# DEEP LEARNING LAB(EEE4423-01) Week 12 - seq2seq and GRU

Hyeon Si Eun[1], 2019142214

[1] Department of Electric and Electronic Engineering, Yonsei University

## 1 Sequence to Seqeunce

DNN (Deep Neural Network) has made steady progress in voice recognition and object recognition. DNN is a model that has good performance for difficult problems such as voice recognition and object recognition because it can perform arbitrary parallel calculation at an appropriate level. The number of N-bits can be sorted using only two hidden layers of qudratic size. Furthermore, large DNNs can be trained with supervised backpropagation whenever the labeled training set has enough information to specify the network's parameters. Thus, if there exists a parameter setting of a large DNN that achieves good results (for example, because humans can solve the task very rapidly), supervised backpropagation will find these parameters and solve the problem.
However, there is a limitation that it can be applied only to problems where inputs and targets are encoded as vectors of fixed dimensions. there was a limit that the sequencial problem could not be solved properly because the input size was fixed. Many problems, such as speech recognition and machine translation, are well represented by sequences of unknown length. A question-answering problem requires mapping a sequence of words representing the question to a sequence of words representing the correct answer. As in the above problem, it can be seen that a method for learning to map a sequence to a sequence without being bound by a domain is useful. DNNs have difficulty solving sequences because the dimensions of the inputs and outputs must be known and fixed. In this paper[3], they show that Long Short-Term Memory (LSTM) architecture can solve the general sequence-to-sequence problem. they tried to solve the sequencial problem by using two LSTM (Long Short Term Memory) as Encoder and Decoder. Through this, a lot of performance improvements were achieved, especially in long sentences. In addition, the performance was improved by placing the words in reverse order.

In this paper, They show a forward-propagating LSTM structure that can be applied to tasks with a general sequence-to-sequence structure. The idea is that LSTMs use sequence inputs to obtain large, fixed-dimensional vector representations. And another LSTM extracts the output from the sequence representation vector. Next, LSTM is a recurrent neural network language model except for the condition that the input is a sequence. Therefore, LSTMs can learn time-dependent long data, and this ability of LSTMs helps solve applications with significant temporal differences between inputs and corresponding outputs.

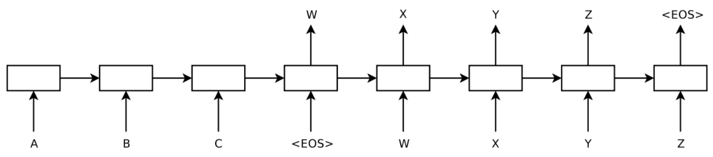


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token.

Given an input sequence, a Recurrent Neural Network (RNN) computes an output sequence using the formula below.

$h_t = sigm(W^{hx}x_t + W^{hh}h_{t-1})$

$y_t = W^{yh}h_t$

RNN can easily map sequences to sequences whenever the alignment between inputs and outputs is known in advance. However, it is not clear how to apply it to problems where the input and output sequences have different lengths. The simplest strategy for general sequence learning is to use one RNN to map the input sequence to a fixed-size vector, and another RNN to map the vector to a target sequence. However, this method can cause a problem called long term dependency. LSTM is known to learn problems with long range temporal dependencies, so an LSTM may succeed in this setting.
The goal of LSTM is to evaluate the conditional probability of an output sequence relative to an input sequence. LSTM first calculates a fixed-dimensional vector $v$ for the input sequence, and calculates the probability of LSTM-LM calculated by the following formula.

$$p(y_1, y_2, ....y_{T'}|x_1, x_2....x_T) = \prod_{t=1}^{T'} p(y_t|v, y_1, y_2....y_{t-1}) \quad (1)$$

Each distribution $p(y_t|v, y_1, y_2....y_{t-1})$ is represented by the softmax value of all words in the word dictionary. Each sentence ends with an end-of-sentence symbol, the <EOS> token. This token can be used to determine the end of a sentence. Both input and output statements end with the corresponding token.
The actual model described in this paper has three important differences from the one shown in Figure 1. First, we use two different LSTMs in this paper: one for the input sequence(Encoder) and one for the output sequence(Decoder). Another model is used because the number of parameters can be increased by increasing the small computational cost. And this makes multiple language pairs natural. Second, They observed that LSTM with deep layers performed better than shallow LSTM, so they used LSTM with 4 layers. Finally, they observed better performance when the word order of input sentence is reversed. Therefore, the performance was improved by putting the order of input in reverse.
In the experiment of this paper, WMT'14 English to French data set was used. Since the general neural network model relies on vector representations for each word, we have pinned all vocabulary in both languages. The most important part of the experiment in this paper is to train large and deep LSTM for many sentence pairs. They conducted training to maximize log probability using the following formula as an objective function.

$$1/|S| \sum_{(T,S)\in S} \log p(T|S) \quad (2)$$

In this case, $S$ is the training set and $T$ is the translation result of the model. After training, the most probable translation was found through the following formula.

$$\hat{T} = \underset{T}{argmax}\,\mathrm{p(T|S)} \quad (3)$$

In the experiment of this paper, the translation with the highest probability was found using the left to right beam search decoder.
If you want to see detailed experimental results, I recommend you to read this paper.
Overall, seq2seq models have been highly effective in many areas of NLP and continue to be a fundamental architecture in the field. The introduction of attention and transformer-based architectures has further improved their capabilities, and they continue to evolve.

## 2 GRU

Gated Recurrent Units (GRUs) are a type of recurrent neural network (RNN) introduced by Kyunghyun Cho and others in 2014. GRUs, like Long Short-Term Memory (LSTM) units, aim to solve the vanishing gradient problem which can be encountered when training traditional RNNs. This problem makes it hard for the RNN to learn and maintain long-term dependencies in sequence data. GRUs control the flow of information like the LSTM unit but without having to use a memory unit. They just expose the full hidden

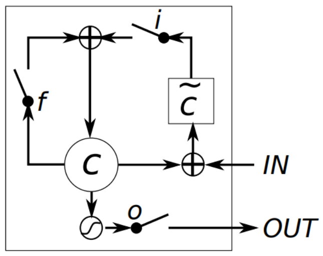
This is an extended abstract. The full paper is available at the this paper and this paper.

content without any control, which simplifies the model and reduces the computational cost.

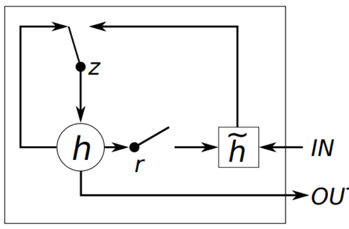
In this paper[1], they compare various types of RNNs. In particular, they focus on RNNs with gating mechanisms such as LSTM and GRU. Polyphonic music modeling and speech signal modeling were evaluated using the corresponding RNN. As a result of the experiment, advanced RNN showed better performance than traditional RNN, and GRU also showed comparable performance to LSTM.

RNNs have shown good performance on machine learning tasks with varying input and output lengths. In particular, they have recently shown good performance in the field of machine translation. One interesting point is that the models that have shown great performance recently are not vanilla RNNs, but those with hidden recurrent units such as LSTMs. In this paper, three models: LSTM, GRU, and the traditional tanh unit were evaluated. A polyphonic music dataset was used to assess these models.
If you want to get background knowledge about RNN and LSTM, read this paper.This paper tells more detailed mathematical theory of RNN and LSTM.

In this paper[1], They compare LSTM and GRU in sequence modeling. The following figure shows the structure of each recurrent unit.



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

Figure 2: Illustration of (a) LSTM and (b) Gated Recurrent Unit.

There are many LSTMs made by slightly modifying the original LSTM. In this paper, Graves' LSTM[2] is introduced. A weighted sum of inputs and nonlinear speech signal modeling were simply evaluated using the corresponding RNN. As a result of the experiment, advanced RNN showed better performance than traditional RNN, and GRU also showed comparable performance to LSTM. Unlike the recurrent unit, which is simply calculated as a weighted sum of inputs, each $j^{th}$ LSTM unit maintains the value of memory $C_t^j$ for each time $t$. The output of $h_t^j$ is calculated by the following formula.

$$h_t^j = \sigma_t^j tanh(c_t^j) \tag{4}$$

In this case, $o_t^j$ is the total sum of memory content exposures, which are output gates. This output gate is calculated by the following formula.

$$\sigma_t^j = \sigma(W_o\mathbf{x_t} + U_o\mathbf{h_{t-1}} + V_o\mathbf{c_t})^j \tag{5}$$

In the above formula, $\sigma$ is a logistic sigmoid function and $V_o$ is a diagonal martix. Memory cell $c_t^j$ is updated by forgetting some existing memory and adding a new memory content $\tilde{c}_t^j$.

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j \tag{6}$$

where the new memory content is $\tilde{c}_t^j = tanh(W_c\mathbf{x_t} + U_c\mathbf{h_{t-1}})^j$. The extent to which the existing memory is forgotten is modulated by a forget gate $f_t^j$, and the degree to which the new memory content is added to the memory cell is modulated by an input gate $i_t^j$. Gates are computed by $f_t^j = \sigma(W_f\mathbf{x_t} + U_f\mathbf{h_{t-1}} + V_f\mathbf{c_{t-1}})^j$, $i_t^j = \sigma(W_i\mathbf{x_t} + U_i\mathbf{h_{t-1}} + V_i\mathbf{c_{t-1}})^j$. Note that $V_f$ and $V_i$ are diagonal matrices.
Unlike to the traditional recurrent unit which overwrites its content at each time-step, an LSTM unit is able to decide whether to keep the existing memory via the introduced gates. Intuitively, if the LSTM unit detects an important feature from an input sequence at early stage, it easily carries this information (the existence of the feature) over a long distance, hence, capturing potential long-distance dependencies.

Gated Recurrent Unit(GRU) was proposed to make each recurrent unit to adaptively capture dependencies of different time scales. Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cells. 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 $\tilde{h}_t^j$:

$$\tilde{h}_t^j = (1 - \tilde{z}_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{7}$$

where an update gate $z_t^j$ decides how much the unit updates its activation, or content. The update gate is computed by

$$z_t^j = \sigma(W_z\mathbf{x_t} + U_z\mathbf{h_{t-1}})^j \tag{8}$$

This process for obtaining a linear sum between the existing state and the new state is similar to the LSTM unit. However, the GRU does not have any mechanism to control the degree to which its state is exposed, but exposes all of the state each time.

The candidate activation $\tilde{h}_t^j$ is computed similarly to that of the traditional recurrent unit.

$$\tilde{h}_t^j = tanh(W\mathbf{x_t} + U(\mathbf{r_t} \odot \mathbf{h_{t-1}}))^j \tag{9}$$

where $\mathbf{r_t}$ is a set of reset gates and $\odot$ is an element-wise multiplication. When $r_t^j$ close to 0,the reset gate effectively makes the unit act as if it is reading the first symbol of an input sequence, allowing it to forget the previously computed state. The reset gate $r_t^j$ is computed similarly to the update gate:

$$r_t^j = \sigma(W_r\mathbf{x_t} + U_r\mathbf{h_{t-1}})^j \tag{10}$$

The most prominent factor that LSTMs and GRUs share is that components are added during the process of updating from $t$ to $t+1$. In traditional RNN models, these are replaced with new values rather than added. Another commonality between LSTM and GRU is that they keep the existing content and add the new content on top of it
This additional element offers two advantages. First, for long series steps, each unit remembers the existence of a specific feature. Another significant point is that LSTM's forget gate and GRU's update gate decide how much previous information to retain, unlike traditional RNN's overwrite method.
Second, the addition creates a shortcut path that can be connected to multiple steps efficiently. This shortcut easily solves the vanishing gradient problem in back-propagation.
These two units however have a number of differences as well. One feature of the LSTM unit that is missing from the GRU is the ability to control the exposure of memory contents. While LSTMs manage the degree to which memory content is exposed via output gates, GRUs do not possess this mechanism.
Another difference lies in the location of the input gate, which corresponds to the reset gate. LSTM computes new memory content without separating it from adjusting the total amount of the old memory. Additionally, the LSTM unit manages the total amount of new memory content by adding an independent cell to the forget gate. On the other hand, the GRU controls the flow of information through previous operations but does not independently control the total amount of information.
GRUs have fewer parameters than LSTMs (as they lack an output gate), so they may train a bit faster or need less data to generalize. However, they might not perform as well on tasks requiring modeling longer sequences. The choice between GRUs and LSTMs often depends on the specific requirements of a task and the amount of data available.
The authors compare the LSTM unit, GRU and tanh unit in the task of sequence modeling. Sequence modeling aims to maximize log-likelibood to learn a probability distribution over sequence.

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} logp(x_t^n | x_1^n, \cdots x_{t-1}^n; \theta) \tag{11}$$

If you want to see detailed experimental results, I recommend you to read this paper.

[1] 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.

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

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