TTIC 31190: Natural Language Processing

Lecture 10: Neural Language Modeling & Sequence-to-Sequence Modeling

Fall 2023

Announcements

- Assignment 2 due on Nov 2, 11:59 pm

- Literature review project midpoint check due on Nov 9, 11:59 pm

- Final exam schedule: Tuesday December 5, 3-5pm
  - Pass/fail option available for this course

Language Models

- Language Model: a probability distribution over strings in a language.

$$P(\boldsymbol{x}) \quad \boldsymbol{x} = x_1, x_2, \ldots, x_n$$

$$P(\text{I'm not a cat}) = 0.0000004$$

$$P(\text{He is hungry}) = 0.000025$$

$$P(\text{Dog the asd@sdf 1124 !?}) \approx 0$$

Language Modeling

- Goal: compute the probability of a sequence of words:

$$P(\boldsymbol{x}_{1:n}) = P(x_1, x_2, \ldots, x_n)$$

- Related task: probability of next word:

$$P(x_4 \mid x_1, x_2, x_3)$$

- A model that computes either of these:

$$P(\boldsymbol{x}_{1:n}) \quad \text{or} \quad P(x_k \mid x_1, x_2, \ldots, x_{k-1})$$

is called a **language model (LM)**

[SLP3: Chapter 3]

Language Modeling

- Building language models
- Generating from a language model
- Evaluating a language model

- Count-based language models
  - MLE estimation
  - Smoothing

- Neural language models
  - Feed-forward models
  - RNN models
  - Attention models

Overview

- Neural language models
  - Feed-forward models
  - RNN models
  - Attention models

- Machine Translation & Sequence-to-sequence models
  - Machine translation
  - Encoder decoder structures
  - + Attention & applications

Language Modeling

$$P(\boldsymbol{x}_{1:n}) = P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \boxed{P(x_i \mid x_1, x_2, \ldots, x_{i-1})}$$

- This is just a probabilistic classification problem!
- We can use any tools from the previous lectures: linear model with features, neural networks, etc.

Count-based Language Models

$$P(\boldsymbol{x}_{1:n}) = P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i \mid x_1, x_2, ..., x_{i-1})$$

- Idea 1: make an k-th order Markov assumption

$$P(x_i \mid \texttt{<s>}, x_1, \ldots, x_{i-2}, x_{i-1}) \approx P(x_i \mid x_{i-k}, \ldots, x_{i-2}, x_{i-1})$$

- E.g. Trigram LM (k=2)

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$$

Count-based Language Models

$$P(\boldsymbol{x}_{1:n}) = P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i \mid x_1, x_2, ..., x_{i-1})$$

- Idea 1: make an k-th order Markov assumption

$$P(x_i \mid \texttt{<s>}, x_1, \ldots, x_{i-2}, x_{i-1}) \approx P(x_i \mid x_{i-k}, \ldots, x_{i-2}, x_{i-1})$$

- Maximum likelihood (e.g. k=2)

$$P(x_i | x_{i-2}, x_{i-1}) = \frac{\#(x_{i-2}, x_{i-1}, x_i)}{\#(x_{i-2}, x_{i-1})}$$

Count-based Language Models

- Equivalent to MLE solution with a linear model with feature vector given by n-grams.

Number of dimensions $= |\mathcal{V}|^2$

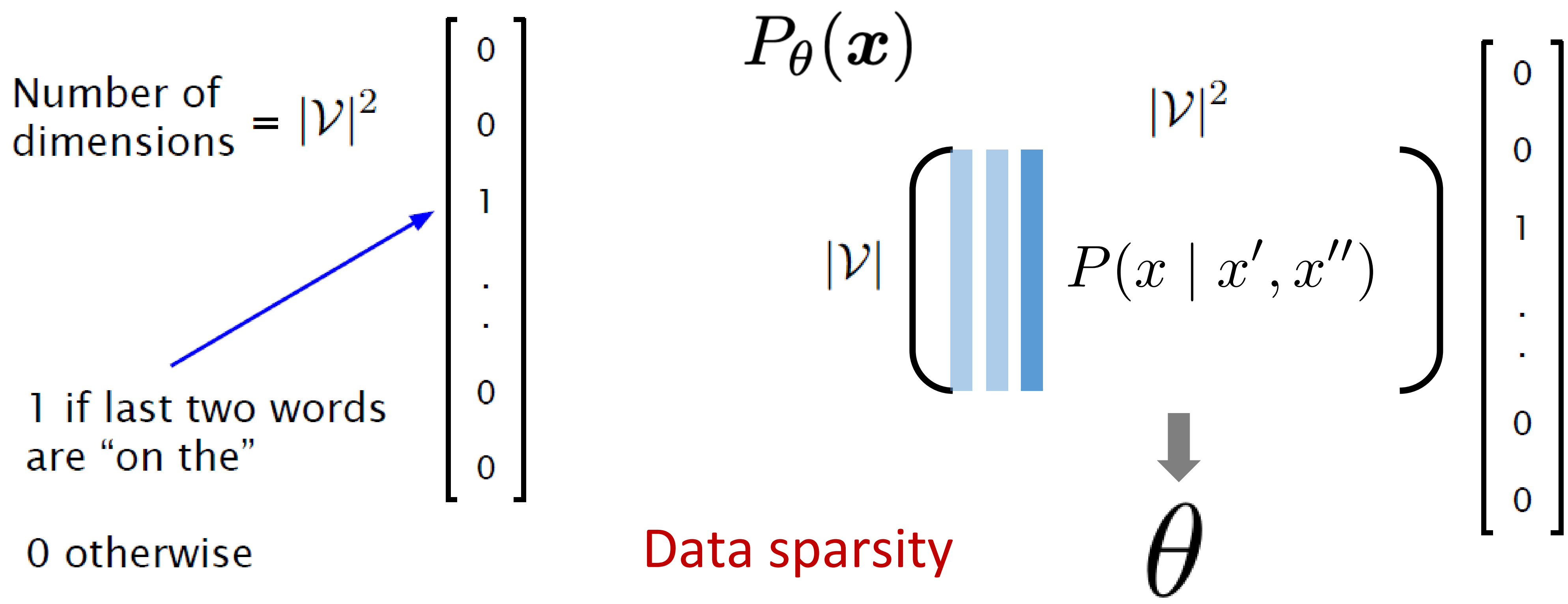
$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

1 if last two words are “on the”

0 otherwise

Count-based Language Models

- Equivalent to MLE solution with a linear model with feature vector given by n-grams.



Neural Language Model

- Idea 2: Use a neural network over of word embeddings

Number of dimensions = $|\mathcal{V}|$

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \cdot \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdot \\ 1 \end{bmatrix}$$

1 if x_{t-1} = “the”

1 if x_{t-2} = “on”

Neural Language Model

- Idea 2: Use a neural network over of word embeddings

$$W \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \text{word embedding of “the”}$$

$d \times |\mathcal{V}|$ *input embedding* matrix

Neural Language Model

Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

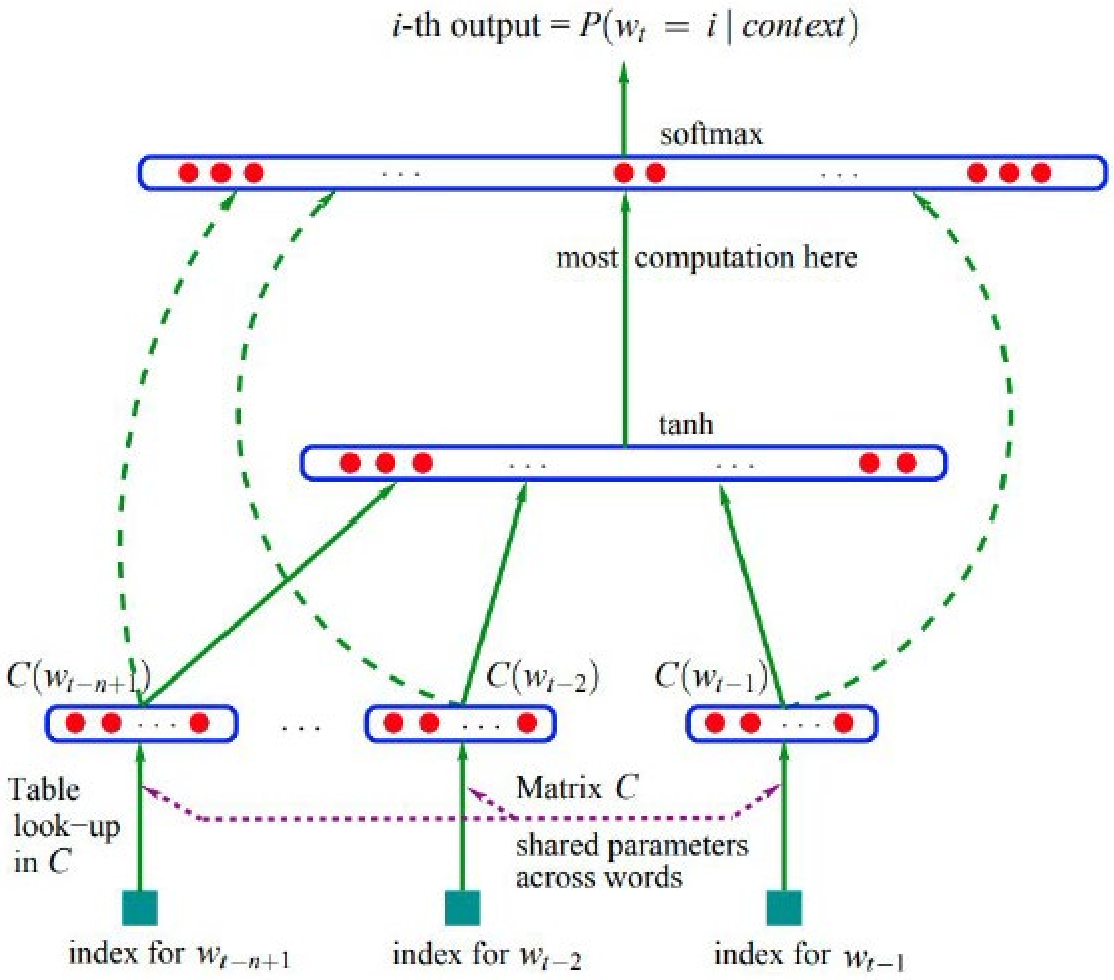
A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

- Idea: use a neural network for *n*-gram language modeling

$$P(x \mid x', x'')$$

Neural Language Model



Recap: Neural Networks

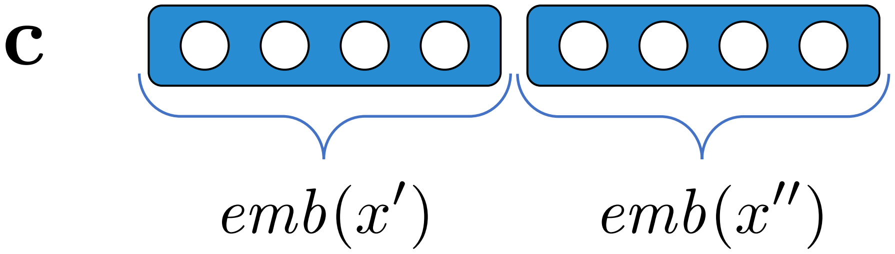
- We can think of a neural network as a continuous function with some learnable parameters
  - it has inputs and outputs, which are usually vectors
  - it’s typically a nonlinear function
- Neural networks / deep learning is best thought of as a modeling strategy that combines:
  - distributed representations (e.g., word embeddings)
  - representation learning
  - nonlinear functions

A Simple Neural Trigram Language Model

- given two previous words, compute probability distribution over possible next words

$$P(x \mid x', x'')$$

- input is concatenation of vectors (embeddings) of previous two words:

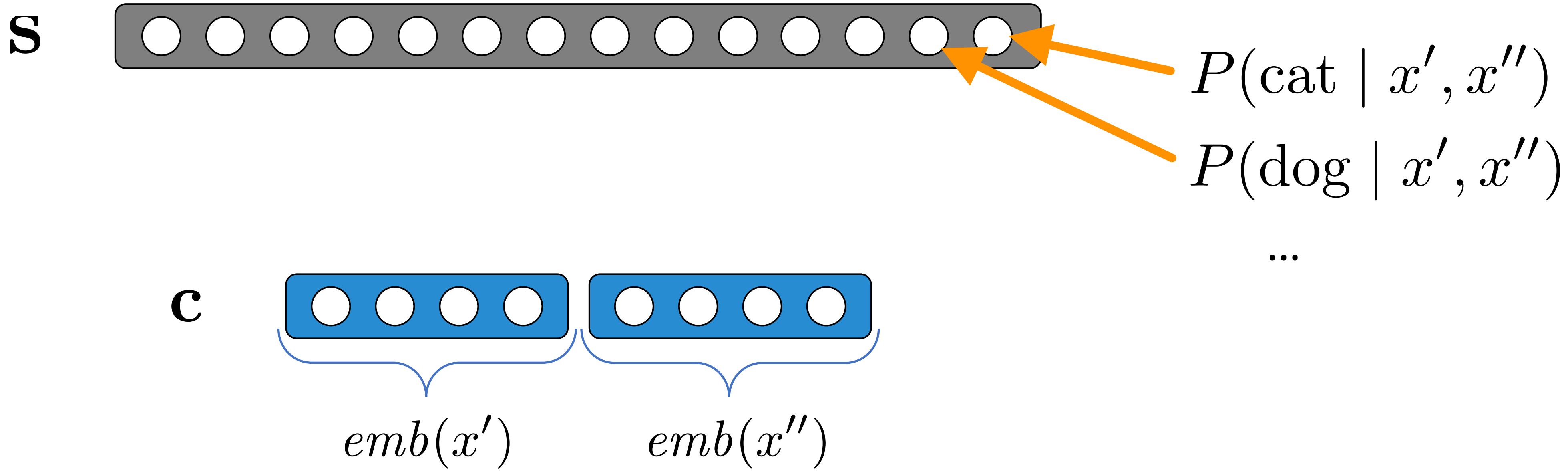


$$emb(\cdot) \in \mathbb{R}^d$$

$$\mathbf{c} = cat(emb(x'), emb(x''))$$

A Simple Neural Trigram Language Model

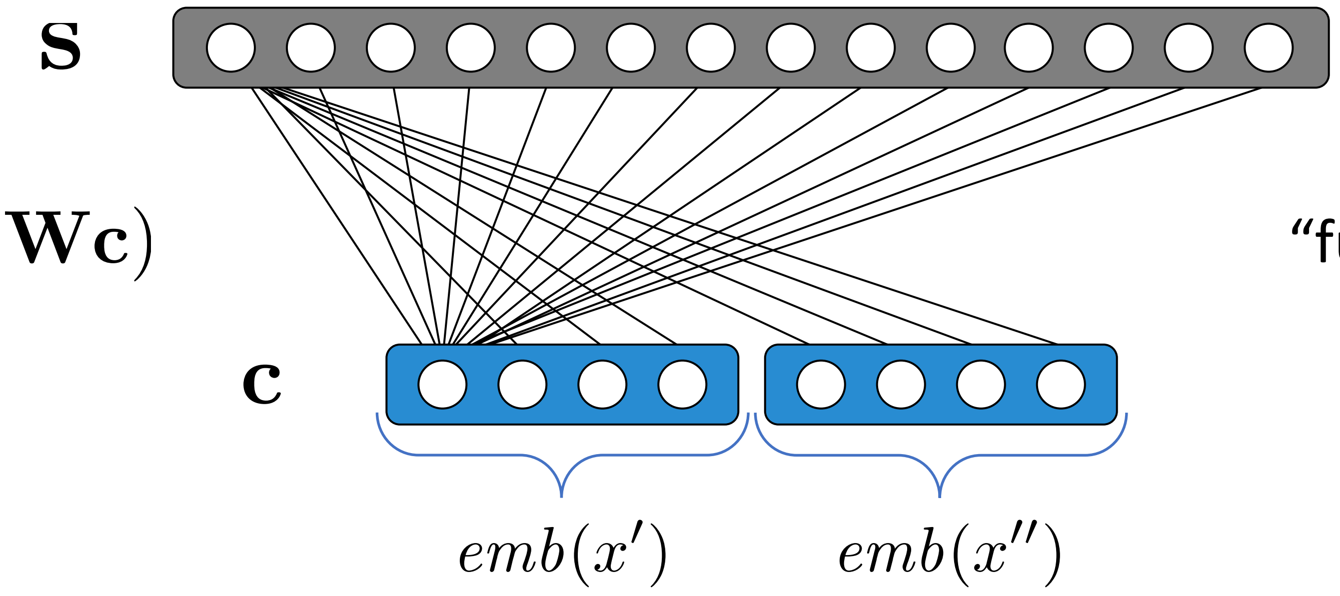
- output is a vector $\mathbf{s}$ containing probabilities of all possible next words:



A Simple Neural Trigram Language Model

- to get $\mathbf{s}$, do matrix multiplication of parameter matrix $\mathbf{W}$ and input, then "softmax" transformation

$$\mathbf{s} = \mathrm{softmax}(\mathbf{W}\mathbf{c})$$

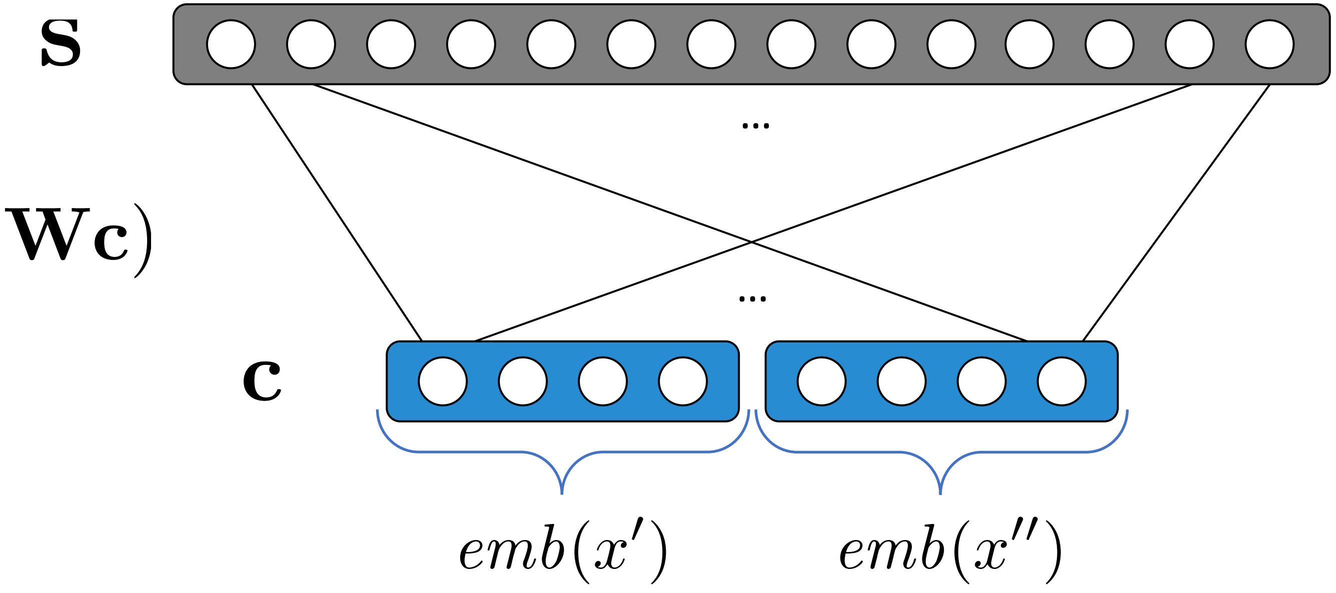


"fully-connected layer"

A Simple Neural Trigram Language Model

- to get $\mathbf{s}$, do matrix multiplication of parameter matrix $\mathbf{W}$ and input, then "softmax" transformation

$$\mathbf{s} = \text{softmax}(\mathbf{W}\mathbf{c})$$



softmax

- function that maps a vector $\mathbf{v}$ of real values (called "logits" or "scores") to a vector $\mathbf{p}$ of probabilities:

$$\mathbf{p} = \mathrm{softmax}(\mathbf{v}) = \frac{\exp\{\mathbf{v}\}}{\sum_i \exp\{v_i\}}$$

  - exponentiate scores (this makes them positive), then normalize to get probabilities

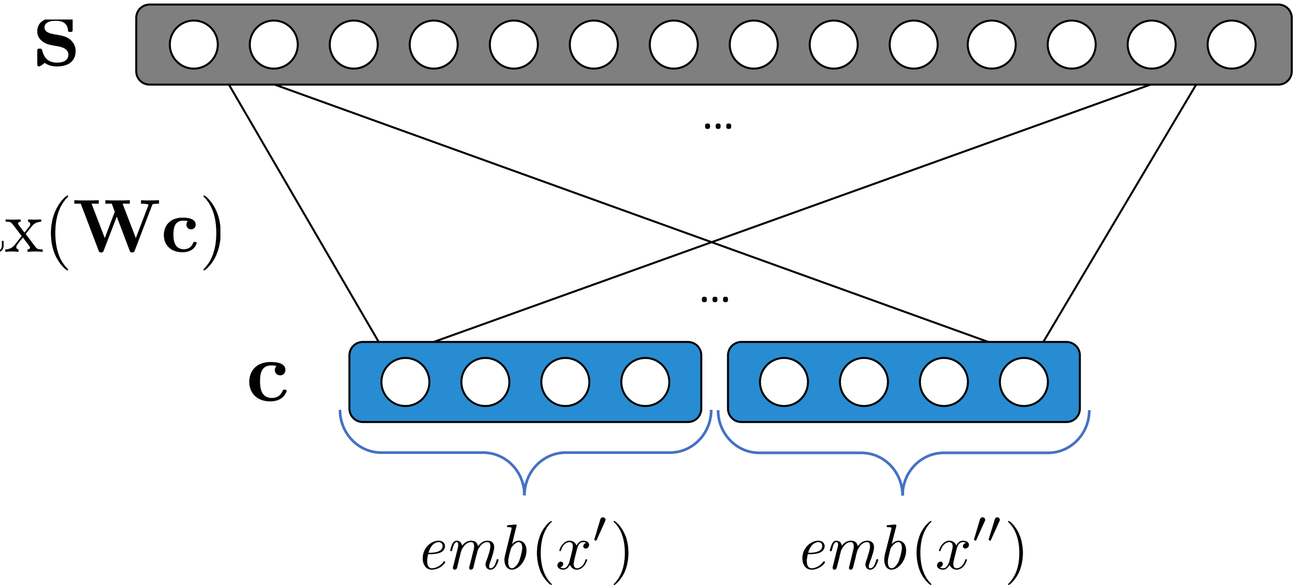
- using scalar notation (computing a single probability p_i):

$$p_i = \frac{\exp\{v_i\}}{\sum_j \exp\{v_j\}} \qquad\qquad p_i \propto \exp\{v_i\}$$

A Simple Neural Trigram Language Model

- What are the dimensionalities?

$$\mathbf{s} = \mathrm{softmax}(\mathbf{W}\mathbf{c})$$



dimensionalities:

$$emb(\cdot) \in \mathbb{R}^{d}$$

$$\mathbf{c} \in \mathbb{R}^{2d}$$

$$\mathbf{s} \in \mathbb{R}^{|\mathcal{V}|}$$

$$\mathbf{W} \in \mathbb{R}^{|\mathcal{V}| \times 2d}$$

A Simple Neural Trigram Language Model

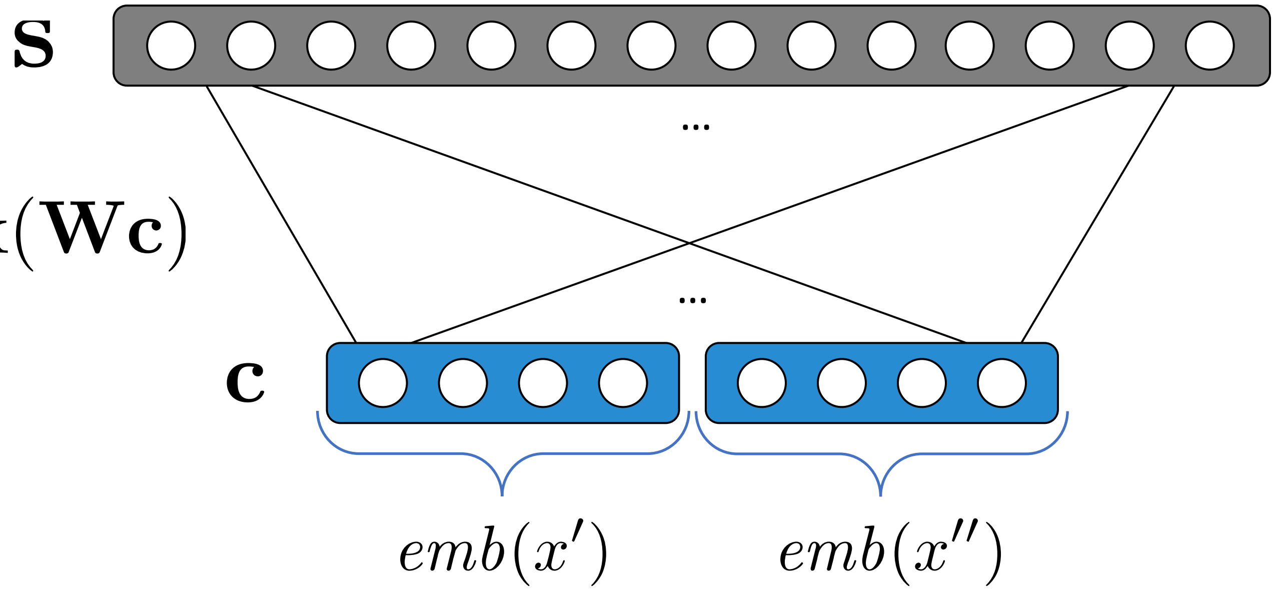
- what are the parameters in this model?

  $emb(\cdot)$ function: $|\mathcal{V}| \times d$ parameters $\mathbf{W}$: $|\mathcal{V}| \times 2d$ parameters

- how many total parameters are in this model?

  $|\mathcal{V}| \times 3d$

$$\mathbf{s} = \mathrm{softmax}(\mathbf{Wc})$$

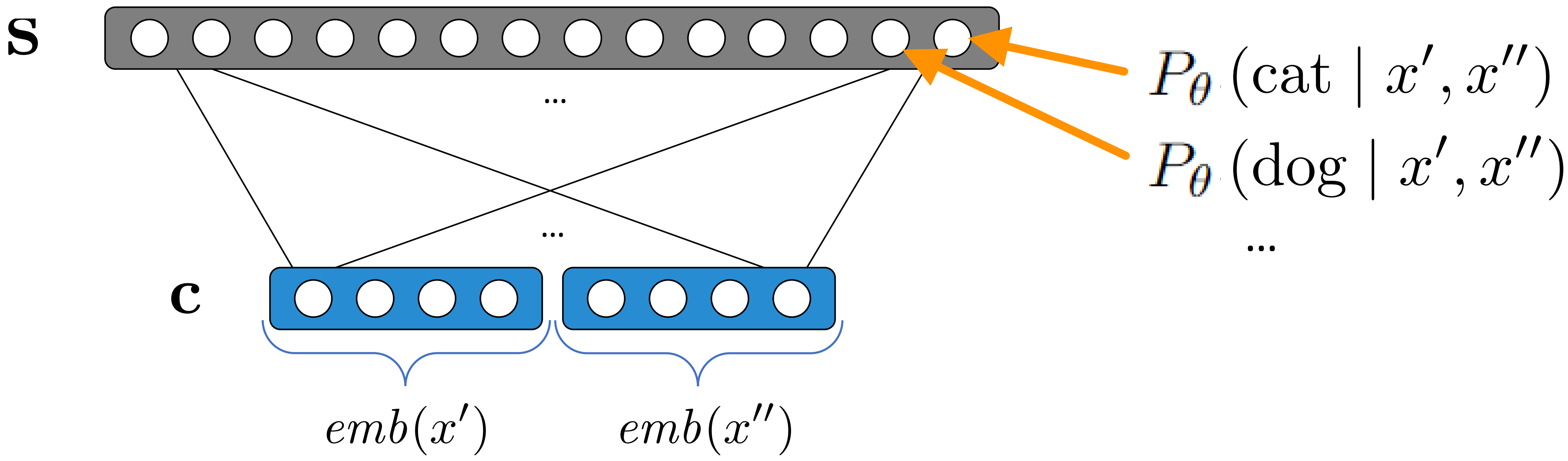


Comparing Models of $P(x \mid x', x'')$

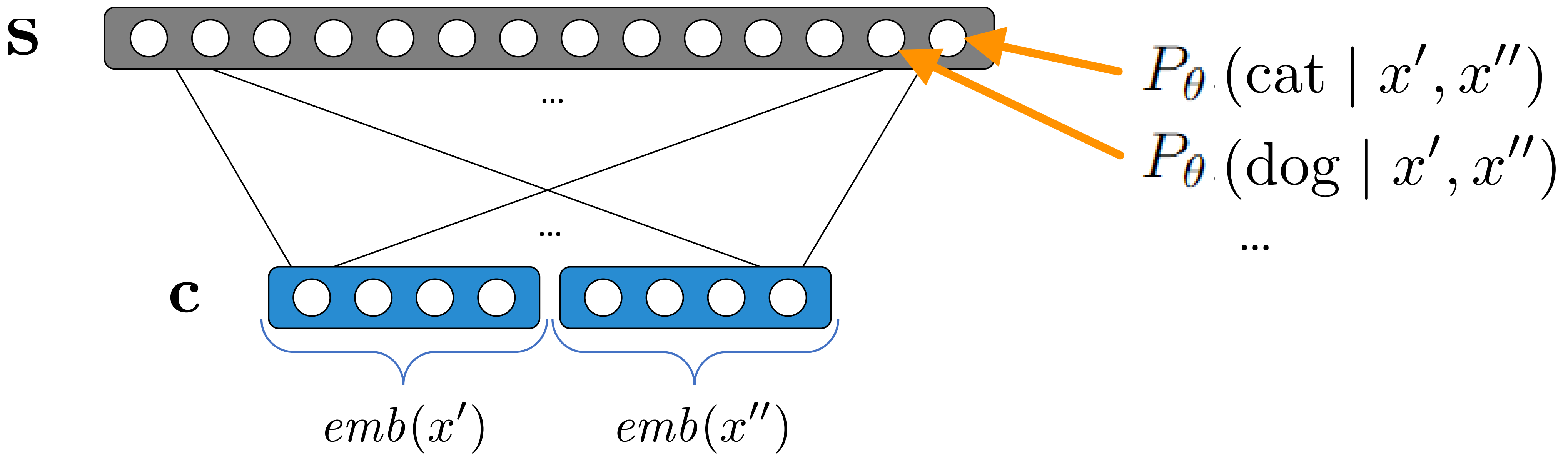
- **trigram language model**
  - separate parameters for every combination of x, x', x''
  - so, approx. $|\mathcal{V}|^3$ parameters
  - # parameters is exponential in *n*-gram size
  - most parameters are zero
  - even with smoothing, many parameters can remain zero

- **neural trigram language model**
  - only has $3d|\mathcal{V}|$ parameters
  - d can be chosen to scale # parameters up or down
  - # parameters linear in *n*-gram size
  - no parameters are zero
  - no explicit smoothing, though smoothing done implicitly via distributed representations

Learning



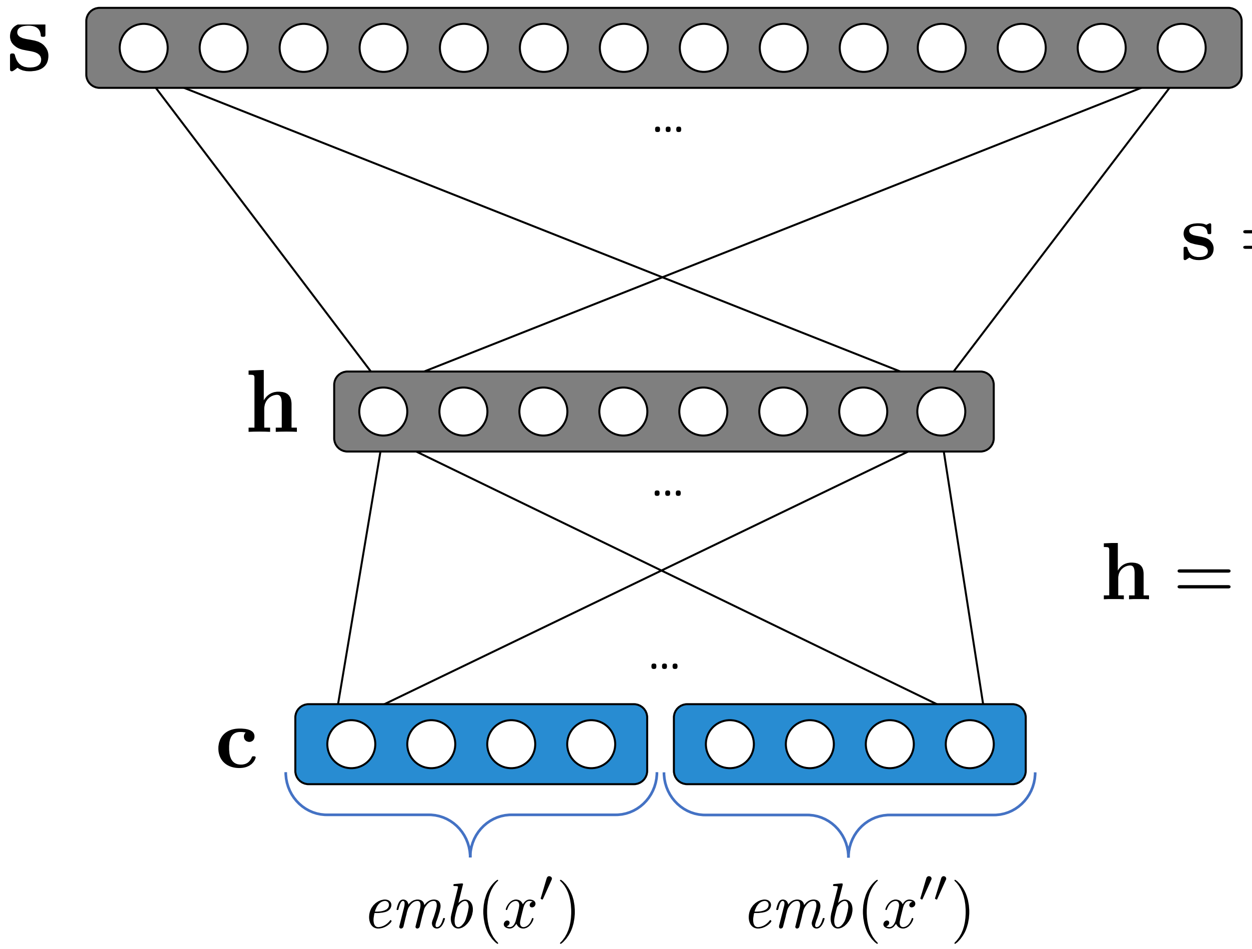
$\mathbf{s}$
...
...
$\mathbf{c}$
$emb(x')$
$emb(x'')$
$P_\theta(\text{cat} \mid x', x'')$
$P_\theta(\text{dog} \mid x', x'')$
...



- with *n*-gram models, we used maximum likelihood estimation (MLE), which has a simple closed-form solution
- however, with neural language models, MLE does not have a closed form!
- solution: minimize **log loss** using gradient-based optimization

$$\text{loss}_{\log}(\langle x', x'' \rangle, x, \boldsymbol{\theta}) = -\log P_{\boldsymbol{\theta}}(x \mid x', x'')$$

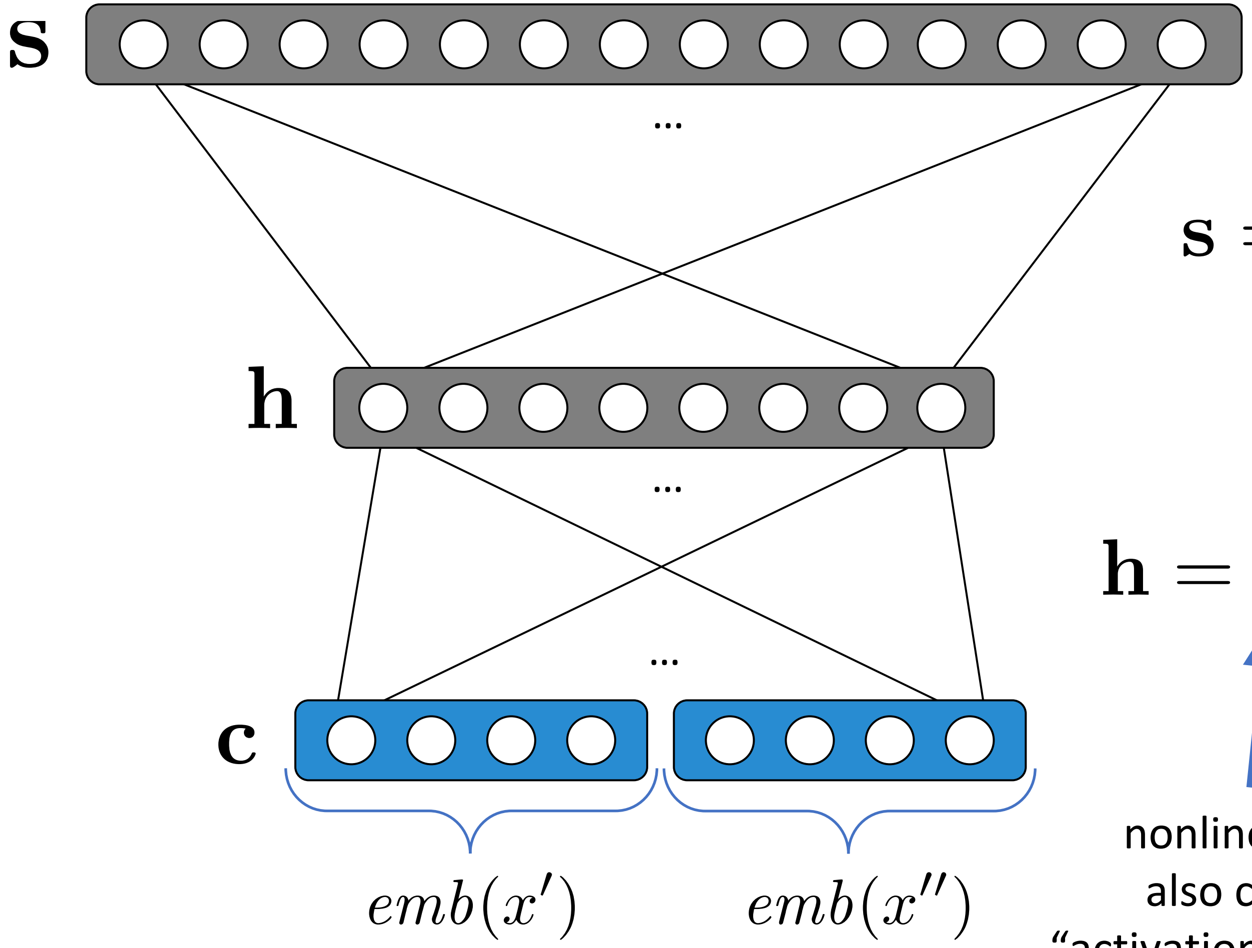
Adding a Hidden Layer



$$\mathbf{s} = \text{softmax}(\mathbf{W}\mathbf{h})$$

$$\mathbf{h} = \mathbf{U}\mathbf{c}$$

Adding a Hidden Layer

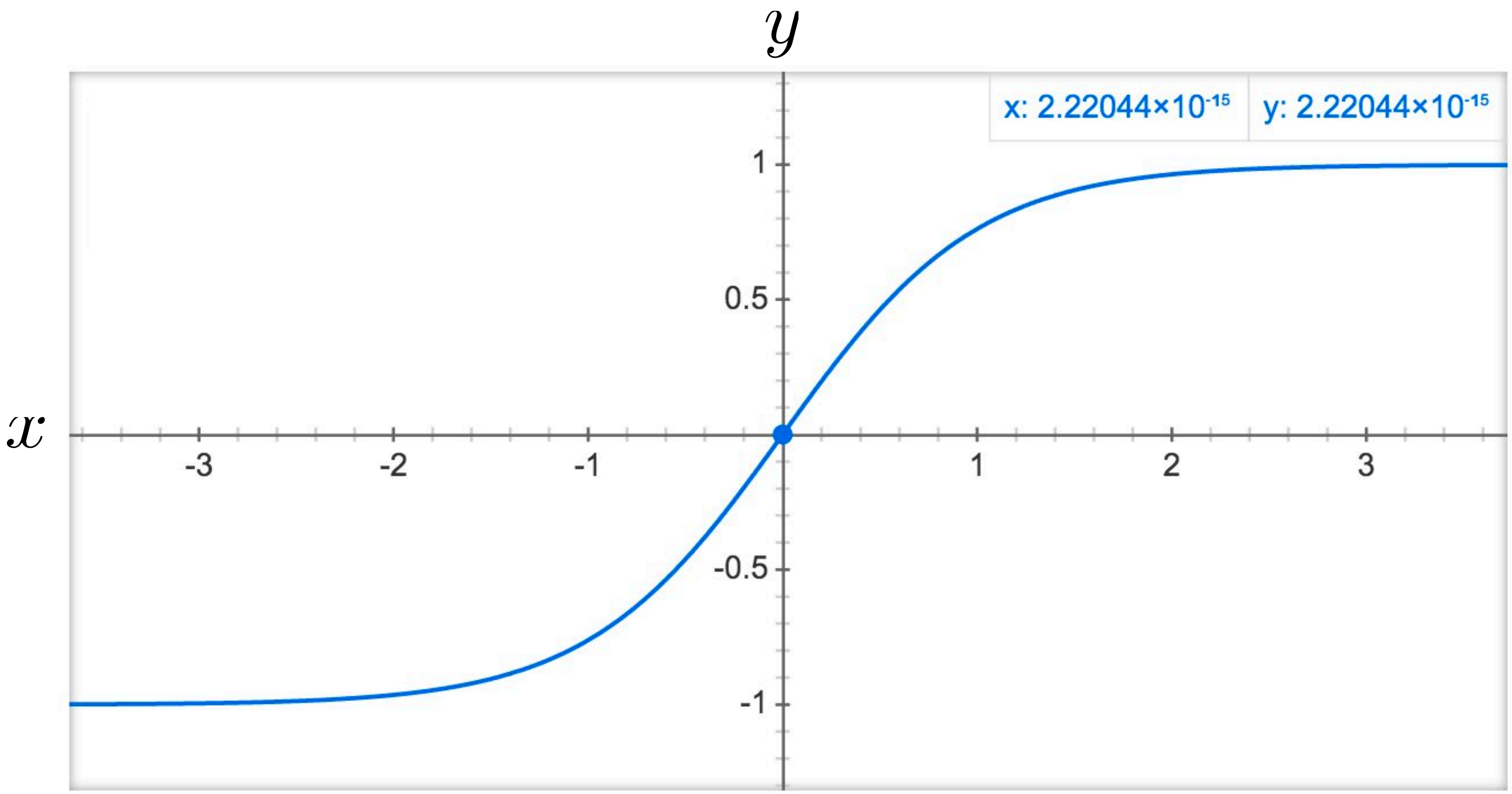


$$\mathbf{s} = \mathrm{softmax}(\mathbf{W}\mathbf{h})$$

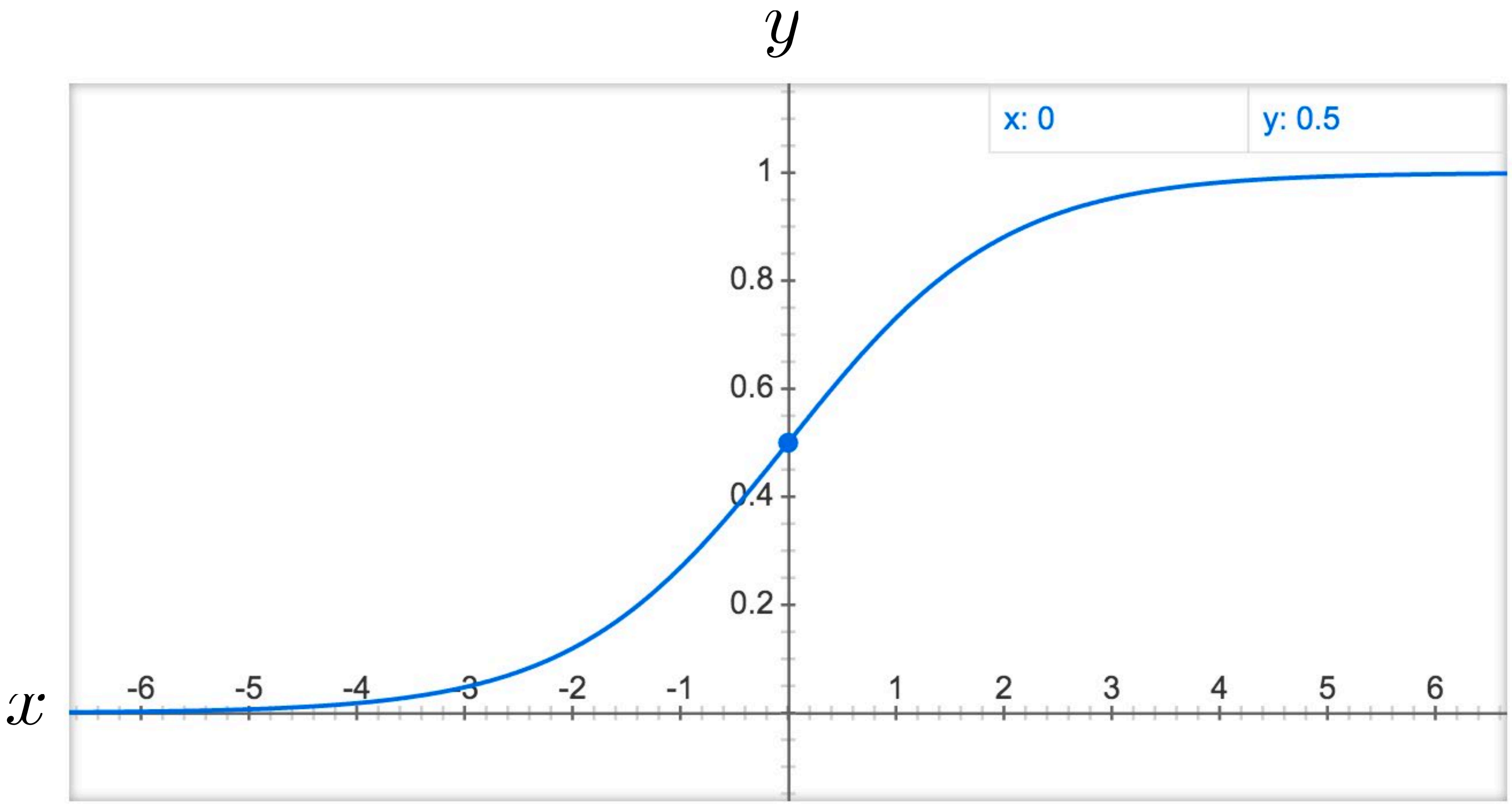
$$\mathbf{h} = g\left(\underbrace{\mathbf{U}\mathbf{c} + \mathbf{b}}_{\text{affine transformation}}\right)$$

nonlinearity,
also called
“activation function”

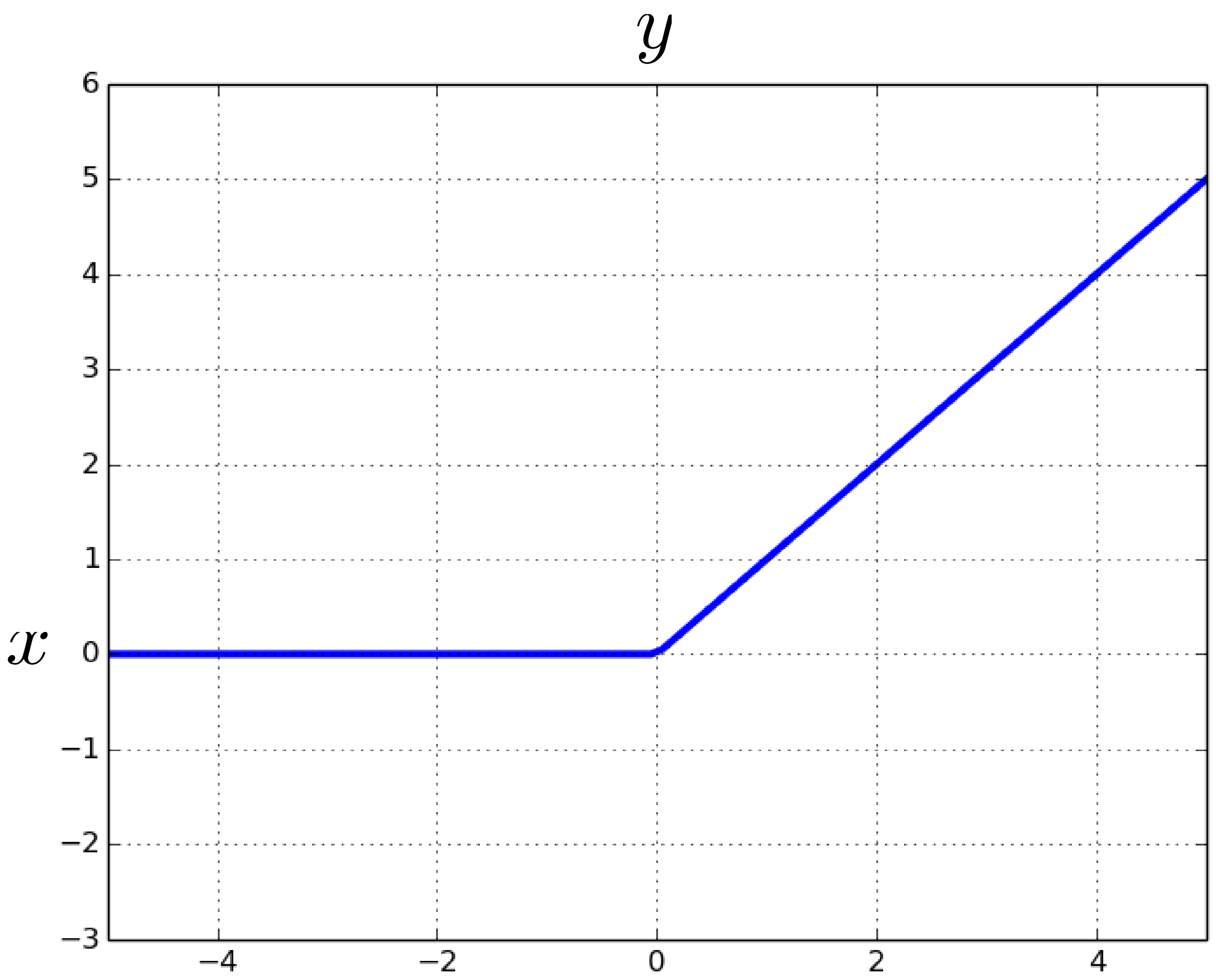
tanh: $y = \tanh(x)$



(logistic) sigmoid: $$y = \frac{1}{1 + \exp\{-x\}}$$



rectified linear unit (ReLU): $y = \max(0, x)$



Recap: Why nonlinearities?

network with 1 hidden layer:

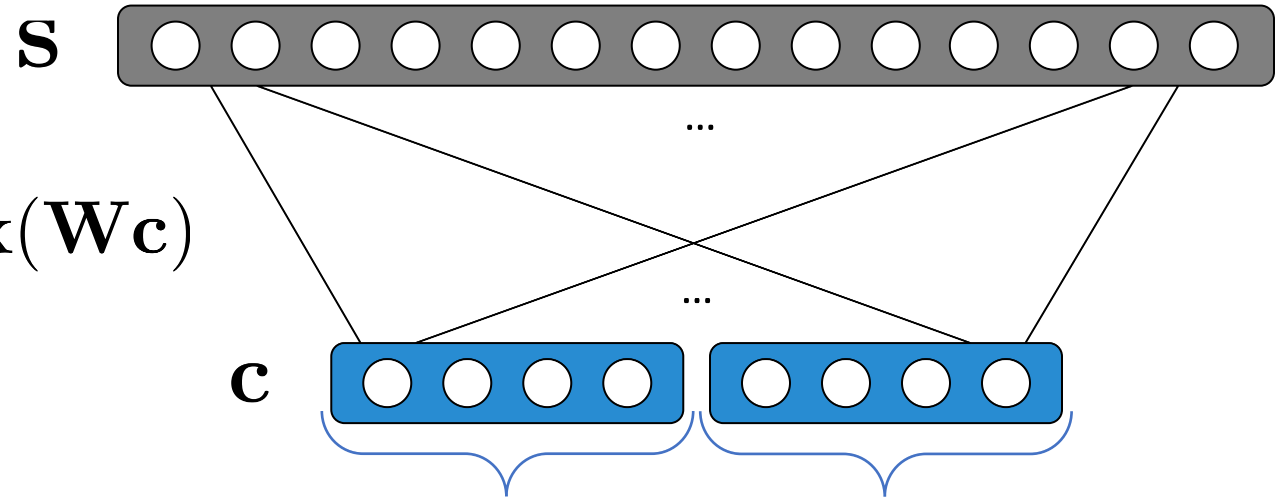
$$\mathbf{h}' = \mathbf{W}'\mathbf{h} + \mathbf{b}'$$

$$\mathbf{h} = g\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

- if *g* is linear, then we can rewrite the above as a single affine transformation (use distributivity of matrix multiplication)
- so, to benefit from multiple layers, we need some kind of nonlinearity

Language Modeling

- "The computer that I just put into the machine room on the fifth floor is crashing."
- "The computers that I just put into the machine room on the fifth floor are crashing."



$$\mathbf{s} = \mathrm{softmax}(\mathbf{W}\mathbf{c})$$

$$\underbrace{\qquad}_{emb(x')} \quad \underbrace{\qquad}_{emb(x'')}$$

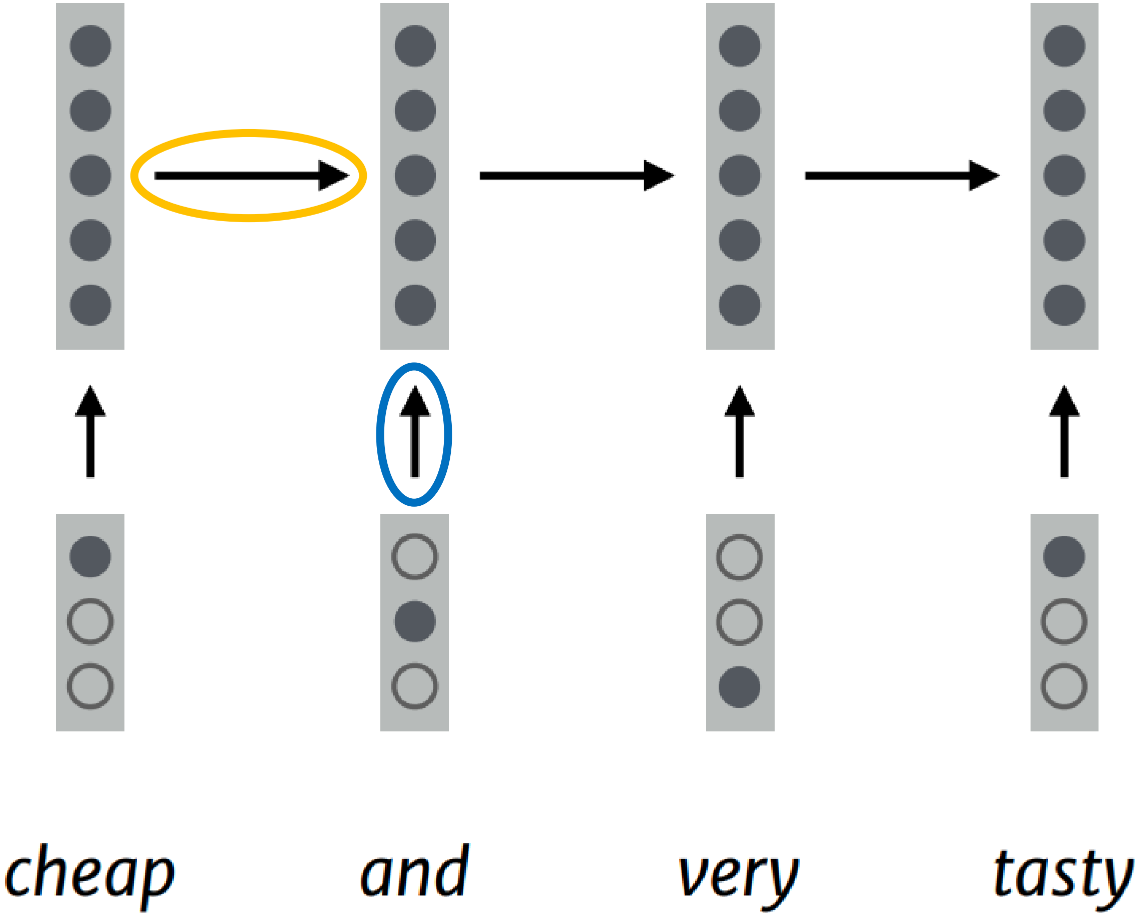
Language Modeling

- "The computer that I just put into the machine room on the fifth floor is crashing."
- "The computers that I just put into the machine room on the fifth floor are crashing."

- Feed-forward neural language models cannot model long-range dependencies
- Problem: How can we encode variable-sized input $\boldsymbol{x} = x_1, x_2, \ldots, x_n$ into fixed dimensional vector $\mathbf{c}$ so we can apply $\mathbf{s} = \mathrm{softmax}(\mathbf{Wc})$
- Summing, max-pooling…? Recurrence?

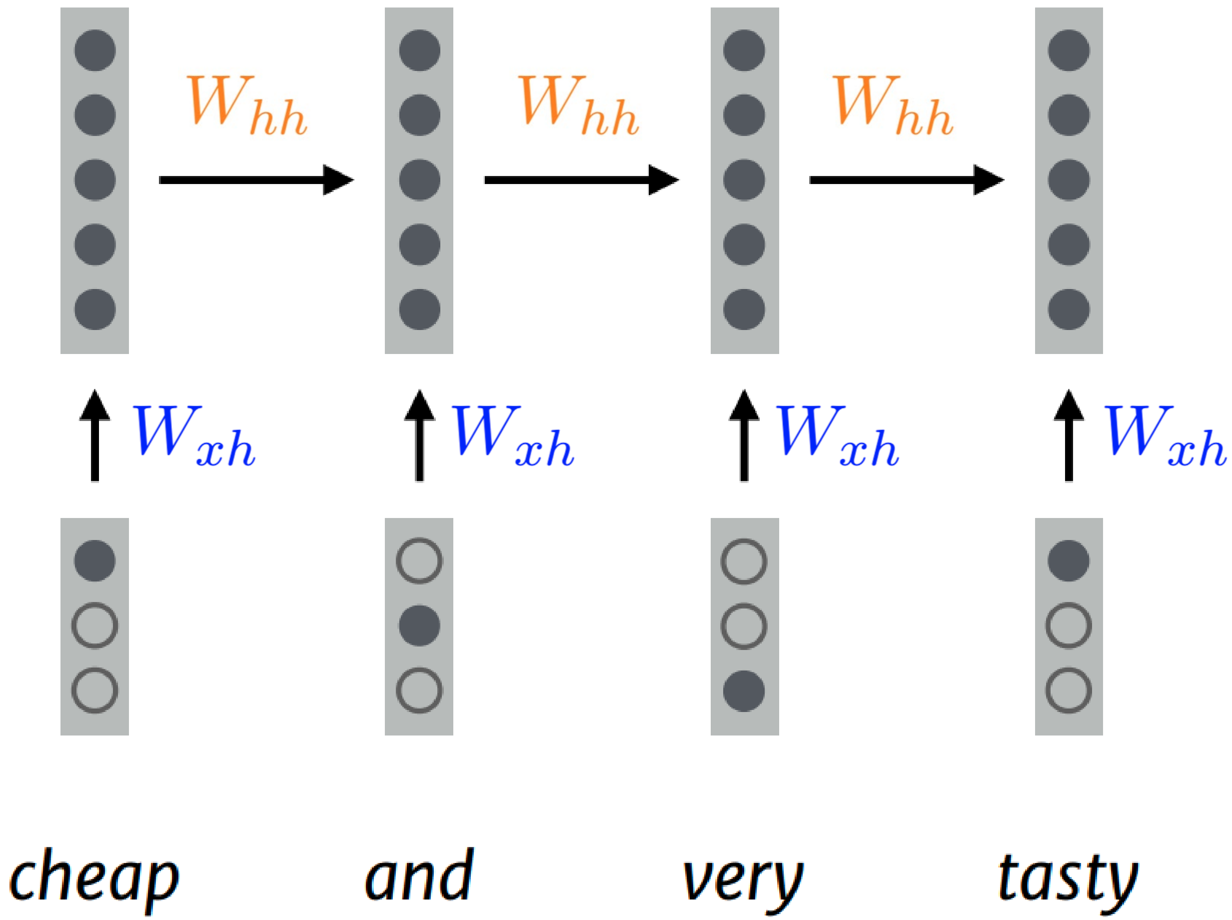
RNN Language Model

- Hidden state is a function of previous hidden state and current input
- Same weights at each state!

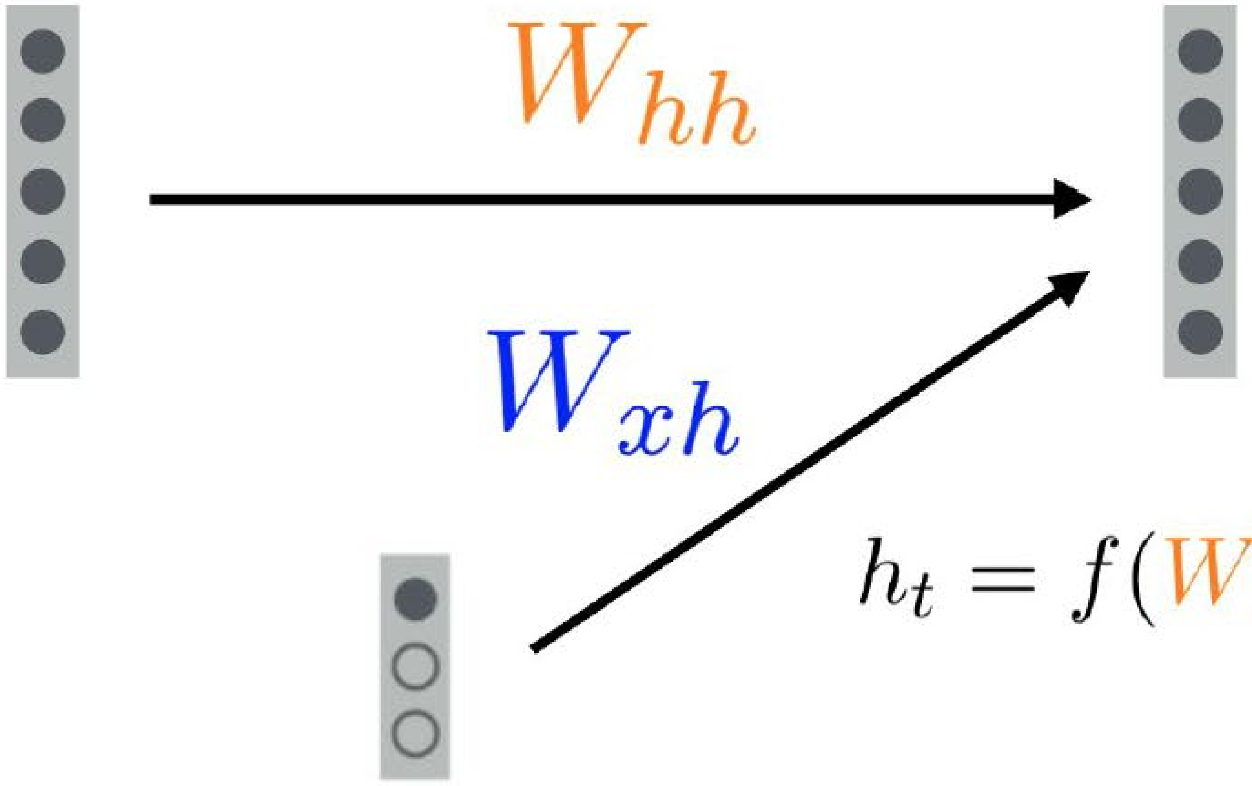


RNN Language Model

- Hidden state is a function of previous hidden state and current input
- Same weights at each state!

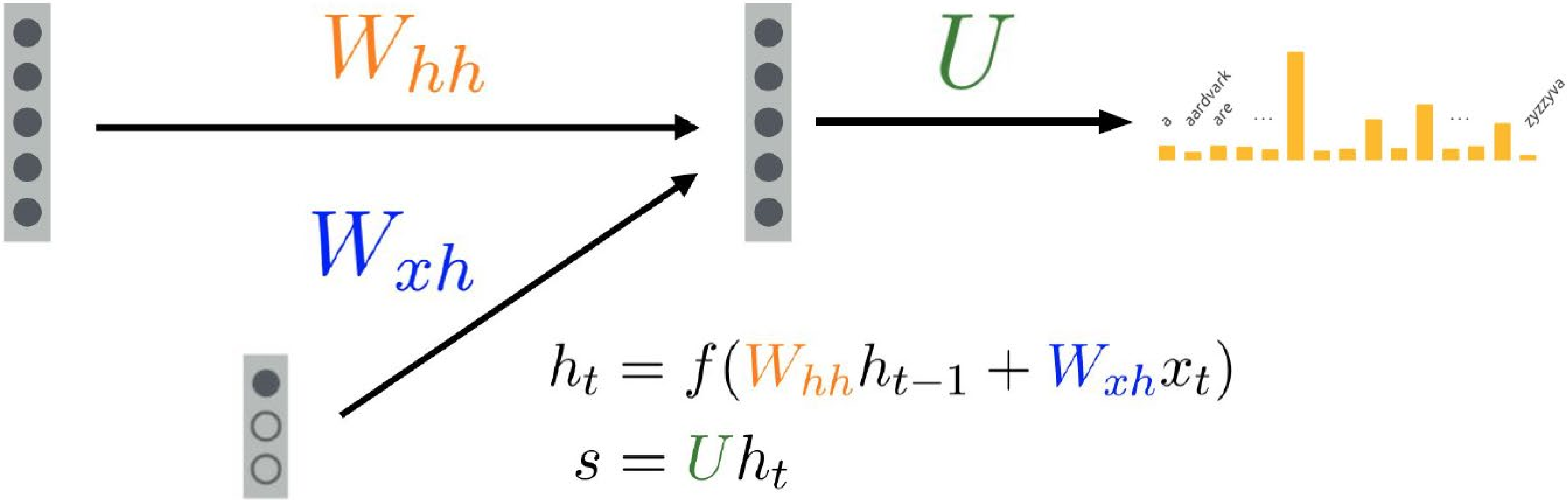


RNN Language Model



$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t)$$

RNN Language Model

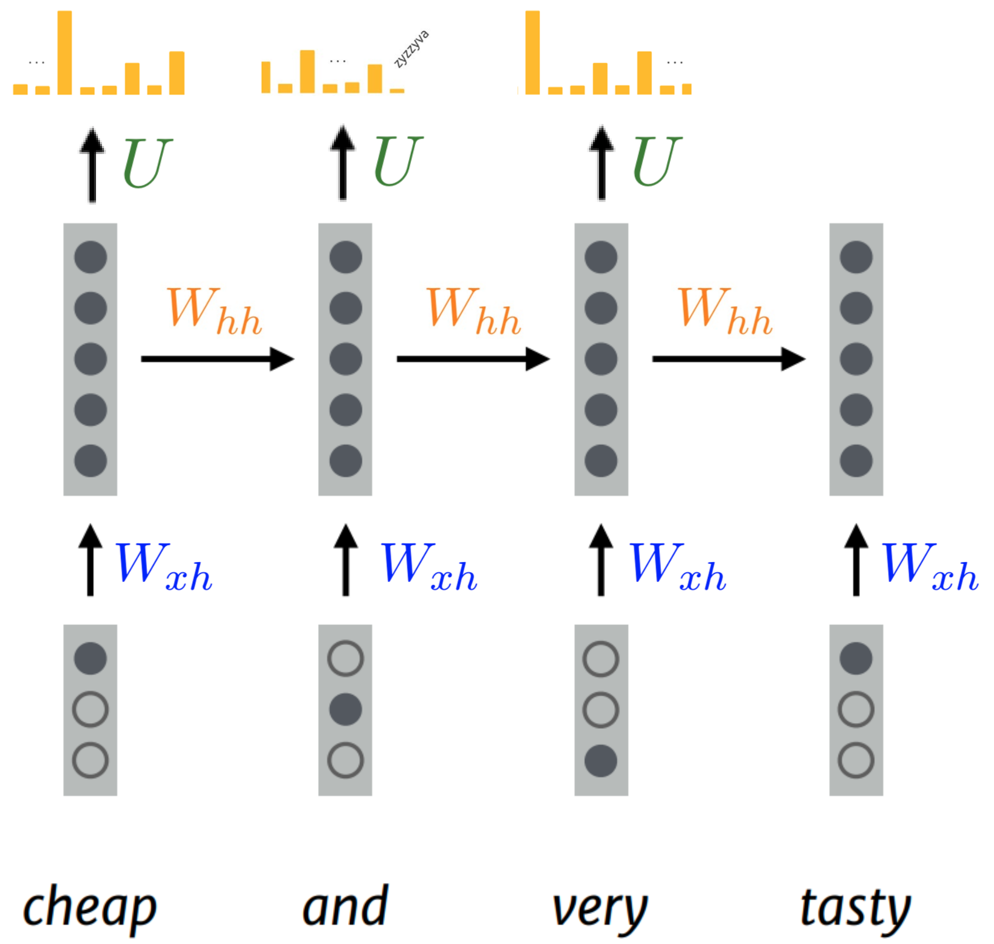


$$h_t = f(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$s = U h_t$$

$$p_\theta(x_{t+1} \mid x_1, \ldots, x_t) = \mathrm{softmax}(s)_{x_{t+1}}$$

RNN Language Model



No Markov assumption!

RNN Language Model

$$W_{hh} = \begin{pmatrix} 0 & 0 \\ I_{d\times d} & 0 \end{pmatrix}$$

$2d \times 2d$ block matrix

$$W_{xh} = \begin{pmatrix} W \\ 0 \end{pmatrix}$$

$2d \times V$ matrix

$f =$ identity

What is $h_t = f(W_{hh}h_{t-1} + W_{xh}x_t)$?

RNN Language Model

$$W_{hh} = \begin{pmatrix} 0 & 0 \\ I_{d\times d} & 0 \end{pmatrix}$$

$2d \times 2d$ block matrix

$$W_{xh} = \begin{pmatrix} W \\ 0 \end{pmatrix}$$

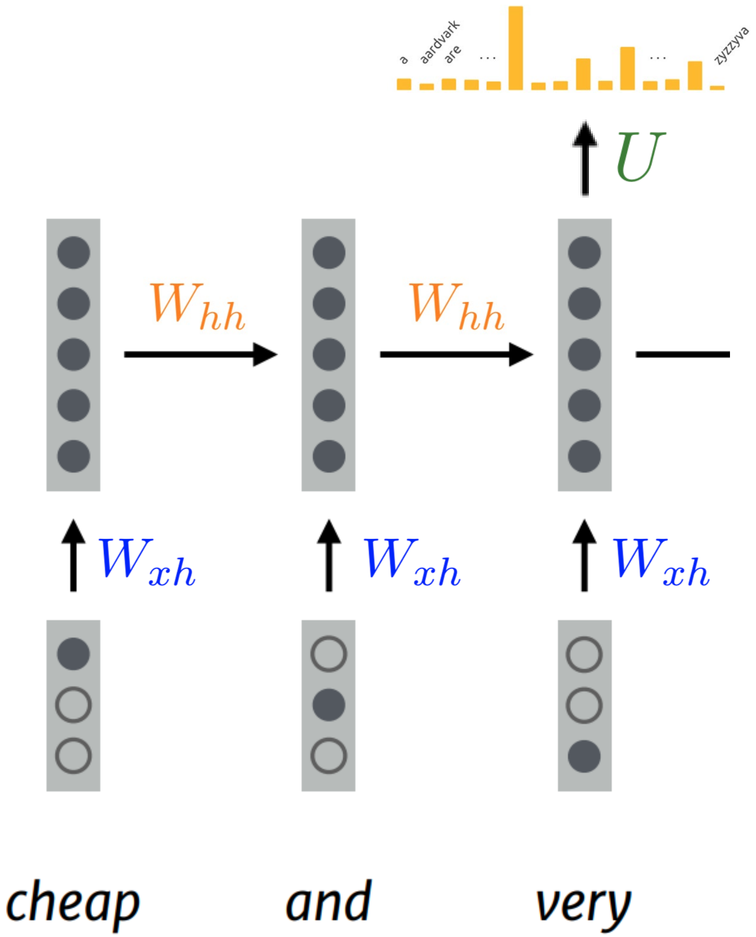
$2d \times V$ matrix

$$f = \text{identity}$$

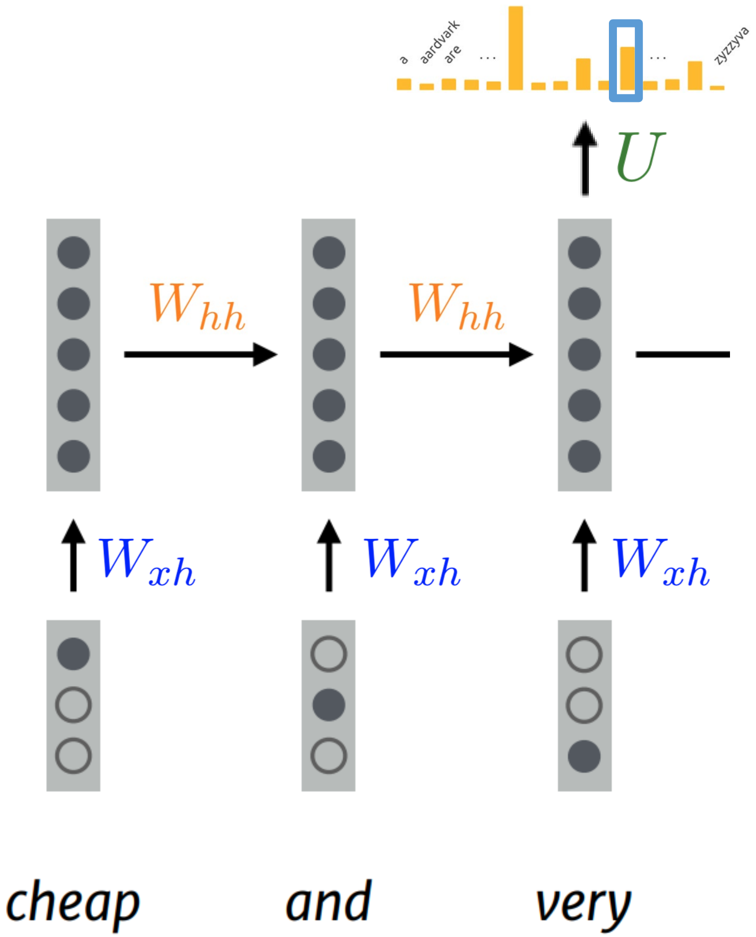
$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= [Wx_t\,, Wx_{t-1}]$$

RNN LMs (unsurprisingly) generalize feedforward LMs

RNN Language Model: Learning

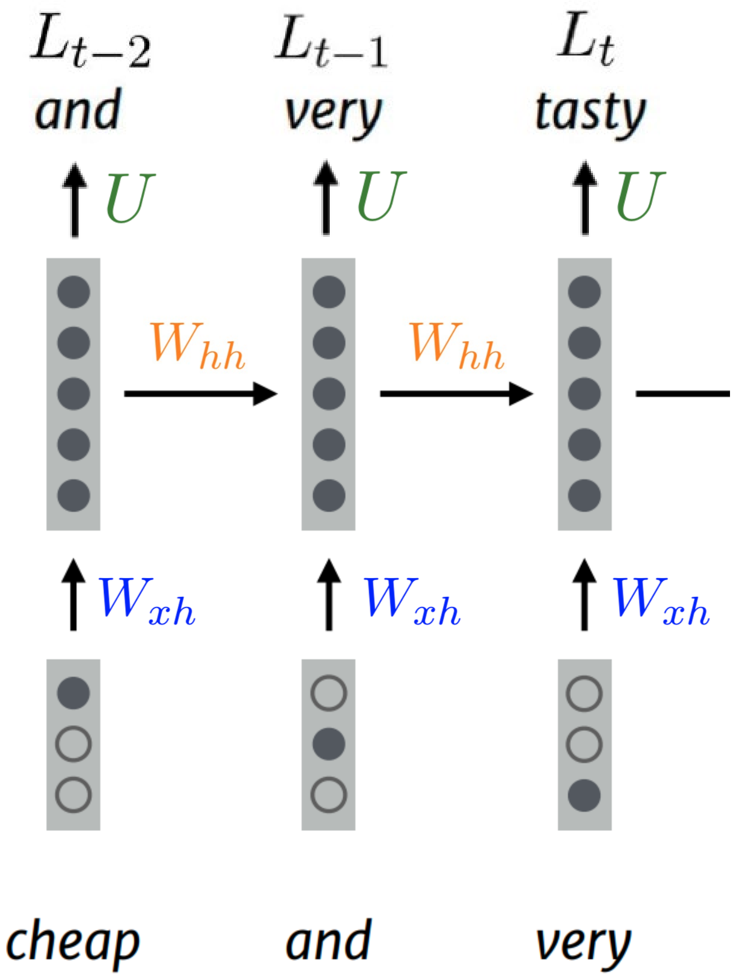


RNN Language Model: Learning



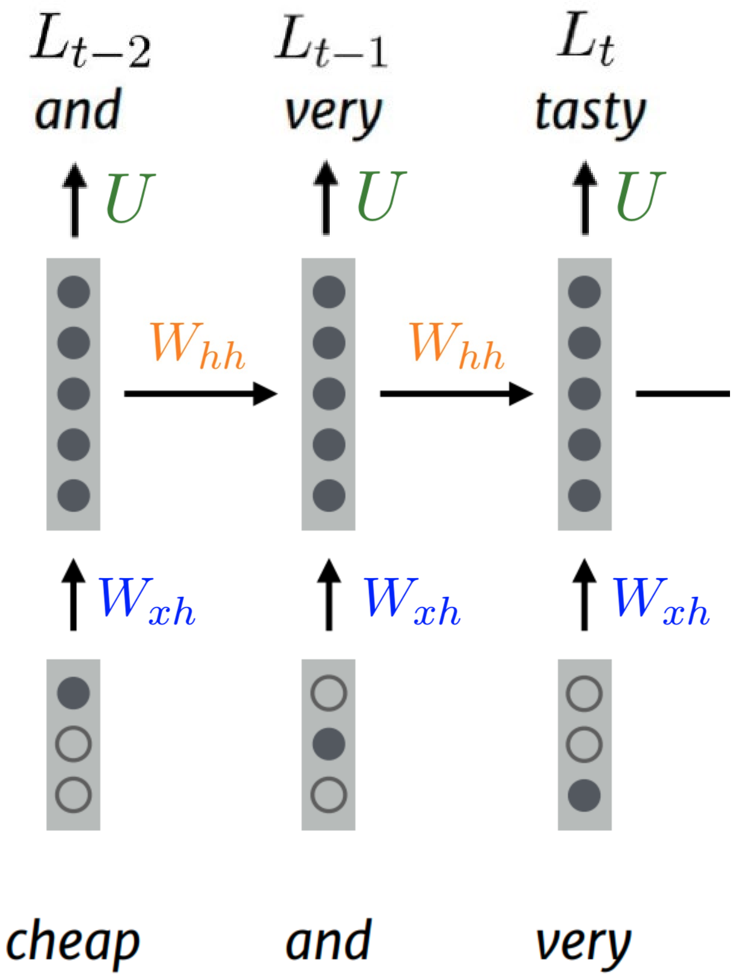
$$L_t = -\log p_\theta(x_t \mid x_1, \ldots, x_{t-1})$$

RNN Language Model: Learning



$$L_t = -\log p_\theta(x_t \mid x_1, \ldots, x_{t-1})$$

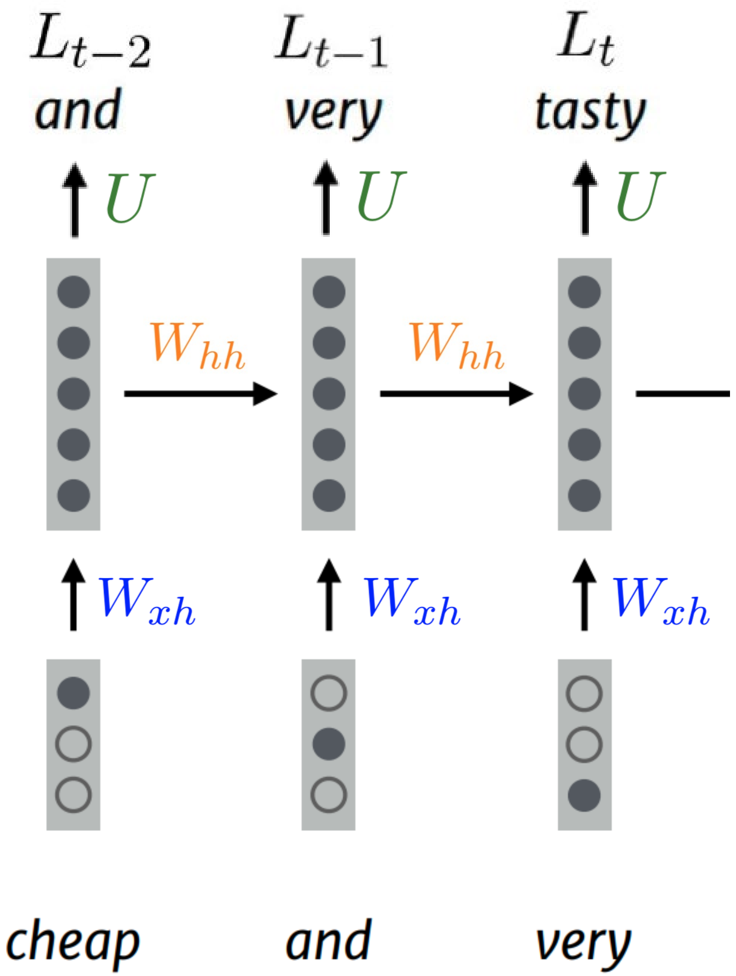
RNN Language Model: Learning



$$L_t = -\log p_\theta(x_t \,|\, x_1, \ldots, x_{t-1})$$

$$L = -\log p_\theta(x) = \sum_{t=1}^{T} L_t$$

RNN Language Model: Learning



$$L_t = -\log p_\theta(x_t \,|\, x_1, \ldots, x_{t-1})$$

$$L = -\log p_\theta(x) = \sum_{t=1}^{T} L_t$$

Obtain ∇_θ via backpropagation as usual ("backpropagation through time")

$$\theta = \{U, W_{hh}, W_{xh}\}$$

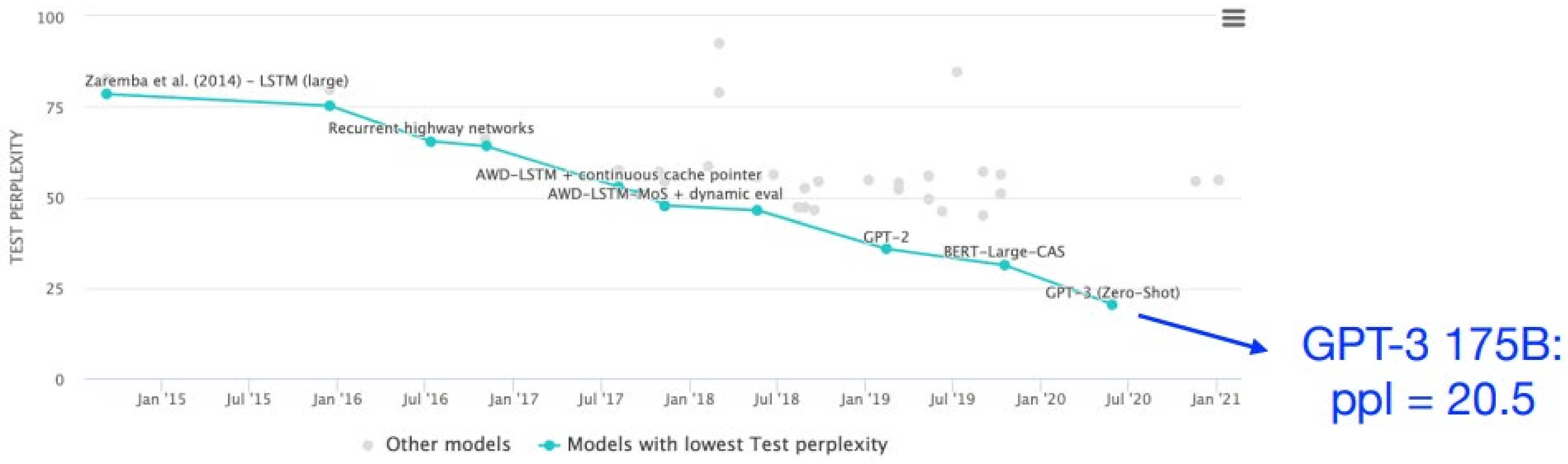
$$\theta \leftarrow \theta + \alpha \nabla_\theta \log p_\theta(x)$$

RNN Language Model: Learning

- RNNs can *theoretically* model infinite history?
- Practical Issues: gradient vanishing or exploding
- Empirical solutions:
  - Gradient clipping
  - RNN variants: LSTM, GRU, etc.

Language Modeling

- Language Modelling on Penn Treebank (Word Level)



https://paperswithcode.com/sota/language-modelling-on-penn-treebank-word

Language Modeling

English Penn Treebank

	Perplexity	Year
Count-based (5-grams)	141.2	2012
RNN	124.7	2012
Deep RNN	107.5	2013
LSTM	78.4	2014
Fancy LSTM cell found with RL	64.0	2016
Vanilla LSTM + Hyperparameter tuning	58.3	2018
Transformer	54.5	2019
Fancier LSTM cell	50.1	2020
GPT2	35.8	2019
GPT3	20.5	2020

RNN Language Model: Sampling

- How do we sample from $P_\theta(\boldsymbol{x})$?

RNN Language Model: Sampling

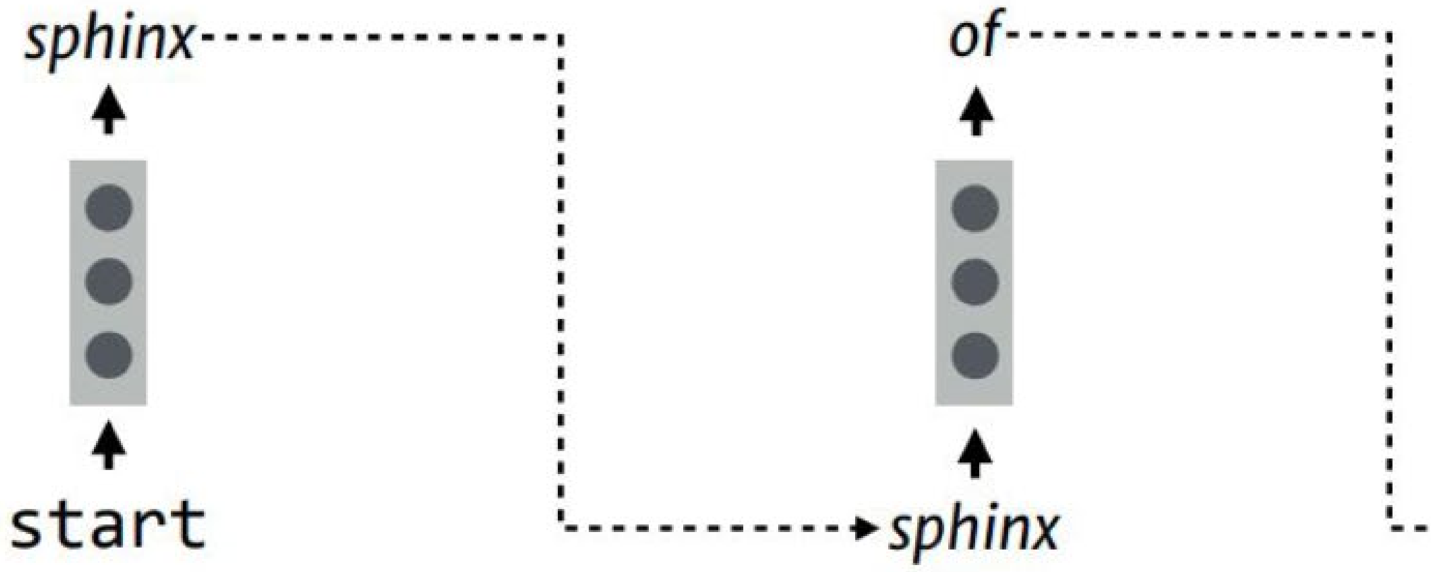
- How do we sample from $P_\theta(\boldsymbol{x})$?

$$x_1 \sim p(x_1 \mid \texttt{start})$$

RNN Language Model: Sampling

- How do we sample from $P_\theta(\boldsymbol{x})$?

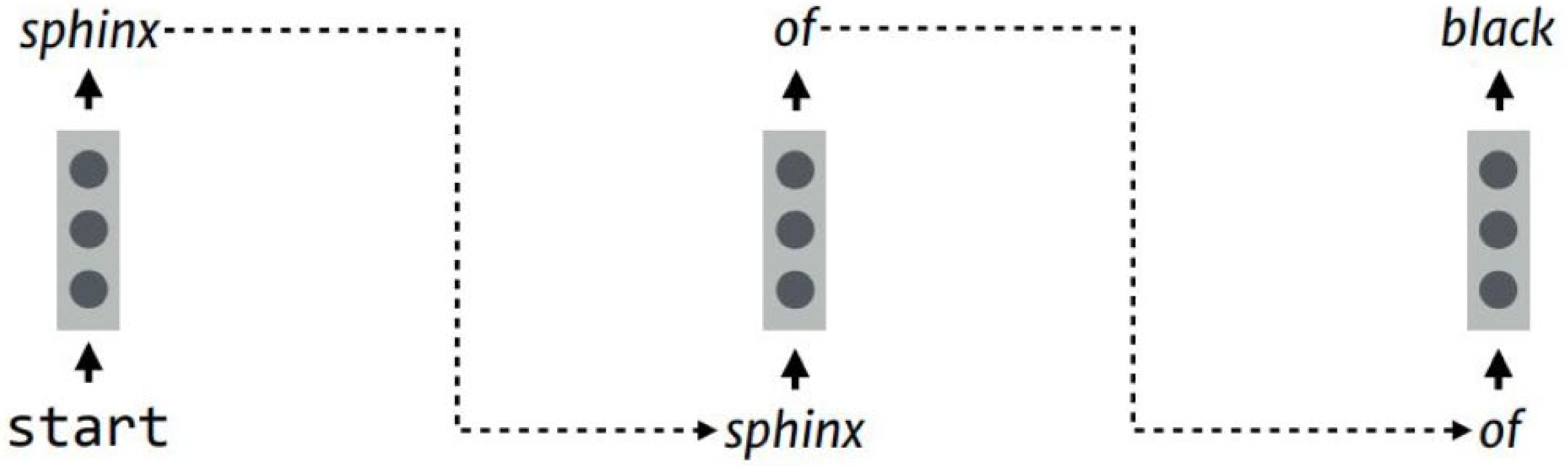
$x_1 \sim p(x_1 \,|\, \texttt{start})$ $\qquad$ $x_2 \sim p(x_2 \,|\, \textsf{sphinx})$



RNN Language Model: Sampling

- How do we sample from $P_\theta(\boldsymbol{x})$?

$x_1 \sim p(x_1 \,|\, \texttt{start})$ $\quad$ $x_2 \sim p(x_2 \,|\, \textsf{sphinx})$ $\quad$ $x_2 \sim p(x_2 \,|\, \textsf{sphinx of})$



Overview

- Neural language models
  - Feed-forward models
  - RNN models
  - Attention models

- Machine Translation & Sequence-to-sequence models
  - Machine translation
  - Encoder decoder structures
  - + Attention & applications

Sequence-to-sequence Models

- input is a sentence (typically)
- output is another sentence
  - Machine translation: representing its translation in another language

$\boldsymbol{x} =$ <s> ich werde das stoppen . </s>

$\boldsymbol{y} =$ <s> i ’m going to stop that . </s>

Sequence-to-sequence Models

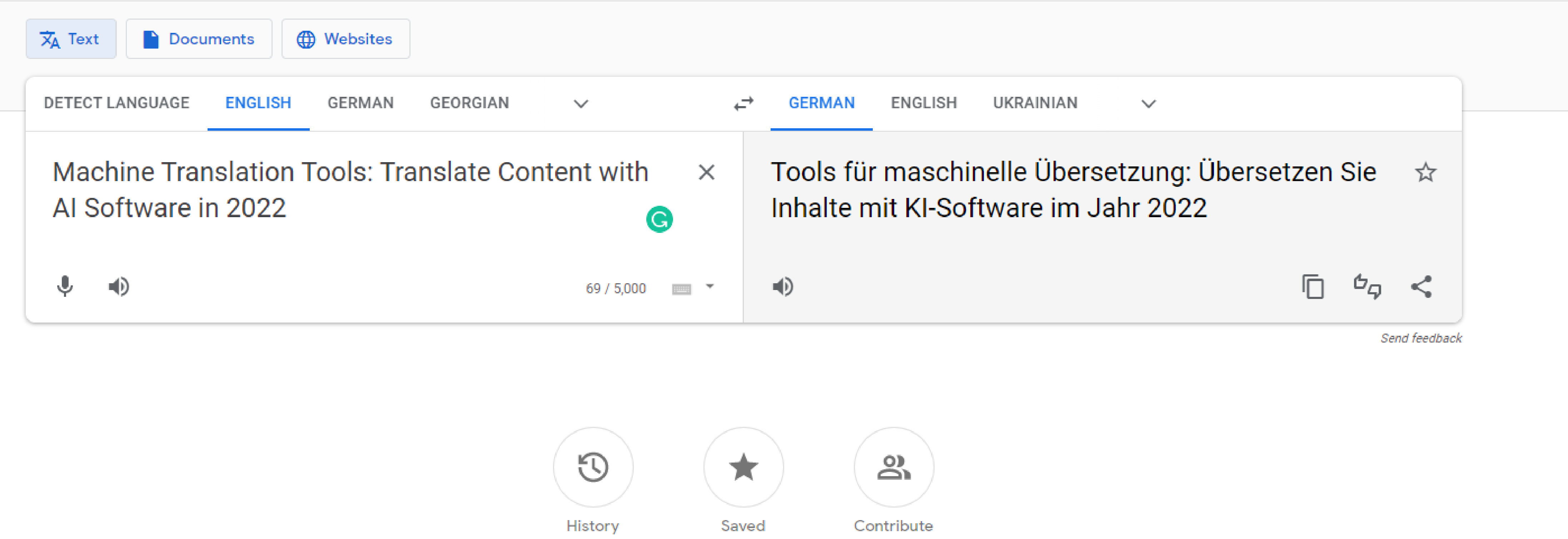
- input and output are sequences of symbols (not necessarily the same length)
- model:

$$p_\Theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{y}|} p_\Theta(y_t \mid \boldsymbol{x}, \boldsymbol{y}_{1:t-1})$$

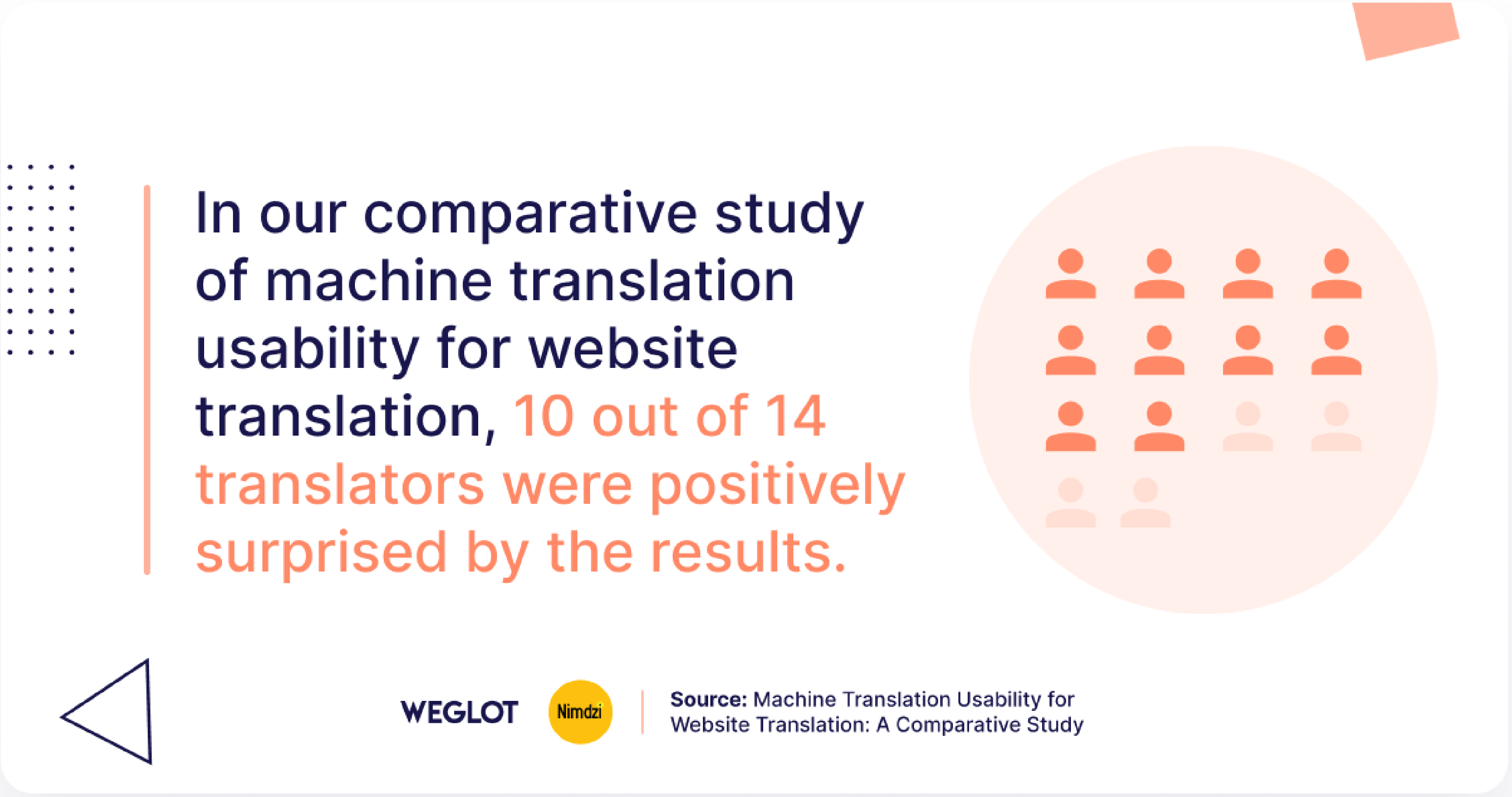
- training loss:

$$\text{loss}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{|\boldsymbol{y}|} -\log p_\Theta(y_t \mid \boldsymbol{x}, \boldsymbol{y}_{1:t-1})$$

Machine Translation



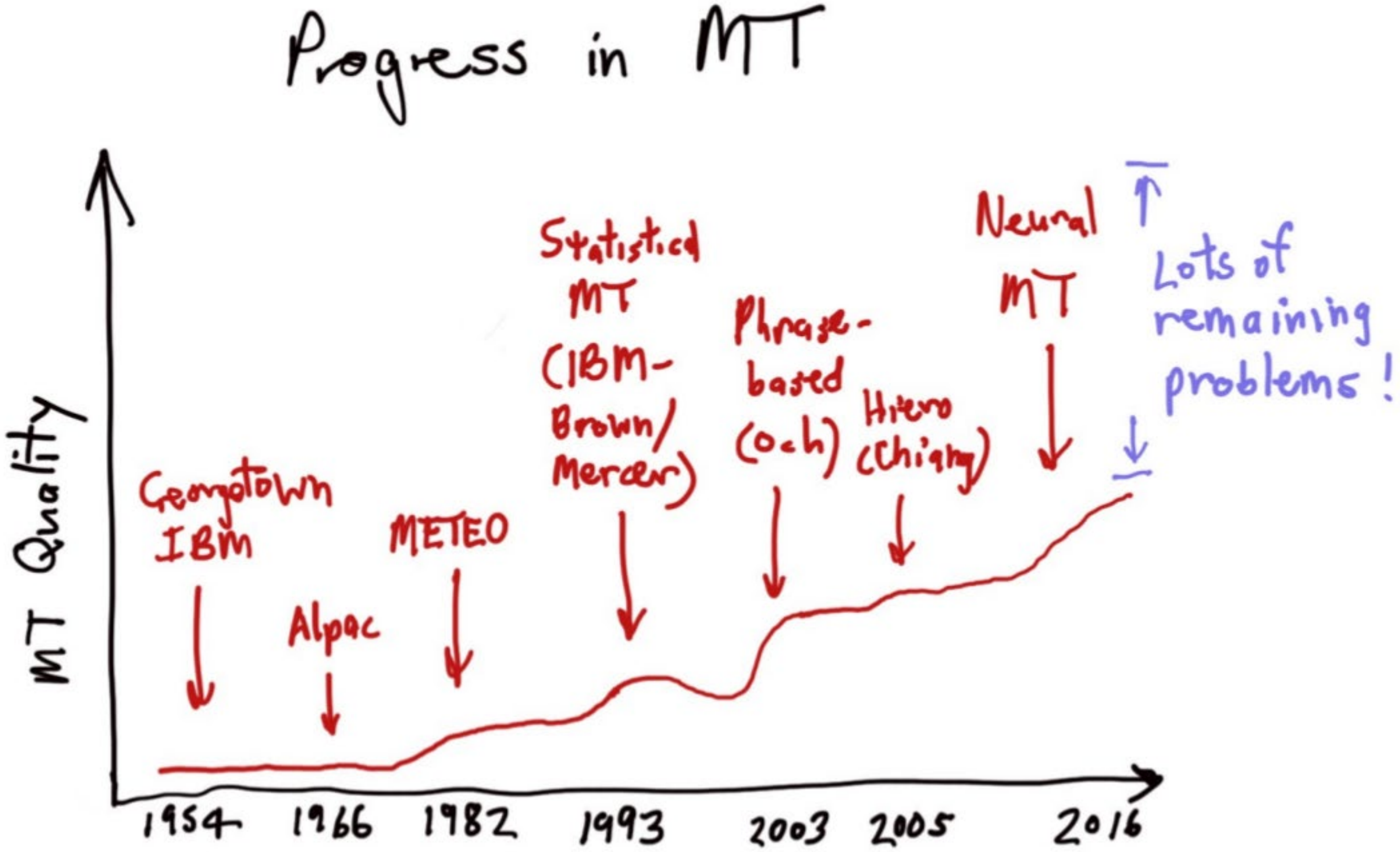
Machine Translation



Machine Translation

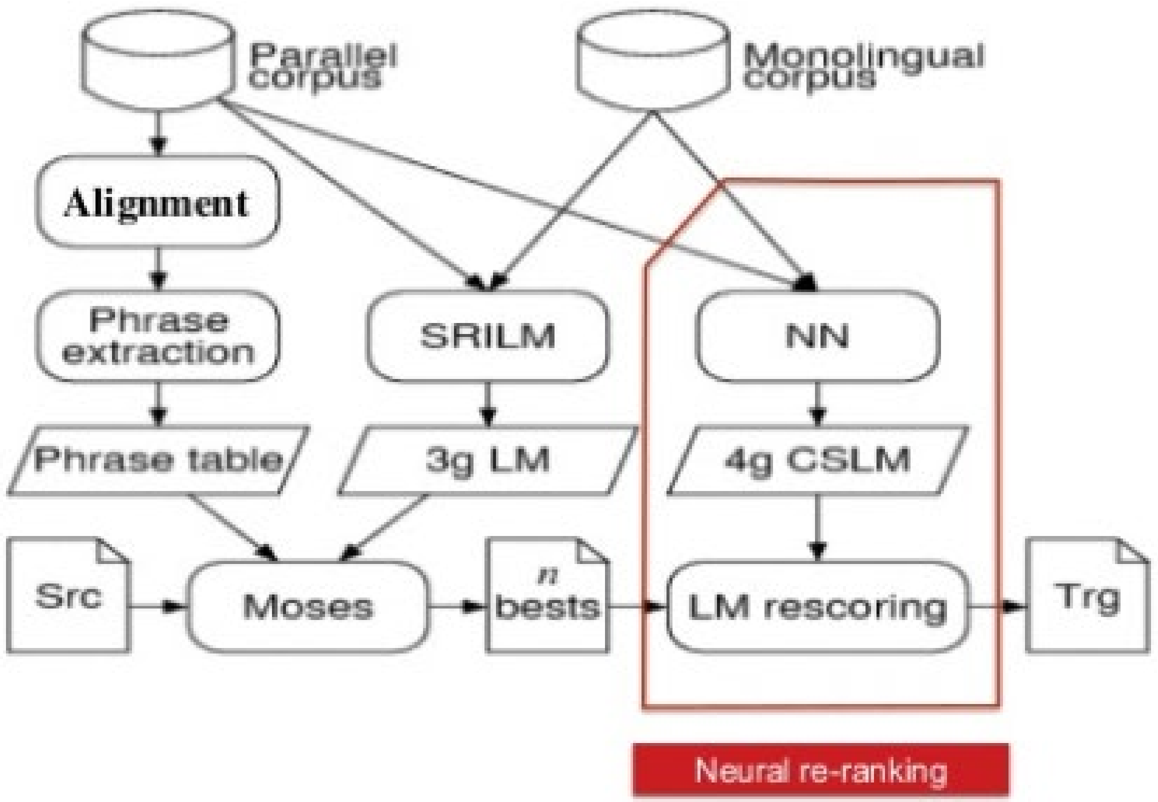
- $40 billion industry
- Google: translates 100 billion words a day

Machine Translation (MT) History



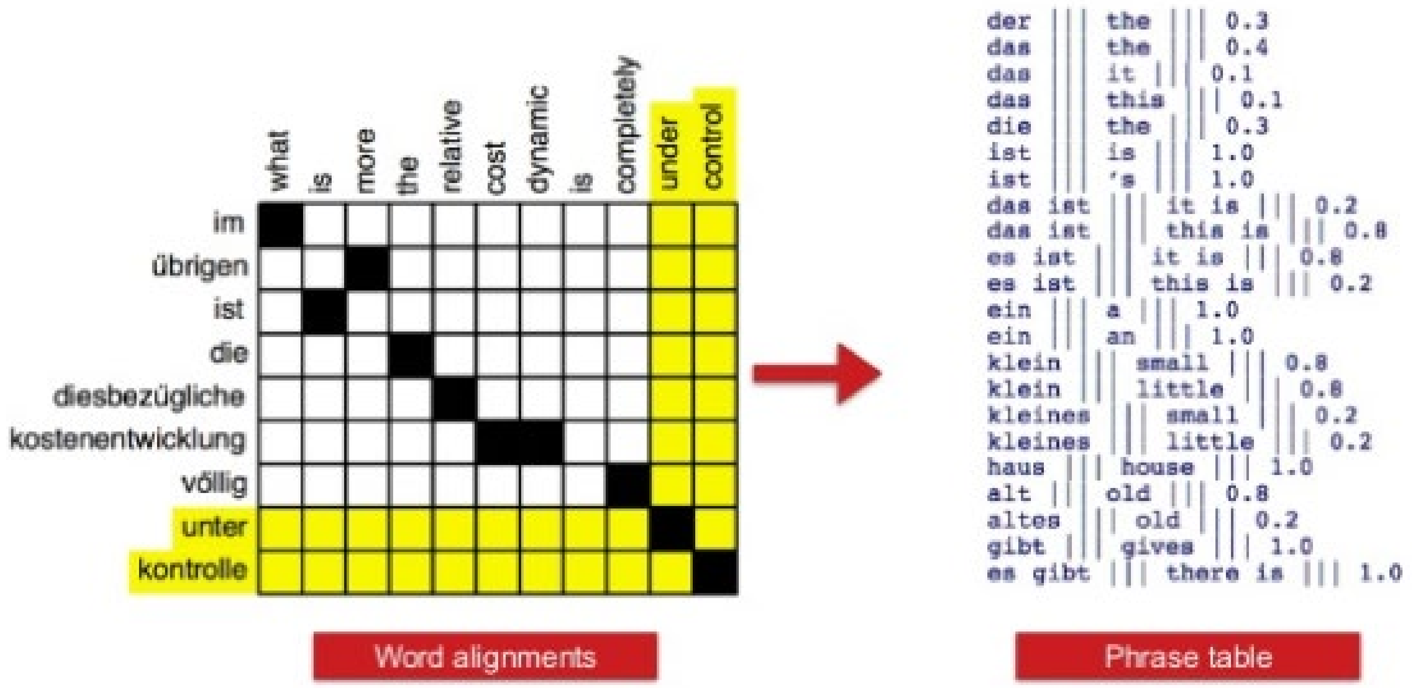
Phrase Based MT

- Complex pipelines, all trained separately



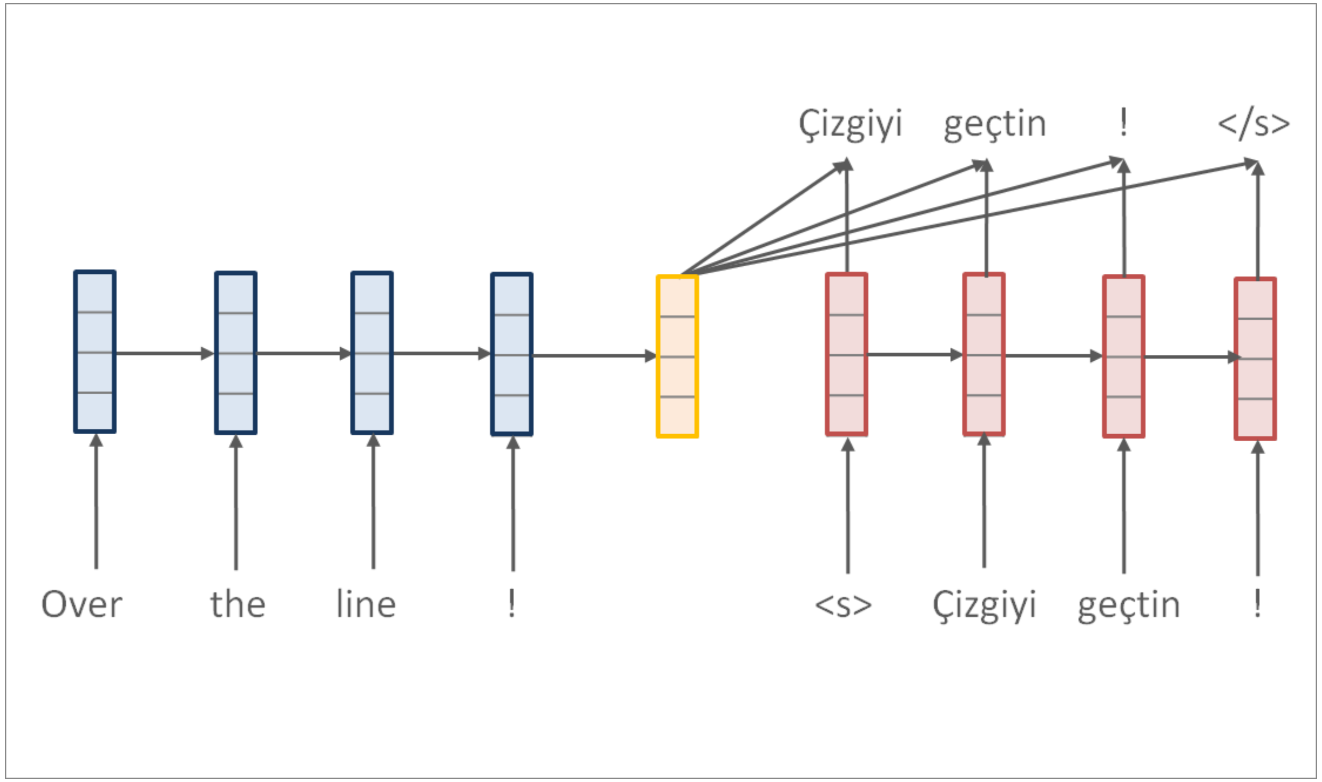
Phrase Based MT

- Alignment model



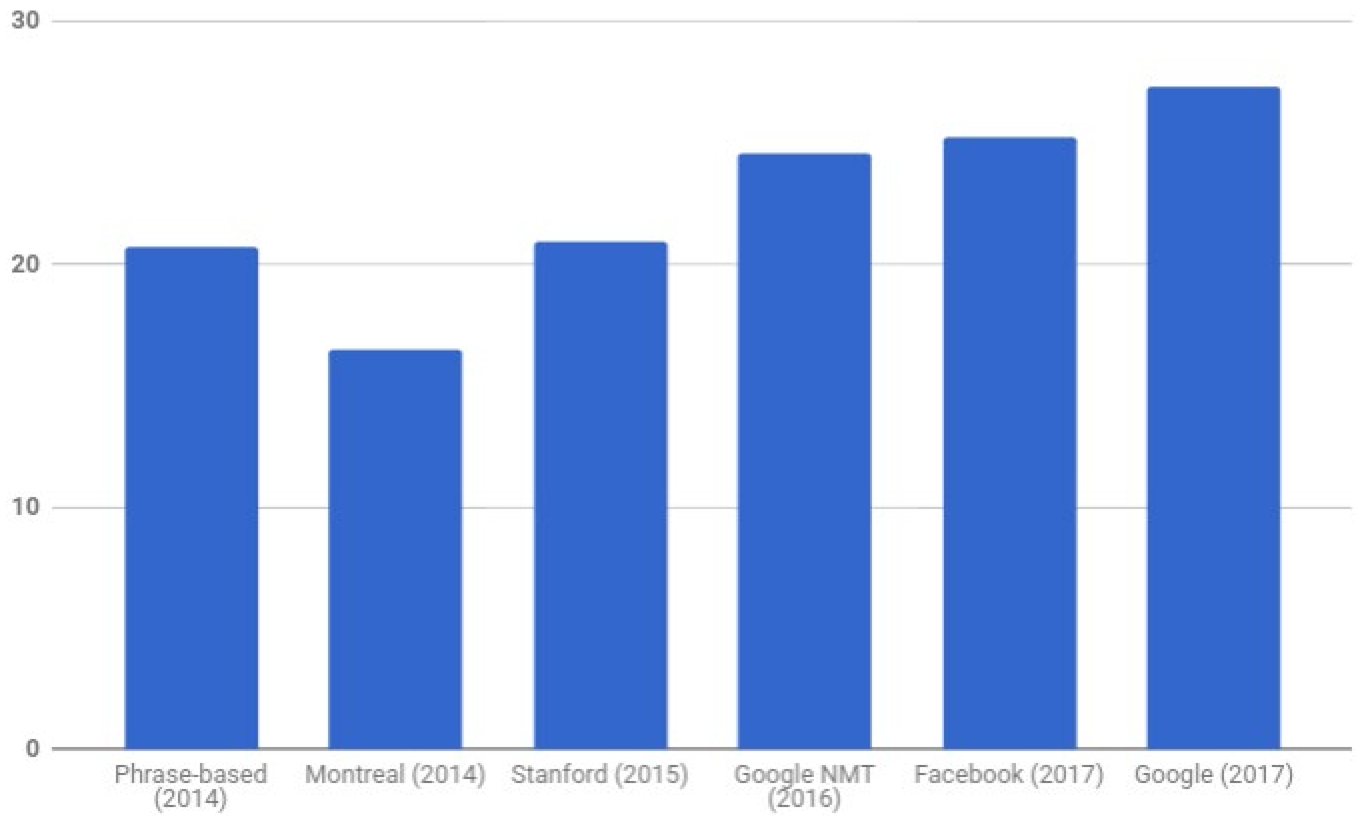
Neural Machine Translation (NMT)

- No pipelines. Single model trained end-to-end with backprop.
- Essentially a conditional language model

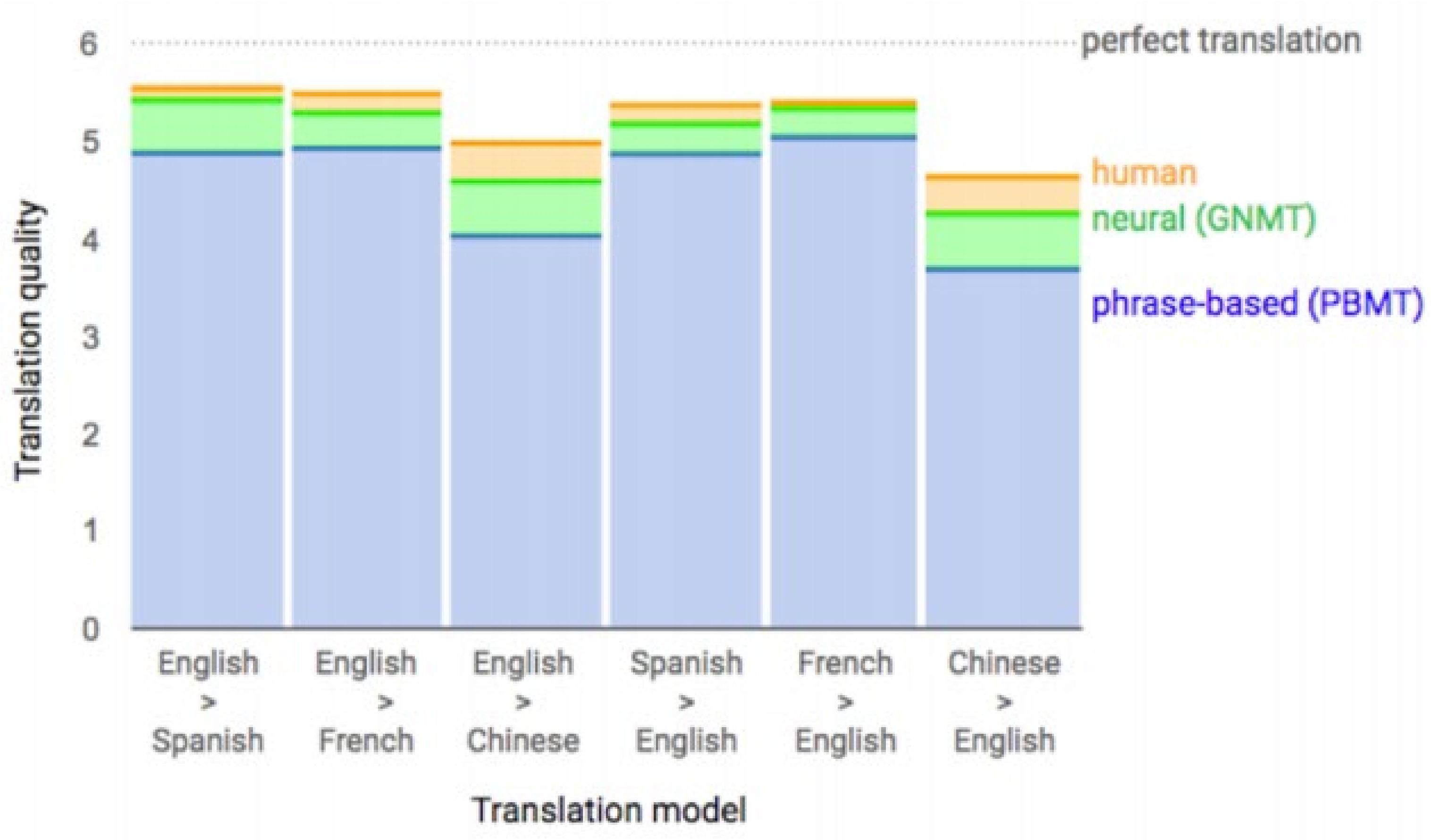


Machine Translation Progress

- English-German



Machines vs. Human



Sequence-to-sequence Modeling

Sequence-to-sequence Modeling

- data: <input sequence, output sequence> pairs
- use one network (**encoder**) to represent input sequence as a sequence of hidden vectors
- use another network (**decoder**) to produce the output sequence from the hidden vectors
- more generally called "**encoder-decoder**" models

Pure Encoder-Decoder Framework

Input (sentence, image, etc.)

⇓

Fixed-Size Encoder (MLP, RNN, CNN)

$\text{Encoder}(\text{input}) \in \mathbb{R}^D$

⇓

Decoder

$\text{Decoder}(\text{Encoder}(\text{input}))$

Seq2seq for NMT

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT-14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous state of the art. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

[Sutskever et al. (2014): Sequence to Sequence Learning with Neural Networks]

Seq2seq for NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

- $h_t = \mathrm{RNN}(x_t, h_{t-1})$ (Encoder RNN)

[Sutskever et al. (2014)]

Seq2seq for NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

- $h_t = \mathrm{RNN}(x_t, h_{t-1})$ (Encoder RNN)

- $h_T =$ Last hidden state of RNN encoder (summary of source)

[Sutskever et al. (2014)]

Seq2seq for NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

- $h_t = \mathrm{RNN}(x_t, h_{t-1})$ (Encoder RNN)
- $h_T =$ Last hidden state of RNN encoder (summary of source)
- $q_i = \mathrm{RNN}(y_l, q_{i-1})$ (Decoder RNN)

[Sutskever et al. (2014)]

Seq2seq for NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

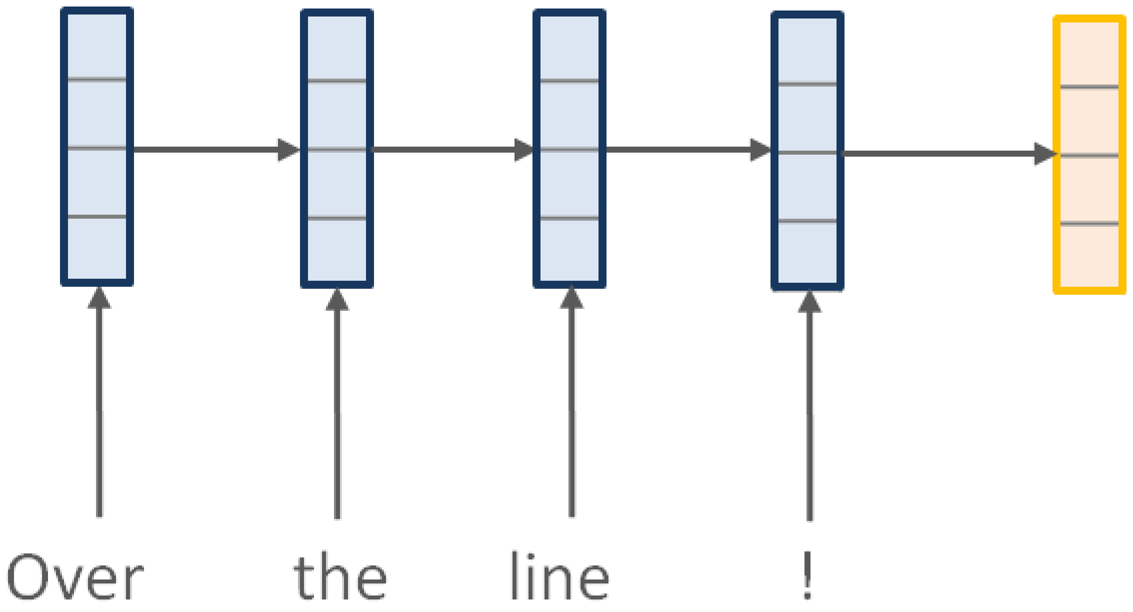
Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

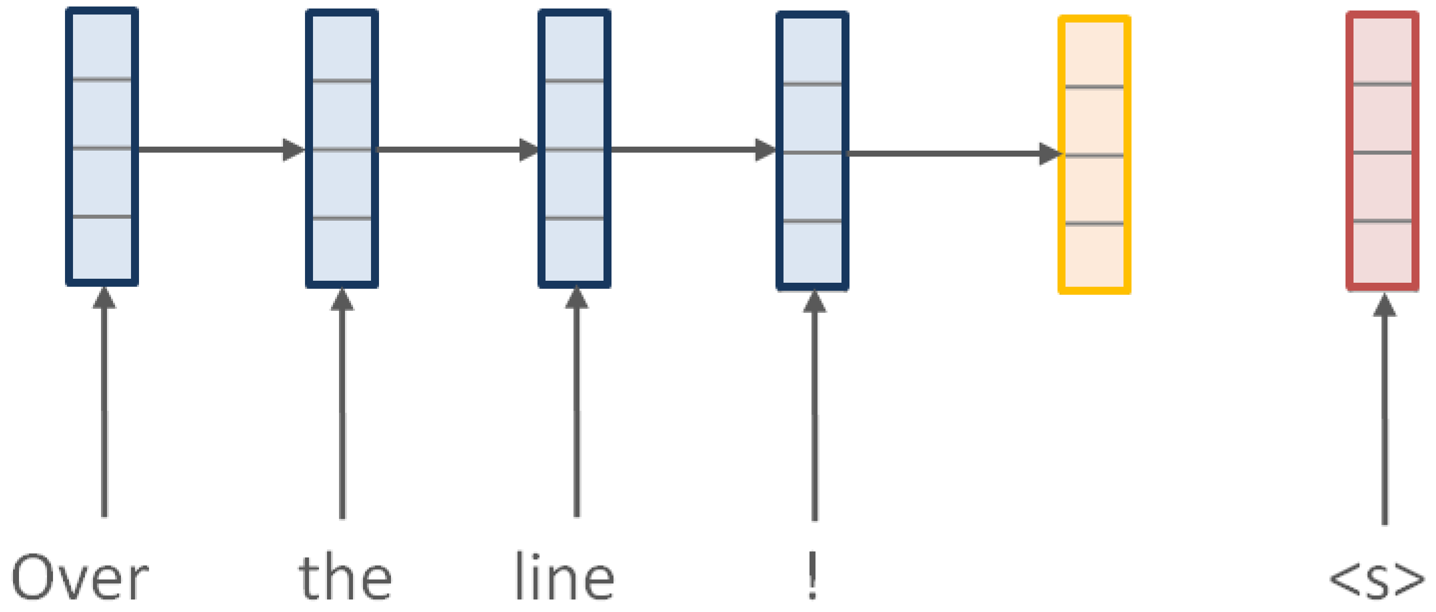
- $h_t = \mathrm{RNN}(x_t, h_{t-1})$ (Encoder RNN)
- $h_T =$ Last hidden state of RNN encoder (summary of source)
- $q_i = \mathrm{RNN}(y_l, q_{i-1})$ (Decoder RNN)
- $p(y_i | y_{<i}, \mathbf{x}) = \mathrm{softmax}(\mathrm{MLP}([q_i, h_T]))$
- Training: word-level maximum likelihood

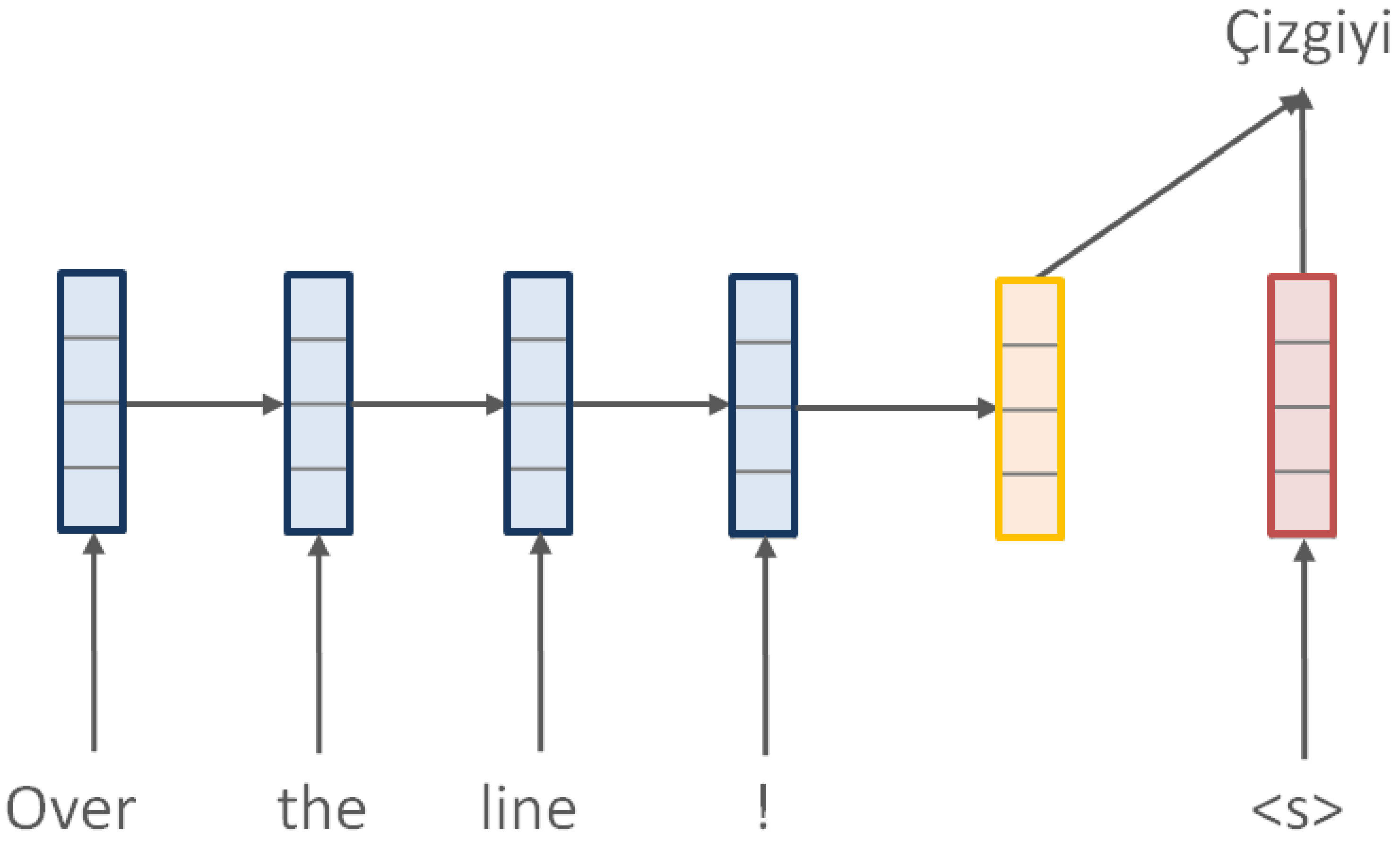
$$\arg\max_{\theta} \sum_{i=1}^{L} \log p(y_i | y_{<i}, \mathbf{x})$$

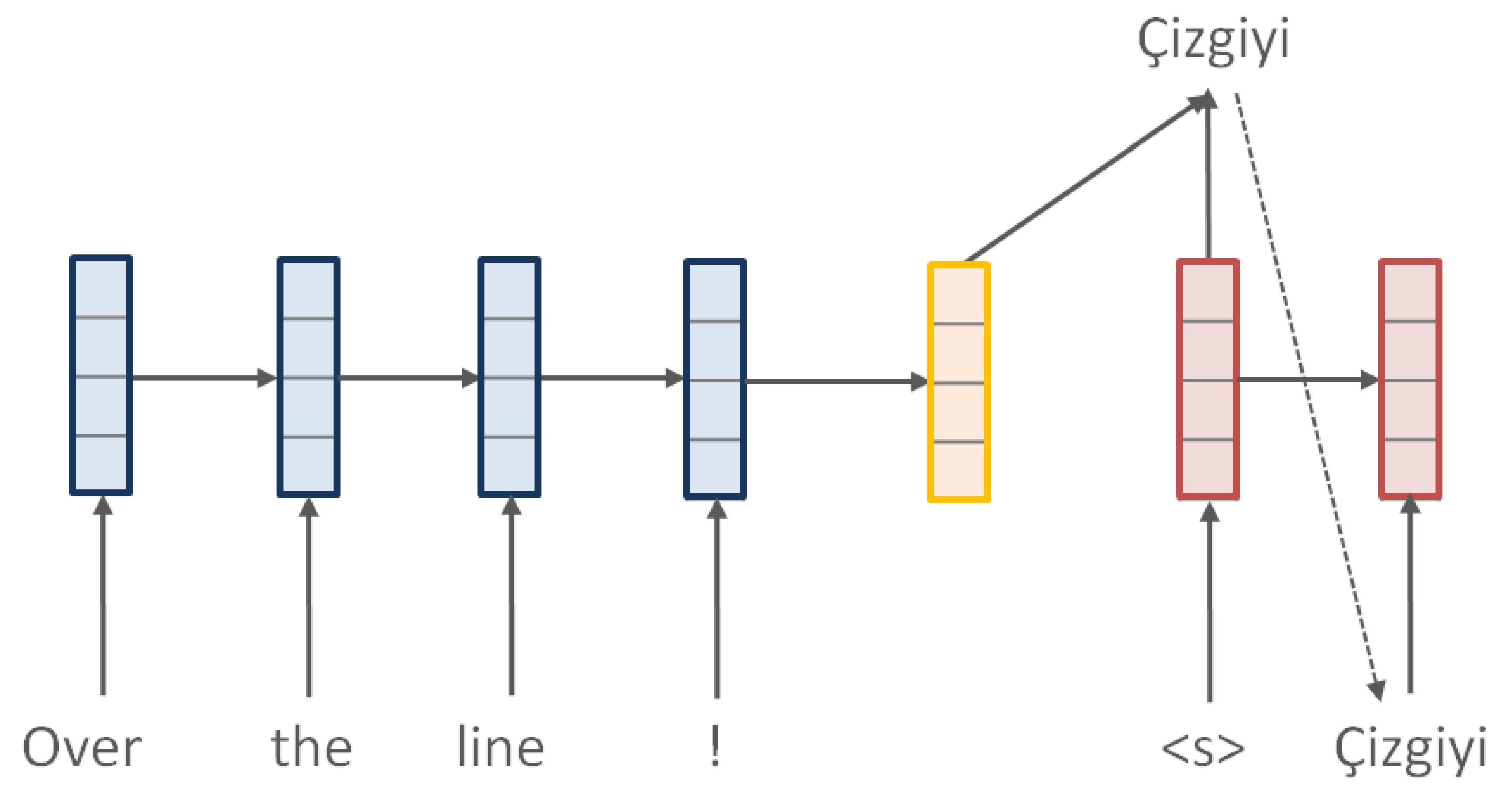
[Sutskever et al. (2014)]

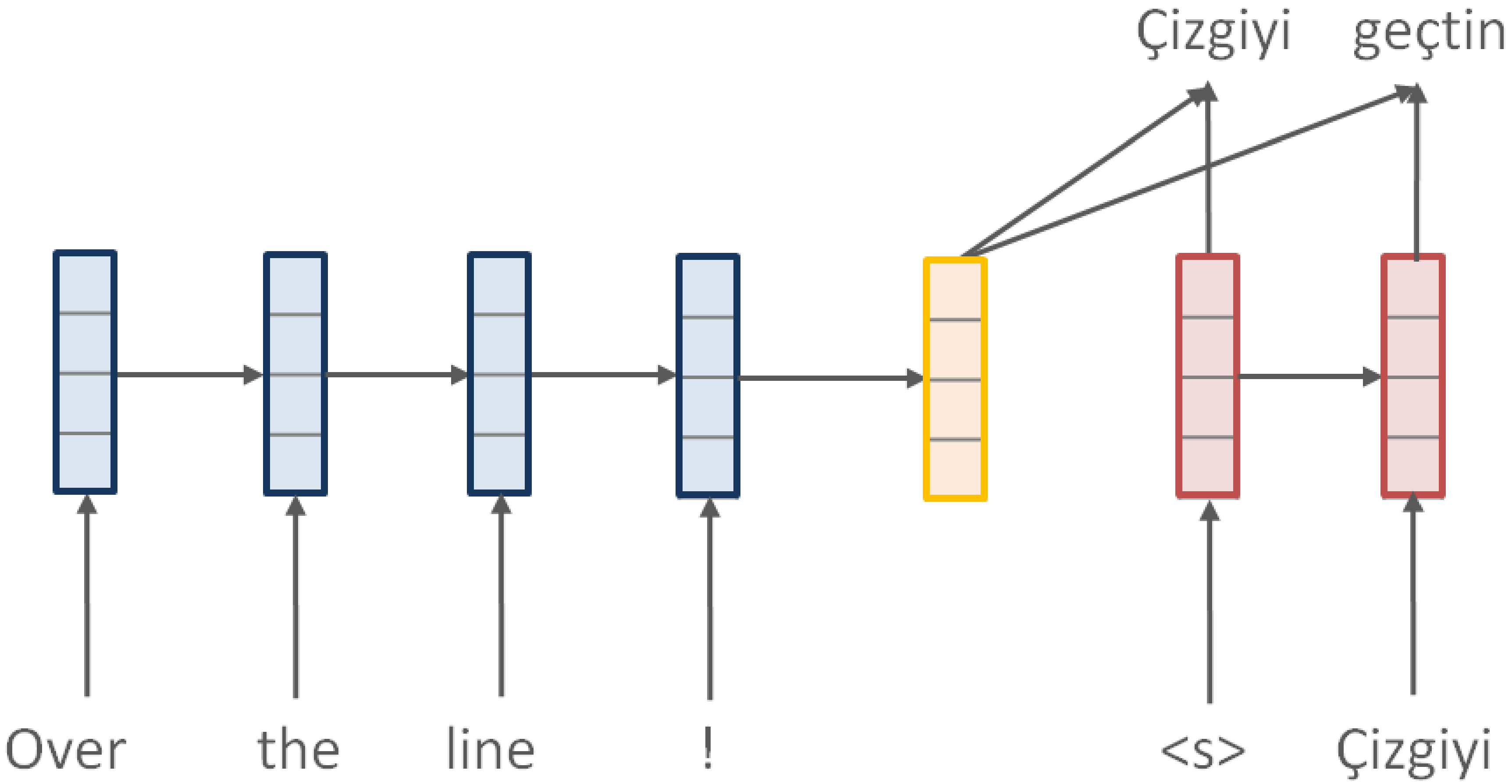
Over the line !


Over
the
line
!

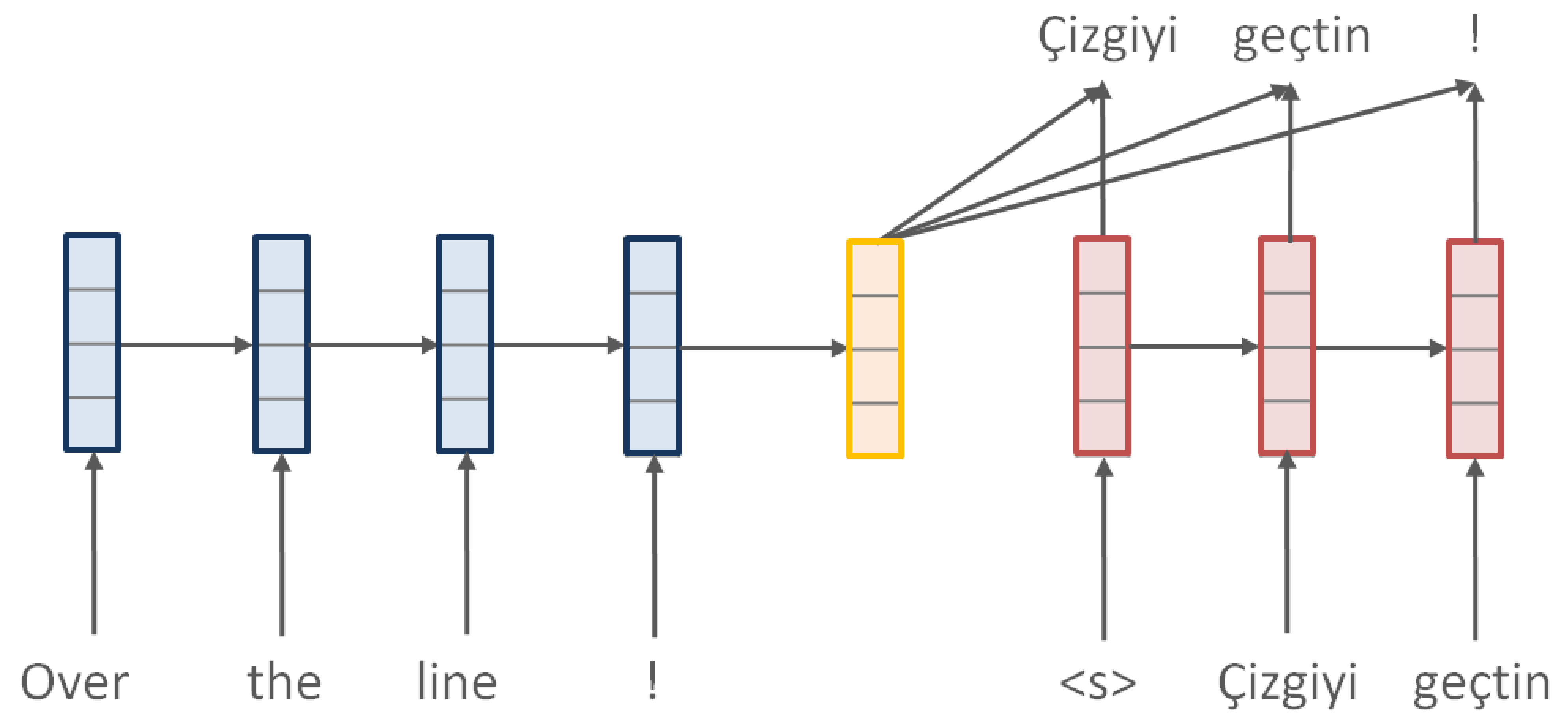

Over
the
line
!
<s>

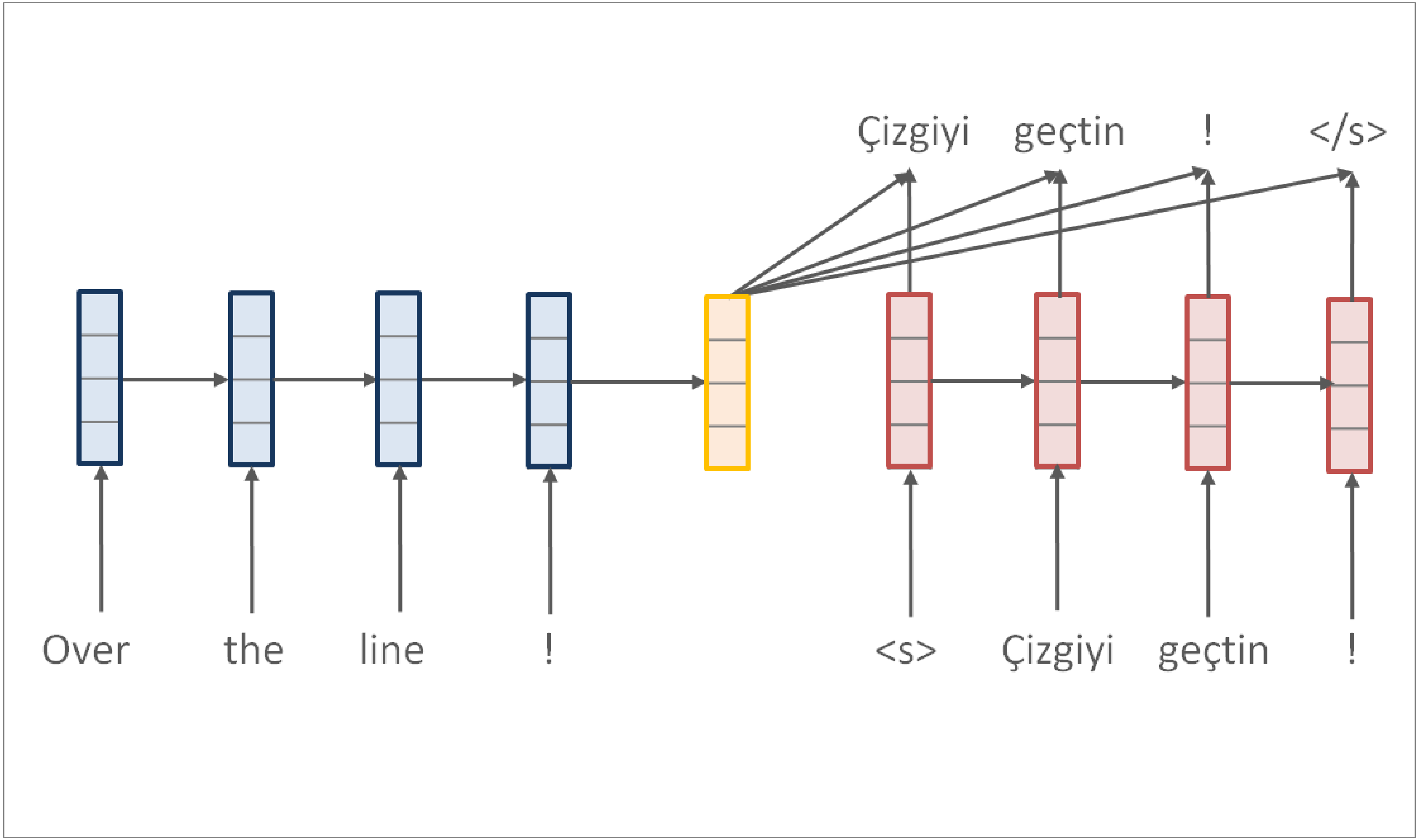

Çizgiyi
Over
the
line
!
<s>


Çizgiyi
Over
the
line
!
<s>
Çizgiyi



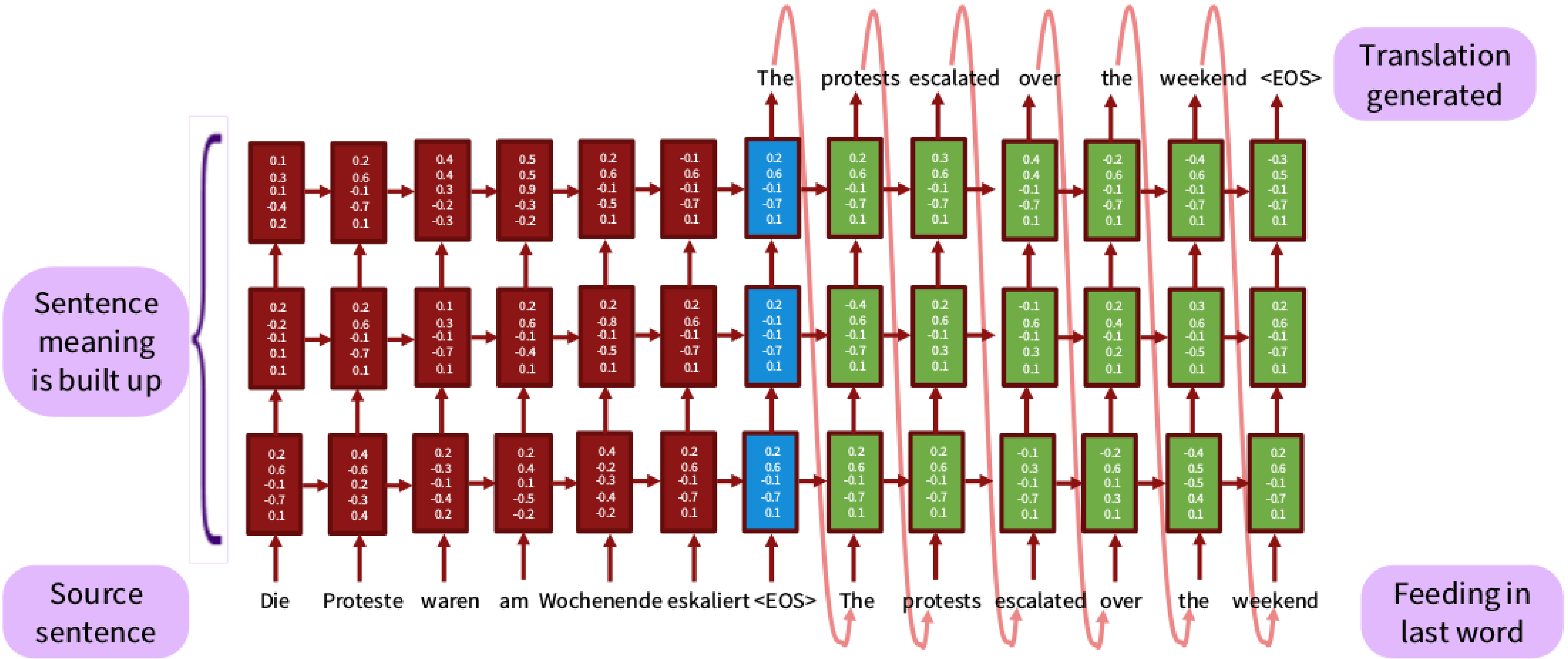
Çizgiyi
geçtin
Over
the
line
!
<s>
Çizgiyi


Çizgiyi
geçtin
!
Over
the
line
!
<s>
Çizgiyi
geçtin



Çizgiyi
geçtin
!
</s>
Over
the
line
!
<s>
Çizgiyi
geçtin
!

Seq2seq for NMT



[Sutskever et al. (2014)]

Communication Bottleneck

- All input information communicated through fixed-size hidden vector.

Encoder(input)

- Training: All gradients have to flow through single bottleneck.
- Test: All input encoded in single vector.

Neural Attention

Input (sentence, image, etc.)

Neural Attention

Input (sentence, image, etc.)

Memory-Bank Encoder (MLP, RNN, CNN)

$$\text{Encoder}(\text{input}) = x_1, x_2, \ldots, x_T$$

Neural Attention

Input (sentence, image, etc.)

↓

Memory-Bank Encoder (MLP, RNN, CNN)

Encoder(input) $= x_1, x_2, \ldots, x_T$

↓

Attention Distribution "where" | Annotation Function "what"

Neural Attention

Input (sentence, image, etc.)

↓

Memory-Bank Encoder (MLP, RNN, CNN)

Encoder(input) $= x_1, x_2, \ldots, x_T$

↓

Attention Distribution "where" | Annotation Function "what"

↓

Context Vector ("soft selection")

Neural Attention

Input (sentence, image, etc.)

Memory-Bank Encoder (MLP, RNN, CNN)

Encoder(input) $= x_1, x_2, \ldots, x_T$

Attention Distribution "where"

Annotation Function "what"

Context Vector ("soft selection")

Decoder

Neural Attention

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

[Bahdanau et al. (2015)]

Attention-based NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

[Bahdanau et al. (2015)]

Attention-based NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

- $[h_1, \ldots, h_T] = \mathrm{RNN}(\mathbf{x})$ (Memory/Annotation Function)

- $q_i = \mathrm{RNN}(y_l, q_{i-1})$ (Decoder RNN)

[Bahdanau et al. (2015)]

Attention-based NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

- $[h_1, \ldots, h_T] = \mathrm{RNN}(\mathbf{x})$ (Memory/Annotation Function)
- $q_i = \mathrm{RNN}(y_l, q_{i-1})$ (Decoder RNN)
- $\alpha_{i,t} = \frac{\exp(q_i^\top h_t)}{\sum_{j=1}^{T} \exp(q_i^\top h_j)}$ (Attention distribution)

[Bahdanau et al. (2015)]

Attention-based NMT

Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

- $[h_1, \ldots, h_T] = \mathrm{RNN}(\mathbf{x})$ (Memory/Annotation Function)
- $q_i = \mathrm{RNN}(y_l, q_{i-1})$ (Decoder RNN)
- $\alpha_{i,t} = \frac{\exp(q_i^\top h_t)}{\sum_{j=1}^{T} \exp(q_i^\top h_j)}$ (Attention distribution)
- $c_i = \sum_{t=1}^{T} \alpha_{i,t} h_t$ (Context vector)

[Bahdanau et al. (2015)]

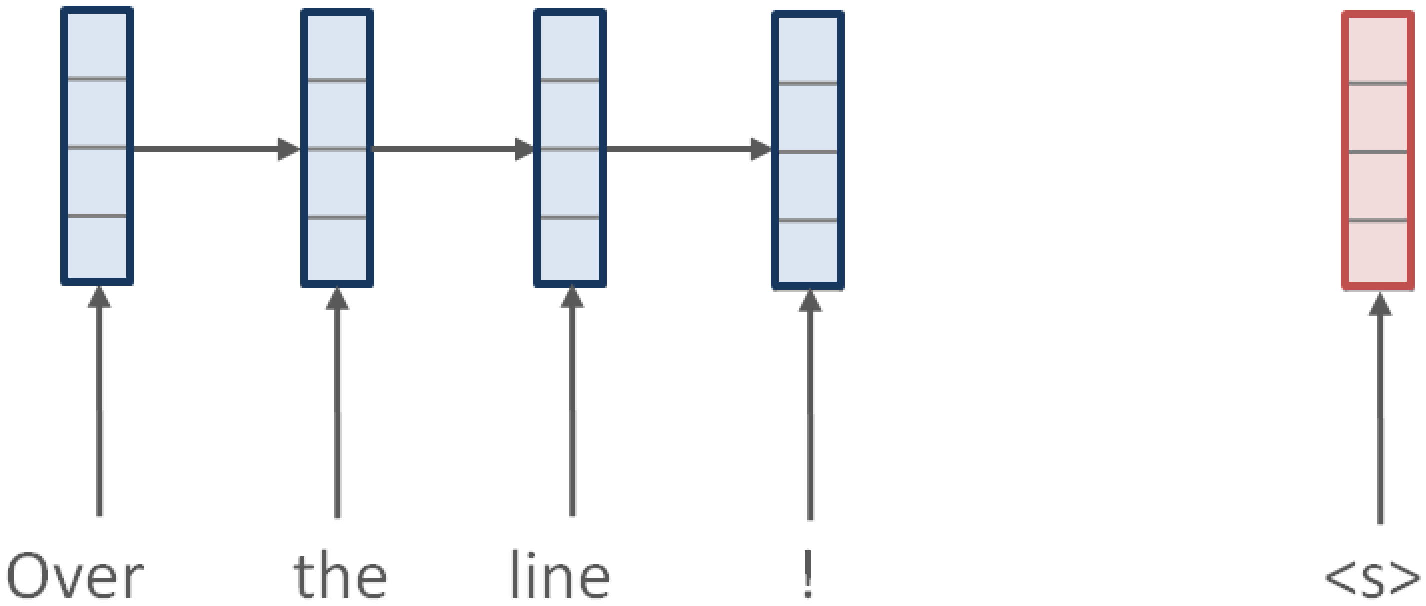
Attention-based NMT

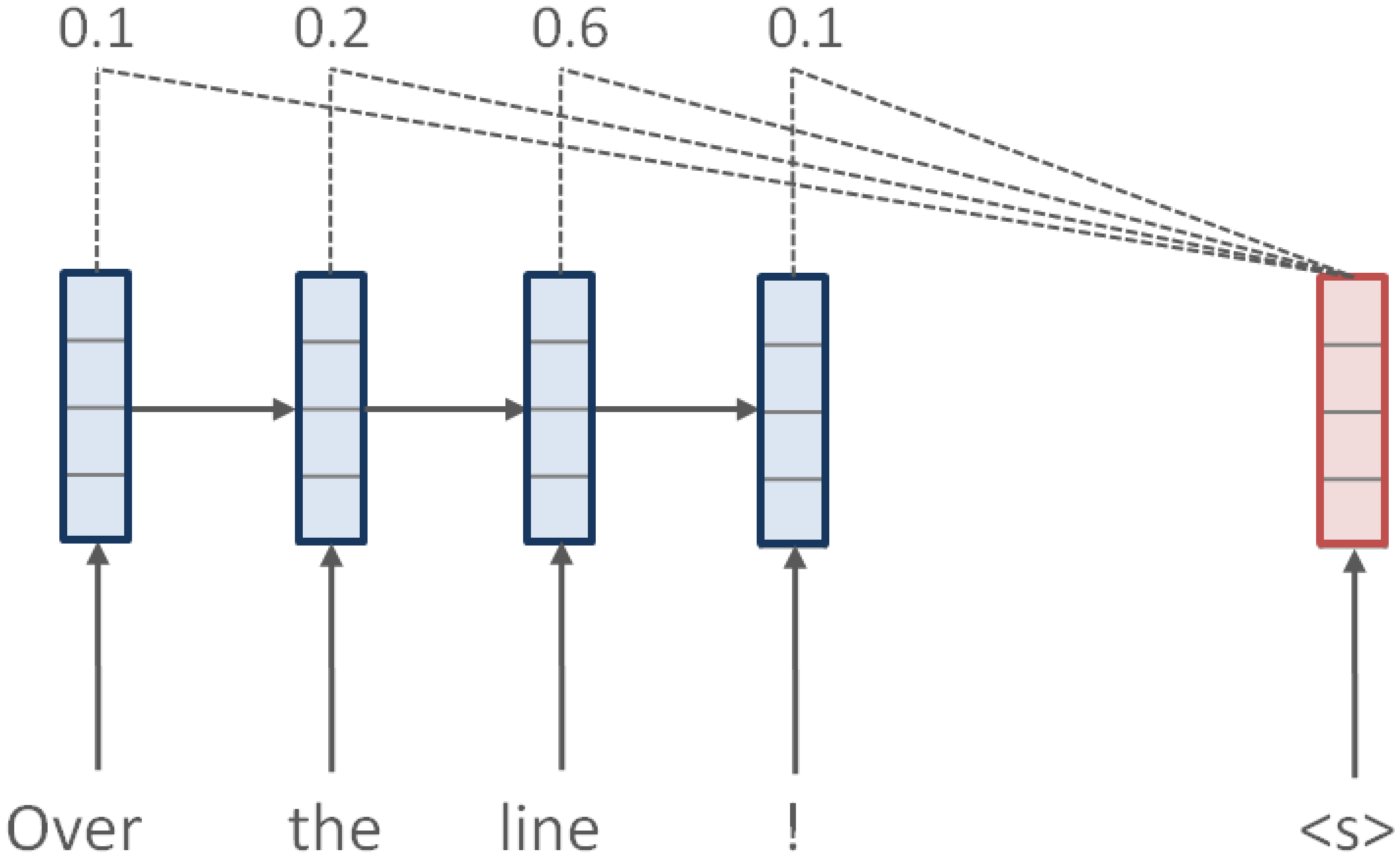
Source sentence: $\mathbf{x} = [x_1, \ldots, x_T]$

Target sentence: $\mathbf{y} = [y_1, \ldots, y_L]$

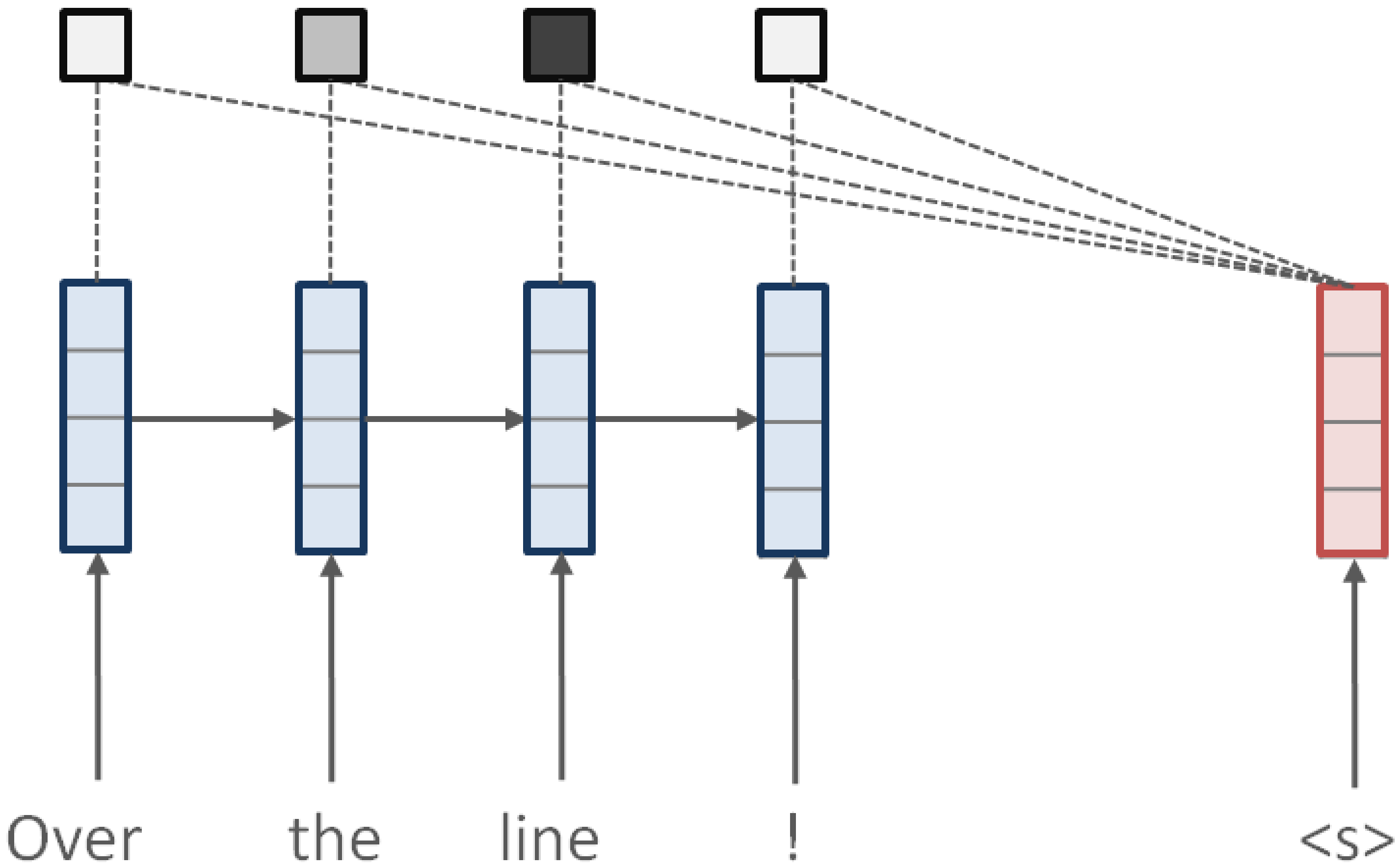
- $[h_1, \ldots, h_T] = \mathrm{RNN}(\mathbf{x})$ (Memory/Annotation Function)
- $q_i = \mathrm{RNN}(y_l, q_{i-1})$ (Decoder RNN)
- $\alpha_{i,t} = \frac{\exp(q_i^\top h_t)}{\sum_{j=1}^{T} \exp(q_i^\top h_j)}$ (Attention distribution)
- $c_i = \sum_{t=1}^{T} \alpha_{i,t} h_t$ (Context vector)
- $p(y_i | y_{<i}, \mathbf{x}) = \mathrm{softmax}(\mathrm{MLP}([q_i, c_i]))$

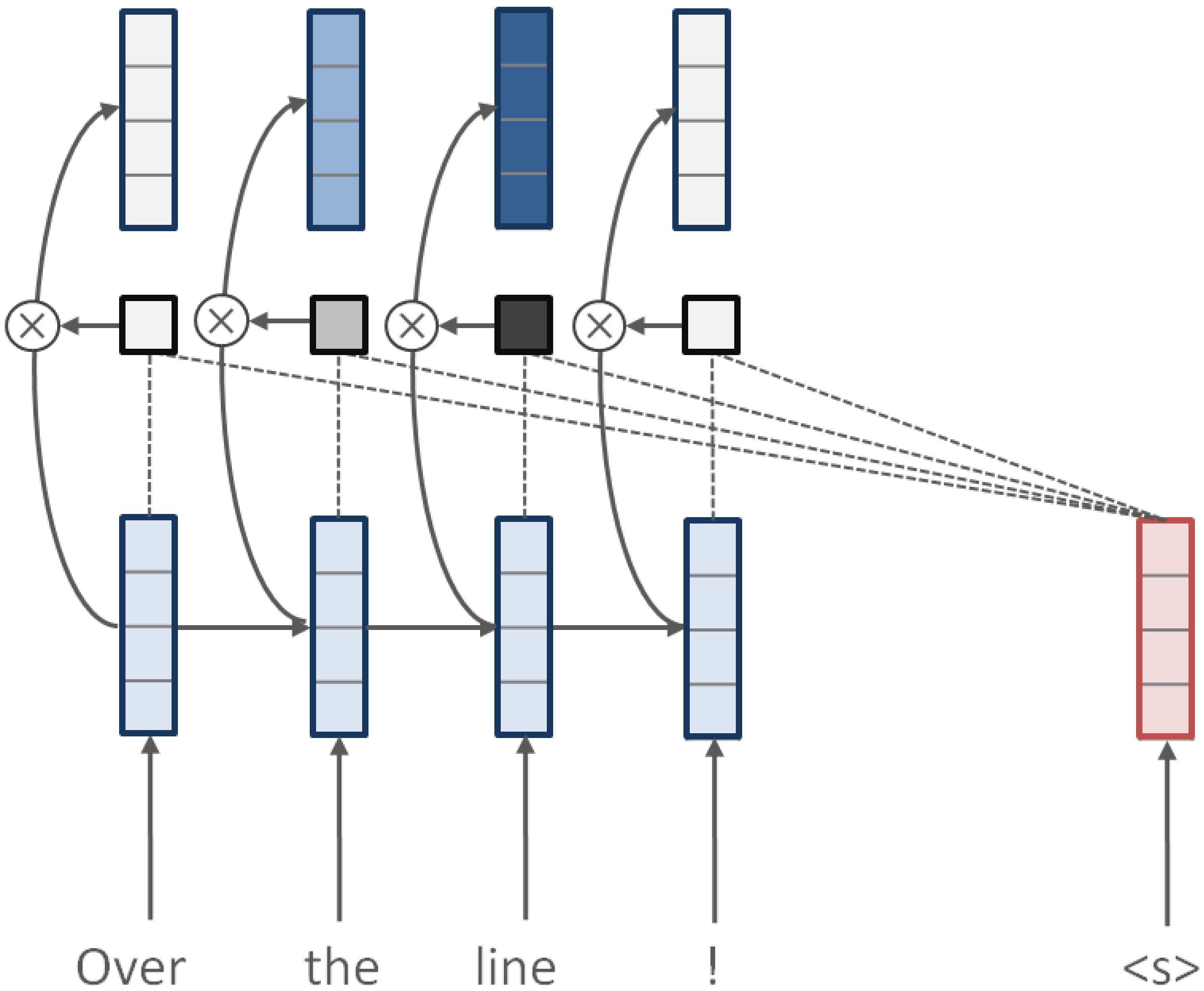
[Bahdanau et al. (2015)]

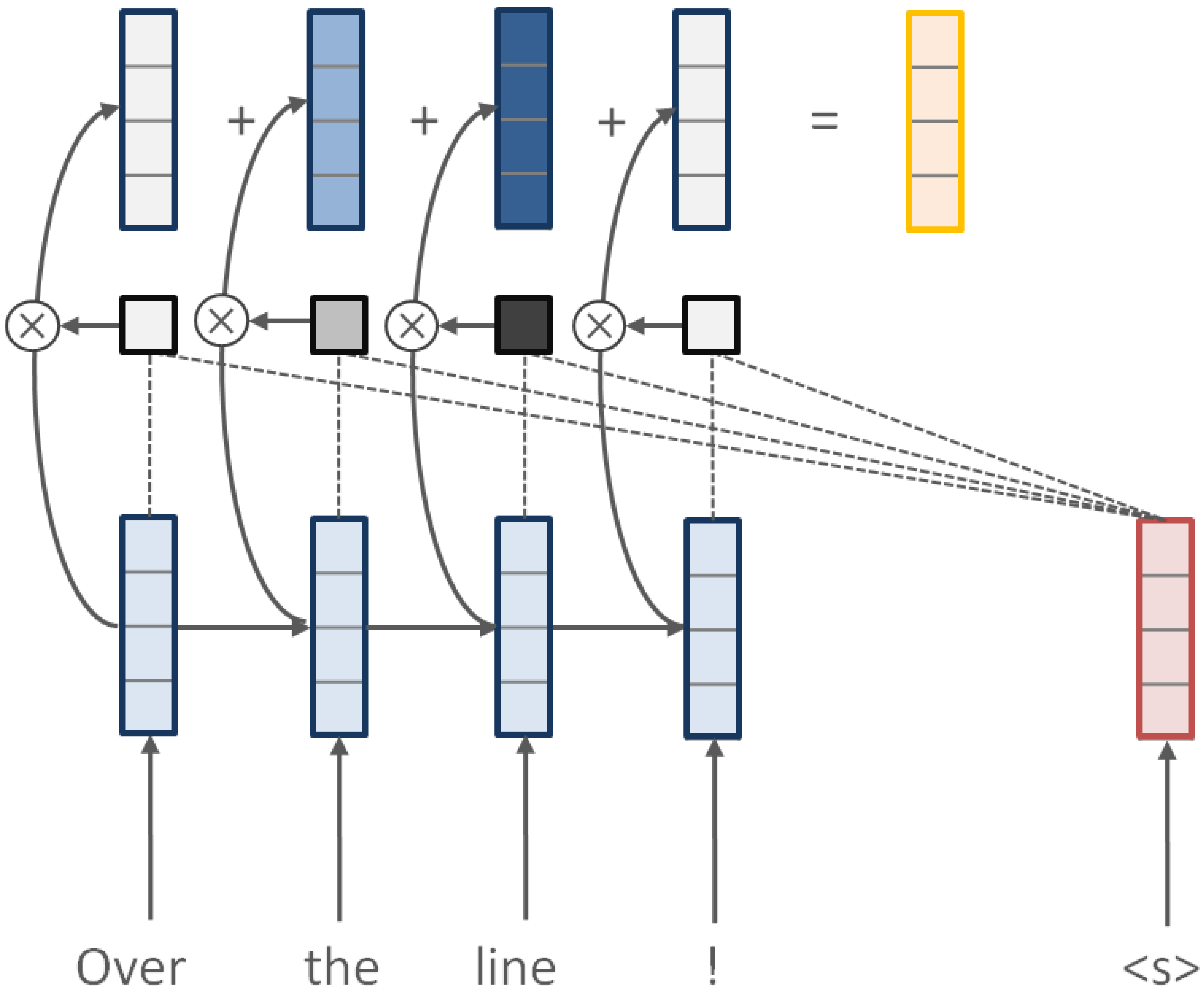

Over
the
line
!
<s>



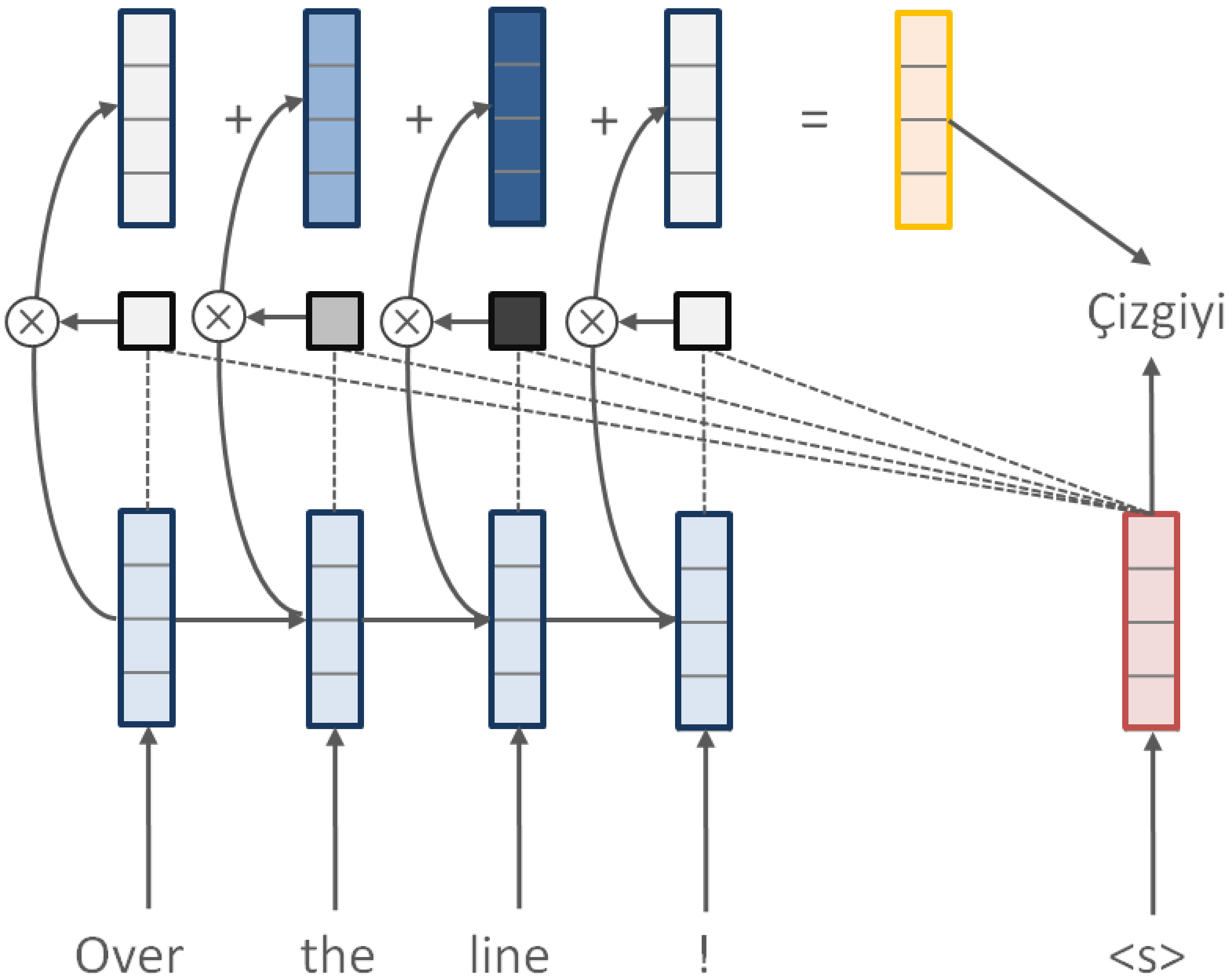
0.1
0.2
0.6
0.1
Over
the
line
!
<s>


Over
the
line
!
<s>

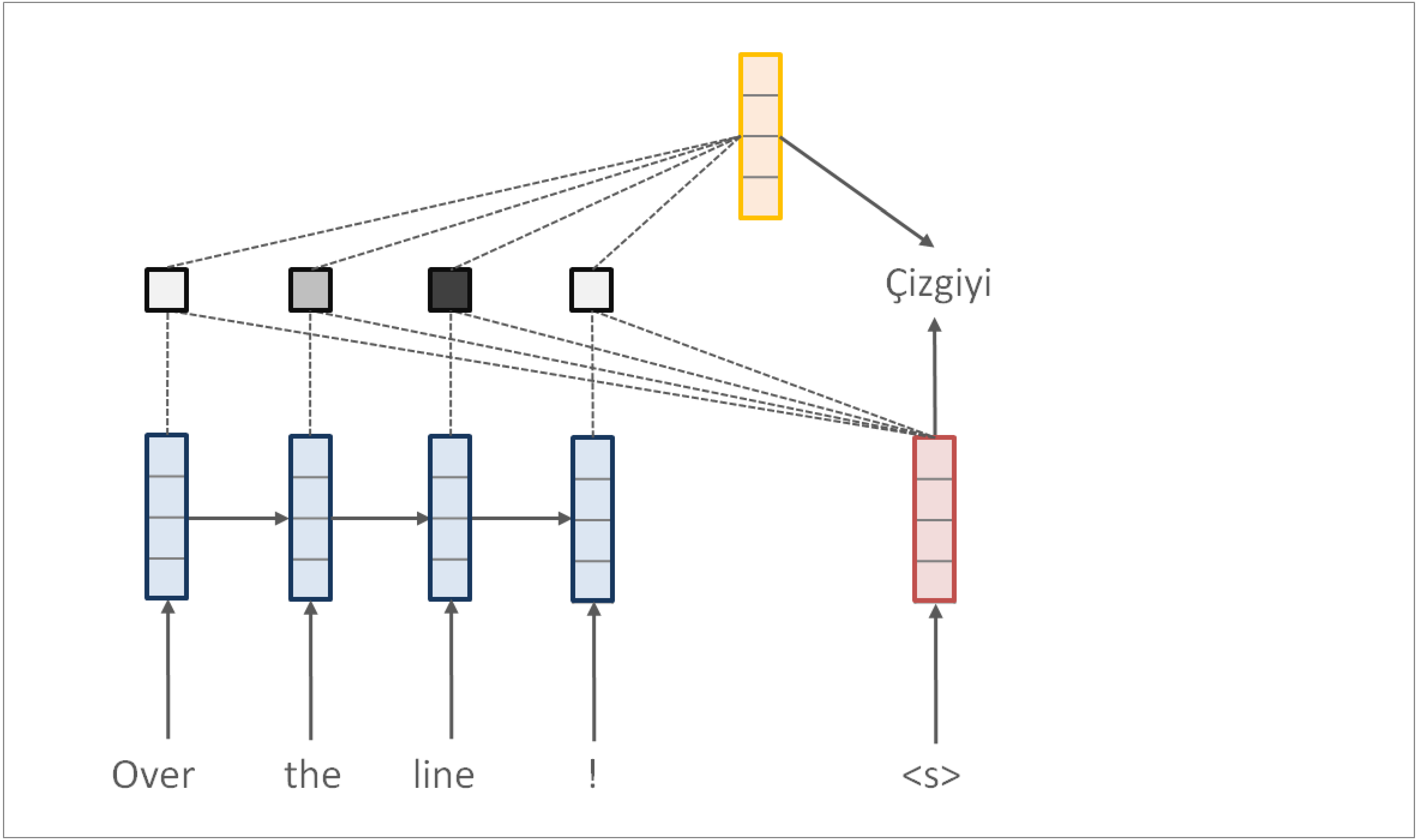

Over
the
line
!
<s>

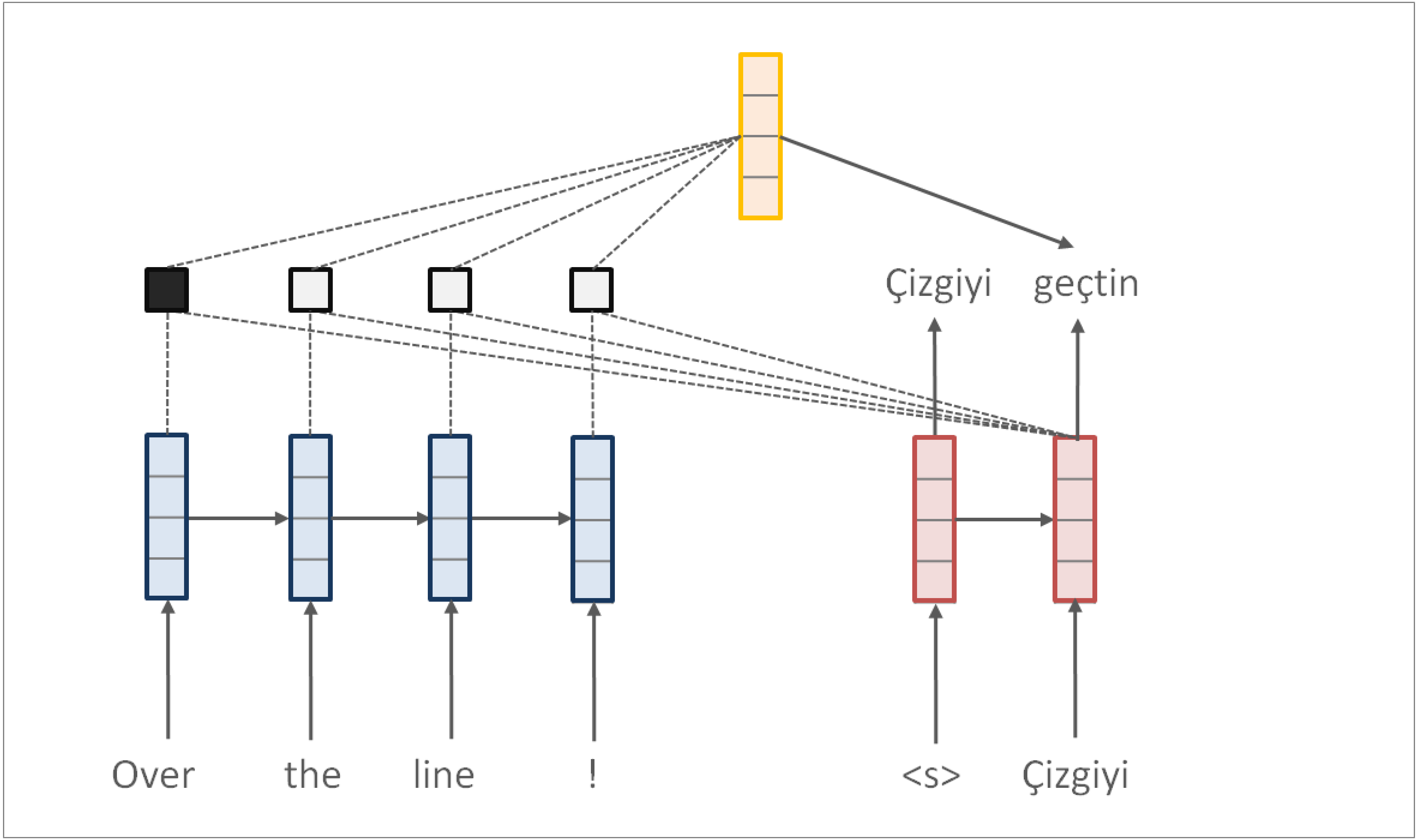


+
+
+
=
Over
the
line
!
<s>

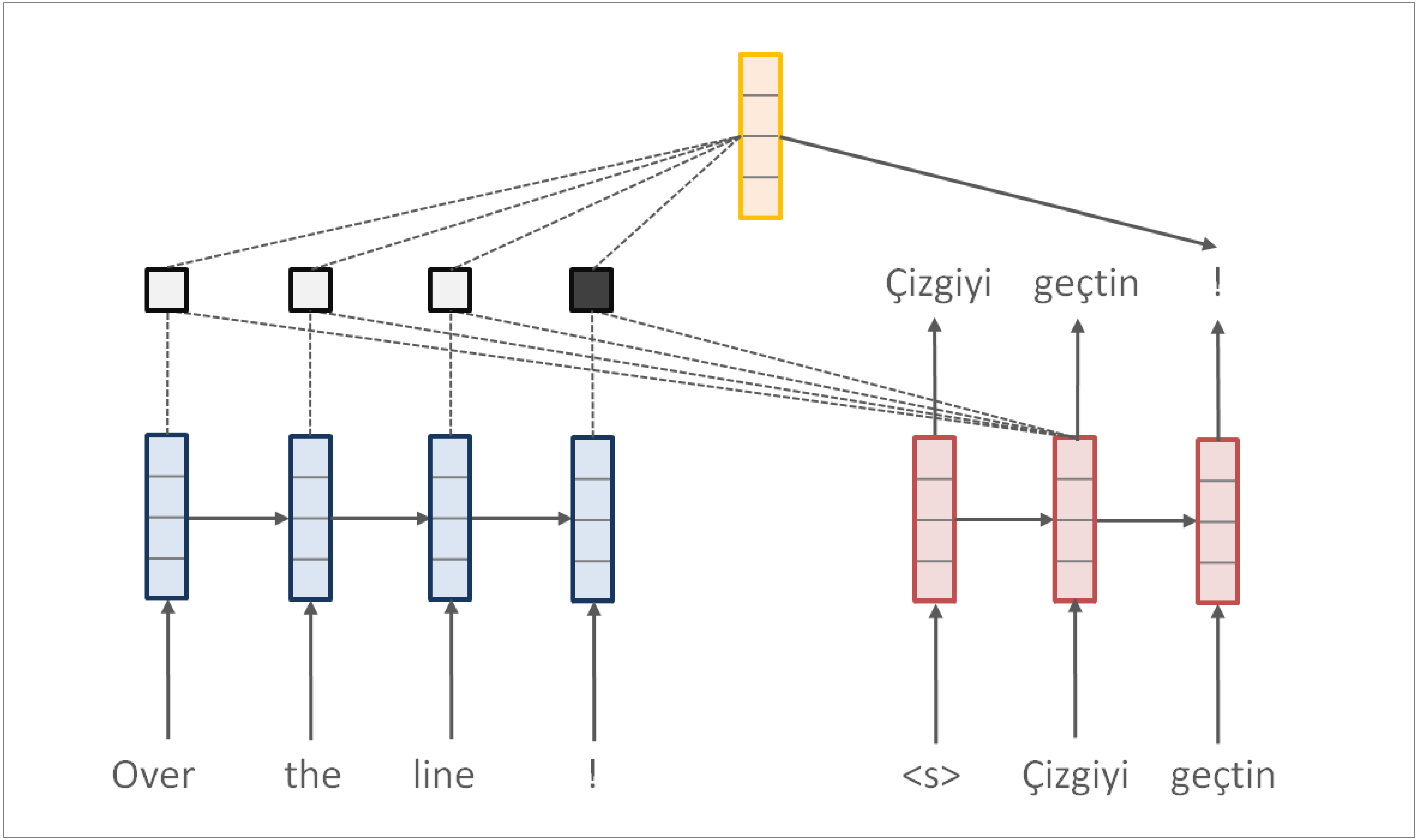


Çizgiyi
Over
the
line
!
<s>

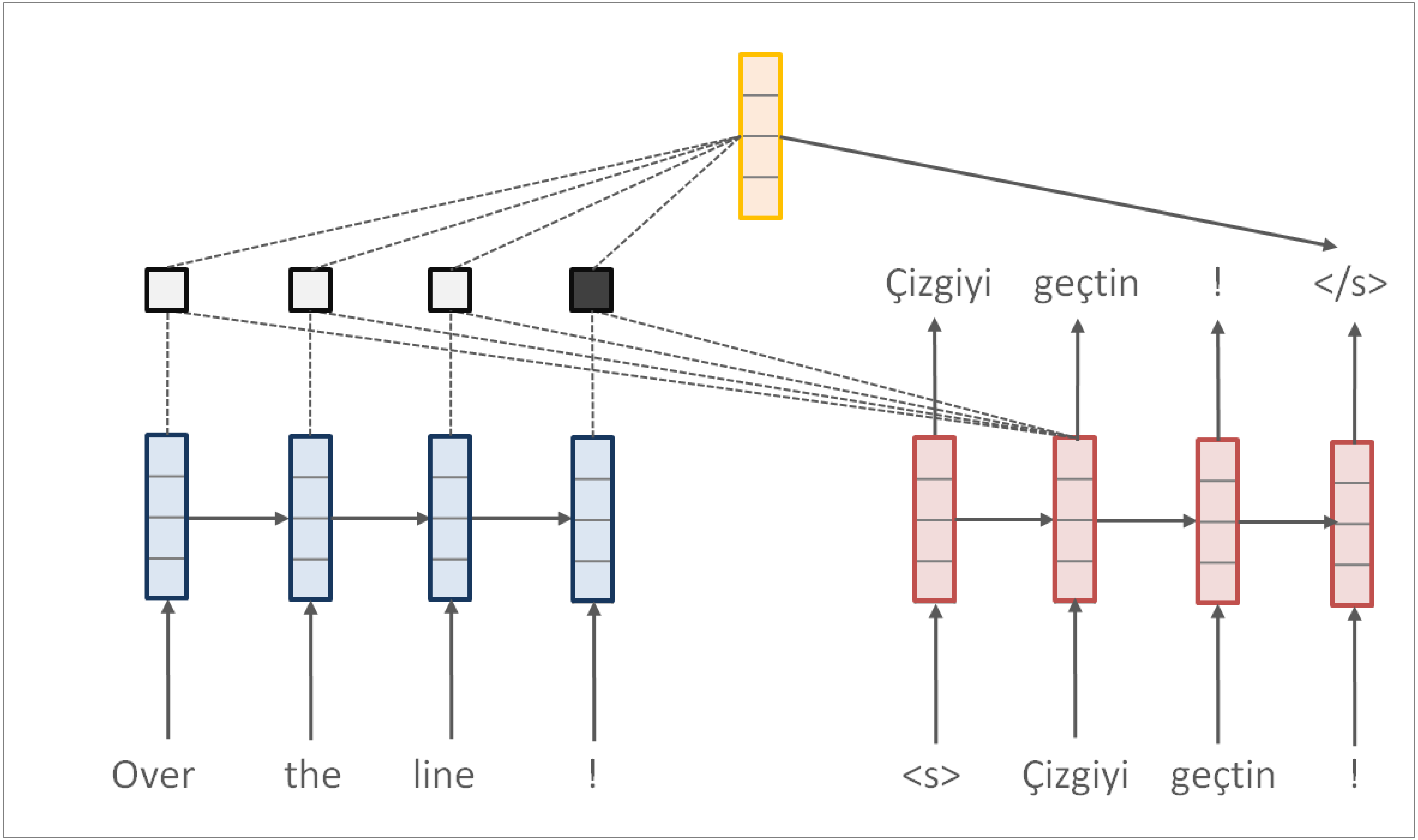

Çizgiyi
Over
the
line
!
<s>



Çizgiyi
geçtin
Over
the
line
!
<s>
Çizgiyi

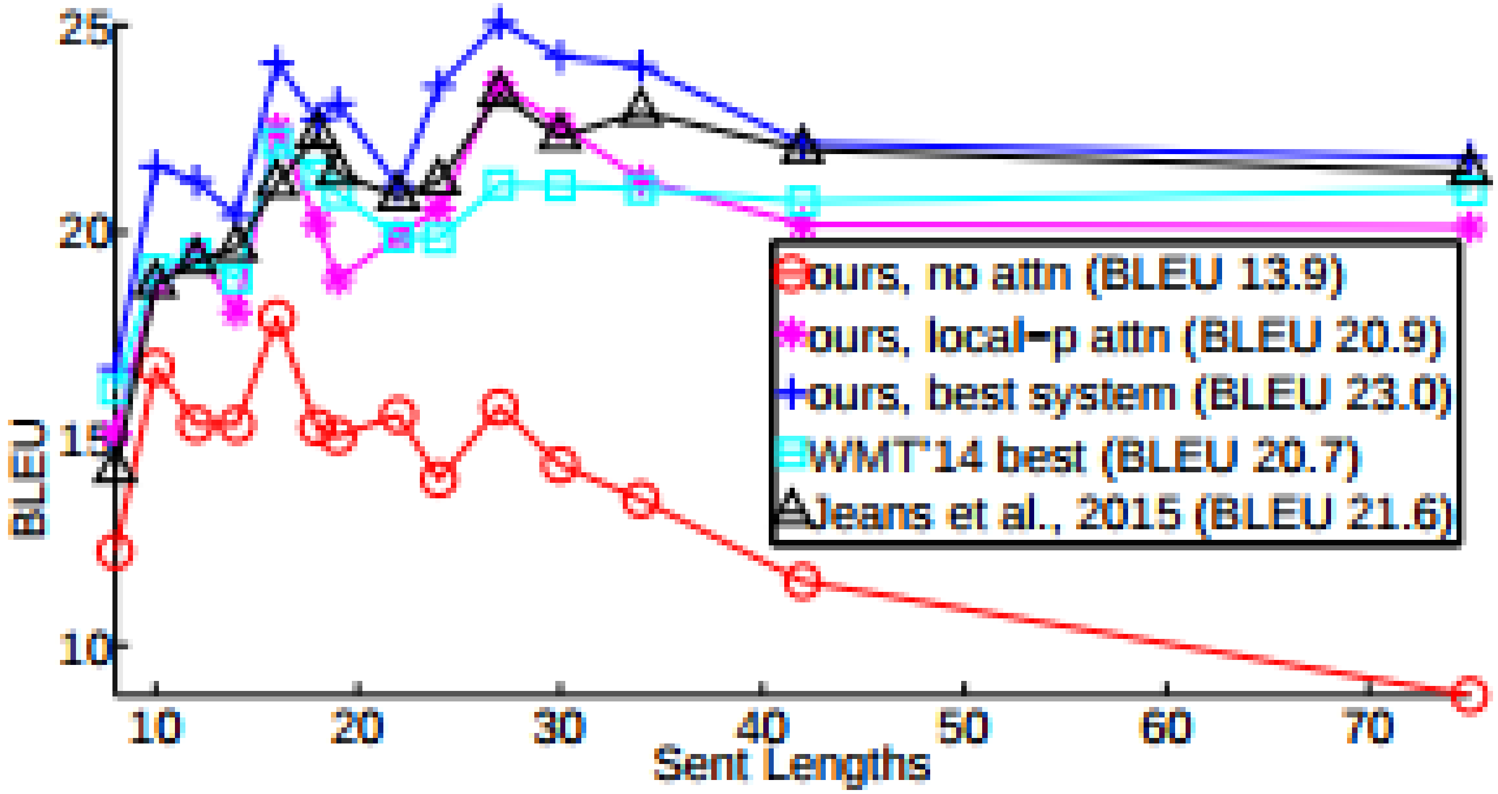


Çizgiyi
geçtin
!
Over
the
line
!
<s>
Çizgiyi
geçtin

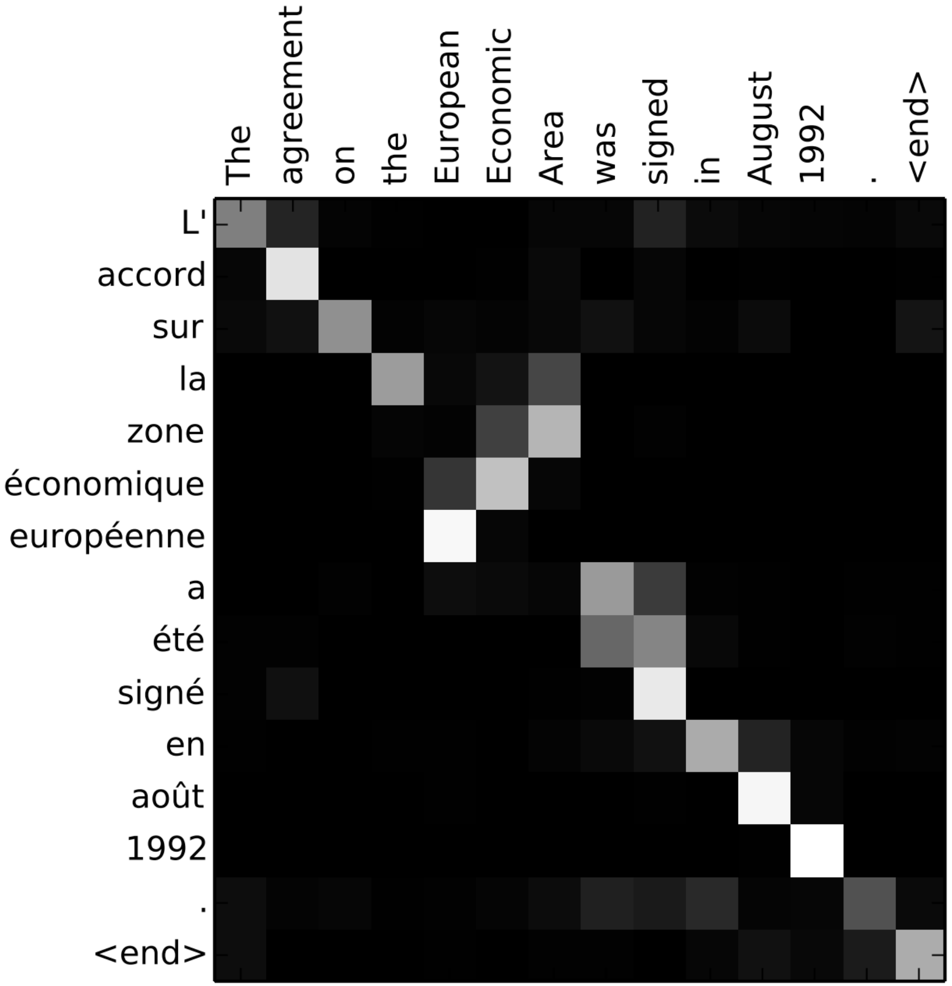


Çizgiyi
geçtin
!
</s>
Over
the
line
!
<s>
Çizgiyi
geçtin
!

Performance vs. Length

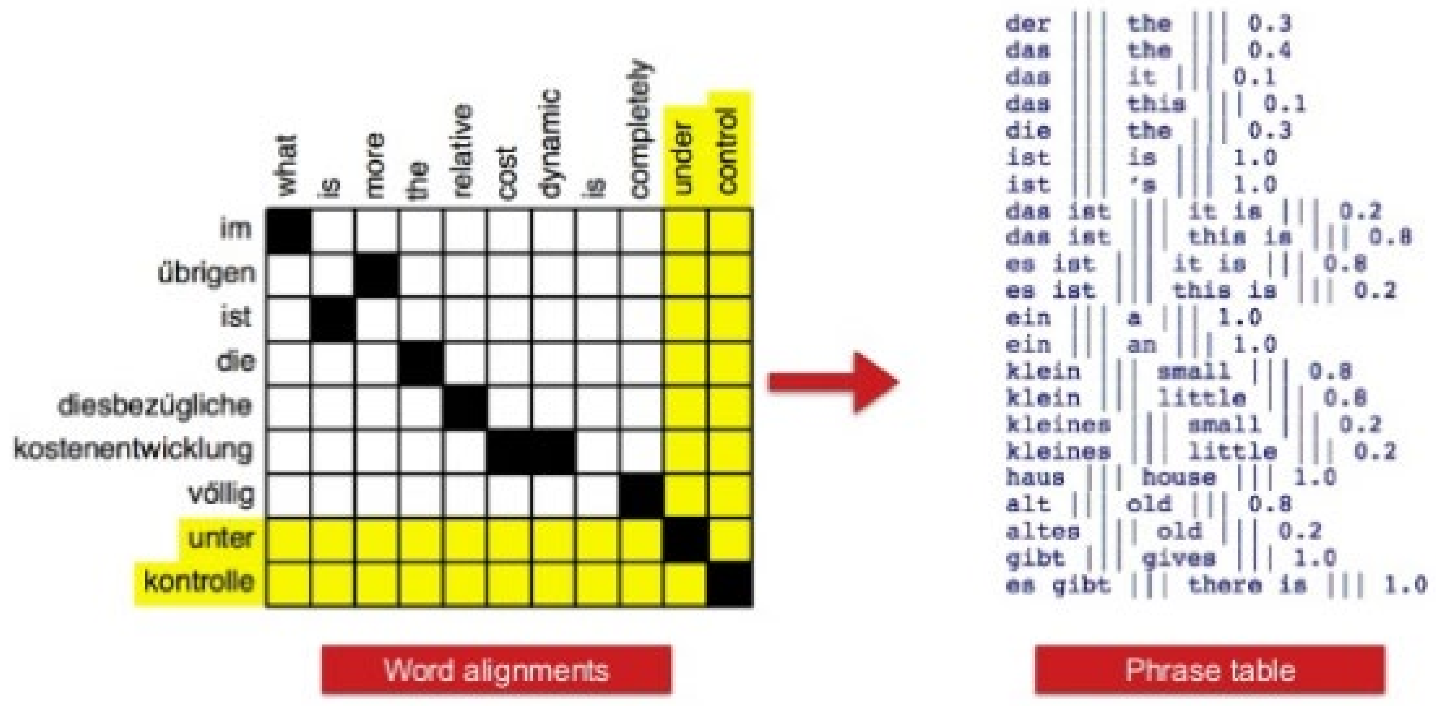


Attention Visualization



[Bahdanau et al. 2015]

Attention Model as a “Hidden Layer”



Attention Applications

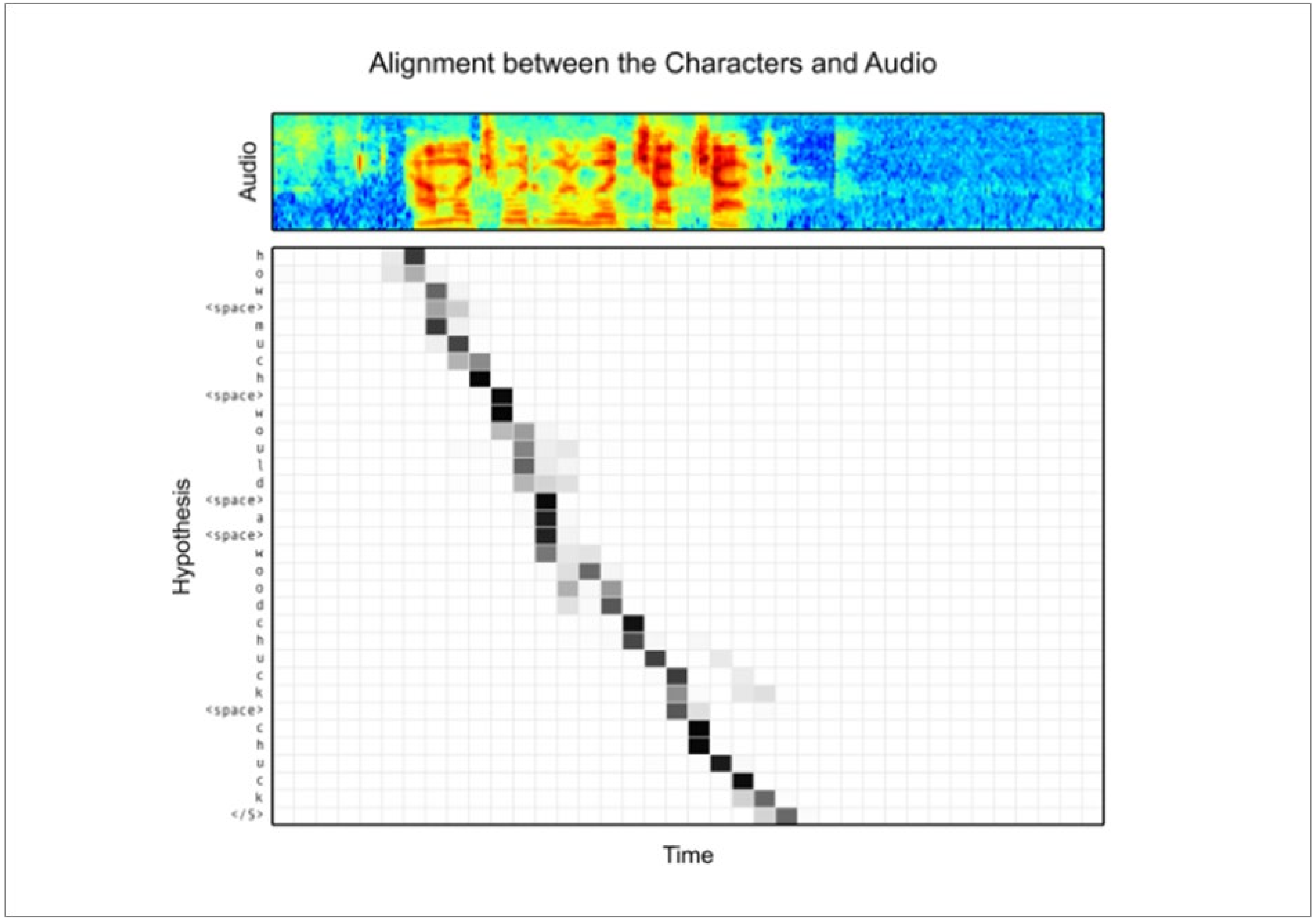
Attention Applications

- Machine Translation (Bahdanau et al., 2015; Luong et al., 2015)
- Question Answering (Hermann et al., 2015; Sukhbaatar et al., 2015)
- Natural Language Inference (Rockẗaschel et al., 2016; Parikh et al., 2016)
- Algorithm Learning (Graves et al., 2014, 2016; Vinyals et al., 2015a)
- Parsing (Vinyals et al., 2015b)
- Speech Recognition (Chorowski et al., 2015; Chan et al., 2015)
- Summarization (Rush et al., 2015)
- Caption Generation (Xu et al., 2015)
- and more...

Image Captioning (Xu et al., 2015)

(b) A woman is throwing a frisbee in a park.

Speech Recognition (Chan et al., 2015)



Summarization (Rush et al., 2015)

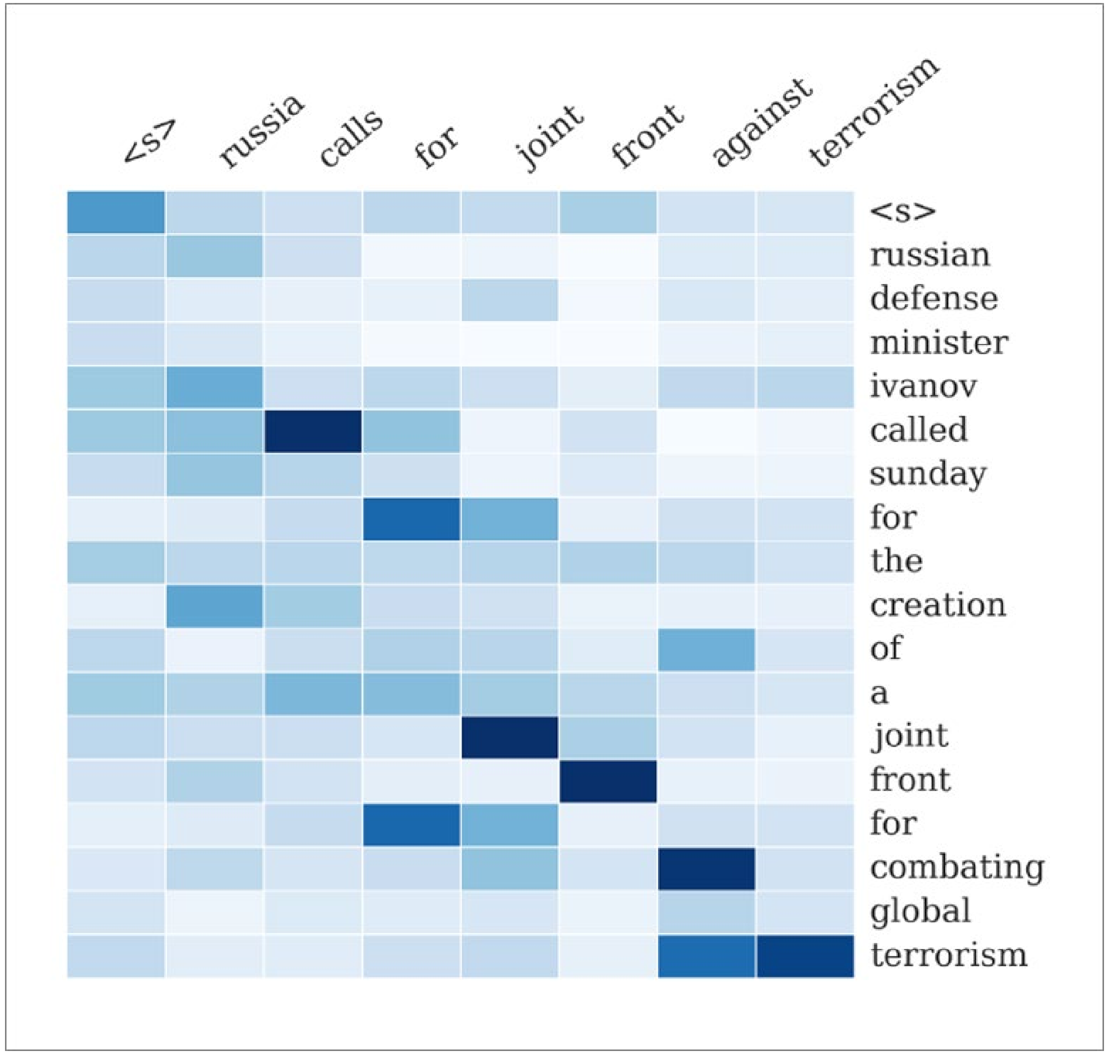
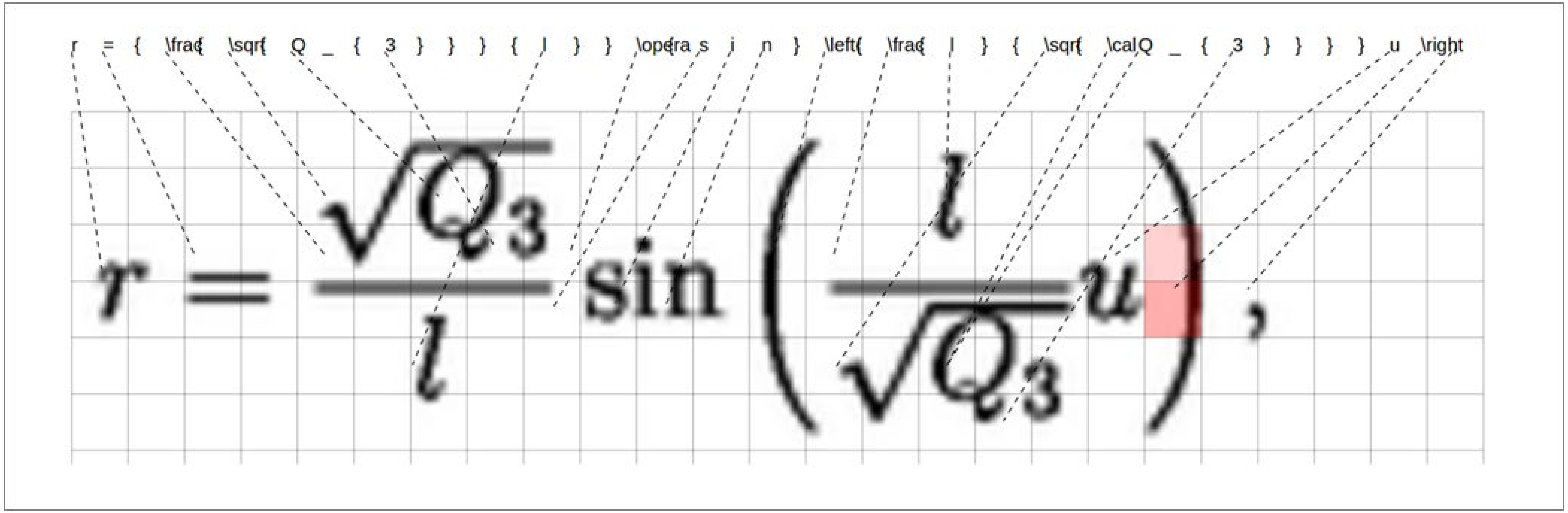
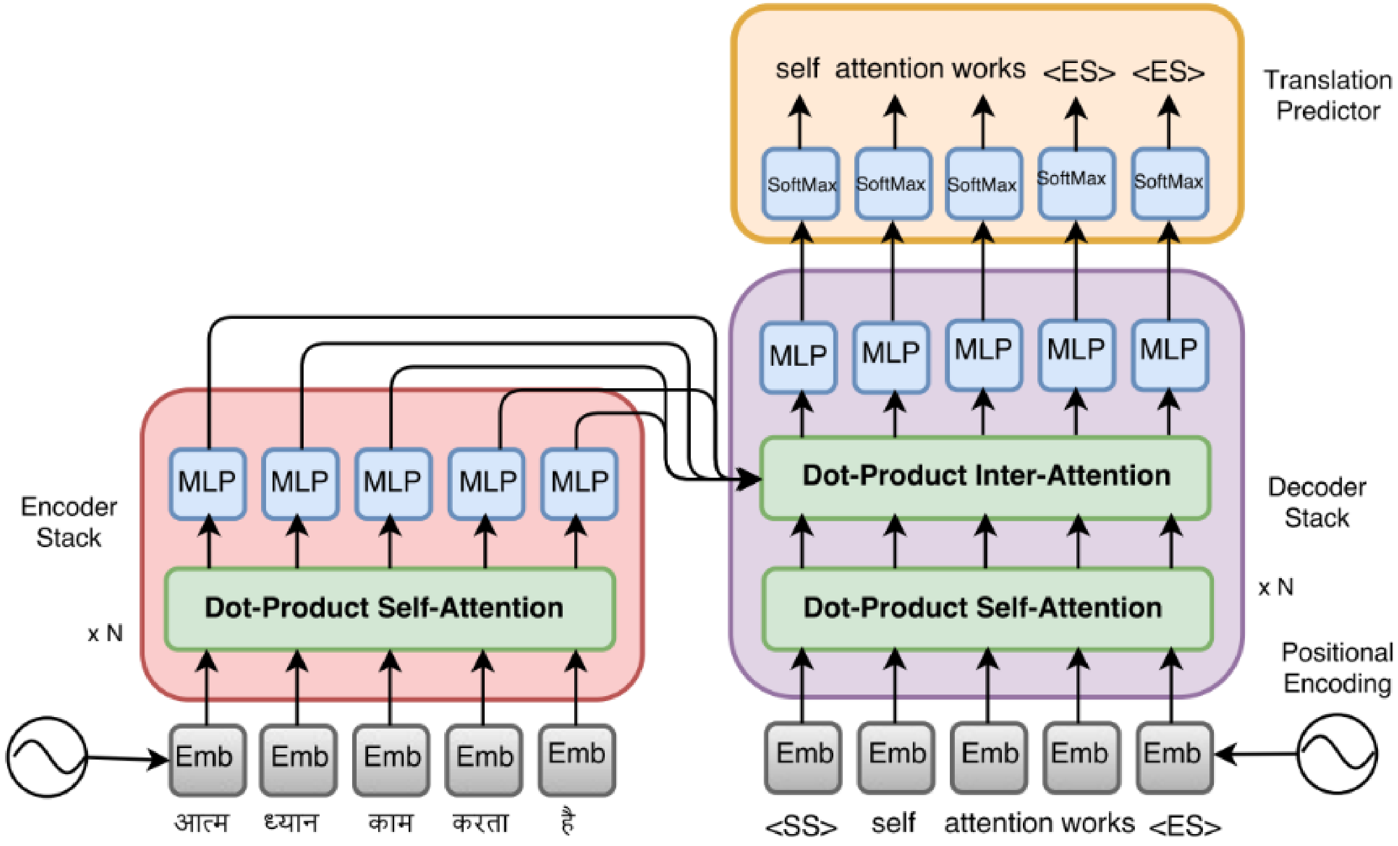


Image-to-Latex (Deng et al., 2016)



Attention → Transformers



[Vaswani et al. 2017]

Practice

- Some practical implementations to think about in more details
  - How to set up data for batch training? (source/target sentences have varying lengths)
  - What type of encoder/decoder architecture (GRU/LSTM/CNN)?
  - How to find $\arg\max_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}|\mathbf{x})$?
  - How to deal with unknown tokens?

Summary

- **Neural language models**
  - Feed-forward models
    - Classifier on next word prediction
    - Concatenate past word representations as features
    - Resolved data sparsity issues; learned dense parameters
  - RNN models
    - Model long history
    - Extends feed-forward LMs
    - Practical issues: vanishing / exploding gradient
    - Variants: LSTM, GRU, etc.

Summary

- **Machine Translation & Sequence-to-sequence** models
  - Machine translation
    - History: statistical MT → Neural MT
  - **Encoder decoder structures** for sequence-to-sequence modeling
    - RNN models
    - Information bottleneck
  - Encoder decoder with **attention**
    - Selecting different "focus" in the source for each step
    - Compute context vectors to summarize the information to condition on
    - Very effective for MT and many applications
  - Attention applications
    - Neural machine translation
    - Image captioning, speech recognition, text summarization, etc.