CS 194-129 Deep Neural Networks
Spring 2018 Canny Lecture 14

1 Reading

Neural Machine Translation

The goal is to construct a single neural network that can be optimized for reading a sentence and outputting a translation. The proposed method extends the traditional sentence encoder-decoder architecture by allowing the model to search for the parts of a source sentence that are relevant to predicting each target word.

The motivation is that previous encoder-decoder approaches are limited in that they are forced to represent all of the necessary information from the source sentence in a fixed-length embedding. Hence they tend to struggle with longer sentences.

On the other hand, the proposed model learns to align and translate jointly. At each step, it searches for a set of positions in the source sentence where the most relevant information resides, and then generates a word based on all previously generated words and the context vector for the found positions.

Importantly, the model does not encode the entire source sentence into a single fixed-length vector. Rather, it encodes the source sentence into a sequence of vectors and selects subsets of these vectors while decoding the translation.

Background

Translation is equivalent to finding a target sentence y for a source sentence x such that the conditional probability of y given x is maximized, i.e. $\arg\max_y p(y \mid x)$. In neural machine translation, we attempt to fit a model to this conditional distribution over all sentence pairs (x, y).

In the traditional encoder-decoder framework, an encoder RNN converts an input sentence, i.e. a sequence of vectors $x = (x_1, ..., x_{T_x})$, into a context vector c as

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ c &= q(\{h_1, ..., h_{T_x}\}) \end{aligned}$$

where $h_t \in \mathbb{R}^n$ is the hidden state at time t and f, q are nonlinear functions.

Then the decoder predicts the next word $y_{t'}$ from the context vector c and all previously predicted words $\{y_1, ..., y_{t'-1}\}$. Formally, it models a probability over the full translation $y = (y_1, ..., y_{T_y})$ by decomposing the joint probability (i.e. of everything) into

$$p(y) = \prod_{t=1}^{T} p(y_t \mid \{y_1, ..., y_{t-1}\}, c)$$

where each $p(y_t \mid \{y_1, ..., y_{t-1}\}, c)$ is output by the RNN as $g(y_{t-1}, s_t, c)$, s.t. s_t is the hidden state.

Learning to Align and Translate

In the proposed architecture, the encoder maps the source sentence into a sequence of annotations $(h_1, ..., h_{T_x})$. We would like the annotation of each word to summarize both the preceding and following words, so we'll

use a bidirectional RNN (BiRNN). This consists of a *forward RNN* which reads the sentence in order and computes a sequence of forward hidden states $(h_{f1}, ..., h_{fT_x})$, and a *backward RNN* which reads the sentence in reverse and computes a sequence of backward hidden states $(h_{b1}, ..., h_{bT_x})$. The annotation h_j for each word x_j is then obtained by concatenating the forward hidden state h_{fj} and the backward hidden state h_{bj}.

Meanwhile, the decoder defines each conditional probability as $p(y_i \mid y_1, ..., y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$ where $s_i = f(s_{i-1}, y_{i-1}, c_i)$ is the RNN hidden state for time i. Note that there is now a distinct context vector c_i for each target word y_i, which is computed as a weighted sum of the annotations $(h_1, ..., h_{T_x})$ from the encoder. (Each annotation describes the entire sentence, with a strong focus on the region surrounding the ith word.)

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

The weights α_{ij} for each annotation h_j are computed as

$$\alpha_{ij} = \frac{e^{a(s_{i-1}, h_j)}}{\sum_{k=1}^{T_x} e^{a(s_{i-1}, h_k)}}$$

where $a(s_{i-1}, h_j)$ is an alignment model which scores how well the inputs around position j and the output at position i match. s_{i-1} is the RNN hidden state used to compute y_i. It is parameterized as a feedforward neural network which is jointly trained with the rest of the system.

Note: we can think of α_{ij} as the probability that the target word y_i is aligned to – i.e. translated from – the source word x_j. In this way, the ith context vector c_i is the "expected annotation" over all possible alignments of the target word with source words.

Essentially, information is spread throughout the sequence of annotations, which are selectively retrieved by the decoder as needed. Alignments are made in a soft fashion, i.e. we use continuous weights to align each target word to each input word. In this way we don't need to hard-align words such as "the" to potentially nonexistent words in the other language.

The proposed model ends up being much better than conventional models at translating long sentences. This is probably because it only needs to accurately encode parts of the source sentence that surround a particular word (and not the entire sentence at once).

Attention Is All You Need

The proposed machine translation architecture relies solely on attention mechanisms in favor of recurrence and convolutions. This allows for greater parallelization (sidestepping the sequential dependencies of RNNs) and improved performance.

Model Architecture

The encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. Given z, the decoder generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. The decoder is auto-regressive, meaning it consumes previously generated symbols as additional input at each step. Both the encoder and decoder use only multi-headed self-attention and fully connected layers.

Each attention function maps a query Q and a set of key-value pairs (K, V) to an output. The output is computed as a weighted sum of the values, where the weight for each value represents the compatibility of the associated key with the query.

Self-attention involves less computational complexity per layer and reduced path lengths between long-range dependencies in the network.

2 Lecture

Recap

Last time we talked about attention, which is generally some kind of mask over the input. For images, soft attention is a spatial mask that applies different $[0, 1]$ weights to different areas of the image.

Sequence-to-Sequence Translation

Translation is a special case of a more general problem, **transduction**, where a sequence of things is transformed into another sequence. In a typical sequence-to-sequence model, an input sequence is fed to an RNN, and then after the entire sequence is seen, the output sequence will be generated one symbol at a time conditioned on the previous symbol and hidden state. (We want to see the entire sequence before generating any output because there might be dependencies in any part of the input sequence, and the length of the output sequence is variable.)

Note: we often reverse the input sequence. This puts the beginning of the input sequence closer to the beginning of the output sequence, so that the RNN doesn't need to remember all the way back to the beginning of the input sequence to get that important initial information.

We can use BLEU scores to compare machine translations against human-generated translations while allowing for variation. Generally speaking, BLEU looks at n-grams (subsequences) from the generated text and compares them with the reference translations. The first component of the BLEU score is the unigram score, which matches individual words from the reference and the candidate sentences. Unigram precision is defined as

$$\frac{\text{correct unigrams occurring in the reference sentence}}{\text{unigrams occurring in the test sentence}}$$

For example, if the candidate sentence is "the the the the the the the" and one of the reference sentences is "The cat is on the mat," then the modified unigram precision will be at least $2/7$.

We also consider n-grams (sequences of n consecutive words); phrases should match too. The BLEU score for n-grams, i.e. n-gram precision, is

$$\frac{\text{correct } n\text{-grams occurring in the reference sentence}}{n\text{-grams occurring in the test sentence}}$$

n-gram scores tend to capture "fluency," while unigram scores tend to capture "adequacy."

To combine scores for different n-grams, BLEU takes a weighted geometric mean of the logs of n-gram precisions up to some length, and also adds a penalty for too-short predictions.

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r, \text{ i.e. if the candidate length } c \text{ is less than the reference translation } r \end{cases}$$

Hence BLEU is roughly counting the fraction of unigrams and longer n-grams that occur in both the candidate sentence and some reference sentence.

One of the biggest limitations of the sequence-to-sequence model is that all of the information from the source sentence has to pass through the bottleneck that is the last unit of the encoder. There's a lot of information to pass across this fixed-size boundary. Also, there's usually something close to a correspondence between each input and output word, so it almost doesn't make sense to encode the entire input sequence and use it as a whole.

Attention

We can therefore try a narrower mapping using soft attention: for each output word, we can generate a relevance distribution over input words. Intuitively, an attention model is just a weighting of input data. In the case of the Bahdanau encoder, it will be a weighting of hidden states h_j from the input encoder.

One of the weaknesses of Bahdanau's model is that it has a complicated attention function that is itself a fully connected network, despite the attention map being a simple heat map on word similarity. Also, the attention path is a secondary recurrent path across time, which is undesirable. Finally, the scheme doesn't generalize easily to deeper networks.

Therefore, Luong and Manning later made the architecture more general and ensured that it would work for models with arbitrary depth (because depth helps a lot). Their attention doesn't have any recurrent paths, which simplifies the model and makes it train faster and more reliably. They also added local attention, which determines the position that aligns best with the input and then adaptively applies a bounded attention filter at that position. (Intuition: we're more likely to care about words that are near the matching word.)

Parsing as Translation

RNNs are capable of generating text that looks like input text, even if the input text is C code or LaTeX. In particular, it does a good job at figuring out things like bracket rules (i.e. tree-structured data). Accordingly, we might want to pose the parsing task as a translation task, where our input is a linear English sentence with a parse tree. If we write our parse tree in prefix tree notation, it will look like parenthesis grouping.

Then we can use neural translation to solve the reduced parsing problem.

Attention-Only Models

Traditionally, neural translation has used unrolled RNNs for both the input and the output. These are flexible, but hard to parallelize (since signal has to propagate from word to word). Also, since information is mostly flowing from one word to the next, information flow is concentrated between adjacent units. This means that long-range dependencies within sentences are hard to remember (from input to output is fine, because we have the attention map). Finally, the RNNs impose a flat, "strong adjacency" design which makes it difficult to learn hierarchical structures.

One alternative to recurrence is convolution (and hierarchies of convolution), which do a good job of handling structure at different levels. This works well enough, but the attention-only model seems to be a bit more general and powerful, so we'll talk about that one instead.

Where convolution applies fixed transform weights, self-attention "blends" using adaptive weights across the full input without transforming it. Self-attention is supposed to capture dependencies within a sentence (e.g. verbs acting on objects). A network in the style of "self-attention only" is called a "Transformer," and transforms one sequence into another without using recurrence. The idea is to have a constant path length between any two positions, a variable receptive field, and attention weighting for controlling information flow.

Benefits include support for hierarchical information flow by stacking self-attention layers and trivial parallelization. Recurrence can be replaced entirely.

Instead of recurrence, we'll have three different types of attention:

- **Encoder-decoder attention:** attention from the decoder back to the encoder positions
- **Encoder self-attention:** where positions in the encoder use input from other positions in the encoder
- **MaskedDecoder self-attention:** pay attention to what we said as output before

Therefore, we replace all previous recurrence in the encoder and decoder with end-to-end attention maps. The output attention is masked so that we only attend to words on our left (i.e. existent ones).

The Transformer model also uses multi-headed attention, i.e. multiple distinct attention weights and maps. This is done in order to replicate the behavior of a convolution, and thereby be able to transform different input pixels by different transformations. (Simple attention only blends the results of all the attended-to inputs.)

English-to-English Translation

We might want to transduct from a very long sequence to a much shorter sequence, e.g. as a summarization task. We can use a Transformer to do this, and just need to make sure it stops reasonably early when it's producing the output sequence. Note: the encoder is now very expensive (as the use case now involves very long sequences), so we'll try to live without it and instead feed in attention that's directly on the position-encoded input.

As a more concrete example, we can think of a Wikipedia article as a summary of its references and ~10 webpages per section title. We can thus train a model to produce a Wikipedia article from this kind of input.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In ICLR, 2015.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. In NIPS, 2017.