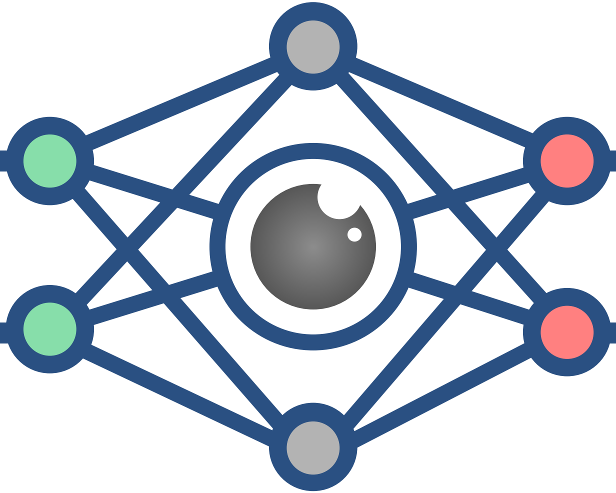
CS3485

Deep Learning for Computer Vision



Lec 20/21: Attention, Transformers and ChatGPT

Announcements

- Interesting application of stylegan: https://facemorph.me/



Announcements

- Mega quiz next Monday:
  - More questions: 40 to 50,
  - More time: 1hr,
  - More content: all of our course!
  - More weight: worth 4 regular quizzes!
- Reminder: lowest grade on regular quizzes will be dropped.
- Finally, let’s finish the previous lecture!

(Tentative) Lecture Roadmap

Basics of Deep Learning

- Intro to Computer Vision
- Linear Classifiers and Perceptron
- Multilayer Perceptron
- Optimization in Deep Learning
- Pytorch I – MLPs
- Pytorch II – Images and Regularization
- Convolutional Neural Networks
- Data Augmentation and Deep CNNs
- Transfer Learning and Residual Nets
- Inception Net and what CNNs learn
- Adversarial Examples and Self-supervision

Deep Learning and Computer Vision in Practice

- Intro to MLOps
- Intro to Object Detection
- Fast Object Detection
- Intro to Image Segmentation
- Applications of Detection and Segmentation
- Autoencoders
- Image Generation with GANs
- Advanced GANs
- The Attention Mechanism
- Transformers and ChatGPT
- Image Generation by Prompt
- Misc. Topics

Deep Learning for Language Modeling

- Besides Computer Vision, Deep Learning has had an (arguably larger) impact on Natural Language Processing:

  Natural Language Processing (NLP) is the study of how computers can gain high-level understanding from **language data**, such as text and speech.

- This impact is now mostly driven by the **Transformer Architecture**, proposed in a ground-breaking 2017 paper called Attention is All You Need.
- The transformer architecture is the basis for one of the most impactful deep learning solutions in the industry, **ChatGPT**, which we'll also cover here.
- To understand transformers, we start with **Attention**.

Retrieval in Databases

- The **Attention Mechanism** is the core operation in Transformers (analogously to how Convolution is core to CNNs), and emerged from the study of **Databases**.
- In their simplest form, databases are collections of **keys** (k) and **values** (v).
- *For example*, a database D might consist of a list of pairs names like

  {("Barclay", "Andrew"), ("Chen", "Li"), ("Barros", "Luis"), ("Dubois", "Nicole")},

  with the last name being the key and the first name being the value.
- When retrieving info from D, the **query** (q) for "Barros" returns the value "Luis", for example. We can call this **hard (or exact) retrieval**.
- If the entry ("Barros", "Luis") were not in D, we could do a couple of things:
  - Return "null" or "entry not found".
  - Return an approximate match, like "Andrew", since "Barros" is somewhat **similar** to "Barclay".
  - Return a combination of values summed according to a weight function (the most values for the most similar keys will weight more). We shall call this strategy **soft-retrieval**.

Add example with numbers/vectors

Attention

- Say $D = \{(k_1, v_1), (k_2, v_2), \ldots, (k_n, v_n)\}$ of keys k_i and values v_i. We can then define our softly retrieved value (or "Query Attention") of a query q according to D as:

$$\text{Attention}(q) = \sum_{i=1}^{n} \alpha(q, k_i) v_i$$

  where the weights* $\alpha(q, k_i)$ can be thought of as "how similar the query q is to the key k_i".
- This is called "**attention**" because the query is "paying particular attention" to the values whose keys are similar to it according to the function α.
- It is desirable that the weights are positive and sum to one, which can be accomplished via a **softmax** operation of the outputs of another similarity function $\alpha'(q, k_i)$:

$$\alpha(q, k_i) = \text{softmax}(\alpha'(q, k_i)) = \frac{\exp(\alpha'(q, k_i))}{\sum_{j=1}^{n} \exp(\alpha'(q, k_j))}$$

* In particular, if $\alpha(q, k_i) = 1$ only if $q = k_i$, and 0 otherwise, we get our traditional/hard database query.

Dot Product Attention

- If the query q and the key k_i are row vectors in d dimensions, an option for their similarity could be their dot product* $qk_i^\top$, or its scaled version**:

$$\alpha'(q, k_i) = qk_i^\top / \sqrt{d}$$

- The above operation, called **Scaled Dot Product Attention**, does what we expect:
  - It is largest if $q = k_i$ and lowest when they have opposite directions,
  - It moves smoothly between those extremes for the other values of q and k_i.
- Say we have a matrix $K \in \mathbb{R}^{n \times d}$ of the n keys, and a matrix $V \in \mathbb{R}^{n \times l}$ of the corresponding n values, each in l dimensions. Then the attention on a query $q \in \mathbb{R}^d$ is given by:

$$\text{Attention}(q) = \text{softmax}\left(\frac{qK^\top}{\sqrt{d}}\right) V$$

which is a vector in in l dimensions.

* Because they are row vectors, the transpose shows up on the second term in a dot product.

** The scaling is there to keep the order of magnitude of $\exp(\alpha'(q, k_i))$ and $\alpha(q, k_i)$ under control.

Attention and Masked Attention

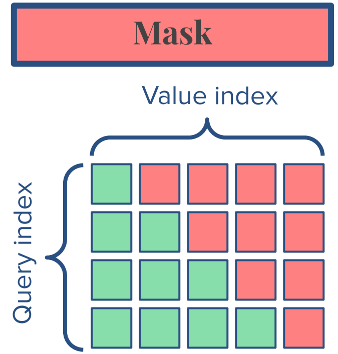
- Finally, say that we have m different queries, and place all of them as columns of a matrix $Q \in \mathbb{R}^{m \times d}$. Then, we can compute a matrix in $\mathbb{R}^{m \times l}$ the attention on all queries via:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

- Besides the attention as above there is the concept of **Masked Attention** where we force the queries to only attend to certain values, making the others "unreacheable".
- This is done by adding a mask $M \in \mathbb{R}^{m \times n}$ to the attention formula:

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top + M}{\sqrt{d}}\right)V$$

where M effectively encodes which queries can attend to which values (*as shown on the right example*).



Each of the 4 queries can attend only the value indices that are green.

An example of Attention

- Let's see review it all via an example. Say we have these keys and values, and one query:

$k_1 = \begin{bmatrix} 1 & 2 \end{bmatrix}$ $k_2 = \begin{bmatrix} 2 & 5 \end{bmatrix}$ $k_3 = \begin{bmatrix} 0 & 1 \end{bmatrix}$ $q_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$

$v_1 = \begin{bmatrix} 5 & 2 & 1 & 4 \end{bmatrix}$ $v_2 = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}$ $v_3 = \begin{bmatrix} 8 & 4 & 2 & 1 \end{bmatrix}$

- Our first step is to compute the inner products between the query and the keys:

$q_1 k_1^\top = \begin{bmatrix} 1 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 \\ 2 \end{bmatrix} = 3$ $q_1 k_2^\top = \begin{bmatrix} 1 & 1 \end{bmatrix} \bullet \begin{bmatrix} 2 \\ 5 \end{bmatrix} = 7$ $q_1 k_3^\top = \begin{bmatrix} 1 & 1 \end{bmatrix} \bullet \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1$

- Since $d = 2$ and noting that $1/\sqrt{2} \cong 0.7$, we compute the attention weights as:

$\alpha'(q_1, k_1) \cong 0.7 \times q_1 k_1^\top = 2.1$ $\alpha'(q_1, k_2) \cong 0.7 \times q_1 k_2^\top = 4.9$ $\alpha'(q_1, k_3) \cong 0.7 \times q_1 k_3^\top = 0.7$

$\alpha(q_1, k_1) = \text{softmax}(\alpha'(q_1, k_1)) = \mathbf{0.06}$ $\alpha(q_1, k_2) = \text{softmax}(\alpha'(q_1, k_2)) = \mathbf{0.93}$ $\alpha(q_1, k_3) = \text{softmax}(\alpha'(q_1, k_3)) = \mathbf{0.01}$

* A reminder about the softmax function: $\text{softmax}(\alpha'(q_1, k_i)) = \exp(\alpha'(q_1, k_i))/[\exp(\alpha'(q_1, k_1)) + \exp(\alpha'(q_1, k_2)) + \exp(\alpha'(q_1, k_3))]$.

An example of Attention

- The attention on q_1 is computed via $\text{Attention}(q_1) = \alpha(q_1, k_1) \times v_1 + \alpha(q_1, k_2) \times v_2 + \alpha(q_1, k_3) \times v_3$, which means:

$$\text{Attention}(q_1) = 0.06 \times \begin{bmatrix} 5 & 2 & 1 & 4 \end{bmatrix} + 0.93 \times \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} + 0.01 \times \begin{bmatrix} 8 & 4 & 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.38 & 1.09 & 1.01 & 1.18 \end{bmatrix}$$

- This whole process can be written using the matrices of keys K (a stacking of k_1, k_2 and k_3) and values V (a stacking of v_1, v_2 and v_3) and usual matrix multiplication:

$$\text{Attention}(q_1) = \text{softmax}\left[\underbrace{0.7}_{1/\sqrt{d}} \times \underbrace{\begin{bmatrix} 1 & 1 \end{bmatrix}}_{q_1} \bullet \underbrace{\begin{bmatrix} 1 & 2 & 0 \\ 2 & 5 & 1 \end{bmatrix}}_{K^\top}\right] \bullet \underbrace{\begin{bmatrix} 5 & 2 & 1 & 4 \\ 0 & 1 & 0 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}}_{V} = \begin{bmatrix} 0.38 & 1.09 & 1.01 & 1.18 \end{bmatrix}$$

An example of Attention

- Say we now have 5 queries instead of just one:

$$q_1 = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad q_2 = \begin{bmatrix} 0 & 1 \end{bmatrix} \quad q_3 = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad q_4 = \begin{bmatrix} 2 & 2 \end{bmatrix} \quad q_5 = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

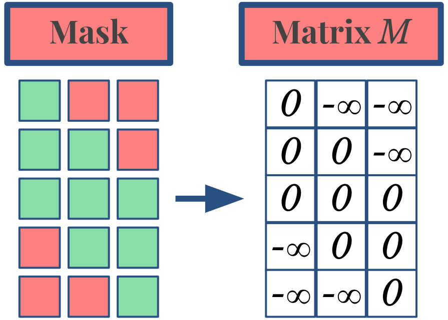
- If one creates the matrix Q by stacking the queries, one can easily use the attention formula to compute all the query attentions using matrix multiplications:

$$\text{Attention}(Q, K, V) = \text{softmax}\left[\underbrace{0.7}_{1/\sqrt{d}} \times \underbrace{\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 2 & 2 \\ 1 & 2 \end{bmatrix}}_{Q} \bullet \underbrace{\begin{bmatrix} 1 & 2 & 0 \\ 2 & 5 & 1 \end{bmatrix}}_{K^{\top}}\right] \bullet \underbrace{\begin{bmatrix} 5 & 2 & 1 & 4 \\ 0 & 1 & 0 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}}_{V} = \begin{bmatrix} 0.38 & 1.09 & 1.01 & 1.18 \\ 0.93 & 1.26 & 0.20 & 1.31 \\ 2.55 & 1.70 & 0.56 & 1.85 \\ 0.02 & 1.00 & 0.00 & 1.01 \\ 0.04 & 1.01 & 0.00 & 1.02 \end{bmatrix}$$

Note that the resulting attention is kind of "encoding" each one of the queries.

An example of Masked Attention

- Now, say we'd like to apply the mask on the right.
- This mask says that q_1 is only allowed to attend to v_1; q_2 can attend to either v_1 or v_2; q_3 can attend to all values; and so on.
- From that we create a matrix M, where the "not allowed" locations are said to be $-\infty$, while the allowed ones are zero.
- Here is how the Masked attention computation looks like using the mask M now:



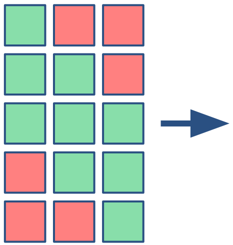
0	-∞	-∞
0	0	-∞
0	0	0
-∞	0	0
-∞	-∞	0

$$\text{Attention}(Q, K, V) = \text{softmax}\left[0.7 \times \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 2 & 2 \\ 1 & 2 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 & 0 \\ 2 & 5 & 1 \end{bmatrix} + 0.7 \times \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \\ -\infty & 0 & 0 \\ -\infty & -\infty & 0 \end{bmatrix}\right] \bullet \begin{bmatrix} 5 & 2 & 1 & 4 \\ 0 & 1 & 0 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}$$

An example of Masked Attention

- Now, say we'd like to apply the mask on the right.
- This mask says that q_1 is only allowed to attend to v_1; q_2 can attend to either v_1 or v_2; q_3 can attend to all values; and so on.
- From that we create a matrix M, where the "not allowed" locations are said to be $-\infty$, while the allowed ones are zero.
- Here is how the Masked attention computation looks like using the mask M now:

Mask



Matrix M

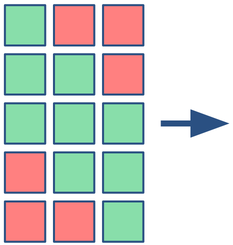
0	-∞	-∞
0	0	-∞
0	0	0
-∞	0	0
-∞	-∞	0

$$\text{Attention}(Q, K, V) = \text{softmax}\left[0.7 \times \begin{bmatrix} 3 & 7 & 1 \\ 2 & 5 & 1 \\ 1 & 2 & 0 \\ 6 & 14 & 2 \\ 5 & 12 & 2 \end{bmatrix} + 0.7 \times \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \\ -\infty & 0 & 0 \\ -\infty & -\infty & 0 \end{bmatrix}\right] \bullet \begin{bmatrix} 5 & 2 & 1 & 4 \\ 0 & 1 & 0 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}$$

An example of Masked Attention

- Now, say we'd like to apply the mask on the right.
- This mask says that q_1 is only allowed to attend to v_1; q_2 can attend to either v_1 or v_2; q_3 can attend to all values; and so on.
- From that we create a matrix M, where the "not allowed" locations are said to be -∞, while the allowed ones are zero.
- Here is how the Masked attention computation looks like using the mask M now:

Mask



Matrix M

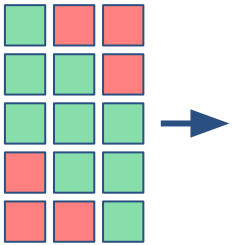
0	-∞	-∞
0	0	-∞
0	0	0
-∞	0	0
-∞	-∞	0

$$\text{Attention}(Q, K, V) = \text{softmax}\left[0.7 \times \begin{bmatrix} 3 & -\infty & -\infty \\ 2 & 5 & -\infty \\ 1 & 2 & 0 \\ -\infty & 14 & 2 \\ -\infty & -\infty & 2 \end{bmatrix}\right] \bullet \begin{bmatrix} 5 & 2 & 1 & 4 \\ 0 & 1 & 0 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}$$

An example of Masked Attention

- Now, say we'd like to apply the mask on the right.
- This mask says that q_1 is only allowed to attend to v_1; q_2 can attend to either v_1 or v_2; q_3 can attend to all values; and so on.
- From that we create a matrix M, where the "not allowed" locations are said to be -∞, while the allowed ones are zero.
- Here is how the Masked attention computation looks like using the mask M now:

Mask



Matrix M

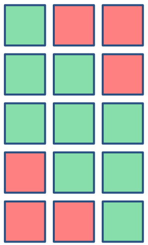
0	-∞	-∞
0	0	-∞
0	0	0
-∞	0	0
-∞	-∞	0

$$\text{Attention}(Q, K, V) = \begin{bmatrix} 1 & 0 & 0 \\ 0.11 & 0.89 & 0 \\ 0.29 & 0.57 & 0.14 \\ 0 & 0.99 & 0.01 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 5 & 2 & 1 & 4 \\ 0 & 1 & 0 & 1 \\ 8 & 4 & 2 & 1 \end{bmatrix}$$

An example of Masked Attention

- Now, say we'd like to apply the mask on the right.
- This mask says that q_1 is only allowed to attend to v_1; q_2 can attend to either v_1 or v_2; q_3 can attend to all values; and so on.
- From that we create a matrix M, where the "not allowed" locations are said to be $-\infty$, while the allowed ones are zero.
- Here is how the Masked attention computation looks like using the mask M now:

Mask



Matrix M

0	$-\infty$	$-\infty$
0	0	$-\infty$
0	0	0
$-\infty$	0	0
$-\infty$	$-\infty$	0

Attention(Q, K, V) =

1	0	0
0.11	0.89	0
0.29	0.57	0.14
0	0.99	0.01
0	0	1

•

5	2	1	4
0	1	0	1
8	4	2	1

=

5	2	1	4
0.54	1.11	1.11	1.31
2.55	1.70	0.56	1.85
0.08	1.03	0.02	1.00
8	4	2	1

Note that, since q_1 and q_5 can only attend to v_1 and v_3, resp., their attention is exactly them.

Multi-head Attention

- Finally, we can compute what is called **Multi-head Attention**.
- Say we have h triples of matrices W_i^q, W_i^k, W_i^v (like linear layers in an MLP) one for each one of the query, key and value matrices; and compute an **attention head** as:

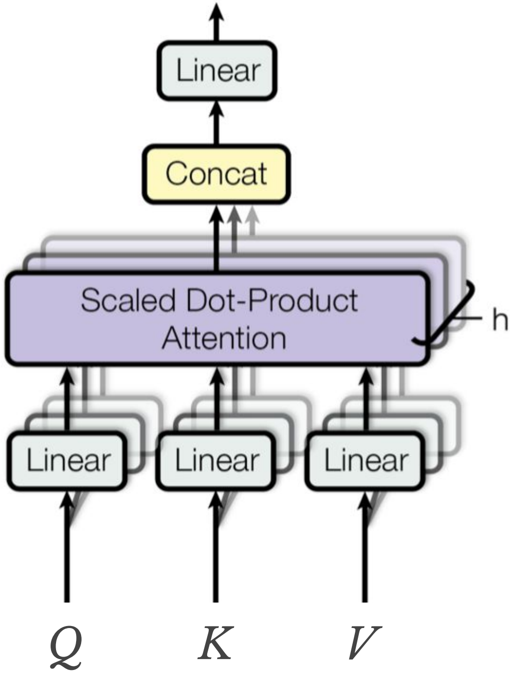
$$\text{Head}_i = \text{Attention}(W_i^q Q, W_i^k K, W_i^v V)$$

Then, we concatenate the heads and multiply the result by another matrix/linear layer W_0 to get:

$$\text{MultiHead}(Q, K, V) = W_0[\text{Head}_1, \text{Head}_2, \ldots, \text{Head}_h]$$

- As we'll see later, the goal of the multihead attention is to learn different ways the keys can attend to the input queries, similarly to how different filters in a ConvLayer can learn different image features.

Multihead Attention Module



Exercise (*in pairs*)

- Try it yourself! Say your database has the following keys and values:

  $k_1 = \begin{bmatrix} 1 & 2 \end{bmatrix}$ $k_2 = \begin{bmatrix} -1 & 1 \end{bmatrix}$ $k_3 = \begin{bmatrix} 0 & 1 \end{bmatrix}$

  $v_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$ $v_2 = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$ $v_3 = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$

  and you have the following queries:

  $q_1 = \begin{bmatrix} 0 & 1 \end{bmatrix}$ $q_2 = \begin{bmatrix} 1 & 1 \end{bmatrix}$

  what is the attention of these queries? You can use $1/\sqrt{2} \cong 0.7$ for simplicity and use this website to compute softmax.
- Say $K \in \mathbb{R}^{n \times d}$, $V \in \mathbb{R}^{n \times l}$ and $Q \in \mathbb{R}^{m \times d}$, what is the worst case memory complexity of computing $\text{Attention}(Q, K, V)$?

Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

The trained transformer takes in an input sentence and outputs its translation.

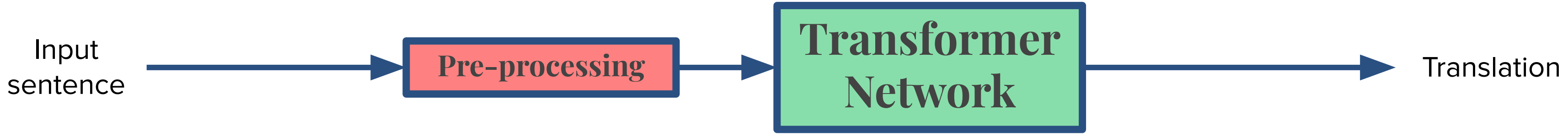
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

Before the input goes into the transformer, it has to go through some preprocessing.



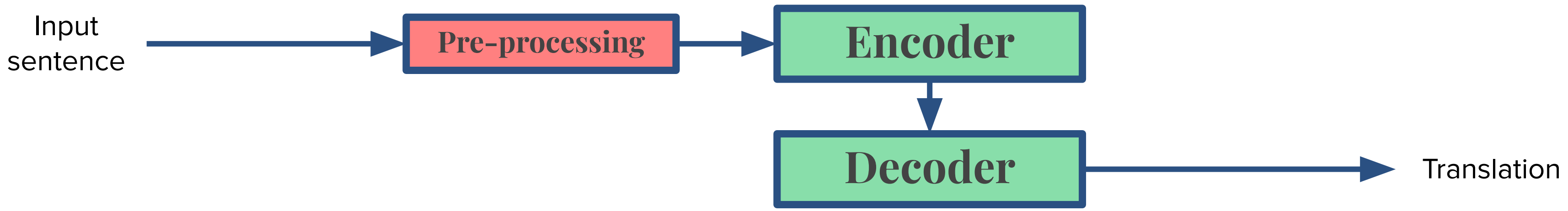
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

The transformer for MT is an encoder/decoder network, so the input first goes to an encoder then to a decoder.



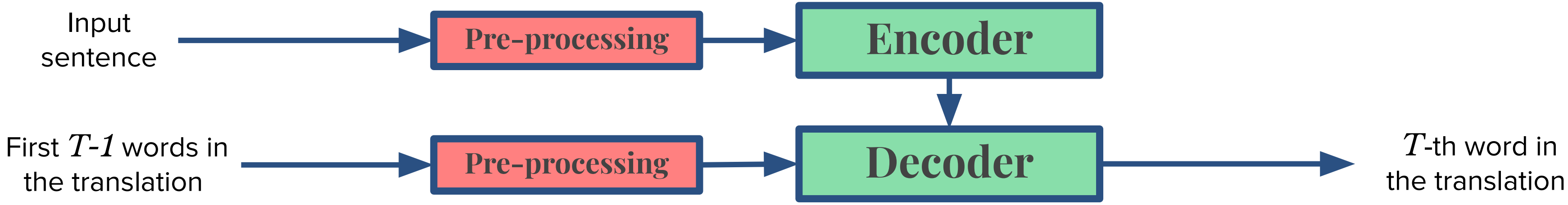
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

One pass through the transformer only generates one (translated) word at a time, so we also input the previously translated words to the decoder after processing them.



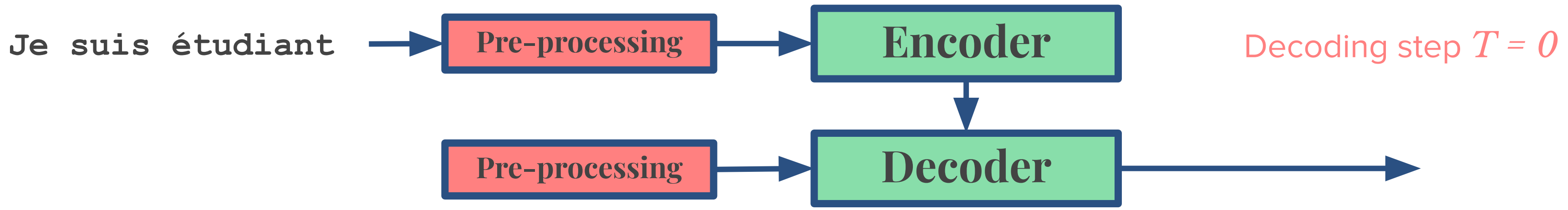
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

An example: say we want to translate a sentence in French ("Je suis étudiant") to English ("I am a student"). A trained transformer will behave as follows.



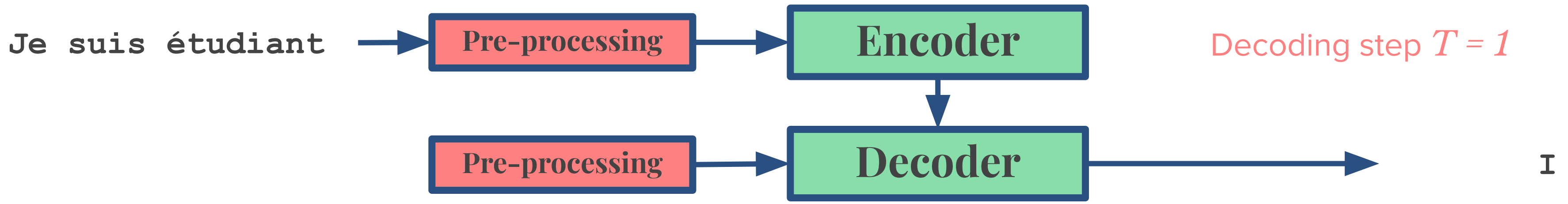
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

The first pass will generate the most likely word for the whole translation ("I" in this case)



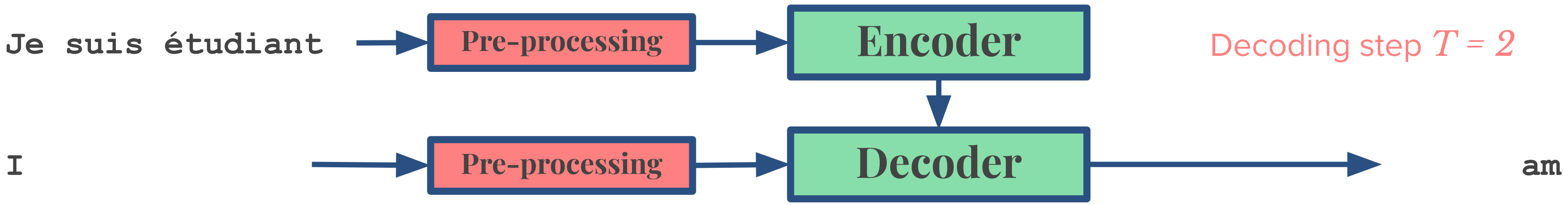
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

After that, "I" becomes an input the decoder and now the transformer has to guess the most likely word for the translation of "Je suis étudiant" after "I" (which is "am" in this case)



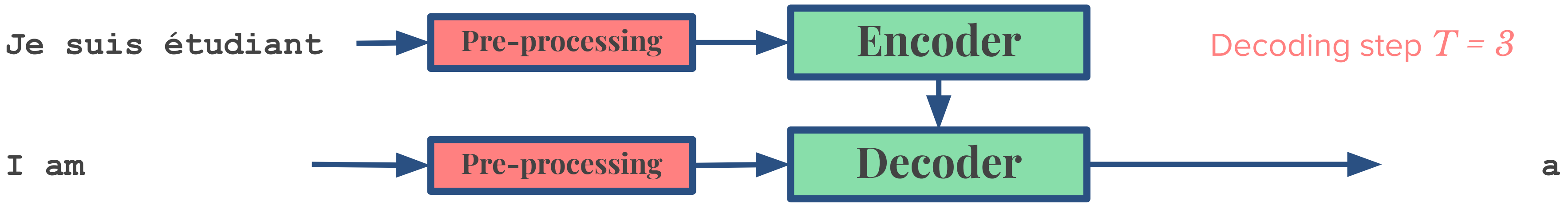
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

The process now repeats for the particle "I am".



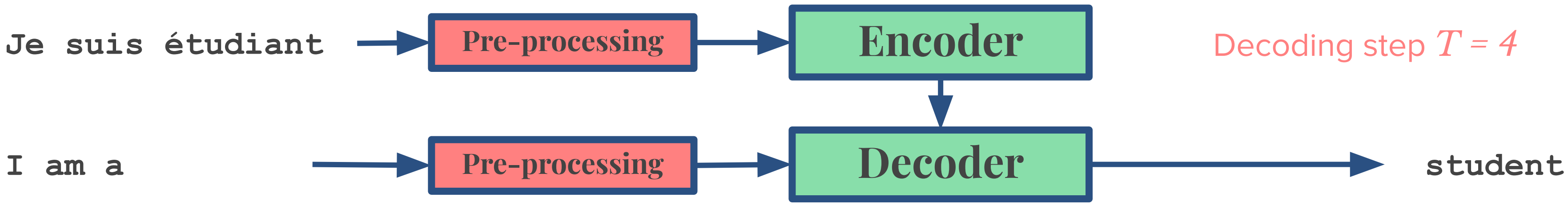
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

And for "I am a".



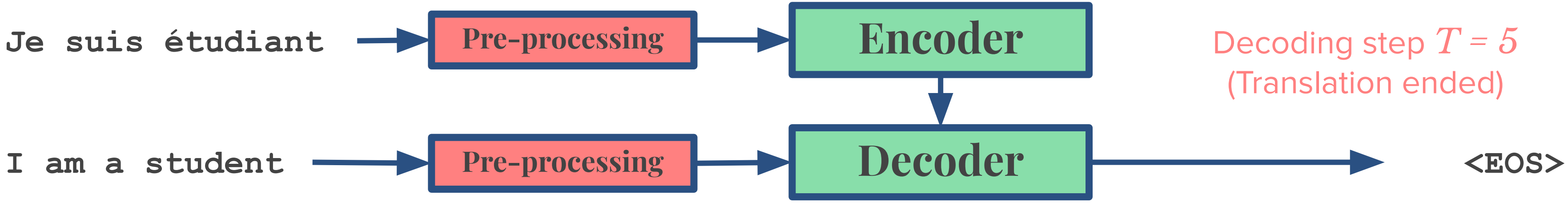
Machine Translation

- We saw that, if we have queries, keys and values, we can find the (Multi-head) attention of each query. Now, let's make it practical and see how it can be used in NLP!
- We'll use the task of Machine Translation to exemplify its usage:

Machine translation (MT) is the task of automatically translating a sentence in a source language to a different target language.

- It was to solve MT that the first Transformer was developed! Here's how it broadly works:

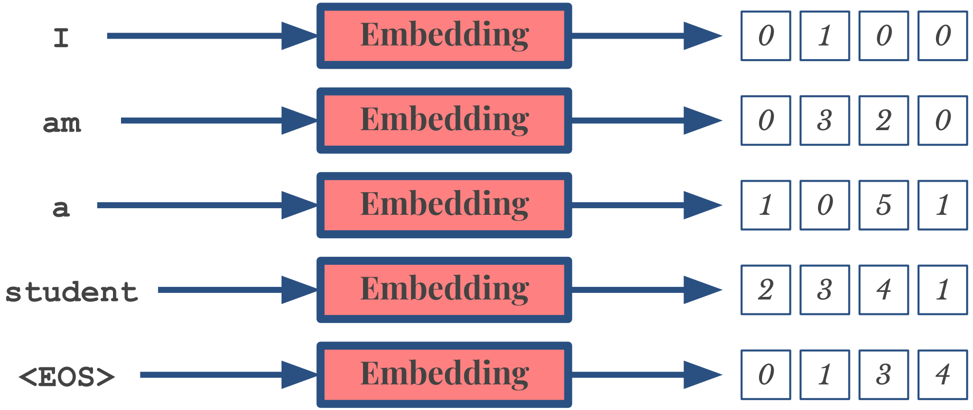
If the transformer thinks the translation is over, it will output a special word to define the End of Sentence (EOS) and finish the translation.



Decoding step $T = 5$
(Translation ended)

Preprocessing the text: token embedding

- Before we see the transformer in action, we need to see how it first preprocesses the data (text sentences).
- As is the case in most NLP applications, we first transform each part (also referred to as "**token**") of each sentence into numerical vectors using an **embedding algorithm***.

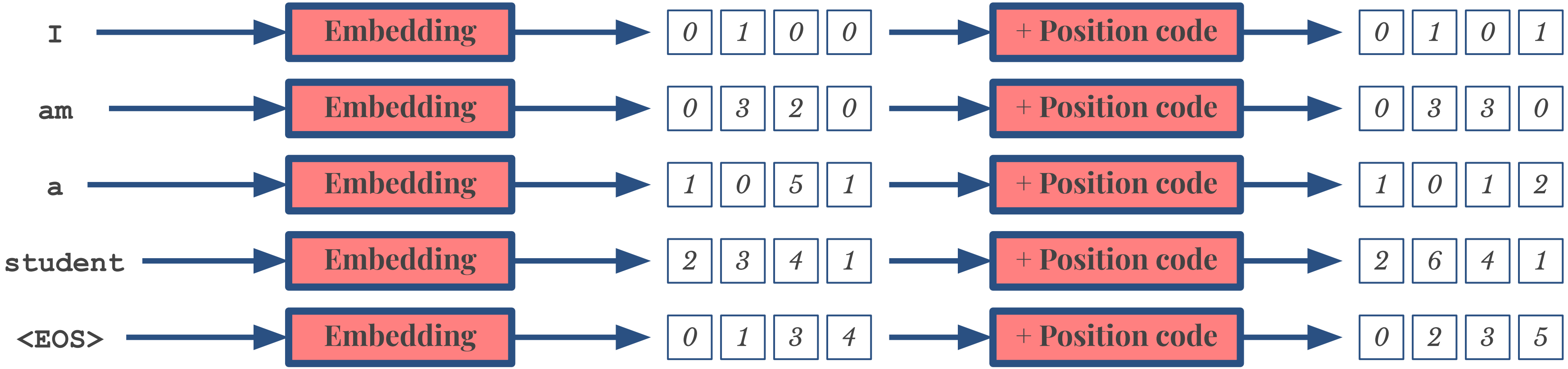


- The embeddings are computed to capture the **semantic meaning** of each of the sentence's **tokens**.
- Tokens usually refer to individual words, but there special tokens for the end and beginning of sentences, punctuation signs, certain characters, etc.

* We won't cover any embedding algorithm in this course, but a common method is the Word2Vec algorithm. More info on it here.

Preprocessing the text: positional encoding

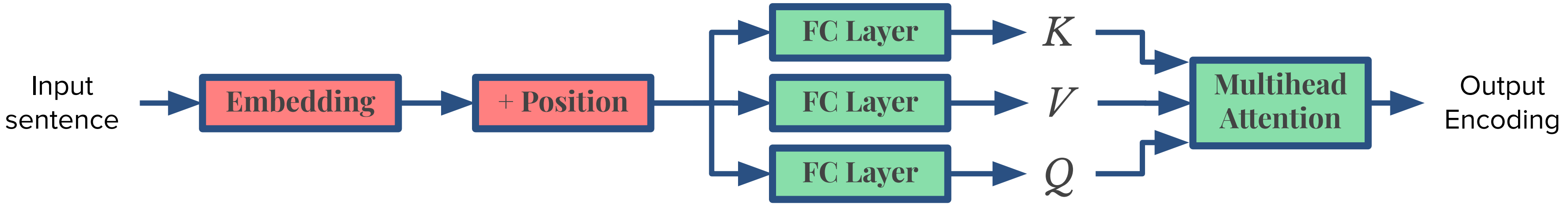
- The embeddings, however, don't carry any data on the tokens' **positions** in the sentence.
- For that reason, we sum each embedding vector to another vector that encodes its position (*1*st, *2*nd, *3*rd,...) in the sentence. This is called **Positional Encoding**.
- These codes can be binary representations of the positions (*0001*, *0010*, *0011*, *0100*...)*.



* More usually, however, people use sine/cosine functions to encode positions. Read more here.

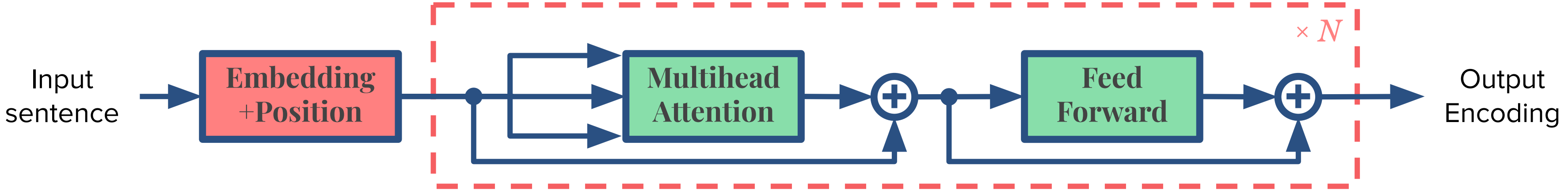
The Encoder of a Transformer

- In the Transformer architecture, our first objective is to learn a network that uses the attention mechanism to **encode** sentences (*later we'll see how to decode them*).
- To do so, we'd like to learn how the tokens in a given sentence **attend to each other**, using a process call **self-attention**:
    - We'll create a network that sends the position encoded word embeddings through **three learnable layers** that output query, key and value matrices.
    - These matrices will then be input to a Multihead Attention module, from which we get an encoding of the input sentence.
    - For example, if we have 512 input embeddings, there will be 512 encoded output vectors.



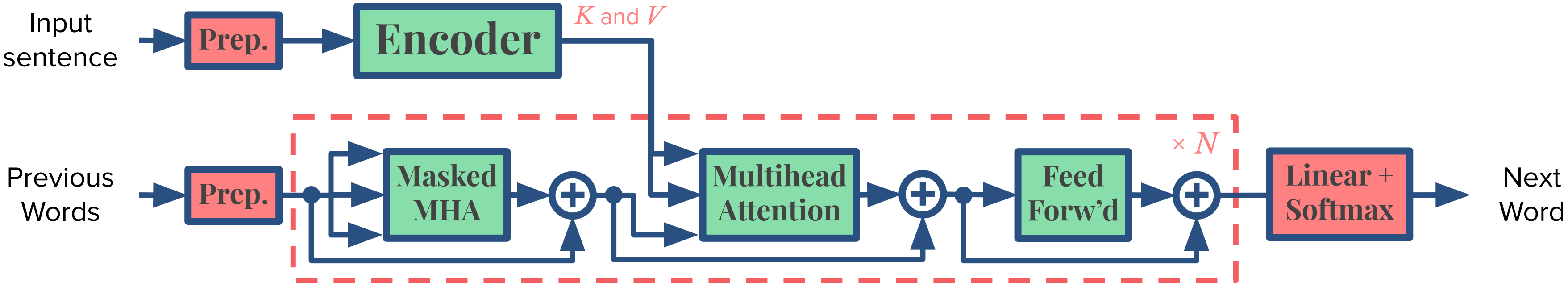
The Encoder of a Transformer

- The creators of the Transformer used the previous encoding network to design a more complex encoder as follows:
  - They added **residual connections** around the initial FC layers and the MHA module.
  - They introduced a **feed forward network** (an MLP) after the attention module and surrounded it with a residual connection.
  - They applied N **instances** of this module sequentially. In this way, if the input of each instance is 512 vectors, the encoded output it's again 512 vectors to be fed to the next instance.
- After each residual residual connection, they also added a **layer normalization** step, which works similarly to batch normalization.



The Decoder of a Transformer

- The decoder part works similarly to the encoder with some major differences:
  - It first goes by a **Masked Multihead Attention**, in order to prevent the network from learning attention scores for future words in the sentence (we only use past words to predict the one).
  - After that, we have the usual Multihead Attention Module, where the **key and value matrices are the same as the encoded input**, while the queries are provided by the Masked MHA.
  - At the end, result goes through a **linear + softmax stage** (like an MLP) to produce a one-hot predictor vector over all possible words in the target vocabulary.

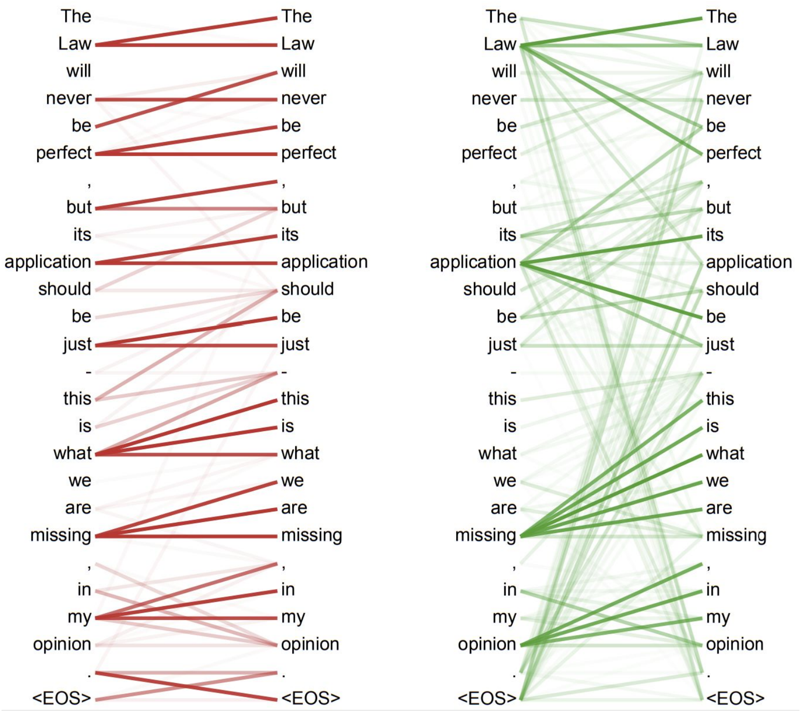


Training a Transformer and Visualizing Attention

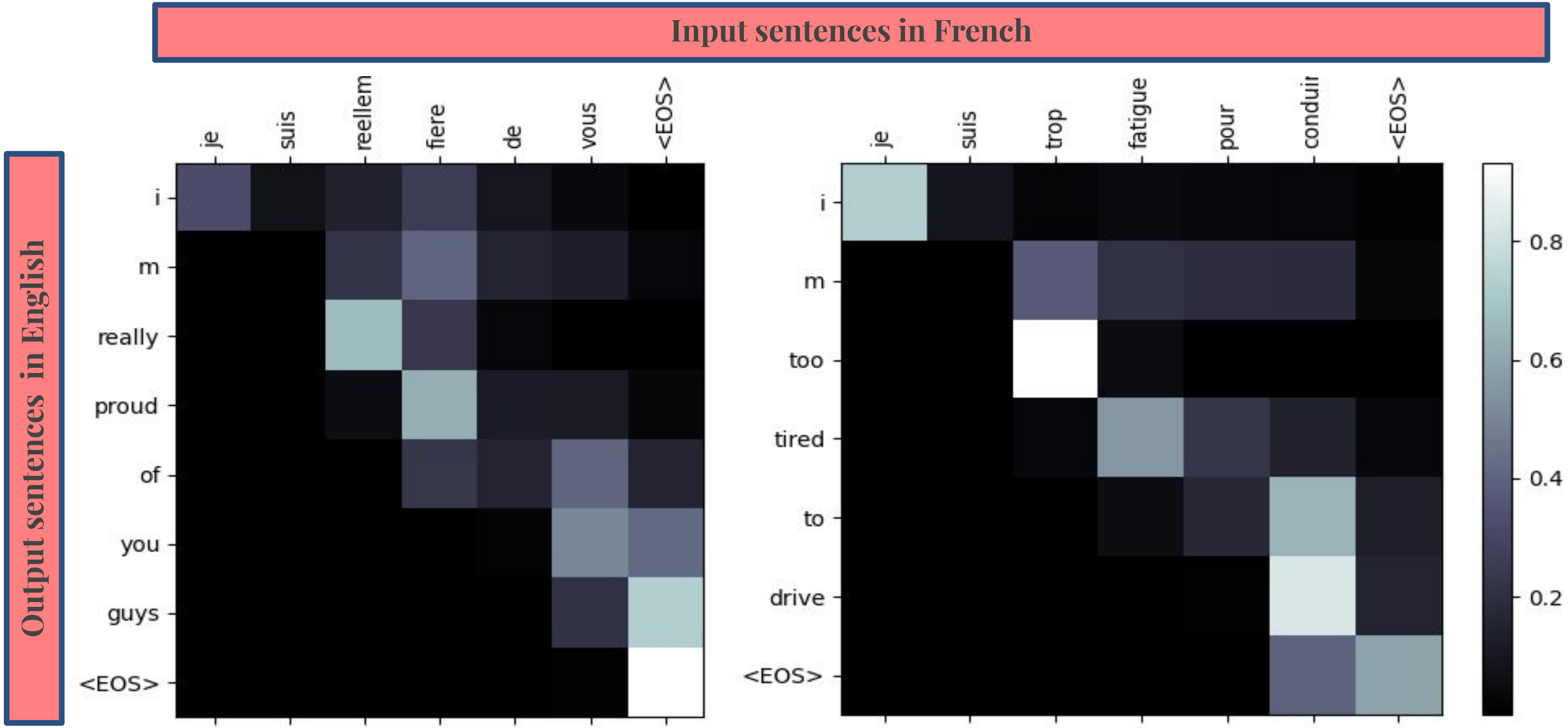
- Training a transformer for MT is simple*:
  - Input a sentence in the original language to the encoder.
  - Predict the translation of that sentence word by word and check if it matches the GT data.
- The original transformer was trained in a English-French dataset consisting of 36 million sentences with vocabulary of around 32000 tokens using 8 GPUs for 3.5 days.
- With the trained transformer, we can **visualize the attention weights** and see how different heads learn different word relationships!

* Here is a [very good video]() that explains this step-by-step of Transformers in more detail with nicer animations!

Self-attention visualization in two different heads



Visualizing Cross-Attention



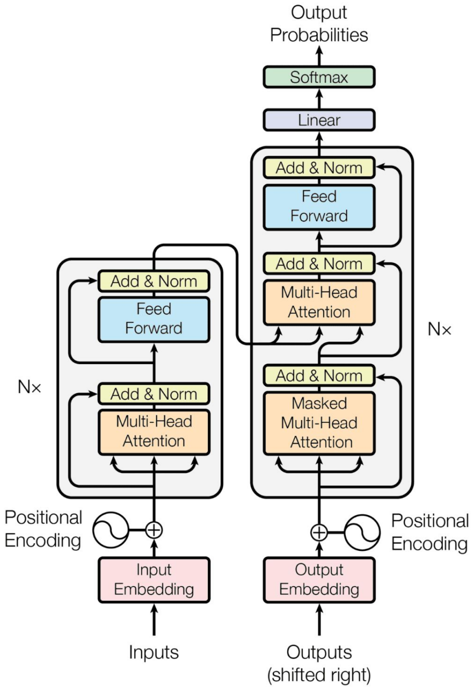
Applications of Transformers

- The architecture we just saw shook the world of Deep Learning when it came out, outperforming many other CNN based methods in many AI tasks, not just MT.
- The success of Attention based networks was so big that led to one of the creators of Deep Learning to make this claim:

Yoshua Bengio: Attention is a core ingredient of 'conscious' AI

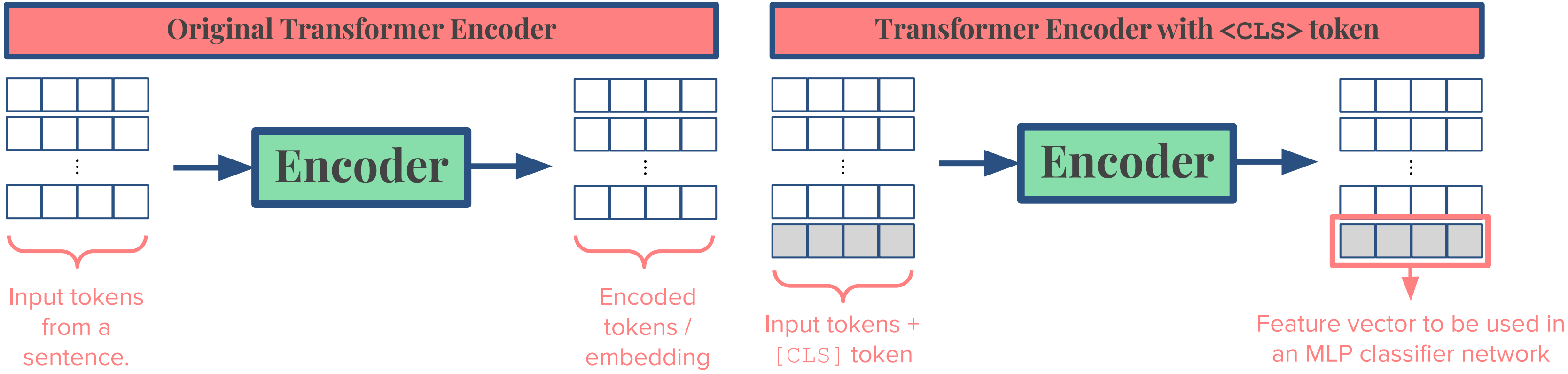
- A lot of the subsequent research in the area saw the transformer architecture **drop either the encoder or decoder** so it could be useful in other tasks.
- Here, we'll see what you can do when you keep one of them.

Original Transformer Diagram



Keeping the Encoder: Text Classification

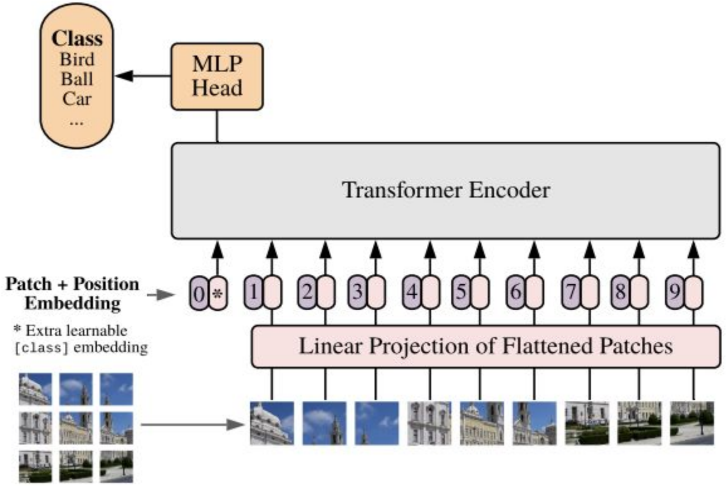
- One popular way researchers use the transformer architecture is in **text classification**.
- To do that, one typical way is to turn input sentences into feature vectors to be used in a MLP classifier.
- Starting with **BERT** (Bidirectional Encoder Representations from Transformer), proposed in 2018, researchers started adding a special token, called <CLS>, to each sentence and used its transformer encoding as a feature vector corresponding to the whole sentence.



Keeping the Encoder: Vision Transformers

- The same idea used for text classification, has been applied to image classification as well.
- Published in 2020, the **Vision Transformer** (ViT) architecture was introduced.
- The training of a ViT has the following steps:
  - It first extracts square patches from the input image and uses them as tokens.
  - These image patches are flattened, go through a Linear Layer and then get the positional code.
  - ViT then adds a <CLS> token to the preprocessed tokens.
  - The tokens go through a transformer encoder.
  - Only the encoded <CLS> goes through an MLP that classifies the whole image.

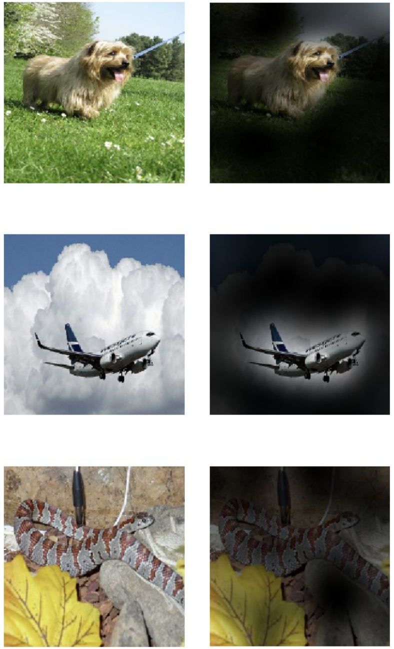
Vision Transformer (ViT) + Model Details



Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Keeping the Encoder: Vision Transformers

What the <cls> token attends to in an image



- When trained, we can understand what ViT learned using the attention weights it found, making it less "black-boxy" than CNNs.
- In practice, ViTs also out performed many CNNs in classification. In the ImageNet challenge, it reaches over 90% on Top-1 acc.
- The success of ViT lead to its application in other vision tasks such as object detection, image segmentation and many more!

Transformers for Obj. Detection (DETR)

Transformers for Segmentation (SegFormer)

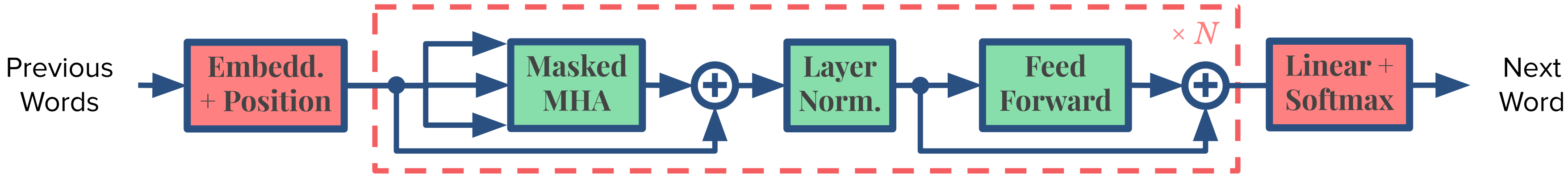
Keeping the Decoder: Text Generation

- The most important task in NLP is **text generation**.
- Much like with images, the goal here is to **artificially create realistic sounding text**, as if it had been a person who wrote it.
- Despite years of attempts by many algorithms, it was with a transformer algorithm, called **Generative Pre-trained Transformer (GPT)**, whose first version was launched in Jun. 2018 by OpenAI, that this task started to get realistically solved.
- In particular, a conversational version of GPT 3.5, called **ChatGPT**, launched in Nov. 2022, took the world by storm and set a milestone in achieving high quality text generation and AI conversation.

ChatGPT logo (with OpenAI long inside)

Keeping the Decoder: Text Generation

- As opposed to BERT, the GPT approach keeps the decoder of the original transformer.
- Since there is no encoder, GPT only relies on repeating the masked attention + feed forward architecture (N = 12 modules for GPT 1) from the transformer decoder:



- Training GPT is simple: given a dataset of sentences/texts, GPT goes through each sentence and tries to **predict the next word** in them given the previous words as inputs.
- This process is called **unsupervised pre-training** (hence the “P” in GPT).
- What is impressive about GPT is the amount parameters in it and of data used in training:
  - Dataset (BookCorpus): 4.5 GB of text, from 7000 unpublished books of various genres.
  - Training (117 million parameters): 30 days on 8 then high end GPUs.

Keeping the Decoder: Text Generation

- The first GPT's size, however, pales compared its follow-up versions:
  - **GPT 2 (Released in Feb. 2019)**: *48* decoder modules / ***1.5* billion parameters**. Trained on WebText dataset (*40* GB of text from *45* million Reddit webpages).
  - **GPT 3 (Released in May 2020)**: *96* decoder modules / ***175* billion parameters.** Trained on CommonCrawl dataset (570 GB of web content) + WebText + English Wikipedia + two books corpora (Books1 and Books2).
  - **GPT 3.5 (Released in Mar. 2022)**: Undisclosed architecture (estimated to also have ***175* billion** parameters) and training data.
  - **GPT 4 (Released in Mar. 2023)**: Undisclosed architecture (estimated to have ***1* trillion** learnable parameters) and training data.

Keeping the Decoder: Text Generation

- ChatGPT uses GPT to create a **conversational engine**, where a user can “provide” parts of the text generation and the neural network provides the next sentences.
- To do so, ChatGPT has a special End of Question token (`<EOQ>`) that signs that this is from where the GPT engine should generate text by itself.

- To make ChatGPT’s (and GPT 3’s) predictions more precise and realistic, OpenAI made use of **Reinforcement Learning from Human Feedback (RLHF)** along with pre-training:
    a. Human annotators are tasked at ranking various ChatGPT answers for certain questions.
    b. The ranks are fed into a reinforcement learning algorithm that learns the agent's policy.
    c. The agent collects more data, and the feedback from human experts is used to refine the agent's policy

Keeping the Decoder: Text Generation

- ChatGPT uses GPT to create a **conversational engine**, where a user can “provide” parts of the text generation and the neural network provides the next sentences.
- To do so, ChatGPT has a special End of Question token (`<EOQ>`) that signs that this is from where the GPT engine should generate text by itself.

- To make ChatGPT’s (and GPT 3’s) predictions more precise and realistic, OpenAI made use of **Reinforcement Learning from Human Feedback (RLHF)** along with pre-training:
  a. Human annotators are tasked at ranking various ChatGPT answers for certain questions.
  b. The ranks are fed into a reinforcement learning algorithm that learns the agent's policy.
  c. The agent collects more data, and the feedback from human experts is used to refine the agent's policy

Keeping the Decoder: Text Generation

- ChatGPT uses GPT to create a **conversational engine**, where a user can “provide” parts of the text generation and the neural network provides the next sentences.
- To do so, ChatGPT has a special End of Question token (`<EOQ>`) that signs that this is from where the GPT engine should generate text by itself.

- To make ChatGPT’s (and GPT 3’s) predictions more precise and realistic, OpenAI made use of **Reinforcement Learning from Human Feedback (RLHF)** along with pre-training:
  a. Human annotators are tasked at ranking various ChatGPT answers for certain questions.
  b. The ranks are fed into a reinforcement learning algorithm that learns the agent's policy.
  c. The agent collects more data, and the feedback from human experts is used to refine the agent's policy

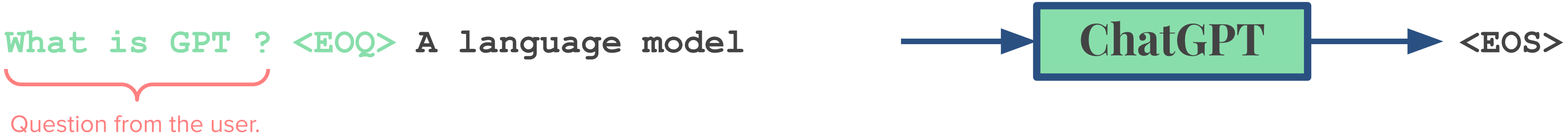
Keeping the Decoder: Text Generation

- ChatGPT uses GPT to create a **conversational engine**, where a user can “provide” parts of the text generation and the neural network provides the next sentences.
- To do so, ChatGPT has a special End of Question token (`<EOQ>`) that signs that this is from where the GPT engine should generate text by itself.

- To make ChatGPT’s (and GPT 3’s) predictions more precise and realistic, OpenAI made use of **Reinforcement Learning from Human Feedback (RLHF)** along with pre-training:
  a. Human annotators are tasked at ranking various ChatGPT answers for certain questions.
  b. The ranks are fed into a reinforcement learning algorithm that learns the agent's policy.
  c. The agent collects more data, and the feedback from human experts is used to refine the agent's policy

Keeping the Decoder: Text Generation

- ChatGPT uses GPT to create a **conversational engine**, where a user can “provide” parts of the text generation and the neural network provides the next sentences.
- To do so, ChatGPT has a special End of Question token (`<EOQ>`) that signs that this is from where the GPT engine should generate text by itself.



- To make ChatGPT’s (and GPT 3’s) predictions more precise and realistic, OpenAI made use of **Reinforcement Learning from Human Feedback (RLHF)** along with pre-training:
    a. Human annotators are tasked at ranking various ChatGPT answers for certain questions.
    b. The ranks are fed into a reinforcement learning algorithm that learns the agent's policy.
    c. The agent collects more data, and the feedback from human experts is used to refine the agent's policy

Amazing Achievements of ChatGPT

- ChatGPT (and GPT4) achieved some impressive achievements (which you probably already know about):
  - ChatGPT can **create resumes**, **cover letters**, and **LinkedIn profiles**,
  - It can write annual HR self-reflections and accomplishments,
  - ChatGPT can generate **essays**, **literary parodies**, and **programming answers**,
  - It can blend **actual facts with made-up ones in a biography** of a public figure or **cite plausible scientific references** for papers that were never written,
  - ChatGPT can hold **engaging discussions**, **respond to inquiries**, and **create original writings** like stories, poems, and essays,
  - **Write the text you are seeing in this slide!**

GPT4 As introduced by OpenAI

How popular is ChatGPT?

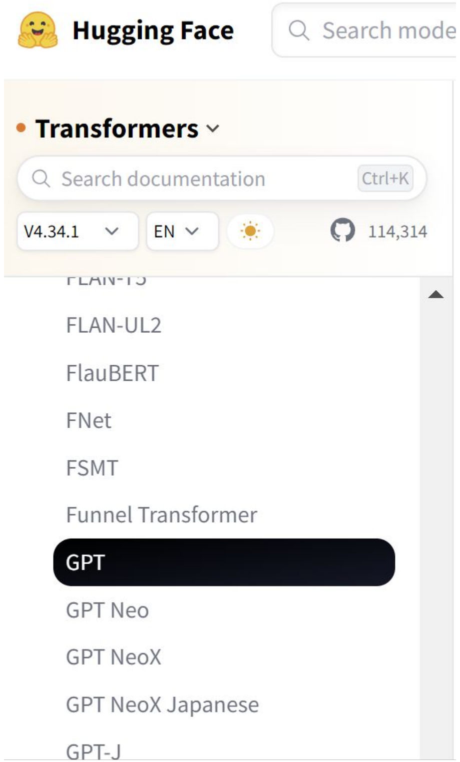
- Some interesting stats about ChatGPT (as of Oct. 2023, source and more stats: here):
  a. ChatGPT was launched on November 30, 2022 and crossed **1 million users in just 5 days** of launch (Instagram took 75 days) and had **100 million active users** by January 2023.
  b. It has **180.5 million** users, generates **$80 million/month revenue**, and crossed over **10 billion** all-time visits in 2023. In September 2024 alone, ChatGPT got **3.1 billion website** visits
  c. OpenAI **spends $700k every day** to run ChatGPT as of August 2023.
  d. As per Reuters, OpenAI generated an estimated **$3.7 billion** in revenue from ChatGPT in 2024, with projections reaching **$11.7 billion** by 2025.
  e. **43% of college students** and **80% of the Fortune 500** companies are using ChatGPT in 2024.
  f. OpenAI was founded by **Sam Altman** and **Elon Musk**. Microsoft invested $10 billion in 2023. It is valued at **$29 billion** and is seeking a new valuation up to **$90 billion** due to higher revenues.
  g. A 2023 survey revealed that **25% of US companies saved $50K-$70K** using ChatGPT, while **11% saved over $100K**.
- Finally, here is nice video for more illustrative details on ChatGPT from OpenAI's CTO.

Transformers in Pytorch

- Hugging Face has a great library (and API) of pretrained transformer models with a great tutorials (check it out here).
- Best part: all free! You just install it:

```
pip install transformers
```

- There you find transformers trained for all kinds of AI tasks:
  - **Natural Language Processing**: text classification, question answering, summarization, translation, and text generation, etc.
  - **Computer Vision**: classification, detection, and segmentation.
  - **Audio**: automatic speech recognition and audio classification.
  - **Multimodal**: table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.
- You can also use BERT, GPT 1 and GPT 2 there for free.



Video: *ChatGPT, AI , Energy and more*