RNA Seminar

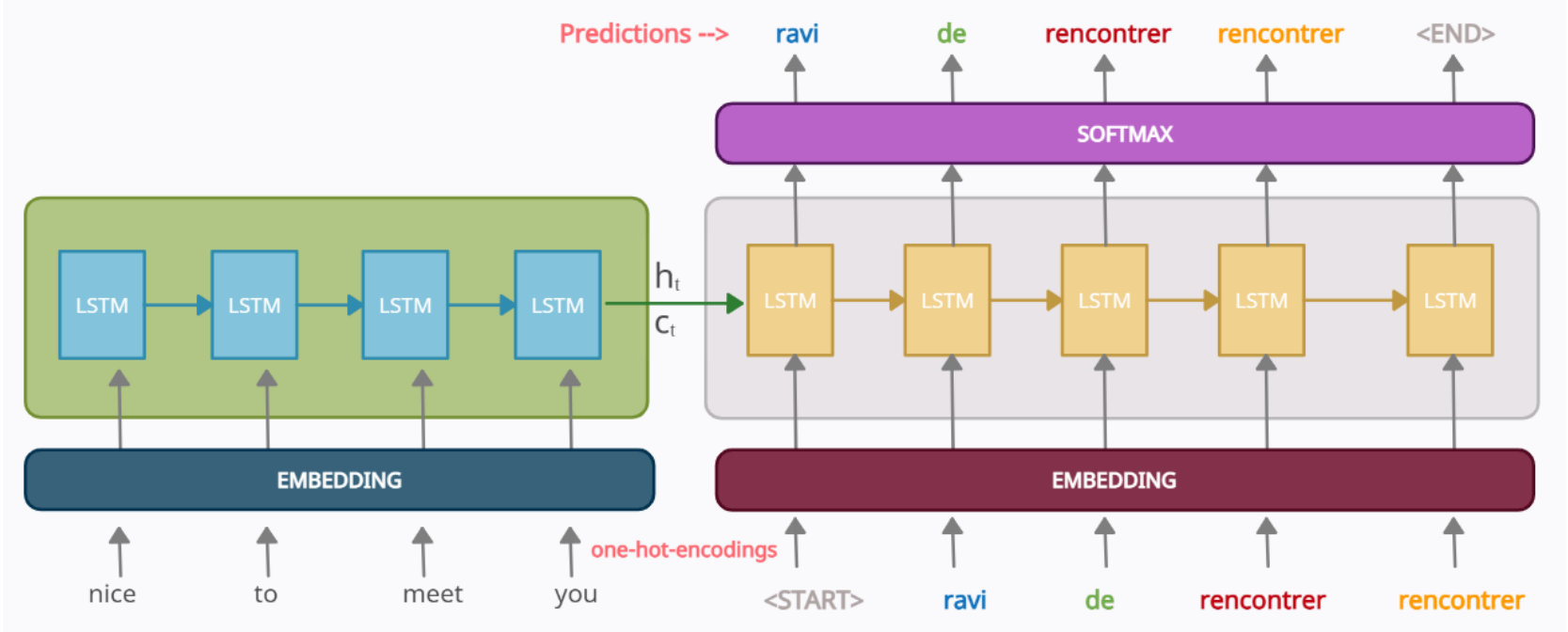
January 23th, 2023
UNIVERSITAT POLITÈCNICA DE VALÈNCIA, SPAIN

Transformers, beyond NLP

Roberto Paredes – PRHLT Group (UPV)

Recurrent Networks and Attention Models

Recurrent Nets

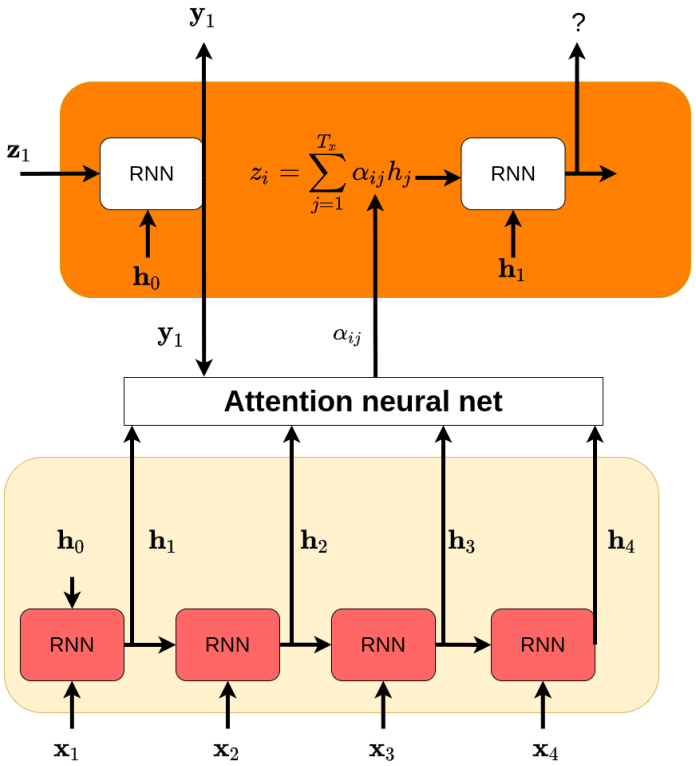


Attention

“Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we **relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector**. With this new approach, the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.”

Neural machine translation by jointly learning to align and translate

Attention



$$z_i = \sum_{j=1}^{T} \alpha_{ij} \mathbf{h}_j$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{Tx} \exp\left(e_{ik}\right)}$$

$$e_{ij} = \text{attention}_{\text{net}}\left(\mathbf{y}_{i-1}, h_j\right)$$

score

Attention

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Self-attention

Self-attention: the key component of the **Transformer** architecture

We can also define the attention of the same sequence, called self-attention. Instead of looking for an input-output sequence association/alignment, **we are now looking for scores between the elements of the sequence**

Self-attention
Probability score matrix

	Hello	I	love	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
love	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

A trained self-attention layer will associate the word “love” with the words ‘I” and “you” with a higher weight than the word “Hello”. From linguistics, we know that these words share a subject-verb-object relationship and that’s an intuitive way to understand what self-attention will capture.

Transformers

Transformers

With Recurrent Neural Networks we used to treat sequences sequentially to keep the order of the sentence in place. To satisfy that design, each RNN component (layer) needs the previous (hidden) output. As such, stacked LSTM computations were performed sequentially.

How to avoid this sequential process and use the hardware at máximum level:

1. **Representing the input sentence: self-attention and positional enconding (sequence as a set)**
2. **Representing the output sentence: masking and positional enconding (sequence as a masked-set)**

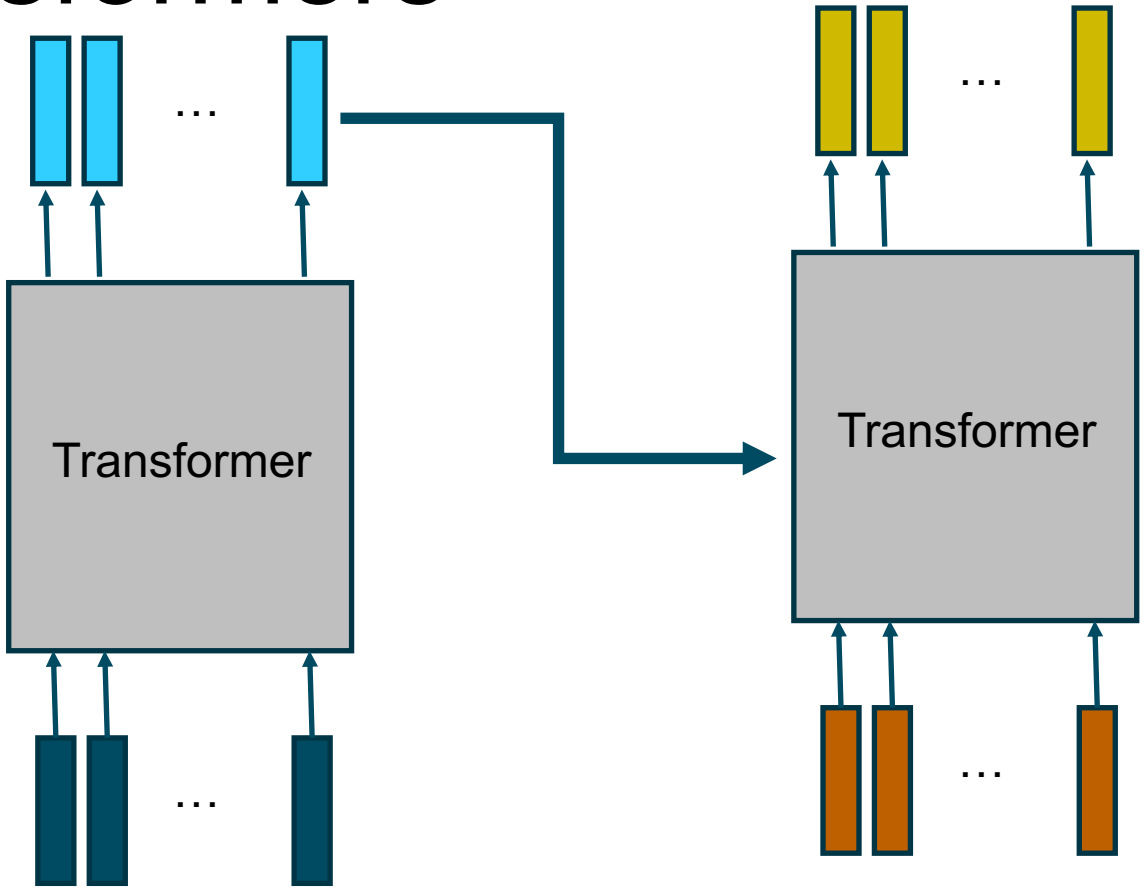
Transformers

Transformers transform

A **set** of N vectors into a **set** of N vectors

The output is supossed to be enriched with contextual information of the whole input and/or contextual information from another Transformer

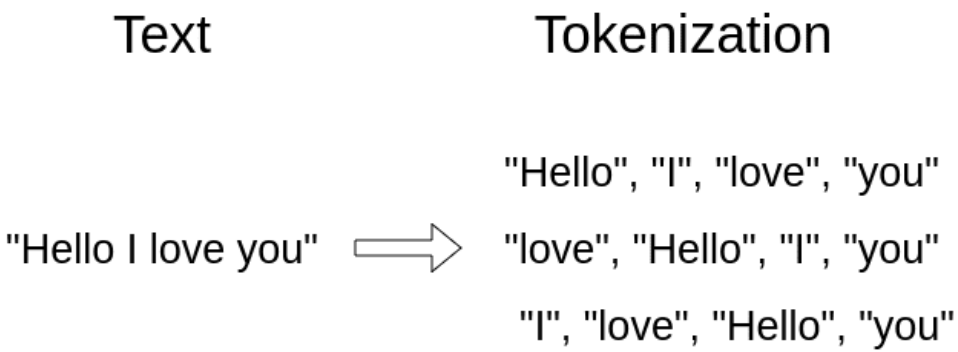
Transformers



Transformers

The transformer revolution started with a simple question: Why don’t we feed the entire input sequence.

Instead of a sequence of elements, we now have a set. In other words, **the order is irrelevant**.



Transformers

When you convert a sequence into a set (**tokenization**), you lose the notion of order.
Positional encoding is a set of small constants, which are added to the word embedding vector before the first self-attention layer.
Therefore, if the same word appears in a different position, the actual representation will be slightly different, **depending on where it appears in the input sentence.**

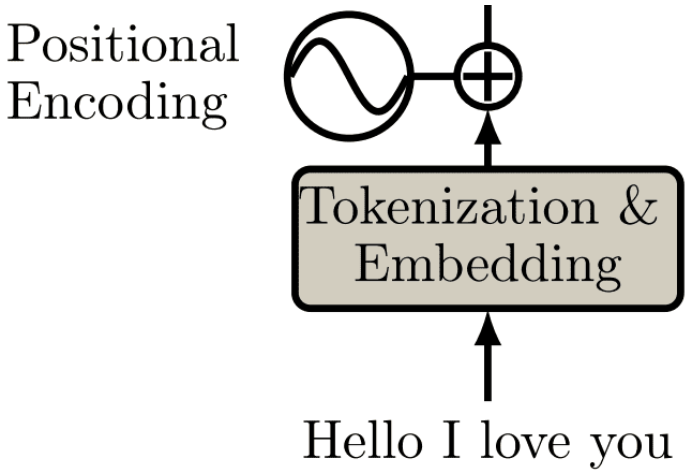
Mathematically:

*Assuming 512 is the dimensionality of the embedding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/512}}\right)$$

$$PE_{(pos,2*i+1)} = \cos\left(\frac{pos}{10000^{2i/512}}\right)$$

Transformers



Transformers

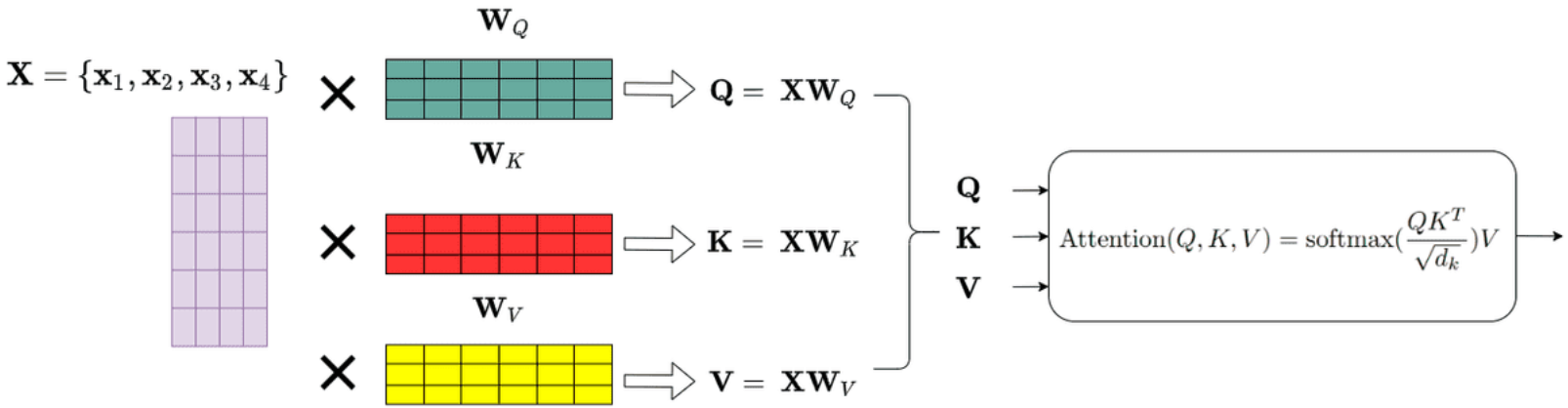
Attention mechanism: Query, Key and Values

Related to retrieval systems where given a Query we find similar Keys and return the associated Values. However here we return all the Values but weighted

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

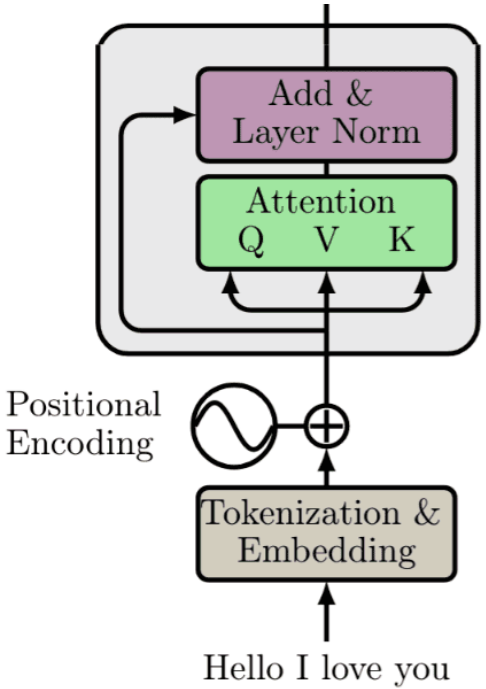
Transformers

Self-attention



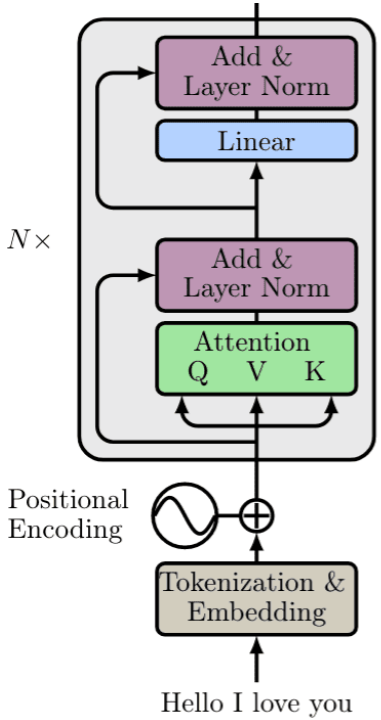
Transformers

Normalization and residual connection



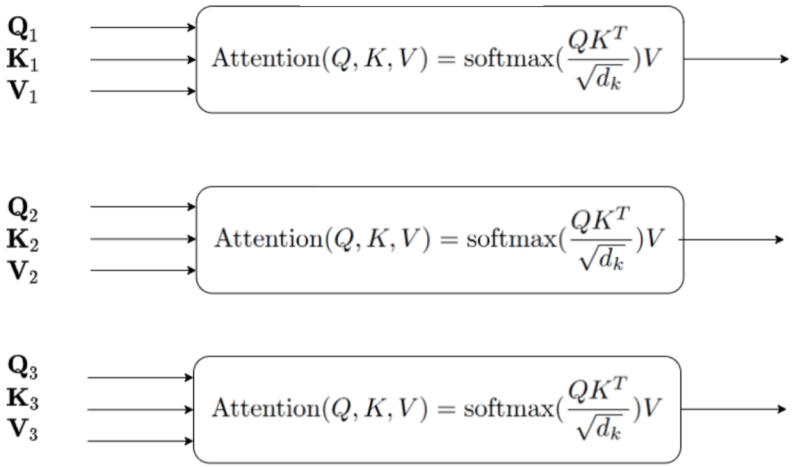
Transformers

Linear projection, normalization and residual connection



Transformers

Multihead attention

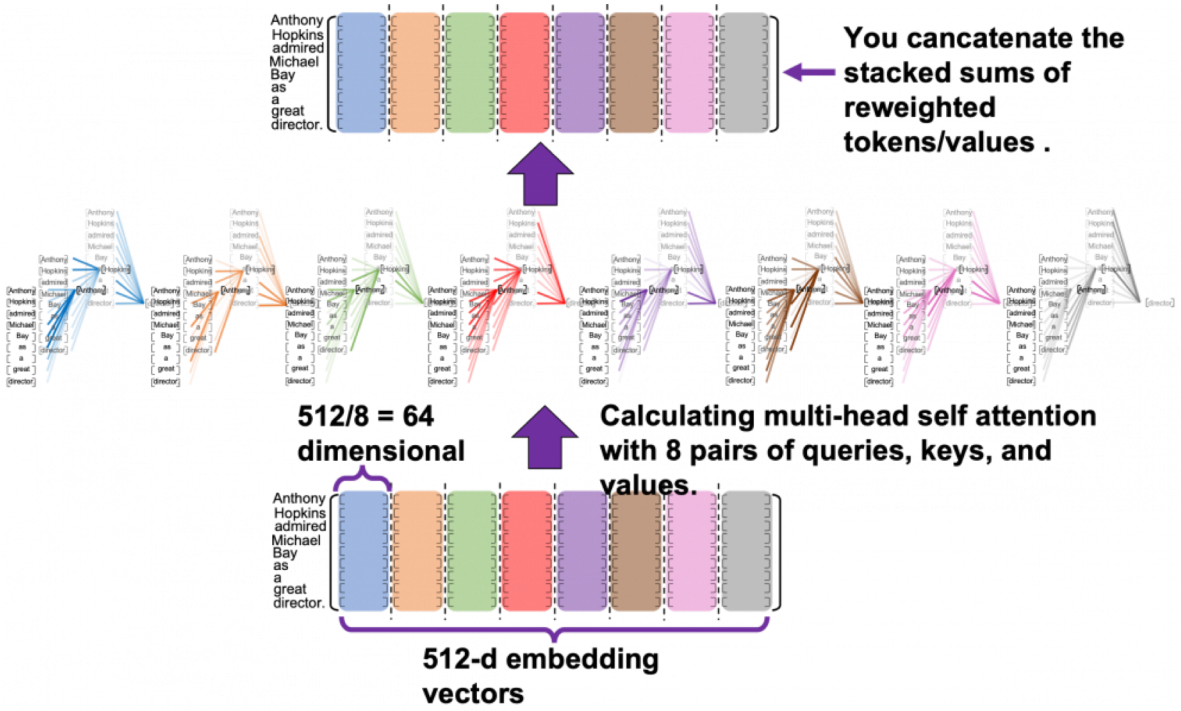


$$\text{MultiHead}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) \mathbf{W}^O$$

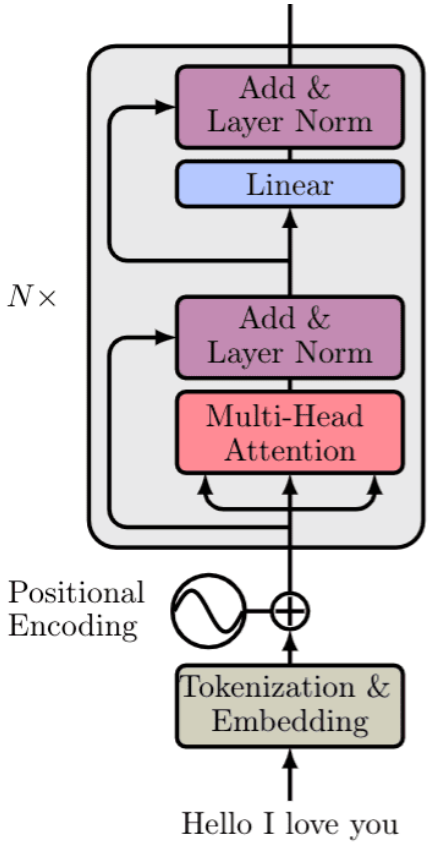
$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$

Transformers

Multihead attention



Transformers - Encoder

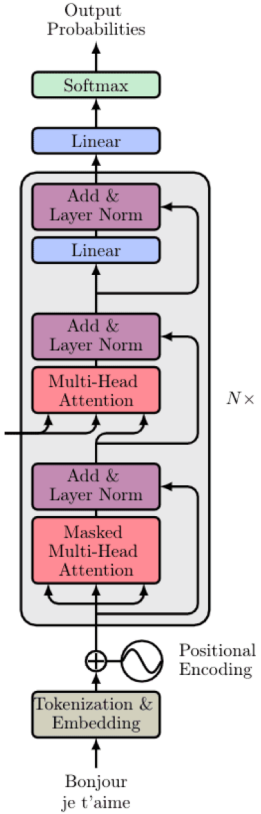


Transformers

Multihead attention

Notebooks

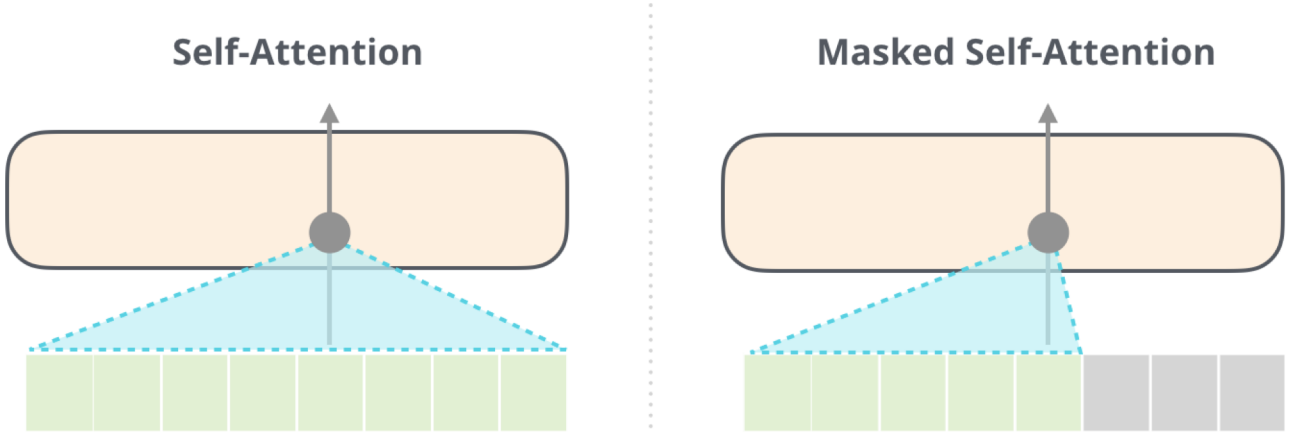
Transformers - Decoder



Each decoder block includes:

- A **Masked** multi-head self-attention layer
- A normalization layer followed by a residual connection
- A new multi-head attention layer (known as **Encoder-Decoder attention**)
- A second normalization layer and a residual connection
- A linear layer and a third residual connection

Transformers

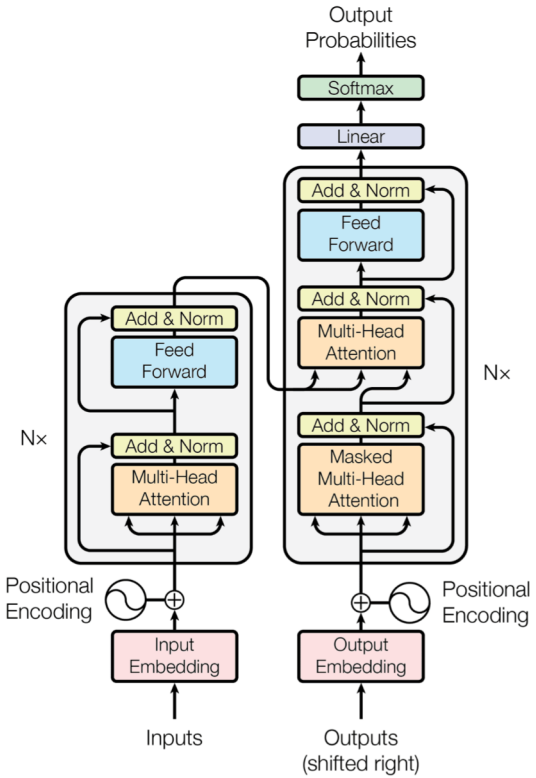


Transformers

Multihead attention

Notebooks

Transformers



In the decoder Multi-Head Attention layer the K and V comes from the encoder

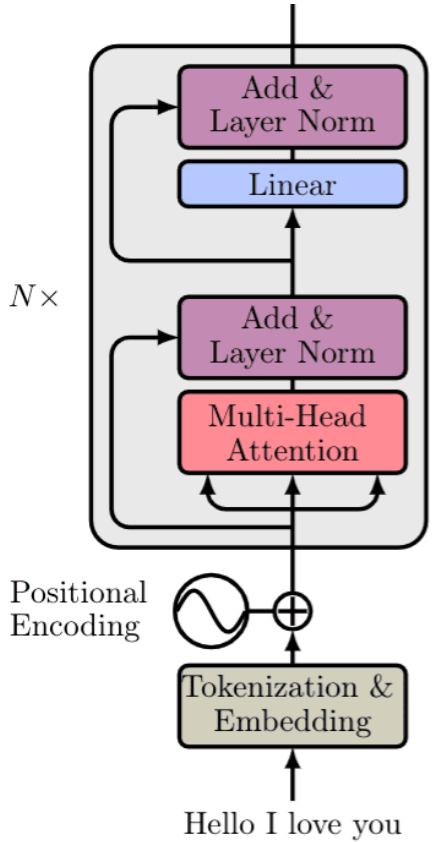
```
class Transformer(nn.Module):
    def __init__(self, src_vocab, trg_vocab, d_model, N, heads):
        super().__init__()
        self.encoder = Encoder(src_vocab, d_model, N, heads)
        self.decoder = Decoder(trg_vocab, d_model, N, heads)
        self.out = nn.Linear(d_model, trg_vocab)
    def forward(self, src, trg, src_mask, trg_mask):
        e_outputs = self.encoder(src, src_mask)
        d_output = self.decoder(trg, e_outputs, src_mask, trg_mask)
        output = self.out(d_output)
        return output
```

Transformers

```
class Encoder(nn.Module):
    def __init__(self, vocab_size, d_model, N, heads):
        super().__init__()
        self.N = N
        self.embed = Embedder(vocab_size, d_model)
        self.pe = PositionalEncoder(d_model)
        self.layers = get_clones(EncoderLayer(d_model, heads), N)
        self.norm = Norm(d_model)
    def forward(self, src, mask):
        x = self.embed(src)
        x = self.pe(x)
        for i in range(N):
            x = self.layers[i](x, mask)
        return self.norm(x)
```

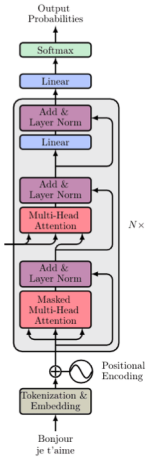
```
class EncoderLayer(nn.Module):
    def __init__(self, d_model, heads, dropout = 0.1):
        super().__init__()
        self.norm_1 = Norm(d_model)
        self.norm_2 = Norm(d_model)
        self.attn = MultiHeadAttention(heads, d_model)
        self.ff = FeedForward(d_model)
        self.dropout_1 = nn.Dropout(dropout)
        self.dropout_2 = nn.Dropout(dropout)

    def forward(self, x, mask):
        x2 = self.norm_1(x)
        x = x + self.dropout_1(self.attn(x2,x2,x2,mask))
        x2 = self.norm_2(x)
        x = x + self.dropout_2(self.ff(x2))
        return x
```



Transformers

```
class Decoder(nn.Module):
    def __init__(self, vocab_size, d_model, N, heads):
        super().__init__()
        self.N = N
        self.embed = Embedder(vocab_size, d_model)
        self.pe = PositionalEncoder(d_model)
        self.layers = get_clones(DecoderLayer(d_model, heads), N)
        self.norm = Norm(d_model)
    def forward(self, trg, e_outputs, src_mask, trg_mask):
        x = self.embed(trg)
        x = self.pe(x)
        for i in range(self.N):
            x = self.layers[i](x, e_outputs, src_mask, trg_mask)
        return self.norm(x)
```




```
class DecoderLayer(nn.Module):
    def __init__(self, d_model, heads, dropout=0.1):
        super().__init__()
        self.norm_1 = Norm(d_model)
        self.norm_2 = Norm(d_model)
        self.norm_3 = Norm(d_model)

        self.dropout_1 = nn.Dropout(dropout)
        self.dropout_2 = nn.Dropout(dropout)
        self.dropout_3 = nn.Dropout(dropout)

        self.attn_1 = MultiHeadAttention(heads, d_model)
        self.attn_2 = MultiHeadAttention(heads, d_model)
        self.ff = FeedForward(d_model).cuda()


def forward(self, x, e_outputs, src_mask, trg_mask):
        x2 = self.norm_1(x)
        x = x + self.dropout_1(self.attn_1(x2, x2, x2, trg_mask))
        x2 = self.norm_2(x)
        x = x + self.dropout_2(self.attn_2(x2, e_outputs, e_outputs,
        src_mask))
        x2 = self.norm_3(x)
        x = x + self.dropout_3(self.ff(x2))
        return x
```

Transformers

Language Models with Transformers - BERT

In its vanilla form, Transformer includes two separate mechanisms, an encoder that reads the text input and a decoder that produces a prediction for the task.

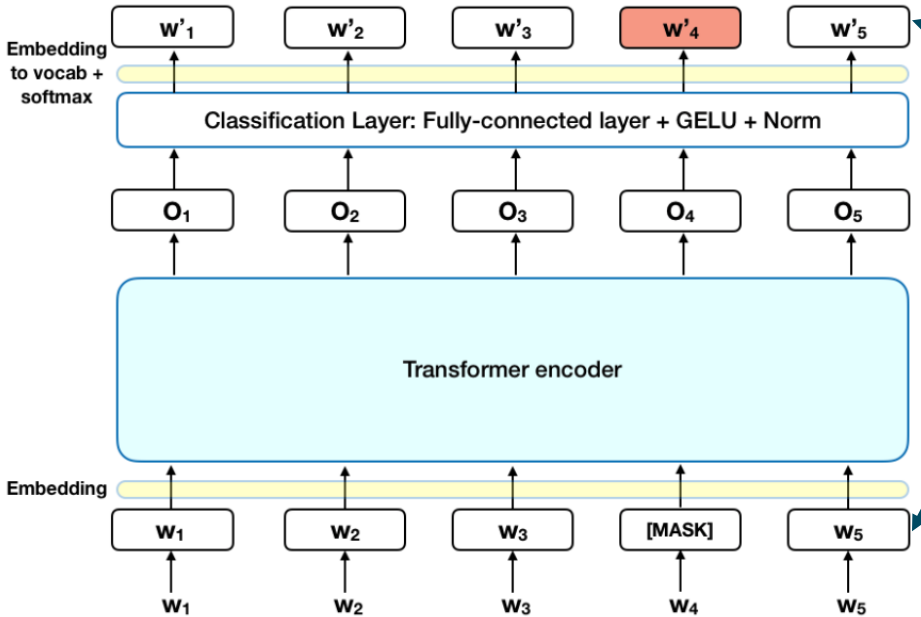
BERT (*Bidirectional Encoder Representations from Transformers)* makes use of Transformer, an attention mechanism that learns contextual relations between words in a text.

Since BERT's goal is to generate a language model, only the encoder mechanism is necessary

Transformer encoder reads the entire sequence of words at once. Therefore, it is considered bidirectional, though it would be more accurate to say that it is non-directional

Transformers

Byte-Pair Encoding: Subword-based tokenization algorithm

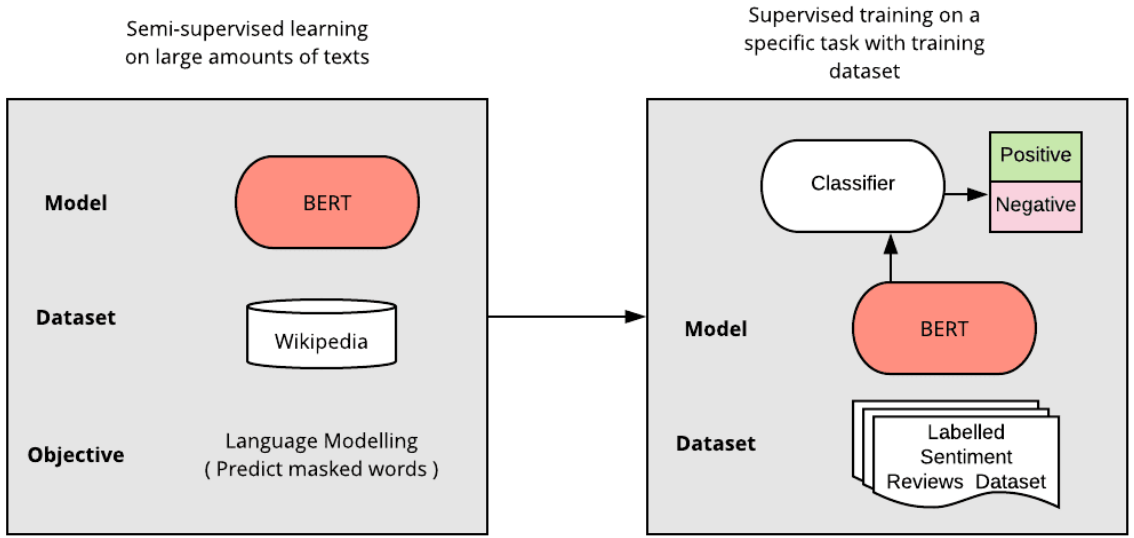


Tokens

https://huggingface.co/bert-base-cased/resolve/main/vocab.txt

Transformers

These Transformer Language Models are used as a backbone for different NLP tasks, for instance sentiment analysis, among others



Transformers

GPT vs BERT

The GPT is built using transformer decoder blocks

BERT, uses transformer encoder blocks

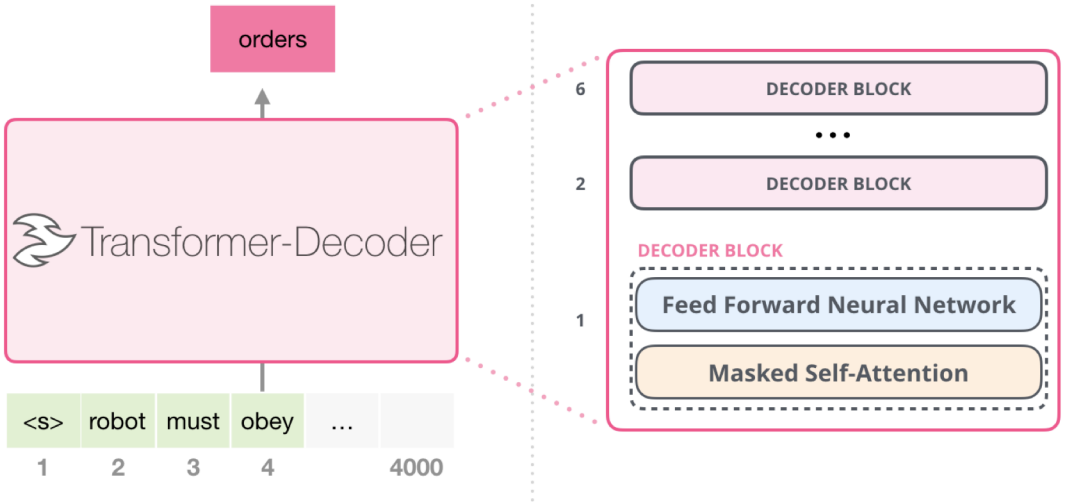
The GPT, and some later models are **auto-regressive**

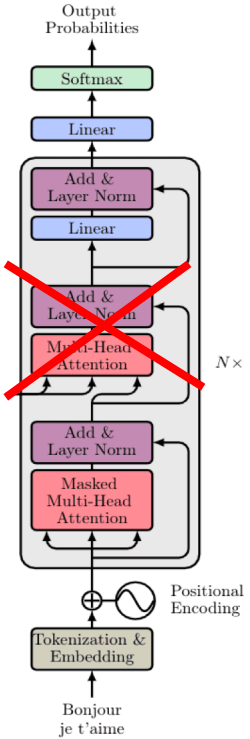
The way these models actually work is that after each token is produced, that token is added to the sequence of inputs.

In losing auto-regression, BERT gained the ability to incorporate the context on both sides of a word to gain better results

Transformers

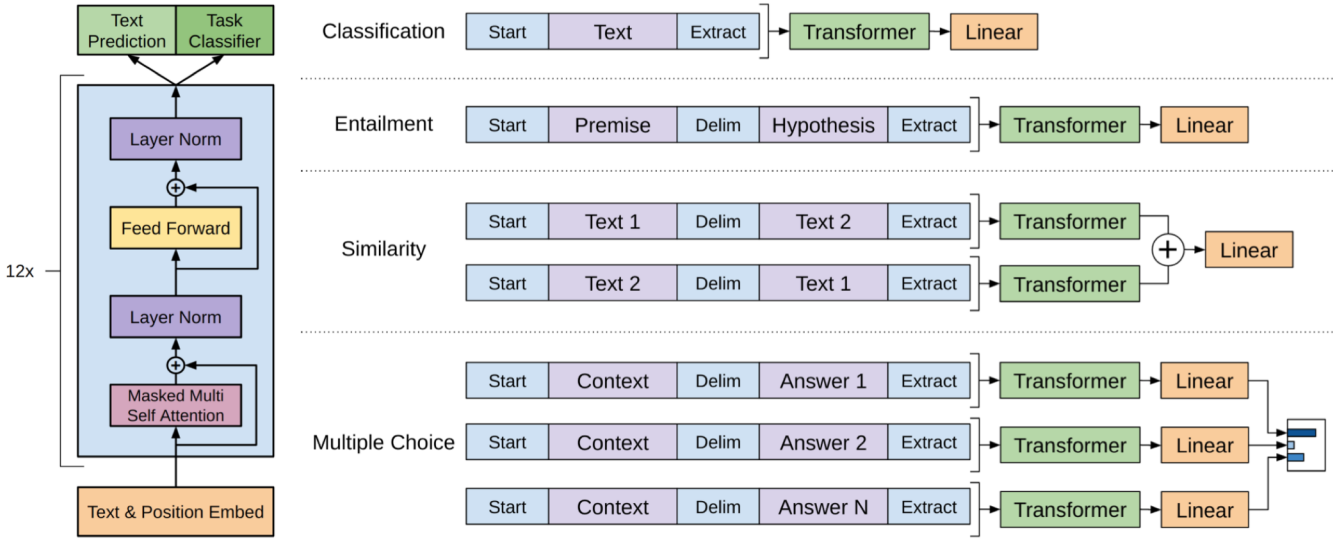
Improving Language Understanding by Generative Pre-Training (GPT)





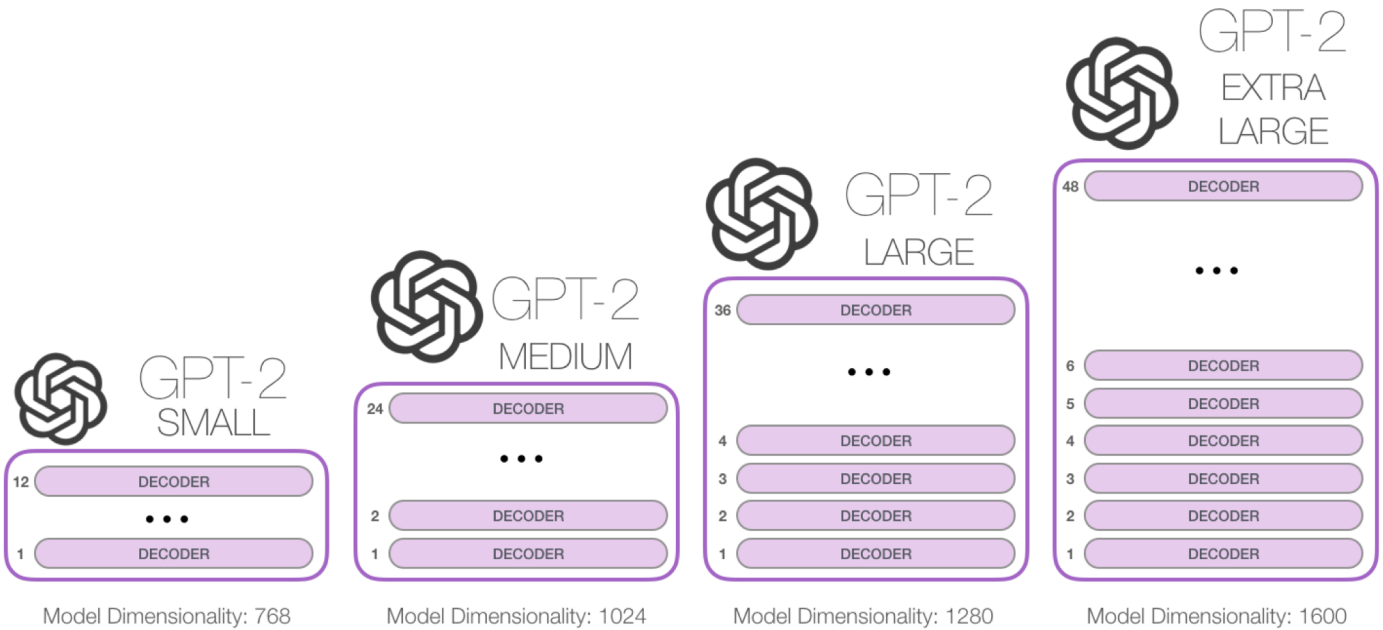
Transformers

GPT: Supervised fine-tuning



Transformers

GPT2: Language Models are Unsupervised Multitask Learners



Transformers

GPT2 Beyond LM – Fine-tuning – Sentiment Analysis

Training prompt (as we want the model to learn this "pattern" to solve the "task")

```
Tweet: I am not feeling well.
Sentiment: Negative
```

Test prompt (as now we hope the model has learned the "task" and hence could complete the "pattern")

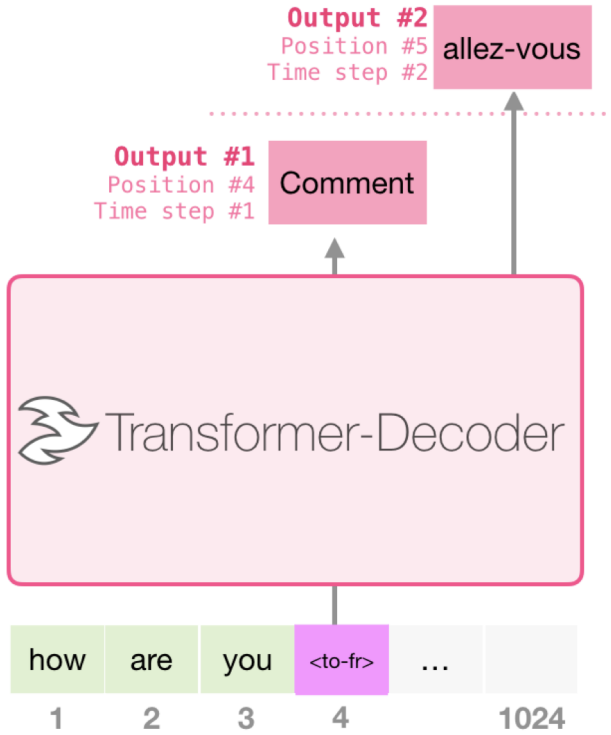
```
Tweet: I am feeling well.
Sentiment:
```

Transformers

GPT2 Beyond LM – Machine Translation

Training Dataset

I	am	a	student	<to-fr>	je	suis	étudiant
let	them	eat	cake	<to-fr>	Qu'ils	mangent	de
good	morning	<to-fr>	Bonjour				

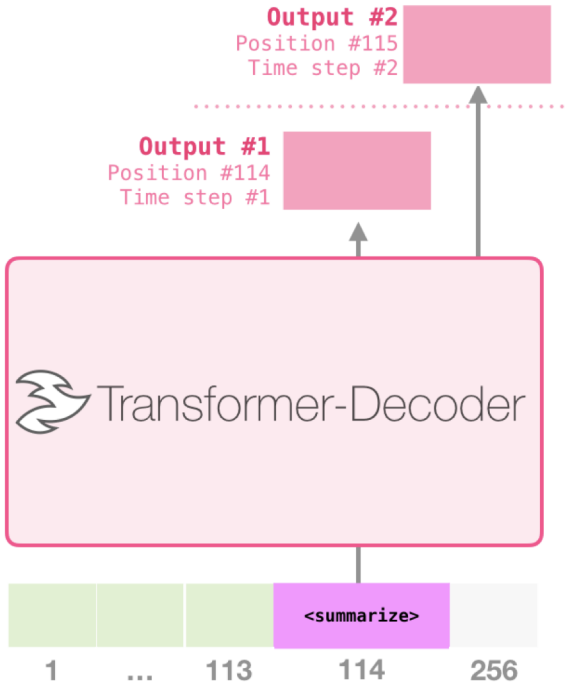


Transformers

GPT2 Beyond LM - Summarization

Training Dataset

Article #1 tokens	<summarize>	Article #1 Summary	
Article #2 tokens	<summarize>	Article #2 Summary	padding
Article #3 tokens	<summarize>	Article #3 Summary	



Vision Transformers

(Computer Vision)