
NLP Canvas

Speakers:

Dhara Kotecha, Falak Shah

Factset Inc

Infocusp Innovations Pvt. Ltd.

At ICDAR 2021

About US!

- Dhara Kotecha:
 - Working as a Machine Learning Engineer 2 at Factset Systems since July 2021.
<https://www.factset.com/>
 - Prior to that, I was working at Infocusp since July 2018 in different projects with different top-notch clients. <https://infocusp.in/>
 - Most part of my industry experience has been working on challenging NLP tasks.
 - My work has involved training a curated set of deep learning based Transformer models and creating a large scale parallel processing data pipeline for the same.
 - You can reach out to me:
 - Via Email: kotechadhara6@gmail.com
 - Via LinkedIn: www.linkedin.com/in/dhara-kotecha

About US!

- Falak Shah:
 - Lead Research Scientist at Infocusp
 - Working with Infocusp since January 2015 in different projects with different top-notch clients.
 - Domain of Work: Natural Language Processing, Machine Learning, Deep Learning, Computer Vision, Convex Optimization
 - You can reach out to me on:
 - Via Email: falak@infocusp.in, falaktheoptimist@gmail.com
 - Via LinkedIn: <https://www.linkedin.com/in/falak-shah/>

Reach out

- Website: <http://infocusp.in/>
- Reach out to the organization:
 - Mr. Nisarg Vyas, Principal Founder and CEO: nisarg@infocusp.in
 - Mr. Urvik Patel, CTO: urvik@infocusp.in

LET'S START!

- Natural Language Processing and its building blocks!
- Advance Topics in NLP
- Code/Demo:
 - Basic NLP
 - BERT based models for different applications (using pretrained networks)

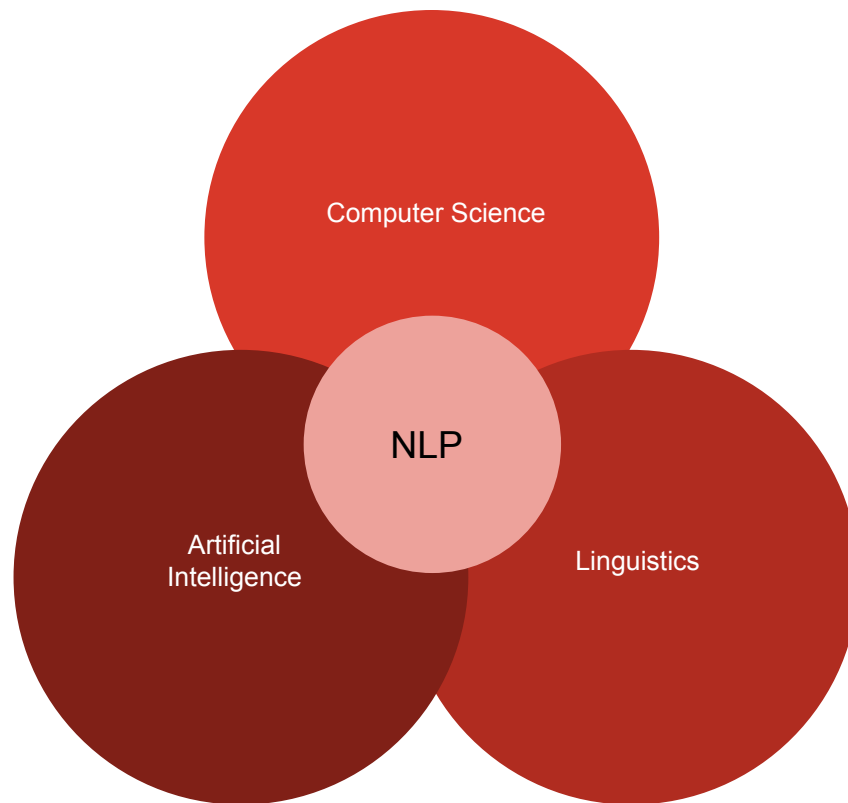
About Infocusp

- As a company, we work in the domain of Machine Learning, Natural Language Processing, Computer Vision and others under the larger umbrella of Artificial Intelligence.
- We work with:
 - Google X
 - LegalSifter: Natural Language Processing and Document Analysis
 - Bryte: Digital Signal Processing and Smart BEDS!
 - Agnetix: Smart Greenhouse Systems
 - RecruitmentSmart (Past Client): Natural Language Processing and Document Analysis
 - and many more...

Natural Language Processing

- Natural Language Processing (NLP) is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages.
- In particular how to program computers to process and analyze large amounts of natural language data.
- Make them understand the rules of language and similarity/ connections between words

Natural Language Processing



Natural Language

- Natural Language refers to the way we, humans, communicate with each other.
- Types:
 - Text
 - Speech

Natural Language

- Natural language is primarily hard because it is messy.
- There are few rules.
- And yet we can easily understand each other most of the time.

- Natural Language is highly ambiguous.
- Example:
 - Woman, without her man, is helpless.
 - Woman! Without her, man is helpless!
 - Homophones and Homographs: See and sea, Quarter to twelve or quarter in my pocket, bat and ball, bat flying

Applications of Natural Language Processing

- Text Recognition
 - Speech Recognition (Post processing)
 - Natural Language Generation
 - Translation
 - Chat bots
 - Text classification
 - Question Answering
-
- In the context of today's tutorial, we will mainly focus on the overall pipeline for any of the NLP tasks.

Let us understand the process!

Volcanic activity occurs around the Pacific Ring of Fire because many destructive plate boundaries are located here. One example is the destructive boundary between the continental South American plate and the oceanic Pacific plate which has formed the Andes Mountains. The denser oceanic plate is subducted underneath the continental plate and melts as it falls into the hot mantle. Magma then rises up through the continental plate and is erupted through volcanoes at the surface. The destructive boundaries all around the Pacific Ring of Fire are the reason for high volcanic activity.

Understanding the process!

- Cleaning the data (Only keeping the required text).
- Case-sensitivity.
- Breaking documents into paragraphs.
- Breaking paragraphs into sentences.
- Breaking sentences into words.
- Grammar
- Focusing on words - their meanings, synonyms and their usages.
- Lemmatization and Stemming
- Part of Speech (Verb, Noun, Adjective, Adverb, Number etc.)
- Named Entity Relation
- Keyword recognition

Sentence Boundary Detection

- Sentence Boundary Detection is a problem in itself.
- **There. Are various way! To detect Sentence Boundary.**
- Why do we do Sentence Boundary Detection?
 - So that we have to look at a smaller sentence and decide the next steps!
 - Long sentences are difficult for a normal human being to read. And so for a machine to understand!
- Erroneous Sentence Boundary might lead to loss of important data semantics.

From Sentences to Words

- Breaking the stone into minute pieces - cleaning, experimenting and polishing might lead to diamonds.
- English is much easier language because we can split sentences on white-spaces to get the words.
- But it is a challenge for many different languages.

Stemming and Lemmatization

- **“Word normalization.”**
- The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.
- **Stemming:** Stemming is the process of converting the words of a sentence to its non-changing portions.
 - Example : The stem of amusing, amusement, and amused would be “amus”.
 - Different types of Stemmers, used commonly: Lancaster Stemmer, Porter Stemmer. Snowball Stemmer etc.
- **Lemmatization:** Lemmatization is the process of converting the words of a sentence to its dictionary form.
 - Example: Lemma for amusement, amusing, and amused will be “amuse”.
 - Lemma for “women” will be “woman”

Linguistics

- Language is unstructured form of data.
- But that are certain rules that govern the language - Linguistics.
- Linguistics is a scientific study of language, including its syntax, grammar, semantics and phonetics.
- How can we leverage this knowledge in any NLP task ?
- There are various libraries that provide such tools and technologies.

Stanford Core NLP

- Stanford CoreNLP provides a set of human language technology tools.
- It can give the base forms of words, their parts of speech, whether they are names of companies, people, etc., normalize dates, times, and numeric quantities, mark up the structure of sentences in terms of phrases and syntactic dependencies, indicate which noun phrases refer to the same entities, indicate sentiment, extract particular or open-class relations between entity mentions, get the quotes people said, etc.
- More details can be found at: <https://stanfordnlp.github.io/CoreNLP/>

Stanford Core NLP

- Tokenization
- Sentence Splitting
- Lemmatization
- Parts of Speech
- Sentiment
- Named Entity Recognition
- Dependency Parsing
- Coreference Resolution

Stanford Core NLP

- **Part-of-Speech (POS):**

- A category or a class to which a word is assigned in accordance with its syntactic functions.
- In English the main parts of speech are noun, pronoun, adjective, determiner, verb, adverb, preposition, conjunction, and interjection.

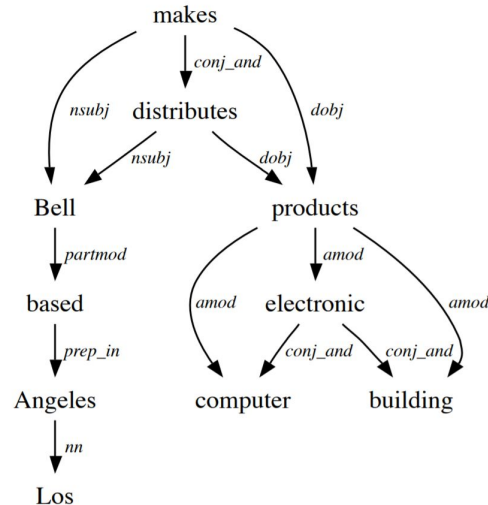
- **Name-Entity-Relation (NER):**

- For English, by default, this annotator recognizes named (PERSON, LOCATION, ORGANIZATION, MISC), numerical (MONEY, NUMBER, ORDINAL, PERCENT), and temporal (DATE, TIME, DURATION, SET) entities.
- 12 classes by default.

Stanford Core NLP visualisation at: <http://nlp.stanford.edu:8080/corenlp/>

Dependency Parsing

- Relations among the words are illustrated above the sentence with directed labeled arcs, from heads to dependents. This is called Dependency Structure.
- When this Dependency Structure is parsed it form a tree like structure called Dependency Tree.
- The relationship between nodes directly encode important information that is often buried in complex sentences. Eg: Bell, based in Los Angeles, makes and distributes electronic, computer and building products



Conference Resolution

- Coreference resolution is the task of finding all expressions that refer to the same entity in a text.
- Example - “I voted for Modi because he was most aligned with my values”, she said.
- In this sentence, the conference resolution will identify that
 - The words “I”, “my” and “she” refer to the same entity and
 - The words “Modi” and “he” refer to the same entity
- This kind of information is very useful in natural language understanding tasks such as summarisation, question answering, etc.

NLTK

- Natural Language Toolkit (NLTK) is a platform for building Python programs to work with human language data.
- It provides easy-to-use interface to over 50 corpora and lexical resources.
- It provides text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

spaCy

- spaCy is an open-source software library for advanced Natural Language Processing, written in the Python programming language.
- spaCy handles large-scale extraction tasks

For more information go to: <https://spacy.io/>

- Features
 - Tokenization
 - Named entity recognition
 - Support for 28+ languages
 - 13 statistical models for 8 languages
 - Pre-trained word vectors
 - Easy deep learning integration
 - Part-of-speech tagging
 - Labelled dependency parsing
 - State-of-the-art speed
 - Robust, rigorously evaluated accuracy

What next?

- So, we saw different libraries and methods to normalize words, find linguistic and semantic relations between them but what do we do with it?
- Or the real question is how to convert the human language to machine understandable language.
- That's where Vectors comes into play!

Machine friendly representation of documents

- Machines are not able to interpret language in the way humans do.
- Machines understand numbers and hence arises a need to convert words, sentences and documents into machine interpretable numerical form.

Why and What is Vectorization?

- The ability to process natural human language is one of the things that makes machine learning algorithms so powerful.
- However, when doing natural language processing, words must be converted into vectors that machine learning algorithms can make use of.
- If a word can be translated into a single number, documents can be translated to a vector.
- Due to variety of words, these vectors tend to be high-dimensional.
- This whole process of representing text data into numbers is called “**Word Vectorization**” or “**Word Embedding**”.

Bag-of-Words (BoW)

- What is the best representation of documents in numerical form?
- Concept of Vocabulary

Count Vectorizer

- Simplest model or algorithm which counts the frequency of words in a document to create a document vector.
- Count the number of times each term in the vocabulary appears in the document.
- The vector formed by this model is nothing but a frequency vector of vocabulary words.

Sample Document: “The quick brown fox. Jumps over the lazy dog!”

	Jumps	The	brown	dog	fox	lazy	over	quick	the
Doc1	0	1	1	0	1	0	0	1	0

Hashing Vectorizer

- Instead of building a vocabulary and using that as feature, we define a hashing function which will help us generate a vector of a pre-defined length by hashing the original features. (Downside: cannot retrieve original words)

- Example:

```
function hashing_vectorizer(features : array of string, N : integer):  
    x := new vector[N]  
    for f in features:  
        h := hash(f)  
        x[h mod N] += 1  
    return x
```

- Suppose, our feature vector is ["cat","dog","cat"] and hash function is: **hash("cat") = 2** and **hash("dog")=1**.
- Suppose, output feature vector dimension (N) to be 4. Then output x will be [0,2,1,0].

Hashing Vectorizer

- When we create vectors by using vocabulary, it utilizes a lot of memory.
- Let's take the example of three documents: (Vocab size = 9)

- *John likes to watch movies.*
- *Mary likes movies too.*
- *John also likes football.*

John	likes	to	watch	movies	Mary	too	also	football
1	1	1	1	1	0	0	0	0
0	1	0	0	1	1	1	0	0
1	1	0	0	0	0	0	1	1

Term Frequency - Inverse Document Frequency (TF-IDF)

$$TF_{i,D} = (\text{Number of times the term } i \text{ appears in the document } D) / (\text{Total number of words in the document } D)$$

Scikit-Learn

- $IDF(t) = \log \frac{1+n}{1+df(t)} + 1$

$$TF-IDF = TF * IDF$$

Standard notation

- $IDF(t) = \log \frac{n}{df(t)}$

df(t) = Number of documents containing term t and
n = Total no of documents

TF-IDF Matrix representation of corpus of documents

	Document 1	Document 2	Document 3	Document 4	Document 5	Document 6	Document 7	Document 8
Term(s) 1	10	0	1	0	0	0	0	2
Term(s) 2	0	2	0	0	0	18	0	2
Term(s) 3	0	0	0	0	0	0	0	2
Term(s) 4	6	0	0	4	6	0	0	0
Term(s) 5	0	0	0	0	0	0	0	2
Term(s) 6	0	0	1	0	0	1	0	0
Term(s) 7	0	1	8	0	0	0	0	0
Term(s) 8	0	0	0	0	0	3	0	0

← Word Vector
(Passage Vector)

Document Vector

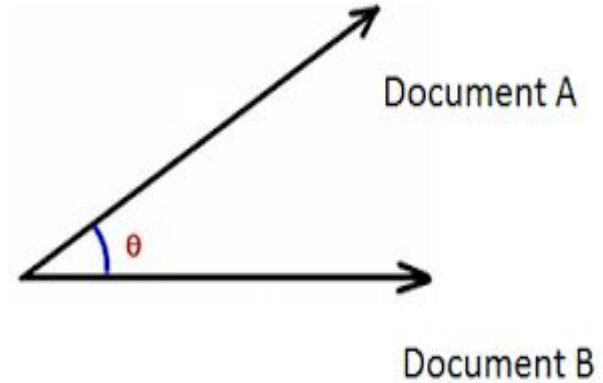
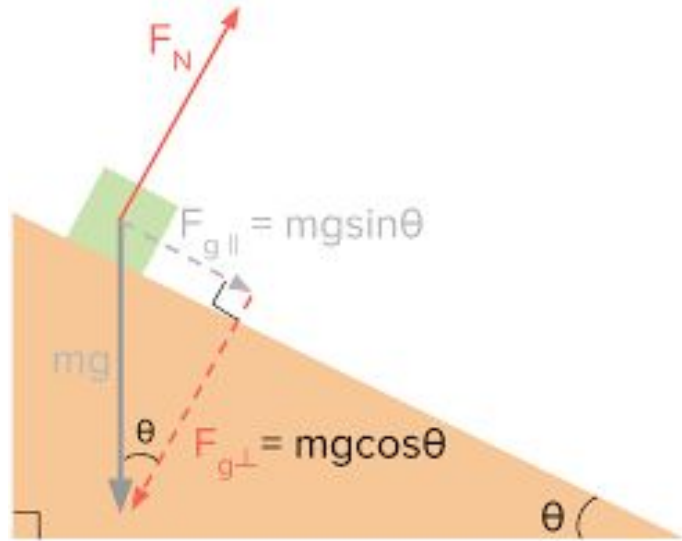
Recap

- What is NLP?
- Standard process and pipeline.
- Open-source/Freely available tools and libraries.
- Need of vectors.
- Word vectorization
 - Count Vectorizer
 - Hashing Vectorizer
 - TF-IDF Vectorizer

Why vectors?

- One of the main focus of NLP is to compare different documents and look at the similarity between documents or similarity between the word-vectors.
- The vector representation of the documents makes the calculation of similarity very easy.
- Let's discuss one of the commonly used similarity measures

Cosine Similarity



$$\text{cosine similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

n-gram Model

- Bag of Words Model considered each word independently. It is an unordered way of looking at the text without using the context.
- Won't be useful for predicting the next word in a sentence (Example: Gmail autocomplete)
- n-gram predicts the sequence of words.
- “n” in n-gram denotes the length of sequence. Based on varying “n” we have:
 - n=1 - monograms / unigrams
 - n=2 - bigrams / digrams
 - n=3 - trigrams and so on...

n-gram Model

- n-gram computes the probability of i th item given a sequence of previous $(i-1)$ items.

$$P(x_i \mid x_{i-(n-1)}, \dots, x_{i-1})$$

- For Example, we want the machine to learn that the words “Machine” & “Learning” are commonly used together. “Machine Learning” is a bigram.
- The probabilities are estimated using the **relative frequencies** of observed outcomes. This process is called **Maximum Likelihood Estimation (MLE) of n-gram models**.

Example

- Compute probability of trigram: “I like bananas”

$$P(\text{bananas} | \text{i like}) = \frac{C(\text{i like bananas})}{C(\text{i like})}$$

Example: Estimate sequence using bigrams

Assumption: Probability
of next word depends
only on previous word.

"the man drank some beer"				
w_1	w_2	$C(w_1w_2)$	$C(w_1)$	$P(w_2 w_1)$
the	man	10,605	5,973,437	0.00178
man	drank	2	58,168	0.00003
drank	some	40	1,306	0.03063
some	beer	18	165,421	0.00011

$$\begin{aligned}P(w_1^n) &\approx \prod_{k=1}^n P(w_k|w_{k-1}) \\&\approx P(\text{man}|\text{the}) \times P(\text{drank}|\text{man}) \times \\&\quad P(\text{some}|\text{drank}) \times P(\text{beer}|\text{some}) \\&\approx 0.00178 \times 0.00003 \times 0.03063 \times 0.00011 \\&\approx 1.79 \times 10^{-13}\end{aligned}$$

Problems...

- The issue with BoW and n-grams models is the lack of context.
 - BoW could not capture any essences of the semantics while n-grams and its variant could only capture the backward semantics.
 - Space and Time Complexity
- What can be the solution?

Word2Vec

- This algorithm is used for learning vector representations of words, called “Word Embeddings”.
- These models are shallow, **two-layer neural networks** that are trained to reconstruct linguistic contexts of words.
- Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.
- Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space.

One can use the **existing pre-trained word vectors** like GloVe, GoogleNews-vectors-negative300 etc. Standard words vectors are open-source and available freely.

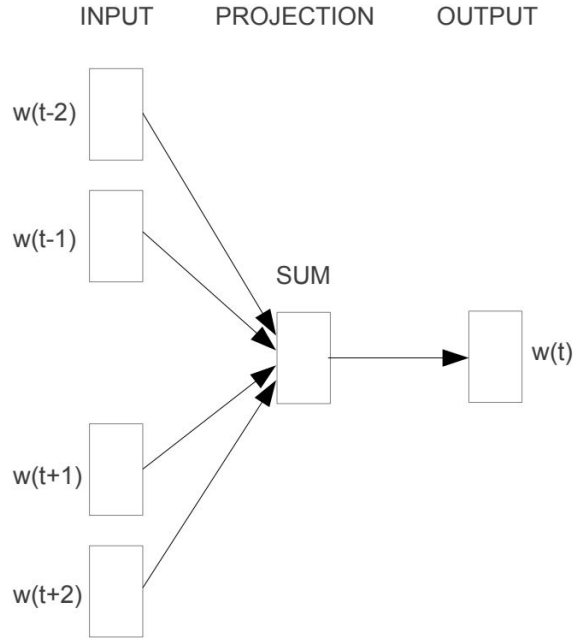
Word2Vec

- The key benefit of the approach is that high-quality word embeddings can be learned efficiently (low space and time complexity), allowing larger embeddings to be learned (more dimensions) from much larger corpora of text (billions of words).

Word2Vec

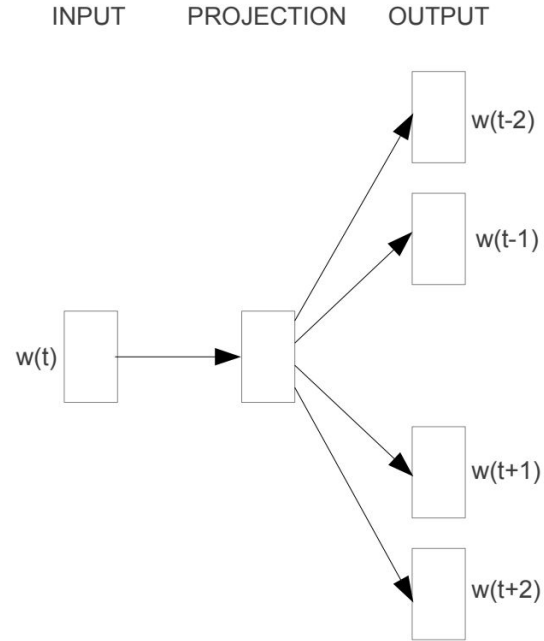
- Two popular models:
 - Continuous Skip-Grams
 - Continuous Bag of Word (CBOW)
- Both try to encode the idea, that words occurring in similar contexts (same surrounding words) have semantic similarity

Continuous Skip-Grams and CBOW in a nutshell



CBOW

[Image Credits](#)



Skip-gram

Generation of Training data for Word2Vec (Skip gram)

Source Text

Training Samples

<div>The quick brown fox jumps over the lazy dog.</div> <div>The quick brown</div>	→	(the, quick) (the, brown)
<div>The quick brown fox jumps over the lazy dog.</div> <div>The quick brown fox</div>	→	(quick, the) (quick, brown) (quick, fox)
<div>The quick brown fox jumps over the lazy dog.</div> <div>The quick brown fox jumps</div>	→	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
<div>The quick brown fox jumps over the lazy dog.</div> <div>The quick brown fox jumps over</div>	→	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Training Word2Vec

- We'll train the neural network by feeding word pairs found in our training documents.
- The output probabilities are going to relate to how likely it is find each vocabulary word nearby our input word.

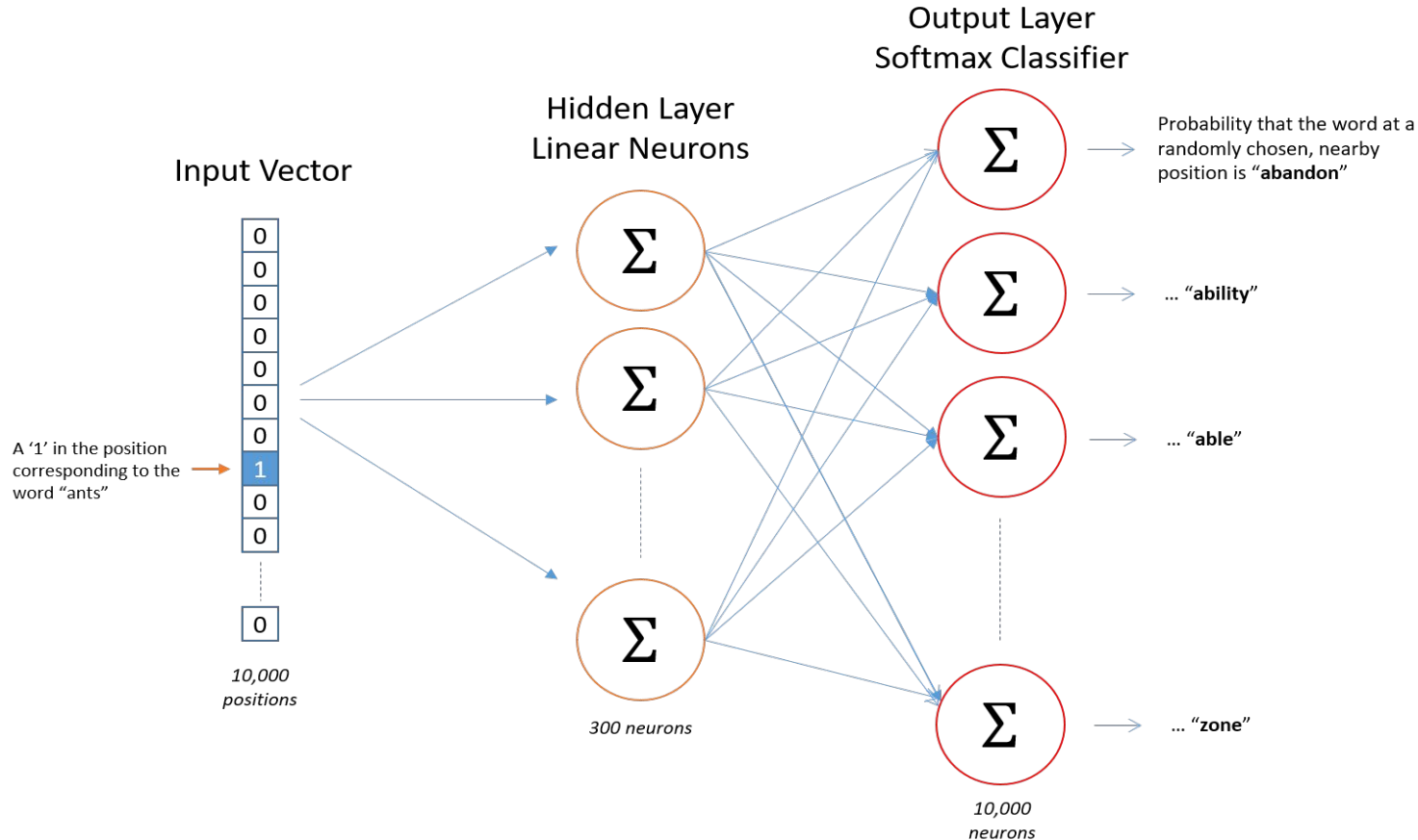
Skip gram

- Given a central word, it will predict its neighbouring words with different probabilities.

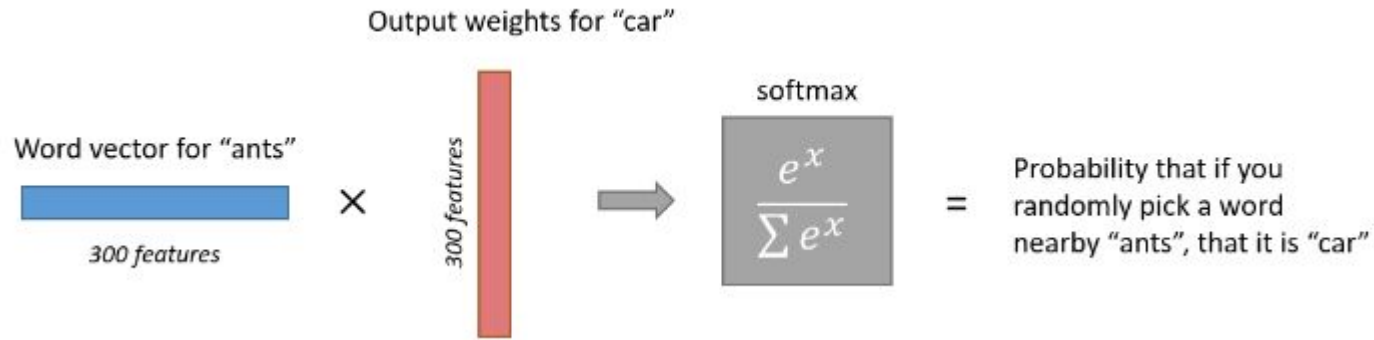
CBOW

- Predict target word from the surrounding contextual words.

Architecture of Word2vec model



Interpretation of Skip-Grams Output

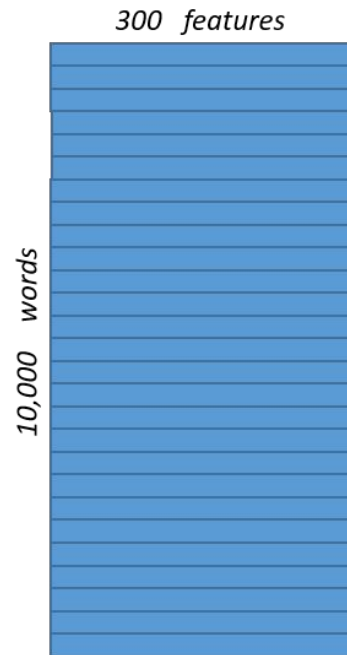
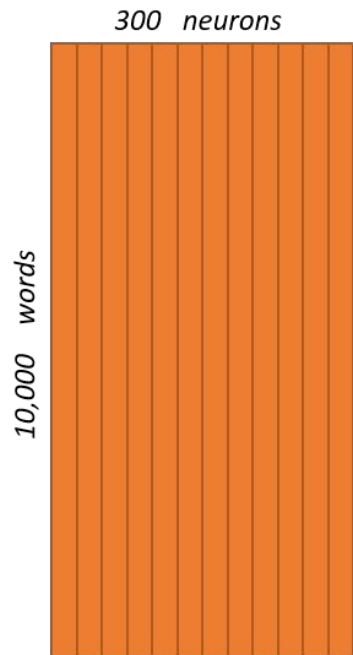


Word2vec

Hidden Layer
Weight Matrix



*Word Vector
Lookup Table!*



GloVe

- GloVe : Global Vectors for Word Representation
- GloVe is an **unsupervised learning** algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus
- Trained on extremely large data set, GloVe comes in a variant from 50D to 300D.
- The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities encode some semantics

GloVe

- The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.
- Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space.
- For this reason, the resulting word vectors perform very well on word analogy tasks, such as in the word2vec package.

GloVe

Probability and Ratio	$k = \text{solid}$	$k = \text{gas}$	$k = \text{water}$	$k = \text{fashion}$
$P(k \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \text{ice})/P(k \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

- As one might expect, ice co-occurs more frequently with solid than it does with gas, whereas steam co-occurs more frequently with gas than it does with solid.
- Both words co-occur with their shared property water frequently, and both co-occur with the unrelated word fashion infrequently.

GloVe

Probability and Ratio	$k = \text{solid}$	$k = \text{gas}$	$k = \text{water}$	$k = \text{fashion}$
$P(k \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \text{ice})/P(k \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

- Only in the ratio of probabilities does noise from non-discriminative words like water and fashion cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.
- In this way, the ratio of probabilities encodes some crude form of meaning associated with the abstract concept of thermodynamic phase.

Visualizing the vectors projected to 3d space:

<http://projector.tensorflow.org/>

NLP Tasks

- These different algorithms and models for Word Embeddings coupled with Machine Learning algorithms like SVM, Linear Regression, Logistic Regression etc. can be used to solve numerous NLP tasks.

Advance Topics in Natural Language Processing

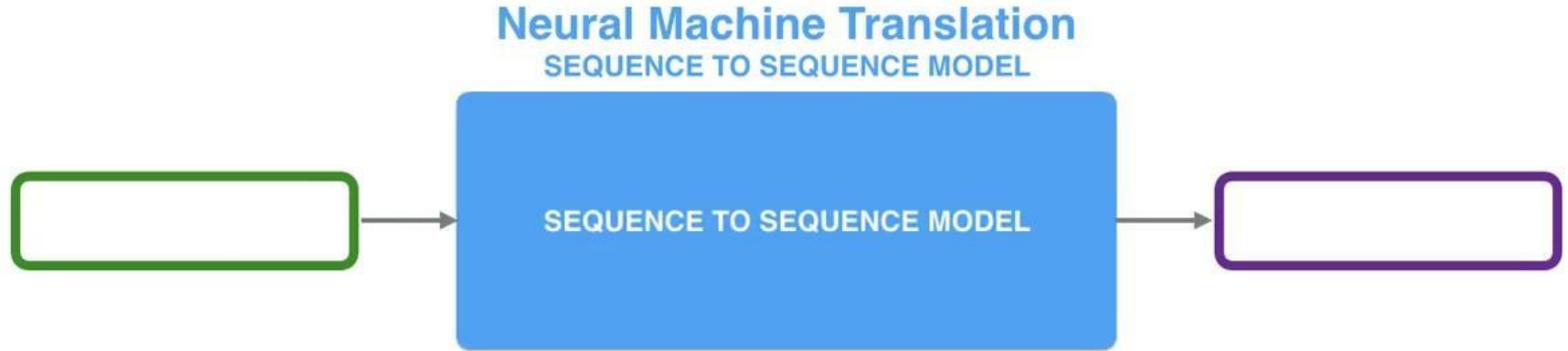
With the recent advances in Machine Learning and Deep Learning techniques,
let us present a gist of some of the advance topics of
Natural Language Processing.

Processing sequence of items



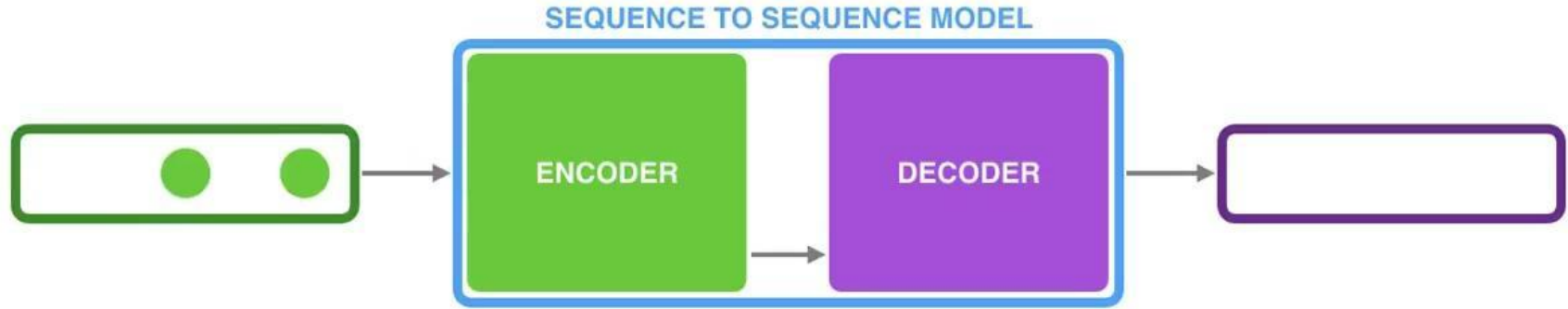
Credits:
<http://jalammar.github.io/illustrated-bert/>

Sequence to Sequence example



Credits:
<http://jalammar.github.io/illustrated-bert/>

Under the black box



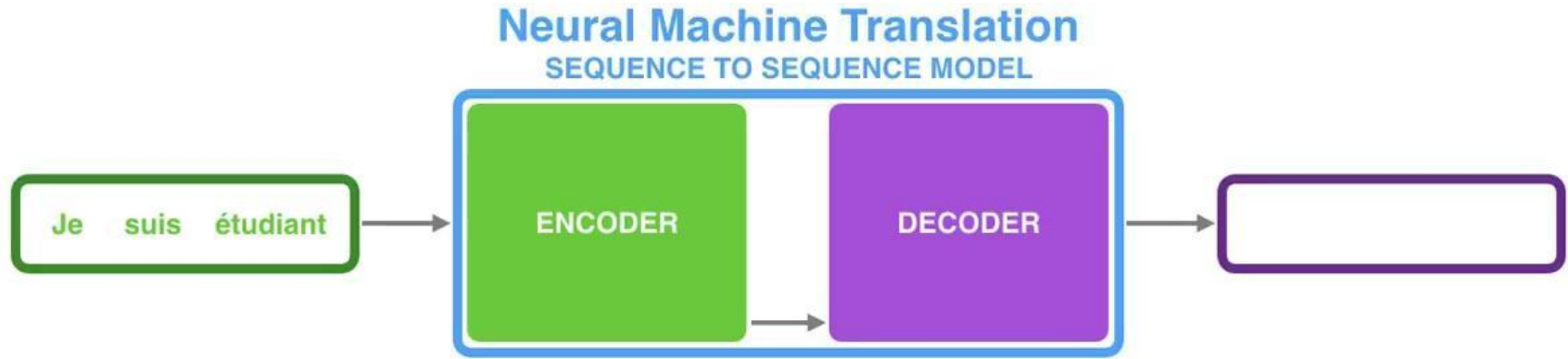
Credits:
<http://jalammar.github.io/illustrated-bert/>

Example

What does this mean ?

“je suis étudiant”

Example



Credits:
<http://jalammar.github.io/illustrated-bert/>

The Context Vector

CONTEXT

0.11
0.03
0.81
-0.62

0.11
0.03
0.81
-0.62

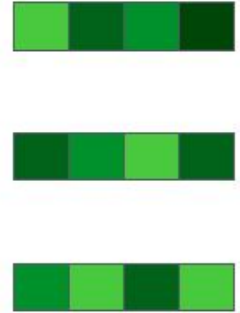
Credits:
<http://jalammar.github.io/illustrated-bert/>

Convert input words to vectors

Input



0.901	-0.651	-0.194	-0.822
-0.351	0.123	0.435	-0.200
0.081	0.458	-0.400	0.480



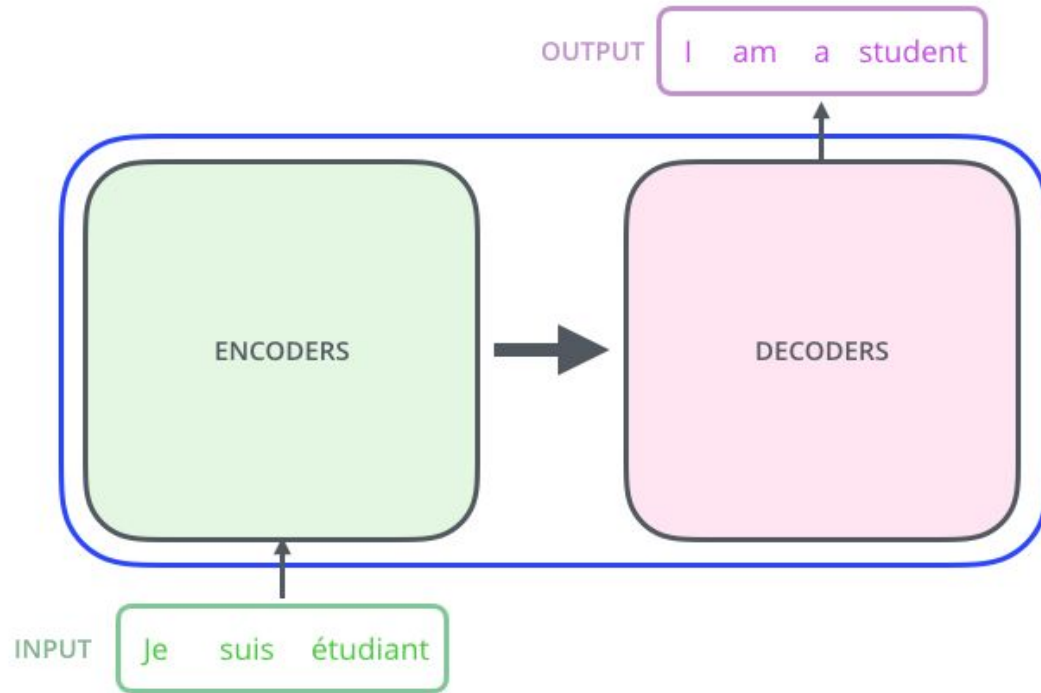
Credits:
<http://jalammar.github.io/illustrated-bert/>

The Transformers!



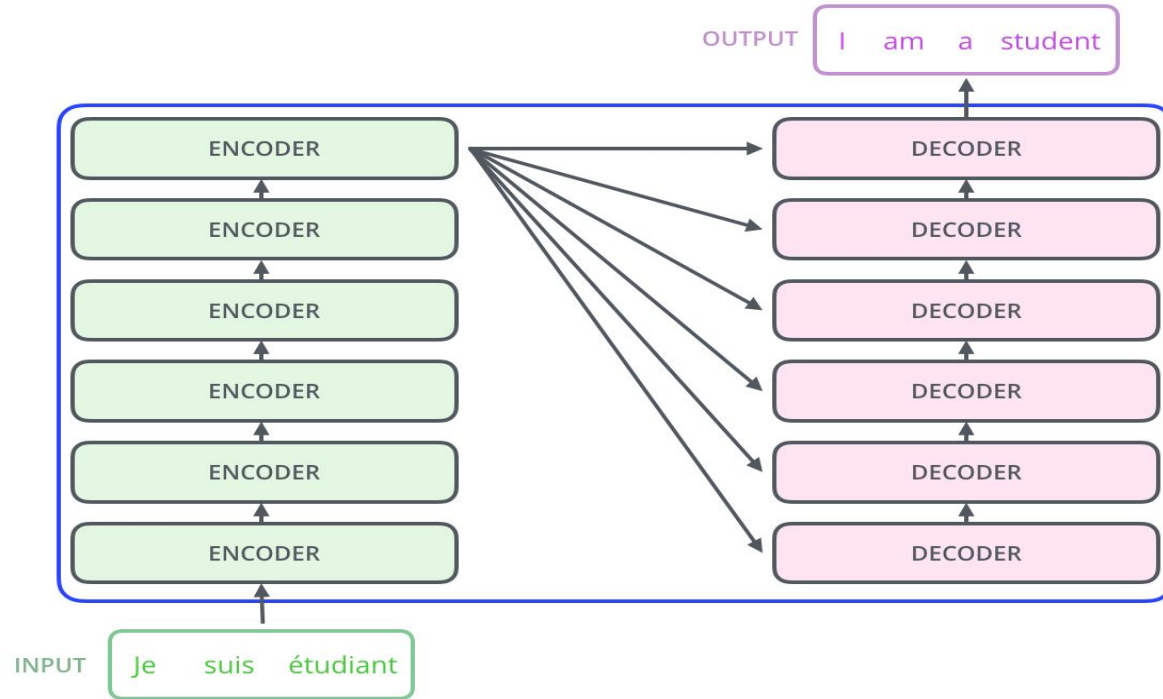
Credits:
<http://jalammar.github.io/illustrated-bert/>

Inside the Transformer!



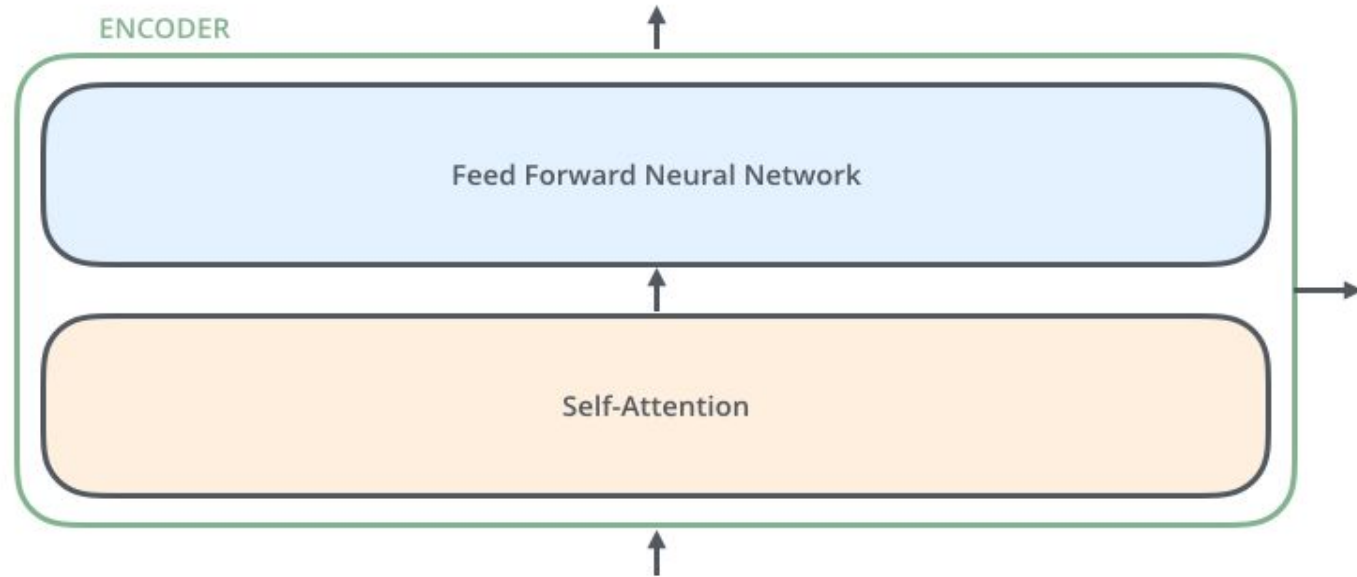
Credits:
<http://jalammar.github.io/illustrated-bert/>

Inside the Transformer



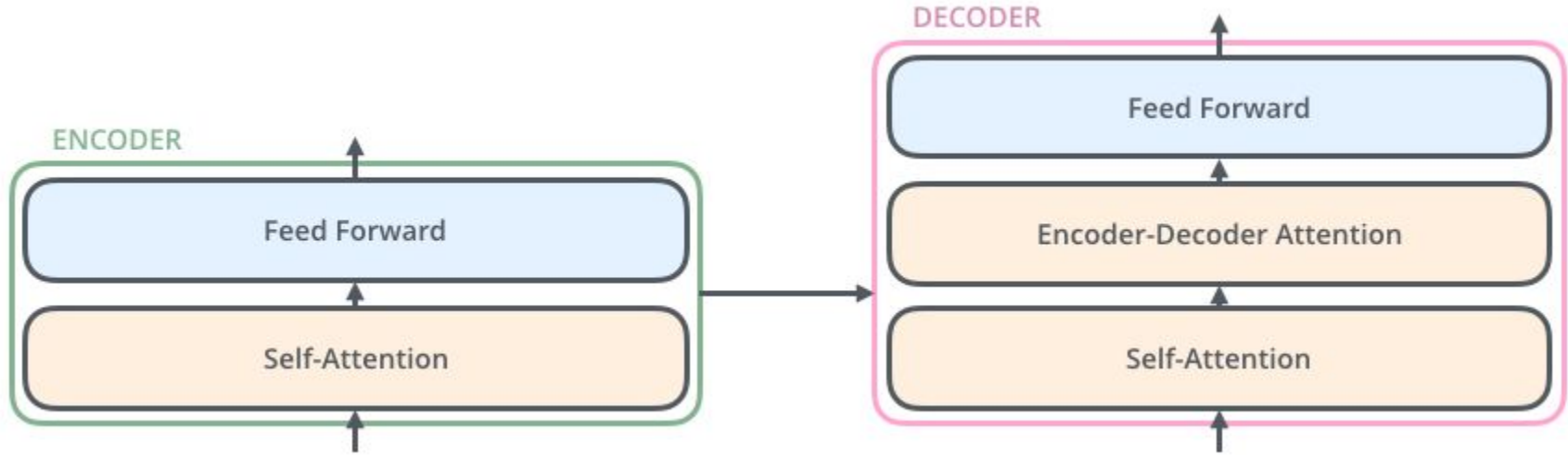
Credits:
<http://jalammar.github.io/illustrated-bert/>

Inside the Encoder!



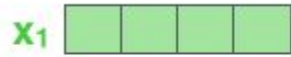
Credits:
<http://jalammar.github.io/illustrated-bert/>

The Decoder!

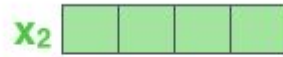


Credits:
<http://jalammar.github.io/illustrated-bert/>

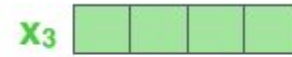
The Vectors



Je



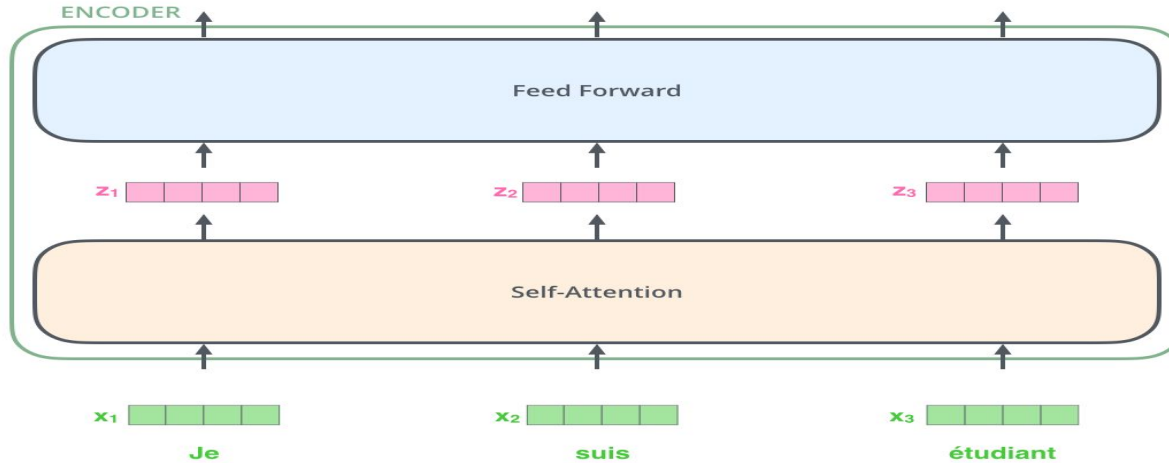
suis



étudiant

Credits:
<http://jalammar.github.io/illustrated-bert/>

Encoder Flow

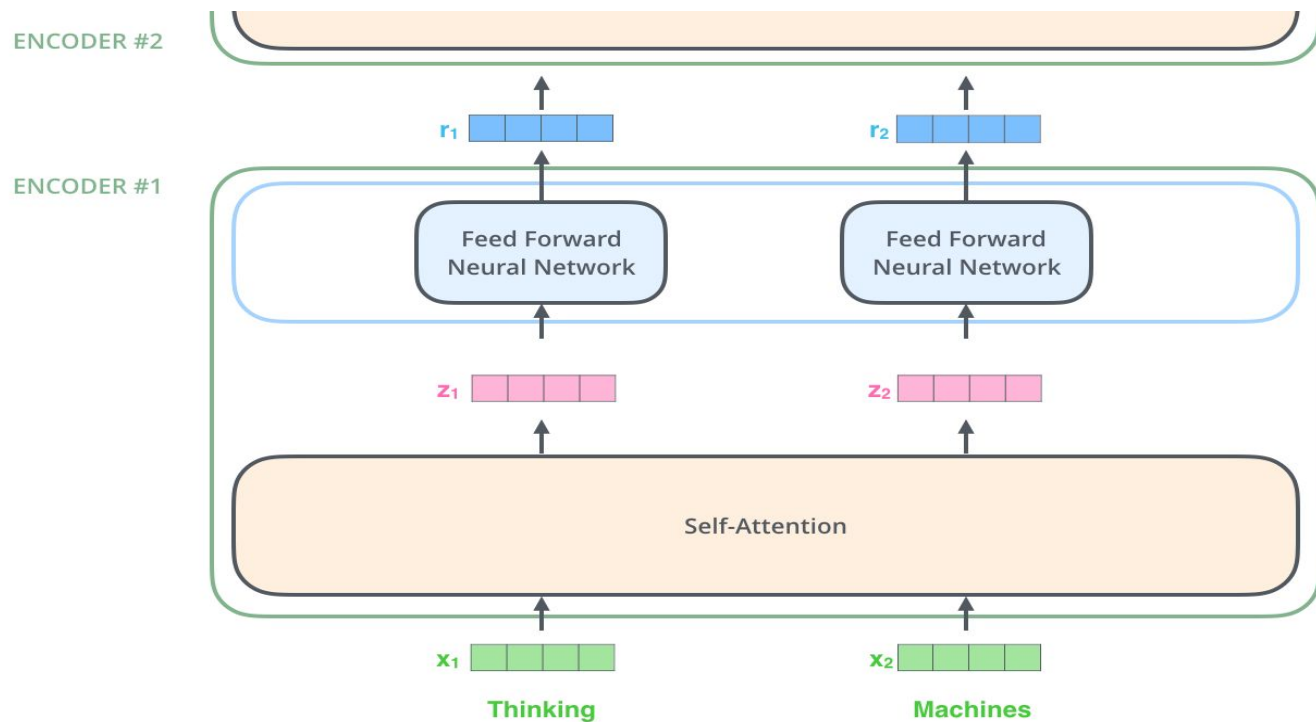


- Only the first encoder block will get the input vectors (word embeddings).
- The word in each position flows through its own path in the encoder.
- There are dependencies between each path in the self attention layer.
- The feed forward neural network does not have dependencies.

Credits:

<http://jalammar.github.io/illustrated-bert/>

Encoder Flow



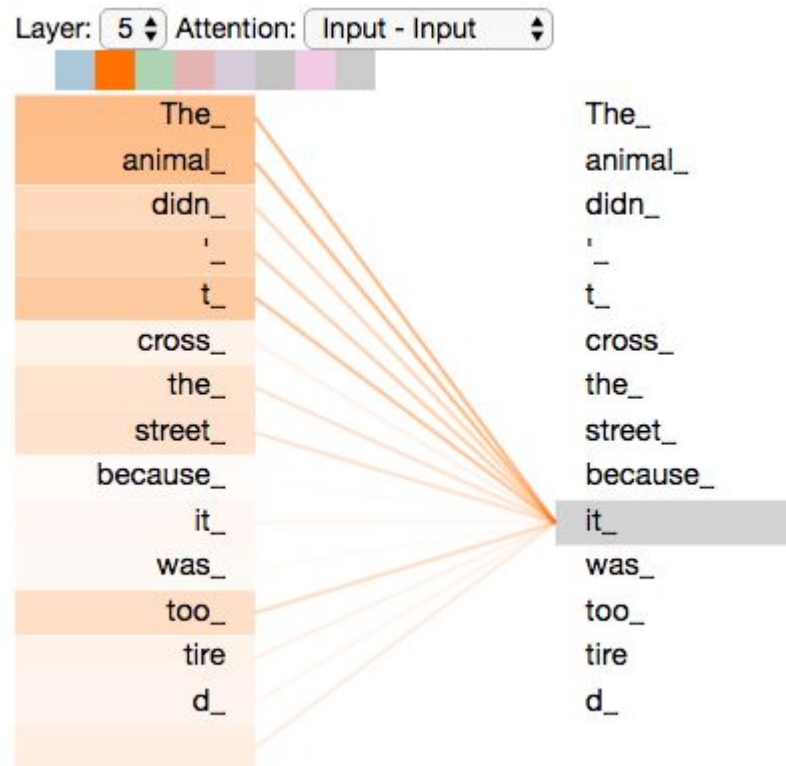
What exactly is “Self-Attention”?

“The animal didn't cross the street because it was too tired”

- What does “it” refer to? Street or animal?
- When model is processing “it”, the self attention allows it to associate “it” with animal.
- While processing a word, the self attention layer allows it to look at other positions in the input sequence for clues that can help encode that word better.

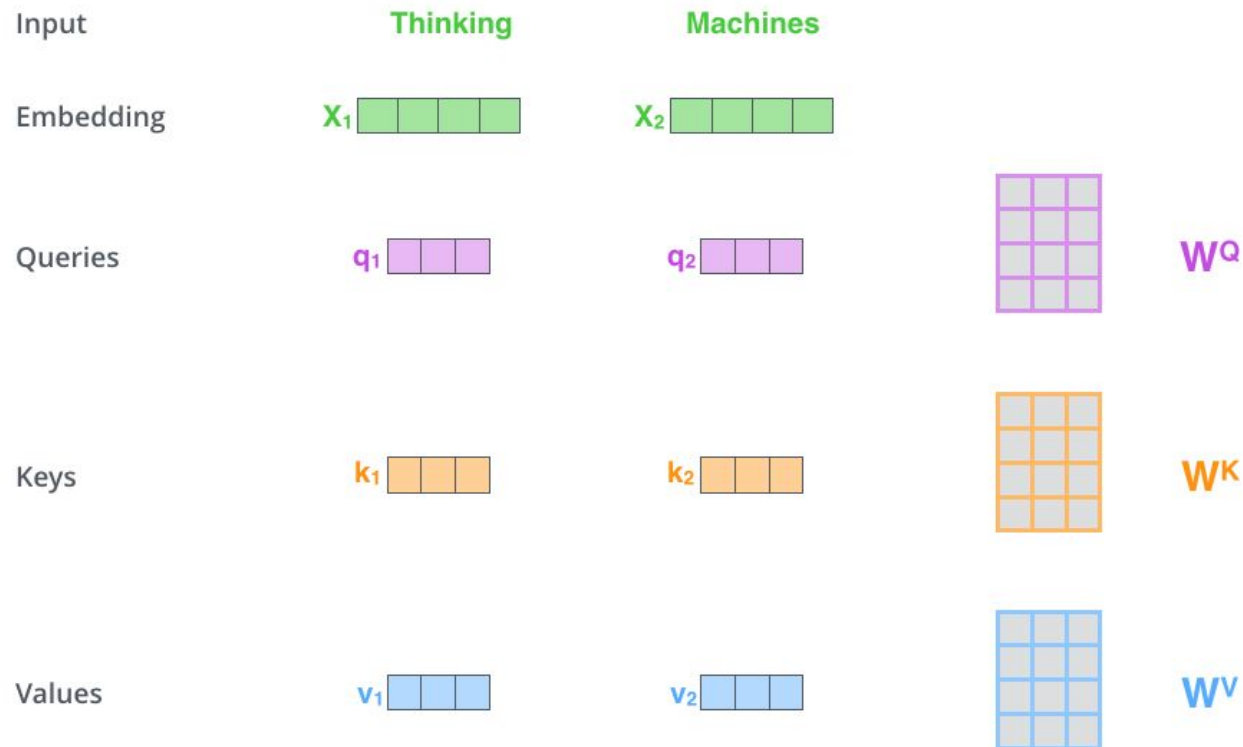
How to pay “attention”?

- While processing a word, the other words are scored to decide how much attention has to be paid to each of them.
- There is a method for scoring - which is done in the self attention layer.
- Each word is thus scored and a combination of vectors of all these words are then used as an extra information to process a word.

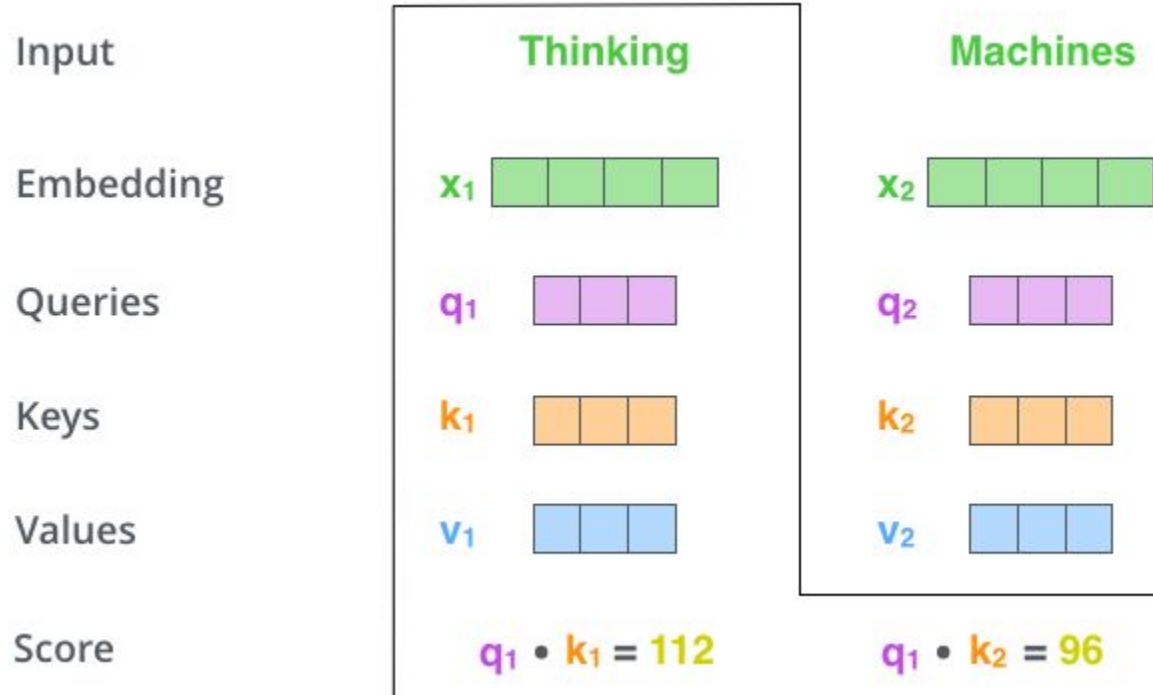


Credits:
<http://jalammar.github.io/illustrated-bert/>

How to calculate attention-score?

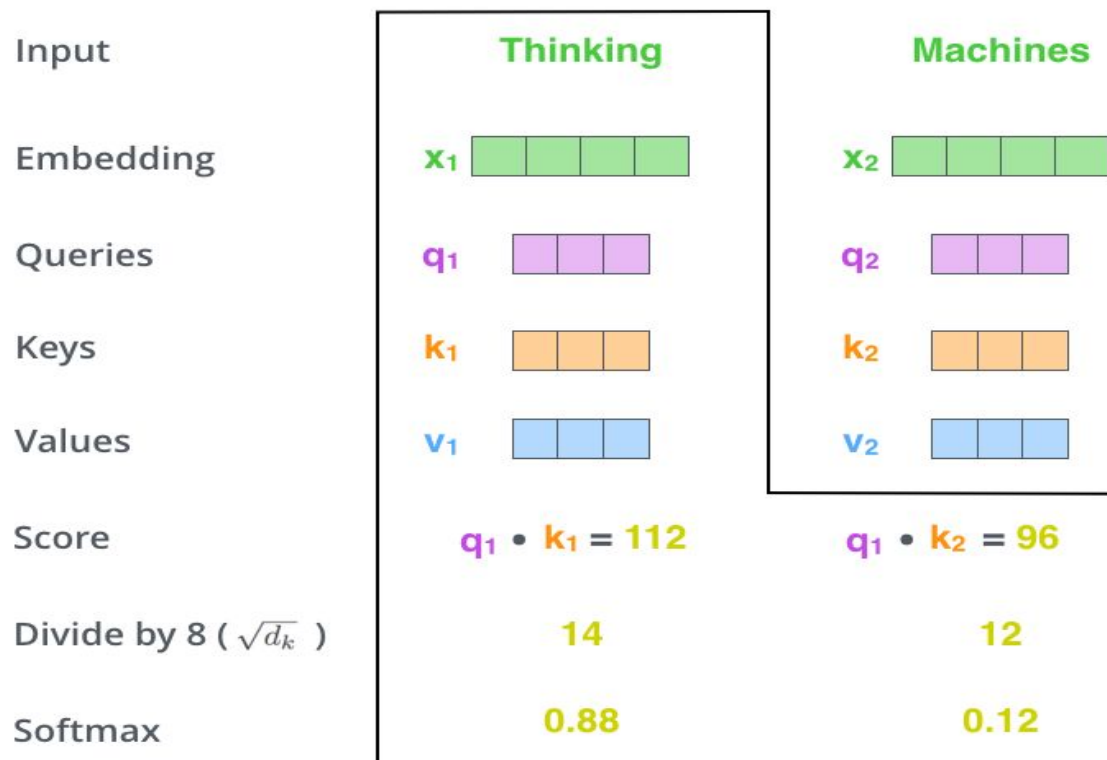


How to calculate attention-score?



Credits:
<http://jalammar.github.io/illustrated-bert/>

How to calculate attention-score?



Credits:
<http://jalammar.github.io/illustrated-bert/>

How to calculate attention-score?

Input

Embedding

Queries

Keys

Values

Score

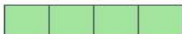
Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax
X
Value

Sum

Thinking

x_1 

q_1 

k_1 

v_1 

$q_1 \cdot k_1 = 112$

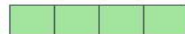
14

0.88

v_1 

z_1 

Machines

x_2 

q_2 

k_2 

v_2 

$q_1 \cdot k_2 = 96$

12

0.12

v_2 

z_2 

Credits:

<http://jalammar.github.io/illustrated-bert/>

Matrix calculation of Self-Attention

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W}^Q \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{Q} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W}^K \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{K} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W}^V \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{V} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

Credits:
<http://jalammar.github.io/illustrated-bert/>

Matrix calculation of Self-Attention

$$\text{softmax} \left(\frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \end{matrix} \right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

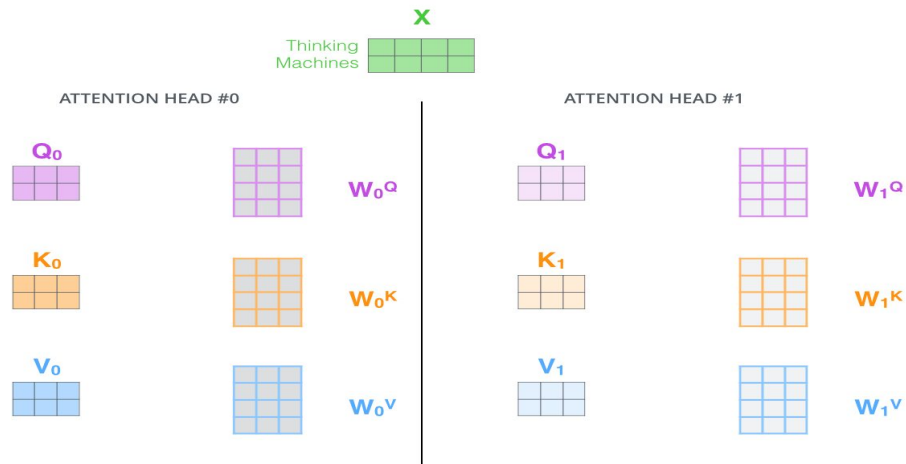
=

Z

$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array}$

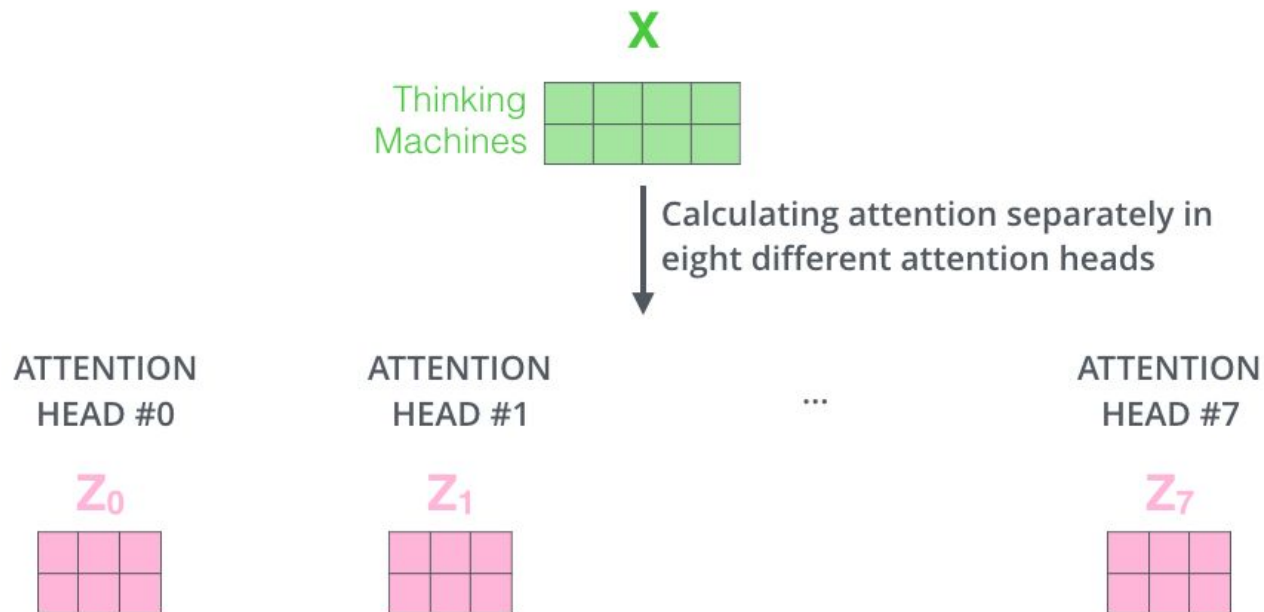
Credits:
<http://jalammar.github.io/illustrated-bert/>

Multiple attention heads



- Multiple attention heads expands the models ability to focus on different positions.
- It gives the model multiple representation subspaces because corresponding to each space, we will have different query/key/value vectors.
- In the paper, they have used 8 attention heads

Multiple attention heads



Credits:
<http://jalammar.github.io/illustrated-bert/>

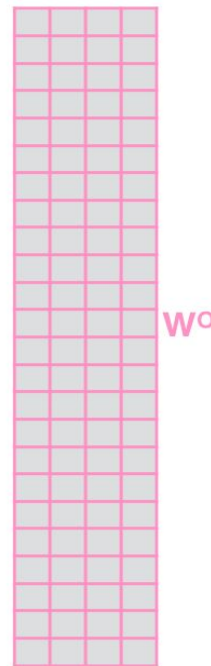
Multiple attention heads

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN



Credits:

<http://jalammar.github.io/illustrated-bert/>

1) This is our
input sentence*

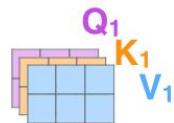
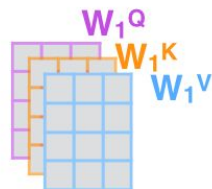
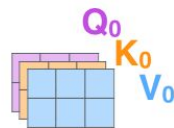
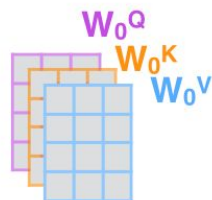
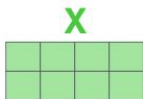
2) We embed
each word*

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

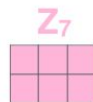
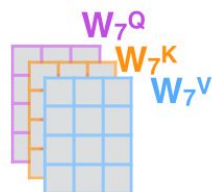
Thinking
Machines



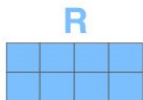
...

...

...

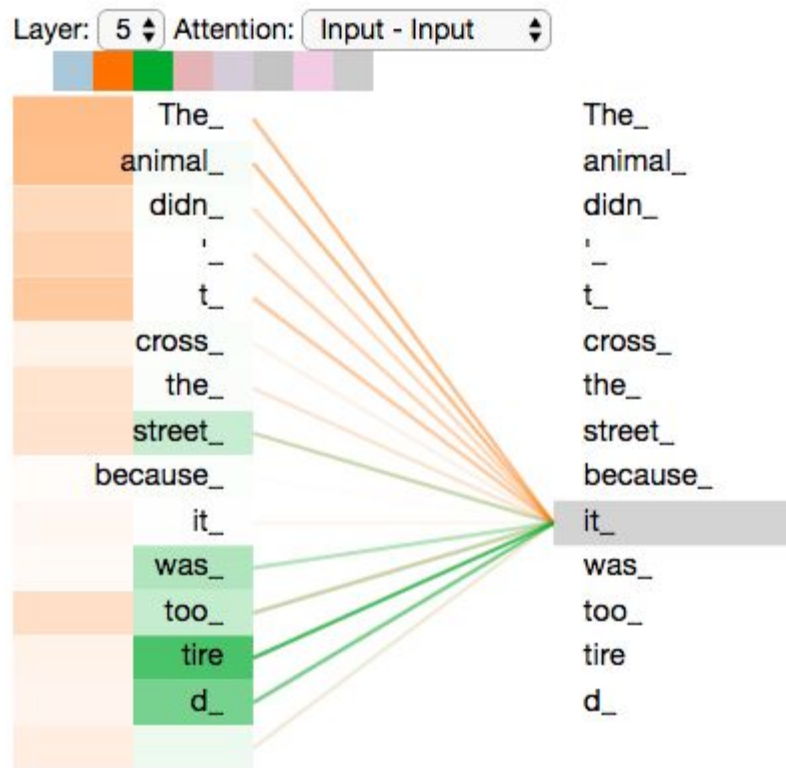


* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one



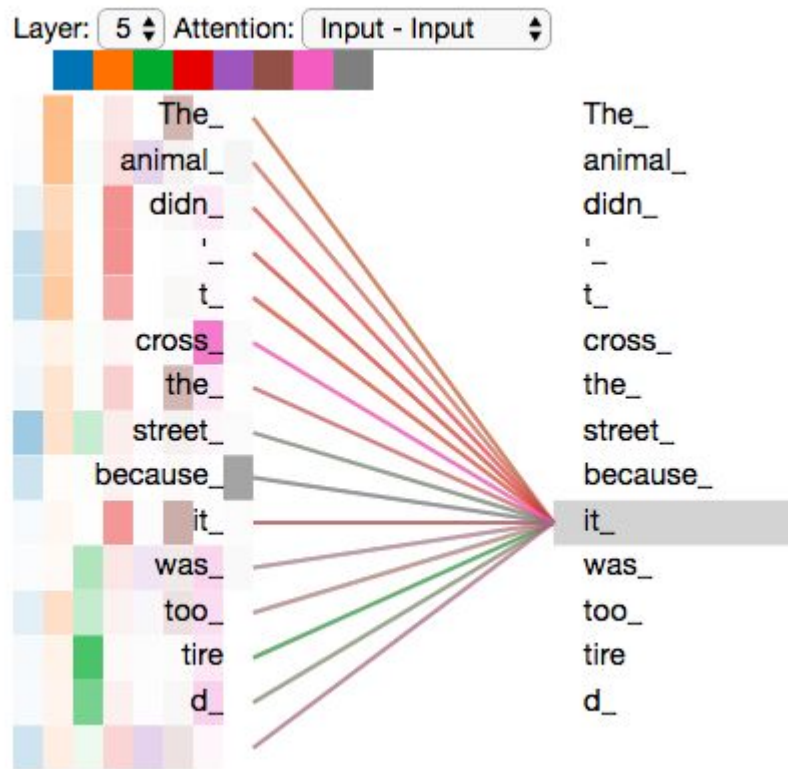
Credits:
<http://jalammar.github.io/illustrated-bert/>

Multiple attention heads



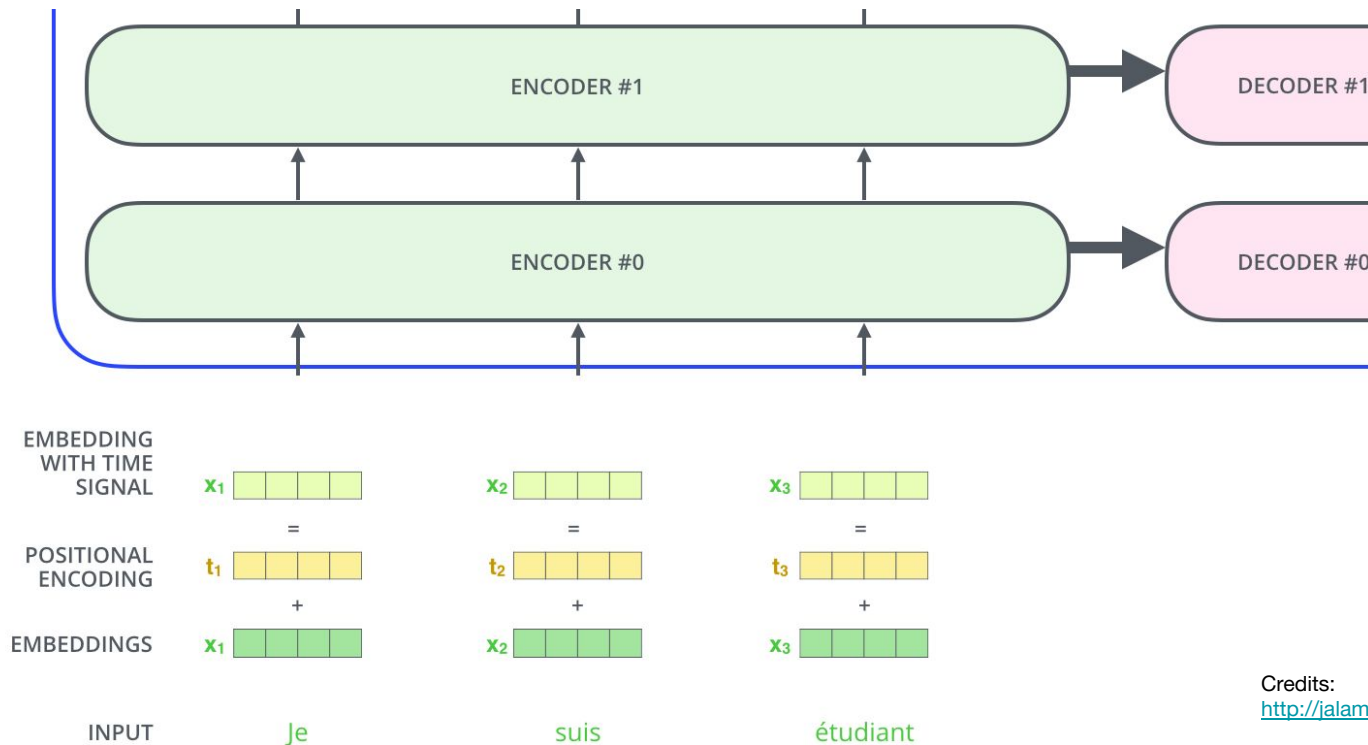
Credits:
<http://jalammar.github.io/illustrated-bert/>

Multiple attention heads



Credits:
<http://jalammar.github.io/illustrated-bert/>

Positional Embedding



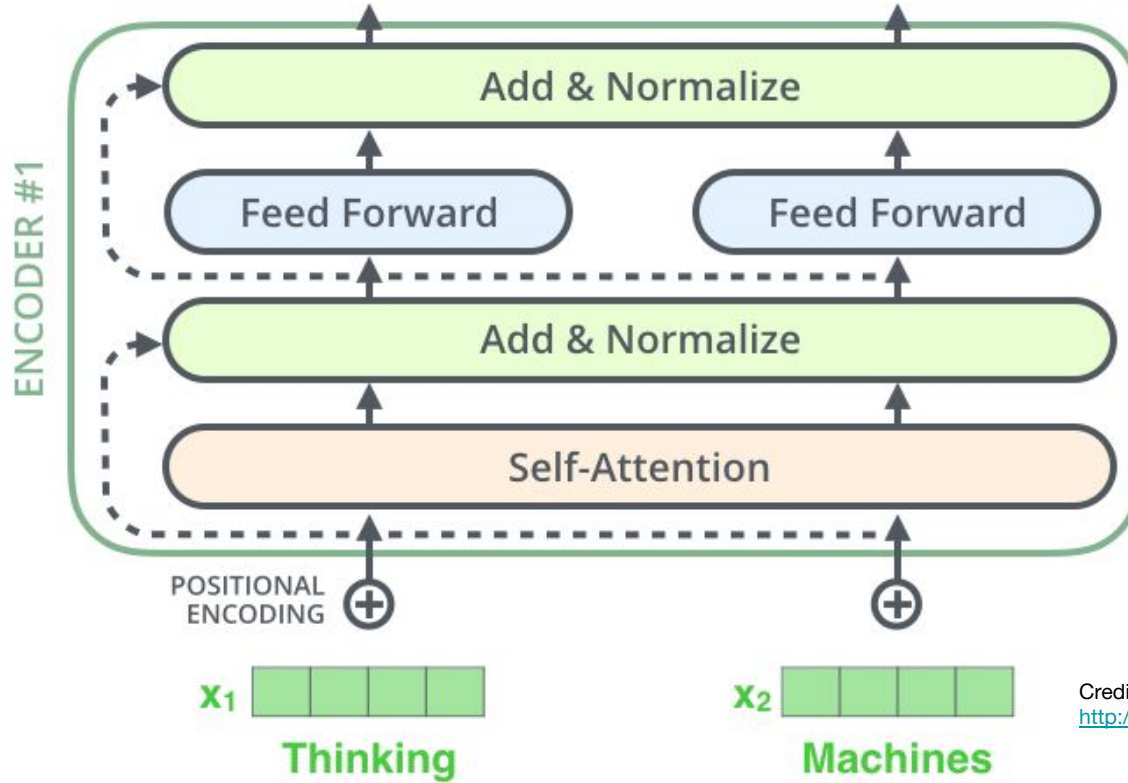
Credits:
<http://jalammar.github.io/illustrated-bert/>

Positional Embedding



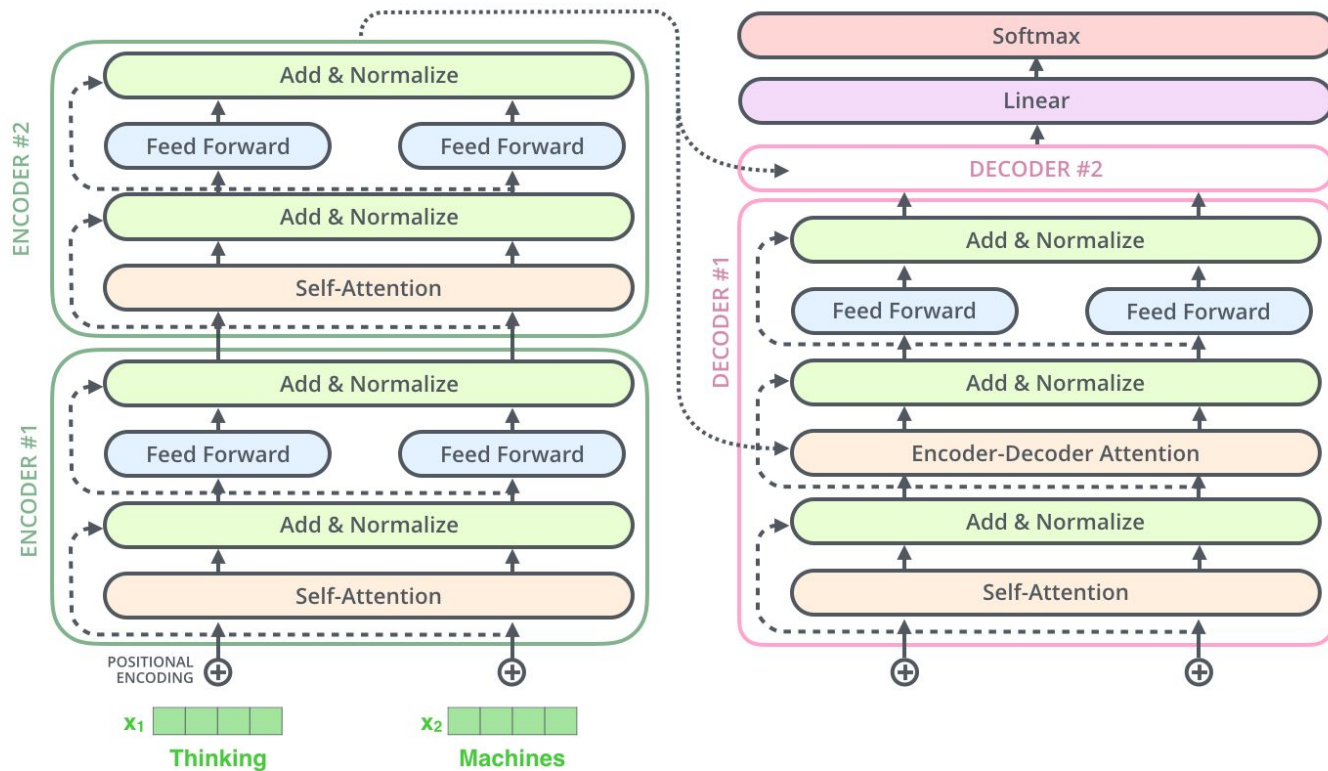
Credits:
<http://jalammar.github.io/illustrated-bert/>

Residual Connections



Credits:
<http://jalammar.github.io/illustrated-bert/>

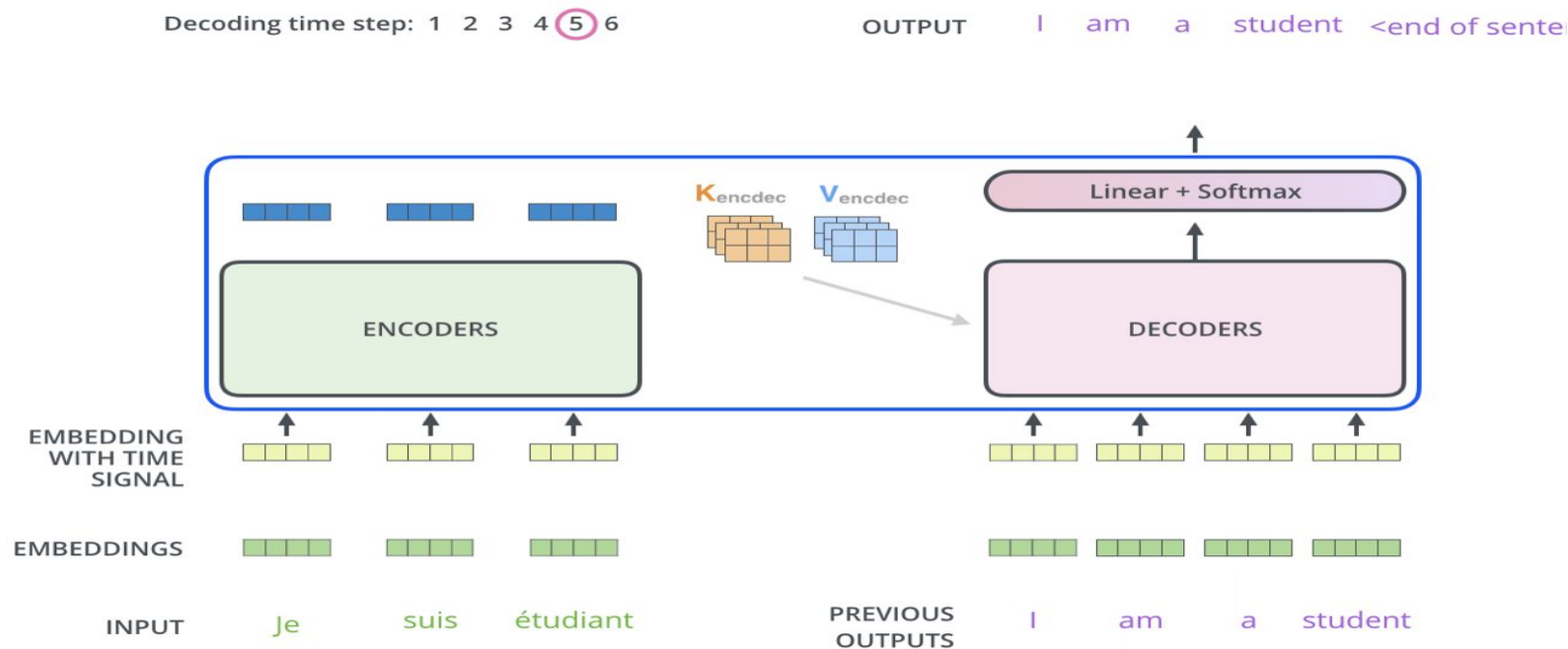
Residual Connections



Credits:

<http://jalammar.github.io/illustrated-bert/>

The Decoder



Credits:
<http://jalammar.github.io/illustrated-bert/>

The BERT!

Bidirectional Encoder Representations from Transformers

- Bidirectional - because it looks at both, the left and the right side of the word for context.
- It does this using the self-attention layer.

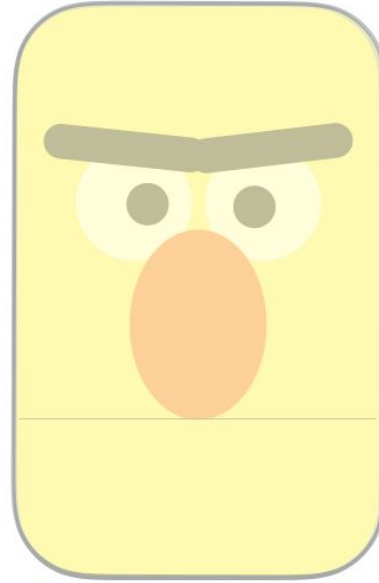
Transfer Learning

- Transfer Learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.
- Researchers cracked Transfer Learning in NLP using the BERT model.

BERT Variants



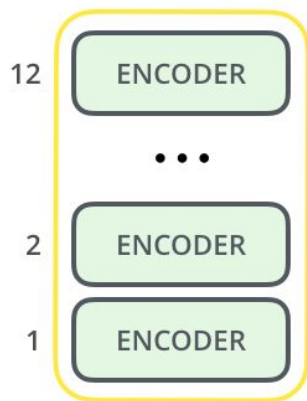
BERT_{BASE}



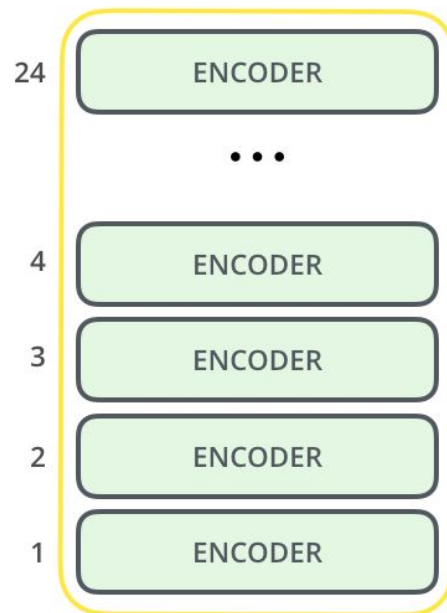
BERT_{LARGE}

Credits:
<http://jalammar.github.io/illustrated-bert/>

BERT Variants



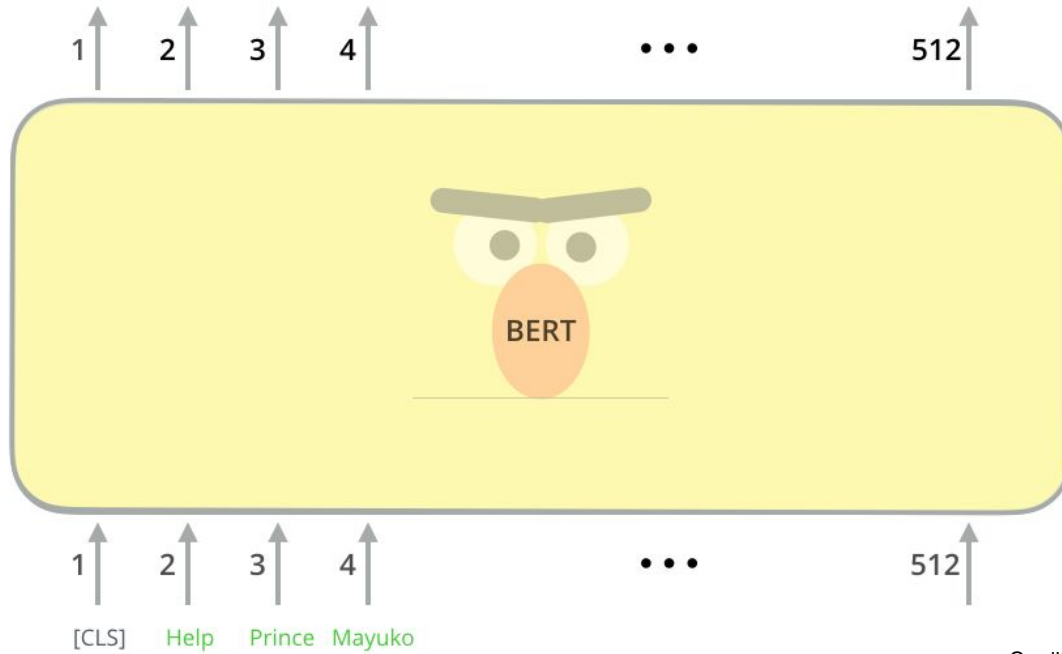
BERT_{BASE}



BERT_{LARGE}

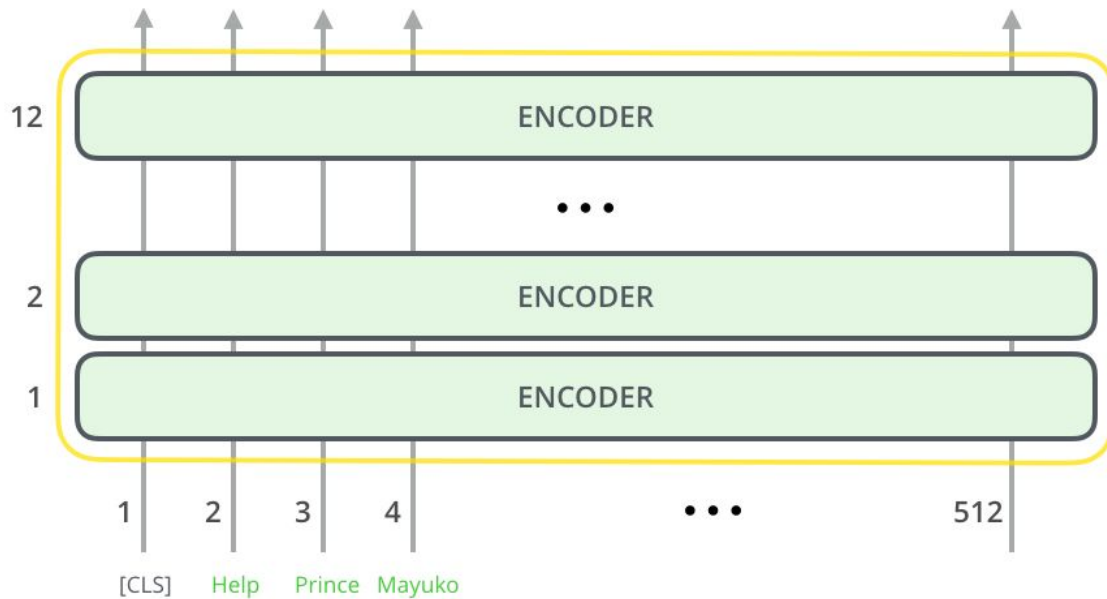
Credits:
<http://jalammar.github.io/illustrated-bert/>

Special Token - CLS



Credits:
<http://jalammar.github.io/illustrated-bert/>

Special Token - CLS



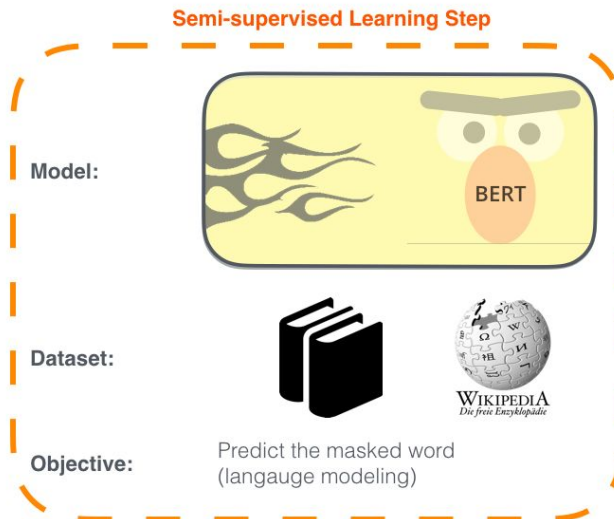
BERT

Credits:
<http://jalammar.github.io/illustrated-bert/>

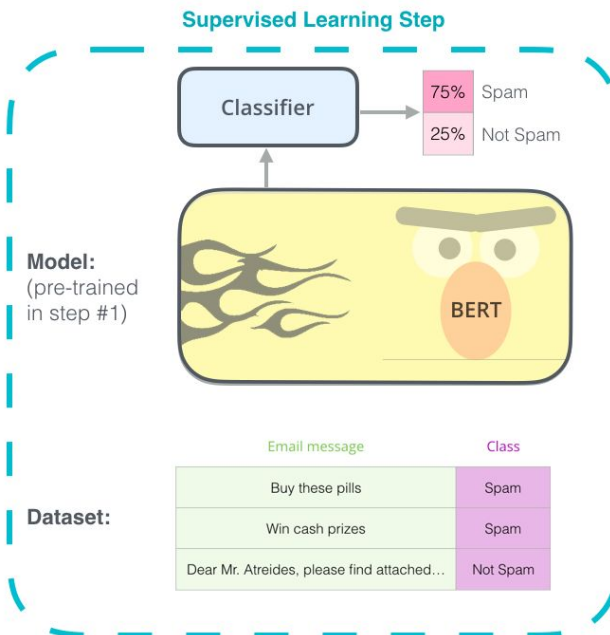
How BERT uses transfer learning?

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



Credits:

<http://jalammar.github.io/illustrated-bert/>

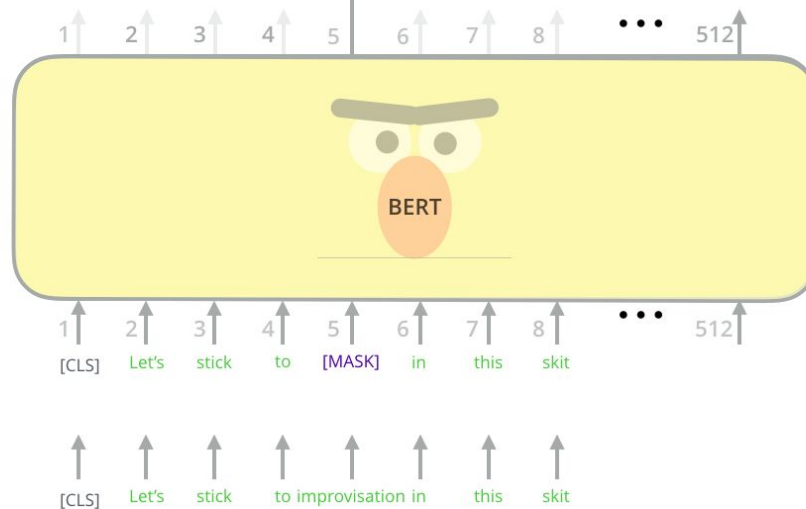
BERT pre-training: Masked Language Model

Use the output of the masked word's position to predict the masked word

Possible classes:
All English words

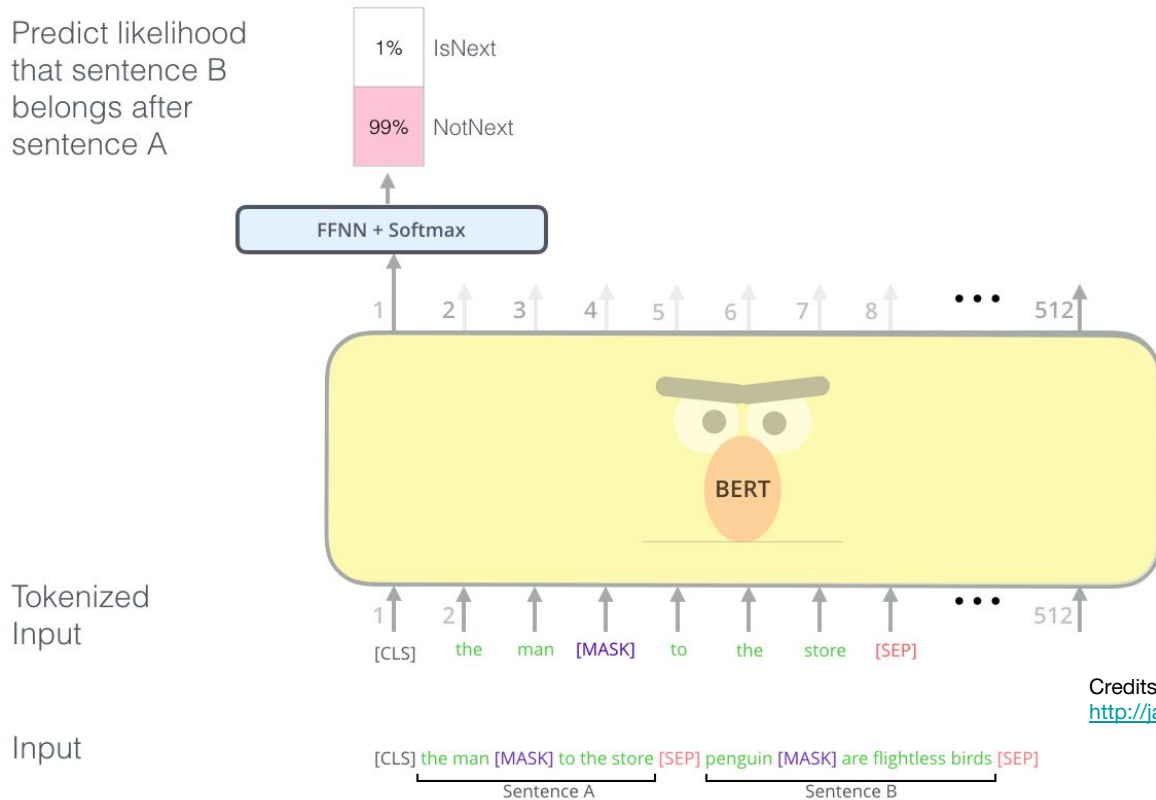
0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzzzva

FFNN + Softmax



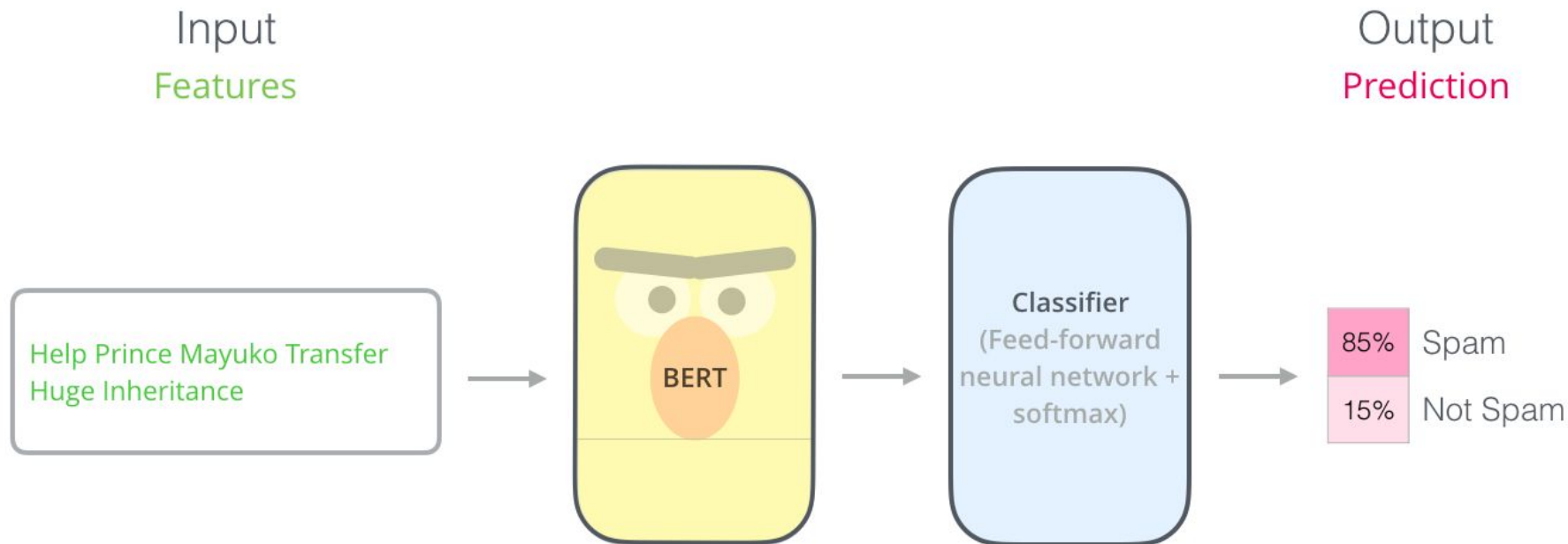
Credits:
<http://jalammar.github.io/illustrated-bert/>

BERT pre-training: Next sentence/not prediction with special token SEP



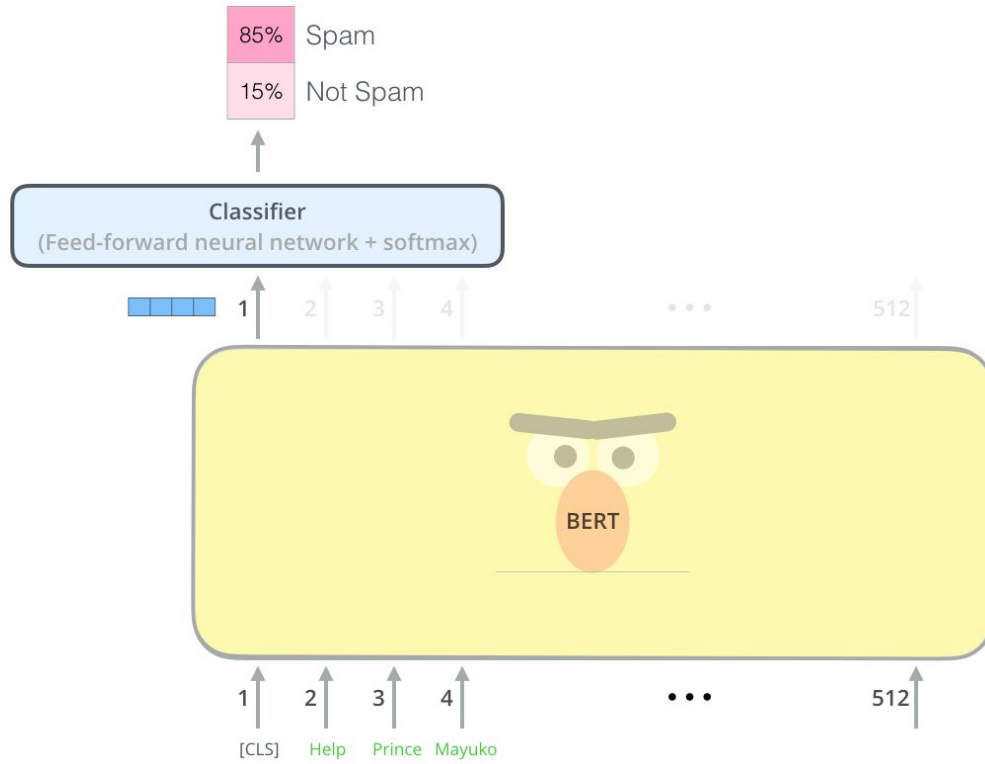
Credits:
<http://jalammar.github.io/illustrated-bert/>

Classification Example



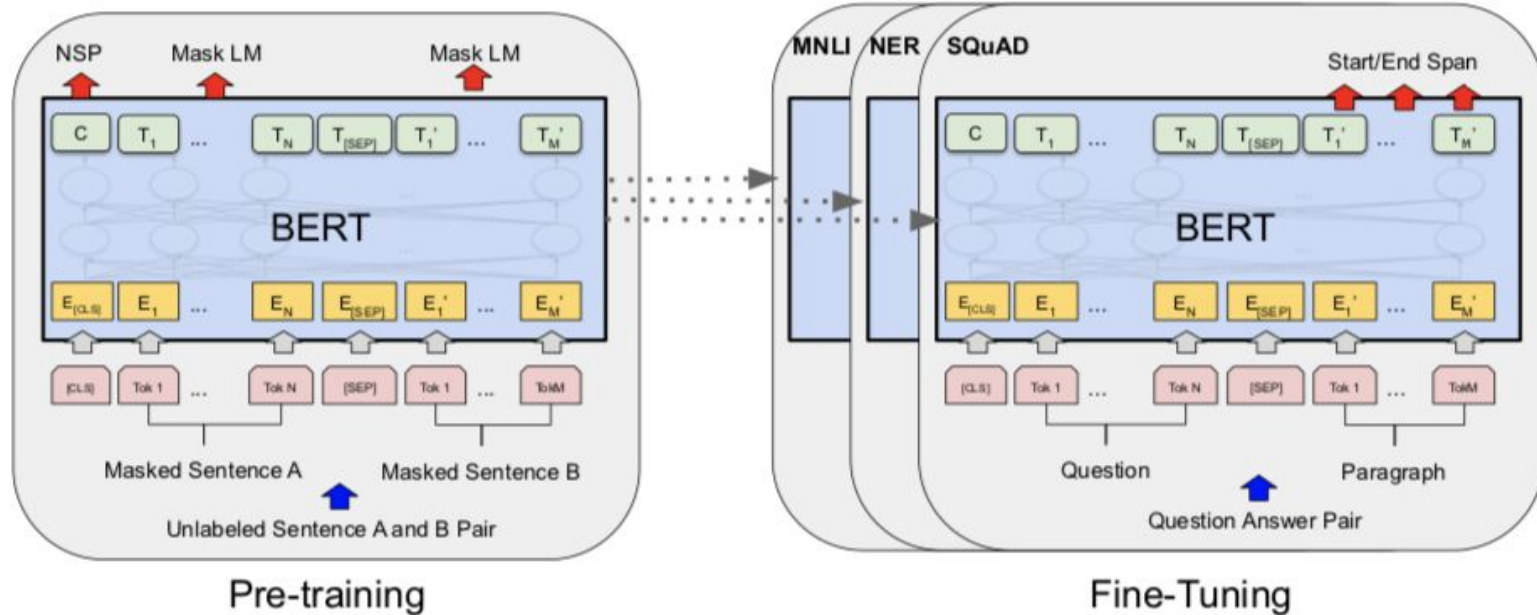
Credits:
<http://jalammar.github.io/illustrated-bert/>

Special Token - CLS for classification

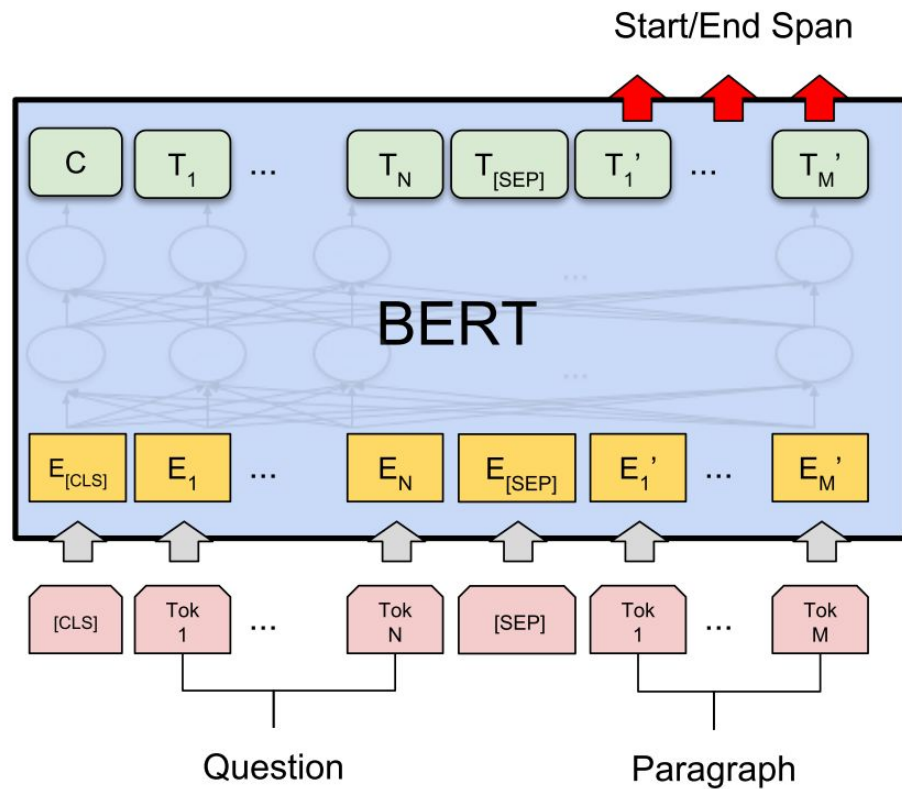


Credits:
<http://jalammar.github.io/illustrated-bert/>

Task specific BERT



BERT for Question-Answering



Text Summarization

Two major types- abstractive and extractive

Extractive - latent semantic analysis or textrank

The one done using Deep Learning is usually abstractive

Sequence to sequence model trained on full articles and their summaries

Tutorials

- NLP Basics: Text vectorization and classification: [Colab link](#)
- Using pretrained word embeddings: [Colab link](#)
- Transformers usage: [Colab link](#)

Directions for Self-Explorations!

- <https://machinelearningmastery.com/what-are-word-embeddings/>
- <https://arxiv.org/pdf/1810.04805.pdf>
- <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>
- <https://jalammar.github.io/illustrated-transformer/>
- <http://jalammar.github.io/illustrated-bert/>

Any Questions?

Thank You!