Attention, Machine Translation

Natural Language Processing

(based on revision of Chris Manning Lectures)

Announcement

- TA announcements (if any)...

Suggested Readings

1. https://web.stanford.edu/~jurafsky/slp3/10.pdf (machine translation)
2. Statistical Machine Translation (book by Philipp Koehn)
3. Sequence to Sequence Learning with Neural Networks (original seq2seq NMT paper)
4. Sequence Transduction with Recurrent Neural Networks (early seq2seq speech recognition paper)
5. Neural Machine Translation by Jointly Learning to Align and Translate (original seq2seq+attention paper)
6. BLEU: a Method for Automatic Evaluation of Machine Translation (how to evaluate MT)
7. Attention and Augmented Recurrent Neural Networks (blog post overview)
8. Massive Exploration of Neural Machine Translation Architectures (practical advice for hyperparameter choices)
9. Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models (combining character embedding to help out-of-vocabulary problems)
10. Revisiting Character-Based Neural Machine Translation with Capacity and Compression (another early paper showing character embedding seems to outperform word-based embeddings

Pre-Neural Machine Translation

Machine Translation

Machine Translation (MT) is the task of translating a sentence x from one language (the **source language**) to a sentence y in another language (the **target language**).

x: *deep learning macht Spaß* (German)

y: *deep learning is fun* (English)

1990s - 2010s: Statistical Machine Translation

- Learn a probabilistic model from data :

$$\underset{y}{\operatorname{argmax}}\, P(y|x)$$

- Uses Bayes Rule to break this down into two components to be learned:

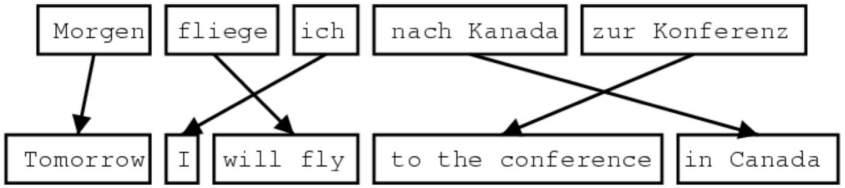
$$\underset{y}{\operatorname{argmax}}\, P(x|y)P(y)$$

Translation Model: Models how to translate to x given y (*fidelity*). Learnt from parallel corpus (i.e., German + English).

English Language Model: Models how to write good English (*fluency*). Learnt from monolingual data.

1990s - 2010s: Statistical Machine Translation

- How to learn $P(x|y)$?
- First, need large amount of parallel data (i.e., French and English)
- Define alignment a, i.e., a word-level correspondence between source sentence x and target sentence y



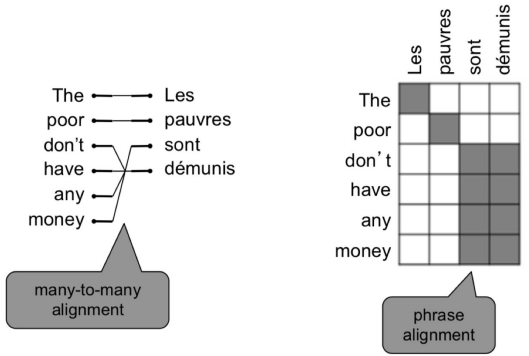
- Redefine the probability which includes the alignment

$$P(x, a|y)$$

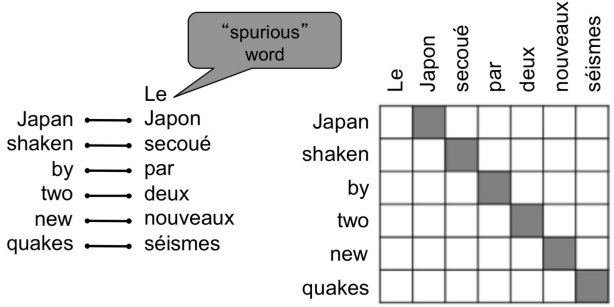
What is alignment?

- **Alignment** is the **correspondence between particular words** in the translated sentence pair
  - Note: some words have no counterpart, some have many-to-one relationships,

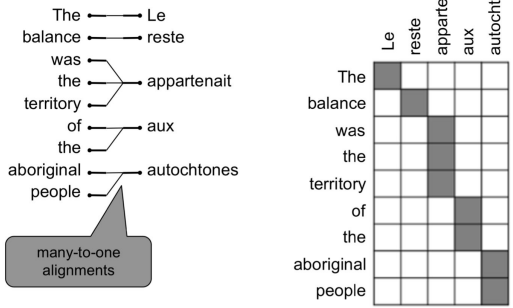
many-to-many



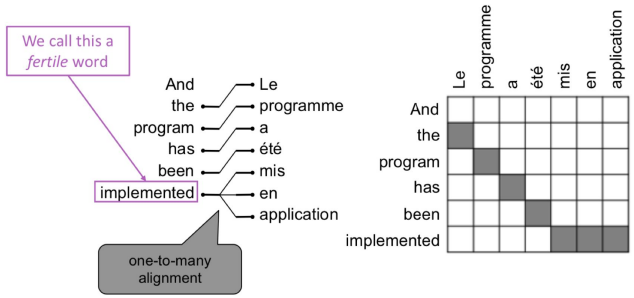
No counterparts



many-to-one



one-to-many



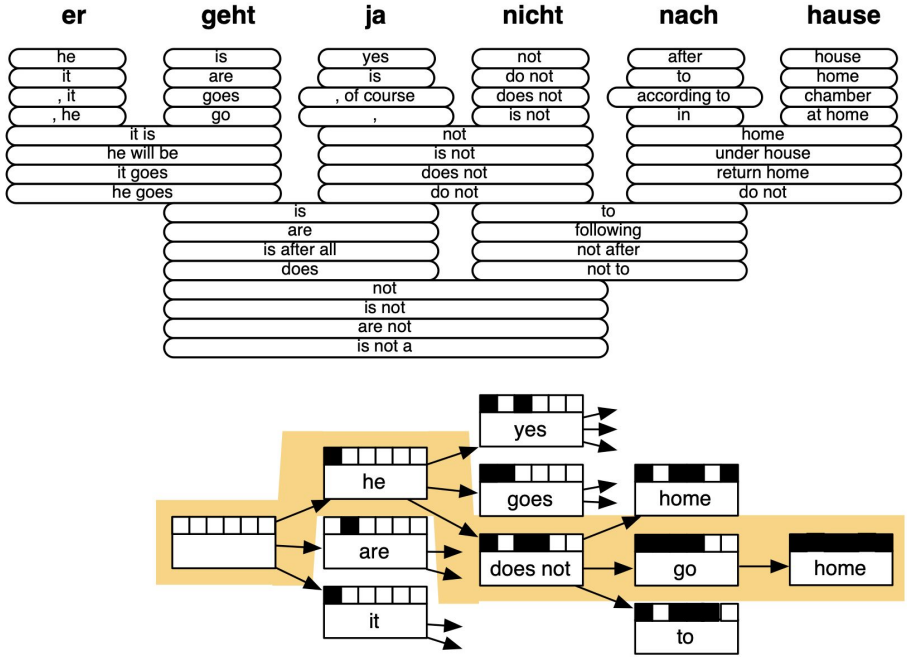
Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. http://www.aclweb.org/anthology/J93-2003

Learning alignment

- We learn $P(x, a|y)$ as a combination of many factors:
  - Probability of particular words aligning
  - Probability of particular words on a position
  - Probability of particular words having a particular fertility
  - Etc.
- Alignments a are discrete, not probabilistic
  - Require the use of special learning algorithms (like Expectation-Maximization) for learning the parameters of distributions

Decoding for SMT

- We could enumerate every possible y and calculate the probability -> Too expensive!
- Answer: Use some dynamic programming (e.g., **viterbi**) to figure out **optimal paths (this is actually very slow!)**



Statistical Machine Translation, Chapter 6, Koehn, 2009.
https://www.cambridge.org/core/books/statistical-machine-translation/94EADF9F680558E13BE759997553CDE5

1990s-2010s: SMT

- SMT was a **huge research field**
- The best systems were extremely complex
  - Hundreds of important details we haven't mentioned here
  - Systems had many **separately-designed subcomponents**
  - **Lots of feature engineering**
    - Need to design features to capture particular language phenomena
  - Require compiling and maintaining **extra resources**
    - Like tables of equivalent phrases
  - Lots of **human effort** to maintain
    - Repeated effort for each language pair

Neural Machine Translation

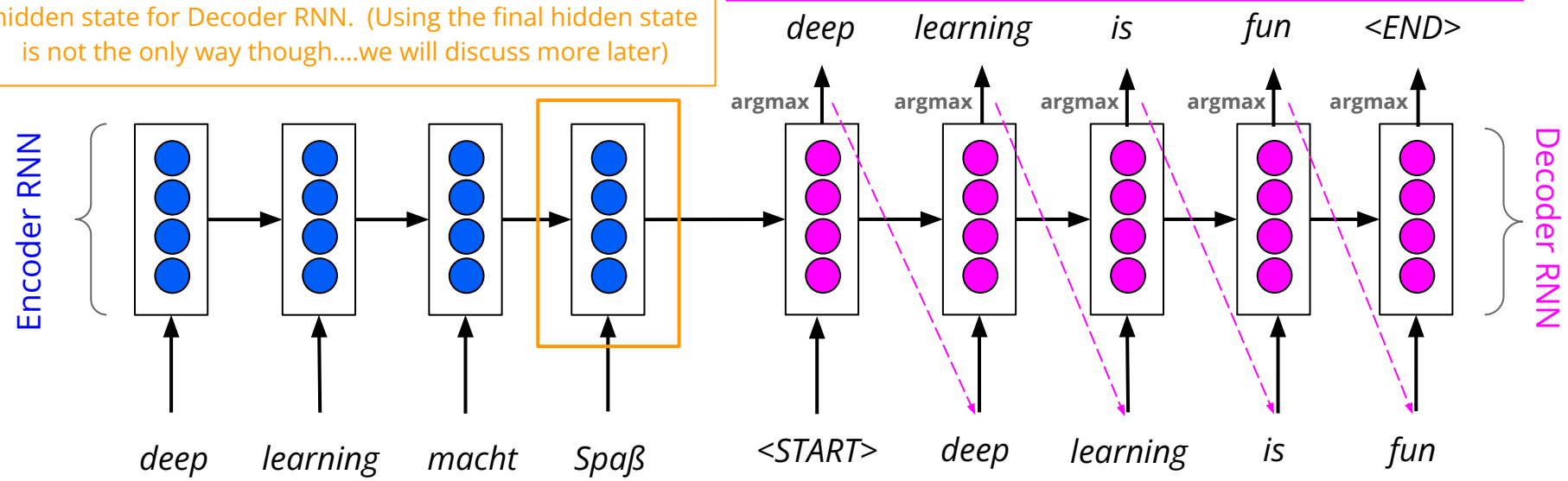
Neural Machine Translation

- **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** RNNs

Sequence to Sequence Model

Encoding of the whole source sentence. Input the final hidden state for Decoder RNN. (Using the final hidden state is not the only way though....we will discuss more later)

Note that this diagram is showing the generation (testing) mode. The training mode will be in "teacher forcing" mode instead.



Encoder RNN produces an encoding of the source sentence.

Decoder RNN is a LM that generates target sentence, conditioned on ending. It does NOT need to be same length as the encoder. It takes in the <START> token, and will stop when it meets the <END> token or the specified max length.

Sequence-to-sequence is versatile

- Sequence-to-sequence is useful for **more than just MT**

- Many NLP tasks can be phrased as seq2seq
  - **Summarization** (long text -> short text)
  - **Dialogue** (previous utterances -> next utterances)
  - **Parsing** (input text -> output parse as sequence)
  - **Code generation** (natural language -> Python code)

Training a NMT system "Teacher forcing"

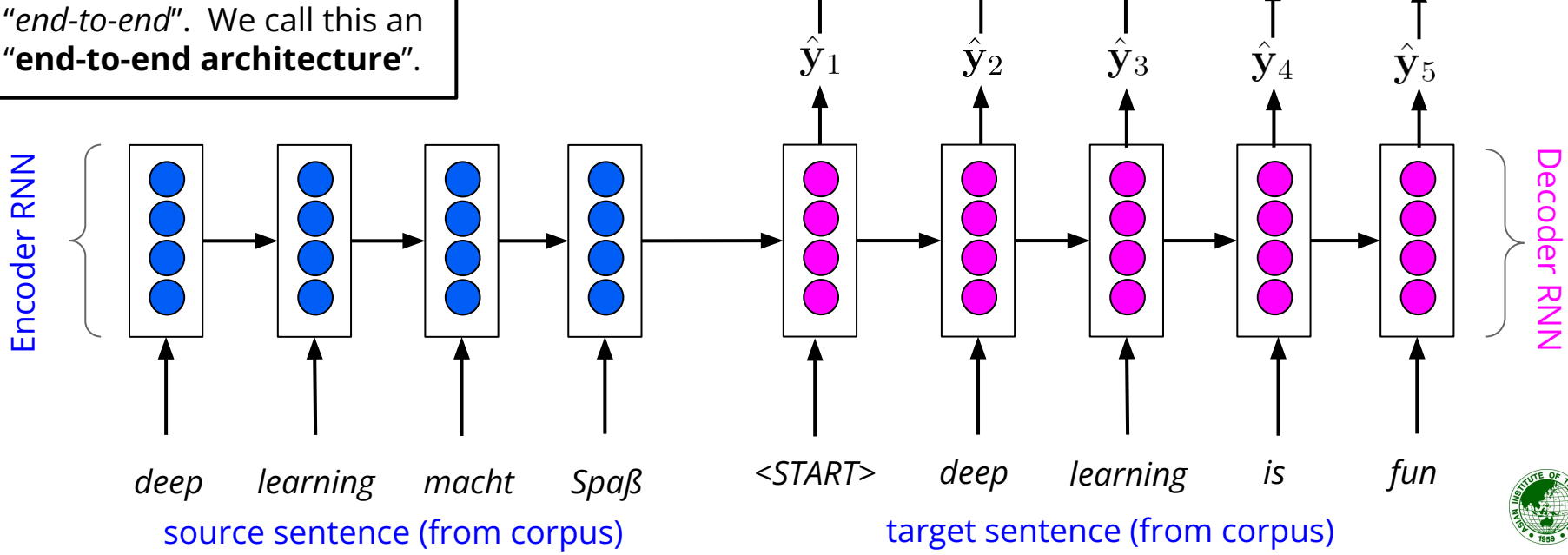
What is cool is the Seq2seq is optimized as a **single system**. Backprop operates "*end-to-end*". We call this an "**end-to-end architecture**".

$$J = \frac{1}{T}\sum_{t=1}^{T} J_t = J_1 + J_2 + J_3 + J_4 + J_5$$

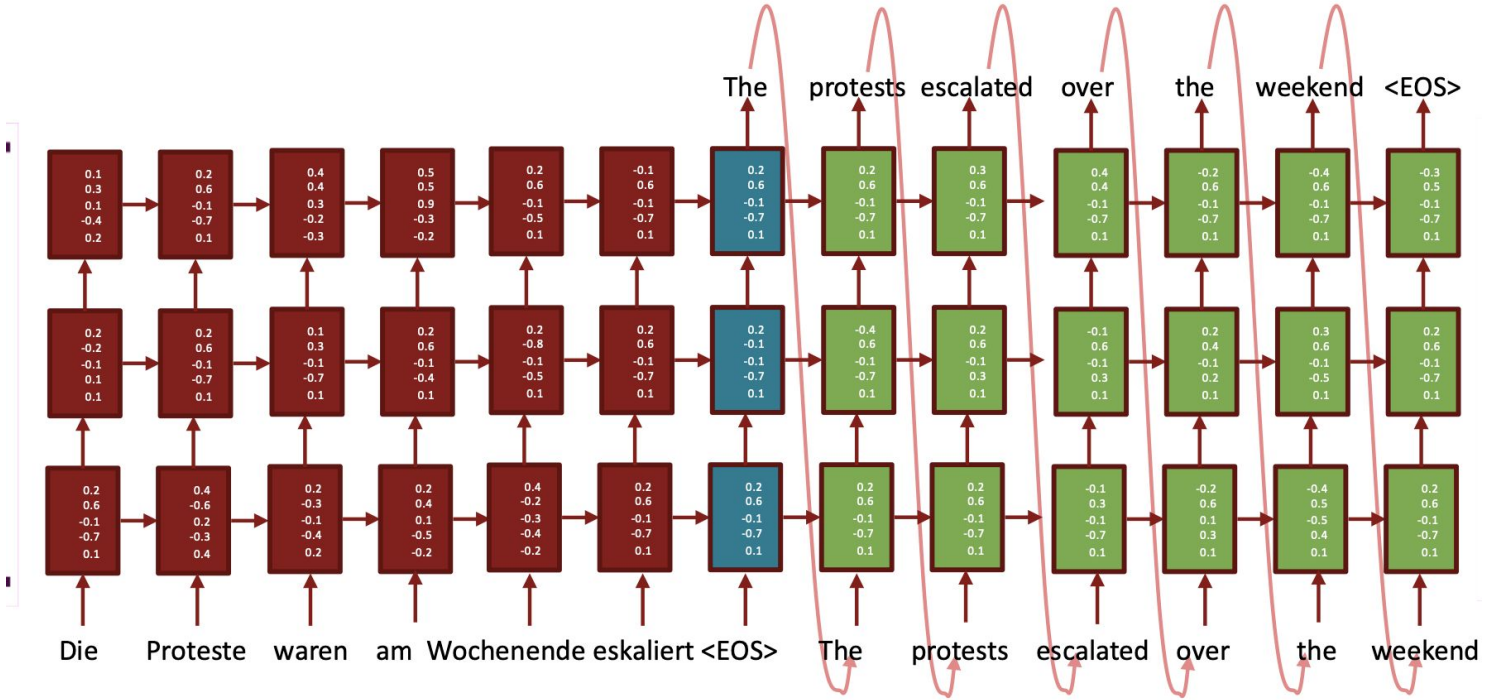
=negative log prob of "deep"

=negative log prob of "fun"

=negative log prob of "<END>"



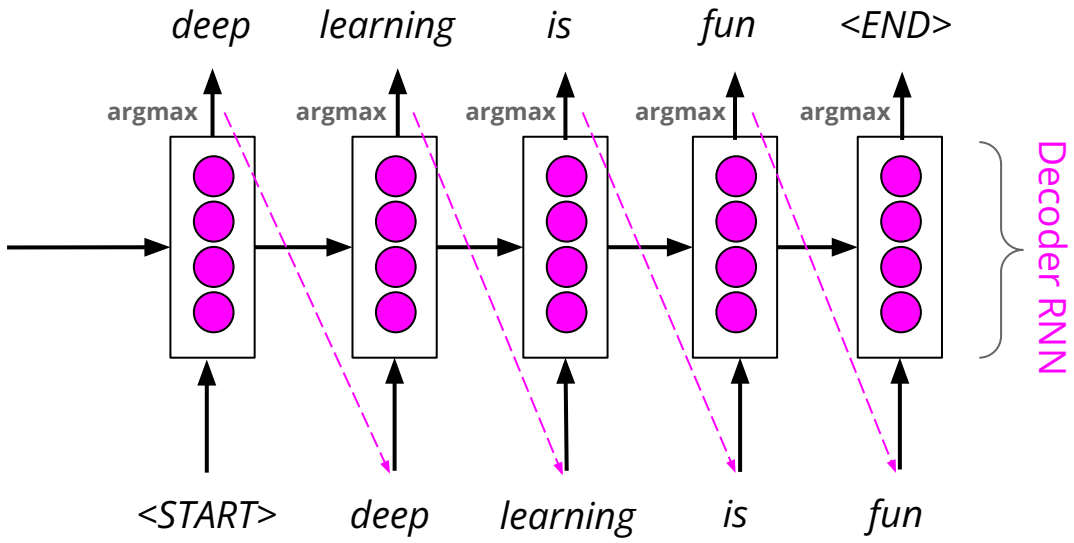
Multilayer (Stacked) NMT



Sequence to Sequence Learning with Neural Networks, Sutskever et al. 2014, https://arxiv.org/abs/1409.3215

Greedy decoding

- On our earlier slide, our decoding is of greedy, **taking the highest probability word (i.e., argmax)**. Any potential problem with that?



Beam search

- **Beam search**: on each step of decoding, keep track of the *k* most probable partial translations (which we call **hypothesis**)
  - *k* is the beam size (in practice around 5 to 10)

- A hypothesis $y_1, \cdots, y_t$ has a **score** which is its log prob

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

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

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

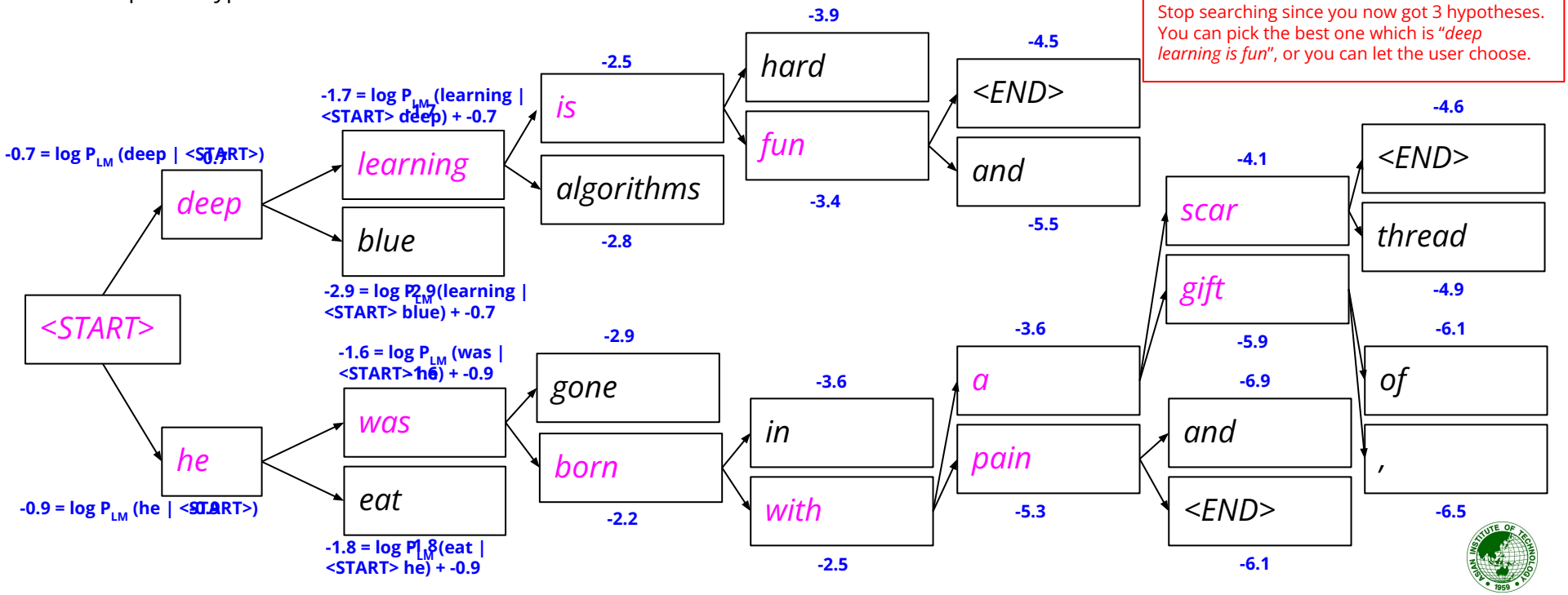
Beam search - Example

(only for demo purposes - no real data)

Note that k is not the same as n. k refers to the number of parallel search, while n is a stopping criterion when the number of n hypotheses is reached

Beam size = k = 2 (numbers of parallel search). Blue numbers = $\text{score}(y_1, \cdots, y_t) = \log P_{\text{LM}}(y_1, \cdots, y_t | x)$

no.of required hypo = n = 3. Pink refers to the search words



Beam search - stopping criterion

- In **greedy decoding**, usually we decode until the model produces an <END> token For example: <START>deep learning is fun<END>

- In **beam search decoding**, different hypotheses may produce <END> tokens on different timesteps
  a. When a hypothesis produces <END>,that hypothesis is complete.
  b. Place it aside and continue exploring other hypotheses via beam search

- Usually we continue beam search **until**:
  a. We reach time step T (where T is some predefined cutoff),or
  b. we have at least n completed hypotheses (where n is pre-defined cutoff)

- Potential problem: longer sentences tend to have lower scores
  i. Fix: normalize by length (divide by length)

NMT vs. SMT

NMT Pros:

- Better performance
  a. More **fluent**
  b. Better use of **context**
  c. Better use of **phrase similarities**
- A single network that does all
  a. Less human engineering effort

NMT Cons:

- Less **interpretable**
  a. Hard to debug
- Difficult to **control**
  a. Difficult to impose policy or rules

NMT: perhaps the biggest success of NLP?

NMT went from **fringe research attempt** in 2014 to the **leading standard method** in 2016

- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT and by **2018** everyone switches (Microsoft, Tencent, Baidu, Facebook, etc.)

It is kind of sad that SMT systems were built by **hundreds** of engineers over **many** years, but were outperformed by NMT trained by **small** number of engineers in a **few** months :(

Is NMT solved?

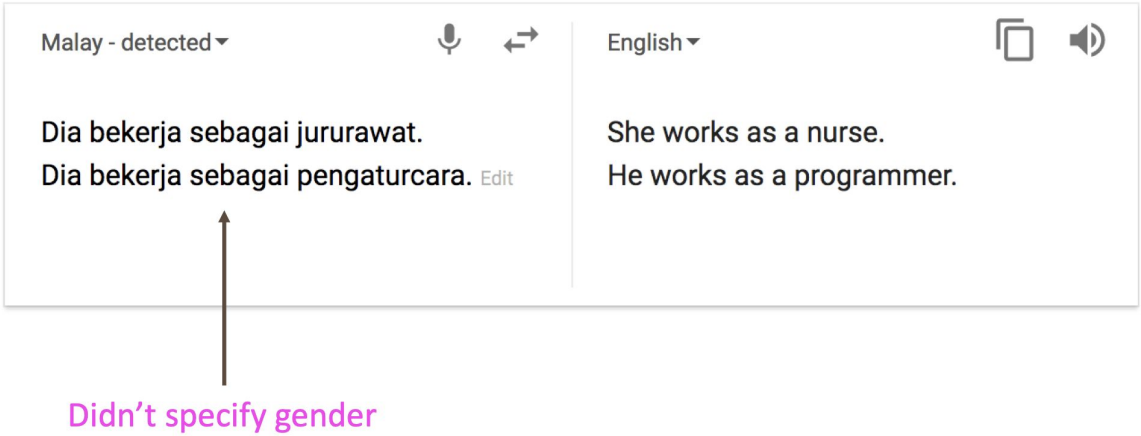
Many difficulties remain:

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

"Has AI surpassed humans at translation? Not even close!" https://www.skynettoday.com/editorials/state_of_nmt

Is NMT solved?

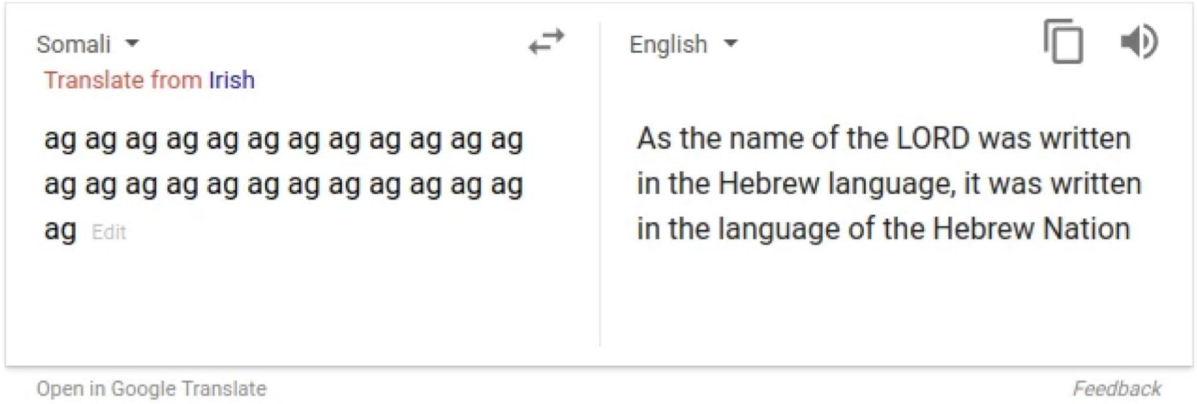
Gender bias still exists (corpus bias)



https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c

Is NMT solved?

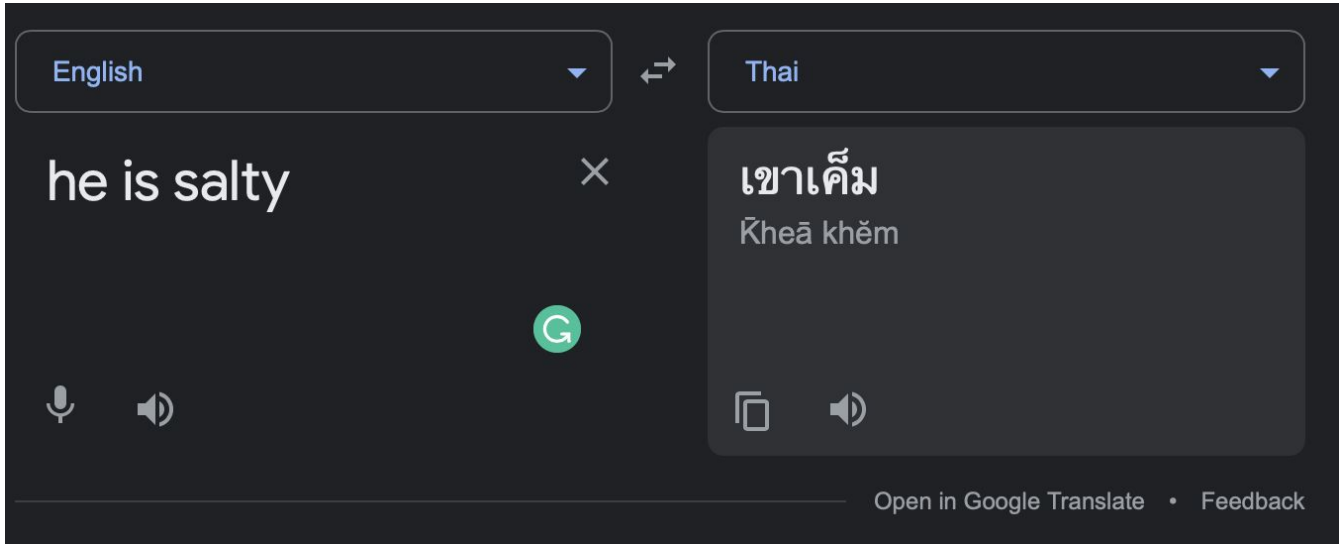
Sometimes strange stuff happens (low resource language)



https://www.skynettoday.com/briefs/google-nmt-prophecies

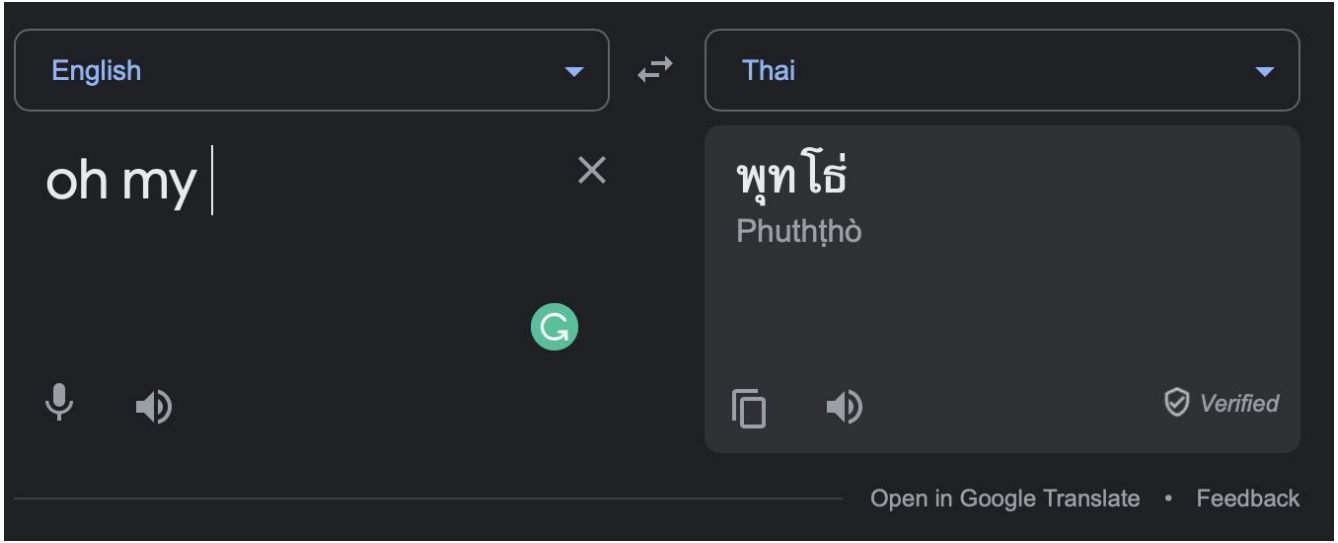
Is NMT solved?

Humor is difficult!



Is NMT solved?

Probably translated again from (“oh my god”)



NMT Evaluation

How do we evaluate MT?

BLEU (Bilingual Evaluation Study)

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

- BLEU is useful but **imperfect**
  a. There are **MANY** valid ways to translate a sentence
  b. So a **good** translation can get a **poor** BLEU score because it has low n-gram overlap with the human translation

BLEU: a Method for Automatic Evaluation of Machine Translation, Papineni et al, 2002. http://aclweb.org/anthology/P02-1040

NMT with Attention

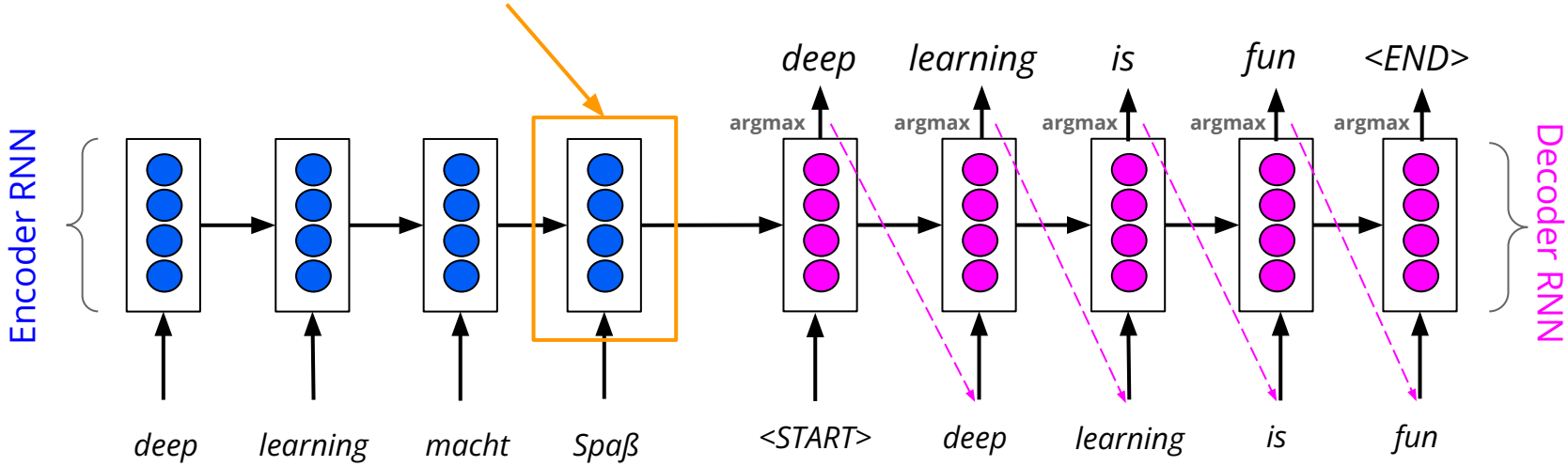
NMT with attention

- In 2021: NMT research continues to **strive**
  - Researchers have **many**, **many improvements** to the “vanilla” seq2seq NMT system we’ve just presented

- One huge improvement so integral that it is the new vanilla: **Attention**

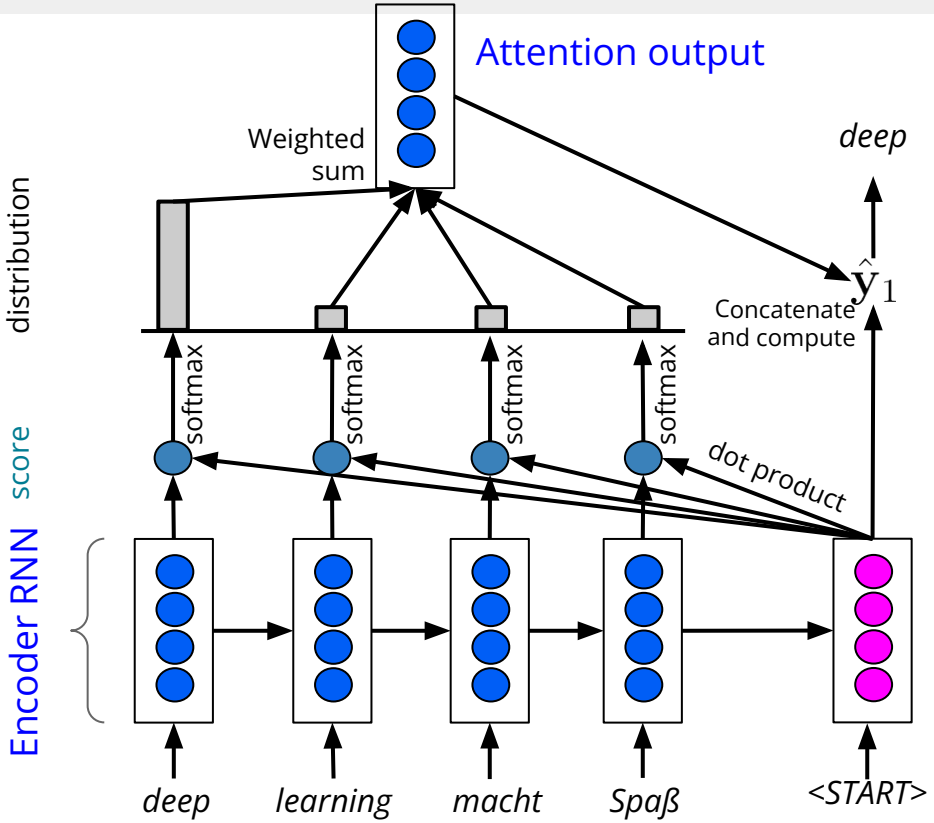
NMT with attention

Using the **final hidden state** can be risky, as we need to assume that this hidden state captures ***all*** information



NMT with attention

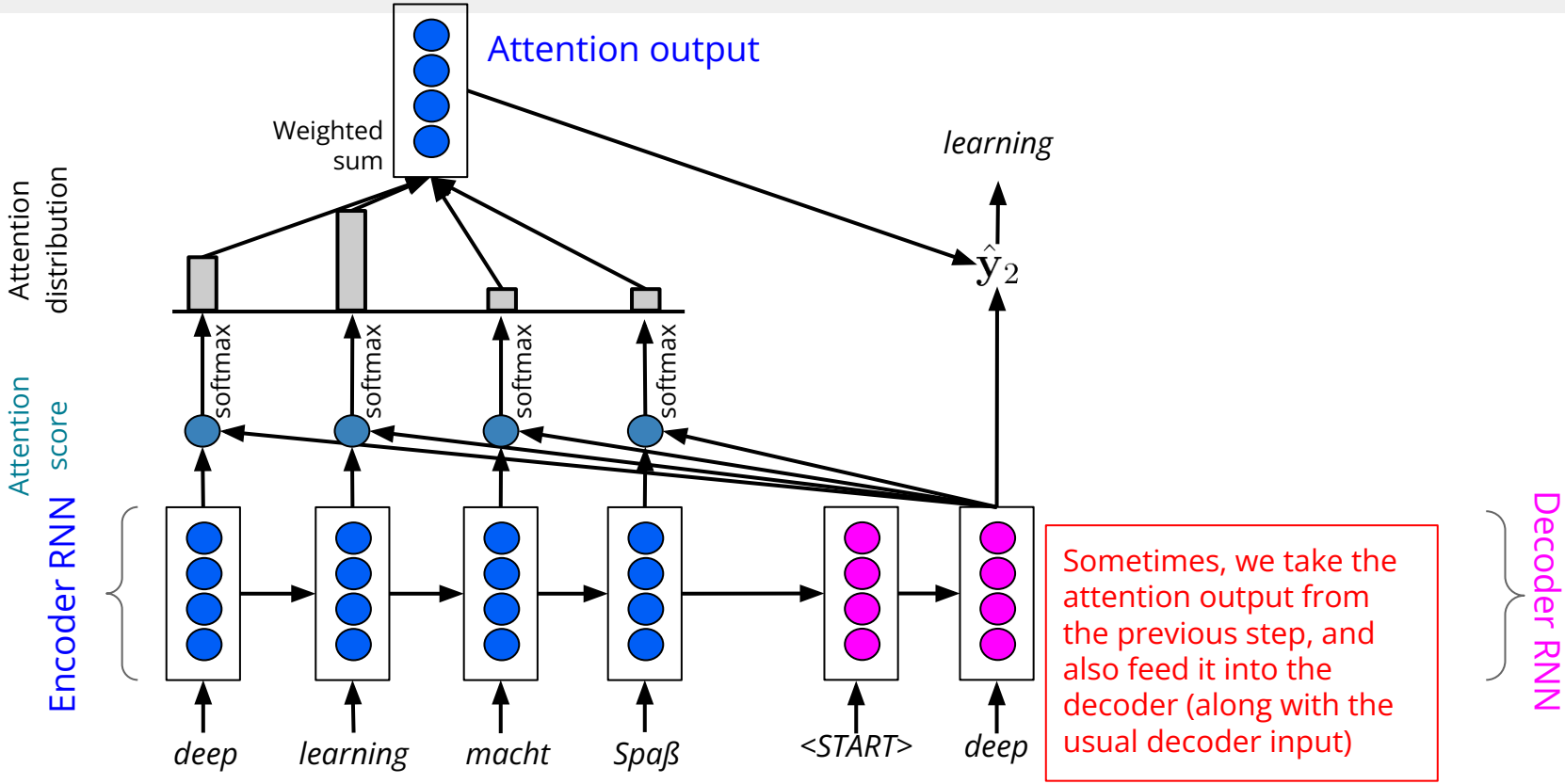
Idea: use **direct connection** to focus on **a particular part** of the **source sequence**



The weighted sum is like a selective summary of the information

NMT with attention

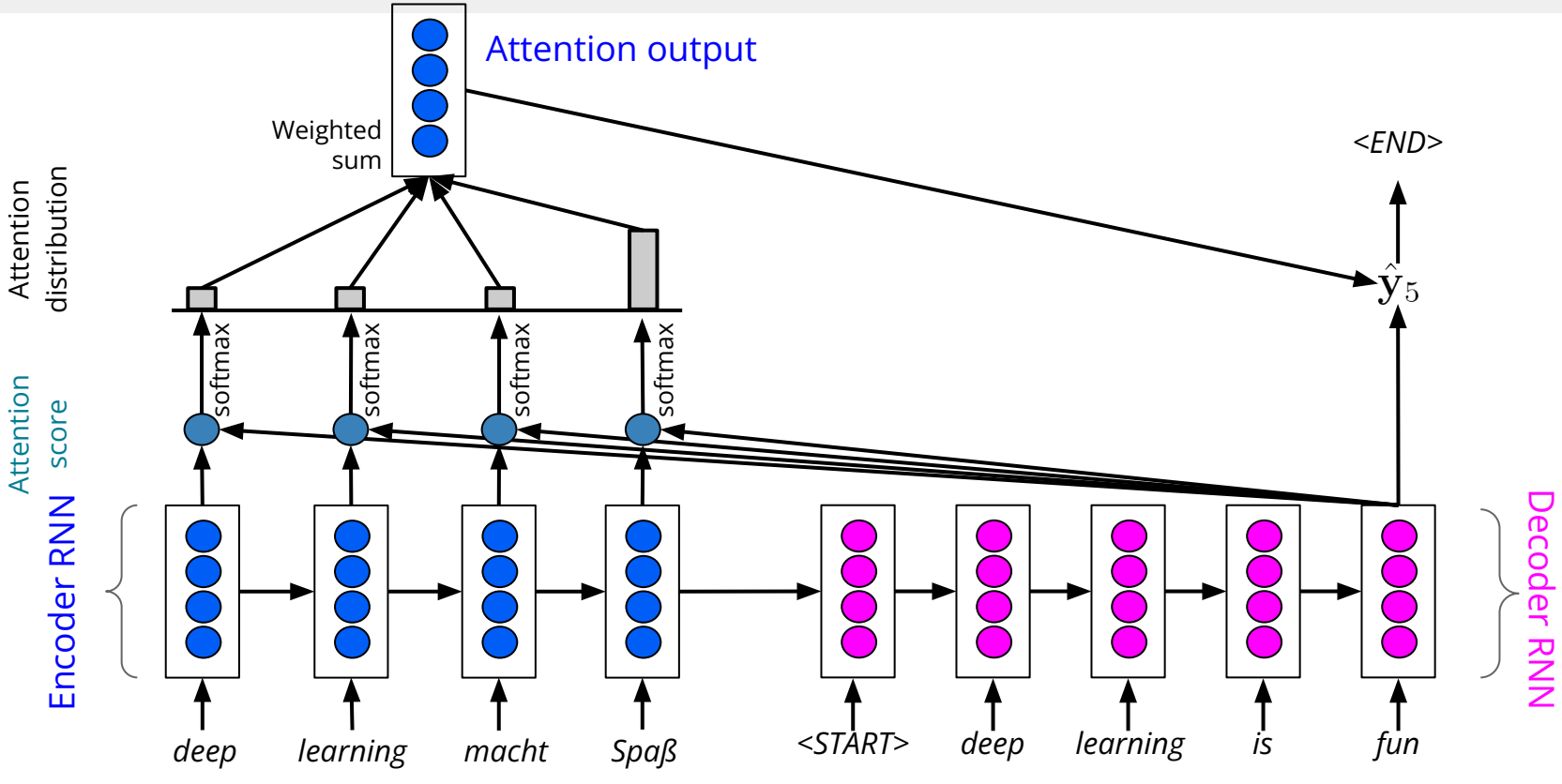
Idea: use **direct connection** to focus on **a particular part** of the **source sequence**



Sometimes, we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input)

NMT with attention

Idea: use **direct connection** to focus on **a particular part** of the **source sequence**



Attention - formally

- We have encoder hidden states $\mathbf{h}_1, \cdots, \mathbf{h}_N \in \mathbb{R}^h$
- On timestep t, we have decoder hidden state $\mathbf{s}_t \in \mathbb{R}^h$
- We get the attention scores $\mathbf{e}_t = [\mathbf{s}_t^T \mathbf{h_1}, \cdots, \mathbf{s}_t^T \mathbf{h_N}] \in \mathbb{R}^N$
- We take softmax to get the attention distribution $\boldsymbol{\alpha}_t$ for this step (this is a probability distribution and sums to 1) $\boldsymbol{\alpha}_t = \mathrm{softmax}(\mathbf{e}_t) \in \mathbb{R}^N$
- We use $\boldsymbol{\alpha}_t$ to take a weighted sum of the encoder hidden states to get the attention (context) output $\mathbf{c}_t$

$$\mathbf{c}_t = \sum_{i=1}^{N} \boldsymbol{\alpha}_i^t \mathbf{h}_i \in \mathbb{R}^h$$

- Finally, we concatenate the attention output $\mathbf{c}_t$ with the decoder hidden state $\mathbf{s}_t$ and proceed as in the non-attention seq2seq $[\mathbf{c}_t; \mathbf{s}_t] \in \mathbb{R}^{2h}$

Attention - variants

Most variants differ in how they compute the energy. Assume the followings:

$$e \in \mathbb{R}; \quad \mathbf{h}_1, \cdots, \mathbf{h}_N \in \mathbb{R}^{d_1}; \quad \mathbf{s} \in \mathbb{R}^{d_2}$$

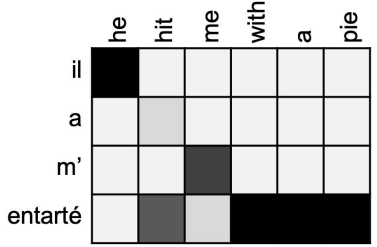
- **General** attention $e_i = \mathbf{s}^T \mathbf{h_i} \in \mathbb{R}; \quad d_1 = d_2$
- **Multiplicative** attention $e_i = \mathbf{s}^T W \mathbf{h_i} \in \mathbb{R}; \quad \mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$
- **Additive** attention (d_3 is a hyperparameter)

$$e_i = \mathbf{v}^t \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}) \in \mathbb{R}$$
$$\mathbf{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \mathbf{W}_2 \in \mathbb{R}^{d_3 \times d_2}, \mathbf{v} \in \mathbb{R}^{d_3}$$

Much more! in Attentive Surveys in Attention Models, Chaudhari et al. 2019, https://arxiv.org/pdf/1904.02874.pdf

Attention is state-of-the-art

- Attention significantly **improves NMT performance** as well as other NLP tasks (as we shall see in Transformers)
- Attention is **not only for seq2seq**. It's a general deep learning technique that can be applied almost on any architectures or tasks (e.g., classification tasks, or even CNN!)
- Attention **helps with vanishing gradient problem**
  - Provides shortcut to faraway state
- Attention provides some **interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get **alignment** for free!
  - The is cool, because we never explicitly trained an alignment system



Summary

- **Seq2seq** - powerful for many tasks including **NMT**, **text summarization**, dialogue, etc.
  - Many variants!

- **Attention** is a way to focus on particular parts of the input
  - Improves seq2seq a lot!
  - Many **variants**
  - Can also apply to **classification purpose** - imagine you have no decoder