Artificial Intelligence Final Report Assignment

Report Answer Sheet

Group Leader

Student ID): 18520952

Name: Ngo Le Hieu Kien

Group Members

Student ID: 18521064

Name: Ngo Dinh Luan

Student ID: 18521029

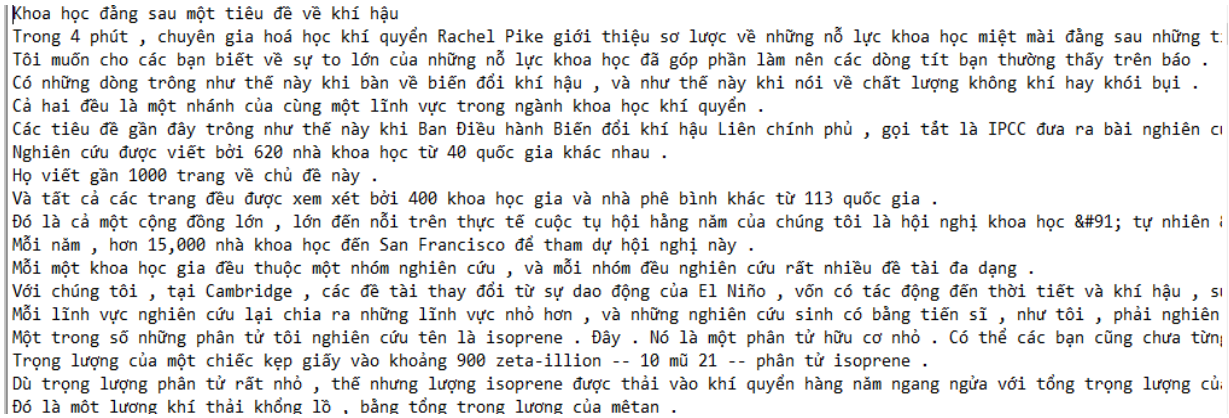
Name: Le Hoang Long

Student ID: 18521024

Name: Le Hoang Long

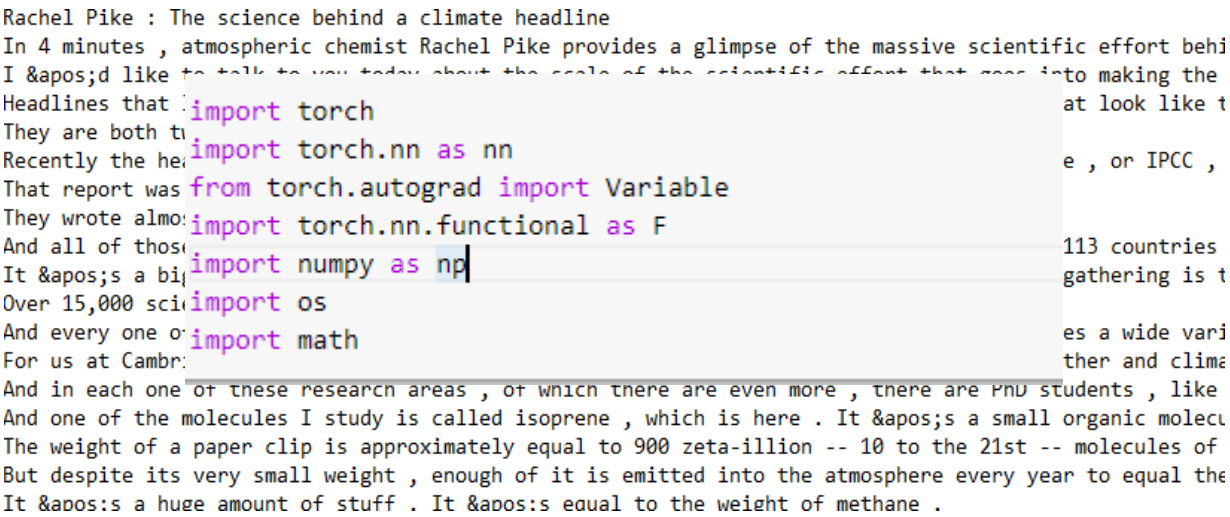
Problem 1

1. Introduce datasets



Khoa học đằng sau một tiêu đề về khí hậu
Trong 4 phút , chuyên gia hoá học khí quyển Rachel Pike giới thiệu sơ lược về những nỗ lực khoa học miệt mài đằng sau những t
Tôi muốn cho các bạn biết về sự to lớn của những nỗ lực khoa học đã góp phần làm nên các dòng tít bạn thường thấy trên báo .
Có những dòng trông như thế này khi bàn về biến đổi khí hậu , và như thế này khi nói về chất lượng không khí hay khói bụi .
Cả hai đều là một nhánh của cùng một lĩnh vực trong ngành khoa học khí quyển .
Các tiêu đề gần đây trông như thế này khi Ban Điều hành Biến đổi khí hậu Liên chính phủ , gọi tắt là IPCC đưa ra bài nghiên c
Nghiên cứu được viết bởi 620 nhà khoa học từ 40 quốc gia khác nhau .
Họ viết gần 1000 trang về chủ đề này .
Và tất cả các trang đều được xem xét bởi 400 khoa học gia và nhà phê bình khác từ 113 quốc gia .
Đó là cả một cộng đồng lớn , lớn đến nỗi trên thực tế cuộc tụ hội hằng năm của chúng tôi là hội nghị khoa học [tự nhiên
Mỗi năm , hơn 15,000 nhà khoa học đến San Francisco để tham dự hội nghị này .
Mỗi một khoa học gia đều thuộc một nhóm nghiên cứu , và mỗi nhóm đều nghiên cứu rất nhiều đề tài đa dạng .
Với chúng tôi , tại Cambridge , các đề tài thay đổi từ sự dao động của El Niño , vốn có tác động đến thời tiết và khí hậu , s
Mỗi lĩnh vực nghiên cứu lại chia ra những lĩnh vực nhỏ hơn , và những nghiên cứu sinh có bằng tiến sĩ , như tôi , phải nghiên
Một trong số những phân tử tôi nghiên cứu tên là isoprene . Đây . Nó là một phân tử hữu cơ nhỏ . Có thể các bạn cũng chưa từn
Trọng lượng của một chiếc kẹp giấy vào khoảng 900 zeta-illion -- 10 mũ 21 -- phân tử isoprene .
Dù trọng lượng phân tử rất nhỏ , thế nhưng lượng isoprene được thải vào khí quyển hàng năm ngang ngửa với tổng trọng lượng củ
Đó là một lượng khí thải khổng lồ , bằng tổng trọng lượng của mêtan .

+ Train_en sets :



Rachel Pike : The science behind a climate headline
In 4 minutes , atmospheric chemist Rachel Pike provides a glimpse of the massive scientific effort behi
I 'd like to talk to you today about the scale of the scientific effort that goes into making the
Headlines that
They are both t
Recently the he
That report was
They wrote almo
And all of thos
It 's a bi
Over 15,000 sci
And every one o
For us at Cambr
And in each one of these research areas , of which there are even more , there are PhD students , like
And one of the molecules I study is called isoprene , which is here . It 's a small organic molecu
The weight of a paper clip is approximately equal to 900 zeta-illion -- 10 to the 21st -- molecules of
But despite its very small weight , enough of it is emitted into the atmosphere every year to equal the
It 's a huge amount of stuff . It 's equal to the weight of methane .

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F
import numpy as np
import os
import math
```

+ train_vn sets :

Libraries need to be prepared :

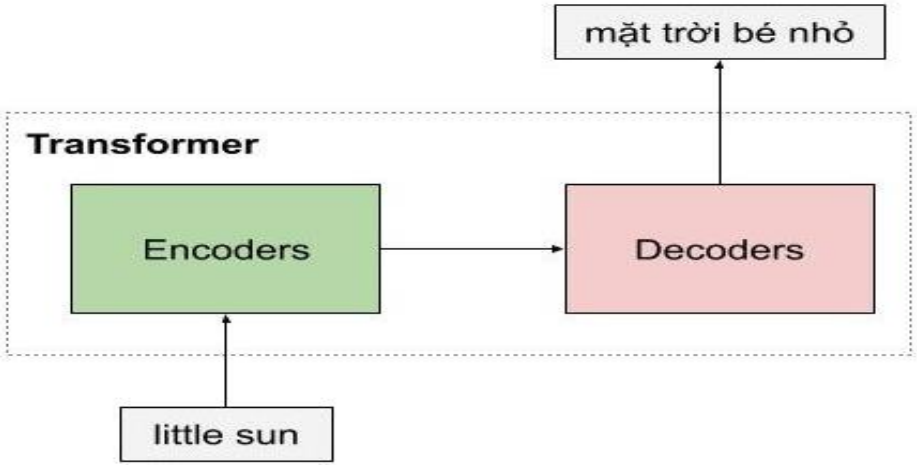
2. Transformer

a. Introduce :

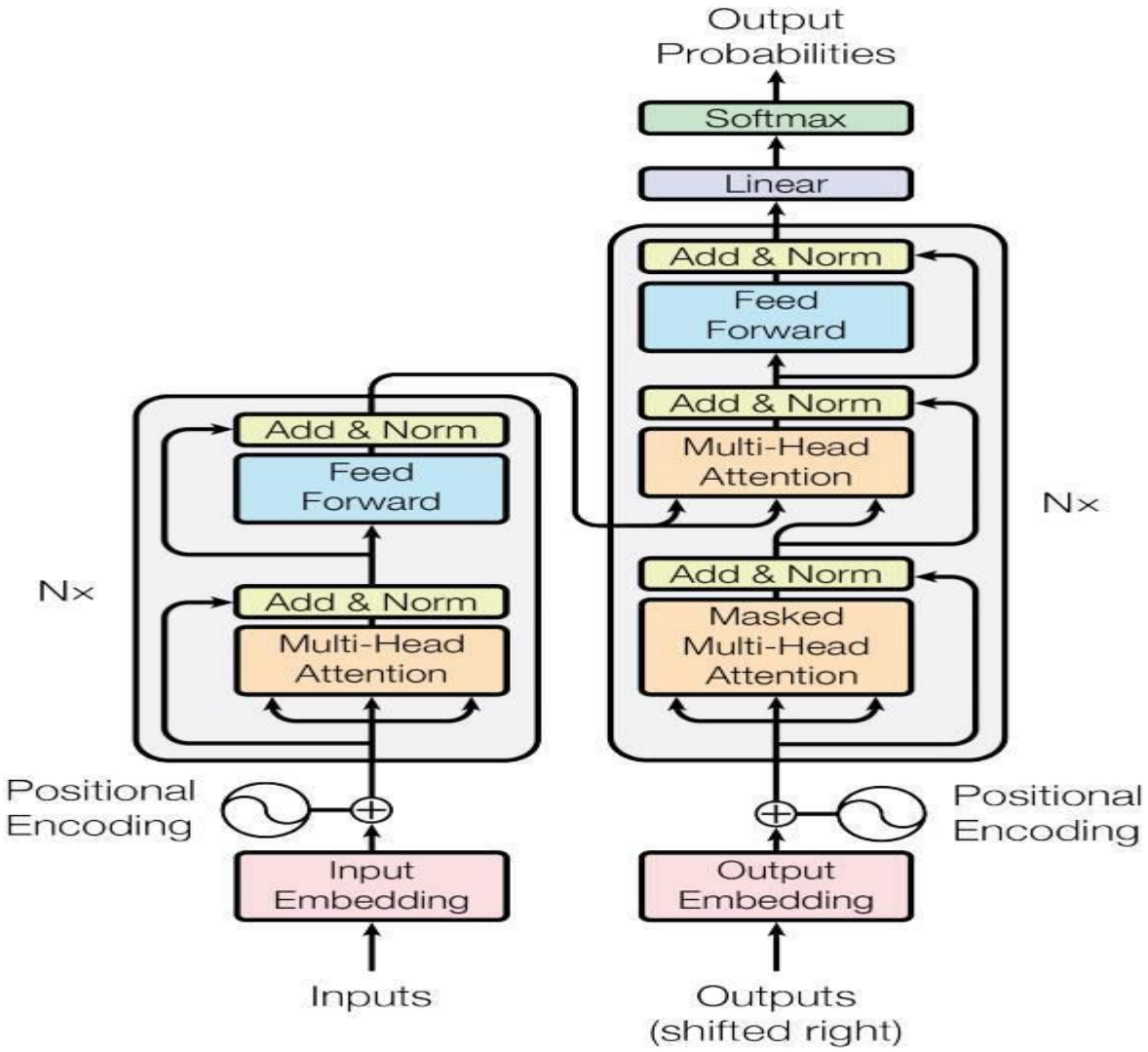
Transformer models popularity is undeniable, because it is the base, the core to develop all other famous model like BERT (Bidirectional Encoder Representations from Transformers) - A model use to learn how to present the best vocabulary and it had been a big step for the NLP community in the year of 2019.

Google is also use BERT model for their searching engine. To start understand more about BERT, you have to know about the Transformer model.

Model overview :



b. Detailed model .

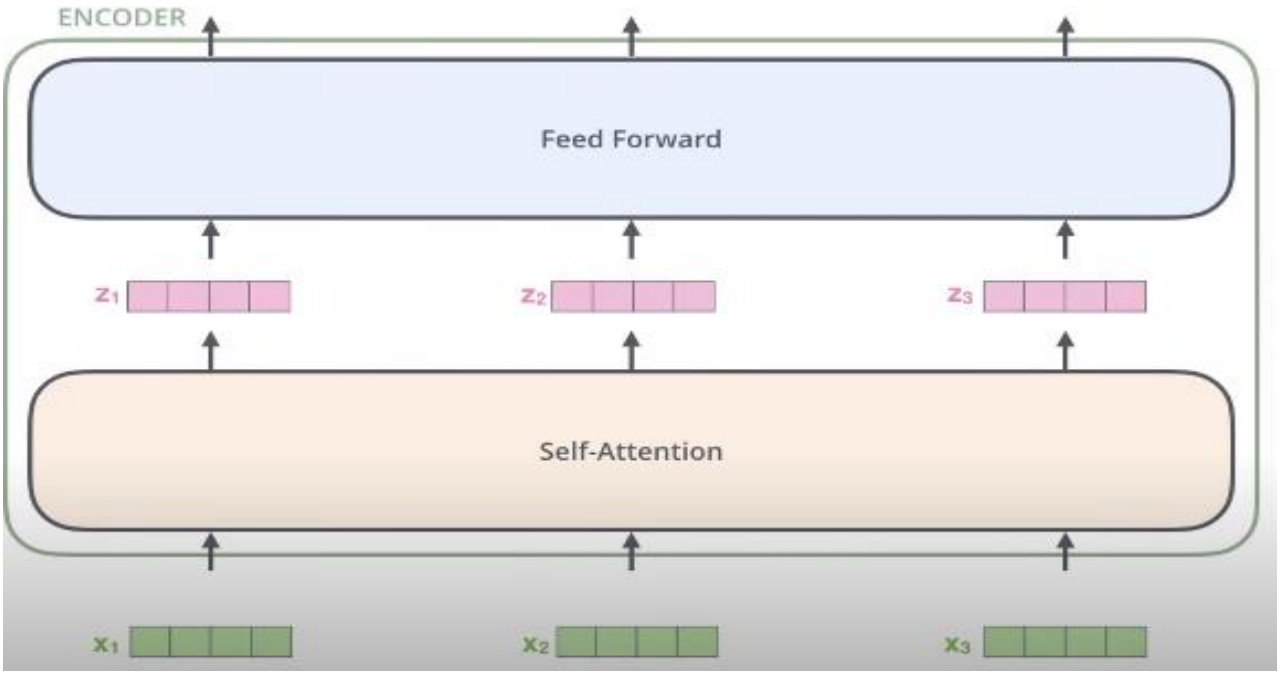


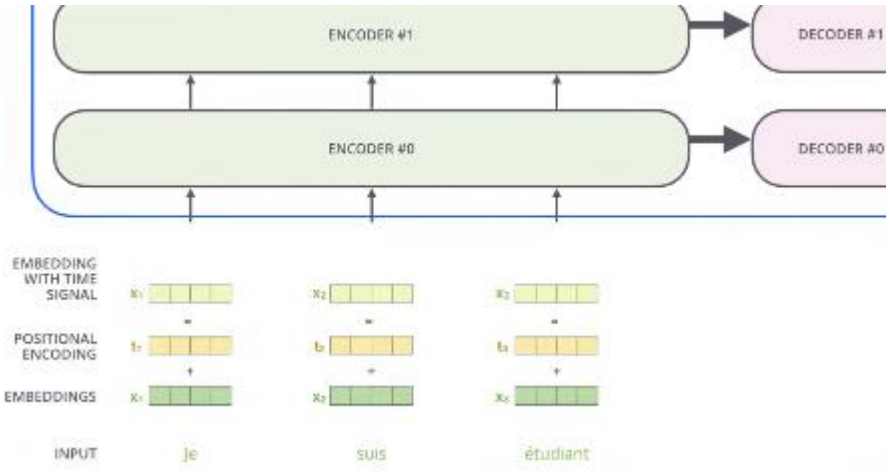
2.1 ENCODER

Transformer models encoder can contain a lot of similar encoder layer, each encoder is created by 2 elements which is multi-head attention and feedforward network, as well as skip connection and normalization layer. of these 2 element,you will be more interested in multi-head attention because it is a new layer which is being introduce in this report, and its also help making the different between Transformers model and LTSM model.

+ Consists of 2 main blocks:

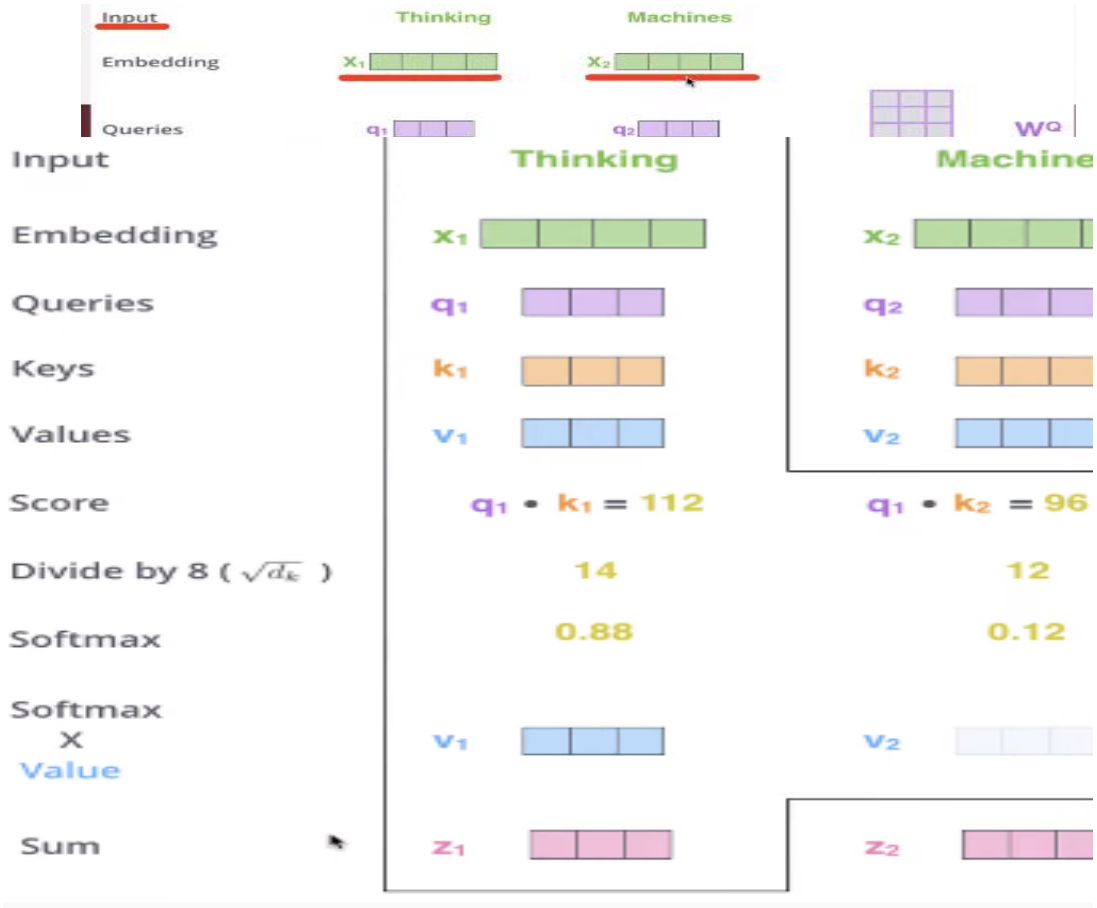
Self-Attention and Feedforward Neural Network (the most important class is Self-Attention)





+ Before the input enters the Self-Attention class, the embedding will go through the Positional Encoding class to ensure word order in the sentence

+ Self-Attention class




```
def attention(q, k, v, mask=None, dropout=None):
    """
    q: batch_size x head x seq_length x d_model
    k: batch_size x head x seq_length x d_model
    v: batch_size x head x seq_length x d_model
    mask: batch_size x 1 x 1 x seq_length
    output: batch_size x head x seq_length x d_model
    """

    # attention score được tính bằng cách nhân q với k
    d_k = q.size(-1)
    scores = torch.matmul(q, k.transpose(-2, -1))/math.sqrt(d_k)

    if mask is not None:
        mask = mask.unsqueeze(1)
        scores = scores.masked_fill(mask==0, -1e9)
    # xong rồi thì chuẩn hóa bằng softmax
    scores = F.softmax(scores, dim=-1)

    if dropout is not None:
        scores = dropout(scores)

    output = torch.matmul(scores, v)
    return output, scores
```

ıring the

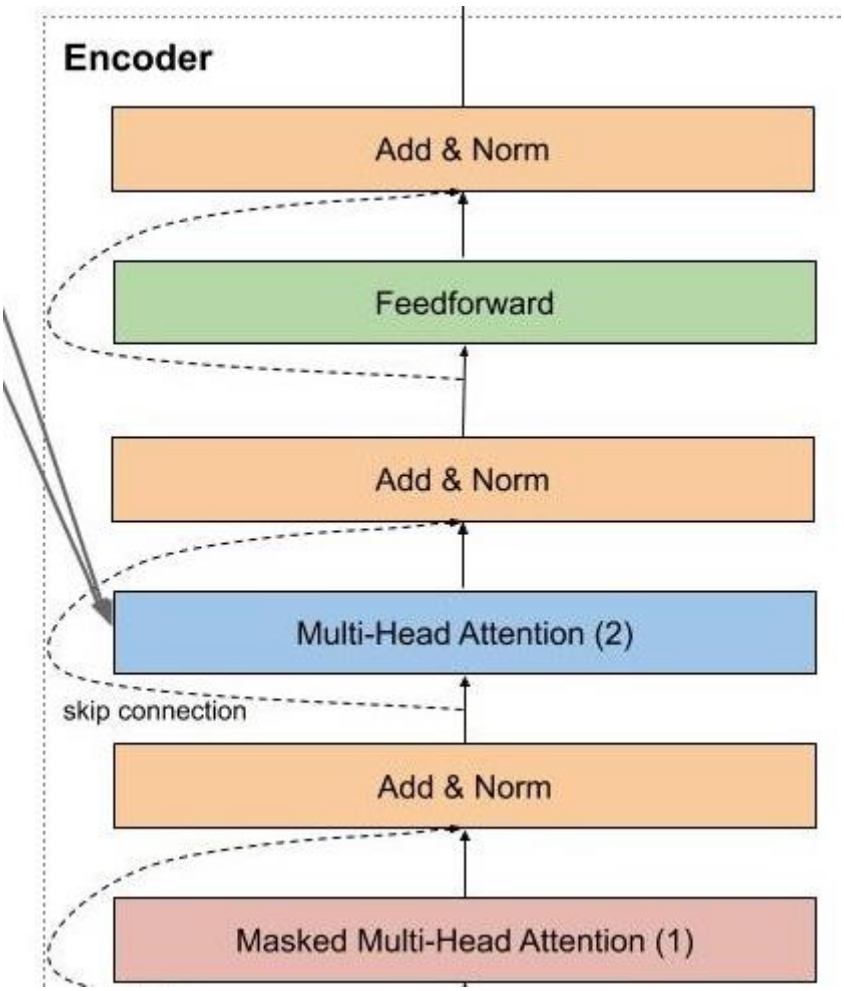
V

+

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, heads, d_model, dropout=0.1):
        super().__init__()
        assert d_model % heads == 0

        self.d_model = d_model
        self.d_k = d_model//heads
        self.h = heads
        self.attn = None

        # tạo ra 3 ma trận trọng số là q_linear, k_linear, v_linear như hình trên
        self.q_linear = nn.Linear(d_model, d_model)
        self.k_linear = nn.Linear(d_model, d_model)
        self.v_linear = nn.Linear(d_model, d_model)

        self.dropout = nn.Dropout(dropout)
        self.out = nn.Linear(d_model, d_model)

    def forward(self, q, k, v, mask=None):
        """
        q: batch_size x seq_length x d_model
        k: batch_size x seq_length x d_model
        v: batch_size x seq_length x d_model
        mask: batch_size x 1 x seq_length
        output: batch_size x seq_length x d_model
        """
        bs = q.size(0)
        # nhân ma trận trọng số q_linear, k_linear, v_linear với dữ liệu đầu vào q, k, v
        # ở bước encode các bạn lưu ý rằng q, k, v chỉ là một (xem hình trên)
        q = self.q_linear(q).view(bs, -1, self.h, self.d_k)
        k = self.k_linear(k).view(bs, -1, self.h, self.d_k)
        v = self.v_linear(v).view(bs, -1, self.h, self.d_k)

        q = q.transpose(1, 2)
        k = k.transpose(1, 2)
        v = v.transpose(1, 2)

        # tính attention score
        scores, self.attn = attention(q, k, v, mask, self.dropout)
```

2.2 DECODER

Decoder performs the function of decoding the source sentence vector into the target sentence, so the decoder will receive information from the encoder as 2 vector key and value. The decoder's architecture is very similar to that of the encoder, except that there is an extra multi head attention in the middle used to learn the relationship between the word being translated and the words in the source sentence.



+Masked Multi Head Attention Layer:

Masked Multi Head Attention is of course the multi head attention that we talked about above, which has the function to encode target sentence words during translation, however, when installing we need to note that we have to hide future words are not yet translated by the model, to do this we simply multiply by a vector containing the values 0.1.

In the decoder there is also another multi head attention function that notices the words in the encoder model, this layer receives the key and value vectors from the encoder model, and outputs from the layer below. Simply because we want to compare the correlation between the word being translated from source.

+ Construction :

```
class DecoderLayer(nn.Module):
    def __init__(self, d_model, heads, dropout=0.1):
        super().__init__()
        self.norm_1 = Norm(d_model)
        self.norm_2 = Norm(d_model)
        self.norm_3 = Norm(d_model)

        self.dropout_1 = nn.Dropout(dropout)
        self.dropout_2 = nn.Dropout(dropout)
        self.dropout_3 = nn.Dropout(dropout)

        self.attn_1 = MultiHeadAttention(heads, d_model, dropout=dropout)
        self.attn_2 = MultiHeadAttention(heads, d_model, dropout=dropout)
        self.ff = FeedForward(d_model, dropout=dropout)

    def forward(self, x, e_outputs, src_mask, trg_mask):
        """
        x: batch_size x seq_length x d_model
        e_outputs: batch_size x seq_length x d_model
        src_mask: batch_size x 1 x seq_length
        trg_mask: batch_size x 1 x seq_length
        """
        # Các bạn xem hình trên, kiến trúc mình vẽ với code ở chỗ này tương đương nhau.
        x2 = self.norm_1(x)
        # multihead attention thứ nhất, chú ý các từ ở target
        x = x + self.dropout_1(self.attn_1(x2, x2, x2, trg_mask))
        x2 = self.norm_2(x)
        # masked mulithead attention thứ 2. k, v là giá trị output của mô hình encoder
        x = x + self.dropout_2(self.attn_2(x2, e_outputs, e_outputs, src_mask))
        x2 = self.norm_3(x)
        x = x + self.dropout_3(self.ff(x2))
        return x
```


```
class DecoderLayer(nn.Module):
    def __init__(self, d_model, heads, dropout=0.1):
        super().__init__()
        self.norm_1 = Norm(d_model)
        self.norm_2 = Norm(d_model)
        self.norm_3 = Norm(d_model)

        self.dropout_1 = nn.Dropout(dropout)
        self.dropout_2 = nn.Dropout(dropout)
        self.dropout_3 = nn.Dropout(dropout)

        self.attn_1 = MultiHeadAttention(heads, d_model, dropout=dropout)
        self.attn_2 = MultiHeadAttention(heads, d_model, dropout=dropout)
        self.ff = FeedForward(d_model, dropout=dropout)

    def forward(self, x, e_outputs, src_mask, trg_mask):
        """
        x: batch_size x seq_length x d_model
        e_outputs: batch_size x seq_length x d_model
        src_mask: batch_size x 1 x seq_length
        trg_mask: batch_size x 1 x seq_length
        """
        # Các bạn xem hình trên, kiến trúc mình vẽ với code ở chỗ này tương đương nhau.
        x2 = self.norm_1(x)
        # multihead attention thứ nhất, chú ý các từ ở target
        x = x + self.dropout_1(self.attn_1(x2, x2, x2, trg_mask))
        x2 = self.norm_2(x)
        # masked mulithead attention thứ 2. k, v là giá trị output của mô hình encoder
        x = x + self.dropout_2(self.attn_2(x2, e_outputs, e_outputs, src_mask))
        x2 = self.norm_3(x)
        x = x + self.dropout_3(self.ff(x2))
        return x
```

2.3 Install transformer

```
class Transformer(nn.Module):
    """ Cuối cùng ghép chúng lại với nhau để được mô hình transformer hoàn chỉnh
    """
    def __init__(self, src_vocab, trg_vocab, d_model, N, heads, dropout):
        super().__init__()
        self.encoder = Encoder(src_vocab, d_model, N, heads, dropout)
        self.decoder = Decoder(trg_vocab, d_model, N, heads, dropout)
        self.out = nn.Linear(d_model, trg_vocab)
    def forward(self, src, trg, src_mask, trg_mask):
        """
        src: batch_size x seq_length
        trg: batch_size x seq_length
        src_mask: batch_size x 1 x seq_length
        trg_mask batch_size x 1 x seq_length
        output: batch_size x seq_length x vocab_size
        """
        e_outputs = self.encoder(src, src_mask)

        d_output = self.decoder(trg, e_outputs, src_mask, trg_mask)
        output = self.out(d_output)
        return output
```

3. Data processing

```
import spacy
import re

class tokenize(object):

    def __init__(self, lang):
        self.nlp = spacy.load(lang)

    def tokenizer(self, sentence):
        sentence = re.sub(
        r"[\*\"“”\n\\…\+\-\/\=\(\)‘•:\[\]\|’\!;]", " ", str(sentence))
        sentence = re.sub(r"[ ]+", " ", sentence)
        sentence = re.sub(r"\!+", "!", sentence)
        sentence = re.sub(r"\,+", ",", sentence)
        sentence = re.sub(r"\?+", "?", sentence)
        sentence = sentence.lower()
        return [tok.text for tok in self.nlp.tokenizer(sentence) if tok.text != " "]
```

```
def create_dataset(src_data, trg_data, max_strlen, batchsize, device, SRC, TRG, istr

    print("creating dataset and iterator... ")

    raw_data = {'src' : [line for line in src_data], 'trg': [line for line in trg_da
    df = pd.DataFrame(raw_data, columns=["src", "trg"])

    mask = (df['src'].str.count(' ') < max_strlen) & (df['trg'].str.count(' ') < max
    df = df.loc[mask]

    df.to_csv("translate_transformer_temp.csv", index=False)

    data_fields = [('src', SRC), ('trg', TRG)]
    train = data.TabularDataset('./translate_transformer_temp.csv', format='csv', fi

    train_iter = MyIterator(train, batch_size=batchsize, device=device,
                            repeat=False, sort_key=lambda x: (len(x.src), len(x.trg)),
                            batch_size_fn=batch_size_fn, train=istrain, shuffle=True)

    os.remove('translate_transformer_temp.csv')

    if istrain:
        SRC.build_vocab(train)
        TRG.build_vocab(train)

    return train_iter
```

4. Evaluate the model based on the BLEU measure.

BLEU stands for Bilingual Evaluation Understudy, is a method of evaluating a translation based on reference translations. The prerequisite to be able to use BLEU is that you must have one (or more) sample sentences. For the machine translation problem, the sample sentence is the output sentence of the pair of sentences in the data set. BLEU evaluates a sentence by matching it with sample sentences and gives it a scale from 0 (absolute deviation) to 1 (absolute match).

BLEU is known as a method that is simple, easy to understand, low computational cost and similar to human assessment. However, the human factor in sample sentence generation makes BLEU not completely objective.

+The calculation of BLEU is to count the number of n-gram matches between the sample sentence (R) and the evaluated sentence (C) and then divide it by the number of tokens of C. The choice of n depends on the language, task, and specific goal. body. The simplest we can use unigram is n-gram containing 1 token (n=1). Visually, the larger n, the smoother the sentence.

$$Count_{Clip} = min(Count, Max\_Ref\_Count)$$

The maximum number of matches is limited by the maximum number of occurrences of n-grams in the sample sentences.

```
epoch: 001 - iter: 01599 - train loss: 4.6193 - time: 0.2198
epoch: 001 - iter: 01799 - train loss: 4.5520 - time: 0.2008
epoch: 001 - iter: 01999 - train loss: 4.5005 - time: 0.1909
epoch: 001 - iter: 02199 - train loss: 4.3856 - time: 0.1975
epoch: 001 - iter: 02399 - train loss: 4.3173 - time: 0.1976
epoch: 001 - iter: 02497 - valid loss: 3.1581 - bleu score: 0.1112 - time: 165.0388
epoch: 002 - iter: 00199 - train loss: 4.2011 - time: 0.2077
epoch: 002 - iter: 00399 - train loss: 4.1463 - time: 0.1960
epoch: 002 - iter: 00599 - train loss: 4.0987 - time: 0.2116
epoch: 002 - iter: 00799 - train loss: 4.0687 - time: 0.1979
epoch: 002 - iter: 00999 - train loss: 3.9827 - time: 0.2102
epoch: 002 - iter: 01199 - train loss: 4.0057 - time: 0.2052
epoch: 002 - iter: 01399 - train loss: 3.9346 - time: 0.1953
epoch: 002 - iter: 01599 - train loss: 3.8950 - time: 0.1884
epoch: 002 - iter: 01799 - train loss: 3.8691 - time: 0.2070
epoch: 002 - iter: 01999 - train loss: 3.8540 - time: 0.1966
epoch: 002 - iter: 02199 - train loss: 3.8372 - time: 0.2066
epoch: 002 - iter: 02399 - train loss: 3.7525 - time: 0.2250
epoch: 002 - iter: 02497 - valid loss: 2.7808 - bleu score: 0.1791 - time: 167.8549
```