Neural Intent Recognition for Question-Answering System

Ajinkya Pradeep Indulkar [1], Srivatsan Varadharajan [2], Krishnamurthy Nayak [3]
[1,3] *Department of E&C, Manipal Institute of Technology, Manipal*
[2] *Philips Lighting India Limited, Bangalore*

***Abstract*- Conversational Agents, commonly known as Chatbots, are a successful result of the collaboration of Natural Language Processing (NLP) and Deep Learning. Major Technology Giants like Google, Amazon, Microsoft, etc. are heavily invested in developing a sophisticated conversational agent which can pass the Turing Test. There are various methodologies which can be used to develop the various components of such an agent from scratch. This paper uses SQuAD, an open Question-Answering dataset, for developing an Intent Recognition System for any Question-Answering system. Inspired by Author-Topic Modelling, a Title-Topic Modelling technique is used in combination with various Deep Learning models to train the Intent Recognition System, achieving an accuracy of 88.36%.**

***Index Terms*- Conversational Agents, Deep Learning, Intent Recognition, Natural Language Processing, Question-Answering System, Title-Topic Modelling.**

1. INTRODUCTION

The origin of conversational agents can be traced back to 1950 when Alan Turing, father of Computer Science, published his first paper [1]. His question, "Can Machines think?" ushered the world into the era of Artificial Intelligence (AI) which began a plethora of research to turn his question into reality.

Numerous sub-fields of AI like Machine Learning, Computer Vision, and recently, Deep Learning have seen more researchers committing themselves to develop novel techniques in the respective fields.

The development of conversational agents falls under the field of Natural Language Processing, which is an area of research concerned with the understanding of human (natural) languages by the machines. Similar to Artificial Intelligence, its origin can also be traced back to 1950s when Alan Turing proposed a criterion for intelligence, known as the "*Turing Test*". A machine passes the test when a human is unable to distinguish the machine's responses from that of a human's.

ELIZA [2] was the first implementation of a human-like dialog system, developed in 1960s. It worked on a rule-based methodology. Since then, the number of research papers published in this field is overwhelming.

NLP Techniques like Dependency Parsing, Named Entity Recognition, etc. are vital nowadays in the development of the most sophisticated conversational agent.

A. Motivation

The current scenario of the chatbot industry mainly involves detecting the intent of a user utterance to the conversational agent by extracting entities and using classic machine learning algorithms such as Support Vector Machines to develop a classifier for the pre-defined intents. This is big step from the previous, rule-based approach but still lacks the true AI capabilities. Natural Languages are complex, and a human can express an intent in more than one way. To be able to understand such complexities can allow any machine to pass the "Turing Test".

This paper explores methodologies involved in developing a Question-Answering system using the concepts from NLP and Deep Learning.

The Neural Natural Language Understanding engine uses Intent Recognition System to understand the intents expressed by the user. For training the Intent Recognition System, Topic Modelling techniques are used to automatically label the training data, as the original dataset is unlabeled.

The techniques used are modular, i.e. they can be used individually in other systems according to the researcher's requirement.

II. LITERATURE REVIEW

A. Word Embeddings

Word Embedding can be defined as a distributed vector representation of words in a sentence or document. It is a technique of projecting words onto the vector space and this proves to be highly useful in various NLP tasks like statistical language modelling, machine translation, etc. The vectors of similar words or phrases can be observed to be closer in the vector space and these vectors represent the importance of a word in a document.

This also helps in finding the semantic as well as syntactic similarities between words and phrases. Since 1986, there has been a great deal of research in introducing new techniques of learning word embeddings. There have been Neural Network Language Model approaches, but this paper implements the Log-Linear Model approach, commonly called as *Word2Vec* [3].

Word2Vec is a log-linear model architecture which is computationally-efficient, and a simple approach compared to the neural network model in learning vector representations of words or phrases in a sentence in the training dataset.

There are 2 architectures of this model:

1. *Continuous Bag-Of-Words* (CBOW) Model
2. *Continuous Skip-Gram* (SG) Model

The Continuous Bag-Of-Words model predicts the probability of a target word from the context in the source sentence, whereas Skip-Gram model performs the opposite task of predicting the context words from a target word. The learning complexity of both models are:

1. CBOW Model:

$$Q = N \times D + D \times log_2(V)$$

2. Skip-Gram Model:

$$Q = C \times (D + D \times log_2(V))$$

Where, Q – Training Complexity, N – number of previous words, D – Dimension of the projection layer (N x D), V – size of the vocabulary, C – maximum distance of words.

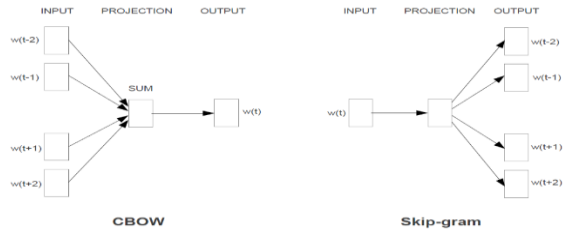

Figure 1: CBOW and Skip-Gram Model Architectures [3]

Fig. 1 clearly describes the architecture of the CBOW model and the Skip-Gram model as mentioned above. It is observed that CBOW model works better than Skip-Gram as it has the effect of smoothening over a lot of the distributional information. Thus, the CBOW model has been chosen for training the word vectors on the source dataset.

While implementing the CBOW model, both the input and target data are one-hot encoded, which is a binary encoding technique where '1' represents the presence of data and '0' represents the absence of it. Both the layers are of the size [1 x V]. In this architecture, there are 2 sets of weights, one between the input layer and the hidden layer, and the other between the hidden layer and the output layer. N has been chosen as the size of the hidden layer. Thus, the dimensions of the weights are [V x N] and [N x V] respectively.

As seen in Fig. 2, The hidden layer uses a *Linear* Activation function and the Output Layer uses a *Softmax* function, a generalization of the logistic function, which works on the *Maximum Likelihood (ML)* principle of maximizing the probability of the next word, w_t, given a set of previous words, h. The softmax function can be represented as:

$$P(w_t, h) = \text{softmax}(\text{score}(w_t, h)) = \frac{\exp\{score(w_t, h)\}}{\sum_{Word\ w'\ in\ Vocab} \exp\{score(w', h)\}}$$

Where *score* (w_t, h) calculates the compatibility of word w_t with the context h. We now train the model by maximizing the log-likelihood on the training dataset.

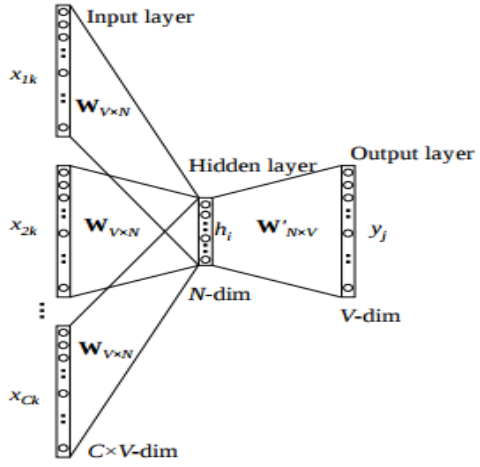

Figure 2: Diagrammatic Representation of CBOW Model for multiple context words [4]

Thus, we maximize,

$$J_{ML} = \log P(w_t, h)$$
$$= score(w_t, h)$$
$$- \log \sum_{Word\ w'\ in\ Vocab} \exp\{score(w', h)\}$$

Sub-sampling of most occurring words [4] in the training dataset is useful as such words provide less information than the rarely occurring words.

This technique thus improves the learning of the word embeddings. The approach used can be computed as follows:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Where, P(w_i) – Probability, f(w_i) – frequency of word w_i, and t – a chosen threshold.

These word embeddings are later used in a Neural Network Architecture designed for the Intent Recognition System.

B. Topic Modelling

Topic Modelling is a method of categorizing a set of documents into various topics. It is an unsupervised technique and thus it is different from other rule-based text mining approaches. The model observes the collections of words in the document corpora and clusters them into different topics. Evidently, this

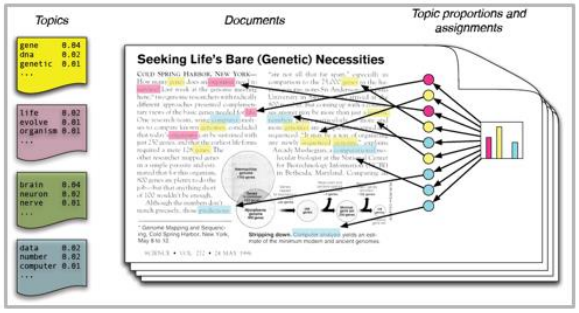

Figure 3: Representation of LDA Topic Modelling [5]

technique is mainly used for document clustering, organization of large blocks of textual data and information retrieval from unstructured text in the dataset which can be used for feature selection.

There are many algorithms for Topic Modelling, but the *Latent Dirichlet Allocation* (LDA) [5] technique for Topic Modelling is found to be the most efficient and accurate.

As evident from Fig. 3, LDA assumes the documents to be produced from a mixture of latent topics. The topics then generate words based on the probability distribution. As LDA is a matrix factorization technique, any corpus (collection of documents) in the vector space can be represented as a *Document-Term matrix*.

The Document-Term matrix is further converted to lower dimensional matrices, M1 and M2, where M1 is the Document-Topic matrix and M2 is a Topic-Term matrix. As these distributions are required to be improved, the core concept of LDA plays an important role. It uses sampling techniques to improve the above matrices.

It iterates through each word *w* in every document *d* and tries to adjust the current Topic-Word assignment with a new assignment. A new topic *k* is assigned to word *w* with a probability *P* which is a product of two probabilities *p1* and *p2*. For every topic, two probabilities p1 and p2 are calculated.

$$p1 = p(topic, t \mid document, d)$$

Where, *p1* represents the proportion of words in document *d*, currently assigned to topic *t*.

And,

$$p2 = p(word, w \mid topic, t)$$

Where, *p2* represents the proportion of assignments to topic *t* over all documents that come from this word *w*.

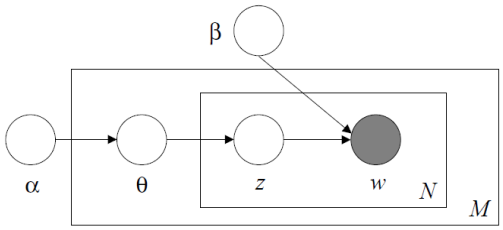

Figure 4: Graphical Model Representation of LDA

The current topic-word assignment is then updated with a new topic whose probability is the product of *p1* and *p2*. In this step, the model assumes that all the existing word-topic assignments except the current word are correct. Essentially, this is the probability that topic *t* generated word *w*, so it makes sense to adjust the current word's topic with new probability. After several iterations, a stable state is achieved where the document-topic and topic-term distributions are optimal leading to the convergence point of LDA.

This concept can be understood mathematically and as LDA is a generative process, the following assumptions can be made:

1. Choose N ~ Poisson(ξ)

2. Choose θ ~ Dir(α)
3. For every *N* words w_n,

a. Choose a Topic z_n ~ Multinomial(θ)

b. Choose a word w_n from $p(w_n | z_n, \beta)$, where p is a multinomial probability under the condition of the topic z_n.

Where, the Dirichlet Distribution has dimensionality equal to k which is also the dimensionality of z. The matrix β with a dimension of $k \times V$ acts as a parameter for the word probabilities, where $\beta_{ij} = p(w^j == 1 | z^i == 1)$.

As seen in Fig. 4, with α and β parameters and *M* documents in the corpora, the joint topic distribution for a collection of *N* topics *z* and *N* words *w* can be formulated as,

$$p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n \mid \theta) p(w_n \mid z_n, \beta)$$

C. Title-Topic Modelling

LDA Topic Modelling is a very efficient technique for extracting topics from documents but when documents belong to certain category or include overlapping titles, it is possible to have a less accurate topic modelling has documents can fall under wrong topics.

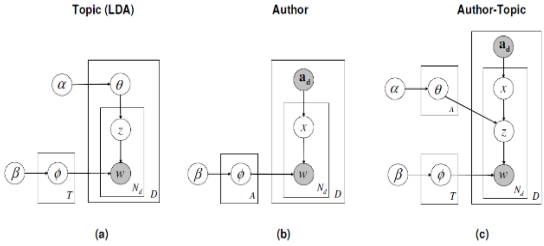

Figure 5: (a) LDA Topic Model (b) Author Model (c) Author-Topic Model [6]

Inspired by the *Author-Topic Model* [6], this paper considers the generalized titles as a metaphorical author of the documents.

As observed in Fig. 5(c), for D documents having A authors, T topics are extracted using the LDA as the foundation to calculate author-topic distributions a_d alongside topic-word and document-topic distributions.

Hence, the various authors (or more specifically titles) are categorized under the extracted topics. Hence, this paper uses a modified version of the Author-Topic Modelling, called *Title-Topic Modelling*, to extract topics for the various titles of the training dataset.

D. Recurrent Neural Networks

Over years, Artificial Neural Networks have outperformed other learning algorithms due to its ability to learn more complex and high-level features from data of all kinds using hidden layers which improve the accuracy of a prediction or classification model.

But one major limitation of such neural networks is that they can't understand sequences, i.e. how the current state is affected by the previous states of the input. The technique of sequence retention based on time was first introduced in *Hopfield Networks*[7]. Hopfield Networks can store memory of the input sequence which can be addressed based on content and it thus falls under the category of *Recurrent Neural Networks*, or RNN.

The intuition behind RNN is that one or more hidden layers of previous timesteps are stacked on top of each other, and each hidden layer depends on their respective inputs at a certain timestep and the previous timestep.

This can be observed in Fig. 6 and calculated using:

$$h_t = f(W_{xh} x_t + W_{hh} h_{t-1})$$

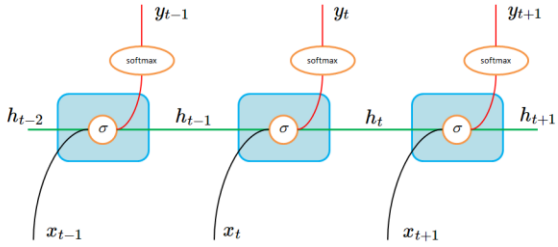

Figure 6: Representation of a Basic RNN Cell

Where, W_{xh} is the weight matrix between the input layer and the hidden layer, W_{hh} is the weight matrix within the hidden layer, x_t is the input at timestep t and h_{t-1} is the hidden layer output of the previous timestep $(t-1)$ which is fed into the next neuron of the hidden layer along with the input of the next timestep.

The output from each neuron in the hidden layer can be calculated using:

$$y_t = softmax(W_{hy} h_t)$$

Where, W_{xh} is the weight matrix between the hidden layer and the output layer, h_t is the hidden layer neuron output at the current timestep *t*.

Thus, Recurrent Neural Networks, with hidden layers inclusive of different timesteps, have the ability to remember sequences. But it can't store the memory over a long timestep due to a phenomenon known as the *Vanishing Gradient*.

Vanishing Gradient causes the network to be indecisive about which information of some timestep is valuable, and thus store it, and which information is not valuable, and thus discard it. Hence, an improvement is required, and this is where *Long Short-term Memory* [8], or LSTM, proves to be vital.

E. Long Short-Term Memory Networks

LSTMs are an improvement over other Recurrent Neural Network Models, which had been implemented since Hopfield Networks, due to its faster training, time complexity of $O(1)$ and overall better performance in solving complex, time-based input sequences.

But to understand LSTM Network which is represented in Fig. 7, it must be compared with the basic RNN Model.

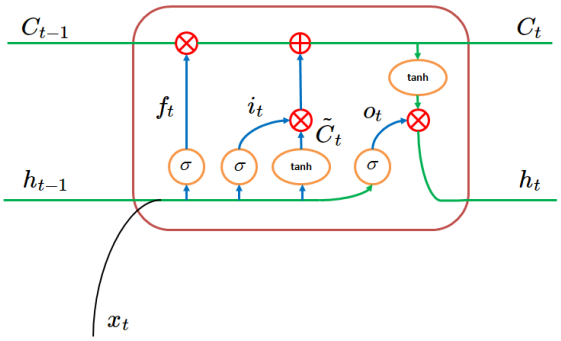

Figure 7: Representation of an LSTM Cell [8]

The output in an RNN architecture can be calculated as:

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

Comparing to RNN, the above equation is exactly like the RNN used to compute the hidden state at timestep t. But it's not the true hidden state in terms of LSTMs, hence we named it as o_t.

Firstly, the LSTM is given the ability to *forget*, which means that it can decide whether to forget the previous hidden state or not. This is achieved by adding a *Forget Gate Layer*. The output of this layer is computed as:

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

In contrast to the forget gate layer, to instruct the model whether to update the current state using its previous state, we need to add an *Input Gate Layer*. The output is calculated as:

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

Next, we compute the temporal cell state for the current timestep. It looks just like the output of RNN above, except that *tanh* activation function is used:

$$\tilde{C_t} = tanh(W_C[h_{t-1}, x_t] + b_C)$$

The *actual* cell state is thus computed for the current timestep t, using the forget gate and input gate, as mentioned above.

Intuitively, doing so helps the LSTM to keep only the necessary information and forget the unnecessary ones.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t}$$

After the current cell state is computed, it is used to compute the current hidden state as follows:

$$h_t = o_t * tanh(C_t)$$

Finally, the output of the hidden state for the current timestep has been calculated.

The remaining task is like that of the RNN model, which is computing the *actual* output y_t:

$$y_t = softmax(W_{hy} h_t)$$

F. Convolution Neural Networks

Convolutional Neural Network, or CNN is a class of feed-forward neural network mainly used for classification in image recognition related applications. The concept of CNNs was first introduced in *Neocognitron* [9] which is a self-learning model for spatially-invariant mechanism for pattern recognition.

Over the years of research, CNNs have been mainly been used for image classification [10] or object detection. But use of CNNs in text classification in have shown promising results. Hence, it's relevant to understand the concept of the convolutional neural network.

CNNs mainly differ from any other neural network as it considers the input as an image which is two dimensional in nature and regular neural networks don't work as excepted with images.

For a clearer understanding, if a color images of size $32 \times 32 \times 3$ are inputs to a regular neural network then the first hidden layer includes 3072 weights. This number of weights become unmanageable when the size of input images increases, say to $200 \times 200 \times 3$, because this would lead to the network

utilizing more neurons per hidden layer and this can result in overfitting.

Overfitting is a Machine Learning concept where a model completing fits the training data and produces

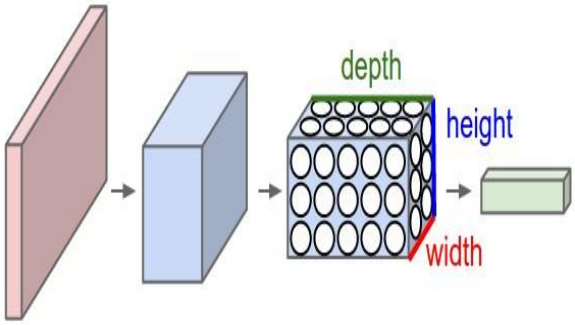

Figure 8: Representation of a CNN Architecture [10]

a 100% accuracy but fails when new test data is introduced to the model.

With such a situation, Convolutional Neural Networks provide a reasonable alternative by the use of 3D neurons as seen in Fig. 8. All the layers of the CNN have 3 dimensions: Width, Height and Depth (it can also refer to the number of channels in an image). This helps the CNN to extract high-level and complex features from the input and encode it as weights to the model with lesser parameters compared to regular neural networks.

But as this paper is more NLP oriented, CNN is used to extract features from the 3D output of the LSTM layer in the Neural NLU Engine. Every layer in CNN transforms an input of 3 dimensions to an output of 3 dimensions with an activation or differentiable function.

There are mainly 3 types of layers in a CNN architecture:

1. Convolutional Layer:

This layer performs convolution on a small regions of input images by performing a dot product between the weights of the layer and the regions of the input image with a filter of different kernel sizes.

2. Pooling Layer:

This layer transforms the input using the down-sampling operation along the width and height, which are considered as the spatial dimensions.

3. Fully-Connected Layer:

This layer is similar to a regular neural network which is used as the output layer in any architecture to perform the classification based on weights from the previous layers.

III. METHODOLOGY

A. Selection of Dataset

Any Machine Learning algorithm requires a dataset which fits perfectly to train a model for the desired purpose. The success of any such paper heavily depends upon how good the dataset is to train the model. There are a number of open-source datasets available for building models for NLP applications. Among such datasets, this paper chose the Stanford Question Answering Dataset (SQuAD) [11]. SQuAD is an extensive dataset mainly created for building machine comprehension systems. It includes more than 100,000 questions and its respective answers created by crowd workers on over 20,000 articles scraped from Wikipedia ranging around 500 individual topics.

Such a large dataset with a huge text corpus is optimum for building highly accurate deep learning models for NLP applications. Although this dataset is mainly targeted for building machine comprehension systems, which can be defined as a system which can produce an answer to a question related to a passage which is fed into the system, this paper extracts only the articles which acts as the training text corpus and the individual questions and their answers as a Knowledge Base. Such open-source datasets are not always structured in a manner required by the task at hand. Hence, data pre-processing is a vital part of any machine learning task where the data is extracted and arranged in a manner most suitable for the further steps. The code used for extracting and storing the data, in excel format, can be found in the Appendix.

B. Generation of Word Vectors

After the dataset is processed and structured in the required manner, the first task towards designing a Neural NLU Engine is to learn the vectors of all the words in the vocabulary of the training text corpus. As mentioned in the *Literature Review* section, this paper uses the Word2Vec approach to learn word vector representations which is later used in building the Intent Recognition System. To implement Word2Vec on Python, Gensim is used. Gensim is a python package mainly for performing Topic Modelling on a collection of documents but also

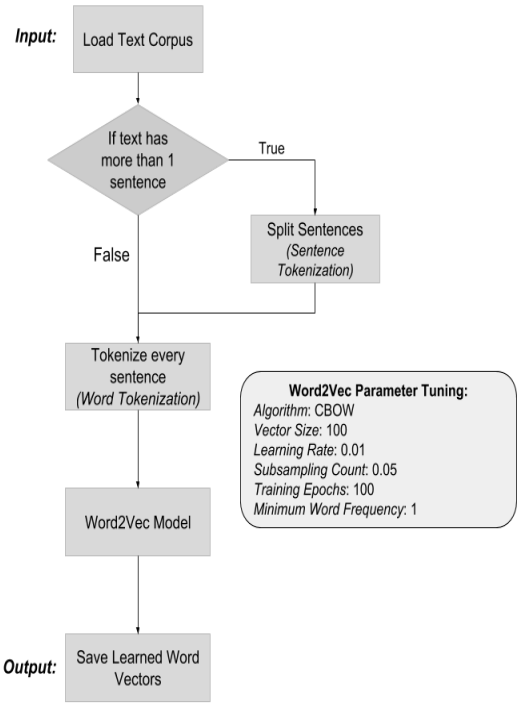

Figure 9: Training Flow for learning Word Vectors

provides implementations of different word embedding algorithms amongst which Word2Vec is used by this paper.

As observed from Fig. 9, the Word2Vec implementation begins with loading the text corpus generated from SQuAD, which is to be used to generate word vectors. Before using the text corpus as an input to the Word2Vec Model, it is vital to perform some text pre-processing steps which can be listed as follows:

- Tokenize every text into multiple individual sentences if the text has more than 1 sentence.
- Tokenize every sentence into a list of words.

Once, the training corpus is clean and ready to be trained, the Word2Vec model from the Gensim is used to train over the training corpus to learn all the word vector representations.

The CBOW Algorithm is used for the above training with a vector size of 100 dimensions, setting the learning rate to 0.01 and subsampling count of 0.05 for common words. The Word2Vec model trains for over 100 epochs (iterations) with the minimum frequency of 1 for any word, meaning the model considers all the words in the entire training corpus.

Once the model training is complete, the model is saved in a text format with every line representing a word and its vector representation. t-SNE (t-distributed Stochastic Neighbor Embedding), a dimensionality reduction technique is used to visualize all the word vectors over the vector space.

C. Extracting Latent Topics

Topic Modelling is unsupervised in nature and thus it can be used to find latent topics from a text corpus with the help of LDA (Latent Dirichlet Allocation) Algorithm. But it is observed that the Author-Topic Model, which is based on the LDA Model is more efficient with SQuAD as each document in the text corpus is labelled with a title.

Thus, the Author-Topic Model can be modified to work like a *Title-Topic Model* in which latent topics are extracted from the titles of the text corpus. As shown in Fig. 10, for the title-topic modelling, the model first loads the target corpus and performs the following pre-processing steps:

- Removing whitespaces, digits and non-ASCII characters.
- Removing stopwords. (This improves the accuracy of the model as stopwords are always in a high frequency in any document).
- Lemmatizing other words and tokenizing each text, transforming it to a list of tokens (words).
- Extending the list of tokens with the entities extracted by the NER, using a python package called Spacy, and bigrams (pair of words) based on collocation of words in a document.

After the documents are pre-processed and ready to train, it is important to generate unique IDs for all documents as well as unique titles of the entire text corpus. These document and topic IDs are then used to create a title to document mapping where every title is mapped to a list of documents labelled to the respective titles.

Before training the author-topic model, it is important to create a dictionary of all unique words in the entire corpus and filtering out extremes, i.e. removing words occurring less than 20 times more than 50% of the entire corpus. This also adds to the improvement of the model accuracy.

To train the model, the corpus needs to be transformed to the Bag-of-Words Representation. Once all the above steps are completed, the author-

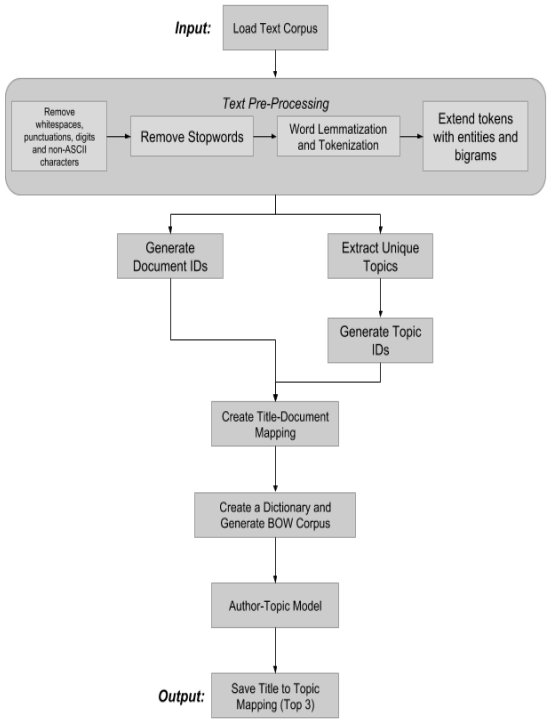

Figure 10: Training flow of Author-Topic Modelling

topic model is trained over the transformed corpus for 25 topics. The alpha and eta parameters are set to *'auto'*. This indicates that these parameters update themselves based on the training corpus.

The Model is trained 5 times with a different random state and the model with the highest coherence value is chosen as the winner model. Coherence is a metric to determine the accuracy of topic modelling, mainly used to find the number of topics which can be extracted from a training corpus.

The trained model is then saved and a title to topic mapping is created as a dataset where every title is mapped to only 3 highest probable topics, as every title can have a mixture of latent topics. With manual inspection, the 25 topics are reduced to 18 topics by combining a few topics on the account of them being similar. Any interactive visualization tool can be used to view the probabilities of tokens (words) in every topic.

D. Intent Recognition System

All the techniques mentioned before play a specific part in completing the design of the Intent Recognition System. This system can be defined as an important component of the Neural NLU Engine where the first step is to understand the intent of a user.

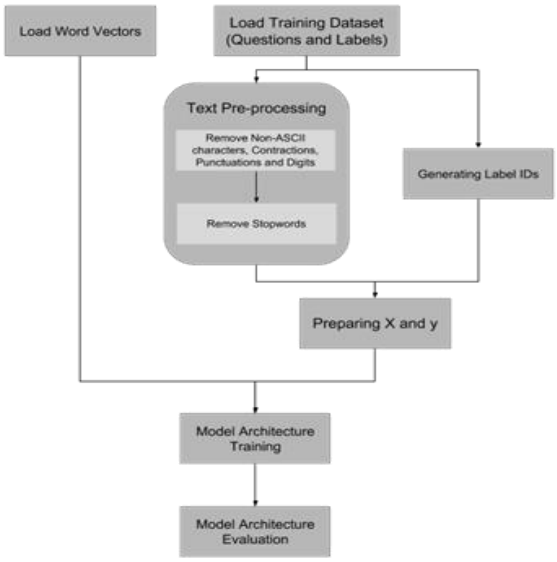

Figure 11: Training Flow of Intent Recognition System

An Intent can be understood as to what the user is trying to express in a sentence. Once the intent has been recognized by the Neural NLU Engine, any conversational agent using the NLU Engine can proceed with the Selection Response System which selects the closest similar question from the database and responds with its corresponding answer.

As shown in Fig. 11, the dataset and the word vectors are loaded which is followed by the performing text pre-processing steps on the questions from the training dataset. While loading the word vectors, a vocabulary of all the unique words is also created and every word is assigned an index value which is later used to transform the training questions to an array of word indices. As the labels against the questions are of *string* datatype, label IDs are generated which convert the list of training labels to an array of integer values representing the output classes of the Intent Recognition System.

While generating a Question-Index matrix *X*, every question in the training dataset is tokenized, and an empty matrix is filled with the word vectors of all the words in the sentence.

This acts as a numerical representation of a sentence which is then fed as input while training the neural network model. A maximum length for every sentence is decided so that all the sentences are equal in length. Any shorter sentence is padded at the end and longer sentences are clipped off at the maximum length.

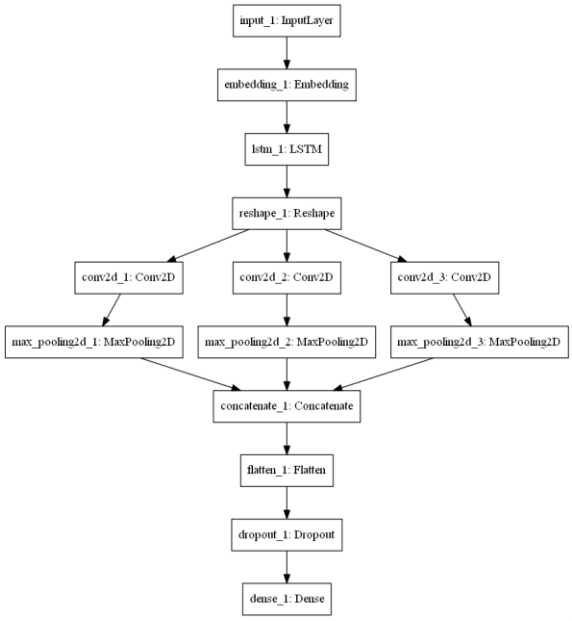

Figure 12: LSTM+CNN Model Architecture

For building the Neural Network Architecture, the paper uses *Keras* which is a framework running on top of Tensorflow, a Python Deep Learning Library developed by Google.

The Architecture includes an Embedding Layer which acts as the Input Layer and a Dense Layer is used as the Output Layer. For the Hidden Layers of the Neural Network Architecture, the paper implements a combination of LSTM and CNN Layers, as observed in Fig. 12.

While compiling the network, the model uses *Categorical-Crossentropy* as the loss function, *Adam* as the optimizer with a learning rate of *0.001* and *accuracy* as the model metric. The Neural Network architecture includes an LSTM layer of size 100 and a concatenation of 3 CNN Layers with 2D Convolutions and Max Pooling with filter sizes of 10, 15 and 20.

Once the designed model is compiled, it is trained on the transformed training dataset for a certain number of *Epochs* (Iterations) and Batch Size. Once the training is complete, the model is tested and validated on the test dataset for checking the model's accuracy. Once the desired accuracy is achieved, the model can is saved.

This saved model is the main component of the Intent Recognition System.

Title	Question	Answer
Antenna (radio)	What radiates two lobes perpendicular to the antennas axis?	horizontal dipole
Napoleon	After his election to First Consul, where did Napoleon take up residence?	the Tuileries.
Thuringia	What is one company that is able to get investments from large companies?	the optics sector at Jena.
Thuringia	What is the top priority of the federal trunk road programme 2015?	The upgrading of federal highways
Napoleon	A 2008 study supporting the original autopsy findings related to Napoleon's death analyzed samples of what substance taken from Napoleon and his family?	hair
USB	A device that is HS capable first connects as a what?	an FS device (D+ pulled high)
USB	a larger three-byte SPLIT token with a seven-bit hub number, 12 bits of control flags, and a five-bit CRC were created to do what?	to perform split transactions
Antenna (radio)	A sphere shows what type of antennas radiation?	isotropic
Association football	A team players layout is a what?	formation

Figure 13: Extracted Questions and Answers

IV. RESULT ANALYSIS

A. Dataset Extraction

In the Methodology section, the SQuAD was described as a dataset mainly used for building machine comprehension systems. But to structure it in a form required by the project, The JSON file had to be parsed to extract titles, content (Wikipedia articles on all titles) and a set of questions and respective answers on every article.

Fig. 13 shows the extracted questions and the respective answers from SQuAD.

B. Word Vectors

As mentioned in the *Methodology* section, the word vector representations are generated using the CBOW algorithm implemented on Python with the help of Gensim. Fig. 14 depicts the word vector representation of a single word as an example.

As it is evident from Fig. 16, the word "*project*" is represented as a vector of 100 dimensions.

Similar to this, the word vector training produced vectors for a little over 100,000 unique tokens (words) from the training corpus. It is possible to find similar words closer to each other in the vector space.

Representing words in a vector space has many advantages as follows:

- Similar words are closer to each other in the vector space.

```
The Word Vector Representation of 'project' :
 [-0.25496444 -0.55840325 -0.76015127 -0.5206102  -0.05711009 -0.16893233
  0.4006314   0.4989204  -0.07414585 -0.3285911  -0.3237757  -0.5940167
 -0.45739222  0.2593712  -0.20143783  0.11004055 -0.15515247  0.05783053
  0.3979669   0.6334464  -0.33665717 -0.08583366  0.7447912   0.26526138
 -0.08053984 -0.273244   -0.4198597  -0.01068262 -0.28419784 -1.0290895
 -0.1011768  -0.6740706  -0.0365251  -0.00343671 -0.31162536  0.0534851
  0.08081938  0.09094461  0.18415391  0.22602752 -0.19347978 -0.4812544
  0.2963764   0.4261831  -0.80784494  0.5995202  -0.0676647  -0.8361303
  0.3536522  -0.8209487  -0.3447851   0.1868344  -0.03822445  0.24933365
 -0.6864659   0.56760883 -0.2840312  -0.31806526  0.3937343   0.7144876
 -0.10789369 -0.1624514  -0.09706189 -0.12562518  0.52469915  0.14201398
 -0.7572984  -0.19066998 -1.221761    0.05007374  0.28505635  0.7058262
 -0.1091525   0.6288783  -0.42591748 -0.20335889 -0.47259474 -0.28197122
  0.01326941  0.69044614  0.15742107 -0.78402257  1.0195425  -0.69100577
  0.20307675 -0.29747763 -0.9128023  -0.14375855  0.59738606 -0.49104837
  0.15720955  0.75287914  0.32588047 -0.01356937  0.6401633   0.29308036
 -0.3222943  -0.27664122 -0.00217398  0.01822198]
```

Figure 14: Word Vector Representation Example

- Vector arithmetic like $w2v(king) + w2v(man) - w2v(queen) = w2v(woman)$, where $w2v()$ represents a vector representation of a word, holds true.

Fig. 15 shows an example of the most similar words of the word "*history*".

```
model.most_similar('history')
```

```
[('science', 0.5673647522926331),
 ('natural', 0.5557897686958313),
 ('life', 0.5481092929840088),
 ('century', 0.5325499176979065),
 ('modern', 0.5251394510269165),
 ('cultural', 0.4959450960159302),
 ('culture', 0.4882017970085144),
 ('last', 0.4864061176776886),
 ('country', 0.48067009449005127),
 ('europe', 0.47800108790397644)]
```

Figure 15: Most Similar Words Example

As observed from Fig. 16, around 500 words from the entire vocabulary are represented in a vector space. As the vector dimension of all words is 100, it is necessary to reduce the dimensionality of all vectors.

Hence t-SNE is used to reduce the dimensions to 2 which then plotted in a scatter form. For a better understanding of the word vectors, it is can be viewed in the vector space that similar words are clustered together just as mentioned above.

C. Title-Topic Modelling

As discussed in the *Methodology* section, instead of using the LDA approach, the paper uses Title-Topic Modelling to find latent topics of the unique titles of the SQuAD. Fig. 17 describes the flow of SQuAD from its original form to a labelled dataset for the Intent Recognition System.

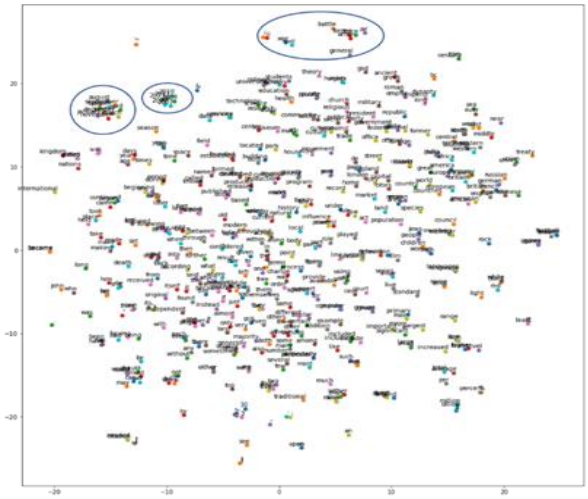

Figure 16: Word Vector Space

In Fig. 18, the 25 topics are reduced to 18 labels upon manual inspection. Hence, the end result of the Title-Topic Modelling is the reduction of *490* titles to *18* labels. This is around *96%* reduction of possible intents of a user question.

During the experimentation phase, it is observed that when Topic Modelling using LDA is implemented, the model gains an accuracy of around *60%* whereas when the Author-Topic Modelling is implemented, the model accuracy increased by almost 18%, thus giving a *78%* accuracy of the model to predict the labels of titles from the text corpus.

D. Intent Recognition System

During the experimentation phase of building the Intent Recognition System, a number of neural network architectures were tested for high accuracies. The training dataset is a collection of almost *90,000* questions labelled during the Title-Topic Modelling.

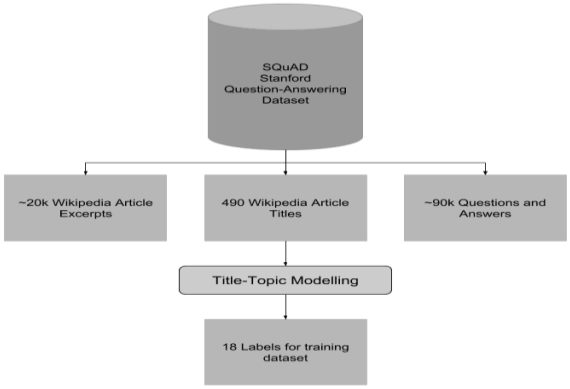

Figure 17: SQuAD Data Flow

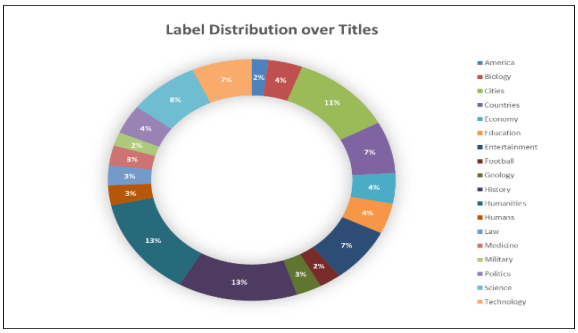

Figure 18: Label Distribution over Titles

This phase led to the design of a Neural Network Architecture for classifying a user question to *18* intents. The approach for designing the model architecture in the Intent Recognition System involved using an LSTM+CNN model architecture. In this architecture, the input Sentence-Index matrix is fed into the Embedding layer with pre-trained word vectors.

The Hidden Layers include an LSTM layer followed by three 2D-Convolution Layers with filter sizes of 10, 15 and 20. The outputs of these layers is then concatenated, flatten and fed into the Fully Connected Layer which outputs a vector of 18 dimensions.

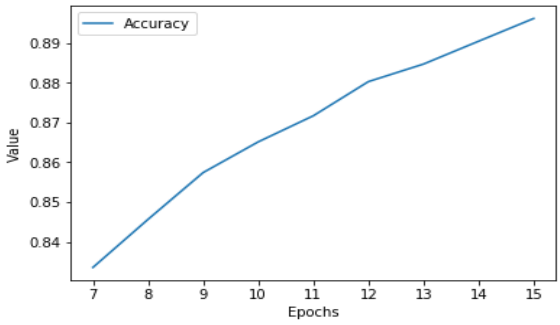

Figure 19: Model Accuracy

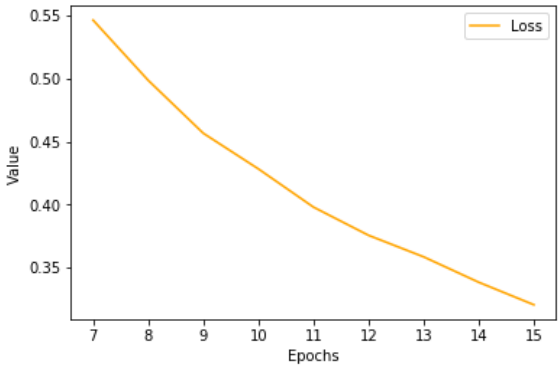

Figure 20: Model Loss

This model also transforms the input to a probability vector for 18 labels in which the label with the maximum probability is chosen as the winner intent. The LSTM+CNN Model Architecture achieves a validation accuracy of *88.36%*.

This model is trained over 20 epochs (iterations) and the training metrics can be observed in Fig. 19 and Fig. 20. It is important to observe the efficiency of this Approach as it is able to achieve a very high accuracy in few epochs.

V. FUTURE SCOPE

The models designed and trained can be improved with better and more complex algorithms, but they currently perform well in a working environment. The word vectors can be improved with continuous hyperparameter tuning, exploring other techniques, as well as using a larger text corpus for the Word2Vec algorithm to run more efficiently. The Named Entity Recognition System can be improved by training it with more entities, labelled in the training corpus.

The Author-Topic Modelling can be improved with better document representations instead of using the current Bag-Of-Words format. The technique used to represent words in a document is an important parameter at achieving a higher accuracy. But with complexity comes the trade-off in regards with the infrastructure necessary for training such models. All the current models have been trained on a system with an *Intel Core i5* Microprocessor and *8GB* RAM. To train models with higher complexity and deeper layers, high-speed GPUs are required as they train a neural network much faster than a CPU does due to the concept of Parallelism which is faster on a GPU for performing all matrix operations. The Intent Recognition System can be improved with the use of other Deep Learning algorithms not explored in this project. The Response Selection System can be designed with techniques other than topic modelling and Deep Learning options can be explored. The SQuAD can be also used for making a Machine Comprehension system which can be integrated into the Neural NLU Engine.

VII. CONCLUSION

Over the course of this paper, the history of Artificial Intelligence has been discussed and how it led to the creation of Conversational Platforms. Understanding

the shortcomings in today's technology is a strong motivation for this paper to overcome those shortcomings with a series of methodologies which combine to form the desired Neural Natural Language Understanding Engine.

Tasks such as generation of word vectors, topic modelling and named entity recognition seem to be mutually exclusive but they all play a vital role in designing an Efficient NLU engine which can power a strong conversational agent which can accurately understand the user message and respond to the user with a relevant answer.

We observed the various plots for LDA topic modelling which describe the model perplexity when the model is trained with changing hyperparameters which led to the implementation of the Author-Topic Modelling which was able to reduce almost *96%* of corpus titles for intent representation with an accuracy of *78%*.

For the Intent Recognition System, after training the neural network using various architectures, it is observed that the most preferred choice is the LSTM+CNN Model Architecture as the model achieves a higher accuracy in fewer training iterations. Upon training the model, it achieves an accuracy of *88.36%*.

REFERENCES

[1] A.M. Turing, "Computing Machinery and Intelligence", *Mind*, Vol. 49, Pg. 433-460, 1950.

[2] Joseph Weizenbaum, "ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine", *Communications of the ACM*, Vol. 9, Pg. 36-45, 1966.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, "Efficient Estimation of Word Representations in Vector Spaces", *Arxiv*, Pg. 1-9, 2013.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, "Distributed Representations of Words and Phrases and their Compositionality", *Advances in Neural Information Processing Systems*, Pg. 3111-3119, 2013.

[5] David M. Blei, Andrew Y. Ng, Michael I. Jordan, "Latent Dirichlet Allocation", *Journal of Machine Learning Research*, Vol. 3, Pg. 993-1022, 2003.

[6] Michael Rosen-Zvi, Thomas Griffiths, Mark Steyvers, Padhraic Smyth, "The Author-Topic Model for Authors and Documents", *UAI '04 Proceedings of the 20th conference on Uncertainty in artificial intelligence*, Canada, 7-11 July, Pg. 487-494, 2004.

[7] J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Sciences of the United States of America*, USA, April, Vol. 79, Pg. 2554-2558, 1982.

[8] Sepp Hochreiter, Jürgen Schmidhuber, "Long Short-Term Memory", *Neural Computation 9(8)*, Pg. 1735-1780, 1997.

[9] Kunihiko Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics*, Vol. 36, Pg. 193-202, 1980.

[10] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", 25th International Conference on Neural Information Processing Systems, USA, 3-6 December, Vol. 1, Pg. 1097-1105, 2012.

[11] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, Percy Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text", *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, USA, 1-5 November, Pg. 1-10, 2016.