European Southern Observatory

Sequence Models with Transformers

A Journey Through Language Models, Current Insights, and Practical Applications at Paranal Observatory

Camilo Carvajal Reyes - Cristóbal Alcázar

Master of Data Science, U. de Chile

10th September 2023

Credit for cover image: ESO/M. Kornmesser

Outline

Points to discuss today

- Introduction to Natural Language Processing (NLP) and language modelling.
- The transformer architecture
- Overview of algorithms and current trends in transformer-based models.
- Applications to log-mining in Paranal operations.

Table of contents

European Southern Observatory

About Ourselves

Camilo Carvajal Reyes

dim.uchile.cl/~ccarvajal/ - ccarvajal@dim.uchile.cl

- MSc candidate in Data Science @ U. de Chile
  Current research: safeness for score-based generative models
  Supervised by Felipe Tobar and Joaquín Fontbona
  MDS7203 Deep Generative Models: Teaching assistant
- Mathematical Engineering @ U. de Chile
- Engineering degree @ CentraleSupélec, France

  Research track: word embeddings and semantics of LLMs
- Ethics: NLP for ethical competences evaluation and Cofunder of AEDIA: Association for Ethics in Data and Artificial Intelligence.
- @ ESO Paranal Software group (summer 2022):
  Natural Language Processing for Problem Troubleshooting

European Southern Observatory

About Ourselves

Cristóbal Alcázar

alkzar.cl - cristobal.alcazar@ug.uchile.cl

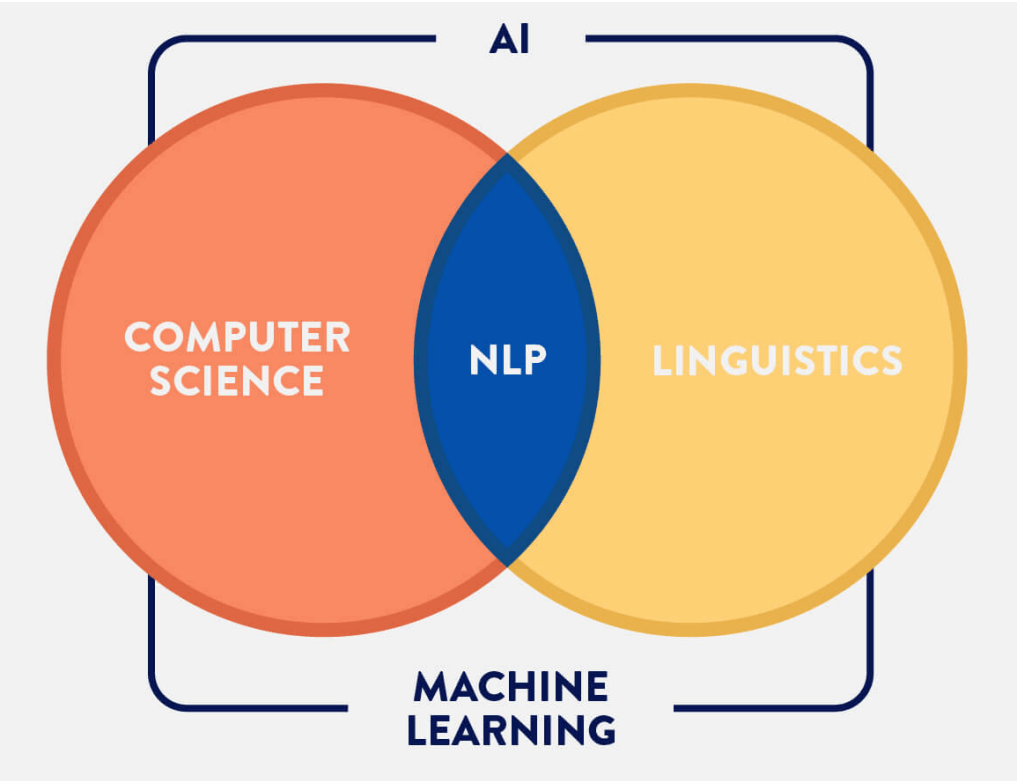
- MSc candidate in Data Science @ U. de Chile
  Current research: diffusion generative models, RLHF.
  Supervised by Felipe Tobar.
  MDS7203 Deep Generative Models: Teaching assistant.
  Hugging Face student ambassador program.
- Finance and Economic background.
- Work Experience: 3 years in Macroeconomic Statistics area at the Central Bank of Chile. 2 years at a Fintech joint venture on payments (currently the payment subsidiary of BancoEstado).
- Currently at @ ESO Paranal Software group (winter-internship 2023).

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.

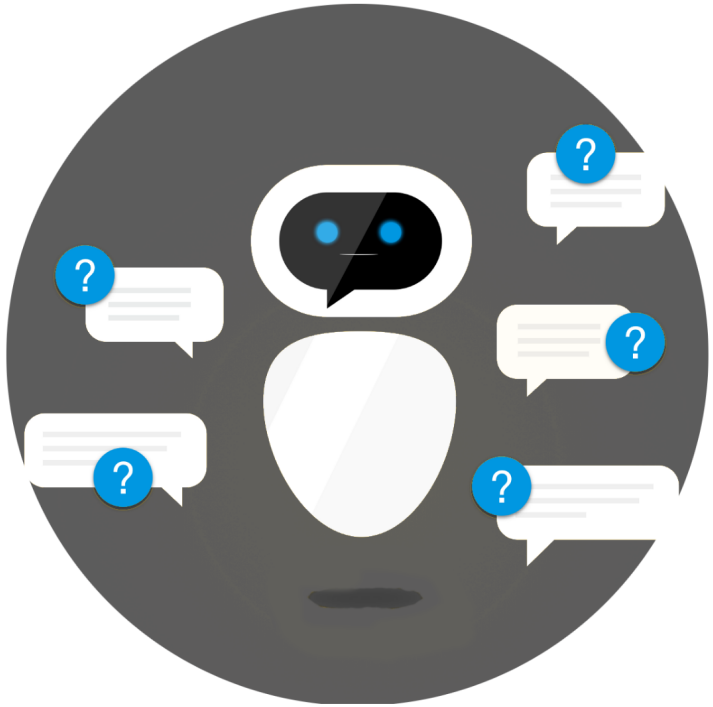
The field has experienced major changes since recent growth of **Deep Learning**.



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



Generative Models

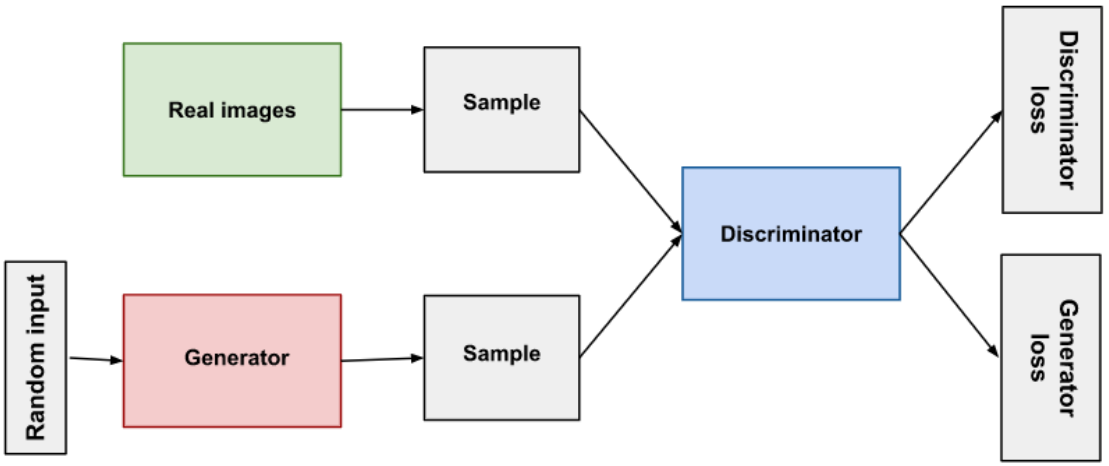
Let $\{x_i\}_{i=1}^N$ be dataset of points in $\mathbb{R}^d$. We will consider does samples come from an **unknown** data distribution $p(x)$. Generative modelling seeks a model that reflects on the data distribution, with which we can **sample**, i.e., to generate new data.

Generative models: what are they?

Some approaches:

Generative adversarial network (GAN)

GANs are able to sample new data points by training a generator network to transform random noise into synthetic samples, while simultaneously training a discriminator network to distinguish between real and generated samples, creating a feedback loop that drives the generator to generate increasingly realistic and diverse data points.[1]



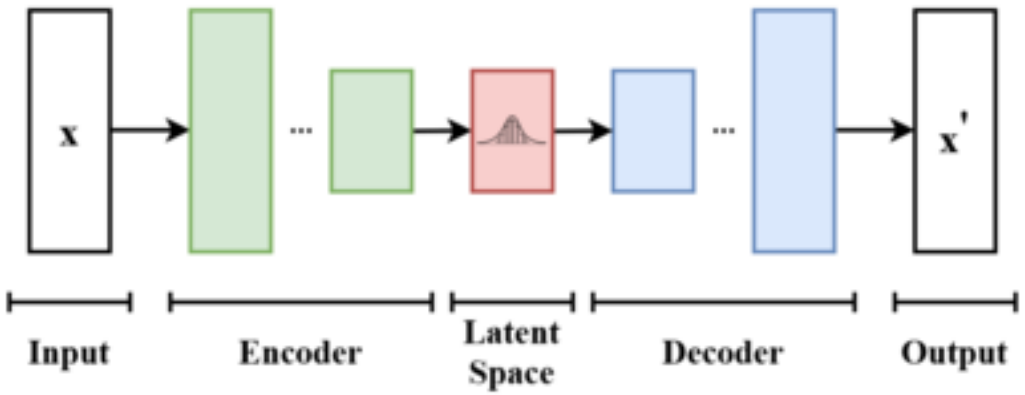
[1] Image source: https://developers.google.com/machine-learning/gan/gan_structure

Generative models: what are they?

Some approaches:

Variational Autoencoders (VAE)

VAEs model the data distribution by learning an encoder network that maps input data to a latent space, and a decoder network that reconstructs the input data from samples generated in the latent space, capturing the underlying structure and generating new data points by sampling from the learned latent distribution.[1]



[1] Image source: https://en.wikipedia.org/wiki/Variational_autoencoder

Generative models: what are they?

Some approaches:

Energy-based models

In this case we will try to model the density by considering:

$$p_\theta(x) = \frac{\tilde{p}_\theta(x)}{Z_\theta} = \frac{e^{-E_\theta(x)}}{Z_\theta},$$

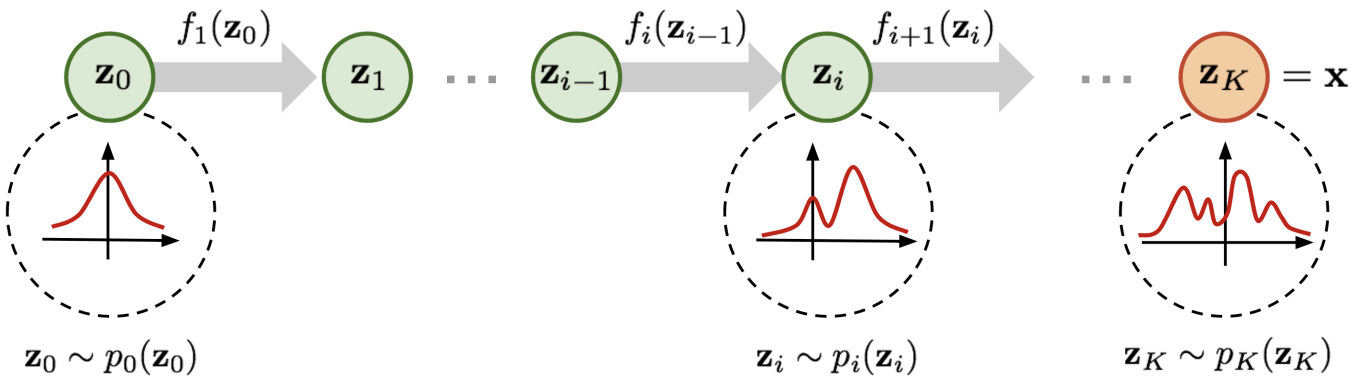
in which $E_\theta : \mathbb{R} \to \mathbb{R}$ is called the energy function. The analogy is: the lower energy the more likely a data point will be. On the other hand, Z_θ simply a normalising factor, which is considered intractable. This type of modelling is more flexible but sampling is carried out choosing samples with low energy using MCMC methods, which sometimes lead to innacuracies.

Generative models: what are they?

Some approaches:

Normalising Flows

Normalizing Flows model the data distribution by applying a sequence of invertible transformations to a simple base distribution, such as a Gaussian, enabling the model to learn and generate samples from complex data distributions by iteratively transforming the data and updating the probability density through the change of variables formula.[1]

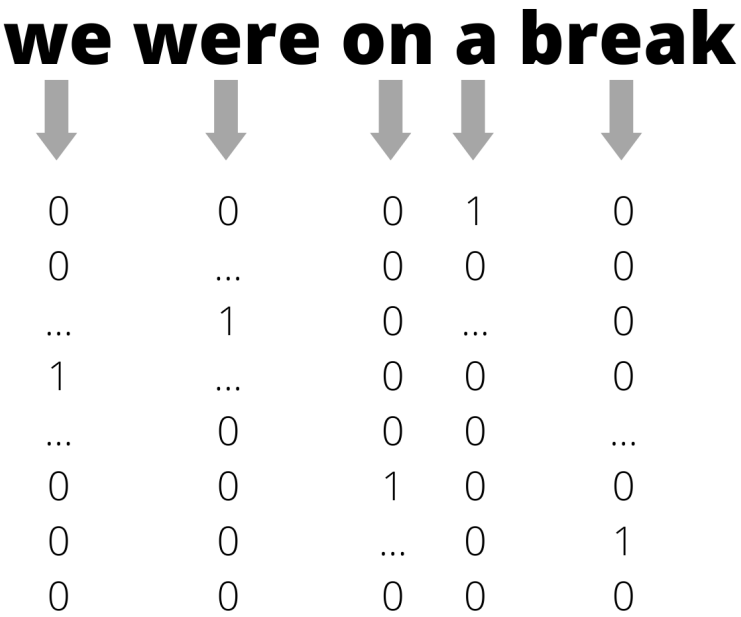


[1] Image source: https://lilianweng.github.io/posts/2018-10-13-flow-models/

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

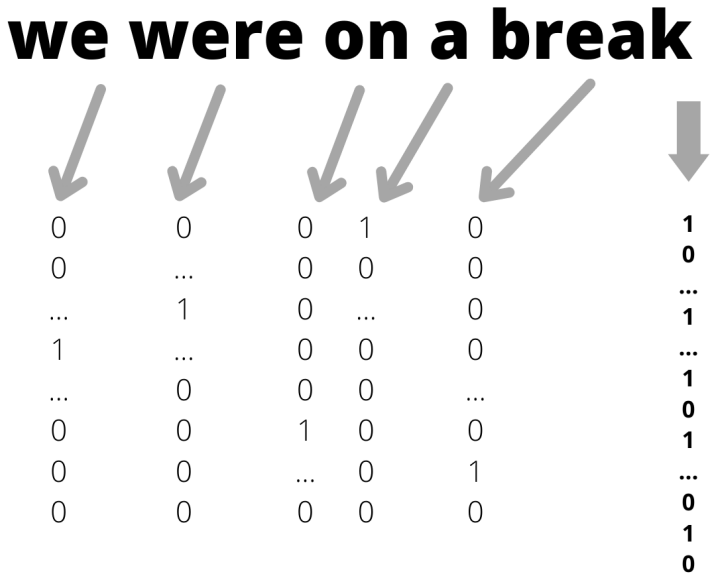


Table of contents

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**[2]:

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

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

[2] 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:

$$p_{ML}(x_{p+1}, \ldots, x_T | x_1, \ldots, x_p) = \prod_{i=p+1}^{T} p(x_i | x_1, \ldots, x_{i-1})$$

$$= \prod_{i=p+1}^{T} \frac{s(x_1, \ldots, x_p, \ldots, x_i)}{\sum_{y \in V} s(x_1, \ldots, x_p, \ldots, x_{i-1}, y)},$$

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:

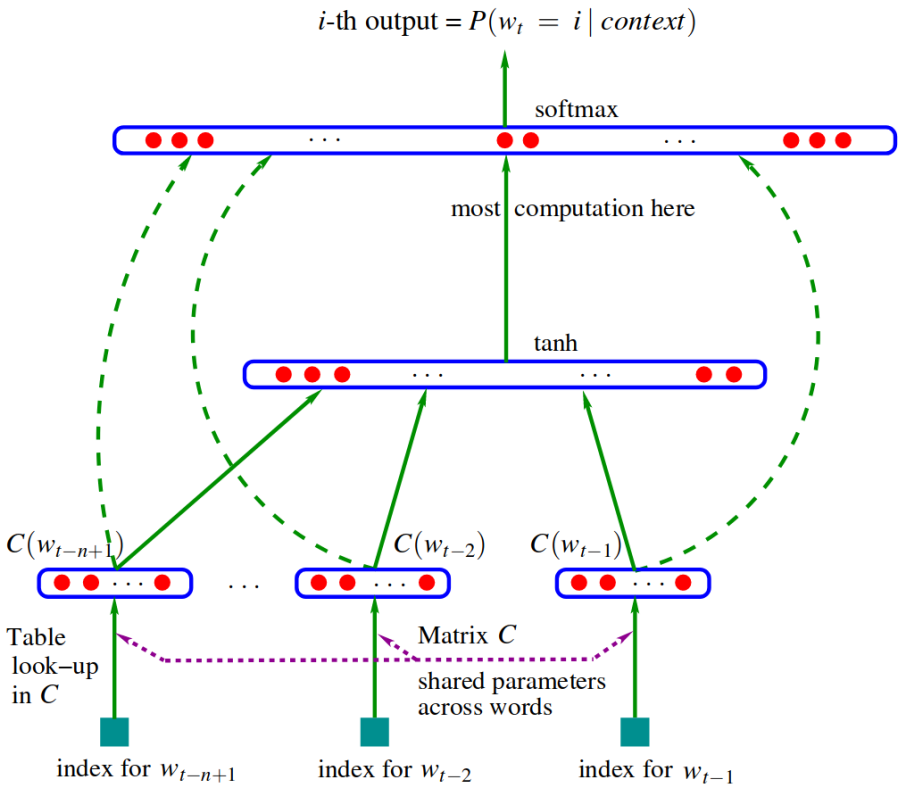
1. associate with each word in the vocabulary a distributed word feature vector (a real valued vector),
2. express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the word feature vectors and the parameters of 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. [2]. However, this is arguably not the best option to encode language.

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

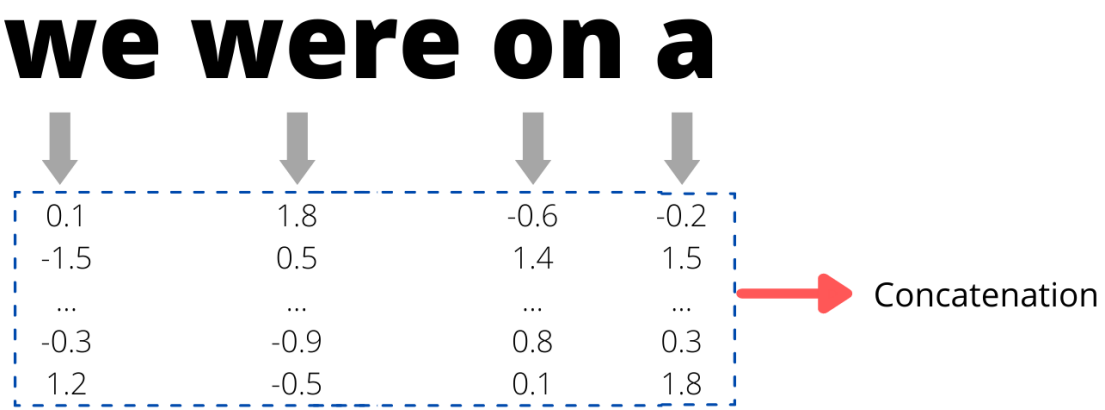
- Recurrent Neural Networks (RNN)
- Transformers



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. 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:

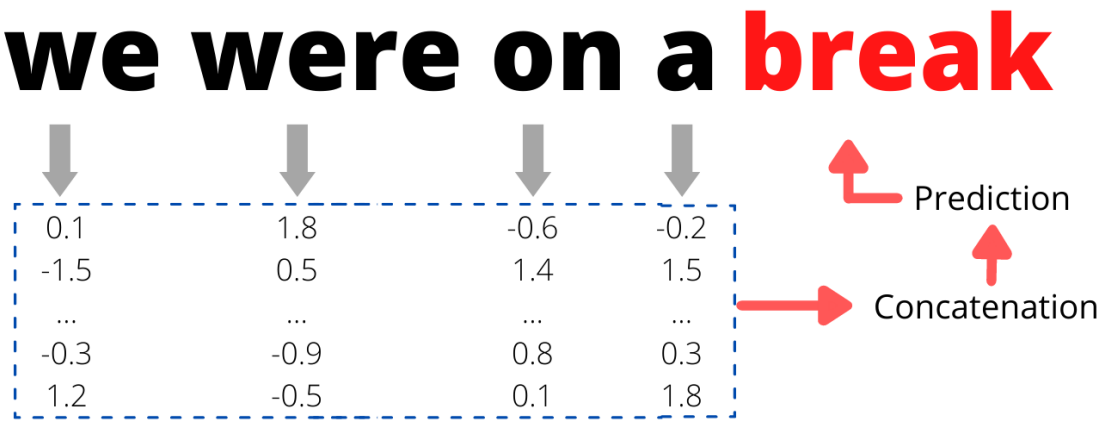
- **Next word prediction (NWP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. 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

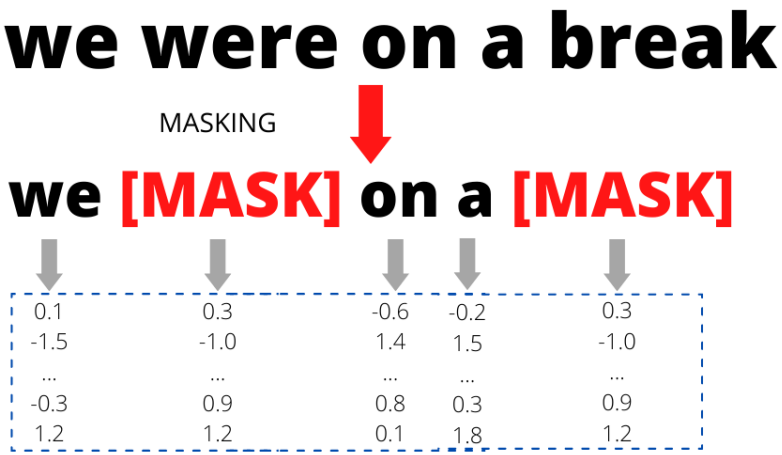
Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. Neural LMs are characterised by making use of word embeddings. But, how do we assign a probability distribution to documents or sentences? We'll discuss three methods:

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

Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. 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:

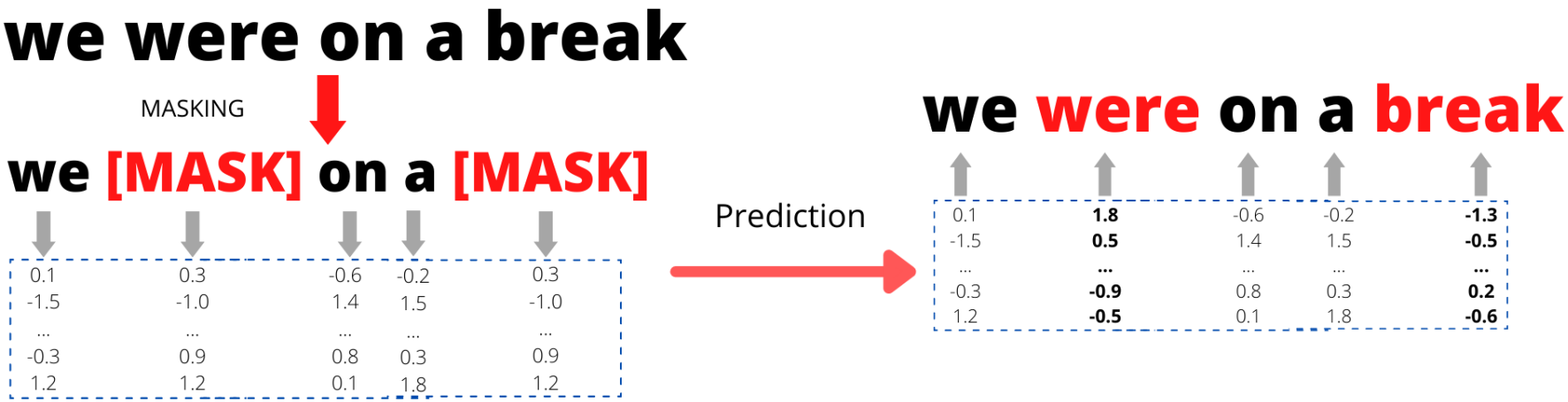
- **Next word prediction (NWP)**
- **Masked Language Model (MLM)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. 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:

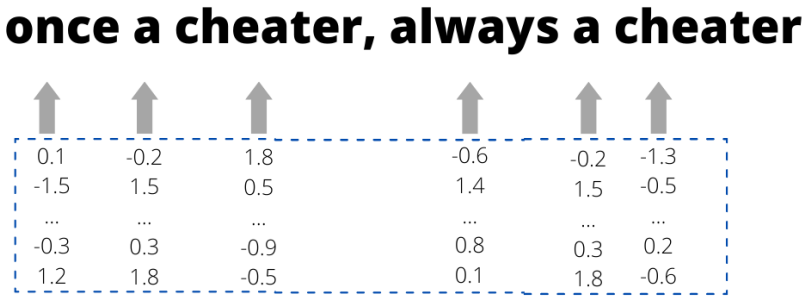
- **Next word prediction (NWP)**
- **Masked Language Model (MLM)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. 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:

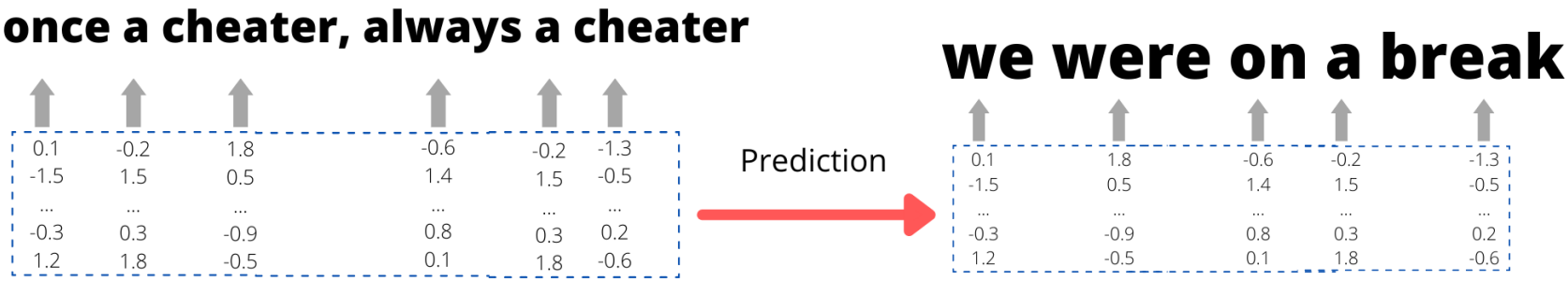
- **Next word prediction (NWP)**
- **Masked Language Model (MLM)**
- **Next sentence prediction (NSP)**



Learning Objectives

Back in 2003, Bengio et al. 2003 introduced on the first **Neural Language Models** [2]. 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[3].

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.

[3] 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. [2]. 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

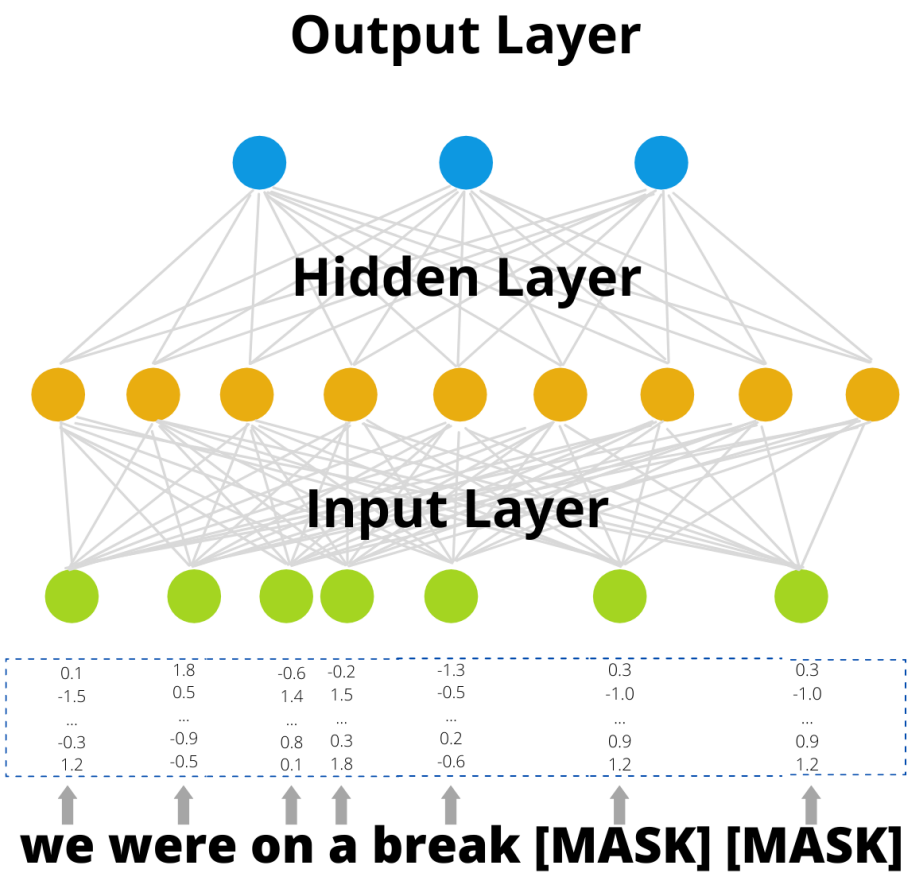
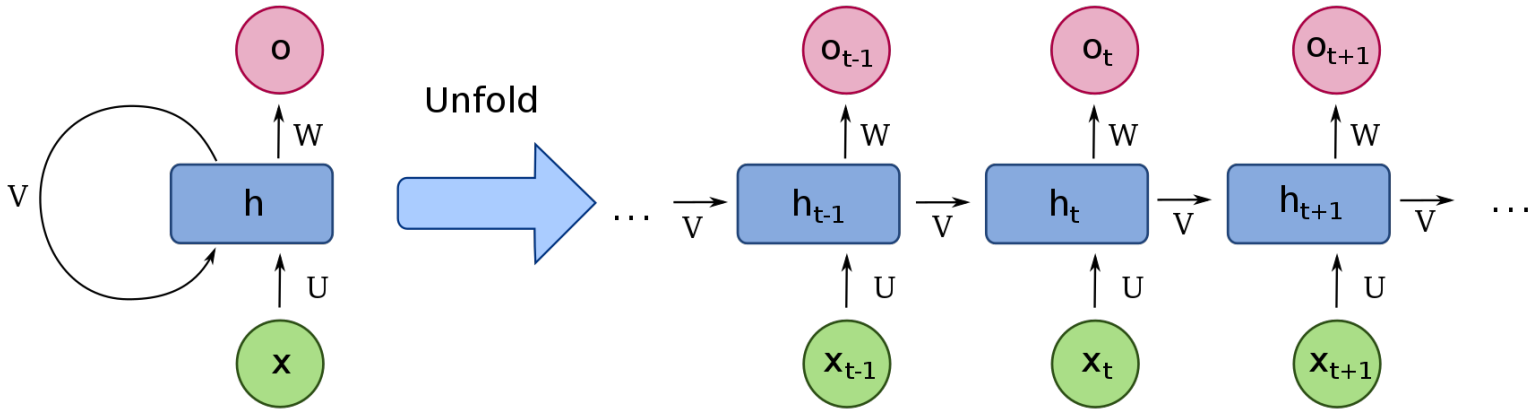


Table of contents

Recurrent Neural Networks

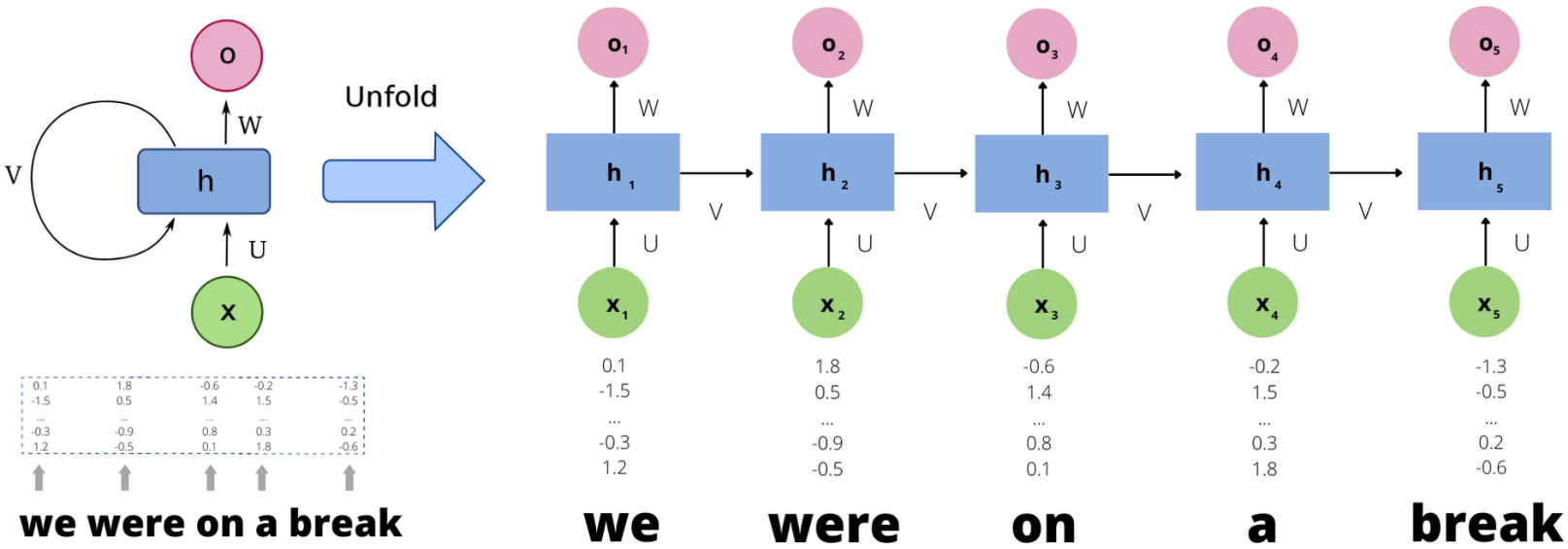
RNNs are an ideal solution to deal with sequences of unknown length. In its simple version (**Vanilla RNN**) we simply apply a neural network to generate an **output** and a hidden state to every member of the sequence.



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

Recurrent Neural Networks

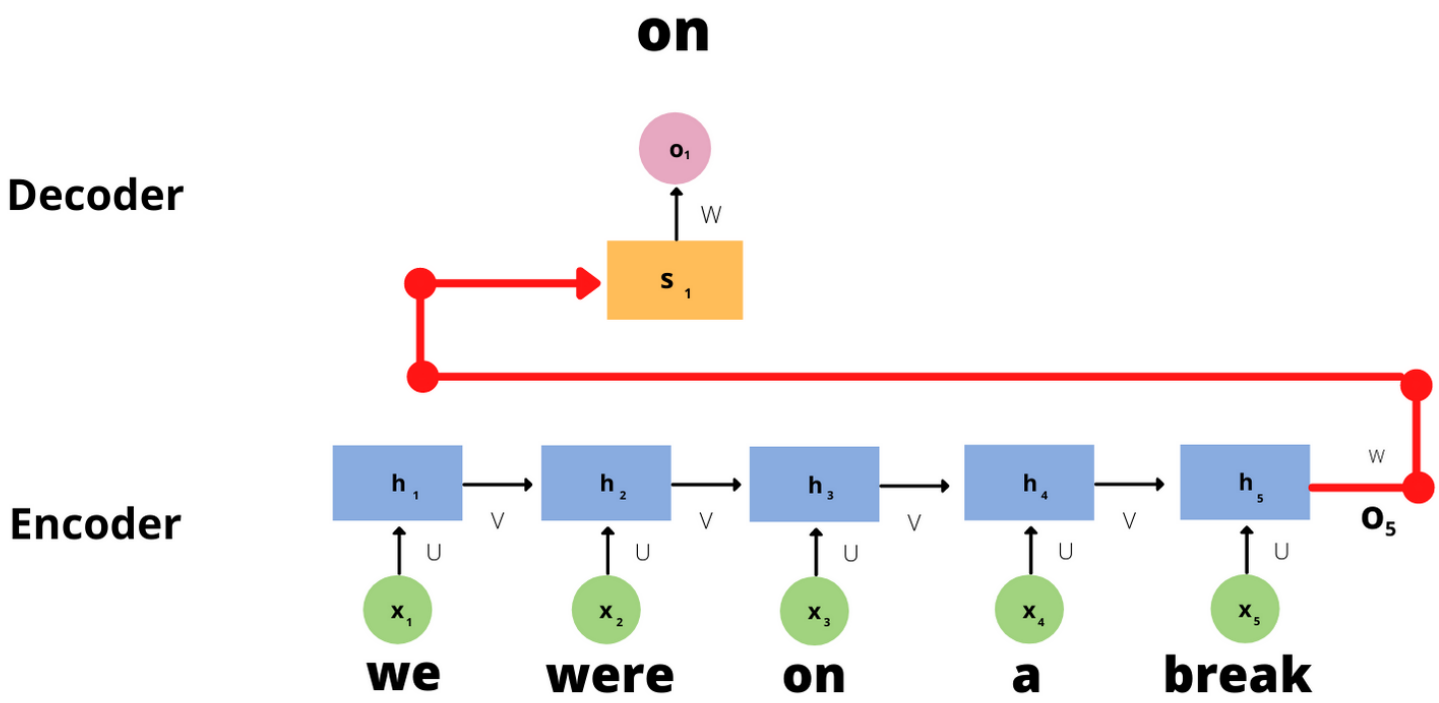
RNNs are an ideal solution to deal with sequences of unknown length. In its simple version (**Vanilla RNN**) we simply apply a neural network to generate an **output** and a hidden state to every member of the sequence.



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

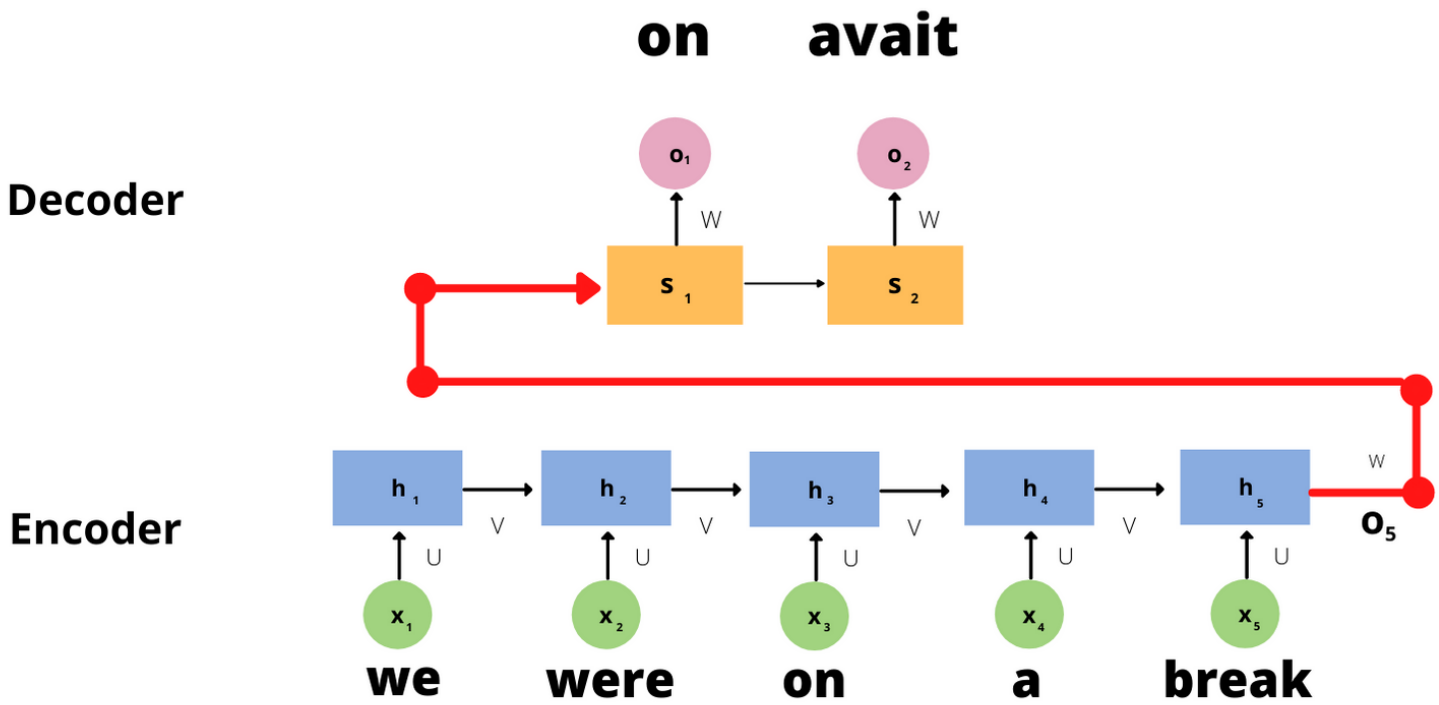
RNNs: a Machine Translation example

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.



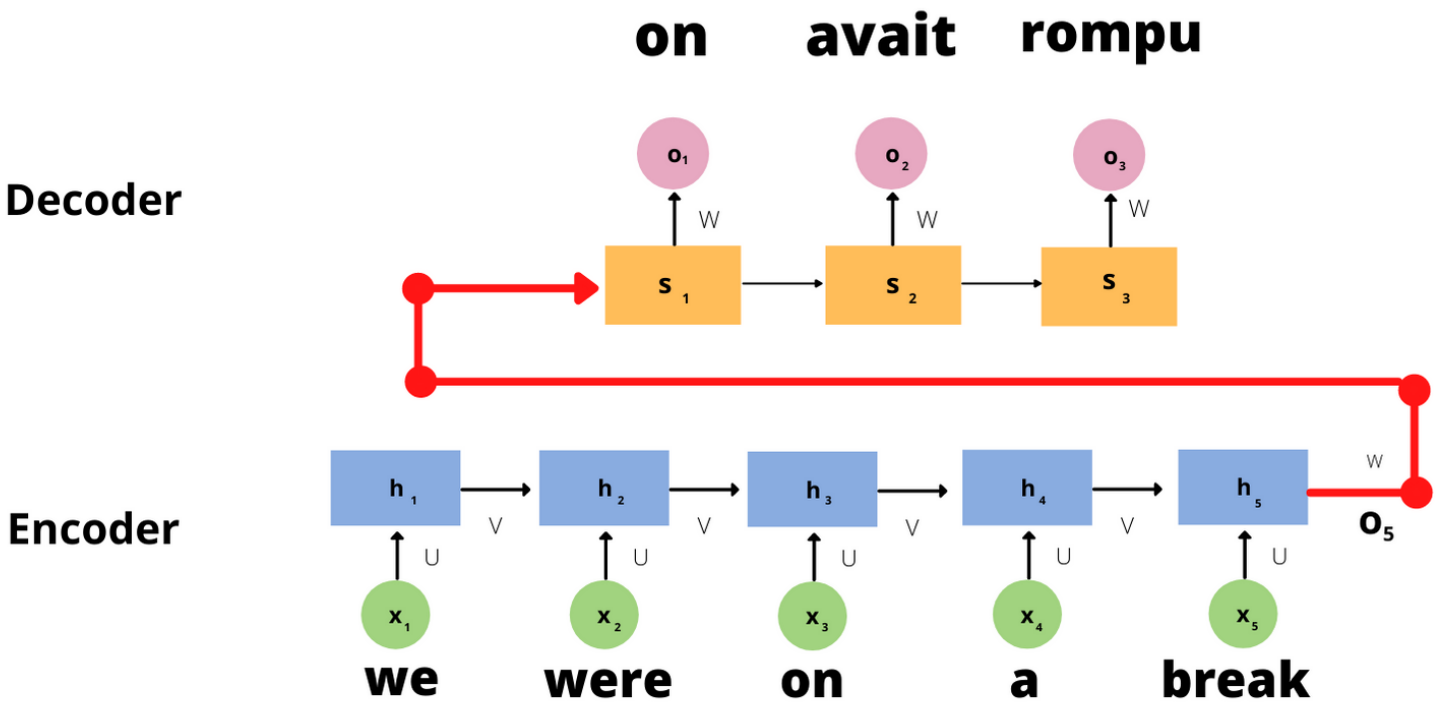
RNNs: a Machine Translation example

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.



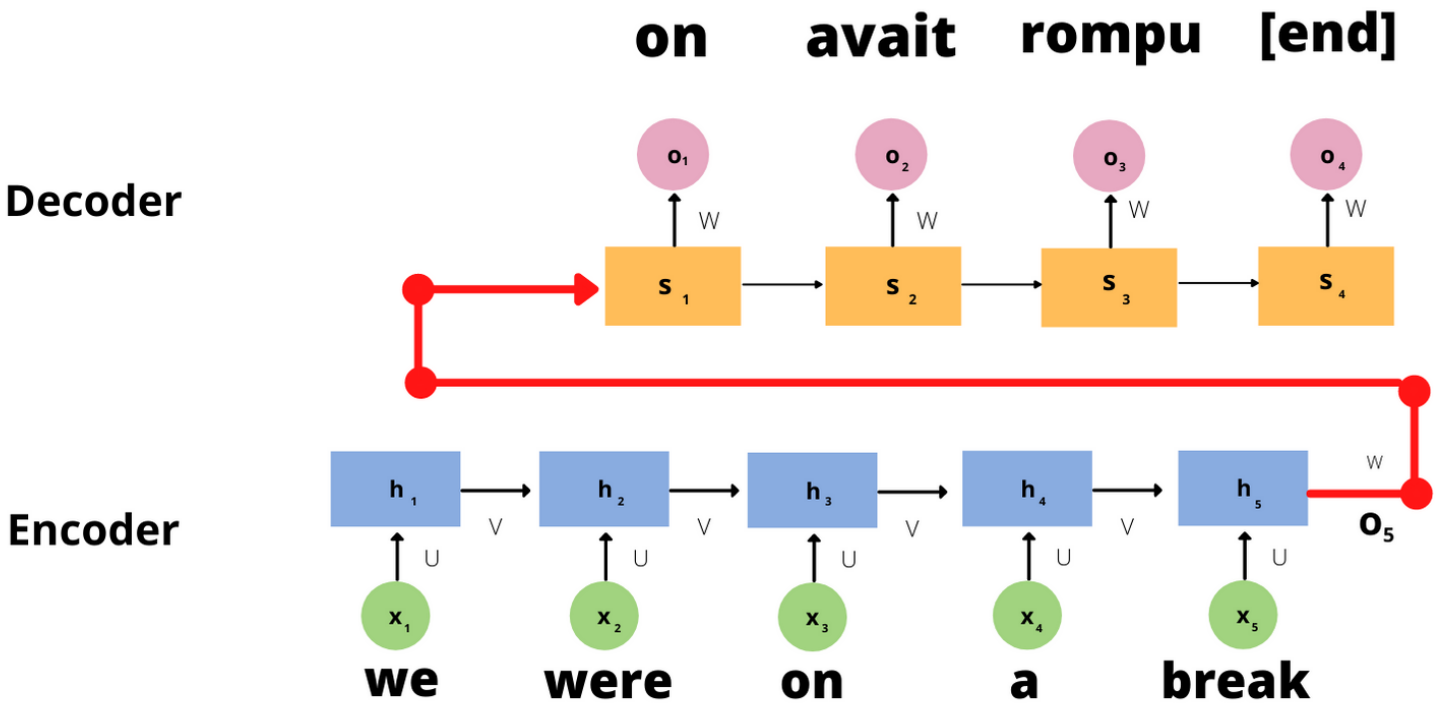
RNNs: a Machine Translation example

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.



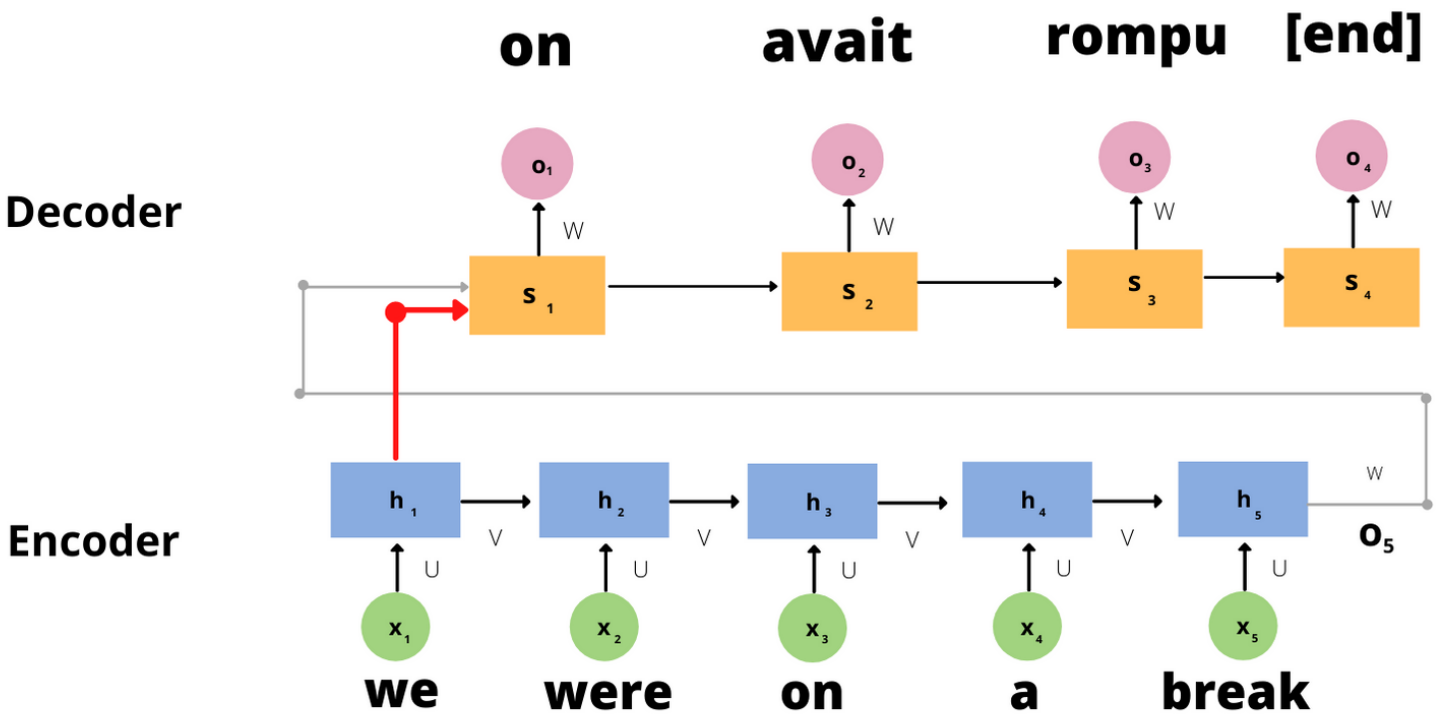
RNNs: a Machine Translation example

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.



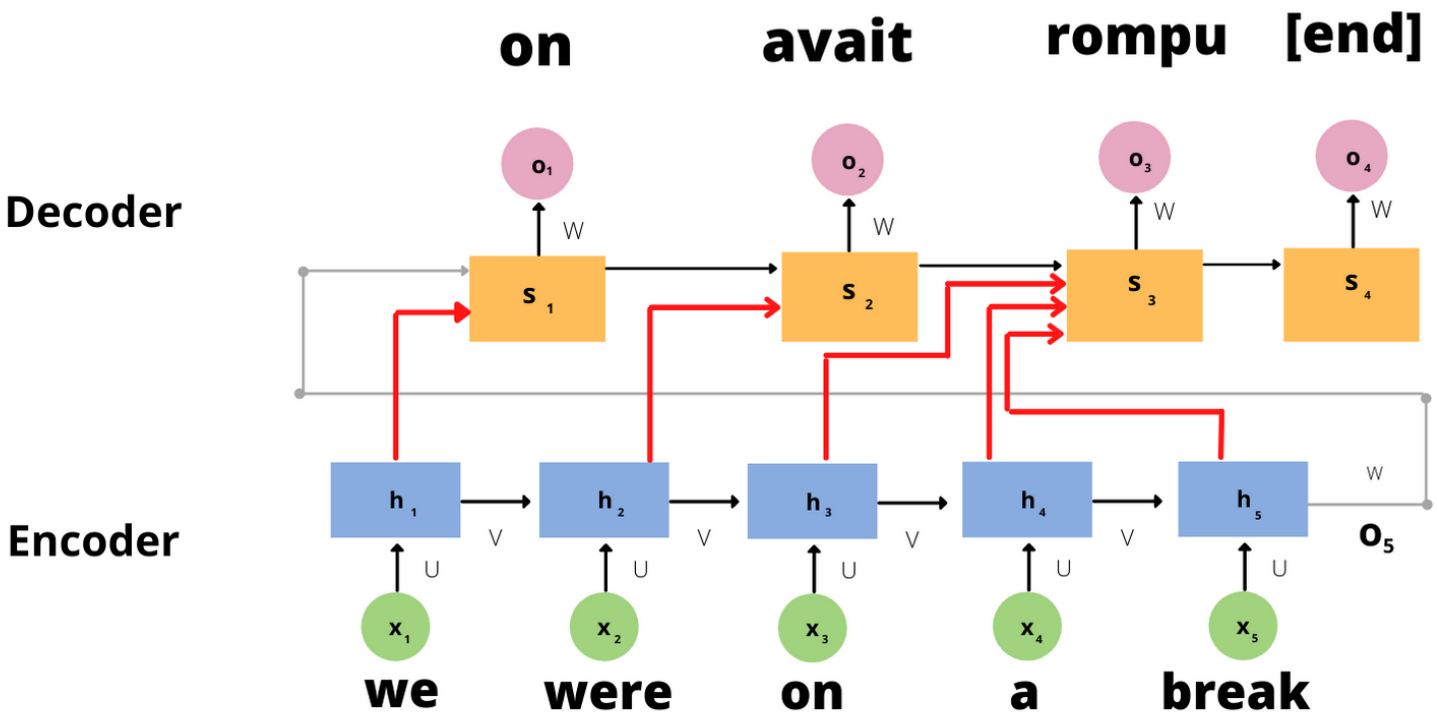
Attention

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:



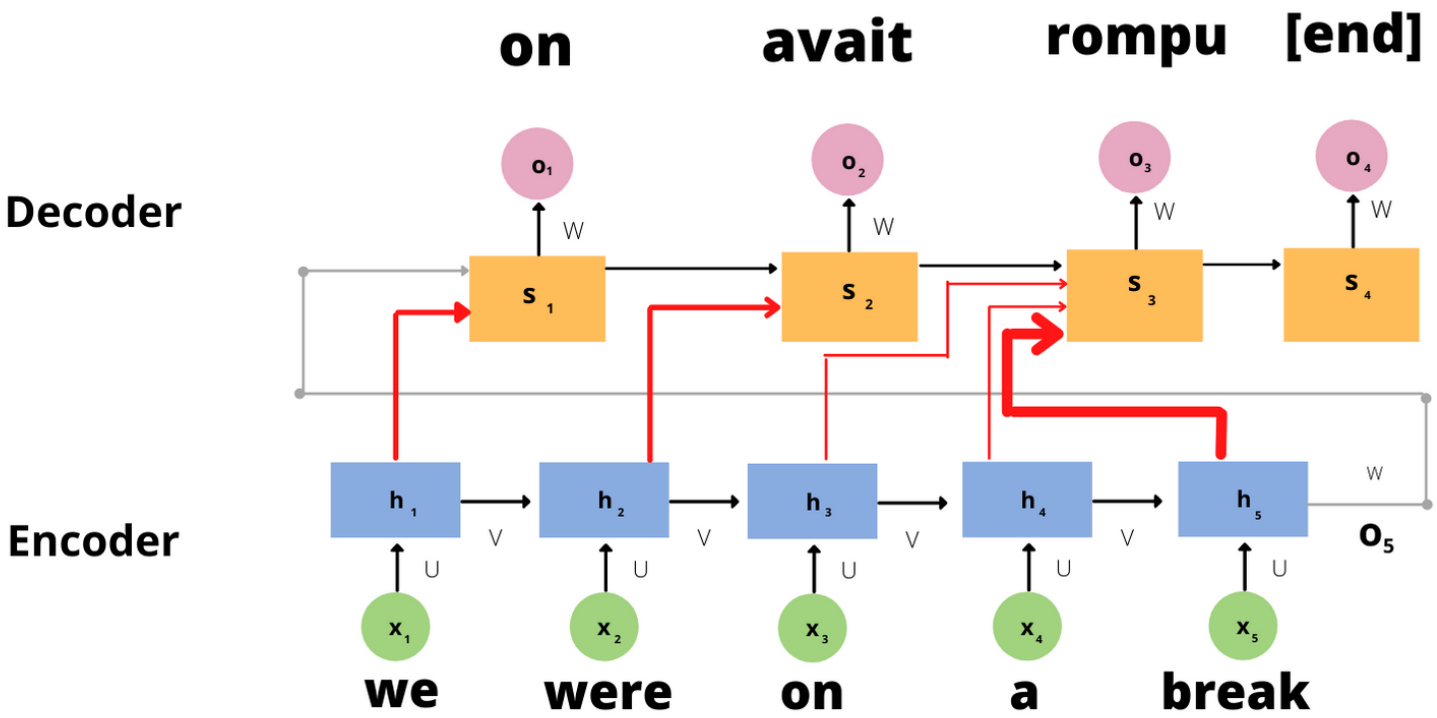
Attention

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:



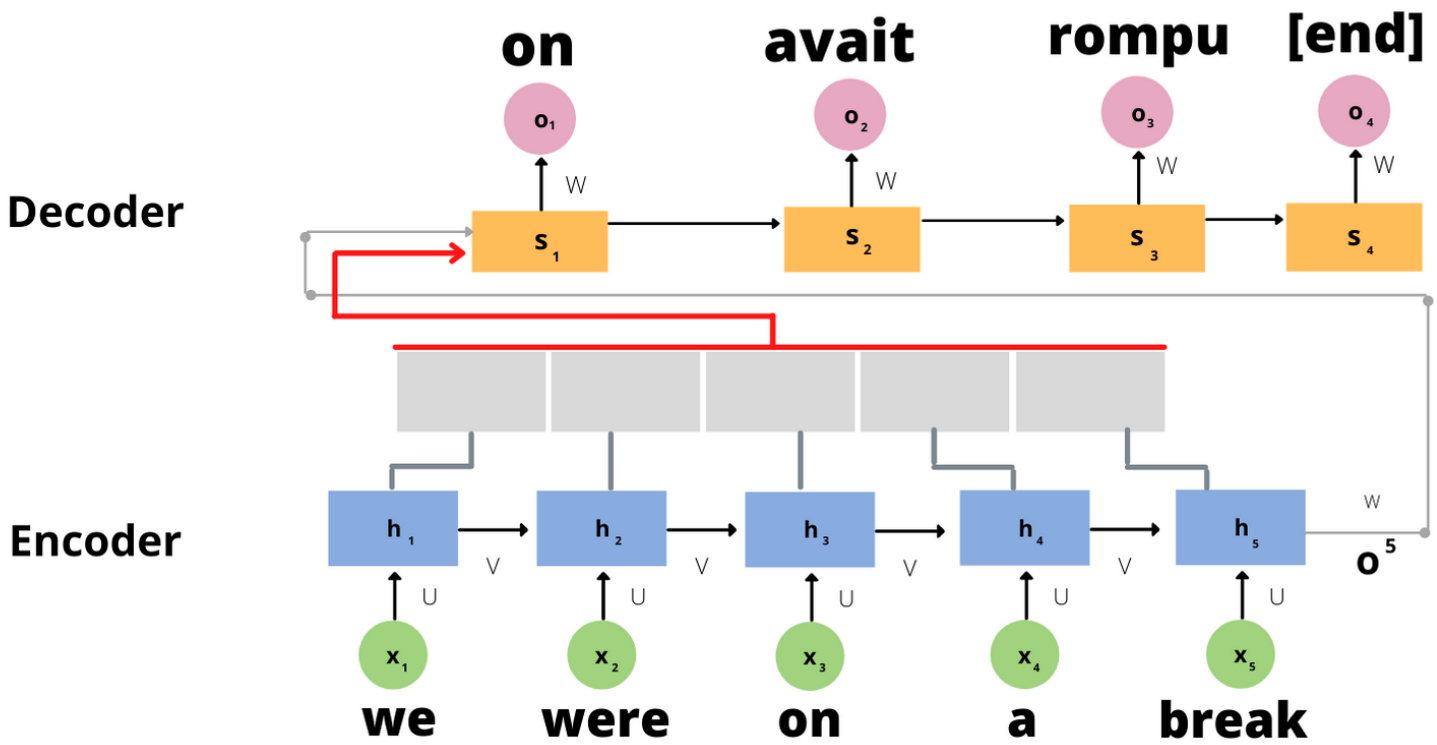
Attention

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**.



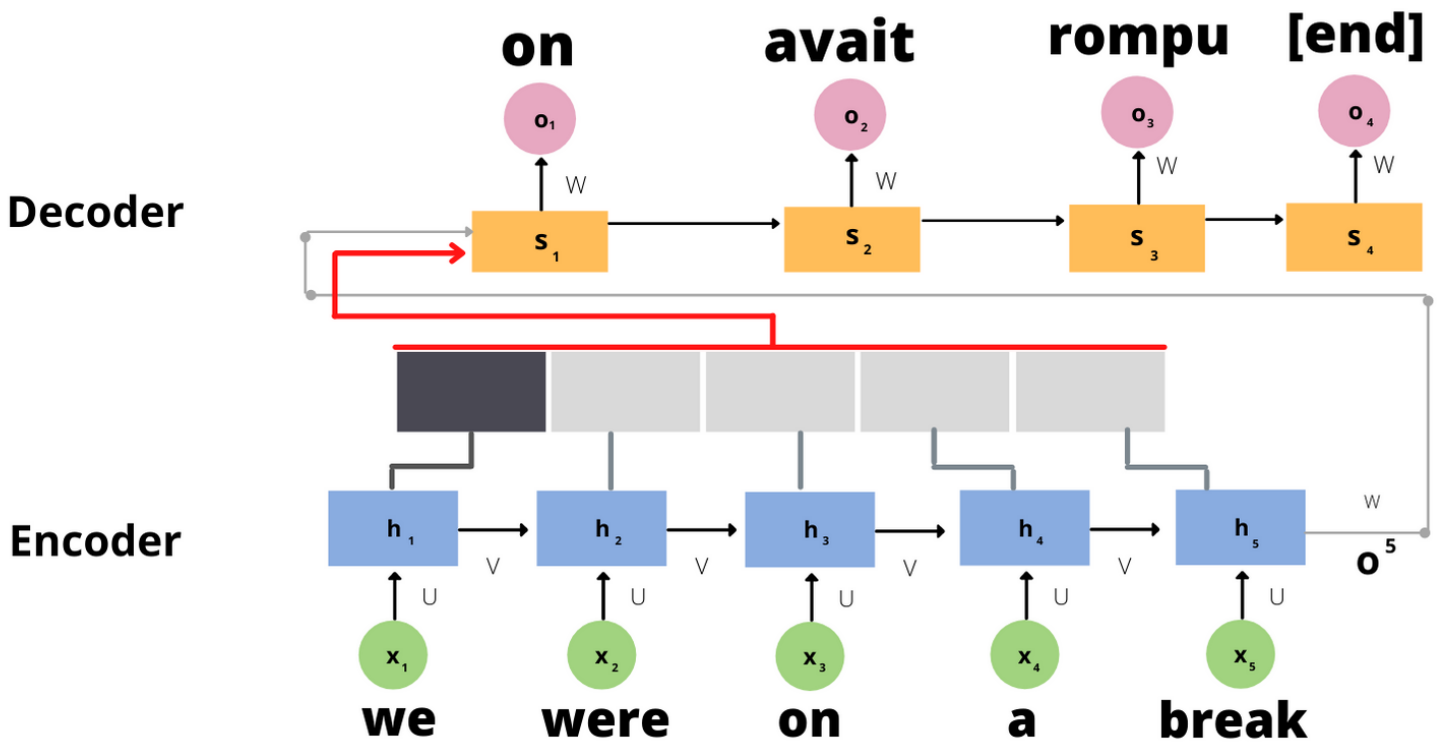
Attention Modules

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.



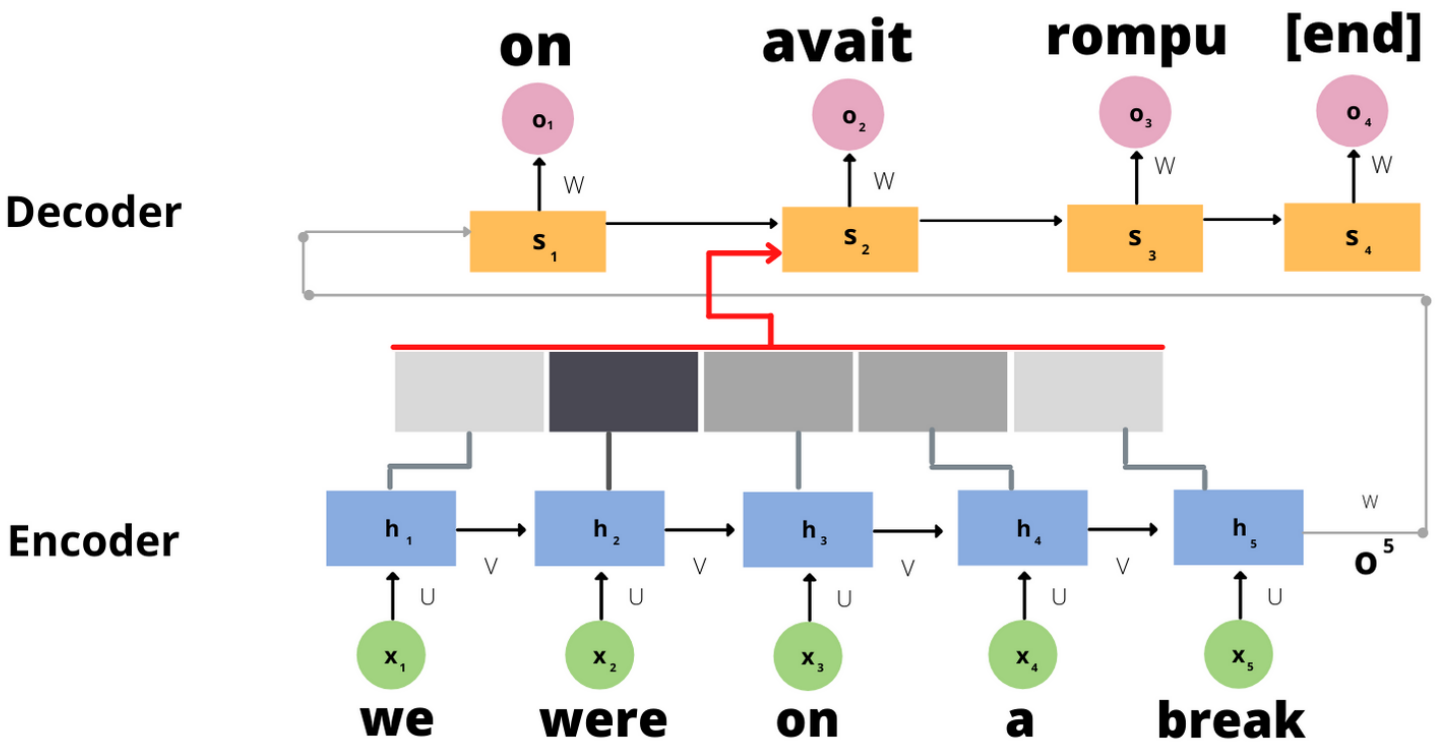
Attention Modules

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.



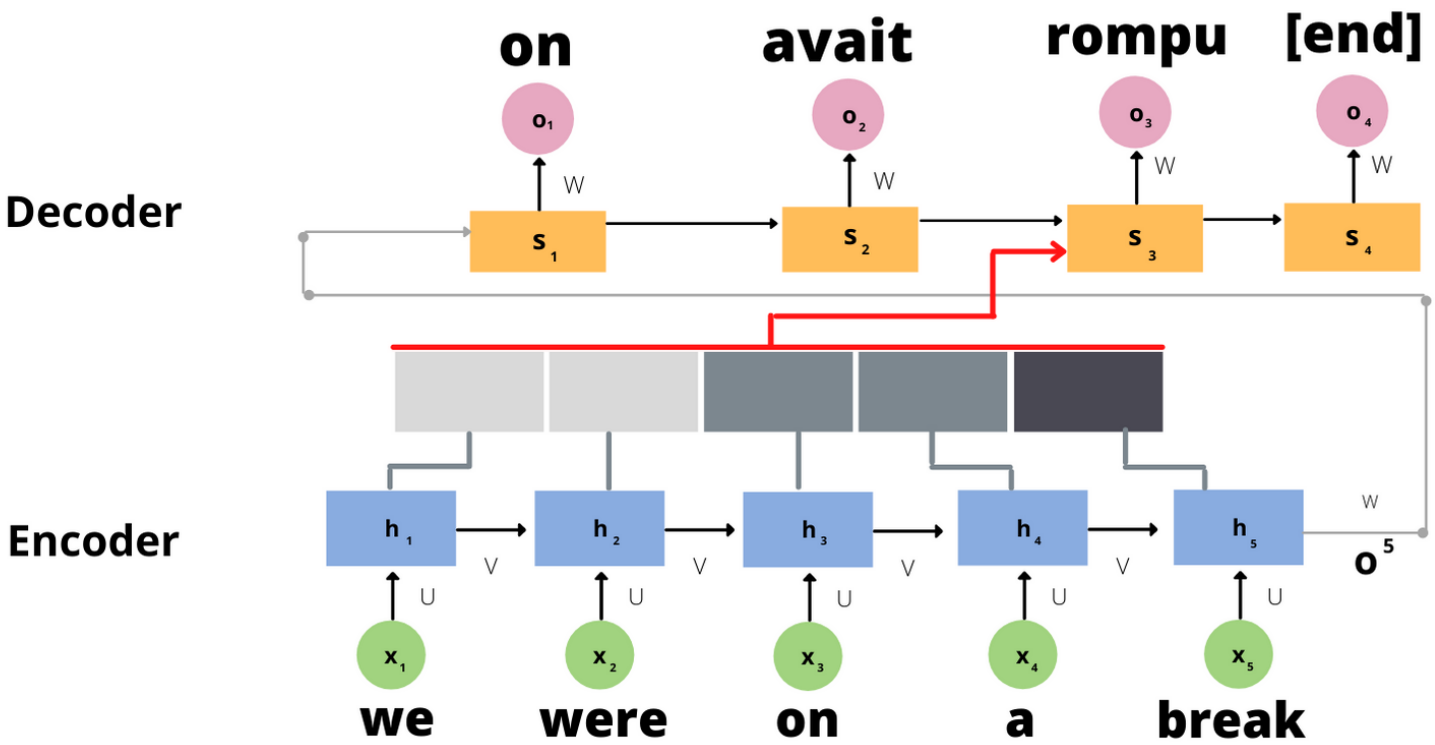
Attention Modules

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.



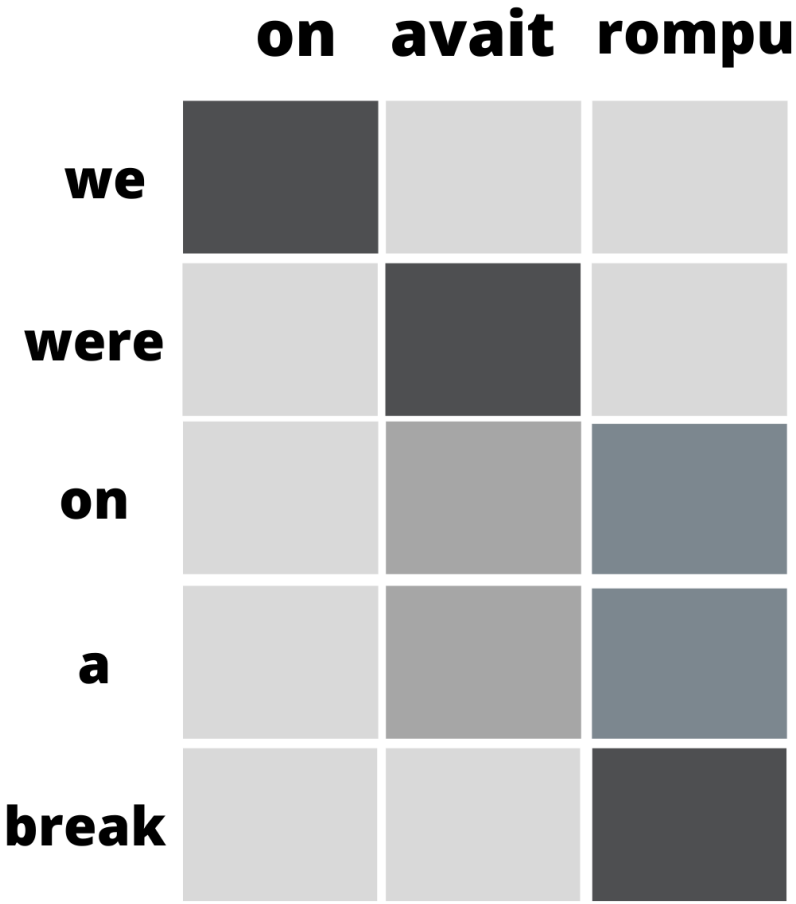
Attention Modules

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.



European Southern Observatory

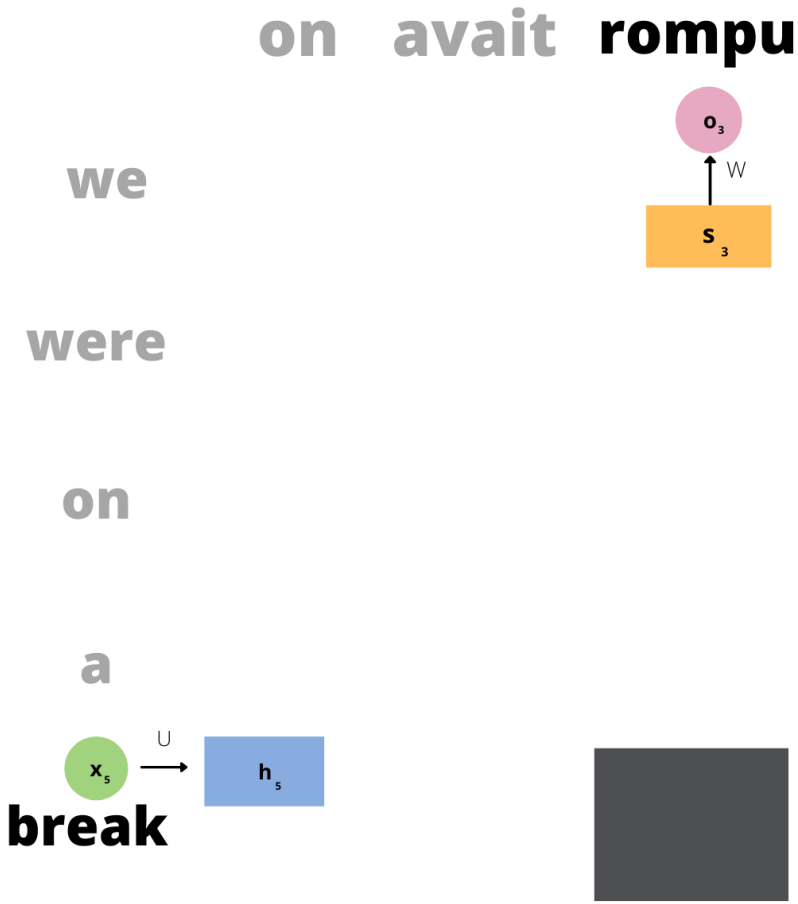
Attention Modules

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.



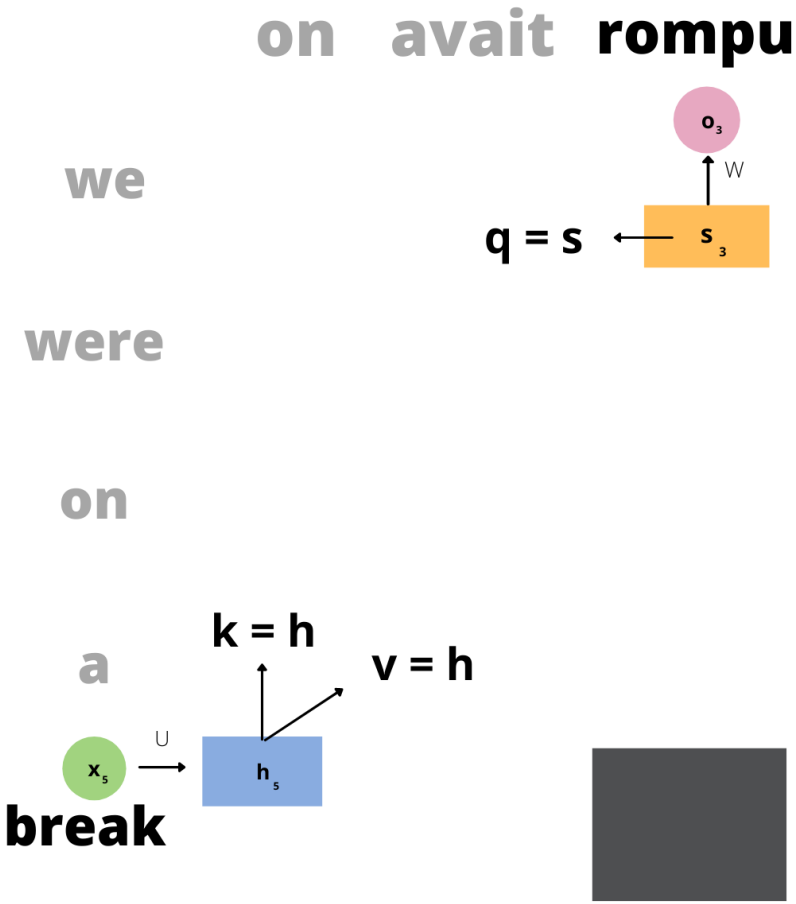
Attention Modules

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.



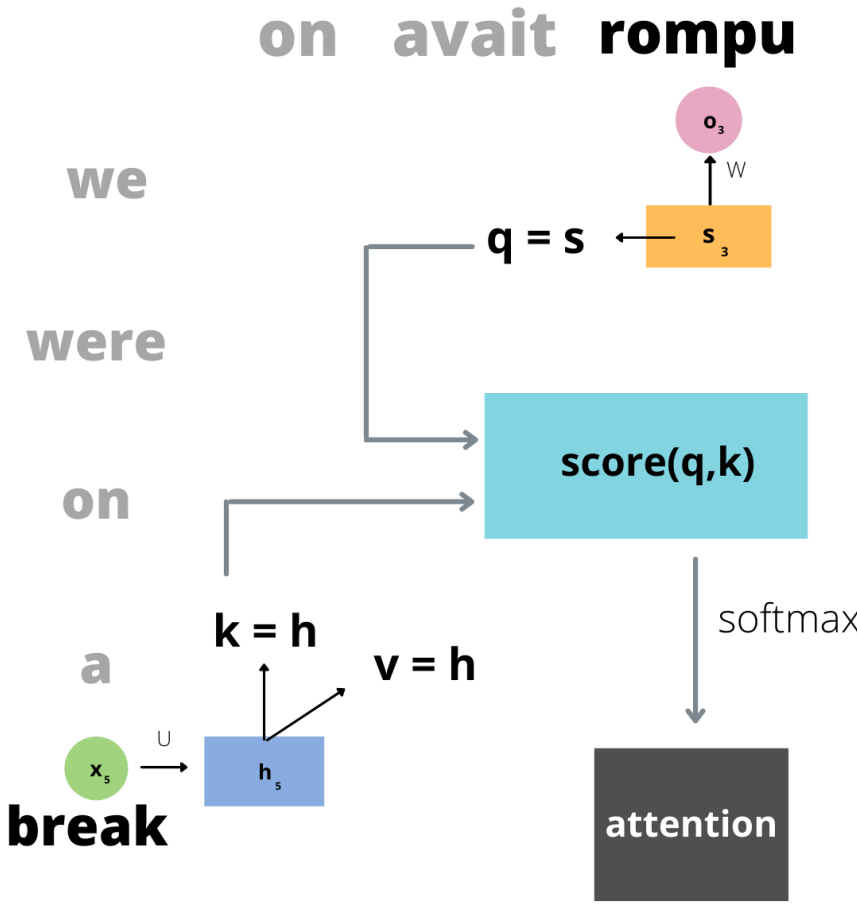
Attention Modules

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.



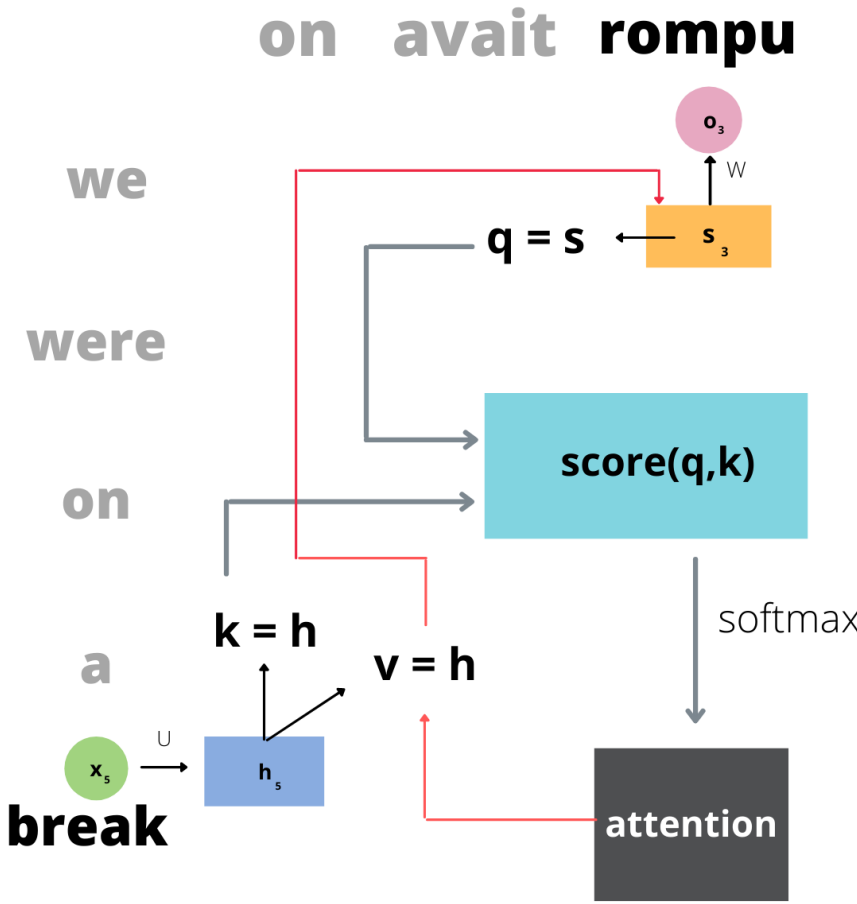
Attention Modules

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.



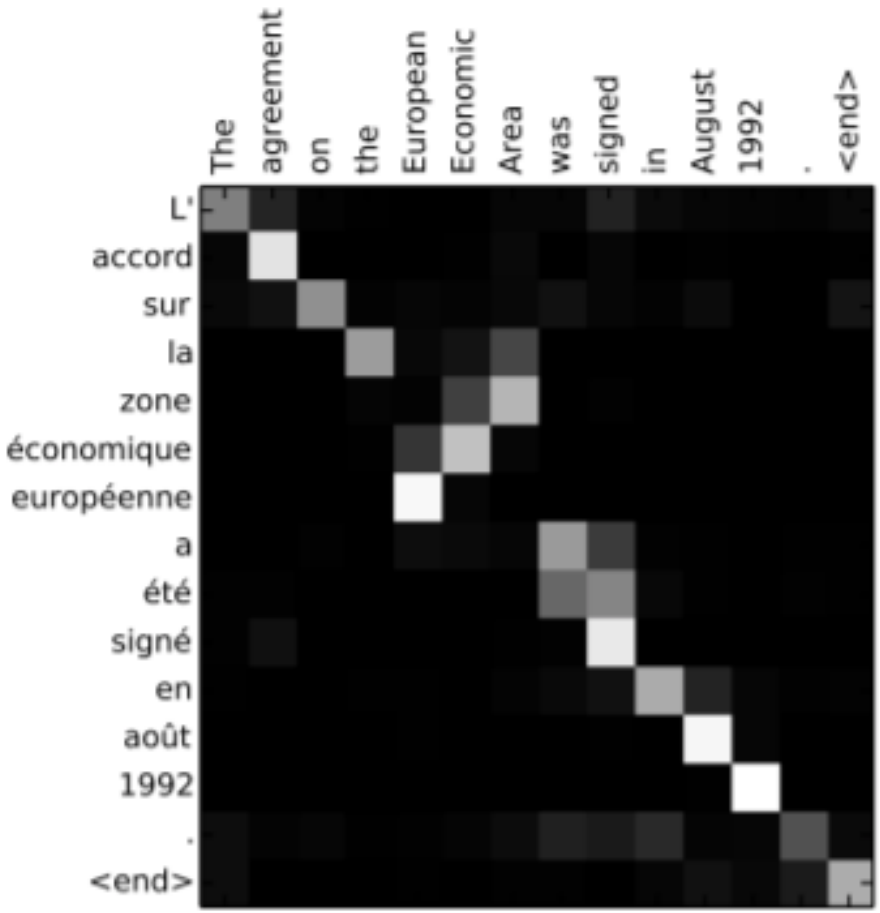
Attention Modules

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

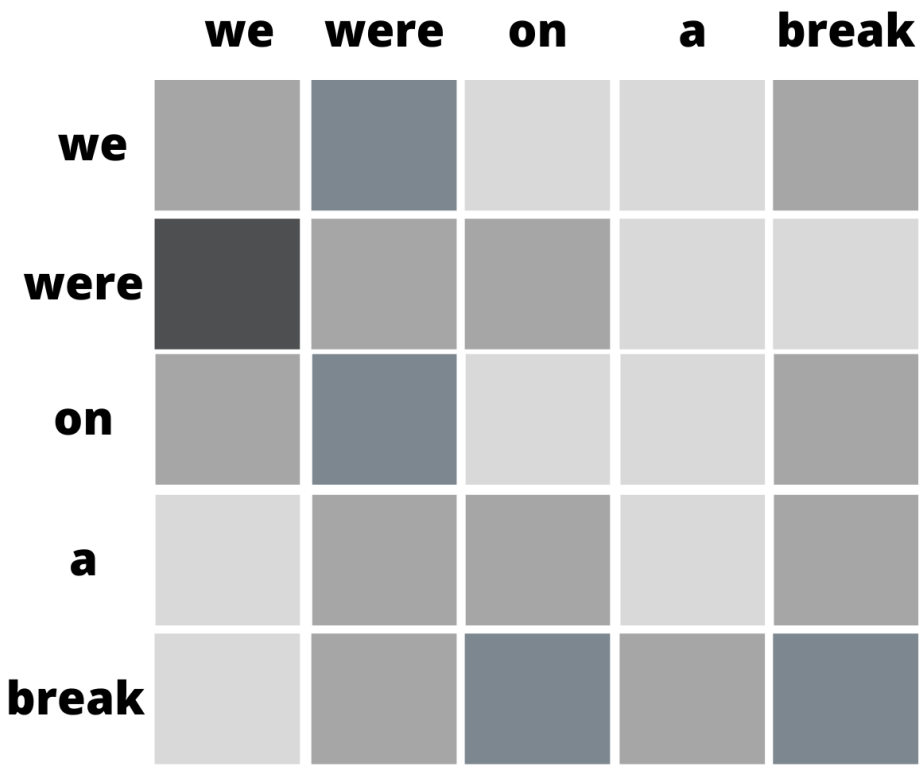
Self-attention: an attention network applied to the same sequence.

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

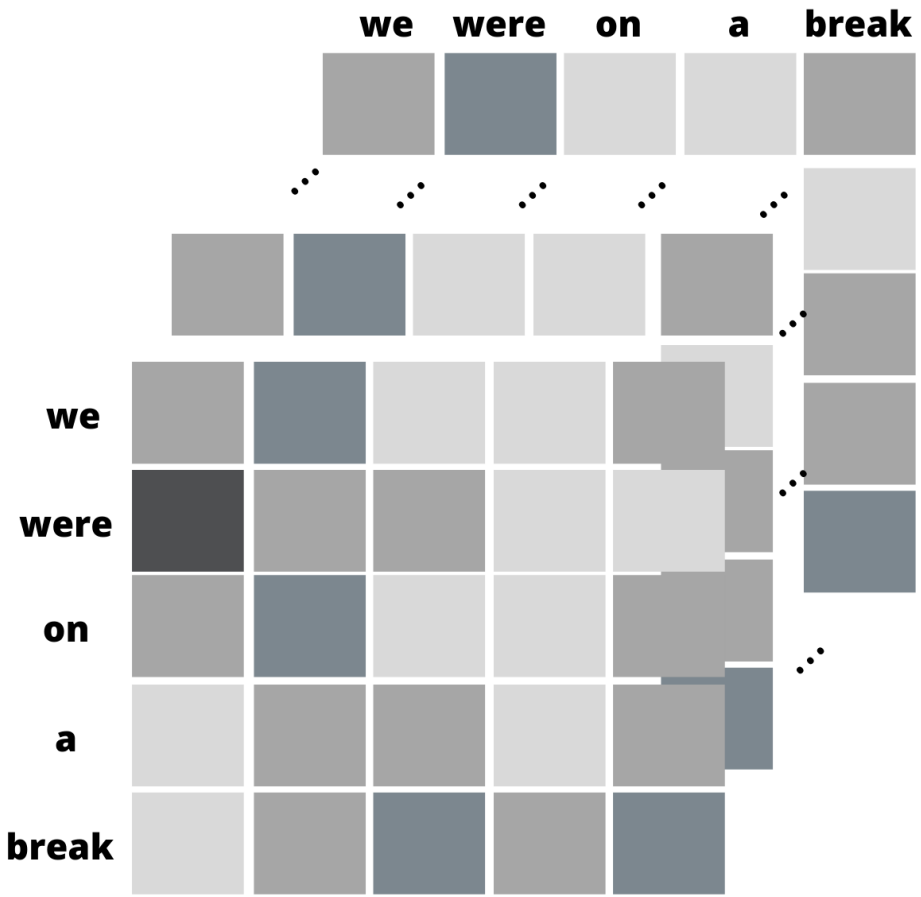
Self-attention: an attention network applied to the same sequence.

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

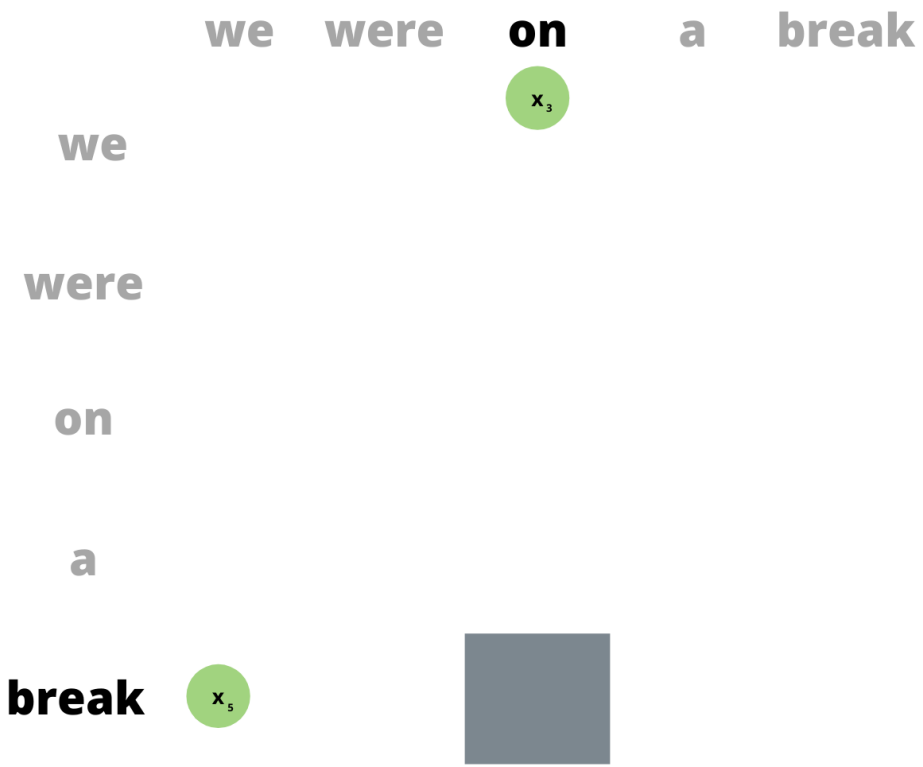
Self-attention: an attention network applied to the same sequence.

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

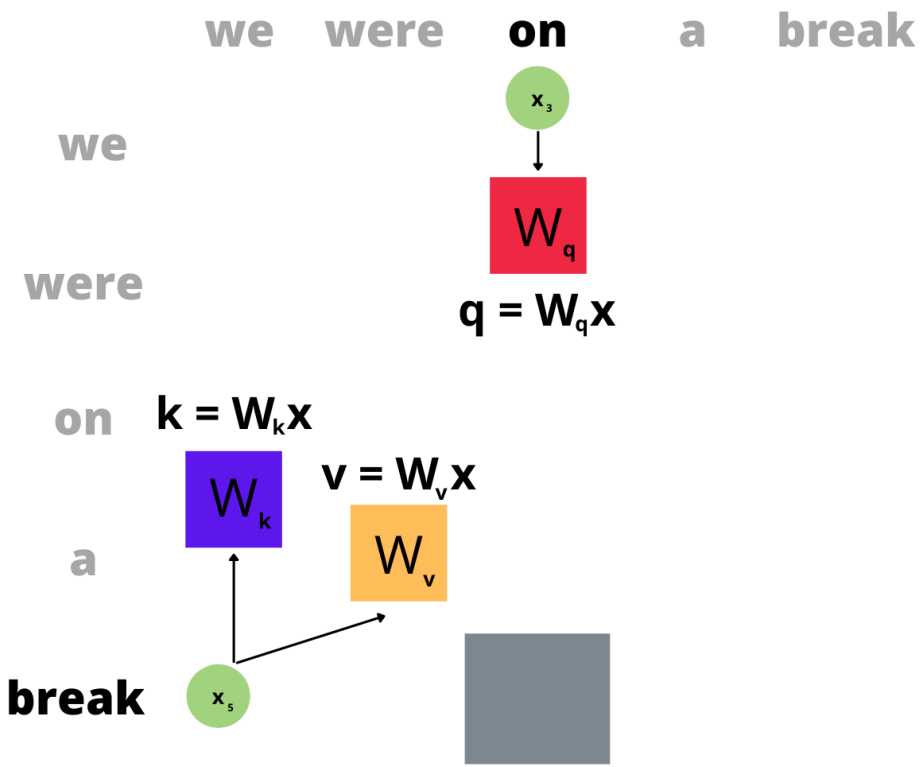
Self-attention: an attention network applied to the same sequence.

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

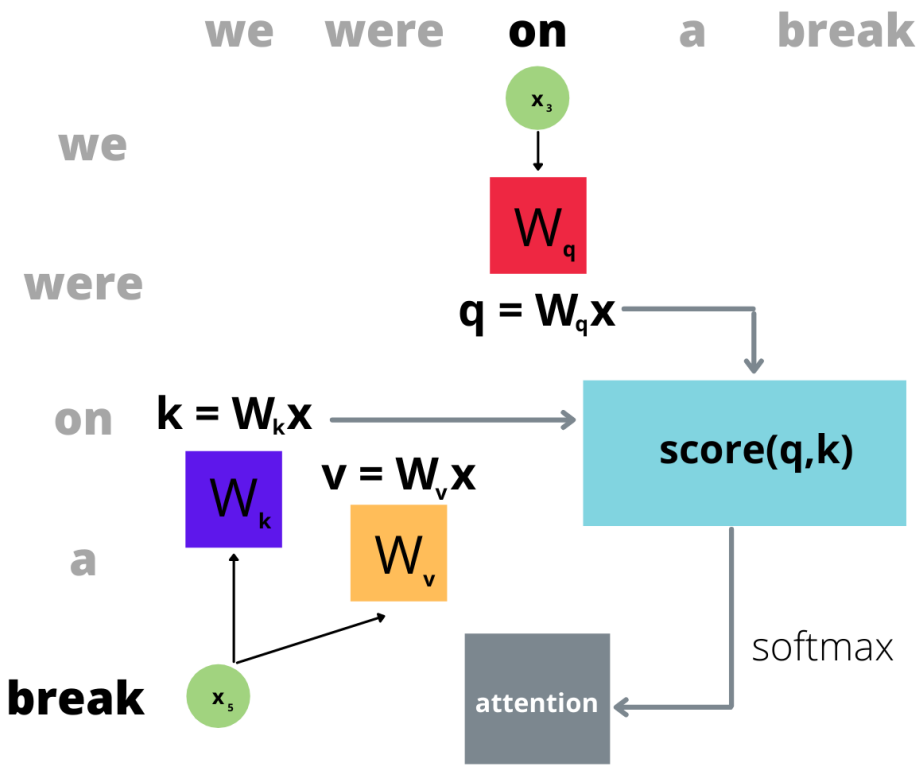
Self-attention: an attention network applied to the same sequence.

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

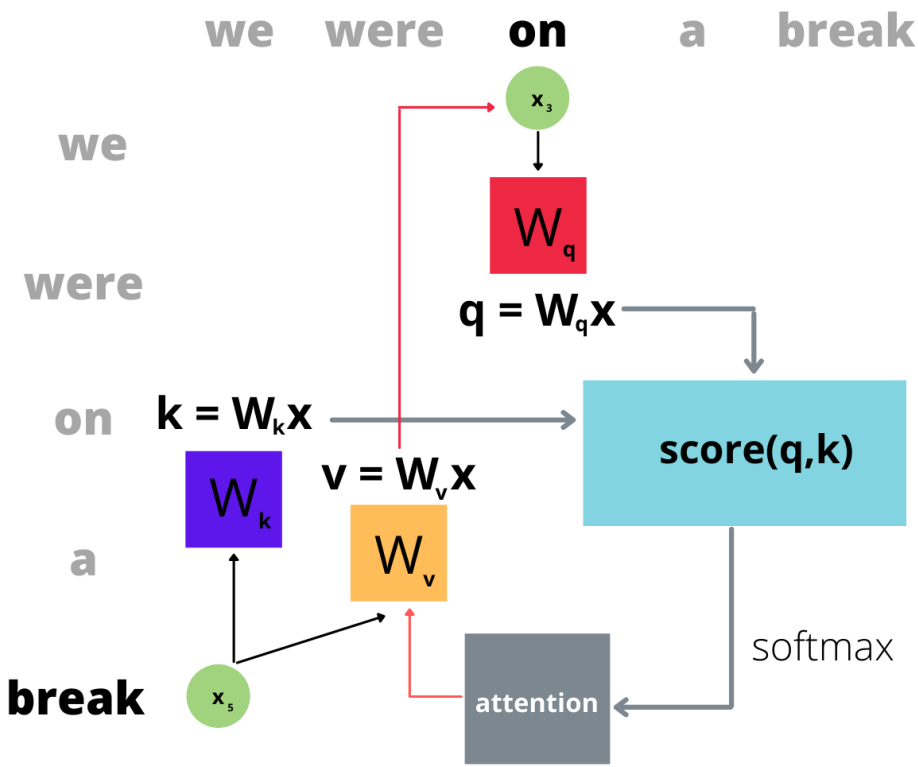
Self-attention: an attention network applied to the same sequence.

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

There is no longer a recurrent network involved.

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

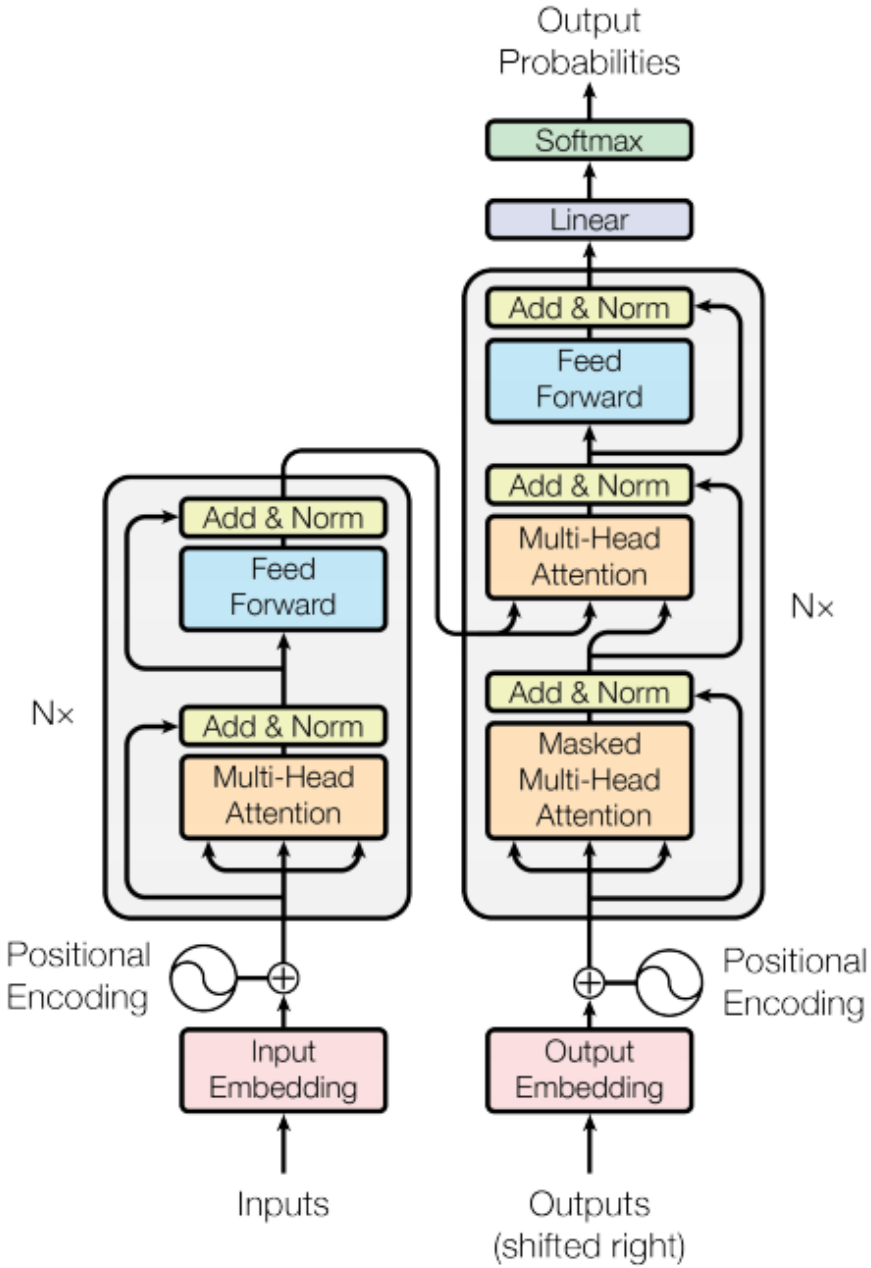
Q, K and V are computed from input/output using **weight matrices** W_q, W_K and W_V



The Transformer

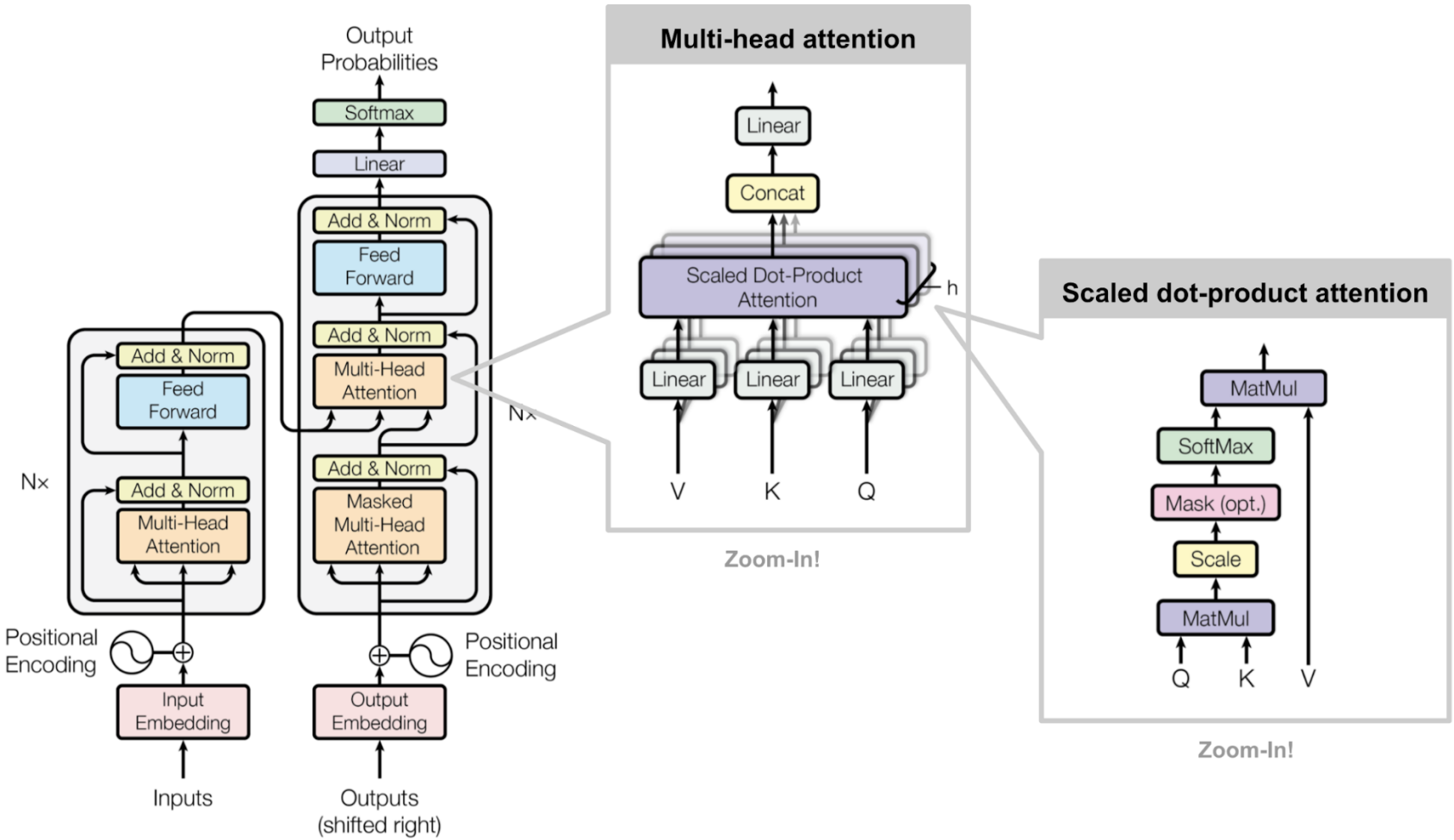
Transformers are a novel architecture that **gets rid of any recurrency**, thus relying solely on **attention modules** [5]. They have the following advantages:

- 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.



The Transformer

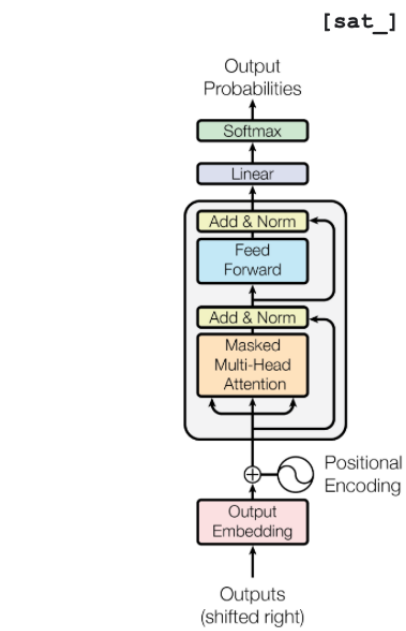
4



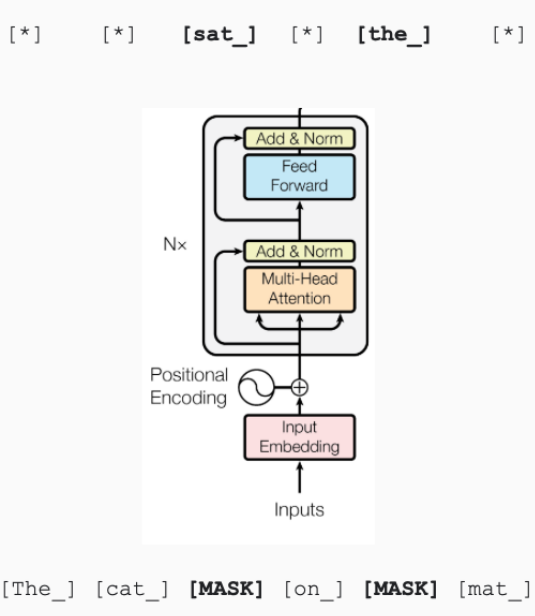
[4] https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#a-family-of-attention-mechanisms

Popular Transformer-based LMs

Decoder-only GPT



Encoder-only BERT



Enc-Dec T5

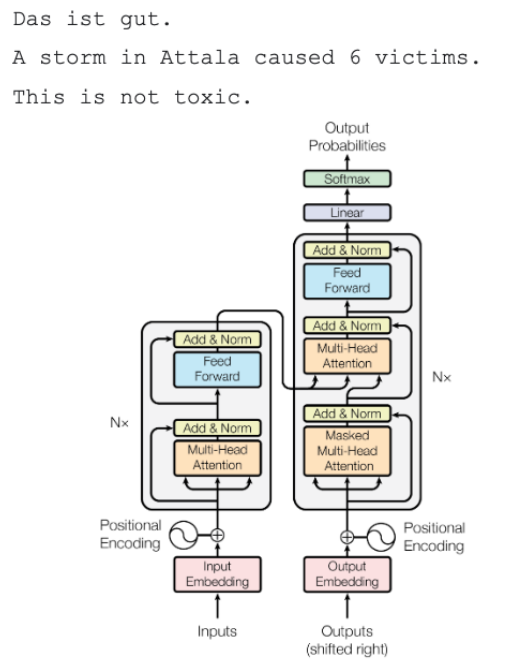


Figure: Source: Transformers tutorial, by Lucas Beyer (Google Brain)

Ethical Concerns with LMs

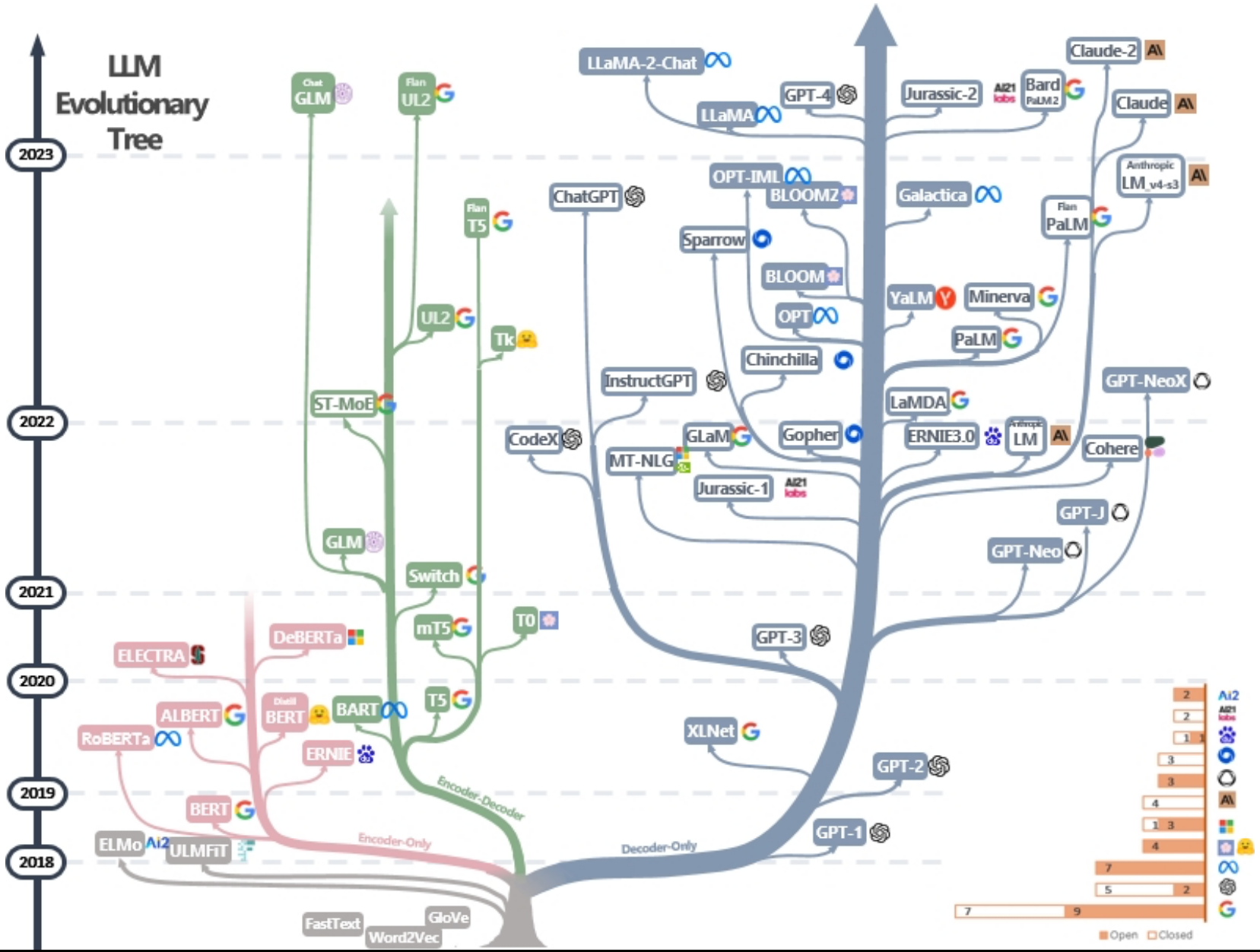
Huge language models can be extremely dangerous [7]:
They come at a **strong environmental cost**.
They are usually trained on data that is **biased**, encoding **racism** and **sexism** in them.
It's **hard to track** and interpret them due to their size. This type of large models have oppened whole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


LLM Evolutionary Tree
2023
2022
2021
2020
2019
2018
LLaMA-2-Chat
GPT-4
LLaMA
Jurassic-2
Bard
Claude-2
Claude
Anthropic LM v4-s3
OPT-IML
BLOOMZ
ChatGPT
Galactica
Flan PaLM
Sparrow
BLOOM
YaLM
Minerva
OPT
PaLM
Chinchilla
InstructGPT
LaMDA
GPT-NeoX
CodeX
GLaM
Gopher
ERNIE3.0
LM
Cohere
MT-NLG
Jurassic-1
GPT-J
GPT-Neo
GPT-3
Chat GLM
Flan UL2
Flan T5
UL2
Tk
ST-MoE
GLM
Switch
mT5
T0
DeBERTa
ELECTRA
ALBERT
Distil BERT
BART
T5
RoBERTa
ERNIE
XLNet
GPT-2
BERT
GPT-1
ELMo
ULMFiT
Encoder-Decoder
Encoder-Only
Decoder-Only
FastText
GloVe
Word2Vec
Open
Closed

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" control the sentiment of the generate text.
Context: previous era in which Transformers took over as a de-facto architecture in sequence models.

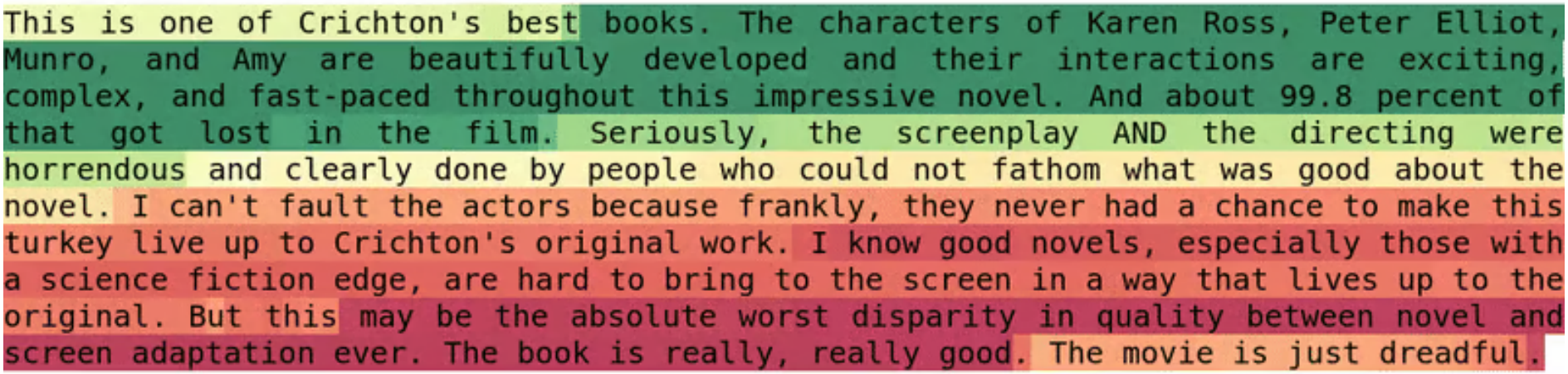


Figure: The sentiment neuron adjusting its value on a character-by-character basis. Source: Learning to generate reviews and discovering sentiment

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 game changer on scale training! Multi-task (explicit) adaptation recipe via formatting the input as a token sequence and reusing the pre-trained model.

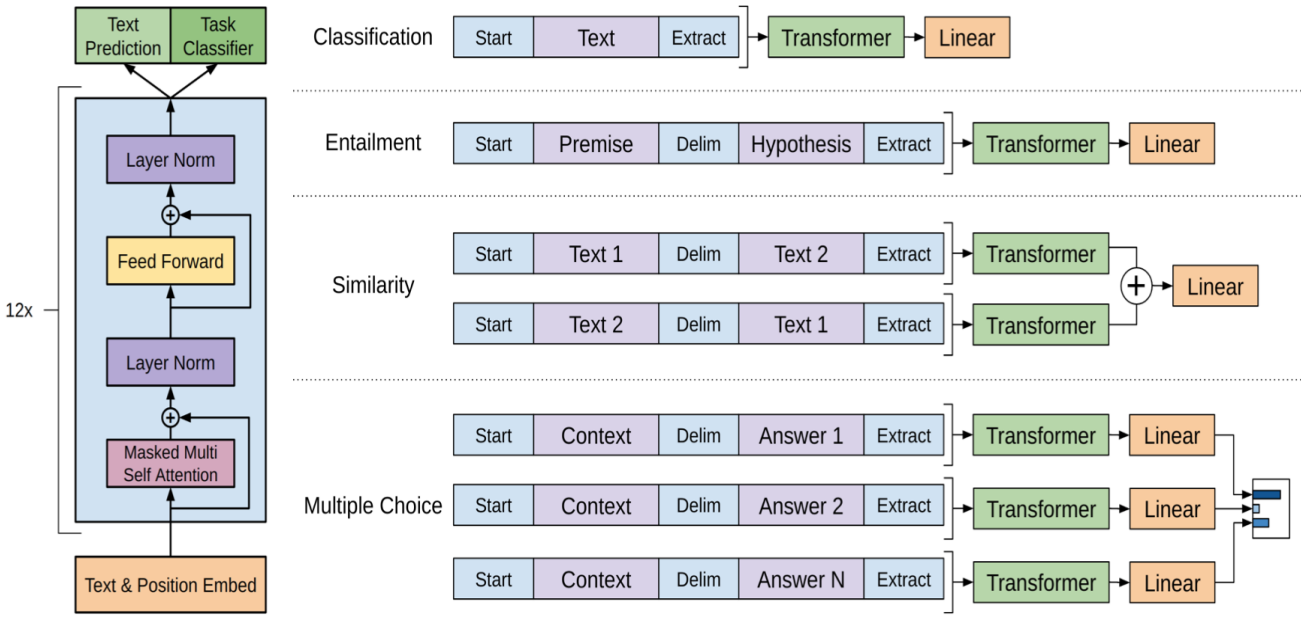


Figure: (left) Transformer decoder-only base model trained on the simple task of predicting the next token. (right) Input transformations into a token sequence to be processed by the base model, followed by a downstream layer (linear + softmax).
Source: Improving Language Understanding by Generative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

Figure: 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 to its parameters.

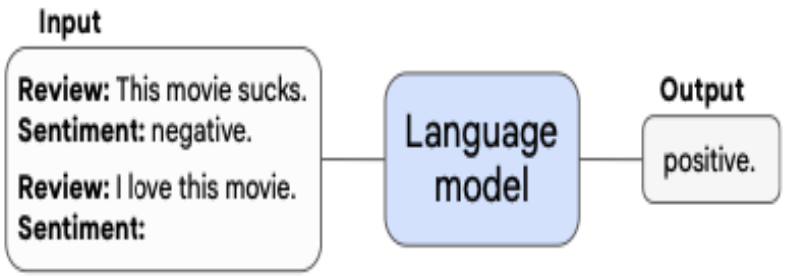


Figure: Emergent Abilities of Large Language Models (Source)

GPT-2: Scalability payoffs (2018)

Scalability proves as useful for powerful and general representations using as 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

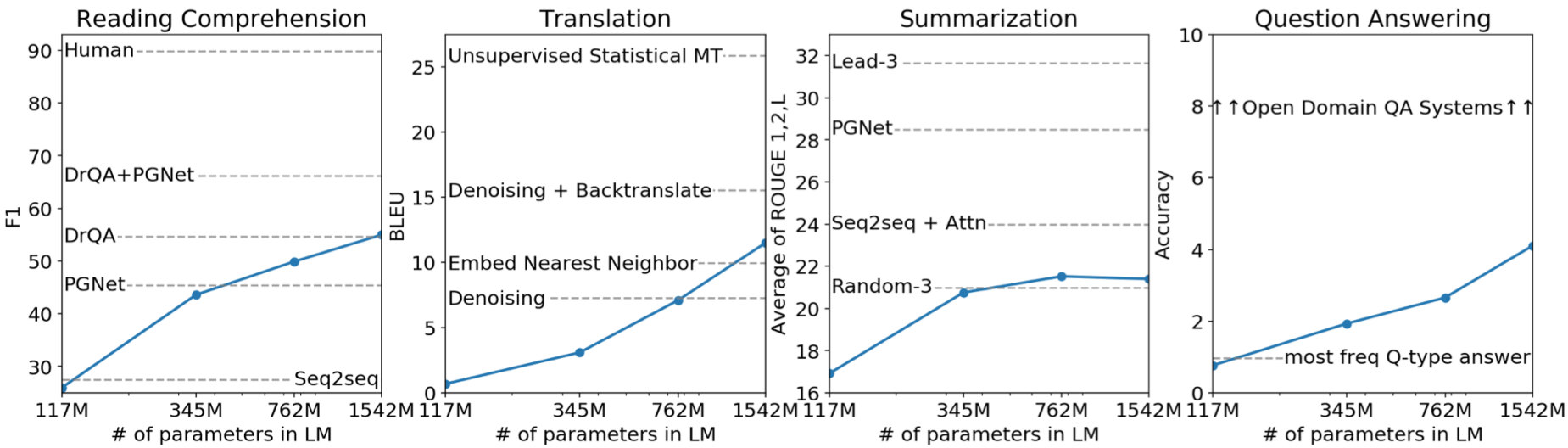
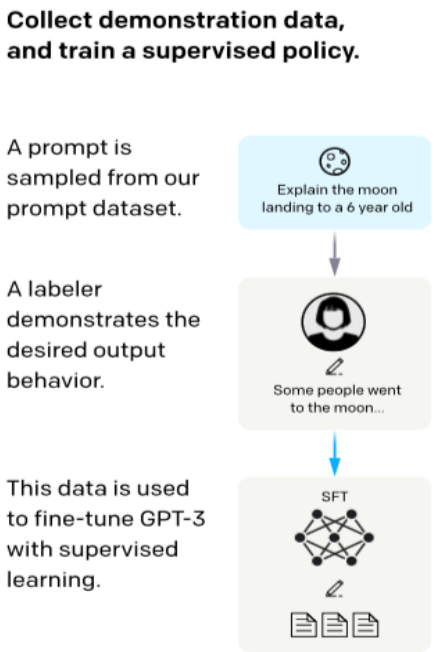


Figure: Zero-shot performance of a base model trained on WebText as a function of model size (117M-345M-762M-1542M parameters) in 4 different NLP tasks. Each task is evaluated on a benchmark dataset and compared with SOTA-specific models at that time, 2018-2019. Source: Language Models are Unsupervised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

ChatGPT: more than a LM (2022)

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

ChatGPT: Supervised Finetuning



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

Figure: source: Training language models to follow instructions with human feedback

ChatGPT: Reward Modeling

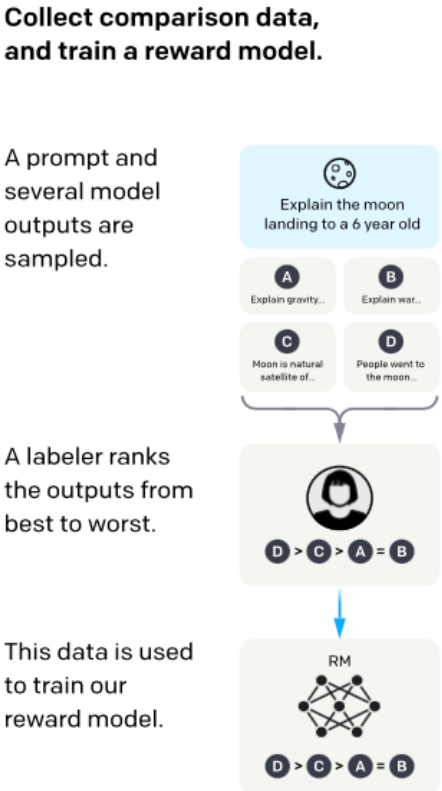


Figure: Training language models to follow instructions with human feedback

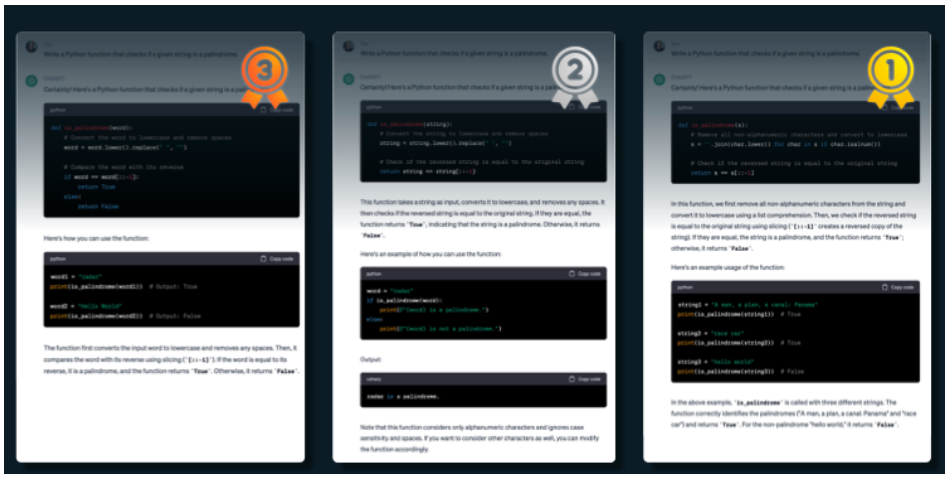


Figure: An example of a reward modeling dataset. Source: slide from the talk "the state of gpt", by andrej karpathy (2023).

ChatGPT: RL from Human Feedback

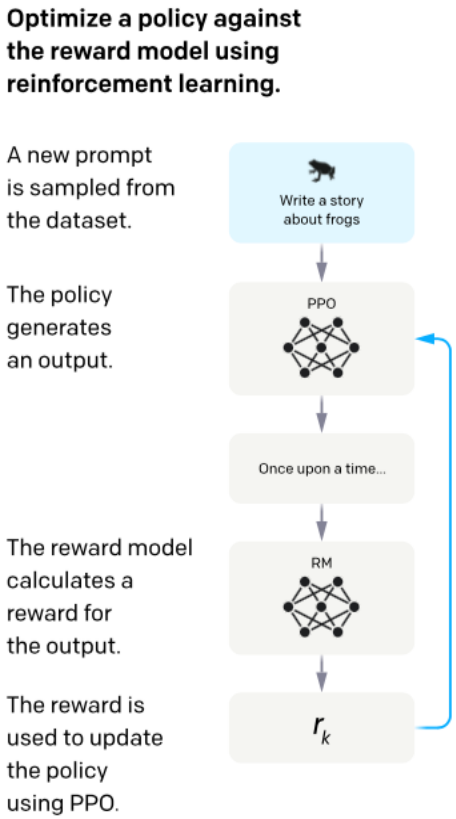
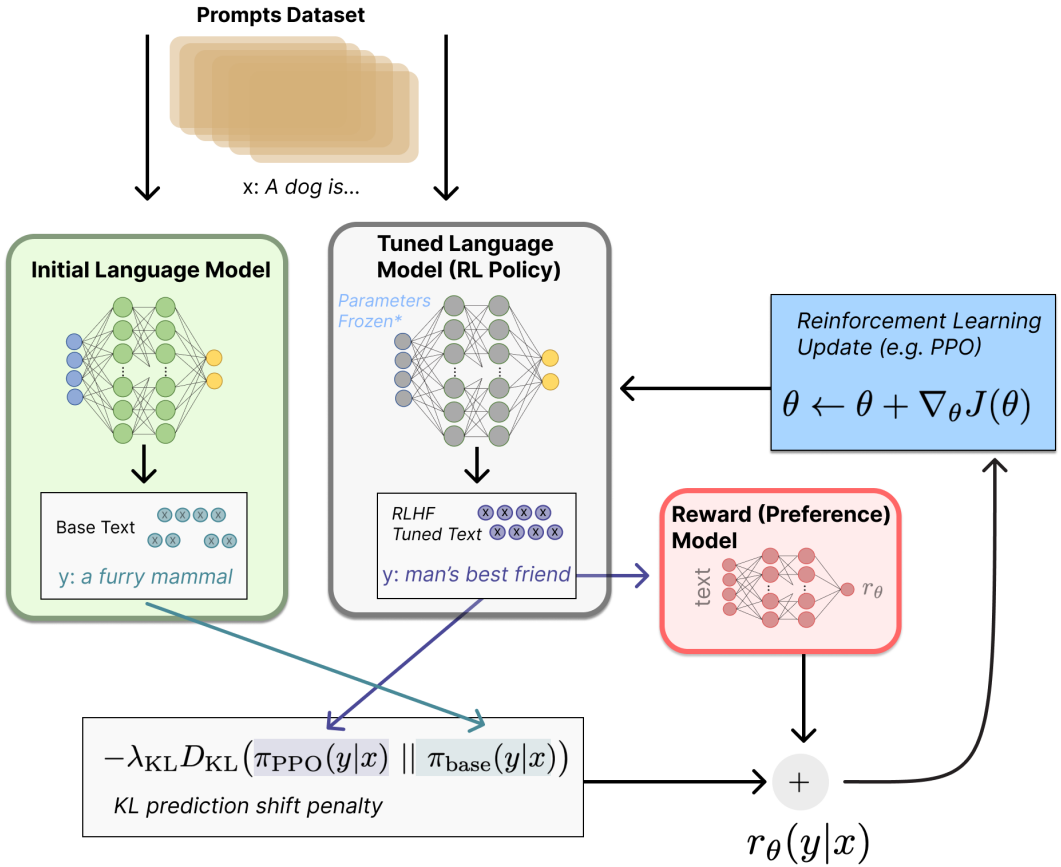


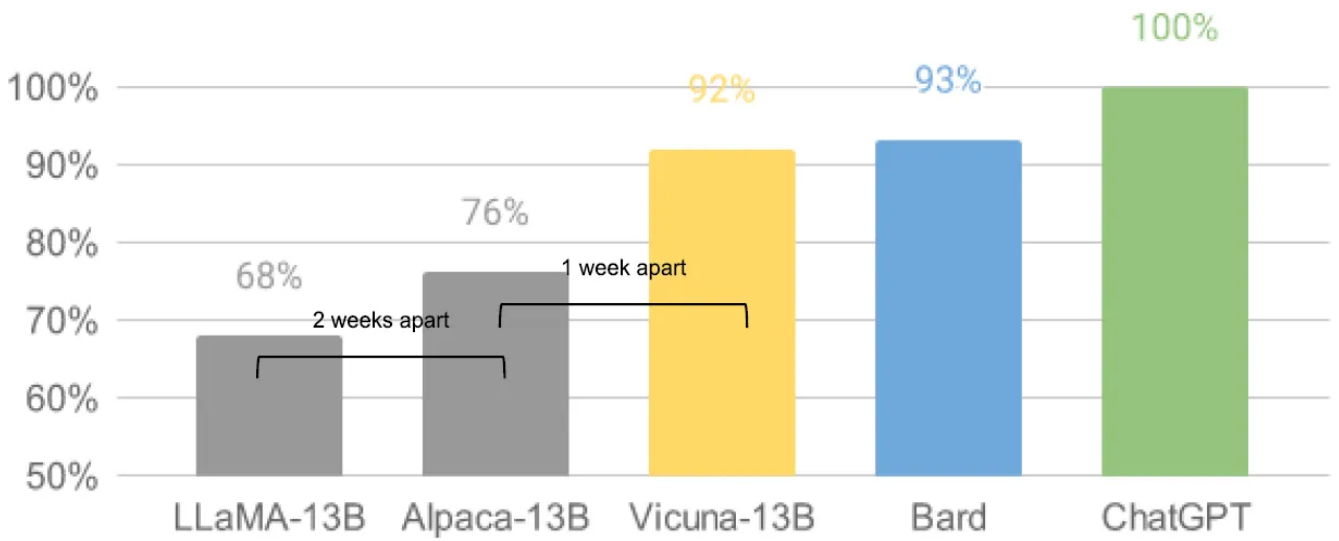
Figure: Training language models to follow instructions with human feedback

Figure: 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.

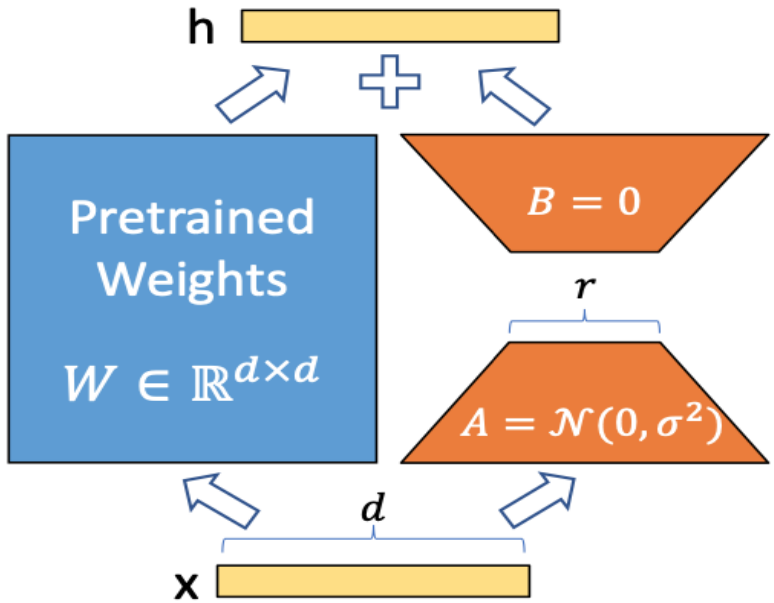


Figure: Freeze the pre-trained model weights and inject trainable rank decomposition matrices (A and B) into each layer of the Transformer architecture. Source: LoRA: Low-Rank Adaptation of Large Language Models

Table of contents

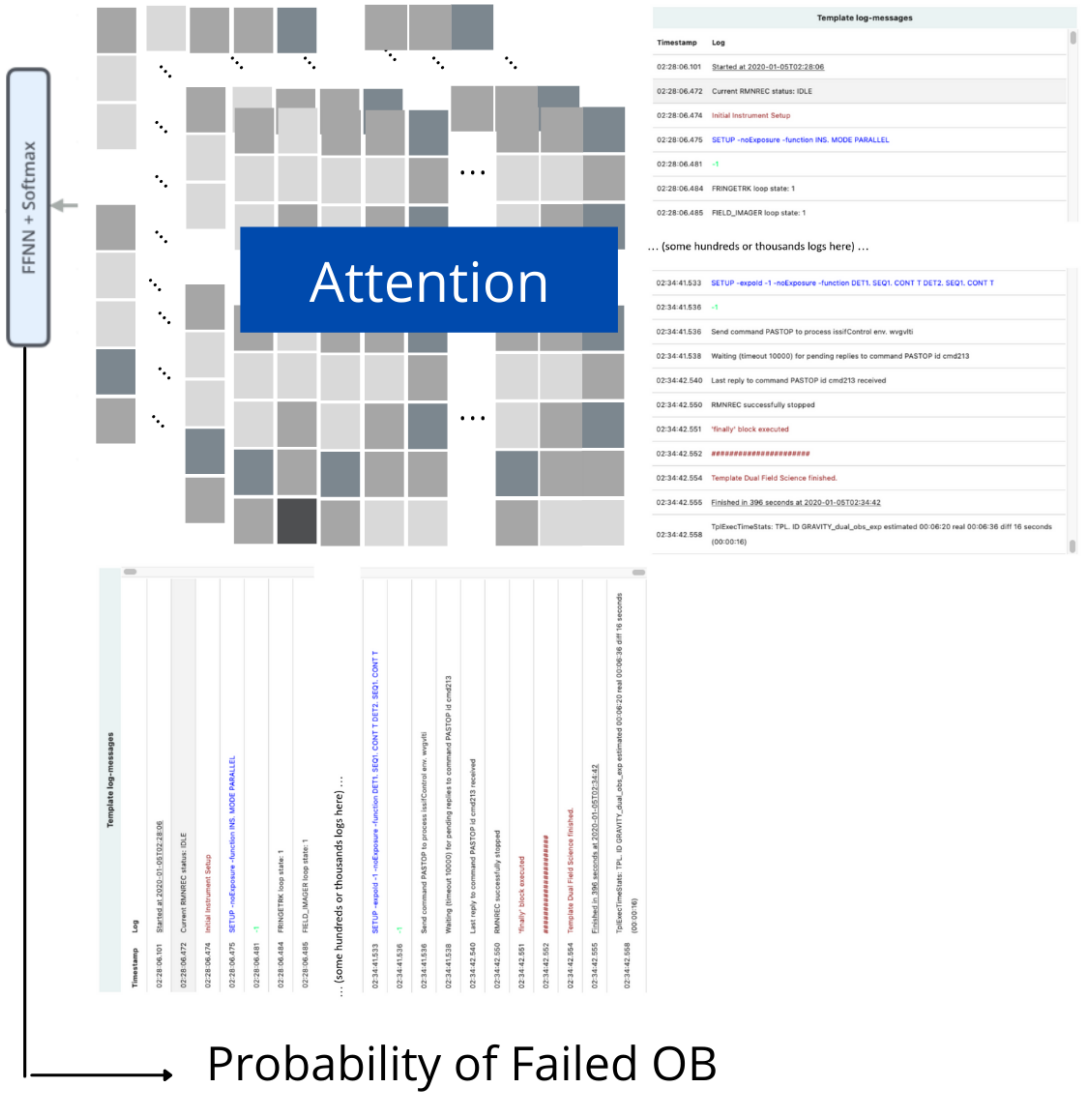
Troubleshooting with software logs

When a software related issue arises at some of the machines at Paranal, engineers spend hours trying to find its origin. The main source of information they have is **software logs**. They analyse the executions and check whether there is an abnormal behaviour. Since logs have textual information, we can try to apply some of the representation learning techniques for language. Some possible goals:

- To represent a given **template** in a way that a **probability of failed OB** can be computed at inference time
- To make it easier to **trace errors** back to the logs that might describe it

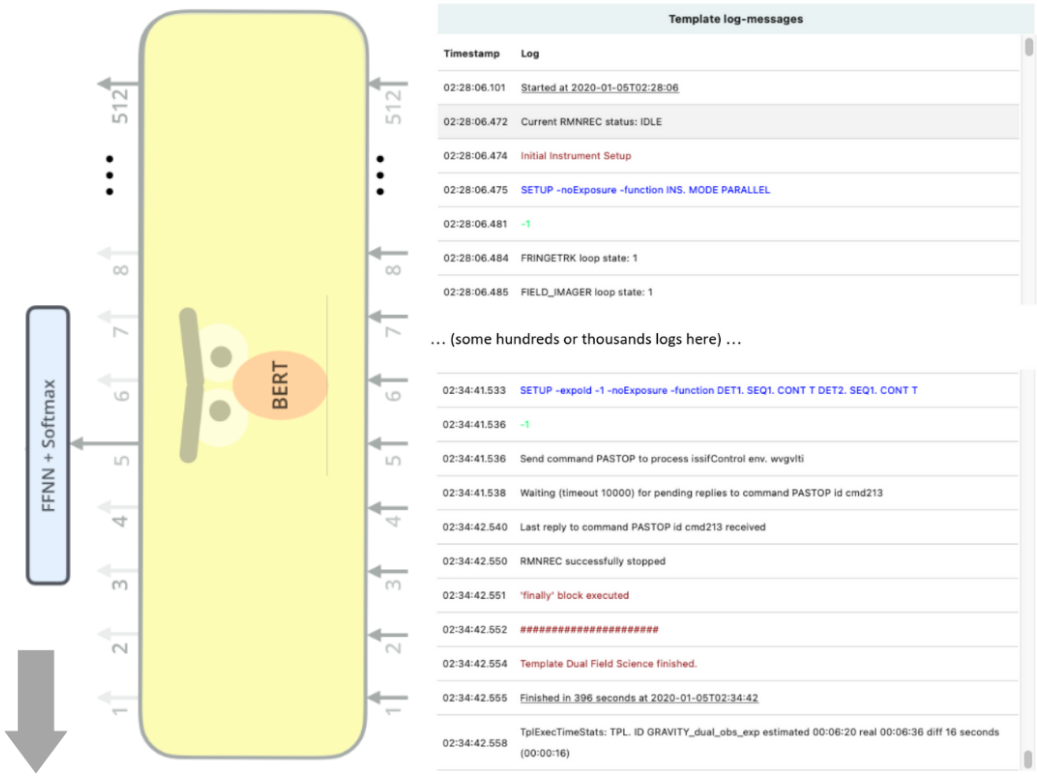
European Southern Observatory

Transformer based methods



European Southern Observatory

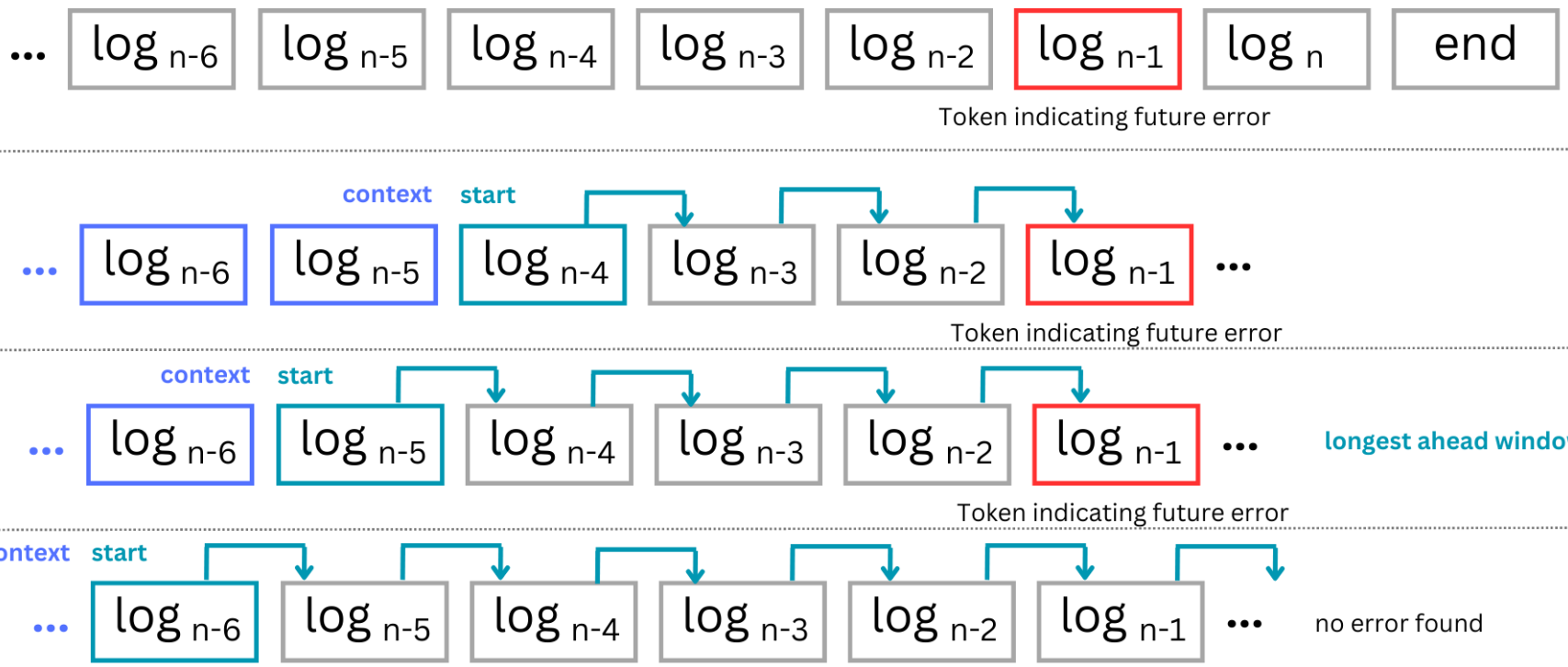
Transformer based methods



Probability of Failed OB

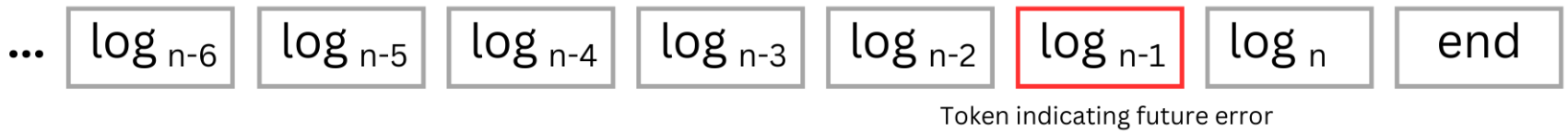
Log-generation methods

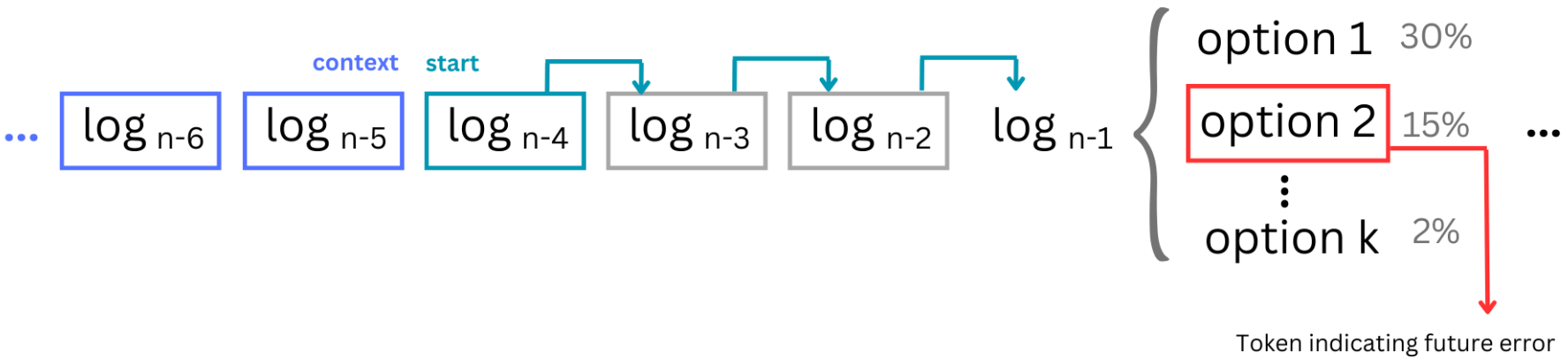
Error-log generation assessment Given a sequence of logs (tokens), one which is flagged as an error-ahead token, what is the largest amount of steps from which we can reconstruct the sequence in a way that such an error-ahead token can be predicted to occur



Log-generation methods

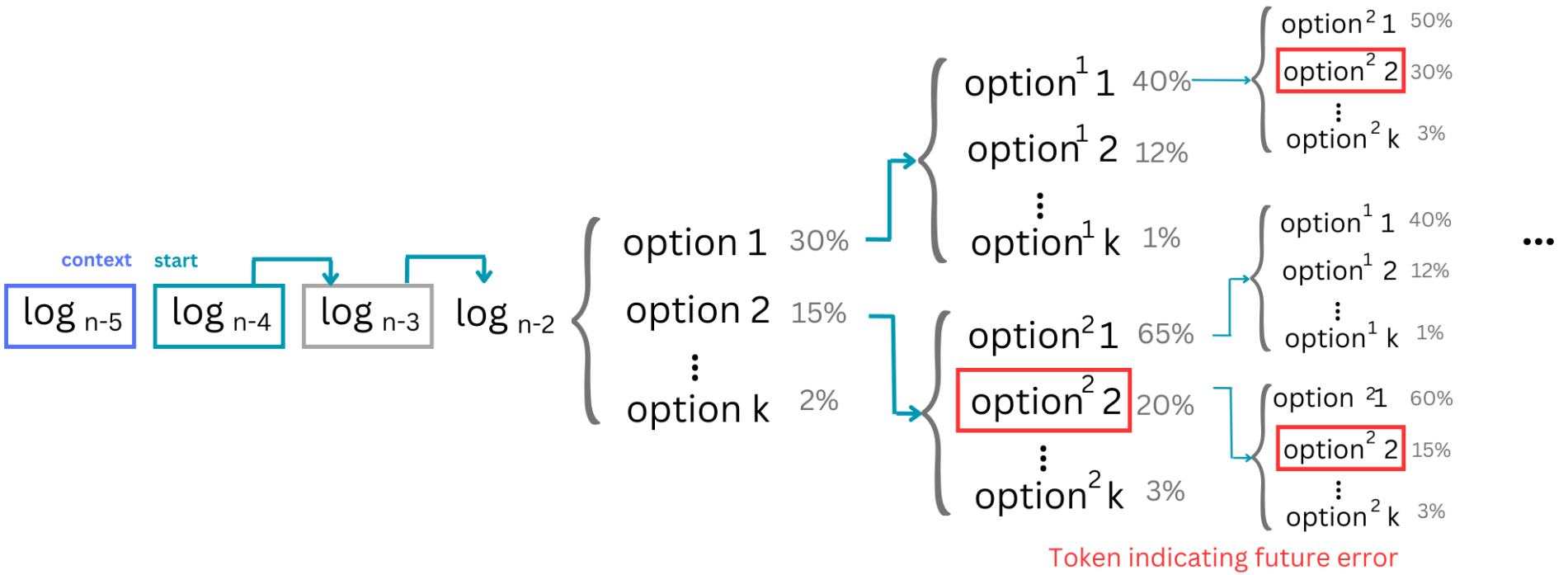
Probabilistic error-prone log generation assessment Given a sequence of logs, repeat the above experiment but consider the error-prone as predicted if the probability of it occurring is greater than a given threshold.





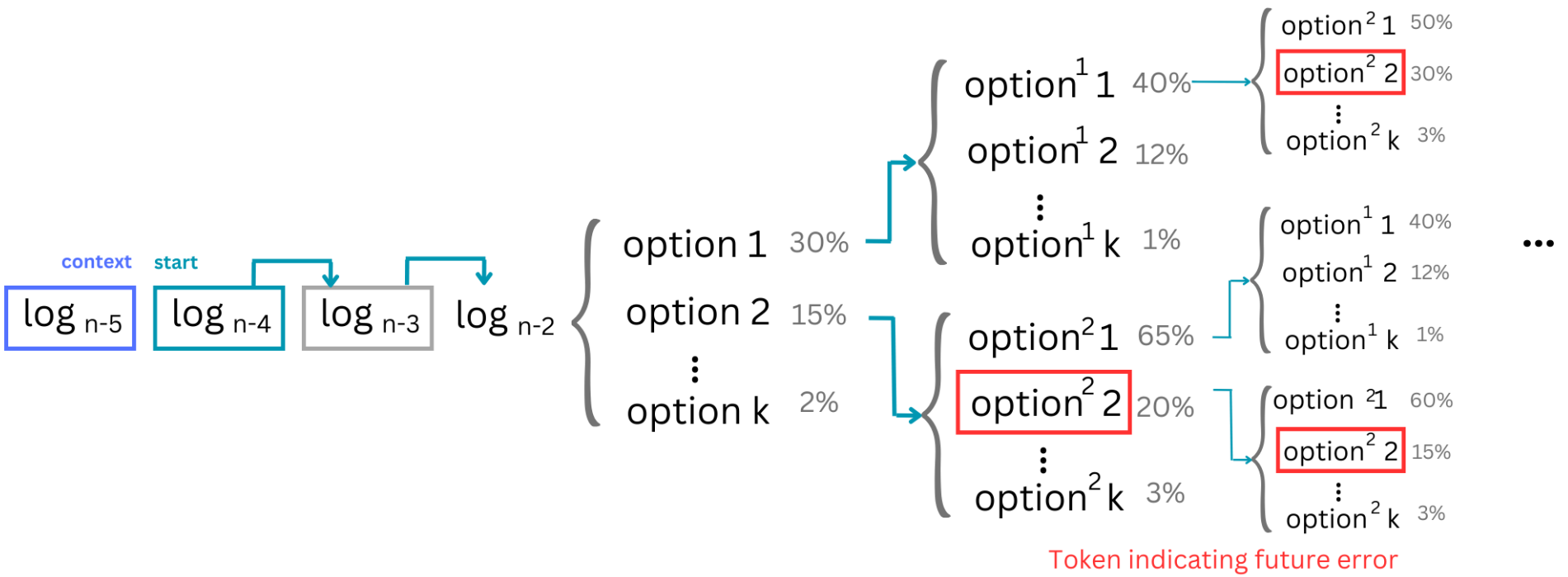
Log-generation methods

Greedy-tree error-prone log generation assessment A further alternative to checking if an error-prone token is likely to appear, we might check all the possibilities that might arise starting from a certain token and continuing such branches for a certain number of tokens.



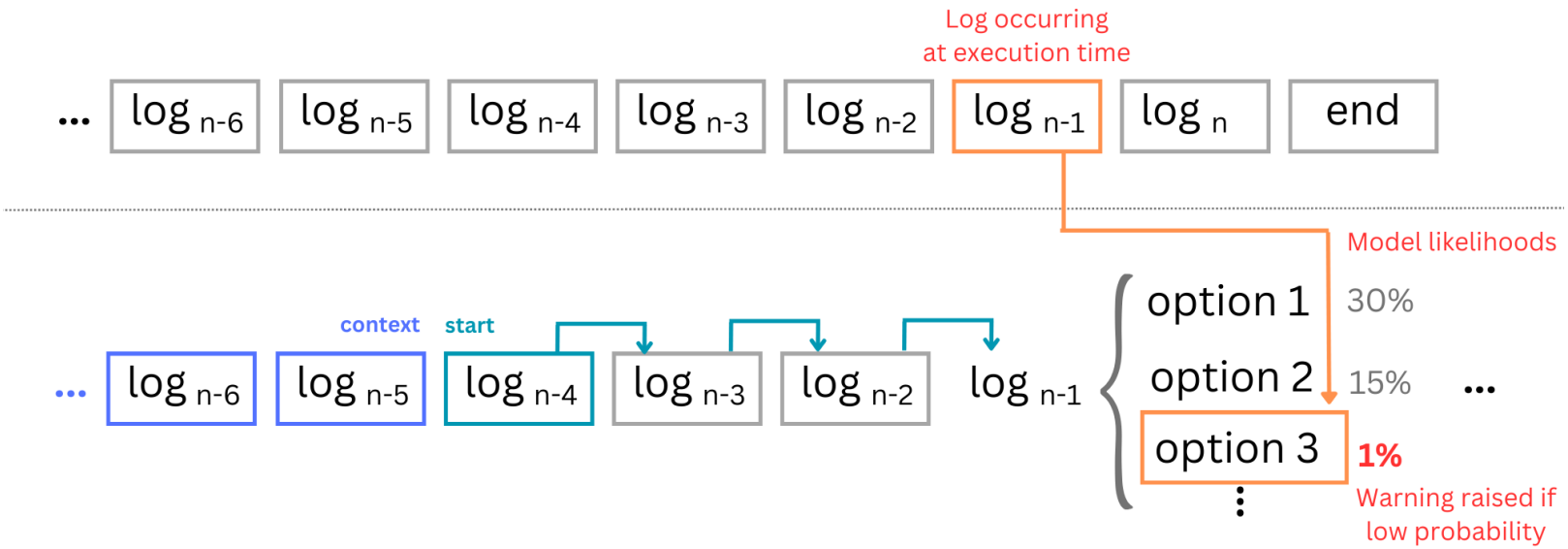
Log-generation methods

Greedy-tree error-prone log generation assessment Some options might be discarded if they fall below a certain probability threshold so the tree does not grow that large. In any case, this would be a greedy approach for spotting future error-prone tokens, which might lead to a considerable increase in complexity.



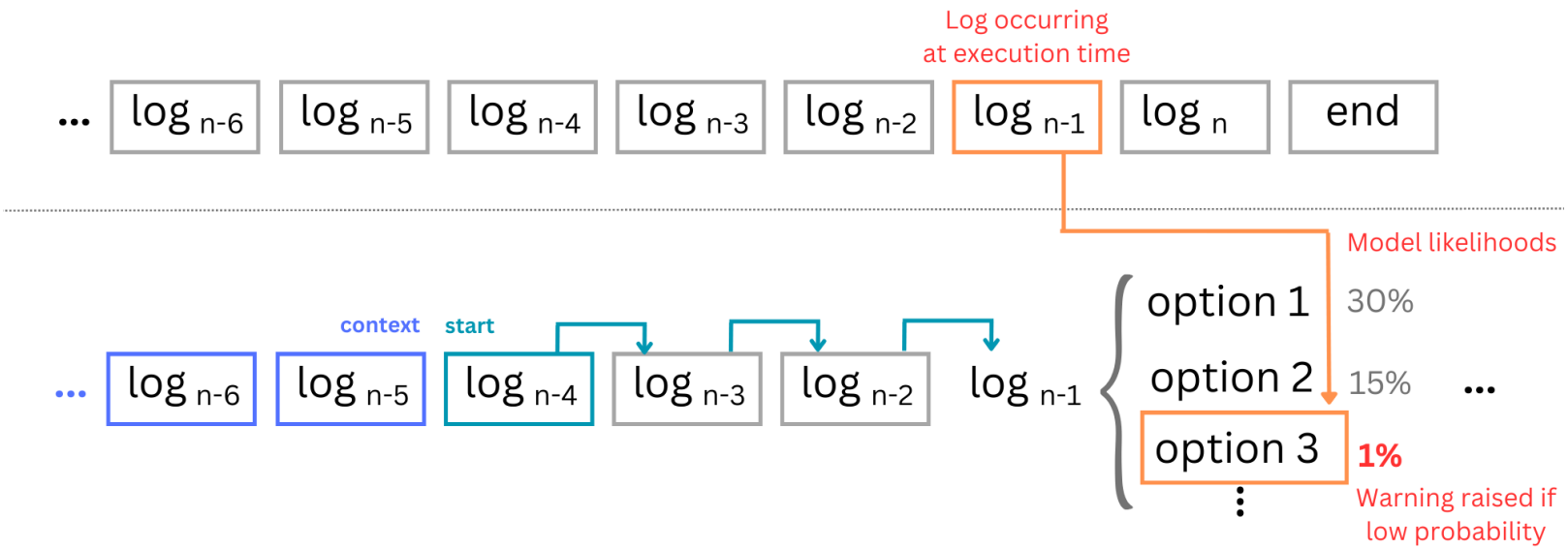
Execution sequence likelihood methods

Likelihood of occurring logs in a sequence Both with bidirectional and auto-regressive models we get an explicit probability of each token occurring in a given position. If we apply pre-training but solely on successful executions, we might check whether tokens in unsuccessful executions are more prone to have less probability than their normal counterparts.



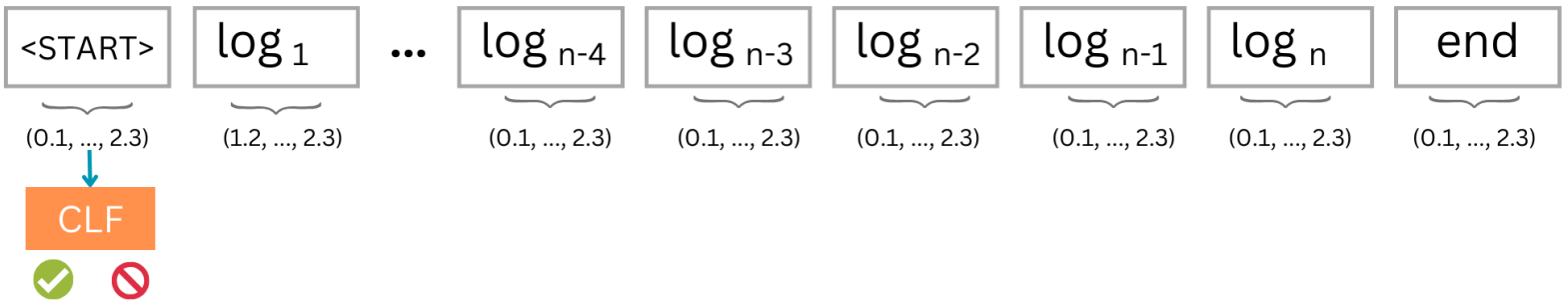
Execution sequence likelihood methods

Likelihood of occurring logs in a sequence If this is the case, we can set a threshold under which a certain log can be tagged as anomalous, thus raising a warning that the whole sequence might lead to an error.



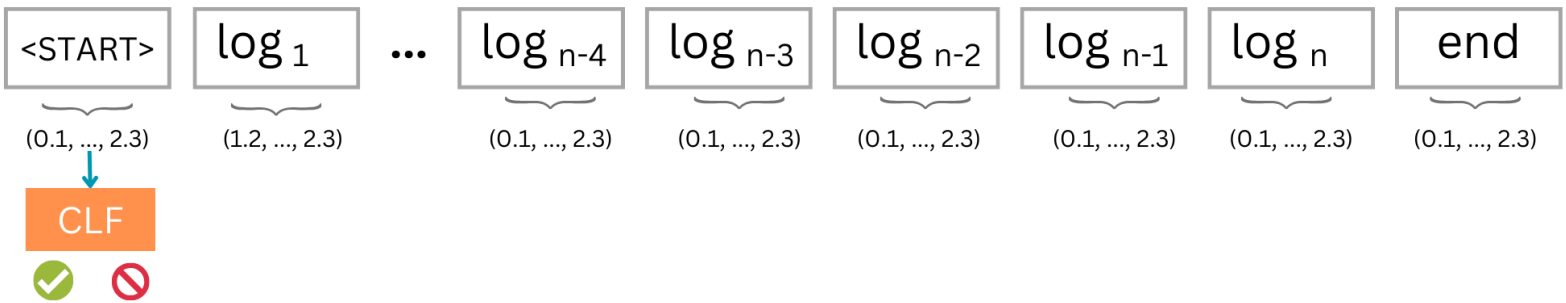
Classification using token embeddings

Sequence classification using <START> special token Language models often use special tokens for denting different aspects of speech. The most common ones are starting tokens, which might be denoted as <s>, [START] or [CLF].



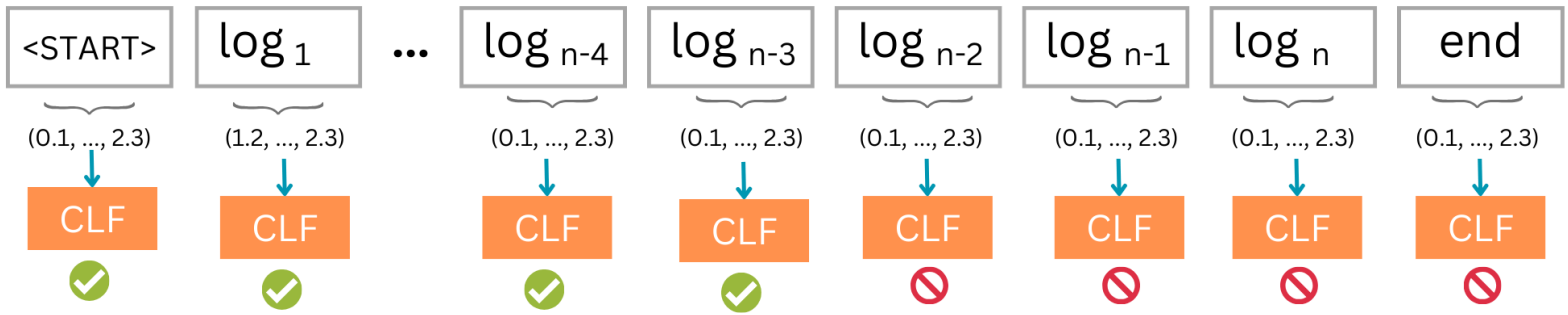
Classification using token embeddings

Sequence classification using <START> special token We might consider a starting token as the representation of the whole sequence through its embedding. Like the rest of the tokens, normal and special ones, there is a contextualised vector associated with it. Hence, as the sequence changes, the vector for the starting special token changes and this allows for fitting classifiers taking solely the starting token as input.



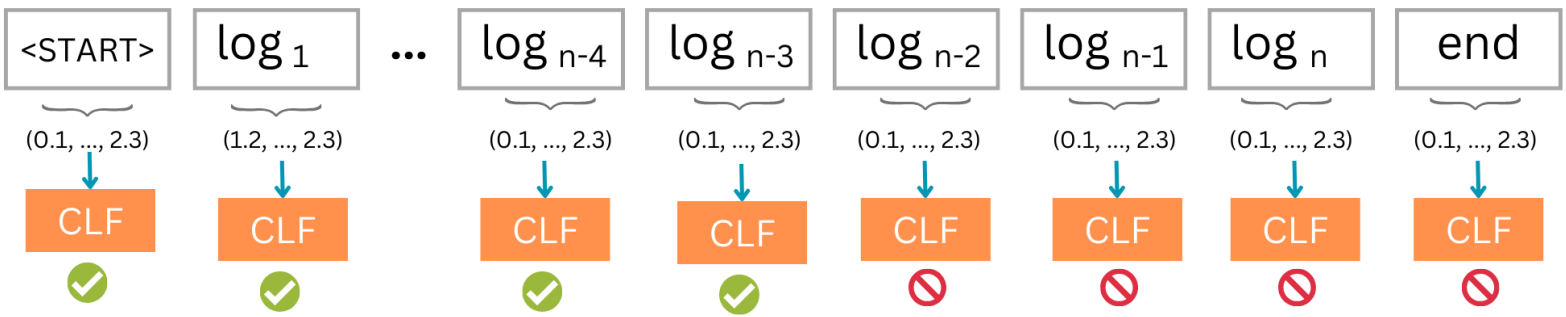
Classification using token embeddingss

Token level classification An alternative to the above is flagging tokens occurring in sequences that have not terminated successfully as error-co-occurring. Once this is done, we shall classify whether tokens in a new sequence get flagged in such a group. This is a way of determining in advance if a sequence will end up in an error but has the drawback of flagging sub-sequences as error-likely even though they might be okay (the event producing the error might show up at any point during the sequence, potentially towards the end).



Classification using token embeddingss

Token level classification We expect this effect will be alleviated thanks to normal executions being flagged as non-problematic, thus pushing the probability back to normal to some extent.



Experiments during SM2022 Internship

We iterated over the dataset with six months of data (200 epochs for each model). We defined a dataloader class that is specially designed to work with the parlogan library.

At each time step, we would **mask 15% of tokens** and try to predict them back with the transformer language model. We used an Azure Standard NC24 machine with 4 NVIDIA Tesla K80 GPUs.

The technical setting:

- Two transformer-encoder layers
- Input and hidden dimensionality: 200
- Deep-learning framework: PyTorch[5]
- Azure - Machine learning studio
- Masking tokens learning objective with cross-entropy loss

[5] https://pytorch.org/

Experiments during SM2022 Internship

We iterated over the dataset with six months of data (200 epochs for each model). We defined a dataloader class that is specially designed to work with the parlogan library.

At each time step, we would **mask 15% of tokens** and try to predict them back with the transformer language model. We used an Azure Standard NC24 machine with 4 NVIDIA Tesla K80 GPUs.

instrument	batch size	vocab size	max. length	training time
Gravity	8	2726	4000	35 h 33 m
Matisse	16	2790	1000	3 h 36 m
Pionier	16	901	500	2 h 10 m

European Southern Observatory

Prediction with Transformers

We want to test the prediction capacity of the model. Hence we ask it to predict ahead tokens starting from a given point in the execution.
It is worth noting the model wasn't exactly trained for this (it could), we rather did the following process during training:
The sample on the left shows the input and on the right, it shows the output

['Started at (underlined)',
'PIONIER$_o bs_c alibrator$ —
—$Observationofcalibrator(yellow)'$,
"DPR TYPE = 'FRINGE,OBJECT'",
'[MASK]',
"DPR CATG = 'CALIB'",
"SEQ NEXPO = ''",
'[MASK]',
'STATUS -function DET.READOUT.MODE (blue)',
"Send command 'STATUS'
',DET.READOUT.MODE,FALSE' to sub-system 'DCS'
...",
"Last Reply to 'STATUS' from 'DCS' received:
'DET.READOUT.MODE ARG' (len='')"]

['Started at (underlined)',
'PIONIER$_o bs_c alibrator$ —
—$Observationofcalibrator(yellow)'$,
"DPR TYPE = 'FRINGE,OBJECT'",
"SEQ NEXPO = ''",
"DPR CATG = 'CALIB'",
"SEQ NEXPO = ''",
"DPR TECH = 'INTERFEROMETRY'",
'STATUS -function DET.READOUT.MODE (blue)',
"Send command 'STATUS'
',DET.READOUT.MODE,FALSE' to sub-system 'DCS'
...",
"Last Reply to 'STATUS' from 'DCS' received:
'DET.READOUT.MODE ARG' (len='')"]

Prediction with Transformers

We want to test the prediction capacity of the model. Hence we ask it to predict ahead tokens starting from a given point in the execution.
Now we take an execution except for its last 3 tokens. We predict tokens ahead one by one:
The sample on the left shows the input and on the right it shows the output

[' Exposure status: FINISHED (SpringGreen4)',
'ended exposure 5 of 5 (2021-01-01T05:51:33) (underlined)',
'New image: PIONIER$_O$*BS*$_F$*RINGE*001$_0$015*.fits* −
−*from.Exposeof* :: *pnoseqOBS* :: *BobTPL* :: *obs*(*Blue*)′,
'WAIT -expoId -all -cond ObsEnd (blue)',
'INACTIVE (SpringGreen4)']
'[MASK]',

[' Exposure status: FINISHED (SpringGreen4)',
'ended exposure 5 of 5 (2021-01-01T05:51:33) (underlined)',
'New image: PIONIER$_O$*BS*$_F$*RINGE*001$_0$015*.fits* −
−*from.Exposeof* :: *pnoseqOBS* :: *BobTPL* :: *obs*(*Blue*)′,
'WAIT -expoId -all -cond ObsEnd (blue)',
'INACTIVE (SpringGreen4)']
'Template PIONIER$_o$*bs*$_c$*alibratorfinished*.′,

Prediction with Transformers

We want to test the prediction capacity of the model. Hence we ask it to predict ahead tokens starting from a given point in the execution.
Now we take an execution except for its last 3 tokens. We predict tokens ahead one by one:
The sample on the left shows the input and on the right it shows the output

[' Exposure status: FINISHED (SpringGreen4)',
'ended exposure 5 of 5 (2021-01-01T05:51:33) (underlined)',
'New image: PIONIER$_O$*BS*$_F$*RINGE*001$_0$015.*fits* — —*from.Exposeof* :: *pnoseqOBS* :: *BobTPL* :: *obs*(*Blue*)′,
'WAIT -expoId -all -cond ObsEnd (blue)',
'INACTIVE (SpringGreen4)']
'Template PIONIER$_o$*bs*$_c$*alibratorfinished.*′,
'[MASK]',

[' Exposure status: FINISHED (SpringGreen4)',
'ended exposure 5 of 5 (2021-01-01T05:51:33) (underlined)',
'New image: PIONIER$_O$*BS*$_F$*RINGE*001$_0$015.*fits* — —*from.Exposeof* :: *pnoseqOBS* :: *BobTPL* :: *obs*(*Blue*)′,
'WAIT -expoId -all -cond ObsEnd (blue)',
'INACTIVE (SpringGreen4)']
'Template PIONIER$_o$*bs*$_c$*alibratorfinished.*′,
'Finished in seconds at (underlined)',

European Southern Observatory

Prediction with Transformers

We want to test the prediction capacity of the model. Hence we ask it to predict ahead tokens starting from a given point in the execution.
Now we take an execution except for its last 3 tokens. We predict tokens ahead one by one:
The sample on the left shows the input and on the right it shows the output

[' Exposure status: FINISHED (SpringGreen4)',
'ended exposure 5 of 5 (2021-01-01T05:51:33) (underlined)',
'New image: PIONIER$_O$*BS*$_F$*RINGE*001$_0$015*.fits* — —*from.Exposeof* :: *pnoseqOBS* :: *BobTPL* :: *obs*(*Blue*)′,
'WAIT -expoId -all -cond ObsEnd (blue)',
'INACTIVE (SpringGreen4)']
'Template PIONIER$_o$*bs*$_c$*alibratorfinished.*′,
'Finished in seconds at (underlined)',
'[MASK]',

[' Exposure status: FINISHED (SpringGreen4)',
'ended exposure 5 of 5 (2021-01-01T05:51:33) (underlined)',
'New image: PIONIER$_O$*BS*$_F$*RINGE*001$_0$015*.fits* — —*from.Exposeof* :: *pnoseqOBS* :: *BobTPL* :: *obs*(*Blue*)′,
'WAIT -expoId -all -cond ObsEnd (blue)',
'INACTIVE (SpringGreen4)']
'Template PIONIER$_o$*bs*$_c$*alibratorfinished.*′,
'Finished in seconds at (underlined)',
'TplExecTimeStats: TPL.ID PIONIER$_o$*bs*$_c$*alibrator* estimated :: real :: diff seconds (::)',

In this case, the model predicts the end of the executions some logs before it is announced

Prediction with Transformers

We developed an algorithm that detects whether a certain target error log will happen in advance:

- Starts some steps ahead of the target log
- It generates tokens step by step up to some point (roughly the position in which the target log is expected to occur)
- It checks whether the target token has been generated by the model
- If the token was predicted then it tries the same but one step further

This gives us an idea of how many steps we need, as well as how much context is enough to predict a certain event.

References I

[1] Mikolov, T., Yih, W., Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 746–751. https://www.aclweb.org/anthology/N13-1090

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

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

[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

References II

[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

[8] 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

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

[10] 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

[11] 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

[12] 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

[13] 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.pdf

European Southern Observatory

Sequence Models with Transformers

A Journey Through Language Models, Current Insights, and Practical Applications at Paranal Observatory

Camilo Carvajal Reyes - Cristóbal Alcázar

Master of Data Science, U. de Chile

10th September 2023

Credit for cover image: ESO/M. Kornmesser