RNN: Sequence to Sequence

Pawan Goyal

CSE, IIT Kharagpur

February 6th, 2023

What we have seen?

- Input sequence to a fixed-sized vector
- Input sequence to an output sequence of the same length

What we have seen?

- Input sequence to a fixed-sized vector
- Input sequence to an output sequence of the same length

Any other constraint?

Mapping input sequence to an output sequence, not necessarily of the same length
machine translation, question answering, chatbots, summarization, ...

What is 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: *L'homme est né libre, et partout il est dans les fers*

↓

y: *Man is born free, but everywhere he is in chains*

\- Rousseau

Sequence-to-sequence architecture

Also known as encoder-decoder architecture

- Input sequence $X = (x^{(1)}, \ldots, x^{(n_x)})$
- Output sequence $Y = (y^{(1)}, \ldots, y^{(n_y)})$

Sequence-to-sequence architecture

Also known as encoder-decoder architecture

- Input sequence $X = (x^{(1)}, \ldots, x^{(n_x)})$
- Output sequence $Y = (y^{(1)}, \ldots, y^{(n_y)})$
- Encoder (reader/input) RNN: Emits the context C, a vector summarizing the input sequence, usually as a simple function of its final hidden state

Sequence-to-sequence architecture

Also known as encoder-decoder architecture

- Input sequence $X = (x^{(1)}, \ldots, x^{(n_x)})$
- Output sequence $Y = (y^{(1)}, \ldots, y^{(n_y)})$
- Encoder (reader/input) RNN: Emits the context C, a vector summarizing the input sequence, usually as a simple function of its final hidden state
- Decoder (writer/output) RNN: Is conditioned on the context C to generate the output sequence.

Sequence-to-sequence architecture

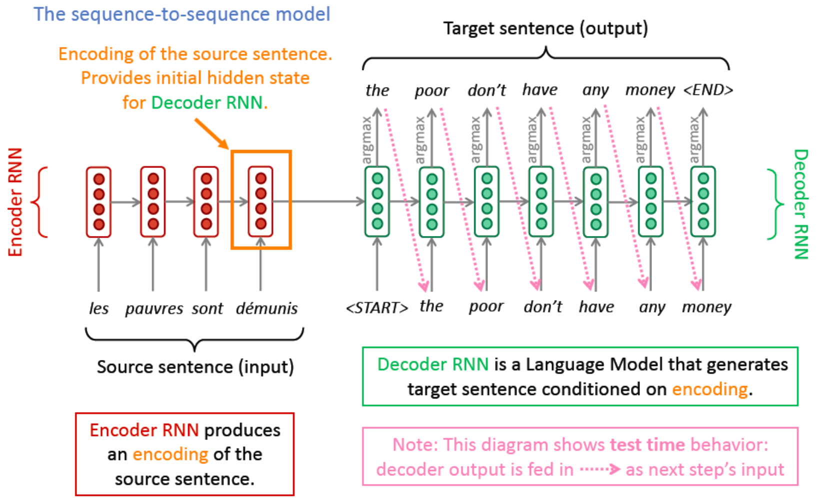
Also known as encoder-decoder architecture

- Input sequence $X = (x^{(1)}, \ldots, x^{(n_x)})$
- Output sequence $Y = (y^{(1)}, \ldots, y^{(n_y)})$
- Encoder (reader/input) RNN: Emits the context C, a vector summarizing the input sequence, usually as a simple function of its final hidden state
- Decoder (writer/output) RNN: Is conditioned on the context C to generate the output sequence.

What is the innovation?

The lengths n_x and n_y can vary from each other

Sequence to Sequence Models for Machine Translation



Sequence to sequence is versatile

Many NLP tasks can be phrased as sequence-to-sequence

- Summarization (long text $\rightarrow$ short text)
- Dialogue (previous utterances $\rightarrow$ next utterance)

Neural Machine Translation (NMT)

An example of a Conditional Language Model

- **Language Model** because the decoder is predicting the next word of the target sentence y
- **Conditional** because the predictions are also conditioned on the sentence x

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

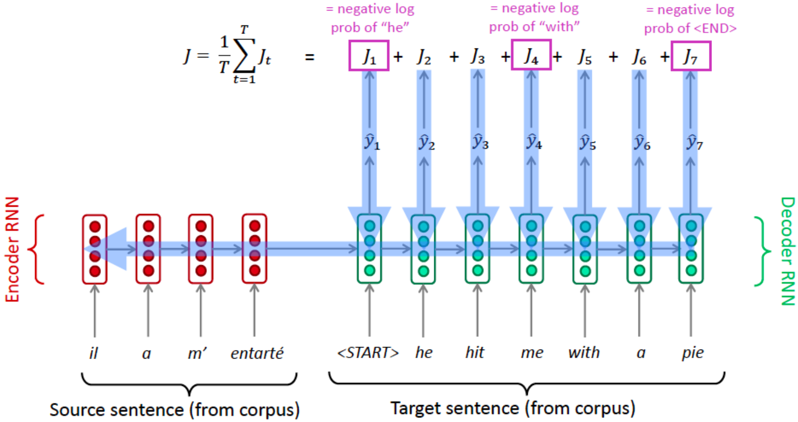
$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}_{\substack{\text{Probability of next target word, given} \\ \text{target words so far and source sentence } x}}$$

How to train an NMT system?

Get a big parallel corpus ...

Training an NMT system

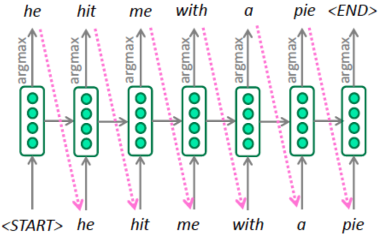
$$J = \frac{1}{T}\sum_{t=1}^{T} J_t \quad = \quad \boxed{J_1} + J_2 + J_3 + \boxed{J_4} + J_5 + J_6 + \boxed{J_7}$$



Seq2seq is optimized as a **single system.**
Backpropagation operates *"end-to-end"*.

Greedy decoding

One possibility is to generate (decode) the target sentence by taking argmax on each step of the decoder



This is greedy decoding

Problems with this method?

Problems with greedy decoding

Greedy decoding has no way to undo decisions!

- Input: *il a m'entarté* *(he hit me with a pie)*
- → *he* ____
- → *he hit* ____
- → *he hit a* ____ (whoops! no going back now...)

How to fix this?

Exhaustive search decoding

Ideally we want to find a (length T) translation y that maximizes
$P(y|x) = P(y_1|x)P(y_2|y_1,x)P(y_3|y_2,y_1,x)\dots P(y_T|y_1,\dots,y_{T-1},x)$
$= \prod_{t=1}^{T} P(y_t|y_1,\dots,y_{t-1},x)$

We could try to compute all possible sequences y exhaustively

- This means that on each step t of the decoder, we are tracking V^t possible partial translations, where V is vocab size
- This $O(V^T)$ complexity is far too expensive!

Beam search decoding

Core Idea

- On each step of the decoder, keep track of the k most probable partial translations (hypotheses)
- k is the beam size (in practice around 5 to 10)

Beam search decoding

Core Idea

- On each step of the decoder, keep track of the k most probable partial translations (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:
$score(y_1, \ldots, y_t) = logP_{LM}(y_1, \ldots, y_t|x) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

- Scores are all negative, higher 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

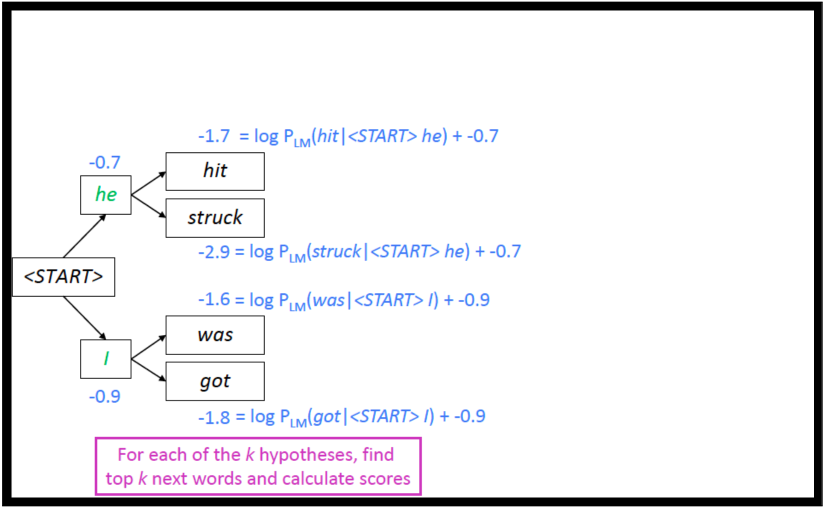
Beam size = k =2. Blue numbers =
$score(y_1,\ldots,y_t) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1,\ldots,y_{i-1},x)$

Beam search decoding: example

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

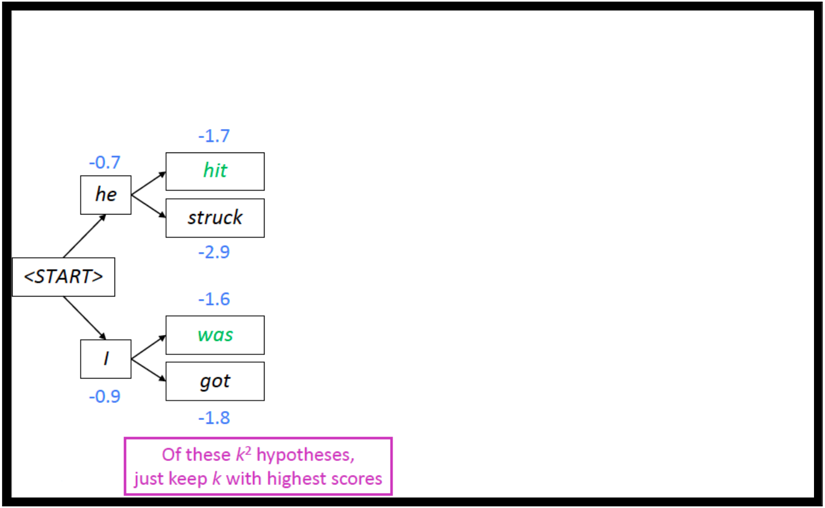
Beam search decoding: example

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



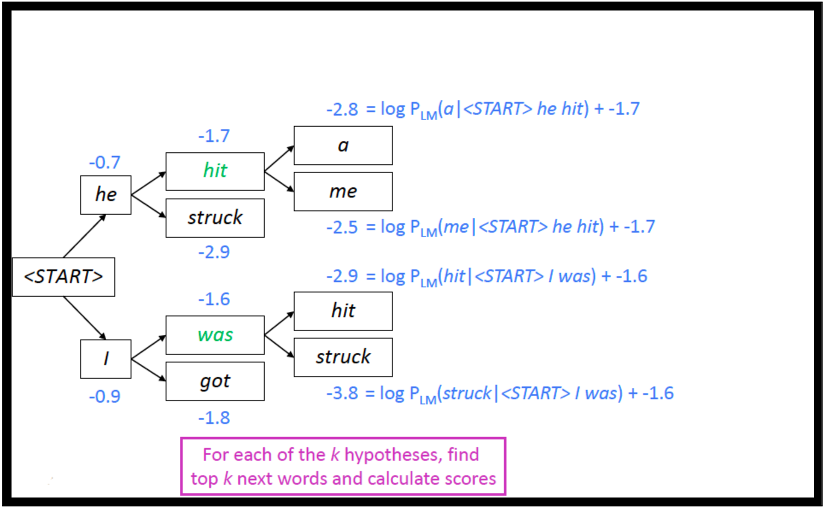
Beam search decoding: example

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



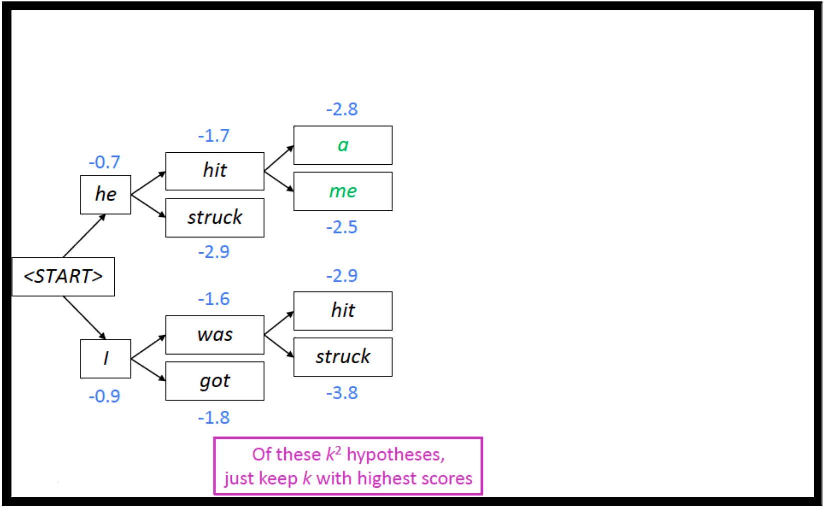
Beam search decoding: example

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

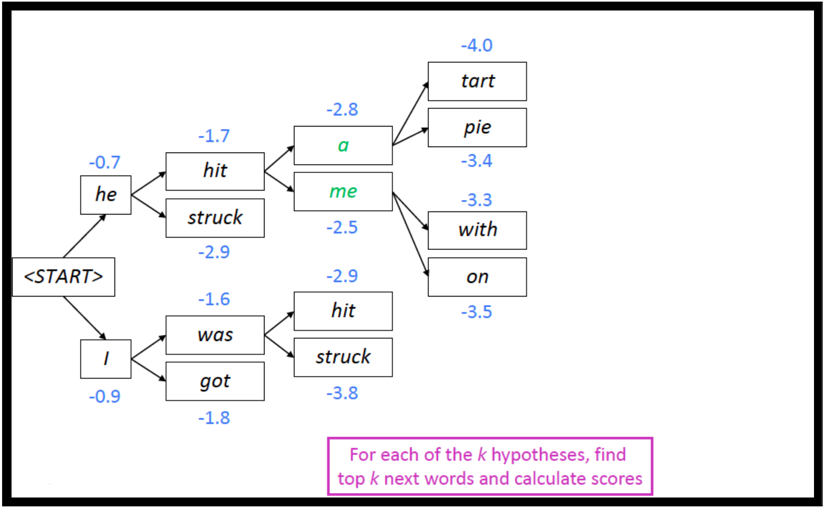


Beam search decoding: example

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



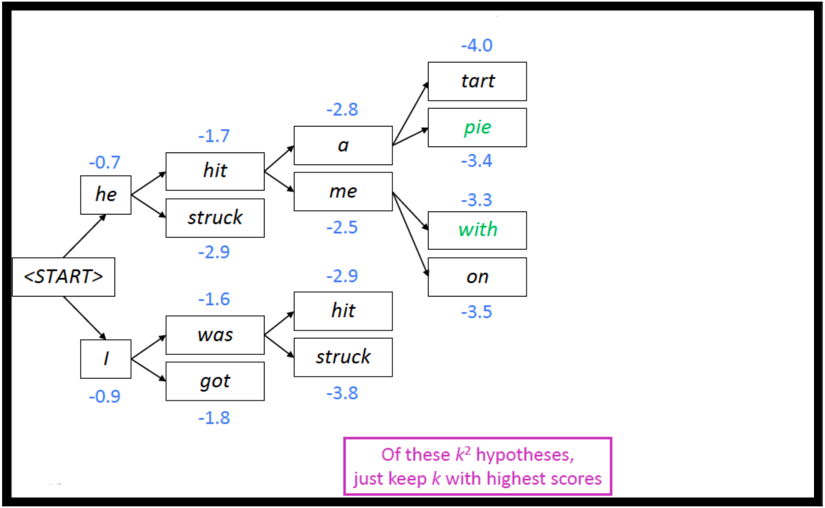
Beam size = k =2. Blue numbers = $score(y_1,\ldots,y_t) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1,\ldots,y_{i-1},x)$



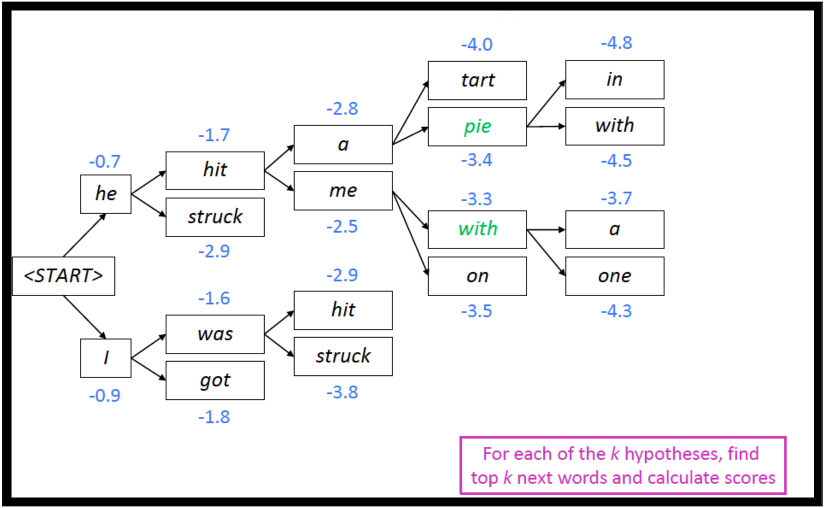
Beam search decoding: example

Beam size = k =2. Blue numbers =

$score(y_1, \dots, y_t) = \sum_{i=1}^{t} logP_{LM}(y_i | y_1, \dots, y_{i-1}, x)$

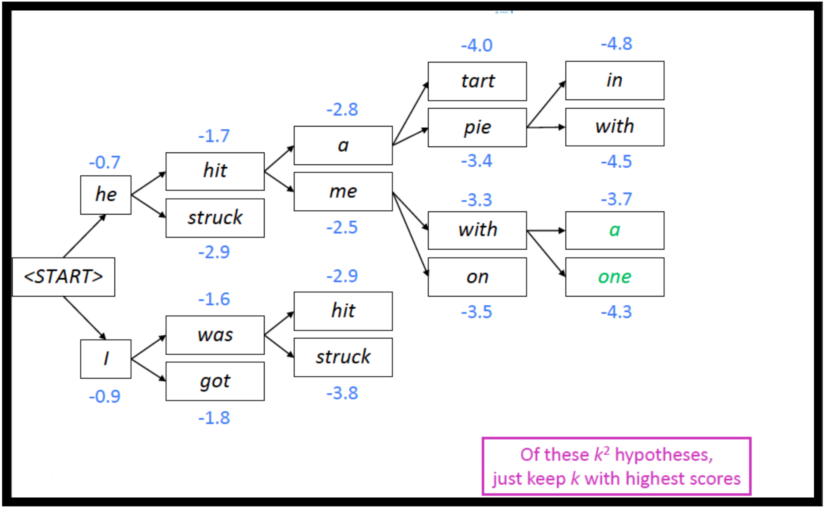


Beam size = k =2. Blue numbers =
$score(y_1,\ldots,y_t)=\sum_{i=1}^{t} logP_{LM}(y_i|y_1,\ldots,y_{i-1},x)$



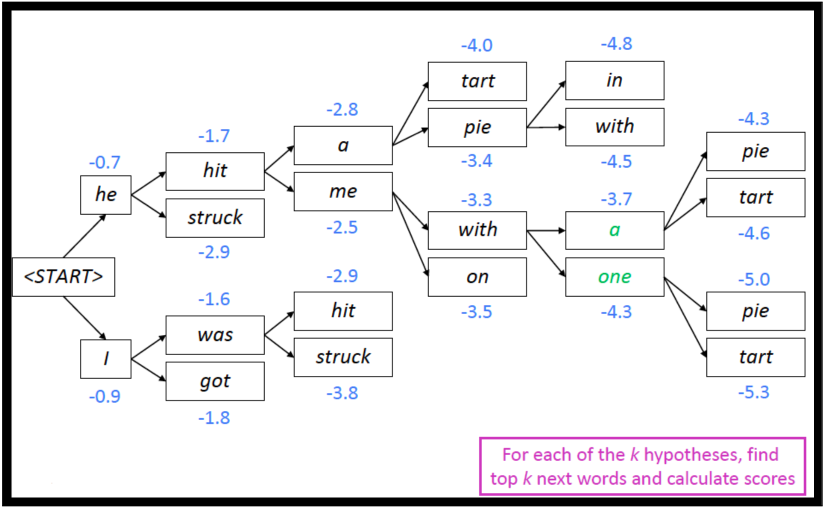
Beam search decoding: example

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

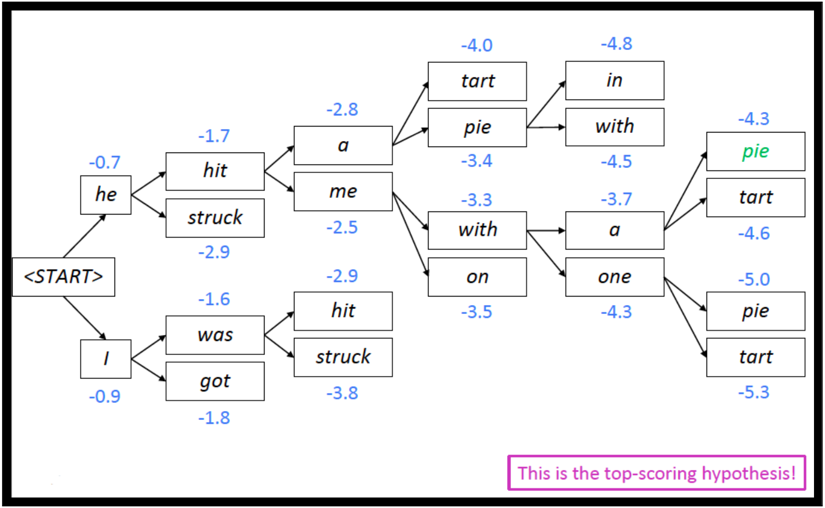


Beam search decoding: example

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

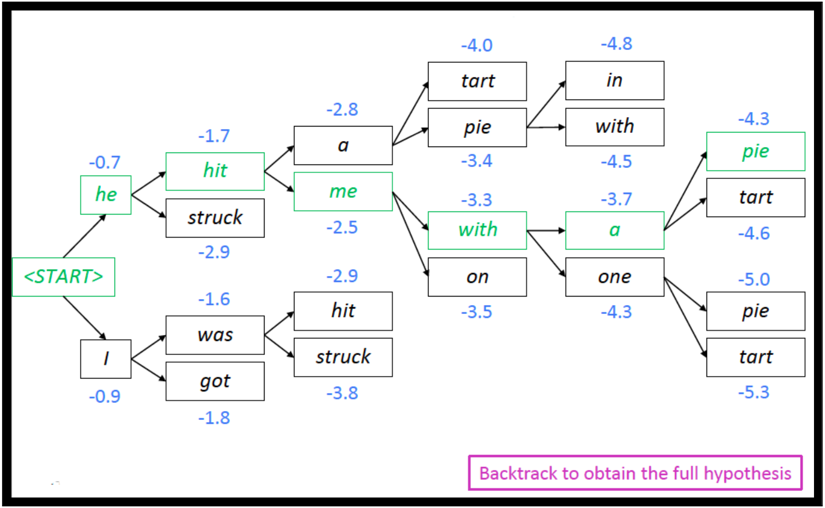


Beam size = k =2. Blue numbers = $score(y_1,\ldots,y_t) = \sum_{i=1}^{t} logP_{LM}(y_i|y_1,\ldots,y_{i-1},x)$



Beam search decoding: example

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



Beam Search Decoding: Stopping Criterion

- In greedy decoding,

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$
- In beam search decoding,

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$
- In beam search decoding, different hypotheses may produce $\langle END \rangle$ tokens on different timesteps

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$
- In beam search decoding, different hypotheses may produce $\langle END \rangle$ tokens on different timesteps
  - When a hypothesis produces $\langle END \rangle$, that hypothesis is complete
  - Place it aside and continue exploring other hypotheses via beam search
- Usually we continue beam search until:

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$
- In beam search decoding, different hypotheses may produce $\langle END \rangle$ tokens on different timesteps
  - When a hypothesis produces $\langle END \rangle$, 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 (some pre-defined cutoff), or

Beam Search Decoding: Stopping Criterion

- In greedy decoding, usually we decode until the model produces an $\langle END \rangle$ token
  - **Example:** $\langle START \rangle$ he hit me with a pie $\langle END \rangle$
- In beam search decoding, different hypotheses may produce $\langle END \rangle$ tokens on different timesteps
  - When a hypothesis produces $\langle END \rangle$, 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 (some pre-defined cutoff), or
  - We have at least n completed hypotheses (some pre-defined cutoff)

Beam search decoding: finishing up

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

Beam search decoding: finishing up

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

Each hypothesis $y_1, \ldots, y_t$ has a score

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

Beam search decoding: finishing up

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

Each hypothesis $y_1, \ldots, y_t$ has a score

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

Problem with this

Beam search decoding: finishing up

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

Each hypothesis $y_1, \ldots, y_t$ has a score

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

Problem with this

Longer hypotheses have lower scores.

Beam search decoding: finishing up

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

Each hypothesis $y_1, \ldots, y_t$ has a score

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

Problem with this

Longer hypotheses have lower scores.

Fix: Normalize the score by length, and use the normalized score to select the top one instead.

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

Other Sampling Strategies

Random Sampling with temperature

More probable words would have more chance of being generated

$$P(x_i|x_{1,\dots,i-1}) = \frac{exp(u_i/t)}{\sum_j exp(u_j/t)}, 0 < t \leq 1$$

Other Sampling Strategies

Random Sampling with temperature

More probable words would have more chance of being generated

$$P(x_i|x_{1,\dots,i-1}) = \frac{exp(u_i/t)}{\sum_j exp(u_j/t)}, 0 < t \leq 1$$

Top-k Sampling

Only Top k tokens are considered for generation, so the less probable words would not have any chance (rescale).

Other Sampling Strategies

Random Sampling with temperature

More probable words would have more chance of being generated

$$P(x_i|x_{1,\ldots,i-1}) = \frac{exp(u_i/t)}{\sum_j exp(u_j/t)}, 0 < t \leq 1$$

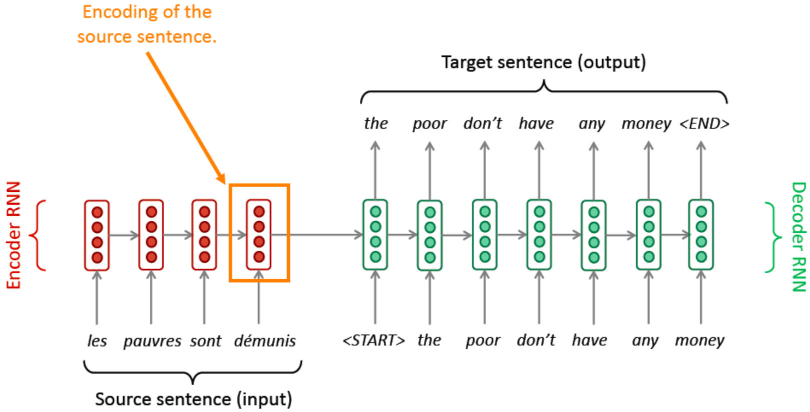
Top-k Sampling

Only Top k tokens are considered for generation, so the less probable words would not have any chance (rescale).

Nucleus Sampling

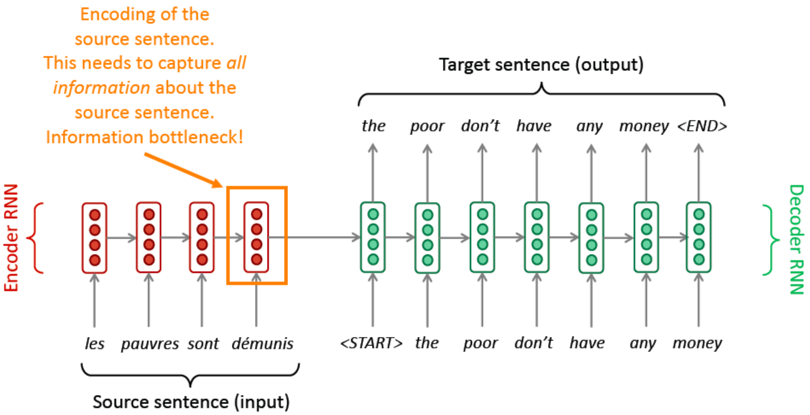
Focus on the smallest set of words such that the sum of their probability is $\geq p$. Helps when the model is certain on some words.

Sequence to Sequence Models: the Bottleneck



Problems with this architecture?

Sequence to Sequence Models: the Bottleneck

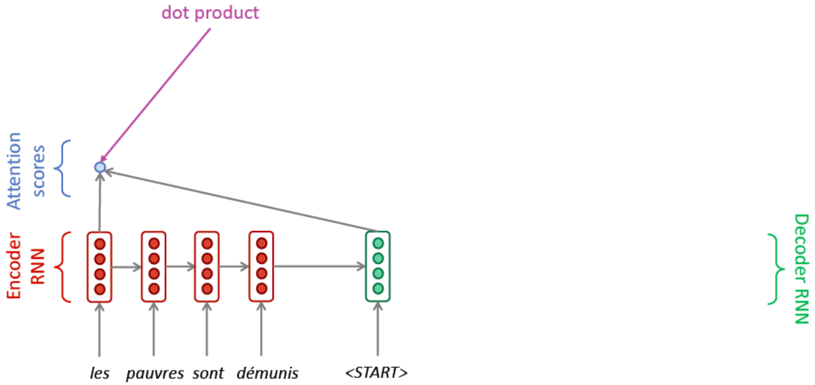


Attention provides a solution to the bottleneck problem.

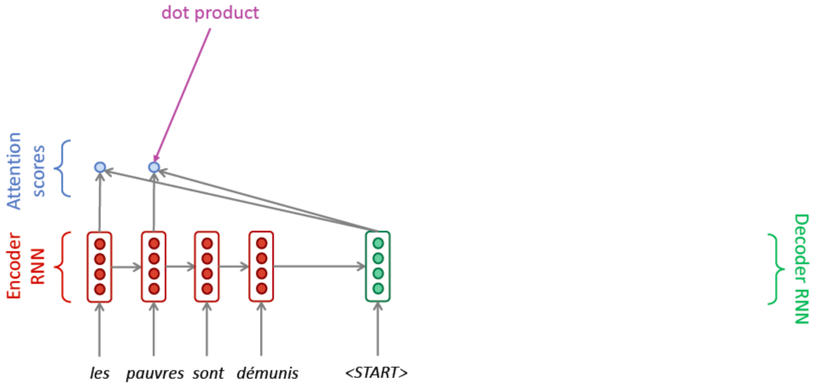
Core Idea

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

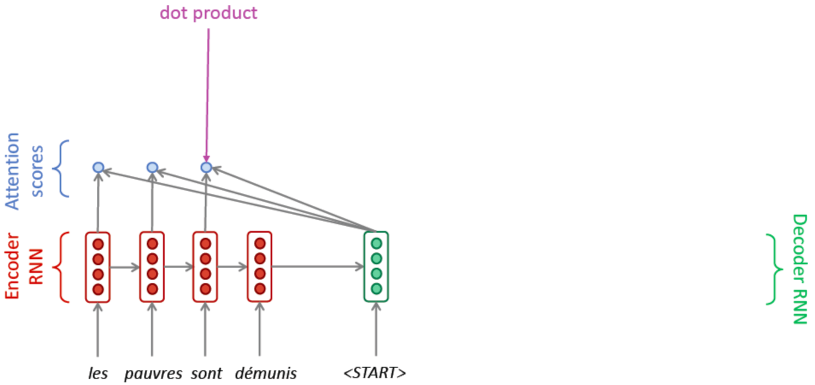
Sequence to Sequence with Attention



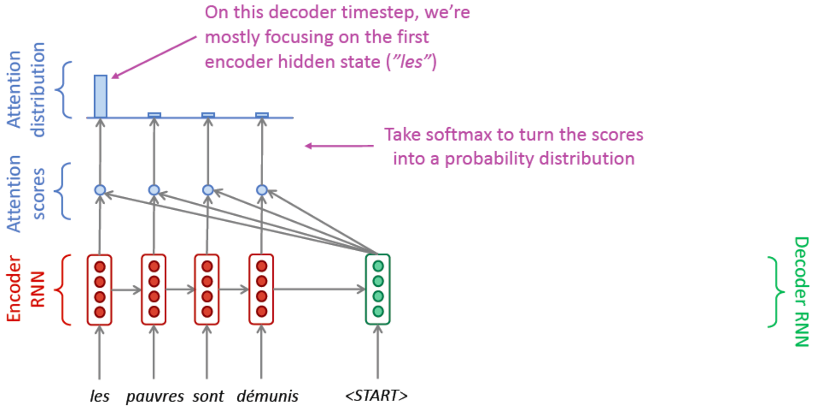
Sequence to Sequence with Attention



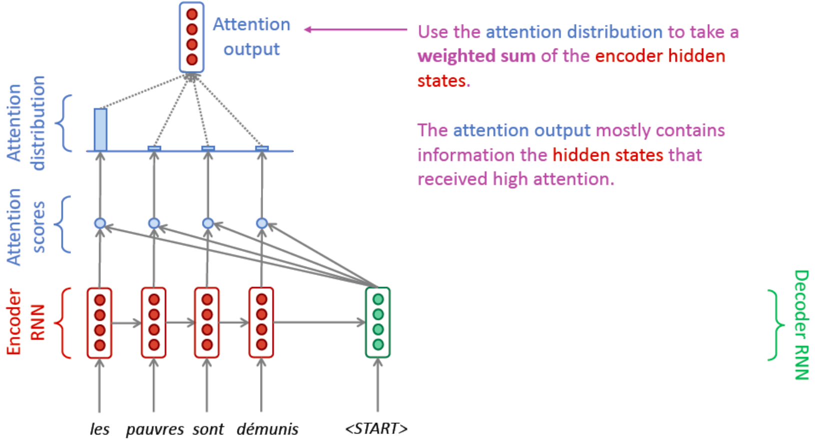
Sequence to Sequence with Attention



Sequence to Sequence with Attention



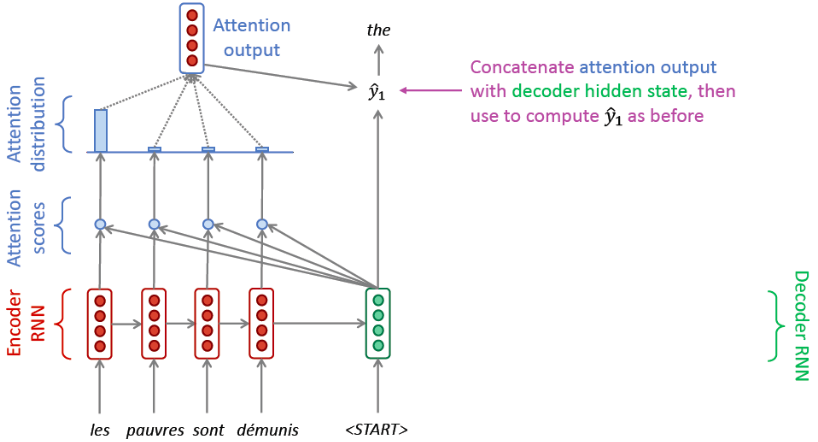
Sequence to Sequence with Attention



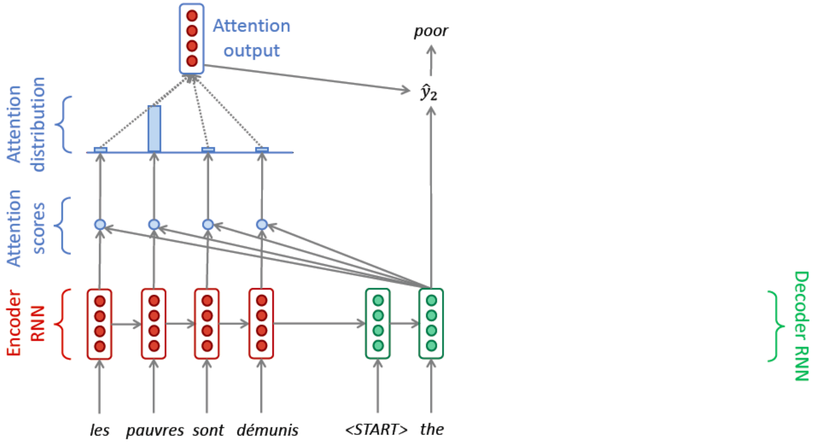
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

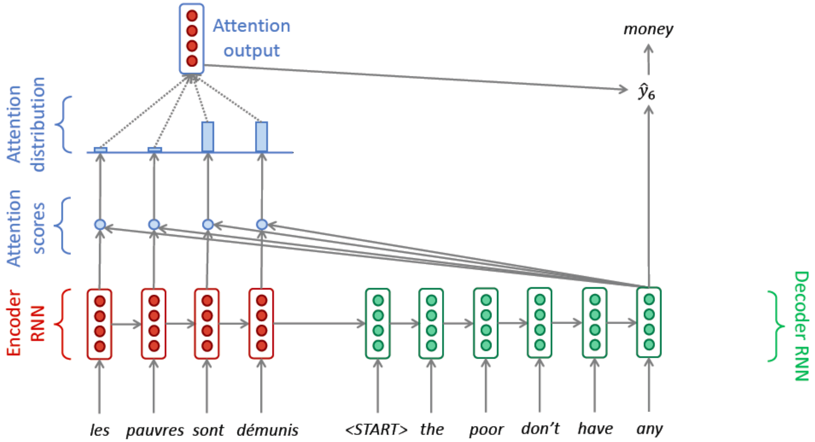
Sequence to Sequence with Attention



Sequence to Sequence with Attention



Sequence to Sequence with Attention



Attention: in equations

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

Attention: in equations

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

Attention: in equations

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

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

Attention: in equations

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

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

- We take softmax to get the attention distribution α^t for this step (probability distribution)

$$\alpha^t = softmax(e^t) \in R^N$$

Attention: in equations

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

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

- We take softmax to get the attention distribution α^t for this step (probability distribution)

$$\alpha^t = softmax(e^t) \in R^N$$

- Use α^t to take a weighted sum of the encoder hidden states to get the attention output a^t

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

Attention: in equations

- We have encoder hidden states $h_1, \dots, h_N \in R^h$
- On timestep t, we have decoder hidden state $s_t \in R^h$
- We get attention scores e^t for this step:

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in R^N$$

- We take softmax to get the attention distribution α^t for this step (probability distribution)

$$\alpha^t = softmax(e^t) \in R^N$$

- Use α^t to take a weighted sum of the encoder hidden states to get the attention output a^t

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

- Finally we concatenate the attention output a^t with the decoder hidden state s_t and proceed as in the non-attention seq2seq: $[a_t; s_t] \in R^{2h}$

Attention is quite helpful

Attention improves NMT performance

It is useful to allow decoder to focus on certain parts of the source

Attention is quite helpful

Attention improves NMT performance

It is useful to allow decoder to focus on certain parts of the source

Attention helps with the long-term dependency problem

Provides shortcut to faraway states

Attention is quite helpful

Attention improves NMT performance

It is useful to allow decoder to focus on certain parts of the source

Attention helps with the long-term dependency problem

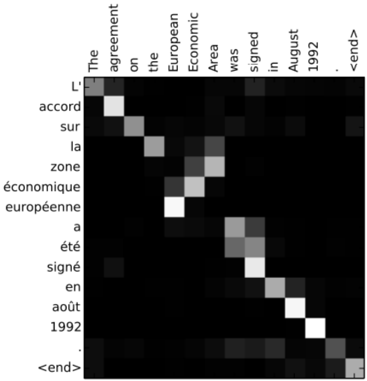
Provides shortcut to faraway states

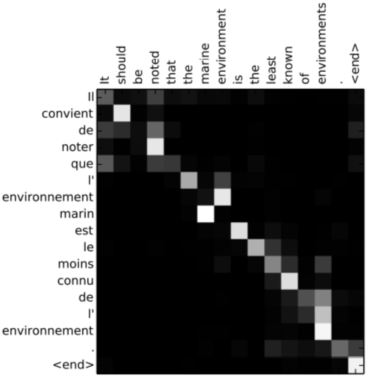
Attention provides some interpretability

- By inspecting attention distribution, we can see what the decoder was focusing on
- We get alignment for free even if we never explicitly trained an alignment system

Example: Machine Translation

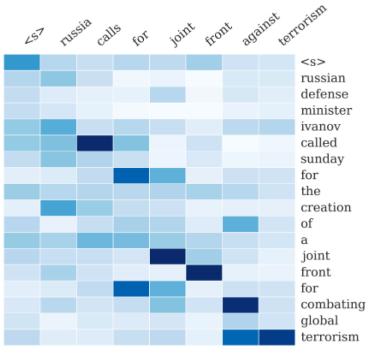
Neural Machine Translation by jointly learning to align and Translate, ICLR 2015





Example: Text Summarization

A Neural Attention Model for Sentence Summarization, EMNLP 2015



Summary

- Attention has proved to be a very impactful idea in NLP
- Lot of new models are based on self-attention, e.g., Transformer, BERT