Deep Learning for Natural Language Processing

Lecture 12 – Contemporary LLMs: Prompting and in-context learning

Dr. Martin Tutek

July 04, 2023

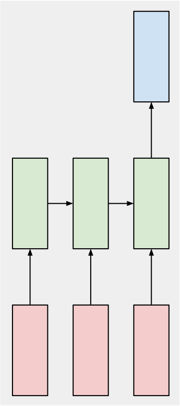
Ubiquitous Knowledge Processing
Department of Computer Science
Technical University of Darmstadt

UBIQUITOUS KNOWLEDGE PROCESSING

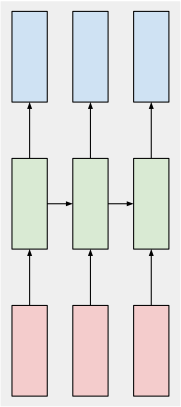
UKP Web

Recap

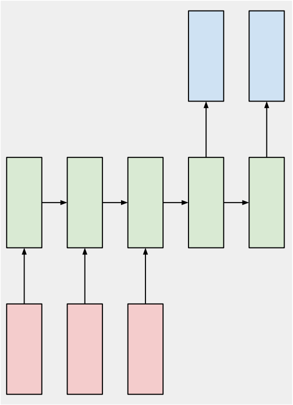
Overview of NLP task types



(a) Seq. classification



(b) Seq. labeling



(c) Sequence-to-sequence

Sequence classification

We want to categorize an input text (or some relation between multiple texts)

- Tasks: sentiment analysis, news article categorization, natural language inference,...
- Input: sequence of tokens $\{x_i\}_{i=1}^n$
- Output: class $y \in \boldsymbol{Y}$
  1. We **embed** the input tokens $\{\boldsymbol{x}_i\}_{i=1}^n \rightarrow \{\boldsymbol{e}_i\}_{i=1}^n$; $\boldsymbol{e}_i \in \mathbb{R}^{d_e}$
  2. We **encode** the input sequence into a **fixed size representation** by using a RNN or Transformer network $\{\boldsymbol{e}_i\}_{i=1}^n \rightarrow \boldsymbol{s}$; $\boldsymbol{s} \in \mathbb{R}^{d_m}$
  3. We feed the sequence representation into a **classification network** (decoder head) $\boldsymbol{s} \rightarrow \hat{\boldsymbol{y}}$; $\boldsymbol{y} \in \mathbb{R}^k$

Sequence labeling

We want to categorize each token from the input sequence

- **Tasks**: part-of-speech tagging, named entity recognition,...
- **Input**: sequence of tokens $\{x_i\}_{i=1}^n$
- **Output**: sequence of labels $\{y_i\}_{i=1}^n \ ; \ y_i \in \boldsymbol{Y}$
    1. We **embed** the input tokens $\{\boldsymbol{x}_i\}_{i=1}^n \rightarrow \{\boldsymbol{e}_i\}_{i=1}^n \ ; \ \boldsymbol{e}_i \in \mathbb{R}^{d_e}$
    2. We **encode** the input sequence using a RNN or Transformer network $\{\boldsymbol{e}_i\}_{i=1}^n \rightarrow \{\boldsymbol{s}_i\}_{i=1}^n \ ; \ \boldsymbol{s}_i \in \mathbb{R}^{d_m}$
    3. We feed **each token** representation into a **classification network** (decoder head) $\{\boldsymbol{s}_i\}_{i=1}^n \rightarrow \{\hat{\boldsymbol{y}}_i\}_{i=1}^n \ ; \ \hat{\boldsymbol{y}}_i \in \mathbb{R}^k$

Sequence to Sequence

Based on an input sequence, we want to **generate** the corresponding output sequence

- **Tasks**: machine translation, text summarization,...
- **Input**: sequence of tokens $\{x_i\}_{i=1}^n$
- **Output**: sequence of **tokens** $\{y_i\}_{i=1}^n$; $y_i \in \boldsymbol{Y}$
  1. We **embed** the input tokens $\{\boldsymbol{x}_i\}_{i=1}^n \rightarrow \{\boldsymbol{e}_i\}_{i=1}^n; \qquad e_i \in \mathbb{R}^{d_e}$
  2. We **encode** the input sequence by using a RNN or Transformer network $\{\boldsymbol{e}_i\}_{i=1}^n \rightarrow \{\boldsymbol{s}_i\}_{i=1}^n$; $\boldsymbol{s}_i \in \mathbb{R}^{d_m}$
  3. We **decode** the output sequence based on the encoded input sequence...
     - 3.1 **Encoder-decoder**: ...with a **different** RNN or Transformer **network**
     - 3.2 **Encoder-only**: ...with the **same network**

Recap

Attention

Attention

Recall: problem of learning **long dependencies**

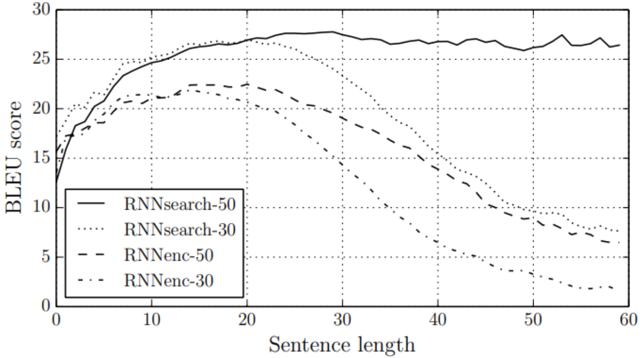


Figure from Bahdanau et al., 2014

Attention

The attention mechanism

Inputs: a **query** vector $\boldsymbol{q}$ and a sequence of **key** $\boldsymbol{K} = \{\boldsymbol{k}_i\}_{i=1}^{n}$ and **value** $\boldsymbol{V} = \{\boldsymbol{k}_i\}_{i=1}^{n}$ vectors.

Output: $\boldsymbol{a}$, a **weighted summary** of **value** vectors.

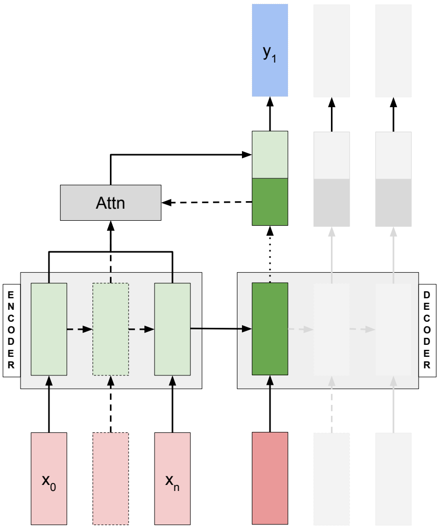
Query, values & keys are computed from neural network (RNN or Transformer) states.

$$\boldsymbol{a} = \sum_{i}^{n} \alpha_i \boldsymbol{v}_i \qquad\qquad \hat{\alpha}_i = f_{\text{attn}}(\boldsymbol{q}, \boldsymbol{k}_i) := \frac{\boldsymbol{q}^T \cdot \boldsymbol{k}_i}{\sqrt{d_m}}$$

Attention variants

- The **energy function**:
  1. (scaled) Dot product attention $\rightarrow \hat{\alpha}_i = \frac{\boldsymbol{q}^T \cdot \boldsymbol{k}_i}{\sqrt{d_k}}$
  2. Bahdanau (tanh) attention $\rightarrow \hat{\alpha}_i = \boldsymbol{W}_2 \text{tanh}(\boldsymbol{W}_1 [\boldsymbol{q} | \boldsymbol{k}_i])$
  3. Bilinear attention $\rightarrow \alpha_i = \boldsymbol{q}^T \boldsymbol{W} \boldsymbol{k}_i$
- **Parametrization**
  1. Use raw network hidden states
  2. Transform network hidden states (e.g. with a linear layer)
- **Direction**
  1. Self-attention (within encoder/decoder)
  2. Cross-attention (between encoder and decoder/two networks)

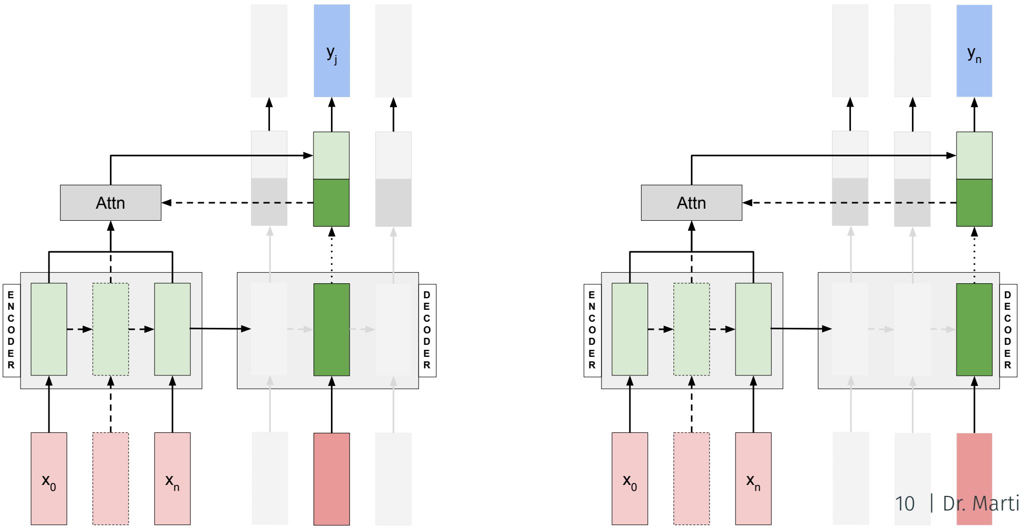
Attention in encoder-decoder RNNs



Encoder-decoder attention:

- We want to obtain information relevant for **current decoder state** from the encoder states
- We **concatenate** the attention output a to the current decoder state prior to prediction

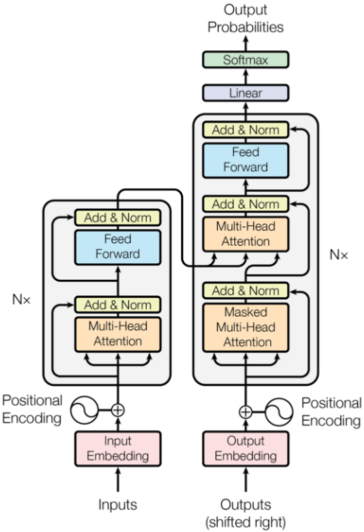
Attention in encoder-decoder RNNs



Recap

Transformers

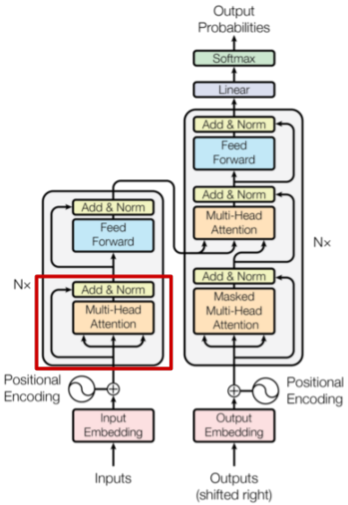
Transformers



Fully attention-based model for NLP

- Self-attention → contextualization
- Positional embeddings → word order
- Byte-pair encodings → open vocabulary
- Residual connections
- Multi-head attention

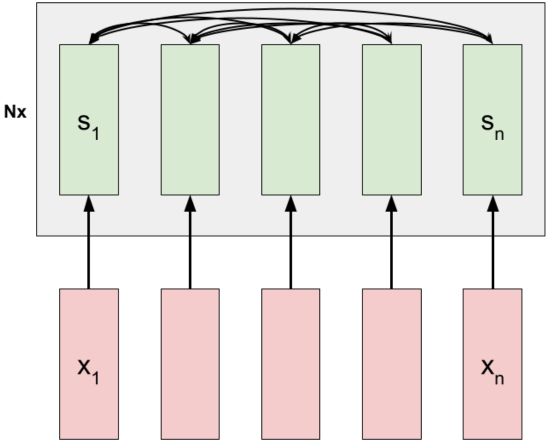
Transformers



Fully attention-based model for NLP

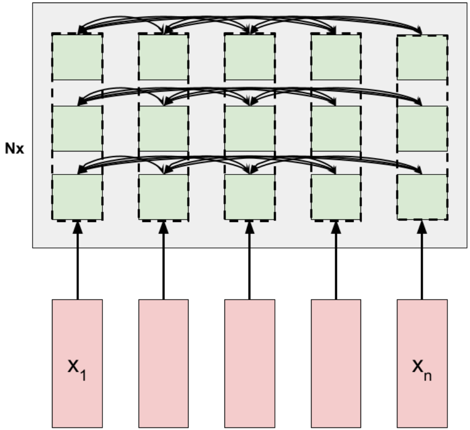
- Self-attention → contextualization
- Positional embeddings → word order
- Byte-pair encodings → open vocabulary
- Residual connections
- Multi-head attention

Attention in Transformers



- Each token representation **simultaneously** attends to *all other* tokens in the sequence
  - Except when we want to **mask out** some tokens
- What about **multi-head** attention?

Attention in Transformers



1. We **split** each token representation into h equally sized chunks
2. We **independently** perform scaled dot-product self-attention for each set of chunks
   - Intuition: multiple **aspects** of similarity between tokens

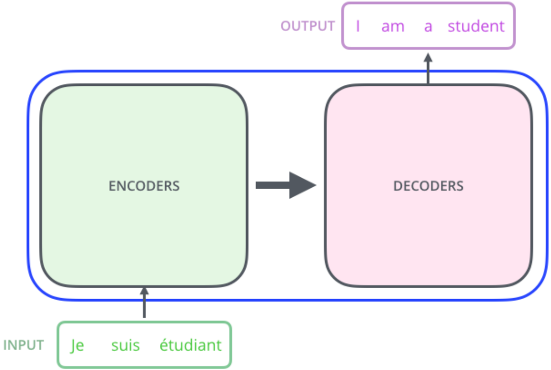
Transformer for machine translation

We studied the Transformer encoder-decoder for machine translation

Image source: The illustrated Transformer

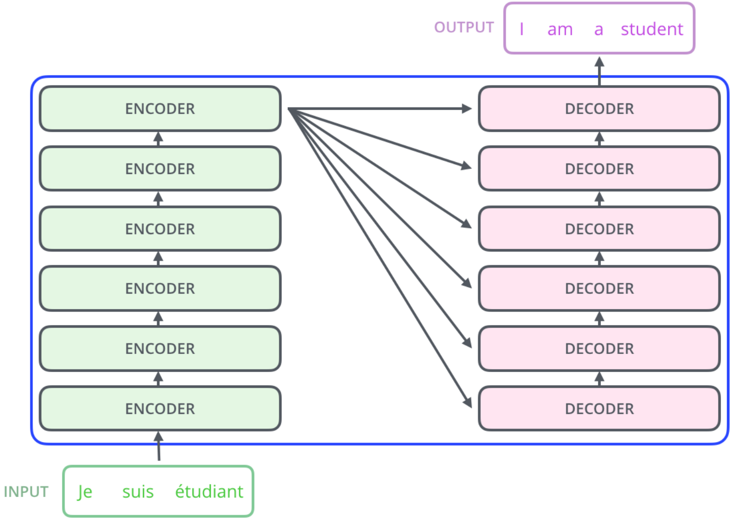
The model is built of stacked encoder and decoder blocks

Transformer encoder-decoder



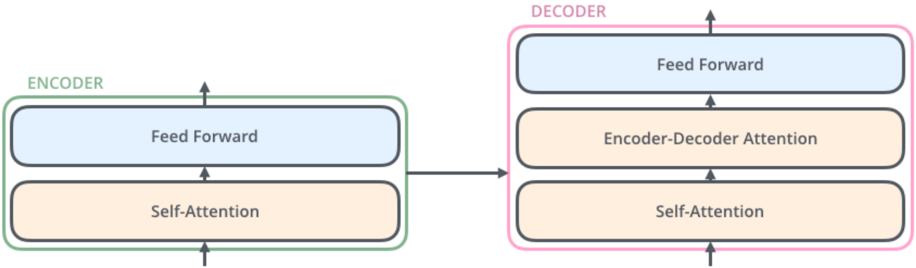
The encoder and decoder are made of stacked **encoder** and **decoder blocks**

Transformer encoder-decoder

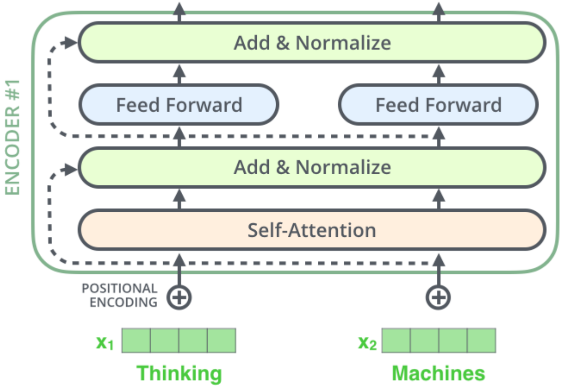


Transformer encoder and decoder block

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



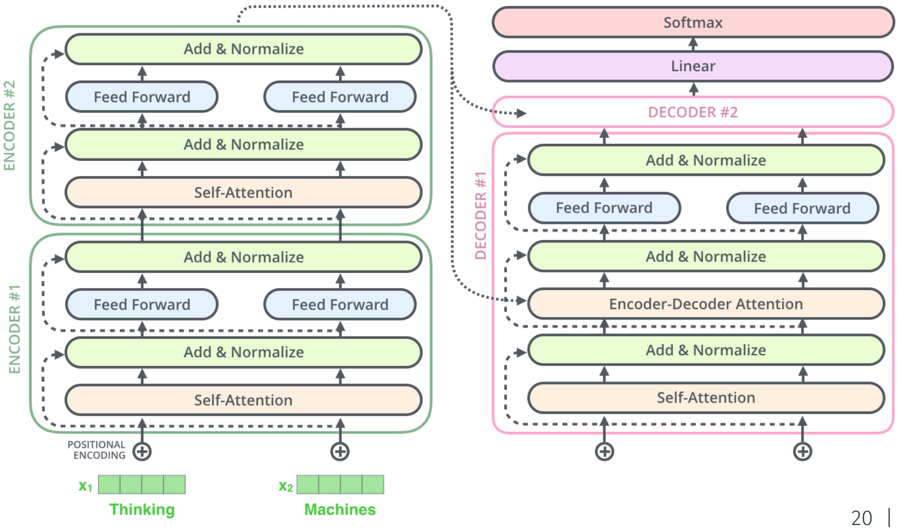
Transformer encoder block



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**

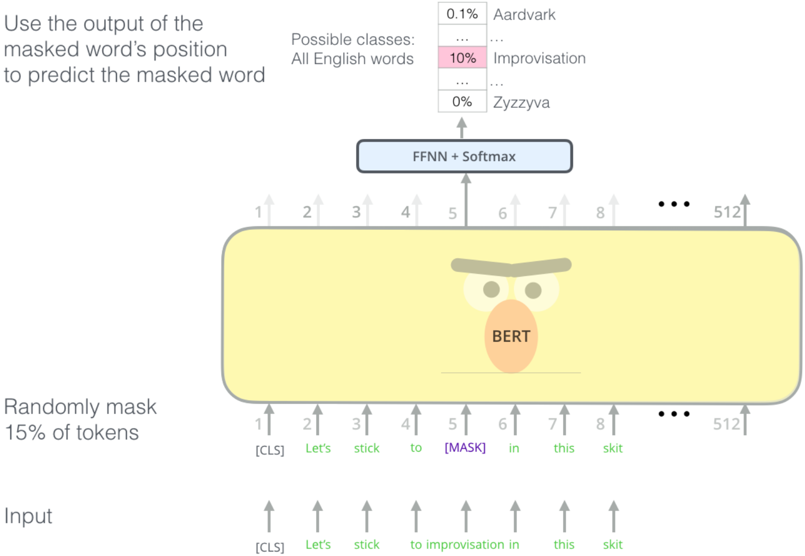
Recap: Transformer encoder-decoder



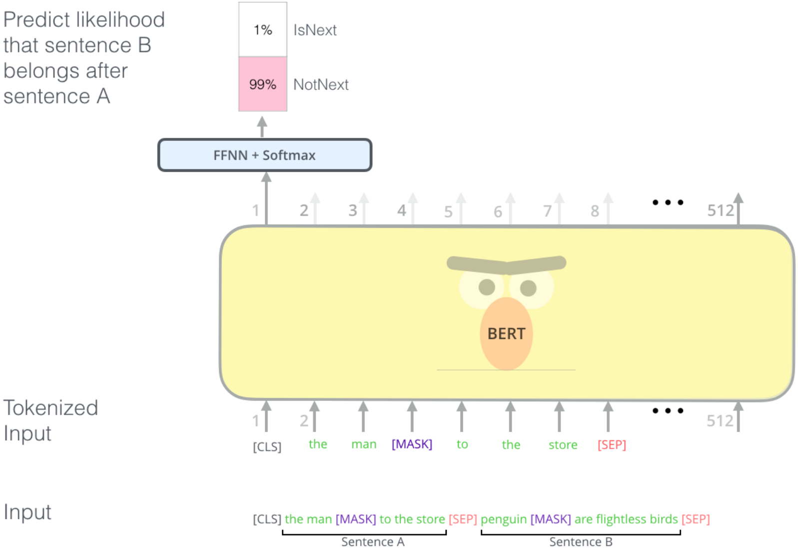
Recap

BERT

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



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



BERT: summary

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

- Through a (masked) language modeling pretraining task, the model has learned to recognize high level **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

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.

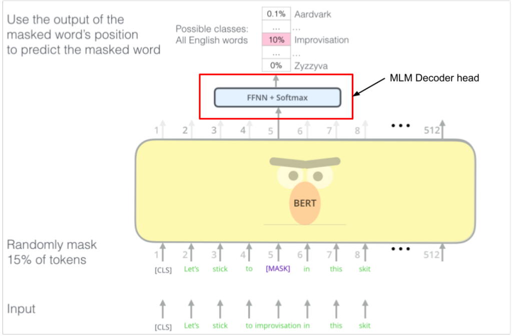
Fine-tuning variants:

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

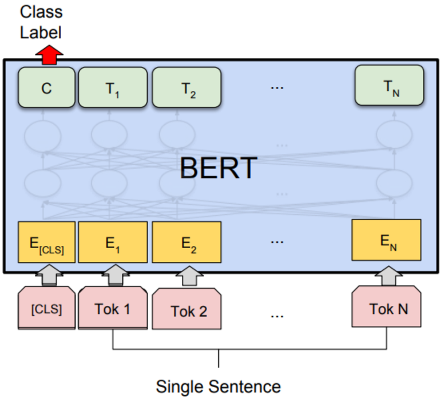


Image from BERT paper

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**

Recap

Pretraining tasks for PLMs

Variants of (unsupervised) pretraining tasks

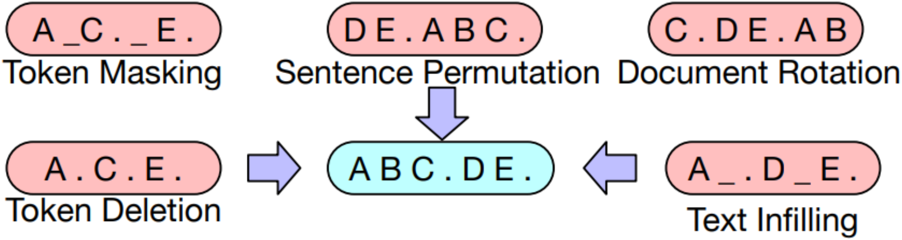


Image from BART paper

Variants of (unsupervised) 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

We can also use **supervised data** – from various datasets

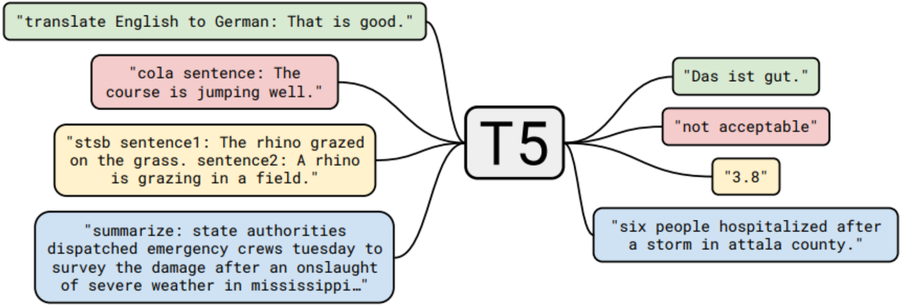
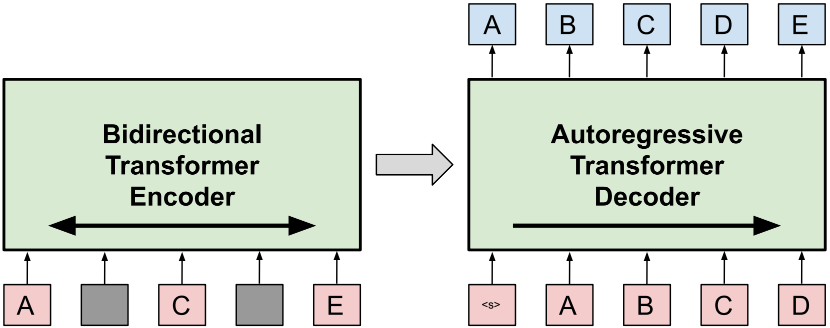


Image from T5 paper

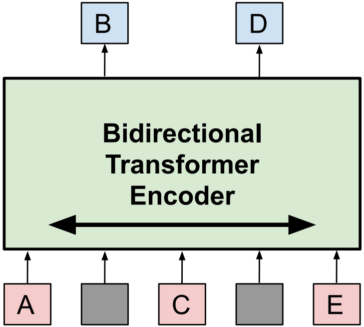
Recap

Types of Transformer Architectures

Encoder-Decoder Transformer

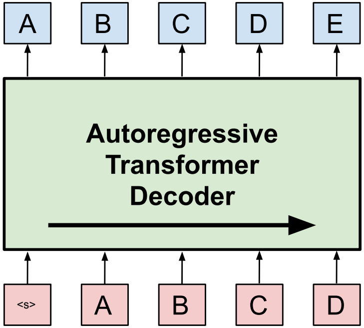


Bidirectional Encoder-only Transformer



- Efficient encoding ✔️
- Versatile base for downstream tasks ✔️
- Can’t **really** generate text ❌

Autoregressive Decoder-only Transformer



An **autoregressive** (causal) language model uses **past** values of a time series to predict future values.

- Didn't we decide not to use these because they were inefficient?

  (RNNs)

- Yes, but…
  1. Hardware has improved
  2. Autoregressive models are *really* good at generating text

Recap

Prompting

Prompt-tuning MLMs

So far, we have **fine-tuned** masked language models

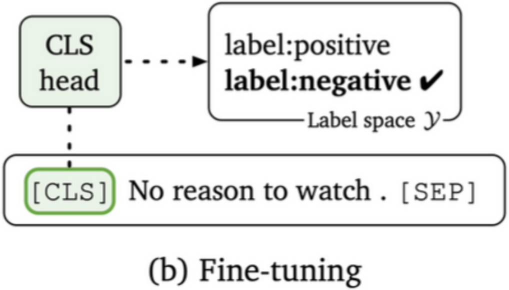
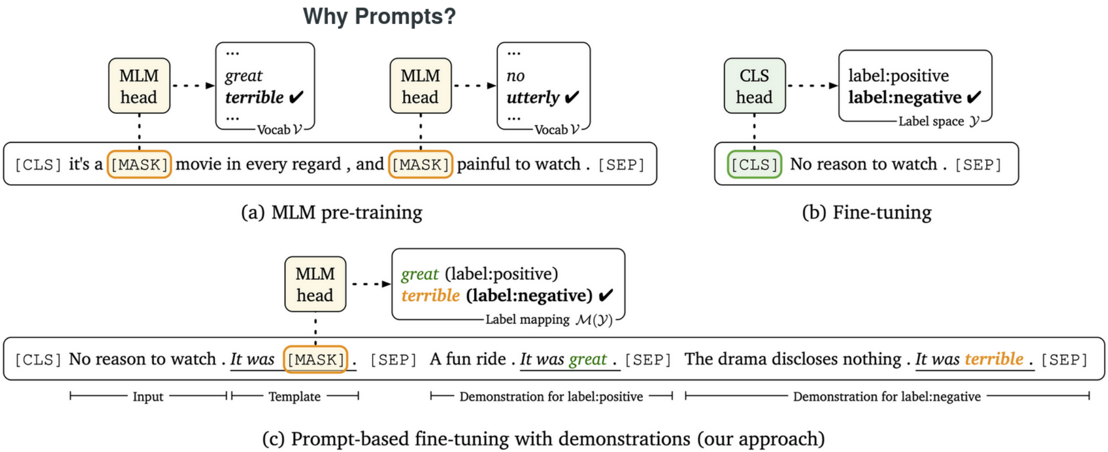


Figure from The Gradient

Is there an **easier way** to adapt encoder-only PLMs such as BERT to downstream tasks?

Prompt-tuning MLMs



Prompt-tuning MLMs

We transform the target task (e.g. sentiment analysis) to **masked language modeling**.

1. Choose the prompt and word/token used for each label
   - Choice of label token **important**
   - Template design also **important**
2. Demonstrate task through a few samples
   - Usually through **fine-tuning** (training the model)
3. **No new parameters needed** to perform task!

In-context learning

When we **don't fine-tune** the base model on prompt samples but provide **demonstrations** within the prompt, this is called **in-context learning**.

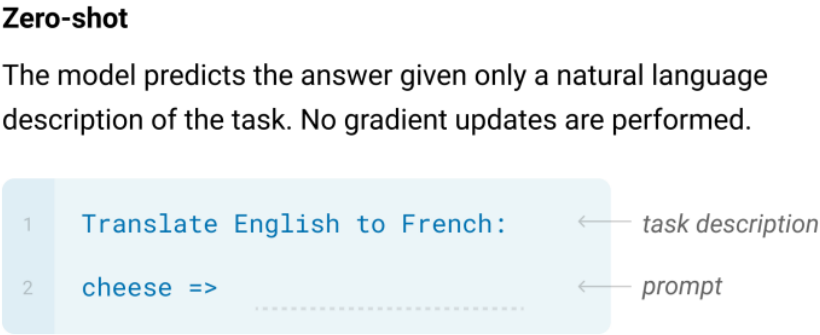


Image from GPT3 paper

Prompting

A prompt is a piece of text inserted in the input examples, so that the original task **can be formulated as** a (masked) **language modeling** problem.

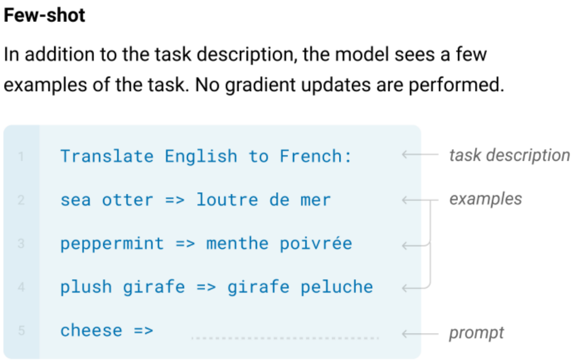


Image from GPT3 paper

License and credits

Credits

Martin Tutek

Content from ACL Anthology papers licensed under CC-BY `https://www.aclweb.org/anthology`