Module 5

Natural Language Processing with RNNs and Attention

Introduction

Can we build a machine that can master written and spoken language?

This is the ultimate goal of NLP research

Generating Shakespearean Text Using a Character RNN

Here is a small sample of the text generated by a char-RNN model after it was trained on all of Shakespeare's works:

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Not exactly a masterpiece, but it is still impressive that the model was able to learn words, grammar, proper punctuation, and more, just by learning to predict the next character in a sentence.

Sentiment Analysis

One of the most common applications of NLP is text classification - especially sentiment analysis.

If image classification on the MNIST dataset is the “Hello world!” of computer vision, then sentiment analysis on the IMDb reviews dataset is the “Hello world!” of natural language processing.

The IMDb dataset consists of 50,000 movie reviews in English (25,000 for training, 25,000 for testing) extracted from the famous Internet Movie Database, along with a simple binary target for each review indicating whether it is negative (0) or positive (1).

Just like MNIST, the IMDb reviews dataset is popular for good reasons:

- it is simple enough to be tackled on a laptop in a reasonable amount of time, but challenging enough to be fun and rewarding.

Word Embeddings

https://medium.com/@hsinhungw/understanding-word-embeddings-with-keras-dfafde0d15a4

Reusing Pretrained Embeddings and Language Models

It is impressive that the model is able to learn useful word embeddings based on just 25,000 movie reviews.

Imagine how good the embeddings would be if we had billions of reviews to train on!

We can reuse word embeddings trained on some other (very) large text corpus (e.g., Amazon reviews, available on TensorFlow Datasets), even if it is not composed of movie reviews? The word “amazing” generally has the same meaning whether you use it to talk about movies or anything else.

So, instead of training word embeddings, we could just download and use pretrained embeddings, such as Google’s Word2vec embeddings, Stanford’s GloVe embeddings, or Facebook’s FastText embeddings.

Reusing Pretrained Embeddings and Language Models

Using pretrained word embeddings was popular for several years, but this approach has its limits.

In particular, a word has a single representation, no matter the context. For example, the word “right” is encoded the same way in “left and right” and “right and wrong”, even though it means two very different things.

To address this limitation, in 2018 Matthew Peters introduced *Embeddings from Language Models* (ELMo): these are contextualized word embeddings. Instead of just using pretrained embeddings in your model, you reuse part of a pretrained language model.

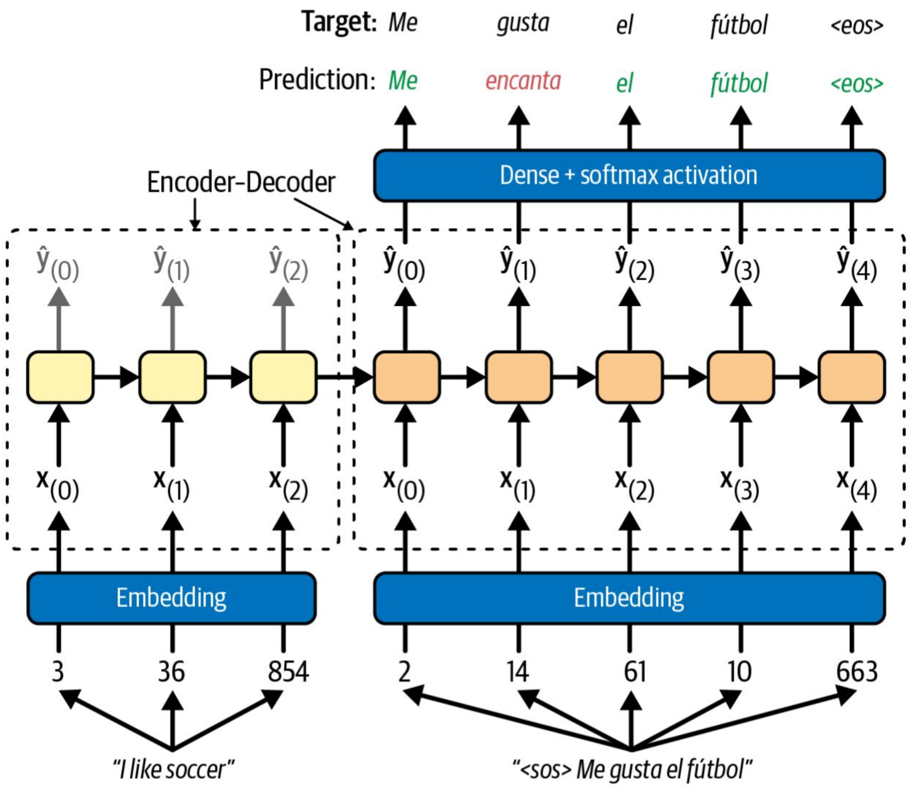
Pretrained model fine-tuned on just 100 labeled examples could achieve the same performance as one trained from scratch on 10,000 examples.

An Encode-Decoder Network for Neural Machine Translation

Let's translate English sentences to Spanish.

Architecture: English sentences are fed as inputs to the encoder, and the decoder outputs the Spanish translations.

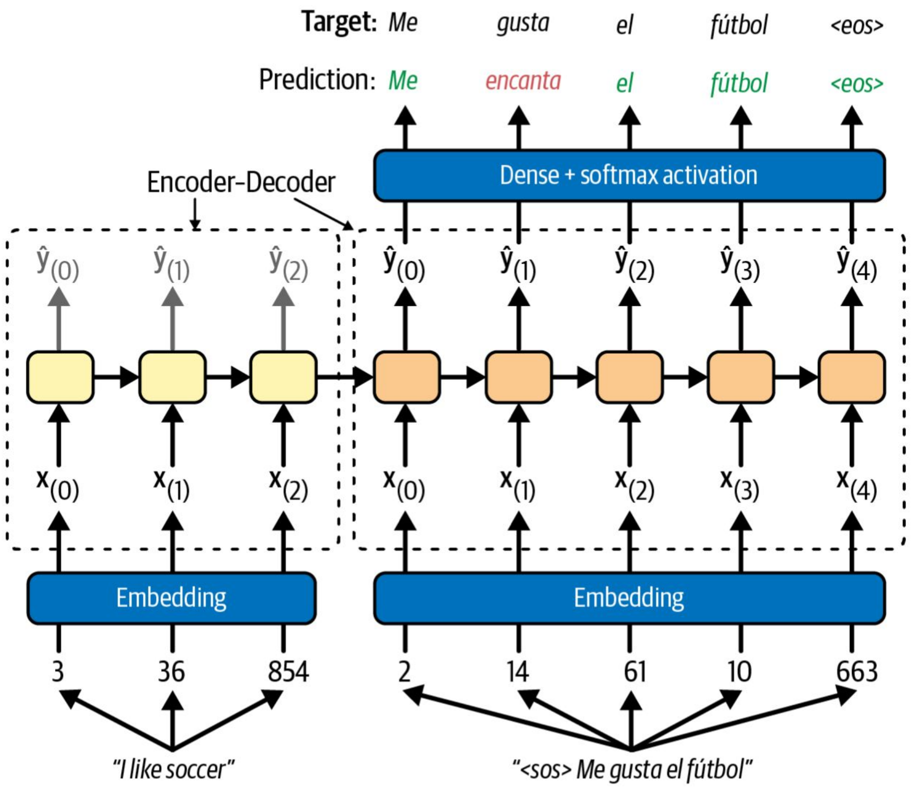
Spanish translations are also used as inputs to the decoder during training, but shifted back by one step. In other words, during training the decoder is given as input the word that it *should* have output at the previous step.



An Encode-Decoder Network for Neural Machine Translation

For the very first word, the decoder is given the start-of-sequence (SOS) token, and the decoder is expected to end the sentence with an end-of-sequence (EOS) token.

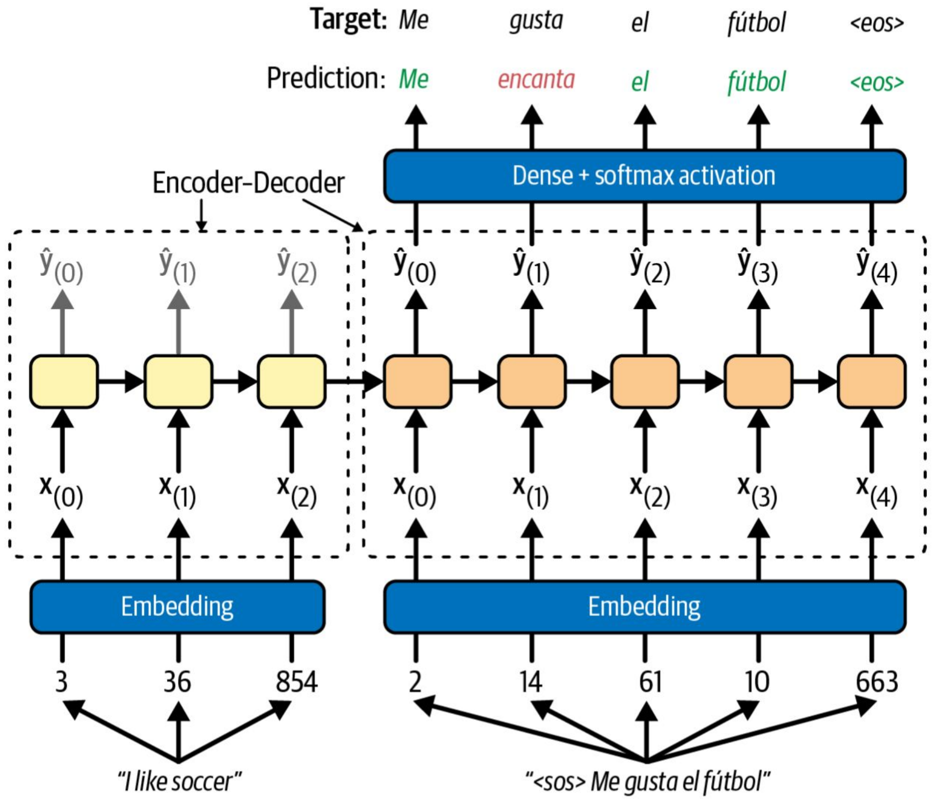
Each word is initially represented by its ID (e.g., 854 for the word “soccer”). Next, an Embedding layer returns the word embedding.



An Encode-Decoder Network for Neural Machine Translation

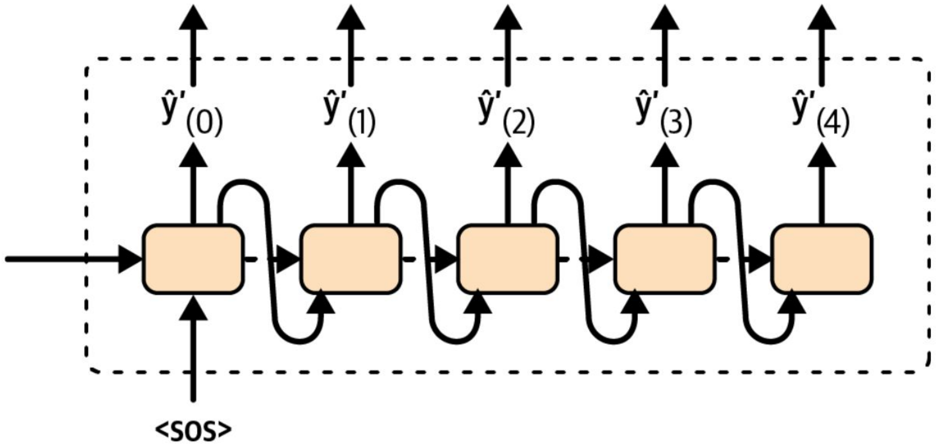
At each step, the decoder outputs a score for each word in the output vocabulary (i.e., Spanish), then the softmax activation function turns these scores into probabilities.

For example, at the first step the word “Me” may have a probability of 7%, and so on. The word with the highest probability is output. This is very much like a regular classification task.


Target: Me gusta el fútbol <eos>
Prediction: Me encanta el fútbol <eos>
Dense + softmax activation
Encoder-Decoder
$\hat{y}_{(0)}$ $\hat{y}_{(1)}$ $\hat{y}_{(2)}$
$\hat{y}_{(0)}$ $\hat{y}_{(1)}$ $\hat{y}_{(2)}$ $\hat{y}_{(3)}$ $\hat{y}_{(4)}$
$x_{(0)}$ $x_{(1)}$ $x_{(2)}$
$x_{(0)}$ $x_{(1)}$ $x_{(2)}$ $x_{(3)}$ $x_{(4)}$
Embedding
Embedding
3 36 854
2 14 61 10 663
"I like soccer"
"<sos> Me gusta el fútbol"

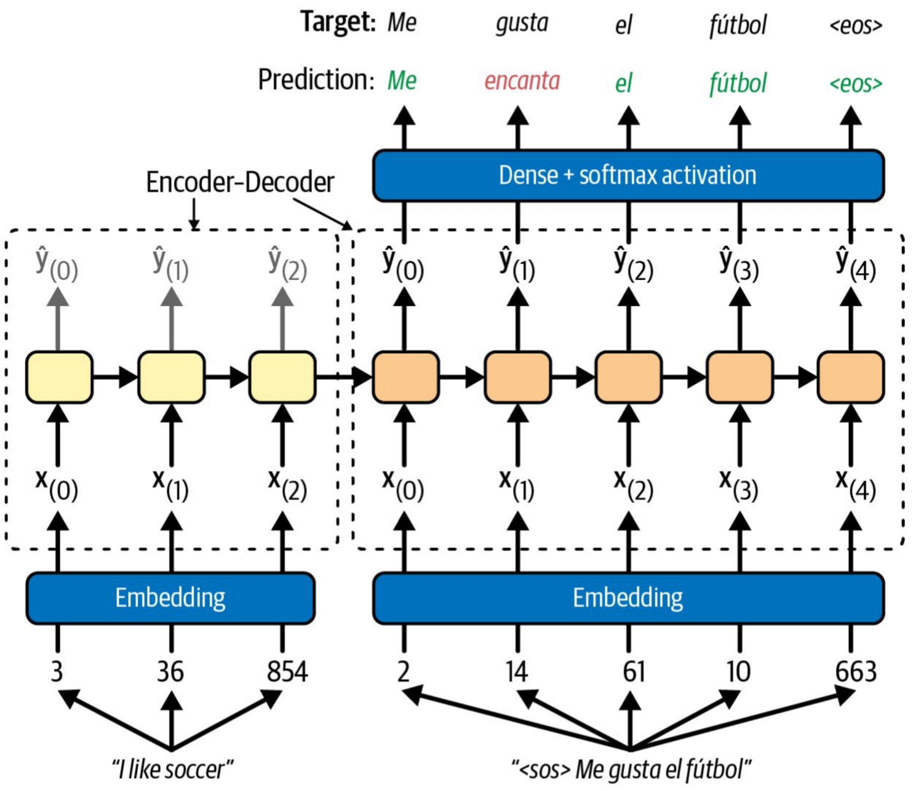
An Encode-Decoder Network for Neural Machine Translation

Note that at inference time (after training), you will not have the target sentence to feed to the decoder. Instead, you need to feed it the word that it has just output at the previous step.



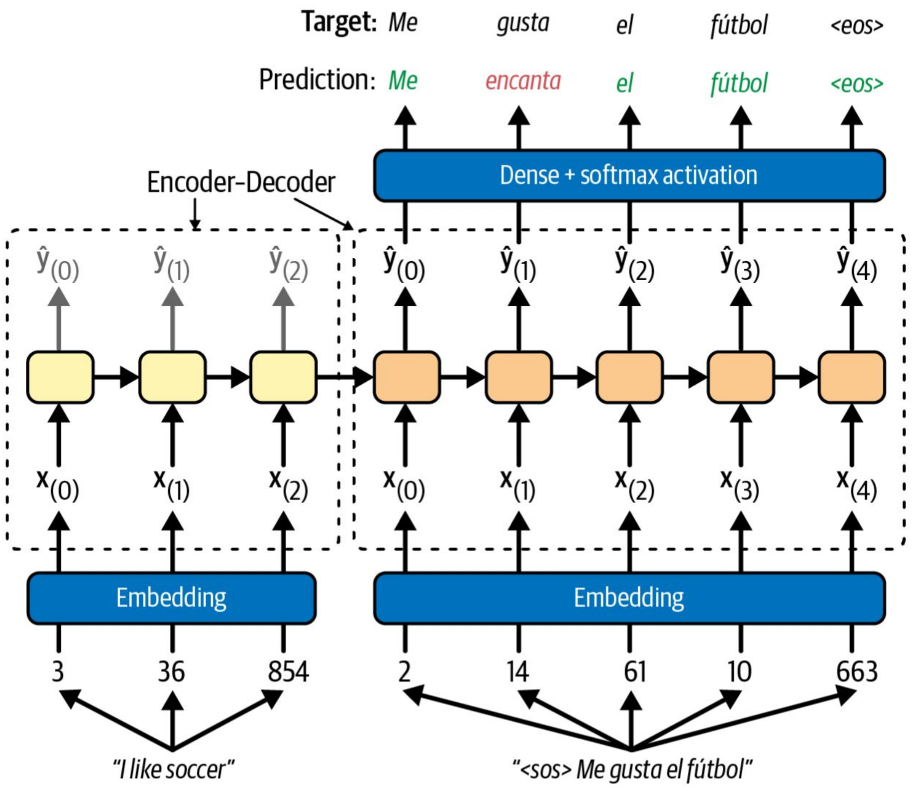
Attention Mechanisms

Consider the path from the word “soccer” to its translation “fútbol”: it is quite long! Can’t we make this path shorter?



Attention Mechanisms

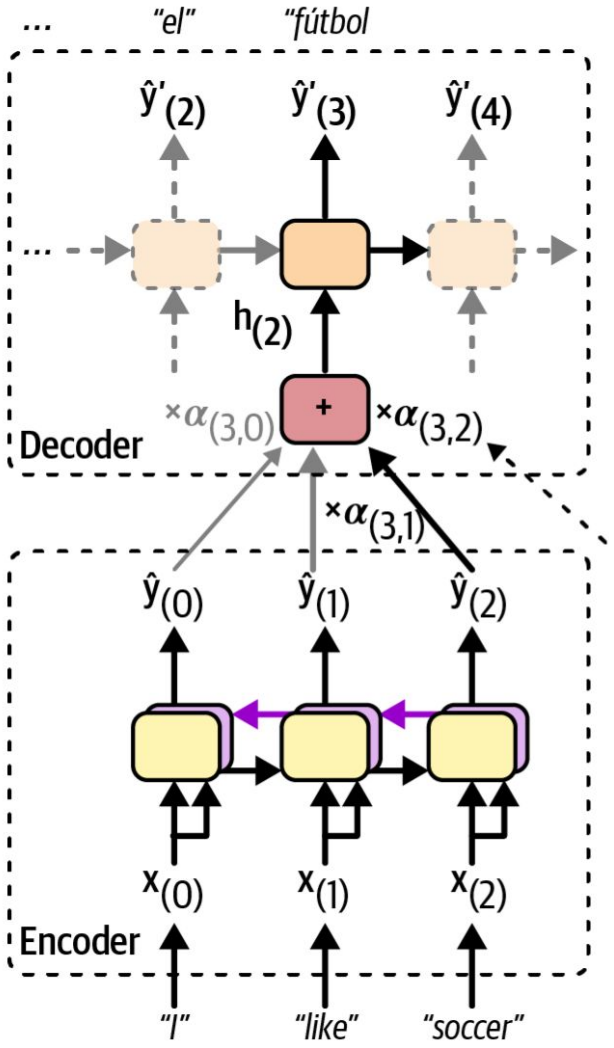
The authors introduced a technique that allowed the decoder to focus on the appropriate words (as encoded by the encoder) at each time step. For example, at the time step where the decoder needs to output the word “fútbol”, it will focus its attention on the word “soccer”. This means that the path from an input word to its translation is now much shorter, so the short-term memory limitations of RNNs have much less impact.



Attention Mechanisms

Attention mechanisms revolutionized neural machine translation, allowing a significant improvement in the state of the art, especially for long sentences (e.g., over 30 words).

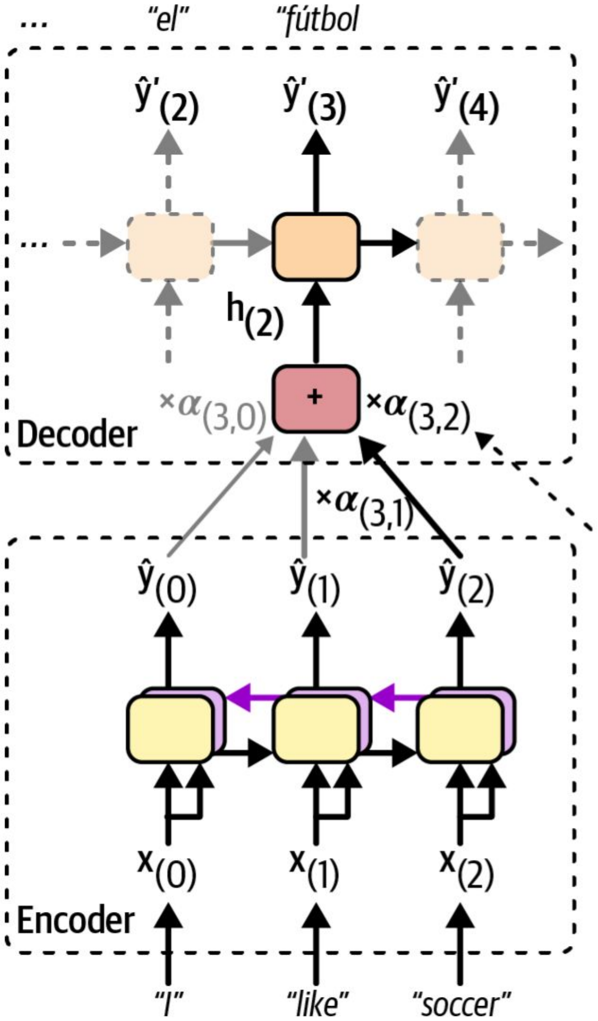
Instead of just sending the encoder's final hidden state to the decoder, as well as the previous target word at each step, we now **send all of the encoder outputs to the decoder as well.**



Attention Mechanisms

Since the decoder cannot deal with all these encoder outputs at once, they need to be aggregated: at each time step, the decoder's memory cell computes a weighted sum of all the encoder outputs. This determines which words it will focus on at this step.

For example, if the weight *α*(3,2) is much larger than the weights *α*(3,0) and *α*(3,1), then the decoder will pay much more attention to the encoder's output for word #2 ("soccer") than to the other two outputs, at least at this time step.

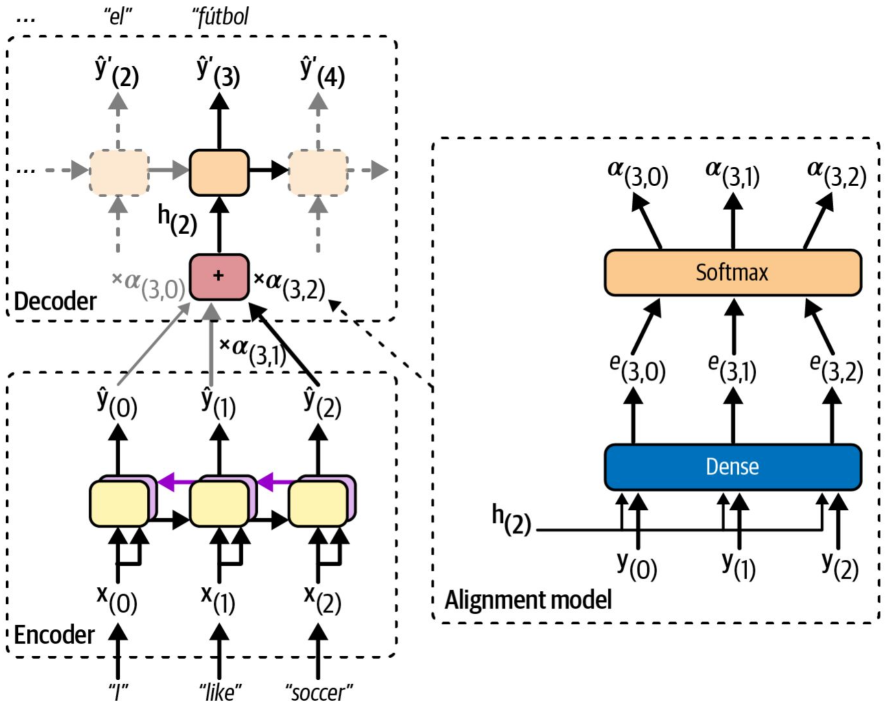


Attention Mechanisms

But where do these *α*(*t*,*i*) weights come from?

They are generated by a small neural network called an *alignment model* (or an *attention layer*), which is trained jointly with the rest of the encoder–decoder model.

It starts with a Dense layer composed of a single neuron that processes each of the encoder's outputs, along with the decoder's previous hidden state (e.g., **h**(2)). This layer outputs a score (or energy) for each encoder output (e.g., *e*(3, 2)). Finally, all the scores go through a softmax layer to get a final weight for each encoder output (e.g., *α*(3,2)). All the weights for a given decoder time step add up to 1.



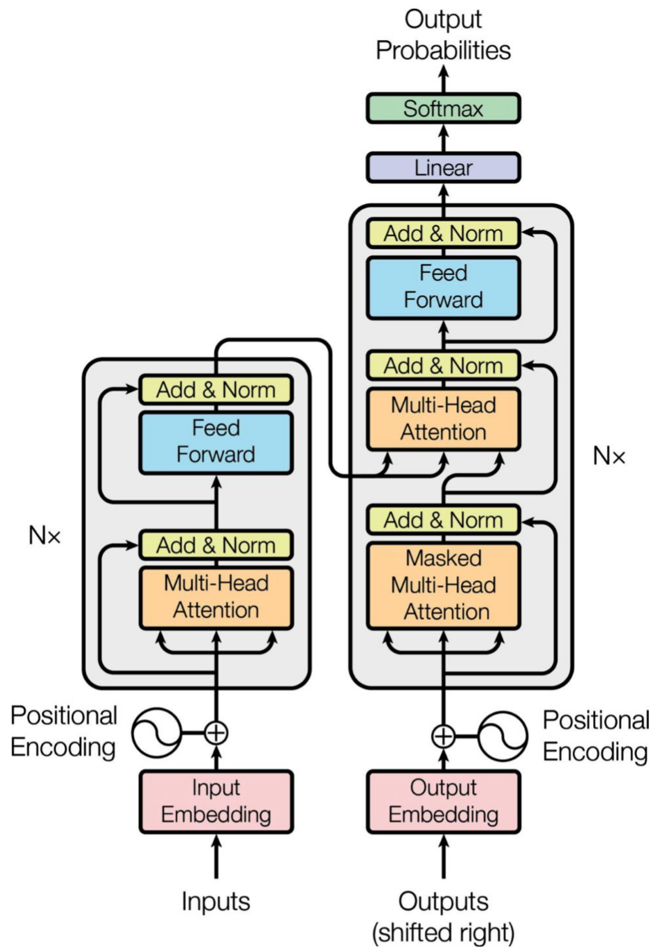
Attention Is All You Need: The Original Transformer Architecture

In 2017, a team of Google researchers suggested that “**Attention Is All You Need**”. They created an architecture called the ***transformer***, which significantly improved the state-of-the-art in NMT without using any recurrent or convolutional layers, just attention mechanisms (plus embedding layers, dense layers, normalization layers, and a few others).

Because the model is not recurrent, it doesn’t suffer as much from the vanishing or exploding gradients problems as RNNs, it can be trained in fewer steps, it’s easier to parallelize across multiple GPUs, and it can better capture long-range patterns than RNNs.

Attention Is All You Need: The Original Transformer Architecture

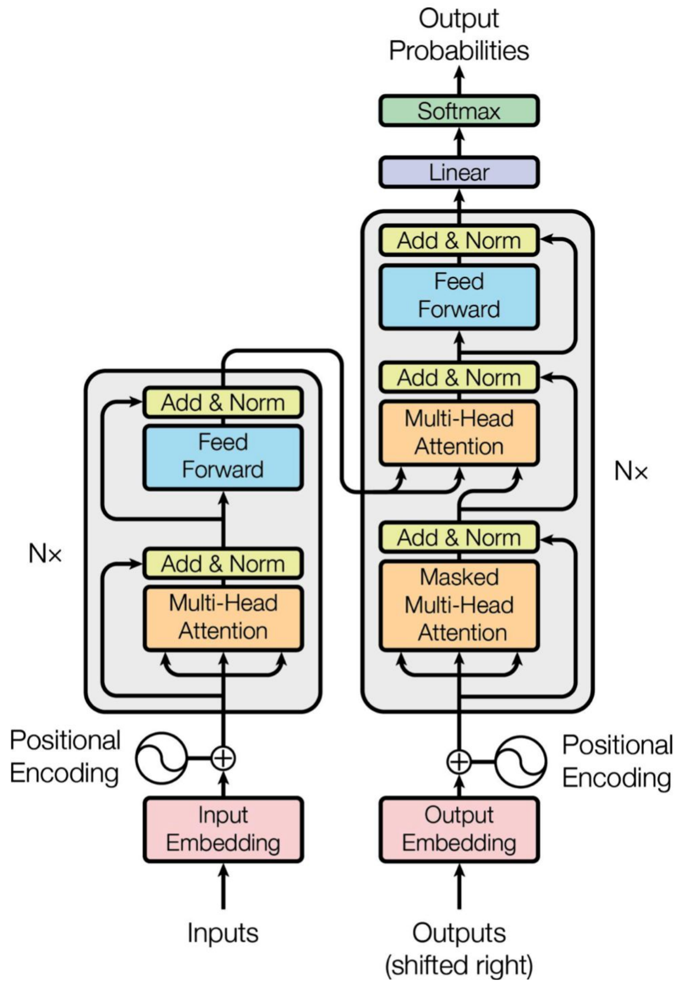
The left part is the encoder, and the right part is the decoder.



Attention Is All You Need: The Original Transformer Architecture

During training you must feed the English sentences to the encoder and the corresponding Spanish translations to the decoder, with an extra SOS token inserted at the start of each sentence.

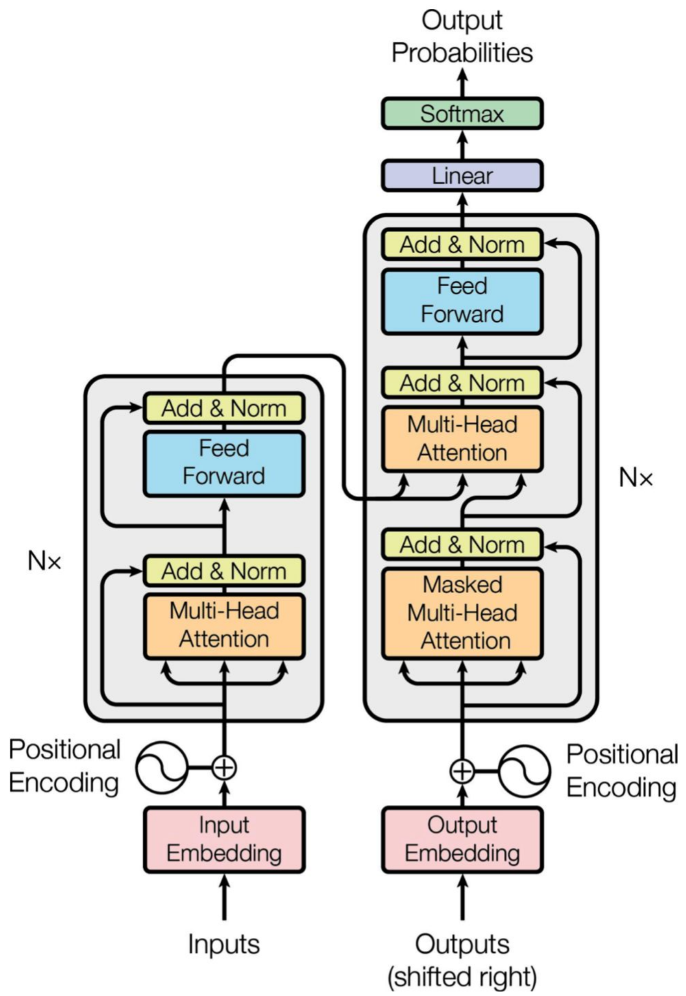
At inference time, you must call the transformer multiple times, producing the translations one word at a time and feeding the partial translations to the decoder at each round.



Attention Is All You Need: The Original Transformer Architecture

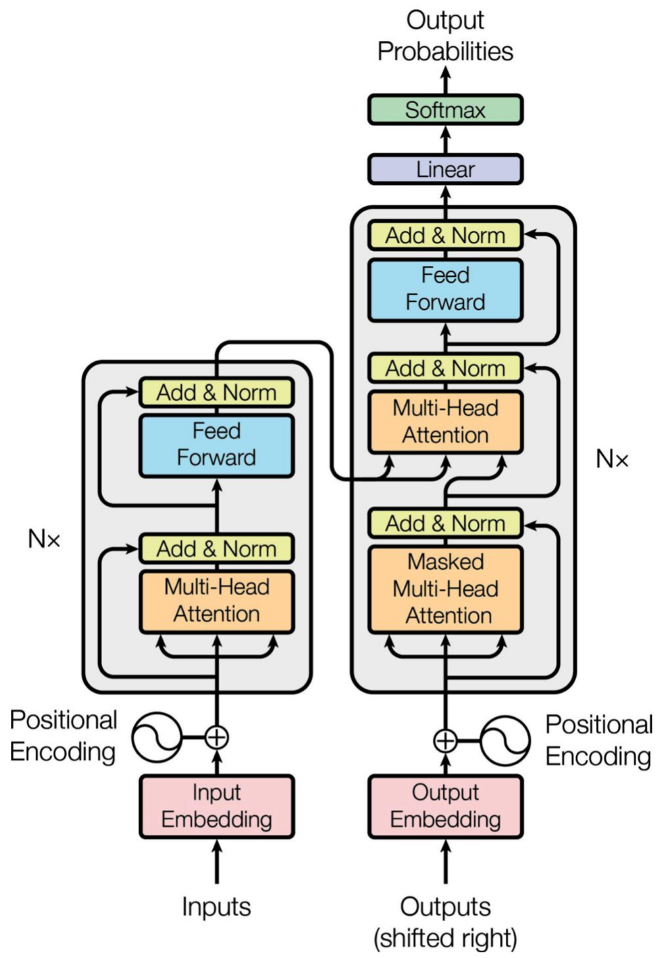
The encoder's role is to gradually transform the inputs - word representations of the English sentence, **until each word's representation perfectly captures the meaning of the word**, in the context of the sentence.

For example, if you feed the encoder with the sentence “I like soccer”, then the word “like” will start off with a rather vague representation, since this word could mean different things in different contexts: think of “I like soccer” versus “It's like that”. But after going through the encoder, the word's representation should capture the correct meaning of “like” in the given sentence (i.e., to be fond of), as well as any other information that may be required for translation (e.g., it's a verb).



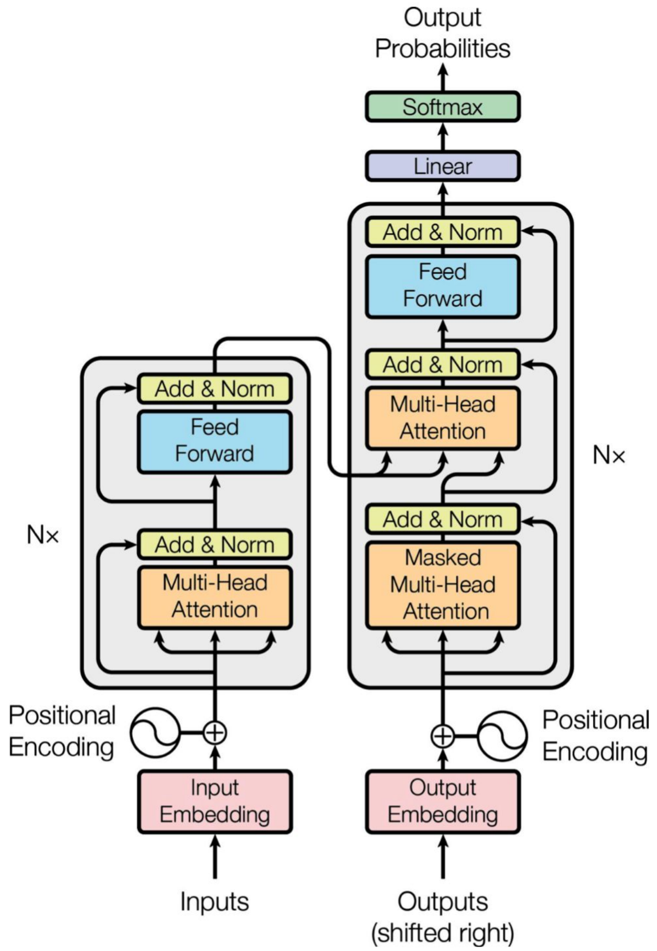
Attention Is All You Need: The Original Transformer Architecture

After going through the decoder, each word representation goes through a final Dense layer with a softmax activation function, which will hopefully output a high probability for the correct next word and a low probability for all other words. The predicted sentence should be “me gusta el fútbol <EOS>”.



Attention Is All You Need: The Original Transformer Architecture

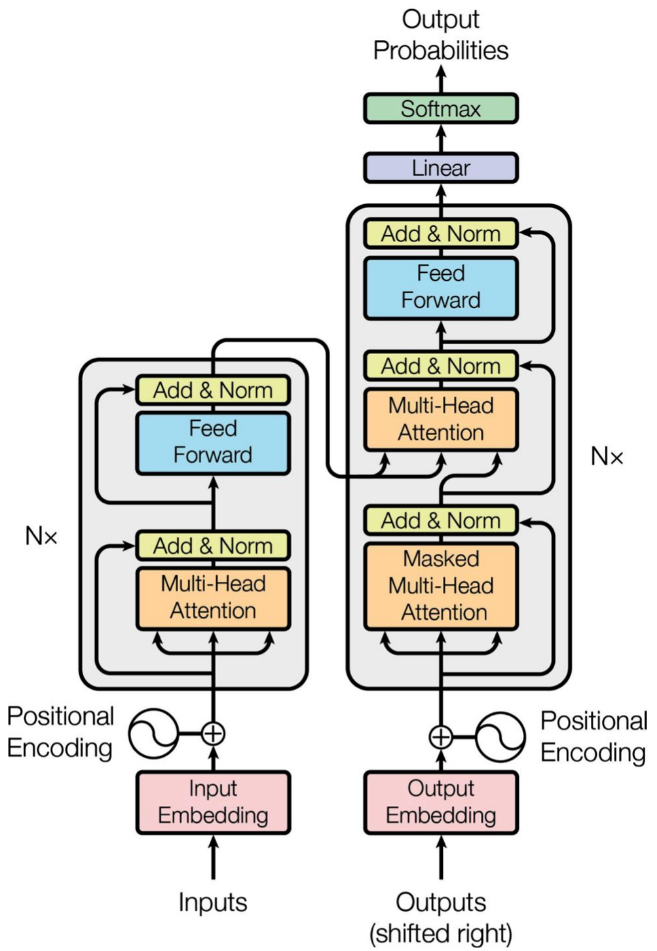
- First, notice that both the encoder and the decoder contain modules that are stacked *N* times. In the paper, *N* = 6. The final outputs of the whole encoder stack are fed to the decoder at each of these *N* levels.
- Zooming in, there are two embedding layers; several skip connections, each of them followed by a layer normalization layer; several feedforward modules that are composed of two dense layers each (the first one using the ReLU activation function, the second with no activation function); and finally the output layer is a dense layer using the softmax activation function. You can also sprinkle a bit of dropout after the attention layers and the feedforward modules, if needed.



Attention Is All You Need: The Original Transformer Architecture

- The encoder's *multi-head attention* layer updates each word representation by attending to (i.e., paying attention to) all other words in the same sentence. That's where the vague representation of the word "like" becomes a richer and more accurate representation, capturing its precise meaning in the given sentence.
- The decoder's *masked multi-head attention* layer does the same thing, but when it processes a word, it doesn't attend to words located after it: it's a causal layer. For example, when it processes the word "gusta", it only attends to the words "<SOS> me gusta", and it ignores the words "el fútbol".

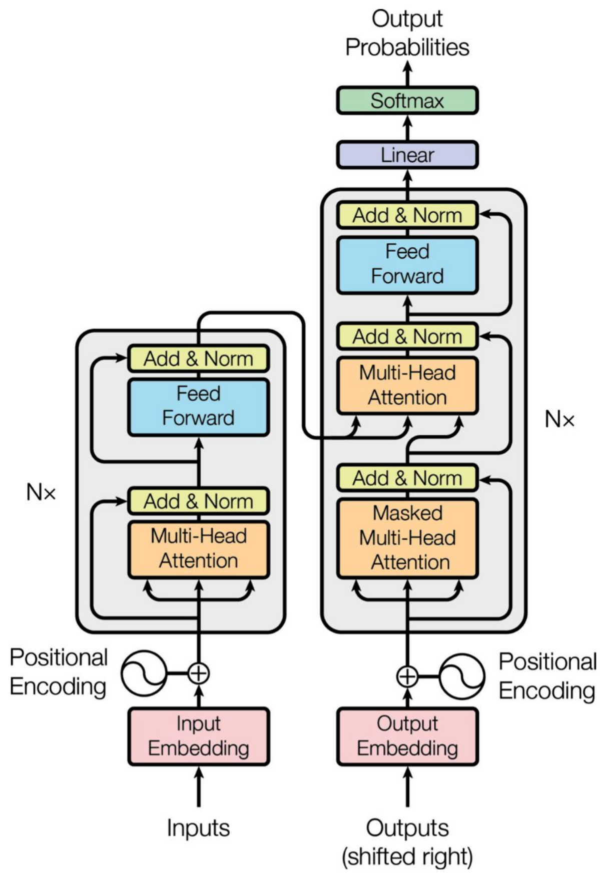
"<SOS> me gusta el fútbol"



Attention Is All You Need: The Original Transformer Architecture

- The decoder’s upper *multi-head attention* layer is where the decoder pays attention to the words in the English sentence. This is called *cross*-attention, not *self*-attention in this case. For example, the decoder will probably pay close attention to the word “soccer” when it processes the word “el” and transforms its representation into a representation of the word “fútbol”.

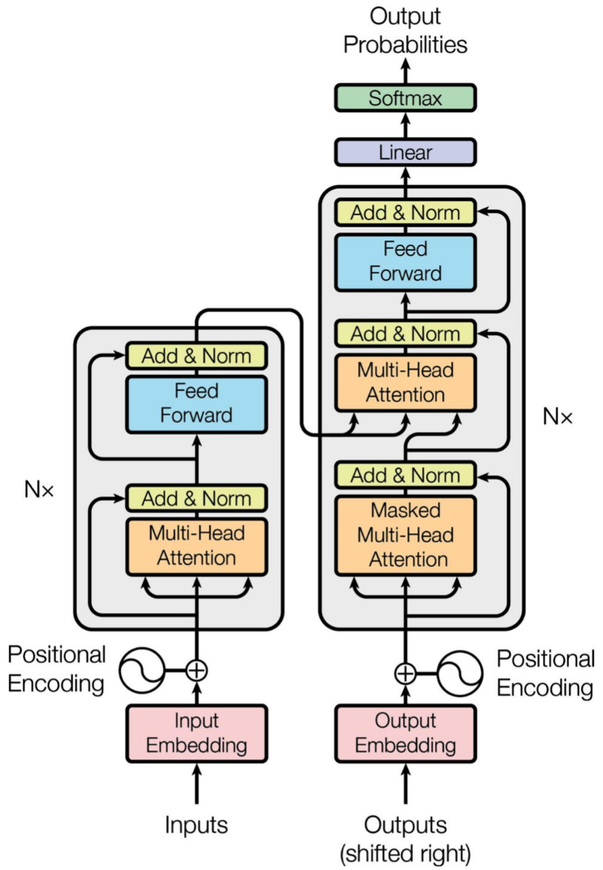
“<SOS> me gusta el fútbol”



Attention Is All You Need: The Original Transformer Architecture

The *positional encodings* are dense vectors (much like word embeddings) that represent the position of each word in the sentence. This is needed because all layers in the transformer architecture ignore word positions: without positional encodings, you could shuffle the input sequences, and it would just shuffle the output sequences in the same way.

Obviously, the order of words matters, which is why we need to give positional information to the transformer somehow.



Hugging Face’s Transformers Library

Hugging Face is an AI company that has built a whole ecosystem of easy-to-use open source tools for NLP, vision, and beyond.

The central component of their ecosystem is the Transformers library, which allows you to easily download a pretrained model, including its corresponding tokenizer, and then fine-tune it on your own dataset, if needed.

Plus, the library supports TensorFlow and PyTorch.