bert-transformer

April 5, 2023

[ ]:
```
!pip install tensorflow_datasets
!pip install -U tensorflow-text
!pip install --upgrade "protobuf<=3.20.1"
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: tensorflow_datasets in
/usr/local/lib/python3.9/dist-packages (4.8.3)
Requirement already satisfied: tensorflow-metadata in
/usr/local/lib/python3.9/dist-packages (from tensorflow_datasets) (1.12.0)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (8.1.3)
Requirement already satisfied: protobuf>=3.12.2 in
/usr/local/lib/python3.9/dist-packages (from tensorflow_datasets) (3.20.1)
Requirement already satisfied: psutil in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (5.9.4)
Requirement already satisfied: absl-py in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (1.4.0)
Requirement already satisfied: requests>=2.19.0 in
/usr/local/lib/python3.9/dist-packages (from tensorflow_datasets) (2.27.1)
Requirement already satisfied: termcolor in /usr/local/lib/python3.9/dist-
packages (from tensorflow_datasets) (2.2.0)
Requirement already satisfied: promise in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (2.3)
Requirement already satisfied: wrapt in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (1.14.1)
Requirement already satisfied: toml in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (0.10.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (1.22.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (4.65.0)
Requirement already satisfied: etils[enp,epath]>=0.9.0 in
/usr/local/lib/python3.9/dist-packages (from tensorflow_datasets) (1.1.1)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.9/dist-packages
(from tensorflow_datasets) (0.1.8)
Requirement already satisfied: importlib_resources in
/usr/local/lib/python3.9/dist-packages (from
```

```
etils[enp,epath]>=0.9.0->tensorflow_datasets) (5.12.0)
Requirement already satisfied: zipp in /usr/local/lib/python3.9/dist-packages
(from etils[enp,epath]>=0.9.0->tensorflow_datasets) (3.15.0)
Requirement already satisfied: typing_extensions in
/usr/local/lib/python3.9/dist-packages (from
etils[enp,epath]>=0.9.0->tensorflow_datasets) (4.5.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.9/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (1.26.15)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-
packages (from requests>=2.19.0->tensorflow_datasets) (3.4)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.9/dist-packages (from
requests>=2.19.0->tensorflow_datasets) (2022.12.7)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages
(from promise->tensorflow_datasets) (1.16.0)
Requirement already satisfied: googleapis-common-protos<2,>=1.52.0 in
/usr/local/lib/python3.9/dist-packages (from tensorflow-
metadata->tensorflow_datasets) (1.59.0)
Collecting protobuf>=3.12.2
  Downloading
protobuf-3.20.3-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.whl (1.0 MB)
                           1.0/1.0 MB
13.7 MB/s eta 0:00:00
Installing collected packages: protobuf
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.20.1
    Uninstalling protobuf-3.20.1:
      Successfully uninstalled protobuf-3.20.1
Successfully installed protobuf-3.20.3

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: tensorflow-text in /usr/local/lib/python3.9/dist-
packages (2.12.0)
Requirement already satisfied: tensorflow-hub>=0.8.0 in
/usr/local/lib/python3.9/dist-packages (from tensorflow-text) (0.13.0)
Requirement already satisfied: tensorflow<2.13,>=2.12.0 in
/usr/local/lib/python3.9/dist-packages (from tensorflow-text) (2.12.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (1.14.1)
```

```
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (0.4.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (2.2.0)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (3.20.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (1.53.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (0.32.0)
Requirement already satisfied: tensorflow-estimator<2.13,>=2.12.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (2.12.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-
packages (from tensorflow<2.13,>=2.12.0->tensorflow-text) (23.0)
Requirement already satisfied: keras<2.13,>=2.12.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (2.12.0)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (3.3.0)
Requirement already satisfied: tensorboard<2.13,>=2.12 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (2.12.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.9/dist-
packages (from tensorflow<2.13,>=2.12.0->tensorflow-text) (0.4.7)
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-
packages (from tensorflow<2.13,>=2.12.0->tensorflow-text) (67.6.1)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.9/dist-
packages (from tensorflow<2.13,>=2.12.0->tensorflow-text) (1.16.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (1.6.3)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (16.0.0)
Requirement already satisfied: numpy<1.24,>=1.22 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (1.22.4)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.9/dist-
packages (from tensorflow<2.13,>=2.12.0->tensorflow-text) (3.8.0)
Requirement already satisfied: flatbuffers>=2.0 in
```

```
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (23.3.3)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.9/dist-packages (from
tensorflow<2.13,>=2.12.0->tensorflow-text) (0.2.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.9/dist-
packages (from tensorflow<2.13,>=2.12.0->tensorflow-text) (1.4.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.9/dist-packages (from
astunparse>=1.6.0->tensorflow<2.13,>=2.12.0->tensorflow-text) (0.40.0)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.9/dist-
packages (from jax>=0.3.15->tensorflow<2.13,>=2.12.0->tensorflow-text) (1.10.1)
Requirement already satisfied: ml-dtypes>=0.0.3 in
/usr/local/lib/python3.9/dist-packages (from
jax>=0.3.15->tensorflow<2.13,>=2.12.0->tensorflow-text) (0.0.4)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.9/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (2.17.0)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.9/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (0.4.6)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (2.27.1)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (0.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.9/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (1.8.1)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.9/dist-
packages (from tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (2.2.3)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.9/dist-
packages (from tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (3.4.3)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-
packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (4.9)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.9/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (0.2.8)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (5.3.0)
```

```
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.9/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->t
ensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in
/usr/local/lib/python3.9/dist-packages (from
markdown>=2.6.8->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (6.1.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-
packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2
.12.0->tensorflow-text) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.
13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.
13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.9/dist-packages (from requests<3,>=2.21.0->tensorboard<2.
13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (1.26.15)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.9/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (2.1.2)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-
packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<2.13,>=2.12
->tensorflow<2.13,>=2.12.0->tensorflow-text) (3.15.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.9/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-
text) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/dist-
packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tenso
rboard<2.13,>=2.12->tensorflow<2.13,>=2.12.0->tensorflow-text) (3.2.2)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting protobuf<=3.20.1
  Using cached
protobuf-3.20.1-cp39-cp39-manylinux_2_5_x86_64.manylinux1_x86_64.whl (1.0 MB)
Installing collected packages: protobuf
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.20.3
    Uninstalling protobuf-3.20.3:
      Successfully uninstalled protobuf-3.20.3
```

```
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
tensorflow 2.12.0 requires protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,
!=4.21.5,<5.0.0dev,>=3.20.3, but you have protobuf 3.20.1 which is incompatible.
googleapis-common-protos 1.59.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.1,!=4.
21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1
which is incompatible.
google-cloud-translate 3.8.4 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.
1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf
3.20.1 which is incompatible.
google-cloud-language 2.6.1 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1
,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf
3.20.1 which is incompatible.
google-cloud-firestore 2.7.3 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.
1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf
3.20.1 which is incompatible.
google-cloud-datastore 2.11.1 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21
.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf
3.20.1 which is incompatible.
google-cloud-bigquery 3.4.2 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1
,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf
3.20.1 which is incompatible.
google-cloud-bigquery-storage 2.19.1 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0
,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have
protobuf 3.20.1 which is incompatible.
google-api-core 2.11.0 requires protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.
21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.19.5, but you have protobuf 3.20.1
which is incompatible.
Successfully installed protobuf-3.20.1
```

[ ]:
```python
import collections
import logging
```

```
import os
import pathlib
import re
import string
import sys
import time

import numpy as np
import matplotlib.pyplot as plt

import tensorflow_datasets as tfds
import tensorflow_text as text
import tensorflow as tf
```

[ ]:
```
logging.getLogger('tensorflow').setLevel(logging.ERROR)
```

#Dowload Data

Bọn em bộ dữ liệu từ TensorFlow để tải bộ dữ liệu tiếng Bồ Đào Nha-Anh từ Dự án dịch mở TED Talks .

Tập dữ liệu này chứa khoảng 50000 ví dụ đào tạo, 1100 ví dụ xác thực và 2000 ví dụ kiểm tra.

[ ]:
```
examples, metadata = tfds.load('ted_hrlr_translate/pt_to_en', with_info=True,
                               as_supervised=True)
train_examples, val_examples = examples['train'], examples['validation']
```

```
Downloading and preparing dataset Unknown size (download: Unknown size,
generated: Unknown size, total: Unknown size) to
/root/tensorflow_datasets/ted_hrlr_translate/pt_to_en/1.0.0…

Dl Completed…: 0 url [00:00, ? url/s]

Dl Size…: 0 MiB [00:00, ? MiB/s]

Extraction completed…: 0 file [00:00, ? file/s]

Generating splits…:   0%|          | 0/3 [00:00<?, ? splits/s]

Generating train examples…: 0 examples [00:00, ? examples/s]

Shuffling /root/tensorflow_datasets/ted_hrlr_translate/pt_to_en/1.0.0.
 ↪incompleteRWXGQ3/ted_hrlr_translate-trai…

Generating validation examples…: 0 examples [00:00, ? examples/s]

Shuffling /root/tensorflow_datasets/ted_hrlr_translate/pt_to_en/1.0.0.
 ↪incompleteRWXGQ3/ted_hrlr_translate-vali…

Generating test examples…: 0 examples [00:00, ? examples/s]

Shuffling /root/tensorflow_datasets/ted_hrlr_translate/pt_to_en/1.0.0.
 ↪incompleteRWXGQ3/ted_hrlr_translate-test…
```

```
Dataset ted_hrlr_translate downloaded and prepared to
/root/tensorflow_datasets/ted_hrlr_translate/pt_to_en/1.0.0. Subsequent calls
will reuse this data.
```

Đối tượng tf.data.Dataset được trả về bởi tập dữ liệu TensorFlow mang lại các cặp ví dụ văn bản:

[ ]:
```
for pt_examples, en_examples in train_examples.batch(3).take(1):
  for pt in pt_examples.numpy():
    print(pt.decode('utf-8'))

  print()

  for en in en_examples.numpy():
    print(en.decode('utf-8'))
```

```
e quando melhoramos a procura , tiramos a única vantagem da impressão , que é a
serendipidade .
mas e se estes fatores fossem ativos ?
mas eles não tinham a curiosidade de me testar .

and when you improve searchability , you actually take away the one advantage of
print , which is serendipity .
but what if it were active ?
but they did n't test for curiosity .
```

Chuyển đổi dữ liệu từ chữ sang số

Tại đây ta sẽ sử dụng mô hình Bert (text.BertTokenizer) để tối ưu tập dữ liệu

Tải xuống và giải nén và nhập mẫu saved_model :

[ ]:
```
model_name = "ted_hrlr_translate_pt_en_converter"
tf.keras.utils.get_file(
    f"{model_name}.zip",
    f"https://storage.googleapis.com/download.tensorflow.org/models/
  ↪{model_name}.zip",
    cache_dir='.', cache_subdir='', extract=True
)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/mod
els/ted_hrlr_translate_pt_en_converter.zip
184801/184801 [==============================] - 0s 2us/step
```

[ ]:
```
'./ted_hrlr_translate_pt_en_converter.zip'
```

[ ]:
```
tokenizers = tf.saved_model.load(model_name)
```

tf.saved_model chứa hai văn bản tokenizer, một cho tiếng Anh và một cho tiếng Bồ Đào Nha.

[ ]:
```
[item for item in dir(tokenizers.en) if not item.startswith('_')]
```

[ ]:
```
['detokenize',
 'get_reserved_tokens',
 'get_vocab_path',
 'get_vocab_size',
 'lookup',
 'tokenize',
 'tokenizer',
 'vocab']
```

Phương thức tokenize hóa chuyển đổi một loạt chuỗi thành một loạt ID mã thông báo được đệm. Phương pháp này tách dấu câu, chữ thường và unicode-chuẩn hóa đầu vào trước khi mã hóa. Sự chuẩn hóa đó không hiển thị ở đây vì dữ liệu đầu vào đã được chuẩn hóa.

[ ]:
```
for en in en_examples.numpy():
  print(en.decode('utf-8'))
```

```
and when you improve searchability , you actually take away the one advantage of
print , which is serendipity .
but what if it were active ?
but they did n't test for curiosity .
```

[ ]:
```
encoded = tokenizers.en.tokenize(en_examples)

for row in encoded.to_list():
  print(row)
```

```
[2, 72, 117, 79, 1259, 1491, 2362, 13, 79, 150, 184, 311, 71, 103, 2308, 74,
2679, 13, 148, 80, 55, 4840, 1434, 2423, 540, 15, 3]
[2, 87, 90, 107, 76, 129, 1852, 30, 3]
[2, 87, 83, 149, 50, 9, 56, 664, 85, 2512, 15, 3]
```

Phương thức detokenize cố gắng chuyển đổi các ID mã thông báo này trở lại thành văn bản có thể đọc được của con người:

[ ]:
```
round_trip = tokenizers.en.detokenize(encoded)
for line in round_trip.numpy():
  print(line.decode('utf-8'))
```

```
and when you improve searchability , you actually take away the one advantage of
print , which is serendipity .
but what if it were active ?
but they did n ' t test for curiosity .
```

Phương pháp lookup cấp thấp hơn chuyển đổi từ mã thông báo-ID thành văn bản mã thông báo:

[ ]:
```
tokens = tokenizers.en.lookup(encoded)
tokens
```

```
[ ]: <tf.RaggedTensor [[b'[START]', b'and', b'when', b'you', b'improve', b'search',
     b'##ability',
       b',', b'you', b'actually', b'take', b'away', b'the', b'one', b'advantage',
       b'of', b'print', b',', b'which', b'is', b's', b'##ere', b'##nd', b'##ip',
       b'##ity', b'.', b'[END]']                                                ,
      [b'[START]', b'but', b'what', b'if', b'it', b'were', b'active', b'?',
       b'[END]']                                                             ,
      [b'[START]', b'but', b'they', b'did', b'n', b"'", b't', b'test', b'for',
       b'curiosity', b'.', b'[END]']                                          ]>
```

Ở đây bạn có thể thấy khía cạnh "subword" của tokenizers. Từ "khả năng tìm kiếm" được phân tách thành "khả năng tìm kiếm ##" và từ "khả năng tìm kiếm" thành "s ##rect ## nd ## ip ## ity"

#Tạo đường liên kết

Để xây dựng một đường dẫn đầu vào phù hợp cho việc đào tạo, bạn sẽ áp dụng một số biến đổi cho tập dữ liệu.

Hàm này sẽ được sử dụng để mã hóa các lô văn bản thô:

```
[ ]: def tokenize_pairs(pt, en):
         pt = tokenizers.pt.tokenize(pt)
         # Convert from ragged to dense, padding with zeros.
         pt = pt.to_tensor()

         en = tokenizers.en.tokenize(en)
         # Convert from ragged to dense, padding with zeros.
         en = en.to_tensor()
         return pt, en
```

Đây là một đường dẫn đầu vào đơn giản có thể xử lý, xáo trộn và phân lô dữ liệu:

```
[ ]: BUFFER_SIZE = 20000
     BATCH_SIZE = 64
```

```
[ ]: def make_batches(ds):
       return (
           ds
           .cache()
           .shuffle(BUFFER_SIZE)
           .batch(BATCH_SIZE)
           .map(tokenize_pairs, num_parallel_calls=tf.data.AUTOTUNE)
           .prefetch(tf.data.AUTOTUNE))


     train_batches = make_batches(train_examples)
     val_batches = make_batches(val_examples)
```

#Vị trí encoding

Các lớp chú ý xem đầu vào của chúng là một tập hợp các vectơ, không có thứ tự tuần tự. Mô hình này cũng không chứa bất kỳ lớp lặp lại hoặc tích tụ nào. Do đó, một "mã hóa vị trí" được thêm vào để cung cấp cho mô hình một số thông tin về vị trí tương đối của các thẻ trong câu.

Vectơ mã hóa vị trí được thêm vào vectơ nhúng. Nhúng đại diện cho một mã thông báo trong không gian d chiều nơi các mã thông báo có ý nghĩa tương tự sẽ gần nhau hơn. Nhưng các nhúng không mã hóa vị trí tương đối của các mã thông báo trong một câu. Vì vậy, sau khi thêm mã hóa vị trí, các mã thông báo sẽ gần nhau hơn dựa trên sự giống nhau về ý nghĩa của chúng và vị trí của chúng trong câu , trong không gian d chiều.

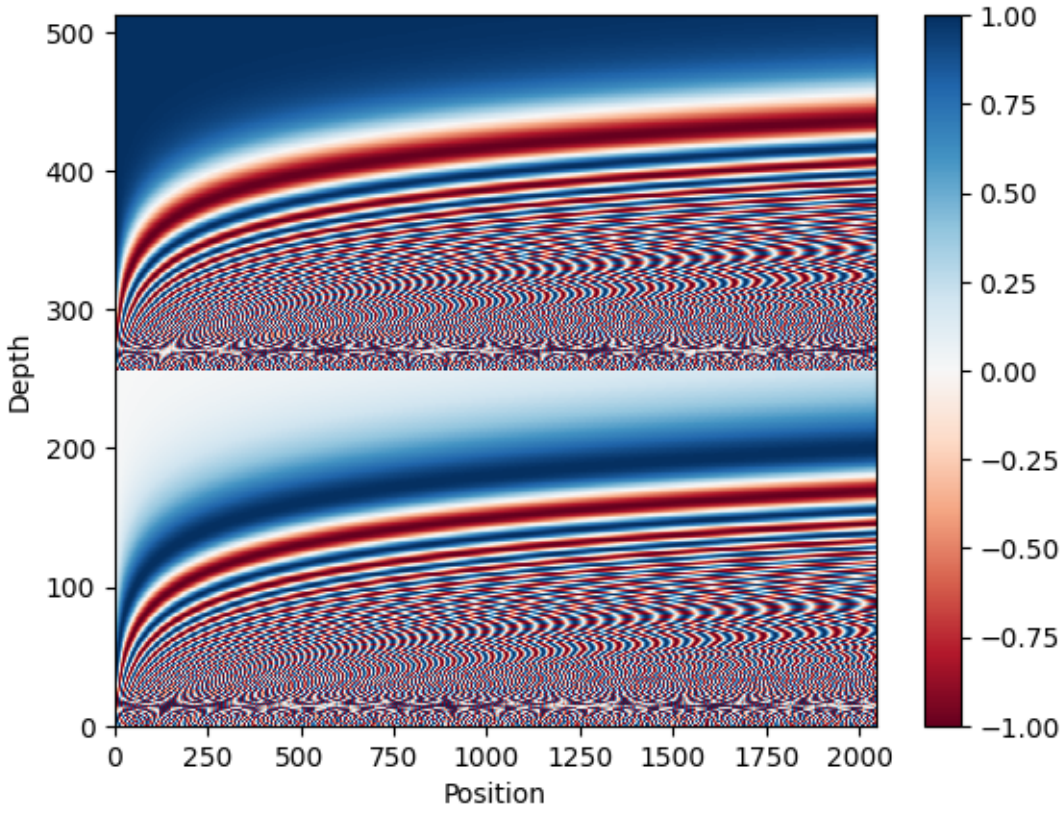
Công thức tính toán mã hóa vị trí như sau:

[ ]:
```
def get_angles(pos, i, d_model):
  angle_rates = 1 / np.power(10000, (2 * (i//2)) / np.float32(d_model))
  return pos * angle_rates
```

[ ]:
```
def positional_encoding(position, d_model):
  angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                          np.arange(d_model)[np.newaxis, :],
                          d_model)

  # apply sin to even indices in the array; 2i
  angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

  # apply cos to odd indices in the array; 2i+1
  angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

  pos_encoding = angle_rads[np.newaxis, ...]

  return tf.cast(pos_encoding, dtype=tf.float32)
```

[ ]:
```
n, d = 2048, 512
pos_encoding = positional_encoding(n, d)
print(pos_encoding.shape)
pos_encoding = pos_encoding[0]

# Juggle the dimensions for the plot
pos_encoding = tf.reshape(pos_encoding, (n, d//2, 2))
pos_encoding = tf.transpose(pos_encoding, (2, 1, 0))
pos_encoding = tf.reshape(pos_encoding, (d, n))

plt.pcolormesh(pos_encoding, cmap='RdBu')
plt.ylabel('Depth')
plt.xlabel('Position')
plt.colorbar()
plt.show()
```

```
(1, 2048, 512)
```



#Dán Nhãn

Việc dán nhãn này nhằm đảm bảo mô hình không coi padding là đầu vào. Dán nhãn cho biết vị trí có giá trị pad 0 : nó xuất ra giá trị 1 tại các vị trí đó và 0 nếu không.

[ ]:
```
def create_padding_mask(seq):
  seq = tf.cast(tf.math.equal(seq, 0), tf.float32)

  # add extra dimensions to add the padding
  # to the attention logits.
  return seq[:, tf.newaxis, tf.newaxis, :]  # (batch_size, 1, 1, seq_len)
```

[ ]:
```
x = tf.constant([[7, 6, 0, 0, 1], [1, 2, 3, 0, 0], [0, 0, 0, 4, 5]])
create_padding_mask(x)
```

[ ]:
```
<tf.Tensor: shape=(3, 1, 1, 5), dtype=float32, numpy=
array([[[[0., 0., 1., 1., 0.]]],


       [[[0., 0., 0., 1., 1.]]],
```

```
       [[[1., 1., 1., 0., 0.]]], dtype=float32)>
```

Dán nhãn nhìn trước được sử dụng để che dấu các mã thông báo trong tương lai theo một trình tự. Nói cách khác, việc dán nhãn cho biết mục nào không nên được sử dụng.

Điều này có nghĩa là để dự đoán mã thông báo thứ ba, chỉ mã thông báo đầu tiên và thứ hai sẽ được sử dụng. Tương tự như vậy để dự đoán mã thông báo thứ tư, chỉ mã thông báo đầu tiên, thứ hai và thứ ba sẽ được sử dụng, v.v.

[ ]:
```
def create_look_ahead_mask(size):
  mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
  return mask  # (seq_len, seq_len)
```

[ ]:
```
x = tf.random.uniform((1, 3))
temp = create_look_ahead_mask(x.shape[1])
temp
```

[ ]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[0., 1., 1.],
       [0., 0., 1.],
       [0., 0., 0.]], dtype=float32)>
```

#Tỷ lệ các sản phẩm

Chức năng chú ý được sử dụng bởi máy biến áp có ba đầu vào: Q (truy vấn), K (phím), V (giá trị). Phương trình được sử dụng để tính toán trọng số chú ý là:

Sự chú ý của sản phẩm chấm được chia tỷ lệ bằng hệ số căn bậc hai của độ sâu. Điều này được thực hiện bởi vì đối với các giá trị độ sâu lớn, sản phẩm chấm phát triển lớn về độ lớn đẩy hàm softmax ở nơi nó có độ dốc nhỏ dẫn đến softmax rất cứng.

Ví dụ, xem xét rằng Q và K có giá trị trung bình bằng 0 và phương sai là 1. Phép nhân ma trận của chúng sẽ có giá trị trung bình bằng 0 và phương sai là dk . Vì vậy, căn bậc hai của dk được sử dụng để chia tỷ lệ, vì vậy bạn sẽ nhận được một phương sai nhất quán bất kể giá trị của dk . Nếu phương sai quá thấp, đầu ra có thể quá phẳng để tối ưu hóa hiệu quả. Nếu phương sai quá cao, softmax có thể bão hòa khi khởi tạo, gây khó khăn cho việc học.

Mặt nạ được nhân với -1e9 (gần với âm vô cực). Điều này được thực hiện bởi vì mặt nạ được tính tổng với phép nhân ma trận tỷ lệ của Q và K và được áp dụng ngay trước một softmax. Mục tiêu là loại bỏ các ô này và đầu vào âm lớn cho softmax gần bằng 0 trong đầu ra.

[ ]:
```
def scaled_dot_product_attention(q, k, v, mask):
  """Calculate the attention weights.
```

```
  q, k, v must have matching leading dimensions.
  k, v must have matching penultimate dimension, i.e.: seq_len_k = seq_len_v.
  The mask has different shapes depending on its type(padding or look ahead)
  but it must be broadcastable for addition.

  Args:
    q: query shape == (..., seq_len_q, depth)
    k: key shape == (..., seq_len_k, depth)
    v: value shape == (..., seq_len_v, depth_v)
    mask: Float tensor with shape broadcastable
          to (..., seq_len_q, seq_len_k). Defaults to None.

  Returns:
    output, attention_weights
  """

  matmul_qk = tf.matmul(q, k, transpose_b=True)  # (..., seq_len_q, seq_len_k)

  # scale matmul_qk
  dk = tf.cast(tf.shape(k)[-1], tf.float32)
  scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

  # add the mask to the scaled tensor.
  if mask is not None:
    scaled_attention_logits += (mask * -1e9)

  # softmax is normalized on the last axis (seq_len_k) so that the scores
  # add up to 1.
  attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)  # (...,␣
↪seq_len_q, seq_len_k)

  output = tf.matmul(attention_weights, v)  # (..., seq_len_q, depth_v)

  return output, attention_weights
```

Khi quá trình chuẩn hóa softmax được thực hiện trên K, các giá trị của nó quyết định mức độ quan trọng đối với Q.

Đầu ra đại diện cho phép nhân của trọng số chú ý và vectơ V (giá trị). Điều này đảm bảo rằng các mã thông báo bạn muốn tập trung vào được giữ nguyên và các mã thông báo không liên quan sẽ bị loại bỏ.

[ ]:
```
def print_out(q, k, v):
  temp_out, temp_attn = scaled_dot_product_attention(
      q, k, v, None)
  print('Attention weights are:')
  print(temp_attn)
  print('Output is:')
```

```
  print(temp_out)
```

[ ]:
```
np.set_printoptions(suppress=True)

temp_k = tf.constant([[10, 0, 0],
                      [0, 10, 0],
                      [0, 0, 10],
                      [0, 0, 10]], dtype=tf.float32)  # (4, 3)

temp_v = tf.constant([[1, 0],
                      [10, 0],
                      [100, 5],
                      [1000, 6]], dtype=tf.float32)  # (4, 2)

# This `query` aligns with the second `key`,
# so the second `value` is returned.
temp_q = tf.constant([[0, 10, 0]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

```
Attention weights are:
tf.Tensor([[0. 1. 0. 0.]], shape=(1, 4), dtype=float32)
Output is:
tf.Tensor([[10.  0.]], shape=(1, 2), dtype=float32)
```

[ ]:
```
# This query aligns with a repeated key (third and fourth),
# so all associated values get averaged.
temp_q = tf.constant([[0, 0, 10]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

```
Attention weights are:
tf.Tensor([[0.  0.  0.5 0.5]], shape=(1, 4), dtype=float32)
Output is:
tf.Tensor([[550.    5.5]], shape=(1, 2), dtype=float32)
```

[ ]:
```
# This query aligns equally with the first and second key,
# so their values get averaged.
temp_q = tf.constant([[10, 10, 0]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

```
Attention weights are:
tf.Tensor([[0.5 0.5 0.  0. ]], shape=(1, 4), dtype=float32)
Output is:
tf.Tensor([[5.5 0. ]], shape=(1, 2), dtype=float32)
```

Vượt qua tất cả các truy vấn cùng nhau.

[ ]:
```
temp_q = tf.constant([[0, 0, 10],
                      [0, 10, 0],
```

```
                      [10, 10, 0]], dtype=tf.float32)  # (3, 3)
print_out(temp_q, temp_k, temp_v)
```

```
Attention weights are:
tf.Tensor(
[[0.  0.  0.5 0.5]
 [0.  1.  0.  0. ]
 [0.5 0.5 0.  0. ]], shape=(3, 4), dtype=float32)
Output is:
tf.Tensor(
[[550.    5.5]
 [ 10.    0. ]
 [  5.5   0. ]], shape=(3, 2), dtype=float32)
```

#Multi-Head Attention

Sự chú ý đa đầu bao gồm bốn phần:

`Các lớp tuyến tính.`

`Sự chú ý theo tỷ lệ chấm-sản phẩm.`

`Lớp tuyến tính cuối cùng.`

Mỗi khối chú ý nhiều đầu nhận được ba đầu vào; Q (truy vấn), K (khóa), V (giá trị). Chúng được đưa qua các lớp tuyến tính (Dày đặc) trước chức năng chú ý nhiều đầu.

Trong sơ đồ trên (K,Q,V) được chuyển qua các lớp tuyến tính riêng biệt (Dense) cho mỗi đầu chú ý. Để đơn giản / hiệu quả, đoạn mã dưới đây thực hiện điều này bằng cách sử dụng một lớp dày đặc duy nhất với số đầu ra num_heads nhiều lần số đầu ra. Đầu ra được sắp xếp lại thành hình dạng của (batch, num_heads, …) trước khi áp dụng hàm chú ý.

Hàm scaled_dot_product_attention được định nghĩa ở trên được áp dụng trong một lệnh gọi duy nhất, được phát sóng để đạt hiệu quả. Trong bước chú ý phải sử dụng mặt nạ thích hợp. Đầu ra chú ý cho mỗi phần đầu sau đó được nối (sử dụng tf.transpose và tf.reshape) và đưa qua một lớp Dense cuối cùng.

Thay vì một đầu chú ý duy nhất, Q, K và V được chia thành nhiều đầu vì nó cho phép mô hình cùng tham gia vào thông tin từ các không gian con biểu diễn khác nhau ở các vị trí khác nhau. Sau khi tách, mỗi phần đầu có số chiều giảm, do đó, tổng chi phí tính toán giống như sự chú ý của phần đầu duy nhất với kích thước đầy đủ.

[ ]:
```
class MultiHeadAttention(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads):
    super(MultiHeadAttention, self).__init__()
    self.num_heads = num_heads
    self.d_model = d_model
```

```
    assert d_model % self.num_heads == 0

    self.depth = d_model // self.num_heads

    self.wq = tf.keras.layers.Dense(d_model)
    self.wk = tf.keras.layers.Dense(d_model)
    self.wv = tf.keras.layers.Dense(d_model)

    self.dense = tf.keras.layers.Dense(d_model)

  def split_heads(self, x, batch_size):
    """Split the last dimension into (num_heads, depth).
    Transpose the result such that the shape is (batch_size, num_heads,␣
↪seq_len, depth)
    """
    x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
    return tf.transpose(x, perm=[0, 2, 1, 3])

  def call(self, v, k, q, mask):
    batch_size = tf.shape(q)[0]

    q = self.wq(q)  # (batch_size, seq_len, d_model)
    k = self.wk(k)  # (batch_size, seq_len, d_model)
    v = self.wv(v)  # (batch_size, seq_len, d_model)

    q = self.split_heads(q, batch_size)  # (batch_size, num_heads, seq_len_q,␣
↪depth)
    k = self.split_heads(k, batch_size)  # (batch_size, num_heads, seq_len_k,␣
↪depth)
    v = self.split_heads(v, batch_size)  # (batch_size, num_heads, seq_len_v,␣
↪depth)

    # scaled_attention.shape == (batch_size, num_heads, seq_len_q, depth)
    # attention_weights.shape == (batch_size, num_heads, seq_len_q, seq_len_k)
    scaled_attention, attention_weights = scaled_dot_product_attention(
        q, k, v, mask)

    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])  #␣
↪(batch_size, seq_len_q, num_heads, depth)

    concat_attention = tf.reshape(scaled_attention,
                                  (batch_size, -1, self.d_model))  #␣
↪(batch_size, seq_len_q, d_model)

    output = self.dense(concat_attention)  # (batch_size, seq_len_q, d_model)
```

```
    return output, attention_weights
```

Tạo một lớp MultiHeadAttention để dùng thử. Tại mỗi vị trí trong chuỗi, y , MultiHeadAttention chạy tất cả 8 đầu chú ý trên tất cả các vị trí khác trong chuỗi, trả về một vectơ mới có cùng độ dài tại mỗi vị trí.

[ ]:
```
temp_mha = MultiHeadAttention(d_model=512, num_heads=8)
y = tf.random.uniform((1, 60, 512))  # (batch_size, encoder_sequence, d_model)
out, attn = temp_mha(y, k=y, q=y, mask=None)
out.shape, attn.shape
```

[ ]:
```
(TensorShape([1, 60, 512]), TensorShape([1, 8, 60, 60]))
```

#Điểm chuyển tiếp dữ liệu

Mạng chuyển tiếp nguồn cấp dữ liệu khôn ngoan bao gồm hai lớp được kết nối đầy đủ với kích hoạt ReLU ở giữa.

[ ]:
```
def point_wise_feed_forward_network(d_model, dff):
  return tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),  # (batch_size, seq_len,␣
 ↪dff)
      tf.keras.layers.Dense(d_model)  # (batch_size, seq_len, d_model)
  ])
```

[ ]:
```
sample_ffn = point_wise_feed_forward_network(512, 2048)
sample_ffn(tf.random.uniform((64, 50, 512))).shape
```

[ ]:
```
TensorShape([64, 50, 512])
```

##Encoder Layer

[ ]:
```
class EncoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(EncoderLayer, self).__init__()

    self.mha = MultiHeadAttention(d_model, num_heads)
    self.ffn = point_wise_feed_forward_network(d_model, dff)

    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask):
```

```
    attn_output, _ = self.mha(x, x, x, mask)  # (batch_size, input_seq_len,␣
↪d_model)
    attn_output = self.dropout1(attn_output, training=training)
    out1 = self.layernorm1(x + attn_output)  # (batch_size, input_seq_len,␣
↪d_model)

    ffn_output = self.ffn(out1)  # (batch_size, input_seq_len, d_model)
    ffn_output = self.dropout2(ffn_output, training=training)
    out2 = self.layernorm2(out1 + ffn_output)  # (batch_size, input_seq_len,␣
↪d_model)

    return out2
```

[ ]:
```
sample_encoder_layer = EncoderLayer(512, 8, 2048)

sample_encoder_layer_output = sample_encoder_layer(
    tf.random.uniform((64, 43, 512)), False, None)

sample_encoder_layer_output.shape  # (batch_size, input_seq_len, d_model)
```

[ ]:
```
TensorShape([64, 43, 512])
```

##Decoder Layer

[ ]:
```
class DecoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(DecoderLayer, self).__init__()

    self.mha1 = MultiHeadAttention(d_model, num_heads)
    self.mha2 = MultiHeadAttention(d_model, num_heads)

    self.ffn = point_wise_feed_forward_network(d_model, dff)

    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm3 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)
    self.dropout3 = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,
           look_ahead_mask, padding_mask):
    # enc_output.shape == (batch_size, input_seq_len, d_model)

    attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)  #␣
↪(batch_size, target_seq_len, d_model)
```

```
    attn1 = self.dropout1(attn1, training=training)
    out1 = self.layernorm1(attn1 + x)

    attn2, attn_weights_block2 = self.mha2(
        enc_output, enc_output, out1, padding_mask)  # (batch_size,␣
↪target_seq_len, d_model)
    attn2 = self.dropout2(attn2, training=training)
    out2 = self.layernorm2(attn2 + out1)  # (batch_size, target_seq_len,␣
↪d_model)

    ffn_output = self.ffn(out2)  # (batch_size, target_seq_len, d_model)
    ffn_output = self.dropout3(ffn_output, training=training)
    out3 = self.layernorm3(ffn_output + out2)  # (batch_size, target_seq_len,␣
↪d_model)

    return out3, attn_weights_block1, attn_weights_block2
```

[ ]:
```
sample_decoder_layer = DecoderLayer(512, 8, 2048)

sample_decoder_layer_output, _, _ = sample_decoder_layer(
    tf.random.uniform((64, 50, 512)), sample_encoder_layer_output,
    False, None, None)

sample_decoder_layer_output.shape  # (batch_size, target_seq_len, d_model)
```

[ ]:
```
TensorShape([64, 50, 512])
```

##Encoder

[ ]:
```
class Encoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Encoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.embedding = tf.keras.layers.Embedding(input_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding,
                                            self.d_model)

    self.enc_layers = [EncoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]

    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask):
```

```
    seq_len = tf.shape(x)[1]

    # adding embedding and position encoding.
    x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]

    x = self.dropout(x, training=training)

    for i in range(self.num_layers):
      x = self.enc_layers[i](x, training, mask)

    return x  # (batch_size, input_seq_len, d_model)
```

[ ]:
```
sample_encoder = Encoder(num_layers=2, d_model=512, num_heads=8,
                         dff=2048, input_vocab_size=8500,
                         maximum_position_encoding=10000)
temp_input = tf.random.uniform((64, 62), dtype=tf.int64, minval=0, maxval=200)

sample_encoder_output = sample_encoder(temp_input, training=False, mask=None)

print(sample_encoder_output.shape)  # (batch_size, input_seq_len, d_model)
```

```
(64, 62, 512)
```

##Decoder

[ ]:
```
class Decoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, d_model, num_heads, dff, target_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Decoder, self).__init__()

    self.d_model = d_model
    self.num_layers = num_layers

    self.embedding = tf.keras.layers.Embedding(target_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding, d_model)

    self.dec_layers = [DecoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,
           look_ahead_mask, padding_mask):

    seq_len = tf.shape(x)[1]
    attention_weights = {}
```

```
    x = self.embedding(x)  # (batch_size, target_seq_len, d_model)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]

    x = self.dropout(x, training=training)

    for i in range(self.num_layers):
      x, block1, block2 = self.dec_layers[i](x, enc_output, training,
                                             look_ahead_mask, padding_mask)

      attention_weights[f'decoder_layer{i+1}_block1'] = block1
      attention_weights[f'decoder_layer{i+1}_block2'] = block2

    # x.shape == (batch_size, target_seq_len, d_model)
    return x, attention_weights
```

[ ]:
```
sample_decoder = Decoder(num_layers=2, d_model=512, num_heads=8,
                         dff=2048, target_vocab_size=8000,
                         maximum_position_encoding=5000)
temp_input = tf.random.uniform((64, 26), dtype=tf.int64, minval=0, maxval=200)

output, attn = sample_decoder(temp_input,
                              enc_output=sample_encoder_output,
                              training=False,
                              look_ahead_mask=None,
                              padding_mask=None)

output.shape, attn['decoder_layer2_block2'].shape
```

[ ]:
```
(TensorShape([64, 26, 512]), TensorShape([64, 8, 26, 62]))
```

#Tạo Transformer

Transformer bao gồm encoder, decoder và linear year. Đầu ra của bộ giải mã là đầu vào của lớp tuyến tính và đầu ra của nó được trả về.

[ ]:
```
class Transformer(tf.keras.Model):
  def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               target_vocab_size, pe_input, pe_target, rate=0.1):
    super().__init__()
    self.encoder = Encoder(num_layers, d_model, num_heads, dff,
                           input_vocab_size, pe_input, rate)

    self.decoder = Decoder(num_layers, d_model, num_heads, dff,
                           target_vocab_size, pe_target, rate)

    self.final_layer = tf.keras.layers.Dense(target_vocab_size)
```

```
  def call(self, inputs, training):
    # Keras models prefer if you pass all your inputs in the first argument
    inp, tar = inputs

    enc_padding_mask, look_ahead_mask, dec_padding_mask = self.
↪create_masks(inp, tar)

    enc_output = self.encoder(inp, training, enc_padding_mask)  # (batch_size,␣
↪inp_seq_len, d_model)

    # dec_output.shape == (batch_size, tar_seq_len, d_model)
    dec_output, attention_weights = self.decoder(
        tar, enc_output, training, look_ahead_mask, dec_padding_mask)

    final_output = self.final_layer(dec_output)  # (batch_size, tar_seq_len,␣
↪target_vocab_size)

    return final_output, attention_weights

  def create_masks(self, inp, tar):
    # Encoder padding mask
    enc_padding_mask = create_padding_mask(inp)

    # Used in the 2nd attention block in the decoder.
    # This padding mask is used to mask the encoder outputs.
    dec_padding_mask = create_padding_mask(inp)

    # Used in the 1st attention block in the decoder.
    # It is used to pad and mask future tokens in the input received by
    # the decoder.
    look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])
    dec_target_padding_mask = create_padding_mask(tar)
    look_ahead_mask = tf.maximum(dec_target_padding_mask, look_ahead_mask)

    return enc_padding_mask, look_ahead_mask, dec_padding_mask
```

[ ]:
```
sample_transformer = Transformer(
    num_layers=2, d_model=512, num_heads=8, dff=2048,
    input_vocab_size=8500, target_vocab_size=8000,
    pe_input=10000, pe_target=6000)

temp_input = tf.random.uniform((64, 38), dtype=tf.int64, minval=0, maxval=200)
temp_target = tf.random.uniform((64, 36), dtype=tf.int64, minval=0, maxval=200)

fn_out, _ = sample_transformer([temp_input, temp_target], training=False)
```

```
fn_out.shape  # (batch_size, tar_seq_len, target_vocab_size)
```

[ ]:
```
TensorShape([64, 36, 8000])
```

#Set hyperparameter

Để giữ cho ví dụ này nhỏ và tương đối nhanh, các giá trị cho num_layers, d_model, dff đã được giảm bớt.

Mô hình cơ sở được mô tả trong bài báo được sử dụng: num_layers=6, d_model=512, dff=2048 .

[ ]:
```
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
dropout_rate = 0.1
```

#Tối ưu hóa công thức

Sử dụng trình tối ưu hóa Adam với công cụ lập lịch tốc độ học tập tùy chỉnh theo công thức trong tài liệu

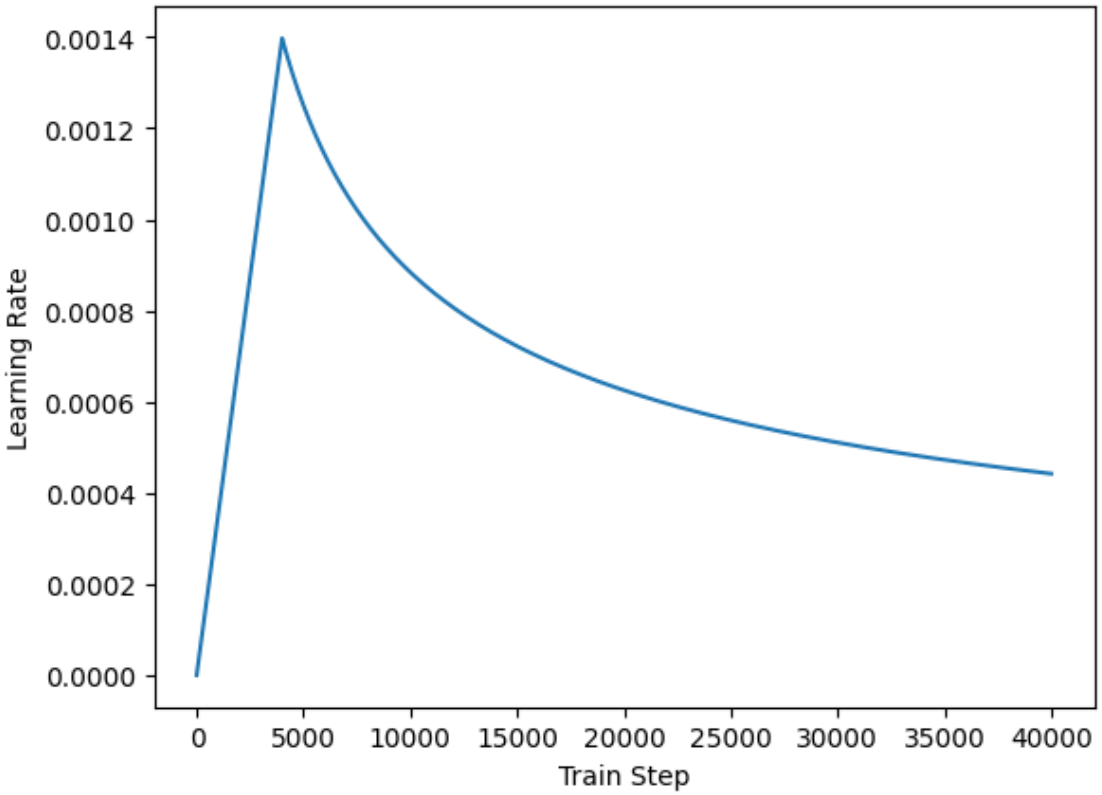
[ ]:
```
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
  def __init__(self, d_model, warmup_steps=4000):
    super().__init__()

    self.d_model = d_model
    self.d_model = tf.cast(self.d_model, tf.float32)

    self.warmup_steps = warmup_steps

  def __call__(self, step):
    step = tf.cast(step, dtype=tf.float32)
    arg1 = tf.math.rsqrt(step)
    arg2 = step * (self.warmup_steps ** -1.5)

    return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)
```

[ ]:
```
learning_rate = CustomSchedule(d_model)

optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9, beta_2=0.98,␣
 ↪epsilon=1e-9)
```

[ ]:
```
temp_learning_rate_schedule = CustomSchedule(d_model)

plt.plot(temp_learning_rate_schedule(tf.range(40000, dtype=tf.float32)))
plt.ylabel("Learning Rate")
```

```
plt.xlabel("Train Step")
```

[ ]: Text(0.5, 0, 'Train Step')



#Model Losses and figures

Vì các chuỗi mục tiêu padded, điều quan trọng là phải áp dụng cushion mask khi tính toán tổn thất

[ ]:
```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')
```

[ ]:
```
def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_sum(loss_)/tf.reduce_sum(mask)
```

```
def accuracy_function(real, pred):
  accuracies = tf.equal(real, tf.argmax(pred, axis=2))

  mask = tf.math.logical_not(tf.math.equal(real, 0))
  accuracies = tf.math.logical_and(mask, accuracies)

  accuracies = tf.cast(accuracies, dtype=tf.float32)
  mask = tf.cast(mask, dtype=tf.float32)
  return tf.reduce_sum(accuracies)/tf.reduce_sum(mask)
```

[ ]:
```
train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.Mean(name='train_accuracy')
```

#Train Transformer

[ ]:
```
transformer = Transformer(
    num_layers=num_layers,
    d_model=d_model,
    num_heads=num_heads,
    dff=dff,
    input_vocab_size=tokenizers.pt.get_vocab_size().numpy(),
    target_vocab_size=tokenizers.en.get_vocab_size().numpy(),
    pe_input=1000,
    pe_target=1000,
    rate=dropout_rate)
```

Tạo đường dẫn điểm kiểm tra và trình quản lý điểm kiểm tra. Điều này sẽ được sử dụng để lưu các điểm kiểm tra mỗi n kỷ nguyên.

[ ]:
```
checkpoint_path = "./checkpoints/train"

ckpt = tf.train.Checkpoint(transformer=transformer, optimizer=optimizer)

ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)

# if a checkpoint exists, restore the latest checkpoint.
if ckpt_manager.latest_checkpoint:
  ckpt.restore(ckpt_manager.latest_checkpoint)
  print('Latest checkpoint restored!!')
```

Mục tiêu được chia thành tar_inp và `tar_real`. tar_inp được chuyển như một đầu vào cho bộ giải mã. `tar_real` là cùng một đầu vào được dịch chuyển bằng 1: Tại mỗi vị trí trong `tar_input` , `tar_real` chứa mã thông báo tiếp theo cần được dự đoán.

Ví dụ, `sentence` = “SOS Một con sư tử trong rừng đang ngủ EOS”

`tar_inp` = “SOS Một con sư tử trong rừng đang ngủ”

`tar_real` = “Một con sư tử trong rừng đang ngủ EOS”

Transformer là một mô hình tự động hồi quy: nó đưa ra dự đoán từng phần một và sử dụng kết quả đầu ra của nó cho đến nay để quyết định phải làm gì tiếp theo.

Trong quá trình đào tạo, ví dụ này sử dụng giáo viên ép buộc (giống như trong hướng dẫn tạo văn bản). Giáo viên buộc phải chuyển đầu ra thực sự cho bước thời gian tiếp theo bất kể mô hình dự đoán những gì ở bước thời gian hiện tại.

Khi Transformer dự đoán từng mã thông báo, khả năng tự chú ý cho phép nó xem xét các mã thông báo trước đó trong trình tự đầu vào để dự đoán tốt hơn mã thông báo tiếp theo.

Để ngăn mô hình nhìn trộm đầu ra dự kiến, mô hình sử dụng mặt nạ nhìn trước.

[ ]:
```
EPOCHS = 10
```

[ ]:
```
# The @tf.function trace-compiles train_step into a TF graph for faster
# execution. The function specializes to the precise shape of the argument
# tensors. To avoid re-tracing due to the variable sequence lengths or variable
# batch sizes (the last batch is smaller), use input_signature to specify
# more generic shapes.

train_step_signature = [
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
]


@tf.function(input_signature=train_step_signature)
def train_step(inp, tar):
  tar_inp = tar[:, :-1]
  tar_real = tar[:, 1:]

  with tf.GradientTape() as tape:
    predictions, _ = transformer([inp, tar_inp],
                                 training = True)
    loss = loss_function(tar_real, predictions)

  gradients = tape.gradient(loss, transformer.trainable_variables)
  optimizer.apply_gradients(zip(gradients, transformer.trainable_variables))

  train_loss(loss)
  train_accuracy(accuracy_function(tar_real, predictions))
```

Tiếng Bồ Đào Nha được sử dụng làm ngôn ngữ đầu vào và tiếng Anh là ngôn ngữ đích.

[ ]:
```
for epoch in range(EPOCHS):
  start = time.time()

  train_loss.reset_states()
  train_accuracy.reset_states()
```

```
  # inp -> portuguese, tar -> english
  for (batch, (inp, tar)) in enumerate(train_batches):
    train_step(inp, tar)

    if batch % 50 == 0:
      print(f'Epoch {epoch + 1} Batch {batch} Loss {train_loss.result():.4f}␣
↪Accuracy {train_accuracy.result():.4f}')

  if (epoch + 1) % 5 == 0:
    ckpt_save_path = ckpt_manager.save()
    print(f'Saving checkpoint for epoch {epoch+1} at {ckpt_save_path}')

  print(f'Epoch {epoch + 1} Loss {train_loss.result():.4f} Accuracy␣
↪{train_accuracy.result():.4f}')

  print(f'Time taken for 1 epoch: {time.time() - start:.2f} secs\n')
```

```
Epoch 1 Batch 0 Loss 8.8384 Accuracy 0.0007
Epoch 1 Batch 50 Loss 8.7863 Accuracy 0.0107
Epoch 1 Batch 100 Loss 8.6917 Accuracy 0.0284
Epoch 1 Batch 150 Loss 8.5780 Accuracy 0.0355
Epoch 1 Batch 200 Loss 8.4362 Accuracy 0.0398
Epoch 1 Batch 250 Loss 8.2676 Accuracy 0.0430
Epoch 1 Batch 300 Loss 8.0807 Accuracy 0.0478
Epoch 1 Batch 350 Loss 7.8848 Accuracy 0.0542
Epoch 1 Batch 400 Loss 7.6967 Accuracy 0.0609
Epoch 1 Batch 450 Loss 7.5296 Accuracy 0.0685
Epoch 1 Batch 500 Loss 7.3820 Accuracy 0.0758
Epoch 1 Batch 550 Loss 7.2471 Accuracy 0.0833
Epoch 1 Batch 600 Loss 7.1206 Accuracy 0.0910
Epoch 1 Batch 650 Loss 7.0065 Accuracy 0.0982
Epoch 1 Batch 700 Loss 6.9001 Accuracy 0.1048
Epoch 1 Batch 750 Loss 6.8005 Accuracy 0.1111
Epoch 1 Batch 800 Loss 6.7056 Accuracy 0.1171
Epoch 1 Loss 6.6909 Accuracy 0.1180
Time taken for 1 epoch: 4043.16 secs

Epoch 2 Batch 0 Loss 5.3432 Accuracy 0.1938
Epoch 2 Batch 50 Loss 5.2541 Accuracy 0.2106
Epoch 2 Batch 100 Loss 5.2080 Accuracy 0.2153
Epoch 2 Batch 150 Loss 5.1723 Accuracy 0.2189
Epoch 2 Batch 200 Loss 5.1469 Accuracy 0.2214
Epoch 2 Batch 250 Loss 5.1230 Accuracy 0.2240
Epoch 2 Batch 300 Loss 5.0969 Accuracy 0.2265
Epoch 2 Batch 350 Loss 5.0737 Accuracy 0.2287
Epoch 2 Batch 400 Loss 5.0499 Accuracy 0.2309
```

```
Epoch 2 Batch 450 Loss 5.0298 Accuracy 0.2329
Epoch 2 Batch 500 Loss 5.0090 Accuracy 0.2346
Epoch 2 Batch 550 Loss 4.9895 Accuracy 0.2363
Epoch 2 Batch 600 Loss 4.9700 Accuracy 0.2381
Epoch 2 Batch 650 Loss 4.9531 Accuracy 0.2396
Epoch 2 Batch 700 Loss 4.9339 Accuracy 0.2413
Epoch 2 Batch 750 Loss 4.9172 Accuracy 0.2428
Epoch 2 Batch 800 Loss 4.9018 Accuracy 0.2441
Epoch 2 Loss 4.8985 Accuracy 0.2444
Time taken for 1 epoch: 3904.39 secs

Epoch 3 Batch 0 Loss 4.5705 Accuracy 0.2696
Epoch 3 Batch 50 Loss 4.5985 Accuracy 0.2690
Epoch 3 Batch 100 Loss 4.5940 Accuracy 0.2696
Epoch 3 Batch 150 Loss 4.5819 Accuracy 0.2707
Epoch 3 Batch 200 Loss 4.5649 Accuracy 0.2721
Epoch 3 Batch 250 Loss 4.5578 Accuracy 0.2729
Epoch 3 Batch 300 Loss 4.5466 Accuracy 0.2739
Epoch 3 Batch 350 Loss 4.5351 Accuracy 0.2754
Epoch 3 Batch 400 Loss 4.5220 Accuracy 0.2766
Epoch 3 Batch 450 Loss 4.5067 Accuracy 0.2786
Epoch 3 Batch 500 Loss 4.4918 Accuracy 0.2802
Epoch 3 Batch 550 Loss 4.4782 Accuracy 0.2817
Epoch 3 Batch 600 Loss 4.4659 Accuracy 0.2832
Epoch 3 Batch 650 Loss 4.4517 Accuracy 0.2847
Epoch 3 Batch 700 Loss 4.4365 Accuracy 0.2866
Epoch 3 Batch 750 Loss 4.4222 Accuracy 0.2884
Epoch 3 Batch 800 Loss 4.4073 Accuracy 0.2902
Epoch 3 Loss 4.4046 Accuracy 0.2906
Time taken for 1 epoch: 3874.52 secs

Epoch 4 Batch 0 Loss 4.1312 Accuracy 0.3059
Epoch 4 Batch 50 Loss 4.0977 Accuracy 0.3222
Epoch 4 Batch 100 Loss 4.0767 Accuracy 0.3262
Epoch 4 Batch 150 Loss 4.0616 Accuracy 0.3284
Epoch 4 Batch 200 Loss 4.0494 Accuracy 0.3300
Epoch 4 Batch 250 Loss 4.0268 Accuracy 0.3331
Epoch 4 Batch 300 Loss 4.0116 Accuracy 0.3353
Epoch 4 Batch 350 Loss 3.9950 Accuracy 0.3377
Epoch 4 Batch 400 Loss 3.9780 Accuracy 0.3401
Epoch 4 Batch 450 Loss 3.9609 Accuracy 0.3421
Epoch 4 Batch 500 Loss 3.9432 Accuracy 0.3446
Epoch 4 Batch 550 Loss 3.9285 Accuracy 0.3466
Epoch 4 Batch 600 Loss 3.9147 Accuracy 0.3484
Epoch 4 Batch 650 Loss 3.9013 Accuracy 0.3503
Epoch 4 Batch 700 Loss 3.8876 Accuracy 0.3523
Epoch 4 Batch 750 Loss 3.8725 Accuracy 0.3544
Epoch 4 Batch 800 Loss 3.8601 Accuracy 0.3561
```

```
Epoch 4 Loss 3.8591 Accuracy 0.3564
Time taken for 1 epoch: 3926.28 secs

Epoch 5 Batch 0 Loss 3.5099 Accuracy 0.4080
Epoch 5 Batch 50 Loss 3.5681 Accuracy 0.3893
Epoch 5 Batch 100 Loss 3.5353 Accuracy 0.3942
Epoch 5 Batch 150 Loss 3.5191 Accuracy 0.3968
Epoch 5 Batch 200 Loss 3.5087 Accuracy 0.3975
Epoch 5 Batch 250 Loss 3.4974 Accuracy 0.3995
Epoch 5 Batch 300 Loss 3.4879 Accuracy 0.4009
Epoch 5 Batch 350 Loss 3.4774 Accuracy 0.4022
Epoch 5 Batch 400 Loss 3.4743 Accuracy 0.4030
Epoch 5 Batch 450 Loss 3.4649 Accuracy 0.4044
Epoch 5 Batch 500 Loss 3.4533 Accuracy 0.4059
Epoch 5 Batch 550 Loss 3.4440 Accuracy 0.4072
Epoch 5 Batch 600 Loss 3.4366 Accuracy 0.4085
Epoch 5 Batch 650 Loss 3.4284 Accuracy 0.4093
Epoch 5 Batch 700 Loss 3.4215 Accuracy 0.4103
Epoch 5 Batch 750 Loss 3.4136 Accuracy 0.4113
Epoch 5 Batch 800 Loss 3.4046 Accuracy 0.4124
Saving checkpoint for epoch 5 at ./checkpoints/train/ckpt-1
Epoch 5 Loss 3.4030 Accuracy 0.4126
Time taken for 1 epoch: 3910.39 secs

Epoch 6 Batch 0 Loss 3.2355 Accuracy 0.4079
Epoch 6 Batch 50 Loss 3.1405 Accuracy 0.4387
Epoch 6 Batch 100 Loss 3.1272 Accuracy 0.4425
Epoch 6 Batch 150 Loss 3.1161 Accuracy 0.4450
Epoch 6 Batch 200 Loss 3.1113 Accuracy 0.4459
Epoch 6 Batch 250 Loss 3.1037 Accuracy 0.4469
Epoch 6 Batch 300 Loss 3.0928 Accuracy 0.4483
Epoch 6 Batch 350 Loss 3.0923 Accuracy 0.4486
Epoch 6 Batch 400 Loss 3.0791 Accuracy 0.4505
Epoch 6 Batch 450 Loss 3.0755 Accuracy 0.4509
Epoch 6 Batch 500 Loss 3.0680 Accuracy 0.4520
Epoch 6 Batch 550 Loss 3.0579 Accuracy 0.4534
Epoch 6 Batch 600 Loss 3.0483 Accuracy 0.4551
Epoch 6 Batch 650 Loss 3.0412 Accuracy 0.4562
Epoch 6 Batch 700 Loss 3.0344 Accuracy 0.4572
Epoch 6 Batch 750 Loss 3.0258 Accuracy 0.4585
Epoch 6 Batch 800 Loss 3.0216 Accuracy 0.4591
Epoch 6 Loss 3.0202 Accuracy 0.4593
Time taken for 1 epoch: 3884.98 secs

Epoch 7 Batch 0 Loss 2.6432 Accuracy 0.4938
Epoch 7 Batch 50 Loss 2.7666 Accuracy 0.4873
Epoch 7 Batch 100 Loss 2.7577 Accuracy 0.4890
Epoch 7 Batch 150 Loss 2.7517 Accuracy 0.4905
```

```
Epoch 7 Batch 200 Loss 2.7508 Accuracy 0.4912
Epoch 7 Batch 250 Loss 2.7453 Accuracy 0.4921
Epoch 7 Batch 300 Loss 2.7372 Accuracy 0.4935
Epoch 7 Batch 350 Loss 2.7334 Accuracy 0.4941
Epoch 7 Batch 400 Loss 2.7296 Accuracy 0.4950
Epoch 7 Batch 450 Loss 2.7279 Accuracy 0.4955
Epoch 7 Batch 500 Loss 2.7232 Accuracy 0.4961
Epoch 7 Batch 550 Loss 2.7177 Accuracy 0.4970
Epoch 7 Batch 600 Loss 2.7129 Accuracy 0.4979
Epoch 7 Batch 650 Loss 2.7073 Accuracy 0.4987
Epoch 7 Batch 700 Loss 2.7015 Accuracy 0.4997
Epoch 7 Batch 750 Loss 2.6978 Accuracy 0.5004
Epoch 7 Batch 800 Loss 2.6940 Accuracy 0.5011
Epoch 7 Loss 2.6926 Accuracy 0.5013
Time taken for 1 epoch: 3847.52 secs

Epoch 8 Batch 0 Loss 2.4496 Accuracy 0.5343
Epoch 8 Batch 50 Loss 2.4452 Accuracy 0.5321
Epoch 8 Batch 100 Loss 2.4590 Accuracy 0.5304
Epoch 8 Batch 150 Loss 2.4664 Accuracy 0.5299
Epoch 8 Batch 200 Loss 2.4572 Accuracy 0.5310
Epoch 8 Batch 250 Loss 2.4668 Accuracy 0.5299
Epoch 8 Batch 300 Loss 2.4686 Accuracy 0.5296
Epoch 8 Batch 350 Loss 2.4685 Accuracy 0.5299
Epoch 8 Batch 400 Loss 2.4684 Accuracy 0.5299
Epoch 8 Batch 450 Loss 2.4661 Accuracy 0.5306
Epoch 8 Batch 500 Loss 2.4592 Accuracy 0.5318
Epoch 8 Batch 550 Loss 2.4570 Accuracy 0.5318
Epoch 8 Batch 600 Loss 2.4565 Accuracy 0.5321
Epoch 8 Batch 650 Loss 2.4533 Accuracy 0.5326
Epoch 8 Batch 700 Loss 2.4521 Accuracy 0.5329
Epoch 8 Batch 750 Loss 2.4520 Accuracy 0.5329
Epoch 8 Batch 800 Loss 2.4508 Accuracy 0.5331
Epoch 8 Loss 2.4509 Accuracy 0.5331
Time taken for 1 epoch: 3862.52 secs

Epoch 9 Batch 0 Loss 2.4516 Accuracy 0.5273
Epoch 9 Batch 50 Loss 2.2725 Accuracy 0.5545
Epoch 9 Batch 100 Loss 2.2955 Accuracy 0.5519
Epoch 9 Batch 150 Loss 2.2795 Accuracy 0.5551
Epoch 9 Batch 200 Loss 2.2792 Accuracy 0.5555
Epoch 9 Batch 250 Loss 2.2790 Accuracy 0.5558
Epoch 9 Batch 300 Loss 2.2755 Accuracy 0.5562
Epoch 9 Batch 350 Loss 2.2715 Accuracy 0.5571
Epoch 9 Batch 400 Loss 2.2761 Accuracy 0.5564
Epoch 9 Batch 450 Loss 2.2762 Accuracy 0.5562
Epoch 9 Batch 500 Loss 2.2752 Accuracy 0.5566
Epoch 9 Batch 550 Loss 2.2726 Accuracy 0.5570
```

```
Epoch 9 Batch 600 Loss 2.2720 Accuracy 0.5570
Epoch 9 Batch 650 Loss 2.2714 Accuracy 0.5572
Epoch 9 Batch 700 Loss 2.2726 Accuracy 0.5571
Epoch 9 Batch 750 Loss 2.2737 Accuracy 0.5570
Epoch 9 Batch 800 Loss 2.2726 Accuracy 0.5572
Epoch 9 Loss 2.2726 Accuracy 0.5573
Time taken for 1 epoch: 3862.54 secs

Epoch 10 Batch 0 Loss 2.2027 Accuracy 0.5744
Epoch 10 Batch 50 Loss 2.1362 Accuracy 0.5764
Epoch 10 Batch 100 Loss 2.1363 Accuracy 0.5753
Epoch 10 Batch 150 Loss 2.1375 Accuracy 0.5760
Epoch 10 Batch 200 Loss 2.1336 Accuracy 0.5768
Epoch 10 Batch 250 Loss 2.1340 Accuracy 0.5765
Epoch 10 Batch 300 Loss 2.1403 Accuracy 0.5756
Epoch 10 Batch 350 Loss 2.1403 Accuracy 0.5757
Epoch 10 Batch 400 Loss 2.1390 Accuracy 0.5760
Epoch 10 Batch 450 Loss 2.1391 Accuracy 0.5757
Epoch 10 Batch 500 Loss 2.1358 Accuracy 0.5764
Epoch 10 Batch 550 Loss 2.1336 Accuracy 0.5766
Epoch 10 Batch 600 Loss 2.1330 Accuracy 0.5768
Epoch 10 Batch 650 Loss 2.1348 Accuracy 0.5767
Epoch 10 Batch 700 Loss 2.1337 Accuracy 0.5770
Epoch 10 Batch 750 Loss 2.1329 Accuracy 0.5772
Epoch 10 Batch 800 Loss 2.1320 Accuracy 0.5775
Saving checkpoint for epoch 10 at ./checkpoints/train/ckpt-2
Epoch 10 Loss 2.1323 Accuracy 0.5776
Time taken for 1 epoch: 3743.19 secs
```

#Chạy suy luận

Các bước sau được sử dụng để suy luận:

- Mã hóa câu đầu vào bằng trình mã hóa tiếng Bồ Đào Nha (`tokenizers.pt`). Đây là đầu vào của bộ mã hóa.
- Đầu vào của bộ giải mã được khởi tạo thành mã thông báo `[START]` .
- Tính toán mặt nạ đệm và mặt nạ nhìn trước.
- Sau đó, decoder đưa ra các dự đoán bằng cách xem đầu ra của encoder output của chính nó (tự chú ý)
- Nối mã thông báo dự đoán với đầu vào của bộ giải mã và chuyển nó tới bộ giải mã.
- Trong cách tiếp cận này, bộ giải mã dự đoán mã thông báo tiếp theo dựa trên các mã thông báo trước đó nó đã dự đoán.

[ ]:
```python
class Translator(tf.Module):
  def __init__(self, tokenizers, transformer):
    self.tokenizers = tokenizers
    self.transformer = transformer
```

```
def __call__(self, sentence, max_length=20):
  # input sentence is portuguese, hence adding the start and end token
  assert isinstance(sentence, tf.Tensor)
  if len(sentence.shape) == 0:
    sentence = sentence[tf.newaxis]

  sentence = self.tokenizers.pt.tokenize(sentence).to_tensor()

  encoder_input = sentence

  # as the target is english, the first token to the transformer should be the
  # english start token.
  start_end = self.tokenizers.en.tokenize([''])[0]
  start = start_end[0][tf.newaxis]
  end = start_end[1][tf.newaxis]

  # `tf.TensorArray` is required here (instead of a python list) so that the
  # dynamic-loop can be traced by `tf.function`.
  output_array = tf.TensorArray(dtype=tf.int64, size=0, dynamic_size=True)
  output_array = output_array.write(0, start)

  for i in tf.range(max_length):
    output = tf.transpose(output_array.stack())
    predictions, _ = self.transformer([encoder_input, output], training=False)

    # select the last token from the seq_len dimension
    predictions = predictions[:, -1:, :]  # (batch_size, 1, vocab_size)

    predicted_id = tf.argmax(predictions, axis=-1)

    # concatentate the predicted_id to the output which is given to the␣
↪decoder
    # as its input.
    output_array = output_array.write(i+1, predicted_id[0])

    if predicted_id == end:
      break

  output = tf.transpose(output_array.stack())
  # output.shape (1, tokens)
  text = tokenizers.en.detokenize(output)[0]  # shape: ()

  tokens = tokenizers.en.lookup(output)[0]

  # `tf.function` prevents us from using the attention_weights that were
  # calculated on the last iteration of the loop. So recalculate them outside
  # the loop.
```

```
    _, attention_weights = self.transformer([encoder_input, output[:,:-1]],␣
 ↪training=False)

    return text, tokens, attention_weights
```

Tạo một phiên bản của lớp `Translator` này và dùng thử một vài lần:

[ ]:
```
translator = Translator(tokenizers, transformer)
```

[ ]:
```
def print_translation(sentence, tokens, ground_truth):
  print(f'{"Input:":15s}: {sentence}')
  print(f'{"Prediction":15s}: {tokens.numpy().decode("utf-8")}')
  print(f'{"Ground truth":15s}: {ground_truth}')
```

[ ]:
```
sentence = "este é um problema que temos que resolver."
ground_truth = "this is a problem we have to solve ."

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : este é um problema que temos que resolver.
Prediction      : this is a problem that we have to solve .
Ground truth    : this is a problem we have to solve .
```

[ ]:
```
sentence = "os meus vizinhos ouviram sobre esta ideia."
ground_truth = "and my neighboring homes heard about this idea ."

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : os meus vizinhos ouviram sobre esta ideia.
Prediction      : my neighbors heard about this idea .
Ground truth    : and my neighboring homes heard about this idea .
```

[ ]:
```
sentence = "vou então muito rapidamente partilhar convosco algumas histórias de␣
 ↪algumas coisas mágicas que aconteceram."
ground_truth = "so i \'ll just share with you some stories very quickly of some␣
 ↪magical things that have happened ."

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : vou então muito rapidamente partilhar convosco algumas
histórias de algumas coisas mágicas que aconteceram.
Prediction      : so i ' m going to share a very quickly thing to share you some
```

```
amazing stories that happened .
Ground truth    : so i 'll just share with you some stories very quickly of some
magical things that have happened .
```

[ ]:
```
sentence = "este é o primeiro livro que eu fiz."
ground_truth = "this is the first book i've ever done."

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)
```

```
Input:          : este é o primeiro livro que eu fiz.
Prediction      : this is the first book i did .
Ground truth    : this is the first book i've ever done.
```

[ ]:
```
def plot_attention_head(in_tokens, translated_tokens, attention):
  # The plot is of the attention when a token was generated.
  # The model didn't generate `<START>` in the output. Skip it.
  translated_tokens = translated_tokens[1:]

  ax = plt.gca()
  ax.matshow(attention)
  ax.set_xticks(range(len(in_tokens)))
  ax.set_yticks(range(len(translated_tokens)))

  labels = [label.decode('utf-8') for label in in_tokens.numpy()]
  ax.set_xticklabels(
      labels, rotation=90)

  labels = [label.decode('utf-8') for label in translated_tokens.numpy()]
  ax.set_yticklabels(labels)
```

[ ]:
```
head = 0
# shape: (batch=1, num_heads, seq_len_q, seq_len_k)
attention_heads = tf.squeeze(
  attention_weights['decoder_layer4_block2'], 0)
attention = attention_heads[head]
attention.shape
```

[ ]:
```
TensorShape([9, 11])
```

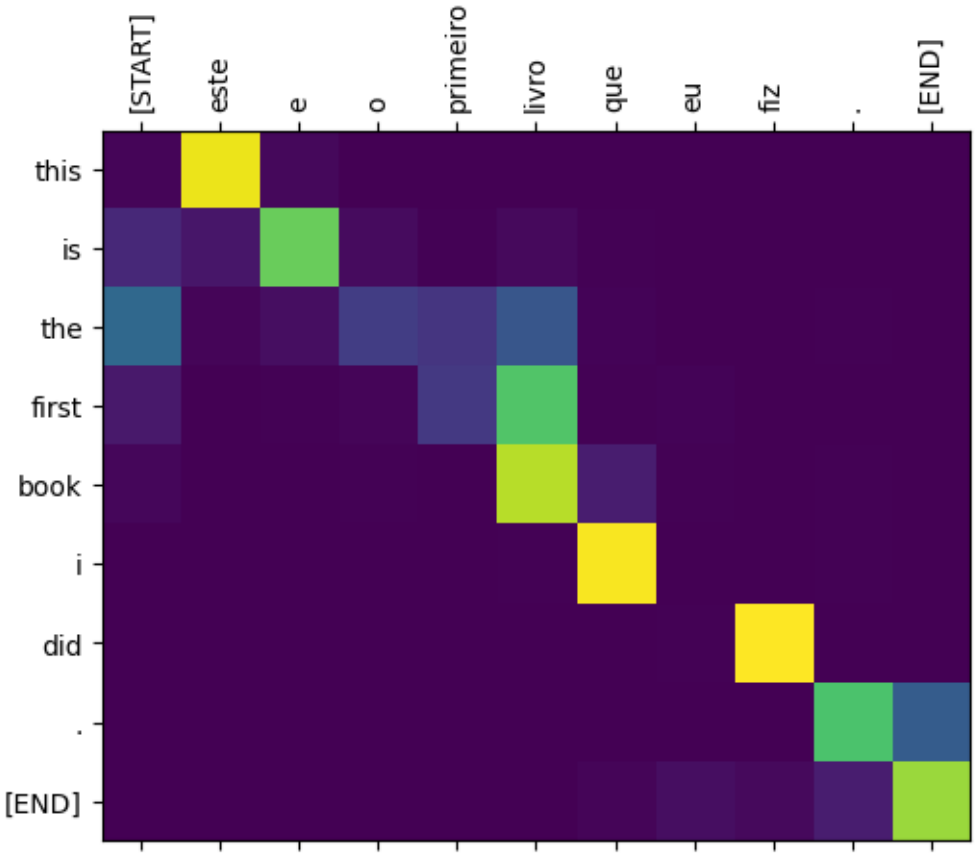
[ ]:
```
in_tokens = tf.convert_to_tensor([sentence])
in_tokens = tokenizers.pt.tokenize(in_tokens).to_tensor()
in_tokens = tokenizers.pt.lookup(in_tokens)[0]
in_tokens
```

[ ]:
```
<tf.Tensor: shape=(11,), dtype=string, numpy=
array([b'[START]', b'este', b'e', b'o', b'primeiro', b'livro', b'que',
       b'eu', b'fiz', b'.', b'[END]'], dtype=object)>
```

[ ]:
```
translated_tokens
```

[ ]:
```
<tf.Tensor: shape=(10,), dtype=string, numpy=
array([b'[START]', b'this', b'is', b'the', b'first', b'book', b'i',
       b'did', b'.', b'[END]'], dtype=object)>
```

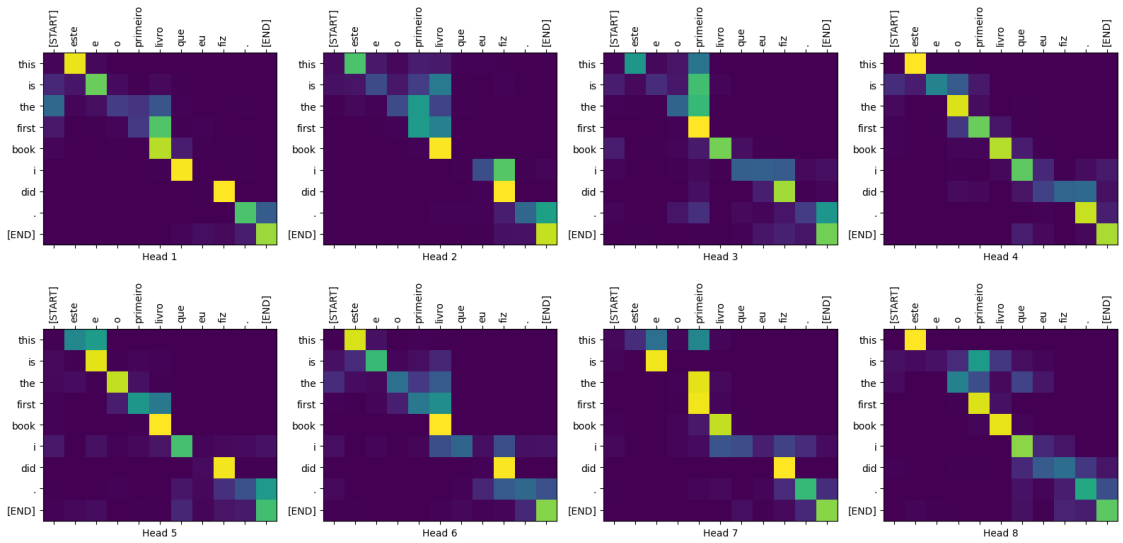
[ ]:
```
plot_attention_head(in_tokens, translated_tokens, attention)
```



[ ]:
```
def plot_attention_weights(sentence, translated_tokens, attention_heads):
  in_tokens = tf.convert_to_tensor([sentence])
  in_tokens = tokenizers.pt.tokenize(in_tokens).to_tensor()
  in_tokens = tokenizers.pt.lookup(in_tokens)[0]
  in_tokens

  fig = plt.figure(figsize=(16, 8))
```

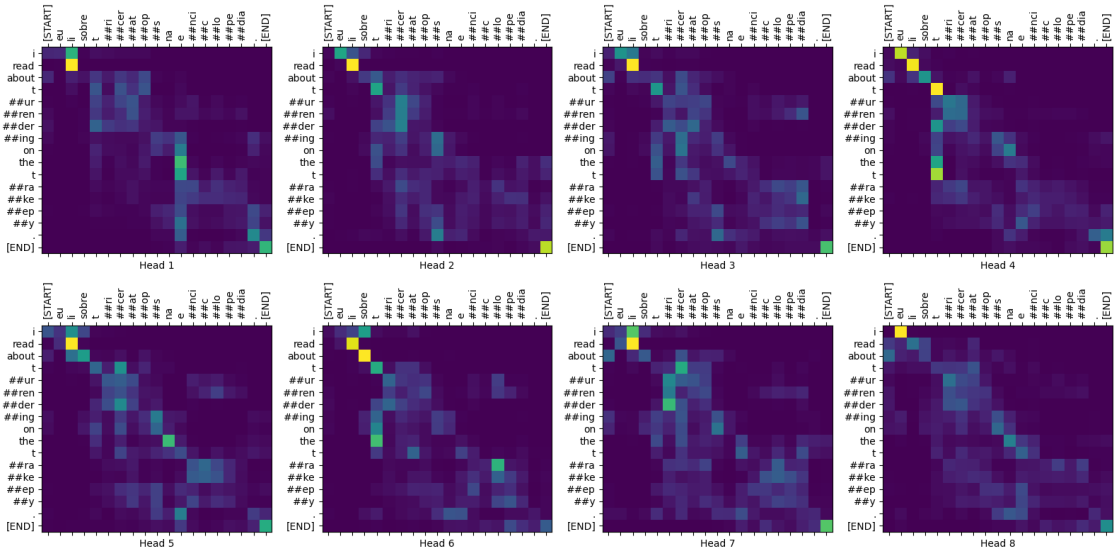
```
  for h, head in enumerate(attention_heads):
    ax = fig.add_subplot(2, 4, h+1)

    plot_attention_head(in_tokens, translated_tokens, head)

    ax.set_xlabel(f'Head {h+1}')

  plt.tight_layout()
  plt.show()
```

[ ]:
```
plot_attention_weights(sentence, translated_tokens,
                       attention_weights['decoder_layer4_block2'][0])
```



[ ]:
```
sentence = "Eu li sobre triceratops na enciclopédia."
ground_truth = "I read about triceratops in the encyclopedia."

translated_text, translated_tokens, attention_weights = translator(
    tf.constant(sentence))
print_translation(sentence, translated_text, ground_truth)

plot_attention_weights(sentence, translated_tokens,
                       attention_weights['decoder_layer4_block2'][0])
```

```
Input:          : Eu li sobre triceratops na enciclopédia.
Prediction      : i read about turrendering on the trakeepy .
Ground truth    : I read about triceratops in the encyclopedia.
```



[ ]:
```
class ExportTranslator(tf.Module):
  def __init__(self, translator):
    self.translator = translator

  @tf.function(input_signature=[tf.TensorSpec(shape=[], dtype=tf.string)])
  def __call__(self, sentence):
    (result,
     tokens,
     attention_weights) = self.translator(sentence, max_length=100)

    return result
```

[ ]:
```
translator = ExportTranslator(translator)
```

[ ]:
```
translator("este é o primeiro livro que eu fiz.").numpy()
```

[ ]:
```
b'this is the first book i did .'
```

[ ]:
```
tf.saved_model.save(translator, export_dir='translator')
```

```
WARNING:absl:Found untraced functions such as embedding_4_layer_call_fn,
embedding_4_layer_call_and_return_conditional_losses, dropout_37_layer_call_fn,
dropout_37_layer_call_and_return_conditional_losses, embedding_5_layer_call_fn
while saving (showing 5 of 224). These functions will not be directly callable
after loading.
```

[ ]:
```
reloaded = tf.saved_model.load('translator')
```

[ ]:
```
reloaded("este é o primeiro livro que eu fiz.").numpy()
```

[ ]: b'this is the first book i did .'