Attention

Rodrigo Serna Pérez

March 2022

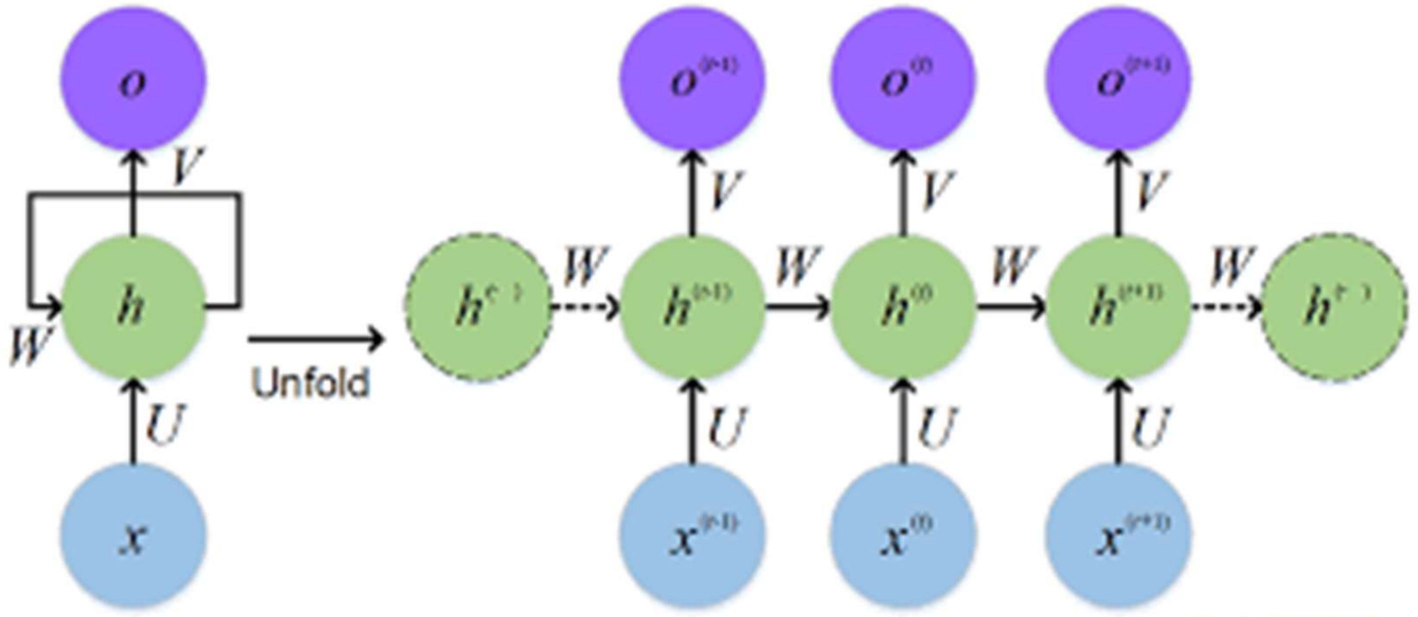
comillas.edu

Today we'll talk about …

1

Origins of attention

Reminder about RNNs

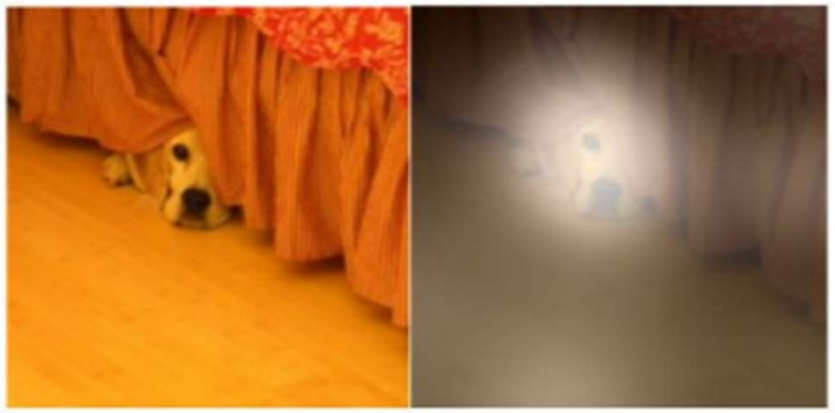


Reminder about RNNs

- **RNNs and LSTMs** are very cool!!
  - Can exploit temporal correlation
  - Number of parameters is independent of the length of the time series.
  - Can be run on time series of different length.
- But they also have some **problems** …
  - They tend to forget too fast. Problematic when working with long texts.
  - Recurrent behaviour cannot parallelize on GPUs

Attention Origins

- Attention is a deep learning **architecture initially developed for image processing.**
- Let's build a dog / cat classifier.

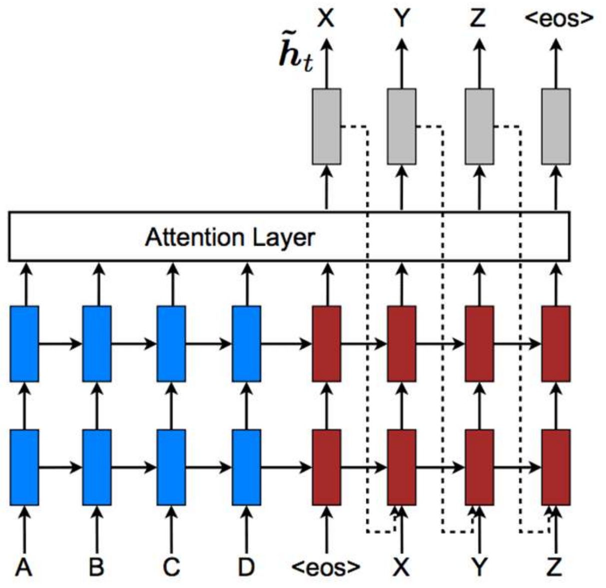


A few pixels contain the needed
information to decide if the image contains a dog or a cat.

Our network must be able to learn to **pay attention** to those pixels and forget about the other.

Attention Origins

- Although initially was developed for images, **focus was soon changed to text**.
- Attention was thought to be able to fix the problems of LSTM about long term dependencies processing.



Attention Layer is put on the top of a RNN / LSTM/ CNN.

Performance was significantly increased in several tasks such as translation.

Attention Origins

- In 2017, a famous paper was published: Attention is All You Need
- In the paper, authors show that attention mechanism is so powerful that you don't need the LSTM!!
- They propose **Transformer: an architecture that only uses stacked attention layers** giving state of the art results in several NLP related tasks.
- And AI field was never the same …

2

Attention

Intuition

- Consider we want to translate the following text to Spanish:

In my opinion, this second hypothesis would imply the failure of Parliament in its duty as a Parliament, as well as introducing an original thesis, an unknown method which consists of making political groups aware, in writing, of a speech concerning the Commission's programme a week earlier—and not a day earlier, as had been agreed—bearing in mind that the legislative programme will be discussed in February, so we could forego the debate, since on the next day our citizens will hear about it in the press and on the Internet and Parliament will no longer have to worry about it.

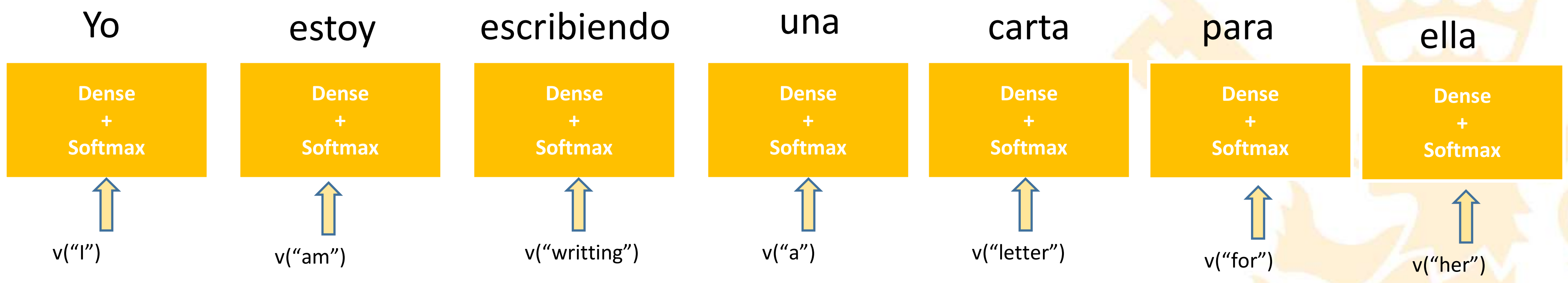
- We could start writing the translation word by word, but sometimes, we would like to **look behind for certain past words**.

Computing contextual word embeddings

- Let’s build a neural network that **translates english text to spanish**.
- Imagine we want to translate the sentence:

  “I am writing a letter to her”

- Idea: perform word wise translation. We can use a pretrained word embedding to help with the task.



Computing contextual word embeddings

•Seems an easy task, only needs to learn **mapping from the spanish words to the english ones** ("nosotros" is "we", "tenemos" is "have" …)

•But we have a **problem with word "letter":**

- Sometimes must be translated by "letra"
- Sometimes must be translated by "carta"

•Remember word embedding works as dictionary! One word means only one vector, even if it has two different meanings!

•We need to build **contextual word embeddings**.

Computing contextual word embeddings

- We want to compute contextual word embeddings for the words so:

$$contexvector(\text{letter}) = f(vector(letter), [vector(I), vector(am), \dots])$$

- Let's think simple ideas for building the vector. The mean?

$$contextvector(letter) = \sum_{words} \frac{1}{8} * vector(word)$$

- Doesn't seem appropiate …

Computing contextual word embeddings

- What about the weighted average?

$$contextvector(letter) = \sum_{words} \alpha_{word} * vector(word)$$

$$\sum_{words} \alpha_{word} = 1$$

- Vector of “letter” is a weighted average of all vectors in the sentence.

- Which should be the coefficients? They must be a function of the same words.

Computing contextual word embeddings

- To compute the weights, let's keep it simple: **inner product.**

$$\alpha_{word} = < vector(word), vector(letter) >$$

- Then we compute softmax so the sum of all the coefficients is 1.

- Now we have the full equation for self - attention:

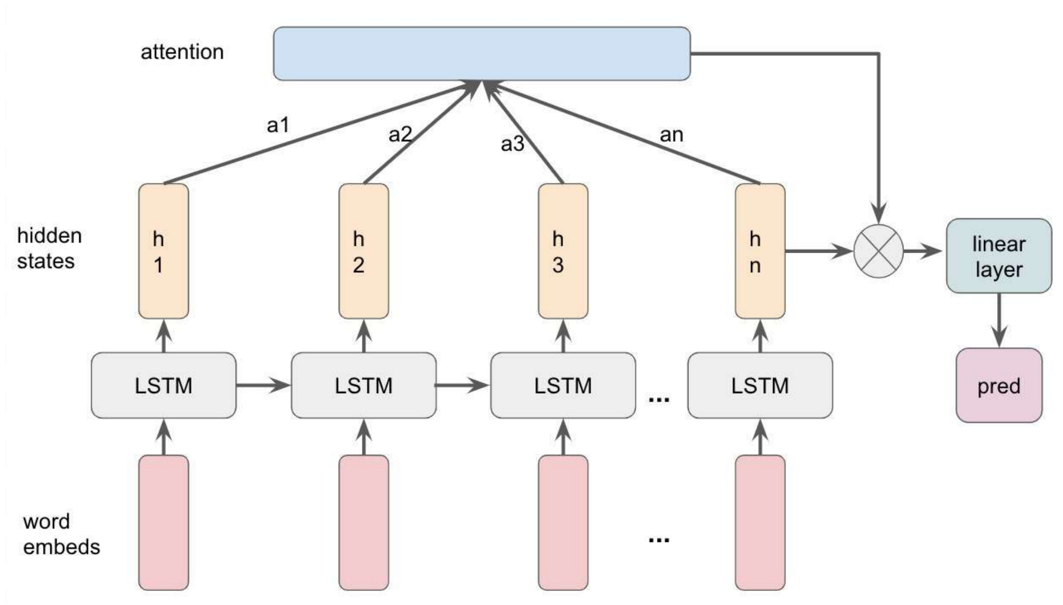
$$contextword(letter) = softmax(W * vector(letter)^T) * W$$

$$W \text{ is the matrix of vectors}$$

Self-attention model

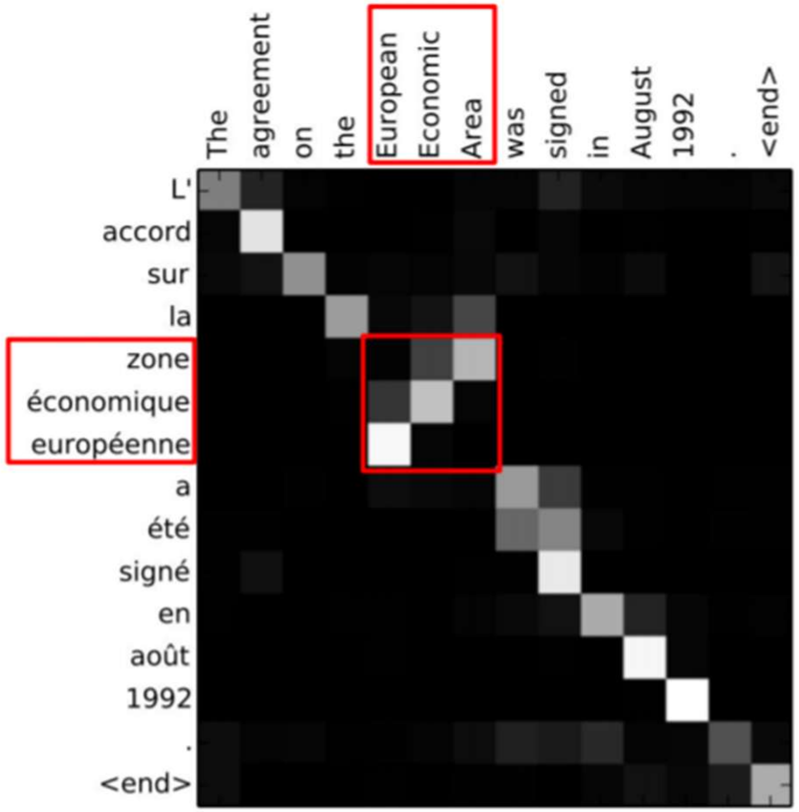
- **Self-Attention:** mechanism to transform embeddings to "more contextual" embeddings.
- Equation:

$$SelfAttention(V) = \frac{softmax(VV^T)}{\sqrt{d_k}} * V \qquad d_k = size\ of\ the\ embedding\ vectors$$



Self-attention model

- Weights are very useful to **give us interpretation**.

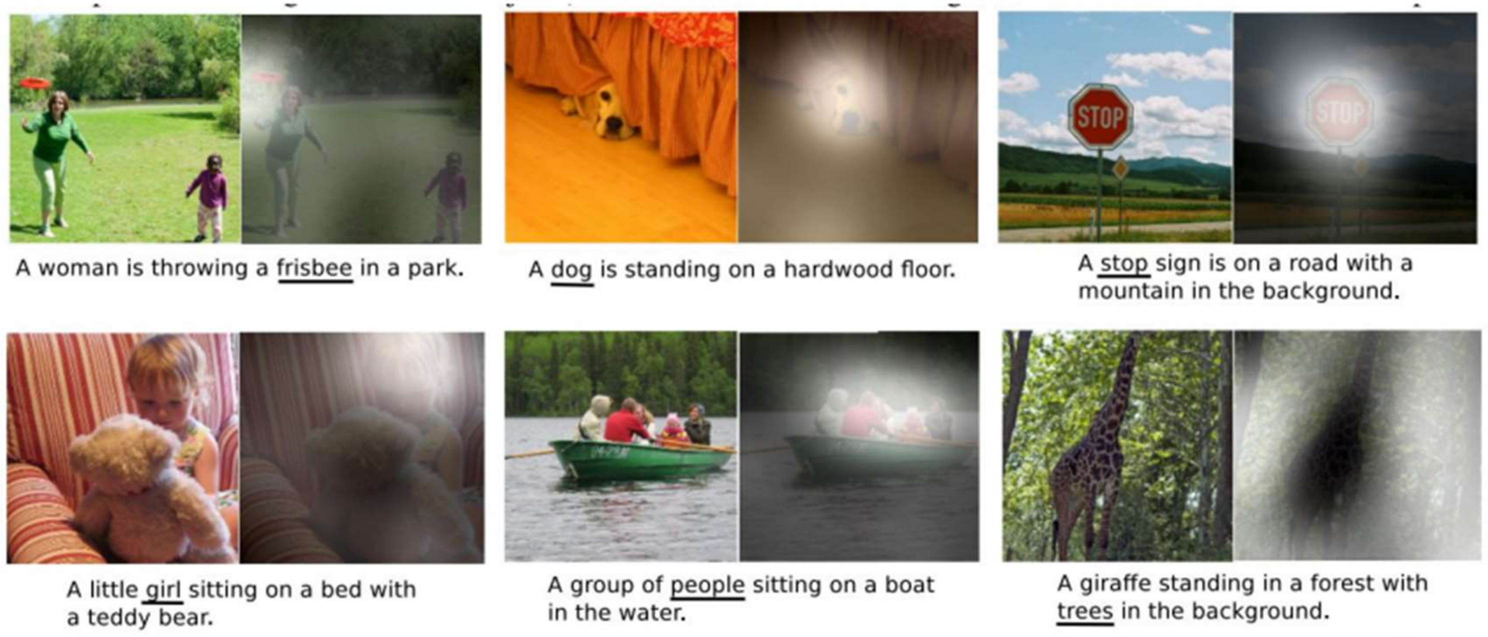


In order to generate the word “Area”, the model has focused mainly in words ”la” and “zone”.

Note that the model has learnt the order of adjectives in french and english is not the same!!

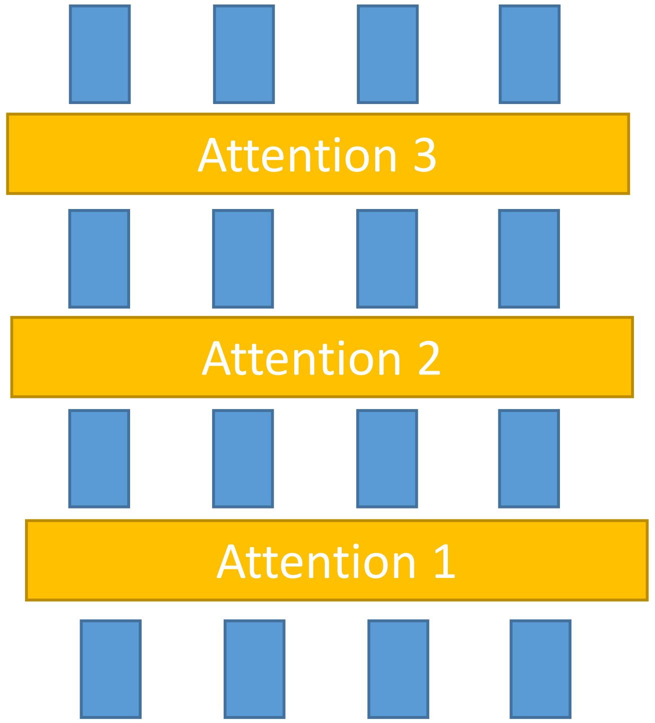
Self-attention model

- **Image Captioning**: given an image generate create the description.
- The model will attend different parts of the image to generate each one of the words.


A woman is throwing a frisbee in a park.
A dog is standing on a hardwood floor.
A stop sign is on a road with a mountain in the background.
A little girl sitting on a bed with a teddy bear.
A group of people sitting on a boat in the water.
A giraffe standing in a forest with trees in the background.

Self-attention model

- Layers of self attention can be stacked in order to generate "more contextual embeddings".

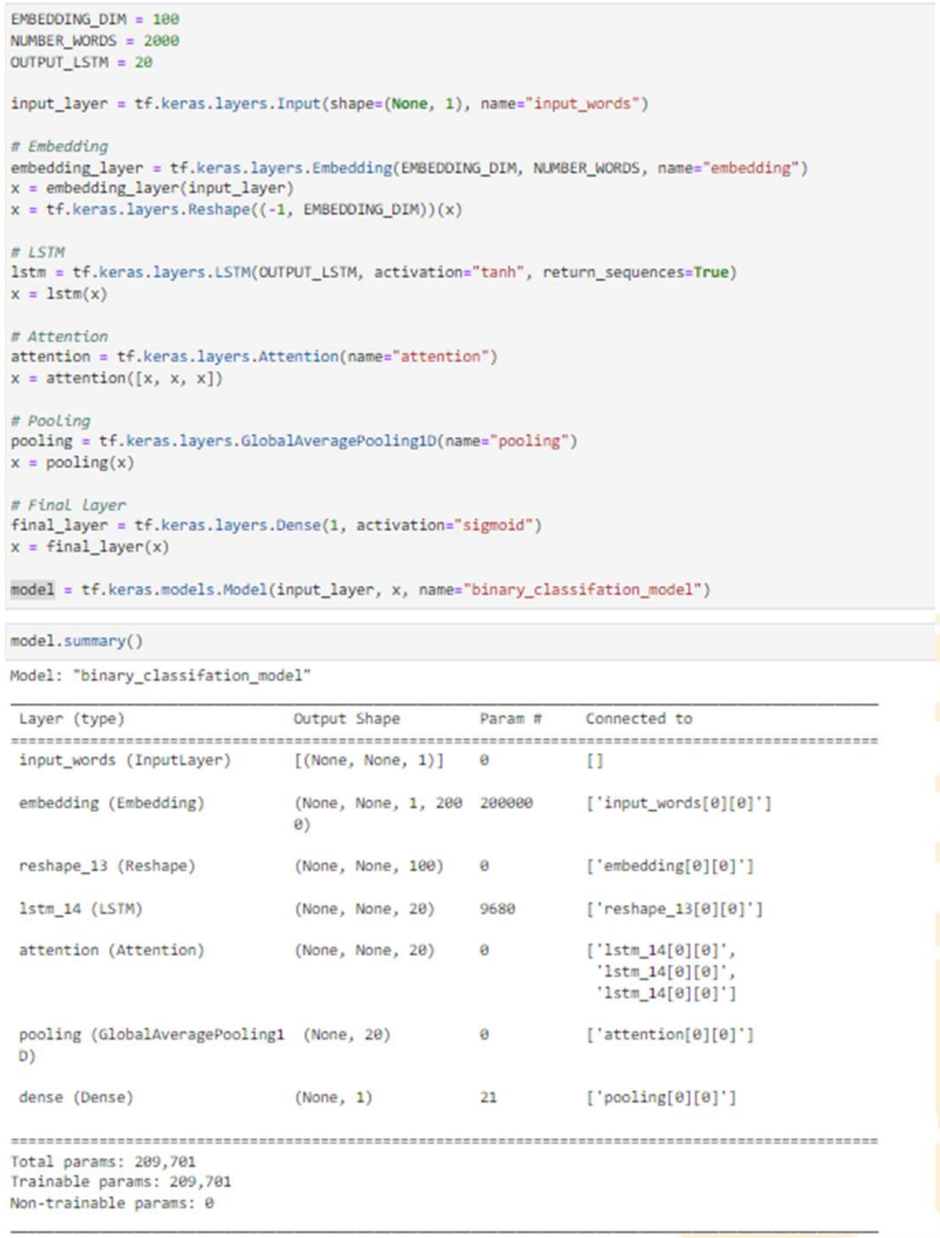


3

Implementation

Implementation

- Attention layer can be implemented on the top of other model for sequences (RNN, LSTM or CNN).



```python
EMBEDDING_DIM = 100
NUMBER_WORDS = 2000
OUTPUT_LSTM = 20

input_layer = tf.keras.layers.Input(shape=(None, 1), name="input_words")

# Embedding
embedding_layer = tf.keras.layers.Embedding(EMBEDDING_DIM, NUMBER_WORDS, name="embedding")
x = embedding_layer(input_layer)
x = tf.keras.layers.Reshape((-1, EMBEDDING_DIM))(x)

# LSTM
lstm = tf.keras.layers.LSTM(OUTPUT_LSTM, activation="tanh", return_sequences=True)
x = lstm(x)

# Attention
attention = tf.keras.layers.Attention(name="attention")
x = attention([x, x, x])

# Pooling
pooling = tf.keras.layers.GlobalAveragePooling1D(name="pooling")
x = pooling(x)

# Final Layer
final_layer = tf.keras.layers.Dense(1, activation="sigmoid")
x = final_layer(x)

model = tf.keras.models.Model(input_layer, x, name="binary_classifation_model")
```

```python
model.summary()
```

```
Model: "binary_classifation_model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_words (InputLayer)       [(None, None, 1)]    0           []

 embedding (Embedding)          (None, None, 1, 200  200000      ['input_words[0][0]']
                                0)

 reshape_13 (Reshape)           (None, None, 100)    0           ['embedding[0][0]']

 lstm_14 (LSTM)                 (None, None, 20)     9680        ['reshape_13[0][0]']

 attention (Attention)          (None, None, 20)     0           ['lstm_14[0][0]',
                                                                  'lstm_14[0][0]',
                                                                  'lstm_14[0][0]']

 pooling (GlobalAveragePooling1  (None, 20)          0           ['attention[0][0]']
 D)

 dense (Dense)                  (None, 1)            21          ['pooling[0][0]']

==================================================================================================
Total params: 209,701
Trainable params: 209,701
Non-trainable params: 0
__________________________________________________________________________________________________
```

4

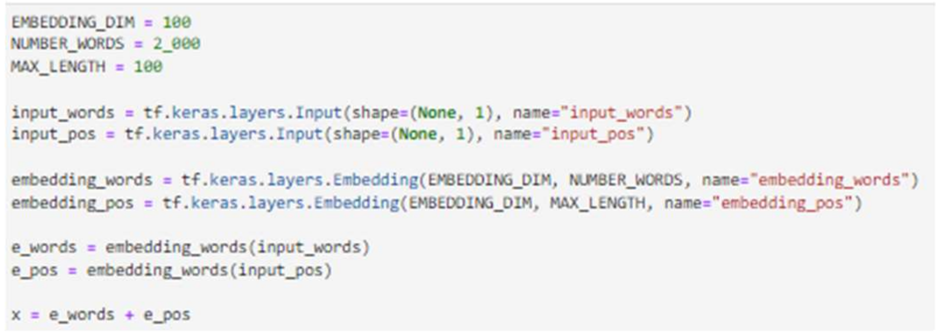
Positional Encoding

Positional Encoding

- **Attention is insensible to the order of the inputs!**
- That makes no sense: in order to understand a word, we must look at surrounded words. But … how could the network do that if the words order is lost?
- Idea: **the embedding of a word will be the sum of two components**:
  - Word Encoding: vector that depends on the word
  - Positional Encodings: vector that depends on the position of the word in the text.

Positional Encoding

- The positional encoding is a vector with the same dimension that the word encoding (let's call it D).
- The are two ways of computing those vectors:
  - **Learned**: using an embedding layer.



```
EMBEDDING_DIM = 100
NUMBER_WORDS = 2_000
MAX_LENGTH = 100

input_words = tf.keras.layers.Input(shape=(None, 1), name="input_words")
input_pos = tf.keras.layers.Input(shape=(None, 1), name="input_pos")

embedding_words = tf.keras.layers.Embedding(EMBEDDING_DIM, NUMBER_WORDS, name="embedding_words")
embedding_pos = tf.keras.layers.Embedding(EMBEDDING_DIM, MAX_LENGTH, name="embedding_pos")

e_words = embedding_words(input_words)
e_pos = embedding_words(input_pos)

x = e_words + e_pos
```

  - **Sinusoidal**:

$$Encoding(pos, 2 * i) = \sin \frac{pos}{10000^{\frac{2i}{D}}}$$

$$Encoding(pos, 2 * i + 1) = \cos \frac{pos}{10000^{\frac{2i}{D}}}$$

5

Generic Attention

Generic Attention Model

- Sometimes a more generic model is used.
- In this model, three different embeddings are computed for the input words: **query (Q), key (K) and value (V).**
- Then the equation for attention is:

$$attention(K, Q, V) = \frac{softmax(KQ^T)}{\sqrt{d_k}} * V$$

- Almost always K = V
- Self attention is the special case in which K = V = Q

Generic Attention Model

- The general model is typically used in two cases:

- **Models with two inputs**: one of them feeds the key embedding, the other feeds the value embedding.
  - **Question Answering**: a question and a context, return the part of the context with the answer of the question.

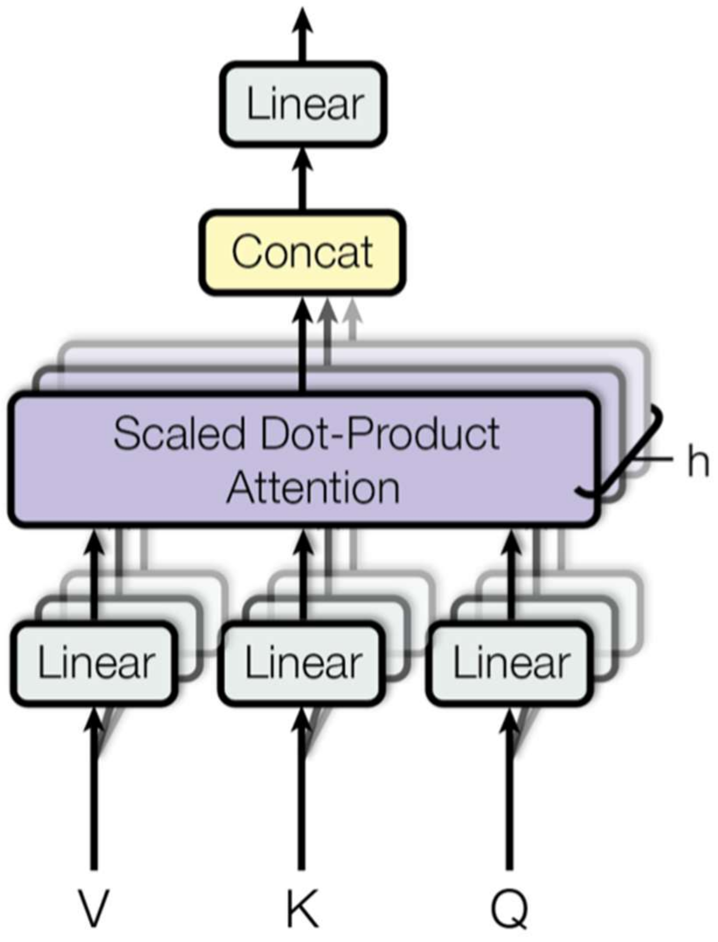
- **Sequence2Sequence models**: convert one time series to another.
  - **Text Translation**: original language text is the values, previous generated words are the key.
  - **Time series conversion**: such as predicting stock price of a company from the price of other companies.

6

Multi Head Attention

Multi Head Attention

- **Multi head attention** is an easy trick very used in order to improve parallelization.
- The embedding for each word is **split into H pieces of equal size**.
- Attention is applied to each one of the pieces.
- The output of each layer of attention (head) is concatenated.



7

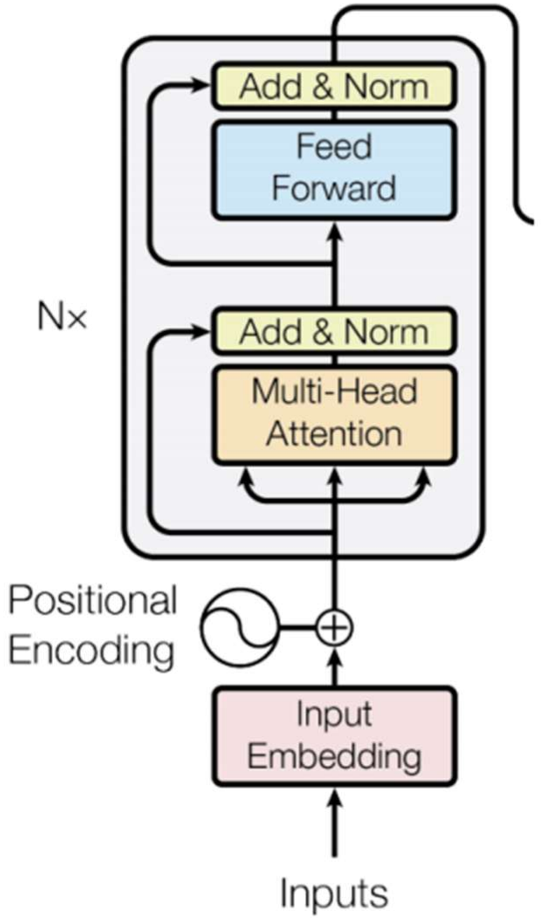
Transformers

Transformers

- We don't need to use a RNN / CNN in order to use attention.
- **Transformer:** deep learning architecture that only uses attention mechanism.
- Currently, **state of the art for almost all AI fields.**
- Transformers is an **encoder-decoder architecture** (similar to an autoencoder)
  - **Encoder**: maps input to an embedding vector
  - **Decoder**: builds output from the embedding vector
- In some cases, only the encoder is used (for example, text classification).
- In others, both are used (such as text translation or text generation).

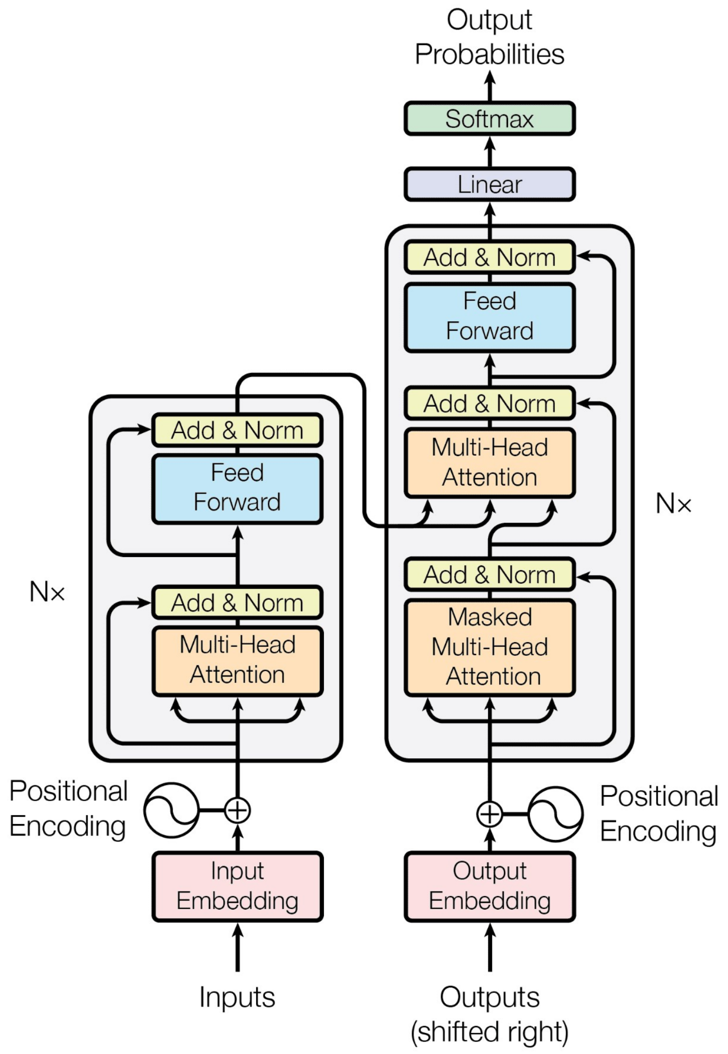
Transformers - Encoder

- A transformer block is composed of:
  - Self attention
  - Residual connection
  - Normalization
  - Feed Forward Layer (Dense)
  - Residual connection
  - Normalization
- Several blocks can be stacked.



Transformer – Encoder Decoder

- Decoder uses Full Attention.
- Decoder is a sequential model.



Transformer – Encoder Decoder

- Decoder is a sequential model.
- Example on text translation: we want to translate sentence “I want to go to the cinema”.
  - The input of the encoder is always the full sentence
  - The input of the decoder is:
    - 1: Inits special token such as <START>. Produces output “Yo”
    - 2: [<START>, Yo]
    - 3: [<START>, Yo, quiero]
    - 4: [<START>, Yo, quiero, ir]
    - …

Transformer

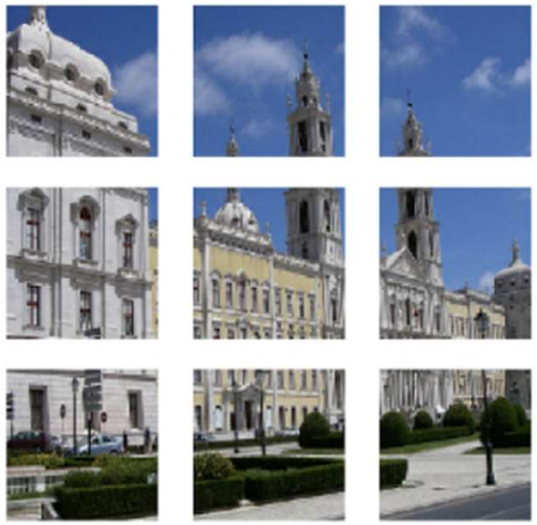
- Transformers are very used in contexts different that language.
- Example: we can use a transformer model in order to predict price stock of company A from price stock of company B.

7

Visual Transformers (VIT)

Visual Transformers

- **Transformers can also be applied to image related tasks**, performing even better than CNN deep networks.
- Images are split into smaller pieces.
- Then, each one of the pieces can be considered a word.



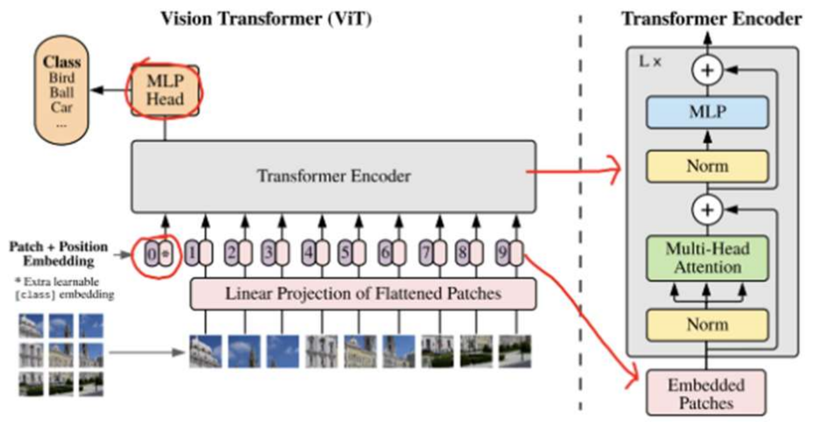
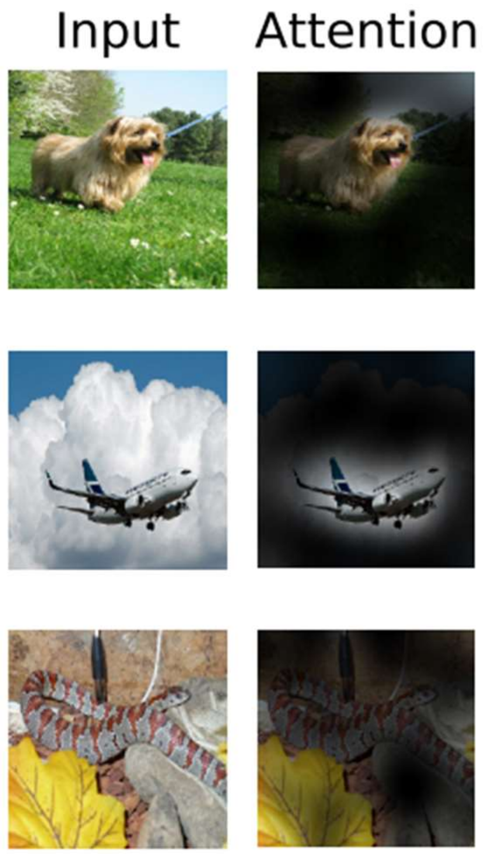


Figure 5. The entire ViT architecture, with the extra learnable embedding -marked in red, leftmost embedding (Source: Image from the original paper)

Link to paper

Visual Transformers

- Using the weights of the attention model, we can give some interpretation to our model.



COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI ICADE CIHS

comillas.edu