CentraleSupélec

Introduction to Computational Linguistics

Part 2: Language Models

Camilo Carvajal Reyes

3rd March 2021

CentraleSupélec

Outline

- *8:30 - 9:00* - **Lecture**
  Word Embeddings: challenges and techniques
- *9:00 - 9:30* - **Practical Work**
  Visualisation of semantic relations
- *9:30 - 9:45* - Break
- *9:45 - 10:30* - **Lecture**
  Language Modelling: deep learning methods for NLP
- *10:30 - 11:00* - **Practical Work**
- *10:30 - 11:00* - **Practical Work**
  Transformer LM Part 1: Exploring a pre-trained model
- *11:00 - 11:15* - Break
- *11:15 - 11:45* - **Practical Work**
  Transformer LM Part 2: Fine-tuning for Sentiment Analysis

Table of contents

CentraleSupélec

Table of contents

What are Language Models?

In the previous section we named some applications of NLP such as Machine Translation, Natural Language Generation, Sentiment Analysis, etc. However, with Word Embeddings, we have only taken the first step.

The aforementioned tasks need **representations of sentences or documents**, therefore the use of Language Models (**LM**). Formally, a LM consists of assigning a probability to a sequence of words $\{w_1, \ldots, w_n\}$. Moreover, this is equivalent to the problem of **finding a missing word in the sequence**[1]:

$$P(w_i|w_1, \ldots, w_n) = \frac{P(w_1, \ldots, w_i)}{P(w_1, \ldots, w_n)}$$

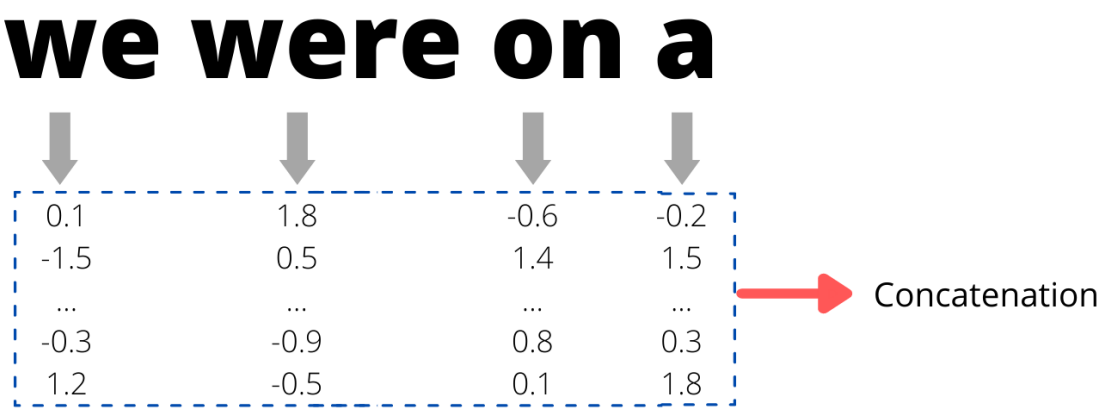
In practice, we will use Language Modelling to train representations rather than directly using the probabilities.

[1] https://towardsdatascience.com/language-models-1a08779b8e12

Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

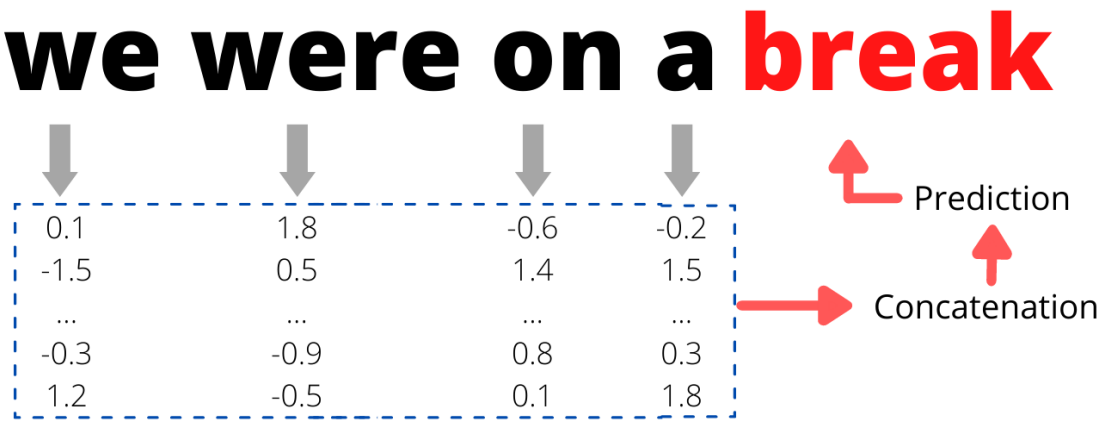
- **Next word prediction (NWP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

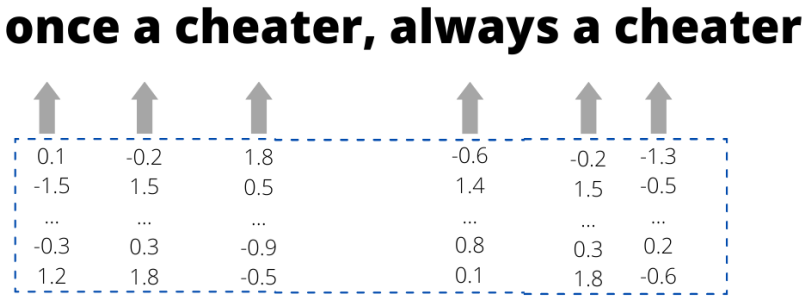
- **Next word prediction (NWP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

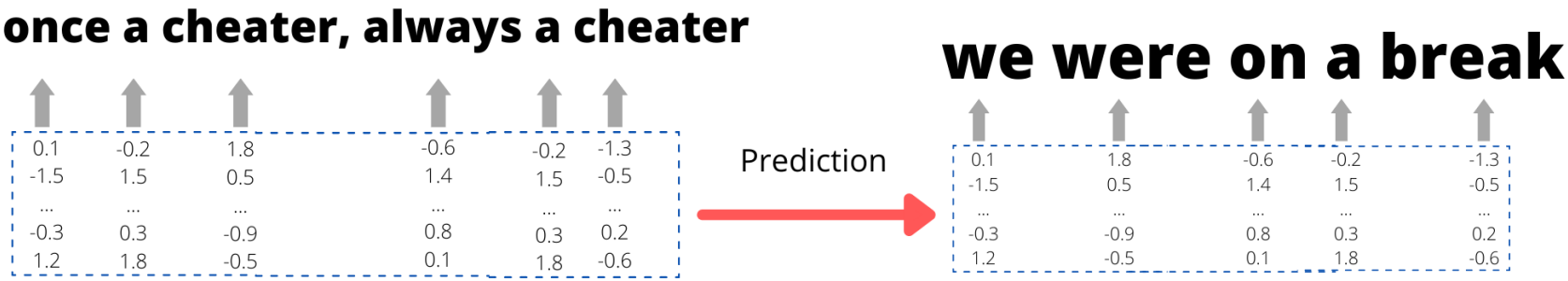
- **Next word prediction (NWP)**
- **Next sentence prediction (NSP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

- **Next word prediction (NWP)**
- **Next sentence prediction (NSP)**



Learning Objectives

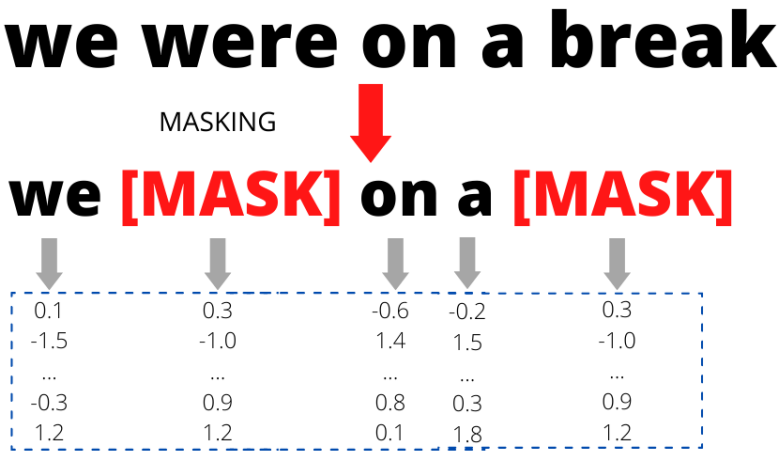
Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

- **Next word prediction (NWP)**
- **Next sentence prediction (NSP)**
- **Masked Language Model (MLM)**

Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

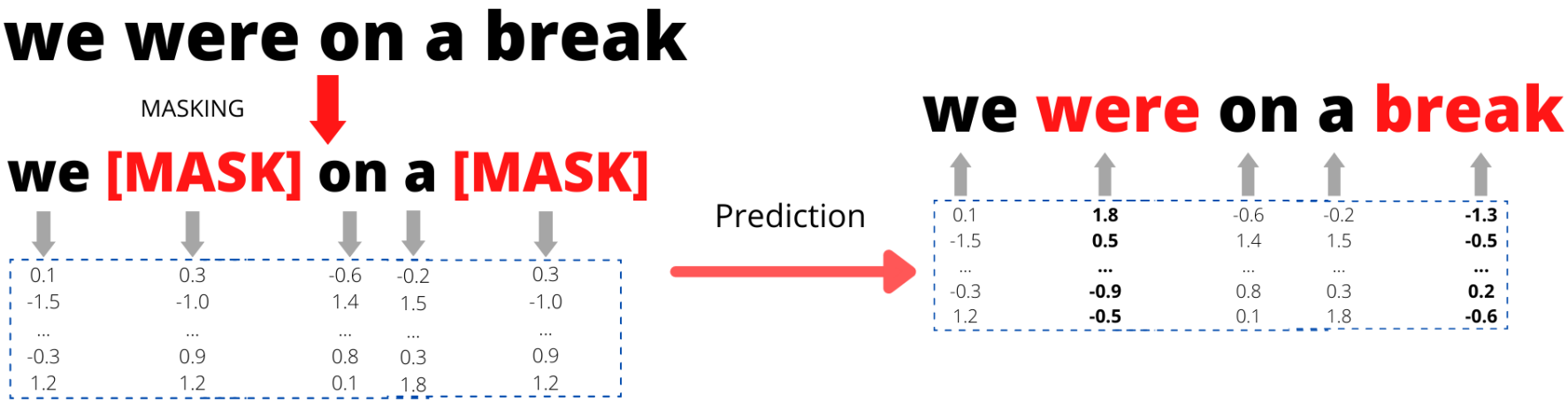
- **Next word prediction (NWP)**
- **Next sentence prediction (NSP)**
- **Masked Language Model (MLM)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [1]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

- **Next word prediction (NWP)**
- **Next sentence prediction (NSP)**

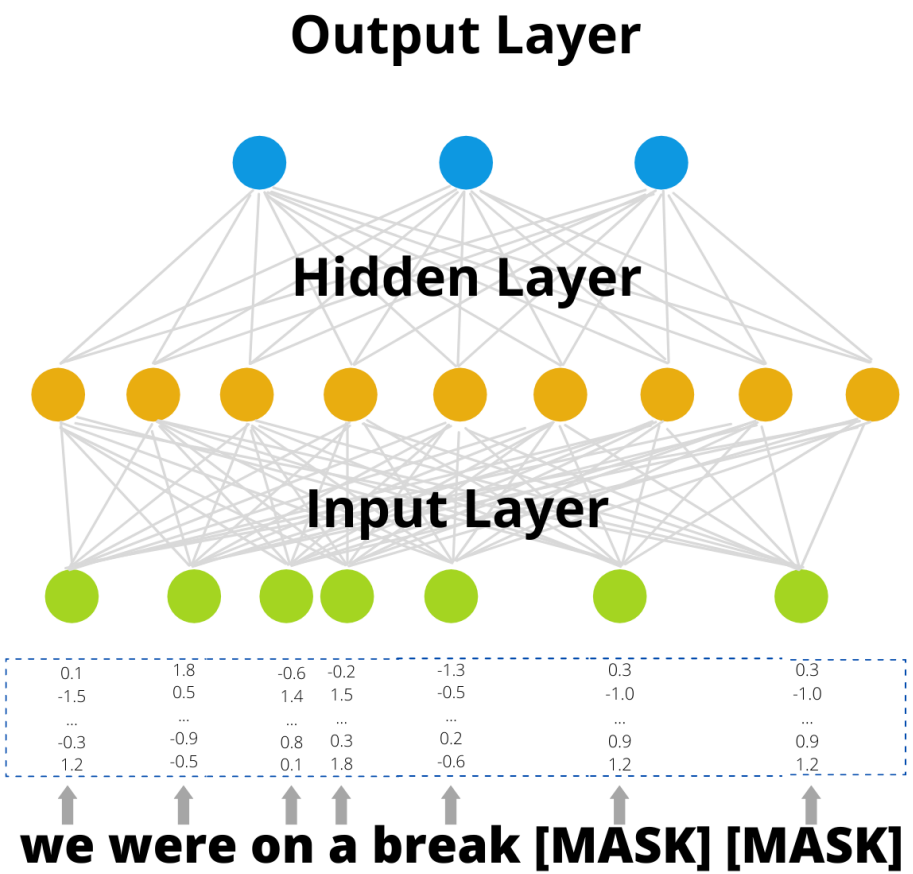


Architectures for Language Modelling

In Neural LM we combine vectors using **neural networks**. The simplest option is using a simple **Multi-layer perceptron** as in Bengio et al. [1]. However, this is arguably not the best option to encode language.

We will go through two more options that are widely used today:

- Recurrent Neural Networks (RNN)
- Transformers

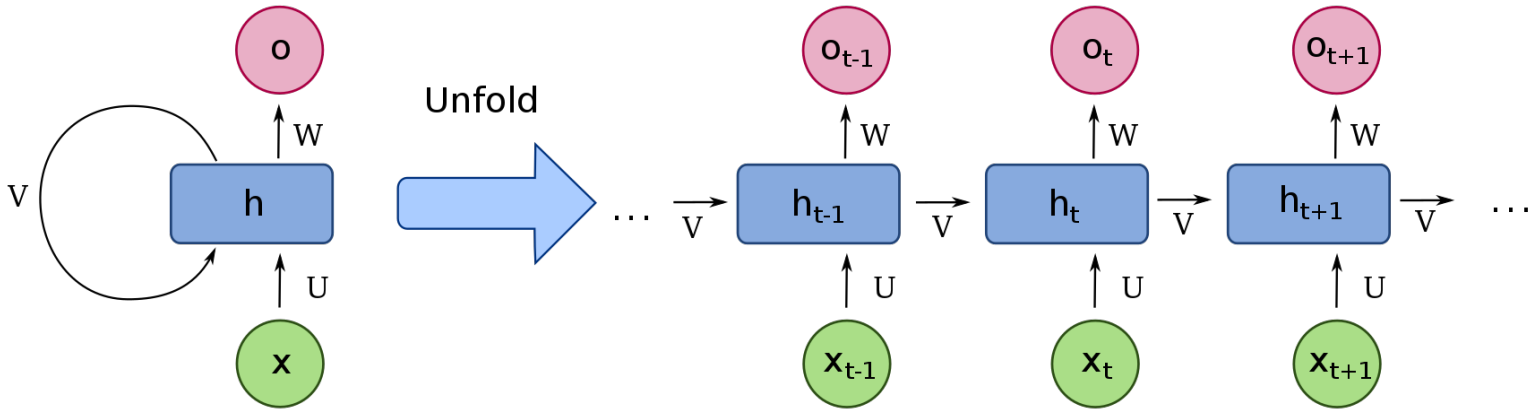


CentraleSupélec

Table of contents

Recurrent Neural Networks

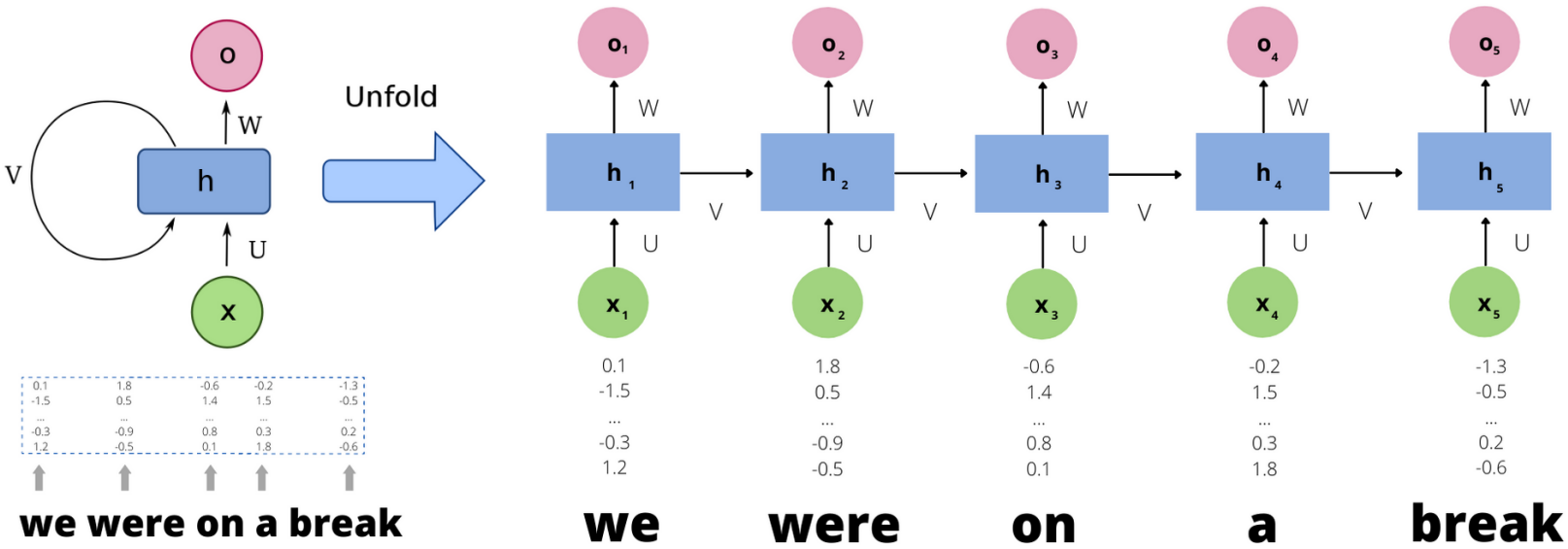
RNNs are an ideal solution to deal with sequences of unknown length. In its simple version (**Vanilla RNN**) we simply apply a neural network to generate an **output** and a hidden state to every member of the sequence.



For any given word, we apply both the input vector and the previous hidden state.

Recurrent Neural Networks

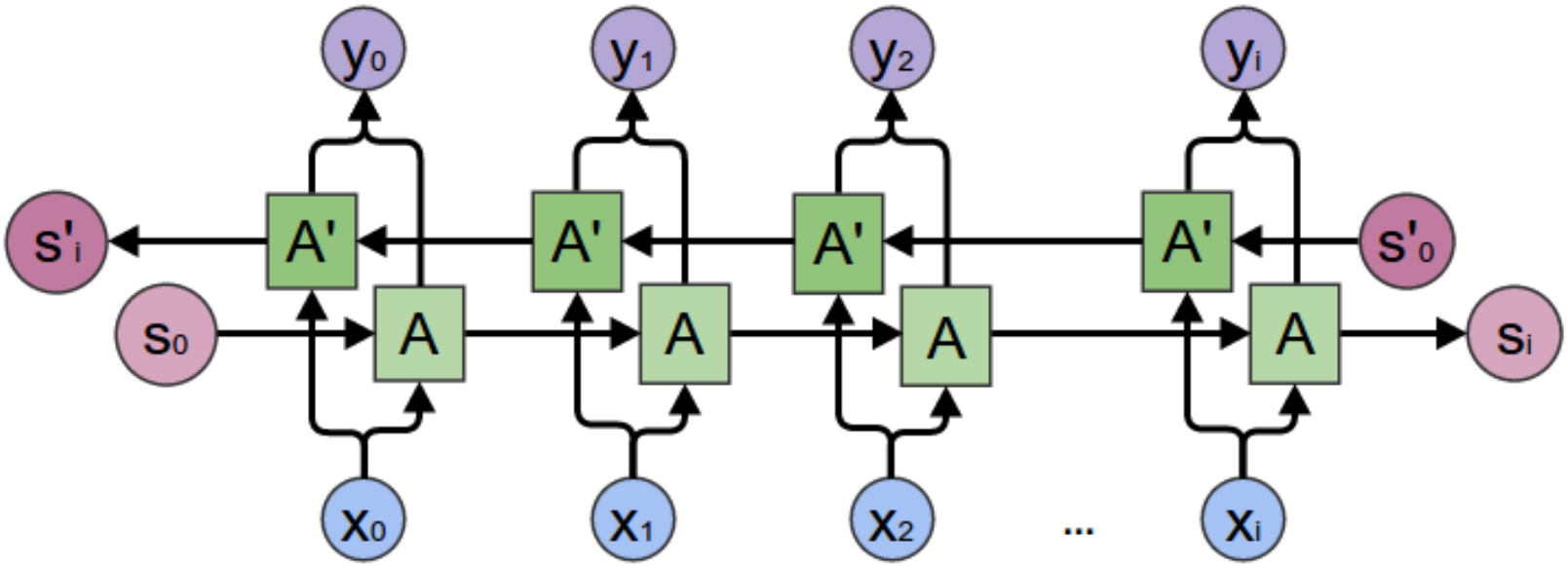
RNNs are an ideal solution to deal with sequences of unknown length. In its simple version (**Vanilla RNN**) we simply apply a neural network to generate an **output** and a hidden state to every member of the sequence.



For any given word, we apply both the input vector and the previous hidden state.

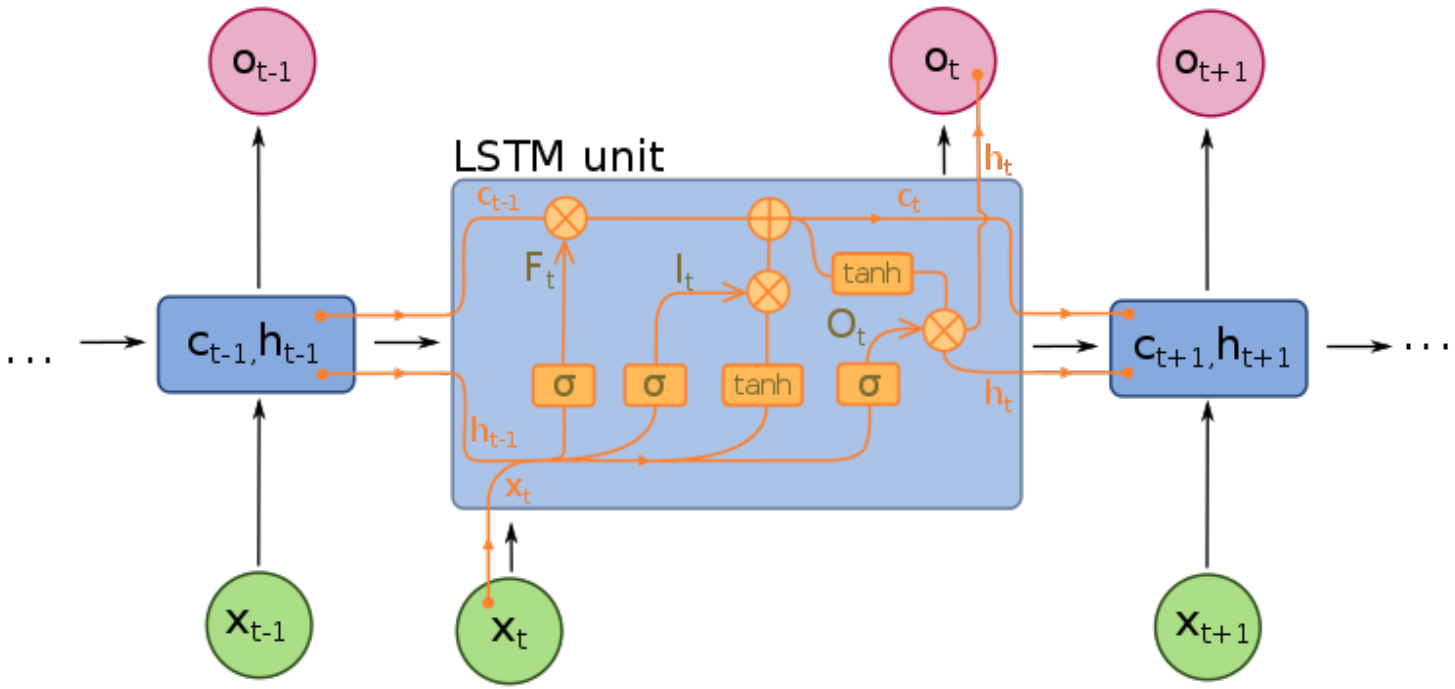
Bi-directional LSTM

RNNs are not perfect: in particular, they suffer from the **vanishing gradient problem**. This is, when we back-propagate the error to update the weights, we loose information, particularly with long sequences.
In reality, we often apply **bi-directionality**.



Bi-directional LSTM

RNNs are not perfect: in particular, they suffer from the **vanishing gradient problem**. This is, when we back-propagate the error to update the weights, we loose information, particularly with long sequences.
In reality, we often apply **bi-directionality** and we use a more complex architecture instead: the **Long Short Term Memory Network**.

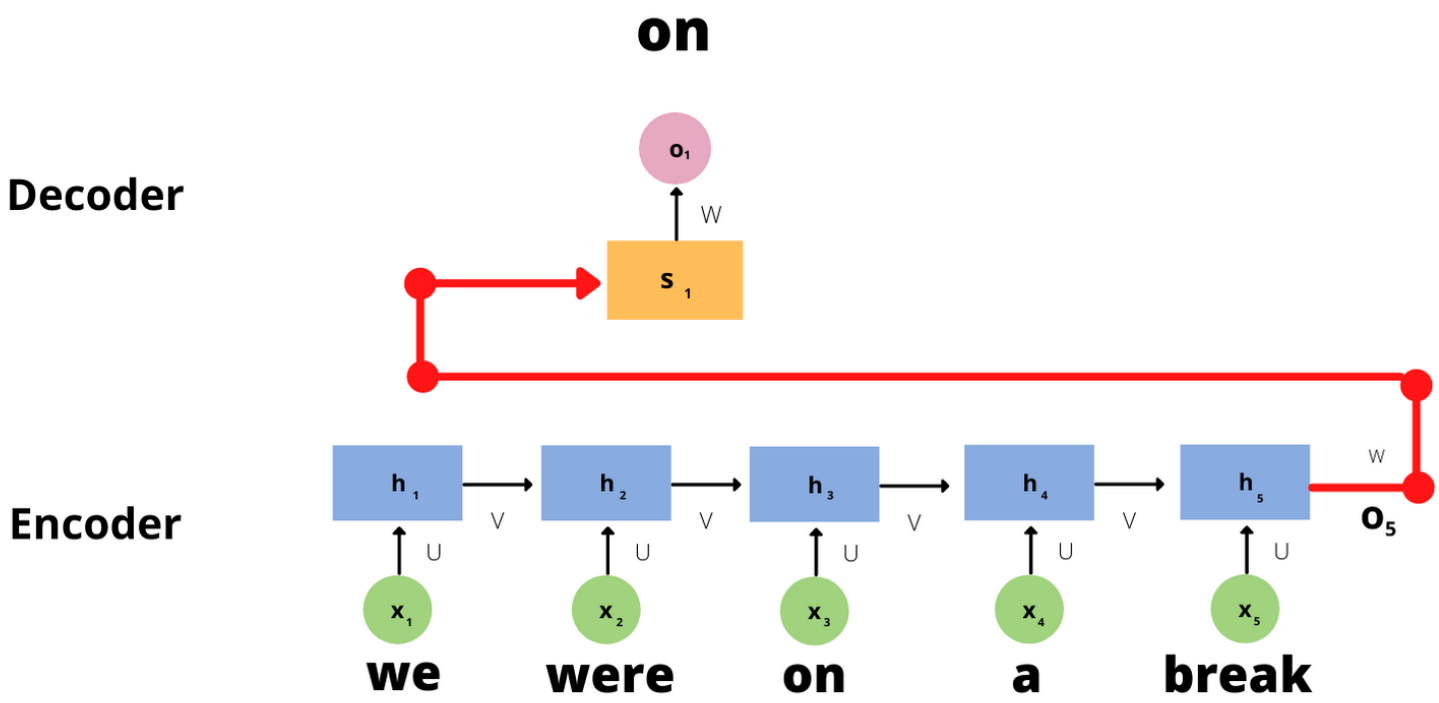


CentraleSupélec

Table of contents

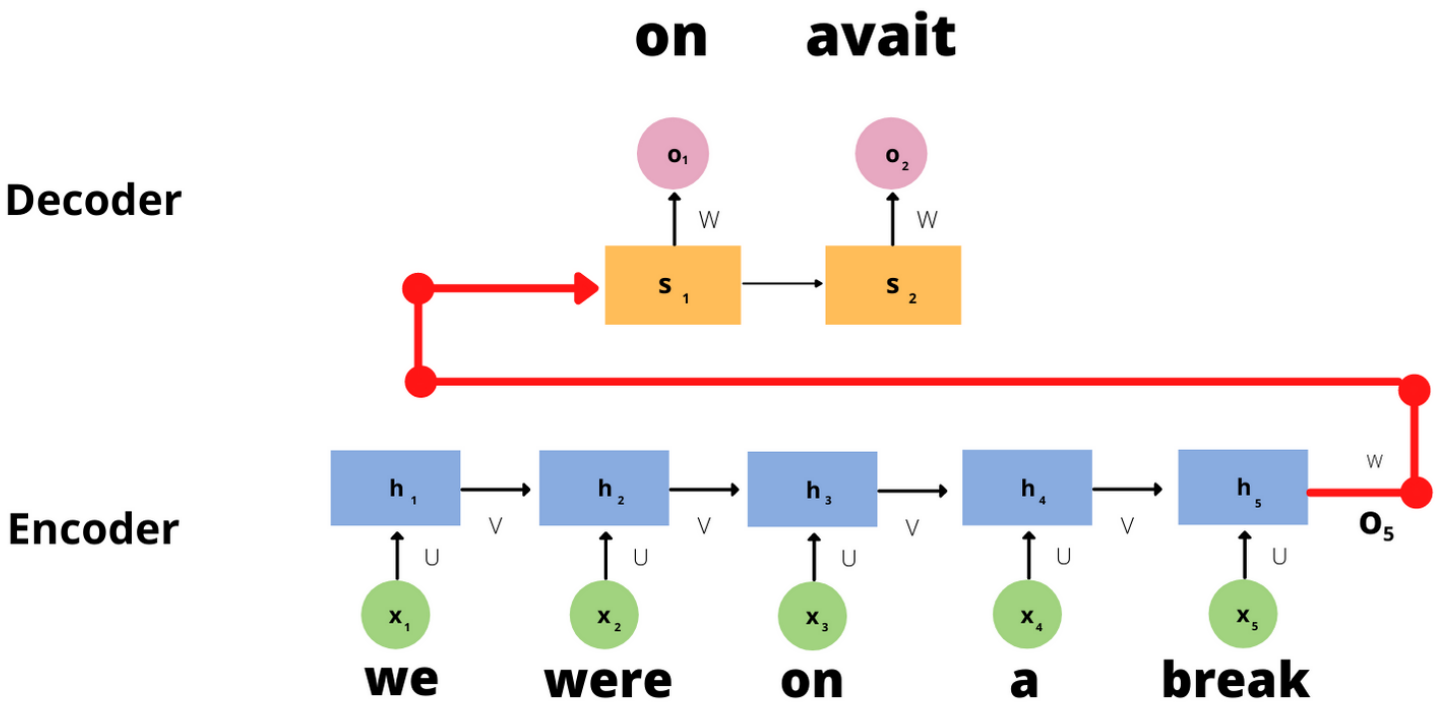
RNNs: a Machine Translation example

Let's consider an example of **machine translation**. We will consider an RNN for **encoding** the input sequence and another one for **decoding** it into the output sequence.



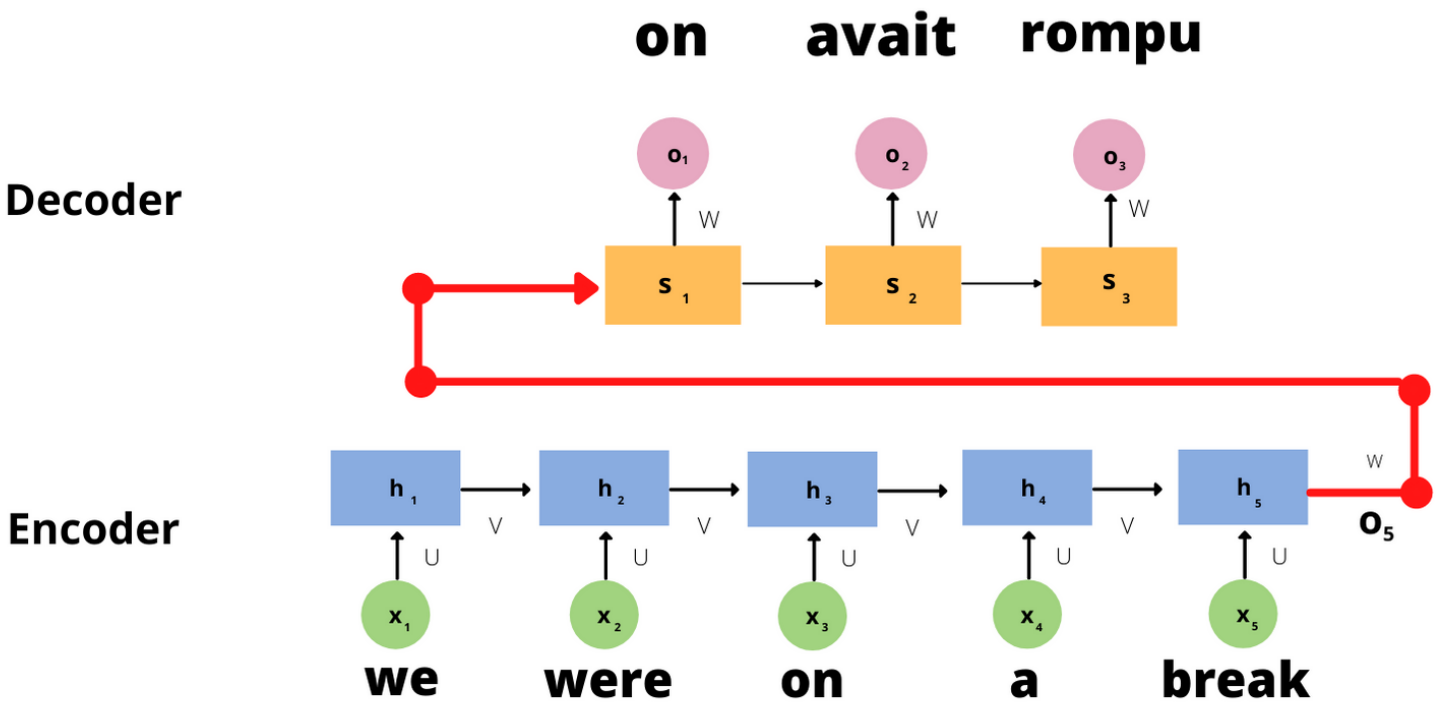
RNNs: a Machine Translation example

Let's consider an example of **machine translation**. We will consider an RNN for **encoding** the input sequence and another one for **decoding** it into the output sequence.



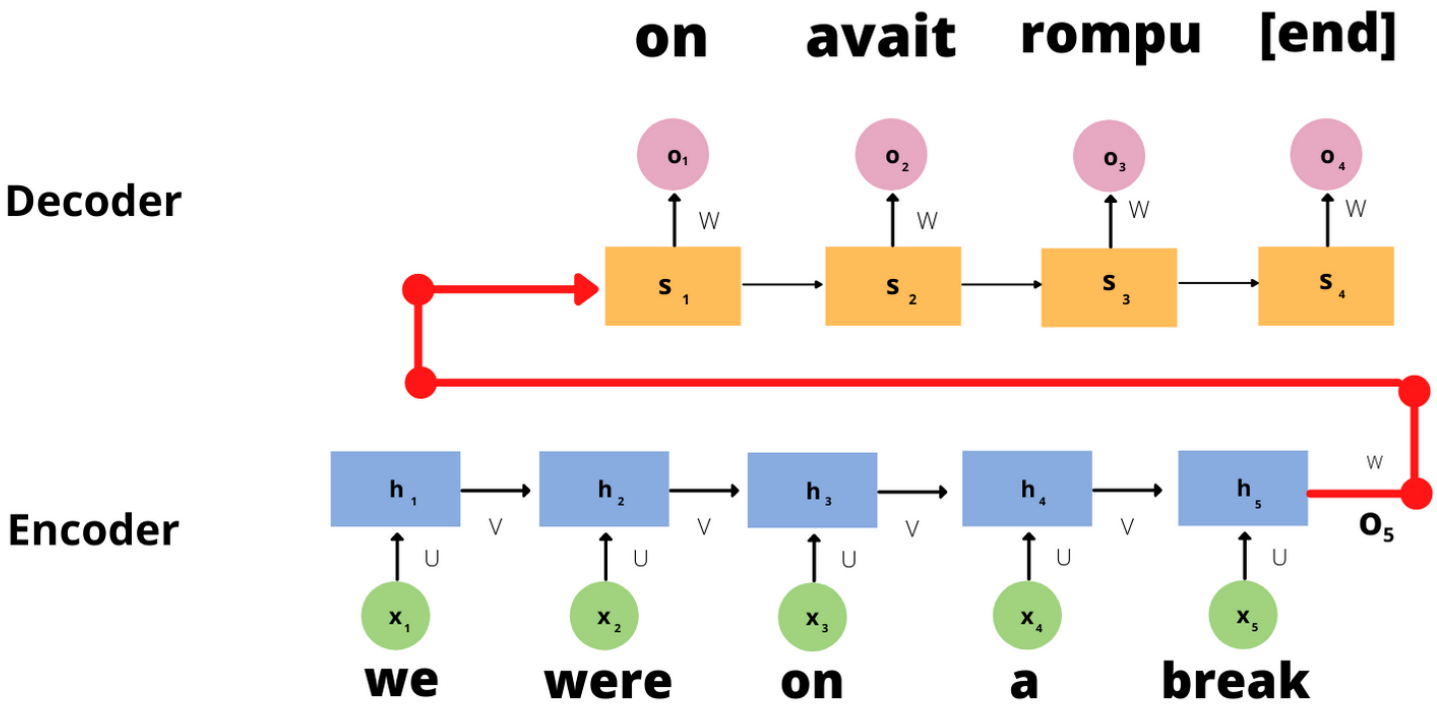
RNNs: a Machine Translation example

Let's consider an example of **machine translation**. We will consider an RNN for **encoding** the input sequence and another one for **decoding** it into the output sequence.



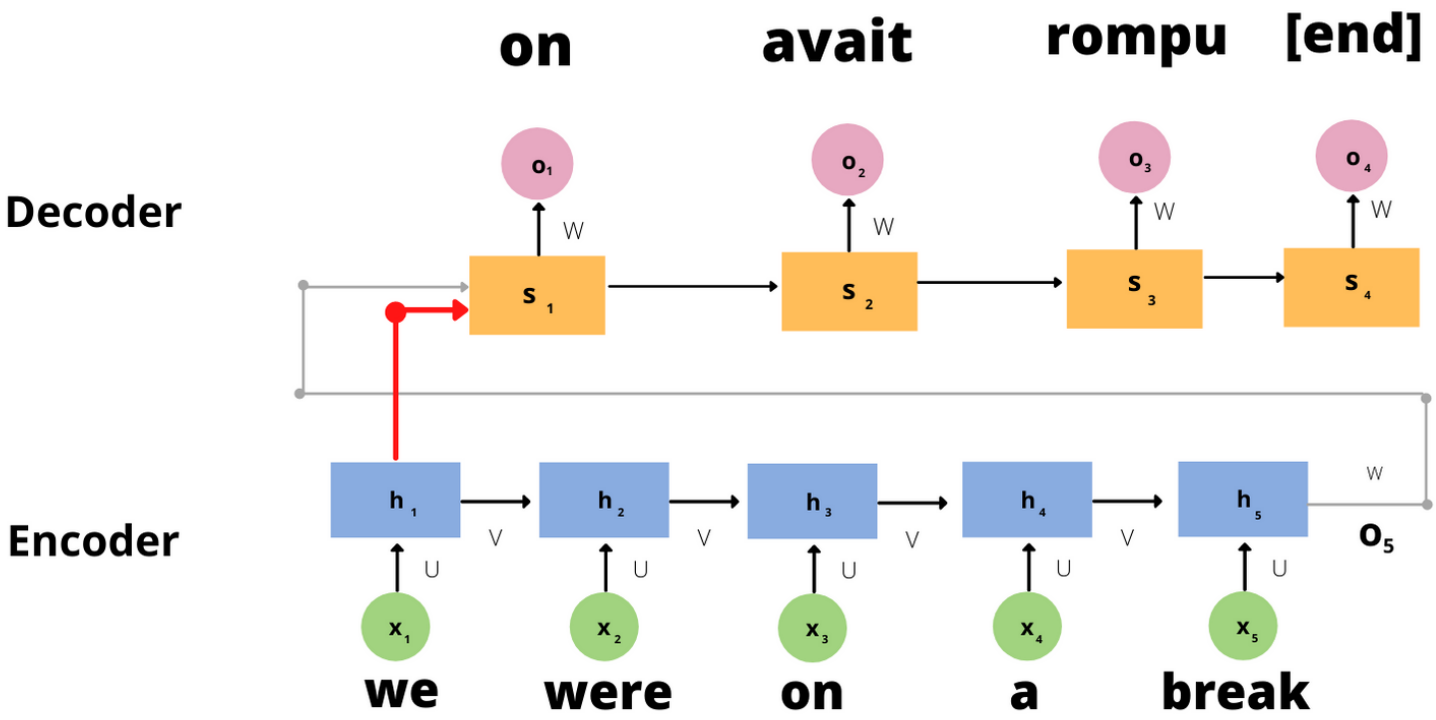
RNNs: a Machine Translation example

A drawback of such a setting is that by encoding the whole sentence in the last output, we are not pushing the decoder to **focus on the important parts of the input**. For example, for computing **on** in French, we only need to pay attention to the word **we** in English.



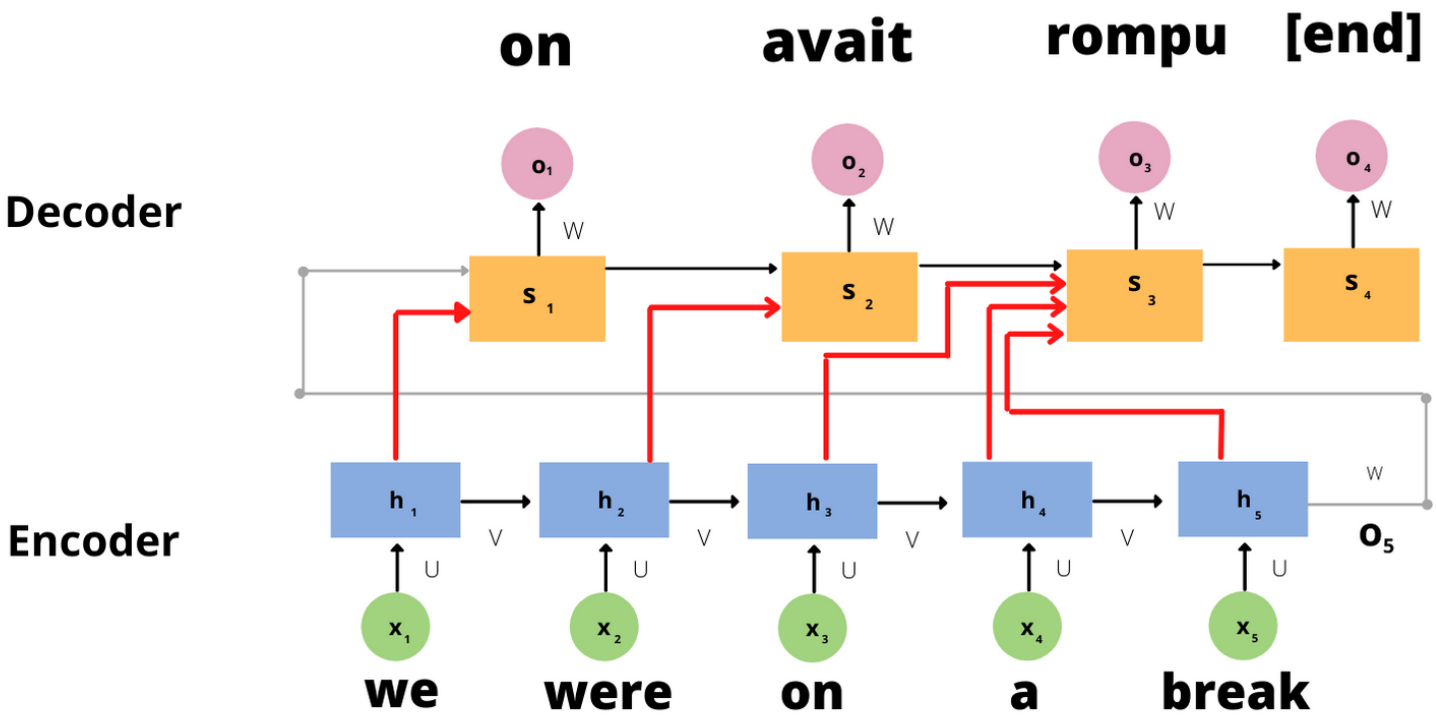
Attention

Attention tries to overcome the challenge we mentioned on the previous slide by making **direct connections** between output tokens and the **important tokens** on the input that correspond to it. Let's look at our example again:



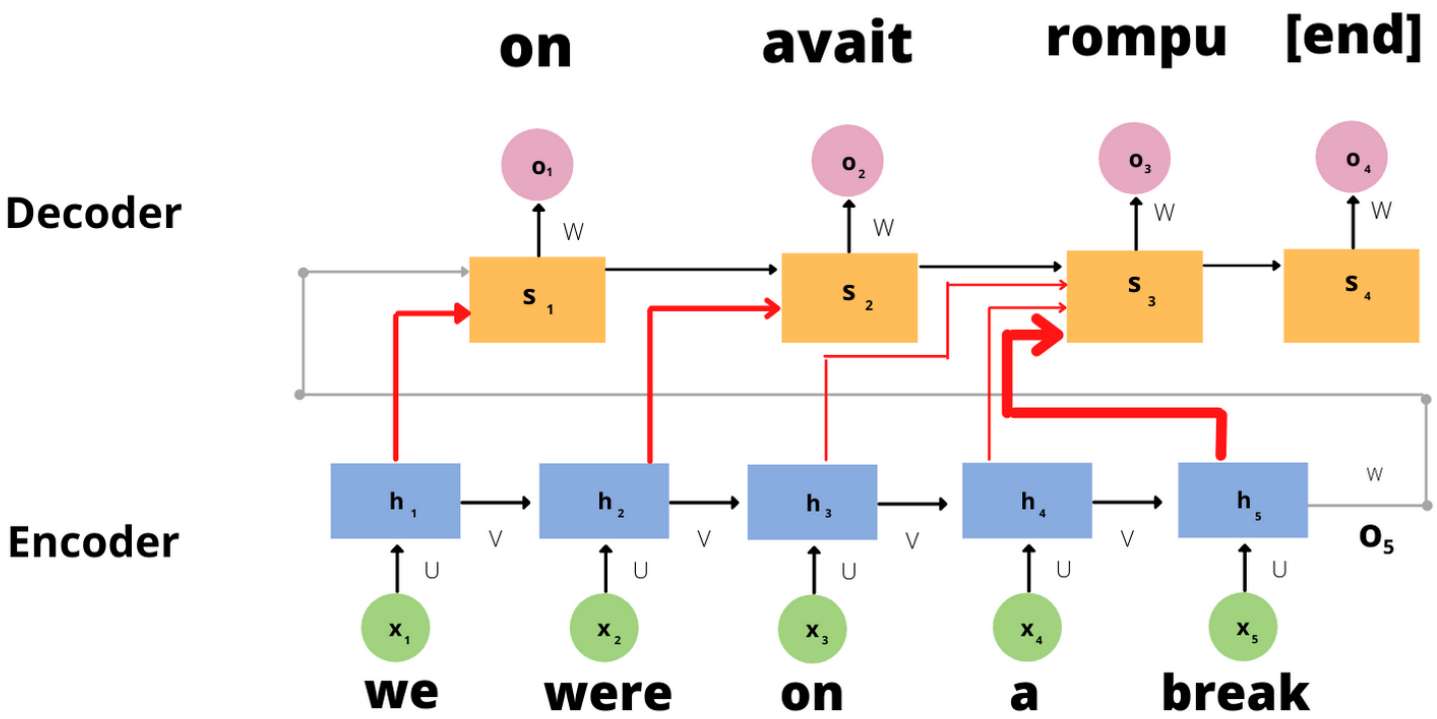
Attention

Attention tries to overcome the challenge we mentioned on the previous slide by making **direct connections** between output tokens and the **important tokens** on the input that correspond to it. Let's look at our example again:



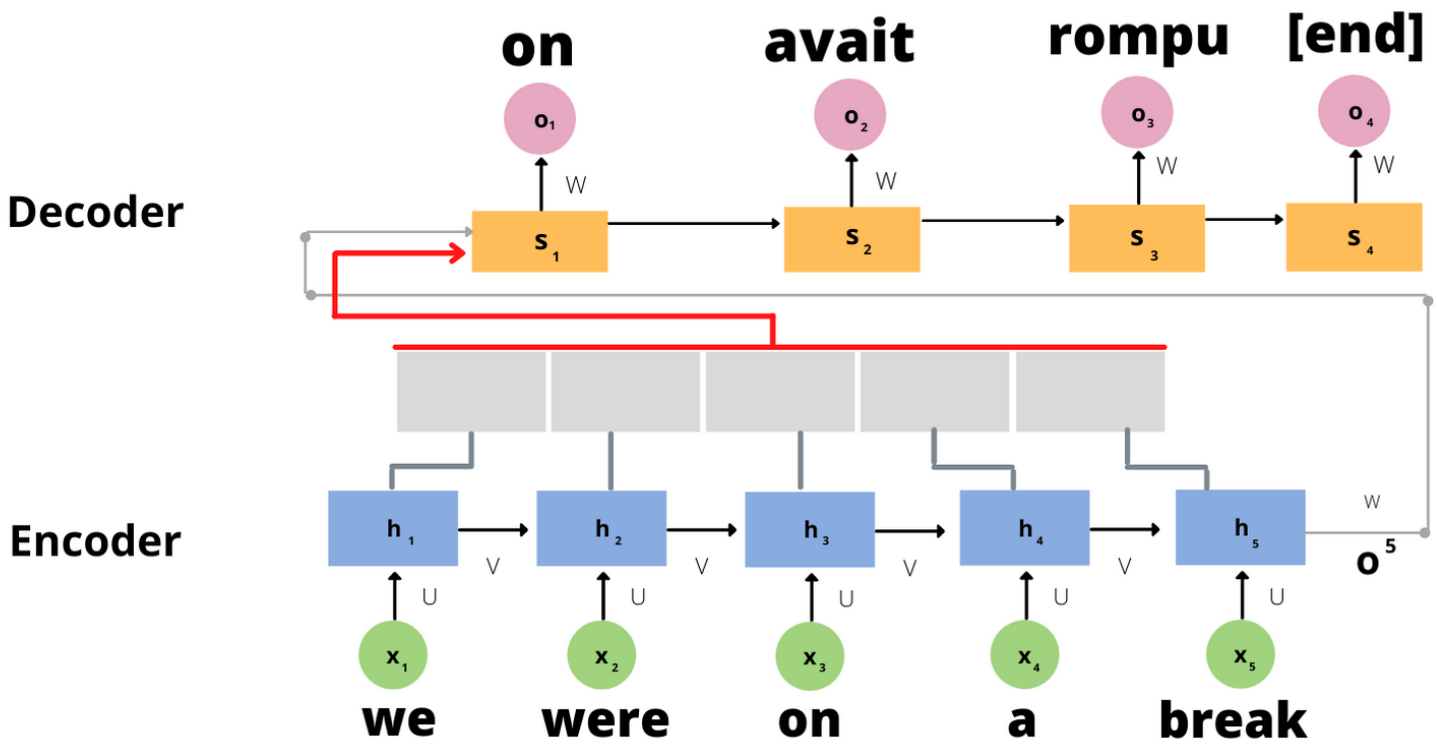
Attention

Attention tries to overcome the challenge we mentioned on the previous slide by making **direct connections** between output tokens and the **important tokens** on the input that correspond to it. Moreover, we want the module to **attend more** to **more important tokens**.



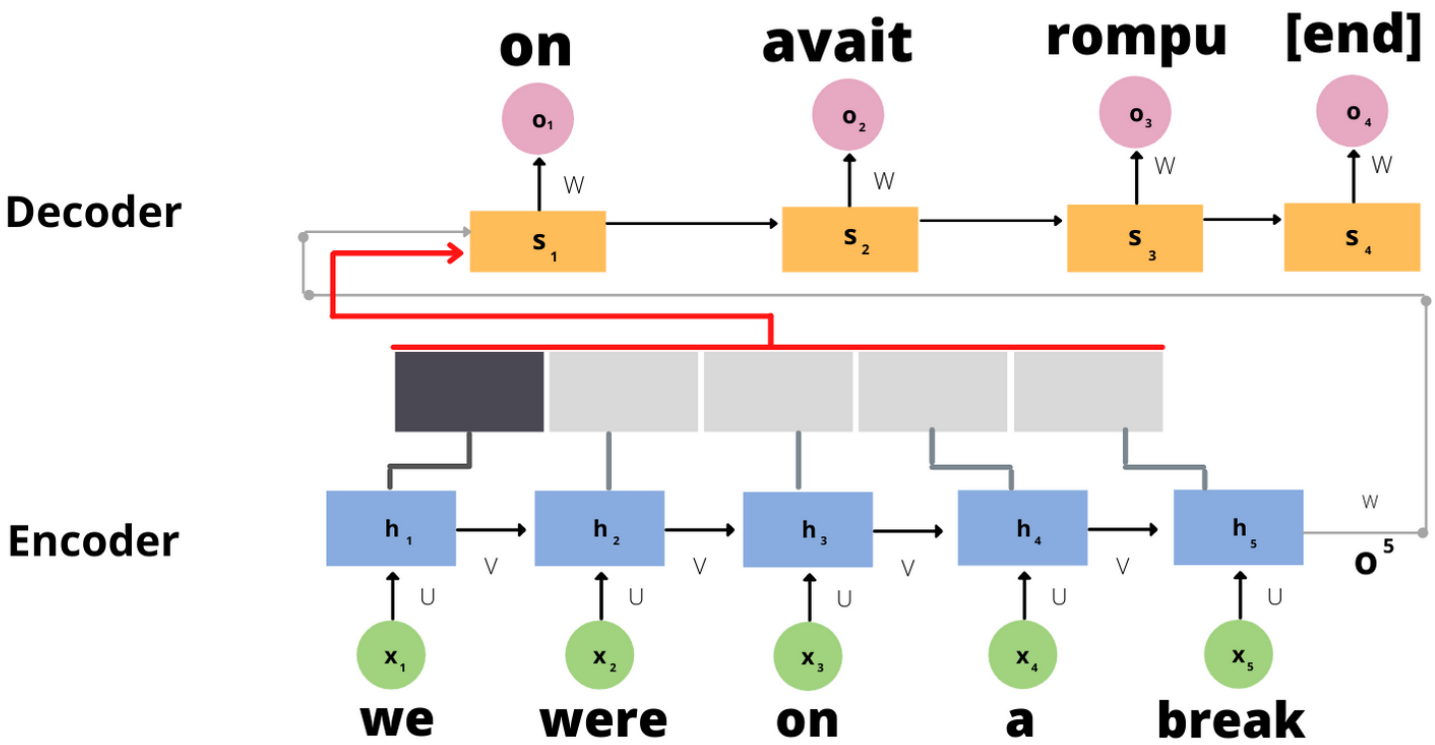
Attention Modules

In other words, attention allows a direct interaction between different token positions by introducing a weighted average of the previous RNN layers into the next one.



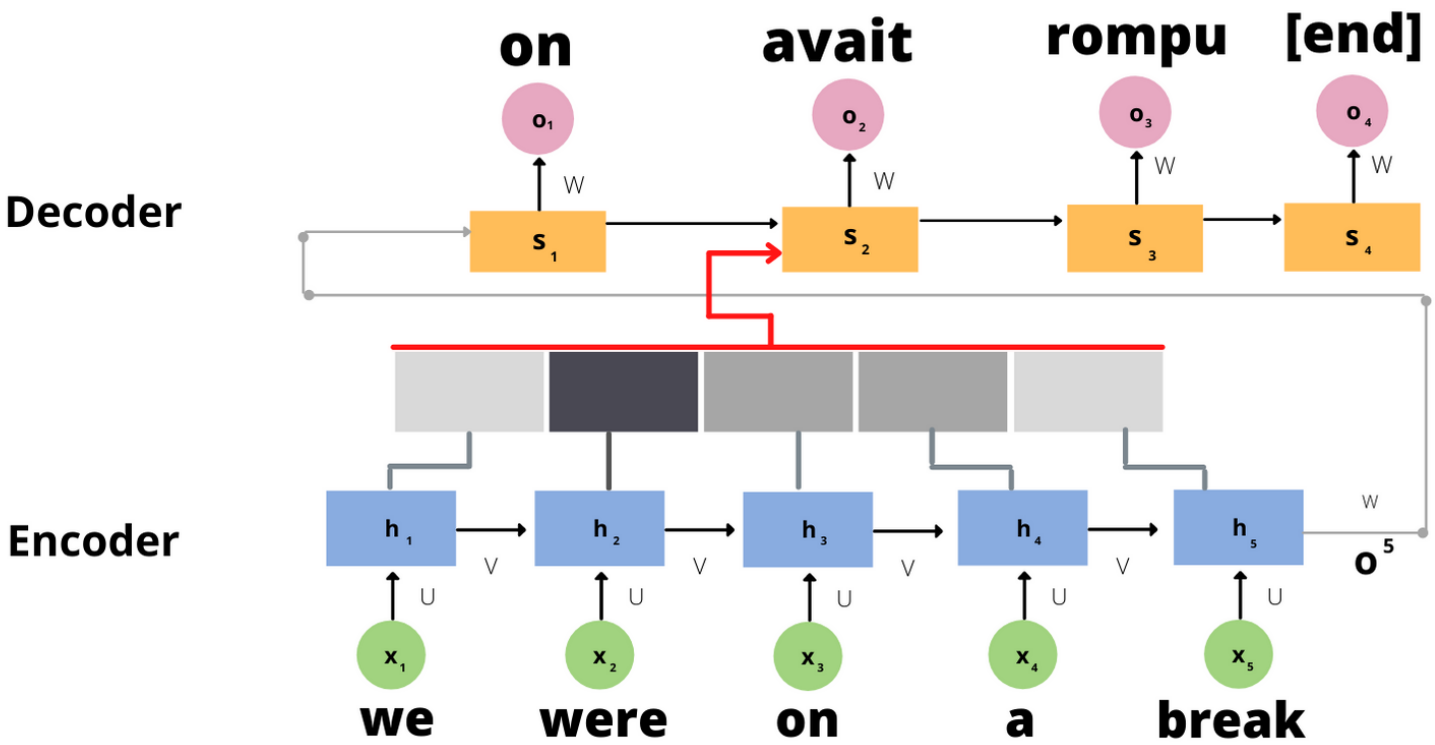
Attention Modules

In other words, attention allows a direct interaction between different token positions by introducing a weighted average of the previous RNN layers into the next one.



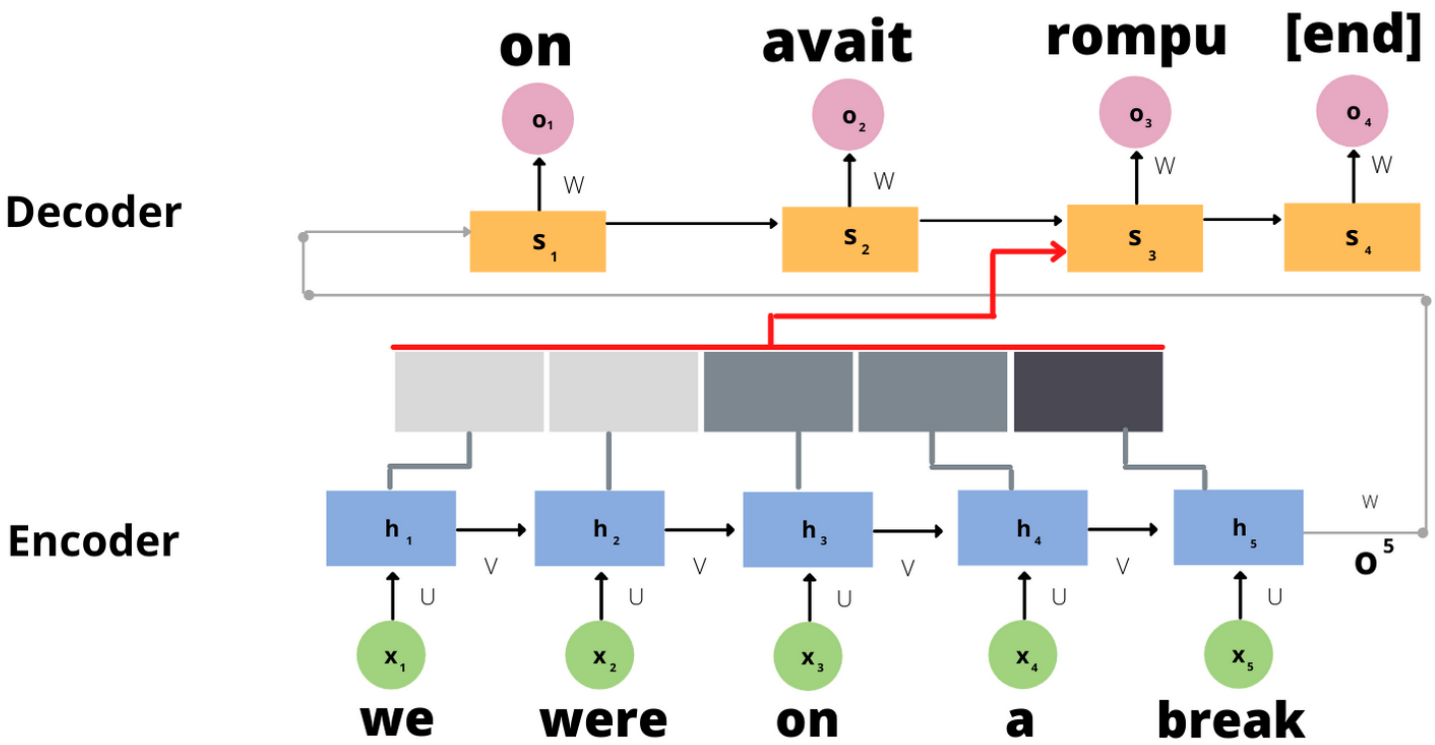
Attention Modules

In other words, attention allows a direct interaction between different token positions by introducing a weighted average of the previous RNN layers into the next one.



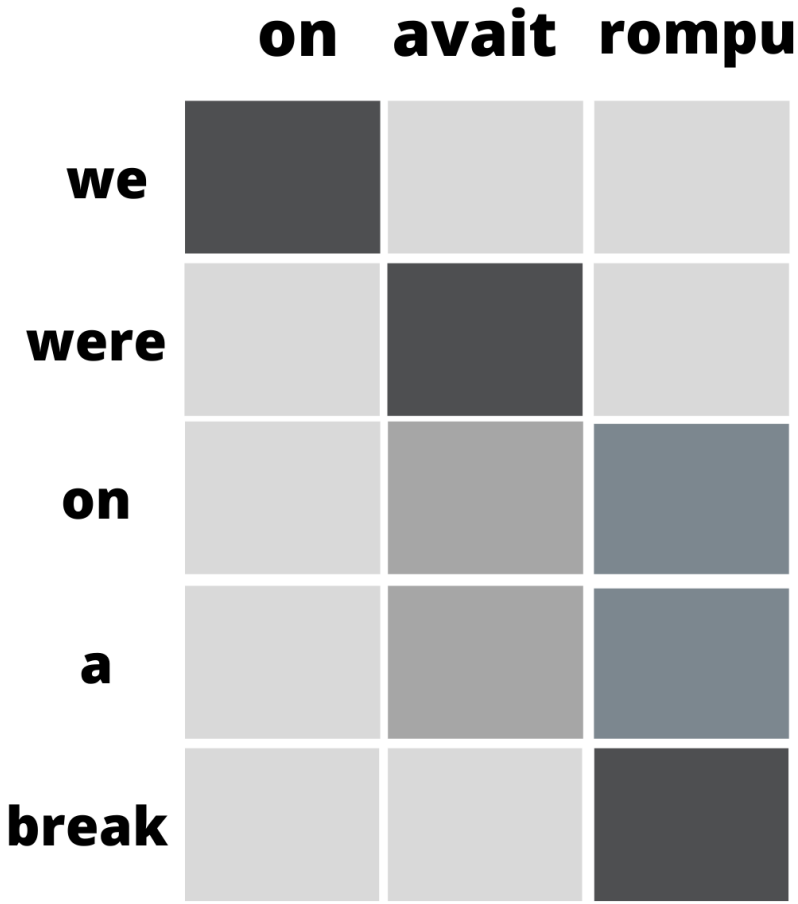
Attention Modules

In other words, attention allows a direct interaction between different token positions by introducing a weighted average of the previous RNN layers into the next one.



Attention Modules

We end up with a structure that resembles a **matrix**. The attention scores encode pairwise dependencies.



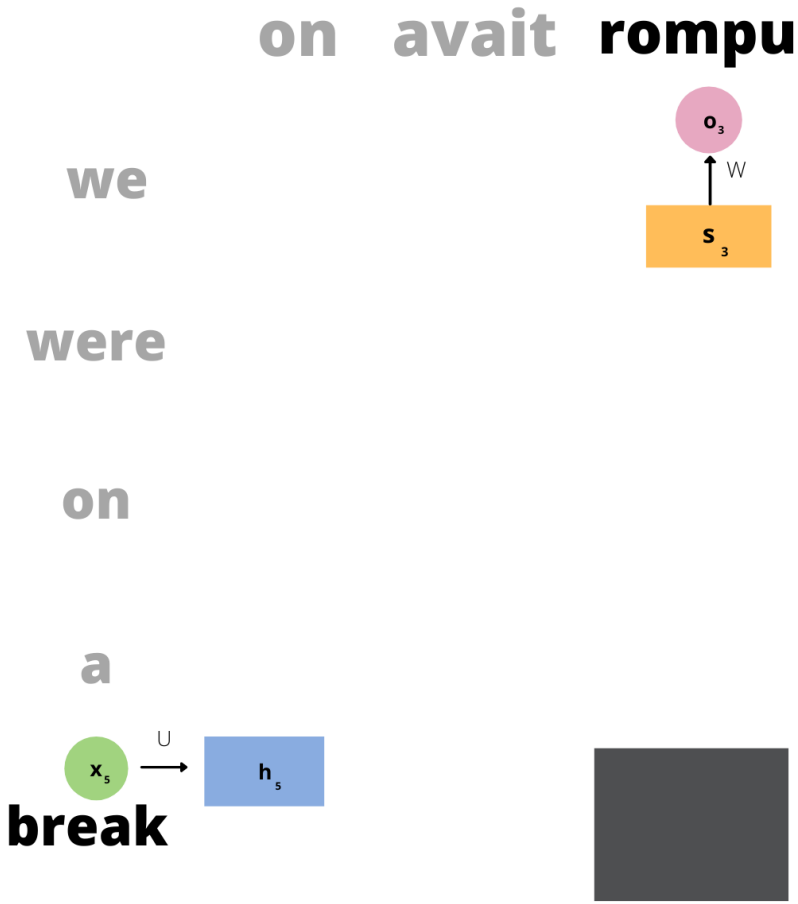
Attention Modules

We end up with a structure that resembles a **matrix**. The attention scores encode pairwise dependencies.
More generally, we will refer to **queries (q)**, **values (v)** and **(keys (k))**. In this case we have:

$$A(q, K, V) = softmax(score(q, K))V$$

$$score(s_t, h_i) = v_a^T tanh(W_a[s_t, h_i])$$

with learnable v_a and W_a. Both $k, v = h_i$ come from the input RNN, while $q = s_t$ comes from the output.



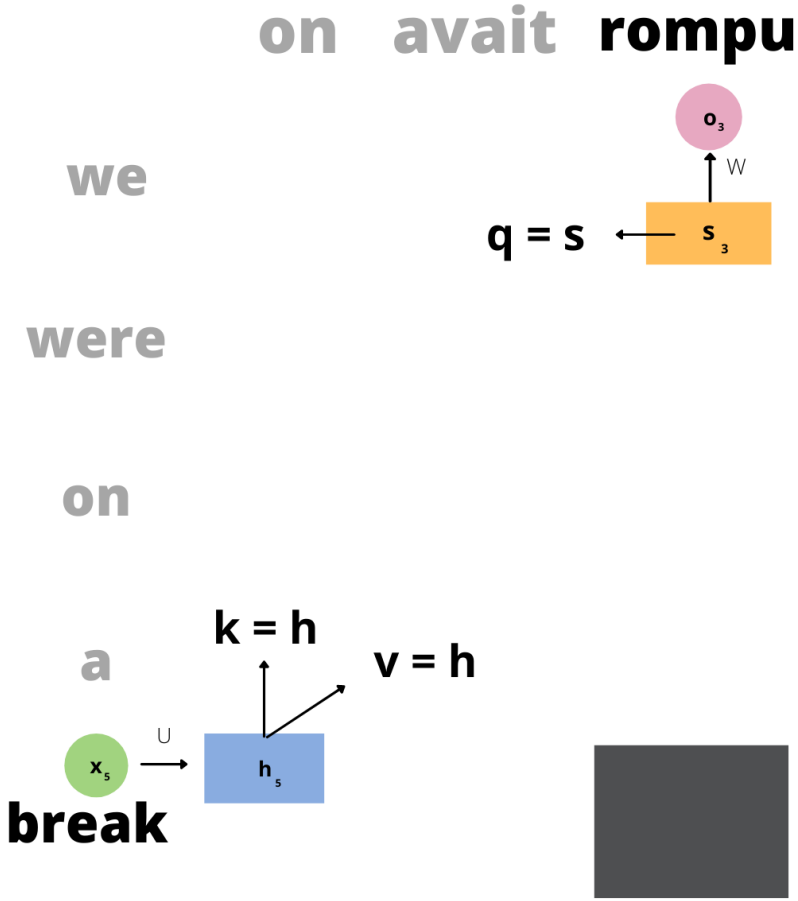
Attention Modules

We end up with a structure that resembles a **matrix**. The attention scores encode pairwise dependencies.
More generally, we will refer to **queries (q)**, **values (v)** and **(keys (k))**. In this case we have:

$$A(q, K, V) = softmax(score(q, K))V$$

$$score(s_t, h_i) = v_a^T tanh(W_a[s_t, h_i])$$

with learnable v_a and W_a. Both $k, v = h_i$ come from the input RNN, while $q = s_t$ comes from the output.



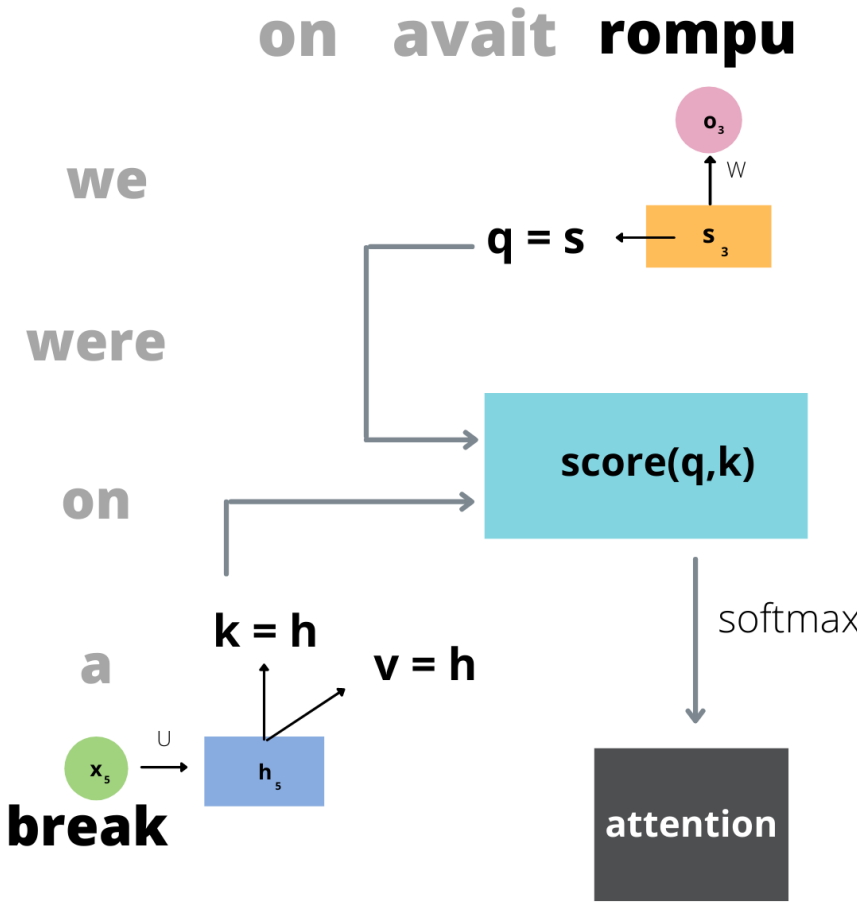
Attention Modules

We end up with a structure that resembles a **matrix**. The attention scores encode pairwise dependencies.
More generally, we will refer to **queries (q)**, **values (v)** and **(keys (k))**. In this case we have:

$$A(q, K, V) = softmax(score(q, K))V$$

$$score(s_t, h_i) = v_a^T tanh(W_a[s_t, h_i])$$

with learnable v_a and W_a. Both $k, v = h_i$ come from the input RNN, while $q = s_t$ comes from the output.



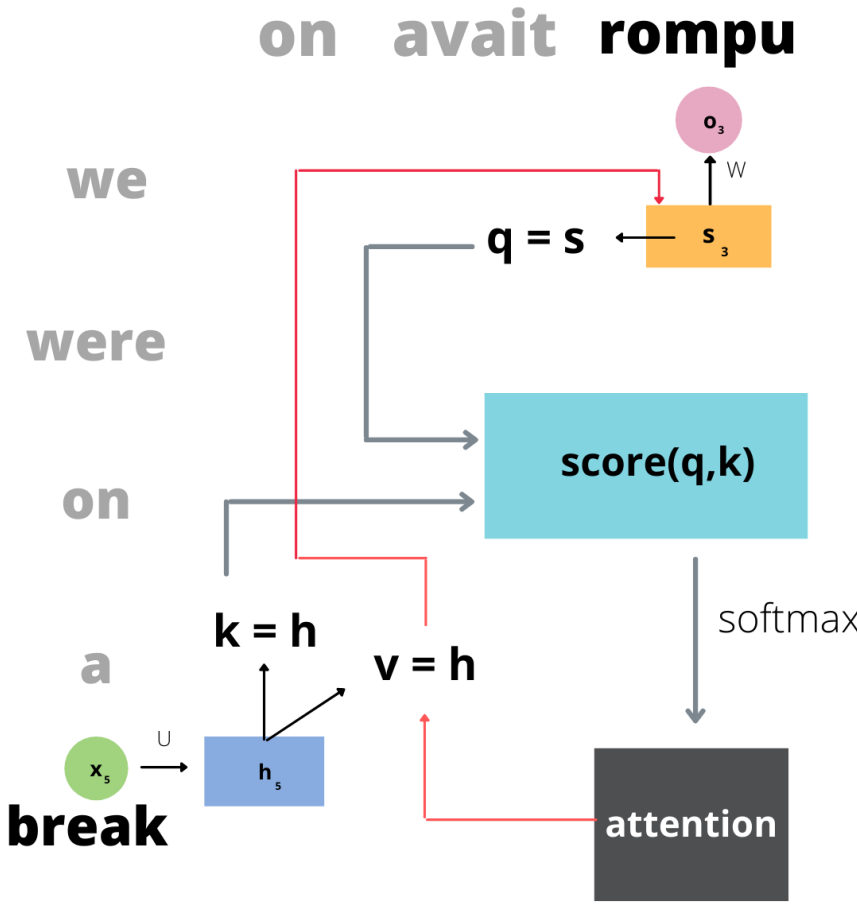
Attention Modules

We end up with a structure that resembles a **matrix**. The attention scores encode pairwise dependencies.
More generally, we will refer to **queries (q)**, **values (v)** and **(keys (k))**. In this case we have:

$$A(q, K, V) = softmax(score(q, K))V$$

$$score(s_t, h_i) = v_a^T tanh(W_a[s_t, h_i])$$

with learnable v_a and W_a. Both $k, v = h_i$ come from the input RNN, while $q = s_t$ comes from the output.



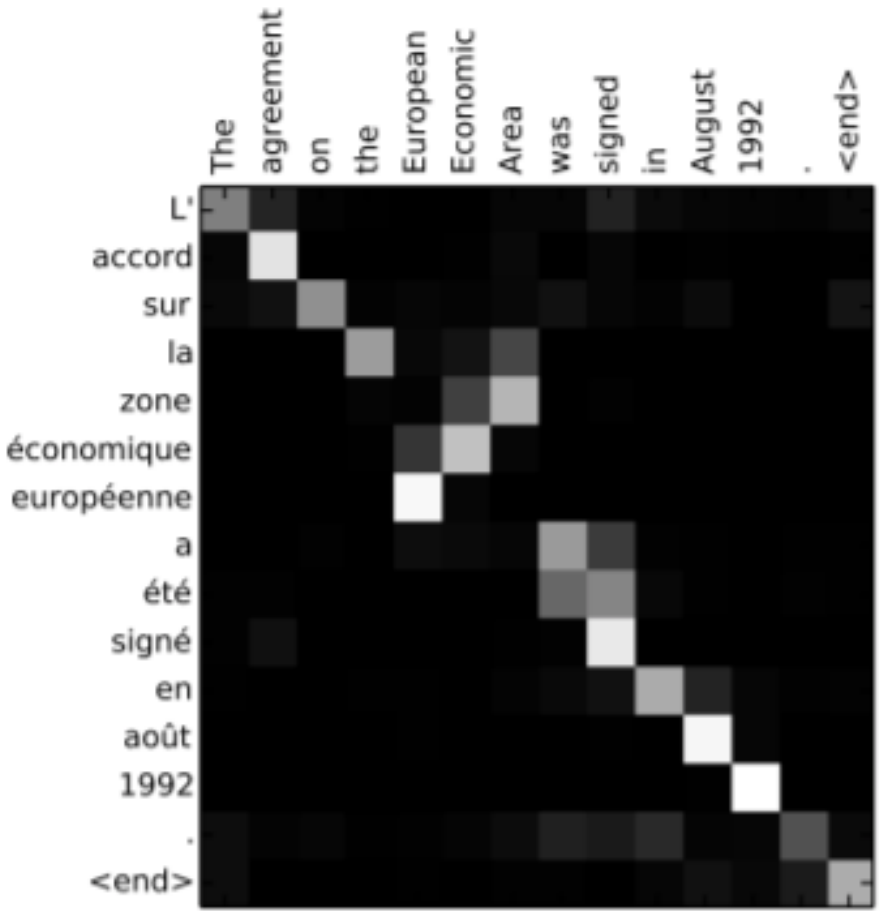
Attention Modules

We end up with a structure that resembles a **matrix**. The attention scores encode pairwise dependencies.
More generally, we will refer to **queries (q)**, **values (v)** and **(keys (k))**. In this case we have:

$$A(q, K, V) = softmax(score(q, K))V$$

$$score(s_t, h_i) = v_a^T tanh(W_a[s_t, h_i])$$

with learnable v_a and W_a. Both $k, v = h_i$ come from the input RNN, while $q = s_t$ comes from the output.



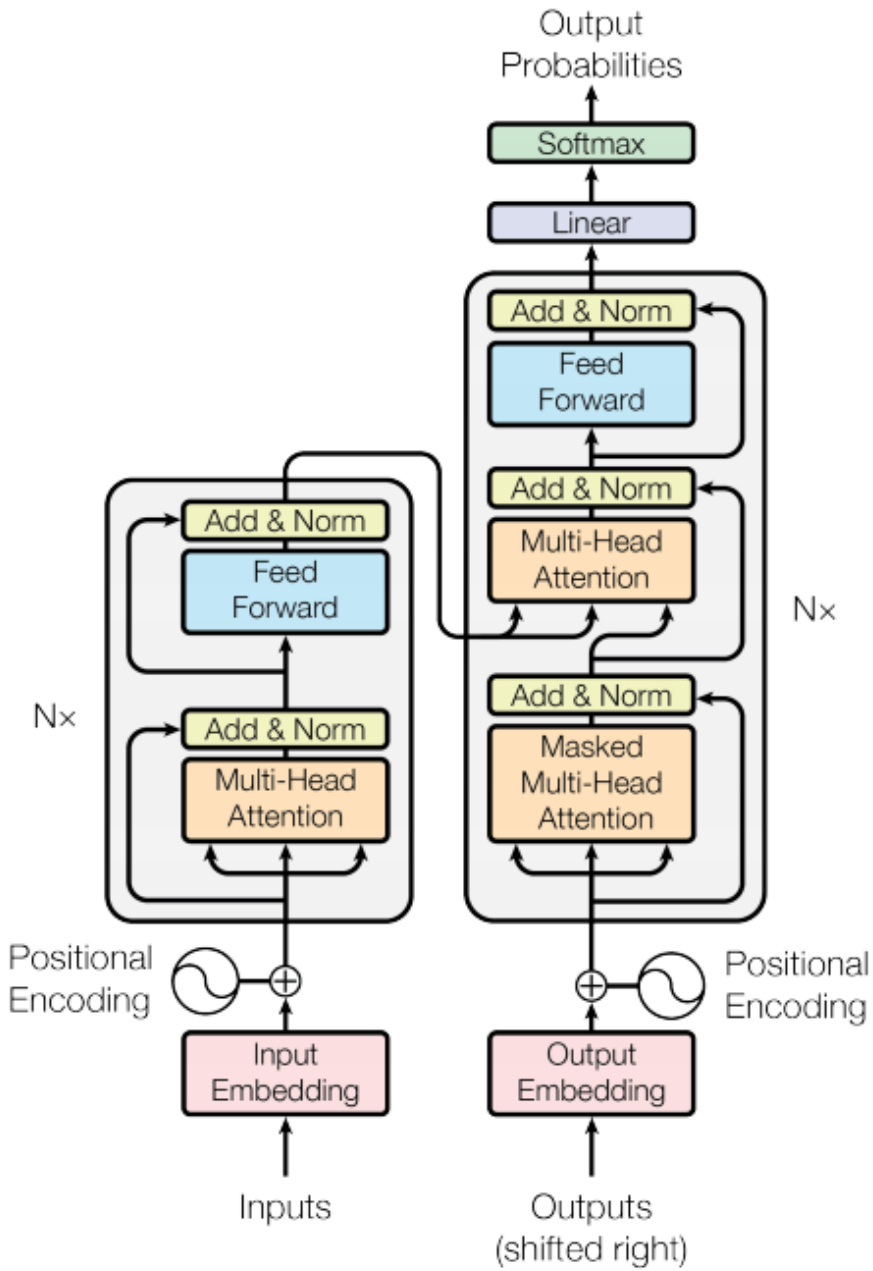
CentraleSupélec

Table of contents

The Transformer

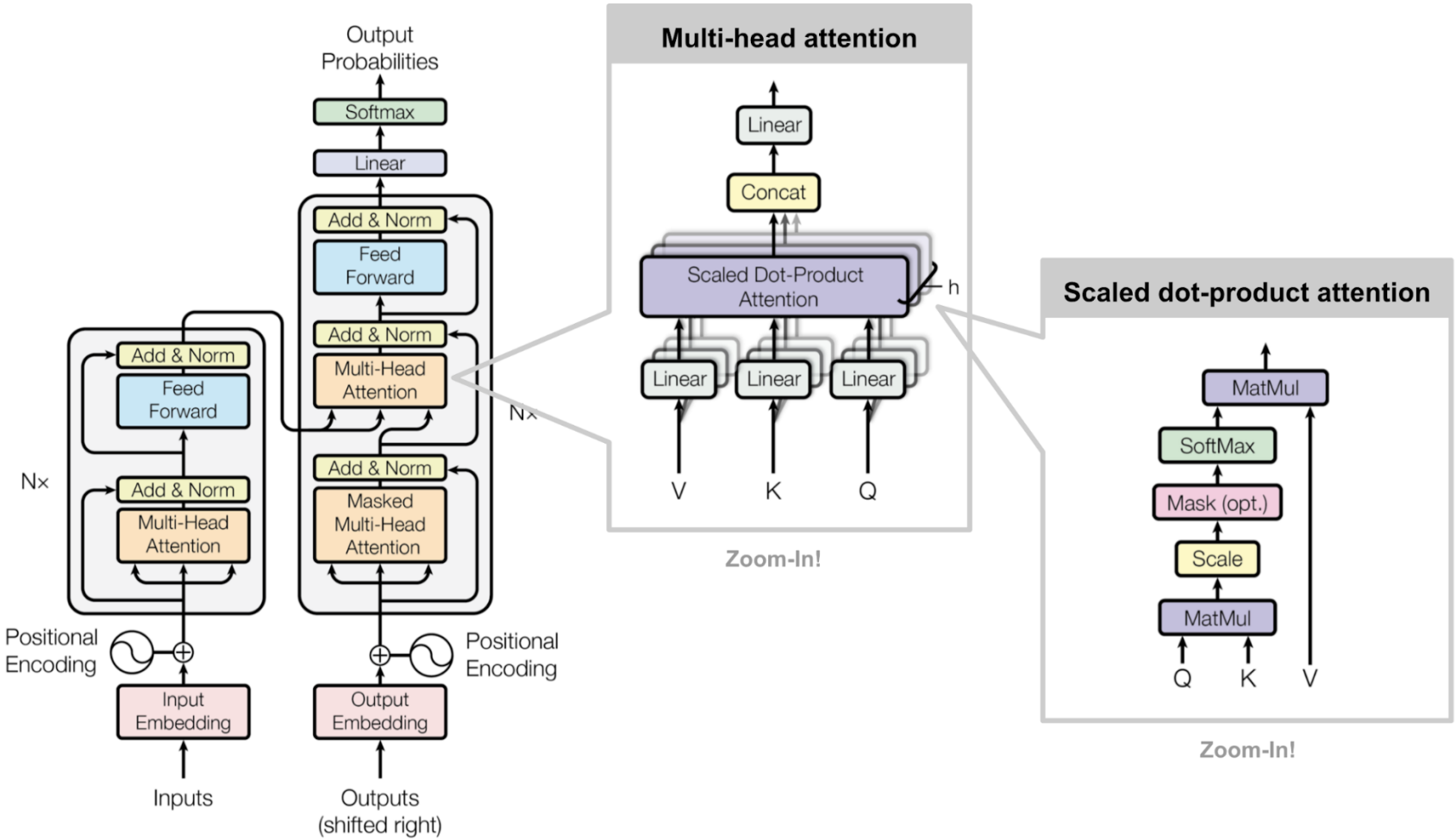
Transformers are a novel architecture that **gets rid of any recurrency**, thus relying solely on **attention modules** [5]. They have the following advantages:

- They allow for **parallelisation**.
- They are **flexible**, thus allowing its use on a wide variety of tasks. They only need to be pre-trained once and then they can be fine-tuned on specific tasks.



Attention in Transformers

[2]



[2] https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#a-family-of-attention-mechanisms

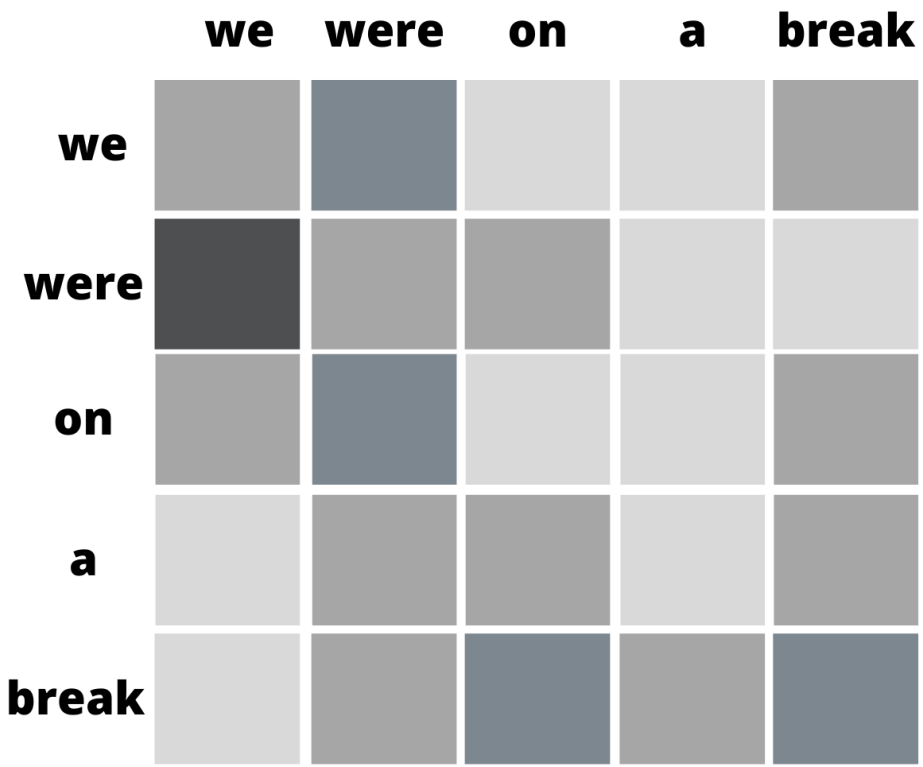
Attention in Transformers

Their novelty in terms of attention architecture is:
The use of **scaled dot-product** score function:

$$score(q, K_i) = \frac{q^T K_i}{\sqrt{d_k}}$$

The use of **multi-head attention**, that is, h attention layers running in parallel.

Q, K and V are computed from input/output using **weight matrices** W_q, W_K and W_V.



Attention in Transformers

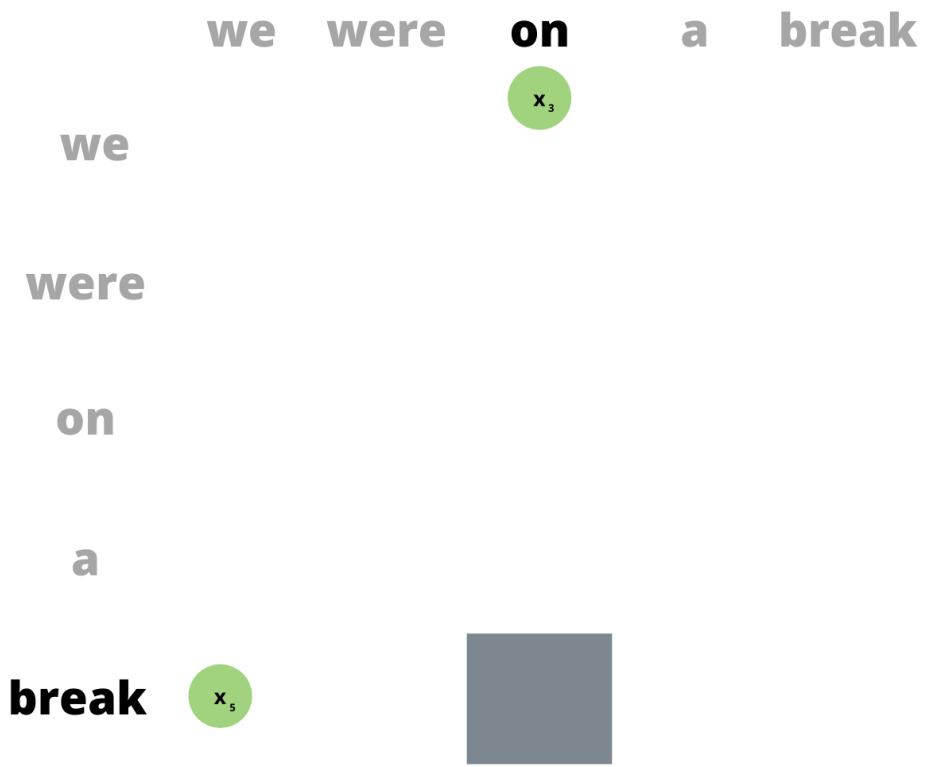
Their novelty in terms of attention architecture is:

The use of **scaled dot-product** score function:

$$score(q, K_i) = \frac{q^T K_i}{\sqrt{d_k}}$$

The use of **multi-head attention**, that is, h attention layers running in parallel.

Q, K and V are computed from input/output using **weight matrices** W_q, W_K and W_V.



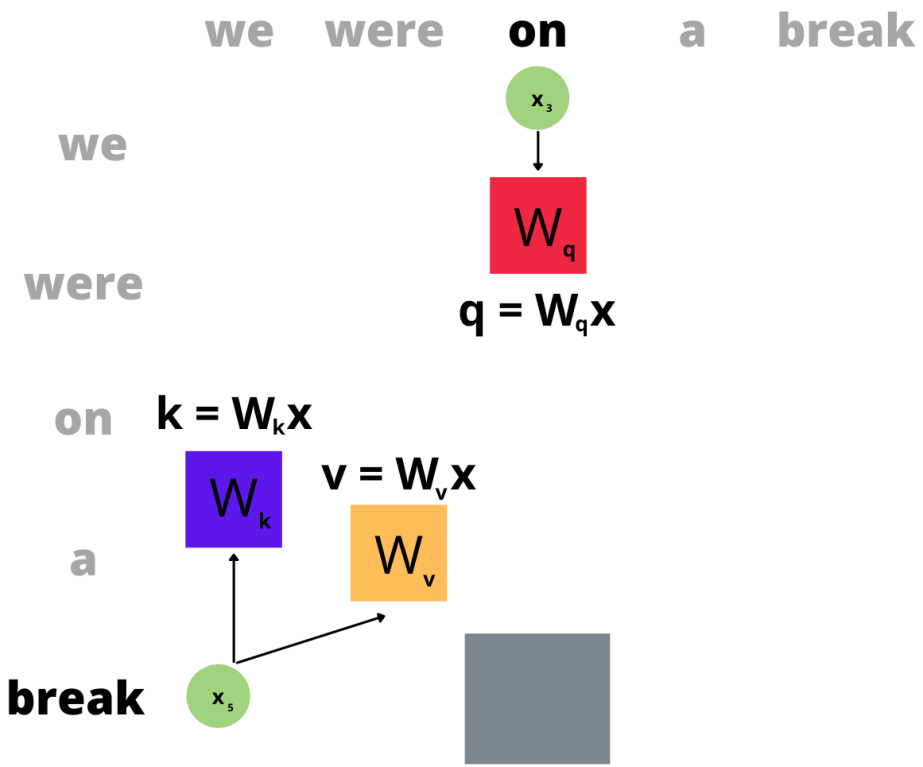
Attention in Transformers

Their novelty in terms of attention architecture is:
The use of **scaled dot-product** score function:

$$score(q, K_i) = \frac{q^T K_i}{\sqrt{d_k}}$$

The use of **multi-head attention**, that is, *h* attention layers running in parallel.

Q, *K* and *V* are computed from input/output using **weight matrices** W_q, W_K and W_V.



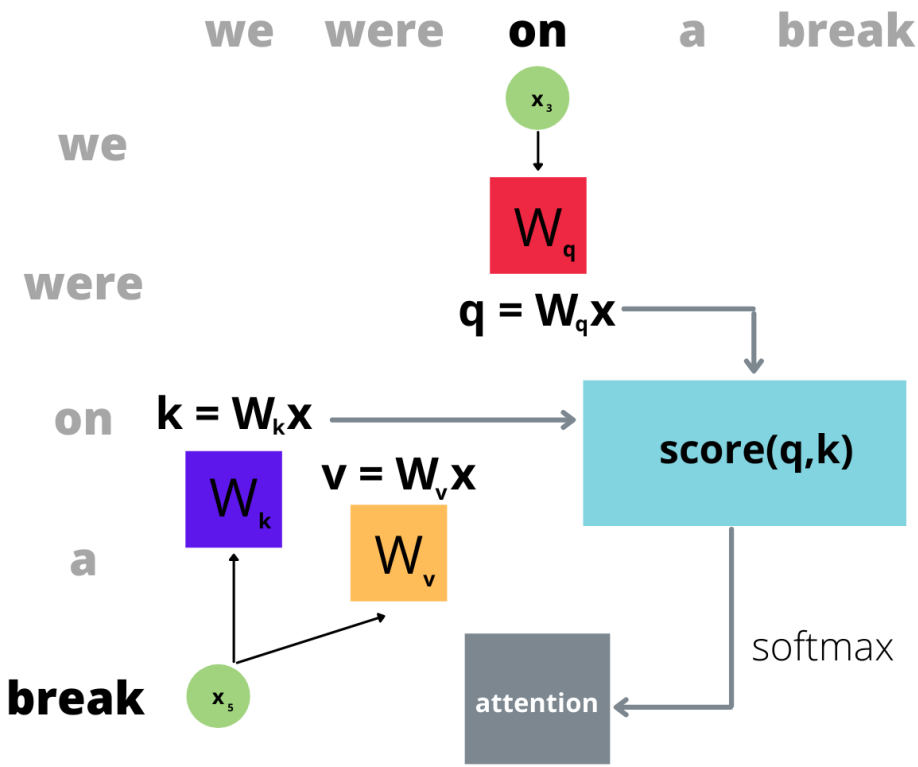
Attention in Transformers

Their novelty in terms of attention architecture is:
The use of **scaled dot-product** score function:

$$score(q, K_i) = \frac{q^T K_i}{\sqrt{d_k}}$$

The use of **multi-head attention**, that is, *h* attention layers running in parallel.

Q, *K* and *V* are computed from input/output using **weight matrices** W_q, W_K and W_V.



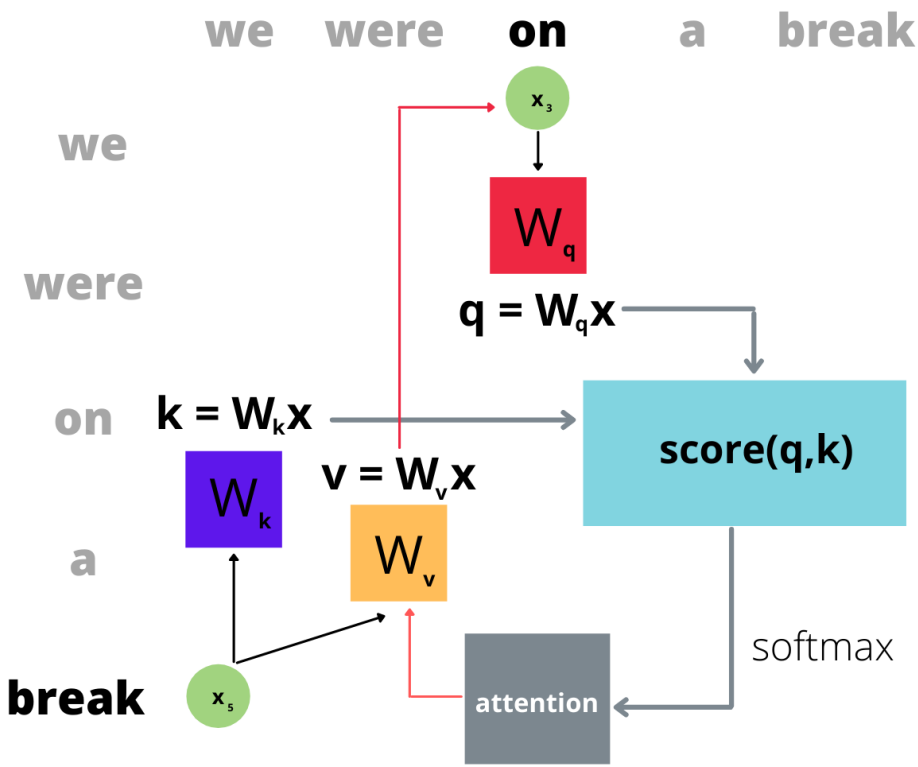
Attention in Transformers

Their novelty in terms of attention architecture is:
The use of **scaled dot-product** score function:

$$score(q, K_i) = \frac{q^T K_i}{\sqrt{d_k}}$$

The use of **multi-head attention**, that is, *h* attention layers running in parallel.

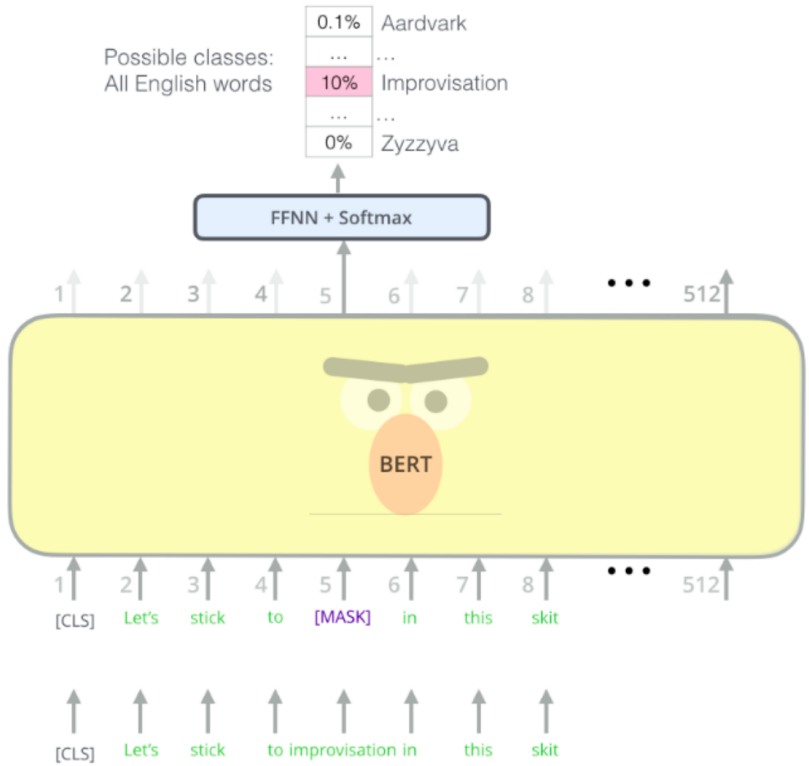
Q, *K* and *V* are computed from input/output using **weight matrices** W_q, W_K and W_V.



BERT: a Transformer based LM

(BERT): A pre-trained Transformer based architecture that achieved state-of-the-art in several tasks, including question answering, text classification and natural language understanding [6][a].

Its pre-training relies on a joint **masked LM** (15% of tokens) and **next sentence prediction** for its pre-training process[b].



[a] Visualising attention: https://huggingface.co/exbert/
[b] Visualisation from: http://jalammar.github.io/illustrated-bert/

Ethical Concerns with LMs

Huge language models can be extremely dangerous [7]:
They come at a **strong environmental cost**.
They are usually trained on data that is **biased**, encoding **racism** and **sexism** in them.
It's **hard to track** and interpret due to their size.

It is important to use this technology **carefully**, always assessing the possible negative outcomes.
Bigger is not always better!

In a Nutshell

- **Language models** are used as a departing point for many NLP applications.
- They are trained with variations of **word prediction** and consuming a lot of data.
- **RNNs** exploit the sequential nature of data by being applied in order to each input word.
- **Attention** improve RNNs by allowing the model to encode complex semantics
- And they are used in **Transformers** for building faster and larger models.
- Nevertheless, we need to be careful when we use these models since they can be **harmful** for certain demographic groups and the environment.

CentraleSupélec

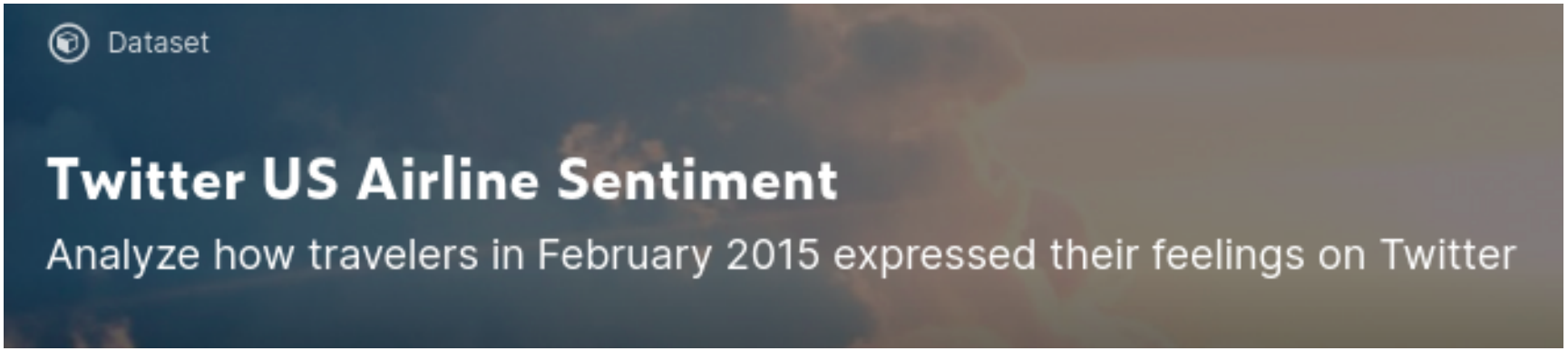
Table of contents

Transformer LMs for Sentiment Analysis

Objective: Students will get familiarised with the use of pre-trained models, how to interpret them and implement them in a simple NLP pipeline.

Tasks: Building a simple sentiment classifier using a pre-trained model from HuggingFace [3].

Visualising the attention weights for model understanding.



[3] https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english/tree/main

References I

[1] Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C. (2003). A Neural Probabilistic Language Model. The Journal of Machine Learning Research, 19.

[2] Elman, J. L. (1990). Finding structure in time. Cognitive science, 14(2), 179-211.

[3] Hochreiter, S., Schmidhuber, J. (1997). Long Short-term Memory. Neural Computation, 9, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[4] Bahdanau, D., Cho, K., Bengio, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate. ArXiv:1409.0473 [Cs, Stat]. http://arxiv.org/abs/1409.0473

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2017). Attention is All you Need. Advances in Neural Information Processing Systems, 30, 5998–6008. https://arxiv.org/abs/1706.03762

[6] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 4171–4186. https://doi.org/10.18653/v1/N19-1423

[7] Bender, E. M., Gebru, T., McMillan-Major, A., Shmitchell, S. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, 610–623. http://faculty.washington.edu/ebender/papers/Stochastic_Parrots.pdf

CentraleSupélec

Introduction to Computational Linguistics

Part 2: Language Models

Camilo Carvajal Reyes

3rd March 2021