Communications for Statistical Applications and Methods
2020, Vol. 27, No. 3, 313–326

https://doi.org/10.29220/CSAM.2020.27.3.313
Print ISSN 2287-7843 / Online ISSN 2383-4757

Understanding recurrent neural network for texts using English-Korean corpora

Hagyeong Lee[a], Jongwoo Song[1,a]

[a]Department of Statistics, Ewha Womans University, Korea

Abstract

Deep Learning is the most important key to the development of Artificial Intelligence (AI). There are several distinguishable architectures of neural networks such as MLP, CNN, and RNN. Among them, we try to understand one of the main architectures called Recurrent Neural Network (RNN) that differs from other networks in handling sequential data, including time series and texts. As one of the main tasks recently in Natural Language Processing (NLP), we consider Neural Machine Translation (NMT) using RNNs. We also summarize fundamental structures of the recurrent networks, and some topics of representing natural words to reasonable numeric vectors. We organize topics to understand estimation procedures from representing input source sequences to predict target translated sequences. In addition, we apply multiple translation models with Gated Recurrent Unites (GRUs) in Keras on English-Korean sentences that contain about 26,000 pairwise sequences in total from two different corpora, colloquialism and news. We verified some crucial factors that influence the quality of training. We found that loss decreases with more recurrent dimensions and using bidirectional RNN in the encoder when dealing with short sequences. We also computed BLEU scores which are the main measures of the translation performance, and compared them with the score from Google Translate using the same test sentences. We sum up some difficulties when training a proper translation model as well as dealing with Korean language. The use of Keras in Python for overall tasks from processing raw texts to evaluating the translation model also allows us to include some useful functions and vocabulary libraries as well.

Keywords: RNN, NLP, Seq2Seq, Neural Machine Translation, Keras

1. Introduction

As a main functional part of artificial intelligence (AI), deep learning enables the computer to capture significant information from numerous data. Research in both supervised and unsupervised learnings are currently underway to obtain a high-level of computer understanding that is equal to human capabilities in fields such as computer vision, natural language processing, and speech recognition. The main objectives of the studies are to represent informal data as reasonable vectors that computers can process and to build a network with efficient architecture. There are several main structures leading the current technology. For example, CNN is the leading architecture of most computer vision tasks including image recognition and feature extraction.

Typical networks such as MLP and CNN take input features all at once, independently. Suppose there is a sequential data with observations made with a series of points or subsequences. We may input all timesteps at once to those networks; therefore, there is no shared information of the order between input points even though the earlier one might have an effect on the later one. Once the

[1] Corresponding author: Department of Statistics, Ewha Womans University, 52, Ewhayeodae-gil, Seodaemun-gu, Seoul 03760, Korea. E-mail: josong@ewha.ac.kr

Published 31 May 2020 / journal homepage: http://csam.or.kr

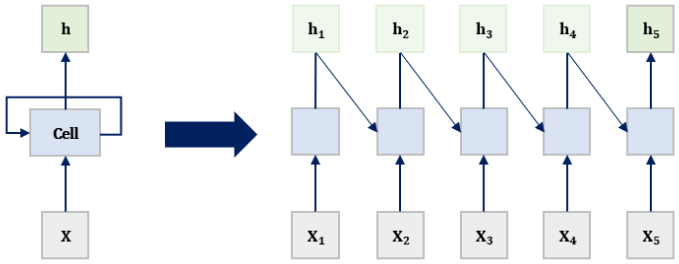

Figure 1: *A recurrent structure.*

layer accepts the inputs, their activation values are passed to the next layer and do not go back called feed-forward networks.

However, recurrent neural network (RNN) is designed for processing sequential values effectively. Jordan (1986) first introduced the concept of recurrence. The network accepts one element (single vector) of a sequence at a time, and iterate the same computation along the elements of the entire sequence. This repetitive state is called a Cell in a recurrent layer between input and output nodes because it acts like a memory cell. Classification or sequential prediction using time-series data, texts, and speech are main tasks where RNN can show an outstanding performance.

In this paper, we summarize overall mechanisms of recurrent networks and their main usage in Sequence-to-Sequence networks. There exist a lot of applications in sequence prediction that RNN can be applied to such as time-series prediction and speech recognition. Among them, we especially pay attention to machine translation, which is one of the main text prediction problem that has successfully developed with RNN. Neural networks based on machine translation have recently been developed. For example, Wu *et al.* (2016) introduced a neural machine translation (NMT) translation system for Google that represented an update over the previous statistical methods (SMT).

The remaining of the paper is organized as follows. In Section 2, we will explore the main concepts that can build a NMT. We first explain three main recurrent architectures from the simplest one to advanced ones, and then summarize some techniques to represent texts as vectors that the network can accept. We describe a Sequence-to-Sequence network with source texts as inputs and target texts as outputs at the end of the section. In Section 3, we discuss the results of applying several translation models to two real English-Korean corpora, provided by AI hub of NIA (http://www.aihub.or.kr/). Section 4 provides concluding remarks.

2. Concepts in neural machine translation with recurrent neural network

2.1. Recurrent neural network

At each time t in a recurrent layer, the cell computes an output based on both the input and the previous 'hidden state' values through several transformation using weights and activation. It passes the current output as an updated hidden state to next time $t + 1$, and reuse it as an optional input. For example, Figure 1 shows the structure of the simplest RNN in five different time points. The left part of the figure represents the recurrent property, and the right part is a sequential description of the recurrent procedure for each time point $t = 1, \ldots, 5$ in a cell. Each input data point at time t is denoted as x_t, and each newly computed state is h_t. The cell produces outputs h_t out of the layer as well as passes them to the next time point of the cell. It uses shared set of parameters to compute each output at time t. If we want to maintain the dimension of time, we may pass all states $h_1, \ldots, h_5$ to the next layer. Otherwise,

Table 1: Vectors at each time t in a recurrent cell

Architecture	Variable	Meaning	Shape
	x_t	An input feature vector at time t	(batch_size, p)
	h_t	A (hidden) state vector at time t	(batch_size, q)
LSTM (optional)	i_t	An input gate vector at time t	(batch_size, q)
	f_t	A forget gate vector at time t	
	o_t	An output gate vector at time t	
	C_t	A Cell state vector at time t	
	$\tilde{C}_t$	A candidate vector for Cell state at time t	
GRU (optional)	z_t	An update gate vector at time t	(batch_size, q)
	r_t	A reset gate vector at time t	
	$\tilde{h}_t$	A candidate vector for hidden state at time t	

LSTM = long short-term memory; GRU = gated recurrent units.

we may pass only the last state h_5 because it is expected to involve the accumulated information from the beginning.

We can generally express the recurrent structure in a numerical form as follows where h_t is a function of x_t and h_{t-1}.

$$h_t = f(x_t, h_{t-1}).$$

RNN can vary according to the internal function f. From the most basic form, 'SimpleRNN', many variants have been introduced to redeem its accuracy and training efficiency. Here we list some most famous developments.

Long short-term memory (LSTM) was designed to prevent the vanishing gradient problem of simple RNN by Hochreiter and Schmidhuber (1997). It has an additional memorable cell state and adjusting inputs and outputs by three gates. Because the architecture turned out to learn high-level sensitive contexts of language, it has been the best recurrent architecture widely used in processing text and speech such as machine translation, speech recognition, and text generation. Even if the training would be slow due to complexity, there are few remarkable variants that significantly outperform standard LSTM's accuracy (Greff *et al.*, 2015).

Gated recurrent units (GRU) was introduced in 2014 as another gated algorithm by Cho *et al.* (2014) It is the most popular one in many applications as a substitute of LSTM, because it reduces the complexity of LSTM and maintains a similar performance.

In this section, we compare the internal procedures of the three architectures using mathematical formulas. Table 1 describes all the notations of the inputs and internal outputs. Note that p is the number of input features and q is the number of latent units in a recurrent layer.

2.1.1. SimpleRNN (Vanilla RNN)

Suppose the number of timesteps in a sequence is T and the batch size is 1. For $t = 1, \ldots, T$, the simplest (vanilla) RNN inputs x_t and h_{t-1} and compute h_t with activation. In the following descriptions, W_{xh} denotes a weight matrix which is multiplied with x to generate h and W_{hh} is the one from previous h to current h. Please note that the dimension of state h_t is $(1, q)$ ignores the time dimension here. Therefore, the parameter set in a vanilla RNN cell contains $p \times q + q \times q$ weights and q biases.

$$\underset{(1,q)}{h_t} = \tanh\Big(\underset{(1,p)}{x_t} \cdot \underset{(p,q)}{W_{xh}} + \underset{(1,q)}{h_{t-1}} \cdot \underset{(q,q)}{W_{hh}} + \underset{(1,q)}{b_h} \Big).$$

As the length of the sequence gets longer, training vanilla RNN becomes harder because of the vanishing gradients.

2.1.2. Long short-term memory

The most remarkable development of LSTM from vanilla RNN is that it divides the state into two states and differentiates 'cell state' C from the original hidden state h. C is expected to store long-term information, while h is expected to have short-term information. Besides, it contains three 'gates' which have values between 0 and 1 by *sigmoid* activation and controls the memorable amount of both states' information. Following descriptions of f (forget gate), i (input gate) and o (output gate) imply how three gates produced in a LSTM cell at time t.

$$\begin{aligned}
f_t &= \text{sigmoid}(x_t \cdot W_{xf} + h_{t-1} \cdot W_{hf} + b_f), \\
i_t &= \text{sigmoid}(x_t \cdot W_{xi} + h_{t-1} \cdot W_{hi} + b_i), \\
\underset{(1,q)}{o_t} &= \text{sigmoid}(x_t \cdot \underset{(p,q)}{W_{xo}} + h_{t-1} \cdot \underset{(q,q)}{W_{ho}} + \underset{(1,q)}{b_o}), \\
\tilde{C}_t &= \tanh(x_t \cdot W_{xc} + h_{t-1} \cdot W_{hc} + b_c).
\end{aligned}$$

At the same time, the cell also compute a candidate of the current cell state, $\tilde{C}_t$, in a similar way as the vanilla RNN computes h_t. Then as the name suggests, the forget gate is multiplied with the existing cells state and adjusts the amount. The input gate is in charge of adjusting the new candidate. The true cell state at time t is a linear combination of both. Finally, the cell produces another state h_t from C_t and adjusts values with the output gate. Since all gates and states are vectors of length q, $*$ denotes element-wise multiplication.

$$\begin{aligned}
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t, \\
h_t &= o_t * \tanh(C_t).
\end{aligned}$$

Because of the parameters in three gates, the LSTM cell has four times as many parameters as the vanilla RNN and therefore training them takes a longer time. It suffers from vanishing gradients less and successfully handles long sequences. However, we found it difficult to clarify how each part in the LSTM cell plays a role that matches their name.

2.1.3. Gated recurrent units

$$\begin{aligned}
z_t &= \text{sigmoid}\left(x_t \cdot W_{xz} + h_{t-1} \cdot W_{hz} + b_z\right), \\
r_t &= \text{sigmoid}\left(x_t \cdot W_{xr} + h_{t-1} \cdot W_{hr} + b_r\right), \\
\tilde{h}_t &= \tanh\left(x_t \cdot W_{xh} + (r_t * h_{t-1}) \cdot W_{hh} + b_h\right), \\
h_t &= z_t * h_{t-1} + (1 - z_t) * \tilde{h}_t.
\end{aligned}$$

GRU has one update gate z instead of the forget and input gate in LSTM. Moreover, it has a reset gate r to control the existing memory. It also generates one hidden state from its candidate and the previous one. While architecture looks simpler than LSTM, several empirical results such as Chung *et al.* (2014) and Jozefowicz *et al.* (2015) have shown that it could have great performances to compete with LSTM. In Figure 2, we described both architectures inside a single cell at time t from inputs to outputs. You can expect that GRU cell makes fewer outputs with simpler operations than LSTM.

Sometimes a sequential data does not have an order in one direction. For example, the appearance of a sequence of words tends to be more crucial than their order in a sentence. A bidirectional RNN is expected to learn more complex contexts using an input sequences in both directions - from the beginning to the end (vise versa).

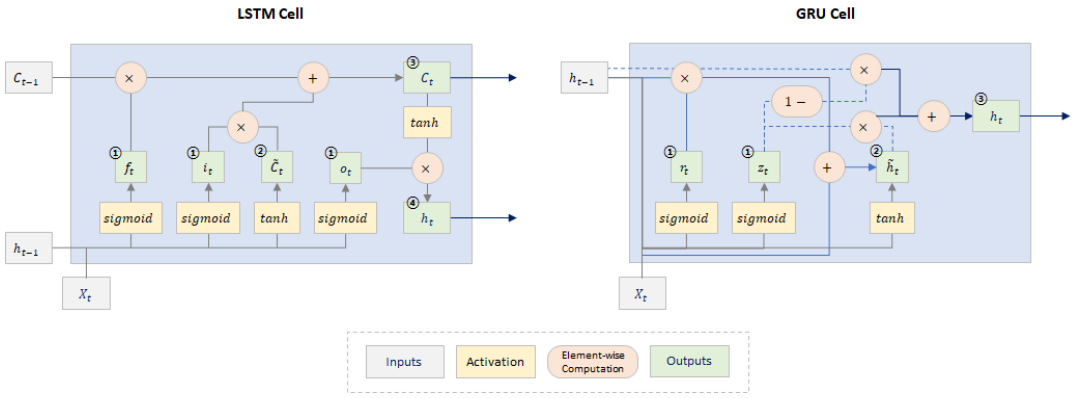

Figure 2: *Descriptions of the architectures: LSTM and GRU. LSTM = long short-term memory; GRU = gated recurrent units.*

2.2. Text representation

Texts is a series of sentences or a single sentence consisting of several words. Suppose that we have a large collection of texts for the purpose of studying NLP tasks, we call it as a Corpus. Using fixed units of words or sentences in our corpus as an input sequence, we could expect a network to learn contexts from an observed text as CNN learns features from an input image. Then, we could use the learned contexts for our objectives about texts such as classification or language modeling. We need to convert raw words to numeric vectors as appropriate as possible since neural networks only accept numerical input values. Here the 'appropriate' means that a word vector implies reasonable values in relation to other words. We summarize two core techniques in terms of representing texts for the most fundamental part of NLP.

2.2.1. Tokenization

A word is mostly considered as the most basic unit of meaning. In general, however, we need to split a sentence with several pieces so that each piece (called a token) becomes a meaningful unit.

We should first determine the unit of token in order to tokenize texts. It can be a character, word, or sentence. Keras provides 'tokenizer' as one of functions for preprocessing texts, and quickly tokenizes a sentence based on spaces after removing punctuation. We can also use word-level tokenizing functions for English texts from natural language toolkit (NLTK) libraries. It provides a suite of text processing libraries written in Python. However, it might not be a suitable method if we tokenize Korean texts because Korean words consists of various postpositions. So we need to split into smaller units. We use Morpheme as token in Korean because it is the smallest unit with meaning. Users can use several morpheme-level tokenizers in the Konlpy package in Python.

2.2.2. Token Embedding

After tokenizing sentences, we temporarily give each token a different number as an index. Each integer value corresponds with a unique token but the size of the number is not meaningful. We consider one-hot encoding for mapping categorical integers with vectors since the tokens are non-continuous. If there exists N unique tokens in the entire dataset, we allocate each token a vector of length N which contains $N - 1$ zeros. It becomes sparse and high-dimensional more and more as the

dataset are larger, resulting in exhausting spatial resources. Also, it cannot represent any relationship or distance between two tokens.

Embedding allocates a token to a dense vector with floating-point values. In contrast to one-hot encoding, we set a desired length p of an embedding vector and its p elements are trained by a neural network using several input tokens for each training step. It is a method that representing tokens in a p-dimensional space as similar words locate as close as possible. Many researchers have focused on embedding methods. For example, Mikolov *et al.* (2013) from Google proposed Word2Vec, an efficient architecture to learn high-quality word vectors regarding that each token as a word. It is based on a simple network with just one hidden projection layer between input and output layer. They trained their networks using the most frequent 1 million words in Google News corpus and verified the qualities of trained vectors by representing word similarities. Other architectures such as GloVe from Stanford (Pennington *et al.* 2014), and fastText from Facebook (Bojanowski *et al.* 2016) have followed. Some of them provide pre-trained embedding vectors trained with large corpus on certain languages. These vectors were trained to maximize the probability for the adjacent appearance of words. Therefore, trained vectors can have completely different values depending on the sentences in the training set (corpus-specific).

We can use an 'Embedding' layer in Keras, as the first layer in our model in order to switch each integer-valued input to a fixed length vector. Similar to other kinds of layers, the network sets random initial values to all vectors in the beginning of the training. Those values are then optimized in the direction of decreasing the loss, with all other parameters in our model. We can initialize the vectors with well-trained embeddings instead of random values if our dataset is not large, as if we use pre-trained filters from a large convolutional network for our own image classification task. It is because we expect the embedding vectors to have reasonable values, even though we do not have enough data to learn the values for ourselves (Chollet, 2017).

For example, we consider an English sentence in our corpus and tokenized it to T_1 tokens, $[X_1, X_2, \ldots, X_{T_1}]$. Now the dimension of the sequence is $(1, T_1)$. Then after it passes through the embedding layer, each token is converted to an embedding vector of length p_1 and the dimension of output becomes $(1, T_1, p_1)$. If we consider a few sentences for a mini-batch, the dimension of the array implies (`batch_size`, `num_timesteps`, `embedding_dim`). The detailed expression follows in Section 2.3.

2.3. Sequence-to-Sequence Network

Applying all the concepts mentioned above, we finally design a model that predicts a target sentence given a source sentence. It is actively used nowadays in areas such as translation and chatbot development. We can build a sequence-to-sequence network with two RNNs as an encoder and a decoder. The architecture was first introduced by Cho *et al.* (2014) with the development of GRU.

Suppose we just finished tokenizing each set of sentences before building an RNN encoder and decoder cells. We enclose the beginning and end of each sequence with $<$ sos $>$ and $<$ eos $>$, which means 'start' and 'end of sentence'. After allocating integers to those tokens, there exists N_1 unique tokens in the source vocabulary and N_2 in the target's. Each integer sequence has its own length (the lengths of sentences are different); therefore, we need to make the length of all sequences equal same to the maximum length in each set by padding shorter ones' back with zeros. We now have source sequences $[X_1, X_2, \ldots, X_{T_1}]$ and target sequences $[<\text{ sos }>, Y_1, Y_2, \ldots, Y_{T_2}, <\text{ eos }>]$. We define both embedding layers mapping N_1 source integers with length p_1 vectors, and N_2 target integers with length p_2 vectors.

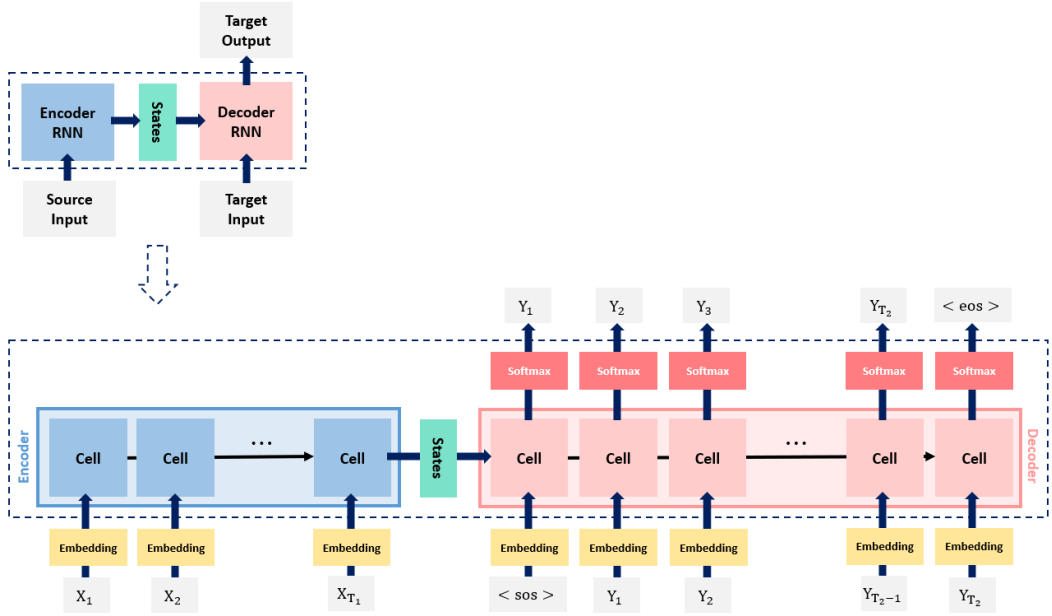

Figure 3: *Training sequence-to-sequence network with two RNNs.*

2.3.1. Training

Figure 3 shows both overall and detailed description of training a seq2seq network. We can simplify the entire procedure using three steps as follows.

(1) The encoder accepts an input source sequence and computes state values.

(2) The encoder passes final state vector as the initial state of the decoder.

(3) From the starting symbol, the decoder sequentially predict the most probable target token that comes to the next.

First, the encoder accepts i^{th} token X_i's embedding vector and computes state values h_i until the end point of the source sequence. If we define the number of units in the recurrent cell as q, the length of states are q. Then, it passes the last hidden state h_{T_1} as the initial state of the decoder RNN. h_{T_1} is called 'contexts' learned from source sequences.

The decoder accepts an embedding vector of the first input token < sos > from the target sequence. Given the initial state and the input, the decoder computes a state vector s_1 with length q. It enters a fully-connected layer with N_2 units. Finally, we can compute the probabilities with softmax activation. It means which token is most likely to appear next among all possible target tokens. Each time while training, the decoder accepts each j^{th} true token Y_j's embedding vector in a target sequence and maximizes the probability of the next true target token Y_{j+1} for the following timesteps $j = 1, \ldots, T_2$. Note that the final output token is < eos >. In summary, the model is trained to maximize each target sequence $[Y_1, Y_2, \ldots, Y_{T_2}, <$ eos $>]$'s probability using both $[X_1, X_2, \ldots, X_{T_1}]$ and $[<$ sos $>$, $Y_1, Y_2, \ldots, Y_{T_2}]$ for the encoder and decoder input. We can summarize the architecture of the model in Table 2 with all predetermined dimensions. All model parameters are optimized to maximize a

Table 2: Summary of a sequence-to-sequence network

		Source	Target
	Vocabulary size	N_1	N_2
	Length of a sequence (number of timesteps)	T_1	T_2
	Embedding dimension of each token	p_1	p_2

	Layer(options)	Output shape	Number of parameters
Encoder	**Input**	(None, None)	-
	Embedding(input_dim = N_1, output_dim = p_1)	(None, None, p_1)	$N_1 \times p_1$
	GRU(units = q, return_state = True)	[(None, q), (None, q)]	$p_1 \times q + q \times q + q$
Decoder	**Input**	(None, None)	-
	Embedding(input_dim = N_2, output_dim = p_2)	(None, None, p_2)	$N_2 \times p_2$
	GRU(units = q, return_sequences = True)	(None, None, q)	$p_2 \times q + q \times q + q$
	Dense(units = N_2, activation = 'softmax')	(None, None, N_2)	$q \times N_2 + N_2$

conditional probability of the target sequence given the source sequence. The loss function of the model is defined with *categorical-crossentropy*, using true token and predicted probability at each timestep.

2.3.2. Prediction

When we predict an unobserved target sequence, the decoder works in a different way from training. From the starting symbol < sos > with the initial state from the encoder, it predicts the probabilities for all tokens and finds the token that matches the highest probability. Then the decoder uses it as the input of the next timestep. It stops the prediction as the predicted token is < eos >.

2.3.3. Evaluation

In addition to the observed one which has been used for learning, there can be more than one possible target sentence for a source sentence. Therefore, it is not easy to evaluate the quality of a translated sentence. We can evaluate the translation performance of the model by intrinsic or extrinsic methods. For the intrinsic evaluation, graders directly evaluate predicted sentences and score the quality of them. Therefore, subjective judgements can be involved.

We use some extrinsic evaluation methods in general. For evaluating the prediction performance of a language model that computes the joint probability for a sequence of n words $w_1, \ldots, w_n$, we can use Perplexity (PPL). It is automatically computed by the model itself, which is equivalent to exponential loss.

$$\begin{aligned}
\mathrm{PPL}(w_1, \ldots, w_n) = P(w_1, \ldots, w_n)^{-\frac{1}{n}} &= \left(\prod_{i=1}^{n} \frac{1}{P(w_i | w_1, \ldots, w_{i-1})} \right)^{\frac{1}{n}} \\
&= \exp\left(\frac{1}{n} \log\left(\prod_{i=1}^{n} \frac{1}{P(w_i | w_1, \ldots, w_{i-1})} \right) \right) \\
&= \exp\underbrace{\left(-\frac{1}{n} \log\left(\sum_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1}) \right) \right)}_{\text{Loss}}.
\end{aligned}$$

If the model is trained well with smaller loss, the perplexity tends to be lower. However, it is the measure about the model rather than about translated sentences. Thus, it does not always imply the

Table 3: Pre-determined dimensions of each corpus

	Colloquialism		News	
	Source	Target	Source	Target
Vocabulary size	10,513	10,967	19,312	17,383
(Word2Vec)	(9,456)	(8,902)	(12,308)	(12,425)
(fastText)	(9,833)	(10,271)	(15,207)	(16,296)
Max length of a sequence	19	32	104	95

Table 4: An example input and target data format from a parallel colloquial sentence

English	Raw	My dream is being the president of a bakery.
	Tokenized	[my, dream, is, being, the, president, of, a, restaurant, .]
	Converted to integers	[14, 261, 7, 226, 3, 939, 12, 6, 3051, 1]
	Padded	[14, 261, 7, 226, 3, 939, 12, 6, 3051, 1, 0, 0, . . . , 0] · · · (A)
Korean	Raw	내 꿈은 레스토랑 사장이 되는 것이에요.
	Tokenized	[< sos >, 내, 꿈, 은, 레스토랑, 사장, 이, 되, 는, 것, 이, 에요, ., < eos >]
	Converted to integers	[1, 61, 262, 12, 4101, 2716, 5, 24, 6, 21, 5, 22, 3, 2]
	Padded	[1, 61, 262, 12, 4101, 2716, 5, 24, 6, 21, 5, 22, 3, 2, 0, 0, . . . , 0] · · · (B)

sentence is translated well in human's viewpoint. The perplexity only considers the probability of a sentence.

For the evaluation of output sentence itself, bilingual evaluation understudy score (BLEU) (Papineni *et al.*, 2002) has been commonly used for a translation quality. Given the source sentence, we compare the model's predicted sentence (candidate) with one or more target sentences (references) from humans from 1-gram to n-gram (usually $n = 4$). By counting the number of concurrence among total n-grams, BLEU numerically suggests the quality of the model's output. The range of BLEU score is between 0 and 1 and the higher value means better translation.

3. Case study: English-Korean corpora

In this section, we build several seq2seq translation models with two GRUs (encoder and decoder) using English-Korean sentences. We obtained those sentences from the text corpora of AI Hub. We focused our experiments on two datasets - (1) Colloquialism: informal style sentences, (2) News: formal style sentences. We sampled 20,000 random sentences in the entire colloquialism corpus. In news corpus, we used 6,165 sentences in total from six selected categories - economy, international, society, sports, and IT science.

We first tokenized all English sentences into source sequences at word-level and Korean sentences into target sequences at morpheme-level. In Table 3, vocabulary size is the number of unique tokens in each corpus. The news corpus contains more unique words (or morphemes) than colloquialism. In addition, the maximum length of the news sequence is longer than the colloquial sequence's. We padded all the sequences of each set to those maximum lengths.

In order to build a translation model, the final form of encoder input data is an integer sequence of length 19 for each observation and each integer represents an English word. Similarly, a decoder input data is an integer sequence of length 32 and each integer represents a Korean morpheme with beginning and ending symbols. Table 4 describes an example of input data in our translation model from an original pair of colloquial English-Korean sentences. The model first accepts (A) and the encoder operates until the end of the sequence. Then the decoder uses each integer of (B) at a time, to predict the next integer. Note that the actual target is the form of one-hot vector of each next integer, which makes the model predict softmax probabilities for all integers in the entire target vocabulary.

Figure 4: *Results of test perplexity and BLEU on the colloquial sentences. BLEU = bilingual evaluation understudy score*

We varied our models in three ways.

A. The number of recurrent `units` in both the encoder and decoder; 256 or 512.

B. `Bidirectional` learning of the encoder GRU or not.

C. Initial values of `Embedding` vectors.

For word vectors, we first randomly initialized all vectors in both embedding layers with random values. Moreover, we expected to get positive impacts by using pre-trained embedding vectors at the initial step available from large networks. For example, we obtained 300-dimensional English word vectors for 3 million words, pre-trained by Word2Vec on Google News dataset. For Korean Word2Vec, we obtained Park's 200-dimensional vectors (2016) pre-trained for 30,185 morphemes on Wikipedia sentences. We also used fastText's 300-dimensional pre-trained vectors (Grave *et al.*, 2018) for both English and Korean. We mapped the pre-trained vectors overlapping with our tokens and trained all models keeping other options the same. We specified the number of tokens that have existing pre-trained vectors by each embedding network in Table 3.

We finally trained $2 \times 3 \times 2 = 12$ different models using the same training (80%), validation (10%) and test (10%) set. We optimized every model to have the minimum validation loss and evaluated on the test set with two extrinsic measures, PPL and BLEU. We computed 2-gram BLEU because our colloquialism sentences were short in general. When training all our models, we used a single GPU

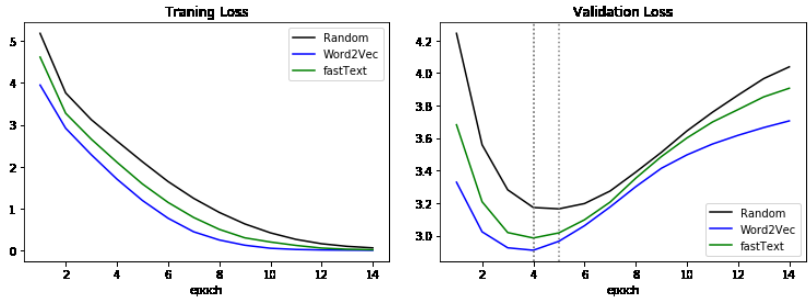

Figure 5: *Comparison of losses by initial embedding values.*

on a system equipped with NVIDIA GeForce GTX 1080 Ti. It took about 52 seconds per epoch on average to train the simplest model with 256 recurrent units. As we increased the model complexity with more recurrent units or bidirectional learning, the maximum computing time for an epoch reached about 2 minutes.

3.1. Colloquialism corpus

Figure 4 summarizes the result of performances on colloquial sentences. The PPL tends to decrease by increasing recurrent units and applying bidirectional learning at the encoder, even if they make a translation model more complex. Initial pre-trained embeddings also helped all models reach a smaller loss and perplexity in the earlier step versus other random values. Model W4 showed the smallest PPL with 512 recurrent units, using bidirectional encoder and initial Word2Vec embeddings. Figure 5 compares three paths of losses by each initial embedding (R4, W4, F4), both on training and validation set. However, the translation performances appeared slightly different. BLEU still tends to improve as the model's complexity increases; however, there is no clear difference between initial embeddings. Considering the translation quality, we should select the fastText-based model F4 that shows the best BLEU.

3.2. News corpus

We also represented the result on the news corpus in Figure 6. As the general length of the sequence becomes longer, both performance measures of all of our models get worse. Word2Vec-based models showed strikingly better PPL on the test set compared to other models. However, PPL tends to increase with more recurrent units and a bidirectional encoder unlike the colloquial sentences. Training recurrent networks would be more difficult as the input sequence length is longer. The best BLEU as the measure of translation sentences appeared in model W2 which does not accord with the perplexity.

4. Conclusion

Deep Learning is gaining more attention in the fields of NLP. We examined how the recurrent network handles sequential data in detail as well as the main concept to build an RNN-based translation model from processing raw input sentences to generating target sentences. We also make use of such sequence-to-sequence structure on any parallel sequential data, such as question-answer or speech-text. However, we need to apply more advanced techniques to improve the performance of the seq2seq network to a fine level. Even though we built various models adjusting some basic structures,

Figure 6: *Results of test perplexity and BLEU on the news sentences. BLEU = bilingual evaluation understudy score.*

Attention (Bahdanau *et al.* 2015) has been widely used in advanced seq2seq networks. For example, Google Neural Machine Translation (GNMT) (Wu *et al.*, 2016) has developed with attention as well as stacking eight LSTMs as the encoder and the decoder respectively and using a bidirectional RNN for the first layer of the encoder. The network with attention connects the decoder with the encoder's state values at all time points, whereas the basic seq2seq network only accepts the last condensed state values from the encoder. It is known as a technique that can handle even long sequences well. In addition to our experiments, we also specified BLEU scores on the same test sentences using APIs of Google Translate and Naver Papago, in Table 5 and Table 6. Even though our basic models show lower performances compared to those state-of-the-art networks, we try to understand some crucial points that could generally improve RNN-based seq2seq network performances based on our experiments.

When training with short sequences, more recurrent units in the network tend to improve the perplexity and BLEU. Bi-directional encoder could learn better contexts by processing input source sequences both from the front and the back and it improved the model. However, the results were different on the longer news sentences. It is because the recurrent network could suffer from a vanishing gradient with long sequences in general and is not easy to be trained. Attention is a technique that can handle long sequences. We also examined the effects of initial embedding values. Using pre-trained embedding vectors from a large network as the initial values of our own model have a good influence on the following learning process. It reduces both the training loss and optimal validation loss in a faster time. Therefore, it is reasonable for users to use pre-trained embedding vectors if their dataset

Table 5: Results of all performance measures on the colloquial sentences

Model	q	Bidirectional	Initial Embeddings	PPL		2-Gram BLEU		4-Gram BLEU	
				Train	Test	Train	Test	Train	Test
R1	256	No	random	2.82	5.15	18.67	17.10	7.46	6.40
R2	512	No	random	2.77	4.93	24.83	20.02	12.40	8.76
R3	256	Yes	random	2.52	4.90	23.53	19.02	11.07	7.74
R4	512	Yes	random	2.52	4.83	29.99	20.79	16.28	8.83
W1	256	No	Word2Vec	2.65	4.66	17.46	16.10	6.38	5.46
W2	512	No	Word2Vec	2.25	4.41	22.70	19.53	10.51	7.90
W3	256	Yes	Word2Vec	2.12	4.29	24.93	20.61	12.21	8.61
W4	512	Yes	Word2Vec	2.22	**4.21**	29.26	20.39	16.93	8.37
F1	256	No	fastText	2.66	4.81	18.80	16.79	7.50	6.27
F2	512	No	fastText	2.26	4.62	22.17	18.38	10.60	7.38
F3	256	Yes	fastText	2.48	4.49	27.81	19.92	14.84	8.38
F4	512	Yes	fastText	2.13	4.37	30.67	**21.92**	16.81	**9.47**
Google							32.13		17.00
Naver							42.71		27.39

PPL = Perplexity; BLEU = bilingual evaluation understudy score.

Table 6: Results of all performance measures on the news sentences

Model	q	Bidirectional	Initial Embeddings	PPL		2-Gram BLEU		4-Gram BLEU	
				Train	Test	Train	Test	Train	Test
R1	256	No	random	4.78	10.90	5.37	5.22	1.59	1.50
R2	512	No	random	5.10	11.32	6.37	5.30	2.15	1.68
R3	256	Yes	random	4.56	11.31	10.08	7.18	3.86	2.46
R4	512	Yes	random	5.91	11.91	9.44	8.81	3.66	3.34
W1	256	No	Word2Vec	3.91	8.83	9.12	8.82	3.44	3.24
W2	512	No	Word2Vec	3.52	8.73	9.40	**9.51**	3.72	**3.67**
W3	256	Yes	Word2Vec	3.42	**8.58**	9.59	9.36	3.77	3.59
W4	512	Yes	Word2Vec	2.79	8.65	7.80	7.53	2.65	2.47
F1	256	No	fastText	4.23	10.03	7.60	6.73	2.44	2.05
F2	512	No	fastText	4.60	10.21	6.93	7.05	2.21	2.24
F3	256	Yes	fastText	3.50	9.95	13.18	9.00	5.79	3.42
F4	512	Yes	fastText	3.39	10.00	8.96	8.91	3.12	3.17
Google							36.84		19.61
Naver							53.05		40.00

PPL = Perplexity; BLEU = bilingual evaluation understudy score.

is small.

We found that BLEU scores do not always correspond to measures used in the translation model. It is because the neural translation model only optimize the loss (objective function) related to the probability of observed sentences that differ from the measure of the translation quality. Moreover, we summarize some difficulties in processing natural Korean languages. Korean has many postpositions and affixes that can easily skipped while maintaining the meaning of the sentence; in addition, unlike English, a sentence has diverse forms depending on the listener. Note that the scores of Google and Naver on colloquial sentences are smaller compared to the scores on the news sentences despite smaller lengths. The reason is that the Korean colloquial sentences in our dataset have different endings including both honorifics and familiar forms, while the news sentences do not. Furthermore, there can exist many possible sentences in Korean that imply the same idea because they are not significantly affected by spaces or the order of words. Therefore, it takes special effort to bring Korean sentences into a refined and consistent state in order to use in a network.

We believe that translation between Korean and English is still in the developing stages of over-

coming difficulties and there must be special efforts to reach a remarkable level of translation.

References

Bahdanau D, Cho K, and Bengio Y (2015). Neural machine translation by jointly learning to align and translate. In *ICLR*.

Bojanowski P, Grave E, Joulin A, and Mikolov T (2016). Enriching Word Vectors with Subword Information.

Cho K, Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, and Bengio Y (2014). Learning Phrase Representations using RNN Encoder Decoder for Statistical Machine Translation, arXiv preprint arXiv:1406.1078.

Chollet F (2017). *Deep Learning with Python*, Manning, New York.

Chung J, Gulcehre C, Cho K, and Bengio Y (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, arXiv preprint arXiv:1412.3555.

Facebook Inc. Word vectors for 157 languages, Available from: https://fasttext.cc/docs/en/crawl-vectors. html

Grave E, Bojanowski P, Gupta P, Joulin A, and Mikolov T (2018). Learning Word Vectors for 157 Languages, arXiv preprint arXiv:1802.06893.

Greff K, Srivastava RK, Koutník J, Steunebrink BR, and Schmidhuber J (2015). LSTM: A Search Space Odyssey, arXiv preprint arXiv:1503.04069.

Hochreiter S and Schmidhuber J (1997). Long short-term memory, *Neural Computation*, **9**, 1735–1780.

Jordan MI (1986). Attractor dynamics and parallelism in a connectionist sequential machine, Cogitive Science Conference, pp 531–546.

Jozefowicz R, Zaremba W, and Sutskever I (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML'15)*, pp 2342–2350.

Kyubyong P (2016). Pre-trained word vectors of 30+ languages. Available from: https://github.com/Kyubyong/wordvectors

Mikolov T, Chen K, Corrado G, and Dean J (2013). Efficient Estimation of Word Representations in Vector Space, arXiv preprint arXiv:1301.3781.

Papineni K, Roukos S, Ward T, and Zhu WJ (2002). BLEU: a Method for Automatic Evaluation of Machine Translation, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 311–318.

Pennington J, Socher R, and Manning CD (2014). GloVe: Global Vectors for Word Representation.

Sutskever I, Vinyals O, and Le QV (2014). Sequence to Sequence Learning with Neural Networks, arXiv preprint arXiv:1409.3215.

Wu Y, Schuster M, Chen Z, Le QV, and Norouzi M (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, arXiv:1609.08144, Retrived 2018.

Received December 16, 2019; Revised February 18, 2020; Accepted February 20, 2020