Raw Data

Masked Language Models and BERT

Cornell CS 5740: Natural Language Processing
Yoav Artzi, Spring 2023

Masked LMs

- LMs so far: predict the next token given the previous tokens
- This enables a self-supervised task
- That we can train on a lot of data
- And get really interesting and useful representations

Masked LMs

- LMs so far: predict the next token given the previous tokens
- What if we have a complete sentence?
  - Can do the same

Masked LMs

- LMs so far: predict the next token given the previous tokens
- What if we have a complete sentence?
  - Can do the same
  - Decode through an LM to compute representations
  - But: representations conditioned on past context only
  - So: missing an opportunity here to incorporate future context
- How can we formulate a self-supervised prediction task?

Masked LMs

- We have many sequence of tokens $\bar{x} = \langle x_1, \ldots, x_n \rangle$
  - Just raw data, like with regular LMs
- Let's create a prediction task by hiding part of the sequence, and then trying to predict them
  - Input: the sequence $\bar{x}^M$ where some tokens are replaced with the token $[\mathrm{MASK}]$, for example: $\bar{x}^M = \langle x_1, \ldots, x_4, [\mathrm{MASK}], x_6, \ldots, x_n \rangle$
  - Output: a probability distribution over tokens for each masked position such that the correct token gets the highest probability, for example $\arg\max_{\mathscr{V}} p(x_7^M \mid \bar{x}^M) = x_7$
  - Training objective: negative log-likelihood for masked tokens

Masked LMs

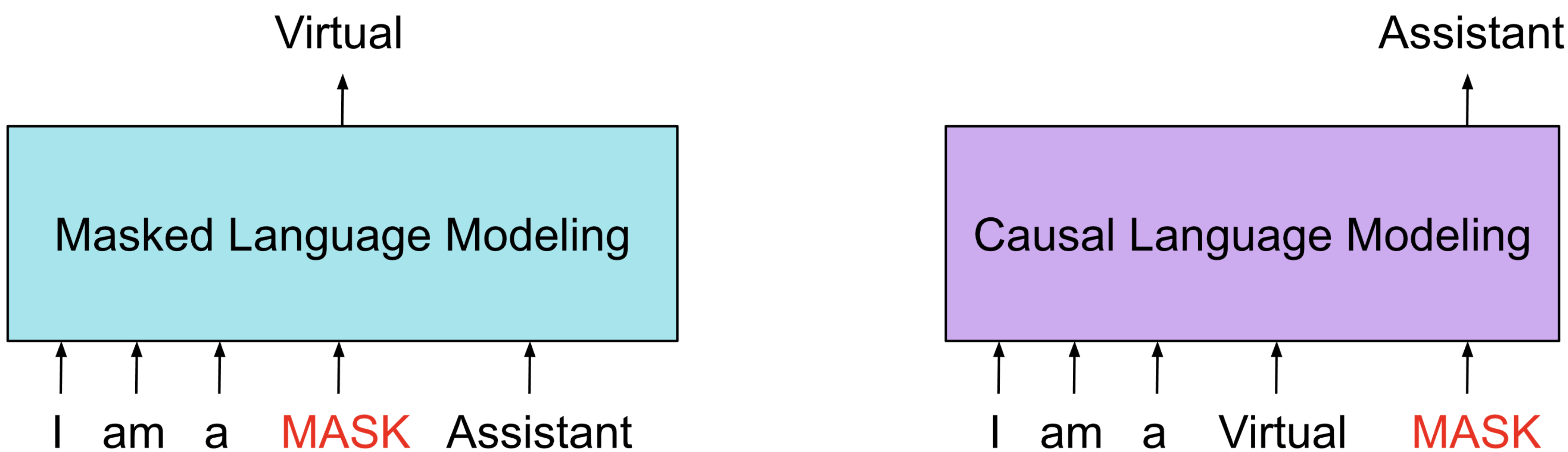


Image from https://www.holisticai.com/blog/from-transformer-architecture-to-prompt-engineering

Encoder Transformer

- So far: Transformers self-attend to past tokens to predict the next token
- This is called a Decoder Transformer
- Encoders assume we have the complete sequence
- There is no generation problem, we just want representations
    - We will learn how to use them later on
- The big difference: self-attention is not masked, so computes weighted sum over entire context (i.e., entire sequence)

The Transformer

Decoder-only Variant (revisit)

$$\text{TransformerBlock}^{\text{k}}(\mathbf{u}_1, \ldots, \mathbf{u}_{\text{i}})$$

$$\mathbf{q}^{(l)} = \mathbf{W}_q^{(l)}\mathbf{u}_i$$

$$\mathbf{K}^{(l)} = \mathbf{W}_k^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_i]$$

$$\mathbf{V}^{(l)} = \mathbf{W}_v^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_i]$$

$$\mathbf{z} = \text{LN}([\text{SelfAttn}(\mathbf{q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}); \cdots; \text{SelfAttn}(\mathbf{q}^{(L)}, \mathbf{K}^{(L)}, \mathbf{V}^{(L)})] + \mathbf{u}_i)$$

$$\mathbf{h}_i^k = \text{LN}(\mathbf{W}''\text{GELU}(\mathbf{W}'\mathbf{z} + \mathbf{b}') + \mathbf{b}'' + \mathbf{z})$$

$$\mathbf{x}_i = \phi(x_i) + \phi_p(i)$$

$$\mathbf{h}_i^1 = \text{TransformerBlock}^1(\mathbf{x}_1, \ldots, \mathbf{x}_{\text{i}})$$

$$\mathbf{h}_i^2 = \text{TransformerBlock}^2(\mathbf{h}_1^1, \ldots, \mathbf{h}_{\text{i}}^1)$$

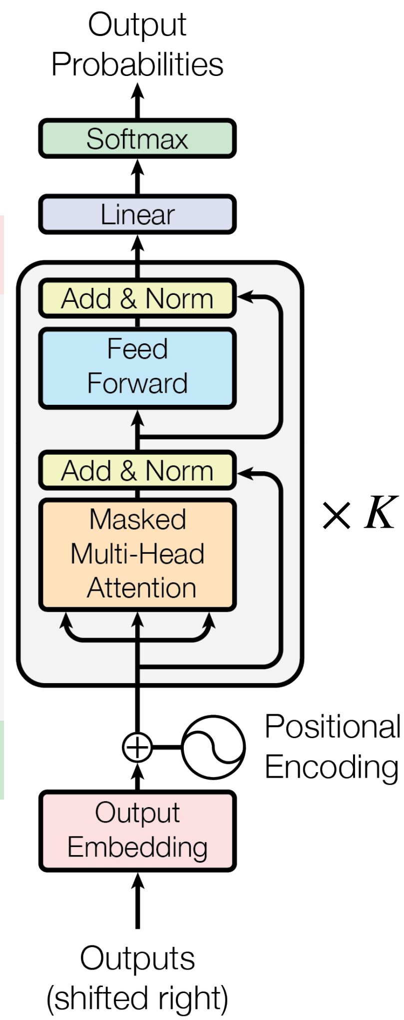
$$\ldots$$

$$\mathbf{h}_i^k = \text{TransformerBlock}^{\text{k}}(\mathbf{h}_1^{\text{k}-1}, \ldots, \mathbf{h}_{\text{i}}^{\text{k}-1})$$

$$\ldots$$

$$\mathbf{h}_i^K = \text{TransformerBlock}^{\text{K}}(\mathbf{h}_1^{\text{K}-1}, \ldots, \mathbf{h}_{\text{i}}^{\text{K}-1})$$

$$p(x_{i+1} \mid x_1, \ldots, x_i) = \text{softmax}(\mathbf{W}^{\mathcal{V}}\mathbf{h}_i^K)$$



Self-attention reminder

$$\text{SelfAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{QK}/\sqrt{d_k})\mathbf{V}$$

During learning, compute the whole sequence at ones by **masking** items you shouldn't attend to in softmax — easy by setting softmax to $-\infty$

[Vaswani et al. 2017]

Encoder Transformer*

$$\text{TransformerBlock}^{k}(\mathbf{u}_1, \ldots, \mathbf{u}_n)$$

$$\mathbf{Q}^{(l)} = \mathbf{W}_q^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_n]$$

$$\mathbf{K}^{(l)} = \mathbf{W}_k^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_n]$$

$$\mathbf{V}^{(l)} = \mathbf{W}_v^{(l)}[\mathbf{u}_1 \cdots \mathbf{u}_n]$$

$$\mathbf{Z} = \text{LN}([\text{SelfAttn}(\mathbf{Q}^{(1)}, \mathbf{K}^{(1)}, \mathbf{V}^{(1)}); \cdots; \text{SelfAttn}(\mathbf{Q}^{(L)}, \mathbf{K}^{(L)}, \mathbf{V}^{(L)})] + [\mathbf{u}_1 \cdots \mathbf{u}_n])$$

$$[\mathbf{h}_1^k \cdots \mathbf{h}_n^k] = \text{LN}(\mathbf{W}''\text{GELU}(\mathbf{W}'\mathbf{Z} + \mathbf{b}') + \mathbf{b}'' + \mathbf{Z})$$

$$\mathbf{x}_i = \phi(x_i^M) + \phi_p(i), i = 1, \ldots, n$$

$$[\mathbf{h}_1^1 \cdots \mathbf{h}_n^1] = \text{TransformerBlock}^1(\mathbf{x}_1, \ldots, \mathbf{x}_n)$$

$$[\mathbf{h}_1^2 \cdots \mathbf{h}_n^2] = \text{TransformerBlock}^2(\mathbf{h}_1^1, \ldots, \mathbf{h}_n^1)$$

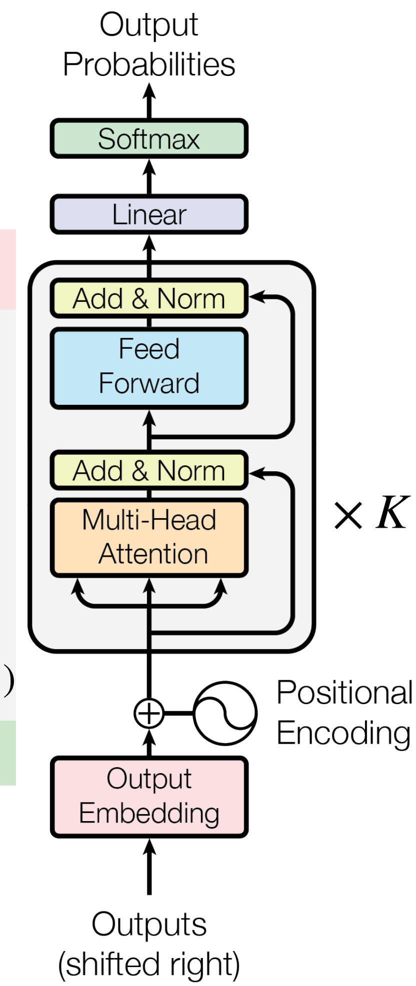
$$\ldots$$

$$[\mathbf{h}_1^k \cdots \mathbf{h}_n^k] = \text{TransformerBlock}^k(\mathbf{h}_1^{k-1}, \ldots, \mathbf{h}_n^{k-1})$$

$$\ldots$$

$$[\mathbf{h}_1^K \cdots \mathbf{h}_n^K] = \text{TransformerBlock}^K(\mathbf{h}_1^{K-1}, \ldots, \mathbf{h}_n^{K-1})$$

$$p(x_i \mid x_1^M, \ldots, x_n^M) = \text{softmax}(\mathbf{W}^{\mathcal{V}} \mathbf{h}_i^K)$$



Self-attention reminder

$$\text{SelfAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}/\sqrt{d_k})\mathbf{V}$$

* for Masked LM

[Vaswani et al. 2017]

BERT

Bidirectional Encoder Representations from Transformers

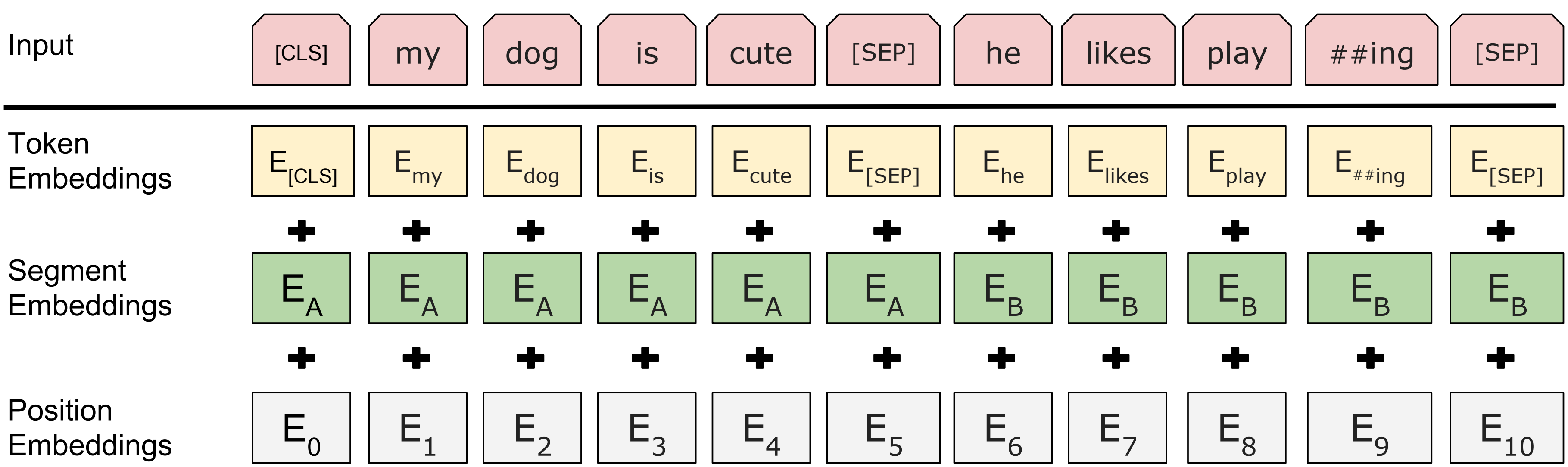
- Encoder transformer
- BERT Base: 12 transformer blocks, 768-dim word-piece tokens, 12 self-attention heads → 110M parameters
- BERT Large: 24 transformer blocks, 1024-dim word-piece tokens, 16 self-attention heads → 340M parameters
- RoBERTa: same model, much more data (160GB of data instead of 16GB)

[Devlin et al. 2018]

BERT

Inputs

- One or two sentences
  - Word-piece token embeddings
  - Position and segment embeddings



[figure from Devlin et al. 2018]

BERT

Word-piece Tokenization (in a nutshell)

1. Initialize with tokens for all characters
2. While vocabulary size is below the target size:
    1. Build a language model over the corpus (e.g., unigram language model)
    2. Merge pieces that lead to highest improvement in language model perplexity

- Need to choose a language model that will make the process tractable
- Often a unigram language model (e.g., SentencePiece library)
- Particularly suitable for machine translation

[Schuster and Nakajima (2012), Wu et al. (2016), Kudo and Richardson (2018)]

BERT

Training

- Data: raw text
- Two objectives:
  - Masked LM
  - Next-sentence prediction
- Later development in RoBERTa:
  - More data, no next-sentence prediction, dynamic masking

BERT

Masking Recipe for Training

- Mask and predict 15% of the tokens
  - For 80% (of 15%) replace with the input token with $[\mathrm{MASK}]$
  - For 10%, replace with a random token
  - For 10%, keep the same

BERT

Next-sentence Prediction

- Input: [CLS] Text chunk 1 [SEP] Text chunk 2
- Training data: 50% of the time, take the true next chunk of text, 50% of the time take a random other chunk
- Predict whether the next chunk is the true next chunk
- Prediction is done on the [CLS] output representation

BERT

Related Techniques

- Central Word Prediction Objective (context2vec) [Melamud et al. 2016]
- Machine Translation Objective (CoVe) [McMann et al. 2017]
- Bi-directional Language Modeling Objective (ELMo) [Peters et al. 2018]
- Then BERT came …
- … and many more followed

BERT

What Do We Get?

- We can feed complete sentences to BERT
- For each token, we get a contextualized representation
  - Meaning: computed taking the other tokens in the sentence into acocunt
- In contrast to word2vec representations that fixed and do not depend on context
- While word2vec vectors are forced to mix multiple senses, BERT can provide more instance-specific vectors

BERT

How Do We Use It?

- Widely supported by existing frameworks
  - E.g., Transformers library by Hugging Face
- We will soon see how to use it when working with annotated data
- Large BERT models quickly outperformed human performance on several NLP tasks
  - But what it meant beyond benchmarking was less clear
- Started an arms race towards bigger and bigger models, which quickly led to the LLMs of today

BERT

What It Is Not Great For?

- BERT cannot generate text (at least not in an obvious way)
    - Not an autoregressive model, can do weird things like stick a $[\mathrm{MASK}]$at the end of a string, fill in the mask, and repeat
- Masked language models are intended to be used primarily for "analysis" tasks

BERT

What does BERT Learn?

- There is a lot of work trying to decipher what BERT learns in its representations
    - Much harder with recent LLMs because they are not as open
- Some very interesting results, but not completely clear how to interpret them

What Does BERT Learn?

- Try to solve different linguistic tasks given each block level, without fine-tuning
  - Specifically: solve tasks using mixing weights on levels
- Goal: see what information each new level adds
- Each task classifier takes a single mixed hidden representation $\mathbf{h}_{i,\tau}$ or a pair of representations for two tokens

i : token index

K : number of block levels

τ : task

γ_τ : task parameter

$\mathbf{a}_\tau$: mixing parameters

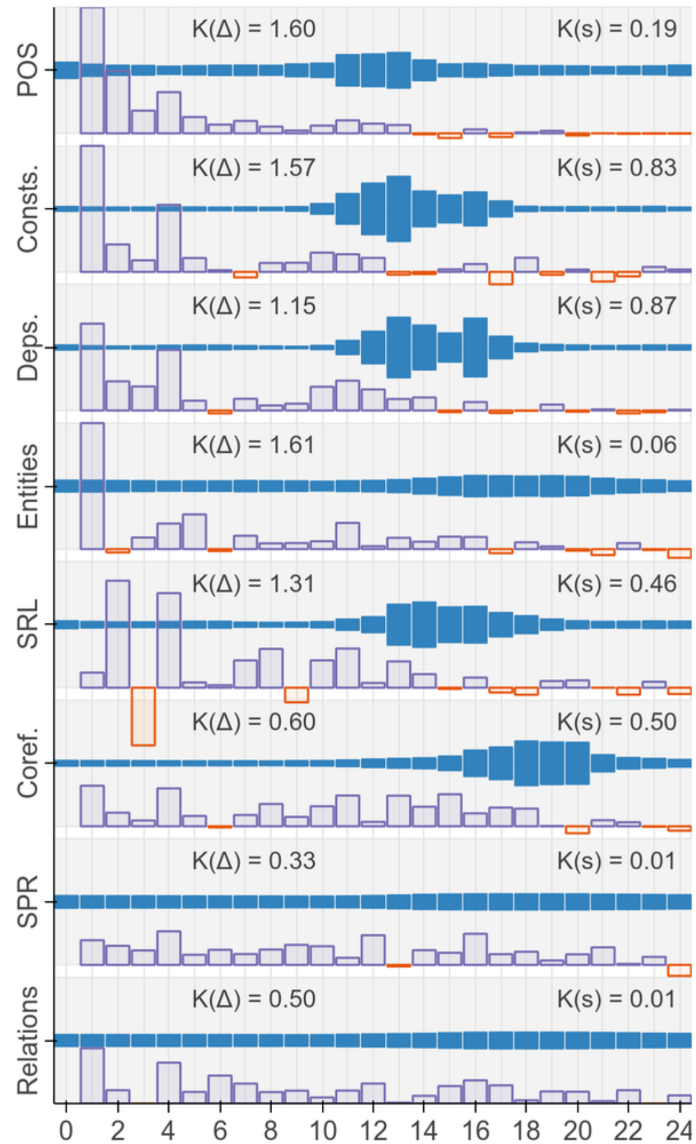
$$\mathbf{s}_\tau = \text{softmax}(\mathbf{a}_\tau)$$

$$\mathbf{h}_{i,\tau} = \gamma_\tau \sum_{k=0}^{K} s_\tau^k \mathbf{h}_i^k$$

[Tenney et al. 2019]

What Does BERT Learn?

- Each plot shows a task
- Plots show $\mathbf{s}_\tau^k$ weights magnitude in blue, and the number of self-attention levels
- The performance delta when adding this layer is in purple
- Largely: higher level semantic tasks happen in later levels



[figure from Tenney et al. (2019)]

Acknowledgements

- Some slides in this deck were adapted from Greg Durrett