NMT notes

Damien Sileo (damien.sileo@kuleuven.be)

Introduction

The goal of machine translation is to predict the correct translation of any sentence **x** written in a source language into sentence **y** with the same meaning but written in another language called the target language..

Neural machine translation systems use a neural network to predict **x** from **y**. To do so, **x** and **y** are split into sequences $\mathbf{x_1 \dots x_N}$, $\mathbf{y_1 \dots y_T}$.

While a monolithic model $M(\mathbf{x})$ could be used to predict **y**, an encoder-decoder architecture is very widely used.
This architecture is also called sequence-to-sequence, since here it predicts a sequence of words from another sequence.

In an encoder-decoder model, the translation is separated into two tasks:

1) encoding the input sentence **x**, i.e. providing helpful representations that capture the meaning of the source sentence into a representation **h**
2) decoding the representation of the source sentence in order to predict the target sentence

Another common architectural choice is the use of a recurrent decoder (see language modeling lecture: language modeling is like machine translation without input, and a decoder is a language model that uses the encoder output as an auxiliary input).

During inference, a recurrent decoder D predict the output sequence step by step:

$\mathbf{y_{t+1}} =$ softmax ($\boldsymbol{U}$ D($\mathbf{h'_t}$, $\mathbf{y_t}$))

The decoder outputs a state $\mathbf{h'_{t+1}}$ which is mapped to the target vocabulary with a projection matrix $\boldsymbol{U}$ and a softmax.

At each timestep t, the most probable token $\mathbf{y_t}$ is selected.

This decoding process is greedy : the prediction of the target sentence is entirely dependent on the prediction of the first word(s). This can be problematic if a model starts in the wrong way and cannot yield the correct output.
To alleviate this problem, it is possible to perform a beam search instead: with a beam search, we select the top-K most plausible tokens at each timestep, which builds up a tree of possible output sentences. When all possible sentences have reached an end of sentence

token, we then select the predicted sentence that has the highest likelihood (sum of log likelihoods, with a possible corrective factor preventing the penalization of long sequences)

Note that decoding for inference is different from training. During training, we use *teacher forcing* : we use the ground truth sequence **y** as inputs for the decoder at each timestep, and not the predicted sequence.

Standard RNN encoder - RNN decoder

The encoding of the source sentence is sequential:
$\mathbf{h_{t+1}} = E(\mathbf{h_t}, \mathbf{x_t})$
The final representation of the input sentence is $\mathbf{h_T}$.

A decoder D can then use this vector to represent the input sentence to start the decoding process.
$\mathbf{h'_1} = D(\mathbf{h_T}, \mathbf{y_0})$.

However, this technique forces the encoder to condense all the information of the input sentence into $\mathbf{h_T}$. This is very challenging, especially for long sequences.

RNN encoder - RNN+Attention decoder

To alleviate that problem, it is possible to use the *local, intermediary* representations of the encoder instead of only the last one.
This can be done with a attention mechanism *A* that takes all the encoder local outputs as input, and uses the current decoder state $\mathbf{h'_t}$ to find a weighted average of the decoder input that his helpful for the decoding:
$\mathbf{c_t} = A(\mathbf{h'_t}, \mathbf{h_1} \dots \mathbf{h_T})$
The decoder can then construct a different combination of the encoder local states at each timestep to perform the decoding.

Transformer encoder - Transformer decoder

Transformers use a different architecture than RNN, where encoding is not done step by step but in parallel. However, decoding is still done step by step in most models.

Transformers work by alternating self-attention, and multilayer-perceptron transformations.

For the decoder, a cross attention (which is the same principle as the attention presented previously) is used between self attention and multilayer-perceptron transformations.

The intuition behind transformer attention is that each tokens are represented with key, values, and queries vectors. These vectors are derived from the token vectors.

The query represents the information that a token needs to update its representation.

The key represents the kind of information that a token can give to the other token to update their representation.
The value represents the actual content that a token can give to the other tokens to update their representation.

Transformers can be seen as a collaborative information sharing system, where each token provides information to the others, and get information from the others.

With multiple attention heads, there are multiple channels for sharing information.

Then, after that attention step, the shared information is synthesized with the multilayer perceptron. Through that iterative process, the correct information is distributed where it is needed to solve the training task (i.e. translation.)