```
import tensorflow as tf
tf.random.set_seed(1234)
AUTO = tf.data.experimental.AUTOTUNE
!pip install tensorflow-datasets==1.2.0
import tensorflow_datasets as tfds
import re
import sys
from time import time
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (de
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98: UserWarning: unable to load libtensorflow_io_plugins.so: unable to open f
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined symbol: _ZN3tsl6StatusC1EN10tensorflo
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system plugins are not loaded: unable to open file: li
caused by: ['/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol: _ZTVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")
Collecting tensorflow-datasets==1.2.0
  Downloading tensorflow_datasets-1.2.0-py3-none-any.whl (2.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 22.5 MB/s eta 0:00:0000:010:01
Requirement already satisfied: absl-py in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (1.4.0)
Requirement already satisfied: attrs in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (23.1.0)
Requirement already satisfied: dill in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (0.3.6)
Requirement already satisfied: future in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (0.18.3)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (1.23.5)
Requirement already satisfied: promise in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (2.3)
Requirement already satisfied: protobuf>=3.6.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (3.20.3)
Requirement already satisfied: psutil in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (5.9.3)
Requirement already satisfied: requests>=2.19.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (2.31.0)
Requirement already satisfied: six in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (1.16.0)
Requirement already satisfied: tensorflow-metadata in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (0.14.0)
Requirement already satisfied: termcolor in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (2.3.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (4.65.0)
Requirement already satisfied: wrapt in /opt/conda/lib/python3.10/site-packages (from tensorflow-datasets==1.2.0) (1.14.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->tensorflow-datasets==1.2.0) (3
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->tensorflow-datasets==1.2.0) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->tensorflow-datasets==1.2.0) (1.26.15
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->tensorflow-datasets==1.2.0) (2023.5.7
Requirement already satisfied: googleapis-common-protos in /opt/conda/lib/python3.10/site-packages (from tensorflow-metadata->tensorflow-datasets==1.2.0)
Installing collected packages: tensorflow-datasets
  Attempting uninstall: tensorflow-datasets
    Found existing installation: tensorflow-datasets 4.9.2
    Uninstalling tensorflow-datasets-4.9.2:
      Successfully uninstalled tensorflow-datasets-4.9.2
Successfully installed tensorflow-datasets-1.2.0
```

```
/kaggle/input/covid-chitchat/9L_dataset.json
/kaggle/input/exemplary-empathy-2490/emotion_train.csv
/kaggle/input/emphetic-dialog-fb/emotion-emotion_69k.csv
/kaggle/input/towards-empathetic/emotion-emotion_69k.csv
```

Read Data

```
! pip -q install datasets
```

DailyDialog

```
from datasets import load_dataset
dataset = load_dataset("roskoN/dailydialog")
```

```
Downloading builder script:   0%|          | 0.00/4.59k [00:00<?, ?B/s]
Downloading and preparing dataset daily_dialog/full to /root/.cache/huggingface/datasets/roskoN___daily_dialog/full/1.0.0/7d96d5a6afcb95cf518611d5147758f4
Downloading data files:   0%|          | 0/3 [00:00<?, ?it/s]
Downloading data:   0%|          | 0.00/1.94M [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/180k [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/179k [00:00<?, ?B/s]
Generating train split: 0 examples [00:00, ? examples/s]
Generating validation split: 0 examples [00:00, ? examples/s]
Generating test split: 0 examples [00:00, ? examples/s]
Dataset daily_dialog downloaded and prepared to /root/.cache/huggingface/datasets/roskoN___daily_dialog/full/1.0.0/7d96d5a6afcb95cf518611d5147758f4a5991ba
  0%|          | 0/3 [00:00<?, ?it/s]
```

```
import csv
conversation = dataset['train']['utterances']
conversation = [utterance for sublist in conversation for utterance in sublist]
prompts = conversation[0::2]
responses = conversation[1::2]

csv_filename = "dailydialog_conversation.csv"
with open(csv_filename, mode='w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(["Prompt", "Response"])

    for prompt, response in zip(prompts, responses):
        csv_writer.writerow([prompt.strip(), response.strip()])

df = pd.read_csv(csv_filename)
```

Empathetic Dialogues

```
from datasets import load_dataset
dataset = load_dataset("benjaminbeilharz/empathetic_dialogues_for_lm")
```

```
Downloading:   0%|          | 0.00/886 [00:00<?, ?B/s]
Downloading and preparing dataset None/None (download: 5.81 MiB, generated: 11.03 MiB, post-processed: Unknown size, total: 16.84 MiB) to /root/.cache/hug
Downloading data files:   0%|          | 0/3 [00:00<?, ?it/s]
Downloading data:   0%|          | 0.00/4.53M [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/801k [00:00<?, ?B/s]
Downloading data:   0%|          | 0.00/767k [00:00<?, ?B/s]
Extracting data files:   0%|          | 0/3 [00:00<?, ?it/s]
Dataset parquet downloaded and prepared to /root/.cache/huggingface/datasets/parquet/benjaminbeilharz--empathetic_dialog_for_lm-050aa011e4709962/0.0.0/0b
  0%|          | 0/3 [00:00<?, ?it/s]
```

```
import csv
c = dataset['train']['conv']
c = [u for sublist in c for u in sublist]
prompts = c[0::2]
responses = c[1::2]

csv_file = "empathetic_dialogues_for_lm.csv"
with open(csv_file, mode='w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(["Prompt", "Response"])

    for prompt, response in zip(prompts, responses):
        csv_writer.writerow([prompt.strip(), response.strip()])

dt = pd.read_csv(csv_file)
```

Concat data

```
#concat_q = pd.concat([dt['Prompt'], d['seeker_post'],d1['question'],df['Prompt']], ignore_index=True)
concat_q = pd.concat([d1['question'], d['seeker_post']], ignore_index=True)
concat_q.dropna(inplace=True)
prompt = concat_q.tolist()

#concat_a = pd.concat([dt['Response'], d['response_post'], d1['answer'],df['Response']], ignore_index=True)
concat_a = pd.concat([d1['answer'], d['response_post']], ignore_index=True)
concat_a.dropna(inplace=True)
response = concat_a.tolist()
```

```
print(len(prompt))
print(len(response))
```

```
22489
22489
```

Hyperparameters

```
max_len = 60
max_sample = 117125
batch_size = 64
buffer_size = 80000
number_of_layer = 2
d_model = 512
number_of_head = 8
unit = 128
dropout = 0.1
```

Data preprocess

```
def text_preprocess(s):
    s = s.lower().strip()
    s= re.sub(r"([?.!,])", r" \1 ", s)
    s = re.sub(r'[" "]+', " ", s)
    s = re.sub(r"[^a-zA-Z?.!,]+", " ", s)
    s = s.strip()
    return s

prompt = [text_preprocess(s) for s in prompt]
response = [text_preprocess(s) for s in response]
```

Build Prompt and Response

```
tokenizer = tfds.features.text.SubwordTextEncoder.build_from_corpus(prompt + response, target_vocab_size=8000)

s_token, e_token = [tokenizer.vocab_size], [tokenizer.vocab_size + 1]

vocab_size = tokenizer.vocab_size + 2

t_prompt, t_response = [], []
```

```
for (i, j) in zip(prompt, response):
    i = s_token + tokenizer.encode(i) + e_token
    j = s_token + tokenizer.encode(j) + e_token
    if len(i) <= max_len and len(j) <= max_len:
        t_prompt.append(i)
        t_response.append(j)

prompt = tf.keras.preprocessing.sequence.pad_sequences(t_prompt, maxlen=max_len, padding='post')
response = tf.keras.preprocessing.sequence.pad_sequences(t_response, maxlen=max_len, padding='post')
```

Create Train and Validation Data

```
data = tf.data.Dataset.from_tensor_slices(({ 'inputs': prompt,'dec_inputs': response[:, :-1] },{'outputs': response[:, 1:]},))
data = data.cache()
data = data.shuffle(buffer_size)
data = data.batch(batch_size)
data = data.prefetch(tf.data.experimental.AUTOTUNE)
dataset_size = len(data)
train_size = int(0.8 * dataset_size)
train_dataset = data.take(train_size)
val_dataset = data.skip(train_size)
```

Multi Head Attention

```
class multi_head_attention(tf.keras.layers.Layer):

    def __init__(self, model_d, n_head, **kwargs):
        super(multi_head_attention, self).__init__(**kwargs)
        self.n_head = n_head
        self.model_d = model_d
        assert model_d % self.n_head == 0
        self.depth = model_d // self.n_head

        self.q_dense = tf.keras.layers.Dense(units=model_d)
        self.k_dense = tf.keras.layers.Dense(units=model_d)
        self.v_dense = tf.keras.layers.Dense(units=model_d)
        self.dense = tf.keras.layers.Dense(units=model_d)

    def get_config(self):
        c = super(multi_head_attention,self).get_config()
        c.update({ 'num_heads':self.n_head,'model_d':self.model_d,})
        return c
```

```
    def splitheads(self, inputs, b_size):
        inputs = tf.keras.layers.Lambda(lambda inputs:tf.reshape(inputs, shape=(b_size, -1, self.n_head, self.depth)))(inputs)
        r = tf.keras.layers.Lambda(lambda inputs: tf.transpose(inputs, perm=[0, 2, 1, 3]))(inputs)
        return r

    def call(self, inputs):
        q, k, v, m = inputs['query'], inputs['key'], inputs['value'], inputs['mask']
        b_size = tf.shape(q)[0]

        q = self.q_dense(q)
        k = self.k_dense(k)
        v = self.v_dense(v)

        q = self.splitheads(q, b_size)
        k = self.splitheads(k, b_size)
        v = self.splitheads(v, b_size)

       # scaled_dot_product_attention
        qk = tf.matmul(q, k, transpose_b=True)
        depth = tf.cast(tf.shape(k)[-1], tf.float32)
        logit = qk / tf.math.sqrt(depth)
        if m is not None:
            logit += (m * -1e9)
        a_weights = tf.nn.softmax(logit, axis=-1)
        s_attention = tf.matmul(a_weights, v)
        #scaled_attention = scaled_dot_product_attention(query, key, value, mask)
        s_attention = tf.keras.layers.Lambda(lambda s_attention: tf.transpose(s_attention, perm=[0, 2, 1, 3]))(s_attention)

        c_attention = tf.keras.layers.Lambda(lambda s_attention: tf.reshape(s_attention,(b_size, -1, self.model_d)))(s_attention)
        a = self.dense(c_attention)
        return a
```

Positional Encoding

```
class p_encoding(tf.keras.layers.Layer):

    def __init__(self, p, model_d, **kwargs):
        super(p_encoding, self).__init__(**kwargs)
        self.p = p
        self.model_d = model_d
        self.en_pos = self.en_positional(p, model_d)

    def get_config(self):
        c = super(p_encoding, self).get_config()
        c.update({'p': self.p,'model_d': self.model_d,})
```

```
        return c

    def getangle(self, p, i, model_d):
        ang = 1 / tf.pow(10000, (2 * (i // 2)) / tf.cast(model_d, tf.float32))
        return p * ang

    def en_positional(self, p, model_d):
        r_angle = self.getangle(p=tf.range(p, dtype=tf.float32)[:, tf.newaxis],i=tf.range(model_d, dtype=tf.float32)[tf.newaxis, :],model_d=model_d)
        sin = tf.math.sin(r_angle[:, 0::2])
        cos = tf.math.cos(r_angle[:, 1::2])
        en_pos = tf.concat([sin, cos], axis=-1)
        en_pos = en_pos[tf.newaxis, ...]
        t = tf.cast(en_pos, tf.float32)
        return t

    def call(self, i):
        r = i + self.en_pos[:, :tf.shape(i)[1], :]
        return r
```

Encoder Blocks

```
def EncoderLayer(units, d_model, num_heads, dropout, name="encoder_layer"):
    inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
    p_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

    at = multi_head_attention(d_model, num_heads, name="attention")({'query': inputs,'key': inputs,'value': inputs,'mask': p_mask})
    at = tf.keras.layers.Dropout(rate=dropout)(at)
    a_at = tf.keras.layers.add([inputs,at])
    at = tf.keras.layers.LayerNormalization(epsilon=1e-6)(a_at)

    op = tf.keras.layers.Dense(units=units, activation='relu')(at)
    op = tf.keras.layers.Dense(units=d_model)(op)
    op = tf.keras.layers.Dropout(rate=dropout)(op)
    a_at = tf.keras.layers.add([at,op])
    op = tf.keras.layers.LayerNormalization(epsilon=1e-6)(a_at)

    return tf.keras.Model(inputs=[inputs, p_mask], outputs=op, name=name)



def Encoder(vocab_size,num_layers, units,d_model,num_heads,dropout, name="encoder"):
    inputs = tf.keras.Input(shape=(None,), name="inputs")
    p_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

    emb = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
    emb *= tf.keras.layers.Lambda(lambda d_model: tf.math.sqrt(tf.cast(d_model, tf.float32)))(d_model)
```

```
    emb = p_encoding(vocab_size,d_model)(emb)

    op = tf.keras.layers.Dropout(rate=dropout)(emb)

    for i in range(num_layers):
        op = EncoderLayer(units=units,d_model=d_model,num_heads=num_heads,dropout=dropout,name="encoder_layer_{}".format(i),)([op, p_mask])

    return tf.keras.Model(inputs=[inputs, p_mask], outputs=op, name=name)
```

Decoder Blocks

```
def DecoderLayer(units, d_model, num_heads, dropout, name="decoder_layer"):
    inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
    e_op = tf.keras.Input(shape=(None, d_model), name="encoder_outputs")
    ahead_mask = tf.keras.Input(shape=(1, None, None), name="look_ahead_mask")
    p_mask = tf.keras.Input(shape=(1, 1, None), name='padding_mask')

    at1 = multi_head_attention(d_model, num_heads, name="attention_1")(inputs={'query': inputs,'key': inputs,'value': inputs,'mask': ahead_mask})
    add_at = tf.keras.layers.add([at1,inputs])
    at1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_at)

    at2 = multi_head_attention(d_model, num_heads, name="attention_2")(inputs={'query': at1,'key': e_op,'value': e_op,'mask': p_mask})
    at2 = tf.keras.layers.Dropout(rate=dropout)(at2)
    add_at = tf.keras.layers.add([at2,at1])
    at2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_at)

    op = tf.keras.layers.Dense(units=units, activation='relu')(at2)
    op = tf.keras.layers.Dense(units=d_model)(op)
    op = tf.keras.layers.Dropout(rate=dropout)(op)
    add_at = tf.keras.layers.add([op,at2])
    op = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_at)

    t = tf.keras.Model(inputs=[inputs, e_op, ahead_mask, p_mask],outputs=op,name=name)
    return t

def Decoder(vocab_size, num_layers, units,d_model,num_heads,dropout,name='decoder'):
    inputs = tf.keras.Input(shape=(None,), name='inputs')
    e_op = tf.keras.Input(shape=(None, d_model), name='encoder_outputs')
    ahead_mask = tf.keras.Input(shape=(1, None, None), name='look_ahead_mask')
    p_mask = tf.keras.Input(shape=(1, 1, None), name='padding_mask')

    emb = tf.keras.layers.Embedding(vocab_size, d_model)(inputs)
    emb *= tf.keras.layers.Lambda(lambda d_model: tf.math.sqrt(tf.cast(d_model, tf.float32)))(d_model)
    emb = p_encoding(vocab_size, d_model)(emb)

    op = tf.keras.layers.Dropout(rate=dropout)(emb)
```

```
    for i in range(num_layers):
        op = DecoderLayer(units=units,d_model=d_model,num_heads=num_heads,dropout=dropout,name='decoder_layer_{}'.format(i),)(inputs=[op, e_op, ahead_mask, p_m

    t = tf.keras.Model(inputs=[inputs, e_op, ahead_mask, p_mask],outputs=op,name=name)
    return t
```

Masking

```
"""
class PaddingMaskLayer(tf.keras.layers.Layer):
    def call(self, x):
        mask = tf.cast(tf.math.equal(x, 0), tf.float32)
        return mask[:, tf.newaxis, tf.newaxis, :]


class LookAheadMaskLayer(tf.keras.layers.Layer):
    def call(self, x):
        seq_len = tf.shape(x)[1]
        look_ahead_mask = 1 - tf.linalg.band_part(tf.ones((seq_len, seq_len)), -1, 0)
        padding_mask = PaddingMaskLayer()
        padding_mask = padding_mask(x)
        return tf.maximum(look_ahead_mask, padding_mask)
"""


def create_padding_mask(x):
    mask = tf.cast(tf.math.equal(x, 0), tf.float32)
    # (batch_size, 1, 1, sequence length)
    return mask[:, tf.newaxis, tf.newaxis, :]

def create_look_ahead_mask(x):
    seq_len = tf.shape(x)[1]
    look_ahead_mask = 1 - tf.linalg.band_part(tf.ones((seq_len, seq_len)), -1, 0)
    padding_mask = create_padding_mask(x)
    return tf.maximum(look_ahead_mask, padding_mask)
```

Transformer

```
inputs = tf.keras.Input(shape=(None,), name="inputs")
dec_inputs = tf.keras.Input(shape=(None,), name="dec_inputs")
```

```
padding_mask_layer = PaddingMaskLayer()
padding_mask_en = padding_mask_layer(inputs)

mask_layer = LookAheadMaskLayer()
ahead_mask = mask_layer(dec_inputs)

d_mask_layer = PaddingMaskLayer()
padding_mask_dec = d_mask_layer(inputs)

e_outputs = Encoder(vocab_size=vocab_size,num_layers=number_of_layer,units=unit,d_model=d_model,num_heads=number_of_head,dropout=dropout,)(inputs=[inputs, padd

d_outputs = Decoder( vocab_size=vocab_size,num_layers=number_of_layer,units=unit,d_model=d_model,num_heads=number_of_head,dropout=dropout,)(inputs=[dec_inputs,

op = tf.keras.layers.Dense(units=vocab_size, name="outputs")(d_outputs)

model =  tf.keras.Model(inputs=[inputs, dec_inputs], outputs=op)
```

Optimizer and Loss

```
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')
```

Train

```
#tf.keras.backend.clear_session()
model.compile(optimizer=optimizer, loss=[loss], metrics=[accuracy])


from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint_callback = ModelCheckpoint(
    filepath='/kaggle/working/model_save.h5',  # Path to save the checkpoint file
    save_best_only=True,             # Save only the best model
    save_weights_only=True,          # Save only the model weights
    monitor='val_loss',              # Monitor validation loss
    verbose=1                         # Show progress
)

model.fit(train_dataset, epochs=2, validation_data = val_dataset, callbacks=[checkpoint_callback])
#loaded_model = tf.keras.models.load_model('/kaggle/working/model_save.h5')
```

```
Epoch 1/2
1278/1278 [==============================] - ETA: 0s - loss: 1.3635 - accuracy: 0.7745
Epoch 1: val_loss improved from inf to 1.00903, saving model to /kaggle/working/model_save.h5
1278/1278 [==============================] - 193s 148ms/step - loss: 1.3635 - accuracy: 0.7745 - val_loss: 1.0090 - val_accuracy: 0.8228
Epoch 2/2
1278/1278 [==============================] - ETA: 0s - loss: 1.2915 - accuracy: 0.7696
Epoch 2: val_loss improved from 1.00903 to 0.98619, saving model to /kaggle/working/model_save.h5
1278/1278 [==============================] - 182s 142ms/step - loss: 1.2915 - accuracy: 0.7696 - val_loss: 0.9862 - val_accuracy: 0.8245
<keras.callbacks.History at 0x780edbd8eb90>
```

```
model.fit(train_dataset, epochs=2, validation_data = val_dataset)
```

```
Epoch 1/2
167/167 [==============================] - 71s 311ms/step - loss: 4.1483 - accuracy: 0.4908 - val_loss: 3.3599 - val_accuracy: 0.5259
Epoch 2/2
167/167 [==============================] - 34s 204ms/step - loss: 3.1971 - accuracy: 0.5485 - val_loss: 2.9479 - val_accuracy: 0.5744
<keras.callbacks.History at 0x7d0b57bdb2b0>
```

```
filename = "model2.h5"
tf.keras.models.save_model(model, filepath=filename, include_optimizer=False)
```

```
del model
tf.keras.backend.clear_session()
```

Perplexity

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')
total_loss = 0.0
num_batches = 0
for inputs, targets_dict in val_dataset:
    targets = targets_dict['outputs']
    predictions = model(inputs, training=False)
    batch_loss = loss_object(targets, predictions)
    average_batch_loss = tf.reduce_mean(batch_loss)
    total_loss += average_batch_loss
    num_batches += 1
average_loss = total_loss / num_batches
perplexity = tf.exp(average_loss)
a = perplexity.numpy()
print(f"Perplexity: {a}")
```

```
Perplexity: 4.207235336303711
```

Inference

```
while True:
    a = input("\nInput: ")
    if a == "exit":
        break
    s = text_preprocess(a)
    s = tf.expand_dims(s_token + tokenizer.encode(s) + e_token, axis=0)
    output = tf.expand_dims(s_token, 0)
    for i in range(max_len):
        predictions = model(inputs=[s, output], training=False)
        predictions = predictions[:, -1:, :]
        predicted_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)
        if tf.equal(predicted_id, e_token[0]):
            break
        output = tf.concat([output, predicted_id], axis=-1)

    p = tf.squeeze(output, axis=0)
    pre_prompt = tokenizer.decode([i for i in p if i < tokenizer.vocab_size])
    print('Output: {}'.format(pre_prompt))
```

```
 Input:  last few days i feel lonely but i don't know why
 Output: well , i feel lonely because you know how you feel .

 Input:  i always feel that if my family stay with me
 Output: you are always a nice person to drive me with that .

 Input:  tomorrow i have a cricket match and i love cricket more than anything else
 Output: are you ready to go to my financial start ?

 Input:  can you tell me where i will find my happiness
 Output: it will be at the right place . i love you having to go with my friends .

 Input:  I'm struggling, to be honest. It's been a really tough week for me.
 Output: ok , then .

 Input:  exit
```