

**LAPORAN UAS DEEP LEARNING  
CHATBOT LAYANAN AKADEMIK UNIB**



**Disusun Oleh :**

- |                                    |                    |
|------------------------------------|--------------------|
| <b>1. Muhamad Rifqi Afriansyah</b> | <b>(G1A021023)</b> |
| <b>2. Vilda Aprilia</b>            | <b>(G1A021033)</b> |
| <b>3. Muhamad Iqbal</b>            | <b>(G1A021073)</b> |

**Dosen Mata Kuliah:**

**Arie Vatesia, S.T., M.T.I., Ph.D**

**PROGRAM STUDI INFORMATIKA  
FAKULTAS TEKNIK  
UNIVERSITAS BENGKULU  
2024**

## 1. Pendahuluan

Pelayanan akademik adalah aspek penting yang perlu diperhatikan dan disediakan oleh para *stakeholder* di universitas untuk seluruh komunitas akademik di dalamnya, guna mendukung keberhasilan proses pembelajaran dan pengelolaan administrasi. Keterbatasan jumlah petugas menjadi kendala yang mengurangi kepuasan civitas akademika Program Studi Informatika Universitas Bengkulu dalam menerima informasi. Sehingga Ketika terjadi kendala seperti petugas yang sedang cuti, petugas yang istirahat, atau yang tidak datang ke kantor masih menjadi kendala dalam mendapatkan informasi akademik. Keterbatasan-keterbatasan ini mengakibatkan pihak yang membutuhkan informasi hanya bisa mendapatkannya dengan datang langsung ke universitas pada waktu jam kerja, yang tentunya kurang efisien dan tidak fleksibel.

Seiring dengan pesatnya kemajuan teknologi, berbagai aspek kehidupan juga mengalami perkembangan, termasuk dalam hal penggunaan dan pengembangan situs web. Salah satu platform yang dapat dimanfaatkan untuk menyampaikan informasi secara efektif adalah chatbot berbasis deep learning. Chatbot ini dirancang untuk memberikan respons yang cepat dan akurat terhadap berbagai pertanyaan mahasiswa terkait layanan akademik. Chatbot ini bertujuan untuk mengurangi beban layanan manual. Hal ini memungkinkan mahasiswa mendapatkan akses informasi yang fleksibel, sehingga meminimalkan ketergantungan pada layanan manual. Selain itu, chatbot juga membantu mengurangi beban kerja administrasi dan meningkatkan efisiensi interaksi antara mahasiswa dan pihak administrasi.

## 2. Analisa Model

Berdasarkan model yang telah dibangun pada chatbot Layanan Akademik menggunakan algoritma LSTM, model ini termasuk ke dalam deep learning dan bukan shallow learn, dapat dilihat dari Analisa berdasarkan karakteristik dan komponen model. Model menggunakan arsitektur LSTM dengan beberapa lapisan, termasuk layer embedding, LSTM, dan fully connected (dense) layer. Arsitektur yang digunakan pada chatbot layanan akademik memiliki 5 lapisan dengan jumlah total parameter 139,936, sehingga jelas tergolong deep learning karena kedalamannya melampaui model shallow learning. Model juga menggunakan embedding layer untuk mengubah data teks menjadi representasi numerik yang dapat diproses oleh LSTM, tanpa memerlukan feature engineering manual. Dengan embedding layer dan LSTM, model mempelajari representasi dari data teks secara bertahap, yang merupakan pendekatan deep learning. Model menggunakan tokenizer dan lemmatization untuk memproses

teks, serta pelatihan dengan 300 epoch, pada proses ini membutuhkan sumber daya komputasi yang signifikan yaitu deep learning. Hasil performa akurasi dan loss sangat baik dalam mempelajari pola dan menyelaraskan prediksi yang menunjukkan efektivitas model deep learning.

### 3. Penjelasan Kode

#### 3.1 Pengumpulan Data

Data untuk pengembangan chatbot ini dikumpulkan dari sumber terkait layanan akademik di Universitas Bengkulu pada laman website [www.unib.ac.id/](http://www.unib.ac.id/). Pertama, data awal diperoleh dengan mengambil informasi terkait layanan akademik melalui dokumen dan materi yang disediakan oleh Universitas Bengkulu pada laman web pusat informasi di bagian informasi akademik, dan bagian informasi lainnya yang relevan. Selanjutnya, data tersebut diproses untuk menyusun dataset percakapan yang mencakup berbagai pertanyaan dan jawaban seputar layanan akademik di Universitas Bengkulu. Dataset ini digunakan untuk melatih model chatbot agar dapat memberikan respons yang tepat dan relevan dengan layanan yang tersedia di Universitas Bengkulu.

#### 3.2 Preprocessing Data

```
[ ] | pip install indoNLP #ur indoNLP -> untuk mengganti slang/singkatan menjadi kata yang lebih formal -> sumber asli bisa liat di kamus alay dari nana
Requirement already satisfied: indoNLP in /usr/local/lib/python3.10/dist-packages (0.3.4)

[ ] | pip install nlp-id # untuk stopwords removal dan lematisasi
Tampilkan output tersembunyi

[ ] | # Menghapus instalasi pustaka yang bermasalah atau konflik versi
# Pustaka yang dihapus: gcsfs, fsspec, huggingface-hub, transformers, peft
!pip uninstall gcsfs fsspec huggingface-hub transformers peft -y

# Menginstal pustaka fsspec dengan versi spesifik (2024.10.0) yang kompatibel dengan gcsfs
# Ini dilakukan untuk menghindari konflik antara gcsfs dan versi fsspec lainnya
!pip install fsspec==2024.10.0

# Menginstal kembali pustaka gcsfs setelah memastikan bahwa fsspec memiliki versi yang sesuai
# gcsfs adalah pustaka yang membutuhkan fsspec dengan versi tertentu
!pip install gcsfs

# Menginstal pustaka huggingface-hub dengan versi minimal 0.25.0
# Versi ini diperlukan untuk kompatibilitas dengan transformers dan peft
!pip install huggingface-hub>=0.25.0

# Menginstal pustaka transformers, yang digunakan untuk pemrosesan model bahasa berbasis Transformer
# Pastikan huggingface-hub sudah kompatibel dengan transformers
!pip install transformers

# Menginstal pustaka peft, yang digunakan untuk fine-tuning model besar dengan pendekatan parameter-efficient
# Pastikan peft kompatibel dengan huggingface-hub dan transformers
!pip install peft
```

Penjelasan :

Kode ini digunakan untuk menginstall library yang dibutuhkan oleh program dan mengelola pustaka Python yang terkait dengan pemrosesan model, indoNLP digunakan untuk mengganti slang/singkatan menjadi kata yang lebih formal, nlp-id digunakan untuk menangani teks dalam bahasa Indonesia, uninstall gcsfs fsspec huggingface-hub transformers peft -y digunakan untuk menghapus pustaka ini yang bermasalah. fsspec digunakan untuk menghindari konflik antara gcsfs dan versi fsspec lainnya, gcsfs diinstal kembali dengan memastikan

kompatibilitasnya dengan versi fsspec yang baru diinstal. Selanjutnya, pustaka huggingface-hub diinstal dengan versi minimal 0.25.0 untuk memastikan kompatibilitasnya dengan pustaka transformers dan peft. transformers diinstal untuk memungkinkan pemrosesan model berbasis Transformer, dan terakhir, pustaka peft diinstal untuk mendukung fine-tuning model besar dengan pendekatan parameter-efficient

```
# Membuka file JSON yang berisi dataset chatbot (Intents) dari Google Drive
# Menggunakan 'with open' untuk memastikan file ditutup setelah digunakan
with open('/content/drive/MyDrive/Dataset Chatbot/intents.json') as content:
    data1 = json.load(content) # Memuat isi file JSON ke dalam variabel 'data1' sebagai dictionary

# Inisialisasi variabel untuk menyimpan data yang diperlukan
tags = [] # Daftar untuk menyimpan semua tag dari dataset
inputs = [] # Daftar untuk menyimpan semua pola (patterns) input
responses = {} # Dictionary untuk menyimpan respons berdasarkan tag
words = [] # Daftar untuk menyimpan semua kata unik dari pola
classes = [] # Daftar untuk menyimpan semua kategori (tag unik)
documents = [] # Daftar untuk menyimpan pasangan kata (token) dengan tag-nya
ignore_words = ['?', '!', '.', ',', '(', ')', '\'', '\"', '/', '\\'] # Karakter yang akan diabaikan

# Iterasi melalui setiap intent dalam file JSON
for intent in data1['intents']:
    responses[intent['tag']] = intent['responses'] # Menyimpan respons untuk setiap tag
    for lines in intent['patterns']:
        inputs.append(lines) # Menambahkan pola input ke daftar
        tags.append(intent['tag']) # Menambahkan tag terkait ke daftar
        for pattern in intent['patterns']:
            w = nltk.word_tokenize(pattern) # Tokenisasi setiap pola menjadi kata-kata (tokens)
            words.extend(w) # Menambahkan kata-kata ke daftar 'words'
            documents.append((w, intent['tag'])) # Menyimpan pasangan kata (tokens) dan tag
            if intent['tag'] not in classes: # Menambahkan tag baru ke daftar kelas jika belum ada
                classes.append(intent['tag'])

# Konversi data input (patterns) dan tag ke dalam DataFrame untuk analisis
data = pd.DataFrame({'patterns': inputs, 'tags': tags}) # Membuat DataFrame dengan kolom 'patterns' dan 'tags'

# Menampilkan semua kata yang telah dikumpulkan
print(words)
```

Penjelasan :

Kode ini digunakan untuk memproses data dari file JSON yang berisi dataset chatbot. Dataset ini berisi berbagai "intent" yang mencakup pola input (pertanyaan atau pernyataan dari pengguna) dan respons yang sesuai dari chatbot. Pertama adalah membuka dan membaca file JSON yang berisi data chatbot. Kemudian, kode ini mengumpulkan informasi penting seperti tag (kategori) dari setiap intent, pola input yang digunakan oleh pengguna, dan respons yang diberikan chatbot. Kata-kata yang muncul dalam pola input juga dikumpulkan dan di-tokenisasi atau dipisah menjadi kata-kata individual. Kata-kata unik dan tag yang berbeda disimpan dalam daftar terpisah untuk digunakan dalam pelatihan chatbot. Data ini akhirnya disusun dalam bentuk tabel (DataFrame) yang mencatat pola input bersama dengan tag yang sesuai.

```

# Menggabungkan semua kata dalam daftar 'words' menjadi satu string
# Kata-kata diubah menjadi huruf kecil (lowercase), dan tanda baca (punctuation) dihapus
words = " ".join([str(word.lower()) for word in words if word not in string.punctuation])

# Menyimpan string hasil penggabungan ke variabel 'qwerty' untuk referensi lebih lanjut
qwerty = words

# Membuat instance dari Lemmatizer dari pustaka nlp_id
# Lemmatizer ini digunakan untuk melakukan lemmatization (mengembalikan kata ke bentuk dasarnya) dalam bahasa Indonesia
lemmatizer = Lemmatizer()

# Melakukan lemmatization pada string 'words' untuk mendapatkan kata dalam bentuk dasar
words = lemmatizer.lemmatize(words)

# Membagi string hasil lemmatization kembali menjadi daftar kata-kata unik
# 'set' digunakan untuk menghilangkan duplikasi kata, dan 'sorted' untuk mengurutkan kata secara alfabetis
words = sorted(list(set(words.split()))))

# Menampilkan jumlah kata unik setelah lemmatization dan daftar kata tersebut
print(len(words), "unique lemmatized words", words)

```

Penjelasan :

Pada kode ini melakukan beberapa tahap pemrosesan, pertama menggabungkan kata pada semua daftar words digabung menjadi satu string, dan setiap kata diubah menjadi huruf kecil untuk konsistensi, string.punctuation untuk menghapus tanda baca. nlp\_id untuk lemmatize (dikembalikan ke bentuk dasarnya). Selanjutnya, kode ini menghilangkan kata yang terduplikasi dengan menggunakan set, yang hanya menyisakan kata-kata unik, dan menampilkan hasil.

```
[ ] # Menampilkan Data
data[30:50]
```

	patterns	tags
30	Dimana Mahasiswa dapat melihat Daftar Nilai ?	daftar_nilai1
31	Bagaimana Cara mengambil Daftar Nilai ?	daftar_nilai2
32	Dimana Saya dapat mengambil Daftar Nilai ?	daftar_nilai3
33	Apabila nilai mata kuliah Saya dari Dosen B ak...	daftar_nilai4
34	Bagaimana mekanisme menghapus nilai / mata kul...	daftar_nilai5
35	Apa yang dimaksud dengan Registrasi ?	Registrasi1
36	Apa beda dari Registrasi dengan Registrasi ?	Registrasi2
37	Kapan Registrasi dilaksanakan ?	Registrasi3
38	Kemana Saya harus mengurus Registrasi ?	Registrasi4
39	Apa saja persyaratan Registrasi ?	Registrasi5
40	Dimana pembayaran dapat dilakukan ?	Registrasi6
41	Bagaimana cara membayar dan apa saja yang haru...	Registrasi7
42	Apabila ada kendala saat mengupdate data mahas...	Registrasi8
43	Apa yang dimaksud dengan Ijazah ?	ijazah1
44	Kapan Ijazah diberikan ?	ijazah2
45	Bagaimana bila Ijazah saya hilang ?	ijazah3
46	Apa yang dimaksud dengan nomor Ijazah ?	ijazah4
47	Bagaimana mekanisme legalisir Ijazah ?	ijazah5
48	Bagaimana apabila Ijazah rusak karena Terbakar...	ijazah6

Penjelasan :

Merupakan output dari kode yang telah dijalankan sebelumnya, output diurutkan secara alfabetis untuk memudahkan analisis lebih lanjut.

```

# Analisis dataset - ketahui jumlah tag terbanyak

def get_tags(x):
    x['tags'] = x['tags'].apply(lambda x: re.sub(r"\d+", "", x))
    x = x.drop_duplicates(subset=['tags'], keep='first')
    return [i for i in x['tags']]

def count_tag(x):
    jumlah_tag = {}
    # Buat list kata yang ingin dihitung
    list_of_tag = get_tags(x)
    # Hitung jumlah kemunculan setiap kata
    for tag in list_of_tag:
        jumlah_tag[tag] = df_tag['tags'].str.contains(re.escape(tag)).sum()
    # Kembalikan hasil
    return jumlah_tag

df_tag = pd.DataFrame(data['tags'])
jumlah_tag = count_tag(df_tag)

[ ] # Menampilkan banyaknya jumlah tag

keys = list(jumlah_tag.keys())
values = list(jumlah_tag.values())
values = [int(x) for x in values]

df_tags = pd.DataFrame({"tag": keys, "jumlah": values})
df_tags[30:50]

```

Penjelasan :

Kode ini digunakan untuk menganalisis dataset dengan menghitung jumlah kemunculan setiap tag dalam kolom. fungsi `get_tags(x)` membersihkan tag dengan menghapus angka dan menghapus duplikasi, lalu mengembalikan daftar tag yang telah dibersihkan. fungsi `count_tag(x)` menghitung jumlah kemunculan setiap tag dalam dataset, DataFrame `df_tag` dibuat dari kolom 'tags' dalam dataset, dan fungsi `count_tag(df_tag)` digunakan untuk menghitung jumlah kemunculan setiap tag. Hasilnya disimpan dalam DataFrame `df_tags`, Terakhir, kode menampilkan baris ke-30 hingga ke-50 untuk melihat distribusi tag yang paling sering muncul dalam dataset. Proses ini berguna untuk menganalisis dan memahami sebaran tag dalam data.

```

# Menghilangkan tanda baca (punctuation) dari kolom 'patterns' pada DataFrame 'data'
data['patterns'] = data['patterns'].apply(lambda wrd: [ltrs.lower() for ltr in wrd if ltr not in string.punctuation])

# Setelah tanda baca dihapus, menggabungkan kembali daftar karakter menjadi string
# Misalnya: ['h', 'e', 'l', 'l', 'o'] -> "hello"
data['patterns'] = data['patterns'].apply(lambda wrd: ''.join(wrd))

# Menampilkan baris ke-30 hingga ke-50 dari DataFrame untuk memeriksa hasil transformasi
data[30:50]

Tampilkan output tersembunyi

[ ] # Melakukan lemmatization pada setiap pola (pattern) di kolom 'patterns' pada DataFrame 'data'
# Setiap kata dalam pattern dilematize menggunakan fungsi 'lemmatizer.lemmatize()'
# Hasilnya adalah daftar kata setelah proses lemmatization
data['patterns'] = data['patterns'].apply(lambda wrd: [lemmatizer.lemmatize(wrd)])

# Setelah lemmatization, daftar kata digabungkan kembali menjadi string utuh
# Misalnya: ['bermain'] -> "bermain"
data['patterns'] = data['patterns'].apply(lambda wrd: ''.join(wrd))

# Menampilkan baris ke-30 hingga ke-50 dari DataFrame untuk memeriksa hasil transformasi
data[30:50]

```

Penjelasan :

Pada kode ini melakukan beberapa langkah untuk membersihkan dan memproses data dalam kolom 'patterns' pada DataFrame `data`. Setiap karakter dalam pola diiterasi dan jika karakter tersebut termasuk tanda baca, maka karakter itu dihapus. Fungsi `lemmatizer.lemmatize()`

digunakan untuk mengubah kata-kata ke bentuk dasar, kemudian kode menampilkan baris ke-30 hingga ke-50 dari DataFrame untuk memeriksa hasil perubahan pada pola input.

```
[ ] # Menyortir data kelas tag
    classes = sorted(list(set(classes)))
    print (len(classes), "classes", classes)

316 classes ['PAK (Portal Akademik)', 'PAK (Portal Akademik)1',

# Mencetak jumlah keseluruhan data teks
# documents = kombinasi antara patterns and intents
print (len(documents), "documents")
```

Penjelasan :

Kode ini mengambil daftar kelas tag yang telah dikumpulkan, menghilangkan duplikasi dengan menggunakan `set()`, `sorted()` untuk mengurutkan kelas secara alfabetis, jumlah kelas dihitung menggunakan `len()`, lalu ditampilkan. Kemudian, `len(documents)` untuk menghitung jumlah keseluruhan dokumen.

```
[ ] # Membuat instance tokenizer dari TensorFlow Keras
    # Parameter 'num_words=2000' menunjukkan bahwa hanya 2000 kata yang paling sering muncul yang akan dipertimbangkan
    tokenizer = Tokenizer(num_words=2000)

    # Melatih tokenizer pada kolom 'patterns' dalam DataFrame 'data'
    # Proses ini menghasilkan indeks untuk setiap kata berdasarkan frekuensinya dalam dataset
    tokenizer.fit_on_texts(data['patterns'])

    # Mengubah teks dalam kolom 'patterns' menjadi urutan numerik berdasarkan indeks kata
    # Misalnya: Jika "main" memiliki indeks 1 dan "belajar" memiliki indeks 2, maka teks "main belajar" -> [1, 2]
    train = tokenizer.texts_to_sequences(data['patterns'])

    # Menampilkan hasil tokenisasi berupa daftar urutan numerik untuk setiap pola dalam dataset
    train
```

Penjelasan :

Pada kode ini menggunakan Tokenizer dari TensorFlow Keras untuk melakukan tokenisasi pada teks dalam kolom 'patterns' dari DataFrame data yang dibuat dengan parameter `num_words=2000`. Selanjutnya, tokenizer dilatih pada kolom 'patterns' menggunakan metode `fit_on_texts()`, kode `texts_to_sequences()` digunakan untuk mengubah teks dalam kolom 'patterns' diubah menjadi urutan numerik.

```
[ ] # Mendapatkan panjang input dari data pelatihan setelah proses padding
# 'X_train.shape' mengembalikan bentuk array NumPy dalam format (jumlah_sampel, panjang_urutan)
# '[1]' merujuk pada dimensi kedua, yaitu panjang urutan input
input_shape = X_train.shape[1]

# Menampilkan panjang input
# Panjang ini akan digunakan sebagai dimensi input dalam model deep learning
print(input_shape)

20

[ ] # Mendapatkan jumlah kata unik dalam tokenizer
# 'tokenizer.word_index' adalah dictionary yang berisi kata-kata unik sebagai kunci dan indeks sebagai nilai
# Fungsi 'len()' digunakan untuk menghitung jumlah kata unik dalam dictionary tersebut
vocabulary = len(tokenizer.word_index)

# Menampilkan jumlah kata unik dalam dataset
print("number of unique words : ", vocabulary)

number of unique words : 496

[ ] # Mendapatkan panjang output dari jumlah kelas unik dalam label encoder
# 'le.classes_' adalah array yang berisi semua label unik yang dipelajari oleh LabelEncoder
# 'le.classes_.shape[0]' memberikan jumlah total kelas unik
output_length = le.classes_.shape[0]

# Menampilkan panjang output (jumlah kelas unik)
print("output length: ", output_length)
```

Penjelasan :

Pada bagian pertama kode menghitung panjang input dari data pelatihan setelah proses padding, dengan mengambil dimensi kedua dari array `X_train` menggunakan `X_train.shape[1]`. Selanjutnya, kode menghitung jumlah kata unik dalam dataset dengan mengambil panjang dari `tokenizer.word_index`, terakhir kode menghitung panjang output berdasarkan jumlah kelas unik dalam label encoder. Dengan menggunakan `le.classes_.shape[0]`, untuk menghitung jumlah kelas unik yang dipelajari oleh `LabelEncoder`.

```
[ ] # Menyimpan daftar 'words' ke dalam file menggunakan modul 'pickle'
# File disimpan dalam format biner ('wb' = write binary)
# Lokasi penyimpanan: '/content/words.pkl'
pickle.dump(words, open('/content/words.pkl', 'wb'))

# Menyimpan daftar 'classes' ke dalam file menggunakan modul 'pickle'
# File disimpan dalam format biner ('wb' = write binary)
# Lokasi penyimpanan: '/content/classes.pkl'
pickle.dump(classes, open('/content/classes.pkl', 'wb'))

[ ] # Menyimpan objek LabelEncoder ('le') ke dalam file menggunakan modul 'pickle'
# File disimpan dalam format biner ('wb' = write binary)
# Lokasi penyimpanan: 'labelencoder.pkl'
pickle.dump(le, open('labelencoder.pkl', 'wb'))

# Menyimpan objek Tokenizer ('tokenizer') ke dalam file menggunakan modul 'pickle'
# File disimpan dalam format biner ('wb' = write binary)
# Lokasi penyimpanan: 'tokenizers.pkl'
pickle.dump(tokenizer, open('tokenizers.pkl', 'wb'))
```

Penjelasan :

Pada kedua cell kode ini menyimpan berbagai objek yang digunakan dalam pelatihan model, seperti daftar kata ('words'), daftar kelas ('classes'), objek 'LabelEncoder' ('le'), dan objek 'Tokenizer' ('tokenizer') ke dalam file biner menggunakan modul 'pickle'. File yang dihasilkan, seperti 'words.pkl', 'classes.pkl', 'labelencoder.pkl', dan 'tokenizers.pkl', memungkinkan untuk memuat kembali objek-objek tersebut di lain waktu, sehingga proses



pelatihan atau prediksi dapat dilanjutkan tanpa perlu memproses data atau melatih ulang model dari awal.

### 3.3 Pelatihan Model

```
[ ] # Membuat arsitektur model menggunakan TensorFlow Keras Functional API

# Layer Input: Mendefinisikan input dengan bentuk sesuai panjang urutan (input_shape)
# 'input_shape' adalah panjang dari setiap urutan setelah padding
i = Input(shape=(input_shape,))

# Layer Embedding: Mengubah indeks kata menjadi representasi vektor berdimensi 20
# Dimensi kosakata adalah 'vocabulary + 1' untuk memasukkan indeks nol (padding)
x = Embedding(vocabulary + 1, 20)(i)

# Layer LSTM: Menambahkan Long Short-Term Memory layer untuk mempelajari urutan kata
# LSTM memiliki 20 unit, dan 'return_sequences=True' untuk mengembalikan urutan keluaran
x = LSTM(20, return_sequences=True)(x)

# Layer Flatten: Mengubah keluaran LSTM (3D tensor) menjadi vektor 1D untuk layer Dense
x = Flatten()(x)

# Layer Dense: Menambahkan layer fully connected dengan jumlah neuron sesuai jumlah kelas
# 'output_length' adalah jumlah kelas unik, menggunakan aktivasi softmax untuk probabilitas
x = Dense(output_length, activation="softmax")(x)

# Membuat model dengan TensorFlow Keras Functional API
# Input adalah 'i' dan output adalah 'x'
model = Model(i, x)

# Kompilasi model: Menentukan fungsi loss, optimizer, dan metrik evaluasi
# - Loss: 'sparse_categorical_crossentropy' cocok untuk target yang berupa integer label
# - Optimizer: 'adam' adalah optimizer adaptif yang bekerja dengan baik untuk berbagai kasus
# - Metrics: 'accuracy' digunakan untuk mengevaluasi performa model
model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
```

Penjelasan :

Pada kode ini mendefinisikan arsitektur model, kode `i = Input(shape=(input_shape,))` digunakan untuk layer input dengan panjang urutan input yang sesuai. Kemudian, `x = Embedding(vocabulary + 1, 20)(i)` layer digunakan untuk mengubah indeks kata menjadi representasi vektor berdimensi 20. Selanjutnya, layer LSTM ditambahkan untuk mempelajari urutan kata, diikuti oleh layer flatten untuk mengubah keluaran LSTM menjadi vektor 1D. Layer dense dengan aktivasi softmax ditambahkan di akhir untuk menghasilkan output sesuai dengan jumlah kelas yang ada. Model kemudian dikompilasi dengan menggunakan fungsi loss `sparse_categorical_crossentropy`, optimizer `adam`, dan metrik `accuracy` untuk evaluasi.

```
# Menampilkan Parameter Model
model.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 20)	0
embedding (Embedding)	(None, 20, 20)	9,940
lstm (LSTM)	(None, 20, 20)	3,200
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 316)	126,716

Total params: 139,936 (546.62 KB)  
Trainable params: 139,936 (546.62 KB)  
Non-trainable params: 0 (0.00 B)

Penjelasan :

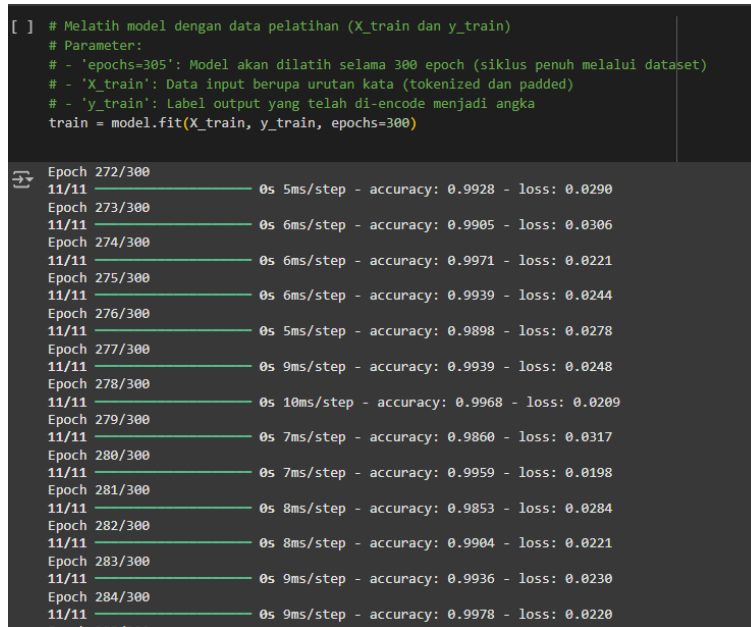
Kode diatas merupakan ringkasan dari model LSTM yang telah dibangun sebelumnya, dan menghasilkan output ringkasan model dengan beberapa layer :

1. **Input Layer:** Menerima input dengan panjang 20 (setelah padding), tanpa parameter.
2. **Embedding Layer:** Mengubah kata menjadi vektor 20 dimensi, dengan 9,940 parameter.
3. **LSTM Layer:** Memproses urutan kata dengan output 20x20, memiliki 3,280 parameter.
4. **Flatten Layer:** Meratakan output LSTM menjadi vektor 1D dengan 400 elemen, tanpa parameter.
5. **Dense Layer:** Menyambungkan output dengan 316 neuron, memiliki 126,716 parameter.

**Total Parameter:** 139,936, semuanya dapat dilatih, membutuhkan 546.62 KB memori.

### ➤ Train Model

```
[ ] # Melatih model dengan data pelatihan (X_train dan y_train)
# Parameter:
# - 'epochs=300': Model akan dilatih selama 300 epoch (siklus penuh melalui dataset)
# - 'X_train': Data input berupa urutan kata (tokenized dan padded)
# - 'y_train': Label output yang telah di-encode menjadi angka
train = model.fit(X_train, y_train, epochs=300)
```



```
Epoch 272/300
11/11 ----- 0s 5ms/step - accuracy: 0.9928 - loss: 0.0290
Epoch 273/300
11/11 ----- 0s 6ms/step - accuracy: 0.9905 - loss: 0.0306
Epoch 274/300
11/11 ----- 0s 6ms/step - accuracy: 0.9971 - loss: 0.0221
Epoch 275/300
11/11 ----- 0s 6ms/step - accuracy: 0.9939 - loss: 0.0244
Epoch 276/300
11/11 ----- 0s 5ms/step - accuracy: 0.9898 - loss: 0.0278
Epoch 277/300
11/11 ----- 0s 9ms/step - accuracy: 0.9939 - loss: 0.0248
Epoch 278/300
11/11 ----- 0s 10ms/step - accuracy: 0.9968 - loss: 0.0209
Epoch 279/300
11/11 ----- 0s 7ms/step - accuracy: 0.9860 - loss: 0.0317
Epoch 280/300
11/11 ----- 0s 7ms/step - accuracy: 0.9959 - loss: 0.0198
Epoch 281/300
11/11 ----- 0s 8ms/step - accuracy: 0.9853 - loss: 0.0284
Epoch 282/300
11/11 ----- 0s 8ms/step - accuracy: 0.9904 - loss: 0.0221
Epoch 283/300
11/11 ----- 0s 9ms/step - accuracy: 0.9936 - loss: 0.0230
Epoch 284/300
11/11 ----- 0s 9ms/step - accuracy: 0.9978 - loss: 0.0220
Epoch 285/300
```

Penjelasan :

Kode diatas merupakan kode untuk melatih model dengan menggunakan data pelatihan X\_train, y\_train dengan epoch sebanyak 300, dan model akan dilatih hingga 300 epoch dengan step per epoch 11.

### 3.4 Evaluasi Model

#### ➤ Akurasi model

```
# Membuat visualisasi akurasi selama pelatihan model
# 'plt.figure()' digunakan untuk membuat kanvas baru dengan ukuran 14x5
plt.figure(figsize=(14, 5))

# Membuat subplot untuk visualisasi akurasi (1 dari 2 grafik)
# '1, 2, 1' berarti 1 baris, 2 kolom, dan ini adalah subplot pertama
plt.subplot(1, 2, 1)

# Memplot akurasi pelatihan dari objek 'train.history'
# 'train.history['accuracy']' berisi daftar akurasi untuk setiap epoch
plt.plot(train.history['accuracy'], label='Training Set Accuracy', color='green')

# Menambahkan legenda pada grafik, diletakkan di sudut kanan bawah ('lower right')
plt.legend(loc='lower right')

# Menambahkan judul grafik
plt.title('Accuracy')

# Menyimpan grafik sebagai file gambar PNG
# File akan disimpan dengan nama 'grafik_accuracy.png'
# 'bbox_inches="tight"' memastikan grafik tidak terpotong
plt.savefig('grafik_accuracy.png', bbox_inches='tight')

# Menampilkan grafik di layar
plt.show()
```

Penjelasan :

Kode ini berisikan kode yang berfungsi untuk memvisualisasikan akurasi model selama pelatihan. `plt.figure(figsize=(14, 5))` membuat area plot dengan ukuran 14x5 inci, `plt.subplot(1,2,1)` menentukan plot pertama dari dua subplot dalam satu baris, `plt.plot(train.history['accuracy'], label='Training Set Accuracy', color='green')` menggambar grafik untuk akurasi pelatihan yang tercatat selama proses pelatihan, `plt.legend(loc='lower right')` menambahkan legenda di sudut kanan bawah, `plt.title('Accuracy')` memberikan judul pada grafik, `plt.savefig('grafik_accuracy.png', bbox_inches='tight')` menyimpan grafik sebagai gambar dengan nama 'grafik\_accuracy.png', `plt.show()` menampilkan grafik di layar.

#### ➤ Loss model

```

# Membuat visualisasi loss selama pelatihan model
# 'plt.figure()' digunakan untuk membuat kanvas baru dengan ukuran 14x5
plt.figure(figsize=(14, 5))

# Membuat subplot untuk visualisasi loss (2 dari 2 grafik)
# '1, 2, 2' berarti 1 baris, 2 kolom, dan ini adalah subplot kedua
plt.subplot(1, 2, 2)

# Memplot loss pelatihan dari objek 'train.history'
# 'train.history['loss']' berisi daftar loss untuk setiap epoch
plt.plot(train.history['loss'], label='Training Set Loss', color='red')

# Menambahkan legenda pada grafik, diletakkan di sudut kanan atas ('upper right')
plt.legend(loc='upper right')

# Menambahkan judul grafik
plt.title('Loss')

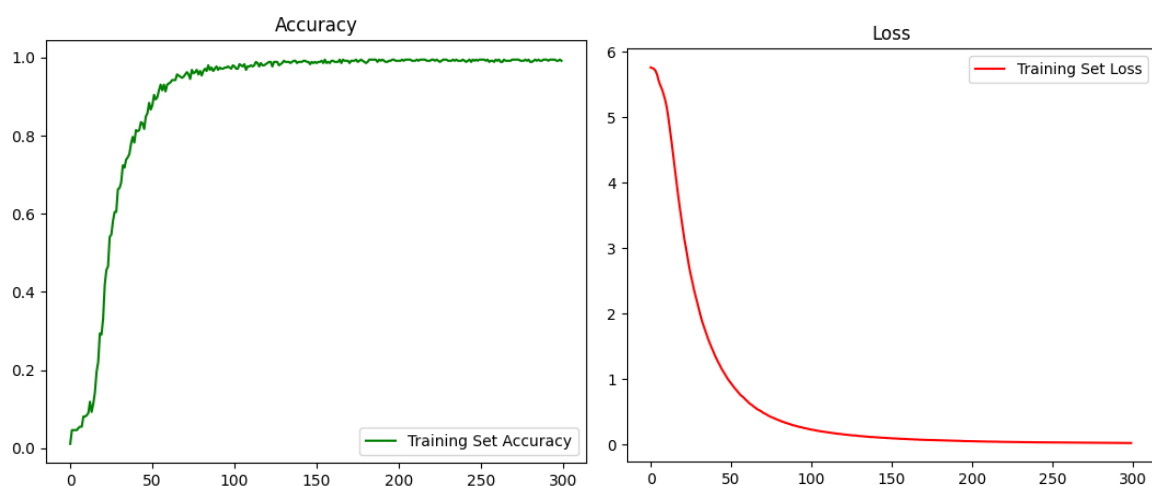
# Menyimpan grafik sebagai file gambar PNG
# File akan disimpan dengan nama 'grafik_loss.png'
# 'bbox_inches="tight"' memastikan grafik tidak terpotong
plt.savefig('grafik_loss.png', bbox_inches='tight')

# Menampilkan grafik di layar
plt.show()

```

Kode ini berisikan kode yang berfungsi untuk memvisualisasikan loss model selama pelatihan. `plt.figure(figsize=(14, 5))` membuat area plot dengan ukuran 14x5 inci, `plt.subplot(1, 2, 2)` menentukan plot kedua dari dua subplot dalam satu baris, `plt.plot(train.history['loss'], label='Training Set Loss', color='red')` menggambar grafik untuk loss yang tercatat selama proses pelatihan, `plt.legend(loc='upper right')` menambahkan legenda di sudut kanan atas, `plt.savefig('grafik_loss.png', bbox_inches='tight')` menyimpan grafik sebagai gambar dengan nama 'grafik\_loss.png'.

#### ➤ Grafik visual



Penjelasan :

Pada kedua gambar diatas merupakan hasil visualisasi grafik akurasi dan loss dari hasil model yang telah di train menggunakan LSTM dengan 300 epoch, grafik menunjukkan hasil yang

sangat bagus dengan kenaikan dan turun secara bertahap dengan nilai akurasi 0.9968 dan loss 0.0178, hal ini menunjukkan bahwa model telah berhasil dilatih dengan baik tanpa overfitting atau underfitting.

➤ Prediksi akurasi

```
[ ] # Membuat prediksi dengan model yang telah dilatih
y_pred = model.predict(X_train)
y_pred = y_pred.argmax(axis=1)

# Menghitung Nilai Akurasi
accuracy = accuracy_score(y_train, y_pred)
print("Accuracy:", accuracy)
```

11/11 ————— 0s 11ms/step  
Accuracy: 0.9941860465116279

Penjelasan :

Kode ini digunakan untuk membuat prediksi dengan model yang telah dilatih dan menghitung akurasi model. Model yang telah dilatih digunakan untuk menghasilkan prediksi pada data pelatihan (X\_train) menggunakan fungsi `model.predict(X_train)`. Hasil prediksi ini berupa probabilitas untuk setiap kelas, kemudian `y_pred.argmax(axis=1)`. Setelah itu, akurasi model dihitung dengan membandingkan hasil prediksi (`y_pred`) dengan label yang sebenarnya (`y_train`) menggunakan `accuracy_score(y_train, y_pred)`, hasilnya ditampilkan dengan `print("Accuracy:", accuracy)`.

➤ Testing Model

```

# Menentukan batas probabilitas
batas_probabilitas = 0.8

# Membuat Input Chat
while True:
    texts_p = []
    prediction_input = input(' Kamu : ')

    # Preprocessing teks
    prediction_input = [letters.lower() for letters in prediction_input if letters not in string.punctuation]
    prediction_input = ''.join(prediction_input)
    prediction_input = lemmatizer.lemmatize(prediction_input)
    texts_p.append(prediction_input)

    # Tokenisasi dan Padding
    prediction_input = tokenizer.texts_to_sequences(texts_p)
    prediction_input = np.array(prediction_input).reshape(-1)
    prediction_input = pad_sequences([prediction_input], input_shape)

    # Mendapatkan hasil keluaran pada model
    output = model.predict(prediction_input)
    output_probabilitas = round(output.max(), 2)
    output = output.argmax()

    # Memeriksa probabilitas
    if output_probabilitas < batas_probabilitas:
        print(" Unibbot : Maaf, saya tidak mengerti pertanyaan anda.")
    else:
        # Menemukan respon sesuai data tag dan memainkan voice bot
        response_tag = le.inverse_transform([output])[0]
        print(" Unibbot : ", random.choice(responses[response_tag]))

    time.sleep(0.88)
    print("-"*60 + "\n")

    # Tambahkan respon 'goodbye' agar bot bisa berhenti
    if response_tag == "goodbye":
        break

```

Penjelasan :

Kode ini adalah bagian dari implementasi chatbot yang menggunakan model yang telah dilatih untuk memberikan respons berdasarkan input pengguna. Pertama, pengguna diminta untuk memasukkan teks melalui `input()`. Kemudian, teks tersebut diproses dengan menghapus tanda baca, mengubah huruf menjadi kecil, dan melakukan lemmatization. Setelah teks diproses, teks tersebut kemudian di-tokenisasi dan dipadatkan (padding) agar sesuai dengan input yang dibutuhkan model. Model kemudian digunakan untuk memprediksi kelas yang paling sesuai dengan input menggunakan `model.predict()`. Prediksi ini menghasilkan probabilitas untuk setiap kelas, dan jika probabilitas tertinggi kurang dari batas yang ditentukan (0.8), bot akan memberi tahu bahwa pertanyaan tidak dipahami. Jika probabilitas lebih tinggi dari batas, bot akan memberikan respons sesuai dengan kelas yang diprediksi, yang diambil dari dataset respons yang telah disiapkan. Proses ini berulang sampai pengguna mengucapkan "goodbye," yang akan menghentikan percakapan chatbot.

#### 4. Kesimpulan

Chatbot layanan akademik ini dirancang untuk meningkatkan efisiensi layanan akademik di Universitas Bengkulu dengan memberikan respons cepat dan relevan terhadap pertanyaan mahasiswa terkait layanan akademik. Melalui implementasi algoritma *Deep Learning* LSTM (*Long Short-Term Memory*), chatbot berhasil mencapai *accuracy* tinggi sebesar 0.9968 dengan nilai loss 0,0178, dan rata-rata keseluruhan rata-rata nilai *accuracy* 0,9212 rata-rata nilai loss: 0,4893. Hasil ini menunjukkan bahwa model LSTM memiliki performa yang sangat baik dalam mempelajari pola data, dengan tingkat error (loss) yang sangat rendah dan tingkat akurasi yang hampir mendekati sempurna pada akhir pelatihan. Model ini siap untuk digunakan dalam pengujian atau implementasi. Penggunaan dataset berbasis layanan akademik dari situs Universitas Bengkulu serta preprocessing data yang teliti memungkinkan chatbot ini untuk memahami dan merespons berbagai pertanyaan dengan akurasi yang tinggi. Dengan adanya chatbot ini, diharapkan dapat mengurangi beban layanan manual dan meningkatkan fleksibilitas serta kepuasan pengguna dalam memperoleh informasi akademik di Universitas Bengkulu.