



Transformers and Attention

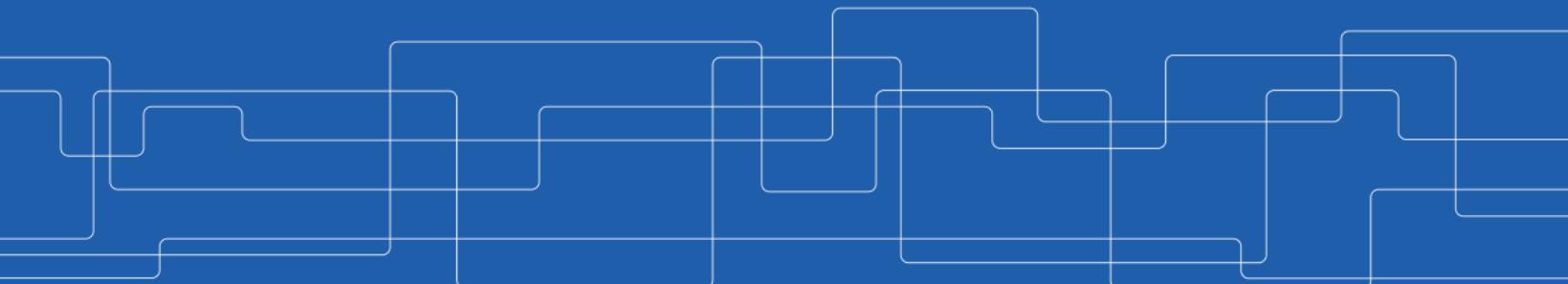
ID2223 Scalable Machine Learning and Deep Learning

Francisco J. Peña

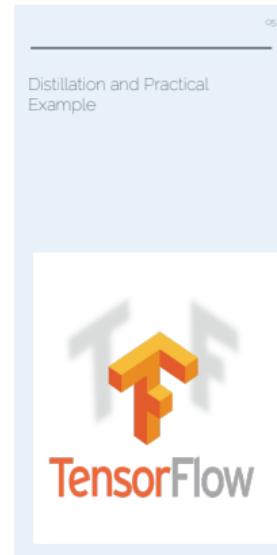
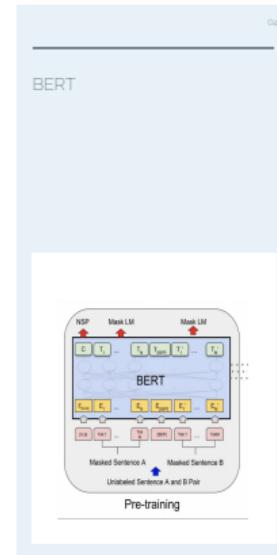
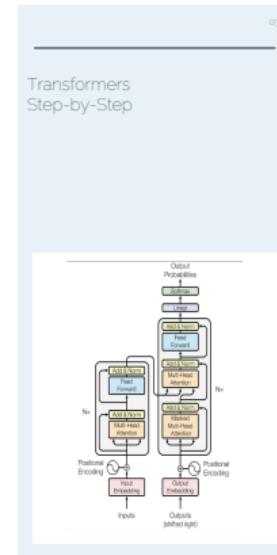
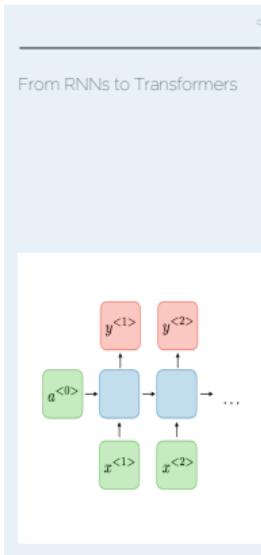
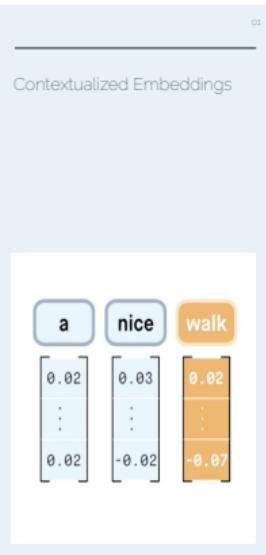
Postdoctoral Researcher, KTH

frape@kth.se

2021-12-02



Roadmap





Acknowledgements

Material based on:

- ▶ Christoffer Manning's [NLP Lectures at Stanford](#)
- ▶ [The Illustrated Transformer](#) by Jay Alammar
- ▶ [Slides](#) from Jacob
- ▶ [Self-attention Video](#) from Peltarion
- ▶ Slides from Karl Erliksson



Contextualized Embeddings

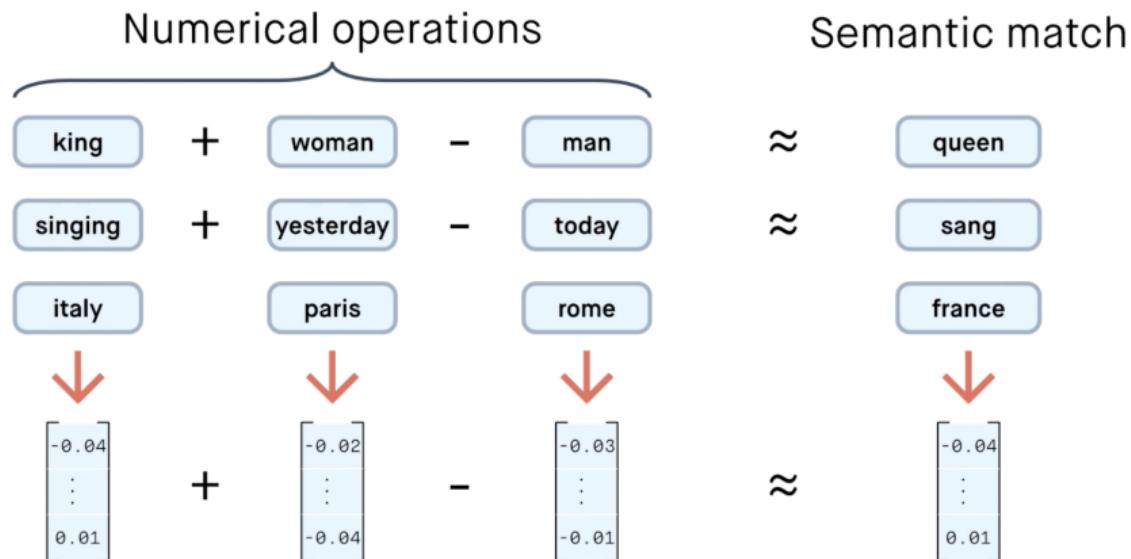
Background to Natural Language Processing (NLP)

- ▶ Word embeddings are the basis of NLP
- ▶ Popular embeddings like GloVe and Word2Vec are pre-trained on large text corpuses based on co-occurrence statistics
- ▶ “A word is characterized by the company it keeps” [Firth, 1957]

best	-	selling	music	artists
-0.11	0.01	-0.01	0.06	-0.02
0.01	0.07	-0.03	0.11	0.00
-0.17	-0.04	0.15	0.05	-0.05
:	:	:	:	:
0.13	-0.05	0.00	0.14	0.05
-0.13	-0.11	-0.07	-0.12	-0.12
-0.09	-0.25	0.05	-0.04	0.02

[Peltarion, 2020]

Word Embeddings



[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are **context-free**

a	nice	walk	by	the	river	bank
0.02	0.03	0.02	-0.00	-0.04	-0.01	-0.02
:	:	:	:	:	:	:
0.02	-0.02	-0.07	0.03	-0.03	-0.04	-0.03

walk	to	the	bank	and	get	cash
0.02	0.01	-0.04	-0.02	-0.02	-0.06	0.01
:	:	:	:	:	:	:
-0.07	0.02	-0.03	-0.03	0.02	0.04	-0.01

[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are **context-free**

a	nice	walk	by	the	river	bank
0.02	0.03	0.02	-0.00	-0.04	-0.01	-0.02
:	:	:	:	:	:	:
0.02	-0.02	-0.07	0.03	-0.03	-0.04	-0.03

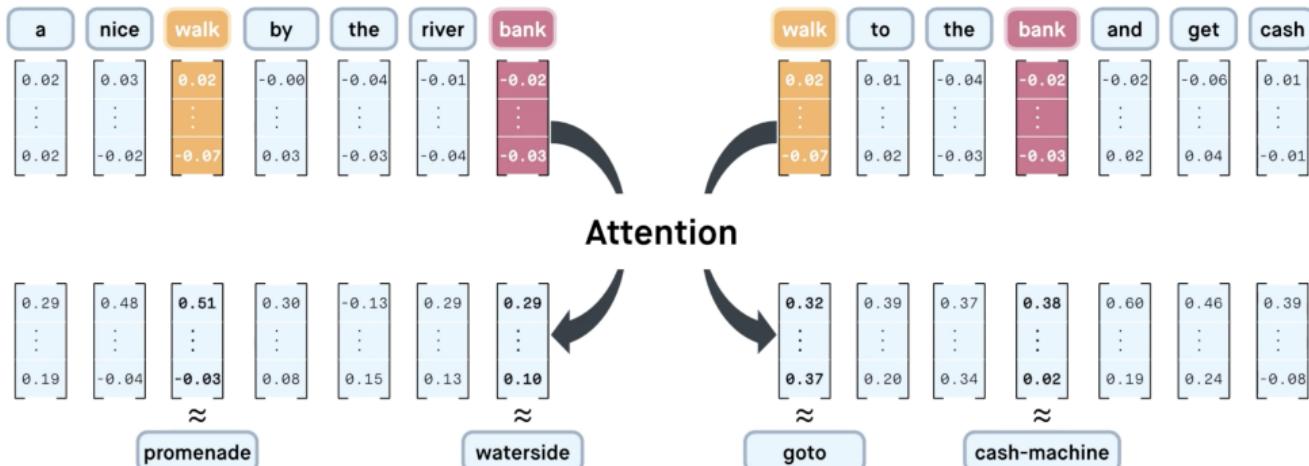
walk	to	the	bank	and	get	cash
0.02	0.01	-0.04	-0.02	-0.02	-0.06	0.01
:	:	:	:	:	:	:
-0.07	0.02	-0.03	-0.03	0.02	0.04	-0.01

[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are **context-free**

Solution: Create **contextualized** representation



[Peltarion, 2020]



From RNNs to Transformers

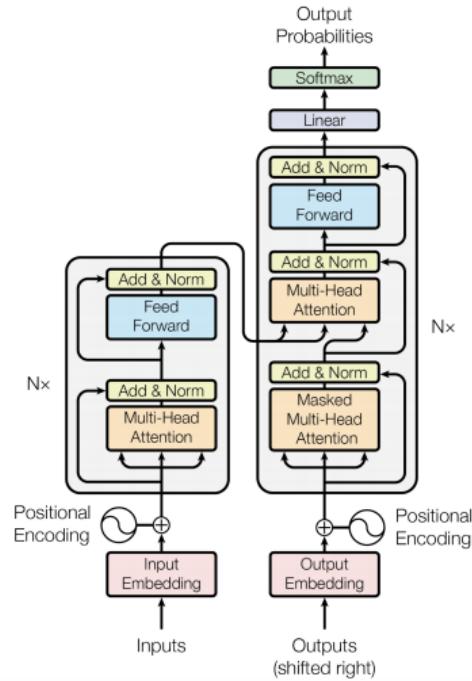


Problems with RNNs - Motivation for Transformers

- ▶ Sequential computations **prevents parallelization**
- ▶ Despite GRUs and LSTMs, RNNs still need attention mechanisms to deal with **long range dependencies**
- ▶ Attention gives us access to any state... Maybe we don't need the costly recursion?
- ▶ Then NLP can have deep models, solves our computer vision envy!

Attention is all you need! [Vaswani, 2017]

- ▶ Sequence-to-sequence model for Machine Translation
- ▶ Encoder-decoder architecture
- ▶ Multi-headed **self-attention**
 - Models context and no locality bias



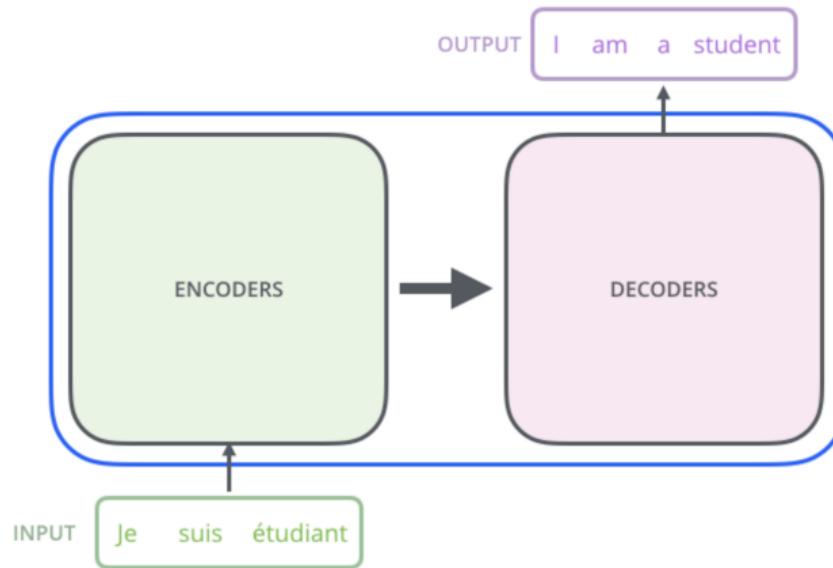
[Vaswani et al., 2017]



Transformers Step-by-Step



Understanding the Transformer: Step-by-Step

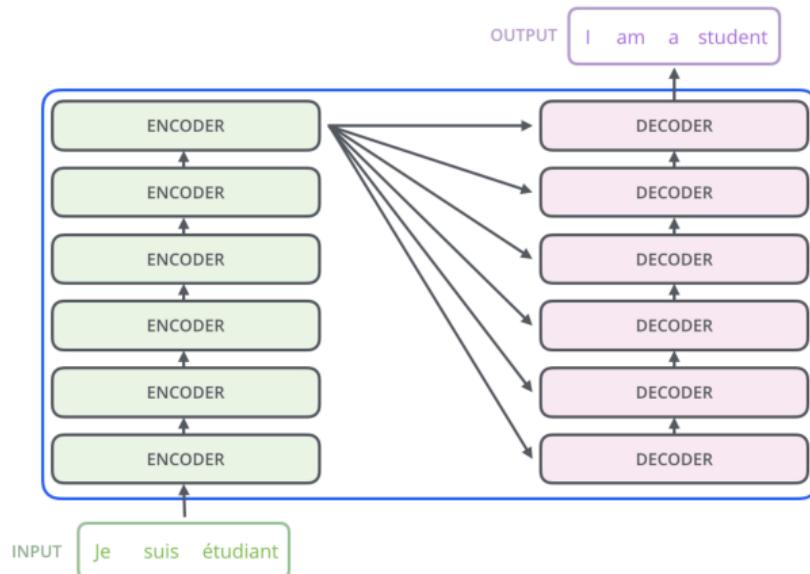


[Alammar, 2018]

Understanding the Transformer: Step-by-Step

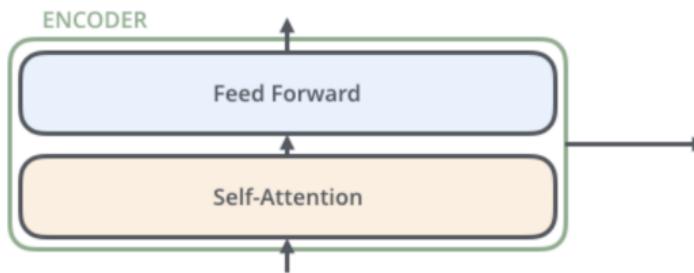
No recursion, instead
stacking encoder and
decoder blocks

- ▶ Originally: 6 layers
- ▶ BERT base: 12 layers
- ▶ BERT large: 24 layers
- ▶ GPT2-XL: 48 layers
- ▶ GPT3: 96 layers



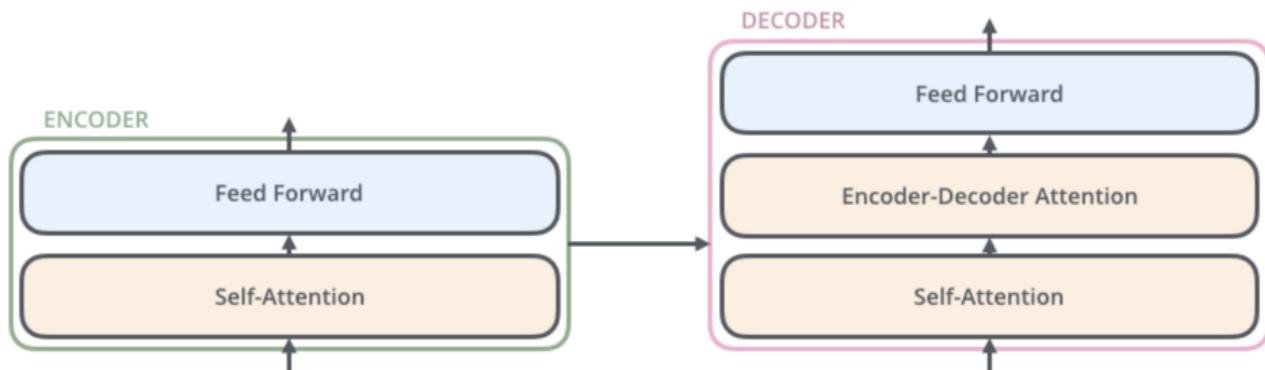
[Alammar, 2018]

The Encoder and Decoder Blocks



[Alammar, 2018]

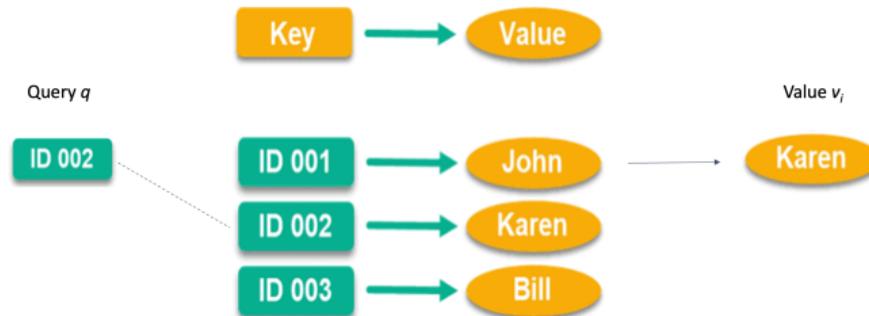
The Encoder Block



[Alammar, 2018]

Attention Preliminaries

Mimics the retrieval of a value v_i for a query q based on a key k_i in a database, but in a probabilistic fashion





Dot-Product Attention

- ▶ Queries, keys and values are vectors
- ▶ Output is a **weighted sum** of the values
- ▶ Weights are computed as the **scaled dot-product** (similarity) between the query and the keys

$$\text{Attention}(q, K, V) = \sum_i \text{Similarity}(q, k_i) \cdot v_i = \sum_i \frac{e^{q \cdot k_i / \sqrt{d_k}}}{\sum_j e^{q \cdot k_j / \sqrt{d_k}}} v_i$$

Output is a
row-vector

- ▶ Can stack multiple queries into a matrix Q

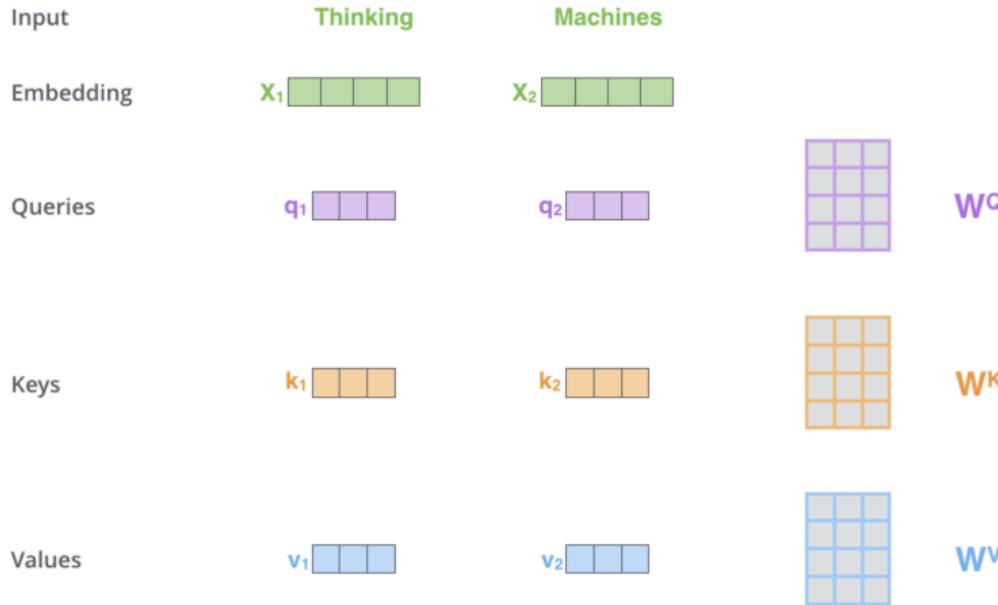
$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V$$

Output is again
a matrix

- ▶ Self-attention: Let the word embeddings be the queries, keys and values, i.e. **let the words select each other**

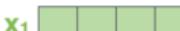
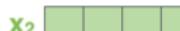
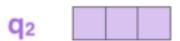
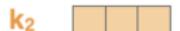


Self-Attention Mechanism



[Alammar, 2018]

Self-Attention Mechanism

Input	Thinking	Machines
Embedding	x_1 	x_2 
Queries	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

[Alammar, 2018]

Self-Attention Mechanism in Matrix Notation

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

A diagram showing the multiplication of a green input matrix \mathbf{X} (3x3) by a purple weight matrix \mathbf{W}^Q (3x3) to produce a purple output matrix \mathbf{Q} (3x3).

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

A diagram showing the multiplication of a green input matrix \mathbf{X} (3x3) by an orange weight matrix \mathbf{W}^K (3x3) to produce an orange output matrix \mathbf{K} (3x3).

$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

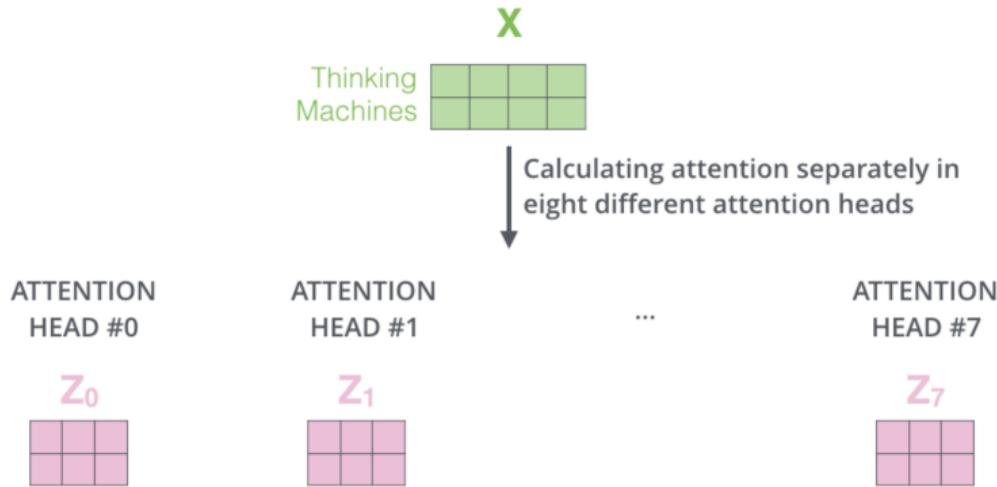
A diagram showing the multiplication of a green input matrix \mathbf{X} (3x3) by a light blue weight matrix \mathbf{W}^V (3x3) to produce a light blue output matrix \mathbf{V} (3x3).

$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} = \mathbf{Z}$$

The final computation of the self-attention mechanism. It shows the product of the query matrix \mathbf{Q} (3x3) and the transpose of the key matrix \mathbf{K}^T (3x3), divided by the square root of the dimension d_k , passed through a softmax function, and multiplied by the value matrix \mathbf{V} (3x3) to produce the output matrix \mathbf{Z} (3x3).

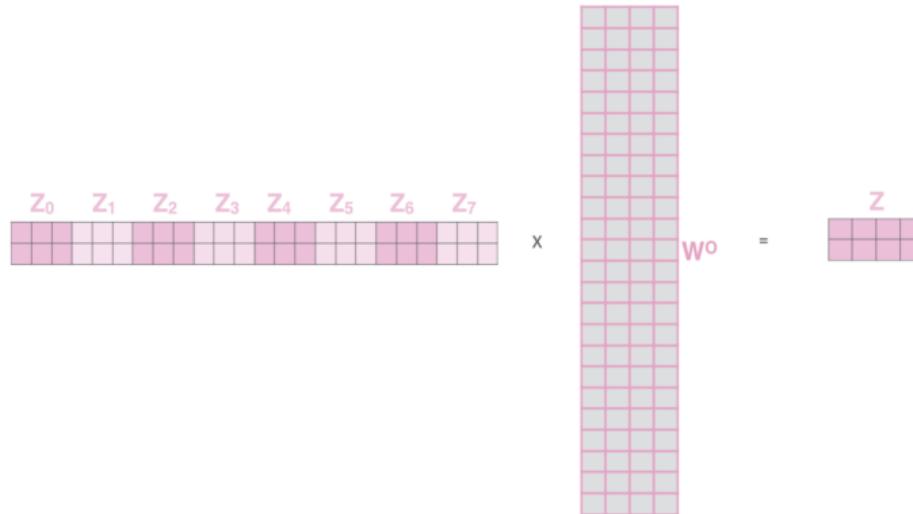
[Alammar, 2018]

Multi-Headed Self-Attention



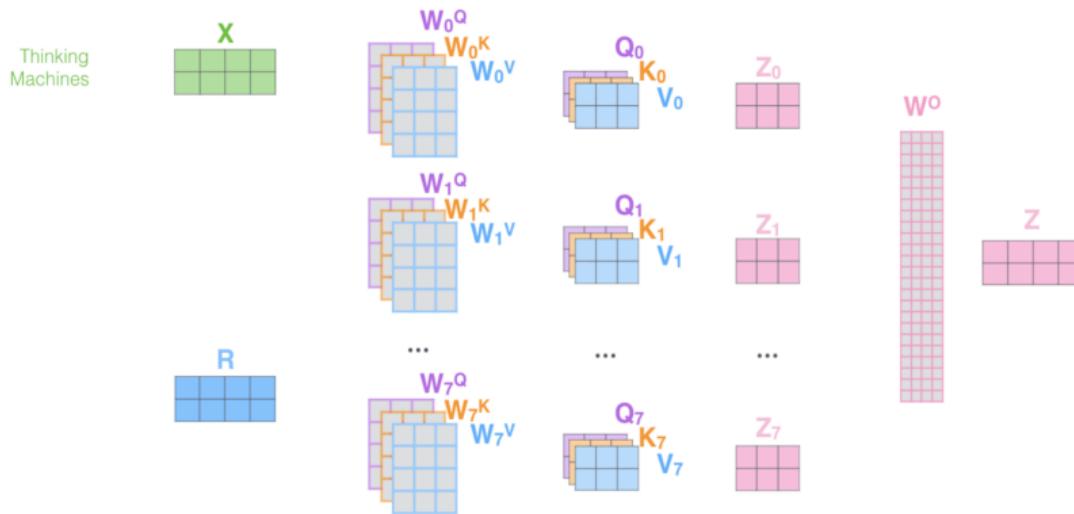
[Alammar, 2018]

Multi-Headed Self-Attention



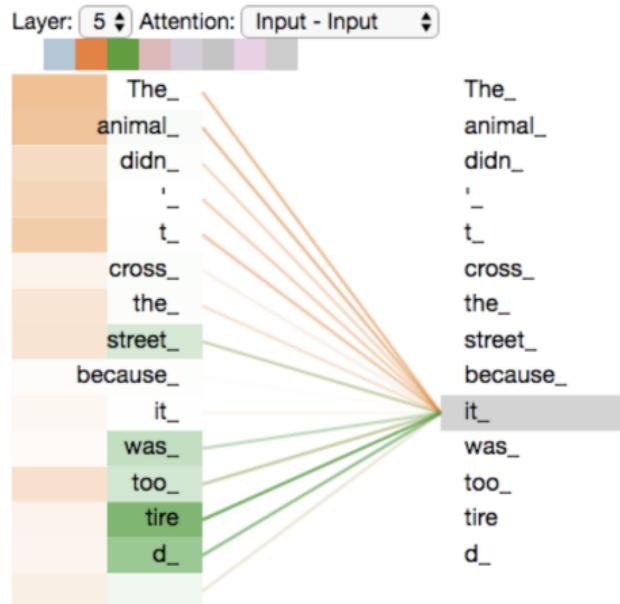
[Alammar, 2018]

Self-Attention: Putting It All Together



[Alammar, 2018]

Attention Visualized

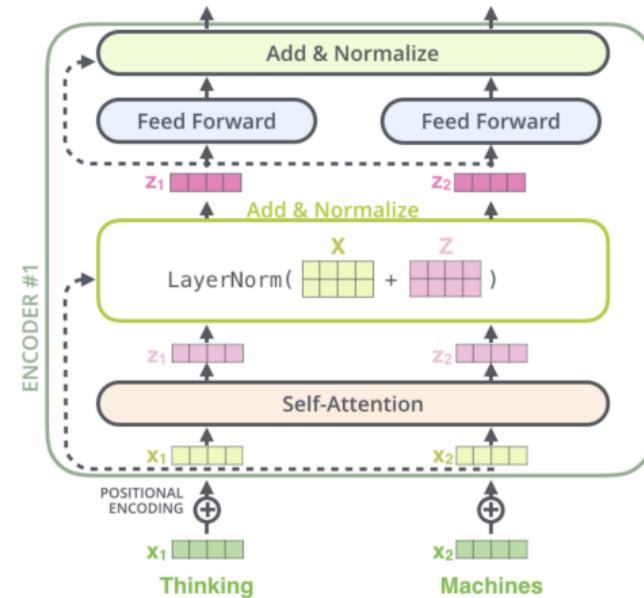


[Alammar, 2018]

The Full Encoder Block

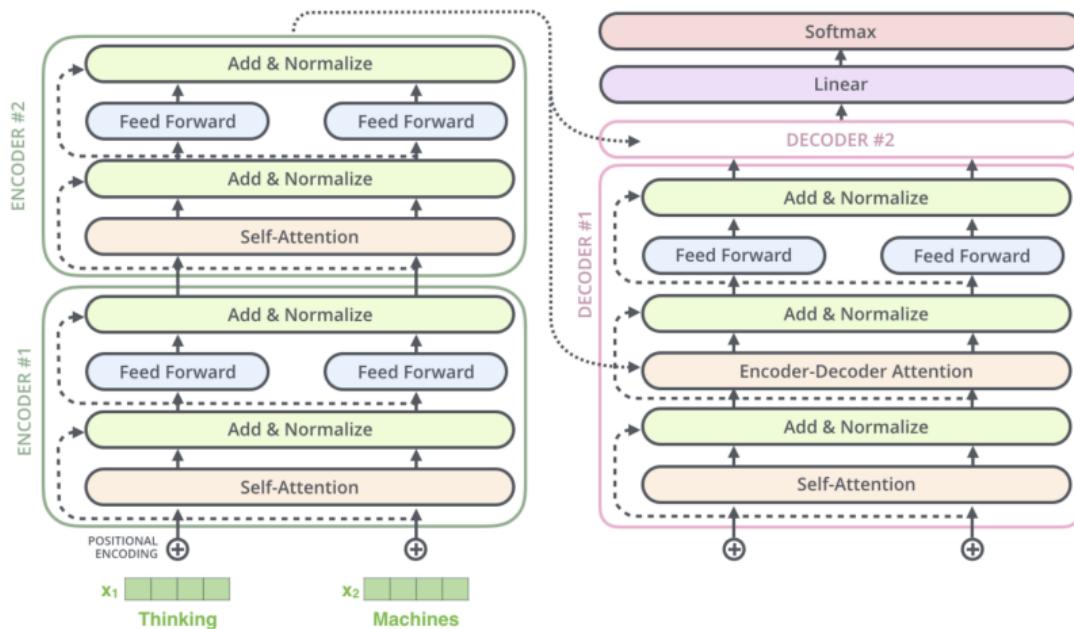
Encoder block consisting of:

- ▶ Multi-headed self-attention
- ▶ Feedforward NN (FC 2 layers)
- ▶ Skip connections
- ▶ Layer normalization - Similar to batch normalization but computed over features (words/tokens) for a single sample



[Alammar, 2018]

Encoder-Decoder Architecture - Small Example



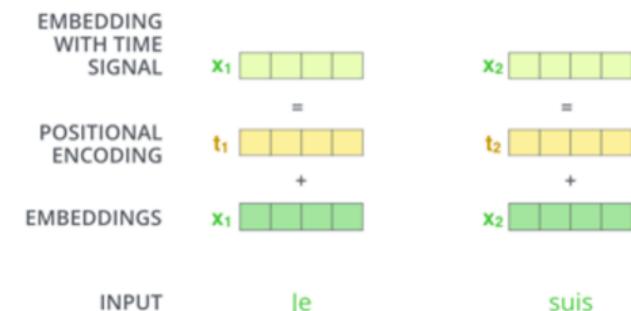
[Alammar, 2018]

Positional Encodings

Encoder block consisting of:

- ▶ Attention mechanism has no locality bias - **no notion of word order**
- ▶ **Add positional encodings** to input embeddings to let model learn relative positioning

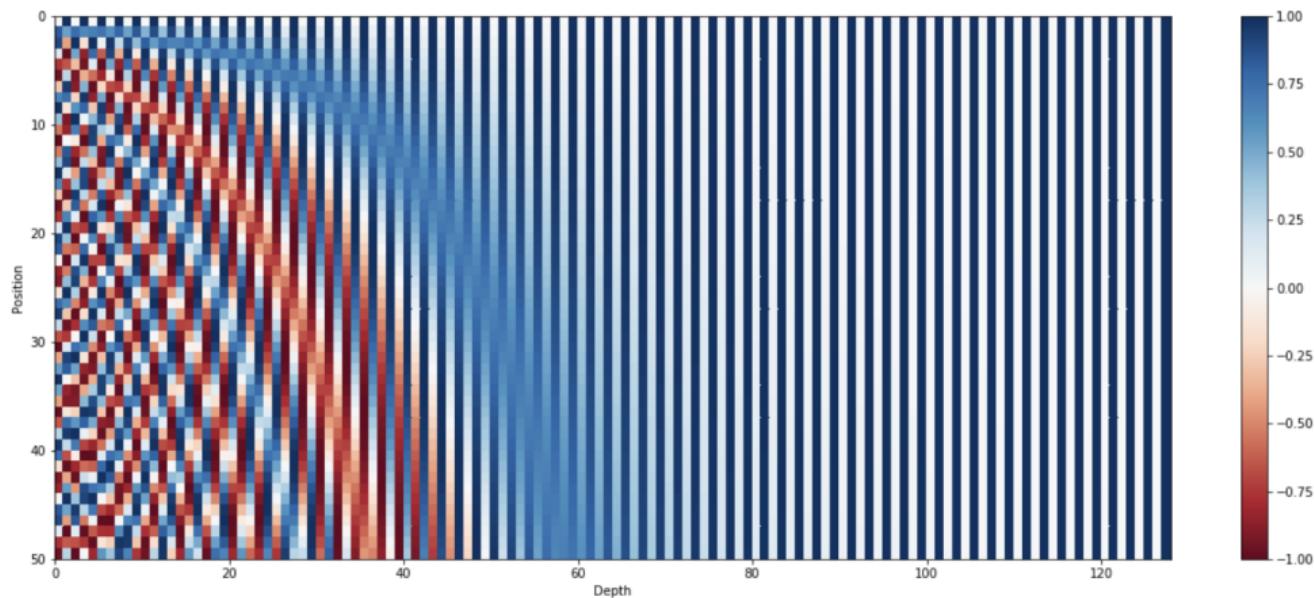
$$\text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$



$$\text{PE}(\text{pos}, 2i + 1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\right)$$

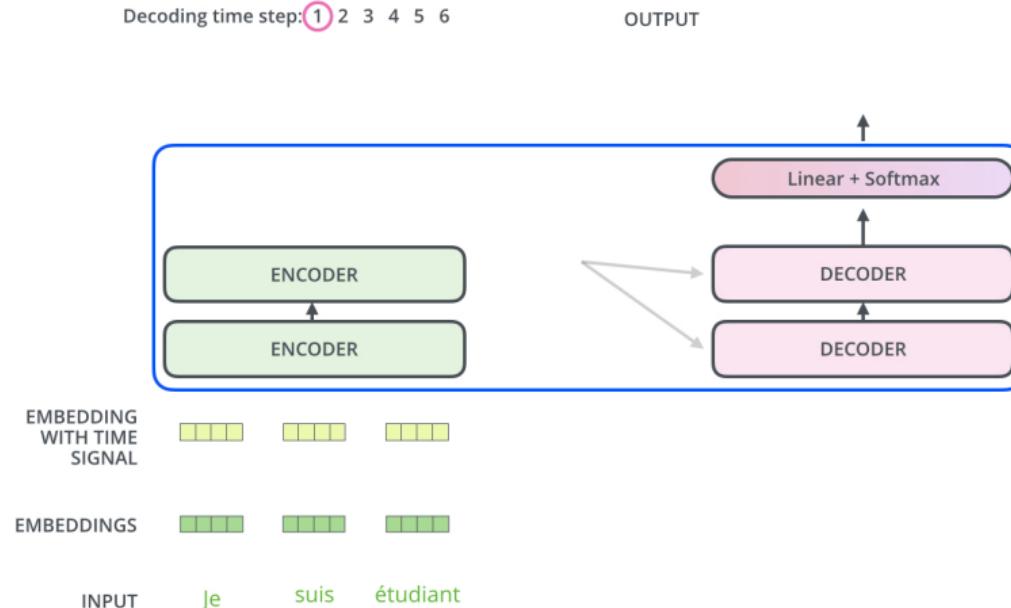
[Alammar, 2018]

Positional Encodings



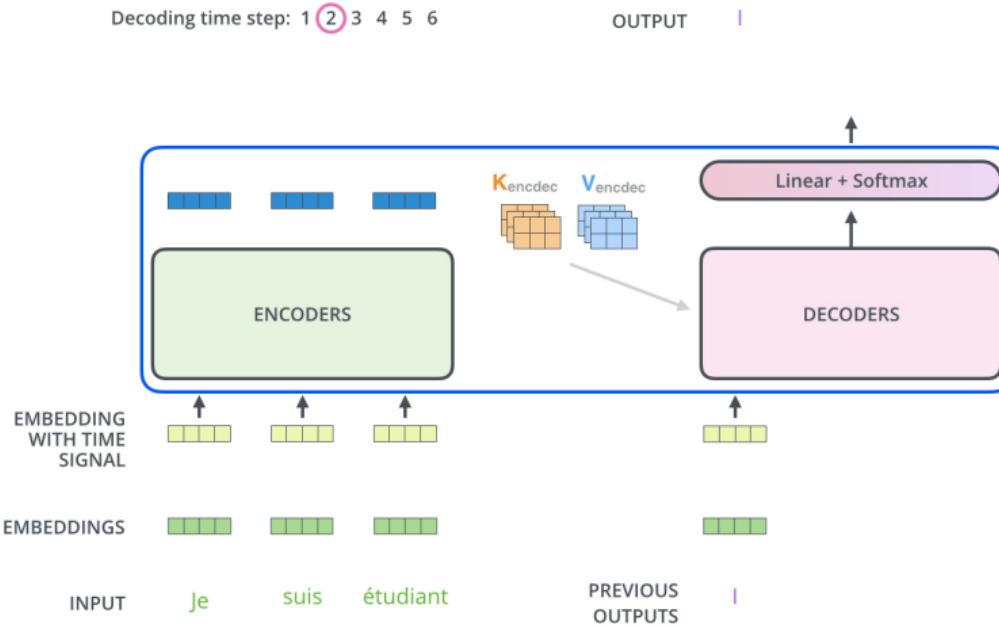
[Kazemnejad, 2019]

Let's start the encoding!



[Alammar, 2018]

Decoding procedure

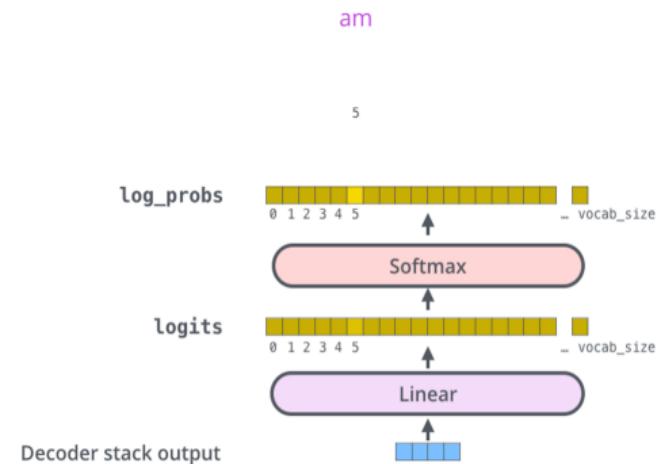


[Alammar, 2018]

Producing the output text

Encoder block consisting of:

- ▶ The output from the decoder is passed through a final fully connected **linear layer** with a **softmax** activation function
- ▶ Produces a probability distribution over the pre-defined vocabulary of output words (tokens)
- ▶ **Greedy decoding** picks the word with the highest probability at each time step



[Alammar, 2018]

Training Objective

Target Model Outputs

Output Vocabulary: a am I thanks student <eos>

position #1	0.0	0.0	1.0	0.0	0.0	0.0
-------------	-----	-----	------------	-----	-----	-----

position #2	0.0	1.0	0.0	0.0	0.0	0.0
-------------	-----	------------	-----	-----	-----	-----

position #3	1.0	0.0	0.0	0.0	0.0	0.0
-------------	------------	-----	-----	-----	-----	-----

position #4	0.0	0.0	0.0	0.0	1.0	0.0
-------------	-----	-----	-----	-----	------------	-----

position #5	0.0	0.0	0.0	0.0	0.0	1.0
-------------	-----	-----	-----	-----	-----	------------

a am I thanks student <eos>



Trained Model Outputs

Output Vocabulary: a am I thanks student <eos>

position #1	0.01	0.02	0.93	0.01	0.03	0.01
-------------	------	------	-------------	------	------	------

position #2	0.01	0.8	0.1	0.05	0.01	0.03
-------------	------	------------	-----	------	------	------

position #3	0.99	0.001	0.001	0.001	0.002	0.001
-------------	-------------	-------	-------	-------	-------	-------

position #4	0.001	0.002	0.001	0.02	0.94	0.01
-------------	-------	-------	-------	------	-------------	------

position #5	0.01	0.01	0.001	0.001	0.001	0.98
-------------	------	------	-------	-------	-------	-------------

a am I thanks student <eos>



[Alammar, 2018]

Complexity Comparison

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

[Vaswani et al., 2017]



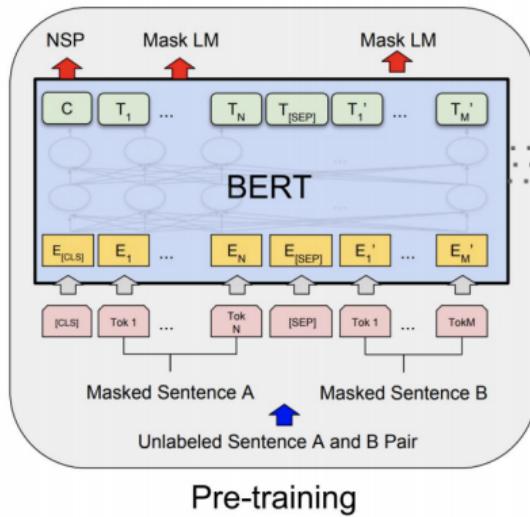
BERT

Bidirectional Encoder Representations from Transformers

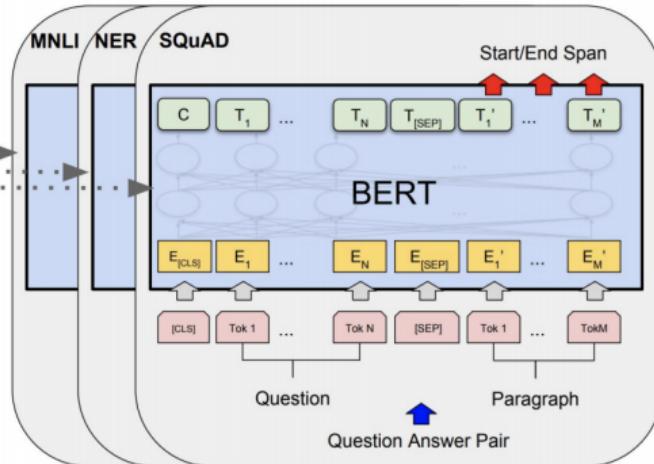
- ▶ Self-supervised pre-training of Transformers encoder for language understanding
- ▶ Fine-tuning for specific downstream task



BERT Training Procedure



Pre-training



Fine-Tuning

[Devlin et al., 2018]



BERT Training Objectives

Masked Language Modelling

the man went to the [MASK] to buy a [MASK] of milk

↑ ↑
store gallon

Next Sentence prediction

Sentence A = The man went to the store.

Sentence B = He bought a gallon of milk.

Label = IsNextSentence

Sentence A = The man went to the store.

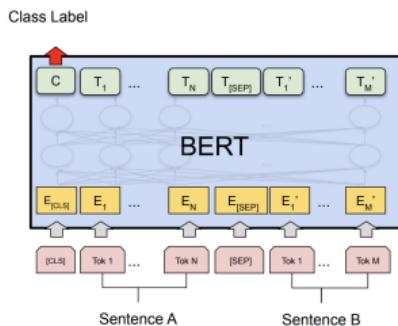
Sentence B = Penguins are flightless.

Label = NotNextSentence

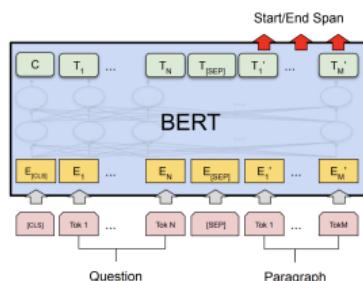
[Devlin et al., 2018]

BERT Fine-Tuning Examples

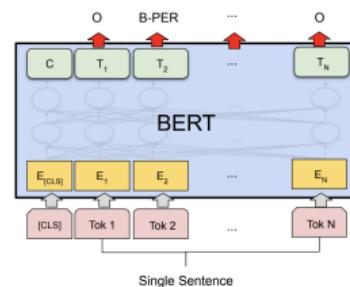
Sentence Classification



Question Answering



Named Entity Recognition



[Devlin et al., 2018]



Exploring the Limits of Transfer Learning (T5)

- ▶ Scaling up **models size** and amount of **training data** helps a lot
- ▶ Best model is 11B (!! parameters
- ▶ Exact **pre-training objective** (MLM, NSP, corruption) doesn't matter too much
- ▶ SuperGLUE benchmark:

Rank	Name	Model	URL	Score	BoolQ	CB	COPA	MultiRC	ReCoRD	RTE	WiC	WSC	AX-b	AX-g
1	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8	89.0	95.8/98.9	100.0	81.8/51.9	91.7/91.3	93.6	80.0	100.0	76.6	99.3/99.7
+	2 T5 Team - Google	T5		89.3	91.2	93.9/96.8	94.8	88.1/63.3	94.1/93.4	92.5	76.9	93.8	65.6	92.7/91.9
+	3 Huawei Noah's Ark Lab	NEZHA-Plus		86.7	87.8	94.4/96.0	93.6	84.6/55.1	90.1/89.6	89.1	74.6	93.2	58.0	87.1/74.4
+	4 Alibaba PAI&ICBU	PAI Albert		86.1	88.1	92.4/96.4	91.8	84.6/54.7	89.0/88.3	88.8	74.1	93.2	75.6	98.3/99.2
+	5 Tencent Jarvis Lab	RoBERTa (ensemble)		85.9	88.2	92.5/95.6	90.8	84.4/53.4	91.5/91.0	87.9	74.1	91.8	57.6	89.3/75.6
6	Zhuiyi Technology	RoBERTa-mtl-adv		85.7	87.1	92.4/95.6	91.2	85.1/54.3	91.7/91.3	88.1	72.1	91.8	58.5	91.0/78.1
7	Facebook AI	RoBERTa		84.6	87.1	90.5/95.2	90.6	84.4/52.5	90.6/90.0	88.2	69.9	89.0	57.9	91.0/78.1

[Raffel et al., 2019]



Practical Examples



BERT in low-latency production settings

GOOGLE \ TECH \ ARTIFICIAL INTELLIGENCE

Google is improving 10 percent of searches by understanding language context

Say hello to BERT

By Dieter Bohn | @backlon | Oct 25, 2019, 3:01am EDT

Bing says it has been applying BERT since April

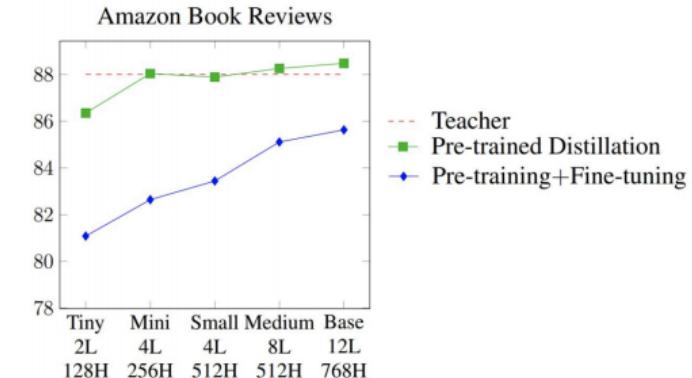
The natural language processing capabilities are now applied to all Bing queries globally.

[George Nguyen](#) on November 19, 2019 at 1:38 pm

[Devlin, 2020]

Distillation

- ▶ Modern pre-trained language models are **huge** and very **computationally expensive**
- ▶ How are these companies applying them to low-latency applications?
- ▶ Distillation!
 - Train SOTA **teacher model** (pre-training + fine-tuning)
 - Train smaller **student model** that **mimics** the teacher's output on a large dataset on unlabeled data
- ▶ Why does it work so well?



[Devlin, 2020] [Turc, 2020]



Transformers in TensorFlow using HuggingFace 😊



HUGGING FACE

[Back to home](#)

All Models and checkpoints

Also check out our list of [Community contributors](#) 🌟 and [Organizations](#) 🌐.

Search models... Tags: All ▾ Sort: Most downloads ▾

- [bert-base-uncased](#) ★
- [deepset/bert-large-uncased-whole-word-masking-squad2](#)
- [distilbert-base-uncased](#) ★
- [dccuchile/bert-base-spanish-wwm-cased](#) ★
- [microsoft/xprophetnet-large-wiki100-cased-xglue-ntg](#) ★
- [deepset/roberta-base-squad2](#) ★
- [jplu/tf-xlm-roberta-base](#) ★
- [cl-tohoku/bert-base-japanese-whole-word-masking](#)
- [distilroberta-base](#) ★
- [bert-base-cased](#) ★
- [xlm-roberta-base](#) ★

- ▶ The [HuggingFace Library](#) contains a majority of the recent pre-trained State-of-the-art NLP models, as well as over 4 000 community uploaded models
- ▶ Works with both [TensorFlow](#) and [PyTorch](#)



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Transformers in TensorFlow using HuggingFace 😊

```
from transformers import BertTokenizerFast, TFBertForSequenceClassification
from datasets import load_dataset
import tensorflow as tf

dataset = load_dataset("imdb").shuffle()
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

train_encodings = tokenizer(dataset['train']['text'], truncation=True, padding=True)
train_dataset = tf.data.Dataset.from_tensor_slices((dict(train_encodings), dataset['train']['label']))
val_dataset = ... // Analogously

optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
model.compile(optimizer=optimizer, loss=model.compute_loss)
model.fit(train_dataset.batch(16), epochs=3, batch_size=16)

model.evaluate(val_dataset.batch(16), verbose=0)
```



Wrap Up

Summary

- ▶ Transformers have blown other architectures out of the water for NLP
- ▶ Get rid of recurrence and rely on **self-attention**
- ▶ NLP pre-training using **Masked Language Modelling**
- ▶ Most recent improvements using **larger models** and **more data**
- ▶ **Distillation** can make model serving and inference more tractable





Thanks!
Questions?

Francisco J. Peña
Postdoctoral Researcher, KTH
frape@kth.se