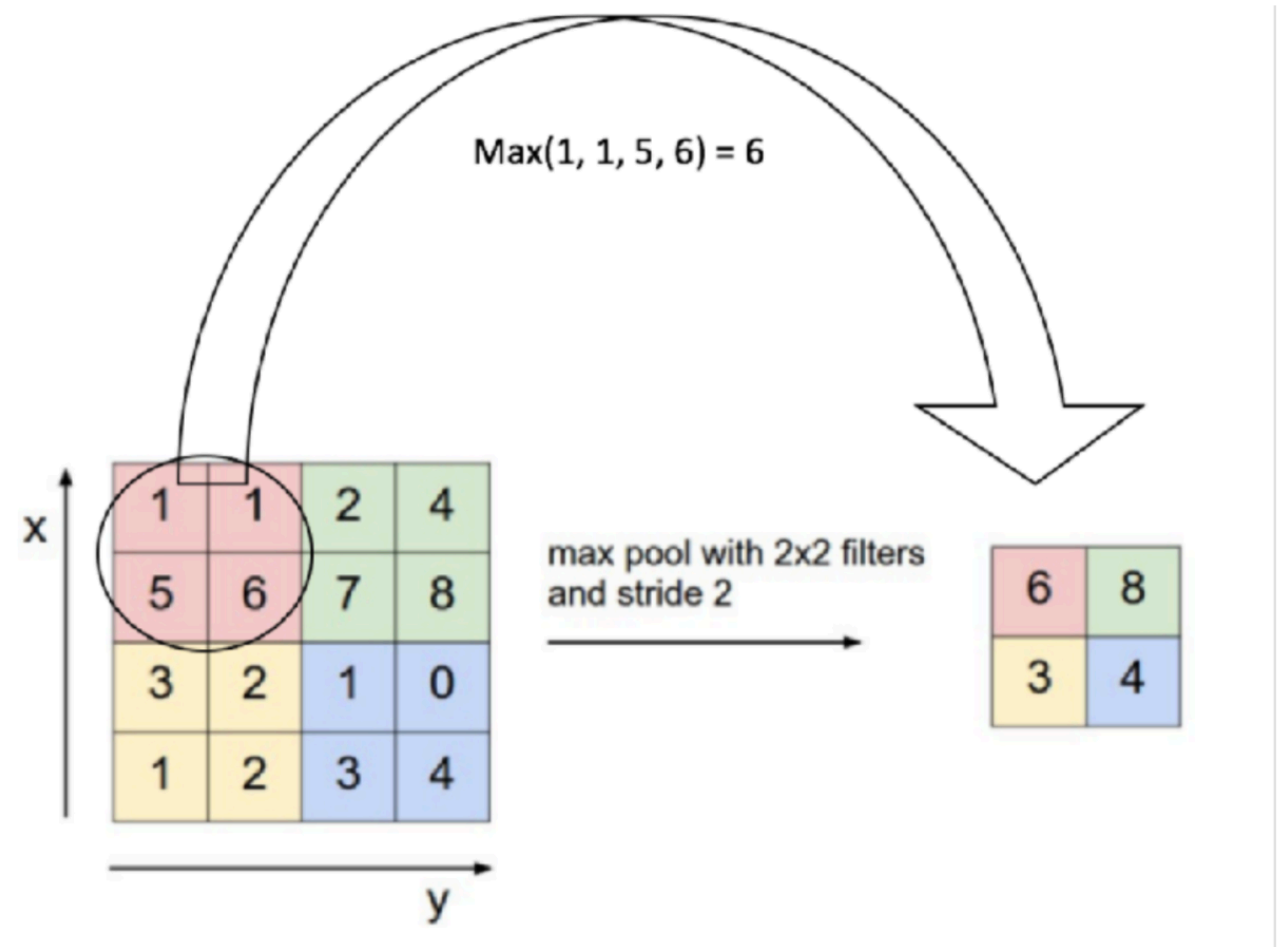
Convolutional Neural Network (CNN)

- Two main operations:

- Convolution (parametrized)

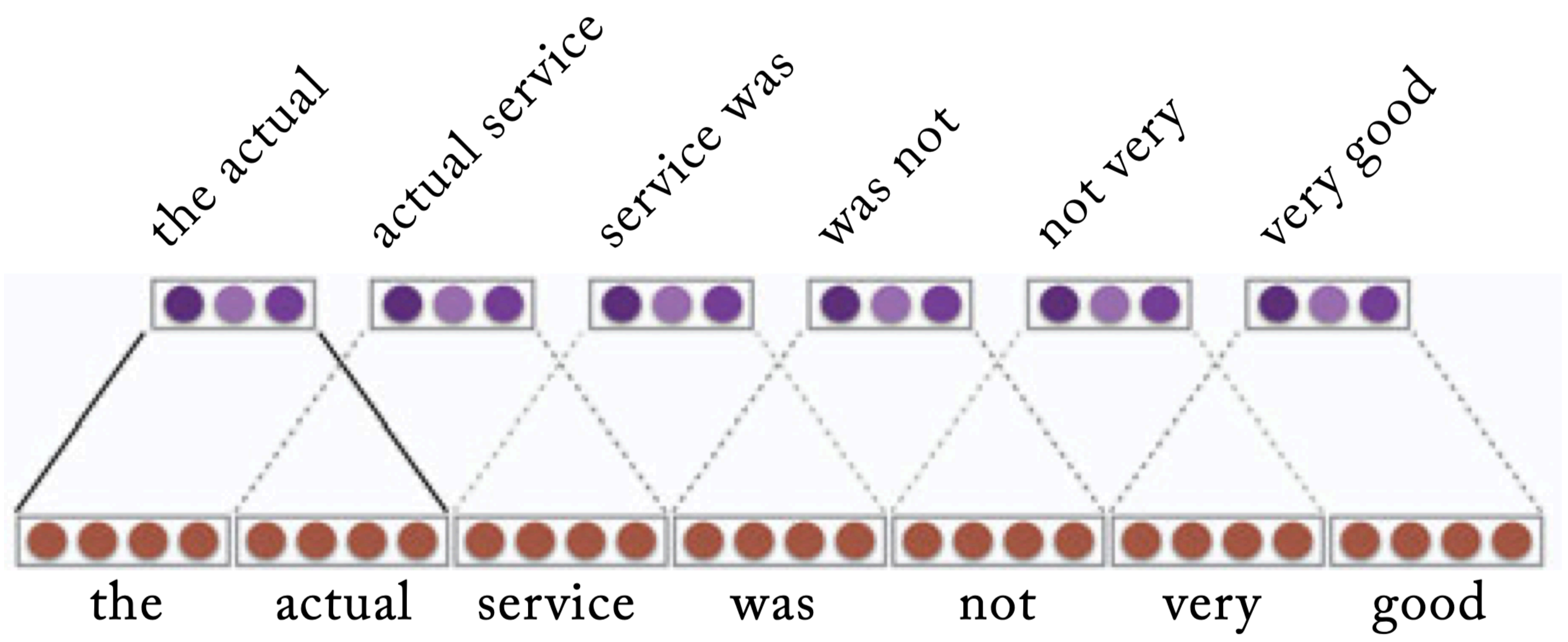


- Pooling (non-parametrized)

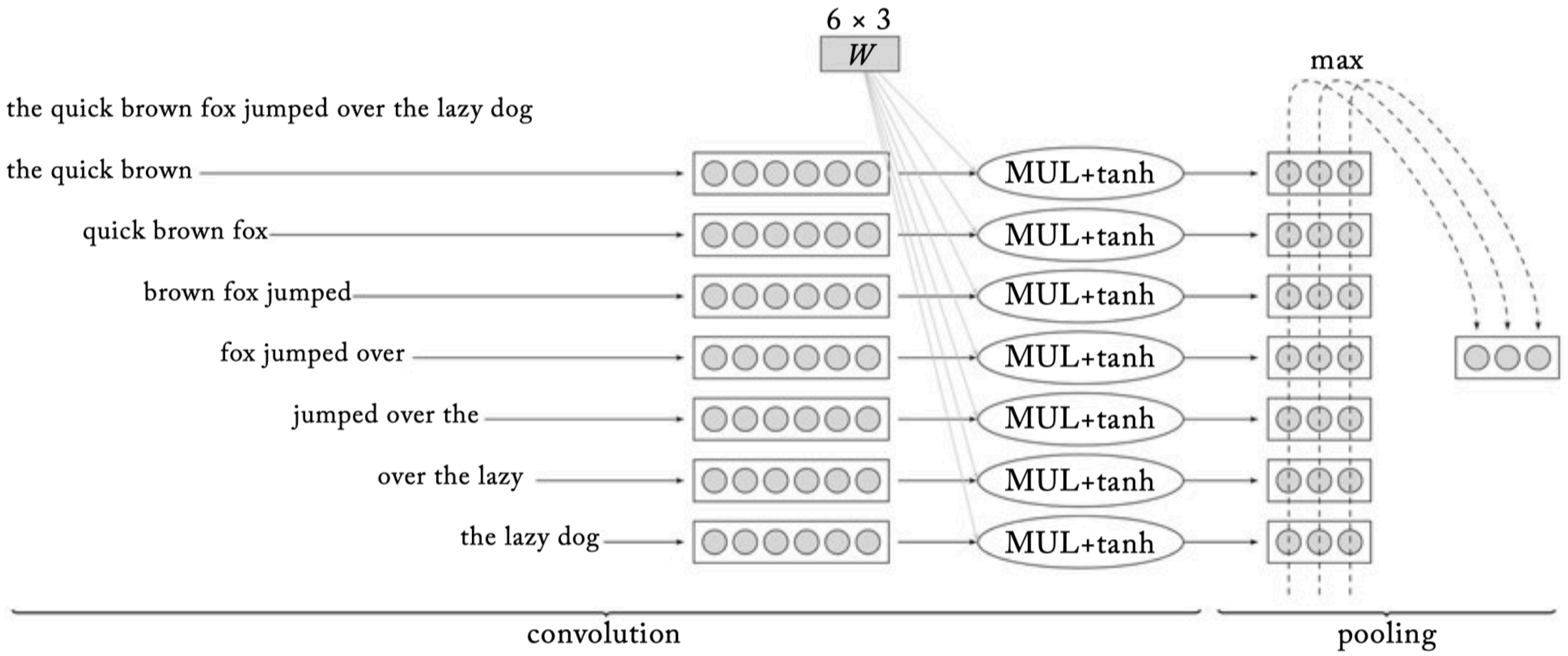


CNN - applied to text classification

- Map (filter) each k-gram to a vector



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



Convolution Layer

ϕ – embedding function

$\bar{x}$ – sentence

$\mathbf{u}$ – filter, a weight vector

$\bar{x} = \langle x_1, \ldots, x_n \rangle$

$\mathbf{x}_i = \phi(x_i)$

$p_i = g([\mathbf{x}_i; \ldots; \mathbf{x}_{i+k-1}] \cdot \mathbf{u})$

Non-linearity function

[input embedding dimension * k]

- Map sequence into a shorter sequence (of a fixed window size, k)
- Map (filter) each k-gram to a single number
- Without padding, the output will be of (n-k+1) dimension

Multiple Filters

$$\mathbf{U} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_l \\ | & | & & | \end{bmatrix}$$ – matrix of l filters, each is a column

ϕ – embedding function

$\bar{x}$ – sentence

$\bar{x} = \langle x_1, \dots, x_n \rangle$

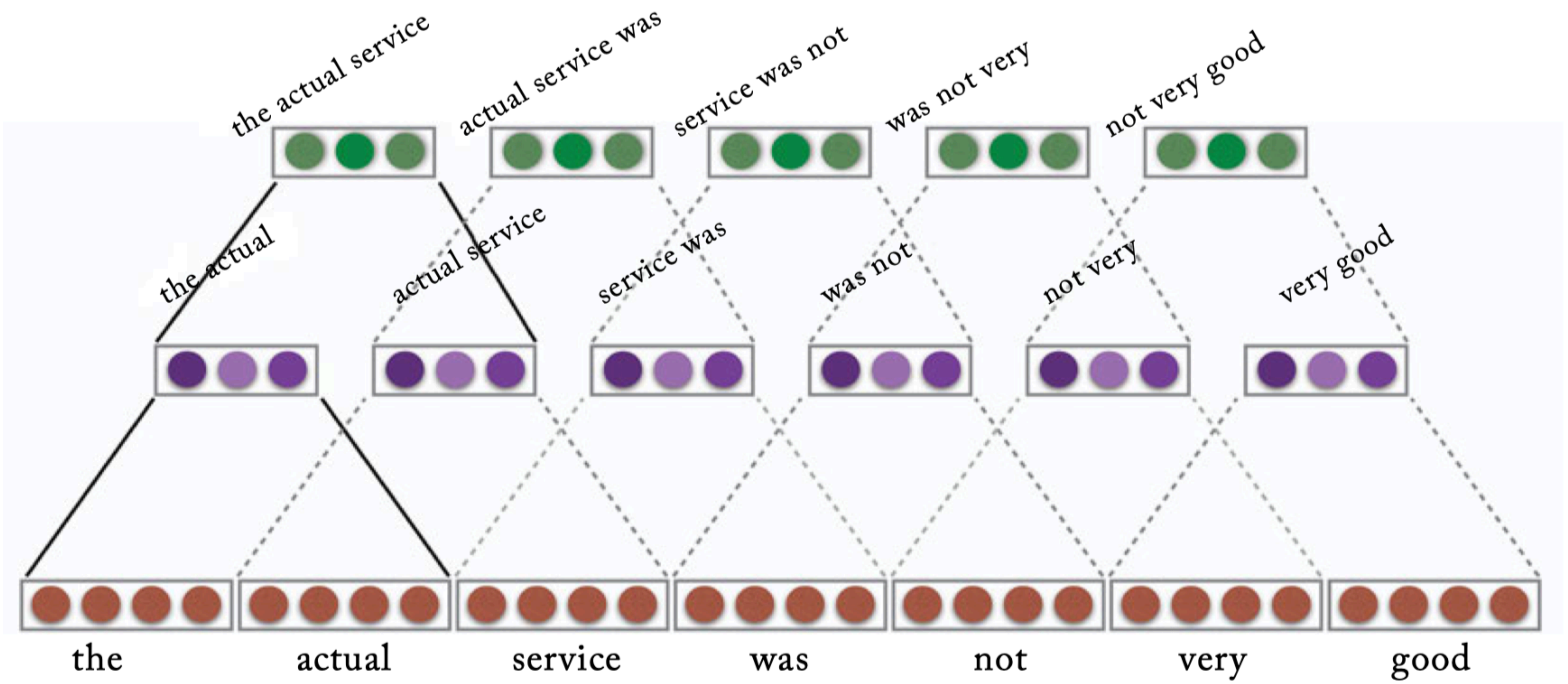
$\mathbf{x}_i = \phi(x_i)$

$\mathbf{p}_i = g([\mathbf{x}_i; \dots; \mathbf{x}_{i+k-1}] \cdot \mathbf{U})$

- Each filter captures different patterns
- Map each k-gram into *l*-dimensional vector

Hierarchical CNN

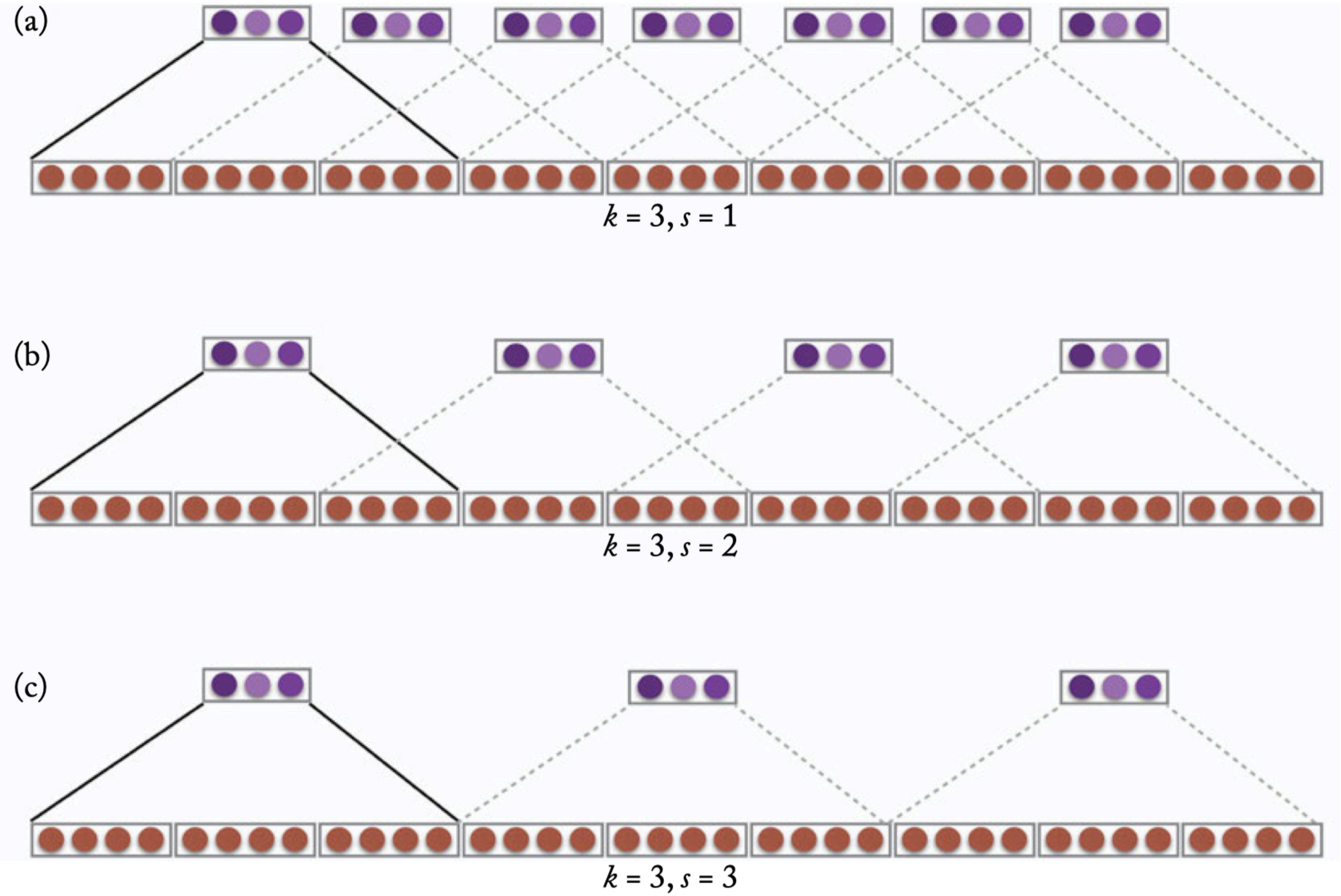
- Stack convolutional layers
- Capture increasingly wider context



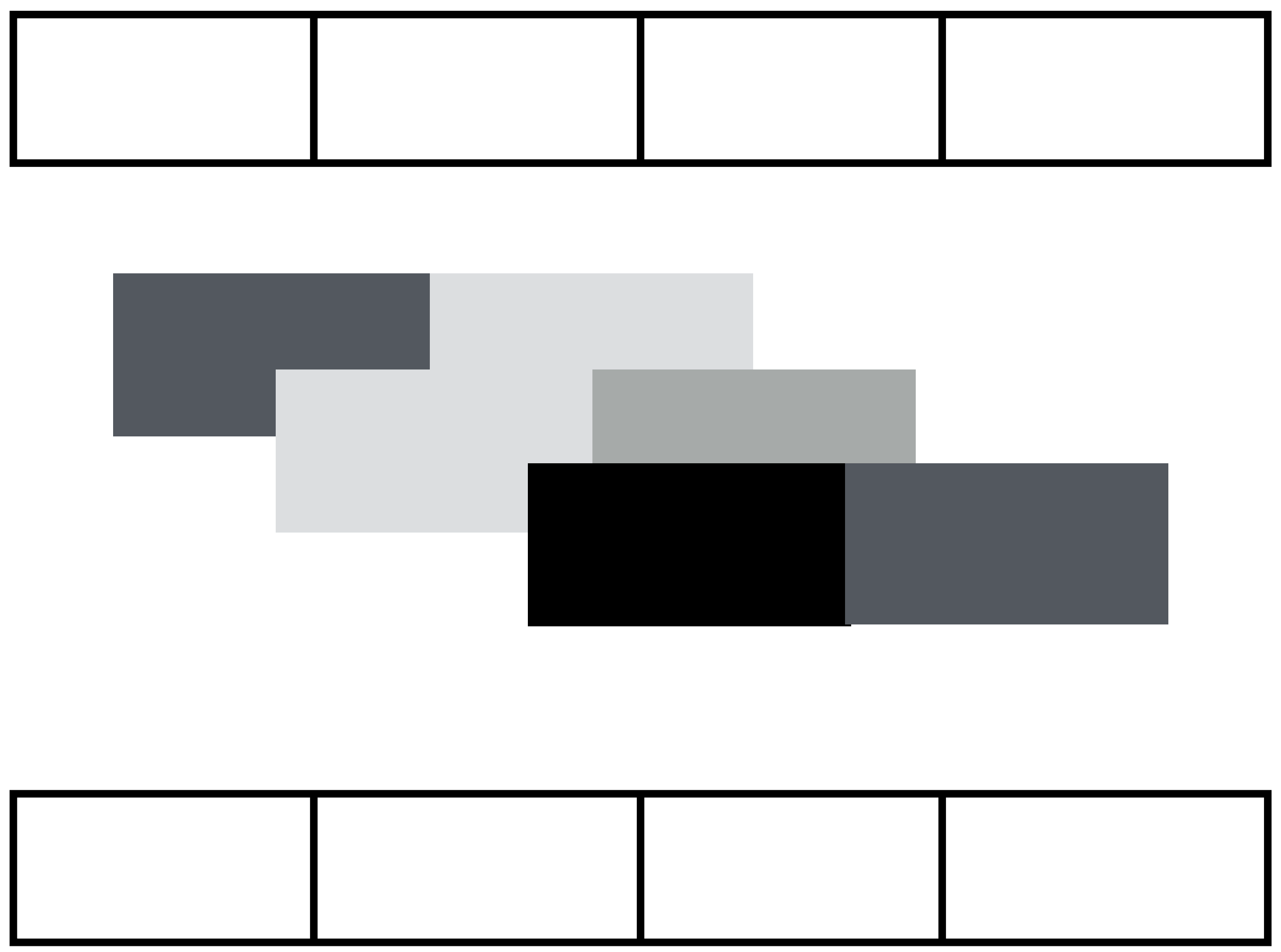
Strides

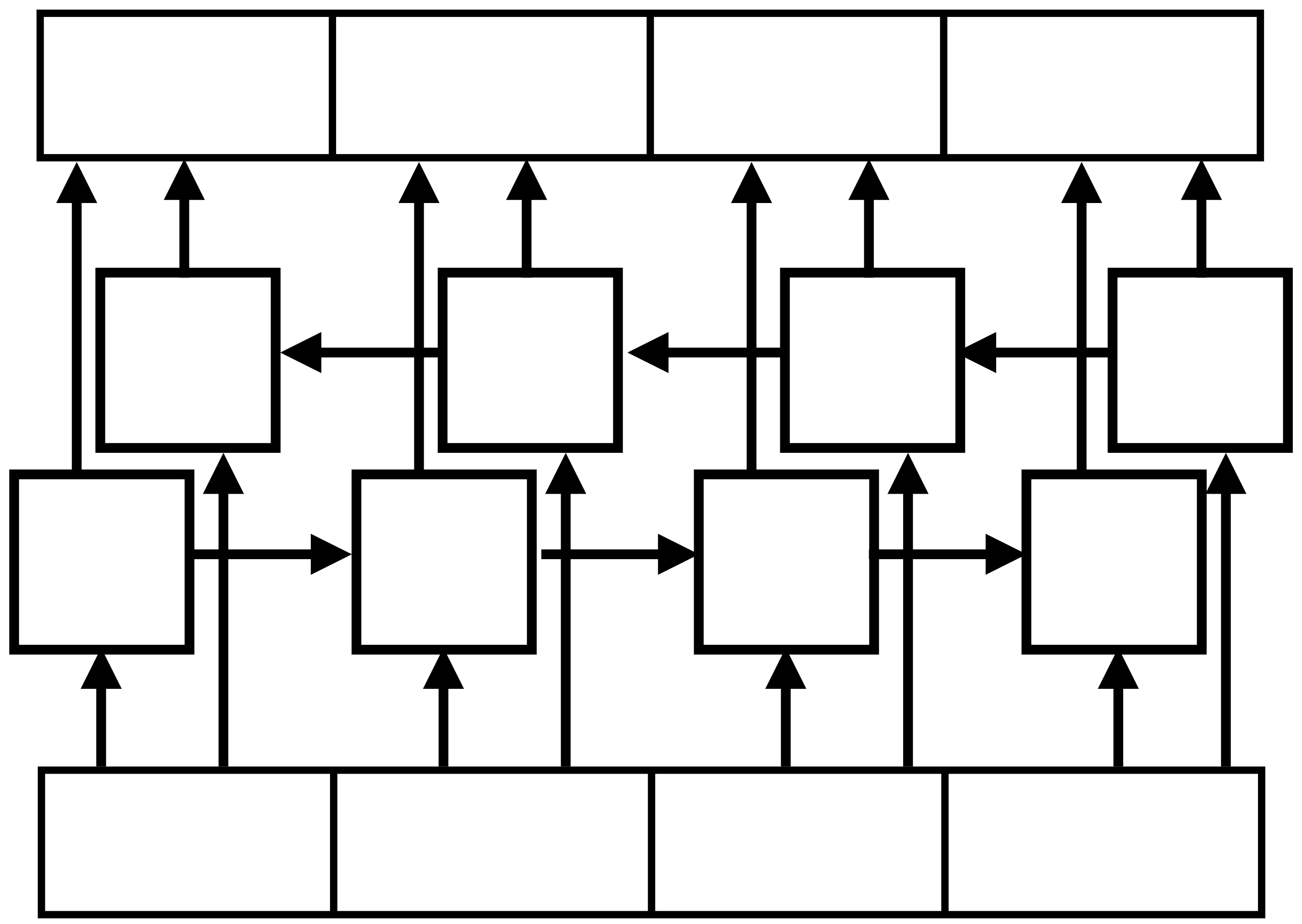
- So far we have seen stride of 1

- Larger stride is also possible (skipping some k-grams)



Comparison: CNNs vs. RNNs

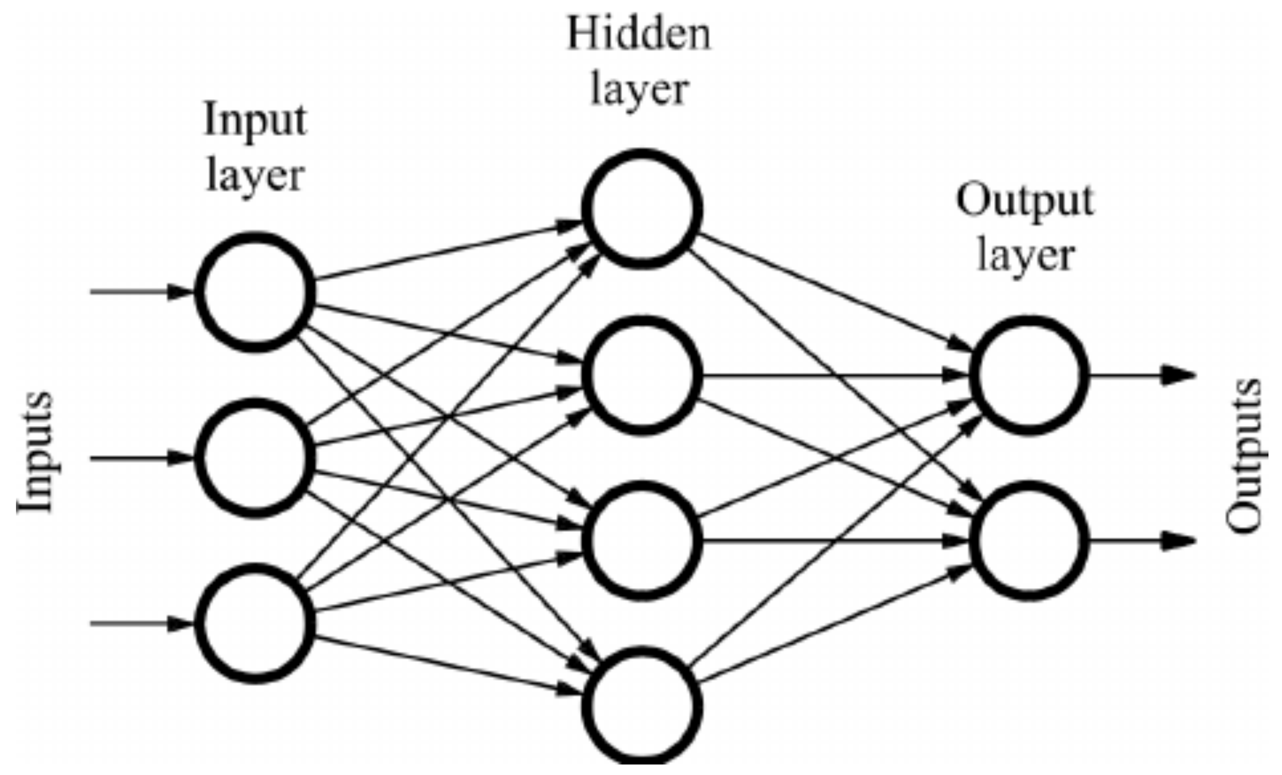




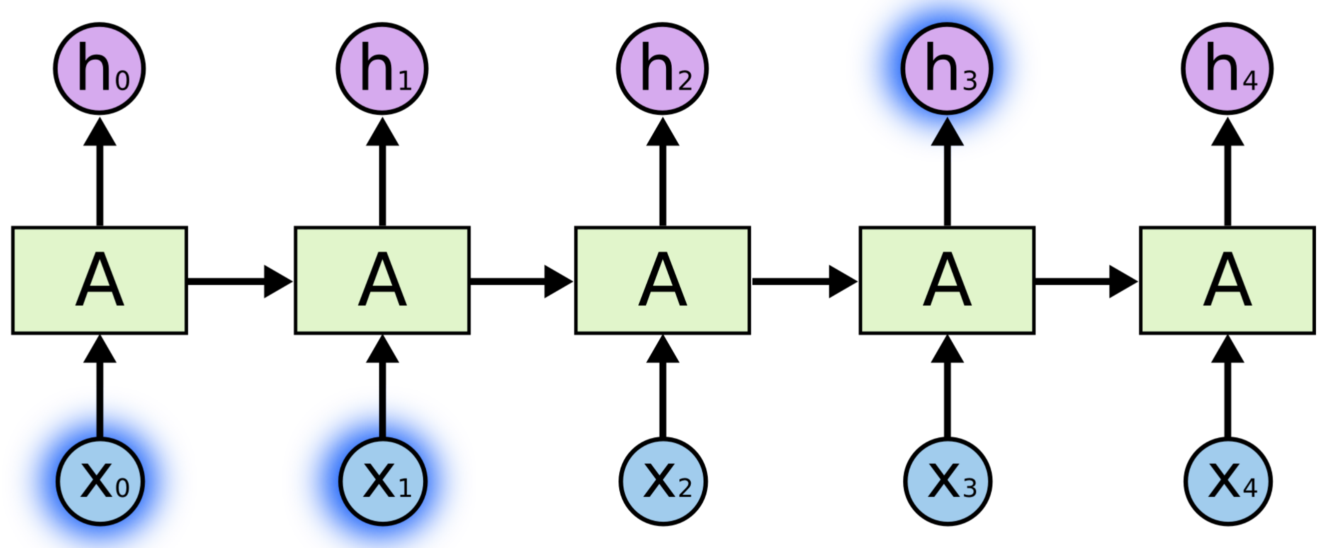
- Both RNNs and convolutional layers transform the input using context
- RNN: “globally” looks at the entire sentence (but local for many problems)
- CNN: local depending on filter width + number of layers

Neural Networks in NLP

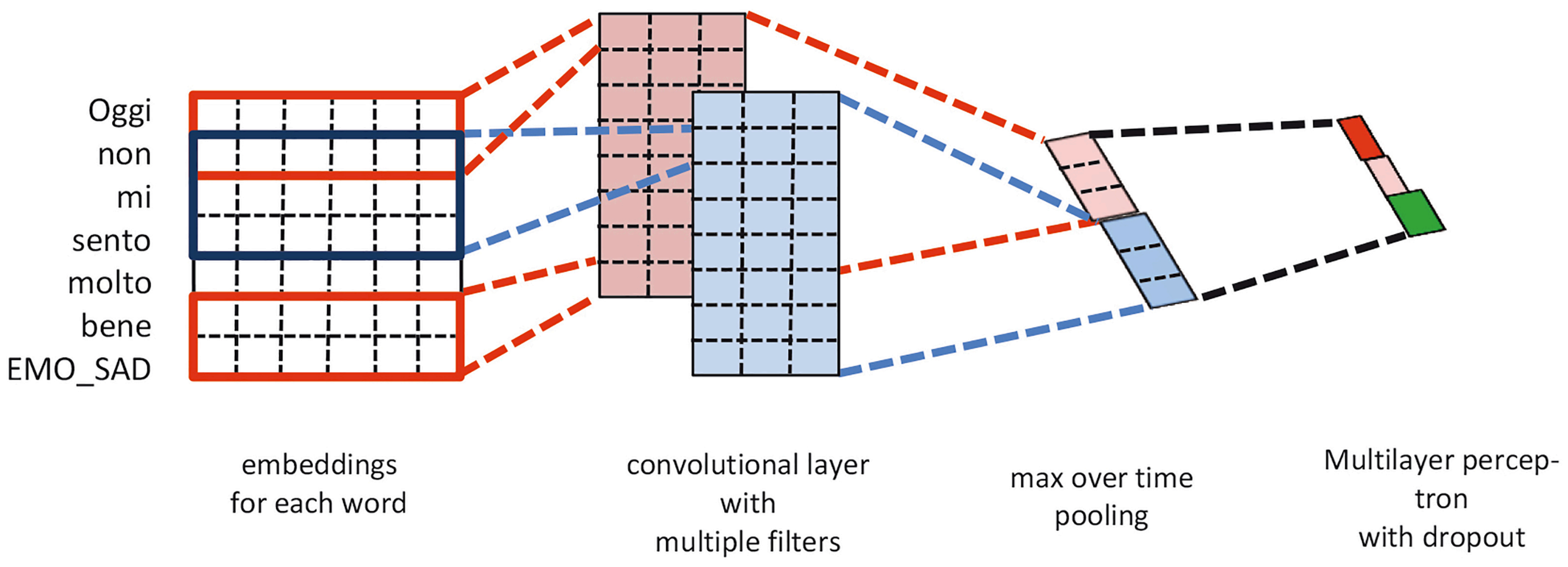
Feed-forward NNs



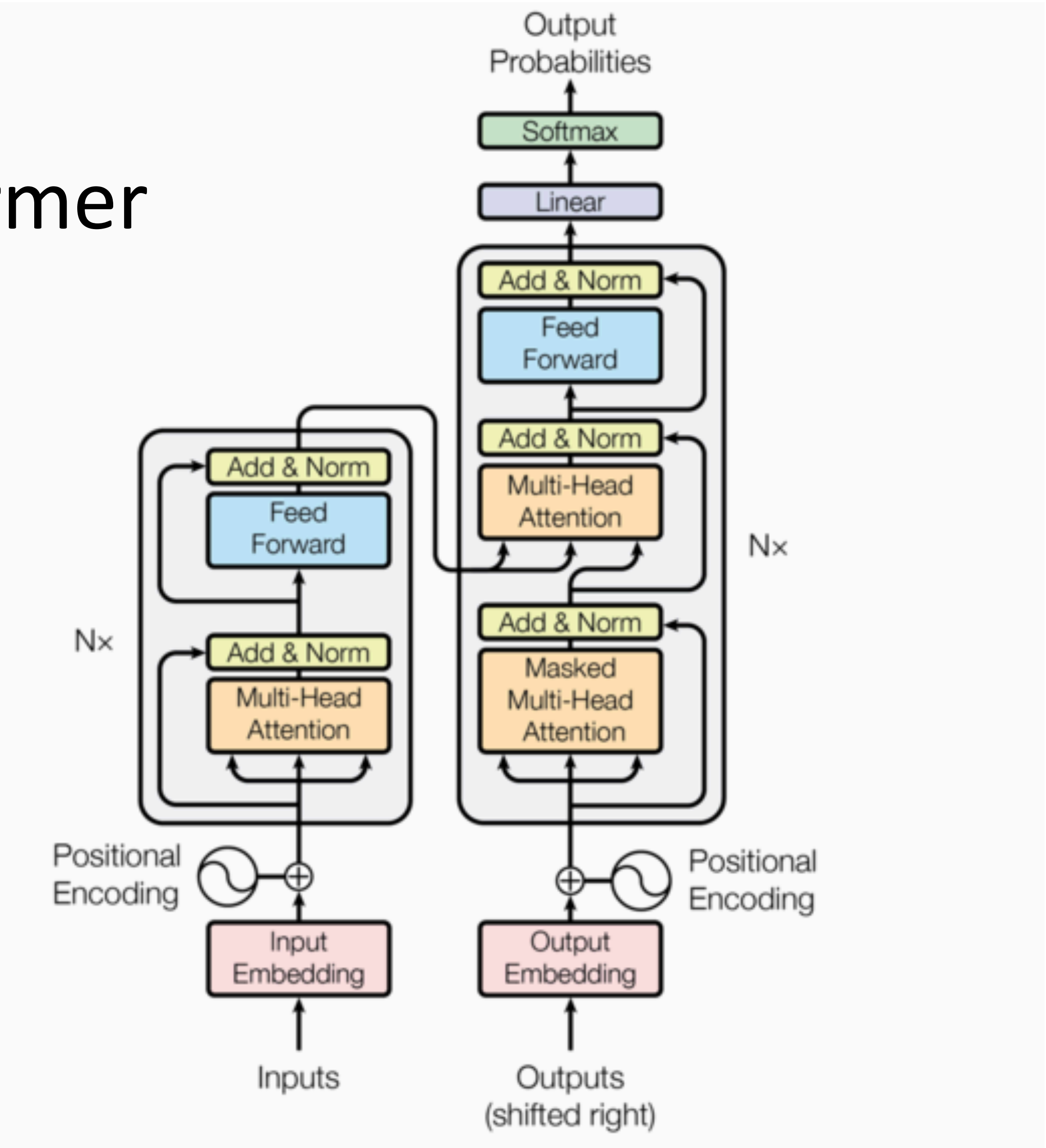
Recurrent NNs



Convolutional NNs



Transformer



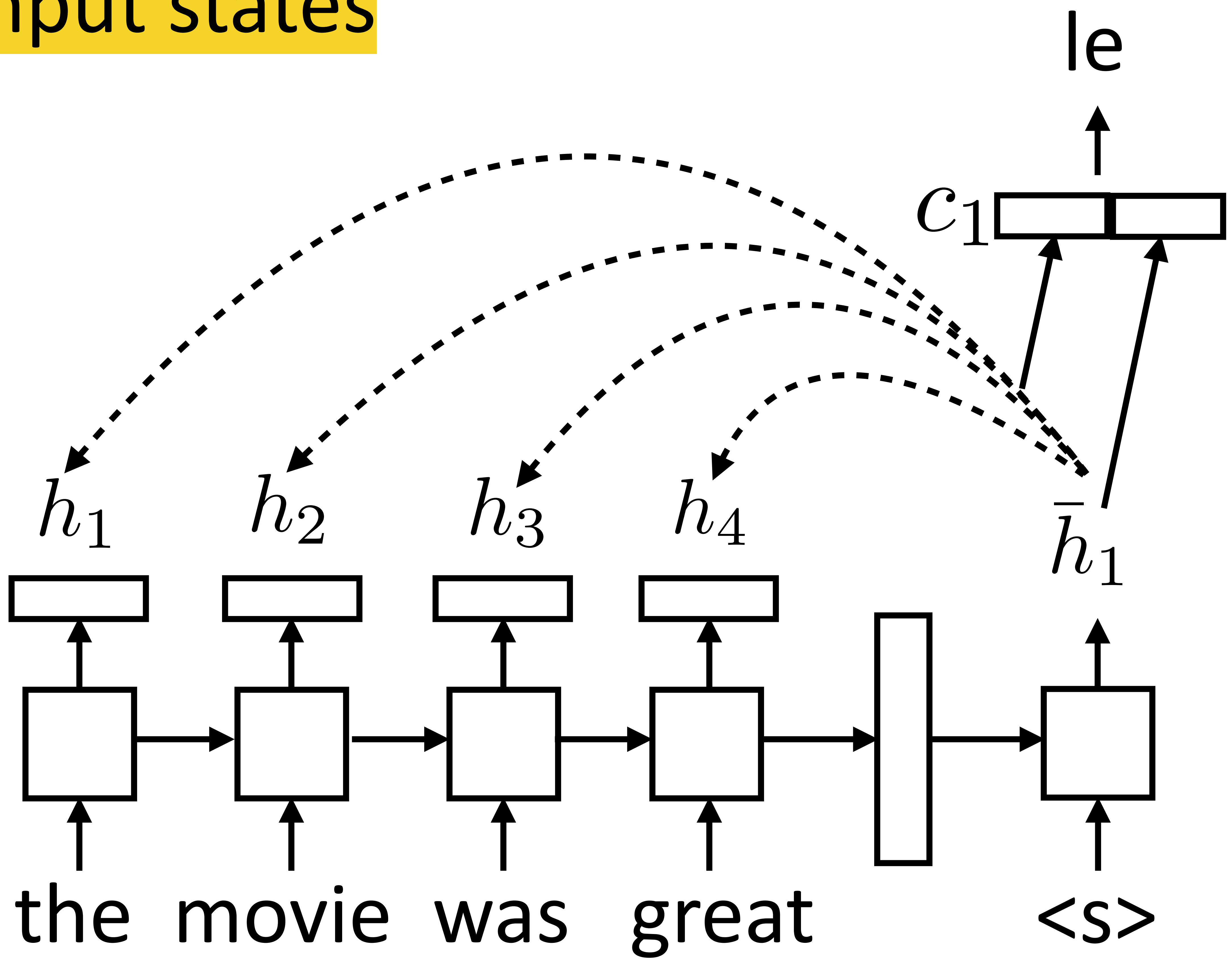
Always coupled with word embeddings…

Motivation for Transformers

- We want parallelization, but RNNs are inherently sequential.
- CNN gives us parallelization, but does not capture long range dependency.
- Despite its state representations and gating mechanisms, RNNs still need attention to deal with long-range dependencies
- If attention gives access to any state (and is required for high performance anyway), can we use attention instead of RNN?

Recap: Attention from Seq2Seq

- For each decoder state, compute weighted sum of input states

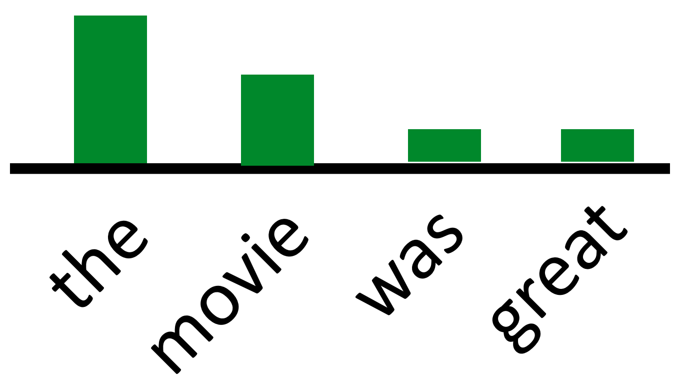


- No attn: $P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h}_i)$

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



$$e_{ij} = f(\bar{h}_i, h_j)$$

Motivation for Transformers

The ballerina is very excited that she will dance in the show.

- We would like to capture long range dependencies!

- CNN, RNN tends to be local

The ballerina is very excited that she will dance in the show.

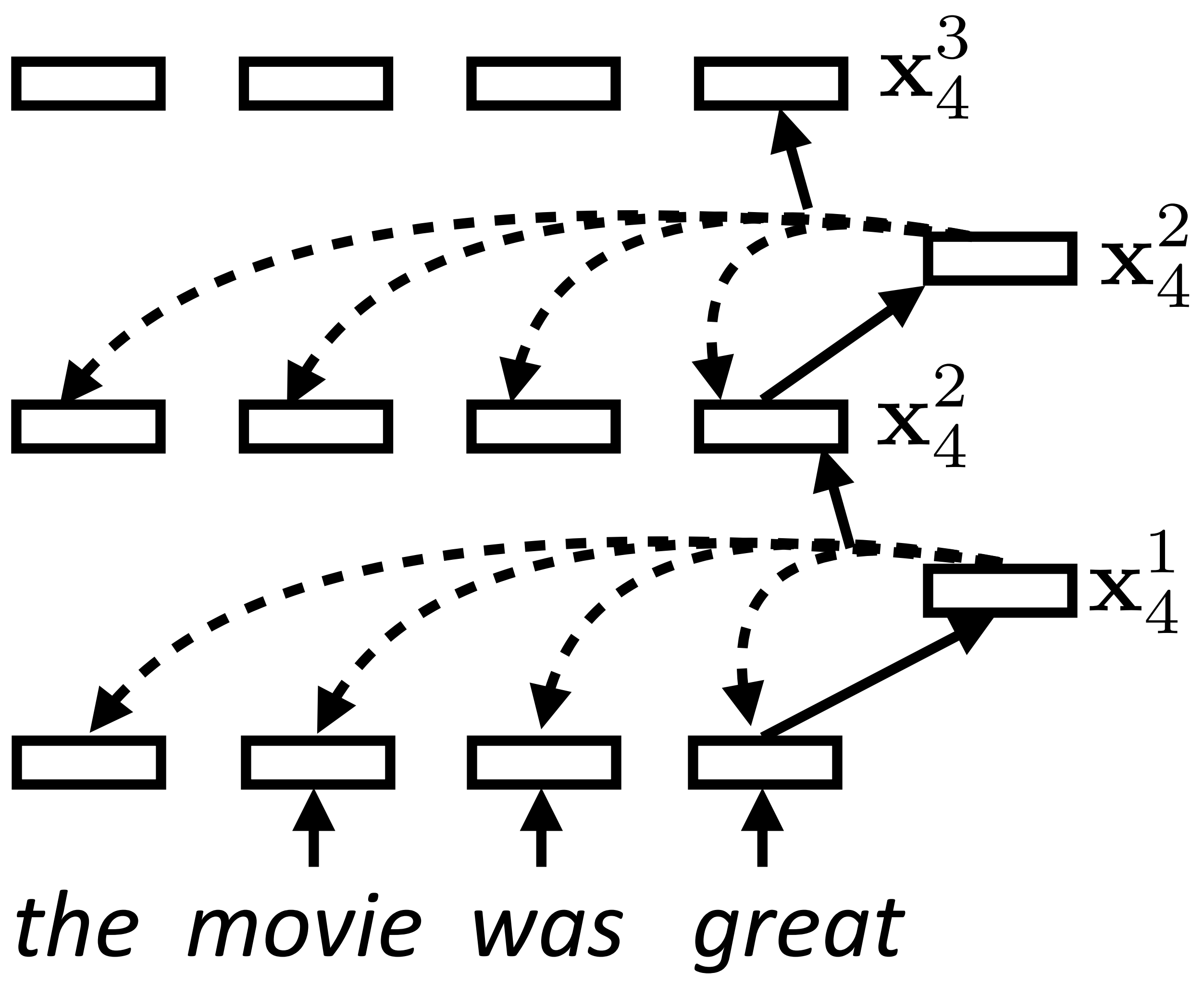
Motivation for Transformers

The ballerina is very excited that she will dance in the show.

- We would like to use:
  - Pronouns context should be the ancedecents (i.e., what they refer to)
  - Ambiguous words should consider local context
  - Word should look at its syntactic parents / children
- Goal: dynamically contextualize, passing information over long distances for each word

Solution: Self Attention

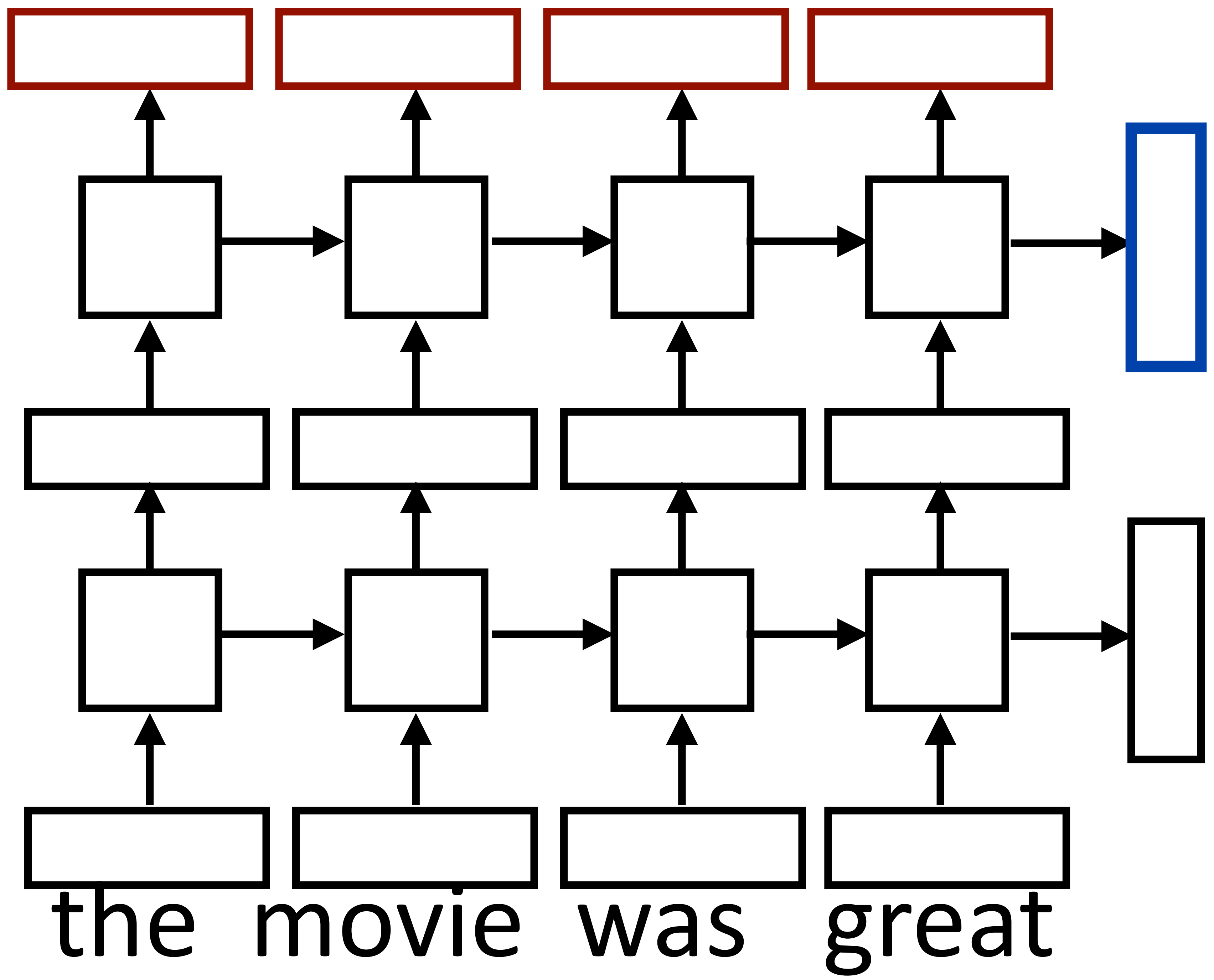
- Using attention for the ***encoder***.
- Each input token is a query to form attention over all tokens.
- Then, attention weights dynamically mix how much is taken from all tokens.

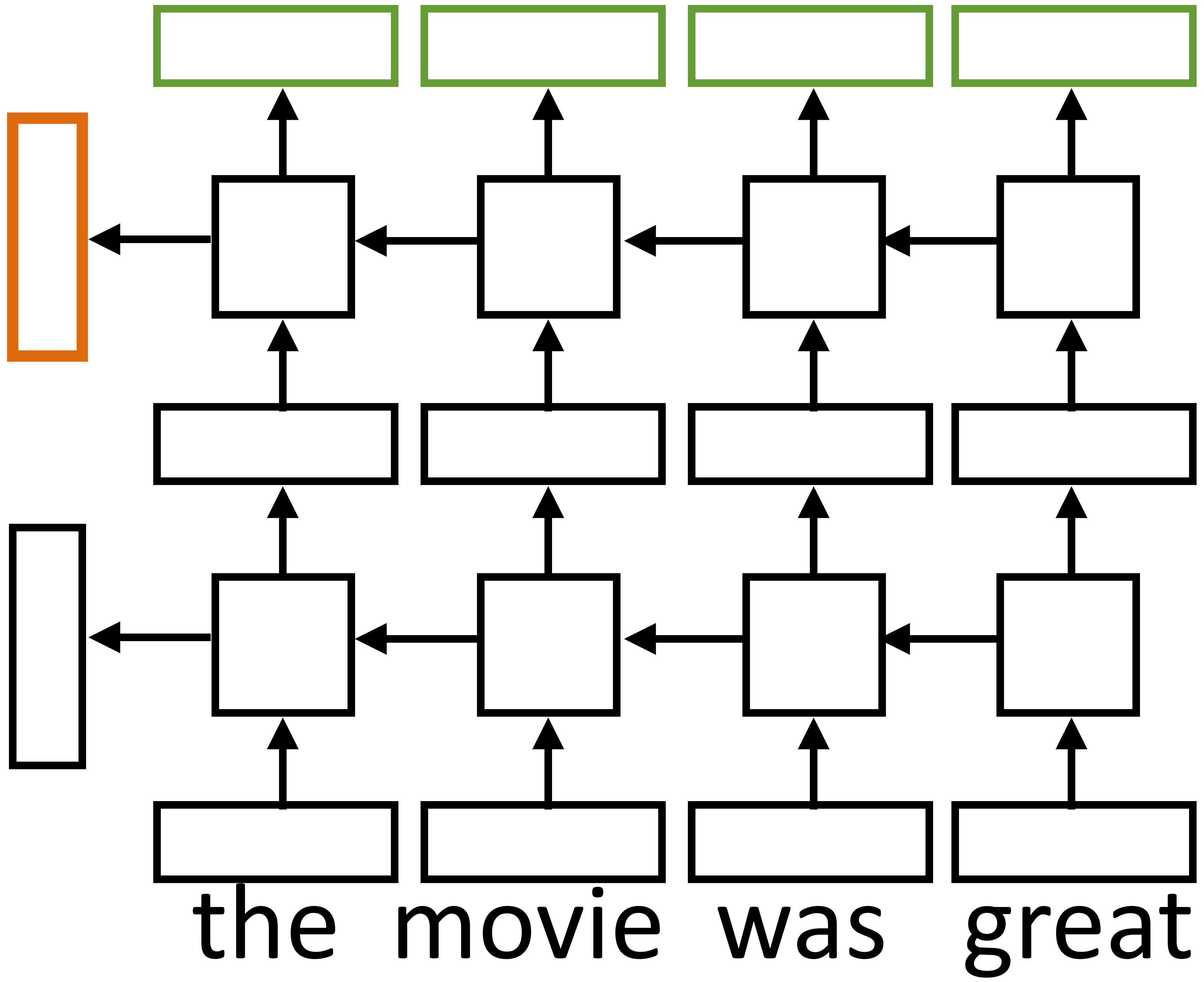


- Context-dependent representation of each token: a weighted sum of all tokens
- This will happen iteratively! Each step computing self-attention on the output of the previous level

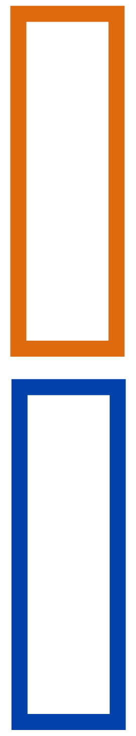
[Vaswani et al. 2017]

Recap: Multilayer Bidirectional RNN

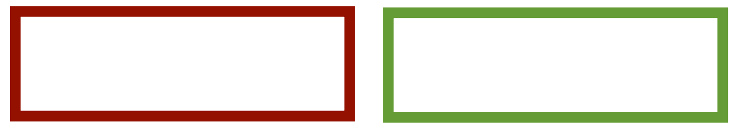




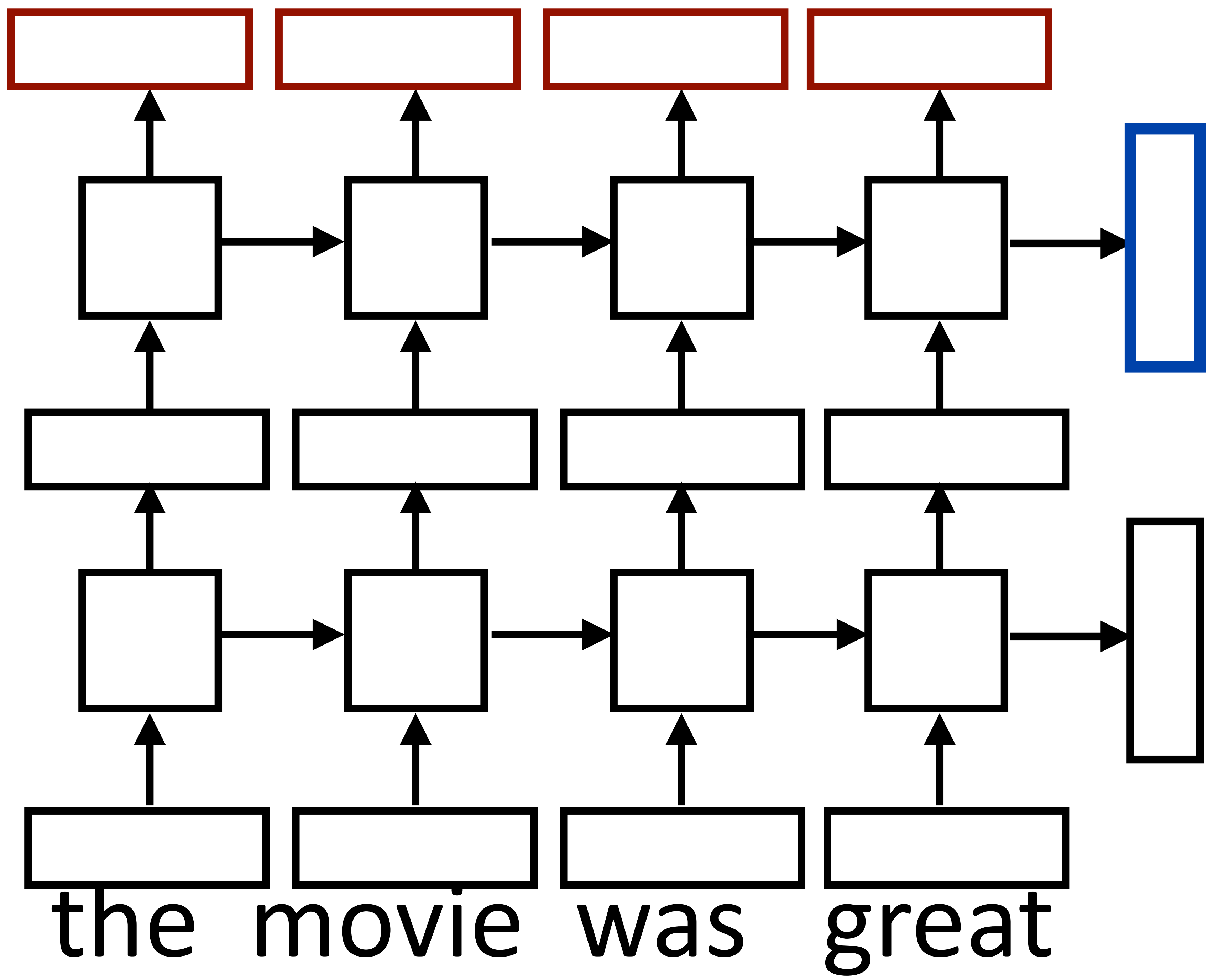
- Sentence classification based on concatenation of both final outputs

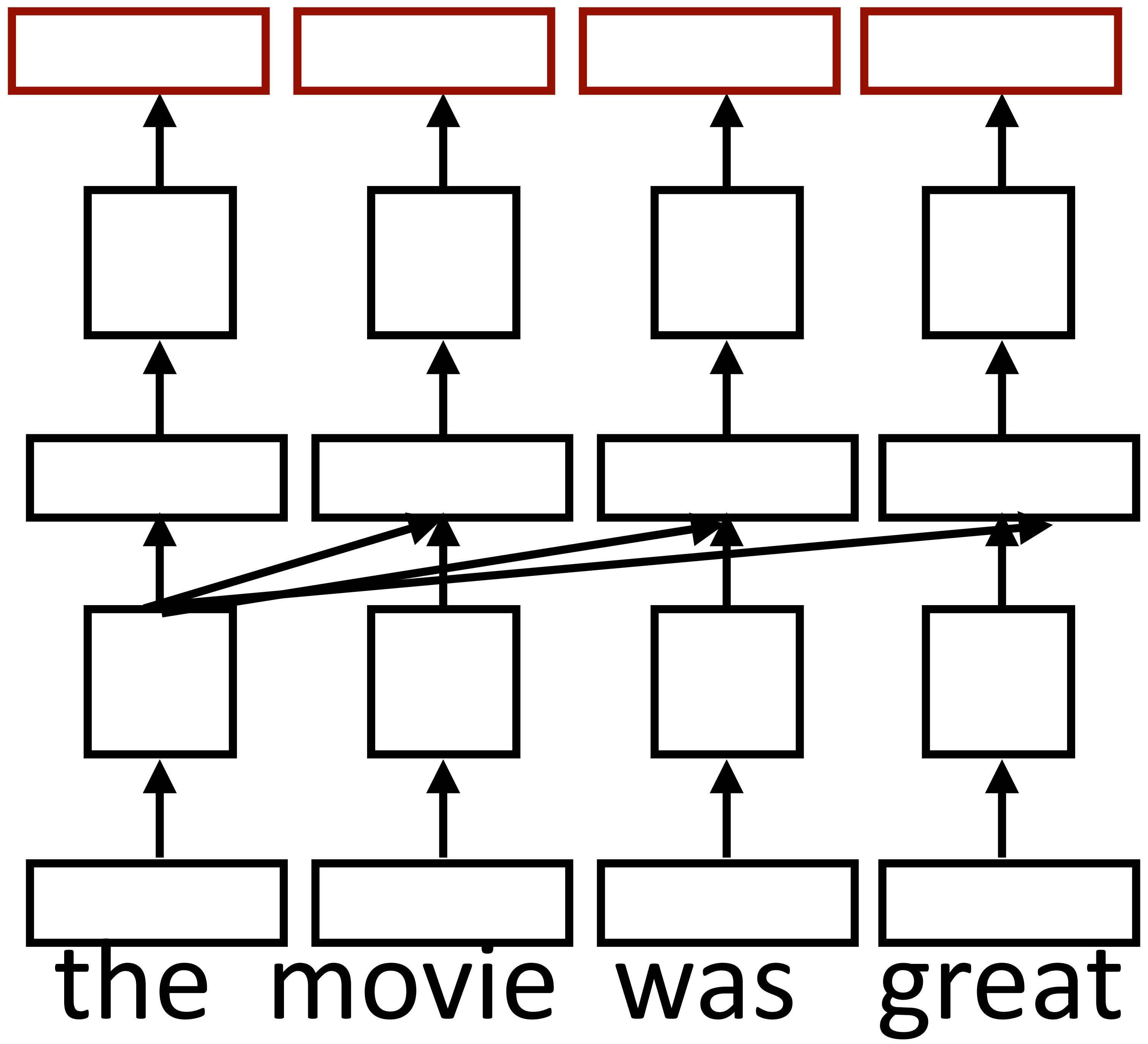


- Token classification based on concatenation of both directions' token representations



RNN vs. Transformers





Self Attention: Equation

$$k : \text{level number}$$

$$X : \text{input vectors}$$

$$X = \mathbf{x}_1, \ldots, \mathbf{x}_n$$

Input sequence of length n

$$\mathbf{x}_i^1 = \mathbf{x}_i$$

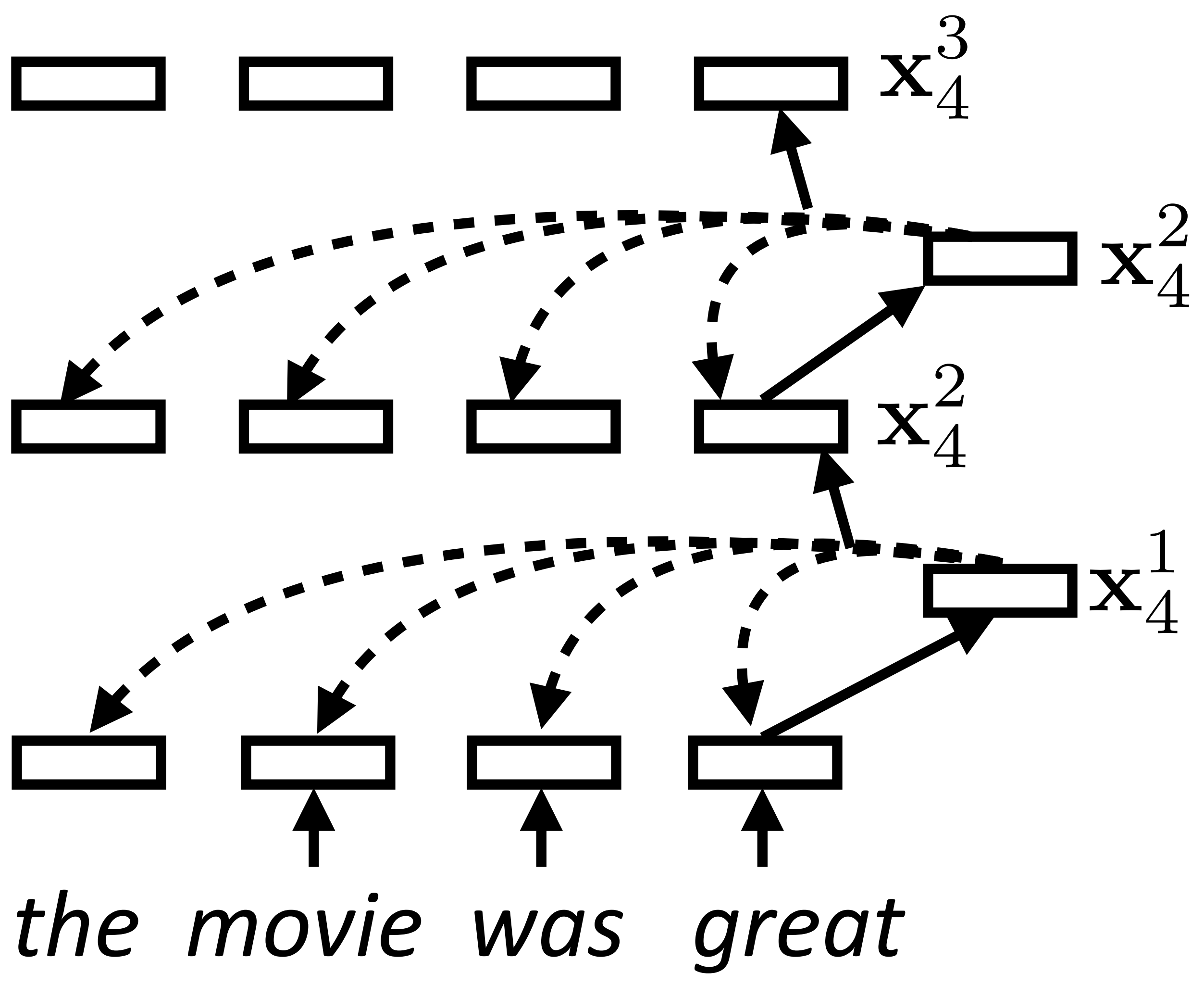
$$\bar{\alpha}_{i,j}^k = \mathbf{x}_i^{k-1} \cdot \mathbf{x}_j^{k-1}$$

Attention score for i-th input for j-th input

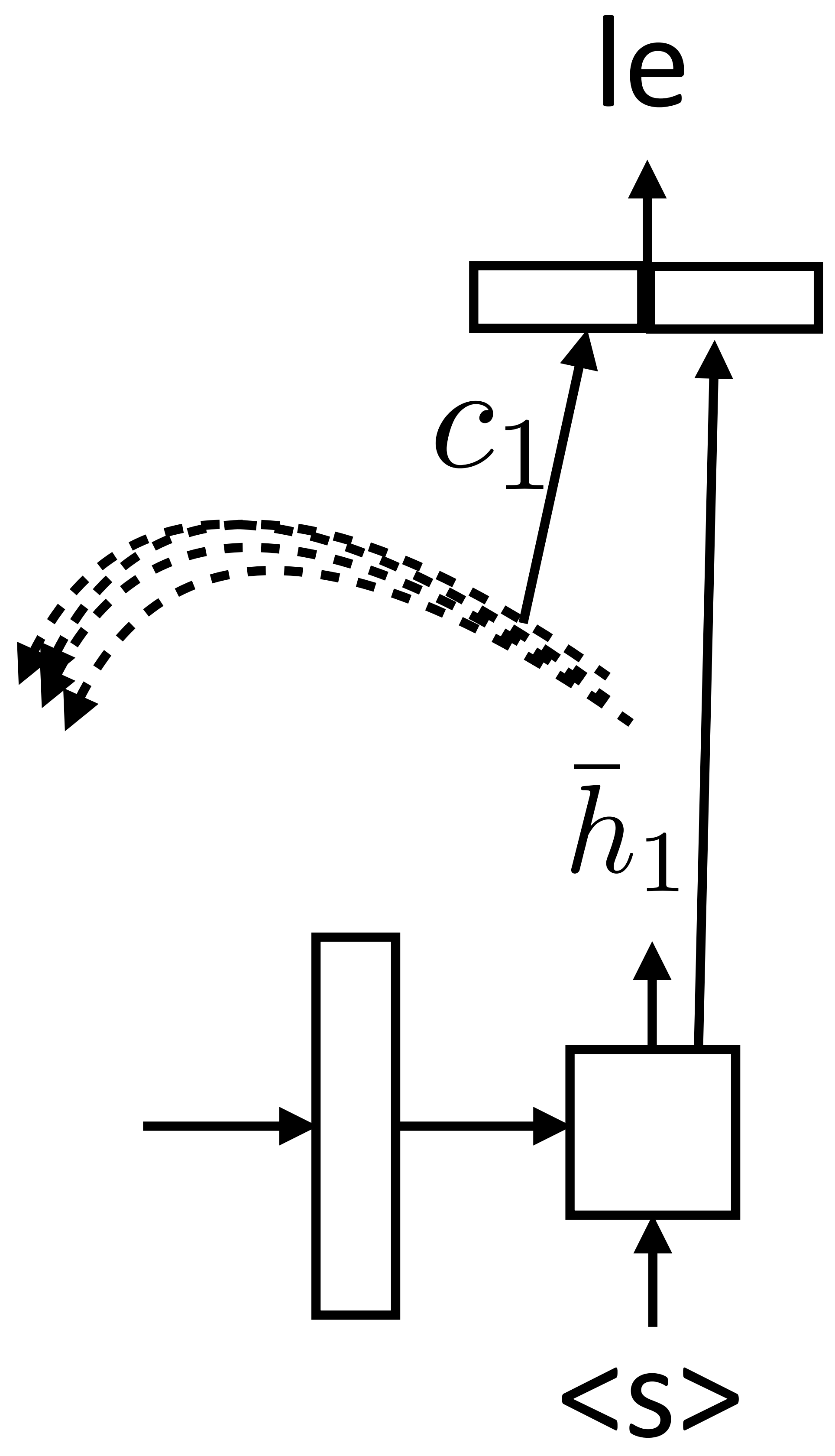
$$\alpha_{i,j}^k = \frac{\exp(\bar{\alpha}_{i,j}^k)}{\sum_j \exp(\bar{\alpha}_{i,j}^k)}$$

$$x_i^k = \sum_j \alpha_{i,j}^k x_j^{k-1}$$

Weighted sum of previous states (vector)



Recap: Attention Score Function



$h_i, h_j \in R^{d_k}$

$$e_{ij} = f(\bar{h}_i, h_j)$$

- Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

- Luong+ (2015): dot product

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

$$f(\hat{h}_i, h_j) = \frac{\bar{h}_i \cdot h_j}{\sqrt{d_k}}$$

- Luong+ (2015): bilinear

$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

Luong et al. (2015)

Multiple Attention Heads

- Why multiple heads? Softmax operations often end up peaky, making it hard to put weight on multiple items
- You can think of each head capturing different dependencies: one for finding subject, one for finding object, etc…

$$
\begin{aligned}
k &: \text{level number} \\
X &: \text{input vectors} \\
X &= \mathbf{x}_1, \ldots, \mathbf{x}_n \\
\mathbf{x}_i^1 &= \mathbf{x}_i \\
\bar{\alpha}_{i,j}^k &= \mathbf{x}_i^{k-1} \cdot \mathbf{x}_j^{k-1}
\end{aligned}
$$

$$\alpha_{i,j}^k = \frac{\exp(\bar{\alpha}_{i,j}^k)}{\sum_j \exp(\bar{\alpha}_{i,j}^k)}$$

$$x_i^k = \sum_j \alpha_{i,j}^k x_j^{k-1}$$

$$
\begin{aligned}
k &: \text{level number} \\
L &: \text{number of heads} \\
X &: \text{input vectors} \\
X &= \mathbf{x}_1, \ldots, \mathbf{x}_n \\
\mathbf{x}_i^1 &= \mathbf{x}_i \\
\bar{\alpha}_{i,j}^{k,l} &= \mathbf{x}_i^{k-1} \mathbf{W}^{k,l} \mathbf{x}_j^{k-1}
\end{aligned}
$$

$$\alpha_{i,j}^{k,l} = \frac{\exp(\bar{\alpha}_{i,j}^{k,l})}{\sum_j \exp(\bar{\alpha}_{i,j}^{k,l})}$$

$$x_i^{k,l} = \sum_j \alpha_{i,j}^{k,l} x_j^{k-1}$$

$$\mathbf{x}_i^k = \mathbf{V}^k [\mathbf{x}_i^{k,1}; \ldots; \mathbf{x}_i^{k,L}]$$

Multiple Attention Heads

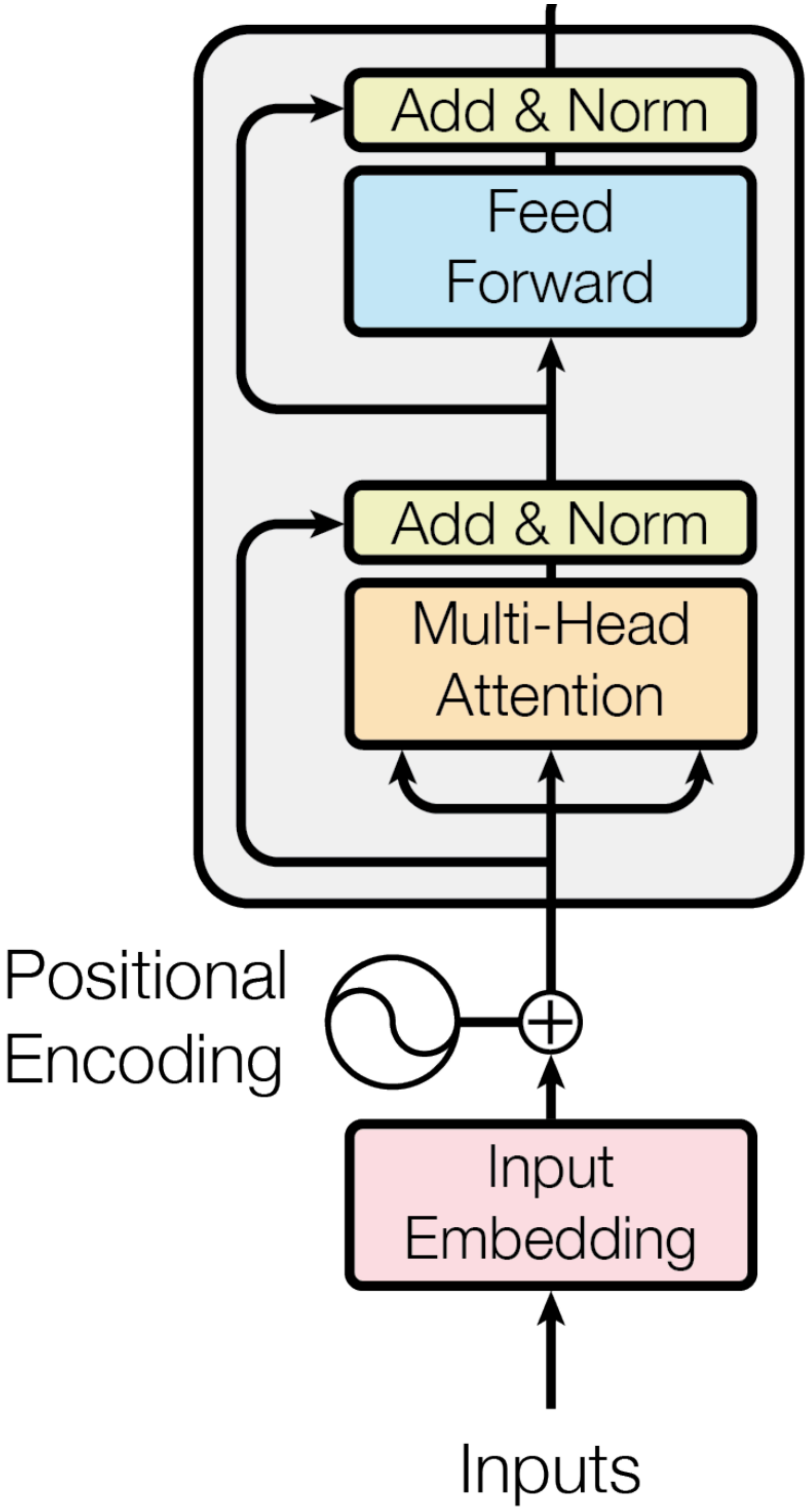
- Why
  mak
- You ... ne
  for f

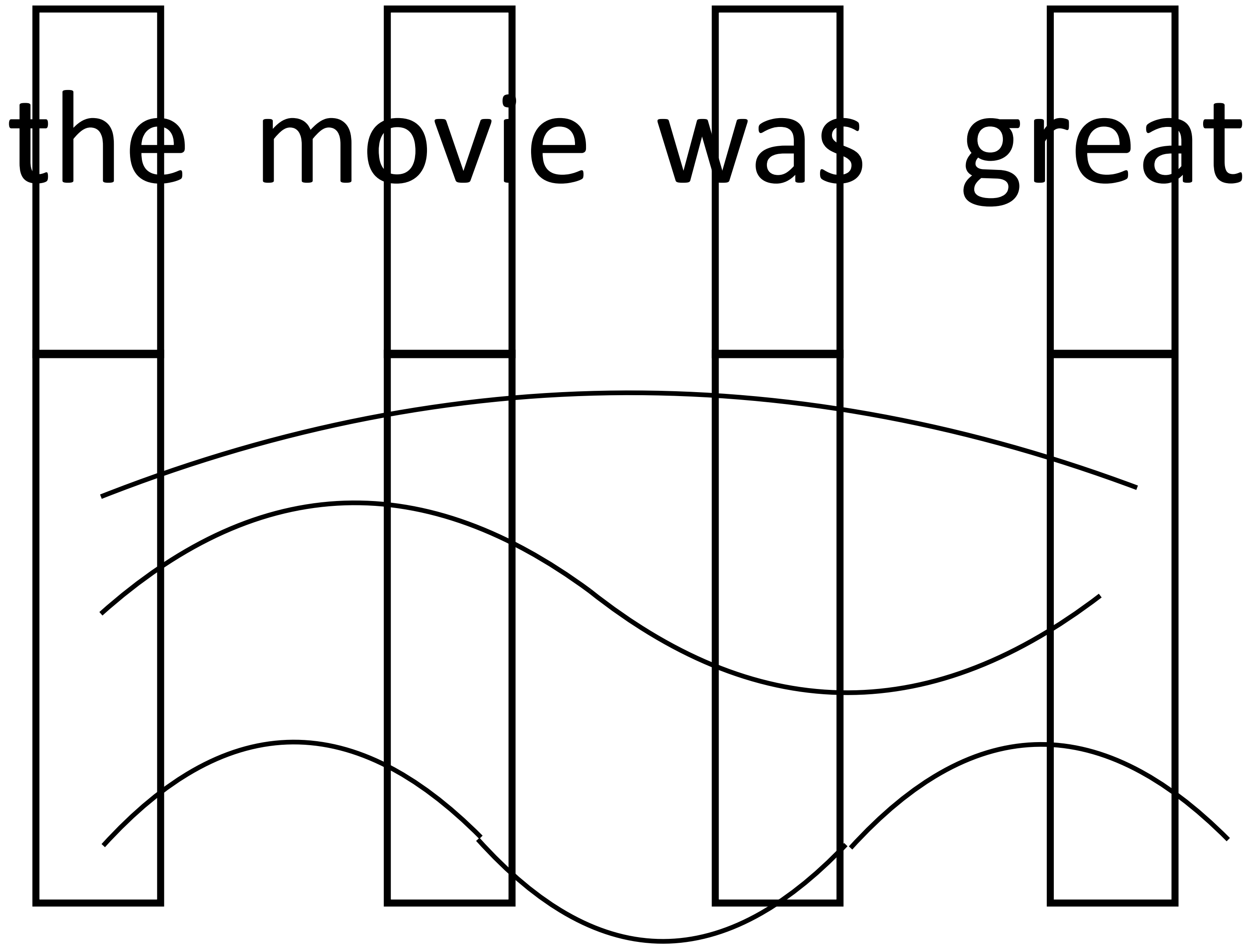
Would this algorithm know the position of the words in the input sequence?

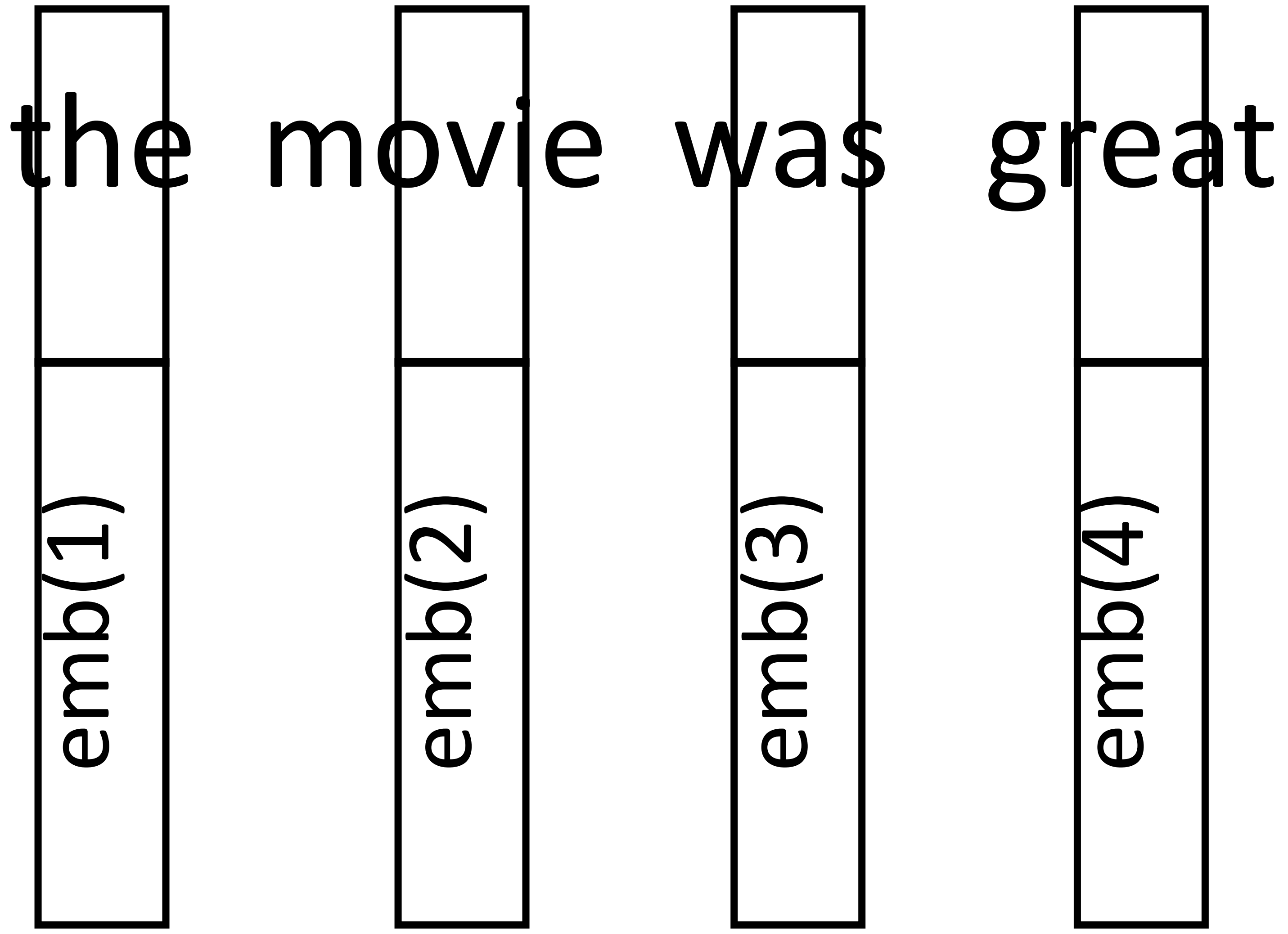
$$k : \text{level number}$$
$$X : \text{input vectors}$$
$$X = \mathbf{x}_1, \ldots, \mathbf{x}_n$$
$$\mathbf{x}_i^1 = \mathbf{x}_i$$
$$\bar{\alpha}_{i,j}^k = \mathbf{x}_i^{k-1} \cdot \mathbf{x}_j^{k-1}$$
$$\alpha_{i,j}^k = \frac{\exp(\bar{\alpha}_{i,j}^k)}{\sum_j \exp(\bar{\alpha}_{i,j}^k)}$$
$$x_i^k = \sum_j \alpha_{i,j}^k x_j^{k-1}$$

$$k : \text{level number}$$
$$L : \text{number of heads}$$
$$X : \text{input vectors}$$
$$X = \mathbf{x}_1, \ldots, \mathbf{x}_n$$
$$\mathbf{x}_i^1 = \mathbf{x}_i$$
$$\bar{\alpha}_{i,j}^{k,l} = \mathbf{x}_i^{k-1} \mathbf{W}^{k,l} \mathbf{x}_j^{k-1}$$
$$\alpha_{i,j}^{k,l} = \frac{\exp(\bar{\alpha}_{i,j}^{k,l})}{\sum_j \exp(\bar{\alpha}_{i,j}^{k,l})}$$
$$x_i^{k,l} = \sum_j \alpha_{i,j}^{k,l} x_j^{k-1}$$
$$\mathbf{x}_i^k = \mathbf{V}^k [\mathbf{x}_i^{k,1}; \ldots; \mathbf{x}_i^{k,L}]$$

Positional Embeddings



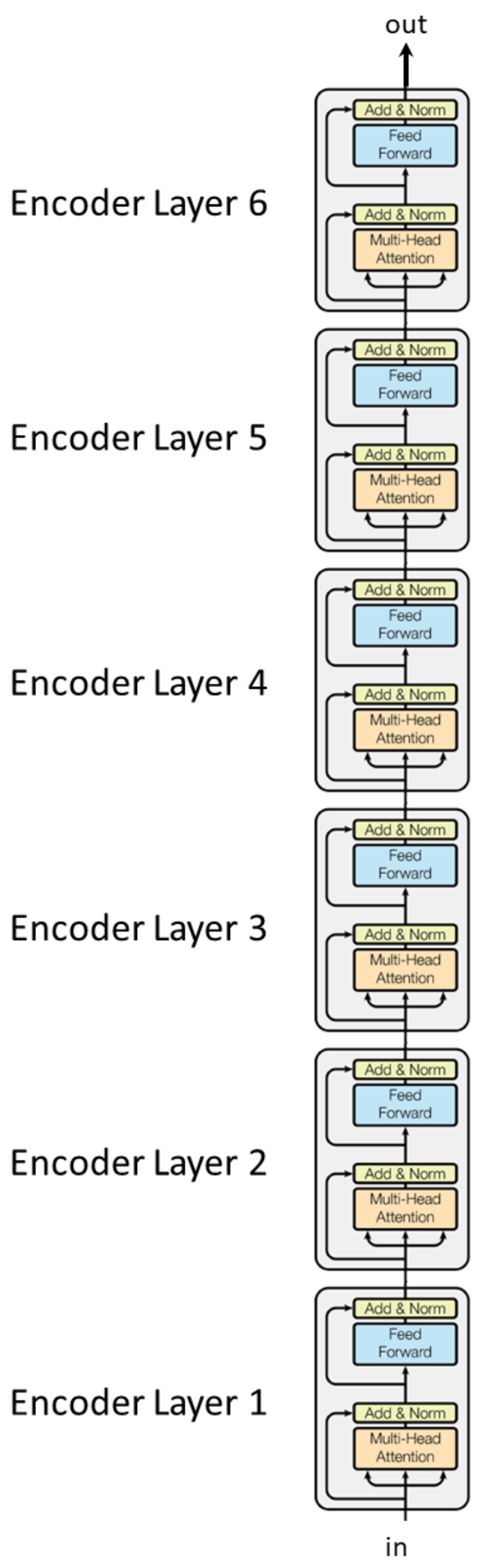


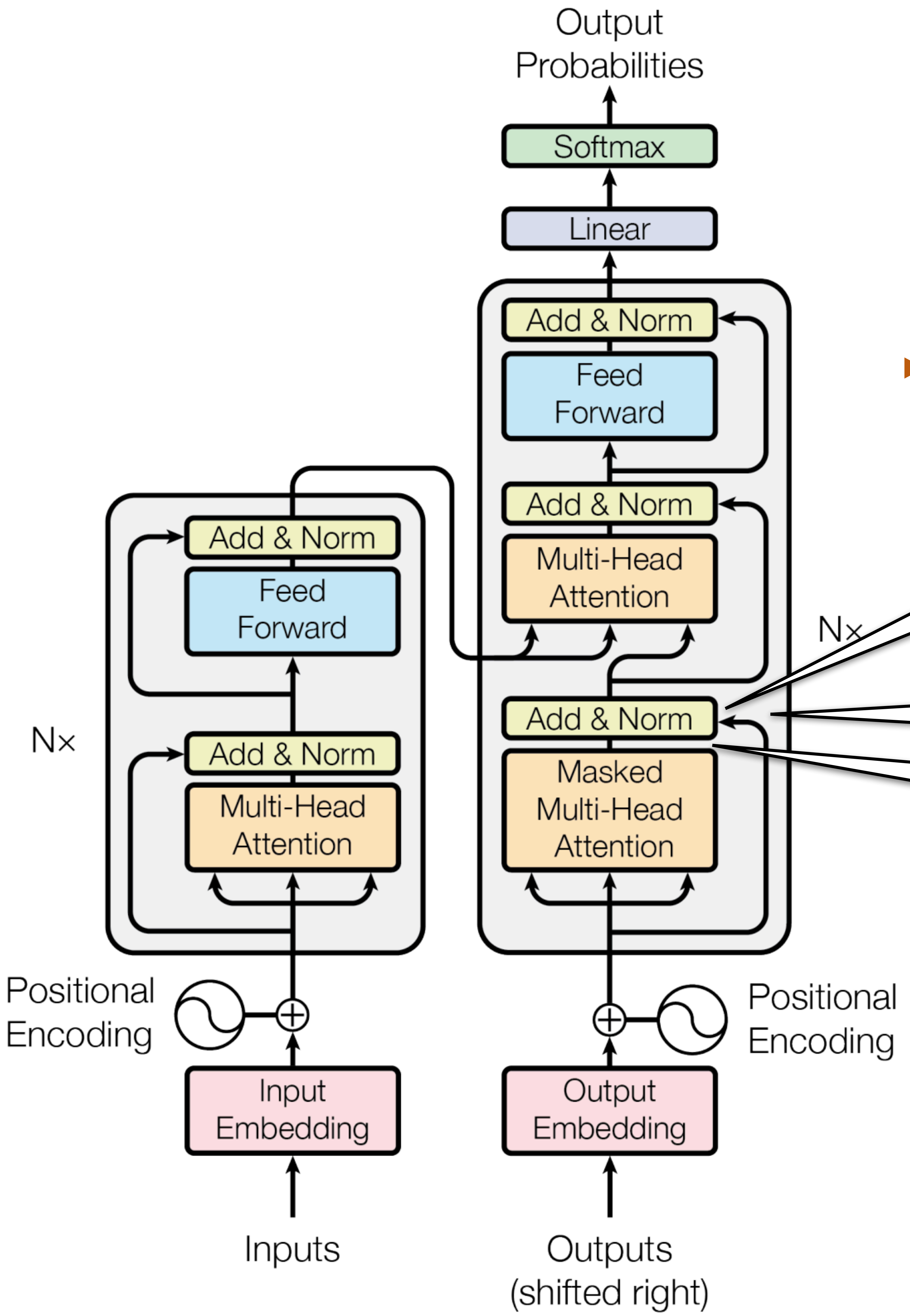


- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

Vaswani et al. (2017)

Transformers





- Encoder and decoder are both transformers
- Details, Details, Details!

ADD: Highway connection

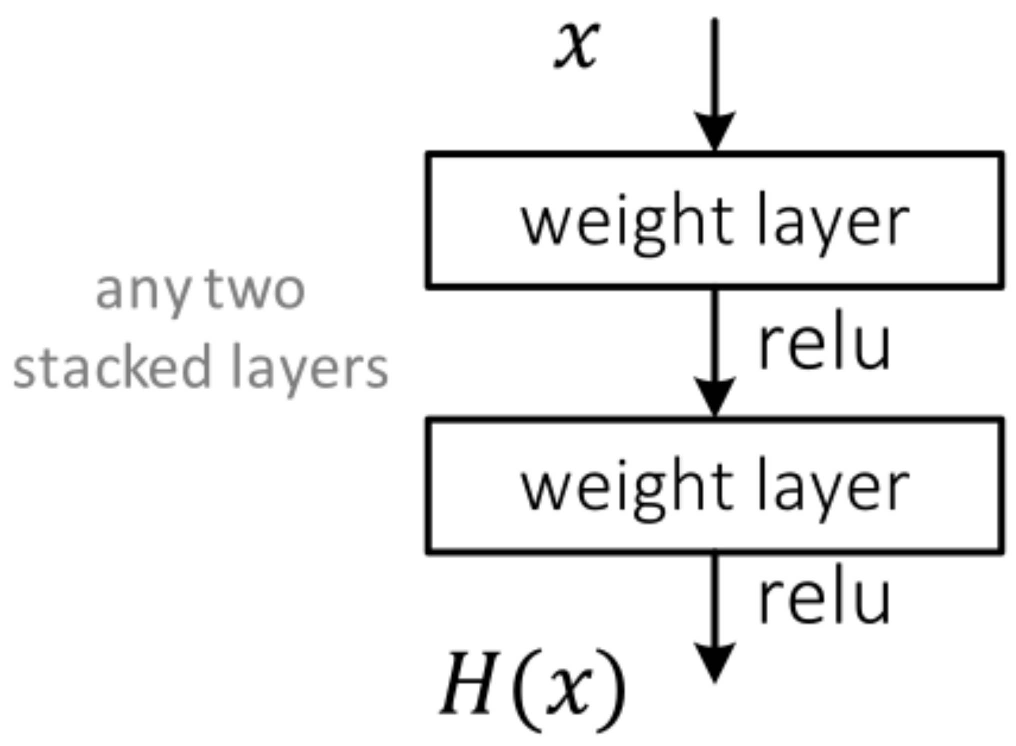
Norm: Layer normalization

Regularization: Dropout

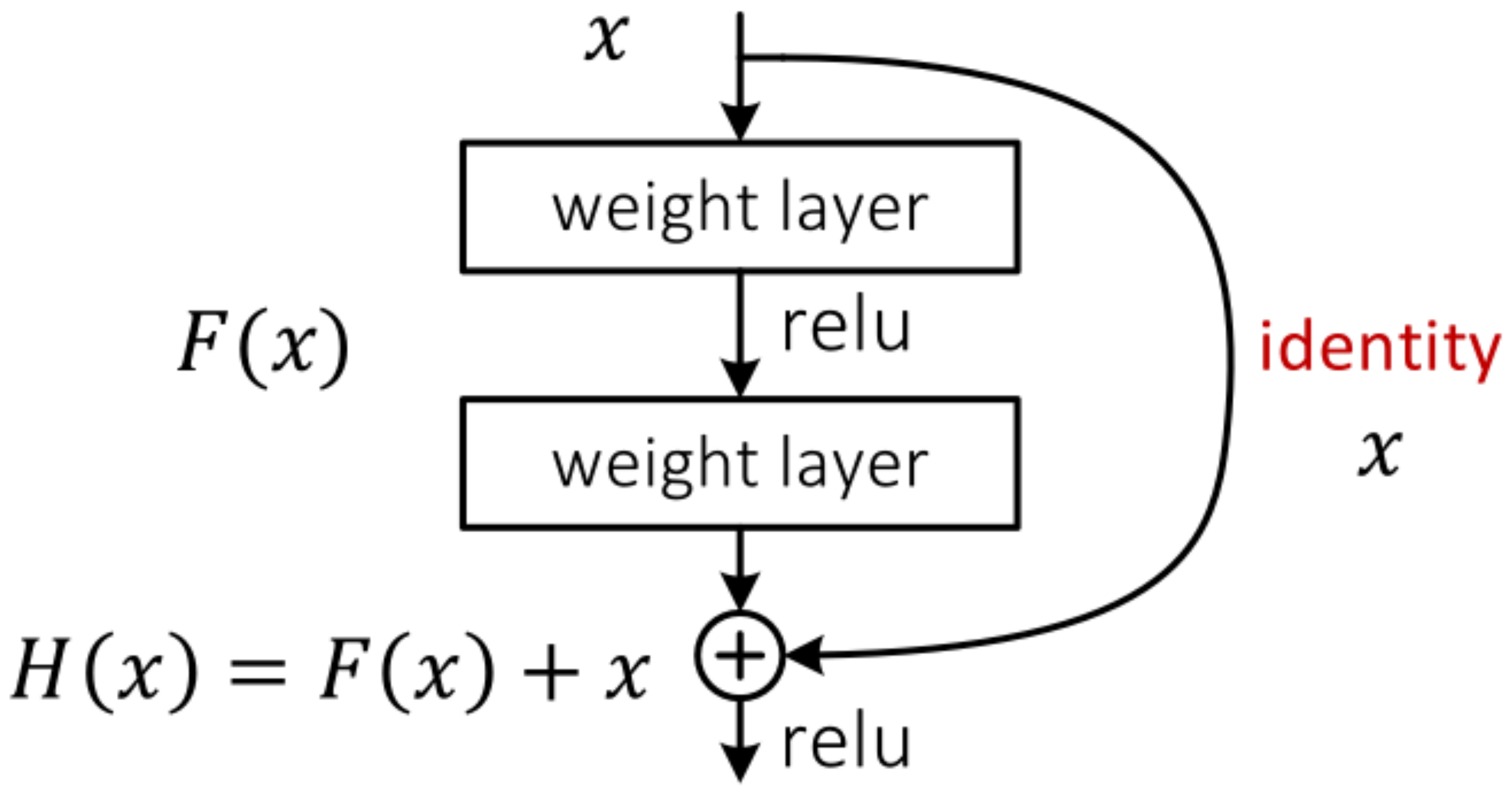
Vaswani et al. (2017)

Residual Network

- Plaint net

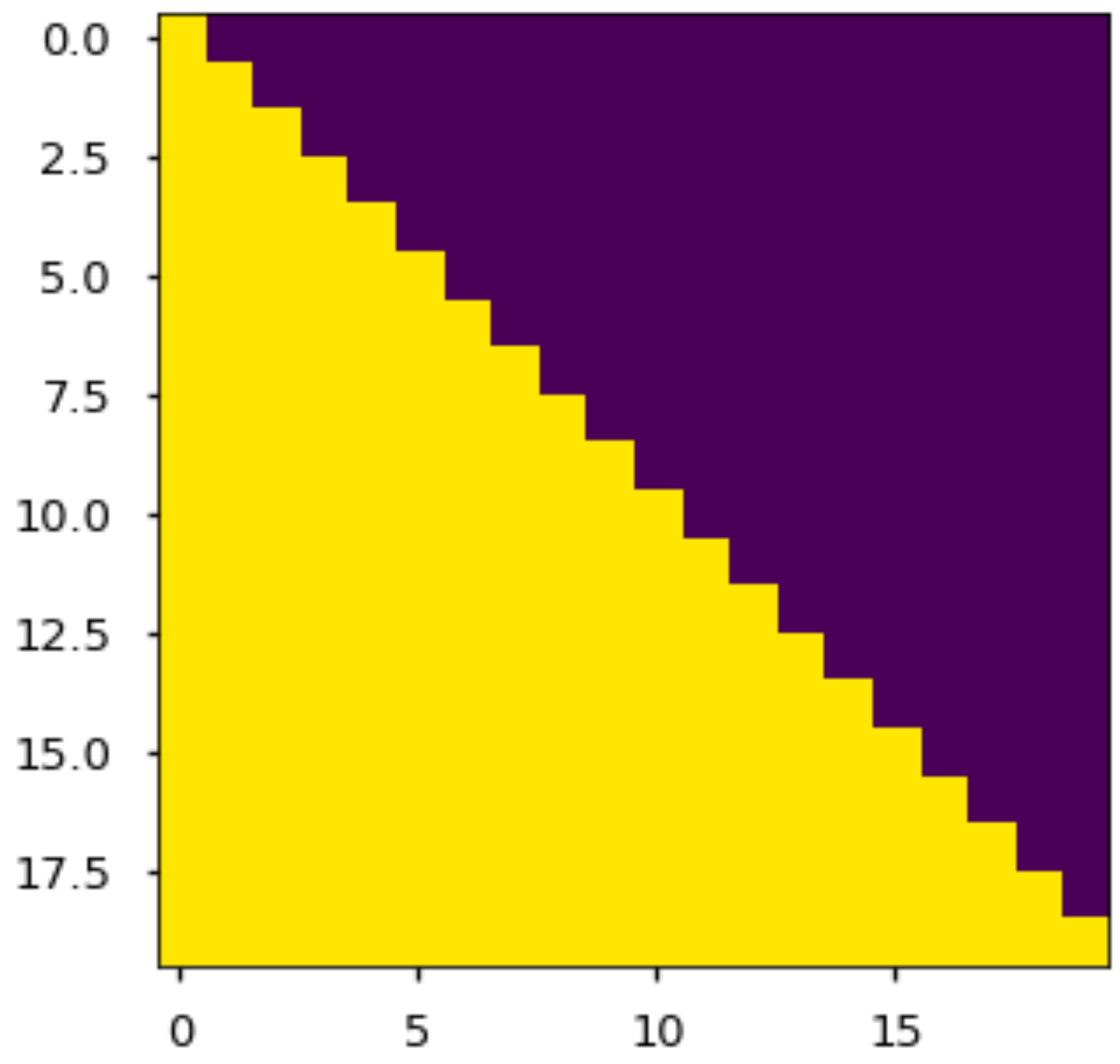


- Residual net



- ResNet (He et al. 2015): first very deep (152 layers) network successfully trained for object recognition

Decoding with Transformers



- Decoder consumes the previous generated token (and attends to input), without *recurrent state*
- Words are blocked for attending to future words

Performances of Transformers: MT

Model	BLEU	
	EN-DE	EN-FR
ByteNet [18]	23.75	
Deep-Att + PosUnk [39]		39.2
GNMT + RL [38]	24.6	39.92
ConvS2S [9]	25.16	40.46
MoE [32]	26.03	40.56
Deep-Att + PosUnk Ensemble [39]		40.4
GNMT + RL Ensemble [38]	26.30	41.16
ConvS2S Ensemble [9]	26.36	**41.29**
Transformer (base model)	27.3	38.1
Transformer (big)	**28.4**	**41.8**

- big = 6 layers, 1000 dim for each token, 16 heads, base = 6 layers + other params halved

Vaswani et al. (2017)

Visualization

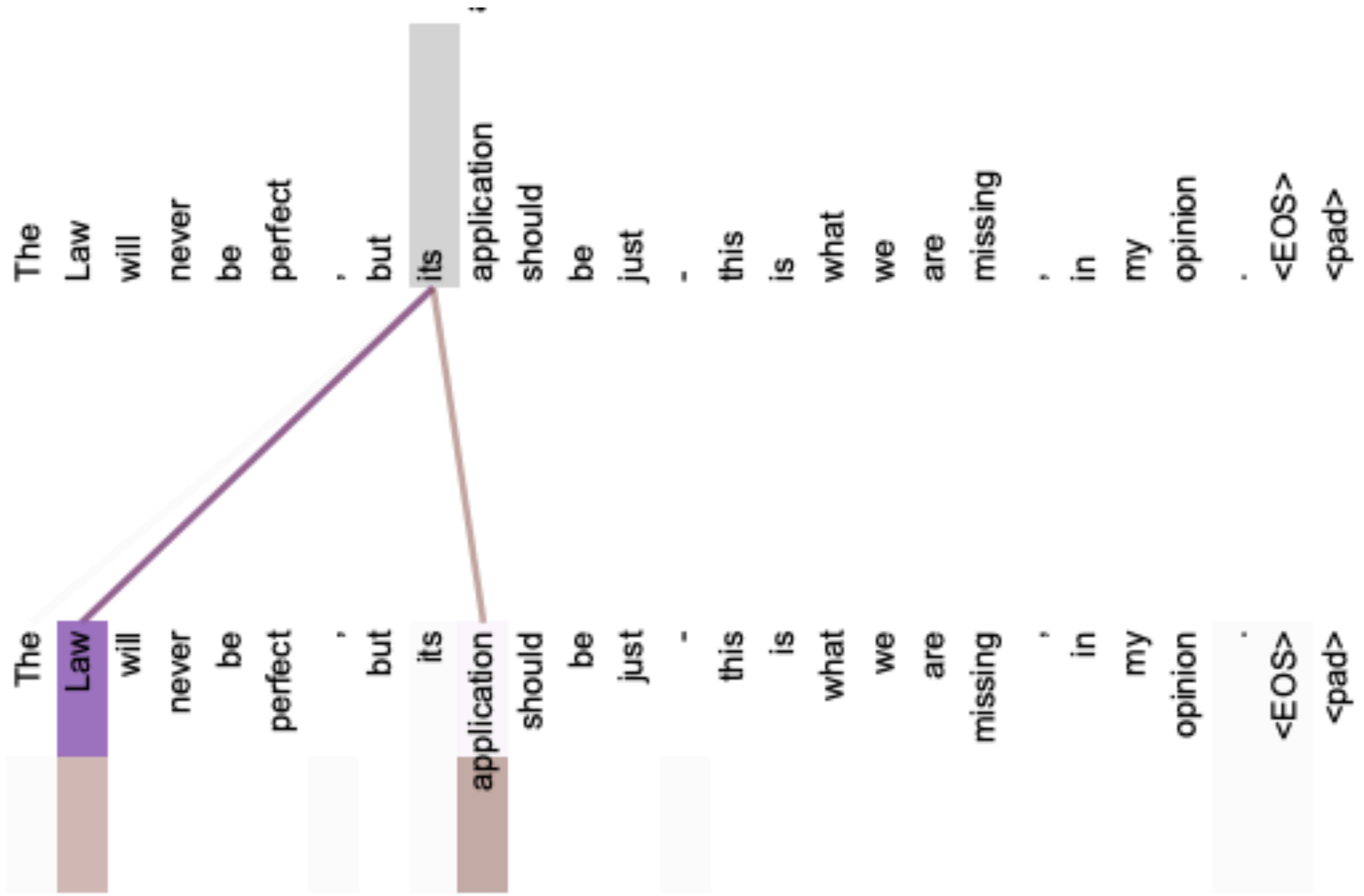
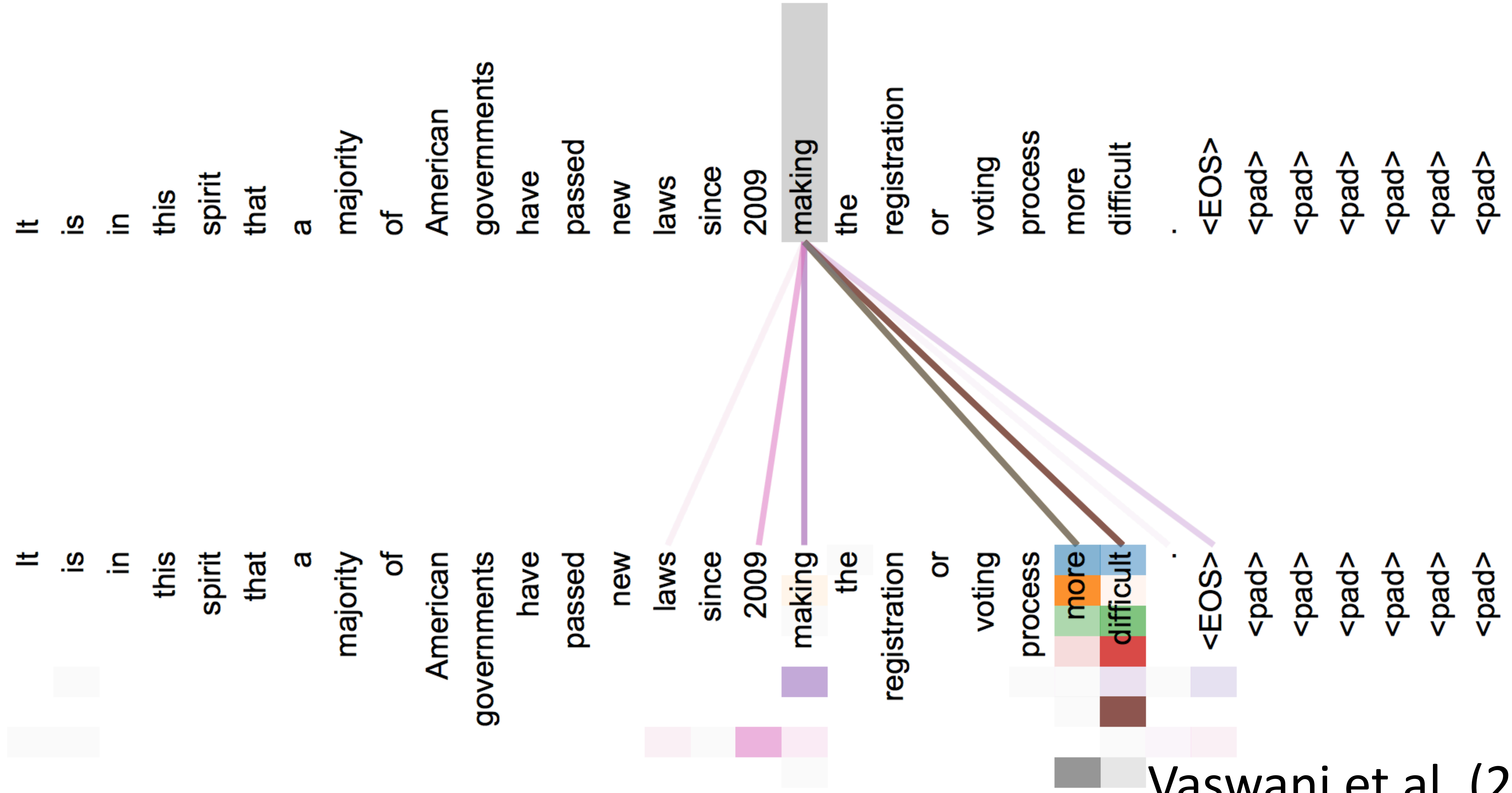


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

et al. (2017)

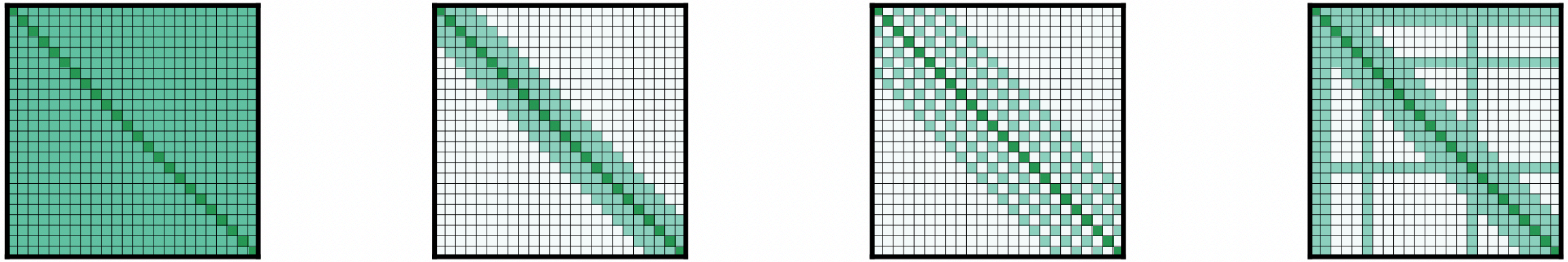
Visualization



Vaswani et al. (2017)

Limitations?

- n^2 attention computation can get expensive when the sequence is long



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

[Beltagy et al, 2020, Zaheer et al, 2020, many others..]

Takeaways

- RNN requires sequential processing, CNN enables parallel processing
- Both CNN and RNN assumes locality
- Transformers are strong, general models we'll see frequently
- Next week: Contextualized word embedding
  - How language models, RNN, Transformers are used for many downstream tasks
- Next Next week: Tree structure
  - Dependency parse
  - Constituency parse