transformer_chatbot

August 7, 2024

1 Minecraft Q/A Chatbot

[1]:

```
from __future__ import absolute_import, division, print_function,␣
 ↪unicode_literals

import sys

import os
import re
import numpy as np
from time import time
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds

tf.keras.utils.set_random_seed(1234)

print(f"Tensorflow version {tf.__version__}")
```

```
C:\Users\Niranjan\anaconda3\envs\py39\lib\site-packages\tqdm\auto.py:21:
TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
Tensorflow version 2.9.1
```

1.0.1 GPU initialization

Setup GPU strategy for hardware acceleration

[13]:

```
gpus = tf.config.experimental.list_physical_devices('GPU')

if gpus:
    # Create a MirroredStrategy.
    strategy = tf.distribute.MirroredStrategy()

    # Print the number of replicas.
    print(f"REPLICAS: {strategy.num_replicas_in_sync}")
else:
    print("No GPU available, falling back to CPU/GPU strategy")
    strategy = tf.distribute.get_strategy()
```

```
No GPU available, falling back to CPU/GPU strategy
```

1.1 Hyperparameters

[2]:

```
# Maximum sentence length
MAX_LENGTH = 50

# For tf.data.Dataset
#BATCH_SIZE = 64 * strategy.num_replicas_in_sync
BUFFER_SIZE = 20000

# For Transformer
NUM_LAYERS = 4
D_MODEL = 256
NUM_HEADS = 8
UNITS = 512
DROPOUT = 0.1

EPOCHS = 60
```

1.1.1 Load and preprocess data

[3]:

```
def preprocess_sentence(sentence):
    sentence = sentence.lower().strip()
    # creating a space between a word and the punctuation following it
    # eg: "he is a boy." => "he is a boy ."
    sentence = re.sub(r"([?.!,])", r" \1 ", sentence)
    sentence = re.sub(r'[" "]+', " ", sentence)
    # removing contractions
    sentence = re.sub(r"i'm", "i am", sentence)
    sentence = re.sub(r"he's", "he is", sentence)
    sentence = re.sub(r"she's", "she is", sentence)
    sentence = re.sub(r"it's", "it is", sentence)
    sentence = re.sub(r"that's", "that is", sentence)
    sentence = re.sub(r"what's", "that is", sentence)
    sentence = re.sub(r"where's", "where is", sentence)
    sentence = re.sub(r"how's", "how is", sentence)
    sentence = re.sub(r"\'ll", " will", sentence)
    sentence = re.sub(r"\'ve", " have", sentence)
    sentence = re.sub(r"\'re", " are", sentence)
    sentence = re.sub(r"\'d", " would", sentence)
    sentence = re.sub(r"\'re", " are", sentence)
    sentence = re.sub(r"won't", "will not", sentence)
    sentence = re.sub(r"can't", "cannot", sentence)
    sentence = re.sub(r"n't", " not", sentence)
    sentence = re.sub(r"n'", "ng", sentence)
    sentence = re.sub(r"'bout", "about", sentence)
    # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")
    sentence = re.sub(r"[^a-zA-Z?.!,]+", " ", sentence)
    sentence = sentence.strip()
    return sentence


from datasets import load_dataset

# Load dataset from Hugging Face
dataset = load_dataset("naklecha/minecraft-question-answer-700k")

# Access train split
train_data = dataset["train"]

# Extract questions, answers, and sources from the dataset
questions = [preprocess_sentence(example["question"]) for example in train_data]
answers = [preprocess_sentence(example["answer"]) for example in train_data]
```

[4]:

```
# Build tokenizer using tfds for both questions and answers
tokenizer = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    questions + answers, target_vocab_size=2**13
)

# Define start and end token to indicate the start and end of a sentence
START_TOKEN, END_TOKEN = [tokenizer.vocab_size], [tokenizer.vocab_size + 1]

# Vocabulary size plus start and end token
VOCAB_SIZE = tokenizer.vocab_size + 2
```

[9]:

```
# Tokenize, filter and pad sentences
def tokenize_and_filter(inputs, outputs):
    tokenized_inputs, tokenized_outputs = [], []

    for (sentence1, sentence2) in zip(inputs, outputs):
        # tokenize sentence
        sentence1 = START_TOKEN + tokenizer.encode(sentence1) + END_TOKEN
        sentence2 = START_TOKEN + tokenizer.encode(sentence2) + END_TOKEN
        # check tokenized sentence max length
        if len(sentence1) <= MAX_LENGTH and len(sentence2) <= MAX_LENGTH:
            tokenized_inputs.append(sentence1)
            tokenized_outputs.append(sentence2)

    # pad tokenized sentences
    tokenized_inputs = tf.keras.preprocessing.sequence.pad_sequences(
        tokenized_inputs, maxlen=MAX_LENGTH, padding="post"
    )
    tokenized_outputs = tf.keras.preprocessing.sequence.pad_sequences(
        tokenized_outputs, maxlen=MAX_LENGTH, padding="post"
    )

    return tokenized_inputs, tokenized_outputs


questions, answers = tokenize_and_filter(questions, answers)
```

[10]:

```
print(f"Vocab size: {VOCAB_SIZE}")
print(f"Number of samples: {len(questions)}")
```

```
Vocab size: 8219
Number of samples: 448771
```

1.1.2 Create tf.data.Dataset

[11]:

```
# decoder inputs use the previous target as input
# remove START_TOKEN from targets
dataset = tf.data.Dataset.from_tensor_slices(
    (
        {"inputs": questions, "dec_inputs": answers[:, :-1]},
        {"outputs": answers[:, 1:]},
    )
)

dataset = dataset.cache()
dataset = dataset.shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE)
dataset = dataset.prefetch(tf.data.AUTOTUNE)
```

[12]:

```
print(dataset)
```

```
<PrefetchDataset element_spec=({'inputs': TensorSpec(shape=(None, 50),
dtype=tf.int32, name=None), 'dec_inputs': TensorSpec(shape=(None, 49),
dtype=tf.int32, name=None)}, {'outputs': TensorSpec(shape=(None, 49),
dtype=tf.int32, name=None)})>
```

1.2 Attention

1.2.1 Scaled dot product Attention

[8]:

```
def scaled_dot_product_attention(query, key, value, mask):
    """Calculate the attention weights."""
    matmul_qk = tf.matmul(query, key, transpose_b=True)

    # scale matmul_qk
```

```python
    depth = tf.cast(tf.shape(key)[-1], tf.float32)
    logits = matmul_qk / tf.math.sqrt(depth)

    # add the mask to zero out padding tokens
    if mask is not None:
        logits += mask * -1e9

    # softmax is normalized on the last axis (seq_len_k)
    attention_weights = tf.nn.softmax(logits, axis=-1)

    output = tf.matmul(attention_weights, value)

    return output
```

1.2.2 Multi-head attention

[5]:

```python
class MultiHeadAttentionLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads, **kwargs):
        assert d_model % num_heads == 0
        super(MultiHeadAttentionLayer, self).__init__(**kwargs)
        self.num_heads = num_heads
        self.d_model = d_model

        self.depth = d_model // self.num_heads

        self.query_dense = tf.keras.layers.Dense(units=d_model)
        self.key_dense = tf.keras.layers.Dense(units=d_model)
        self.value_dense = tf.keras.layers.Dense(units=d_model)

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

    def get_config(self):
        config = super(MultiHeadAttentionLayer, self).get_config()
        config.update(
            {
                "num_heads": self.num_heads,
                "d_model": self.d_model,
            }
        )
        return config

    def split_heads(self, inputs, batch_size):
        inputs = tf.keras.layers.Lambda(
            lambda inputs: tf.reshape(
                inputs, shape=(batch_size, -1, self.num_heads, self.depth)
            )
        )(inputs)
        return tf.keras.layers.Lambda(
            lambda inputs: tf.transpose(inputs, perm=[0, 2, 1, 3])
        )(inputs)

    def call(self, inputs):
        query, key, value, mask = (
            inputs["query"],
            inputs["key"],
            inputs["value"],
            inputs["mask"],
        )
        batch_size = tf.shape(query)[0]

        # linear layers
        query = self.query_dense(query)
        key = self.key_dense(key)
        value = self.value_dense(value)

        # split heads
        query = self.split_heads(query, batch_size)
        key = self.split_heads(key, batch_size)
        value = self.split_heads(value, batch_size)

        # scaled dot-product attention
        scaled_attention = scaled_dot_product_attention(query, key, value, mask)
        scaled_attention = tf.keras.layers.Lambda(
            lambda scaled_attention: tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
        )(scaled_attention)

        # concatenation of heads
        concat_attention = tf.keras.layers.Lambda(
            lambda scaled_attention: tf.reshape(
                scaled_attention, (batch_size, -1, self.d_model)
            )
        )(scaled_attention)

        # final linear layer
        outputs = self.dense(concat_attention)

        return outputs
```

1.3 Transformer

1.3.1 Masking

[10]:

```
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, :]
```

[11]:

```
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)
```

1.3.2 Positional encoding

[12]:

```
class PositionalEncoding(tf.keras.layers.Layer):
    def __init__(self, position, d_model, **kwargs):
        super(PositionalEncoding, self).__init__(**kwargs)
        self.position = position
        self.d_model = d_model
        self.pos_encoding = self.positional_encoding(position, d_model)

    def get_config(self):
        config = super(PositionalEncoding, self).get_config()
        config.update(
            {
                "position": self.position,
                "d_model": self.d_model,
            }
        )
        return config

    def get_angles(self, position, i, d_model):
        angles = 1 / tf.pow(10000, (2 * (i // 2)) / tf.cast(d_model, tf.
 ↪float32))
        return position * angles

    def positional_encoding(self, position, d_model):
        angle_rads = self.get_angles(
            position=tf.range(position, dtype=tf.float32)[:, tf.newaxis],
            i=tf.range(d_model, dtype=tf.float32)[tf.newaxis, :],
            d_model=d_model,
        )
        # apply sin to even index in the array
        sines = tf.math.sin(angle_rads[:, 0::2])
        # apply cos to odd index in the array
        cosines = tf.math.cos(angle_rads[:, 1::2])

        pos_encoding = tf.concat([sines, cosines], axis=-1)
        pos_encoding = pos_encoding[tf.newaxis, ...]
        return tf.cast(pos_encoding, tf.float32)

    def call(self, inputs):
        return inputs + self.pos_encoding[:, : tf.shape(inputs)[1], :]
```

1.3.3 Encoder Layer

[13]:

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

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

    outputs = tf.keras.layers.Dense(units=units, activation="relu")(attention)
    outputs = tf.keras.layers.Dense(units=d_model)(outputs)
    outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)
    add_attention = tf.keras.layers.add([attention, outputs])
    outputs = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_attention)

    return tf.keras.Model(inputs=[inputs, padding_mask], outputs=outputs,␣
 ↪name=name)
```

1.3.4 Encoder

[14]:

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

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

    outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)
    for i in range(num_layers):
        outputs = encoder_layer(
            units=units,
            d_model=d_model,
            num_heads=num_heads,
            dropout=dropout,
            name="encoder_layer_{}".format(i),
        )([outputs, padding_mask])

    return tf.keras.Model(inputs=[inputs, padding_mask], outputs=outputs,␣
  ↪name=name)
```

1.3.5 Decoder Layer

[15]:

```
def decoder_layer(units, d_model, num_heads, dropout, name="decoder_layer"):
    inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
    enc_outputs = tf.keras.Input(shape=(None, d_model), name="encoder_outputs")
    look_ahead_mask = tf.keras.Input(shape=(1, None, None),␣
  ↪name="look_ahead_mask")
    padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

    attention1 = MultiHeadAttentionLayer(d_model, num_heads,␣
  ↪name="attention_1")(
        inputs={
            "query": inputs,
            "key": inputs,
            "value": inputs,
            "mask": look_ahead_mask,
        }
    )
    add_attention = tf.keras.layers.add([attention1, inputs])
    attention1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_attention)

    attention2 = MultiHeadAttentionLayer(d_model, num_heads,␣
  ↪name="attention_2")(
        inputs={
            "query": attention1,
            "key": enc_outputs,
            "value": enc_outputs,
            "mask": padding_mask,
        }
    )
    attention2 = tf.keras.layers.Dropout(rate=dropout)(attention2)
    add_attention = tf.keras.layers.add([attention2, attention1])
    attention2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_attention)

    outputs = tf.keras.layers.Dense(units=units, activation="relu")(attention2)
    outputs = tf.keras.layers.Dense(units=d_model)(outputs)
    outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)
    add_attention = tf.keras.layers.add([outputs, attention2])
    outputs = tf.keras.layers.LayerNormalization(epsilon=1e-6)(add_attention)

    return tf.keras.Model(
        inputs=[inputs, enc_outputs, look_ahead_mask, padding_mask],
        outputs=outputs,
        name=name,
    )
```

1.3.6 Decoder

[16]:

```
def decoder(vocab_size, num_layers, units, d_model, num_heads, dropout,␣
 ↪name="decoder"):
    inputs = tf.keras.Input(shape=(None,), name="inputs")
    enc_outputs = tf.keras.Input(shape=(None, d_model), name="encoder_outputs")
    look_ahead_mask = tf.keras.Input(shape=(1, None, None),␣
 ↪name="look_ahead_mask")
    padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")

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

    outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)

    for i in range(num_layers):
        outputs = decoder_layer(
            units=units,
            d_model=d_model,
            num_heads=num_heads,
            dropout=dropout,
            name="decoder_layer_{}".format(i),
        )(inputs=[outputs, enc_outputs, look_ahead_mask, padding_mask])

    return tf.keras.Model(
        inputs=[inputs, enc_outputs, look_ahead_mask, padding_mask],
        outputs=outputs,
        name=name,
    )
```

1.3.7 Transformer

[17]:

```
def transformer(
    vocab_size, num_layers, units, d_model, num_heads, dropout,␣
 ↪name="transformer"
):
    inputs = tf.keras.Input(shape=(None,), name="inputs")
    dec_inputs = tf.keras.Input(shape=(None,), name="dec_inputs")

    enc_padding_mask = tf.keras.layers.Lambda(
        create_padding_mask, output_shape=(1, 1, None), name="enc_padding_mask"
    )(inputs)
    # mask the future tokens for decoder inputs at the 1st attention block
    look_ahead_mask = tf.keras.layers.Lambda(
        create_look_ahead_mask, output_shape=(1, None, None),␣
 ↪name="look_ahead_mask"
    )(dec_inputs)
    # mask the encoder outputs for the 2nd attention block
    dec_padding_mask = tf.keras.layers.Lambda(
        create_padding_mask, output_shape=(1, 1, None), name="dec_padding_mask"
    )(inputs)

    enc_outputs = encoder(
        vocab_size=vocab_size,
        num_layers=num_layers,
        units=units,
        d_model=d_model,
        num_heads=num_heads,
        dropout=dropout,
    )(inputs=[inputs, enc_padding_mask])

    dec_outputs = decoder(
        vocab_size=vocab_size,
        num_layers=num_layers,
        units=units,
        d_model=d_model,
        num_heads=num_heads,
        dropout=dropout,
    )(inputs=[dec_inputs, enc_outputs, look_ahead_mask, dec_padding_mask])

    outputs = tf.keras.layers.Dense(units=vocab_size,␣
 ↪name="outputs")(dec_outputs)

    return tf.keras.Model(inputs=[inputs, dec_inputs], outputs=outputs,␣
 ↪name=name)
```

1.4 Train model

1.4.1 Loss function

[29]:

```
def loss_function(y_true, y_pred):
    y_true = tf.reshape(y_true, shape=(-1, MAX_LENGTH - 1))

    loss = tf.keras.losses.SparseCategoricalCrossentropy(
        from_logits=True, reduction="none"
    )(y_true, y_pred)

    mask = tf.cast(tf.not_equal(y_true, 0), tf.float32)
    loss = tf.multiply(loss, mask)

    return tf.reduce_mean(loss)
```

1.4.2 Custom learning rate

[30]:

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

        self.d_model = tf.constant(d_model, dtype=tf.float32)
        self.warmup_steps = warmup_steps

    def get_config(self):
        return {"d_model": self.d_model, "warmup_steps": self.warmup_steps}

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

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

1.4.3 Initialize and compile model

[27]:

```
# clear backend
tf.keras.backend.clear_session()

learning_rate = CustomSchedule(D_MODEL)

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

```python
def accuracy(y_true, y_pred):
    # ensure labels have shape (batch_size, MAX_LENGTH - 1)
    y_true = tf.reshape(y_true, shape=(-1, MAX_LENGTH - 1))
    return tf.keras.metrics.sparse_categorical_accuracy(y_true, y_pred)


# initialize and compile model within strategy scope
with strategy.scope():
    model = transformer(
        vocab_size=VOCAB_SIZE,
        num_layers=NUM_LAYERS,
        units=UNITS,
        d_model=D_MODEL,
        num_heads=NUM_HEADS,
        dropout=DROPOUT,
    )

    model.compile(optimizer=optimizer, loss=loss_function, metrics=[accuracy])

model.summary()
```

```
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

Model: "transformer"

--------------------------------------------------------------------------------
------------------
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 inputs (InputLayer)            [(None, None)]       0            []
 dec_inputs (InputLayer)        [(None, None)]       0           []

 enc_padding_mask (Lambda)      (None, 1, 1, None)   0
['inputs[0][0]']

 encoder (Functional)           (None, None, 256)    4212480
['inputs[0][0]',
'enc_padding_mask[0][0]']

 look_ahead_mask (Lambda)       (None, 1, None, Non  0
['dec_inputs[0][0]']

                                e)

 dec_padding_mask (Lambda)      (None, 1, 1, None)   0
['inputs[0][0]']

 decoder (Functional)           (None, None, 256)    5267200
['dec_inputs[0][0]',
'encoder[0][0]',
'look_ahead_mask[0][0]',
'dec_padding_mask[0][0]']

 outputs (Dense)                (None, None, 8219)   2112283
['decoder[0][0]']

================================================================================
==================
Total params: 11,591,963
Trainable params: 11,591,963
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

1.4.4 Fit model

[28]:

```
model.fit(dataset, epochs=EPOCHS)
```

```
Epoch 1/60
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).
```

```
INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

INFO:tensorflow:Reduce to /job:localhost/replica:0/task:0/device:CPU:0 then
broadcast to ('/job:localhost/replica:0/task:0/device:CPU:0',).

7013/7013 [==============================] - 488s 68ms/step - loss: 1.7241 -
accuracy: 0.2264
Epoch 2/60
7013/7013 [==============================] - 397s 57ms/step - loss: 1.0192 -
accuracy: 0.3080
Epoch 3/60
7013/7013 [==============================] - 467s 67ms/step - loss: 0.8952 -
accuracy: 0.3285
Epoch 4/60
7013/7013 [==============================] - 457s 65ms/step - loss: 0.8326 -
accuracy: 0.3391
Epoch 5/60
7013/7013 [==============================] - 440s 63ms/step - loss: 0.7931 -
accuracy: 0.3463
Epoch 6/60
7013/7013 [==============================] - 357s 51ms/step - loss: 0.7649 -
accuracy: 0.3511
Epoch 7/60
7013/7013 [==============================] - 373s 53ms/step - loss: 0.7433 -
accuracy: 0.3551
Epoch 8/60
7013/7013 [==============================] - 385s 55ms/step - loss: 0.7260 -
accuracy: 0.3582
Epoch 9/60
7013/7013 [==============================] - 383s 55ms/step - loss: 0.7120 -
accuracy: 0.3607
Epoch 10/60
7013/7013 [==============================] - 401s 57ms/step - loss: 0.6998 -
```

```
accuracy: 0.3630
Epoch 11/60
7013/7013 [==============================] - 399s 57ms/step - loss: 0.6893 -
accuracy: 0.3649
Epoch 12/60
7013/7013 [==============================] - 404s 58ms/step - loss: 0.6805 -
accuracy: 0.3666
Epoch 13/60
7013/7013 [==============================] - 382s 54ms/step - loss: 0.6725 -
accuracy: 0.3680
Epoch 14/60
7013/7013 [==============================] - 385s 55ms/step - loss: 0.6653 -
accuracy: 0.3694
Epoch 15/60
7013/7013 [==============================] - 416s 59ms/step - loss: 0.6588 -
accuracy: 0.3706
Epoch 16/60
7013/7013 [==============================] - 489s 70ms/step - loss: 0.6531 -
accuracy: 0.3717
Epoch 17/60
7013/7013 [==============================] - 406s 58ms/step - loss: 0.6475 -
accuracy: 0.3727
Epoch 18/60
7013/7013 [==============================] - 306s 44ms/step - loss: 0.6425 -
accuracy: 0.3737
Epoch 19/60
7013/7013 [==============================] - 302s 43ms/step - loss: 0.6381 -
accuracy: 0.3745
Epoch 20/60
7013/7013 [==============================] - 303s 43ms/step - loss: 0.6338 -
accuracy: 0.3753
Epoch 21/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.6296 -
accuracy: 0.3761
Epoch 22/60
7013/7013 [==============================] - 304s 43ms/step - loss: 0.6258 -
accuracy: 0.3768
Epoch 23/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.6224 -
accuracy: 0.3775
Epoch 24/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.6192 -
accuracy: 0.3781
Epoch 25/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.6158 -
accuracy: 0.3788
Epoch 26/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.6129 -
```

```
accuracy: 0.3793
Epoch 27/60
7013/7013 [==============================] - 304s 43ms/step - loss: 0.6099 -
accuracy: 0.3799
Epoch 28/60
7013/7013 [==============================] - 304s 43ms/step - loss: 0.6074 -
accuracy: 0.3803
Epoch 29/60
7013/7013 [==============================] - 305s 44ms/step - loss: 0.6049 -
accuracy: 0.3808
Epoch 30/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.6022 -
accuracy: 0.3813
Epoch 31/60
7013/7013 [==============================] - 304s 43ms/step - loss: 0.5997 -
accuracy: 0.3818
Epoch 32/60
7013/7013 [==============================] - 304s 43ms/step - loss: 0.5976 -
accuracy: 0.3822
Epoch 33/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.5955 -
accuracy: 0.3826
Epoch 34/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.5933 -
accuracy: 0.3831
Epoch 35/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.5914 -
accuracy: 0.3835
Epoch 36/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.5895 -
accuracy: 0.3837
Epoch 37/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.5876 -
accuracy: 0.3841
Epoch 38/60
7013/7013 [==============================] - 305s 43ms/step - loss: 0.5858 -
accuracy: 0.3845
Epoch 39/60
7013/7013 [==============================] - 304s 43ms/step - loss: 0.5840 -
accuracy: 0.3848
Epoch 40/60
7013/7013 [==============================] - 301s 43ms/step - loss: 0.5824 -
accuracy: 0.3851
Epoch 41/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5809 -
accuracy: 0.3854
Epoch 42/60
7013/7013 [==============================] - 290s 41ms/step - loss: 0.5792 -
```

```
accuracy: 0.3857
Epoch 43/60
7013/7013 [==============================] - 290s 41ms/step - loss: 0.5774 -
accuracy: 0.3861
Epoch 44/60
7013/7013 [==============================] - 290s 41ms/step - loss: 0.5760 -
accuracy: 0.3863
Epoch 45/60
7013/7013 [==============================] - 292s 42ms/step - loss: 0.5746 -
accuracy: 0.3865
Epoch 46/60
7013/7013 [==============================] - 291s 42ms/step - loss: 0.5731 -
accuracy: 0.3869
Epoch 47/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5714 -
accuracy: 0.3872
Epoch 48/60
7013/7013 [==============================] - 291s 42ms/step - loss: 0.5702 -
accuracy: 0.3874
Epoch 49/60
7013/7013 [==============================] - 291s 42ms/step - loss: 0.5688 -
accuracy: 0.3877
Epoch 50/60
7013/7013 [==============================] - 292s 42ms/step - loss: 0.5676 -
accuracy: 0.3879
Epoch 51/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5663 -
accuracy: 0.3881
Epoch 52/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5648 -
accuracy: 0.3884
Epoch 53/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5638 -
accuracy: 0.3886
Epoch 54/60
7013/7013 [==============================] - 291s 42ms/step - loss: 0.5625 -
accuracy: 0.3889
Epoch 55/60
7013/7013 [==============================] - 291s 42ms/step - loss: 0.5611 -
accuracy: 0.3891
Epoch 56/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5601 -
accuracy: 0.3892
Epoch 57/60
7013/7013 [==============================] - 292s 42ms/step - loss: 0.5590 -
accuracy: 0.3895
Epoch 58/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5579 -
```

```
accuracy: 0.3896
Epoch 59/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5568 -
accuracy: 0.3899
Epoch 60/60
7013/7013 [==============================] - 291s 41ms/step - loss: 0.5559 -
accuracy: 0.3900
```

[28]:

```
<keras.callbacks.History at 0x28a5ddbfa60>
```

1.4.5 Save and load model

[29]:

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

[30]:

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

[18]:

```python
#import tensorflow as tf
filename = "model.h5"
model = tf.keras.models.load_model(
    filename,
    custom_objects={
        "PositionalEncoding": PositionalEncoding,
        "MultiHeadAttentionLayer": MultiHeadAttentionLayer,
    },
    compile=False,
)
```

1.5 Evaluate and predict

[19]:

```python
def evaluate(sentence):
    sentence = preprocess_sentence(sentence)

    sentence = tf.expand_dims(
        START_TOKEN + tokenizer.encode(sentence) + END_TOKEN, axis=0
    )

    output = tf.expand_dims(START_TOKEN, 0)

    for i in range(MAX_LENGTH):
        predictions = model(inputs=[sentence, output], training=False)

        # select the last word from the seq_len dimension
        predictions = predictions[:, -1:, :]
        predicted_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)

        # return the result if the predicted_id is equal to the end token
        if tf.equal(predicted_id, END_TOKEN[0]):
            break

        # concatenated the predicted_id to the output which is given to the␣
 ↪decoder
        # as its input.
        output = tf.concat([output, predicted_id], axis=-1)

    return tf.squeeze(output, axis=0)


def predict(sentence):
    prediction = evaluate(sentence)
    predicted_sentence = tokenizer.decode(
        [i for i in prediction if i < tokenizer.vocab_size]
    )
    return predicted_sentence
```

[20]:
```
import random
import requests
import nltk
import os

end_prompt = "Or type 'leave' to end then chat"

start_prompts = [
    "Greetings, fellow Minecraft enthusiast! What's your handle? And where do␣
 ↪you call home in the real world?",
    "Hi there! This bot is here to talk all things Minecraft. What should I␣
 ↪call you? Where are you from?",
    "Greetings! I'm here to discuss Minecraft with you. What's your name? Where␣
 ↪are you from?",
    "Hey there! Ready to dive into the world of Minecraft? What can I call you?␣
 ↪Where do you hail from?",
    "Hi! Are you ready to chat about Minecraft? What's your name? Where are you␣
 ↪from?"
]

query_prompts = [
    "What specific aspect of Minecraft are you curious about today?",
    "Interested in learning more about a particular Minecraft update or feature?
 ↪ Let me know!",
    "Is there a Minecraft fact or trivia you're unclear about? Feel free to ask!
 ↪",
    "Looking to discuss Minecraft's gameplay or mechanics? Just tell me what␣
 ↪you'd like to know.",
    "Want to delve into the details of Minecraft's world or development? Ask␣
  ↪away!"
]

sorry_prompts = [
        "I'm sorry, I couldn't find a relevant response.",
        "Apologies, but I couldn't find an answer.",
        "I'm sorry, I couldn't locate the information you're looking for.",
        "Unfortunately, I couldn't find the answer to your question.",
        "Sorry, I couldn't find any information related to your query.",
        "I'm afraid I don't have an answer for that at the moment.",
        "Sorry, I couldn't find any relevant information.",
        "Unfortunately, I'm unable to provide an answer at this time.",
        "I'm sorry, but I couldn't find any relevant information on that topic.
  ↪",
        "Apologies, I couldn't find what you were looking for."
    ]

thank_you_prompts = [
    "Thank you for using the bot! Have a great day!",
    "Thanks for chatting with me! Take care!",
    "Appreciate your interaction with the bot! If you'd like to chat again,␣
  ↪you're always welcome.",
    "Thank you for your questions! If you ever want to chat again, don't␣
  ↪hesitate to return.",
    "Thanks for using the bot! Have a wonderful day ahead!",
]

ask_list = [
    "Is there anything else I can assist you with regarding Minecraft?",
    "Do you have any other questions about Minecraft?",
    "Would you like to inquire about something else related to Minecraft?",
    "Is there any other aspect of Minecraft you're curious about?",
    "Do you need information on another Minecraft update or feature?",
    "Are there additional Minecraft-related queries you'd like to explore?",
    "Is there something else about Minecraft's gameplay or mechanics you'd like␣
  ↪to know?",
    "Are there any further questions you have about Minecraft?"
]
```

[21]:

```
def extract_personal_info(user_response):
    # Tokenize the user's response into words
    words = nltk.word_tokenize(user_response)

    # Perform Named Entity Recognition (NER)
    ne_tags = nltk.pos_tag(words)
    named_entities = nltk.ne_chunk(ne_tags)
    # Initialize variables to store extracted information
    personal_info = {'Location': None, 'Organization': None}

    # Extract personal information from the named entities
    for entity in named_entities:
        if isinstance(entity, nltk.tree.Tree):
            if entity.label() == 'GPE' and personal_info['Location'] is None:
                # Extract location if not already extracted
                personal_info['Location'] = ' '.join([leaf[0] for leaf in␣
↪entity.leaves()])
            elif entity.label() == 'ORGANIZATION' and␣
↪personal_info['Organization'] is None:
                # Extract organization if not already extracted
                personal_info['Organization'] = ' '.join([leaf[0] for leaf in␣
↪entity.leaves()])

    return personal_info
```

[22]:

```
def process_name(response):
    # Tokenize the user's response into words
    words = nltk.word_tokenize(response)

    # Perform Named Entity Recognition (NER)
    ne_tags = nltk.pos_tag(words)
    named_entities = nltk.ne_chunk(ne_tags)

    # Extract names from the named entities
    names = []
    for entity in named_entities:
        if isinstance(entity, nltk.tree.Tree) and entity.label() == 'PERSON':
            names.append(' '.join([leaf[0] for leaf in entity.leaves()]))

    # Return extracted names
    if names:
        if len(names) < 2:
            return names[0]
        else:
            return names[0] + ' ' + names[1]
    else:
        return None
```

[23]:

```
USER_DATA_DIR = "user_data"
def check_user_model(name):
    user_file_path = os.path.join(USER_DATA_DIR, f"{name.lower()}.txt")
    if os.path.exists(user_file_path):
        return True
    return False

def load_user_model(name):
    user_file_path = os.path.join(USER_DATA_DIR, f"{name.lower()}.txt")
    if os.path.exists(user_file_path):
        with open(user_file_path, 'r') as user_file:
            lines = user_file.readlines()
            user_model = {
                "name": name,
                "personal_info": {},
                "likes": [],
                "dislikes": []
            }
            for line in lines[1:]:
                key, value = line.strip().split(':', 1)
                value = value.replace("[", "").replace("]", "").replace("'", "")
                value = value.strip()
                if key == "likes" or key == "dislikes":
                    user_model[key].append(value)
                else:
                    user_model["personal_info"][key] = value
        return user_model
    else:
        return {"name": name, "personal_info": {}, "likes": [], "dislikes": []}

def save_user_model(user_model):
    name = user_model["name"]
    user_file_path = os.path.join(USER_DATA_DIR, f"{name.lower()}.txt")
    try:
        os.makedirs(USER_DATA_DIR, exist_ok=True)
        with open(user_file_path, 'w') as user_file:
            user_file.write(f"Name: {user_model['name']}\n")
            for key, value in user_model["personal_info"].items():
                user_file.write(f"{key}: {value}\n")
            for like in user_model["likes"]:
                user_file.write(f"likes: {like}\n")
            for dislike in user_model["dislikes"]:
                user_file.write(f"dislikes: {dislike}\n")
    except Exception as e:
        return
```

[24]:

```
import spacy

def find_subject(prompt):
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(prompt)
    dobj_toks = [tok for tok in doc if tok.dep_ == "dobj"]
    if dobj_toks:
        return dobj_toks[0]
    else:
        pobj_toks = [tok for tok in doc if tok.dep_ == "pobj"]
        if pobj_toks:
            return pobj_toks[0]
        else:
            return None
```

[25]:

```
from nltk.sentiment import SentimentIntensityAnalyzer

def process_sentiment(user_model, term, user_response):

    senti_analyzer = SentimentIntensityAnalyzer()

    # Perform sentiment analysis on the user response
    sentiment_score = senti_analyzer.polarity_scores(user_response)['compound']

    # Update user model based on sentiment score
    if sentiment_score > 0.05:  # Positive sentiment
        if not term in user_model['likes']:
            user_model['likes'].append(term)
        if term in user_model['dislikes']:
            user_model['dislikes'].remove(term)
    elif sentiment_score < -0.05:  # Negative sentiment
        if not term in user_model['dislikes']:
            user_model['dislikes'].append(term)
        if term in user_model['likes']:
            user_model['likes'].remove(term)

    return user_model
```

[26]:

```
def main():

    print(random.choice(start_prompts))
    name = input("You: ")
    personal_info = extract_personal_info(name)
    name = process_name(name)

    while name == None:
        name = print("Sorry, Please Give your name again")
        name = input("You: ")
        personal_info = extract_personal_info(name)
        name = process_name(name)

    if check_user_model(name) == True:
        print(f"Welcome back, {name}!")
```

```
  else:
      print(f"Nice to meet you, {name}!")

  user = load_user_model(name)

  if personal_info:
      if personal_info['Location']:
          user['personal_info']['Location'] = personal_info['Location']
      if personal_info['Organization']:
          user['personal_info']['Organization'] =␣
↪personal_info['Organization']
  if user['likes']:
      likes = ", ".join(user['likes'])
      print("I remember that you like the topics",likes)
  print(random.choice(query_prompts))
  # Loop to continuously prompt for input and process responses

  user_input = input("You: ")
  while not user_input:
      user_input = input("You: ")

  term = find_subject(user_input)
  response = predict(user_input)
  if response:
    print(response)
  else:
    print(random.choice(sorry_prompts))
  if term:
    print(f"Do you like {term}?")
    sentiment_response = input("You: ")
    while not sentiment_response:
        sentiment_response = input("You: ")
    process_sentiment(user, term, sentiment_response)

  while True:
      print(random.choice(ask_list), end_prompt)
      user_input = input("You: ")
      while not user_input:
          user_input = input("You: ")

      if user_input.lower() in ["exit", "quit", "leave"]:
          print(random.choice(thank_you_prompts))
          save_user_model(user)
          break
      else:
          term = find_subject(user_input)
          response = predict(user_input)
```

```python
            if response:
              print(response)
            else:
              print(random.choice(sorry_prompts))
            if term:
              print(f"Do you like {term}?")
              sentiment_response = input("You: ")
              while not sentiment_response:
                  sentiment_response = input("You: ")
              process_sentiment(user, term, sentiment_response)

if __name__ == "__main__":
    main()
```

```
Hey there! Ready to dive into the world of Minecraft? What can I call you? Where
do you hail from?

You:  I am from minesota

Sorry, Please Give your name again

You:  My name is Niranjan

Welcome back, Niranjan!
I remember that you like the topics health
What specific aspect of Minecraft are you curious about today?

You:  how to sleep in minecraft

a bed can be used as a light source in minecraft , allowing players to sleep in
the end or the game s terrain .
Is there anything else I can assist you with regarding Minecraft? Or type
'leave' to end then chat

You:  leave

Thank you for using the bot! Have a great day!
```

[ ]:

```

```