

# Self Attention & BERT using TensorFlow 2.0



Souradip Chakraborty - Data Scientist @Walmart Labs  
Indian Statistical Institute



**TensorFlow**  
**2.0**

# Session Agenda



- ❖ Intro to Natural Language Processing and Application Domains
- ❖ History & Evolution of NLP Embeddings
- ❖ Sequence Learning Tasks and Recurrent Neural Network(RNNs & LSTMs)
- ❖ Introduction to Attention Mechanism – Attention is all you need !!
- ❖ Self Attention (Single & Multiheaded) & Transformer Architecture
- ❖ State of the Art Language Model for NLP - BERT
- ❖ Code walk-through in TensorFlow 2.0

# Natural Language Processing and Application Domains

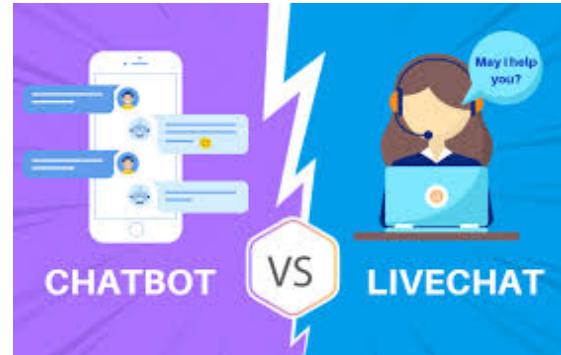
## NLP in Healthcare

### Associated Use Cases



Mainstay use cases of NLP in Healthcare that have a proven ROI

 Speech Recognition	 Improvement in Clinical Documentation	 Data Mining Research
 Computer-assisted Coding	 Automated Registry Reporting	



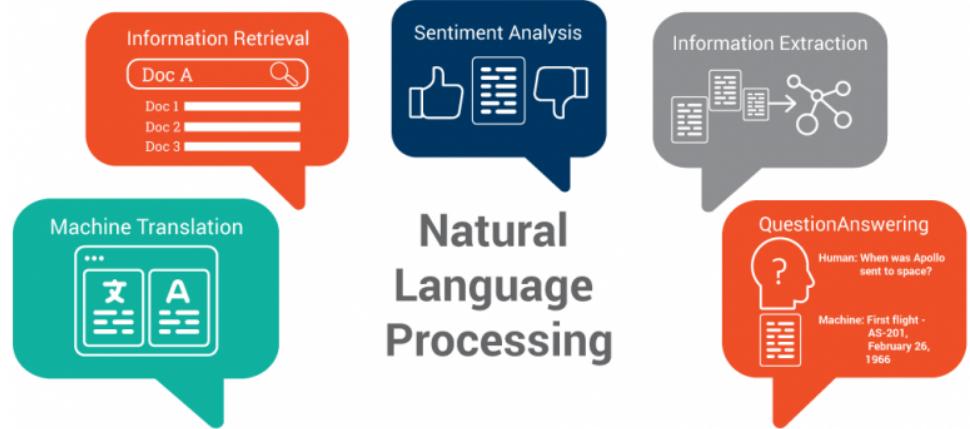
### Emerging use cases of NLP in Healthcare that will have an immediate impact

 Clinical Trial Matching	 Prior Authorization	 Clinical Decision Support	 Risk Adjustment and Hierarchical Condition Categories
---	---	---	---

### Next-gen use cases of NLP in Healthcare that are on the horizon

 Ambient Virtual Scribe	 Computational Phenotyping and Biomarker Discovery	 Population Health Management & Analysis
--	---	---

## Natural Language Processing



Information Retrieval

Doc A

Doc 1  
Doc 2  
Doc 3

Sentiment Analysis

Information Extraction

Machine Translation

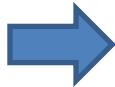
Human: When was Apollo sent to space?  
Machine: First flight - AS-201, February 26, 1966

# History & Evolution of NLP Embeddings – Bag of Words approach

**1. Term Frequency Vectorizer** - It only takes into consideration the frequency of words in a document. A document's important words are words that occur a lot of times in said document.

Suppose our set of documents consists of three sentences:

1. I love dogs.
2. I hate dogs and knitting.
3. Knitting is my hobby and my passion.



	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

**2. Term Frequency Inverse Document Vectorizer** - It takes into consideration the frequency of words in a document as well as the occurrence of that word in other documents. A document's important words are words that occur a lot of times in said document and must be absent in the other documents .

$$w_{x,y} = tf_{x,y} \times \log \left( \frac{N}{df_x} \right)$$

**TF-IDF**

Term  $x$  within document  $y$

$tf_{x,y}$  = frequency of  $x$  in  $y$

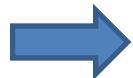
$df_x$  = number of documents containing  $x$

$N$  = total number of documents

**1. I love dogs.**

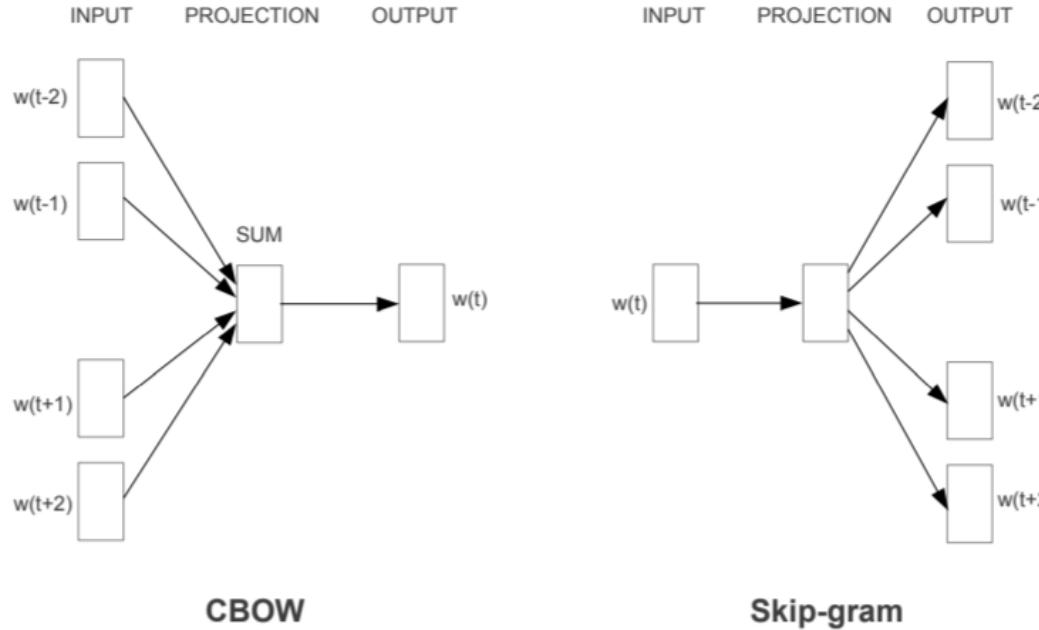
**2. I hate dogs and knitting.**

**3. Knitting is my hobby and my passion.**



	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	0.18	<b>0.48</b>	0.18							
Doc 2	0.18		0.18	<b>0.48</b>	0.18	0.18				
Doc 3					0.18	0.18	<b>0.48</b>	<b>0.95</b>	<b>0.48</b>	<b>0.48</b>

## Continuous Bag of Words Model (CBOW) and Skip-gram

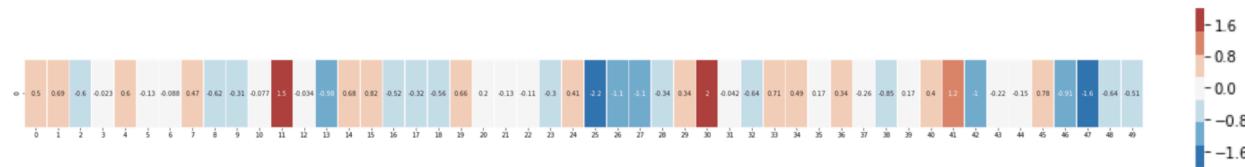


In the CBOW model, the distributed representations of context (or surrounding words) are combined to predict the word in the middle. While in the Skip-gram model, the distributed representation of the input word is used to predict the context.

# History & Evolution of NLP Embeddings – Word Embeddings

## Glove Embeddings (Trained on Wikipedia) – Visual Explanation

Let's color code the cells based on their values (red if they're close to 2, white if they're close to 0, blue if they're close to -2):



We'll proceed by ignoring the numbers and only looking at the colors to indicate the values of the cells. Let's now contrast "King" against other words:

**"king"**



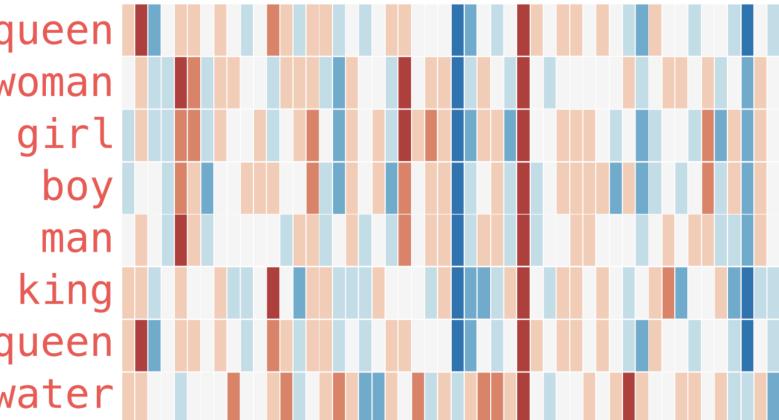
**"Man"**



**"Woman"**



Ref : <http://jalammar.github.io/illustrated-word2vec/>



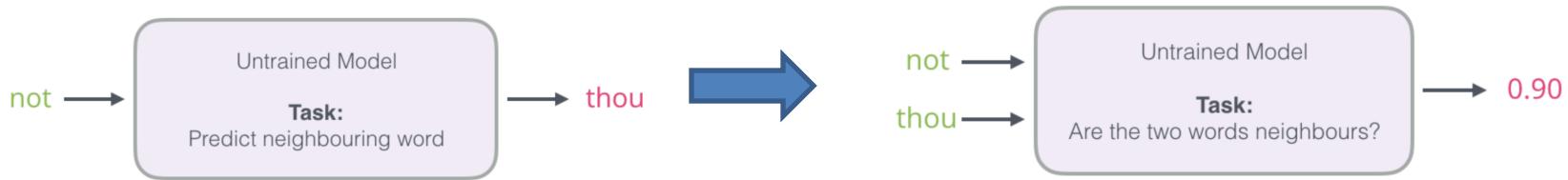
king – man + woman  $\approx$  queen



Ref : <http://jalammar.github.io/illustrated-word2vec/>

# History & Evolution of NLP Embeddings – Negative Sampling

Change Task from



input word	target word	
not	thou	
not	shalt	
not	make	
not	a	
make	shalt	
make	not	
make	a	
make	machine	

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Ref : <http://jalammar.github.io/illustrated-word2vec/>

# History & Evolution of NLP Embeddings – Negative Sampling

**Problem** : One major drawback of the approach is if we don't have negative samples, model will generate useless embeddings as it is a trivial problem of single class.

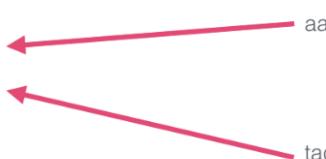
**Negative Sample Mining** : To solve the problem, we introduce *negative samples* to our dataset – samples of words that are not neighbours, which will be labelled as 0.

Pick randomly from vocabulary  
(random sampling)

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Word Count Probability

aardvark  
aarhus  
aaron  
taco  
thou  
zyzzyva



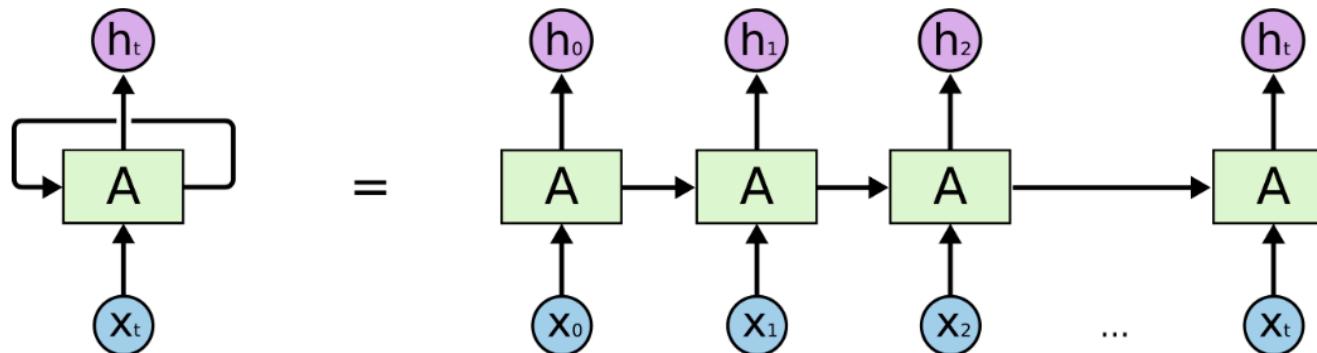
Ref : <http://jalammar.github.io/illustrated-word2vec/>

# Sequence Learning Tasks and Recurrent Neural Network(RNNs & LSTMs)

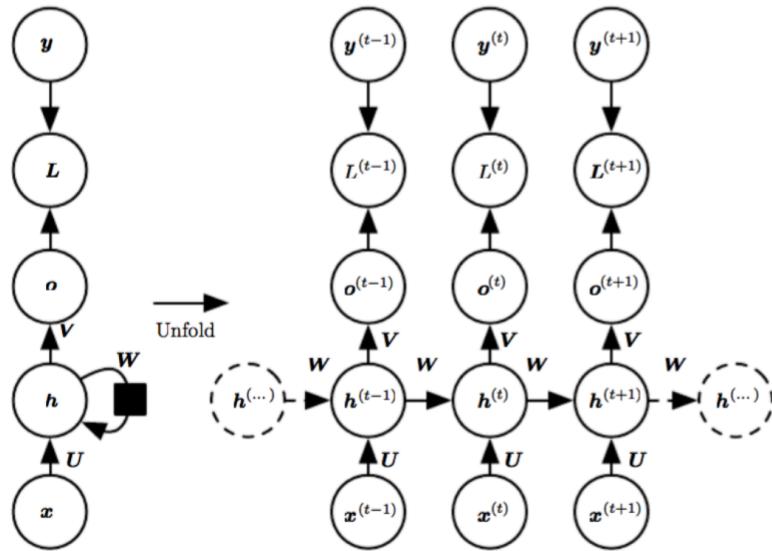
**Recurrent Neural Network** : The inherent sequential nature of the Natural language is being captured by RNNs.

Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

Traditional neural networks can't do this, and they don't have the power to use the sequence/dependencies of previous state



Ref : <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



$$\begin{aligned}
 \mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \\
 \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\
 \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \\
 \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)})
 \end{aligned}$$

Loss computation :  $L(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}) = \sum_t L^{(t)} = \sum_t -\log \hat{y}_{y^{(t)}}^{(t)}.$

# Introduction to Attention Mechanism – Attention is all you need !!

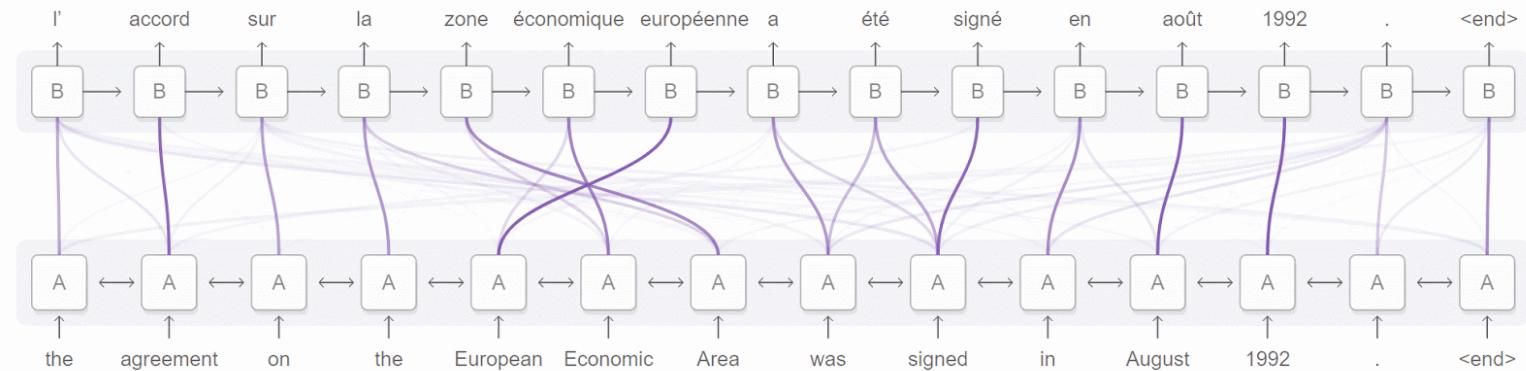
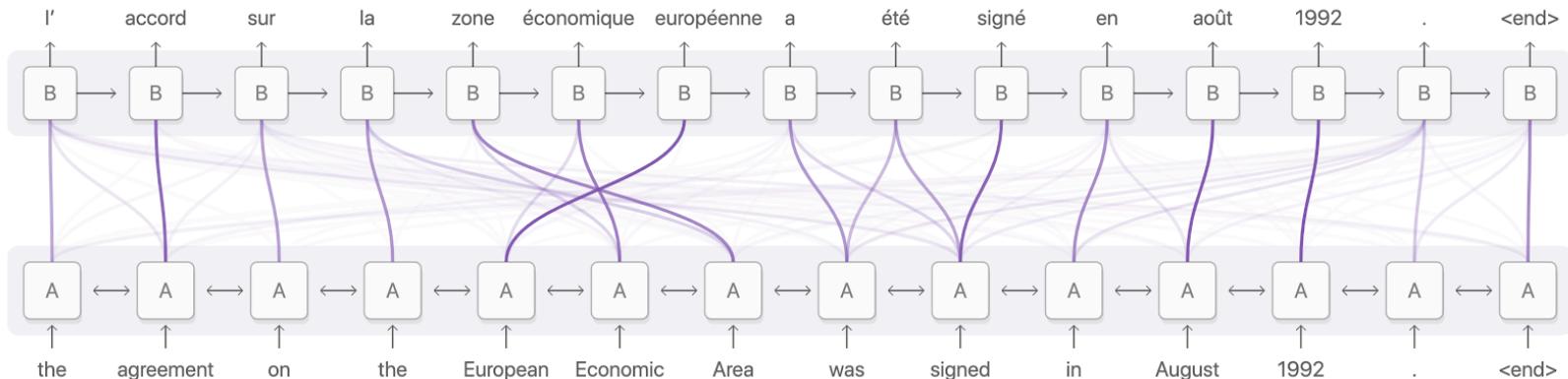


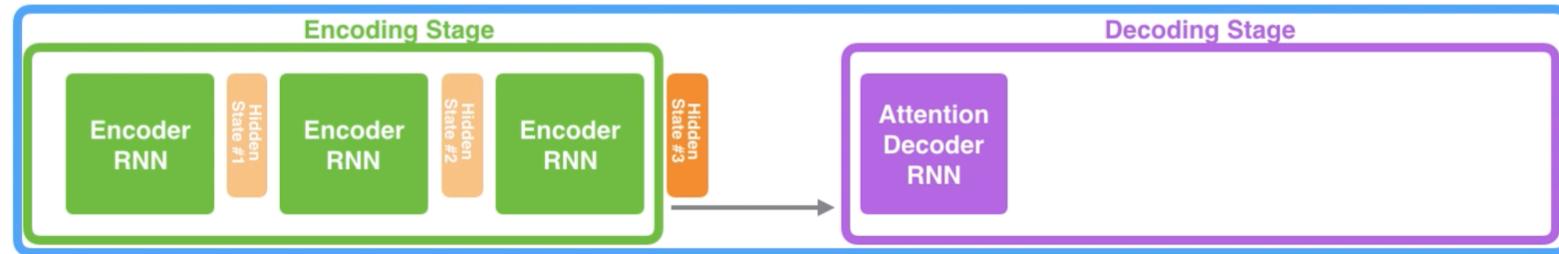
Diagram derived from Fig. 3 of Bahdanau, et al. 2014



Ref : <http://jalammar.github.io/illustrated-transformer/>

# Sequence to Sequence Model with Attention

## Neural Machine Translation SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



I                    am                    a                    student



Je                    suis                    etudiant

Ref : <http://jalammar.github.io/illustrated-transformer/>

# Introduction to Attention Mechanism – Obtaining Context Vector using Attention

## Global Attention

1. Prepare inputs



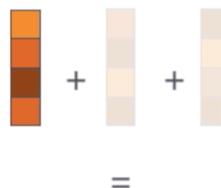
2. Score each hidden state



3. Softmax the scores



4. Multiply each vector by its softmaxed score

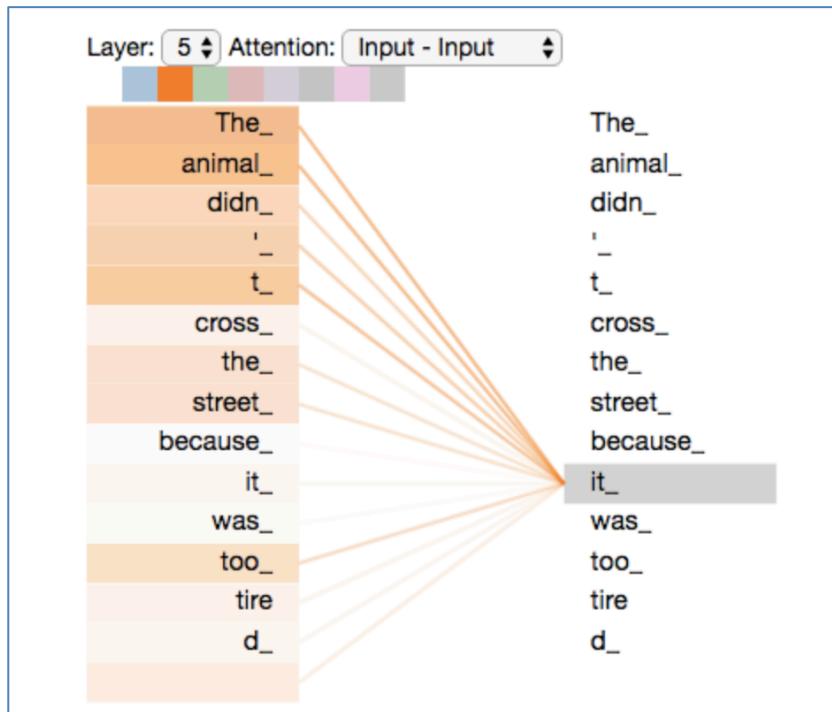


5. Sum up the weighted vectors

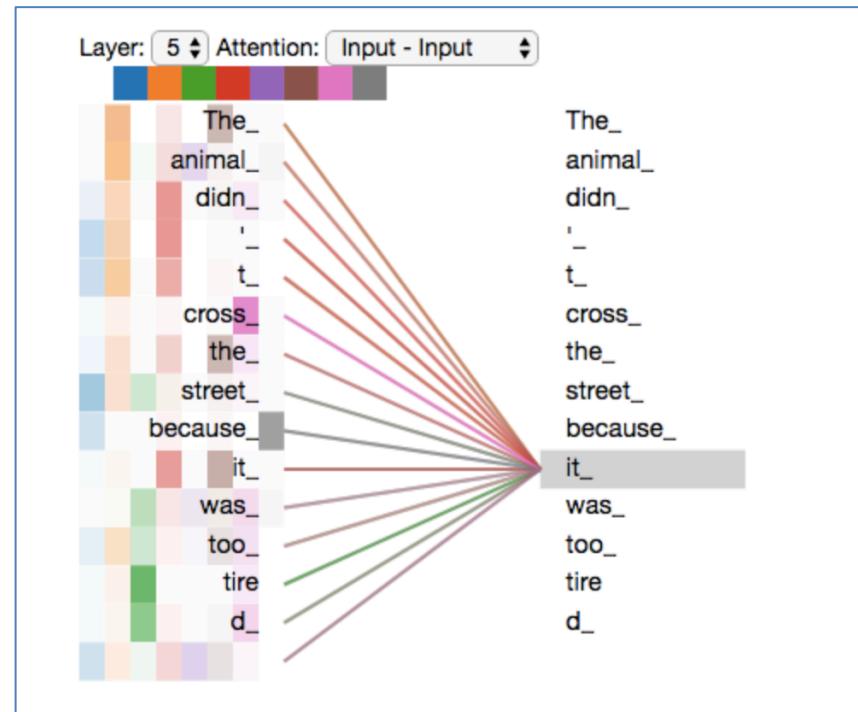
Context vector for decoder time step #4

## Self Attention & Transformers

Single Head Self Attention Block



Multi Head Self Attention Block



Ref : <http://jalammar.github.io/illustrated-transformer/>

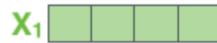
## Self Attention & Transformers - Steps

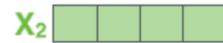
Input

Thinking

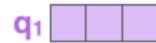
Machines

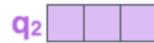
Embedding

$X_1$  A horizontal row of four green squares representing the embedding vector  $X_1$ .

$X_2$  A horizontal row of four green squares representing the embedding vector  $X_2$ .

Queries

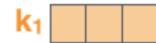
$q_1$  A horizontal row of three purple squares representing the query vector  $q_1$ .

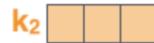
$q_2$  A horizontal row of three purple squares representing the query vector  $q_2$ .



$W^Q$

Keys

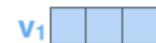
$k_1$  A horizontal row of three orange squares representing the key vector  $k_1$ .

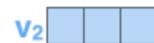
$k_2$  A horizontal row of three orange squares representing the key vector  $k_2$ .



$W^K$

Values

$v_1$  A horizontal row of three blue squares representing the value vector  $v_1$ .

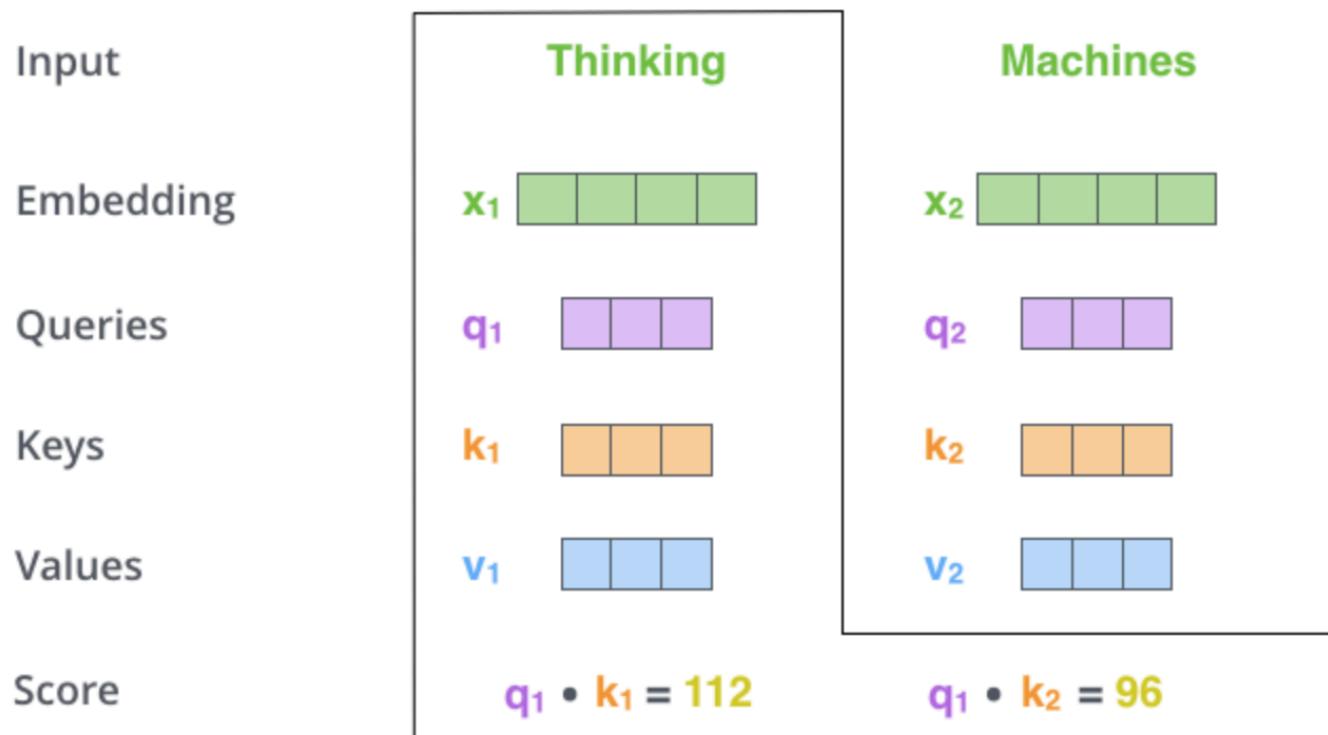
$v_2$  A horizontal row of three blue squares representing the value vector  $v_2$ .



$W^V$

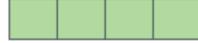
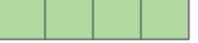
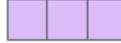
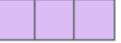
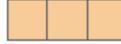
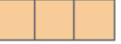
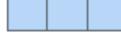
Ref : <http://jalammar.github.io/illustrated-transformer/>

## Self Attention & Transformers - Steps



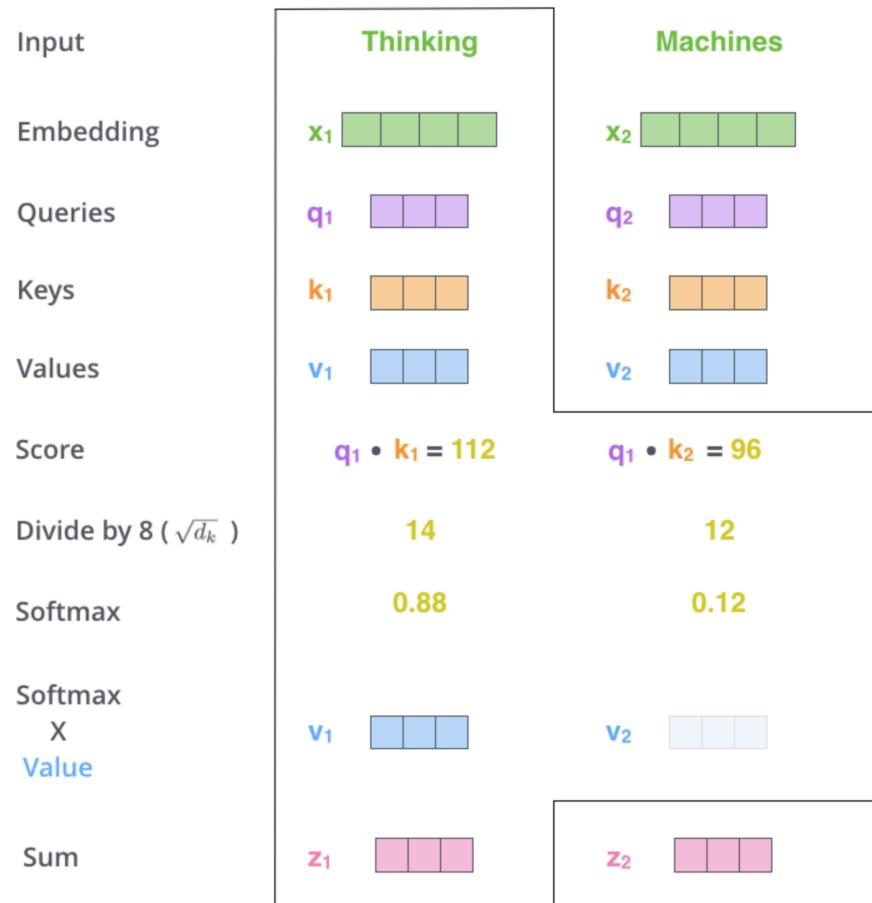
Ref : <http://jalammar.github.io/illustrated-transformer/>

## Self Attention & Transformers - Steps

Input	<b>Thinking</b> $x_1$ 		<b>Machines</b> $x_2$ 	
Embedding	$q_1$		$q_2$	
Keys	$k_1$		$k_2$	
Values	$v_1$		$v_2$	
Score	$q_1 \cdot k_1 = 112$		$q_1 \cdot k_2 = 96$	
Divide by 8 ( $\sqrt{d_k}$ )	14		12	
Softmax	0.88		0.12	

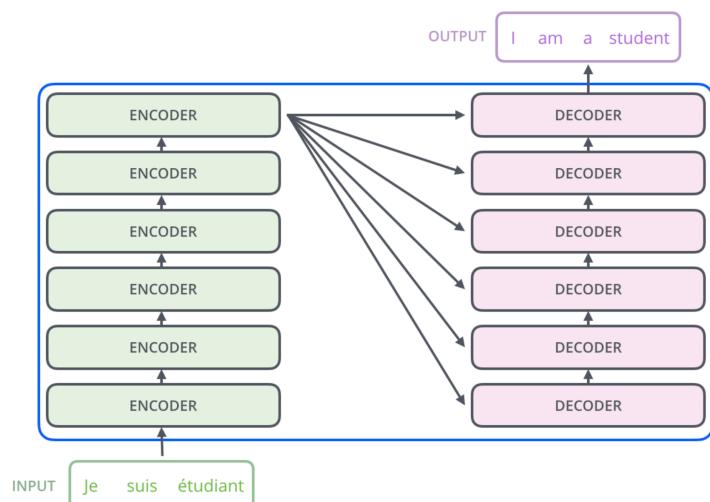
Ref : <http://jalammar.github.io/illustrated-transformer/>

## Self Attention & Transformers - Steps



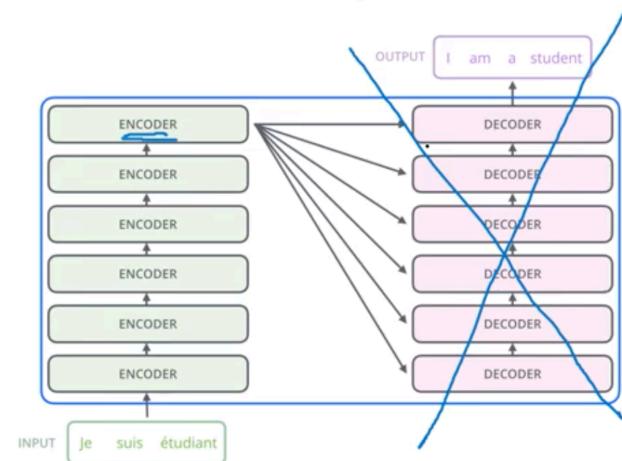
## Bidirectional Encoder Representations from Transformers

### Transformer Architecture



### BERT Architecture

*Attention is all you need*



## Bidirectional Encoder Representations from Transformers

### BERT's "Fake Tasks"

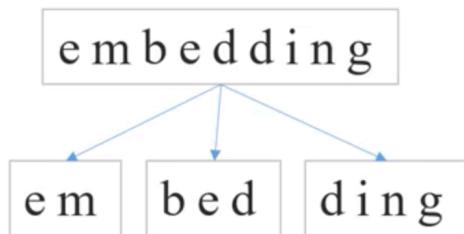
1. Masked Language Model
  - Predict the masked word.
2. Next Sentence Prediction
  - Was sentence B found immediately after sentence A?

## Bidirectional Encoder Representations from Transformers – Word Piece Tokenizer

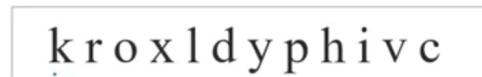
### BERT's Vocabulary

1. BERT is pre-trained → Vocabulary is fixed

2. Break down **unknown words** into **subwords**:

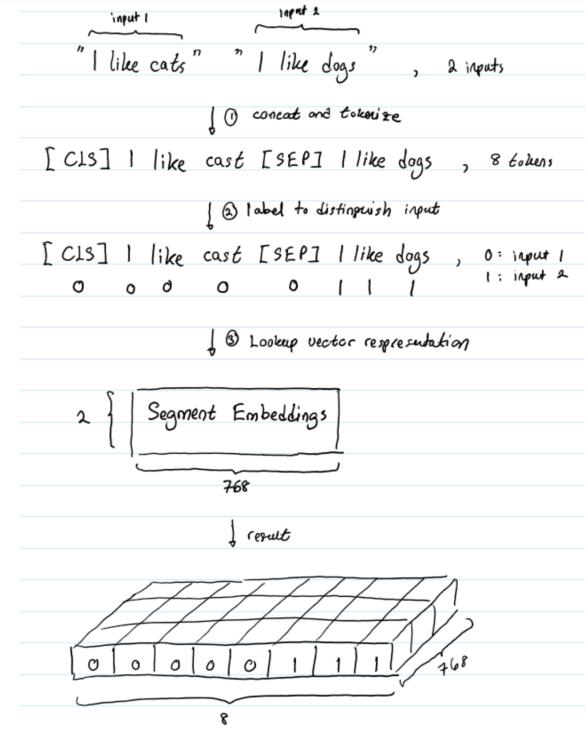
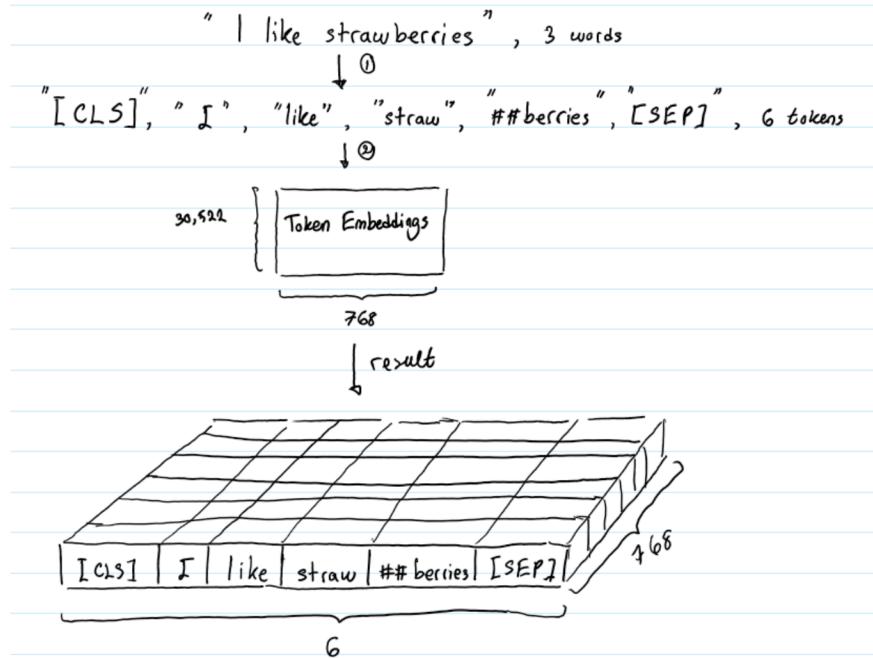


3. A subword exists for every character:



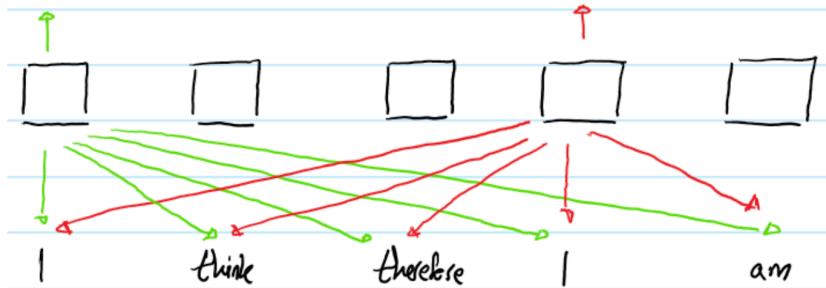
# Introduction to Attention Mechanism – Attention is all you need !!

## Bidirectional Encoder Representations from Transformers – Word Piece Tokenizer



[https://medium.com/@\\_init\\_/how-self-attention-with-relative-position-representations-works-28173b8c245a](https://medium.com/@_init_/how-self-attention-with-relative-position-representations-works-28173b8c245a)

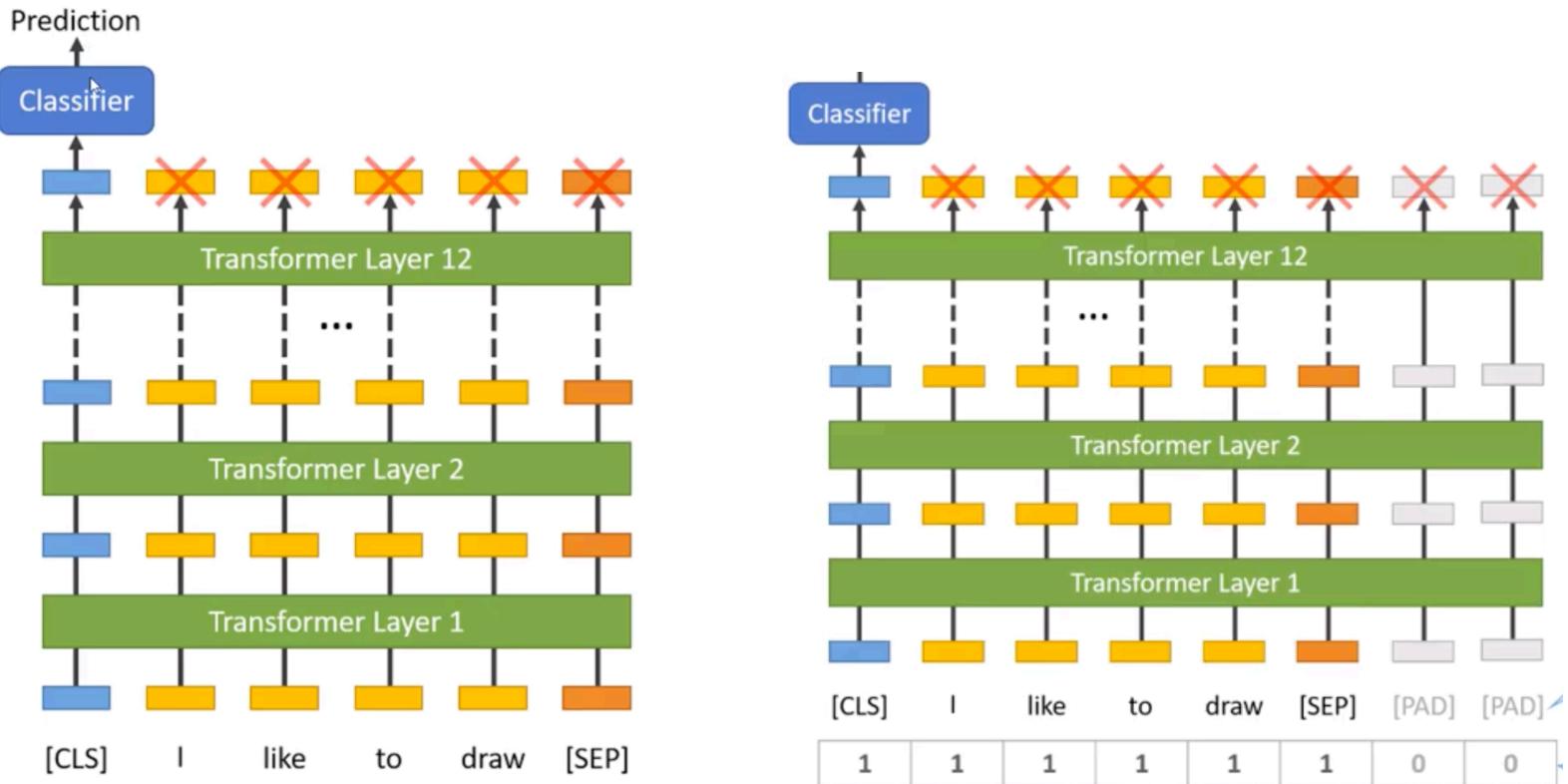
## Bidirectional Encoder Representations from Transformers – Position Embedding



<u>Index</u>	<u>Interpretation</u>
0	dist between word at position i and i - 4
1	dist between word at position i and i - 3
2	dist between word at position i and i - 2
3	dist between word at position i and i - 1
4	dist between word at position i and i
5	dist between word at position i and i + 1
6	dist between word at position i and i + 2
7	dist between word at position i and i + 3
8	dist between word at position i and i + 4

# Introduction to Attention Mechanism – Attention is all you need !!

## Bidirectional Encoder Representations from Transformers – Tokens



## Code walkthrough. !!!!



## Thank you