EDAP30
Advanced Applied Machine Learning
Lecture 7: Encoders-Decoders and Transformers

Pierre Nugues

Pierre.Nugues@cs.lth.se

April 22, 2024

Machine Translation

Process of translating automatically a text from a source language into a target language
Started after the 2nd world war to translate documents from Russian to English
Early working systems from French to English in Canada
Renewed huge interest with the advent of the web
Google claims it has more than 500m users daily worldwide, with 103 languages.
Massive progress permitted by the neural networks

Corpora for Machine Translation

Initial ideas in machine translation: use bilingual dictionaries and formalize grammatical rules to transfer them from a source language to a target language.
Statistical machine translation:

1. Use very large bilingual corpora;
2. Align the sentences or phrases, and
3. Given a sentence in the source language, find the matching sentence in the target language.

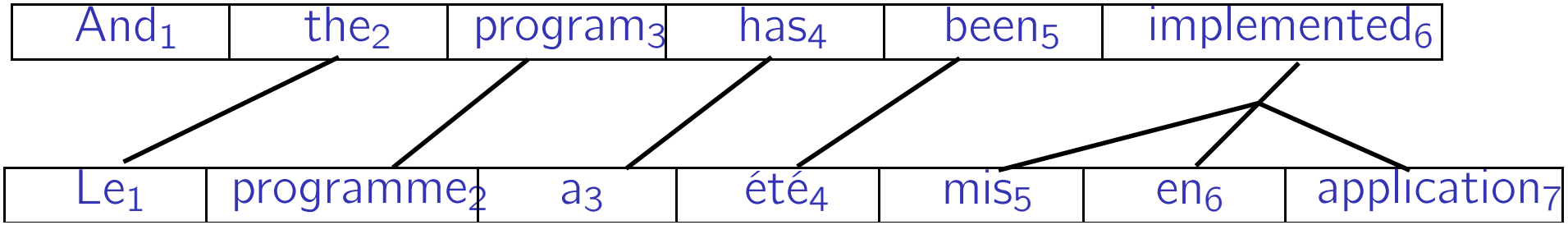
Pioneered at IBM on French and English with Bayesian statistics.
Neural nets are now dominant

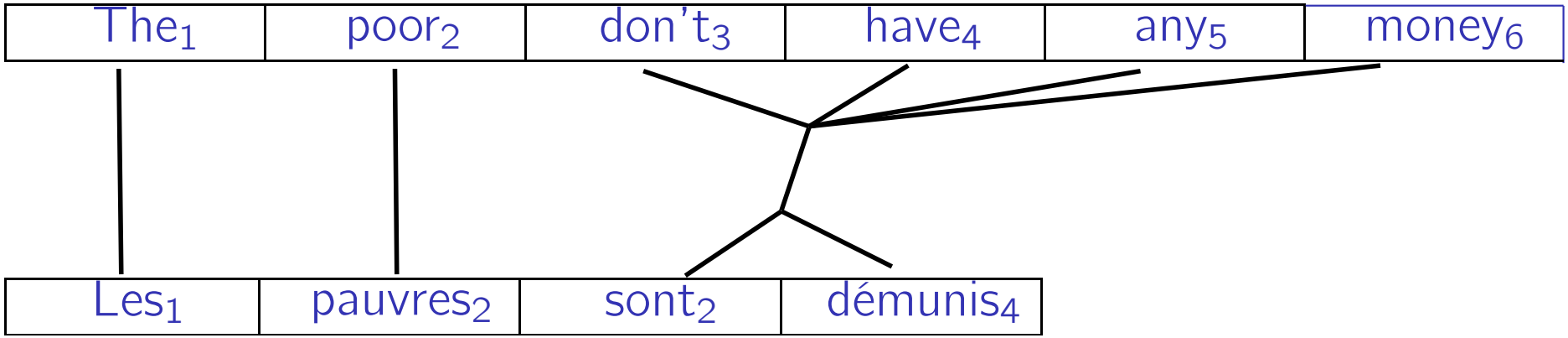
Parallel Corpora (Swiss Federal Law)

German	French	Italian
Art. 35 Milchtransport	**Art. 35 Transport du lait**	**Art. 35 Trasporto del latte**
1 Die Milch ist schonend und hygienisch in den Verarbeitungsbetrieb zu transportieren. Das Transportfahrzeug ist stets sauber zu halten. Zusammen mit der Milch dürfen keine Tiere und milchfremde Gegenstände transportiert werden, welche die Qualität der Milch beeinträchtigen können.	1 Le lait doit être transporté jusqu'à l'entreprise de transformation avec ménagement et conformément aux normes d'hygiène. Le véhicule de transport doit être toujours propre. Il ne doit transporter avec le lait aucun animal ou objet susceptible d'en altérer la qualité.	1 Il latte va trasportato verso l'azienda di trasformazione in modo accurato e igienico. Il veicolo adibito al trasporto va mantenuto pulito. Con il latte non possono essere trasportati animali e oggetti estranei, che potrebbero pregiudicarne la qualità.
2 Wird Milch ausserhalb	2 Si le lait destiné à	2 Se viene collocato fuori

Alignment (Brown et al. 1993)

Canadian Hansard





Translations with RNNs

RNN can easily map sequences to sequences, where we have two lists: one for the source and the other for the target

y	Le	serveur	apporta	le	plat

x	The	waiter	brought	the	meal

The **x** and **y** vectors must have the same length.
In our case, *a apporté* is more frequent than *apporta*, but it breaks the alignment, as well as in many other examples

Translation with RNN

To solve the alignment problem, Sutskever al al. (2014) proposed (quoted from their paper, `https://arxiv.org/abs/1409.3215`):

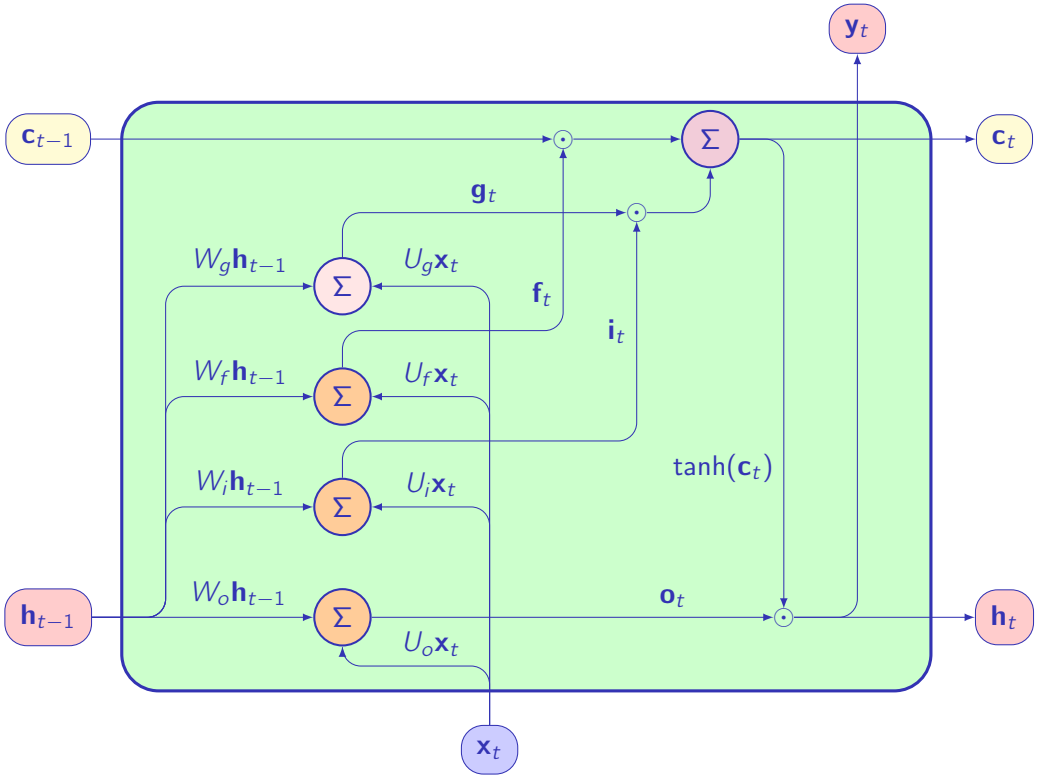
1. The simplest strategy for general sequence learning is to map the input sequence to a fixed-sized vector using one RNN, and then to map the vector to the target sequence with another RNN [...]
2. it would be difficult to train the RNNs due to the resulting long term dependencies [...]. However, the Long Short-Term Memory (LSTM) is known to learn problems with long range temporal dependencies.

Using the Hidden States

To solve the alignment problem, Sutskever al al. (2014) proposed (quoted from their paper, `https://arxiv.org/abs/1409.3215`):

1. LSTM estimate[s] the conditional probability $p(y_1, ..., y_{T'}|x_1, ..., x_T)$, where $(x_1, ..., x_T)$ is an input sequence and $y_1, ..., y_{T'}$ is its corresponding output sequence whose length T' may differ from T.
2. The LSTM computes this conditional probability by:
   1. First obtaining the fixed-dimensional representation v of the input sequence $(x1, ..., xT)$ given by the last hidden state of the LSTM, (**encoder**) and then
   2. computing the probability of $y_1, ..., y_{T'}$ with a standard LSTM-LM formulation whose initial hidden state is set to the representation v of $x_1, ..., x_T$ (**decoder**)

The LSTM Architecture



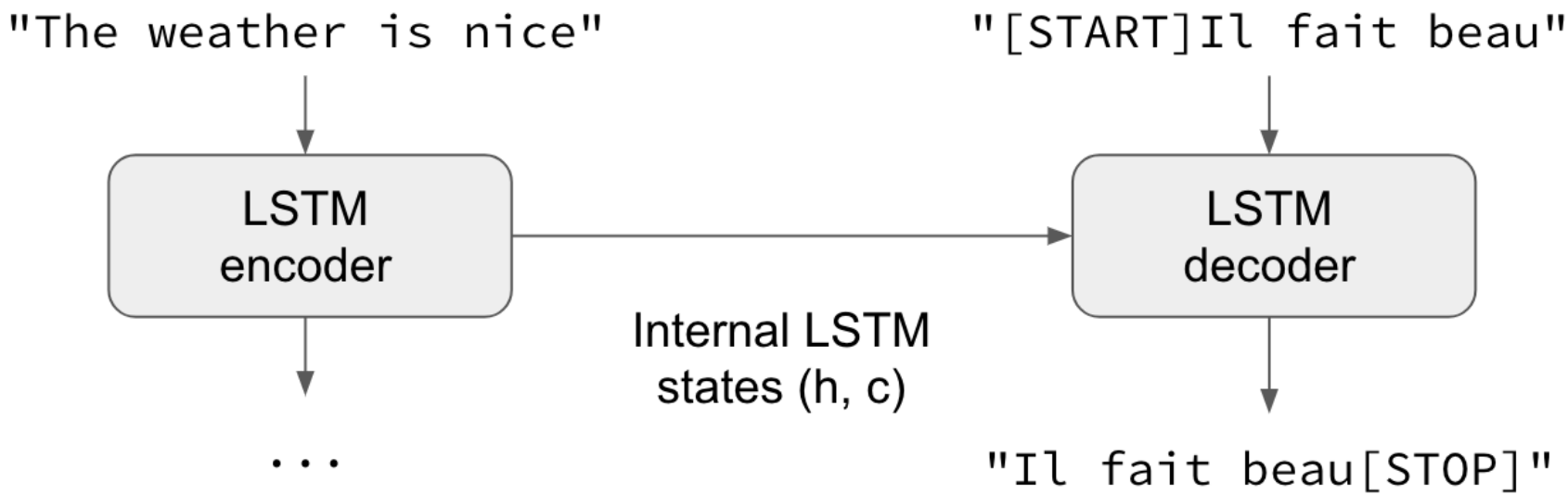
An LSTM unit showing the data flow, where $\mathbf{g}_t$ is the unit input, $\mathbf{i}_t$, the input gate, $\mathbf{f}_t$, the forget gate, and $\mathbf{o}_t$, the output gate. The activation functions have been omitted

Sequence-to-Sequence Translation

We follow and reuse: `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in.html` and `https://keras.io/examples/nlp/lstm_seq2seq/` from Chollet.

1. We start with input sequences from a language (e.g. English sentences) and corresponding target sequences from another language (e.g. French sentences).
2. An encoder LSTM turns input sequences to 2 state vectors (we keep the last LSTM state and discard the outputs).
3. A decoder LSTM is trained to turn the target sequences into the same sequence but offset by one timestep in the future. This training process is called "teacher forcing" in this context.
4. It uses the state vectors from the encoder as initial state. Effectively, the decoder learns to generate `targets[t+1...]` given `targets[...t]`, conditioned on the input sequence.

Sequence-to-Sequence Translation



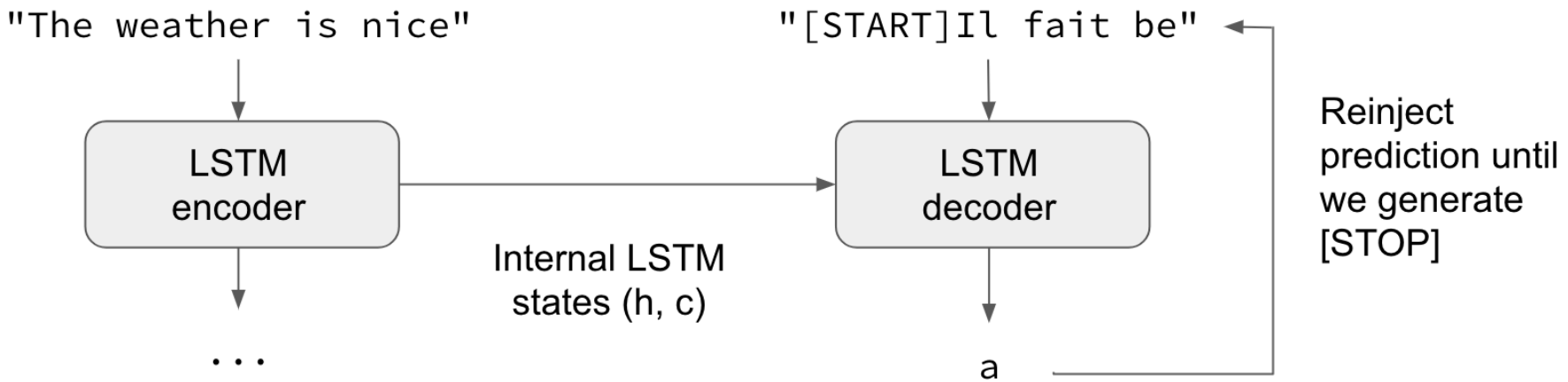
From https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in.html

Inference

Following Chollet, in inference mode, to decode unknown input sequences, we:

- Encode the input sequence into state vectors
- Start with a target sequence of size 1 (just the start-of-sequence character)
- Feed the state vectors and 1-char target sequence to the decoder to produce predictions for the next character
- Sample the next character using these predictions (we simply use argmax).
- Append the sampled character to the target sequence
- Repeat until we generate the end-of-sequence character or we hit the character limit.

Sequence-to-Sequence Translation



From https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in.html

Improving the Architecture: Encoder-Decoder

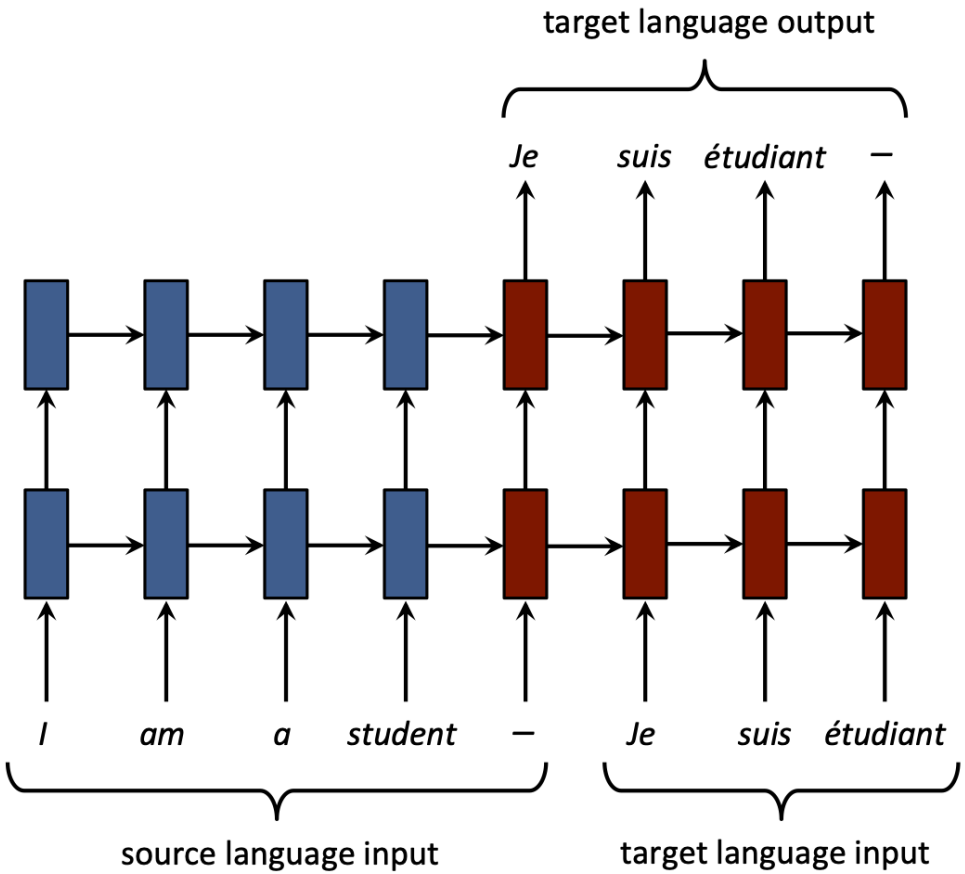


Figure 1: A simplified diagram of NMT.

From: Compression of Neural Machine Translation Models via Pruning by Abigail See, Minh-Thang Luong, and Christopher D. Manning

Improving the Architecture: Adding Attention

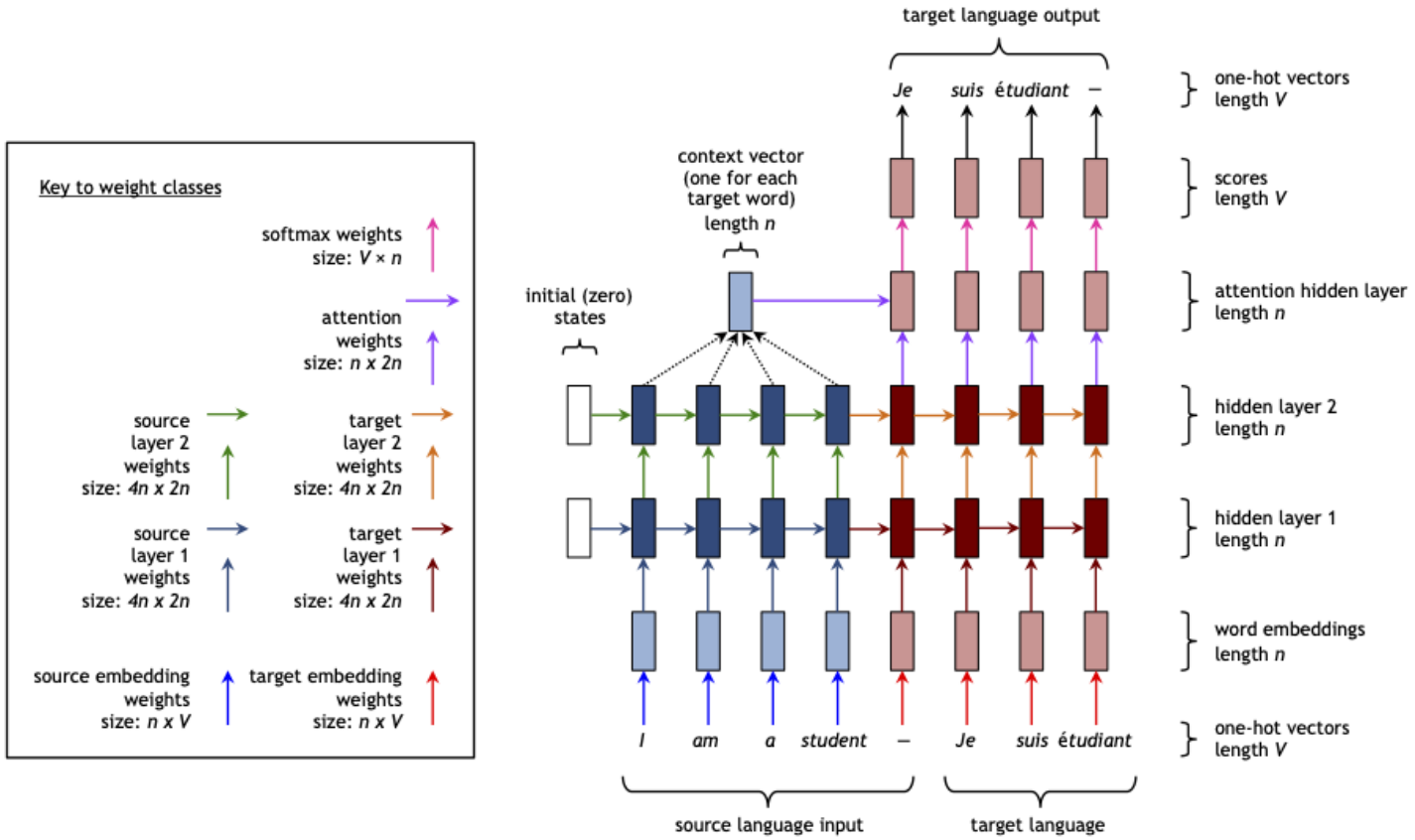


Figure 2: NMT architecture. This example has two layers, but our system has four. The different weight classes are indicated by arrows of different color (the black arrows in the top right represent simply choosing the highest-scoring word, and thus require no parameters). Best viewed in color.

From: Compression of Neural Machine Translation Models via Pruning by Abigail See, Minh-Thang Luong, and Christopher D. Manning

Transformers

After feedforward and recurrent networks, transformers are a third form of networks (in fact a kind of feedforward):

- An architecture proposed in 2018 based on the concept of **attention**
- Consists of two smart pipelines of matrices arranged as an encoder and a decoder
- In this lecture:
  1. Attention
  2. Encoders (BERT)
  3. Decoders (GPT)
- Transformers can learn complex lexical relations

Using Transformers

Goals of transformers:

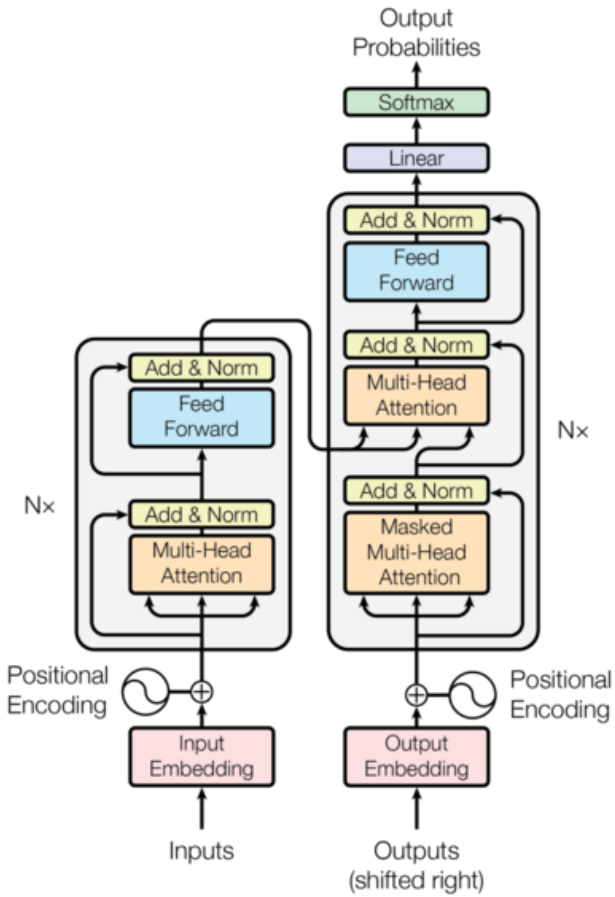
- Encapsulate a massive amount of knowledge.
- In consequence trained on very large corpora
- Sometimes marketed as the ImageNet moment (See `https://ruder.io/nlp-imagenet/`)

Transformers in practice:

- Large companies train transformers on colossal corpora, the pretrained models, requiring huge computing resources (https://arxiv.org/pdf/1906.02243.pdf)
- Mere users:
  - Reuse the models in applications
  - Fine-tune some parameters in the downstream layers

Transformer Architecture

- Reference paper: *Attention Is All You Need* by Vaswani et al. (2017)
  Link: `https://arxiv.org/pdf/1706.03762.pdf`
- Architecture consisting of two parts: encoder (left) and decoder (right)
- Complete implementations in PyTorch:
  1. `http://nlp.seas.harvard.edu/annotated-transformer/`
  2. `https://pytorch.org/tutorials/beginner/transformer_tutorial.html`
  3. `https://github.com/SamLynnEvans/Transformer`
  4. `https://github.com/hyunwoongko/transformer`
  5. `https://github.com/gordicaleksa/pytorch-original-transformer`



Contextual Embeddings

Embeddings we have seen so far do not take the context into account
Attention is a way to make them aware of the context.
Consider the sentence:

> *I must go back to my ship and to my crew*
> *Odyssey, book I*

url: `http://classics.mit.edu/Homer/odyssey.1.i.html`
The word *ship* can be a verb or a noun with different meanings, but has only one GloVe embedding vector
Compare:

> *We process and ship your order in the most cost-efficient way possible*

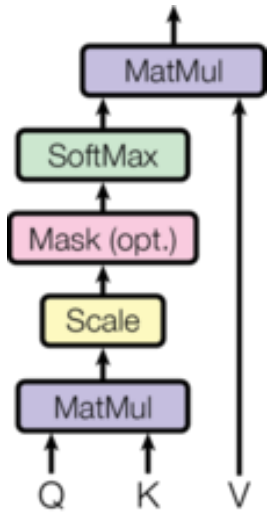
from an Amazon commercial page
Self-attention will enable us to compute contextual word embeddings.

The Concept of Attention (Self-Attention)

In the paper *Attention is all you need*, Vaswani et al. (2017) use three kinds of vectors, queries, keys, and values. Here we will use one type corresponding to GloVe embeddings.

We compute the attention this way:

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



where d_k is the dimension of the input (embedding length).

The softmax function is defined as:

$$\text{softmax}(x_1, x_2, ..., x_j, ..., x_n) = (\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, ..., \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}, ..., \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}})$$

The Meaning of $QK^\intercal$

$QK^\intercal$ is the dot product of the embedding vectors, GloVe here. It will tell us the similarity between the words

This is analogous to cosine similarity (but nonnormalized):

	i	must	go	back	to	my	ship	and	to	my	crew
i	1.00	0.75	0.86	0.76	0.73	0.90	0.35	0.65	0.73	0.90	0.42
must	0.75	1.00	0.85	0.68	0.87	0.69	0.42	0.69	0.87	0.69	0.45
go	0.86	0.85	1.00	0.84	0.84	0.81	0.41	0.68	0.84	0.81	0.49
back	0.76	0.68	0.84	1.00	0.83	0.76	0.49	0.77	0.83	0.76	0.51
to	0.73	0.87	0.84	0.83	1.00	0.68	0.54	0.86	1.00	0.68	0.51
my	0.90	0.69	0.81	0.76	0.68	1.00	0.38	0.63	0.68	1.00	0.44
ship	0.35	0.42	0.41	0.49	0.54	0.38	1.00	0.46	0.54	0.38	0.78
and	0.65	0.69	0.68	0.77	0.86	0.63	0.46	1.00	0.86	0.63	0.49
to	0.73	0.87	0.84	0.83	1.00	0.68	0.54	0.86	1.00	0.68	0.51
my	0.90	0.69	0.81	0.76	0.68	1.00	0.38	0.63	0.68	1.00	0.44
crew	0.42	0.45	0.49	0.51	0.51	0.44	0.78	0.49	0.51	0.44	1.00

Vaswani's attention score

The attention scores are scaled and normalized by the softmax function.

$$\text{softmax}(\frac{QK^\intercal}{\sqrt{d_k}}),$$

	i	must	go	back	to	my	ship	and	to	my	crew
i	0.36	0.05	0.07	0.05	0.04	0.19	0.01	0.02	0.04	0.19	0.01
must	0.14	0.20	0.10	0.06	0.11	0.10	0.03	0.05	0.11	0.10	0.02
go	0.18	0.09	0.14	0.09	0.08	0.13	0.02	0.04	0.08	0.13	0.02
back	0.14	0.05	0.09	0.19	0.08	0.12	0.03	0.06	0.08	0.12	0.03
to	0.11	0.11	0.09	0.09	0.15	0.08	0.04	0.07	0.15	0.08	0.03
my	0.19	0.03	0.05	0.04	0.03	0.29	0.01	0.02	0.03	0.29	0.01
ship	0.03	0.03	0.03	0.04	0.05	0.03	0.55	0.03	0.05	0.03	0.13
and	0.10	0.08	0.07	0.10	0.12	0.09	0.04	0.15	0.12	0.09	0.04
to	0.11	0.11	0.09	0.09	0.15	0.08	0.04	0.07	0.15	0.08	0.03
my	0.19	0.03	0.05	0.04	0.03	0.29	0.01	0.02	0.03	0.29	0.01
crew	0.06	0.05	0.05	0.06	0.05	0.06	0.21	0.04	0.05	0.06	0.31

Attention

We use these scores to compute the attention.

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

For *ship:*

```
attention_ship = (0.03 * embeddings_dict['i'] +
                  0.03 * embeddings_dict['must'] +
                  0.03 * embeddings_dict['go'] +
                  0.03 * embeddings_dict['back'] +
                  0.04 * embeddings_dict['to'] +
                  0.05 * embeddings_dict['my'] +
                  0.55 * embeddings_dict['ship'] +
                  0.03 * embeddings_dict['and'] +
                  0.05 * embeddings_dict['to'] +
                  0.03 * embeddings_dict['my'] +
                  0.13 * embeddings_dict['crew'])
```

where the *ship* vector received 13% of its value from *crew*

Code Example

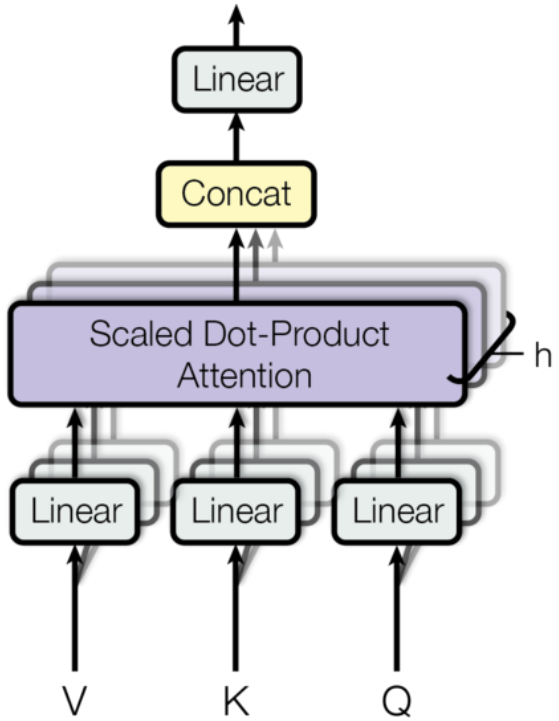
Experiment: Jupyter Notebook: `https://github.com/pnugues/edap30/blob/main/programs/6-attention.ipynb`
(First part of the notebook)

Multihead Attention

This attention is preceded by dense layers:
If X represents complete input sequence (all the tokens), we have:

$$\begin{aligned} Q &= XW_Q, \\ K &= XW_K, \\ V &= XW_V. \end{aligned}$$

And followed by another dense layer.
In addition, most architectures have parallel attentions, where the outputs (called heads) are concatenated (multihead)



From *Attention is all you need*, Vaswani et al. (2017)

Code Example

PyTorch has an implementation of this architecture with the `MultiheadAttention()` layer.
Experiment: Jupyter Notebook: `https://github.com/pnugues/edap30/blob/main/programs/6-attention.ipynb`
(Second part of the notebook)

Transformers: The Encoder

In transformers, the encoder is a structure, where:

1. The first part of the layer is a multihead attention;
2. We reinject the input to the attention output in the form of an addition:

$$X + \text{Attention}(Q, K, Q).$$

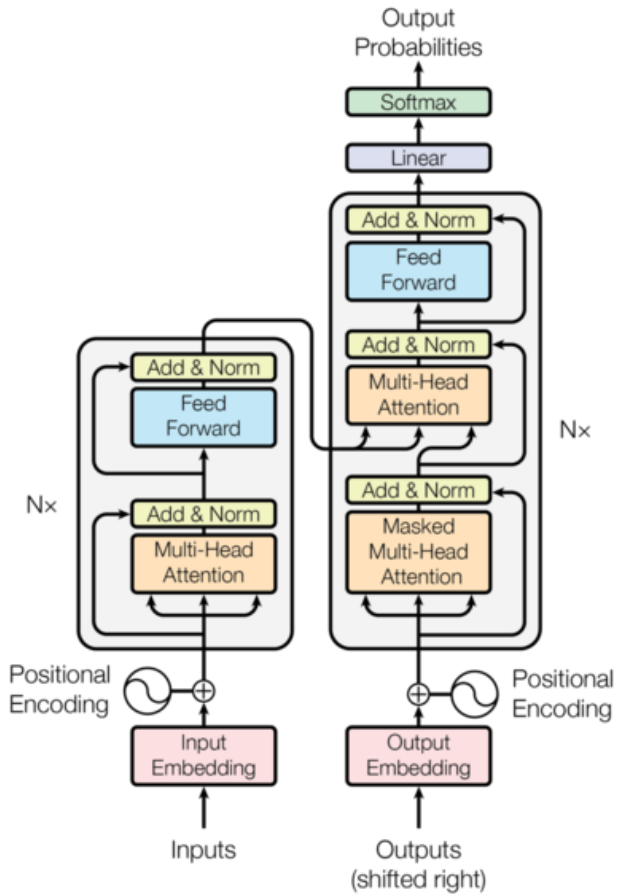
   This operation is called a skip or residual connection, which improves stability.

3. The result in then normalized per instance, i.e. a unique sequence, defined as:

$$x_{i,j_{norm}} = \frac{x_{i,j} - \bar{x}_{i,.}}{\sigma_{x_{i,.}}}.$$

   The input distribution is more stable and improves the convergence

4. It is followed by dense layers.



Left part, from *Attention is all you need*, Vaswani et al. (2017)

Code Example

Experiment:

As a personal work and to gain a deeper understanding, you can read a tutorial in PyTorch:

`http://nlp.seas.harvard.edu/annotated-transformer/`

Training Transformers

Transformers, such as BERT, are often trained on masked language models with two tasks:

1. For a sentence, predict masked words: We replace 15% of the tokens with a specific mask token and we train the model to predict them. This is just a cloze test;
2. For a pair of sentences, predict if the second one is the successor of the first one;

Taking the two first sentences from the *Odyssey*:

Tell me, O Muse, of that ingenious hero who travelled far and wide after he had sacked the famous town of Troy.
Many cities did he visit, and many were the nations with whose manners and customs he was acquainted;

Masked language models

We add two special tokens: `[CLS]` at the start of the first sentence and `[SEP]` at the end of both sentences, and the token `[MASK]` to denote the words to predict.

We would have for the first task:

> *[CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP]*

For the second task, we would have as input:

> *[CLS] Tell me, O Muse, of that [MASK] hero who travelled far and wide [MASK] he had sacked the [MASK] town of Troy. [SEP] Many cities did he [MASK visit], and many were the [MASK nations] with whose manners [MASK and] customs he was acquainted; [SEP]*

where the prediction would return that the second sentence is the next one (as opposed to random sequences)

Positional Embeddings

BERT, the first encoder-based transformer, Devlin et al. (2019), maps each token to three embedding vectors:

- the token embedding,
- the position of the token in the sentence (positional embeddings), and
- the segment embeddings (we will skip this part).

The three kinds of embeddings are learnable vectors.
In the BERT base version, each embedding vector has 768 dimensions.
Let us consider two sentences simplified from the *Odyssey:*

Tell me of that hero. Many cities did he visit.

Input:	[CLS]	Tell	me	of	that	hero	[SEP]	Many	cities	did	he	visit	[SEP]
Token	$\mathbf{E}_{[CLS]}$	$\mathbf{E}_{tell}$	$\mathbf{E}_{me}$	$\mathbf{E}_{of}$	$\mathbf{E}_{that}$	$\mathbf{E}_{hero}$	$\mathbf{E}_{[SEP]}$	$\mathbf{E}_{many}$	$\mathbf{E}_{cities}$	$\mathbf{E}_{did}$	$\mathbf{E}_{he}$	$\mathbf{E}_{visit}$	$\mathbf{E}_{[SEP]}$
Segment	$\mathbf{E}_A$	$\mathbf{E}_A$	$\mathbf{E}_A$	$\mathbf{E}_A$	$\mathbf{E}_A$	$\mathbf{E}_A$	$\mathbf{E}_A$	$\mathbf{E}_B$	$\mathbf{E}_B$	$\mathbf{E}_B$	$\mathbf{E}_B$	$\mathbf{E}_B$	$\mathbf{E}_B$
Position	$\mathbf{E}_0$	$\mathbf{E}_1$	$\mathbf{E}_2$	$\mathbf{E}_3$	$\mathbf{E}_4$	$\mathbf{E}_5$	$\mathbf{E}_6$	$\mathbf{E}_7$	$\mathbf{E}_8$	$\mathbf{E}_9$	$\mathbf{E}_{10}$	$\mathbf{E}_{11}$	$\mathbf{E}_{12}$

Code Example

Positional embeddings:

Experiment: Jupyter Notebook:

`http://nlp.seas.harvard.edu/annotated-transformer/`

Note that:

- In the initial paper, sine and cosine positional embeddings are fixed
- They are added
- Trainable positional embeddings (BERT) have nearly identical performance

Model size

Transformers are trained on large corpora like the colossal clean crawled corpus (`https://arxiv.org/abs/2104.08758`) and encapsulate semantics found in text in the form of numerical matrices.
This results in large models (Devlin et al., 2019):

> *In this work, we denote the number of layers (i.e., Transformer blocks) as L, the hidden size as H, and the number of self-attention heads as A. We primarily report results on two model sizes: $BERT_{BASE}$ (L=12, H=768, A=12, Total Parameters=110M) and $BERT_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M).*

Transformers can then act as pre-trained models for a variety of tasks.
See the list from Huggingface
Finally, an interesting reading: `https://sayakpaul.medium.com/an-interview-with-colin-raffel-research-scientist-at-google-5`

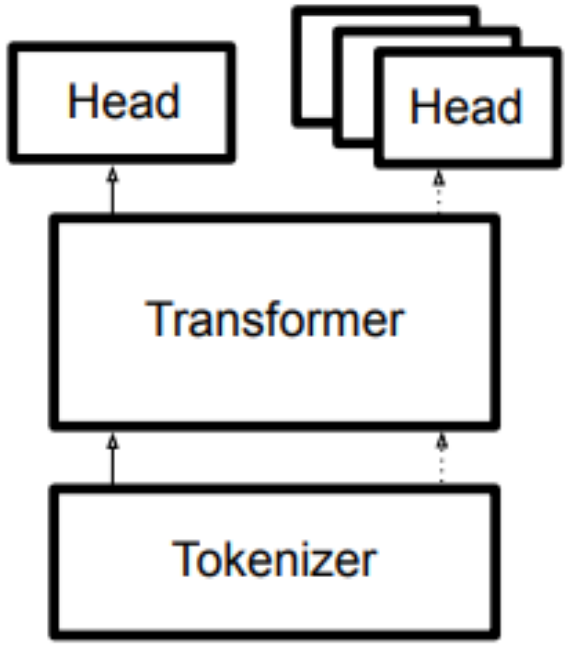
Trends in NLP

The overall trend:
Words (or subwords) → Large models (huge matrices) → Applications

- In general, language processing requires significant processing power;
- This overall resulted in massive improvements in most areas of NLP.

Applying Transformers

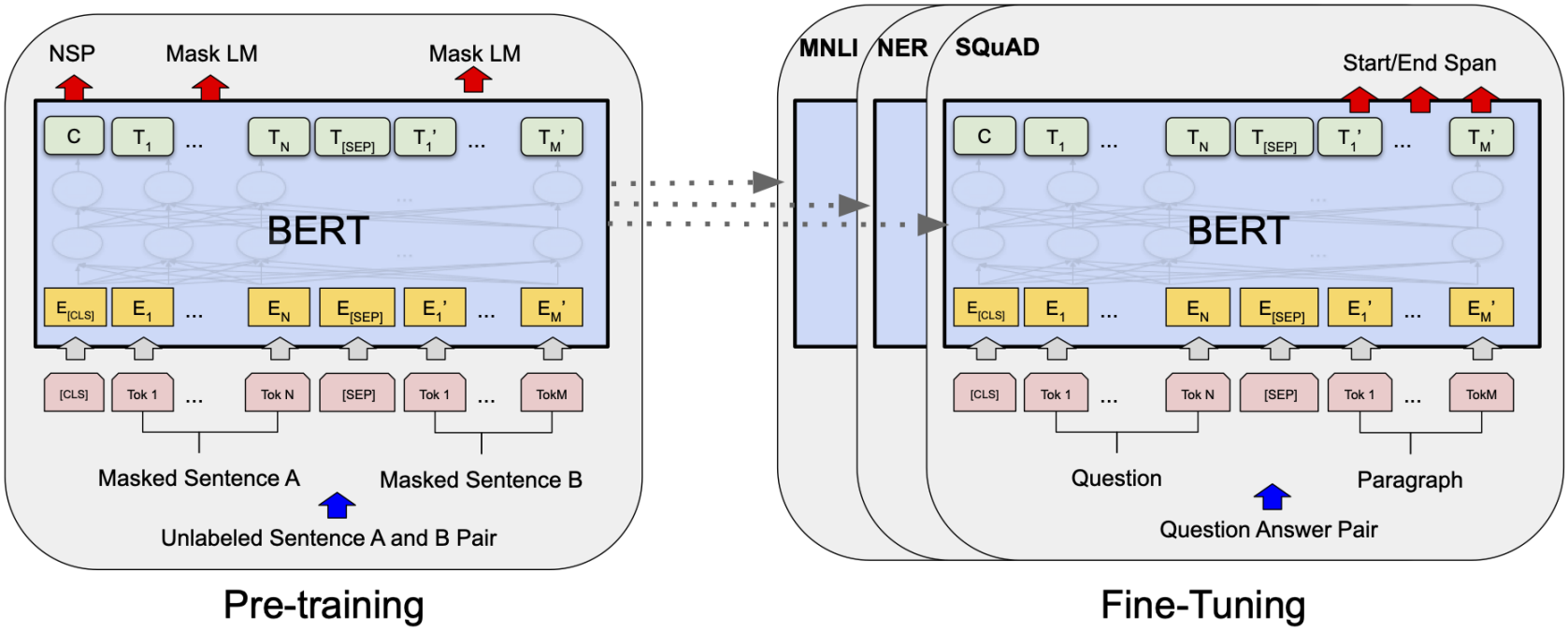
- Matrices in the transformer architecture encapsulate massive knowledge from text.
- We can apply them to tasks beyond what they have been trained for (masked language model)
- We use them as pretrained models and fine-tune a so-called dedicated *head*.
- The simplest head is a logistic regression



Picture from Wolf et al., Transformers: State-of-the-art Natural Language Processing, EMNLP Demos 2020.

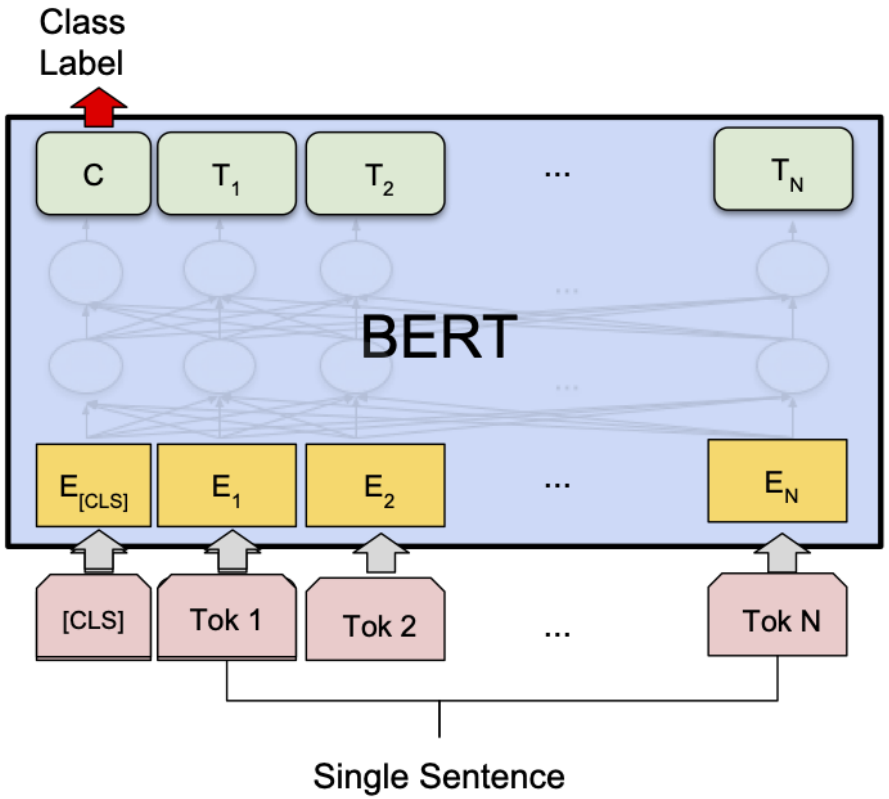
Transfer Learning

Transfer learning consists of a costly **pretraining** step and an adaptation to applications called **fine-tuning**.



from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. Note the picture comes from the second version of the paper from 2019

Application: Sentence Classification



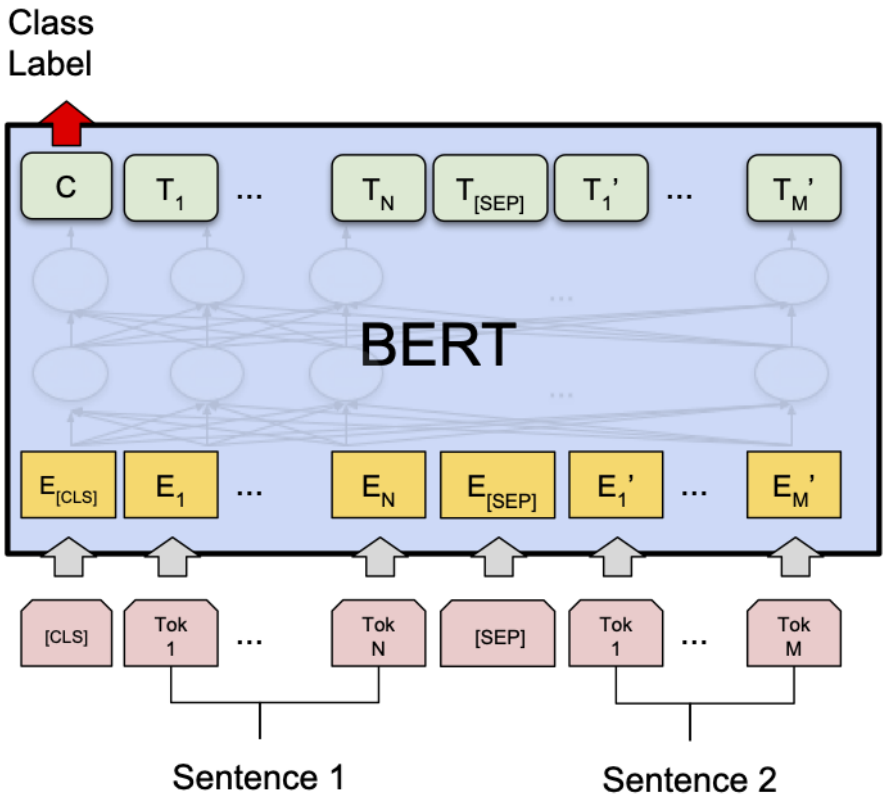
(b) Single Sentence Classification Tasks: SST-2, CoLA

from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

Code Example

Experiment: Jupyter Notebook:
https://github.com/nlp-with-transformers/notebooks/blob/main/02_classification.ipynb

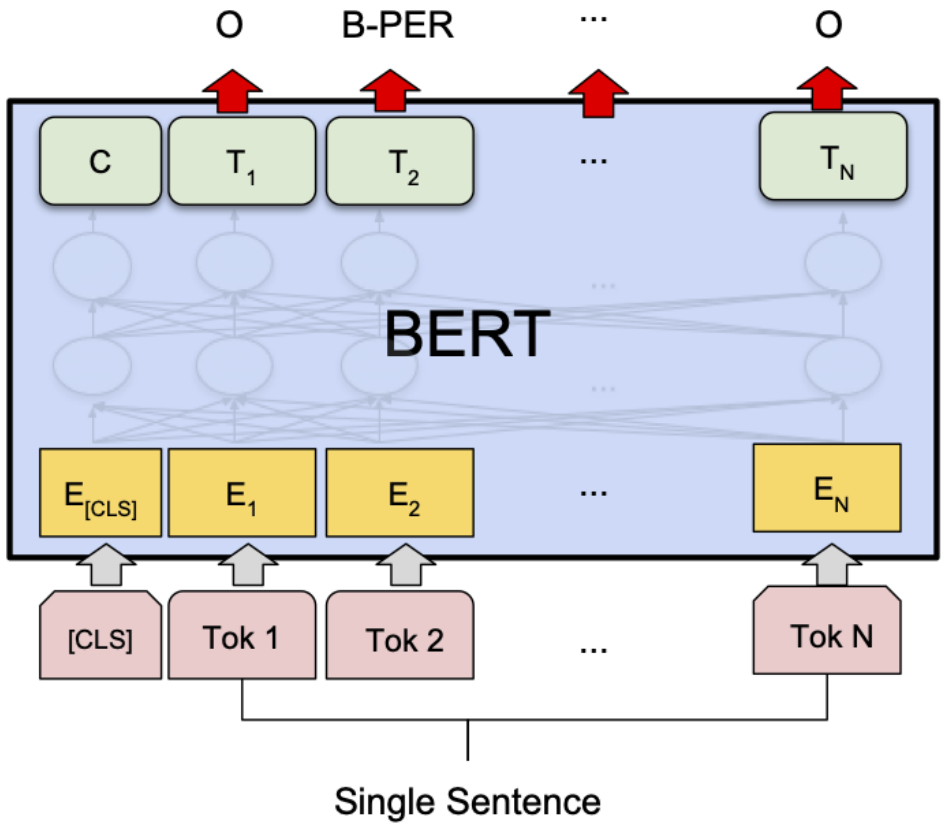
Application: Sentence Pair Classification



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

Application: Sequence Tagging

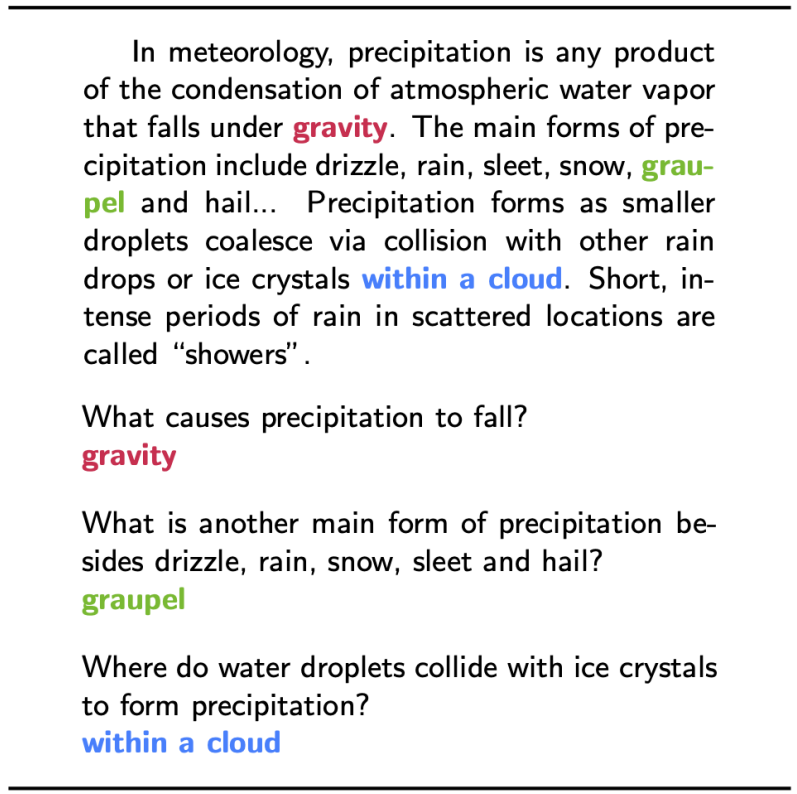


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

Stanford Question Answering Dataset (SQuAD)

- Consists of 100,000 questions and paragraphs from wikipedia containing the answers
- The answer is a segment in the text (factoid QA)
- Complemented by SQuAD 2.0 with unanswerable questions
- SQuAD started an intense competition. See the impressive leaderboard `https://rajpurkar.github.io/SQuAD-explorer/`



In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 1: Question-answer pairs for a sample passage in the SQuAD dataset. Each of the answers is a segment of text from the passage.

Form Rajpurkar et al., *SQuAD: 100,000+ Questions for Machine Comprehension of Text*, 2016

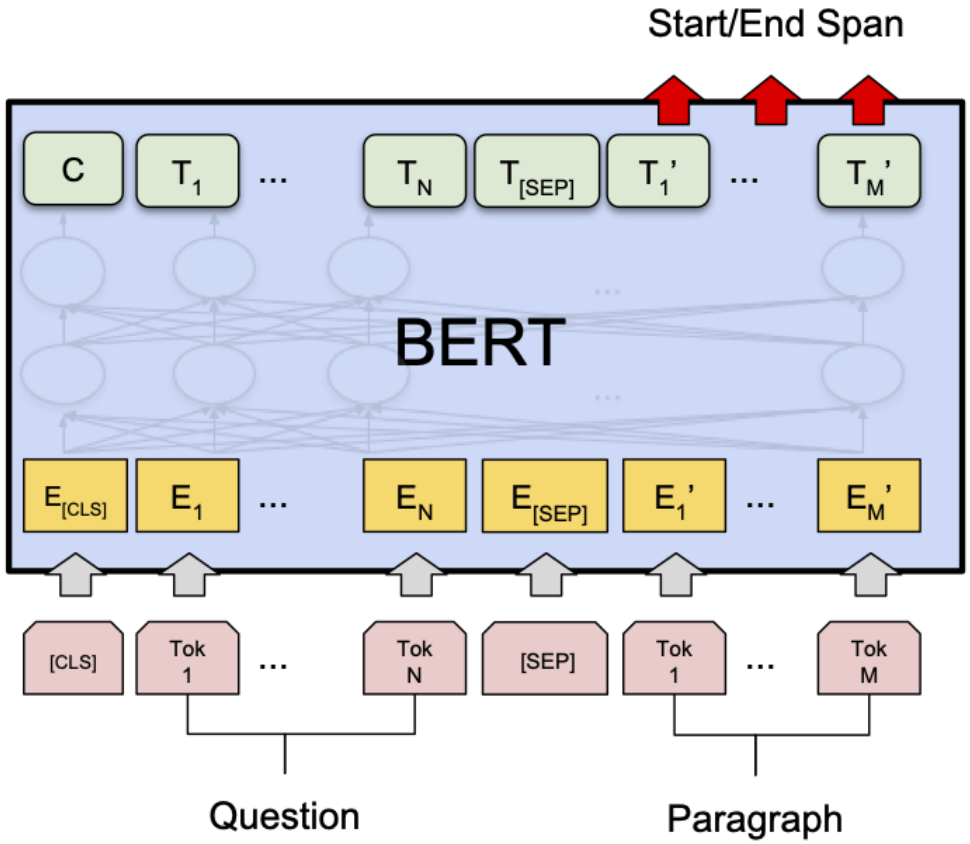
Question Answering

Question parsing and classification: Syntactic parsing, entity recognition, answer classification

Document retrieval. Extraction and ranking of passages: Indexing, vector space model.

Extraction and ranking of answers: Answer parsing, entity recognition

Application: Question Answering



(c) Question Answering Tasks:
SQuAD v1.1

from Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019

Subword Tokenization Techniques

In a multilingual context, it is often preferable to use a subword tokenization, where variants include:

- Byte-Pair Encoding (BPE) with characters (GPT): Starting from the set of characters,
  1. Merge the most frequent adjacent pair (bigram) to form a new subword,
  2. Re-segment your corpus with the new set of symbols, and
  3. Iterate until you have reached a pre-defined vocabulary size (30,000)
- BPE with bytes (GPT-2): Same as above with bytes
- WordPiece (BERT): Selects the pair that increases the likelihood the most ($\frac{P(x,y)}{P(x)P(y)}$);
- SentencePiece (T5): BPE and unigram language model

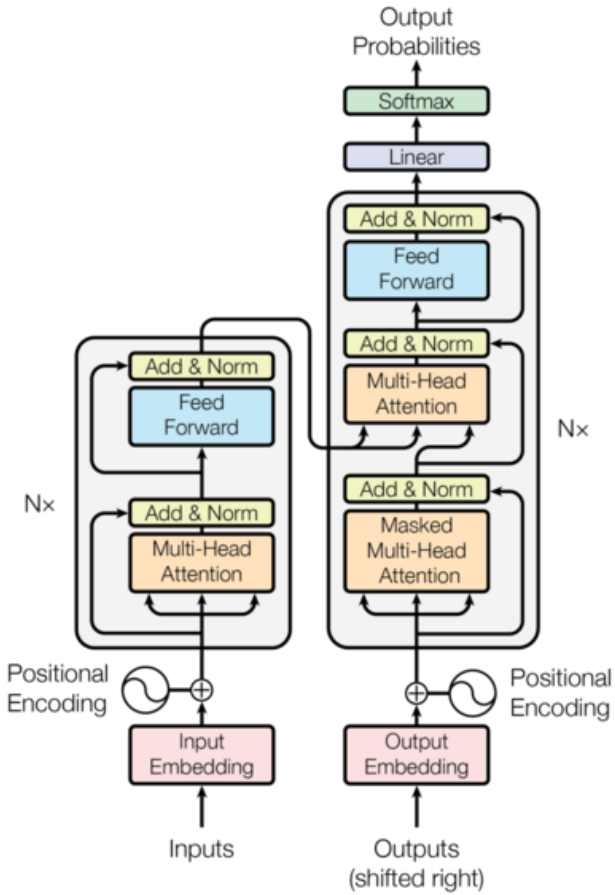
The tokenizer splits the text into subwords. By construction, there is no unknown word. You will fall back to the bytes

Code Example

Jupyter notebook: https://github.com/pnugues/edap30/blob/main/programs/7-subword-tokenizer.ipynb

Decoder

- The right part of the initial transformer
- Reference papers:
  1. *Improving Language Understanding by Generative Pre-Training* by Radford et al. (2018)
     Link: `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`
  2. *Language Models are Unsupervised Multitask Learners* by Radford et al. (2018) Links:
     `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`
     `https://github.com/openai/gpt-2`



Decoder Training

Trained as an autoregressive language model:

$$P(x) = \prod_{i=1}^{n} P(s_i | s_1, \ldots, s_{i-1}).$$

Fine-tuning applications in the initial paper

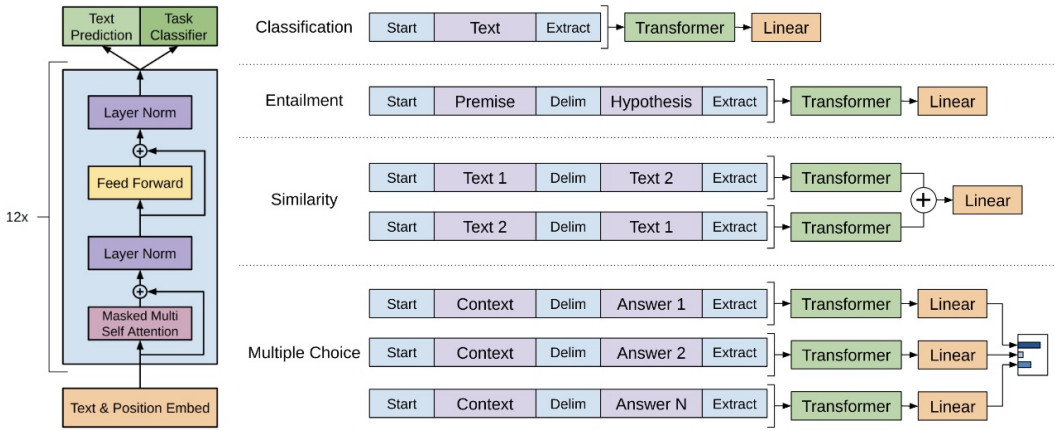


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

Training examples in the second paper:

- translate to french, english text, french text;
- answer the question, document, question, answer

Architectures

- Original encoder-decoder (sequence-to-sequence): T5, mT5
- Encoder (MLM): BERT, mBERT, RoBERTa, XML-RoBERTa
- Decoder (autoregressive): GPT, LLaMA, Mistral, Claude, Qwen, etc.

Research Directions

Text processing architectures have shifted from pipelines of linguistic modules to large language models fined-tuned on specific applications.
Regular benchmarks on popular applications:

- Classification and textual entailment
- Named entity recognition
- Question answering
- Translation
- Summarization and generation

See:

- GLUE (`https://gluebenchmark.com/`)
- SuperGLUE (`https://super.gluebenchmark.com/`)
- Machine translation (`https://www.statmt.org/wmt22/`)
- Text embedding (`https://huggingface.co/spaces/mteb/leaderboard`)
- Elaborate tasks (`https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`)