VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MACHINE LEARNING (CO3117)

CC01 — Assignment Report

Machine Translation and Applying Transformers to Solve the Problem

Supervisor: PhD Nguyen Duc Dung, CSE-HCMUT
Students: Le Thi Phuong Thao - 2252757
Nguyen Thuy Tien - 2252806

Ho Chi Minh City, January 2025

Contents

1 Abstract

This report explores the Transformer architecture in depth and its application to machine translation tasks. The Transformer utilizes self-attention mechanisms to efficiently capture long-range dependencies, overcoming limitations of previous models like RNNs and LSTMs. The report examines the architecture behind the Transformer, including multi-head self-attention, positional embeddings, and other important components, before discussing how it has improved machine translation by addressing challenges faced in earlier models.

The implementation of the Transformer base model is presented in Colab, and we run the code on Kaggle to train the model on a machine translation dataset translating from English to German. Key aspects such as the encoder-decoder structure, attention mechanism, and optimization through the categorical cross-entropy loss function are explored. The report also discusses other approaches to solving machine translation problems and compares them with the Transformer.

Github: Machine Learning Assignment Github
Drive: Output Sample

2 Motivation

2.1 Machine Translation

Machine Translation is a sub-field of computational linguistics that focuses on developing systems capable of automatically translating text or speech from one language to another.
Machine Translation is the task of automatically converting one natural language into another, preserving the meaning of the input text or speech and producing text or speech in the output language. The goal of Machine Translation is to produce translations that not only grammatically correct but also convey the meaning of the original content accurately.

2.2 History of Machine Translation

There have been three primary uses of machine translation in the past:

- Rough translation, conveys the "gist" of a foreign statement or document but is riddled with inaccuracies
- The original source content is written in a limited language that makes machine translation easier, the outputs are often edited by a person to rectify any flaws
- Restricted-source translation is totally automated for highly stereotyped language (weather report, etc)

Machine Translation is one of the first applications for computers that were imagined (Weaver, 1949) and there are many approaches to solve the problem.

The traditional models, RNN and CNN, face significant challenges in handling sequential data because they lack the ability to effectively capture long-term context and relationships

between distant elements in the data.
The attention mechanism began to be used in seq2seq models to handle this problem, with Neural Machine Translation (NMT) being the first model to apply attention for machine translation.

This led to the development of the Transformer model, introduced in the paper *"Attention is All You Need" by Vaswani et al. (2017)*, revolutionized the way sequential data is processed, overcoming many limitations of traditional RNNs and CNNs. It allows for more efficient capturing of relationships between tokens, while also enabling easier parallelization and handling of sequential data.

3 Approaches to solve the problem

There are several approaches that have been developed to solve this problem. From Rule Based Machine Translation (RBMT) in 1950s, Example Based Machine Translation (EBMT) in 1980s, Statistical Machine Translation (SMT) in 2000s, and the newest are Neural Machine Translation (NMT) using Deep learning, which appeared from 2015 till now. In this report, we will only discuss the two newest one: **SMT** - a type of machine learning, and **NMT** - a type of deep learning.

3.1 Statistical Machine Translation (SMT) [1]

SMT is a data-driven approach that uses probabilistic models to determine the most likely translation for a given source text.

SMT focuses on creating models that estimate the probability of a target-language sentence $P(T)$, given a source-language sentence S, represented as $P(T|S)$. This probability is broken down into components using Bayes' theorem:

$$P(T|S) = \frac{P(S|T) \cdot P(T)}{P(S)}$$

where:

- **Language Model** $P(T)$: How probable is the sentence T in the target language. This will help to choose the grammatically correct sentence.
- **Translation Model** $P(S|T)$: The probability that S comes from T.

The decoding process of this approach, which choosing the most likely translation can be capture by the formula:

$$\hat{T} = \arg\max_{T} P(S|T)P(T)$$

We do not need to consider the full application of Bayes' Theorem since the probability of the source sentence $P(S)$ is independent of T, it does not affect the decoding process. For this approach, the reordering task will base on different grammatical structures:

- **Word-based SMT:**
  - Translates individual words without considering phrases or higher linguistic units. Hence, it will lead to a lack of context awareness. This model also need a precise word alignment
  - Example: For "I have a pen," it may translate "pen" as "bút" without considering the phrase "a pen."
- **Phrase-based SMT:**
  - Translates sequences of words (phrases) rather than individual words, capturing local context. This structure handles idiomatic expressions and word reordering better than word-based models.
  - Example: For "I have a pen," it treats "a pen" as a unit and translates it to "mt cây bút," improving accuracy.
- **Syntax-based SMT:**

- Incorporates syntactic structure using parse trees to understand grammatical relationships. It can handles long-range dependencies and complex reordering effectively, making translations more fluent and accurate.
- Example: For "The boy who is running is my brother," it correctly reorders "who is running" based on its grammatical role.

3.2 Neural Machine Translation (NMT)

Unlike earlier methods that relied on linguistic rules or probabilistic models, NMT uses deep learning to directly optimize translation quality in an end-to-end manner. NMT can learn from large-scale bilingual datasets, it also has ability to generate translations by capturing complex relationships between words and sentences in different languages.

The core of NMT is the encoder-decoder architecture. The encoder processes the input sentence (source sentence) into a continuous vector representation, often referred to as the "context vector". The decoder then uses this context vector to generate the translated sentence, one word at a time. While this foundational approach has proven effective, various enhancements such as attention, self-attention have addressed some of its limitations.

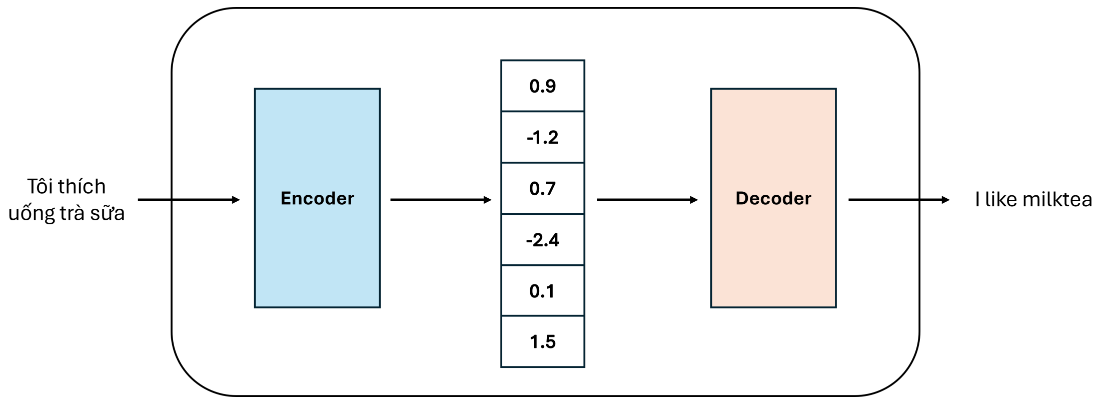

Figure 1: *The Encode-Decode process in NMT*

3.3 Models in NMT

3.3.1 Recurrent Neural Networks (RNNs)

RNNs are neural networks that process sequences of data by maintaining a hidden state, which acts as a memory of previous computations. Unlike feedforward networks, RNNs can handle variable-length input sequences because they pass information from one time step to the next.

3.3.1.1 Deal with Sequential Data Languages are inherently sequential, the word order will affect meaning of the sentence. For instance: "The cat chased the mouse." and "The mouse chased the cat." have completely different meanings. Machine translation relies heavily on the correct understanding of this sequential structure to generate accurate translations. RNN can deal with this task since at each step, it considers the current word and the context provided by previous words by using the encoder-decoder framework:

The encoder will compute a representation s for each source sentence. The decoder which generates one target word at a time will decompose the conditional probability as:

$$log\ p(y|x) = \sum_{i=1}^{n} log\ p(y_j|y_0 \dots y_j - 1, s)$$

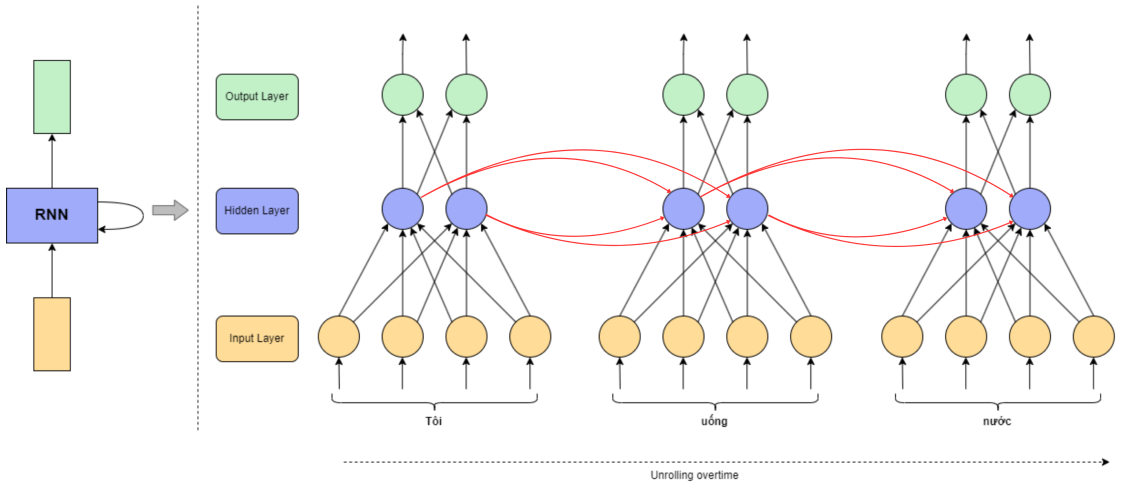

Figure 2: *RNN over each time step*

3.3.1.2 Gating mechanism for handling sequential dependency

An variant of RNN is Long-short term memory neural network (LSTM) using Gating mechanism which include:

- **Forget Gate:** Decides what past information to discard based on its relevance to the current word in the sequence.
- **Input Gate:** Updates the memory cell with new information relevant to the current word.
- **Cell State:** Combines past and current information in the memory cell.
- **Output Gate:** Produces the current hidden state which encapsulating the sequence's context up to this word.

Although RNN can solve the problem of sequence characteristic, It uses a huge computational cost to process one word at a time, the next word need to wait for the previous word to be calculate. Hence the ability of parallel computing in CPU and GPU cannot be utilized. In the other hand, for very long sentences, compressing the entire source sentence into a single context vector through gradient calculation can cause a loss of sequential information.

To overcome these challenges, a new approach was introduced in 2017: Transformer Architecture using Attention mechanism, which is a type of NMT.

4 Transformer Architecture

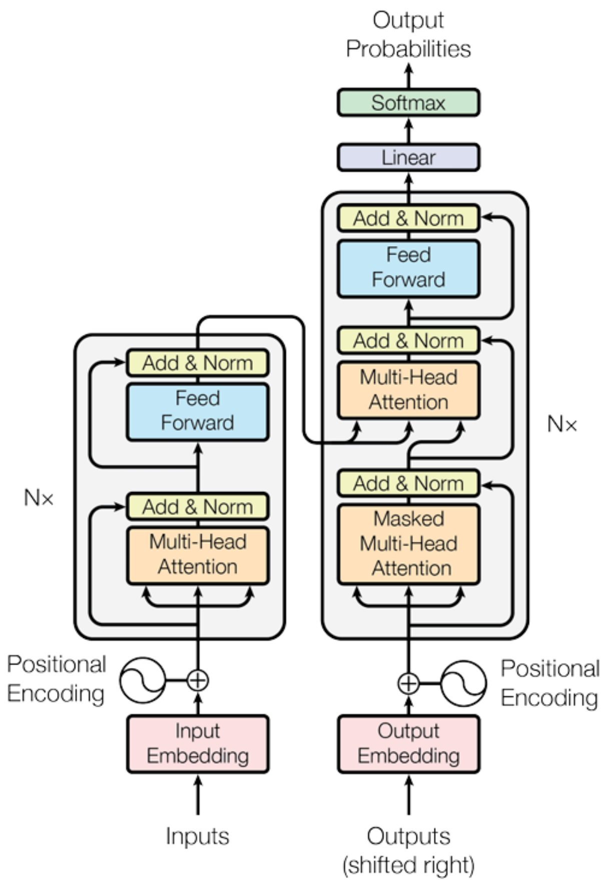

Figure 3: *The Transformer - Model Architecture*

4.1 Overview of Transformer

Overall, the Transformer using stacked self-attention and point-wise, consists of two main components: the Encoder and the Decoder, designed to process sequential input data in parallel.
Encoder: Process the input and generates a hidden presentation. It consists of several identical layers, each with two main sub-components:

- Self-Attention Mechanism: Allows each word in the input to ”attend to” other words, capturing their relationships.
- Feed-Forward Neural Network (FFN): A simple neural network applied independently to each position in the sequence.

Decoder: Processes the hidden representation from the Encoder and generates the output sequence. It also consists of several identical layers, each with three main sub-components:

- Masked Self-Attention: Ensure the Encoder only "attends to" previous words, avoiding future information
- Encoder - Decoder Attention: Focuses on the relevant parts of the Encoder's output
- Feed-Forward Neural Network (FFN)

4.2 Attention Mechanism

Attention Mechanism is a technique used in machine learning and artificial intelligence to improve the performance of models by focusing on relevant information.
Attention Mechanism allows models to attend to different specific parts of the input data, assigning varying degrees of weight to different elements based on its relevance to the current task. The attention mechanism typically involves three key components:

- Query: Represents the current context or focus of the model
- Key: Represents the elements or features of the input data
- Value: Represents the values associated with the elements or features

The attention mechanism computes the attention weights by measuring the similarity between the query and the keys. The values are then weighted by the attention weights and combined to produce the final output of the attention mechanism.

4.3 Self-Attention (Scaled-dot Attention)

In comparison with the attention mechanism, self-attention is a more general mechanism for capturing relationships within a sequence. Self-attention (intra - attention) is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence.

4.3.1 Query, Key and Value

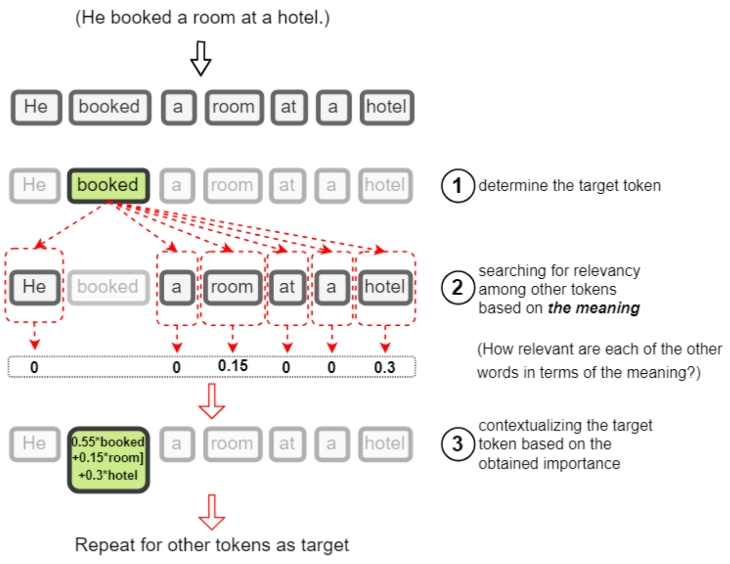
(He booked a room at a hotel.)
He booked a room at a hotel
1 determine the target token
2 searching for relevancy among other tokens based on the meaning
(How relevant are each of the other words in terms of the meaning?)
0 0 0.15 0 0 0.3
0.55*booked +0.15*room] +0.3*hotel
3 contextualizing the target token based on the obtained importance
Repeat for other tokens as target

Figure 4: *An example - Q, K, V*

The Transformer operates on the principle of transforming a query and its corresponding key-value pairs into an output, where query (Q), key (K), and value (V) are d-dimensional vectors.

- Query (Q): A vector that represents the current input element or token for which we want to find relevant information from the entire input sequence.
  Acts as the "question" or "focus point" in the attention mechanism.
- Key (K): A vector that represents each element in the input sequence.
  It serves as a reference or "index" to determine the relevance of each element to the query.
- Value (V): A vector that contains the actual content or information associated with each key.
  The attention mechanism retrieves and combines values based on the relevance of their corresponding keys to the query.

After that, by calculating the relevance (or attention scores) between queries and keys, the Transformer selectively augments the values to generate the output representation.
For each token:

- We compare its Query vector to all other tokens' Key vectors.
- Calculate a vector similarity score between each pair (the dot-product similarity). This score represents the semantic distance between words: words that are semantically related to the current word will have a high score, while unrelated words will have a lower score. These scores are then scaled by multiplying with the factor $\frac{1}{\sqrt{d_k}}$.

- Transform these similarity scores into weights by scaling them into the range $[0, 1]$ (using Softmax).
- Add the weighted context by weighting their corresponding Value vectors.

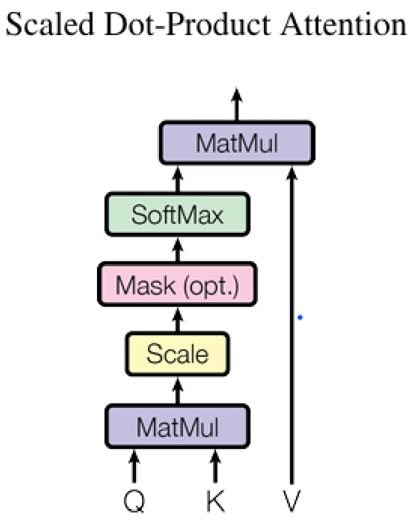

Figure 5: *Scaled-dot Attention*

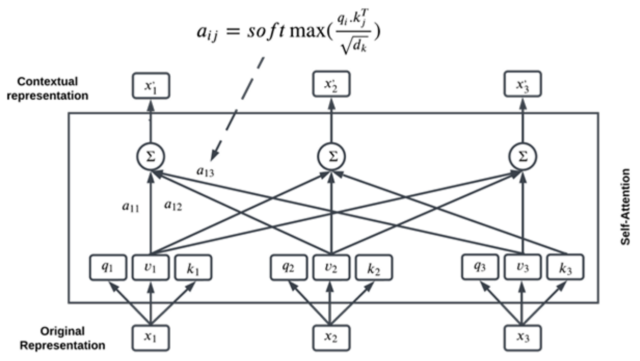

Figure 6: *Self Attention*

In practice, the query, key, and value vectors are different representations of a word in vector form and are arranged into the matrices $Q \in \mathbb{R}^{n\times d}$, $K \in \mathbb{R}^{n\times d}$, and $V \in \mathbb{R}^{n\times d}$, respectively, where n is the number of words in the sequence and d is the dimension. The attention mechanism is then represented by these equations:

- $Q = XW_Q, \quad K = XW_K, \quad V = XW_V$

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$
$$C = \text{attn}(Q, K, V) = AV$$

where $X \in \mathbb{R}^{n\times d'}$ is the matrix of encoded input, $W_i \in \mathbb{R}^{d'\times d}$ are weight matrices, $A \in \mathbb{R}^{n\times n}$ is the attention distribution matrix, and $C \in \mathbb{R}^{n\times d}$ is the context matrix.

4.4 Multi-Head Attention

In the Transformer architecture, multi-head attention is used in both the encoder and decoder, but with a key difference in the decoder: the use of masked multi-head attention.

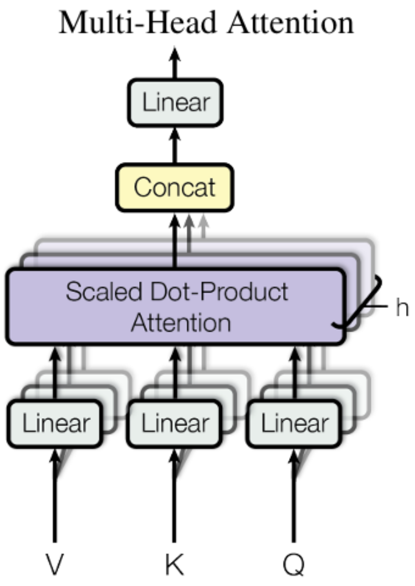

Figure 7: *Multi-Head Attention*

4.4.1 Multi-Head Attention

Multi-Head Attention is a key component of the Transformer architecture, enabling the model to focus on different parts of the input sequence simultaneously. It extends the basic scaled dot-product attention mechanism by applying it multiple times in parallel, with each "head" learning to capture distinct aspects of the input.
Instead of using just one self-attention mechanism, multi-head attention divides the initial dataset into N heads, allowing multiple self-attention mechanisms to run in parallel. The idea behind this approach is to let each head independently explore local information, which helps the model generalize better and "collect more diverse information." This enables the Transformer to capture a broader range of dependencies and relationships within the input data, enhancing its overall learning capacity.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

where

$$\text{head}_i = \text{Attention}(QW_{Q_i}, KW_{K_i}, VW_{V_i})$$

The projections are parameter matrices:

$$W_{Q_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_{K_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_{V_i} \in \mathbb{R}^{d_{\text{model}} \times d_v}, \quad W_O \in \mathbb{R}^{hd_v \times d_{\text{model}}}.$$

In the paper, they employ $h = 8$ parallel attention layers, or heads. For each of these, we use:

$$d_k = d_v = \frac{d_{\text{model}}}{h} = 64.$$

Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.

- **Divide into Heads:** Split the input embedding into `NUM_HEADS` heads, where each head has a dimension of `NUM_TOKENS` × `DIM/NUM_HEADS`.
- **Initialize Weight Matrices:** Define and initialize three weight matrices: W_Q, W_K, W_V with dimensions `DIM` × `DIM/NUM_HEADS`.
- **Compute Q, K, and V:** Obtain Q, K, and V by multiplying the input embedding with their respective weight matrices, and then calculate the attention scores.

4.4.2 Masked Multi-Head Attention

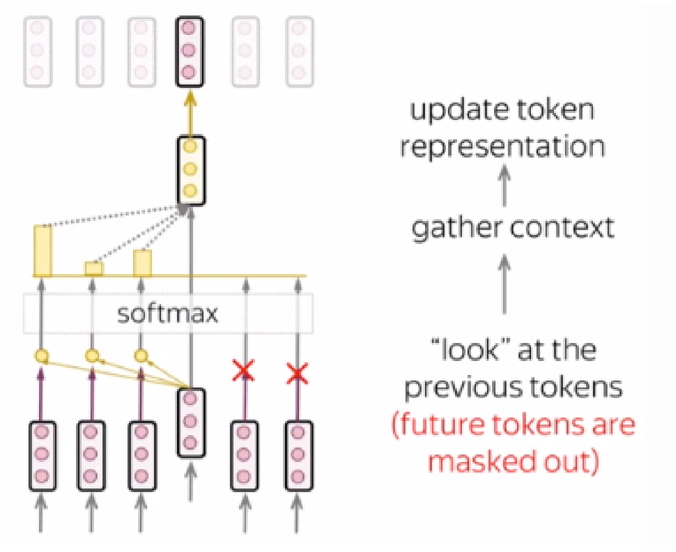
update token representation
gather context
softmax
"look" at the previous tokens
(future tokens are masked out)

Figure 8: *Masked Self-Attention*

In the decoder, **masked multi-head attention** is introduced to enforce the autoregressive property. The masking ensures that, during training and inference, the decoder cannot "see" future tokens in the sequence it is generating.
This is critical for tasks like text generation (e.g., machine translation), where the model generates the output sequence one token at a time.
The mask is applied to the attention scores before the softmax operation.

For each position i, only tokens from positions 1 to i are visible.
All future token positions are masked by setting their scores to $-\infty$, effectively assigning them a probability of zero in the softmax.

To apply this, we create **a causal mask matrix** (a square matrix of size `NUM_TOKENS` $\times$ `NUM_TOKENS`), ensuring that each token only attends to preceding tokens by assigning a value of $-\infty$ to future positions. Next, we calculate the scaled dot-product attention, apply the causal mask matrix, and compute the softmax.

4.5 Cross Attention

While self-attention captures relationships between elements within a single input sequence, cross-attention captures relationships between elements of two difference input sequences.
Cross-attention computes attention scores based on Query (Q), Key (K) and Value (V) vectors.
In cross-attention, these vectors are derived from different sequences: Q from the target sequence (decoder input) and K and V from the source sequence (encoder output). In machine translation problem, it ensures the translation output attend to the original sentence.

4.6 Components in Transformer Architecture

4.6.1 Input embedding

Input embeddings are basically vector representations of discrete tokens like words, sub-words, or characters. These vectors capture the semantic meaning of the tokens that enables the model to understand and manipulate the text data effectively.
The input embedding layer in the Transformer architecture is the first step in processing input sequences. Its purpose is to convert discrete tokens into continuous vector representations that the model can process.
The role of the input embedding sub-layer in Transformer is to convert the input tokens in a high-dimensional space d_model = 512.

4.6.1.1 Word Embedding Each token in the input sequence is mapped to a fixed-dimensional dense vector using a pre-learned embedding matrix. These embeddings capture the semantic meaning of the tokens and allow the model to generalize across similar words.

Given a vocabulary size N and an embedding dimension d_{model}, the embedding matrix $W \in \mathbb{R}^{N \times d_{\text{model}}}$ is used to transform token indices into vectors.

4.6.1.2 Positional Embedding Since the Transformer model does not have a sequential architecture like RNNs, a *positional embedding* layer is added before feeding the input into the encoder and decoder to ensure the positions of the words in the sentence are captured. This embedding represents the relative or absolute position of the tokens in the sequence.
The positional encodings have the same dimension d_{model} as the embeddings, allowing them to be summed together.
We use sine and cosine functions of different frequencies:

$$\text{PE}(\text{pos}, 2i) = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

$$\text{PE}(\text{pos}, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

where pos is the position and i is the dimension. The final input to the Transformer is the sum of the word embedding and positional embedding for each token:

$$\text{Input Representation} = \text{Word Embedding} + \text{Positional Embedding}$$

4.6.2 Feed Forward

After passing through the attention layer in the Transformer, the output is a feature vector with weighted context. The attention layer computes attention scores between a token and all other tokens in the sentence (including itself). These scores represent the importance of other tokens to the current token.
The Feed Forward Layer is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

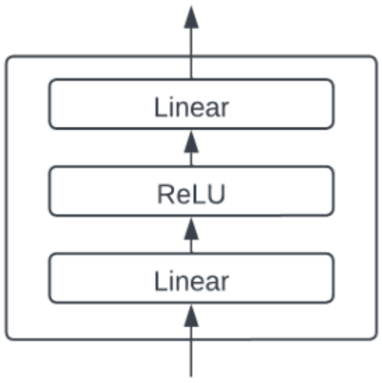

Figure 9: *Feed Forward Network*

In the encoder, after passing through the attention layer, the output vector continues to pass through the following steps in the feed-forward network:

1. The first fully connected layer projects the input into a higher-dimensional space with a weight matrix W_1 and bias b_1.
2. The output of this layer is passed through a ReLU activation function.
3. The second fully connected layer projects the output back to the original dimension d_{model} using a weight matrix W_2 and bias b_2.

Mathematically, this can be written as:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

where x is the input to the feed-forward network.

4.6.3 Add & Norm

The Add & Norm layer in the Transformer model normalizes the output of the multi-head attention mechanism. This normalization improves the model's ability to converge faster during training by stabilizing the learning process. The residual connections (adding the input back to the output) help prevent issues like vanishing gradients, while layer normalization ensures that the output has a consistent distribution, speeding up convergence and making the model more robust during training.

4.6.3.1 Add

After the outputs from the Attention or Feed Forward layers are computed, they are added to their respective inputs.

- The Attention layer, the output of the Attention mechanism is added to its original input (residual connection).
- The output of the Feed Forward layer is added to its original input.

This use of residual connections helps mitigate the vanishing gradient problem during deep training, allowing gradients to flow through deeper layers without diminishing too much.

4.6.3.2 Norm:

Layer Normalization is applied to normalize the output of the Attention or Feed Forward layers.
The normalization step helps improve the stability of the training process and make the optimization more efficient, reducing the dependency on the distribution of the input data.

5 Implementation

5.1 How Transformers Are Applied to the Machine Translation Problem

- **Model Architecture and Setup**
  - **Encoder**: Processes the input sentence (e.g., in German) and converts it into a sequence of continuous vector representations.
  - **Decoder**: Generates the output sentence (e.g., in English) using the encoder's representations and previously generated words.
  - **Attention Mechanism**: The self-attention mechanism allows the model to focus on relevant parts of the input sentence while generating the output.
- **Training Data Preparation**
  - A *parallel corpus* is used, consisting of sentence pairs where each German sentence has a corresponding English translation.
  - The model learns to map sentences from one language to another by capturing patterns in structure, syntax, and vocabulary from the training data.
- **Training Process**
  - The model is trained using *supervised learning*, aiming to minimize the *cross-entropy loss* between its predicted outputs and the target sentences.
  - During training, the model's parameters (weights) are updated iteratively to improve translation accuracy.

5.2 Dataset

General information:
The data set we used to implement the English-German translation task is taken from WMT 2014: wmt14. For detail, we have chosen the *Europarl v7*, *Common Crawl corpus*, and *News Commentry* options in the section 'Download'>>'parallel-data'.

Dataset information:

- ≈ 4.5 millions sentence pairs.
- Sentence were encoded using byte-pair encoding.
- Shared source-target vocabulary file is included.
- The vocabulary size is 36709.
- Sentence pairs were batched together by approximate sequence length. Each training batch contained a set of sentence pairs containing approximately 25000 source tokens and 25000 target tokens.

5.3 Base Model

To understand the basic workings of Transformer, we've implemented a base model incorporating key components:

- **Scaled Dot-Product Attention:** Calculates attention scores for the input sequences.

```
class ScaledDotProductAttention(Layer):
  def __init__(self, d_model, num_heads):
    super(ScaledDotProductAttention, self).__init__()
    self.scaling_factor = tf.sqrt(tf.cast(d_model, tf.float32)/num_heads)

  def call(self, Q, K, V, mask = None):
    output = tf.matmul(Q,K,transpose_b = True) # Q*K^T
    output = output/self.scaling_factor
    if mask is not None:
      output += (mask * -1e9)

    output = tf.nn.softmax(output, axis = -1) #softmax
    output = tf.matmul(output,V) #matmul
    return output
```

Listing 1: Scaled Dot-Product Attention

- **Multi-Head Attention:** Executes multiple attention mechanisms by dividing the model dimension by the number of heads.
- **Masked Multi-Head Attention:** Ensures that future tokens are not visible during decoding by applying a causal matrix.

```
class MultiHeadAttention(Layer):
  def __init__(self, d_model, num_heads):
    super(MultiHeadAttention, self).__init__()

    self.num_heads = num_heads
    self.d_model = d_model
    self.attention = ScaledDotProductAttention(d_model, num_heads)

    self.W_Q = Dense(d_model)
    self.W_K = Dense(d_model)
    self.W_V = Dense(d_model)
    self.W_O = Dense(d_model)

  def call(self, Q,K,V, mask = None):
    Q = self.W_Q(Q)
    K = self.W_K(K)
    V = self.W_V(V)

    attention_output = self.attention(Q,K,V,mask)
    output = self.W_O(attention_output)
    return output
```

Listing 2: Multihead Attention

- **Feed-Forward Networks:** Applies position-independent transformations to each token.
- **Encoder and Decoder:** The encoder processes the input, while the decoder generates the output, calling attention and feed-forward layers within the class.

```python
class FeedForward(Layer):
  def __init__(self, d_model, d_ff):
    super(FeedForward, self).__init__()
    self.dense1 = Dense(d_ff, activation='relu')
    self.dense2 = Dense(d_model)

  def call(self,x):
    x = self.dense1(x)
    x = self.dense2(x)
    return x
```

Listing 3: Feed Forward Layer

```python
    class EncoderLayer(Layer):
  def __init__(self, d_model, num_heads, d_ff):
    super(EncoderLayer, self).__init__()
    self.multihead = MultiHeadAttention(d_model, num_heads)
    self.feedforward = FeedForward(d_model, d_ff)

    self.layernorm1 = LayerNormalization()
    self.layernorm2 = LayerNormalization()

  def call(self, x, mask = None):
    attn_output = self.multihead(x,x,x,mask)
    x = self.layernorm1(x)

    ff_output = self.feedforward(x)
    x = self.layernorm2(x)
    return x
```

Listing 4: Encoder Layer

```python
  class DecoderLayer(Layer):
  def __init__(self, d_model, num_heads, d_ff):
    super(DecoderLayer, self).__init__()

    self.mha1 = MultiHeadAttention(d_model, num_heads)
    self.mha2 = MultiHeadAttention(d_model, num_heads)
    self.ff = FeedForward(d_model, d_ff)

    self.layernorm1 = LayerNormalization()
    self.layernorm2 = LayerNormalization()
    self.layernorm3 = LayerNormalization()

  def call(self, x, encoder_output, ahead_mask = None, padding_mask = None):
    attention_output1 = self.mha1(x, x, x, ahead_mask)
    x = self.layernorm1(x)

    attention_output2 = self.mha2(x, encoder_output, encoder_output,
        padding_mask)
    x = self.layernorm2(x)

    ff_output = self.ff(x)
    x = self.layernorm3(x)
    return x
```

Listing 5: Decoder Layer

The 'base model' code is included in the GitHub repository and in Google Colab here: Transformer Base Model Using the TensorFlow library, we implement basic components such as scaled dot-product attention, multi-head attention, and feed-forward networks, and stack them into the encoder and decoder layers to understand how the Transformer works.

5.4 Train model & Loss & Output

5.4.1 Training model

The transformer model will be trained according to the following steps:

Step 1: `Process vocabulary file:`
From the vocabulary file included in the dataset, we add special tokens: `<UNK>`, `<StartOfSequence>`,`<EndOfSequence>`. Then create a Dictionary mapping for these vocabulary.

Step 2: `preprocess the dataset:`

- **Training data:** The dataset are zipped together to form correct pairs of input and labels. Then this dataset will be shuffled to ensure randomness, preprocessed with a mapping function, and batched into groups of 128 sentences for efficient training. Prefetching is also added to optimize data pipeline performance.
- **Validation data:** Sentences are tokenized, padded to a fixed length, and batch processed. English inputs serve as encoder input, while German inputs act as decoder input and one-hot-encoded labels.
- **Test data:** test data file `newstest2014` will be used

Step 3: `Define the Transformer model:`
The model will be defined with configuration:

- Vector's dimension: 512
- Encoder/Decoder stacks: 6
- Attention heads: 8
- Dropout rate: 0.1
- Feed-forward network size: 2048

Step 4: `Training`
Running the fitting process base on the loss function of categorical cross-entropy with label smoothing and masking.

5.4.2 Loss function

For each batch, we calculate the Categorical Cross-Entropy loss:

$$L_{batch} = -\sum_{i=1}^{\text{batch size}} \sum_{t=1}^{\text{sequence length}} m_i(t) \cdot \sum_{j=2}^{\text{vocab size}} \tilde{y}_{ijt} \log(\hat{y}_{ijt})$$

where:

- $\tilde{y}_{ijt}$ and $\hat{y}_{ijt}$ are the smoothed label and predicted probability for batch i, time step t, and vocabulary (class) j. We use **Label smoothing** (with a value of 0.1) to reduce the confidence in the correct class by distributing some of the probability mass to other classes.

- $m_i(t)$: mask layer, added to ensure the padding tokens (zeros in the feature dimension) do not contribute to the loss calculation. This variable get value 1 for valid tokens, value 0 for padding tokens.

5.4.3 Output

We have trained the model through 2 epochs: `epoch0` and `epoch1`. Below is the losses of training and validation:

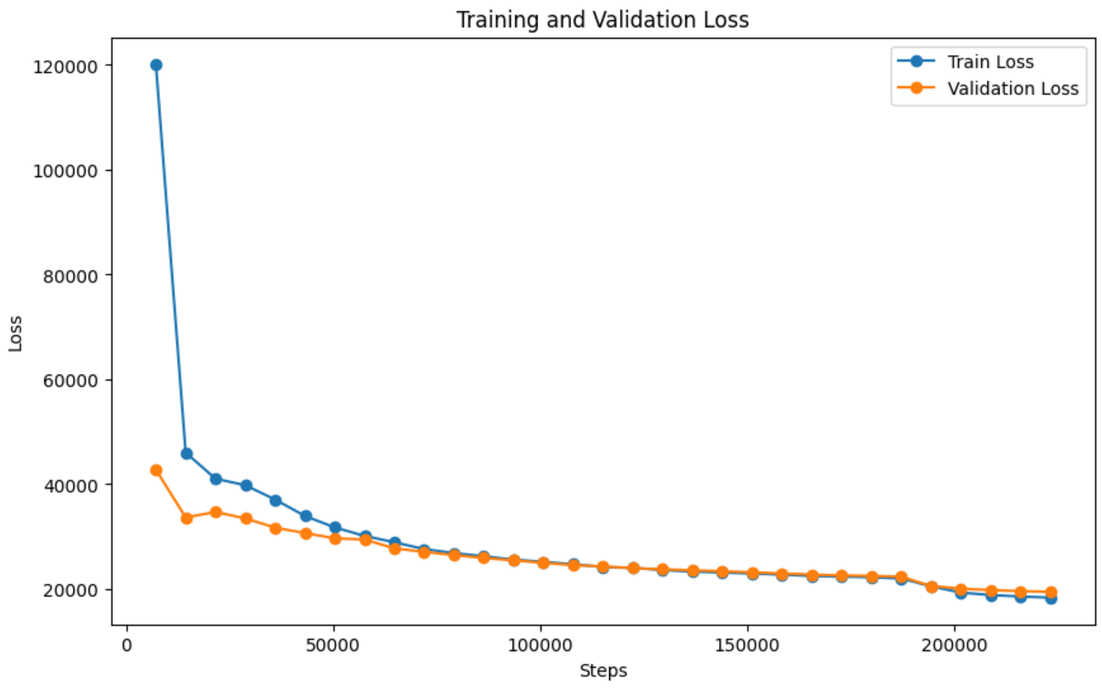

Figure 10: *Losses received in the training process*

Through the training, the model indeed learn and predict better. Since we implement a simple training process, there are still some aspects can be enhanced:

- We can use beam search for prediction as used in the paper [**?**] instead of using maximum probability.
- The batch size is also an important attribute to be considered. This is because a batch size determines the number of examples processed per step, and their effect on parameter update. Due to the storage limitation, we use a maximum batch size of 128. For the lower batch size, the model performed poorer. Hence, higher batch size seems to better.

6 Discussion

6.1 Comparison with Statistical Machine Translation (SMT)[2]

To compare the Transformer with Statistical Machine Translation (SMT) models: These two approach use different architecture. While SMT uses statistical methods as in section 3.1, the transformer is a deep neural network using self-attention mechanisms (section 4).

- **Handling Long-Term Dependencies**
  - **SMT:** SMT struggles with long-range dependencies because it models translations based on phrases and word alignments, often making it difficult to maintain coherence over long sentences. SMT typically relies on local context, so when translation involves distant words, SMT may fail to capture the correct meaning.
  - **Transformer:** The Transformers are good at handling long-term dependencies due to its self-attention mechanism. This allows it to weigh the relevance of all words in the input sequence, regardless of their distance. It doesn't process words in isolation but takes the entire sequence into account, leading to better performance on complex and long sentences.
- **Translation quality and Data quantity**
  - **SMT**: Can operate effectively with smaller datasets by utilizing statistical probabilities and linguistic rules. While suitable for low-resource contexts, SMT generally delivers less fluent and contextually nuanced translations compared to transformer when large data sets are available.
  - **Transformers**: Requires large datasets to achieve high translation quality, as they learn complex language patterns and context from extensive data. While they can produce more natural and context-aware translations than SMT, performance can diminish in low-resource scenarios unless advanced techniques like transfer learning are used.
- **Error Handling and Robustness**
  - **Error handling:** errors in SMT are more predictable due to reliance on fixed statistical models and rules. Meanwhile, transformers in NMT exhibit fewer repetitive errors and are more adaptable, benefiting from retraining and dynamic learning.
  - **Robustness:** SMT is generally robust in low-resource settings but struggles with idiomatic and complex language. Meanwhile, transformers are stronger at handling context and variation but can be sensitive to domain shifts and noisy data.

Ultimately, transformers is preferable for high-quality translation when sufficient data and computational resources are available. In contrast, SMT remains practical for scenarios with limited data or specific linguistic challenges.

6.2 Other modern approaches

After the advent of Transformer model, machine translation has seen several innovations and improvements. Base on the transformer, new approaches are developed with many advancements.

1. **Pre-trained Language Models:**[3]
One of the most impactful developments has been the use of large pre-trained language models. Models like BERT, GPT, and their variants are pre-trained on vast amounts of text data to learn general language understanding. These models are then fine-tuned for translation tasks, allowing them to leverage their broad language knowledge for more accurate translations.

2. **Develop in multilingual capability and resource utilization:**[5]
- Prime example: Massively Multilingual Models (M2M-100).
- Instead of training separate models for each language pair, researchers have developed multilingual models that can translate between multiple languages. These models, like mBART and M2M-100, are trained on data from many languages simultaneously, allowing them to share knowledge across languages and even perform zero-shot translation (translating between language pairs they weren't explicitly trained on).

3. **Document-level Translation:**
While sentence-level translation has been the norm, there's increasing focus on document-level translation. This approach considers the broader context of a document, helping to maintain consistency in things like pronoun references, tense, and style across multiple sentences.

These approaches often overlap and can be combined in various ways. The field continues to evolve rapidly, with new techniques and refinements constantly emerging. The overarching trend is towards more flexible, efficient, and context-aware systems that can handle a wider range of translation scenarios with increasing accuracy.

References

[1] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

[2] G. Datta, N. Joshi, and K. Gupta. Performance comparison of statistical vs. neural-based translation system on low-resource languages. *International Journal on Smart Sensing and Intelligent Systems*, 16(1):–, 2023.

[3] L. Han, G. Erofeev, I. Sorokina, S. Gladkoff, and G. Nenadic. Examining large pre-trained language models for machine translation: What you don't know about it, 2022.

[4] D. D. Hung. *Learn about Transformer Architecture.* https://viblo.asia/p/tim-hieu-ve-kien-truc-transformer-Az45byM6lxY. Accessed: 2025-05-01.

[5] A. Mohammadshahi, V. Nikoulina, A. Berard, C. Brun, J. Henderson, and L. Besacier. SMaLL-100: Introducing shallow multilingual machine translation model for low-resource languages. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8348–8359. Association for Computational Linguistics, Dec. 2022.

[6] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.