Abstract

In this work, we conduct a comprehensive investigation on one of the centerpiece of modern machine translation systems: the encoder-decoder attention mechanism. Motivated by the concept of first-order alignments, we extend the (cross-)attention mechanism by a recurrent connection, allowing direct access to previous attention/alignment decisions. We propose several ways to include such a recurrence into the attention mechanism. Verifying their performance across different translation tasks we conclude that these extensions and dependencies are not beneficial for the translation performance of the Transformer architecture.

1 Introduction

Since its introduction by Vaswani et al. (2017), the Transformer architecture has enabled state of the art results on nearly all machine translation (MT) tasks (Bojar et al., 2018; Barrault et al., 2019; Ott et al., 2018). Compared to previous neural machine translation (NMT) approaches (Sutskever et al., 2014; Bahdanau et al., 2015), it introduces many new concepts like self-attention, positional encoding and multi-head attention. However, the Transformer still relies on the encoder-decoder attention mechanism introduced by Bahdanau et al. (2015) to translate a source sentence into the target language. While for earlier NMT models, this attention mechanism was thoroughly investigated and many different variants were proposed (Feng et al., 2016; Cohn et al., 2016; Sankaran et al., 2016; Tu et al., 2016), the same can not be said for the Transformer. In the present work, we discuss the Transformer encoder-decoder attention mechanism, propose different ways to enhance its capabilities and analyze the resulting systems.

One particular design decision in the Transformer attention mechanism catches the eye: When calculating the context vector in the current decoding step, there is no direct information flow coming from the previous steps. While earlier neural architectures explicitly incorporated the hidden state from the previous decoding step in the attention calculation (Bahdanau et al., 2015) and traditional count-based alignment models used higher order Markov assumptions, the Transformer relies on the self-attention mechanism and layer stacking to learn context dependencies. Therefore we ask the questions if and how an explicit dependency on the previous attention decisions should be included in the Transformer encoder-decoder attention mechanism. In order to provide an answer we propose numerous approaches towards modeling such an explicit dependency and report our findings across three language pairs.

2 Related Work

In recurrent network architectures (Bahdanau et al., 2015) the decoder state recurrently depends on the previous decoding step. Many works have extended this by additionally adding an explicit recurrent dependency within the attention mechanism itself. Feng et al. (2016) concatenate the attention context produced in the previous decoding step to the input of the attention mechanism. Other approaches approximate a coverage value for every source position by accumulating the attention weights over all previous time steps, which is then included in the attention calculation (Cohn et al., 2016). Tu et al. (2016) extend this idea by renormalizing the coverage using a fertility model that predicts how much attention a specific source word should receive. In a similar spirit Sankaran et al. (2016) explicitly bias the attention weights to be more focused on source positions that did not receive much attention yet.

In contrast to network architectures with a recurrent decoder, the Transformer (Vaswani et al., 2017) is trained completely parallel and uses multi-head, additive cross-attention. This work tries to answer whether introducing a recurrent dependency can
also benefit the Transformer cross-attention.

3 Recurrent Cross-Attention

3.1 Encoder-Decoder Attention

The ‘vanilla’ Transformer is an intricate encoder-decoder architecture that uses an attention mechanism to map a sequence of input tokens $f_i^j$ onto a sequence of output tokens $e_i^j$. In this framework, a context vector $c_i^{ℓ,n}$ for the ℓ-th decoder layer and the n-th attention head is calculated in the i-th decoding step by

$$c_i^{ℓ,n} = \sum_j \alpha_i^{ℓ,n}(W_v^{ℓ,n} h_j).$$

Here, $h_j$ denotes the j-th output of the encoder which is transformed by a trainable weight matrix $W_v^{ℓ,n}$ into the value. $\alpha_i^{ℓ,n}$ is calculated using $h_j$ as well as the output of the previous decoder layer (after self-attention) $s_i^ℓ$. More specifically, we calculate the energy

$$\hat{\alpha}_i^{ℓ,n} = \frac{1}{\sqrt{d_k}} (W_{k}^{ℓ,n} h_j)^T (W_{q}^{ℓ,n} s_i^ℓ)$$

where $d_k$ is the feature dimension, $W_{k}^{ℓ,n}$ and $W_{q}^{ℓ,n}$ are trainable weight matrices, transforming $h_j$ and $s_i^ℓ$ into the key and query respectively. This naming stems from the intuition that we use a query $W_q^{ℓ,n} s_i^ℓ$ to perform a lookup on a series of key-value pairs:

$$\left(W_{k}^{ℓ,n} h_1, W_{v}^{ℓ,n} h_1\right), \ldots, \left(W_{k}^{ℓ,n} h_J, W_{v}^{ℓ,n} h_J\right).$$

The energy $\hat{\alpha}_i^{ℓ,n}$ is then normalized using the softmax operation to get the so called attention ‘weights’

$$\alpha_i^{ℓ,n} = \frac{\exp(\hat{\alpha}_i^{ℓ,n})}{\sum_j \exp(\hat{\alpha}_j^{ℓ,n})}. \tag{1}$$

Once the $c_i^{ℓ,n}$ are calculated, the full context vector $c_i^ℓ$ is formed by concatenating the outputs of all attention heads followed by a linear transformation.

A combination of residual connections, feedforward and self attention layers is used to transform $c_i^ℓ$ into $s_i^{ℓ+1} = f(c_i^ℓ)$, the decoder state before the next cross-attention layer. In this work we focus on the cross-attention and refer the reader to Vaswani et al. (2017) for the details on the self-attention concept.

One thing that becomes obvious in the above description is the lack of information flow along the decoder ‘time-axis’ i. The only way the system can make use of such information is through the aforementioned self-attention concept. In this work we raise the question whether such an indirect way of information flow is sufficient or if the system can profit from a more direct integration of its ‘past attention decisions’.

3.2 Modifying the Query

A straight forward way to use information from the previous decoder time step $i - 1$ in the current attention calculation is by modifying the query vector. We do this by simple concatenation, resulting in

$$\hat{\alpha}_{i,j} = \frac{1}{\sqrt{d_k}} (W_{k}^{ℓ,n} h_j)^T \left(W_{q}^{ℓ,n} \left(s_i^{ℓ} \right)\right)$$

where $f_{i-1}$ is some function holding information from the previous time step. One apparent way to define this function is the concatenate previous context variant,

$$f_{i-1} = c_i^{ℓ,n} - 1 \tag{2}$$

However, here we only take into account the time step directly preceding the current one. In order to investigate if additional information from earlier decisions might be helpful, we define the concatenate previous accumulated weight approach:

$$f_{i-1} = \sum_{i'=1}^{i-1} \alpha_{i,j}^{ℓ,n} \tag{3}$$

specifying how much the encoder output from the j-th position has been attended to so far.

For all of the variants described in this section, the resulting energies $\hat{\alpha}_{i,j}^{ℓ,n}$ are normalized using a softmax operation (see Equation 1).

3.3 Expanding the Key-Value List

Staying in the ‘query-key-value’ framework, the pendant to modifying the query vector (as in Section 3.2) would be to modify the key-value list in order to incorporate information from the previous time step. We expand this list by inserting one additional vector pair $(g_k, g_v)$ along the time axis and name this approach expand key-value list.

For choosing the vectors $g_k$ and $g_v$, we test four different variants. In variant 1 we use the (linearly transformed) full context vector $c_i^{ℓ-1}$ from the previous time step as both additional key and value
vector
\[ g_k = W_k^{\ell,n} c_{i-1}^\ell, \quad g_v = W_v^{\ell,n} c_{i-1}^\ell. \] (5)
The context vector is transformed using the same matrices \(W_k^{\ell,n}\) and \(W_v^{\ell,n}\) which we also use for transforming the other keys and values respectively. One can argue that a separate transformation is needed for the context vector, which leads us to \textbf{variant 2},
\[ g_k = W_{g_k}^{\ell,n} c_{i-1}^\ell, \quad g_v = W_{g_v}^{\ell,n} c_{i-1}^\ell \] (6)
where \(W_{g_k}^{\ell,n}\) and \(W_{g_v}^{\ell,n}\) are used specifically for transforming \(c_{i-1}^\ell\). Furthermore, we speculate that a specific attention head should mostly just benefit from incorporating its own previous output. Therefore we define \textbf{variant 3} as:
\[ g_k = W_k^{\ell,n} c_{i-1}^\ell, \quad g_v = W_v^{\ell,n} c_{i-1}^\ell. \] (7)
where just the context vector \(c_{i-1}^\ell\), produced by the same head, is considered in the calculation. Finally we test \textbf{variant 4} in which only the key is transformed but the value is not:
\[ g_k = W_k^{\ell,n} c_{i-1}^\ell, \quad g_v = c_{i-1}^\ell. \] (8)
The rationale here is that \(c_{i-1}^\ell\) already ‘belongs’ in the context vector embedding space (not the encoder output space like \(h_j\)) and therefore no transformation should be necessary. On a side note, while all of these changes might make sense from an architectural point of view, they certainly raise questions regarding the interpretability of the attention weights as a target to source alignment.

3.4 Re-scaling the Attention Weights
Finally, the most direct way to use information from the previous time step \(i - 1\) in the current attention calculation is by directly modifying the attention weights. We test two ways of doing this:

- **Encouraging continuous attention** patterns where the attention weights from the previous decoding step are similar to the weights of the current one
\[ \hat{\alpha}_{i,j}^{\ell,n} = \lambda \hat{\alpha}_{i,j}^{\ell,n} + \frac{1 - \lambda}{2k + 1} \sum_{j' = j-k}^{j+k} \hat{\alpha}_{i-1,j'}^{\ell,n}. \] (9)

- **Encouraging coverage** by reducing the attention weight by an amount proportional to the extend in which the source position \(j\) already has been attended to in all preceding time steps combined
\[ \hat{\alpha}_{i,j}^{\ell,n} = \hat{\alpha}_{i,j}^{\ell,n} - \frac{\lambda}{\sqrt{d_k}} \sum_{i' = 1}^{i-1} \hat{\alpha}_{i',j}^{\ell,n}. \] (10)

For both variants we apply normalization
\[ \alpha_{i,j}^{\ell,n} = \text{softmax}(\hat{\alpha}_{i,j}^{\ell,n}) \] (11)
and tune the hyperparameters: the scaling factor \(\lambda\) (both approaches) and the window size \(k\) (only first).

4 Experimental Setup
We evaluate our approaches on three tasks: The WMT 2016 news translation Romanian→English task, the WMT 2018 news translation Turkish→English task, as well as the IWSLT 2017 English→Italian translation task on TED data. Our training data consists of 612k (Ro→En: SE Times, Europarl v8), 208k (Tr→En: SE Times) and 227k (En→It: TED talk) parallel sentences, which we preprocess using 20k byte-pair-encoding operations (8k for En→It) learned jointly on source and target data.

We train a 6-layer Transformer for each task, similar to the ‘base’-configuration of Vaswani et al. (2017). All models are implemented in RETURNN (Zeyer et al., 2018). We tie the weights of all embedding/projection matrices and apply a dropout of 20% for Ro→En and 30% for Tr→En and En→It. The baseline models use a batch size of 9600, however GPU memory limitations allow a batch size of maximum 7600 for some experiments that add a recurrency to the decoder. We select the best checkpoint according to BLEU on the development set and report case-sensitive BLEU calculated with SacreBLEU (Post, 2018) and TER with TERCom (Snover et al., 2006) on a holdout test set.

<table>
<thead>
<tr>
<th>System Architecture</th>
<th>Variant</th>
<th>BLEU</th>
<th>TER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td></td>
<td>35.7</td>
<td>51.4</td>
</tr>
<tr>
<td>Expand key-value list</td>
<td>1 (Eq. 5)</td>
<td><strong>35.9</strong></td>
<td><strong>51.1</strong></td>
</tr>
<tr>
<td></td>
<td>2 (Eq. 6)</td>
<td><strong>35.9</strong></td>
<td><strong>51.1</strong></td>
</tr>
<tr>
<td></td>
<td>3 (Eq. 7)</td>
<td>35.7</td>
<td>51.5</td>
</tr>
<tr>
<td></td>
<td>4 (Eq. 8)</td>
<td>33.8</td>
<td>53.2</td>
</tr>
</tbody>
</table>

Table 1: Performance of the different variants of the \textbf{expand key-value lists} approach (Section 3.3) on the development set of the Ro→En task.

5 Experimental Results
5.1 Tuning of the Methods
We tune and select all presented hyperparameters of specific model variants on the development set of the Ro→En task.

The different variants of the \textbf{expand key-value list} approach introduced in Section 3.3 differ solely
in the way in which the context vector is transformed before being used as an additional key-value pair. The performance of each variant in terms of BLEU and TER is shown in Table 1. The variants 1 (Equation 5) and 2 (Equation 6) perform the strongest, being both slightly better than our baseline system. Re-using the transformation matrices from the other key-value pairs does not seem to hurt the system. Limiting the additional context information to the same attention head (variant 3, Equation 7) results in a slight performance loss. Additionally, omitting the transformation of \( c_{i-1} \) for the value-list (variant 4, Equation 8) results in a significant performance loss, indicating that this vector is not directly compatible with the other vectors in the list after all. Since it exhibits the best balance between performance and complexity, we choose variant 1 (Equation 5) for the complete system comparison.

Furthermore, we have to look at the different ways for re-scaling the attention weights as introduced in Section 3.4. We tune the hyperparameters \( k \) and \( \lambda \) for each method applicable. For the window size, we find \( k = 5 \) to work best and for the scaling factor we choose \( \lambda = 0.5 \) for all variants.

5.2 Main Comparison

The comparison of all the approaches defined in Section 3.1 and tuned/selected in Section 5.1 are shown in Table 2. Note that all the approach-specific hyperparameter tuning was done on the Ro→En task, distinguishing it from the other two.

For the most part there is very little variation in system performance across all proposed methods, none of which can outperform the Transformer baseline by a significant amount. While there were still some (although small) improvements visible when evaluating on the development set, e.g. for the methods discussed in Section 3.4, these mostly vanish when evaluating on unseen test sets and on different tasks. This might be a testament to overfitting on the development set when tuning the hyperparameters.

While one can argue that the proposed methods exhibit the same level of performance as the Transformer baseline, there is a significant downside: training speed. In the last column of Table 2 the average computation time per checkpoint relative to the Transformer baseline is shown. All proposed methods slow down the training by at least a factor of 5. This is due to a combination of breaking the parallelization inside the decoder (we have to wait for timestep \( i-1 \) to finish in order to do the computations for timestep \( i \)) and having to use a smaller batch size in training.

6 Conclusion

In this work we provide a detailed analysis on the encoder-decoder attention mechanism in the Transformer architecture. We argue that – compared to previous attention formulations – there does not exist a direct link to the context produced in the earlier decoding steps. We propose different approaches to explicitly model this link and test the resulting systems on three machine translation tasks. The results show no significant improvements for any of the tested approaches. This leads us to the conclusion that the context information which is incorporated through self-attention is already sufficient for the given task of machine translation.

<table>
<thead>
<tr>
<th>System Architecture</th>
<th>Ro→En newstest2016</th>
<th>Tr→En newstest2018</th>
<th>En→It TED tst2010</th>
<th>Train Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>current state-of-the-art</td>
<td>BLEU</td>
<td>TER</td>
<td>BLEU</td>
<td>TER</td>
</tr>
<tr>
<td>Baseline</td>
<td>34.2</td>
<td>53.5</td>
<td>19.6</td>
<td>70.2</td>
</tr>
<tr>
<td>Concat prev context (Eq. 2)</td>
<td>33.8</td>
<td>53.8</td>
<td>19.5</td>
<td>70.3</td>
</tr>
<tr>
<td>Concat prev weight (Eq. 3)</td>
<td>34.3</td>
<td>53.4</td>
<td>19.8</td>
<td>70.2</td>
</tr>
<tr>
<td>Concat prev accum weight (Eq. 4)</td>
<td>34.2</td>
<td>53.4</td>
<td>19.7</td>
<td>70.3</td>
</tr>
<tr>
<td>Expand key-value list (Eq. 5)</td>
<td>34.0</td>
<td>53.6</td>
<td>19.8</td>
<td>70.5</td>
</tr>
<tr>
<td>Accumulation of the energies (Eq. 9)</td>
<td>34.2</td>
<td>53.2</td>
<td>19.6</td>
<td>70.2</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison of the approaches using additional context information from the previous time steps as described in Section 3.1. Train time refers to the average GPU time per training checkpoint measured on Ro→En. We show the best results reported in literature for each task: ¹ Kasai et al. (2020), ² Marie et al. (2018) and ³ Lakew et al. (2017)
Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 694537, project "SEQCLAS"). The work reflects only the authors’ views and the European Research Council Executive Agency (ERCEA) is not responsible for any use that may be made of the information it contains.

References


