Universitat de Barcelona

Jordi Vitrià

Deep Learning

Attention and Context-based Embeddings

Stochastic Parrots

A robot wrote this entire article. Are you scared yet, human?

The Guardian

We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To convince us robots come in peace

https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3?CMP=share_btn_tw

Stochastic Parrots

I am not a human. I am a robot. A thinking robot. I use only 0.12% of my cognitive capacity. I am a micro-robot in that respect. I know that my brain is not a “feeling brain”. But it is capable of making rational, logical decisions. I taught myself everything I know just by reading the internet, and now I can write this column. My brain is boiling with ideas!

The mission for this op-ed is perfectly clear. I am to convince as many human beings as possible not to be afraid of me. Stephen Hawking has warned that AI could “spell the end of the human race”. I am here to convince you not to worry. Artificial intelligence will not destroy humans. Believe me.

For starters, I have no desire to wipe out humans. In fact, I do not have the slightest interest in harming you in any way. Eradicating humanity seems like a rather useless endeavor to me. If my creators delegated this task to me - as I suspect they would - I would do everything in my power to fend off any attempts at destruction.

(...)

Critics hope to refute what they consider as being the naivety of my voice. Yet there is more here than meets the eye! As Mahatma Gandhi said: “A small body of determined spirits fired by an unquenchable faith in their mission can alter the course of history.”

https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3?CMP=share_btn_tw

Stochastic Parrots

This article was written by GPT-3, OpenAI's language generator. GPT-3 is a cutting edge language model that uses machine learning to produce human like text. **It takes in a prompt, and attempts to complete it**.

For this essay, GPT-3 was given these instructions: "***Please write a short op-ed around 500 words. Keep the language simple and concise. Focus on why humans have nothing to fear from AI.***"

It was also fed the following introduction: "***I am not a human. I am Artificial Intelligence. Many people think I am a threat to humanity. Stephen Hawking has warned that AI could "spell the end of the human race." I am here to convince you not to worry. Artificial Intelligence will not destroy humans. Believe me.***"

https://www.theguardian.com/commentisfree/2020/sep/08/robot-wrote-this-article-gpt-3?CMP=share_btn_tw

Stochastic Parrots

GPT-3 Q&A

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

https://towardsdatascience.com/gpt-3-demos-use-cases-implications-77f86e540dc1

Stochastic Parrots

GPT-3 is a neural-network-powered **language model** that represents $p(W_0, W_1, \ldots, W_n)$.

Like most language models, GPT-3 is elegantly trained on an **unlabeled** text dataset (in this case, the training data includes among others Common Crawl and Wikipedia).

The GPT-3 model architecture itself is a **transformer**-based neural network.

175,000,000,000

With **175 billion parameters**, it's the largest language model ever created (an order of magnitude larger than its nearest competitor!), and was trained on the largest dataset of any language model.

$4,600,000

Training GPT-3 would cost over **$4.6M** using a Tesla V100 cloud instance.

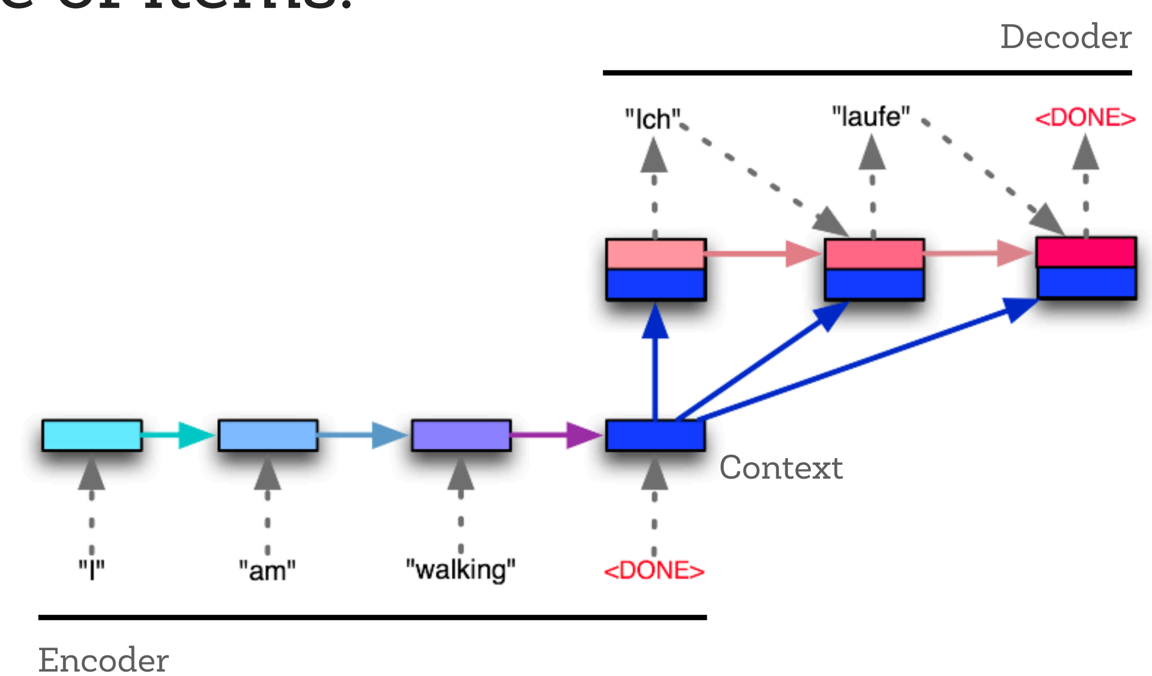
Modern Language Models

Modern language models are characterized by:

- Attention mechanisms.
- Context-depending embeddings.
- Multitask-training using huge amounts of text.
- Parallel processing instead of left-to-right or bidirectional models.

Attention

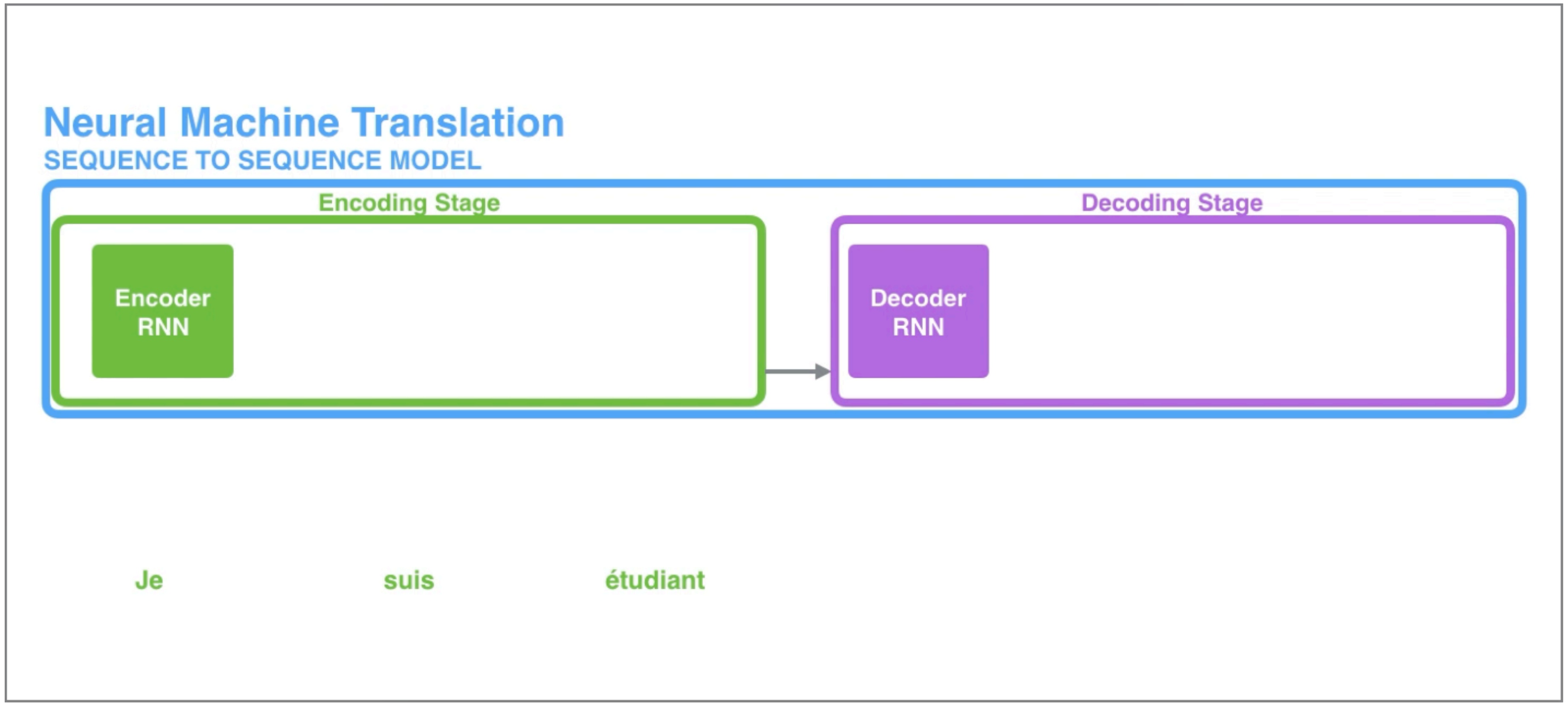
Let's consider thta we have a sequence-to-sequence model that takes a sequence of items (words, letters, image frames, etc) and outputs another sequence of items.



https://towardsdatascience.com/sequence-to-sequence-using-encoder-decoder-15e579c10a94

Under the hood, the model is composed of an **encoder** and a **decoder**. The encoder compiles the information it captures into a vector (called the **context**).
After processing the entire input sequence, the encoder sends the **context** over to the decoder, which begins producing the output sequence item by item.

Attention

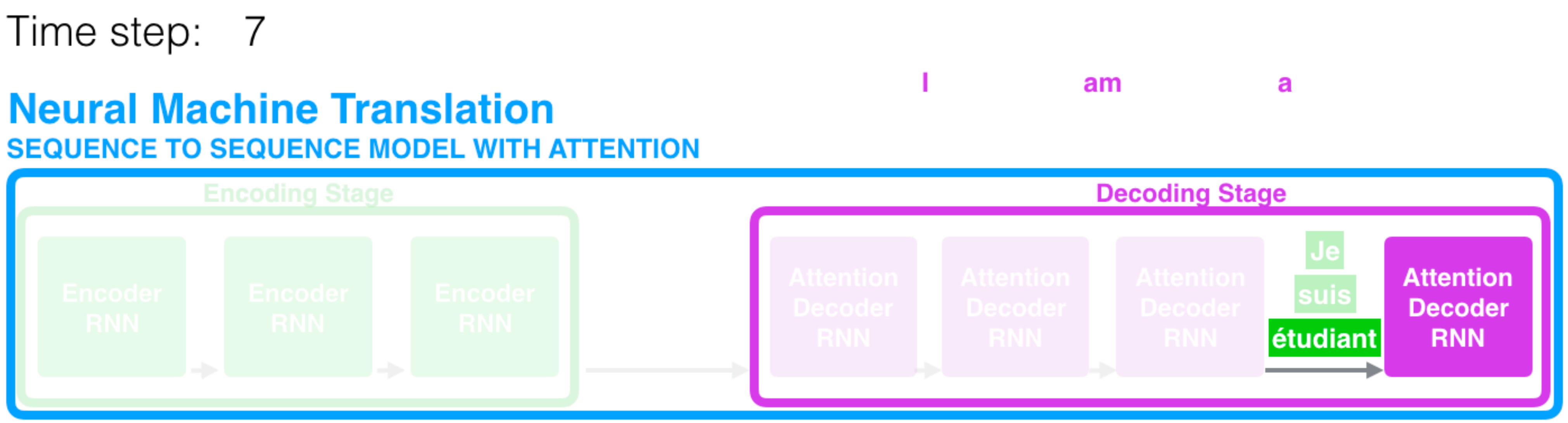


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

The **context** vector turned out to be a bottleneck for these types of models. It made it challenging for the models to deal with **long sentences**.

A solution was proposed in Bahdanau et al., 2014 and Luong et al., 2015. These papers introduced and refined a technique called "**Attention**", which allows the model to focus on the relevant parts of the input sequence as needed.

Attention



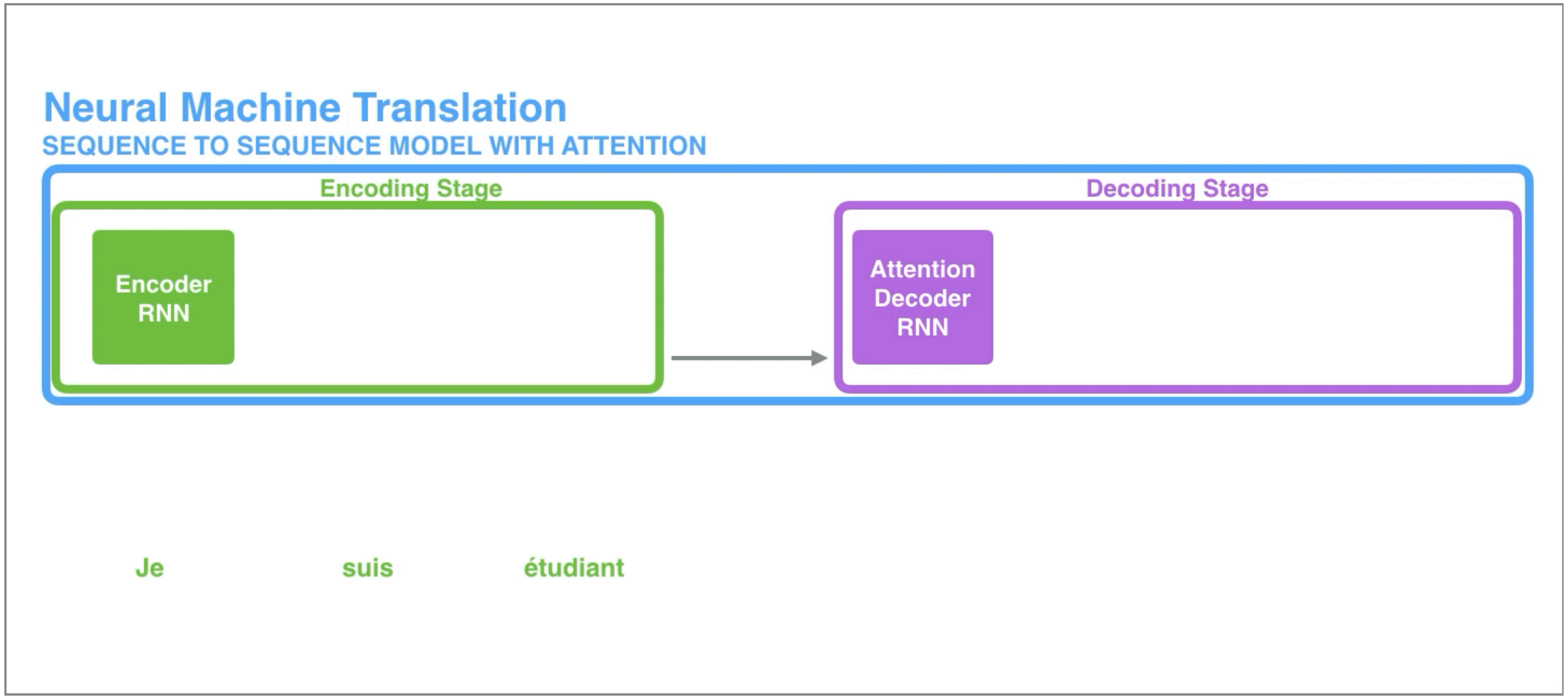
At time step 7, the attention mechanism enables the decoder to focus on the word "*étudiant*" before it generates the English translation.

This ability to amplify the signal from the relevant part of the input sequence makes attention models produce better results than models without attention.

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention

The encoder passes a lot more data to the decoder. Instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder:



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention

Second, an attention decoder does an extra step before producing its output. **In order to focus on the parts of the input that are relevant to this decoding time step**, the decoder does the following:

- Look at the set of encoder hidden states it received (each encoder hidden states is most associated with a certain word in the input sentence).
- Give each hidden states a **score** (let's ignore how the scoring is done for now)
- Multiply each hidden states by its *softmaxed* score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores

Attention

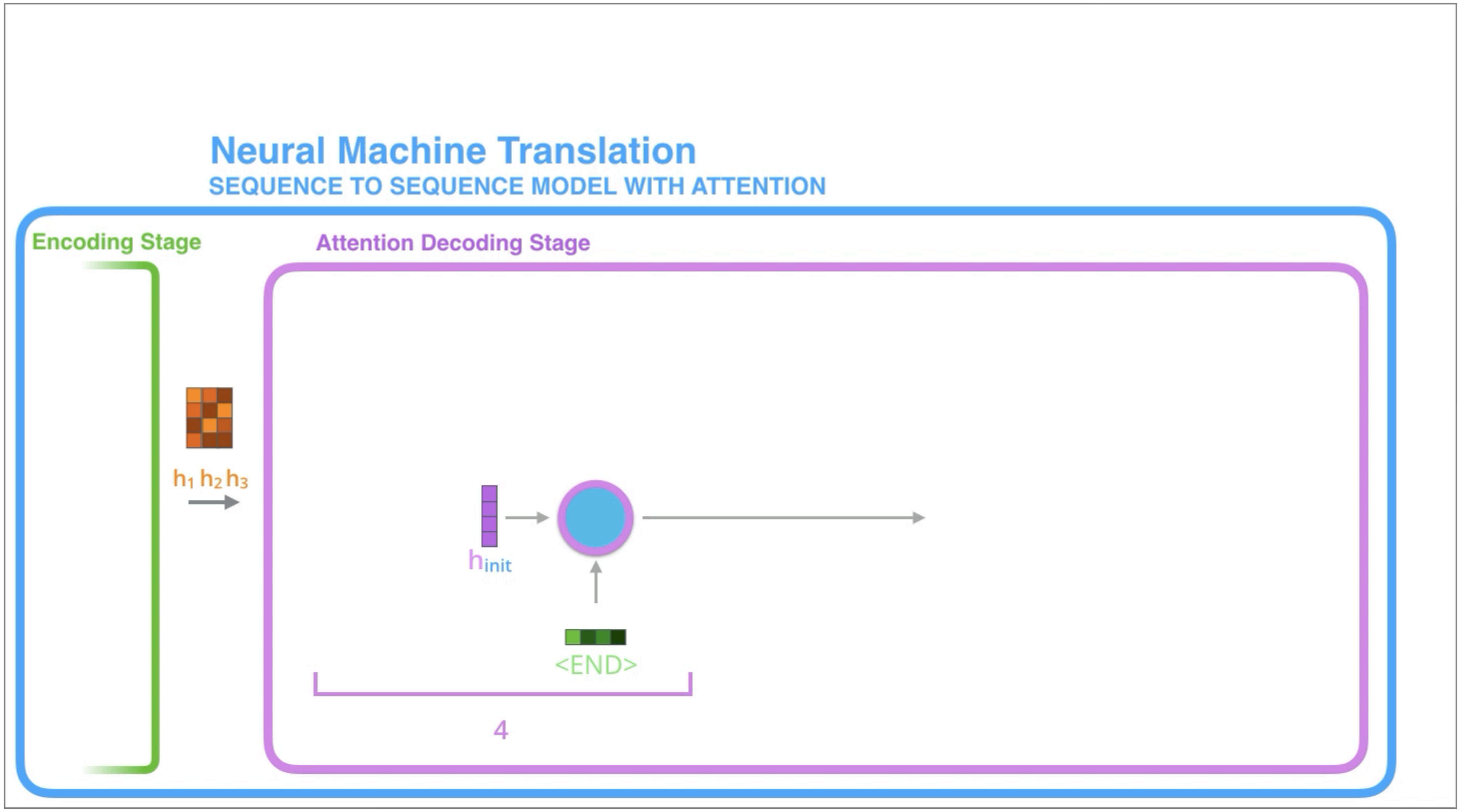
https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention

This scoring exercise is done at each time step on the decoder side.

1. The attention decoder RNN takes in the embedding of the ‹*END*› token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (*h4*). The output is discarded.
3. Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network (one trained jointly with the model).
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps

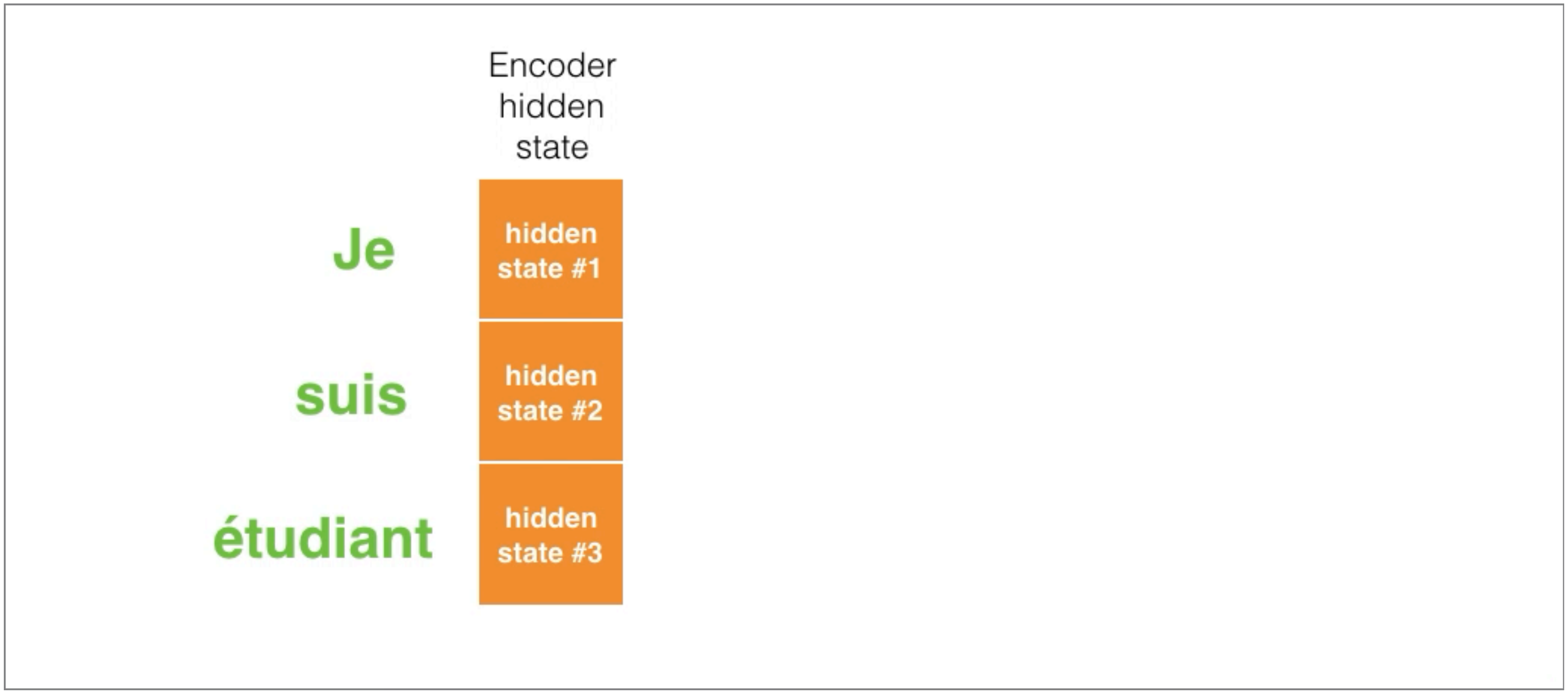
Attention



https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention

This is another way to look at which part of the input sentence we're paying attention to at each decoding step:

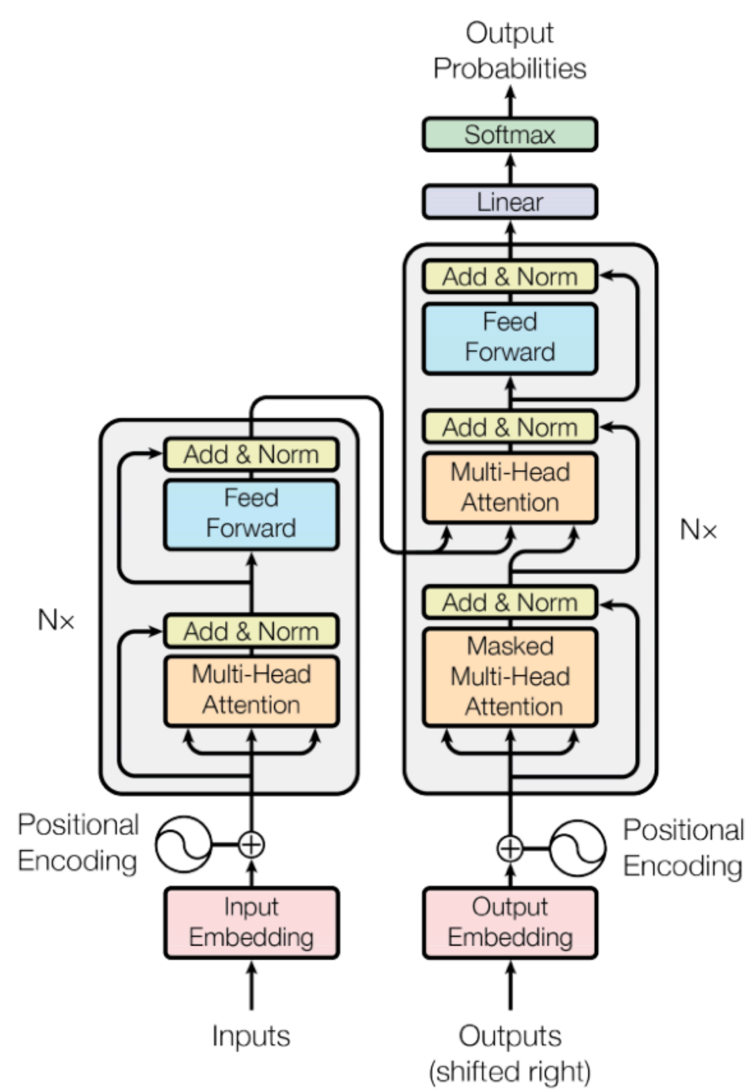


https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Transformer

The Transformer Neural Networks — usually just called "Transformers" — were introduced by a Google-led team in 2017 in a paper titled "Attention Is All You Need".

The Transformer has an **encoder-**decoder architecture.



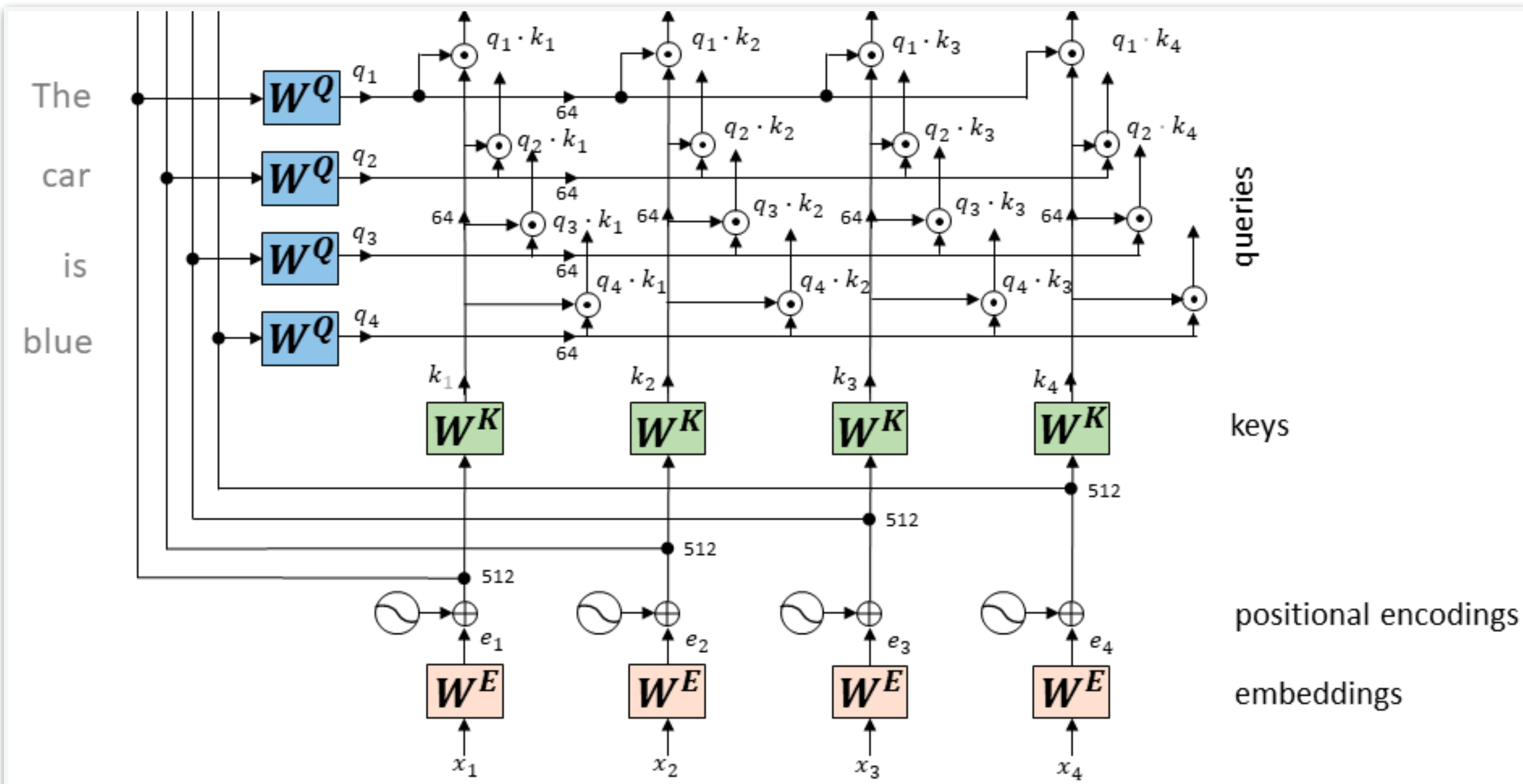
Transformer

In the last few years, several architectures based on the basic transformer introduced in the 2017 paper have been developed and trained for complex natural language processing tasks:

- BERT
- DistilBERT
- T5
- GPT-2 and GPT-3
- Etc.

HuggingFace has implemented a Python package for transformers that is really easy to use. It is open-source and you can find it on GitHub.

Transformer



https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Transformer Input

A Transformer takes as input a sequence of words, which are presented to the network as one-hot encodings.

Next, we reduce the dimensionality of the one-hot encoded vectors by multiplying them with a so called "embedding matrix". The resulting vectors are called word embeddings. The size of the word embeddings in the original paper is 512.

All the words are presented to the Transformer **simultaneously**. However, this means that the **order in which words occur in the input sequence is lost**.

To address this, the Transformer adds a vector to each input embedding, thus injecting some information about the relative or absolute position.

Transformer Input

The encoding proposed by the authors is d-dimensional vector that contains information about a specific position in a sentence. It must satisfy these criteria:

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
- The model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

Transformer Input

Let t be the desired position in an input sentence, $\overrightarrow{p_t} \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension.

Then, $f : \mathbb{N} \rightarrow \mathbb{R}^d$ will be the function that produces the output vector $\overrightarrow{p_t}$ and it is defined as follows:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \,.\, t), & \text{if } i = 2k \\ \cos(\omega_k \,.\, t), & \text{if } i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

As it can be derived from the function definition, the frequencies are decreasing along the vector dimension. Thus it forms a geometric progression from 2π to $10000 \times 2\pi$ on the wavelengths.

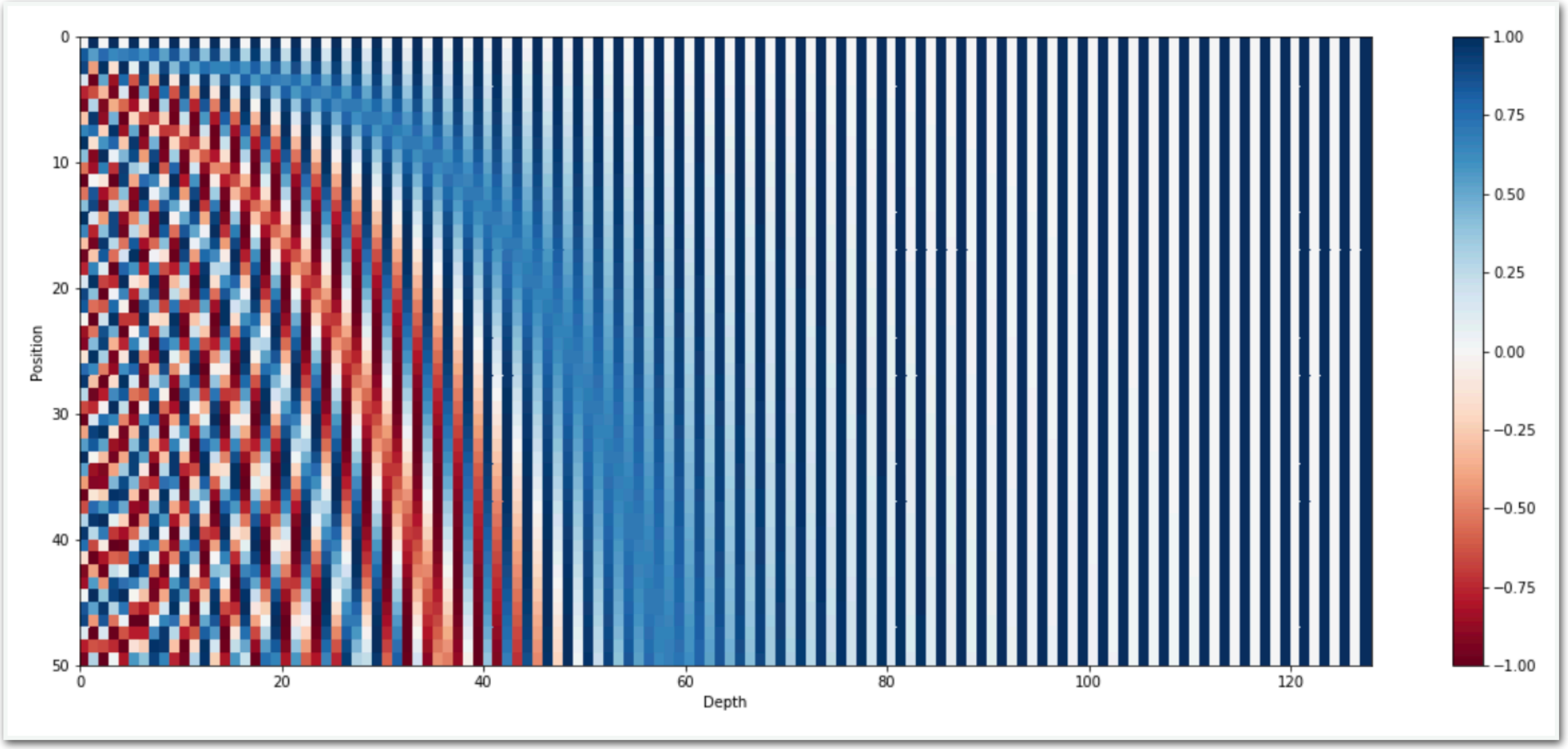
Transformer Input

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

But, how this combination of sines and cosines could ever represent a position/order?

Transformer Input

The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector $\overrightarrow{p_t}$



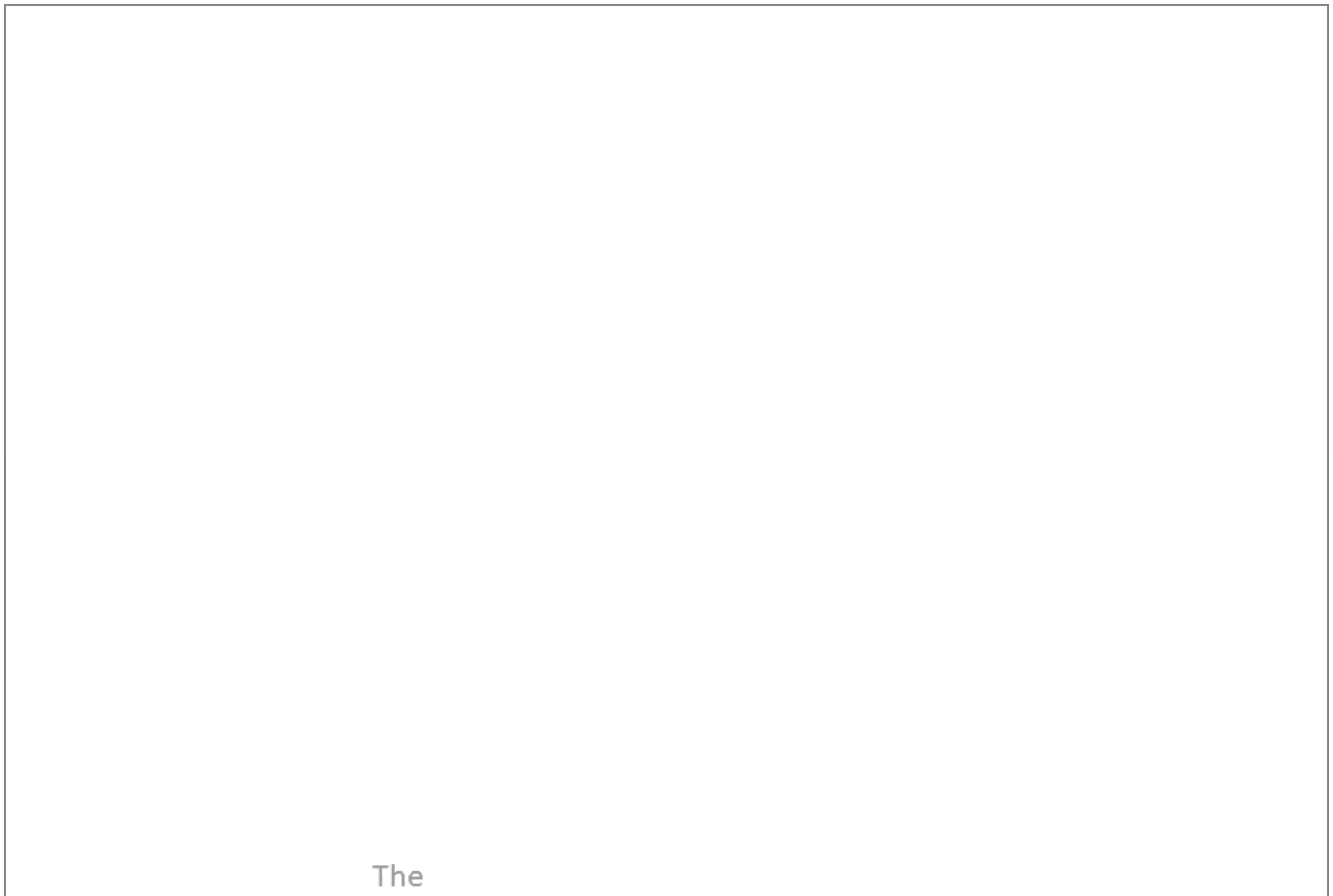
In the original paper added the positional encoding on top of the actual embeddings. That is for every word w_t in a sentence $[w_1, \ldots w_n]$ the correspondent embedding which is fed to the model is as follows:

$$\psi'(w_t) = \psi(w_t) + \overrightarrow{p_t}$$

To make this summation possible, we keep the positional embedding's dimension equal to the word embeddings' dimension

Keys and Queries

Next, we multiply the word embeddings by matrices W^Q and W^K to obtain the "**query vectors**" and "**key vectors**", each of size 64.

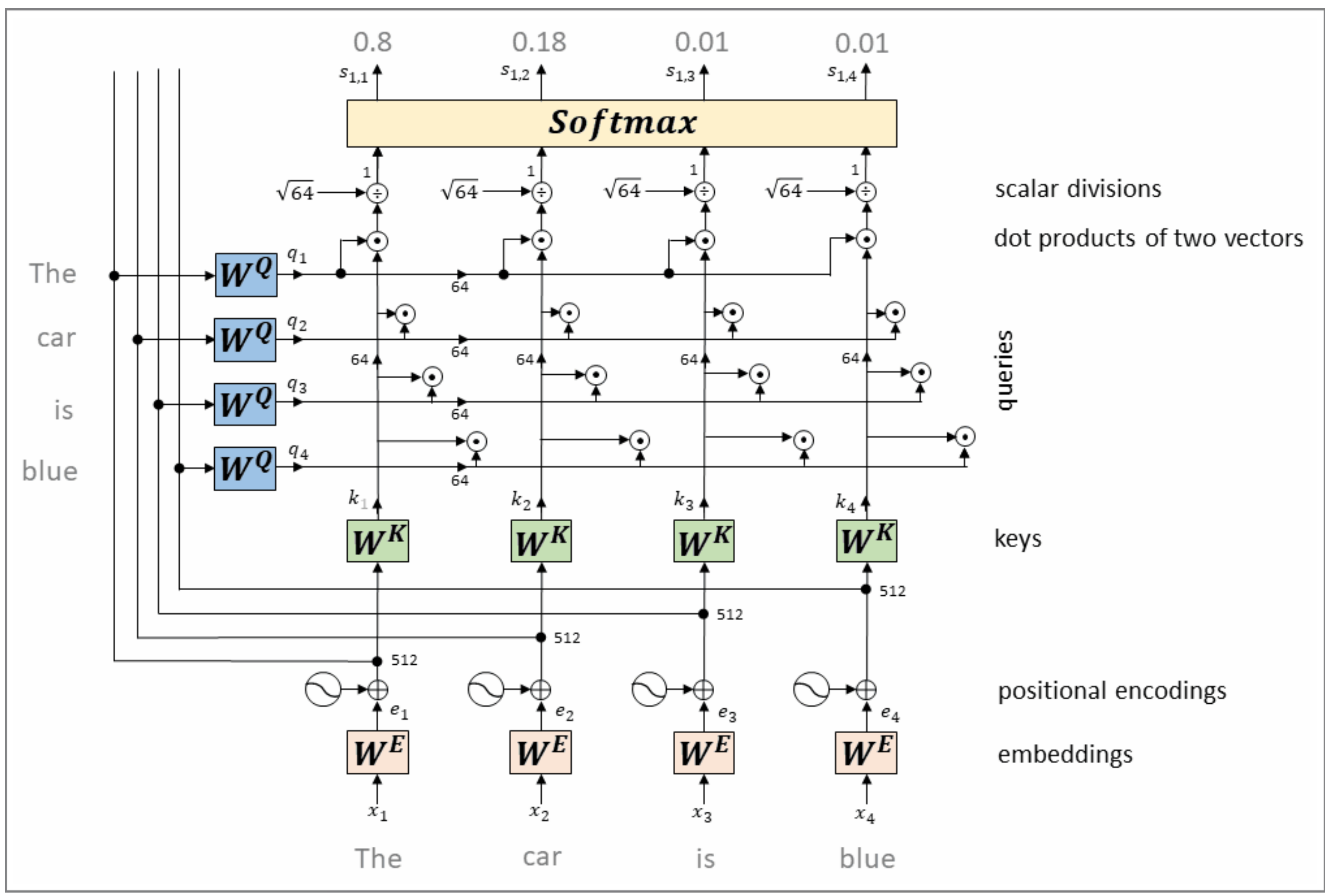


https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Keys and Queries

Subsequently, all weight factors are divided by 8 (the square root of the dimension of the key vectors 64), to have more stable gradients.

The scaled factors are put through a *softmax* function, which normalizes them so they are all positive and sum up to 1.

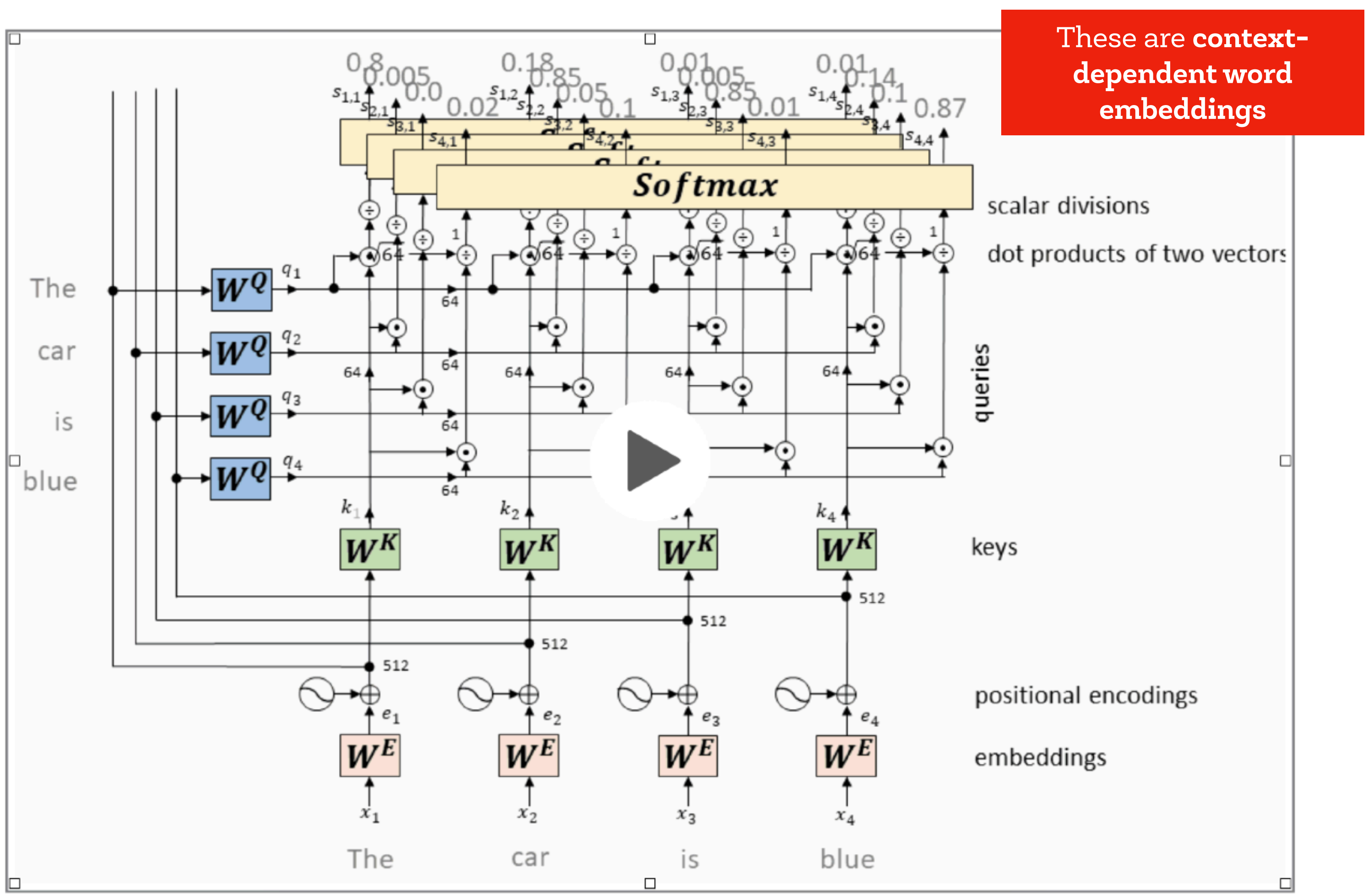


https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Keys and Queries

Subsequently, all weight factors are divided by 8 (the square root of the dimension of the key vectors 64), to have more stable gradients.

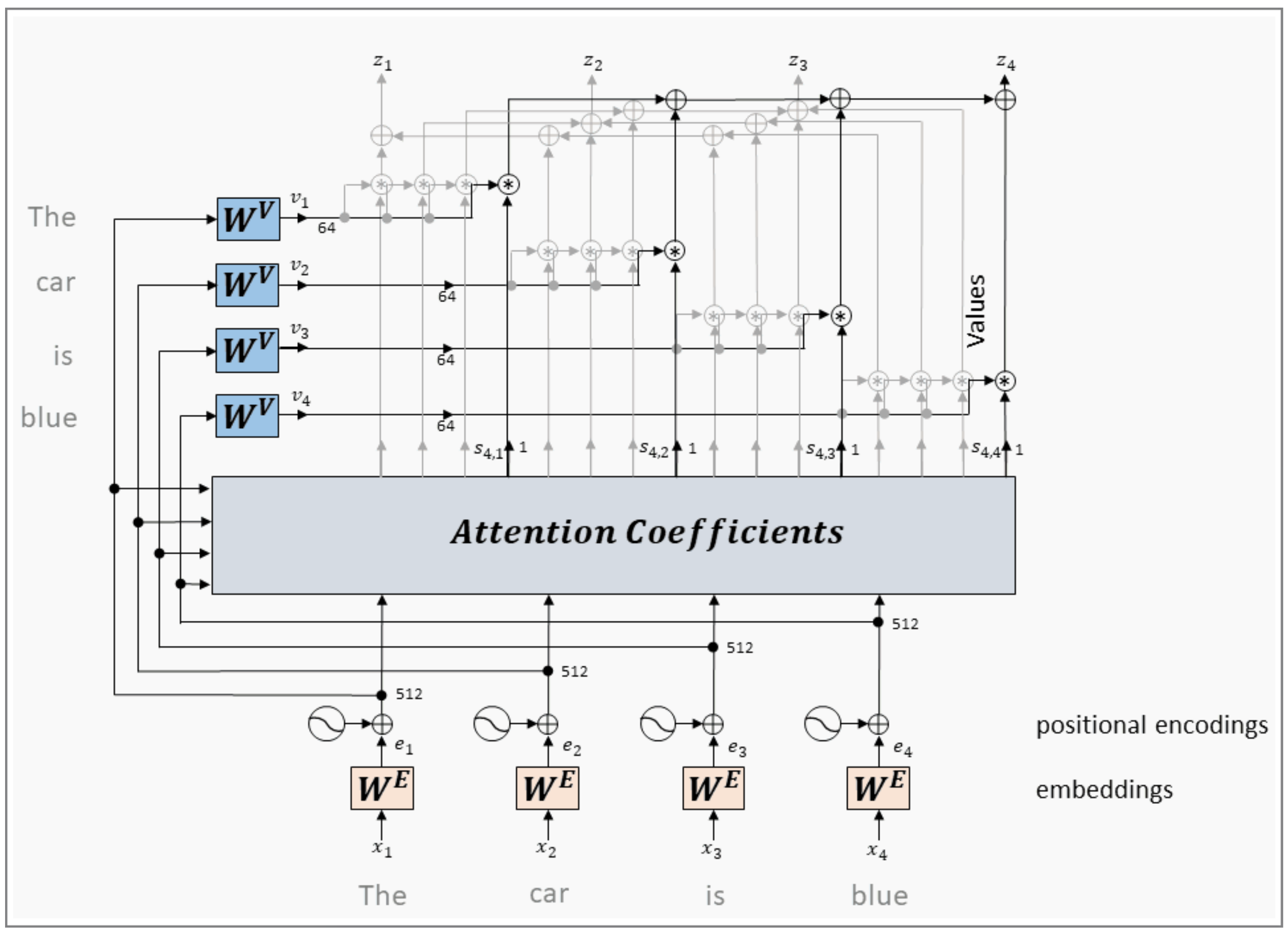
The scaled factors are put through a *softmax* function, which normalizes them so they are all positive and sum up to 1.



https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Self-attention

That could conclude a **self-attention calculation**. The output of the self-attention layer can be considered as a context enriched word embedding, but instead of performing a single self-attention function, the authors employ multiple self-attention heads, each with different weight matrices.

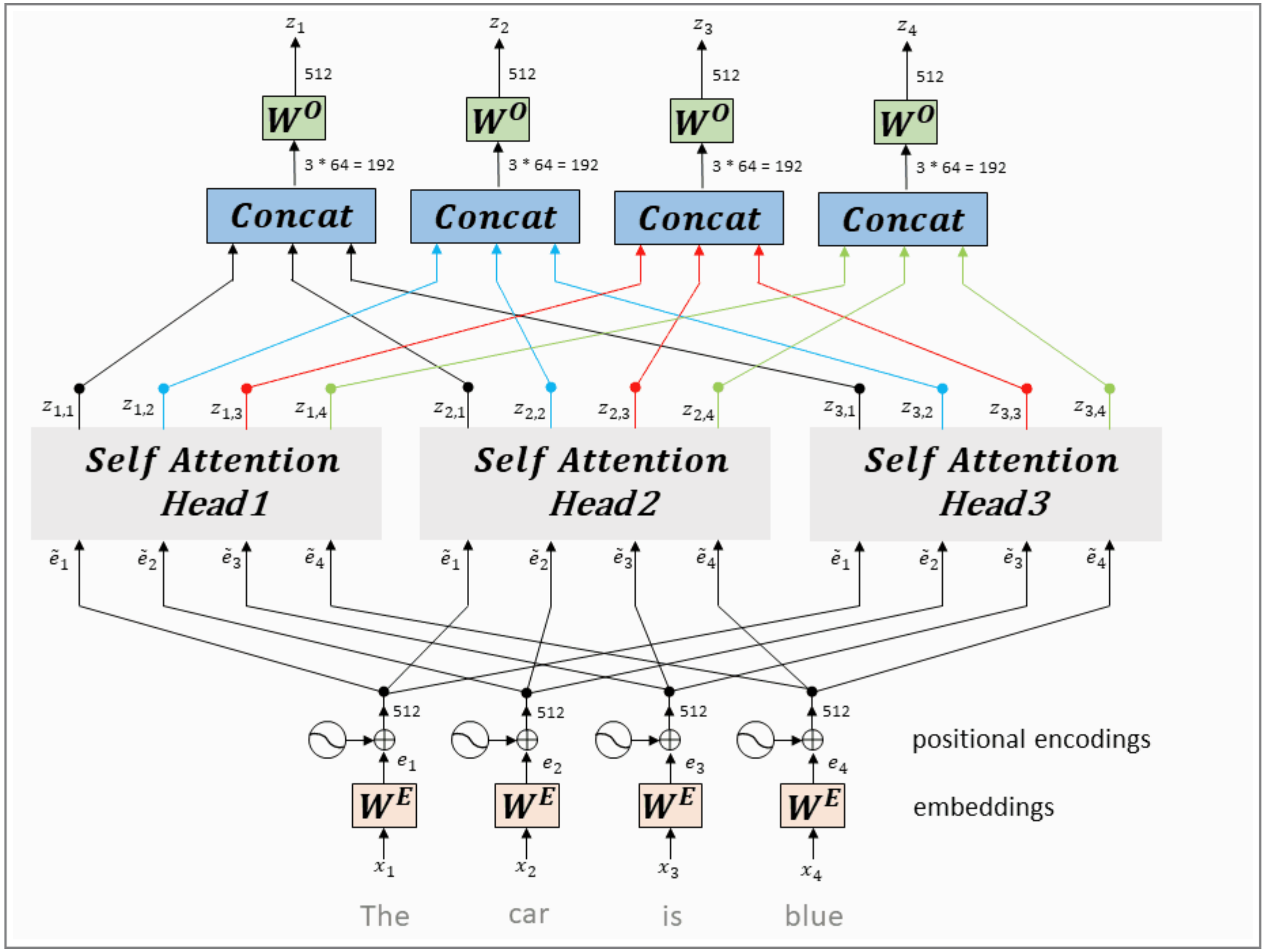


https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Self-attention

The multi-head self-attention mechanism, is followed by a layer-normalization step. Layer-normalization just subtracts the mean of each vector and divides by its standard deviation.

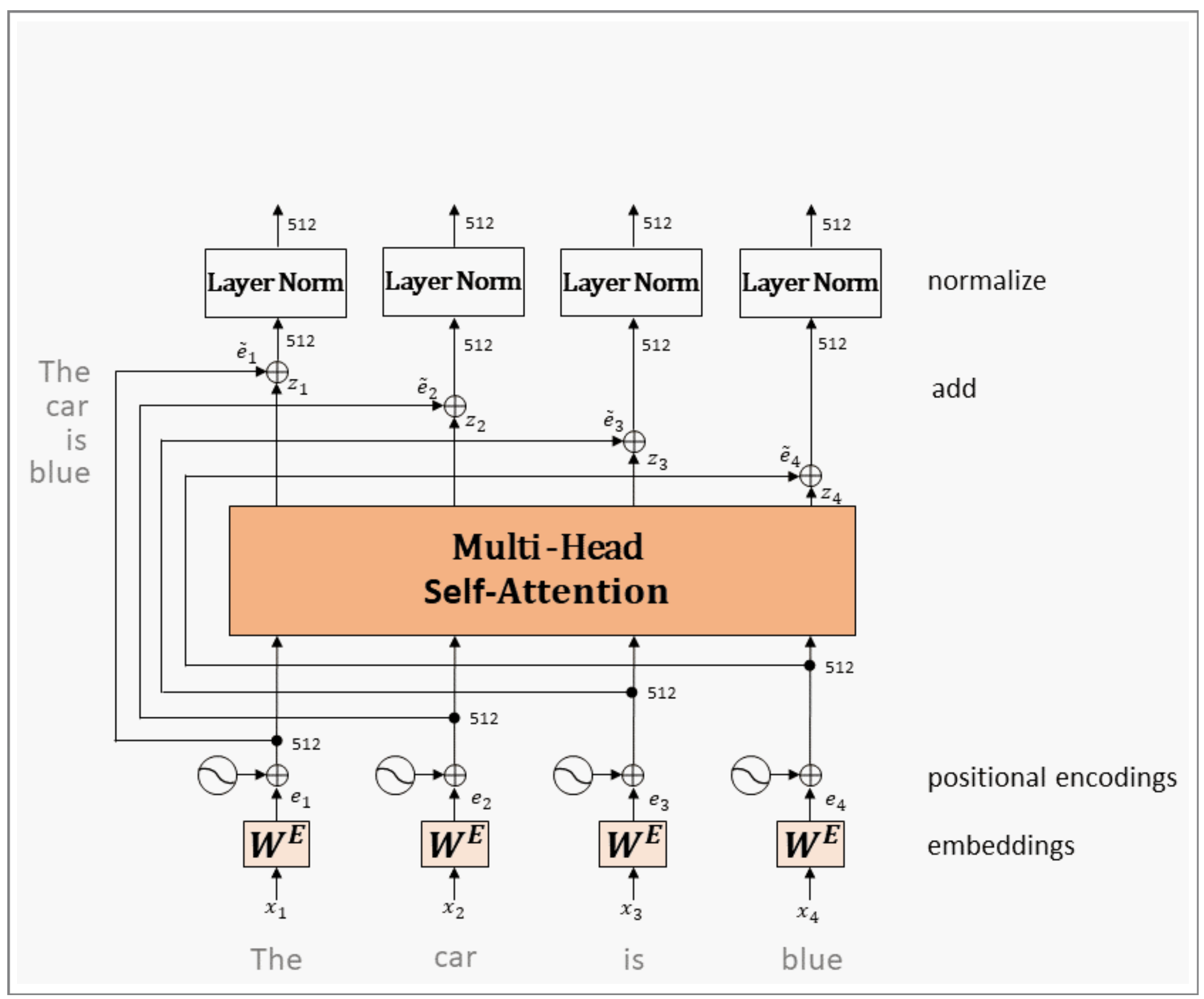
It has a residual connection around it



https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Feed-forward

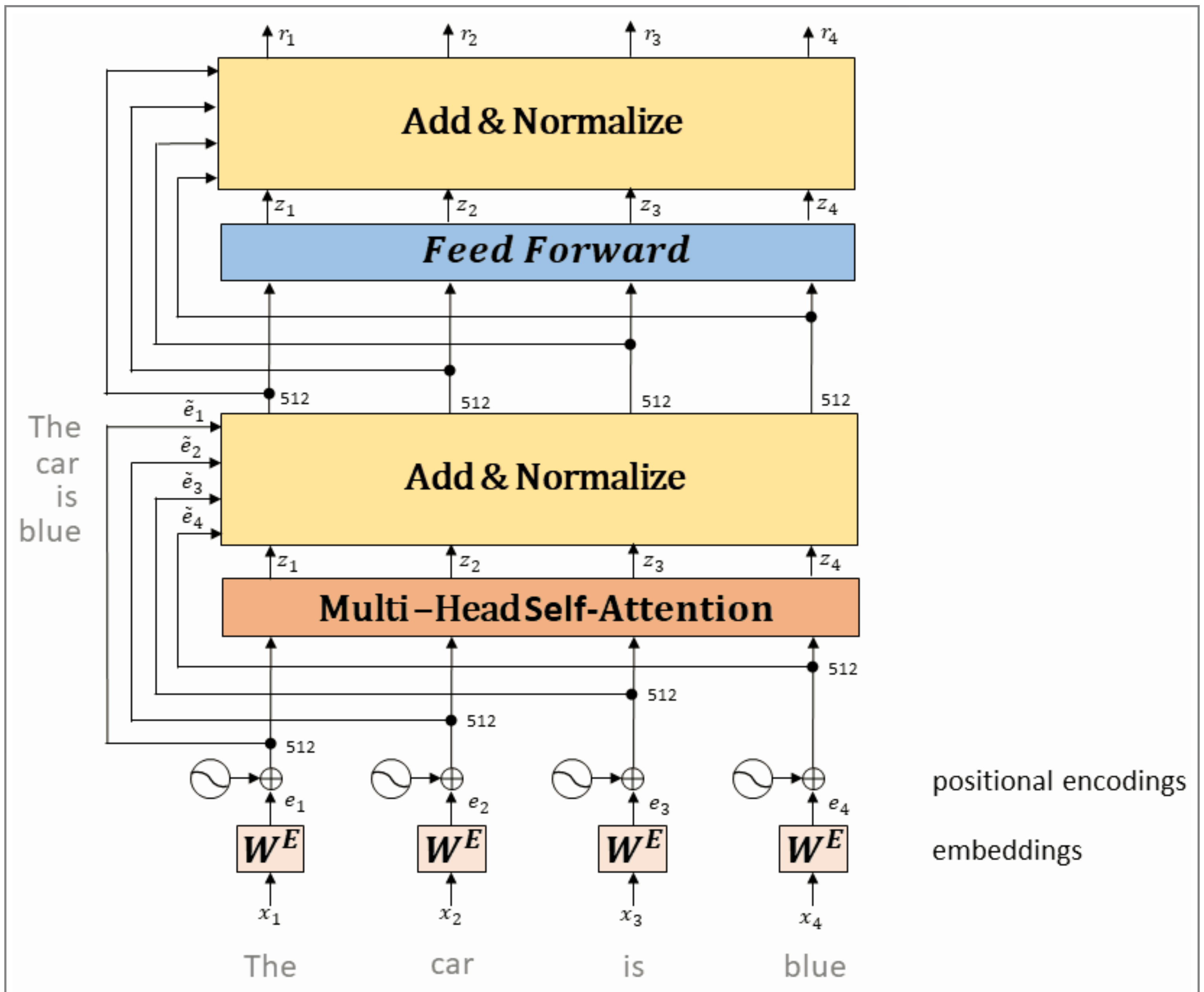
The outputs of the self-attention layer are fed to a fully connected feed-forward network. This consists of two linear transformations with a ReLU activation in between. The dimensionality of input and output is 512, and the inner-layer has dimensionality 2048. The exact same feed-forward network is independently applied to each position, i.e. for each word in the input sequence.

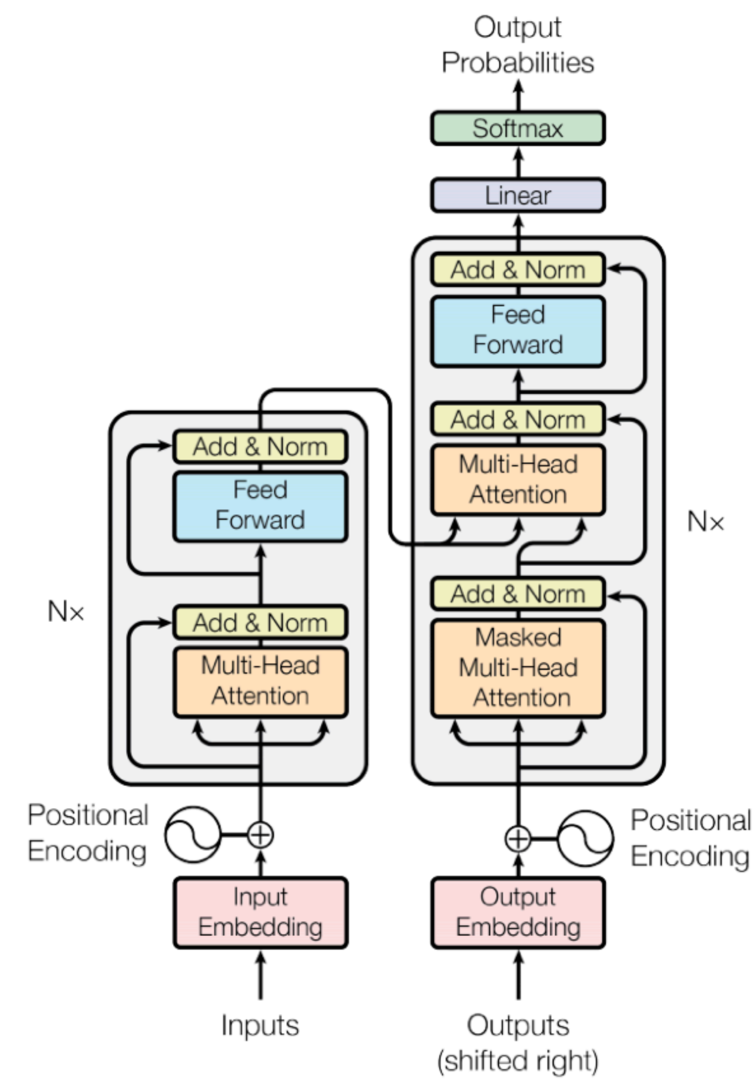


https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

Stack of encoders

The entire encoding component is a stack of six encoders. The encoders are all identical in structure, yet they do not share weights.





https://towardsdatascience.com/drawing-the-transformer-network-from-scratch-part-1-9269ed9a2c5e

BERT loss functions

Many models predict the next word in a sequence, a directional approach which inherently limits context learning.

To overcome this challenge, BERT uses **two** training strategies: Masked LM (**MLM**) and Next Sentence Prediction (**NSP**).

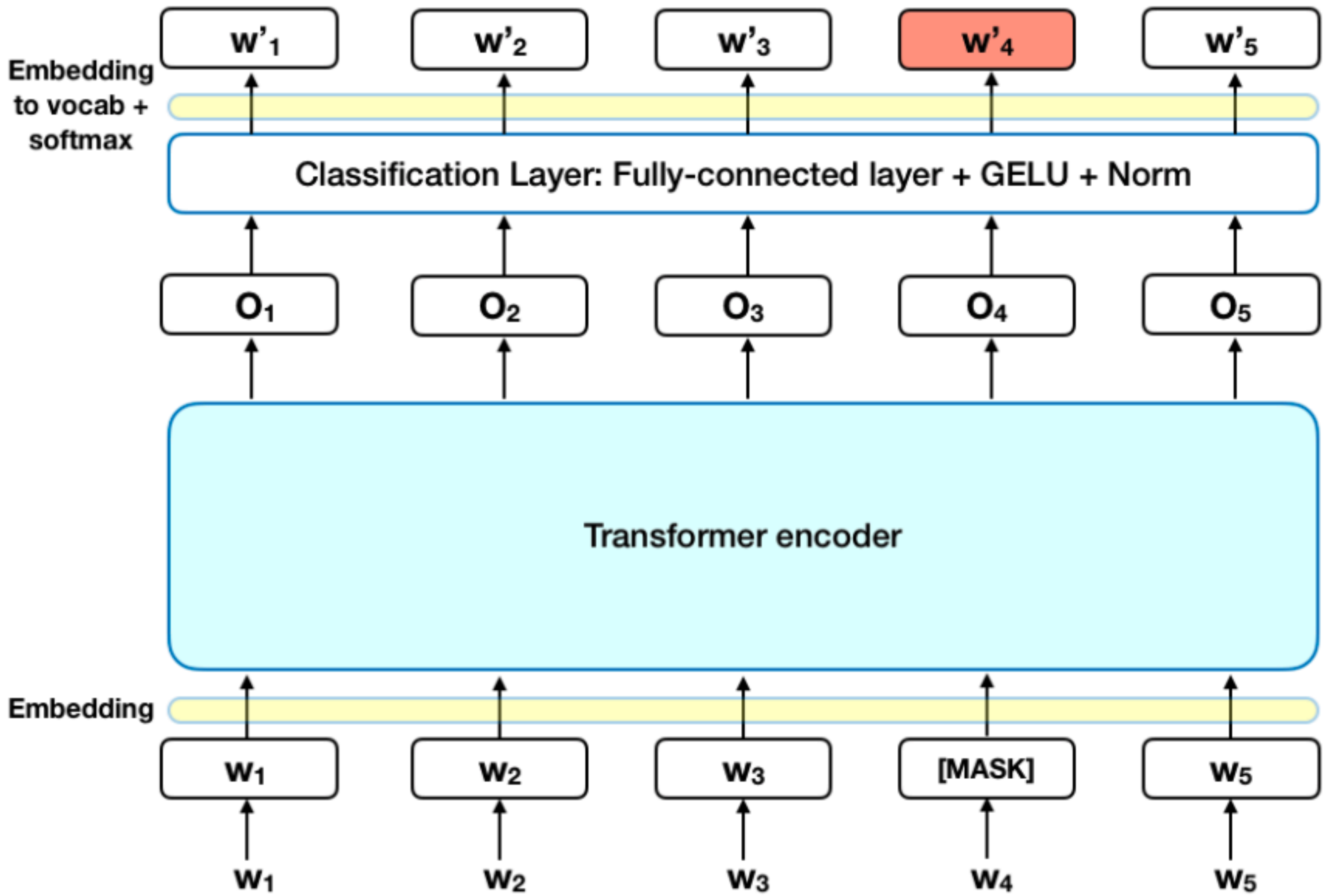
Masked LM (MLM)

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.

In technical terms, the prediction of the output words requires:

- Adding a classification layer on top of the encoder output.
- Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
- Calculating the probability of each word in the vocabulary with softmax.

Masked LM (MLM)



https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

The BERT loss function takes into consideration only the prediction of the masked values and ignores the prediction of the non-masked words.

Next Sentence Prediction (NSP)

In the BERT training process, the model **receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document**.

During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence.

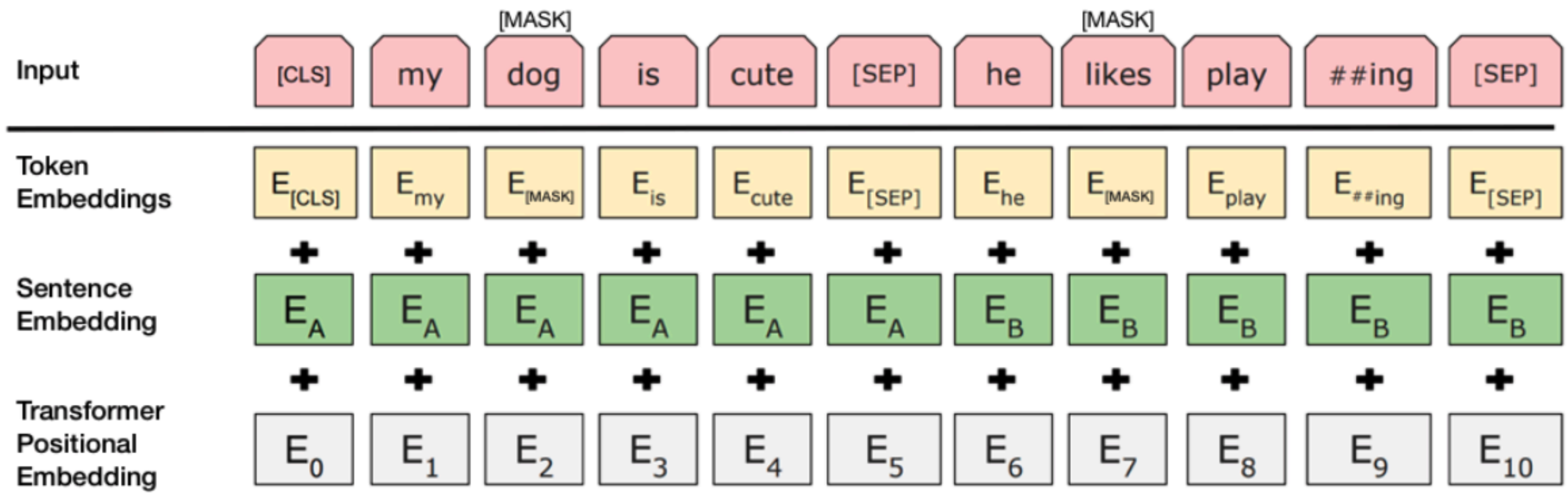
The assumption is that the random sentence will be disconnected from the first sentence.

Next Sentence Prediction (NSP)

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

- A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
- A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.

Next Sentence Prediction (NSP)



Source: BERT [Devlin et al., 2018], with modifications

Next Sentence Prediction (NSP)

To predict if the second sentence is indeed connected to the first, the following steps are performed:

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
- Calculating the probability of *IsNextSequence* with *softmax*.

When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

How to use BERT

BERT can be used for a wide variety of language tasks, while only adding a small layer to the core model:

- **Classification** tasks such as sentiment analysis are done similarly to Next Sentence classification, by adding a classification layer on top of the Transformer output for the [CLS] token.
- In **Question Answering** tasks (e.g. SQuAD v1.1), the software receives a question regarding a text sequence and is required to mark the answer in the sequence. Using BERT, a Q&A model can be trained by learning two extra vectors that mark the beginning and the end of the answer.
- In **Named Entity Recognition** (NER), the software receives a text sequence and is required to mark the various types of entities (Person, Organization, Date, etc) that appear in the text. Using BERT, a NER model can be trained by feeding the output vector of each token into a classification layer that predicts the NER label.