Week12 Pre-Report: Overview of Sequence to Sequence Learning with Neural Networks (Seq2Seq) and Empirical Evaluation of Gated Recurrent Neural Networks (GRU) on Sequence Modeling

Euijin Hong[1]
[1]Department of Electrical and Electronic Engineering, Yonsei University.

1 Introduction

A sequence to sequence model[10] (Seq2Seq) is used for generating the sequence of a different domain from an input sequence. The model proposed in the paper uses a multilayered Long Short-Term Memory[9] (LSTM) to map the input sequence to map the input sequence to a vector with a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. The main result in this paper is that on an English to French translation task from the WMT'14 dataset and learning sensible phrase and sentence expressions, while showing that reversing the order of the words in the source sentences improved the performance of the proposed deep LSTM.

On the other hand, gated recurrent unit[3, 5] (GRU) is an alternative of LSTM, which gives a solution to a long term dependance problem of LSTM and reducing the computation for updating the hidden state. So, the performance of GRU is almost identical to that of LSTM, but shows advantage in terms of structural simplicity. The major difference of GRU and LSTM is the number of gates, where LSTM has three gates (forget, input, and output gate) but GRU has only two gates (update and reset gate). The given paper compares the performance of LSTM and GRU, while evaluating them on polyphonic music modeling and speech signal modeling tasks. The research found out that advanced recurrent units are better than traditional recurrent units, and the performance of GRU is comparable to that of LSTM.

2 Theory and Configuration of Seq2Seq Model

2.1 Theory

As mentioned before, Seq2Seq model can be applied in tasks to generate sequences with different length and format from those of the input sequence, especially for the pairs of complicated and non-monotonic relationships. Although this can be implemented by double layers of RNN, its performance is limited according to the long term dependencies[2, 4, 6, 8]. However, LSTM is known to solve this problems with long range temporal dependencies. The LSTM aims to estimate the conditional probability $p(y_1,...,y_{T'}|x_1,...,x_T)$ where $(y_1,...,y_{T'}$ is the corresponding output sequence of input sequence $x_1,...,x_T$, which differs in terms of length T and T'. In LSTM, the conditional probability can be obtained by first receiving a fixed dimensional representation v, provided by the previous hidden state, and then computing with the LSTM formulation in [7], where the initial hidden state is set to v. The model requires an 'end of sentence' symbol ("<EOS>") which enables the model to define a distribution over sequences of all possible lengths. The "<EOS>" in the input sequence indicates as the beginning of output generation, until the "<EOS>" comes out from the output sequence.

$$p(y_1,\ldots,y_{T'}|x_1,\ldots,x_T) = \prod_{t=1}^{T'} p(y_t|v,y_1,\ldots,y_{t-1})$$

The proposed Seq2Seq model in the paper[10] is comprised of two different LSTMs, one for the input sequence (the encoder) and the other for the output sequence (the decoder). This allows the model to increase its number of parameters negligibly and multiple language pairs trained simultaneously. Also, the researchers found out that deep LSTM outperforms the shallow LSTM, so they used four-layered LSTM. Furthermore, researchers mapped the input and output sequences in a reverse order, c,b,a as α,β,γ, where the translation is a,b,c to α,β,γ for instance. This ensures a to have close proximity to α and so on, which makes SGD to easily 'establish communication' between the input and the output.

2.2 Configuration

The encoder of Seq2Seq model receives all the words of the input sentence sequentially and compresses all this word information into a single vector, which is called the *context vector*. Once the information of the input sentence is compressed into a single context vector, the encoder sends it to the decoder. The decoder takes the context vector and outputs translated words one by one sequentially as shown in Figure 1.

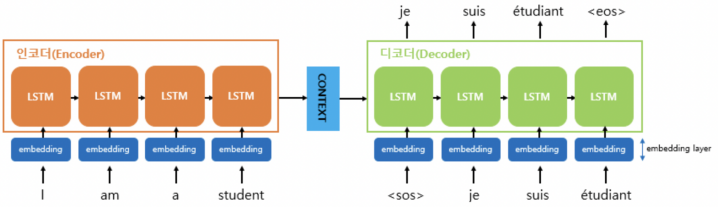

Figure 1: The overall process of generating an output in a Seq2Seq model.

ENCODER: If we look closer at the encoder, it undergoes word tokenization to split the input sentence into individual word tokens. Each word token then becomes the input at each time step of the LSTM cell. After receiving all the words as input, the encoder LSTM cell passes its final hidden state to the decoder LSTM cell, which is referred to as the context vector. The context vector is used as the initial hidden state of the decoder LSTM cell.

DECODER: The next module is the decoder, which obtains the context vector generated from the encoder as an initial hidden state, and generates the output sentence sequentially. In Figure 1, the decoder takes the symbol "<sos>" as the initial input, indicating the start of the sentence. When "<sos>" is inputted, the decoder predicts the word with the highest probability that will appear next. At the first time step, the decoder LSTM cell predicts the word "je" as the next word. The predicted word "je" is then fed as input to the LSTM cell of the next time step. The second time step of the decoder LSTM cell predicts the word "suis" based on the input word "je," and this process continues by feeding the predicted word as input to the next LSTM cell. The decoder essentially predicts the next word and repeats this process by inputting the predicted word to the next time step's LSTM cell until the symbol "<eos>", representing the end of the sentence, is predicted as the next word.

Embedding vectors and context vectors: Furthermore, each of the words that are inputted in the Seq2Seq are converted into a vector form, an embedding vector, by embedding layers. Each of these embedding vectors and the hidden and cell states of the previous LSTM cell are inputted to every LSTM cell. Then, the LSTM cell generates the hidden state and cell state of the current time step, where they serve as inputs of the upper layer cell or the cell of the next time step. In this structure, the hidden and cell state at the current time step (t) can be seen as the accumulated value of the hidden and cell states from all previous time steps in the same LSTM cell. Therefore, the mentioned context vector is actually referring to the hidden state of the last LSTM cell in the encoder. This hidden state summarizes the information of all word tokens in the input sentence.

The aim of training the Seq2Seq model in the paper was maximizing the log probability of a translated sentence T for a given source sentence S. This eventually makes the training objective to be:

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

where $\mathcal{S}$ is the training set. Then, the most likely translation result is selected by a greedy method as:

$$\hat{T} = \arg\max_T p(T|S)$$

This is an extended abstract. The full paper is available at the Sequence to Sequence Learning with Neural Networks, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

A simple left-to-right beam search decoder is used to find the most likely translation. It maintains a small number of partial hypotheses and extends each hypothesis with possible words from the vocabulary. The hypotheses are pruned to keep only the most likely ones based on the model's log probability. When the "<EOS>" symbol is added, a hypothesis is considered complete and moved to the set of complete hypotheses. Despite its simplicity, this decoder performs well, even with a beam size of 1. Using an LSTM, the 1000-best lists generated by the baseline system are rescored by computing the log probability of each hypothesis and taking an average with the LSTM's score.

3 Theory and Evaluation of Gated Recurrent Unit

3.1 Theory

The Gated Recurrent Unit (GRU) was first proposed by Cho in [3], for making each of the recurrent unit to adaptively capture the dependencies of different time scales. The GRU unit has gating units that modulates the inflow of information while not having separate memory cells.

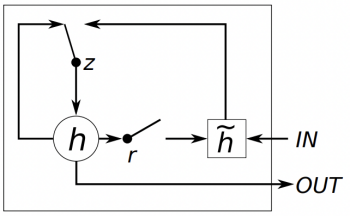

Figure 2: A graphical illustration of an overall structure of a GRU. r and z represent the reset gate and update gate, respectively, while h and $\tilde{h}$ refer to the activation and the candidate activation, respectively.

The activation h_t^j of the GRU at time t is a linear interpolation between the previous activation h_{t-1}^j and the candidate activation $\tilde{h}_t^j$ as follows:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j,$$

The *update gate* z_t^j determines how much the unit updates its activation or content. The update gate is computed by the following equation:

$$z_t^j = \sigma \left(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1}\right)^j .$$

The process of combining the existing state with the newly computed state through a linear sum is similar to the LSTM unit. However, unlike the GRU, the LSTM has a mechanism to control the extent to which its state is exposed, whereas the GRU exposes the entire state at each step without such control.

The candidate activation $\tilde{h}_t^j$ can be computed by a similar method as that computing the traditional recurrent unit [1], which can be represented as:

$$\tilde{h}_t^j = \tanh \left(W \mathbf{x}_t + U \left(\mathbf{r}_t \odot \mathbf{h}_{t-1}\right)\right)^j ,$$

In the above equation, $\mathbf{r}_t$ means the set of reset gates and $\odot$ is an element-wise multiplication.

When the reset gate is off (r_t^j is close to 0), it efficiently makes the unit to act like it is reading the initial character of a given input sequence, and *forget* the previously computed state.

The reset gate r_t^j can be computed similarly as the update gate:

$$r_t^j = \sigma \left(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1}\right)^j .$$

The paper discusses about the similarities and differences of LSTM and GRU, in terms of their configuration and performances. The main shared feature between the LSTM unit and the GRU is the additive component in their update from time step t to $t+1$, which is not present in the traditional recurrent unit. Unlike the traditional recurrent unit, which replaces the unit's activation or content entirely with a new value, both LSTM and GRU units preserve the existing content and add new content on top of it.

This additive nature offers two advantages. Firstly, it allows units to remember specific features in the input stream over a long series of steps without overwriting them, based on the decisions made by the forget gate in the LSTM unit or the update gate in the GRU unit. Secondly, the additive update creates shortcut paths that bypass multiple time steps, enabling error back-propagation without vanishing gradients. This helps address the problem of vanishing gradients that arise when passing through multiple bounded non-linearities.

However, these units also have notable differences. One difference is that the LSTM unit has a controlled exposure of its memory content through the output gate, allowing for controlled access by other units in the network. In contrast, the GRU unit exposes its entire memory content without control. Another difference lies in the location of the input gate (or reset gate). The LSTM unit computes the new memory content independently of the amount of information flowing from the previous time step, while the GRU unit controls the information flow from the previous activation when computing the new candidate activation but lacks independent control over the amount of candidate activation being added (the control is tied via the update gate).

Based on these similarities and differences, it is challenging to determine which type of gating unit would perform better in general. Although preliminary experiments in [1] showed comparable performance of LSTM and GRU units in machine translation, it remains unclear if this generalizes to tasks beyond machine translation. Therefore, further empirical comparisons between LSTM and GRU units are warranted.

3.2 Evaluation

Settings The researchers compared the performance of LSTM, GRU, and tanh model in a sequence modeling task. The sequence modeling set a goal to learn the probability distribution over the sequences, by maximizing the log-likelihood of the model given in a set of training sequences, where θ is the set of model parameters as the following equation.

$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p\left(x_t^n \mid x_1^n, \ldots, x_{t-1}^n; \boldsymbol{\theta}\right),$$

Especially, the researchers evaluated these units in the tasks of polyphonic music modeling and speech signal modeling. The sizes of each unit are shown in detail in Figure 3.

Unit	# of Units	# of Parameters
Polyphonic music modeling		
LSTM	36	$\approx 19.8 \times 10^3$
GRU	46	$\approx 20.2 \times 10^3$
tanh	100	$\approx 20.1 \times 10^3$
Speech signal modeling		
LSTM	195	$\approx 169.1 \times 10^3$
GRU	227	$\approx 168.9 \times 10^3$
tanh	400	$\approx 168.4 \times 10^3$

Figure 3: The sizes of the models tested in the experiments.

Figure 4 presents the results of the experiments. For the polyphonic music datasets, the GRU-RNN outperformed the LSTM-RNN and tanh-RNN on all datasets except Nottingham. However, all three models showed similar performance on these music datasets.

			tanh	GRU	LSTM
Music Datasets	Nottingham	train	3.22	2.79	3.08
		test	**3.13**	3.23	3.20
	JSB Chorales	train	8.82	6.94	8.15
		test	9.10	**8.54**	8.67
	MuseData	train	5.64	5.06	5.18
		test	6.23	**5.99**	6.23
	Piano-midi	train	5.64	4.93	6.49
		test	9.03	**8.82**	9.03
Ubisoft Datasets	Ubisoft dataset A	train	6.29	2.31	1.44
		test	6.44	3.59	**2.70**
	Ubisoft dataset B	train	7.61	0.38	0.80
		test	7.62	**0.88**	1.26

Figure 4: The average negative log-probabilities of the training and test sets.

In contrast, the RNNs with gating units (GRU-RNN and LSTM-RNN) clearly outperformed the traditional tanh-RNN on both Ubisoft datasets. The LSTM-RNN performed best on Ubisoft A, while the GRU-RNN performed best on Ubisoft B.

These results highlight the advantages of gating units over traditional recurrent units. Convergence is often faster, and the final solutions tend to be better. However, the comparison between LSTM and GRU is inconclusive, suggesting that the choice of gated recurrent unit type may heavily depend on the dataset and task at hand.

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[3] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[7] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[8] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.