Applied Natural Language Processing

Info 256
Lecture 13: Transformers 2 (Oct 9, 2023)

David Bamman, UC Berkeley
How do we use word embeddings for document classification?
I loved the movie!
I loved the movie!

---

I loved the movie!
Transformers

- Vaswani et al. 2017, “Attention is All You Need”

- Transforms map an input sequence of vectors to an output sequence of vectors of the same dimensionality
Let's assume (for the moment) that our input vectors are static word2vec embeddings of words.
The value for time step $j$ at layer $i$ is the result of attention over all time steps in the previous layer $i-1$. The values for $e_{1,1}$, $e_{1,2}$, and $e_{1,3}$ are as follows:

- $e_{1,1} = \begin{bmatrix} -0.7 & -1.3 & 0.4 & -0.4 & -0.7 \end{bmatrix}$
- $e_{1,2} = \begin{bmatrix} 0.3 & 0.3 & -1.7 & 0.7 & -1.1 \end{bmatrix}$
- $e_{1,3} = \begin{bmatrix} 1.6 & -0.3 & -0.9 & -0.7 & 0.2 \end{bmatrix}$
• Let’s separate out the different functions that an input vector has in attention by transforming it into separate representations for its role in a weighted sum (the value) from the roles used to assess compatibility (the query and key).

\[ q_{1,1} \in \mathbb{R}^{37} \quad \left( e_{1,1} W^Q \right) \]
\[ k_{1,1} \in \mathbb{R}^{37} \quad \left( e_{1,1} W^K \right) \]
\[ v_{1,1} \in \mathbb{R}^{100} \quad \left( e_{1,1} W^V \right) \]
\[ e_{1,1} \in \mathbb{R}^{100} \]

These are all parameters we learn. 100 is the original input dimension; 37 is a hyper-parameter we choose.
Self attention from “The” at position 1 to every token in the sentence

The dog barked
• The compatibility score between two words is the dot product between their respective query and key vectors.

\[
score(e_i, e_j) = q_i \cdot k_j
\]

<table>
<thead>
<tr>
<th>a</th>
<th>0.07</th>
<th>0.58</th>
<th>0.35</th>
<th>(a = \text{softmax(scores)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>scores</td>
<td>-1.4</td>
<td>0.64</td>
<td>0.14</td>
<td></td>
</tr>
<tr>
<td>(q_1 \cdot k_1)</td>
<td>(q_1 \cdot k_2)</td>
<td>(q_1 \cdot k_3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc}
-0.2 & 1 & 0.1 & -0.8 & -1.1 \\
\hline
\text{e}_{1,1} & \text{e}_{1,2} & \text{e}_{1,3}
\end{array}
\]

The dog barked
• The output of attention is a weighted sum over the values of the previous layer.
This whole process defines one attention block. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.
This whole process defines one attention **block**. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

Transformers can stack many such blocks; where the output from block $b$ is the input to block $b+1$. 

The dog barked
I loved the movie!

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.9</td>
<td>-0.2</td>
<td>-1.1</td>
<td>-0.2</td>
<td>-0.7</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>3.1</td>
<td>-3.1</td>
<td>-2.3</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>-0.8</td>
<td>3.3</td>
<td>-0.2</td>
<td>6.5</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>1.5</td>
<td>2.2</td>
<td>1.4</td>
<td>4.2</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>-1.5</td>
<td>-0.8</td>
<td>0.9</td>
<td>9.9</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>-0.7</td>
<td>-1.6</td>
<td>0.2</td>
<td>5.4</td>
<td></td>
</tr>
</tbody>
</table>

Transformer

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>3.1</td>
<td>-1.4</td>
<td>-2.3</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>-0.7</td>
<td>-0.8</td>
<td>-1.3</td>
<td>-0.2</td>
<td>-0.9</td>
<td></td>
</tr>
<tr>
<td>2.3</td>
<td>1.5</td>
<td>1.1</td>
<td>1.4</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>-0.9</td>
<td>-1.5</td>
<td>-0.7</td>
<td>0.9</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>-0.1</td>
<td>-0.7</td>
<td>-1.6</td>
<td>0.2</td>
<td>0.6</td>
<td></td>
</tr>
</tbody>
</table>
• In this setup, a transformer has to learn *everything* from the labeled training data — including the fundamentals of the language (e.g., that “love” and “like” function similarly in determining sentiment).

• If a word is not observed in the labeled training data (e.g., “adore”), then a model has no idea what to do with it (*UNK*).
Enter Language Models

- Language modeling is the task of estimating $P(w)$

- We considered these in the context of count-and-normalize LMs that make a Markov assumption in order to make estimation tractable.

- But there are many other models that we can use to perform language modeling.
Classical (causal) language model

Consider only the left context to predict the next word (i.e., the final word in a sequence is masked)

\[ P \left( w_t \mid w_1, \ldots, w_{t-1} \right) \]
Markov LMs

\[
\prod_{i}^{n} P(w_i \mid w_{i-1}) \times P(\text{STOP} \mid w_n)
\]

\[
\prod_{i}^{n} P(w_i \mid w_{i-2}, w_{i-1}) \times P(\text{STOP} \mid w_{n-1}, w_n)
\]
Recurrent neural network

- RNN allow arbitrarily-sized conditioning contexts; condition on the entire sequence history.
Recurrent neural network

Goldberg 2017
Recurrent neural network

• Each time step has two inputs:
  • $x_i$ (the observation at time step $i$); one-hot vector, feature vector or distributed representation.
  • $s_{i-1}$ (the output of the previous state); base case: $s_0 = 0$ vector.
RNN LM

\[ s_i = \text{relu}(s_{i-1} W^s + x_i W^x + b) \]

\[ y_i = \text{softmax}(s_i W^o + b^o) \]
RNN Language model

\[ P(w \mid I) \quad P(w \mid I, \text{loved}) \quad P(w \mid I, \text{loved, the}) \]

\[
\begin{align*}
&0.1 \ 0.2 \ 0.4 \ 0.1 \ 0.2 \ \\
&0.4 \ 0.1 \ 0.2 \ 0.1 \ 0.2 \ \\
&0.1 \ 0.1 \ 0.2 \ 0.2 \ 0.4
\end{align*}
\]

\[
\text{softmax}(s_1 W^o + b^o)
\]

\[
\begin{align*}
&0.8 \ 0.5 \ -0.9 \ -2.4 \ -1.8 \ \\
&-0.2 \ 0.6 \ 0.1 \ -0.2 \ -0.3 \ \\
&1.7 \ 0.8 \ 0.2 \ -0.8 \ -0.1
\end{align*}
\]

\[
\text{softmax}(s_2 W^o + b^o)
\]

\[
\begin{align*}
&2.7 \ 3.1 \ -1.4 \ -2.3 \ 0.7 \ \\
&-0.7 \ -0.8 \ -1.3 \ -0.2 \ -0.9 \ \\
&2.3 \ 1.5 \ 1.1 \ 1.4 \ 1.3
\end{align*}
\]

\[
\text{softmax}(s_3 W^o + b^o)
\]
Masked language model

Use any context (left or right) to predict a masked word

$$P \left( w_t \mid w_{-t} \right)$$
BERT

• Transformer-based model (Vaswani et al. 2017) to predict masked word using bidirectional context + next sentence prediction.

• Core idea: use attention mechanism to learn which words in the context to pay more attention to.

• Generates multiple layers of representations for each token sensitive to its context of use.
Each token in the input starts out represented by token and position embeddings.
The value for time step $j$ at layer $i$ is the result of attention over all time steps in the previous layer $i-1$.
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.7</td>
<td>-1.3</td>
<td>0.4</td>
<td>-0.4</td>
<td>-0.7</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.2</td>
<td>1</td>
<td>0.1</td>
<td>-0.8</td>
<td>-1.1</td>
</tr>
</tbody>
</table>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.7</td>
<td>-1.3</td>
<td>0.4</td>
<td>-0.4</td>
<td>-0.7</td>
</tr>
</tbody>
</table>

The dog barked
The dog barked.
The dog barked
The dog barked
The dog barked
The dog barked.
At the end of this process, we have one representation for each layer for each token.

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_{1,1}</td>
<td>e_{1,2}</td>
<td>e_{1,3}</td>
</tr>
<tr>
<td>e_{2,1}</td>
<td>e_{2,2}</td>
<td>e_{2,3}</td>
</tr>
<tr>
<td>e_{3,1}</td>
<td>e_{3,2}</td>
<td>e_{3,3}</td>
</tr>
</tbody>
</table>

The dog barked
<table>
<thead>
<tr>
<th>e₁,₁</th>
<th>e₁,₂</th>
<th>e₁,₃</th>
<th>e₁,₄</th>
<th>e₁,₅</th>
<th>e₁,₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.7</td>
<td>-0.6</td>
<td>-0.5</td>
<td>-1.6</td>
<td>-0.6</td>
<td>-0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e₂,₁</th>
<th>e₂,₂</th>
<th>e₂,₃</th>
<th>e₂,₄</th>
<th>e₂,₅</th>
<th>e₂,₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5</td>
<td>-0.2</td>
<td>0.7</td>
<td>0.6</td>
<td>-0.5</td>
<td>-0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e₃,₁</th>
<th>e₃,₂</th>
<th>e₃,₃</th>
<th>e₃,₄</th>
<th>e₃,₅</th>
<th>e₃,₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.2</td>
<td>0.7</td>
<td>0.1</td>
<td>0.7</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

-0.5 -0.6 -0.3 -0.4 -1.1 -0.6 -0.3

The [MASKED] bark #ed [SEP]
The bark -0.5 -0.6 -0.3 -1.6 0.6 0.2 2.0 9 0.5 1.9 -1.2 -0.2 -0.6 -0.7 -1.4 -2.1 -1.1 0 -1.6 -0.7 1.9 0.6 -0.4 -0.3 -0.5 -0.7 0.7 -0.1 0.2 0.7 -0.4 0.8 -0.1 -0.6 -0.3 -0.4 -0.7 -1.2 -0.6 -0.3 -1.1 -0.4 -1.1 -0.6 -0.3

[CLS] The [MASKED] bark #ed [SEP]
BERT

• Deep layers (12 for BERT base, 24 for BERT large)

• Large representation sizes (768 per layer)

• Pretrained on English Wikipedia (2.5B words) and BooksCorpus (800M words).
**WordPiece**

- BERT uses WordPiece tokenization, which segments some morphological structure of tokens
- Vocabulary size: 30,000

<table>
<thead>
<tr>
<th>The</th>
<th>The</th>
</tr>
</thead>
<tbody>
<tr>
<td>dog</td>
<td>dog</td>
</tr>
<tr>
<td>barked</td>
<td>bark #ed</td>
</tr>
</tbody>
</table>
BERT

• Learn the parameters of this model with two objectives:
  • Masked language modeling
  • Next sentence prediction
Masked LM

• Mask one word from the input and try to predict that word as the output

• More powerful than an RNN LM since it can reason about context on both sides of the word being predicted.
The [MASKED] bark #ed [SEP]
The [MASKED] bark

[CLS] The [MASKED] bark #ed [SEP]
The dog [MASKED] ed
Next sentence prediction

• For a pair of sentences, predict from [CLS] representation whether they appeared sequentially in the training data:

➕ [CLS] The dog bark #ed [SEP] He was hungry

➖ [CLS] The dog bark #ed [SEP] Paris is in France
• BERT also encodes each sentence by appending a special token to the beginning ([CLS]) and end ([SEP]) of each sequence.

• This helps provides a single token that can be optimized to represent the entire sequence (e.g., for document classification)
• We can represent the entire document with this *one* [CLS] vector
• Why does this work? When we design our network so that a classification decision relies entirely on that one vector and allow all the parameters of the network to be updated, the parameters of the model are optimized to compress all the relevant information into that one vector so that it can predict well (and minimize the loss).

neutral sentiment

The dog bark ed
• We can represent the entire document with this *one* [CLS] vector
• Why does this work? When we design our network so that a classification decision relies entirely on that one vector and allow all the parameters of the network to be updated, the parameters of the model are optimized to compress all the relevant information into that one vector so that it can predict well (and minimize the loss).
## BERT

<table>
<thead>
<tr>
<th>L</th>
<th>H=128</th>
<th>H=256</th>
<th>H=512</th>
<th>H=768</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2/128 (BERT-Tiny)</td>
<td>2/256</td>
<td>2/512</td>
<td>2/768</td>
</tr>
<tr>
<td>4</td>
<td>4/128</td>
<td>4/256 (BERT-Mini)</td>
<td>4/512 (BERT-Small)</td>
<td>4/768</td>
</tr>
<tr>
<td>6</td>
<td>6/128</td>
<td>6/256</td>
<td>6/512</td>
<td>6/768</td>
</tr>
<tr>
<td>8</td>
<td>8/128</td>
<td>8/256</td>
<td>8/512 (BERT-Medium)</td>
<td>8/768</td>
</tr>
<tr>
<td>10</td>
<td>10/128</td>
<td>10/256</td>
<td>10/512</td>
<td>10/768</td>
</tr>
<tr>
<td>12</td>
<td>12/128</td>
<td>12/256</td>
<td>12/512</td>
<td>12/768 (BERT-Base)</td>
</tr>
</tbody>
</table>

[https://github.com/google-research/bert](https://github.com/google-research/bert)
Pretrained models

Here is a partial list of some of the available pretrained models together with a short presentation of each model.

For the full list, refer to https://huggingface.co/models.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Model id</th>
<th>Details of the model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bert-base-uncased</td>
<td>12-layer, 768-hidden, 12-heads, 110M parameters. Trained on lower-cased English text.</td>
</tr>
<tr>
<td></td>
<td>bert-large-uncased</td>
<td>24-layer, 1024-hidden, 16-heads, 338M parameters. Trained on lower-cased English text.</td>
</tr>
</tbody>
</table>
Lost in (language-specific) BERT models? We are here to help!

We currently have indexed 31 BERT-based models, 19 Languages and 28 Tasks.
We have a total of 178 entries in this table; we also show Multilingual Bert (mBERT) results if available! (see our paper)

Curious which BERT model is the best for named entity recognition in Italian? Just type "Italian NER" in the search bar!

<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
<th>NLP Task</th>
<th>Dataset</th>
<th>Dataset-Domain</th>
<th>Measure</th>
<th>Performance</th>
<th>mBERT</th>
<th>Difference with mBERT</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>Arabert v1</td>
<td>SA</td>
<td>AJGT</td>
<td>twitter</td>
<td>Accuracy</td>
<td>93.8</td>
<td>83.6</td>
<td>10.2</td>
<td></td>
</tr>
<tr>
<td>Arabic</td>
<td>Arabert v1</td>
<td>SA</td>
<td>HARD</td>
<td>hotel reviews</td>
<td>Accuracy</td>
<td>96.1</td>
<td>95.7</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>Arabic</td>
<td>Arabert v1</td>
<td>SA</td>
<td>ASTD</td>
<td>twitter</td>
<td>Accuracy</td>
<td>92.6</td>
<td>80.1</td>
<td>12.5</td>
<td></td>
</tr>
<tr>
<td>Arabic</td>
<td>Arabert v1</td>
<td>SA</td>
<td>ArSenT-D-Lev</td>
<td>twitter</td>
<td>Accuracy</td>
<td>59.4</td>
<td>51.0</td>
<td>8.4</td>
<td></td>
</tr>
</tbody>
</table>
Activity

6.classification/
BERTClassification_TODO