AN INTELLIGENT DEEP CONCURRENT MODEL FOR DIALOGUE ACT CLASSIFICATION, INTENT DETECTION AND SLOT FILLING FOR AN AUTOMATIC INQUIRY SYSTEM

by

GOKUL S	**2018103026**
STEVEN F GILBERT	**2018103071**
SRIHARI S	**2018103601**

A project report submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ANNA UNIVERSITY

CHENNAI 600025

JUNE 2022

BONAFIDE CERTIFICATE

Certified that this project report titled **AN INTELLIGENT DEEP CONCURRENT MODEL FOR DIALOGUE ACT CLASSIFICATION, INTENT DETECTION AND SLOT FILLING FOR AN AUTOMATIC INQUIRY SYSTEM** is the *bonafide* work of **GOKUL S (2018103026)**, **STEVEN F GILBERT (2018103071)** and **SRIHARI S (2018103601)** who carried out project work under my supervision, for the fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge and belief, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on this or any other candidate.

PLACE: Chennai
DATE:

Dr. S. Chitrakala
Professor
Department of Computer Science and Engineering
Anna University, Chennai – 25

COUNTERSIGNED

Head of the Department,
Department of Computer Science and Engineering,
Anna University Chennai,
Chennai – 600025

ACKNOWLEDGEMENT

We express our deepest gratitude to our guide , Dr. S. Chitrakala , Professor for guiding us through every phase of the project. We appreciate her thoroughness, tolerance and ability to share her knowledge with us. We thank her for being easily approachable and quite thoughtful. Apart from adding her own inputs, she has encouraged us to think on our own and give form to our thoughts. Without her immense support through every step of the way, we could never have made it to this extent. We express our thanks to the panel of reviewers Dr.T.V.Gopal, Professor, Dr.Uma Maheshwari, Associate Professor and Dr.Angelin Gladston, Associate Professor for their valuable suggestions and critical reviews throughout the course of our project. We would like to extend our thanks to the HOD of the Department of Computer Science and Engineering, Dr.S.Valli,Professor for providing us with the required facilities to see our work to successful completion. We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us for undertaking this course in such a prestigious institution.

Gokul S **Steven F Gilbert** **Srihari S**

ABSTRACT

The work is intended to tackle a vital field that lies at the intersection of speech processing and natural language processing: Spoken Language Understanding (SLU). Its idea is to understand the essence of machine-directed human speech in order to facilitate its further processing and take on board its cognitive impact. The proposed system is CIDIS –Concurrent Intelligent Model for Dialogue Act Classification, Intent Detection and Slot Filling, that uses a deep concurrent multi-task paradigm to perform the three fundamental tasks of the SLU domain: Dialogue Act Classification, Intent Detection and Slot Filling. Since the model is orchestrated in a multi-task fashion, every task interacts with the other to have a global understanding of the input query. It follows an intelligent encoding strategy involving concatenation of the query's BERT and CharCNN embedding to handle all possible edge cases and ambiguities involved in human speech queries. This intelligent encoding is passed through a Stacked BiLSTM architecture followed by task-specific attention layers. The three supplementary outputs are in turn fed to the final module that generates the expected query response in real-time based on the dialogue act, intent and slot. The developed models are evaluated against standard benchmark datasets like ATIS, TRAINS and FRAMES and the achieved state-of-the-art performances are eventually tabulated. The various applications of such a system like CIDIS includes voice-assisted enquiry systems, in which queries are just to be spoken and the system can extract the relevant information and present the results. Another application is a real-time control system in which time is sensitive, so it can make use of the ability for the system to understand spoken queries rather than a lengthy set of actions to be performed to obtain the same results.

Keywords: Spoken Language Understanding , Multi-task Learning , Intent detection , Slot Filling , Dialogue Act Classification.

ABSTRACT

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CIDIS	Concurrent Intelligent Model for Dialogue Act Classification, Intent Detection and Slot Filling
SLU	Spoken Language Understanding
DAC	Dialogue Act Classification
ID	Intent Detection
SF	Slot Filling
LSTM	Long Short Term Memory
BiLSTM	Bidirectional Long Short Term Memory
CRF	Conditional Random Field
RNN	Recurrent Neural Network
NER	Named Entity Recognition
IoT	Internet Of Things
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CharCNN	Character Convolutional Neural Network
OOV	Out of Vocabulary
POS	Part of Speech
BIO	Beginning Inside Outside
DNN	Deep Neural Network
RE	Regular Expression

CHAPTER 1

INTRODUCTION

1.1 GENERAL OVERVIEW

For any modern dialogue system, irrespective of being conversational or not, understanding the language spoken and extracting information out of it before its formal analysis is a key component and that is where the domain of Spoken Language Understanding (SLU) begins. The proliferation of Internet of things and smart devices has brought about a surge in these dialogue systems, necessitating systems which can handle domain specific idiosyncrasies.

The methodologies involved in this domain are prevalent in various other real time systems like Interactive Voice Response and Human Computer Interfaces. Catering to a cogent response for the request in these scenarios have been achieved using the fundamental tasks of Natural Language Understanding, which includes Dialogue Act Classification, Intent Detection and Slot Filling.

From using individual deep learning models [1],[2],[3], the industry has shifted to the trend of using joint models that are pipelined to each other so that each task has a predecessor to pick an essence of the overall task. The skill to identify the essence of the discourse involved is of utmost importance in this domain of Spoken Language Understanding. This is achieved using a commensurate amalgamation of the syntax, semantics and pragmatics involved, which are represented as the dialogue act of the utterance. It denotes the type of speech act involved. Recognition of these dialogue acts which can also be termed as Spoken Utterance classification aids in fabricating a coherent response for the user. With a model that can accurately demarcate the dialogue act of the

user's utterance, the requested queries are embraced in a better way.

Subsequently, the next step involved after discerning the dialogue act is the task of detecting the intention of the user. More formally, Intent detection is the mechanism of identifying the ultimate goal of the utterance. It is generally modeled as a semantic utterance classification problem. Inherently domain-oriented, modeling a resilient intent detector is crucial as the fallacies in this step have an enduring effect on the overall system, which may lead to fallacious responses.

Furthermore, the refined details in the input utterance are extracted as slots in the process of Slot Filling. This task is modeled in a sequence labeling fashion, where each word in the utterance is denoted using an appropriate label. This facilitates in transforming user's intention into a more obvious instruction to complete the information collection. Based on the information extracted, the system resorts to appropriate actions, in order to assist the user in achieving their needs.

In order to exploit the semantic correlations between dialogue acts, intents and slots, multi task models are proposed. These are characterized by shared parameters among the tasks and strives to ameliorate the performance of each by leveraging useful information among them.

The proposed architecture follows an approach wherein a multi-task model post following an intelligent encoding strategy attends to the important parts of the text to come up with the final classes of outputs that they belong to. Each class decides its own output template whose slots are then filled by scraping information from associated websites that host the required content.

1.2 ABOUT THE PROJECT

The focus is mainly on understanding the meaning of a user query to come up with a cohesive response to it in an efficient, and more importantly, accurate manner. The ability to concisely fetch the nuances involved in the query is imperative for these systems. This calls for a robust intelligent model which is capable of understanding the context from quotidian commands, thereby a more user-friendly product.

1.2.1 Objective

To design and develop multi-task model using global representation of the input query for parallely performing the first three tasks followed by the added application module:

- **Dialogue Act Classification** : To understand the mode of the query.
- **Intent Detection** : To understand the user's intent (i.e) what the user is expressing as the goal.
- **Slot Filling** : To obtain the parameters used in the query to help generate the response.
- **Query Response Retrieval** : Using the above the three mentioned tasks to generate the appropriate response for the user query.

1.2.2 Problem Statement

- Performing "intelligent" feature encoding thereby achieving a robust information extraction strategy for the input.

- Adding sense of context to the same using sentence representation techniques like Stacked Bi-LSTMs/ Bi-GRUs.
- Producing an appropriate intent class with corresponding slot types filled.
- Constructing a template for the generated intent class, which is also based on the number and type of the slot values that are filled.
- Filling the actual slot values through scraping from intent-specific credible websites.

1.2.3 Need for the system

To comprehend the need for an intelligent encoding strategy, the modality needs to be fundamentally understood. In order to exploit the semantic correlations between dialogue acts, intents and slots, multi task models are proposed. These models are characterized by shared parameters among the tasks and strives to ameliorate the performance of each by leveraging useful information among them. In such a situation, the encoding should be such that different senses of the word as well as the ambiguities in sentence structures need to be analysed and accounted for. Hence, interesting approach is proposed where ambiguities are attended by the BERT of the concatenated embedding through subword tokenization procedure and OOV words in the input queries are taken care by the CharCNN part.

1.2.4 Challenges of the system

All the currently used models have some common issues like poor semantic understanding of human-like inputs and less robust to ambiguities in speech.

1.2.5 Organization Of Thesis

The outline of the thesis is as follows:

Chapter 2 : Literature Survey of related works

Chapter 3 : Detailed description of the proposed system and each functioning modules.

Chapter 4 : Discussion of results and analysis.

Chapter 5 : The final conclusion and exploration of other future works that can be implemented.

CHAPTER 2

LITERATURE SURVEY

One of the major challenges in Natural Language Understanding is to achieve efficient results with a meager amount of data. To circumvent this issue, Dadas et. al [4] worked on a data enrichment strategy where external lexicons assisted in generating random mutations based on heuristics. Subsequently, Su et. al [5] put forth a semi-supervised two agent learning approach which operated on cyclic feedback between the models. A Dual Learning - Two Agent Game is discussed which operates on the feedbacks provided by the natural language understanding model and the semantic to sentence generation model. The local and global representations of the vector forms of the Intent-Slot pairs are fetched to emphasize on the critical aspects of the sentences, a semantically controlled Long Short-Term Memory cell is used to incorporate attention. This approach has pioneered applying dual learning algorithm in the field of intent detection and slot filling, and has also provided a solution to the data sparsity problem in the domain of natural language understanding by exploiting semantic forms and unlabeled data. However, the proposed framework is skeptical of the backbone model of the NLU task and different NLU models have not been explored. The bottleneck where this approach hits the performance limit has not been observed. Appropriate rewards to develop a better model can open the possibilities of deploying this approach across domains.

Though the rudimentary approaches adopted machine learning , it was the advent of Deep Learning which expedited more accurate results in this realm of intent detection and slot filling. Experiments in this domain were initially carried out separately where the models were trained independently. However, a paradigm shift in this approach was witnessed with the inclusion of

joint models which are capable of utilizing the refined details from all the tasks, thereby comprehending the other.

For modeling the interdependent relations between the tasks in unified manner, Qin et. al [6] proposed the co-interactive transformer approach which was experimented on the benchmark SNIPS and ATIS corpus. Further Maujama et. al [7] worked in a hierarchical joint fashion, where character level Convolutional Neural Network (CNN) embedding framework aided in dealing with infrequent terms. Moreover Hui et. al [8] work on the continual learning interrelated model caters to semantic details with varied characteristics. Subsequently Kane et. al [9] approach incorporating CNNs with Long Short Term Memory (LSTM) and Conditional Random Field (CRF) enabled modeling the tasks independently and together. CRFs were also used in the masked graph based approach worked upon by Hao et. al [10]

Joint modelling of Intent detection and Slot filling is incorporated using a hybrid architecture comprising of Bidirectional Gated Recurrent Units, Capsule Neural Networks and Attention mechanism. This approach enhanced the semantic capture capability of each of the individual models and combining specialized models built independently to achieve a complete joint task, optimal performance has been recorded on each task. The above implemented model has performed better than the Region Based Convolutional Neural Network for the intent identification task. Slot filling task starts with Bidirectional Gated Recurrent Units for encoding, trailed by Conditional Random Field predicting the most probable slot label. This approach has been transferred to the medical field with the construction of a clinical voice assistant using the joint task of clinical domain detection and entity recognition. MedBERT which adapts the Bidirectional Encoder Representations from Transformers for the medical field has been used with the two forementioned models on Intent Detection. However, the performance of this model on low resource task has been subpar.

Incorporating supplementary details of the text helps ameliorate the performance of the tasks. Waheed et. al [11] worked on a multi task approach which encompassed an amalgamation of state-of-the-art deep neural networks like the CNN, Recurrent Neural Network (RNN) and regular expressions. On the other hand, Jose et. al [12] approach to juxtapose the intents with the slots aided in dealing with multi-domain joint semantic frame parsing. Part of Speech (POS) and Named Entity Recognition (NER) Tags were utilized in [13] to exploit hierarchical relations between the tasks.

Other accomplishments in this field dealt with domain specific system. Pin et. al [14] proposed a hybrid joint model utilizing Bidirectional Gated Recurrent Units and CRF for an IoT based voice interaction system. Stoica et. al [15] put forth a home assistant system by working on a set of language and data related challenges.

In order to extract more contextual details, many experiments were conducted by using the attention mechanism with the deep learning architectures. Peng et. al [16] proposed a position aware multi head masked attention strategy which catered to intents and slots explicitly. The main objective of this model is to explore the explicit interactions between the Intent detection and Slot filling to improve performance. It models the interactions between the word encoding features and intent-slot features to generate better contextual features. The multi-head attention mechanism improves robustness by capturing sentence level-semantics. The model contains three main components: Shared word encoder, Multi-head attention for intent detection and PMMAtt for the slot decoder. With the encodings from the BiLSTM, the intent detection modelled as a sentence-level classification, is passed to multi-head attention, which provides attention scores for each word in a sentence based on contribution towards the context of the input. The output from this is given to the linear classifier for intent detection. The PMMAtt requires three inputs: word encoding, intent

embedding and left side slot embedding sequence. Together with the multi-head attention with masking for parallel computation and the positional encodings, a content-based and position-based attention is obtained. Each head produces a context vector as a weighted sum of the intent and slot features. While the model experiments with interactions between the intents and slots, it can be explored more as the interactions were less and the overall model produced a trivial improvement over the existing model. On the other hand, Tingting et al. [17] worked on the Hohhot bus query dataset using attention with Bi-directional LSTMs.

A previous work [18] on the multi task learning approach for the three tasks used stacked Bi-LSTMs as the shared layer. Despite showing promising results over other alternative methodologies, it falls short in incorporating semantic and syntactic information to the fullest extent.

Observations from the literature survey

- A lack of semantic information can pose to be an issue when it comes to the system ability to draw out information to understand contextual information.
- No interactions between the intent and slot filling will yield a lower understanding of the input query can be solved with multi-task style system.
- Dialogue state tracking is essential to understanding, without which the system's understanding of the input query is incomplete.

CHAPTER 3

SYSTEM DESIGN AND IMPLEMENTATION

3.1 PROPOSED SYSTEM

The proposed system - Concurrent Intelligent Model for Dialogue Act Classification, Intent Detection and Slot Filling (CIDIS) is natural language understanding model that attempts to comprehend the internal structure and meaning of an input natural language query through three subtasks namely Dialogue Act Classification, Intent Detection and Slot Filling in a parallel multi task manner. Based on the dialogue act and intent, predetermined templates are defined and are filled in real-time by scraping the required details from credible sources. The contents to be scraped are clearly dependent on the slots that were determined in the previous step. Initially, the text input is transformed to an embedding vector space by passing it simultaneously to two models, a BERT-based paraphrase miniLM-L1-V2 to perform subword tokenization and retrieve an embedding that takes care of ambiguities and contexts, and a trained CharCNN model to perform character-wise tokenisation and embedding. The embeddings are then concatenated to retrieve an intelligent representation for the input sentence. This when passed through a stacked BiLSTM layer gives the structured sentence representation. Now, since a multi-task paradigm is followed here, each task has its own task-specific layers. Here, the task-specific layers are self-attention layers and deep neural networks whose hyperparameters are altered and experimented on. Once the BIO-tagged slots are converted to their elementary format by adding corresponding POS tags and parsing the tree in-order, the intermediate outputs are then fed to the final application module that retrieves the required query response real time by scraping the information from a credible source. Here, there is better semantic understanding of the input

query as BERT embeddings are used and the multi-task architecture ensures that there are better interactions between the three supplementary NLU tasks.

3.2 SYSTEM MODELS

3.2.1 Use Case Diagram

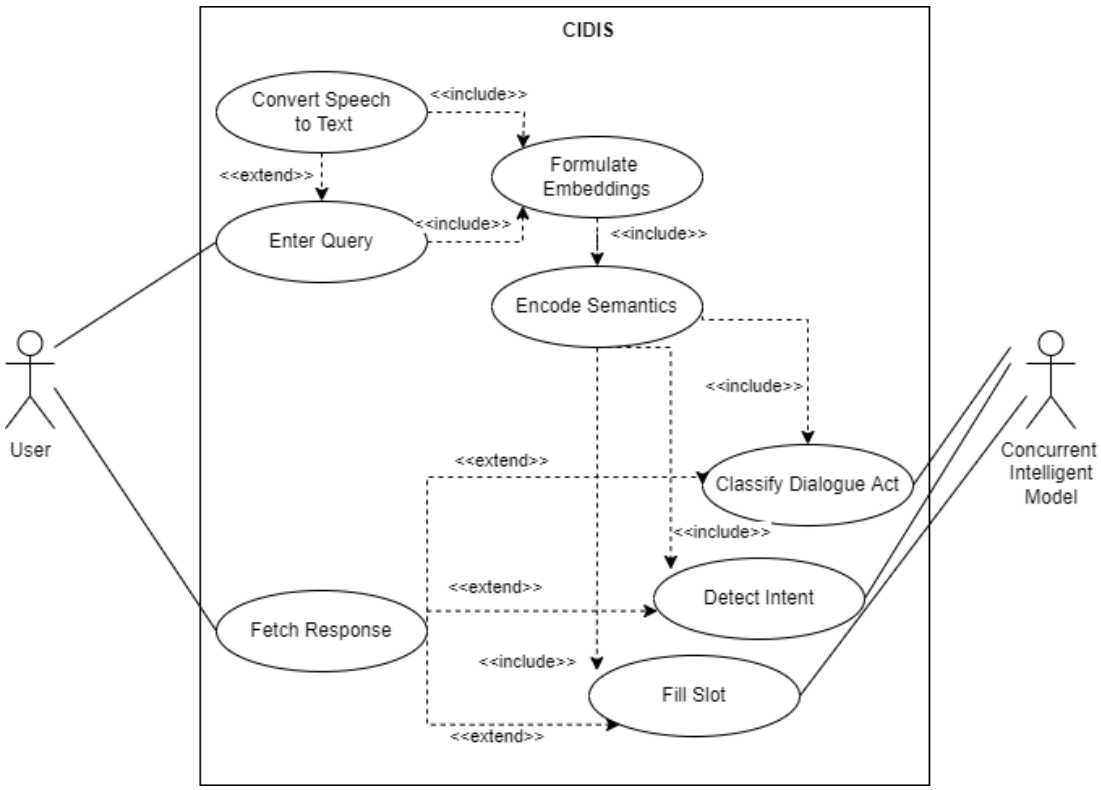


Figure 3.1: Use Case Diagram

As shown in Figure 3.1, the primary actor of the system is the User and the secondary actor is the Concurrent Intelligent Model. There are 8 use cases in total which starts of with an extend relationship between speech-to-text conversion and the query entering use case as speech-to-text is just an alternate pathway to give inputs to the system. This is followed by include relationships to Formulate Embeddings, Encode Semantics and to the three supplementary tasks as this pathway is mandatory. Fetch Response use case is connected to the three tasks through an extend relationship as in some cases, the user might be

interested only in the supplementary outputs and not the final real-time query response.

3.2.2 Sequence Diagram

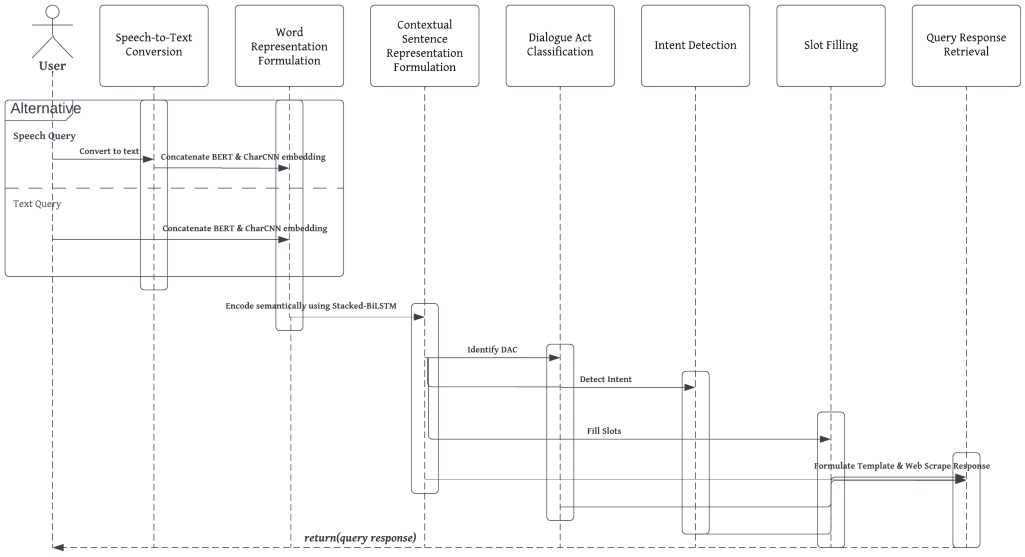


Figure 3.2: Sequence Diagram

In Figure 3.2, the flow of messages between different modules of the system are clearly depicted. The alternate frame in the beginning potrays the fact that the input to the system could either be in a speech or text format, as it would eventually be converted to text at some point. This is followed by formulating the word embedding and injecting it to the Stacked BiLSTMs to represent the sentence as a whole in a contextual manner. At this point, the multi-task architecture comes into picture by forking to the three supplementary tasks and finally collecting the outputs from all tasks to formulate an appropriate query response.

3.3 SYSTEM ARCHITECTURE

Figure 3.3 depicts the overall system architecture that CIDIS follows.

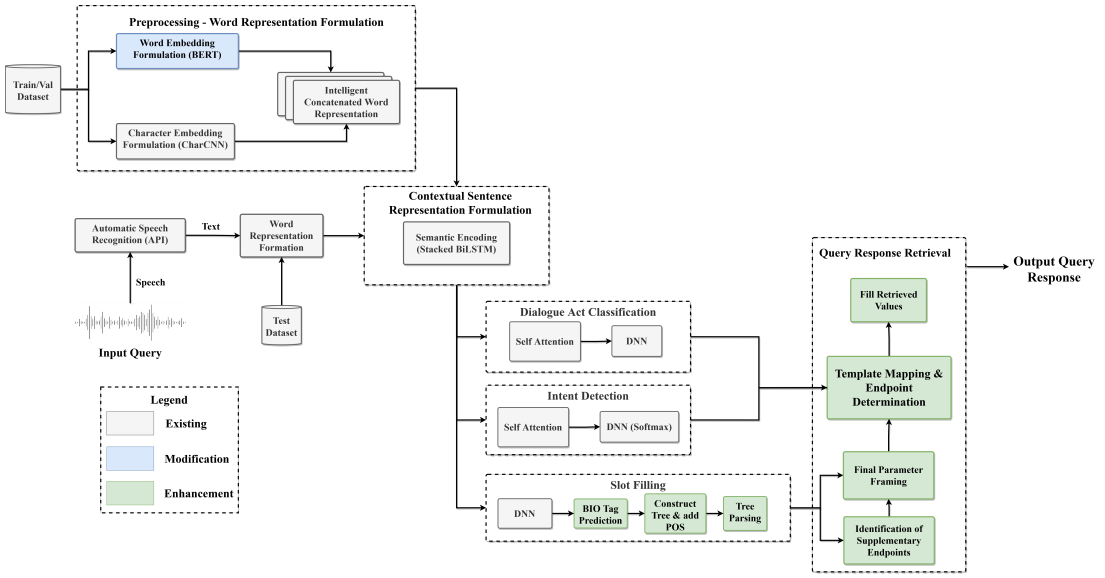


Figure 3.3: System Architecture

It starts off with Module 1 dealing with formulating an intelligent embedding that is a concatenation of the BERT embedding through sub-word tokenization and character embedding using a trained CharCNN model. This is followed by Module 2 that takes in the word level embeddings and puts them together through Stacked BiLSTM to get contextual sentence representation formulation. Now, three tasks branch out in a parallel fashion by utilizing the common hidden level representation. Modules 3 and 4 contain self-sttention layers and DNNs to affine transform the common representations to outputs of required dimensions. Module 5 contains an additional BIO-tag removal where a tree is constructed and parsed to obtain the final slots in their clear and concise form. This is followed by Module 6 that takes the supplementary outputs and web scrapes information form credible real time sources to get back with the required query response.

3.4 LIST OF MODULES

- Preprocessing – Word Representation Formulation

- Contextual Sentence Representation Formulation
- Dialogue Act Classification
- Intent Detection
- Slot Filling
- Query Response Retrieval

3.5 MODULES

3.5.1 Preprocessing – Word Representation Formulation

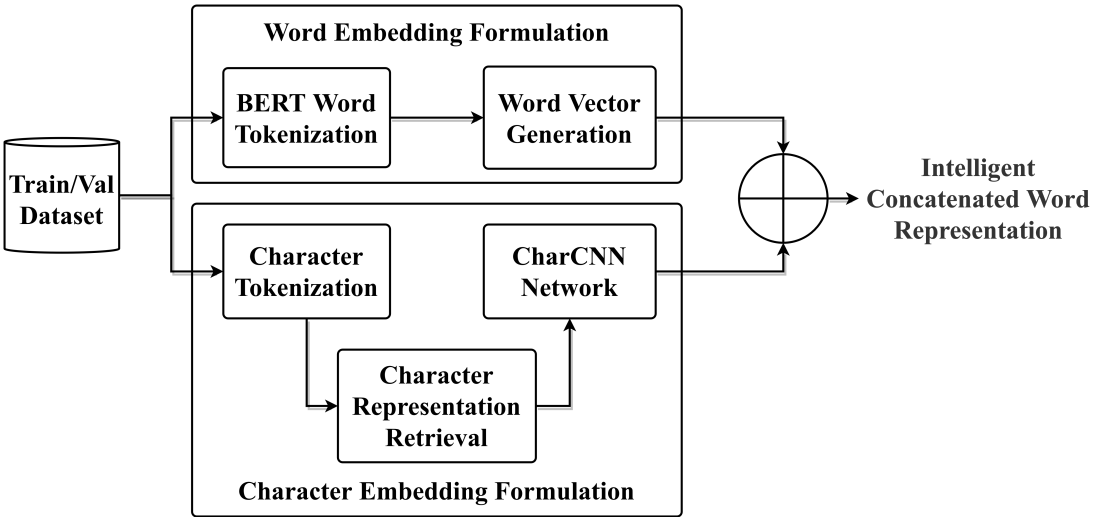


Figure 3.4: Module 1

The text form of the user's utterance is taken as the input to prepare the word embeddings. The input text is tokenized using BERT based word tokenization by which the original word is split into smaller sub-words and characters. Upon breaking the text into tokens, the sentences are converted from a list of strings to a list of vocabulary indices. After being converted into the prescribed format, the word vectors are generated via a summing mechanism.

To deal with Out-Of-Vocabulary words, character embeddings are prepared by constructing the vector from the composed characters as shown in the bottom part of Figure 3.4. Features are extracted from this vector using a CharCNN layer and is concatenated with the word vectors to formulate the desired word representation. The following algorithm 1 explains each involved step in a concise manner.

Algorithm 1 Preprocessing - Word Representation Formulation

Input: Text from user's utterance
Output: Concatenated Word Representation
for *text in utterance* **do**
    tokenizedWords ← BERT.Tokenize(text)
    vocabIndexList ← tokenToIndex(tokenizedWords)
    for *word in vocabIndexList* **do**
        $wordVector \leftarrow BertModel(word)$
    tokenizedCharacters ← characterTokenization(text)
    for *character in tokenizedCharacters* **do**
        characterVector.append(oneHot(character))
    characterVector ← CharCNN(characterVector)
    concatenatedWordRepresentation ← Concat(wordVector,characterVector)

Intelligent Concatenated Representation

```
[ ]  # Sample Embedding
     index = int(input("Enter index: "))
     print("Input Query: " + train_df.iloc[index, 0])
     print("BERT Embedding Shape: " , np.array(train_df.iloc[index, 4]).shape)
     print("CharCNN Embedding Shape: ", np.array(train_df.iloc[index, 5]).shape)
     print("Intelligent Concatenated Word Representation Shape: ", np.array(train_df.iloc[index, 7]).shape)

     Enter index: 5
     Input Query: list all flights from pittsburgh to philadelphia
     BERT Embedding Shape:  (24, 384)
     CharCNN Embedding Shape:  (24, 100)
     Intelligent Concatenated Word Representation Shape:  (24, 484)
```

Figure 3.5: Shapes of each representation

Figure 3.5 represents the shape of the individual embeddings as well as the concatenated intelligent word embedding.

```
[ ]  print("\nIntelligent Concatenated Word Representation: ", train_df.iloc[index, 7])
```

```
Intelligent Concatenated Word Representation:  [array([ 1.05455756e-01,  5.33394776e-02, -4.75901328e-02,  4.21749264e-01,
        1.32951245e-01, -3.53912920e-01,  8.06346416e-01, -1.74798191e-01,
       -1.03638761e-01,  1.05727715e-02, -1.14848912e-01, -5.29299736e-01,
       -7.33419657e-02,  2.71434158e-01, -1.78288613e-02,  1.79571316e-01,
        4.87265736e-01, -1.29274026e-01,  3.27462345e-01, -5.07399738e-01,
        5.71645856e-01,  2.68918753e-01, -3.12753230e-01,  2.59546131e-01,
        4.33816344e-01,  8.12115192e-01, -2.94468641e-01,  2.82674823e-02,
       -1.74646690e-01, -3.78950387e-01, -4.81065989e-01,  4.74012166e-01,
        8.12968552e-01,  3.49523336e-01,  1.09348200e-01, -1.97766021e-01,
       -2.03077700e-02,  1.47855029e-01,  4.54230636e-01, -2.96758264e-02,
        3.97327393e-02, -3.99981380e-01,  3.26626189e-02,  5.15525453e-02,
       -6.53692424e-01, -5.83321035e-01, -4.97856885e-01, -8.89840245e-04,
       -2.12940410e-01,  1.79659769e-01, -6.97538376e-01,  6.12996258e-02,
       -5.89153528e-01, -4.18132871e-01,  8.35041761e-01, -9.22385380e-02,
        2.09632143e-02, -2.25390002e-01, -2.12472916e-01, -7.02514648e-01,
```

Figure 3.6: Intelligent Concatenated Representation

In Figure 3.6, a sample intelligent representation is shown.

```
[ ]  print("\nCharCNN Embedding: ", train_df.iloc[index, 5])
```

```
CharCNN Embedding:  [array([0.9620004 , 0.        , 1.2165515 , 0.        , 0.70012015,
       0.        , 1.422039  , 0.0598212 , 0.7984161 , 1.5171223 ,
       0.        , 0.36067742, 0.        , 0.8321939 , 1.2603669 ,
       0.        , 0.        , 0.5462933 , 0.96808434, 1.3488799 ,
       0.        , 0.35696313, 0.542168  , 1.2221024 , 0.26828927,
       0.96800196, 0.44237804, 0.24675098, 1.20941   , 1.3402786 ,
       0.8846285 , 0.47121418, 0.68158484, 0.        , 0.        ,
       0.7066314 , 1.439575  , 0.13860813, 0.3291201 , 0.        ,
       0.        , 0.        , 1.0211809 , 0.8308318 , 0.        ,
       0.6624733 , 0.        , 0.        , 0.08371684, 0.09087047,
       0.        , 1.2155466 , 0.7473688 , 0.6624509 , 0.33974043,
       1.0285767 , 0.12979275, 0.18740195, 0.27342975, 0.9958683 ,
       0.59755933, 0.        , 0.11815178, 0.95345926, 0.        ,
       0.52276146, 0.        , 0.5248372 , 1.1741784 , 0.65238523,
       1.064513  , 1.0615201 , 0.        , 0.74504054, 0.        ,
```

Figure 3.7: CharCNN Embedding

An example CharCNN embedding is shown in Figure 3.7. In Figure 3.8, sample BERT embedding of dimensions mentioned in Figure 3.5 is shown.

CNN - Convolutional Neural Network

A Convolutional Neural Network (CNN) is a Deep Learning system that can take an input image, assign relevance (learnable weights and biases) to various aspects/objects in the image, and distinguish between them. When compared to other classification methods, the amount of pre-processing required by CNN is significantly less. While basic approaches require hand-engineering

```
[ ]  print("\nBERT Embedding: ", train_df.iloc[index, 4])
```

```
BERT Embedding:  [array([ 1.05455756e-01,  5.33394776e-02, -4.75901328e-02,  4.21749264e-01,
        1.32951245e-01, -3.53912920e-01,  8.06346416e-01, -1.74798191e-01,
       -1.03638761e-01,  1.05727715e-02, -1.14848912e-01, -5.29299736e-01,
       -7.33419657e-02,  2.71434158e-01, -1.78288613e-02,  1.79571316e-01,
        4.87265736e-01, -1.29274026e-01,  3.27462345e-01, -5.07399738e-01,
        5.71645856e-01,  2.68918753e-01, -3.12753230e-01,  2.59546131e-01,
        4.33816344e-01,  8.12115192e-01, -2.94468641e-01,  2.82674823e-02,
       -1.74646690e-01, -3.78950387e-01, -4.81065989e-01,  4.74012166e-01,
        8.12968552e-01,  3.49523336e-01,  1.09348200e-01, -1.97766021e-01,
       -2.03077700e-02,  1.47855029e-01,  4.54230636e-01, -2.96758264e-02,
        3.97327393e-02, -3.99981380e-01,  3.26626189e-02,  5.15525453e-02,
       -6.53692424e-01, -5.83321035e-01, -4.97856885e-01, -8.89840245e-04,
       -2.12940410e-01,  1.79659769e-01, -6.97538376e-01,  6.12996258e-02,
       -5.89153528e-01, -4.18132871e-01,  8.35041761e-01, -9.22385380e-02,
        2.09632143e-02, -2.25390002e-01, -2.12472916e-01, -7.02514648e-01,
```

Figure 3.8: BERT Embedding

of filters, CNN models can learn these characteristics with enough training. Through the application of suitable filters, CNNs are able to successfully capture the Spatial and Temporal relationships in a picture. Due to the reduced number of parameters involved and the reusability of weights, the architecture performs superior fitting to the picture dataset. In other words, the network may be trained to better understand the image's sophistication.

The Convolution Operation's goal is to extract high-level characteristics from the input image, such as edges. There is no need to limit ConvNets to just one Convolutional Layer. The first ConvLayer is traditionally responsible for capturing Low-Level information such as edges, colour, gradient direction, and so on. The architecture adjusts to the High-Level characteristics as well with the addition of layers.

CharCNN

Convolutional Neural Networks have been used in NLP recently, and it has been widely seen to yield good performance in these tasks. When concerning any NLP task it is important for the model to understand underlying features of the sentence, but can fall short when it comes to out-of-vocabulary. CharCNN comes into play by to solve this problem. As the name implies, it treats each sentence in the character level, decreasing the unknown words to

a great extent, making it a good choice to increase model performance. Its architecture is shown in Figure 3.9

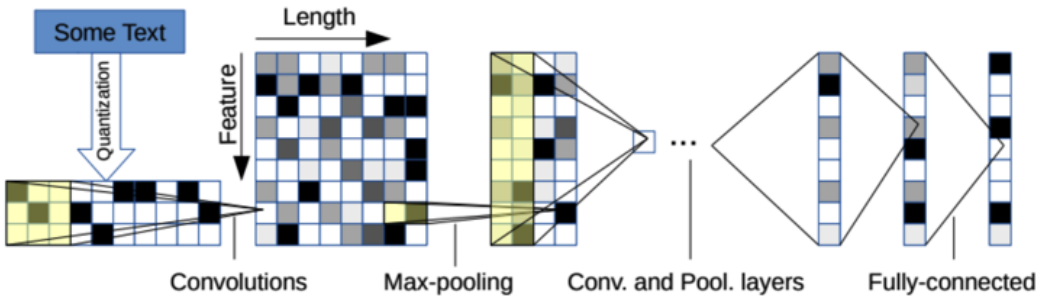


Figure 3.9: CharCNN

Transformers

A transformer is a deep learning model that uses the self-attention mechanism to weight importance of each element of the input data differently. It is largely used in natural language processing and computer vision applications. Transformers are used to process sequential input data like natural language, and can be used for tasks like translation and text summarization. Transformers, on the other hand, process the full input at once, unlike RNNs. Any place in the input sequence the context is given by the attention mechanism. When concerning natural language sentences, the transformer does not need to process each word individually. This allows for more parallelization resulting in faster training.

Transformer is an architecture consisting of two parts - Encoder Decoder for transforming one sequence to another, although it varies from existing sequence to sequence models in that it does not utilise Recurrent Networks (GRU, LSTM, etc.). Both the Encoder and the Decoder are made up of modules that can be layered on top of one another. Multi-Head Attention and Feed Forward layers make up the majority of the modules. Because we can't use strings directly,

the inputs and outputs (target sentences) are first embedded in an n-dimensional space. An important part of is the positional encoding of the different words. When it comes to recurrent networks, the have memory to remember how the sequences are fed to the model. But that is not the case here, so instead the positions are added to the embedded representation of each word.

BERT - Bidirectional Encoder Representations from Transformers

Bert uses transformers, an attention mechanism that learns contextual relationships between words (or sub-words) in a text. Transformer incorporates two different mechanisms in its basic form: an encoder that reads the text input and a decoder that generates a job prediction. Only the encoder technique is required because BERT's purpose is to generate a language model. The Transformer encoder reads the complete sequence of words at once, unlike directional models that read text input sequentially (left-to-right / right-to-left). As a result, it is classified as bidirectional, however it is more correct to describe it as non-directional. This property enables the model to deduce the context of a word from its surroundings (left and right of the word).

The BERT model used here is **paraphrase-MiniLM-L6-v2**. This particular model maps sentences and paragraphs to a 384 dimensional vector space and is used for tasks like clustering and semantic search.

3.5.2 Contextual Sentence Representation Formulation

Each word is represented as a concatenation of the following two vectors: The Word Embedding from the BERT model and the output layer from the single layer CharCNN Network over the Character Embeddings. For sequentially encoding information, obtained word representations are operated with multiple stacked Bi - LSTM layers having residual connections between the

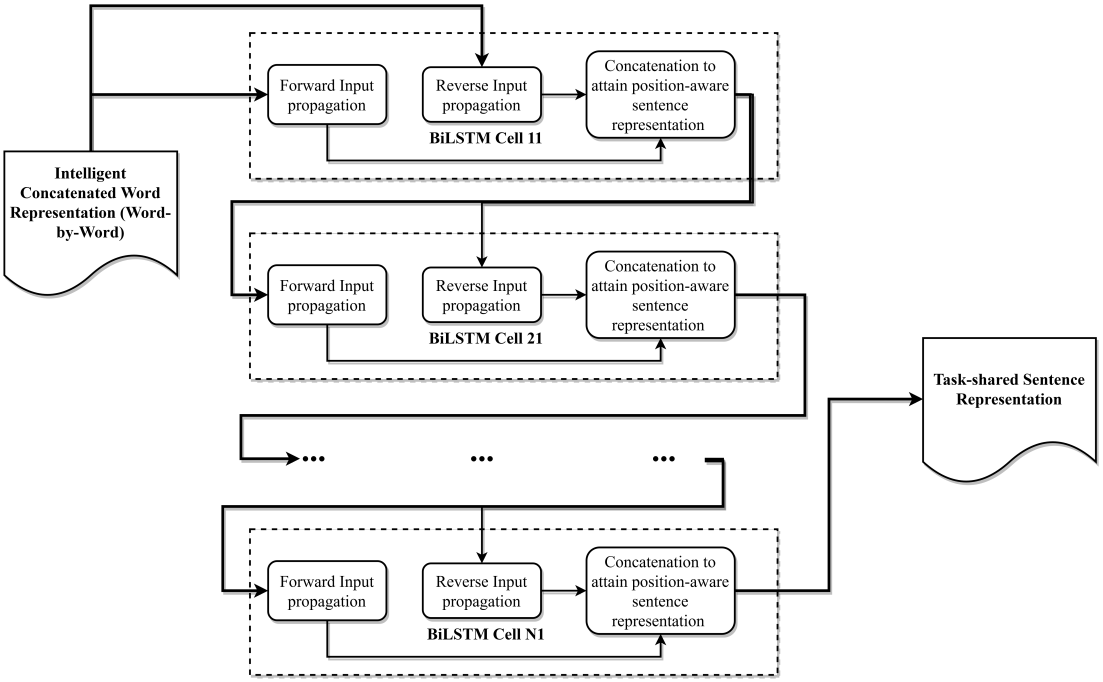


Figure 3.10: Module 2

consecutive layers. The semantically encoded sentence representation is then fed as the input to the subsequent modules for further processing. In Figure 3.11, a Stacked BiLSTM architecture with three layers is depcited where the concatenated hidden representation outputs are passed in a ;ayer-wise manner to the BiLSTM layer above.

The residual and non-residual forms of the LSTM architectures is experimented.

The residual LSTM learns the residual nodes with reference to the hidden state. It delivers efficient training and validation when compared with an ordinary LSTM model by providing a shortcut path during training. The residual LSTM in Figure 3.10 provides an additional spatial shortcut path from lower layers for efficient training of deep networks with multiple LSTM layers. Residual connections are the same thing as 'skip connections'. They are used to allow gradients to flow through a network directly, without passing through non-linear activation functions. Non-linear activation functions, nature of being nonlinear,

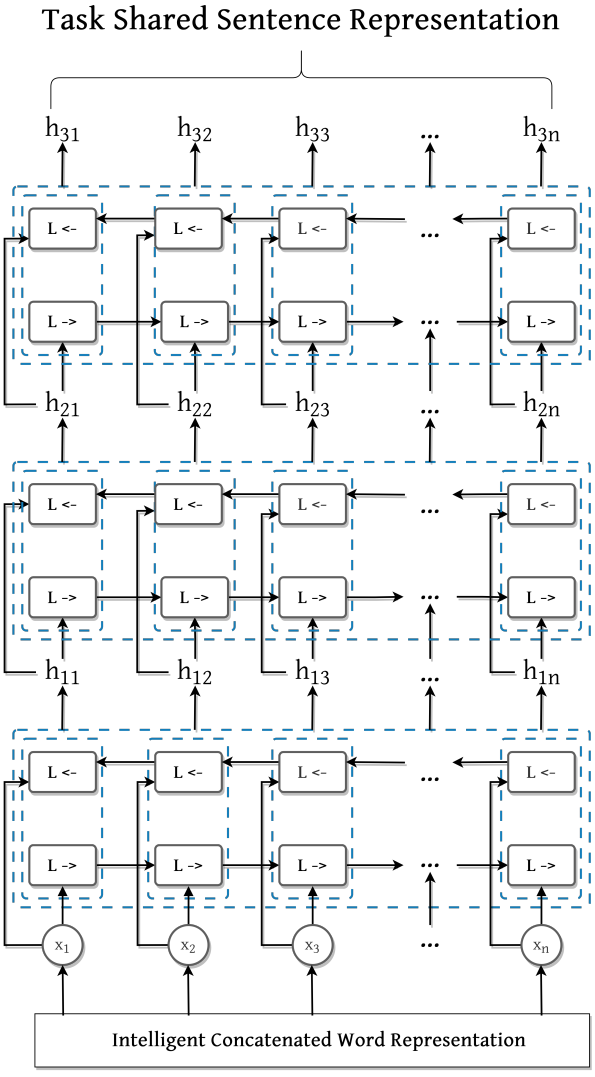


Figure 3.11: Stacked BiLSTM

cause the gradients to explode or vanish (depending on the weights). Networks with large numbers of layers can be trained easily without increasing the training error percentage. It helps in tackling the vanishing gradient problem using identity mapping. It helps to accelerate speed of training of the deep networks. Also it reduces the effect of Vanishing Gradient Problem and helps obtain a higher accuracy in network performance.

Algorithm 2 Contextual Sentence Representation Formulation

Input: Intelligent Concatenated Word Representation
Output: Task Shared Sentence Representation
for *word in sentence* **do**
    for *layer in range*(*num_stacked_layers*) **do**
        if *layer is* 0 **then**
            cell_fw ← ForwardPropagation(word)
            cell_bw ← ReversePropagation(word)
            final_cell_state ← Concat(cell_fw, cell_bw)
        else
            cell_fw ← ForwardPropagation(final_cell_state)
            cell_bw ← ReversePropagation(final_cell_state)
            final_cell_state ← Concat(cell_fw, cell_bw)
        wordVector ← *BertModel*(*word*)

Algorithm 2 explains the concatenation procedure followed by the BiLSTM to combine its forward and reverse propagations. This has been placed inside a loop in order to depict the stacked nature of the BiLSTM layers.

In Figure 3.12, sample contextual taskshared sentence representation is shown for a random input query from the dataset.

LSTM

The Long Short Term Memory Network (LSTM) is an enhanced RNN (sequential network) that permits information to be stored indefinitely.

```
[ ] # Input Query
    index = int(input("Enter index: "))
    print("Input Query: " + train_df.iloc[index, 0])
    print()
    # Task shared sentence representation
    print("Task Shared Sentence Representation:","\n",task_shared_model.predict(np.expand_dims(train_df.iloc[index,7],axis=0)))
    print()
    # Shape of Task shared sentence representation
    task_shared_model.predict(np.expand_dims(train_df.iloc[index,7],axis=0)).shape
```

```
Input Query: show afternoon flights from dallas to san francisco

Task Shared Sentence Representation:
 [[[-3.99863496e-02 -1.65925696e-02  1.30337607e-02 ...  2.22470537e-01
    6.76990673e-02  3.98390777e-02]
  [-7.74624050e-02 -3.02435644e-02  2.86813322e-02 ...  2.36059189e-01
    6.27216771e-02  5.09501994e-02]
  [-1.08953252e-01 -4.01648283e-02  4.33600657e-02 ...  2.46034950e-01
    5.62693663e-02  6.05579242e-02]
  ...
  [-2.00704545e-01  1.20975889e-01 -9.79848206e-02 ...  3.85983475e-02
    1.32175619e-02  1.12959193e-02]
  [-1.93903878e-01  1.22621335e-01 -9.11747664e-02 ...  2.07144096e-02
    5.42862806e-03  2.25385930e-03]
  [-1.81466863e-01  1.19346738e-01 -8.14116746e-02 ...  7.64229987e-03
   -1.79333874e-04 -2.21578707e-03]]]

(1, 24, 256)
```

Figure 3.12: Task shared representation using stacked BiLSTMs

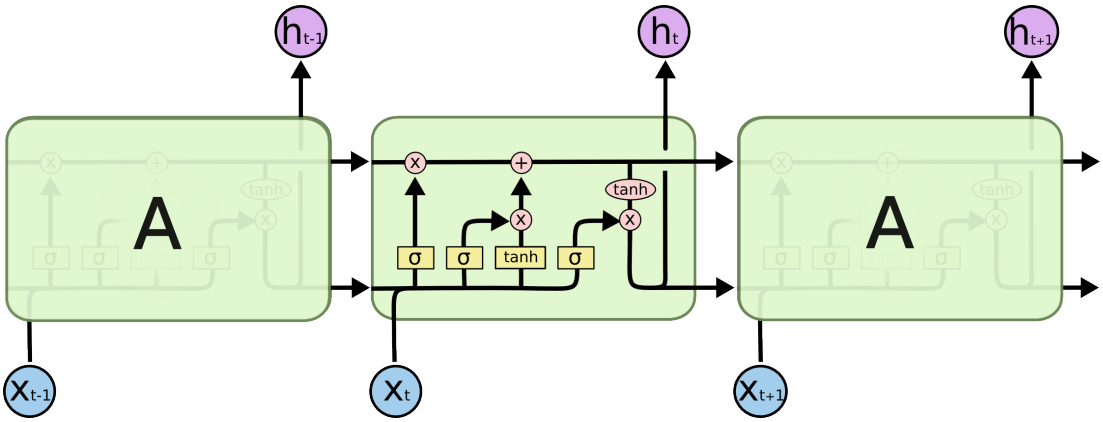


Figure 3.13: LSTM Architecture

It can deal with the vanishing gradient problem that RNN has due to its peculiar architecture shown in Figure 3.13. A recurrent neural network is also known as RNN is used for persistent memory.

$$\tilde{c}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \tag{3.1}$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{3.2}$$

$$h_t = o_t * tanh(c_t) \tag{3.3}$$

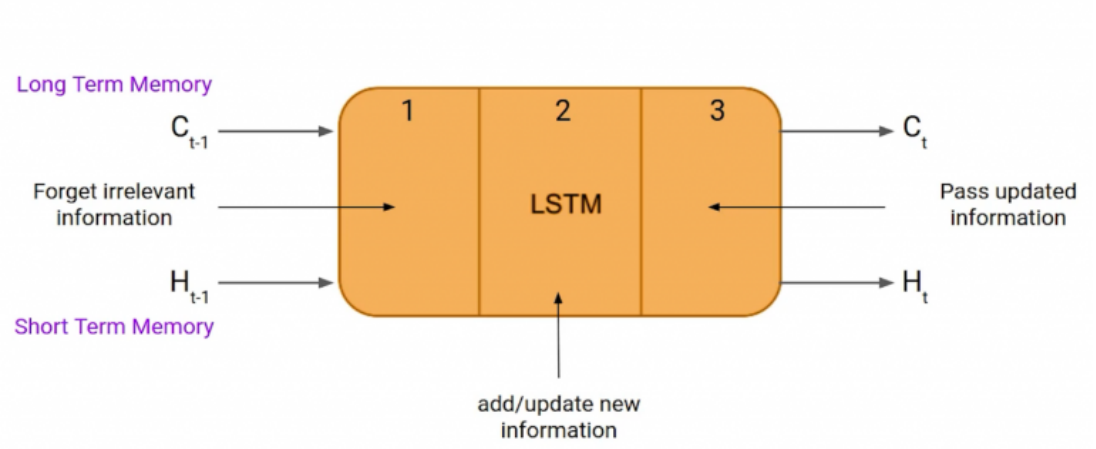


Figure 3.14: LSTM Cell

In Figure 3.14, first part determines whether the previous timestamp information should be remembered or is irrelevant and can be ignored. The cell attempts to learn new information from the input in the second part. Finally, the cell passes updated information from the current timestamp to the next timestamp in the third part.

The LSTM Cell consists of gates :

- Forget Gate : Decides if we should keep or forget the information

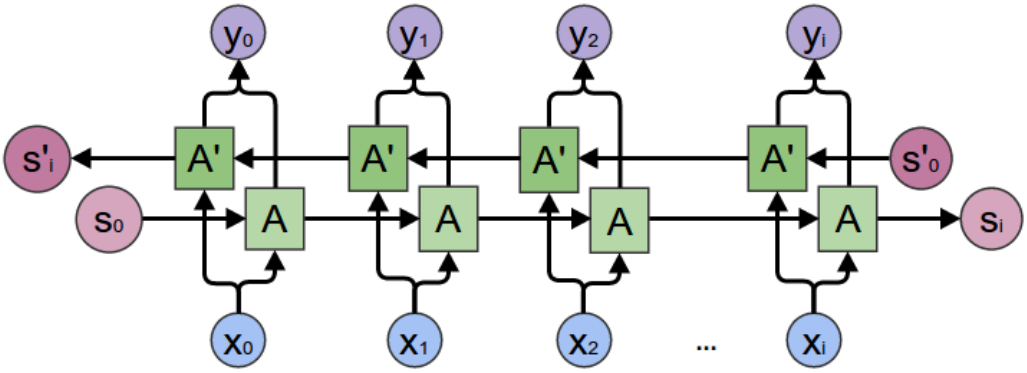


Figure 3.15: Bidirectional LSTM Approach

from the preceding timestamp.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.4}$$

- Input Gate : Used to quantify the importance of the new information carried by the input.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{3.5}$$

- Output Gate : Gives the output word.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.6}$$

BiLSTM

Bidirectional Long Short Term Memory (LSTM) is combining two independent LSTMs together. This structure allows the networks to have both backward and forward information about the sequence at every time step. Using bidirectional will run your inputs in two directions, one from past to future and the other from future to past. What distinguishes this from unidirectional is that in the LSTM that runs backward, information from the future is preserved, whereas using the two hidden states combined, you can preserve information

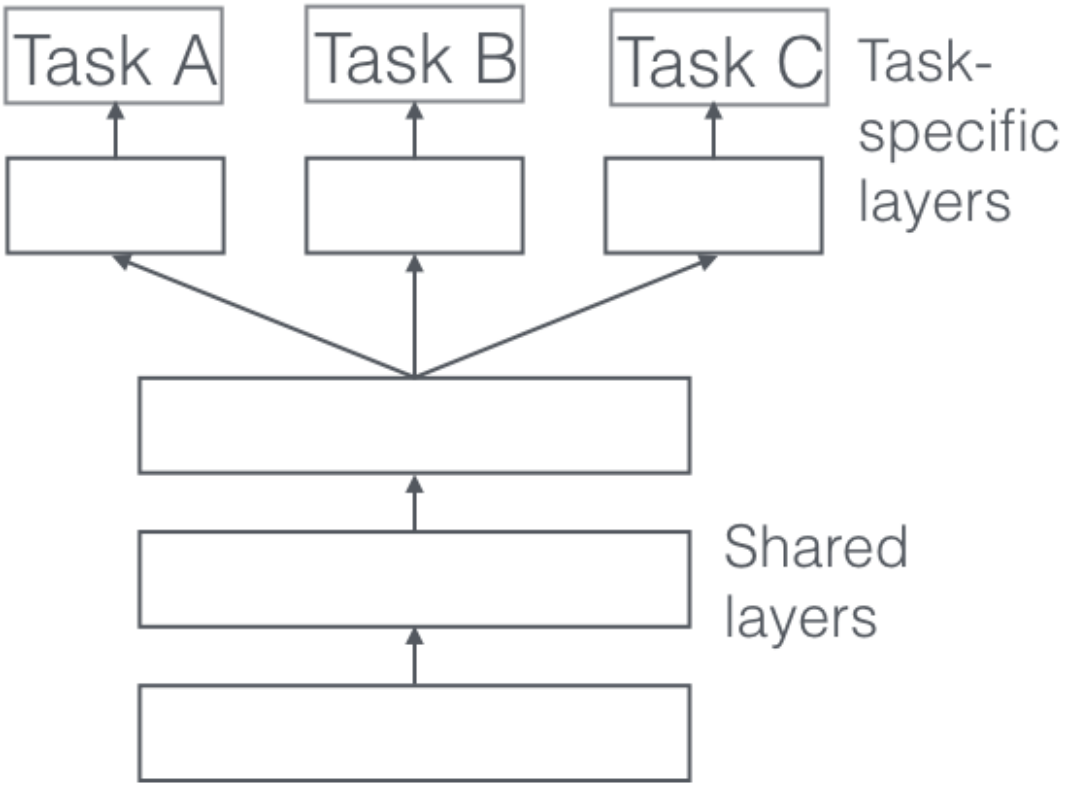


Figure 3.16: Multi Task Learning with Hard Parameter Sharing

from both past and future at any point in time. This approach is shown in Figure 3.15

This architecture offers numerous benefits in real-world issues, particularly in NLP. The major reason for this is that every component of an input sequence contains data from the past as well as the present. As a result, by integrating LSTM layers from both directions, BiLSTM can create a more relevant output.

Multi Task Learning

Multi-task learning (MTL) aims to improve the performance of multiple related learning tasks by leveraging useful information among them. The two dominant approaches for performing MTL with neural networks - hard and soft parameter sharing, in which we seek to learn shared or “similar” hidden representation(s)

for the different tasks. In order to impose these similarities between tasks, the model is simultaneously learned for all tasks and with some constraint or regularization on the relationship between related parameters.

In hard parameter sharing a common space representation is learnt for all tasks (i.e. completely share weights/parameters between tasks). This shared feature space is used to model the different tasks, usually with additional, task-specific layers (that are learned independently for each task), as depicted in the top part of Figure 3.16. Hard parameter sharing acts as regularization and reduces the risk of overfitting, as the model learns a representation that will generalize well for all tasks. Based on an assumption that all the tasks, or at least a subset of them, are related, jointly learning multiple tasks is empirically and theoretically found to lead to better performance than learning them independently.

Hard Parameter Sharing has been employed here.

3.5.3 Dialogue Act Classification

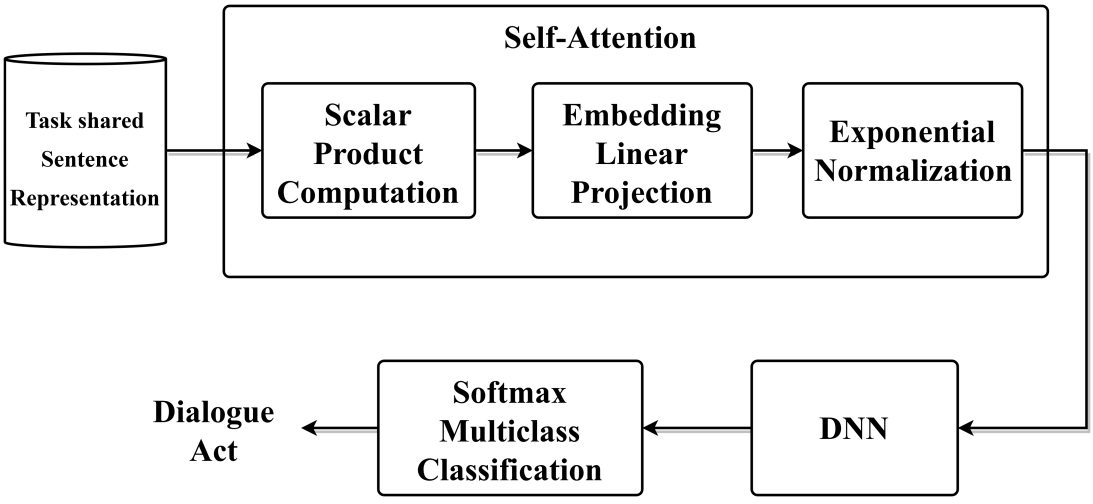


Figure 3.17: Module 3

Identifying the dialogue act of the sentence along with its intent helps in answering the user's query appropriately. To accomplish this, self-attention

mechanism is incorporated over the task shared sentence representation from the Bi-LSTM layers. On passing through a Dropout layer and dense layer, a non-linear combination of the extracted features is obtained. This is fed to a softmax layer to identify the dialogue act associated with the input utterance. This flow is diagramatically represented in Figure 3.17.

The following attention equation is employed to calculate alignment scores between the key, query and value matrices.

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.7}$$

In Algorithm 3, Dialogue Act class of the input query is determined through self-attention followed by affine transforming to a final output layer post passing through a softmax activation layer.

Algorithm 3 Dialogue Act Classification

Input: Task Shared Sentence Representation
Output: Dialogue act of the utterance

function CLASSIFYDIALOGUEACTS(($sentenceRepresentation$))
    sentenceRepresentation ← selfAttention(sentenceRepresentation)
    extractedFeatures ← DNN(sentenceRepresentation)
    extractedFeatures ← affineTransformation(extractedFeatures)
    predictedDialogueAct ← Softmax(extractedFeatures)
    return *predictedDialogueAct*
end function

The following figure 3.18 renders some dialogue act predictions.

Self Attention

The attention mechanism allows output to focus attention on input

```
[ ] import random
    #Predict the dialogue acts
    for i in random.sample(range(1, 498), 7):
        #print(i)
        prediction = (model.predict(np.expand_dims(X_test[i],axis=0)) > 0.5).astype("int32")
        print("Input Query  "+test_df["sentences"][i])
        print("Expected DAC:  "+test_df["dac"][i])
        print("Predicted DAC: "+dac_label_encoder.inverse_transform([np.argmax(prediction, axis=None, out=None)]))
        print('*'*75)
```

```
Input Query  list flights from newark to houston
Expected DAC:  command
['Predicted DAC: command']
***************************************************************************
Input Query  please list flights from milwaukee to philadelphia
Expected DAC:  command
['Predicted DAC: command']
***************************************************************************
Input Query  what is the lowest fare for a flight from washington dc to boston
Expected DAC:  question
['Predicted DAC: question']
***************************************************************************
```

Figure 3.18: Predicted Dialogue Acts of input query

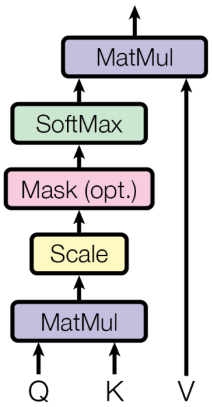


Figure 3.19: Self Attention Mechanism

while producing output while the self-attention model allows inputs to interact with each other.

The self attention works in such a way we first multiply each of the encoder input vectors with the three weight matrices - W(Q), W(K), W(V). This multiplication will give us three vectors : key vector, query vector, value vector. Next is actually calculating the self-attention by multiplying the query vector of current input with other inputs key vector. After which the score is divided by the square root of the key vector dimensions (d_k) and applying softmax on all attention scores. Finally summing up the weighted value vectors from the previous step will give the self-attention output for the given word. By this method we can caculate the self-attention with respect to each individual word.

As shown in Figure 3.19, self attention helps improve performance minimizing total computation while maximizing the amount of parallel computations. Also supports shorter path length between input and output positions, which makes it easier to learn long-range dependencies.

3.5.4 Intent Detection

The semantically encoded sentence representation from the stacked Bi-LSTMs is fed as input to this module, upon which self-attention mechanism is applied. The weighted mean of the hidden states of all timesteps is calculated and is passed to a Multi-Layer Perceptron network. This dense network helps in arriving at a non-linear combination of the extracted features from the previous step. Affine transformation is performed and the multi-class classification is accomplished using the softmax layer upon normalizing the probability scores of each class. Thus, the intent of the user's utterance is detected and is used in formatting a cogent response to the input query. This is diagramatically represented in Figure 3.20

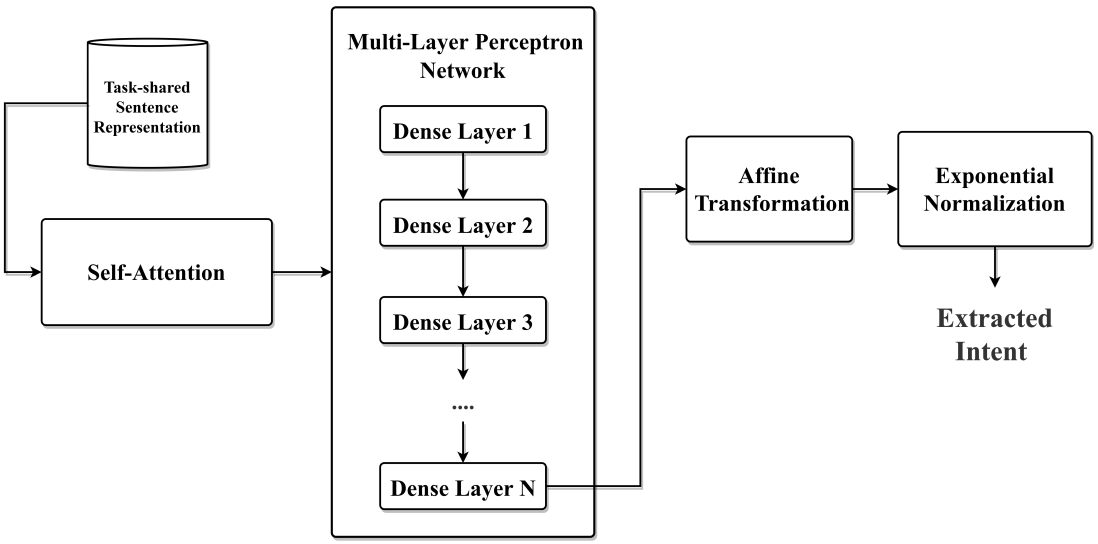


Figure 3.20: Module 4

Algorithm 4 Intent Detection

Input: Task Shared Sentence Representation
Output: Intent of the utterance

function DETECTINTENTS(($sentenceRepresentation$))
    sentenceRepresentation ← selfAttention(sentenceRepresentation)
    extractedFeatures ← DNN(sentenceRepresentation)
    extractedFeatures ← affineTransformation(extractedFeatures)
    predictedIntent ← Softmax(extractedFeatures)
    return *predictedIntent*
end function

In Algorithm 4, the deep neural network on the self-attended layer helps to extract the required features and intent is predicted after passing through a Softmax activation layer to normalise the units of the output layer.

In Figure 3.21, sample intents predicted by proposed CIDIS model is shown.

3.5.5 Slot Filling

```
[ ] import random
    #Predict the intents
    for i in random.sample(range(1, 893), 7):
        #print(i)
        prediction = (model.predict(np.expand_dims(X_test[i],axis=0)) > 0.5).astype("int32")
        print("Input Query  "+test_df["sentences"][i])
        print("Expected Intent:  "+test_df["intents"][i])
        print("Predicted Intent: "+intent_label_encoder.inverse_transform([np.argmax(prediction, axis=None, out=None)]))
        print('*'*75)
```

```
Input Query  what does hp stand for
Expected Intent:  atis_abbreviation
['Predicted Intent: atis_abbreviation']
***************************************************************************
Input Query  how far is toronto international from downtown
Expected Intent:  atis_distance
['Predicted Intent: atis_distance']
***************************************************************************
Input Query  what airline is dl
Expected Intent:  atis_airline
['Predicted Intent: atis_airline']
***************************************************************************
```

Figure 3.21: Predicted Intent of input query

Algorithm 5 Slot Filling

Input: Task Shared Sentence Representation
Output: Extracted slots from the utterance

function FILLSLOTS(($sentenceRepresentation$))
    extractedFeatures ← DNN(sentenceRepresentation)
    predictedBIOtags ← Softmax(extractedFeatures)
    for *tag* in *predictedBIOtags* **do**
    posTags ← pos_tags(tag)
    slotTree ← BIOtoTree(predictedBIOtags,posTags)
    extractedSlots ← parse(slotTree)
    return *extractedSlots*
end function

Algorithm 5 primarily deals with predicting the BIO-tagged slots and converting them to their cleaned format. This is performed by appending the POS tags to the respective tag followed by generation of a tree and parsing the same.

Slot filling helps to extract the semantic constituents from the input text, and to fill in the values for a predefined set of slots in the semantic frame. The input slots are in the BIO [Beginning – Inside - Outside] tagging format. The BIO format is a common tagging format for tagging tokens in a chunking

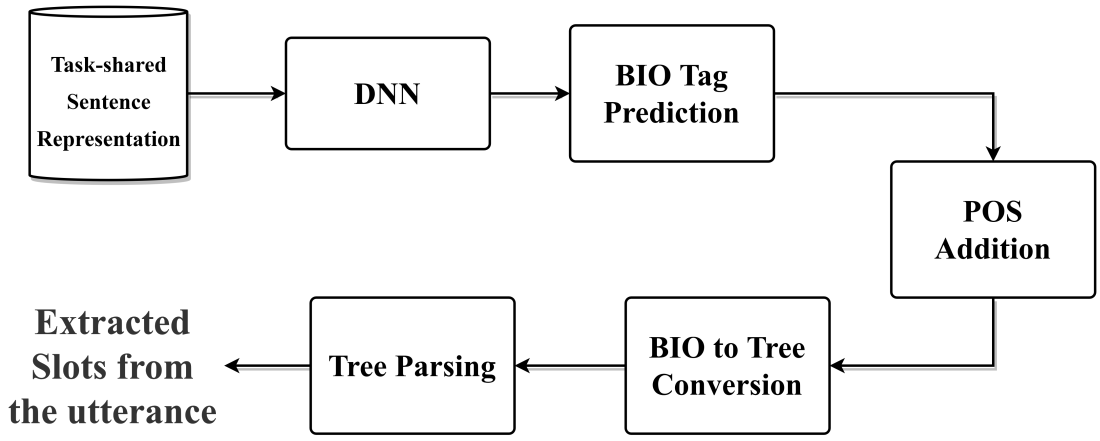


Figure 3.22: Module 5

task in computational linguistics like named-entity recognition. The B- prefix before a tag indicates that the tag is the beginning of a chunk, and an I- prefix before a tag indicates that the tag is inside a chunk. The B- tag is used only when a tag is followed by a tag of the same type without O tokens between them. An O tag indicates that a token belongs to no entity / chunk.

```
[ ] label_list = list(y_tokenizer.word_index.keys())
    index_list = list(y_tokenizer.word_index.values())

    import random
    #Predict the slots
    for i in random.sample(range(1, 893), 7):
        #print(i)
        prediction = bilstm_model.predict(np.expand_dims(X_test[i],axis=0))
        slots = [label_list[index_list.index(j)] for j in [np.argmax(x) for x in prediction[0][:]] if j in index_list]
        print("Input Query  "+test_df["sentences"][i])
        print("Expected Slots:  "+test_df["slots"][i])
        print("Predicted Slots: "+slots)
        print('*'*75)
```

```
272
cleveland to nashville flight friday morning
['B-fromloc.city_name', 'O', 'B-toloc.city_name', 'O', 'B-depart_date.day_name', 'B-depart_time.period_of_day']
***************************************************************************
106
what is the latest flight from baltimore to oakland that serves dinner
['O', 'O', 'O', 'B-flight_mod', 'O', 'O', 'B-fromloc.city_name', 'O', 'B-toloc.city_name', 'O', 'O', 'B-meal_description']
***************************************************************************
315
and now show me ground transportation that i could get in boston late night
['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'B-toloc.city_name', 'B-period_of_day', 'B-period_of_day']
***************************************************************************
```

Figure 3.23: Output Slots of input query

The hidden unit for each time step coming from the output of the shared sentence representation is transformed by a dense layer and then by a Softmax layer. The dense layer shown in Figure 3.22 helps to combine the hidden features for that time step nonlinearly. The Softmax layer first projects the output from the dense layer to the number of possible slot classes and

then transforms the scores for each class into a probability distribution. Upon prediction, the original text is extracted from the BIO Tagged slot outputs. First tag each token with supplementary information, such as its part of speech. Then the BIO tags are converted to a tree which is parsed to get the original text.

In Figure 3.23, sample slots filled by the model is shown.

3.5.6 Query Response Retrieval

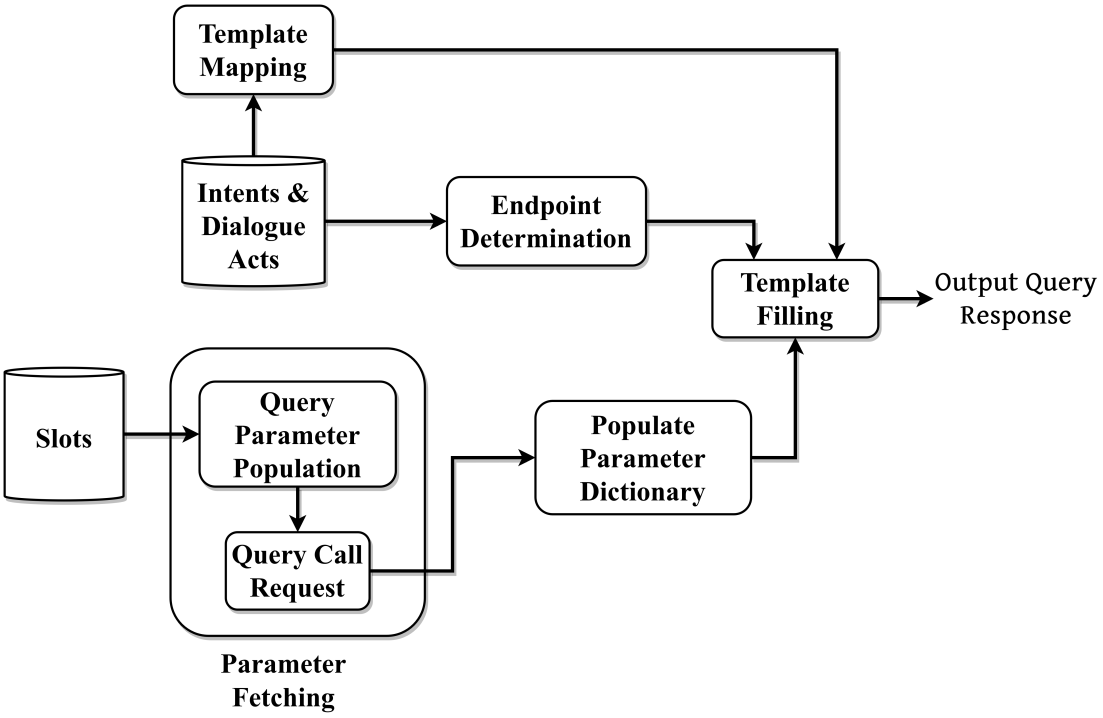


Figure 3.24: Module 6

To provide the user with the requested details, the identified intents, slots and dialogue acts are brought together in this final module. The appropriate real-time information is fetched by making http requests to the corresponding endpoints of an authenticated source. The slot values determine the endpoints to which the requests are to be sent and upon receiving the response, the list of parameters to be sent to the terminal end point is formulated. This terminal endpoint which is necessary to get the requested details is determined by the

detected intent and the corresponding dialogue act. On obtaining the required details, the predetermined template is filled and is returned back to the user as the output of the system. This is depicted in Figure 3.24

Algorithm 6 Query Response Retrieval

Input: Identified Intents, Dialogue Acts and Slots from the user's utterance
Output: Response to the user's query

function FETCHRESPONSE(*intent*, *dialogueAct*, *slots*)
  for *slot in slots* **do**
    supplementaryEndpoints.append(endpoints[slot])
  for *endpoint in supplementaryEndpoints* **do**
    parameters ← getParameters(endpoint)
  respTemplate ← determineTemplate(intent, dialogueAct)
  terminalEndpoint ← determineEndpoint(intent, dialogueAct, parameters)
  queryResponse ← fillTemplate(respTemplate,terminalEndpoint,parameters)
  return *queryResponse*

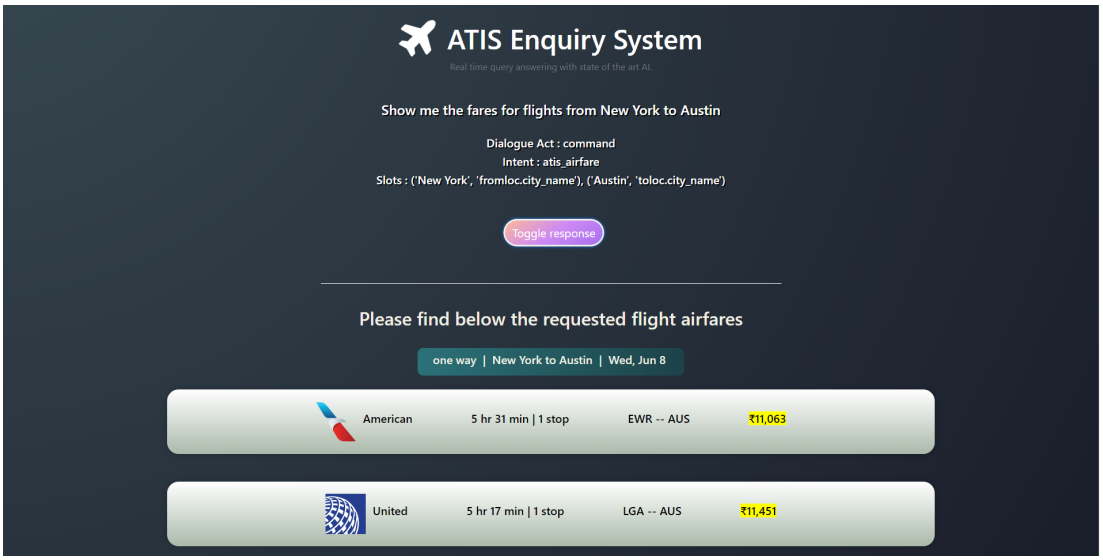


Figure 3.25: Query Response for air fare

As seen in Algorithm 6, using the identified dialogue acts,intents and slots, we first use the slots to obtain the parameters required for the http request. Next the intent and the dialogue act is used to determine the final end point and

required templates. Once the http request is made, the page content is scraped and the necessary data tags for the response are fetched. Finally the required data is filled into the template and sent to the user as the query response.

Query Response

In the Figure 3.25, a sample query is injected to the system to analyse its behaviour. The dialogue act, intent and slots are correctly predicted using the multi-task architecture and the requested flight details are returned in real time from a credible source.

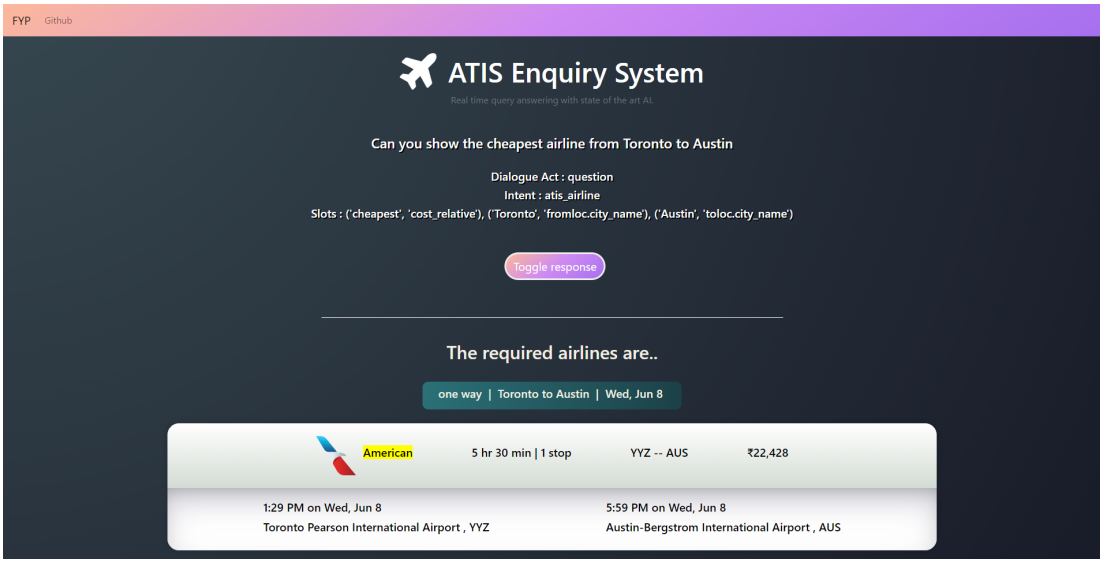


Figure 3.26: Query Response for cheapest flight

In the situation of Figure 3.26, not all flight details are returned. Only the details of the cheapest one is fetched and returned.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 DATASET

The proposed concurrent intelligent deep model is evaluated on three benchmark datasets - ATIS, FRAMES and TRAINS. The utterance, dialogue act, intent and slot distribution for the datasets are given in Table 4.1.

Table 4.1: Dataset Details

Dataset	#DA	#Intent	#Slot	#Train	#Val	#Test
ATIS	3	17	127	4978	498	893
TRAINS	5	12	32	4819	536	1336
FRAMES	10	24	136	19155	2129	5321

4.1.1 ATIS

The benchmark ATIS dataset comprises utterances of people making flight reservations and those asking for flight information on automated airline travel inquiry systems. There are 4978 utterances in the training set and 498 in the validation. The test set comprises 893 utterances. There are 17 distinct intent classes in the corpus. There are three dialogue acts and 127 distinct slots in the dataset. The train set consists of approximately 75% of the dataset, while the test and validation sets have 15% and 10% respectively.

4.1.2 FRAMES

The benchmark FRAMES dataset comprises multi-domain dialogues dealing with hotel bookings. There are 19155 utterances in the training set of the

corpus and 2129 in the validation. The test set comprises 5321 utterances. There are 24 distinct intents, ten dialogue acts and 136 distinct slots in the dataset. The train set consists of approximately 72% of the dataset, while the test and validation sets have 8% and 20% respectively.

4.1.3 TRAINS

The benchmark TRAINS dataset comprises utterances of problem solving dialogues between two people. There are 4819 utterances in the training set of the corpus and 536 in the validation. The test set has 1336 utterances. There are 12 distinct intent classes in the corpus. There are five dialogue acts in the corpus and 32 distinct slots in the dataset. The train set has 73% of the dataset, while the test and validation sets have 8% and 19% respectively.

Reason for using Multiple Datasets

Examining the performance of the model under different datasets ensures that the model is able to generalize across different domains. Further upon achieving a good performance across the various datasets, the model can understand the nuances involved in the input text provided. Upon recording state-of-the-art results in more than one dataset, it can be corroborated that the trained model is not overfit. The various hyperparameter tuning carried out to determine the combination which fetches the ideal values, is extended across the three chosen datasets. Hence an unanimous choice regarding the best performing hyperparameter can be achieved.

4.2 SAMPLE TEST CASES

The various possible test cases for evaluating the performance of the proposed system are tabulated in Table. 4.2.

Table 4.2: Sample Testcases

INPUT	OUTPUTS		
Input Query	**Dialogue Act**	**Intent**	**Slot**
show me all flights from pittsburgh to san francisco	Command	atis_flight	1) pittsburgh: B-fromloc.city_name 2) san francisco: B-toloc.city_name I-toloc.city_name
hello i'd like a flight from atlanta to boston	Statement	atis_flight	1) atlanta: B-fromloc.city_name 2) boston: B-toloc.city_name
what is the cost for a one way trip from pittsburgh to atlanta	Question	atis_airfare	1) round trip: B-round_trip 2) pittsburgh: B-fromloc.city_name 3) atlanta: B-toloc.city_name
cheapest fare from san francisco to dallas	Statement	atis_airfare	1) cheapest: B-cost_relative 2) san francisco: B-fromloc.city_name I-fromloc.city_name 3) dallas: B-toloc.city_name
what airlines fly between boston and atlanta	Question	atis_airline	1) boston: B-fromloc.city_name 2) atlanta: B-toloc.city_name
airline and flight number from columbus to minneapolis	Statement	atis_airline	1) columbus: B-fromloc.city_name 2) minneapolis: B-toloc.city_name
show me airports in dallas	Command	atis_airport	1) dallas: B-city_name

4.3 PERFORMANCE EVALUATION

For the evaluation of results, four metrics have been used, which are based on the number of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) in the predictions of the binary classifiers:

- Accuracy, which is the percentage of correct predictions.

$$Accuracy = \frac{No.of correct predictions}{Total Number of Predictions} \tag{4.1}$$

- Recall, which captures the ability of the classifier to find all the positive samples.

$$Recall = \frac{True Positive}{True Positive + False Negative} \tag{4.2}$$

- Precision, which is the ability of the classifier not to label a negative sample positive.

$$Precision = \frac{True Positive}{True Positive + False Positive} \tag{4.3}$$

- F1 score is the harmonic mean of precision and recall, computes values in the range [0,1].

$$F1 Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.4}$$

4.3.1 Performance on ATIS dataset

For implementation purposes, keras package has been used. The Stacked Bi-LSTMs are experimented under two different conditions. One with residual connections between subsequent layers and another devoid of these connections. The number of units in the LSTM was varied between 100 and 128. The corresponding metrics recorded for the ATIS dataset with residual networks is tabulated in Table. 4.3.

The word representation uses a 484 dimensional embedding which is obtained after the concatenation of 384 dimensions from pretrained BERT

Table 4.3: Experimentation on Residual Networks - ATIS Dataset

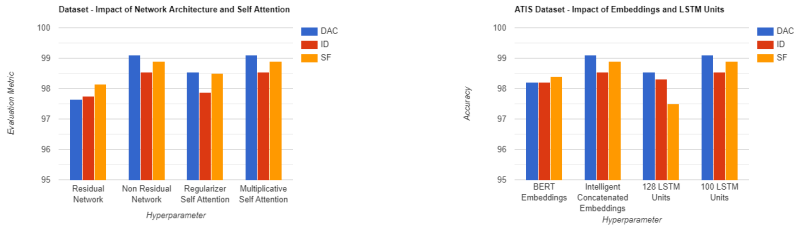
Self-Attention	Embedding	#Units	ATIS		
		(LSTM)	DA (Accuracy)	Intent (Accuracy)	Slot (F1 Score)
Multiplicative	BERT + Char CNN	100	0.9765	0.9776	0.9814
Multiplicative	BERT + Char CNN	128	0.9854	0.9832	0.9749
Multiplicative	BERT	100	0.9798	0.9709	0.9794
Multiplicative	BERT	128	0.9742	0.9832	0.9796
Regularizer	BERT + Char CNN	100	0.9854	0.9787	0.9850
Regularizer	BERT + Char CNN	128	0.9765	0.9810	0.9772
Regularizer	BERT	100	0.9742	0.9810	0.9818
Regularizer	BERT	128	0.9798	0.9832	0.9861

(paraphrase-MiniLM-L6-v2) embeddings and 100 dimensions from CharCNN. The text inputs before being fed to the LSTMs are padded or truncated to a maximum sequence length which has been determined based on the median lengths of the text in the dataset. The batch size is fixed as 128 and the training has been carried out for 50 epochs (Early Stopping included). The metrics obtained for the non residual networks with the intelligent embedding strategy is provided in Table. 4.4

Table 4.4: Experimentation on Non-Residual Networks - ATIS

Self-Attention	Embedding	#Units	ATIS		
		(LSTM)	DA (Accuracy)	Intent (Accuracy)	Slot (F1 Score)
Multiplicative	BERT + Char CNN	100	***0.991***	***0.9854***	***0.9889***
Multiplicative	BERT + Char CNN	128	0.9843	0.9698	0.9806
Multiplicative	BERT	100	0.9765	0.9821	0.9841
Multiplicative	BERT	128	0.9597	0.9854	0.9889
Regularizer	BERT + Char CNN	100	0.9745	0.9776	0.9832
Regularizer	BERT + Char CNN	128	0.9698	0.9787	0.9864
Regularizer	BERT	100	0.9821	0.9821	0.9839
Regularizer	BERT	128	0.9664	0.9698	0.9821

The impact of each hyperparameter has been visualized in Fig. 4.1 from which it can be observed that for the task of dialogue act classification, non residual networks provide a better performance than the residual connections. Under intent detection, multiplicative self attention aid in achieving a superior performance than regularizer. A similar trend is seen for the sequence labelling task of slot filling where non-residual and multiplicative self attention fetches the best performance.



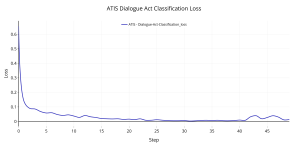
((a)) ((b))

Figure 4.1: Impact of hyperparameters on ATIS dataset

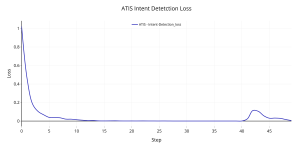
While the model was trained the losses associated had been plotted using Wandb package. The overall loss in the proposed multi task architecture is the aggregation of the losses associated with the individual tasks, which is shown in Fig. 4.2

Figure 4.2: Overall Loss for ATIS dataset

The loss recorded for each of the task under the ATIS dataset is shown in Fig. 4.3. It can be observed that loss values reduced from 1 with constant training, however, it appears to raise after the 40^{th} epoch. Nevertheless, the early stopping regularization mechanism ensures that further degradation of the model is not recorded. In a multi task architecture, it the sum of the losses associated with each of the individual tasks that constitues the overall loss which is observed in Fig. 4.2 .



((a)) DAC Loss for ATIS dataset



((b)) ID Loss for ATIS dataset

((c)) SF Loss for ATIS dataset

Figure 4.3: Loss for ATIS dataset

4.3.2 Performance on TRAINS dataset

The parameters of the model are updated using categorical cross entropy. Adam optimizer has been used and the metrics which are recorded for each task includes accuracy, precision, recall and loss. The results of the TRAINS datasets have been recorded in Table. 4.5 for the non-residual network.

Table 4.5: Experimentation on Non-Residual Networks - TRAINS

Self-Attention	Embedding	#Units	TRAINS		
		(LSTM)	DA (Accuracy)	Intent (Accuracy)	Slot (F1 Score)
Multiplicative	BERT + Char CNN	100	***0.9242***	***0.9091***	***0.9901***
Multiplicative	BERT + Char CNN	128	0.9143	0.8124	0.9798
Multiplicative	BERT	100	0.9368	0.8758	0.9734
Multiplicative	BERT	128	0.9090	0.9001	0.9898
Regularizer	BERT + Char CNN	100	0.9113	0.8897	0.9890
Regularizer	BERT + Char CNN	128	0.9240	0.8768	0.9900
Regularizer	BERT	100	0.8998	0.9102	0.9854
Regularizer	BERT	128	0.9200	0.8912	0.9904

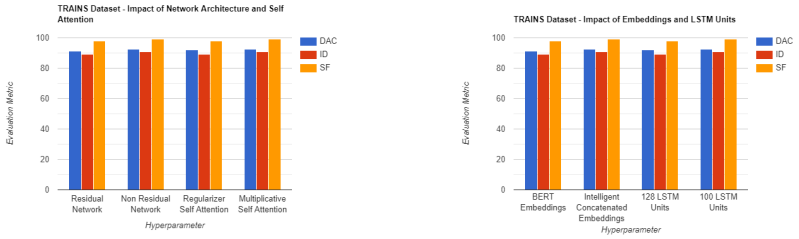
The Functional interface of keras had been utilized to implement the multi task architecture. In Table. 4.6 the metrics obtained for residual connections associated with TRAINS dataset is recorded. It can be inferred that multiplicative self attention with the intelligent embeddings provide the ideal performance. The loss plot obtained for the proposed model when executed on the TRAINS dataset is shown in Fig. 4.5. It can be observed that the loss reduces smoothly with training and the model is able to produce germane predictions.

Subsequently in Table. 4.6, the performance of the concurrent deep model has been observed on residual networks. Various hyperparameters such

Table 4.6: Experimentation on Residual Networks - TRAINS

Self-Attention	Embedding	#Units	TRAINS		
		(LSTM)	DA (Accuracy)	Intent (Accuracy)	Slot (F1 Score)
Multiplicative	BERT + Char CNN	100	0.9142	0.8945	0.9900
Multiplicative	BERT + Char CNN	128	0.9012	0.8823	0.9864
Multiplicative	BERT	100	0.9233	0.8901	0.9812
Multiplicative	BERT	128	0.8961	0.8914	0.9901
Regularizer	BERT + Char CNN	100	0.9003	0.9110	0.9817
Regularizer	BERT + Char CNN	128	0.9156	0.8965	0.9845
Regularizer	BERT	100	0.9210	0.8999	0.9912
Regularizer	BERT	128	0.9139	0.9023	0.9883

as the embeddings used, no of lstm units and the type of self attention and their influence has been recorded.



((a)) ((b))

Figure 4.4: Impact of hyperparameters on TRAINS dataset

The results from these experimentations are represented visually in the form of bar charts for the TRAINS dataset as shown in Figure. 4.4. It can be observed that slot filling performs superior compared to the other two tasks. This can be attributed to the relatively lesser count of slots.

Furthermore, while the model was training, the loss values were recorded using Wandb python package. The overall loss of the model for this dataset is as shown in Fig. 4.5. From this graph, it can be seen that the loss reduces as training proceeded. Early stopping mechanism with a patience of 5, was used to ensure that ideal performance is saved.

In the multi task paradigm adopted, the overall loss is the weighted

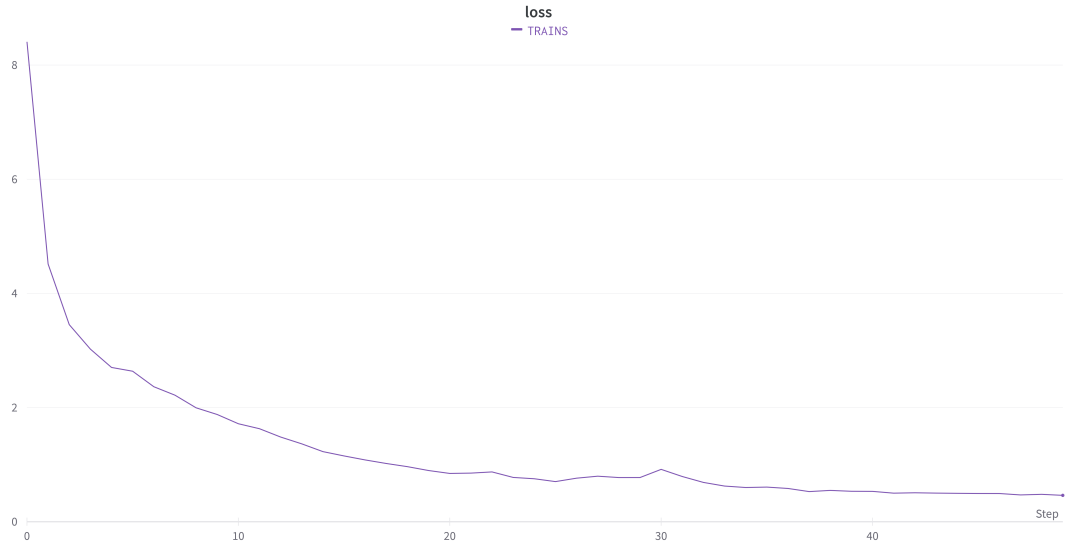
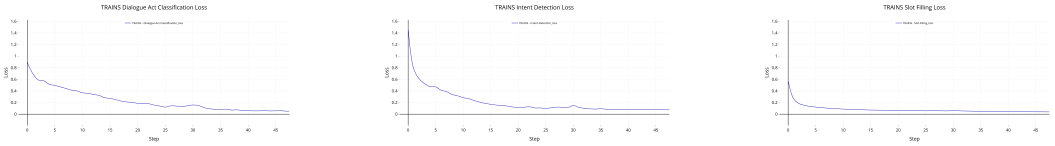


Figure 4.5: Overall Loss for TRAINS dataset

sum of the loss for each of the individual tasks. The resulting loss values obtained for the three tasks - Dialogue Act Classification, Intent Detection and Slot Filling for the TRAINS dataset is as shown in Fig. 4.6



((a)) DAC Loss for TRAINS dataset

((b)) ID Loss for TRAINS dataset

((c)) SF Loss for ATIS dataset

Figure 4.6: Loss for TRAINS dataset

From Fig. 4.6, it is observed that the performance of the model ameliorates with training and optimal results are achieved for each of the tasks. Upon the experimentations carried out the set of hyperparameters which provide the best performance for this dataset are recorded in Table. 4.9. The LSTM layer with 100 units outperforms the one with 128 units. The Self Attention width has been fixed at 15 for the task specific layers of Intent Detection and Dialogue Act Classification.

4.3.3 Performance on FRAMES dataset

The performance of the proposed model is evaluated on the third dataset - FRAMES. Upon the experimentations carried out, the results of the residual architecture are as shown in Table. 4.7. The non residual architecture has also been implemented and the results are as shown in Table. 4.8.

Table 4.7: Experimentation on Residual Networks - FRAMES Dataset

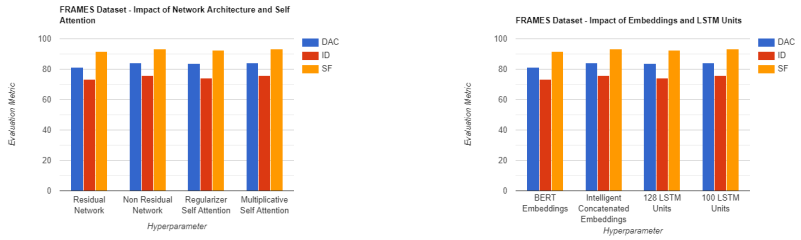
Self-Attention	**Embedding**	**#Units**	**FRAMES**		
		(LSTM)	DA (Accuracy)	Intent (Accuracy)	Slot (F1 Score)
Multiplicative	BERT + Char CNN	100	0.8335	0.7589	0.9417
Multiplicative	BERT + Char CNN	128	0.8207	0.7402	0.9476
Multiplicative	BERT	100	0.8312	0.7479	0.9329
Multiplicative	BERT	128	0.8299	0.7309	0.9456
Regularizer	BERT + Char CNN	100	0.8278	0.7500	0.9461
Regularizer	BERT + Char CNN	128	0.8337	0.7320	0.9377
Regularizer	BERT	100	0.8401	0.7490	0.9439
Regularizer	BERT	128	0.8347	0.7467	0.9304

It can be observed that the experimental results obtained by using BERT embeddings in isolation and when concatenated with CharCNN, indicate that a better performance is achieved on the latter. The residual characteristic

Table 4.8: Experimentation on Non-Residual Networks - FRAMES

Self-Attention	**Embedding**	**#Units**	**FRAMES**		
		(LSTM)	DA (Accuracy)	Intent (Accuracy)	Slot (F1 Score)
Multiplicative	BERT + Char CNN	100	***0.8429***	***0.7571***	***0.9443***
Multiplicative	BERT + Char CNN	128	0.8345	0.7463	0.9470
Multiplicative	BERT	100	0.8299	0.7409	0.9412
Multiplicative	BERT	128	0.8400	0.7378	0.9491
Regularizer	BERT + Char CNN	100	0.8316	0.7427	0.9476
Regularizer	BERT + Char CNN	128	0.8366	0.7502	0.9538
Regularizer	BERT	100	0.8440	0.7499	0.9424
Regularizer	BERT	128	0.8320	0.7378	0.9391

of the architecture comes handy in achieving the best performing model in FRAMES dataset, while the non-residual connections produce ideal results in the other two datasets. From Fig. 4.7, it is noted that the LSTM layer with 128 units outperforms the one with 100 units. The Self Attention width has been fixed at 15 for the task specific layers of Intent Detection and Dialogue Act Classification.



((a)) **((b))**

Figure 4.7: Impact of hyperparameters on FRAMES dataset

The Multiplicative self attention and Regularizer self attention have been tested upon, revealing the Regularizer. The hyperparameters which yield the optimal performance has been recorded for each dataset in Table 4.9

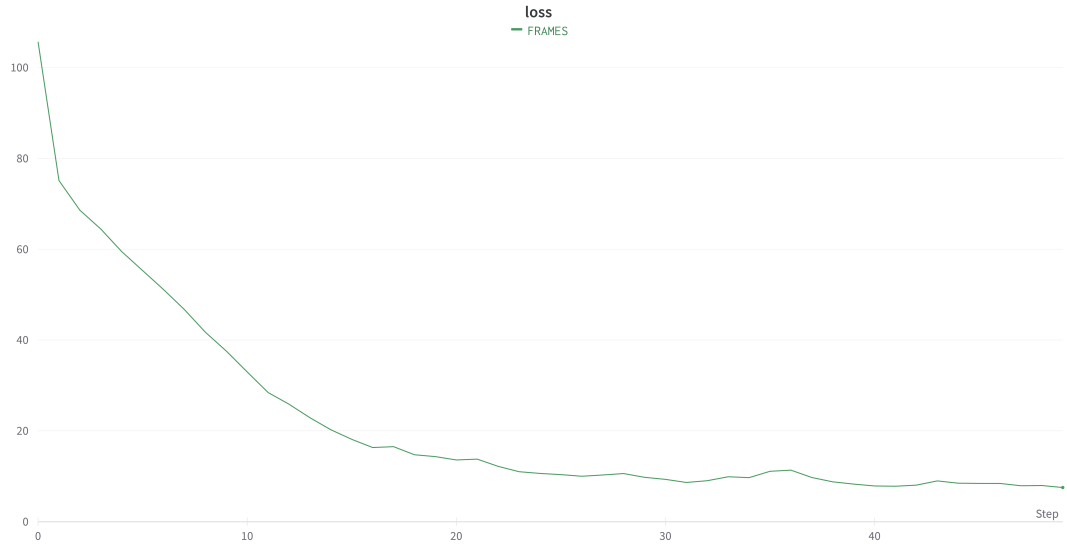
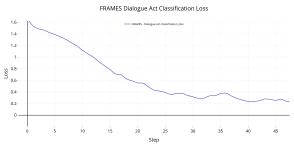


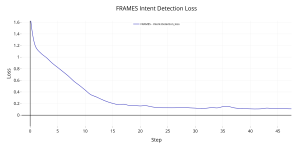
Figure 4.8: Overall Loss for FRAMES dataset

In order to ensure that the model does not overfit the training data, its loss values were constantly monitored during the training process. Early stopping mechanism and the multi task paradigm restricts the model from being overfit thereby ensuring optimal performance. The loss obtained for each tasks during the training carried out on the FRAMES datasets are as shown in Fig. 4.9.

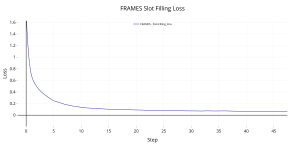
The overall loss which is the sum of the loss values for the individual



((a)) DAC Loss for FRAMES dataset



((b)) ID Loss for FRAMES dataset



((c)) SF Loss for FRAMES dataset

Figure 4.9: Loss for FRAMES dataset

tasks is shown in Fig. 4.9. The smooth reduction in the loss values implies that the performance improves with training.

Table 4.9: Best Performing Hyperparameters

Dataset	Self-Attention	Embedding	LSTM units	Residual Connections
ATIS	Multiplicative - 15	BERT + Char CNN	100	No
FRAMES	Multiplicative - 15	BERT + Char CNN	100	No
TRAINS	Multiplicative - 15	BERT + Char CNN	100	No

With the experimentations carried using various hyperparameters, the proposed intelligent encoding strategy which is the concatenation of the BERT and the Char CNN embeddings provides better performance that BERT embeddings in isolation. Non residual connections are preferred to the residual ones for the task shared layers of stacked Bi-LSTMs. Attention width of 15 for the multiplicative self attention fetches better outcomes than the regularizer attention.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In this work, a multi-task model, named CIDIS, has been proposed for Dialogue Act Classification, Intent Detection and Slot Filling which can harness the ability of the tasks interacting with each other to yield a better interpretation of the input query. As seen in the experiments, the intelligent encoding strategy consisting of BERT and CharCNN embeddings has improved the model's performance by benefitting all three tasks as compared to a single encoding technique. Compared to existing multi-task paradigms, in the ATIS dataset the model has outperformed in three tasks - Dialogue Act Classification, Intent Detection and Slot Filling by 0.47%, 0.48% in accuracy and 0.0078 in F1-score respectively, and in FRAMES, Dialogue Act Classification has been outperformed by 12.98%, Intent Detection has been outperformed by 13.28% and Slot Filling task has been outperformed with an F1 score of 0.9901. When it comes to TRAINS dataset, Dialogue Act Classification has been outperformed by 8.59%, Intent Detection by 6.03% and Slot Filling by an F1 score of 0.23. The superiority of the work can certainly be attributed to the usage of BERT embeddings due to their robust encoding strategy. This is due to the fact that these transformers were trained on huge numbers and varieties of disparate natural language corpuses that make sure that possible senses of most common words are covered in the training procedure. Once the supplementary outputs from all three tasks are obtained in a multi-task fashion, they are in turn fed to the final application module that uses them in a specific sequence of steps to arrive at the required final query response. This happens in real time as post understanding the input query at its grassroots level through the three clearly

defined Natural Language Understanding tasks, the required details are fetched from credible real-time sources by employing the process of web-scraping and are populated into a customised User Interface that the user can interact to get the inquiry results from.

5.2 Future Work

Taking forward, in our future work, the proposed model can be taken to a more gentrified system by attending to it at two levels.

- First, the fundamental modality could be altered where the current input query instead of being just one-shot could be transformed more conversational. In such a scenario, the query would consist of a sequence of sentences and hence, to cater to this particular case, the architecture should have a state parameter that stores the current state of the entire input sequence that's been processed currently. Here, the subsequent queries would not be completely independent of the penultimate query and that is how conversationality would come into play.
- Another approach is to include more details by employing Graph Convolutional Neural Networks (GCN) and Semantic Web structures that inject more structural and semantic dependencies into the system architecture thereby improving the performance of the model.
- In the second level, the application to which the model can be used for could be improved instead of an intuitional one. An automatic inquiry system could be employed in life saving situations like an army combat field. Here, instead of the a situation where the field soldiers have to contact the commander or intelligent services to

understand the current state of the mission or the position of the enemies manually by taking a lot of time, our system could be used where it should be trained on conversations of all previous missions. This would enable them to receive responses at a faster pace as the modality is completely automatic in nature.

- Another application is that of a spacecraft information system during interplanetary missions where instead of retrieving life saving details from earth, for which light has to travel back and forth, a system like this trained on conversations from previous missions of those sorts could be used to gather information at the mission site itself.

APPENDIX A

HUGGINGFACE TRANSFORMERS

Transformers provides means to easily download and train state-of-the-art pretrained models. Using pretrained models can reduce your compute costs, carbon footprint, and save you time from training a model from scratch. The models can be used across different modalities such as:

- **Text** : text classification, information extraction, question answering, summarization, translation, and text generation in over 100 languages.
- **Images** : image classification, object detection, and segmentation
- **Audio** : speech recognition and audio classification.
- **Multimodal** : table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

Our library supports seamless integration between three of the most popular deep learning libraries: PyTorch, TensorFlow and JAX. Train your model in three lines of code in one framework, and load it for inference with another.

APPENDIX B

BERT

BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible.

BERT is at its core a transformer language model with a variable number of encoder layers and self-attention heads. The architecture is "almost identical" to the original transformer implementation.

BERT was pretrained on two tasks: language modelling (15% of tokens were masked and BERT was trained to predict them from context) and next sentence prediction (BERT was trained to predict if a chosen next sentence was probable or not given the first sentence). As a result of the training process, BERT learns contextual embeddings for words.

REFERENCES

[1] Quansheng Dou and Panpan Cui. Slot filling using en-training. In *2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 271–276, 2019.

[2] Pratik Jayarao and Aman Srivastava. Intent detection for code-mix utterances in task oriented dialogue systems. In *2018 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 583–587, 2018.

[3] Manshu Tu, Bing Wang, and Xuemin Zhao. Chinese dialogue intention classification based on multi-model ensemble. *IEEE Access*, 7:11630–11639, 2019.

[4] Sławomir Dadas, Jarosław Protasiewicz, and Witold Pedrycz. A deep learning model with data enrichment for intent detection and slot filling. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3012–3018. IEEE, 2019.

[5] Su Zhu, Ruisheng Cao, and Kai Yu. Dual learning for semi-supervised natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:1936–1947, 2020.

[6] Libo Qin, Tailu Liu, Wanxiang Che, Bingbing Kang, Sendong Zhao, and Ting Liu. A co-interactive transformer for joint slot filling and intent detection. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8193–8197. IEEE, 2021.

[7] Mauajama Firdaus, Ankit Kumar, Asif Ekbal, and Pushpak Bhattacharyya. A multi-task hierarchical approach for intent detection and slot filling. *Knowledge-Based Systems*, 183:104846, 2019.

[8] Yanfei Hui, Jianzong Wang, Ning Cheng, Fengying Yu, Tianbo Wu, and Jing Xiao. Joint intent detection and slot filling based on continual learning model. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7643–7647. IEEE, 2021.

[9] Bamba Kane, Fabio Rossi, Ophélie Guinaudeau, Valeria Chiesa, Ilhem Quénel, and Stéphane Chau. Joint intent detection and slot filling via cnn-lstm-crf. In *2020 6th IEEE Congress on Information Science and Technology (CiSt)*, pages 342–347. IEEE, 2021.

[10] Hao Tang, Donghong Ji, and Qiji Zhou. End-to-end masked graph-based crf for joint slot filling and intent detection. *Neurocomputing*, 413:348–359, 2020.

[11] Waheed Ahmed Abro, Guilin Qi, Zafar Ali, Yansong Feng, and Muhammad Aamir. Multi-turn intent determination and slot filling with neural networks and regular expressions. *Knowledge-Based Systems*, 208:106428, 2020.

[12] KJ Jose and KS Lakshmi. Joint slot filling and intent prediction for natural language understanding in frames dataset. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pages 179–181. IEEE, 2018.

[13] Ieva Staliūnaitė and Ignacio Iacobacci. Auxiliary capsules for natural language understanding. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8154–8158. IEEE, 2020.

[14] Pin Ni, Yuming Li, Gangmin Li, and Victor Chang. Natural language understanding approaches based on joint task of intent detection and slot filling for iot voice interaction. *Neural Computing and Applications*, 32(20):16149–16166, 2020.

[15] Anda Stoica, Tibor Kadar, Camelia Lemnaru, Rodica Potolea, and Mihaela Dînşoreanu. The impact of data challenges on intent detection and slot filling for the home assistant scenario. In *2019 IEEE 15th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 41–47. IEEE, 2019.

[16] Peng Yang, Dong Ji, Chengming Ai, and Bing Li. Aise: Attending to intent and slots explicitly for better spoken language understanding. *Knowledge-Based Systems*, 211:106537, 2021.

[17] Chen Tingting, Lin Min, and Li Yanling. Joint intention detection and semantic slot filling based on blstm and attention. In *2019 IEEE 4th international conference on cloud computing and big data analysis (ICCCBDA)*, pages 690–694. IEEE, 2019.

[18] Mauajama Firdaus, Hitesh Golchha, Asif Ekbal, and Pushpak Bhattacharyya. A deep multi-task model for dialogue act classification, intent detection and slot filling. *Cognitive Computation*, 13(3):626–645, 2021.

[19] Anda Diana Stoica, Andrei-Cristian Rad, Ioan Horia Muntean, George Daian, Camelia Lemnaru, Rodica Potolea, and Mihaela Dinsoreanu. The impact of romanian diacritics on intent detection and slot filling. In *2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pages 1–6. IEEE, 2020.