LSTM auto-encoder / Sequence-to-Sequence

Mr.Ilyas Nhasse
Ecole National des Sciences Appliquées
Dep. Electrique et Informatique
Génie Systèmes Embarqués et Informatique Industrielle
Fes, Morocco

Mr. Mohammed Handa
Ecole National des Sciences Appliquées
Dep. Electrique et Informatique
Génie Systèmes Embarqués et Informatique Industrielle
Fes, Morocco

Mr. Oussama Zaim
Ecole National des Sciences Appliquées
Dep. Electrique et Informatique
Génie Systèmes Embarqués et Informatique Industrielle
Fes, Morocco

Mr. Mouad Ghourdou
Ecole National des Sciences Appliquées
Dep. Electrique et Informatique
Génie Systèmes Embarqués et Informatique Industrielle
Fes, Morocco

Mr. Abdelhadi El Ghazi
Ecole National des Sciences Appliquées
Dep. Electrique et Informatique
Génie Systèmes Embarqués et Informatique Industrielle
Fes, Morocco

D. Pr. Hiba Chougrad
Ecole National des Sciences Appliquées
Dep. Electrique et Informatique
Génie Systèmes Embarqués et Informatique Industrielle
Fes, Morocco

***Abstract*— In recent years, the use of deep learning techniques has led to significant advancements in various fields, including natural language processing, computer vision, and speech recognition. Among these techniques, the LSTM auto-encoder and sequence-to-sequence models have gained popularity in sequence modeling tasks. In this article, we provide a comprehensive review of these models, including their architecture, training procedures, and applications. We also discuss the challenges associated with these models and the techniques used to overcome them. Our review highlights the strengths and limitations of these models and provides insights into their potential applications in various fields. We hope that this article will serve as a useful reference for researchers and practitioners interested in the LSTM auto-encoder and sequence-to-sequence models**

I. INTRODUCTION

In recent years, deep learning techniques have revolutionized the field of Natural Language Processing (NLP) and have made significant advancements in machine translation tasks. Machine translation is the process of automatically translating text from one language to another, and it has become an essential tool for global communication. In this context, Long Short-Term Memory (LSTM) auto-encoder and sequence-to-sequence models have emerged as powerful techniques for machine translation.

LSTM auto-encoder / sequence-to-sequence model have been successfully used in various NLP tasks, including machine translation. The LSTM auto-encoder model is an unsupervised learning approach that can learn the underlying representation of the input sequence by compressing and decompressing the sequence.

In this article, we focus on the use of LSTM auto-encoder model for English to French translation. We discuss the architecture of the model, the training and evaluation methods, and compare its performance to other traslators. The aim of this study is to demonstrate the effectiveness of LSTM auto-encoder model in machine translation tasks, and to provide insights into their respective strengths and weaknesses.

The rest of the article is organized as follows. A Problematic that provides a brief overview of related difficulties in machine translation. Architecture of the model is a section where we will present the LSTM cell's. Section 3 describes the medothology of our model and how it works. Section 4takes on the Results we got and the discussion about what we found it reports the experimental results and analysis. Finally, Section 6 concludes the article and outlines future directions of research in machine translation using deep learning techniques.

II. PROBLEMATIC

Long Short-Term Memory Cells (LSTMs) are a type of Recurrent Neural Netwroks (RNNs), an architecture that is based on Sequencing the data outputted from the cell and treating it with the new input data, One of the main challenges that RNNs faced is their inability to capture long-term dependencies of the inputted sequence, a problem that arises from the vanishing gradient phenomenon.

III. LONG SHORT-TERM MEMORY CELL'S ARCHITECTURE

This Section describes how LSTMs work and how they bypass the issue of RNNs and their inability to store information.

LSTMs overcome this issue by introducing a memory cell that can store information over long period of time, through the use of gates, An Input Gate, A Forget Gate, and An Output Gate, of which are responsible for regulating the flow of information stored and coming in and out of the cell.
The memory cell is the key component of LSTMs and consists of a continuous vector state that can be read from and written to by each on of the gates, At each iteration, the input of the LSTM Cell is a combination of the current inputted value and the outputted value from the previous Cell. The Gates are responsible for how much of that outputted data is used or forgotten and how much information is passed to the next iteration. Each Gate has a separate set of learnable parameters, and each controls a different aspect of the cell state.

Now we have a general idea on how an LSTM Cell works, lets see how its looks like:

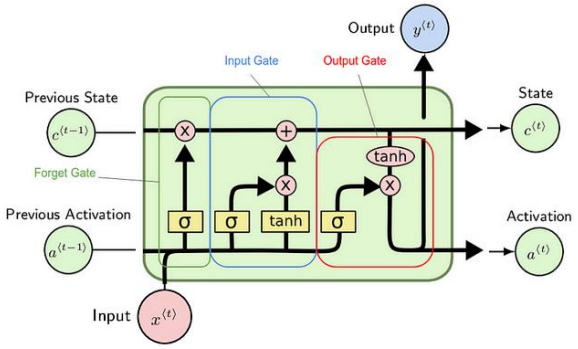

Figure 1: Illustration of an LSTM Cell

This diagram represents an LSTM Cell, these are the parts that make it up:

- The "Cell State"
- The "Hidden State" (noted "Activation" in the diagram)
- The Gates: "Forget "," Input", and "Output" (As we] seen before)

Let's first talk about the new terms, "Cell State and Hidden State" these are two very different in term of their functionality in the LSTM Cell. the cell state is responsible for encoding all of the data from all the previous iteration (time-steps) that have already been processed, while the hidden state is responsible for encoding the characterization of the previous iteration's data as we will see later in dept. At each iteration we will have the following changes applied on each gate, in the form of this equation:

$$f_t = \sigma((W_{hf} \times h_{t-1}) + (W_{xf} \times x_t) + b_f)$$

The Input Gate:

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

The Forget Gate:

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

The Output Gate

$$o_t = \sigma(W^{(c)}x_t + U^{(c)}h_{t-1})$$

Note: for the simplicity of the model, we're not using the bias in this representation.

The Forget Gate:

We use the sigmoid function so their outputs will lie between 0 and 1, which means their output vectors can define how much of another vector can pass through, either amplify(remember) or diminish(forget) the information that's being passed. For example, at the forget gate, if the forget gate outputs a matrix of values that are all very close to 1, it means that the forget gate has concluded that based on the current input, the time-series' history is very important, and therefore, when the cell state from the previous time-step is multiplied by the forget gate's output, the cell state continues to retain most of its original value, or "remember its past". If the forget gate outputs a matrix of values that are close to 0, the cell state's values are scaled down to a set of tiny numbers, meaning that the forget gate has told the network to forget most of its past up until this point.

The Input Gate:

The purpose of the tanh gate:

It is merged together with the sigmoid gate as the "Input Gate", but the tanh gate itself is called the "Candidate Gate", it is simply responsible for normalizing the data inputted in between -1 and 1 instead of 0 and 1 as we've seen in the sigmoid function.

Then the output of this tanh is point-wise or element-wise multiplication with the sigmoid function output. The tanh output can be considered as an encoded, normalized version of the hidden state combined with the current iteration value. We call it feature-extraction, the sigmoid is responsible for scaling how much of "these features" are remembered or forgotten based on how the data looks like before adding it to the cell state.

To summarize all of this, the input gate does feature-extraction once to encode(normalize) the data that is meaningful to the LSTM for its purposes, and another time to determine how memory-worthy this hidden state (old data) and current iteration data (current data) is. The feature-extracted matrix is then scaled by its memory-worthiness before getting added to the cell state.

The Output Gate:

The Output Gate has two main points:

1. Get the value incorporated into the cell state.
2. Form an output hidden state that can be later used the next iteration to make predictions.

It takes as input the current hidden state and the current cell state and applies a sigmoid function and depending on both the sigmoid function output and tanh function output, it decides whether to keep certain "features"/information to communicate with the next time-step. That helps the network remember important information from the past, as well as ignoring irrelevant or redundant ones.

Cell State — Conceptual Interpretation

The cell state, however, is more concerned with the entire data so far. If you're right now processing the word "elephant", the cell state contains information of all words right from the start of the phrase. As you can see in the diagram, each time a time-step of data passes through an LSTM cell, a copy of the time-step data is filtered through a forget gate, and another copy through the input gate; the result of both gates are incorporated into the cell state from processing the previous time-step and gets passed on to get modified by the next time-step yet again. The weights in the forget gate and input gate figure out how to extract features from such information so as to determine which time-steps are important (high forget weights), which are not (low forget weights), and how to encode information

from the current time-step into the cell state (input weights). As a result, not all time-steps are incorporated equally into the cell state — some are more significant, or worth remembering, than others. This is what gives LSTMs their characteristic ability of being able to dynamically decide how far back into history to look when working with time-series data.

To summarize, the cell state is basically the global or aggregate memory of the LSTM network over all time-steps.

Hidden State — Conceptual Interpretation

The characterization (not an official term in literature) of a time-step's data can mean different things. Let's pretend we are working with Natural Language Processing and are processing the phrase "the sky is blue, therefore the baby elephant is crying", for example. If we want the LSTM network to be able to classify the sentiment of a word in the context of the sentence, the hidden state at t = 3 would be an encoded version of "is", which we would then further process (by a mechanism outside the LSTM network) to obtain the predicted sentiment. If we want the LSTM network to be able to predict the next word based on the current series of words, the hidden state at t = 3 would be an encoded version of the prediction for the next word (ideally, "blue" [edited]), which we would again process outside of the LSTM to get the predicted word. As seen, characterization takes on different meanings based on what you want the LSTM network to do. In terms of the mathematics behind it, it can indeed be this flexible because ultimately, what we want the LSTM to do dictates how we train it and what kind of data we use; the weights will tune themselves accordingly to best approximate the answer that we seek. Characterization is an abstract term that merely serves to illustrate how the hidden state is more concerned with the most recent time-step.

It is important to note that the hidden state does not equal the output or prediction, it is merely an encoding of the most recent time-step. That said, the hidden state, at any point, can be processed to obtain more meaningful data.

IV. Methodology

This section describes the structure of the machine-learning model used in this paper and the algorithm behind it.

The overall idea is to transform a variable-length English sentence into a constant-length representation without losing its meaning, and then use that representation to generate a variable-length French sentence that has the same meaning as the English sentence.

We can think of this as two funnels pointed at each other where a sequence from the input domain is compressed into an intermediate representation before being decompressed into another sequence in a different domain.

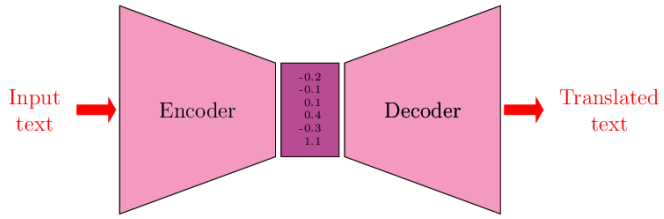

Figure 2: Model illustration

The part in the middle is a 2-by-n (n is latent dimensionality) matrix that holds the meaning of the input sequence (it can be thought of as a different language where all sentences have the same length).

Before we discuss the different algorithms and mechanisms, we must transform our data (letters and words) into a more machine-friendly form (numbers). One way to achieve that is through hot one encoding, where each character is transformed into a vector (phrases are transformed into matrices then).

After our data is prepared, we can then start building our model, which is composed of two sub-models. The encoder and the decoder.

The encoder model looks like this:

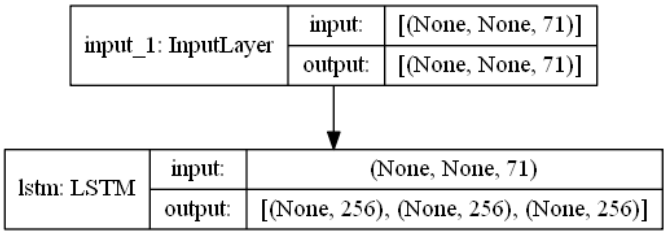

Figure 3: Encoder Model

The first layer is just an input layer (with no weight or biases)
The second layer is our encoder that will transform n inputs into a 2-by-256 matrix (cell state + hidden state)
The encoder algorithm works as follows:

- At each time step feed a character from the input sequence until all characters are fed
- Save the cell states and hidden states of the encoder LSTM layer

After the encoding is done, we need a decoder that looks like this:

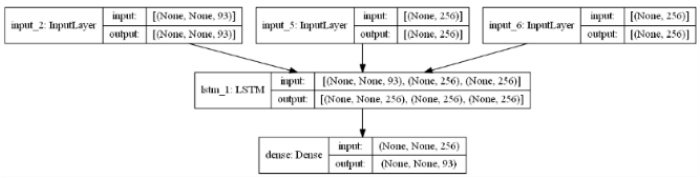

Figure 4: Decoder Model

We can see that there are three input layers

- The first one (input_2) is where the output character from the previous time step is fed
- The second one (input_5) is where the final cell state of the encoder is fed

- And the third one (input_6) is where the final hidden state of the encoder is fed

The decoder algorithm works as follows:

- In the first time step, the start character is provided as an input along the encoder cell and hidden states.
- The decoder then processes that input and gives us back:
    - The cell state
    - The hidden state
    - The next character of the output sequence
- At each time step, we keep feeding the outputs of the previous time step until the end character is returned

V. Results and Discussion

This section describes the results of the machine-learning model used in this paper and the algorithm behind it.

Since the new inputs were not included in the pretrained dataset and the model was not specifically trained on this type of data, the results are inaccurate. Only patterns and structures that the model has learned from the data it was trained on can be used to generate translations.

To improve the translation, we can adjust the pre-trained model with a new dataset containing similar data to the new input to translate. We can also provide more training data, optimize hyperparameters, and tune training parameters to get better results. Additionally, we can use different model architectures or combine multiple models to improve translation accuracy.

Finally we can use a Transformer for machine translation. In fact, the Transformer model architecture is specifically designed for machine translation tasks and has been shown to achieve state-of-the-art results in many translation benchmarks.

A Transformer model is a type of neural network that uses a self-aware mechanism to process an input sequence and produce an output sequence. It has proven to be very effective at handling long-term dependencies and producing fluent translations.

Many pre-trained Transformer models are available, such as: B. Google's popular "Transformer" model (also known as "BERT") or OpenAI's "GPT" model. By providing a dataset of source- and target-language sentence pairs, these pre-trained models can be tailored to specific translation tasks. Note that training a Transformer machine translation model is computationally intensive and requires a lot of resources, including large amounts of data and powerful hardware such as GPUs. Furthermore, the quality of our translations will depend on the quality and quantity of our training data, as well as certain design decisions we make in our model.

VI. Conclusion

In this article, we have explored the use of LSTM auto-encoder sequence-to-sequence model for English to French translation. We have discussed the architecture of the model, the training and evaluation methods,. Our results show that the model still have room for some improvements we can achieve high accuracy in translating English to French by surpassing some diffeculties in the model.

In conclusion, our study highlights the potential of LSTM auto-encoder / sequence-to-sequence model in machine translation tasks, and provides insights into their strengths and weaknesses. Future research can focus on improving the performance of both models and exploring their applications in other NLP tasks. Overall, the advancements in deep learning techniques have opened up new possibilities for machine translation, making it an exciting area of research and development.

Acknowledgment

We would like to express our sincere gratitude to all those who have contributed to this research project. We also thank the developers of the open-source software used in this study, including TensorFlow and Keras, for providing the tools and resources needed to carry out this research.

We are grateful to our colleagues at ENSA FES for their support and encouragement throughout this project. We would like to acknowledge Pr. Hiba Chougrad for their guidance and expertise in the field of machine learning and NLP. We also thank the participants who contributed to the dataset used in this study, without whom this research would not have been possible.

References

[1] Gago, Jennifer Joana & Vasco, Valentina & Łukawski, Bartek & Pattacini, Ugo & Tikhanoff, Vadim & Victores, Juan & Balaguer, Carlos. (2019). Sequence-to-Sequence Natural Language to Humanoid Robot Sign Language. Link

[2] *Wu, Yonghui; Schuster, Mike; Chen, Zhifeng; Le, Quoc V.; Norouzi, Mohammad; Macherey, Wolfgang; Krikun, Maxim; Cao, Yuan; Gao, Qin (2016-09-26). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". arXiv:1609.08144 [cs.CL].*

[3] Recurrent Neural Networks with over 30 LSTM papers by Jürgen Schmidhuber's group at IDSIA K. Elissa, "Title of paper if known," unpublished.

[4] Machine Translation Archive Archived 1 April 2019 at the Wayback Machine by John Hutchins. An electronic repository (and bibliography) of articles, books and papers in the field of machine translation and computer-based translation technology .

[5] Deep Learning: Long Short-Term Memory Networks (LSTMs) - YouTube

[6] Seq2Seq modelling for English to French Translations | by Daniel Chow | Towards Data Science