Abstract

The scholarly article titled "Encoder-Decoder Methods" provides an in-depth examination of Neural Machine Translation (NMT) systems, which are predicated exclusively on neural network architectures. NMT frameworks consist of two components: an encoder that distills a fixed-length vectorial representation from an input sentence of arbitrary length, and a decoder that crafts an accurate translation predicated upon this vectorial representation.

Here provides a comparative analysis of two architectures: the RNN Encoder-Decoder and the Gated Recurrent Convolutional Neural Network (grConv). It found the efficacy of NMT in processing succinct sentences. Although it also notes that with the increase in sentence length and the occurrence of unknown lexemes, the accuracy of translation sharply declines.

Furthermore, the grConv model is even capable of autonomously parsing and integrating the grammatical structures within sentences.

2. Neural Machine Translation system

Neural Machine Translation (NMT) based on pure neural networks is essentially an Encoder-Decoder architecture, typically composed of an encoder and a decoder. The encoder extracts a fixed-length vector representation from a variable-length input sentence, while the decoder generates the correct translation based on this vector. Here, two models are primarily discussed: the RNN and the grConv models.

2.1 Encoder-Decoder Approach

Encoder: This part of the network is responsible for processing the input sequence, compressing the information from the sequence into a fixed-size context vector (usually the hidden state of the last time step). The encoder can be an RNN or its variants (such as LSTM or GRU), which are capable of efficiently handling input sequences and capturing temporal dependencies. The first model discussed in this paper uses an RNN, while the second employs a grConv.

Decoder: The decoder part receives the context vector generated by the encoder and begins to produce the output sequence. In both models discussed in this article, the decoder uses an RNN, which may refer to the context vector at each time step. It generates the output sequence step by step, with each time step's output depending on the output and hidden state of the previous step.

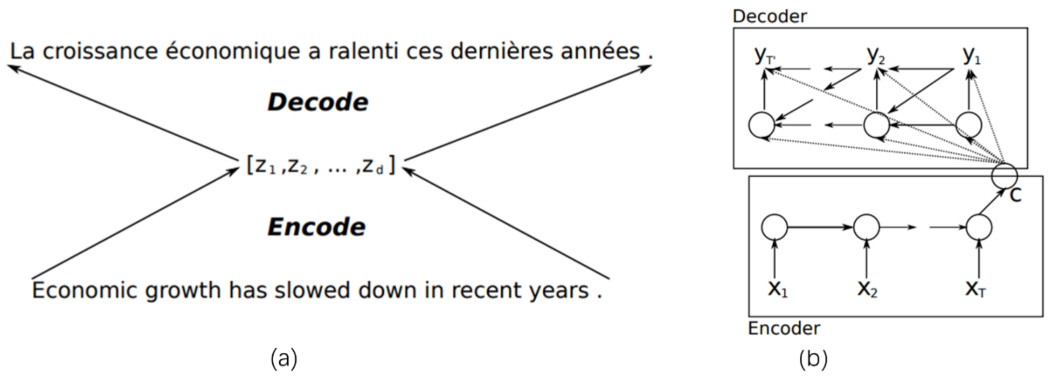
La croissance économique a ralenti ces dernières années .
Decode
[z₁ ,z₂ , … ,z_d]
Encode
Economic growth has slowed down in recent years .
(a)
Decoder
y_T
y₂
y₁
c
x₁
x₂
x_T
Encoder
(b)

Figure 1: (a) The encoder-decoder architecture
(b) An illustration of the proposed RNN Encoder-Decoder

General workflow of the Encoder-Decoder Approach:

1. Input Processing:

The input sequence is fed into the encoder, where each element is processed in turn.

2. Context Vector Generation:

The encoder processes the input sequence through the states of its hidden layers, producing a context vector.

3. Sequence Generation:

The decoder begins to generate the output sequence based on the context vector.

4. Step-by-Step Decoding:

At each time step, the decoder predicts the next output symbol and uses it as the input for the next time step.

5. End of Sequence Generation:

The generation of the sequence is terminated once the decoder produces a special end symbol or reaches the maximum length limit.

2.2 Model 1: Recurrent Neural Network with Gated Hidden Neurons

Preliminary: Recurrent Neural Networks

RNN processes variable-length sequences $x = (x_1, x_2, \ldots, x_T)$ by maintaining a hidden state h over time. The hidden state h is updated at each time step using the following formula.

$$h^{(t)} = f\left(h^{(t-1)}, x_t\right)$$

h^{(t)} = f \left(h^{(t-1)}, x_t \right)

f is an activation function which often involves a linear transformation of the input vectors followed by a sum and then the application of an element-wise logistic sigmoid function.

The RNN is capable of effectively learning distributions over variable-length sequences by learning the distribution of the next input in the sequence $p(x_{t+1}|x_t, \ldots, x_1)$[p(x_{t+1} | x_t, \ldots, x_1)]. a multinomial distribution (1-of-K coding) can be output using asoftmax activation function as:

$$p\left(x_{t,j} = 1 \,\middle|\, x_{t-1}, \ldots, x_1\right) = \frac{exp\left(w_j h_{<t>}\right)}{\sum_{j'=1}^{K} exp\left(w_{j'} h_{<t>}\right)}$$

p(x_{t,j}=1\,|\,x_{t-1},\ldots,x_1)=
\frac{exp(w_jh_{<t>})}{\sum_{j^{'}=1}^{K}exp(w_{j^{'}}h_{<t>})}

For every potential symbol within the range of $j = 1, \ldots. K$, where w_j represents the individual row vectors of the weight matrix W, the following expression gives the joint distribution.

$$p(x) = \prod_{t=1} T\, p(x_t \,|\, x_{t-1}, \ldots, x_1)$$

p(x) = \prod_{t=1}{T} p(x_t\,|\,x_{t-1}, \ldots,x_1)

Hidden Unit that Adaptively Remembers and Forgets

This is a new variant of RNN that employs a gating mechanism, inspired by the LSTM unit, has been simplified so that can be easy to compute and implement, and it is better suit our needs and more effectively handle long-term dependencies.

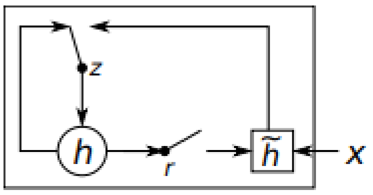

Figure 2: An illustration of the hidden activate function.

In this model, the state update of each hidden unit at every time step depends on two gating mechanisms:

Reset gate r: This determines how much of the previous hidden state will be considered. When r approaches 0, the previous hidden state is almost ignored. The activation of the j-th hidden unit **r**$_j$ is computed by :

$$r_j = \sigma\left([\boldsymbol{W_r x}]_j + [\boldsymbol{U_r h}_{(t-1)}]_j\right)$$

r_j = \sigma \left([W_r x]_j + [U_r h_{(t-1)}]_j \right)

Update gate z: This controls the extent of the update to the hidden state. When z is close to 1, the old state is nearly retained; when z is close to 0, the new candidate state will completely influence the hidden state. The update gate **z** is computed by:

$$z_j = \sigma\left([W_z x]_j + \left[U_z h_{(t-1)}\right]_j\right)$$

z_j = \sigma \left([W_{z} x]_j + [U_{z} h_{(t-1)}]_j \right)

Actual activation $\boldsymbol{\hat{h}}$**:** The new hidden state is calculated through a combination of the update gate z, reset gate r, the current input, and the previous hidden state. The actual activation of the proposed unit $\boldsymbol{h}$ is then computed by:

$$h_j^{(t)} = z_j h_j^{(t-1)} + (1 - z_j)\tilde{h}_j^{(t)}$$

h_j^{(t)} = z_j {h_j}^{(t-1)} + (1 - z_j) {\tilde{h}}_j^{(t)}

where

$$\tilde{h}_j^{(t)} = \phi\left([Wx]_j + \left[U\left(r \odot h_{(t-1)}\right)\right]_j\right)$$

{\tilde{h}}_j^{(t)} = \phi \left([Wx]_j + [U \left(r \odot h_{(t-1)} \right)]_j \right)

This approach essentially allows the hidden state to retain only that information, which is deemed relevant for the future, enabling a more compact representation.
On the other hand, the update gate controls how much information from the previous hidden state

should be carried over to the current hidden state. This helps the RNN to remember long-term information and acts in a role similar to the memory cells in an LSTM network. It could be seen as an adaptive form of a leaky integrator unit.

Since each hidden unit has its own reset and update gates, each unit will learn to capture dependencies on different time scales. Those units with frequently activated reset gates will tend to capture short-term dependencies, while those with active update gates will tend to capture long-term dependencies. In our preliminary experiments, we found that without any gating, we were unable to achieve meaningful results.

2.3 Model 2： Gated Recursive Convolutional Neural Network

Another natural method to handle variable-length sequences is to use a recursive convolutional neural network, in which the parameters of each layer are shared throughout the entire network. The paper introduces a network structure that combines the characteristics of Recursive Neural Networks (RNNs) and Convolutional Neural Networks (CNNs), whose weights are recursively applied through the input sequence until a single fixed-length vector is outputted. In addition to the usual convolutional architecture, the model also introduces a gating mechanism, which allows the network to learn the structure of sentences while processing the input sequence.

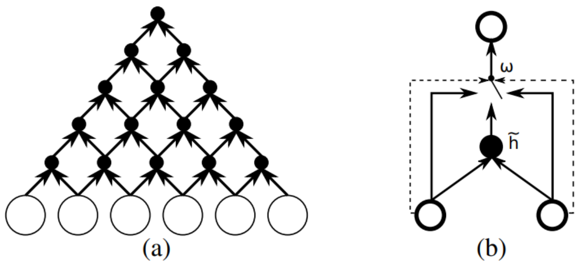

Figure 3: (a) The recursive convolutional neural network
(b) the proposed gated unit for the recursive convolutional neural network

Network Architecture

This network also maintains a hidden state h over time to process variable-length sequence $\boldsymbol{x} = (x_1, x_2, \dots, x_T)$.

The network architecture include four weight matrices $\mathbf{W^l}$, $\mathbf{W^T}$, $\mathbf{G^l}$, $\mathbf{G^T}$, which are shared across different levels of the network. For each recursion level $t \in [1, T-1]$, the activation of j-th hidden unit will be computed by:

$$h_j^{(t)} = w_c \tilde{h}_j^{(t)} + w_l h_{j-1}^{(t-1)} + w_r h_j^{(t-1)}$$

h_j^{(t)} = w_c {\tilde{h}}_j^{(t)} + w_l h_{j-1}^{(t-1)} + w_r h_j^{(t-1)}

Noted that w_c , w_l , w_r sum to 1 which are the values of a gater.

And the hidden unit is initialized by:

$$h_j^{(0)} = Ux_j$$

h_j^{(0)} = U x_j

where U is a matrix to project the input into a hidden space.

Working principle of the gating unit:

The new activation $\widetilde{h_j^{(t)}}$ is computed by:

$$\tilde{h}_j^{(t)} = \phi\left(W^l h_{j-1}^{(t)} + W^r h_j^{(t)}\right)$$

{\tilde{h}}_j^{(t)} =\phi \left(W^l h_{j-1}^{(t)} + W^r h_j^{(t)}\right)

where ϕ is an element-wise nonlinear function

The gating coefficient w are computed by

$$\begin{bmatrix} w_c \\ w_l \\ w_r \end{bmatrix} = \frac{1}{Z}[\exp\left(G^l h_{j-1}^{(t)} + G^r h_j^{(t)}\right)]_k$$

\begin{bmatrix}
W_c \\
W_i \\
W_r \\
\end{bmatrix}

= \frac{1}{Z} exp(G^l h_{j-1}^{(t)} + G^r h^{(t)}_j)

Where $G^l, G^r \in R^{3\times d}$ and Z is the normalization factor, ensuring that the sum of w equals 1, which is computed by

$$Z = \sum_{k=1}^{3}\left[\exp\left(G^l h_{j-1}^{(t)} + G^r h_j^{(t)}\right)\right]_k$$

Z = \sum_{k=1}^{3}[exp(G^l h^{(t)}_{j-1} +G^r h^{(t)}_j]_k

Based on this activation, the activation of a node at the recursive level t can be seen as selecting new components of activation, which can be obtained either from the left child node or the right child node or maintaining the activation from the left child node or obtaining activation from the right child node. This selection allows the overall structure of the recursive convolution to adaptively change with the input.

6. Experimental results

Finally, the performance is tested using the trained network for French-to-English translation. According to the analysis, as the length of the source sentence increases, the performance of the NMT model rapidly declines. Additionally, we find that the size of the vocabulary has a significant impact on translation performance. Nevertheless, in terms of quality, both models are mostly capable of generating correct translations most of the time. Moreover, the newly proposed grConv model can learn some syntactic structure of the source language in an unsupervised manner.