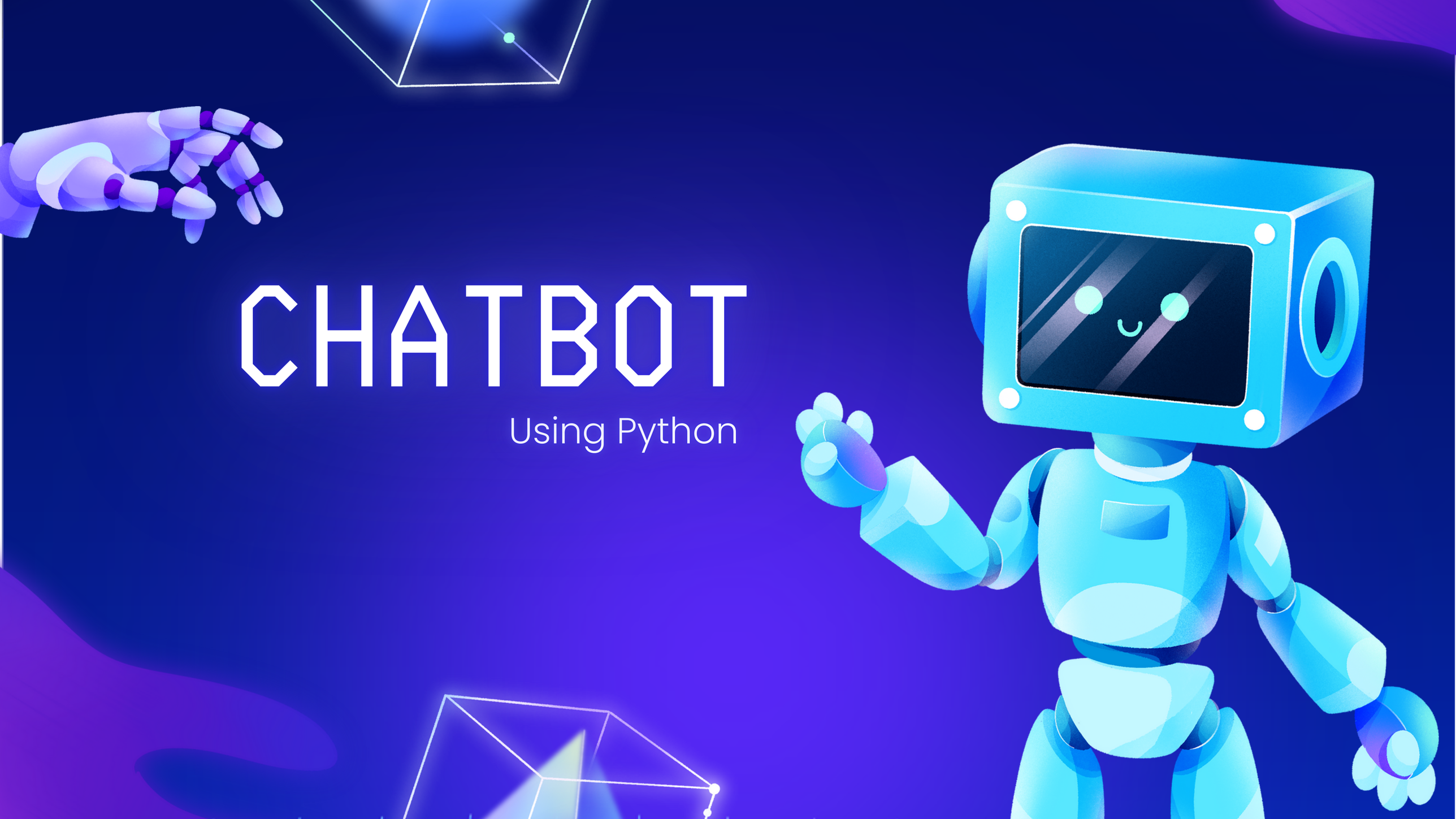
CHATBOT

Using Python



Importing libraries

```
#model
import tensorflow as tf
from sklearn.model_selection import train_test_split
#nlp processing
import unicodedata
import re
import numpy as np
import warnings
warnings.filterwarnings('ignore')
/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 (detected version 1.23.5
      warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Data preprocessing

The basic text processing in NLP are:
1. Sentence Segmentation
2. Normalization
3. Tokenization

Segmentation

formatting data to be in a question answer format

```
#reading data
data=open('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt','r'). read()
#paried list of question and corresponding answer QA_list= [QA.split('\t') for QA in data.split('\n')]
print(QA_list[:5]) [['hi, how are you doing?', "i'm fine. how about yourself?"], ["i'm fine. how about yourself?", "i'm pretty good. thanks for asking."], ["i'm pretty good. thanks for asking.", 'no problem. so how have you been?'], ['no problem. so how have you been?', "i've been great. what
```

```
#reading data
data=open('/kaggle/input/simple-dialogs-for-chatbot/dialogs.txt','r').read()
#paried list of question and corresponding answer
QA_list=[QA.split('\t') for QA in data.split('\n')]
print(QA_list[:5])
[['hi, how are you doing?', "i'm fine. how about yourself?"], ["i'm fine. how about yourself?", "i'm pretty good. thanks for asking."], ["i'm pretty good. thanks for asking.", 'no problem. so how have you been?'], ['no problem. so how have you been?', "i've been great. what about you?"], ["i've been great. what about you?", "i've been good. i'm in school right now."]] questions=[row[0] for row in QA_list] answers=[row[1] for row in QA_list] print(questions[0:5]) print(answers[0:5]) ['hi, how are you doing?', "i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no problem. so how have you been?', "i've been great. what about you?"] ["i'm fine. how about yourself?", "i'm pretty good. thanks for asking.", 'no problem. so how have you been?', "i've been great. what about you?", "i've been good. i'm in school right now."]
```

Normalization

To reduce its randomness, bringing it closer to a predefined “standard”

```
def remove_diacritic(text):
return ''.join(char for char in unicodedata.normalize('NFD',text)
if unicodedata.category(char) !='Mn')
def preprocessing(text):
#Case folding and removing extra whitespaces
text=remove_diacritic(text.lower().strip())
#Ensuring punctuation marks to be treated as tokens
text=re.sub(r"([?.!,¿])", r" \1 ", text)
#Removing redundant spaces
text= re.sub(r'[" "]+', " ", text)
#Removing non alphabetic characters
text=re.sub(r"[^a-zA-Z?.!,¿]+", " ", text)
text=text.strip()
#Indicating the start and end of each sentence
text='<start> ' + text + ' <end>'
return text
preprocessed_questions=[preprocessing(sen) for sen in questions]
preprocessed_answers=[preprocessing(sen) for sen in answers]
print(preprocessed_questions[0])
print(preprocessed_answers[0])
 <start>hi , how are you doing ?<end>
<start>i m fine . how about yourself ?<end>
```

Tokenization

```
def tokenize(lang):
lang_tokenizer = tf.keras.preprocessing.text.Tokenizer( filters='')
#build vocabulary on unique words
lang_tokenizer.fit_on_texts(lang)
return lang_tokenizer
```

Word Embedding

representing words in form of real-valued vetors

```
def vectorization(lang_tokenizer,lang):
#word embedding for training the neural network
tensor = lang_tokenizer.texts_to_sequences(lang)
tensor=tf.keras.preprocessing.sequence.pad_sequences(tensor,
padding='post')
return tensor
```

Buliding Model Architecture

Encoder

```
class Encoder(tf.keras.Model):
def init (self, vocab_size, embedding_dim, enc_units, batch_sz):
super(Encoder, self). init ()
self.batch_sz = batch_sz
self.enc_units = enc_units
self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
self.gru = tf.keras.layers.GRU(self.enc_units, return_sequences=True,
return_state=True, recurrent_initializer='glorot_uniform')
def call(self, x, hidden):
 x = self.embedding(x)
output, state = self.gru(x, initial_state = hidden)
return output, state
def initialize_hidden_state(self): return tf.zeros((self.batch_sz,
self.enc_units)) encoder = Encoder(vocab_inp_size, embedding_dim, units,
BATCH_SIZE)
```

```
.# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch,
sample_hidden)
print ('Encoder output shape: (batch size, sequence length, units)
{}'.format(sample_output.shape))
print ('Encoder Hidden state shape: (batch size, units)
{}'.format(sample_hidden.shape))
Encoder output shape: (batch size, sequence length, units) (64, 24, 1024)
Encoder Hidden state shape: (batch size, units) (64, 1024)
```

Decoder

```
class Decoder(tf.keras.Model):
def init (self, vocab_size, embedding_dim, dec_units, batch_sz):
super(Decoder, self). init ()
self.batch_sz = batch_sz
self.dec_units = dec_units
self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
```

```
self.gru = tf.keras.layers.GRU(self.dec_units, return_sequences=True,
return_state=True, recurrent_initializer='glorot_uniform')
self.fc = tf.keras.layers.Dense(vocab_size)
# used for attention
self.attention = BahdanauAttention(self.dec_units)
def call(self, x, hidden, enc_output):
# enc_output shape == (batch_size, max_length, hidden_size)
context_vector, attention_weights = self.attention(hidden, enc_output)
# x shape after passing through embedding == (batch_size, 1, embedding_dim)
x = self.embedding(x)
# x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
# passing the concatenated vector to the GRU output, state = self.gru(x)
# output shape == (batch_size * 1, hidden_size)
output = tf.reshape(output, (-1, output.shape[2]))
# output shape == (batch_size, vocab)
x = self.fc(output) return x, state, attention_weights
```

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
sample_hidden, sample_output)
print ('Decoder output shape: (batch_size, vocab size)
{}'.format(sample_decoder_output.shape))
```

```
Decoder output shape: (batch_size, vocab size) (64, 2349)
```

THANK YOU