Seq2seq, NMT, Transformer

Milan Straka

📅 Apr 11, 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:

$$P(y_i | x_1, \ldots, x_N, y_1, \ldots, y_{i-1}).$$
Sequence-to-Sequence Architecture

Figure 1 of "Sequence to Sequence Learning with Neural Networks", https://arxiv.org/abs/1409.0473
Sequence-to-Sequence Architecture

Figure 1 of "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", https://arxiv.org/abs/1406.1078
Sequence-to-Sequence Architecture

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.
Tying Word Embeddings

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
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

\[ s_{i-1} \]
\[ h_1 \]
\[ s_{i-1} \]
\[ h_2 \]
\[ \vdots \]
\[ s_{i-1} \]
\[ h_N \]
\[ e_{i,1} \]
\[ e_{i,2} \]
\[ e_{i,N} \]
\[ \alpha_{i,1} \cdot h_1 \]
\[ \alpha_{i,2} \cdot h_2 \]
\[ \alpha_{i,N} \cdot h_N \]
\[ c_i \]

\[ \text{tanh} \]
\[ \text{softmax} \]

\[ + \]
Trained Attention Visualization

![Diagram](image)

Figure 3 of "Neural Machine Translation by Jointly Learning to Align and Translate", https://arxiv.org/abs/1409.0473
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 \( \cdot \). 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:

\[
\begin{align*}
  r \cdot & \rightarrow r \cdot \\
  lo & \rightarrow lo \\
  lo \ w & \rightarrow low \\
  e \ r \cdot & \rightarrow er \cdot 
\end{align*}
\]
Subword Units

- **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 that increases the unigram language model likelihood the most.

Both approaches give very similar results; the 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 (because we use the output of BPE as training data),
- 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

Figure 1 of "Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144

NPFL114, Lecture 9
Figure 5 of "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", https://arxiv.org/abs/1609.08144
Figure 6 of “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation“, https://arxiv.org/abs/1609.08144
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

- What vegetable is the dog chewing on?
  - MCB: carrot
  - GT: carrot

- What kind of dog is this?
  - MCB: husky
  - GT: husky

- What kind of flooring does the room have?
  - MCB: carpet
  - GT: carpet

- What color is the traffic light?
  - MCB: green
  - GT: green

- Is this an urban area?
  - MCB: yes
  - GT: yes

- Where are the buildings?
  - MCB: in background
  - GT: on left
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>Proposed system detok. SacreBLEU*</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Supervised</td>
<td>WMT best*</td>
</tr>
<tr>
<td></td>
<td>Vaswani et al. (2017)</td>
</tr>
<tr>
<td></td>
<td>Edunov et al. (2018)</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 "Attention Is All You Need", https://arxiv.org/abs/1706.03762
Transformer

**INPUT:** Je suis étudiant

**OUTPUT:** I am a student

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 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

$$Q = X W^Q$$
$$K = X W^K$$
$$V = X W^V$$

for trainable weight matrices $W^Q, W^K \in \mathbb{R}^{d \times d_k}$ and $W^V \in \mathbb{R}^{d \times d_v}$. 
Transformer – Self-Attention

Input

Embedding

Queries

Keys

Values

Thinking

Machines

X1

X2

q1

q2

k1

k2

v1

v2

WQ

WK

WV

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

$x_1$  $x_2$

$q_1$  $q_2$

$k_1$  $k_2$

$v_1$  $v_2$

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

14  12

0.88  0.12

$v_1$  $v_2$

$z_1$  $z_2$

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

The self-attention mechanism involves calculating the attention weights for each element in a sequence. The process can be represented as follows:

- **Input Sequence (X)**
- **Query (Q)**
- **Key (K)**
- **Value (V)**

The equations for self-attention are:

\[
\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)Z
\]

- **Softmax** function is used to calculate the attention weights.
- **Z** represents the output of the attention layer.

The matrices **W^Q**, **W^K**, and **W^V** are used to transform the input sequence into query, key, and value matrices, respectively.

支撑图：
- [Self-Attention Matrix Calculation](http://jalammar.github.io/images/t/self-attention-matrix-calculation.png)
- [Self-Attention Matrix Calculation 2](http://jalammar.github.io/images/t/self-attention-matrix-calculation-2.png)