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 [MASK], for example: $\bar{x}^M = \langle x_1, \ldots, x_4, [\text{MASK}], x_6, \ldots, x_n \rangle$
  - Output: a probability distribution over tokens for each masked position such that the correct token gets the highest probability, for example $\arg \max_{\forall} p(x_7^M | \bar{x}^M) = x_7$
  - Training objective: negative log-likelihood for masked tokens
Masked LMs

Masked Language Modeling

Causal Language Modeling

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)

### TransformerBlock\(^k(u_1, \ldots, u_i)\)

\[
q^{(l)} = W^{(l)}_q u_i \\
K^{(l)} = W^{(l)}_k [u_1 \cdots u_i] \\
V^{(l)} = W^{(l)}_v [u_1 \cdots u_i] \\
z = \text{LN}([\text{SelfAttn}(q^{(1)}, K^{(1)}, V^{(1)}); \cdots; \text{SelfAttn}(q^{(L)}, K^{(L)}, V^{(L)})] + u_i) \\
h_i^k = \text{LN}(W''\text{GELU}(W'z + b') + b'' + z)
\]

<table>
<thead>
<tr>
<th>(x_i = \phi(x_i) + \phi_p(i))</th>
<th>(h_i^1 = \text{TransformerBlock}^1(x_1, \ldots, x_i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(h_i^2 = \text{TransformerBlock}^2(h_i^1, \ldots, h_i^1))</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(h_i^k = \text{TransformerBlock}^k(h_i^{k-1}, \ldots, h_i^{k-1}))</td>
<td>(\ldots)</td>
</tr>
<tr>
<td>(h_i^K = \text{TransformerBlock}^K(h_i^{K-1}, \ldots, h_i^{K-1}))</td>
<td></td>
</tr>
</tbody>
</table>

\[
p(x_{i+1} | x_1, \ldots, x_i) = \text{softmax}(W^T h_i^K)
\]

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

---

Self-attention reminder

\[
\text{SelfAttn}(Q, K, V) = \text{softmax}(QK/\sqrt{d_k})V
\]
Encoder Transformer*

TransformerBlock^k(u_1, ..., u_n)

Q^{(l)} = W_q^{(l)}[u_1 \ldots u_n]
K^{(l)} = W_k^{(l)}[u_1 \ldots u_n]
V^{(l)} = W_v^{(l)}[u_1 \ldots u_n]
Z = LN([SelfAttn(Q^{(1)}, K^{(1)}, V^{(1)}); \ldots; SelfAttn(Q^{(L)}, K^{(L)}, V^{(L)})] + [u_1 \ldots u_n])

[h_1^k \ldots h_n^k] = LN(W''GELU(W'Z + b') + b'' + Z)

Self-attention reminder

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

x_i = \phi(x_i^M) + \phi_p(i), i = 1, \ldots, n

[h_1^1 \ldots h_n^1] = TransformerBlock^1(x_1, \ldots, x_n)
[h_1^2 \ldots h_n^2] = TransformerBlock^2(h_1^1, \ldots, h_n^1)

\ldots

[h_1^K \ldots h_n^K] = TransformerBlock^K(h_1^{K-1}, \ldots, h_n^{K-1})

\ldots

p(x_i | x_1^M, \ldots, x_n^M) = softmax(W'\mathbf{h}_i^K)

* 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 $\rightarrow$ 110M parameters

• BERT Large: 24 transformer blocks, 1024-dim word-piece tokens, 16 self-attention heads $\rightarrow$ 340M parameters

• RoBERTa: same model, much more data (160GB of data instead of 16GB)
• One or two sentences
  - Word-piece token embeddings
  - Position and segment embeddings
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 [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 account
- 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 [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 $h_{i,\tau}$ or a pair of representations for two tokens

\[
i : \text{token index} \\
K : \text{number of block levels} \\
\tau : \text{task} \\
\gamma_{\tau} : \text{task parameter} \\
a_{\tau} : \text{mixing parameters} \\
\]

\[
s_{\tau} = \text{softmax}(a_{\tau})
\]

\[
h_{i,\tau} = \gamma_{\tau} \sum_{k=0}^{K} s_{\tau}^{k} h_{i}^{k}
\]
What Does BERT Learn?

- Each plot shows a task
- Plots show $s^k_\tau$ 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
Acknowledgements

• Some slides in this deck were adapted from Greg Durrett