Neural Language Models

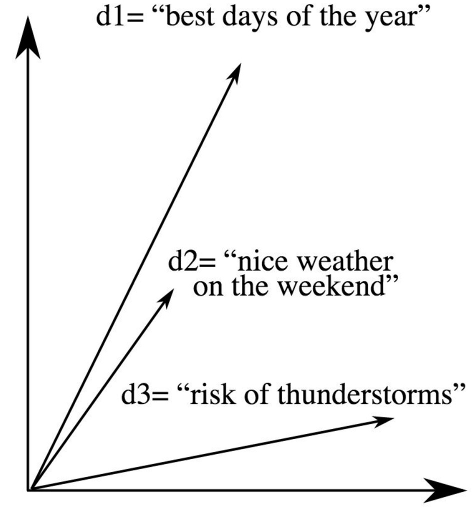
Andrea Esuli

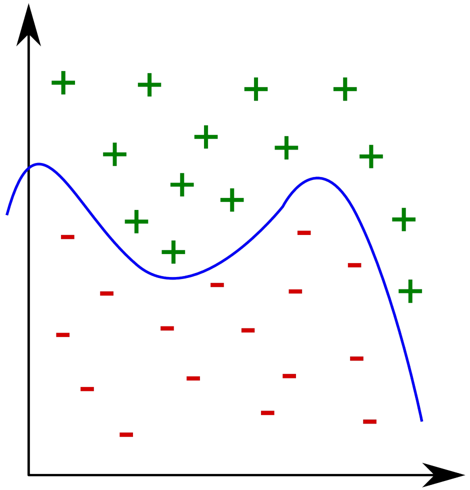
Vector Space Model

The *Vector Space Model* (VSM) is a typical machine-processable representation adopted for text.

Each vector positions a document into an *n*-dimensional space, on which learning algorithms operate to build their models

$$v(d_1) = [w_1, w_2 \ldots \ldots, w_{n-1}, w_n]$$


d1= "best days of the year"
d2= "nice weather on the weekend"
d3= "risk of thunderstorms"



Vector Space Model

After text processing, tokenization… a document is usually represented as vector in $R^{|F|}$, where F is the set of all the distinct *features* observed in documents.

Each feature is mapped to a distinct dimension in $R^{|F|}$ using a *one-hot* vector:

$$
\begin{array}{ll}
v(\text{'played'}) & = [1, 0, 0, \dots, 0, 0, \dots, 0, 0, 0] \\
v(\text{'game'}) & = [0, 1, 0, \dots, 0, 0, \dots, 0, 0, 0] \\
v(\text{'match'}) & = [0, 0, 1, \dots, 0, 0, \dots, 0, 0, 0] \\
 & \quad\quad\quad\quad \vdots \\
v(\text{'trumpet'}) & = [0, 0, 0, \dots, 0, 1, \dots, 0, 0, 0] \\
 & \quad\quad\quad\quad \vdots \\
v(\text{'bwoah'}) & = [0, 0, 0, \dots, 0, 0, \dots, 0, 0, 1]
\end{array}
$$

Vector Space Model

A document is represented as the weighted sum of its features vectors:

$$v(d) = \sum_{f \in d} w_{fd} v(f)$$

For example:

d = `'you played a good game'`

$v(d)$ = $[0, w_{\text{played},d}, w_{\text{game},d}, 0, \dots \dots 0, w_{\text{good},d}, 0 \dots \dots 0, 0]$

The resulting document **vectors are sparse**:

$$|\{i | v_i(d) \neq 0\}| \ll n$$

Sparse representations

d_1 = 'you played a game'
d_2 = 'you played a match'
d_3 = 'you played a trumpet'

$v(d_1)$ = [0, $w_{played,d1}$, $w_{game,d1}$, 0 , 0, … , 0, 0 , 0]
$v(d_2)$ = [0, $w_{played,d2}$, 0 , $w_{match,d2}$, 0, … , 0, 0 , 0]
$v(d_3)$ = [0, $w_{played,d3}$, 0 , 0 , 0, … , 0, $w_{trumpet,d3}$, 0]

Semantic similarity between features (game~match) is not captured:

$$sim(v(d_1), v(d_2)) \sim sim(v(d_1), v(d_3)) \sim sim(v(d_2), v(d_3))$$

Modeling word similarity

How do we model that *game* and *match* are related terms and *trumpet* is not?

Using linguistic resources: it requires a lot of human work to build them.

Observation: co-occurring words are semantically related.

Pisa	*is a*	*province*	*of*	*Tuscany*
Red	*is a*	*color*	*of the*	*rainbow*
Wheels	*are a*	*component*	*of the*	*bicycle*
**Red*	*is a*	*province*	*of the*	*bicycle*

We can exploit this propriety of language, e.g., following the *distributional hypothesis*.

Distributional hypothesis

“You shall know a word by the company it keeps!” Firth (1957)

Distributional hypothesis: the meaning of a word is determined by the contexts in which it is used.

*Yesterday we had **bim** at the restaurant.*
*I remember my mother cooking me **bim** for lunch.*
*I don't like **bim**, it's too sweet for my taste.*
*I like to dunk a piece **bim** in my morning coffee.*

Word-Context matrix

A word-context (or word-word) matrix is a $|F| \cdot |F|$ matrix X that **counts** the frequencies of co-occurrence of words in a collection of contexts (i.e, text spans of a given length).

You cook the cake twenty minutes in the oven at 220 C.
I eat my steak rare.
I'll throw the steak if you cook it too much.
The engine broke due to stress.
I broke a tire hitting a curb, I changed the tire.

Context$_{-2,+2}$('cake') = {['cook','the','twenty', 'minutes']}
Context$_{-2,+2}$('tire') = {['broke','a','hitting', 'a'], ['changed', 'the']}

Word-Context matrix

		Context words						
		…	cook	eat	…	changed	broke	…
Words	cake	…	10	20	…	0	0	…
	steak	…	12	22	…	0	0	…
	bim	…	7	10	…	0	0	…
	engine	…	0	0	…	3	10	…
	tire	…	0	0	…	10	1	…
	…	…	…	…	…	…	…	…

$Words \equiv Context\ words$

Context vector

Rows of Word-Context matrix capture similarity of use, and thus some syntactic/semantic similarity.

Rows of the Word-Context matrix can be used as non-orthogonal representations of words.

We can use them as *v(f)* to build a more "semantic" representation of documents

$$v(d) = \sum_{f \in d} w_{fd} v(f)$$

yet such vectors are still high dimensional and largely sparse.

Dense representations

We can learn a projection of feature vectors *v(f)* into a *low dimensional* space R^k, $k \ll |F|$, of *continuous space word representations* (i.e. word embeddings).

$$Embed\colon R^{|F|} \rightarrow R^k$$

$$Embed(v(f)) = e(f)$$

We force features to share dimensions on a reduced dense space

↓

Let’s group/align/project them by their syntactic/semantic similarities!

SVD

Singular Value Decomposition is a decomposition method of a matrix X of size $m{\cdot}n$ into three matrices $U\Sigma V^*$, where:

U is an orthonormal matrix of size $m{\cdot}n$

Σ is a diagonal matrix of size $n{\cdot}n$, with values $\sigma_1, \sigma_2 \ldots \sigma_n$

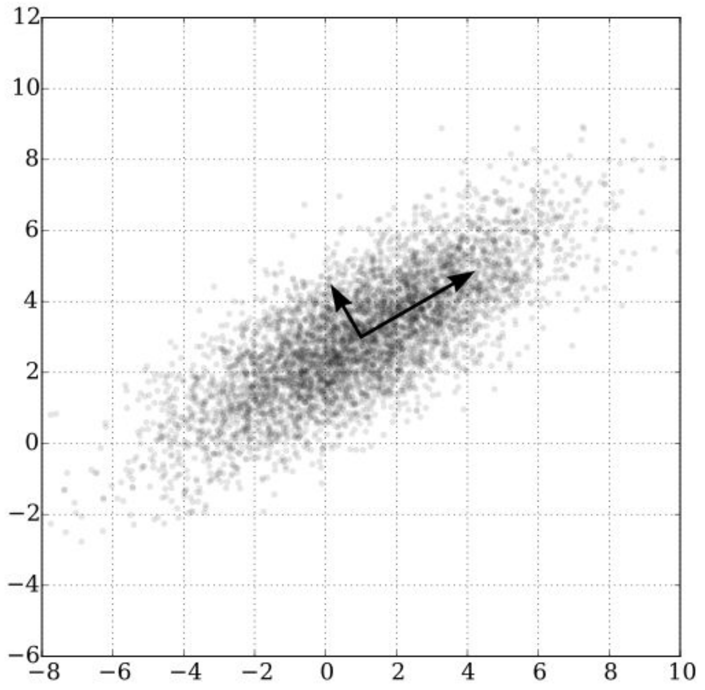
V is an orthonormal matrix of size $n{\cdot}n$, V^* is its conjugate transpose

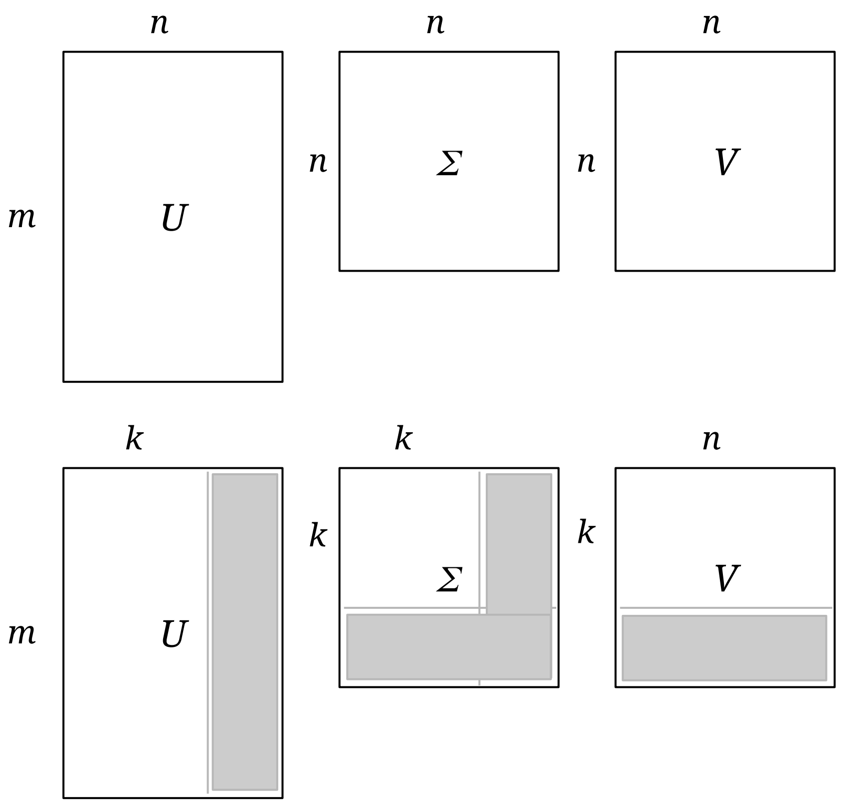
$\sigma_1, \sigma_2 \ldots \sigma_n$ of Σ are the singular values of X, sorted by decreasing magnitude.

Keeping the top k values is a least-square approximation of X

Rows of U_k of size $m{\cdot}k$ are the dense representations of the features

SVD





By Nicoguaro (Own work) [CC BY 4.0], via Wikimedia Commons

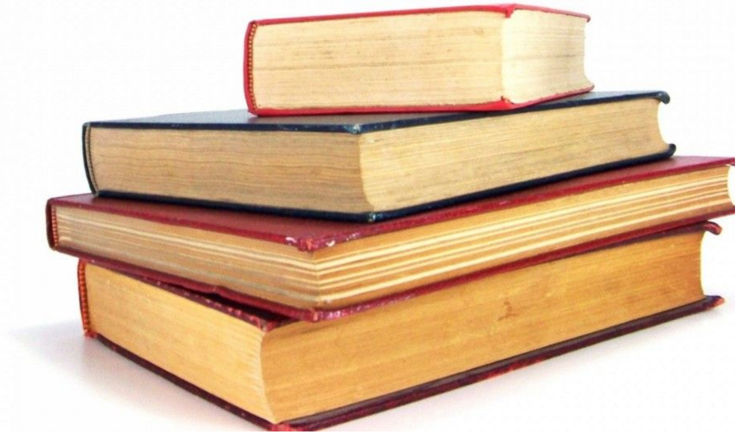
Neural Language Models

Neural models

We can use a neural network to model the distributional hypothesis, providing it with examples of use of terms in context, i.e., **a lot of text**.

Neural models started with simple formulations, already showing to be at par with previous statistical models.

Recent complex, and **large**, neural models have shown that language models not only can capture knowledge about the use of language, but also common knowledge about the world.



Word2Vec

What are the missing words?

machine learning uses a ___________ set to learn a model
the distance between ________ and the Sun is

Telling the missing word given a context is a prediction task on which we can train a ML model on.

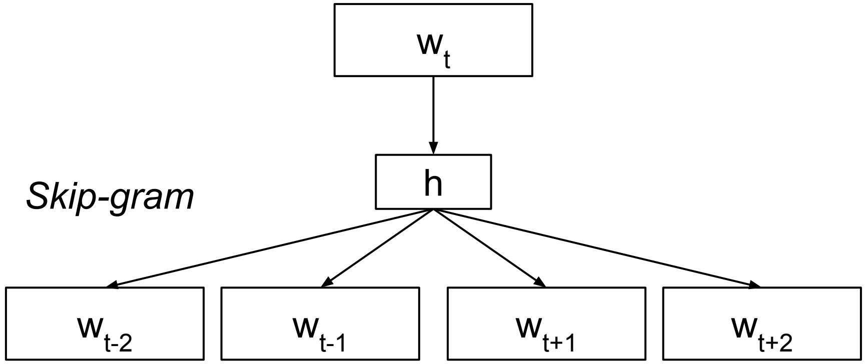
We need just (a lot of) text as the training data, no need for additional human-labeled data!

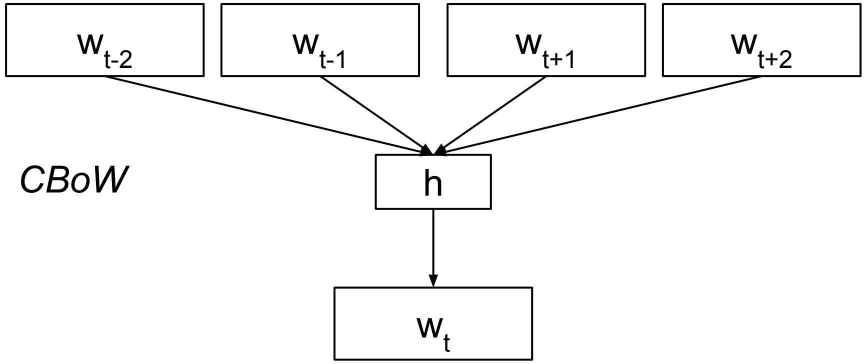
The context window size can change, longer windows capture more semantic, less syntax (typically 5-10 words), similarly to n-gram models.

Word2Vec

Skip-gram and CBoW models of Word2Vec define tasks of **predicting** a *context from a word* (Skip-gram) or a *word from its context* (CBoW).

They are both implemented as a two-layers linear **neural network** in which input and output words one-hot representations which are encoded/decoded into/from a dense representation of smaller dimensionality.





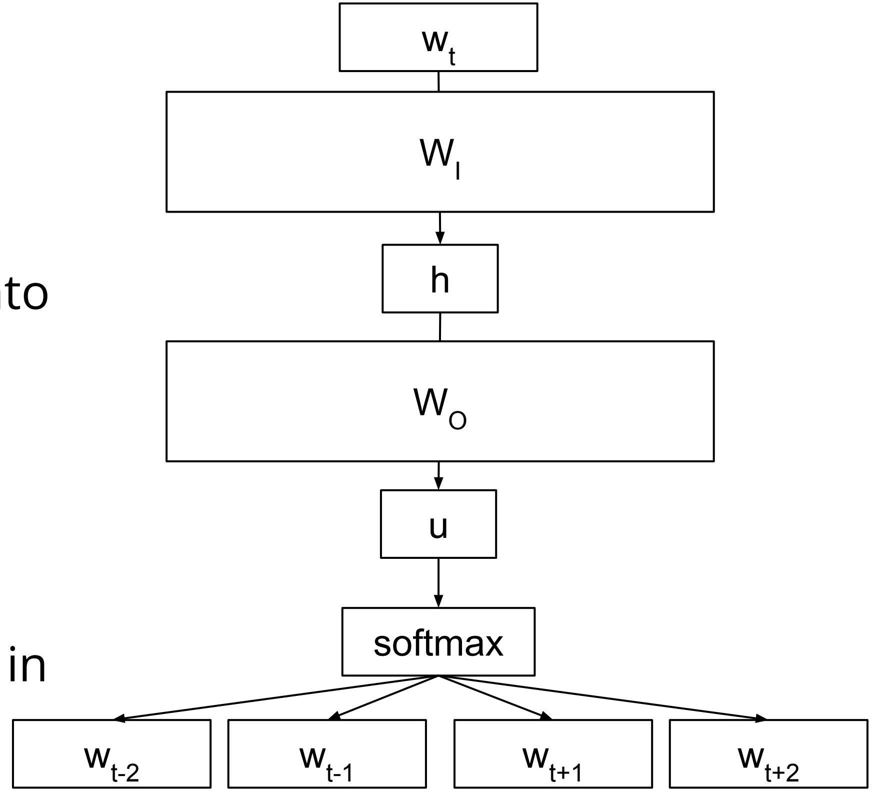
Skip-gram

w vectors are high dimensional, $|F|$

h is low dimensional (it is the size of the embedding space)

W_I matrix is $|F| \cdot |h|$. It encodes a word into a hidden representation.
Each row of W_I defines the embedding of the a word.

W_O matrix is $|h| \cdot |F|$. It defines the embeddings of words when they appears in contexts.



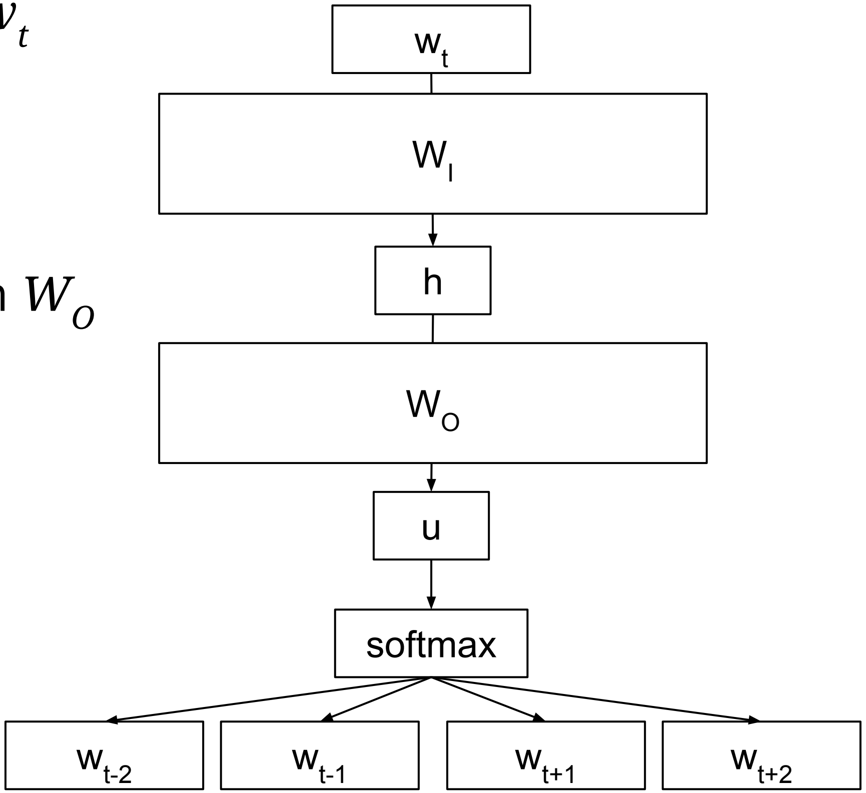
Skip-gram

$h = w_t W_I \leftarrow h$ is the embedding of word w_t

$u = h\ W_O \leftarrow u_i$ is the similarity of h with context embedding of w_i in W_O

Softmax converts u to a probability distribution y:

$$y_i = exp(u_i)/\sum_{j \in F} exp(u_j)$$



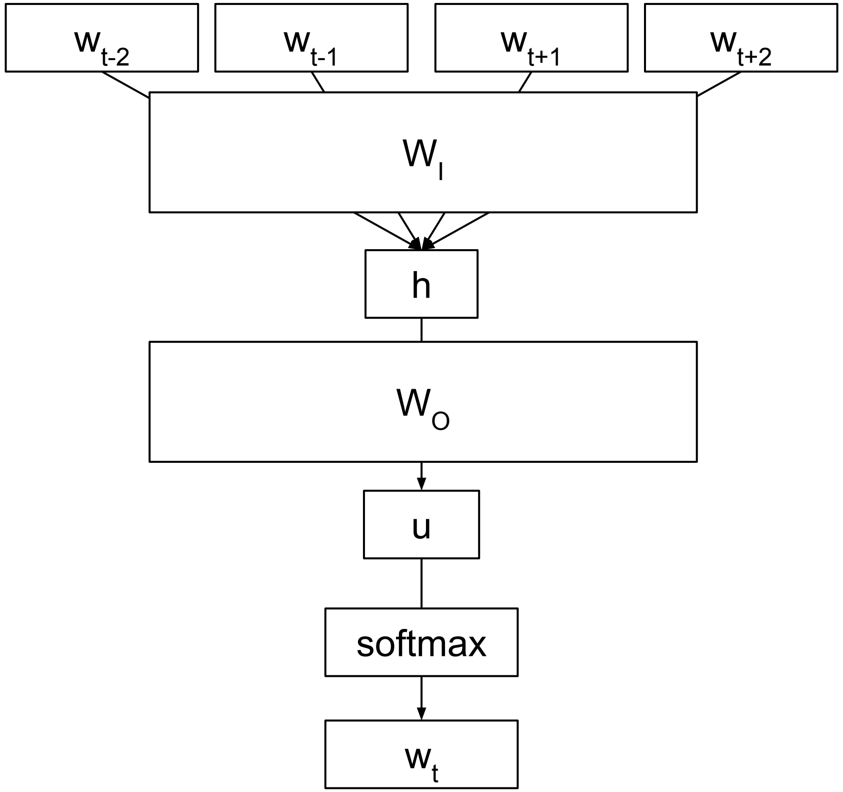
CBoW

CBoW stands for Continuous Bag of Word.

It's a mirror formulation of the skip-gram model, as context words are used to predict a target word.

h is the average of the embedding for the input context words.

u_i is the similarity of h with the word embedding w_t in W_O



Computing embeddings

The training cost of Word2Vec is linear in the size of the input.

The training algorithm works well in parallel, given the sparsity of words in contexts and the use of negative sampling. The probability of concurrent update of the same values by two processes is minimal → let's ignore it when it happens (a.k.a., asynchronous stochastic gradient descent).

Can be halted/restarted at any time.

The model can be updated with any data (concept drift/ domain adaptation).

Computing embeddings

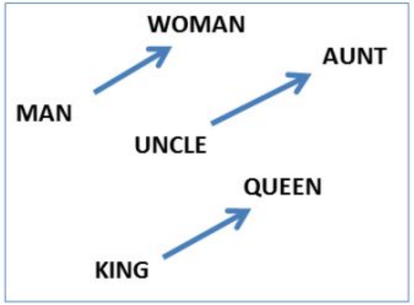
Gensim provides an efficient and detailed implementation.

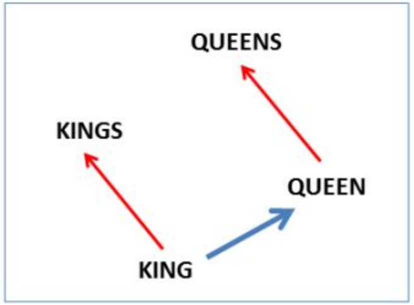
```
sentences = [['this','is','a','sentence'],
             ['this','is','another','sentence']]

from gensim.models import Word2Vec
model = Word2Vec(sentences)
```

This is a clean implementation of skip-grams using pytorch.

Testing embeddings

Testing if embeddings capture syntactic/semantic properties.





Analogy test:

Paris	stands to	France	as	Rome	stands to	?
Writer	stands to	book	as	painter	stands to	?
Cat	stands to	cats	as	mouse	stands to	?

$$e('France') - e('Paris') + e('Rome') \sim e('Italy')$$

$$a : b = c : d$$

$$d = \arg\max_x \frac{(e(b)-e(a)+e(c))^T e(x)}{||e(b)-e(a)+e(c)||}$$

The impact of training data

The source on which a model is trained determines what semantic is captured.

WIKI	BOOKS	WIKI	BOOKS
sega		chianti	
dreamcast	motosega	radda	merlot
genesis	seghe	gaiole	lambrusco
megadrive	seghetto	montespertoli	grignolino
snes	trapano	carmignano	sangiovese
nintendo	smerigliatrice	greve	vermentino
sonic	segare	castellina	sauvignon

FastText word representation

FastText extends the W2V embedding model to *ngrams of the words*.

The word "goodbye" is also represented with a set of ngrams, including start of word and end of word ('<' and '>':

"<go", "goo","ood", "odb", "dby", "bye", "ye>"

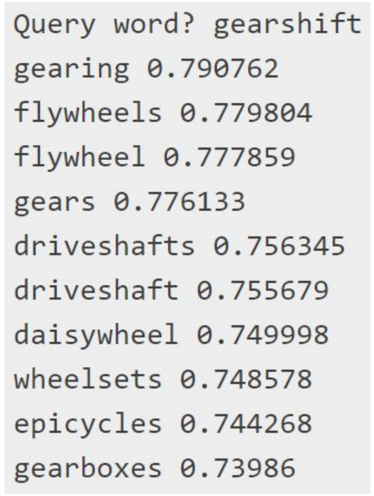
The length of the ngram is a parameter.

Typically all n-grams of length from 3 to 6 are included.

FastText word representation

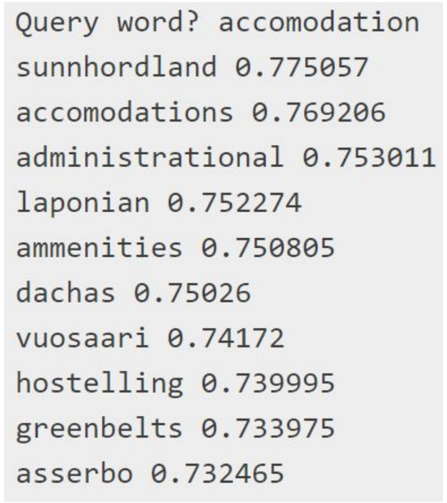
The embedding of a word is determined as the sum of the embedding of the word and of the embedding of its ngrams.

Subword information allows to give an embedding to OOV words.

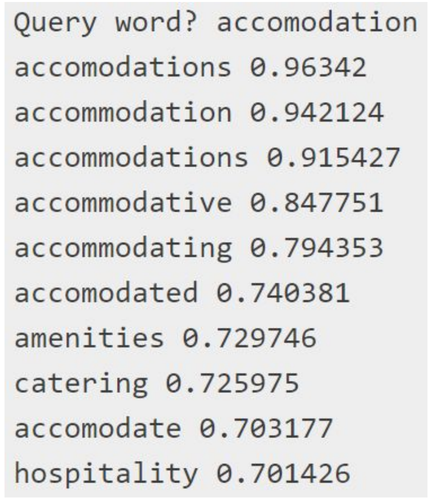


```
Query word? gearshift
gearing 0.790762
flywheels 0.779804
flywheel 0.777859
gears 0.776133
driveshafts 0.756345
driveshaft 0.755679
daisywheel 0.749998
wheelsets 0.748578
epicycles 0.744268
gearboxes 0.73986
```

Subword information improves the quality of misspelled words.



```
Query word? accomodation
sunnhordland 0.775057
accomodations 0.769206
administrational 0.753011
laponian 0.752274
ammenities 0.750805
dachas 0.75026
vuosaari 0.74172
hostelling 0.739995
greenbelts 0.733975
asserbo 0.732465
```



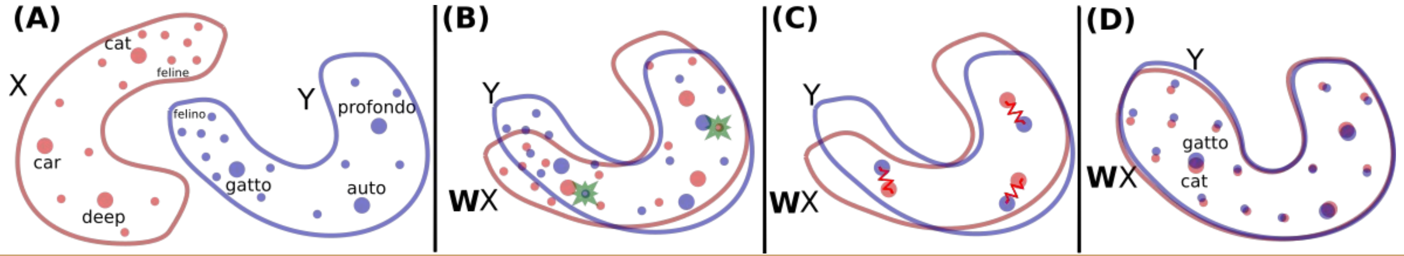
```
Query word? accomodation
accomodations 0.96342
accommodation 0.942124
accommodations 0.915427
accommodative 0.847751
accommodating 0.794353
accomodated 0.740381
amenities 0.729746
catering 0.725975
accomodate 0.703177
hospitality 0.701426
```

Pretrained embeddings for 200+ languages.

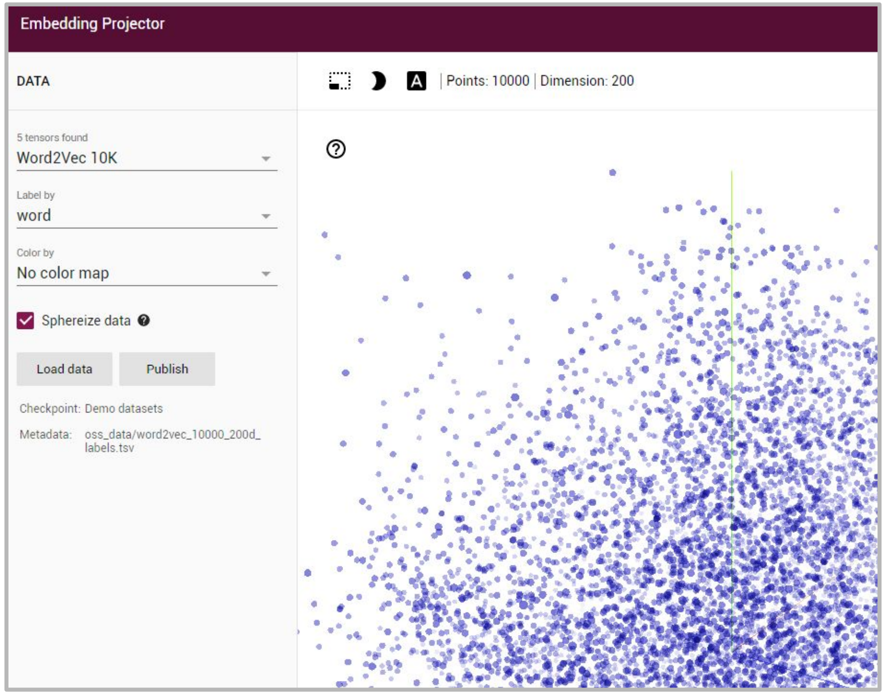
Multilingual Embeddings

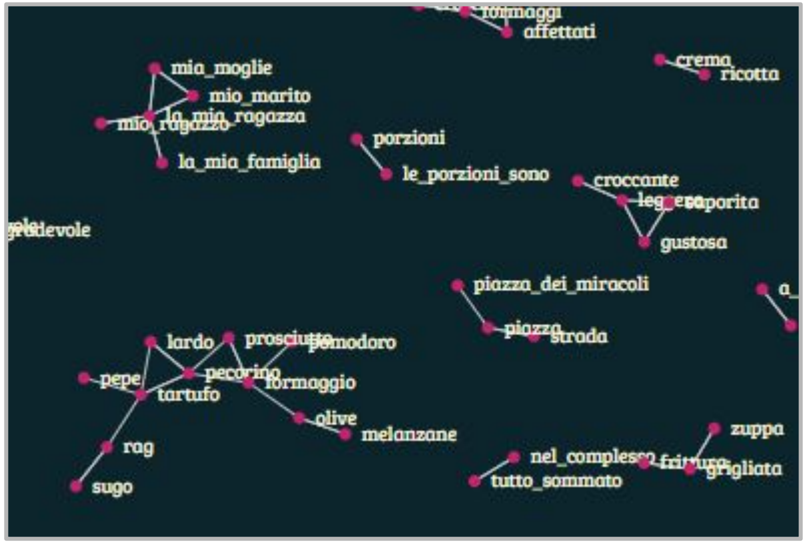
MUSE (Multilingual Unsupervised and Supervised Embeddings) aligns language models for different languages using two distinct approaches:

- *supervised*: using a *bilingual dictionary* (or same string words) to transform one space into the other, so that a word in one language is projected to the position of its translation in the other language.
- *unsupervised*: using *adversarial learning* to find a space transformation that matched the distribution of vectors in the two space (without looking at the actual words).



Exploring embeddings





Word embeddings to documents

How can we represent *a document* in an embedding space?

A document is composed by a set of words, so we can t**ake their embeddings and combine them in some way** to obtain a single document embedding vector (e.g., by taking the **average** or **max** of all the word embeddings).

We can extend the Word2Vec model to **explicitly model the embeddings of document**, e.g., the **Doc2Vec** model.

We can **directly use the word embeddings as the first layer in a neural network**, letting the neural network learn to implicitly model the content of the document.

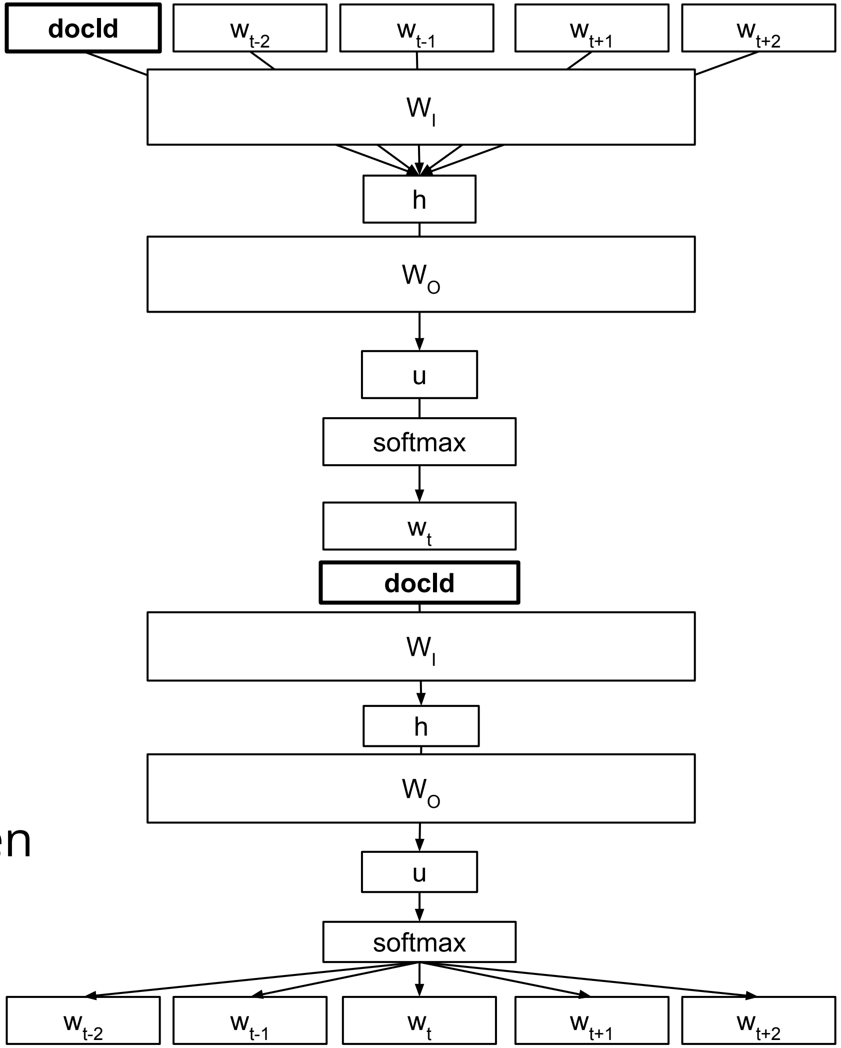
Doc2Vec

Proposed by Le and Mikolov, Doc2Vec extends Word2Vec by adding input dimensions for identifiers of documents.

W_I matrix is $(|D|+|F|)\cdot|h|$.

Documents ids are projected in the same space of words.

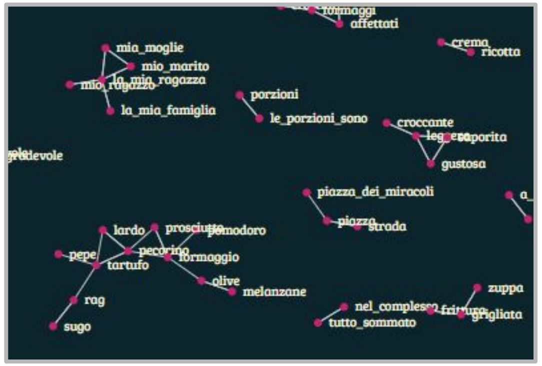
The trained model can be used to infer document embeddings for previously unseen documents - by passing the words composing them.



Exploring embeddings

Documents embedding can be used as vectorial representations of documents in any task.

When the document id is associated to more than one actual document (e.g., id of a product with multiple reviews), Doc2Vec is a great tool to model similarity between objects with multiple descriptions.



Embeddings in neural networks

An embedding layer in neural networks is typically the first layer of the network.

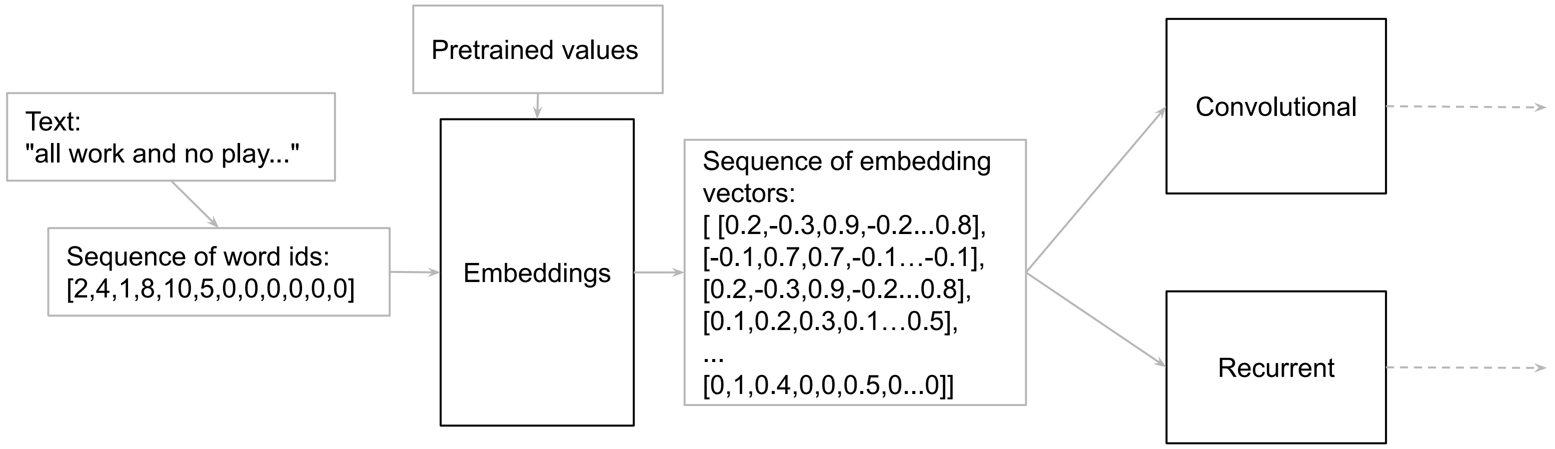
It consists of a matrix W of size $|F| \cdot n$, where n is the size of the embedding space.

It maps words to dense representations.

It can be initialized with random weights or *pretrained* embeddings.

During learning weights can be kept fixed (it makes sense only when using pretrained weights) or updated, to adapt embeddings to the task.

Embeddings in neural networks



Example:
https://github.com/fchollet/keras/blob/master/examples/imdb_cnn.py

Another example:
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Deep Neural Language Models

Recurrent Neural Networks

$$\begin{tikzcd}
x \arrow[r, "U"] & s \arrow[r, "V"] \arrow[loop below, "W"] & o
\end{tikzcd}$$

A Recurrent Neural Network (RNN) is a neural network in which connections between units form a directed cycle.

$$\begin{tikzcd}
x_{t-1} \arrow[r, "U"] & s_{t-1} \arrow[r, "V"] \arrow[d, "W"'] & o_{t-1} \\
x_{t} \arrow[r, "U"] & s_{t} \arrow[r, "V"] \arrow[d, "W"'] & o_{t} \\
x_{t+1} \arrow[r, "U"] & s_{t+1} \arrow[r, "V"] \arrow[d, "W"'] & o_{t+1} \\
 & {} &
\end{tikzcd}$$

Cycles allow the network to have a memory of previous inputs, combining it with current input.

RNNs are fit to process sequences, such as text.

Text can be seen as a sequence of values at many different levels: characters, words, phrases…

Suggested read

Char-level LM & text generation

RNNs are key tools in the implementation of many NLP applications, e.g., machine translation, summarization, or image captioning.

A RNN can be used to learn a language model that predicts the next character from the sequence of previous ones.

The typical RNN node that is used is an Long Short Term Memory (LSTM), which is robust to typical issues of RNNs.

$$\begin{tikzcd}
\mathbf{a} \arrow[r, "U"] & s \arrow[r, "V"] \arrow[d, "W"'] & \mathbf{n} \\
\mathbf{n} \arrow[r, "U"] & s \arrow[r, "V"] \arrow[d, "W"'] & {'\,'} \\
{'\,'} \arrow[r, "U"] & s \arrow[r, "V"] \arrow[d, "W"'] & \mathbf{a} \\
\mathbf{a} \arrow[r, "U"] & s \arrow[r, "V"] \arrow[d, "W"'] & \mathbf{p} \\
\mathbf{p} \arrow[r, "U"] & s \arrow[r, "V"] \arrow[d, "W"'] & \mathbf{p} \\
\mathbf{p} \arrow[r, "U"] & s \arrow[r, "V"] \arrow[d, "W"'] & \mathbf{l} \\
 & {} &
\end{tikzcd}$$

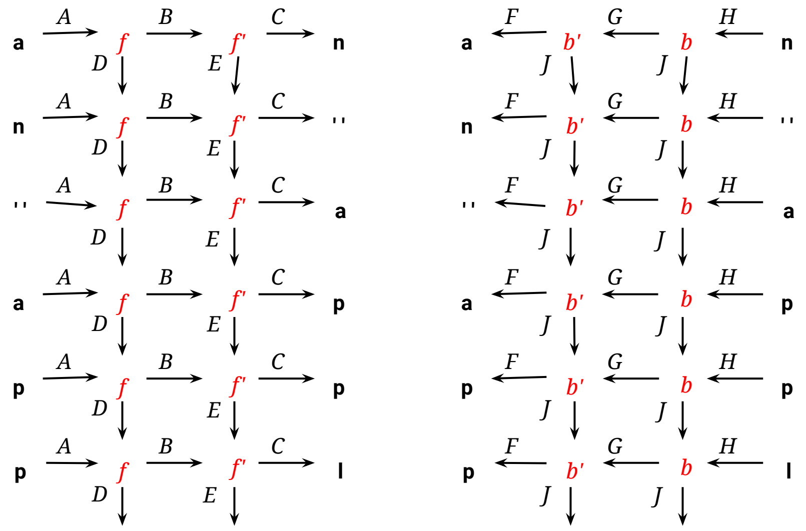
ELMo

ELMo (Embeddings from Language Models) exploits the *hidden states* of a *deep*, *bi-directional*, *character-level* LSTM to give *contextual* representations to words in a sentence.

- Contextual: representation for each word depends on the entire context in which it is used.
- Deep: word representations combine all layers of a deep pre-trained neural network.
- Character based: ELMo representations are character based, allowing the network to use morphological clues

ELMo

ELMo (Embeddings from Language Models) exploits the *hidden states* of a *deep*, *bi-directional*, *character-level* LSTM to give *contextual* representations to words in a sentence.



The embedding for a word is a *task-specific* weighted sum of the concatenation of the f,b vectors for each level of the LSTM, e.g.:

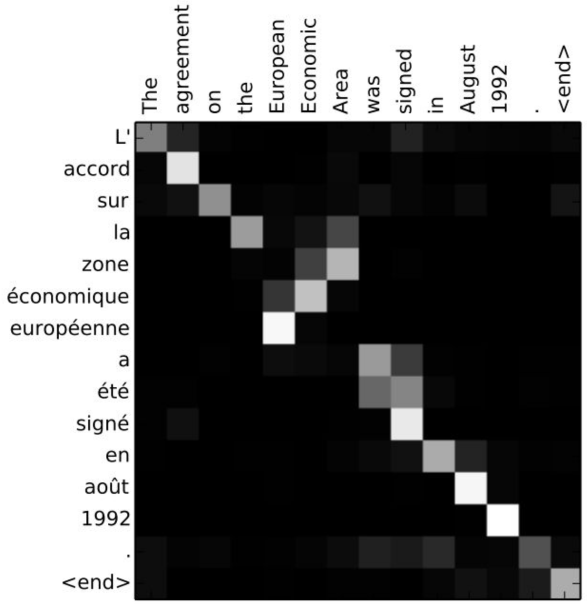
$$v(token) = w\,[f, b]_{token} + w'\,[f' ,b']_{token}$$

Attention-based models

Attention is a simple mechanism that relates the elements of two sequences so as to identify correlations among them.

Attention proved to be effective in sequence-to-sequence machine translation models.

Attention captures the contribution of each input token to determine the output tokens.



Transformer

The Transformer is a network architecture for sequence-to-sequence problems, e.g., machine translation.

It replaces the traditionally used LSTM elements with a set of encoder/decoder elements based on the attention mechanism.

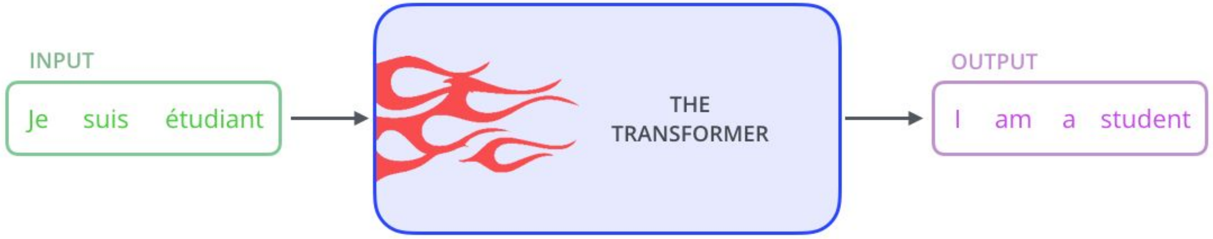
The transformer was proposed in the paper Attention is All You Need.

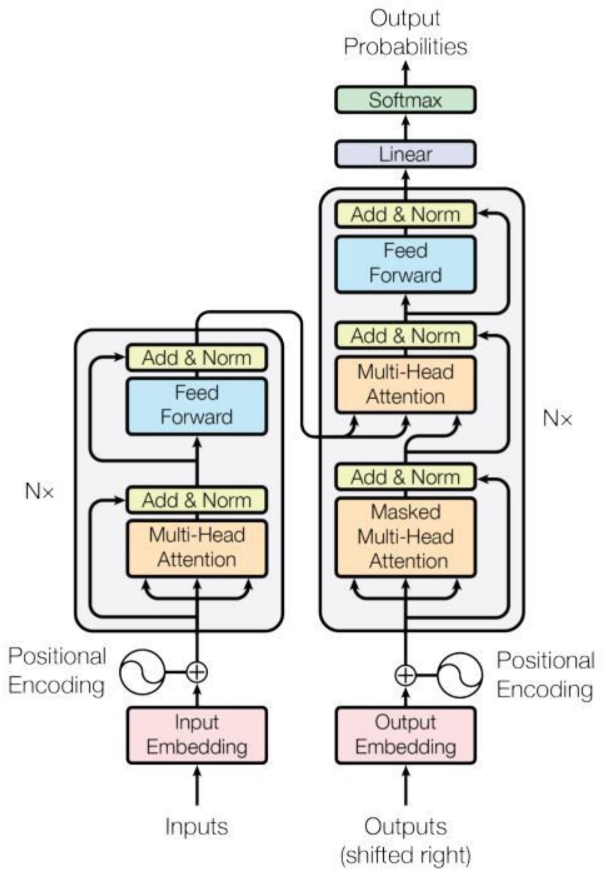
The elements of the transformer are at the base of many recently proposed language models.

Picture in the following slides are taken from The Illustrated Transformer by Jay Alammar.

Transformer

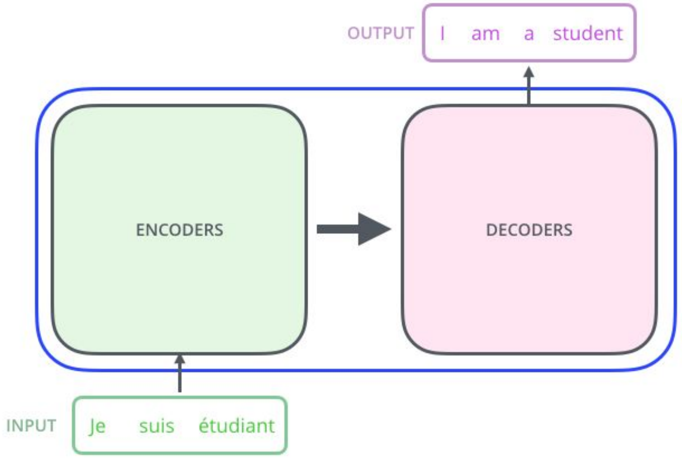
The Transformer is a network architecture for sequence-to-sequence problems, e.g., machine translation.

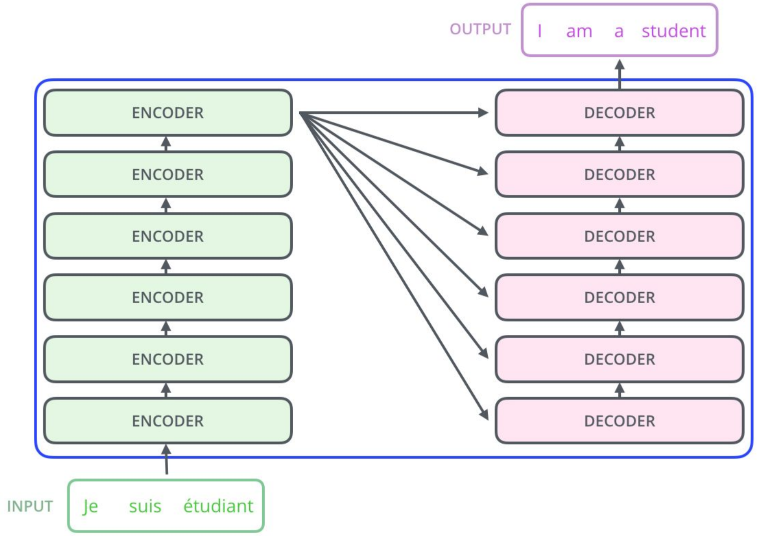




Transformer

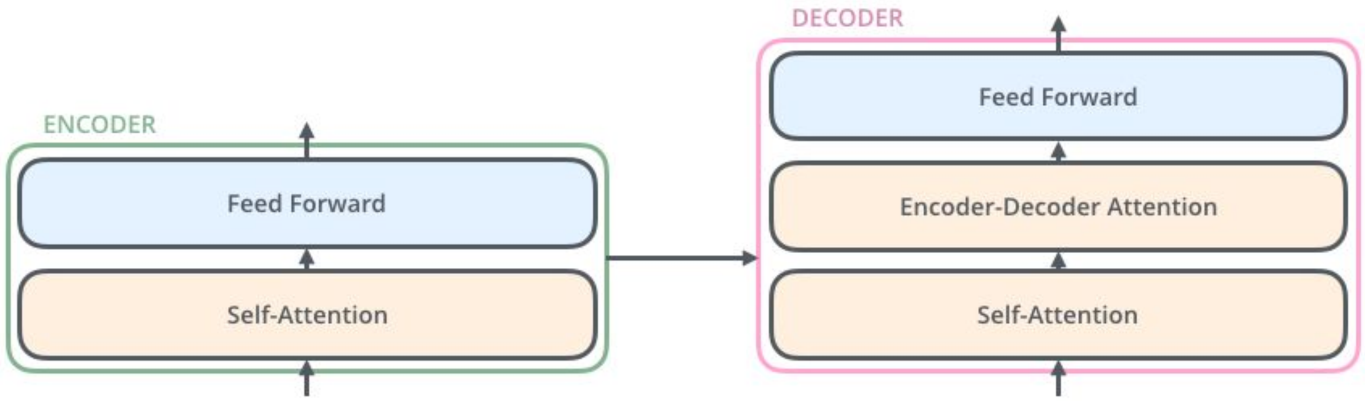
It replaces the traditionally used LSTM elements with a set of encoder/decoder elements based on the attention mechanism.





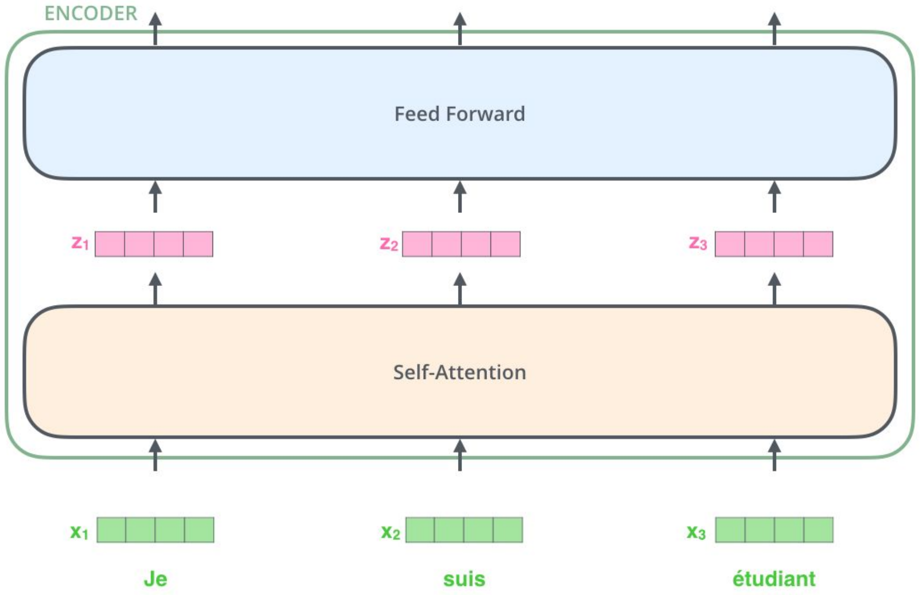
Transformer

The encoder and decoder elements exploit attention (self-attention) to combine and transform the input from lower layers (and the last encoder in the case of the decoder).



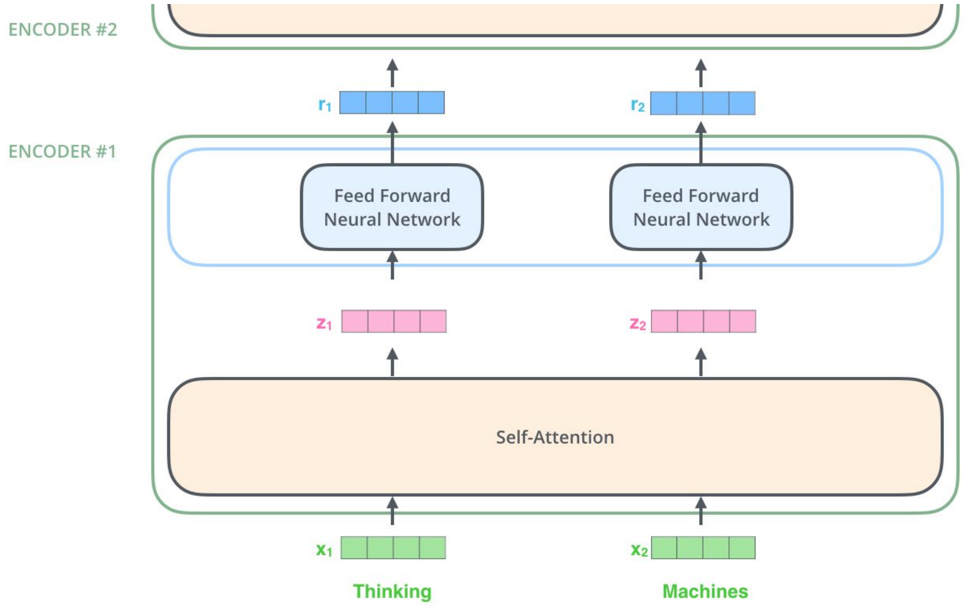
Transformer - encoder

The encoder elements exploit self-attention to combine and transform the input from lower layers.



Self-attention

Self-attention correlates all the inputs between themselves.

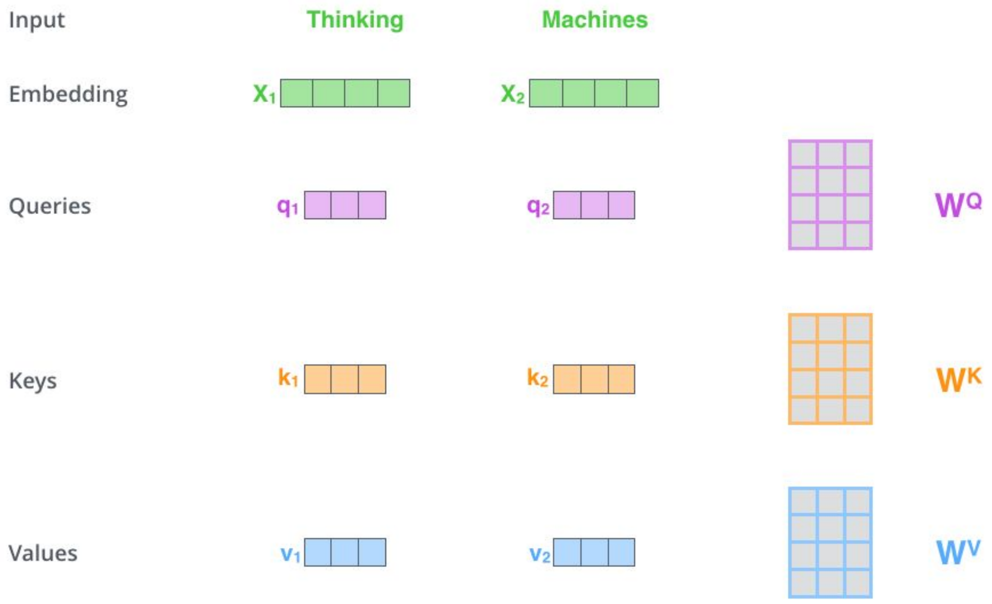


Self-attention

Self-attention first transforms the embedding vector of each token into three vectors: a query vector, a key vector and a value vector.

The query represents the token as the input.
The key is used to match the other tokens against the query token.
The value is the contribution of every token to the self attention output.

The three transformations are defined by three dedicated matrices W^Q, W^K, W^V.



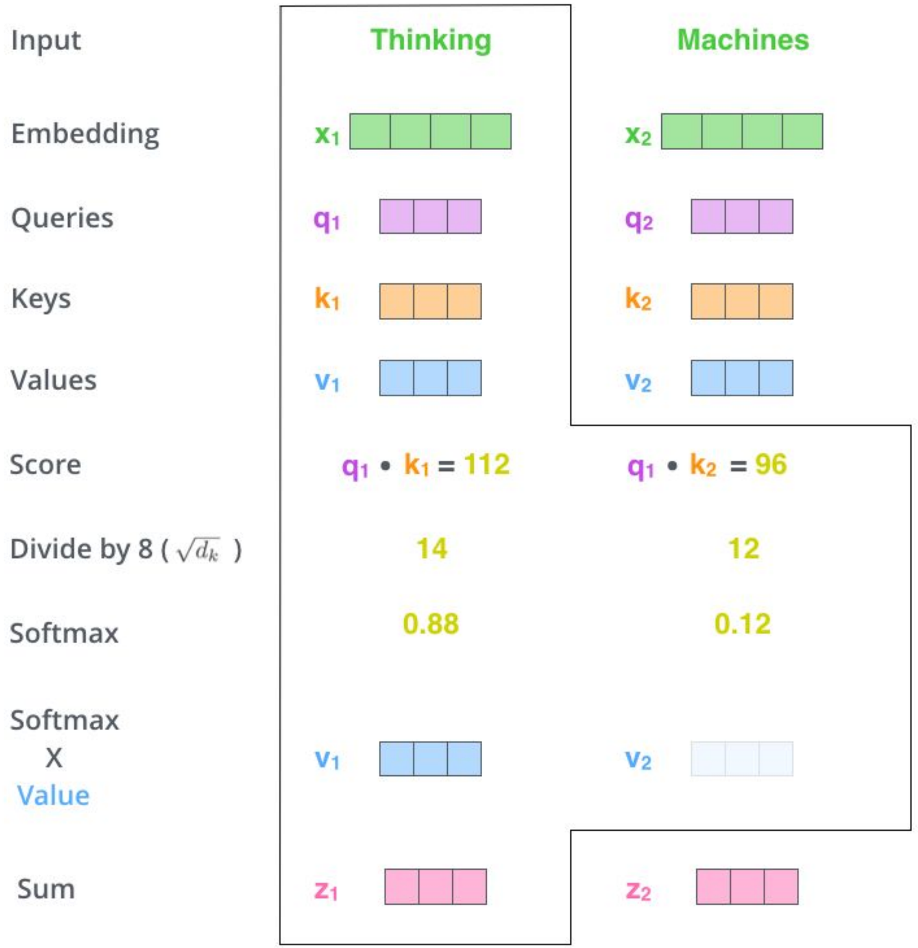
Self-attention

The query vector for a token is multiplied (dot product) with every key vector.

The factor eight stabilizes the computation.

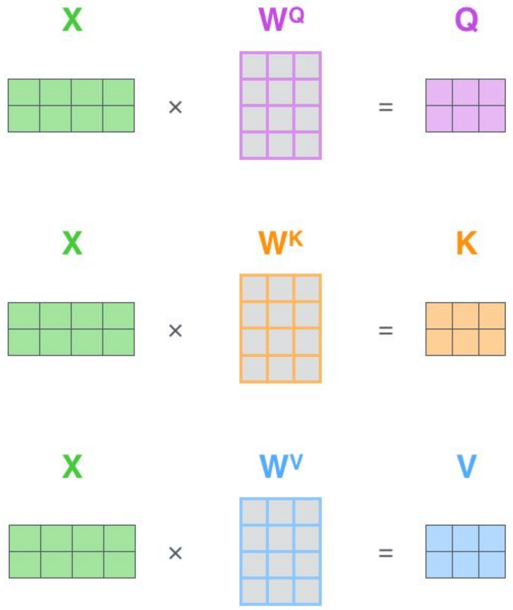
The resulting numbers are normalized with a softmax and they become the weights for a weighted sum of the value vectors.

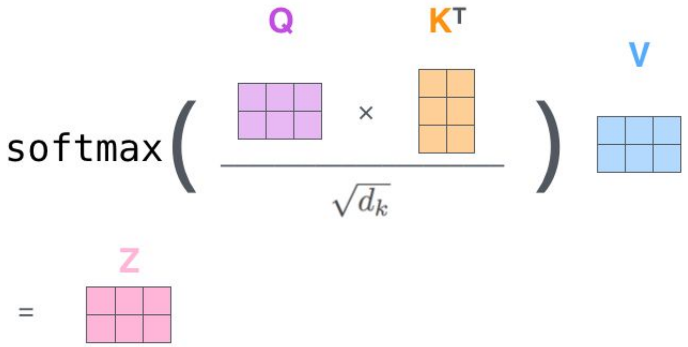
This concludes *single headed* self-attention.



Self-attention

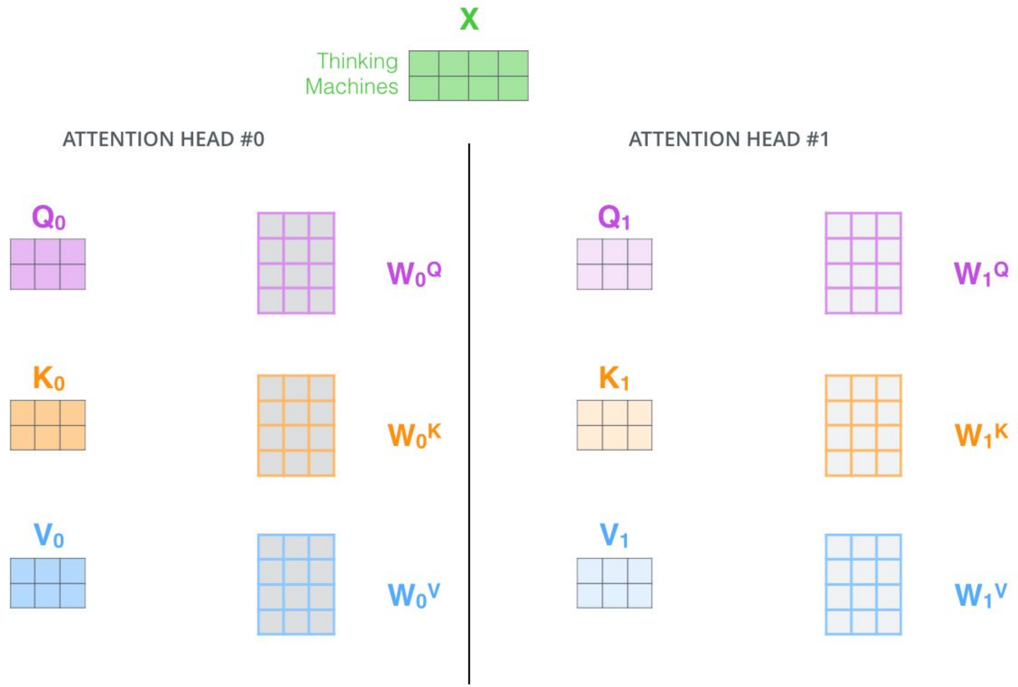
Self attention can be expressed and efficiently implemented with matrices





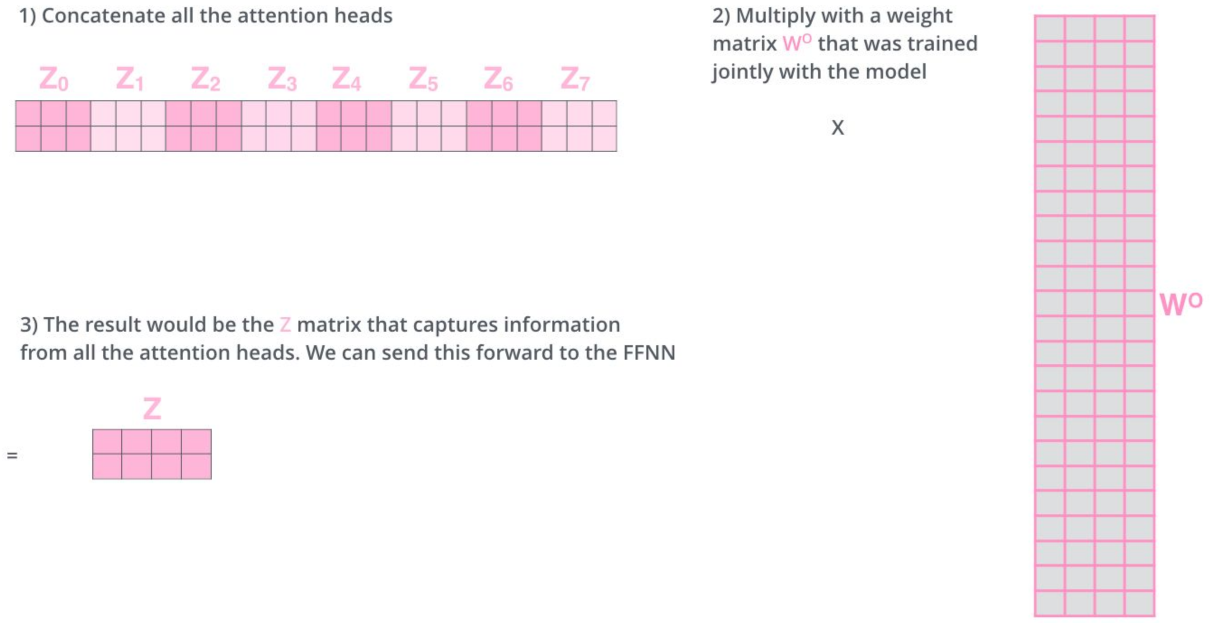
Self-attention

Different sets of matrices W^Q, W^K, W^V, can be used to capture many different relations among tokens, in a *multi-headed* attention model.

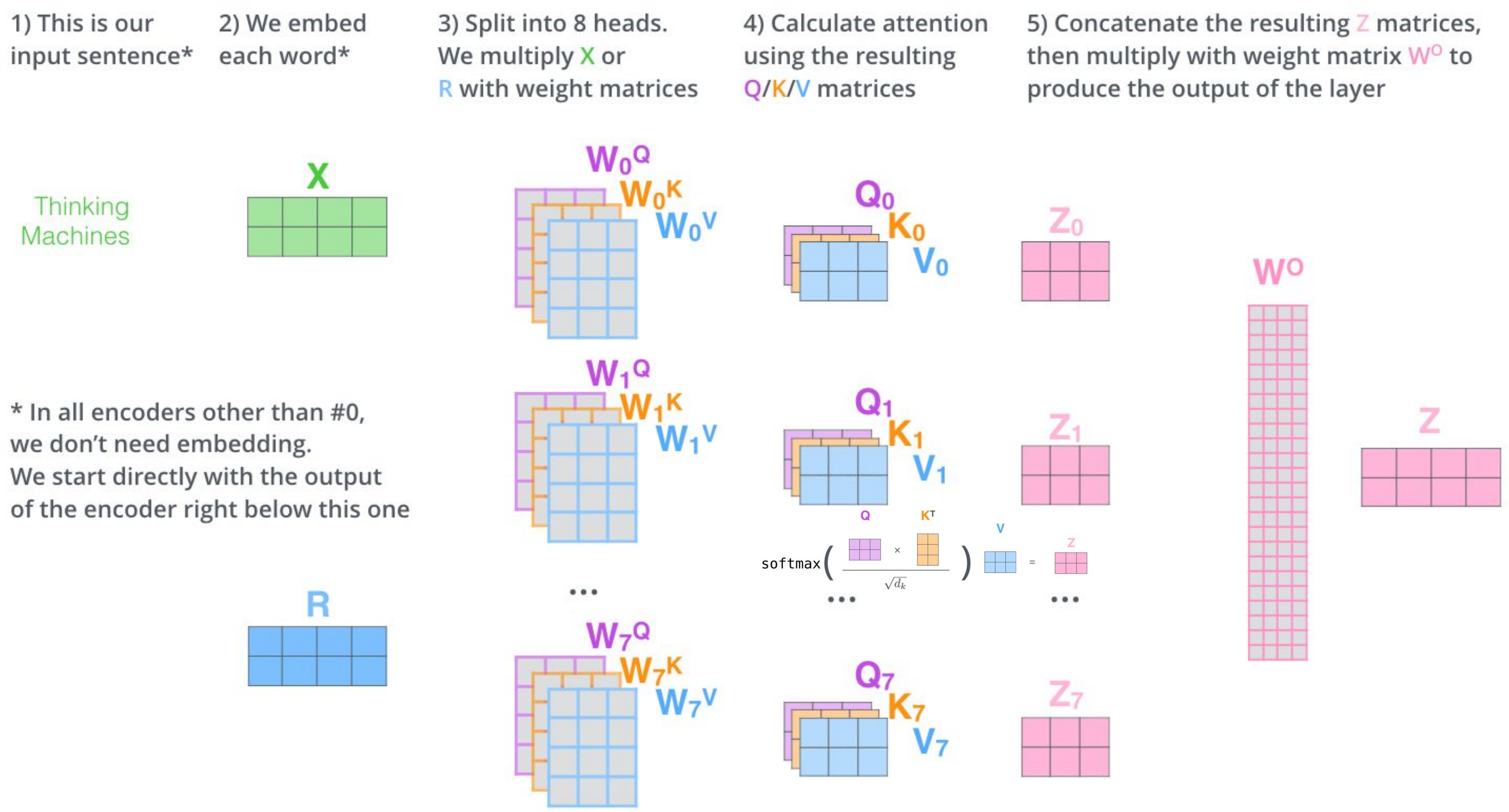


Self-attention

Outputs from each head of the attention model are combined back into a single matrix with a vector for each token.


1) Concatenate all the attention heads
Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7
2) Multiply with a weight matrix W^O that was trained jointly with the model
X
W^O
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
Z
=

Self-attention - complete view



1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

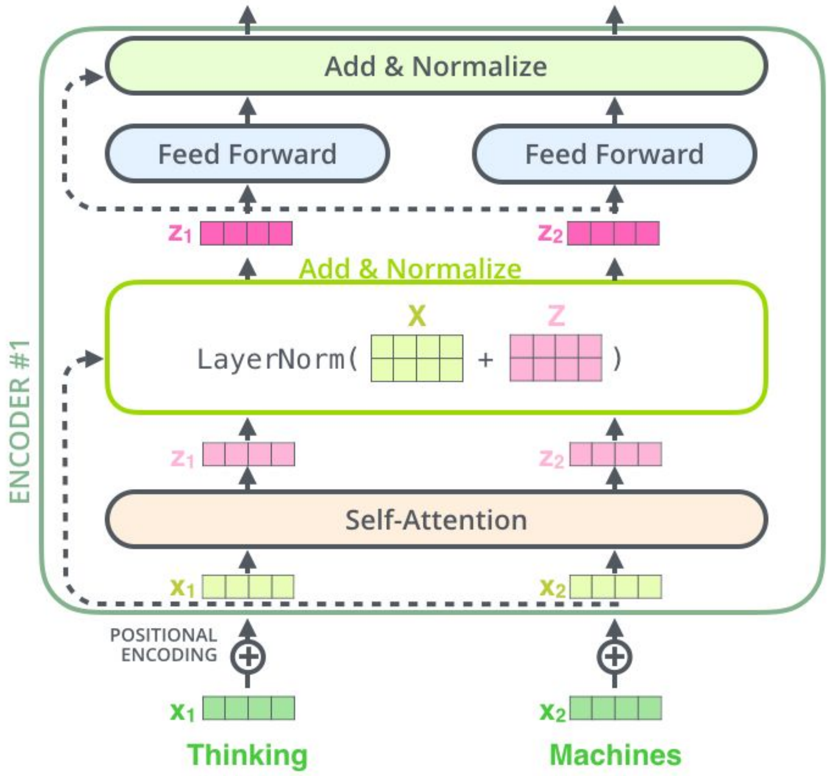
Residual connections

A residual connection simply consists in skip connections that sum the input of a layer (or more layers) to its output.

Residual connections let the network learn faster, and be more robust to the problem of vanishing gradients, allowing much deeper network to be used.

Residual connections are put around any attention or feed forward layer.

Layer normalization supports the numerical stability of the process.

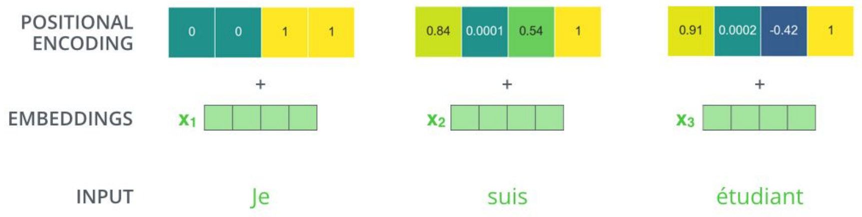


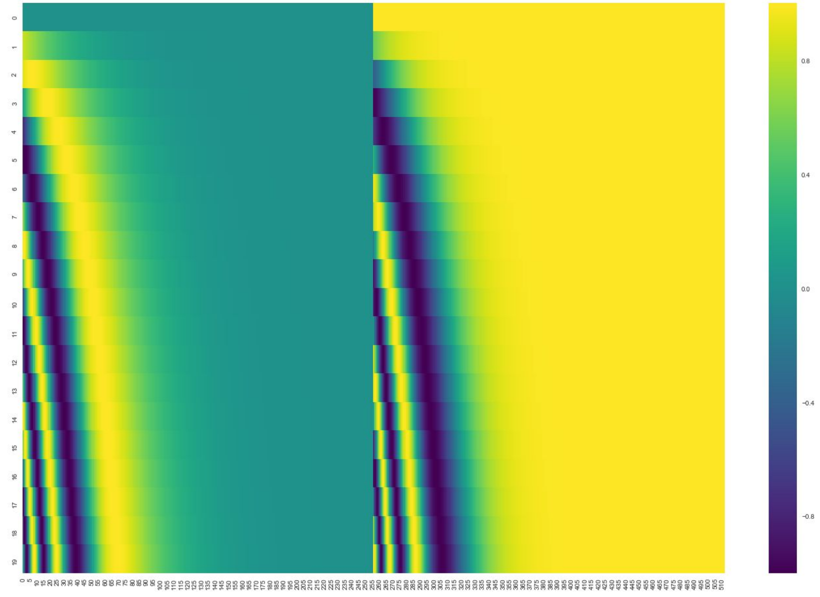
Word order

In order to take into account of the actual position of tokens in the sequence, the input embeddings are added with a *positional encoding* vector.

The positional encoding function is a linear transformation of the token representation

$$f(t,i) = \begin{cases} \sin\left(\frac{t}{10000^{\frac{i}{d}}}\right), & \text{if } i \equiv_2 0 \\ \cos\left(\frac{t}{10000^{\frac{i-1}{d}}}\right), & \text{if } i \equiv_2 1 \end{cases}$$

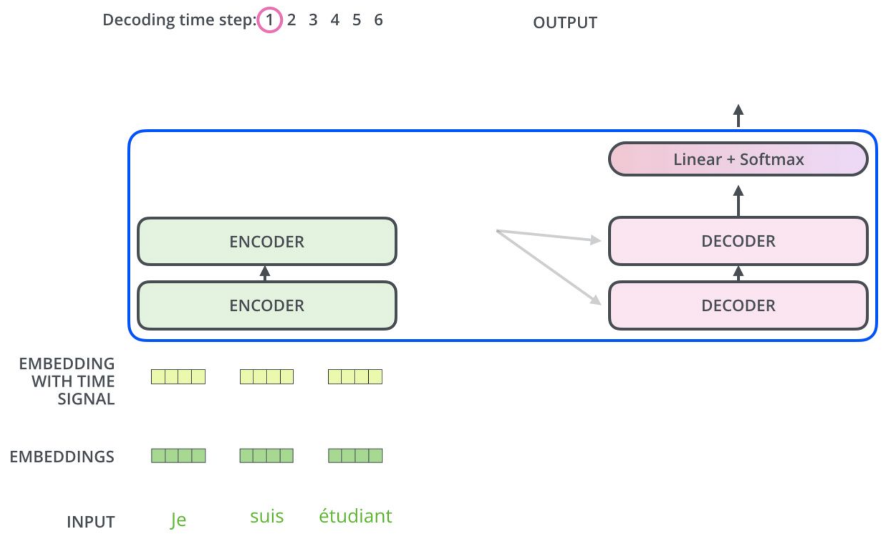




Transformer - decoder

The decoder elements exploit attention and self-attention to combine and transform the input from lower layers and the last encoder.

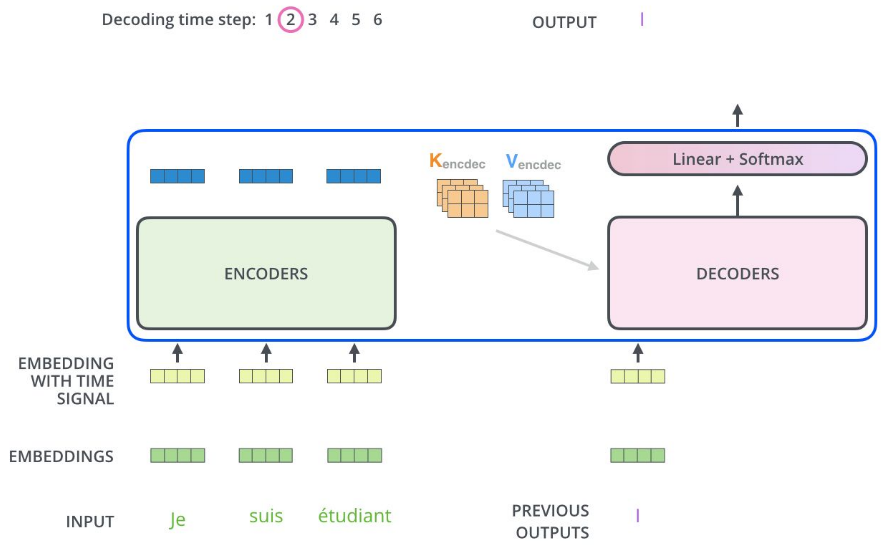
The decoder is applied repeatedly to form the output one token at a time:



Transformer - decoder

The decoder elements exploit attention and self-attention to combine and transform the input from lower layers and the last encoder.

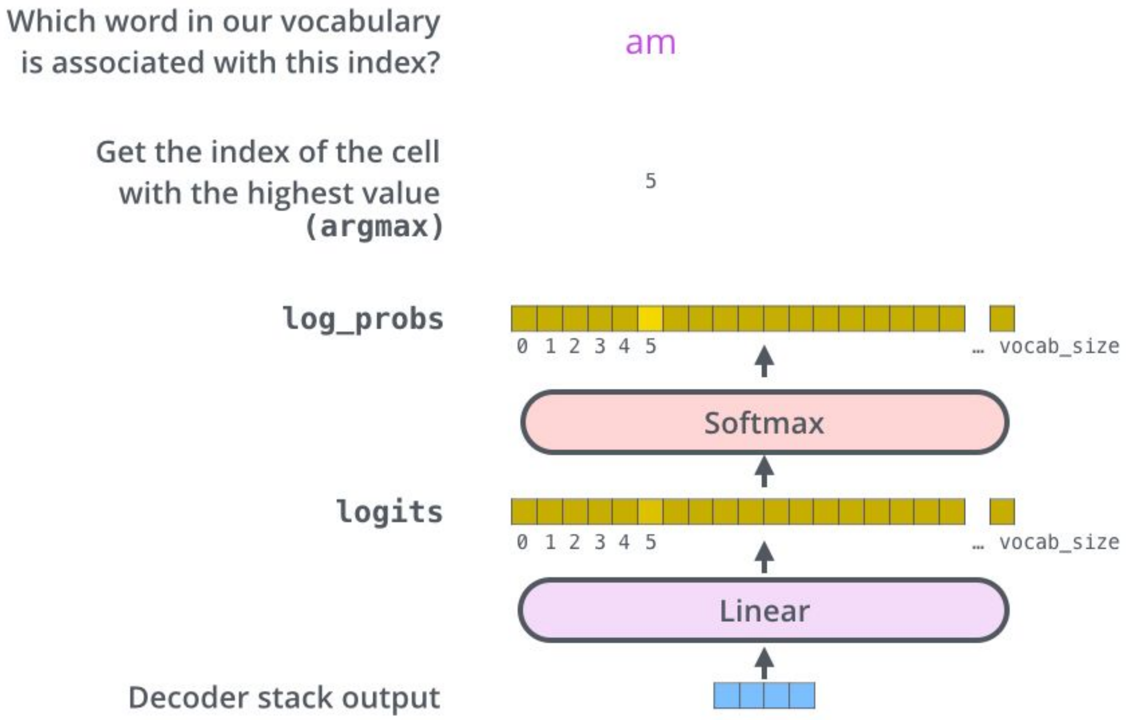
The decoder is applied repeatedly to form the output one token at a time:



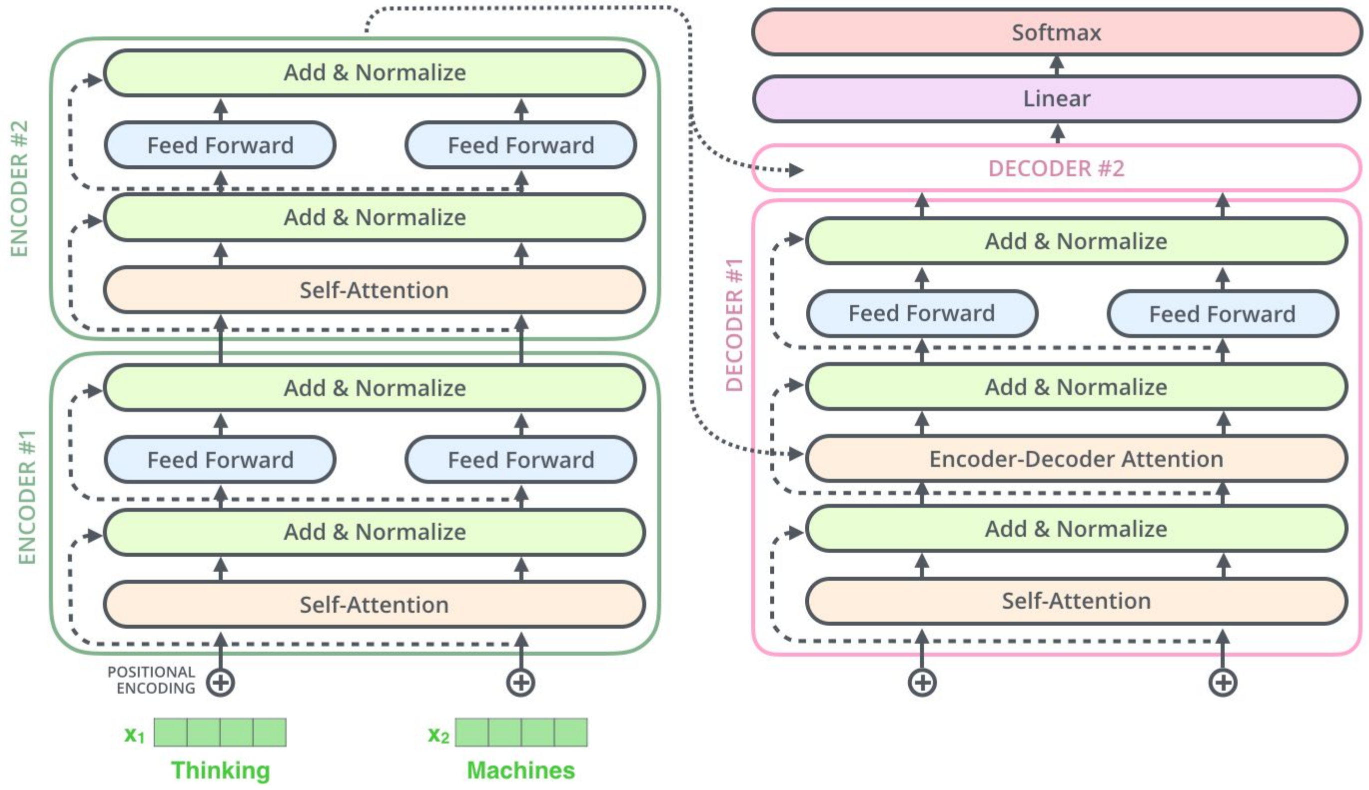
Transformer - decoder

The last linear+softmax layer converts the output of the decoder stack into probabilities over the vocabulary.

Once trained, this last element can be replaced with other structures depending on the tasks, e.g., classification, see GPT.

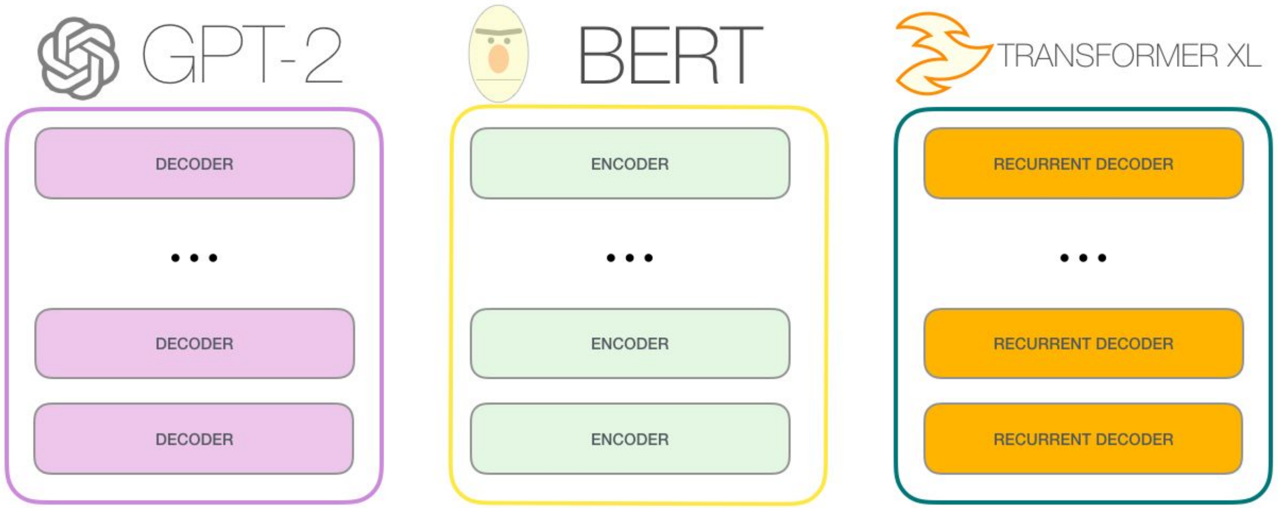

Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5
… vocab_size
Softmax
logits
0 1 2 3 4 5
… vocab_size
Linear
Decoder stack output

Transformer - complete view



Transformer-based language models

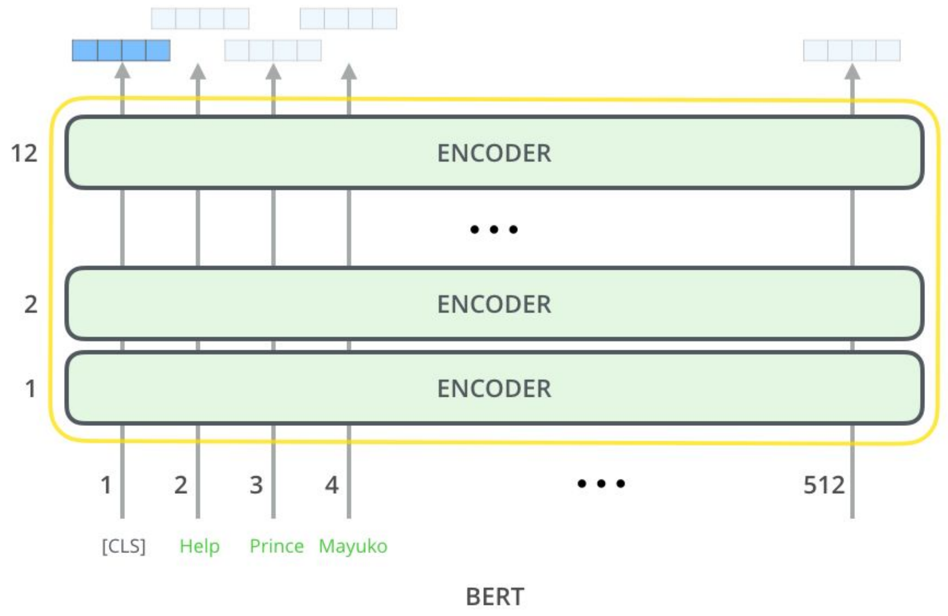
Many language models stemmed from the Transformer



BERT

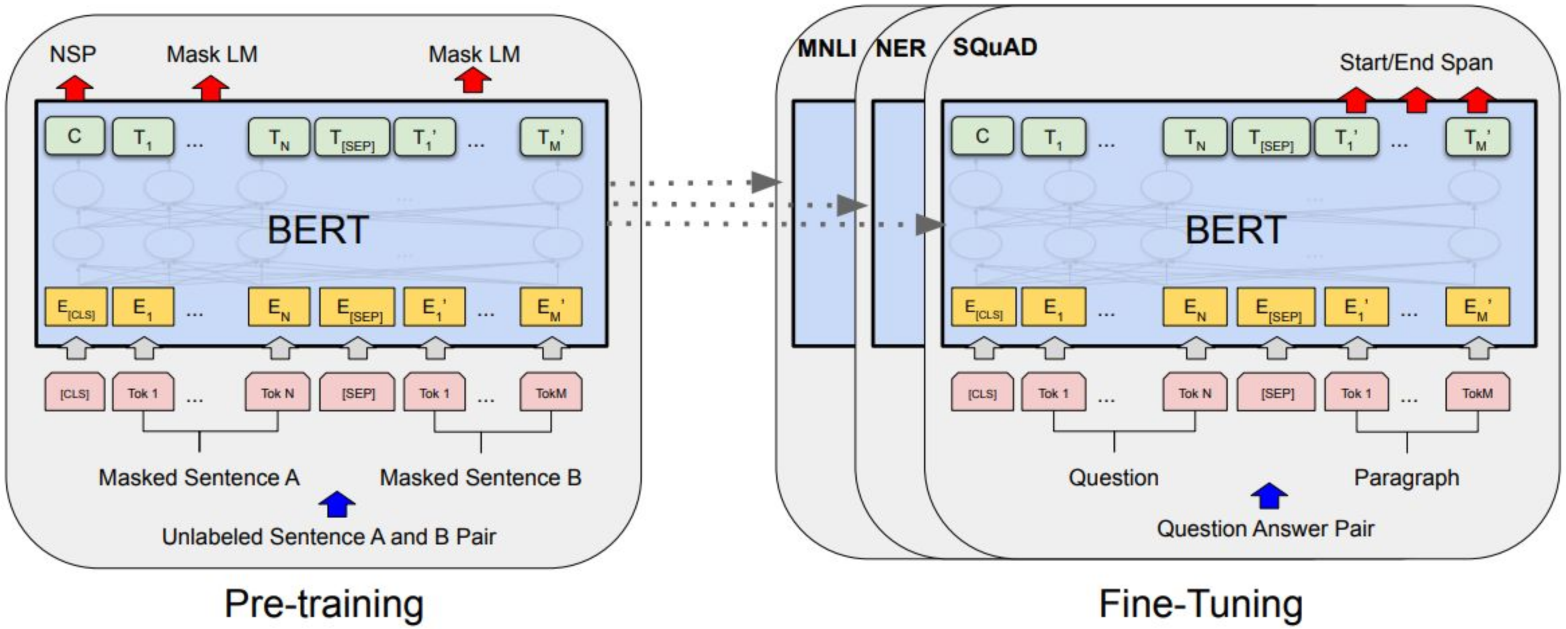
BERT (Bidirectional Encoder Representations from Transformers), uses the Transformer encoders as building blocks.

The model is bidirectional in the sense that the attention model can peek at both left and right contexts.



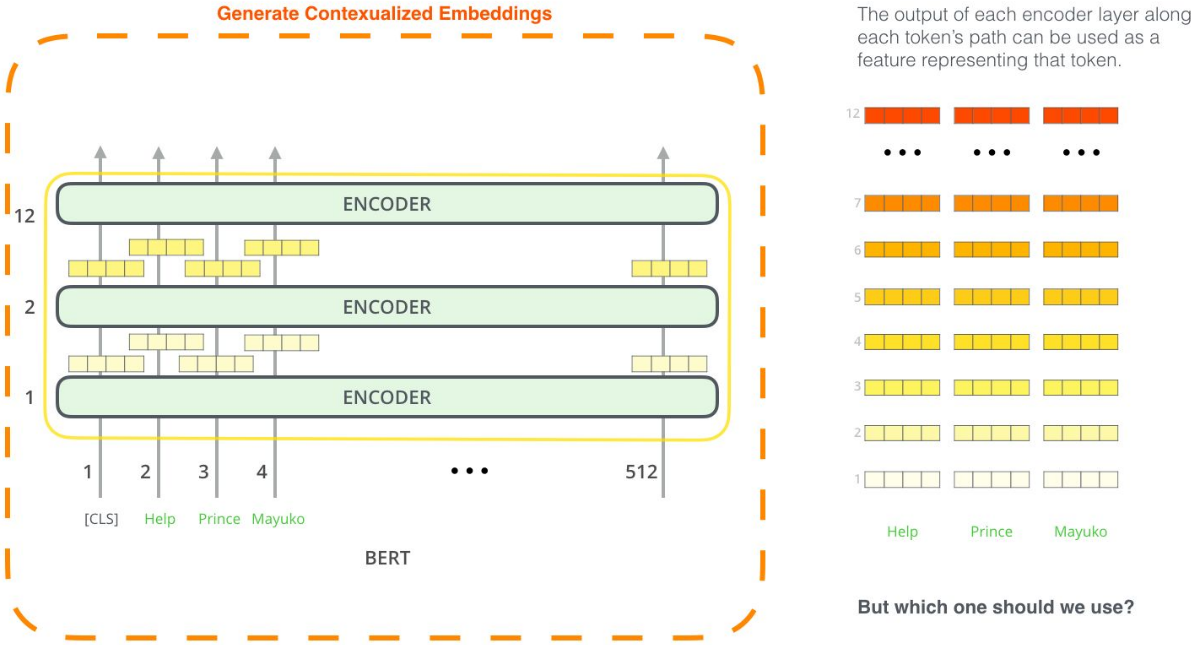
BERT

The model is pre-trained on masked word prediction and next sentence classification tasks. Fine-tuning on final task is relatively quick.



BERT

BERT produces contextualized embeddings at each level, if we use them without fine tuning, which should be used?

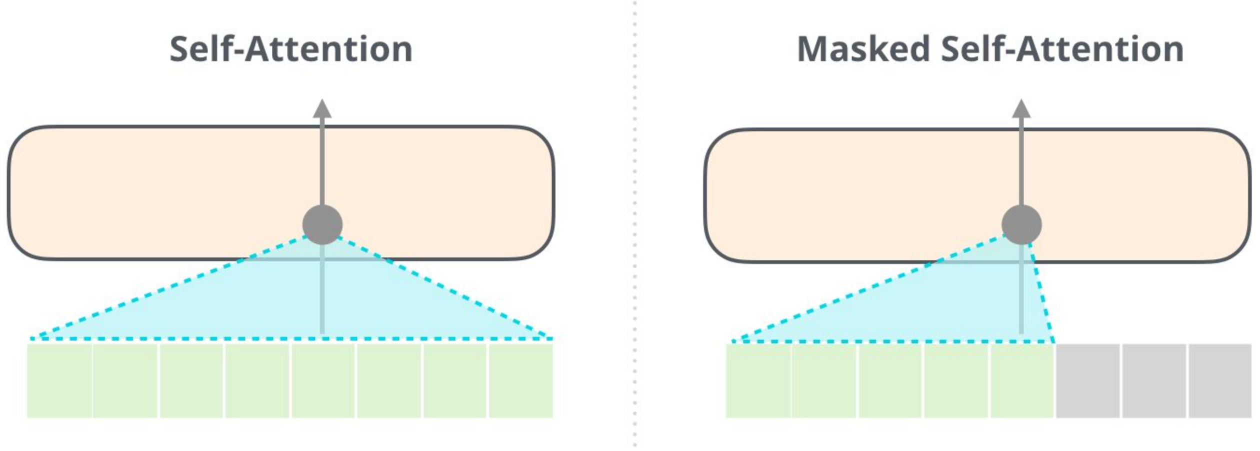


Generate Contexualized Embeddings
12
2
1
ENCODER
ENCODER
ENCODER
1
2
3
4
• • •
512
[CLS]
Help
Prince
Mayuko
BERT
The output of each encoder layer along each token's path can be used as a feature representing that token.
12
7
6
5
4
3
2
1
Help
Prince
Mayuko
But which one should we use?

GPT

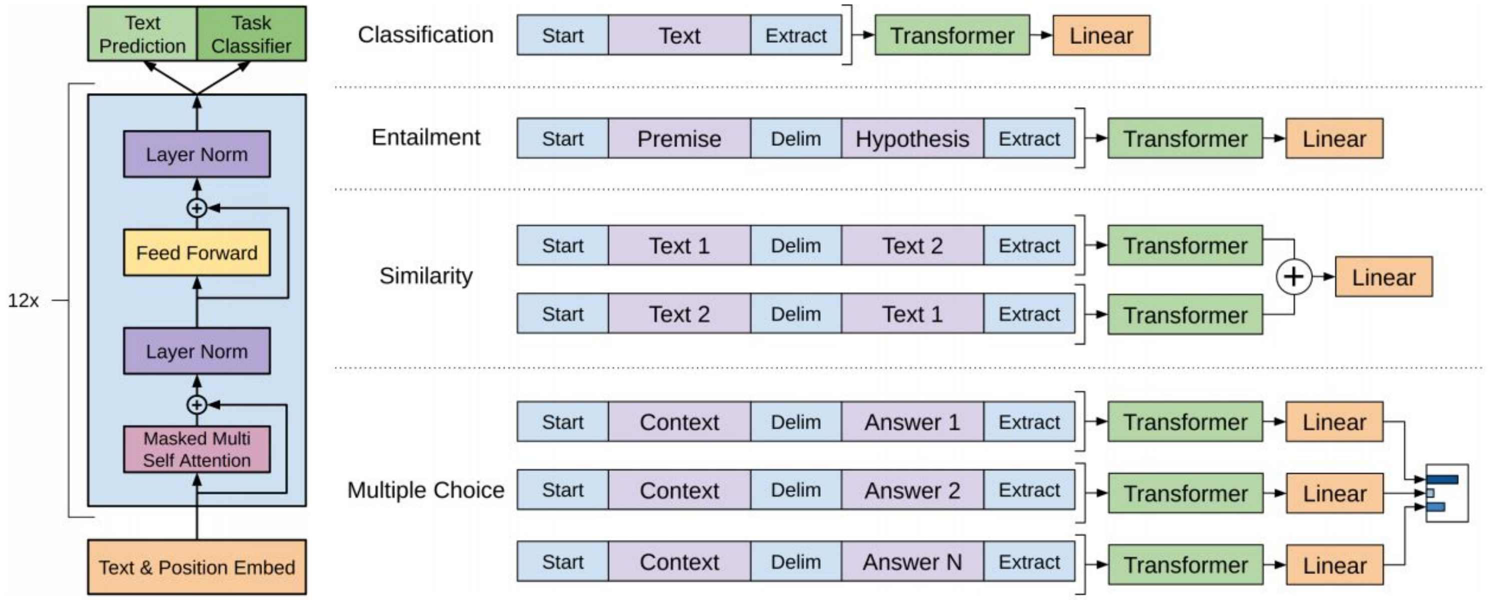
GPT (Generative Pre-Training), learns a language model using a variant of the Transformer's decoders that uses *masked self-attention*.

Masked self-attention allows the prediction to be based only on the left context of the predicted token.



GPT

GPT is trained on language generation in an unsupervised fashion, then it can be attached to additional layers to perform a supervised task.

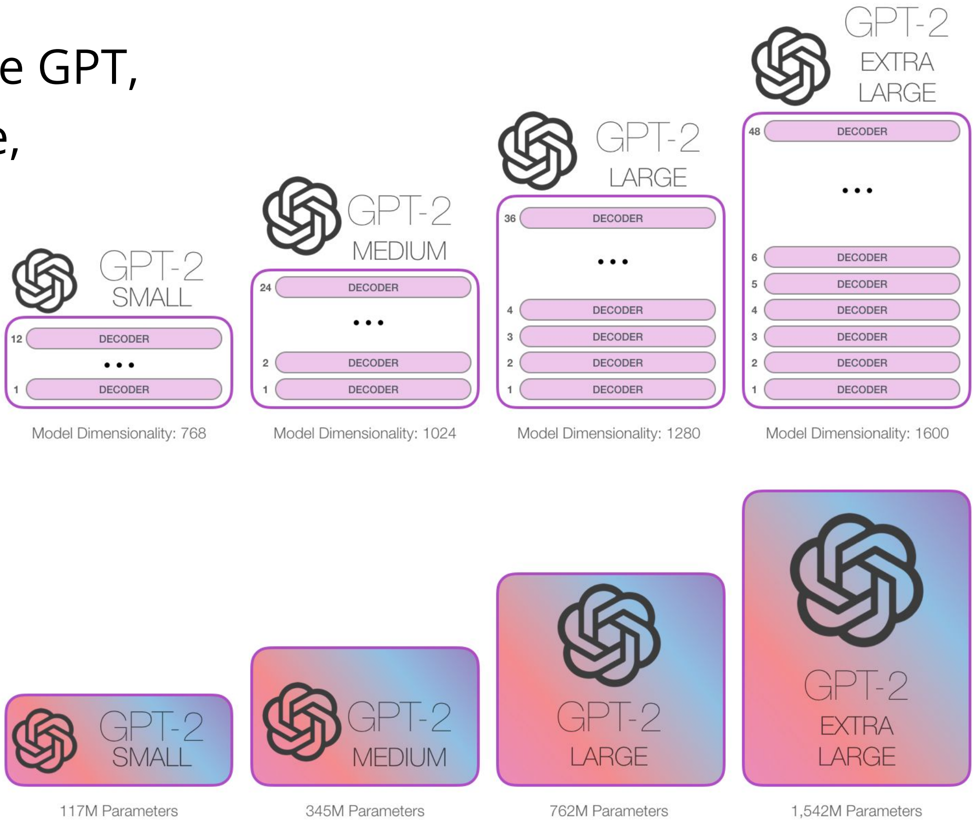


GPT-2

GPT-2 (paper) is a minor variant of the GPT, its main characteristic being the scale, with models up to 48 layers and 1.5 billion parameters.

It exhibited excellent capabilities of controlled text generation, opening a discussion about possible unethical or illegal uses.

Demo



Software

Transformers provides general-purpose architectures (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet...) for Natural Language Understanding (NLU) and Natural Language Generation (NLG) with over 32+ pretrained models in 100+ languages and deep interoperability between TensorFlow 2.0 and PyTorch.

Keras Transformers implements BERT and other elements of the Transformer to support the definition of Transformer-based language models.

Keras-GPT-2 is an example of how to use pretrained GPT-2 model weights to make predictions and generate text.

BERT repo.

XLNet repo.

Summary

Language models are a powerful tool to collect unsupervised information from a language or domain and to model it in machine-readable latent representations.

Such representations allow us to:

- explore a language, a domain, and discover its relevant entities and their syntactic and semantic relations.
- infuse general knowledge about a language or a domain into a supervised learning task, enabling the learned model to express a better generalization ability.