Stanford CS224N NLP with Deep Learning

Winter 2021

Lecture 7 – Machine Translation, Sequence-to-Sequence and Attention

Machine Translation

Pre-Neural Machine Translation

Machine translation first originated as a task during the Cold War and was one of the first non-numeric applications of computers. The first attempts were rule-based and extremely simple, as necessitated by the lack of power of the computers at the time.

The 1990s saw the rise of statistical methods, which attempted to learn a model P to approximate $P(y \mid x)$, where x is the source sentence and y its translation in the target language. This problem is frequently broken down into finding $P(x \mid y)P(y)$, and further into $P(x, a \mid y)$, where a represents the alignment between the sentences.

Alignment is the correspondence between particular words in the pair of sentences. Thus the probabilistic model is learnt as a combination of many factors, including the probability of particular words aligning, the fertilities of various words, and so on.
As a is a *latent variable*, it requires the use of special learning algorithms.

The second part of the model, $P(y)$, is obtained using a language model.

Neural Machine Translation

The Seq2seq Architecture

NMT is a system that carries out the complete task of machine translation, end to end. The network it relies on is called a sequence-to-sequence or seq2seq model, and it involves the use of two RNNs – the encoder and the decoder.

The encoder produces an encoding of the complete source sentence, compressed into a single vector. This representation is given to the decoder, which is a language model that generates the target sentence.

Seq2seq models have other applications, like summarisation, dialogue, parsing, and code generation.

The complete seq2seq model is an example of a *conditional language model*: it predicts the following word, conditioned on the source sentence. Mathematically, it could be expressed as

$$P(y \mid x) = P(y_1 \mid x)P(y_2 \mid y_1, x) \cdots P(y_T \mid y_1, \ldots, y_{T-1}, x).$$

Training Seq2seq Models

Seq2seq models are trained by optimising the loss of the output, compared to the gold standard target sentence. This loss backpropagates through the network, all the way to the weights of the encoder.

Frequently, multi-layer RNNs (also called stacked RNNs) are also used; the idea behind these is that the lower layers ought to compute simpler features and the higher layers ought to compute more complex features. The optimal number of layers is usually 2-4; models with more layers need *skip-connections* to be trained.

Decoding

It is common to take the most probable predicted word at each step of the decoder, but the problem with this method is that it cannot undo decisions or backtrack.

An alternative, therefore, is *beam search decoding*: we keep track of the k most probable partial translations, or *hypotheses*. Common values of k fall between 5 and 10.
The score of a hypothesis $y_1, \ldots, y_t$ is its log probability

$$\sum_{i=1}^{t} \log P_{\text{LM}}(y_i \mid y_1, \ldots, y_{i-1}, x).$$

However, this method is not guaranteed to find an optimal solution either.

The stopping criterion is another point of difference between these two methods. Greedy decoding is stopped as soon as it produces an `<EOS>` (end-of-sentence) token, while beam search decoding continues the search even after one or more paths generates an `<EOS>` token. It stops at a predefined *cutoff length*.

The final output of the beam search method is scored by the *average* log probability (rather than the sum), as longer translations tend to have lower log probabilities.

Evaluation

Machine translation systems are evaluated by a function called BLEU (bilingual evaluation understudy). It compares the machine-written translation to one or more human-written translations, and computes a similarity score based on n-gram precision ($1 \leq n \leq 4$).

Attention

Attention is an important component of modern sequence-to-sequence models. However, it has a bottleneck problem – information from the entire sentence has to be compressed into a single, small vector that is passed from the encoder to the decoder.

This problem is solved by getting *more* information from the decoder – taking *all* all the vectors given by the encoder. The core idea is to use these representations to learn which part of the input sentence to focus on in translation.