Deep Learning for Natural Language Processing

Lecture 10 – Text classification 4: self-attention and BERT

Dr. Martin Tutek

June 20, 2023

Ubiquitous Knowledge Processing
Department of Computer Science
Technical University of Darmstadt

UBIQUITOUS KNOWLEDGE PROCESSING

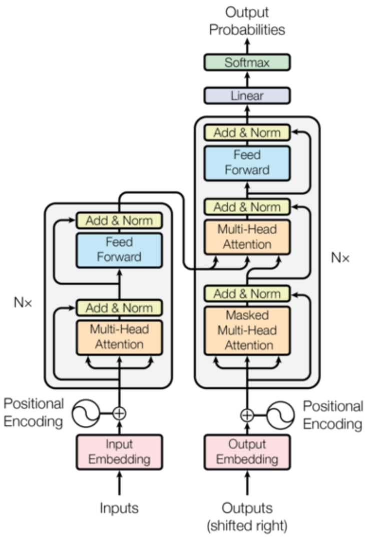
UKP Web

Recap

In the previous lecture we:

- Introduced the Transformer architecture
- Explained what we gain from *contextualized* representations
- Analyzed the Transformer attention block
- Introduced byte-pair encodings & what we gain by them
- Introduced positional embeddings & why we need them

Recap: the Transformer architecture



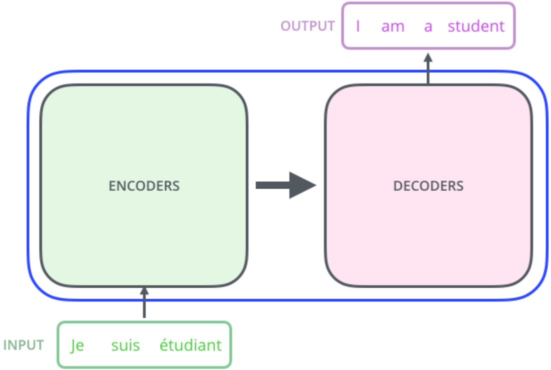
Recap: Transformer for machine translation

We studied the Transformer encoder-decoder for machine translation

Image source: The illustrated Transformer

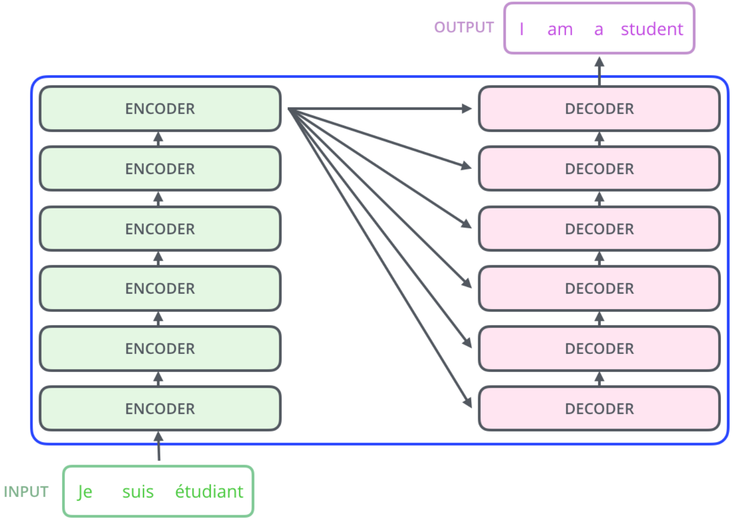
The model is built of stacked encoder and decoder blocks

Recap: Transformer encoder-decoder



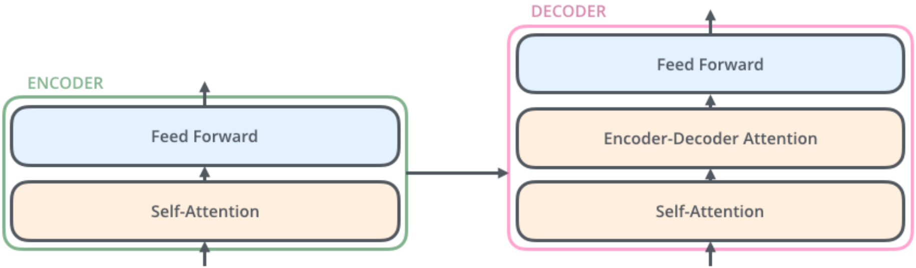
The encoder and decoder are made of stacked **encoder** and **decoder blocks**

Recap: Transformer encoder-decoder

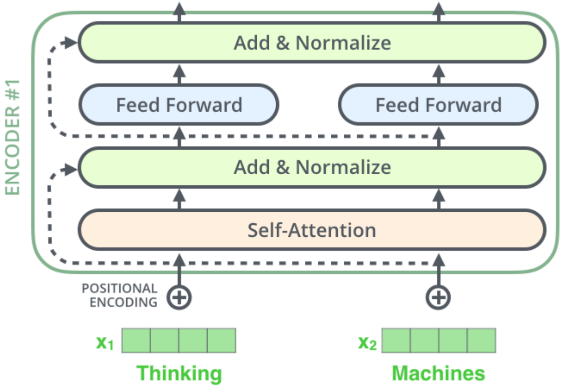


Recap: Transformer encoder and decoder block

The encoder and decoder blocks are different – the decoder block has an additional **encoder-decoder attention** (cross-attention) layer



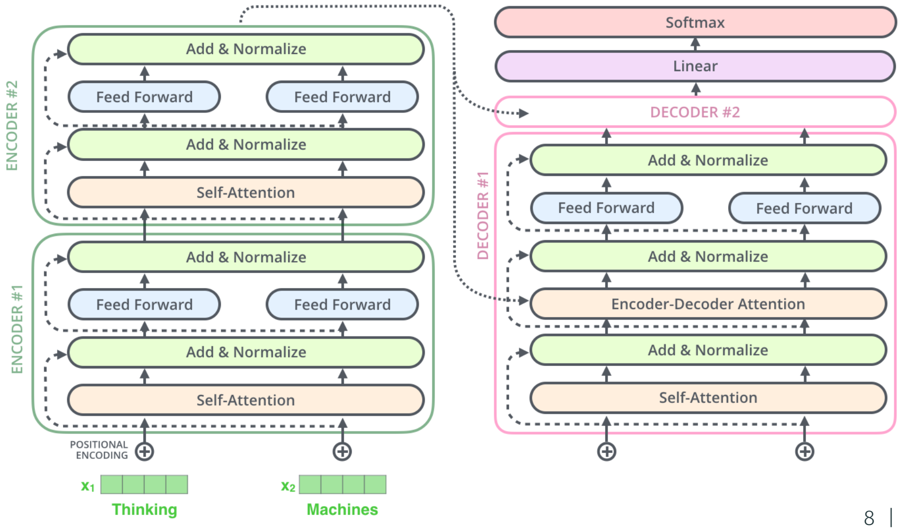
Recap: Transformer encoder block



Each Transformer encoder block consists of:

1. Self-attention: **contextualize** representations
2. **Residual** connection + **normalization**
3. **Feed-forward** layer (1 hidden layer NN)
4. **Residual** connection + **normalization**

Recap: Transformer encoder-decoder



Recap: Transformer decoding in practice

Transformer decoding gif

Next steps?

Next steps?

BERT

- Fine-tuning BERT
- Variants of pretraining tasks

Motivation

The transformer encoder-decoder model is **really** good at sequence-to-sequence tasks

- It **scales** well (to many layers & parameters)
- It **performs** well on long sequences
- It's **easier to optimize** (residual connections)
- It's **faster to run** (parallel processing in encoder)

However – we can **only** use it for the **task it was trained on**

Motivation

Transfer learning

... is applying knowledge gained when **solving one task** to a **related** task.

Gained knowledge $\rightarrow$ encoded in a **trained model**.

Where have we seen something like this before?

- **Word2vec** (CBOW & Skip-gram)

We **train** the word embeddings on an *auxiliary task*, then use them as input for other models Can we apply this to **Transformer** encoders to obtain pretrained **contextualized** embeddings?

BERT

Next steps?

BERT

- Fine-tuning BERT
- Variants of pretraining tasks

BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language

BERT

What **we have**

- A model: the Transformer

What **we want**

- Pretrained *contextualized* word representations

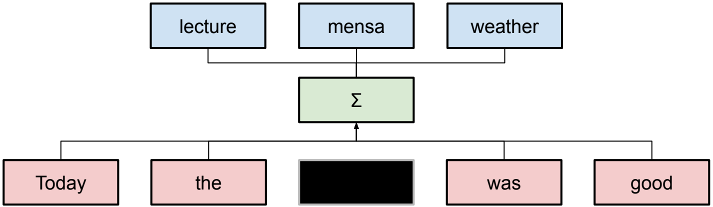
What **we need**

- The **auxiliary pretraining task**
- ... ideally, it should not require labeled data (expensive)

BERT: pretraining objective

Recall: what was the word2vec training objective?

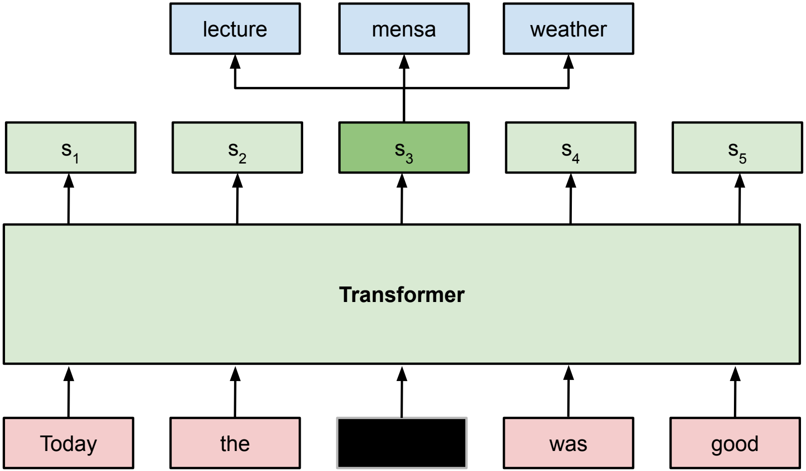
- **CBOW**: predict **center word** given **context words**



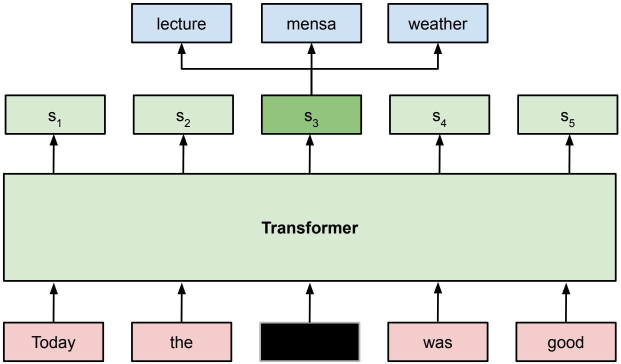
- **Skip-gram**: predict **context words** given **center word**

Can we use a **similar** task with Transformer models?

BERT: pretraining objective



BERT: pretraining objective



1. For an input sequence $\{\boldsymbol{x}_i\}_{i=1}^{n}$, we **mask** an input token(s)
2. We encode the inputs with a Transformer **encoder**
3. We **reconstruct** the masked token(s)

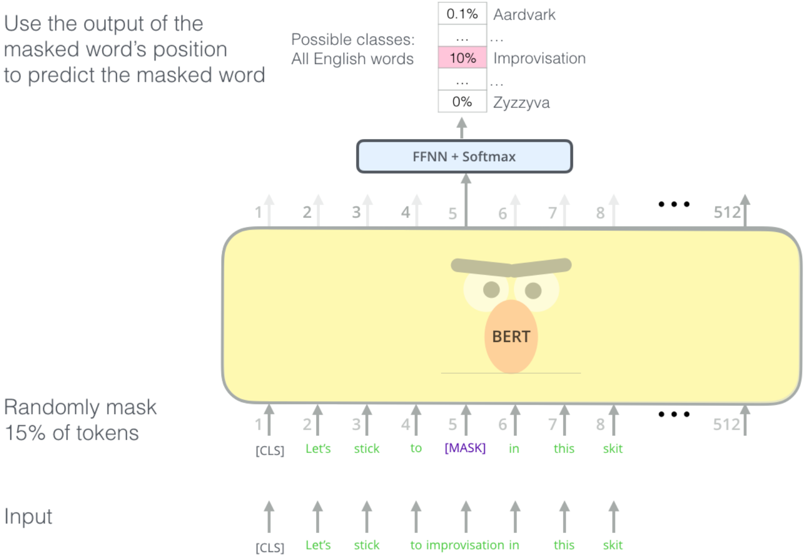
What is *masking*?

BERT: masked language modeling (MLM)

The **Transformer** model contextualizes an input sequence $\{\boldsymbol{x}_i\}_{i=1}^n$ of (subword) tokens into a sequence of hidden states $\{\boldsymbol{s}_i\}_{i=1}^n$.

1. With probability p_{mlm}, mask **each input token** ($p_{mlm} = 0.15$)
2. **If** a token is masked
   - 80% of the time, replace it with a special `<MASK>` token
   - 10% of the time, replace it with a **random** token
   - 10% of the time, **do not mask it**
3. **Only** for the **masked tokens**
   - Predict which token was masked

BERT: masked language modeling (MLM) [Image source]



BERT: masked language modeling (MLM)

And... it works

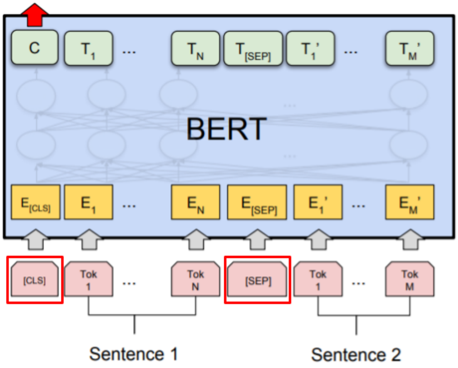
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

GLUE Test results (GLUE benchmark). Table from BERT paper

However – we are *not there yet*

- How do we obtain **sentence representations**?

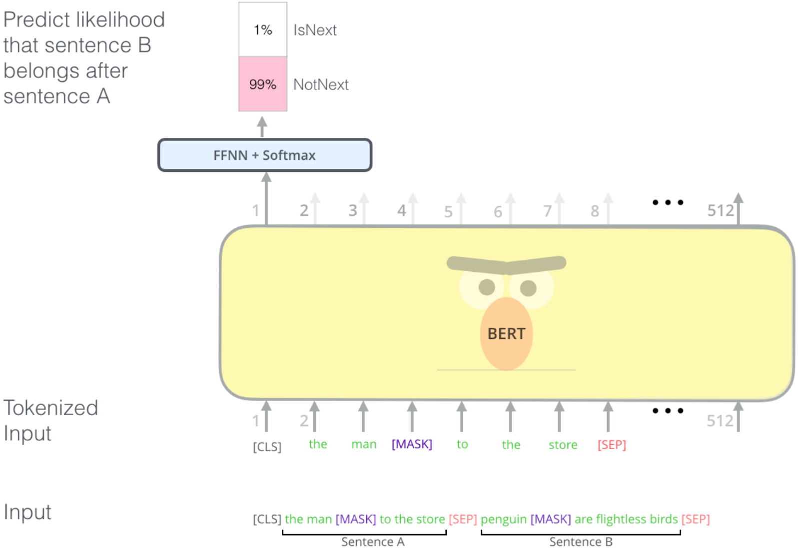
BERT: next sentence prediction (NSP)



We want to have a sentence representation **out-of-the-box**

1. We add a special **CLS** token as the **sentence representation**
2. We add a special **SEP** token to separate **two input sentences**
3. We predict (based on the CLS token) if Sent.1 **directly precedes** Sent.2 in the pretraining dataset

BERT: next sentence prediction (NSP)[Image source]



BERT: pretraining

Combining the **MLM** and **NSP** objectives:

1. Take a **large corpus** of unstructured text (Wikipedia, BookCorpus) and retain information about position of **sentences** in each article
2. Create the input sequence: $\mathbf{in} = [\text{CLS}]\{\boldsymbol{x}_i^1\}_{i=1}^{n_1}[\text{SEP}]\{\boldsymbol{x}_i^2\}_{i=1}^{n_2}$
   2.1 Sample Sentence_1 from the dataset
   2.2 With $p_{nsp} = 0.5$, take Sentence_2 as the following sentence (heads) **or** randomly sample it (tails)
3. Mask $p_{mlm} = 0.15$ of **non-special** input tokens (recall: $80/10/10$)
4. Encode inputs with transformer encoder: $\mathbf{trf}(\mathbf{in}) \rightarrow \{\boldsymbol{s}_i\}_{i=1}^{n_1+n_2+2}$
5. Pretraining tasks
   5.1 **MLM**: reconstruct masked tokens $\boldsymbol{s}_i \rightarrow \boldsymbol{x}_i \quad \forall i \in \{\text{masked}\}$;
   5.2 **NSP**: predict if sentences are successors $\boldsymbol{s}_1 \rightarrow \{0, 1\}$.

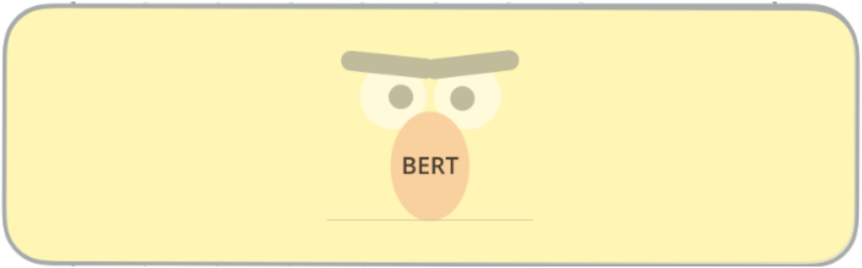
BERT: summary

BERT is a **pretrained language model (PLM)**

- Through a language modeling pretraining task, the model has learned to recognize **patterns of language** and apply them for the task of **text reconstruction**
- Text reconstruction is, however, **rarely useful** in isolation
- In practice: use the PLM as a **starting point** (a very good initialization) for **fine-tuning** (additional training) for another task

BERT: applications

Now we have the pretrained BERT – a Transformer encoder.



What's next?

- How to use this model for **downstream tasks**?

BERT

Fine-tuning BERT

Using BERT for NLP tasks

Fine-tuning is the procedure where we start from a base **pretrained** model and adapt its internal representations to our task.

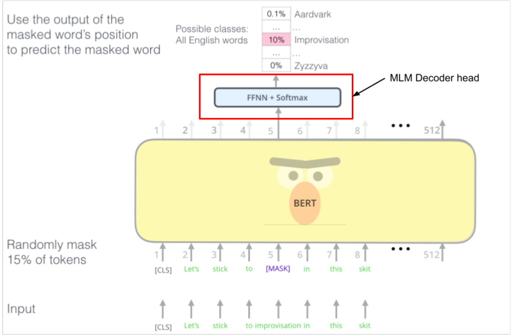
Fine-tuning variants:

1. **Vanilla fine-tuning**: add a **decoder head** to the model, then:
   1.1 Train **only** the decoder head;
   1.2 Train **progressively** more layers: first the decoder head, then also the last layer, then also the second to last ...;
   1.3 Train the **entire network** at the same time, maybe with **different learning rates** per layer.
2. **Adapters**: additional randomly initialized layers inserted inside the transformer layers
3. **Prompting** & **in-context learning**: future lectures

Using BERT for NLP tasks

What exactly are **decoder heads**?

Randomly initialized additional layers (usually linear) added **on top** of the pretrained model which **perform the downstream task**.



Using BERT: single sequence classification

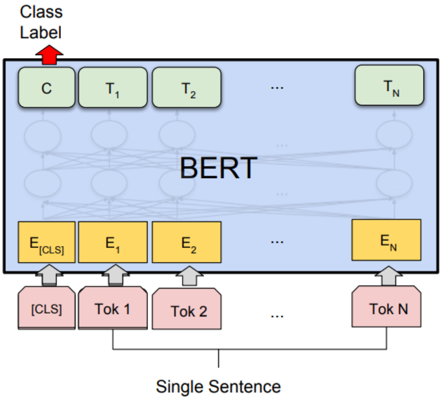


Image from BERT paper

For single **sequence** classification:

1. Add a randomly initialized **task decoder head** to the model
2. **Encode** the sequence along with the [CLS] special token
3. Use the **[CLS]** representation as input to decoder head

Alternatives to using CLS?

- Averaging over **token representations**
- … from the **last four layers**

Using BERT: sentence pair classification

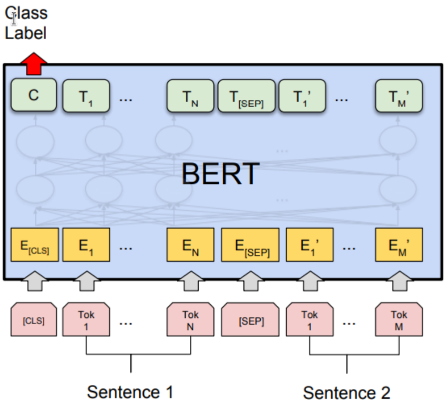


Image from BERT paper

For **pair sequence** classification:

1. Add a randomly initialized **task decoder head** to the model
2. **Encode both sequences** along with the [CLS] special token
3. Use the **[CLS]** token representation as input to decoder head

Using BERT: span extraction QA

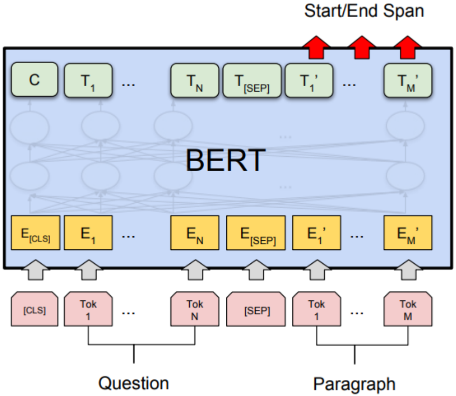


Image from BERT paper

For **span extraction** QA:

1. Add randomly initialized **start-of-span** and **end-of-span vectors** to the model.
2. **Encode both sequences**
3. Highest dot product of **token representation** with start-of-span and end-of-span is the predicted span
   - … such that end-of-span > start-of-span

Using BERT: sequence labeling

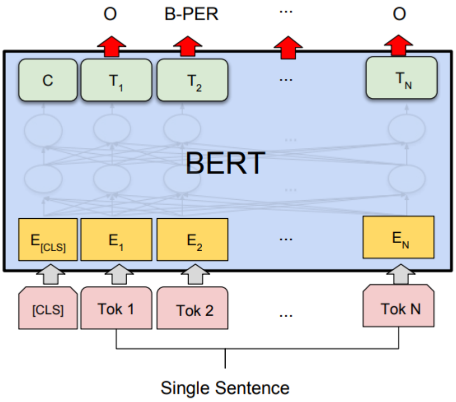


Image from BERT paper

For **sequence labeling**:

1. Add a randomly initialized **task decoder head** to the model
2. **Encode** the sequence along with the [CLS] special token
3. Use the **token representations** as inputs to decoder head

BERT

Variants of pretraining tasks

Variants of pretraining tasks

We have used **MLM** and **NSP** – are there some *better* tasks for pretraining language models?

Perhaps in an **encoder-decoder** setup?

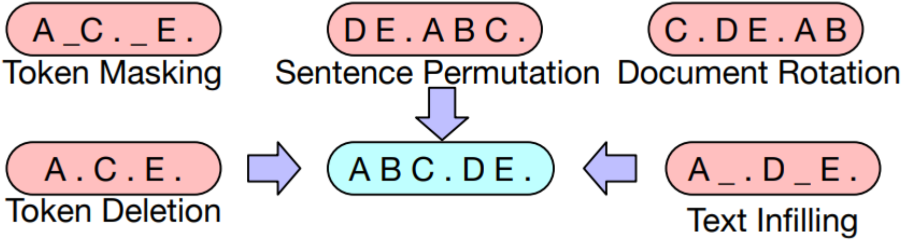


Image from BART paper

Variants of pretraining tasks

- **Language modeling**
- **Token masking**: MLM
- **Token deletion**: masking, but **completely removes tokens** from input – model needs to determine where a token is missing
- **Text infilling**: masking, but **multiple** tokens are replaced with a **single** [MASK] token at the same time
- **Sentence permutation**: input **permuted sentence**, reconstruct correct word order (*linearization*)
- **Document rotation**: document is rotated so that it starts from a **random token**. The model has to determine the actual start of the document.

Variants of (supervised) pretraining tasks

What if we decide to use **supervised data** – but from various datasets?

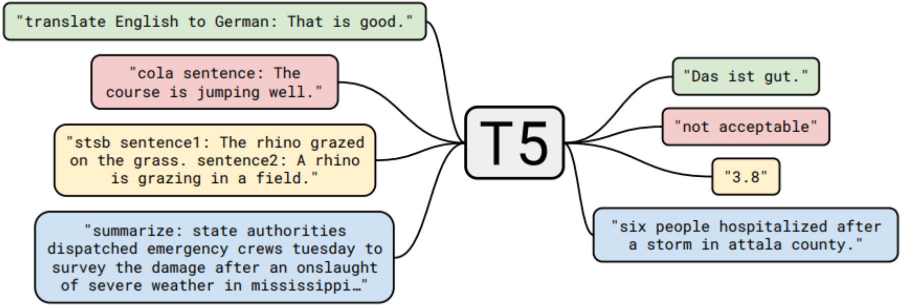
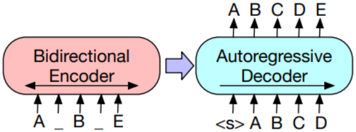


Image from [T5 paper]

Pretrained language model architectures

(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.

(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Image from [BART paper]

Takeaways

- BERT is a pretrained language model which produces **contextualized** token representations of input TeXstudio
- It can be used as an **initialization** (starting point) for **fine-tuning** task-specific models
  - Extras: [CLS] and [SEP] tokens
  - Applications of BERT in classification, sequence labeling and span-extraction QA
- Other pretraining tasks are also viable
  - **Unsupervised**: sentence permutation, text inifling
  - **Supervised**: translation, summarization

Useful resources

- The annotated Transformer by Sasha Rush
- The illustrated Transformer by Jay Allamar
- The illustrated BERT by Jay Allamar

License and credits

Credits

Martin Tutek