Clase 7: Modelos Auto-regresivos

MDS7203 Modelos Generativos Profundos

Felipe Tobar
Cristóbal Alcázar - Camilo Carvajal Reyes

Iniciativa de Datos e Inteligencia Artificial
Universidad de Chile

22 de septiembre 2023

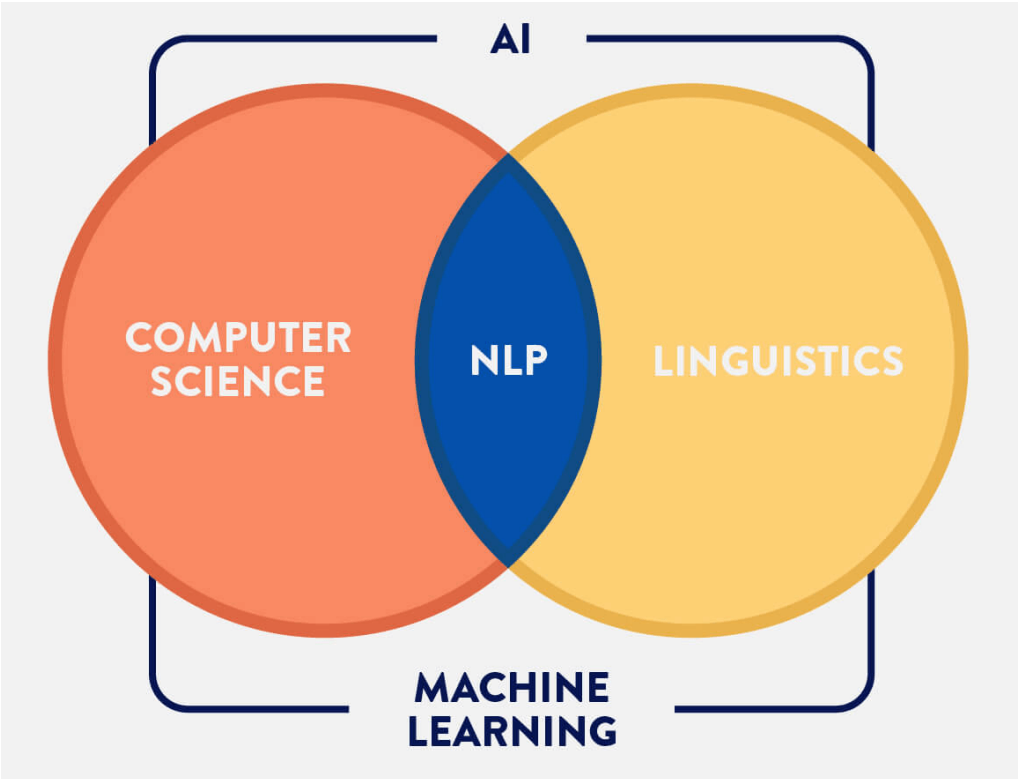
Tabla de contenidos

Table of contents

What is NLP?

Natural Language Processing is a sub-field of linguistics, computer science and artificial intelligence, that connects computers and human language.

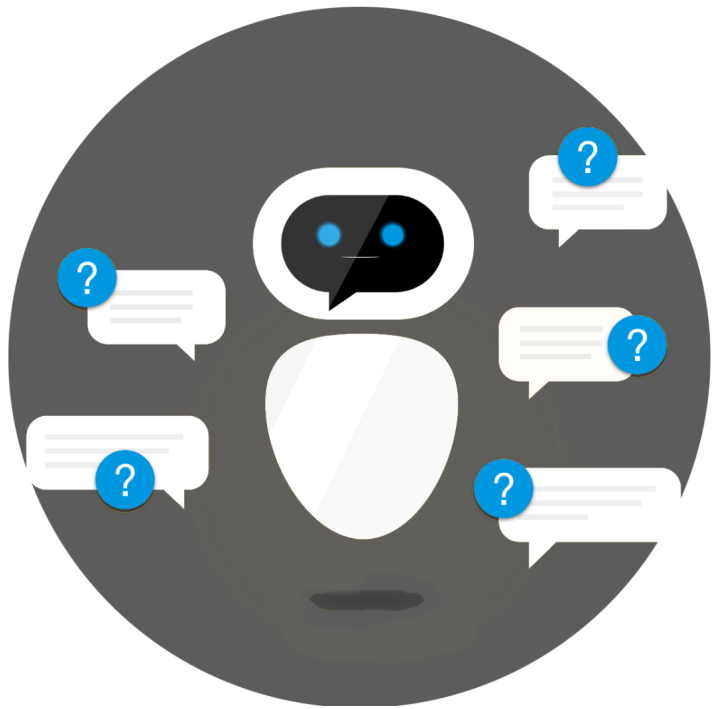
The field has experienced major changes since recent growth of **Deep Learning**.



Why bothering at all with NLP?

Computational encoding of text opens the gate for:

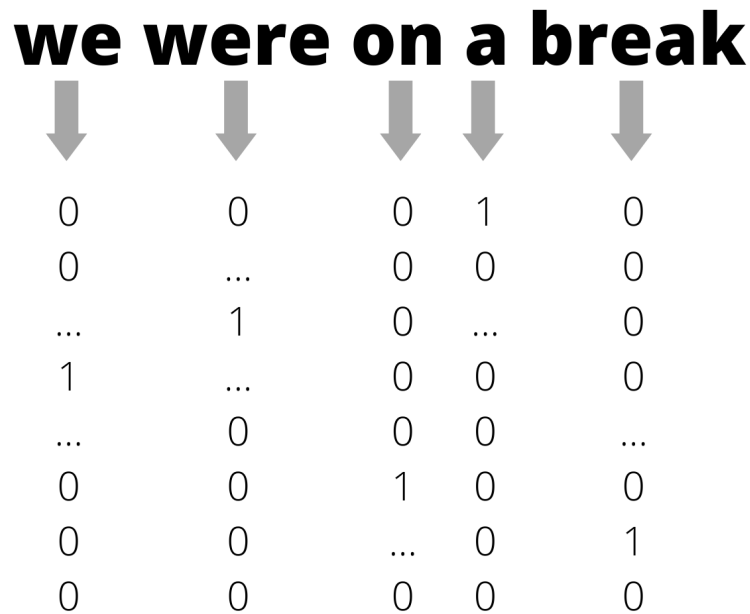
- Machine Translation
- Natural Language Understanding
- Automatic Text Summarisation
- Computational Linguistics
- Natural Language Generation
- Question-Answering
- Sentiment Analysis



Representation of text

One hot-encoding

It is one possible approach for giving a vector to a word. Let's consider a vocabulary of possible words V with n terms. A one-hot representation for the word w, indexed by i in the vocabulary will correspond to the vector that has only zeros, except for the position i. That way, we can distinguish two different words.



Representation of text

Bag-of-words
Corresponds to a document representation in which we add up the corresponding one hot-encoding vectors. The result is an n-dimensional vector with the frequency of each word.
We may also consider a **vector of occurrence**, in which the i-th position has 1 if the word is in the document, regardless of the repetitions

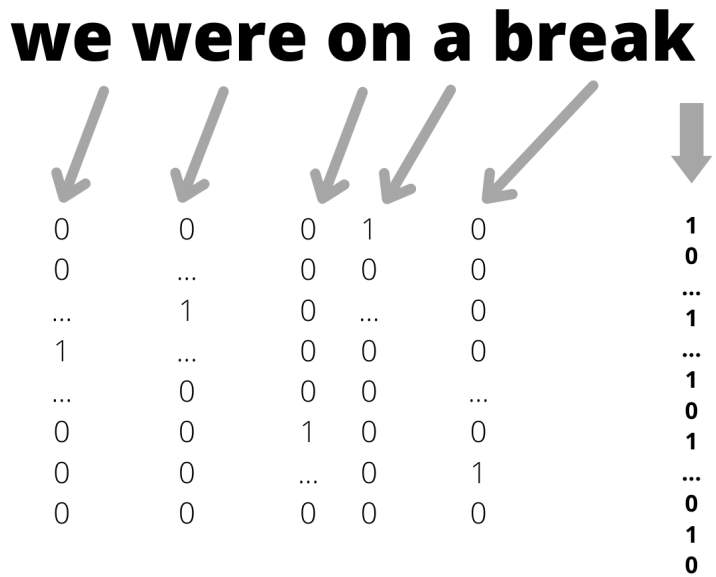


Table of contents

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`

Local Normalisation

Some LMs we will study correspond to deep auto-regressive models, i.e., generative models that predict future values of a sequence given its past. Formally, we are concerned with thode modelling probabilities in the following way:

$$
\begin{aligned}
p_{ML}(x_{p+1}, \ldots, x_T | x_1, \ldots, x_p) &= \prod_{i=p+1}^{T} p(x_i | x_1, \ldots, x_{i-1}) \\
&= \prod_{i=p+1}^{T} \frac{s(x_1, \ldots, x_p, \ldots, x_i)}{\sum_{y \in V} s(x_1, \ldots, x_p, \ldots, x_{i-1}, y)} ,
\end{aligned}
$$

where $s(x_1, \ldots, x_p, \ldots, x_{i-1}, y)$ is a score of a token y given the input $x_1, \ldots, x_p$ and the prediction history $x_{p+1}, \ldots, x_{i-1}$. This is called "local normalisation". This way, models are capable of producing language in a sequential way.

Neural probabilistic LMs

In Neural LM we combine vectors using **neural networks**. Literally from Bengio et al. 2003:

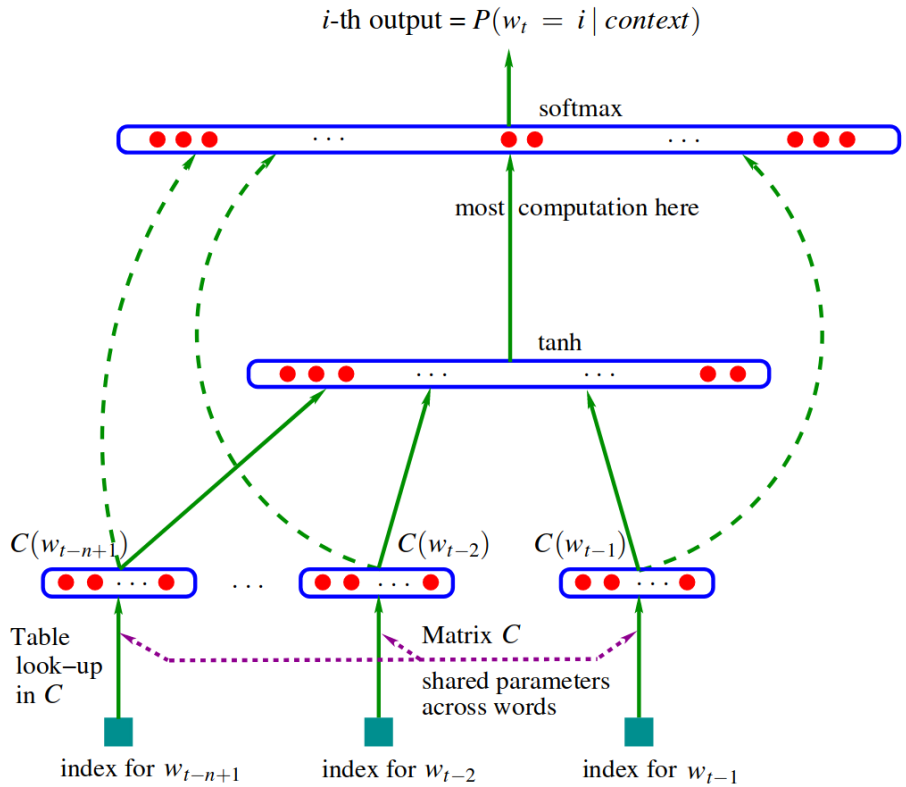
1. associate with each word in the vocabulary a distributed word feature vector (a real valued vector),
2. express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the word feature vectors and the parameters of that probability function.

Neural probabilistic LMs

Results depend heavily on the architecture to use. The simplest option is using a simple **Multi-layer perceptron** as in Bengio et al.. However, this is arguably not the best option to encode language.

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

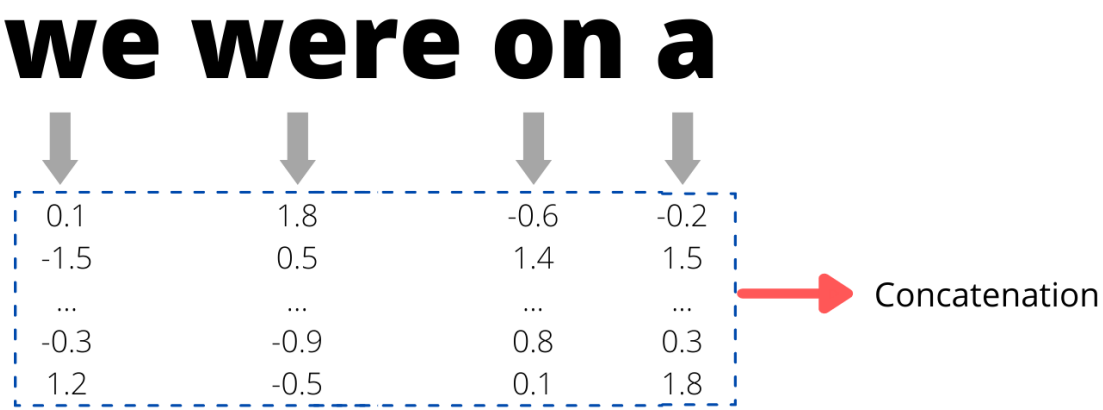
- Recurrent Neural Networks (RNN)
- Transformers



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. 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:

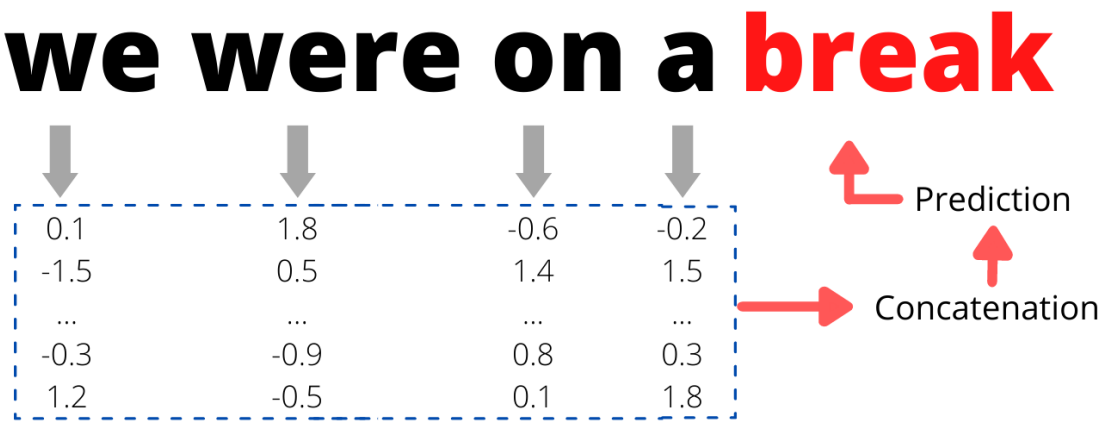
- **Next word prediction (NWP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. 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

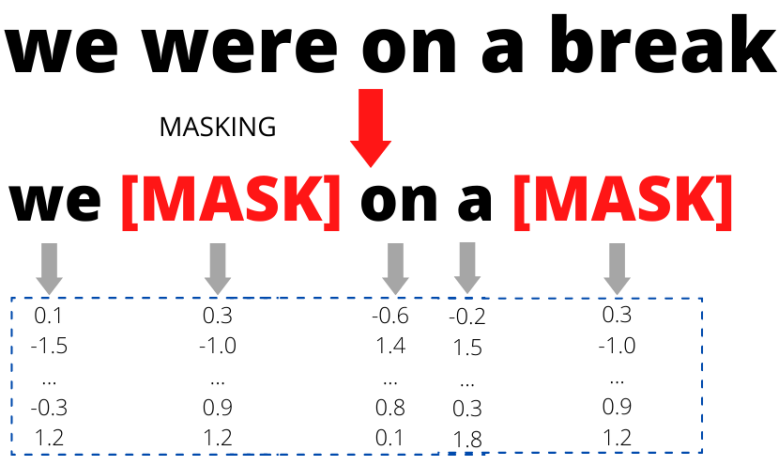
Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

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

Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. 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:

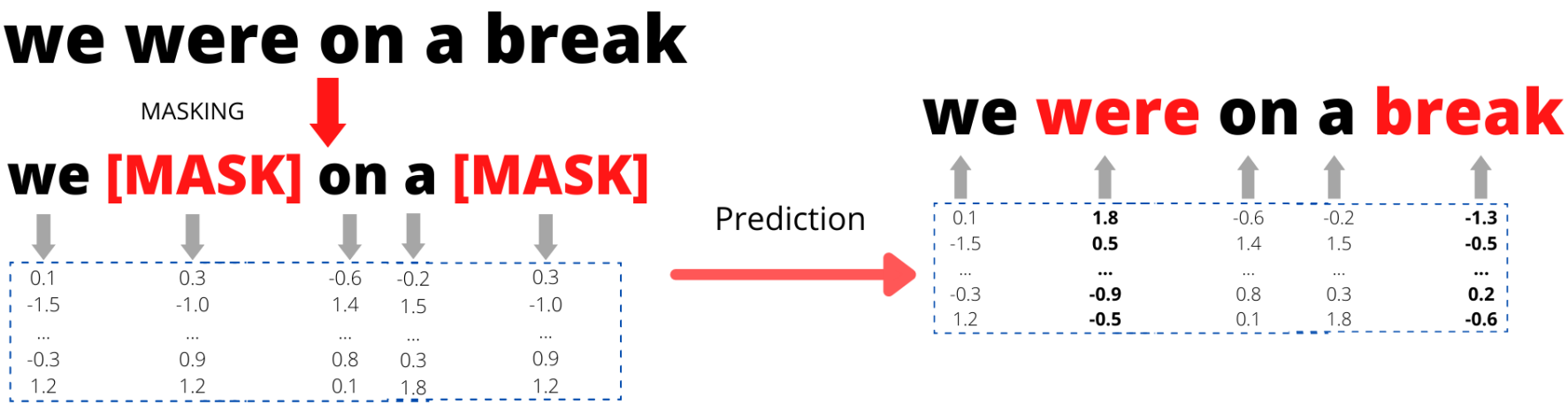
- **Next word prediction (NWP)**
- **Masked Language Model (MLM)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. 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:

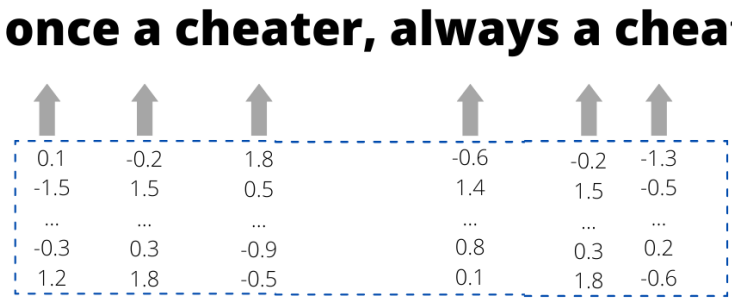
- **Next word prediction (NWP)**
- **Masked Language Model (MLM)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. 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:

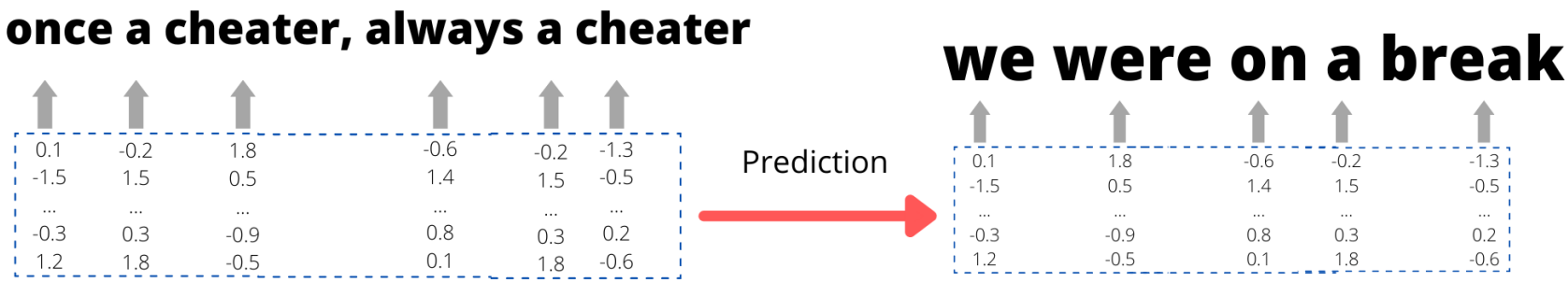
- **Next word prediction (NWP)**
- **Masked Language Model (MLM)**
- **Next sentence prediction (NSP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models**. 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)**
- **Masked Language Model (MLM)**
- **Next sentence prediction (NSP)**



Evaluation: Perplexity

Perplexity (PPL) is a widely used evaluation metric for language models, specially when the target task is text generation. It is defined as

$$2^{-\frac{1}{T-p}\sum_{i=p+1}^{T}\log_2 p(x_i|x_{i-1},\ldots,x_1)}.$$

It can be interpreted as the mean of the number of tokens for which the model is unsure at each generation step. This is equivalent to the exponentiation of the cross-entropy between the data and model predictions[2].

We are often maximising other functions when fitting language models, for instance when using it for other NLP tasks, for which domain specific target metrics exist.

[2] `https://huggingface.co/docs/transformers/perplexity`

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.. 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

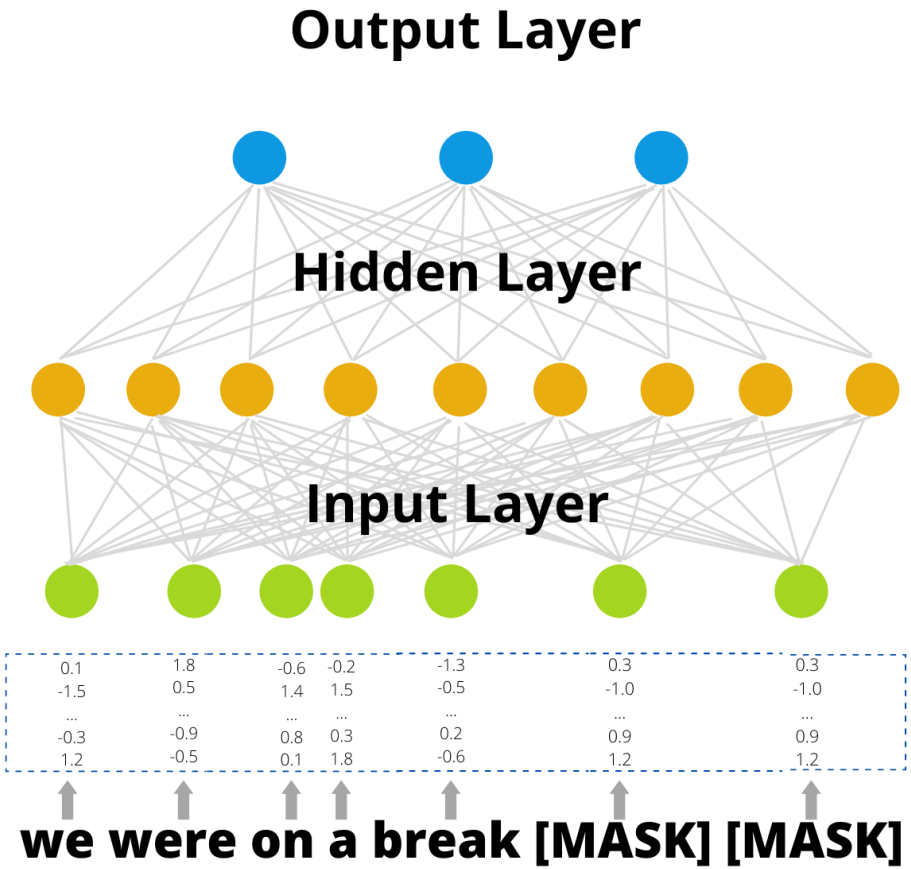
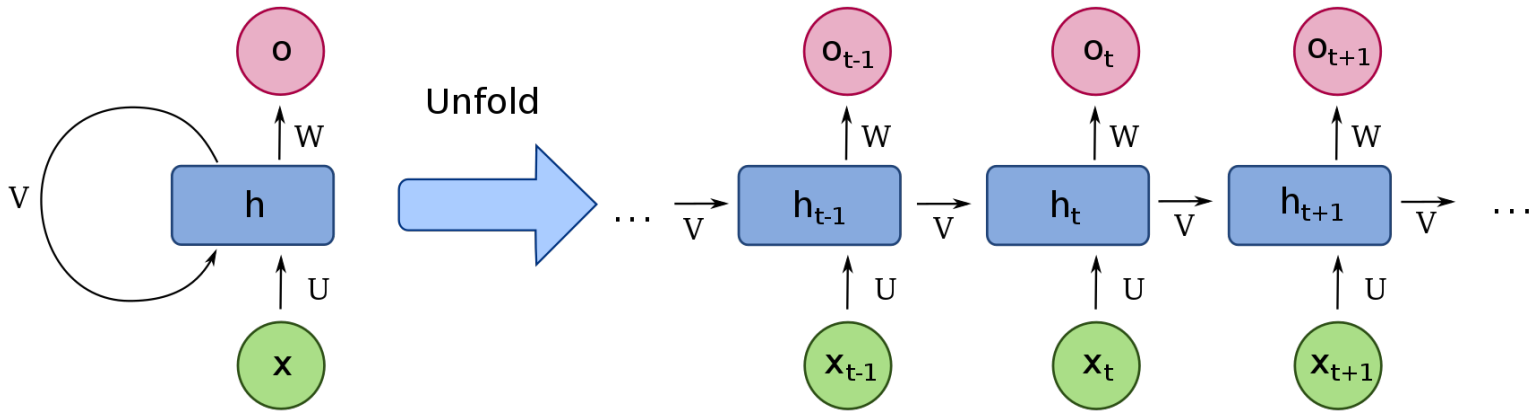


Table of contents

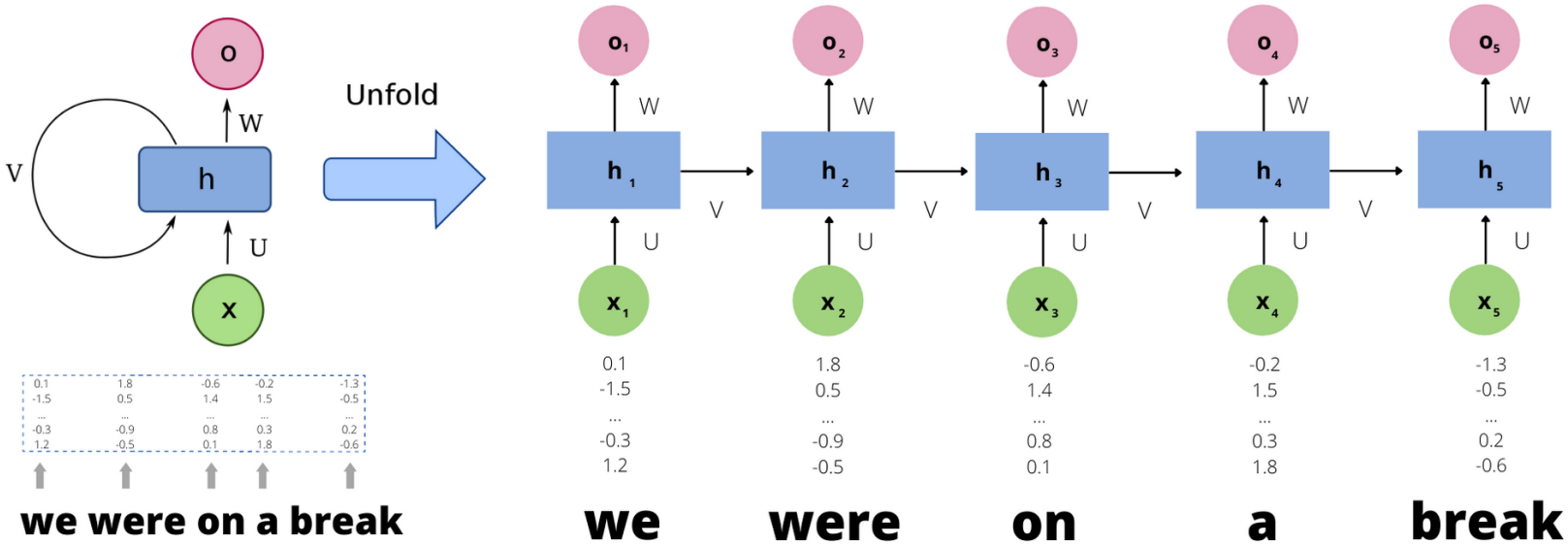
Recurrent Neural Networks

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.



Recurrent Neural Networks

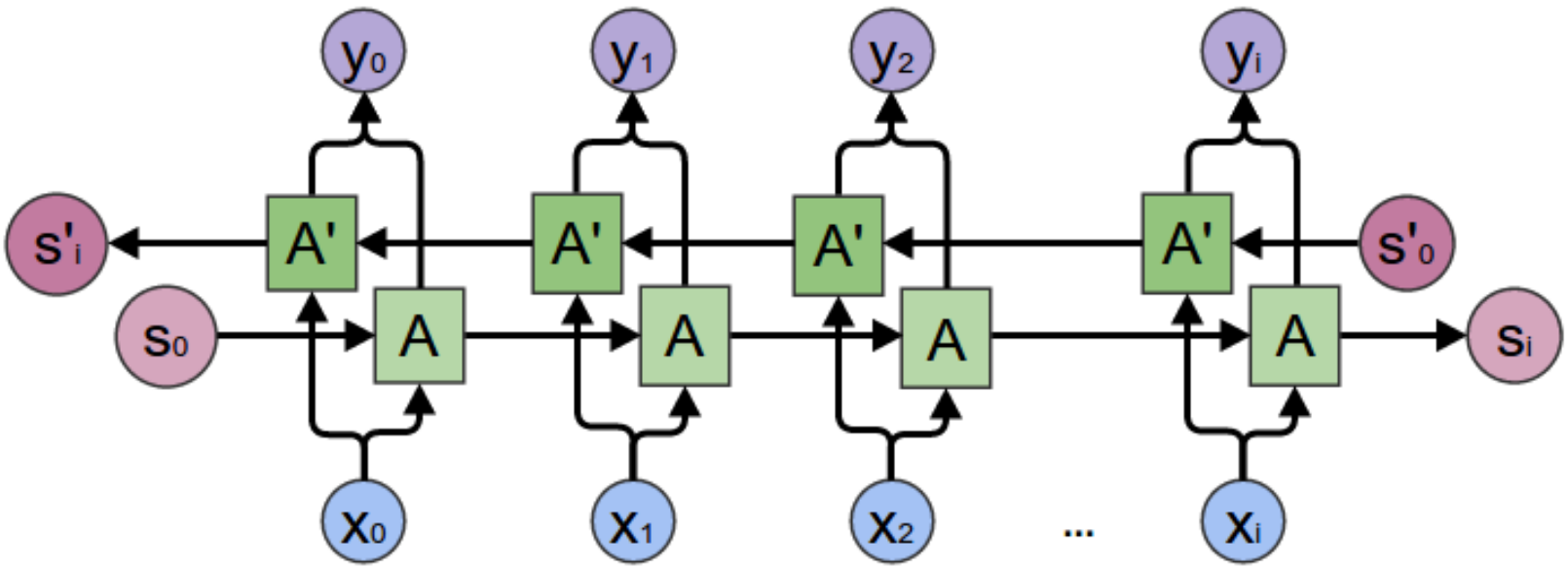
RNNs are an ideal solution to deal with sequences of unknown length. In its simple version (**Vanilla RNN**) we simply apply a neural network to generate an **output** and a hidden state to every member of the sequence.



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.

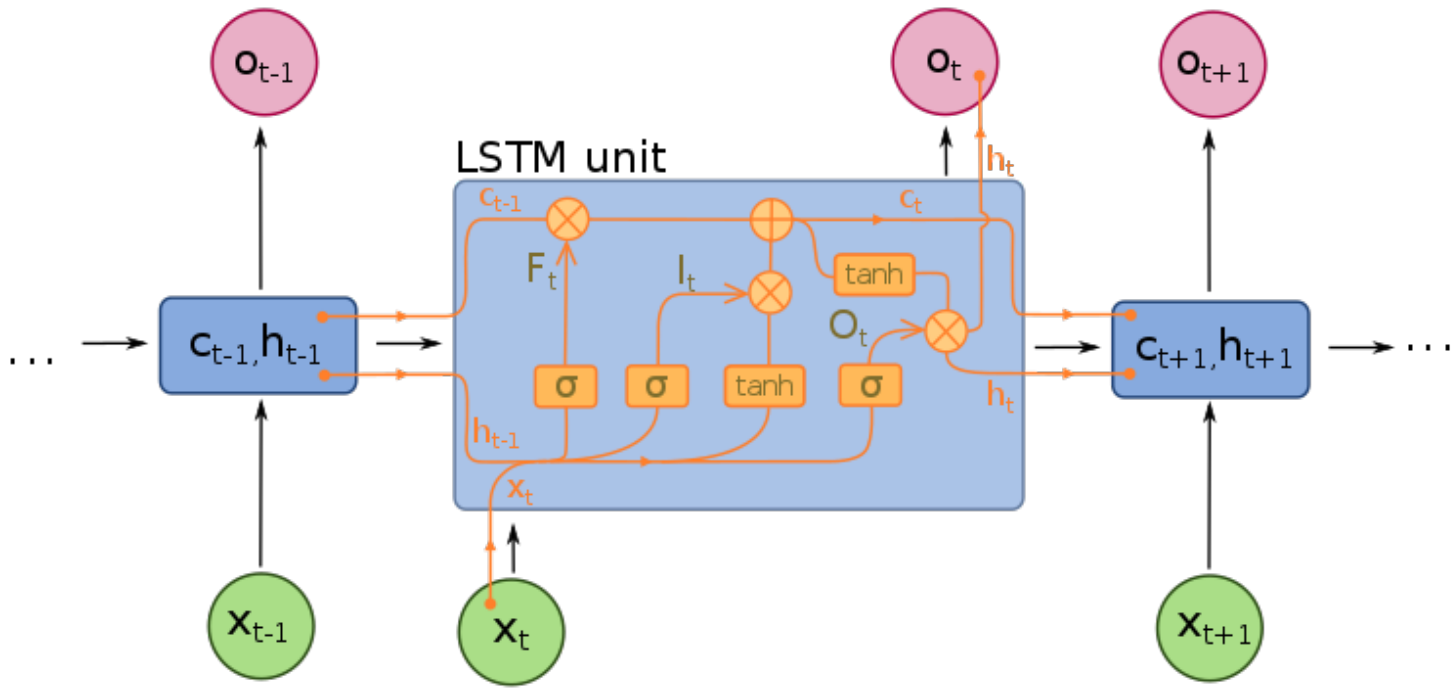
In reality, we often apply **bi-directionality**.



Bi-directional LSTM

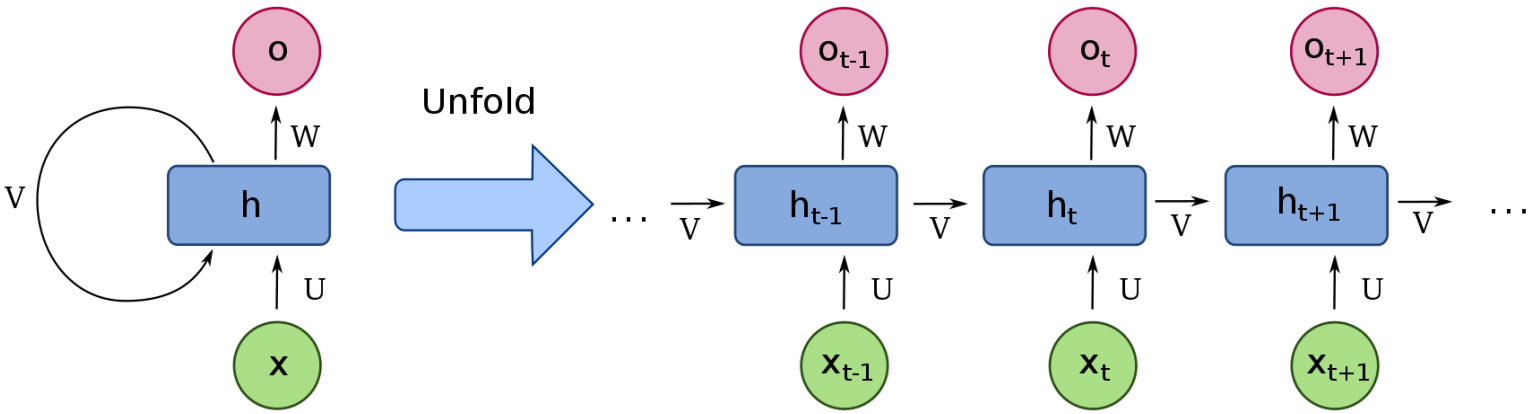
RNNs are not perfect: in particular, they suffer from the **vanishing gradient problem**. This is, when we back-propagate the error to update the weights, we loose information, particularly with long sequences.

In reality, we often apply **bi-directionality** and we use a more complex architecture instead: the **Long Short Term Memory Network**.



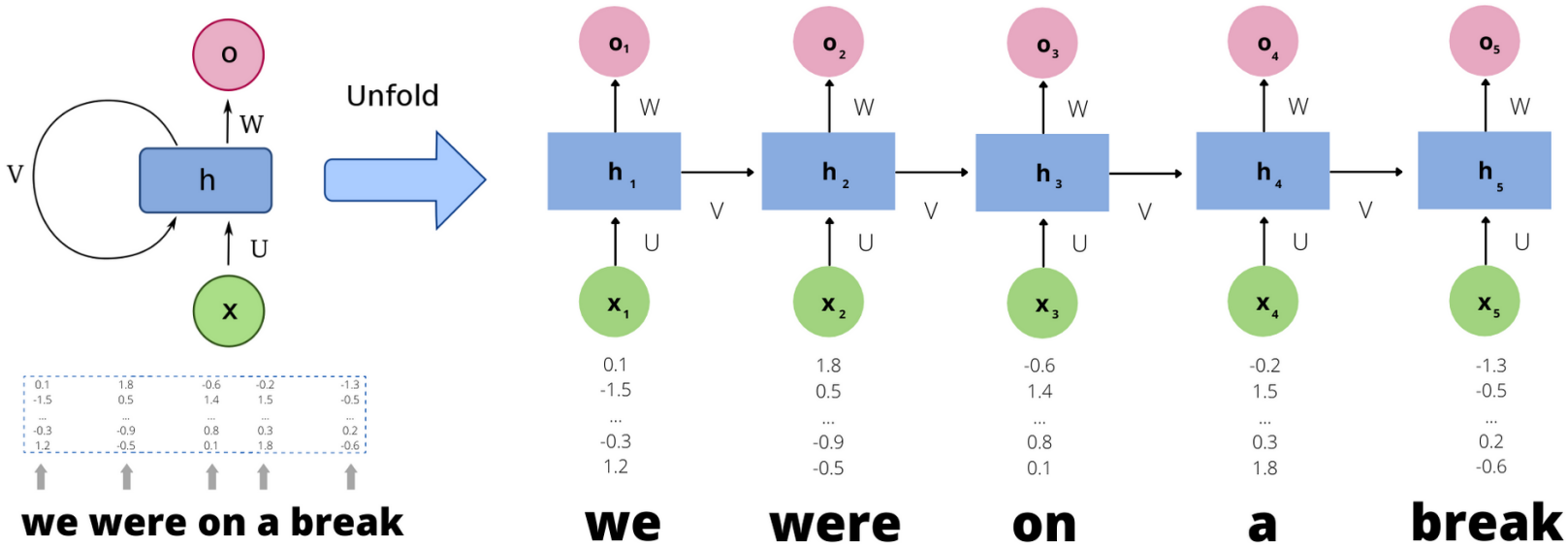
Recurrent Neural Networks

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.



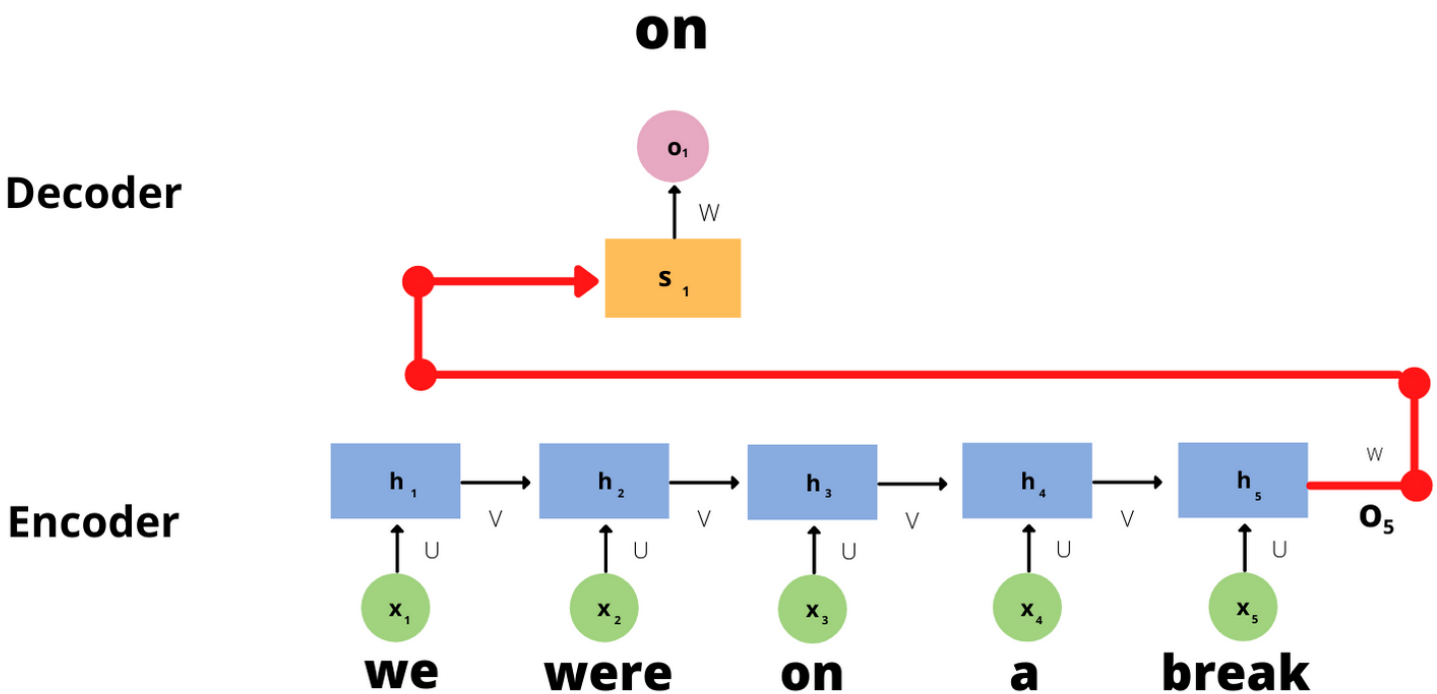
Recurrent Neural Networks

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.



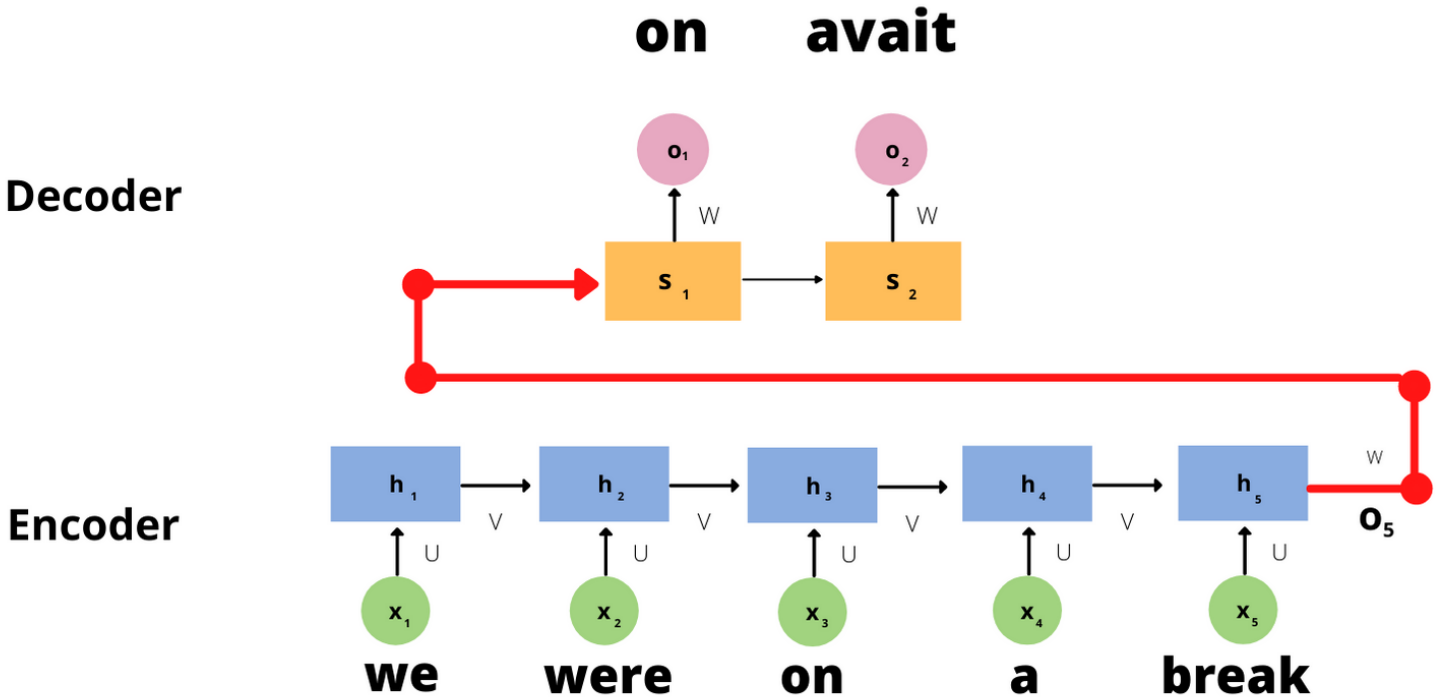
RNNs: a Machine Translation example

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.



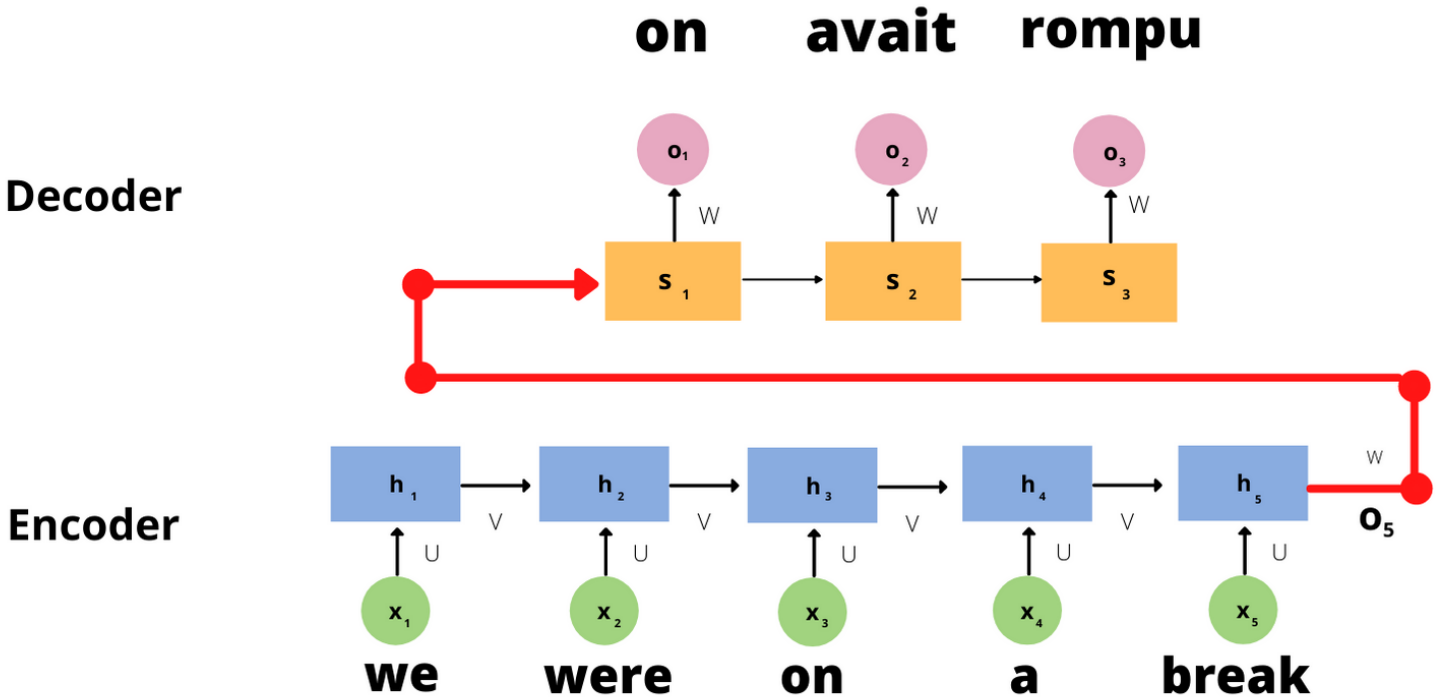
RNNs: a Machine Translation example

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.



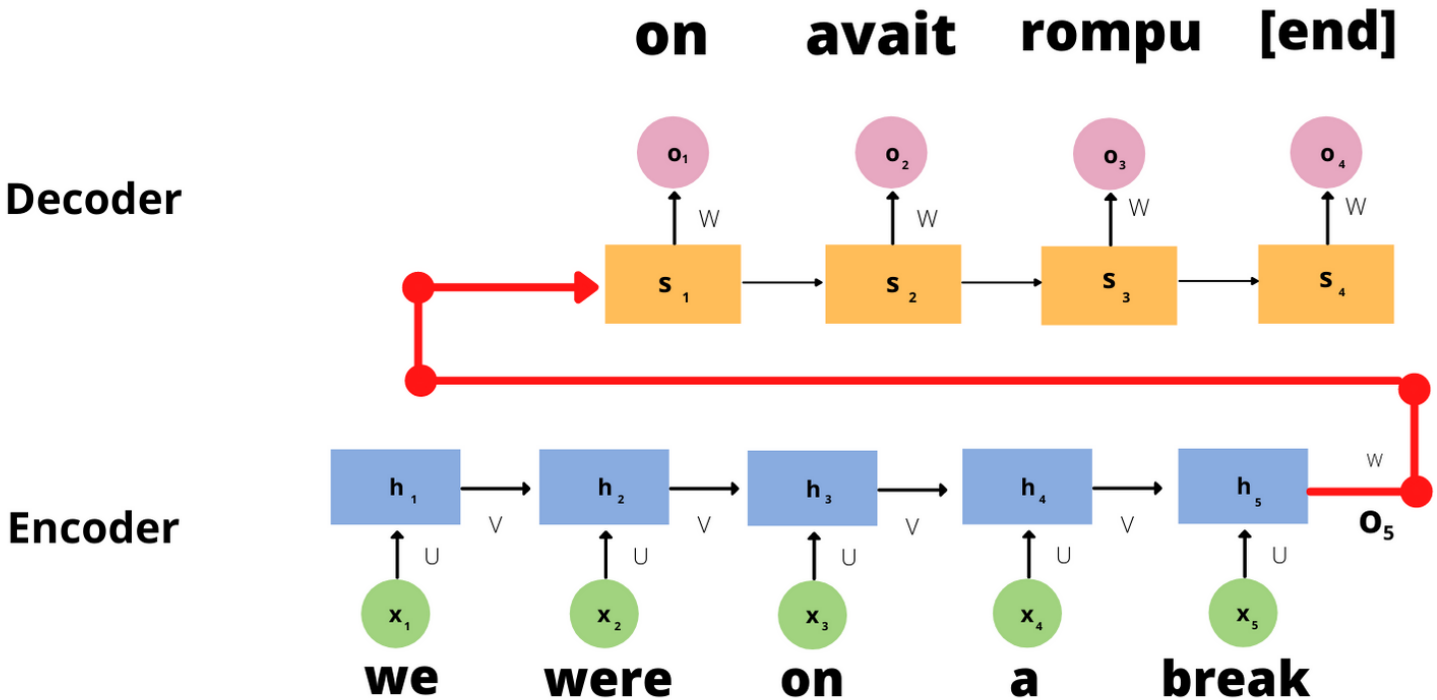
RNNs: a Machine Translation example

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.



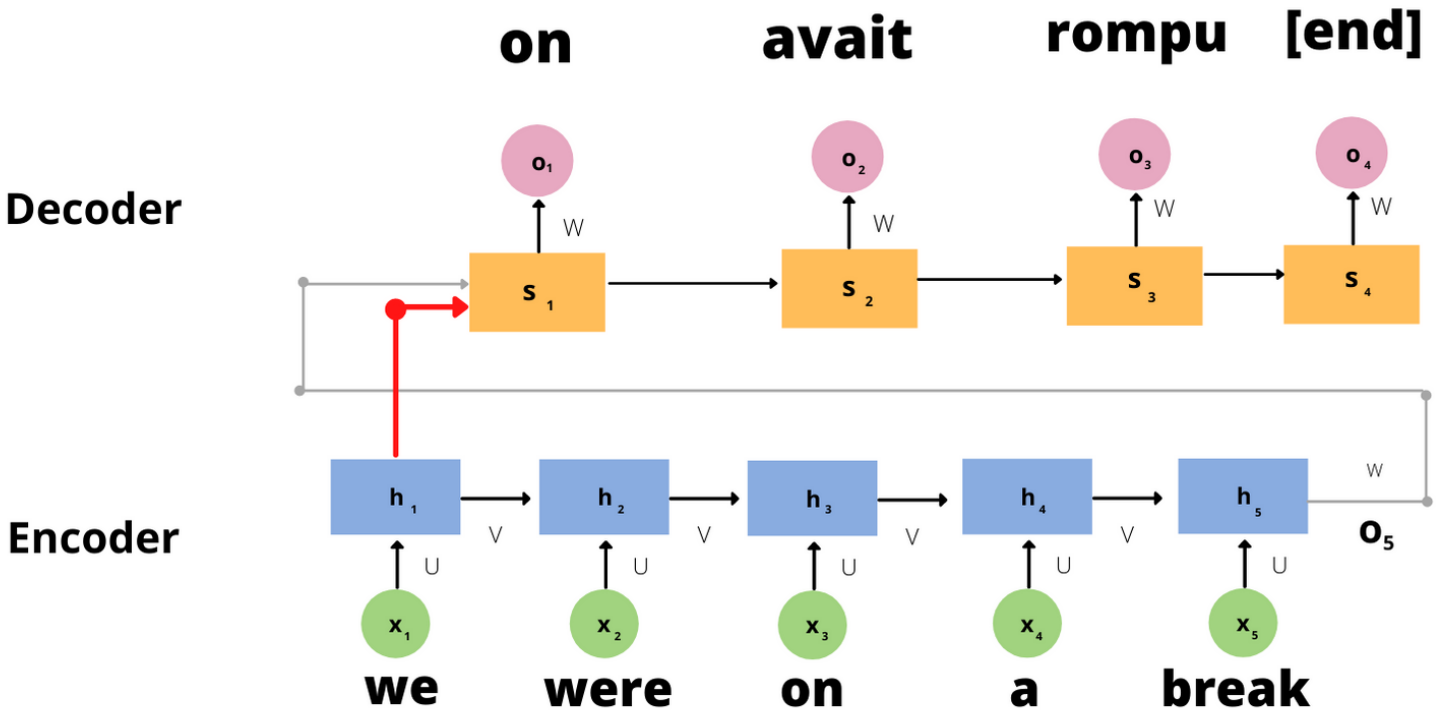
RNNs: a Machine Translation example

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.



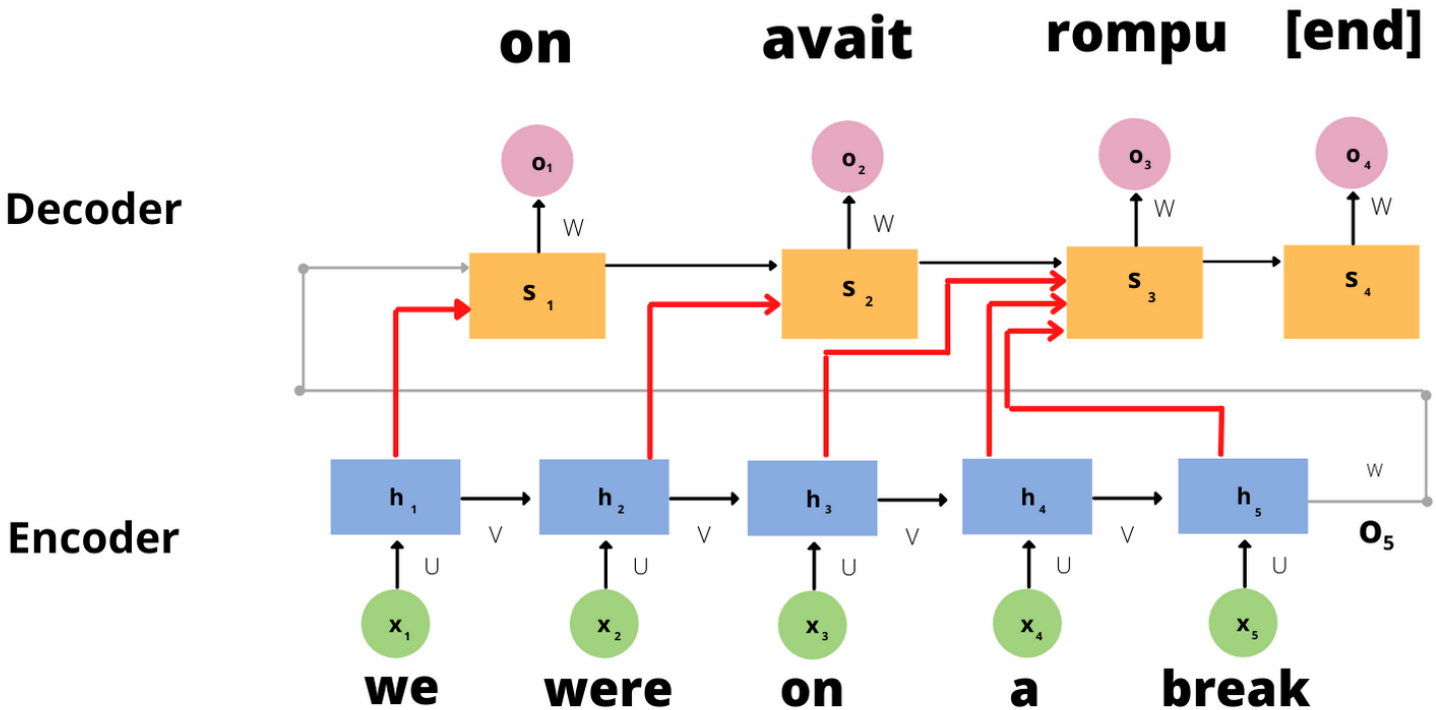
Attention

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:



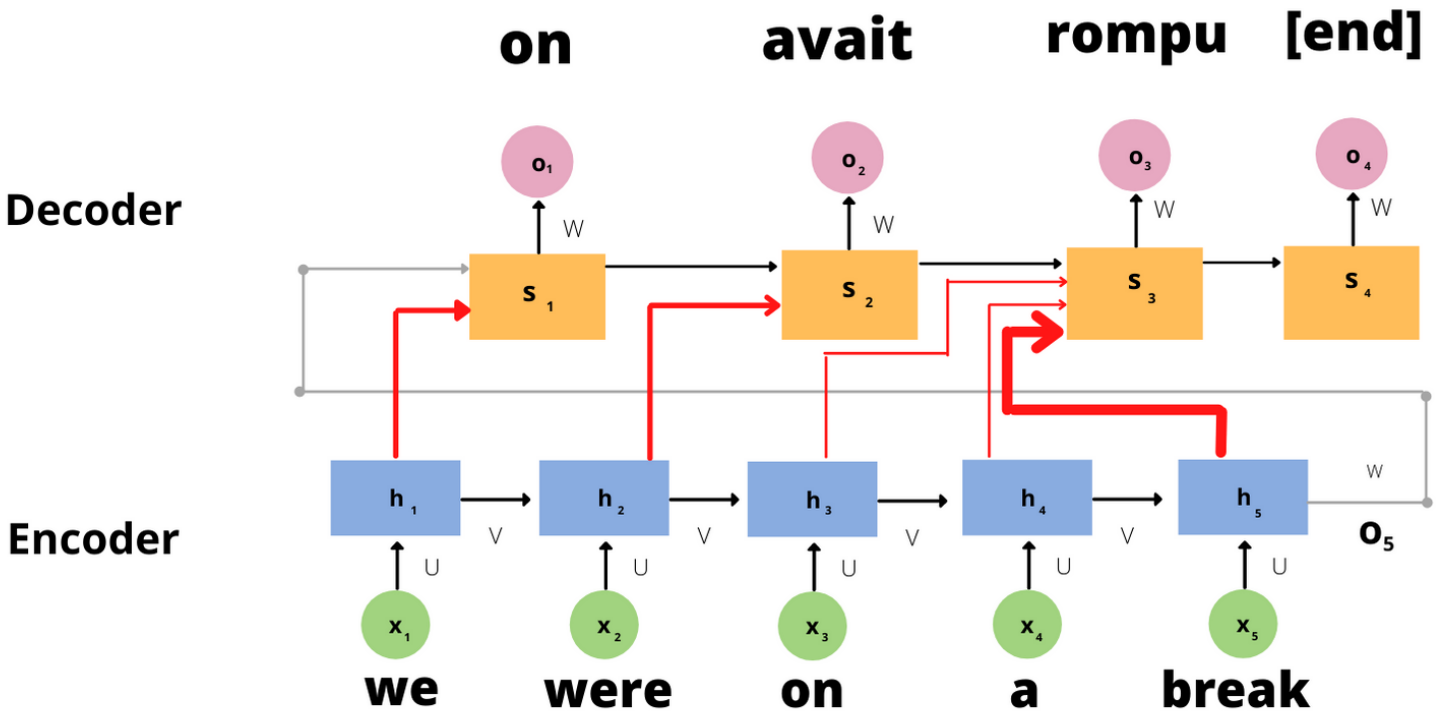
Attention

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:



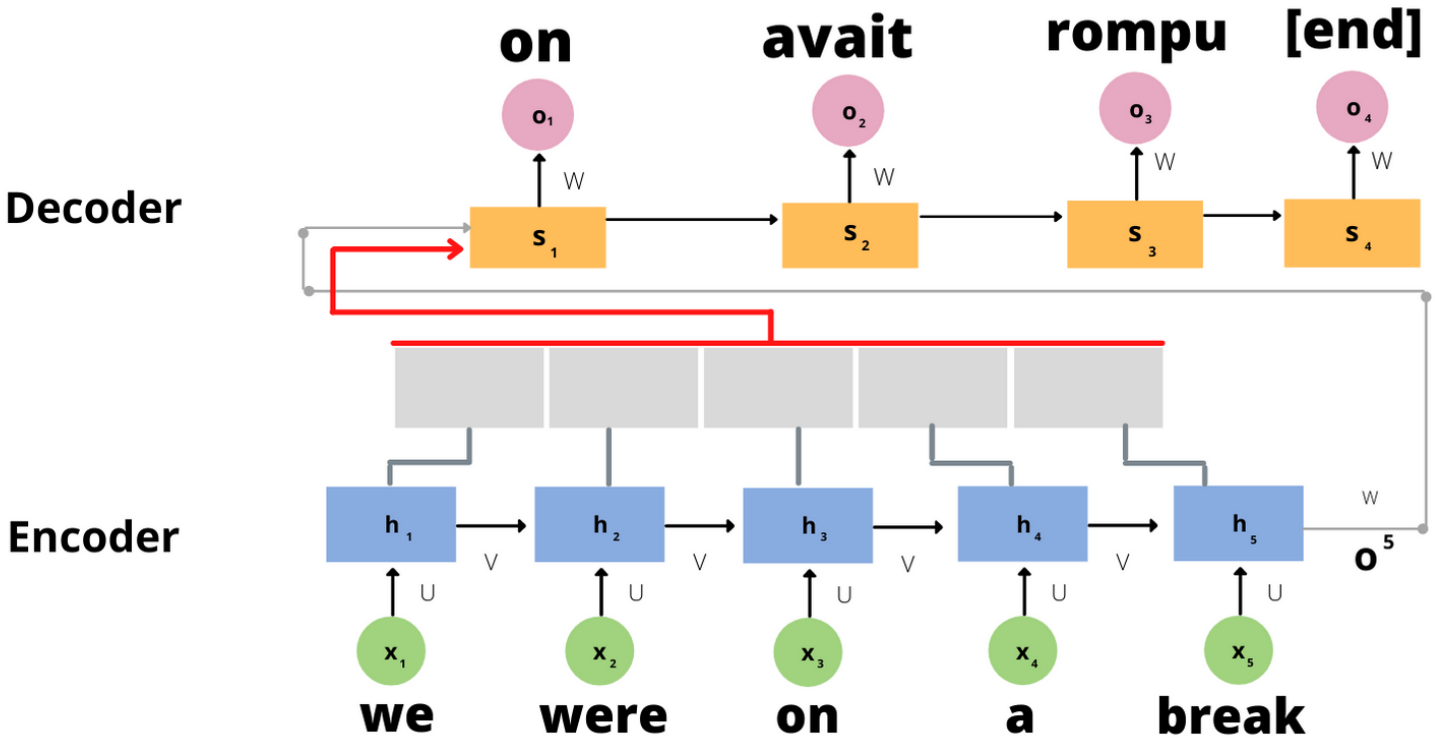
Attention

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**.



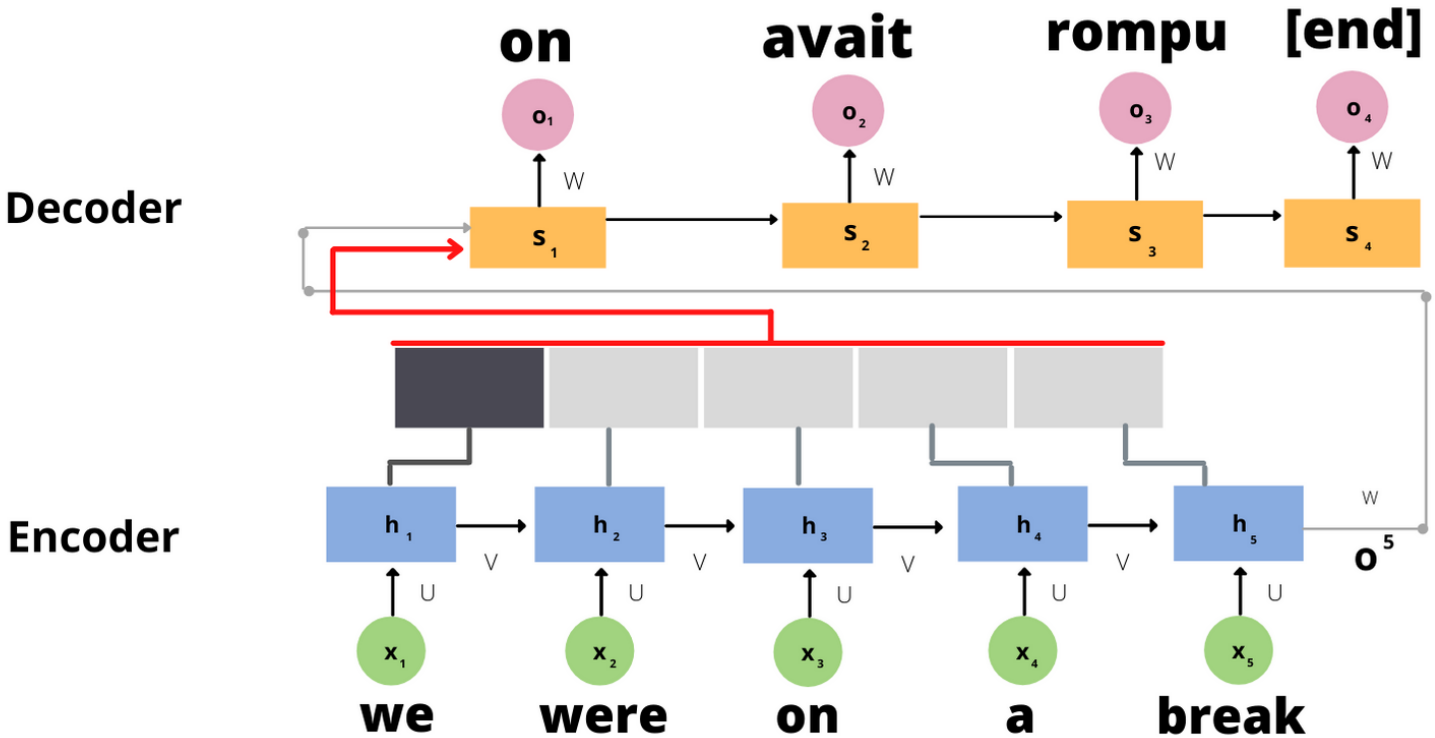
Attention Modules

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.



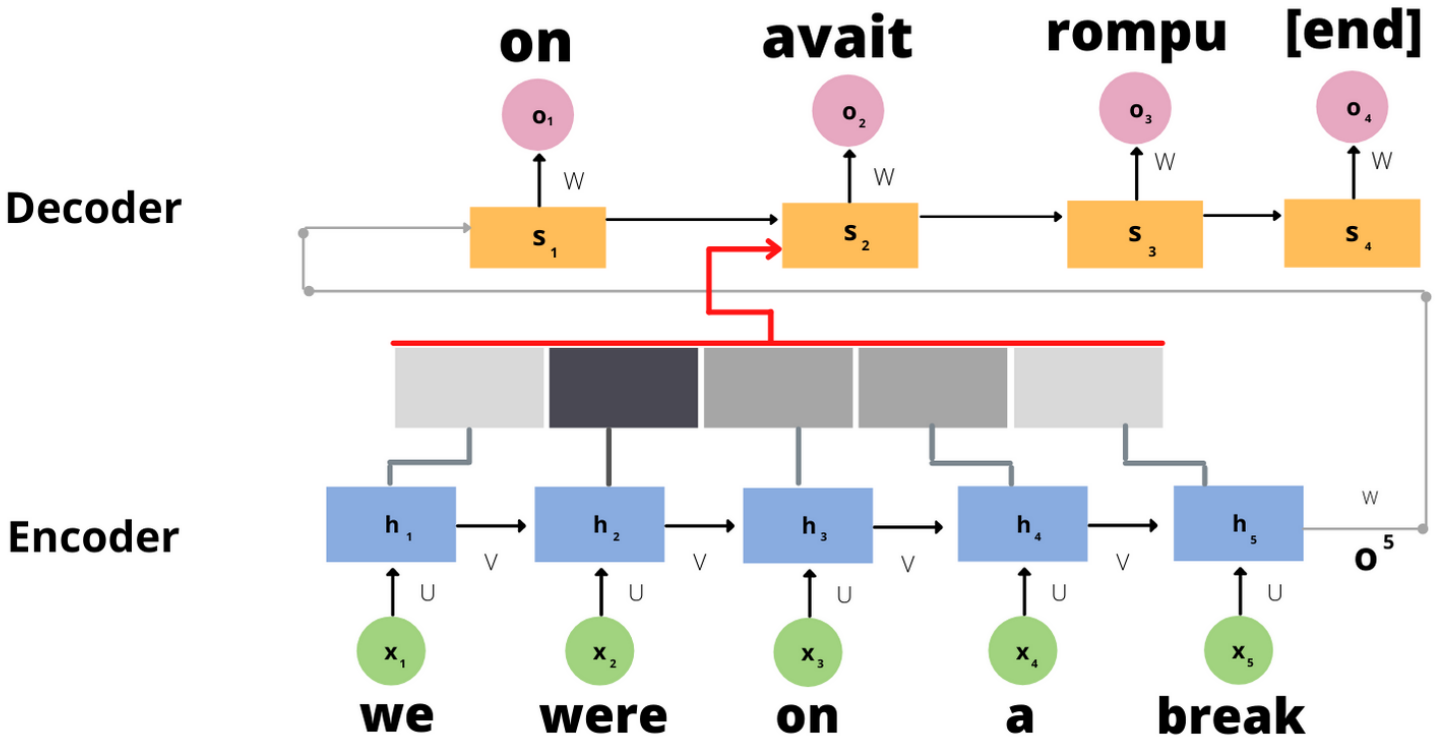
Attention Modules

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.



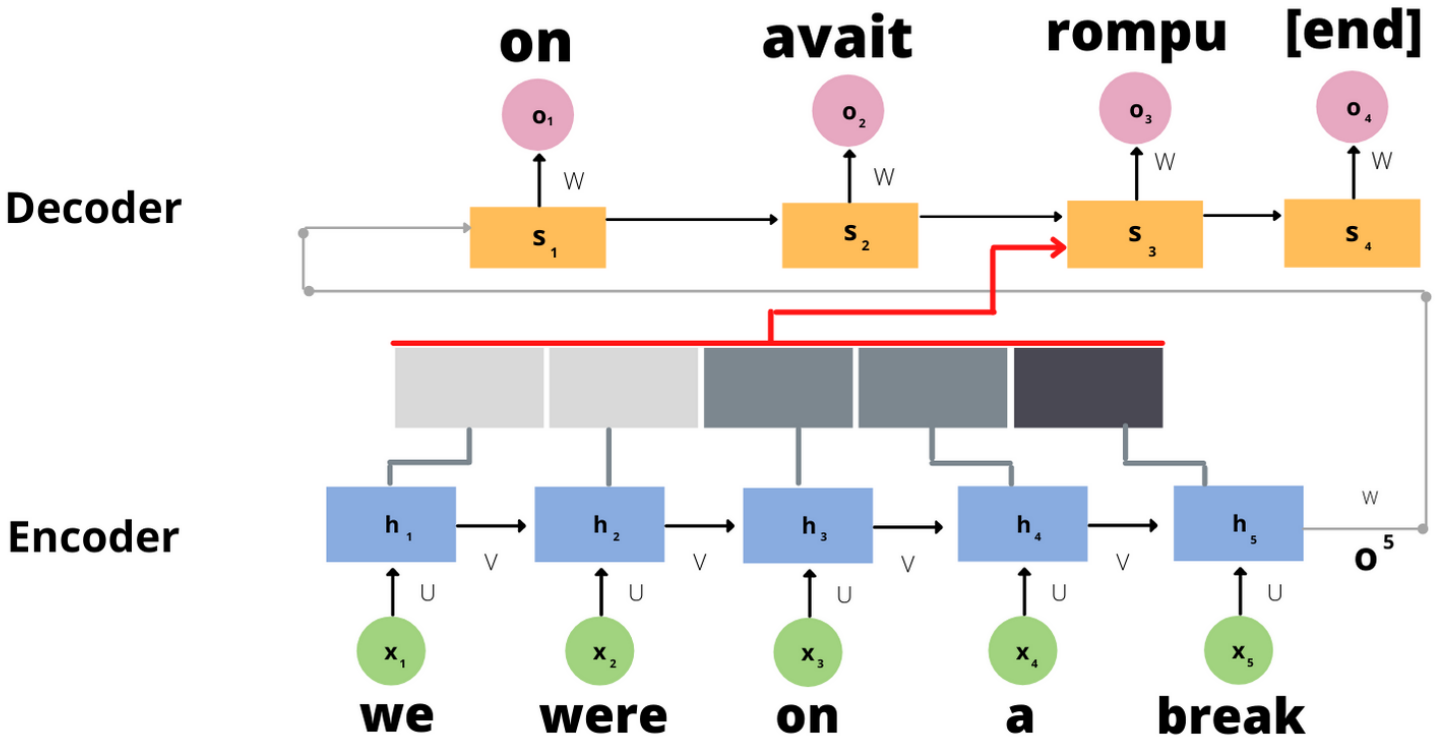
Attention Modules

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.



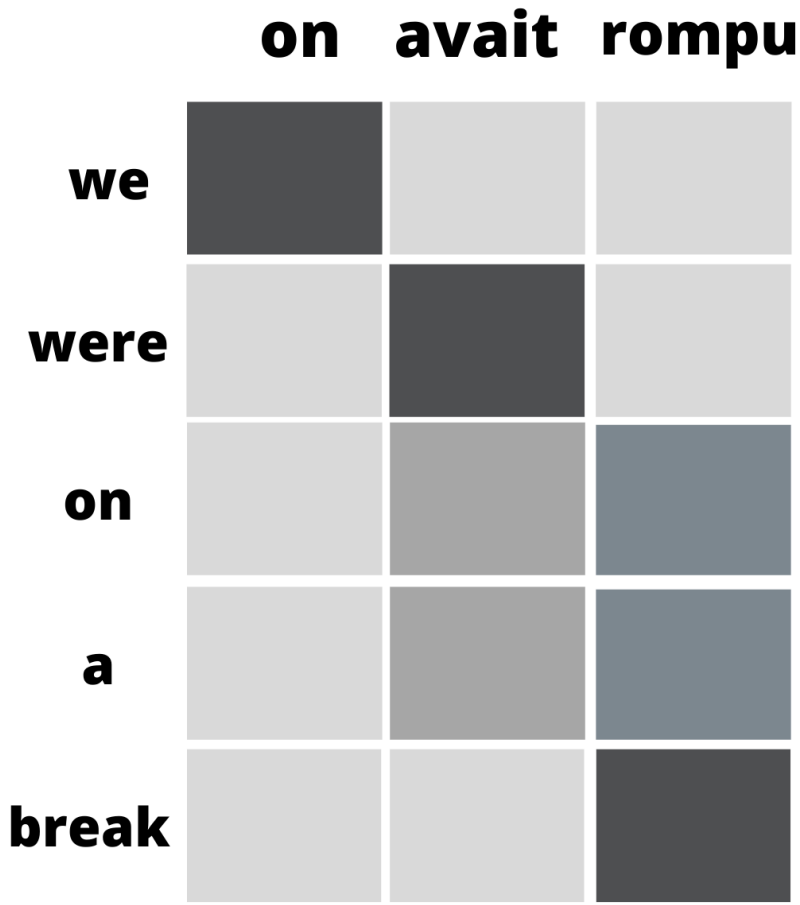
Attention Modules

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.



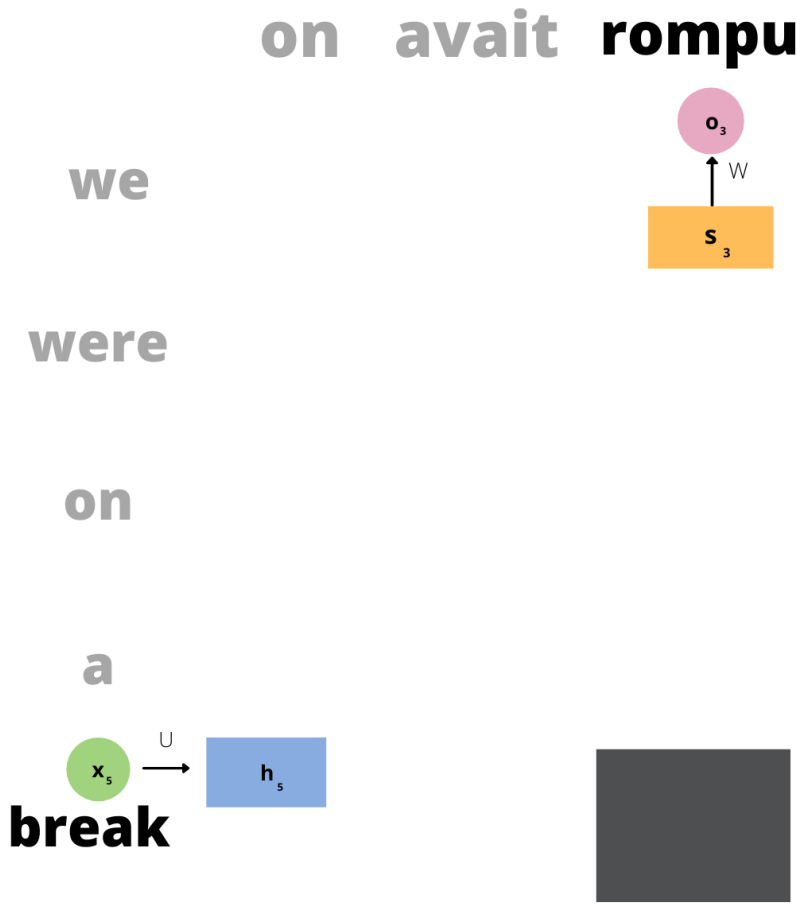
Attention Modules

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.



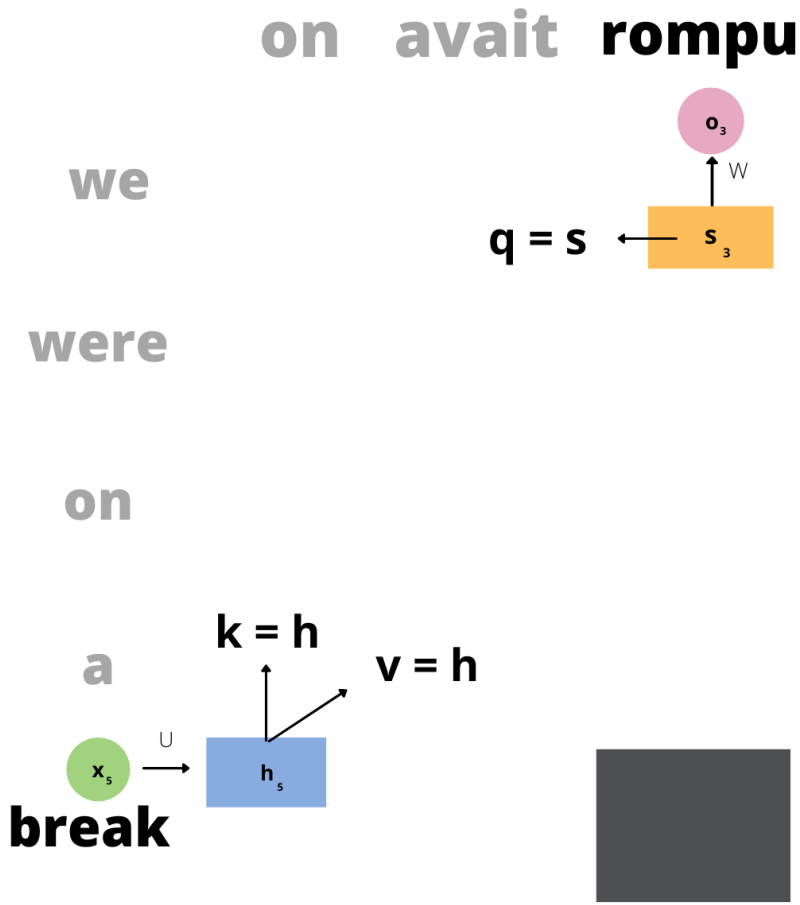
Attention Modules

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.



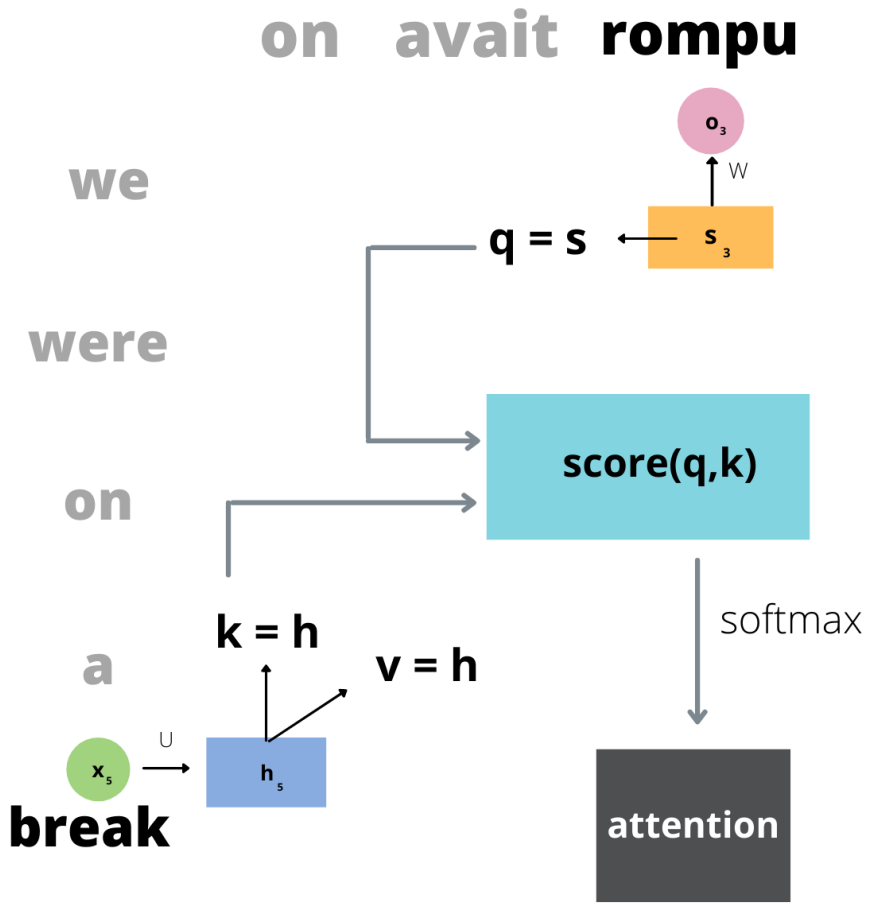
Attention Modules

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.



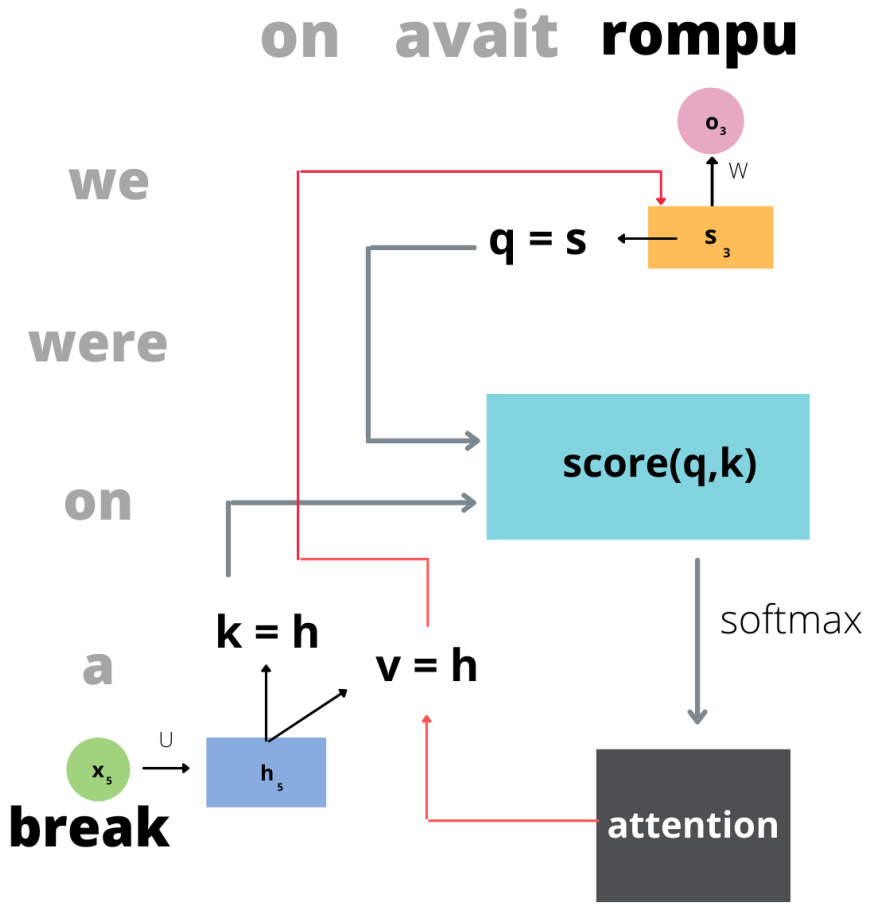
Attention Modules

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.



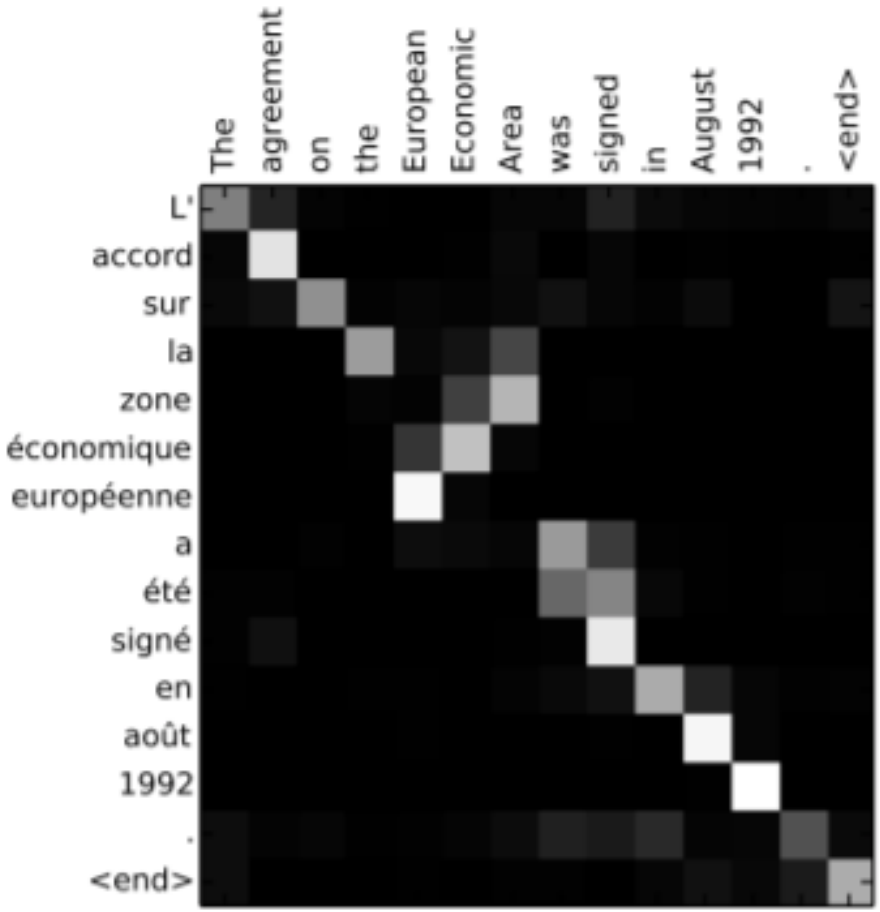
Attention Modules

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.



Attention in Transformers

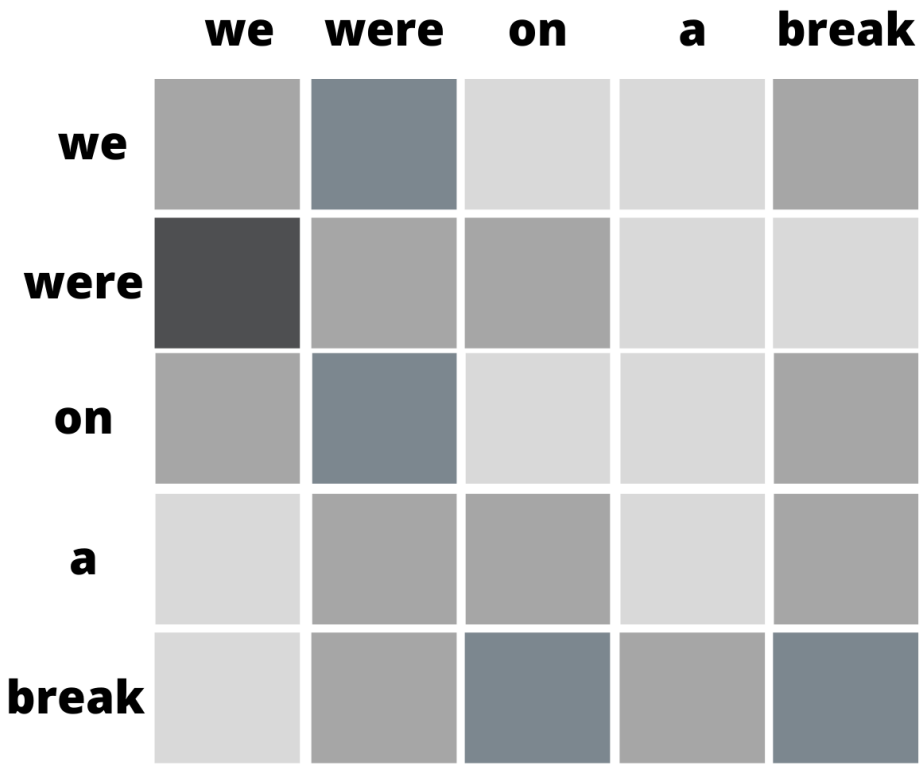
Self-attention: an attention network applied to the same sequence.

Multi-head attention, that is, h attention layers running in parallel.

There is no longer a recurrent network involved.

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

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



Attention in Transformers

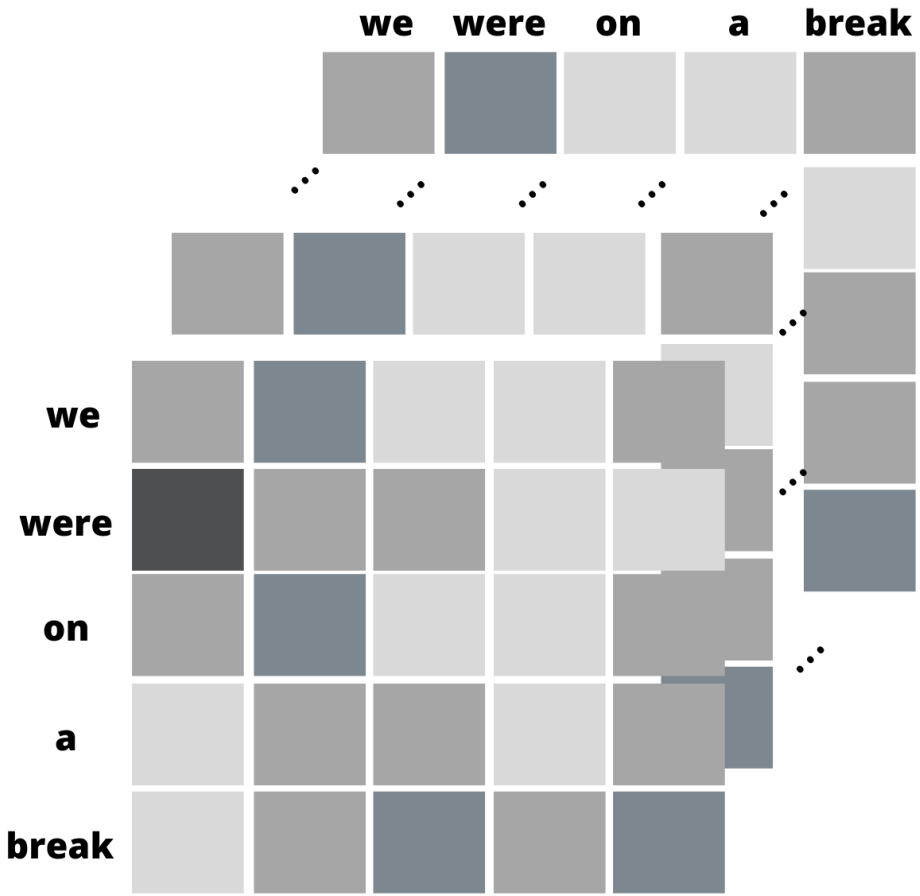
Self-attention: an attention network applied to the same sequence.

Multi-head attention, that is, h attention layers running in parallel.

There is no longer a recurrent network involved.

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

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



Attention in Transformers

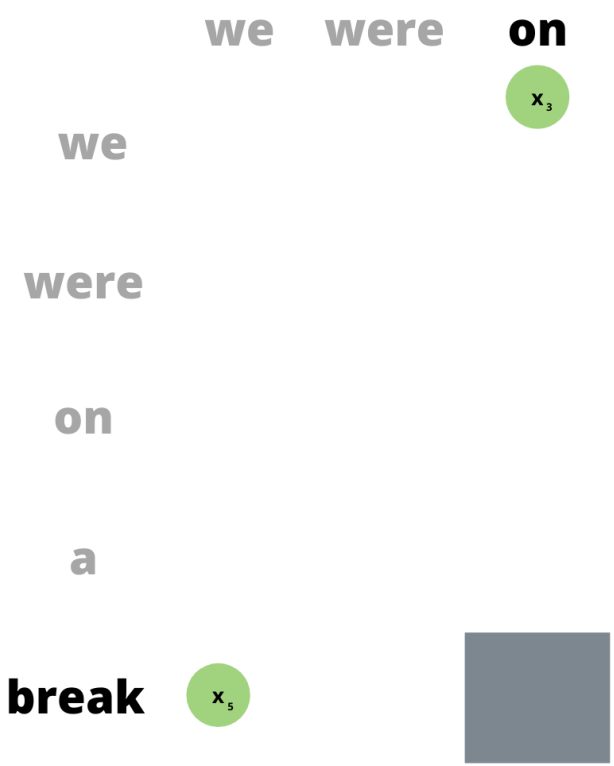
Self-attention: an attention network applied to the same sequence.

Multi-head attention, that is, h attention layers running in parallel.

There is no longer a recurrent network involved.

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

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



Attention in Transformers

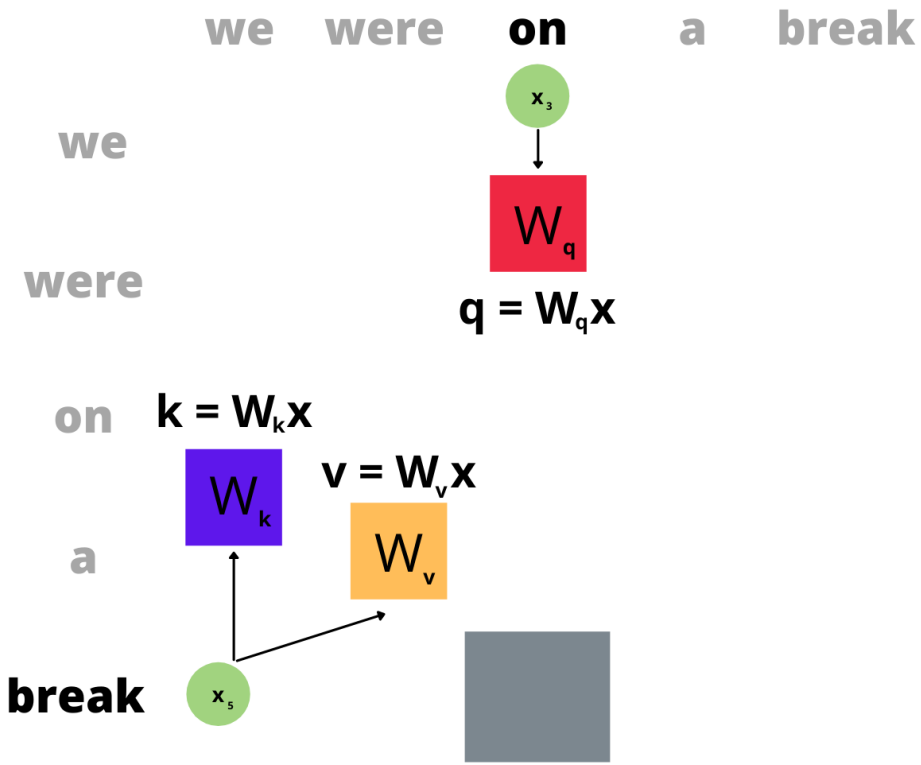
Self-attention: an attention network applied to the same sequence.

Multi-head attention, that is, h attention layers running in parallel.

There is no longer a recurrent network involved.

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

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



Attention in Transformers

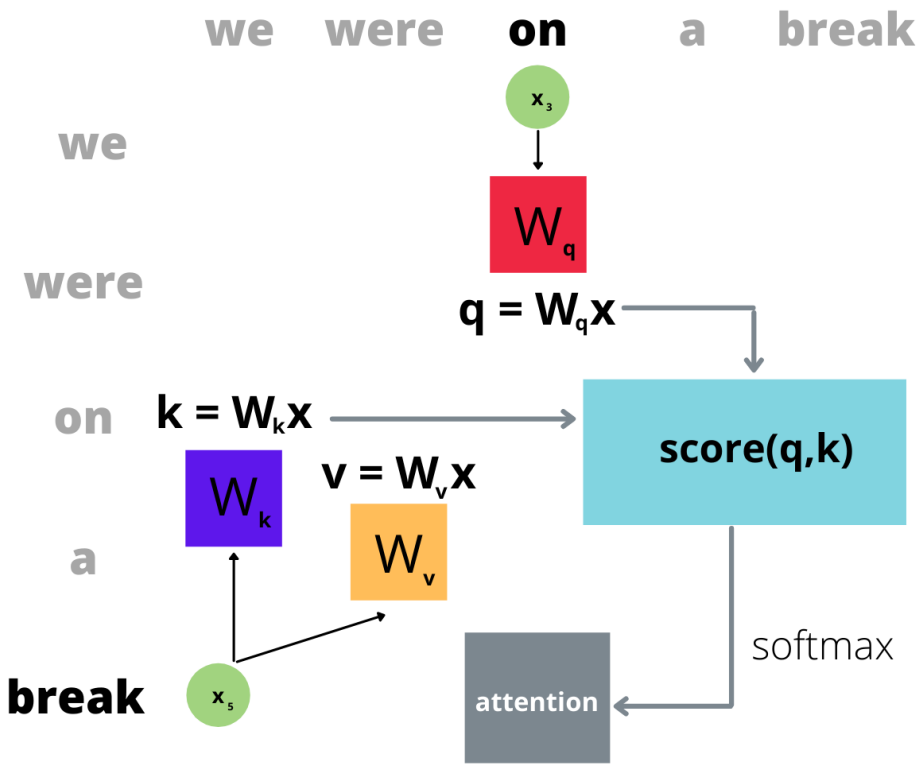
Self-attention: an attention network applied to the same sequence.

Multi-head attention, that is, h attention layers running in parallel.

There is no longer a recurrent network involved.

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

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



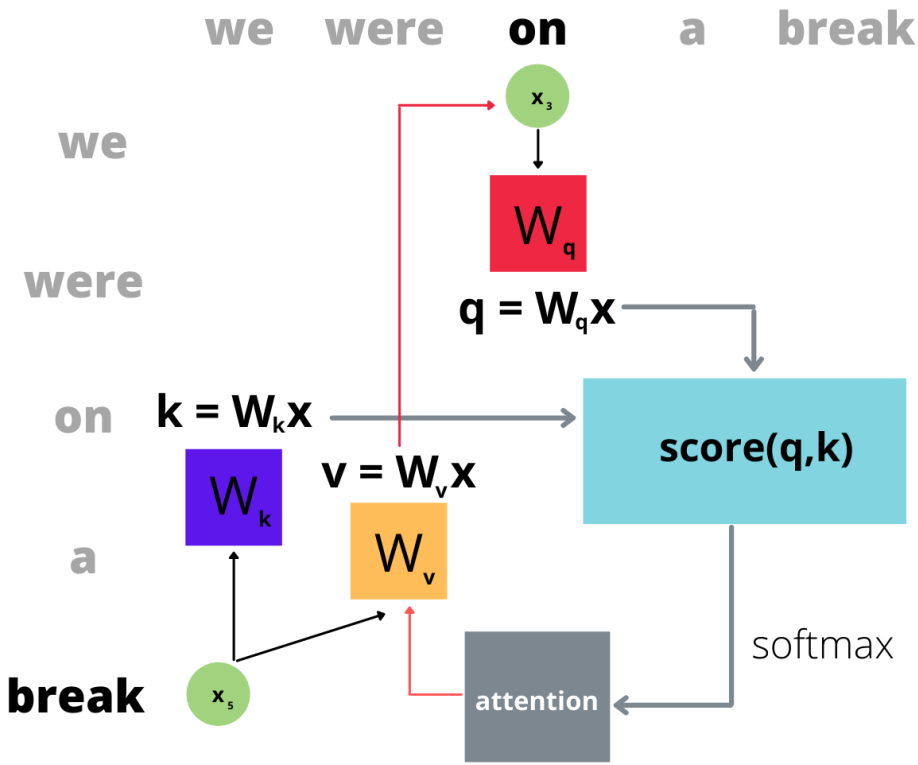
Attention in Transformers

Self-attention: an attention network applied to the same sequence.

Multi-head attention, that is, h attention layers running in parallel.

There is no longer a recurrent network involved.
Afterwards, the contextualised token for the j-th token in the sequence becomes

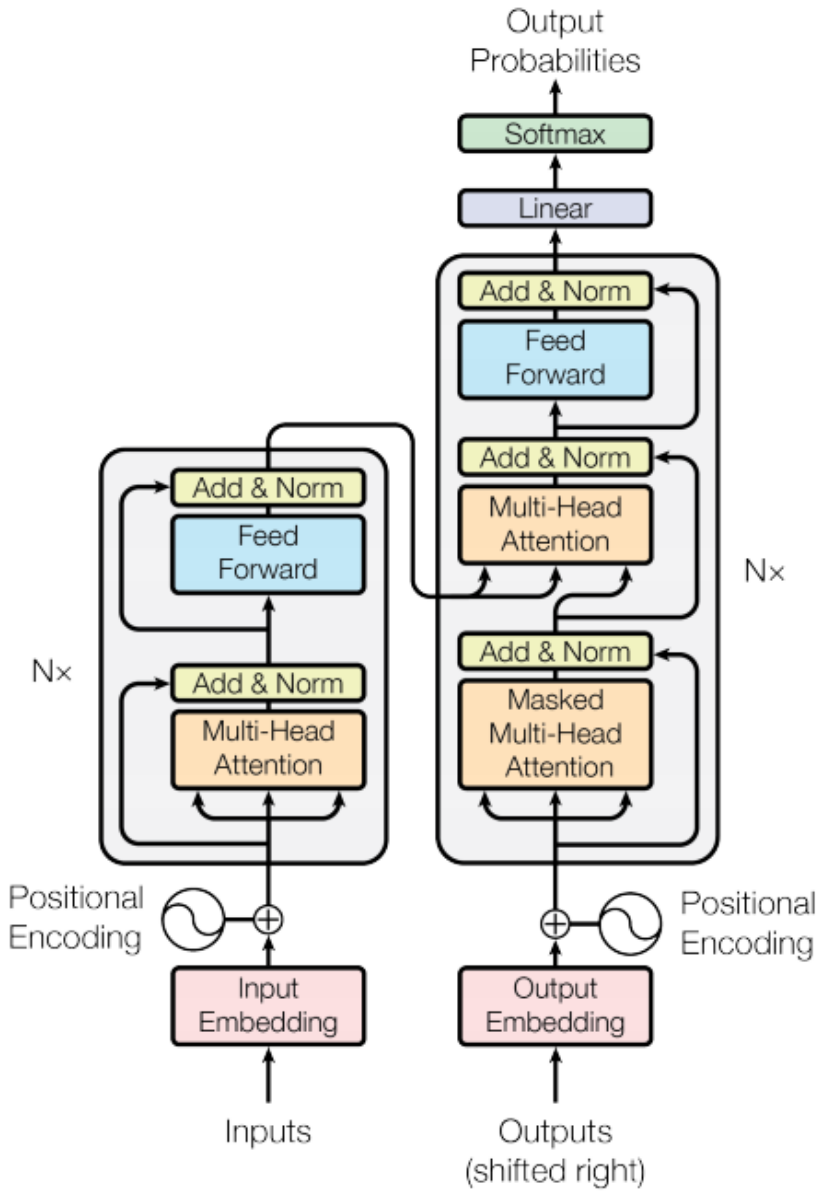
$$z_j = \sum_{i=1}^{C} score(q, K_j) v_j = S^T V$$



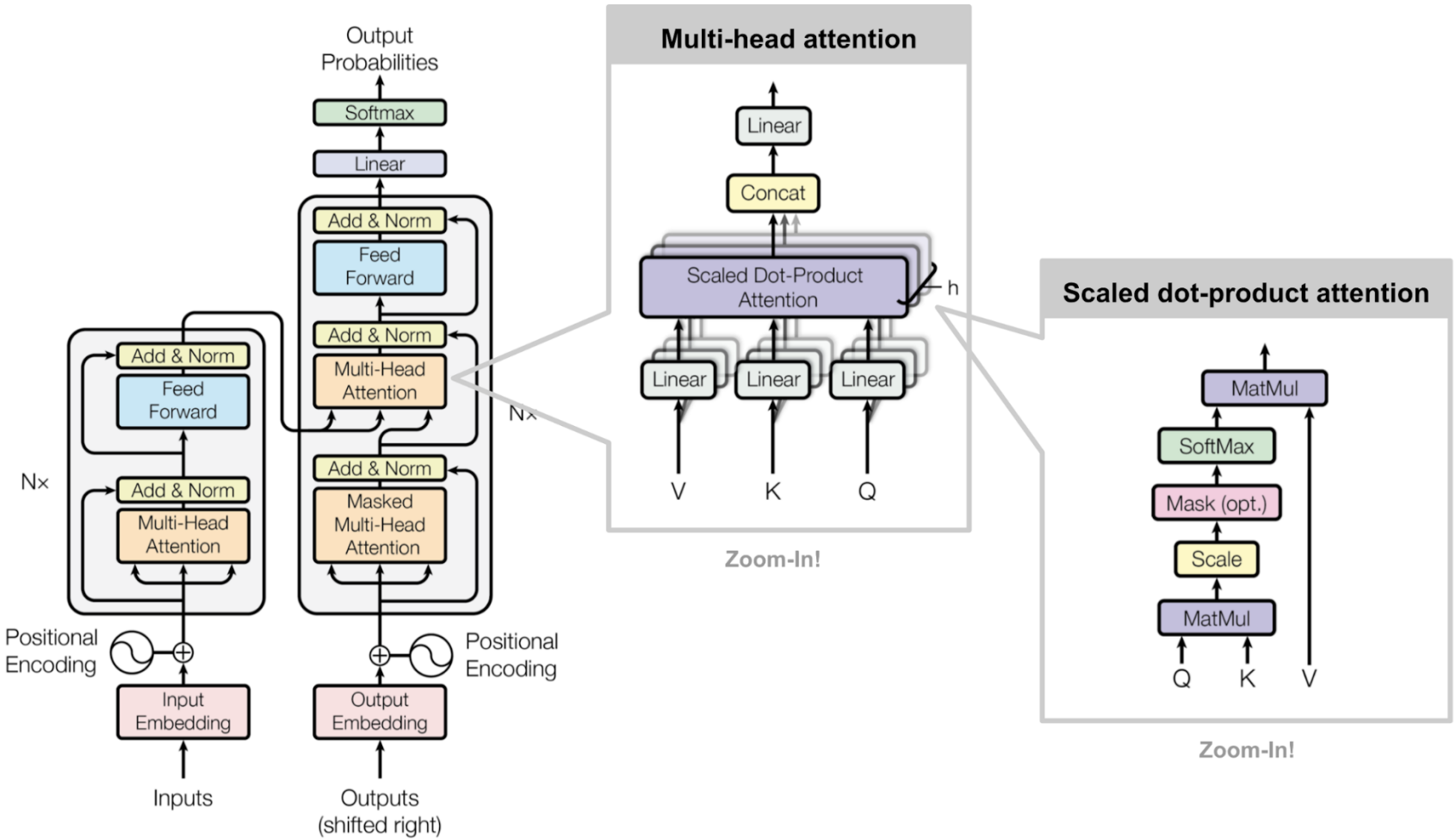
The Transformer

Transformers are a novel architecture that **gets rid of any recurrency**, thus relying solely on **attention modules**. 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.



The Transformer



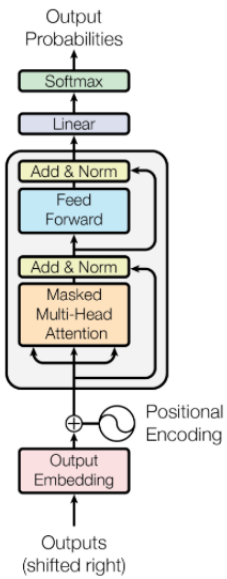
3

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

Popular Transformer-based LMs

Decoder-only GPT

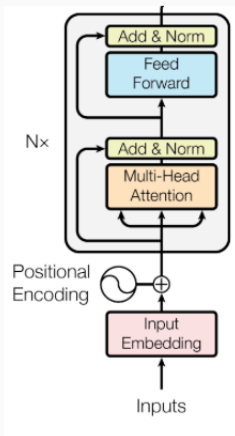
[sat_]



[START] [The_] [cat_]

Encoder-only BERT

[*] [*] **[sat_]** [*] **[the_]** [*]



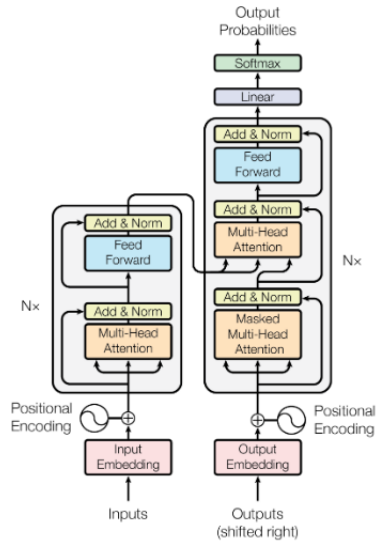
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

Enc-Dec T5

Das ist gut.

A storm in Attala caused 6 victims.

This is not toxic.



Translate EN-DE: This is good.

Summarize: state authorities dispatched…

Is this toxic: You look beautiful today!

Fig.. Source: Transformers tutorial, by Lucas Beyer (Google Brain)

Ethical Concerns with LMs

Huge language models can be extremely dangerous :
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 them due to their size. This type of large models have oppened whole new areas of research, like BERTology , where the goal is to understand LMs inner behaviour.

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.

Table of contents

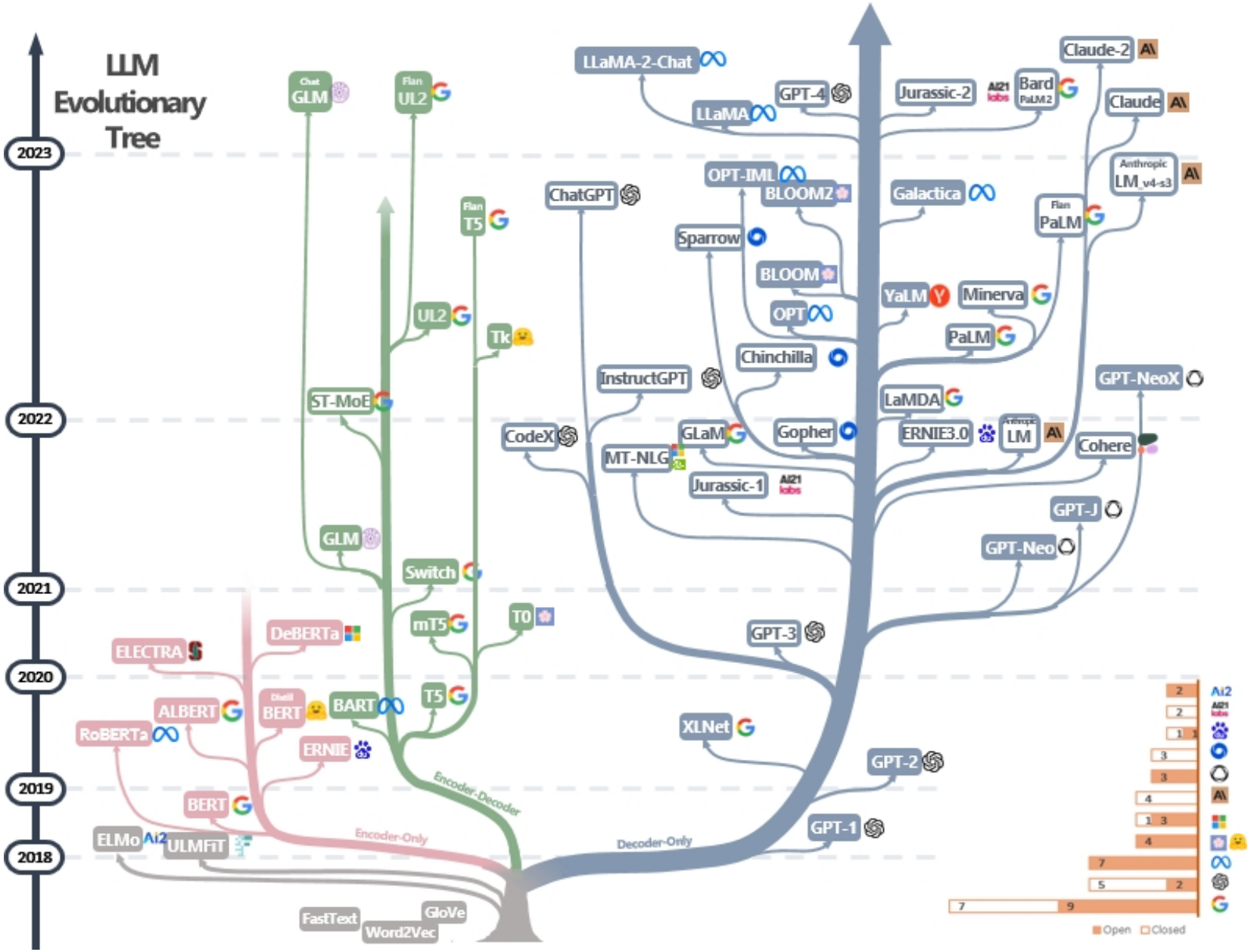


Fig.. The sentiment neuron adjusting its value on a character-by-character basis. Source: `https://openai.com/research/unsupervised-sentiment-neuron`

GPT-0: The sentiment neuron (2017)

Training on a significant text corpus on the task to predict the next token (LM), can we learn valuable representations for another task?
Yes! A single "sentiment neuron" may control the sentiment of the generated text.
Context: an era previous to the one in which Transformers took over as a de-facto architecture in sequence models.

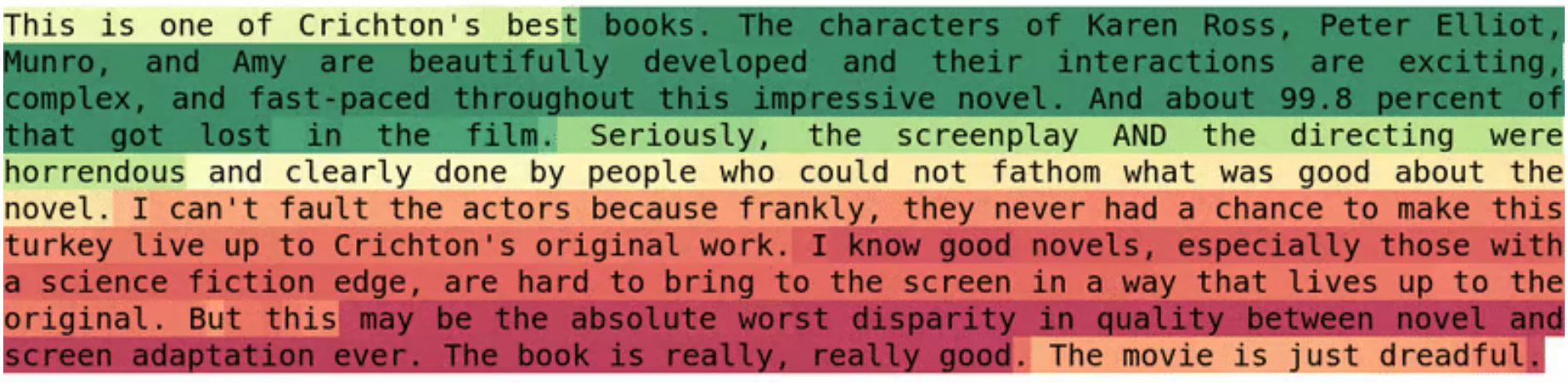


Fig.. The sentiment neuron adjusting its value on a character-by-character basis. Source: Learning to generate reviews and discovering sentiment

GPT-1: Decoder-only transformer (2018)

OpenAI exploits and doubles the efforts on the idea that a base model can learn powerful, general representations. Transformer became a game changer on scale training! A multi-task (explicit) adaptation recipe was formed via formatting the input as a token sequence and reusing the pre-trained model.

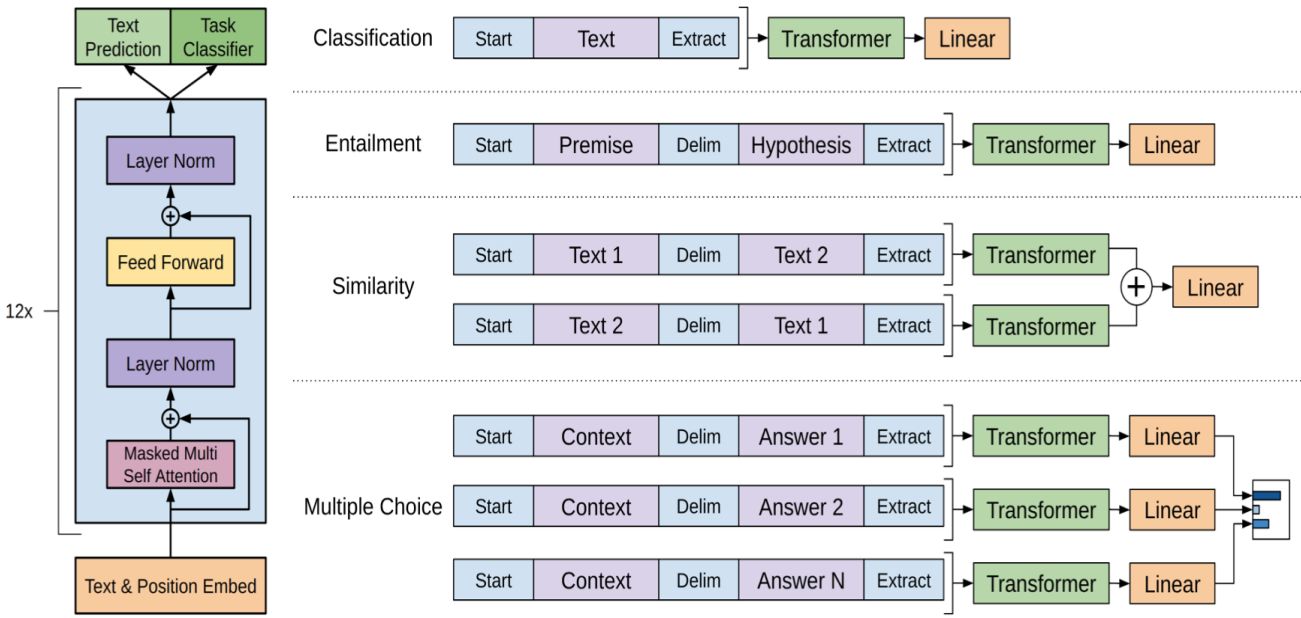


Fig.. (left) Transformer decoder-only base model trained on the simple task of predicting the next token. (right) Input transformations into a token sequence to be processed by the base model, followed by a downstream layer (linear + softmax). Source: Improving Language Understanding by Generative Pre-Training (GPT1 paper)

GPT-2: Prompt era unlocked (2018)

"I'm not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile [I'm not a fool].**

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: "**Mentez mentez, il en restera toujours quelque chose**," which translates as, "**Lie lie and something will always remain.**"

"I hate the word '**perfume**,"' Burr says. 'It's somewhat better in French: '**parfum**.'

If listened carefully at 29:55, a conversation can be heard between two guys in French: "**-Comment on fait pour aller de l'autre coté? -Quel autre coté?**", which means "**- How do you get to the other side? - What side?**".

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?

"**Brevet Sans Garantie Du Gouvernement**", translated to English: "**Patented without government warranty**".

Fig.. Make your model look like a document to auto-complete translation without explicit supervision, with no downstream task layer at all. Source: Language Models are Unsupervised Multitask Learners (GPT2 paper).

GPT-2: Fake it till you make it (2018)

GPT-2 kicked off the era of prompting over fine-tuning. No explicit supervision. Make your model look like a document, so the inherent task is performed by taking the model and asking to complete the document (prompt era unlocked). There is no further training or gradient updates [4] to its parameters.

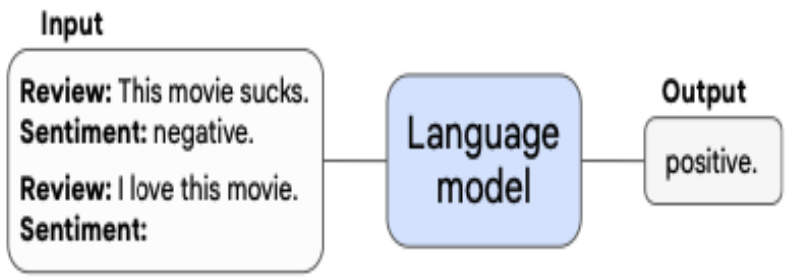


Fig.. Emergent Abilities of Large Language Models (Source)

[4] Some articles explore the idea of an explicit gradient-like update of the parameters when doing prompting.

GPT-2: Scalability payoffs (2018)

Scalability proves useful for powerful and general representations when performing zero-shot transfer. In a nutshell, **How do we exploit general capabilities without intervening with the model head—the re-purpose layer for downstream tasks—in the context of predicting the next token?** Scaling + prompting

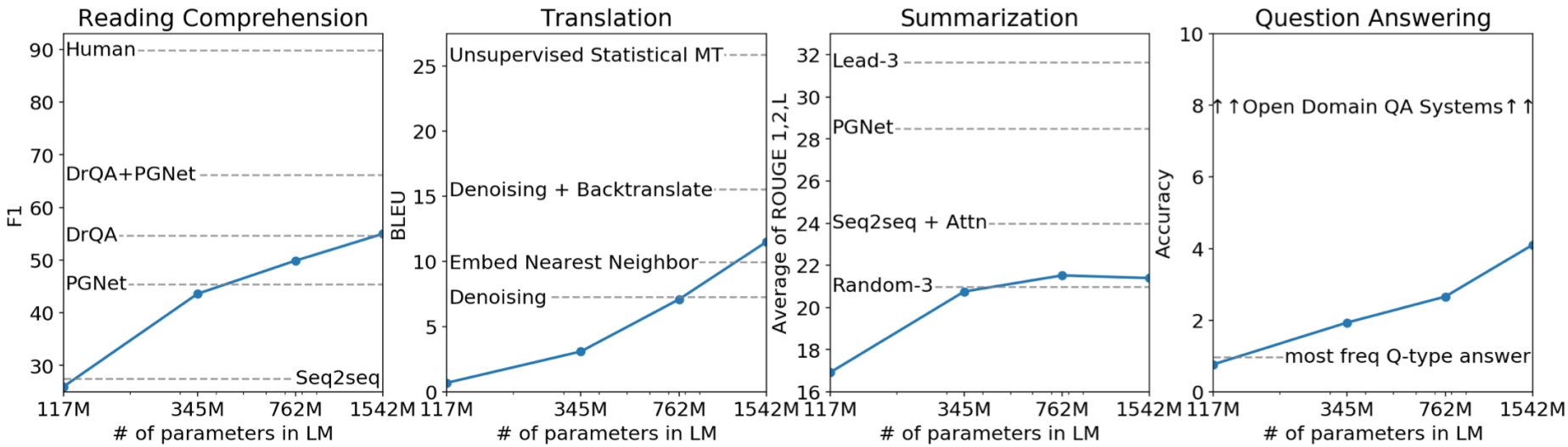


Fig.. Zero-shot performance of a base model trained on WebText as a function of model size (117M-345M-762M-1542M parameters) in 4 different NLP tasks. Each task is evaluated on a benchmark dataset and compared with SOTA-specific models at that time, 2018-2019. Source: Language Models are Unsupervised Multitask Learners (GPT2 paper).

ChatGPT: more than a LM (2022)

Fig.. Slide from the talk "The State of GPT", by Andrej Karpathy (2023).

ChatGPT: Supervised Finetuning

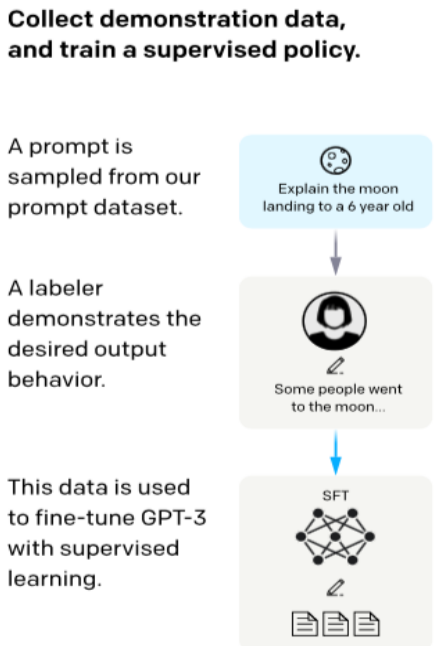


Fig.. source: Training language models to follow instructions with human feedback

Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful**, **truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not giving overly long or rambling answers, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.

By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

ChatGPT: Reward Modeling

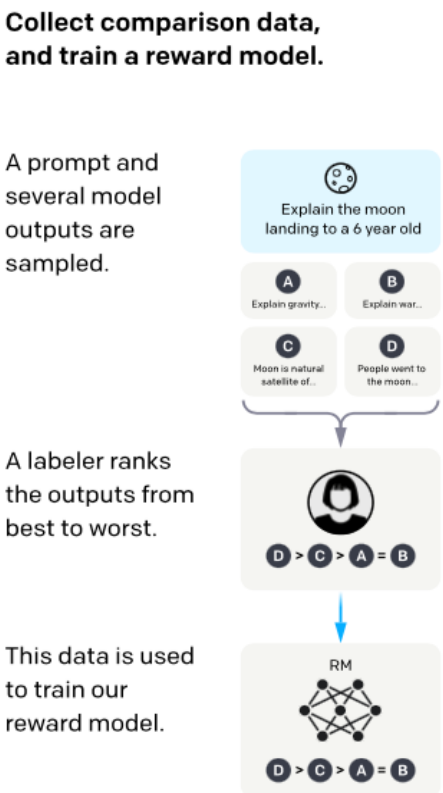


Fig.. Training language models to follow instructions with human feedback

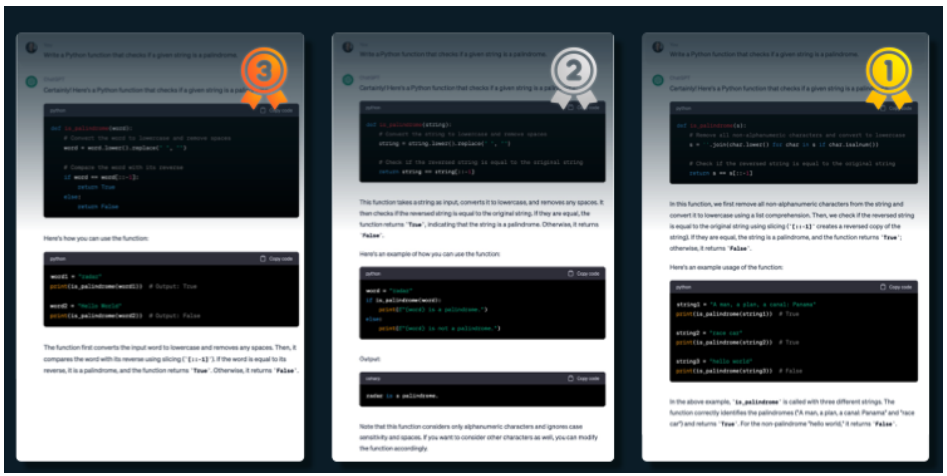


Fig.. An example of a reward modelling dataset. Source: slide from the talk "the state of gpt", by Andrej Karpathy (2023).

ChatGPT: RL from Human Feedback

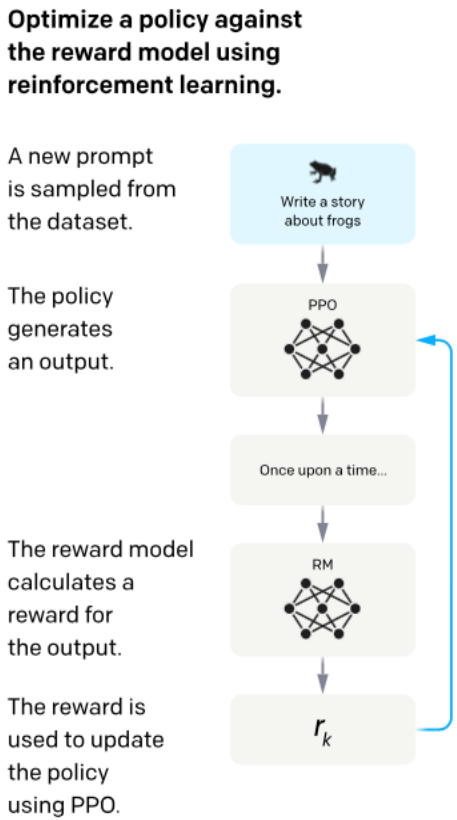
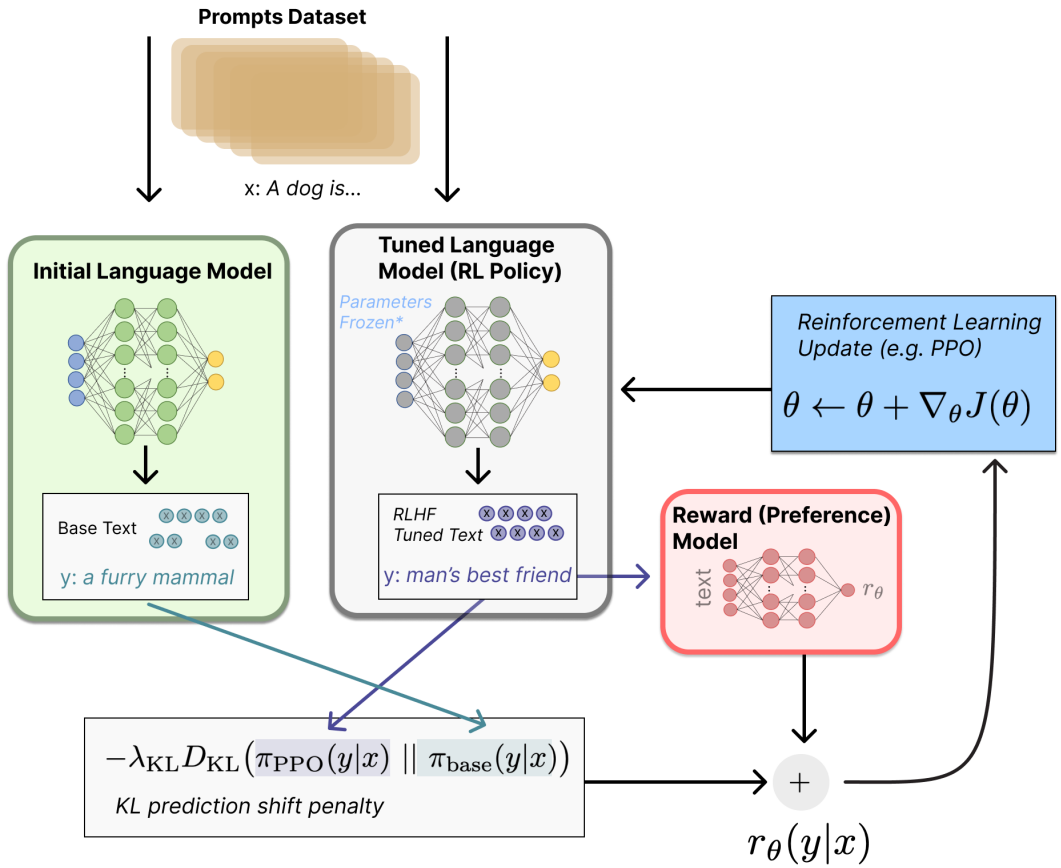


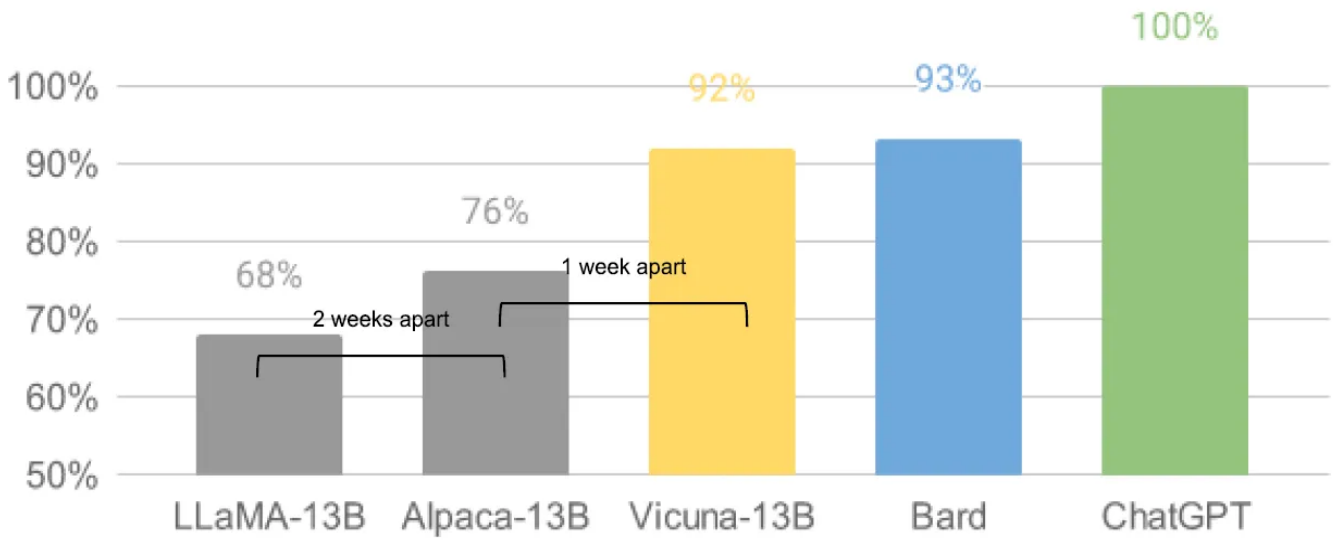
Fig.. Training language models to follow instructions with human feedback

Fig.. Reinforcement Learning from Human Feedback (Loop). Source: Illustrating Reinforcement Learning from Human Feedback (RLHF)



Open Source models are faster

Meta LLaMA-13B Leak to the public, March 03/2023 → Fine-tuning with ChatGPT conversation (prompt-response) → Alpaca-13B (2 weeks apart, budget $USD500) → Fine-tuning with 70k user-shared ChatGPT (ShareGPT.com) conversations → Vicuna-13B (1 week apart, budget $USD150).



*GPT-4 grades LLM outputs. Source: https://vicuna.lmsys.org/

Constraints + Community = Innovations.

Finetuning an LLM is prohibitive for the majority of individuals.

Finetuning a model requires saving an entire copy.

Finetuning could downgrade model performance on other tasks.

LoRA uses low-rank matrices to update and freeze the original parameters. Modular. Easy to share.

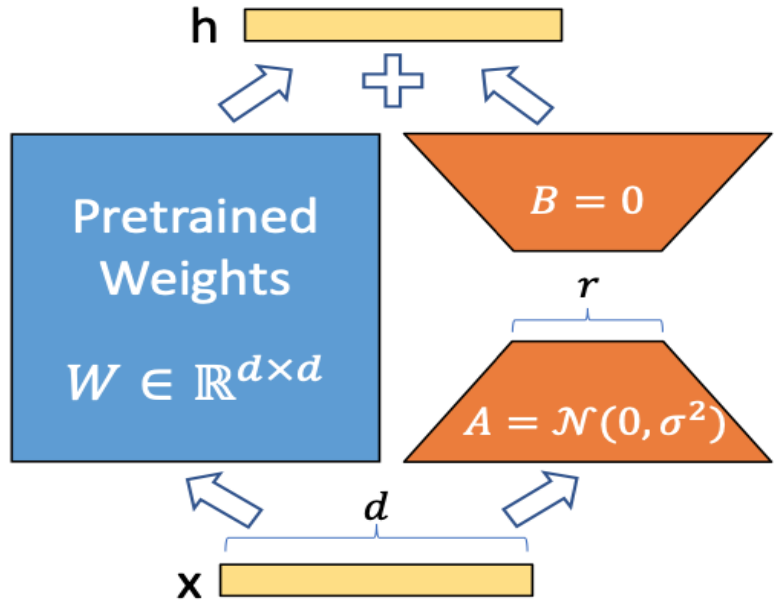


Fig.. Freeze the pre-trained model weights and inject trainable rank decomposition matrices (A and B) into each layer of the Transformer architecture. Source: LoRA: Low-Rank Adaptation of Large Language Models

PixelRNN (Van den Oord et al. 2016)

Typically, we saw NLP examples linked to autoregressive models, but it is also possible to generate images.

PixelRNN is a generative model used for image generation, mainly focused on generating images one pixel at a time.

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and above (LSTM recurrence).

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

Problem: Very slow during training and testing; $N \times N$ image requires $2N - 1$ sequential steps.

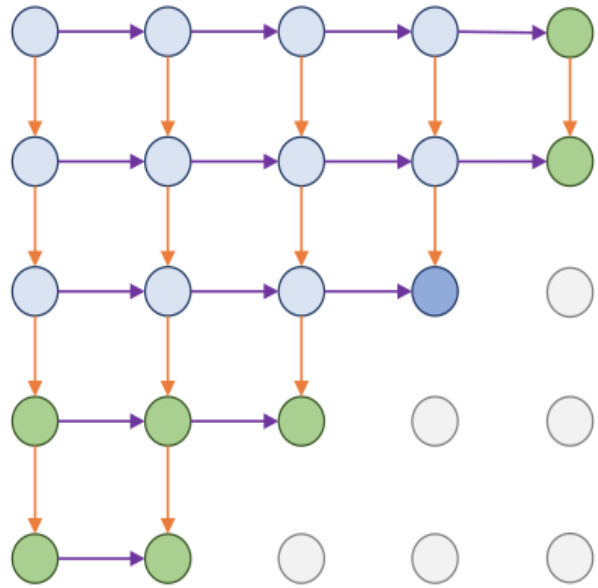


Fig.. The blue pixel only depends on the light blue pixels (left and above). Predict red, blue, and green (0-255) using the hidden states for the previous and current pixels. Source: Pixel Recurrent Neural Networks

PixelCNN (Van den Oord et al. 2016)

PixelCNN extends the idea of capturing pixel dependencies by using masked convolutions, which restrict the network's access to information about future pixels, ensuring a causal generation process.

Training is faster than PixelRNN; we can parallelise convolutions since context region values are known in advance.

Generation is still slow; we must proceed sequentially, generating pixel by pixel.

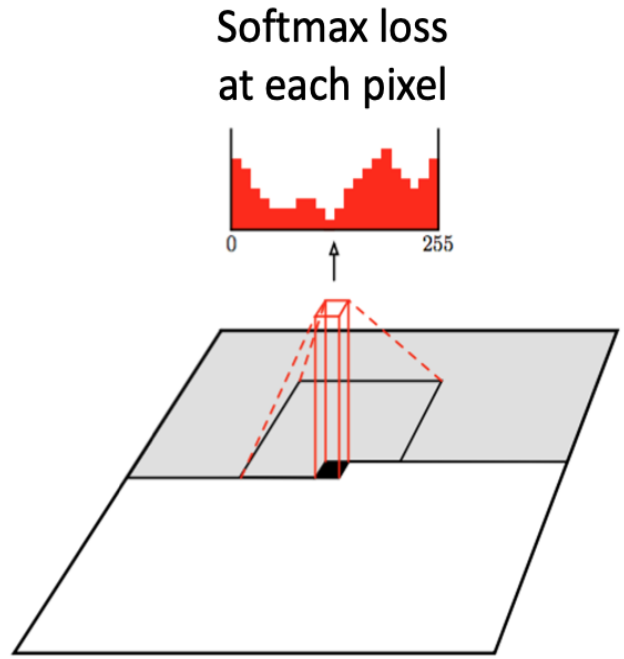


Fig.. The black square depicts a predicted pixel that uses convolution to capture previous pixel dependencies but ensures that it does not see subsequent pixels. The task is to predict the following pixel so the masked, or causal convolution, trims that portion of the local context. Source: Pixel Recurrent Neural Networks

References I

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

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

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

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

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

References II

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

Radford, A., Jozefowicz, R., Sutskever, I. (2017). Learning to generate reviews and discovering sentiment. arXiv 2017. arXiv preprint arXiv:1704.01444. https://openai.com/research/unsupervised-sentiment-neuron

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. (2018). Improving language understanding by generative pre-training. https://openai.com/research/language-unsupervised

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

References III

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., Vladymyrov, M. (2023, July). Transformers learn in-context by gradient descent. In International Conference on Machine Learning (pp. 35151-35174). PMLR. https://proceedings.mlr.press/v202/von-oswald23a.html

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35, 27730-27744. https://openai.com/research/instruction-following#sample5

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... Chen, W. (2021). Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685. https://arxiv.org/abs/2106.09685

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., ... Fedus, W. (2022). Emergent abilities of large language models. arXiv preprint arXiv:2206.07682. https://arxiv.org/pdf/2206.07682.pdfA

References IV

Van Den Oord, A., Kalchbrenner, N., Kavukcuoglu, K. (2016, June). Pixel recurrent neural networks. In International conference on machine learning (pp. 1747-1756). PMLR.https://arxiv.org/pdf/1601.06759.pdf

Clase 7: Modelos Auto-regresivos

MDS7203 Modelos Generativos Profundos

Felipe Tobar
Cristóbal Alcázar - Camilo Carvajal Reyes

Iniciativa de Datos e Inteligencia Artificial
Universidad de Chile

22 de septiembre 2023