An Introduction to Attention Models

Carlos Valle

Departamento de Informática
Universidad Técnica Federico Santa María

cvalle@inf.utfsm.cl

February 14, 2020

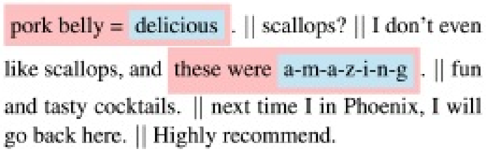
Overview

Motivation

- Attention Models(AMs), first introduced for Machine Translation (Bahdanau et al., Neural machine translation by jointly learning to align and translate,2014) is now a popular concept in neural network literature.
- Attention has been applied to important applications such as Natural Language Processing, Statistical Learning, Speech Recognition and Computer Vision.
- In several problems involving language, speech or vision, some parts of the input can be more relevant compared to others.
- For instance, in translation and summarization tasks, only certain words in the input sequence may be relevant for predicting the next word.
- Also, in an image captioning problem, some regions of the input image may be more relevant for generating the next word in the caption.

Motivation (2)

- Attention Models allow the model to dynamically pay attention to only certain parts of the input that help in performing the task at hand effectively (relevance).
- The example below, from sentiment classification of Yelp reviews, show that out of five sentences, the first and third sentences are more relevant.
- Moreover, the words delicious and amazing within those sentences are more meaningful to determine the sentiment of the review



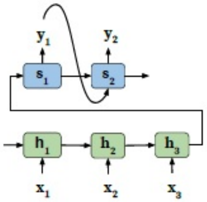
pork belly = delicious . || scallops? || I don't even like scallops, and these were a-m-a-z-i-n-g . || fun and tasty cocktails. || next time I in Phoenix, I will go back here. || Highly recommend.

Encoder-decoder for translation

- A sequence-to-sequence model consists of an encoder-decoder architecture
- The encoder is an RNN f that takes an input sequence $\{x_1, x_2, \ldots, x_T\}$, where T is the length of input sequence.
- It encodes the sequence into fixed length vectors $\{h_1, h_2, \ldots, h_T\}$.
- This means that $h_t = f(x_t, h(t-1))$, where f is the encoder RNN.
- And a vector $c = q(\{h_1, h_2, \ldots, h_T\})$. For example $c = h_t$.

Encoder-decoder for translation (2)

- The decoder is also an RNN g which takes a single fixed length vector $\{h_1, h_2, \ldots, h_T\}$ as its input and generates an output sequence $\{y_1, y_2, \ldots, y_{T}\}$ token by token, where T' is the length of output sequence.
- At each position t, h_t and s_t denote the hidden states of the encoder and decoder respectively.
- So $p(y_t | \{y_1, \ldots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$.



ED issues

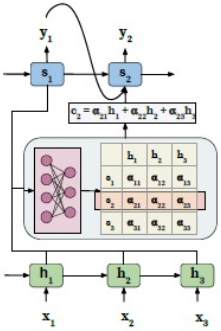
- There are two well known challenges with this traditional encoder-decoder framework:
    1. The encoder has to compress all the input information into a single fixed length vector $\{h_1, h_2, \ldots, h_T\}$. that is passed to the decoder. Then, to compress long and detailed input sequences may lead to loss of information.
    2. ED framework is unable to model alignment between input and output sequences, which is an essential aspect of some tasks such as translation or summarization. Intuitively, in sequence-to-sequence tasks, each output token is expected to be more influenced by some specific parts of the input sequence. However, decoder lacks any mechanism to selectively focus on relevant input tokens while generating each output token.

How to solve this problems?

- An attention model try to mitigate these issues by allowing the decoder to access the entire encoded input sequence $\{h_1, h_2, \ldots, h_T\}$.
- The central idea is to introduce attention weights α over the input sequence to prioritize the set of positions where relevant information is present for generating the next output token.
- The attention block in the architecture is responsible for automatically learning the attention weights α_{ij} , which capture the relevance between h_i (the encoder hidden state, which we refer to as candidate state) and s_j (the decoder hidden state, which we refer to as query state).
- These attention weights are then used for building a context vector c, which is passed as an input to the decoder.
- Now we model $p(y_j|\{y_1, \ldots, y_{j-1}\}, c) = g(y_{j-1}, s_j, c_j)$.

Attention model architecture

- At each decoding position j, the context vector c_j is a weighted sum of all hidden states of the encoder and their corresponding attention weights, i.e. $c_j = \sum_{i=1}^{T} \alpha_{ij} h_i$.
- The context vector allows the encoder to access the entire input sequence and also focus on the relevant positions in the input sequence.

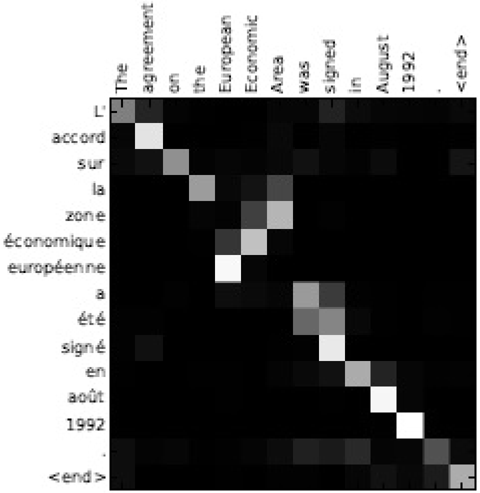


y_1
y_2
s_1
s_2
$c_2 = \alpha_{21}h_1 + \alpha_{22}h_2 + \alpha_{23}h_3$
h_1 h_2 h_3
s_1 α_{11} α_{12} α_{13}
s_2 α_{21} α_{22} α_{23}
s_3 α_{31} α_{32} α_{33}
h_1
h_2
h_3
x_1
x_2
x_3

How to train it?

- The attention weights are learned by using an additional feed forward neural network within the architecture.
- A FNN a learns a particular attention weight α_{ij} as a function of two states, h_i (candidate state) and s_{j-1} (query state) which are taken as input by the neural network.
- $\alpha_{ij} = \frac{exp(e_i j)}{\sum_{i=1}^{T} exp(e_i j)}$
- $e_{ij} = a(h_i, s_{j-1})$
- Finally, this FNN is jointly trained with encoder-decoder components of the architecture.

How well does it function?



Memory Networks

- Memory Networks were designed to alleviate the problem of learning long-term dependencies in sequential data, by providing an explicit memory representation for each token in the sequence.
- The original concept of Memory Networks comes from (Weston et al. Memory Networks, 2015). However, we will present the network proposed by Sukhbaatar et al. End-to-End Memory Networks, 2015. Since it is more realistic for common scenarios.
- It takes a discrete set of inputs $x_1, \ldots, x_n$ of dimension e stored in the memory, a query q, and outputs an answer a.
- Each of the x_i, q, and a contains symbols coming from a dictionary with V words.
- The model writes all x to the memory up to a fixed buffer size, and then finds a continuous representation for the x and q.

Input memory Representation

- This representation is an embedding of dimension d.
- Thus the entire set of $\{x_i\}$ are converted into memory vectors $\{m_i\}$.
- Usually by means of an $e \times d$ embedding matrix A.
- The query q is also embedded (again, in the simplest case via another embedding matrix B with the same dimensions as A) to obtain an internal state u.
- In the embedding space, we compute the match between u and each memory m_i as

$$p_i = \frac{exp(u^T m_i)}{\sum_i u^T m_i} = \text{Softmax}(u^T m_i)$$

Output memory Representation

- Each x_i has a corresponding output vector c_i (given in the simplest case by another embedding matrix C).
- Memory networks use attention to generate a response vector from the memory o.
- This vector is computed considering all transformed inputs c_i, weighted by the probability vector from the input:

$$O = \sum_i p_i c_i.$$

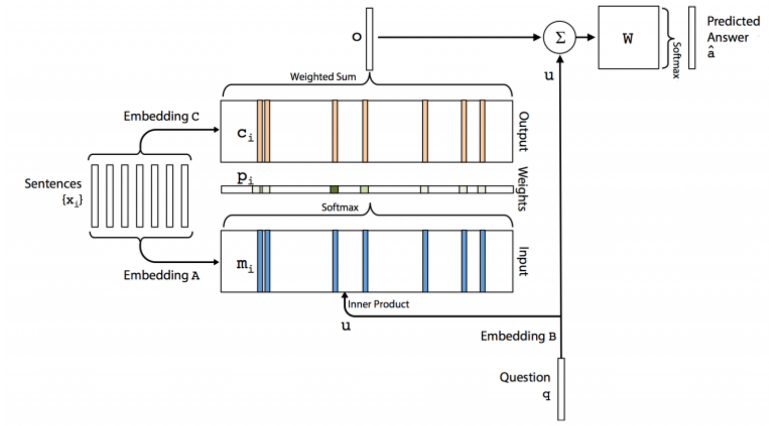
Generating the final prediction

- The final prediction is computed as a softmax of the sum of the output vector o and the input embedding u is then passed through a final weight matrix W (of size $d \times V$). This is:

$$\hat{a} = \text{Softmax}(W(o + u)).$$

- The reason why input query embedding u is considered at the end, is precisely so that we can make a module out of this architecture that can be repeated to compute more and more abstract output representations.

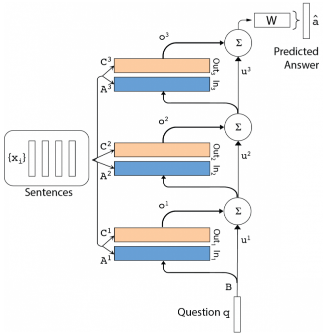
One layer memory network scheme



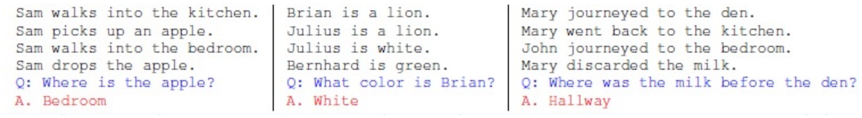
Extending this idea for multiple layers

- To do that, instead of classifying the output vector o, we use $(o + u)$ as the new u for a new round of this process, up to a fixed but unbounded number of times (in practice, less than 10).
- Each such recursive call is called a hop, and is regarded as the model reconsidering its answer over and over until it gets to a "good enough" answer (i.e. one that can be better classified).
- Then we compute final answer $\hat{a}$ only on the last hop.

Extending this idea for multiple layers (2)



How good is this approach?

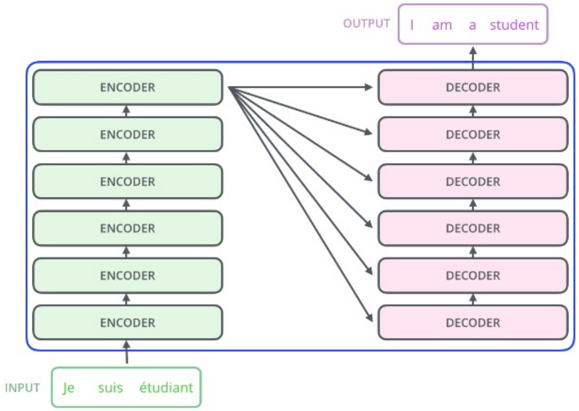

Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.
Q: Where is the apple?
A. Bedroom

Brian is a lion.
Julius is a lion.
Julius is white.
Bernhard is green.
Q: What color is Brian?
A. White

Mary journeyed to the den.
Mary went back to the kitchen.
John journeyed to the bedroom.
Mary discarded the milk.
Q: Where was the milk before the den?
A. Hallway

Transformer

- In this architecture, encoders and decoders are composed of a stack of identical layers with two sub-layers: positionwise Feed Forward Network(FFN) layer and multi-head self attention layer.

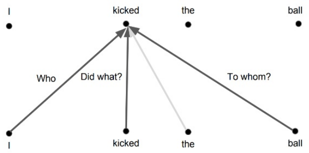


Transformer Architecture

- Position-wise FFN: The encoder phase in the Transformer generates content embedding as well as position encoding for each token of the input sequence using position-wise FFN.
- Multi-Head Self-Attention: Several attention layers are stacked in parallel, with different linear transformations of the same input. This helps the model to capture various aspects of the input and improves its expressiveness.

Self-attention

- Self-atention also known as intra-attention, is an attention mechanism relating tokens from different positions of a single sequence in order to compute a representation of the same sequence.



Self-attention (2)

- For each word, we need a query vector, a key vector, and a value vector.
- For computing the attention function on a set of queries simultaneously, packed together into a matrix Q.
- The keys and values are also packed together into matrices K and V.
- Then, we compute the matrix of outputs as:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right),$$

- where d_k is the dimension of each key.

Self-attention (3)

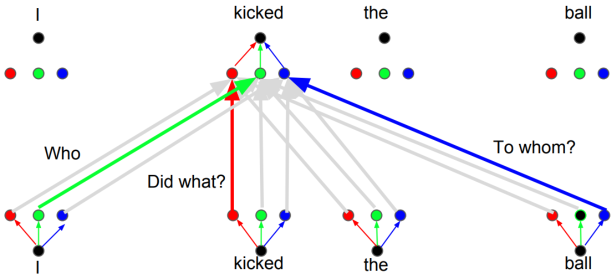
- Note that we are computing the dot product of each key between each query as in memory network approach but it is scale by $1/\sqrt{d_k}$.
- The reason is the authors suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients.

Multi-head attention

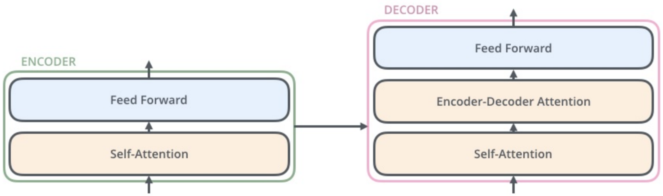
- Transformers use the concept of Multihead attention.
- The idea behind it is that whenever you are translating a word, you may pay different attention to each word based on the type of question that you are asking.
- The images below show what that means. For example, whenever you are translating “kicked” in the sentence “I kicked the ball”, you may ask “Who kicked”. Depending on the answer, the translation of the word to another language can change. Or ask other questions, like “Did what?”.

Multi-head attention (2)

- You can see more details in `http://jalammar.github.io/illustrated-transformer/`.



Encoder-decoder scheme



Where queries, keys and values came from?

- The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.
- Self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position.

Position-wise FNN

- Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

$$FFN(x) = max(0; xW_1 + b_1)W_2 + b_2$$

Positional encoding

- To make use of the order of the sequence, transformer add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks.
- Transformer use sine and cosine functions of different frequencies:

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_{model}}) \\ PE(pos, 2i+1) &= \cos(pos/10000^{2i/d_{model}}) \end{aligned}$$

  where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.

Taxonomy of Attention

- We can distinguish four broad categories of attention:
  1. Number of sequences (used to feed the AM)
  2. Number of abstraction levels (how deep is the AM)
  3. Number of positions (of the input sequence where attention function is calculated)
  4. Number of representations (of the input through multiple feature representations).

Number of sequences

- In this category we can identify:
  1. Distinctive: Is used when candidate and query states belong to two distinct input and output sequences respectively. (translation: Bahdanau et al.,2014)(summarization: Rush et al. A neural attention model for abstractive sentence summarization, 2015). (image captioning: Xu et al. Show, attend and tell: Neural image caption generation with visual attention, 2015)(speech recognition: Chan et al. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition, 2016)
  2. Co-Attention: It operates on multiple input sequences at the same time and jointly learns their attention weights, to capture interactions between these inputs. (visual question answering: Lu et al. Hierarchical question-image co-attention for visual question answering, 2016)
  3. Self-Attention (Yang et al. Hierarchical attention networks for document classification, 2016)

Number of abstraction levels

- We can distinguish:
  1. Single level: attention weights are computed only for the original input sequence.
  2. Multi levels: The output (context vector) of the lower abstraction level becomes the query state for the higher abstraction level. For example: word level and sentence level, for the document classification task (Yang et al., 2016)

Number of positions

- We can distinguish four different types:
  1. Soft (Bahdanau et al.,2014) : it uses a weighted average of all hidden states of the input sequence to build the context vector. It makes the neural network amenable to efficient learning through backpropagation, but also results in quadratic computational cost.
  2. Hard (Xu et al, 2015): the context vector is computed from stochastically sampled hidden states in the input sequence. This is accomplished using a multinoulli distribution parameterized by the attention weights. It decreases computational cost, but making a hard decision at every position of the input renders the resulting framework non-differentiable and difficult to optimize. Variational learning methods and policy gradient methods in reinforcement learning have been proposed in the literature to overcome this limitation.

Number of positions

3. Local (Luong et al. Effective approaches to attention-based neural machine translation, 2015): only a subset of source words are considered at a time. In practice, this is similar to the soft approach.
4. Global (Luong et al., 2015): all source words are attended. Is intermediate between soft and hard attention. The key idea is to first detect an attention point or position within the input sequence and pick a window around that position to create a local soft attention model.

Number of representations

- We can distinguish three different types:
  1. Single: a single feature representation of the input sequence. It is the most used approach.
  2. Multi-representational: Attention is used to assign importance weights to these different representations which can determine the most relevant aspects, disregarding noise and redundancies in the input. The final representation is a weighted combination of these multiple representations and their attention weights.
  3. Multi-dimensional: weights are induced for determining the relevance of each dimension of the input embedding vector. Computing a score for each feature of the vector can select the features that can best describe the token's specific meaning in any given context. This is especially useful for natural language applications where word embeddings suffer from the polysemy problem.

Any questions?