A crash course in neural machine translation

Alessandro Raganato & Jörg Tiedemann

Slide credits: Abigail See, Rico Sennrich

What will we cover?

Preliminaries

Common architectures

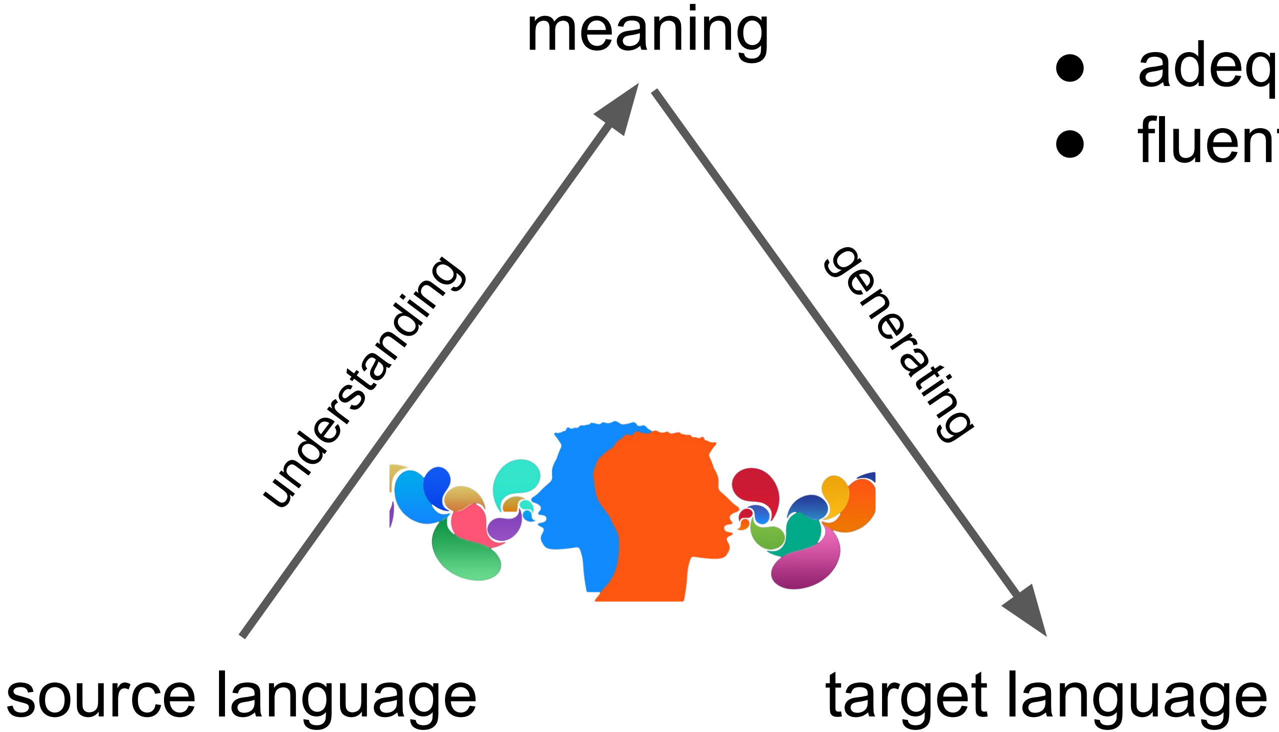
Training and decoding

Multilingual translation models

Multi-task learning and the flexibridge model

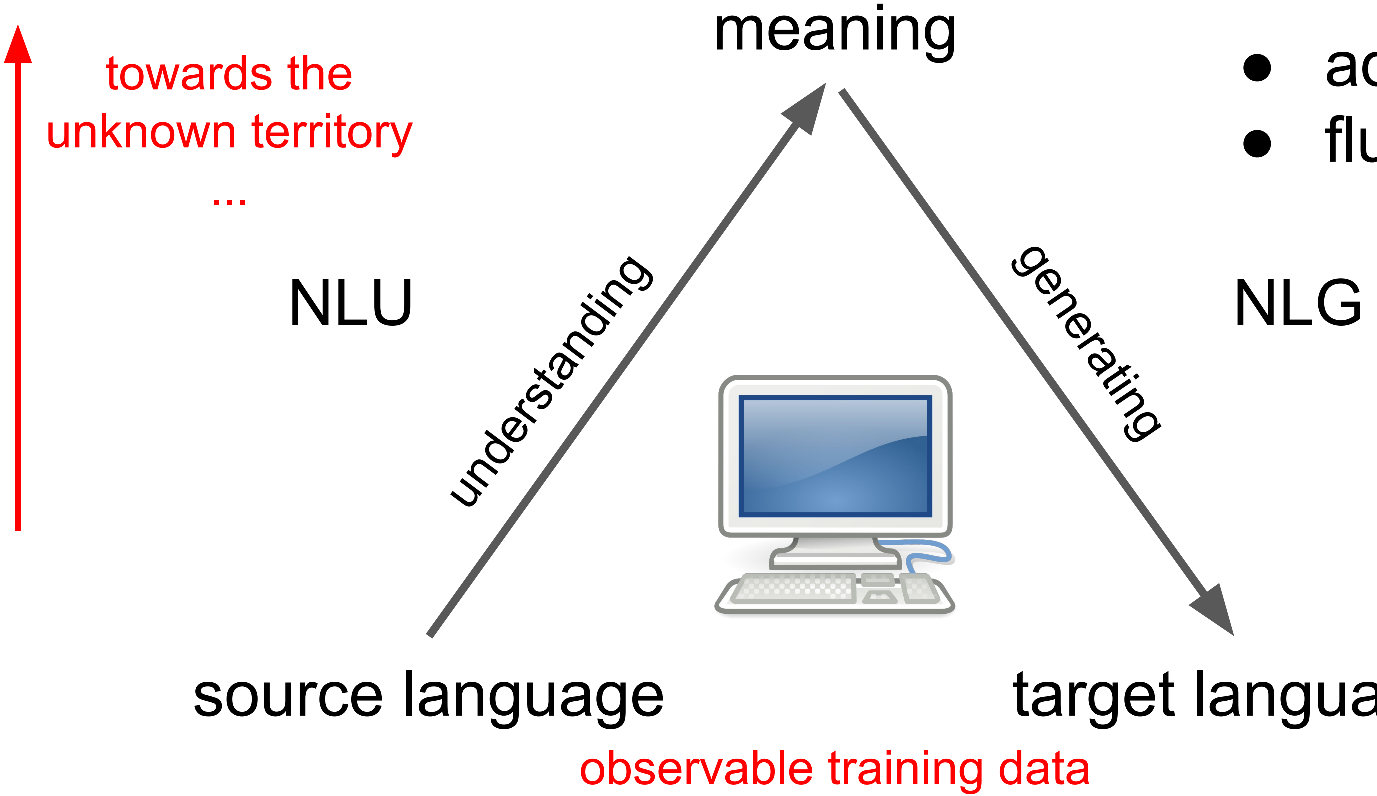
Important things we do not cover: fine-tuning / domain adaptation, document-level models, unsupervised MT, hyperparameter optimisation, data selection / augmentation / distillation, convolutional models, multi-source, factors, ...

What is translation?



- adequate
- fluent

What is machine translation?



- adequate
- fluent

What is machine translation?

Natural task

- Naturally occurring data (no annotation needed)
- Real world application and big demand

Some issues

- Typically there is no single-best solution (ambiguity, interpretation, context)
- Not clear what is a good translation (the challenge of evaluation)
- Limited data for most language pairs and domains (the challenge of training)

Practical considerations

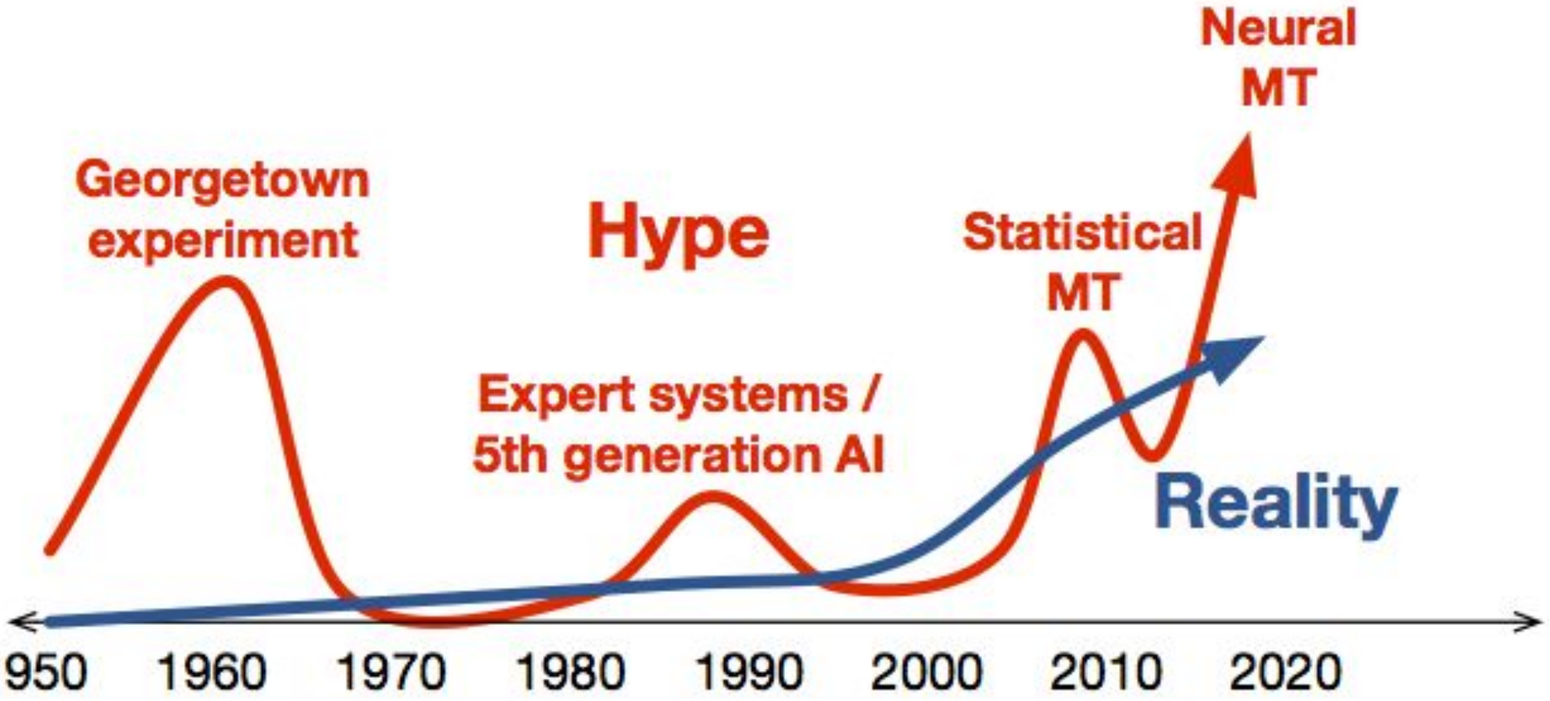
Traditional modeling assumptions

- Text input and text output
- Translate sentences instead of documents (ignore discourse-wide context)
- Tokenize sentences into sequences of words or subword units
- Any training data is good and our idea about “domain” is very fuzzy ...

Evaluation approaches

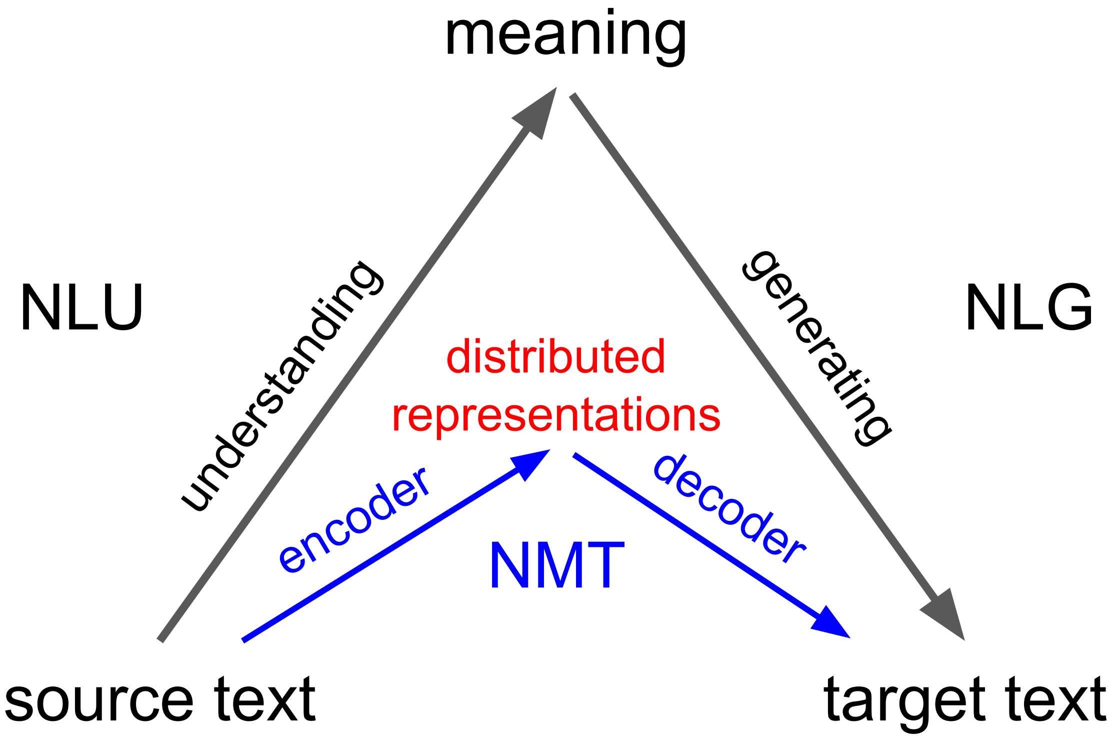
- Comparison to human reference translations (rough metrics)
- Subjective manual evaluation with some statistical analyses
- Task-based evaluation (keystrokes for post-editing efforts …)

The machine translation hype cycle

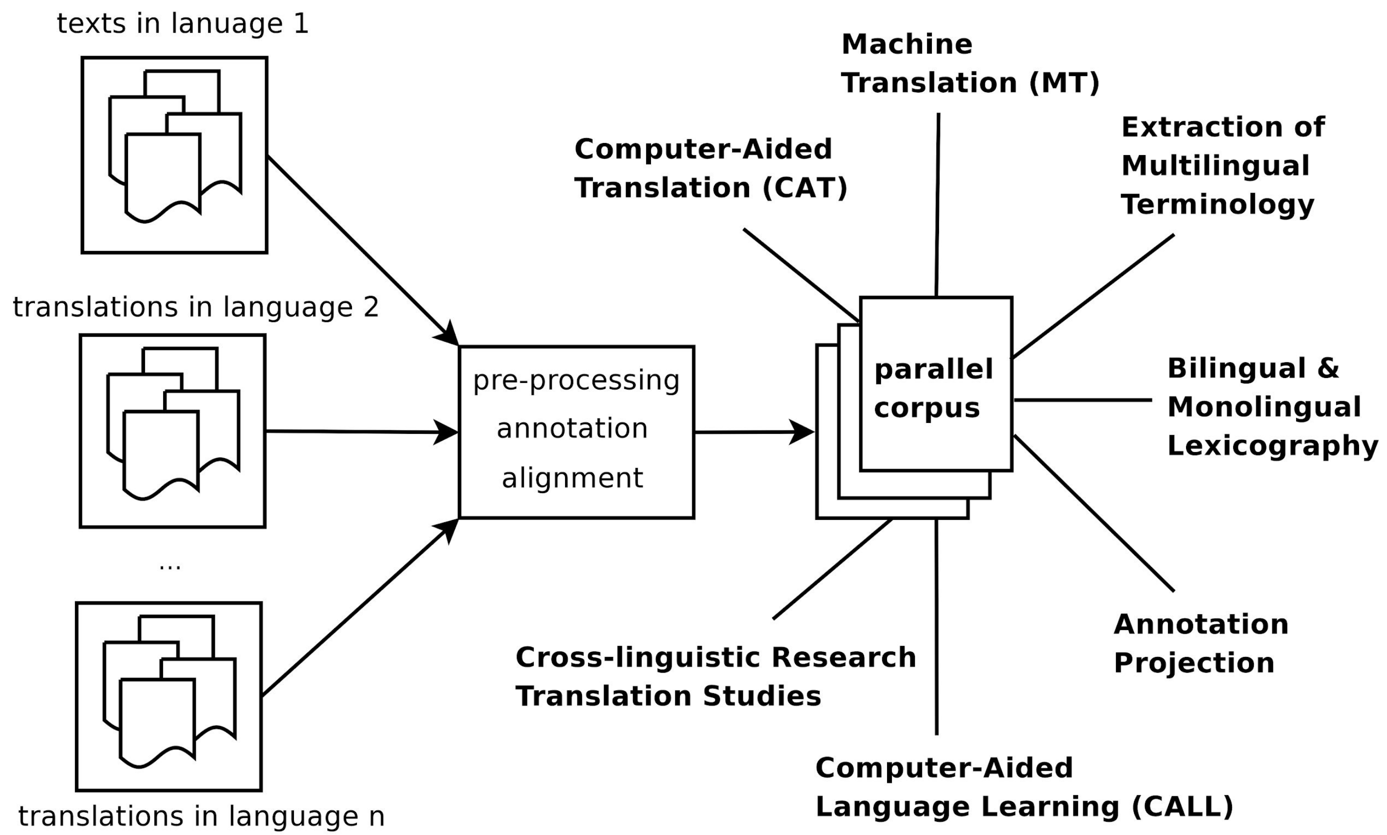


from Philipp Koehn: Neural Machine Translation

What is neural machine translation?



Training data



link to corpus website

select source language

select target language

select size

UD parsed

word alignment

bilingual dictionaries

alternative alignments

Search & download resources: en (English) | fr (French) | >10M

Language resources: click on [tmx | moses | xces | lang-id] to download the data! (raw = untokenized, ud = parsed with universal dependencies, alg = word alignments and phrase tables)

corpus	doc's	sent's	en tokens	fr tokens	XCES/XML	raw	TMX	Moses	mono	raw	ud	alg	dic	freq			other files
EUbookshop	16947	10.8M	406.8M	431.8M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr		en-fr		en fr	[query]	[sample]	
MultiUN	87480	14.2M	373.8M	454.6M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr		en-fr		en fr	[query]	[sample]	
OpenSubtitles2018	55650	45.2M	363.4M	338.0M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr	en fr	en-fr	dic	en fr	[query]	[sample]	[alt]
OpenSubtitles2016	44253	37.3M	299.0M	276.0M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr	en			en fr	[query]	[sample]	
DGT	26879	3.1M	72.8M	68.7M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr	en fr	en-fr	dic	en fr	[query]	[sample]	
Europarl	9428	2.1M	59.9M	65.7M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr	en fr	en-fr	dic	en fr	[query]	[sample]	
JRC-Acquis	12056	0.8M	34.2M	36.4M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr				en fr		[sample]	
Wikipedia	2	0.8M	23.0M	17.8M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr		en-fr	dic	en fr	[query]	[sample]	
EMEA	1933	1.1M	12.0M	14.8M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr	en fr	en-fr	dic	en fr	[query]	[sample]	
GlobalVoices	14501	0.3M	7.0M	7.4M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr		en-fr	dic	en fr	[query]	[sample]	
ECB	1	0.2M	5.7M	6.5M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr		en-fr		en fr	[query]	[sample]	
News-Commentary11	7398	0.2M	6.7M	5.2M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr		en-fr	dic	en fr	[query]	[sample]	
GNOME	2293	0.9M	5.6M	5.3M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr				en fr		[sample]	
News-Commentary	1	0.2M	4.7M	5.4M	[xces en fr]	[en fr]	[tmx]	[moses]	en fr	en fr						[sample]	
total	278822	117.2M	1.7G	1.7G	117.2M		89[illegible]M	108[illegible]M									

data in XML (tokenized)

untokenized XML

bilingual TMX

aligned plain text

monolingual plain text

word frequencies

search interface

alignment sample

= training data for machine translation

OPUS

http://opus.nlpl.eu

OpusTools (https://pypi.org/project/opustools/)

Convenient tools for accessing and processing OPUS data:

- **opus_read:** read parallel data sets and convert to different output formats
- **opus_express:** Create test/dev/train sets from OPUS data.
- **opus_cat:** extract given OPUS document from release data
- **opus_get:** download files from OPUS
- **opus_langid:** add language ids to sentences in xml files in zip archives
- **opus_filter:** filter out noise and select domain-specific data

Other common pre-processing tools

- Moses corpus cleaning script

training/clean-corpus-n.perl

- Moses normalization tools

tokenizer/replace-unicode-punctuation.perl
tokenizer/remove-non-printing-char.perl
tokenizer/normalize-punctuation.perl

- Basic tokenization, e.g. Moses tools:

tokenizer/tokenizer.perl -a -threads 4 -l fi

https://github.com/moses-smt/mosesdecoder
pip install mosestokenizer
pip install polyglot
pip install sacremoses

pip install subword-nmt
pip install sentencepiece
pip install tokenizers

Test data and benchmarks

Annual conference on machine translation WMT (http://statmt.org/wmt20/)

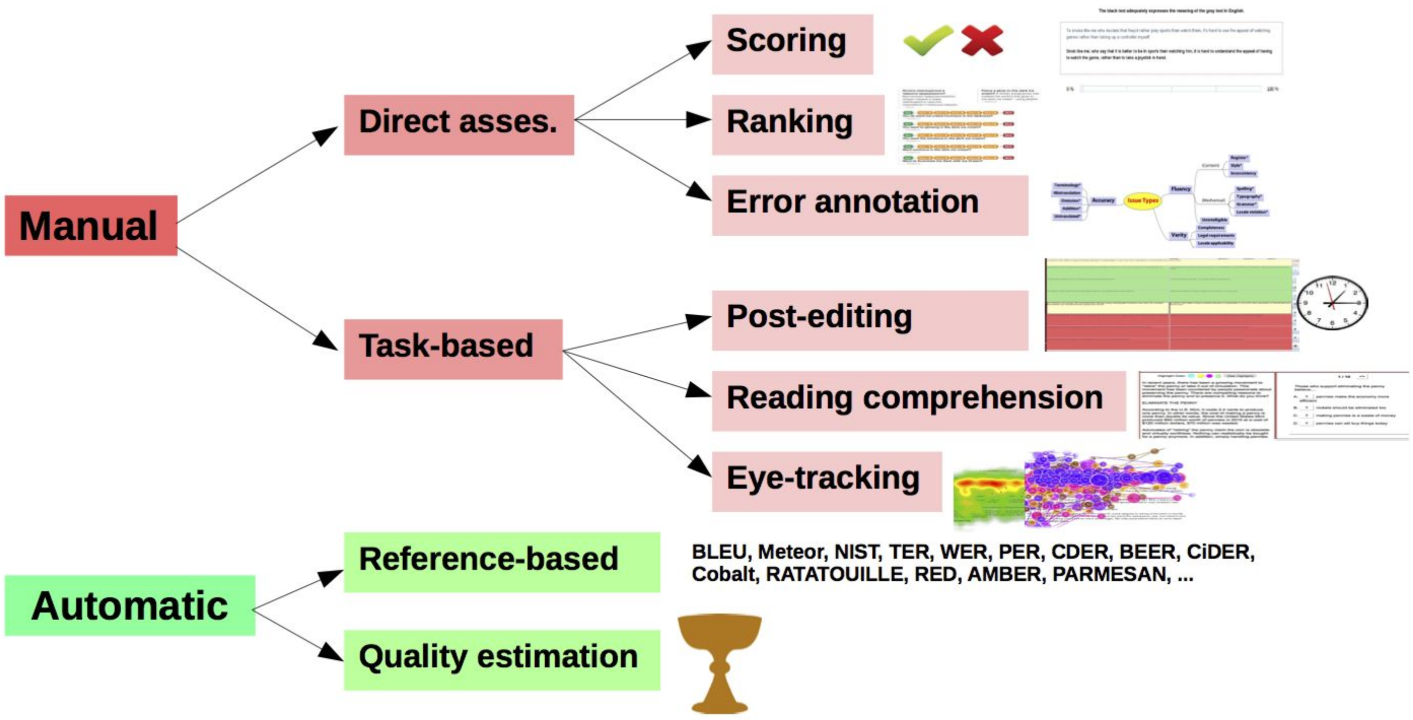
- News translation tasks (http://matrix.statmt.org)
- Special domain tasks (biomedical, similar languages, chat)
- Metrics task, post-editing, quality estimation, ...

Spoken language translation (IWSLT) (http://iwslt.org)

- Speech-to-text translation (e.g. English audio to German text)
- Different domains (speeches, conversational settings, …)

Test suites for various linguistic phenomena (agreement, ambiguity, discourse, …)

A taxonomy of MT evaluation approaches



Lucia Specia: Automatic Evaluation of Machine Translation: Moving Away from Word Matching Metrics (Gala presentation, 2016)

General idea: Train a conditional language model

美国关岛国际机场及其办公室均接获一
名自称沙地阿拉伯富商拉登等发出的电
子邮件，威胁将会向机场等公众地方发
动生化袭击後，关岛经保持高度戒备。

Training = find optimal model parameters

Translation = decoding

Probabilistic Model: $P(\mathbf{e}|\mathbf{c})$

Search problem (decoding):

$$\boldsymbol{e}^* = \underset{\boldsymbol{e}}{argmax}\ P(\boldsymbol{e}|\boldsymbol{c})$$

The U.S. island of Guam is maintaining a high state of alert after the Guam airport and its offices both received an e-mail from someone calling himself the Saudi Arabian Osama bin Laden and threatening a biological/chemical attack against public places such as the airport.

Modelling Translation

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single neural network*

Suppose we are translating English to Finnish

- We want to find best Finnish sentence $T\ (x_1, \ldots, x_m)$, given a English sentence $S\ \ (y_1, \ldots, y_n)$
- We can express translation as a probabilistic model:

$$T^* = \arg\max_T p(T|S)$$

- Expanding using the chain rule gives:

$$\begin{aligned} p(T|S) &= p(y_1, \ldots, y_n | x_1, \ldots, x_m) \\ &= \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1}, x_1, \ldots, x_m) \end{aligned}$$

Differences Between Translation and Language Model

- Target-side language model: $$p(T) = \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1})$$

- Translation model: $$p(T|S) = \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1}, x_1, \ldots, x_m)$$

- We could just treat sentence pair as one long sequence, but:
  - We do not care about $p(S)$
  - We may want different vocabulary, network architecture for source text

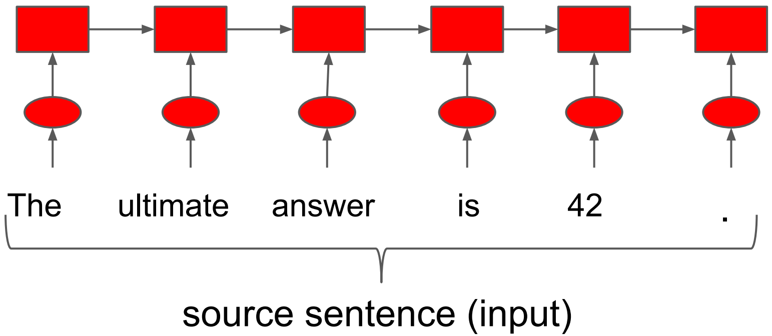
⇨ Use separate RNNs for source and target.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

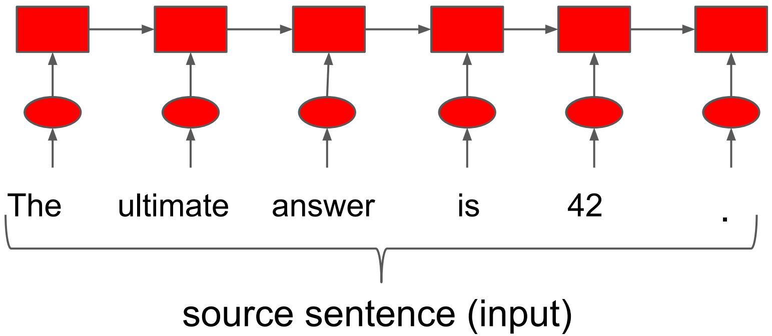
Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.



Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

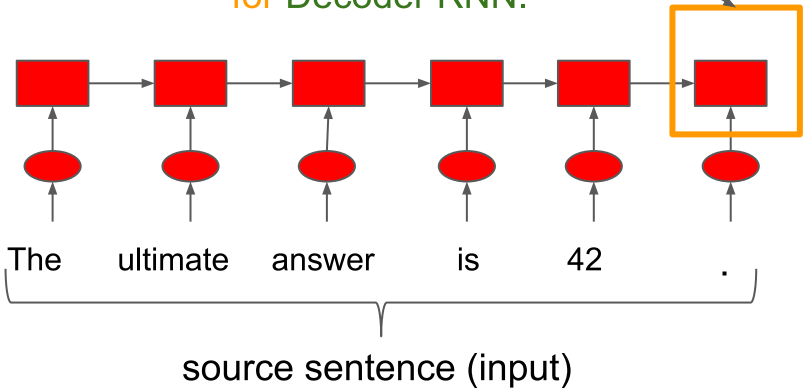


The **encoder** creates a representation of the source sentence.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

Encoding of the source sentence.
Provides initial hidden state for Decoder RNN.

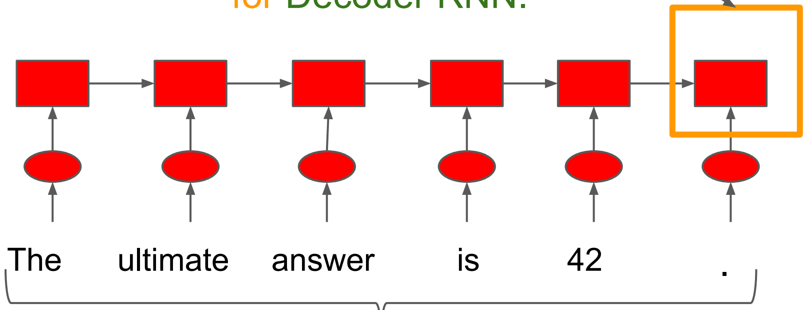


The **encoder** creates a representation of the source sentence.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

Encoding of the source sentence.
Provides initial hidden state for Decoder RNN.



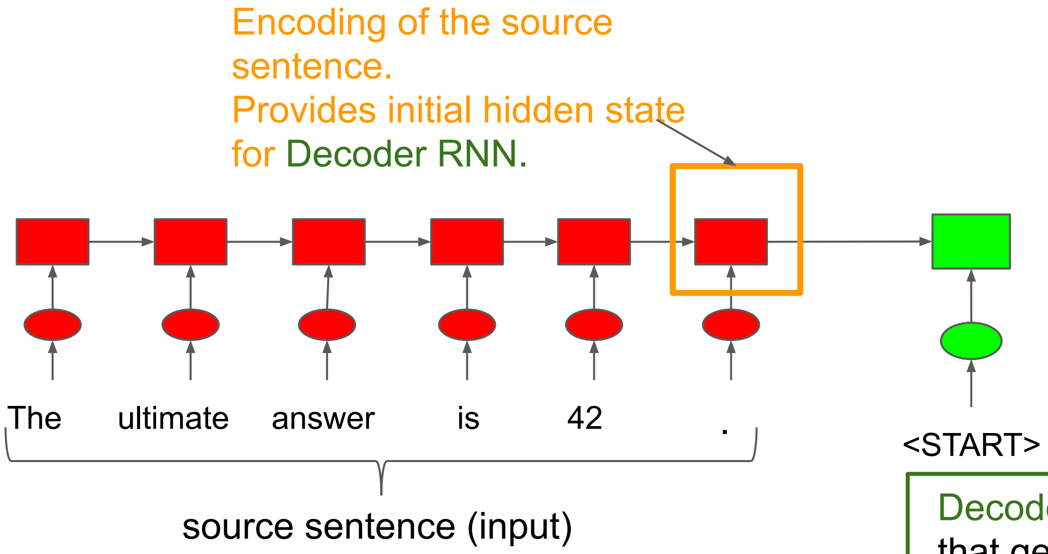
source sentence (input)

The **encoder** creates a representation of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.



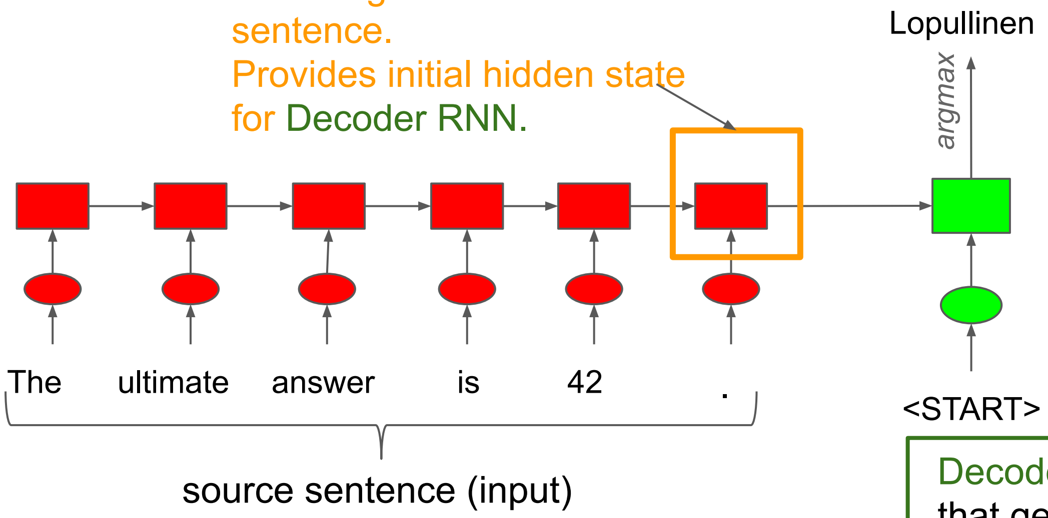
The **encoder** creates a representation of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

Encoding of the source sentence.
Provides initial hidden state for Decoder RNN.

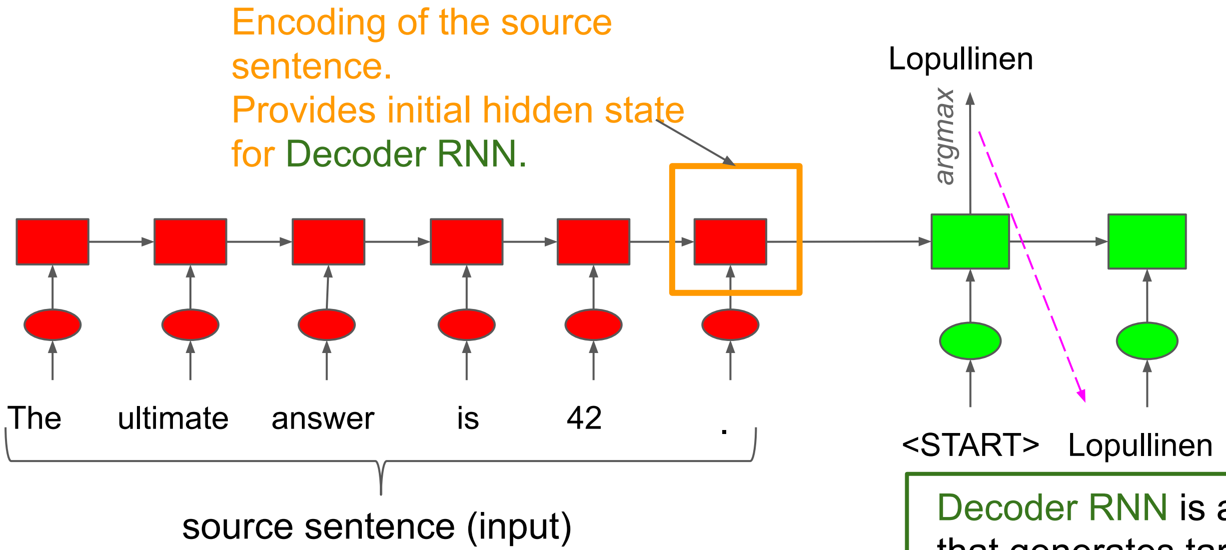


Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

The **encoder** creates a representation of the source sentence.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

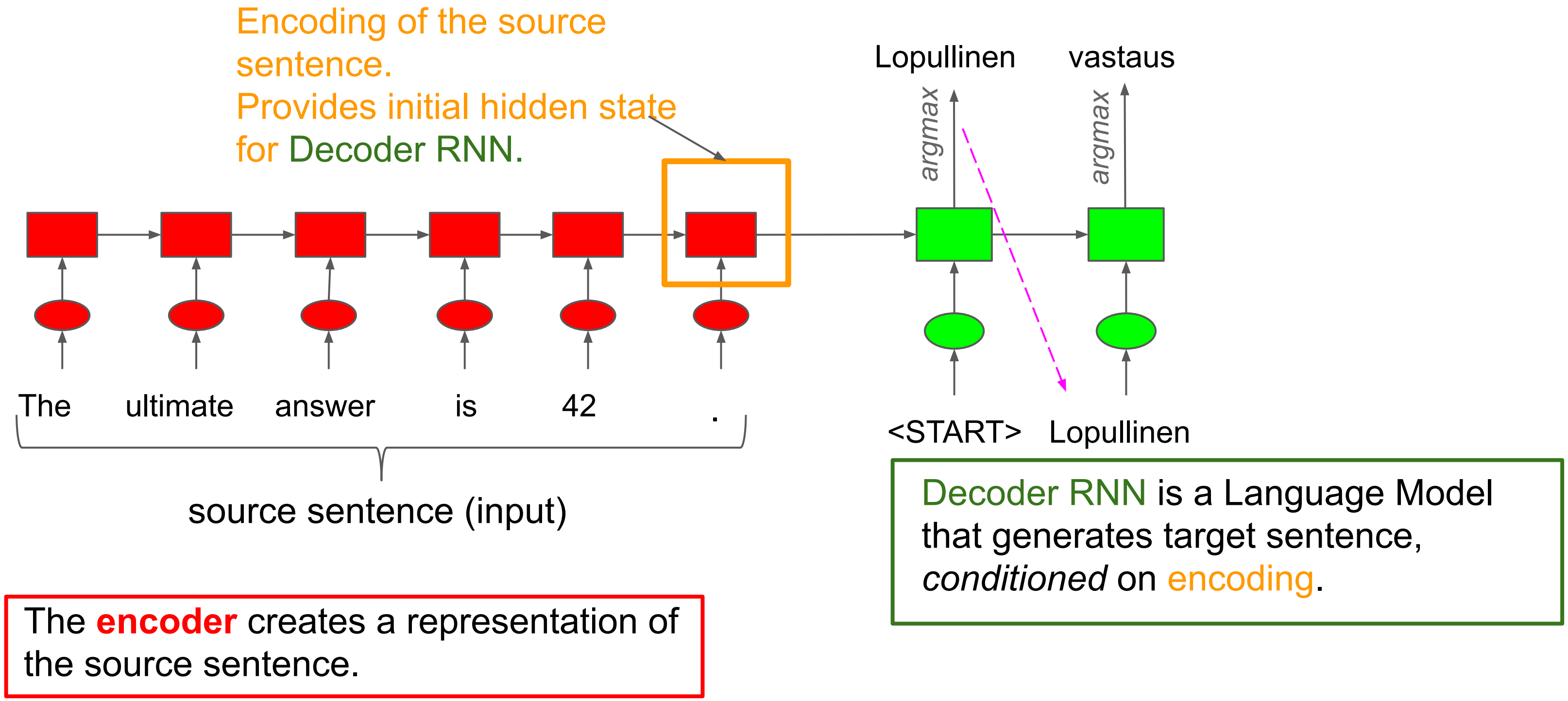


The **encoder** creates a representation of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

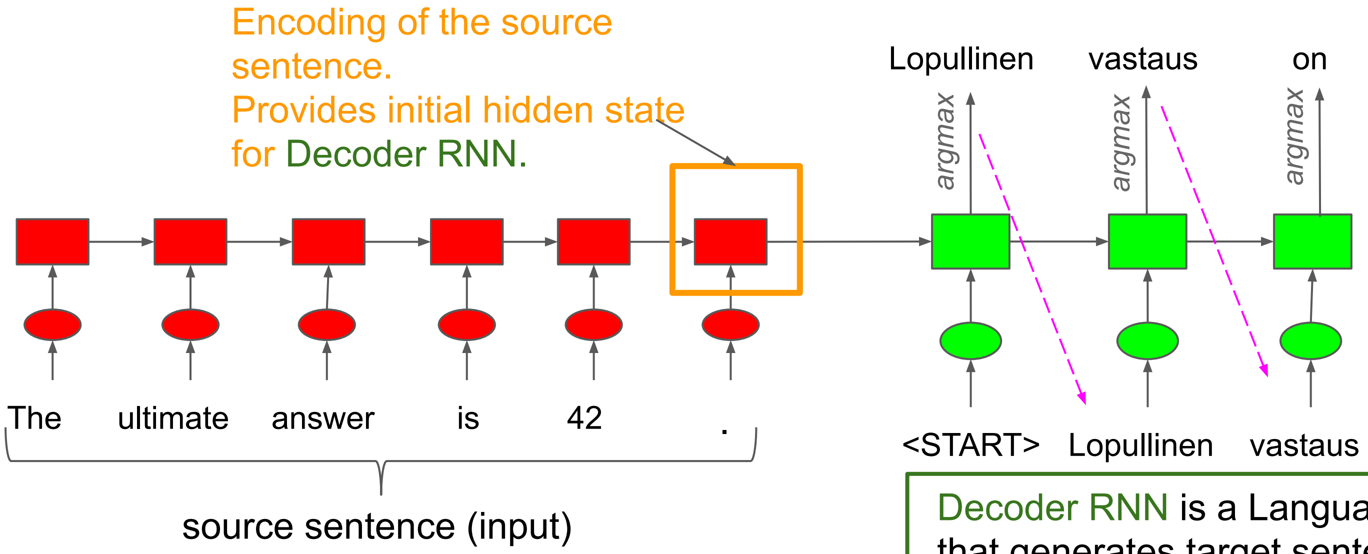
Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.



Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

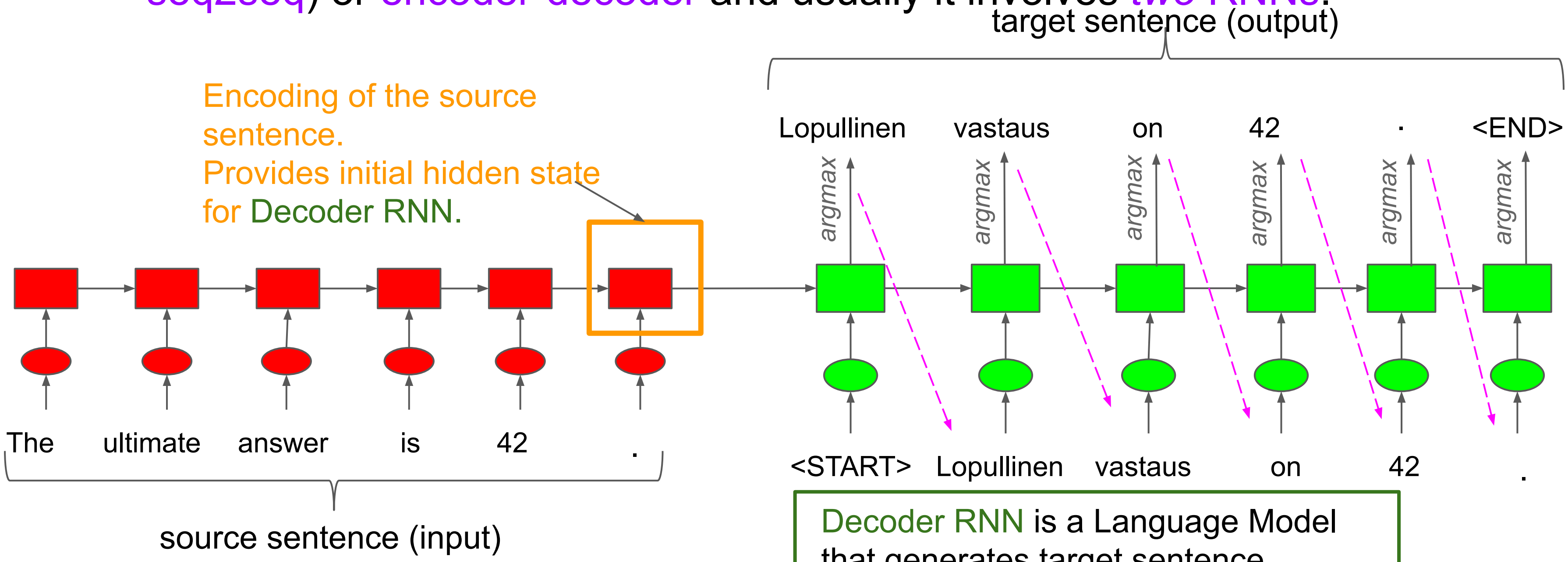


The **encoder** creates a representation of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

Neural Machine Translation (NMT)

- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.



Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

The **encoder** creates a representation of the source sentence.

Neural Machine Translation (NMT)

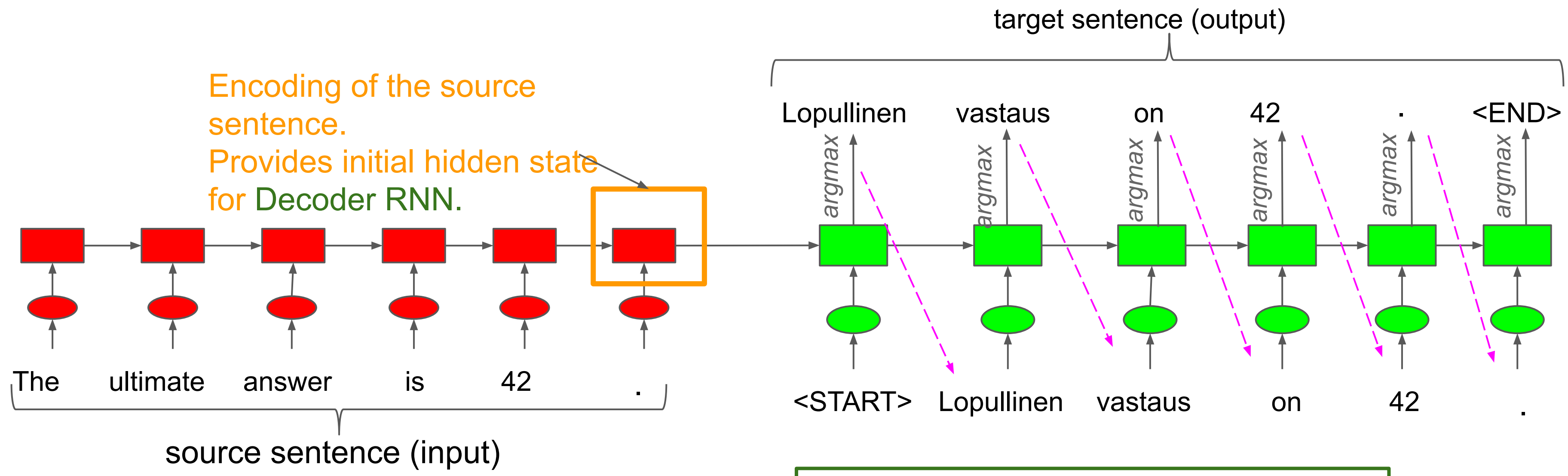
- The neural network architecture is called sequence-to-sequence (aka seq2seq) or encoder-decoder and usually it involves *two* RNNs.

- Last encoder hidden-state “summarises” source sentence

- Sequence-to-sequence is versatile! It is useful for more than just MT
- ➢ Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

NMT: Issues

- The sequence-to-sequence model is an example of a Conditional Language Model:
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

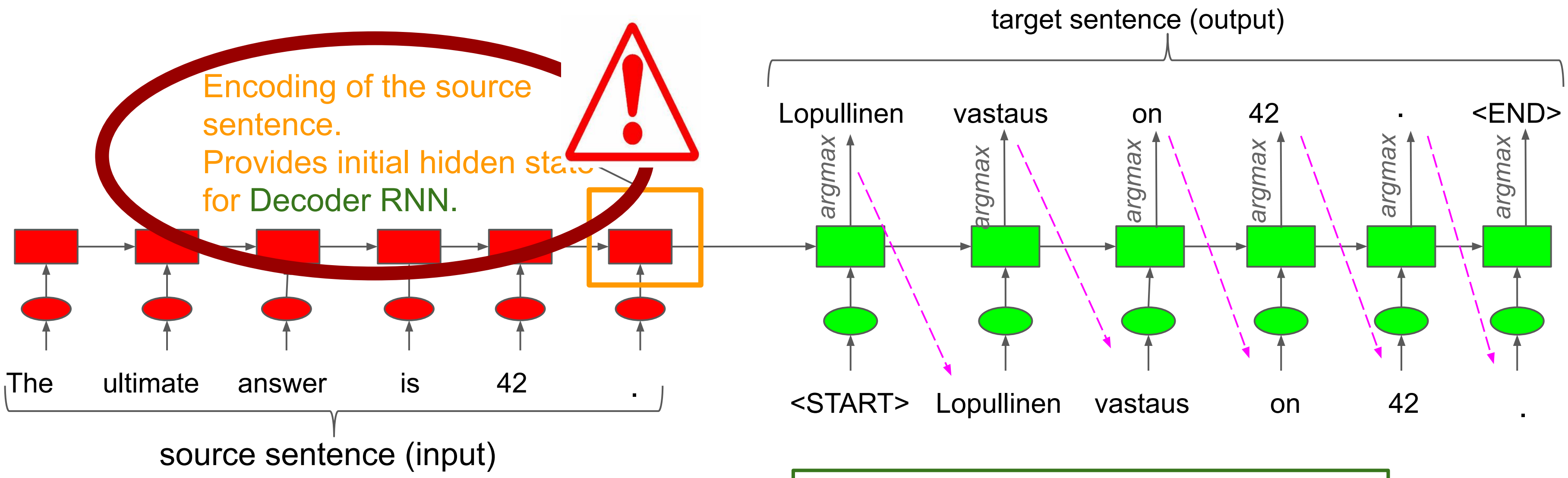


The **encoder** creates a representation of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

NMT: Issues

- The sequence-to-sequence model is an example of a Conditional Language Model:
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*



The **encoder** creates a representation of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned* on encoding.

Summary vector as information bottleneck

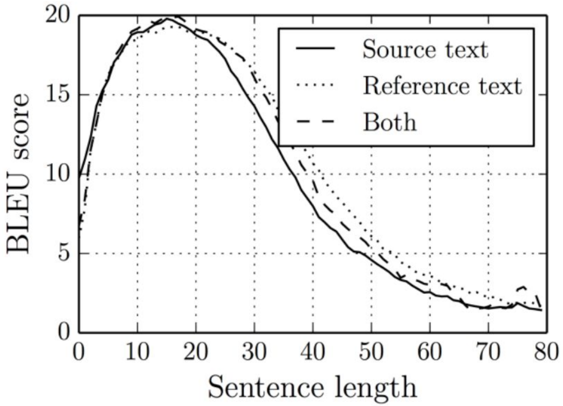
- Last encoder hidden-state “summarises” source sentence.
- This needs to capture all information about the source sentence.

- **Problem**: *Information bottleneck!*

Summary vector as information bottleneck

- Last encoder hidden-state "summarises" source sentence.
- This needs to capture all information about the source sentence.

- **Problem**: *Information bottleneck!*
- Fixed sized representation degrades as sentence length increases



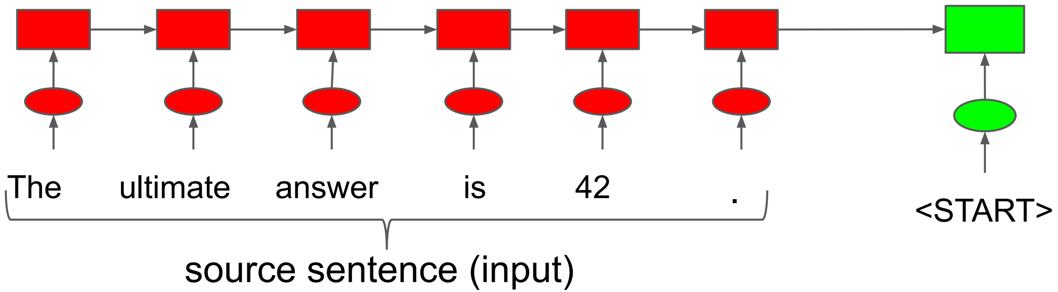
[Cho et al., 2014]

Attention

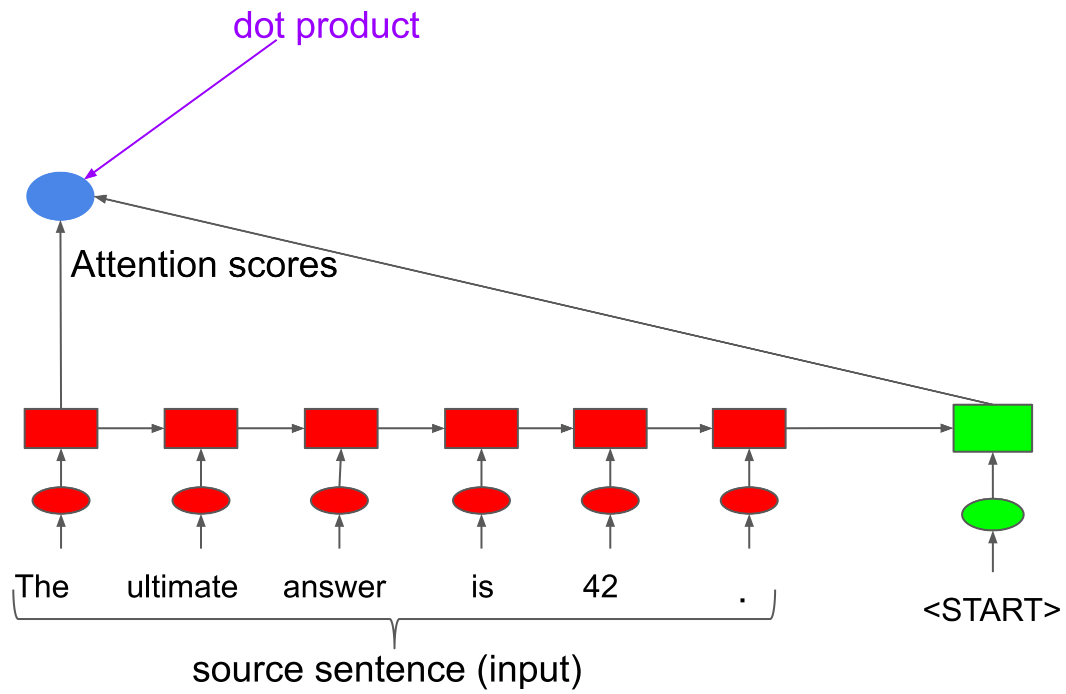
- **Attention** provides a solution to the bottleneck problem.

- Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

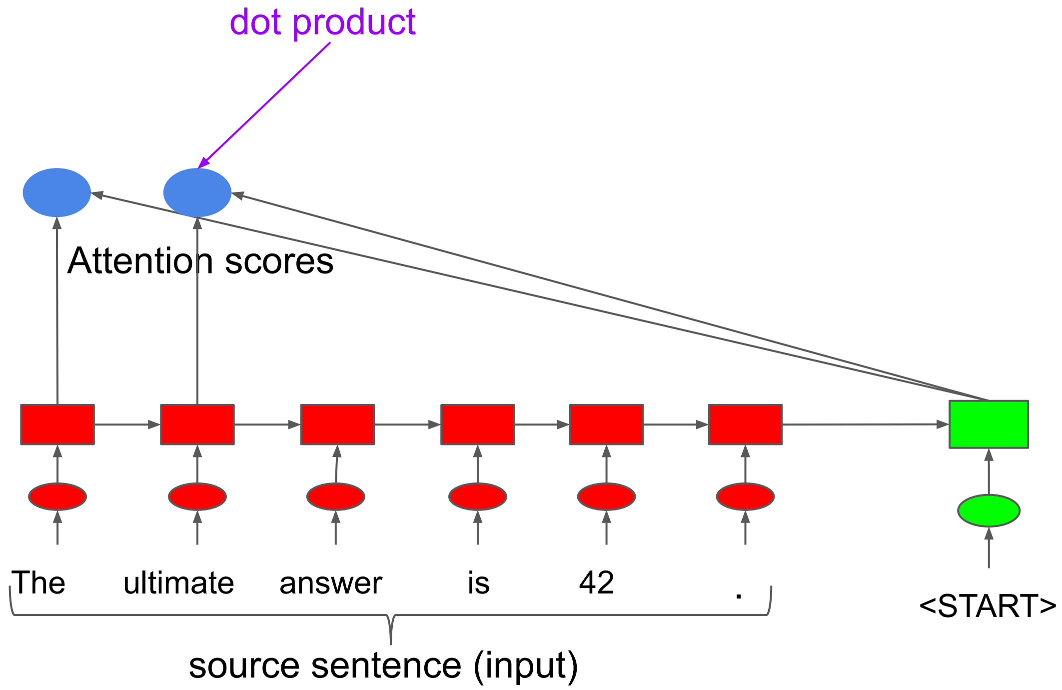
Sequence-to-sequence with attention



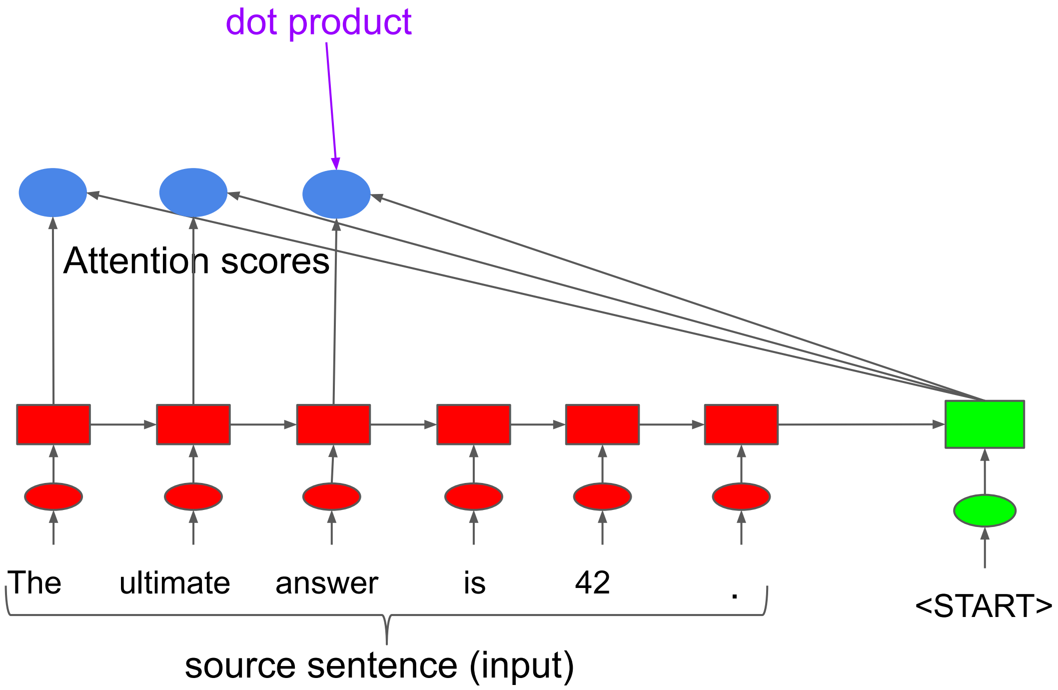
Sequence-to-sequence with attention



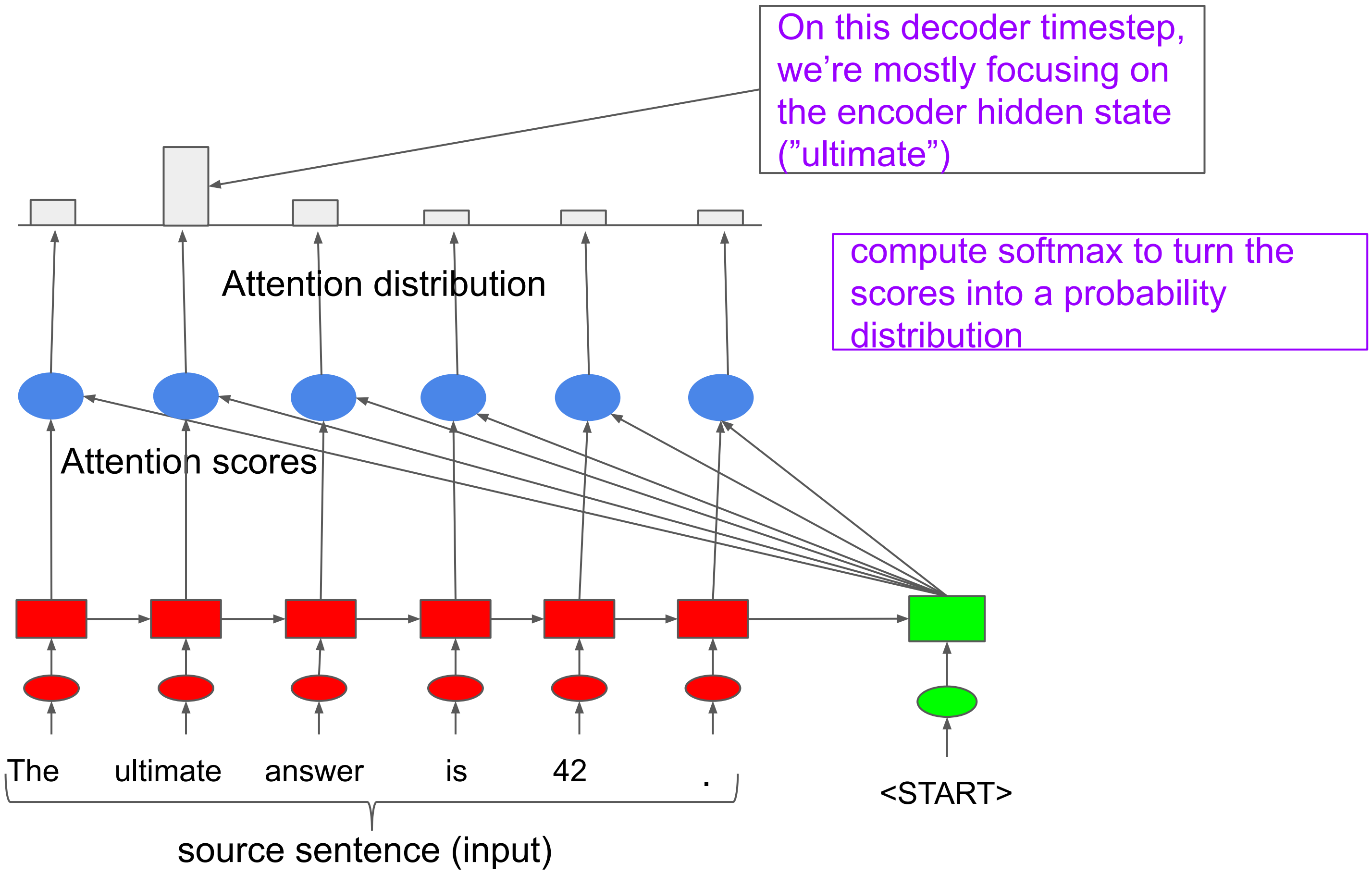
Sequence-to-sequence with attention



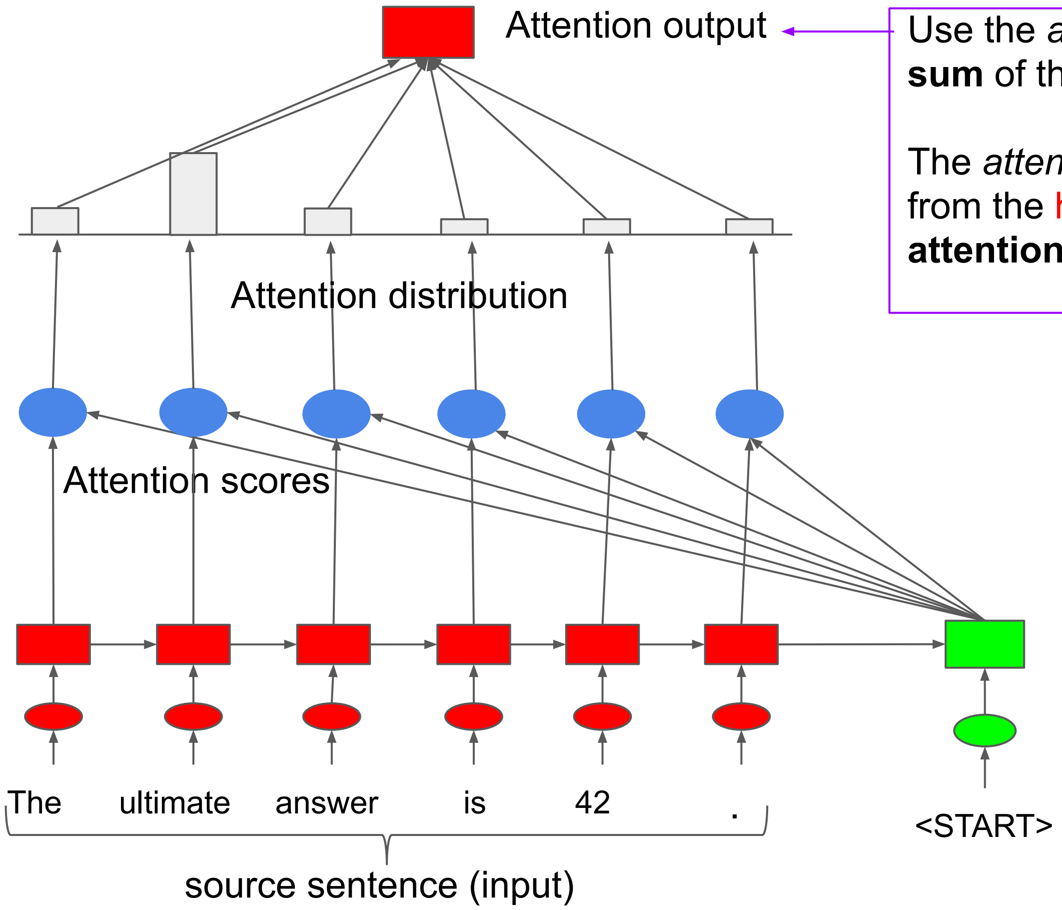
Sequence-to-sequence with attention



Sequence-to-sequence with attention



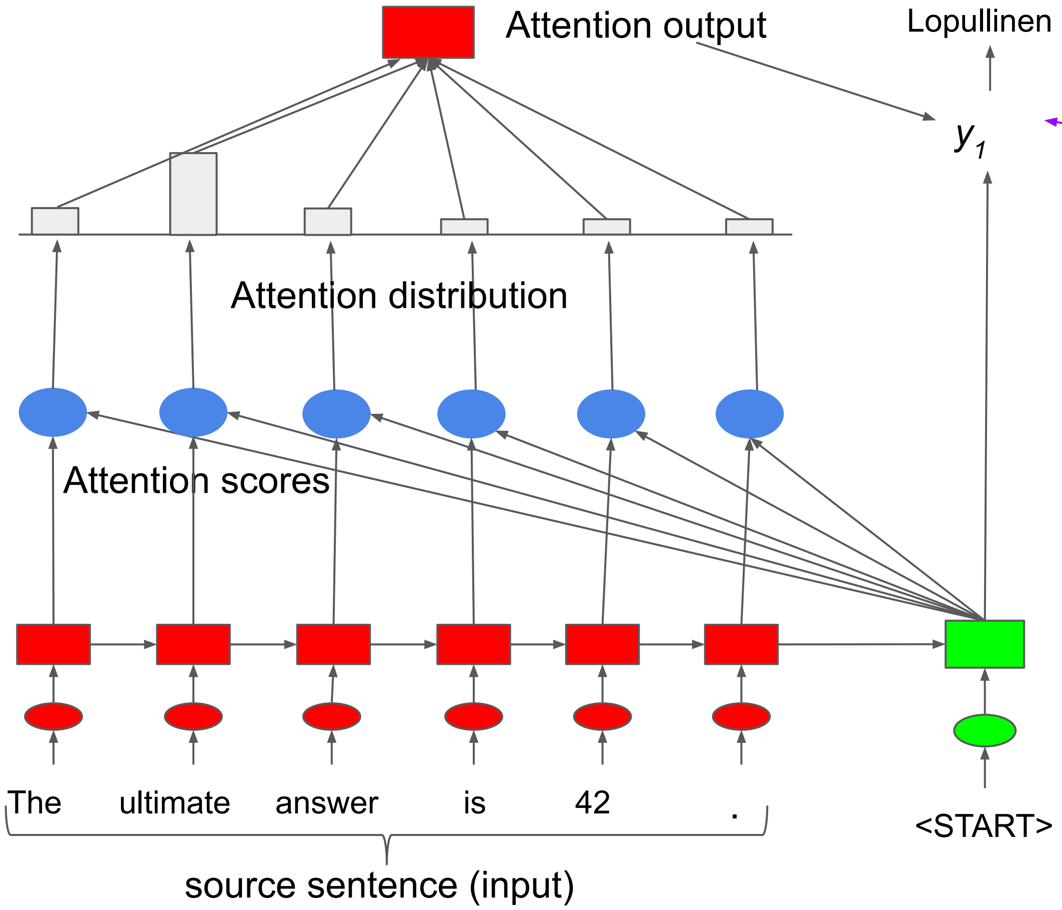
Sequence-to-sequence with attention



Use the *attention distribution* to take a **weighted sum** of the encoder hidden states.

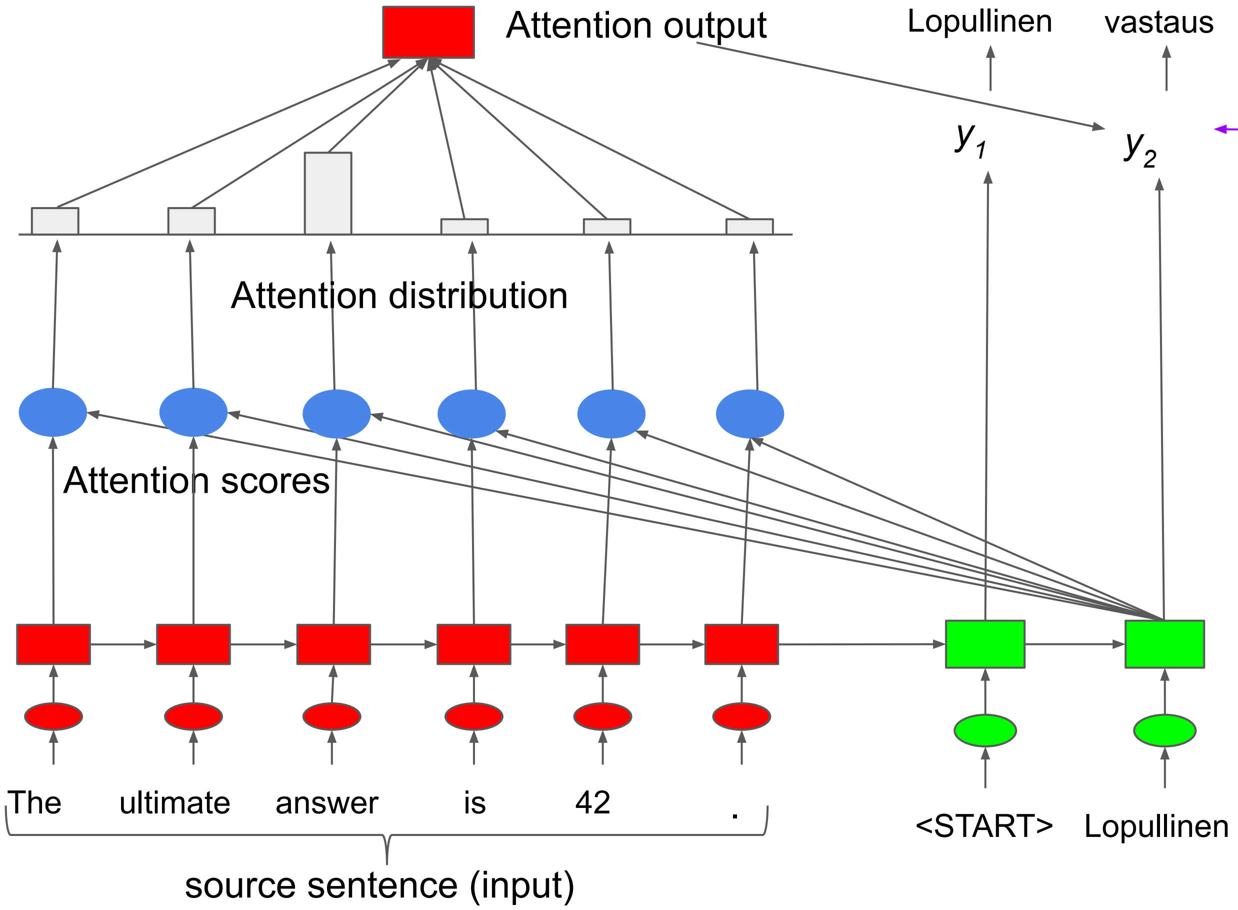
The *attention output* mostly contains information from the hidden states that received **high attention.**

Sequence-to-sequence with attention



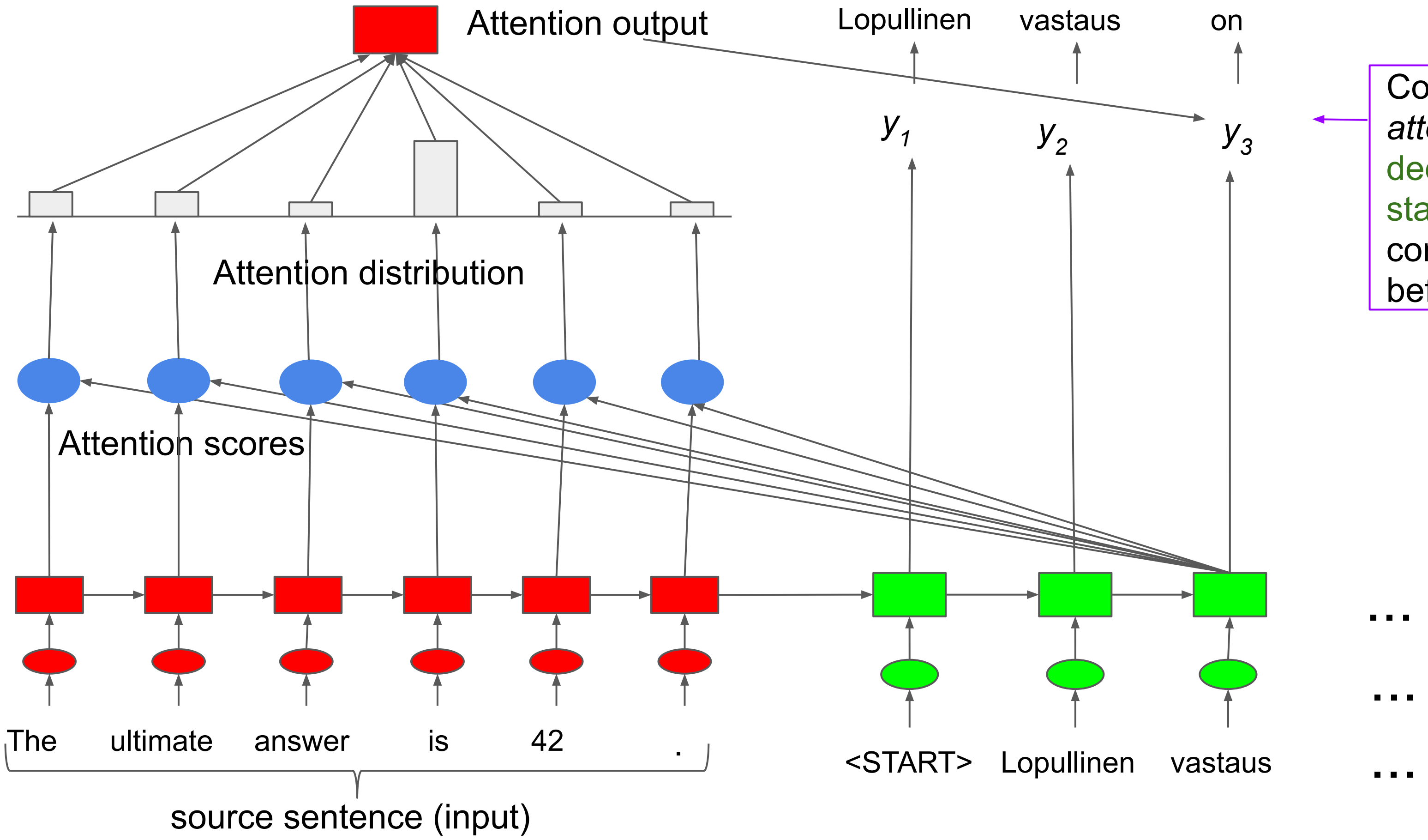
Concatenate *attention output* with decoder hidden state, then use to compute y_1 as before

Sequence-to-sequence with attention



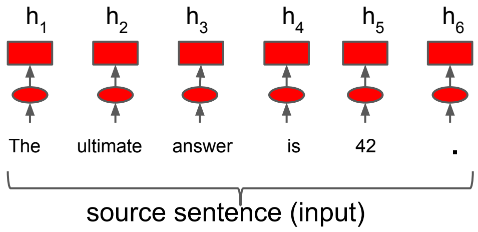
Concatenate *attention output* with decoder hidden state, then use to compute y_2 as before

Sequence-to-sequence with attention



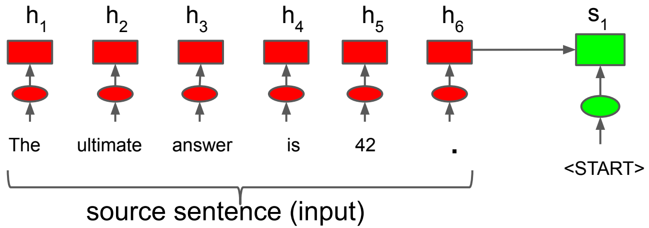
Concatenate *attention output* with decoder hidden state, then use to compute y_3 as before

Attention: in equations



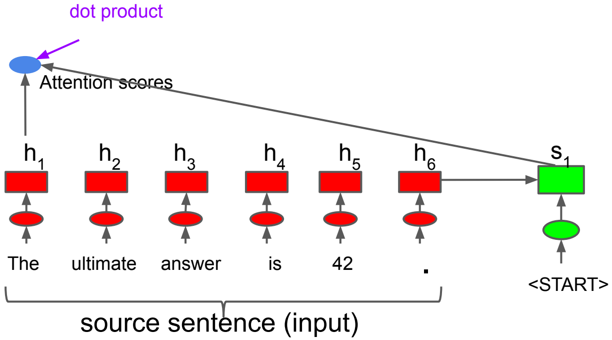
- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$

Attention: in equations



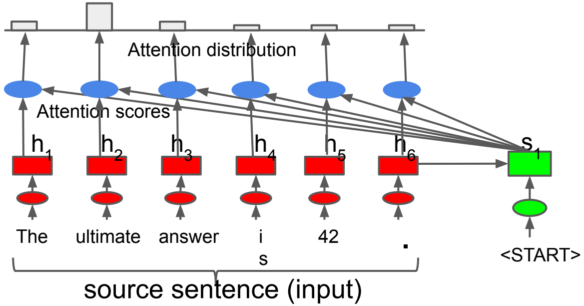
- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$

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:

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



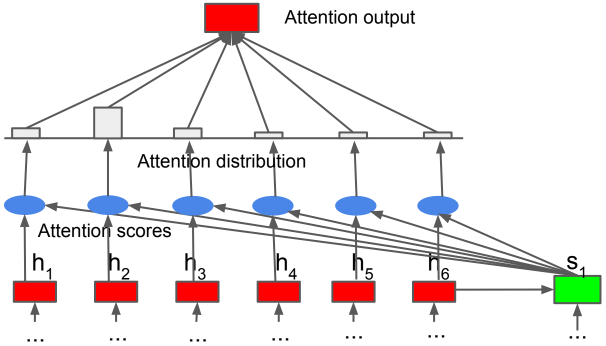
Attention: in equations

- 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)

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



Attention: in equations

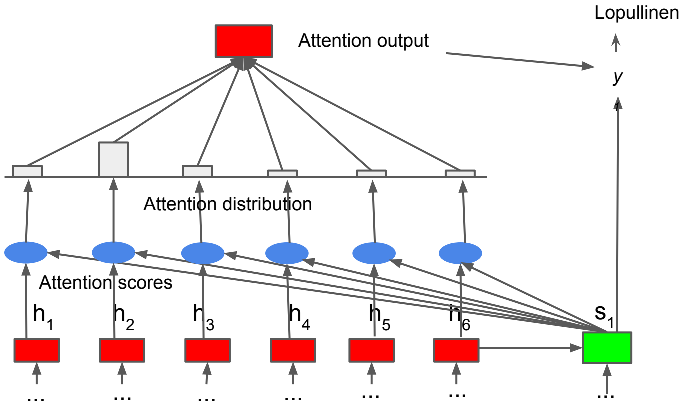
- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the *attention scores* $\boldsymbol{e}^t$ for this step:

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

- 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* $\boldsymbol{a}_t$

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

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

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

Attention is great

- Attention significantly improves NMT performance
    - It’s very useful to allow decoder to focus on certain parts of the source

Attention is great

- Attention significantly improves NMT performance
  - It’s very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck

Attention is great

- Attention significantly improves NMT performance
  - It’s very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention provides some interpretability:
  - By inspecting attention distribution, we can see what the decoder was focusing on

Attention is a *general* Deep Learning technique

- We’ve seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- <u>However</u>: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

Attention is a *general* Deep Learning technique

- We’ve seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and

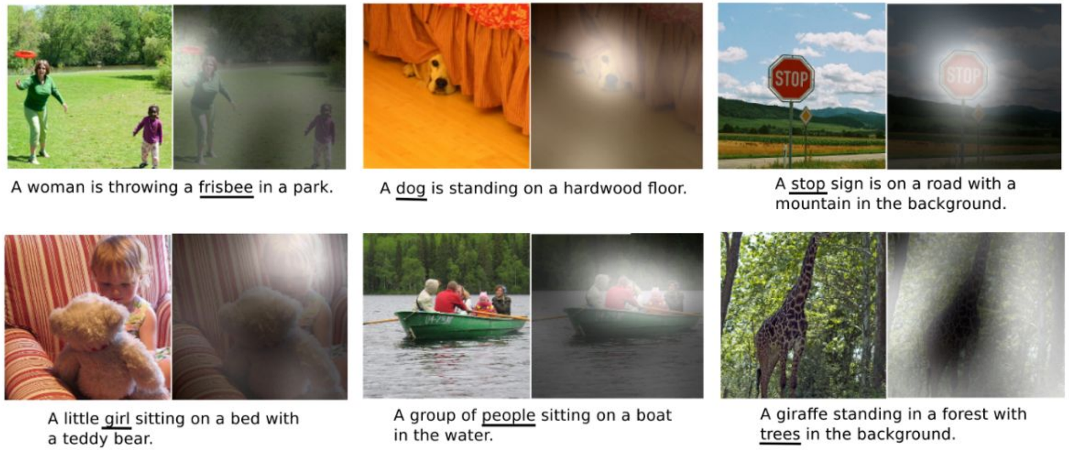


Fig. 5. Examples of the attention-based model attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word) [22]

[Xu et al., 2015]

Attention is a *general* Deep Learning technique

- We’ve seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- <u>However</u>: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.

[Xu et al., 2015]

Attention is a *general* Deep Learning technique

- We’ve seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- More general definition of attention:

  Given a set of vector values, and a vector query, *attention* is a technique to compute a weighted sum of the values, dependent on the query.

Attention is a *general* Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- More general definition of attention:

  Given a set of vector values, and a vector query, ***attention*** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends* to the values.
- For example, in the seq2seq + attention model, each decoder hidden state (**query**) *attends* to all the encoder hidden states (**values**).

Attention is a *general* Deep Learning technique

- More general definition of attention:

  Given a set of vector values, and a vector query, ***attention*** is a technique to compute a weighted sum of the values, dependent on the query.

- **Intuition**:
  - The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
  - Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), dependent on some other representation (the query).

There are *several* attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

There are *several* attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$
  2. Taking softmax to get *attention distribution* α:

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

  3. Using attention distribution to take weighted sum of values:

     $$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

     thus obtaining the *attention output* $\boldsymbol{a}$ (sometimes called the context vector)

There are *several* attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ ← There are multiple ways to do this

  2. Taking softmax to get *attention distribution* α:

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

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

thus obtaining the *attention output* $\boldsymbol{a}$ (sometimes called the context vector)

Attention variants

- There are several ways you can compute the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ from the values $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

Attention variants

- There are several ways you can compute the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ from the values $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$
- Basic dot-product attention:
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$

Attention variants

- There are several ways you can compute the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ from the values $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$
- <u>Basic dot-product attention</u>:
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$

- <u>Multiplicative attention:</u> $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix

Attention variants

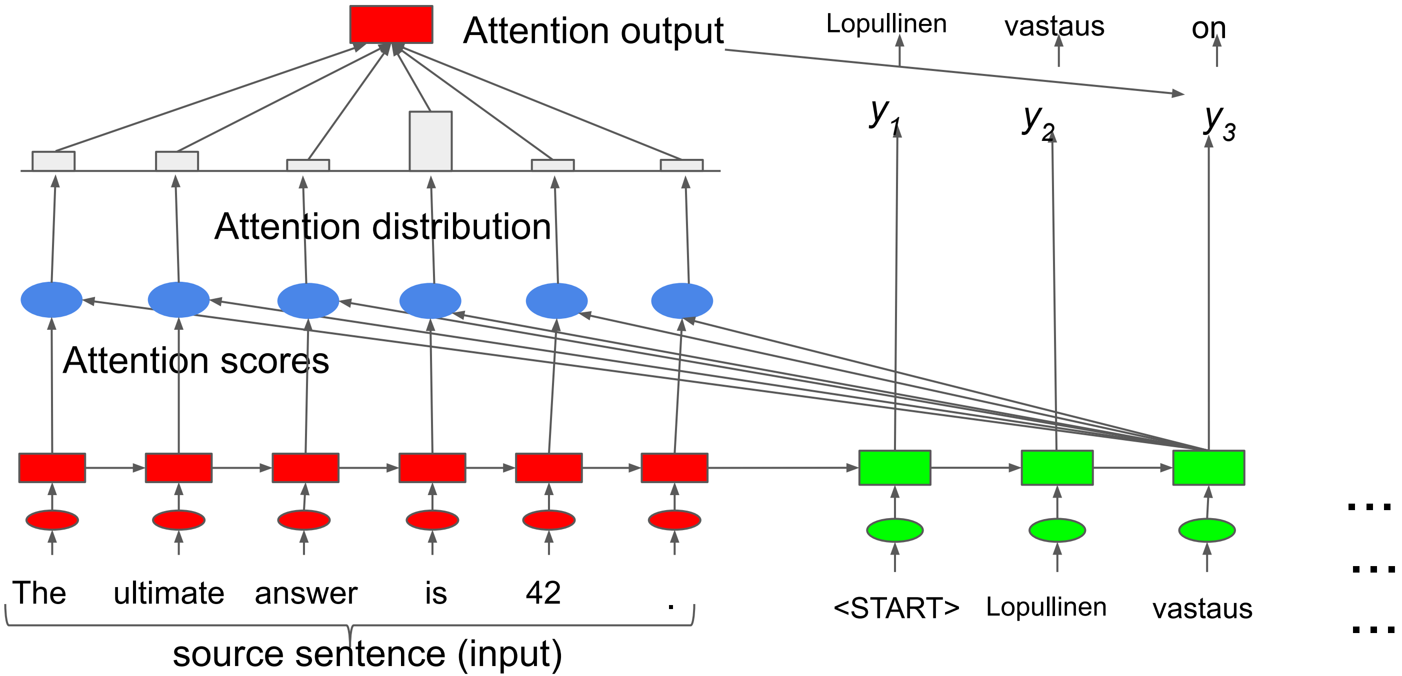
- There are several ways you can compute the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ from the values $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$
- Basic dot-product attention:
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
- 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

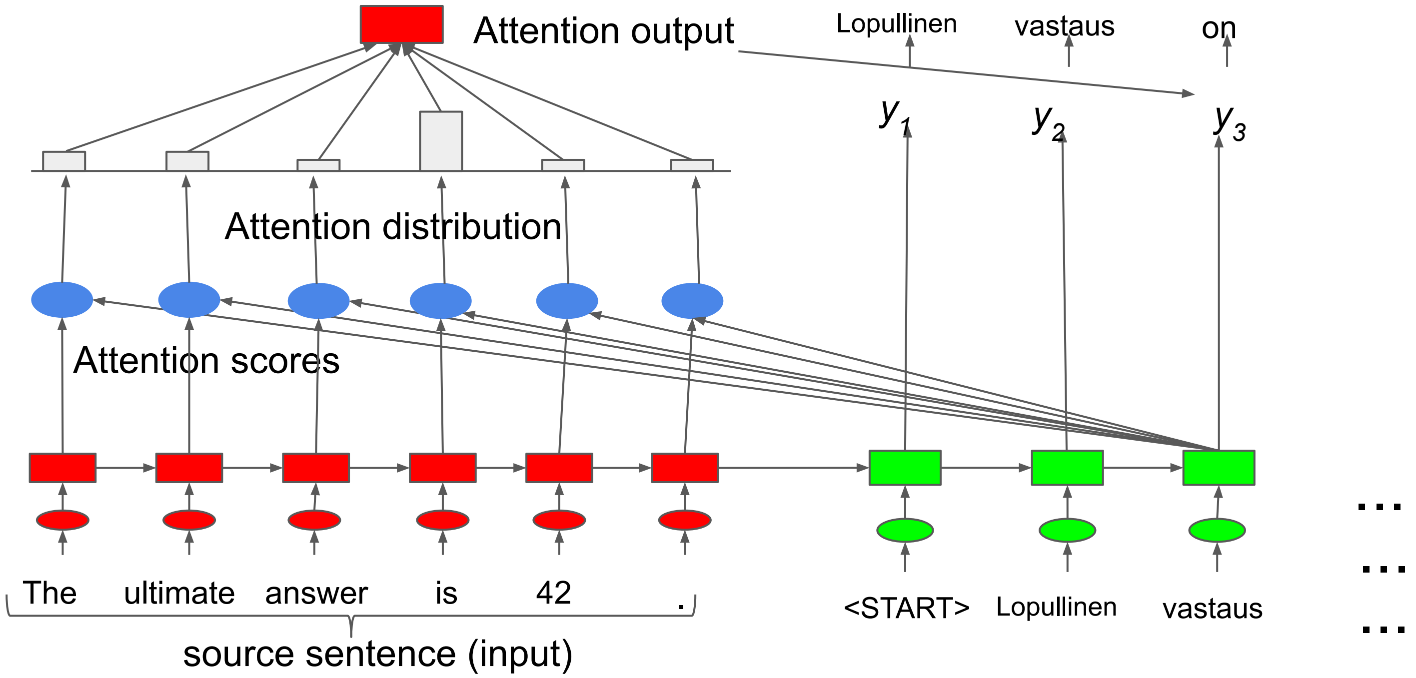
“Massive Exploration of Neural Machine Translation Architectures”, Britz et al, 2017, https://arxiv.org/pdf/1703.03906.pdf

Sequence-to-sequence with attention



- <u>Are we forgetting something behind?</u>

Sequence-to-sequence with attention

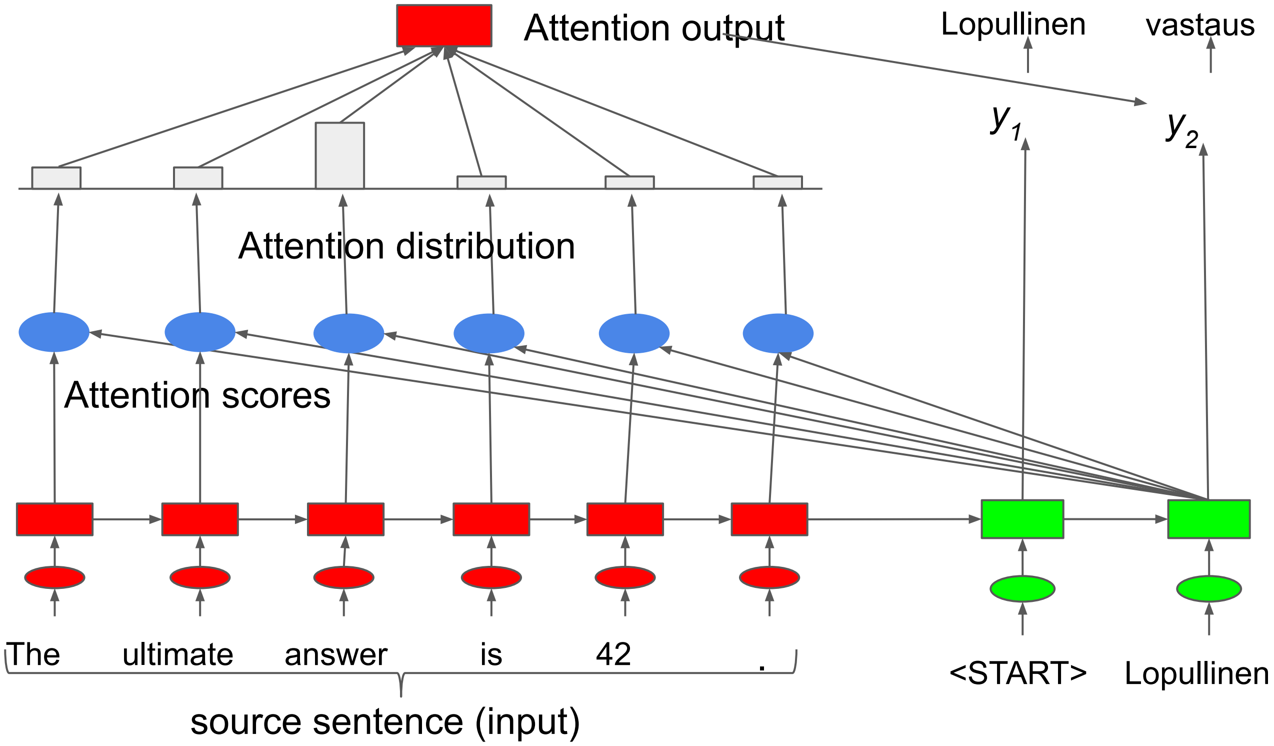


- <u>Are we forgetting something behind?</u>

  Attention decisions are made independently (which is *suboptimal*)

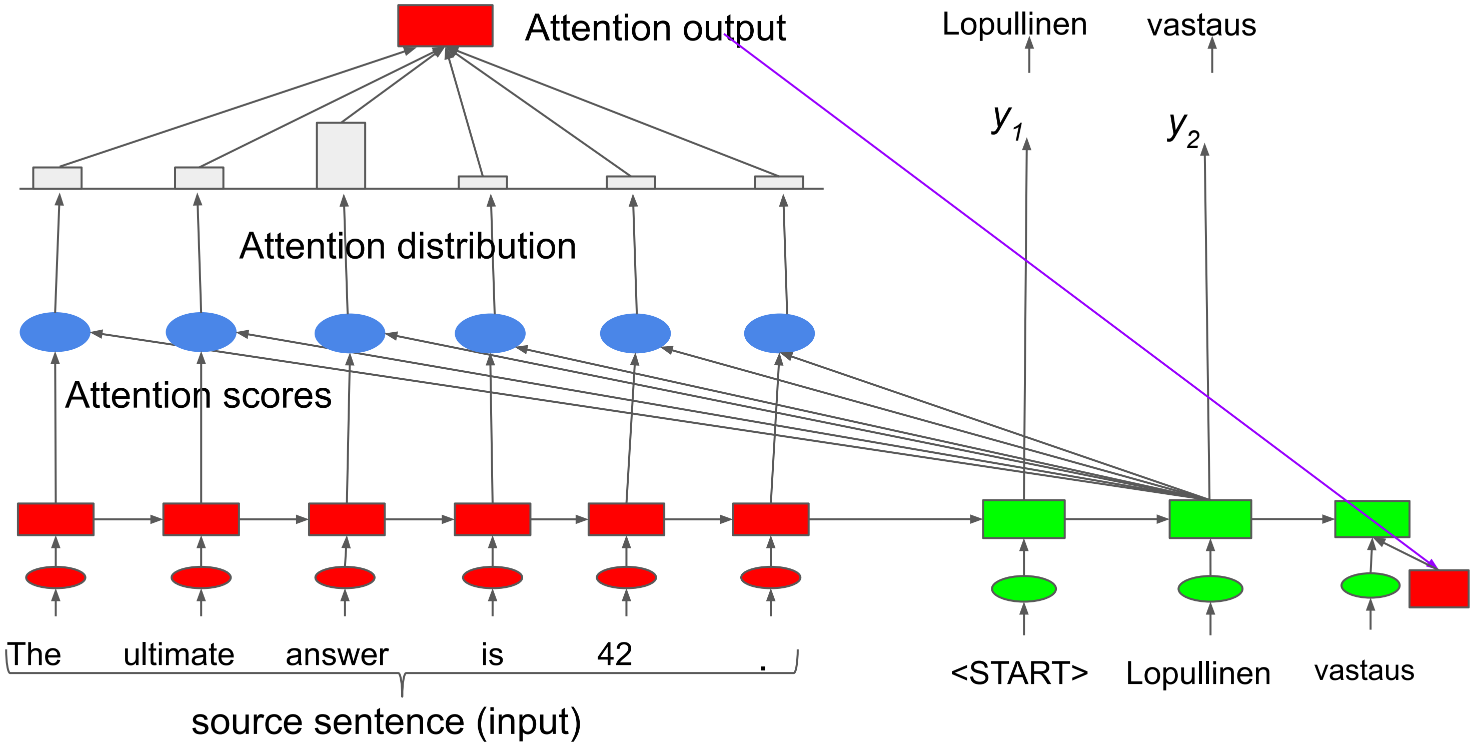
Attention feeding

- Attention decisions should be made jointly taking into account past attention information.



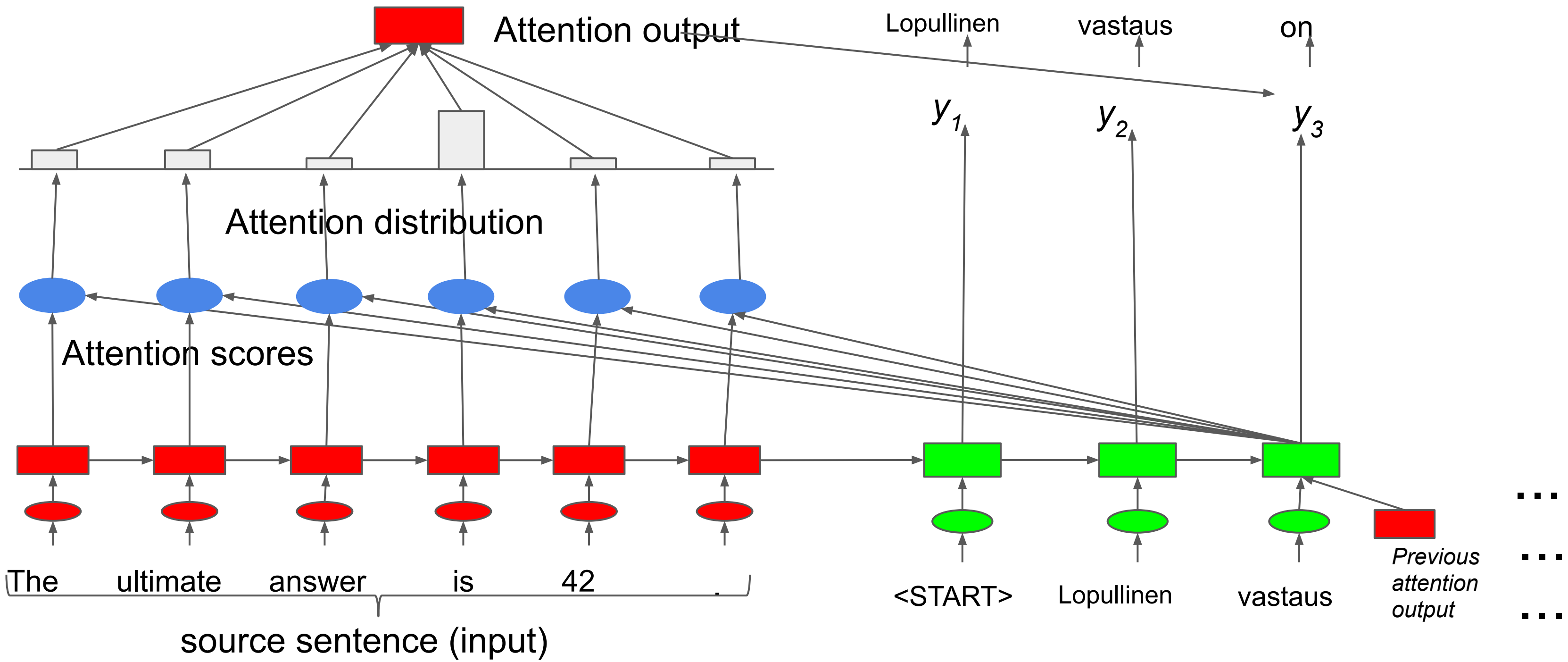
Attention feeding

- Attention decisions should be made jointly taking into account past attention information.



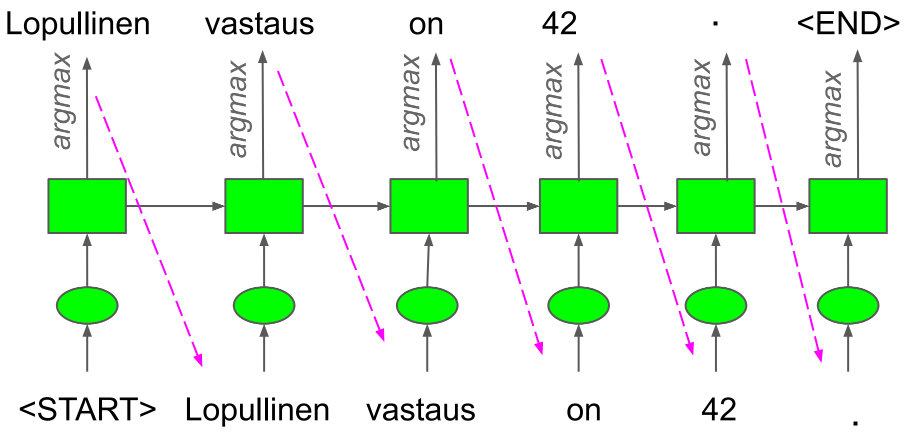
Attention feeding

- Attention decisions should be made jointly taking into account past attention information:
  - we hope to make the model fully aware of previous attention choices
  - usually the attentional vector is concatenated with the input at the next time step



Greedy decoding

- We saw how to generate (or “decode”) the target sentence by taking argmax on each step of the decoder



Problems with greedy decoding

- Greedy decoding has no way to undo decisions!
  - ➢ Input: The ultimate answer is 42 .   (Lopullinen vastaus on 42 .)

    ⇨

    ⇨ Lopullinen _

    Lopullinen lähtölaskenta _   (no going back now...)

- How to fix this?

Exhaustive search decoding

- Ideally we want to find a (length *T*) translation *y* that maximizes

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

- We could try computing all possible sequences *y*
  - This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
  - This $O(V^T)$ complexity is far too expensive!

Beam search decoding

- <u>Core idea</u>: 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!

Beam search decoding: example

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

<START>

Calculate prob dist of next word

Beam search decoding: example

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

$-0.7 = \log P_{LM}(\text{Lopullinen}|\text{<START>})$

Lopullinen

<START>

Finaali

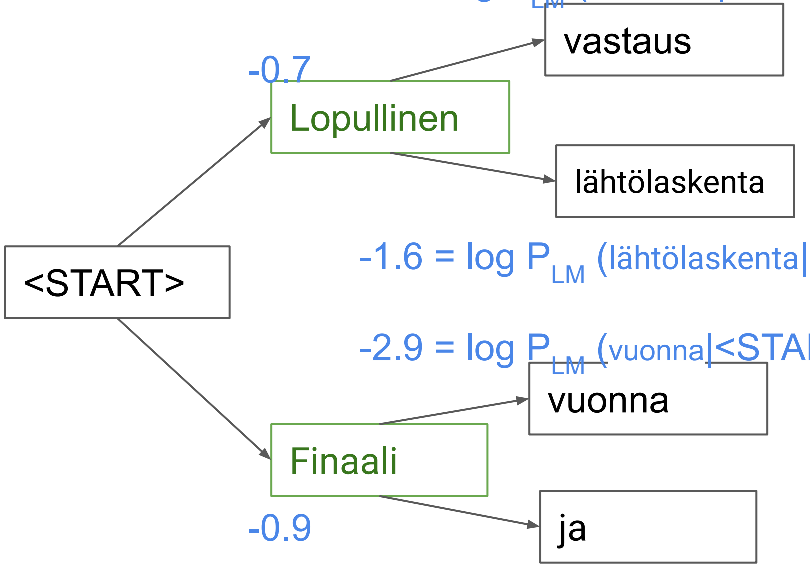
$-0.9 = \log P_{LM}(\text{Finaali}|\text{<START>})$

Take top *k* words and compute scores

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 = log P_{LM} (vastaus|<START> Lopullinen) + -0.7



-1.6 = log P_{LM} (lähtölaskenta|<START> Lopullinen) + -0.7

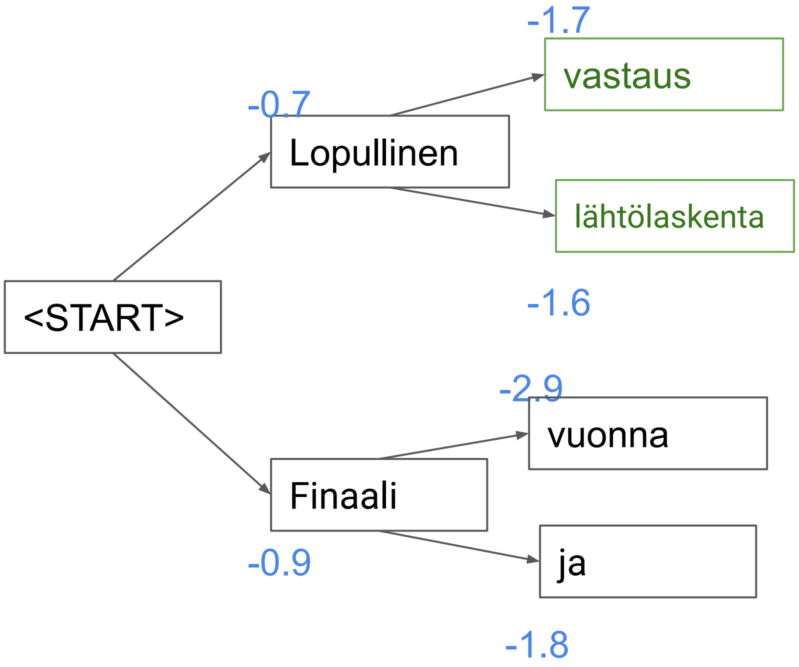
-2.9 = log P_{LM} (vuonna|<START> Finaali) + -0.9

-1.8 = log P_{LM} (ja|<START> Finaali) + -0.9

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

Beam search decoding: example

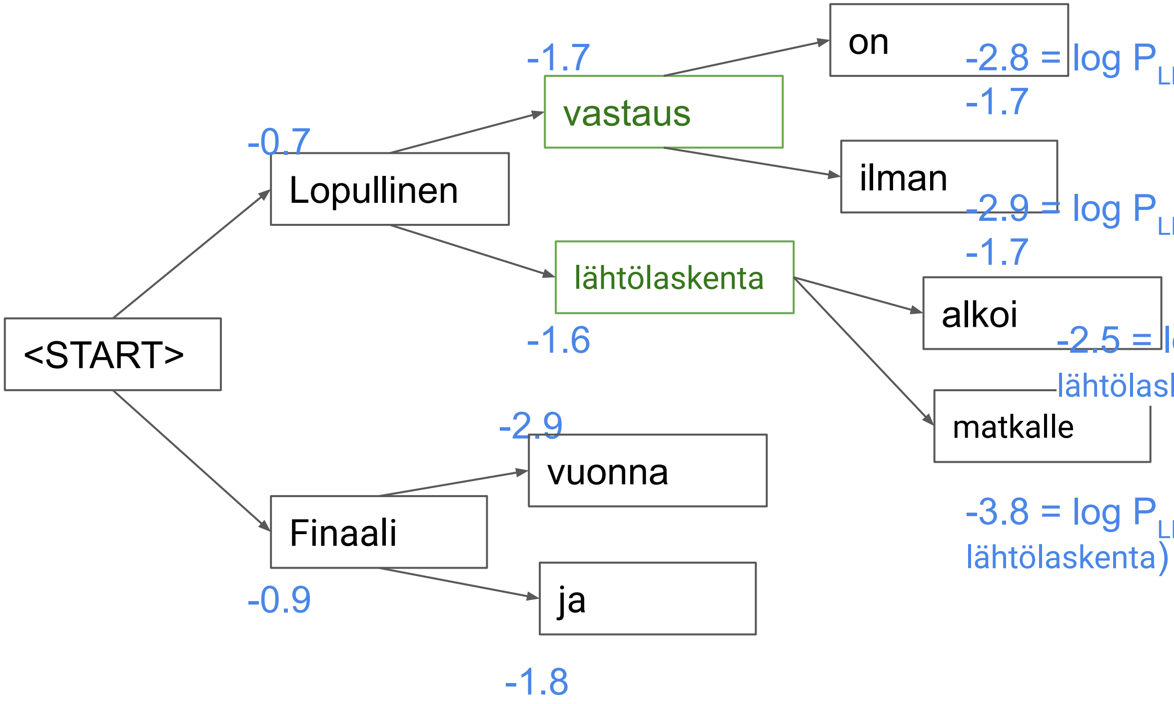
- Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

Beam search decoding: example

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



-2.8 = log P_{LM} (on|<START> Lopullinen vastaus) + -1.7

-2.9 = log P_{LM} (ilman|<START> Lopullinen vastaus) + -1.7

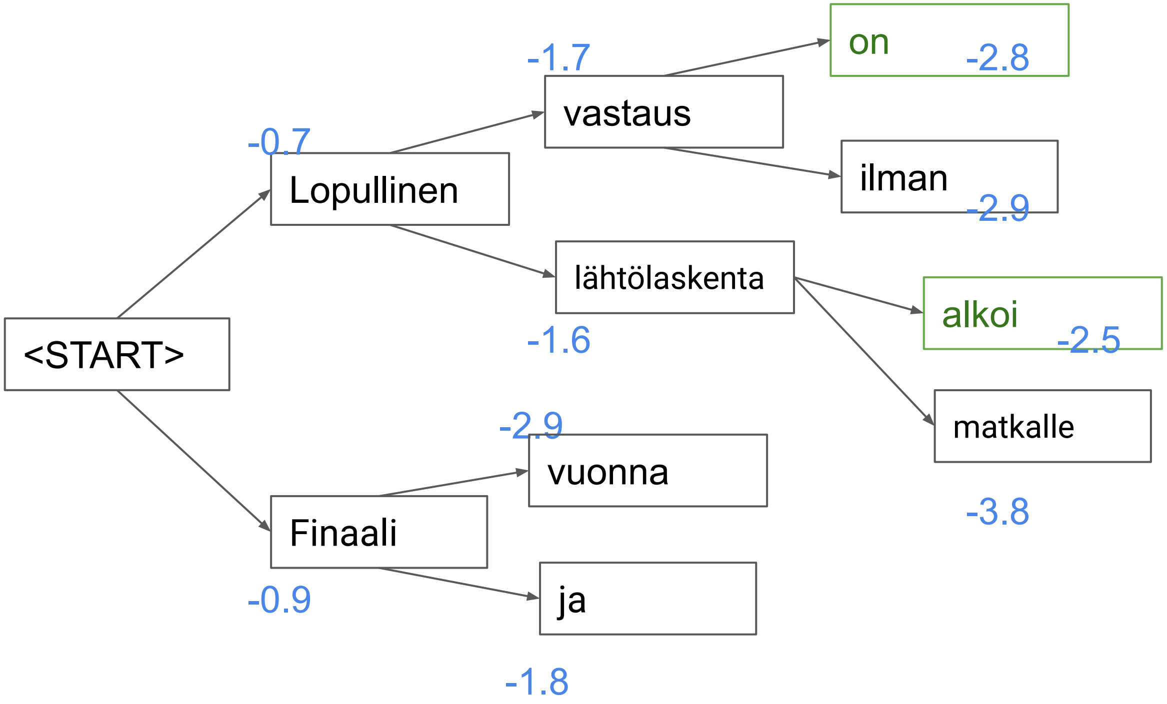
-2.5 = log P_{LM} (alkoi|<START> Lopullinen lähtölaskenta) + -1.6

-3.8 = log P_{LM} (matkalle|<START> Lopullinen lähtölaskenta) + -1.6

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

Beam search decoding: example

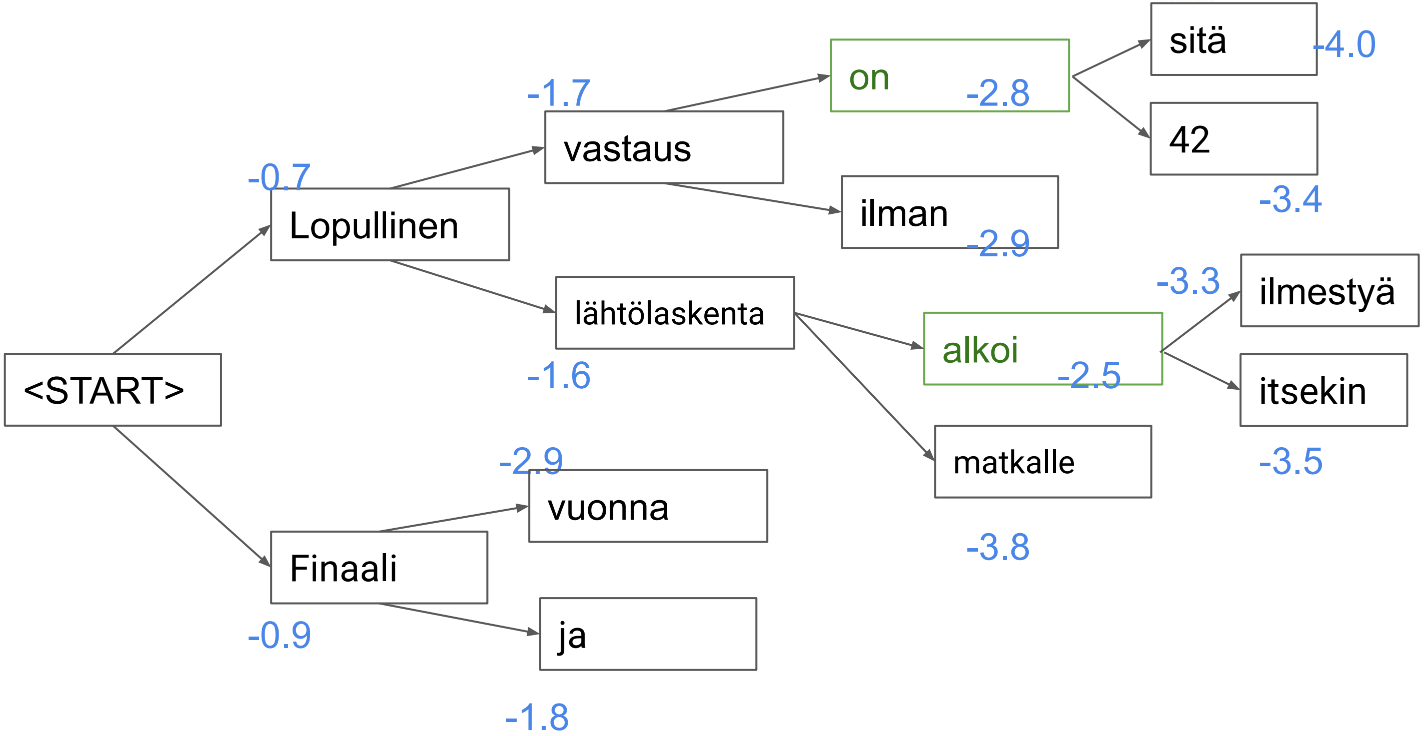
- Beam size = k = 2. Blue numbers = $\text{score}(y_1,\ldots,y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i|y_1,\ldots,y_{i-1},x)$



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

Beam search decoding: example

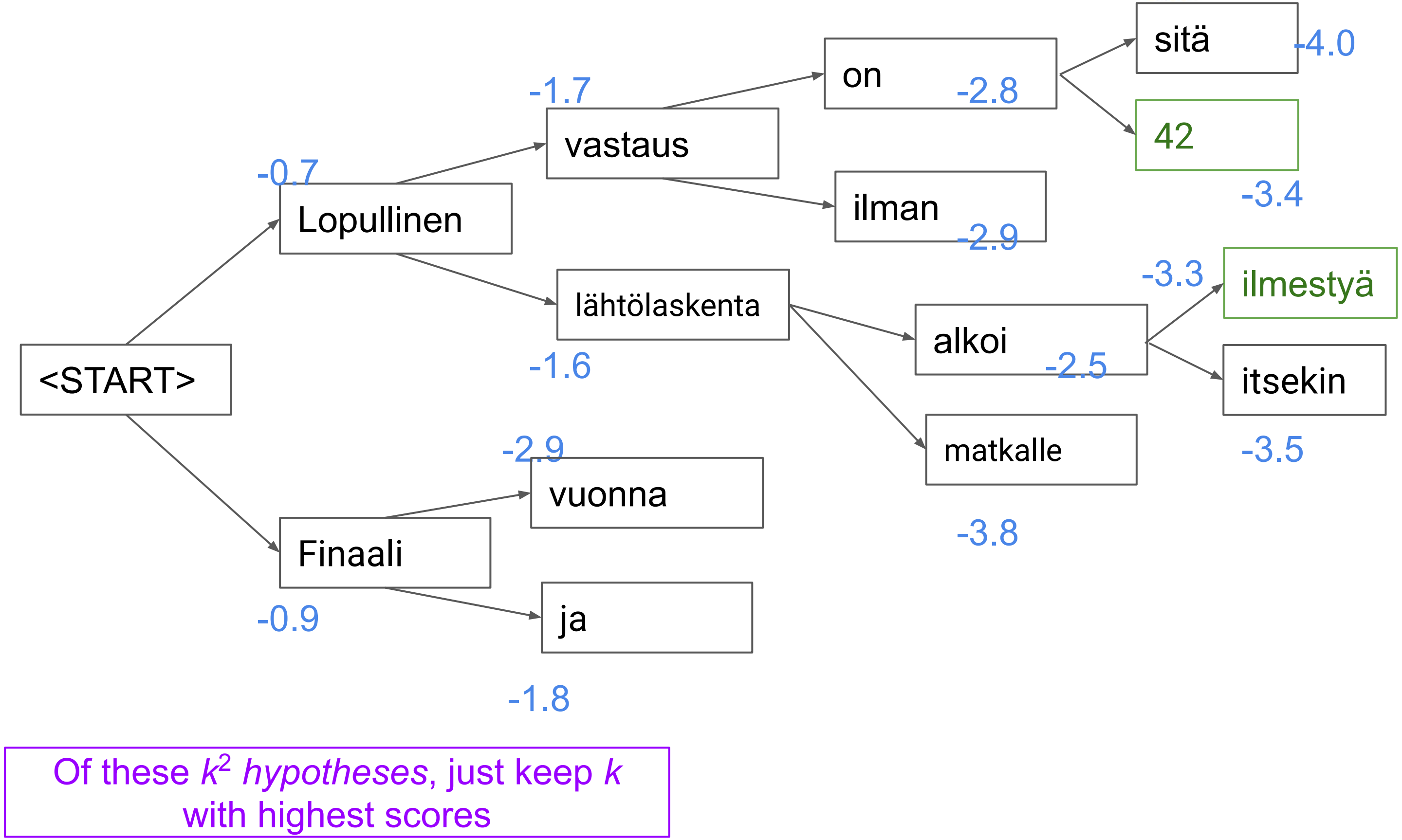
- Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



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

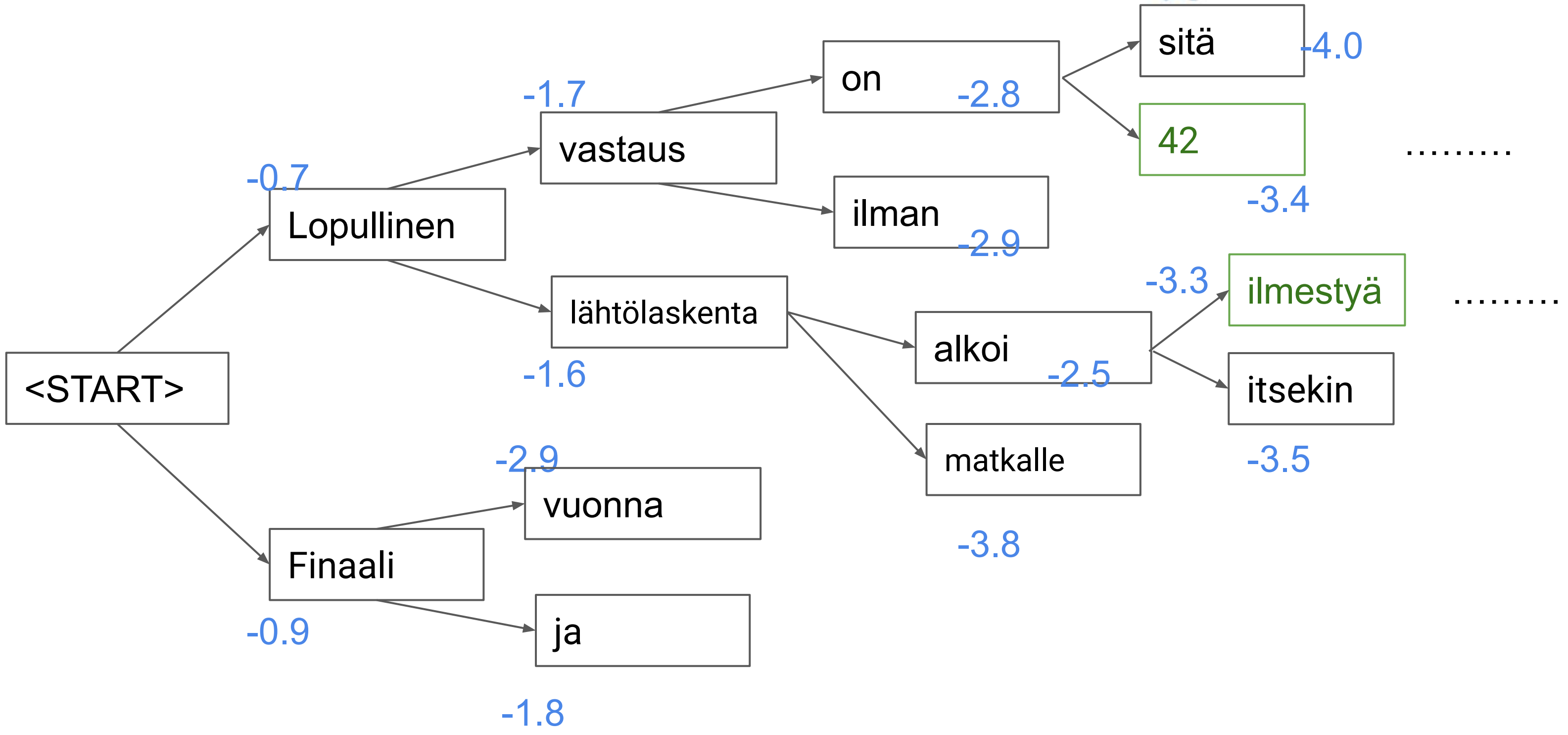
Beam search decoding: example

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



Beam search decoding: example

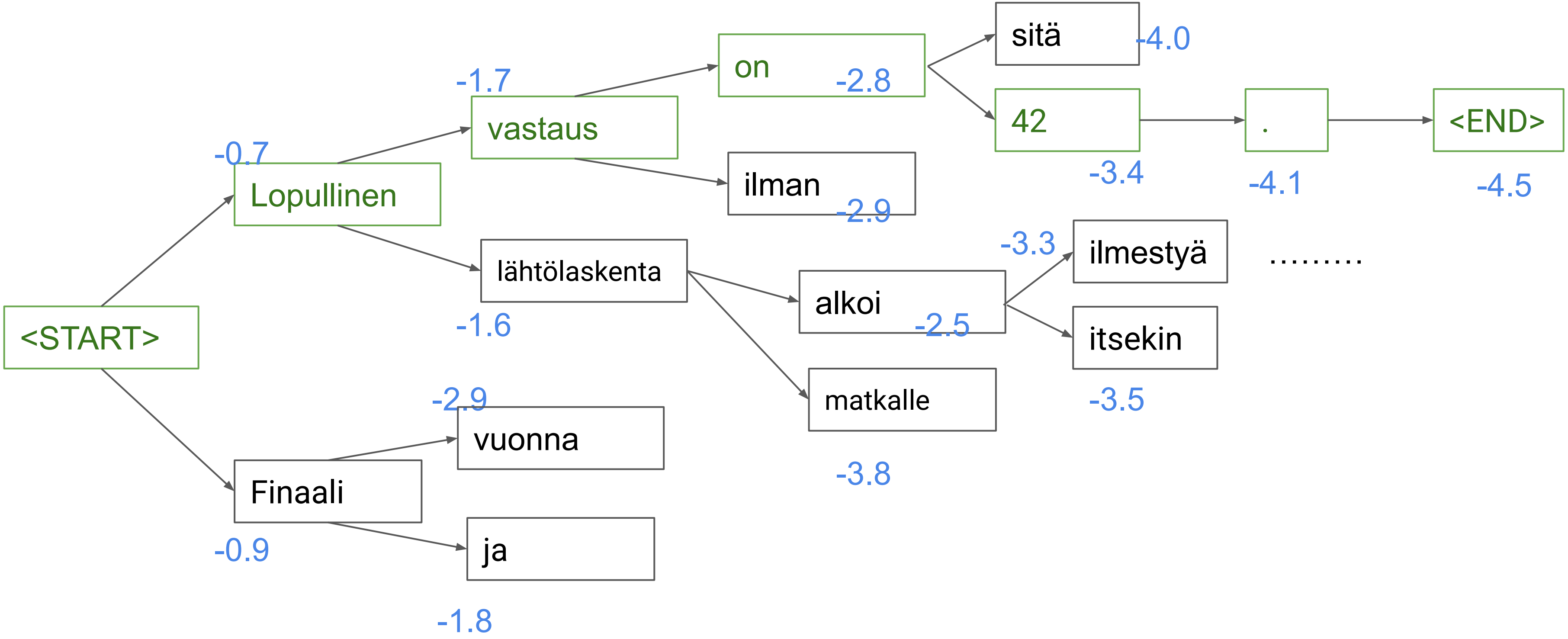
- Beam size = k = 2. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$



Continue until the <END> token

Beam search decoding: example

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



This is the top-scoring hypothesis!
Backtrack to obtain the full hypothesis

Beam search decoding: stopping criterion

- In greedy decoding, usually we decode until the model produces a <END> token
    - For example: <START> Lopullinen vastaus on 42 . <END>
- In beam search decoding, different hypotheses may produce <END> tokens on different timesteps
    - When a hypothesis produces <END>, that hypothesis is complete.
    - Place it aside and continue exploring other hypotheses via beam search.
- Usually we continue beam search until:
    - We reach timestep *T* (where *T* is some pre-defined cutoff), or
    - We have at least *n* completed hypotheses (where *n* is pre-defined cutoff)

Beam search decoding: finishing up

- We have our list of completed hypotheses.
- How to select top one with highest score?

- Each hypothesis $y_1,\ldots,y_t$ on our list has a score

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

- Problem with this: longer hypotheses have lower scores

- Fix: Normalize by length. Use this to select top one instead:

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

Advantages of NMT

NMT has many qualities:

- Fluent translations
  - Better use of context
- A single neural network to be optimized end-to-end
  - No subcomponents to be individually optimized
- Requires much less human engineering effort
  - No feature engineering
  - Same method for all language pairs

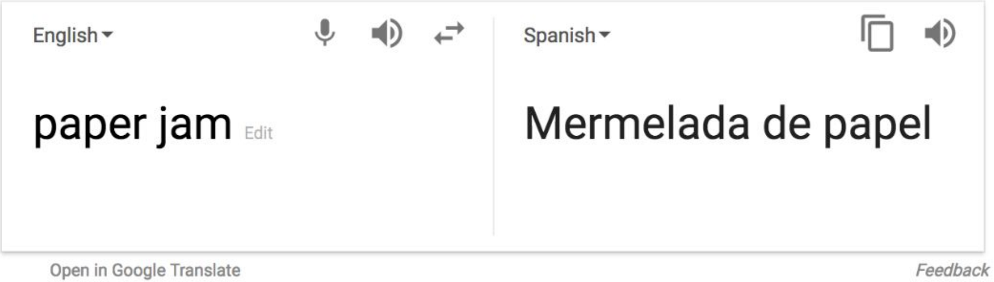
Disadvantages of NMT

- NMT is difficult to interpret and control
  - Hard to debug
  - can’t easily specify rules or guidelines for translation

- Many difficulties remain:
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs
  - Using common sense is still hard
  - NMT picks up biases in training data

Disadvantages of NMT

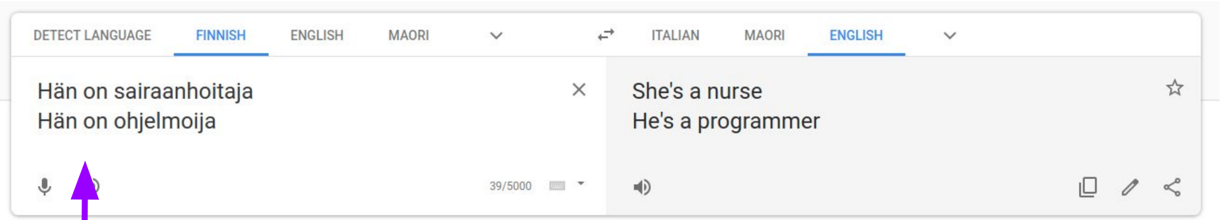
- Using common sense is still hard



?

Disadvantages of NMT

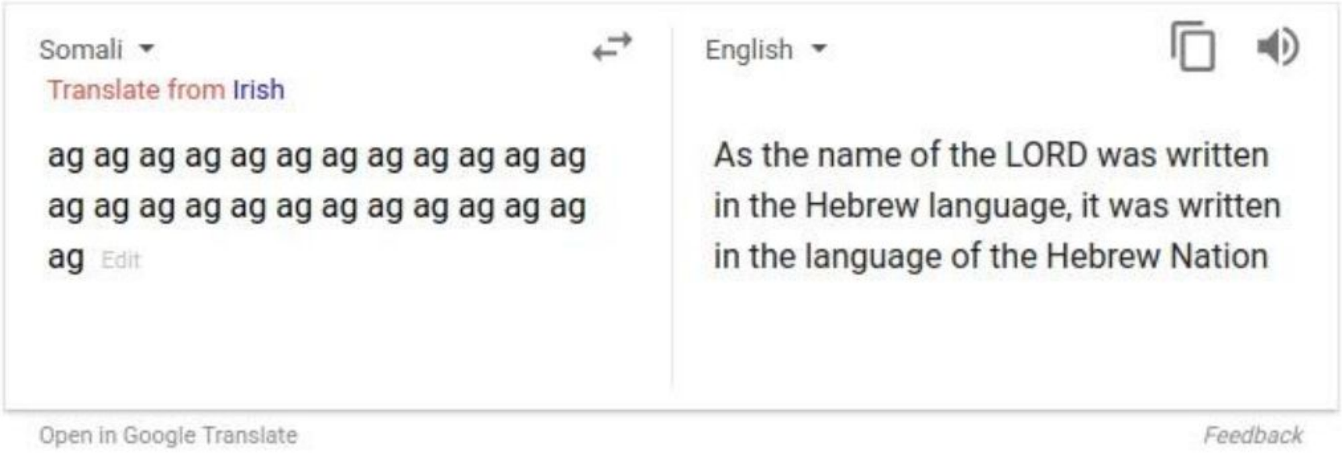
- NMT picks up biases in training data



Didn’t specify gender

Disadvantages of NMT

- Uninterpretable systems do strange things



Picture source: https://www.vice.com/en_uk/article/j5npeg/why-is-google-translate-spitting-out-sinister-religious-prophecies
Explanation: https://www.skynettoday.com/briefs/google-nmt-prophecies

Disadvantages of NMT

- Out-Of-Vocabulary (OOV) words
  words not in the vocabulary of the trained NMT model
  - networks have fixed vocabulary
  - poor translation of rare/unknown words

Open-Vocabulary Neural Machine Translation

- How do we represent text in NMT?
- 1-hot encoding:
  - Lookup of word embedding for input
  - Probability distribution over vocabulary for output

- Large vocabularies
  - Increase network size
  - Decrease training and decoding speed

- Typical network vocabulary size: 10 000 - 100 000 symbols

vocabulary		representation of "cat"	
		1-hot vector	embedding
$\begin{matrix} 0 \\ 1 \\ 2 \\ . \\ 1024 \end{matrix}$	$\begin{bmatrix} \text{the} \\ \text{cat} \\ \text{is} \\ . \\ \text{mat} \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \\ 0 \\ . \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0.1 \\ 0.3 \\ 0.7 \\ 0.5 \end{bmatrix}$

Open-Vocabulary Neural Machine Translation

- Translation is an **open-vocabulary** problem:
  - many training corpora contain millions of word types
  - productive word formation processes (compounding; derivation) allow formation and understanding of unseen words
  - names, numbers are morphologically simple, but open word classes

Open-Vocabulary Neural Machine Translation

- Translation is an **open-vocabulary** problem:
  - many training corpora contain millions of word types
  - productive word formation processes (compounding; derivation) allow formation and understanding of unseen words
  - names, numbers are morphologically simple, but open word classes

- Solution 1 - **Back-off Models**:
  1. replace rare words with UNK at training time
  2. when system produces UNK, translate with a back-off method, for example a dictionary

- What are the limitations with this method?

Open-Vocabulary Neural Machine Translation

- Solution 1 - **Back-off Models**:
  1. replace rare words with UNK at training time
  2. when system produces UNK, translate with a back-off method, for example a dictionary

- What are the limitations with this method?
  - **compounds**: hard to model 1-to-many relationships
  - **morphology**: hard to predict inflection with back-off dictionary
  - **names**: if alphabets differ, we need transliteration

- Can we do better?

Open-Vocabulary Neural Machine Translation

- Solution 2 - *wishlist*:
    1. open-vocabulary NMT: encode all words through small vocabulary
    2. encoding generalizes to unseen words
    3. small text size
    4. good translation quality

Open-Vocabulary Neural Machine Translation

- Solution 2 - *wishlist*:
  1. open-vocabulary NMT: encode all words through small vocabulary
  2. encoding generalizes to unseen words
  3. small text size
  4. good translation quality

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Start with a vocabulary of **characters**.
  - Most frequent ngram pairs ↦ a new ngram.
  - *hyperparameter*: when to stop ↦ controls vocabulary size

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Start with a vocabulary of **characters**.
  - Most frequent ngram pairs ↦ a new ngram.
  - *hyperparameter*: when to stop ↦ controls vocabulary size

- Example:

Dictionary

l o w	5
l o w e r	2
n e w e s t	6
w i d e s t	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d

Start with all characters in dictionary

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Start with a vocabulary of **characters**.
  - Most frequent ngram pairs ↦ a new ngram.
  - *hyperparameter*: when to stop ↦ controls vocabulary size

- Example:

Dictionary

l o w	5
l o w e r	2
n e w **es** t	6
w i d **es** t	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, **es**

Add a pair (e, s) with freq 9

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Start with a vocabulary of **characters**.
  - Most frequent ngram pairs ↦ a new ngram.
  - *hyperparameter*: when to stop ↦ controls vocabulary size

- Example:

Dictionary

l o w	5
l o w e r	2
n e w **est**	6
w i d **est**	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, **est**

Add a pair (es, t) with freq 9

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Start with a vocabulary of **characters**.
  - Most frequent ngram pairs ↦ a new ngram.
  - *hyperparameter*: when to stop ↦ controls vocabulary size

- Example:

Dictionary

lo w	5
lo w e r	2
n e w est	6
w i d est	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, **lo**

Add a pair (l, o) with freq 7

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Automatically decide vocabs for NMT
  - Open-vocabulary: operations learned on training set can be applied to unknown words
  - compression of frequent character sequences improves efficiency
  - trade-off between text length and vocabulary size

Dictionary

lo w	5
lo w e r	2
n e w est	6
w i d est	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, lo

e s → es
es t → est
l o → lo

- Suppose we have the following new word:
  l o w e s t

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Automatically decide vocabs for NMT
  - Open-vocabulary: operations learned on training set can be applied to unknown words
  - compression of frequent character sequences improves efficiency
  - trade-off between text length and vocabulary size

Dictionary

lo w	5
lo w e r	2
n e w est	6
w i d est	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, lo

e s → es
es t → est
l o → lo

- Suppose we have the following new word:
  l o w **es** t

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Automatically decide vocabs for NMT
  - Open-vocabulary: operations learned on training set can be applied to unknown words
  - compression of frequent character sequences improves efficiency
  - trade-off between text length and vocabulary size

Dictionary

lo w	5
lo w e r	2
n e w est	6
w i d est	3

Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, lo

e s → es
es t → est
l o → lo

- Suppose we have the following new word:
  l o w **est**

Subword units - Byte Pair Encoding (BPE)

- **Subword units - Byte Pair Encoding (BPE)** for word segmentation**:**
  - Automatically decide vocabs for NMT
  - Open-vocabulary: operations learned on training set can be applied to unknown words
  - compression of frequent character sequences improves efficiency
  - trade-off between text length and vocabulary size

Dictionary

lo w	5
lo w e r	2
n e w est	6
w i d est	3

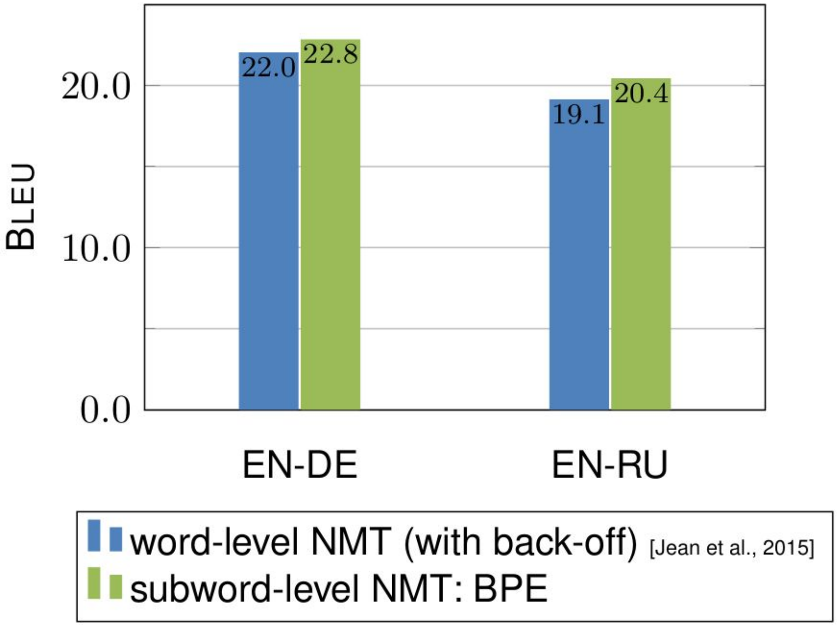
Vocabulary

l, o, w, e, r, n, w, s, t, i, d, es, est, lo

e s → es
es t → est
l o → lo

- Suppose we have the following new word:
  lo w est

Byte Pair Encoding (BPE)
Translation quality



Byte Pair Encoding (BPE) Examples

system	sentence
source	health research institutes
reference	Gesundheitsforschungsinstitute
word-level (with back-off)	Forschungsinstitute
BPE	Gesundheits|forsch|ungsin|stitute
source	rakfisk
reference	ракфиска (rakfiska)
word-level (with back-off)	rakfisk $\rightarrow$ UNK $\rightarrow$ rakfisk
BPE	rak|f|isk $\rightarrow$ рак|ф|иска (rak|f|iska)

Byte Pair Encoding (BPE)

- BPE-level subword segmentation is currently the most widely used technique for open-vocabulary NMT
- BPE allows open vocabulary
  - how well it generalizes is still an open question

- Segmentation Variants:
  - morphologically motivated subword units [Sánchez-Cartagena and Toral, 2016, Tamchyna et al., 2017, Huck et al., 2017, Pinnis et al., 2017]
  - probabilistic segmentation and sampling [Kudo, 2018]
  - fully character-level Models [Ling et al. 2015, Lee et al. 2016]

Multilingual neural MT

Transfer learning

- Knowledge transfer between languages and language pairs
- Make use of linguistic relationships of languages
- Practical reason: support low resource scenarios (languages and domains)

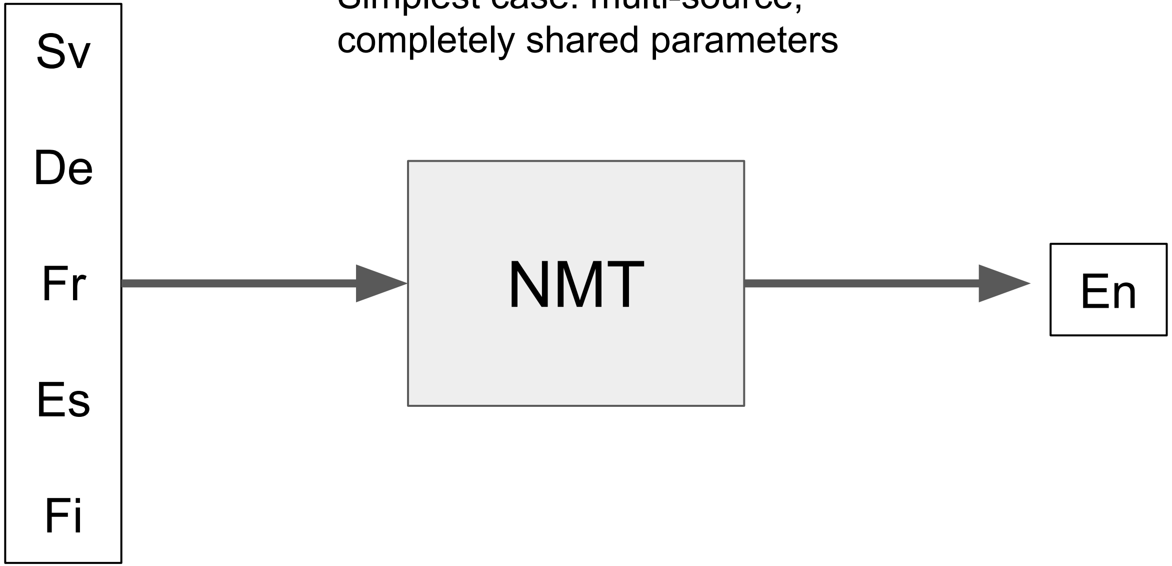
Zero-shot translation

- Translate between languages without explicit training examples
- Unseen task trained through multi-task learning

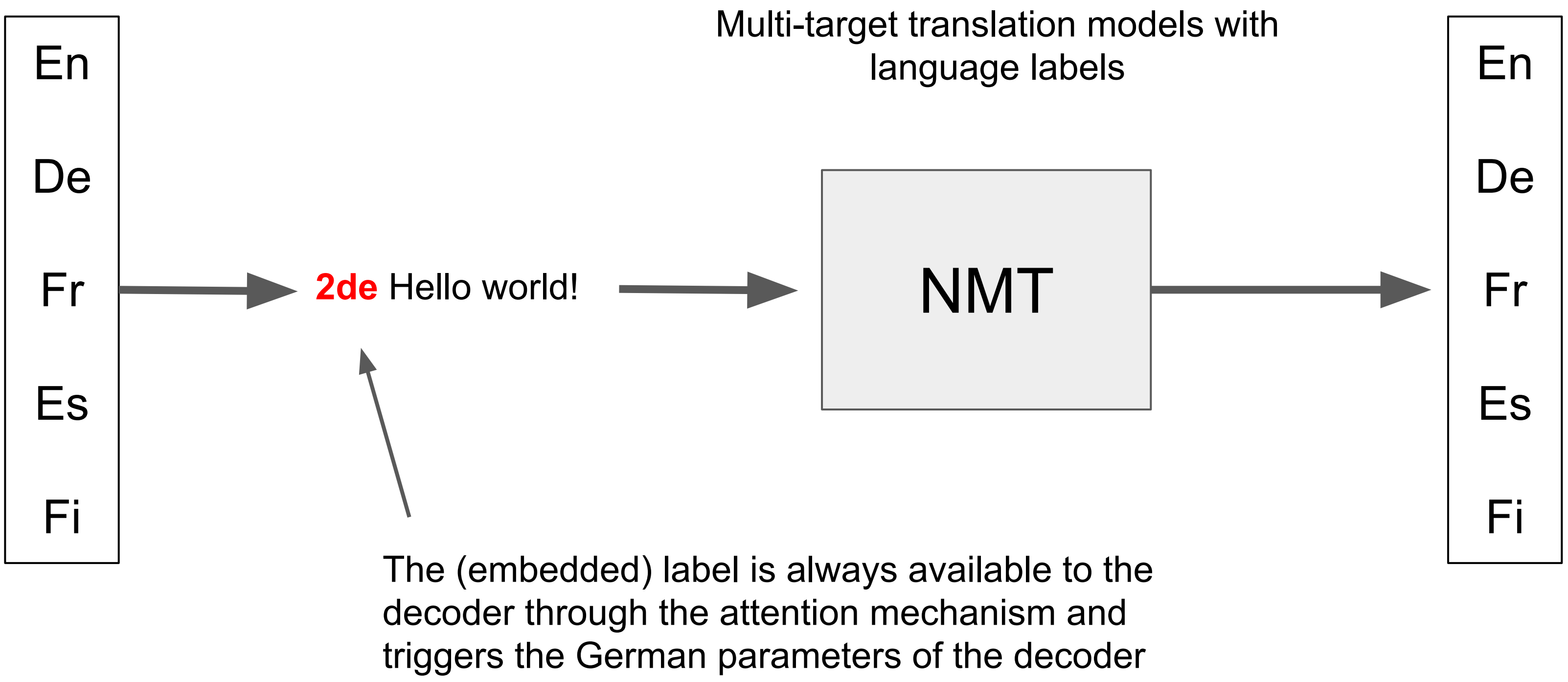
Approaches differ in the amount of parameter sharing

Language labels and completely shared parameters

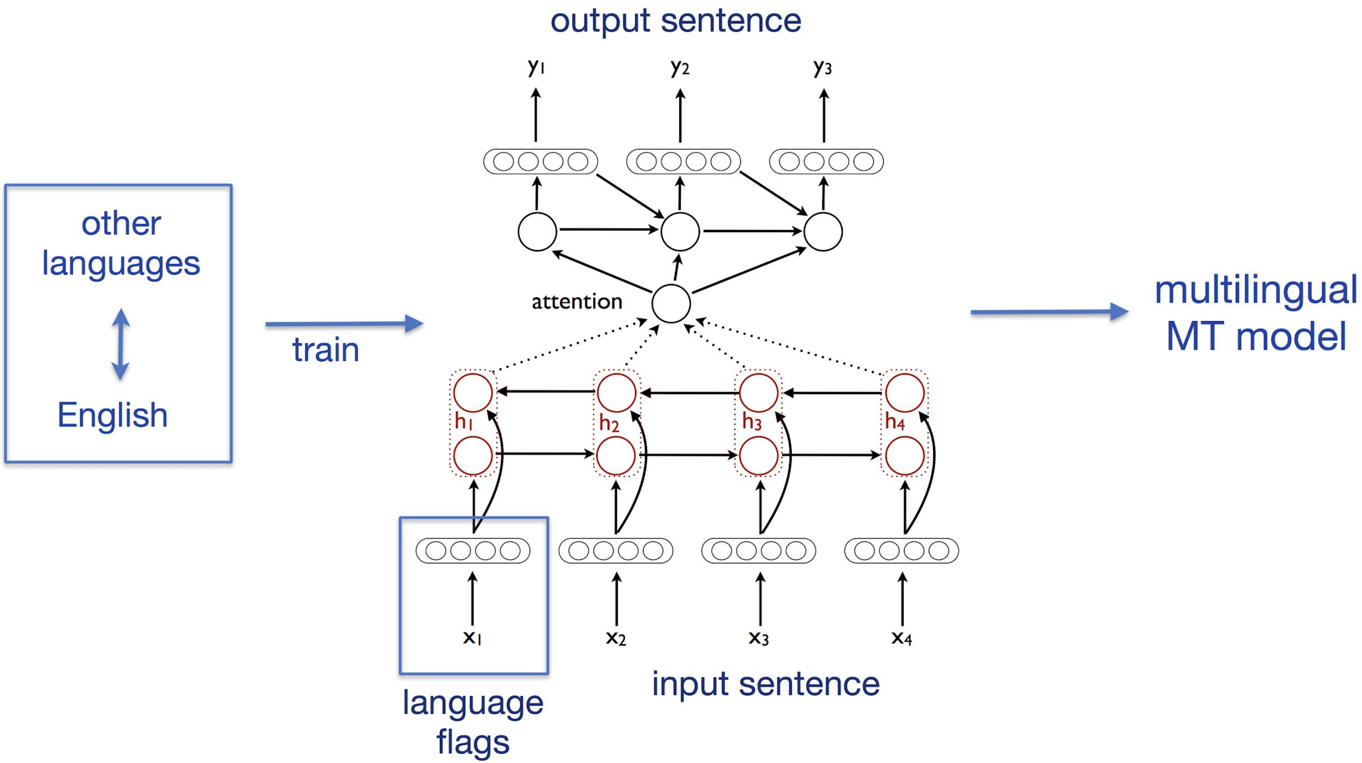
Simplest case: multi-source,
completely shared parameters



Language labels and completely shared parameters

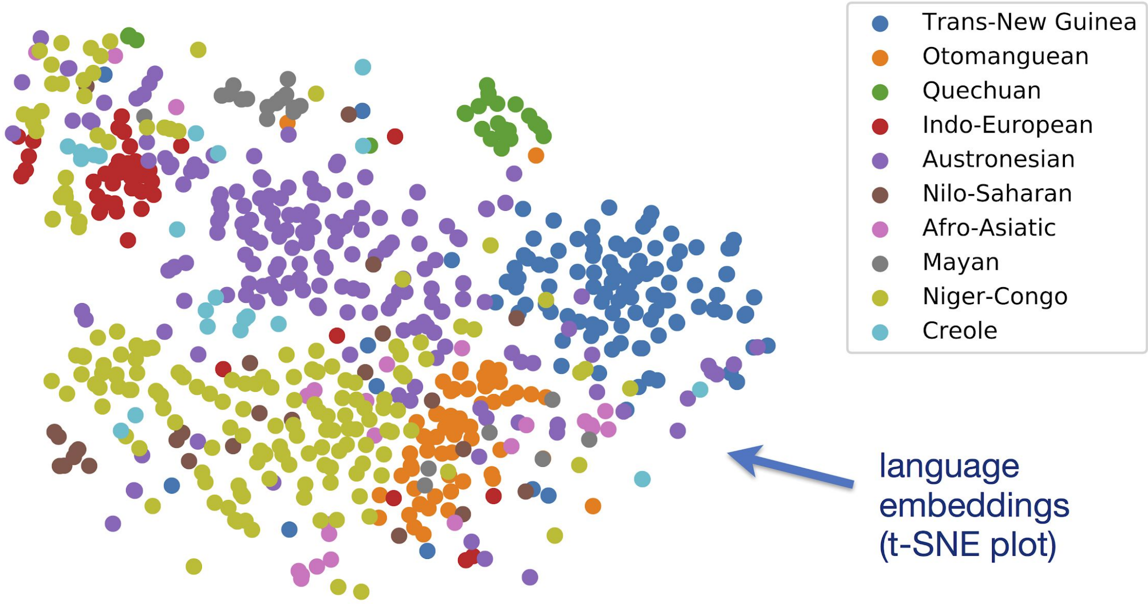


Emerging language spaces



Emerging language spaces

Rough clusters of language families



language embeddings (t-SNE plot)

Emerging Language Spaces Learned From Massively Multilingual Corpora (https://arxiv.org/abs/1802.00273)

Scaling up to many languages

… fails!

BLEU	aln	bar	bre	chu	eng	fao	fry	glv	hns	isl	...
aln		2.95	4.75	3.77	17.38	8.63	5.38	4.27	4.00	5.52	
bar	3.73		4.19	2.90	13.99	7.00	4.85	3.98	3.60	4.79	
bre	1.26	0.66		1.63	9.72	1.36	0.00	2.41	1.29	1.25	
chu	7.36	3.54	4.97		20.10	9.48	7.78	4.93	4.09	6.18	
eng	11.49	6.44	5.46	7.93		13.77	14.08	5.38	5.60	9.48	
fao					18.82						
fry					18.58						
glv					9.28						
hns					7.87						
isl					14.51						

zero-shot

supervised

Language labels and completely shared parameters

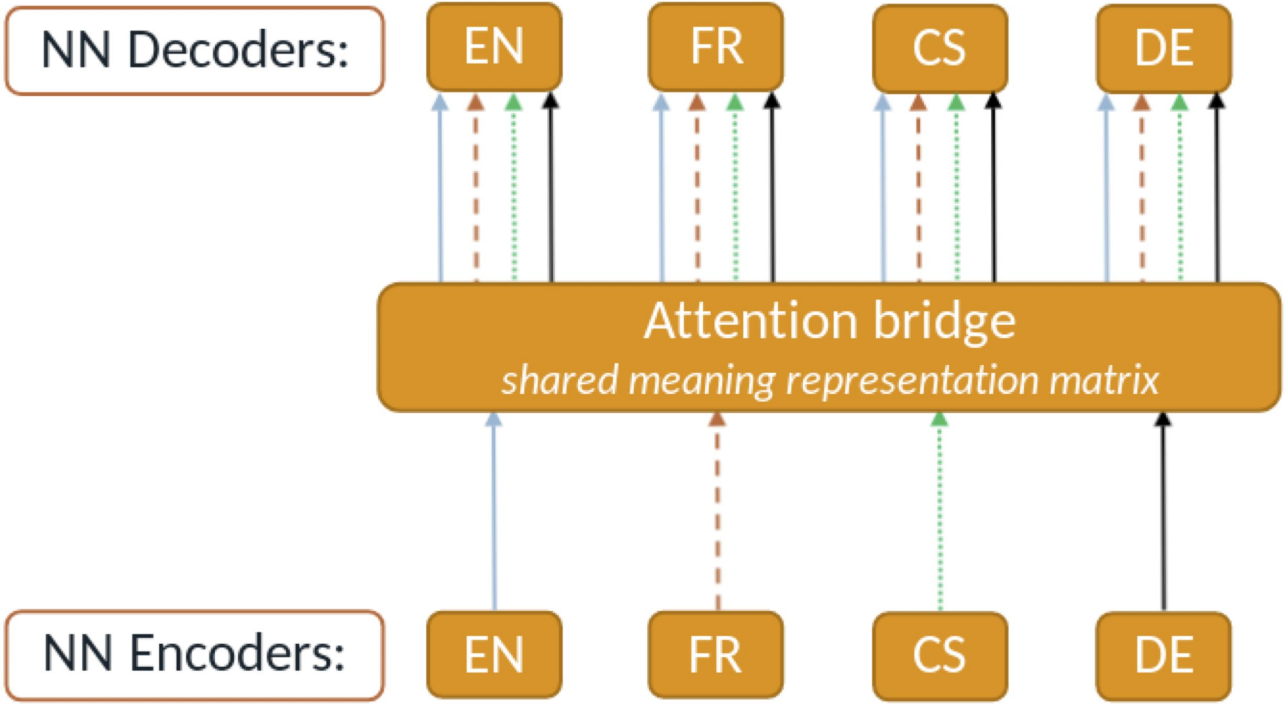
Transfer and zero-shot with language labels

- Very easy and surprisingly effective (especially for related languages)
- Improves low-resource scenarios
- Enables zero-shot translation (but only) if source and target language appear in different combinations with other languages during training
- Single model for many languages, mixed language support

Limits

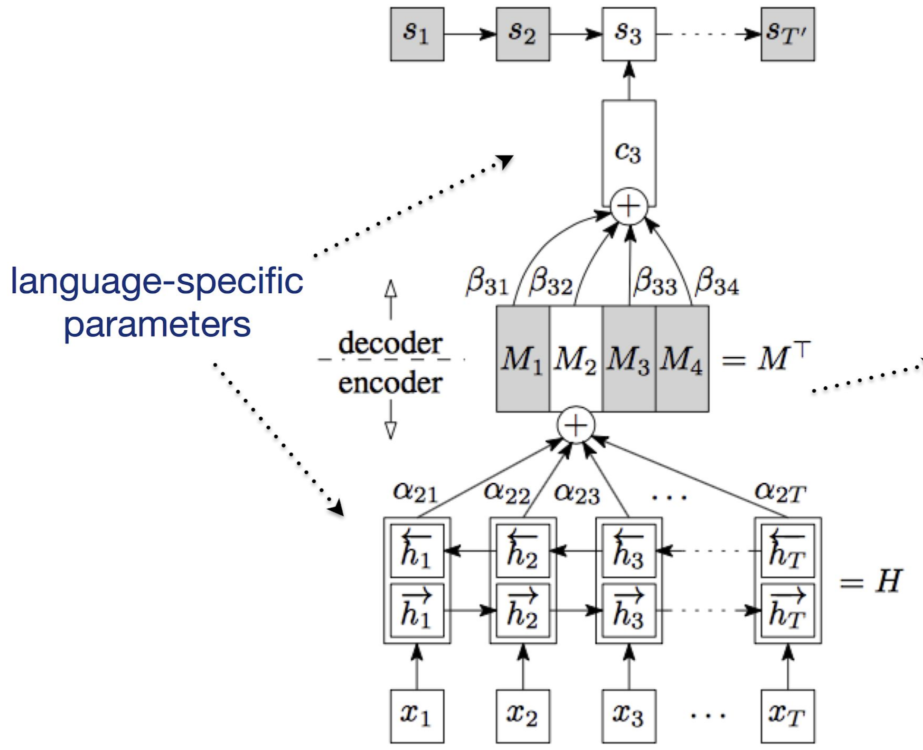
- Capacity bottleneck: doesn’t scale to many languages
- Typically no improvement for high-resource languages

Multilingual NMT with partially shared parameters



Multilingual NMT with a Language-Independent Attention Bridge, Raúl Vázquez, Alessandro Raganato, Jörg Tiedemann, Mathias Creutz (Rep4NLP 2019) https://github.com/Helsinki-NLP/OpenNMT-py/tree/att-brg

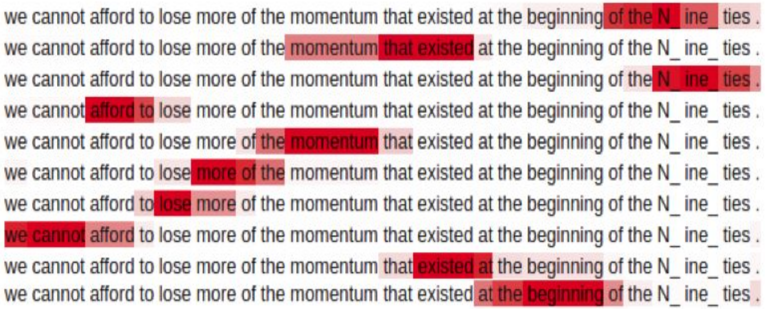
The attention bridge model



Architecture proposed by Cífka and Bojar (2018). *Our implementation in OpenNMT-py (MTM2018)*

Focus of attention

(a) k = 1



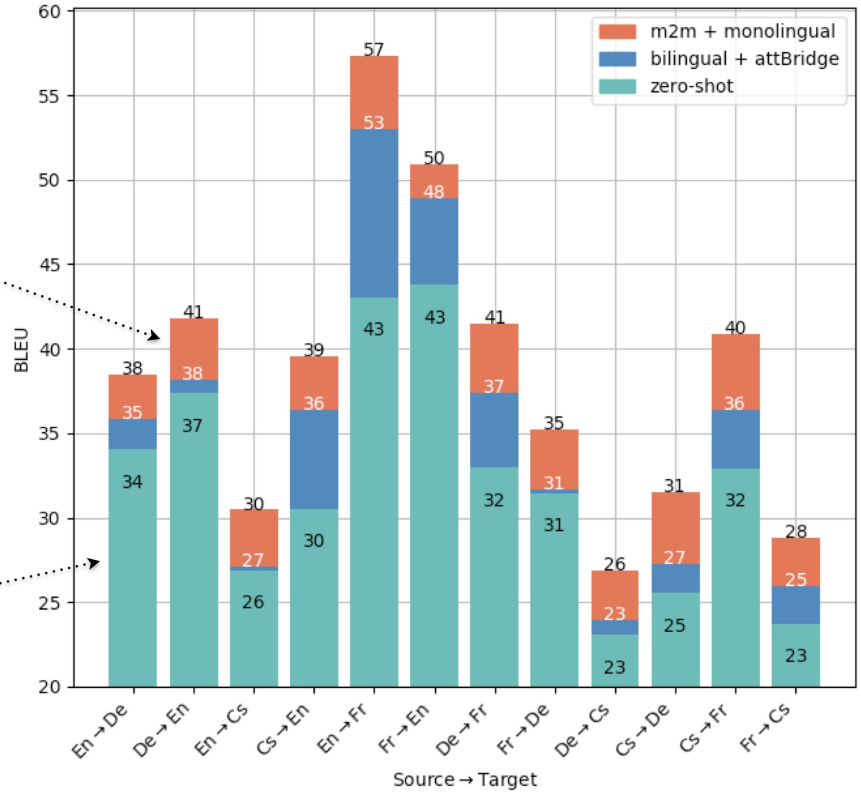
(b) k = 10

(d) k = 50

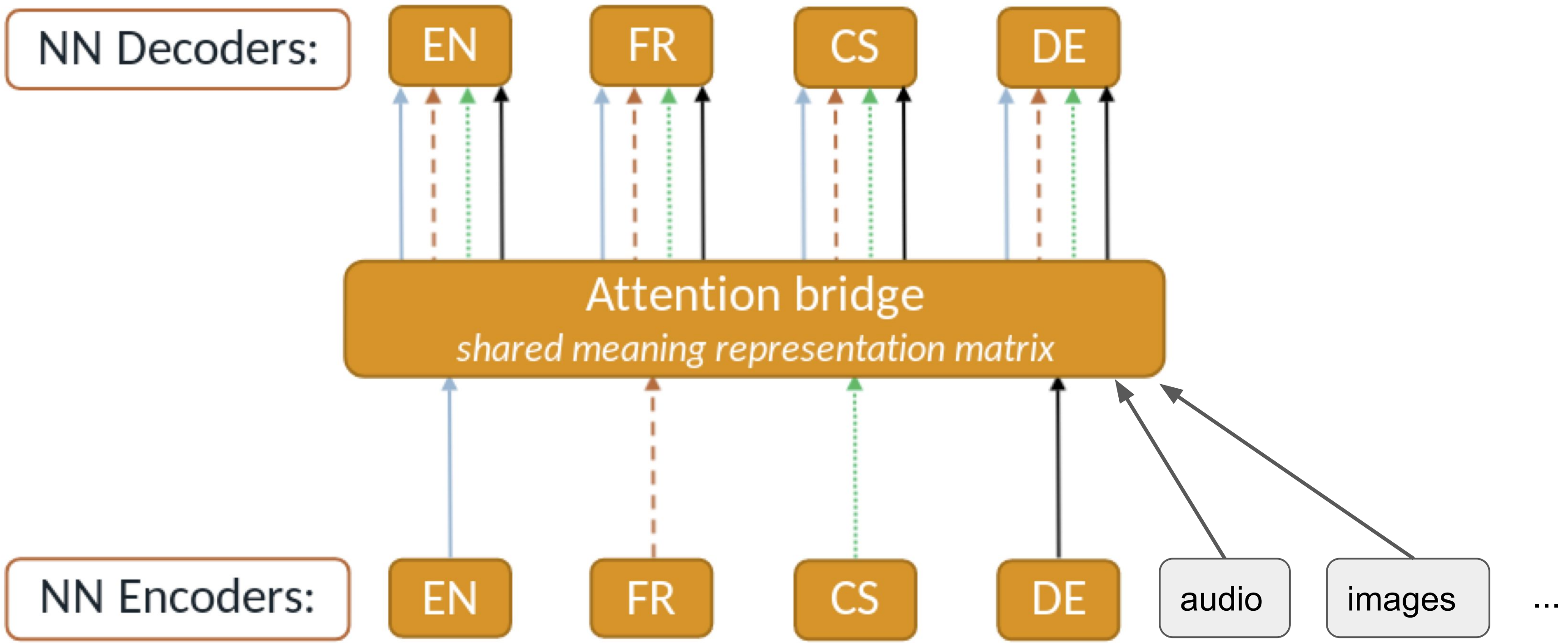
Multilingual image caption translation

model with
all languages

language pair
not seen in
training data
(zero shot)



Multi-task learning and multimodality



Hands on …

The attention bridge implementation:
https://github.com/Helsinki-NLP/OpenNMT-py/tree/att-brg

Tutorial with some practical tips about how to train a neural machine translation system:
https://github.com/neubig/nmt-tips