Contextualized Text Representations

Zihang Dai
April 18, 2019
Outline

- From word embedding to *contextualized* representation
- ELMo (best paper NAACL 2018) [4]
- BERT (best paper NAACL 2019) [1]
• From word embedding to contextualized representation
  • ELMo
  • BERT
Basic idea: “A word is characterized by the company it keeps.” [2].

⇒ The representation of a word is decided by the “surrounding context” that it appears in.

Computationally,

- A word can be *predicted by* its context ⇒ CBOW [3]
- A word is able to *predict* its context ⇒ Skip-Gram [3]
Mathematically, given a sequence of \(2w\) words \(s = [x_{i-w}, \ldots, x_{i+w-1}]\)

- **CBOW** (a word can be *predicted by* its context)

\[
\max_{\theta} \log P_{\theta}(x_i \mid \underbrace{x_{i-w:i}, x_{i+1:i+w}}_{\text{context } c_i}) = \log \frac{\exp \left( e_{\theta}(x_i)^\top g_{\theta}(c_i) \right)}{\sum_{x' \in \mathcal{V}} \exp \left( e_{\theta}(x')^\top g_{\theta}(c_i) \right)}
\]

where \(\mathcal{V}\) denotes the entire vocabulary and

\[
e_{\theta}(x_i) = \text{word embedding of } x_i
\]

\[
g_{\theta}(c_i) = \frac{1}{2w - 1} \sum_{j \neq i, x_j \in s} e'_{\theta}(x_j) \rightarrow \text{average of context embeddings}
\]

- **Skip-Gram** (a word is able to *predict* its context)

\[
\max_{\theta} \frac{1}{2w - 1} \sum_{j \neq i, x_j \in s} \log P_{\theta}(x_j \mid x_i)
\]
Pros & Cons of Word Embedding

Pros: unsupervised training on a huge corpus

Cons: The representation (embedding) of a word is always the same regardless of the specific context.

A classic example:

- I took a walk by the river bank after dinner.
- I went to the PNC bank this morning.
Contextualized Representation

A motivating **example** of text classification.

Given a sentence $x$, we want to classify it to some category $y$:

- A deep neural model (e.g. LSTM) is stacked upon the word embeddings to produce a sequence of hidden states
  \[ [h_1, h_2, \cdots, h_T] = \text{LSTM}(e(x_1), e(x_2), \cdots, e(x_T)) \]

- Take the averaged hidden states to create the classifier:
  \[ h_{\text{avg}} = \text{Average}([h_1, h_2, \cdots, h_T]) \]

  \[ P(Y = y \mid x) = \frac{\exp \left( W_y^\top h_{\text{avg}} \right)}{\sum_{y' \in Y} \exp \left( W_{y'}^\top h_{\text{avg}} \right)} \]

- Effectively, $h_t$ can be seen as the representation of $x_t$ in the context $x$
  \[ \implies h_t = \text{contextualized representation of } x_t \]

**Problem**: training relies on supervised/labeled data (expensive)
Unsupervised Contextualized Representation

Key Desiderata:

- Unsupervised (does not rely on labeled data)
- Efficient (can scale to a lot of data)
- Transferable (good performance for downstream task)

Two Notable Approaches:

- ELMo: LSTM + language modeling (LM)
- BERT: Transformer + masked language modeling (MLM) + next sentence prediction (NSP)
Outline

- From word embedding to contextualized representation
- ELMo
- BERT
Given a document (sequence) of $T$ words $x = [x_1, x_2, \cdots, x_T]$, model the data distribution $P(x)$.

Standard formulation based on the auto-regressive factorization:

$$P(x) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$$

Employ an LSTM to encode the context and construct the next-word distribution

$$h_{t-1} = \text{LSTM}_\theta(x_{<t})[-1] \quad \leftarrow \text{last hidden state}$$

$$P_\theta(x_t \mid x_{<t}) = \frac{\exp\left(\mathbf{e}_\theta(x_t)^\top h_{t-1}\right)}{\sum_{x' \in \mathcal{V}} \exp\left(\mathbf{e}_\theta(x')^\top h_{t-1}\right)} \quad \leftarrow \text{next-word distribution}$$
Training: maximum likelihood estimation

$$\max_\theta \log P_\theta(x) = \sum_{t=1}^{T} \log P_\theta(x_t \mid x_{<t})$$

- The objective is differentiable w.r.t. all model parameters
- Stochastic gradient ascent (maximization)

Notes:

- In theory, $h_{t-1}$ can capture / compress the entire history $x_{<t}$ given a large LSTM
- Language modeling is an *unsupervised* objective
**Key Idea:** use the hidden state of an LSTM unsupervised pretrained by language modeling as the contextualized presentation

**Problem:** the hidden state $h_t$ only captures the history information up to $x_t$, not the future information

**Solution:**

- Use two separate LSTMs, one from left to right and the other from right to left
- Concatenate the hidden states from the two LSTMs
ELMo: Merge Two LSTMs

Left-to-right factorization:

\[
P^\rightarrow_{\theta}(x) = \prod_{t=1}^{T} P^\rightarrow_{\theta}(x_t \mid x_{<t}) = \prod_{t=1}^{T} P^\rightarrow_{\theta}(x_t \mid \text{LSTM}^\rightarrow_{\theta}(x_{<t})_{h_{t-1}})
\]

Right-to-left factorization:

\[
P^\leftarrow_{\phi}(x) = \prod_{t=1}^{T} P^\leftarrow_{\phi}(x_t \mid x_{>t}) = \prod_{t=1}^{T} P^\leftarrow_{\phi}(x_t \mid \text{LSTM}^\leftarrow_{\phi}(x_{>t})_{\hat{h}_{t+1}})
\]

Concatenate:

\[
h_t = \left[\hat{h}^\rightarrow_t, \hat{h}^\leftarrow_t\right]
\]
Given a multi-layer LSTM, each layer has a sequence of hidden states, i.e., for \( m = 1, \ldots, M \)

\[
\begin{bmatrix}
    h_1^{(0)}, h_2^{(0)}, \ldots, h_T^{(0)} \\
    h_1^{(m)}, h_2^{(m)}, \ldots, h_T^{(m)}
\end{bmatrix} = \begin{bmatrix} e(x_1), e(x_2), \ldots, e(x_T) \\
    \text{LSTM}^{(m)}\left( \begin{bmatrix}
        h_1^{(m-1)}, h_2^{(m-1)}, \ldots, h_T^{(m-1)}
    \end{bmatrix}\right)\end{bmatrix}
\]

Then, ELMo relies on a trainable weighted average to obtain the final contextualized representation:

\[
h_t = \sum_{m=1}^{M} \alpha_m h_t^{(m)}, \quad \text{s.t.} \quad \sum_{m=1}^{M} \alpha_m = 1,
\]

where \( \{\alpha_m\}_{m=1}^{M} \) are parameters finetuned on the downstream task.
ELMo: Empirical Performance

6 state-of-the-art performances across 6 different NLP tasks

<table>
<thead>
<tr>
<th>Model</th>
<th>SQuAD</th>
<th>SNLI</th>
<th>SRL</th>
<th>Coref</th>
<th>NER</th>
<th>SST-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous SOTA</td>
<td>84.4</td>
<td>88.6</td>
<td>81.7</td>
<td>67.2</td>
<td>91.93 ± 0.19</td>
<td>53.7</td>
</tr>
<tr>
<td>Baseline</td>
<td>81.1</td>
<td>88.0</td>
<td>81.4</td>
<td>67.2</td>
<td>90.15</td>
<td>62.9</td>
</tr>
<tr>
<td>Baseline + ELMo</td>
<td>85.8</td>
<td>88.7 ± 0.17</td>
<td>84.6</td>
<td>70.4</td>
<td>92.22 ± 0.10</td>
<td>54.7 ± 0.5</td>
</tr>
<tr>
<td>Improve on Baseline</td>
<td>4.7</td>
<td>0.7</td>
<td>3.2</td>
<td>3.2</td>
<td>2.06</td>
<td>3.3</td>
</tr>
<tr>
<td>Improve on SOTA</td>
<td>1.4</td>
<td>0.1</td>
<td>2.9</td>
<td>3.2</td>
<td>0.29</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 1: At the time of publication (January 2018).

- Pre-trained on 1 billion tokens with GPUs
- The LSTM model is not very large (but definitely not small)
Outline

- From word embedding to contextualized representation
- ELMo
- BERT
A Problem of ELMo

Recall that

\[ h_t = \left[ \overset{\rightarrow}{h}_t(x_{\leq t}; \theta^*), \overset{\leftarrow}{h}_t(x_{\geq t}; \phi^*) \right] \]

where

\[ \theta^* = \arg\max_{\theta} \mathbb{E}_{x \sim \text{Data}} \left[ \sum_{t=1}^{\left| x \right|} \log P_{\theta}(x_t \mid x_{< t}) \right] \]

\[ \phi^* = \arg\max_{\phi} \mathbb{E}_{x \sim \text{Data}} \left[ \sum_{t=\left| x \right|}^{1} \log \overset{\leftarrow}{P}_{\phi}(x_t \mid x_{> t}) \right] \]

Notes:

- \( \theta^* \) and \( \phi^* \) are separately optimized
- \( \overset{\rightarrow}{h}_t(x_{\leq t}; \theta^*) \) and \( \overset{\leftarrow}{h}_t(x_{\geq t}; \phi^*) \) do NOT know how to "collaborate" to create synergy
- Local optimal v.s. Global optimal
**Joint Bidirectional Optimization**

**One idea:** masked language modeling (i.e. predict the masked words)

```
I went to the PNC ? this morning.
```

![Bank](#)

Formally, for a sequence $\mathbf{x}$, we sample a position $t$ to mask and optimize

$$
\max_{\theta} \log P_{\theta}(x_t \mid x_{\neq t}) = \log P_{\theta}(x_t \mid h_t(x_{\neq t}; \theta))
$$

**Notes:**

- $h_t(x_{\neq t}; \theta)$ is computed by a deep neural model (we’ll see later)
- $h_t(x_{\neq t}; \theta)$ has access both $x_{<t}$ and $x_{>t}$ at the same time $\implies$ joint bidirectional optimization.
- After training, we can use $h_t$ as the contextualized representation
**Efficiency Problem:** for each sampled position \( t \), \( h_t(x_{\neq t}; \theta) \) has to be recomputed since \( x_{\neq t} \) is different

**A practical solution:**

- **Raw input** \( x \): \( x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8 \ x_9 \ x_{10} \)
- **Masked input** \( \hat{x} \): \( x_1 \ x_2 \ [\text{mask}] \ x_4 \ x_5 \ x_6 \ [\text{mask}] \ x_8 \ [\text{mask}] \ x_{10} \)
- **Hidden states** \( h \): \( h_1 \ h_2 \ h_3 \ h_4 \ h_5 \ h_6 \ h_7 \ h_8 \ h_9 \ h_{10} \)

- **Target to pred.**:
  - \( x_3 \)
  - \( x_7 \)
  - \( x_9 \)

- **Mask multiple positions** for each sequence \( \implies \) **Masked input** \( \hat{x} \)

- **Share the same** \( \hat{x} \) to reconstruct original input:

\[
\max_{\theta} \sum_{t \in T_{\text{mask}}} \log P_\theta \left( x_t \mid h(\hat{x}; \theta)[t] \right)
\]

where \( T_{\text{mask}} \) is the set of masked positions.

\[19/30\]
(1) Mask 15% of tokens in each sequence

- cannot be too large $\implies$ accuracy
- cannot be too small $\implies$ efficiency

(2) Use Transformer [5] as the neural model

$$h = [h_1, \ldots, h_T] = \text{Transformer}(\hat{x}; \theta)$$

(3) An additional objective: next-sentence prediction

(4) Finetune the entire model for downstream task
Transformer is a neural network that relies on “self-attention”.

**Self-attention**: collects information based on pairwise similarity:

\[
\forall i = 1, \ldots, T : \quad h_i^{(m)} = \sum_{j=1}^{T} \frac{\exp(s_{i,j})}{\sum_{k=1}^{T} \exp(s_{i,k})} \left( W h_j^{(m-1)} \right)
\]

where

- \( s_{i,j} \) is the similarity score between \( h_i^{(m-1)} \) and \( h_j^{(m-1)} \)
- \( W \) is a weight matrix (linear projection)
**Observation:** Direct connection between each pair of nodes $\implies$ Self-attention is significantly easier to optimize than RNNs

**Facts:**
- Transformer = self-attention + feed-forward layers (non-linearity)
- Transformer is easier to optimize
- Transformer has a better performance given large-scale data
Motivations

(1) Contextualized representation is often used for downstream classification tasks

- Question: How can we extract a single vector to represent a sentence for classification?

(2) In practice, we often need to deal with a pair of sequences

- Examples: question answering, similarity of two sentences, relation of two sentences (contradiction, supporting, etc.)
(1) **Auxiliary task**: classify whether two sequences are consecutive ones

- Positive pairs: sample consecutive sequences from documents
- Negative pairs: sample two random sequences

(2) **[CLS] symbol**: for a pair of sequences \((a, b)\) with label \(y\) (neg / pos)

Merged input \(x\):  

\[
\text{[CLS]} \ a_1 \ldots \ a_{|a|} \ \text{[SEP]} \ b_1 \ldots \ b_{|b|}
\]

Hidden states \(h\):

\[
\begin{align*}
h_{\text{CLS}} & \quad h_{a_1} \\ & \quad \vdots \\ \text{Next-sent Pred.} & \quad y
\end{align*}
\]

- \(h_{\text{CLS}}\) corresponds to the special symbol \([CLS] \rightarrow \text{"classify"}\).
- Hence, \(h_{\text{CLS}}\) can be used as the “sequence-level representation” to construct the logistic classifier:

\[
P(\text{pos} \mid a, b) = \text{sigmoid}(W_{\text{pos}}^T h_{\text{CLS}}) \quad \text{and} \quad P(\text{neg} \mid a, b) = 1 - P(\text{pos} \mid a, b)
\]
BERT Finetune: Classification

(a) Two sentence classification

(b) One sentence classification

[Diagram showing BERT architecture for two sentence and single sentence classification]
(a) Reading Comprehension

(b) Sequence Tagging (NER)
BERT: Empirical Performance (1/2)

State-of-the-art (SOTA) performance by a large margin for two-sentence & one-sentence classification tasks.

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-GPT SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.9</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>88.1</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.2</td>
</tr>
<tr>
<td>BERT base</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.1</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT large</td>
<td><strong>86.7/85.9</strong></td>
<td><strong>72.1</strong></td>
<td><strong>91.1</strong></td>
<td><strong>94.9</strong></td>
<td><strong>60.5</strong></td>
<td><strong>86.5</strong></td>
<td><strong>89.3</strong></td>
<td><strong>70.1</strong></td>
<td><strong>81.9</strong></td>
</tr>
</tbody>
</table>

Table 2: GLUE Test results, scored by the GLUE evaluation server. All results obtained from https://gluebenchmark.com/leaderboard.

- Trained on about 3.3 billion words (Wikipedia + book corpus).
- Larger models compared to ELMo and consume lots of memory
  - BERT base: 12-layer Transformer
  - BERT large: 24-layer Transformer
SOTA performances on question answering tasks:

<table>
<thead>
<tr>
<th>System</th>
<th>EM</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>82.3</td>
<td>91.2</td>
</tr>
<tr>
<td>Prev SOTA (Single)</td>
<td>83.5</td>
<td>90.1</td>
</tr>
<tr>
<td>Prev SOTA (Ensemble)</td>
<td>86.0</td>
<td>91.7</td>
</tr>
<tr>
<td>BERT large (Single)</td>
<td>85.1</td>
<td>91.8</td>
</tr>
<tr>
<td>BERT large (Ensemble)</td>
<td>87.4</td>
<td>93.2</td>
</tr>
</tbody>
</table>

**Table 3:** SQuAD Test results (10/08/2018)

<table>
<thead>
<tr>
<th>System</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human (expert)</td>
<td>-</td>
<td>85.0</td>
</tr>
<tr>
<td>ESIM+GloVe</td>
<td>51.9</td>
<td>52.7</td>
</tr>
<tr>
<td>ESIM+ELMo</td>
<td>59.1</td>
<td>59.2</td>
</tr>
<tr>
<td>BERT base</td>
<td>81.6</td>
<td>-</td>
</tr>
<tr>
<td>BERT large</td>
<td>86.6</td>
<td>86.3</td>
</tr>
</tbody>
</table>

**Table 4:** SWAG Dev and Test results
Unsupervised representation learning starts to **really** work in NLP

- A large amount of cheap data
- Scalable pre-training objective (language modeling)
- Powerful neural networks that are easy to optimize (Transformer)

Try these pretrained models for your problem (available online)!

- BERT (TensorFlow):
  [https://github.com/google-research/bert](https://github.com/google-research/bert)

- BERT (PyTorch):
  [https://github.com/huggingface/pytorch-pretrained-BERT](https://github.com/huggingface/pytorch-pretrained-BERT)
Bert: Pre-training of deep bidirectional transformers for language understanding. 

A synopsis of linguistic theory, 1930-1955. 

Distributed representations of words and phrases and their compositionality. 

Deep contextualized word representations. 

Attention is all you need. 