

# Machine Translation with Transformer

# 1. Input

Bộ dữ liệu IWSLT15-en-vi gồm khoảng 136k với:

- Train (133k dòng)
- Valid (1.27k dòng)
- Test (1.27k dòng)

Dataset Viewer

Auto-converted to Parquet API

Subset

Split

iwslt2015-en-vi (136k rows)

train (133k rows)

translation

translation

{ "en": "Rachel Pike : The science behind a climate headline", "vi": "Khoa học đằng sau một tiêu đề về khí hậu" }

{ "en": "In 4 minutes , atmospheric chemist Rachel Pike provides a glimpse of the massive scientific effort behind the bold headlines on climate change , with her team -- one of thousands who contributed -- taking a risky flight over the rainforest in pursuit of data..." }

{ "en": "I d like to talk to you today about the scale of the scientific effort that goes into making the headlines you see in the paper .", "vi": "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..." }

# 1. Input


- Vì ta đang làm bài toán dịch máy nên cần phải chia data thành các file chứa các dòng có câu tiếng Anh và tiếng Việt riêng biệt.

 validation.vi

---

 validation.en

---

 train.vi


---

 train.en

---

 test.vi

---

 test.en

## 2. Tokenization

Sử dụng thư viện [sentencepiece](#) để mã hóa câu đầu vào

```
sp_cfg = template.format(  
    train_file, # file txt gồm các câu ở mỗi dòng  
    cfg.pad_id, # index của token padding  
    cfg.sos_id, # index của token start of sentence  
    cfg.eos_id, # index của token end of sentence  
    cfg.unk_id, # index của token unknow  
    model_prefix, # tên của model tokenize sau khi train  
    cfg.sp_vocab_size, # kích thước bộ từ vựng (ở đây được đặt là 10k)  
    cfg.character_coverage, # tỉ lệ số lượng kí tự được mô hình sử dụng  
    cfg.model_type) # được đặt là "unigram" -> tokenize ở mức độ từ đơn  
  
spm.SentencePieceTrainer.Train(sp_cfg)
```

## 2. Tokenization

Ví dụ sau khi train model tokenize với file train.vi ta được 2 file sau:



sp\_vi.model

Mô hình tokenize sau khi được huấn luyện



sp\_vi

Bộ vocab

## 2. Tokenization

Ví dụ tokenize một câu trong file train.vi với sequence length là 150

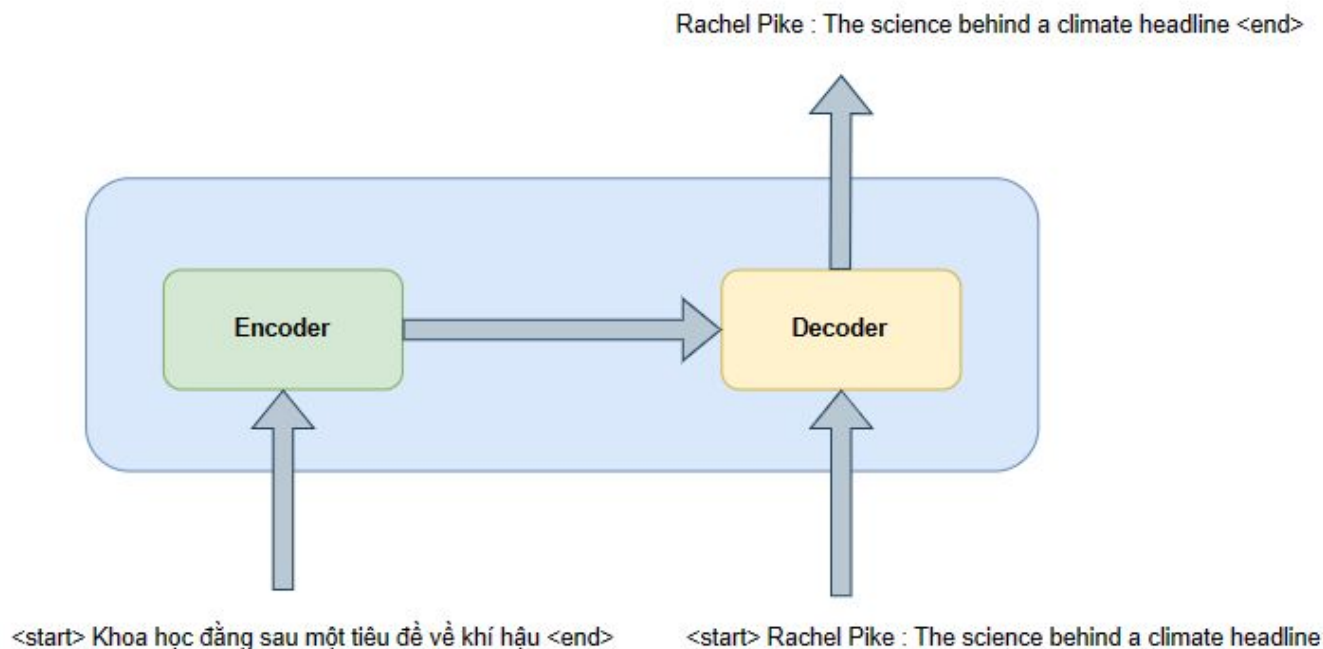
Khoa học đăng sau một tiêu đề về khí hậu

Kiểm tra thử trong  
file sp\_vi ta thấy:

<u>_Khoa</u> 1960	<u>_học</u> 74	<u>_đ</u> 128	<u>ăng</u> 974	<u>_sau</u> 146	<u>_một</u> 9	<u>_tiêu</u> 418
<u>_tiêu</u> 127	<u>_về</u> 41	<u>_khí</u> 404	<u>_hậu</u> 817			

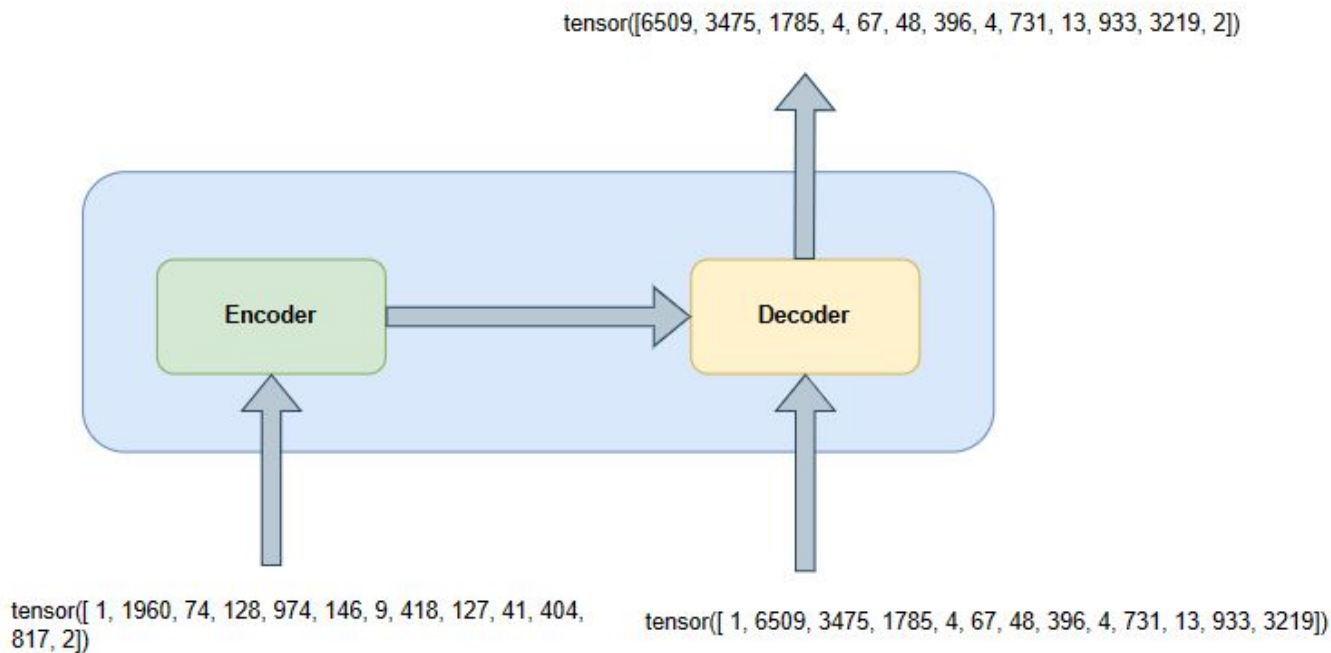
## 2. Tokenization

Đối với quá trình training, ta phải thêm các token start <s>, end </s> vào trong câu như sau



## 2. Tokenization

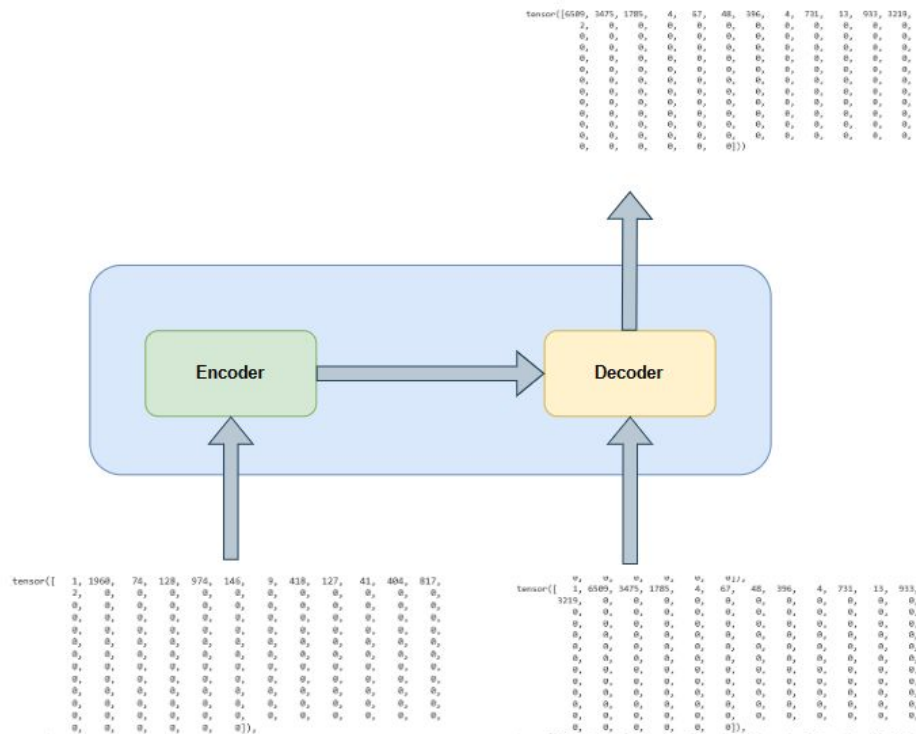
Đối với quá trình training, ta phải thêm các token start <s>, end </s> vào trong câu như sau





## 2. Tokenization

Nhưng, do chúng ta đã đặt sequence length bằng 150 nên phải thêm các token padding (có giá trị 0) vào sau



### 3. Training

Sau khi tokenize, chúng ta sẽ bắt đầu training mô hình transformer với các tham số được cài đặt trong config

```
# Model
num_heads = 8
num_layers = 6
d_model = 512
d_ff = 2048
drop_out = 0.1

# Training
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
learning_rate = 1e-4
batch_size = 64
seq_len = 150
```

### 3. Training

## Load data train

```
1 input_data = NMTDataset(cfg=cfg, data_type="train")
```

```
==> Load data from: ./transformer/data/train.vi
```

```
==> Load data from: ./transformer/data/train.en
```

```
100%|██████████| 133317/133317 [00:07<00:00, 17127.35it/s]
```

100% 133317/133317 [00:06<00:00, 19664.11it/s]

## Một mẫu trong data train

```
1 input_data[0]
```

[illegible]

### 3. Training

#### Khởi tạo mô hình, trọng số mô hình và tokenizer

```
class Trainer():
    def __init__(self, cfg, is_train=True, load_ckpt=True):
        self.cfg = cfg

        print("Loading Transformer model & Adam optimizer...")
        self.model = Transformer(self.cfg).to(self.cfg.device)

        self.optim = torch.optim.Adam(self.model.parameters(), lr=self.cfg.learning_rate)

        self.best_loss = 100.0
        if load_ckpt:
            print("Loading checkpoint...")
            checkpoint = torch.load(f"{self.cfg.ckpt_dir}/{self.cfg.ckpt_name}", map_location=self.cfg.device)
            self.model.load_state_dict(checkpoint['model_state_dict'])
            self.optim.load_state_dict(checkpoint['optim_state_dict'])
            self.best_loss = checkpoint['loss']
        else:
            print("Initializing the model...")
            for p in self.model.parameters():
                if p.dim() > 1:
                    nn.init.xavier_uniform_(p)

        # Prepare Tokenizer
        self.prepare_tokenizer()
```

### 3. Training

Khai báo hàm loss([negative log likelihood loss](#)) và load data train và valid theo từng batch với batch size=64

```
def get_data_loader(cfg, data_type='train'):
    dataset = NMTDataset(cfg, data_type)

    if data_type == 'train':
        shuffle = True
    else:
        shuffle = False

    dataloader = DataLoader(dataset, batch_size=cfg.batch_size, shuffle=shuffle)

    return dataset, dataloader
```

```
if is_train:
    # Load loss function
    print("Loading loss function...")
    self.criterion = nn.NLLLoss()

    # Load dataloaders
    print("Loading dataloaders...")
    self.train_dataset, self.train_loader = get_data_loader(self.cfg, 'train')
    self.valid_dataset, self.valid_loader = get_data_loader(self.cfg, 'validation')
```

# 3. Training

## Hàm train

```
def train(self):
    print("Training...")

    for epoch in range(1, self.cfg.num_epochs+1):
        print(f"##### Epoch: {epoch} #####")

        self.model.train() #
        train_losses = []
        start_time = datetime.datetime.now()

        bar = tqdm(enumerate(self.train_loader), total=len(self.train_loader), desc='TRAINING')

        for batch_idx, batch in bar:
            src_input, tgt_input, tgt_output = batch
            src_input, tgt_input, tgt_output = src_input.to(self.cfg.device), tgt_input.to(self.cfg.device), tgt_output.to(self.cfg.device)

            e_mask, d_mask = self.create_mask(src_input, tgt_input)

            logits = self.model(src_input, tgt_input, e_mask, d_mask)

            self.optim.zero_grad()

            loss = self.criterion(
                logits.view(-1, logits.shape[-1]),
                tgt_output.reshape(-1)
            )

            loss.backward()
            self.optim.step()

            train_losses.append(loss.item())
```

### 3. Training

Một mẫu trong 1 batch của train\_loader

```
1 print(src_input)
2 print(src_input.shape)
```

```
tensor([[ 1, 268, 31, ..., 0, 0, 0],
        [ 1, 3565, 92, ..., 0, 0, 0],
        [ 1, 84, 14, ..., 0, 0, 0],
        ...,
        [ 1, 360, 122, ..., 0, 0, 0],
        [ 1, 203, 6, ..., 0, 0, 0],
        [ 1, 6605, 3024, ..., 0, 0, 0]])
torch.Size([64, 150])
```

```
1 print(tgt_input)
2 print(tgt_input.shape)
```

```
tensor([[ 1, 48, 19, ..., 0, 0, 0],
        [ 1, 4, 4569, ..., 0, 0, 0],
        [ 1, 70, 10, ..., 0, 0, 0],
        ...,
        [ 1, 184, 20, ..., 0, 0, 0],
        [ 1, 26, 58, ..., 0, 0, 0],
        [ 1, 2064, 3094, ..., 0, 0, 0]])
```

```
1 print(tgt_output)
2 print(tgt_output.shape)
```

```
tensor([[ 48, 19, 10, ..., 0, 0, 0],
        [ 4, 4569, 4, ..., 0, 0, 0],
        [ 70, 10, 12, ..., 0, 0, 0],
        ...,
        [ 184, 20, 129, ..., 0, 0, 0],
        [ 26, 58, 17, ..., 0, 0, 0],
        [2064, 3094, 2108, ..., 0, 0, 0]])
torch.Size([64, 150])
```

### 3. Training

Tạo mask cho **encoder** và **decoder** để tránh attention vào các token padding

```
def create_mask(src_input, tgt_input):  
    e_mask = (src_input != cfg.pad_id).unsqueeze(1) # (B, 1, L)  
    d_mask = (tgt_input != cfg.pad_id).unsqueeze(1) # (B, 1, L)  
  
    nopeak_mask = torch.ones([1, cfg.seq_len, cfg.seq_len], dtype=torch.bool) # (1, L, L)  
    nopeak_mask = torch.tril(nopeak_mask).to(cfg.device) # (1, L, L) to triangular shape  
    d_mask = d_mask & nopeak_mask # (B, L, L) padding false  
  
    return e_mask, d_mask
```

```
e_mask, d_mask = create_mask(src_input, tgt_input)
```



### 3. Training

## Tạo mask cho **encoder** và **decoder** để tránh attention vào các token padding

Ví dụ:

```
1 input_data[0] # 13
```

(tensor([	1,	1960,	74,	128,	974,	146,	9,	418,	127,	41,	404,	817,
	2,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,	0,
	0,	0,	0,	0,	0,	0],						

```
1 print(e_mask[0]) # 13
```

[illegible]

### 3. Training

Đưa src input, tgt input, e mask, d mask vào mô hình Transformer

```
logits = self.model(src_input, tgt_input, e_mask, d_mask)
```

Ví dụ đưa 1 mẫu `src input`, `tgt input`, `e mask`, `d mask` với `batch size = 1`

[illegible]

```
tensor([[[ True,   True,   True,   True,   True,   True,   True,   True,   True,   True,  
           True,   True,   True, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False,  
           False, False, False, False, False, False, False, False, False, False]]]])  
  
torch.Size([1, 1, 150])  
  
tensor([[[ True, False, False, ..., False, False, False],  
         [ True,  True, False, ..., False, False, False],  
         [ True,  True,  True, ..., False, False, False],  
         ...  
         [ True,  True,  True, ..., False, False, False],  
         [ True,  True,  True, ..., False, False, False],  
         [ True,  True,  True, ..., False, False, False]])  
  
torch.Size([1, 150, 150])
```

### 3. Training

Đưa `src_input`, `tgt_input`, `e_mask`, `d_mask` vào mô hình Transformer

```
logits = self.model(src_input, tgt_input, e_mask, d_mask)
```

Ví dụ đưa 1 mẫu `src_input`, `tgt_input`, `e_mask`, `d_mask` với batch size = 1

Output:

```
1 print(logits)
2 print(logits.shape)
```

```
tensor([[[[-7.5021e+00, -1.2412e+01, -7.0736e+00, ..., -1.2030e+01,
          -1.2297e+01, -1.2295e+01],
          [-6.9140e+00, -1.2693e+01, -6.5903e+00, ..., -1.1733e+01,
          -1.1865e+01, -1.2309e+01],
          [-7.4816e+00, -1.4167e+01, -8.6468e+00, ..., -1.2696e+01,
          -1.2840e+01, -1.3195e+01],
          ...,
          [-7.8794e-05, -1.9599e+01, -1.3170e+01, ..., -1.9381e+01,
          -1.8749e+01, -1.9419e+01],
          [-7.9033e-05, -1.9655e+01, -1.3468e+01, ..., -1.9412e+01,
          -1.8816e+01, -1.9467e+01],
          [-7.8913e-05, -1.9616e+01, -1.3367e+01, ..., -1.9321e+01,
          -1.8779e+01, -1.9378e+01]]]], grad_fn=<LogSoftmaxBackward0>)
torch.Size([1, 150, 10000])
```

### 3. Training

## Tính loss và sau đó cập nhật trọng số

```
1 print(logits.view(-1, logits.shape[-1]))
2 print(logits.view(-1, logits.shape[-1]).shape)
```

```
tensor([[ -7.5021e+00, -1.2412e+01, -7.0736e+00, ..., -1.2030e+01,
         -1.2297e+01, -1.2295e+01],
        [ -6.9140e+00, -1.2693e+01, -6.5903e+00, ..., -1.1733e+01,
         -1.1865e+01, -1.2309e+01],
        [ -7.4816e+00, -1.4167e+01, -8.6468e+00, ..., -1.2696e+01,
         -1.2840e+01, -1.3195e+01],
        ...,
        [ -7.8794e-05, -1.9599e+01, -1.3170e+01, ..., -1.9381e+01,
         -1.8749e+01, -1.9419e+01],
        [ -7.9033e-05, -1.9655e+01, -1.3468e+01, ..., -1.9412e+01,
         -1.8816e+01, -1.9467e+01],
        [ -7.8913e-05, -1.9616e+01, -1.3367e+01, ..., -1.9321e+01,
         -1.8779e+01, -1.9378e+01]], grad_fn=<ViewBackward0>)
torch.Size([150, 10000])
```

```
1 print(tgt_output[0].reshape(-1))
2 print(tgt_output[0].reshape(-1).shape)
```

[illegible]

```
1 loss = criterion(  
2     logits.view(-1, logits.shape[-1]),  
3     tgt_output[0].reshape(-1)  
4 )  
5 loss.item()
```

0.3642583191394806

```
1 loss.backward()  
2 optim.step()
```

## 4. Quá trình inference

Ví dụ với câu “Tôi yêu bạn.”

## Tiền xử lí câu đầu vào

```
1 input_sentence = 'Tôi yêu bạn.'
```

```
1 print("Preprocessing input sentence...")
2 tokenized = sp_src.EncodeAsIds(input_sentence)
3 tokenized
```

```
Preprocessing input sentence...
[49, 355, 20, 308]
```

```
1 src_data = torch.LongTensor(  
2     pad_or_truncate([cfg.sos_id] + tokenized + [cfg.eos_id], cfg.seq_len, cfg.pad_id)  
3 ).unsqueeze(0).to(cfg.device)  
4 src_data
```

[illegible]

## 4. Quá trình inference

## Tạo mask

```
1 e_mask = (src_data != cfg.pad_id).unsqueeze(1).to(cfg.device) # (1, 1, L)
2 e_mask
```

[illegible]



## 4. Quá trình inference

### Embedding + positional encoding

```
1 print("Encoding input sentence...")
2 src_data = model.src_embedding(src_data)
3 src_data
```

```
Encoding input sentence...
tensor([[[ 0.0171,  0.0053, -0.0175, ...,  0.0109, -0.0197,  0.0021],
          [ 0.0038, -0.0334,  0.0255, ..., -0.0058,  0.0320,  0.0002],
          [-0.0155,  0.0117,  0.0165, ...,  0.0103, -0.0025,  0.0053],
          ...,
          [-0.0216, -0.0148,  0.0131, ..., -0.0124,  0.0135,  0.0092],
          [-0.0216, -0.0148,  0.0131, ..., -0.0124,  0.0135,  0.0092],
          [-0.0216, -0.0148,  0.0131, ..., -0.0124,  0.0135,  0.0092]]],
        grad_fn=<EmbeddingBackward0>)
```

```
1 src_data = model.positional_encoder(src_data)
2 src_data
```

```
tensor([[[ 0.3870,  1.1194, -0.3957, ...,  1.2476, -0.4466,  1.0479],
          [ 0.9268, -0.1862,  1.3790, ...,  0.8686,  0.7241,  1.0046],
          [ 0.5589, -0.0866,  1.3306, ...,  1.2332, -0.0575,  1.1208],
          ...,
          [ 0.1197, -1.2419, -0.6955, ...,  0.7183,  0.3047,  1.2071],
          [-0.8277, -0.5062, -0.1887, ...,  0.7183,  0.3047,  1.2071],
          [-1.4640,  0.3774,  0.7079, ...,  0.7183,  0.3047,  1.2071]]],
        grad_fn=<PositionalEncodingBackward0>)
```

## 4. Quá trình inference

### Embedding + positional encoding

```
1 print("Encoding input sentence...")
2 src_data = model.src_embedding(src_data)
3 src_data
```

```
Encoding input sentence...
tensor([[[ 0.0171,  0.0053, -0.0175, ...,  0.0109, -0.0197,  0.0021],
          [ 0.0038, -0.0334,  0.0255, ..., -0.0058,  0.0320,  0.0002],
          [-0.0155,  0.0117,  0.0165, ...,  0.0103, -0.0025,  0.0053],
          ...,
          [-0.0216, -0.0148,  0.0131, ..., -0.0124,  0.0135,  0.0092],
          [-0.0216, -0.0148,  0.0131, ..., -0.0124,  0.0135,  0.0092],
          [-0.0216, -0.0148,  0.0131, ..., -0.0124,  0.0135,  0.0092]]],
        grad_fn=<EmbeddingBackward0>)
```

```
1 src_data = model.positional_encoder(src_data)
2 src_data
```

```
tensor([[[ 0.3870,  1.1194, -0.3957, ...,  1.2476, -0.4466,  1.0479],
          [ 0.9268, -0.1862,  1.3790, ...,  0.8686,  0.7241,  1.0046],
          [ 0.5589, -0.0866,  1.3306, ...,  1.2332, -0.0575,  1.1208],
          ...,
          [ 0.1197, -1.2419, -0.6955, ...,  0.7183,  0.3047,  1.2071],
          [-0.8277, -0.5062, -0.1887, ...,  0.7183,  0.3047,  1.2071],
          [-1.4640,  0.3774,  0.7079, ...,  0.7183,  0.3047,  1.2071]]],
```



## 4. Quá trình inference

Đưa câu đã được embedding và positional encoding qua khối **encoder**

```
1 e_output = model.encoder(src_data, e_mask) # (1, L, d_model)

1 print(e_output)
2 print(e_output.shape)

tensor([[[[-0.1970, -1.6769,  1.1399, ..., -0.3952, -1.1380,  1.1183],
          [-0.1228, -1.9183,  1.3894, ..., -0.7759, -0.5548,  1.0149],
          [-0.2277, -1.9047,  1.2152, ..., -0.4457, -1.1053,  0.8765],
          ...,
          [ 0.0669, -2.1646,  1.2392, ..., -0.5328, -0.7571,  1.0111],
          [-0.1282, -2.0022,  1.3519, ..., -0.5859, -0.8206,  1.0823],
          [-0.2480, -1.8211,  1.5326, ..., -0.6345, -0.8304,  1.1566]]],
        grad_fn=<NativeLayerNormBackward0>)]
torch.Size([1, 150, 512])
```

## 4. Quá trình inference

Đưa e\_output qua hàm `greedy_search(e_output, e_mask)` để thu được output cuối cùng

```
1 result = greedy_search(e_output, e_mask)
```

```
1 result
```

```
'I love you .'
```

Xem chi tiết chạy từng bước tại [đây](#)