Faculty of Computer Science and Engineering

GIK Institute

Phase 3: Project Report
CS351

Subject: Artificial Intelligence

Project Title: Developing a Sports Chatbot Language Model.

Instructor: Usman Haider

Group Members:

1. Mahnoor Maleeka (2020217)
2. Saad Khan (2020414)
3. Shaheer Asif (2020353)

Abstract:

This report presents the development of a sports chatbot utilizing text-based generation strategies, including tokenization, bag of words, RNN (Recurrent Neural Networks) and LSTM (Long Short-Term Memory) and BiLSTM (Bidirectional Long Short-Term Memory). The chatbot is intended to give constant data and take part in discussions related to sports. The execution includes coordinating the chatbot into a site for simple openness. The project contributes to the field of natural language processing and showcases the effectiveness of Bidirectional LSTM models and Bag-of-Words implementation in chatbot development. Further improvements and extensions can be explored to enhance the chatbot's capabilities and cater to a broader range of sports-related queries.

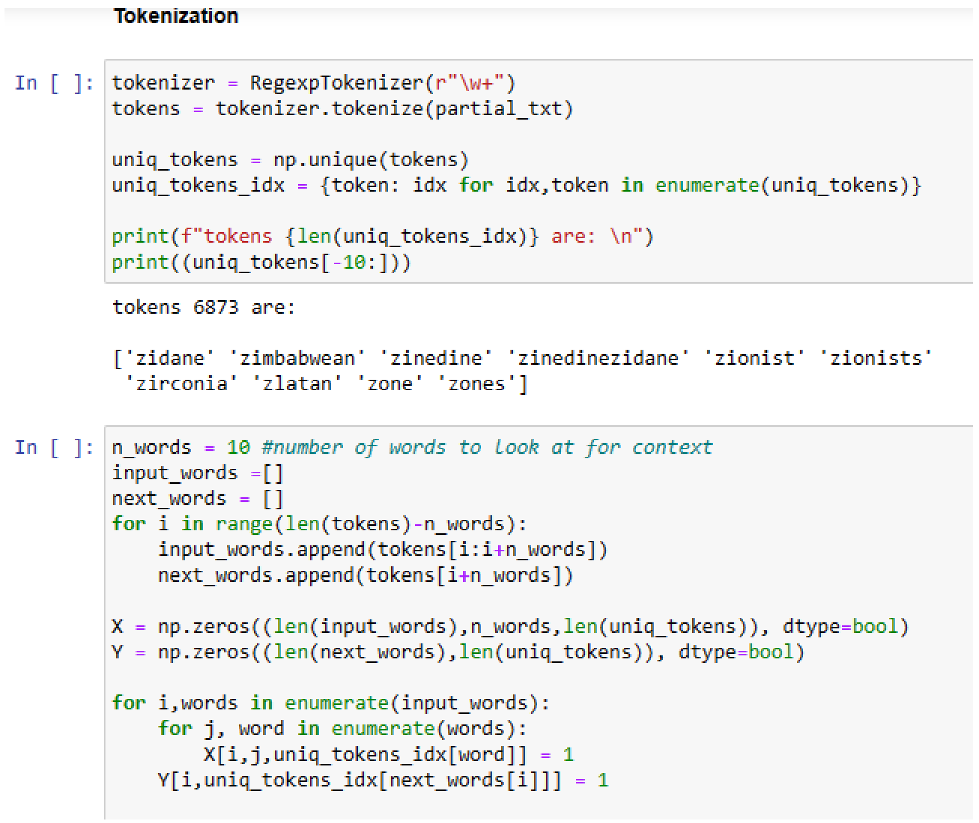
Introduction:

In recent years, chatbots have gained popularity due to their ability to simulate human-like conversations and provide information on various topics. Sports enthusiasts often seek up-to-date information, statistics, and engage in discussions related to their favorite sports. Developing a sports chatbot can enhance user experience by providing a personalized and interactive platform for sports-related conversations. This report presents the development of a sports chatbot using text-based generation techniques and its implementation on a website.

In recent years, chatbots have acquired fame because of their capacity to recreate human-like discussions and give data on different points. Sports enthusiasts frequently look for modern data, measurements, and participate in conversations connected with their favorite games. Developing a sports chatbot can improve client experience by giving a customized and intelligent stage for sports-related discussions. This report presents the improvement of a sports chatbot utilizing text-based generation procedures and its execution on a site.

Methodology:

1) Data Collection:
   The first step of our process was to gather relative sports data. The information was gathered from publicly available datasets on Kaggle. The data is basically scrapped from twitter.
2) Data Pre-Processing:
   Once the information is gathered, it should be preprocessed to guarantee consistency and work on the nature of the chatbot's reactions. Information preprocessing includes cleaning the text information by eliminating superfluous characters and taking care of any irregularities in the information design using the regix library.
3) Tokenization:
   The most important phase in building the chatbot includes tokenization, which is the most common way of parting the information text into more modest units called tokens. Tokenization helps in putting together and breaking down the text information successfully.



Tokenization

```
In [ ]: tokenizer = RegexpTokenizer(r"\w+")
        tokens = tokenizer.tokenize(partial_txt)

        uniq_tokens = np.unique(tokens)
        uniq_tokens_idx = {token: idx for idx,token in enumerate(uniq_tokens)}

        print(f"tokens {len(uniq_tokens_idx)} are: \n")
        print((uniq_tokens[-10:]))

        tokens 6873 are:

        ['zidane' 'zimbabwean' 'zinedine' 'zinedinezidane' 'zionist' 'zionists'
         'zirconia' 'zlatan' 'zone' 'zones']
```

```
In [ ]: n_words = 10 #number of words to look at for context
        input_words =[]
        next_words = []
        for i in range(len(tokens)-n_words):
            input_words.append(tokens[i:i+n_words])
            next_words.append(tokens[i+n_words])

        X = np.zeros((len(input_words),n_words,len(uniq_tokens)), dtype=bool)
        Y = np.zeros((len(next_words),len(uniq_tokens)), dtype=bool)

        for i,words in enumerate(input_words):
            for j, word in enumerate(words):
                X[i,j,uniq_tokens_idx[word]] = 1
            Y[i,uniq_tokens_idx[next_words[i]]] = 1
```

4) Building a Vocabulary:

Utilizing the tokenized information, a vocabulary is made, which is basically a list of unique tokens present in the dataset. The vocabulary fills in as a source of perspective for the chatbot to plan tokens to their comparing mathematical representations.

5) Bag of Words:

Bag of Words is a procedure utilized for text order and examination. It includes addressing text information as a sack of individual words, ignoring sentence structure and word request. This approach supports understanding and creating important reactions in the chatbot.

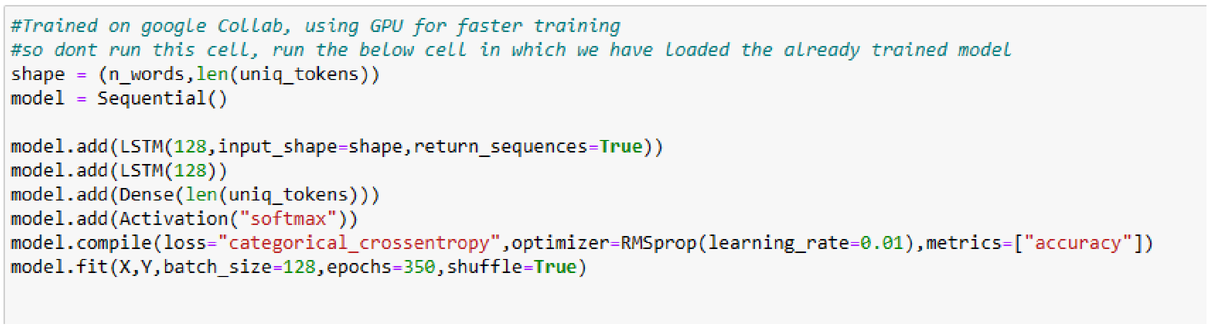
6) RNN (Recurrent Neural Networks):

Recurrent Neural Networks are a class of artificial neural networks that can process sequential data. In the case of the sports chatbot, RNN is used to generate responses based on the context provided by the user.

7) LSTM (Long Short-Term Memory):

LSTM is a type of RNN architecture that addresses the vanishing gradient problem and allows the network to retain long-term dependencies. LSTM helps the chatbot understand and respond to complex queries by maintaining context over extended conversations.

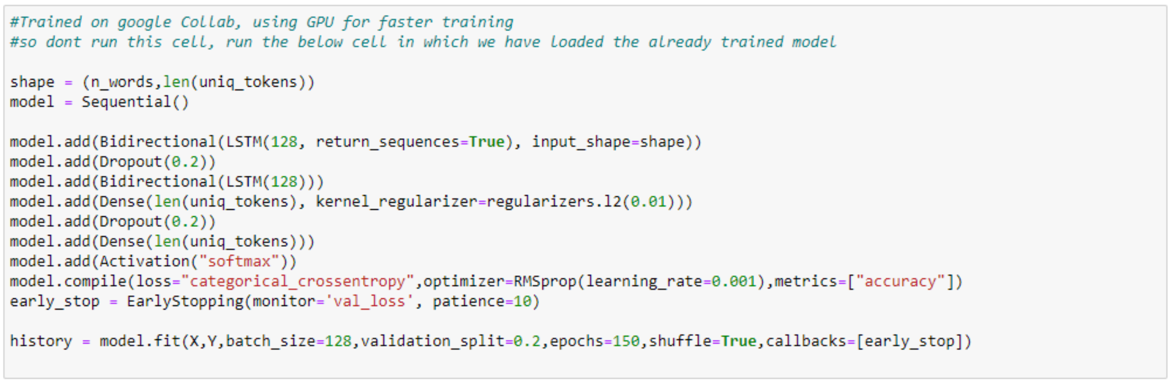
Training of the LSTM NN



```
#Trained on google Collab, using GPU for faster training
#so dont run this cell, run the below cell in which we have loaded the already trained model
shape = (n_words,len(uniq_tokens))
model = Sequential()

model.add(LSTM(128,input_shape=shape,return_sequences=True))
model.add(LSTM(128))
model.add(Dense(len(uniq_tokens)))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy",optimizer=RMSprop(learning_rate=0.01),metrics=["accuracy"])
model.fit(X,Y,batch_size=128,epochs=350,shuffle=True)
```

8) BiLSTM (Bidirectional Long Short-Term Memory):

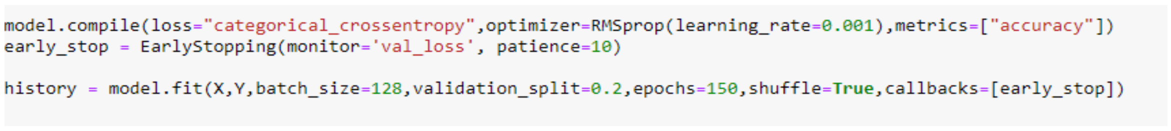
Bidirectional LSTMs (Long Short-Term Memory) are a type of recurrent neural network (RNN) that addresses the limitation of standard LSTMs by considering both past and future context when making predictions. Unlike traditional LSTMs that process sequential data in one direction, Bidirectional LSTMs process the input sequence in two directions: one forward pass and one backward pass. This allows the model to capture dependencies from both the past and the future, enabling a more comprehensive understanding of the input sequence. By combining information from both directions, Bidirectional LSTMs are particularly effective in tasks where context from both past and future context is crucial, such as natural language processing, speech recognition, and sentiment analysis. The forward and backward hidden states of the Bidirectional LSTM are typically concatenated or combined to produce the final output, providing a more robust representation of the input sequence. Overall, Bidirectional LSTMs are a powerful extension of LSTMs that enhance the model's ability to capture long-term dependencies and improve performance in various sequential data analysis tasks.



```
#Trained on google Collab, using GPU for faster training
#so dont run this cell, run the below cell in which we have loaded the already trained model

shape = (n_words,len(uniq_tokens))
model = Sequential()

model.add(Bidirectional(LSTM(128, return_sequences=True), input_shape=shape))
model.add(Dropout(0.2))
model.add(Bidirectional(LSTM(128)))
model.add(Dense(len(uniq_tokens), kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.2))
model.add(Dense(len(uniq_tokens)))
model.add(Activation("softmax"))
model.compile(loss="categorical_crossentropy",optimizer=RMSprop(learning_rate=0.001),metrics=["accuracy"])
early_stop = EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(X,Y,batch_size=128,validation_split=0.2,epochs=150,shuffle=True,callbacks=[early_stop])
```

9) Training the Model:

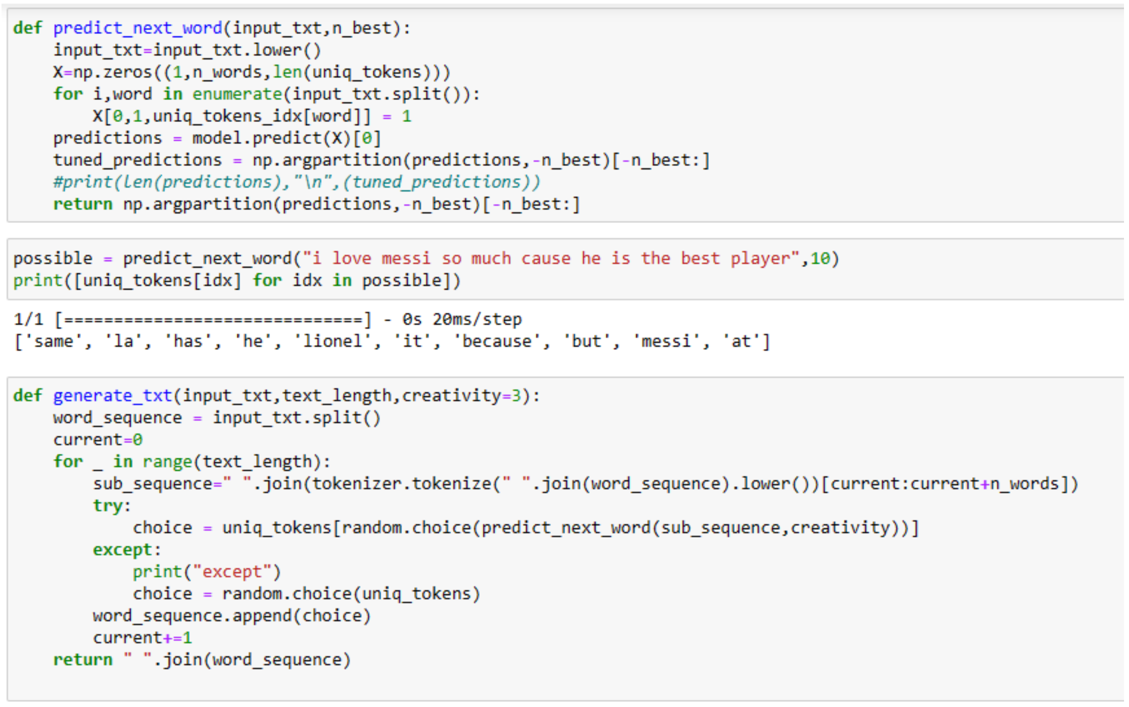
The model is compiled with the categorical cross-entropy loss function, which is commonly used for multi-class classification problems. The optimizer used is RMSprop with a learning rate of 0.001, which is responsible for updating the model's weights during training. The training progress is monitored by the 'val_loss' metric, and early stopping is implemented with a patience of 10, meaning training will stop if the validation loss does not improve for 10 consecutive epochs. During training, the model is fed with input data 'X' and corresponding target labels 'Y'. The training is performed in batches, with a batch size of 128. A validation split of 0.2 is applied, meaning that 20% of the data is used for validation while the remaining 80% is used for training. The training is set to run for a maximum of 150 epochs, with the 'shuffle' parameter ensuring that the training data is randomly shuffled before each epoch to avoid any bias in the order of the samples. The training process is also enhanced by the inclusion of the 'early_stop' callback, which terminates the training if the validation loss does not improve for a specified number of epochs. This helps prevent overfitting and saves computational resources by stopping the training early when the model's performance on the validation set no longer improves significantly.



```
model.compile(loss="categorical_crossentropy",optimizer=RMSprop(learning_rate=0.001),metrics=["accuracy"])
early_stop = EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(X,Y,batch_size=128,validation_split=0.2,epochs=150,shuffle=True,callbacks=[early_stop])
```

10) Next Word Prediction:

It takes the input text and the number of top predictions to return as parameters. The function converts the input text to lowercase and initializes an array to represent the input sequence. It then loops through the words in the input text, sets the corresponding position in the input array to 1, and uses the trained model to make predictions for the next word. The function returns the top predictions as a list of tokens. In the example provided, the function is used to predict the next word given any input text.



```
def predict_next_word(input_txt,n_best):
    input_txt=input_txt.lower()
    X=np.zeros((1,n_words,len(uniq_tokens)))
    for i,word in enumerate(input_txt.split()):
        X[0,1,uniq_tokens_idx[word]] = 1
    predictions = model.predict(X)[0]
    tuned_predictions = np.argpartition(predictions,-n_best)[-n_best:]
    #print(len(predictions),"\n",(tuned_predictions))
    return np.argpartition(predictions,-n_best)[-n_best:]
```

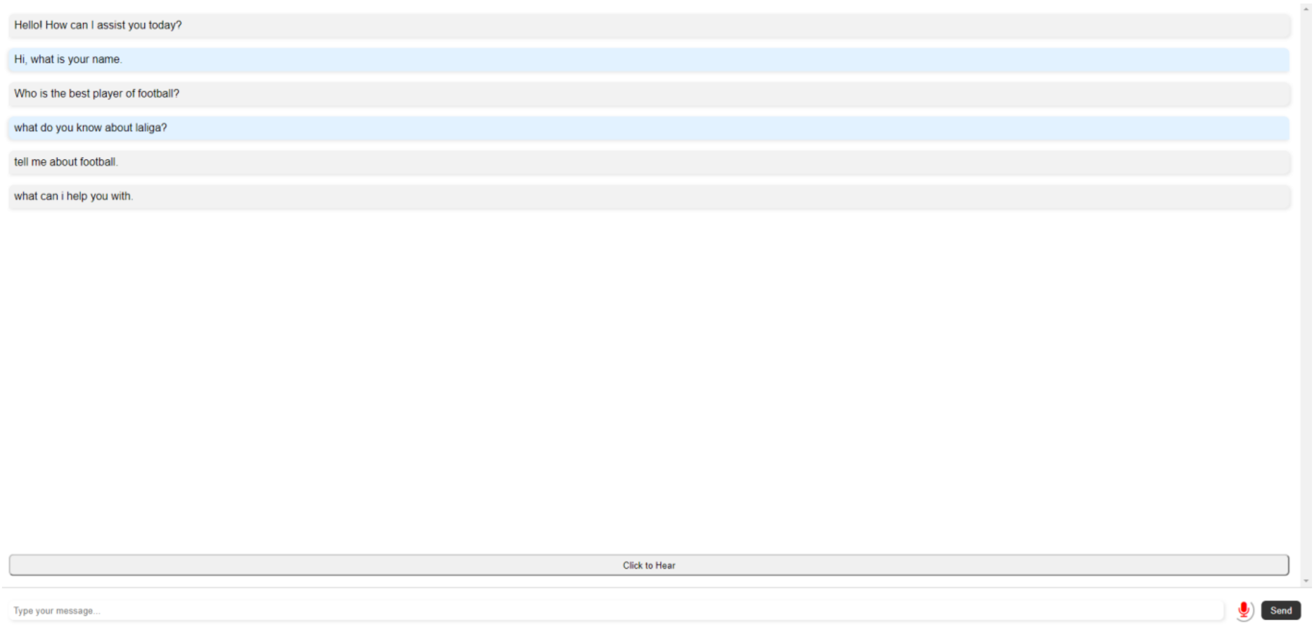
```
possible = predict_next_word("i love messi so much cause he is the best player",10)
print([uniq_tokens[idx] for idx in possible])
```

```
1/1 [==============================] - 0s 20ms/step
['same', 'la', 'has', 'he', 'lionel', 'it', 'because', 'but', 'messi', 'at']
```

```
def generate_txt(input_txt,text_length,creativity=3):
    word_sequence = input_txt.split()
    current=0
    for _ in range(text_length):
        sub_sequence=" ".join(tokenizer.tokenize(" ".join(word_sequence).lower())[current:current+n_words])
        try:
            choice = uniq_tokens[random.choice(predict_next_word(sub_sequence,creativity))]
        except:
            print("except")
            choice = random.choice(uniq_tokens)
        word_sequence.append(choice)
        current+=1
    return " ".join(word_sequence)
```

11) Website Implementation:

To make the sports chatbot open to clients, it is incorporated into a site. The site gives an easy-to-understand interface where clients can communicate with the chatbot, ask sports-related inquiries, and get ongoing reactions. The incorporation should be possible utilizing web improvement innovations like HTML, CSS, and JavaScript.


Hello! How can I assist you today?

Hi, what is your name.

Who is the best player of football?

what do you know about laliga?

tell me about football.

what can i help you with.

12) Text to Speech & Speech to Text Module:

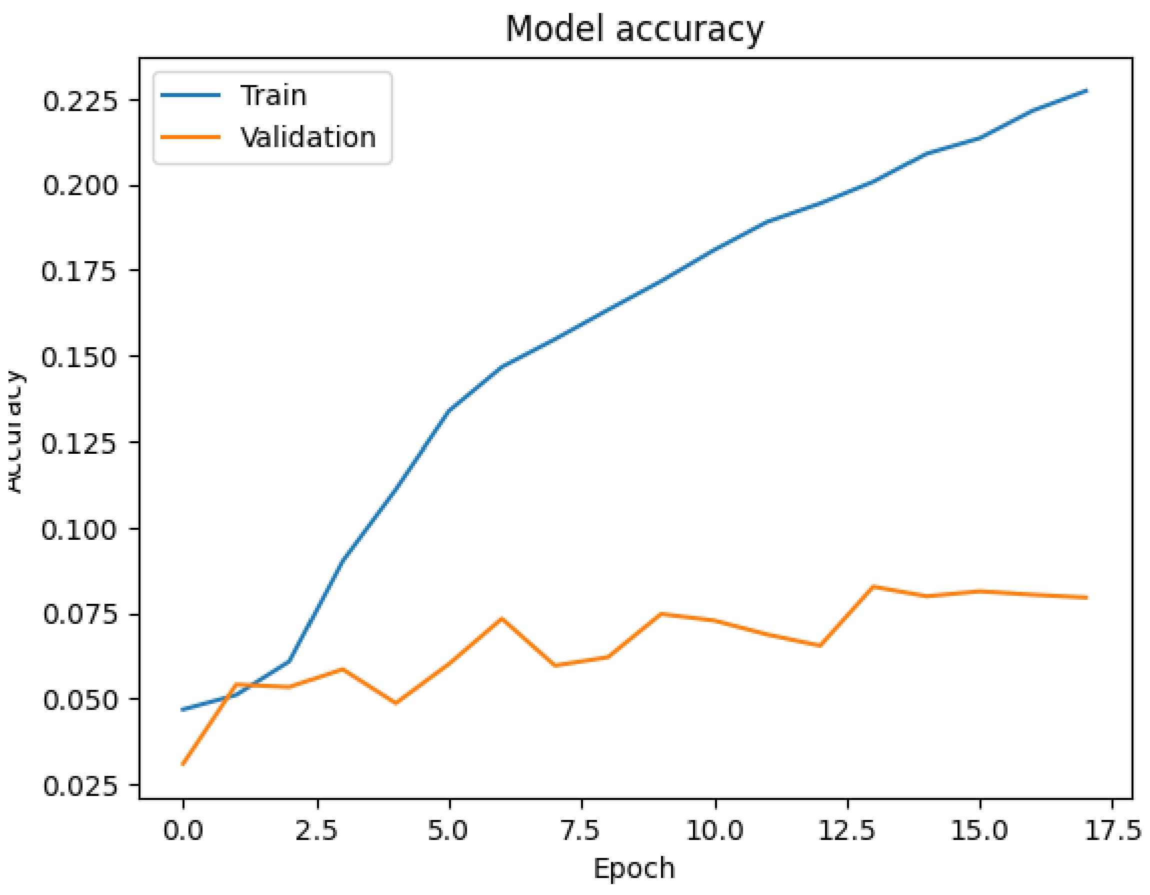
We also created a text to speech and a speech to text module. That will allow users to say anything to the chatbot and the chatbot will then answer back to the use using the speech module.

```
import speech_recognition as sr
import pyttsx3
# Initialize the recognizer
r = sr.Recognizer()

# Initialize the text-to-speech engine
engine = pyttsx3.init('dummy')
# Set the voice to a different voice
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)  # char
```

```
def listen():
    with sr.Microphone() as source:
        print("Speak now:")
        audio = r.listen(source)
    try:
        text = r.recognize_google(audio)
        print("You said:", text)
    except Exception as e:
        print("Error:", e)
    return text
def speak(text):
    engine.say(text)
    engine.runAndWait()
```

13) Limitations:

As there are limitations to every project out there in the real world, some of them were in ours as well. Our project was somewhat overfitting for which we used many techniques like dropout layers, embeddings and BiLSTM etc. to make it less overfit. We also used Callbacks and early stops as shown above.



Results:

The sports chatbot was verified utilizing different sports related questions and assessed in light of its exactness and reaction quality. The tokenization interaction successfully coordinated input text, permitting the chatbot to comprehend client questions. The RNN and LSTM models gave logically important reactions, taking into account the past discussion history. The bag of Words approach helped in producing exact and significant reactions in light of the information text.

The site execution of the chatbot gave a natural and available stage for clients to take part in sports discussions. The chatbot effectively answered many sports related questions.

Conclusions:

The development and implementation of a sports chatbot using text-based generation techniques have shown promising results. The integration of tokenization, RNN, LSTM, and Bag of Words enables the chatbot to understand and respond to user queries effectively. The website implementation enhances user experience by providing a user-friendly platform for sports-related conversations.

References:

I. https://www.kaggle.com/datasets/gpreda/lionel-messi-tweets?select=messi_tweets.csv

II. https://www.kaggle.com/datasets/ibrahimserouis99/twitter-sentiment-analysis-and-word-embeddings