Seq2seq, NMT, Transformer

Milan Straka

May 03, 2021
Sequence-to-Sequence Architecture
Sequence-to-Sequence is a name for an architecture allowing to produce an arbitrary output sequence $y_1, \ldots, y_M$ from an input sequence $x_1, \ldots, x_N$.

Unlike CRF/CTC, no assumptions are necessary and we condition each output sequence element on all input sequence elements and all already generated output sequence elements:

$$P(y_i | x_1, \ldots, x_N, y_1, \ldots, y_{i-1}).$$
Figure 1 of paper "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473.
Sequence-to-Sequence Architecture

**Decoder**

![Diagram of Decoder]

**Encoder**

![Diagram of Encoder]

Training
The so-called *teacher forcing* is used during training – the gold outputs are used as inputs during training.

Inference
During inference, the network processes its own predictions – such an approach is called *autoregressive decoding*.

Usually, the generated logits are processed by an $\arg\max$, the chosen word embedded and used as next input.
In the decoder, we both:

- embed the previous prediction, using a matrix of size $\mathbb{R}^{V \times D}$, where $V$ is the vocabulary size and $D$ is the embedding size;
- classify the hidden state into current prediction, using a matrix of size $\mathbb{R}^{D \times V}$.

Both these matrices have the same meaning – they represent the target-side words in the embedding space (the first explicitly represents the words by these embeddings, the second chooses the embedding in a sense “closest” to the produced hidden state).

Therefore, it makes sense to tie these matrices, i.e., to represent one of them as a transposition of the other.
Attention

\[ x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_N \]

\[ \text{E} \quad \text{E} \quad \text{E} \quad \cdots \quad \text{E} \]

\[ \begin{array}{c}
\text{Attention} \\
\text{Attention} \\
\text{Attention}
\end{array} \]

\[ \begin{array}{c}
BOS \\
y_1 \\
y_2 \\
\text{EOS}
\end{array} \]

\[ \begin{array}{c}
\text{E} \\
\text{E} \\
\text{E} \\
\text{E}
\end{array} \]

\[ \begin{array}{c}
\text{Cls} \\
\text{Cls} \\
\text{Cls}
\end{array} \]
As another input during decoding, we add context vector $c_i$:

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

We compute the context vector as a weighted combination of source sentence encoded outputs:

$$c_i = \sum_j \alpha_{ij} h_j$$

The weights $\alpha_{ij}$ are softmax of $e_{ij}$ over $j$,

$$\alpha_i = \text{softmax}(e_i),$$

with $e_{ij}$ being

$$e_{ij} = v^\top \tanh(V h_j + W s_{i-1} + b).$$
Attention Implementation

\[
\begin{align*}
    s_{i-1} & \rightarrow \tanh & \text{softmax} & \rightarrow \alpha_{i,1} \cdot h_1 \\
h_1 & & & \\
    s_{i-1} & \rightarrow \tanh & & \alpha_{i,2} \cdot h_2 \\
h_2 & & & \\
    \vdots & \rightarrow \vdots & & \\
    s_{i-1} & \rightarrow \tanh & & \alpha_{i,N} \cdot h_N \\
h_N & & & \\
\end{align*}
\]
Trained Attention Visualization

Figure 3 of paper "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473.
Subword Units

Translate *subword units* instead of words. The subword units can be generated in several ways, the most commonly used are:

- **BPE**: Using the *byte pair encoding* algorithm. Start with individual characters plus a special end-of-word symbol ·. Then, merge the most occurring symbol pair $A, B$ by a new symbol $AB$, with the symbol pair never crossing word boundary (so that the end-of-word symbol cannot be inside a subword).

Considering a dictionary with words *low, lowest, newer, wider*, a possible sequence of merges:

\[ r \cdot \rightarrow r\cdot \]
\[ lo \rightarrow lo \]
\[ lo\ w \rightarrow low \]
\[ e\ r\cdot \rightarrow er\cdot \]
• **Wordpieces**: Given a text divided into subwords, we can compute unigram probability of every subword, and then get the likelihood of the text under a unigram language model by multiplying the probabilities of the subwords in the text.

When we have only a text and a subword dictionary, we divide the text in a greedy fashion, iteratively choosing the longest existing subword.

When constructing the subwords, we again start with individual characters, and then repeatedly join such a pair of subwords, which increases the unigram language model likelihood the most.

Both approaches give very similar results; a biggest difference is that during the inference:

• for BPE, the sequence of merges must be performed in the same order as during the construction of the BPE;
• for Wordpieces, it is enough to find longest matches from the subword dictionary.

Usually quite little subword units are used (32k-64k), often generated on the union of the two vocabularies (the so-called joint BPE or shared wordpieces).
Google NMT

Beyond one Language Pair

Fig. 5. A selection of evaluation results, grouped by human rating. 

*Figure 5 of “Show and Tell: Lessons learned from the 2015 MSCOCO...”, https://arxiv.org/abs/1609.06647.*
Beyond one Language Pair

Figure 6 of "Multimodal Compact Bilinear Pooling for VQA and Visual Grounding", https://arxiv.org/abs/1606.01847.
Many attempts at multilingual translation.

- Individual encoders and decoders, shared attention.
- Shared encoders and decoders.

Surprisingly, even unsupervised translation is attempted lately. By unsupervised we understand settings where we have access to large monolingual corpora, but no parallel data.

In 2019, the best unsupervised systems were on par with the best 2014 supervised systems.

<table>
<thead>
<tr>
<th></th>
<th>WMT-14</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>fr-en</td>
</tr>
<tr>
<td>Unsupervised</td>
<td></td>
</tr>
<tr>
<td>Proposed system</td>
<td></td>
</tr>
<tr>
<td><em>detok. SacreBLEU</em></td>
<td>33.5</td>
</tr>
<tr>
<td></td>
<td>33.2</td>
</tr>
<tr>
<td>Supervised</td>
<td></td>
</tr>
<tr>
<td>WMT best*</td>
<td>35.0</td>
</tr>
<tr>
<td>Vaswani et al. (2017)</td>
<td>-</td>
</tr>
<tr>
<td>Edunov et al. (2018)</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3: Results of the proposed method in comparison to different supervised systems (BLEU).

For some sequence processing tasks, *sequential* processing (as performed by recurrent neural networks) of its elements might be too restrictive.

Instead, we may want to be able to combine sequence elements independently on their distance. Such processing is allowed in the *Transformer* architecture, originally proposed for neural machine translation in 2017 in *Attention is All You Need* paper.
Figure 1 of paper “Attention Is All You Need”, https://arxiv.org/abs/1706.03762
Transformer

![The_transformer_encoder_decoder_stack.png](http://jalammar.github.io/images/t/The_transformer_encoder_decoder_stack.png)
Transformer – Self-Attention

Assume that we have a sequence of $n$ words represented using a matrix $X \in \mathbb{R}^{n \times d}$.

The attention module for a queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$ and values $V \in \mathbb{R}^{n \times d_v}$ is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{Q K^\top}{\sqrt{d_k}} \right) V.$$

The queries, keys and values are computed from the input word representations $X$ using a linear transformation as

$$Q = W^Q \cdot X$$
$$K = W^K \cdot X$$
$$V = W^V \cdot X$$
# Transformer – Self-Attention

![Diagram of Transformer Self-Attention](http://jalammar.github.io/images/t/transformer_self_attention_vectors.png)

<table>
<thead>
<tr>
<th>Input</th>
<th>Thinking</th>
<th>Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedding</td>
<td>X₁</td>
<td>X₂</td>
</tr>
<tr>
<td>Queries</td>
<td>q₁</td>
<td>q₂</td>
</tr>
<tr>
<td>Keys</td>
<td>k₁</td>
<td>k₂</td>
</tr>
<tr>
<td>Values</td>
<td>v₁</td>
<td>v₂</td>
</tr>
</tbody>
</table>

- **Q**: Queries
- **K**: Keys
- **V**: Values

```python
http://jalammar.github.io/images/t/transformer_self_attention_vectors.png
```
Transformer – Self-Attention

Input

Embedding

Queries

Keys

Values

Score

Divide by $8 \sqrt{d_k}$

Softmax

Softmax $X$

Value

Sum

Thinking

Machines

$x_1$  
$q_1$  
$k_1$  
$v_1$  

$x_2$  
$q_2$  
$k_2$  
$v_2$  

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

14

0.88

0.12

12

$v_1$  

$v_2$  

$z_1$  

$z_2$  

http://jalammar.github.io/images/t/self-attention-output.png
Transformer – Self-Attention

The diagram illustrates the attention mechanism in a Transformer model. It shows how the query (Query q), key (Key k), and the result of the elementwise multiplication (q \times k) are computed. The attention mechanism allows the model to focus on specific parts of the input sequence, improving its ability to understand the context of the text.

For more detailed information, refer to the source image at https://miro.medium.com/max/2000/1*5jBSVNOOcJ-13tsLVgni_w.png.
Transformer – Self-Attention

\[
\begin{align*}
X & \times W^Q = Q \\
X & \times W^K = K \\
X & \times W^V = V
\end{align*}
\]

\[
\text{softmax} \left( \frac{Q \cdot K^T}{\sqrt{d_k}} \right) = Z
\]
