Transformers and Attention

ID2223 Scalable Machine Learning and Deep Learning

Francisco J. Peña
Postdoctoral Researcher, KTH
frape@kth.se
2021-12-02

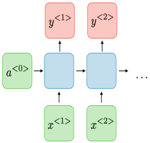
Roadmap

01

Contextualized Embeddings

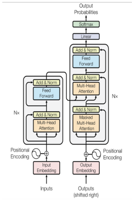
02

From RNNs to Transformers



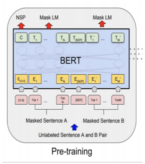
03

Transformers Step-by-Step



04

BERT



05

Distillation and Practical Example

Acknowledgements

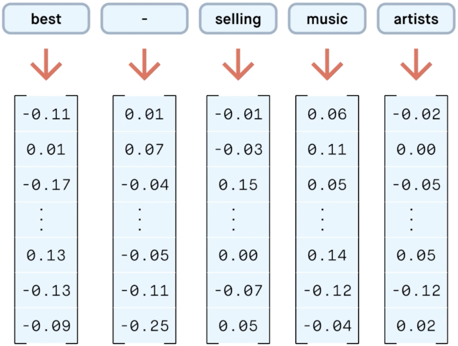
Material based on:

- 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

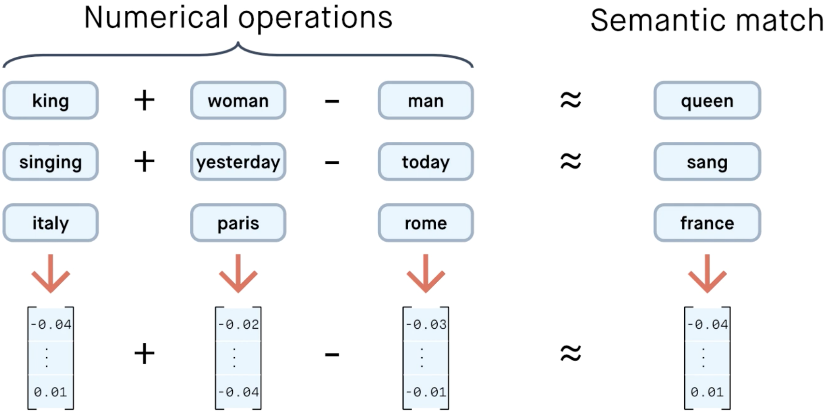
Background to Natural Language Processing (NLP)

- 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]



[Peltarion, 2020]

Word Embeddings



[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are context-free

[Peltarion, 2020]

Word Embeddings

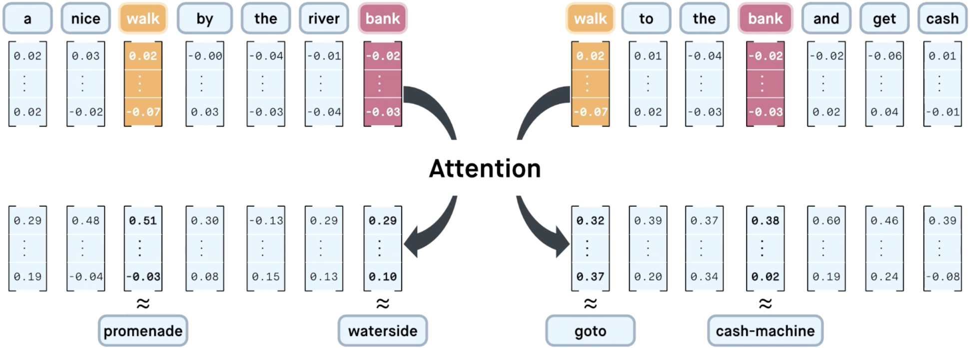
Problem: Word embeddings are context-free

[Peltarion, 2020]

Word Embeddings

Problem: Word embeddings are context-free

Solution: Create contextualized representation



[Peltarion, 2020]

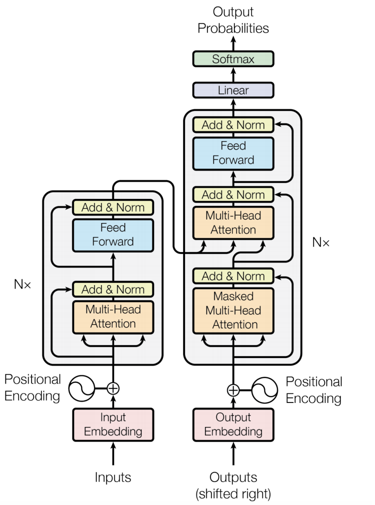
From RNNs to Transformers

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]

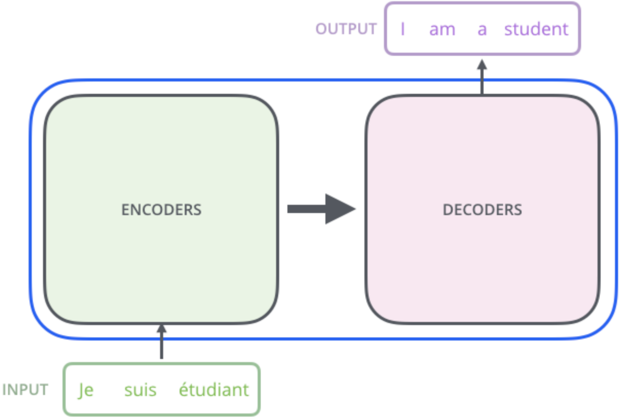
- Sequence-to-sequence model for Machine Translation
- Encoder-decoder architecture
- Multi-headed self-attention
  - Models context and no locality bias



[Vaswani et al., 2017]

Transformers Step-by-Step

Understanding the Transformer: Step-by-Step

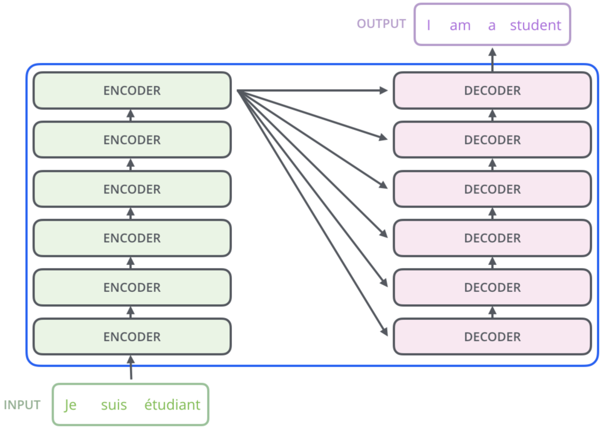


[Alammar, 2018]

Understanding the Transformer: Step-by-Step

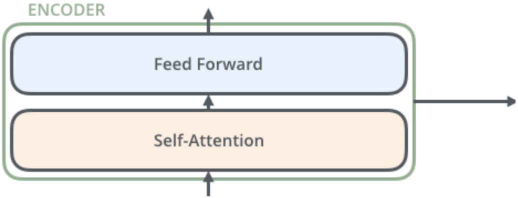
No recursion, instead stacking encoder and decoder blocks

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



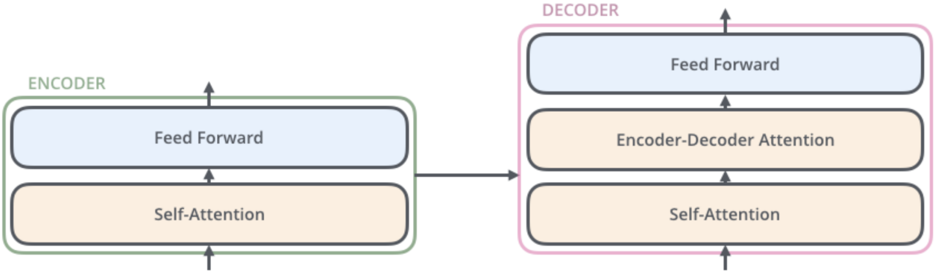
[Alammar, 2018]

The Encoder and Decoder Blocks



[Alammar, 2018]

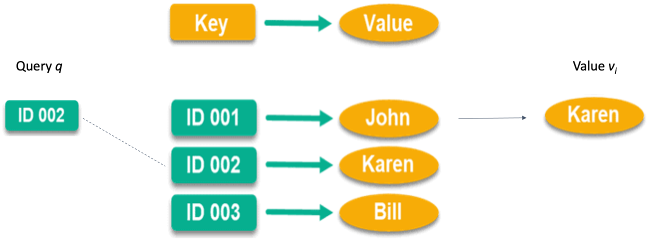
The Encoder Block



[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 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

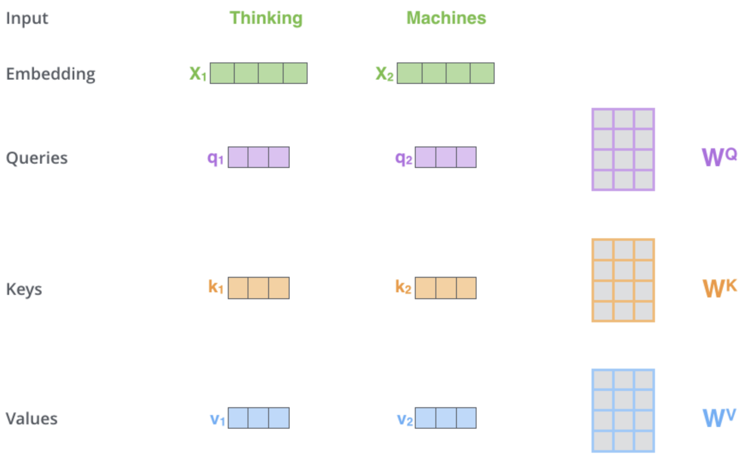
- Can stack multiple queries into a matrix Q

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

Output is again a matrix

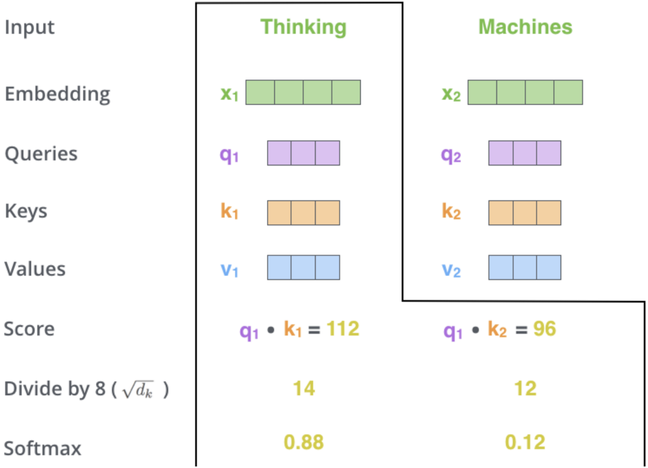
- Self-attention: Let the word embeddings be the queries, keys and values, i.e. let the words select each other

Self-Attention Mechanism



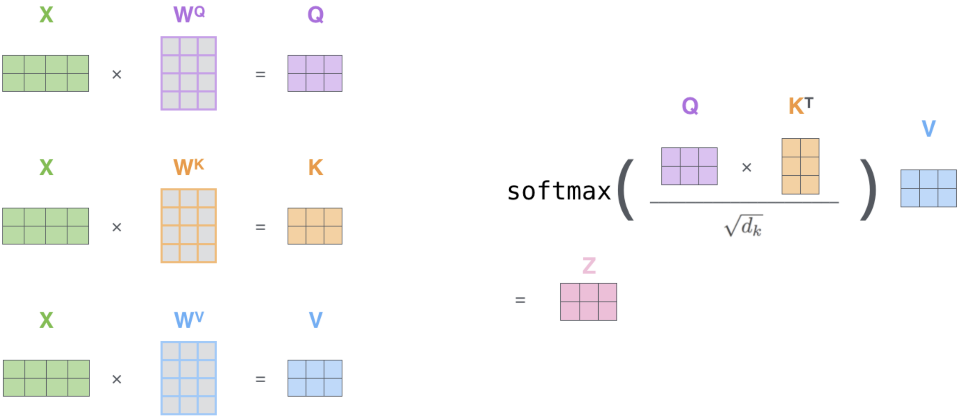
[Alammar, 2018]

Self-Attention Mechanism



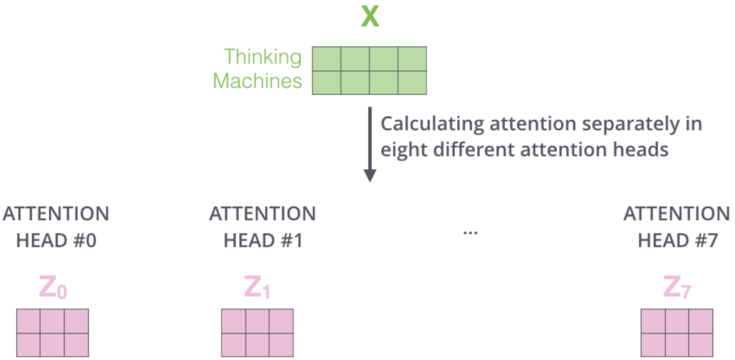
[Alammar, 2018]

Self-Attention Mechanism in Matrix Notation



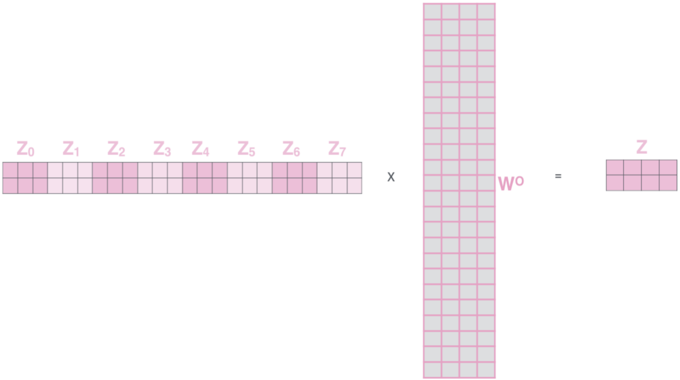
[Alammar, 2018]

Multi-Headed Self-Attention



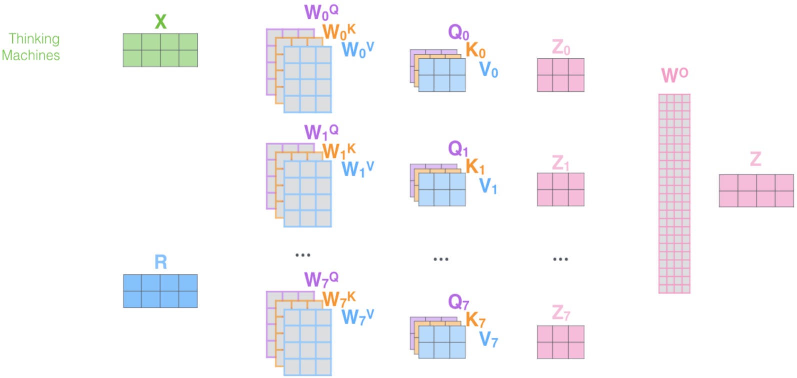
[Alammar, 2018]

Multi-Headed Self-Attention



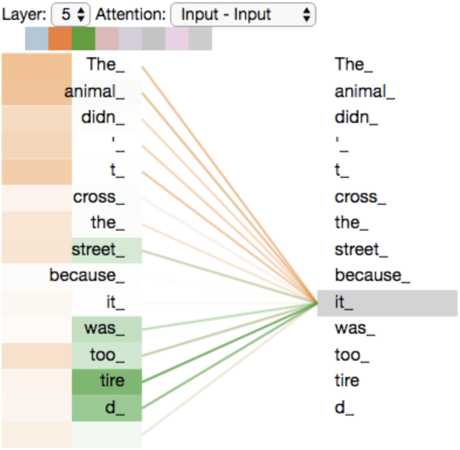
[Alammar, 2018]

Self-Attention: Putting It All Together



[Alammar, 2018]

Attention Visualized

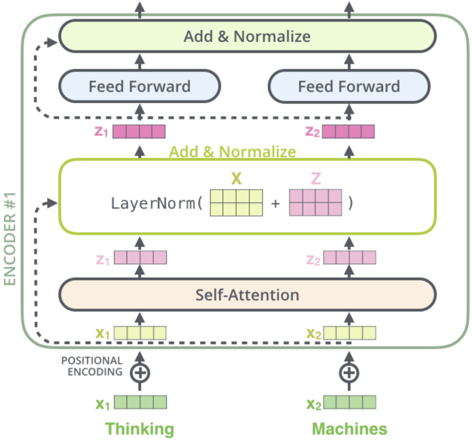


[Alammar, 2018]

The Full Encoder Block

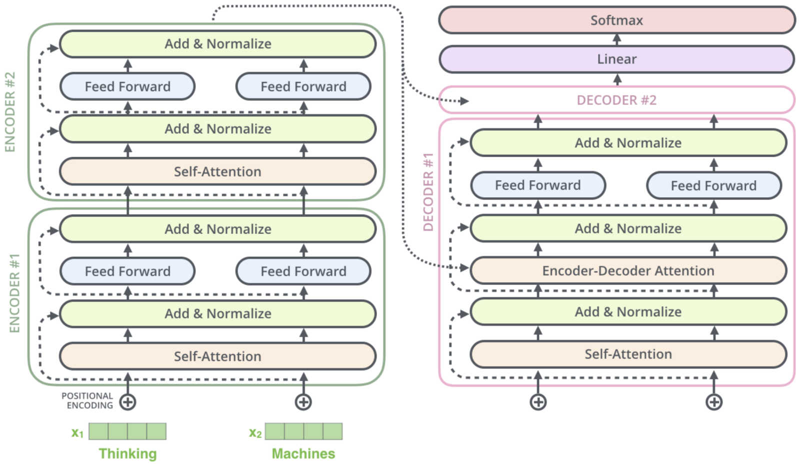
Encoder block consisting of:

- 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]

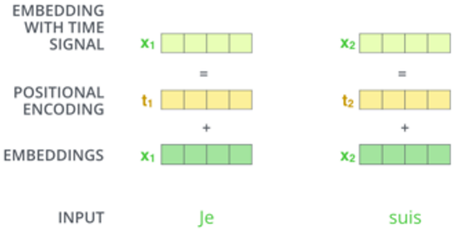
Positional Encodings

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

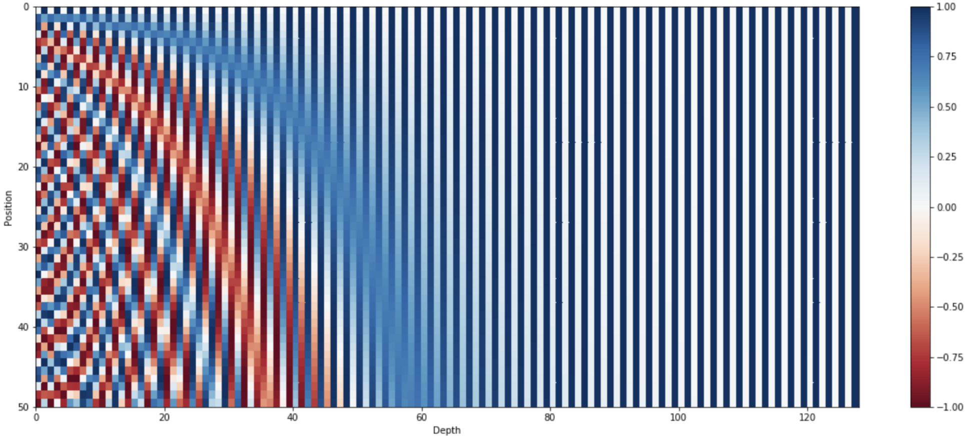
$$\mathrm{PE}(\mathrm{pos}, 2i) = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{model}}}}\right)$$

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



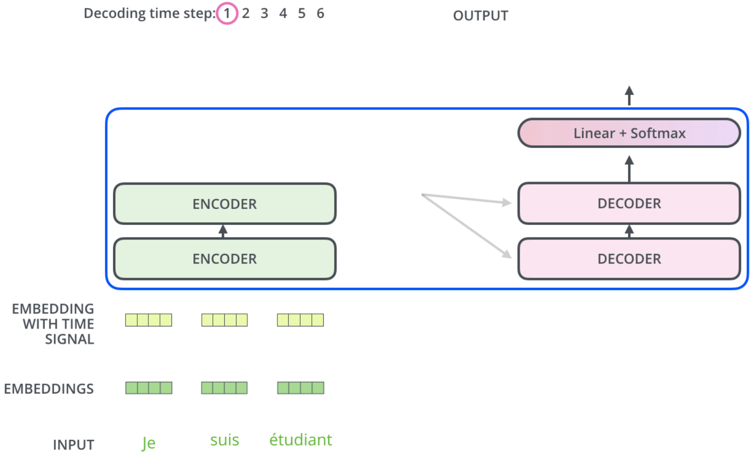
[Alammar, 2018]

Positional Encodings



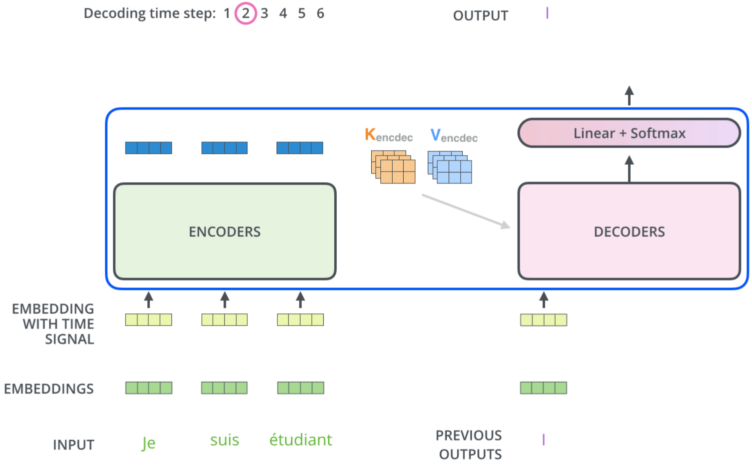
[Kazemnejad, 2019]

Let's start the encoding!



[Alammar, 2018]

Decoding procedure

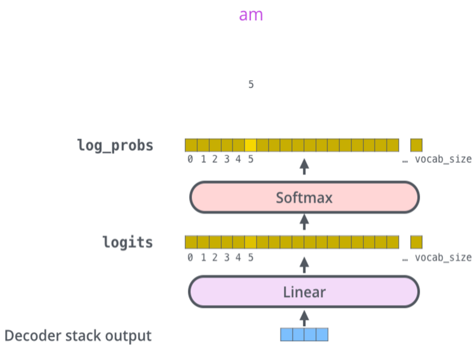


[Alammar, 2018]

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]

Results

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

[Vaswani et al., 2017]

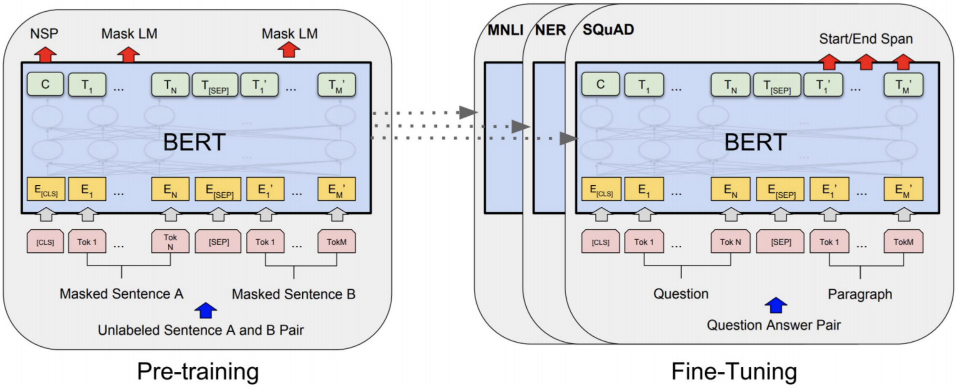
BERT

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

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

BERT Training Procedure



[Devlin et al., 2018]

BERT Training Objectives

Masked Language Modelling

```
                        store         gallon
                          ↑              ↑
the man went to the [MASK] to buy a [MASK] of milk
```

Next Sentence prediction

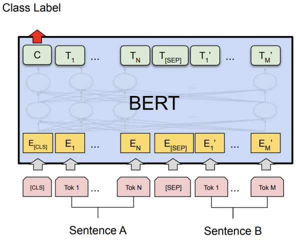
Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

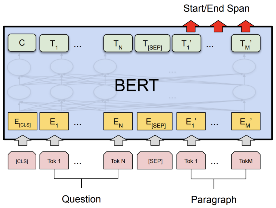
[Devlin et al., 2018]

BERT Fine-Tuning Examples

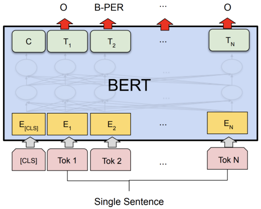
Sentence Classification



Question Answering



Named Entity Recognition



[Devlin et al., 2018]

How good are transformers?

- Scaling up models size and amount of training data helps a lot
- Best model is 10B (!!) parameters
- Two models have already surpassed human performance!!!
- 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-g	AX-b
	1	ERNIE Team - Baidu	ERNIE 3.0		90.6	91.0	98.6/99.2	97.4	88.6/63.2	94.7/94.2	92.6	77.4	97.3	92.7/94.7	68.6
+	2	Zirui Wang	T5 + UDG, Single Model (Google Brain)		90.4	91.4	95.8/97.6	98.0	88.3/63.0	94.2/93.5	93.0	77.9	96.6	92.7/91.9	69.1
+	3	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.3	90.4	95.7/97.6	98.4	88.2/63.7	94.5/94.1	93.2	77.5	95.9	93.3/93.8	66.7
	4	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	99.3/99.7	76.6
+	5	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	92.7/91.9	65.6
+	6	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	87.1/74.4	58.0

[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

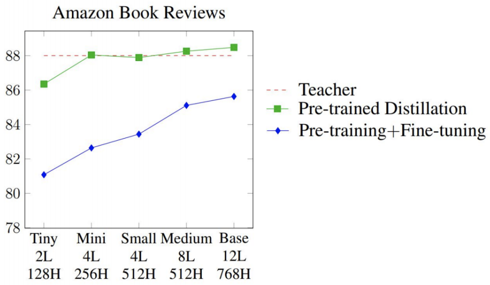
The natural language processing capabilities are now applied to all Bing queries globally.

George Nguyen on November 19, 2019 at 1:38 pm

[Devlin, 2020]

Distillation

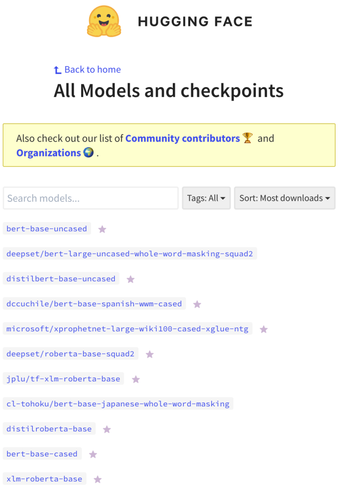
- Modern pre-trained language models are huge and very computationally expensive
- How are these companies applying them to low-latency applications?
- Distillation!
  - Train SOTA teacher model (pre-training + fine-tuning)
  - Train smaller student model that mimics the teacher's output on a large dataset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



[Devlin, 2020] [Turc, 2020]

Transformers in TensorFlow using HuggingFace 🤗

- 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