Language models

Based on lecture by Ekaterina Artemova

Previously in course:

- We discussed embedding models:
  - document – vector
  - word – vector

Previously in course:

- We discussed embedding models:
  - document – vector
  - word – vector
- Both models do not take into account word order

Previously in course:

- We discussed embedding models:
  - document – vector
  - word – vector
- Both models do not take into account word order
- Language model – a probabilistic model that takes into account word order

Language model

- Predict a word in a sequence:

  «it’s snowy in Moscow and sunny in_______»

Language model

- Predict a word in a sequence:

«it's snowy in Moscow and sunny in _Sochi__»

Language model

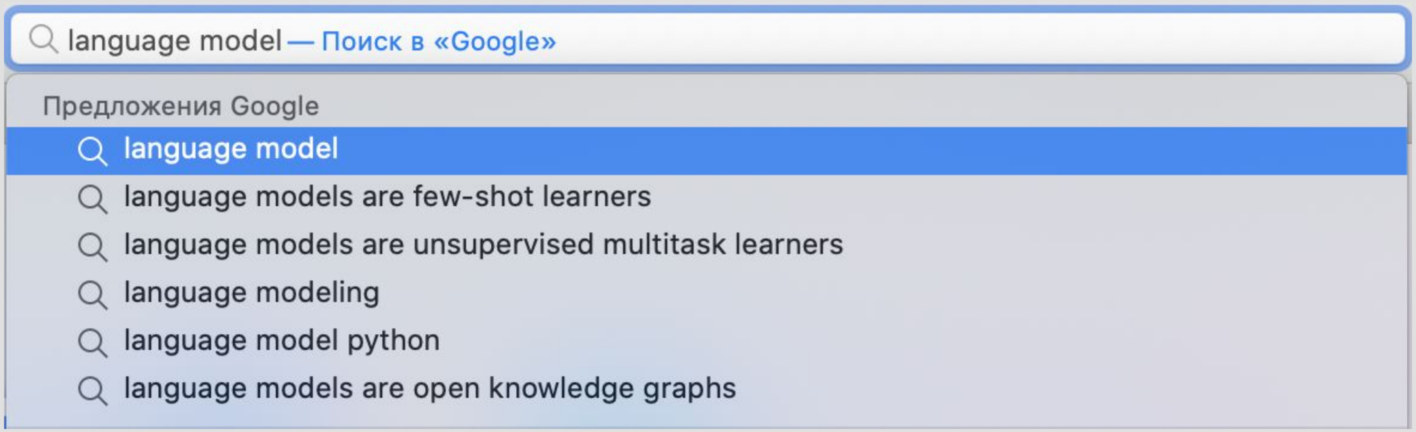
- Predict a word in a sequence:

«it’s snowy in Moscow and sunny in <u> Sochi </u>»

- Choose a more likely sequence:

P(«it’s sunny in <u> Sochi </u>») >P(«it’s sunny in <u> Siberia</u> »)

Example: search hints



Probabilistic language model

- Formal task:
- suppose W = $(w_1, w_2, \dots, w_n)$ - sentence, w_i - a word, V - vocabulary

Probabilistic language model

- Formal task:
- suppose W = $(w_1, w_2, \ldots, w_n)$ - sentence, w_i - a word, V - vocabulary

- Probability of sentence W:
  - $P(W) = P(w_1, w_2, \ldots, w_n)$

Probabilistic language model

- Formal task:
- suppose W = $(w_1, w_2, \ldots, w_n)$ - sentence,
  w_i - a word, V - vocabulary

- Probability of sentence W:
  - $P(W) = P(w_1, w_2, \ldots, w_n)$

- Probability of the next word in sequence:
  - $P(w_n | w_{n-1}, w_{n-2}, \ldots, w_1)$

How to estimate the probability of a sentence?

- Expand the word sequence:

How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$

How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$
$$= P(w_1, w_2, \ldots, w_n) =$$

How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$
$$= P(w_1, w_2, \ldots, w_n) =$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\ldots P(w_n|w_1, \ldots w_n - 1)$$

How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$
$$= P(w_1, w_2, \ldots, w_n) =$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\ldots P(w_n|w_1,\ldots w_n - 1)$$

P (“it is sunny today”) =

How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$
$$= P(w_1, w_2, \ldots, w_n) =$$
$$= P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\ldots P(w_n|w_1, \ldots w_n - 1)$$

P (“it is sunny today”) = P (“it”) • P (“is | it”) • P (“sunny | it is”) • P (“today | it is sunny”)

How to estimate the probability of a sentence?

- Denote $w_{1:i-1}$ - left context of word w_i

How to estimate the probability of a sentence?

- Denote $w_{1:i-1}$ - left context of word w_i
- Expand the word sequence:

$$P(W) = P(w_1, w_2, \ldots, w_n) = \prod_i P(w_i | w_{1:i-1})$$

How to estimate the probability of a sentence?

$$P(w_i|w_{1:w_i-1}) = \frac{\text{count}(w_{1:i-1}w_i)}{\text{count}(w_{1:i-1})}$$

A. A. Markov

1856 – 1922

one of the most famous Russian mathematicians

Markov model

- Given a word w_i and its left context $w_{1:i-1}$

Markov model

- Given a word w_i and its left context $w_{1:i-1}$
- Limit the length of left context

Markov model

- Given a word w_i and its left context $w_{1:i-1}$
- Limit the length of left context:

  unigram model

P (“it is sunny today”) = P (“it”) • P (“is”) • P (“sunny”) • P (“today”)

Markov model

- Given a word w_i and its left context $w_{1:i-1}$
- Limit the length of left context:

unigram model

P (“it is sunny today”) = P (“it”) • P (“is”) • P (“sunny”) • P (“today”)

bigram model

P (“it is sunny today”) = P (“it”) • P (“is | it”) • P (“sunny | is”) • P (“today | sunny”)

Maximum likelihood estimation

- Estimation $P(w_i)$: $\frac{\text{count}(w_i)}{\sum_{w_j \in V} \text{count}(w_j)}$

where V - the vocabulary

Maximum likelihood estimation

- Estimation $P(w_i): \frac{\text{count}(w_i)}{\sum_{w_j \in V} \text{count}(w_j)}$

where V - the vocabulary

- Estimation $P(w_i | w_{i-1}): \frac{\text{count}(w_{i-1} w_i)}{\text{count}(w_{i-1})}$

Problems

- $\text{count}(w_{i-1}w_i)$ can be equal to zero

Then $\frac{\text{count}(w_{i-1}w_i)}{\text{count}(w_{i-1})} = 0$ and $P(W) = 0$

Problems

- $\text{count}(w_{i-1}w_i)$ can be equal to zero

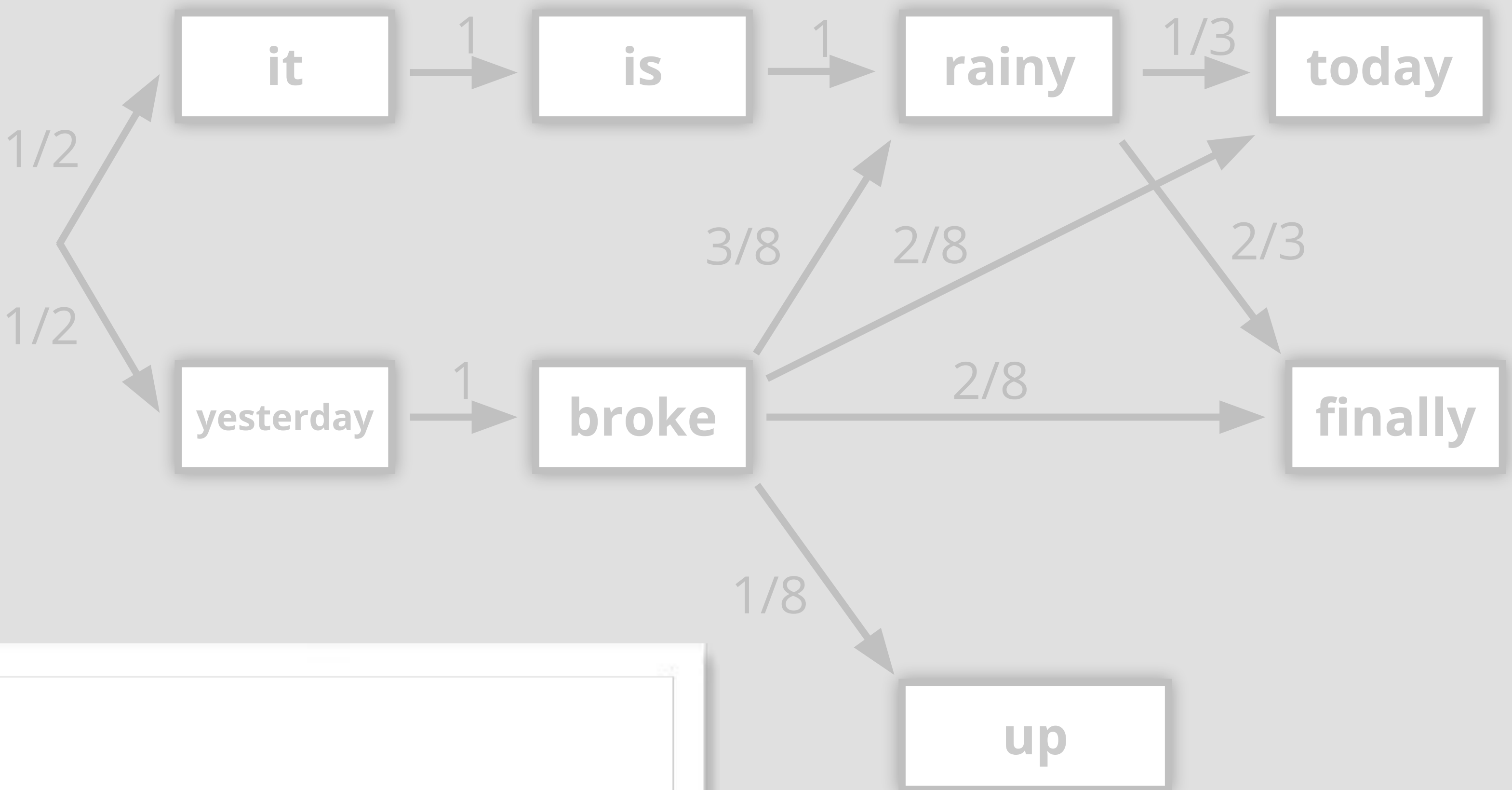
Then $\frac{\text{count}(w_{i-1}w_i)}{\text{count}(w_{i-1})} = 0$ and $P(W) = 0$

Solution: smoothing

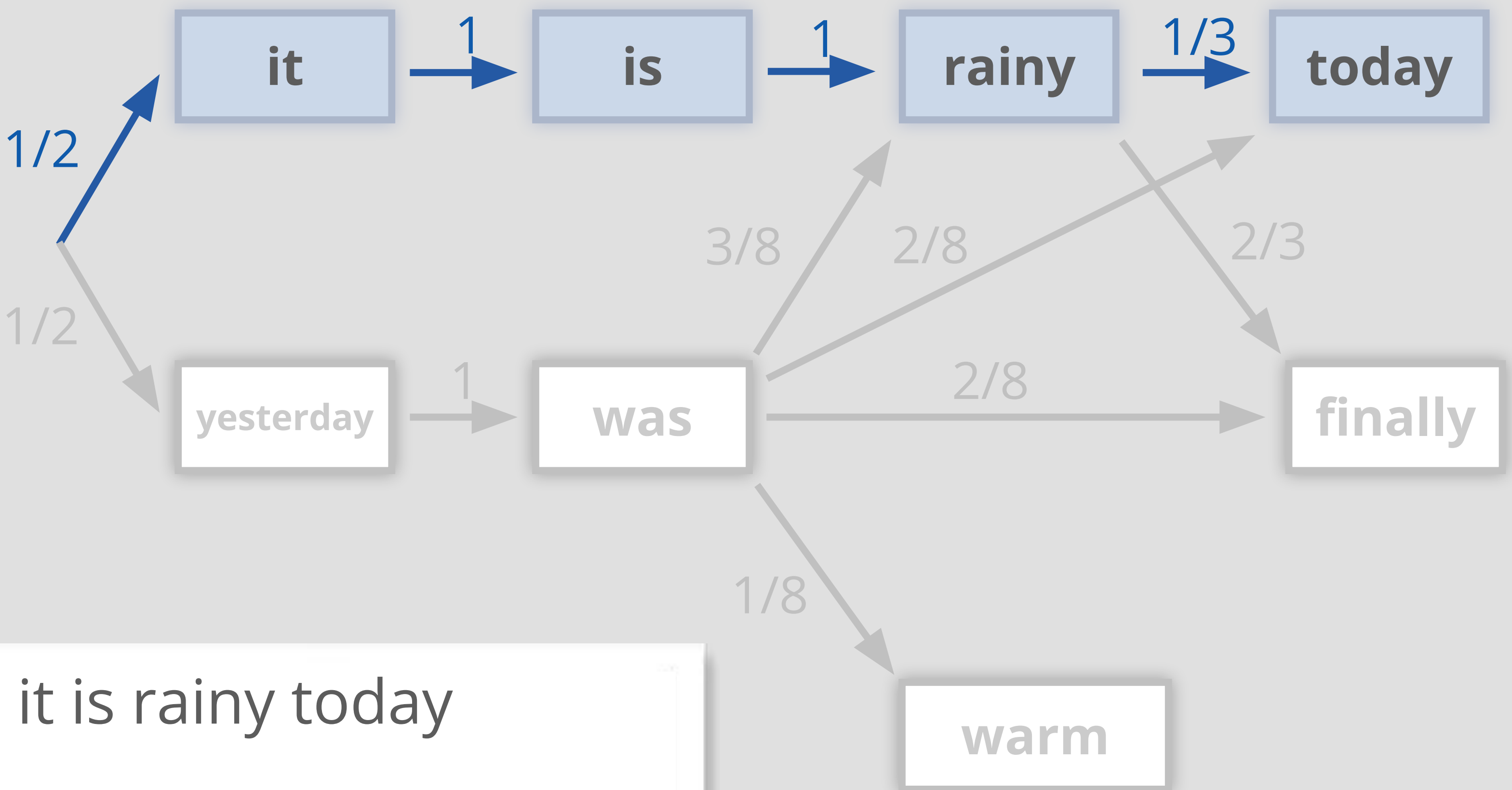
- suppose each word occurs in text at least α times

Then $\frac{\text{count}(w_{i-1}w_i) + \alpha}{\text{count}(w_{i-1}) + \alpha|V|}$

Text generation

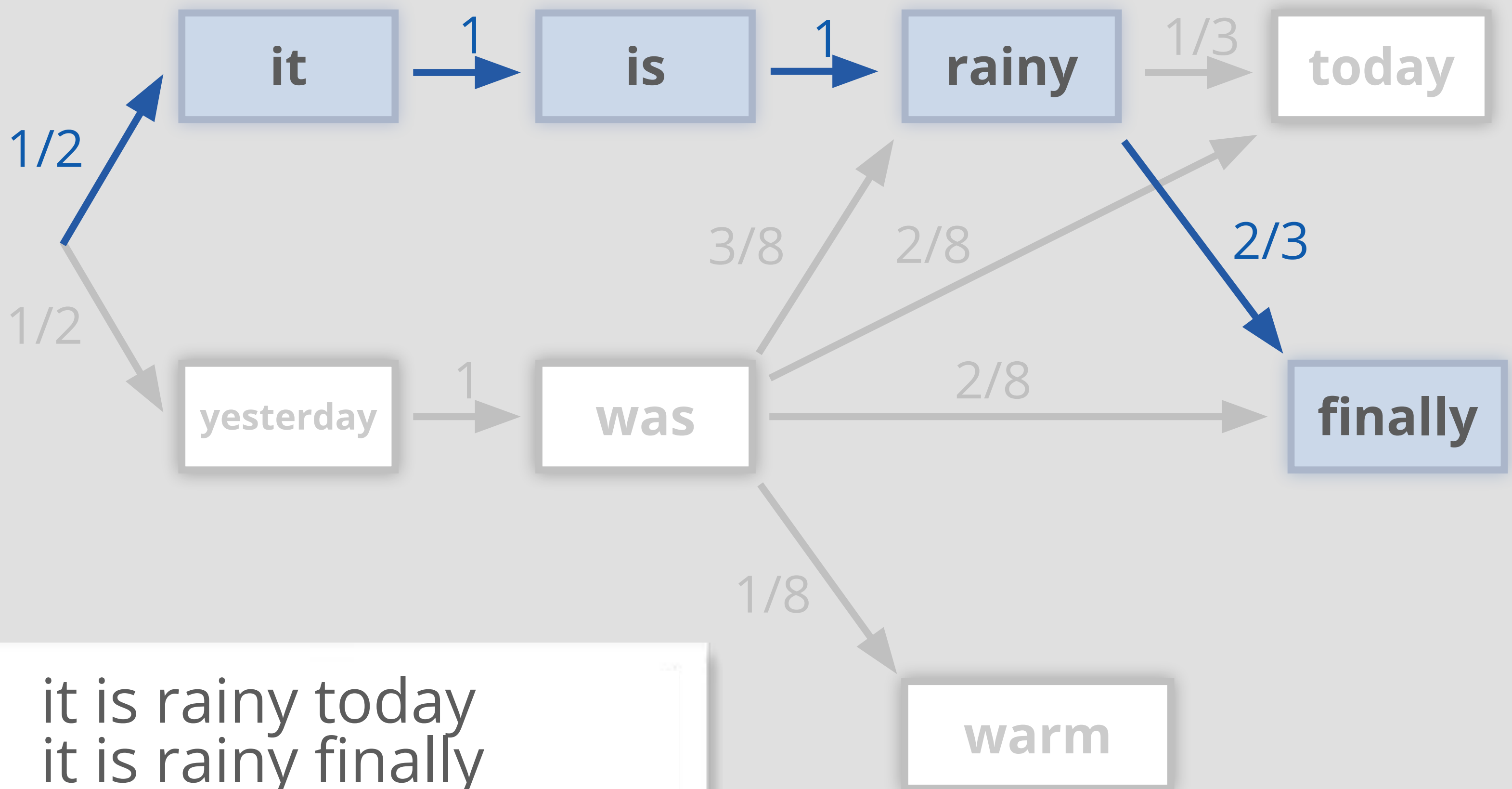


Text generation



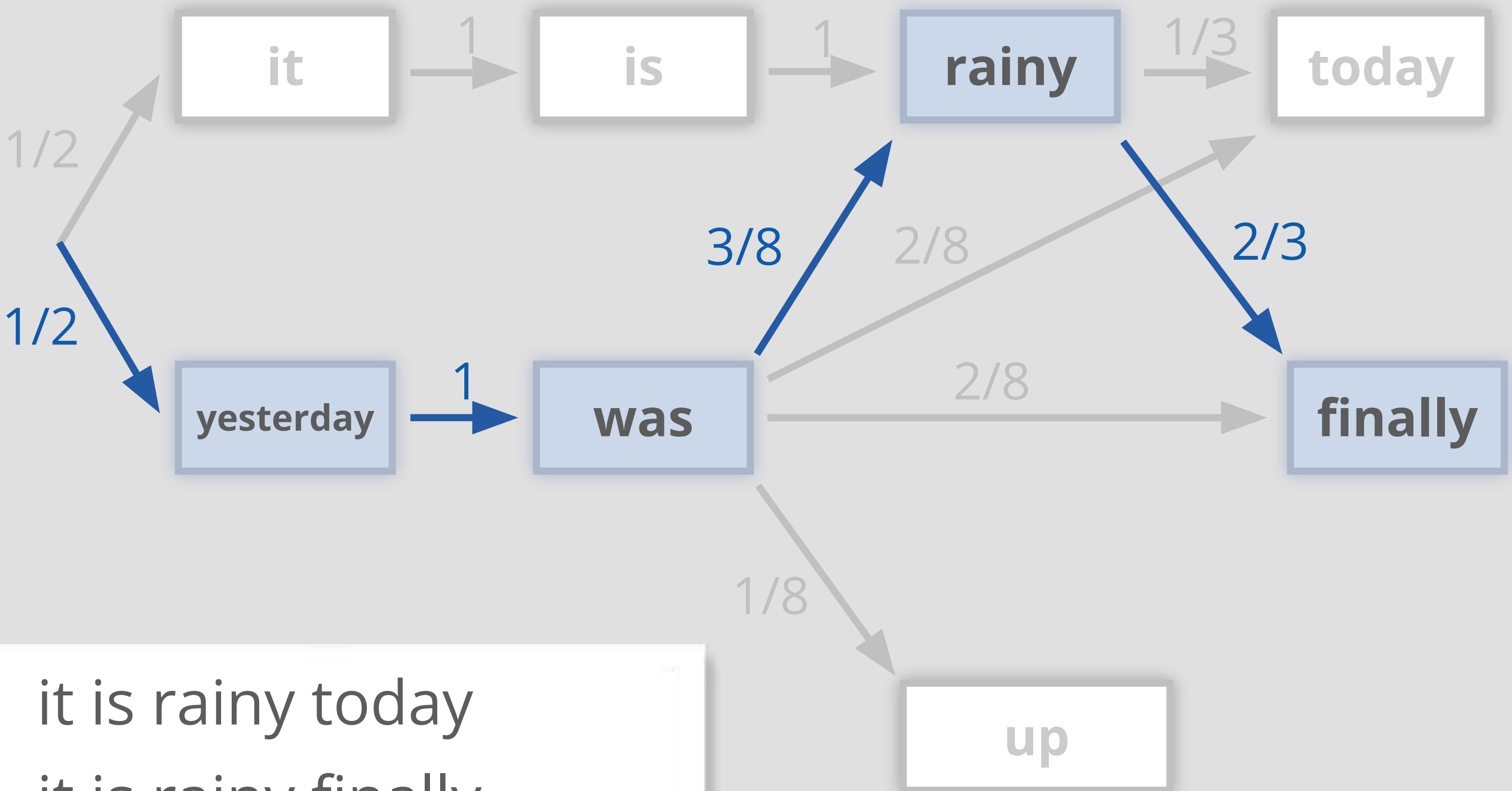
- it is rainy today

Text generation



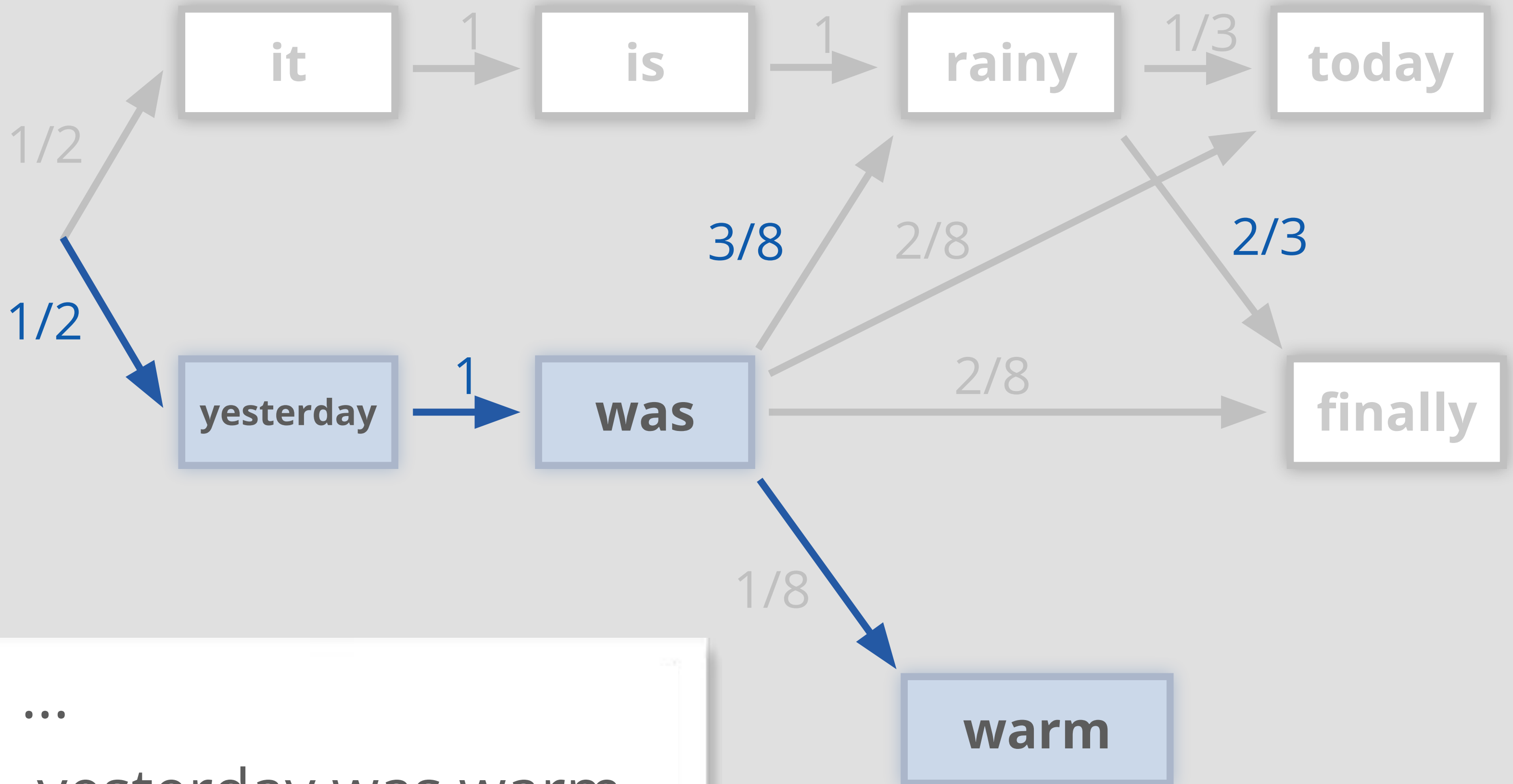
- it is rainy today
- it is rainy finally

Text generation



- it is rainy today
- it is rainy finally
- yesterday was rainy finally

Text generation



- ...
- yesterday was warm

Countable language models problems

- They generate nonsense text

Countable language models problems

- They generate nonsense text
  - It was rainy like your new shoes where is my wallet

Countable language models problems

- They generate nonsense text
  - It was rainy like your new shoes where is my wallet
- Require very large number of parameters:

  bigram model = $|V|^2$ parameters

Language models quality

- External quality evaluation:
  - Use the language model in the next task
  - Evaluate the metrics for this task

Language models quality

- Internal quality evaluation: perplexity

$$P(W) = P(w_1, w_2, \dots, w_N) = P(w_1)P(w_2)\ P(w_N) = \prod_{i=1}^{N} P(w_i)$$

Language models quality

- Internal quality evaluation: perplexity

$$P(W) = P(w_1, w_2, \dots, w_N) = P(w_1)P(w_2)\ P(w_N) = \prod_{i=1}^{N} P(w_i)$$

$$P(W)^{1/N} = \left(\prod_{i=1}^{N} \boldsymbol{P(wi)}\right)^{1/N}$$

Language models quality

- Internal quality evaluation: perplexity

$$P(W) = P(w_1, w_2, \dots, w_N) = P(w_1)P(w_2)\ P(w_N) = \prod_{i=1}^{N} P(w_i)$$

$$P(W)^{1/N} = (\prod_{i=1}^{N} P(wi))^{1/N}$$

$$PP(W) = P(w_1, w_2, \dots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$

Excersise

- Compute perplexity of the dice, given the following dataset: W = {1,2,3,4,5,6,6,5,4,3}

Excersise

- Compute perplexity of the dice, given the following dataset: W = {1,2,3,4,5,6,6,5,4,3}

$$PP(W) = \frac{\mathbf{1}}{\left(\left(\frac{\mathbf{1}}{\mathbf{6}}\right)^{\mathbf{10}}\right)^{\mathbf{1}/\mathbf{10}}}$$

Excersise

- Compute perplexity of the dice, given the following dataset: W = {1,2,3,4,5,6,6,5,4,3}

$$PP(W) = \frac{\mathbf{1}}{\left(\left(\frac{\mathbf{1}}{\mathbf{6}}\right)^{\mathbf{10}}\right)^{\mathbf{1}/\mathbf{10}}} = \mathbf{6}$$

Excersise

- Compute perplexity of the biased dice, given the following dataset: W = {1,1,2,1,3,1,1,4,1,5,6,1} and P(1) = 7/12

$$PP(W) = \frac{\mathbf{1}}{\left(\left(\frac{\mathbf{7}}{\mathbf{12}}\right)^{\mathbf{7}} \cdot \left(\frac{\mathbf{1}}{\mathbf{12}}\right)^{\mathbf{5}}\right)^{\mathbf{1}/\mathbf{12}}}$$

Excersise

- Compute perplexity of the biased dice, given the following dataset: W = {1,1,2,1,3,1,1,4,1,5,6,1} and P(1) = 7/12

$$PP(W) = \frac{\mathbf{1}}{\left(\left(\frac{\mathbf{7}}{\mathbf{12}}\right)^{\mathbf{7}} \cdot \left(\frac{\mathbf{1}}{\mathbf{12}}\right)^{\mathbf{5}}\right)^{\mathbf{1}/\mathbf{12}}} \approx \mathbf{4}$$

Language models quality

- PP(X)1 = 962
- PP(X)2 = 170
- PP(X)2 = 109

Unigram

Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

Bigram

Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

Trigram

They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Perplexity and entropy

- Internal quality evaluation: perplexity

$$PP(W) = 2^{H(W)} = 2^{-\frac{1}{N}\log_2 P(w_1,w_2,\ldots,w_N)}$$

$$= \left(2^{\log_2 P(w_1,w_2,\ldots,w_N)}\right)^{-\frac{1}{N}}$$

$$= P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1,w_2,\ldots,w_N)}}$$

Language models quality

- Internal quality evaluation: perplexity

$$PP(W) = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

- The less perplexity the better
- The better the language model, the better the predicted text

Language models quality

- Internal quality evaluation: perplexity

$$PP(W) = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, \ldots, w_N)}}$$

- The less perplexity the better
- The better the language model, the better the predicted text

- Perplexity for bigram model: $PP(W) = \sqrt[N]{\frac{1}{\prod_{i=1}(w_i|w_{i-1})}}$

Language models on counters

- Models simulate the probability of word sequence

Language models on counters

- Models simulate the probability of word sequence
- Bigram model: estimate the probability of word pairs

Language models on counters

- Models simulate the probability of word sequence
- Bigram model: estimate the probability of word pairs
- Smoothing allows to avoid the zero probability problem

Language models on counters

- Models simulate the probability of word sequence
- Bigram model: estimate the probability of word pairs
- Smoothing allows to avoid the zero probability problem
- Quality metrics: perplexity; the less perplexity the better

Language models
based on neural networks

First generation of language models

- Language models based on probabilities have a list of problems:
  - Zero probabilities
  - Difficult to operate with long contexts
  - Too many parameters
  - High perplexity
  - The generated text is often nonsense

Second generation of language models

Language models based on neural networks predict the next word based in the previous context

Language model based on neural networks

- N-gram model: predict word *n* based on previous *n-1*
- Input: $w_1, w_2, \ldots, w_{n-1}$
- Output: w_n

Language model based on neural networks

- N-gram model: predict word *n* based on previous *n-1*
- Input: $w_1, w_2, \ldots, w_{n-1}$
- Output: w_n

«love is all around ________»

Input: «love is all around»

Output: «________»

Language model based on neural networks

- N-gram model: predict word *n* based on previous *n-1*
- Input: $w_1, w_2, \ldots, w_{n-1}$
- Output: w_n

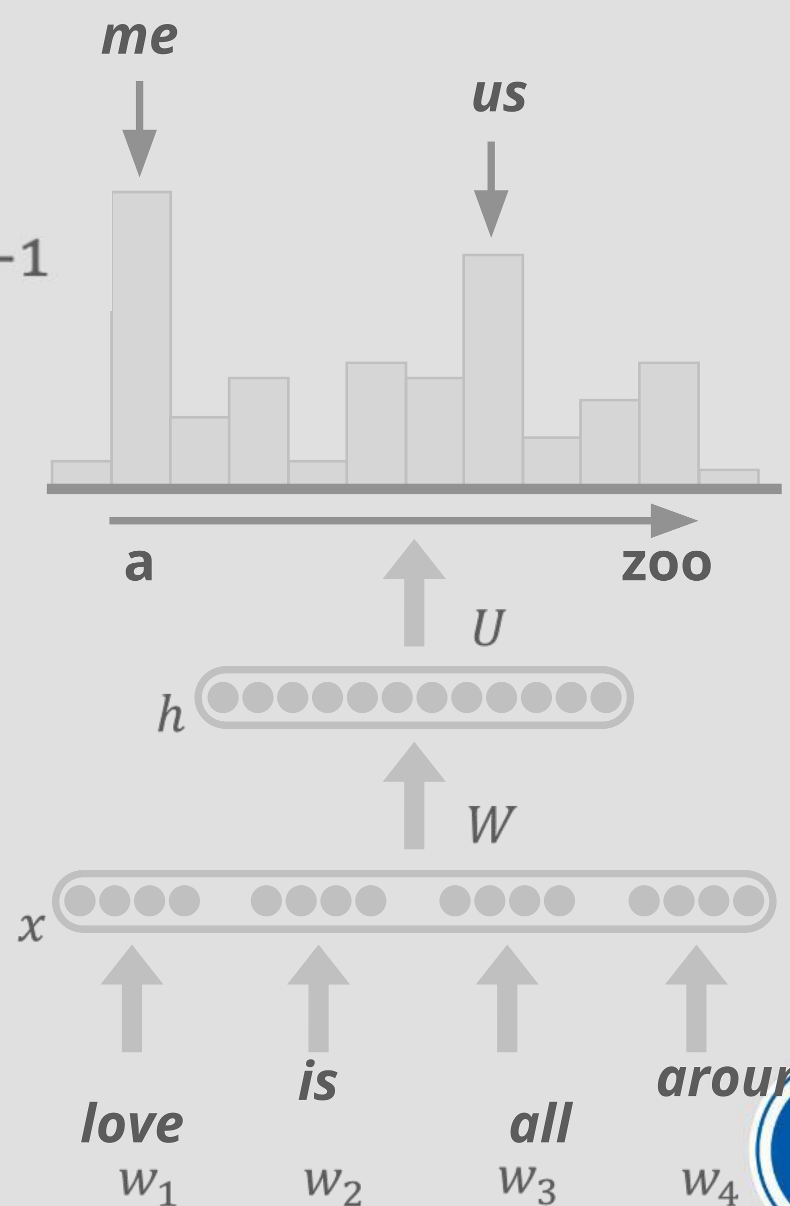
«love is all around ________»

Input: «love is all around»

Output: «_me_»

Language model based on neural networks

- N-gram model
- Input: $w_1, w_2, \ldots, w_{n-1}$
- Output: w_n



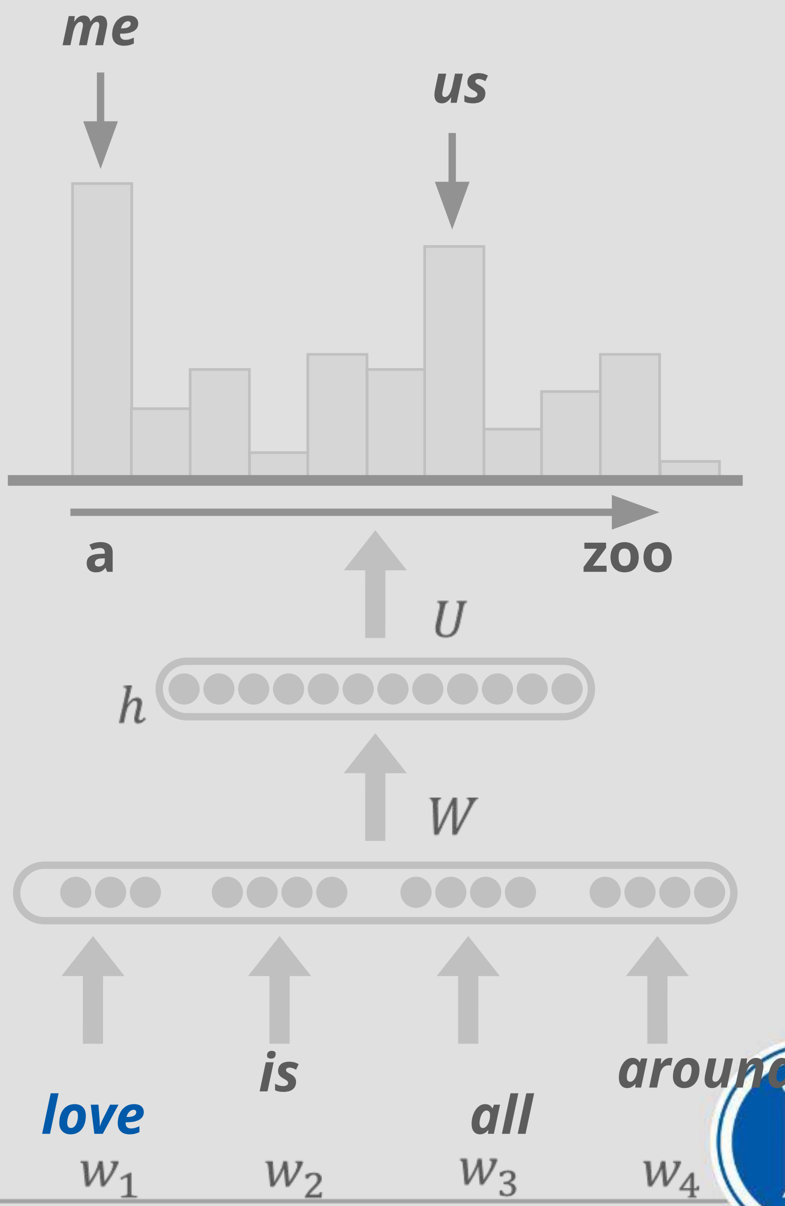
«love is all around ________»

Input: «love is all around»

Output: «_me_»

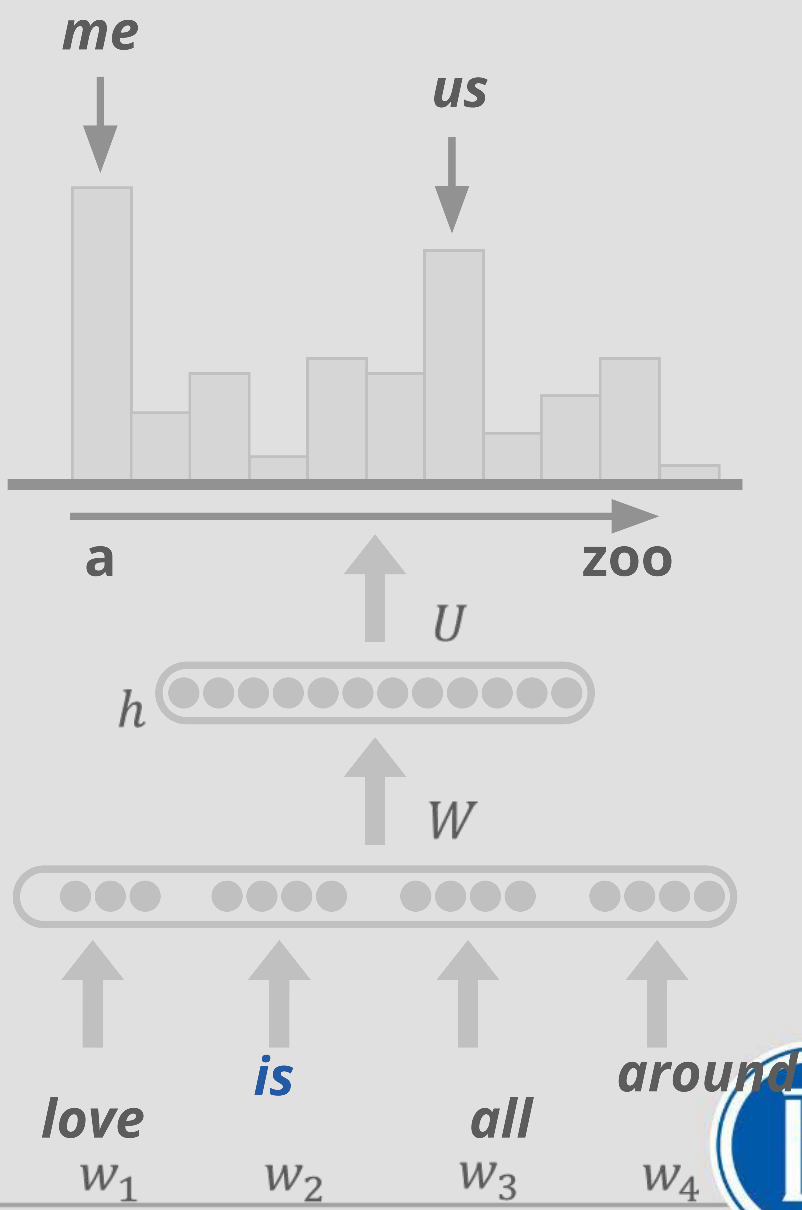
Language model based on neural networks

- First layer: each word corresponds to an embedding of length d



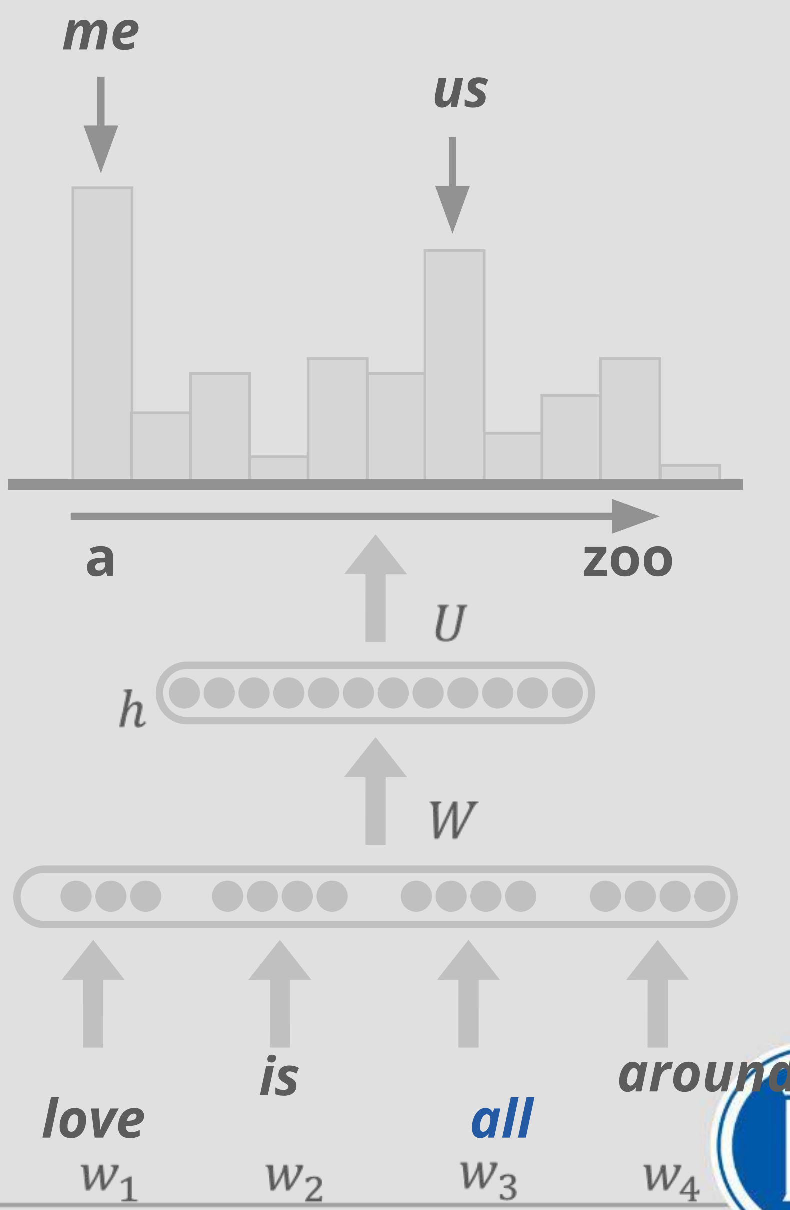
Language model based on neural networks

- First layer: each word corresponds to an embedding of length d



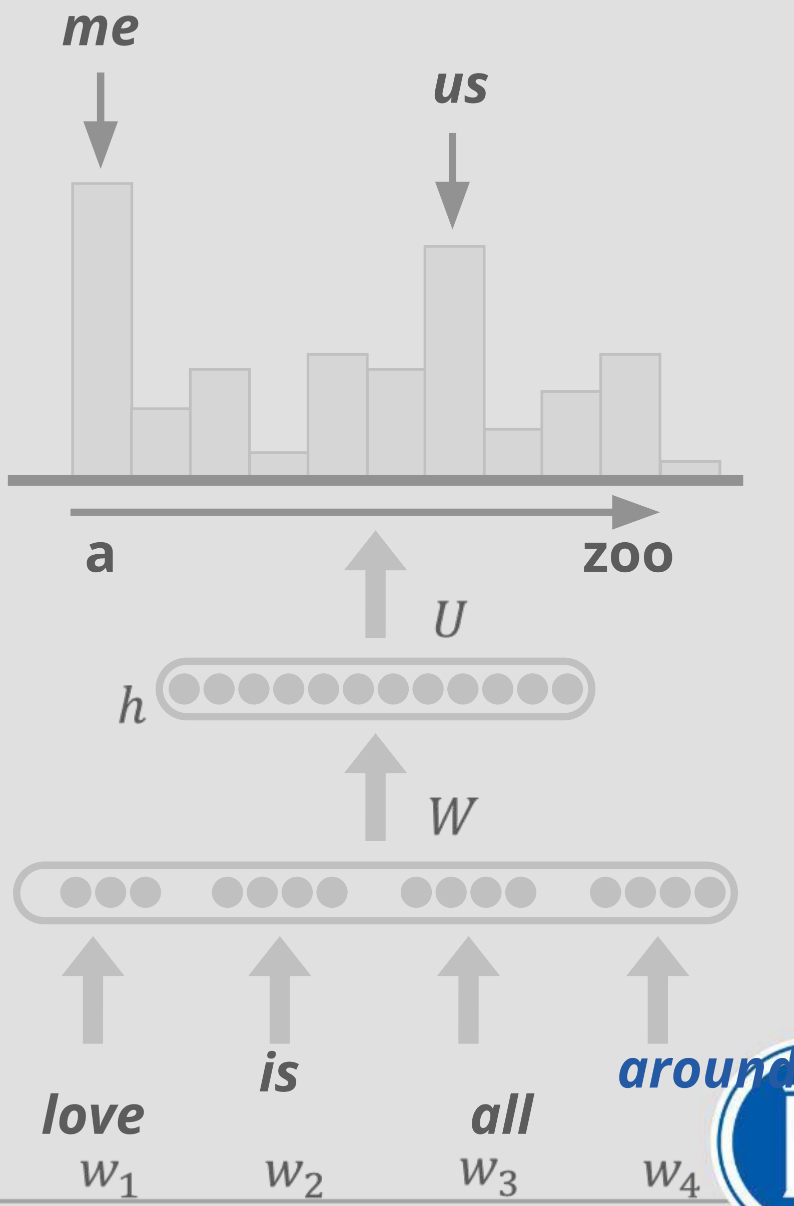
Language model based on neural networks

- First layer: each word corresponds to an embedding of length d



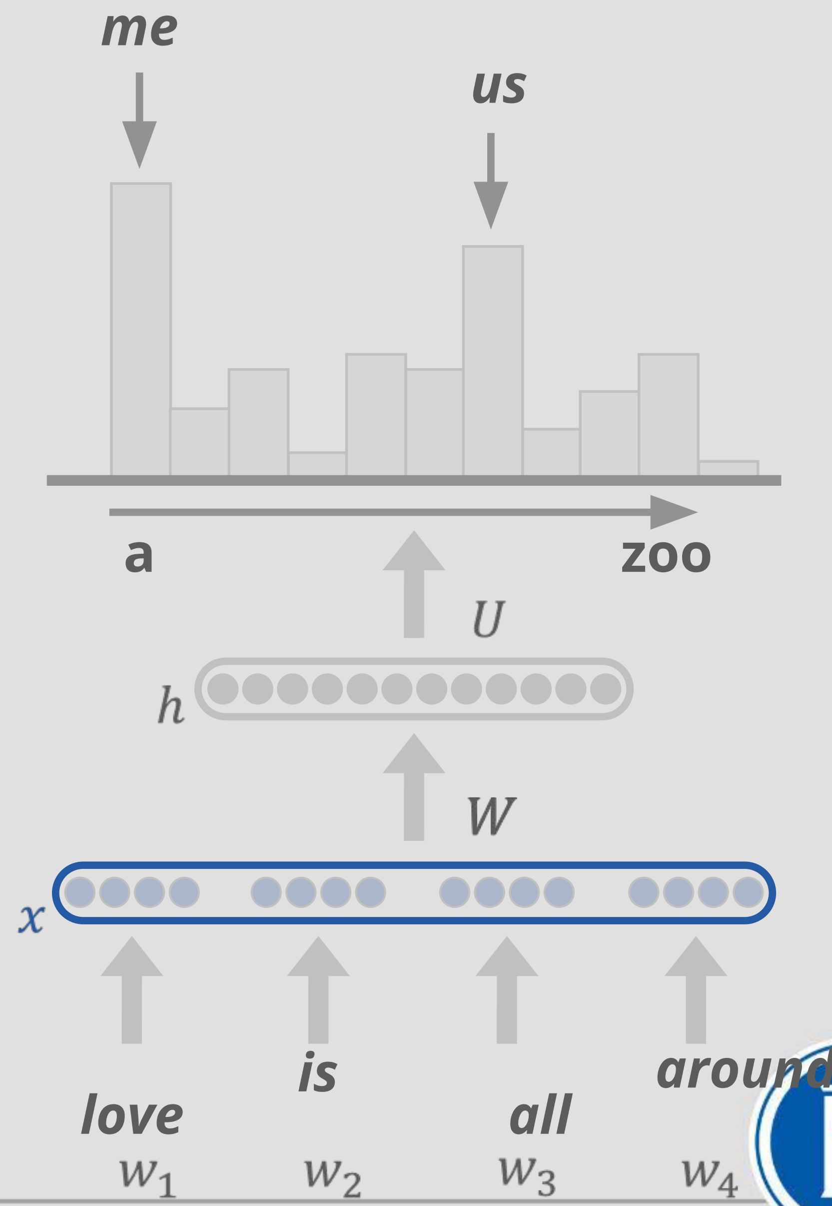
Language model based on neural networks

- First layer: each word corresponds to an embedding of length d



Language model based on neural networks

- First layer: each word corresponds to an embedding of length d
- Concatenate embeddings of input layer: $x = [f_1, f_2, \ldots, f_{n-1}]$

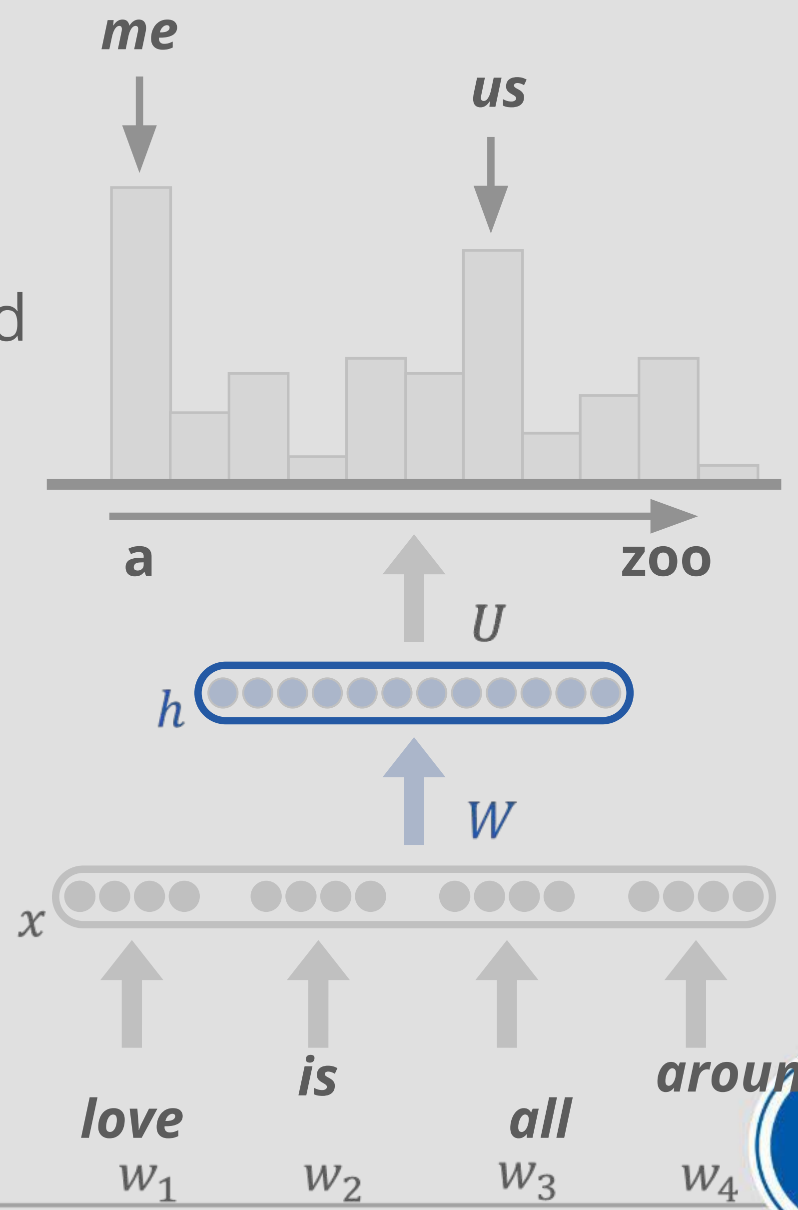


Language model based on neural networks

- Hidden layer:

$$h = \sigma(Wx + b)$$

- W, b_1 - weight matrix and bias vector

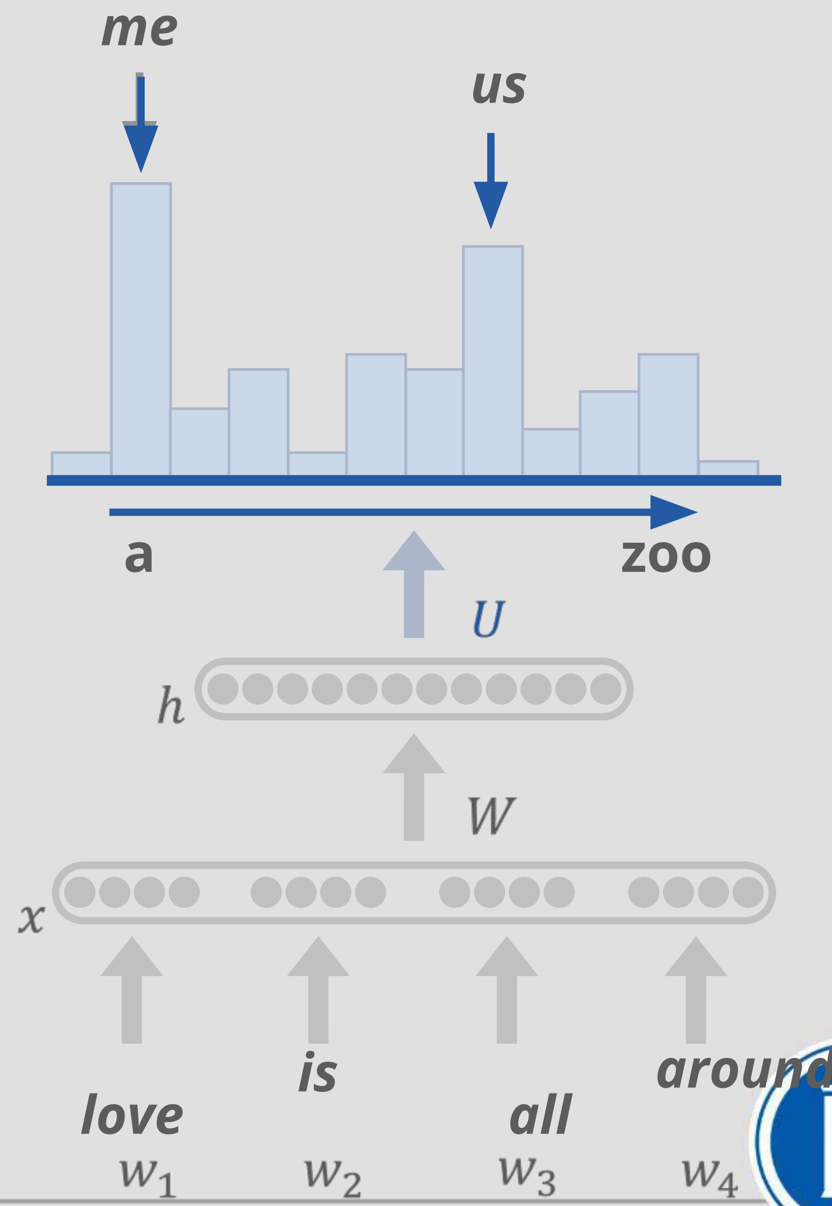


Language model based on neural networks

- The probability on next word:

$$\hat{y} = \mathrm{softmax}(Uh + b_2)$$

$$\mathrm{softmax}(y) = \frac{e^{y}k}{\sum_i e^{y_i}}$$



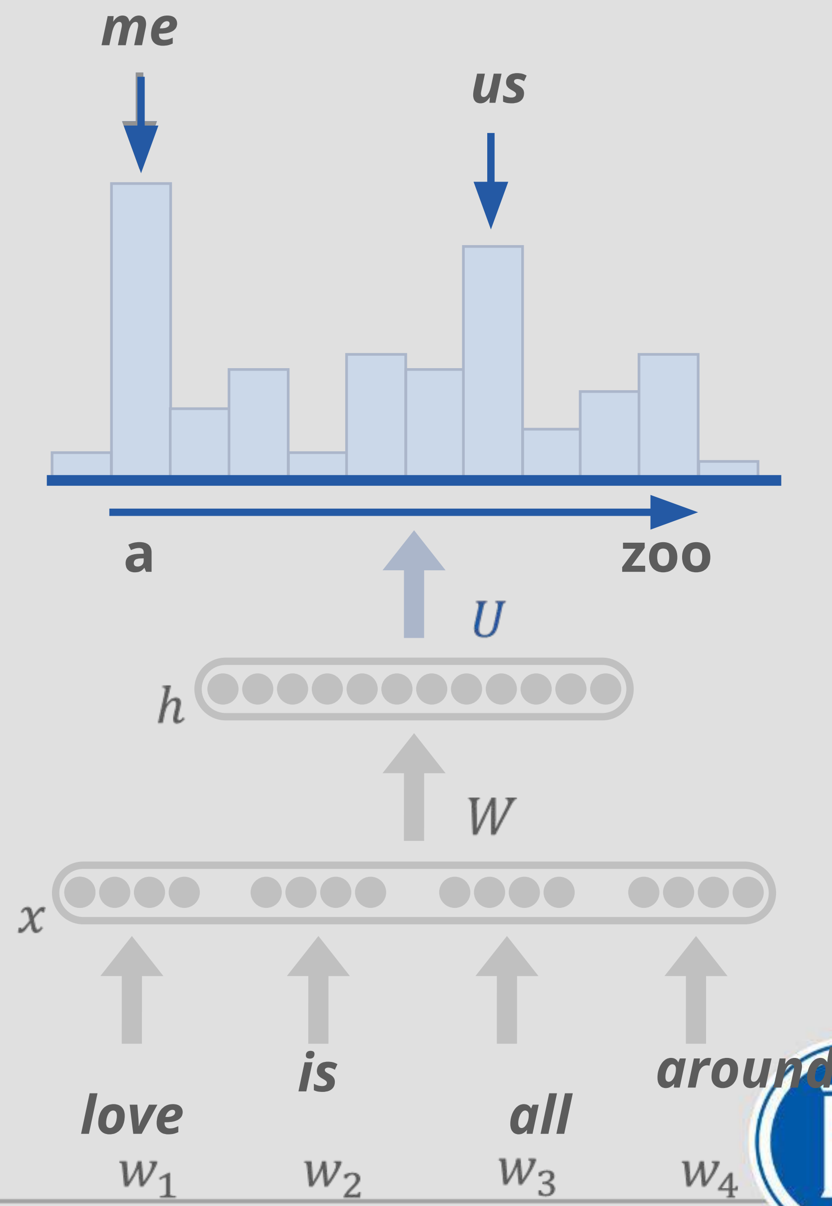
Language model based on neural networks

- The probability on next word:

$$\hat{y} = \text{softmax}(Uh + b_2)$$

$$\text{softmax}(y) = \frac{e^{y_k}}{\sum_i e^{y_i}}$$

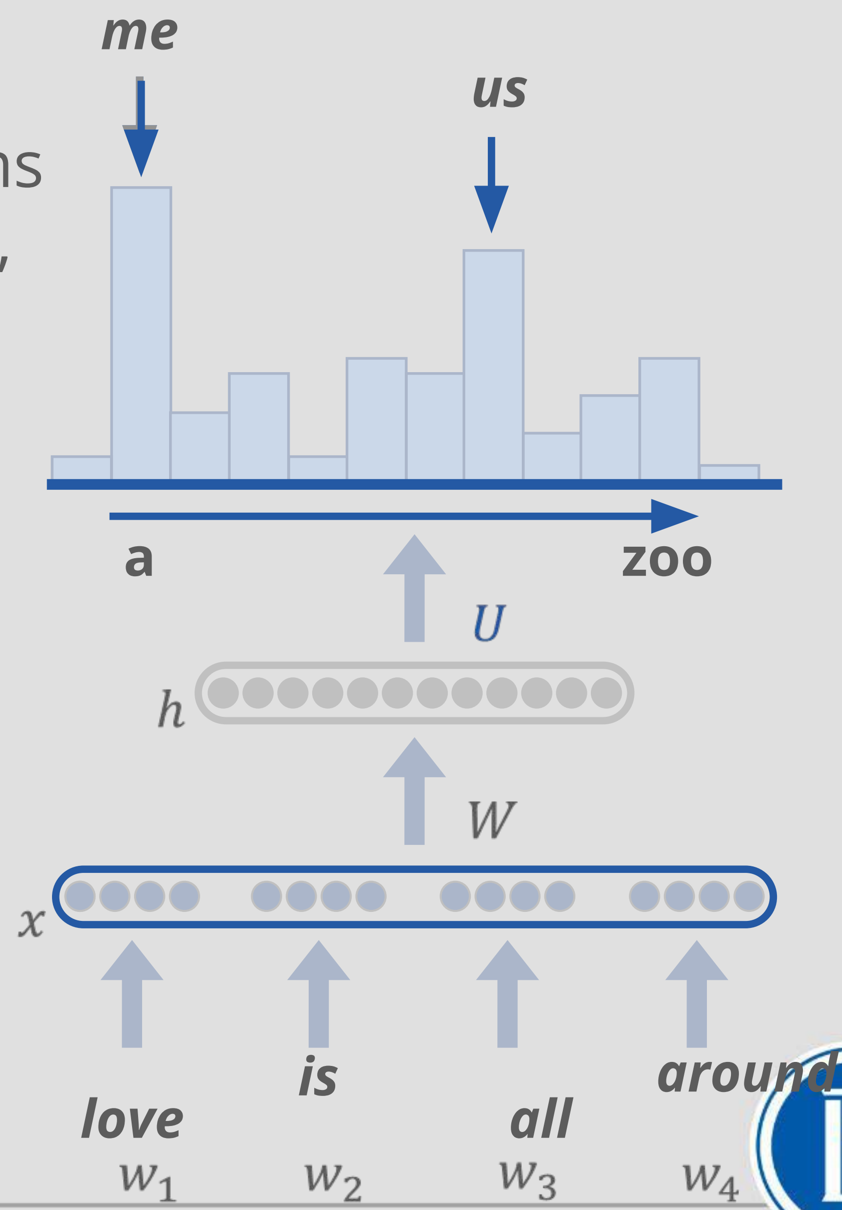
- $\hat{y} \in \mathbb{R}^{|V|}$
- U, b_1 - weight matrix and bias vector, both trainable



Language model based on neural networks

Training:

- split corpus into n-grams
- take *n-1* words as input, predict word *n*



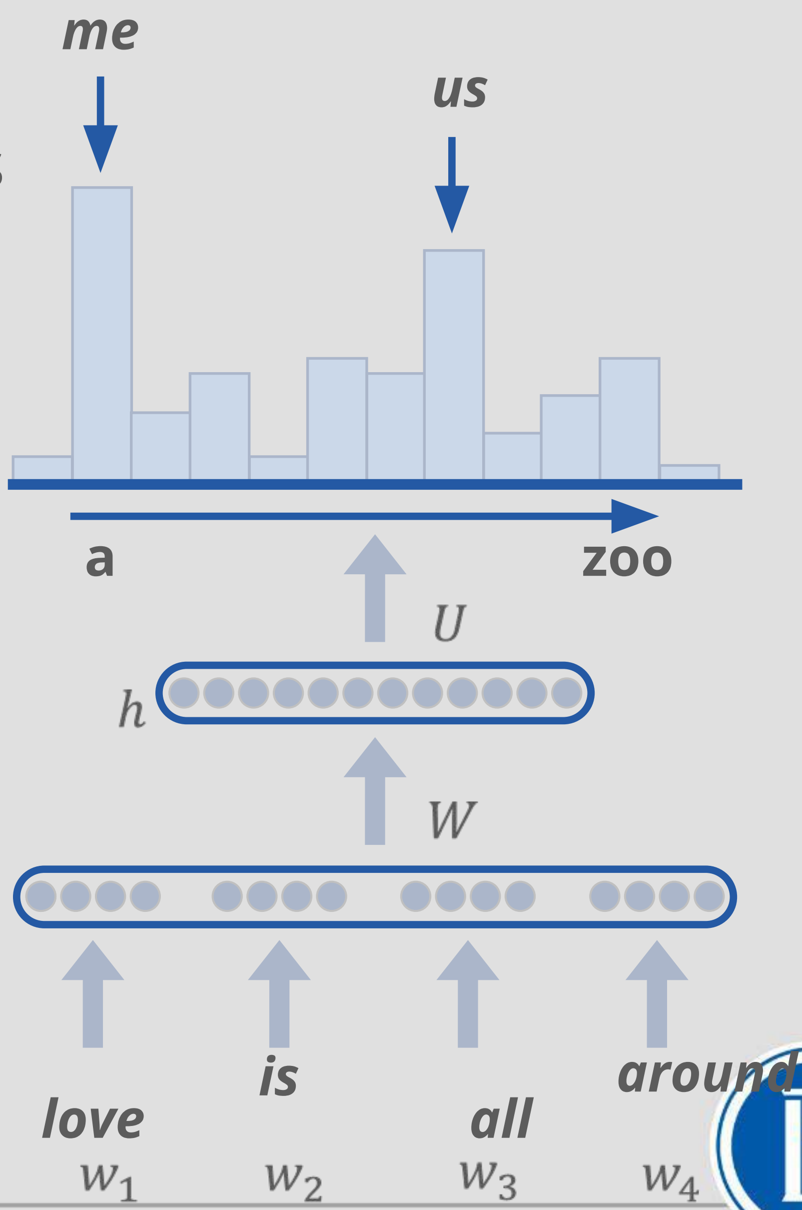
Language model based on neural networks

Training:

- split corpus into n-grams
- take *n-1* words as input, predict word *n*

Loss function: negative likelihood logarithm

$$L = -\frac{1}{T} y \log(P(\hat{y}))$$



Language models

Based on probabilities

Based on neural networks

Language models

Based on probabilities

Based on neural networks

- Many parameters
- High perplexity
- Fast inference
- Fixed length context

Language models

Based on probabilities

- Many parameters
- High perplexity
- Fast computations
- Fixed length context

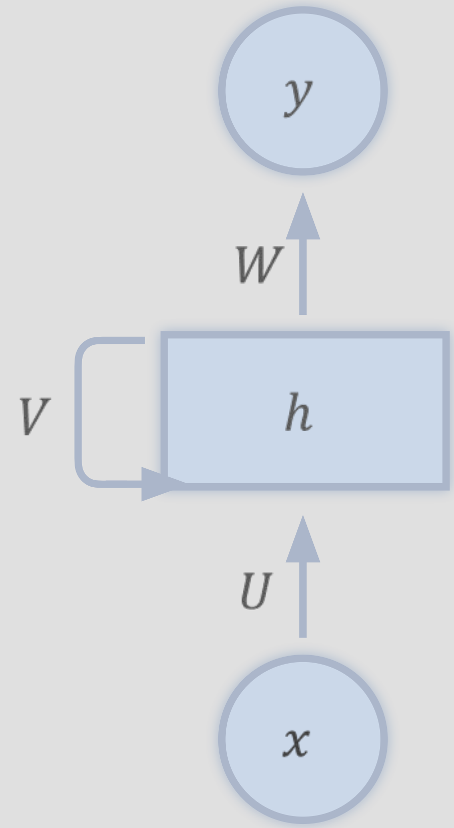
Based on neural networks

- No need to store all n-grams
- Slow computations
- Fixed length context

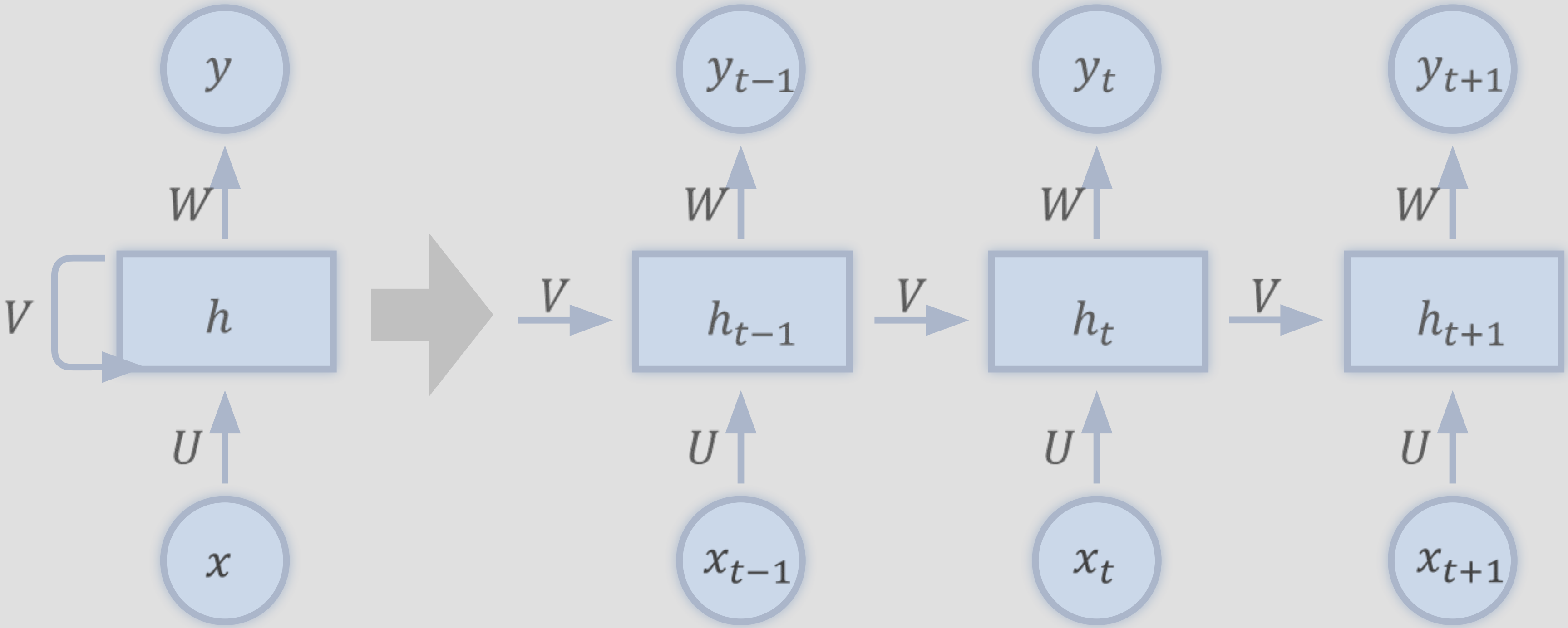
Recurrent neural networks

Recurrent neural network

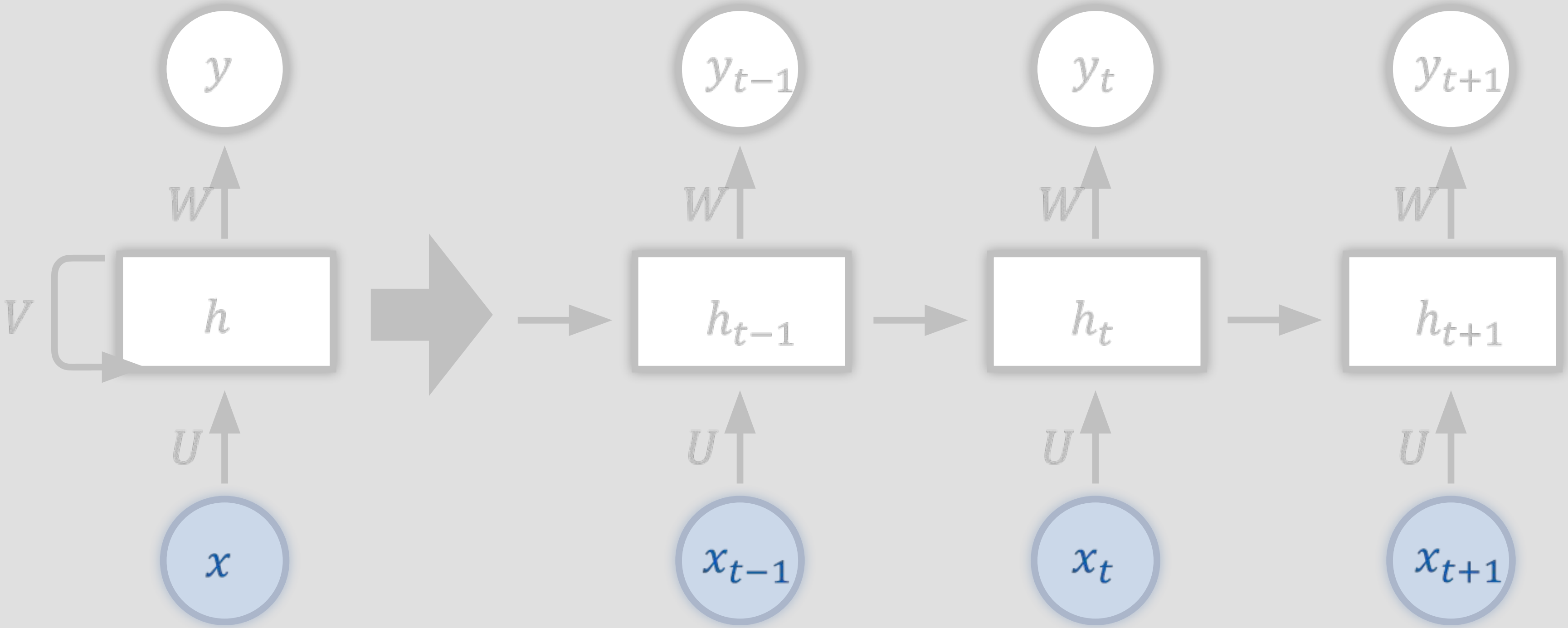
- Works with input sequences of arbitrary length
- Has a state embedding that keeps information about processed input



Recurrent neural network

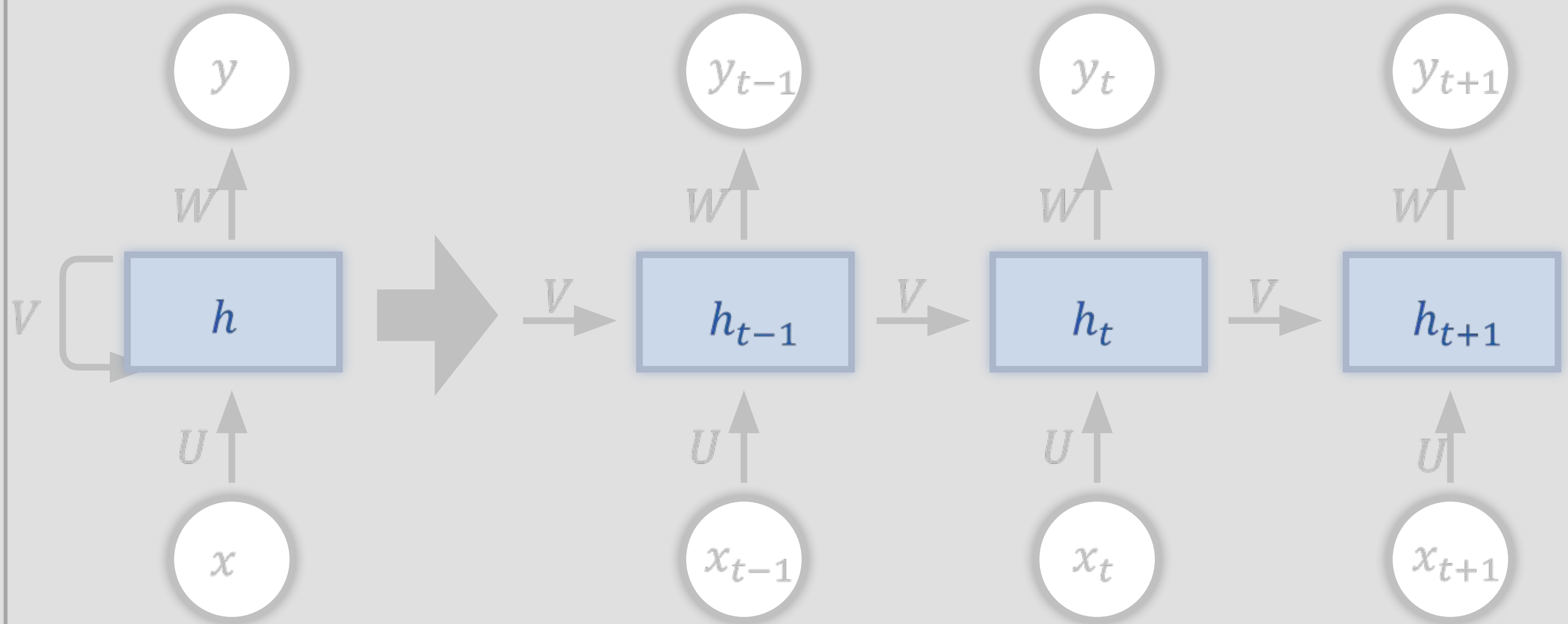


Recurrent neural network



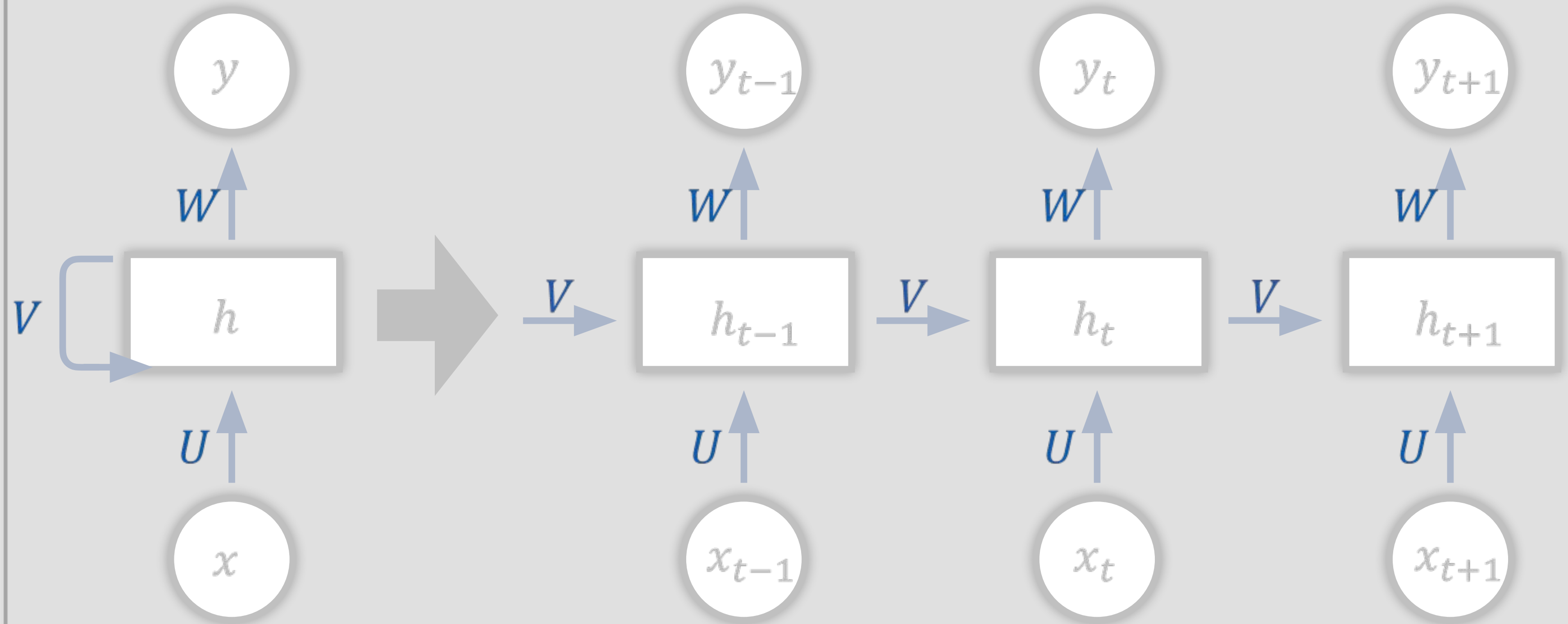
- x - input words (embeddings)

Recurrent neural network



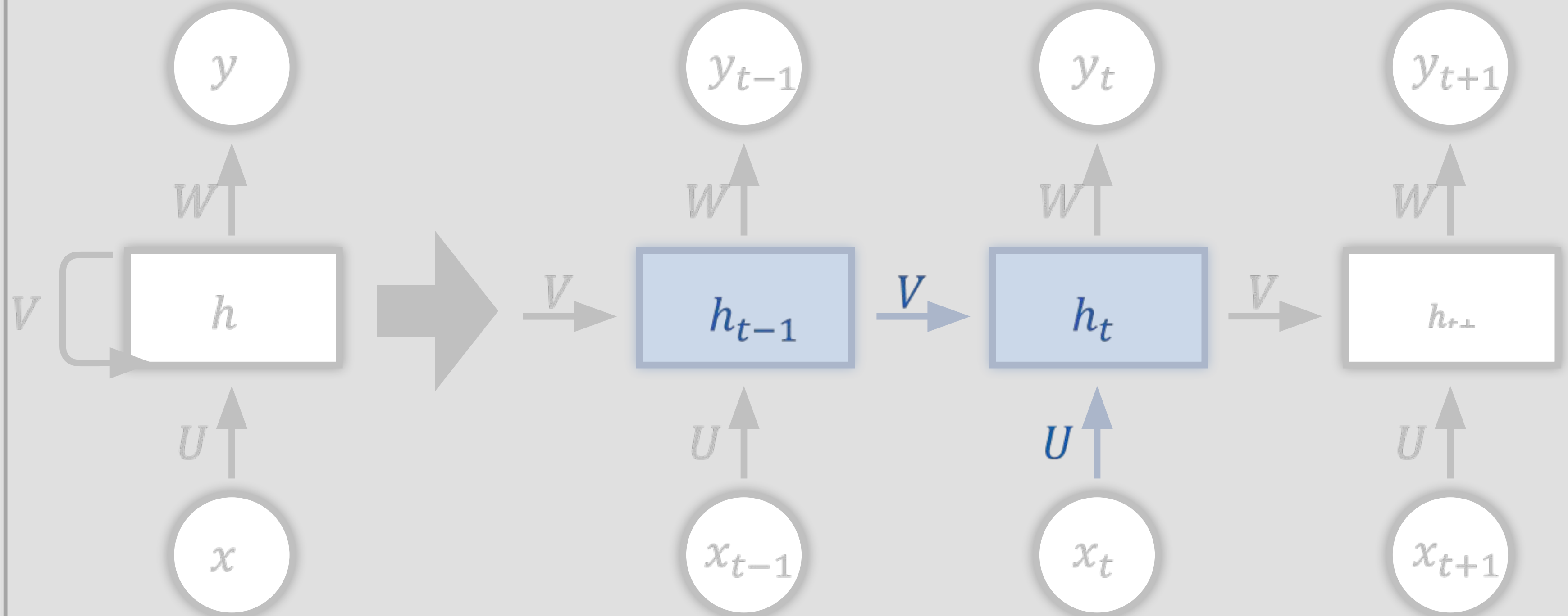
- x - input words (embeddings)
- h - memory embedding

Recurrent neural network



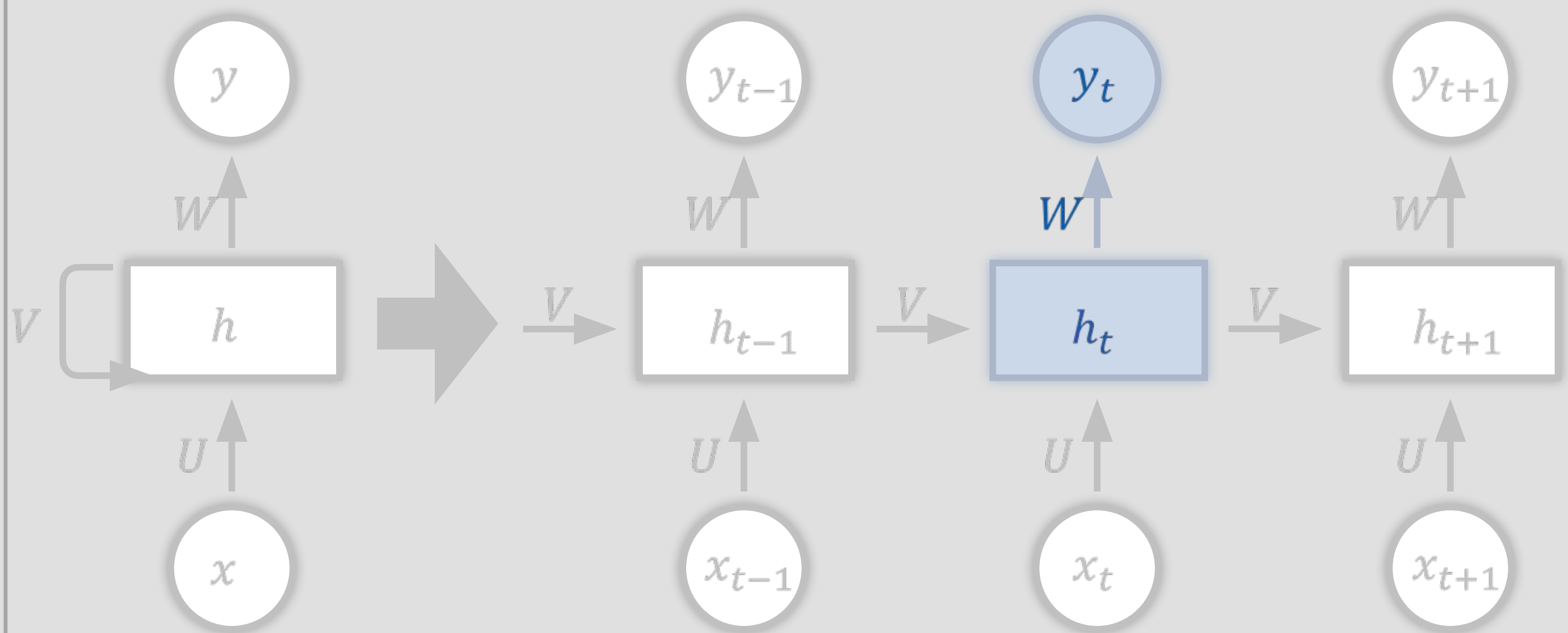
- x - input words (embeddings)
- h - memory embedding
- U, W, V - trainable weight matrices
- y - output

Recurrent neural network



- $h_t = \sigma(Vh_{t-1} + Ux_t + b_1)$

Recurrent neural network

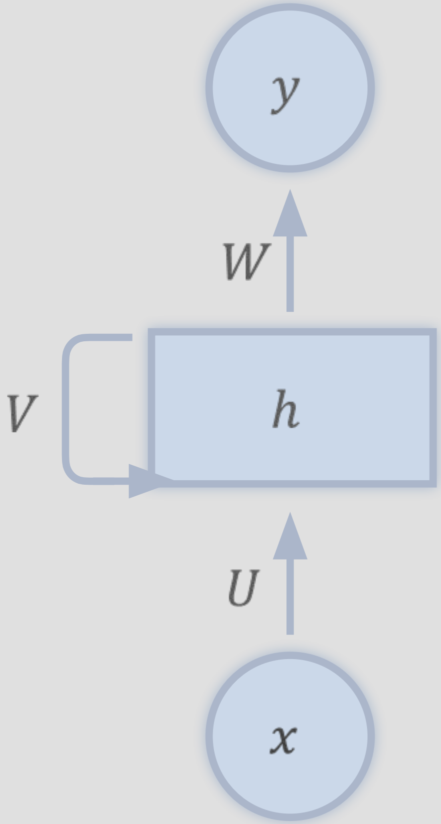


- $h_t = \sigma(Vh_{t-1} + Ux_t + b_1)$
- $\hat{y}_t = softmax(Wh_t + b_2) \in \mathbb{R}^{|V|}$

Training process

- split corpus into words

$$w_1, w_2, \dots, w_n$$



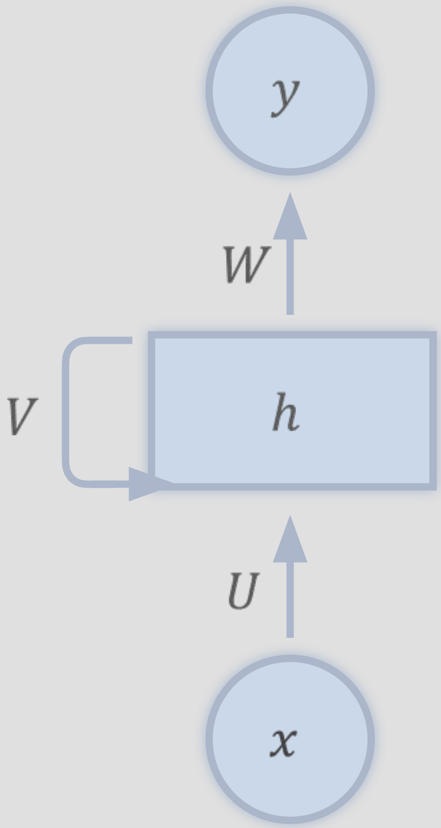
Training process

- split corpus into words

$$w_1, w_2, \ldots, w_n$$

- word embeddings

$$x_i = Ew_i$$



Training process

- split corpus into words

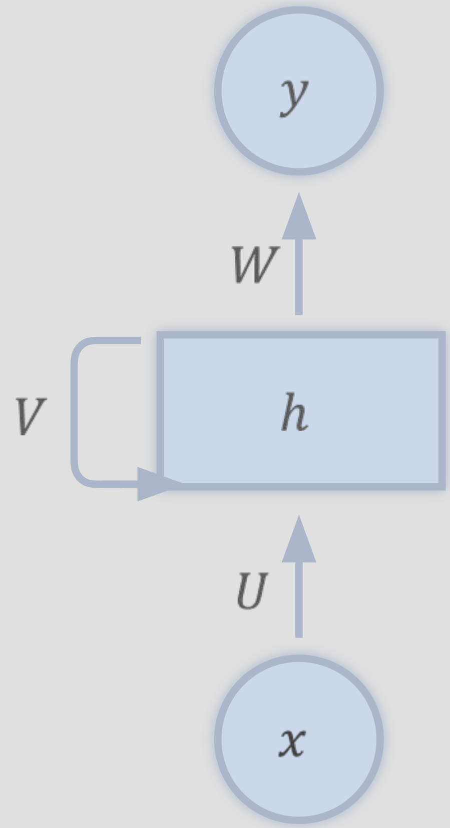
$$w_1, w_2, \ldots, w_n$$

- word embeddings

$$x_i = Ew_i$$

- slide over corpus

$$y_1 = w_2, y_{n-1} = w_n$$



Training process

- split corpus into words

$$w_1, w_2, \ldots, w_n$$

- word embeddings

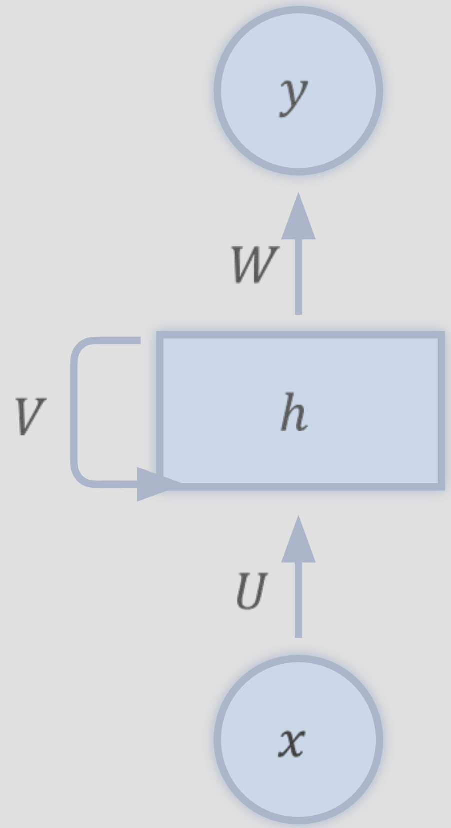
$$x_i = E w_i$$

- slide over corpus

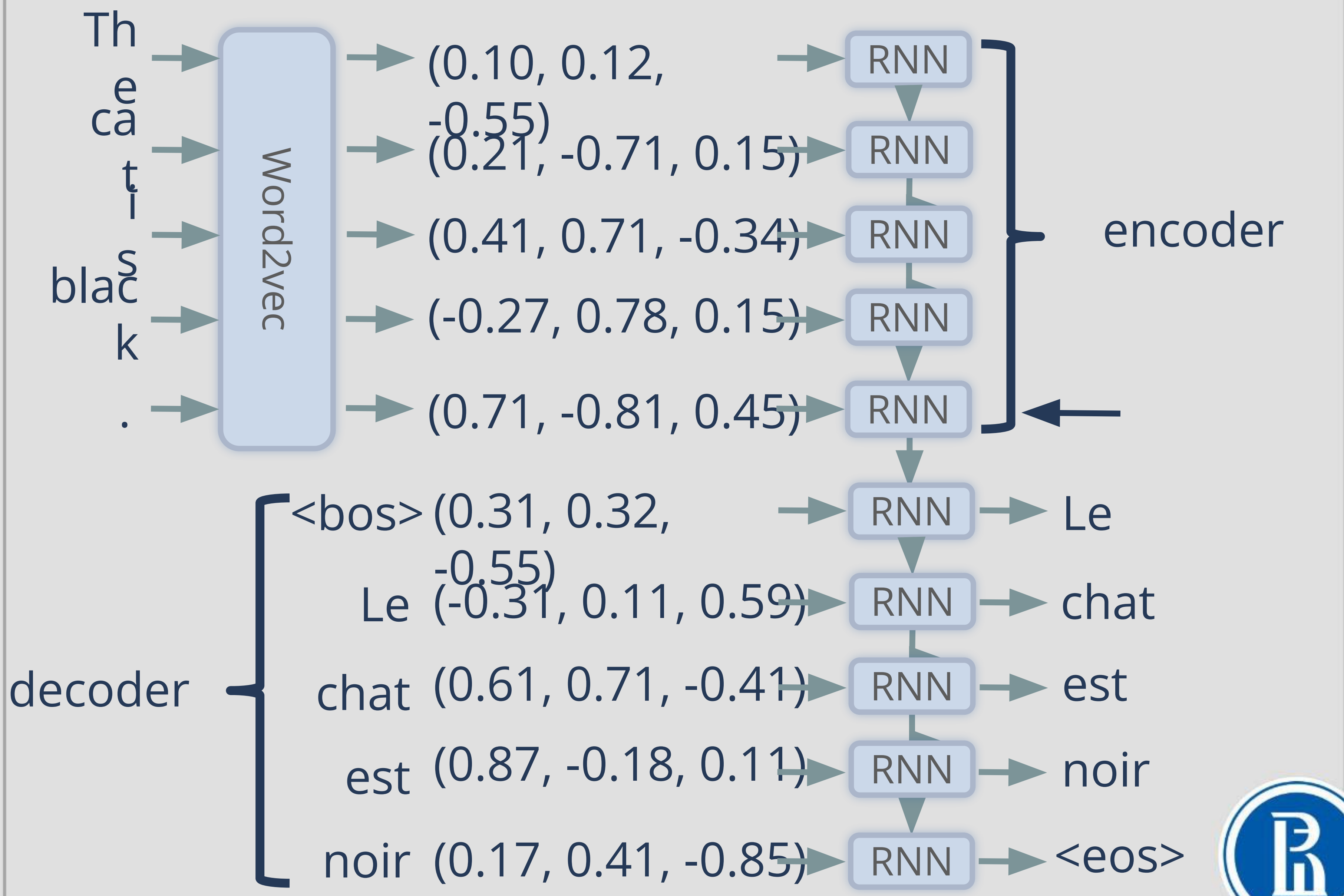
$$y_1 = w_2, y_{n-1} = w_n$$

- predict

$$(x_i, y_i)$$

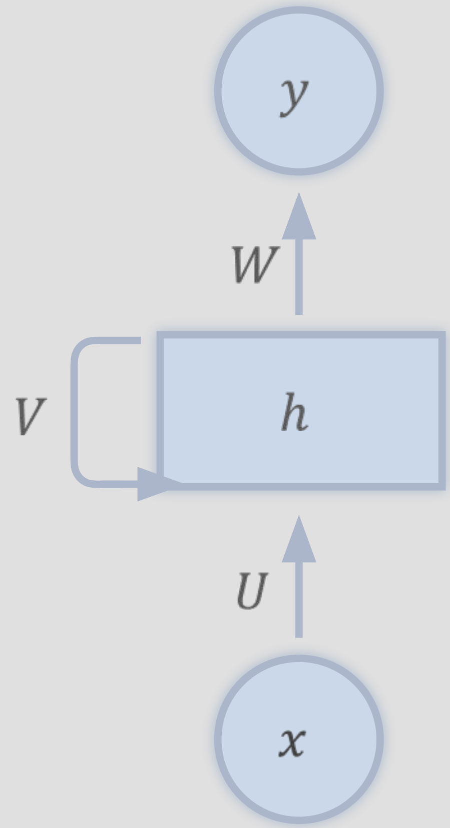


Training process



Training process

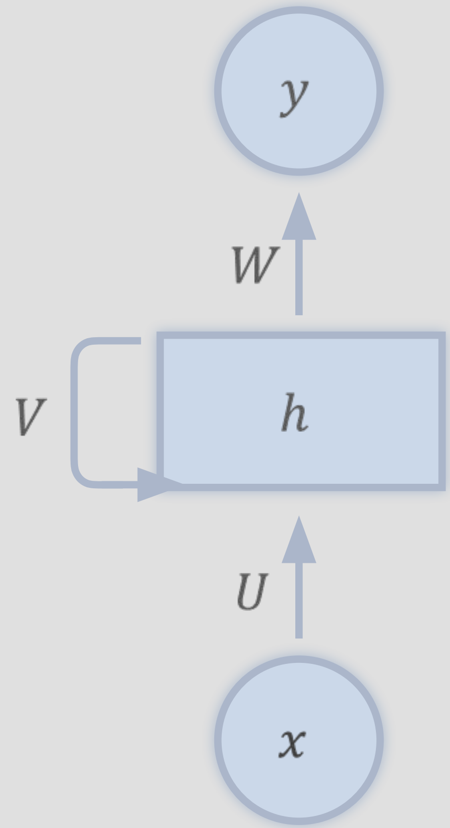
- Supplementary characters for the beginning and end of the sentence: <s>, </s>



Training process

- at each step calculate the loss function, for example, cross entropy

$$\text{loss}_{\text{local}} = CE(y_t, \hat{y_t}) = -\sum_{w \in V} y_t^w \log \widehat{y_t^w}$$

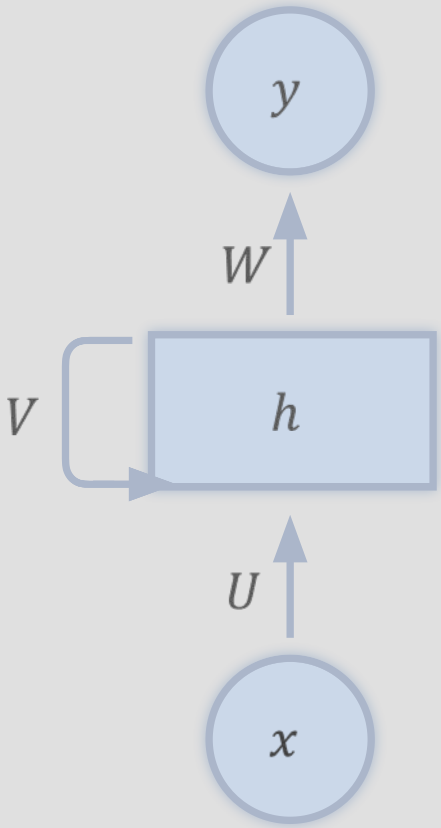


Training process

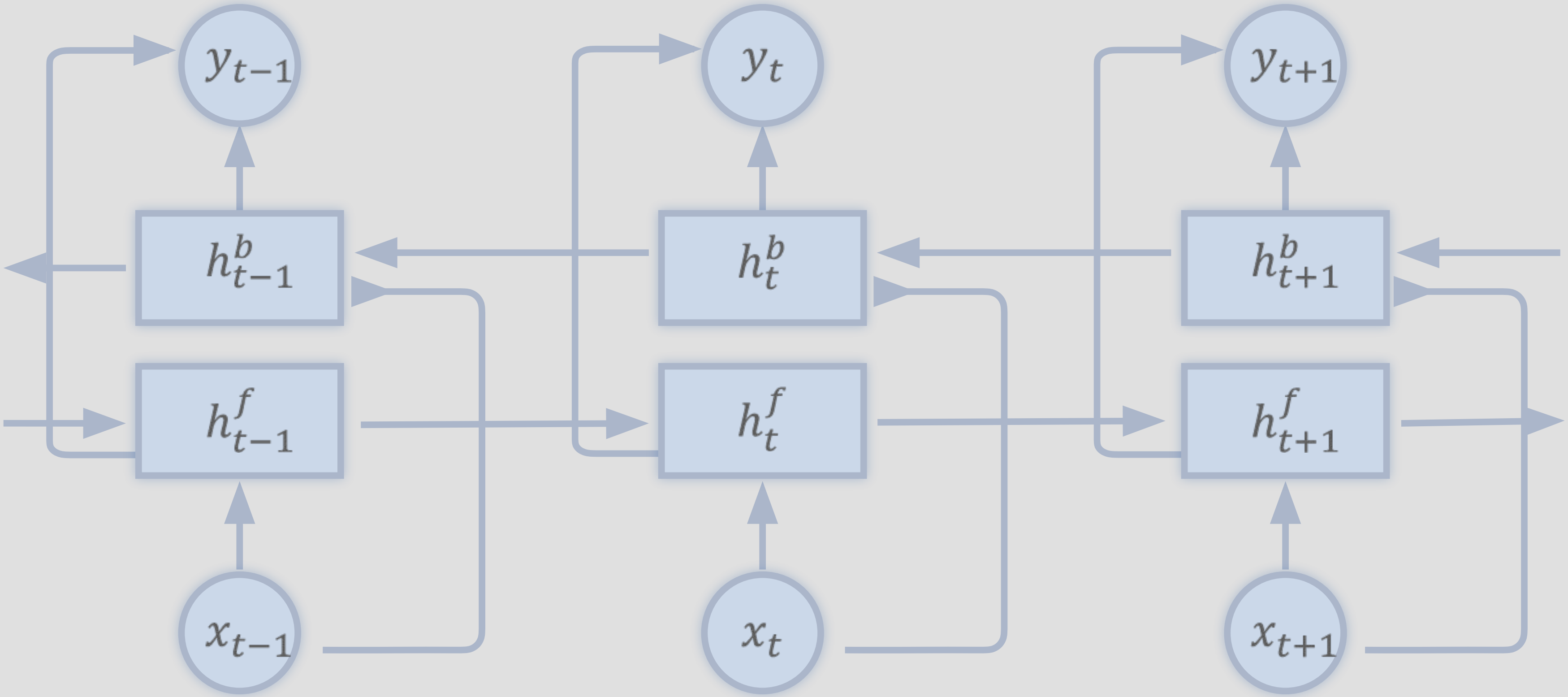
- at each step calculate the loss function, for example, cross entropy

$$\mathrm{loss}_{\mathrm{local}} = CE(y_t, \hat{y_t}) = -\sum_{w \in V} y_t^w \log \widehat{y_t^w}$$

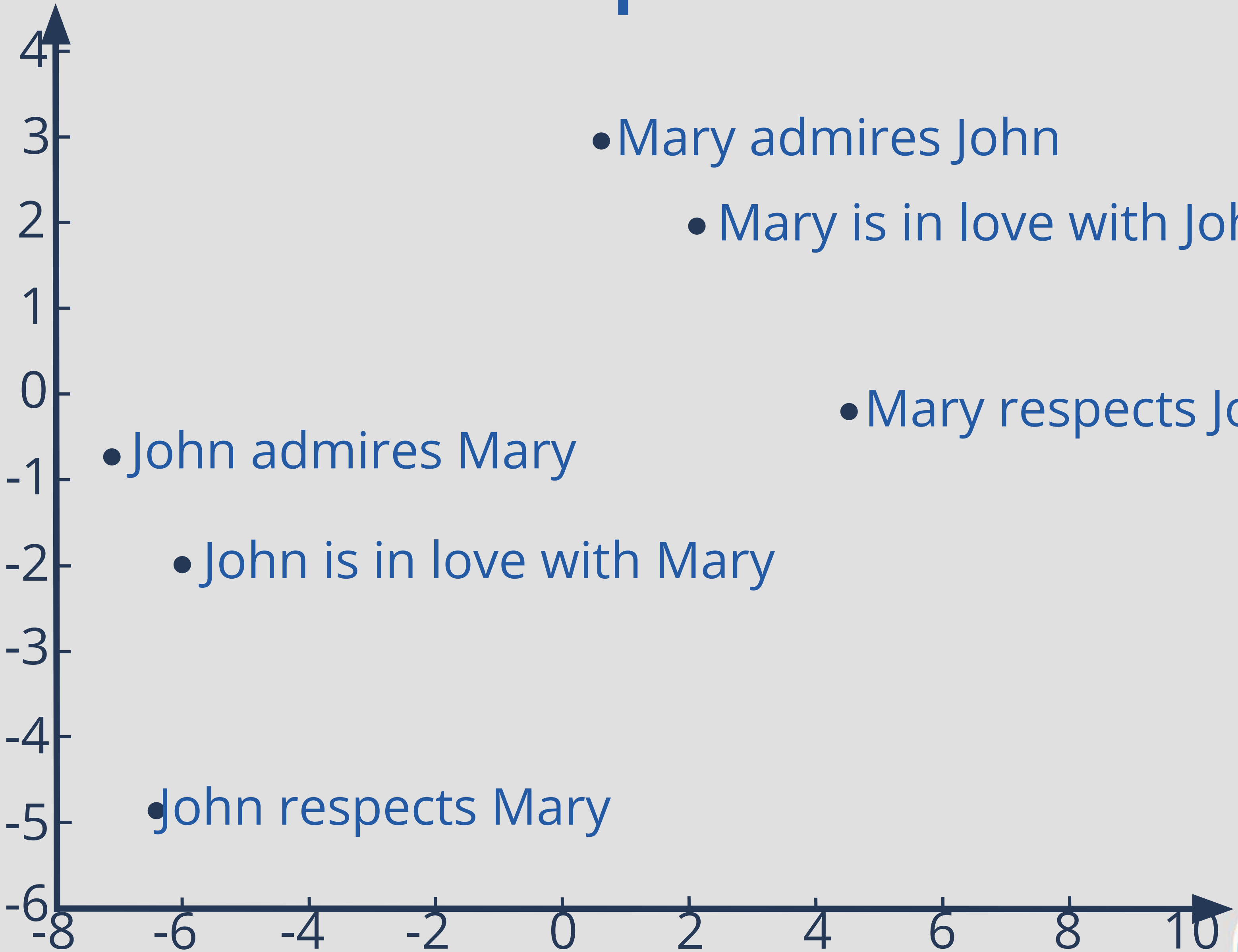
$$\mathrm{loss}_{\mathrm{global}} = \sum \mathrm{loss}_{\mathrm{local}}$$



Bidirectional RNN



Encoder final state representation



[Sequence to Sequence Learning with Neural Networks]() (NeurIPS'14)

Recurrent neural network

Advantages

- Works with input sequences of arbitrary length
- Has a state embedding that keeps information about processed input
- The number of parameters doesn’t depend on the corpus size

Recurrent neural network

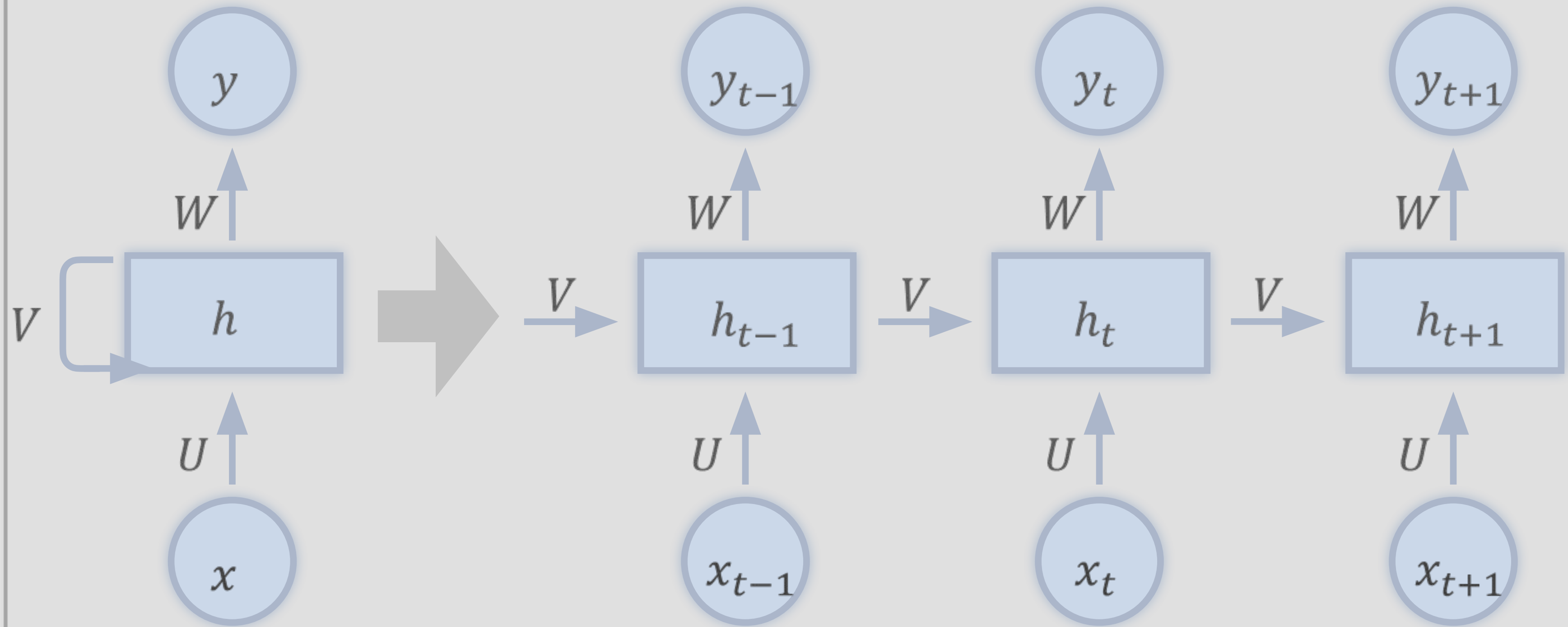
Advantages

- Works with input sequences of arbitrary length
- Has a state embedding that keeps information about processed input
- The number of parameters doesn’t depend on the corpus size

Disadvantages

- Slow computations
- Memory mechanism doesn’t work well

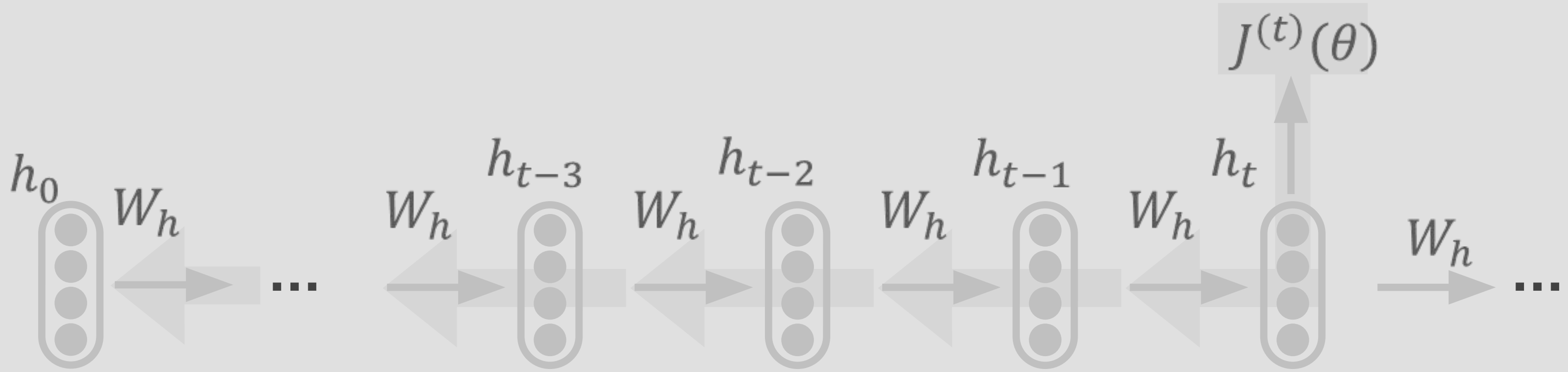
Memory mechanism



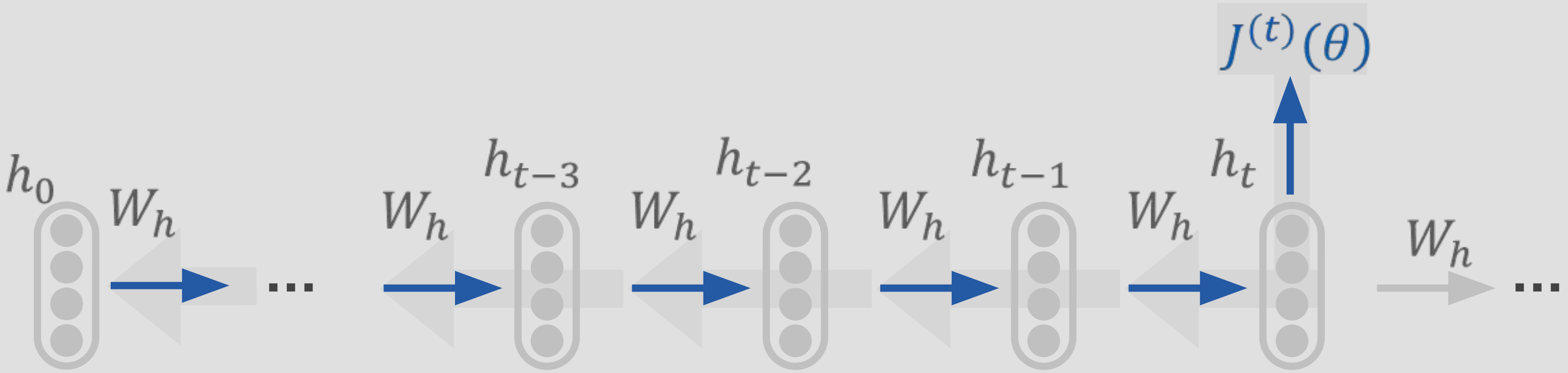
$$h_n = \sigma(h_{n-1}, x_n) = \sigma\left(\overbrace{\sigma(h_{n-2}, x_{n-1})}^{h_{n-1}}, x_n\right) = \sigma\left(\sigma\left(\overbrace{\sigma(h_{n-3}, x_{n-3})}^{h_{n-2}} x_{n-1}\right), x_n\right)$$

We update memory state at each step

Backpropogation through time mechanism

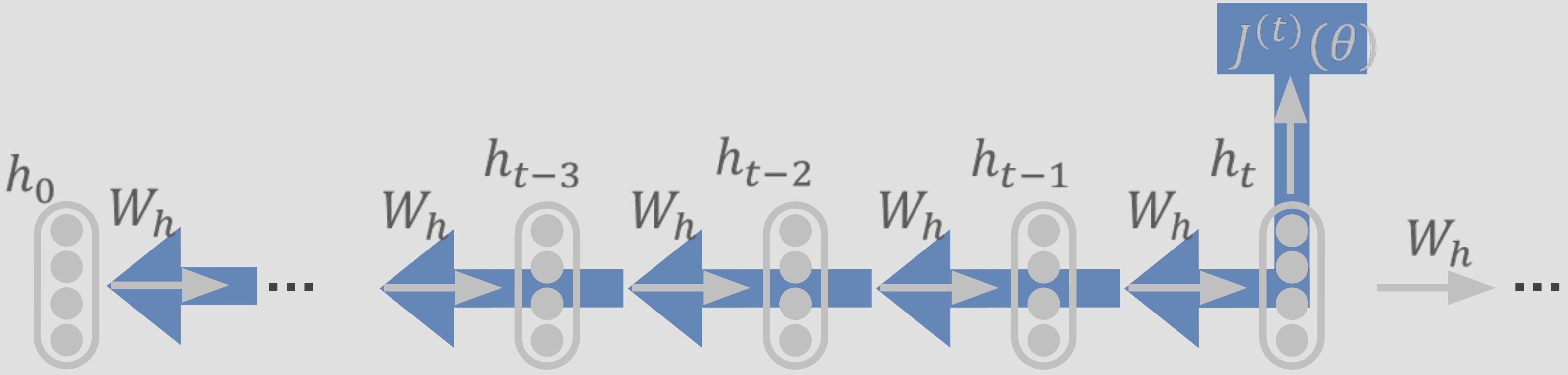


Backpropogation through time mechanism



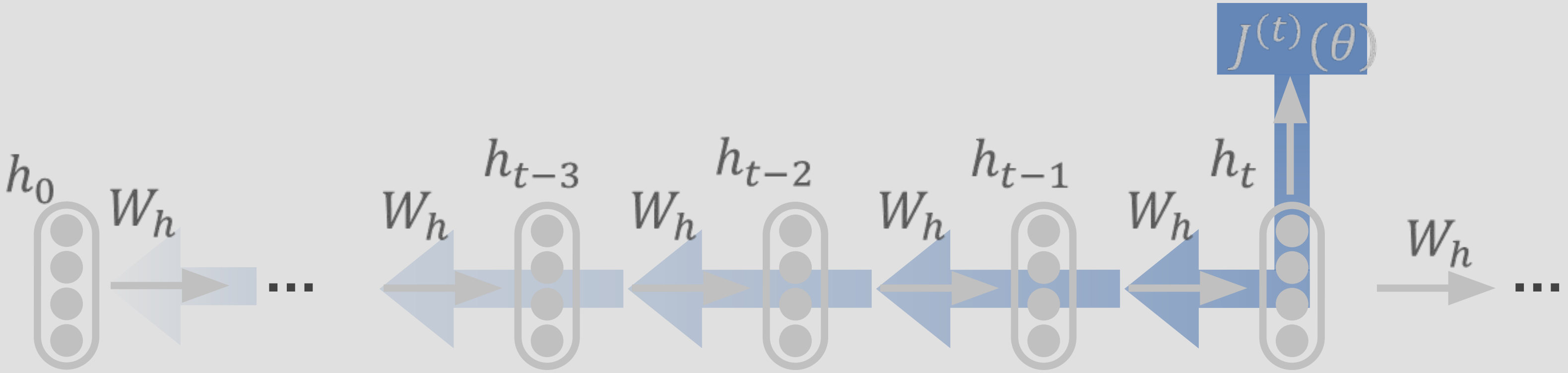
- calculate all local losses

Backpropogation through time mechanism



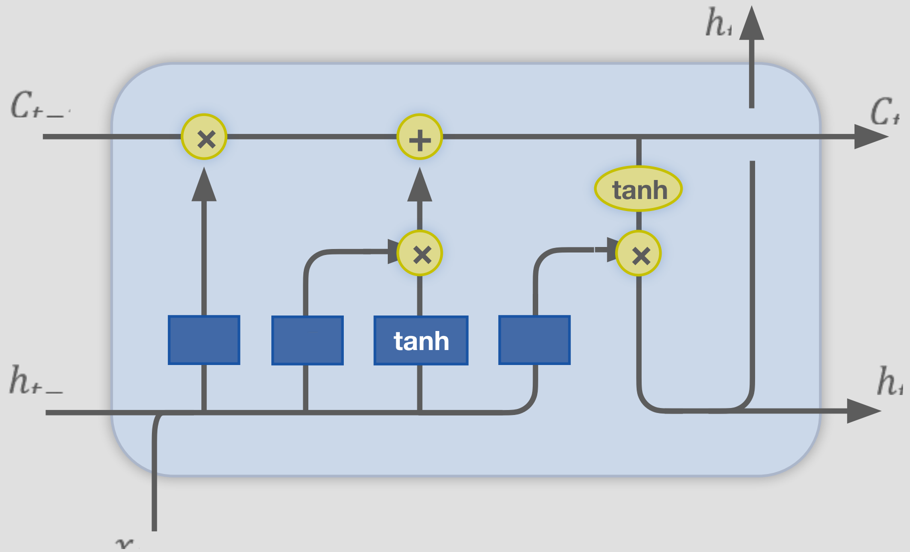
- calculate all local losses
- calculate the gradient from the last word down to the first one

Backpropogation through time mechanism

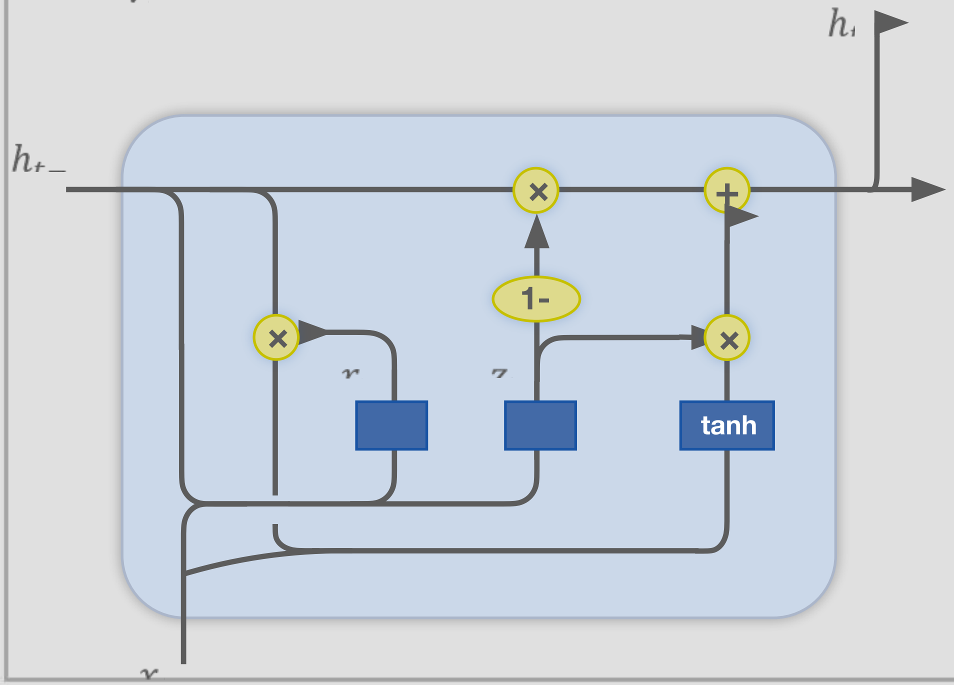


- calculate all local losses
- calculate the gradient from the last word down to the first one
- gradient vanishes due to the large amount of partial derivatives

Advanced RNN architectures

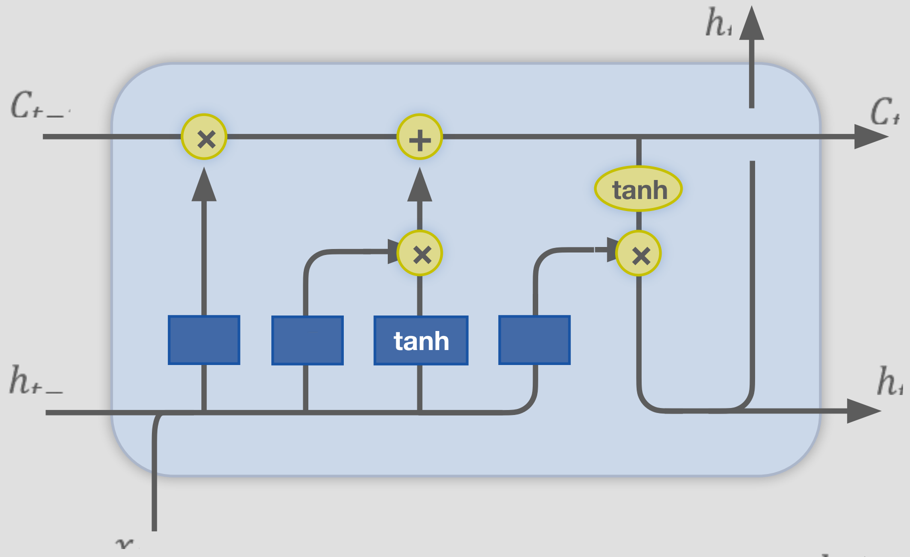


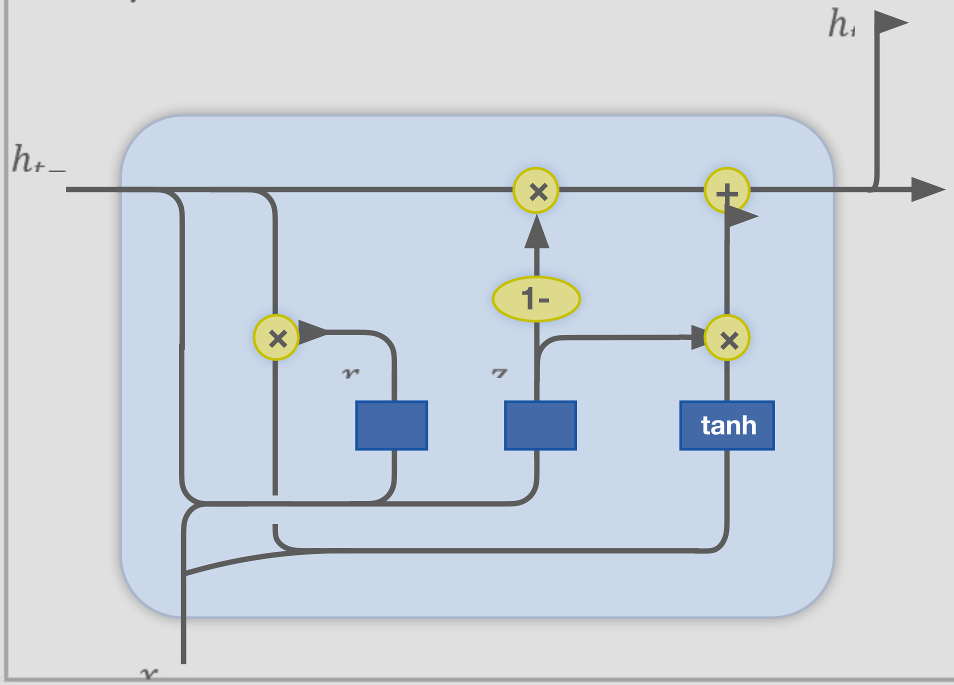
Long short term memory (LSTM)



Gated Recurrent Unit (GRU)

Advanced RNN architectures





RNN variations use additional masks for memory mechanism

Recurrent neural networks

- Architectures that handle sequences of derived length
- Can be used in a variety of text processing tasks, including text classification, text generation, and named entity recognition

Text generation

Problem statement

- given a context $x_1, \ldots, x_m$
  generate next n words $x_{m+1}, \ldots, x_{m+n}$
- generation strategy P defines which word will be generated on every step:

$$P(x_{1:m+n}) = \prod_{i=1}^{m+n} P(x_i | x_1, \ldots, x_{i-1})$$

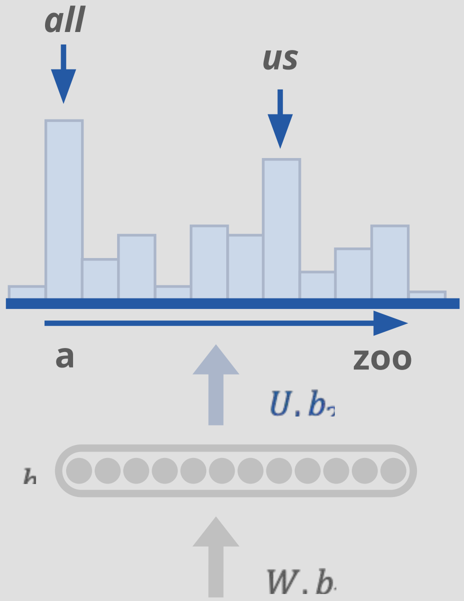
Problem statement

- given a context $x_1, \ldots, x_m$
  generate next n words $x_{m+1}, \ldots, x_{m+n}$
- generation strategy P defines which word will be generated on every step:

$$P(x_{1:m+n}) = \prod_{i=1}^{m+n} P(x_i | x_1, \ldots, x_{i-1})$$

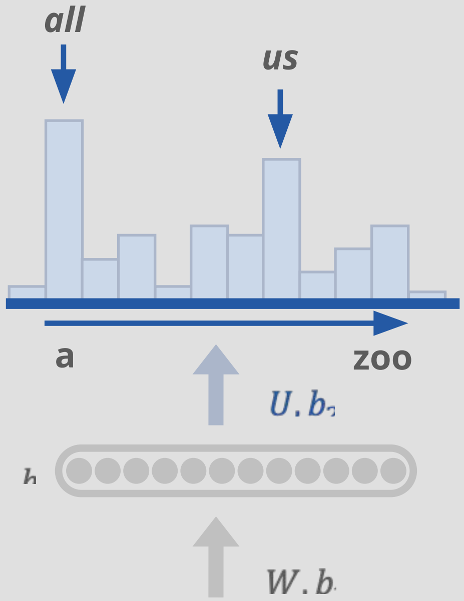
- estimation all possible combinations is too difficult

Generation strategies



softmax gives a probability distribution over words

Generation strategies



softmax gives a probability distribution over words

- greedy sampling
- beam search
- greedy sampling with temperature
- top-k sampling
- nucleus sampling

Decoding strategies

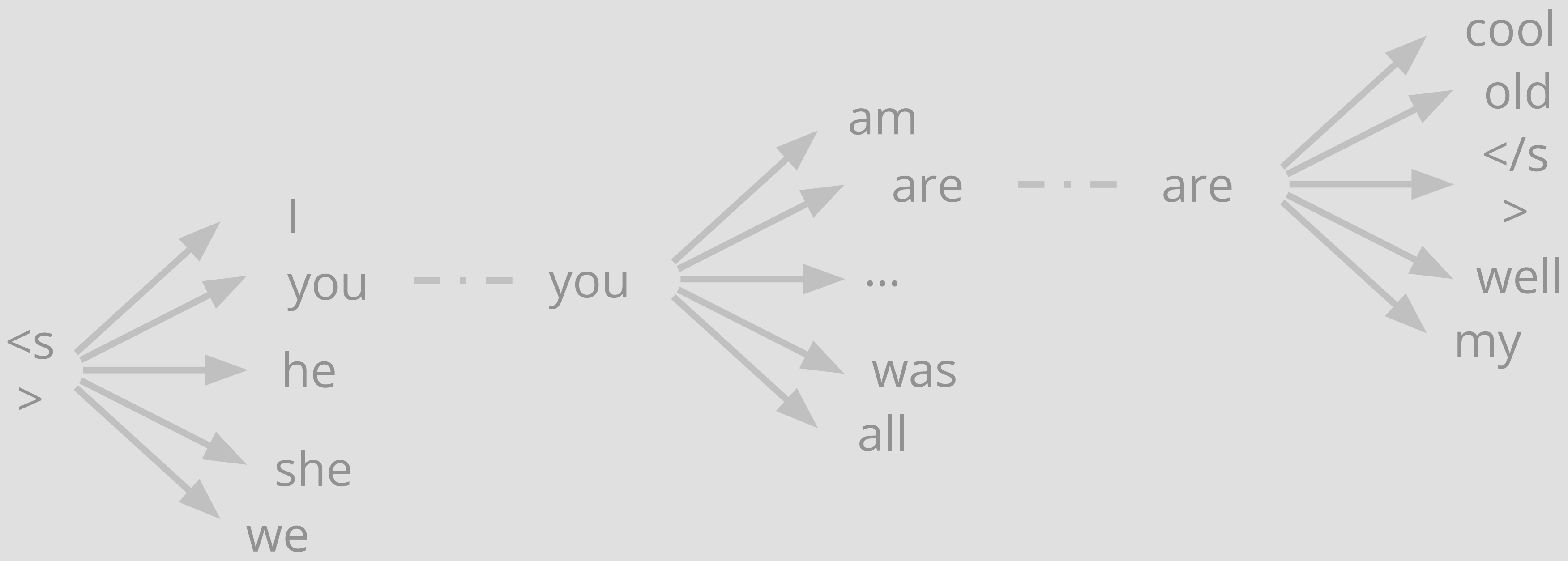
Search

- Deterministic algorithm
- Chooses the most likely words at each step

Generation

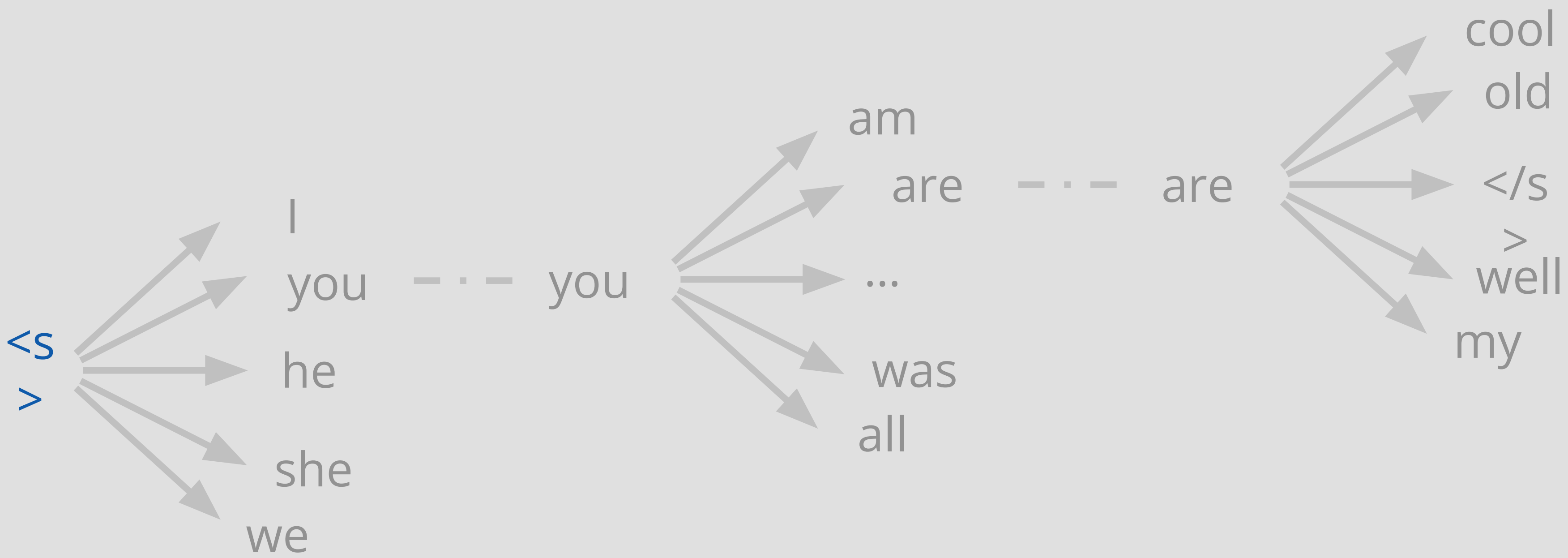
- Non-deterministic algorithms
- Generate from the distribution specified on the last layer

Greedy search



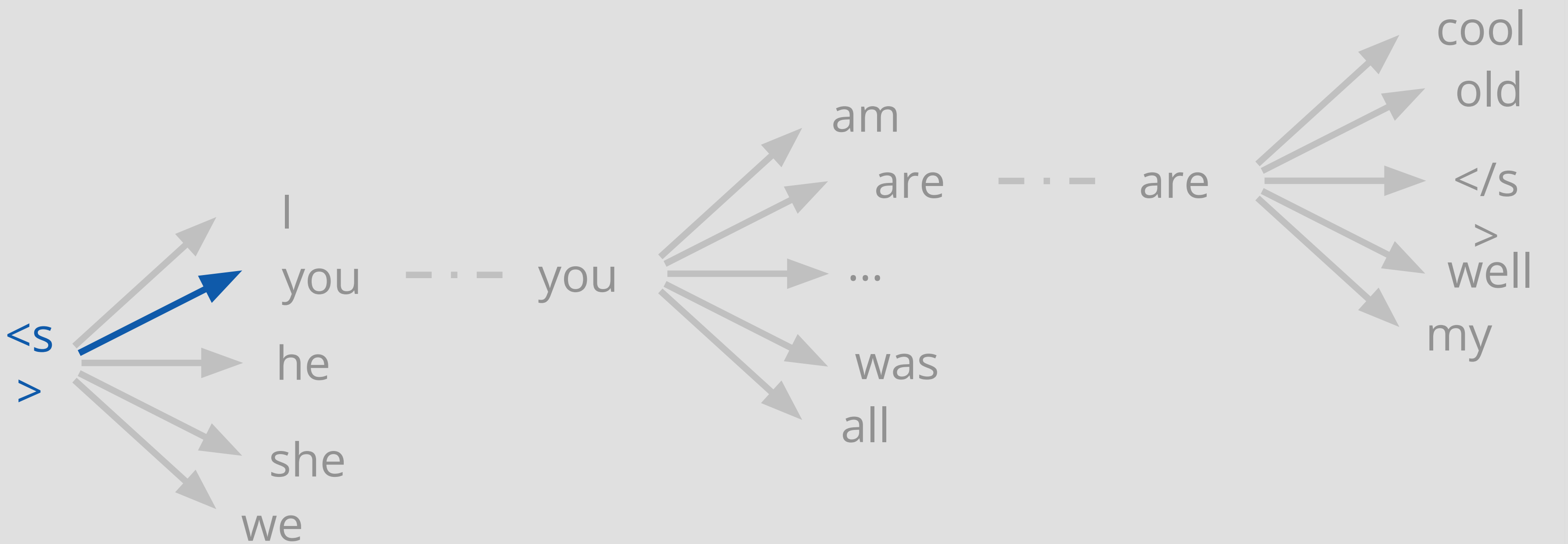
Greedy search

- Start with the beginning symbol <s>



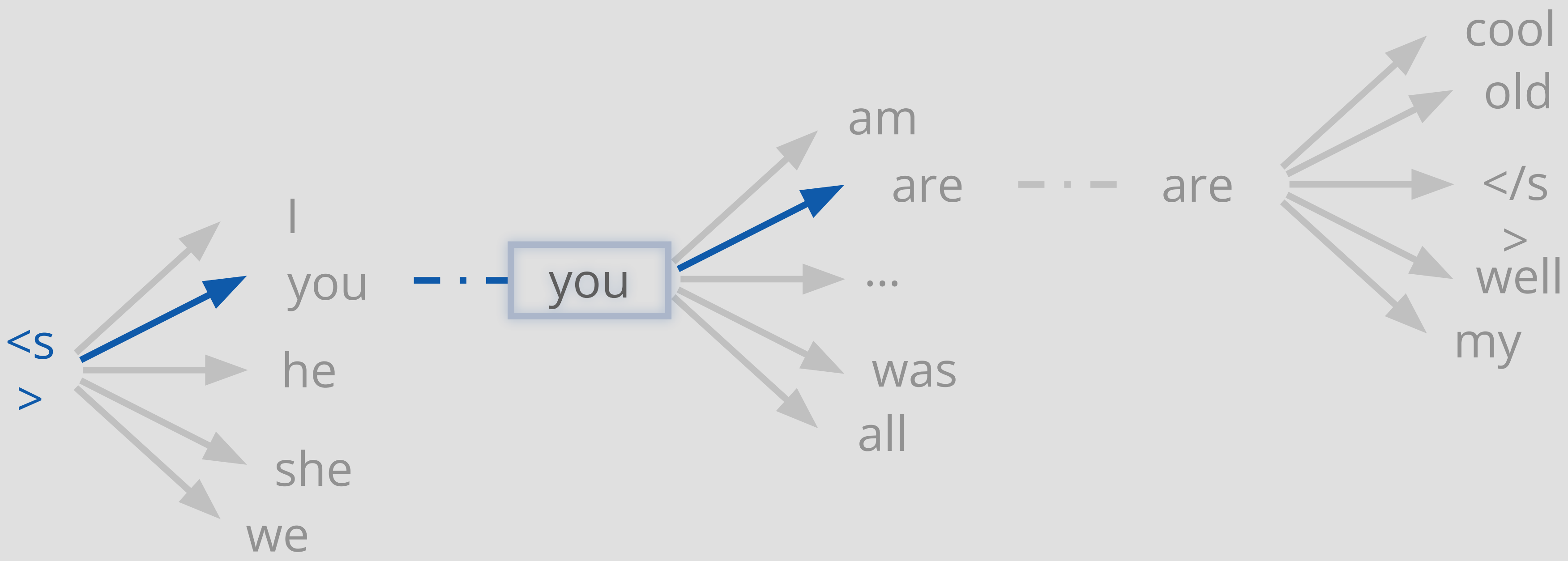
Greedy search

- Choose the most possible first word



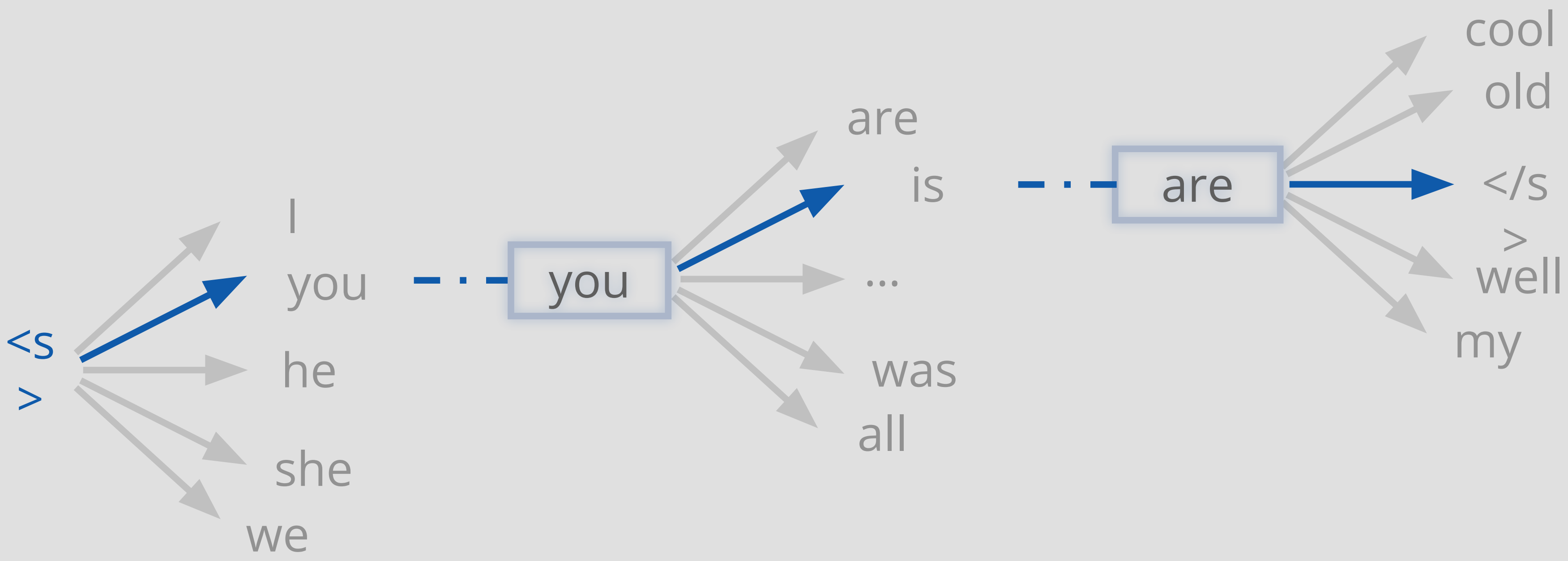
Greedy search

- With the first word as input generate the most probable second word

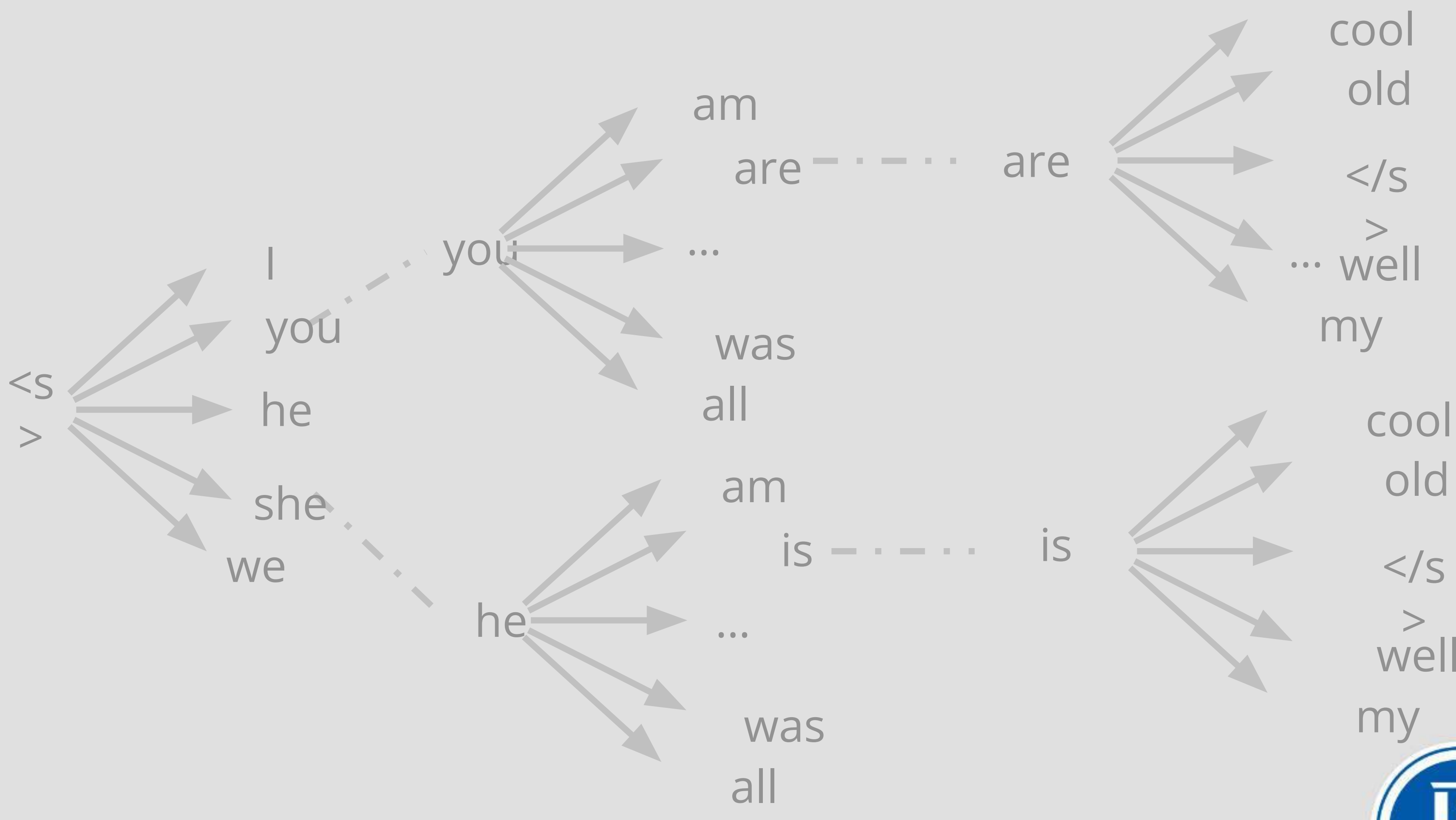


Greedy search

- generate until we get the </s> symbol



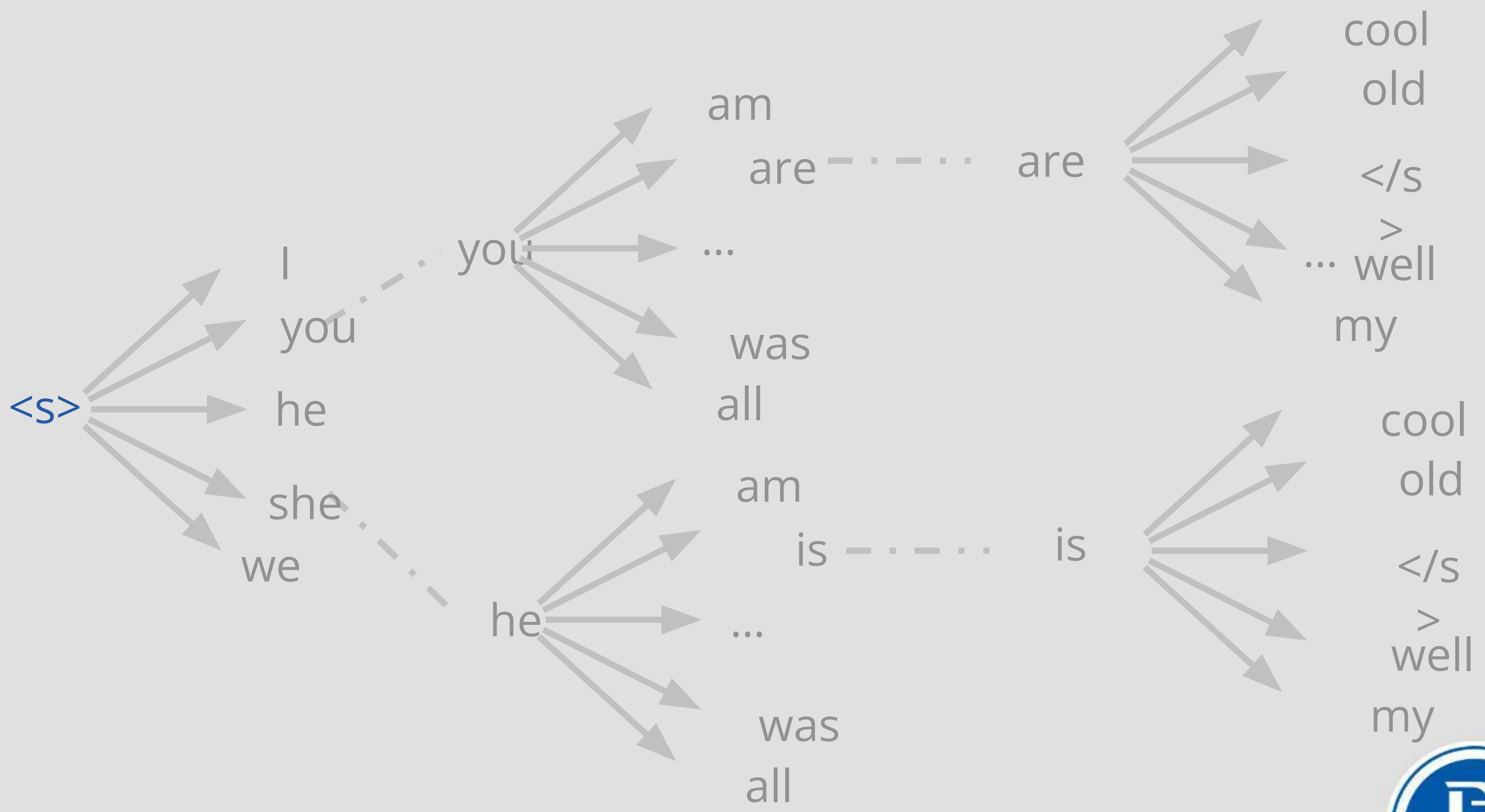
Beam search



<s>
I
you
he
she
we
you
am
are
...
was
all
he
am
is
...
was
all
are
cool
old
</s>
...
well
my
is
cool
old
</s>
well
my

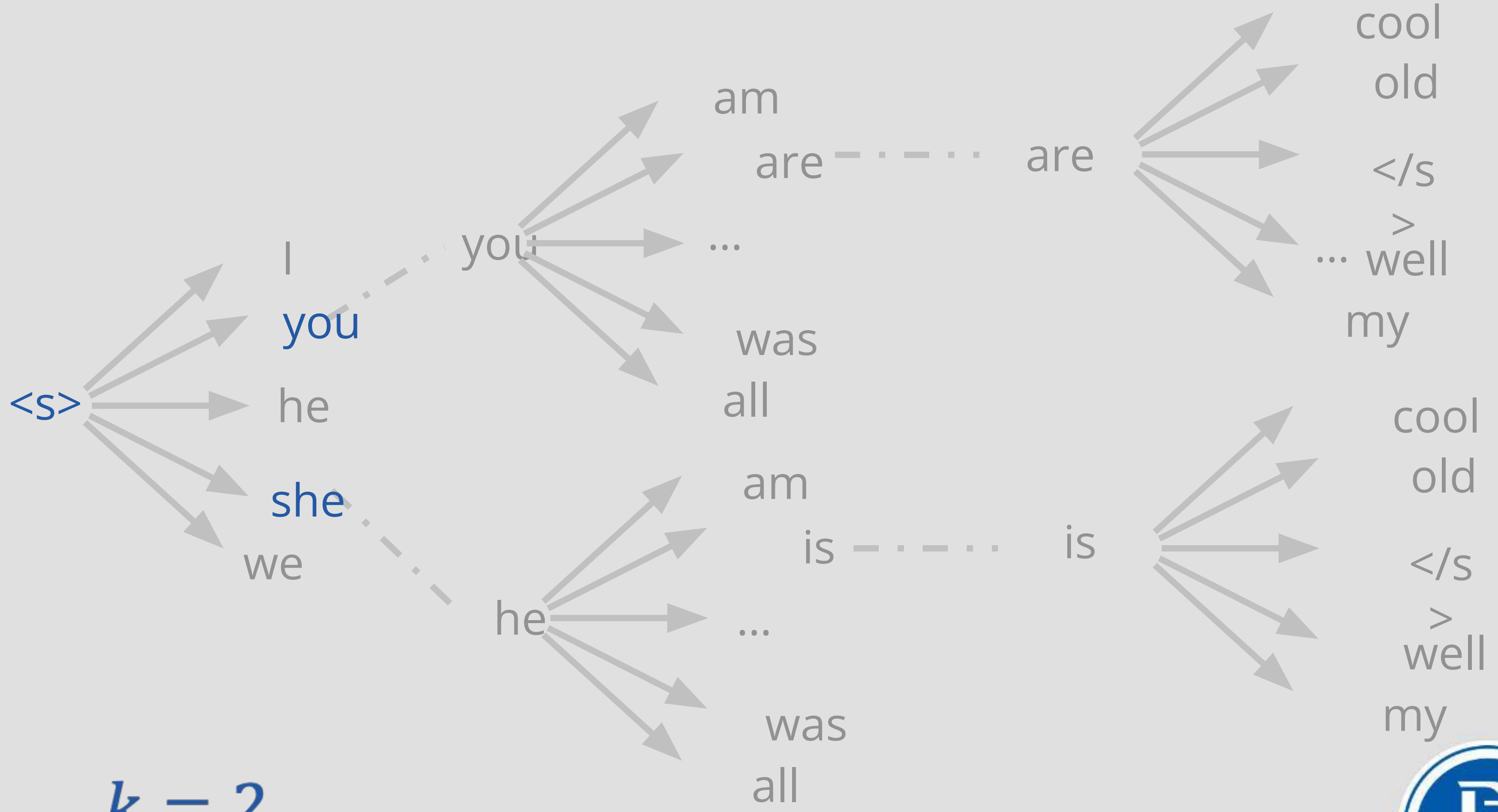
Beam search

- Start with the beginning symbol <s>



Beam search

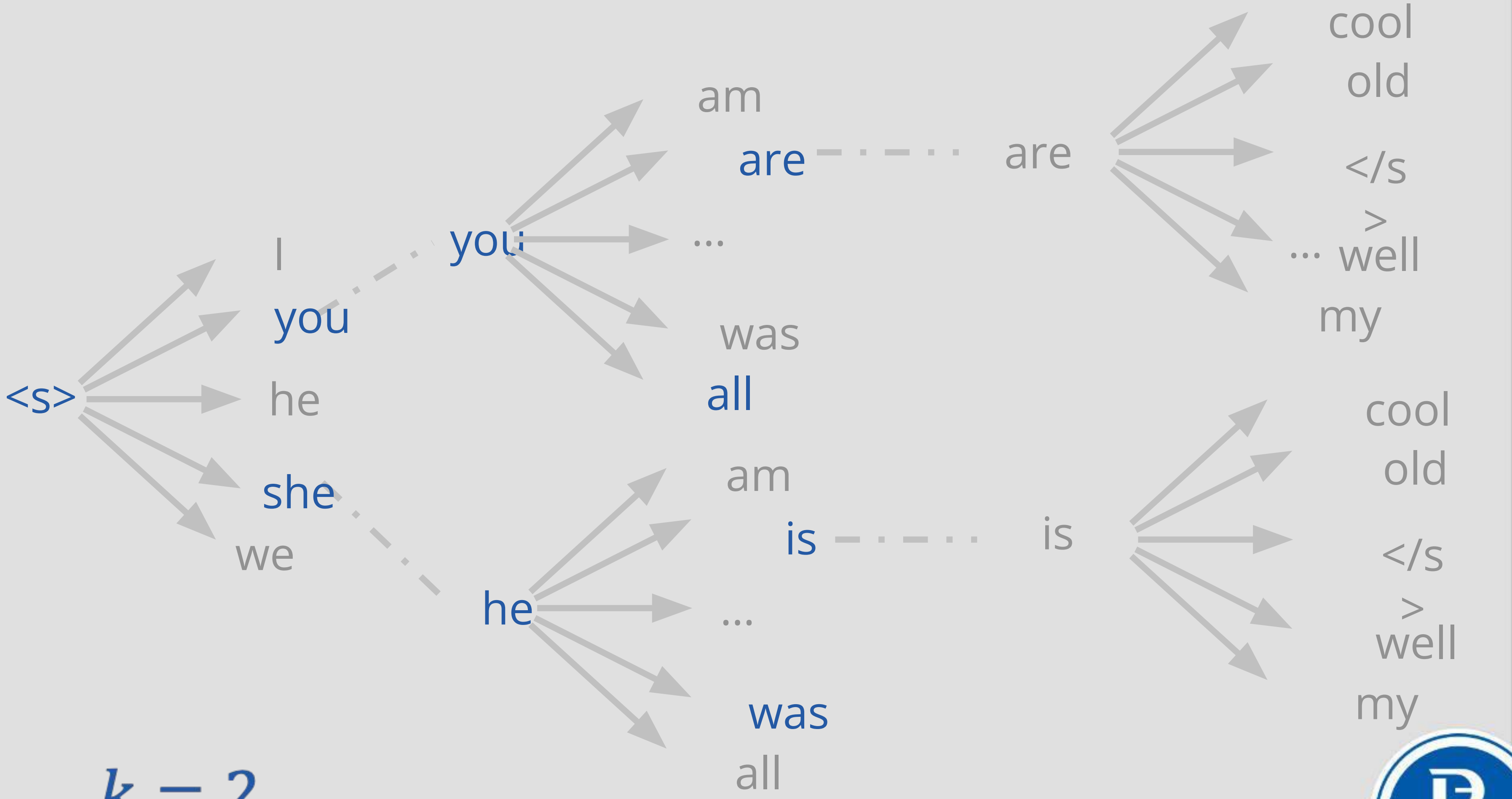
- Choose k most probable words



$k = 2$

Beam search

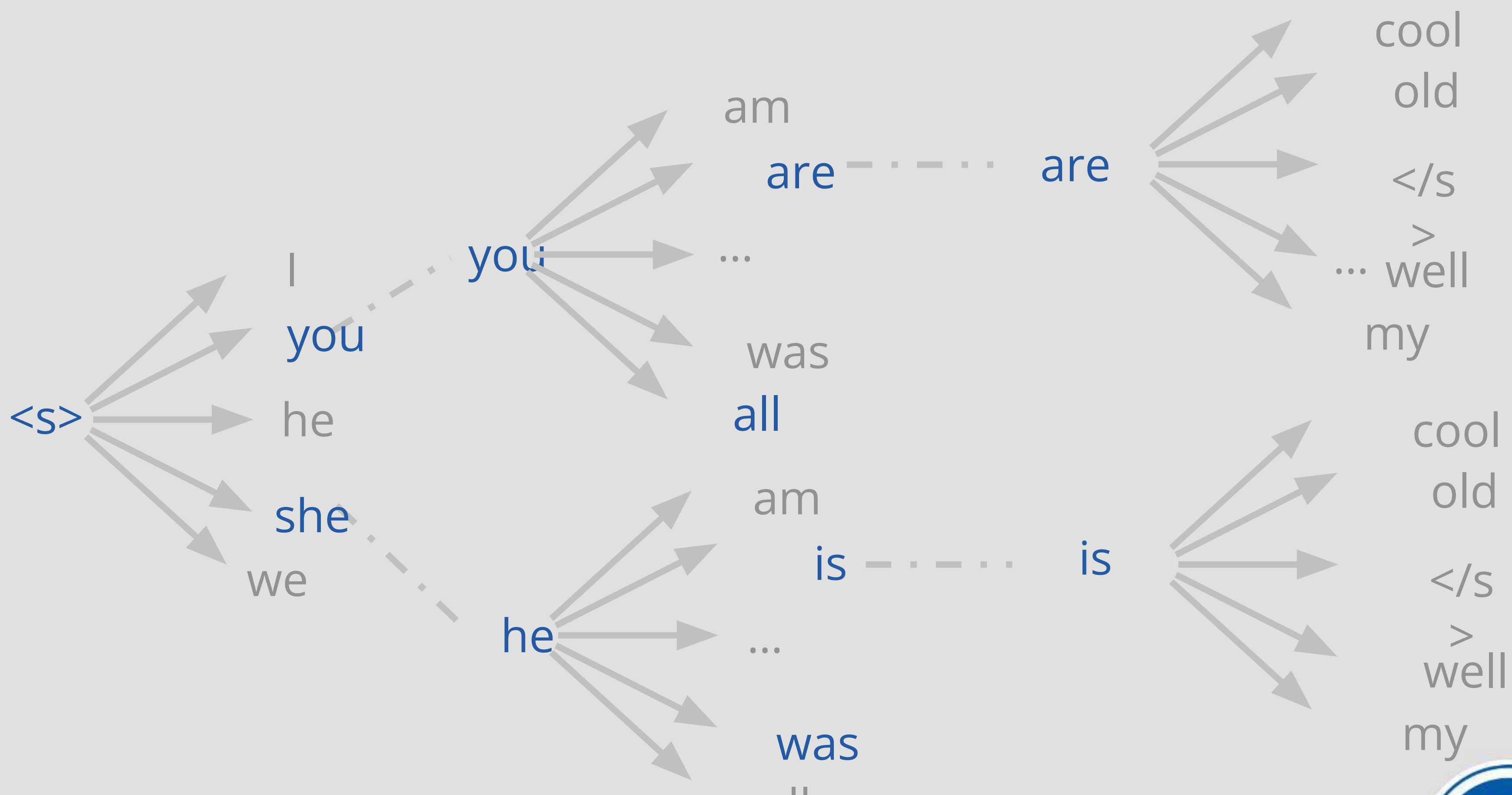
- For each word choose k next most probable words



$k = 2$

Beam search

- For each k*k options choose k most probable



$k = 2$

Greedy search VS beam search

Greedy search

Beam search

Greedy search VS beam search

Greedy search

- deterministic
- quick
- chooses an optimal word at each step
- doesn’t give diversity

Beam search

Greedy search VS beam search

Greedy search

- deterministic
- quick
- chooses an optimal word at each step
- doesn’t give diversity

Beam search

- deterministic
- equivalent to greedy search at k =1
- difficult to compute
- gives more diversity
- generated replics can be too general