AIT3001: Final project report

Khong Ngoc Anh 22022549 | Bui Trong Anh 22022572 | Le Thanh Dat 22022627 | Pham Cong Duc 22022607 | Nguyen Minh Duc 21021292 | Nguyen Minh Huong 22022542

***Abstract*—The neural translation problem addresses the challenge of designing computational models capable of accurately translating text or speech between languages while preserving semantic meaning, contextual nuances, and cultural subtleties. Traditional approaches relied on rule-based systems or statistical methods, which were often brittle and required extensive linguistic expertise. Neural machine translation (NMT) emerged as a transformative solution, leveraging deep learning techniques, particularly sequence-to-sequence architectures with recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformers. In this project, we researched and experimented on several different approaches in the neural translation problem. The results provided insights about the efficiency, advantages and drawbacks of those approaches, which help us develop a better understanding about them.**

***Index Terms*—deep learning, convolution, gated recurrent unit, attention, transformers.**

I. Introduction

The history of neural translation reflects the evolution of machine learning techniques in tackling one of the most complex problems in natural language processing: accurately translating text or speech across languages. From its early foundations to the current state-of-the-art systems, advancements in neural architectures have played a pivotal role in improving translation quality, speed, and scalability.

The Multilayer Perceptrons (MLPs) were among the first to demonstrate the potential of deep learning for processing language data. However,due to their inability to handle sequential information effectively, their application in translation was limited. The development of Convolutional Neural Networks (CNNs) brought a significant breakthrough. Initially designed for image processing, CNNs were adapted for sequence tasks, including machine translation. Their parallel processing capabilities and ability to capture local dependencies made them faster than traditional models. However, they struggled with long-range dependencies, which are essential for translating complex sentences. Recurrent Neural Network (RNNs) revolutionized neural translation by introducing mechanisms to process sequential data more effectively, allows efficient learning of long-range dependencies, making them a popular choice in Sequence-to-Sequence (Seq2Seq) models. These architectures laid the groundwork for many modern translation systems, providing a robust framework for encoding and decoding linguistic information.The appearance of the Transformer architecture marked a paradigm shift in neural translation. Unlike RNNs or GRUs, Transformers rely entirely on self-attention mechanisms to process sequences, enabling them to model dependencies over arbitrary lengths efficiently. This design not only improved translation accuracy but also allowed for unparalleled scalability, making it a significant progress.

Throughout the development of the machine translation field, numerous architectures have been introduced, each with its own strengths and weaknesses. This makes the selection of a model for use a matter that requires careful consideration. Therefore, in this exercise, we conducted research and experiments on three models, including: The Convolutional Sequence to Sequence model (Conv-Seq2Seq), the Recurrent Neural Network with Gated Recurrent Unit (RNN-GRU) and the Transformer. Firstly, we will talk about our research on the architectures. Then we will show the experiments we conducted. Finally, we will discuss about our results and the potential for improvement.

II. Methods

A. Convolutional Sequence to Sequence

The Sequence-to-Sequence (Seq2Seq) model is a popular architecture used in tasks like machine translation, text transformation, and other deep learning applications involving input and output sequences. In this setup, instead of traditional RNNs, LSTMs, or Transformers, the model employs Convolutional Neural Networks (CNNs) to speed up computation and enable parallel processing [1].

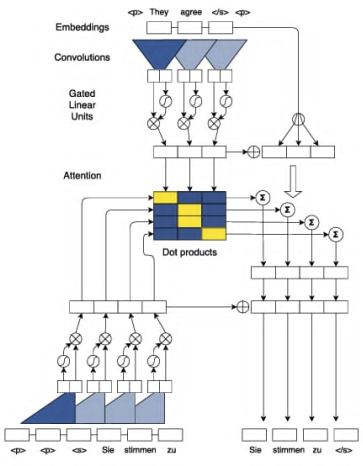

Fig. 1. Convolutional Sequence to Sequence

Like the traditional Seq2Seq, the model's architecture consists of 2 main components, the Encoder and the Decoder:

*1) **Encoder**:* The Encoder's role is to process the input sequence and generate a semantic representation. The input is the source sequence, and the output is combined representation that merges hidden features and embeddings, which serves as input for the decoder. The Encoder is made from:

- **Embedding Layer**: Converts tokens in the input sequence into continuous vector representations (embedding vectors). Combines embedding with positional encoding to include word position information in the sequence.
- **Linear Projection**: Maps embeddings into a feature space
- **Convolutional Layers**: A stack of 1D CNN layers with a kernel size of 3. Performs convolutions and applies Gated Linear Units (GLU) to capture non-linear relationships in the data.
- **Residual Connections**: Links input directly to the output of each convolutional layer to improve gradient flow during training.

*2) **Decoder**:* The decoder generates the target output sequence token by token, using the encoded input and its own previous states. The input is the target sequence , starting with a special start-of-sequence token, and the output is a probability distribution predicting the next token in the target sequence. The Decoder consists of:

- **Embedding Layer**: Similar to the encoder, it combines embeddings with positional encoding to process the target sequence for CNN input.
- **Attention Mechanism**: Uses a multiplicative attention method to focus on relevant parts of the encoded sequence while decoding each token.
- **Convolutional Layers**: Processes the target sequence through CNN layers with residual connections to maintain gradient optimization.
- **Fully Connected Layer**: Converts the features into probability distributions over possible output tokens.

B. Gated Recurrent Unit

GRU, or Gated Recurrent Unit, is an improvement on the Recurrent Neural Network (RNN) [2]. GRU was designed to address the vanishing gradient problem encountered by RNNs. GRU is very similar to Long Short-Term Memory (LSTM). Like LSTM, GRU uses gates to control the flow of information but has a simpler architecture than LSTM. GRU layers do not use memory blocks with feedback loops within a memory cell.

The architecture of a GRU cell is illustrated in figure 2, comprising two gates: the reset gate and the update gate. These two gates determine whether to reject or accept information passing through the GRU cell.

- **Reset Gate**: Determines the amount of past information to forget. This decision is made using the sigmoid activation function. If the sigmoid value equals 1, the data is passed into the GRU algorithm; if the value is 0, the data is discarded. The inputs to the reset gate include the current input x_t and the previous hidden state h_{t-1}. The formula is as follows:

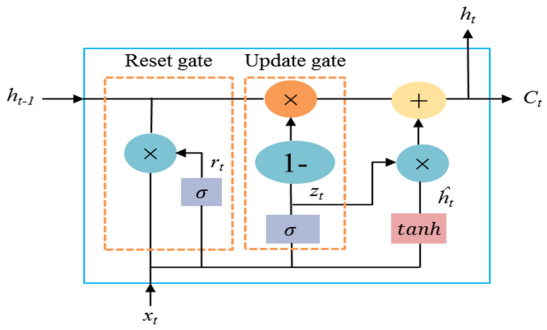

Fig. 2. Gated Recurrent Unit

$$r_t = \sigma(W_r[h_{t-1}, x_t]) \tag{1}$$

where W_r represents the weights of the reset gate, which are updated during the training process.

- **Update Gate**: Determines which information will be updated and passed to the future state. This gate depends partially on the previous state. The update gate also uses the sigmoid activation function to update the state of the cell, with the sigmoid output value ranging between 0 and 1:

$$z_t = \sigma(W_z[h_{t-1}, x_t]) \tag{2}$$

where W_z represents the weights of the update gate, which are updated during the training process.

- The output z_t is multiplied by a tanh function to create the new cell state $\hat{h}t$. The tanh function outputs values in the range of -1 to 1. The result of this multiplication is added to the current cell state Ct, where h_t represents the current hidden state of the cell. The formula is as follows:

$$h_t = (1 - z_t) \times h_{t-1} + z_t \times \hat{h}_t \tag{3}$$

The model architecture consists of two main components: the encoder and the decoder:

1) **Encoder**:
   - **Embedding layer**: The input size is the vocabulary size, and the output size is the embedding dimension. This layer converts word indices into embedding vectors.
   - **GRU layer**: Used to extract sequential features from the input data.
2) **Decoder**:
   - **Embedding layer**: Similar to the encoder's embedding layer.
   - **GRU layer**: Processes sequential data from the previous output.
   - **LuongAttention mechanism**: Integrated by computing the dot product between the GRU output and the encoder states. This attention mechanism uses the softmax function as its activation function.

- Uses the tanh activation function.
- The output layer employs a linear layer, followed by the softmax function to predict the next word.

C. Transformers

In machine learning, attention is a method which determines the relative importance of each component in a sequence relative to the other components in that sequence

In [3], attention is defined as a function mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key

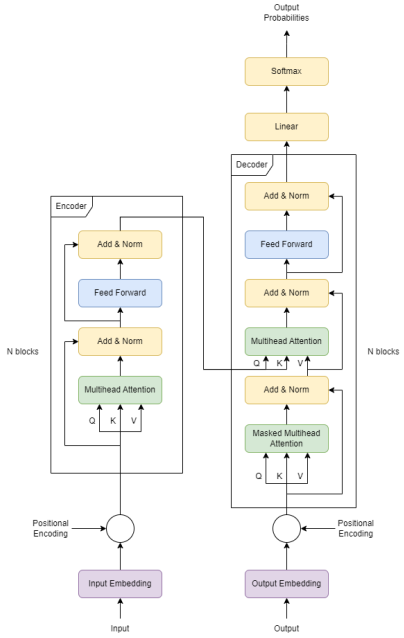

Fig. 3. Transformer Architecture

The transformer model's architecture for sentence translation tasks centers around attention mechanisms. However, the structure can vary across different applications, depending on particular needs. The original transformer design has two main components: the encoder and the decoder. These components can be executed multiple times, depending on the task requirements. Each component includes multiple layers with the attention mechanism. Particularly, the architecture contains multiple "Attention Heads", which execute the attention mechanism in parallel multiple times [3].

*1) **Scaled Dot-product Attention**:* In particular, the attention function used in Transformer is called "Scaled Dot-product Attention" [3]. The input consists of queries and keys of dimension d_k, and values of dimension d_v. The weights of the values is calculated by taking the dot products of the query with all keys, divide each by $\sqrt{d_k}$ then apply a softmax function. In practice, three matrices **Q, K, V** are used

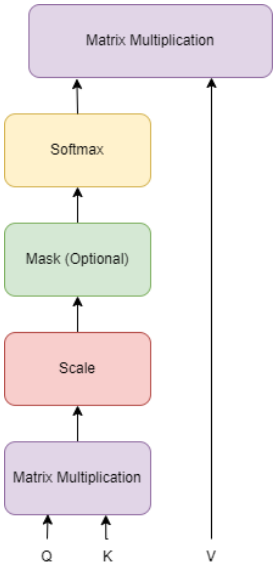

Fig. 4. Scaled Dot-product Attention

to represent a set of queries, keys and values, respectively. The function is:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{4}$$

where

- $\mathbf{Q}$ is the query matrix, $\mathbf{Q} \in \mathbb{R}^{d_{model} \times d_k}$
- $\mathbf{K}$ is the query matrix, $\mathbf{K} \in \mathbb{R}^{d_{model} \times d_k}$
- $\mathbf{V}$ is the query matrix, $\mathbf{V} \in \mathbb{R}^{d_{model} \times d_v}$
- d_k is the dimension of query and key vectors
- d_v is the dimension of value vectors
- d_{model} is the dimensionality of the representations used as input of attention, which is the same as the dimensionality of the output. It is also the same as the dimensionality of the word embedding vector

Q, K, V are extracted from the input matrix through the learned weight matrices.

*2) **Multi-headed Attention**:* The authors of Transformer did not perform a single attention step with keys, values and queries. Instead, the keys, values and queries are linearly projected h times with different, learned projections to d_k, d_k and d_v dimensions, respectively. The attention can be computed in parallel with these projected vectors, resulting in outputs with dimension of d_v. The outputs are concatenated and once again projected, resulting in the final values. The matrix of multi-head attention is:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, ..., \text{head}_h)\mathbf{W}^O \tag{5}$$

with each head is:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \tag{6}$$

where the projections are parameter matrices:

- $\mathbf{W}_i^Q$ is the projection matrix of $\mathbf{Q}$, $\mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$
- $\mathbf{W}_i^K$ is the projection matrix of $\mathbf{K}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$

- $\mathbf{W}_i^V$ is the projection matrix of $\mathbf{V}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$
- $\mathbf{W}^O$ is the learned weight matrix $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{model}}$

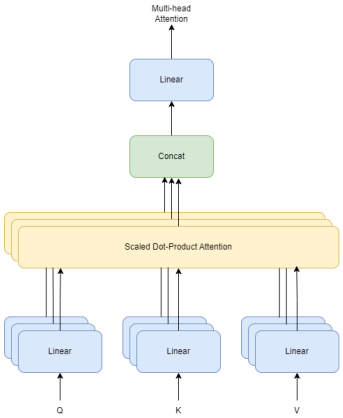

Fig. 5. Multi-head Attention

Multi-head attention simultaneously focus on information from various representation subspaces at different positions. As a result, it improves the representation of input contexts by combining information from different features of the attention mechanism over a range, whether short or long. This is equivalent to focus on different part of the texts, then extract different patterns, which leads to better network performance.

*3) **Encoder**:* The encoder module is composed of two main layers: the multi-head attention layer and the feed-forward layer. It also includes residual connections around these layers, along with two add and norm layers [3]. The encoder module processes an input of embeddings, which is generated based using the embedding and positional encoding layers. From the embedding input, three parameter matrices—query (**Q**), key (**K**), and value (**V**)—are created, along with positional data, and are passed through the multi-head attention layer. Following this step, the feed-forward layer addresses the issue of rank collapse that may occur during computation. Additionally, the weights is normalized to prevent gradient explosion, reducing layer dependencies. To prevent gradient vanishing, residual connections are added to the outputs of both the attention and feed-forward layer.

*4) **Decoder**:* The decoder is similar to the encoder , featuring feed-forward layers, multi-head attention, residual connections, and add and norm layers. However, it also includes masked multi-head attention layers that use masking operations to ignore future predictions, focusing only on past outputs [3]. The attention mechanism is applied twice in the decoder: first, to compute attention within the elements of the target output, and second, to find attention between the encoded inputs and the target output. Each attention vector is then processed the feed-forward layer to enhance the output's interpretability for the subsequent layers. The final decoding result is then processed by linear layer and softmax layer to produce the transformer's final output. This process is repeated until the end of the sequence.

*5) **Feed forward layer**:* Each of the layers in the encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. The network is made from two linear layer with a ReLU activation in between.

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{7}$$

where:

- $\mathbf{x}$ is the input of the feed-forward
- $\mathbf{W}_1, \mathbf{b}_1$ is the weight and bias of the first linear transformation layer, respectively
- $\mathbf{W}_2, \mathbf{b}_2$ is the weight and bias of the second linear transformation layer, respectively

The linear transformations are the same at every position, but the parameters are different from layer to layer.

*6) **Embeddings and Softmax**:* The embedding layers use learned embeddings to convert the input and output tokens to vectors of dimension d_{model}. Then the learned linear transformation and softmax function is used to convert the decoder output to predicted next-token probabilities. In Transformer, two embedding layers and the pre-softmax linear transformation share the same weight matrix, with the weights multiplied by $\sqrt{d_{model}}$ in the embedding layers.

*7) **Positional encoding**:* Since transformers do not use recurrence or convolution, it is essential to incorporate information regarding the relative or absolute position of the tokens in the sequence to enable the model to utilize the order of the sequence. To solve this problem, "positional encodings" were added to the input embeddings of both input and output. The positional encodings and the embeddings share the same dimension d_{model} , so they can be added together. The original Transformer used sine and cosine functions of different frequencies for positional encodings:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \end{aligned} \tag{8}$$

where:

- pos is the position
- i is the dimension

III. Experiments

We conducted experiments with the architectures mentioned in section II. We used a public dataset from the Huggingface platform [5]. The details will be further elaborated in the following sections:

A. The dataset

The dataset we used has a total of over 250k pair of English-Vietnamese sentences. The dataset is splitted into three sets: The training set, the validation set and the test set with 8:1:1 ratio. We perform data analysis and gained some useful statistics.

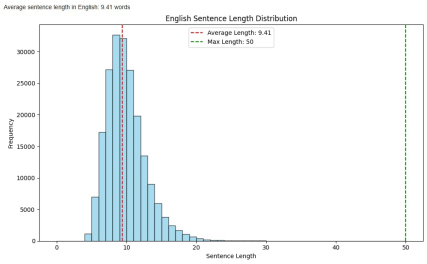

Fig. 6. Distribution of English sentence length

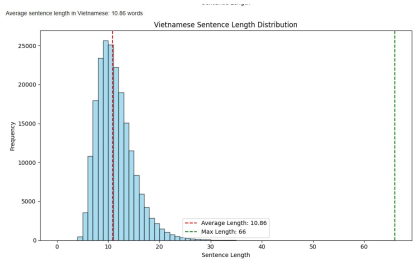

Fig. 7. Distribution of Vietnamese sentence length

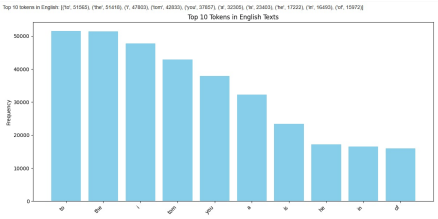

Fig. 8. Top 10 most common English words

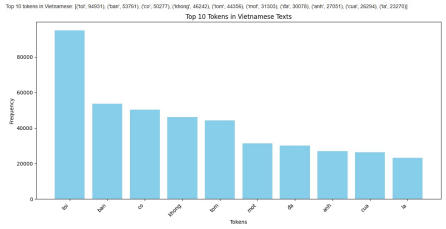

Fig. 9. Top 10 most common Vietnamese words

B. The models

In the experiments, our models have the following settings:

1) Convolutional Sequence to Sequence:
   - embed_size: 256
   - hidden_size: 256
   - encoder_layers and decoder_layers: 6
   - dropout: 0.1
   - kernel_size: 3
2) Gated Recurrent Unit:
   - embedding_dim: 256
   - encoder_units: 512
   - decoder_units: 512
3) Transformer
   - d_model: 512
   - n_layers: 6
   - heads: 8
   - dropout: 0.1

C. The Results

The training hyperparameters and results of our models are shown in table I and figure 10, 11, 12

Model	Conv-Seq2Seq	GRU	Transformer
Trainable parameters	13,140,240	13,506,254	66,239,419
Learning rate	0.0001	0.0001	0.0001
Batch size	256	64	1500
Epochs	25	80	25
Loss function	CrossEntropy	CrossEntropy	CrossEntropy
Optimizer	Adam	Adam	Adam
BLEU	0.3479	0.3478	0.3403

TABLE I
THE MODELS HYPERPARAMETERS AND PERFORMANCE

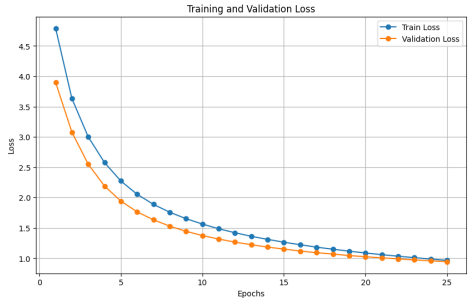

Fig. 10. Training loss and validation loss of Convolutional Seq2Seq

Fig. 11. Training loss and validation loss of GRU

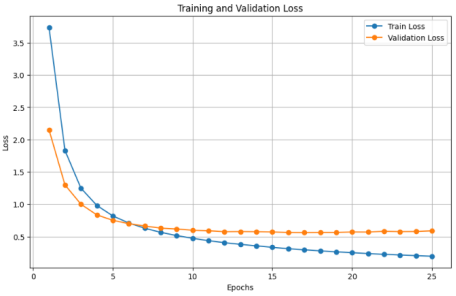

Fig. 12. Training loss and validation loss of Transformer

IV. Discussions

From the results of our experiment, we can derive that: The Convolutional Seq2Seq model converges slower than the other two models but achieves convergence in both training loss and validation loss. The GRU model converges quickly but encounters overfitting early on, where the training loss decreases while the validation loss increases. The Transformer model converges relatively quickly and does not exhibit overfitting like the GRU.

In our training process, the batch size when we train Transformer is significantly greater than other model. That is due to the Transformers being a very complex model, which make it has a lot more trainable parameters than the other two model. Therefore, to increase the speed of training, we have to increase the batch size when training Transformer. Yet, this will potentially decrease the accuracy of the model, as we can see the BLEU score of Transformers is a bit lower than Convolutional Seq2Seq and GRU. This method also not plausible if we have limited storage capacity, as a larger batchsize mean the model have to process more data in the same time, which can vastly increase the memory needed.

V. Conclusion

Neural translation has significantly advanced the field of natural language processing, offering unprecedented accuracy and fluency in machine translation. By leveraging powerful architectures such as Sequence-to-Sequence models, attention mechanisms, and Transformers, these systems have successfully captured the complexity of human language. In this project, we have successfully implement and experiment on several models. Yet, our results are far from ideal when compared to the current state-of-the-art, which shows unprecedented performance. That is because the modern models are built on architectures that surpassed the architectures in this work by a wide margin. Therefore, our future direction will include improving existing works while also researching and implementing the newer architectures in this field. We hope that our work can serve as a resource for further research and exploration of this field.

References

[1] Gehring, Jonas, et al. "Convolutional sequence to sequence learning." International conference on machine learning. PMLR, 2017.

[2] Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).

[3] Vaswani, A. "Attention is all you need." Advances in Neural Information Processing Systems (2017).

[4] Bahdanau, Dzmitry. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[5] https://huggingface.co/datasets/harouzie/vi_en-translation