CS224n: NLP with Deep Learning

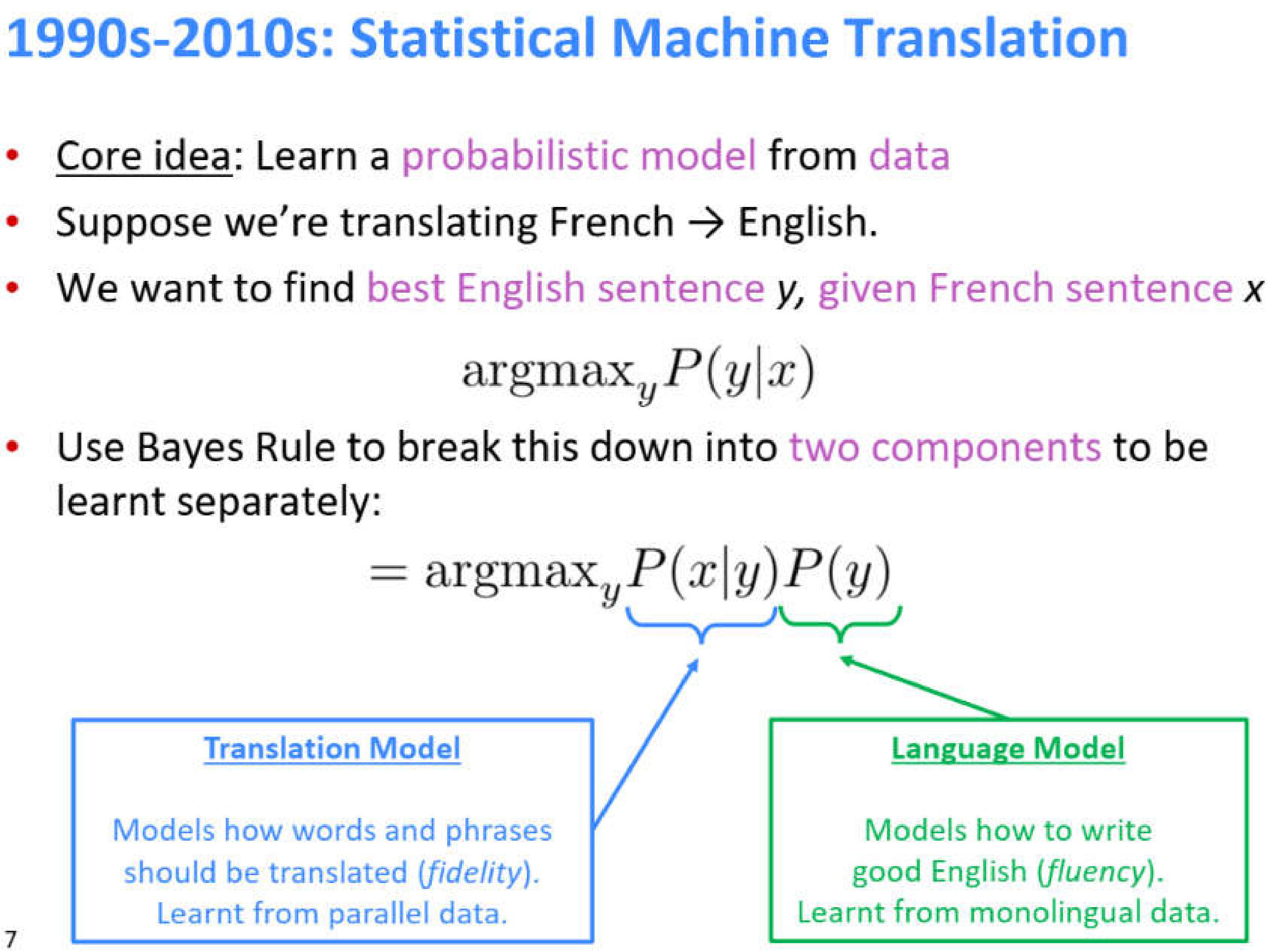
Lecture 8: Machine Translation

Statistical Machine Translation

Idea:
Learn a probabilistic model, using our data

Using Bayes Rule, it can be decomposed into:

- Translation Model:
  Know about translation of local chunks of phrases
- Language Model:
  Writing good English (target language)

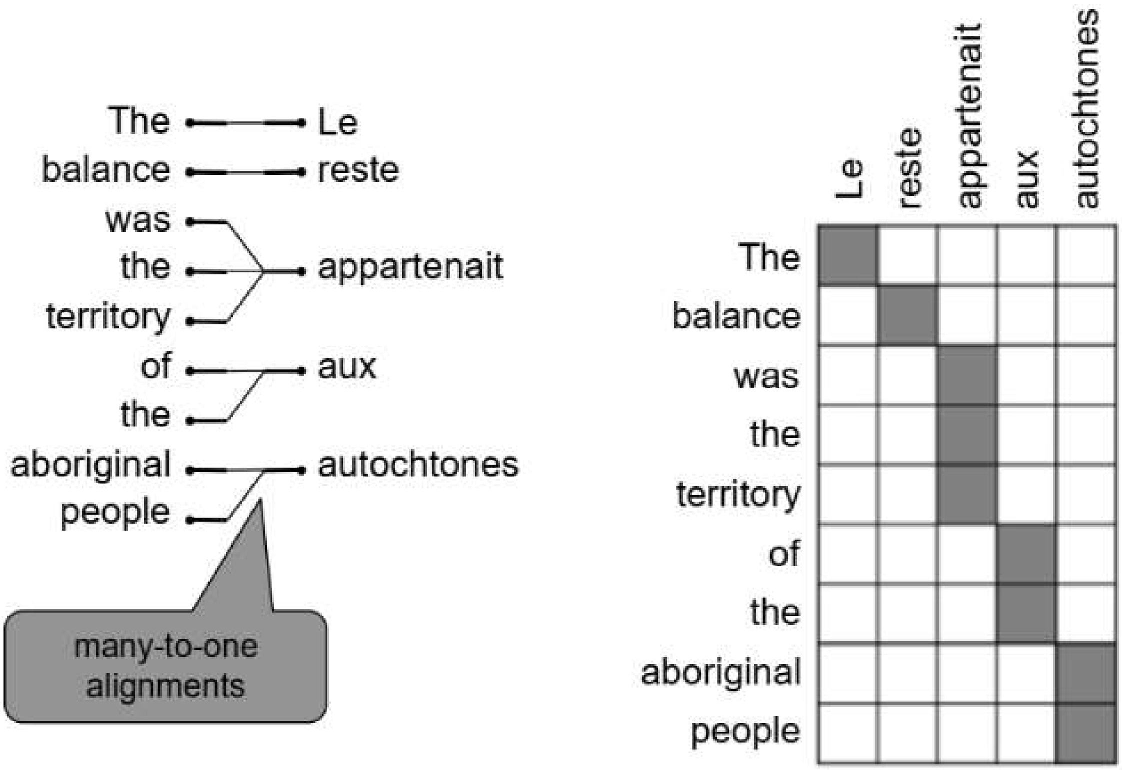


Alignment

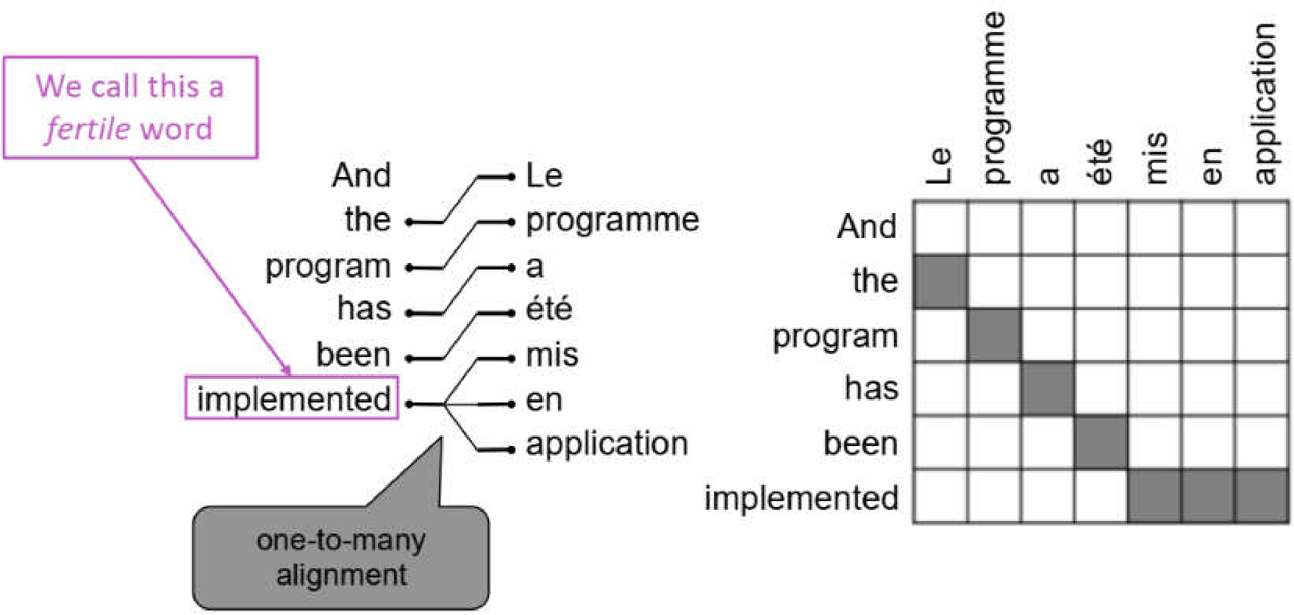
Alignment = correspondence between particular words

- Not necessarily bijective:
  Can be:
  - Many-to-one
  - One-to-many: the **one** word is called a *fertile* word
  - Many-to-many

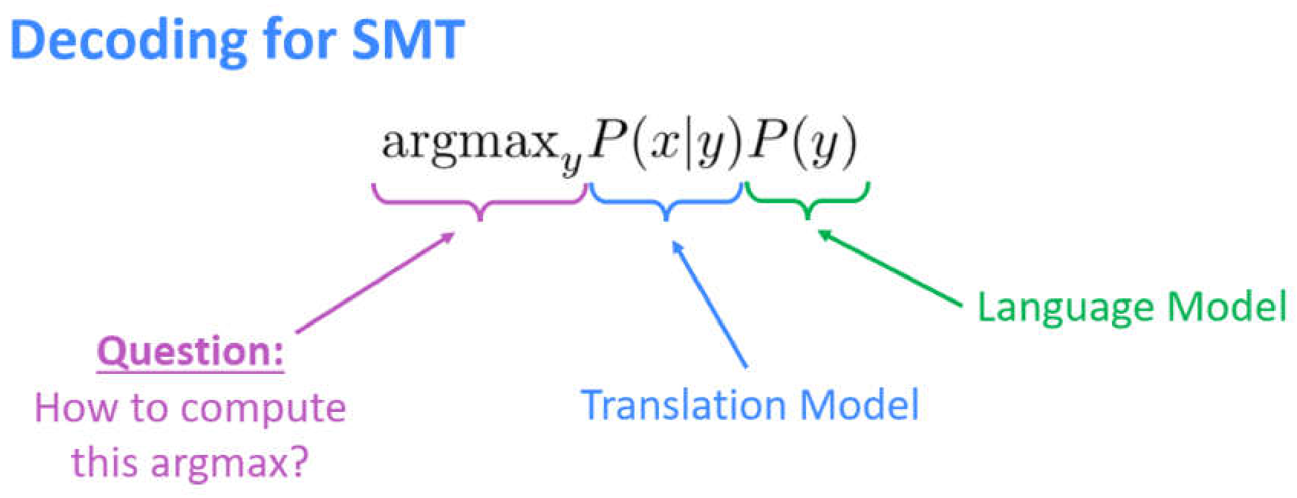
Alignment can be many-to-one



Alignment can be one-to-many



Back to SMT



- We can't enumerate every y to calculate this probability

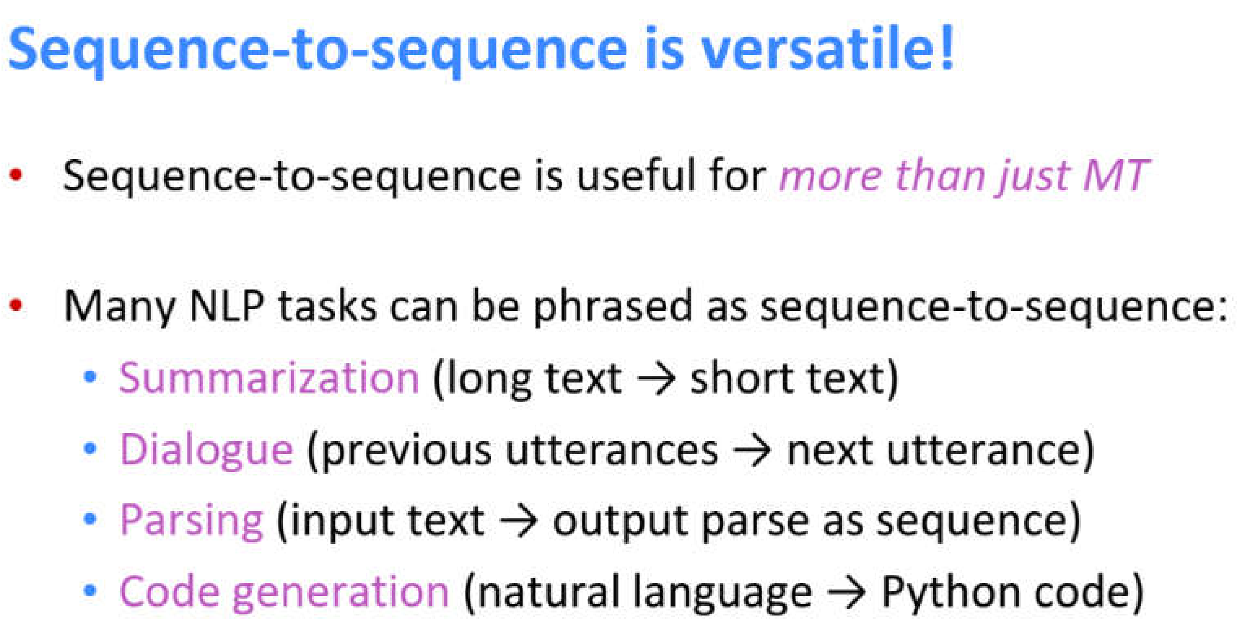
- We explore the different possibilities, and discard the low-probability ones as we go $\rightarrow$ Beam Search

---

Neural Machine Translation

- Neural Machine Translation is called **sequence-to-sequence** (seq2seq)
- Involves 2 RNN

Other usages of Seq2seq

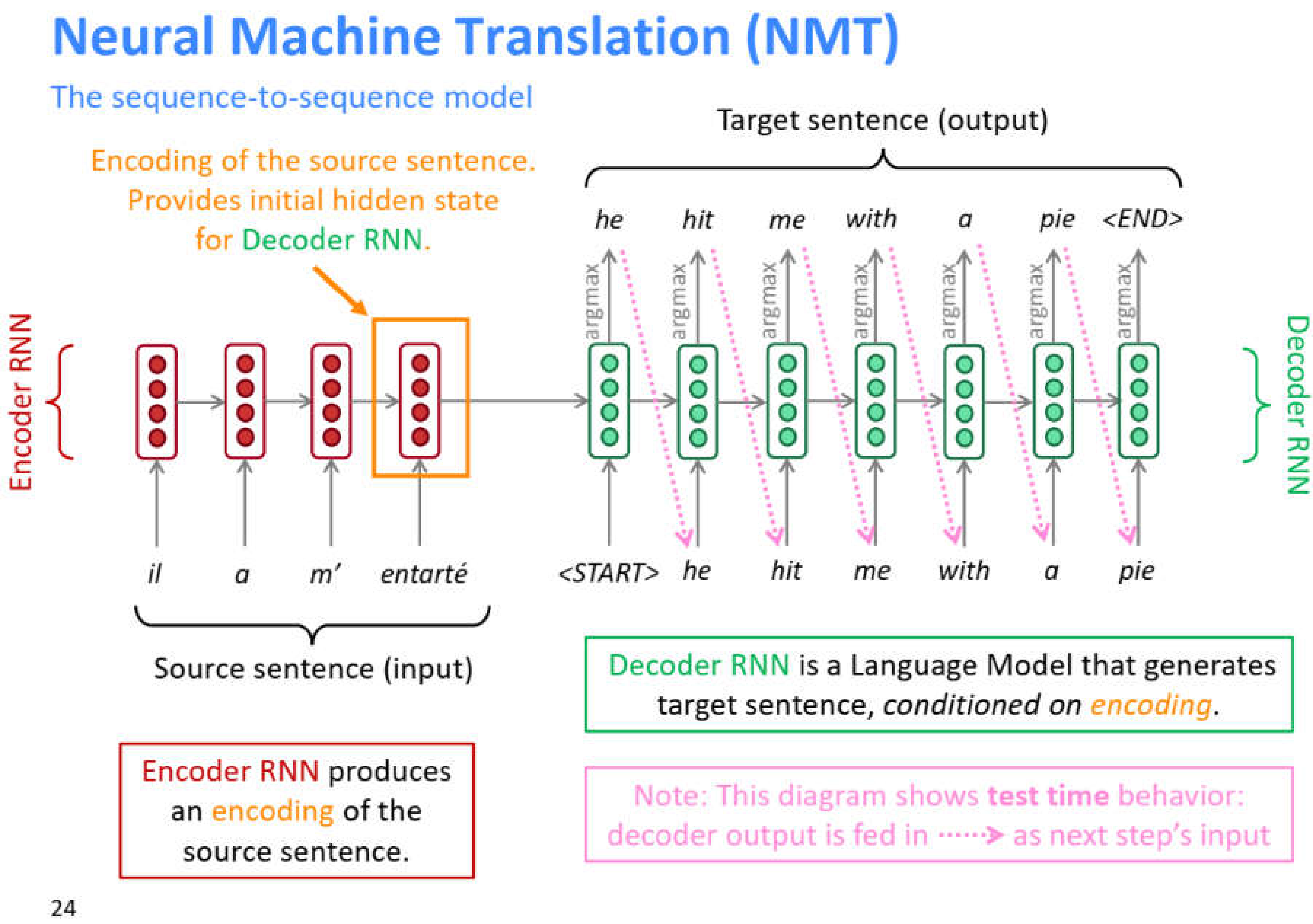


Encoding / Decoding

- Encoding of the source sentence = last hidden state of the Decoder RNN

- Decoder RNN is a **conditional** Language Model, as it is conditioned on this encoding

At Test time



NMT Principle

Direct calculation of $P(y|x)$:

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots P(y_T|y_1,\ldots,y_{T-1},x)$$

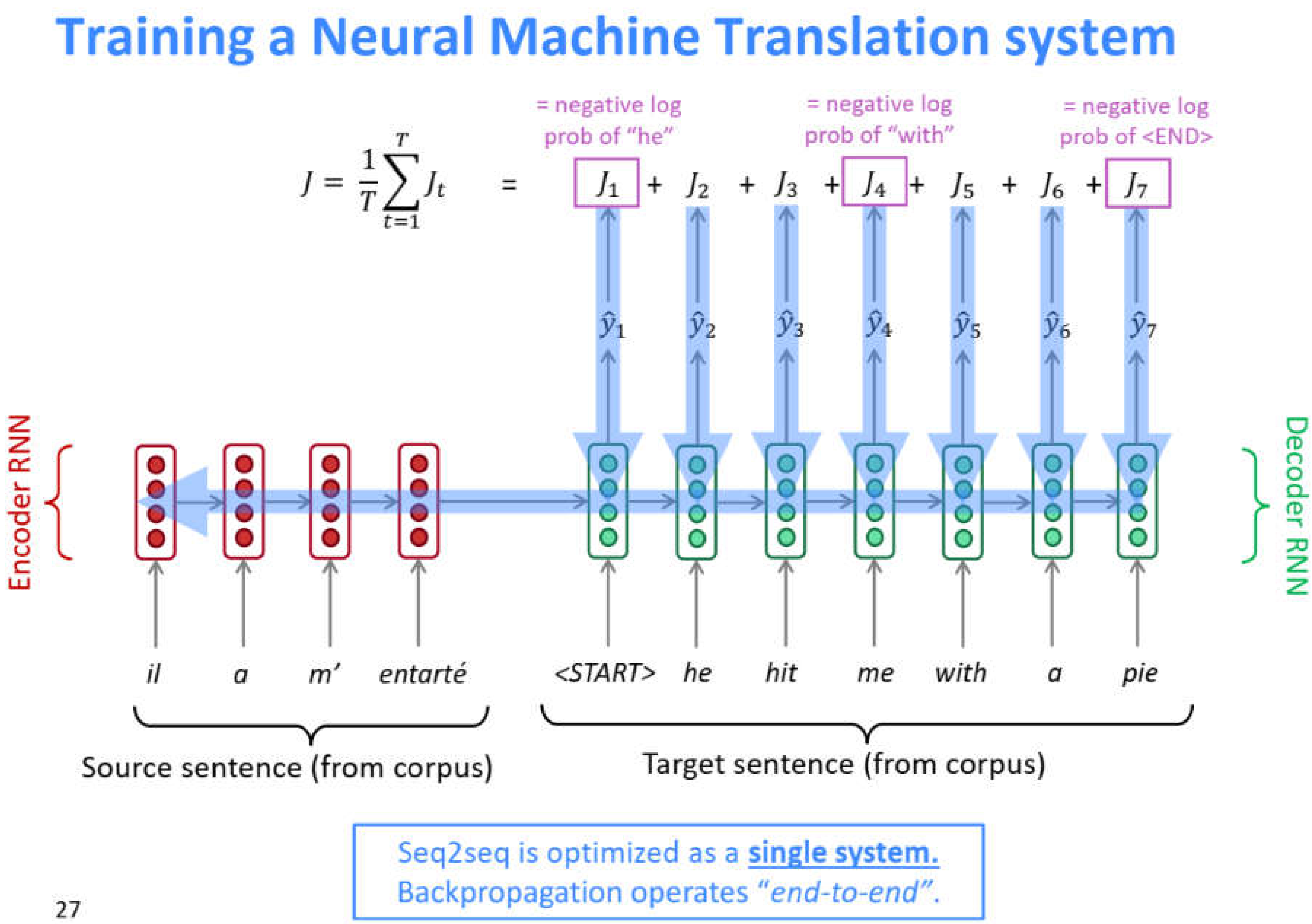
Training a Model

We need:

- A parallel corpus

- Words embeddings for words in both languages (source & target)

- Here, in NMT, we learn the translation objective directly
- While in SMT, we were doing it indirectly, by separating the task into different subtasks



The backprop is happening end-to-end:

- one end is the loss functions
- the other end is the beginning state of the encoder RNN

Backprop flows through the entire system

⚠ Difference between Training & Testing ⚠

During Training, in the Decoder RNN, we don't feed the previous prediction into the next step
Instead, we feed each state the correct previous word from the corpus

It is possible to train the 2 RNN separately:
for example, training a strong Language Model on its own, then initializing the Decoder RNN with it

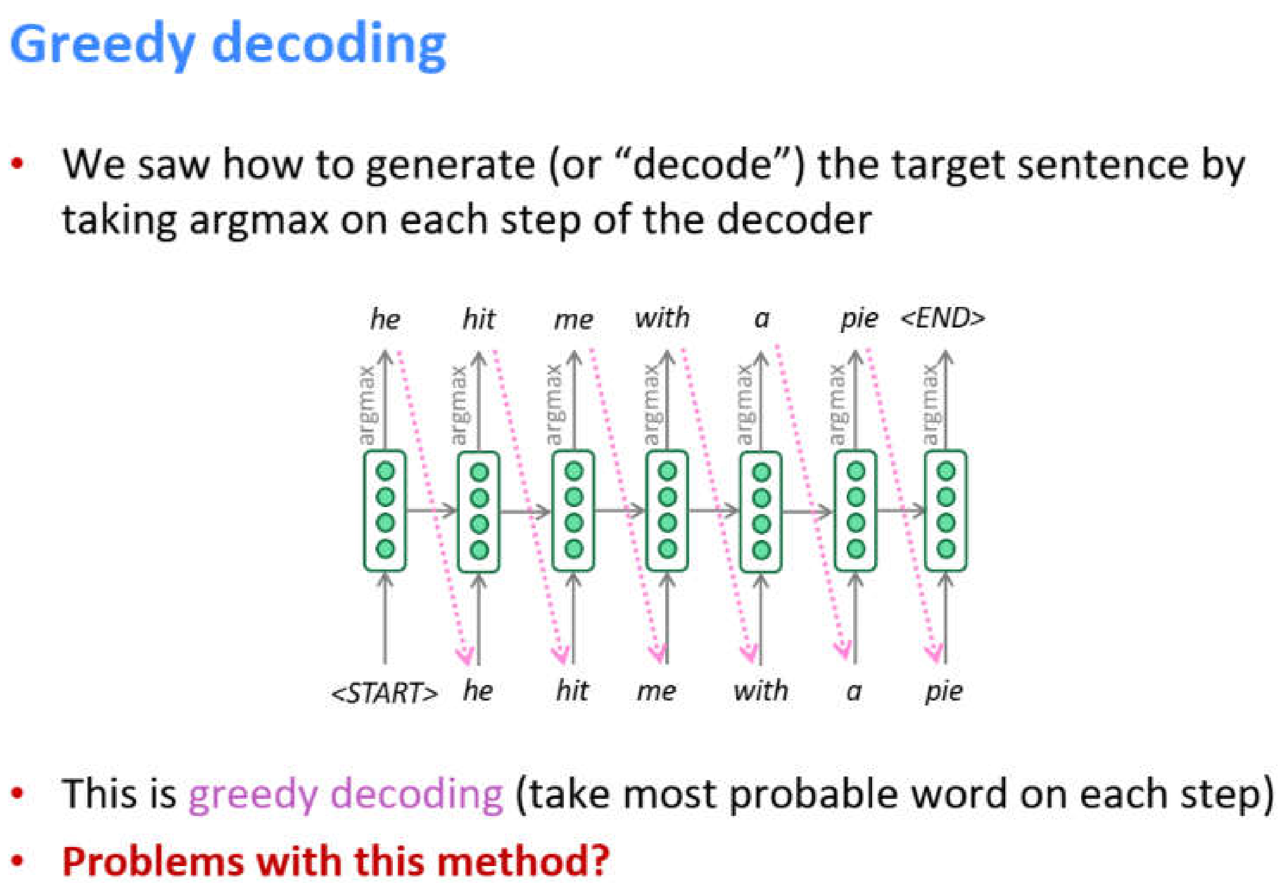
Remarks

- In practice, we pad the short sentences up to a certain length

- We can use pre-trained word embeddings, or use word2vec or GloVe to train them
  And then, do the training of the Encoder/Decoder

Towards Beam search Decoding

Problem of Greedy Decoding



⚠ **Problem** ⚠
The argmax at each step doesn't necessarily give us the global argmax for the whole sentence !

(ie, usually, the global optimum is not the combination of all the local optima)

- We can't search all the possibilities for the argmax, as it would be much too expensive to compute

→ Use a search algorithm: **Beam search**

Beam Search Decoding

- Beam size k:
  how big our search space is at any time

Beam search decoding

- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)

- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1, \ldots, y_{i-1}, x)$$

  - Scores are all negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step

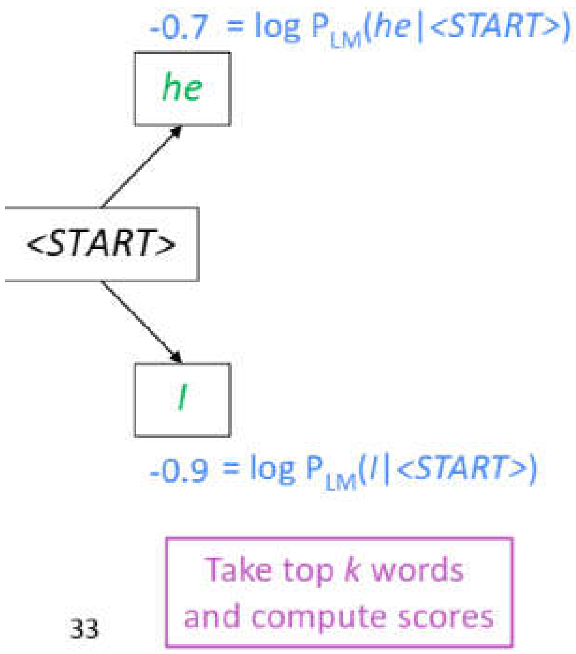
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

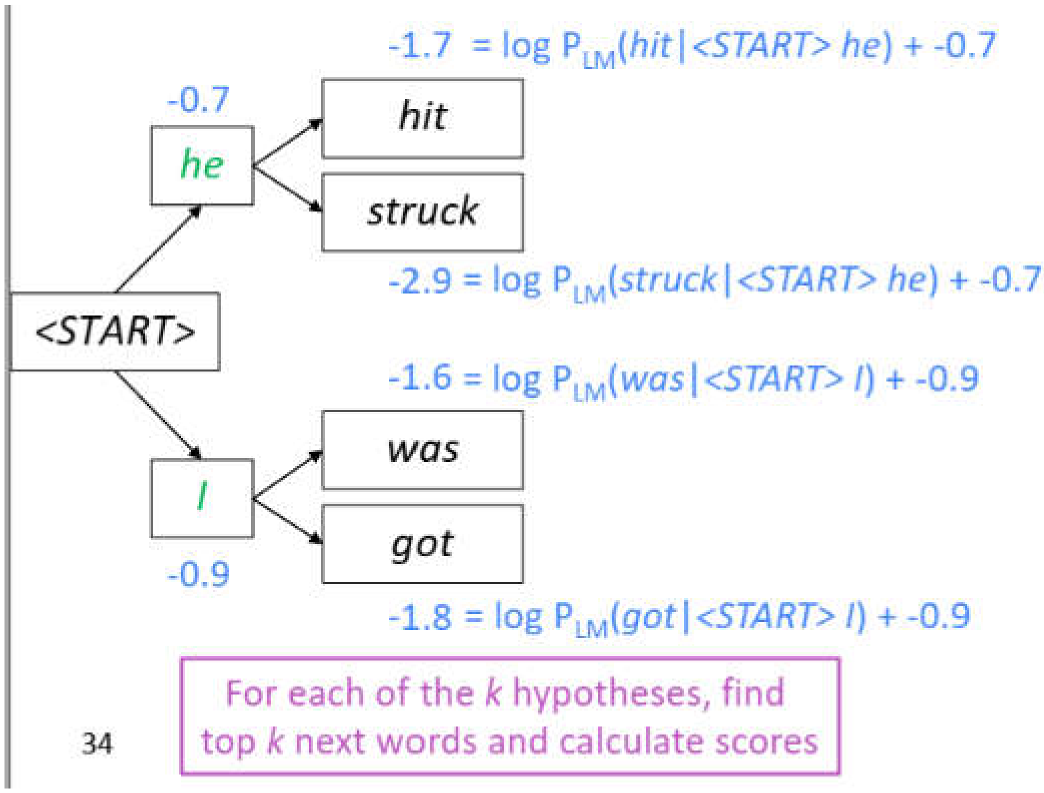
31

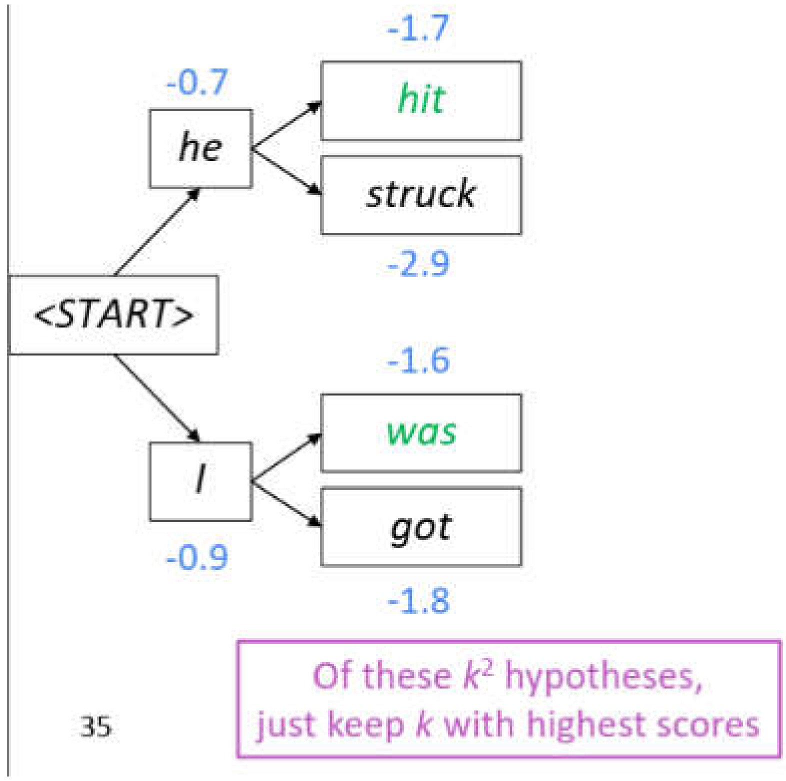
Example

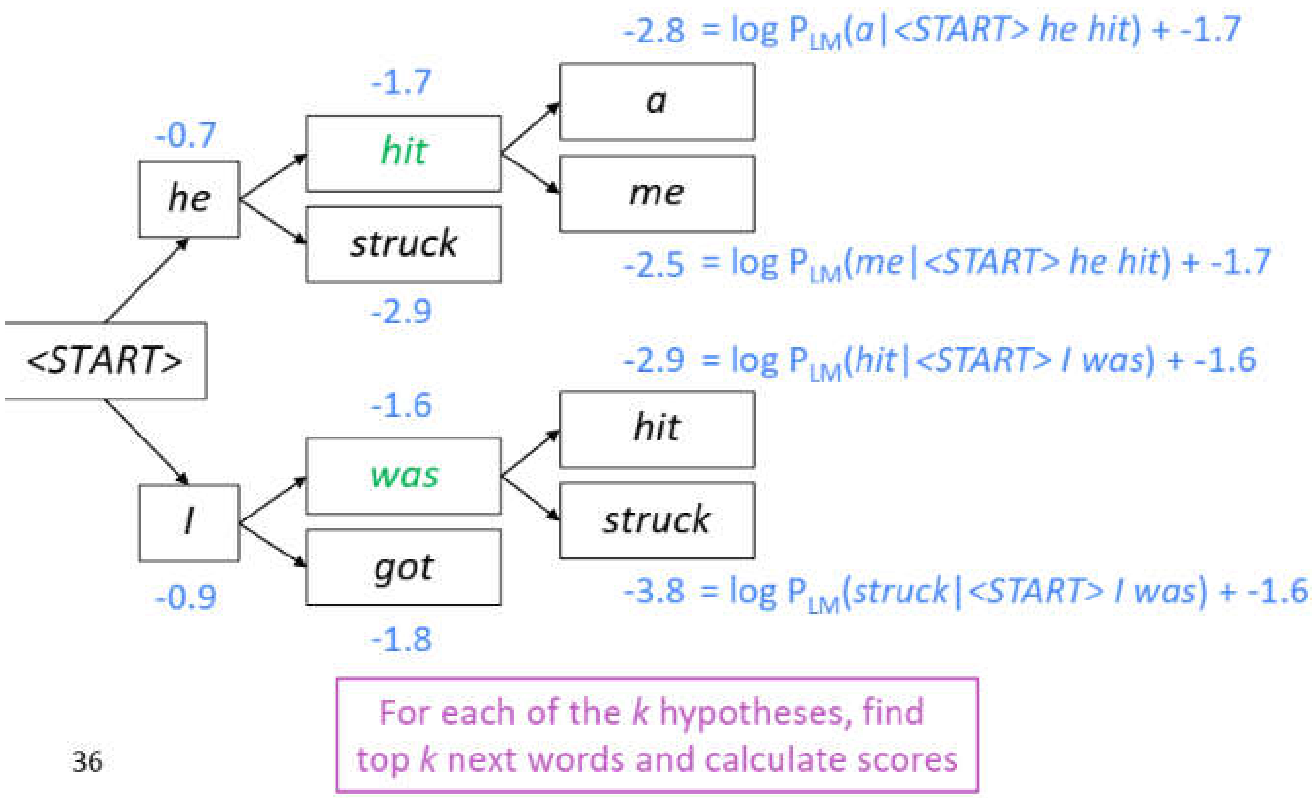
Beam search decoding: example

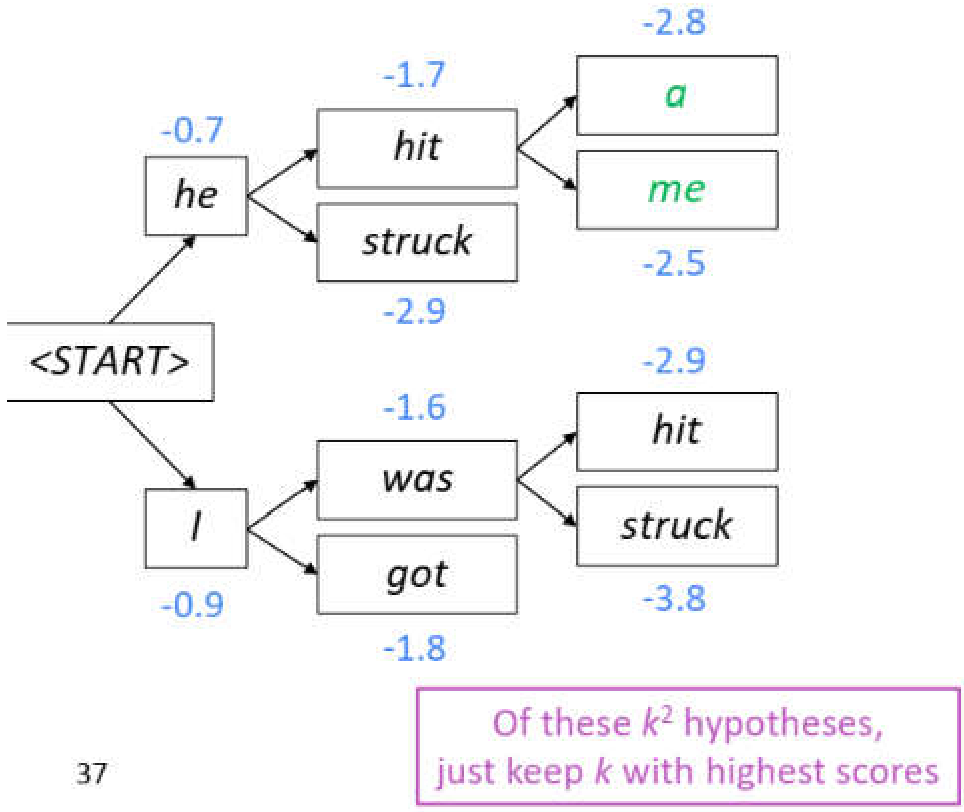
Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$

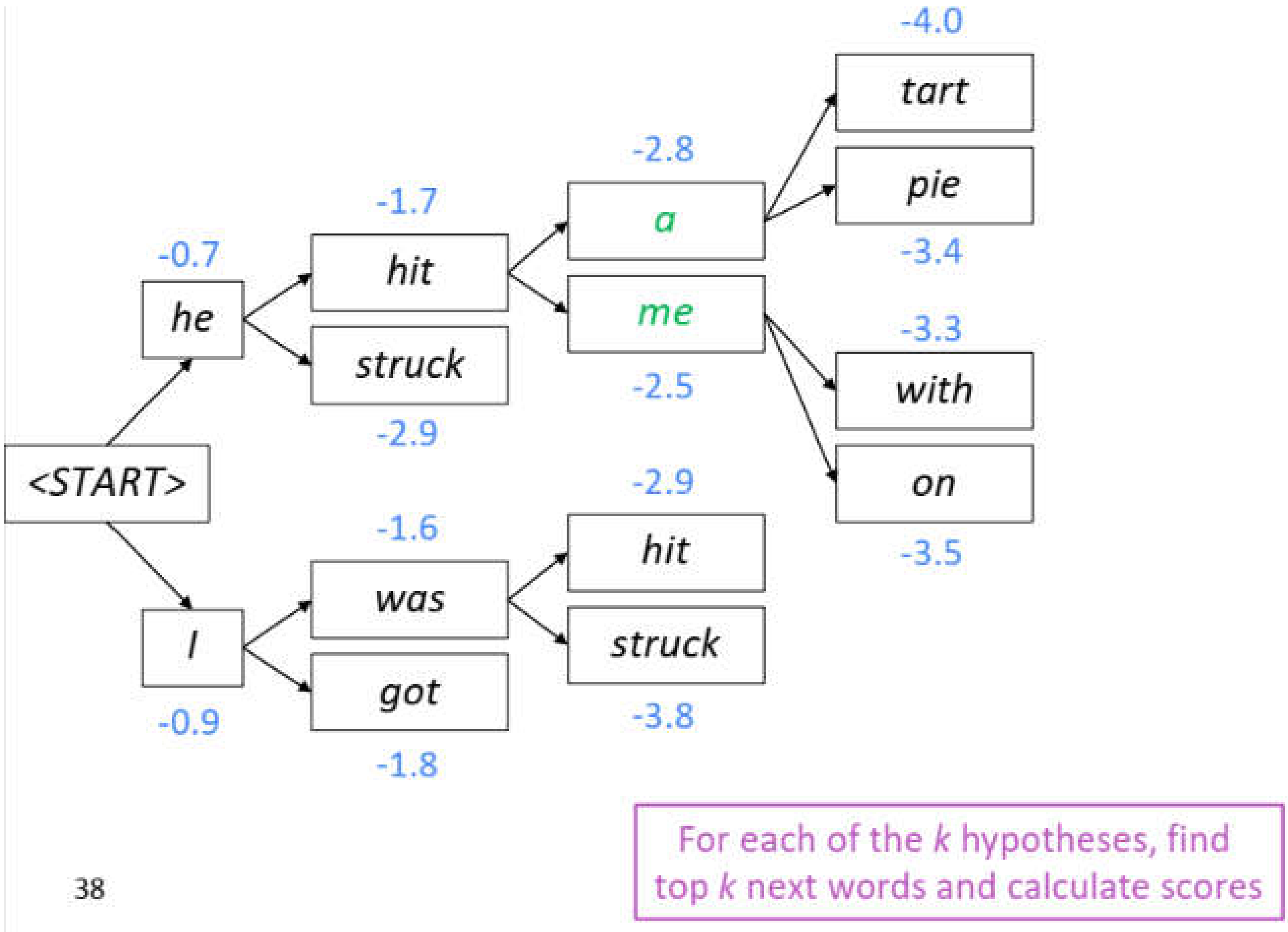


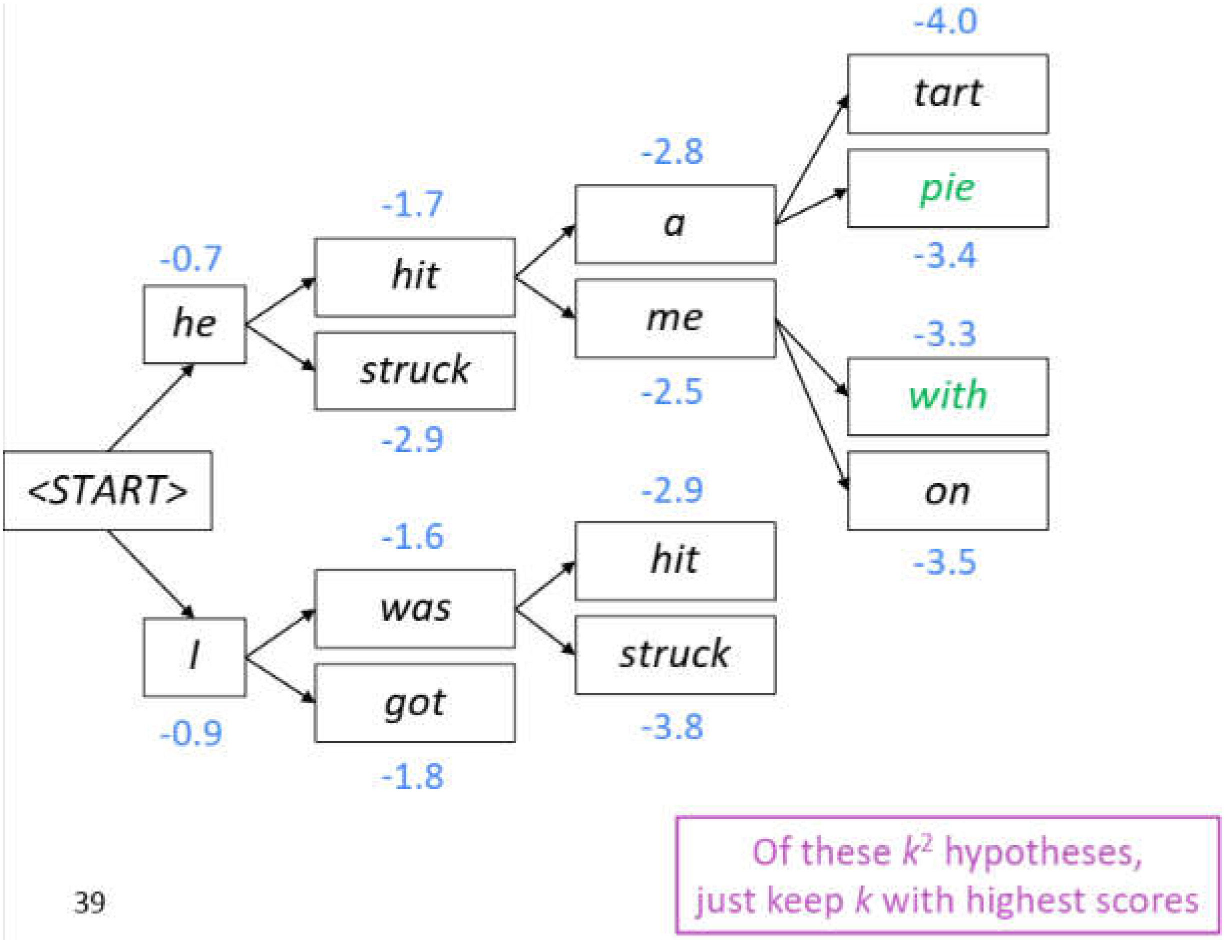


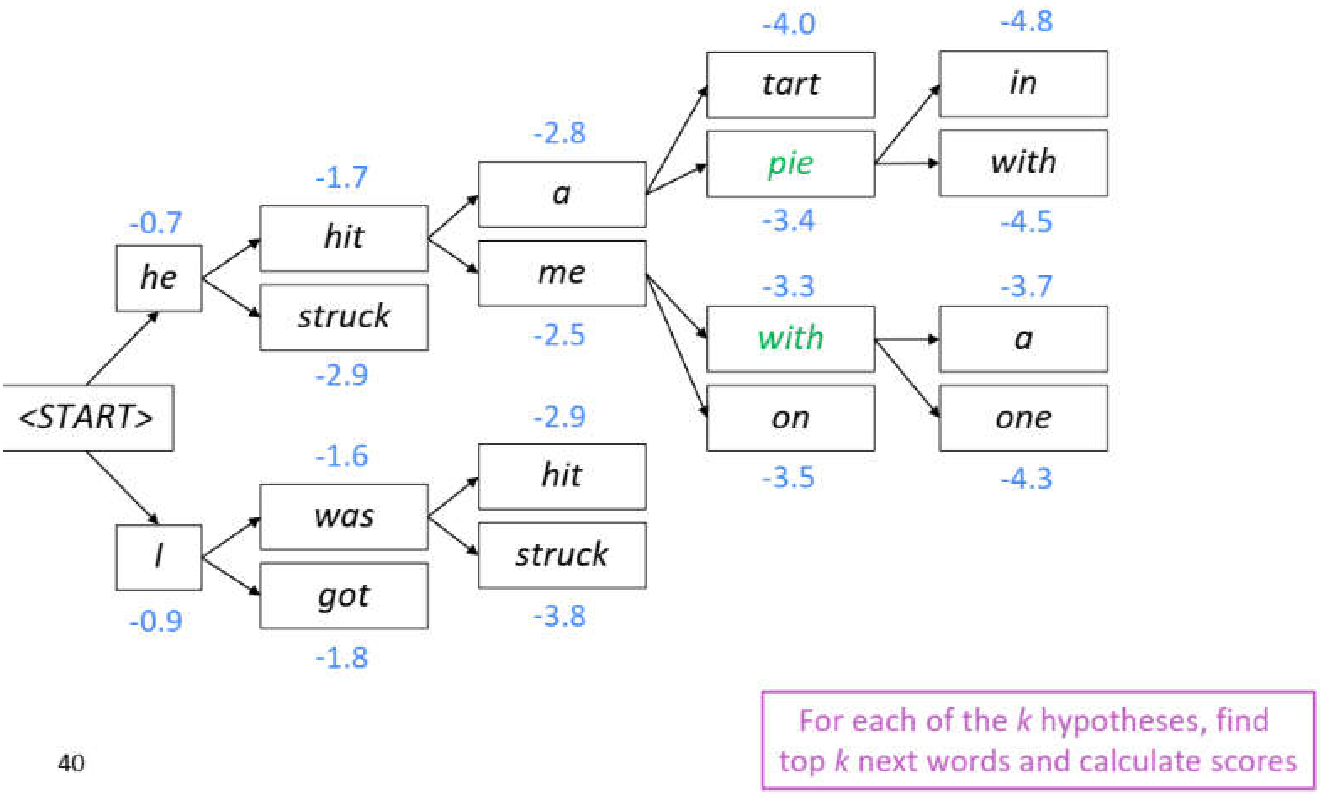

-1.7
-0.7
hit
he
struck
-2.9
<START>
-1.6
was
I
got
-0.9
-1.8
Of these k² hypotheses,
just keep k with highest scores
35

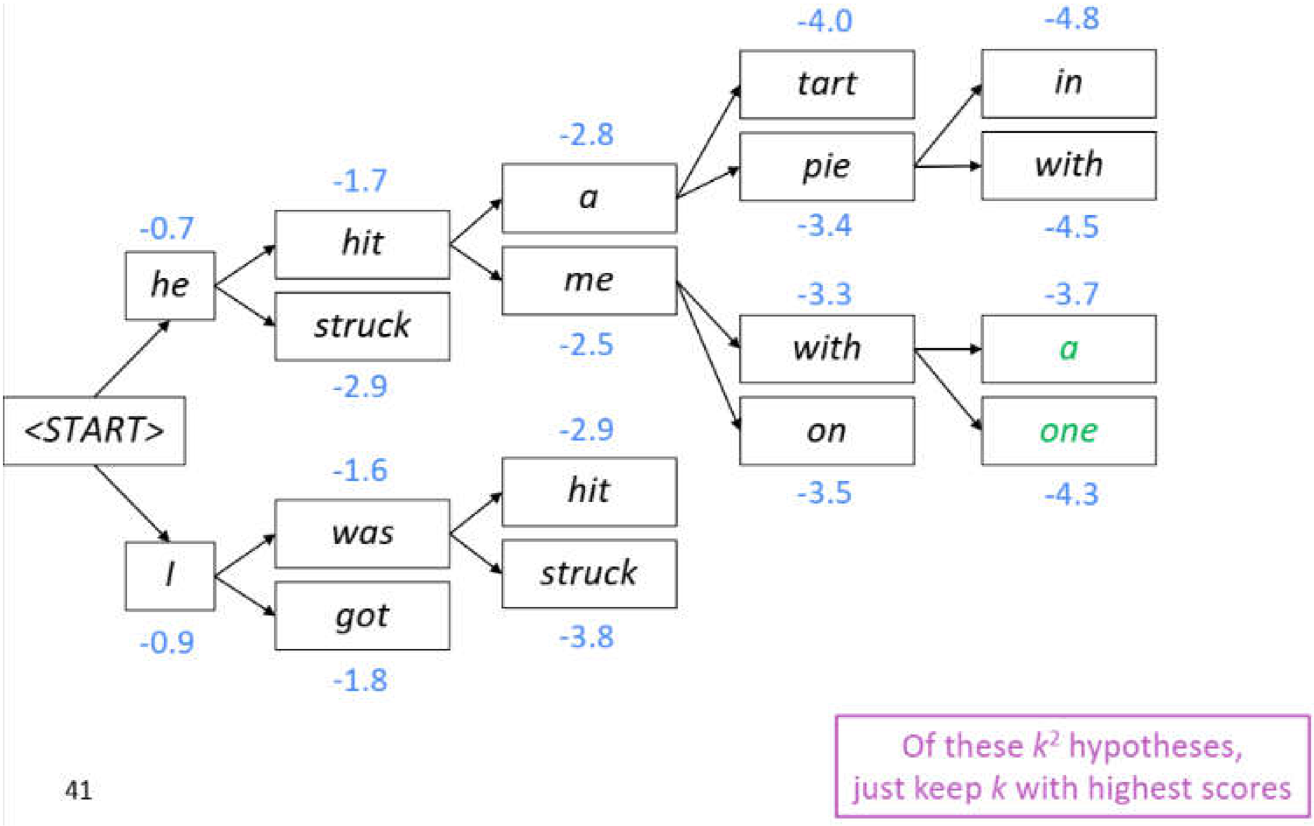

-2.8 = log P_LM(a|<START> he hit) + -1.7
-1.7
-0.7
a
hit
he
me
struck
-2.5 = log P_LM(me|<START> he hit) + -1.7
-2.9
<START>
-2.9 = log P_LM(hit|<START> I was) + -1.6
-1.6
hit
was
I
struck
got
-0.9
-3.8 = log P_LM(struck|<START> I was) + -1.6
-1.8
For each of the k hypotheses, find
top k next words and calculate scores
36

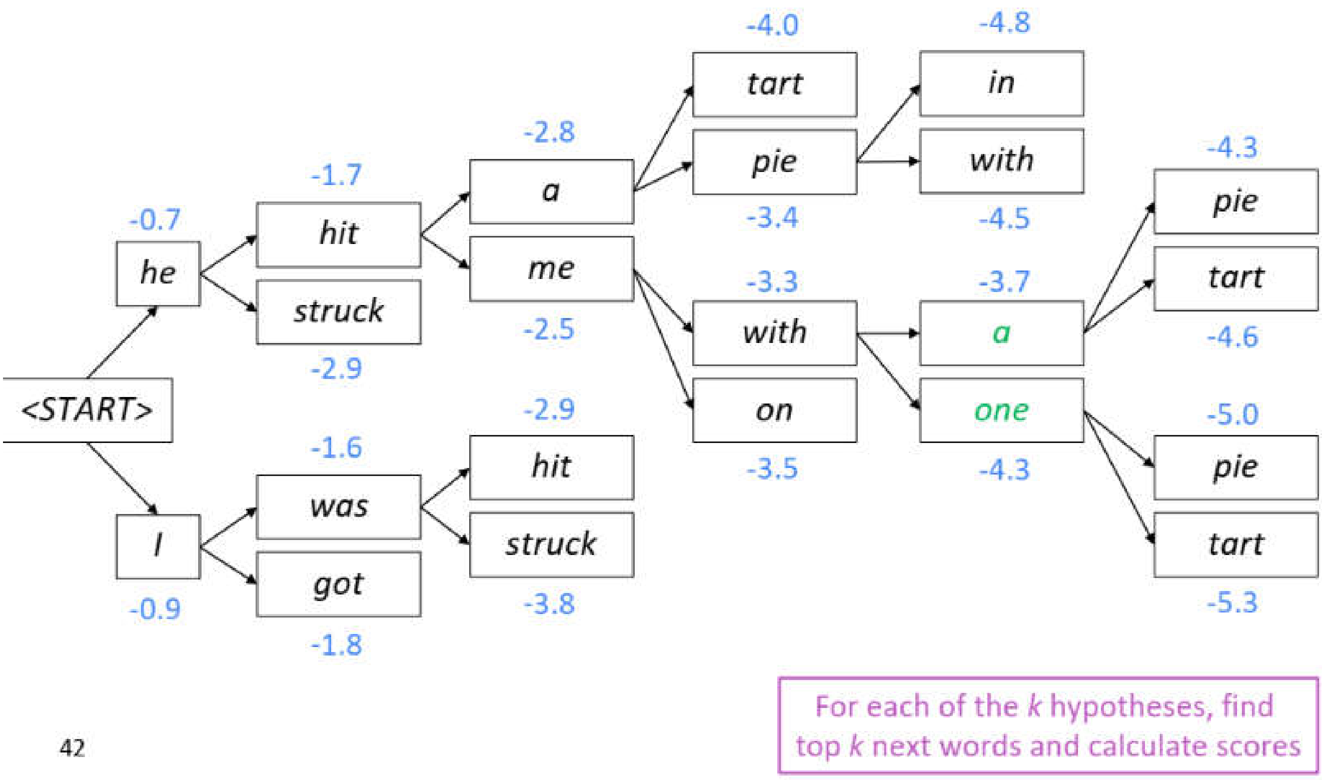

-2.8
a
-1.7
hit
-0.7
he
me
struck
-2.5
-2.9
<START>
-2.9
-1.6
hit
was
I
struck
got
-0.9
-3.8
-1.8
37
Of these k^2 hypotheses, just keep k with highest scores

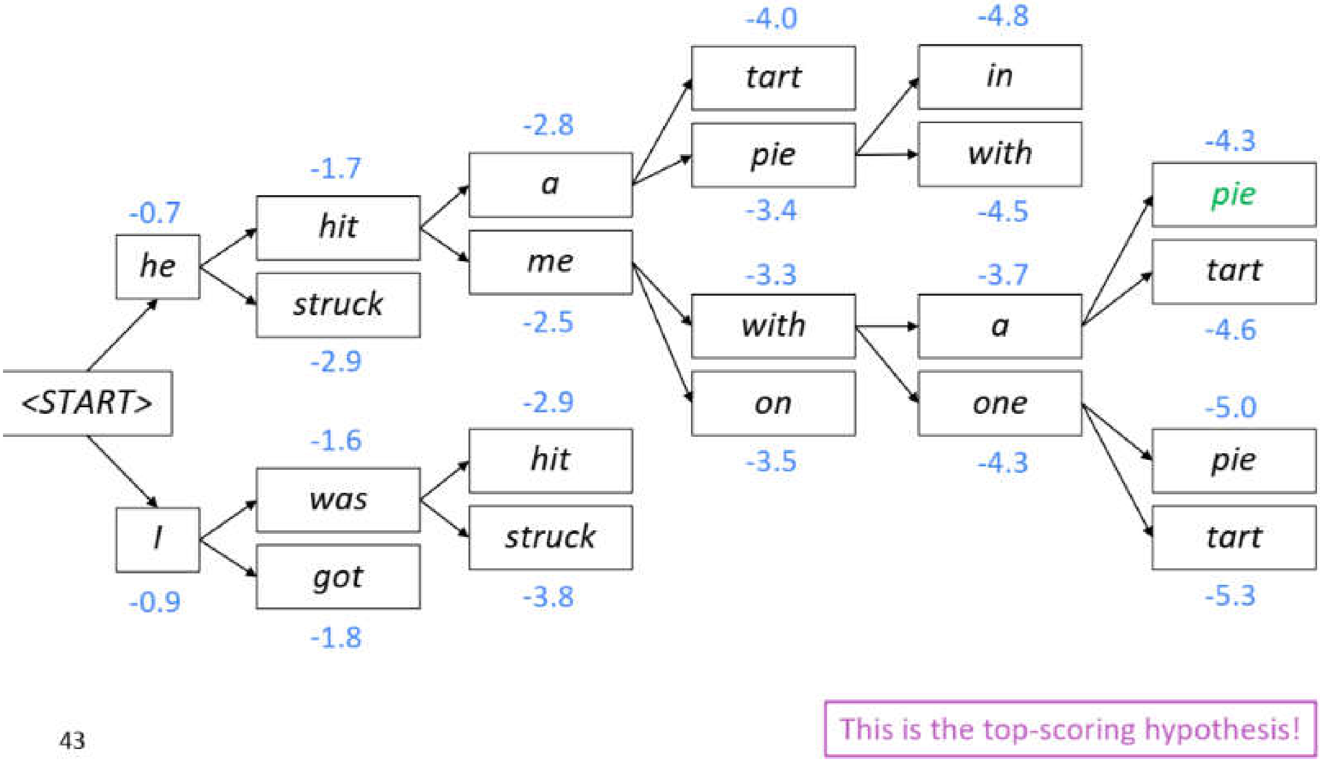

-4.0
tart
-2.8
pie
-1.7
a
-3.4
-0.7
hit
he
me
-3.3
struck
-2.5
with
-2.9
<START>
on
-2.9
-1.6
-3.5
hit
was
I
struck
got
-0.9
-3.8
-1.8
38
For each of the k hypotheses, find top k next words and calculate scores

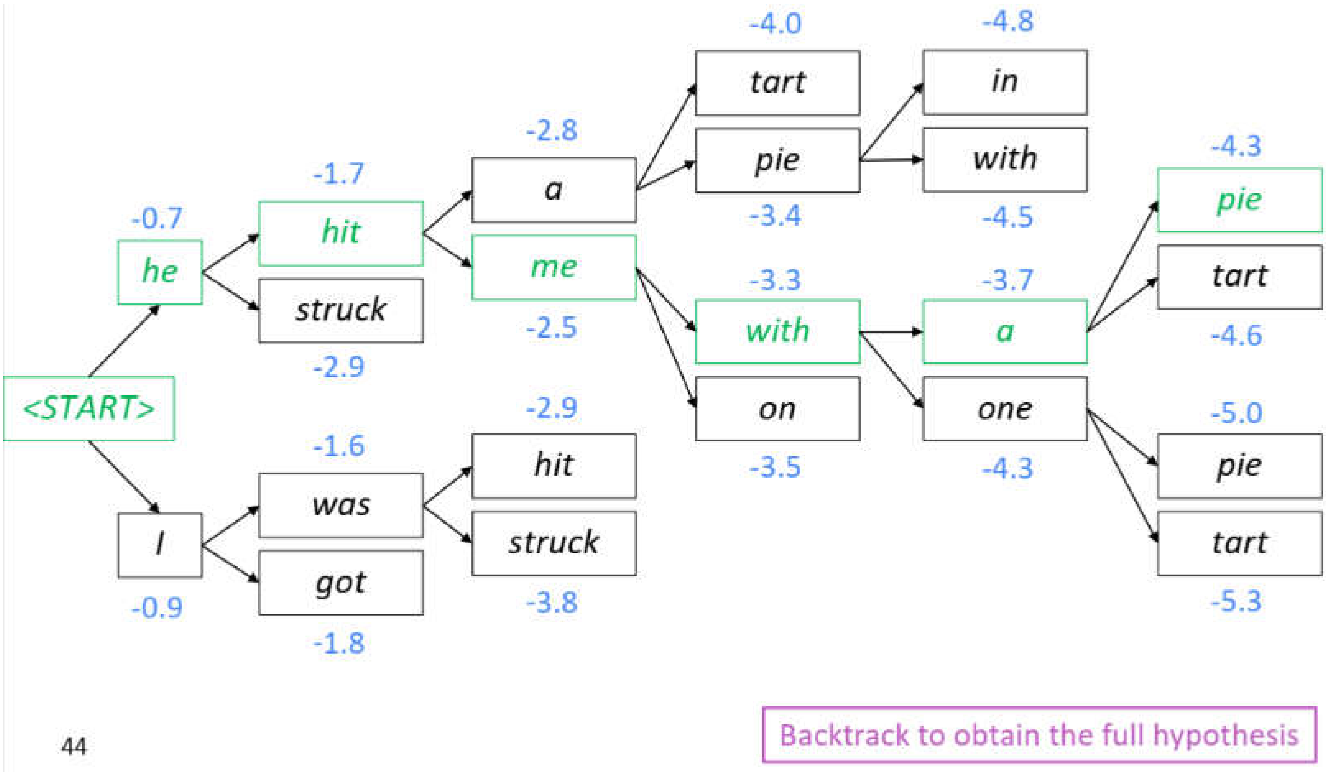

<START>
he
-0.7
I
-0.9
hit
-1.7
struck
-2.9
was
-1.6
got
-1.8
a
-2.8
me
-2.5
hit
-2.9
struck
-3.8
tart
-4.0
pie
-3.4
with
-3.3
on
-3.5
39
Of these k^2 hypotheses,
just keep k with highest scores


<START>
he
-0.7
I
-0.9
hit
-1.7
struck
-2.9
was
-1.6
got
-1.8
a
-2.8
me
-2.5
hit
-2.9
struck
-3.8
tart
-4.0
pie
-3.4
with
-3.3
on
-3.5
in
-4.8
with
-4.5
a
-3.7
one
-4.3
40
For each of the k hypotheses, find
top k next words and calculate scores


<START>
he -0.7
I -0.9
hit -1.7
struck -2.9
was -1.6
got -1.8
a -2.8
me -2.5
hit -2.9
struck -3.8
tart -4.0
pie -3.4
with -3.3
on -3.5
in -4.8
with -4.5
a -3.7
one -4.3
41
Of these k^2 hypotheses, just keep k with highest scores


<START>
he -0.7
I -0.9
hit -1.7
struck -2.9
was -1.6
got -1.8
a -2.8
me -2.5
hit -2.9
struck -3.8
tart -4.0
pie -3.4
with -3.3
on -3.5
in -4.8
with -4.5
a -3.7
one -4.3
pie -4.3
tart -4.6
pie -5.0
tart -5.3
42
For each of the k hypotheses, find top k next words and calculate scores





Stopping Criterion

Contiune exploring our hypothesis until:

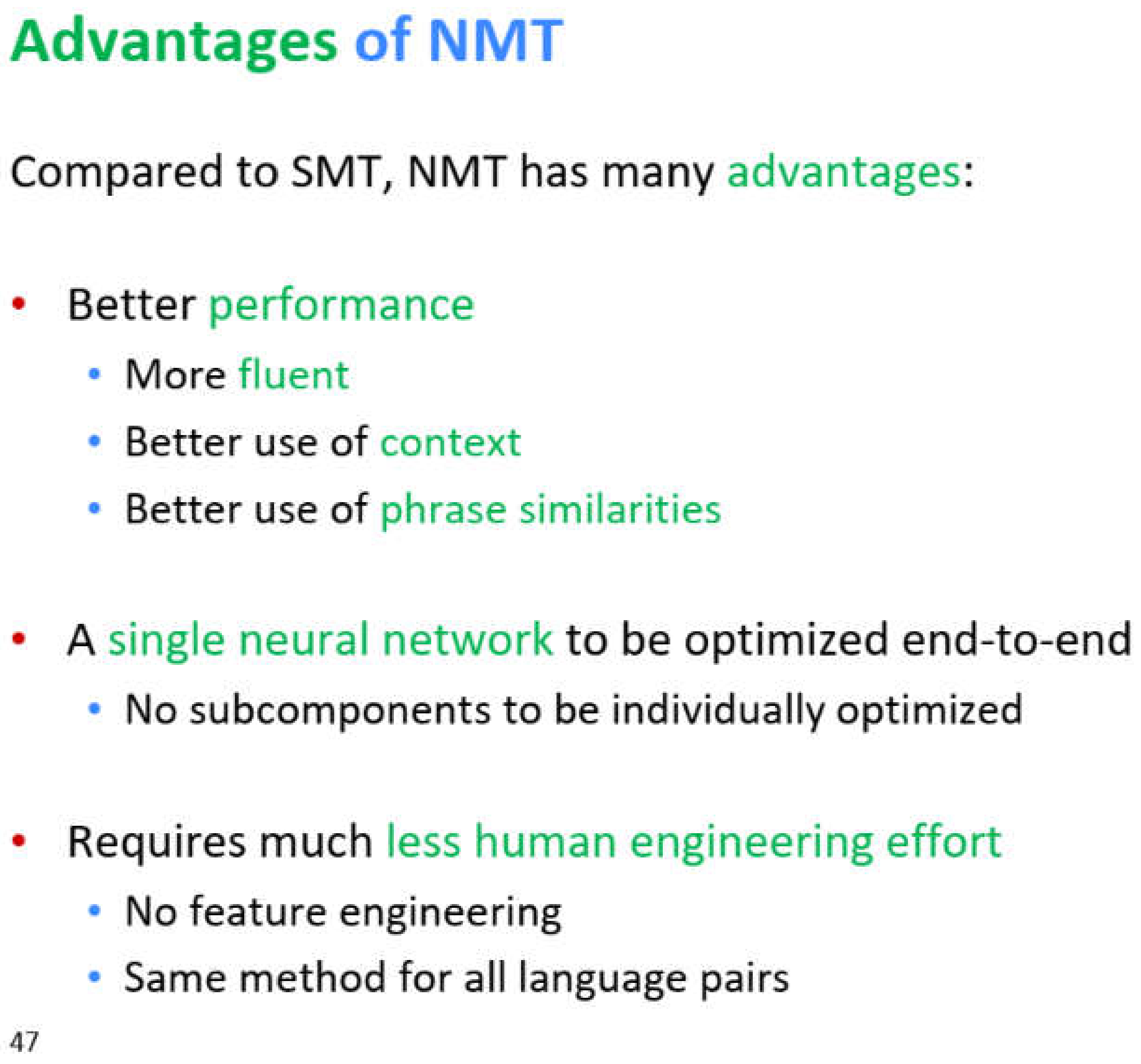
- We reach a certain predefined timestep T
- We have a certain number of completed hypothesis n
  (ie, where the hypothesis has produced)

⚠ ***Problem*** ⚠

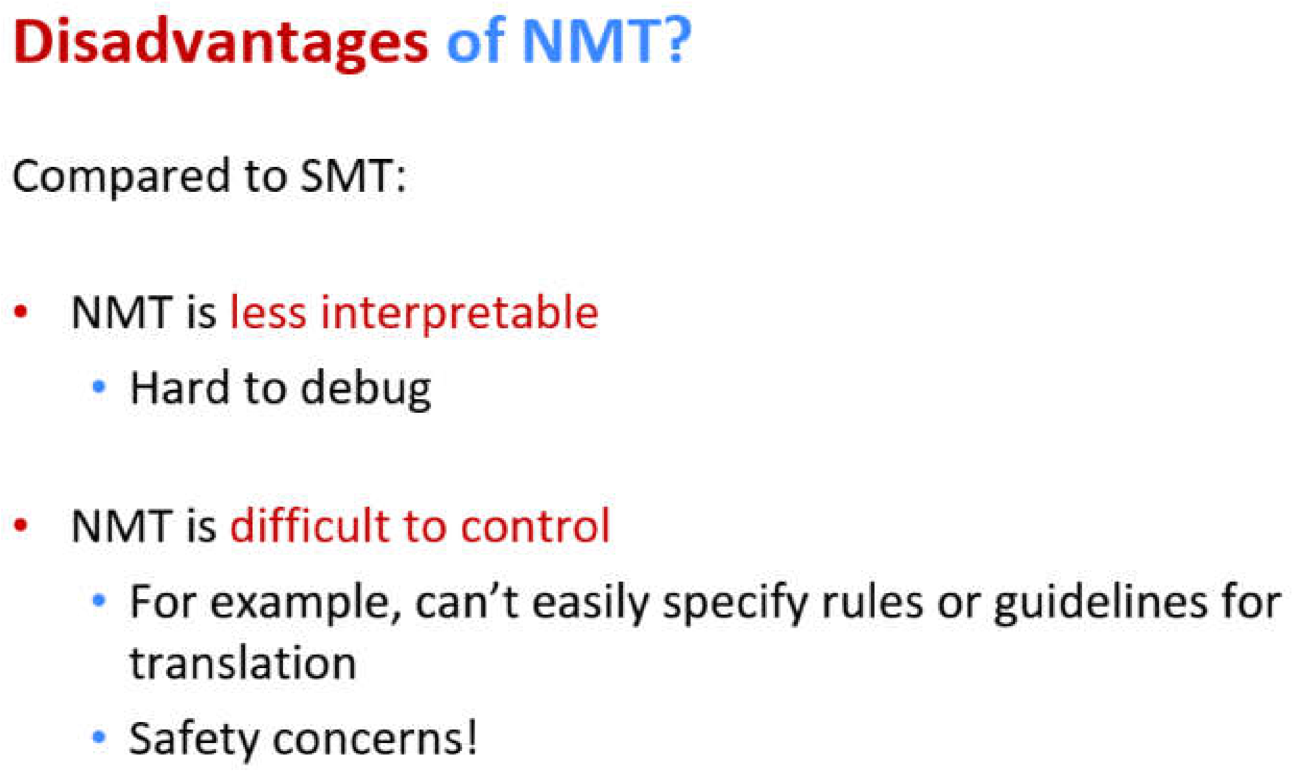
- As our hypothesis get longer, their probabilities get smaller, and thus their scores get smaller

→ Need to normalize these log-scores by the length of the sequence

Advantages of NMT



Disadvantages of NMT



- Can't create rules such as:
  "I always want to translate this word A by A' "

Evaluation of Machine Translation

BLEU (Bilingual Evaluation)

- Compare Machine Translation with several Human translations
- Compute a similarity score:
  - n-gram precision:
    "For all the n-grams that appear in the Machine Translation, how many actually appear in one of the Human translations?"
  - Penalty score to penalize for too-short translations
    (which could try to game the n-gram precision!)

Reward translations that have a high n-gram overlap with Human translations

⚠ **Problem** ⚠
There can be several right translations, we may be giving accurate ones bad scores !

Looking back at the past

- NMT managed to outperform SMT rapidly, with only a few engineers compared to SMT !!

Remaining Difficulties

- How to translate out-of-vocabulary words?
- Domain mismatch: formal ↔ informal
- Maintaning context over long text, without getting too expensive computationally
- Low-resource language pairs (languages with not much parallel data available online)
  For low-resource languages, one of the best sources of parallel text is the Bible...
  (ex of gibberish, translated as a Biblical sentence, from Somali to English)

- Still a lack of common sense:
  There can't be "jam of paper" !!
- Keeps biases of the training data:
  (Nurse → She) & (Programmer → He)
- Uninterpretability:
  Really weird effects can happen, and it's difficult to understand why !

Attention

Why Attention?

- Our whole source sentence has been captured as the last hidden state of the Encoder RNN

Thus, we have lost all the information that is not stored there
Plus, it may suffer from a recency bias

→ Information **bottleneck** !

What is Attention?

Idea

On each step of the decoder, use direct connection to the Encoder to focus on a particular part of the source sequence

- We take the dot products between the decoder hidden state, and all the Encoder hidden states

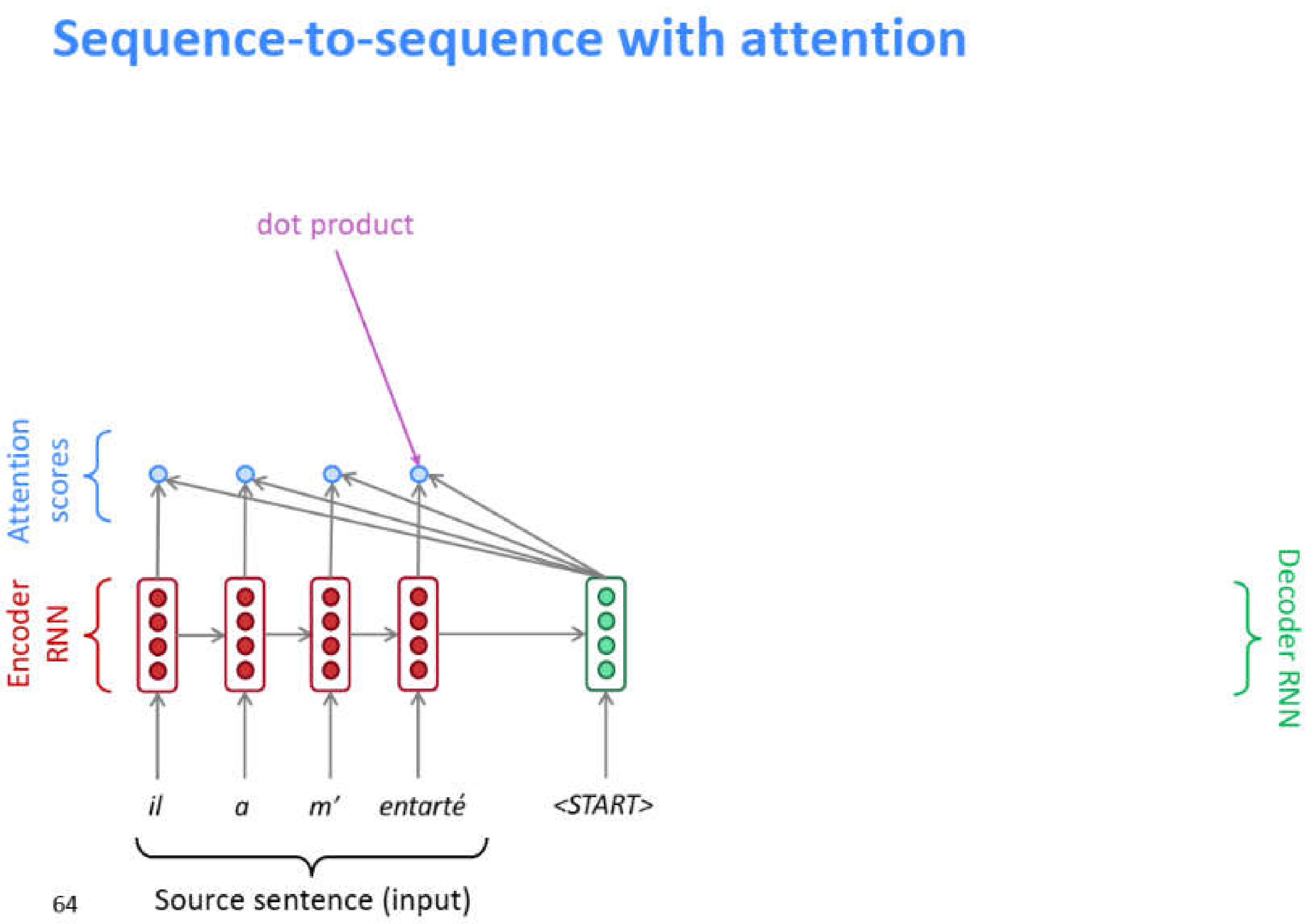
→ We get **attention scores**

- We apply softmax to all these attention scores to get a probability distribution
  to know which parts of the encoded sentence are most relevant to our Decoder hidden state
  → **Attention Distribution**

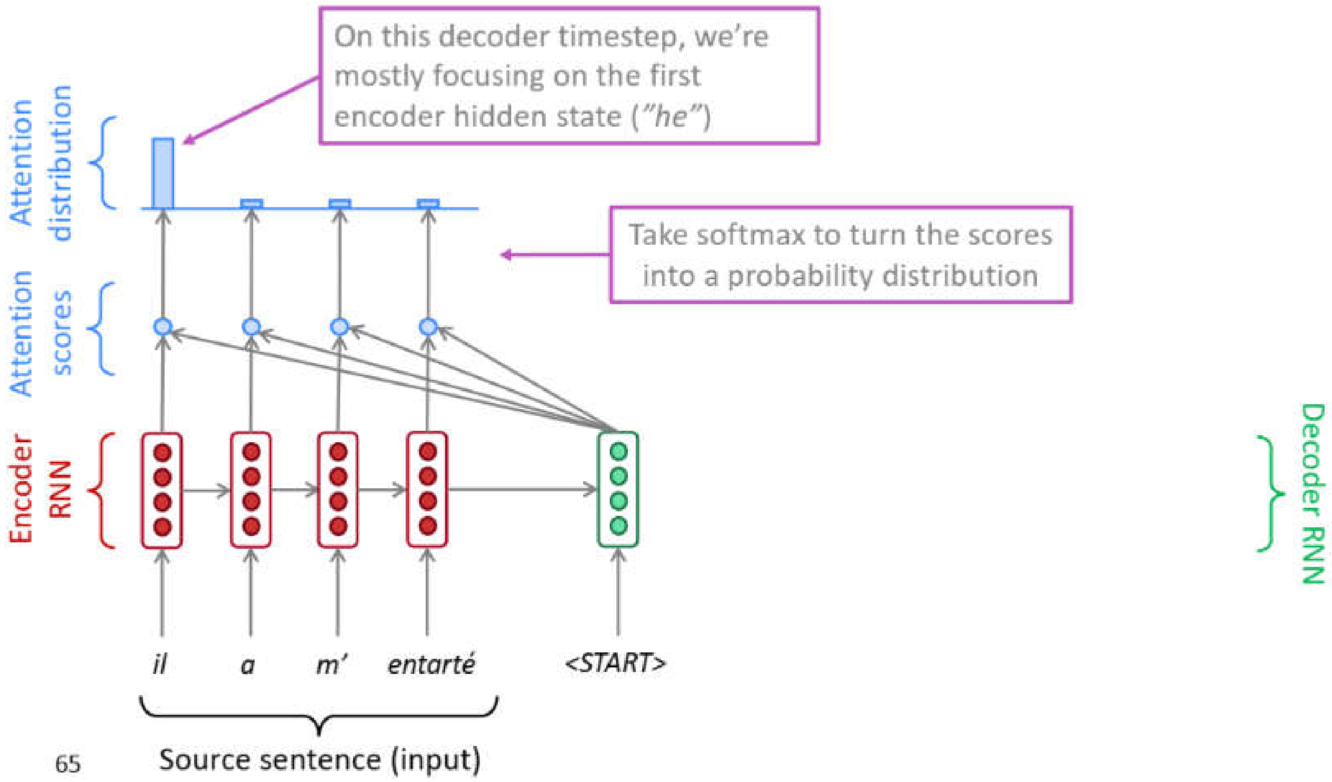
- Use the Attention Distribution to make a weighted score of the Encoder hidden states
  → **Attention Output**
  It mostly contains information from the Hidden states with high Attention !

- Concatenate this Attention Output with Decoder Hidden State, to compute our output

Example

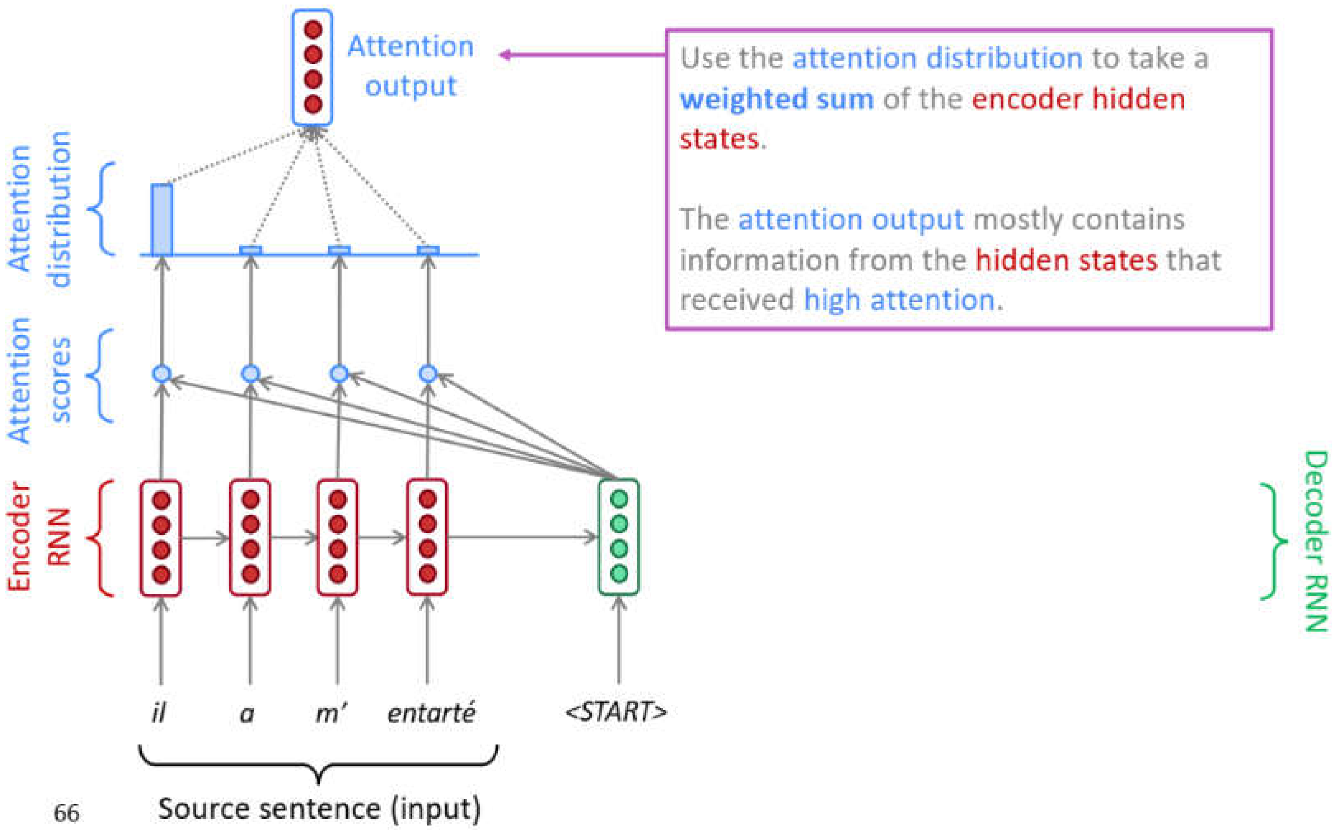


Sequence-to-sequence with attention

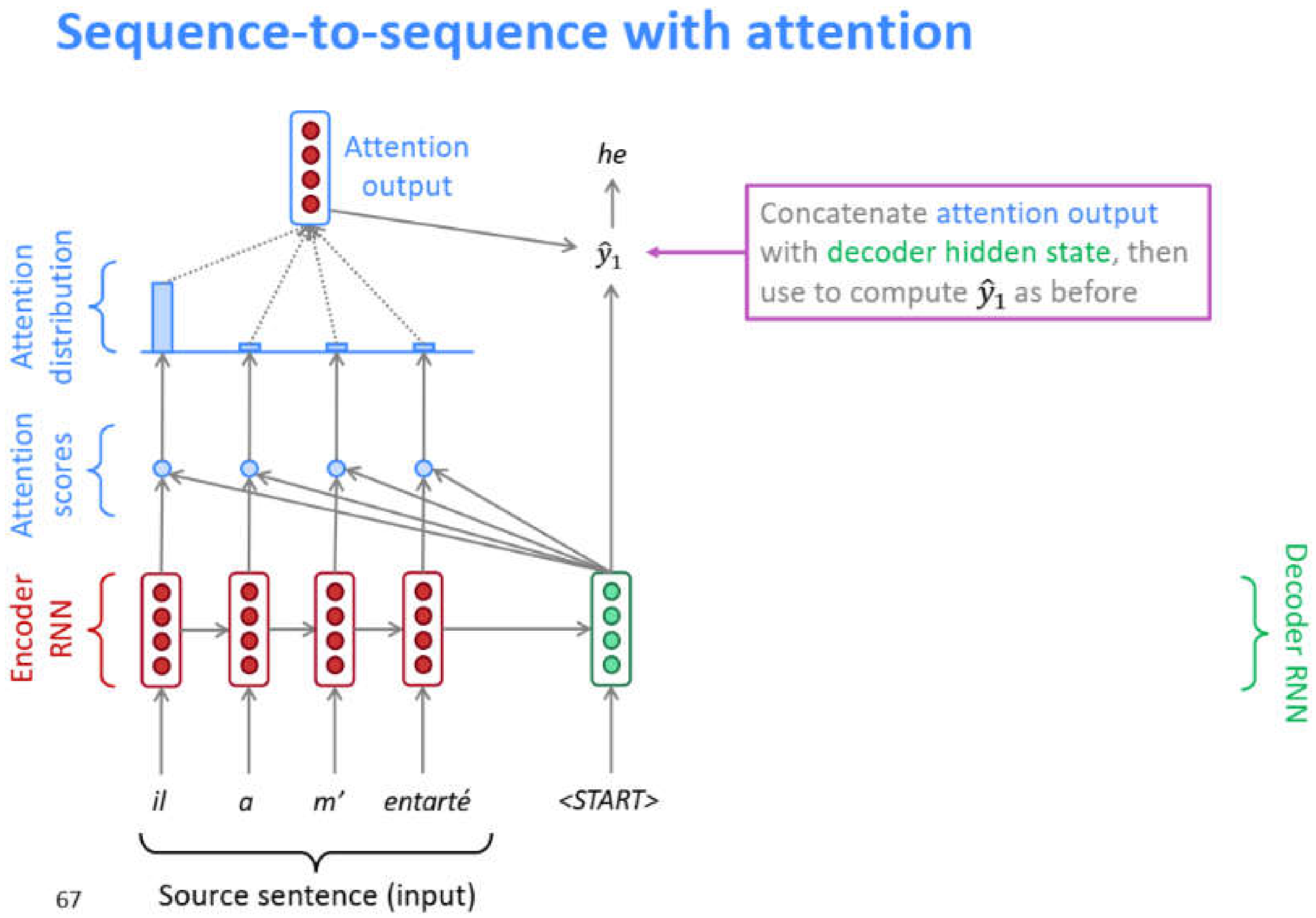


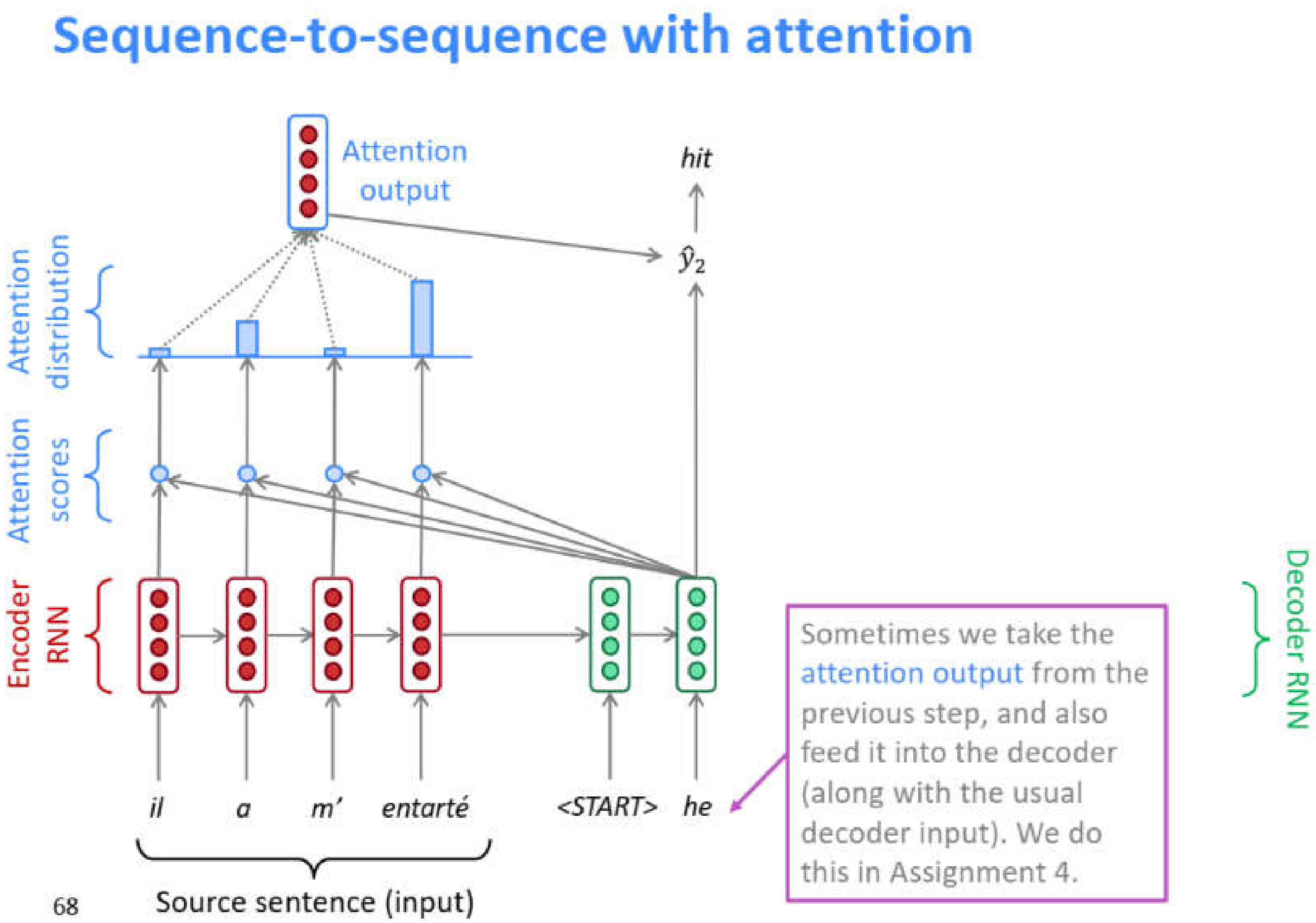
65

Sequence-to-sequence with attention



66





We can see here that Attention is a bit like a softer version of Alignment, which was much restricitve (binary)

Attention: in equations

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

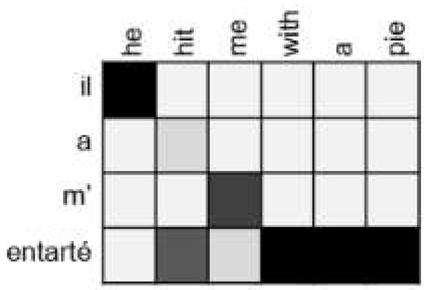
- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

73

Attention Advantages

- Attention allows Decoder to focus on only some parts of the source text
- Attention bypasses the bottleneck:
  Direct access to the source !
- The shortcut (skipping connections) helps with the **vanishing gradient** problem !!
- Provides some **interpretability** (~soft alignment)



Other Attention uses

Attention can also be used for other Deep Learning tasks

Attention needs:

- Vector values
- Vector query

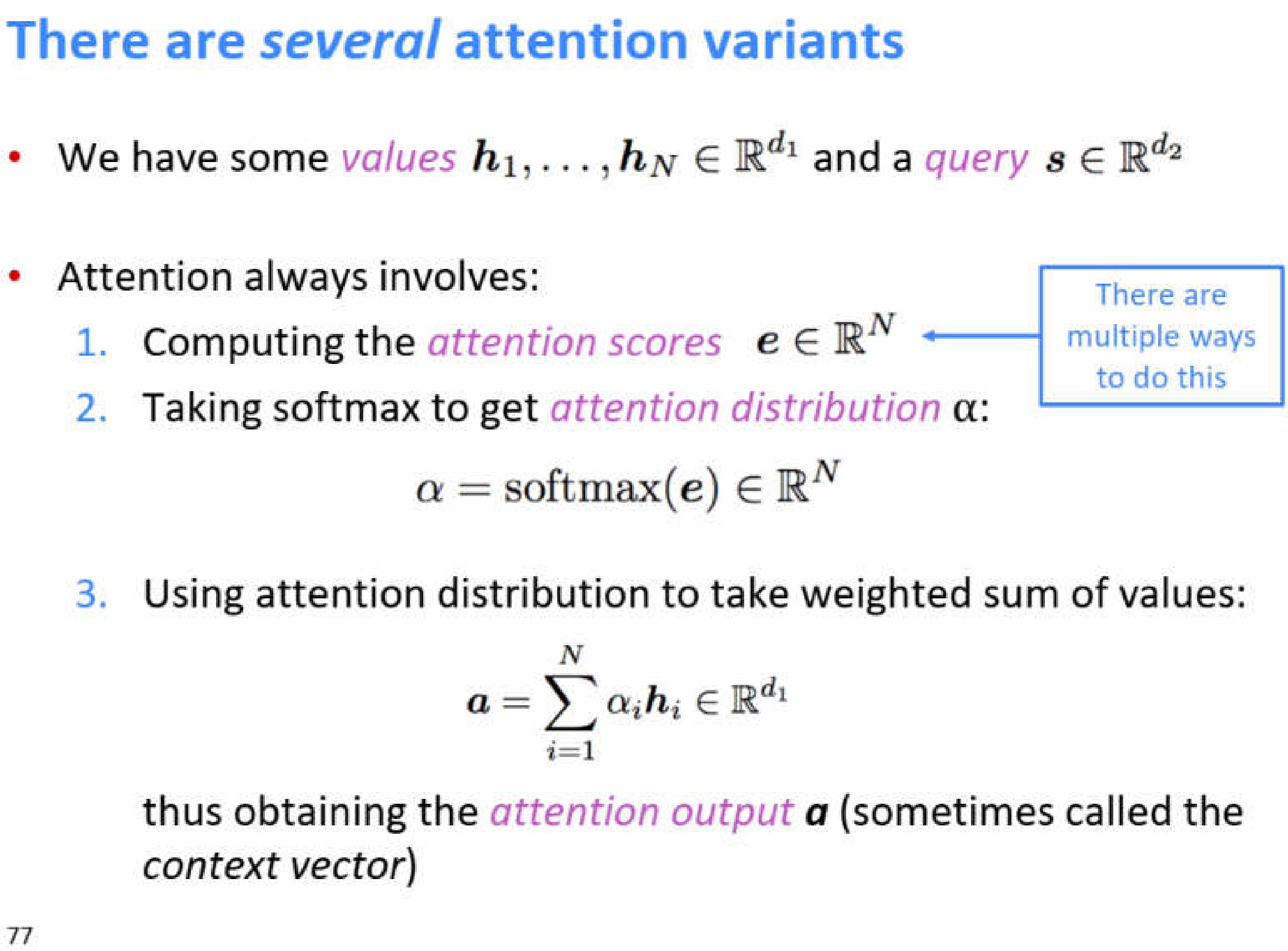
→ Compute a weighted sum of values, which is dependent on the query

- The weighted sum is a selective summary of the values information
- The query determines which values we focus on in our weighted sum

~ Similar to LSTM Gates, which depned on the context

- Attention is a way to get a **fixed-size** representation of any number of values

Variants



Attention variants

You'll think about the relative advantages/disadvantages of these in Assignment 4!

There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$:

- Basic dot-product attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier
- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - Where $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\boldsymbol{v} \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter

More information:
"Deep Learning for NLP Best Practices", Ruder, 2017. http://ruder.io/deep-learning-nlp-best-practices/index.html#attention
"Massive Exploration of Neural Machine Translation Architectures", Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

78

Lecture Summary

Summary of today's lecture

- We learned some history of Machine Translation (MT)

- Since 2014, Neural MT rapidly replaced intricate Statistical MT

- Sequence-to-sequence is the architecture for NMT (uses 2 RNNs)

- Attention is a way to *focus on particular parts* of the input
    - Improves sequence-to-sequence a lot!