Sequence Models

1. [Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation]

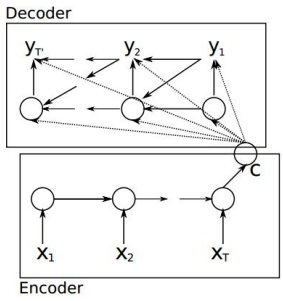
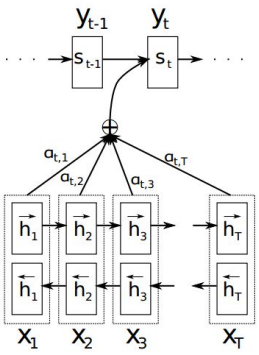


Figure 1: An illustration of the proposed RNN Encoder–Decoder.

   - Why: proposed RNN encoder-decoder structure and GRU for neural machine translation.
   - Task: Machine Translation
   - Encoder: same as the usual RNN, without any output. The hidden state of the last step as the sentence representation c.
   - Decoder: can have different length from encoder. Note that different from the usual RNN, (1) its output y_{t} depends on not only the hidden state of time t, but also previous output y_{t-1} and the sentence representation c. (2) its hidden states rely on not only the hidden states of last step, but also the sentence representation c and the output of the last step.

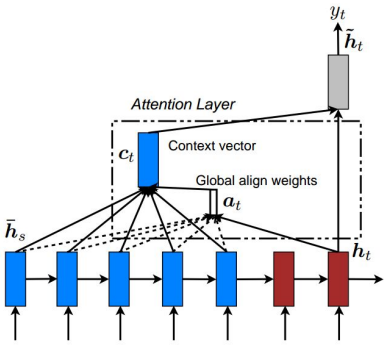
2. [Sequence to Sequence Learning with Neural Networks]
   - Proposed a different way to train the sequence-to-sequence model which is the model we use today.
   - Task: Machine Translation
   - Differences from the Cho el.at: (1) use LSTM instead of GRU. (2) add the layers of LSTM, which from 1 to 4. (3) The **most important** trick: reverse the source sentence and keep the target sentence to train the network. It seems that by reversing the source sentence, the encoder-decoder model adds more short-term dependence between source and target, which makes the decoding process easier.
   - How to deal with the exploding gradient: use the gradient clips.
   - How to deal with different length sentence: in same mini-batch, try to contain sentences of same length.

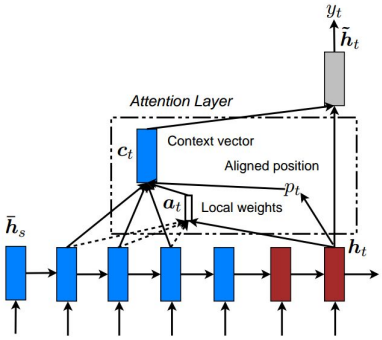
3. [Neural Machine Translation by Jointly Learning to Align and Translate]



- Why: since it turns out that when the encoder encodes the whole source sentence into a fix-sized vector, it is difficult for the model to translate long sentences. So this paper proposed **Attention**, by which the decoder can focus on different part of the hidden states of source sentence dynamically.
- Task: Machine Translation
- Different from the previous paper, this paper uses bidirectional LSTM for the encoder. The intuition of it is that when translating some word, we hope the decoder not only keep focus on the preceding context, but also the following context. After getting the hidden layers from forward and backward LSTM, they concatenate them row by row to build a overall hidden state at a time step.
- For Decoder part, when it generates some word, it (1) computes the energy function e according the decoder hidden state of last time step and each encoder hidden state. (2) uses softmax to normalize the energy function for each encoder hidden state. (3) get the attention score which is the convex combination of encoder hidden states.

4. [Effective Approaches to Attention-based Neural Machine Translation]

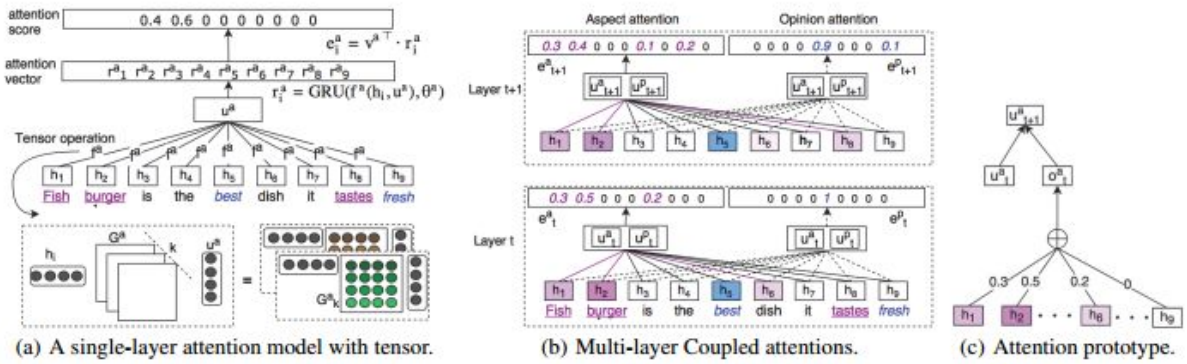




- Task: Machine translation
- Why: proposed a global attention, which is much like the attention before, and a local attention, which is more flexible and computationally cheaper than the global attention.

- Same for both global and local attention: after get the context vector c and the hidden state of the decoder, then concatenate them, and compute a new hidden state which determines the softmax score.
- Global Attention: similar to the attention proposed in paper 3, it proposes more methods to compute the score. Unlike paper 3, which uses concatenation of bidirectional LSTM as the source hidden states, this paper only uses the hidden states of the top layer of stacked LSTM as the source hidden states.
- Local Attention: first generate the position for different time step, which is the center of the focus area. Then the decoder only attends on this area. Note the width of the area is chose beforehand and thus, is fixed. How to determine the position: 1) local-m: the position is the same as the position in the decoder. 2) local-p: the position is determined by the current non-linear transformation of the position in the decoder.
- Input-feeding Approach: since in global and local attention, unlike attention in paper 3, each attention determination is independent, without taking the context into account. So in order to compensate for that, the paper concatenate the constructed hidden states, which encodes the decoder hidden states and context vector, and the input for the next step.

5. [Coupled Multi-Layer Attentions for Co-Extraction of Aspect and Opinion Terms]



(a) A single-layer attention model with tensor. (b) Multi-layer Coupled attentions. (c) Attention prototype.

- Task: aspect and opinion terms co-extraction
- Why: previous methods 1) depends on the dependent parsers, which cannot extract syntactic and dependency structures, especially for long sentences. 2) depends on handmade feature engineering, which needs specific knowledge and also very laborious.
- Single-layer attention model: explore the direct relationships between different words. 1) use word2vec and GRU to attain the token representations. 2) generate the query vector to represent the high-level information for aspect and opinion terms. 3) use the tensor operator to get the relationship between word representation and the query vector as showed at the bottom of (a). 4) use GRU on result of 3) to contain the context information for each representation. 5)

multiply each context-dependent representation with weight vector to get the final classification score.

- Multi-layer coupled attention: put the relationships between aspect and opinion terms into account, and explore indirect relationships between different words in the sentence. 1) word2vec and GRU. 2) generate query aspect and opinion vector at the same time. 3) use different tensor operator for a and p, then concatenate them together to propagate them together. Note for different classification outputs (here, we have 2, one for a, one for p), they only share word2vec and hidden states attained by GRU working on the word2vec, and query vectors. 4) in different layers, the relationship between them is as (c) , in which u_{t-1} includes the information of the entire sentence, and o_{t-1} only includes information of attended words of last layer.

6. Hierarchical Attention Networks for Document Classification

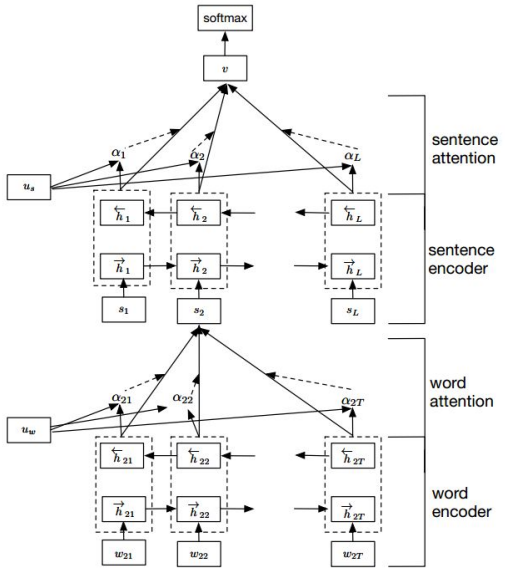


Figure 2: Hierarchical Attention Network.

- Task: document classification
- Why: the hierarchical attention network sufficiently explore the hierarchical structure of the document, which are words and sentences.
- Word-level encoder: encoder uses bi-GRU and concatenate them together used as the word annotation.
- Word-level attention: first feed the word annotation into a MLP to generate word representations. Then compute the inner product of them with a word-level query vector, and use softmax to get the attention score for each word. Then the sentence vector is the linear combination of these attentional word representations.
- Sentence-level encoder and attention model: like these in Word-level encoder and attention. Finally, the hierarchical model generates the vector for the final classification problem.

7. [Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models]

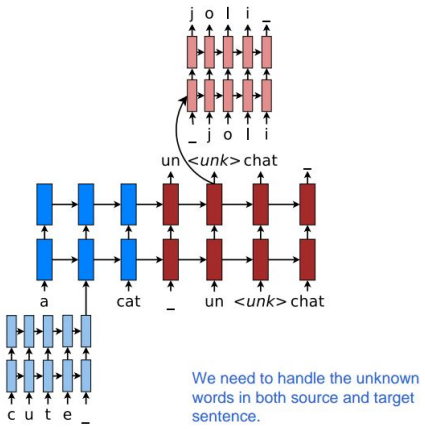


Figure 1: **Hybrid NMT** – example of a word-character model for translating "a cute cat" into "un joli chat". Hybrid NMT translates at the word level. For rare tokens, the character-level components build source representations and recover target <*unk*>. "_" marks sequence boundaries.

- Task: Machine Translation
- Why: previously method only translates those words which are highly frequent. For those uncommon words, they represent them as unk word. After obtaining the target sentence, unk words are replaced by the corresponding words in the look-up dictionary directly. Those methods ignore the relationships between words, and different characters between different languages. Also, translation only on the character-level does not take the word relations into account.
- Backbone: for the word-level encoder decoder, this paper uses the structure in paper 4 with the global attention and bilinear form score function. <unk> for unknown words in source and target, one for each language..
- Source Character-based representation: the paper uses character-level LSTM to get the <unk> representation for source sentences. Note that to solve the problem that different sentences have different unknown word positions in the same mini-batch, the paper initializes all hidden layers using zeros so that the representation of each <unk> is independent and unique for same word, so that we can precompute the representation of those unknown words before actual training. Note the same unknown word's representation is the same.
- Target character-level generation: instead of using the same-path method, which uses directly the hidden state input for the softmax layer, the paper uses a separate-path method, which mimic the states in the same-path to generate another hidden states which has the same form. According to the paper, this way alleviate the pressure of the hidden state, which encodes the relationship between context vector and hidden state for each word in the source sentence, predicting <unk> word and the exact character-level decoding at the same time.

Note since here the decoding is context-dependent, so the representation for each word token of same word is not necessarily the same. This separate-path hidden state only initializes the hidden layer of the first layer, while other layers are initialized by zeros.

- Highlight trick: for character-level decoder, the paper decouples it with the word-level decoder. If the paper used the last hidden state of the character-level decoder as the <unk> representation and fed into the next word-level decoder, then since different sentences have different positions of unknown word, so the character-level decoder cannot be trained in mini-batch. In practice, the paper feed the <unk> word embedding in word level into the next time step, same as there was no character-level decoder. Here, <unk> word embedding is in simple-path and fed into next time step. At the same time, we can get the hidden states in the separate-path. After the mini-batch, we feed all those hidden states into the character-level decoder to decode them in the batch mode.

8. Long Short-Term Memory-Networks for Machine Reading

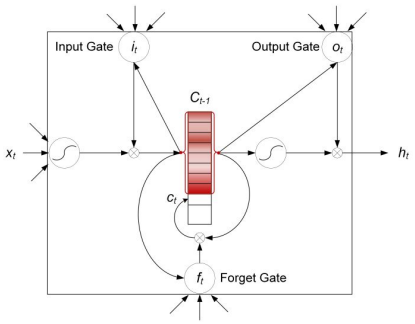


Figure 2: Long Short-Term Memory-Network. Color indicates degree of memory activation.

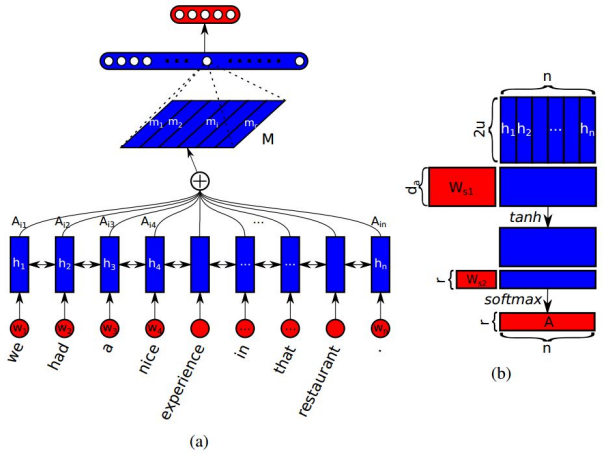
- Task: Machine Reading, compressing a sentence.
- Why: in terms of the shortcomings of RNN(1. RNN works in a way of Markov chain manner, on the assumption that each cell can summarize prior context very well, but in practice, it is hard to store all information in a single dense vector, 2. Cannot handle the inherent structure of sentences since it reads the text token by token), the paper utilizes memory and attention mechanism to overcome them.
- To solve the aforementioned problems of LSTM, the paper modifies the traditional LSTM: instead of only storing and attending hidden states of earlier time, it also stores and attends the memory.
- Difference between LSTMN attention and previous traditional attentions: here, since LSTMN is used for extract information from and compress the whole sentence, it only attends on the previous hidden states of the sentence it is **currently processing** instead of the source hidden states which has been processed. Therefore, it is **self-attention**.
- Difference between LSTMN and LSTM: 1. Besides maintaining the attention part of hidden states when compress the input sentence, LSTMN also stores and

attends the previous memory. 2. Memory has the same attention score with Hidden states, but the score only depends on previous hidden states. 3. The gates function does not rely on the hidden states of the last time step, intead, it depends on the weighted hidden states. 3. New memory depends on the weighted prior memory and the newly-constructed memory at this time step, which depends on the input and the weighted hidden states. 4. In traditional LSTM, the hidden states will be fed into the next time step to decide the gates and memory. In LSTMN, it only determines the attention score. Gates functions relies on the weighted hidden state. In summary, the memory and hidden state in the last time step have been substituted with the attentionally weighted memory and hidden state.

- How to use this as a seq-to-seq model for the machine translation: maintain the hidden states and memories for source and target sentences. The LSTMN seq-to-seq model fuses the inter- and intra-memory. In the decoder, the memory needs to take weighted memory in the **source** into account.

9. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention
    - Task: image captioning
    - Why: the paper proposed two attention model: “hard” attention, which is trained by REINFORCE, and “soft” attention, which is trained by backpropagation.
    - Broad view: Broadly speaking, the network draws on the idea of seq2seq, encoding the whole image as a sequence of vectors, each of which stands for a position. CNN and LSTM are used as encoder and decoder, respectively. How to compute the attention score is similar to paper 3. Difference between hard and soft attention lies in the ways to computing the context vector.
    - How to get the sequence of vectors in image: use VGG as the feature extractor to get the feature maps from the fourth conv layer. Each spatial pixel in the feature map is a vector, which is of length of the number of channels.
    - Hard Attention: modeled as the one-hot variable of each position as a multinoulli distribution which is parameterized by the output of softmax in paper 3, then use the Monte Carlo sampling to approximate this distribution. Note hard attention only focuses on one position.
    - Soft Attention: as before, the context vector is the convex combination of all the image vectors. Also, the paper uses backpropagation to train it.

10. [A structured self-attentive sentence embedding]



- Task: Author profiling, sentiment analysis and textual entailment. Actually, this model is to **encode** a whole sentence.
- Why: Using a traditional LSTM to encode a sentence is not optimal, since: 1) it needs to carry on every information in the whole sentence, so it is hard to encode a very long text sequence. 2) the previous attention model primitive summation of hidden states is suboptimal, and cannot attend on different semantic parts. Therefore, this paper proposed 1) a **self-attention** model with **multiple** vectors representing a sentence, and 2) a penalization term for rendering attention model focus on different semantic parts.
- Self-attention model: the paper uses bi-LSTM model to make independent word related, then use (b) to get the attention vector if r=1. If r=2, the model can get more than one attentional scores, each of which keeps focus on different semantic meaning of the text. Please Note that as parameters in score function, all dimensions should not be related to the sentence length. Then use the fully connected layer to get the representation of the whole sequence from the representation matrix M.
- Penalization term: To make different attention vector m keep focus on different part, the paper adds an another regularization term besides the loss function for the downstream task. This Frobenius norm regularization enforces the dot product of different attention score to approximate to zero.
- Two ways to visualize the attention: since we have more than one attention score vector for each sentence, so there are two ways to visualize the result: 1) draw heat map for each score vector. 2) sum up along column in order to get a attention vector, then normalize the score.