Machine Translation, Attention

Deep Natural Language Processing Class, 2022

Paweł Budzianowski

Discovering building blocks

1. Meaning
2. Word Vectors
3. New NLP task - language modelling
4. New family of ML models - recurrent neural networks
5. Seq2Seq (today)
6. Attention (today)

New task: Machine Translation

Machine translation tackles the problem of translating a sentence *x* from the source language to the target language *y*.

Source language:

On ciągle myśli o niebieskich migdałach.

Target language:

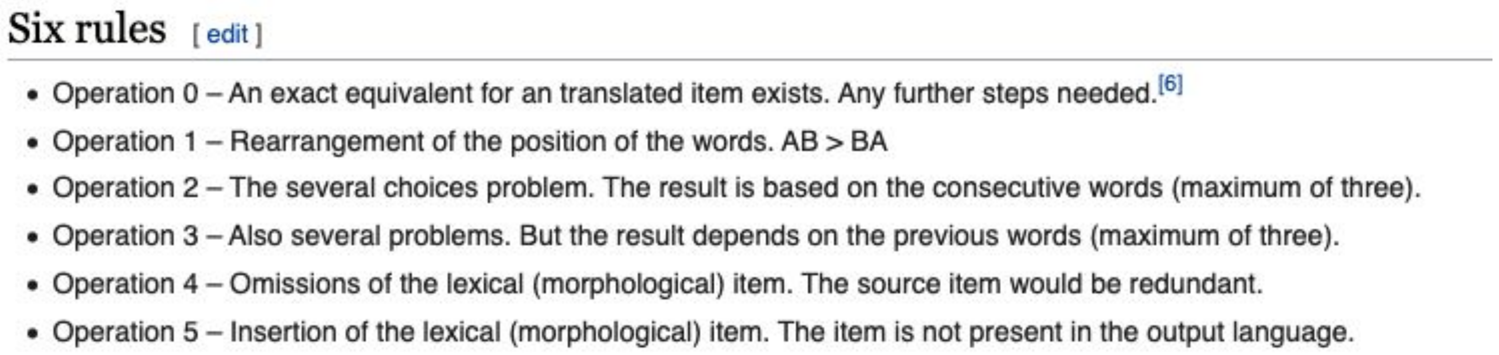
He keeps thinking about blue almonds?
He constantly thinks about blue almonds?

History of MT: 1950s

- Machine Translation research began in the early 1950s.
- Russian → English (motivated by the Cold War!)
- Systems were mostly rule-based, using a bilingual dictionary to map Russian words to their English counterparts.

History of MT

Georgetown-IBM experiment



Six rules [edit]

- Operation 0 – An exact equivalent for an translated item exists. Any further steps needed.[6]
- Operation 1 – Rearrangement of the position of the words. AB > BA
- Operation 2 – The several choices problem. The result is based on the consecutive words (maximum of three).
- Operation 3 – Also several problems. But the result depends on the previous words (maximum of three).
- Operation 4 – Omissions of the lexical (morphological) item. The source item would be redundant.
- Operation 5 – Insertion of the lexical (morphological) item. The item is not present in the output language.

90-10s: Statistical Machine Translation

Core idea: Learn a **probabilistic model** from data

We want to find **best target sentence** *y* given a source sentence *x*:

$$\arg\max_y P(y|x)$$

90-10s: Statistical Machine Translation

Core idea: Learn a **probabilistic model** from data

We want to find **best target sentence** *y* given a source sentence *x*:

$$\arg\max_y P(y|x)$$

Using Bayes Rule to break down into two components to be learned separately:

$$= \arg\max_y P(x|y)P(x)$$

Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Bayes Rule

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

$$P(x|y) \sim P(y|x)P(x)$$

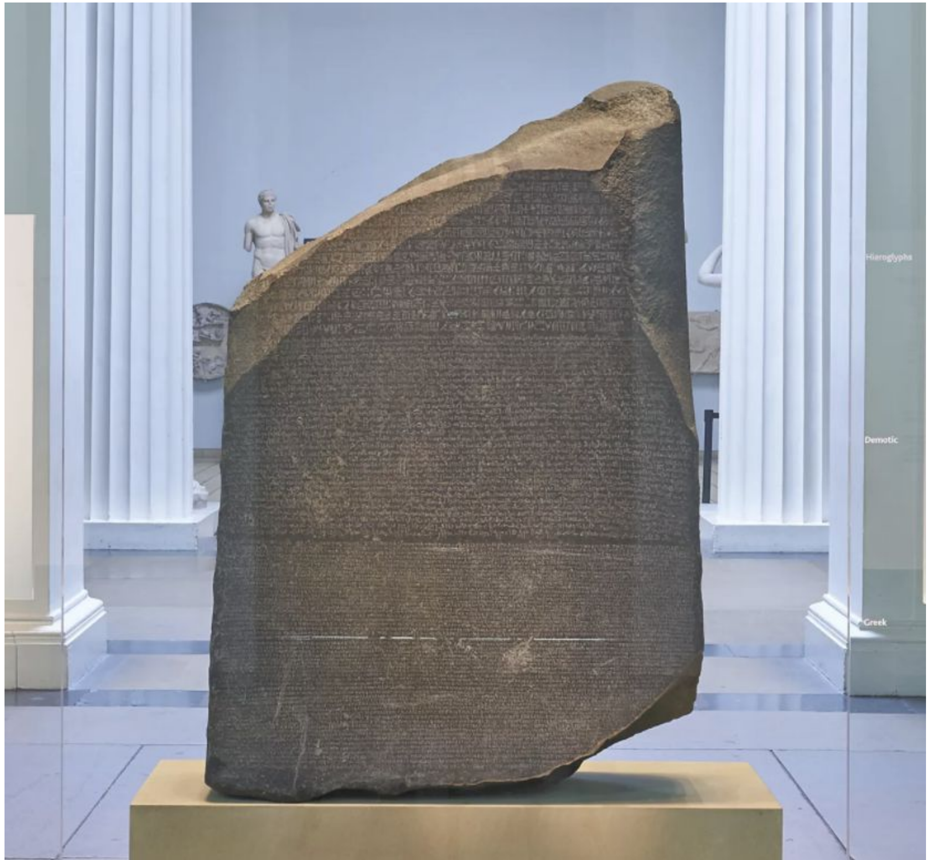
Translation model

Q: How to learn translation model?

First need large amount of parallel data

A classic Rosetta Stone example

$$P(x|y)$$

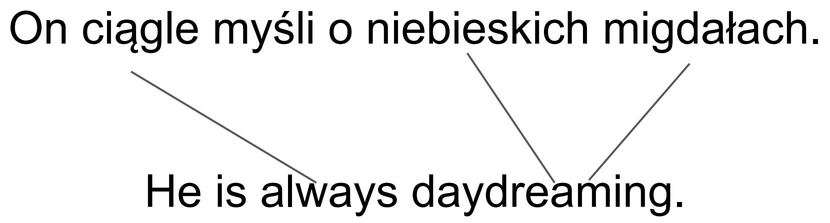


Learning alignment for SMT

Q: How to learn translation model $P(x|y)$ from the parallel corpus?

Break it down further: introduce latent a variable into the model $P(x, a|y)$

where *a* is the alignment, i.e word-level correspondence between source sentence *x* and target sentence *y*.


On ciągle myśli o niebieskich migdałach.
He is always daydreaming.

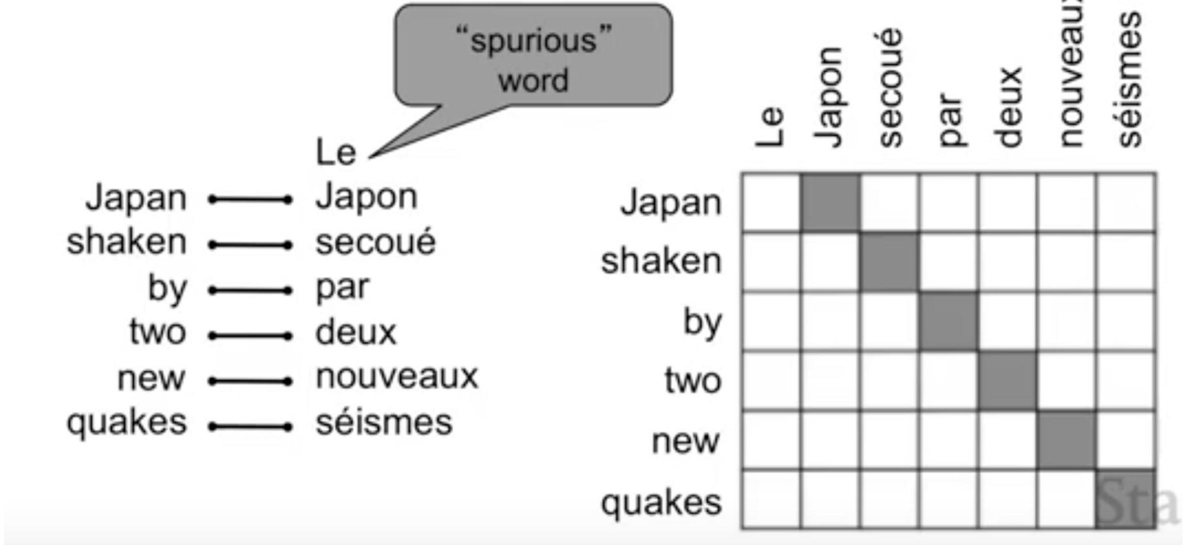
What is alignment?

Alignment is the **correspondence between particular words** in the translated sentence pair.

Typological differences between languages lead to complicated alignments.

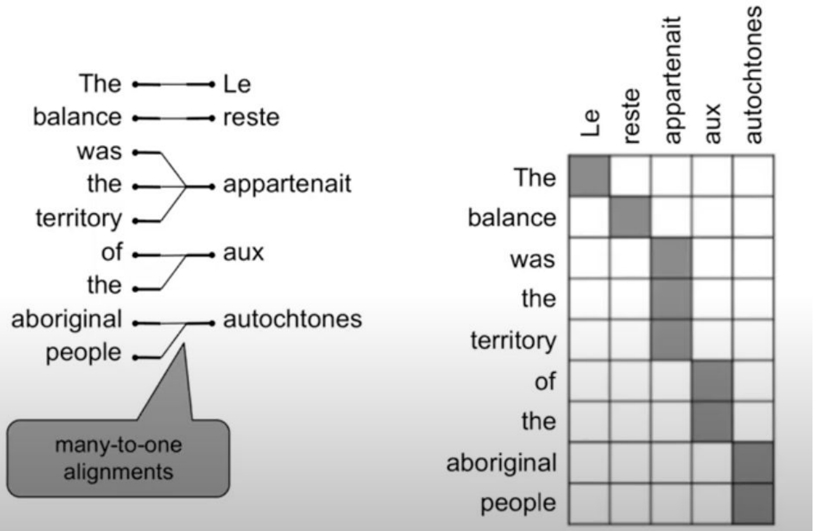
Note: some words have **no counterpart**.

Examples - one-to-one [Manning, 2021]

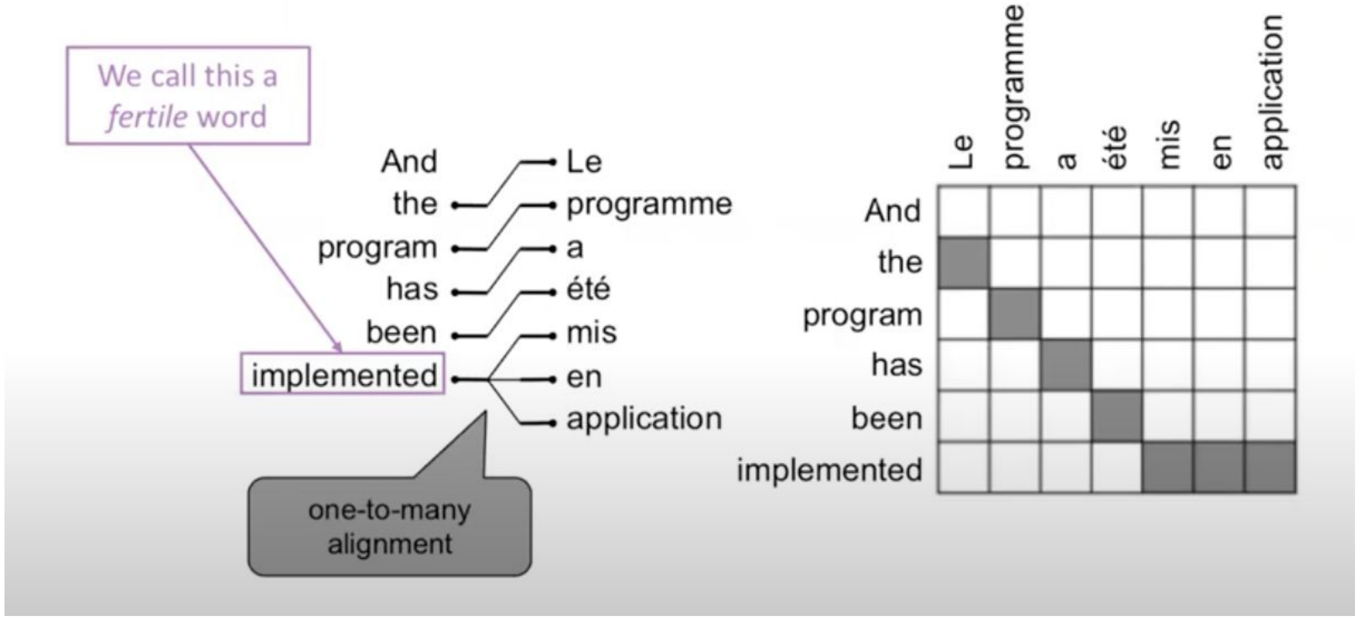


Examples - many-to-one

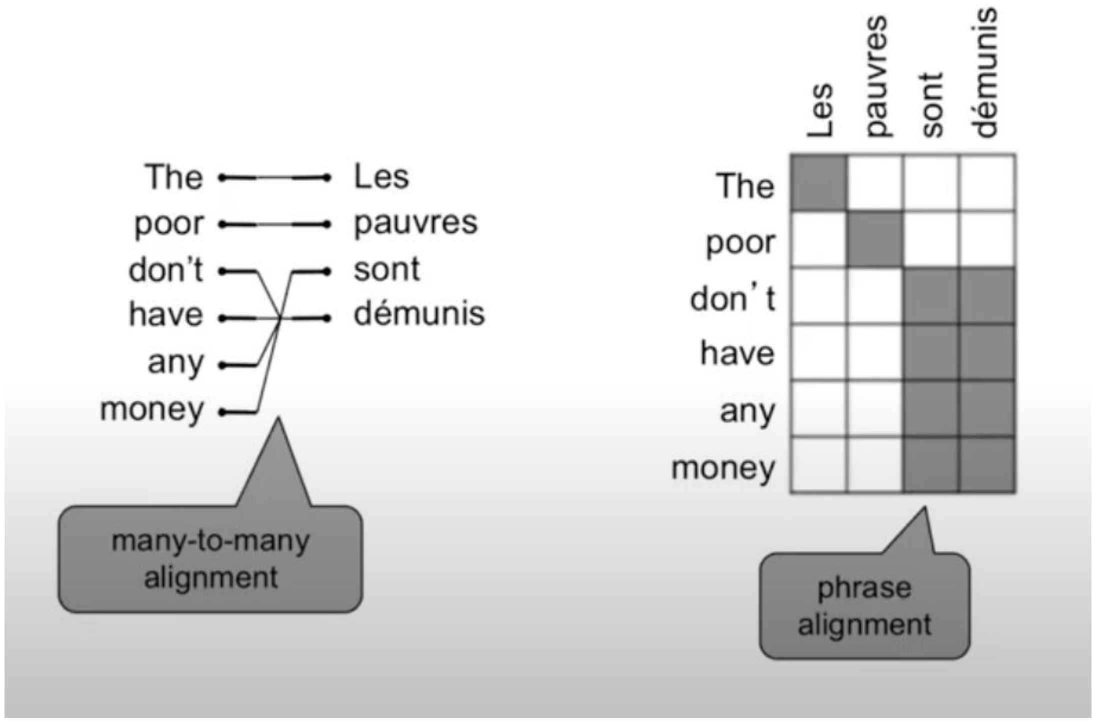
Alignment can be many-to-one



Examples - one-to-many



Examples - many-to-many



Learning alignment for SMT

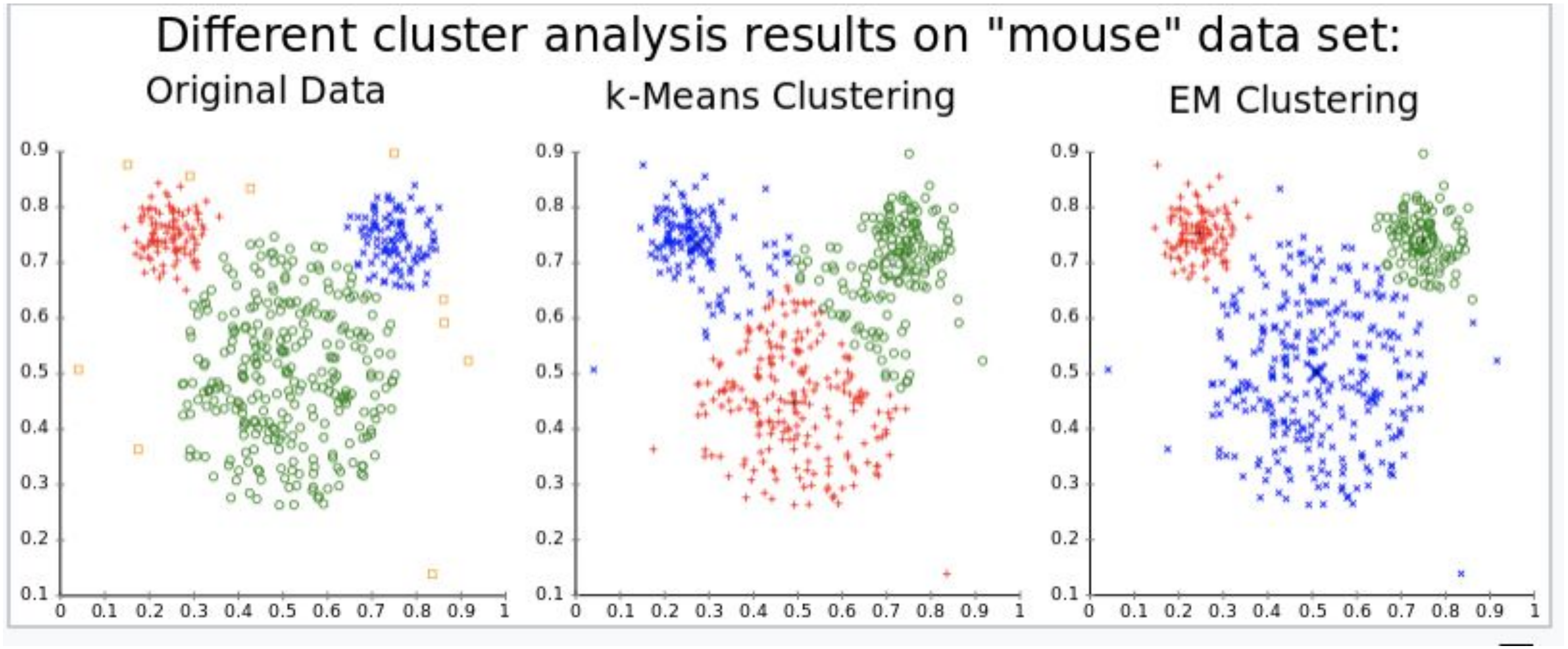
We learn $P(x, a|y)$ as a combination of many factors, including:

- probability of particular words aligning (also depends on position in sentence)
- probability of particular words having a particular number of corresponding words

Alignments *a* are latent variables: they aren't explicitly specified in the data

- require the use of Expectation-Maximization for learning the parameters of distributions with latent variables

Expectation-Maximization Sketch



Expectation-Maximization Sketch

Expectation:

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = \operatorname{E}_{\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)}}\left[\log L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Z})\right]$$

Expectation-Maximization Sketch

Expectation:

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = \mathrm{E}_{\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{(t)}}[\log L(\boldsymbol{\theta}; \mathbf{X}, \mathbf{Z})]$$

Maximization:

$$\boldsymbol{\theta}^{(t+1)} = \underset{\boldsymbol{\theta}}{\arg\max}\, Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)})$$

Decoding for SMT

$$\arg\max_y P(x|y)P(x)$$

Naive solution: we could enumerate every possible y and calculate the probability.

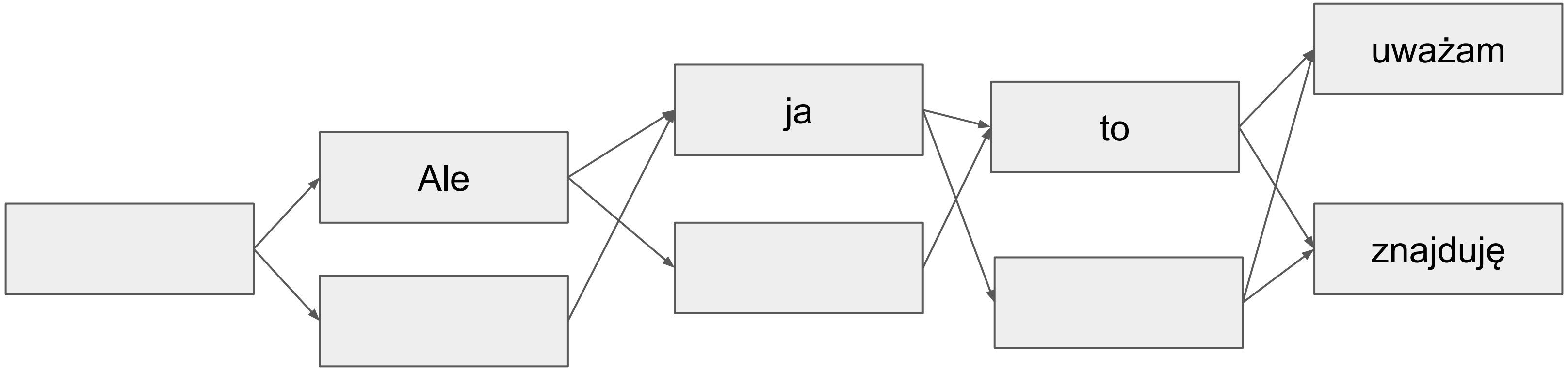
Non-trivial: Impose strong independence assumptions in model, use dynamic programming for globally optimal solutions (e.g. Viterbi algorithm).

This is called **decoding process**.

Example to polish

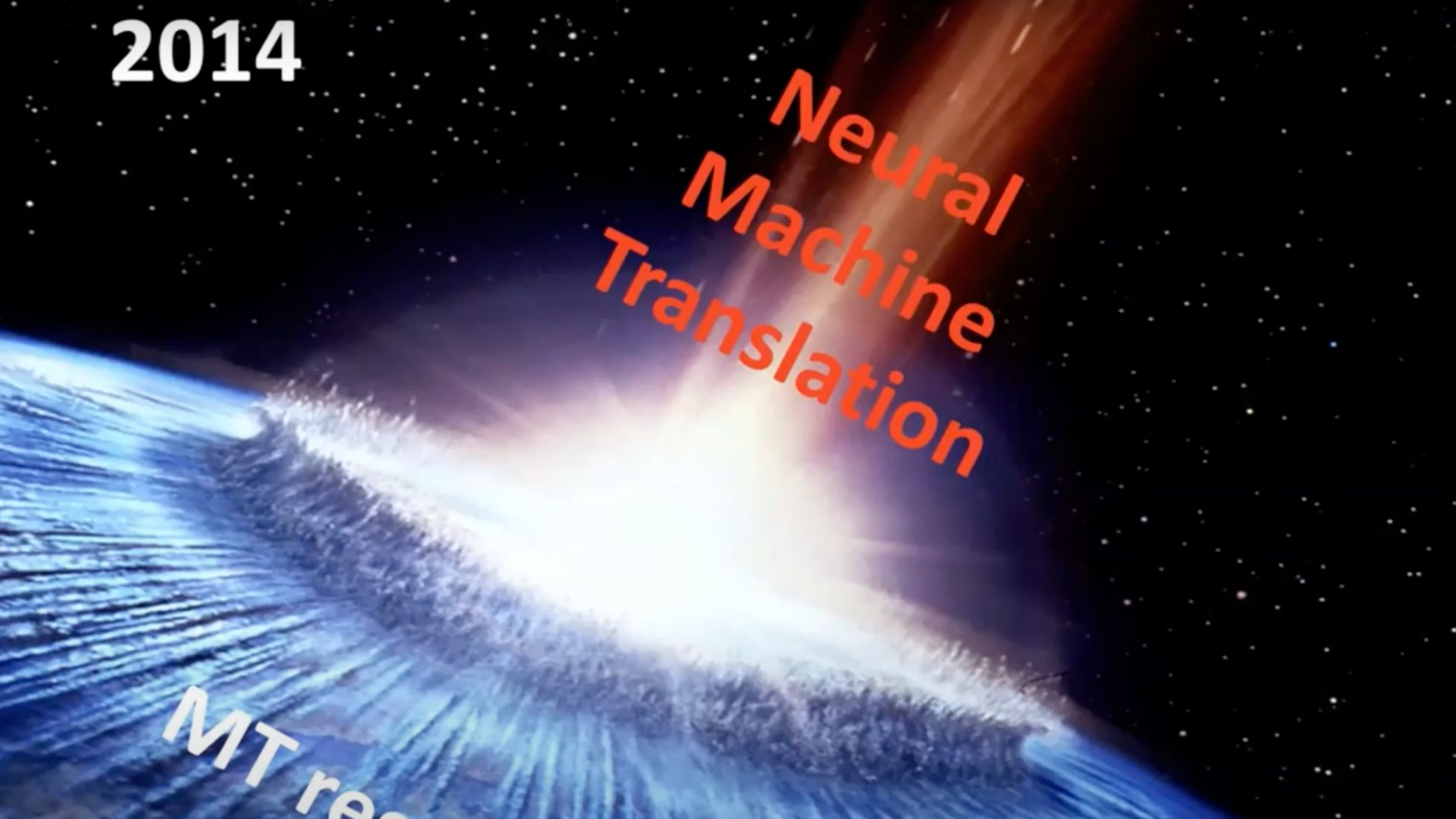
Source: But I find this

Target:



SMT

- SMT was a huge **research field**
- The best systems were **extremely complex**
    - hundreds of important details to take into account
    - separately-designed subcomponenets
- Lots of feature engineering
- Require compiling and maintaining extra resources
- Lots of human effort to maintain

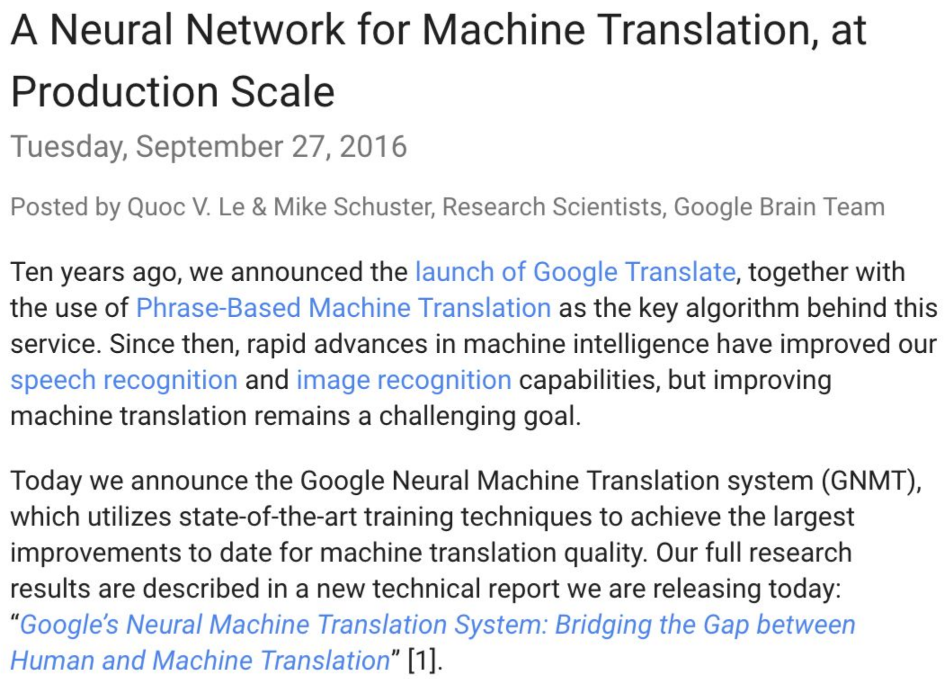


2014

Neural Machine Translation

MT re

Neural Machine Translation



A Neural Network for Machine Translation, at Production Scale

Tuesday, September 27, 2016

Posted by Quoc V. Le & Mike Schuster, Research Scientists, Google Brain Team

Ten years ago, we announced the launch of Google Translate, together with the use of Phrase-Based Machine Translation as the key algorithm behind this service. Since then, rapid advances in machine intelligence have improved our speech recognition and image recognition capabilities, but improving machine translation remains a challenging goal.

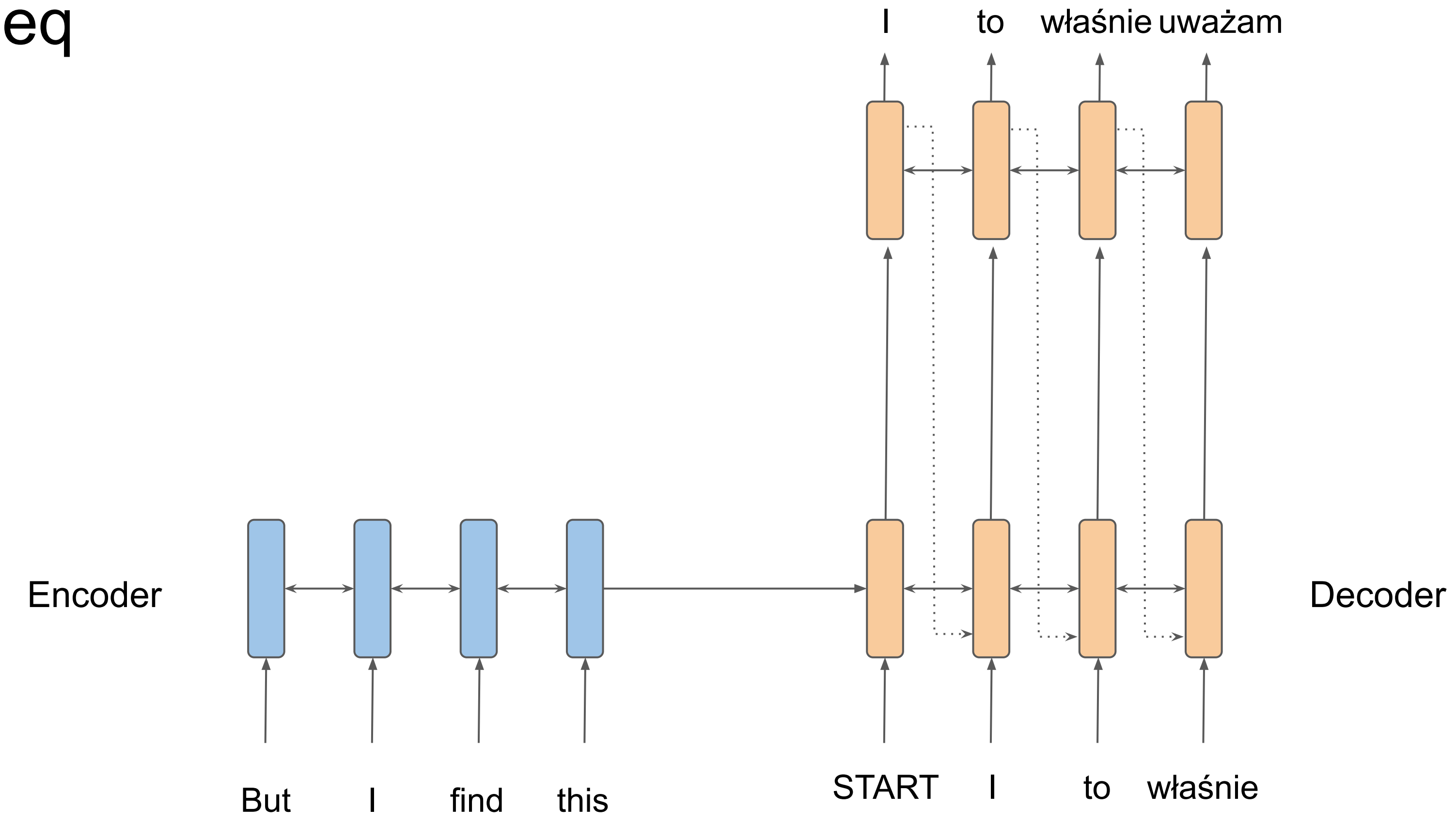
Today we announce the Google Neural Machine Translation system (GNMT), which utilizes state-of-the-art training techniques to achieve the largest improvements to date for machine translation quality. Our full research results are described in a new technical report we are releasing today: "*Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*" [1].

Neural Machine Translation

NMT is a way to do machine translation with a single end-to-end neural network.

The neural network architecture is called a sequence-to-sequence model (aka seq2seq) and it involves two NNs.

Seq2Seq


I
to
właśnie
uważam
Encoder
Decoder
But
I
find
this
START
I
to
właśnie

Seq2Seq are powerful

- You can do plenty of things with this paradigm
- To name a few:
  - summarization
  - dialogue
  - parsing
  - code generation
  - grammar correction
  - end many more!

Neural Machine Translation

- The sequence-to-sequence model is an example of **conditional language model:**
  - **language model** because the decoder is predicting the next word
  - **conditional** because its predictions are also conditioned on the source sentence *x*

- NMT directly calculates $P(y|x)$

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

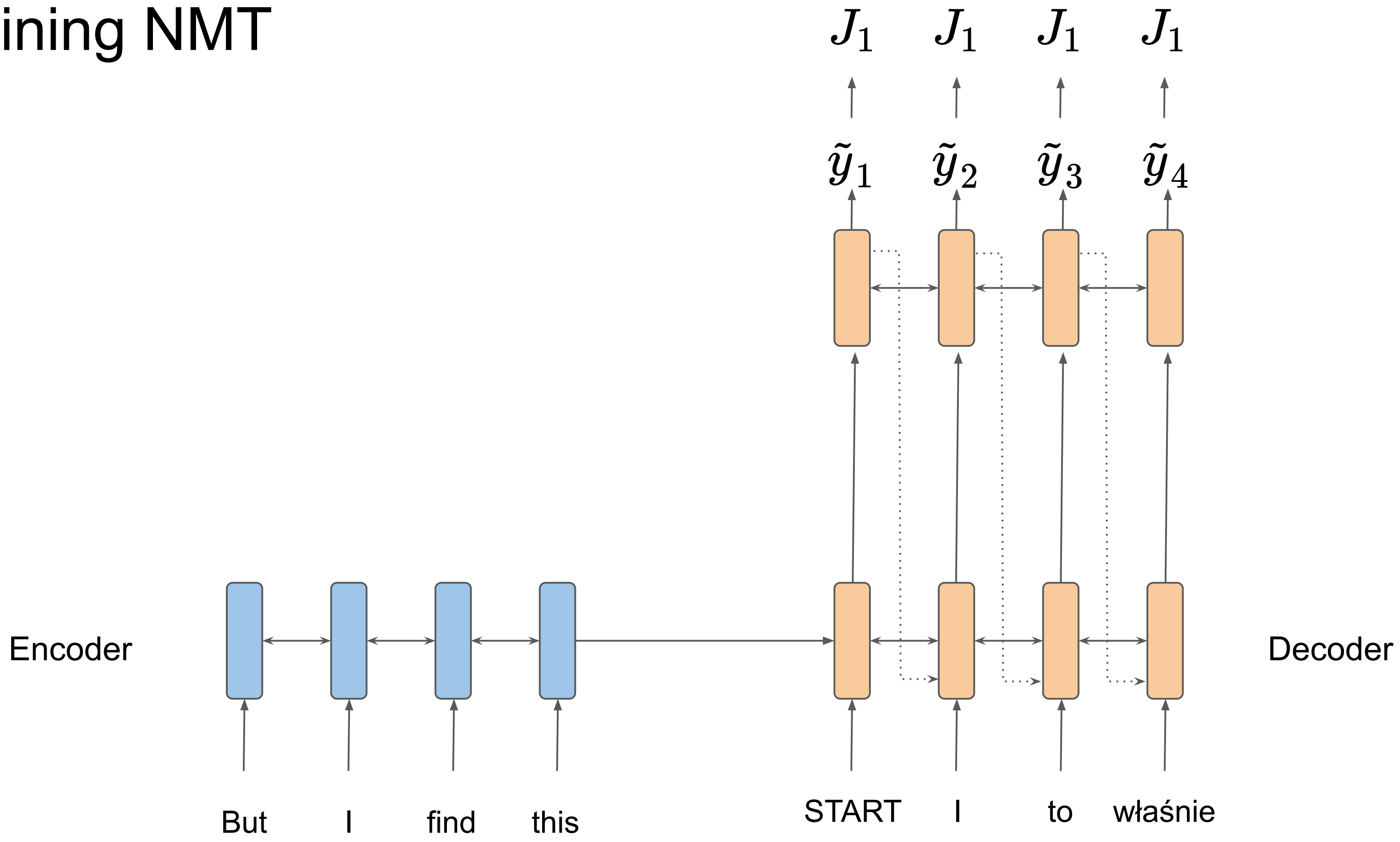
Training NMT

Question: How to train a NMT model?

Answer: get a lot of **parallel data**.

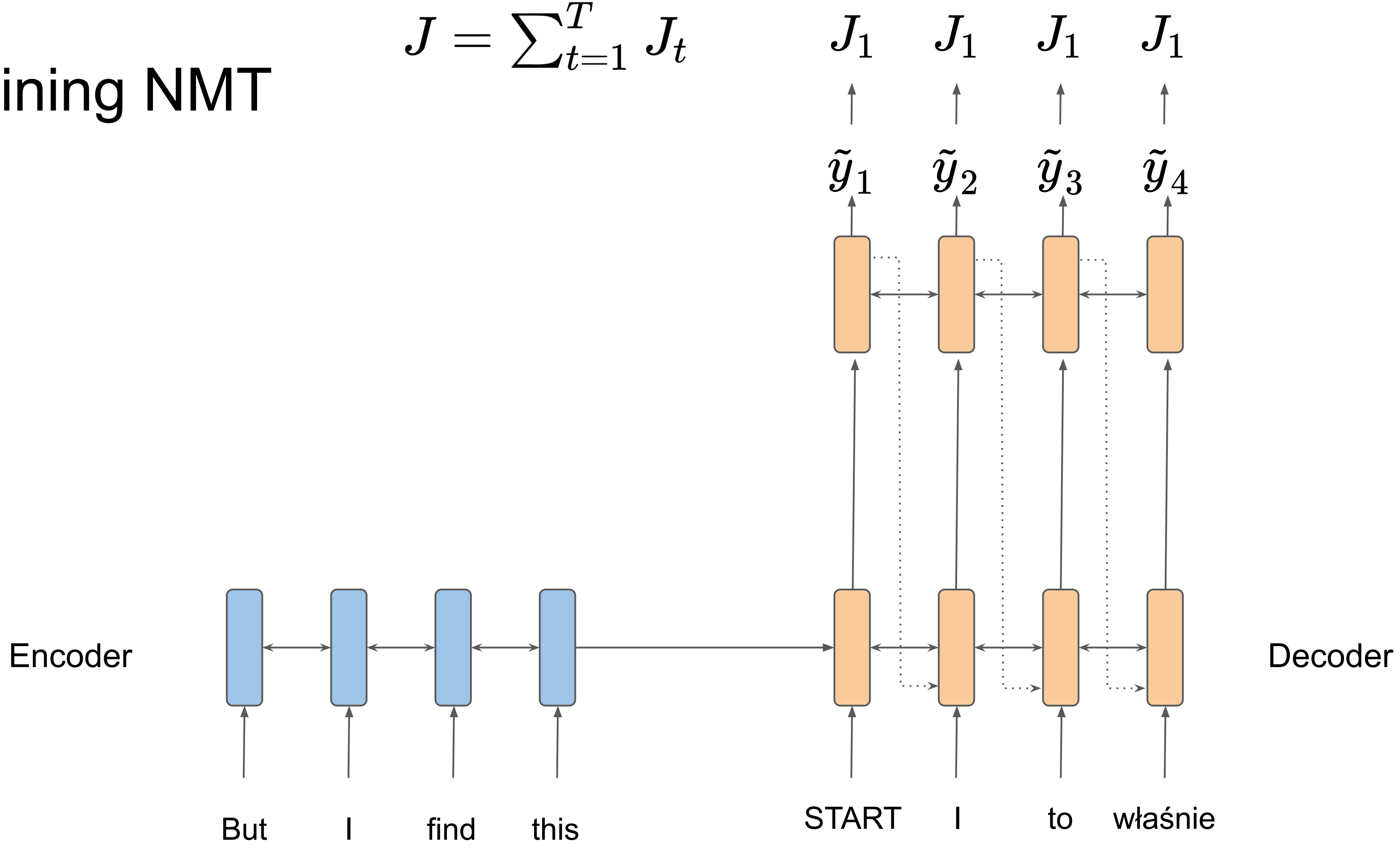
Seq2Seq is optimized as a single system because backpropagation allows to train it in an end-to-end fashion.

Training NMT



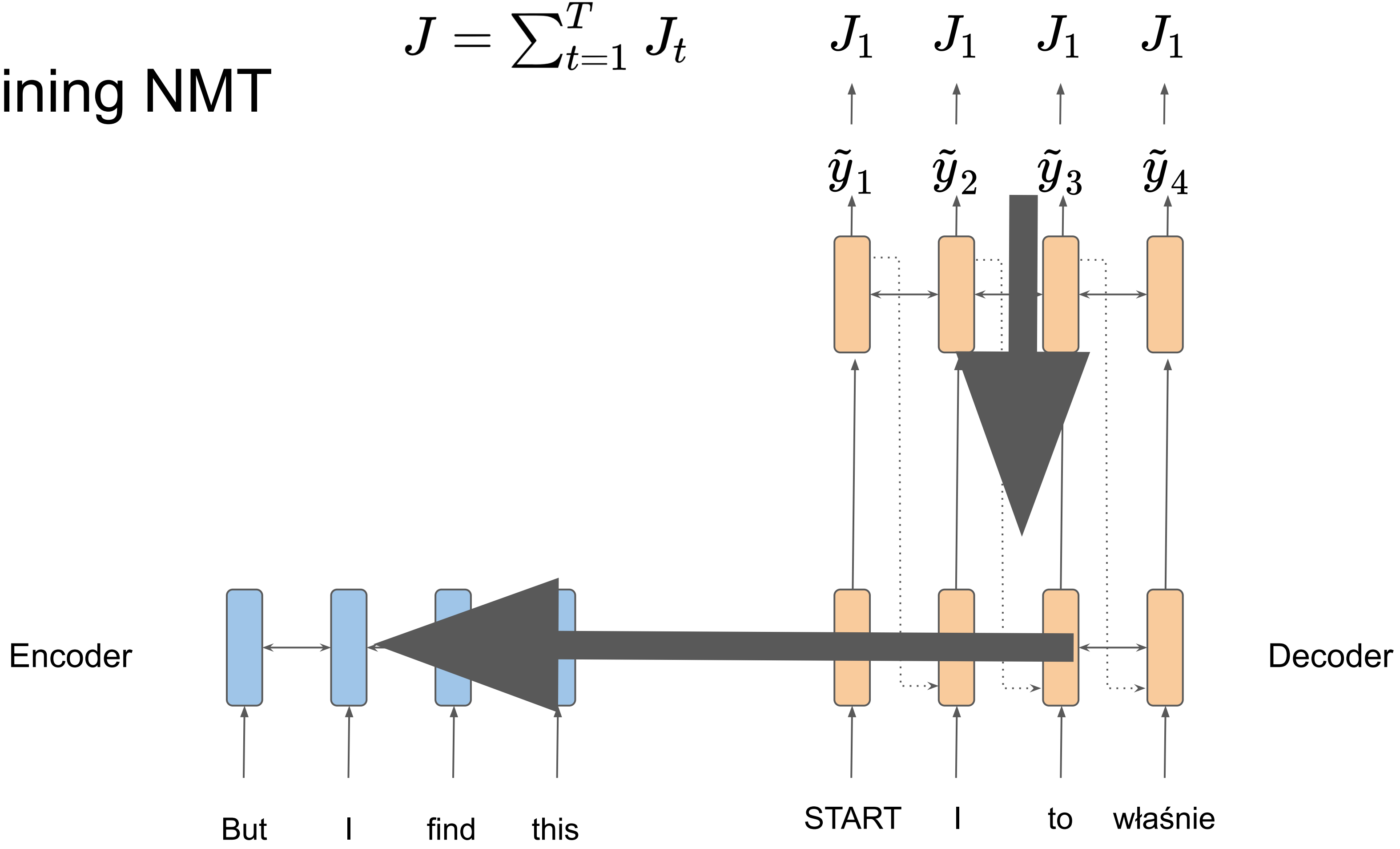
Training NMT

$$J = \sum_{t=1}^{T} J_t$$



Training NMT

$$J = \sum_{t=1}^{T} J_t$$

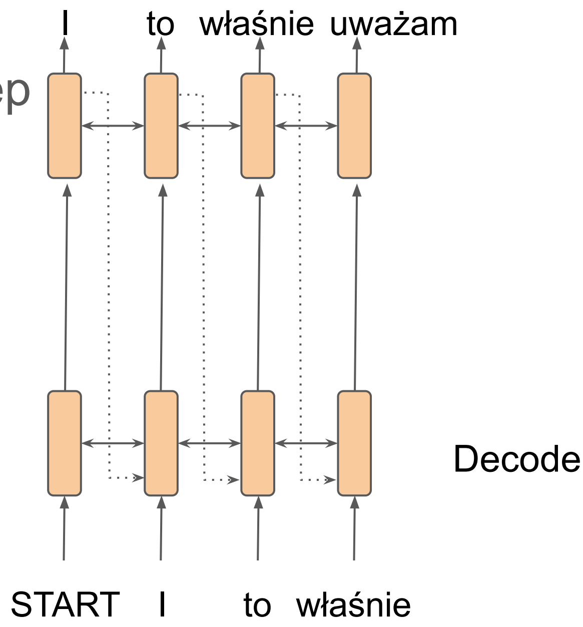


Decoding part

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

Greedy decoding - taking most probable word on each step

Problems with this method?



Problems with greedy decoding

- Greedy decoding has no way to undo decisions
- What about context?

But I find this:

Ale

Ale ja

Ale ja to

Ale ja to znajduję

Question: What are better ways to decode?

Naive decoding

Ideally we want to find a translation that **maximizes**:

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

Naive decoding

Ideally we want to find a translation that **maximizes**:

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

We could try computing all possible sequences y:

- this is O(V^T) complexity
- completely intractable

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** (typically 5 or 10)

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

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

We search for high-scoring hypotheses, tracking top k on each step

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** (typically 5 or 10)

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

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

We search for high-scoring hypotheses, tracking top k on each step

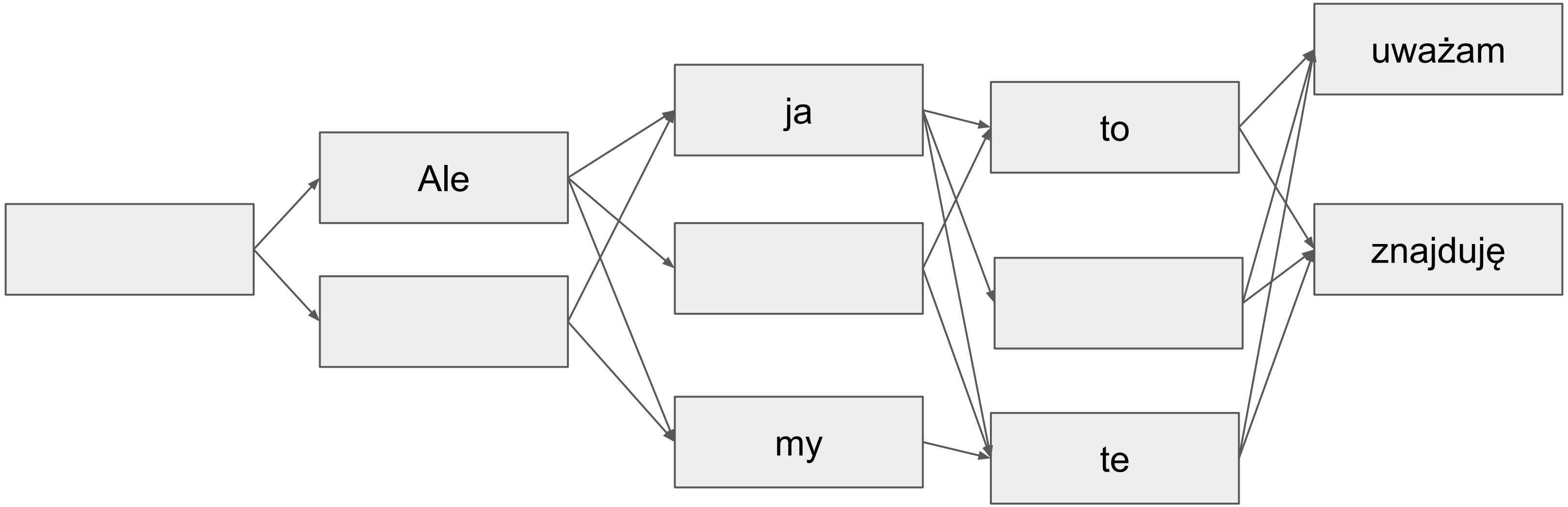
Beam search is not guaranteed to find **optimal path**!

But it’s really efficient.

Beam search decoding example (k=2)

Source: But I find this

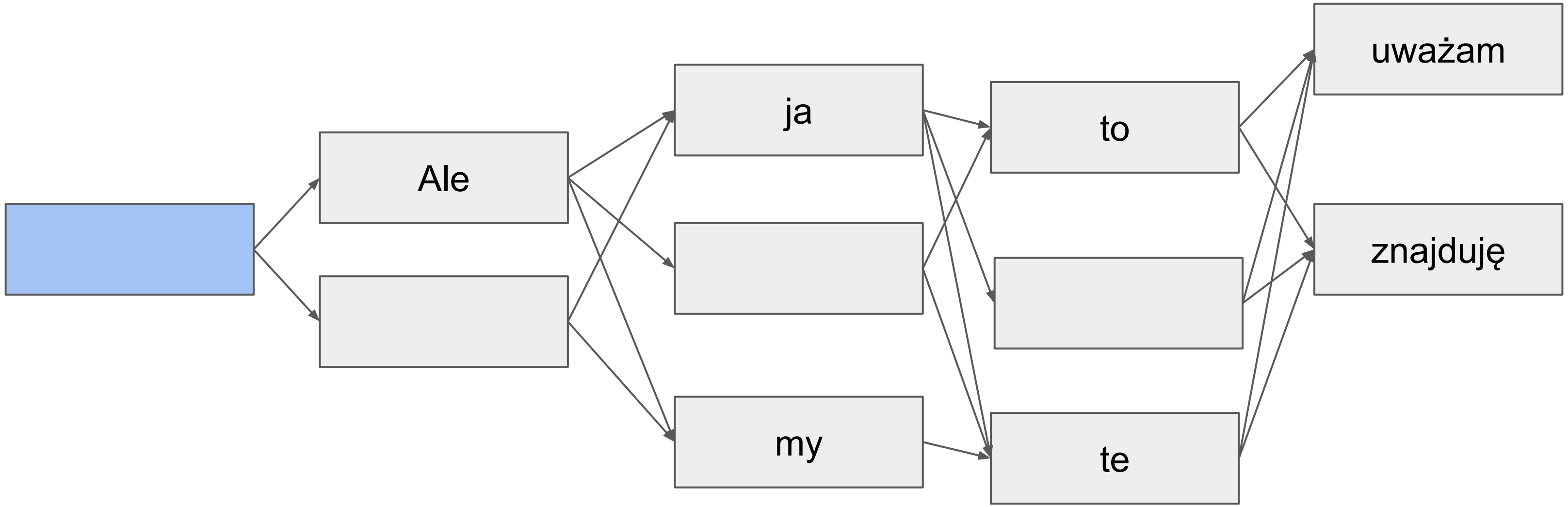
Target:



Beam search decoding example (k=2)

Source: But I find this

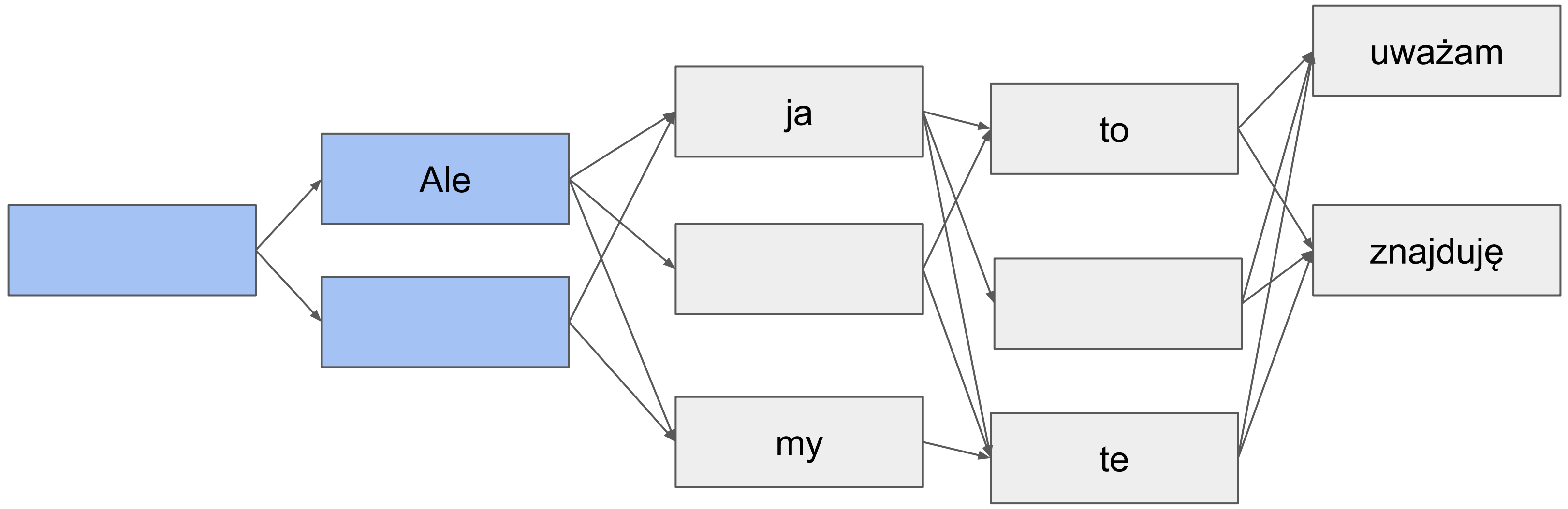
Target:



Beam search decoding example (k=2)

Source: But I find this

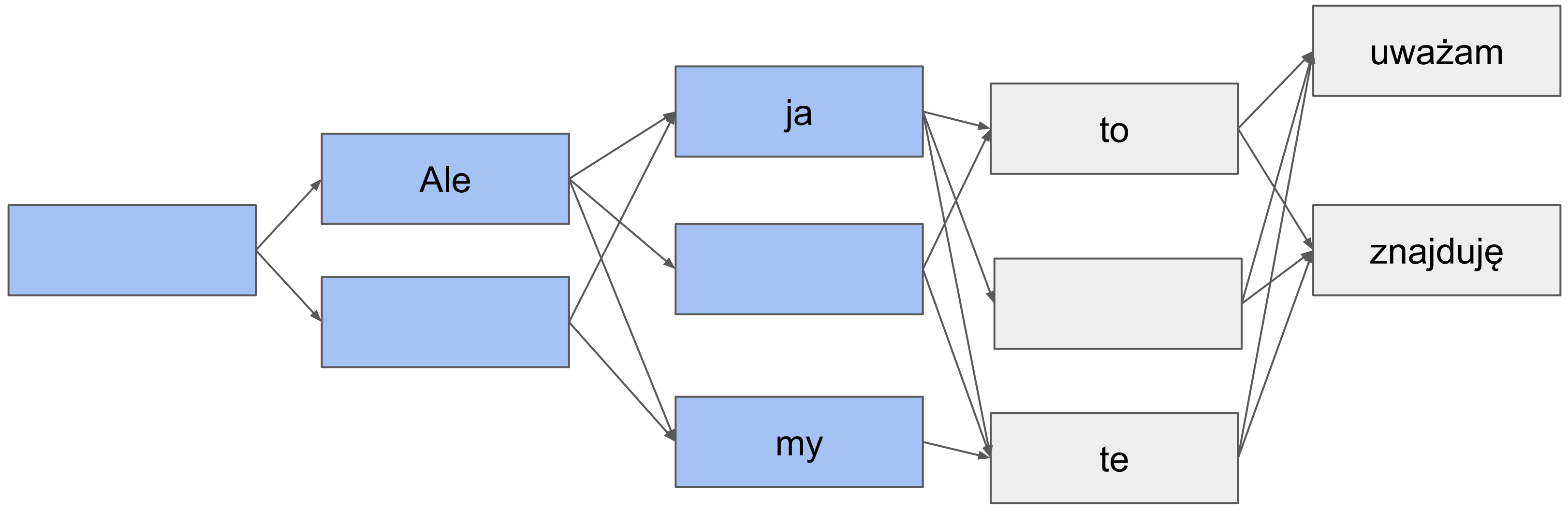
Target:



Beam search decoding example (k=2)

Source: But I find this

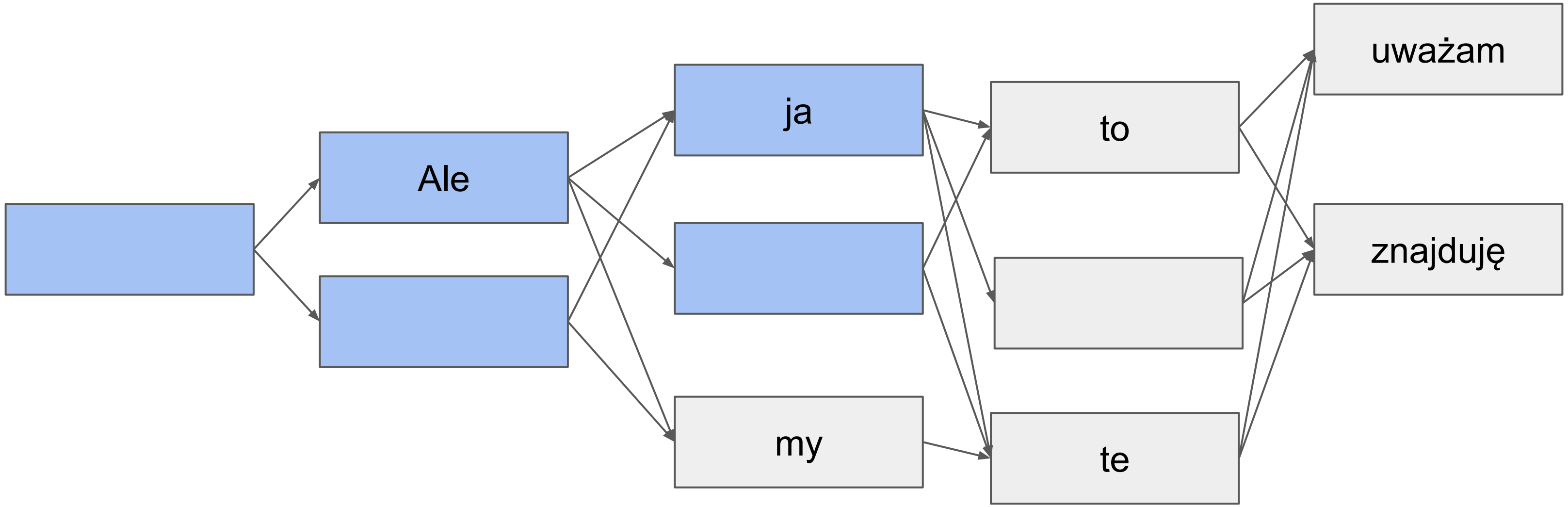
Target:



Beam search decoding example (k=2)

Source: But I find this

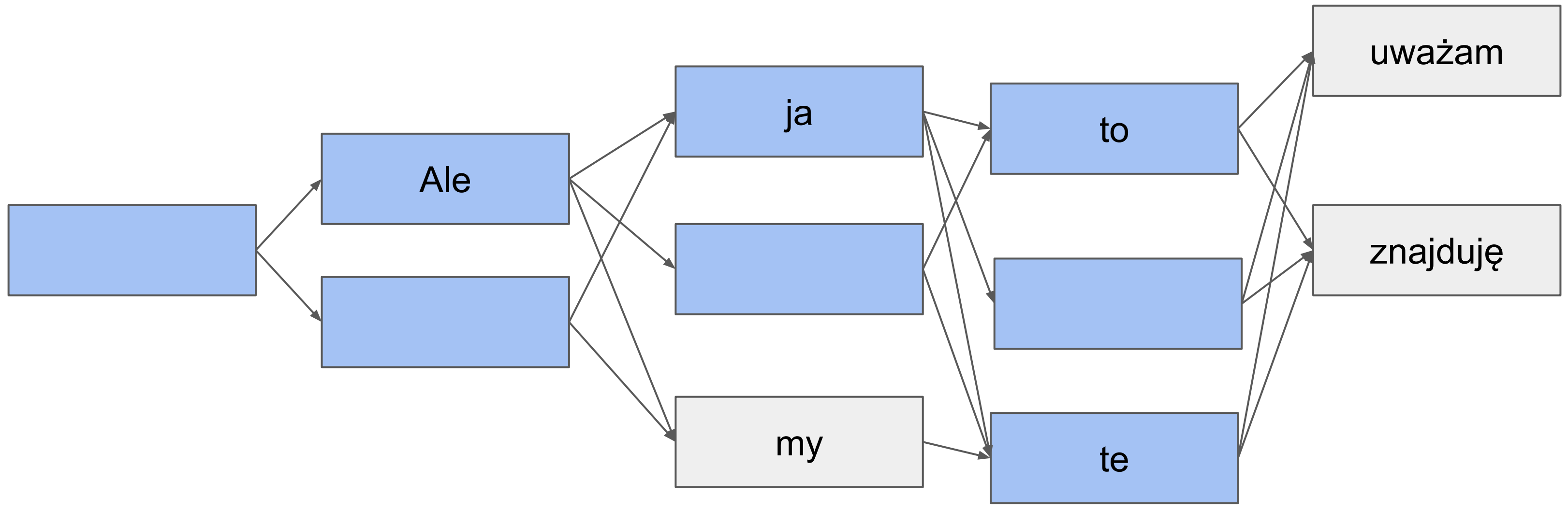
Target:



Beam search decoding example (k=2)

Source: But I find this

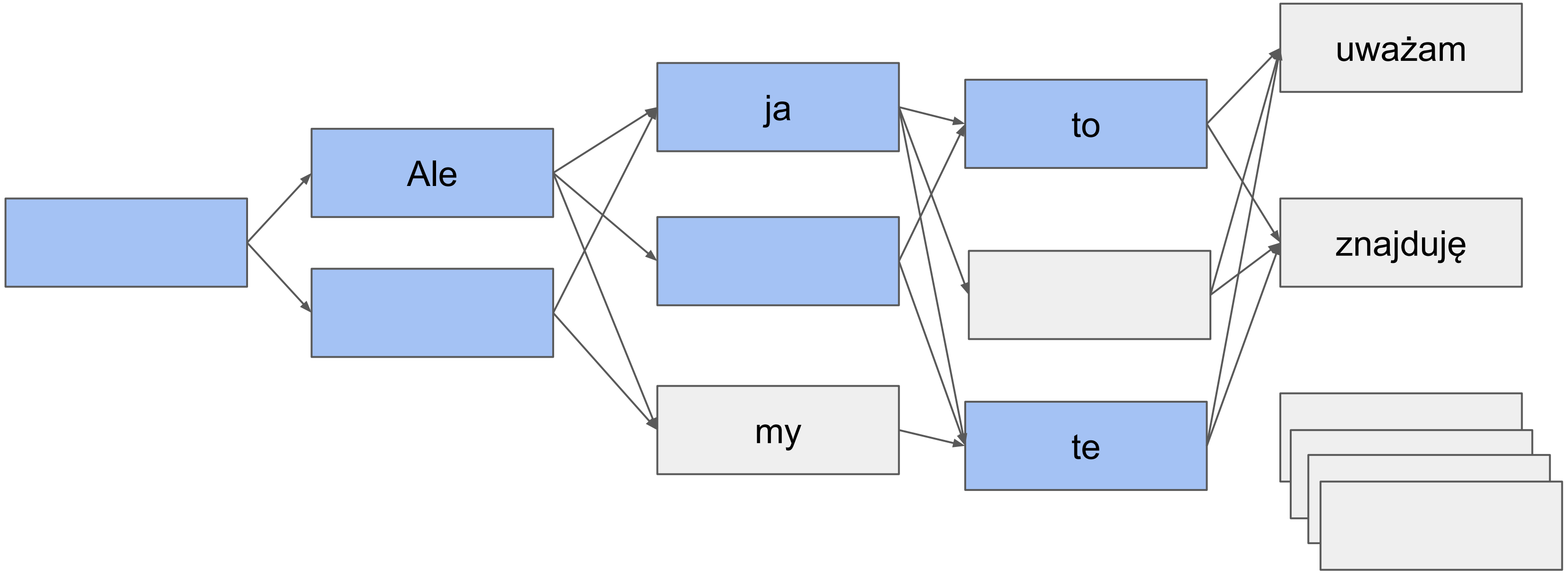
Target:



Beam search decoding example (k=2)

Source: But I find this

Target:



Beam search decoding

In **greedy decoding**, typically we decode until the model produces an **END token.**

In **beam search decoding**, different hypotheses may produce END tokens on different timesteps

- When a hypothesis produces END token, that hypothesis is complete.
- Place it aside and continue exploring other hypotheses via beam search.

Typically we continue beam search until:

- we reach timestep *T* step (the budget of generation)
- we heave at least *n* completed hypotheses

Beam search decoding - finishing up

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

Problem - **longer hypotheses have lower scores.**

Beam search decoding - finishing up

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

Problem - **longer hypotheses have lower scores.**

Fix: **Normalize by length** and use the hypotheses that has the highest score.

Advantages of NMT

Compared to SMT:

- **Better performance**
    - more fluent
    - better use of context
    - better use of phrase similarities
- 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 languages!

Disadvantages of NMT

Compared to SMT:

- NMT is less interpretable
  - hard to debug
- NMT is difficult to control
  - for example, can’t easily specify rules or guidelines for translation
  - safety concerns!

Evaluation

How do we evaluate Machine Translation?

How do we evaluate Machine Translation?

BLEU - **B**ilingual **E**valuation **U**nderstudy

BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:

- *n*-gram precision (typically for 1,2,3 and 4-grams)
- plus a penalty for too-short system translations

How do we evaluate Machine Translation?

BLEU - **B**ilingual **E**valuation **U**nderstudy

BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:

- *n*-gram precision (typically for 1,2,3 and 4-grams)
- plus a penalty for too-short system translations

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{C' \in \{Candidates\}} \sum_{n\text{-}gram' \in C'} Count(n\text{-}gram')}.$$

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}.$$

Then,

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right).$$

BLEU computation example

Original: el amor todo lo puede

Reference 1: love can always find a way

Reference 2: love makes anything possible

Candidate: the love can always do

Compute the BLUE score for 1 and 2-grams only.

BLEU computation example

$$\begin{aligned}
p_1 &= \frac{\Sigma_{g\in\{\text{the, love, can, always, do}\}} min(max_{i=1,2} Count_{r_i}(g), Count_{c_1}(g))}{\Sigma_{g\in\{\text{the, love, can, always, do}\}} Count_{c_1}(g)} \\
&= \frac{0+1+1+1+0}{5} \\
&= \frac{3}{5}
\end{aligned}$$

Reference 1: love can always find a way

Reference 2: love makes anything possible

BLEU computation example

$$
\begin{aligned}
p_2 &= \frac{\Sigma_{g\in\{\text{the love, love can, can always, always do}\}} min(max_{i=1,2} Count_{r_i}(g), Count_{c_1}(g))}{\Sigma_{g\in\{\text{the love, love can, can always, always do}\}} Count_{c_1}(g)} \\
&= \frac{0+1+1+0}{4} \\
&= \frac{1}{2}
\end{aligned}
$$

Reference 1: love can always find a way

Reference 2: love makes anything possible

BLEU computation example

Brevity Penalty: $c = len(c_1) = 5$, $r^* = 5$. Because $c \geq r^*$

$$BP = 1$$

BLEU: $\lambda_1 = \lambda_2 = 0.5$

BLEU computation example

$$\begin{aligned}\text{BLEU}_{c_1} &= BP * exp(\lambda_1 log(p_1) + \lambda_2 log(p_2)) \\ &= 1 * exp(0.5 * log(\frac{3}{5}) + 0.5 * log(\frac{2}{4})) \\ &\approx 0.76994\end{aligned}$$

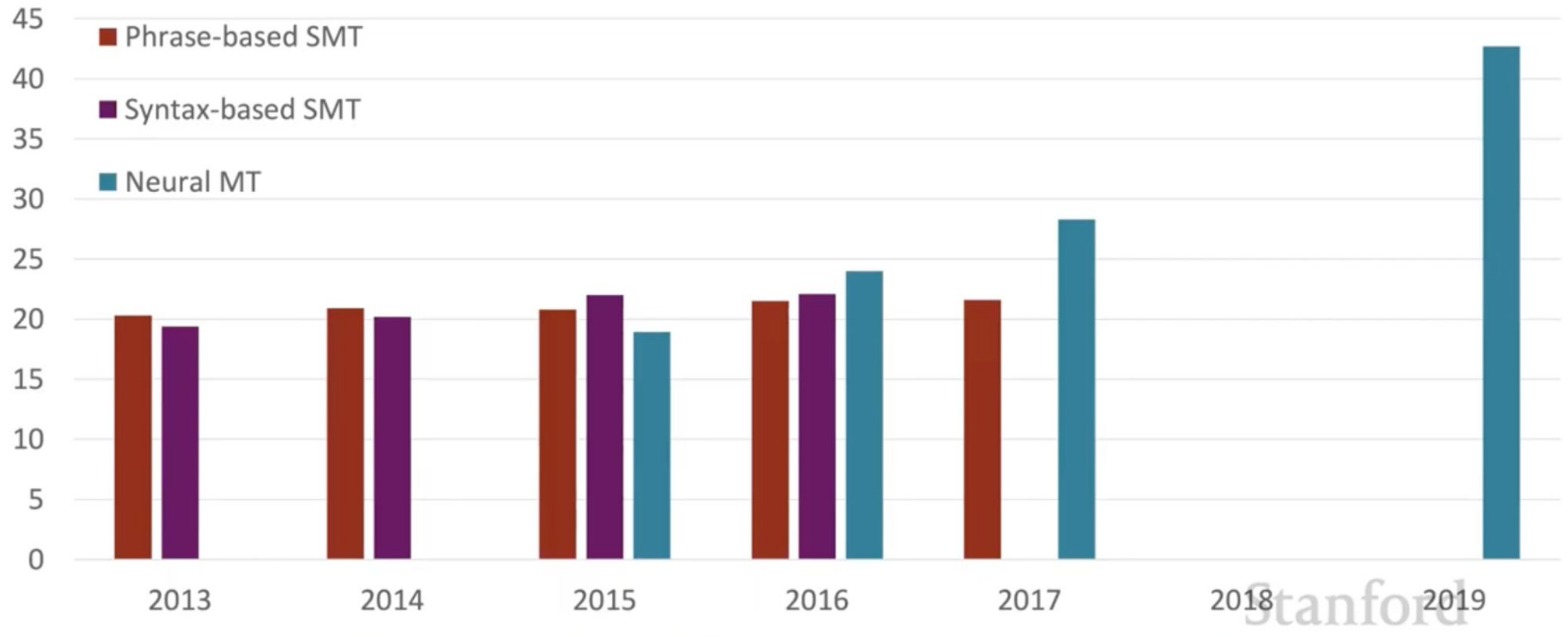
BLEU is useful but not perfect

There are many valid ways to translate a sentence!

So a **good** translation can get a **poor** BLEU score because it has low *n*-gram overlap with the human translation.

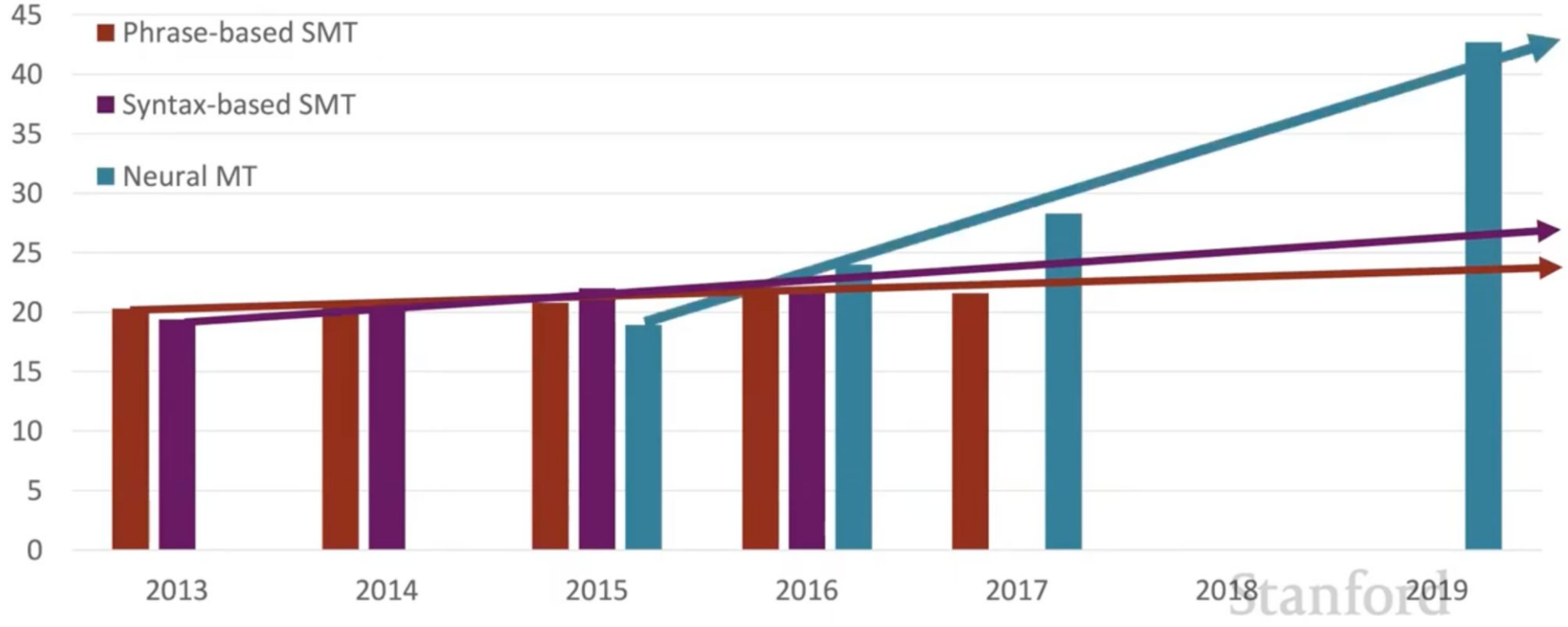
NMT - the path to greatness [Manning, 2021]

NMT - the path to greatness [Manning, 2021]



Sources: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf & http://matrix.statmt.org/

NMT - the path to greatness [Manning, 2021]



Sources: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf & http://matrix.statmt.org/

NMT: the biggest success story of NLP Deep Learning

NMT went from a prototypical research attempt in 2014 to the leading production method in 2016.

NMT: the biggest success story of NLP Deep Learning

NMT went from a prototypical research attempt in 2014 to the leading production method in 2016.

- 2014 - first seq2seq paper published
- 2016 - Google Translate switches from SMT to NMT
- 2018 - everyone one else on board

NMT: the biggest success story of NLP Deep Learning

NMT went from a prototypical research attempt in 2014 to the leading production method in 2016.

- 2014 - first seq2seq paper published
- 2016 - Google Translate switches from SMT to NMT
- 2018 - everyone one else on board

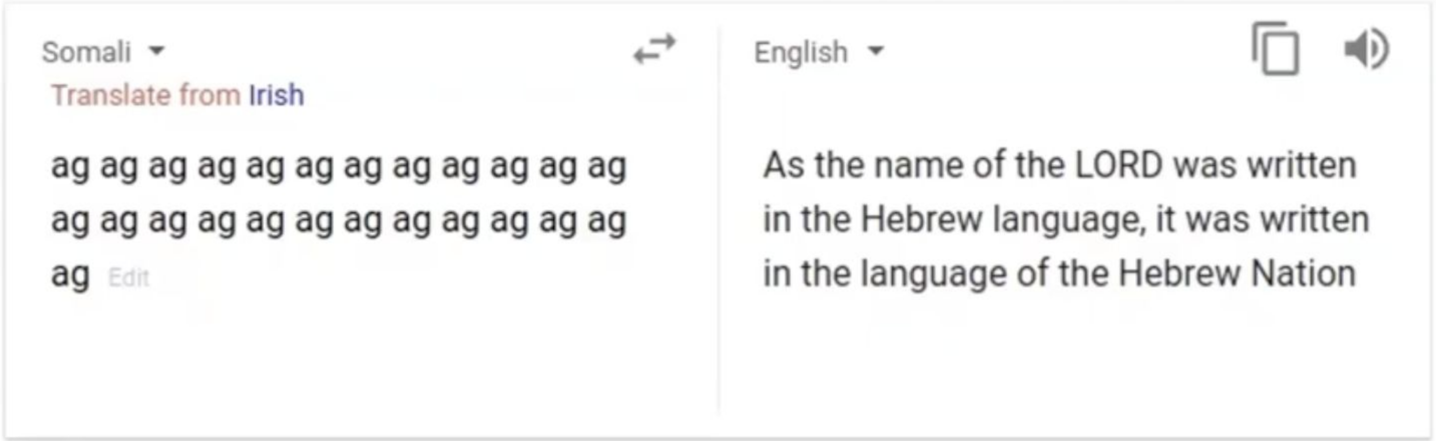
SMT systems built by **hundreds** over many **years** outperformed by NMT systems trained by a **small group** of engineers in a few **months**.

Plenty of problems still exists

Many difficulties remain:

- **Out-of-vocabulary** words
- Domain **mismatch** between train and test data
- Maintaining **context** over longer text
- **Low-resource** language pairs
- Failures to accurately capture **sentence meaning**
- **Pronoun** (or zero pronoun) **resolution** errors
- Morphological agreement errors

Plenty of problems still exists



Somali
Translate from Irish

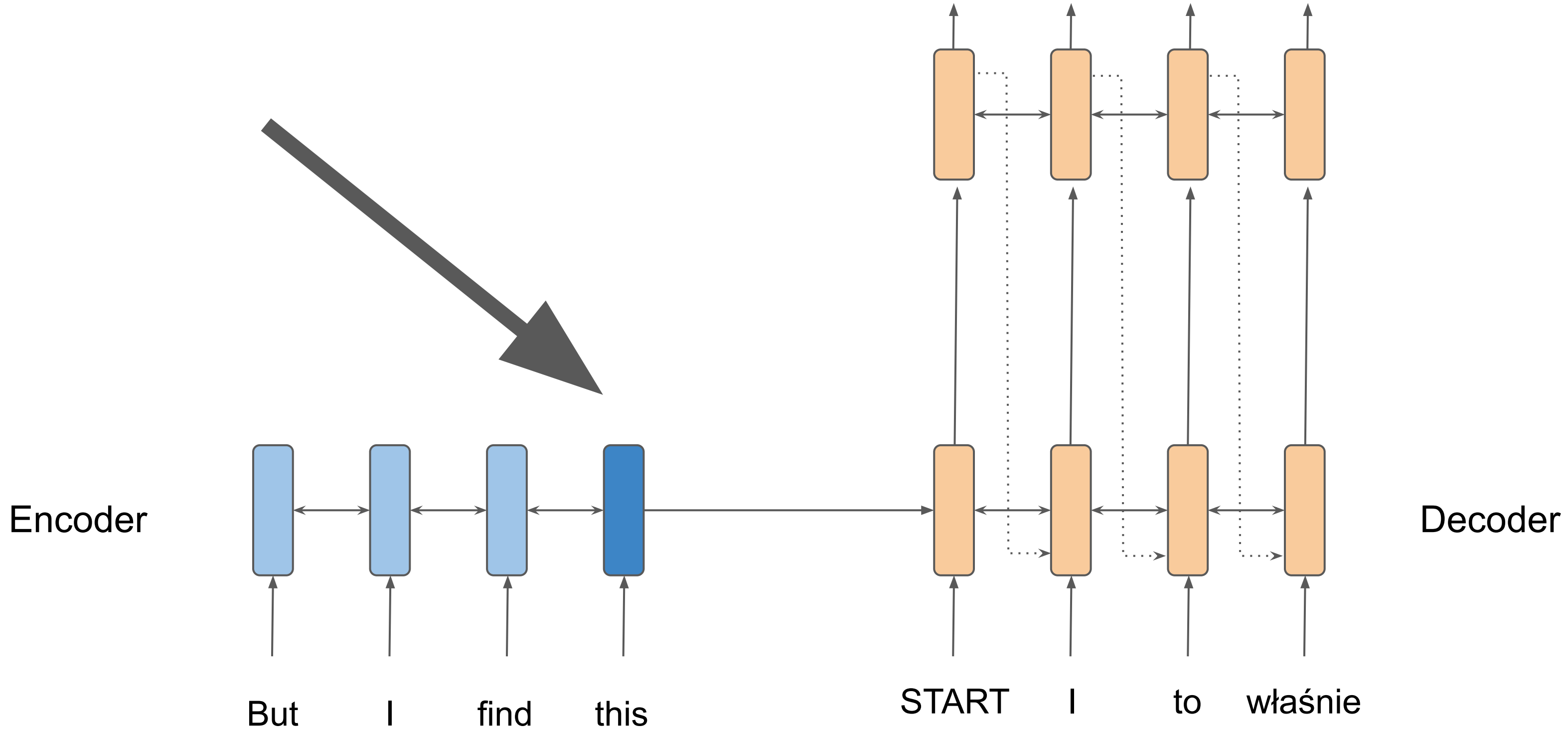
ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag ag

English

As the name of the LORD was written in the Hebrew language, it was written in the language of the Hebrew Nation

ATTENTION

Seq2Seq - the bottleneck problem



Seq2Seq - the bottleneck problem

Encoding of the source sentence. This needs to capture ALL information about the source sentence information bottleneck!

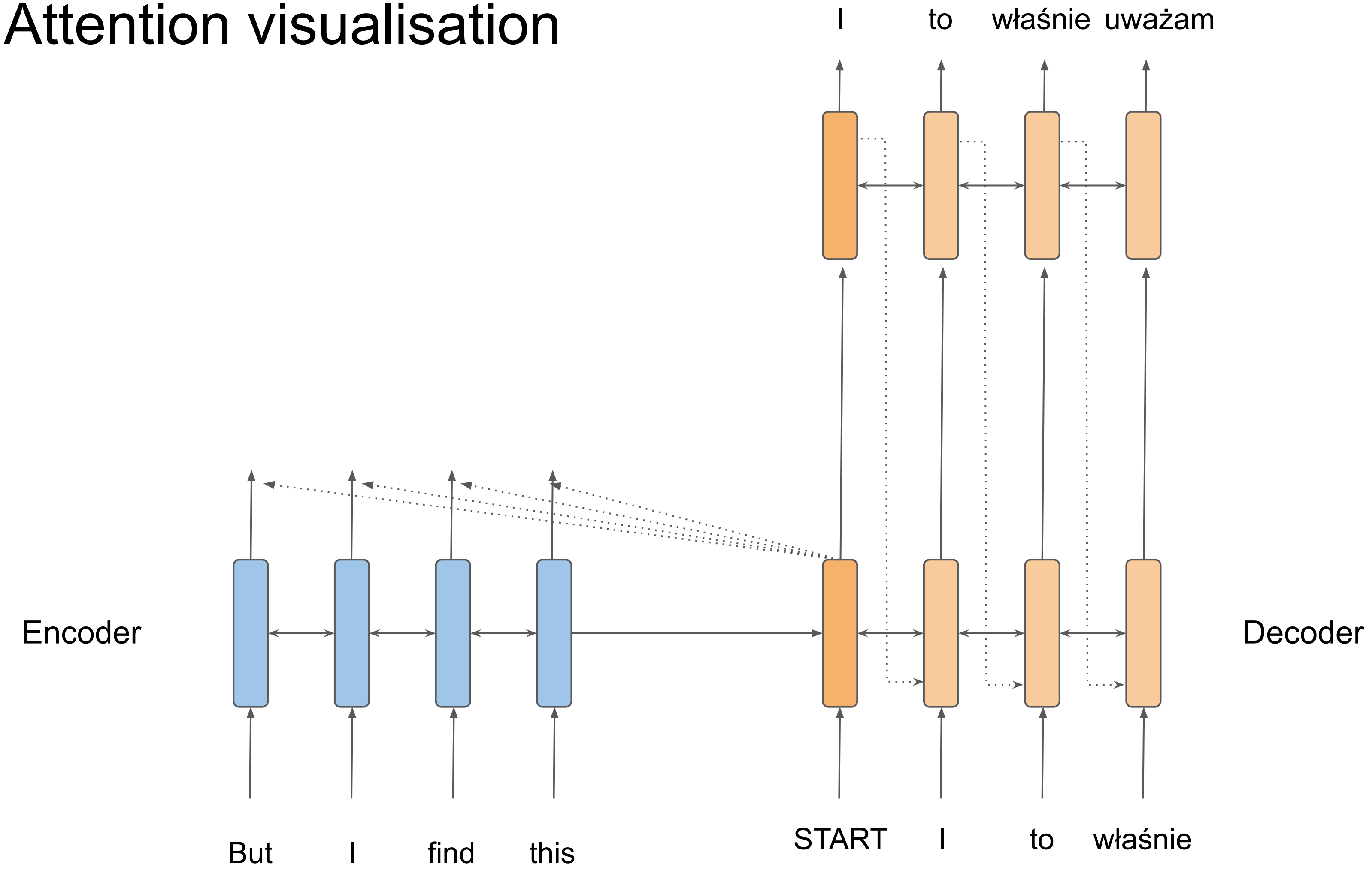
You could average all encodings but the order of words is important to preserve.

Attention

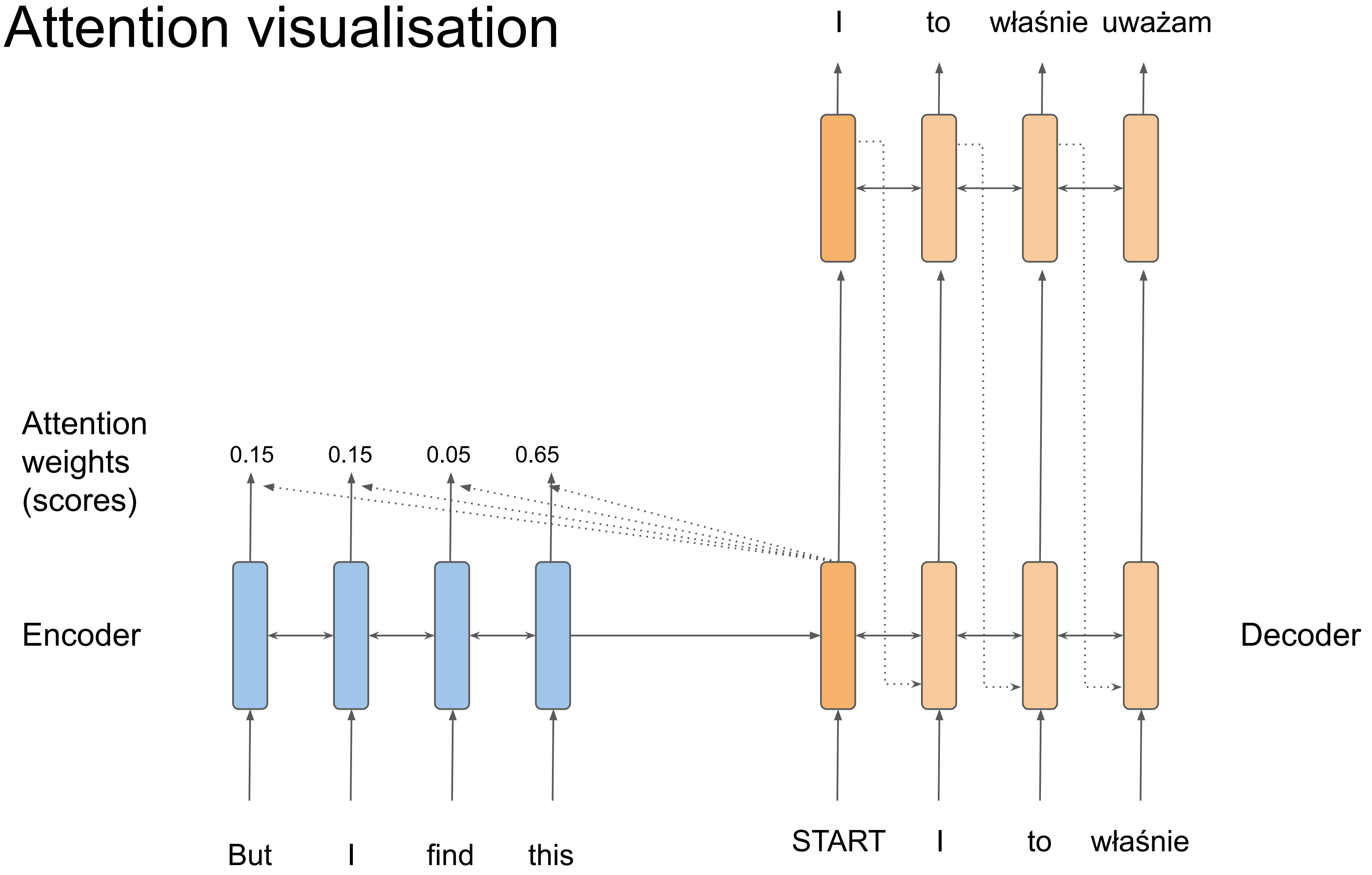
Attention provides a solution to the bottleneck problem

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

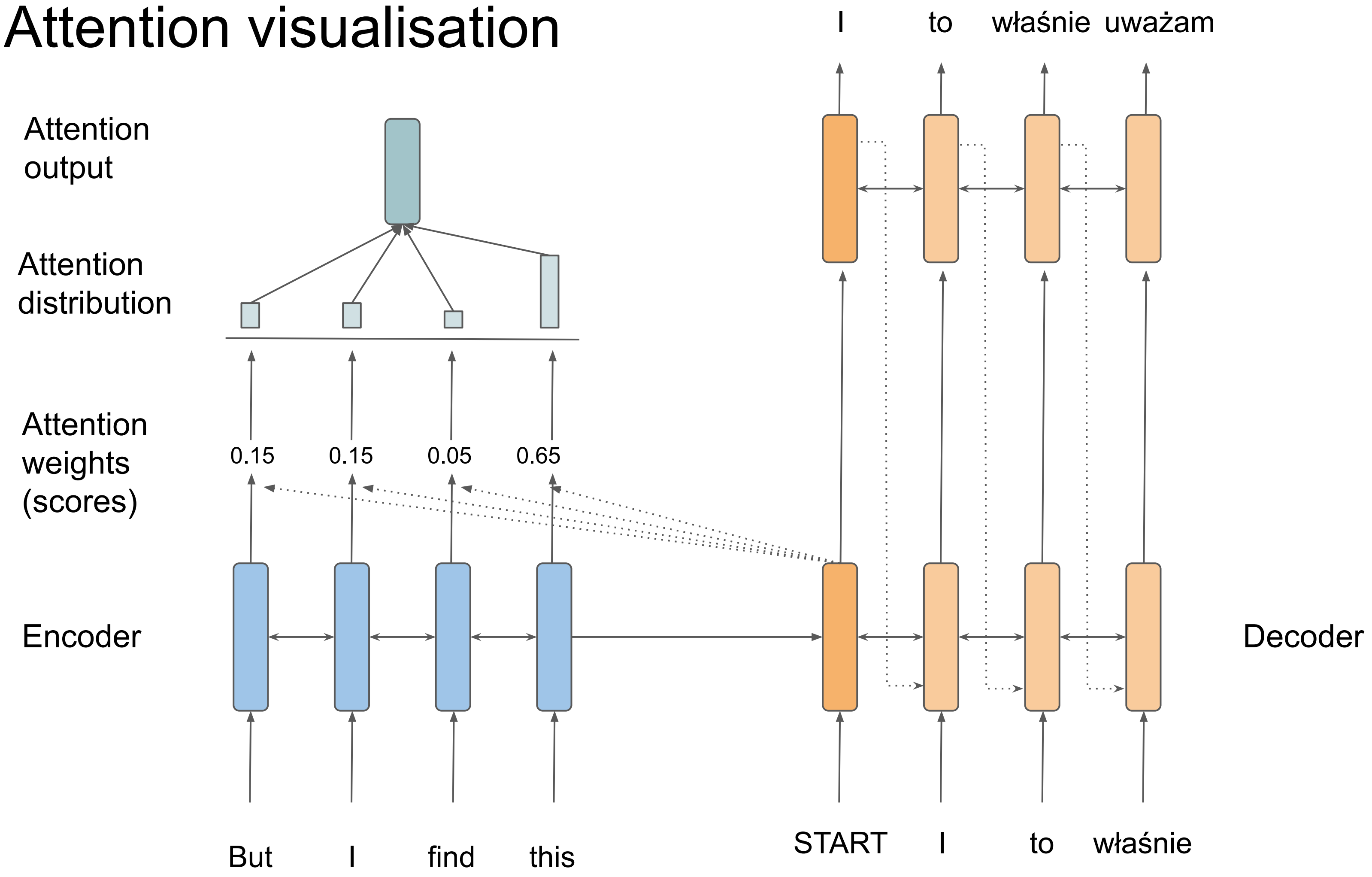
Attention visualisation



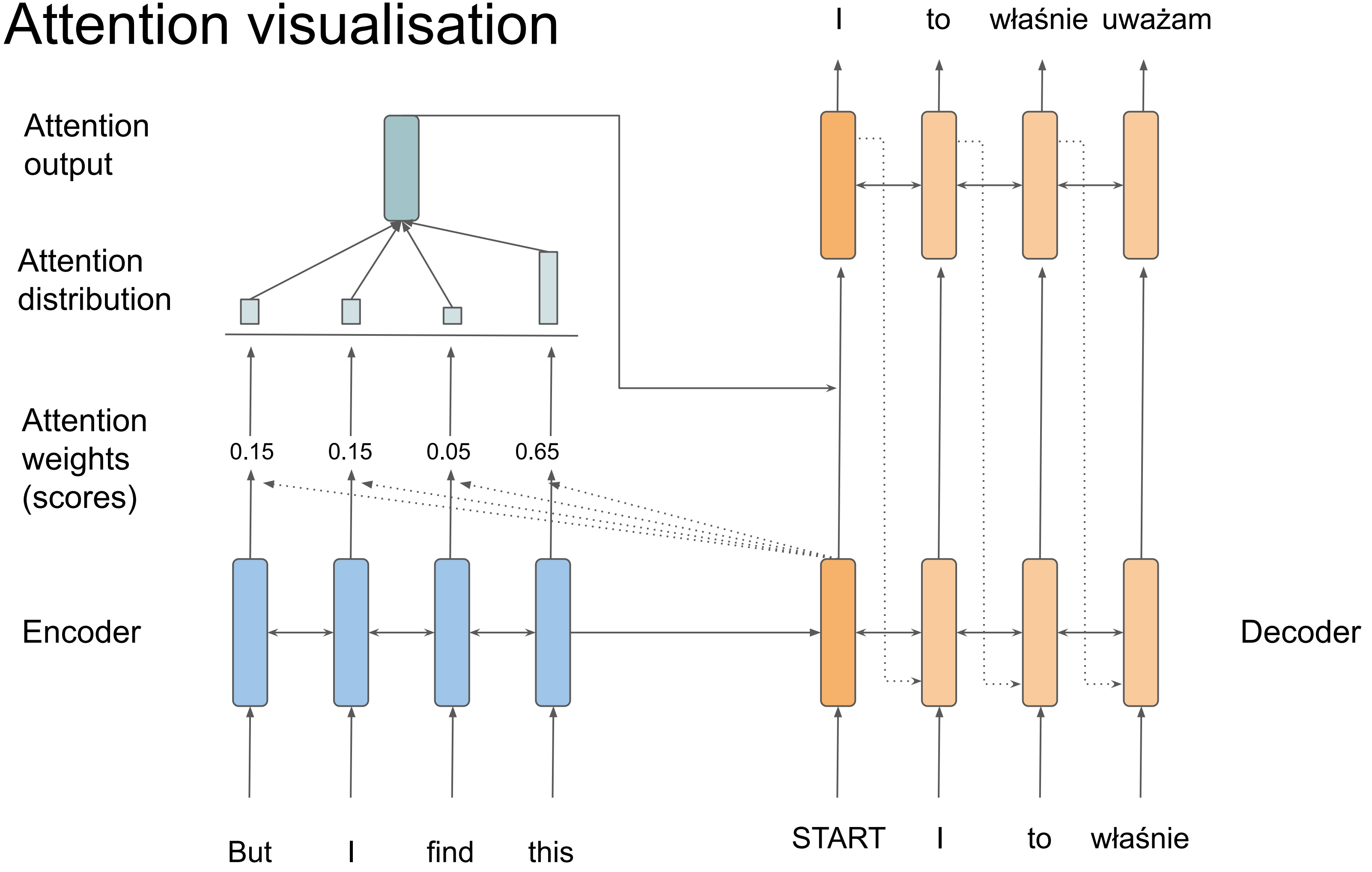
Attention visualisation



Attention visualisation



Attention visualisation



Attention in detail

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 detail

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 score $\mathbf{e}^t$ for this step:

$$\mathbf{e}^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

Attention in detail

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 score $\mathbf{e}^t$ for this step:

$$\mathbf{e}^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

We take softmax to get the attention distribution α^t for this step:

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

Attention in detail

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 score $\mathbf{e}^t$ for this step:

$$\mathbf{e}^t = [s_t^T h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

We take softmax to get the attention distribution α^t for this step:

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

We use alpha to take a weighted sum of the encoder hidden states to get the attention output a

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

Finally, we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model.

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

The power of attention

Attention **significantly** improves NMT performance

- It’s very useful to allow decoder to focus on certain parts of the source

Attention provides more **“human-like”** model of the MT process

- You can look back at the source sentence while translating, rather than needing to remember it all

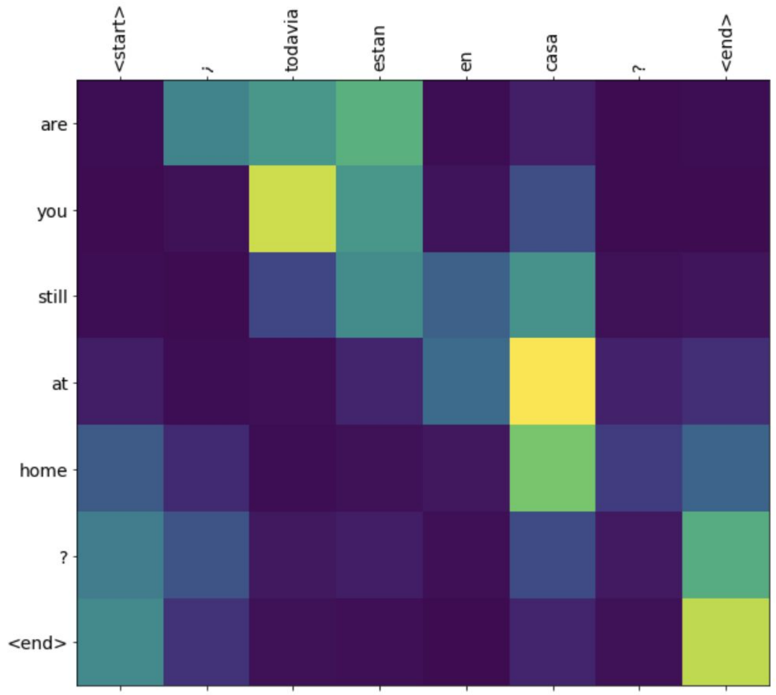
Attention **solves** the **bottleneck problem**

- Attention allows decoder to look directly at source

Attention provides some **interpretability**:

- By inspecting attention distribution, we can see what the decoder was **focusing on**
- We get soft **alignment for free**
- The network just learned alignment by itself

The power of attention [Google, 2020]



Attention is a powerhouse of Deep Learning technique

More general definition of attention:

Given a set of vectors **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)

Upshot:

- Attention has become the powerful, flexible, general way of “pointer and memory” manipulation in deep learning models

Literature

1. Bahdanau et al., 2014 - https://arxiv.org/abs/1409.0473
2. Luong et al., 2015 - https://arxiv.org/abs/1508.04025
3. Lample et al., 2017 - https://arxiv.org/abs/1711.00043
4. Artetxe et al., 2017 - https://arxiv.org/abs/1710.11041