Natural Language Processing with Deep Learning

Sequence-to-sequence Models with Attention

Navid Rekab-Saz

navid.rekabsaz@jku.at

Institute of Computational Perception

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- Sequence-to-sequence models
- Attention Mechanism
- seq2seq with Attention

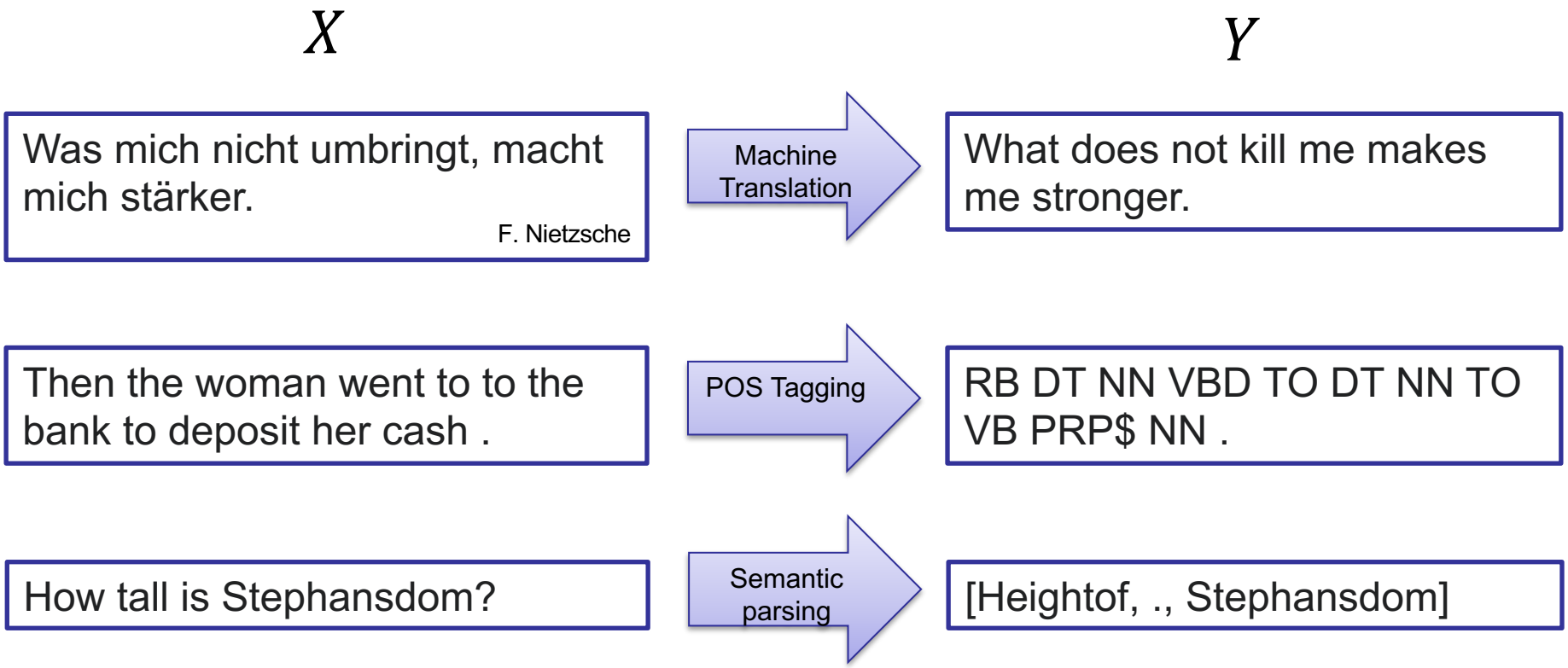
Some slides are adopted from http://web.stanford.edu/class/cs224n/

Agenda

- **Sequence-to-sequence models**
- Attention Mechanism
- seq2seq with Attention

Sequence in – sequence out!

- Several NLP tasks are defined as:
  - Given the source sequence $X = \{x^{(1)}, x^{(2)}, \dots, x^{(L)}\}$
  - Create/Generate the target sequence $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(T)}\}$



Sequence in – sequence out!

- Tasks such as:
  - Machine Translation (source language → target language)
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Code generation (natural language → SQL/Python code)
  - Named entity recognition
  - Dependency/semantic/ POS Parsing (input text → output parse as sequence)

but also …

  - Image captioning (image → caption)
  - Automatic Speech Recognition (speech → manuscript)

Image captioning

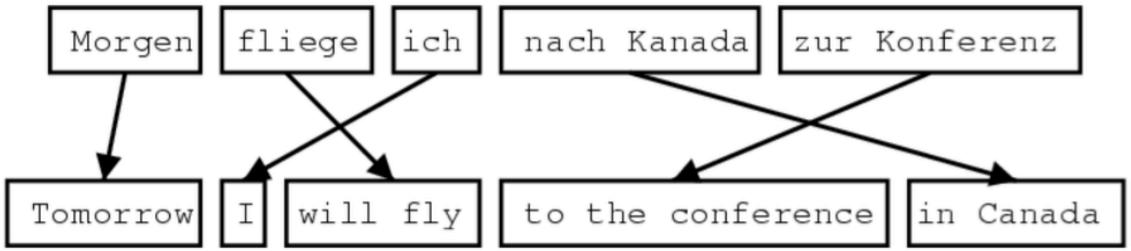
some elephants standing around a tall tree

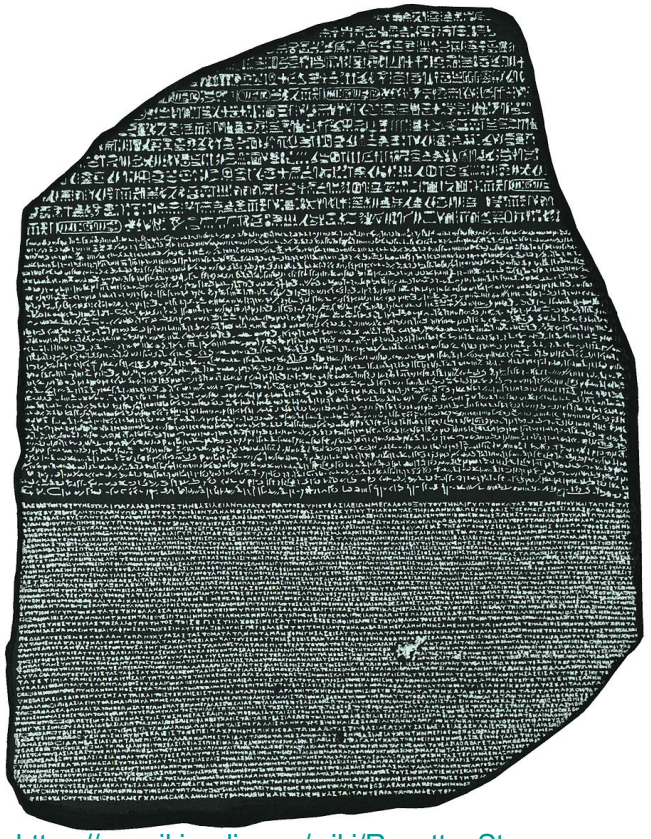
Machine Translation (MT)

- A long-history (since 1950)
- Statistical Machine Translation (1990-2010) – and also Neural MT – use large amount of parallel data to calculate:

$$\underset{Y}{\operatorname{argmax}}\, P(Y|X)$$

- Challenges:
  - Alignment
  - Common sense
  - Idioms!
  - Low-resource language pairs





https://en.wikipedia.org/wiki/Rosetta_Stone

Machine Translation (MT) – Evaluation

- BLEU (Bilingual Evaluation Understudy)

- BLEU computes a similarity score between the machine-written translation to one or several human-written translation(s), based on:
  - *n*-gram precision (usually for 1, 2, 3 and 4-grams)
  - plus a penalty for too-short machine translations

- BLEU is precision-based, while ROUGE is recall-based

Details of how to calculate BLEU: https://www.coursera.org/lecture/nlp-sequence-models/bleu-score-optional-kC2HD

Sequence-to-sequence model

- Sequence-to-sequence model (aka seq2seq) is the neural network architecture to approach …
  - given the source sequence $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(L)}\}$,
  - generate the target sequence $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(T)}\}$

- A seq2seq model first creates a model to estimate the conditional probability:

$$P(Y|X)$$

- and then generates a new sequence Y^* by solving:

$$Y^* = \underset{Y}{\operatorname{argmax}}\, P(Y|X)$$

Seq2seq model

- In fact, a seq2seq model is a conditional Language Model
- It calculates the probability of the next word of target sequence, conditioned on the previous words of target sequence and the source sequence:

for $y^{(1)} \rightarrow P(y^{(1)}|X)$

for $y^{(2)} \rightarrow P(y^{(2)}|X, y^{(1)})$

…

for $y^{(i)} \rightarrow P(y^{(i)}|X, y^{(1)}, \dots, y^{(i-1)})$

… and for whole the target sequence:

$$P(Y|X) = P(y^{(1)}|X) \times P(y^{(2)}|X, y^{(1)}) \times \cdots \times P(y^{(T)}|X, y^{(1)}, \dots, y^{(T-1)})$$

$$P(Y|X) = \prod_{t=1}^{T} P(y^{(t)}|X, y^{(1)}, \dots, y^{(t-1)})$$

Seq2seq – steps

- Like Language Modeling, we …
- … design a model that predicts the probabilities of the next words of the target sequence, one after each other (in auto-regressive fashion): $P(y^{(i)}|X, y^{(1)}, \ldots, y^{(i-1)})$
- We train the model by maximizing these probabilities for the correct next words, appearing in training data
- At inference time (or during decoding), we use the model to generate new target sequences, that have high generation probabilities: $P(Y|X)$

Seq2seq with two RNNs

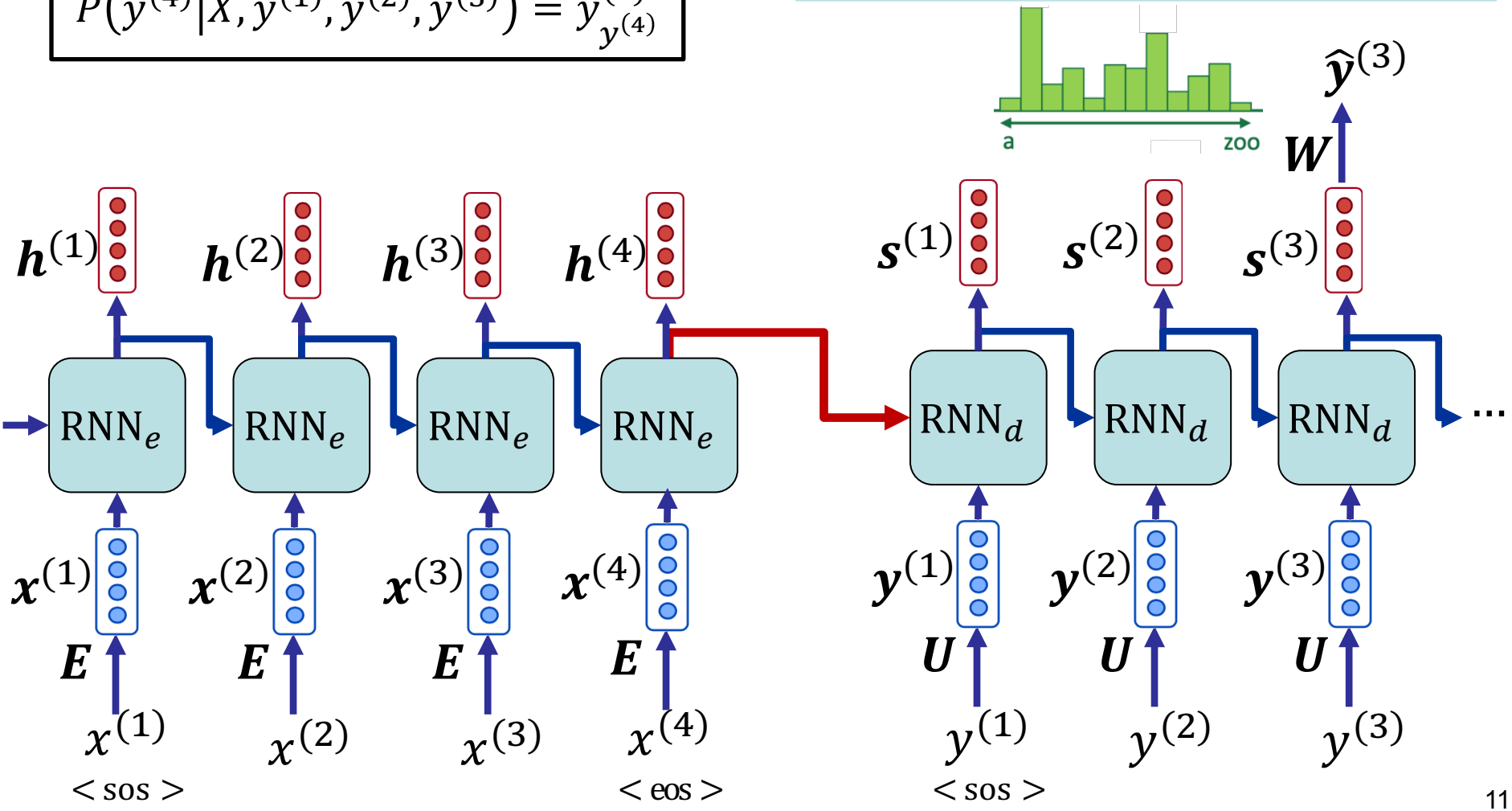
ENCODER

Probability of appearance of the next target word:

$$P\left(y^{(4)}\middle|X, y^{(1)}, y^{(2)}, y^{(3)}\right) = \hat{y}^{(3)}_{y^{(4)}}$$

DECODER

$\hat{\boldsymbol{y}}^{(i)}$: predicted probability distribution of the next target word, given the source sequence and previous target words



Seq2seq with two RNNs – formulation

- There are two sets of vocabularies
  - $\mathbb{V}_e$ is the set of vocabularies for source sequences
  - $\mathbb{V}_d$ is the set of vocabularies for target sequences

ENCODER

- Encoder embedding
  - Encoder embeddings for source words ($\mathbb{V}_e$) $\rightarrow$ $\boldsymbol{E}$
  - Embedding of the source word $x^{(l)}$ (at time step l) $\rightarrow$ $\boldsymbol{x}^{(l)}$

- Encoder RNN:

$$\boldsymbol{h}^{(l)} = \mathrm{RNN}(\boldsymbol{h}^{(l-1)}, \boldsymbol{x}^{(l)})$$

Parameters are shown in red

Seq2seq with two RNNs – formulation

DECODER

- Decoder embedding
  - Decoder embeddings *at input* for target words ($\mathbb{V}_d$) → $\boldsymbol{U}$
  - Embedding of the target word $y^{(t)}$ (at time step t) → $\boldsymbol{y}^{(t)}$

- Decoder RNN

$$\boldsymbol{s}^{(t)} = \mathrm{RNN}(\boldsymbol{s}^{(t-1)}, \boldsymbol{y}^{(t)})$$

  - The values of the last hidden state of the encoder RNN are passed to the initial hidden state of the decoder RNN:

$$\boldsymbol{s}^{(0)} = \boldsymbol{h}^{(L)}$$

Parameters are shown in red

Seq2seq with two RNNs – formulation

DECODER

- Decoder output prediction
  - Predicted probability distribution of words at the next time step:

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{W}\boldsymbol{s}^{(t)} + \boldsymbol{b}\right) \in \mathbb{R}^{|\mathbb{V}_d|}$$

  - Probability of the next target word (at time step $t + 1$):

$$P\left(y^{(t+1)} \middle| X, y^{(1)}, \dots, y^{(t-1)}, y^{(t)}\right) = \hat{y}^{(t)}_{y^{(t+1)}}$$

Parameters are shown in red

Training Seq2seq

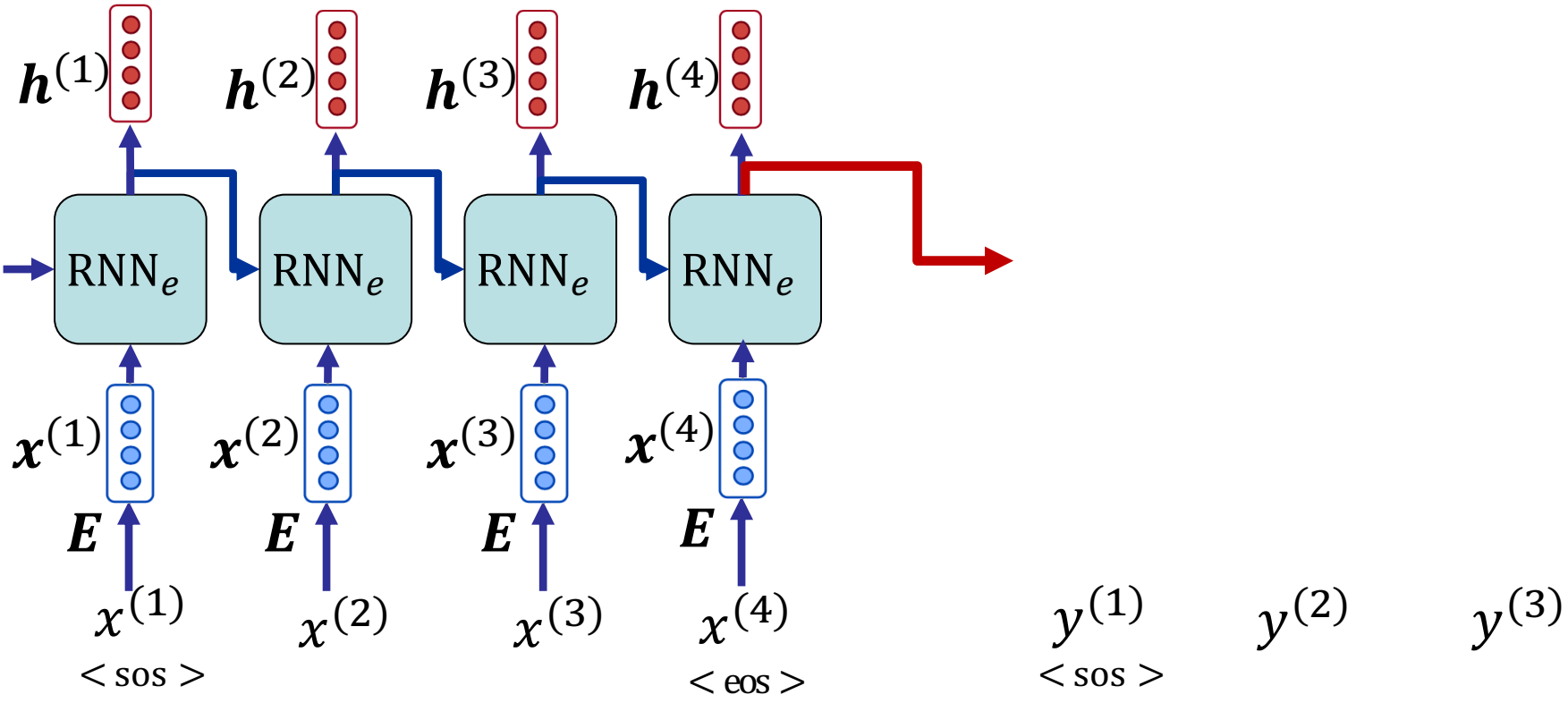
- Training a seq2seq is the same as training a Language Model
  - We predict the next word, calculate loss, backpropagate, and update parameters
  - Since seq2seq is an end-to-end model, gradient flows from loss to all parameters (both RNNs and embeddings)

- Loss function: Negative Log Likelihood of the predicted probability of the correct next target word $y^{(t+1)}$

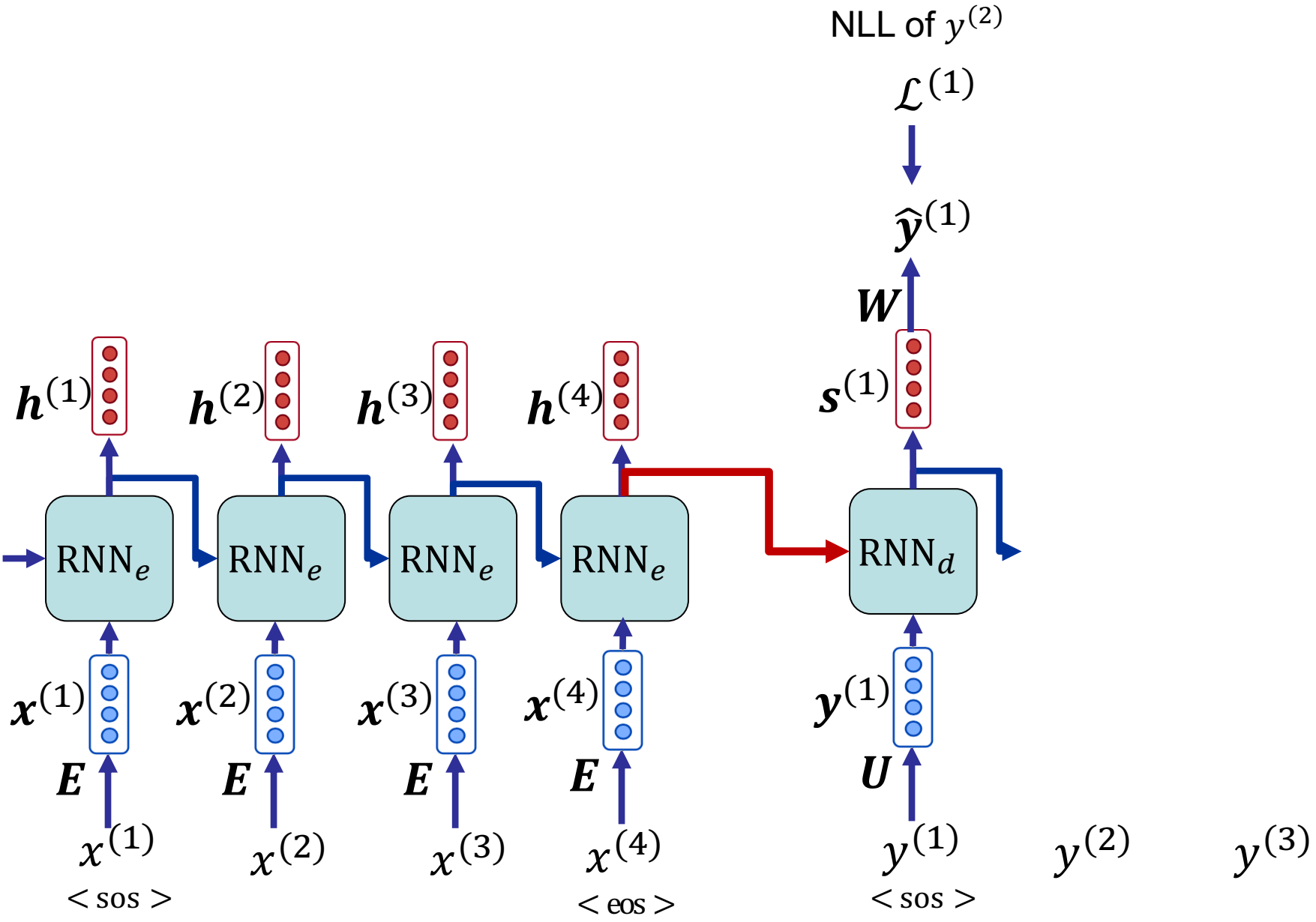
$$\mathcal{L}^{(t)} = -\log \hat{y}^{(t)}_{y^{(t+1)}} = -\log P\left(y^{(t+1)} \middle| X, y^{(1)}, \dots, y^{(t)}\right)$$

- Overall loss: $\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{(t)}$

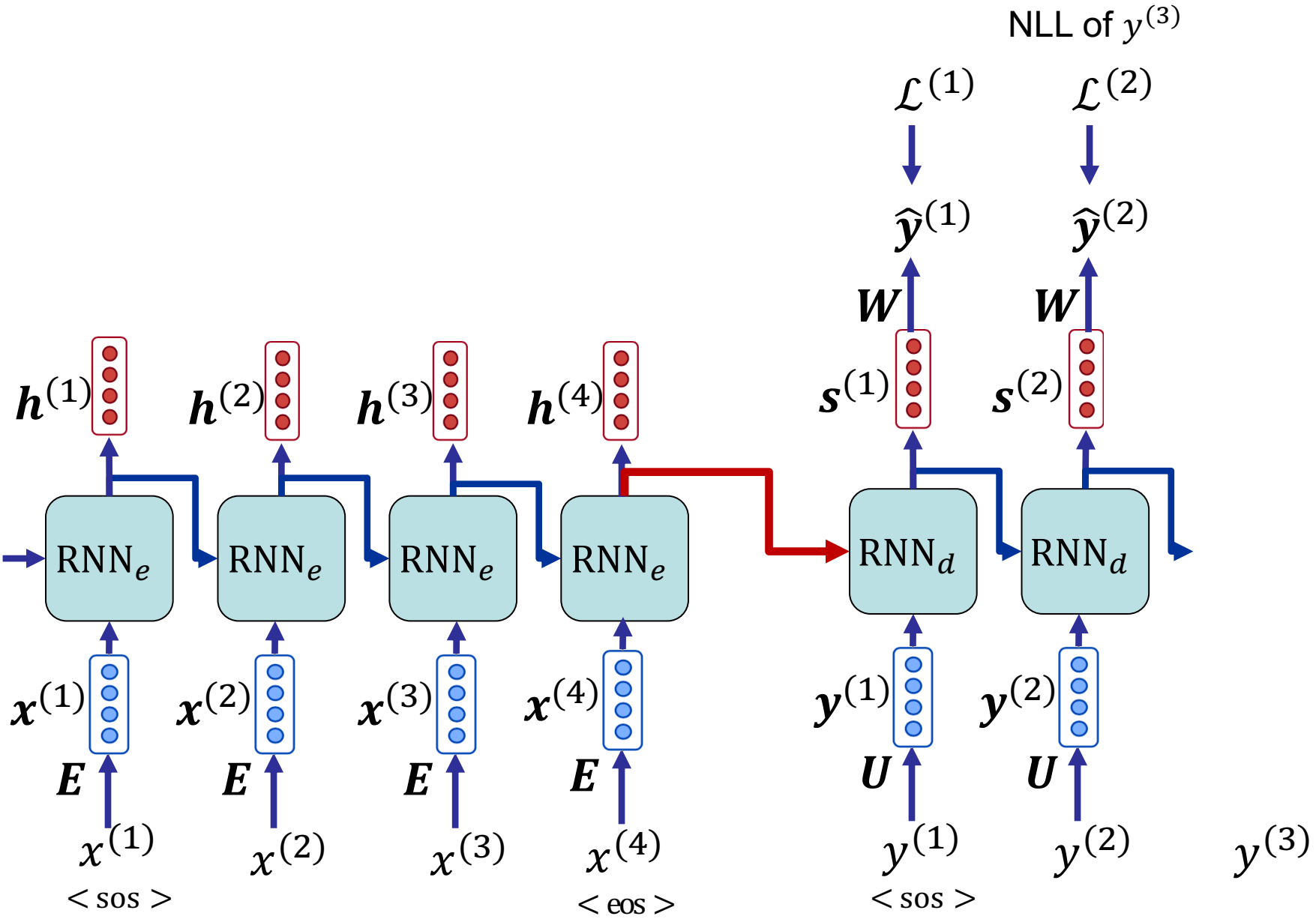
Training Seq2seq



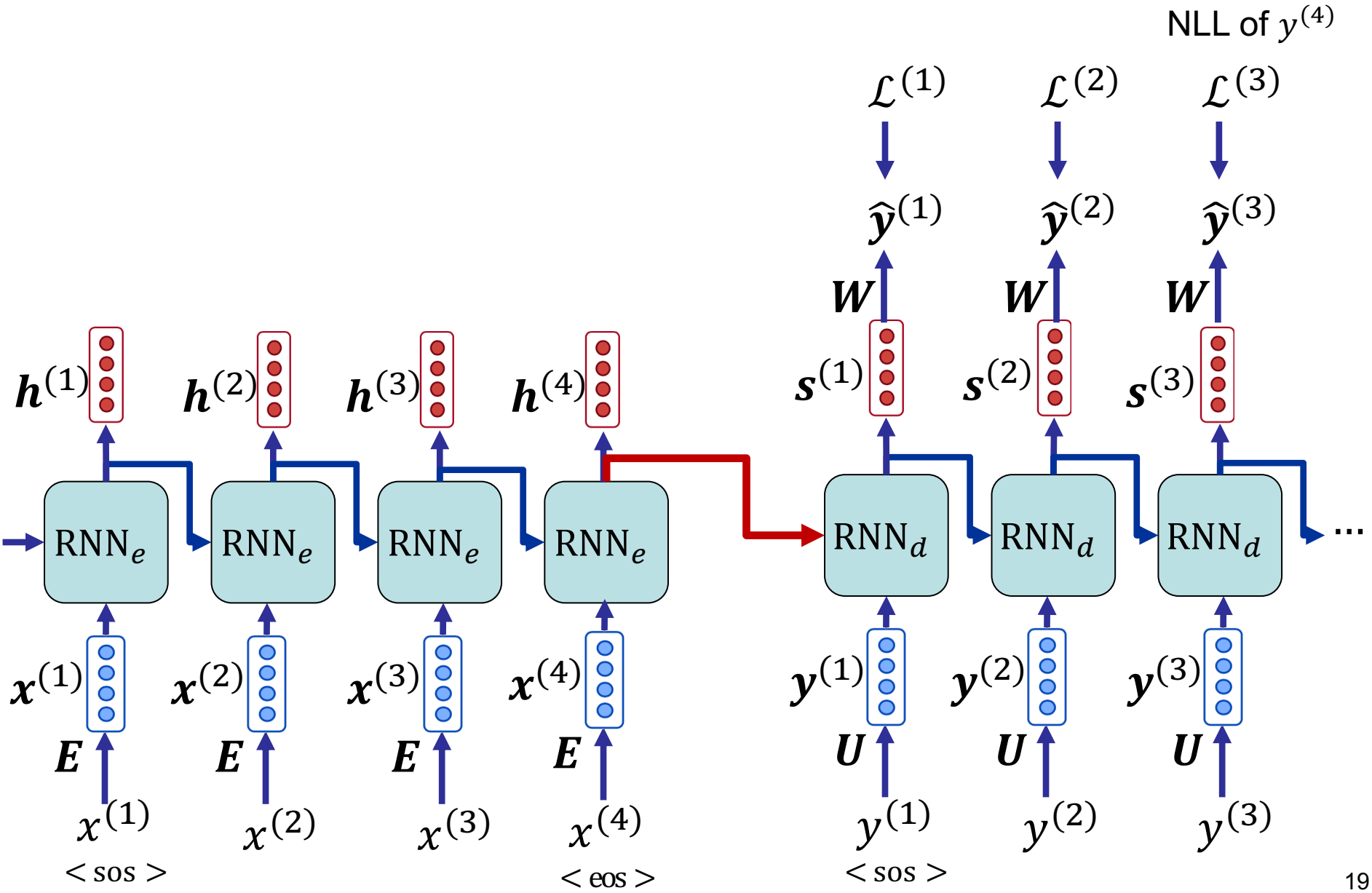
Training Seq2seq



Training Seq2seq



Training Seq2seq



Parameters

- Encoder embeddings $\boldsymbol{E} \rightarrow |\mathbb{V}_e| \times d_e$
- Encoder RNN parameters
- Decoder embeddings $\boldsymbol{U} \rightarrow |\mathbb{V}_d| \times d_u$
- Decoder RNN parameters
- Decoder output projection $\boldsymbol{W} \rightarrow d_w \times |\mathbb{V}_d|$

- bias terms are discarded
- d_e, d_u, d_w are embedding dimensions
- RNNs can be an LSTM, GRU, or vanilla (Elman) RNN

Practical points: vocabs & embeddings

- In Machine Translation
  - Encoder and decoder vocabularies belong to two different languages
- In summarization
  - Encoder and decoder vocabularies are typically the same set (as they are in the same language)
  - Encoder and decoder embeddings ($\boldsymbol{E}$ and $\boldsymbol{U}$) can also share parameters
- Weight tying
  - can be done by sharing the parameters of $\boldsymbol{U}$ and $\boldsymbol{W}$ in decoder

Decoding

Recap

- After training, we use the model to generate a target sequence given the source sequence (decoding). We aim to find the optimal output sequence Y^* that maximizes $P(Y|X)$:

$$Y^* = \underset{Y}{\operatorname{argmax}}\, P(Y|X)$$

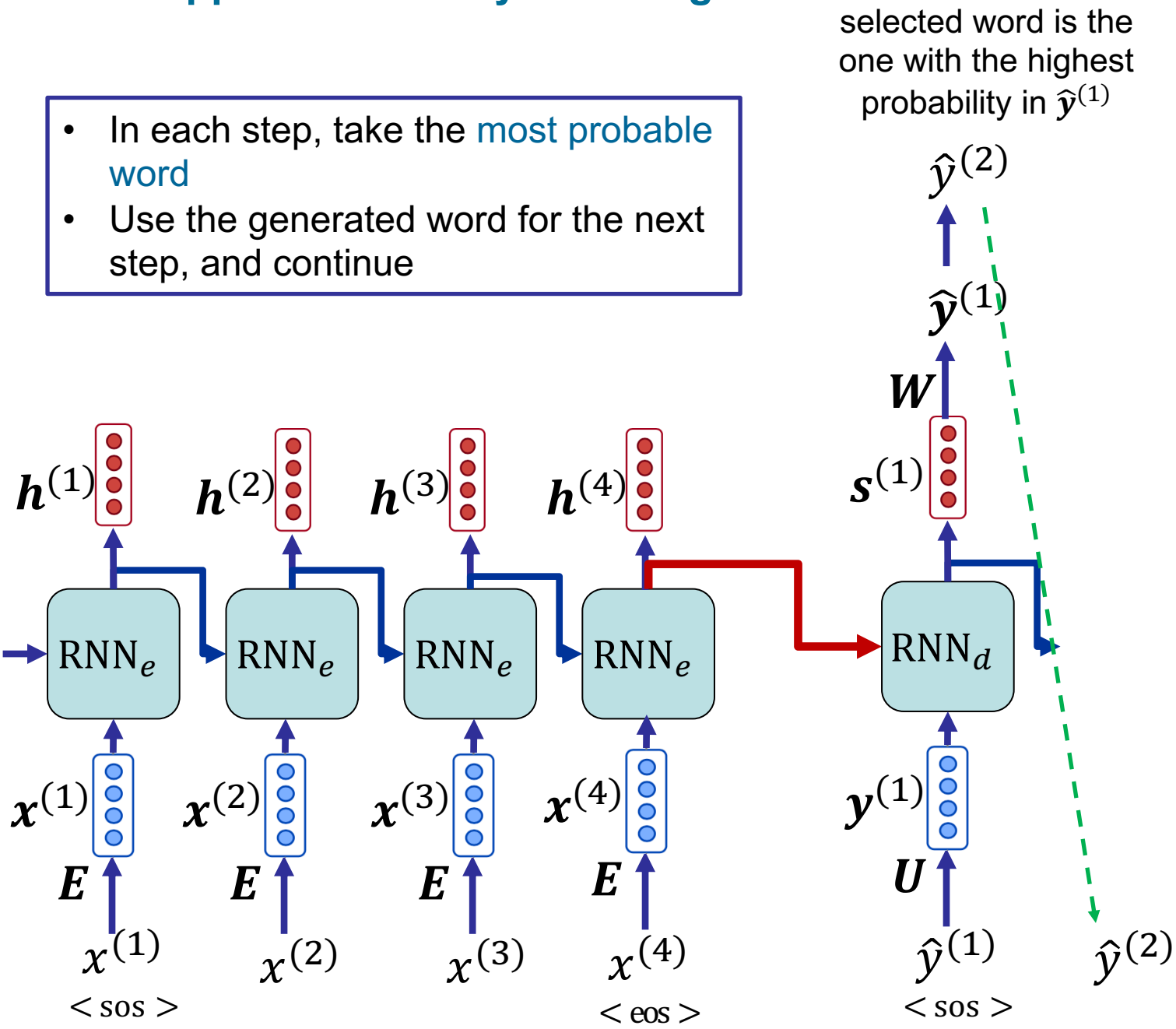
where $P(Y|X)$ for any arbitrary $Y = \{y^{(1)}, y^{(2)}, \dots, y^{(T)}\}$ is:

$$P(Y|X) = \prod_{t=1}^{T} P(y^{(t)}|X, y^{(1)}, \dots, y^{(t-1)})$$

- *Question: among all possible Y sequences, how can we find Y^*?*

A first approach: Greedy decoding

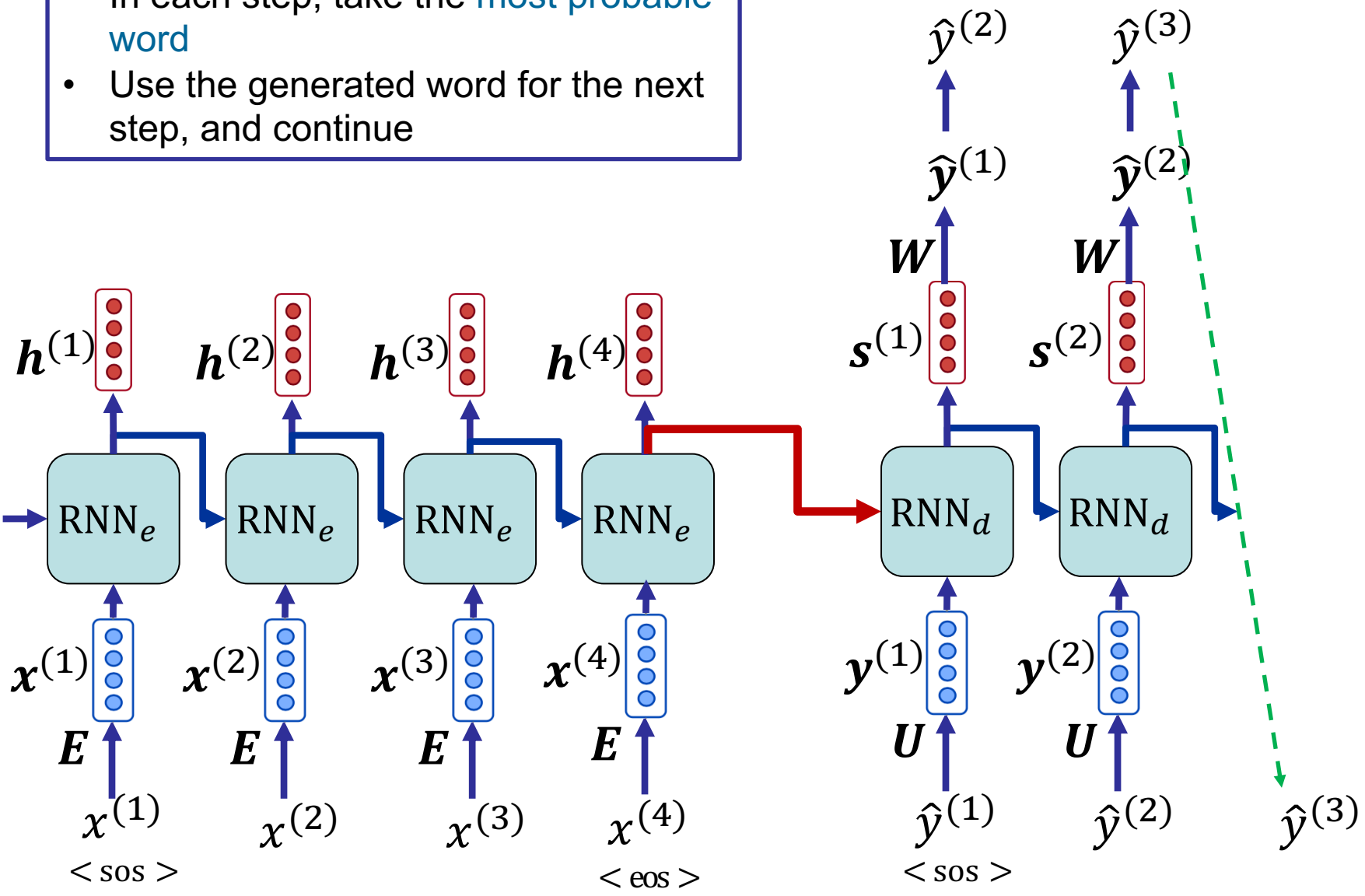
- In each step, take the most probable word
- Use the generated word for the next step, and continue



A first approach: Greedy decoding

- In each step, take the most probable word
- Use the generated word for the next step, and continue

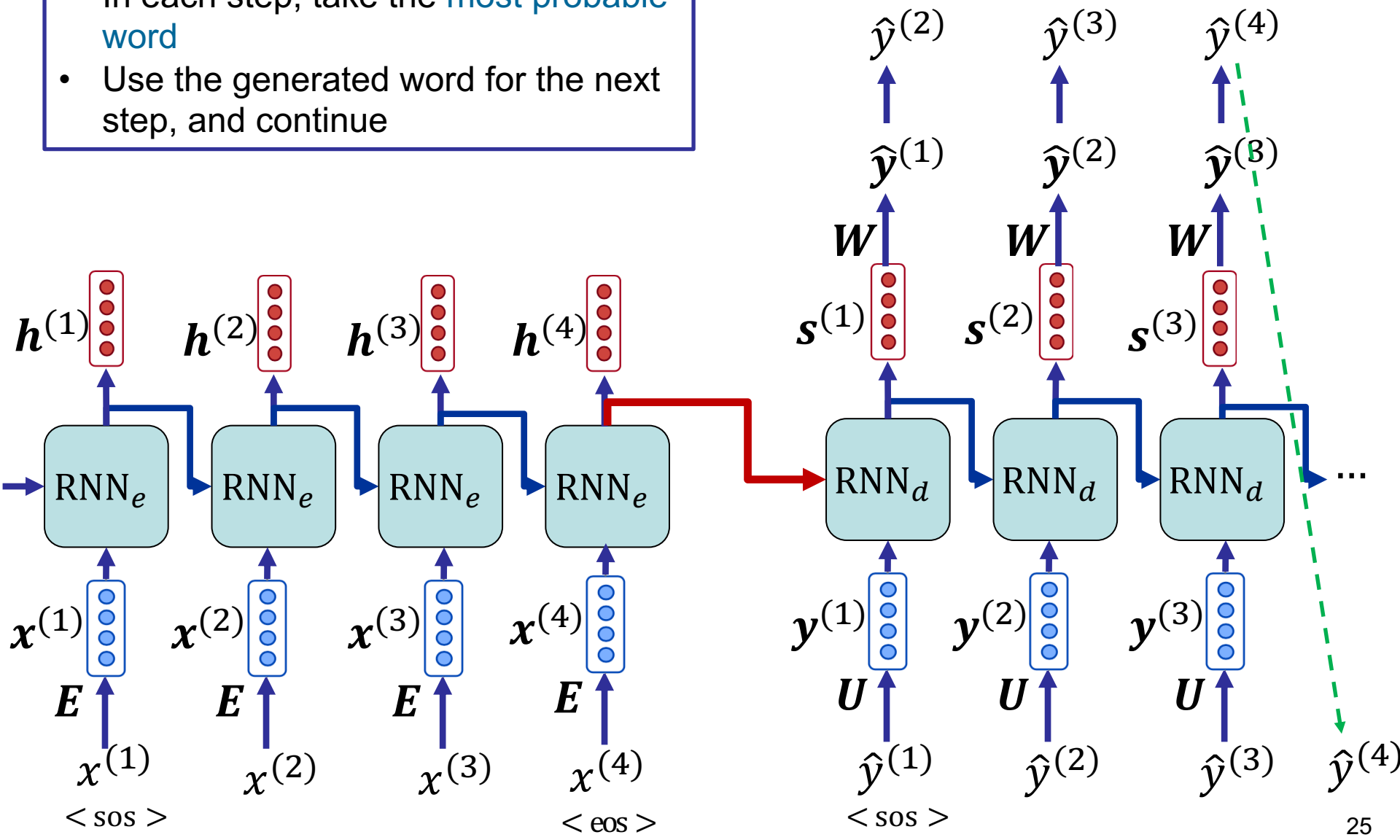
selected word is the one with the highest probability in $\hat{y}^{(2)}$



A first approach: Greedy decoding

- In each step, take the most probable word
- Use the generated word for the next step, and continue

selected word is the one with the highest probability in $\hat{y}^{(3)}$



Decoding

- Greedy decoding
  - Fast but …
  - … decisions are only based on immediate local knowledge
  - A non-optimal local decision can get propagated
  - It does not explore other decoding possibilities

- Exhaustive search decoding
  - We *can* compute all possible decodings
  - It means a decoding tree with $|\mathbb{V}_d| \times T$ leaves!
  - Far too expensive!

- Beam search decoding
  - A compromise between exploration and exploitation!

Beam search decoding

- Core idea: on each time step of decoding, keep only k most probable intermediary sequences (hypotheses)
  - k is the beam size (in practice around 5 to 10)

- To do it, beam search calculates of the following score for each hypothesis till time step l (denoted as $y^{(1\dots l)}$) :

$$\text{score}\left(y^{(1\dots l)}\right) = \log P\left(y^{(1\dots l)}|X\right) = \sum_{i=1}^{l} \log P(y^{(i)}|X, y^{(1)}, \dots, y^{(i-1)})$$

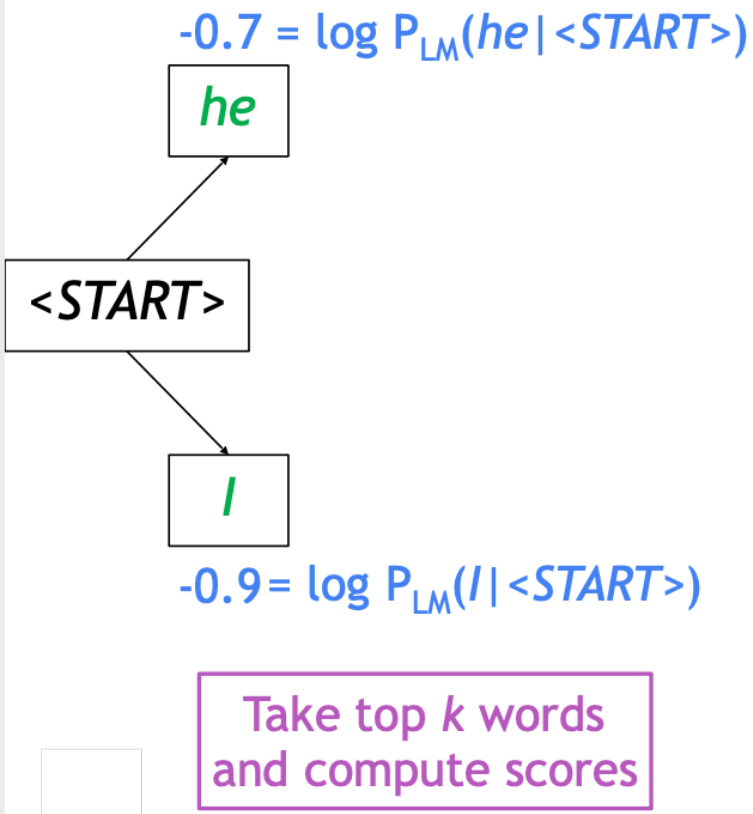
- In each decoding step, we only keep k hypotheses with the highest scores, and don't continue the rest

Beam search decoding – example

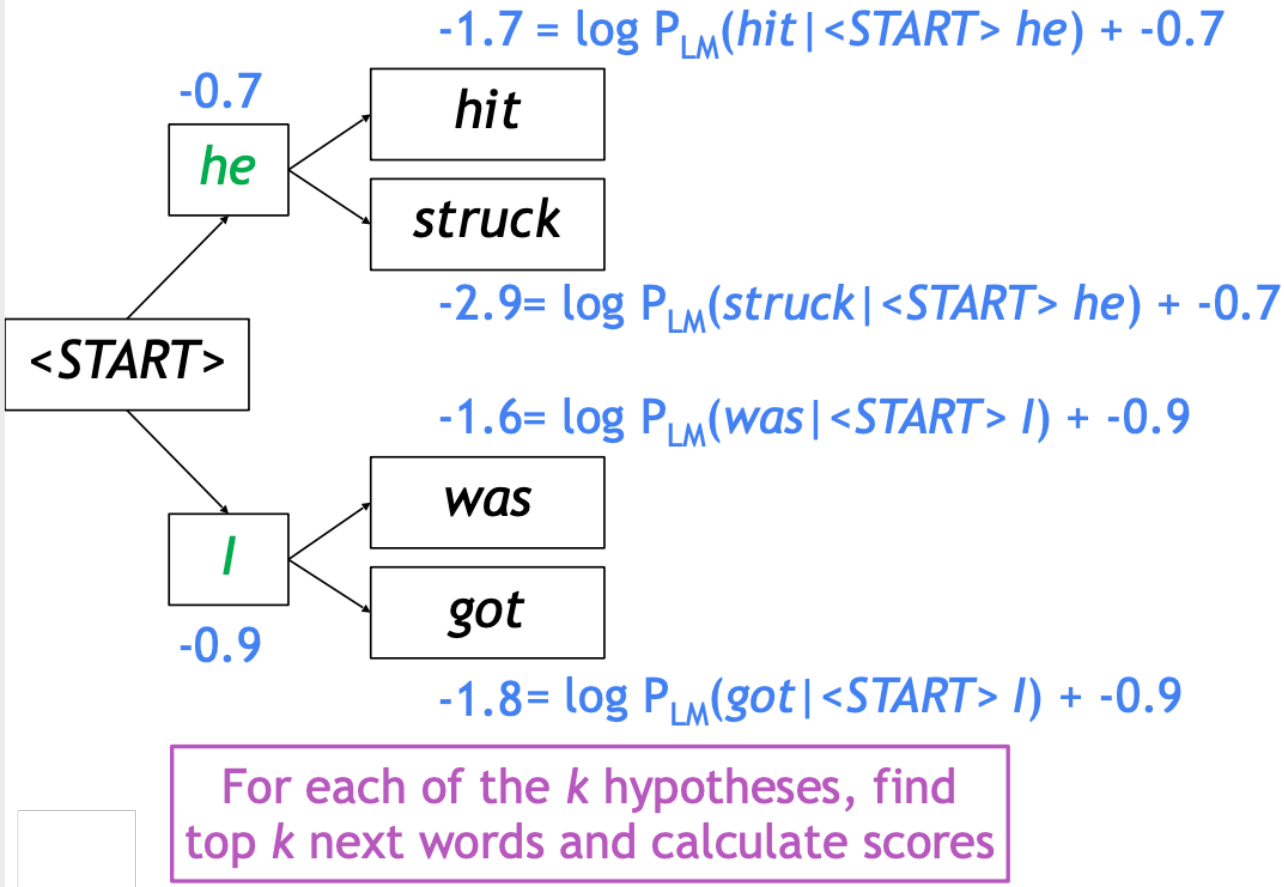
<START>

Calculate prob dist of next word

Beam search decoding – example

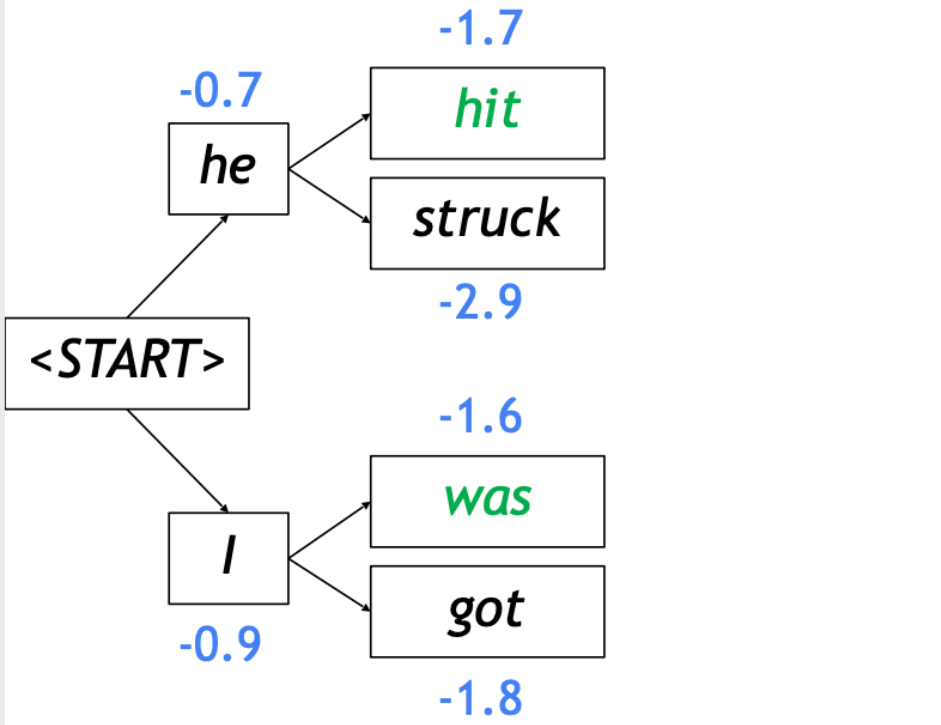


Beam search decoding – example



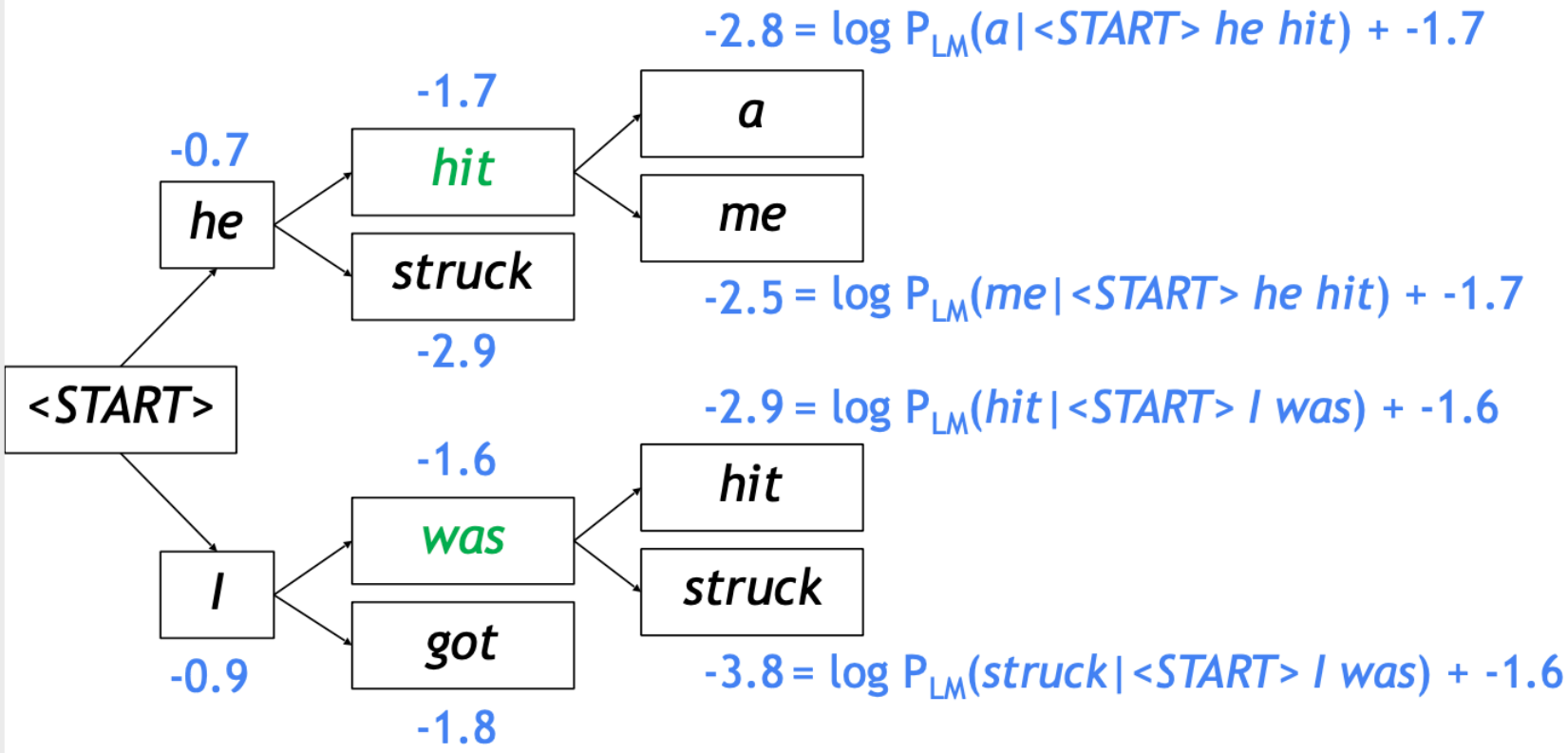
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



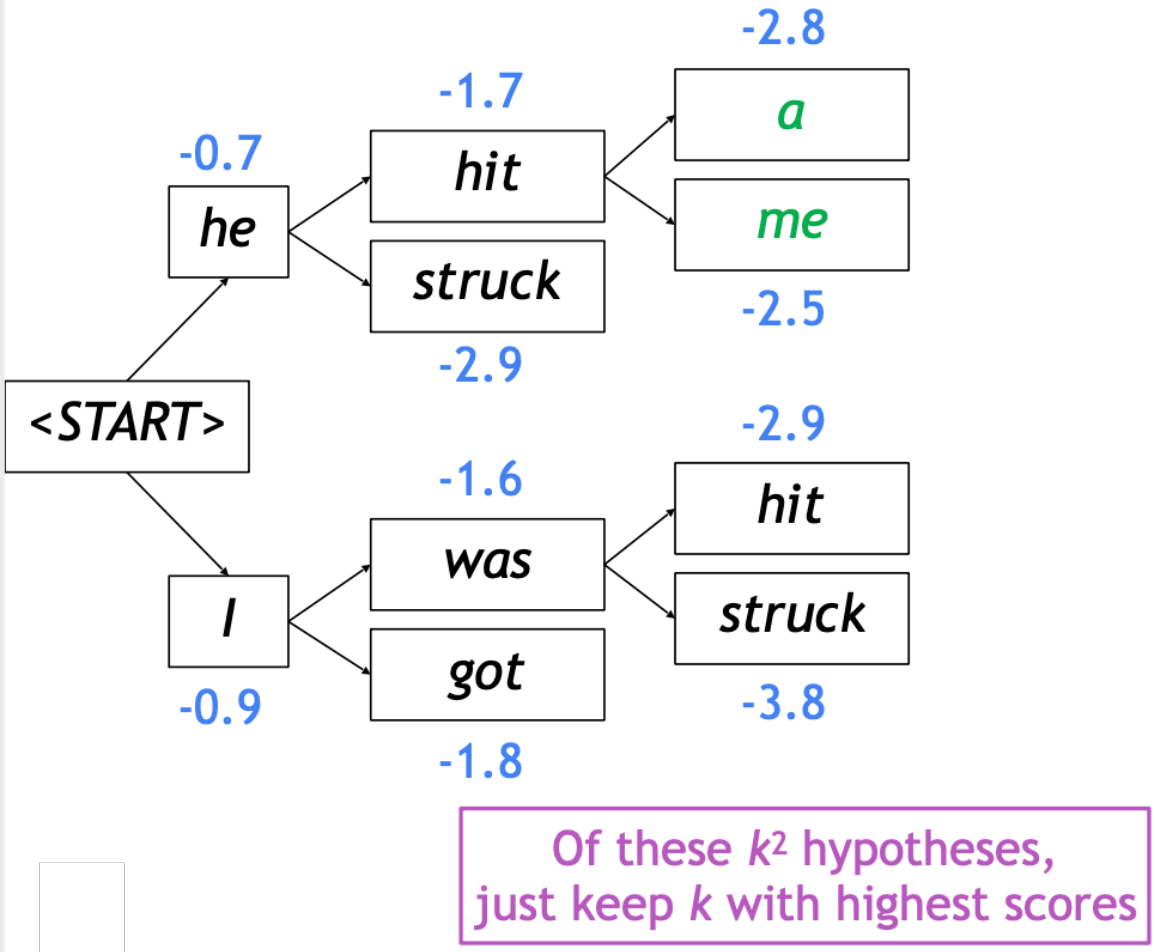
Of these k^2 hypotheses,
just keep k with highest scores

Beam search decoding – example



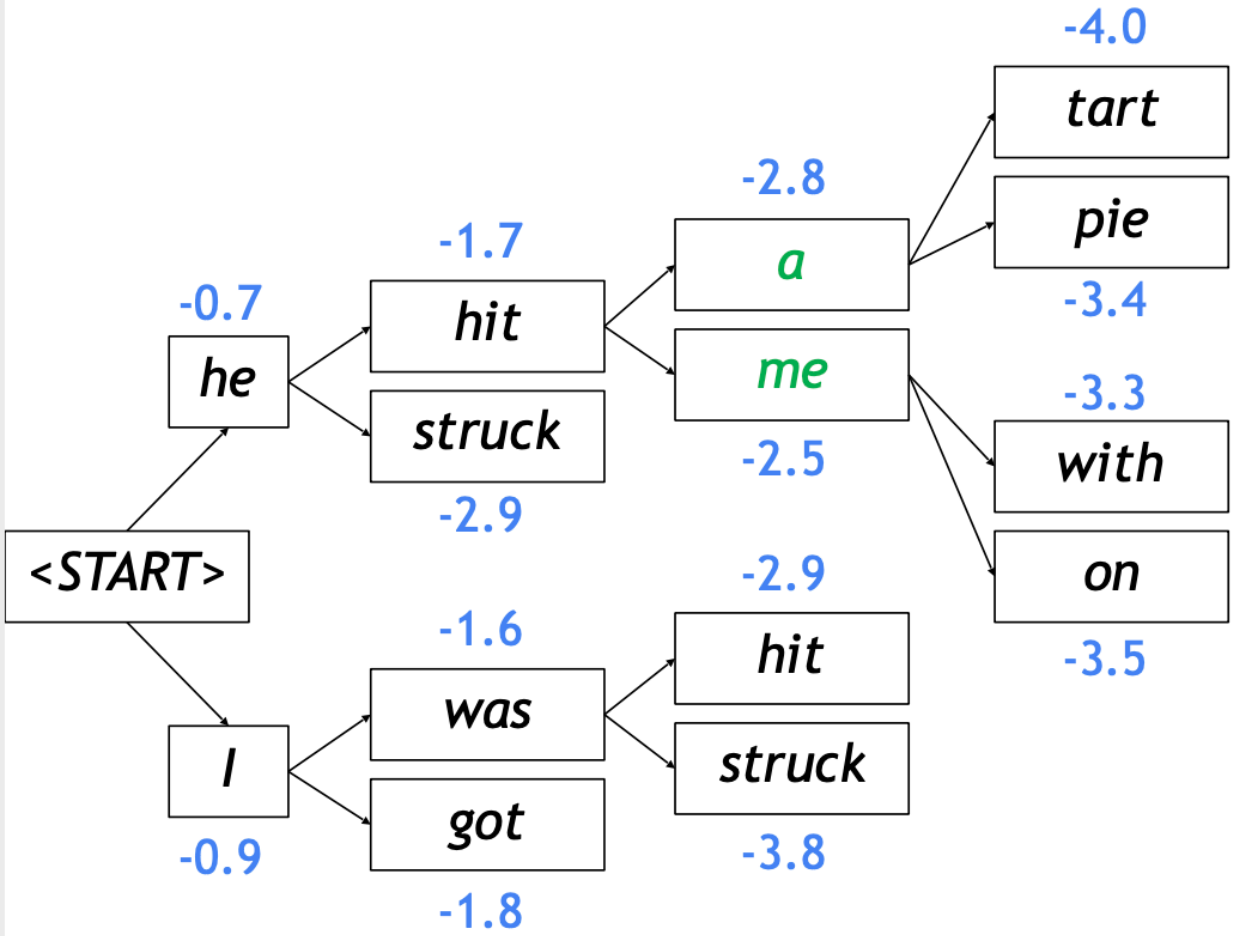
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



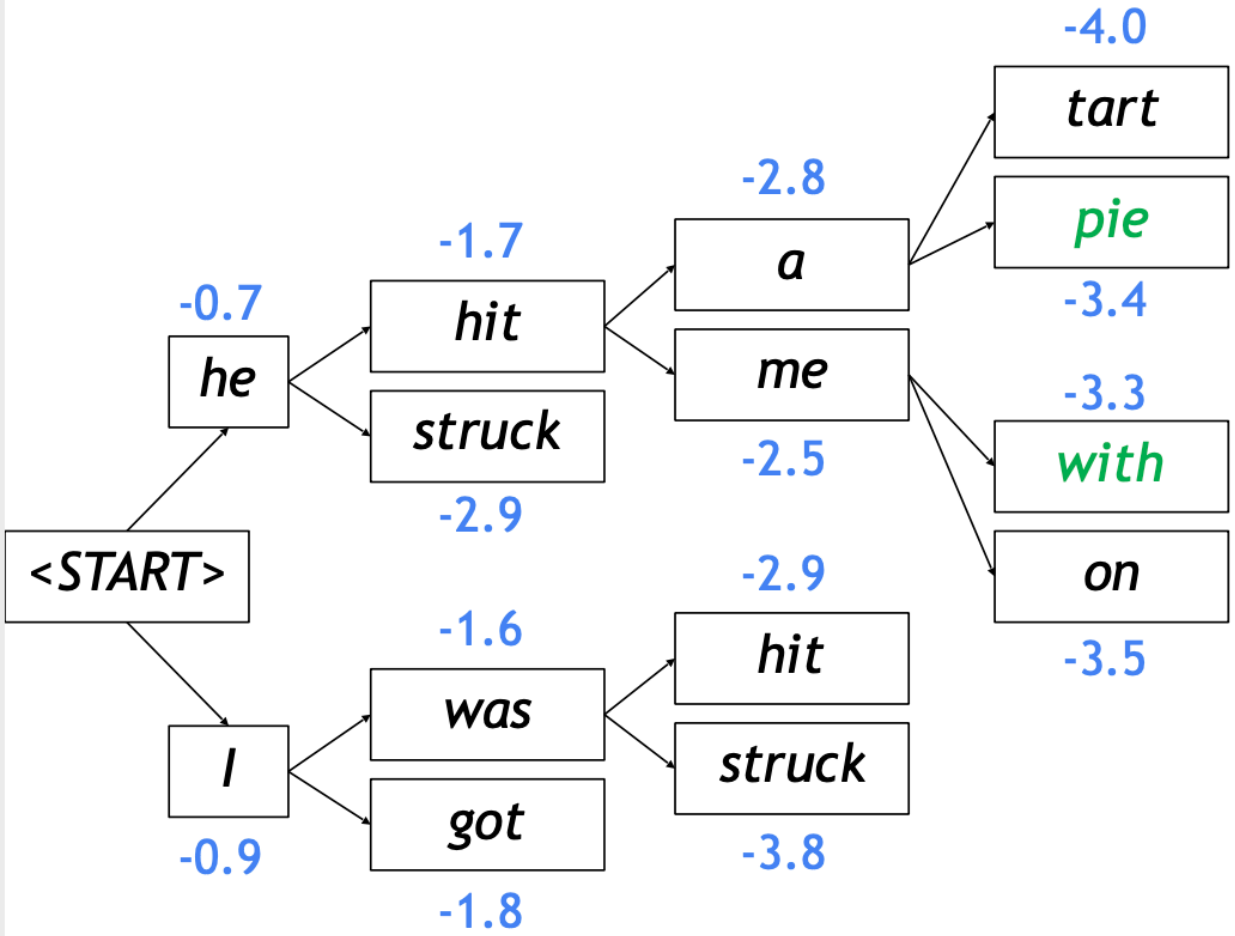
Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding – example



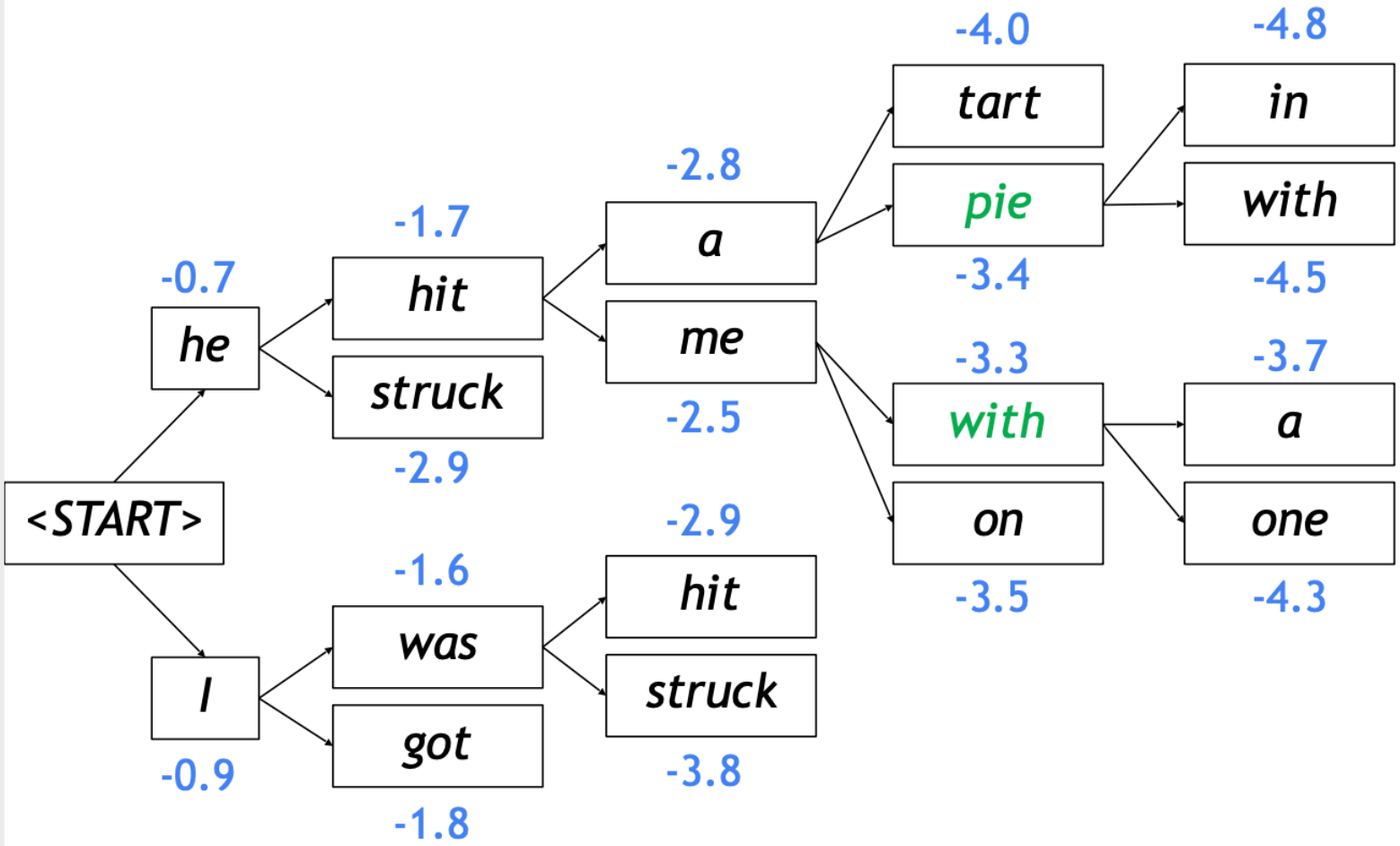
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



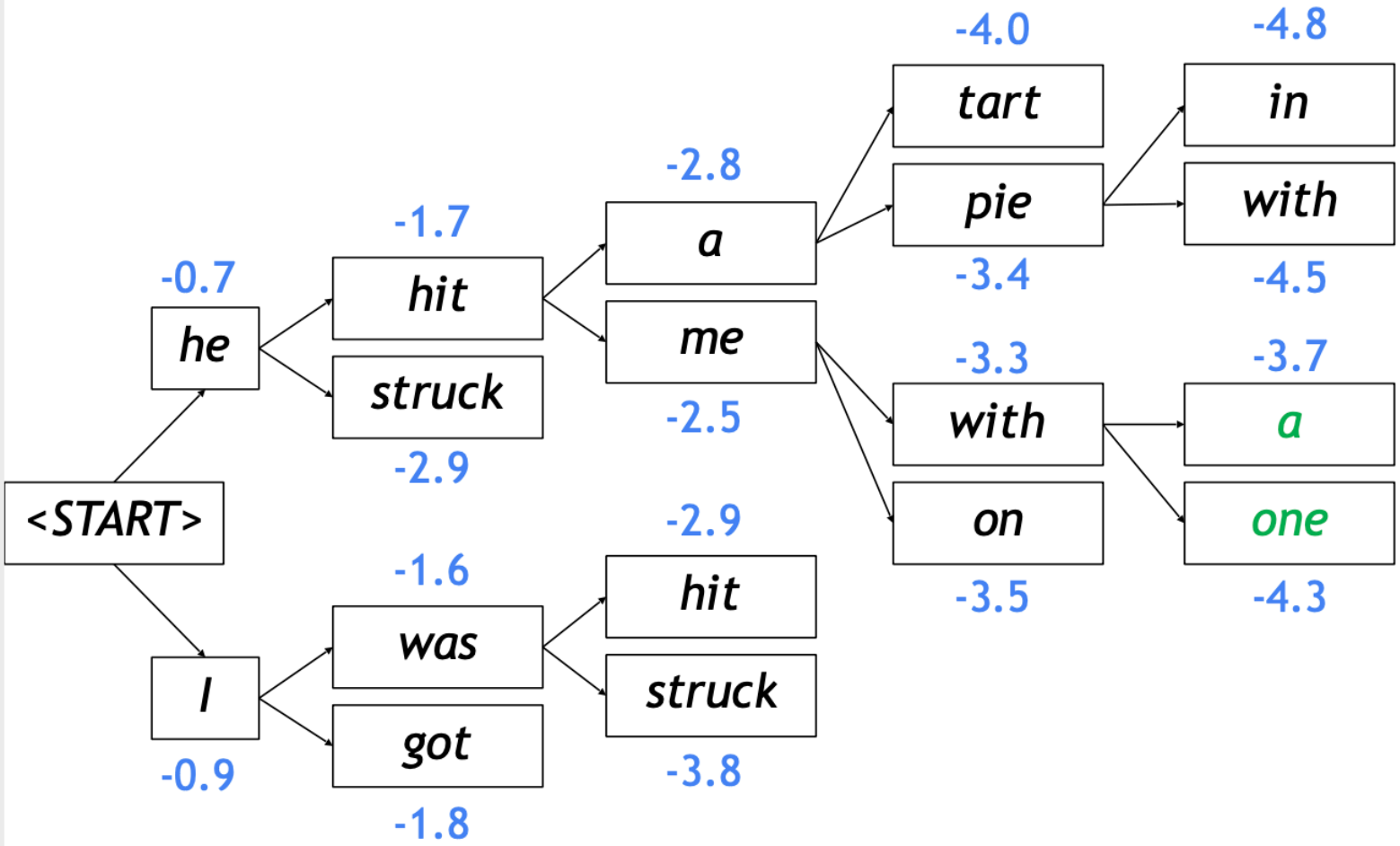
Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding – example



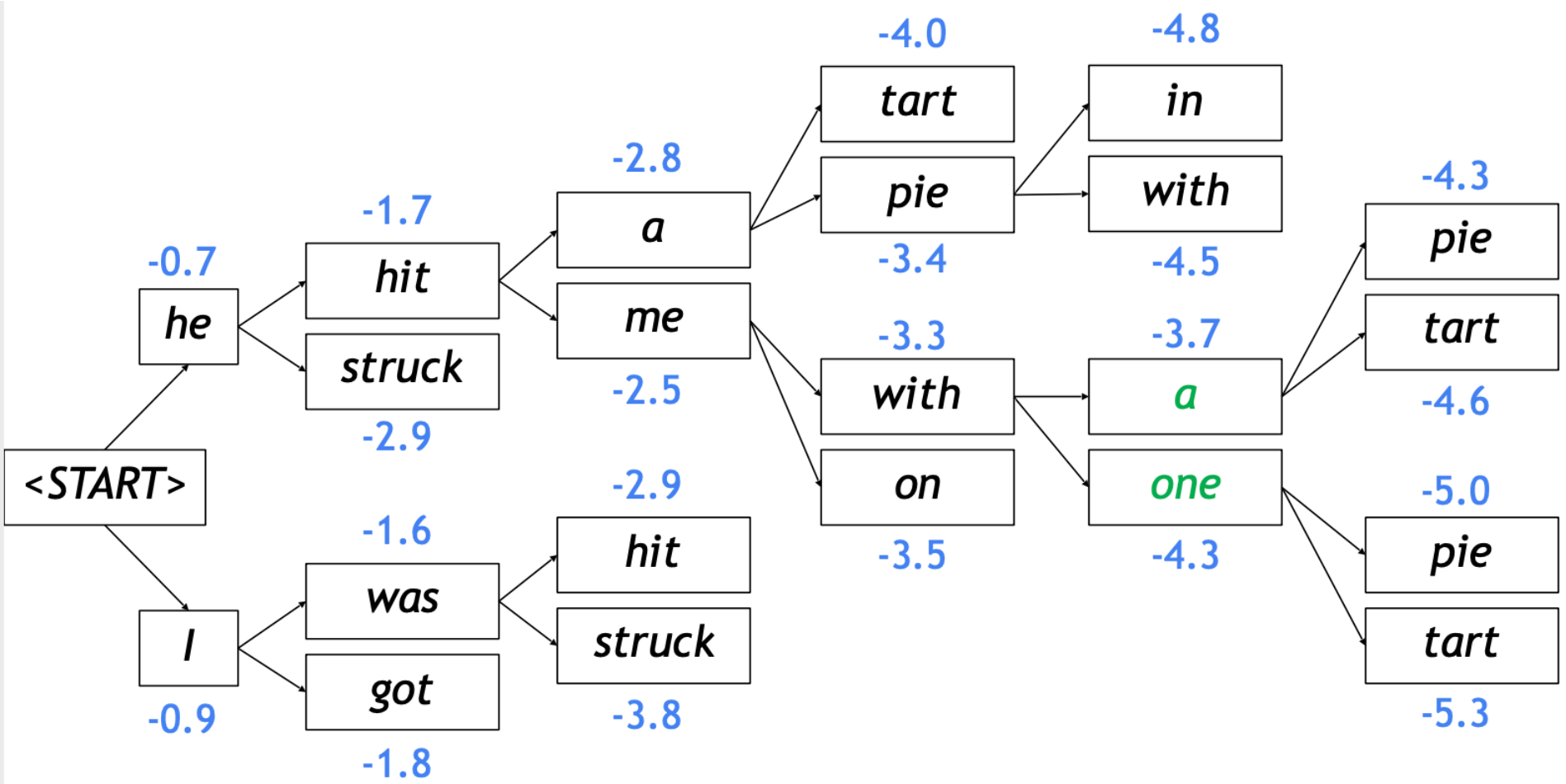
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



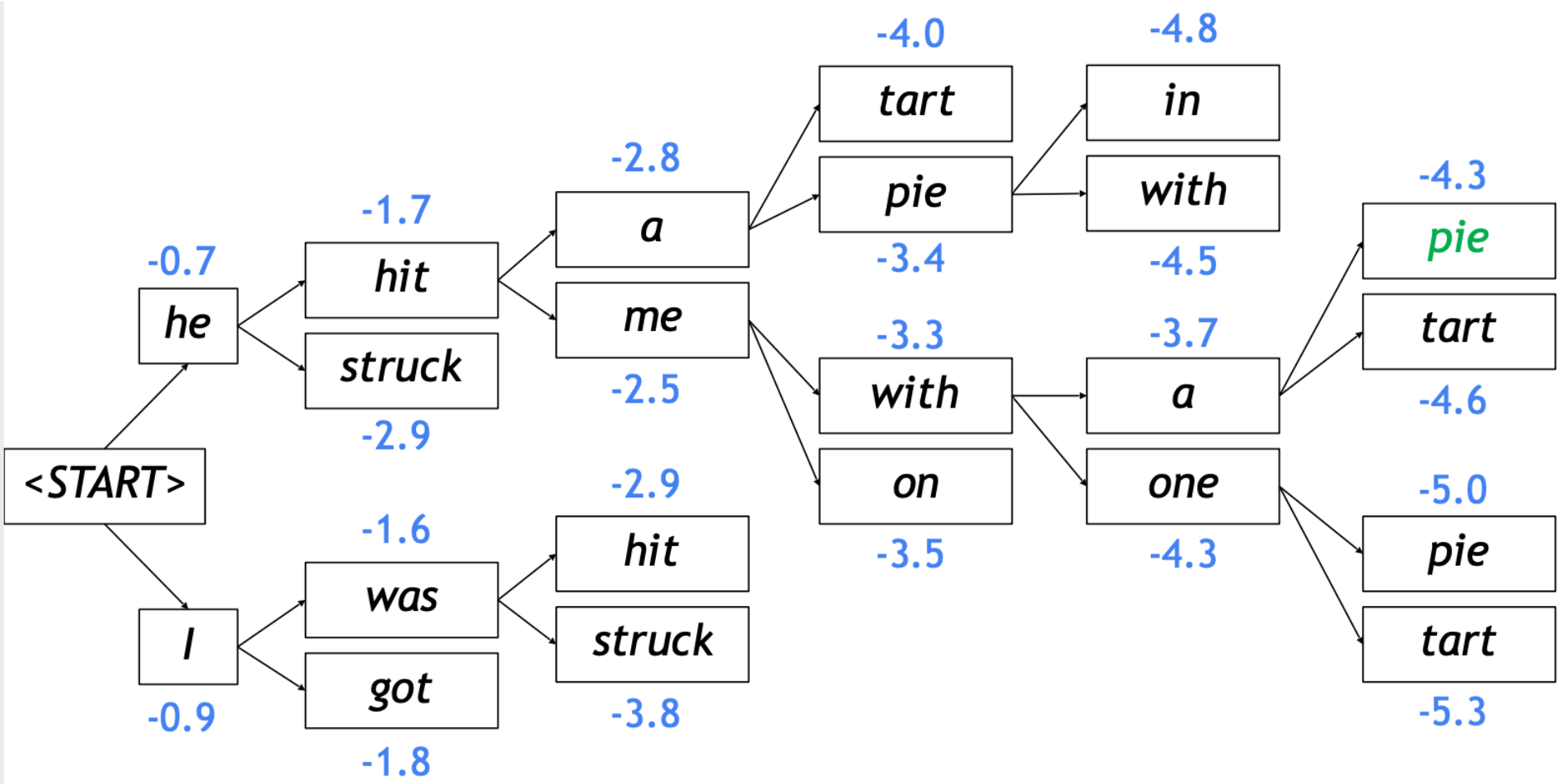
Of these k^2 hypotheses, just keep k with highest scores

Beam search decoding – example



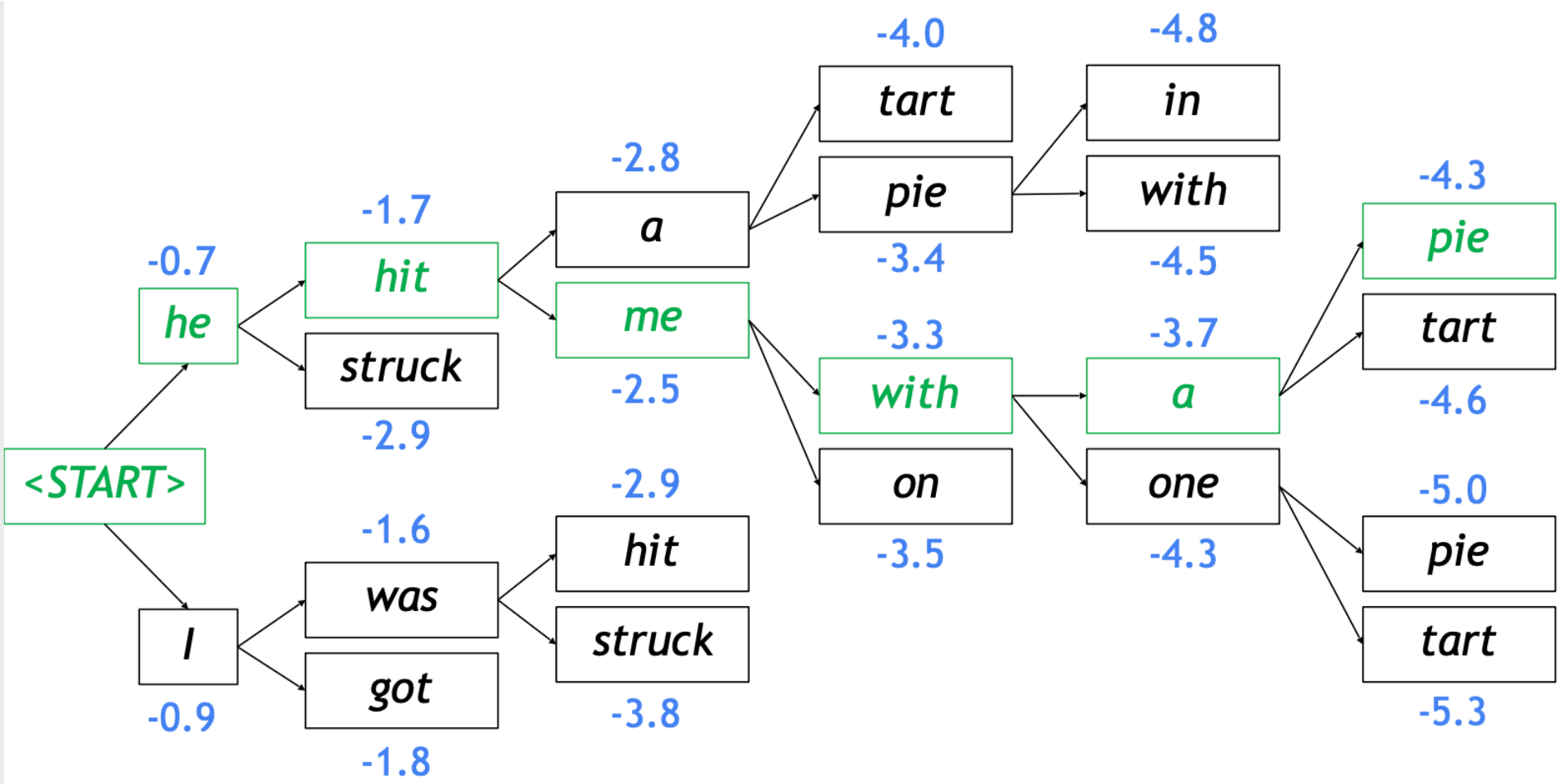
For each of the *k* hypotheses, find top *k* next words and calculate scores

Beam search decoding – example



This is the top-scoring hypothesis!

Beam search decoding – example



Backtrack to obtain the full hypothesis

Beam search decoding – last words!

- Achieving the optimal solution is not guaranteed …
  - ... but it is much more efficient than exhaustive search decoding

- Stopping criteria:
  - Each hypothesis continues till reaching the <eos> token
  - Usually beam search decoding continues until:
    - We reach a cutoff timestep T (a hyperparameter), or
    - We have at least n completed hypotheses (another hyperparameter)

Agenda

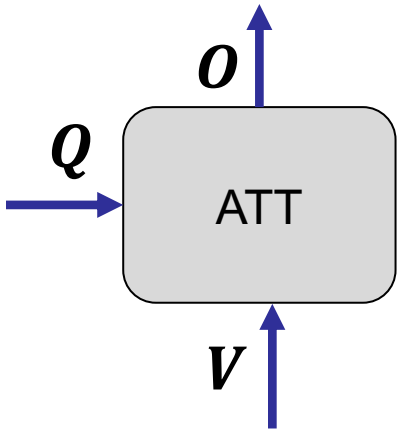
- Sequence-to-sequence models
- **Attention Mechanism**
- seq2seq with Attention

Attention Networks

- Attention is a general Deep Learning method to
  - obtain a composed representation (output) …
  - from an arbitrary size of representations (values) …
  - depending on a given representation (query)

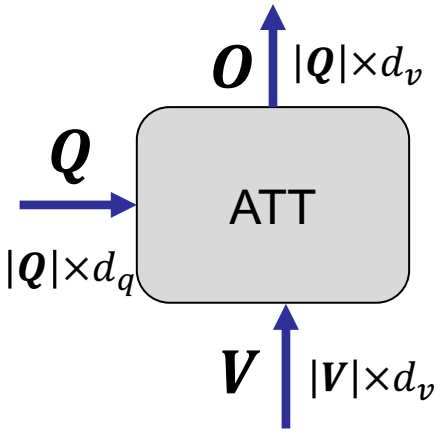
- General form of an attention network:

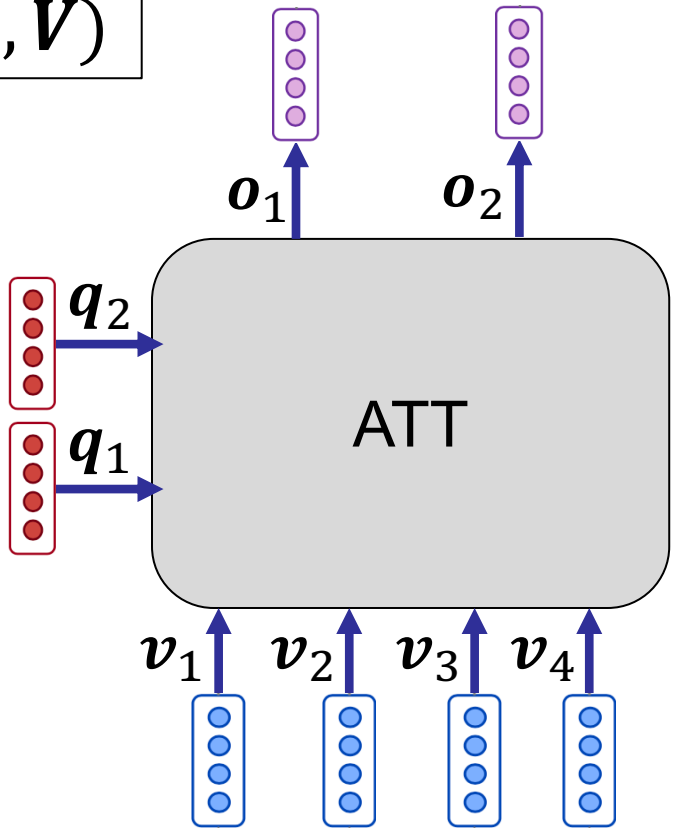
$$\boldsymbol{O} = \mathrm{ATT}(\boldsymbol{Q}, \boldsymbol{V})$$



Attention Networks

$$\boldsymbol{O} = \mathrm{ATT}(\boldsymbol{Q}, \boldsymbol{V})$$





- d_q, d_v are embedding dimensions of query and value vectors, respectively

We sometime say, each query vector $\boldsymbol{q}$ "*attends to*" the values

Attention Networks – definition

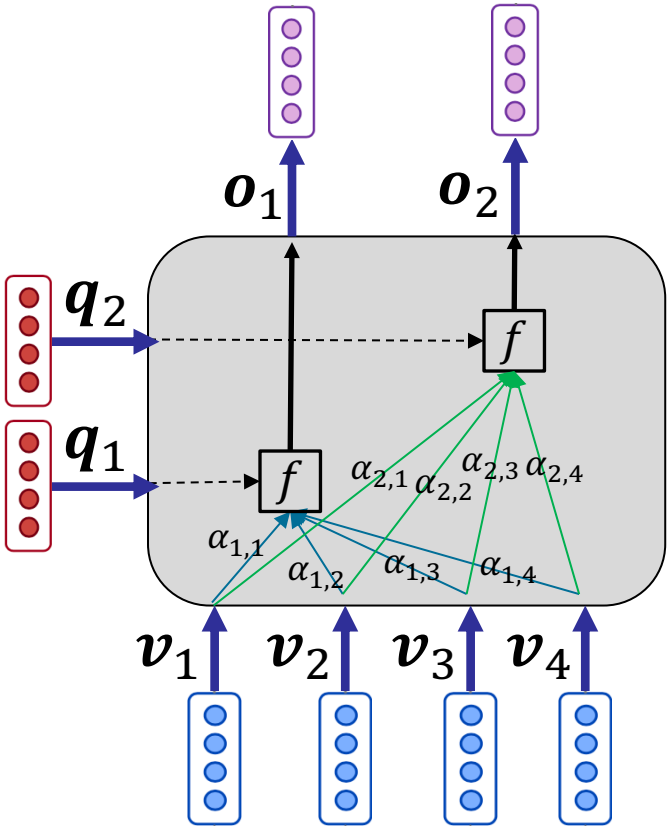
Formal definition:

- Given a set of vector *values* $\boldsymbol{V}$, and a set of vector *queries* $\boldsymbol{Q}$, **attention** is a technique to compute a weighted sum of the values, dependent on each query

- The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on

- The weight in the weighted sum – for each query on each value – is called attention, and denoted by α

Attentions!



$\alpha_{i,j}$ is the attention of query $\boldsymbol{q}_i$ on value $\boldsymbol{v}_j$

$\boldsymbol{\alpha}_i$ is the vector of attentions of query $\boldsymbol{q}_i$ on value vectors $\boldsymbol{V}$

$\boldsymbol{\alpha}_i$ is a probability distribution

f is attention function

Attention Networks – formulation

- Given the query vector $\boldsymbol{q}_i$, an attention network assigns attention $\alpha_{i,j}$ to each value vector $\boldsymbol{v}_j$ using attention function f:

$$\alpha_{i,j} = f(\boldsymbol{q}_i, \boldsymbol{v}_j)$$

such that $\boldsymbol{\alpha}_i$ (vector of attentions for the ith query vector) forms a probability distribution

- The output regarding each query is the weighted sum of the value vectors (attentions as weights):

$$\boldsymbol{o}_i = \sum_{j=1}^{|\boldsymbol{V}|} \alpha_{i,j} \boldsymbol{v}_j$$

Attention variants

Basic dot-product attention

- First, non-normalized attention scores:

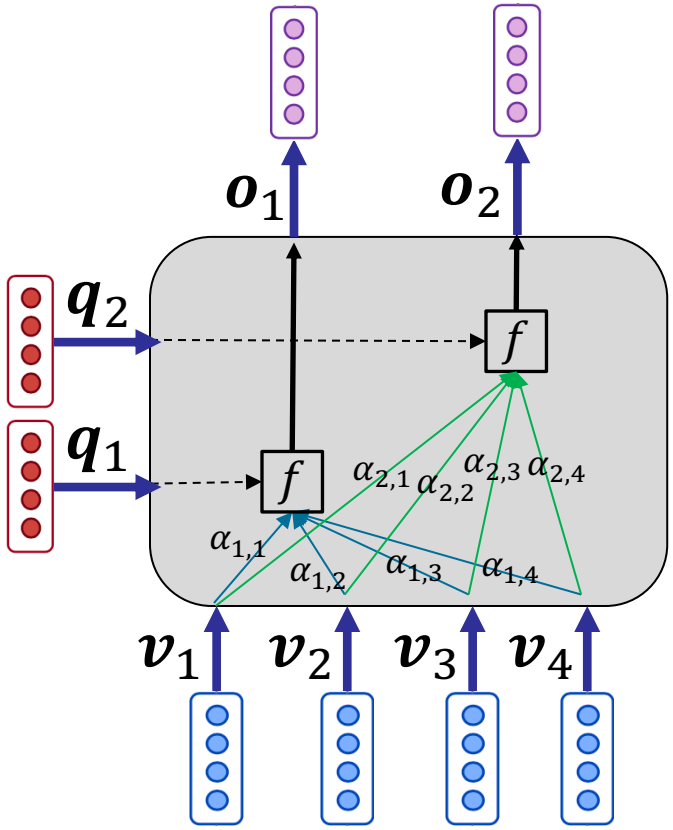
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i^{\mathrm{T}} \boldsymbol{v}_j$$

  - In this variant $d_q = d_v$
  - There is no parameter to learn!

- Then, softmax over values:

$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



Attention variants

Multiplicative attention

- First, non-normalized attention scores:

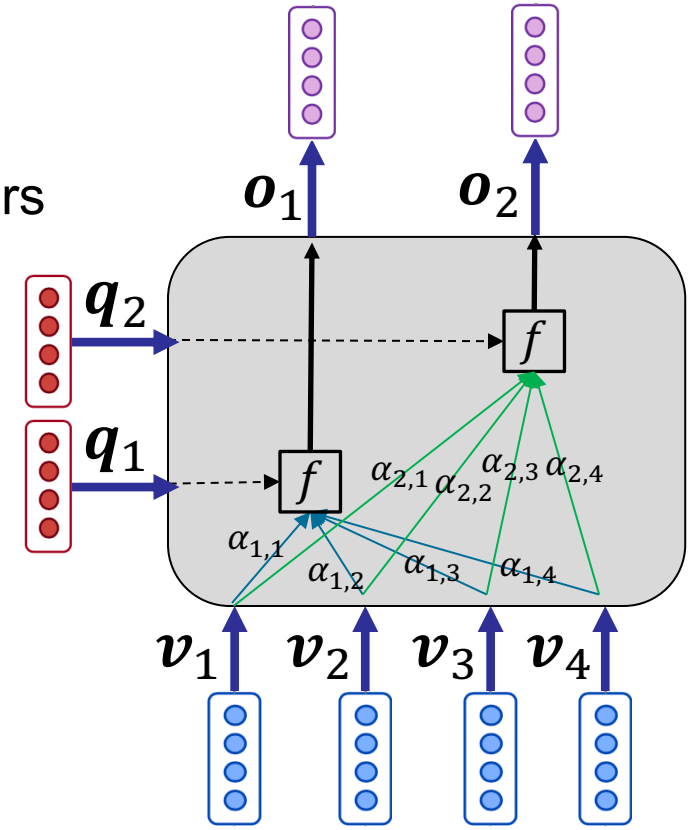
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i^{\mathrm{T}} \boldsymbol{W} \boldsymbol{v}_j$$

  - $\boldsymbol{W}$ is a matrix of model parameters
  - provides a linear function for measuring relations between query and value vectors

- Then, softmax over values:

$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



Attention variants

Additive attention

- First, non-normalized attention scores:

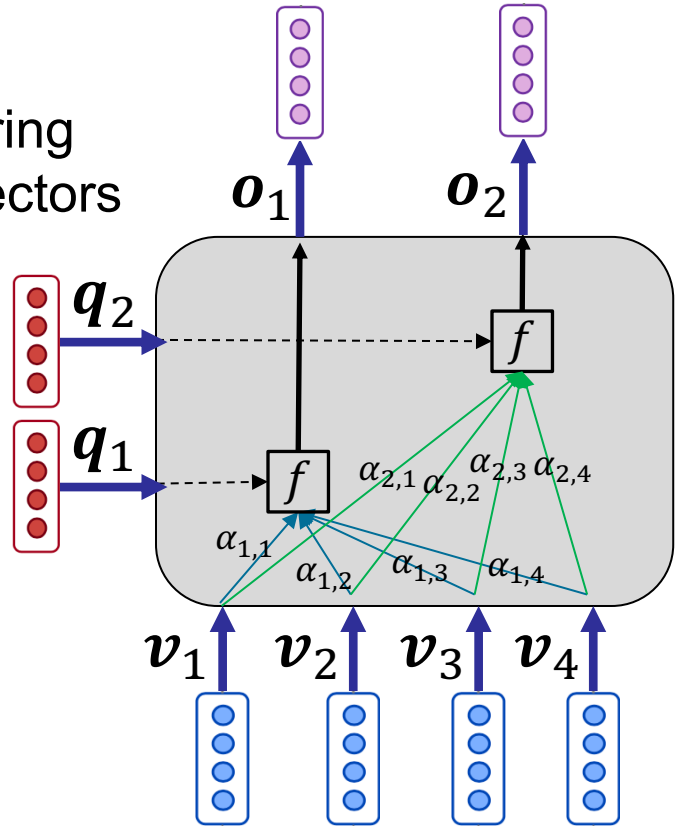
$$\tilde{\alpha}_{i,j} = \boldsymbol{u}^{\mathrm{T}} \tanh(\boldsymbol{q}_i \boldsymbol{W}_1 + \boldsymbol{v}_j \boldsymbol{W}_2)$$

  - $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, and $\boldsymbol{u}$ are model parameters
  - provides a non-linear function for measuring relations between the query and value vectors

- Then, softmax over values:

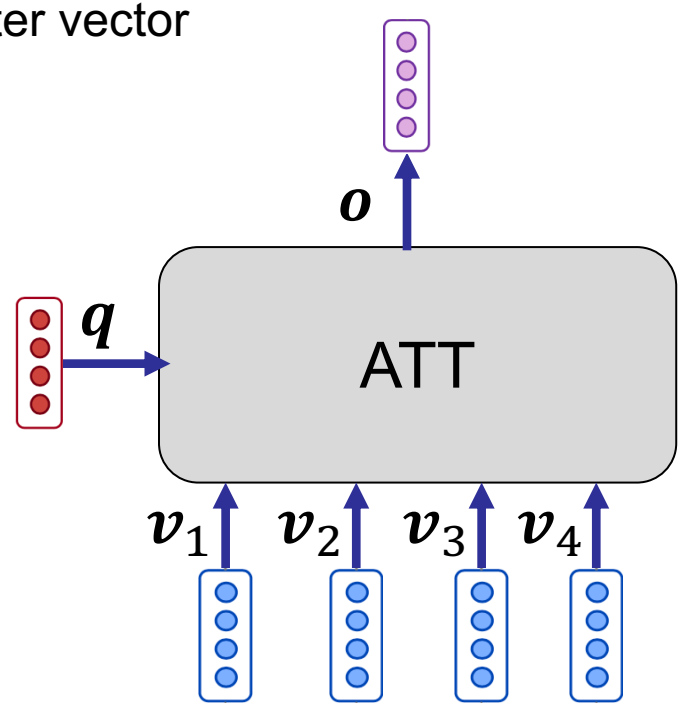
$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



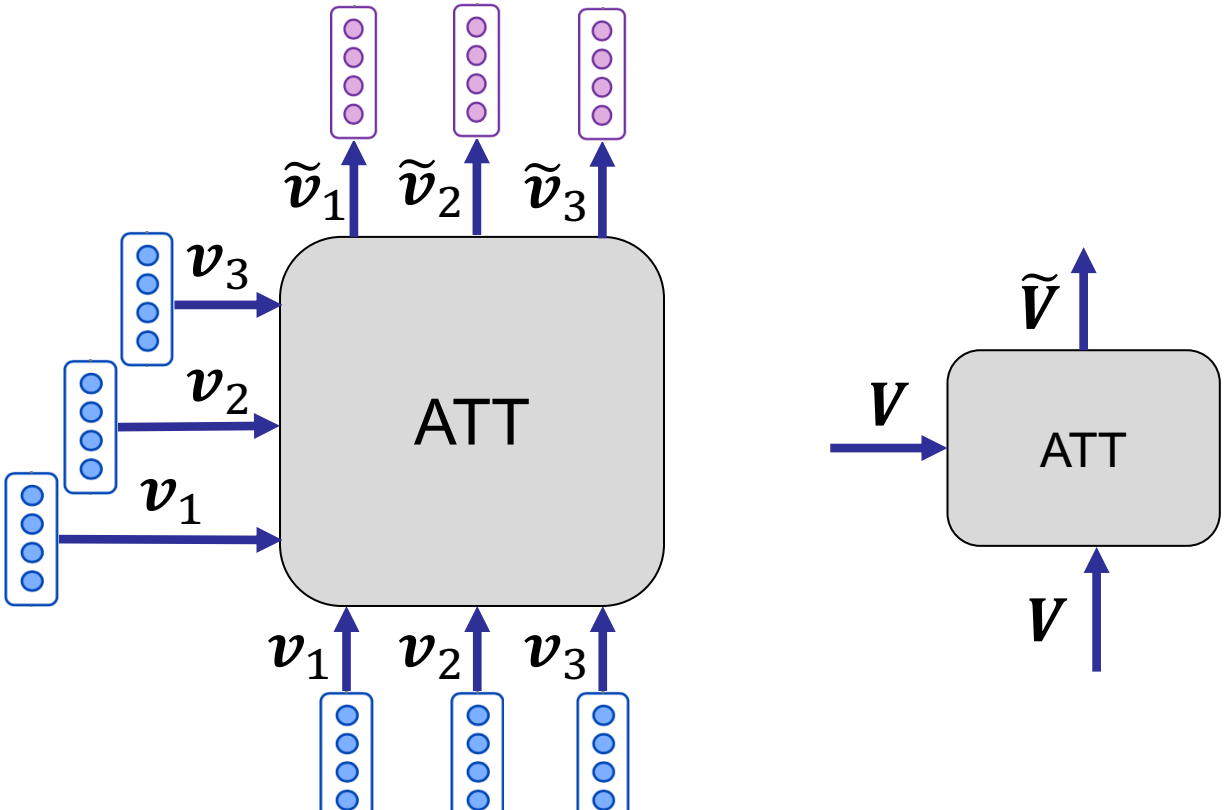
Attention in practice

- Attention is used to create a compositional embedding of value vectors, according to a query
  - E.g. in seq2seq models (comes next)
    - Where values are the vectors at encoding, and query is the current decoding state
  - or in document classification (Assignment 4)
    - Where values are document's word vectors, and query is a parameter vector



Self-attention

- In self-attention, values are the same as queries: $\boldsymbol{Q} = \boldsymbol{V}$
- Mainly used to encode a sequence $\boldsymbol{V}$ to another sequence $\tilde{\boldsymbol{V}}$
- Each encoded vector is a contextual embedding of the corresponding input vector
  - $\tilde{\boldsymbol{v}}_i$ is the contextual embedding of $\boldsymbol{v}_i$



Attention – summary

- Attention is a way to focus on particular parts of the input, and create a compositional embedding

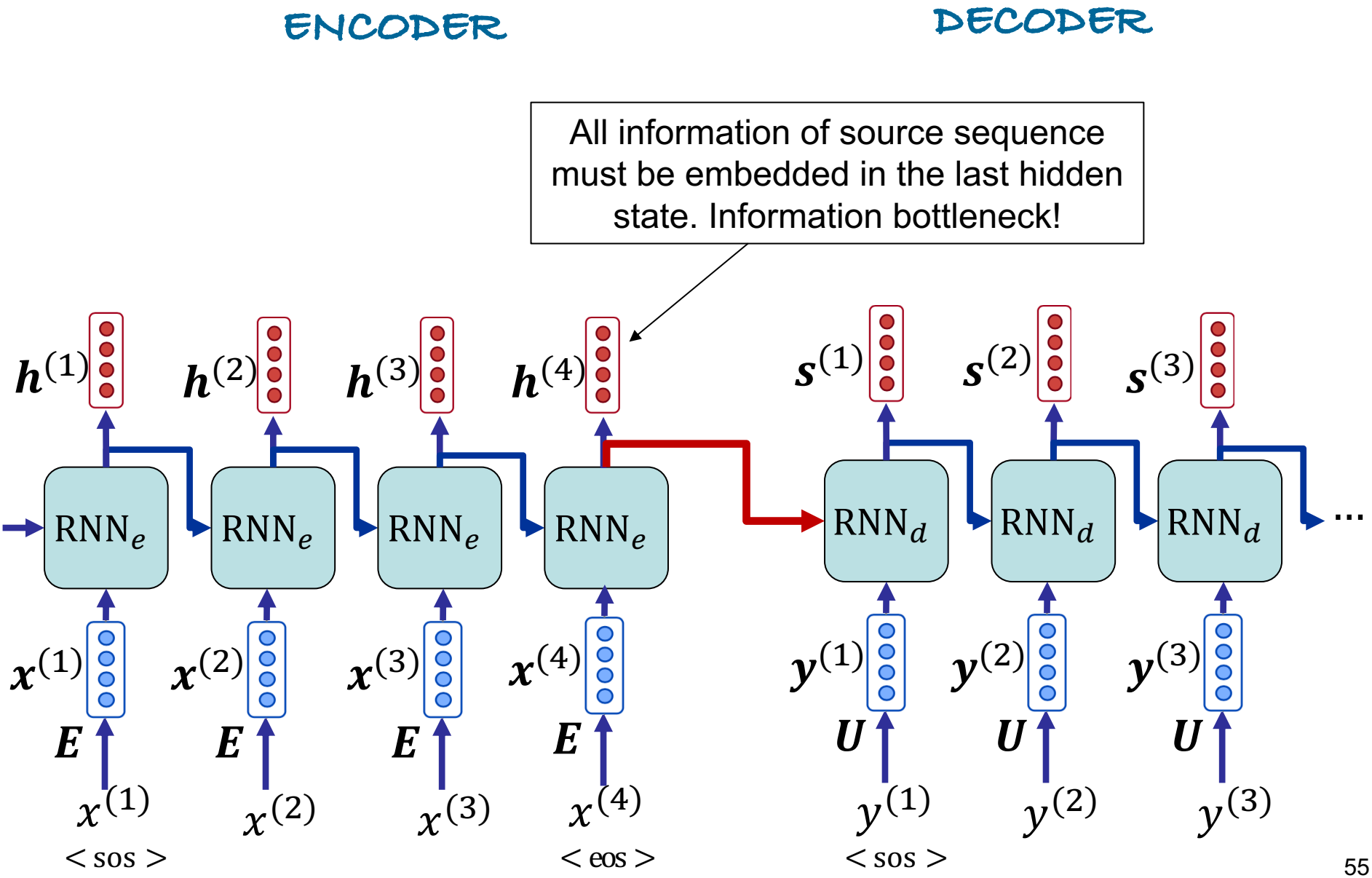
- It is done by defining an attention distribution over inputs, and calculating their weighted sum

- A more generic definition of attention network has two inputs: key vectors $\boldsymbol{K}$, and value vectors $\boldsymbol{V}$
  - Key vectors are used to calculate attentions
  - and, as before, output is the weighted sum of value vectors
  - In practice, in most cases $\boldsymbol{K} = \boldsymbol{V}$. So we consider our (slightly simplified) definition in most parts of this course

Agenda

- Sequence-to-sequence models
- Attention Mechanism
- **seq2seq with Attention**

Bottleneck problem in basic seq2seq



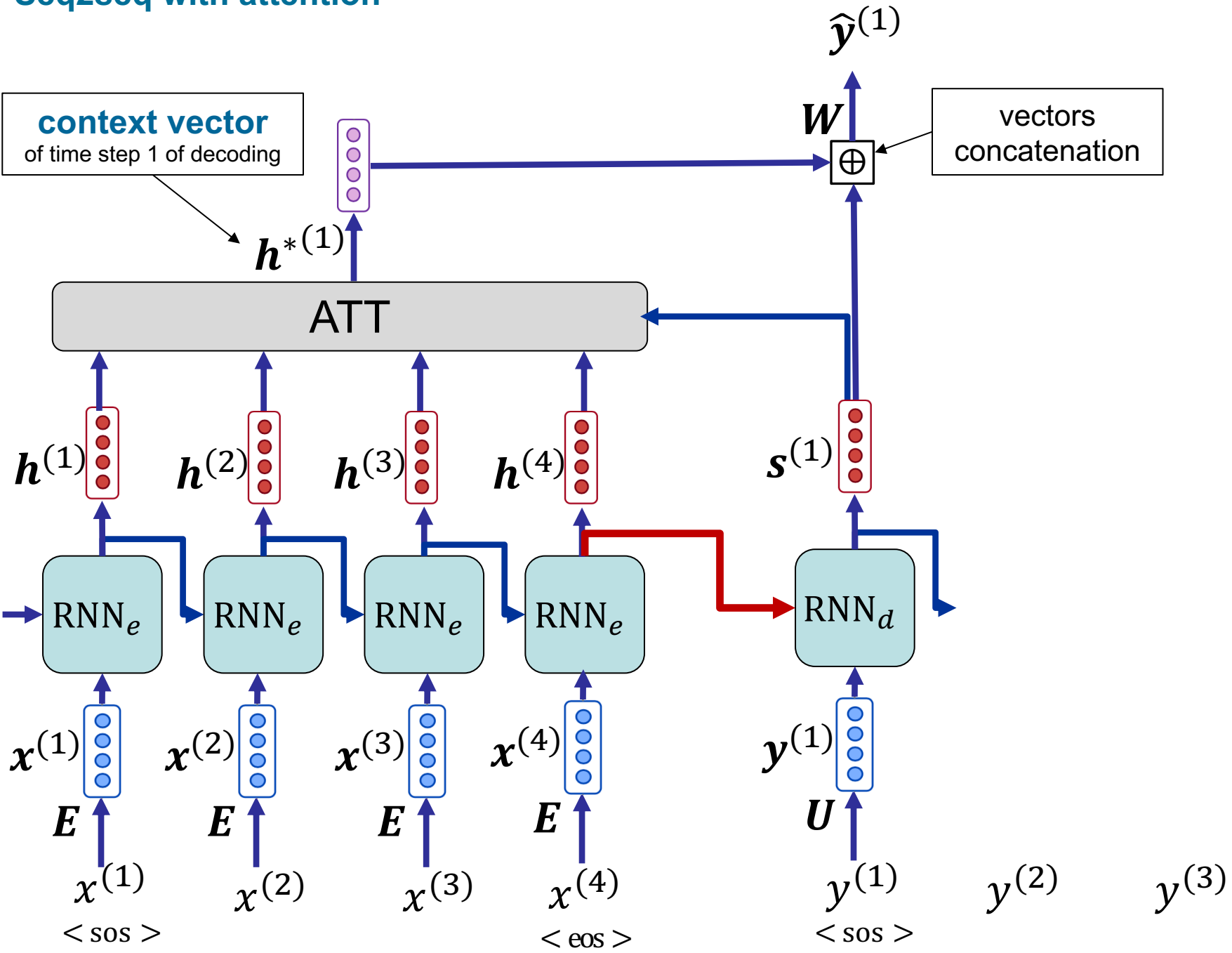
Seq2seq + Attention

- It can be useful, if we allow decoder directly access all elements of source sequence, and to decide where on source sequence to focus

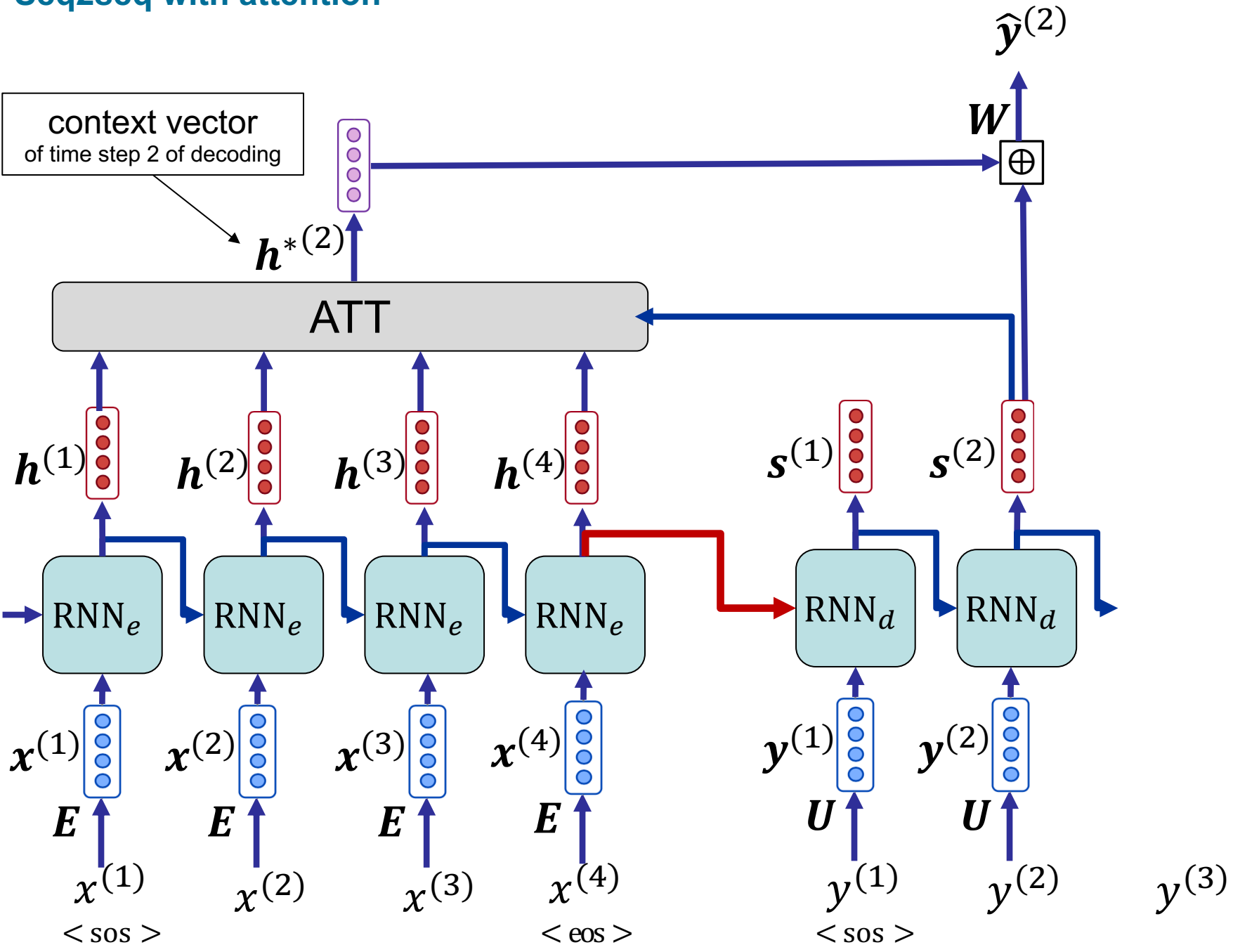
- Attention is a solution to the bottleneck problem

- At each decoding time step, decoder attends on vectors of source sequence,
  - and therefore bypasses the bottleneck!

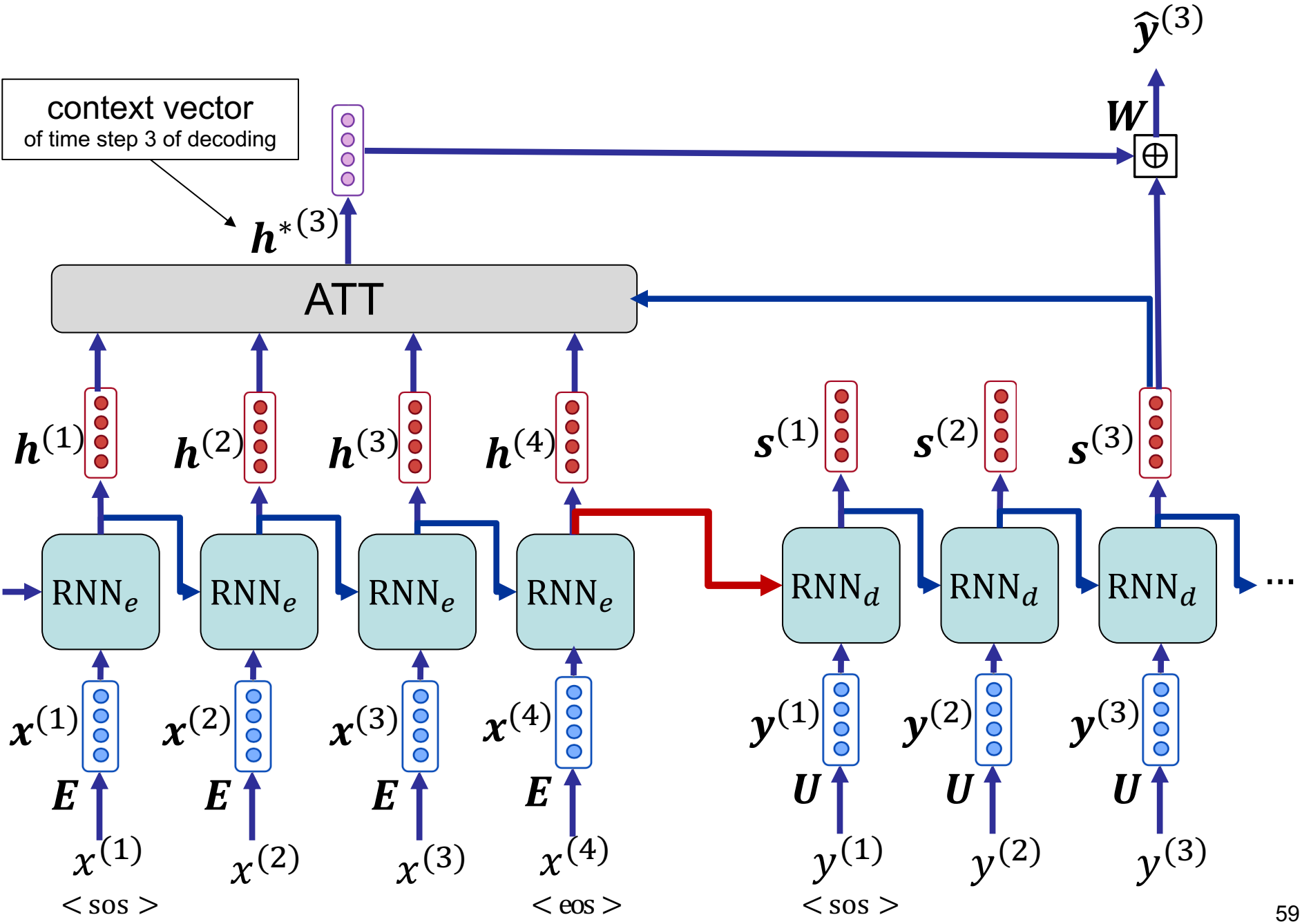
Seq2seq with attention



Seq2seq with attention



Seq2seq with attention



Seq2seq with attention – formulation

ENCODER is the same as basic seq2seq

DECODER – input

- Decoder embedding
  - Decoder embeddings *at* input for target words $(\mathbb{V}_d) \rightarrow \boldsymbol{U}$
  - Embedding of the target word $y^{(t)}$ (at time step t) $\rightarrow \boldsymbol{y}^{(t)}$

- Decoder RNN

$$\boldsymbol{s}^{(t)} = \mathrm{RNN}(\boldsymbol{s}^{(t-1)}, \boldsymbol{y}^{(t)})$$

  - The values of the last hidden state of the encoder RNN are passed to the initial hidden state of the decoder RNN:

$$\boldsymbol{s}^{(0)} = \boldsymbol{h}^{(L)}$$

Parameters are shown in red

Seq2seq with attention – formulation

DECODER – attention

- Attention context vector

$$\boldsymbol{h}^{*(t)} = \mathrm{ATT}\left(\boldsymbol{s}^{(t)}, \left\{\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(L)}\right\}\right)$$

For instance, if ATT is a "basic dot-product attention", this is done by:

- First calculating non-normalized attentions:

$$\tilde{\alpha}_j^{(t)} = \boldsymbol{s}^{(t)^{\mathrm{T}}} \boldsymbol{h}_j$$

- Then, normalizing the attentions:

$$\alpha_j^{(t)} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}^{(t)})_j$$

- and finally calculating the weighted sum of encoder hidden states

$$\boldsymbol{h}^{*(t)} = \sum_{j=1}^{\boldsymbol{L}} \alpha_j^{(t)} \boldsymbol{h}_j$$

Parameters are shown in red

Seq2seq with attention – formulation

DECODER – output

- Decoder output prediction
  - Predicted probability distribution of words at the next time step:

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{W}[\boldsymbol{s}^{(t)}; \boldsymbol{h}^{*(t)}] + \boldsymbol{b}\right) \in \mathbb{R}^{|\mathbb{V}|}$$

[;] denotes the concatenation of two vectors

- Training and inference of a seq2seq with attention are the same as a basic seq2seq model

Parameters are shown in red

Seq2seq with attention – summary

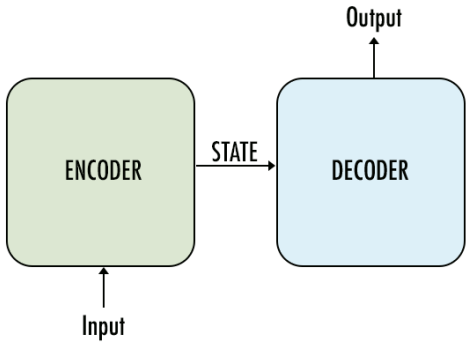
- Attention on source sequence facilitates the selection of relevant parts, and flow of information

- Attention in seq2seq helps avoiding vanishing gradient problem
  - Provides shortcut to faraway states

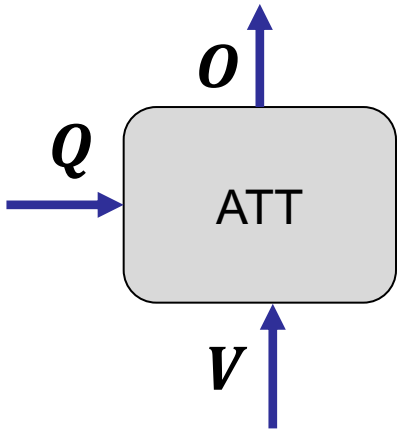
- Attention provides some interpretability
  - Looking at attention distribution, we can see what the decoder is focusing on (try it here: https://distill.pub/2016/augmented-rnns/#attentional-interfaces)
  - However, it is still disputable, whether attention (especially in Transformers) really provides explanation!

Recap

- Sequence-to-sequence models generate language given an input text



- Attention is a general deep learning approach to learn to focus on certain parts, and compose outputs



- Attention significantly helps seq2seq models!