Encoder-decoder sequence-to-sequence models

LING83600

1 Introduction: the problem of length

Many tasks in speech and language processing can be cast as mapping from an source sequence $\mathbf{X} = x_0, x_1, \ldots x_n$ to an target sequence $\mathbf{Y} = y_0, y_1, \ldots y_m$. When the two sequences are the same length (i.e., $n = m$), then the task reduces to tagging. One merely extracts the features, either using custom domain-specific feature extraction functions or using a recurrent neural network (RNN), and then either

- greedily decodes the tag sequence left-to-right,
- uses an approximate decoding strategy like beam search, or
- decodes using conditional random fields (Lafferty et al. 2001).

In addition to part-of-speech tagging, named entity recognition, and similar tasks, this general approach can also be used for contextual lemmatization: for each token x_i the tag y_i is a "pointer" to a rewrite rule r such that $r(x_i)$ gives the ith lemma (Chrupała et al. 2008).

However, this is not the case for many interesting problems, including grapheme-to-phoneme conversion, morphological generation, and machine translation. For all of these problems, the relationship between input length n and output length m is complex and conditioned not just by n but by the contents of the input sequence $\mathbf{X}$.

Bibliographic note

This handout is based in part on unpublished slides by Roger Grosse and Jimmy Ba, as well as the cited works.

2 Neural sequence models

In a neural sequence(-to-sequence) model, we observe the source sequence $\mathbf{X} = x_1, x_2, \ldots, x_n$. We then use some method—usually an RNN—to compute a real-valued "encoding" or "annotation" of the source sequence $\mathbf{Z} \in \mathbf{R}^{a \times n}$. Then, we use $\mathbf{Z}$ to generate the target sequence $\mathbf{Y} = y_1, y_2, \ldots, y_m$ one element at a time. By convention, we add end-of-string symbols `<eos>` to the end of the output sequence and halt decoding (i.e., output generation) whenever this symbol is generated.

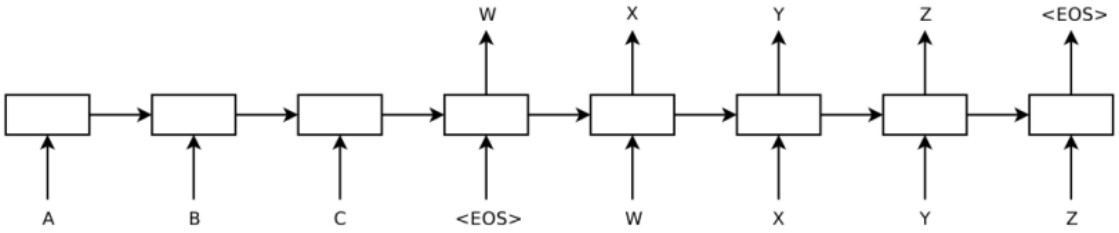

Figure 1: A simple neural sequence-to-sequence model, after Sutksever et al. 2014.

In the simplest form of this model (e.g., Sutksever et al. 2014), the prediction for y_i is conditioned by z_n, the encoding at the end of the input sequence, and the previously generated outputs $y_1, y_2, \ldots, y_{i-1}$. The model is said to be *auto-regressive* because it can "see" the encoding of its previous predictions. This architecture is illustrated in Figure 1.

The major limitation of this simple architecture is that it forces the encoder portion of the model—the part that produces encodings for the input symbols—to "stuff" all of the information about the input sequence into a single vector $\mathbf{z}_n$, the encoding the RNN outputs at the end of the input sequence. Furthermore, it does not have a mechanism to express the intuition that some target word y_i "translates" "aligns to", or "covers" some source word x_i, though we know this to be true and could express this simple intuition even in all but the simplest statistical machine translation (e.g. Brown et al. 1993).

3 The encoder-decoder model with attention

Bahdanau et al. (2015) and Luong et al. (2015) propose an alternative model. Their intuition is that these alignments between source and target words are real and we simply need to learn which source word(s) each target word aligns to. Like before, the decoder makes predictions one target word at a time, and is fed encodings of its previous predictions. However, it also is fed a *context vector* $\mathbf{c}$ computed by selectively "attending to" the encodings of one or more source words. The context vector is assumed to be some kind of weighted sum of those embeddings, thus

$$\mathbf{c} = \sum_{j=1}^{n} \alpha_{i,j} \mathbf{z}_j.$$

How are the attention weights α computed? The attention weight for target position i aligning to source token j is given by concatenating the source encoding $\mathbf{z}_j$ and the decoder state $\mathbf{h}_i$. This is then run through a softmax (over $j = 1, \ldots, n-1$) so that the attention weights are a probability distribution over source tokens. (This can be somewhat painfully converted to matrix form so it can be computed efficiently by your GPU.) Crucially, the attention weights depend only on the source and decoder encodings; they do not depend on source or target position. For instance, we could imagine that the decoder is expecting an adjectives and we find this by looking for adjective-like encodings in the source. This model is illustrated in Figure 2. Figure 3 shows an example *attention mask* for a translation problem.

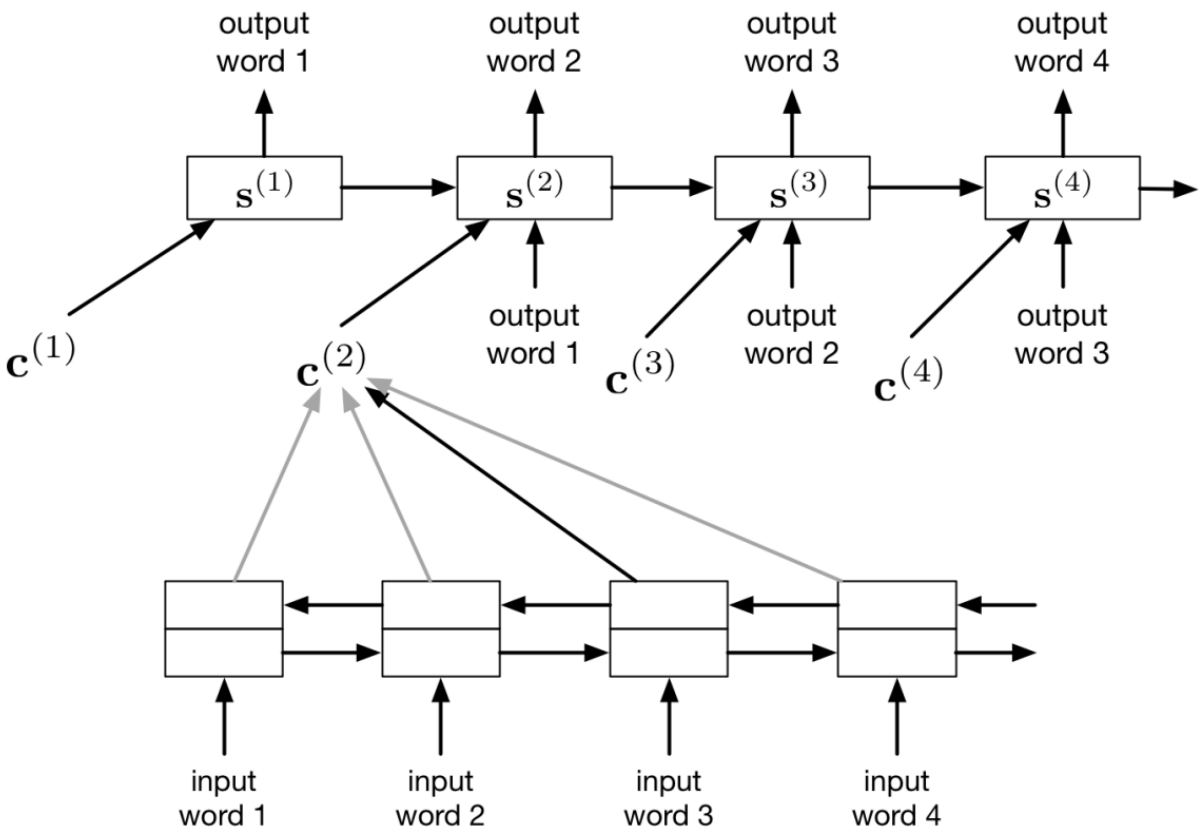

Figure 2: An encoder-decoder model with local attention, after Bahdanau et al. 2015.

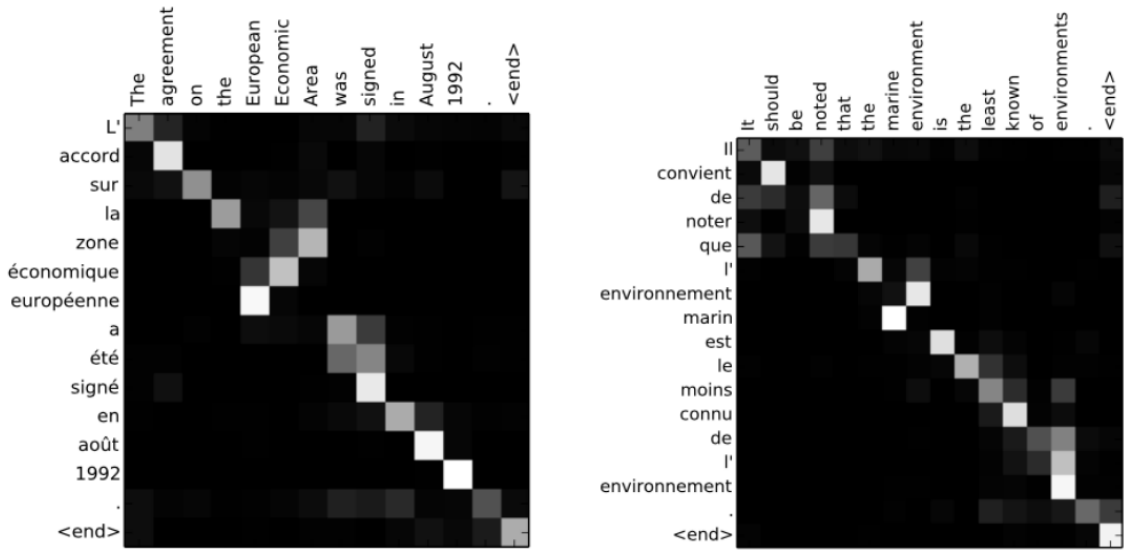

Figure 3: Example attention masks for a translation task.

References

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics* 19:263–311.

Chrupała, Grzegorz, Georgiana Dinu, and Josef van Genabith. 2008. Learning morphology with Morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation*, 2362–2367.

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 282–289.

Luong, Thang, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421.

Sutksever, Ilya, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processsing Systems*, 3104–3112.