Sequence to Sequence Modelling

Patrick Kahardipraja

September 2019

Mapping of a sequence to another sequence is an important paradigm because of the vast amount of problems that can be formulated in this manner. For instance, in automatic speech recognition (ASR), chunks of speech signals can be mapped to sequence of phonemes while in machine translation, a sequence of words in one language can be mapped to another language. Interestingly, many other tasks such as text summarization, question answering, and image caption generation can be phrased as a sequence to sequence problem. In this paper-style essay, I will attempt to distill how sequence to sequence learning works and the motivation behind it, with a particular focus on machine translation.

Introduction

Prior to neural machine translation (NMT), phrase-based stastistical machine translation (SMT) systems are widely used as it offers reliable performance. Despite its success, most of them are extremely complex and require a huge amount of effort, as it is often tailored to a specific language pair and do not generalize well to another languages. In a phrase-based SMT system,

a lot of feature engineering are required in order to capture a specific language phenomena, which prompt researchers to explore another approach. Furthermore, phrase-based systems still experience difficulty in capturing long-term dependencies.

The resurgence of deep neural networks (DNNs) in early 2010s, thanks to faster, parallel computation using GPUs and availability of large and high-quality datasets, bring a new wave of enthusiasm in deep models. With the capability to learn features automatically with multiple, hierarchical representation, DNNs achieve excellent performance on difficult tasks in computer vision [AlexNet] and speech recognition []. Albeit powerful, DNNs has its own limitation, as it requires input and output vectors with a fixed dimension and thus not suitable for sequence to sequence problem whose lengths are unknown beforehand. In addition, DNNs also do not generalize well across temporal patterns, because each neuron has its own specific connection and as a result, a single pattern may look totally different at different timesteps.

The natural remedy for this problem is to look onto recurrent neural networks (RNNs), as it allows operations over sequences of vectors. However, mapping using RNNs typically have one-to-one correspondence between the input vectors and the output vectors. It also has another problem, as the input and output sequences can have different lengths and non-monotonic alignments. Standard RNN architecture is also not reliable for learning long-range dependencies due to the vanishing gradient problem. This issue is addressed by Sutskever et al. [Seq2Seq], where they introduce a novel and straightforward method to solve general sequence to sequence mapping using Long-Short Term Memory (LSTM) architecture. With the success of sequence to sequence learning in machine translation tasks, research in neural machine translation continue to thrive, eventually resulting in many significant improvements such as attention mechanism [Bahdanau] and subword units to deal with rare words [WordPiece]. But, before delving in too deep,

I will give some brief insight into the mechanism behind RNN and LSTM in the next section.

Recurrent Neural Networks

Recurrent neural networks [Rumelhart] are type of neural network that is able to process arbitrary sequential input via combination of its internal state and input vector. At every timestep t, the hidden state vector h_t is overwritten as a function of the hidden state at the previous timestep h_{t-1} and the current input vector x_t. The input vector x_t itself could be a representation of t-th word in a sentence, which is usually obtained using pre-trained word embeddings [GloVe, Word2Vec, ElMo]. The hidden state of RNNs can be perceived as a memory with a fixed dimensionality that can be tuned, containing distributed representation of the processed input sequence up to time t.

In a RNN, the forward step function consists of an affine transformation followed by a non-linear activation function. The hidden state then can be used to make predictions:

$$h_t = a(W^{(x)}x_t + W^{(h)}h_{t-1} + b_h) \tag{1}$$

$$y_t = g(W^{(y)}h_t + b_y) \tag{2}$$

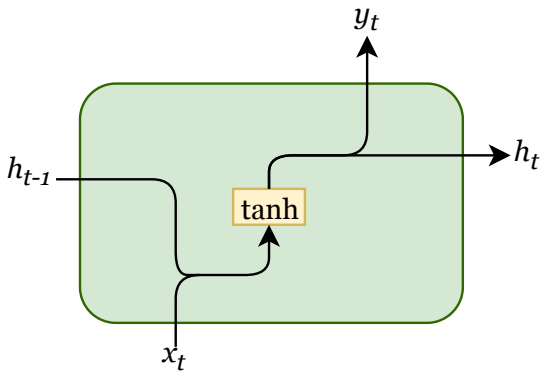
where in a typical application, a is the hyperbolic tangent function and g is the softmax function.

Although proven to be effective, RNN still has its own shortcoming. The main problem with RNN is that during training, magnitude of gradient can get weaker or stronger exponentially when backpropagating the error through time, especially with long sequences [Hochreiter, Bengio]. This phenomena

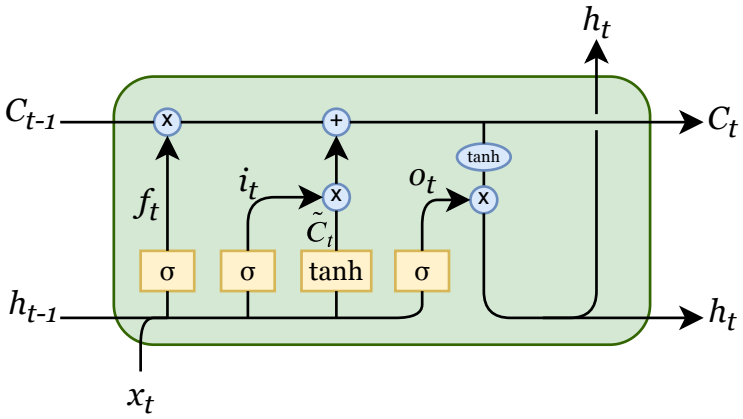
is called vanishing or exploding gradient problem, which causes RNN model to experience difficulty when handling "long-term dependencies" that occur in a sequence.

Long Short Term Memory (LSTM) architecture [Hochreiter and Schmidhuber] addresses the problem of "long-term dependencies" by integrating a memory cell that is capable to memorize state that span over long sequences of time. The memory cell is controlled by gates, which have the ability to regulate how much information are added or removed in the memory cell. This means that while in a RNN a completely new hidden state is computed at every new timestep, in LSTM the hidden state is not competely overwritten, and updated according to the memory cell. The architecture of both RNNs and LSTMs are depicted in Figure 1.

A LSTM unit consists of 3 gates (input gate i_t, forget gate f_t, output gate o_t), memory cell C_t and hidden state h_t. In a high-level sense, the input gate decides how much and which values will be updated, the forget gate controls the amount of information to be forgotten in the previous memory cell, and the output gate decides the hidden state by filtering the internal memory cell for each timestep. Each gate produces vectors, where their values are between 0 (completely closed) and 1 (completely open) using the sigmoid activation function.



(a) RNN unit



(b) LSTM unit, colah

Figure 1: Architecture of RNN (a) and LSTM (b)

The formula of LSTM is described with the following equations:

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1} + b_i) \tag{3}$$

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1} + b_f) \tag{4}$$

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1} + b_o) \tag{5}$$

$$\tilde{C}_t = \tanh(W^{(c)}x_t + U^{(c)}h_{t-1} + b_c) \tag{6}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{7}$$

$$h_t = o_t \odot \tanh(C_t) \tag{8}$$

where x_t is the input vector for timestep t, σ is the sigmoid activation function and $\odot$ denotes the Hadamard product of matrices.

Variants of LSTM

Beside the standard LSTM architecture in literature that is introduced above, there also exist several popular variants which are commonly used. One of the variations which is introduced by [Gers, Schmidhueber] use "peephole connections". These connections allow the gates to look into the memory cell state in order to learn precise and stable timings.

Other notable LSTM variants is the Gated Recurrent Unit (GRU), introduced by [Cho et al.]. Unlike LSTM, GRU is less complex and requires less computation. GRU only have 2 gates, the update gate that decides how much information will be transferred from previous and candidate hidden state to the current one and reset gate that controls to what extent the previous hidden state will affect the candidate hidden state. In this manner, the update gate can be thought as a combination of input and forget gates of LSTM unit. This architecture also merges the internal memory cell and hidden state of LSTM into a single hidden state.

Related Works

The first model that is able to map a sentence into a vector and then to its translation is introduced by [Kalchbrenner et al.]. The model, which they called Recurrent Continuous Translation Models (RCTM), is composed of 2 separate parts: convolutional neural network (CNN) for modelling the source sentence as the encoder and recurrent neural network for translation generation as a language modelling task, conditioned on the source sentence

as the decoder. With this approach, the encoder can capture all the information contained in the source word representations and create a representation of the source sentences. The representation for the source sentences also restraint the generation of the target words in the language modelling phase.

The Recurrent Continuous Translation Models estimate the probability distribution over the sentence in the target language given sentences in the source language. Suppose that there exist a target sentence $f = f_1, f_2, ..., f_m$, which is a translation of source sentence $e = e_1, e_2, ..., e_n$. Then $p(f|e)$ can be obtained with the formula:

$$p(f|e) = \prod_{i=1}^{m} p(f_i|f_{1:i-1}, e) \tag{9}$$

As can be seen in the formulation above, the model estimates $p(f|e)$ by calculating the conditional probability $p(f_i|f_{1:i-1}, e)$ for every translated word occuring at position i, given the preceding generated words $f_{1:i-1}$ in the target sentence and the source sentence e. Conditioning the translation model to the preceding target words also ensure that it incorporates the target language model [Kalchbrenner et al.].

In RCTM, prediction of the target sentence use a language model based on a recurrent neural network [Mikolov et al.]. The recurrent language model (RLM) predict the i-th word of the target sentence depending on all the previous generated words $f_{1:i-1}$, making no Markov assumption about the words dependencies in the target sentence. However, using the standard RNN architecture makes the prediction to be strongly affected by words close to f_i and weakly influenced by long-range dependencies that occur in the target language due to the nature of RNN.

The RLM models probability of a sequence of words f that occur in a language, which is denoted by $p(f)$. The equation for $p(f)$ is almost identical

to Eq. 9 :

$$p(f) = \prod_{i=1}^{m} p(f_i | f_{1:i-1}) \tag{10}$$

It also contains a vocabulary V for words f_i of the language and 3 transformation matrices for input vocabulary $\mathbf{I}$, recurrent transformation $\mathbf{R}$ and output vocabulary transformation $\mathbf{O}$. Each word $f_k \in V$ is distinguished by one-hot vector $v(f_k)$. The computation then proceed as follows:

$$h_1 = g(\mathbf{I} \cdot v(f_1) + b_h) \tag{11}$$

$$h_{i+1} = g(\mathbf{R} \cdot h_i + \mathbf{I} \cdot v(f_{i+1}) + b_h) \tag{12}$$

$$o_{i+1} = \mathbf{O} \cdot h_i + b_o \tag{13}$$

and the probability distribution is obtained using the softmax function,

$$p(f_i = v | f_{1:i-1}) = \frac{\exp(o_{i,v})}{\sum_{v=1}^{V} \exp(o_{i,v})} \tag{14}$$

where g is a nonlinearity and $\mathbf{I} \cdot v(f_i)$ is a continuous representation of word f_i.

There are 2 types of conditioning architecture in RCTM using CNN, using convolutional sentence model (CSM) and convolutional n-gram model (CGM). The CSM creates sentence representation in a bottom-up manner, using n-grams representations in the sentence itself. The hierarchical structure that is created by the model act quite similar like a parse tree in a implicit way. Using this type of structure, the model is able to capture the small, local representations in the lower layers of the model and more globally in the upper layers of the model as it spans more n-grams that comprise the sentence representations. This model also offers several advantages as it does not rely on a parse tree [Grefenstette 2011, Socher 2012]. As there exist

many languages for which highly accurate and reliable parsers are not available, this model can still be robustly applied. Furthermore, the distribution of translation probability is learned by the model and does not depend on the chosen parse tree.

Using the continuous representations of words in the sentence, CSM models the representation of the sentence by applying sequence of one-dimensional convolution operations. The kernel of the convolutional layers is able to learn pattern within n-grams that convey syntactic, semantic or structural information relevant for constructing the sentence representation. After several convolution operations, the sentence vector representation $\mathbf{e}$ is created at the topmost layer of the network for the source sentence e. This vector representation is then used in the RLM, after applying learned sentence transformation $\mathbf{S}$. However, this model has a bias as the RLM tend to predict target sentences with shorter length. The sentence vector representation $\mathbf{e}$ also constraint the target words, which is counterintuitive as it often occurs that the target translation has a strong dependencies on some parts of the source sentence and less on the other parts. In order to address these aspects, [Kalchbrenner et al.] also proposes the convolutional n-gram model as another conditioning architecture.

The CGM is a truncated version of the CSM, where the n-grams representation is extracted from a specific CSM layer for a chosen value of n. Using the n-grams representation of the source sentence e, the CGM can also be inverted to obtain representation for the target sentence f with deconvolutional operation, where the length of the target sentence m is estimated using Poisson distribution. This inverted CGM can also be thought as the truncated version of the inverted CSM for sentence length m. Before the inverted CGM unfolds the n-gram representation to a target sentence, a learned translation transformation $\mathbf{T}$ is applied. The reconstructed vector for the source sentence representation is then added in an incremental man-

ner to the corresponding hidden state h_i in the RLM to predict the word f_i in the target language. The issue that is addressed with the CGM model, where generation of the target words can now incorporate different parts of the reconstructed source sentence representation, is also later improved by [Bahdanau et al.], where they propose attention mechanism to learn soft-alignment between the source and target sentences.

While the model proposed by [Kalchbrenner et al.] works quite well in for rescoring translation hypotheses from SMT system and computing perplexity of reference translations, using CNN as encoder means that the ordering of the words are not preserved. In an almost similar manner to this approach, [Cho et al.] attempt to map source sentence to a fixed vector representation then back to the target sentence, but with two RNNs as encoder and decoder. The encoder RNN reads each word in the source sentence sequentially until it reach the end of the sequence, which is marked by an end-of-sequence symbol. The hidden state of RNN after completely reading the source sentence is then encoded to a context vector $\mathbf{c}$, which contains the summary of the whole source sentences. Consider source sentence x with length N and target sentence y with length M. The encoder is formulated as follows:

$$h_t = RNN_{enc}(h_{t-1}, \text{emb}(x_t)) \tag{15}$$

$$\mathbf{c} = \tanh(\mathbf{V}_{enc} \cdot h_N) \tag{16}$$

where $\text{emb}(x_t)$ is a continuous representation of input word at timestep t and $\mathbf{V}_{enc}$ is a learned transformation for the encoder.

On another part, the decoder RNN is a recurrent language model, conditioned on all the previous generated target words and the context vector $\mathbf{c}$. It is computed as follows, where the decoder hidden state is initialized using

the context vector:

$$h'_0 = \tanh(\mathbf{V}_{dec} \cdot \mathbf{c}) \tag{17}$$

$$h'_t = RNN_{dec}(h'_{t-1}, \text{emb}(y_{t-1}), \mathbf{c}) \tag{18}$$

where $\mathbf{V}_{dec}$ is a learned transformation for the decoder. The probability distribution of target words are obtained from a softmax function applied to the output of a feedforward neural network that consists of a single intermediate layer with maxout units [Goodfellow], using the decoder hidden state, context vector and target word generated from the previous timestep as inputs.

For words representation, one-hot encoding is used to distinguish words in the vocabulary, which are then projected twice, yielding a 100-dimensional embedding for each word. Both of the encoder and decoder use GRU instead of LSTM as it is easier to compute and implement. The encoder and decoder components of the model are then trained in an end-to-end fashion in order to estimate the conditional probability of the target sentence given the source sentence:

$$p(y|x) = \prod_{t=1}^{M} p(y_t|y_{t-1}, y_{t-2}, ..., y_1, \mathbf{c}) \tag{19}$$

In their paper, they focus on integrating the RNN encoder-decoder pair for conventional phrase-based SMT system. The trained RNN encoder-decoder pair is used to rescore phrase pairs between the source and target sentences. This new score is then added to the existing phrase table and used as additional features in the log-linear model for the phrase-based SMT system. Furthermore, the model is also able to produce well-formed phrases in the target language independently without any influence from the actual phrase table.

Seq2Seq Model

The works of [Kalchbrenner] and [Cho] however focus only on rescoring and not direct translation, although [Cho] mentions the possibility of replacing phrase table with phrases generated by the RNN encoder-decoder model. [Sutskever et al.] introduces a sequence-to-sequence model, often called "Seq2Seq", which achieves success to produce direct translation from English to French. The Seq2Seq model involves two RNNs as encoder and decoder, similar to [Cho].

The task of the encoder is to process the input sequence (source sentence in this case) in a sequential manner until its reaches the end-of-sentence (EOS) symbol to generate a fixed-dimensional context vector $\mathbf{c}$ that represents the input sequence. It is also important to note that the encoder of Seq2Seq model processes input sequence in reverse. The idea behind reversing the input sequence is to reduce the "minimal time lag" [schmidhueber] problem. By doing this, the distance between first few words in the source and target languages are now much closer to each other, therefore making it easier for backpropagation to relate the source sentence and the target sentence. This method make it easier for the decoder to generate first few words correctly, and in turn improve the probability of generating the correct target translation. The illustration of the input sequence reversal is depicted on Fig..

The decoder architecture of the Seq2Seq model is almost similar to the decoder in the model proposed by [Cho et al.]. It is a recurrent language model, conditioned on all the target words that have been generated until the current timestep and the context vector **c** that encapsulates all the information contained within the input sequence. The main difference between the Seq2Seq model and the RNN encoder-decoder model is that in Seq2Seq model, the context vector **c** is only used for initializing the decoder hidden state whereas in the RNN encoder-decoder model, **c** is used for decoder hidden state initialization and also for target words generation at all timestep, which is indicated in Eq. 18.

When the source sentence is already encoded into **c** and the decoder is set up with the context vector, a special symbol is passed on to the decoder to mark the generation of the output sequence. In the paper, this is an EOS symbol that signifies the end of the input sequence. Afterwards, the decoder will continuously generate words in the target language until it generates the EOS symbol. The architecture of Seq2Seq model is depicted on Fig..

The encoder and decoder architecture of the Seq2Seq model use LSTM as it is capable of learning long range temporal dependencies. Compared to GRU, LSTM cells consistently performs better on language modelling [Jozefowicz et al. An Empirical Exploration of Recurrent Network Architectures] and machine translation tasks [Thang Luong Massive Exploration of NMT architectures]. The performance of LSTM can also be improved further by setting a large bias to the forget gate. To increase the model capacity, the implementation of Seq2Seq use deep LSTMs (4 layers in the paper) for the encoder and the decoder where the hidden state of encoder final layer at the last timestep is used as context vector $\mathbf{c}$ to initialize the first layer of the decoder LSTM. Each additional layer of the deep LSTMs was also found to be able to reduce the perplexity by 10% [Sutskever et al.].

For Seq2Seq, the goal of the model is to estimate the conditional probability of the input sequences given the output sequences:

$$p(y|x) = \prod_{t=1}^{M} p(y_t|\mathbf{c}, y_1, ..., y_{t-1}) \tag{20}$$

where M is output sequence length which may differ from input sequence length. The formulation of the conditional distribution is almost similar to the RNN encoder-decoder model (Eq. 19) but with minor difference, where here $\mathbf{c}$ is only used to initialize the hidden state of the decoder LSTM and not connected to the decoder at all times. Each distribution $p(y_t|\mathbf{c}, y_1, ..., y_{t-1})$ in Eq. 20 is produced by a softmax operation over all the existing words in the vocabulary.

Both the encoder and decoder LSTMs are trained in an end-to-end approach to maximize the conditional log-likelihood of the correct translation

T given the source sentence S, which is defined as the following:

$$1/|\mathcal{S}| \sum_{(T,S)\in\mathcal{S}} \log p(T|S) \tag{21}$$

where $\mathcal{S}$ is the training set. The most likely translation is approximated by using a left-to-right beam search decoder instead of a greedy search decoder, as greedy decoding only generate most probable word on each timestep which may reduce the overall quality of the predicted translation. During the decoding process, a B number of most probable partial translations (hypotheses) are tracked according to beam size B. Each partial hypothesis is extended on each decoder step with all the words in the vocabulary while discarding all but the B most likely partial hypotheses. When the decoder produces the EOS symbol for a hypothesis, it is then removed from the beam search and added to the list of complete hypothesis, while other hypotheses are still explored by the beam search. Additionally, the score of each complete hypothesis is normalized by its length to produce translations with the highest score, due to the bias of the decoding method to select shorter translations.

Seq2Seq is the first model that prove that the performance of a pure neural machine translation system is able to surpass a conventional phrase-based SMT system with a significant margin [Sutskever et al.]. The model also surprisingly able to generate high quality translations even on very long sentences, which was initially thought to be difficult due to limited capacity of LSTM memory cell. Furthermore, vector representations that are produced by the encoder component of the Seq2Seq model are aware of the syntactic and semantic information conveyed in the source language.

Other Aspects of Seq2Seq and Neural Machine Translation

The aforementioned works [Cho, Kalchbrenner, Sutskever] have drastically changed the field of machine translation. A neural machine translation system offers better performance in term of capturing phrase similarities and long-term dependencies compared to a phrase-based SMT system. In terms of complexity, a NMT system also requires lesser engineering effort as it requires no feature engineering and applicable to all language pairs without a need for language-specific tailoring. On the other side, NMT approach is less interpretable due to the continuous representations and non-linear properties of neural networks, which causes difficulty in associating hidden states with the language structures and understanding of the translation process.

Seq2Seq, as one of the most promising NMT system at the time of its inception, also still has some limitations on its own. First of all, the model only learn long-term dependencies in a left-to-right direction, which means that information from words after the current word are not taken into account. This problem can be fixed by using bidirectional RNNs [Schuster] for the encoder component to read the input sequence from both directions with 2 different RNN cells. Final hidden states from the left-to-right and right-to-left RNNs are concatenated or summed to generate a final context vector, which in turn used to initialize decoder hidden state. Illustration of bidirectional encoder using LSTM cells is depicted on Fig..

Another issue with Seq2Seq and NMT system in general is that the model experiences difficulty in correctly translating rare words because of the limited vocabulary size, which is often attributed to the computationally expensive operation of the softmax function. As a result, a special "UNK" symbol is used to represent every out-of-vocabulary (OOV) word during the

decoding process. [Minh-Thang Luong et al.] attempt to address this issue by introducing the idea to annotate the training data with alignment information between source and target sentences. The Seq2Seq model is trained on the annotated corpus to keep track of the origins of unknown words that is generated during the translation process, creating links between sentence pairs. The alignment links are used to construct a dictionary, which is used in the post-processing step to replace "UNK" symbol with translation of its corresponding source word. In the case where the translation does not exist in the dictionary, then identity translation is applied.

In the direction of using units smaller than words to solve rare words problem, [Sennrich et al.] proposed to encode rare and unknown words as sequences of subword units to achieve open-vocabulary translation. Words that appear in the corpus are segmented using adapted version of Byte Pair Encoding (BPE), where characters are merged instead of bytes. This method initialize the vocabulary with characters and expand the vocabulary content with most frequent character n-grams, which are eventually merged into a single symbol. Subsequently, word segments in the vocabulary can be used to train a NMT system. Compared with word-level approaches, this technique is able to generate unseen words in the training data. This makes it more appropriate for translation tasks, where morphological changes are often required, in particular for languages that use compounding and agglutination widely.

[Minh Thang-Luong, Manning] also proposed a hybrid model using mixture of words and characters, where the translation task is mostly performed at the word level while utilizing the character components to deal with rare words.

Conclusion