DAFTAR PUSTAKA

Agatonovic-Kustrin, S., & Beresford, R. (2000). Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis*, *22*(5), 717–727. https://doi.org/10.1016/S0731-7085(99)00272-1

algoritma. (2022, March 23). *Memahami Recurrent Neural Network*. Algoritma Data Science School. https://algorit.ma/blog/rnn-adalah-2022/

Annapurna P. Patil, Amrita Behera, & Anusha P. (2019). *Speech Enabled Visual Question Answering using LSTM and CNN with Real Time Image Capturing for assisting the Visually Impaired*.

Bashar, S. K., Al Fahim, A., & Chon, K. H. (2020). Smartphone Based Human Activity Recognition with Feature Selection and Dense Neural Network. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, *2020-July*, 5888–5891. https://doi.org/10.1109/EMBC44109.2020.9176239

Cai, L. Q., Wei, M., Zhou, S. T., & Yan, X. (2020). Intelligent Question Answering in Restricted Domains Using Deep Learning and Question Pair Matching. *IEEE Access*, *8*, 32922–32934. https://doi.org/10.1109/ACCESS.2020.2973728

Chen, K.-H., Kao, J.-I., Wu, C.-H., Chen, Y.-R., & Huang, Y.-C. (2019). A Chinese Oral Question-and-Answering System Based on LSTM. *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*. http://arxiv.org/abs/1608.07905

Cuayáhuitl, H., Lee, D., Ryu, S., Choi, S., Hwang, I., & Kim, J. (2017). *Deep Reinforcement Learning for Chatbots Using Clustered Actions and Human-Likeness Rewards*. http://parl.ai/

Ferreira, E., Jabaian, B., & Lefèvre, F. (2015). Online adaptative zero-shot learning spoken language understanding using word-embedding. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, *2015-Augus*, 5321–5325. https://doi.org/10.1109/ICASSP.2015.7178987

Hahn, U., & Duan, T. (2019). Corpus assembly as text data integration from digital libraries and the web. *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries*, *2019-June*, 25–28. https://doi.org/10.1109/JCDL.2019.00014

Ho, H., Mawardi, V. C., & Dharmawan, A. B. (2017). *Question Answering System with Hidden Markov Model Speech Recognition*. https://ieeexplore.ieee.org/document/8257121

Keya, M., Kaisar, A., Masum, M., Majumdar, B., Hossain, S. A., & Abujar, S. (2020). Bengali Question Answering System Using Seq2Seq Learning Based on General Knowledge Dataset. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*.

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1–15.

Lattifia, T., Wira Buana, P., & Rusjayanthi, N. K. D. (2022). Model Prediksi Cuaca Menggunakan Metode LSTM. *JITTER Jurnal Ilmiah Teknologi Dan Komputer*, *3*(1), 994–1000.

Lee, C.-H., Chen, Y.-N., & Lee, H.-Y. (2019). *MITIGATING THE IMPACT OF SPEECH RECOGNITION ERRORS ON SPOKEN QUESTION ANSWERING BY ADVERSARIAL DOMAIN ADAPTATION*. 7020.

Lestari, D. P., & Nugraha, R. R. (2017). *A Spoken-Based Question Answering System for Train Route Service using the Frame-Based Approach*. https://ieeexplore.ieee.org/document/8312384

Luo, S.-B., Lee, H.-S., Chen, K.-Y., & Wang, H.-M. (2019). *SPOKEN MULTIPLE-CHOICE QUESTION ANSWERING USING MULTIMODAL CONVOLUTIONAL NEURAL NETWORKS*. https://ieeexplore.ieee.org/document/9003966

Moharm, K., Eltahan, M., & Elsaadany, E. (2020). Wind speed forecast using LSTM and Bi-LSTM algorithms over gabal el-zayt wind farm. *Proceedings - 2020 International Conference on Smart Grids and Energy Systems, SGES 2020*, 922–927. https://doi.org/10.1109/SGES51519.2020.00169

Muangkammuen, P., Intiruk, N., & Runapongsa Saikaew, K. (2018). *Automated Thai-FAQ Chatbot using RNN-LSTM*.

Nazari, F., & Yan, W. (2021). *Convolutional versus Dense Neural Networks : Comparing the Two Neural N etworks ' Performance in Predicting Building Operational Energy Use Based on the Building Shape Department of Architecture , Texas A & M University , College Station , TX , USA Abstra*. https://doi.org/https://doi.org/10.48550/arXiv.2108.12929

Peyton, K., & Unnikrishnan, S. (2023). A comparison of chatbot platforms with the state-of-the-art sentence BERT for answering online student FAQs. *Results in Engineering*, *17*(October 2022), 100856. https://doi.org/10.1016/j.rineng.2022.100856

Su, D., & Fung, P. (2020). *IMPROVING SPOKEN QUESTION ANSWERING USING CONTEXTUALIZED WORD REPRESENTATION*.

Sultana Ritu, Z., Nowshin, N., Mahadi Hasan Nahid, M., & Ismail, S. (2018). Performance Analysis of Different Word Embedding Models on Bangla

Language. *2018 International Conference on Bangla Speech and Language Processing, ICBSLP 2018*, 1–5. https://doi.org/10.1109/ICBSLP.2018.8554681

Taniya, Bhardwaj, V., & Kadyan, V. (2020). Deep Neural Network Trained Punjabi Children Speech Recognition System Using Kaldi Toolkit. *2020 IEEE 5th International Conference on Computing Communication and Automation, ICCCA 2020*, 374–378. https://doi.org/10.1109/ICCCA49541.2020.9250780

Unlu, M., & Arisoy, E. (2021). Uncertainty-Aware Representations for Spoken Question Answering. *2021 IEEE Spoken Language Technology Workshop, SLT 2021 - Proceedings*, 943–949. https://doi.org/10.1109/SLT48900.2021.9383547

Utomo, A. D. (2020). *EDUCATIONAL ASSISTANCE ROBOT FEATURES: ORAL - BASED QUESTION ANSWERING SYSTEM IN THE DOMAIN OF EARLY CHILDHOOD EDUCATION*.

Utomo, A. D., Zainuddin, Z., & Syarif, S. (2020). Question Answering System in the Domain of Early Childhood Education in Bahasa Indonesia. *Proceedings - 2020 6th International Conference on Science and Technology, ICST 2020*. https://doi.org/10.1109/ICST50505.2020.9732850

Yeung, M., Sala, E., Schönlieb, C. B., & Rundo, L. (2022). Unified Focal loss: Generalising Dice and cross entropy-based losses to handle class imbalanced medical image segmentation. *Computerized Medical Imaging and Graphics*, *95*(May 2021). https://doi.org/10.1016/j.compmedimag.2021.102026

Yossy, E. H., Karim, S., & Rachmantyo, D. D. (2020). Development of Indonesian Language Speech Recognition Algorithm Model in Knowledge Database. *7th International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2020 - Proceedings*, 126–129. https://doi.org/10.1109/ICITACEE50144.2020.9239241

Zahrotun, L., Putri, N. H., & Khusna, A. N. (2018). The Implementation Of K-Means Clustering Method in Classifying Undergraduate Thesis Title. *Konferensi Internasional Ke-12 Tentang Sistem, Layanan, Dan Aplikasi Telekomunikasi (TSSA) Ke-12 Tahun 2018*. https://ieeexplore.ieee.org/document/8708817

Zhao, L., Shang, Z., Zhao, L., Qin, A., & Tang, Y. Y. (2019). Siamese Dense Neural Network for Software Defect Prediction with Small Data. *IEEE Access*, *7*(c), 7663–7677. https://doi.org/10.1109/ACCESS.2018.2889061

Zishan, M. A. O., Shihab, H. M., Islam, S. S., Riya, M. A., Rahman, G. M., & Noor, J. (2024). Dense neural network based arrhythmia classification on low-cost and low-compute micro-controller[Formula presented]. *Expert Systems with Applications*, *239*(June 2023), 122560. https://doi.org/10.1016/j.eswa.2023.122560.

LAMPIRAN

Lampiran I Listing kode sistem tanya jawab

```
# import necessary libraries
import warnings
warnings.filterwarnings("ignore")
import nltk
from nltk.stem import WordNetLemmatizer
import json
import pickle
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import SGD
from keras.models import Sequential
from keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout
import tensorflow as tf
from keras.models import load_model
#model untuk reinforcement learning
import numpy as np
import random
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Embedding, Bidirectional, LSTM
from collections import deque
import time

# create an object of WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# importing the GL Bot corpus file for pre-processing

words=[]
classes = []
documents = []
ignore_words = ['?', '!']
data_file = open("intents.json").read()
intents = json.loads(data_file)

 # preprocessing the json data
 # tokenization
 #nltk.download('punkt')
 #nltk.download('wordnet')
 for intent in intents['intents']:
     for pattern in intent['patterns']:

         #tokenize each word
         w = nltk.word_tokenize(pattern)
         words.extend(w)
         #add documents in the corpus
         documents.append((w, intent['tag']))

         # add to our classes list
         if intent['tag'] not in classes:
             classes.append(intent['tag'])
```

```
words = [lemmatizer.lemmatize(w.lower()) for w in words if w not in ignore_words]
words = sorted(list(set(words)))

# sort classes
classes = sorted(list(set(classes)))

# documents = combination between patterns and intents
print (len(documents), "documents")

# classes = intents
print (len(classes), "classes", classes)

# words = all words, vocabulary
print (len(words), "unique lemmatized words", words)

# creating a pickle file to store the Python objects which we will use while predicting
pickle.dump(words,open('texts.pkl','wb'))
pickle.dump(classes,open('label.pkl','wb'))


# create our training data
training = []

# create an empty array for our output
# Initialize lists to store training data
output_empty = [0] * len(classes)

# Iterate through documents
for doc in documents:
    # Initialize our bag of words and output row
    bag = []
    output_row = list(output_empty)

    # List of tokenized words for the pattern
    pattern_words = doc[0]

    # Lemmatize each word - create base word to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]

    # Create our bag of words array with 1 if word match found in current pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # Set the corresponding class index in the output_row to 1
    output_row[classes.index(doc[1])] = 1

    # Append bag and output_row as a tuple
    training.append((bag, output_row))

# Shuffle features and converting them into numpy arrays
random.shuffle(training)

# Split the training data into features (X) and labels (Y)
train_x = np.array([item[0] for item in training])
train_y = np.array([item[1] for item in training])

print("Training data created")
```

```
#model dense
# buat model trainingnya
model = Sequential()
model.add(Dense(1024, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives good results for this model
sgd = tf.keras.optimizers.legacy.SGD(learning_rate=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy', 'Precision'])

#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=1000, batch_size=16, verbose=1)
model.save('chatbot.h5', hist) # we will pickle this model to use in the future
print("\n")
print("*"*50)
print("\nModel Created Successfully!")
```

```
# Menampilkan grafik akurasi
plt.plot(hist.history['accuracy'])

plt.title('Grafik Akurasi')
plt.xlabel('Epoch')
plt.ylabel('Akurasi')
plt.show()

print("\n")
print("*" * 50)
print("\nModel Berhasil Dibuat!")
```

```
# load dan save model
model = load_model('chatbot.h5')
intents = json.loads(open("intents.json").read())
words = pickle.load(open('texts.pkl','rb'))
classes = pickle.load(open('label.pkl','rb'))


def clean_up_sentence(sentence):

    # tokenize the pattern - split words into array
    sentence_words = nltk.word_tokenize(sentence)

    # stem each word - create short form for word
    sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
    return sentence_words


# return bag of words array: 0 or 1 for each word in the bag that exists in the sentence

def bow(sentence, words, show_details=True):

    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)

    # bag of words - matrix of N words, vocabulary matrix
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:

                # assign 1 if current word is in the vocabulary position
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)
    return(np.array(bag))
```

```
def predict_class(sentence, model):

    # filter out predictions below a threshold
    p = bow(sentence, words,show_details=False)
    res = model.predict(np.array([p]))[0]
    error = 0.25
    results = [[i,r] for i,r in enumerate(res) if r>error]

    # sort by strength of probability
    results.sort(key=lambda x: x[1], reverse=True)
    return_list = []

    for r in results:
        return_list.append({"intent": classes[r[0]], "probability": str(r[1])})
    return return_list

# function to get the response from the model

def getResponse(ints, intents_json):
    tag = ints[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if(i['tag'] == tag):
            result = random.choice(i['responses'])
            break
    else:
        result = "Maaf, saya tidak memiliki jawaban untuk pertanyaan Anda saat ini."
    return result

# function to predict the class and get the response

##def chatbot_response(text):
    ##ints = predict_class(text, model)
   ## res = getResponse(ints, intents)
   ## return res
def chatbot_response(text):
    ints = predict_class(text, model)

    # Periksa apakah tidak ada intent yang cocok
    if not ints:
        return "Maaf, pertanyaan Anda tidak sesuai dengan apa yang saya pahami."

    res = getResponse(ints, intents)
    return res

import time
import speech_recognition as sr
from gtts import gTTS
import vlc
import threading

def start_chat():
    print("Robot: Hai ada yang bisa saya bantu.\n\n")

    # Initialize the VLC player
    player = vlc.MediaPlayer()

    # Create initial greeting audio
    bot_response_audio1 = gTTS(text="Hai ada yang bisa saya bantu", lang='id')
    bot_response_audio1.save("bot_response1.mp3")

    # Play the initial greeting audio
    player.set_mrl("bot_response1.mp3")
    player.play()

    # Wait for the audio to finish playing
    time.sleep(3)

    recognizer = sr.Recognizer()
```

```
while True:
    with sr.Microphone() as source:
        print("You: ", end='', flush=True)
        audio = recognizer.listen(source)

    try:
        inp = recognizer.recognize_google(audio, language='id-ID').lower()
        print("You:", inp)

        if inp == "end":
            break
        elif inp == '' or inp == '*':
            print('Please re-phrase your query!')
            print("-" * 50)
        else:
            bot_response = chatbot_response(inp)

            # Convert the bot response to speech
            bot_response_audio = gTTS(text=bot_response, lang='id')
            bot_response_audio.save("bot_response.mp3")

            # Wait for the previous audio playback to finish before playing the new one
            if player.get_state() == vlc.State.Playing:
                player.stop()

            player.set_mrl("bot_response.mp3")
            player.play()

        print("Bot:", end='', flush=True)
        for karakter in f" {bot_response}":
            print(karakter, end='', flush=True)
            time.sleep(0.07)
        print('\n')
        print("-" * 50)

except sr.UnknownValueError:
    print("Bot: Maaf, saya tidak mengerti apa yang diucapkan. Tolong ulangi.")
    print("-" * 50)
except sr.RequestError as e:
    print(f"Bot: An error occurred: {str(e)}")

    print("-" * 50)

start_chat()
```