

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 Sochi »

- Choose a more likely sequence:

$P(\text{«it's sunny in Sochi »}) > P(\text{«it's sunny in Siberia »})$



Example: search hints



🔍 language model — Поиск в «Google»

Предложения Google

- 🔍 language model
- 🔍 language models are few-shot learners
- 🔍 language models are unsupervised multitask learners
- 🔍 language modeling
- 🔍 language model python
- 🔍 language models are open knowledge graphs



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 $W = (w_1, w_2, \dots, w_n)$ - sentence,
 w_i - a word, V - vocabulary
- Probability of sentence W :
 - $P(W) = P(w_1, w_2, \dots, w_n)$



Probabilistic language model

- Formal task:
 - suppose $W = (w_1, w_2, \dots, w_n)$ - sentence,
 w_i - a word, V - vocabulary
- Probability of sentence W :
 - $P(W) = P(w_1, w_2, \dots, w_n)$
- Probability of the next word in sequence:
 - $P(w_n | w_{n-1}, w_{n-2}, \dots, w_1)$



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:

$$\begin{aligned} P(W) &= \\ &= P(w_1, w_2, \dots, w_n) = \end{aligned}$$



How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$

$$= P(w_1, w_2, \dots, w_n) =$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\dots P(w_n|w_1, \dots, w_{n-1})$$



How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$

$$= P(w_1, w_2, \dots, w_n) =$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\dots P(w_n|w_1, \dots, w_{n-1})$$

P ("it is sunny today") =



How to estimate the probability of a sentence?

- Expand the word sequence:

$$P(W) =$$

$$= P(w_1, w_2, \dots, w_n) =$$

$$= P(w_1)P(w_2|w_1)P(w_3|w_1w_2)\dots P(w_n|w_1, \dots, w_{n-1})$$

$$P(\text{"it is sunny today"}) = P(\text{"it"}) \cdot P(\text{"is" | "it"}) \cdot P(\text{"sunny" | "it is"}) \\ \cdot P(\text{"today" | "it is sunny"})$$



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:

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



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

$$P(\text{"it is sunny today"}) = P(\text{"it"}) \cdot P(\text{"is"}) \cdot P(\text{"sunny"}) \cdot P(\text{"today"})$$



Markov model

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

unigram model

$$P(\text{"it is sunny today"}) = P(\text{"it"}) \cdot P(\text{"is"}) \cdot P(\text{"sunny"}) \cdot P(\text{"today"})$$

bigram model

$$P(\text{"it is sunny today"}) = P(\text{"it"}) \cdot P(\text{"is" | it}) \cdot P(\text{"sunny" | is}) \cdot P(\text{"today" | sunny})$$



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 $P(W) = 0$

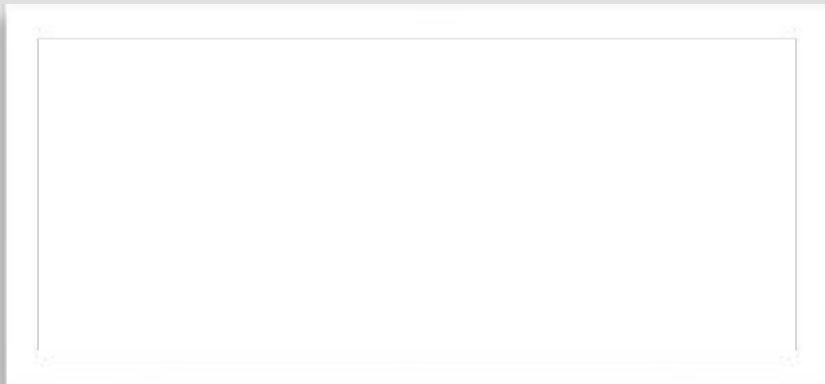
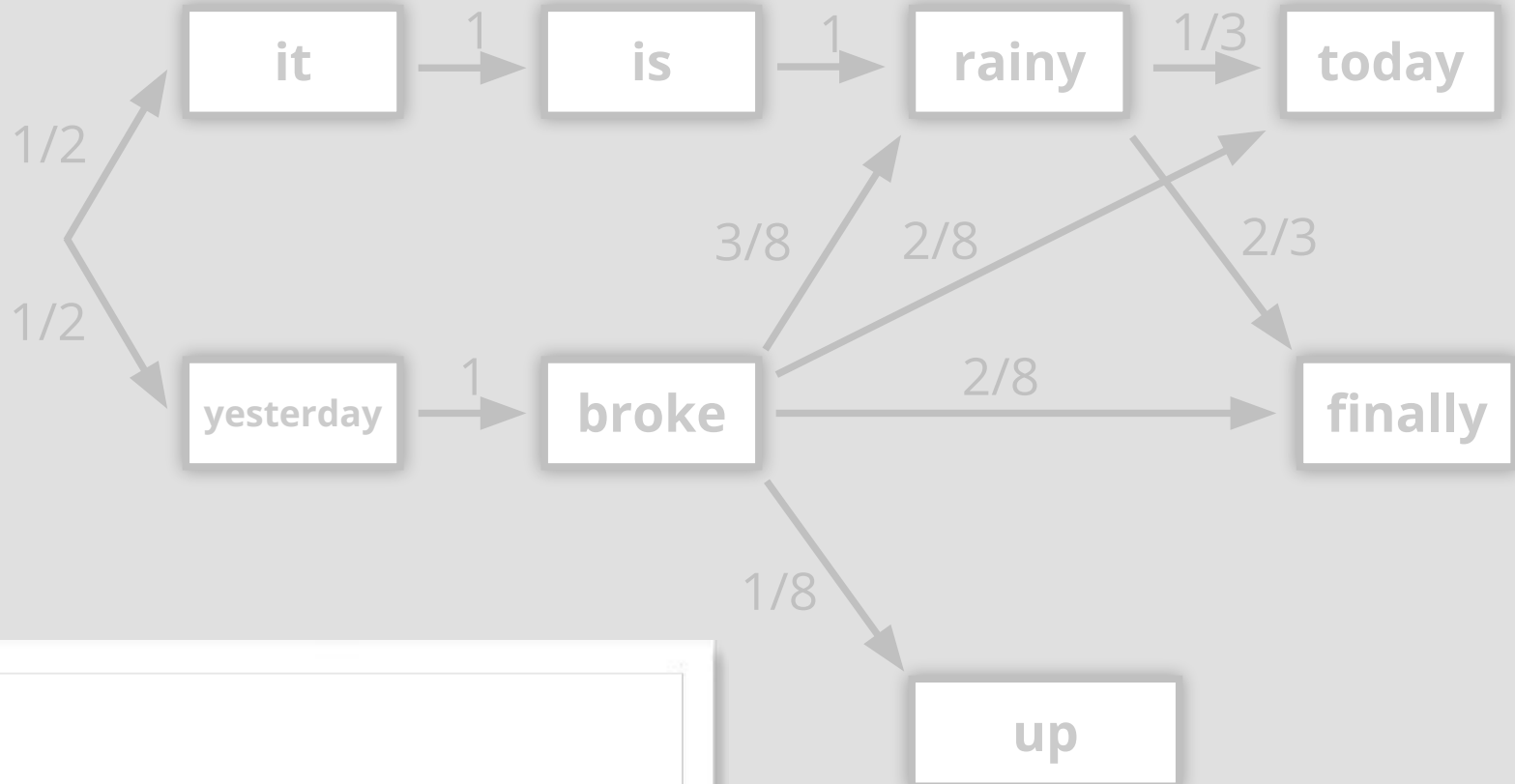
Solution: smoothing

- suppose each word occurs in text at least α times

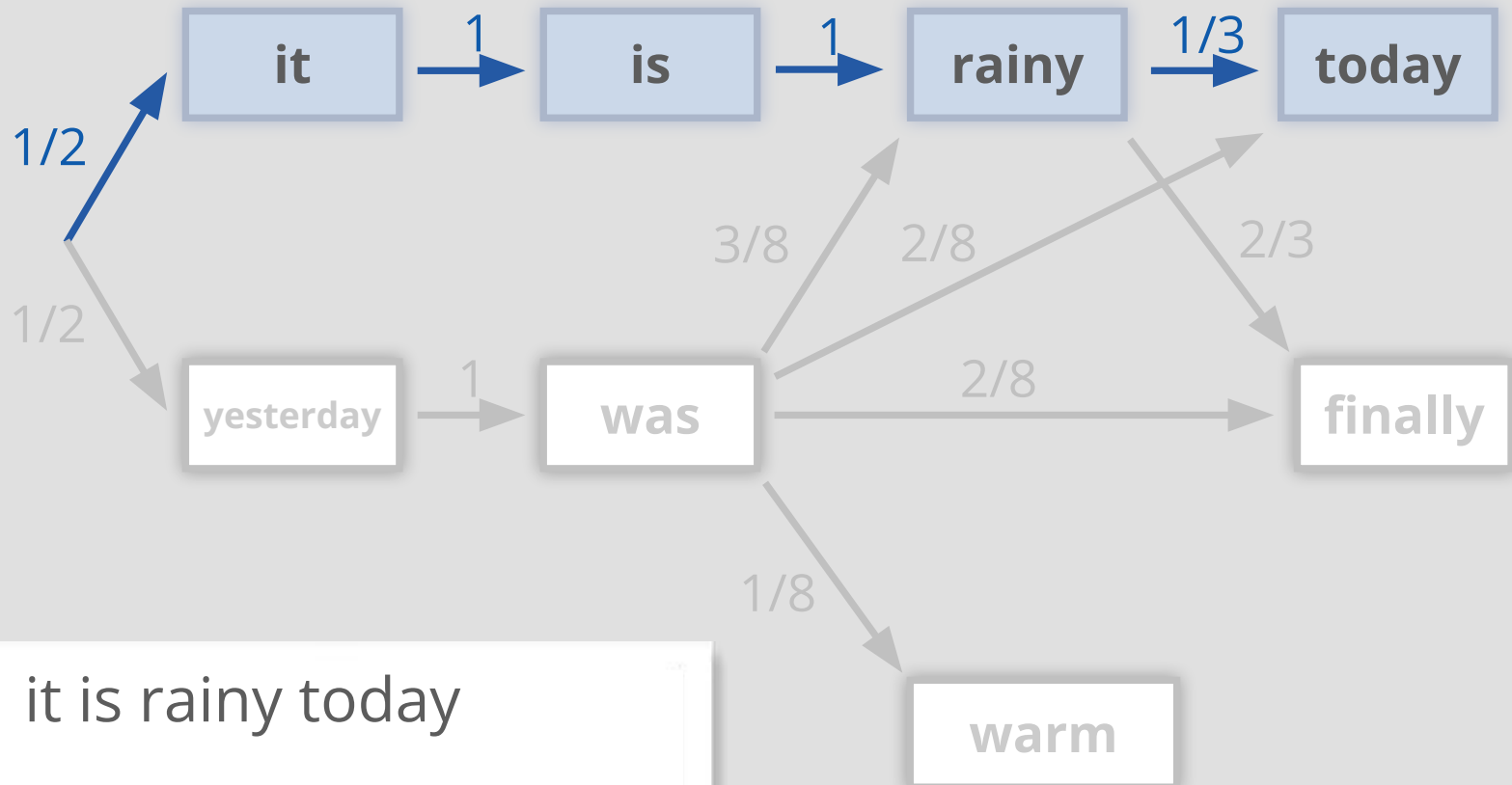
Then $\frac{\text{count}(w_{i-1}w_i) + \alpha}{\text{count}(w_{i-1}) + \alpha|V|}$



Text generation



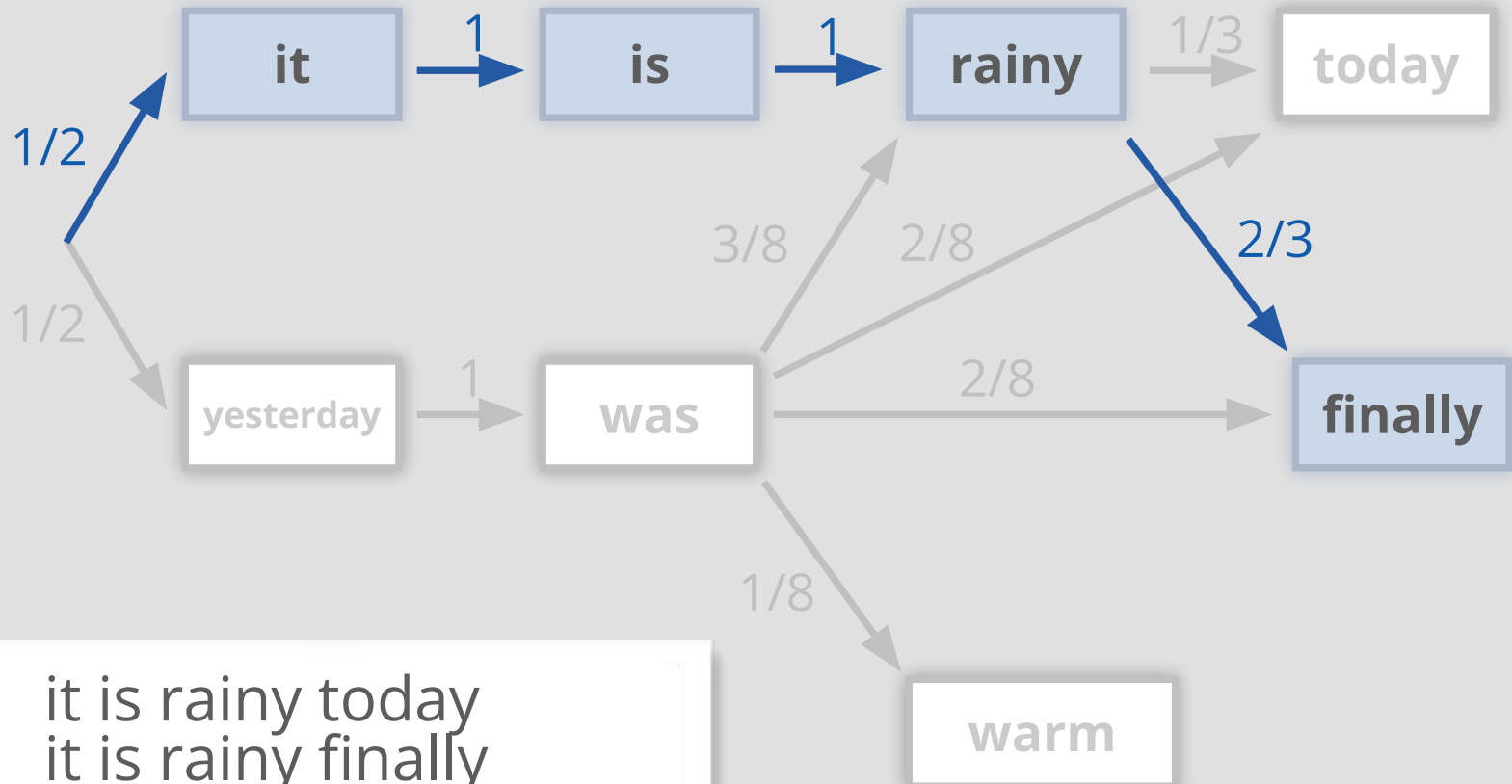
Text generation



- it is rainy today



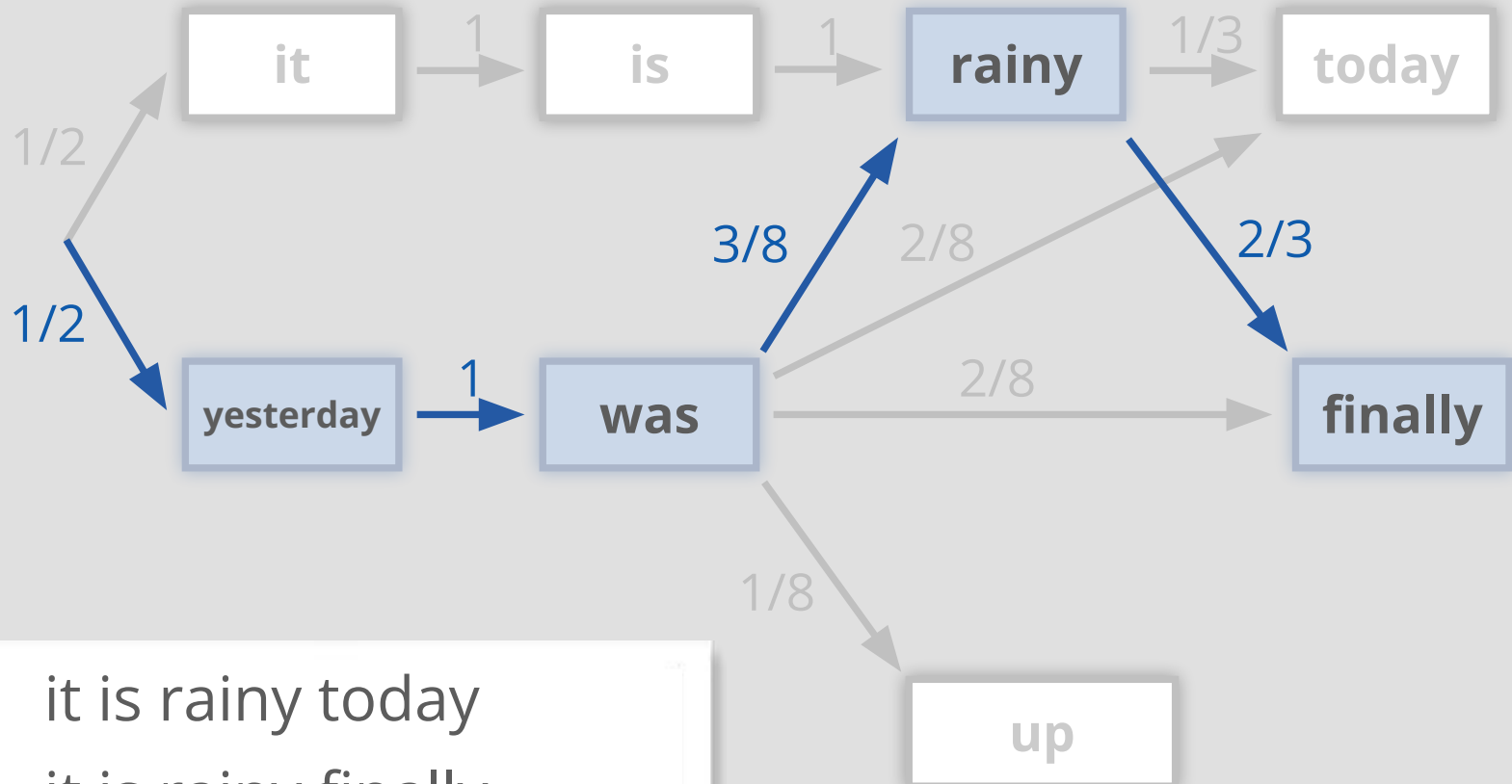
Text generation



- it is rainy today
- it is rainy finally



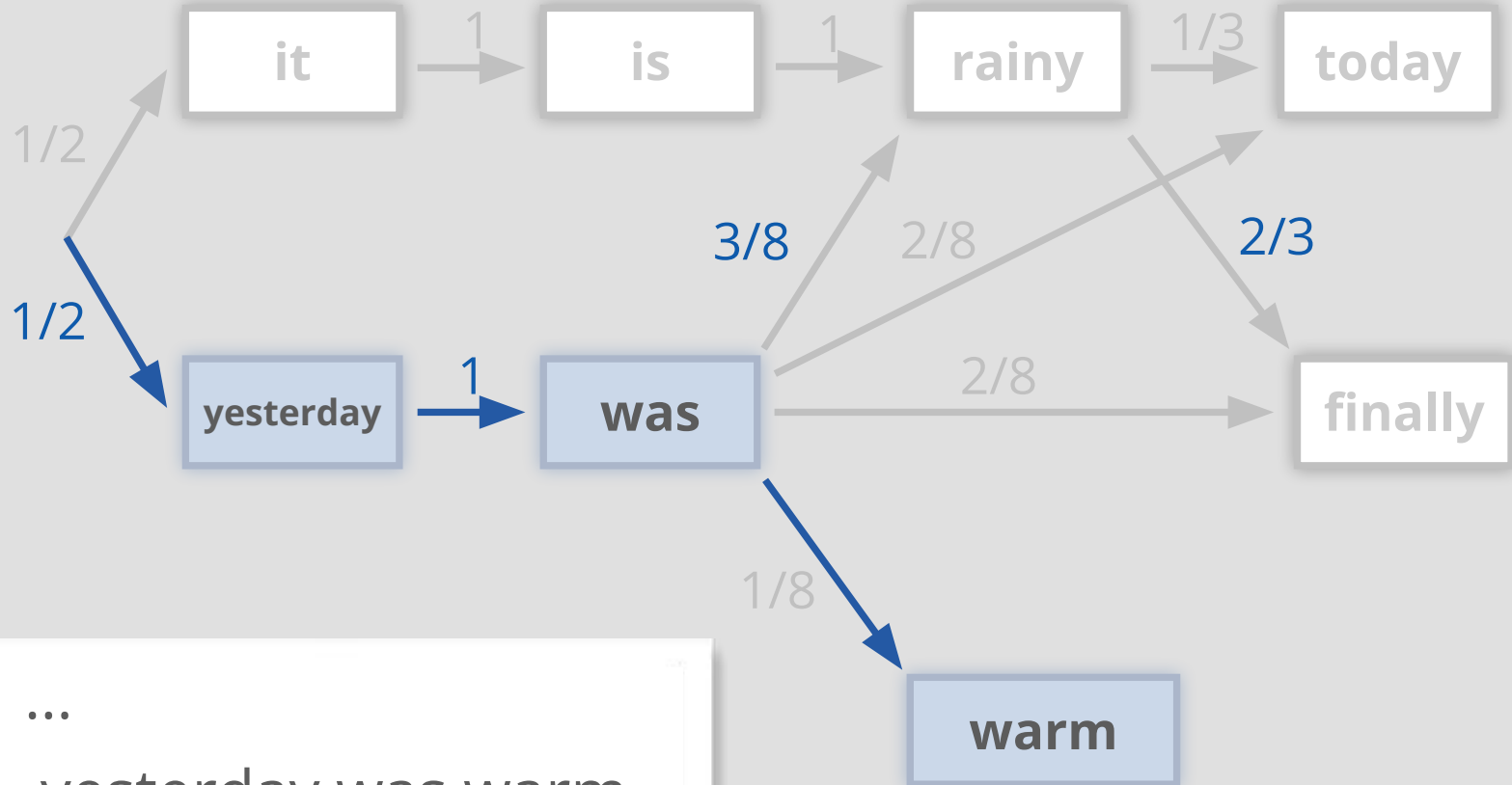
Text generation



- 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

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



Language models quality

- Internal quality evaluation: perplexity

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

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



Language models quality

- Internal quality evaluation: perplexity

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

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

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



Language models on counters

- Языковые модели моделируют вероятность последовательности слов



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, \dots, 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, \dots, 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, \dots, w_{n-1}
- Output: w_n

«love is all around _____»

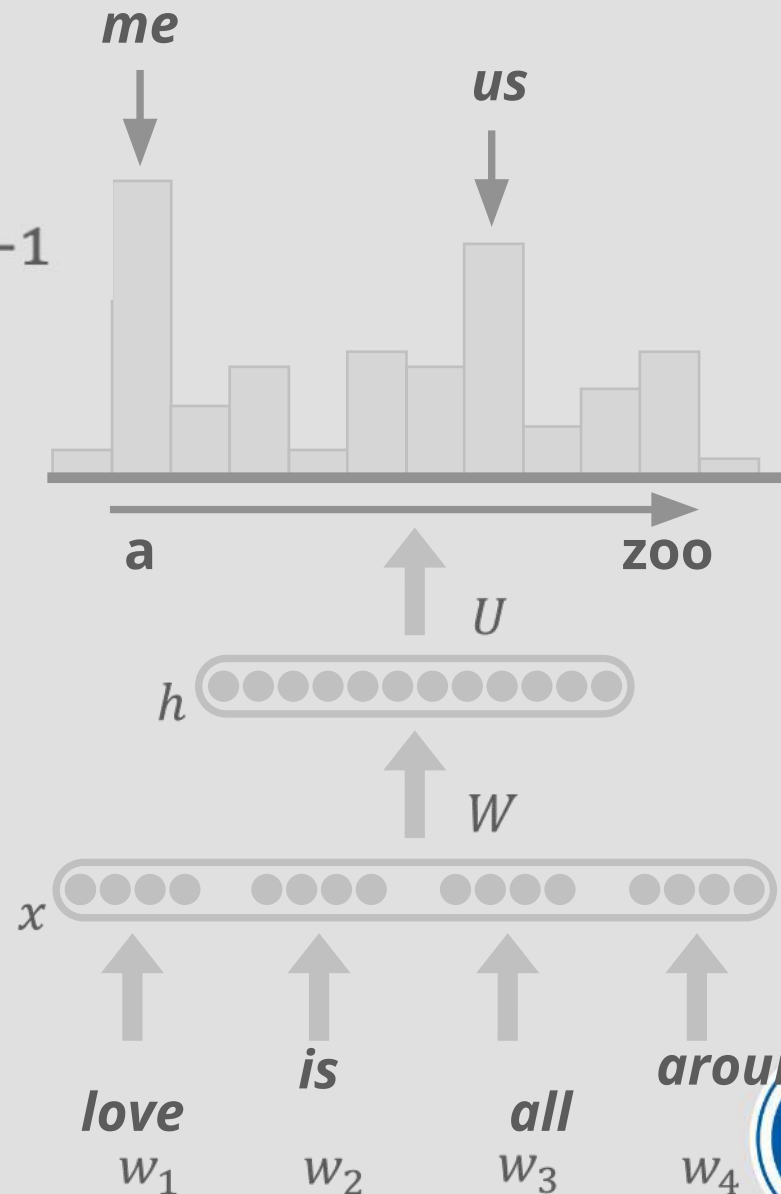
Input: «love is all around»

Output: «me»



Language model based on neural networks

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



«love is all around _____»

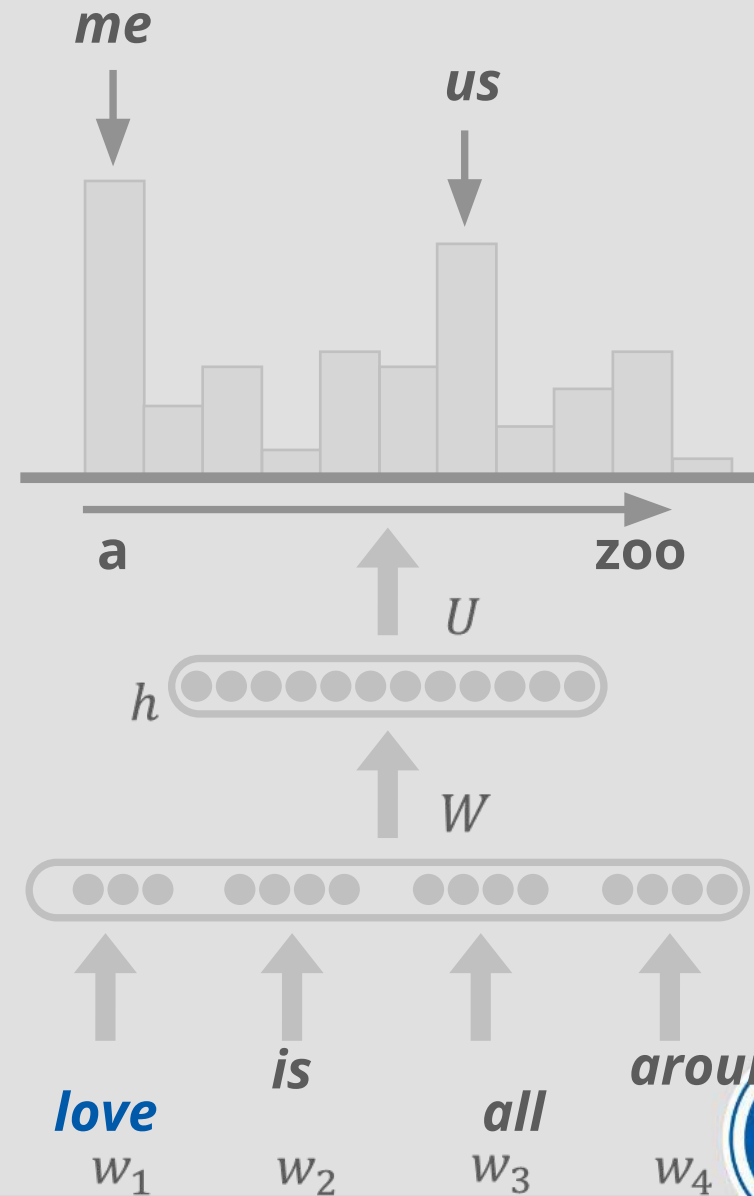
Input: «love is all around»

Output: «me»



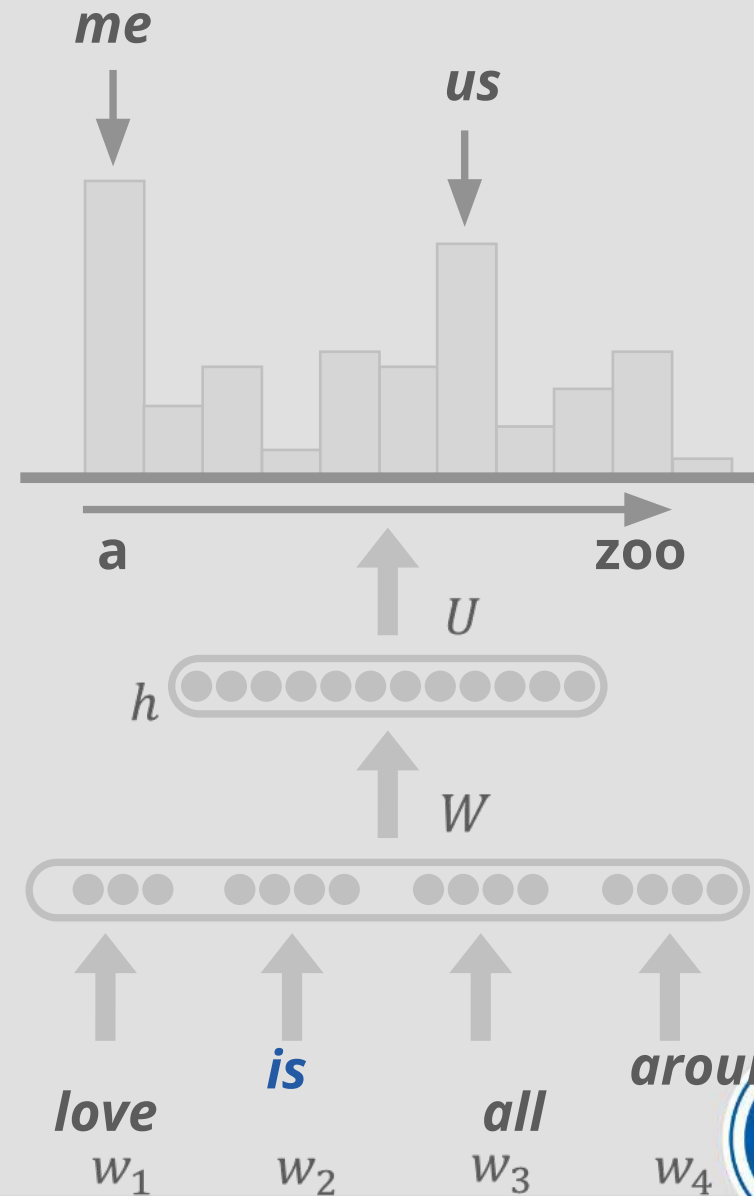
Language model based on neural networks

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



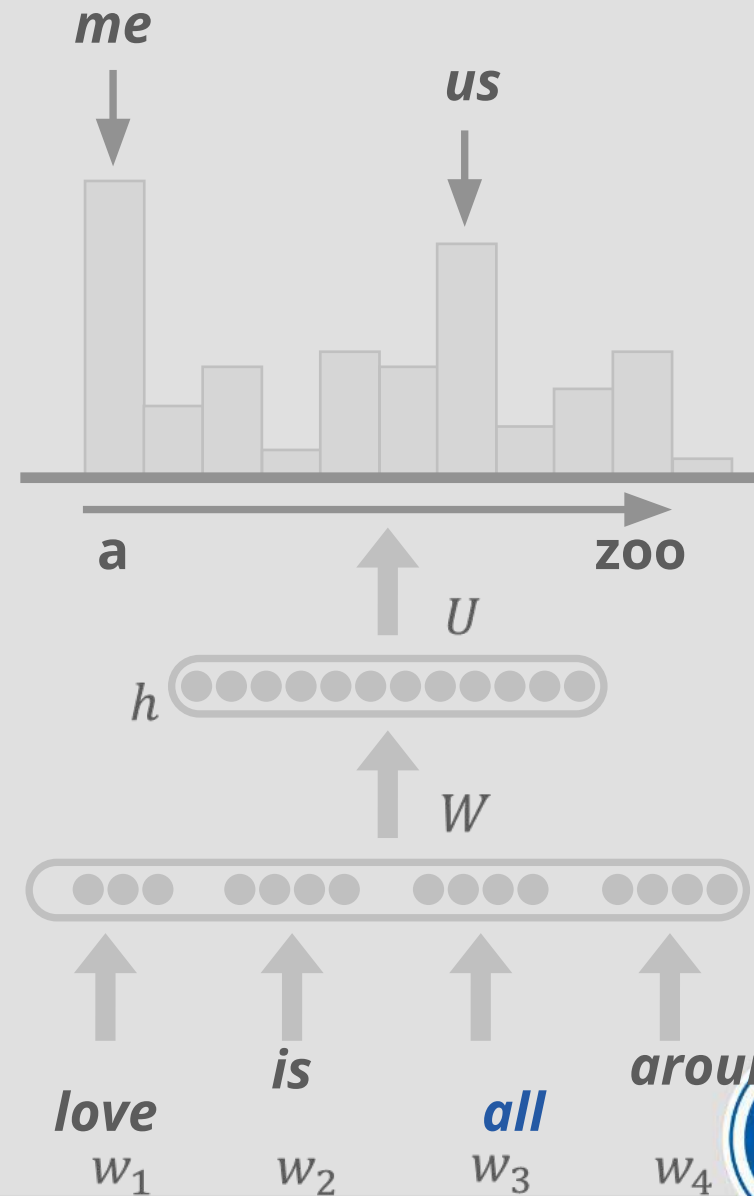
Language model based on neural networks

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



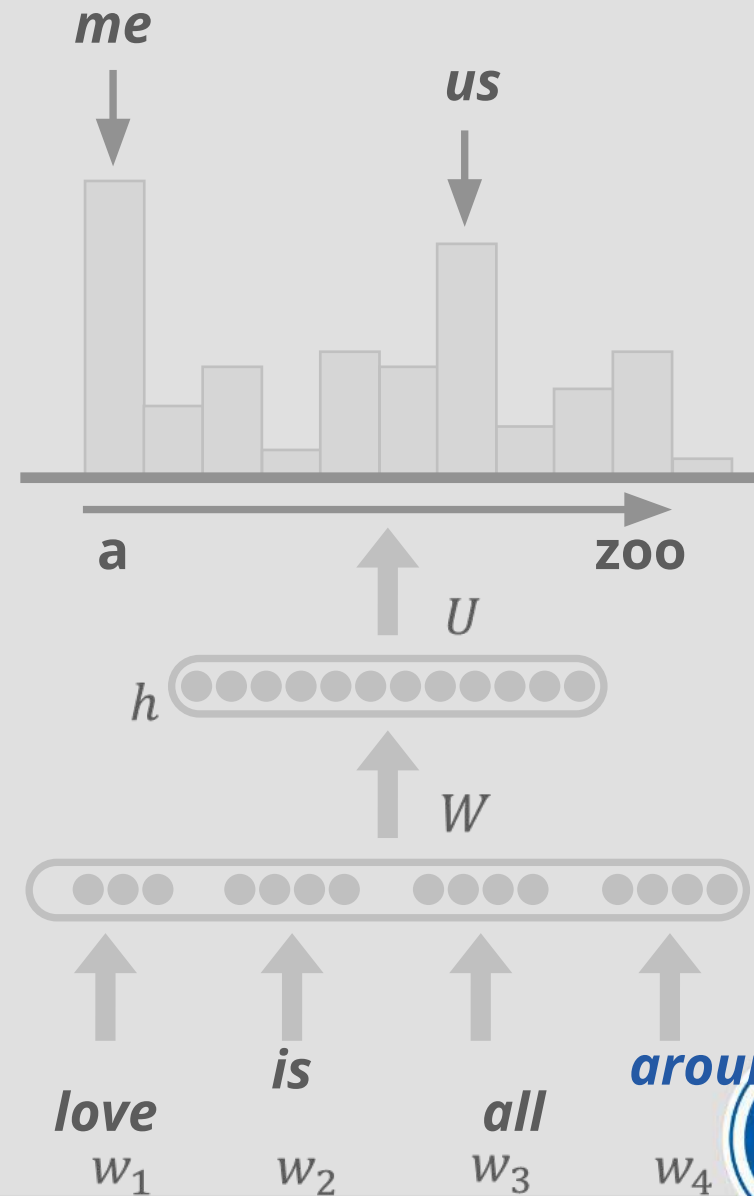
Language model based on neural networks

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



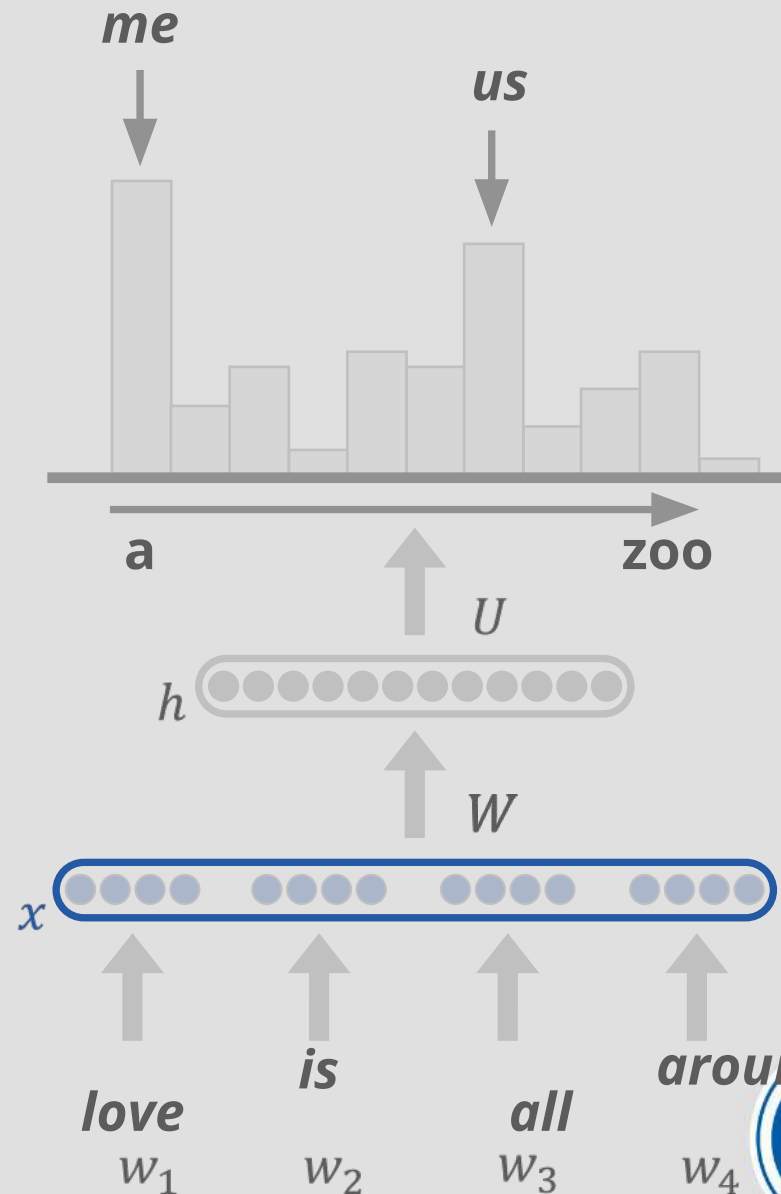
Language model based on neural networks

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



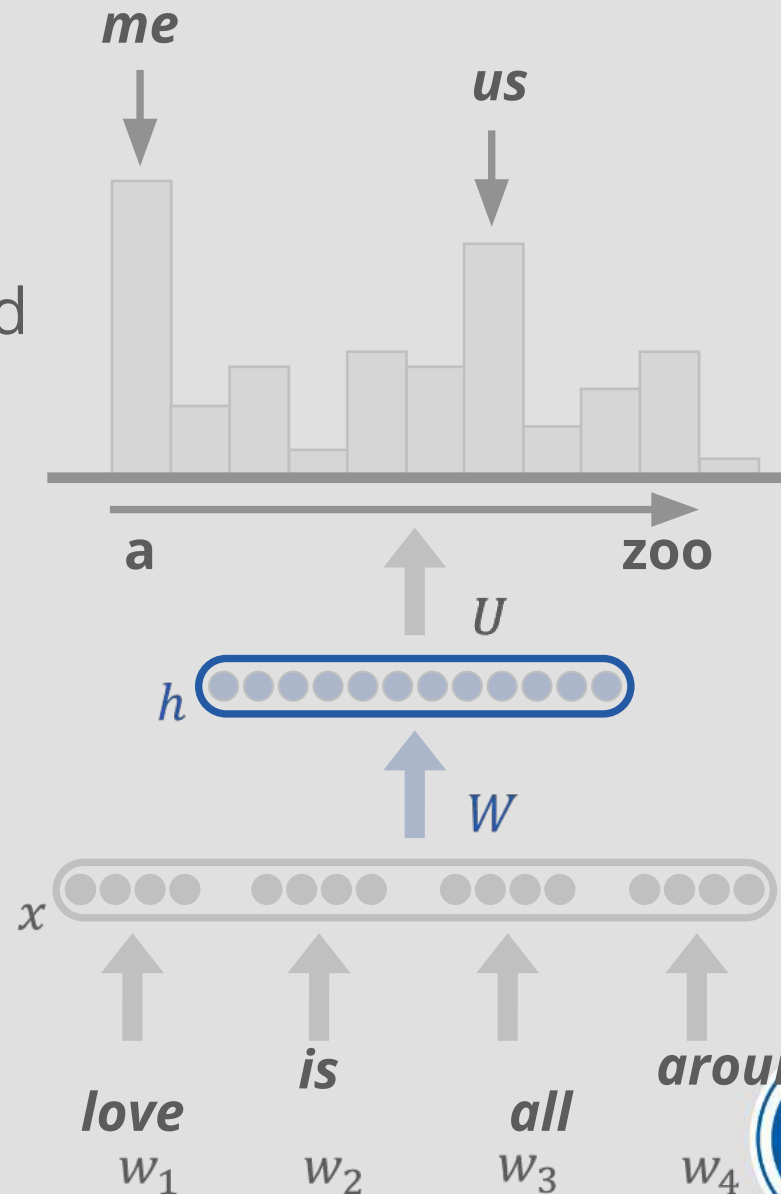
Language model based on neural networks

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



Language model based on neural networks

- Hidden layer:
$$h = \sigma(Wx + b)$$
- W, b_1 - weight matrix and bias vector

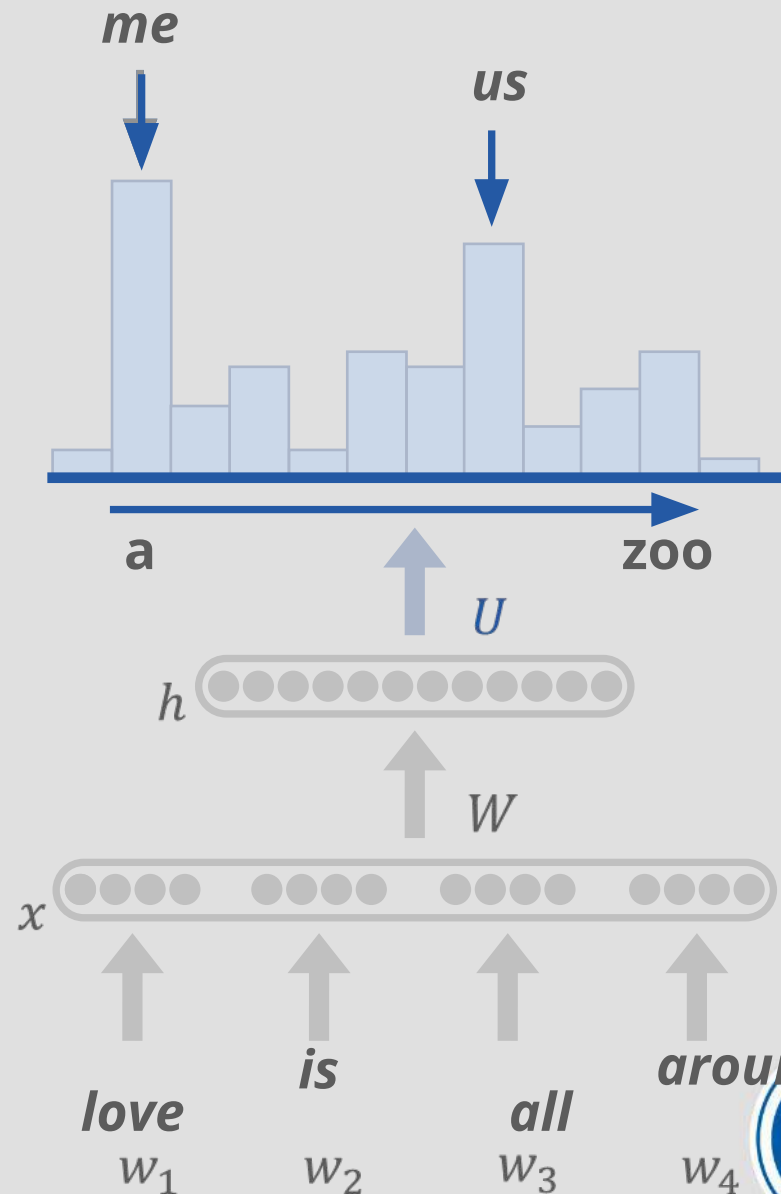


Language model based on neural networks

- The probability on next word:

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

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



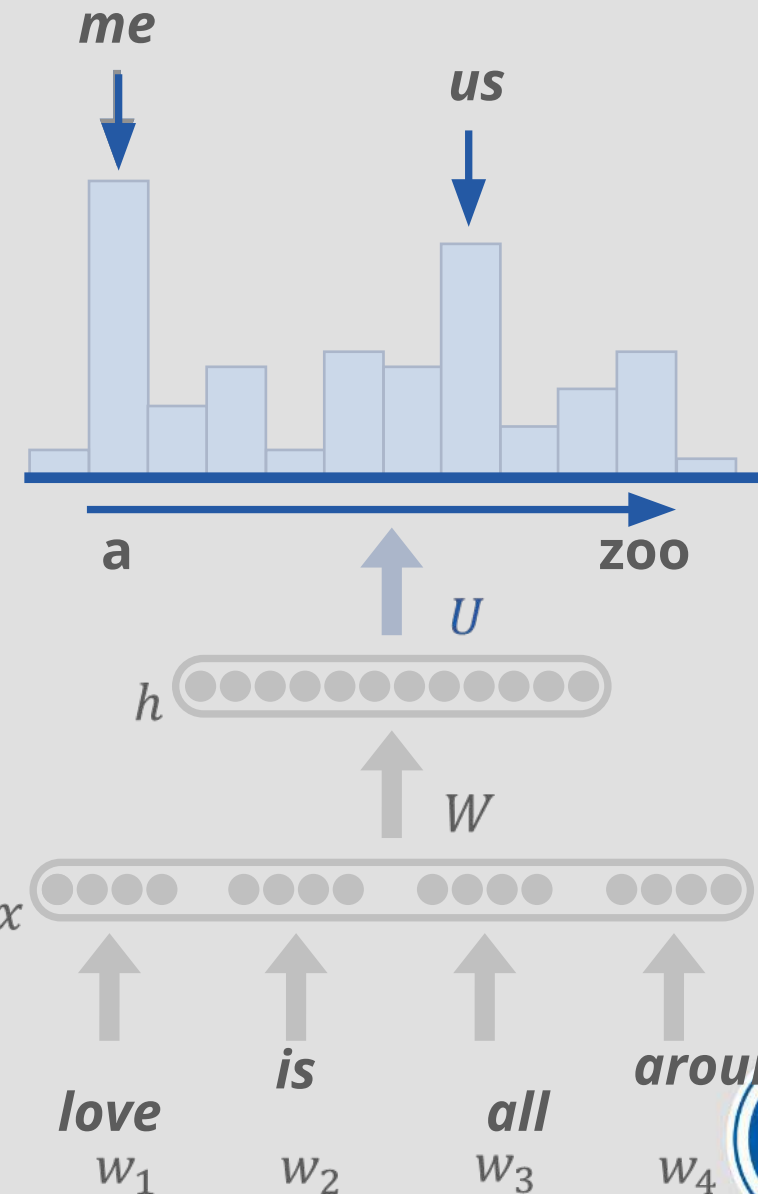
Language model based on neural networks

- The probability on next word:

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

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

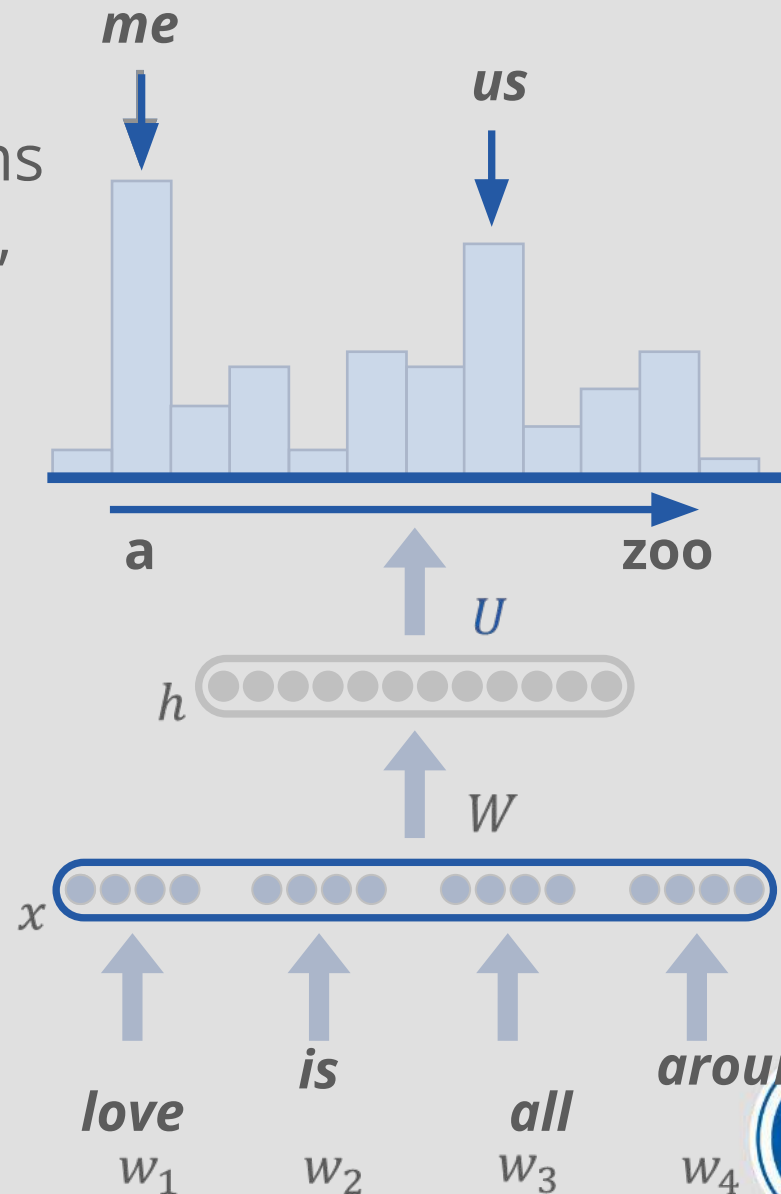
- $\hat{y} \in \mathbb{R}^{|V|}$
- U, b_1 - weight matrix and bias vector, both trainable



Language model based on neural networks

Training:

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



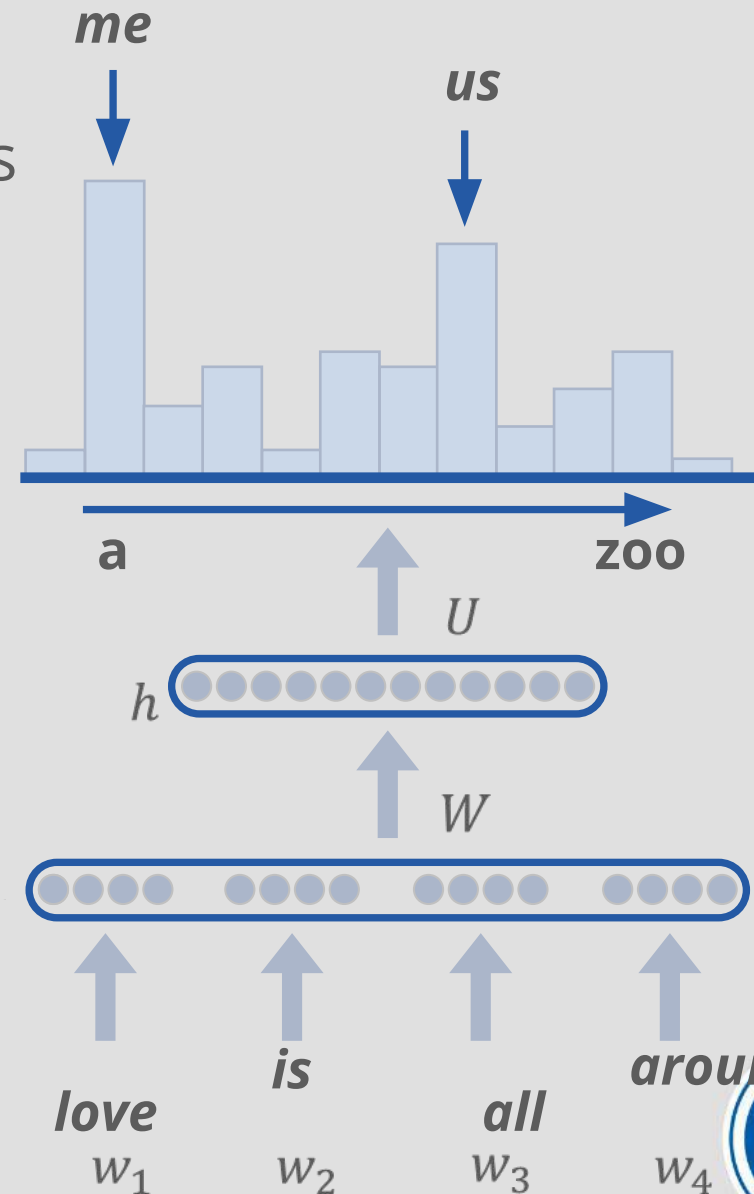
Language model based on neural networks

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

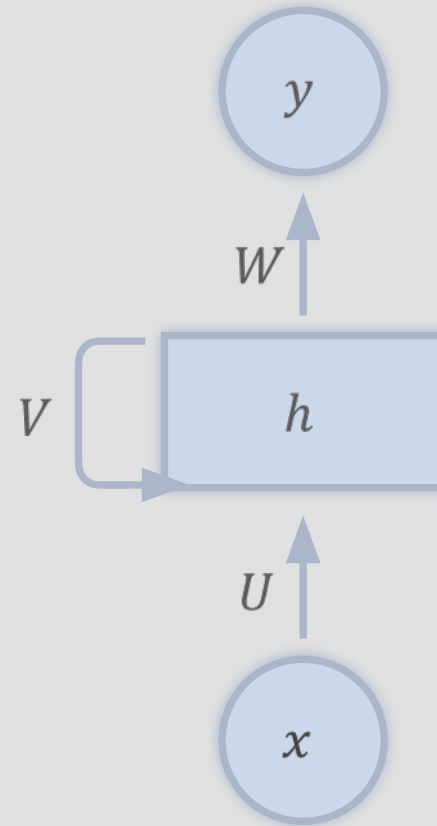


Recurrent neural networks

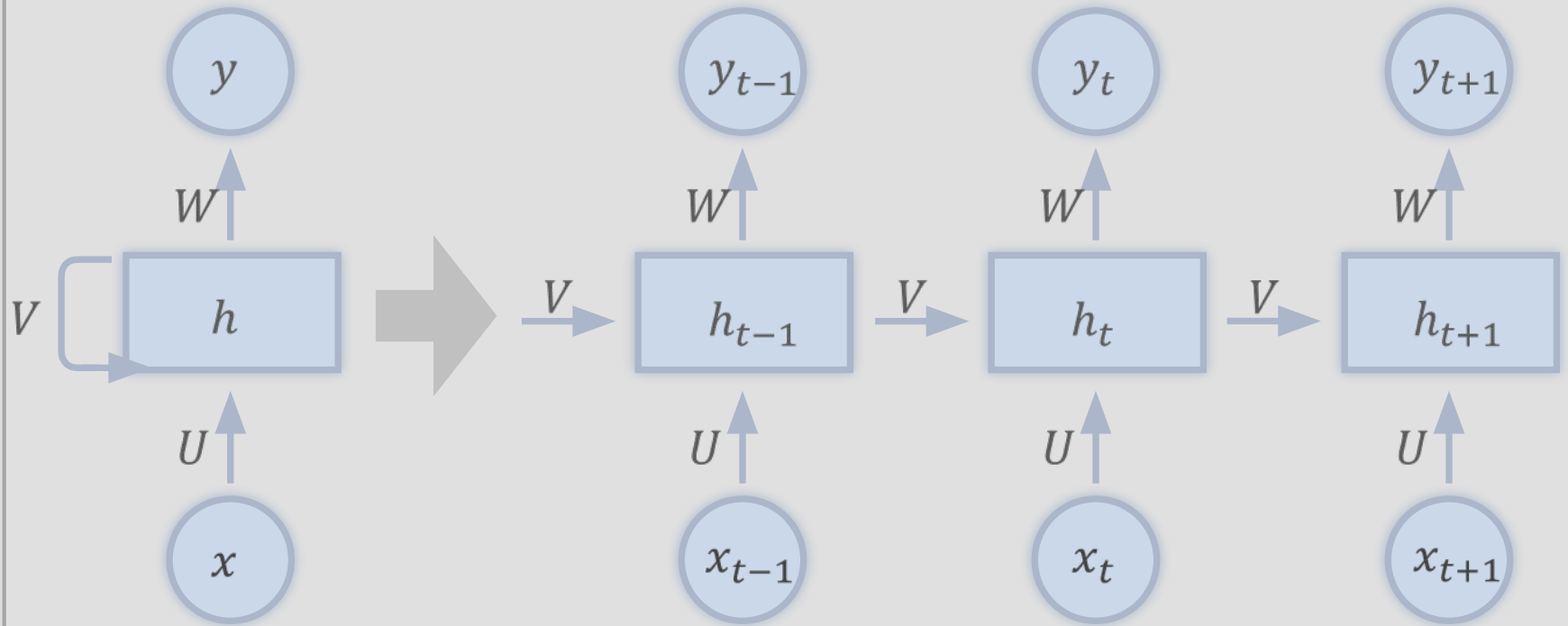


Recurrent neural network

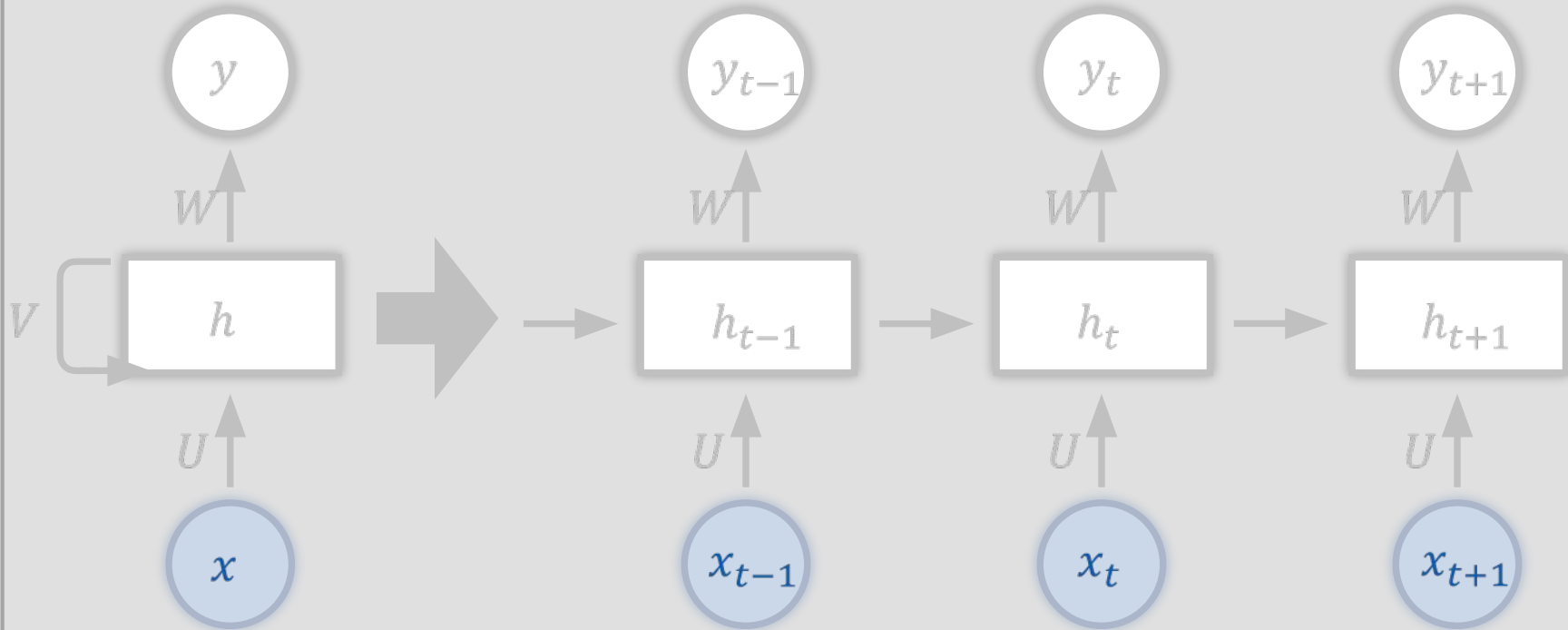
- Works with input sequences of arbitrary length
- Has a state embedding that keeps information about processed input



Recurrent neural network



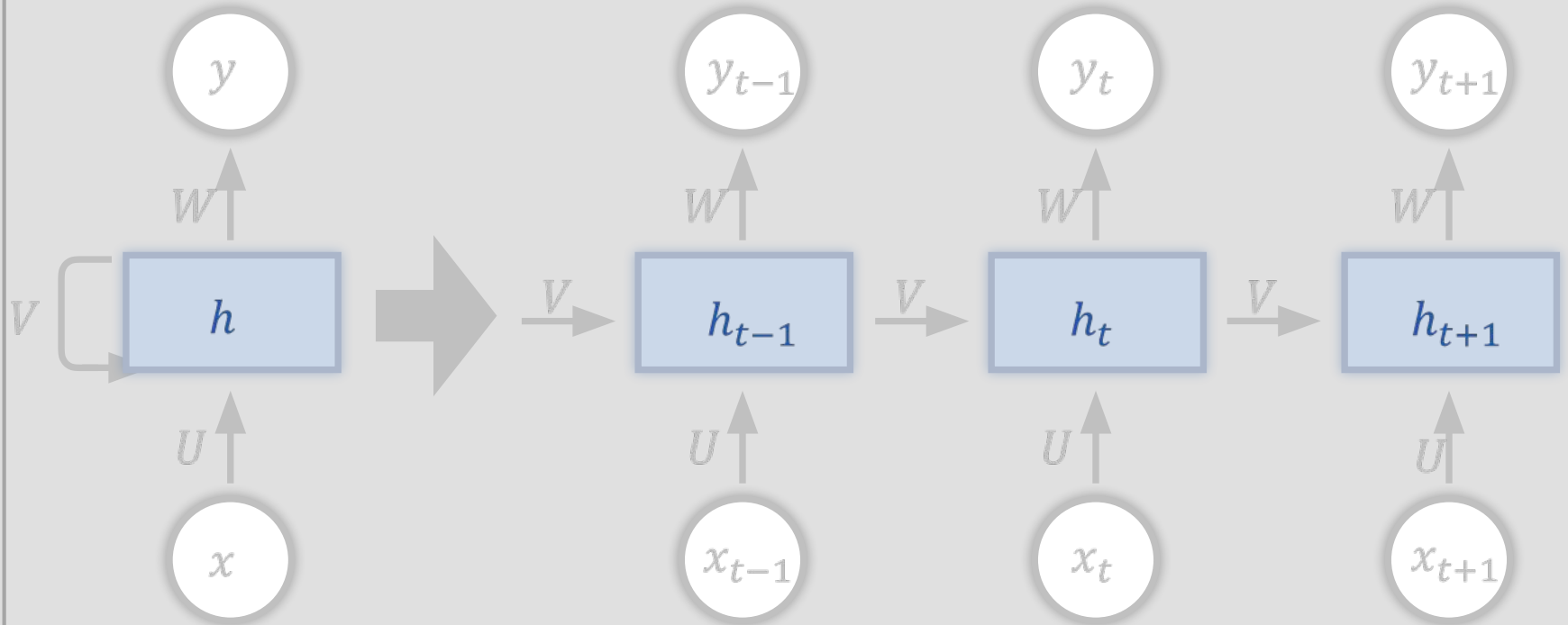
Recurrent neural network



- x - input words (embeddings)



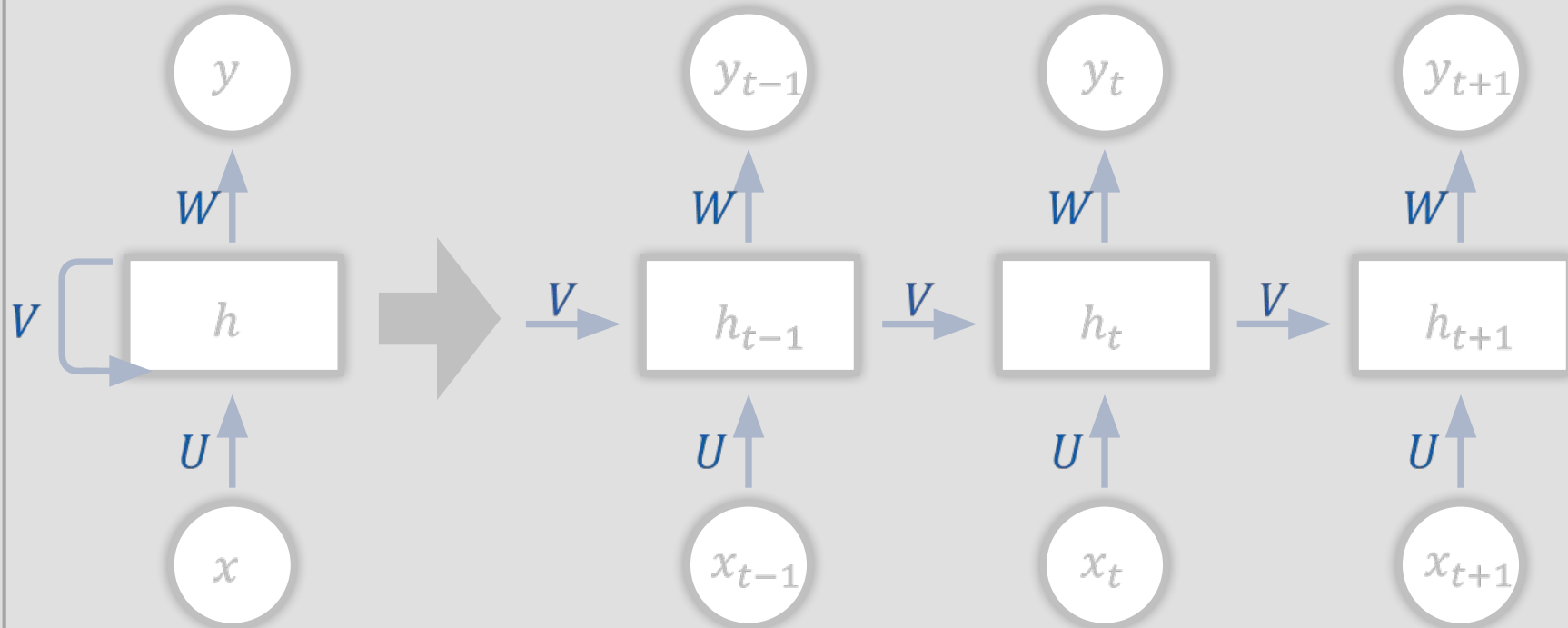
Recurrent neural network



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



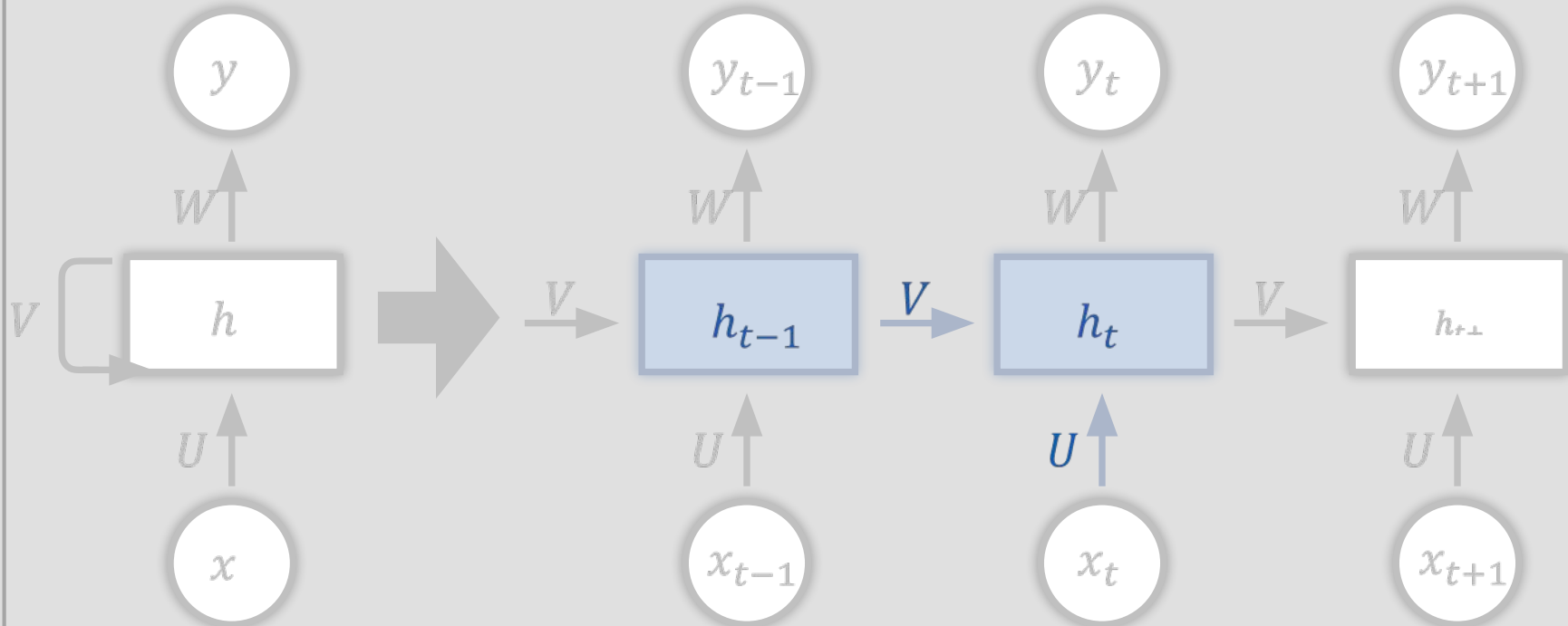
Recurrent neural network



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



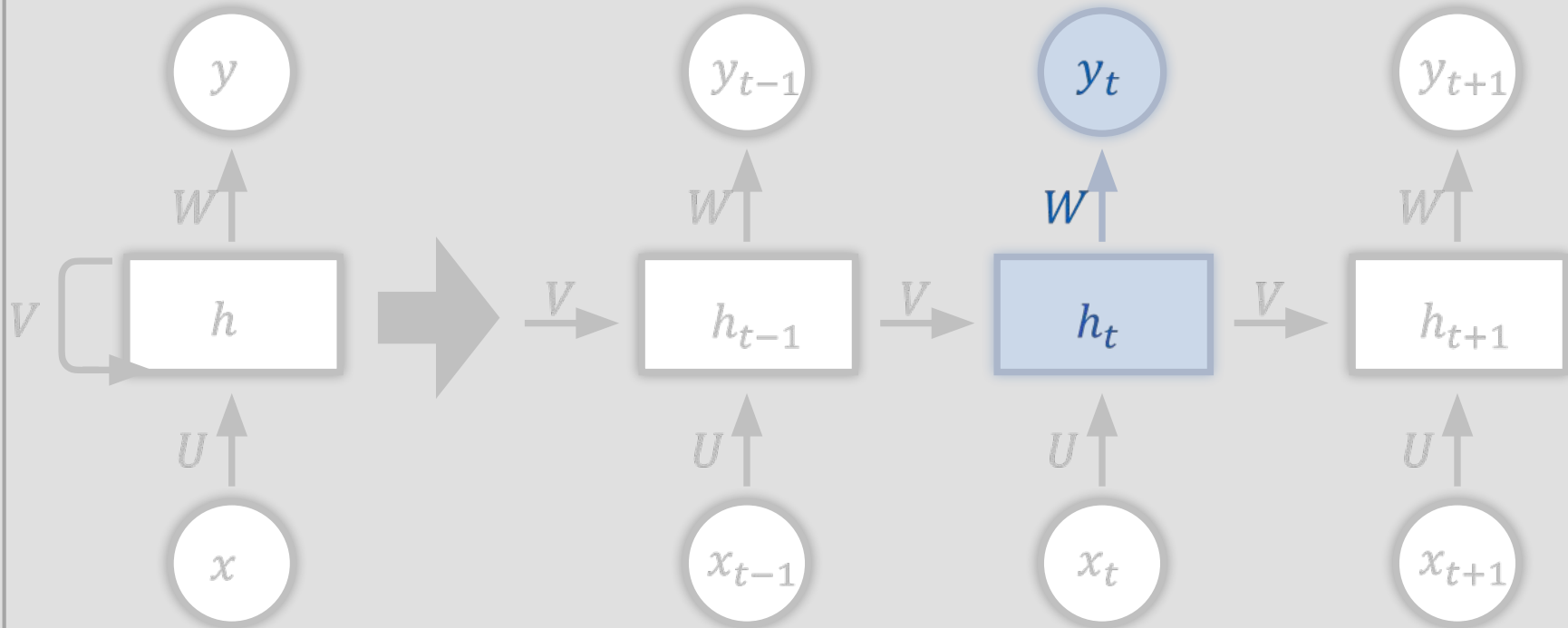
Recurrent neural network



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



Recurrent neural network



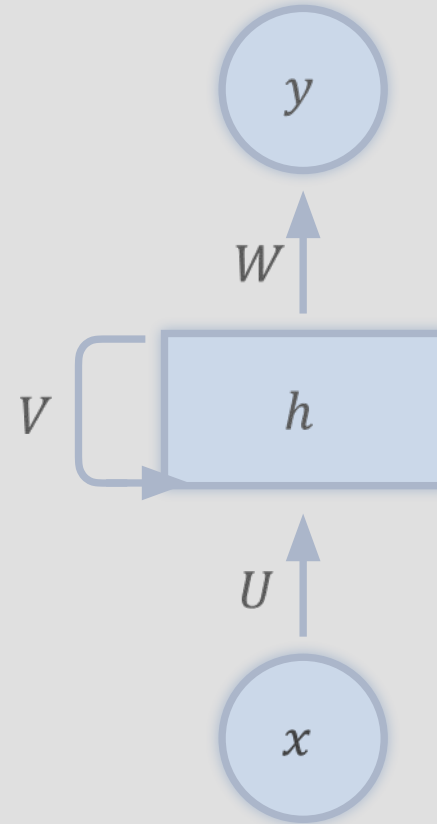
- $h_t = \sigma(Vh_{t-1} + Ux_t + b_1)$.
- $\hat{y}_t = \text{softmax}(Wh_t + b_2) \in \mathbb{R}^{|V|}$



Training process

- split corpus into words

w_1, w_2, \dots, w_n



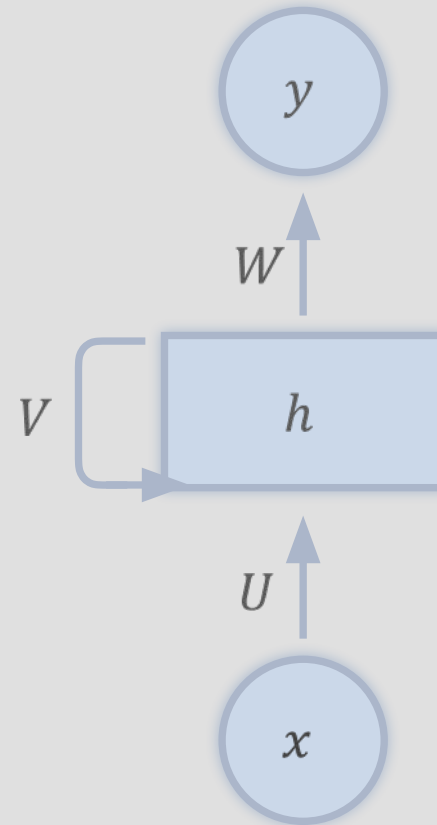
Training process

- split corpus into words

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

- word embeddings

$$x_i = Ew_i$$



Training process

- split corpus into words

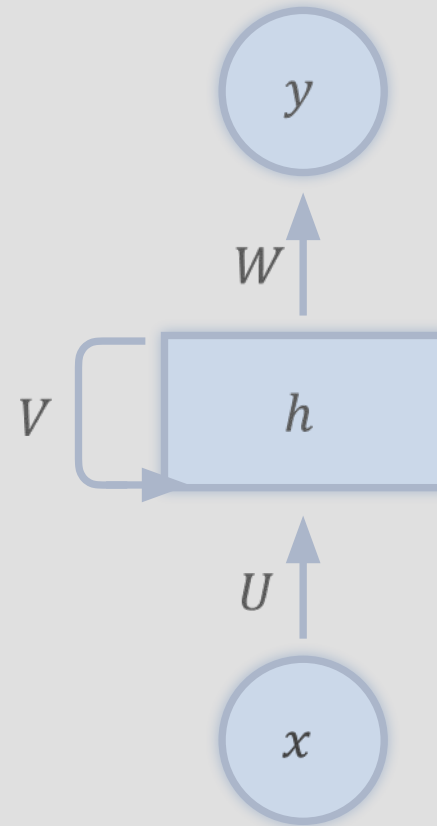
$$w_1, w_2, \dots, w_n$$

- word embeddings

$$x_i = E w_i$$

- slide over corpus

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



Training process

- split corpus into words

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

- word embeddings

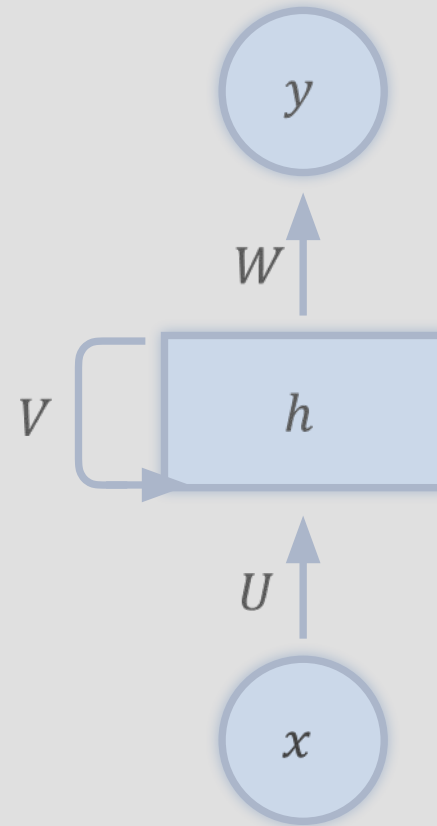
$$x_i = E w_i$$

- slide over corpus

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

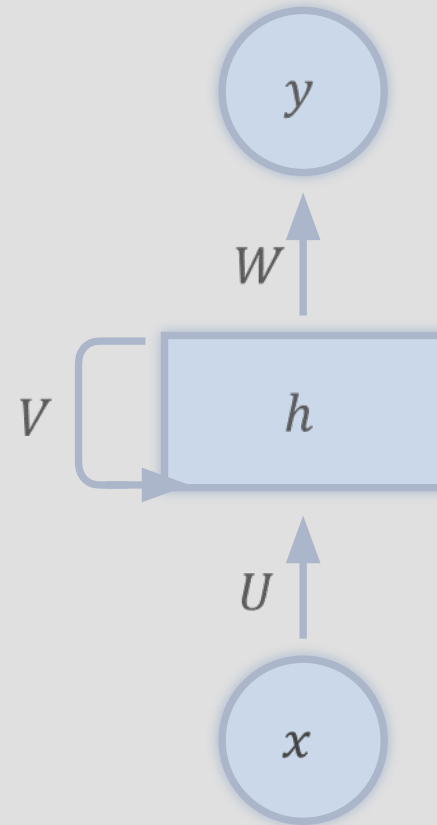
- predict

$$(x_i, y_i)$$



Training process

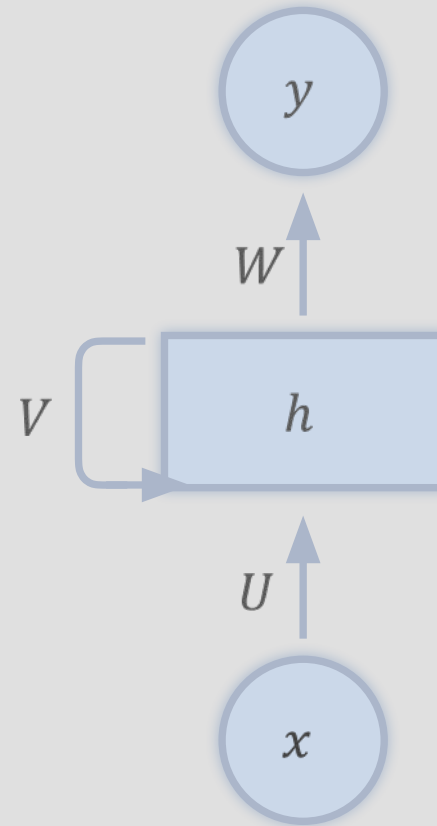
- Supplementary characters for the beginning and end of the sentence: $\langle s \rangle$, $\langle /s \rangle$



Training process

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

$$\begin{aligned}\text{loss}_{\text{local}} &= CE(y_t, \hat{y}_t) \\ &= - \sum_{w \in V} y_t^w \log \hat{y}_t^w\end{aligned}$$

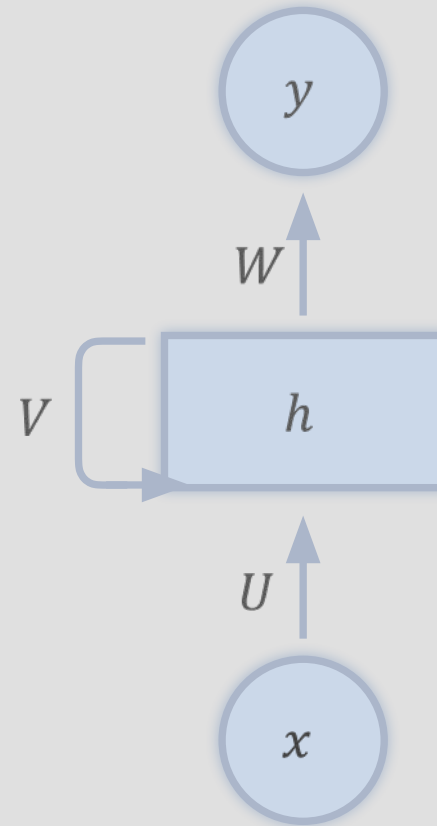


Training process

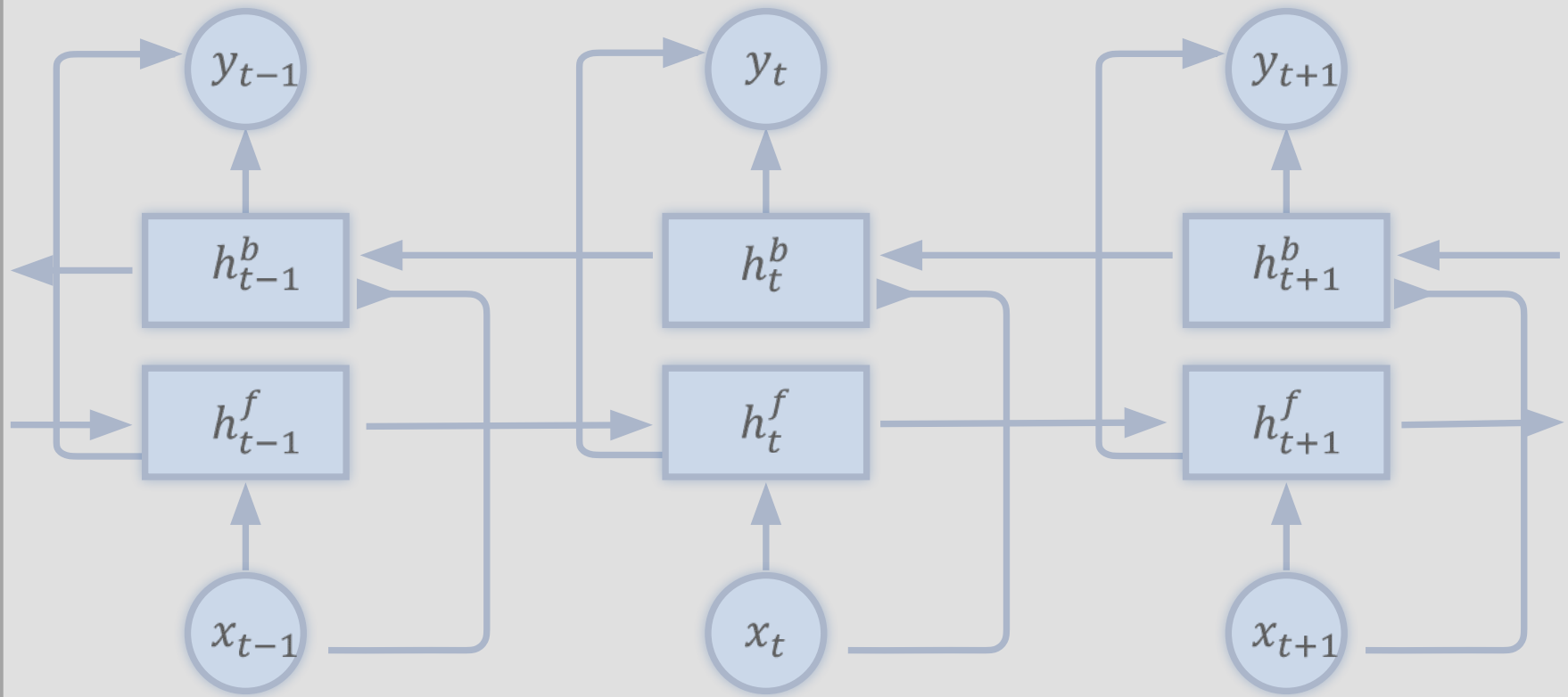
- at each step calculate the loss function, for example, cross entropy

$$\begin{aligned}\text{loss}_{\text{local}} &= CE(y_t, \hat{y}_t) \\ &= - \sum_{w \in V} y_t^w \log \hat{y}_t^w\end{aligned}$$

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



Bidirectional RNN



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

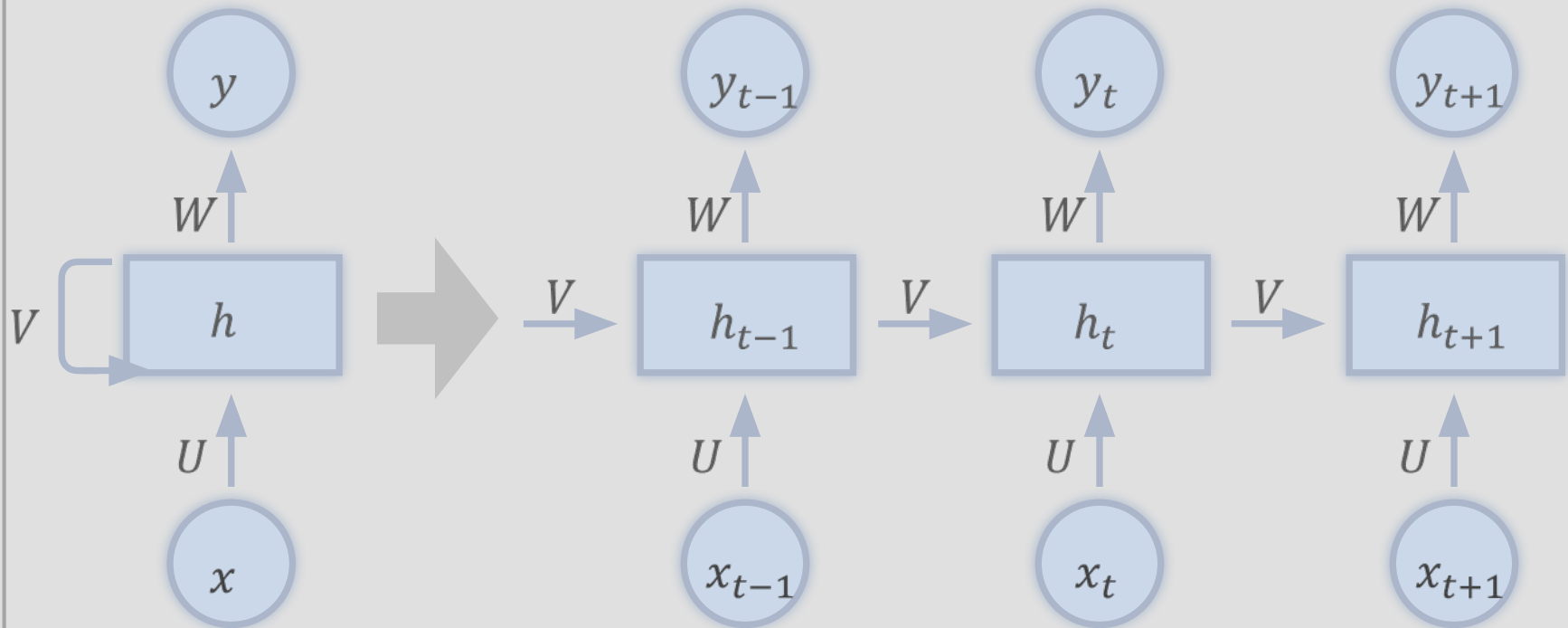
- Works with input sequences of arbitrary length
- Has a state embedding that keeps information about processed input
- The number of parameters doesn't depend on the corpus size

Disadvantages

- Slow computations
- Memory mechanism doesn't work well



Memory mechanism

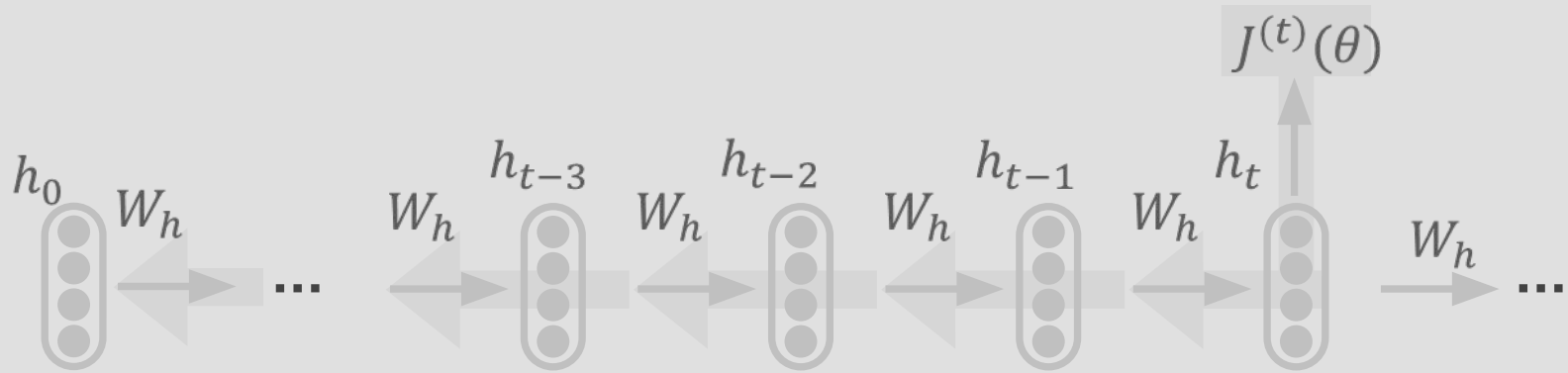


$$h_n = \sigma(h_{n-1}, x_n) = \sigma\left(\overbrace{\sigma(h_{n-2}, x_{n-1})}^{h_{n-1}}, x_n\right) = \sigma\left(\sigma\left(\overbrace{\sigma(h_{n-3}, x_{n-3})}^{h_{n-2}} x_{n-1}\right), x_n\right)$$

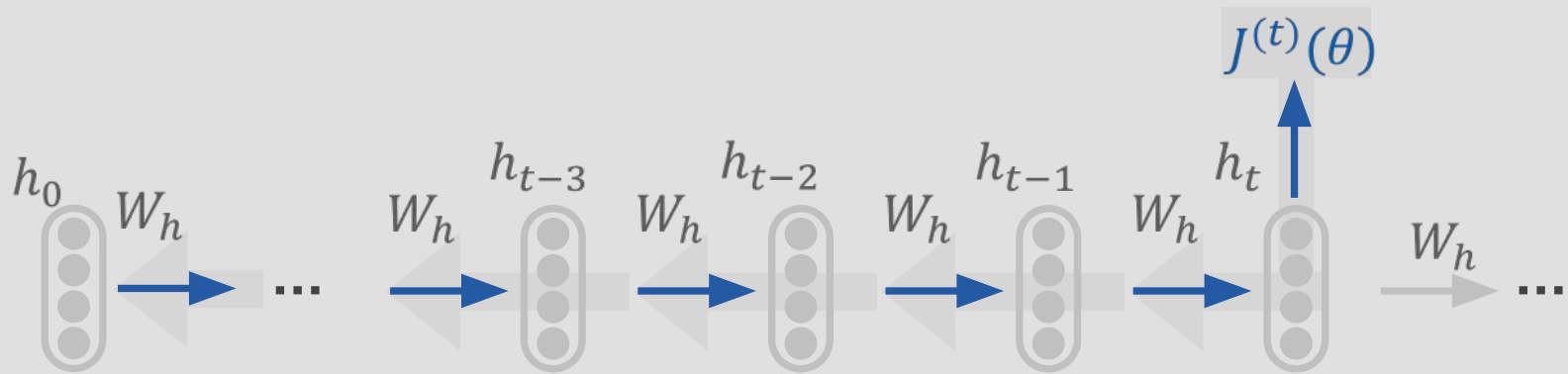
We update memory state at each step



Backpropagation through time mechanism



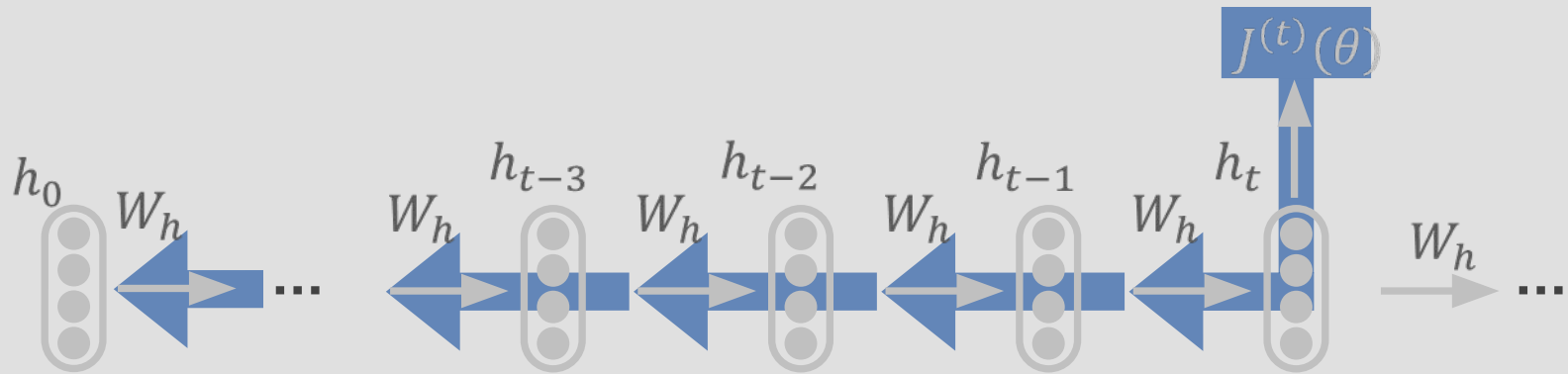
Backpropagation through time mechanism



- calculate all local losses



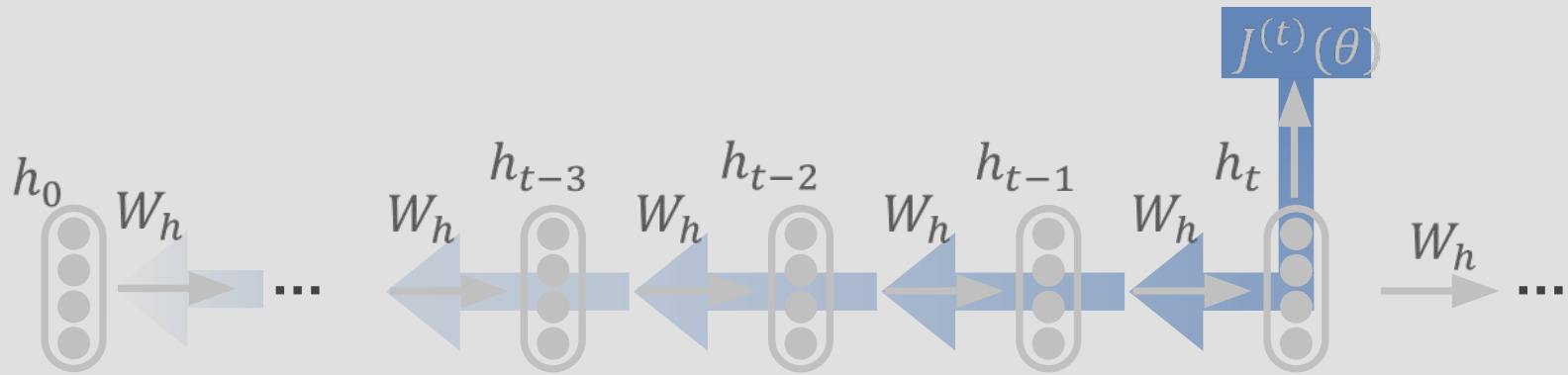
Backpropagation through time mechanism



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



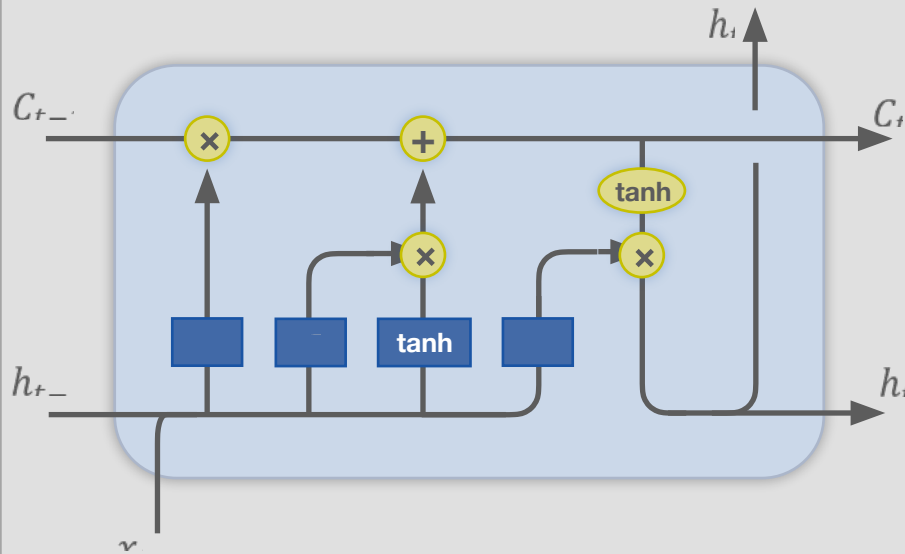
Backpropagation through time mechanism



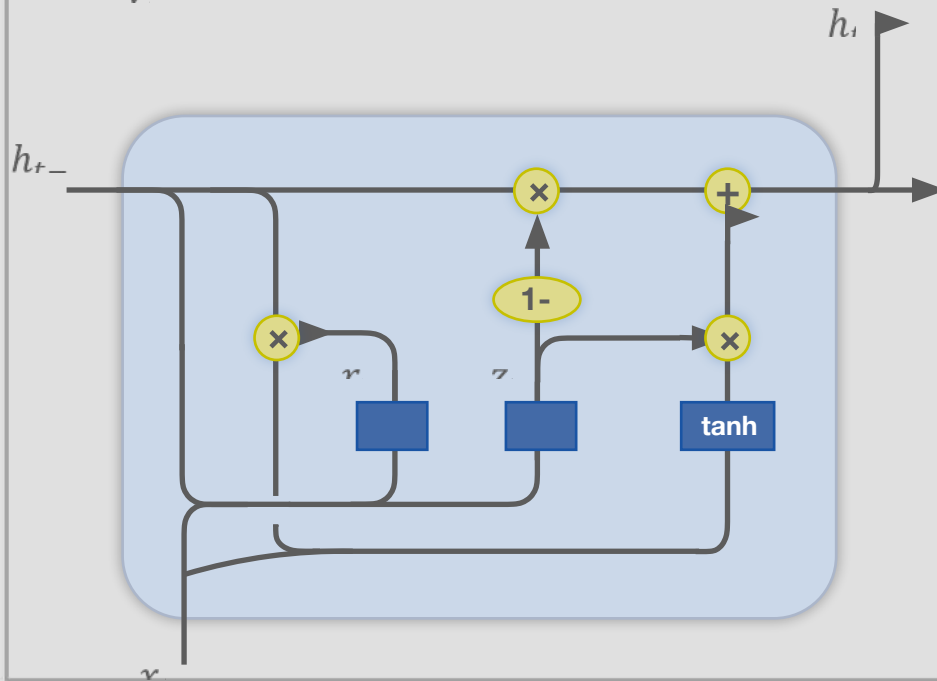
- calculate all local losses
- calculate the gradient from the last word down to the first one
- gradient vanishes due to the large amount of partial derivatives



Advanced RNN architectures



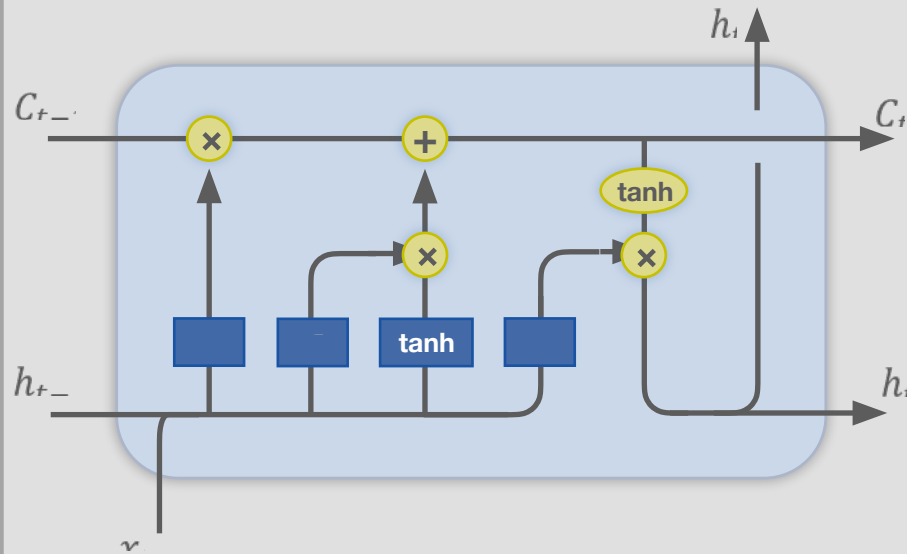
Long short
term memory
(LSTM)



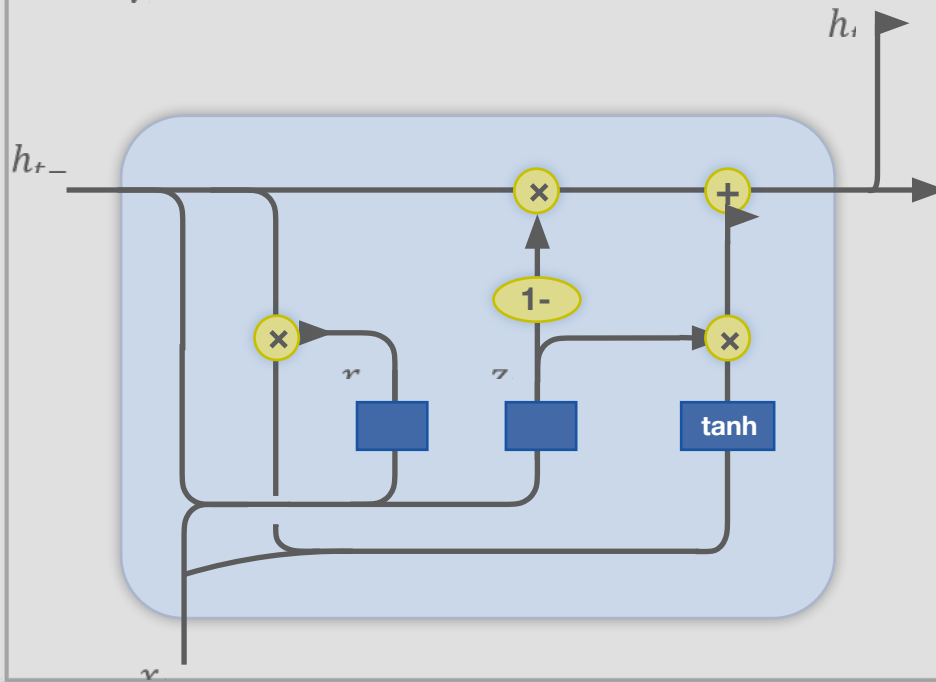
Gated
Recurrent
Unit
(GRU)



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, \dots, x_m
generate next n words x_{m+1}, \dots, x_{m+n}
- generation strategy P defines which word will be generated on every step:

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



Problem statement

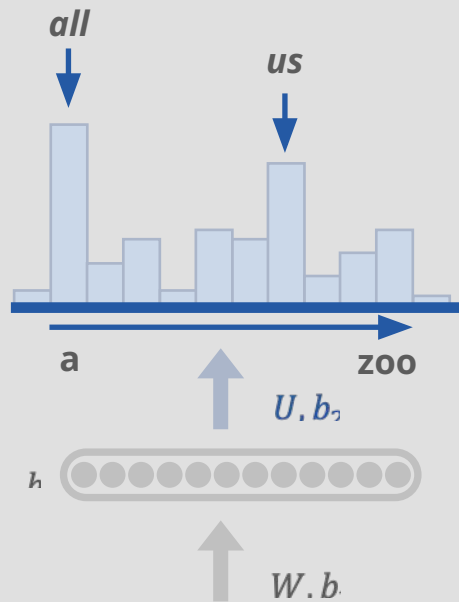
- given a context x_1, \dots, x_m
generate next n words x_{m+1}, \dots, x_{m+n}
- generation strategy P defines which word will be generated on every step:

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

- estimation all possible combinations is too difficult



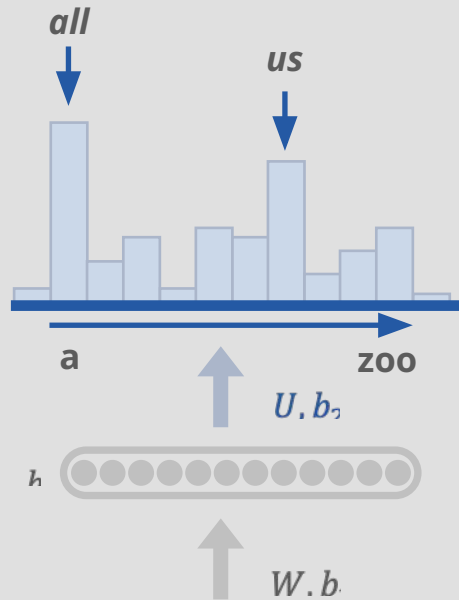
Generation strategies



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

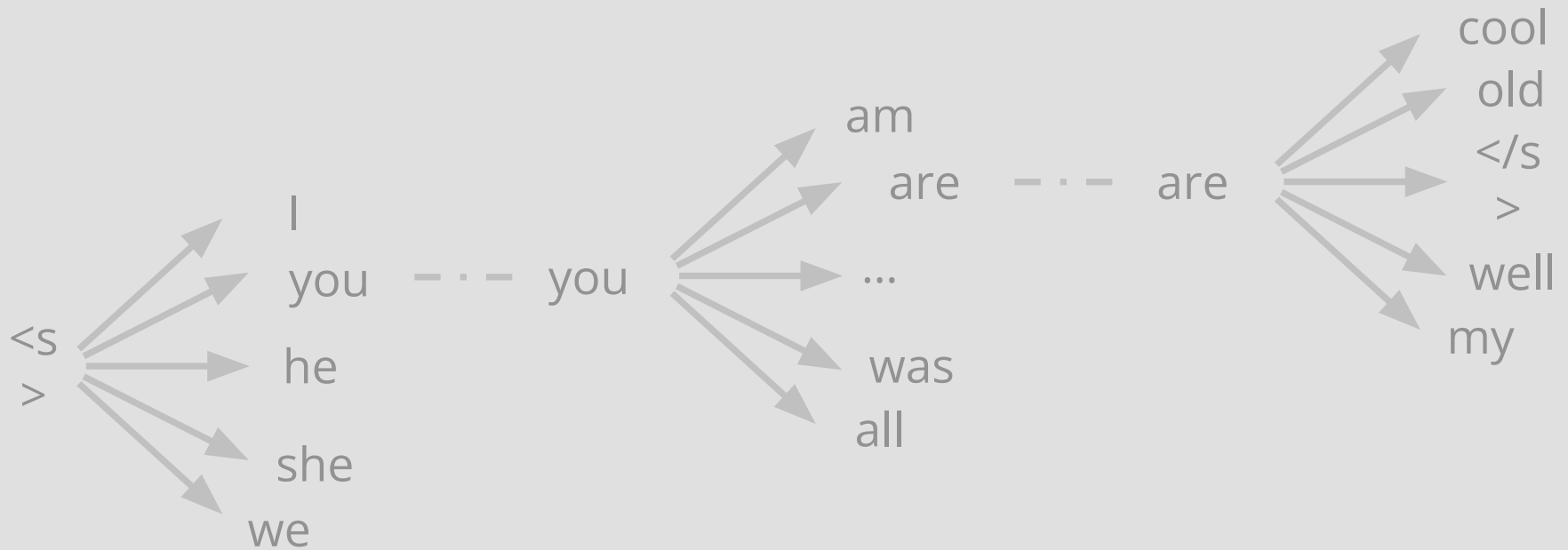
- Deterministic algorithm
- Chooses the most likely words at each step

Generation

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

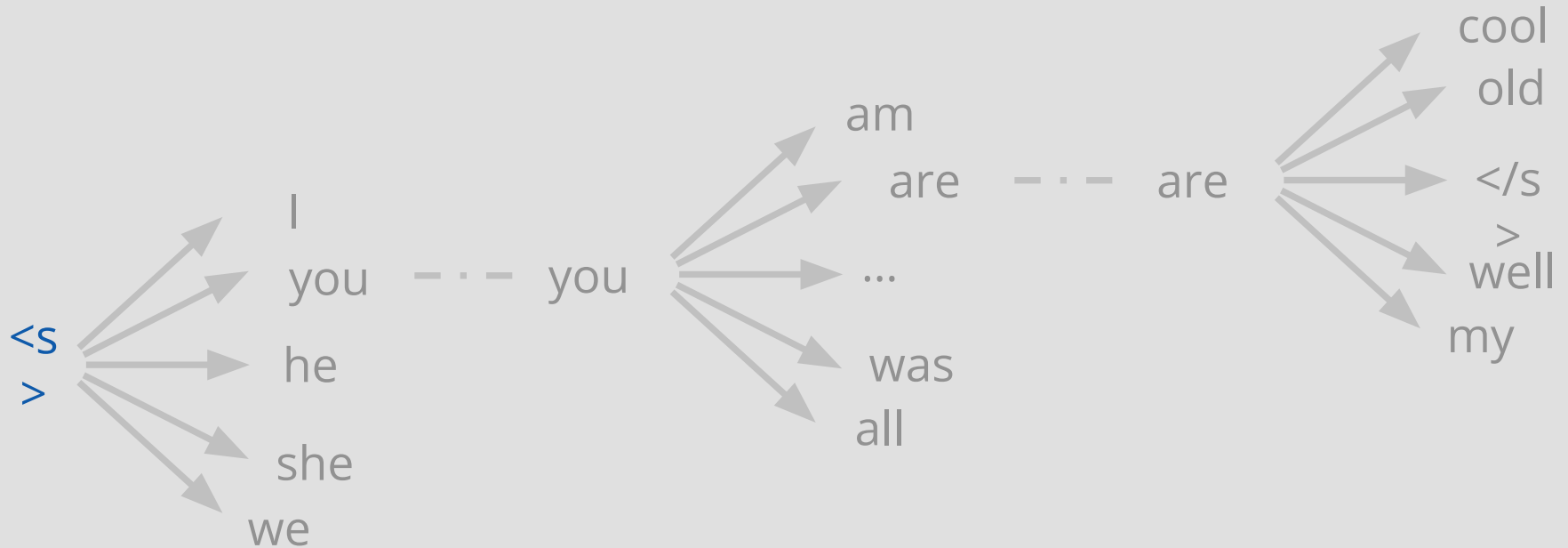


Greedy search



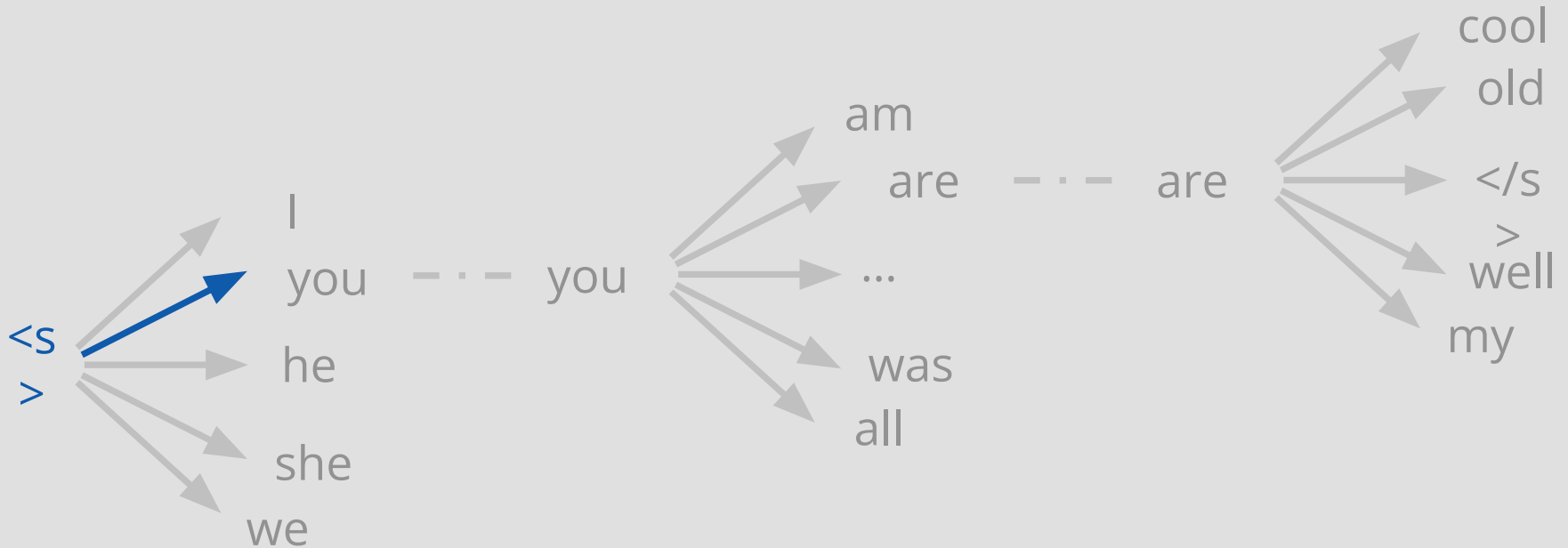
Greedy search

- Start with the beginning symbol <s>



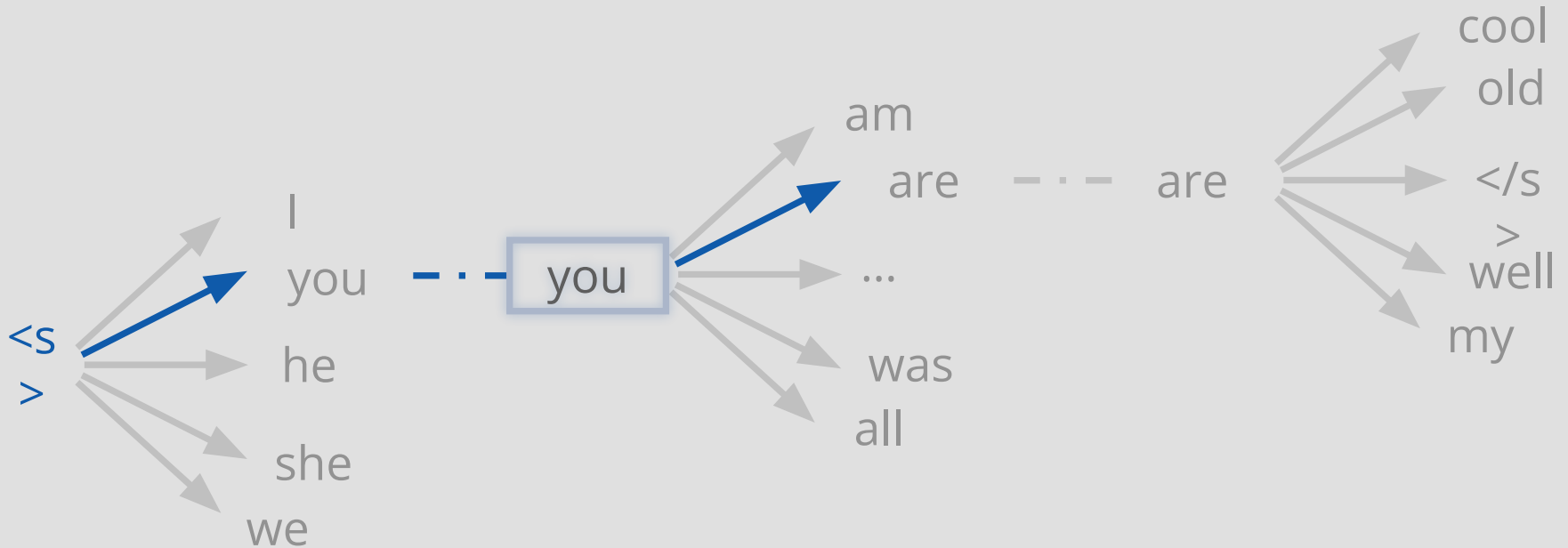
Greedy search

- Choose the most possible first word



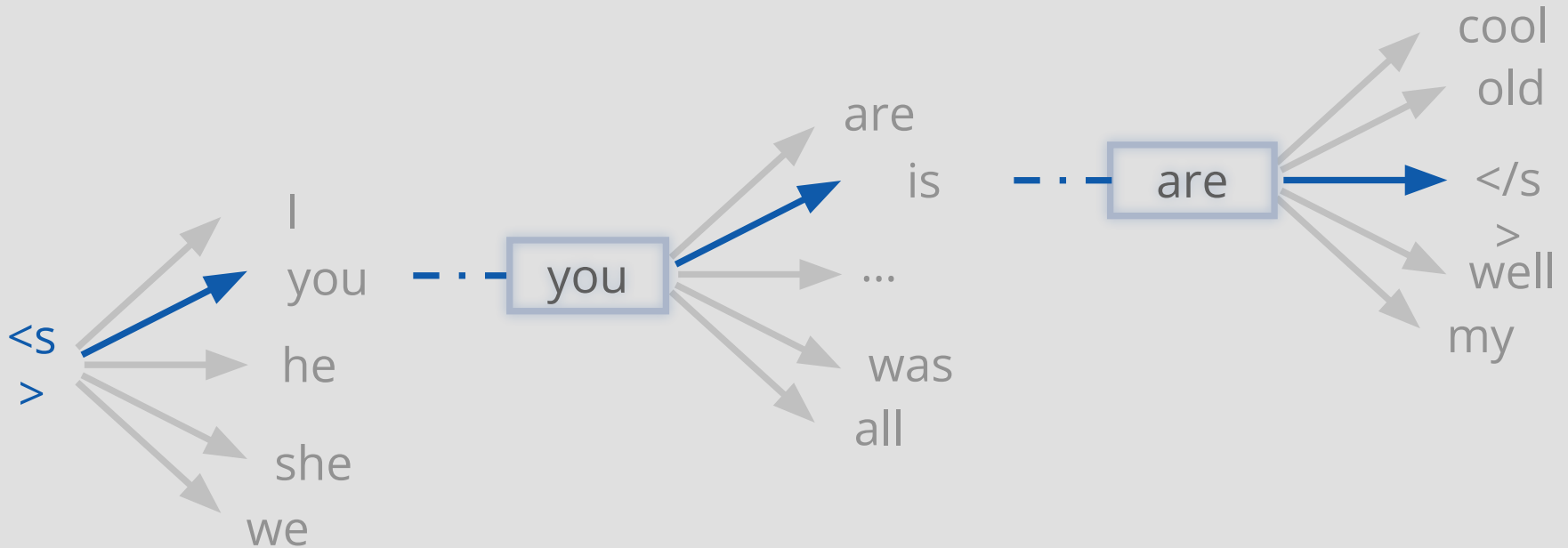
Greedy search

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

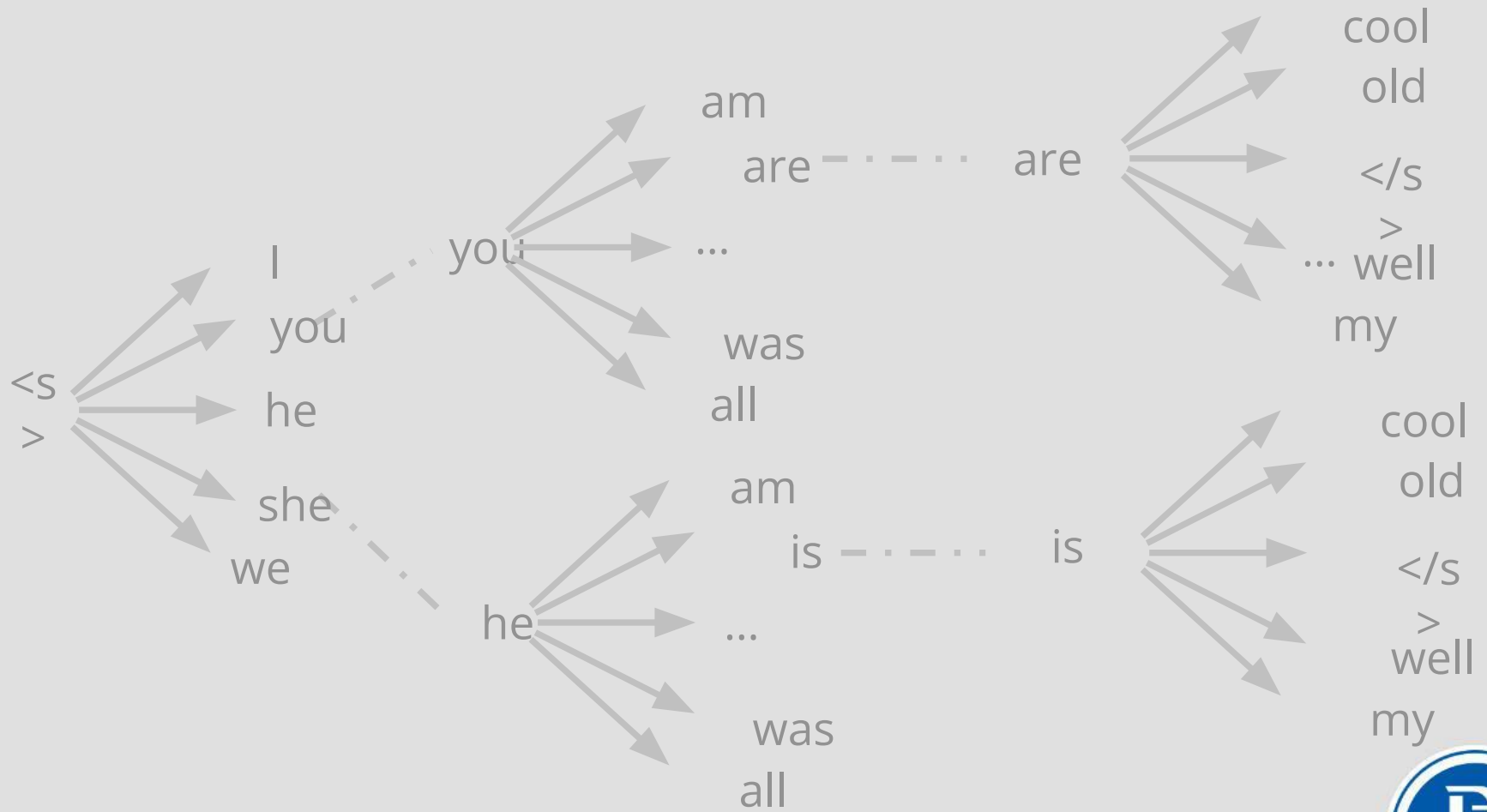


Greedy search

- generate until we get the `</s>` symbol

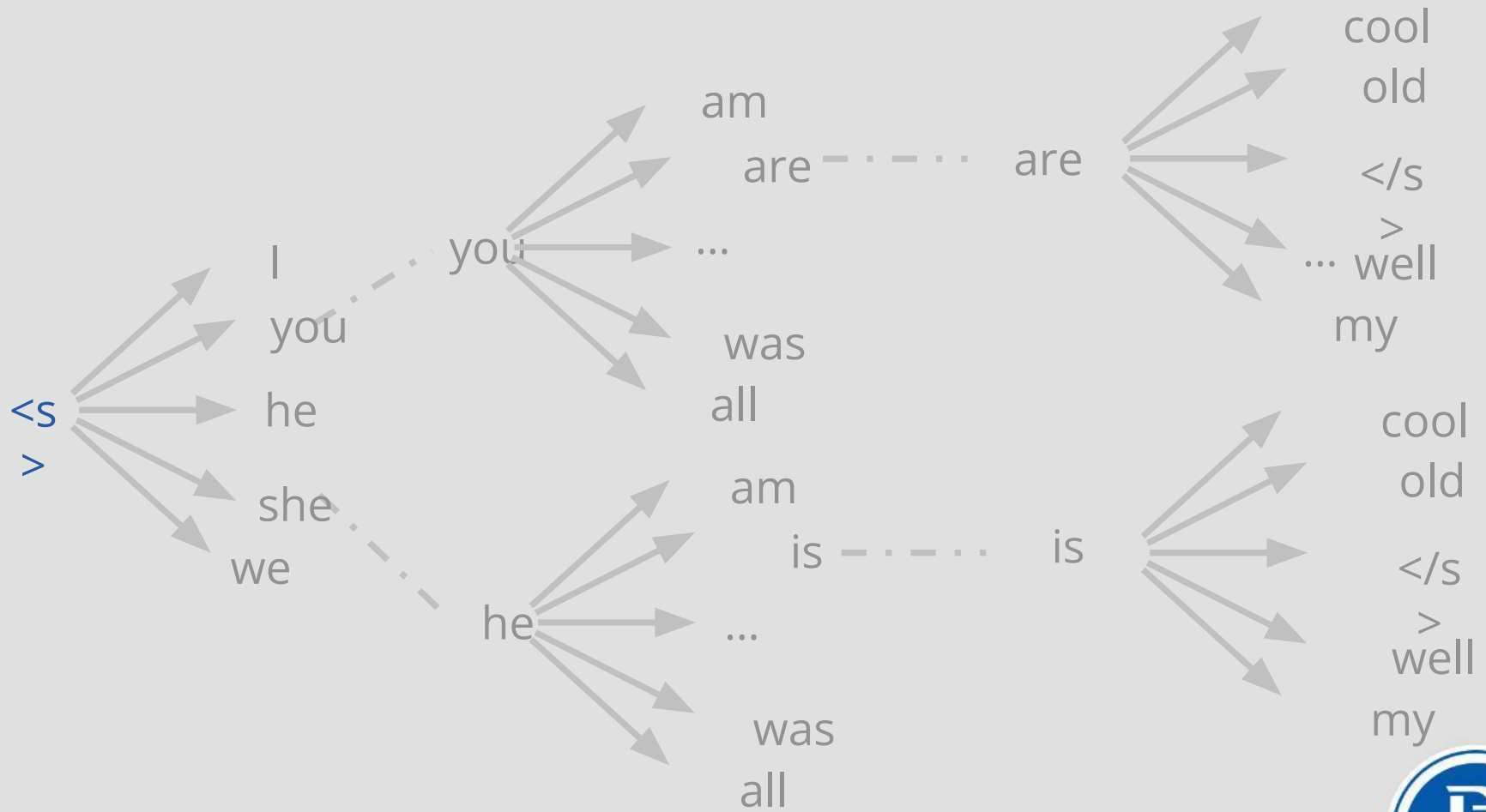


Beam search



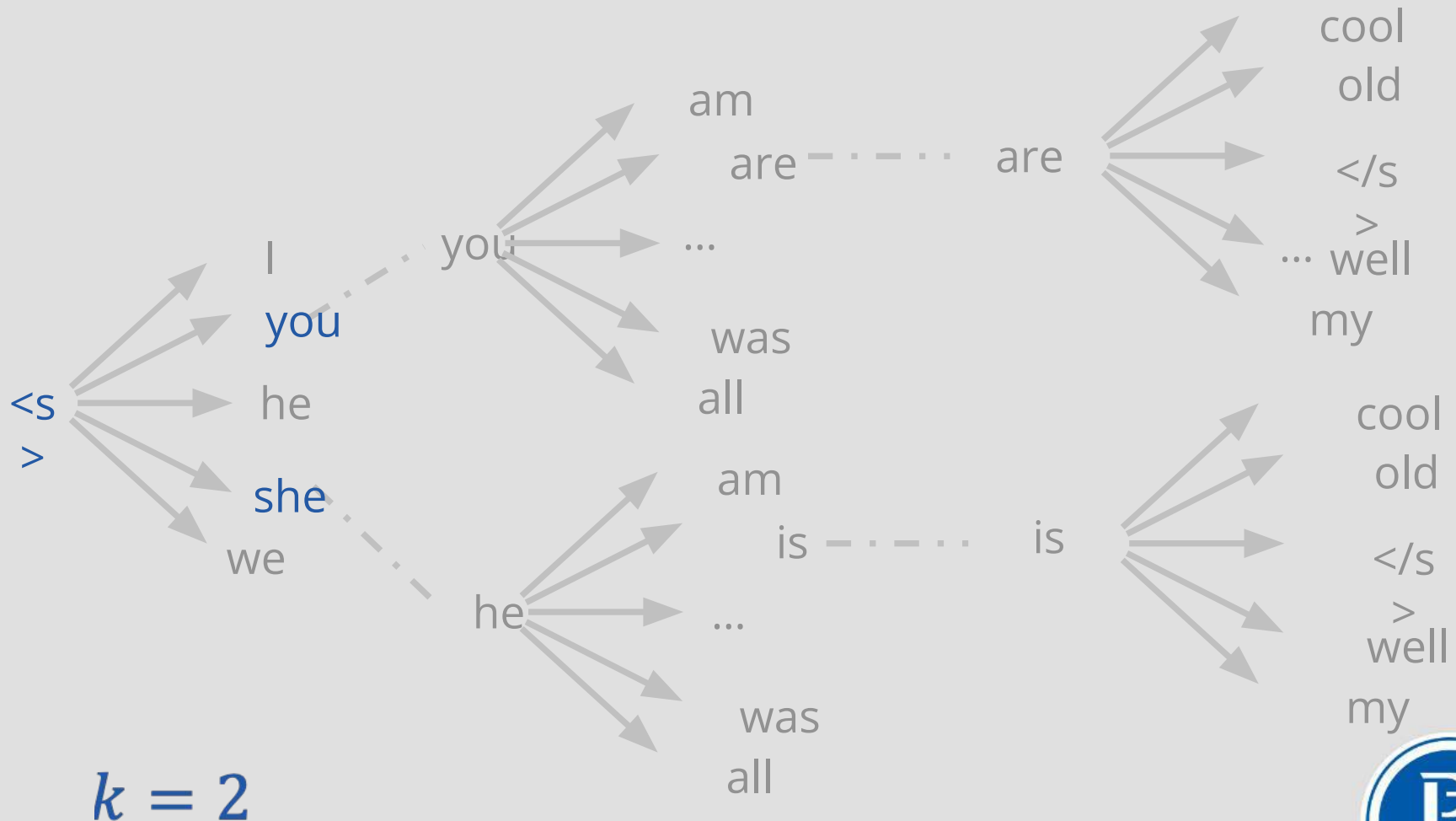
Beam search

- Start with the beginning symbol <s>



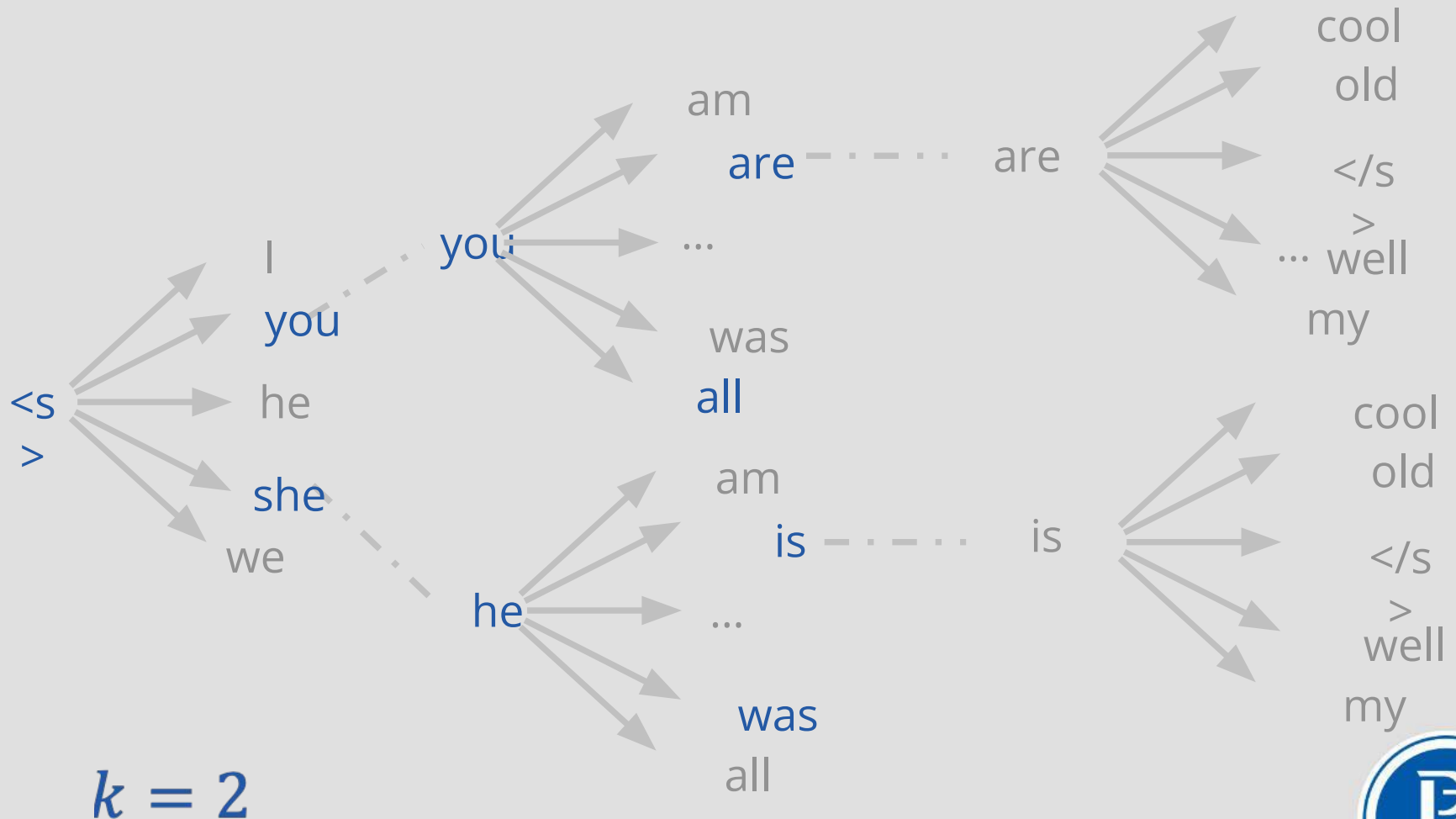
Beam search

- Choose k most probable words



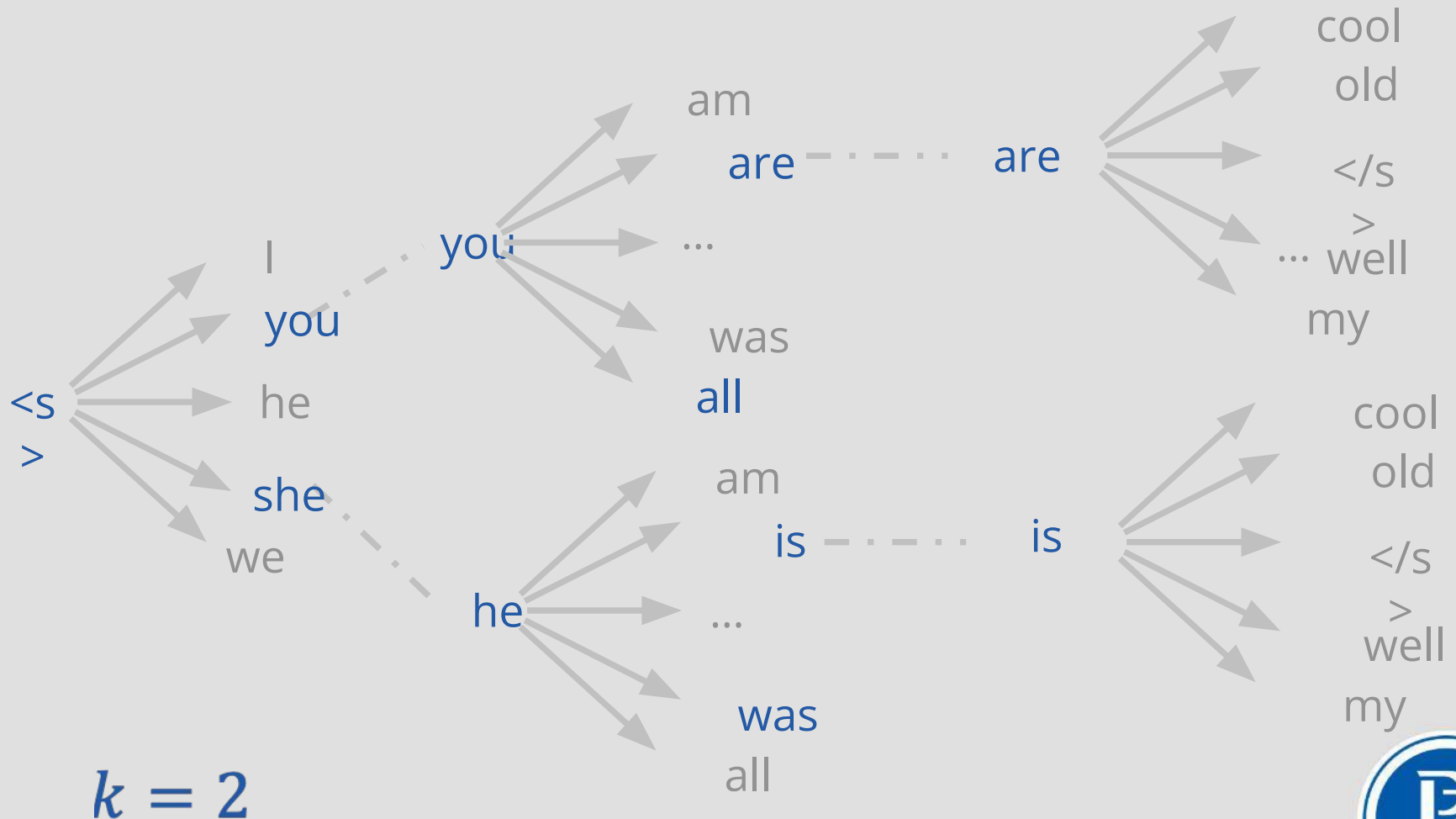
Beam search

- For each word choose k next most probable words



Beam search

- For each $k \times k$ options choose k most probable



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 replicas can be too general

