## LABORATORY 4: CHATBOT USING NEURAL NETWORKS

**Learning Outcomes**

By the end of this laboratory, student should be able to

* Gain hands-on experience in building a simple chatbot with intent recognition.
* Understand the importance of data preprocessing in NLP tasks.
* Gain insights into how neural networks can be used for natural language understanding.
* Experiment with model parameters and training data to observe their impact on the chatbot's performance.
* Explore the challenges and possibilities of improving chatbot interactions, including contextual understanding.

**Activities**

* Data Exploration and Modification (explore the intents.json file and manually add a new intent with patterns and responses).
* Tuning Model Parameters (modify the assigned parameter and observe how it affects the model's training and chatbot responses).
* Research on another type of simple chatbot (discuss the advantage and shortcomings of both chatbot).

**Review**

* Summarize the key concepts covered during the lab, including natural language processing (NLP), intent recognition, and the basics of neural networks.
* Discuss the importance of data preprocessing in NLP tasks. Highlight specific steps taken in the code to preprocess the training data.
* Emphasize how the chatbot predicts intents based on user input and the confidence threshold used to determine the appropriate response.
* Discuss real-world applications of chatbots and NLP technologies. Explore how these technologies are used in industries such as customer service, healthcare, and finance.

**Equipment**

* Laptop (non-gpu will do)
* IDE (VSCode, Jupyter notebook, Pycharm, etc.)
* Virtual Environment (Venv, Conda)

Getting started

1. Download Lab 4 folder from Brightspace.
2. Inside the Lab4\_Code folder, check if have these 5 files “training.py” [1], “inference.py” [1], “intents.json”, “tfidfChatbot.py” [2], “5GAIoT.txt”, “quantized\_falcon.py” [3]

A screenshot of a computer

Description automatically generated

1. Download VSCode from this link: <https://code.visualstudio.com/download>
2. Launch VSCode. Click file, click open folder, select **Lab4\_Code** folder, and open it.

**\*\*OPEN ONLY THE CODE FOLDER IN VSCODE**

A screenshot of a computer

Description automatically generated

1. Click on training.py and open it up. You will see a pop-up notification to install python extension. Click “Install”.

A screen with text on it

Description automatically generated

*Figure 1: Pop-up Notification on python extensions*

**Alternatively,** you can directly install the python extensions by typing the name “python” when you clicked on the “extensions” icon.

A screenshot of a computer program

Description automatically generated

1. You will need to download a python interpreter (preferably version 3.10.5) from this link: [Python from python.org](https://www.python.org/downloads/)

A screenshot of a computer

Description automatically generated

1. Set up a virtual environment in VScode with python interpreter version 3.10.5. **Ctrl + Shift + P** toopen command palette. **Click “Python: Select Interpreter” > “Create Virtual Environment” > “Venv” > “Delete and Recreate” > “Python 3.10.5 64-bit”**. (Venv is lightweight compared to Conda, which is more suitable for this lab). The alternative way is to use this command in the same directory as the Code folder to make Venv in command prompt “**python -m venv .venv”** (Not recommended because your file path may be wrong)

A screenshot of a computer program

Description automatically generated

*Figure 2: Setting up a Virtual Environment*

**\*\*CLOSE AND REOPEN VSCODE IF YOU CANNOT SEE THE PYTHON INTERPRETER**

**Launching Virtual Environment**

1. Open up a new terminal in VScode and execute this command:

**.\.venv\Scripts\activate**

1. If encounter “cannot be loaded because running scripts is disabled on this system. Use this command:

**Set-ExecutionPolicy Unrestricted -Scope Process**

Then try the previous command again. If you see the (.venv), it means that the virtual environment is activated.

A screen shot of a computer

Description automatically generated

*Figure 3: Launching Virtual Environment*

**\*\*IF YOU CANNOT CREATE A VIRTUAL VENV OR CANNOT LAUNCHED the VENV, USE A DIFFERENT PYTHON INTERPRETER 3.9/3.11**

**Install Necessary Libraries (DO NOT START THIS WITHOUT LUANCHING VENV)**

1. Install scikit-learn.

**pip install scikit-learn**

1. Install natural language toolkit.

**pip install nltk**

1. Install colorama.

**pip install colorama**

1. Install Google Deep Learning (DL) framework tensorflow version 2.15 library.

**pip install tensorflow==2.15**

\*\*If cannot, use another python interpreter version 3.9 or 3.11 (tensorflow v2.15 only works on python 3.9-3.11)

**A computer screen with white and green text

Description automatically generated**

*Figure 4: Downloading Python Libraries for Chatbot*

Read more on Google tensorflow here: [https://www.tensorflow.org/](https://www.tensorflow.org/learn%20and%20their%20competitor%20)

Meta pytorch here: <https://github.com/pytorch/pytorch>

**Play around with Chatbot!**

1. Run the training.py by executing this command in a terminal:

**.\.venv\Scripts\python training.py**

1. You will start seeing the Deep Learning model is being trained.

A screenshot of a computer

Description automatically generated

Try running the training.py multiple times until you see most of the epochs have a accuracy of 1.0

1. After it finished training, run the inference.py by executing this command in a terminal:

**.\.venv\Scripts\python inference.py**

A screen shot of a computer code

Description automatically generated

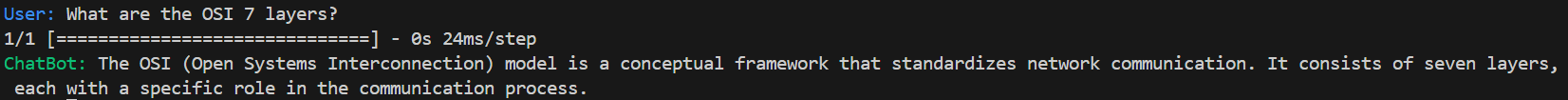
1. Key in random questions like “How to bake a bread?”

A black screen with white text

Description automatically generated

You can see that it gives an answer that is completely wrong, this means that the training data is insufficient.

1. Try another question. You will see that it is able to answer the question!



1. To exit the chatbot, type in the word “quit”.
2. Go to “intents.json” file and search “chatbot”. You will find that the exact response above is in the file.

A screenshot of a computer

Description automatically generated

How does chatbot answer to user’s input?

1. JSON in chatbots is typically used to define intents, responses, data storage, configuration settings, dialog flow, context, and training data for machine learning models. It plays a crucial role in structuring and managing data that supports chatbot interactions.
2. Each intent has several properties, and these intents are designed to help the chatbot understand and respond appropriately to user inputs.

A screenshot of a computer

Description automatically generated

1. **“tag”** is a unique identifier for each intent. It plays a crucial role in categorizing and distinguishing different types of user inputs or queries.
2. **“patterns”** is an array containing different patterns or phrases that a user might input to express this particular intent. For example, the "greeting" intent includes patterns like "Hi," "How are you," etc.
3. **“responses”** is an array containing possible responses that the chatbot can use when it recognizes this intent based on user input. For example, the "greeting" intent has responses like "Hello!" and "Good to see you again!"

**Let us dive into the model training code**

How is the training data from intent.json loaded and pre-processed?

1. The code reads the "intents.json" file and loads its content into the data variable.

A close up of words

Description automatically generated

1. The code extracts training sentences, labels, and responses from the intents.json file. The 'patterns' are sentences or phrases associated with specific 'intents,' and 'responses' are potential replies for those intents.

A screen shot of a computer program

Description automatically generated

1. Labels are encoded using sklearn's LabelEncoder to convert string labels into numerical representations.

A close-up of a code

Description automatically generated

1. The code tokenized and padded the training sentences to create numerical sequences suitable for input to a neural network. The primary reason this tokenization matters is that it helps machines understand human language by breaking it down into bite-sized pieces, which are easier to analyse [4].

A computer screen shot of a computer code

Description automatically generated

*Figure 5: How neural networks can be used for natural language understanding*

1. This section defines a simple neural network model with an embedding layer, global average pooling, and densely connected layers with ReLU activation. The output layer uses softmax activation for multiclass classification.

**model = Sequential():** Creates a sequential neural network model. This type of model allows the definition of a linear stack of layers.

**model.add(Embedding(vocab\_size, embedding\_dim, input\_length=max\_len)):** The main purpose of the Embedding layer is to map each word in the vocabulary to a unique vector in a high-dimensional space. The dimensionality of this space is a hyperparameter that is often set based on the size of the vocabulary and the complexity of the task. The size of the embedding dimension is a hyperparameter and should be chosen based on the size of the vocabulary and the complexity of the task.

**model.add(GlobalAveragePooling1D()):** Adds a GlobalAveragePooling1D layer to a neural network model. This layer is often used in conjunction with sequences, such as text data in natural language processing (NLP) tasks.

Unlike traditional pooling layers that operate on local regions, global average pooling computes the average value of each feature (channel) over the entire sequence. In the context of 1D sequences, such as word embeddings in NLP, it computes the average along the entire sequence length.

**model.add(Dense(16, activation='relu')):** Adds another dense layer with 16 units and ReLU activation.

**model.add(Dense(num\_classes, activation='softmax')):** Adds the output layer with as many units as there are unique classes in the output data. Each neuron in this layer corresponds to a class. The softmax activation function is used for multi-class classification, as it provides probabilities for each class.

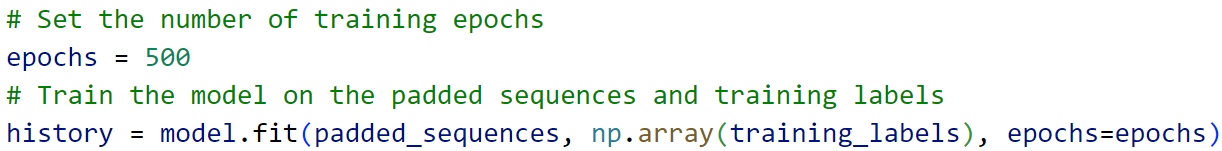
**model.compile(loss=’sparse\_categorical\_crossentropy’, optimizer='adam', metrics=['accuracy']):** Compiles the model. It specifies the categorical sparse\_categorical\_crossentropy as the loss function (suitable for multi-class classification), the Adam optimizer, and accuracy as the evaluation metric.

A screenshot of a computer code

Description automatically generated

1. The model undergoes training using the provided **'padded\_sequences’** which serve as the input data for the training process. These **'padded\_sequences'** consist of sequences of 'word indices,' representing the numerical representation of words within a predefined vocabulary. These indices are obtained through tokenization and vocabulary construction. The sequences are padded to a fixed length using the **pad\_sequences** function, where each row corresponds to a sequence.

The target labels for training are represented by **np.array(training\_labels)**, encapsulating the ground truth or actual values associated with the input data. Ground truth provides the genuine and accurate information about what the model aims to predict, serving as a reference against which the model's predictions are compared. The neural network is trained to map the input sequences **(padded\_sequences)** to these target labels, with each element in **np.array(training\_labels)** corresponding to the label (class) for the respective sequence in **padded\_sequences**.



1. The trained model is saved to a file named "**chat\_model**" and the fitted tokenizer and label encoder are saved as pickle files for future use.

‘Pickle’ provides a convenient way to serialize and deserialize Python objects. Serialization is the process of converting a Python object into a byte stream, and deserialization is the process of reconstructing the original object from a byte stream. The primary purpose of using pickle is to save and load Python objects, making it easy to store data structures, models, and other complex objects persistently.

A screen shot of a computer program

Description automatically generated

Let us dive into the inference code.

1. The **chat()** function encapsulates the chatbot logic. It loads the pre-trained neural network model (**chat\_model**) using Keras. It also loads the tokenizer and label encoder objects from the saved pickle files (tokenizer.pickle and label\_encoder.pickle). These objects were used during the model training phase. Maximum length for input sequences should match the length used during model training.

A screen shot of a computer code

Description automatically generated

33. The code enters an infinite loop marked by while True: to create an interactive chat session. Within this loop, the program continuously prompts the user for input, printing a message in light blue. If the user inputs "quit," the loop is terminated, and the chat session ends. For each user input, the code preprocesses the text and leverages a pre-trained neural network model to predict the intent of the input. The predicted intent is then decoded using a label encoder. Subsequently, the program searches for the corresponding intent in the loaded data and prints a response from that intent. The conversation continues until the user decides to quit by typing "quit." The code provides a simple and interactive chatbot experience, demonstrating the integration of a trained model for natural language understanding.A screenshot of a computer code

Description automatically generated

**Activity 1)** Edit the intents.json file to answer the question “How to bake a bread?”, then execute the training.py and inference.py.

{

"tag": "Baking\_Bread",

"patterns": [

"How to bake a bread?",

"Explain in steps on how to bake a bread.",

"Tell me how to bake a perfect bread."

],

"responses": [

"Mix all the ingredients together to form a dough, let it rise until doubled in size,then bake in a preheated oven until golden brown and hollow-sounding when tapped on the bottom.",

"Here are the following steps: \n Step 1: Mix flour, salt, yeast, and water to form dough \n Step 2: Let dough rise until doubled in size \n Step 3: Shape dough and let it rise again \n Step 4: Bake in a preheated oven until golden brown and hollow-sounding when tapped.",

"To bake the perfect bread, mix high-quality ingredients, knead until smooth, let it rise twice, shape, score, bake until golden, then cool before enjoying."

],

"context\_set": ""

},

**Insert the following code above into the intents.json file within the intents. Make sure that the sentences within the quotation marks are in a single line in VScode.**

**A screenshot of a computer program

Description automatically generated**

*Figure 6: Chatbot can answers the question correctly now*

**Dropout layer:** randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent **overfitting.** Inputs not set to 0 are scaled up by 1 / (1 - rate) such that the sum over all inputs is unchanged.

**Flatten layer:** Flattens the input. Does not affect the batch size.

**Activity 2)** Experiment with tuning of the Neural Network model’s layers. Try adding **dropout** and **flatten** layers. Try changing the number of **neurons** in the dense layers. Create a new python file named training2.py and modify that file. Then run the training script multiple times and observe the accuracy of epochs. Are there fewer or more epochs having an accuracy of 1.0? (Hint: Import Dropout, Flatten and use model.add() to add dropout and flatten layers)

**Activity 3 and 4 are self-directed learning (SDL)**

**Activity 3)** Refer to Lab 1 to 3 and improve the key-pair values in intents.json file to make this chatbot your “supervised quiz” mate (feel free to create a new intents.json file).

**Activity 4)** Read up the Readme.md file in this page, <https://github.com/parulnith/Building-a-Simple-Chatbot-in-Python-using-NLTK/tree/master>. This is another simple chatbot but without neural network. It uses rule-based techniques and TF-IDF cosine similarity for response generation [2]. The chatbot's responses are based on similarity to existing patterns in the corpus. Run “tfidfChatbot.py” and see the differences in the behaviour of the chatbot.

Qn 1) Why is it that the chatbot can answer, “What is a chatbot”?

**Answer: The tag, patterns and responses for chatbot are defined in the intents.json file.**



Qn 2) How will you be using this chatbot to study for your upcoming supervised quiz?

**Answer: Modify the tag, patterns and responses for supervised quiz.**



Qn 3) Suggest 2 other ways to make the chatbot more “context-aware”.

**Answer: Keep track of conversation state, contextual words**

Fig. 1. A flowchart showing how a chatbot engine processes an input string and gives a valid reply.

A diagram of a program

Description automatically generated

This lab has been inspired by Amila Viraj [1].

**Appendix**

Here is a Large Language Model (LLM) - Falcon code [3] developed by Technology Innovation Institute (TII), which could run on a laptop with 16 GB RAM without the need of GPU. It is the lightest out of all LLM Falcon model at 7 billion parameters. It was one of the most popular open-source LLM in 2023. Feel free read up the resources here and to try to run quantized\_falcon.py.

“vilsonrodrigues/falcon-7b-instruct-sharded” is the pre-trained model hosted on the Hugging Face Model Hub [4].

It is normal for this chatbot to run for at least 10 minutes as we are running it on a locally.

Falcon-refinedweb is the dataset used to train this chatbot [5] [6].



**References**

[1] Viraj, A. (2020) How to build your own chatbot using Deep Learning, Medium. Available at: https://towardsdatascience.com/how-to-build-your-own-chatbot-using-deep-learning-bb41f970e281 (Accessed: 27 January 2024).

[2] P. Pandey, “Parulnith/building-a-simple-chatbot-in-python-using-NLTK: Building a simple chatbot from scratch in Python (using NLTK),” GitHub, https://github.com/parulnith/Building-a-Simple-Chatbot-in-Python-using-NLTK/tree/master (accessed Nov. 12, 2023).

[3] Rodrigues, V. (2023) Run your private LLM: Falcon-7B-Instruct with less than 6GB of GPU using 4-bit quantization. Available at: <https://vilsonrodrigues.medium.com/run-your-private-llm-falcon-7b-instruct-with-less-than-6gb-of-gpu-using-4-bit-quantization-ff1d4ffbabcc> (Accessed: 27 January 2024).

[4] Awan, A.A. (2023) What is Tokenization? Available at: https://www.datacamp.com/blog/what-is-tokenization. (Accessed: 27 January 2024).

[5] tiiuae/falcon-7b-instruct · Hugging Face. (2023) Available at: https://huggingface.co/tiiuae/falcon-7b-instruct (Accessed: 27 January 2024).

[6] The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only. Available at:

https://doi.org/10.48550/arXiv.2306.01116 (Accessed: 27 January 2024).